SPRINGER
REFERENCE

Ling Liu
M. Tamer Özsu
Editors-in-Chief

Encyclopedia of Database Systems

Springer

Encyclopedia of Database Systems

Ling Liu, M. Tamer Özsu (Eds.)

Encyclopedia of Database Systems

With 3,067 Entries
With 871 Authors
With 1,176 Figures and 101 Tables
With 6,900 Cross-references
With 10,696 Bibliographic references

Springer

LING LIU
Professor
College of Computing
Georgia Institute of Technology
266 Ferst Drive
Atlanta, GA 30332-0765
USA

M. TAMER ÖZSU
Professor and Director, University Research Chair
Database Research Group
David R. Cheriton School of Computer Science
University of Waterloo
200 University Avenue West
Waterloo, ON
Canada N2L 3G1

Library of Congress Control Number: 2009931217

ISBN: 978-0-387-35544-3

This publication is available also as:
Electronic publication under ISBN: 978-0-387-39940-9 and
Print and electronic bundle under ISBN: 978-0-387-49616-0

springer.com

Printed on acid free paper SPIN: 11752127 2109SPi– 5 4 3 2 1 0

To our families

Preface

We are in an information era where generating and storing large amounts of data are commonplace. A growing number of organizations routinely handle terabytes and exabytes of data, and individual digital data collections easily reach multiple gigabytes. Along with the increases in volume, the modality of digitized data that requires efficient management and the access modes to these data have become more varied. It is increasingly common for business and personal data collections to include images, video, voice, and unstructured text; the retrieval of these data comprises various forms, including structured queries, keyword search, and visual access. Data have become a highly valued asset for governments, industries and individuals, and the management of these data collections remains a critical technical challenge.

Database technology has matured over the past four decades and is now quite ubiquitous in many applications that deal with more traditional business data. The challenges of expanding data management to include other data modalities while maintaining the fundamental tenets of database management (data independence, data integrity, data consistency, etc.) are issues that the community continues to work on. The lines between database management and other fields such as information retrieval, multimedia retrieval, and data visualization are increasingly blurred.

This multi-volume *Encyclopedia of Database Systems* provides easy access to important concepts on all aspects of database systems, including areas of current interest and research results of historical significance. It is a comprehensive collection of over 1,250 in-depth entries (3,067 including synonyms) that present coverage of the important concepts, issues, emerging technology and future trends in the field of database technologies, systems, and applications. The content of the *Encyclopedia* was determined through wide consultations. We were assisted by an Advisory Board in coming up with the overall structure and content. Each of these areas were put under the control of Area Editors (70 in total) who further developed the content for each area, soliciting experts in the field as contributors to write the entries, and performed the necessary technical editing. Some of them even wrote entries themselves. Nearly 1,000 authors were involved in writing entries.

The intended audience for the *Encyclopedia* is technically broad and diverse. It includes anyone concerned with database system technology and its applications. Specifically, the *Encyclopedia* can serve as a valuable and authoritative reference for students, researchers and practitioners who need a quick and authoritative reference to the subject of databases, data management, and database systems. We anticipate that many people will benefit from this reference work, including database specialists, software developers, scientists and engineers who need to deal with (structured, semi-structured or unstructured) large datasets. In addition, database and data mining researchers and scholars in the many areas that apply database technologies, such as artificial intelligence, software engineering, robotics and computer vision, machine learning, finance and marketing are expected to benefit from the *Encyclopedia*.

We would like to thank the members of the Advisory Board, the Editorial Board, and the individual contributors for their help in creating this *Encyclopedia*. The success of the *Encyclopedia* could not have been achieved without the expertise and the effort of the many contributors. Our sincere thanks also go to Springer's editors and staff, including Jennifer Carlson, Susan Lagerstrom-Fife, Oona Schmid, and Susan Bednarczyk for their support throughout the project.

Finally, we would very much like to hear from readers for any suggestions regarding the *Encyclopedia's* content. With a project of this size and scope, it is quite possible that we may have missed some concepts. It is also possible that some entries may benefit from revisions and clarifications. We are committed to issuing periodic updates and we look forward to the feedback from the community to improve the *Encyclopedia*.

Ling Liu
M. Tamer Özsu

Editors-in-Chief

Ling Liu is a Professor in the School of Computer Science, College of Computing, at Georgia Institute of Technology. Dr. Liu directs the research programs in Distributed Data Intensive Systems Lab (DiSL), examining various aspects of data intensive systems, ranging from database and Internet data management, data storage, network computing, and mobile and wireless computing, with the focus on performance, availability, security, privacy, and energy efficiency in building very large database and data management systems and services. She has published over 200 international journal and conference articles in the areas of databases, data engineering, and distributed computing systems. She is a recipient of the best paper award of ICDCS 2003, the best paper award of WWW 2004, the 2005 Pat Goldberg Memorial Best Paper Award, and the best data engineering paper award of Int. Conf. on Software Engineering and Data Engineering (2008). Dr. Liu served on the editorial board of *IEEE Transactions on Knowledge and Data Engineering* and *International Journal of Very Large Databases* from 2004 to 2008 and is currently serving on the editorial board of several international journals, including *Distributed and Parallel Databases Journal, IEEE Transactions on Service Computing* (TSC), *International Journal of Peer-to-Peer Networking and Applications* (Springer), and *Wireless Network* (Springer). Dr. Liu's current research is primarily sponsored by NSF, IBM, and Intel.

M. Tamer Özsu is a Professor of Computer Science and Director of the David R. Cheriton School of Computer Science at the University of Waterloo. He holds a Ph.D. (1983) and an MS (1981) in Computer and Information Science from The Ohio State University (1983) and a B.S. (1974) and M.S. (1978) in Industrial Engineering from the Middle East Technical University, Turkey (1974).

Dr. Özsu's current research focuses on three areas: (a) Internet-scale data distribution that emphasizes stream data management, peer-to-peer databases, and Web data management; (b) multimedia data management, concentrating on similarity-based retrieval of time series and trajectory data; and (c) the integration of database and information retrieval technologies, focusing on XML query processing and optimization. His previous research focused on distributed databases, interoperable information systems, object database systems and image databases. He is the co-author of the book *Principles of Distributed Database Systems* (Prentice Hall), which is now in its second edition (third edition to publish in 2009).

He currently holds a University Research Chair and has held a Faculty Research Fellowship at the University of Waterloo (2000-2003), and a McCalla Research Professorship (1993-1994) at the University of Alberta where he was faculty member between 1984 and 2000. He is a fellow of the Association for Computing Machinery (ACM), a senior member of Institute of Electrical and Electronics Engineers (IEEE), and a member of Sigma Xi. He was awarded the ACM SIGMOD Contributions Award in 2006. He is also the 2008 recipient of Ohio State University College of Engineering Distinguished Alumnus Award.

He has held visiting positions at GTE Laboratories (USA), INRIA Rocquencourt (France), GMD-IPSI (Germany), University of Jyväskylä (Finland), Technical University of Darmstadt (Germany), University of Udine (Italy), University of Milano (Italy), ETH Zürich (Switzerland), and National University of Singapore (Singapore).

Dr. Özsu serves on the editorial boards of *ACM Computing Surveys*, *Distributed and Parallel Databases Journal*, *World Wide Web Journal*, *Information Technology and Management*, and Springer Book Series on *Advanced Information & Knowledge Processing*. Previously he was the Coordinating Editor-in-Chief of *The VLDB Journal* (1997-2005) and was on the Editorial Board of *Encyclopedia of Database Technology and Applications* (Idea Group). He has served as the Program Chair of VLDB (2004), WISE (2001), IDEAS (2003), and CIKM (1996) conferences and the General Chair of CAiSE (2002), as well as serving on the Program Committees of many conferences including SIGMOD, VLDB, and ICDE. He is also a member of Association for Computing Machinery's (ACM) Publications Board and is its Vice-Chair for New Publications.

Dr. Özsu was the Chair of *ACM Special Interest Group on Management of Data* (SIGMOD; 2001-2005) and a past trustee of the *VLDB Endowment* (1996-2002). He was a member and chair of the Computer and Information Science Grant Selection Committee of the Natural Sciences and Engineering Research Council of Canada during 1991-94, and served on the Management Committee of the Canadian Genome Analysis and Technology Program during 1992-93. He was Acting Chair of the Department of Computing Science at the University of Alberta during 1994-95, and again, for a brief period, in 2000.

Advisory Board

Area Editors

Peer-to-Peer Data Management

Karl Aberer
EPFL-IC-IIF-LSIR
Lausanne
Switzerland

Database Management System Architectures

Anastasia Ailamaki
EPF Lausanne
Lausanne
Switzerland

Information Retrieval Models

Giambattista Amati
Fondazione Ugo Bordoni
Rome
Italy

XML Data Management

Sihem Amer-yahia
Yahoo! Research
New York, NY
USA

Database Middleware

Christiana Amza
University of Toronto
Toronto, ON
Canada

Database Tools
Database Tuning

PHILIPPE BONNET
University of Copenhagen
Copenhagen
Denmark

Visual Interfaces

TIZIANA CATARCI
University of Rome
Rome
Italy

Stream Data Management

UGUR CETINTEMEL
Brown University
Providence, RI
USA

Querying Over Data Integration Systems

KEVIN CHANG
University of Illinois at Urbana-Champaign
Urbana, IL
USA

Self Management

SURAJIT CHAUDHURI
Microsoft Research
Redmond, WA
USA

Text Mining

ZHENG CHEN
Microsoft Research Asia
Beijing
China

Extended Transaction Models (Advanced Concurrency Control Theory)

PANOS K. CHRYSANTHIS
University of Pittsburgh
Pittsburgh, PA
USA

Privacy-Preserving Data Mining

CHRIS CLIFTON
Purdue University
West Lafayette, IN
USA

Active Databases

KLAUS DITTRICH
University of Zürich
Zürich
Switzerland

Digital Libraries

AMR EL ABBADI
University of California-Santa Barbara
Santa Barbara, CA
USA

Data Models (Including Semantic Data Models)

DAVID EMBLEY
Brigham Young University
Provo, UT
USA

Complex Event Processing

OPHER ETZION
IBM Research Lab in Haifa
Haifa
Israel

Database Security and Privacy

ELENA FERRARI
University of Insubria
Varese
Italy

Semantic Web and Ontologies

AVIGDOR GAL
Technion - Israel Institute of Technology
Haifa
Israel

Data Cleaning

VENKATESH GANTI
Microsoft Research
Redmond, WA
USA

Web Data Extraction

GEORG GOTTLOB
Oxford University
Oxford
UK

Sensor Networks

LE GRUENWALD
The University of Oklahoma
Norman, OK
USA

Data Clustering

DIMITRIOS GUNOPULOS
University of Athens
Athens
Greece
University of California – Riverside
Riverside, CA
USA

Scientific Databases

AMARNATH GUPTA
University of California-San Diego
La Jolla, CA
USA

Geographic Information Systems

RALF HARTMUT GÜTING
University of Hagen
Hagen
Germany

Data Visualization

HANS HINTERBERGER
ETH Zürich
Zürich
Switzerland

Web Services and Service Oriented Architectures

HANS-ARNO JACOBSEN
University of Toronto
Toronto, ON
Canada

Temporal Databases

CHRISTIAN JENSEN
Aalborg University
Aalborg
Denmark

Metadata Management

MANFRED JEUSFELD
Tilburg University
Tilburg
The Netherlands

Health Informatics Databases

VIPUL KASHYAP
Partners Health Care System
Wellesley, MA
USA

Visual Data Mining

DANIEL KEIM
University of Konstanz
Konstanz
Germany

Data Replication

BETTINA KEMME
McGill University
Montreal, QC
Canada

Advanced Storage Systems
Storage Structures and Systems

MASARU KITSUREGAWA
The University of Tokyo
Tokyo
Japan

Views and View Management

YANNIS KOTIDIS
Athens University of Economics and Business
Athens
Greece

Semi-Structured Text Retrieval

MOUNIA LALMAS
University of Glasgow
Glasgow
UK

Information Quality

YANG LEE
Northeastern University
Boston, MA
USA

Relational Theory

LEONID LIBKIN
University of Edinburgh
Edinburgh
UK

Information Retrieval Evaluation Measures

WEIYI MENG
State University of New York at Binghamton
Binghamton, NY
USA

Data Integration

No Photo
available

RENÉE MILLER
University of Toronto
Toronto, ON
Canada

Database Design

JOHN MYLOPOULOS
University of Toronto
Toronto, ON
Canada

Text Indexing Techniques

MARIO NASCIMENTO
University of Alberta
Edmonton, AB
Canada

Data Quality

FELIX NAUMANN
Hasso Plattner Institute
Potsdam
Germany

Web Search and Crawl

CHRISTOPHER OLSTON
Yahoo! Research
Santa Clara, CA
USA

Multimedia Databases

VINCENT ORIA
New Jersey Institute of Technology
Newark, NJ
USA

Spatial, Spatiotemporal, and Multidimensional Databases

DIMITRIS PAPADIAS
Hong Kong University of Science and Technology
Hong Kong
China

Data Warehouse

TORBEN BACH PEDERSEN
Aalborg University
Aalborg
Denmark

Association Rule Mining

JIAN PEI
Simon Fraser University
Burnaby, BC
Canada

Workflow Management

BARBARA PERNICI
Politecnico di Milano
Milan
Italy

Query Processing and Optimization

EVAGGELIA PITOURA
University of Ioannina
Ioannina
Greece

Data Management for the Life Sciences

LOUIQA RASCHID
University of Marlyand
College Park, MD
USA

Information Retrieval Operations

EDIE RASMUSSEN
The University of British Columbia
Vancouver, BC
Canada

Query Languages

TORE RISCH
Uppsala University
Uppsala
Sweden

Data Warehouse

STEFANO RIZZI
University of Bologna
Bologna
Italy

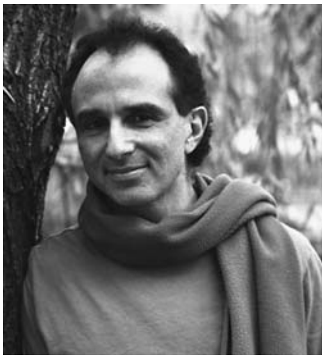
Multimedia Databases

SHIN'ICHI SATOH
National Institute of Informatics
Tokyo
Japan

Spatial, Spatiotemporal, and Multidimensional Databases

TIMOS SELLIS
National Technical University of Athens
Athens
Greece

Database Tools
Database Tuning



DENNIS SHASHA
New York University
New York, NY
USA

Classification and Decision Trees

KYUSEOK SHIM
Seoul National University
Seoul
Republic of Korea

Temporal Databases

RICK SNODGRASS
University of Arizona
Tuscon, AZ
USA

Stream Mining

DIVESH SRIVASTAVA
AT&T Labs – Research
Florham Park, NJ
USA

Distributed Database Systems

KIAN-LEE TAN
National University of Singapore
Singapore
Singapore

Logics and Databases

VAL TANNEN
University of Pennsylvania
Philadelphia, PA
USA

Structured and Semi-Structured Document Databases

FRANK WM. TOMPA
University of Waterloo
Waterloo, ON
Canada

Indexing

VASSILIS TSOTRAS
University of California – Riverside
Riverside, CA
USA

Parallel Database Systems

PATRICK VALDURIEZ
INRIA and LINA
Nantes
France

Advanced Storage Systems Storage structures and systems

KALADHAR VORUGANTI
Network Appliance
Sunnyvale, CA
USA

Transaction Management

GOTTFRIED VOSSEN
University of Münster
Münster
Germany

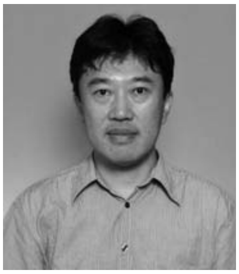
Self Management

GERHARD WEIKUM
Max Planck Institute for Informatics
Saarbrücken
Germany

Mobile and Ubiquitous Data Management

OURI WOLFSON
University of Illinois at Chicago
Chicago, IL
USA

Multimedia Information Retrieval



JEFFREY XU YU
Chinese University of Hong Kong
Hong Kong
China

Approximation and Data Reduction Techniques

XIAOFANG ZHOU
The University of Queensland
Brisbane, QLD
Australia

List of Contributors

W. M. P. van der Aalst
Eindhoven University of Technology
Eindhoven
The Netherlands

Daniel Abadi
Yale University
New Haven, CT
USA

Alberto Abelló
Polytechnic University of Catalonia
Barcelona
Spain

Serge Abiteboul
INRIA, Saclay
Orsay, Cedex
France

Ioannis Aekaterinidis
University of Patras
Rio Patras
Greece

Nitin Agarwal
Arizona State University
Tempe, AZ
USA

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY
USA

Lalitha Agnihotri
Philips Research
Eindhoven
The Netherlands

Yanif Ahmad
Brown University
Providence, RI
USA

Gail-Joon Ahn
Arizona State University
Tempe, AZ
USA

Anastasia Ailamaki
EPFL
Lausanne
Switzerland

Yousef J. Al-Houmaily
Institute of Public Administration
Riyadh
Saudi Arabia

Robert B. Allen
Drexel University
Philadelphia, PA
USA

Gustavo Alonso
ETH Zurich
Zurich
Switzerland

Omar Alonso
University of California at Davis
Davis, CA
USA

Bernd Amann
Pierre & Marie Curie University (UPMC)
Paris
France

Giambattista Amati
Fondazione Ugo Bordoni
Rome
Italy

Rainer von Ammon
Center for Information Technology Transfer GmbH
(CITT)
Regensburg
Germany

Robert A. Amsler
CSC
Falls Church, VA
USA

Cristiana Amza
University of Toronto
Toronto, ON
Canada

George Anadiotis
VU University Amsterdam
Amsterdam
The Netherlands

Mihael Ankerst
Allianz
Munich
Germany

Sameer Antani
National Institutes of Health
Bethesda, MD
USA

Grigoris Antoniou
Foundation for Research and Technology-Hellas
(FORTH)
Heraklion
Greece

Arvind Arasu
Microsoft Research
Redmond, WA
USA

Danilo Ardagna
Politecnico di Milano
Milan
Italy

Walid G. Aref
Purdue University
West Lafayette, IN
USA

Marcelo Arenas
Pontifical Catholic University of Chile
Santiago
Chile

Samuel Aronson
Harvard Medical School
Boston, MA
USA

Paavo Arvola
University of Tampere
Tampere
Finland

Noboru Babaguchi
Osaka University
Osaka
Japan

Shivnath Babu
Duke University
Durham, NC
USA

Kenneth Paul Baclawski
Northeastern University
Boston, MA
USA

Ricardo Baeza-Yates
Yahoo! Research
Barcelona
Spain

James Bailey
University of Melbourne
Melbourne, VIC
Australia

Peter Bak
University of Konstanz
Konstanz
Germany

Magdalena Balazinska
University of Washington
Seattle, WA
USA

Farnoush Banaei-Kashani
University of Southern California
Los Angeles, CA
USA

Stefano Baraldi
University of Florence
Florence
Italy

Mauro Barbieri
Philips Research
Eindhoven
The Netherlands

Denilson Barbosa
University of Alberta
Edmonton, AL
Canada

Pablo Barceló
University of Chile
Santiago
Chile

Luciano Baresi
Politecnico di Milano
Milan
Italy

Ilaria Bartolini
University of Bologna
Bologna
Italy

Sugato Basu
Google Inc.
Mountain View, CA
USA

Carlo Batini
University of Milan Bicocca
Milan
Italy

Michal Batko
Masaryk University
Brno
Czech Republic

Peter Baumann
Jacobs University
Bremen
Germany

Robert Baumgartner
Vienna University of Technology
Vienna, Austria
Lixto Software GmbH
Vienna
Austria

Sean Bechhofer
University of Manchester
Manchester
UK

Steven M. Beitzel
Telcordia Technologies
Piscataway, NJ
USA

Ladjel Bellatreche
LISI/ENSMA–Poitiers University
Futuroscope Cedex
France

Omar Benjelloun
Google Inc.
Mountain View, CA
USA

Véronique Benzaken
University Paris 11
Orsay Cedex
France

Mikael Berndtsson
University of Skövde
Skövde
Sweden

Philip A. Bernstein
Microsoft Corporation
Redmond, WA
USA

Damon Andrew Berry
University of Massachusetts
Lowell, MA
USA

Leopoldo Bertossi
Carleton University
Ottawa, ON
Canada

Claudio Bettini
University of Milan
Milan
Italy

Nigel Bevan
Professional Usability Services
London
UK

Bharat Bhargava
Purdue University
West Lafayette, IN
USA

Arnab Bhattacharya
Indian Institute of Technology
Kanpur
India

Ernst Biersack
Eurecom
Sophia Antipolis
France

Alberto Del Bimbo
University of Florence
Florence
Italy

Alan F. Blackwell
University of Cambridge
Cambridge
UK

Carlos Blanco
University of Castilla-La Mancha
Ciudad Real
Spain

Marina Blanton
University of Notre Dame
Notre Dame, IN
USA

Philip Bohannon
Yahoo! Research
Santa Clara, CA
USA

Michael H. Böhlen
Free University of Bozen-Bolzano
Bozen-Bolzano
Italy

Christian Böhm
University of Munich
Munich
Germany

Peter Boncz
CWI
Amsterdam
The Netherlands

Philippe Bonnet
University of Copenhagen
Copenhagen
Denmark

Alexander Borgida
Rutgers University
New Brunswick, NJ
USA

Chavdar Botev
Yahoo Research! and Cornell University
Ithaca, NY
USA

Sara Bouchenak
University of Grenoble I — INRIA
Grenoble
France

Luc Bouganim
INRIA Paris-Rocquencourt
Le Chesnay Cedex
France

Nozha Boujemaa
INRIA Paris-Rocquencourt
Le Chesnay Cedex
France

Shawn Bowers
University of California-Davis
Davis, CA
USA

Stéphane Bressan
National University of Singapore
Singapore
Singapore

Martin Breunig
University of Osnabrueck
Osnabrueck
Germany

Scott A. Bridwell
University of Utah
Salt Lake City, UT
USA

Thomas Brinkhoff
Institute for Applied Photogrammetry and
Geoinformatics (IAPG)
Oldenburg
Germany

Andrei Broder
Yahoo! Research
Santa Clara, CA
USA

Nicolas Bruno
Microsoft Corporation
Redmond, WA
USA

François Bry
University of Munich
Munich
Germany

Yingyi Bu
Chinese University of Hong Kong
Hong Kong
China

Alejandro Buchmann
Darmstadt University of Technology
Darmstadt
Germany

Chiranjeeb Buragohain
Amazon.com
Seattle, WA
USA

Thorsten Büring
Ludwig-Maximilians-University Munich
Munich
Germany

Benjamin Bustos
Department of Computer Science
University of Chile
Santiago
Chile

David Buttler
Lawrence Livermore National Laboratory
Livermore, CA
USA

Yanli Cai
Shanghai Jiao Tong University
Shanghai
China

Guadalupe Canahuate
The Ohio State University
Columbus, OH
USA

K. Selcuk Candan
Arizona State University
Tempe, AZ
USA

Turkmen Canli
University of Illinois at Chicago
Chicago, IL
USA

Alan Cannon
Napier University
Edinburgh
UK

Cornelia Caragea
Iowa State University
Ames, IA
USA

Barbara Carminati
University of Insubria
Varese
Italy

Michael W. Carroll
Villanova University School of Law
Villanova, PA
USA

Ben Carterette
University of Massachusetts Amherst
Amherst, MA
USA

Marco A. Casanova
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro
Brazil

Giuseppe Castagna
C.N.R.S. and University Paris 7
Paris
France

Tiziana Catarci
University of Rome
Rome
Italy

James Caverlee
Texas A&M University
College Station, TX
USA

Emmanuel Cecchet
EPFL
Lausanne
Switzerland

Wojciech Cellary
Poznan University of Economics
Poznan
Poland

Michal Ceresna
Lixto Software GmbH
Vienna
Austria

Uğur Çetintemel
Brown University
Providence, RI
USA

Soumen Chakrabarti
Indian Institute of Technology of Bombay
Mumbai
India

Don Chamberlin
IBM Almaden Research Center
San Jose, CA
USA

Allen Chan
IBM Toronto Software Lab
Markham, ON
Canada

Chee Yong Chan
National University of Singapore
Singapore
Singapore

K. Mani Chandy
California Institute of Technology
Pasadena, CA
USA

Edward Y. Chang
Google Research
Mountain View, CA
USA

Kevin C. Chang
University of Illinois at Urbana-Champaign
Urbana, IL
USA

Surajit Chaudhuri
Microsoft Research
Redmond, WA
USA

Elizabeth S. Chen
Partners HealthCareSystem
Boston, MA
USA

James L. Chen
University of Illinois at Chicago
Chicago, IL
USA

Jinjun Chen
Swinburne University of Technology
Melbourne, VIC
Australia

Lei Chen
Hong Kong University of Science and Technology
Hong Kong
China

Peter P. Chen
Louisiana State University
Baton Rouge, LA
USA

Hong Cheng
University of Illinois at Urbana-Champaign
Urbana, IL
USA
Chinese University of Hong Kong
Hong Kong
China

Reynold Cheng
The University of Hong Kong
Hong Kong
China

Vivying S. Y. Cheng
Hong Kong University of Science and Technology
(HKUST)
Hong Kong
China

InduShobha N. Chengalur-Smith
University at Albany – SUNY
Albany, NY
USA

Mitch Cherniack
Brandeis University
Wattham, MA
USA

Yun Chi
NEC Laboratories America
Cupertino, CA
USA

Rada Chirkova
North Carolina State University
Raleigh, NC
USA

Jan Chomicki
State University of New York at Buffalo
Buffalo, NY
USA

Stephanie Chow
University of Ontario Institute of Technology (UOIT)
Oshawa, ON
Canada

Vassilis Christophides
University of Crete
Heraklion
Greece

Panos K. Chrysanthis
University of Pittsburgh
Pittsburgh, PA
USA

Paolo Ciaccia
University of Bologna
Bologna
Italy

John Cieslewicz
Columbia University
New York, NY
USA

Gianluigi Ciocca
University of Milano-Bicocca
Milan
Italy

Eugene Clark
Harvard Medical School
Boston, MA
USA

Charles L. A. Clarke
University of Waterloo
Waterloo, ON
Canada

Eliseo Clementini
University of L'Aguila
L'Aguila
Italy

Chris Clifton
Purdue University
West Lafayette, IN
USA

Edith Cohen
AT&T Labs-Research
Florham Park, NJ
USA

Sara Cohen
The Hebrew University of Jerusalem
Jerusalem
Israel

Sarah Cohen-Boulakia
University of Pennsylvania
Philadelphia, PA
USA

Carlo Combi
University of Verona
Verona
Italy

Mariano P. Consens
University of Toronto
Toronto, ON
Canada

Dianne Cook
Iowa State University
Ames, IA
USA

Graham Cormode
AT&T Labs–Research
Florham Park, NJ
USA

Antonio Corral
University of Almeria
Almeria
Spain

Maria Francesca Costabile
University of Bari
Bari
Italy

Nick Craswell
Microsoft Research Cambridge
Cambridge
UK

Fabio Crestani
University of Lugano
Lugano
Switzerland

Marco Antonio Cristo
FUCAPI
Manaus
Brazil

Maxime Crochemore
King's College London
London
UK
University of Paris-East
Paris
France

Matthew G. Crowson
University of Illinois at Chicago
Chicago, IL
USA

Michel Crucianu
National Conservatory of Arts and Crafts
Paris
France

Philippe Cudré-Mauroux
Massachussetts Institute of Technology
Cambridge, MA
USA

Francisco Curbera
IBM T.J. Watson Research Center
Hawthorne, NY
USA

Peter Dadam
University of Ulm
Ulm
Germany

Mehmet M. Dalkiliç
Indiana University
Bloomington, IN
USA

Nilesh Dalvi
Yahoo! Research
Santa Clara, CA
USA

Manoranjan Dash
Nanyang Technological University
Singapore
Singapore

Anwitaman Datta
Nanyang Technological University
Singapore
Singapore

Ian Davidson
University of California-Davis
Davis, CA
USA

Antonios Deligiannakis
University of Athens
Athens
Greece

Alex Delis
University of Athens
Athens
Greece

Alan Demers
Cornell University
Ithaca, NY
USA

Ke Deng
University of Queensland
Brisbane, OLD
Australia

Amol Deshpande
University of Maryland
College Park, MD
USA

Zoran Despotovic
NTT DoCoMo Communications Laboratories Europe
Munich
Germany

Alin Deutsch
University of California-San Diego
La Jolla, CA
USA

Yanlei Diao
University of Massachusetts
Amherst, MA
USA

Suzanne W. Dietrich
Arizona State University
Phoenix, AZ
USA

Nevenka Dimitrova
Philips Research
Eindhoven
The Netherlands

Bolin Ding
University of Illinois at Urbana-Champaign
Champaign, IL
USA

Chris Ding
University of Texas at Arlington
Arlington, TX
USA

Alan Dix
Lancaster University
Lancaster
UK

Hong-Hai Do
SAP AG
Dresden
Germany

Gillian Dobbie
University of Auckland
Auckland
New Zealand

Alin Dobra
University of Florida
Gainesville, FL
USA

Vlastislav Dohnal
Masaryk University
Brno
Czech Republic

Mario Döller
University of Passau
Passau
Germany

Carlotta Domeniconi
George Mason University
Fairfax, VA
USA

Josep Domingo-Ferrer
Universitat Rovira i Virgili
Tarragona
Spain

Guozhu Dong
Wright State University
Dayton, OH
USA

Xin Luna Dong
AT&T Labs–Research
Florham Park, NJ
USA

Chitra Dorai
IBM T. J. Watson Research Center
Hawthorne, NY
USA

Zhicheng Dou
Nankai University
Tianjin
China

Yang Du
Northeastern University
Boston, MA
USA

Marlon Dumas
University of Tartu
Tartu
Estonia

Susan Dumais
Microsoft Research
Redmond, WA
USA

Schahram Dustdar
Technical University of Vienna
Vienna
Austria

Curtis Dyreson
Utah State University
Logan, UT
USA

Todd Eavis
Concordia University
Montreal, QC
Canada

Johann Eder
University of Vienna
Vienna
Austria

Ibrahim Abu El-Khair
Minia University
Minia
Egypt

Ahmed K. Elmagarmid
Purdue University
West Lafayette, IN
USA

Sameh Elnikety
Microsoft Research
Cambridge
UK

David W. Embley
Brigham Young University
Provo, UT
USA

Vincent Englebert
University of Namur
Namur
Belgium

AnnMarie Ericsson
University of Skövde
Skövde
Sweden

Martin Ester
Simon Fraser University
Burnaby, BC
Canada

Opher Etzion
IBM Research Labs-Haifa
Haifa
Israel

Patrick Eugster
Purdue University
West Lafayette, IN
USA

Ronald Fagin
IBM Almaden Research Center
San Jose, CA
USA

Hui Fang
University of Delaware
Newark, DE
USA

Wei Fan
IBM T.J. Watson Research
Hawthorne, NY
USA

Wenfei Fan
University of Edinburgh
Edinburgh
UK

Alan Fekete
University of Sydney
Sydney, NSW
Australia

Jean-Daniel Fekete
INRIA, LRI University Paris Sud
Orsay Cedex
France

Pascal Felber
University of Neuchatel
Neuchatel
Switzerland

Paolino Di Felice
University of L'Aguila
L'Aguila
Italy

Hakan Ferhatosmanoglu
The Ohio State University
Columbus, OH
USA

Eduardo B. Fernandez
Florida Atlantic University
Boca Raton, FL
USA

Eduardo Fernández-Medina
University of Castilla-La Mancha
Ciudad Real
Spain

Paolo Ferragina
University of Pisa
Pisa
Italy

Elena Ferrari
University of Insubria
Varese
Italy

Dennis Fetterly
Microsoft Research
Mountain View, CA
USA

Stephen E. Fienberg
Carnegie Mellon University
Pittsburgh, PA
USA

Peter M. Fischer
ETH Zurich
Zurich
Switzerland

Simone Fischer-Hübner
Karlstad University
Karlstad
Sweden

Leila De Floriani
University of Genova
Genova
Italy

Christian Fluhr
CEA LIST, Fontenay-aux
Roses
France

Greg Flurry
IBM SOA Advanced Technology
Armonk, NY
USA

Edward A. Fox
Virginia Tech
Blacksburg, VA
USA

Chiara Francalanci
Politecnico di Milano University
Milan
Italy

Andrew U. Frank
Vienna University of Technology
Vienna
Austria

Michael J. Franklin
University of California-Berkeley
Berkeley, CA
USA

Keir Fraser
University of Cambridge
Cambridge
UK

Juliana Freire
University of Utah
Salt Lake City, UT
USA

Elias Frentzos
University of Piraeus
Piraeus
Greece

Johann-Christoph Freytag
Humboldt University of Berlin
Berlin
Germany

Ophir Frieder
Georgetown University
Washington, DC
USA

Oliver Frölich
Lixto Software GmbH
Vienna
Austria

Tim Furche
University of Munich
Munich
Germany

Ariel Fuxman
Microsoft Research
Mountain View, CA
USA

Ada Wai-Chee Fu
Hong Kong University of Science and Technology
Hong Kong
China

Silvia Gabrielli
Bruno Kessler Foundation
Trento
Italy

Isabella Gagliardi
National Research Council (CNR)
Milan
Italy

Avigdor Gal
Technion – Israel Institute of Technology
Haifa
Israel

Wojciech Galuba
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne
Switzerland

Johann Gamper
Free University of Bozen-Bolzano
Bolzano
Italy

Vijay Gandhi
University of Minnesota
Minneapolis, MN
USA

Venkatesh Ganti
Microsoft Research
Redmond, WA
USA

Dengfeng Gao
IBM Silicon Valley Lab
San Jose, CA
USA

Like Gao
Teradata Corporation
San Diego, CA
USA

Wei Gao
The Chinese University of Hong Kong
Hong Kong
China

Minos Garofalakis
Technical University of Crete
Chania
Greece

Wolfgang Gatterbauer
University of Washington
Seattle, WA
USA

Bugra Gedik
IBM T.J. Watson Research Center
Hawthorne, NY
USA

Floris Geerts
University of Edinburgh
Edinburgh
UK

Johannes Gehrke
Cornell University
Ithaca, NY
USA

Betsy George
University of Minnesota
Minneapolis, MN
USA

Lawrence Gerstley
PSMI Consulting
San Francisco, CA
USA

Michael Gertz
University of California - Davis
Davis, CA
USA

Giorgio Ghelli
University of Pisa
Pisa
Italy

Gabriel Ghinita
National University of Singapore
Singapore
Singapore

Phillip B. Gibbons
Intel Research
Pittsburgh, PA
USA

Sarunas Girdzijauskas
EPFL
Lausanne
Switzerland

Fausto Giunchiglia
University of Trento
Trento
Italy

Kazuo Goda
The University of Tokyo
Tokyo
Japan

Max Goebel
Vienna University of Technology
Vienna
Austria

Bart Goethals
University of Antwerp
Antwerp
Belgium

Martin Gogolla
University of Bremen
Bremen
Germany

Aniruddha Gokhale
Vanderbilt University
Nashville, TN
USA

Lukasz Golab
AT&T Labs-Research
Florham Park, NJ
USA

Matteo Golfarelli
University of Bologna
Bologna
Italy

Michael F. Goodchild
University of California-Santa Barbara
Santa Barbara, CA
USA

Georg Gottlob
Oxford University
Oxford
UK

Valerie Gouet-Brunet
CNAM Paris
Paris
France

Ramesh Govindan
University of Southern California
Los Angeles, CA
USA

Goetz Graefe
Hewlett-Packard Laboratories
Palo Alto, CA
USA

Gösta Grahne
Concordia University
Montreal, QC
Canada

Fabio Grandi
University of Bologna
Bologna
Italy

Tyrone Grandison
IBM Almaden Research Center
San Jose, CA
USA

Peter M. D. Gray
University of Aberdeen
Aberdeen
UK

Todd J. Green
University of Pennsylvania
Philadelphia, PA
USA

Georges Grinstein
University of Massachusetts
Lowell, MA
USA

Tom Gruber
RealTravel
Emerald Hills, CA
USA

Le Gruenwald
The University of Oklahoma
Norman, OK
USA

Torsten Grust
University of Tübingen
Tübingen
Germany

Ralf Hartmut Güting
University of Hagen
Hagen
Germany

Dirk Van Gucht
Indiana University
Bloomington, IN
USA

Carlos Guestrin
Carnegie Mellon University
Pittsburgh, PA
USA

Dimitrios Gunopulos
University of California-Riverside
Riverside, CA
USA
University of Athens
Athens
Greece

Amarnath Gupta
University of California-San Diego
La Jolla, CA
USA

Himanshu Gupta
Stony Brook University
Stony Brook, NY
USA

Cathal Gurrin
Dublin City University
Dublin
Ireland

Marc Gyssens
University of Hasselt & Transnational University of Limburg
Diepenbeek
Belgium

Karl Hahn
BMW AG
Munich
Germany

Jean-Luc Hainaut
University of Namur
Namur
Belgium

Alon Halevy
Google Inc.
Mountain View, CA
USA

Maria Halkidi
University of Piraeus
Piraeus
Greece

Terry Halpin
Neumont University
South Jordan, UT
USA

Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, IL
USA

Alan Hanjalic
Delft University of Technology
Delft
The Netherlands

David Hansen
The Australian e-Health Research Centre
Brisbane, QLD
Australia

Jörgen Hansson
Carnegie Mellon University
Pittsburgh, PA
USA

Nikos Hardavellas
Carnegie Mellon University
Pittsburgh, PA
USA

Theo Härder
University of Kaiserslautern
Kaiserslautern
Germany

David Harel
The Weizmann Institute of Science
Rehovot
Israel

Jayant R. Haritsa
Indian Institute of Science
Bangalore
India

Stavros Harizopoulos
HP Labs
Palo Alto, CA
USA

Per F. V. Hasle
Aalborg University
Aalborg
Denmark

Jordan T. Hastings
University of California-Santa Barbara
Santa Barbara, CA
USA

Alexander Hauptmann
Carnegie Mellon University
Pittsburgh, PA
USA

Helwig Hauser
University of Bergen
Bergen
Norway

Ben He
University of Glasgow
Glasgow
UK

Pat Helland
Microsoft Corporation
Redmond, WA
USA

Joseph M. Hellerstein
University of California-Berkeley
Berkeley, CA
USA

Jean Henrard
University of Namur
Namur
Belgium

John Herring
Oracle Corporation
Nashua, NH
USA

Nicolas Hervé
INRIA Paris-Rocquencourt
Le Chesnay Cedex
France

Marcus Herzog
Vienna University of Technology
Vienna
Austria
Lixto Software GmbH
Vienna
Austria

Jean-Marc Hick
University of Namur
Namur
Belgium

Jan Hidders
University of Antwerp
Antwerpen
Belgium

Djoerd Hiemstra
University of Twente
Enschede
The Netherlands

Linda L. Hill
University of California-Santa Barbara
Santa Barbara, CA
USA

Alexander Hinneburg
Martin-Luther-University Halle-Wittenberg
Halle/Saale
Germany

Hans Hinterberger
ETH Zurich
Zurich
Switzerland

Erik Hoel
Environmental Systems Research Institute
Redlands, CA
USA

Vasant Honavar
Iowa State University
Ames, IA
USA

Mingsheng Hong
Cornell University
Ithaca, NY
USA

Haruo Hosoya
The University of Tokyo
Tokyo
Japan

Wynne Hsu
National University of Singapore
Singapore
Singapore

Jian Hu
Microsoft Research Asia
Haidian
China

Kien A. Hua
University of Central Florida
Orlando, FL
USA

Xian-Sheng Hua
Microsoft Research Asia
Beijing
China

Jun Huan
University of Kansas
Lawrence, KS
USA

Haoda Huang
Microsoft Research Asia
Beijing
China

Michael Huggett
University of British Columbia
Vancouver, BC
Canada

Patrick C. K. Hung
University of Ontario Institute of Technology (UOIT)
Oshawa, ON
Canada

Jeong-Hyon Hwang
Brown University
Providence, RI
USA

Ichiro Ide
Nagoya University
Nagoya
Japan

Alfred Inselberg
Tel Aviv University
Tel Aviv
Israel

Yannis Ioannidis
University of Athens
Athens
Greece

Panagiotis G. Ipeirotis
New York University
New York, NY
USA

Zachary Ives
University of Pennsylvania
Philadelphia, PA
USA

Hans-Arno Jacobsen
University of Toronto
Toronto, ON
Canada

H. V. Jagadish
University of Michigan
Ann Arbor, MI
USA

Alejandro Jaimes
Telefonica R&D
Madrid
Spain

Ramesh Jain
University of California-Irvine
Irvine, CA
USA

Sushil Jajodia
George Mason University
Fairfax, VA
USA

Greg Janée
University of California-Santa Barbara
Santa Barbara, CA
USA

Kalervo Järvelin
University of Tampere
Tampere
Finland

Christian S. Jensen
Aalborg University
Aalborg
Denmark

Eric C. Jensen
Twitter, Inc.
San Fransisco, CA
USA

Manfred A. Jeusfeld
Tilburg University
Tilburg
The Netherlands

Heng Ji
New York University
New York, NY
USA

Ricardo Jimenez-Peris
Universidad Politecnica de Madrid
Madrid
Spain

Jiashun Jin
Carnegie Mellon University
Pittsburgh, PA
USA

Ryan Johnson
Carnegie Mellon University
Pittsburg, PA
USA

Theodore Johnson
AT&T Labs Research
Florham Park, NJ
USA

Christopher B. Jones
Cardiff University
Cardiff
UK

Rosie Jones
Yahoo! Research
Burbank, CA
USA

James B. D. Joshi
University of Pittsburgh
Pittsburgh, PA
USA

Vanja Josifovski
Uppsala University
Uppsala
Sweden

Marko Junkkari
University of Tampere
Tampere
Finland

Jan Jurjens
The Open University
Buckinghamshire
UK

Mouna Kacimi
Max-Planck Institute for Informatics
Saarbrücken
Germany

Tamer Kahveci
University of Florida
Gainesville, FL
USA

Panos Kalnis
National University of Singapore
Singapore
Singapore

Jaap Kamps
University of Amsterdam
Amsterdam
The Netherlands

James Kang
University of Minnesota
Minneapolis, MN
USA

Carl-Christian Kanne
University of Mannheim
Mannheim
Germany

Aman Kansal
Microsoft Research
Redmond, WA
USA

Murat Kantarcioglu
University of Texas at Dallas
Dallas, TX
USA

George Karabatis
University of Maryland Baltimore County (UMBC)
Baltimore, MD
USA

Grigoris Karvounarakis
University of Pennsylvania
Philadelphia, PA
USA

George Karypis
University of Minnesota
Minneapolis, MN
USA

Vipul Kashyap
Partners Healthcare System
Wellesley, MA
USA

Yannis Katsis
University of California-San Diego
La Jolla, CA
USA

Raghav Kaushik
Microsoft Research
Redmond, WA
USA

Gabriella Kazai
Microsoft Research Cambridge
Cambridge
UK

Daniel A. Keim
University of Konstanz
Konstanz
Germany

Jaana Kekäläinen
University of Tampere
Tampere
Finland

Anastasios Kementsietsidis
IBM T.J. Watson Research Center
Hawthorne, NY
USA

Bettina Kemme
McGill University
Montreal, QC
Canada

Jessie Kennedy
Napier University
Edinburgh
UK

Vijay Khatri
Indiana University
Bloomington, IN
USA

Ashfaq Khokhar
University of Illinois at Chicago
Chicago, IL
USA

Daniel Kifer
Yahoo! Research
Santa Clara, CA
USA

Stephen Kimani
CSIRO Tasmanian ICT Centre
Hobart, TAS
Australia

Craig A. Knoblock
University of Southern California
Marina del Rey, CA
USA

Christoph Koch
Cornell University
Ithaca, NY
USA

Solmaz Kolahi
University of British Columbia
Vancouver, BC
Canada

George Kollios
Boston University
Boston, MA
USA

Poon Wei Koot
Nanyang Technological University
Singapore
Singapore

Flip R. Korn
AT&T Labs–Research
Florham Park, NJ
USA

Harald Kosch
University of Passau
Passau
Germany

Cartik R. Kothari
University of British Columbia
Vancouver, BC
Canada

Yannis Kotidis
Athens University of Economics and Business
Athens
Greece

Spyros Kotoulas
VU University Amsterdam
Amsterdam
The Netherlands

Manolis Koubarakis
University of Athens
Athens
Greece

Konstantinos Koutroumbas
Institute for Space Applications and Remote Sensing
Athens
Greece

Bernd J. Krämer
University of Hagen
Hagen
Germany

Peer Krögerand
Ludwig-Maximilians University of Munich
Munich
Germany

Werner Kriechbaum
IBM Development Lab
Böblingen
Germany

Hans-Peter Kriegel
Ludwig-Maximilians-University
Munich
Germany

Rajasekar Krishnamurthy
IBM Almaden Research Center
San Jose, CA
USA

Ravi Kumar
Yahoo Research
Santa Clara, CA
USA

Nicholas Kushmerick
Decho Corporation
Seattle, WA
USA

Mary Laarsgard
University of California-Santa Barbara
Santa Barbara, CA
USA

Alexandros Labrinidis
University of Pittsburgh
Pittsburgh, PA
USA

Zoé Lacroix
Arizona State University
Tempe, AZ
USA

Alberto H. F. Laender
Federal University of Minas Gerais
Belo Horizonte
Brazil

Bibudh Lahiri
Iowa State University
Ames, IA
USA

Laks V. S. Lakshmanan
University of British Columbia
Vancouver, BC
Canada

Mounia Lalmas
University of Glasgow
Glasgow
UK

Lea Landucci
University of Florence
Florence
Italy

Birger Larsen
Royal School of Library and Information Science
Copenhagen
Denmark

Per-Åke Larson
Microsoft Corporation
Redmond, WA
USA

Robert Laurini
LIRIS, INSA-Lyon
Lyon
France

Georg Lausen
University of Freiburg
Freiburg
Germany

Jens Lechtenbörger
University of Münster
Münster
Germany

Thierry Lecroq
University of Rouen
Rouen
France

Dongwon Lee
The Pennsylvania State University
University Park, PA
USA

Yang W. Lee
Northeastern University
Boston, MA
USA

Pieter De Leenheer
Vrije Universiteit Brussel, Collibra nv
Brussels
Belgium

Wolfgang Lehner
Dresden University of Technology
Dresden
Germany

Ronny Lempel
Yahoo! Research
Haifa
Israel

Kristina Lerman
University of Southern California
Marina del Rey, CA
USA

Ulf Leser
Humboldt University of Berlin
Berlin
Germany

Carson Kai-Sang Leung
University of Manitoba
Winnipeg, MB
Canada

Stefano Levialdi
Sapienza University of Rome
Rome
Italy

Brian Levine
University of Massachusetts
Amherst, MA
USA

Changqing Li
Duke University
Durham, NC
USA

Chen Li
University of California-Irvine
Irvine, CA
USA

Chengkai Li
University of Texas at Arlington
Arlington, TX
USA

Hua Li
Microsoft Research Asia
Beijing
China

Jinyan Li
Nanyang Technological University
Singapore
Singapore

Ninghui Li
Purdue University
West Lafayette, IN
USA

Ping Li
Cornell University
Ithaca, NY
USA

Qing Li
City University of Hong Kong
Hong Kong
China

Xue Li
The University of Queensland
Brisbane, QLD
Australia

Ying Li
IBM T.J. Watson Research Center
Hawthorne, NY
USA

Yunyao Li
IBM Almaden Research Center
San Jose, CA
USA

Leonid Libkin
University of Edinburgh
Edinburgh
UK

Sam S Lightstone
IBM, Canada Ltd.
Markham, ON
Canada

Jimmy Lin
University of Maryland
College Park, MD
USA

Tsau Young (T.Y.) Lin
San Jose State University
San Jose, CA
USA

Xuemin Lin
University of New South Wales
Sydney, NSW
Australia

Tok Wang Ling
National University of Singapore
Singapore
Singapore

Bing Liu
University of Illinois at Chicago
Chicago, IL
USA

Danzhou Liu
University of Central Florida
Orlando, FL
USA

Guimei Liu
National University of Singapore
Singapore
Singapore

Huan Liu
Arizona State University
Tempe, AZ
USA

Jinze Liu
University of Kentucky
Lexington, KY
USA

Ning Liu
Microsoft Research Asia
Beijing
China

Qing Liu
CSIRO Tasmanian ICT Centre
Hobart, TAS
Australia

Vebjorn Ljosa
Broad Institute of MIT and Harvard
Cambridge, MA
USA

David Lomet
Microsoft Research
Redmond, WA
USA

Phillip Lord
Newcastle University
Newcastle-Upon-Tyne
UK

Nikos A. Lorentzos
Agricultural University of Athens
Athens
Greece

Lie Lu
Microsoft Research Asia
Beijing
China

Bertram Ludäscher
University of California-Davis
Davis, CA
USA

Yan Luo
University of Illinois at Chicago
Chicago, IL
USA

Yves A. Lussier
University of Chicago
Chicago, IL
USA

Craig MacDonald
University of Glasgow
Glasgow
UK

Ashwin Machanavajjhala
Cornell University
Ithaca, NY
USA

Sam Madden
Massachussetts Institute of Technology
Cambridge, MA
USA

Paola Magillo
University of Genova
Genova
Italy

David Maier
Portland State University
Portland, OR
USA

Paul Maier
Technical University of Munich
Munich
Germany

Nikos Mamoulis
University of Hong Kong
Hong Kong
China

Stefan Manegold
CWI
Amsterdam
The Netherlands

Murali Mani
Worcester Polytechnic Institute
Worcester, MA
USA

Serge Mankovski
CA Labs, CA Inc.
Thornhill, ON
Canada

Ioana Manolescu
INRIA, Saclay–Île-de-France
Orsay
France

Yannis Manolopoulos
Aristotle University of Thessaloniki
Thessaloniki
Greece

Svetlana Mansmann
University of Konstanz
Konstanz
Germany

Florian Mansmann
University of Konstanz
Konstanz
Germany

Shahar Maoz
The Weizmann Institute of Science
Rehovot
Israel

Amélie Marian
Rutgers University
Piscataway, NJ
USA

Volker Markl
IBM Almaden Research Center
San Jose, CA
USA

Maria De Marsico
Sapienza University of Rome
Rome
Italy

David Martin
SRI International
Menlo Park, CA
USA

Maria Vanina Martinez
University of Maryland
College Park, MD
USA

Maristella Matera
Polytechnico di Milano
Milan
Italy

Marta Mattoso
Federal University of Rio de Janeiro
Rio de Janeiro
Brazil

Andrea Maurino
University of Milan Bicocca
Milan
Italy

Jan Małuszyński
Linköping University
Linköping
Sweden

Jose-Norberto Mazón
University of Alicante
Alicante
Spain

Kevin S. McCurley
Google Research
Mountain View, CA
USA

Andrew McGregor
Microsoft Research
Mountain View, CA
USA

Timothy McPhillips
University of California-Davis
Davis, CA
USA

Brahim Medjahed
The University of Michigan–Dearborn
Dearborn, MI
USA

Carlo Meghini
The Italian National Research Council
Pisa
Italy

Tao Mei
Microsoft Research Asia
Beijing
China

Jonas Mellin
University of Skövde
Skövde
Sweden

Massimo Melucci
University of Padua
Padua
Italy

Weiyi Meng
State University of New York at Binghamton
Binghamton, NY
USA

Ahmed Metwally
Google Inc.
Mountain View, CA
USA

Gerome Miklau
University of Massachusetts
Amherst, MA
USA

Harvey J. Miller
University of Utah
Salt Lake City, UT
USA

Renée J. Miller
University of Toronto
Toronto, ON
Canada

Tova Milo
Tel Aviv University
Tel Aviv
Israel

Prasenjit Mitra
The Pennsylvania State University
University Park, PA
USA

Michael Mitzenmacher
Harvard University
Boston, MA
USA

Mukesh Mohania
IBM India Research Lab
New Delhi
India

Mohamed F. Mokbel
University of Minnesota
Minneapolis, MN
USA

Angelo Montanari
University of Udine
Udine
Italy

Reagan W. Moore
University of California - San Diego
La Jolla, CA
USA

Konstantinos Morfonios
University of Athens
Athens
Greece

Peter Mork
The MITRE Corporation
McLean, VA
USA

Mirella M. Moro
Federal University of Rio Grande do Sol
Porto Alegre
Brazil

Edleno Silva de Moura
Federal University of Amazonas
Manaus
Brazil

Kyriakos Mouratidis
Singapore Management University
Singapore
Singapore

Kamesh Munagala
Duke University
Durham, NC
USA

Ethan V. Munson
University of Wisconsin-Milwaukee
Milwaukee, WI
USA

Shawn Murphy
Massachusetts General Hospital
Boston, MA
USA

John Mylopoulos
University of Toronto
Toronto, ON
Canada

Frank Nack
University of Amsterdam
Amsterdam
The Netherlands

Marc Najork
Microsoft Research
Mountain View, CA
USA

Ullas Nambiar
IBM India Research Lab
New Delhi
India

Alexandros Nanopoulos
Aristotle University
Thessaloniki
Greece

Vivek Narasayya
Microsoft Corporation
Redmond, WA
USA

Mario A. Nascimento
University of Alberta
Edmonton, AB
Canada

Alan Nash
Aleph One LLC
La Jolla, CA
USA

Harald Naumann
Vienna University of Technology
Vienna
Austria

Gonzalo Navarro
University of Chile
Santiago
Chile

Wolfgang Nejdl
University of Hannover
Hannover
Germany

Thomas Neumann
Max-Planck Institute for Informatics
Saarbrücken
Germany

Frank Neven
Hasselt University and Transnational University of Limburg
Diepenbeek
Belgium

Chong-Wah Ngo
City University of Hong Kong
Hong Kong
China

Peter Niblett
IBM United Kingdom Limited
Winchester
UK

Naoko Nitta
Osaka University
Osaka
Japan

Igor Nitto
University of Pisa
Pisa
Italy

Cheng Niu
Microsoft Research Asia
Beijing
China

Vilém Novák
University of Ostrava
Ostrava
Czech Republic

Chimezie Ogbuji
Cleveland Clinic Foundation
Cleveland, OH
USA

Peter Øhrstrøm
Aalborg University
Aalborg
Denmark

Christine M. O'Keefe
CSIRO Preventative Health National Research Flagship
Acton, ACT
Australia

Patrick O'Neil
University of Massachusetts
Boston, MA
USA

Iadh Ounis
University of Glasgow
Glasgow
UK

Mourad Ouzzani
Purdue University
West Lafayette, IN
USA

Fatma Özcan
IBM Almaden Research Center
San Jose, CA
USA

M. Tamer Özsu
University of Waterloo
Waterloo, ON
Canada

Esther Pacitti
University of Nantes
Nantes
France

Chris D. Paice
Lancaster University
Lancaster
UK

Noël De Palma
INPG – INRIA
Grenoble
France

Nathaniel Palmer
Workflow Management Coalition
Hingham, MA
USA

Biswanath Panda
Cornell University
Ithaca, NY
USA

Ippokratis Pandis
Carnegie Mellon University
Pittsburgh, PA
USA

Dimitris Papadias
Hong Kong University of Science and Technology
Hong Kong
China

Spiros Papadimitriou
IBM T.J. Watson Research Center
Hawthorne, NY
USA

Apostolos N. Papadopoulos
Aristotle University
Thessaloniki
Greece

Yannis Papakonstantinou
University of California-San Diego
La Jolla, CA
USA

Jan Paredaens
University of Antwerp
Antwerpen
Belgium

Christine Parent
University of Lausanne
Lausanne
Switzerland

Gabriella Pasi
University of Milano-Bicocca
Milan
Italy

Chintan Patel
Columbia University
New York, NY
USA

Jignesh M. Patel
University of Wisconsin-Madison
Madison, WI
USA

Marta Patiño-Martinez
Universidad Polytecnica de Madrid
Madrid
Spain

Norman W. Paton
University of Manchester
Manchester
UK

Cesare Pautasso
University of Lugano
Lugano
Switzerland

Torben Bach Pedersen
Aalborg University
Aalborg
Denmark

Fernando Pedone
University of Lugano
Lugano
Switzerland

Jovan Pehcevski
INRIA Paris-Rocquencourt
Le Chesnay Cedex
France

Jian Pei
Simon Fraser University
Burnaby, BC
Canada

Ronald Peikert
ETH Zurich
Zurich
Switzerland

Mor Peleg
University of Haifa
Haifa
Israel

Fuchun Peng
Yahoo! Inc.
Sunnyvale, CA
USA

Liam Peyton
University of Ottawa
Ottawa, ON
Canada

Mario Piattini
University of Castilla-La Mancha
Ciudad Real
Spain

Benjamin C. Pierce
University of Pennsylvania
Philadelphia, PA
USA

Karen Pinel-Sauvagnat
IRIT-SIG
Toulouse Cedex
France

Leo L. Pipino
University of Massachusetts
Lowell, MA
USA

Peter Pirolli
Palo Alto Research Center
Palo Alto, CA
USA

Evaggelia Pitoura
University of Ioannina
Ioannina
Greece

Benjamin Piwowarski
University of Glasgow
Glasgow
UK

Vassilis Plachouras
Yahoo! Research
Barcelona
Spain

Catherine Plaisant
University of Maryland
College Park, MD
USA

Claudia Plant
University of Munich
Munich
Germany

Christian Platzer
Technical University of Vienna
Vienna
Austria

Dimitris Plexousakis
Foundation for Research and Technology-Hellas (FORTH)
Heraklion
Greece

Neoklis Polyzotis
University of California Santa Cruz
Santa Cruz, CA
USA

Raymond K. Pon
University of California - Los Angeles
Los Angeles, CA
USA

Lucian Popa
IBM Almaden Research Center
San Jose, CA
USA

Alexandra Poulovassilis
University of London
London
UK

Sunil Prabhakar
Purdue University
West Lafayette, IN
USA

Cecilia M. Procopiuc
AT&T Labs
Florham Park, NJ
USA

Enrico Puppo
University of Genova
Genova
Italy

Ross S. Purves
University of Zurich
Zurich
Switzerland

Vivien Quéma
CNRS, INRIA
Saint-Ismier Cedex
France

Christoph Quix
RWTH Aachen University
Aachen
Germany

Sriram Raghavan
IBM Almaden Research Center
San Jose, CA
USA

Erhard Rahm
University of Leipzig
Leipzig
Germany

Krithi Ramamritham
IIT Bombay
Mumbai
India

Maya Ramanath
Max-Planck Institute for Informatics
Saarbrücken
Germany

Georgina Ramírez
Yahoo! Research Barcelona
Barcelona
Spain

Edie Rasmussen
University of British Columbia
Vancouver, BC
Canada

Indrakshi Ray
Colorado State University
Fort Collins, CO
USA

Diego Reforgiato Recupero
University of Maryland
College Park, MD
USA

Colin R. Reeves
Coventry University
Coventry
UK

Payam Refaeilzadeh
Arizona State University
Tempe, AZ
USA

Bernd Reiner
Technical University of Munich
Munich
Germany

Frederick Reiss
IBM Almaden Research Center
San Jose, CA
USA

Harald Reiterer
University of Konstanz
Konstanz
Germany

Matthias Renz
Ludwig Maximillian University of Munich
Munich
Germany

Andreas Reuter
EML Research gGmbH Villa Bosch
Heidelberg
Germany
Technical University Kaiserslautern
Kaiserslautern
Germany

Peter Revesz
University of Nebraska-Lincoln
Lincoln, NE
USA

Mirek Riedewald
Cornell University
Ithaca, NY
USA

Rami Rifaieh
University of California-San Diego
San Diego, CA
USA

Stefanie Rinderle
University of Ulm
Ulm
Germany

Tore Risch
Uppsala University
Uppsala
Sweden

Thomas Rist
University of Applied Sciences
Augsburg
Germany

Stefano Rizzi
University of Bologna
Bologna
Italy

Stephen Robertson
Microsoft Research Cambridge
Cambridge
UK

Roberto A. Rocha
Partners Healthcare System, Inc.
Boston, MA
USA

John F. Roddick
Flinders University
Adelaide, SA
Australia

Thomas Roelleke
Queen Mary University of London
London
UK

Didier Roland
University of Namur
Namur
Belgium

Oscar Romero
Polytechnic University of Catalonia
Barcelona
Spain

Rafael Romero
University of Alicante
Alicante
Spain

Timothy Roscoe
ETH Zurich
Zurich
Switzerland

Kenneth A. Ross
Columbia University
New York, NY
USA

Prasan Roy
Aster Data Systems, Inc.
Redwood City, CA
USA

Yong Rui
Microsoft China R&D Group
Redmond, WA
USA

Dan Russler
Oracle Health Sciences
Redwood Shores, CA
USA

Michael Rys
Microsoft Corporation
Sammamish, WA
USA

Giovanni Maria Sacco
University of Torino
Torino
Italy

Simonas Šaltenis
Aalborg University
Aalborg
Denmark

Kenneth Salem
University of Waterloo
Waterloo, ON
Canada

George Samaras
University of Cyprus
Nicosia
Cyprus

Giuseppe Santucci
University of Rome
Roma
Italy

Maria Luisa Sapino
University of Turin
Turin
Italy

Sunita Sarawagi
IIT Bombay
Mumbai
India

Anatol Sargin
University of Augsburg
Augsburg
Germany

Kai-Uwe Sattler
Technical University of Ilmenau
Ilmenau
Germany

Monica Scannapieco
University of Rome
Rome
Italy

Matthias Schäfer
University of Konstanz
Konstanz
Germany

Sebastian Schaffert
Salzburg Research
Salzburg
Austria

Ralf Schenkel
Max-Planck Institute for Informatics
Saarbrücken
Germany

Raimondo Schettini
University of Milano-Bicocca
Milan
Italy

Peter Scheuermann
Northwestern University
Evanston, IL
USA

Ulrich Schiel
Federal University of Campina Grande
Campina Grande
Brazil

Markus Schneider
University of Florida
Gainesville, FL
USA

Marc H. Scholl
University of Konstanz
Konstanz
Germany

Michel Scholl
Cedric-CNAM
Paris
France

Tobias Schreck
Darmstadt University of Technology
Darmstadt
Germany

Michael Schrefl
University of Linz
Linz
Austria

Matthias Schubert
Ludwig-Maximilians-University
Munich
Germany

Heiko Schuldt
University of Basel
Basel
Switzerland

Heidrun Schumann
University of Rostock
Rostock
Germany

Felix Schwagereit
University of Koblenz-Landau
Koblenz
Germany

Nicole Schweikardt
Johann Wolfgang Goethe-University
Frankfurt
Germany

Fabrizio Sebastiani
The Italian National Research Council
Pisa
Italy

Nicu Sebe
University of Amsterdam
Amsterdam
The Netherlands
University of Trento
Trento
Italy

Monica Sebillo
University of Salerno
Salerno
Italy

Thomas Seidl
RWTH Aachen University
Aachen
Germany

Manuel Serrano
University of Castilla – La Mancha
Ciudad Real
Spain

Amnon Shabo (Shvo)
IBM Research Lab-Haifa
Haifa
Israel

Mehul A. Shah
HP Labs
Palo Alto, CA
USA

Nigam Shah
Stanford University
Stanford, CA
USA

Cyrus Shahabi
University of Southern California
Los Angeles, CA
USA

Jayavel Shanmugasundaram
Yahoo Research!
Santa Clara, CA
USA

Marc Shapiro
INRIA Paris-Rocquencourt and LIP6
Paris
France

Mohamed A. Sharaf
University of Toronto
Toronto, ON
Canada

Mehdi Sharifzadeh
Google
Santa Monica, CA
USA

Jayant Sharma
Oracle Corporation
Nashua, NH
USA

Guy Sharon
IBM Research Labs-Haifa
Haifa
Israel

Dennis Shasha
New York University
New York, NY
USA

Carpendale Sheelagh
University of Calgary
Calgary, AB
Canada

Shashi Shekhar
University of Minnesota
Minneapolis, MN
USA

Dou Shen
Microsoft Corporation
Redmond, WA
USA

Heng Tao Shen
The University of Queensland
Brisbane, QLD
Australia

Jialie Shen
Singapore Management University
Singapore
Singapore

Rao Shen
Yahoo!
Sunnyvale, CA
USA

Xuehua Shen
Google, Inc.
Mountain View, CA
USA

Frank Y. Shih
New Jersey Institute of Technology
Newark, NJ
USA

Arie Shoshani
Lawrence Berkeley National Laboratory
Berkeley, CA
USA

Pavel Shvaiko
University of Trento
Trento
Italy

Wolf Siberski
University of Hannover
Hannover
Germany

Ronny Siebes
VU University Amsterdam
Amsterdam
The Netherlands

Adam Silberstein
Yahoo! Research Silicon Valley
Santa Clara, CA
USA

Sonia Fernandes Silva
Etruria Telematica Srl
Siena
Italy

Fabrizio Silvestri
ISTI-CNR
Pisa
Italy

Alkis Simitsis
IBM Almaden Research Center
San Jose, CA
USA

Simeon J. Simoff
University of Western Sydney
Sydney, NSW
Australia

Radu Sion
Stony Brook University
Stony Brook, NY
USA

Mike Sips
Stanford University
Stanford, CA
USA

Cristina Sirangelo
University of Edinburgh
Edinburgh
UK

Yannis Sismanis
IBM Almaden Research Center
Almaden, CA
USA

Spiros Skiadopoulos
University of Peloponnese
Tripoli
Greece

Richard T. Snodgrass
University of Arizona
Tucson, AZ
USA

Cees Snoek
University of Amsterdam
Amsterdam
The Netherlands

Il-Yeol Song
Drexel University
Philadelphia, PA
USA

Ruihua Song
Microsoft Research Asia
Beijing
China

Stefano Spaccapietra
EPFL
Lausanne
Switzerland

Greg Speegle
Baylor University
Waco, TX
USA

Padmini Srinivasan
The University of Iowa
Iowa City, IA
USA

Venkat Srinivasan
Virginia Tech
Blacksburg, VA
USA

Divesh Srivastava
AT&T Labs–Research
Florham Park, NJ
USA

Steffen Staab
University of Koblenz-Landau
Koblenz
Germany

Maarten van Steen
VU University
Amsterdam
The Netherlands

Constantine Stephanidis
Foundation for Research and Technology – Hellas
(FORTH)
Heraklion
Greece

Robert Stevens
University of Manchester
Manchester
UK

Andreas Stoffel
University of Konstanz
Konstanz
Germany

Michael Stonebraker
Massachusetts Institute of Technology
Cambridge, MA
USA

Umberto Straccia
The Italian National Research Council
Pisa
Italy

Martin J. Strauss
University of Michigan
Ann Arbor, MI
USA

Diane M. Strong
Worcester Polytechnic Institute
Worcester, MA
USA

Jianwen Su
University of California-Santa Barbara
Santa Barbara, CA
USA

Kazimierz Subieta
Polish-Japanese Institute of Information Technology
Warsaw
Poland

V. S. Subrahmanian
University of Maryland
College Park, MD
USA

Dan Suciu
University of Washington
Seattle, WA
USA

S. Sudarshan
Indian Institute of Technology
Bombay
India

Torsten Suel
Yahoo! Research
Sunnyvale, CA
USA

Jian-Tao Sun
Microsoft Research Asia
Beijing
China

Subhash Suri
University of California-Santa Barbara
Santa Barbara, CA
USA

Stefan Tai
University of Karlsruhe
Karlsruhe
Germany

Kian-Lee Tan
National University of Singapore
Singapore
Singapore

Pang-Ning Tan
Michigan State University
East Lansing, MI
USA

Wang-Chiew Tan
University of California-Santa Cruz
Santa Cruz
CA, USA

Letizia Tanca
Politecnico di Milano University
Milan
Italy

Lei Tang
Arizona State University
Tempe, AZ
USA

Wei Tang
Teradata Corporation
El Segundo, CA
USA

Egemen Tanin
University of Melbourne
Melbourne, VIC
Australia

Val Tannen
University of Pennsylvania
Philadelphia, PA
USA

Abdullah Uz Tansel
Baruch College – CUNY
New York, NY
USA

Yufei Tao
Chinese University of Hong Kong
Hong Kong
China

Sandeep Tata
IBM Almaden Research Center
San Jose, CA
USA

Nesime Tatbul
ETH Zurich
Zurich
Switzerland

Christophe Taton
INPG – INRIA
Grenoble
France

Paolo Terenziani
University of Turin
Turin
Italy

Evimaria Terzi
IBM Almaden Research Center
San Jose, CA
USA

Bernhard Thalheim
Christian-Albrechts University Kiel
Kiel
Germany

Martin Theobald
Stanford University
Stanford, CA
USA

Sergios Theodoridis
University of Athens
Athens
Greece

Yannis Theodoridis
University of Piraeus
Piraeus
Greece

Alexander Thomasian
Thomasian and Associates
Pleasantville, NY
USA

Bhavani Thuraisingham
The University of Texas at Dallas
Richardson, TX
USA

Srikanta Tirthapura
Iowa State University
Ames, IA
USA

Wee Hyong Tok
National University of Singapore
Singapore
Singapore

David Toman
University of Waterloo
Waterloo, ON
Canada

Frank Wm. Tompa
University of Waterloo
Waterloo, ON
Canada

Rodney Topor
Griffith University
Nathan, QLD
Australia

Riccardo Torlone
University of Rome
Rome
Italy

Kristian Torp
Aalborg University
Aalborg
Denmark

Nicola Torpei
University of Florence
Florence
Italy

Nerius Tradišauskas
Aalborg University
Aalborg
Denmark

Goce Trajcevski
Northwestern University
Evanston, IL
USA

Peter Triantafillou
University of Patras
Rio Patras
Greece

Silke Triβl
Humboldt University of Berlin
Berlin
Germany

Andrew Trotman
University of Otago
Dunedin
New Zealand

Juan Trujillo
University of Alicante
Alicante
Spain

Theodora Tsikrika
Center for Mathematics and Computer Science
Amsterdam
The Netherlands

Vassilis J. Tsotras
University of California-Riverside
Riverside, CA
USA

Peter A. Tucker
Whitworth University
Spokane, WA
USA

Anthony K. H. Tung
National University of Singapore
Singapore
Singapore

Theodoros Tzouramanis
University of the Aegean
Salmos
Greece

Antti Ukkonen
Helsinki University of Technology
Helsinki
Finland

Mollie Ullman-Cullere
Harvard Medical School
Boston, MA
USA

Antony Unwin
Augsburg University
Augsburg
Germany

Ali Ünlü
University of Augsburg
Augsburg
Germany

Susan D. Urban
Texas Tech University
Lubbock, TX
USA

Jaideep Vaidya
Rutgers University
Newark, NJ
USA

Shivakumar Vaithyanathan
IBM Almaden Research Center
San Jose, CA
USA

Athena Vakali
Aristotle University
Thessaloniki
Greece

Patrick Valduriez
INRIA and LINA
Nantes
France

Christelle Vangenot
EPFL
Lausanne
Switzerland

Stijn Vansummeren
Hasselt University and Transnational University of Limburg
Diepenbeek
Belgium

Vasilis Vassalos
Athens University of Economics and Business
Athens
Greece

Michael Vassilakopoulos
University of Central Greece
Lamia
Greece

Panos Vassiliadis
University of Ioannina
Ioannina
Greece

Michalis Vazirgiannis
Athens University of Economics & Business
Athens
Greece

Olga Vechtomova
University of Waterloo
Waterloo, ON
Canada

Erik Vee
Yahoo! Research
Silicon Valley, CA
USA

Jari Veijalainen
University of Jyvaskyla
Jyvaskyla
Finland

Yannis Velegrakis
University of Trento
Trento
Italy

Suresh Venkatasubramanian
University of Utah
Salt Lake City, UT
USA

Rossano Venturini
University of Pisa
Pisa
Italy

Victor Vianu
University of California-San Diego
La Jolla, CA
USA

K. Vidyasankar
Memorial University of Newfoundland
St. John’s, NL
Canada

Millist Vincent
University of South Australia
Adelaide, SA
Australia

Giuliana Vitiello
University of Salerno
Salerno
Italy

Michail Vlachos
IBM T.J. Watson Research Center
Hawthorne, NY
USA

Agnès Voisard
Fraunhofer Institute for Software and Systems
Engineering (ISST)
Berlin
Germany

Kaladhar Voruganti
Network Appliance
Sunnyvale, CA
USA

Gottfried Vossen
University of Münster
Münster
Germany

Kenichi Wada
Hitachi Limited
Tokyo
Japan

Feng Wang
City University of Hong Kong
Hong Kong
China

Jianyong Wang
Tsinghua University
Beijing
China

Jun Wang
Queen Mary University of London
London
UK

Meng Wang
Microsoft Research Asia
Beijing
China

X. Sean Wang
University of Vermont
Burlington, VT
USA

Xin-Jing Wang
Microsoft Research Asia
Beijing
China

Matthew O. Ward
Worcester Polytechnic Institute
Worcester, MA
USA

Segev Wasserkrug
IBM Research
Haifa
Israel

Hans Weda
Philips Research
Eindhoven
The Netherlands

Gerhard Weikum
Max-Planck Institute for Informatics
Saarbrücken
Germany

Michael Weiner
Indiana University School of Medicine
Indianapolis, IN
USA

Michael Weiss
Carleton University
Ottawa, ON
Canada

Ji-Rong Wen
Microsoft Research Asia
Beijing
China

Chunhua Weng
Columbia University
New York, NY
USA

Mathias Weske
University of Potsdam
Potsdam
Germany

Thijs Westerveld
Teezir Search Solutions
Ede
The Netherlands

Karl Wiggisser
University of Klagenfurt
Klagenfurt
Austria

Jef Wijsen
University of Mons-Hainaut
Mons
Belgium

Mark D. Wilkinson
University of British Columbia
Vancouver, BC
Canada

Graham Wills
SPSS Inc.
Chicago, IL
USA

Ian H. Witten
University of Waikato
Hamilton
New Zealand

Kent Wittenburg
Mitsubishi Electric Research Laboratories, Inc.
Cambridge, MA
USA

Eric Wohlstadter
University of British Columbia
Vancouver, BC
Canada

Dietmar Wolfram
University of Wisconsin-Milwaukee
Milwaukee, WI
USA

Ouri Wolfson
University of Illinois at Chicago
Chicago, IL
USA

Janette Wong
IBM Canada Ltd.
Markham, ON
Canada

Raymond Chi-Wing Wong
Hong Kong University of Science and Technology
Hong Kong
China

Peter T. Wood
Birkbeck, University of London
London
UK

David Woodruff
IBM Almaden Research Center
San Jose, CA
USA

Marcel Worring
University of Amsterdam
Amsterdam
The Netherlands

Adam Wright
Partners HealthCare
Boston, MA
USA

Yuqing Wu
Indiana University
Bloomington, IN
USA

Alex Wun
University of Toronto
Toronto, ON
Canada

Ming Xiong
Bell Labs
Murray Hill, NJ
USA

Guandong Xu
Victoria University
Melbourne, VIC
Australia

Hua Xu
Columbia University
New York, NY
USA

Jun Yan
Microsoft Research Asia
Haidian
China

Xifeng Yan
IBM T. J. Watson Research Center
Hawthorne, NY
USA

Jun Yang
Duke University
Durham, NC
USA

Li Yang
Western Michigan University
Kalamazoo, MI
USA

Ming-Hsuan Yang
University of California at Merced
Merced, CA
USA

Seungwon Yang
Virginia Tech
Blacksburg, VA
USA

Yu Yang
City University of Hong Kong
Hong Kong
China

Yun Yang
Swinburne University of Technology
Melbourne, VIC
Australia

Yong Yao
Cornell University
Ithaca, NY
USA

Mikalai Yatskevich
University of Trento
Trento
Italy

Hiroshi Yoshida
Fujitsu Limited
Yokohama
Japan

Masatoshi Yoshikawa
University of Kyoto
Kyoto
Japan

Matthew Young-Lai
Sybase iAnywhere
Waterloo, ON
Canada

Cong Yu
Yahoo! Research
New York, NY
USA

Hwanjo Yu
University of Iowa
Iowa City, IA
USA

Jeffrey Xu Yu
Chinese University of Hong Kong
Hong Kong
China

Philip S. Yu
IBM T.J. Watson Research Center
Yorktown Heights, NY
USA

Ting Yu
North Carolina State University
Raleigh, NC
USA

Vladimir Zadorozhny
University of Pittsburgh
Pittsburgh, PA
USA

Ilya Zaihrayeu
University of Trento
Trento
Italy

Mohammed J. Zaki
Rensselaer Polytechnic Institute
Troy, NY
USA

Carlo Zaniolo
University of California-Los Angeles
Los Angeles, CA
USA

Hugo Zaragoza
Yahoo! Research
Barcelona
Spain

Stan Zdonik
Brown University
Providence, RI
USA

Demetrios Zeinalipour-Yazti
University of Cyprus
Nicosia
Cyprus

Hans Zeller
Hewlett-Packard Laboratories
Palo Alto, CA
USA

Pavel Zezula
Masaryk University
Brno
Czech Republic

ChengXiang Zhai
University of Illinois at Urbana-Champaign
Urbana, IL
USA

Aidong Zhang
State University of New York at Buffalo
Buffalo, NY
USA

Benyu Zhang
Microsoft Research Asia
Beijing
China

Donghui Zhang
Northeastern University
Boston, MA
USA

Ethan Zhang
University of California-Santa Cruz and Yahoo! Inc.
Santa Cruz, CA
USA

Jin Zhang
University of Wisconsin-Milwaukee
Milwaukee, WI
USA

Kun Zhang
Xavier University of Louisiana
New Orleans, LA
USA

Lei Zhang
Microsoft Research Asia
Beijing
China

Li Zhang
Peking University
Beijing
China

Qing Zhang
The Australian e-health Research Center
Brisbane, QLD
Australia

Rui Zhang
University of Melbourne
Melbourne, VIC
Australia

Yanchun Zhang
Victoria University
Melbourne, VIC
Australia

Yi Zhang
University of California-Santa Cruz
Santa Cruz, CA
USA

Yue Zhang
University of Pittsburgh
Pittsburgh, PA
USA

Zhen Zhang
University of Illinois at Urbana-Champaign
Urbana, IL
USA

Feng Zhao
Microsoft Research
Redmond, WA
USA

Ying Zhao
Tsinghua University
Beijing
China

Baihua Zheng
Singapore Management University
Singapore
Singapore

Yi Zheng
University of Ontario Institute of
Technology (UOIT)
Oshawa, ON
Canada

Jingren Zhou
Microsoft Research
Redmond, WA
USA

Li Zhou
Partners HealthCare System Inc. and Harvard
Medical School
Boston, MA
USA

Zhi-Hua Zhou
Nanjing University
Nanjing
China

Huaiyu Zhu
IBM Almaden Research Center
San Jose, CA
USA

Xingquan Zhu
Florida Atlantic University
Boca Ration, FL
USA

Cai-Nicolas Ziegler
Siemens AG
Munich
Germany

Hartmut Ziegler
University of Konstanz
Konstanz
Germany

Arthur Zimek
Ludwig-Maximilians University of Munich
Munich
Germany

Esteban Zimányi
Free University of Brussels
Brussels
Belgium

A

Absolute Time

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

A temporal database contains time-referenced, or time-stamped, facts. A time reference in such a database is *absolute* if its value is independent of the context, including the current time, *now*.

Key Points

An example is "Mary's salary was raised on March 30, 2007." The fact here is that Mary's salary was raised. The absolute time reference is March 30, 2007, which is a time instant at the granularity of day.

Another example is "Mary's monthly salary was $ 15,000 from January 1, 2006 to November 30, 2007." In this example, the absolute time reference is the time period [January 1, 2006 – November 30, 2007].

Absolute time can be contrasted with *relative time*.

Cross-references

▶ Now in Temporal Databases
▶ Relative Time
▶ Time Instant
▶ Time Period
▶ Temporal Database
▶ Temporal Granularity

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice. O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS, vol. 1399. Springer, 1998, pp. 406–413.
2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS, vol. 1399. Springer, 1998, pp. 367–405.

Abstract Versus Concrete Temporal Query Languages

Jan Chomicki[1], David Toman[2]
[1]State University of New York at Buffalo, Buffalo, NY, USA
[2]University of Waterloo, Waterloo, ON, Canada

Synonyms

Historical query languages

Definition

Temporal query languages are a family of query languages designed to query (and access in general) time-dependent information stored in temporal databases. The languages are commonly defined as extensions of standard query languages for non-temporal databases with *temporal features*. The additional features reflect the way dependencies of data on time are captured by and represented in the underlying temporal data model.

Historical Background

Most databases store time-varying information. On the other hand, SQL is often the language of choice for developing applications that utilize the information in these databases. Plain SQL, however, does not seem to provide adequate support for temporal applications.

Example. To represent the *employment histories* of persons, a common relational design would use a schema

`Employment(FromDate,ToDate, EID, Company)`,

with the intended meaning that a person identified by `EID` worked for `Company` continuously from `FromDate` to `ToDate`. Note that while the above schema is a standard relational schema, the additional assumption that the values of the attributes `FromDate` and `ToDate` represent *continuous periods* of time is itself *not* a part of the relational model.

Formulating even simple queries over such a schema is non-trivial. For example, the query GAPS: "*List all persons with gaps in their employment history, together* *with the gaps*" leads to a rather complex formulation in, e.g., SQL over the above schema (this is left as a challenge to readers who consider themselves SQL experts; for a list of appealing, but incorrect solutions, including the reasons why, see [9]). The difficulty arises because a single tuple in the relation is conceptually a *compact representation of a set of tuples*, each tuple stating that an employment fact was true on a particular day.

The tension between the conceptual abstract temporal data model (in the example, the property that employment facts are associated with individual *time instants*) and the need for an efficient and compact representation of temporal data (in the example, the representation of continuous periods by their start and end instants) has been reflected in the development of numerous temporal data models and temporal query languages [3].

Foundations

Temporal query languages are commonly defined using *temporal extensions* of existing non-temporal query languages, such as relational calculus, relational algebra, or SQL. The temporal extensions can be categorized in two, mostly orthogonal, ways:

- *The choice of the actual temporal values manipulated by the language.* This choice is primarily determined by the underlying temporal data model. The model also determines the associated operations on these values. The meaning of temporal queries is then defined in terms of temporal values and operations on them, and their interactions with *data* (non-temporal) values in a temporal database.
- *The choice of syntactic constructs to manipulate temporal values in the language.* This distinction determines whether the temporal values in the language are accessed and manipulated *explicitly*, in a way similar to other values stored in the database, or whether the access is *implicit*, based primarily on *temporally extending* the meaning of constructs that already exist in the underlying non-temporal language (while still using the operations defined by the temporal data model).

Additional design considerations relate to *compatibility* with existing query languages, e.g., the notion of temporal upward compatibility.

However, as illustrated above, an additional hurdle stems from the fact that many (early) temporal query languages allowed the users to manipulate a *finite underlying representation* of temporal databases rather than the actual temporal values/objects in the associated temporal data model. A typical example of this situation would be an approach in which the temporal data model is based on time instants, while the query language introduces interval-valued attributes. Such a discrepancy often leads to a complex and unintuitive semantics of queries.

In order to clarify this issue, Chomicki has introduced the notions of *abstract* and *concrete* temporal databases and query languages [2]. Intuitively, *abstract temporal query languages* are defined at the conceptual level of the temporal data model, while their *concrete* counterparts operate directly on an actual *compact encoding* of temporal databases. The relationship between abstract and concrete temporal query languages is also implicitly present in the notion of snapshot equivalence [7]. Moreover, Bettini et al. [1] proposed to distinguish between *explicit* and *implicit* information in a temporal database. The explicit information is stored in the database and used to derive the implicit information through *semantic assumptions*. Semantic assumptions related to fact persistence play a role similar to mappings between concrete and abstract databases, while other assumptions are used to address time-granularity issues.

Abstract Temporal Query Languages

Most temporal query languages derived by temporally extending the relational calculus can be classified as abstract temporal query languages. Their semantics are defined in terms of abstract temporal databases which, in turn, are typically defined within the point-stamped temporal data model, in particular *without* any additional hidden assumptions about the meaning of tuples in instances of temporal relations.

Example. The *employment histories* in an abstract temporal data model would most likely be captured by a simpler schema "`Employment(Date, EID, Company)`", with the intended meaning that a person identified by `EID` was working for `Company` on a particular `Date`. While instances of such a schema can potentially be very large (especially when a fine granularity of time is used), formulating queries is now much more natural.

Choosing abstract temporal query languages over concrete ones resolves the first design issue: the temporal values used by the former languages are time instants equipped with an appropriate temporal ordering (which is typically a linear order over the instants), and possibly other predicates such as temporal distance. The second design issue – access to temporal values – may be resolved in two different ways, as exemplified by two different query languages. They are as follows:

- Temporal Relational Calculus (TRC): a two-sorted first-order logic with variables and quantifiers explicitly ranging over the time and data domains.
- First-order Temporal Logic (FOTL): a language with an implicit access to timestamps using temporal connectives.

Example. The GAPS query is formulated as follows:

TRC: $\exists t_1, t_3 . t_1 < t_2 < t_3 \wedge \exists c.\texttt{Employment}(t_1, x, c) \wedge$
$(\neg \exists c.\texttt{Employment}(t_2, x, c)) \wedge$
$\exists c.\texttt{Employment}(t_3, x, c);$

FOTL: $\blacklozenge \exists c.\texttt{Employment}(x,c) \wedge$
$(\neg \exists c.\texttt{Employment}(x, c)) \wedge$
$\lozenge \exists c.\texttt{Employment}(x, c)$

Here, the explicit access to temporal values (in TRC) using the variables t_1, t_2, and t_3 can be contrasted with the implicit access (in FOTL) using the temporal operators ◆ (read "sometime in the past") and ◇ (read "sometime in the future"). The conjunction in the FOTL query represents an implicit temporal join. The formulation in TRC leads immediately to an equivalent way of expressing the query in SQL/TP [9], an extension of SQL based on TRC.

Example. The above query can be formulated in SQL/TP as follows:

```
SELECT t.Date, e1.EID
FROM Employment e1, Time t, Employment e2
WHERE e1.EID = e2.EID
AND e1.Date < e2.Date
   AND NOT EXISTS ( SELECT *
      FROM Employment e3
      WHERE e1.EID = e3.EID
      AND t.Date = e3.Date
        AND e1.Date < e3.Date
        AND e3.Date < e2.Date )
```

The unary constant relation `Time` contains all time instants in the time domain (in our case, all `Dates`) and is only needed to fulfill syntactic SQL-style requirements on attribute ranges. However, despite the fact that the instance of this relation is not finite, the query can be efficiently evaluated [9].

Note also that in all of the above cases, the formulation is *exactly the same* as if the underlying temporal database used the *plain* relational model (allowing for attributes ranging over time instants).

The two languages, FOTL and TRC, are the counterparts of the snapshot and timestamp models (cf. the entry Point-stamped Data Models) and are the roots of many other temporal query languages, ranging from the more TRC-like temporal extensions of SQL to more FOTL-like temporal relational algebras (e.g., the conjunction in temporal logic directly corresponds to a temporal join in a temporal relational algebra, as both of them induce an *implicit equality* on the associated time attributes).

Temporal integrity constraints over point-stamped temporal databases can also be conveniently expressed in TRC or FOTL.

Multiple Temporal Dimensions and Complex Values.

While the abstract temporal query languages are typically defined in terms of the point-based temporal data model, they can similarly be defined with respect to complex temporal values, e.g., pairs (or tuples) of time instants or even sets of time instants. In these cases, particularly in the case of set-valued attributes, it is important to remember that the set values are treated as *indivisible objects*, and hence truth (i.e., query semantics) is associated with the entire objects, but not necessarily with their components/subparts.

Concrete Temporal Query Languages

Although abstract temporal query languages provide a convenient and clean way of specifying queries, they are not immediately amenable to implementation. The main problem is that, in practice, the facts in temporal databases persist over periods of time. Storing all true facts individually *for every time instant* during a period would be prohibitively expensive or, in the case of infinite time domains such as *dense time*, even impossible.

Concrete temporal query languages avoid these problems by operating directly on the compact encodings of temporal databases. The most commonly used encoding is the one that uses *intervals*. However, in this setting, a tuple that associates a fact with such an interval is a compact representation of the association between the same fact and *all the time instants that belong to this interval*. This observation leads to the design choices that are commonly present in such languages:

- Coalescing is used, explicitly or implicitly, to consolidate representations of (sets of) time instants associated *with the same fact*. In the case of interval-based encodings, this leads to coalescing adjoining or overlapping intervals into a single interval. Note that coalescing only changes the *concrete representation* of a temporal relation, not its meaning (i.e., the abstract temporal relation); hence it has no counterpart in abstract temporal query languages.
- Implicit *set operations* on time values are used in relational operations. For example, conjunction (join) typically uses set intersection to generate a compact representation of the time instants attached to the facts in the result of such an operation.

Example. For the running example, a concrete schema for the employment histories would typically be defined as "`Employment(VT, EID, Company)`" where `VT` is a valid time attribute ranging over periods (intervals). The GAPS query can be formulated in a calculus-style language corresponding to TSQL2 (see the entry on TSQL2) along the following lines:

$$\begin{aligned}&\exists I_1, I_2.[\exists c.\mathtt{Employment}(I_1, x, c)] \wedge\\ &\quad [\exists c.\mathtt{Employment}(I_2, x, c)] \wedge I_1 \;\mathtt{precedes}\; I_2\\ &\quad \wedge I = [\mathtt{end}(I_1)+1, \mathtt{begin}(I_2)-1].\end{aligned}$$

In particular, the variables I_1 and I_2 range over periods and the `precedes` relationship is one of Allen's interval relationships. The final conjunct,

$$I = [\mathtt{end}(I_1)+1, \mathtt{begin}(I_2)-1],$$

creates a new period corresponding to the time instants related to a person's *gap in employment*; this interval value is explicitly constructed from the end and start points of I_1 and I_2, respectively. For the query to be correct, however, the results of evaluating the bracketed subexpressions, e.g., "$[\exists c.\mathtt{Employment}(I_1, x, c)]$," have to be *coalesced*. Without the insertion of the explicit coalescing operators, the query is *incorrect*. To see that, consider a situation in which a person p_0 is first employed by a company c_1, then by c_2, and finally by c_3, without any gaps in employment. Then without coalescing of the bracketed subexpressions of the above query, p_0 will be returned as a part of the result of the query, which is incorrect. Note also that it is not enough for the underlying (concrete) database to be coalesced.

The need for an explicit use of coalescing often makes the formulation of queries in some concrete SQL-based temporal query languages cumbersome and error-prone.

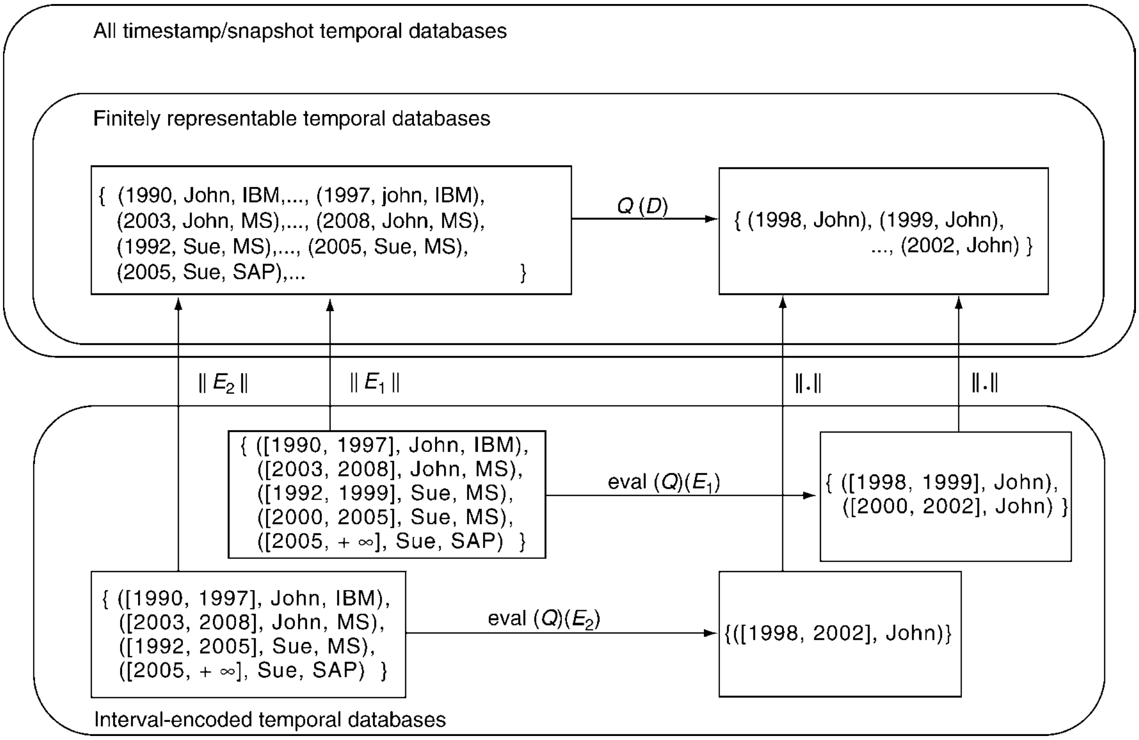
An orthogonal issue is the difference between explicit and implicit access to temporal values. This distinction also carries over to the concrete temporal languages. Typically, the various temporal extensions of SQL are based on the assumption of an explicit access to temporal values (often employing a built-in *valid time* attribute ranging over intervals or temporal elements), while many temporal relational algebras have chosen to use the implicit access based on temporally extending standard relational operators such as temporal join or temporal projection.

Compilation and Query Evaluation. An alternative to allowing users direct access to the encodings of temporal databases is to develop techniques that allow the evaluation of *abstract temporal queries* over these encodings. The main approaches are based on *query compilation* techniques that map abstract queries to concrete queries, while preserving query answers. More formally:

$$Q(\|E\|) = \|\mathtt{eval}(Q)(E)\|,$$

where Q an abstract query, eval(Q) the corresponding concrete query, E is a concrete temporal database, and ||.|| a mapping that associates encodings (concrete temporal databases) with their abstract counterparts (cf. Fig. 1). Note that a single abstract temporal database, D, can be encoded using several *different* instances of the corresponding concrete database, e.g., E_1 and E_2 in Fig. 1.

Most of the practical temporal data models adopt a common approach to physical representation of temporal databases: with every fact (usually represented as a tuple), a *concise encoding* of the set of time points at which the fact holds is associated. The encoding is commonly realized by *intervals* [6,7] or temporal elements (finite unions of intervals). For such an encoding it has been shown that both First-Order Temporal Logic [4] and Temporal Relational Calculus [8] queries can be *compiled* to first-order queries over a natural relational representation of the interval encoding of the database. Evaluating the resulting queries yields the interval encodings of the answers to the original queries, as if the queries were directly evaluated on the point-stamped temporal database. Similar results can be obtained for more complex encodings, e.g., periodic


All timestamp/snapshot temporal databases
Finitely representable temporal databases
{ (1990, John, IBM),..., (1997, john, IBM), (2003, John, MS),..., (2008, John, MS), (1992, Sue, MS),..., (2005, Sue, MS), (2005, Sue, SAP),... }
Q (D)
{ (1998, John), (1999, John), ..., (2002, John) }
	E2	
	E1	
	.	
	.	
{ ([1990, 1997], John, IBM), ([2003, 2008], John, MS), ([1992, 1999], Sue, MS), ([2000, 2005], Sue, MS), ([2005, + ∞], Sue, SAP) }
eval (Q)(E1)
{ ([1998, 1999], John), ([2000, 2002], John) }
{ ([1990, 1997], John, IBM), ([2003, 2008], John, MS), ([1992, 2005], Sue, MS), ([2005, + ∞], Sue, SAP) }
eval (Q)(E2)
{([1998, 2002], John)}
Interval-encoded temporal databases

Abstract Versus Concrete Temporal Query Languages. Figure 1. Query evaluation over interval encodings of point-stamped temporal databases.

sets, and for abstract temporal query languages that adopt the duplicate semantics matching the SQL standard, such as SQL/TP [9].

Key Applications

Temporal query languages are primarily used for querying temporal databases. However, because of their generality they can be applied in other contexts as well, e.g., as an underlying conceptual foundation for querying sequences and data streams [5].

Cross-references

- ▶ Allen's Relations
- ▶ Bitemporal Relation
- ▶ Constraint Databases
- ▶ Key
- ▶ Nested Transaction Models
- ▶ Non First Normal Form
- ▶ Point-Stamped Temporal Models
- ▶ Relational Model
- ▶ Snapshot Equivalence
- ▶ SQL
- ▶ Telic Distinction in Temporal Databases
- ▶ Temporal Coalescing
- ▶ Temporal Data Models
- ▶ Temporal Element
- ▶ Temporal Granularity
- ▶ Temporal Integrity Constraints
- ▶ Temporal Joins
- ▶ Temporal Logic in Database Query Languages
- ▶ Temporal Relational Calculus
- ▶ Time Domain
- ▶ Time Instant
- ▶ Transaction Time
- ▶ TSQL2
- ▶ Valid Time

Recommended Reading

1. Bettini C., Wang X.S., and Jajodia S. Temporal Semantic Assumptions and Their Use in Databases. Knowl. Data Eng., 10(2):277–296, 1998.
2. Chomicki J. Temporal query languages: a survey. In Proc. 1st Int. Conf. on Temporal Logic, 1994, pp. 506–534.
3. Chomicki J. and Toman D. Temporal databases. In Handbook of Temporal Reasoning in Artificial Intelligence, Fischer M., Gabbay D., and Villa L. (eds.). Elsevier Foundations of Artificial Intelligence, 2005, pp. 429–467.

4. Chomicki J., Toman D., and Böhlen M.H. Querying ATSQL databases with temporal logic. ACM Trans. Database Syst., 26(2):145–178, 2001.
5. Law Y.-N., Wang H., and Zaniolo C. Query languages and data models for database sequences and data streams. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 492–503.
6. Navathe S.B. and Ahmed R. In Temporal Extensions to the Relational Model and SQL. Tansel A., Clifford J., Gadia S., Jajodia S., Segev A., and Snodgrass R.T. (eds.). Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, Menlo Park, CA, 1993, pp. 92–109.
7. Snodgrass R.T. The temporal query language TQuel. ACM Trans. Database Syst., 12(2):247–298, 1987.
8. Toman D. Point vs. interval-based query languages for temporal databases. In Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1996, pp. 58–67.
9. Toman D. Point-based temporal extensions of SQL. In Proc. 5th Int. Conf. on Deductive and Object-Oriented Databases, 1997, pp. 103–121.

Abstraction

BERNHARD THALHEIM
Christian-Albrechts University Kiel, Kiel, Germany

Synonyms

Component abstraction; Implementation abstraction; Association; Aggregation; Composition; Grouping; Specialization; Generalisation; Classification

Definition

Abstraction allows developers to concentrate on the essential, relevant, or important parts of an application. It uses a mapping to a model from things in reality or from virtual things. The model has the truncation property, i.e., it lacks some of the details in the original, and a pragmatic property, i.e., the model use is only justified for particular model users, tools of investigation, and periods of time. Database engineering uses construction abstraction, context abstraction, and refinement abstraction. Construction abstraction is based on the principles of hierarchical structuring, constructor composition, and generalization. Context abstraction assumes that the surroundings of a concept are commonly understood by a community or within a culture and focuses on the concept, turning away attention from its surroundings such as the environment and setting. Refinement abstraction uses the principle of modularization and information hiding. Developers typically use conceptual models or languages for representing and conceptualizing abstractions. The enhanced entity-relationship model schema are typically depicted by an EER diagram.

Key Points

Database engineering distinguishes three kinds of abstraction: construction abstraction, context abstraction, and refinement abstraction.

Constructor composition depends on the constructors as originally introduced by J. M. Smith and D.C.W. Smith. Composition constructors must be well founded and their semantics must be derivable by inductive construction. There are three main methods for construction: development of ordered structures on the basis of hierarchies, construction by combination or association, and construction by classification into groups or collections. The set constructors ⊂ (subset), × (product), and $\mathcal{P}$ (powerset) for subset, product and nesting are complete for the construction of sets.

Subset constructors support hierarchies of object sets in which one set of objects is a subset of some other set of objects. Subset hierarchies are usually a rooted tree. Product constructors support associations between object sets. The schema is decomposed into object sets related to each other by association or relationship types. Power set constructors support a classification of object sets into clusters or groups of sets – typically according to their properties.

Context abstraction allows developers to commonly concentrate on those parts of an application that are essential for some perspectives during development and deployment of systems. Typical types of context abstraction are component abstraction, separation of concern, interaction abstraction, summarization, scoping, and focusing on typical application cases.

Component abstraction factors out repeating, shared or local patterns of components or functions from individual concepts. It allows developers to concentrate on structural or behavioral aspects of similar elements of components. Separation of concern allows developers to concentrate on those concepts under development and to neglect all other concepts that are stable or not under consideration. Interaction abstraction allows developers to concentrate on parts of the model that are essential for interaction with other systems or users. Summarisation maps the conceptualizations within the scope to more abstract concepts. Scoping is typically used to select those concepts that are necessary for current development and removes those concepts which that do not have an impact on the necessary concepts.

Database models may cover a large variety of different application cases. Some of them reflect exceptional, abnormal, infrequent and untypical application situations. Focusing on typical application cases explicitly separates models intended for the normal or typical application case from those that are atypical. Atypical application cases are not neglected but can be folded into the model whenever atypical situations are considered.

The context abstraction concept is the main concept behind federated databases. Context of databases can be characterized by schemata, version, time, and security requirements. Sub-schemata, types of the schemata or views on the schemata, are associated with explicit import/export bindings based on a name space. Parametrization lets developers consider collections of objects. Objects are identifiable under certain assumptions and completely identifiable after instantiation of all parameters.

Interaction abstraction allows developers to display the same set of objects in different forms. The view concept supports this visibility concept. Data is abstracted and displayed in various levels of granularity. Summarization abstraction allows developers to abstract from details that are irrelevant at a certain point. Scope abstraction allows developers to concentrate on a number of aspects. Names or aliases can be multiply used with varying structure, functionality and semantics.

Refinement abstraction mainly concerns implementation and modularisation. It allows developers to selectively retain information about structures. Refinement abstraction is defined on the basis of the development cycle (refinement of implementations). It refines, summarizes and views conceptualizations, hides or encapsulates details, or manages collections of versions. Each refinement step transforms a schema to a schema of finer granularity. Refinement abstraction may be modeled by refinement theory and infomorphisms.

Encapsulation removes internal aspects and concentrates on interface components. Blackbox or graybox approaches hide all aspects of the objects being considered. Partial visibility may be supported by modularization concepts. Hiding supports differentiation of concepts into public, private (with the possibility to be visible as "friends") and protected (with visibility to subconcepts). It is possible to define a number of visibility conceptualizations based on inflection. Inflection is used for the injection of combinable views into the given view, for tailoring, ordering and restructuring of views, and for enhancement of views by database functionality. Behavioral transparency is supported by the glassbox approach. Security views are based on hiding. Versioning allows developers to manage a number of concepts which can be considered to be versions of each other.

Cross-references

- ▶ Entity Relationship Model
- ▶ Extended Entity-Relationship Model
- ▶ Language Models
- ▶ Object Data Models
- ▶ Object-Role Modeling
- ▶ Specialization and Generalization

Recommended Reading

1. Börger E. The ASM refinement method. Formal Aspect. Comput., 15:237–257, 2003.
2. Smith J.M. and Smith D.C.W. Data base abstractions: aggregation and generalization. ACM Trans. Database Syst., 2 (2):105–133, 1977.
3. Thalheim B. Entity-Relationship Modeling – Foundations of Database Technology. Springer, 2000.

Access Control

Elena Ferrari
University of Insubria, Varese, Italy

Synonyms

Authorization verification

Definition

Access control deals with preventing unauthorized operations on the managed data. Access control is usually performed against a set of *authorizations* stated by Security Administrators (SAs) or users according to the *access control policies* of the organization. Authorizations are then processed by the *access control mechanism* (or *reference monitor*) to decide whether each access request can be authorized or should be denied.

Historical Background

Access control models for DBMSs have been greatly influenced by the models developed for the protection of operating system resources. For instance, the model proposed by Lampson [16] is also known as the *access matrix* model since authorizations are represented as a matrix. However, much of the early work on database protection was on inference control in statistical databases.

Then, in the 1970s, as research in relational databases began, attention was directed towards access control issues. As part of the research on System R at IBM Almaden Research Center, there was much work on access control for relational database systems [11,15], which strongly influenced access control models and mechanisms of current commercial relational DBMSs. Around the same time, some early work on multilevel secure database management systems (MLS/DBMSs) was reported. However, it was only after the Air Force Summer Study in 1982 [1] that developments on MLS/DBMSs began. For instance, the early prototypes based on the integrity lock mechanisms developed at the MITRE Corporation. Later, in the mid-1980s, pioneering research was carried out at SRI International and Honeywell Inc. on systems such as SeaView and LOCK Data Views [9]. Some of the technologies developed by these research efforts were transferred to commercial products by corporations such as Oracle, Sybase, and Informix. In the 1990s, numerous other developments were made to meet the access control requirements of new applications and environments, such as the World Wide Web, data warehouses, data mining systems, multimedia systems, sensor systems, workflow management systems, and collaborative systems. This resulted in several extensions to the basic access control models previously developed, by including the support for temporal constraints, derivation rules, positive and negative authorizations, strong and weak authorizations, and content and context-dependent authorizations [14]. Role-based access control has been proposed [12] to simplify authorization management within companies and organizations. Recently, there have been numerous developments in access control, mainly driven by developments in web data management. For example, standards such as XML (eXtensible Markup Language) and RDF (Resource Description Framework) require proper access control mechanisms [7]. Also, web services and the semantic web are becoming extremely popular and therefore research is currently carried out to address the related access control issues [13]. Access control is currently being examined for new application areas, such as knowledge management [4], data outsourcing, GIS [10], peer-to-peer computing and stream data management [8]. For example, in the case of knowledge management applications, it is important to protect the intellectual property of an organization, whereas when data are outsourced, it is necessary to allow the owner to enforce its access control policies, even if data are managed by a third party.

Foundations

The basic building block on which access control relies is a set of *authorizations*: which state, who can access which resource, and under which mode. Authorizations are specified according to a set of *access control policies*, which define the high-level rules according to which access control must occur. In its basic form, an authorization is, in general, specified on the basis of three components (*s,o,p*), and specifies that subject *s* is authorized to exercise privilege *p* on object *o*. The three main components of an authorization have the following meaning:

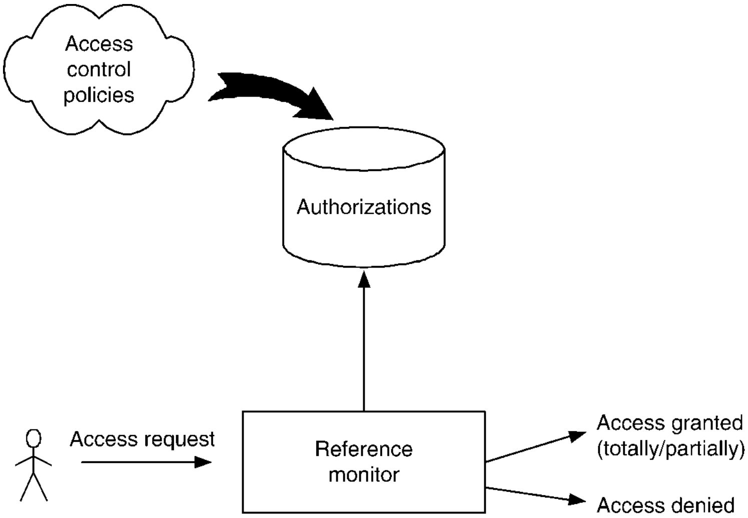
- *Authorization subjects*: They are the "active" entities in the system to which authorizations are granted. Subjects can be further classified into the following, not mutually exclusive, categories: *users*, that is, single individuals connecting to the system; *groups*, that is, sets of users; roles, that is, named collection of privileges needed to perform specific activities within the system; and *processes*, executing programs on behalf of users.
- *Authorization objects*: They are the "passive" components (i.e., resources) of the system to which protection from unauthorized accesses should be given. The set of objects to be protected clearly depends on the considered environment. For instance, files and directories are examples of objects of an operating system environment, whereas in a relational DBMS, examples of resources to be protected are relations, views and attributes. Authorizations can be specified at different granularity levels, that is, on a whole object or only on some of its components. This is a useful feature when an object (e.g., a relation) contains information (e.g., tuples) of different sensitivity levels and therefore requires a differentiated protection.
- *Authorization privileges*: They state the types of operations (or access modes) that a subject can exercise on the objects in the system. As for objects, the set of privileges also depends on the resources

to be protected. For instance, read, write, and execute privileges are typical of an operating system environment, whereas in a relational DBMS privileges refer to SQL commands (e.g., select, insert, update, delete). Moreover, new environments such as digital libraries are characterized by new access modes, for instance, usage or copying access rights.

Depending on the considered domain and the way in which access control is enforced, objects, subjects and/or privileges can be hierarchically organized. The hierarchy can be exploited to propagate authorizations and therefore to simplify authorization management by limiting the set of authorizations that must be explicitly specified. For instance, when objects are hierarchically organized, the hierarchy usually represents a "part-of" relation, that is, the hierarchy reflects the way objects are organized in terms of other objects. In contrast, the privilege hierarchy usually represents a subsumption relation among privileges. Privileges towards the bottom of the hierarchy are subsumed by privileges towards the top (for instance, the write privilege is at a higher level in the hierarchy with respect to the read privilege, since write subsumes read operations). Also roles and groups can be hierarchically organized. The group hierarchy usually reflects the membership of a group to another group. In contrast, the role hierarchy usually reflects the relative position of roles within an organization. The higher the level of a role in the hierarchy, the higher its position in the organization.

Authorizations are stored into the system and are then used to verify whether an access request can be authorized or not. How to represent and store authorizations depends on the protected resources. For instance, in a relational DBMS, authorizations are modeled as tuples stored into system catalogs. In contrast, when resources to be protected are XML documents, authorizations are usually encoded using XML itself. Finally, the last key component of the access control infrastructure is the *access control mechanism* (or *reference monitor*), which is a trusted software module in charge of enforcing access control. It intercepts each access request submitted to the system (for instance, SQL statements in case of relational DBMSs) and, on the basis of the specified authorizations, it determines whether the access can be partially or totally authorized or should be denied. The reference monitor should be *non-bypassable*. Additionally, the hardware and software architecture should ensure that the reference monitor is *tamper proof*, that is, it cannot be maliciously modified (or at least that any improper modification can be detected). The main components of access control are illustrated in Fig. 1.

A basic distinction when dealing with access control is between *discretionary* and *mandatory* access control. Discretionary access control (DAC) governs the access of subjects to objects on the basis of subjects' identity and a set of explicitly specified authorizations that specify, for each subject, the set of objects that



Access Control. Figure 1. Access control: main components.

he/she can access in the system and the allowed access modes. When an access request is submitted to the system, the access control mechanism verifies whether or not the access can be authorized according to the specified authorizations. The system is discretionary in the sense that a subject, by proper configuring the set of authorizations, is both able to enforce various access control requirements and to dynamically change them when needed (simply by updating the authorization state). In contrast, mandatory access control (MAC) specifies the accesses that subjects can exercise on the objects in the system, on the basis of subjects and objects security classification [14]. Security classes usually form a partially ordered set. This type of security has also been referred to as *multilevel security*, and database systems that enforce multilevel access control are called *Multilevel Secure Database Management Systems* (MLS/DBMSs). When mandatory access control is enforced, authorizations are implicitly specified, by assigning subjects and objects proper security classes. The decision on whether or not to grant an access depends on the access mode and the relation existing between the classification of the subject requesting the access and that of the requested object. In addition to DAC and MAC, role-based access control (RBAC) has been more recently proposed [12]. RBAC is an alternative to discretionary and mandatory access control, mainly conceived for regulating accesses within companies and organizations. In RBAC, permissions are associated with roles, instead of with users, and users acquire permissions through their membership to roles. The set of authorizations can be inferred by the sets of user-role and role-permission assignments.

Key Applications

Access control techniques are applied in almost all environments that need to grant a controlled access to their resources, including, but not limited, to the following: DBMSs, Data Stream Management Systems, Operat ing Systems, Workflow Management Systems, Digital Libraries, GIS, Multimedia DBMSs, E-commerce services, Publish-subscribe systems, Data warehouses.

Future Directions

Altough access control is a mature area with consolidated results, the evolution of DBMSs and the requirements of new applications and environments pose new challenges to the research community. An interesting discussion on open research issues in the field can be found in [6]. Some research issues which complement those presented in [6] are discussed below.

Social networks. Web-based social networks (WBSNs) are online communities where participants can establish relationships and share resources across the web with other users. In recent years, several WBSNs have been adopting semantic web technologies, such as FOAF, for representing users' data and relationships, making it possible to enforce information interchange across multiple WBSNs. Despite its advantages in terms of information diffusion, this raised the need for giving content owners more control on the distribution of their resources, which may be accessed by a community far wider than they expected. So far, this issue has been mainly addressed in a very simple way, by some of the available WBSNs, by only allowing users to state whether a specific information (e.g., personal data and resources) should be public or accessible only by the users with whom the owner of such information has a direct relationship. Such simple access control strategies have the advantage of being straightforward, but they are not flexible enough in denoting authorized users. In fact, they do not take into account the type of the relationships existing between users and, consequently, it is not possible to state that only, say, my "friends" can access a given information. Moreover, they do not allow to grant access to users who have an indirect relationship with the resource owner (e.g., the "friends of my friends"). Therefore, more flexible mechanisms are needed, making a user able to decide which network participants are authorized to access his/her resources and personal information. Additionally, since the number of social network users is considerably higher than those in conventional DBMSs, the traditional server-side way of enforcing access control, that is, the one relying on a centralized trusted reference monitor, should be revised and more efficient and distributed strategies should be devised for WBSNs. Until now, apart from [3], most of the security research on WBSNs has focused on privacy-preserving mining of social network data. The definition of a comprehensive framework for efficiently enforcing access control in social networks is therefore still an issue to be investigated.

- *Data streams*. In many applications, such as telecommunication, battle field monitoring, network monitoring, financial monitoring, sensor networks, data arrive in the form of high speed data streams. These data typically contain sensitive information (e.g., health information, credit card numbers) and thus unauthorized accesses should be avoided. Although many data stream processing systems have been developed so far (e.g., Aurora, Borealis, STREAM, TelegraphCQ, and StreamBase), the focus of these systems has been mainly on performance issues rather than on access control. On the other hand, though the data security community has a very rich history in developing access control models [9], these models are largely tailored to traditional DBMSs and therefore they cannot be readily applied to data stream management systems [8]. This is mainly because: (i) traditional data are static and bounded, while data streams are unbounded and infinite; (ii) queries in traditional DBMSs are one time and ad-hoc, whereas queries over data streams are typically continuous and long running; (iii) in traditional DBMSs, access control is enforced when users access the data; (iv) in data stream applications access control enforcement is data-driven (i.e., whenever data arrive), as such access control is more computational intensive in data stream applications and specific techniques to handle it efficiently should be devised; (v) temporal constraints (e.g., sliding windows) are more critical in data stream applications than in traditional DBMSs.

- *Semantic web*. The web is now evolving into the semantic web. The semantic web [5] is a web that is intelligent with machine-readable web pages. The major components of the semantic web include web infrastructures, web databases and services, ontology management and information integration. There has been much work on each of these areas. However, very little work has been devoted to access control. If the semantic web is to be effective, it is necessary to ensure that the information on the web is protected from unauthorized accesses and malicious modifications. Also, it must be ensured that individual's privacy is maintained. To cope with these issues, it is necessary to secure all the semantic web related technologies, such as XML, RDF, Agents, Databases, web services, and Ontologies and ensure the secure interoperation of all these technologies [13].

Cross-references

▶ Access Control Policy Languages
▶ Discretionary Access Control
▶ Mandatory Access Control
▶ Multilevel Secure Database Management System
▶ Role Based Access Control
▶ Storage Security

Recommended Reading

1. Air Force Studies Board, Committee on Multilevel Data Management Security. Multilevel data management security. National Research Council, 1983.
2. Berners-Lee T. et al. The semantic web. Scientific American, 2001.
3. Bertino E., and Sandhu R.S. Database security: concepts, approaches, and challenges. IEEE Trans. Dependable and Secure Computing, 2(1):2–19, 2005.
4. Bertino E., Khan L.R., Sandhu R.S., and Thuraisingham B.M. Secure knowledge management: confidentiality, trust, and privacy. IEEE Trans. Syst. Man Cybern. A, 36(3):429–438, 2006.
5. Carminati B., Ferrari E., and Perego A. Enforcing access control in web-based social networks. ACM trans. Inf. Syst. Secur., to appear.
6. Carminati B., Ferrari E., and Tan K.L. A framework to enforce access control over Data Streams. ACM Trans. Inf. Syst. Secur., to appear.
7. Carminati B., Ferrari E., and Thuraisingham B.M. Access control for web data: models and policy languages. Ann. Telecomm., 61 (3–4):245–266, 2006.
8. Carminati B., Ferrari E., and Bertino E. Securing XML data in third party distribution systems. In Proc. of the ACM Fourteenth Conference on Information and Knowledge Management, 2005.
9. Castano S., Fugini M.G., Martella G., and Samarati P. Database security. Addison Wesley, 1995.
10. Damiani M.L. and Bertino E. Access control systems for geo-spatial data and applications. In Modelling and management of geographical data over distributed architectures, A. Belussi, B. Catania, E. Clementini, E. Ferrari (eds.). Springer, 2007.
11. Fagin R. On an authorization mechanism. ACM Trans. Database Syst., 3(3):310–319, 1978.
12. Ferraiolo D.F., Sandhu R.S., Gavrila S.I., Kuhn D.R., and Chandramouli R. Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur., 4(3):224–274, 2001.
13. Ferrari E. and Thuraisingham B.M. Security and privacy for web databases and services. In Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 17–28.
14. Ferrari E. and Thuraisingham B.M. Secure database systems. In O. Diaz, M. Piattini (eds.). Advanced databases: technology and design. Artech House, 2000.
15. Griffiths P.P. and Wade B.W. An authorization mechanism for a relational database system. ACM Trans. Database Syst., 1 (3):242–255, 1976.
16. Lampson B.W. Protection. Fifth Princeton Symposium on Information Science and Systems, Reprinted in ACM Oper. Sys. Rev., 8(1):18–24, 1974.

Access Control Administration Policies

Elena Ferrari
University of Insubria, Varese, Italy

Synonyms

Authorization administration policies; Authorization administration privileges

Definition

Administration policies regulate who can modify the authorization state, that is, who has the right to grant and revoke authorizations.

Historical Background

Authorization management is a an important issue when dealing with access control and, as such, research on this topic is strongly related to the developments in access control. A milestone in the field is represented by the research carried out in the 1970s at IBM in the framework of the System R project. In particular, the work by Griffiths and Wade [9] defines a semantics for authorization revocation, which had greatly influenced the way in which authorization revocation has been implemented in commercial Relational DBMSs. Administrative policies for Object-oriented DBMSs have been studied in [8]. Later on, some extensions to the System R access control administration model, have been defined [3], with the aim of making it more flexible and adaptable to a variety of access control requirements. Additionally, as the research on extending the System R access control model with enhanced functionalities progresses, authorization administration has been studied for these extensions, such as temporal authorizations [2], strong and weak and positive and negative authorizations [4]. Also, administrative policies for new environments and data models such as WFMSs [1] and XML data [12] have been investigated. Back in the 1990s, when research on role-based access control began, administration policies for RBAC were investigated [6,11,10,13]. Some of the ideas developed as part of this research were adopted by the current SQL:2003 standard [7].

Foundations

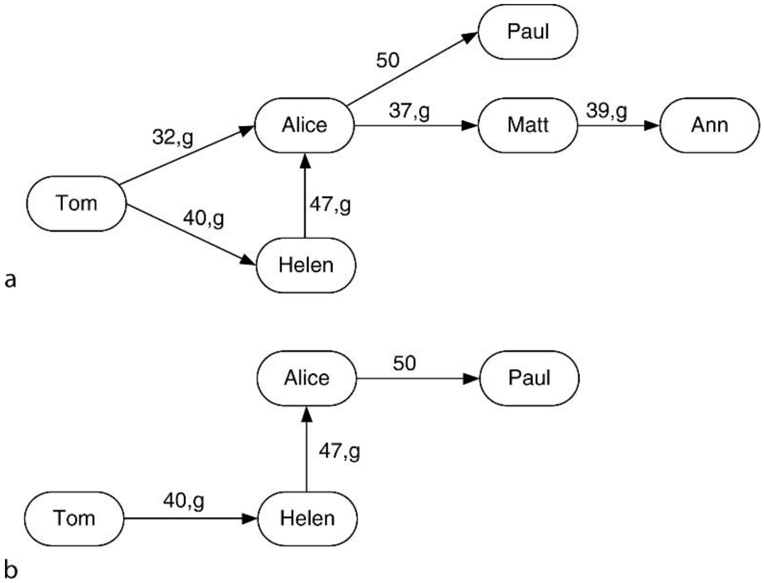
Access control administration deals with granting and revoking of authorizations. This function is usually regulated by proper *administration policies*. Usually, if mandatory access control is enforced, the adopted administration policies are very simple, so that the Security Administrator (SA) is the only one authorized to change the classification level of subjects and objects. In contrast, discretionary and role-based access control are characterized by more articulated administration policies, which can be classified according to the following categories [3]:

- *SA administration*. According to this policy, only the SA can grant and revoke authorizations. Although the SA administration policy has the advantage of being very simple and easily implemented, it has the disadvantage of being highly centralized (even though different SAs can manage different portions of the database) and is seldom used in current DBMSs, apart from very simple systems.
- *Object owner administration*. This is the policy commonly adopted by DBMSs and operating systems. Under this policy, whoever creates an object become its owner and he/she is the only one authorized to grant and revoke authorizations on the object.
- *Joint administration*. Under this policy, particularly suited for collaborative environments, several subjects are jointly responsible for administering specific authorizations. For instance, under the joint administration policy it can be a requirement that the authorization to write a certain document is given by two different users, such as two different job functions within an organization. Authorizations for a subject to access a data object requires that all the administrators of the object issue a grant request.

The object owner administration policy can be further combined with *administration delegation*, according to which the administrator of an object can grant other subjects the right to grant and revoke authorizations on the object. Delegation can be specified for selected privileges, for example only for read operations. Most current DBMSs support the owner administration policy with delegation. For instance, the `Grant` command provided by the SQL:2003 standard [7] supports a `Grant Option` optional clause. If a privilege p is granted with the grant option on an object o, the subject receiving it is not only authorized to exercise p on object o but he/she is also authorized to grant other subjects authorizations for p on object o with or without the grant option. Moreover, SQL:2003 provides an optional `Admin Option` clause, which has the same meaning as the Grant option clause but it applies to roles instead of to standard authorizations. If a subject is granted the authorization to play a role with the admin option he/she not only receives all the authorizations associated with the role, but he/she can also authorize other subjects to play that role.

If administration delegation is supported, different administrators can grant the same authorization to the same subject. A subject can therefore receive an authorization for the same privilege on the same object by different sources. An important issue is therefore related to the management of revoke operations, that is, what happens when a subject revokes some of the authorizations he/she previously granted. For instance, consider three users: Ann, Tom, and Alice. Suppose that Ann grants Tom the privilege to select tuples from the Employee relation with the grant option and that, by having this authorization, Tom grants Alice the same privilege on the Employee relation. What happens to the authorization of Alice when Ann revokes Tom the privilege to select tuples from the Employee relation? The System R authorization model [9] adopts the most conscious approach with respect to security by enforcing *recursive revocation*: whenever a subject revokes an authorization on a relation from another subject, all the authorizations that the revokee had granted because of the revoked authorization are recursively removed from the system. The revocation is iteratively applied to all the subjects that received an authorization from the revokee. In the example above, Alice will lose the privilege to select tuples from the Employee relation when Ann revokes this privilege to Tom.

Implementing recursive revocation requires keeping track of the *grantor* of each authorization, that is, the subject who specifies the authorization, since the same authorization can be granted by different subjects, as well as of its *timestamp*, that is, the time when it was specified. To understand why the timestamp is important in correctly implementing recursive revocation, consider the graph in Fig. 1a, which represents the authorization state for a specific privilege p on a specific object o. Nodes represent subjects, and an edge from node n_1 to node n_2 means that n_1 has granted privilege p on object o to n_2. The edge is labeled with the timestamp of the granted privilege and, optionally, with symbol "g," if the privilege has been granted with the grant option. Suppose that Tom revokes the authorization to Alice. As a result, the authorizations also held by Matt and Ann are recursively revoked because they could not have been granted if Alice did not receive authorization from Tom at time 32. In contrast, the authorization held by Paul is not revoked since it could have been granted even without the authorization granted by Tom to Alice at time 32, because of the privilege Alice had received by Helen at time



Access Control Administration Policies. Figure 1. Recursive revocation.

47. The authorization state resulting from the revoke operation is illustrated in Fig. 1b. Although recursive revocation has the advantage of being the most conservative solution with regard to security, it has the drawback of in some cases the unnecessarily revoking of too many authorizations. For instance, in an organization, the authorizations a user possesses are usually related to his/her job functions within the organization, rather than to his/her identity. If a user changes his/her tasks (for instance, because of a promotion), it is desirable to remove only the authorizations of the user, without revoking all the authorizations granted by the user before changing his/her job function. For this reason, research has been carried out to devise alternative semantics for the revoke operation with regard to recursive revocation. Bertino et al. [5] have proposed an alternative type of revoke operation, called *noncascading revocation*. According to this, no recursive revocation is performed upon the execution of a revoke operation. Whenever a subject revokes a privilege on an object from another subject, all authorizations which the subject may have granted using the privilege received by the revoker are not removed. Instead, they are restated as if they had been granted by the revoker.

SQL:2003 [7] adopts the object owner administration policy with delegation. A revoke request can either be issued to revoke an authorization from a subject for a particular privilege on a given object, or to revoke the authorization to play a given role. SQL:2003 supports two different options for the revoke operation. If the revoke operation is requested with the `Restrict` clause, then the revocation is not allowed if it causes the revocation of other privileges and/or the deletion of some objects from the database schema. In contrast, if the `Cascade` option is specified, then the system implements a revoke operation similar to the recursive revocation of the System R, but without taking into account authorization timestamps. Therefore, an authorization is recursively revoked only if the grantor no longer holds the grant/admin option for that, because of the requested revoke operation. Otherwise, the authorization is not deleted, regardless of the time the grantor had received the grant/admin option for that authorization. To illustrate the differences with regard to recursive revocation, consider once again Fig. 1a, and suppose that `Tom` revokes privilege p on object o to `Alice` with the `Cascade` option. With difference to the System R access control model, this revoke operation does not cause any other changes to the authorization state. The authorization granted by `Alice` to `Matt` is not deleted, because `Alice` still holds the grant option for that access (received by `Helen`).

Key Applications

Access control administration policies are fundamental in every environment where access control services are provided.

Cross-references

- ▶ Access Control
- ▶ Discretionary Access Control
- ▶ Role Based Access Control

Recommended Reading

1. Atluri V., Bertino E., Ferrari E., and Mazzoleni P. Supporting delegation in secure workflow management systems. In Proc. 17th IFIP WG 11.3 Conf. on Data and Application Security, 2003, pp. 190–202.
2. Bertino E., Bettini C., Ferrari E., and Samarati P. Decentralized administration for a temporal access control model. Inf. Syst., 22:(4)223–248, 1997.
3. Bertino E. and Ferrari E. Administration policies in a multipolicy authorization system. In Proc. 11th IFIP WG 11.3 Conference on Database Security, 1997, pp. 341–355.
4. Bertino E., Jajodia S., and Samarati P. A flexible authorization mechanism for relational data management systems. ACM Trans. Inf. Syst., 17:(2)101–140, 1999.
5. Bertino E., Samarati P., and Jajodia S. An extended authorization model. IEEE Trans. Knowl. Data Eng., 9:(1)85–101, 1997.
6. Crampton J. and Loizou G. Administrative scope: a foundation for role-based administrative models. ACM Trans. Inf. Syst. Secur., 6:(2)201–231, 2003.
7. Database Languages – SQL,ISO/IEC 9075-*, 2003.
8. Fernandez E.B., Gudes E., and Song H. A model for evaluation and administration of security in object-oriented databases. IEEE Trans. Knowl. Data Eng., 6:(2)275–292, 1994.
9. Griffiths P.P. and Wade B.W. An authorization mechanism for a relational database system. ACM Trans. Database Syst., 1:(3) 242–255, 1976.
10. Oh S., Sandhu R.S., and Zhang X. An effective role administration model using organization structure. ACM Trans. Inf. Syst. Secur., 9:(2)113–137, 2006.
11. Sandhu R.S., Bhamidipati V., and Munawer Q. The ARBAC97 model for role-based administration of roles. ACM Trans. Inf. Syst. Secur., 2:(1)105–135, 1999.
12. Seitz L., Rissanen E., Sandholm T., Sadighi Firozabadi B., and Mulmo O. Policy Administration control and delegation using XACML and delegent. In Proc. 6th IEEE/ACM Int. Workshop on Grid Computing, 2005, pp. 49–54.
13. Zhang L., Ahn G., and Chu B. A rule-based framework for role-based delegation and revocation. ACM Trans. Inf. Syst. Secur., 6:(3)404–441, 2003.

Access Control Policy Languages

ATHENA VAKALI
Aristotle University, Thessaloniki, Greece

Synonyms

Authorization policy languages

Definition

An access control policy language is a particular set of grammar, syntax rules (logical and mathematical), and operators which provides an abstraction-layer for access control policy specifications. Such languages combine individual rules into a single policy set, which is the basis for (user/subject) authorization decisions on accessing content (object) stored in various information resources. The operators of an access control policy language are used on attributes of the subject, resource (object), and their underlying application framework to facilitate identifying the policy that (most appropriately) applies to a given action.

Historical Background

The evolution of access control policy languages is inline with the evolving large-scale highly distributed information systems and the Internet, which turned the tasks of authorizing and controlling of accessing on a global enterprise (or on Internet) framework increasingly challenging and difficult. Obtaining a solid and accurate view of the policy in effect across its many and diverse systems and devices has guided the development of access control policy languages accordingly.

Access control policy languages followed the Digital Rights Management (DRM) standardization efforts, which had focused in introducing DRM technology into commercial and mainstream products. Originally, access control was practiced in the most popular RDBMSs by policy languages that were SQL based. Certainly, the access control policy languages evolution was highly influenced by the wide adoption of XML (late 1990s) mainly in the enterprise world and its suitability for supporting access control acts. XML's popularity resulted in an increasing need to support more flexible provisional access decisions than the initial simplistic authorization acts which were limited in an accept/deny decision. In this context, proposals of various access control policy languages were very active starting around the year 2000. This trend seemed to stabilize around 2005.

The historical pathway of such languages should highlight the following popular and general-scope access control policy languages:

- 1998: the Digital Property Rights Language (DPRL, Digital Property Rights Language, http://xml.coverpages.org/dprl.html) mostly addressed to commercial and enterprise communities was specified for describing rights, conditions, and fees to support commerce acts
- 2000: XML Access Control Language (XACL, XML Access Control Language, http://xml.coverpages.org/xacl.html) was the first XML-based access control language for the provisional authorization model
- 2001: two languages were publicized:
  - ► the eXtensible rights Markup Language (XrML, The Digital Rights Language for Trusted Content and Services, http://www.xrml.org/) promoted as the digital rights language for trusted content and services
  - ► the Open Digital Rights Language (ODRL, Open Digital Rights Language, http://odrl.net/) for developing and promoting an open standard for rights expressions for transparent use of digital content in all sectors and communities
- 2002: the eXtensible Media Commerce Language (XMCL, eXtensible Media Commerce Language, http://www.w3.org/TR/xmcl/) to communicate usage rules in an implementation-independent manner for interchange between business systems and DRM implementations
- 2003: the eXtensible Access Control Markup Language (XACML, eXtensible Access Control Markup Language, http://www.oasis-open.org/committees/xacml/) was accepted as a new OASIS, Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org/, Open Standard language, designed as an XML specification with emphasis on expressing policies for information access over the Internet.
- 2005: Latest version XACML 2.0 appeared and policy languages which are mostly suited for Web services appear. These include WS-SecurityPolicy, http://www-128.ibm.com/developerworks/library/specification/ws-secpol/, which defines general security policy assertions to be applied into Web services security frameworks.

Foundations

Since Internet and networks in general are currently the core media for data and knowledge exchange, a primary issue is to assure authorized access to (protected) resources located in such infrastructures. To support access control policies and mechanisms, the use of an appropriate and suitable language is the core requirement in order to express all of the various components of access control policies, such as subjects, objects, constraints, etc. Initial attempts for expressing access control policies (consisting of authorizations) involved primary "participants" in a policy, namely the `subject` (client requesting access), the `object` (protected resource), and the `action` (right or type of access).

To understand the access control policy languages the context in which they are applied must be explained. Hence, the following notions which appear under varying terminology must be noted:

- *Content/objects*: Any physical or digital content which may be of different formats, may be divided into subparts and must be uniquely identified. Objects may also be encrypted to enable secure distribution of content.
- *Permissions/rights/actions*: Any task that will enforce permissions for accessing, using and acting over a particular content/object. They may contain constraints (limits), requirements (obligations), and conditions (such as exceptions, negotiations).
- *Subjects/users/parties*: Can be humans (end users), organizations, and defined roles which aim in consuming (accessing) content.

Under these three core entities, the policies are formed under a particular language to express offers and agreements. Therefore, the initial format of such languages authorization was (subject, object, and action) defining which subject can conduct what type of action over what object. However, with the advent of databases, networking, and distributed computing, users have witnessed (as presented in the section "Historical background") a phenomenal increase in the automation of organizational tasks covering several physical locations, as well as the computerization of information related services [6,7]. Therefore, new ideas have been added into modern access control models, like time, tasks, origin, etc. This was evident in the evolution of languages which initially supported an original syntax for policies limited in a

Access Control Policy Languages. Table 1. Summary of most popular access control policy languages

Language/ technology	Subject types	Object types	Protection granularity	Accessing core formats	Focus
DPRL/XML DTDs	Registered users	Digital XML data sources, stored on repositories	Fine-grained	Digital licenses assigned for a time-limited period	
XACL/XML syntax	Group or organization members	Particular XML documents	Fine-grained	Set of particular specified privileges	
XrML/XML schema	Registered users and/or parties	digital XML data sources	Fine-grained	Granted rights under specified conditions	
ODRL/open-source schema-valid XML syntax	Any user	Trusted or untrusted content	coarse-grained	Digital or physical rights	
XMCL/XML namespaces	Registered users	Trusted multimedia content	Coarse-grained	Specified keyword-based licenses	Particular business models
XACML/XML schema	Any users organized in categories	Domain-specific input	Fine-grained	Rule-based permissions	
WS-Security policy/ XML, SOAP	Any Web users/ Web services	Digital data sources	Fine-grained	Protection acts at SOAP Web services messages level	Web services security

3-tuple (subject, Subject primitive allows user IDs, groups, and/or role names. object, Object primitive allows granularity as fine as a single element within an XML document, and action, Action primitive consists of four kinds of actions: read, write, create, and delete.) which then was found quite simplistic and limited and it was extended to include non-XML documents, to allow roles and collections as subjects and to support more actions (such as approve, execute, etc).

Table 1 summarizes the most important characteristics of the popular general scope access control policy languages. It is evident that these languages differentiate on the subjects/users types, on the protected object/content type (which is considered as trusted when it is addressed to trusted audience/users) and on the capabilities of access control acts, which are presented under various terms and formats (rights, permissions, privileges, etc). Moreover, this table highlights the level at which the access control may be in effect for each language, i.e., the broad categorization into fine- and coarse-grained protection granularity, respectively, refers to either partitions/detailed or full document/object protection capability. Moreover, the extensibility of languages which support Web-based objects and content is noted.

To expand on the above, specific-scope languages have also emerged mainly to support research-oriented applications and tools. The most representative of such languages include:

- X-Sec [1]: To support the specification of subject credentials and security policies in Author-X and Decentral Author-X [2]. X-Sec adopts the idea of credentials which is similar to roles in that one user can be characterized by more than one credentials.

Access Control Policy Languages. Table 2. Specific-scope access control languages characteristics

	X-Sec	XACL	RBXAC	XAS syntax
Objects				
Protected resources	XML documents and DTDs	XML documents and DTDs	XML documents	XML documents and DTDs
Identification	XPath	XPath	XPath	XPath
Protection granularity	Content, attribute	Element	Content, attribute	Element
Subjects				
Identification	XML-expressed credentials	Roles, UIDs, groups	Roles	User ID, location
Grouping of subjects	No	Yes	No	Yes
Subjects hierarchy	No	Yes	Role trees	Yes
Support public subject	No	Yes	No	Yes
Policies				
Expressed in	Policy base	XACL policy file	Access control files	XAS
Closed/open	Closed	Both	Closed	Closed
Permissions/denials	Both	Both	Permissions	Both
Access modes	Authoring, browsing	Read, write, create, delete	RI, WI, RC, WC	Read
Propagation	No-prop, first-level, cascade	No/up/down	According to role tree	Local, recursive
Priority	Implicit rules	ntp, ptp, dtd	-	Hard, soft
Conflict resolution	Yes	According to priorities and implicit rules	-	Implicitly, explicitly
Other issues				
Subscription-based	Yes	Yes	Yes	Yes
Ownership	No	No	Yes	No

- XAS Syntax: Designed to support the ACP (Access Control Processor) tool [3]. It is a simplified XML-based syntax for expressing authorizations.
- RBXAC: A specification XML-based language supporting the role-based access control model [4].
- XACL: Which was originally based on a provisional authorization model and it has been designed to support ProvAuth (Provisional Authorizations) tool. Its main function is to specify security policies to be enforced upon accesses to XML documents.
- Cred-XACL [5]: A recent access control policy language focusing on credentials support on distributed systems and the Internet.

The core characteristics of these specific-scope languages are given in Table 2, which summarizes them with respect to their approach for objects and subjects management, their policies practicing and their subscription and ownership mechanisms. Such a summary is important in order to understand the "nature" of each such language in terms of objects and subjects identification, protection (sources) granularity and (subject) hierarchies, policies expression and accessing modes under prioritization, and conflict resolution constraints. Finally, it should be noted that these highlighted characteristics are important in implementing security service tasks which support several security requirements from both the system and the sources perspective.

Key Applications

Access control policy languages are involved in the transparent and innovative use of digital resources which are accessed in applications related to key nowadays areas such as publishing, distributing and consuming of electronic publications, digital images, audio and movies, learning objects, computer software and other creations in digital form.

Future Directions

From the evolution of access control policy languages, it appears that, in the future, emphasis will be given on languages that are mostly suited for Web-accessed repositories, databases, and information sources. This trend is now apparent from the increasing interest on languages that control accessing on Web services and Web data sources. At the same time, it manages the challenges posed by acknowledging and identifying users/subjects on the Web.

URL to Code

Code, examples, and application scenarios may be found for: ODRL application scenarios at http://www.w3.org/TR/odrl/#46354 and http://odrl.net/, XrML at http://www.xrml.org/, XMCL at http://www.w3.org/TR/xmcl/, XACML at http://www.oasis-open.org/committees/xacml/, and WS-SecurityPolicy at http://www-128.ibm.com/developerworks/library/specification/ws-secpol/.

Cross-references

- ▶ Access Control
- ▶ Database Security
- ▶ Role-Based Access Control
- ▶ Secure Database Development

Recommended Reading

1. Bertino E., Castano S., and Ferrari E. On specifying security policies for web documents with an XML-based language. In Proc. 6th ACM Symp. on Access Control Models and Technologies, 2001, pp. 57–65.
2. Bertino E., Castano S., and Ferrari E. Securing XML documents with author-X. IEEE Internet Computing, May–June 2001, pp. 21–31.
3. Damiani E., De Capitani di Vimercati S., Paraboschi S., and Samarati P. Desing and implementation of an access control processor for XML documents. In Proc. 9th Int. World Wide Web Conference, 2000, pp. 59–75.
4. He H. and Wong R.K. A role-based access control model for XML repositories. In Proc. 1st Int. Conf. on Web Information Systems Eng., 2000, pp. 138–145.
5. Stoupa K. Access Control Techniques in distributed systems and the Internet, Ph.D. Thesis, Aristotle University, Department of Informatics, 2007.
6. Stoupa K. and Vakali A. Policies for web security services, Chapter III. In Web and Information Security, E. Ferrari, B. Thuraisingham (eds.), Idea-Group Publishing, USA, 2006.
7. Vuong N.N., Smith G.S., and Deng Y. Managing security policies in a distributed environment using eXtensible markup language (XML). In Proc. 16th ACM Symp. on Applied Computing, 2001, pp. 405–411.

Access Methods

▶ Access Path

Access Path

Evaggelia Pitoura
University of Ioannina, Ioannina, Greece

Synonyms

Access path; Access methods

Definition

An access path specifies the path chosen by a database management system to retrieve the requested tuples from a relation. An access path may be either (i) a sequential scan of the data file or (ii) an index scan with a matching selection condition when there are indexes that match the selection conditions in the query. In general, an index matches a selection condition, if the index can be used to retrieve all tuples that satisfy the condition.

Key Points

Access paths are the alternative ways for retrieving specific tuples from a relation. Typically, there is more than one way to retrieve tuples because of the availability of indexes and the potential presence of conditions specified in the query for selecting the tuples. Typical access methods include sequential access of unordered data files (heaps) as well as various kinds of indexes. All commercial database systems implement heaps and B+ tree indexes. Most of them also support hash indexes for equality conditions.

To choose an access path, the optimizer first determines which matching access paths are available by examining the conditions specified by the query. Then, it estimates the selectivity of each access path using any available statistics for the index and data file. The *selectivity of an access path* is the number of pages (both index and data pages) accessed when the specific access path is used to retrieve the requested tuples. The access path having the smallest selectivity is called the most *selective access path*. Clearly, using the most selective access path minimizes the cost of data retrieval. Additional information can be found in [1].

Cross-references

- ▶ Index Structures for Biological Sequences
- ▶ Query Optimization
- ▶ Selectivity Estimation

Recommended Reading

1. Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., Price T.G. Access path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.

Accountability

▶ Auditing and Forensic Analysis

ACID Properties

Gottfried Vossen
University of Münster, Münster, Germany

Synonyms

ACID properties; Atomicity; Isolation; Consistency preservation; Durability; Persistence

Definition

The conceptual *ACID properties* (short for atomicity, isolation, consistency preservation, and durability) of a transaction together provide the key abstraction which allows application developers to disregard irregular or even malicious effects from concurrency or failures of transaction executions, as the transactional server in charge guarantees the consistency of the underlying data and ultimately the correctness of the application [1–3]. For example, in a banking context where debit/credit transactions are executed this means that no money is ever lost in electronic funds transfers and customers can rely on electronic receipts and balance statements. These cornerstones for building highly dependable information systems can be successfully applied outside the scope of online transaction processing and classical database applications as well.

Key Points

The *ACID properties* are what a database server guarantees for transaction executions, in particular in the presence of multiple concurrently running transactions and in the face of failure situations; they comprise the following four properties (whose initial letters form the word "ACID"):

Atomicity. From the perspective of a client and an application program, a transaction is executed completely or not at all, i.e., in an all-or-nothing fashion. So the effects of a program under execution on the underlying data server(s) will only become visible to the outside world or to other program executions if and when the transaction reaches its "*commit*" operation. This case implies that the transaction could be processed completely, and no errors whatsoever were discovered while it was processed. On the other hand, if the program is abnormally terminated before reaching its commit operation, the data in the underlying data servers will be left in or automatically brought back to the state in which it was before the transaction started, i.e., the data appears as if the transaction had never been invoked at all.

Consistency preservation: Consistency constraints that are defined on the underlying data servers (e.g., keys, foreign keys) are preserved by a transaction; so a transaction leads from one consistent state to another. Upon the commit of a transaction, all integrity constraints defined for the underlying database(s) must be satisfied; however, between the beginning and the end of a transaction, inconsistent intermediate states are tolerated and may even be unavoidable. This property generally cannot be ensured in a completely automatic manner. Rather, it is necessary that the application is programmed such that the code between the beginning and the commit of a transaction will eventually reach a consistent state.

Isolation: A transaction is isolated from other transactions, i.e., each transaction behaves as if it was operating alone with all resources to itself. In particular, each transaction will "see" only consistent data in the underlying data sources. More specifically, it will see only data modifications that result from committed transactions, and it will see them only in their entirety, and never any effects of an incomplete transaction. This is the decisive property that allows to hide the fallacies and pitfalls of concurrency from the application developers. A sufficient condition for isolation is that concurrent executions are equivalent to sequential ones, so that all transactions appear as if they were executed one after the other rather than in an interleaved manner; this condition is made precise through serializability.

Durability: When the application program from which a transaction derives is notified that the transaction has been successfully completed (i.e., when the commit point of the transaction has been reached), all updates the transaction has made in the underlying data servers are guaranteed to survive subsequent software or hardware failures. Thus, updates of committed transactions are durable (until another transaction later modifies the same data items) in that they persist even across failures of the affected data server(s).

Therefore, a transaction is a set of operations executed on one or more data servers which are issued by an application program and are guaranteed to have the ACID properties by the runtime system of the involved servers. The "ACID contract" between the application program and the data servers requires the program to demarcate the boundaries of the transaction as well as the desired outcome – successful or abnormal termination – of the transaction, both in a dynamic manner. There are two ways a transaction can finish: it can commit, or it can abort. If it commits, all its changes to the database are installed, and they will remain in the database until some other application makes further changes. Furthermore, the changes will seem to other programs to take place together. If the transaction aborts, none of its changes will take effect, and the DBMS will rollback by restoring previous values to all the data that was updated by the application program. A programming interface of a transactional system consequently needs to offer three types of calls: (i) "*begin transaction*" to specify the beginning of a transaction, (ii) "*commit transaction*" to specify the successful end of a transaction, and (iii) "*rollback transaction*" to specify the unsuccessful end of a transaction with the request to abort the transaction.

The core requirement for a transactional server is to provide the ACID guarantees for sets of operations that belong to the same transaction issued by an application program requires that the server. This requires a *concurrency control* component to guarantee the isolation properties of transactions, for both committed and aborted transactions, and a *recovery* component to guarantee the atomicity and durability of transactions. The server may or may not provide explicit support for consistency preservation. In addition to the ACID contract, a transactional server should meet a number of technical requirements: A transactional data server (which most often will be a database system) must provide *good performance* with a given hardware/software configuration, or more generally, a good cost/performance ratio when the configuration is not yet fixed. Performance typically refers to the two metrics of *high throughput*, which is defined as the number of successfully processed transactions per time unit, and of *short response times*, where the response time of a transaction is defined as the time span between issuing the transaction and its successful completion as perceived by the client.

While the ACID properties are crucial for many applications in which the transaction concept arises, some of them are too restrictive when the transaction model is extended beyond the read/write context. For example, business processes can be cast into various forms of *business transactions*, i.e., long-running transactions for which atomicity and isolation are generally too strict. In these situations, additional or alternative guarantees need to be employed.

Cross-references

▶ Atomicity
▶ Concurrency Control
▶ Extended Transaction Models
▶ Multi-Level Recovery and the ARIES Algorithm
▶ Serializability
▶ Snapshot Isolation
▶ SQL Isolation Levels
▶ Transaction Model

Recommended Reading

1. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, MA, 1987.
2. Bernstein P.A. and Newcomer E. Principles of Transaction Processing for the Systems Professional. Morgan Kaufmann, San Francisco, CA, 1997.
3. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1993.

ACID Transaction

▶ Transaction

Acquisitional Query Languages

▶ Database Languages for Sensor Networks

Active and Real-Time Data Warehousing

Mukesh Mohania[1], Ullas Nambiar[1], Michael Schrefl[2], Millist Vincent[3]
[1]IBM India Research Lab, New Delhi, India
[2]University of Linz, Linz, Austria
[3]University of South Australia, Adelaide, SA, Australia

Synonyms

Right-time data warehousing

Definition

Active Data Warehousing is the technical ability to capture transactions when they change, and integrate them into the warehouse, along with maintaining batch or scheduled cycle refreshes. An active data warehouse offers the possibility of automating routine tasks and decisions. The active data warehouse exports decisions automatically to the On-Line Transaction Processing (OLTP) systems.

Real-time Data Warehousing describes a system that reflects the state of the warehouse in real time. If a query is run against the real-time data warehouse to understand a particular facet about the business or entity described by the warehouse, the answer reflects the state of that entity at the time the query was run. Most data warehouses have data that are highly latent – or reflects the business at a point in the past. A real-time data warehouse has low latency data and provides current (or real-time) data.

Simply put, a real-time data warehouse can be built using an active data warehouse with a very low latency constraint added to it. An alternate view is to consider active data warehousing as being a design methodology suited to tactical decision-making based on very current data while real-time data warehousing is a collection of technologies that refresh a data warehouse frequently. A real-time data warehouse is one that acquires, cleanses, transforms, stores, and disseminates information in real time. An active data warehouse, on the other hand, operates in a non-real-time response mode with one-or-more OLTP systems.

Historical Background

A *data warehouse* is a decision support database that is periodically updated by extracting, transforming, and loading operational data from several OLTP databases.

In the data warehouse, OLTP data is arranged using the (multi) dimensional data modeling approach (see [1] for a basic approach and [2] for details on translating an OLTP data model into a dimensional model), which classifies data into *measures* and *dimensions*. In recent years, several multidimensional data models have been proposed [3–6]. An in-depth comparison is provided by Pedersen and Jensen in [5]. The basic unit of interest in a data warehouse is a *measure* or *fact* (e.g., sales), which represent countable, semisummable, or summable information concerning a business process. An instance of a measure is called *measure value*. A measure can be analyzed from different perspectives, which are called the *dimensions* (e.g., location, product, time) of the data warehouse [7]. A dimension consists of a set of *dimension levels* (e.g., time: Day, Week, Month, Quarter, Season, Year, ALLTimes), which are organized in multiple hierarchies or *dimension paths* [6] (e.g., Time[Day] → Time[Month] → Time[Quarter] → Time[Year] → Time[ALLTimes]; Time[Day] → Time [Week] → Time[Season] → Time[ALLTimes]). The hierarchies of a dimension form a lattice having at least one top dimension level and one bottom dimension level. The measures that can be analyzed by the same set of dimensions are described by a *base cube* or *fact table*. A base cube uses level instances of the lowest dimension levels of each of its dimensions to identify a measure value. The relationship between a set of measure values and the set of identifying level instances is called *cell*. Loading data into the data warehouse means that new *cells* will be added to *base cubes* and new level instances will be added to *dimension levels*. If a dimension D is related to a measure m by means of a base cube, then the hierarchies of D can be used to aggregate the *measure values* of m using operators like SUM, COUNT, or AVG. Aggregating measure values along the hierarchies of different dimensions (i.e., *rollup*) creates a multidimensional view on data, which is known as *data cube* or *cube*. Deaggregating the measures of a cube to a lower dimension level (i.e., *drilldown*) creates a more detailed cube. Selecting the subset of a cube's cells that satisfy a certain selection condition (i.e., *slicing*) also creates a more detailed cube.

The data warehouses are used by analysts to find solutions for decision tasks by using OLAP (On-Line Analytical Processing) [7] systems. The decision tasks can be split into three, viz. *non-routine, semi-routine*, and *routine*. Non-routine tasks occur infrequently and/or do not have a generally accepted decision criteria. For example, strategic business decisions such as introducing a new brand or changing an existing business policy are non-routine tasks. Routine tasks, on the other hand, are well structured problems for which generally accepted procedures exist and they occur frequently and at predictive intervals. Examples can be found in the areas of product assortment (change price, withdraw product, etc.), customer relationship management (grant loyalty discounts etc.), and in many administrative areas (accept/reject paper based on review scores). Semi-routine tasks are tasks that require a non-routine solution – e.g., paper rated contradictory must be discussed by program committee. Since, most tasks are likely to be routine, it is logical to automate processing of such tasks to reduce the delay in decision-making.

Active data warehouses [8] were designed to enable data warehouses to support automatic decision-making when faced with routine decision tasks and routinizable elements of semi-routine decision tasks. The active data warehouse design extends the technology behind active database systems. Active database technology transforms passive database systems into reactive systems that respond to database and external events through the use of rule processing features [9,10]. Limited versions of active rules exist in commercial database products [11,12].

Real-time data warehousing captures business activity data as it occurs. As soon as the business activity is complete and there is data about it, the completed activity data flows into the data warehouse and becomes available instantly. In other words, real-time data warehousing is a framework for deriving information from data as the data becomes available. Traditionally, data warehouses were regarded as an environment for analyzing historic data, either to understand what has happened or simply to log the changes as they happened. However, of late, businesses want to use them to predict the future: e.g., to predict customers likely to churn; and thereby seek better control of the business. However, until recently, it was not practical to have *zero-latency* data warehouses – the process of extracting data had too much of an impact on the source systems concerned, and the various steps needed to cleanse and transform the data required multiple temporary tables and took several hours to run. However, the increased visibility of (the value of) warehouse data, and the take-up by a wider audience within the organization, has lead to a number of product developments by IBM [13], Oracle [14], and other vendors that make real-time data warehousing now possible.

Foundations

The two example scenarios below describe typical situations in which active rules can be used to automate decision-making:

Scenario 1: Reducing the price of an article. Twenty days after a soft drink has been launched on a market, analysts compare the quantities sold during this period with a standardized indicator. This indicator requires the total quantities sold during the 20-day period do not drop below a threshold of 10,000 sold items. If the analyzed sales figures are below this threshold, the price of the newly launched soft drink will be reduced by 15.

Scenario 2 : Withdrawing articles from a market. At the end of every quarter, high-priced soft drinks which are sold in Upper Austrian stores will be analyzed. If the sales figures of a high-priced soft drink have continuously dropped, the article will be withdrawn from the Upper Austrian market. Analysts inspect sales figures at different granularities of the time dimension and at different granularities of the location dimension. Trend, average, and variance measures are used as indicators in decision-making.

Rules that mimic the analytical work of a business analyst are called *analysis rules* [8]. The components of analysis rules constitute the *knowledge model* of an active data warehouse (and also a real-time data warehouse). The knowledge model determines *what* an analyst must consider when he specifies an active rule to automate a routine decision task.

An analysis rule consists of (i) the *primary dimension level* and (ii) the *primary condition*, which identify the objects for which decision-making is necessary, (iii) the *event*, which triggers rule processing, (iv) the *analysis graph*, which specifies the cubes for analysis, (v) the *decision steps*, which represent the conditions under which a decision can be made, and (vi) the *action*, which represents the rule's decision task. Below is a brief description of the components of an analysis rule. Detailed discussion is given in [8].

Event: Events are used to specify the timepoints at which analysis rules should be carried out. Active data warehouses provide three kinds of events: (i) OLTP method events, (ii) relative temporal events, and (iii) calendar events. OLTP method events describe basic happenings in the data warehouse's sources. Relative temporal events are used to define a temporal distance between such a basic happening and carrying out an analysis rule. Calendar events represent fixed points in time at which an analysis rule may be carried out. Structurally, every event instance is characterized by an occurrence time and by an event identifier. In its event part, an analysis rule refers to a calendar event or to a relative temporal event.

An *OLTP method event* describes a happening in the data warehouse's source systems that is of interest to analysis rules in the active data warehouse. Besides occurrence time and event identifier, the attributes of an OLTP method event are a reference to the dimension level for which the OLTP method event occurred and the parameters of the method invocation. To make OLTP method events available in data warehouses, a data warehouse designer has to define the schema of OLTP method events and extend the data warehouse's extract/transform/load mechanism. Since instances of OLTP method events are loaded some time after their occurrence, analysis rules cannot be triggered directly by OLTP method events.

Temporal events determine the timepoints at which decision-making has to be initiated. Scenario 1 uses the relative temporal event "twenty days after launch" while Scenario 2 uses the periodic temporal event "end of quarter." The *conditions* for decision-making are based on indicators, which have been established in manual decision-making. Each condition refers to a multidimensional cube and therefore "analyzing" means to evaluate the condition on this cube. Scenario 1 uses a quantity-based indicator, whereas scenario 2 uses value-based indicators for decision-making. The decision whether to carry out the rule's *action* depends on the result of evaluating the conditions. The action of scenario 1 is to reduce the price of an article, whereas the action of scenario 2 is to withdraw an article from a market.

Primary Condition: Several analysis rules may share the same OLTP method as their action. These rules may be carried out at different timepoints and may utilize different multidimensional analyses. Thus, a certain analysis rule usually analyzes only a subset of the level instances that belong to the rule's primary dimension level. The primary condition is used to determine for a level instance of the primary dimension level whether multidimensional analysis should be carried out by the analysis rule. The primary condition is specified as a Boolean expression, which refers to the describing attributes of the primary dimension level. If omitted, the primary condition evaluates to TRUE.

Action: The purpose of an analysis rule is to automate decision-making for objects that are available in OLTP systems and in the data warehouse. A *decision* means to invoke (or not to invoke) a method on a certain object in an OLTP system. In its action part, an analysis rule may refer to a single OLTP method of the primary dimension level, which represents a transaction in an OLTP system. These methods represent the *decision space* of an active data warehouse. To make the transactional behavior of an OLTP object type available in the active data warehouse, the data warehouse designer must provide (i) the specifications of the OLTP object type's methods together with required parameters, (ii) the preconditions that must be satisfied before the OLTP method can be invoked in the OLTP system, and (iii) a conflict resolution mechanism, which solves contradictory decisions of different analysis rules. Since different analysis rules can make a decision for the same level instance of the rules' primary dimension level during the same active data warehouse cycle, a *decision conflict* may occur. Such conflicts are considered as interrule conflicts. To detect interrule conflicts, a *conflict table* covering the OLTP methods of the decision space is used. The tuples of the conflict table have the form <*m*1, *m*2, *m*3> , where *m*1 and *m*2 identify two conflicting methods and *m*3 specifies the conflict resolution method that will be finally executed in OLTP systems. If a conflict cannot be solved automatically it has to be reported to analysts for manual conflict resolution.

Analysis Graph: When an analyst queries the data warehouse to make a decision, he or she follows an incremental topdown approach in creating and analyzing cubes. Analysis rules follow the same approach. To automate decision-making, an analysis rule must "know" the cubes that are needed for multidimensional analysis. These cubes constitute the *analysis graph*, which is specified once by the analyst. The *n* dimensions of each cube of the analysis graph are classified into one *primary dimension*, which represents the level instances of the primary dimension level, and $n - 1$ *analysis dimensions*, which represent the multidimensional space for analysis. Since a level instance of the primary dimension level is described by one or more cells of a cube, multidimensional analysis means to compare, aggregate, transform, etc., the measure values of these cells. Two kinds of multidimensional analysis are carried out at each cube of the analysis graph: (i) select the level instances of the primary dimension level whose cells comply with the decision-making condition (e.g., withdraw an article if the sales total of the last quarter is below USD 10,000) and (ii) select the level instances of the primary dimension level whose cells comply with the condition under which more detailed analysis (at finer grained cubes) are necessary (e.g., continue analysis if the sales total of the last quarter is below USD 500,000). The multidimensional analysis that is carried out on the cubes of the analysis graph are called decision steps. Each decision step analyzes the data of exactly one cube of the analysis graph. Hence, analysis graph and decision steps represent the knowledge for multidimensional analysis and decision-making of an analysis rule.

Enabling real-time data warehousing: As mentioned earlier, real-time data warehouses are active data warehouses that are loaded with data having (near) zero latency. Data warehouse vendors have used multiple approaches such as *hand-coded scripting* and data extraction, transformation, and loading (ETL) [15] solutions to serve the data acquisition needs of a data warehouse. However, as users move toward real-time data warehousing, there is a limited choice of technologies that facilitate real-time data delivery. The challenge is to determine the right technology approach or combination of solutions that best meets the data delivery needs. Selection criteria should include considerations for frequency of data, acceptable latency, data volumes, data integrity, transformation requirements and processing overhead. To solve the real-time challenge, businesses are turning to technologies such as enterprise application integration (EAI) [16] and transactional data management (TDM) [17], which offer high-performance, low impact movement of data, even at large volumes with sub-second speed. EAI has a greater implementation complexity and cost of maintenance, and handles smaller volumes of data. TDM provides the ability to capture transactions from OLTP systems, apply mapping, filtering, and basic transformations and delivers to the data warehouse directly. A more detailed study of the challenges involved in implementing a real-time data warehouse is given in [18].

Key Applications

Active and Real-time data warehouses enable businesses across all industry verticals to gain competitive advantage by allowing them to run *analytics* solutions over the most recent data of interest that is captured in the warehouse. This will provide them with the ability to make intelligent business decisions and better understand and predict customer and business trends based on accurate, up-to-the-second data. By introducing real-time flows of information to data warehouses, companies can increase supply chain visibility, gain a complete view of business performance, and increase service levels, ultimately increasing customer retention and brand value.

The following are some additional business benefits of active and real-time data warehousing:

- *Real-time Analytics:* Real-time analytics is the ability to use all available data to improve performance and quality of service at the moment they are required. It consists of dynamic analysis and reporting, right at the moment (or very soon after) the resource (or information) entered the system. In a practical sense, real time is defined by the need of the consumer (business) and can vary from a few seconds to few minutes. In other words, more frequent than daily can be considered real-time, because it crosses the overnight-update barrier. With increasing availability of active and real-time data warehouses, the technology for capturing and analyzing real-time data is increasingly becoming available. Learning how to apply it effectively becomes the differentiator. Implementing real-time analytics requires the integration of a number of technologies that are not interoperable off-the-shelf. There are no established best practices. Early detection of fraudulent activity in financial transactions is a potential environment for applying real-time analytics. For example, credit card companies monitor transactions and activate counter measures when a customer's credit transactions fall outside the range of expected patterns. However, being able to correctly identify fraud while not offending a well-intentioned valuable customer is a critical necessity that adds complexity to the potential solution.
- *Maximize ERP Investments:* With a real-time data warehouse in place, companies can maximize their Enterprise Resource Planning (ERP) technology investment by turning integrated data into business intelligence. ETL solutions act as an integral bridge between ERP systems that collect high volumes of transactions and business analytics to create data reports.
- *Increase Supply Chain Visibility:* Real-time data warehousing helps streamline supply chains through highly effective business-to-business communications and identifies any weak links or bottlenecks, enabling companies to enhance service levels and gain a competitive edge.
- *Live 360° View of Customers:* The active database solutions enable companies to capture, transform, and flow all types of customer data into a data warehouse, creating one seamless database that provides a 360° view of the customer. By tracking and analyzing all modes of interaction with a customer, companies can tailor new product offerings, enhance service levels, and ensure customer loyalty and retention.

Future Directions

Data warehousing has greatly matured as a technology discipline; however enterprises that undertake data warehousing initiatives continue to face fresh challenges that evolve with the changing business and technology environment. Most future needs and challenges will come in the areas of active and real-time data warehousing solutions. Listed below are some future challenges:

- *Integrating Heterogeneous Data Sources:* The number of enterprise data sources is growing rapidly, with new types of sources emerging every year. Enterprises want to integrate the unstructured data generated from customer emails, chat and voice call transcripts, feedbacks, and surveys with other internal data in order to get a complete picture of their customers and integrate internal processes. Other sources for valuable data include ERP programs, operational data stores, packaged and homegrown analytic applications, and existing data marts. The process of integrating these sources into a data warehouse can be complicated and is made even more difficult when an enterprise merges with or acquires another enterprise.
- *Integrating with CRM tools:* Customer relationship management (CRM) is one of the most popular business initiatives in enterprises today. CRM helps enterprises attract new customers and develop loyalty among existing customers with the end result of increasing sales and improving profitability. Increasingly, enterprises want to use the holistic view of the customer to deliver value-added services to

the customer based on her overall value to the enterprise. This would include, automatically identifying when an important life event is happening and sending out emails with necessary information and/or relevant products, gauging the mood of the customer based on recent interactions, and alerting the enterprise before it is too late to retain the customer and most important of all identifying customers who are likely to accept suggestions about upgrades of existing products/services or be interested in newer versions. The data warehouse is essential in this integration process, as it collects data from all channels and customer touch points, and presents a unified view of the customer to sales, marketing, and customer-care employees. Going forward, data warehouses will have to provide support for analytics tools that are embedded into the warehouse, analyze the various customer interactions continuously, and then use the insights to trigger *actions* that enable delivery of the above-mentioned value-added services. Clearly, this requires an active data warehouse to be tightly integrated with the CRM systems. If the enterprise has low latency for insight detection and value-added service delivery then a real-time data warehouse would be required.

- *In-built data mining and analytics tools:* Users are also demanding more sophisticated business intelligence tools. For example, if a telecom customer calls to cancel his call-waiting feature, real-time analytic software can detect this and trigger a special offer of a lower price in order to retain the customer. The need is to develop a new generation of data mining algorithms that work over data warehouses that integrate heterogeneous data and have self-learning features. These new algorithms must automate data mining and make it more accessible to mainstream data warehouse users by providing explanations with results, indicating when results are not reliable and automatically adapting to changes in underlying predictive models.

Cross-references

▶ Cube Implementations
▶ Data Warehouse Interoperability
▶ Data Warehousing Systems: Foundations and Architectures
▶ ETL
▶ Multidimensional Modeling
▶ On-Line Analytical Processing
▶ Query Processing in Data Warehouses
▶ Transformation

Recommended Reading

1. Kimball R. and Strethlo K. Why decision support fails and how to fit it. ACM SIGMOD Rec., 24(3):91–97, 1995.
2. Golfarelli M., Maio D., and Rizzi S. Conceptual design of data warehouses from E/R schemes. In Proc. 31st Annual Hawaii Int. Conf. on System Sciences, Vol. VII. 1998, pp. 334–343.
3. Lehner W. Modeling large scale OLAP scenarios. In Advances in Database Technology, Proc. 6th Int. Conf. on Extending Database Technology, 1998, pp. 153–167.
4. Li C. and Wang X.S. A data model for supporting on-line analytical processing. In Proc. Int. Conf. on Information and Knowledge Management, 1996, pp. 81–88.
5. Pedersen T.B. and Jensen C.S. Multidimensional data modeling for complex data. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 336–345.
6. Vassiliadis P. Modeling multidimensional databases, cubes and cube operations. In Proc. 10th Int. Conf. on Scientific and Statistical Database Management, 1998, 53–62.
7. Chaudhuri S. and Dayal U. An overview of data warehousing and OLAP technology. ACM SIGMOD Rec., 26(1):65–74, 1997.
8. Thalhammer T., Schrefl M., and Mohania M. Active data warehouses: complementing OLAP with analysis rules. Data Knowl. Eng., 39(3):241–269, 2001.
9. ACT-NET Consortium. The active database management system manifesto: a rulebase of ADBMS featueres. ACM SIGMOD Rec., 25(3), 1996.
10. Simon E. and Dittrich A. Promises and realities of active database systems. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 642–653.
11. Brobst S. Active data warehousing: a new breed of decision support. In Proc. 13th Int. Workshop on Data and Expert System Applications, 2002, pp. 769–772.
12. Borbst S. and Rarey J. The five stages of an active data warehouse evolution. Teradata Mag., 38–44, 2001.
13. IBM DB2 Data Warehouse Edition. http://www-306.ibm.com/software/data/db2/dwe/.
14. Rittman M. Implementing Real-Time Data Warehousing Using Oracle 10g. Dbazine.com. http://www.dbazine.com/datawarehouse/dw-articles/rittman5.
15. Kimball R. and Caserta J. The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. Wiley, 2004.
16. Linthicum R.S. Enterprise Application Integration. Addison-Wesley, 1999.
17. Improving SOA with Goldengate TDM Technology. GoldenGate White Paper, October 2007.
18. Langseth J. Real-Time Data Warehousing: Challenges and Solutions. DSSResources.COM, 2004.
19. Paton N.W. and Diaz O. Active Database Systems. ACM Comput. Surv., 1(31),1999.

Active Database, Active Database (Management) System

Mikael Berndtsson, Jonas Mellin
University of Skövde, Skövde, Sweden

Definition

An active database (aDB) or active database (management) system (aDBS/aDBMS) is a database (management) system that supports reactive behavior through ECA-rules.

Historical Background

The term active database was first used in the early 1980s [12]. Some related active database work was also done within the area of expert database systems in the mid 1980s, but it was not until the mid/late 1980s that the research on supporting ECA rules in database systems took off, for example [10,12,18]. During the 1990s the area was extensively explored through more than twenty suggested active database prototypes and a large body of publications:

- Seven workshops were held between 1993 and 1997: RIDS [12,16,17], RIDE-ADS [20], Dagstuhl Seminar [5] and ARTDB [3,4].
- Two special issues of journals [8,9] and one special issue of ACM Sigmod Record [1].
- Two text books [13,19] and one ACM Computing Survey paper [15].

In addition, the groups within the ACT-NET consortium (A European research network of Excellence on active databases 1993–1996) reached a consensus on what constitutes an active database management system with the publication of the Active Database System Manifesto [2].

Most of the active databases are monolithic and assume a centralized environment, consequently, the majority of the prototype implementations do not consider distributed issues. Initial work on how active databases are affected by distributed issues are reported in [7].

Foundations

An active database can automatically react to events such as database transitions, time events, and external signals in a timely and efficient manner. This is in contrast to traditional database systems, which are passive in their behaviors, so that they only execute queries and transactions when they are explicitly requested to do so.

Previous approaches to support reactive behavior can broadly be classified into:

- Periodically polling the database.
- Embedding or encoding event detection and related action execution in the application code.

The first approach implies that the queries must be run exactly when the event occurs. The frequency of polling can be increased in order to detect such an event, but if the polling is too frequent, then the database is overloaded with queries and will most often fail. On the other hand, if the frequency is too low, the event will be missed.

The second approach implies that every application which updates the database needs to be augmented with condition checks in order to detect events. For example, an application may be extended with code to detect whether the quantity of certain items has fallen below a given level. From a software engineering point of view, this approach is inappropriate, since a change in a condition specification implies that every application that uses the modified condition needs to be updated.

Neither of the two previous approaches can satisfactorily support reactive behavior in a database context [10]. An active database system avoids the previous disadvantages by moving the support for reactive behavior inside the database (management) system. Reactive behavior in an active database is supported by ECA-rules that have the following semantics: when an event is detected, evaluate a condition, and if the condition is true, execute an action.

Similar to describing an object by its static features and dynamic features, an active database can be described by its knowledge model (static features) and execution model (dynamic features). Thus, by investigating the knowledge model and execution model of an active database, one can identify what type of ECA rules that can be defined and how the active database behave at run-time.

Key Applications

An aDB or aDBS/aDBMS is useful for any non-mission critical application that require reactive behavior.

Future Directions

Looking back, the RIDS'97 workshop marks the end of the active database period, since there are very few active database publications after 1997. However, the concept of ECA-rules has resurfaced and has been picked up by other research communities such as Complex Event Processing and Semantic Web. In contrast to typical active database approaches that assume a centralized environment, the current research on ECA rules within Complex Event Processing and Semantic Web assume that the environment is distributed and heterogeneous. Thus, as suggested within the REWERSE project [3], one cannot assume that the event, condition, and action parts of an ECA rule are defined in one single ECA rule language. For example, the event part of an ECA-rule can be defined in one language (e.g., Snoop), whereas the condition part and action part are defined in a completely different rule language.

The popularity of using XML for manipulating data has also led to proposals of ECA rule markup languages. These ECA rule markup languages are used for storing information about ECA rules and facilitates exchange of ECA-rules between different rule engines and applications.

One research question that remains from the active database period is how to model and develop applications that use ECA rules. Some research on modeling ECA rules has been carried out, but there is no widely agreed approach for modeling ECA rules explicitly in UML, or how to derive ECA rules from existing UML diagrams.

Cross-references

- ▶ Active Database Execution Model
- ▶ Active Database Knowledge Model
- ▶ Complex Event Processing
- ▶ ECA Rules

Recommended Reading

1. ACM SIGMOD Record. Special Issue on Rule Management and Processing in Expert Databases, 1989.
2. ACT-NET Consortium The active database management system manifesto: a rulebase of ADBMS features. ACM SIGMOD Rec., 25(3):40–49, 1996.
3. Alferes J.J., Amador R., and May W. A general language for evolution and reactivity in the semantic Web. In Proc. 3rd Workshop on Principles and Practice of Semantic Web Reasoning, 2005, pp. 101–115.
4. Andler S.F. and Hansson J. (eds.). In Proc. 2nd International Workshop on Active, Real-Time, and Temporal Database Systems, LNCS, vol. 1553, Springer, 1998.
5. Berndtsson M. and Hansson J. Workshop report: the first international workshop on active and real-time database systems. SIGMOD Rec., 25(1):64–66, 1996.
6. Buchmann A., Chakravarthy S., and Dittrich K. Active Databases. Dagstuhl Seminar No. 9412, Report No. 86, 1994.
7. Bültzingsloewen G., Koschel A., Lockemann P.C., and Walter H.D. ECA Funtionality in a Distributed Environment. Monographs in Computer Science, chap. 8, Springer, 1999, pp. 147–175.
8. Chakravarthy S. (ed.), Special Issue on Active Databases, vol. 15, IEEE Quarterly Bulletin on Data Engineering, 1992.
9. Chakravarthy S. and Widom J. (eds.), Special Issue on the Active Database Systems, Journal of Intelligent Information Systems 7, 1996.
10. Dayal U., Blaustein B., Buchmann A., et al. S.C. HiPAC: A Research Project in Active, Time-Constrained Database Management. Tech. Rep. CCA-88-02, Xerox Advanced Information Technology, Cambridge, 1988.
11. Dittrich K.R., Kotz A.M., and Mulle J.A. An Event/Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases. ACM SIGMOD Rec., 15(3):22–36, 1986.
12. Geppert A. and Berndtsson M. (eds.). Proc. 3rd International Workshop on Rules in Database Systems, LNCS, vol. 1312, Springer, 1997.
13. Morgenstern M. Active Databases as a Paradigm for Enhanced Computing Environments. In Proc. 9th Int. Conf. on Very Data Bases, 1983, pp. 34–42.
14. Paton N.W. (ed.) Active Rules in Database Systems. Monographs in Computer Science, Springer, 1999.
15. Paton N.W. and Diaz O. Active Database Systems. ACM Comput. Surv, 31(1):63–103, 1999.
16. Paton N.W. and Williams M.W. (eds.). In Proc. 1st International Workshop on Rules in Database Systems, Springer, Berlin, 1994.
17. Sellis T. (ed.). In Proc. 2nd International Workshop on Rules in Database Systems, vol. 905, Springer, 1995.
18. Stonebraker M., Hearst M., and Potamianos S. Commentary on the POSTGRES Rules System. SIGMOD Rec., 18(3):5–11, 1989.
19. Widom J. and Ceri S. (eds.) Active Database Systems: Triggers and Rules For Advanced Database Processing. Morgan Kaufmann, 1996.
20. Widom J. and Chakravarthy S. (eds.). In Proc. 4th International Workshop on Research Issues in Data Engineering – Active Database Systems, 1994.

Active Database Management System Architecture

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Synonyms

ADBMS infrastructure; ADBMS framework; ADBMS

Definition

The active database management system (ADBMS) architecture is the software organization of a DBMS with active capabilities. That is, the architecture defines support for active capabilities expressed in terms of services, significant components providing the services as well as critical interaction among these services.

Historical Background

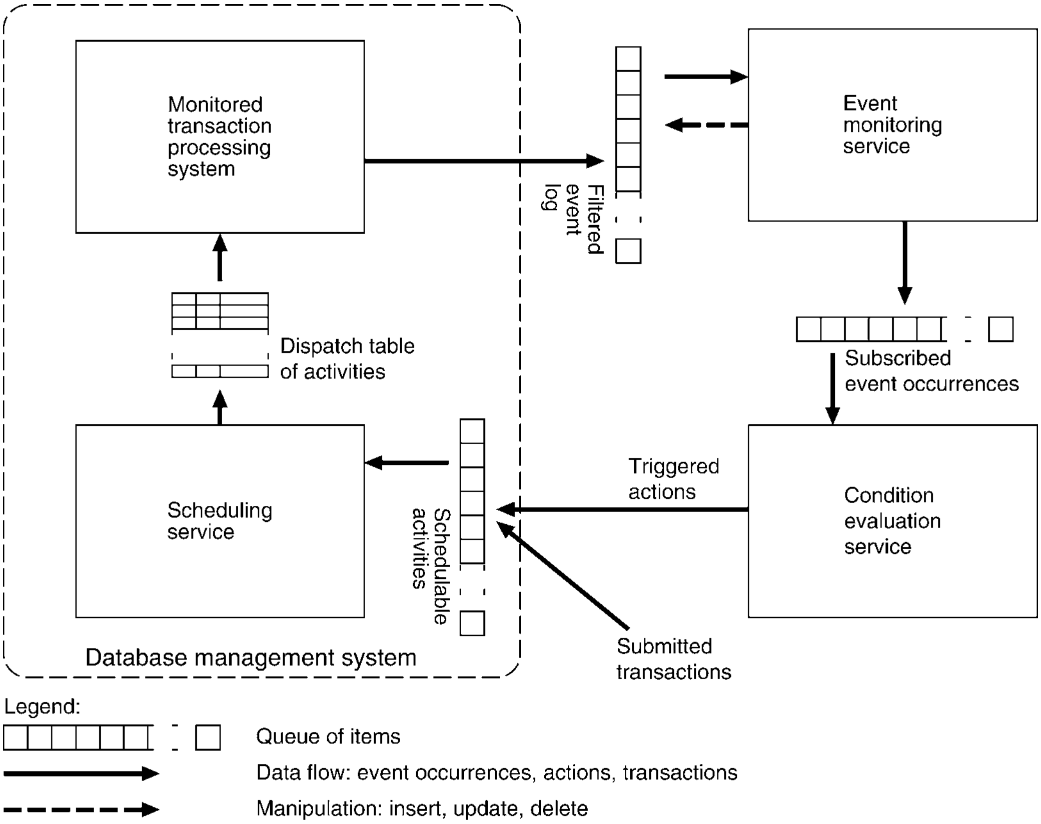
Several architectures has been proposed: HiPAC [5,8], REACH [4], ODE [14], SAMOS [10], SMILE [15], and DeeDS [1]. Each of these architectures emphasize particular issues concerning the actual DBMS that they are based on as well as the type of support for active capabilities. Paton and Diaz [18] provide an excellent survey on this topic. Essentially, these architectures propose that the *active capabilities* of an ADBMS require the services specified in Table 1. It is assumed that queries to the database are encompassed in transactions and hence transactions imply queries as well as database manipulation operations such as insertion, updates and deletion of tuples.

The services in Table 1 interact as depicted in Fig. 1. Briefly, transactions are submitted to the scheduling service that updates the dispatch table read by the transaction processing service. When these transactions are processed by the transaction processing service events are generated. These events are signaled to the event monitoring service that analyzes them. Events that are associated with rules (subscribed events) are signaled to the rule evaluation service that evaluates the conditions of *triggered rules* (i.e., rules associated with signaled events). The actions of the rules whose conditions are true are submitted for scheduling and are executed as dictated by the scheduling policy. These actions execute as part of some transaction according to the coupling mode and can, in turn, generate events. This is a general description of the service interaction and it can be optimized by refining it for a specific purpose, for example, in immediate coupling mode no queues between the services are actually needed.

In more detail, transactions are submitted to the scheduling service via a queue of schedulable activities; this queue of schedulable activities is processed and a dispatch table of schedulable activities is updated. This scheduling service encompasses scheduling of transactions as well as ECA rule actions in addition to other necessary schedulable activities. It is desirable for the scheduling service to encompass all these types of schedulable activities, because they impact each other, since they compete for the same resources. The next step in the processing chain is the monitored transaction processing service, which includes the transaction management, lock management, and log management [11, Chap. 5], as well as a database query engine (cf. query processor [20, Chap. 1]), but not the scheduling service. Another way to view the transaction processing service is as a passive database management system without the transaction scheduling service. The transaction processing service is denoted "monitored," since it generates events that are handled by the active capabilities. The monitored transaction processing service executes transactions and ECA rule actions according to the dispatch table. When transactions execute, event occurrences are signaled to the event monitoring service via a filtered event log. When event monitoring executes, it updates the filtered event log and submits subscribed events to the rule evaluation service. An example of event log filtering is that if a composite event occurrence is detected, then for optimization reasons (cf. dynamic programming) this event occurrence is stored in the filtered event log. Another example is that when events are no longer needed, then they are pruned; for example, when a transaction is aborted, then all event occurrences can be pruned unless intertransaction events are allowed (implying that dirty reads may occur). The rule evaluation service reads the

Active Database Management System Architecture. Table 1. Services in active database management systems

Service	Responsibility
Event monitoring	The event monitoring service is responsible for collecting events, analyzing events and disseminating results of the analysis (in terms of events) to subscribers, in particular, ECA rules.
Rule evaluation	The rule evaluation service is responsible for invoking condition evaluation of triggered ECA rules and submit actions for execution to the scheduler.
Scheduling service	The scheduling service is responsible for readying and ordering schedulable activities such as ECA rule actions, transactions etc. for execution.


Monitored transaction processing system
Filtered event log
Event monitoring service
Dispatch table of activities
Subscribed event occurrences
Scheduling service
Schedulable activities
Triggered actions
Condition evaluation service
Submitted transactions
Database management system
Legend:
Queue of items
Data flow: event occurrences, actions, transactions
Manipulation: insert, update, delete

Active Database Management System Architecture. Figure 1. Service interaction view of architecture (based on architecture by Paton and Diaz [18]).

queue of subscribed events, finds the triggered rules and evaluates their conditions. These conditions may be queries, logical expressions or arbitrary code depending on the active database system [9]. The rule evaluation results in a set of actions that is submitted to the scheduling service for execution.

The general view of active capabilities (in Fig. 1) can be refined and implemented in different ways. As mentioned, it is possible to optimize an implementation by removing the queues between the services if only immediate coupling mode is considered; this result in less overhead, but restricts the expressibility of ECA-rules significantly. A service can be implemented via one or more servers. These servers can be replicated to different physical nodes for performance or dependability reasons (e.g., availability, reliability).

In active databases, a set of issues have a major impact on refinement and implementation of the general service-oriented view depicted in Fig. 1. These issues are: (i) coupling modes; (ii) interaction with typical database management services such as transaction management, lock management, recovery management (both pre-crash such as logging and checkpointing and post-crash such as the actually recovery) (cf., for example, transaction processing by Gray and Reuter [11, Chap. 4]); (iii) when and how to invoke services; and (iv) active capabilities in distributed active databases.

The coupling modes control how rule evaluation is invoked in response to events and how the ECA rule actions are submitted, scheduled, dispatched and executed for rules whose conditions are true (see entry “Coupling modes” for more detailed description). There are different alternatives to interaction with a database system. One alternative is to place active database services on top of existing database management systems. However, this is problematic if the database management system is not extended with active capabilities [4]. For example, the deferred coupling mode require that when a transaction is requested to commit, then queued actions should be evaluated. This requires that the transaction management to interact with the rule evaluation and scheduling services during commit processing (e.g., by using back hooks in the database management system). Further, to be useful, the detached coupling mode has a set of significant varieties [4] that require the possibility to express constraints between transactions.

The nested transaction model [16] is a sound basis for active capabilities. For example, deferred actions can be executed as subtransactions that can be committed or aborted independently of the parent transaction. Nested transactions still require that existing services are modified. Alternatively rule evaluation can be performed as subtransactions.

To achieve implicit events the database schema translation process needs to automatically instrument the monitored systems. An inferior solution is to extend an existing schema with instrumented entities, for example, each class in an object-oriented database can be inherited to an instrumented class. In this example, there is no way to enforce that the instrumented classes are actually used. The problem is to modify the database schema translation process, since this is typically an intrinsic part in commercial DBMSs.

Concerning issue (iii), the services must be allocated to existing resources and scheduled together with the transactions. Typically, the services are implemented as a set of server processes and transactions are performed by transaction programs running as processes (cf., [11]). These processes are typically scheduled, dispatched and executed as a response to the requests from outside the database management system or as a direct or indirect response to a timeout. Each service is either invoked when something is stored in the queue or table or explicitly invoked, for example, when the system clock is updated to reflect the new time. The issues concerning scheduling are paramount in any database management system for real-time system purposes [2].

Event monitoring can either be (i) implicitly invoked whenever an event occurs, or it can be (ii) explicitly invoked. This is similar to coupling modes, but it is between the event sources (e.g., transaction processing service and application) and the event monitoring service rather than in between the services of the active capabilities. Case (i) is prevalent in most active database research, but it has a negative impact in terms of determinism of the result of event monitoring. For example, the problem addressed in the event specification and event detection entry concerning the unintuitive semantics of the disjunctive event operator is a result of implicit invocation. In distributed and real-time systems, explicit invocation is preferable in case (ii), since it provides the operating system with the control when something should be evaluated. Explicit invocation solves the problem of disjunction operator (see event specification and event detection entries), since the event expressions defining composite event types can be explicitly evaluated when all events have been delivered to event monitoring rather than implicitly evaluated whenever an event is delivered.

In explicit invocation of event monitoring, the different event contexts can be treated in different ways. For example, in recent event context, only the most recent result is of interest in implicit invocation. However, in terms of explicit invocation, all possible most recent event occurrences may be of interest, not only the last one. For example, it may be desirable to keep the most recent event occurrence per time slot rather than per terminating event.

Issue (iv) has been addressed in, for example, DeeDS [1], COBEA [15], Hermes [19], X2TS [5]. Further, it has been addressed in event based systems for mobile networks by Mühl et al. [17]. Essentially, it is necessary to perform event detection in a moving time window, where the end of the time window is the current time. All events that are older than the beginning of the time window can be removed and ignored. Further, the heterogeneity must be addressed and there are XML-based solutions (e.g., Common Base Events [6]).

Another issue that is significant in distributed active databases is the time and order of events. For example, in Snoop [8] it is suggested to separate global and local event detection, because of the difference in the time granularity of the local view of time and the global (distributed) view of time.

Foundations

For a particular application domain, common significant requirements and properties as well as pre-requisites of available resources need to be considered to refine the general architecture. Depending on the requirements, properties and pre-requisites, different compromises are reached. One example is the use of composite event detection in active real-time databases. In REACH [4], composite event detection is disallowed for real-time transactions. The reason for this is that during composite event detection, contributing events are locked and this locking affects other transaction in a harmful way with respect to meeting deadlines. A different approach is proposed in DeeDS [1], where events are stored in the database and cached in a special filtered event log; during event composition, events are not locked thus enabling the use of composite event detection for transaction with critical deadlines. The cost is that isolation of transactions can be violated unless it is handled by the active capabilities.

Availability is an example of a property that significantly affects the software architecture. For example, availability is often considered significant in distributed systems; that is, even though physical nodes may fail, communications links may be down, or the other physical nodes may be overloaded, one should get, at least, some defined level of service from the system. An example of availability requirements is that emergency calls in phone switches should be prioritized over non-emergency calls, a fact that entails that existing phone call connections can be disconnected to let an emergency call through. Another example to improve availability is pursued in DeeDS [1], where eventual consistency is investigated as a mean to improve availability of data. The cost is that data can temporarily be inconsistent.

As addressed in the aforementioned examples, different settings affect the architecture. Essentially, there are two approaches that can be mixed: (i) refine or invent new method, tools, techniques to solve a problem, and these method, tools, techniques can stem from different but relevant research areas; (ii) refine the requirements or pre-requisites to solve the problem (e.g., weaken the ACID properties of transactions).

Key Applications

The architecture of ADBMSs is of special interest to developers of database management systems and their applications. In particular, software engineering issues are of major interest. Researchers performing experiments can make use of this architecture to enable valid experiments, study effects of optimizations etc.

Concerning real examples of applications, only simple things such as using rules for implementing alerters, for example, when an integrity constraint is violated. SQL Triggers implement simple ECA rules in immediate coupling mode between event monitoring and rule evaluation as well as between rule evaluation and action execution.

Researchers have aimed for various application domains such as:

- Stock market
- Inventory control
- Bank applications

Essentially, any application domain in which there is an interest to move functionality from the applications to the database schema to reduce the interdependence between applications and databases.

Future Directions

There are no silver bullets in computer science or software engineering and each refinement of the architecture (in Fig. 1) is a compromise providing or enabling certain features and properties. For example, by allowing only detached coupling mode it is easier to achieve timeliness, an important property of real-time systems; however, the trade-off is that it is difficult to specify integrity rules in terms of ECA-rules, since the integrity checks are performed in a different transaction. The consequence is that dirty transactions as well as compensating transactions that perform recovery from violated integrity rules must be allowed.

It is desirable to study architectures addressing how to meet specific requirement of the application area (e.g., accounting information in mobile ad-hoc networks), the specific environment in which the active database are used (e.g., distributed systems, real-time systems, mobile ad-hoc networks, limited resource equipment). The major criteria for a successful architecture (e.g., by refining an existing architecture) is if anyone can gain something from using it. For example, Borr [3] reported that by refining their architecture by employing transaction processing they improved productivity, reliability as well as average throughput in their heterogenous distributed reliable applications.

An area that has received little attention in active database is optimization of processing. For example, how can queries to the database be optimized with condition evaluation if conditions are expressed as arbitrary queries? Another question is how to group actions to optimize performance? So far, the emphasis has been on expressibility as well as techniques how to enable active support in different settings. Another area that has received little attention is recovery processing, both pre-crash and post-crash recovery. For example, how should recovery with respect to detached but dependent transactions be managed?

Intertransaction events and rules has been proposed by, for example, Buchmann et al. [4]. How should this be managed with respect to the isolation levels proposed by Gray and Reuter [11, Chap. 7]?

There are several other areas with which active database technology can be combined. Historical examples include real-time databases, temporal databases, main-memory databases, geographical information systems. One area that has received little attention is how enable reuse of database schemas.

Cross-references

- ▶ Active Database Coupling Modes
- ▶ Active Database Execution Model
- ▶ Active Database Knowledge Model
- ▶ Event Detection
- ▶ Event Specification

Recommended Reading

1. Andler S., Hansson J., Eriksson J., Mellin J., Berndtsson M., and Eftring B. DeeDS Towards a Distributed Active and Real-Time Database System. ACM SIGMOD Rec., 25(1), 1996.
2. Berndtsson M. and Hansson J. Issues in active real-time databases. In Proc. 1st Int. Workshop on Active and Real-Time Database System, 1995, pp. 142–150.
3. Borr A.J. Robustness to crash in a distributed database: A non shared-memory multi-processor approach. In Proc. 10th Int. Conf. on Very Large Data Bases, 1984, pp. 445–453.
4. Buchmann A.P., Zimmermann J., Blakeley J.A., and Wells D.L. Building an Integrated Active OODBMS: Requirements, Architecture, and Design Decisions. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 117–128.
5. Chakravarthy S., Blaustein B., Buchmann A.P., Carey M., Dayal U., Goldhirsch D., Hsu M., Jauhuri R., Ladin R., Livny M., McCarthy D., McKee R., and Rosenthal A. HiPAC: A Research Project In Active Time-Constrained Database Management. Tech. Rep. XAIT-89-02, Xerox Advanced Information Technology, 1989.
6. Common Base Events. Http://www.ibm.com/developerworks/library/specification/ws-cbe/.
7. Chakravarthy S., Krishnaprasad V., Anwar E., and Kim S.K. Composite Events for Active Database: Semantics, Contexts, and Detection. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 606–617.
8. Dayal U., Blaustein B., Buchmann A., Chakravarthy S., Hsu M., Ladin R., McCarty D., Rosenthal A., Sarin S., Carey M.J., Livny M., and Jauharu R. The HiPAC Project: Combining active databases and timing constraints. ACM Sigmod Rec., 17(1), 1988.
9. Eriksson J. Real-Time and Active Databases: A Survey. In Proc. 2nd Int. Workshop on Active, Real-Time, and Temporal Database Systems, 1997, pp. 1–23.
10. Gatziu S. Events in an Active Object-Oriented Database System. Ph.D. thesis, University of Zurich, Switzerland, 1994.
11. Gray J. and Reuter A. Transaction processing: Concepts and techniques. Morgan Kaufmann, Los Altos, CA, 1994.
12. Jaeger U. Event Detection in Active Databases. Ph.D. thesis, University of Berlin, 1997.
13. Liebig C.M. and Malva A.B. Integrating Notifications and Transactions: Concepts and X2TS Prototype. In Second International Workshop on Engineering Distributed Objects, 2000, pp. 194–214.
14. Lieuwen D.F., Gehani N., and Arlein R. The ODE active database: Trigger semantics and implementation. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 412–420.
15. Ma C. and Bacon J. COBEA: A CORBA-based Event Architecture. In Proc. 4th USENIX Conf. Object-Oriented Technologies and Syst., 1998, pp. 117–132.
16. Moss J.E.B. Nested transactions: An approach to reliable distributed computing. MIT, 1985.
17. Mühl G., Fiege L., and Pietzuch P.R. Distributed event-based systems. Springer, Berlin, 2006.
18. Paton N. and Diaz O. Active database systems. ACM Comput. Surv., 31(1):63–103, 1999.
19. Pietzuch P. and Bacon J. Hermes: A Distributed Event-Based Middleware Architecture. In Proc. 22nd Int. Conf. on Distributed Computing Systems Workshop. Vienna, Austria, 2002, pp. 611–618.
20. Ullman J.D. Principles of Database Systems. Computer Science, 1982.

Active Database Coupling Modes

Mikael Berndtsson, Jonas Mellin
University of Skövde, Skövde, Sweden

Definition

Coupling modes specify execution points for ECA rule conditions and ECA rule actions with respect to the triggering event and the transaction model.

Historical Background

Coupling modes for ECA rules were first suggested in the HiPAC project [2,3].

Foundations

Coupling modes are specified for event-condition couplings and for condition-action couplings. In detail, the event-condition coupling specifies when the condition should be evaluated with respect to the triggering event, and the condition-action coupling specifies when the rule action should be executed with respect to the evaluated rule condition (if condition is evaluated to true).

The three most common coupling modes are: immediate, deferred, and decoupled. The immediate coupling mode preempts the execution of the transaction and immediately initiates condition evaluation and action execution. In the deferred coupling mode, condition evaluation and action execution is deferred to the end of the transaction (before transaction commit). Finally, in decoupled (also referred to as detached) coupling mode, condition evaluation and action execution is performed in separate transactions.

Specifying event-condition couplings and condition-action couplings in total isolation from each other is not a good idea. What first might seem to be one valid coupling mode for event-condition and one valid coupling mode for condition-action, can be an invalid coupling mode when used together. Thus, when combining event-condition couplings and condition-action couplings, not all combinations of coupling modes are valid. The HiPAC project [2,3] proposed seven valid coupling modes, see Table 1.

- *Immediate, immediate*: the rule condition is evaluated immediately after the event, and the rule action is executed immediately after the rule condition.
- *Immediate, deferred*: the rule condition is evaluated immediately after the event, and the execution of the rule action is deferred to the end of the transaction.
- *Immediate, decoupled*: the rule condition is evaluated immediately after the event, and the rule action is decoupled in a totally separate and parallel transaction.
- *Deferred, deferred*: both the evaluation of the rule condition and the execution of the rule action is deferred to the end of the transaction.
- *Deferred, decoupled*: the evaluation of the rule condition is deferred to the end of the transaction, and the rule action is decoupled in a totally separate and parallel transaction.
- *Decoupled, immediate*: the rule condition is decoupled in a totally separate and parallel transaction, and the rule action is executed (in the same parallel transaction) immediately after the rule condition.
- *Decoupled, decoupled*: the rule condition is decoupled in a totally separate and parallel transaction, and the rule action is decoupled in another totally separate and parallel transaction.

The two invalid coupling modes are:

- *Deferred, immediate*: this combination violates the semantics of ECA rules. That is, rule conditions must be evaluated before rule actions are executed. One cannot preempt the execution of the transaction immediately after the event and execute the rule action and at the same time postpone the condition evaluation to the end of the transaction.
- *Decoupled, deferred*: this combination violates transaction boundaries. That is, one cannot decouple the condition evaluation in a separate and parallel transaction and at the same time postpone the

Active Database Coupling Modes. Table 1. Coupling modes

	Condition-Action		
Event-Condition	*Immediate*	*Deferred*	*Decoupled*
Immediate	condition evaluated and action executed after event	condition evaluated after event, action executed at end of transaction	condition evaluated after event, action executed in a separate transaction
Deferred	not valid	condition evaluated and action executed at end of transaction	condition evaluated at end of transaction, action executed in a separate transaction
Decoupled	in a separate transaction: condition evaluated and action executed after event	not valid	condition evaluated in one separate transaction, action executed in another separate transaction

execution of the rule action to the end of the original transaction, since one cannot know when the condition evaluation will take place. Thus, there is a risk that the action execution in the original transaction will run before the condition has been evaluated in the parallel transaction.

Rule actions executed in decoupled transactions can either be dependent upon or independent of the transaction in which the event took place.

The research project REACH (REal-time ACtive Heterogeneous System) [1] introduced two additional coupling modes for supporting side effects of rule actions that are irreversible. The new coupling modes are variants of the detached casually dependent coupling mode: sequential casually dependent, and exclusive casually dependent. In sequential casually dependent, a rule is executed in a separate transaction. However, the rule execution can only begin once the triggering transaction has committed. In exclusive casually dependent, a rule is executed in a detached parallel transaction and it can commit only if the triggering transaction failed.

Cross-references

- ▶ Active Database Execution Model
- ▶ ECA-rules

Recommended Reading

1. Branding H., Buchmann A., Kudrass T., and Zimmermann J. Rules in an Open System: The REACH Rule System. In Proc. 1st International Workshop on Rules in Database Systems, Workshops in Computing, 1994, pp. 111–126.
2. Dayal U., Blaustein B.A., Buchmann S.C., et al. The HiPAC project: Combining active databases and timing constraints. ACM SIGMOD Rec., 17(1):51–70, 1988.
3. Dayal U., Blaustein B., Buchmann A., Chakravarthy S., and et al. HiPAC: A Research Project in Active, Time-Constrained Database Management. Tech. Rep. CCA-88-02, Xerox Advanced Information Technology, Cambridge, 1988.

Active Database Execution Model

MIKAEL BERNDTSSON, JONAS MELLIN
University of Skövde, Skövde, Sweden

Definition

The execution model of an active database describes how a set of ECA rules behave at run time.

Key Points

The execution model describes how a set of ECA rules (i.e., active database rulebase) behave at run time [2,4]. Any execution model of an active database must have support for: (i) detecting event occurrences, (ii) evaluating conditions, and (iii) executing actions.

If an active database supports composite event detection, it needs a policy that describes how a composite event is computed. A typical approach is to use the event consumption modes as described in Snoop [1]: recent, chronicle, continuous, and cumulative. In the recent event context, only the most recent constituent events will be used to form composite events. In the chronicle event context, events are consumed in chronicle order. The earliest unused initiator/terminator pair are used to form the composite event. In the continuous event context, each initiator starts the detection of a new composite event and a terminator may terminate one or more composite event occurrences. The difference between continuous and chronicle event contexts is that in the continuous event context, one terminator can detect more than one occurrence of the composite event. In the cumulative event context, all events contributing to a composite event are accumulated until the composite event is detected. When the composite event is detected, all contributing events are consumed. Another approach to these event consumption modes is to specify a finer semantics for each event by using logical events as suggested in [3].

Once an event has been detected, there are several execution policies related to rule conditions and rule actions that must be in place in the execution model. Thus an execution model for an active database should provide answers to the following questions [2,4,5]:

- When should the condition be evaluated and when should the action should be executed with respect to the triggering event and the transaction model? This is usually specified by coupling modes.
- What happens if an event triggers several rules?
  - Are all rules evaluated, a subset, or only one rule?
  - Are rules executed in parallell, according to rule priority, or non-deterministically?
- What happens if one's rules trigger another set of rules?
  - What happens if the rule action of one rule negates the rule condition of an already triggered rule?
  - Can cycles appear? For example, can a rule trigger itself?

The answers to the above questions are important to know, as they dictate how a ECA rule system will behave at run time. If the answers to the above questions are not known, then the behavior of the ECA rule application becomes unpredictable.

Cross-references

- ▶ Active Database Coupling Modes
- ▶ Active Database Rulebase
- ▶ Composite Event
- ▶ Database Trigger
- ▶ ECA Rules

Recommended Reading

1. Chakravarthy S., Krishnaprasad V., Anwar E., and Kim S.K. Composite Events for Active Databases: Semantics Contexts and Detection. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 606–617.
2. Dayal U., Blaustein B., Buchmann A., and Chakravarthy S. et al. HiPAC: A Research Project in Active, Time-Constrained Database Management. Technical Report CCA-88-02, Xerox Advanced Information Technology, Cambridge, 1988.
3. Gehani N., Jagadish H.V., and Smueli O. Event specification in an active object-oriented database. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1992, pp. 81–90.
4. Paton N.W. and Diaz O. Active Database Systems. ACM Comput. Surv, 31(1):63–103, 1999.
5. Widom J. and Finkelstein S. Set-Oriented Production Rules in Relational Database Systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 259–270.

Active Database Knowledge Model

Mikael Berndtsson, Jonas Mellin
University of Skövde, Skövde, Sweden

Definition

The knowledge model of an active database describes what can be said about the ECA rules, that is what type of events are supported, what type of conditions are supported, and what type of actions are supported?

Key Points

The knowledge model describes what types of events, conditions, and actions that are supported in an active database. Another way to look at the knowledge model is to imagine what type of features are available in an ECA rule definition language.

A framework of dimensions for the knowledge model is presented in [3]. Briefly, each part of an ECA rule is associated with dimensions that describe supported features. Thus, an event can be described as either a primitive event or a composite event, how it was generated (source), whether the event is generated for all instances in a given set or only for a subset (event granularity), what type (if event is a composite event) of operators and event consumption modes are used in the detection of the composite event.

Conditions are evaluated against a database state. There are three different database states that a rule condition can be associated with [3]: (i) the database state at the start of the transaction, (ii) the database state when the event was detected, and (iii) the database state when the condition is evaluated.

There are four different database states that a rule action can be associated with [3]: (i) the database state at the start of the transaction, (ii) the database state when the event was detected, and (iii) the database state when the condition is evaluated, and (iv) the database state just before action execution. The type of rule actions range from internal database updates (e.g., update a table) to external programs (e.g., send email).

Within the context of the knowledge model it is also useful to consider how ECA rules are represented, for example inside classes, as data members, or first class objects. Representing ECA rules as first class objects [1,2] is a popular choice, since rules can be treated as any other object in the database and traditional database operations can be used to manipulate the ECA rules. Thus, representing ECA rules as first class objects implies that ECA rules are not dependent upon the existence of other objects.

The knowledge model of an active database should also describe whether the active database supports passing of parameters between the ECA rule parts, for example passing of parameters from the event part to the condition part.

Related to the knowledge model is the execution model that describes how ECA rules behave at run time.

Cross-references

- ▶ Active Database Execution Model
- ▶ ECA Rules

Recommended Reading

1. Dayal U., Blaustein B., Buchmann A. et al. S.C. HiPAC: A Research Project in Active, Time-Constrained Database Management. Tech. Rep. CCA-88-02, Xerox Advanced Information Technology, Cambridge, 1988.

2. Dayal U., Buchmann A., and McCarthy D. Rules are objects too: a knowledge model for an active, object-oriented database system. In Proc. 2nd Int. Workshop on Object-Oriented Database Systems, 1988, pp. 129–143.
3. Paton N.W. and Diaz O. Active database systems. ACM Comput. Surv., 31(1):63–103, 1999.

Active Database Rulebase

AnnMarie Ericsson[1], Mikael Berndtsson[2], Jonas Mellin[3]
University of Skövde, Skövde, Sweden

Definition

An active database rulebase is a set of ECA rules that can be manipulated by an active database.

Key Points

An active database rulebase is a set of ECA rules that can be manipulated by an active database. Thus, an ADB rulebase is not static, but it evolves over time. Typically, ECA rules can be added, deleted, modified, enabled, and disabled. Each update of the ADB rulebase can potentially lead to different behaviors of the ECA rules at run time, in particular with respect to termination and confluence.

Termination concerns whether a set of rules is guaranteed to terminate. A set of rules may have a non-terminating behavior if rules are triggering each other in a circular order, for example, if the execution of rule R1 triggers rule R2 and the execution of rule R2 triggers rule R1. A set of rules is confluent if the outcome of simultaneously triggered rules is unique and independent of execution order.

Cross-references

► ECA Rules

Active Databases

► Event Driven Architecture

Active Disks

► Active Storage

Active Document

► Active XML

Active Storage

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Synonyms

Active Disks; Intelligent Disks

Definition

Active Storage is a computer system architecture which utilizes processing power in disk drives to execute application code. Active Storage was introduced in separate academic papers [1–3] in 1998. The term Active Storage is sometimes identified merely with the computer systems proposed in these papers. Two synonyms, Active Disk and Intelligent Disk, are also used to refer to Active Storage. The basic idea behind Active Storage is to offload computation and data traffic from host computers to the disk drives themselves such that the system can achieve significant performance improvements for data intensive applications such as decision support systems and multimedia applications.

Key Points

A research group at Carnegie Mellon University proposed, in [3], a storage device called Active Disk, which has the capability of downloading application-level code and running it on a processor embedded on the device. Active Disk has a performance advantage for I/O bound scans, since processor-per-disk processing can potentially reduce data traffic on interconnects to host computers and yield great parallelism of scans. E. Riedel et al. carefully studied the potential benefits of using Active Disks for four types of data intensive applications, and introduced analytical performance models for comparing traditional server systems and Active Disks. They also prototyped ten Active Disks, each having a DEC Alpha processor and two Seagate disk drives, and demonstrated almost linear scalability in the experiments.

A research group at University of California at Berkeley discussed a vision of Intelligent Disks (IDISKs) in [2]. The approach of Intelligent Disk is similar to that of Active Disk. K. Keeton et al. carefully studied the weaknesses of shared-nothing clusters of workstations and then explored the possibility of replacing the cluster nodes with Intelligent Disks for large-scale decision support applications. Intelligent Disks assumed higher complexity of applications and hardware resources in comparison with CMU's Active Disks.

Another Active Disk was presented by a research group at the University of California at Santa Barbara and University of Maryland in [1]. A. Acharya et al. carefully studied programming models to exploit disk-embedded processors efficiently and safely and proposed algorithms for typical data intensive operations such as selection and external sorting, which were validated by simulation experiments.

These three works are often recognized as opening the gate for new researches of Intelligent Storage Systems in the post-"database machines" era.

Cross-references

► Database Machine
► Intelligent Storage Systems

Recommended Reading

1. Acharya A., Mustafa U., and Saltz J.H. Active disks: programming model, algorithms and evaluation. In Proc. 8th Int. Conf. Architectural Support for Programming Lang. and Operating Syst., 1998, pp. 81–91.
2. Keeton K., Patterson D.A., and Hellerstein J.M. A case for intelligent disks (IDISKs). SIGMOD Rec., 27(3):42–52, 1998.
3. Riedel E., Gibson G.A., and Faloutsos C. Active storage for large-scale data mining and multimedia. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 62–73.

Active XML

Serge Abiteboul[1], Omar Benjelloun[2],
Tova Milo[3]
[1]INRIA, Saclay Île-de-France, Orsay, Cedex, France
[2]Google Inc., Mountain view, CA, USA
[3]Tel Aviv University, Tel Aviv, Israel

Synonyms

Active document; AXML

Definition

Active XML documents (AXML documents, for short) are XML documents [12] that may include embedded calls to Web services [13]. Hence, AXML documents are a combination of regular "extensional" XML data with data that is defined "intensionally," i.e., as a description that enables obtaining data dynamically (by calling the corresponding service).

AXML documents evolve in time when calls to their embedded services are triggered. The calls may bring data once (when invoked) or continually (e.g., if the called service is a continuous one, such as a subscription to an RSS feed). They may even update existing parts of the document (e.g., by refreshing previously fetched data).

Historical Background

The AXML language was originally proposed at INRIA around 2002. Work around AXML has been going there in the following years. A survey of the research on AXML is given in [13]. The software, primarily under the form of an AXML system, is available as open source software. Resources on Active XML may be found on the project's Web site [11].

The notion of embedding function calls into data is old. Embedded functions are already present in relational systems as stored procedures. Of course, method calls form a key component of object databases. For the Web, scripting languages such as PHP or JSP have made popular the integration of processing inside HTML or XML documents. Combined with standard database interfaces such as JDBC and ODBC, functions are used to integrate results of (SQL) queries. This idea can also be found in commercial software products, for instance, in Microsoft Office XP, SmartTags inside Office documents can be linked to Microsoft's .NET platform for Web services.

The originality of the AXML approach is that it proposed to *exchange* such documents, building on the fact that Web services may be invoked from anywhere. In that sense, this is truly a language for distributed data management. Another particularity is that the logic (the AXML language) is a subset of the AXML algebra.

Looking at the services in AXML as queries, the approach can be viewed as closely related to recent works based on XQuery [14] where the query language is used to describe query plans. For instance the DXQ project [7] developed at ATT and UCSD emphasizes the distributed evaluation of XQuery queries. Since one can describe documents in an XQuery syntax, such approaches encompass in some sense AXML documents where the service calls are XQuery queries.

The connection with deductive databases is used in [1] to study the diagnosis problems in distributed networks. A similar approach is followed in [8] for declarative network routing.

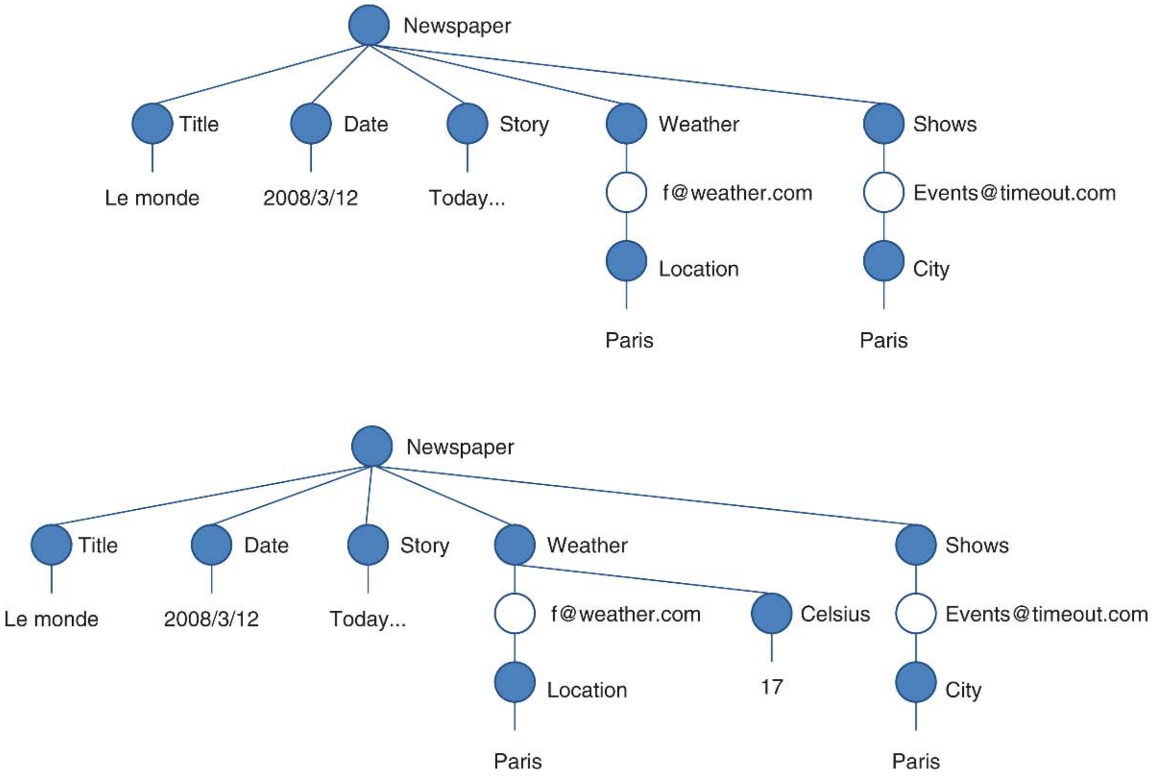
It should be observed that the AXML approach touches upon most database areas. In particular, the presence of intensional data leads to views, deductive databases and data integration. The activation of calls contained in a document essentially leads to active databases. AXML services may be activated by external servers, which relates to subscription queries and stream databases. Finally, the evolution of AXML documents and their inherent changing nature lead to an approach of workflows and service choreography in the style of business artifacts [10].

The management of AXML document raises a number of issues. For instance, the evaluation of queries over active documents is studied in [2]. The "casting" of a document to a desired type is studied in [9]. The distribution of documents between several peers and their replication is the topic of [4].

Foundations

An AXML document is a (syntactically valid) XML document, where service calls are denoted by special XML elements labeled *call*. An example AXML document is given in Fig. 1. The figure shows first the XML serialized syntax, then a more abstract view of the same document as a labeled tree. The document in the figure describes a (simplified) newspaper homepage consisting of (i) some extensional information (the name of the newspaper, the current date, and a news story), and (ii) some intensional information (service calls for the weather forecast, and for the current exhibits). When the services are called, the tree evolves. For example, the tree at the bottom is what results from a call to the service *f* at *weather.com* to obtain the temperature in Paris.

AXML documents fit nicely in a peer-to-peer architecture, where each peer is a persistent store of AXML



Active XML. Figure 1. An AXML document.

documents, and may act both as a client, by invoking the service calls embedded in its AXML documents, and as a server, by providing Web services over these documents.

Two fundamental issues arise when dealing with AXML documents. The first one is related to the exchange of AXML documents between peers, and the second one is related to query evaluation over such data.

Documents Exchange: When exchanged between two applications/peers, AXML documents have a crucial property: since Web services can be called from anywhere on the Web, data can either be materialized before sending, or sent in its intensional form and left to the receiver to materialize if and when needed. Just like *XML Schemas* do for standard XML, *AXML schemas* let the user specify the desired format of the exchanged data, including which parts should remain intensional and which should be materialized. Novel algorithms allow the sender to determine (statically or dynamically) which service invocations are required to "cast" the document to the required data exchange format [9].

Query evaluation: Answering a query on an AXML document may require triggering some of the service calls it contains. These services may, in turn, query other AXML documents and trigger some other services, and so on. This recursion, based on the management of intensional data, leads to a framework in the style of *deductive databases*. Query evaluation on AXML data can therefore benefit from techniques developed in deductive databases such as Magic Sets [6]. Indeed, corresponding AXML query optimization techniques where proposed in [1,2].

Efficient query processing is, in general, a critical issue for Web data management. AXML, when properly extended, becomes an algebraic language that enables query processors installed on different peers to collaborate by exchanging streams of (A)XML data [14]. The crux of the approach is (i) the introduction of generic services (i.e., services that can be provided by several peers, such as query processing) and (ii) some explicit control of distribution (e.g., to allow delegating part of some work to another peer).

Key Applications

AXML and the AXML algebra target all distributed applications that involve the management of distributed data. AXML is particularly suited for data integration (from databases and other data resources exported as Web services) and for managing (active) views on top of data sources. In particular, AXML can serve as a formal foundation for mash-up systems. Also, the language is useful for (business) applications based on evolving documents in the style of business artifacts, and on the exchange of such information. The fact that the exchange is based on flows of XML messages makes it also well-adapted to the management of distributed streams of information.

Cross-references

- ▶ Active Document
- ▶ BPEL
- ▶ Web Services
- ▶ W3C XML Query Language
- ▶ XML
- ▶ XML Types

Recommended Reading

1. Abiteboul S., Abrams Z., and Milo T. Diagnosis of Asynchronous Discrete Event Systems – Datalog to the Rescue! In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 358–367.
2. Abiteboul S., Benjelloun O., Cautis B., Manolescu I., Milo T., and Preda N. Lazy Query Evaluation for Active XML. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 227–238.
3. Abiteboul S., Benjelloun O., and Milo T. The Active XML project, an overview, VLDB J, 17(5):1019–1040, 2008.
4. Abiteboul S., Bonifati A., Cobena G., Manolescu I., and Milo T. Dynamic XML Documents with Distribution and Replication. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 527–538.
5. Abiteboul S., Manolescu I., and Taropa E. A Framework for Distributed XML Data Management. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006.
6. Bancilhon F., Maier D., Sagiv Y., and Ullman J.D. Magic Sets and Other Strange Ways to Implement Logic Programs. In Proc. 5th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1986, pp. 1–15.
7. DXQ: Managing Distributed System Resources with Distributed XQuery. http://db.ucsd.edu/dxq/.
8. Loo B.T., Condie T., Garofalakis M., Gay D.E, Hellerstein J.M., Maniatis P., Ramakrishnan R., Roscoe T., and Stoica I. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 97–108.
9. Milo T., Abiteboul S., Amann B., Benjelloun O., and Ngoc F.D. Exchanging Intensional XML data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 289–300.
10. Nigam A. and Caswell N.S. Business artifacts: an approach to operational specification. IBM Syst. J., 42(3):428–445, 2003.
11. The Active XML homepage. http://www.activexml.net/.

12. The Extensible Markup Language (XML) 1.0 (2nd edn). http://www.w3.org/TR/REC-xml.
13. The W3C Web Services Activity. http://www.w3.org/2002/ws.
14. The XQuery language. http://www.w3.org/TR/xquery.

Activity

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

Step; Node; Task; Work element

Definition

A description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. A workflow activity requires human and/or machine resources to support process execution; where human resource is required an activity is allocated to a workflow participant.

Key Points

A process definition generally consists of many process activities which are logically related in terms of their contribution to the overall realization of the business process.

An activity is typically the smallest unit of work which is scheduled by a workflow engine during process enactment (e.g., using transition and pre/post-conditions), although one activity may result in several work items being assigned (to a workflow participant).

Wholly manual activities may form part of a business process and be included within its associated process definition, but do not form part of the automated workflow resulting from the computer supported execution of the process.

An activity may therefore be categorized as "manual," or "automated." Within this document, which is written principally in the context of workflow management, the term is normally used to refer to an automated activity.

Cross-references

- ▶ Activity Diagrams
- ▶ Actors/Agents/Roles
- ▶ Workflow Model

Activity Diagrams

Luciano Baresi
Politecnico di Milano University, Milan, Italy

Synonyms

Control flow diagrams; Object flow diagrams; Flowcharts; Data flow diagrams

Definition

Activity diagrams, also known as control flow and object flow diagrams, are one of the UML (unified modeling language [11]) behavioral diagrams. They provide a graphical notation to define the sequential, conditional, and parallel composition of lower-level behaviors. These diagrams are suitable for business process modeling and can easily be used to capture the logic of a single use case, the usage of a scenario, or the detailed logic of a business rule. They model the workflow behavior of an entity (system) in a way similar to *state diagrams* where the different activities are seen as the states of doing something. Although they could also model the internal logic of a complex operation, this is not their primary use since tangled operations should always be decomposed into simpler ones [1,2].

An activity [3] represents a behavior that is composed of individual elements called *actions*. Actions have incoming and outgoing edges that specify control and data flow from and to other nodes. Activities may form invocation hierarchies invoking other activities, ultimately resolving to individual actions.

The execution of an activity implies that each contained action be executed zero, one, or more times depending on the execution conditions and the structure of the activity. The execution of an action is initiated by the termination of other actions, the availability of particular objects and data, or the occurrence of external events. The execution is based on token flow (like Petri Nets). A token contains an object, datum, or locus of control, and is present in the activity diagram at a particular node. When an action begins execution, tokens are accepted from some or all of its input edges and a token is placed on the node. When an action completes execution, a token is removed from the node and tokens are moved to some or all of its output edges.

Historical Background

OMG (Object Management Group, [10]) proposed and standardized *activity diagrams* by borrowing concepts from flow-based notations and some formal methods. As for the first class, these diagrams mimic *flowcharts* [6] in their idea of step-by-step representation of algorithms and processes, but they also resemble data and control flow diagrams [4]. The former provide a hierarchical and graphical representation of the "flow" of data through a system inspired by the idea of *data flow graph*. They show the flow of data from external entities into the system, how these data are moved from one computation to another, and how they are logically stored. Similarly, *object flow diagrams* show the relationships among input objects, methods, and output objects in object-based models. Control flow diagrams represent the paths that can be traversed while executing a program. Each node in the graph represents a basic block, be it a single line or an entire function, and edges render how the execution jumps among them.

Moving to the second group, activity diagrams are similar to state diagrams [8], where the evolution of a system is rendered by the identification of the states, which characterize the element's life cycle, and of the transitions between them. A state transition can be constrained by the occurrence of an event and by an additional condition; its firing can cause the execution of an associated action. Mealy et al. propose different variations: Mealy assumes that actions be only associated with transitions, Moore only considers actions associated with states, and Harel's *state charts* [7] merge the two approaches with actions on both states and transitions, and enhance their flat model with nested and concurrent states.

The dynamic semantics of activity diagrams is clearly inspired by Petri Nets [9], which are a simple graphical formalism to specify the behavior of concurrent and parallel systems. The nodes are partitioned into places and transitions, with arcs that can only connect nodes of different type. Places may contain any number of tokens and a distribution of tokens over the places of a net is called a marking. A transition can only fire when there is at least a token in all its input places (i.e., those places connected to the transition by means of incoming edges), and its firing removes a token for all these places and produces a new one in each output place (i.e., a place connected to the transition through an outgoing edge). P/T nets only consider tokens as placeholders, while colored nets augment them with typed data and thus with firing conditions that become more articulated and can predicate on the tokens' values in the input places.

Activity diagrams also borrow from SDL (Specification and Description Language, [5]) as event handling. This is a specification language for the unambiguous description of the behavior of reactive and distributed systems. Originally, the notation was conceived for the specification of telecommunication systems, but currently its application is wider and includes process control and real-time applications in general. A system is specified as a set of interconnected abstract machines, which are extensions of finite state machines. SDL offers both a graphical and a textual representation and its last version (known as SDL-2000) is completely object-orientated.

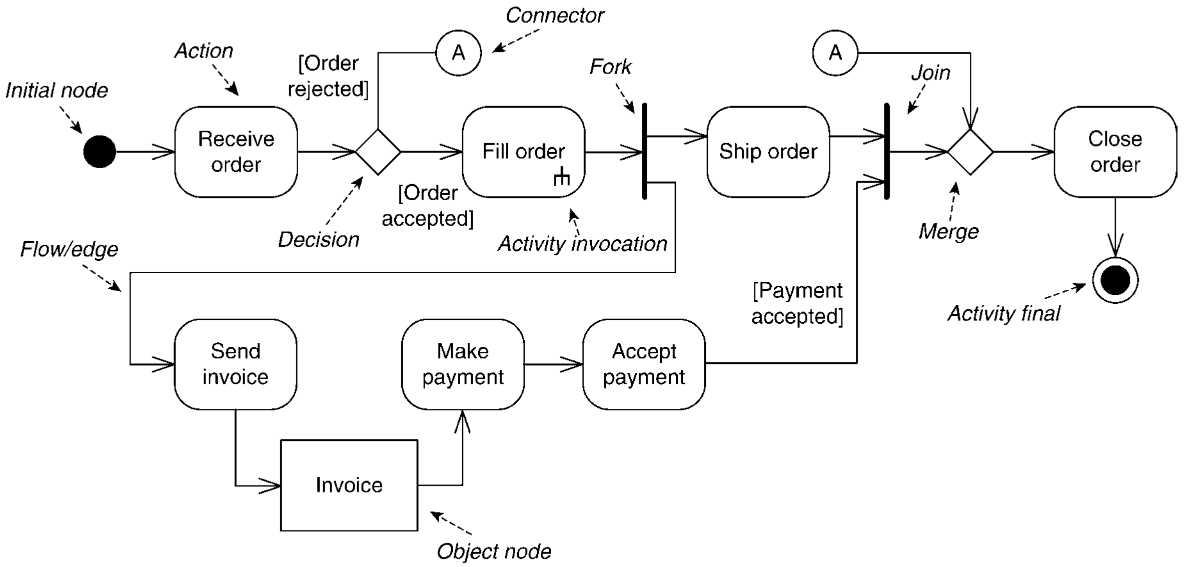
Foundations

Figure 1 addresses the well-known problem of order management and proposes a first *activity diagram* whose aim is twofold: it presents a possible formalization of the process, and it also introduces many of the concepts supplied by these diagrams.

Each atomic step is called *action*, with an *initial node* and *activity final* nodes to delimit their ordering as sequences, parallel threads, or conditional flows. A *fork* splits a single execution thread into a set of parallel ones, while a *join*, along with an optional *join specification* to constrain the unification, is used to re-synchronize the different threads into a single execution. Similarly, a *decision* creates alternative paths, and a *merge* re-unifies them. To avoid misunderstandings, each path must be decorated with the condition, in brackets, that must be verified to make the execution take that path.

The diagram of Fig. 1 also exemplifies the use of *connectors* to render *flows/edges* that might tangle the representation. This is nothing but an example, but the solution is interesting to avoid drawing flows that cross other elements or move all around the diagram. Another key feature is the use of a rake to indicate that action `Fill Order` is actually an *activity invocation*, and hides a hierarchical decomposition of actions into activities.

Besides the control flow, activity diagrams can also show the data/object flow among the actions. The use of *object nodes* allows users to state the artifacts exchanged between two actions, even if they are not directly connected by an edge. In many cases



Activity Diagrams. Figure 1. Example activity diagram.

control and object flows coincide, but this is not mandatory.

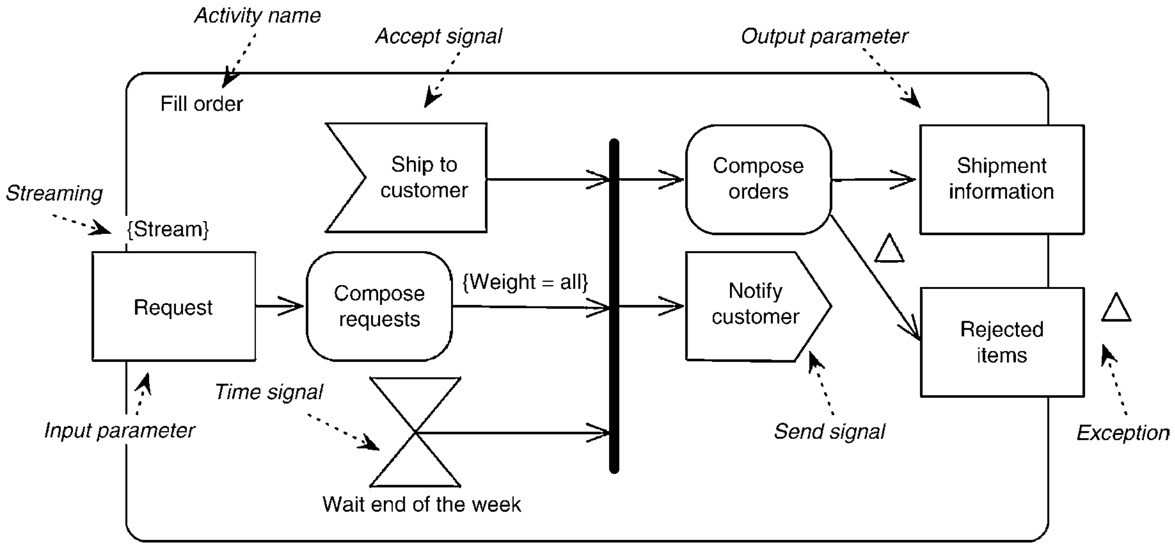
Activities can also comprise *input* and *output parameters* to render the idea that the activity's execution initiates when the inputs are available, and produces some outputs. For example, activity `Fill Order` of Fig. 2, which can be seen as a refinement of the invocation in Fig. 1, requires that at least one `Request` be present, but then it considers the parameter as a *stream*, and produces `Shipment Information` and `Rejected Items`. While the first outcome is the "normal" one, the second object is produced only in case of *exceptions* (rendered with a small triangle on both the object and the flow that produces it). In a stream, the flow is annotated from action `Compose Requests` to the join with its *weight* to mean that the subsequent processing must consider *all* the requests received when the composition starts.

The execution can also consider signals as enablers or outcomes of special-purpose actions. For example, Fig. 2 shows the use of an *accept signal*, to force that the composition of orders (`Compose Orders`) must be initiated by an external event, a *time signal*, to make the execution wait for a given timeframe (be it absolute or relative), and a *send signal*, to produce a notification to the customer as soon as the action starts.

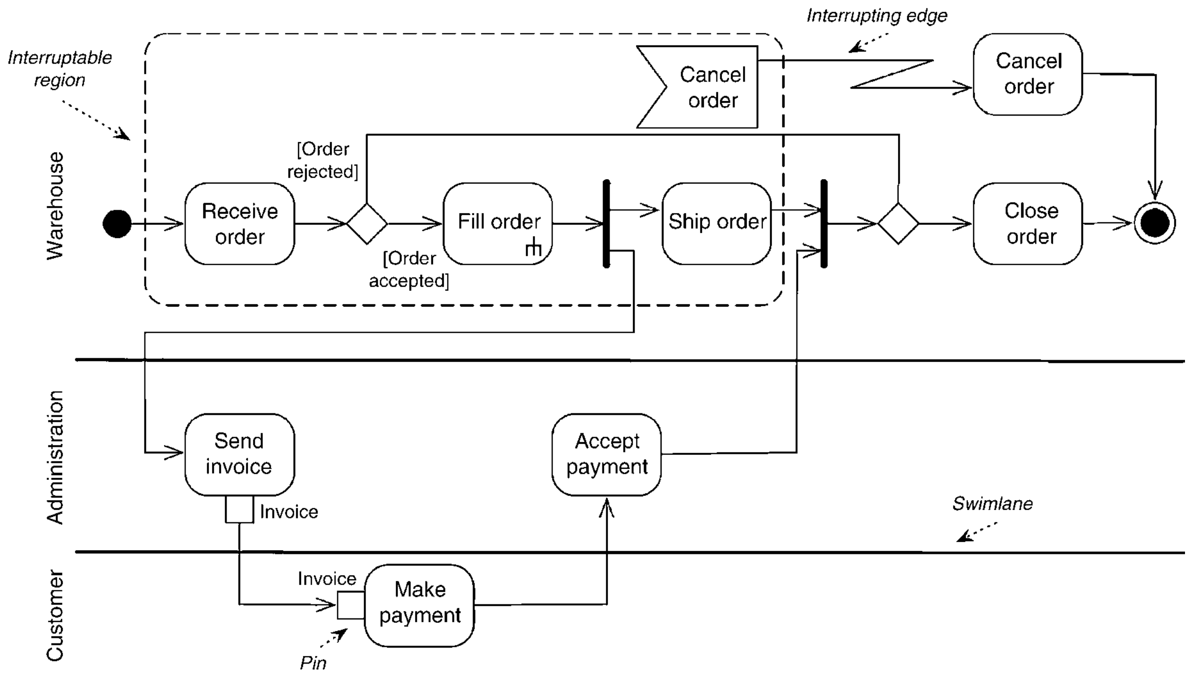
Basic diagrams can also be enriched with *swimlanes* to partition the different actions with respect to their responsibilities. Figure 3 shows a simple example: The primitive actions are the same as those of Fig. 1, but now they are associated with the three players in charge of activate the behaviors in the activity. The standard also supports hierarchical and multi-dimensional partitioning, that is, hierarchies of responsible actors or matrix-based partitions.

The `Warehouse` can also receive `Cancel Order` notifications to asynchronously interrupt the execution as soon as the external event arrivers. This is obtained by declaring an *interruptable region*, which contains the accept signal node and generates the interrupt that stops the computation in that region and moves the execution directly to action `Cancel Order` by means of an *interrupting edge*. More generally, this is a way to enrich diagrams with specialized *exception handlers* similarly to many modern programming and workflow languages. The figure also introduces *pins* as a compact way to render the objects exchanged between actions: empty boxes correspond to discrete elements, while filled ones refer to streams.

The discussion thus far considers the case in which the outcome of an action triggers a single execution of another action, but in some cases conditions may exist in which the "token" is structured and a single result triggers multiple executions of the same action. For example, if the example of Fig. 1 were slightly modified and after receiving an order, the user wants to check the items in it, a single execution of action



Activity Diagrams. Figure 2. Activity diagrams with signals.

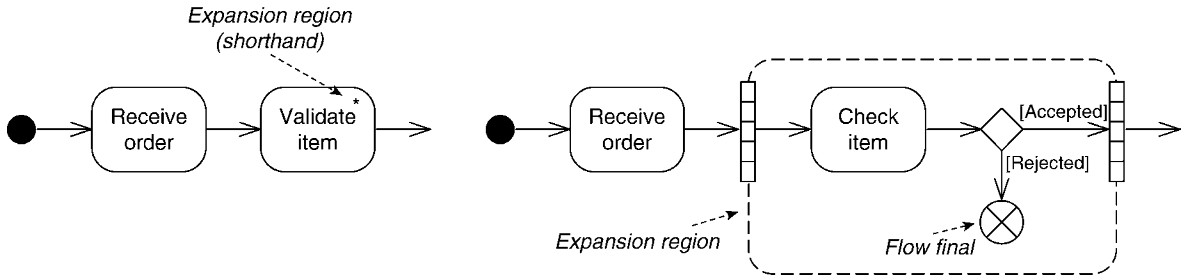


Activity Diagrams. Figure 3. Example swimlanes.

`Receive Order` would trigger multiple executions of `Validate Item`. This situation is depicted in the left-hand side of Fig. 4, where the star * conceives the information described so far.

The same problem can be addressed in a more complete way (right-hand side of figure) with an *expansion region*. The two arrays are supposed to store the input and output elements. In some cases, the number of input and output tokens is the same, but it might also be the case that the behavior in the region filters the incoming elements.

In the left-hand side of Fig. 4, it is assumed that some items are accepted and fill the output array, while others are rejected and thus their execution flow ends



Activity Diagrams. Figure 4. Expansion region.

there. This situation requires that a *flow final* be used to state that only the flow is ended and not the whole activity. Flow final nodes are a means to interrupt particular flows in this kind of regions, but also in loops or other similar cases.

The execution leaves an expansion region as soon as all the output tokens are available, that is, as soon as all the executions of the behavior embedded in the region are over. Notice that these executions can be carried out both concurrently (by annotating the rectangle with stereotype *concurrent*) or iteratively (with stereotype *iterative*). The next action considers the whole set of tokens as a single entity.

Further details about exceptions and other advanced elements, like pre- and post-conditions associated with single actions or whole activities, central buffers, and data stores are not discussed here, but the reader is referred to [11] for a thorough presentation.

Key Applications

Activity diagrams are usually employed to describe complex behaviors. This means that they are useful to model tangled processes, describe the actions that need to take place and when they should occur in use cases, render complicated algorithms, and model applications with parallel and alternative flows. Nowadays, these necessities belong to ICT specialists, like software engineering, requirements experts, and information systems architects, but also to experts in other fields (e.g., business analysts or production engineers) that need this kind of graphical notations to describe their solutions.

Activity diagrams can be used in isolation, when the user needs a pure control (data) flow notation, but they can also be adopted in conjunction with other modeling techniques such as interaction diagrams, state diagrams, or other UML diagrams. However, activity diagrams should not take the place of other diagrams. For example, even if the border between activity and state diagrams is sometimes blurred, activity diagrams provide a procedural decomposition of the problem under analysis, while state diagrams mostly concentrate on how studied elements behave. Moreover, activity diagrams do not give details about how objects behave or how they collaborate.

Cross-references

▶ Unified Modeling Language
▶ Web Services
▶ Workflow modeling

Recommended Reading

1. Arlow J. and Neustadt I. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design, 3rd edn. Addison-Wesley, Reading, MA, 2005.
2. Booch G., Rumbaugh J., and Jacobson I. The Unified Modeling Language User Guide, 2nd edn. Addison-Wesley, Reading, MA, 2005.
3. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd edn. Addison-Wesley, Reading, MA, 2003.
4. Gane C. and Sarson T. Structured System Analysis. Prentice-Hall, Englewood Cliffs, NJ, 1979.
5. Gaudin E., Najm E., and Reed R. In Proceedings of SDL 2007: Design for Dependable Systems, 13th International SDL Forum, LNCS, vol. 4745, Springer, 2007.
6. Goldstine H. The Computer from Pascal to Von Neumann. Princeton University Press, Princeton, NJ, 1972, pp. 266–267.
7. Harel D. and Naamad A. The STATEMATE Semantics of Statecharts. ACM Trans. Softw. Eng. Methodol., 5(4):293–333, 1996.
8. Hopcroft J. and Ullman J. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, MA, 2002.
9. Murata T. Petri Nets: Properties, Analysis, and Applications. Proc. IEEE, 77(4):541–580, 1989.
10. Object Management Group, http://www.omg.org/
11. OMG, Unified Modeling Language, http://www.uml.org/

Actors/Agents/Roles

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

Workflow participant; Player; End user; Work performer

Definition

A resource that performs the work represented by a workflow activity instance.

This work is normally manifested as one or more work items assigned to the workflow participant via the worklist.

Key Points

These terms are normally applied to a human resource but it could conceptually include machine-based resources such as an intelligent agent.

Where an activity requires no human resource and is handled automatically by a computer application, the normal terminology for the machine-based resource is Invoked Application.

An Actor, Agent or Role may be identified directly within the business process definition, or (more normally) is identified by reference within the process definition to a role, which can then be filled by one or more of the resources available to the workflow system to operate in that role during process enactment.

Cross-references

▶ Activity
▶ Workflow Model

Ad hoc Retrieval models

▶ Information Retrieval Models

Adaptation

▶ Mediation

Adaptive Database Replication

▶ Autonomous Replication

Adaptive Interfaces

Maristella Matera
Politecnico di Milano University, Milan, Italy

Synonyms

Context-aware interfaces; Personalized interfaces

Definition

A specific class of user interfaces that are able to change in some way in response to different characteristics of the user, of the usage environment or of the task the user is supposed to accomplish. The aim is to improve the user's experience, by providing both interaction mechanisms and contents that best suit the specific situation of use.

Key Points

There are a number of ways in which interface adaptivity can be exploited to support user interaction. The interaction dimensions that are adapted vary among functionality (e.g., error correction or active help), presentation (user presentation of input to the system, system presentation of information to the user), and user tasks (e.g., task simplification based on the user's capabilities). Adaptivity along such dimensions is achieved by capturing and representing into some models a number of characteristics: the user's characteristics (preferences, experience, etc.); the tasks that the user accomplishes through the system; the characteristics of the information with which the user must be provided.

Due to current advances in communication and network technologies, adaptivity is now gaining momentum. Different types of mobile devices indeed offer support to access – at any time, from anywhere, and with any media – services and contents customized to the users' preferences and usage environments. In this new context, *content personalization*, based on user profile, has demonstrated its benefits for both users and content providers and has been commonly recognized as fundamental factor for augmenting the overall effectiveness of applications. Going one step further, the new challenge in adaptive interfaces is now *context-awareness*. It can be interpreted as a natural evolution of personalization, addressing not only the user's identity and preferences, but also the environment that hosts users, applications, and their interaction, i.e., the *context*. Context-awareness, hence, aims at enhancing the application usefulness by taking into account a wide range of properties of the context of use.

Cross-references

▶ Visual Interaction

Adaptive Message-Oriented Middleware

▶ Adaptive Middleware for Message Queuing Systems

Adaptive Metric Techniques

▶ Learning Distance Measures

Adaptive Middleware for Message Queuing Systems

Christophe Taton[1], Noel De Palma[1], Sara Bouchenak[2]
[1]INPG - INRIA, Grenoble, France
[2]University of Grenoble I - INRIA, Grenoble, France

Synonyms

Autonomous message queuing systems; Adaptive message-oriented middleware; Autonomous message-oriented middleware

Definition

Distributed database systems are usually built on top of middleware solutions, such as message queuing systems. Adaptive message queuing systems are able to improve the performance of such a middleware through load balancing and queue provisioning.

Historical Background

The use of message oriented middlewares (MOMs) in the context of the Internet has evidenced a need for highly scalable and highly available MOM. A very promising approach to the above issue is to implement performance management as an autonomic software. The main advantages of this approach are: (i) Providing a high-level support for deploying and configuring applications reduces errors and administrator's efforts. (ii) Autonomic management allows the required reconfigurations to be performed without human intervention, thus improving the system reactivity and saving administrator's time. (iii) Autonomic management is a means to save hardware resources, as resources can be allocated only when required (dynamically upon failure or load peak) instead of pre-allocated.

Several parameters may impact the performance of MOMs. Self-optimization makes use of these parameters to improve the performance of the MOM. The proposed self-optimization approach is based on a queue clustering solution: a *clustered queue* is a set of queues each running on different servers and sharing clients. Self-optimization takes place in two parts: (i) the optimization of the clustered queue load-balancing and (ii) the dynamic provisioning of a queue in the clustered queue. The first part allows the overall improvement of the clustered queue performance while the second part optimizes the resource usage inside the clustered queue. Thus the idea is to create an autonomic system that fairly distributes client connections among the queues belonging to the clustered queue and dynamically adds and removes queues in the clustered queue depending on the load. This would allow to use the adequate number of queues at any time.

Foundations

Clustered Queues

A queue is a staging area that contains messages which have been sent by message producers and are waiting to be read by message consumers. A message is removed from the queue once it has been read. For scalability purpose, a queue can be replicated forming a clustered queue. The clustered queue feature provides a load balancing mechanism. A clustered queue is a cluster of queues (a given number of queue destinations knowing each other) that are able to exchange messages depending on their load. Each queue of a cluster periodically reevaluates its load factor and sends the result to the other queues of the cluster. When a queue hosts more messages than it is authorized to do, and according to the load factors of the cluster, it distributes the extra messages to the other queues. When a queue is requested to deliver messages but is empty, it requests messages from the other queues of the cluster. This mechanism guarantees that no queue is hyper-active while some others are lazy, and tends to distribute the work load among the servers involved in the cluster.

Clustered Queue Performance

Clustered queues are standard queues that share a common pool of message producers and consumers, and that can exchange message to balance the load. All the queues of a clustered queue are supposed to be directly connected to each other. This allows message exchanges between the queues of a cluster in order to empty flooded queues and to fill draining queues.

The clustered queue Q_c is connected to N_c message producers and to M_c message consumers. Q_c is composed of standard queues $Q_i(i \in [1..k])$. Each queue Q_i is in charge of a subset of N_i message producers and of a subset of M_i message consumers:

$$\begin{cases} N_c = \sum_i N_i \\ M_c = \sum_i M_i \end{cases}$$

The distribution of the clients between the queues Q_i is described as follows: x_i (resp. y_i) is the fraction of message producers (resp. consumers) that are directed to Q_i.

$$\begin{cases} N_i = x_i \cdot N_c \\ M_i = y_i \cdot M_c \end{cases}, \begin{cases} \sum_i x_i = 1 \\ \sum_i y_i = 1 \end{cases}$$

The standard queue Q_i to which a consumer or producer is directed to cannot be changed after the client connection to the clustered queue. This way, the only action that may affect the client distribution among the queues is the selection of an adequate queue when the client connection is opened.

The clustered queue Q_c is characterized by its aggregate message production rate p_c and its aggregate message consumption rate c_c. The clustered queue Q_c also has a virtual clustered queue length l_c that aggregates the length of all contained standard queues:

$$l_c = \sum_i l_i = p_c - c_c, \begin{cases} p_c = \sum_i p_i \\ c_c = \sum_i c_i \end{cases}$$

The clustered queue length l_c obeys to the same law as a standard queue:

1. Q_c is globally stable when $\Delta l_c = 0$. This configuration ensures that the clustered queue is globally stable. However Q_c may observe local unstabilities if one of its queues is draining or is flooded.
2. If $\Delta l_c > 0$, the clustered queue will grow and eventually saturate; then message producers will have to wait.
3. If $\Delta l_c < 0$, the clustered queue will shrink until it is empty; then message consumers will also have to wait.

Now, considering that the clustered queue is globally stable, several scenarios that illustrate the impact of client distribution on performance are given below.

Optimal client distribution of the clustered queue Q_c is achieved when clients are fairly distributed among the k queues Q_i. Assuming that all queues and hosts have equivalent processing capabilities and that all producers (*resp.* consumers) have equivalent message production (*resp.* consumption) rates (and that all produced messages are equivalent: message cost is uniformly distributed), this means that:

$$\begin{cases} x_i = 1/k \\ y_i = 1/k \end{cases}, \begin{cases} N_i = \frac{N_c}{k}, \\ M_i = \frac{M_c}{k} \end{cases}$$

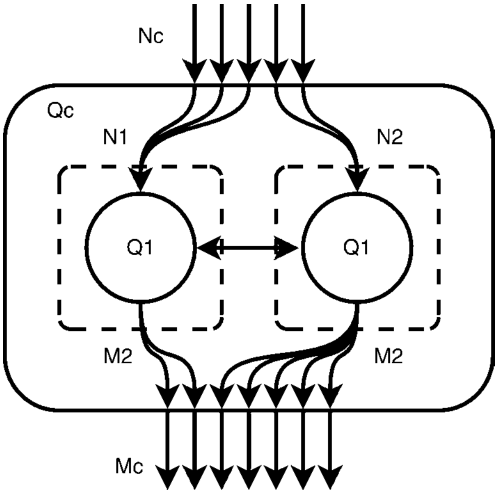
In these conditions, all queues Q_i are stable and the queue cluster is balanced. As a consequence, there are no internal queue-to-queue message exchanges, and performance is optimal. Queue clustering then provides a quasi-linear speedup.

The worst clients distribution appears when one queue only has message producers or only has message consumers. In the example depicted in Fig. 1, this is realized when:

$$\begin{cases} x_1 = 1 \\ y_1 = 0 \end{cases}, \begin{cases} x_2 = 0 \\ y_2 = 1 \end{cases}, \begin{cases} N_1 = N_c \\ M_1 = 0 \end{cases}, \begin{cases} N_2 = 0 \\ M_2 = M_c \end{cases}$$

Indeed, this configuration implies that the whole message production is directed to queue Q_1. Q_1 then forwards all messages to Q_2 that in turn delivers messages to the message consumers.

Local instability is observed when some queues Q_i of Q_c are unbalanced. This is characterized by a



Adaptive Middleware for Message Queuing Systems. Figure 1. Clustered queue Q_c.

mismatch between the fraction of producers and the fraction of consumers directed to Q_i:

$$x_i \neq y_i$$

In the example showed in Fig. 1, Q_c is composed of two standard queues Q_1 and Q_2. A scenario of local instability can be envisioned with the following clients distribution:

$$\begin{cases} x_1 = 2/3 \\ y_1 = 1/3 \end{cases}, \begin{cases} x_2 = 1/3 \\ y_2 = 2/3 \end{cases}$$

This distribution implies that Q_1 is flooding and will have to enqueue messages, while Q_2 is draining and will see its consumer clients wait. However the queue cluster Q_c ensures the global stability of the system thanks to internal message exchanges from Q_1 to Q_2.

A stable and unfair distribution can be observed when the clustered queue is globally and locally stable, but the load is unfairly balanced within the queues. This happens when the client distribution is non-uniform.

In the example presented in Fig. 1, this can be realized by directing more clients to Q_1 than Q_2:

$$\begin{cases} x_1 = 2/3 \\ y_1 = 2/3 \end{cases}, \begin{cases} x_2 = 1/3 \\ y_2 = 1/3 \end{cases}$$

In this scenario, queue Q_1 processes two third of the load, while queue Q_2 only processes one third. Suc situation can lead to bad performance since Q_1 may saturates while Q_2 is lazy.

It is worthwhile to indicate that these scenarios may all happen since clients join and leave the system in an uncontrolled way. Indeed, the global stability of a (clustered) queue is under responsability of the application developper. For instance, the queue can be flooded for a period; it is assumed that it will get inverted and draining after, thus providing global stability over time.

Provisioning

The previous scenario of stable and non-optimal distribution raises the question of the capacity of a queue. The capacity C_i of standard queue Q_i is expressed as an optimal number of clients. The queue load L_i is then expressed as the ratio between its current number of clients and its capacity:

$$L_i = \frac{N_i + M_i}{C_i}$$

1. $L_i < 1$: queue Q_i is underloaded and thus lazy; the message throughput delivered by the queue can be improved and resources are wasted.
2. $L_i > 1$: queue Q_i is overloaded and may saturate; this induces a decreased message throughput and eventually leads to thrashing.
3. $L_i = 1$: queue Q_i is fairly loaded and delivers its optimal message throughput.

These parameters and indicators are transposed to queue clusters. The clustered queue Q_c is characterized by its aggregated capacity C_c and its global load L_c:

$$C_c = \sum_i C_i, \; L_c = \frac{N_c + M_c}{C_c} = \frac{\sum_i L_i \cdot C_i}{\sum_i C_i}$$

The load of a clustered queue obeys to the same law as the load of a standard queue.

However a clustered queue allows to control k, the number of inside standard queues, and thus to control its aggregated capacity $C_c = \sum_{i=1}^{k} C_i$. This control is indeed operated with a re-evaluation of the clustered queue provisioning.

1. When $L_c < 1$, the clustered queue is underloaded: if the clients distribution is optimal, then all the standard queues inside the cluster will be underloaded. However, as the client distribution may be non-optimal, some of the single queues may be overloaded, even if the cluster is globally lazy. If the load is too low, then some queues may be removed from the cluster.

2. When $L_c > 1$, the clustered queue is overloaded: even if the distribution of clients over the queues is optimal, there will exist at least one standard queue that will be overloaded. One way to handle this case is to re-provision the clustered queue by inserting one or more queues into the cluster.

Control Rules for a Self-Optimizing Clustered Queue

The global clients distribution D of the clustered queue Q_c is captured by the fractions of message producers x_i and consumers y_i. The optimal clients distribution D_{opt} is realized when all queues are stable ($\forall i\ x_i = y_i$) and when the load is fairly balanced over all queues ($\forall i,\ j x_i = x_j,\ y_i = y_j$). This implies that the optimal distribution is reached when $x_i = y_i = 1/k$.

$$D = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_k & y_k \end{bmatrix},\ D_{opt} = \begin{bmatrix} 1/k & 1/k \\ \vdots & \vdots \\ 1/k & 1/k \end{bmatrix}$$

Local instabilities are characterized by a mismatch between the fraction of message producers x_i and consumers y_i on a standard queue. The purpose of this rule is the stability of all standard queues so as to minimize internal queue-to-queue message transfer.

1. $[(R_1)]$ $x_i > y_i$: Q_i is flooding with more message production than consumption and should then seek more consumers and/or fewer producers.
2. $[(R_2)]$ $x_i < y_i$: Q_i is draining with more message consumption than production and should then seek more producers and/or fewer consumers.

Load balancing rules control the load applied to a single standard queue. The goal is then to enforce a fair load balancing over all queues.

1. $[(R_3)]$ $L_i > 1$: Q_i is overloaded and should avoid accepting new clients as it may degrade its performance.
2. $[(R_4)]$ $L_i < 1$: Q_i is underloaded and should request more clients so as to optimize resource usage.

Global provisioning rules control the load applied to the whole clustered queue. These rules target the optimal size of the clustered queue while the load applied to the system evolves.

1. $[(R_5)]$ $L_c > 1$: the queue cluster is overloaded and requires an increased capacity to handle all its clients in an optimal way.
2. $[(R_6)]$ $L_c < 1$: the queue cluster is underloaded and could accept a decrease in capacity.

Key Applications

Adaptive middleware for message queuing systems helps building autonomous distributed systems to improve their performance while minimizing their resource usage, such as distributed Internet services and distributed information systems.

Cross-references

▶ Distributed Database Systems
▶ Distributed DBMS
▶ Message Queuing Systems

Recommended Reading

1. Aron M., Druschel P., and Zwaenepoel W. Cluster reserves: a mechanism for resource management in cluster-based network servers. In Proc. 2000 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Comp. Syst., 2000, pp. 90–101.
2. Menth M. and Henjes R. Analysis of the message waiting time for the fioranoMQ JMS server. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2006, pp. 1.
3. Shen K., Tang H., Yang T., and Chu L. Integrated resource management for cluster-based internet services. In Proc. 5th USENIX Symp. on Operating System Design and Implementation, 2002.
4. Urgaonkar B. and Shenoy P. Sharc: Managing CPU and network bandwidth in shared clusters. IEEE Trans. Parall. Distrib. Syst., 15(1):2–17, 2004.
5. Zhu H., Ti H., and Yang Y. Demand-driven service differentiation in cluster-based network servers. In Proc. 20th Annual Joint Conf. of the IEEE Computer and Communications Societies, vol. 2, 2001, pp. 679–688.

Adaptive Query Optimization

▶ Adaptive Query Processing

Adaptive Query Processing

EVAGGELIA PITOURA
University of Ioannina, Ioannina, Greece

Synonyms

Adaptive query optimization; Eddies; Autonomic query processing

Definition

While in traditional query processing, a query is first optimized and then executed, *adaptive query processing* techniques use runtime feedback to modify query processing in a way that provides better response time, more efficient CPU utilization or more useful incremental results. Adaptive query processing makes query processing more robust to optimizer mistakes, unknown statistics, and dynamically changing data, runtime and workload characteristics. The spectrum of adaptive query processing techniques is quite broad: they may span the executions of multiple queries or adapt within the execution of a single query; they may affect the query plan being executed or just the scheduling of operations within the plan.

Key Points

Conventional query processing follows an optimize-then-execute strategy: after generating alternative query plans, the query optimizer selects the most cost-efficient among them and passes it to the execution engine that directly executes it, typically with little or no runtime decision-making. As queries become more complex, this strategy faces many limitations such as missing statistics, unexpected correlations, and dynamically changing data, runtime, and workload characteristics. These problems are aggregated in the case of long-running queries over data streams as well as in the case of queries over multiple potentially heterogeneous data sources across wide-area networks. Adaptive query processing tries to address these shortcomings by using feedback during query execution to tune query processing. The goal is to increase throughput, improve response time or provide more useful incremental results.

To implement adaptivity, regular query execution is supplemented with a control system for monitoring and analyzing at run-time various parameters that affect query execution. Based on this analysis, certain decisions are made about how the system behavior should be changed. Clearly, this may introduce considerable overheads.

The complete space of adaptive query processing techniques is quite broad and varied. Adaptability may be applied to query execution of multiple queries or just a single one. It may also affect the whole query plan being executed or just the scheduling of operations within the plan. Adaptability techniques also differ on how much they interleave plan generation and execution. Some techniques interleave planning and execution just a few times, by just having the plan re-optimized at specific points, whereas other techniques interleave planning and execution to the point where they are not even clearly distinguishable.

A number of fundamental adaptability techniques include:

- *Horizontal partitioning*, where different plans are used on different portions of the data. Partitioning may be explicit or implicit in the functioning of the operator.
- *Query execution by tuple routing*, where query execution is treated as the process of routing tuples through operators and adaptability is achieved by changing the order in which tuples are routed.
- *Plan partitioning*, where execution progresses in stages, by interleaving optimization and execution steps at a number of well-defined points during query execution.
- *Runtime binding decisions*, where certain plan choices are deferred until runtime, allowing the execution engine to select among several alternative plans by potentially re-invoking the optimizer.
- *In-operator adaptive logic,* where scheduling and other decisions are made part of the individual query operators, rather than the optimizer.

Many adaptability techniques rely on a symmetric hash join operator that offers a non-blocking variant of join by building hash tables on both the input relations. When an input tuple is read, it is stored in the appropriate hash table and probed against the opposite table, thus producing incremental output. The symmetric hash join operator can process data from either input, depending on availability. It also enables additional adaptivity, since it has frequent moments of symmetry, that is, points at which the join order can be changed without compromising correctness or losing work.

The eddy operator provides an example of fine-grained run-time control by tuple routing through operators. An eddy is used as a tuple router; it monitors execution, and makes routing decisions for the tuples. Eddies achieve adaptability by simply changing the order in which the tuples are routed through the operators. The degree of adaptability achieved depends on the type of the operators. Pipelined operators, such as the symmetric hash join, offer the most freedom, whereas, blocking operators, such as the sort-merge join, are less suitable since they do not produce output before consuming the input relations in their entirety.

Cross-references

- ▶ Adaptive Stream Processing
- ▶ Cost Estimation
- ▶ Multi-query Optimization
- ▶ Query Optimization
- ▶ Query Processing

Recommended Reading

1. Avnur R. and Hellerstein J.M. Eddies: continuously adaptive query processing. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 261–272.
2. Babu S. and Bizarro P. Adaptive query processing in the looking glass. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 238–249.
3. Deshpande A., Ives Z.G., and Raman V. Adaptive query processing. Found. Trends Databases, 1(1):1–140, 2007.

Adaptive Stream Processing

Zachary Ives
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

Adaptive query processing

Definition

When querying long-lived data streams, the characteristics of the data may change over time or data may arrive in bursts – hence, the traditional model of optimizing a query prior to executing it is insufficient. As a result, most data stream management systems employ feedback-driven *adaptive stream processing*, which continuously re-optimizes the query execution plan based on data and stream properties, in order to meet certain performance or resource consumption goals. Adaptive stream processing is a special case of the more general problem of *adaptive query processing*, with the special property that intermediate results are bounded in size (by stream windows), but where query processing may have quality-of-service constraints.

Historical Background

The field of adaptive stream processing emerged in the early 2000s, as two separate developments converged. *Adaptive* techniques for database query processing had become an area of increasing interest as Web and integration applications exceeded the capabilities of conventional static query processing [10]. Simultaneously, a number of data stream management systems [1,6,8,12] were emerging, and each of these needed capabilities for query optimization. This led to a common approach of developing feedback-based re-optimization strategies for stream query computation. In contrast to Web-based adaptive query processing techniques, the focus in adaptive stream processing has especially been on maintaining quality of service under overload conditions.

Foundations

Data stream management systems (DSMSs) typically face two challenges in query processing. First, the data to be processed comes from remote feeds that may be subject to significant variations in distribution or arrival rates over the lifetime of the query, meaning that no single query evaluation strategy may be appropriate over the entirety of execution. Second, DSMSs may be *underprovisioned* in terms of their ability to handle bursty input at its maximum rate, and yet may still need to meet certain *quality-of-service* or resource constraints (e.g., they may need to ensure data is processed within some latency bound). These two challenges have led to two classes of adaptive stream processing techniques: those that attempt to *minimize the cost* of computing query results from the input data (the problem traditionally faced by query optimization), and those that attempt to manage query processing, possibly at reduced accuracy, in the presence of *limited resources*. This article provides an overview of significant work in each area.

Minimizing Computation Cost

The problem of adaptive query processing to minimize computation cost has been well-studied in a variety of settings [10]. What makes the adaptive stream processing setting unique (and unusually tractable) is the fact that joins are performed over *sliding windows* with size bounds: As the data stream exceeds the window size, old data values are expired. This means intermediate state within a query plan operator has constant maximum size; as opposed to being bounded by the size of the input data. Thus a windowed join operator can be modeled as a pair of filter operators, each of which joins its input with the bounded intermediate state produced from the other input. Optimization of joins in data stream management systems becomes a minor variation on the problem of optimizing selection or filtering operators; hence certain theoretical optimality guarantees can actually be made.

Eddies Eddies [2,11,14] are composite dataflow operators that model select-project-join expressions. An eddy consists of a *tuple router*, plus a set of primitive *query operators* that run concurrently and each have input queues. Eddies come in several variations; the one proposed for distributed stream management uses *state modules* (SteMs) [14,11]. Figure 1 shows an example of such an eddy for a simplified stream SQL query, which joins three streams and applies a selection predicate over them.

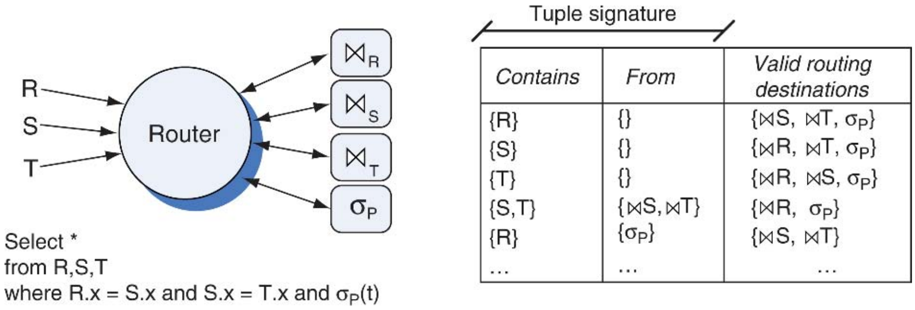
Eddy creation. The eddy is created prior to execution by an optimizer: every selection operation (σ_P in the example) is converted to a corresponding operator; additionally, each base relation to be joined is given a *state module*, keyed on the join attribute, to hold the intermediate state for each base relation [14] ($\bowtie_R$, $\bowtie_S$, $\bowtie_T$). If a base relation appears with multiple different join attributes, then it may require multiple SteMs. In general, the state module can be thought of as one of the hash tables within a symmetric or pipelined hash join. The optimizer also determines whether the semantics of the query force certain operators to execute before others. Such constraints are expressed in an internal routing table, illustrated on the right side of the figure. As a tuple is processed, it is annotated with a *tuple signature* specifying what input streams' data it contains and what operator may have last modified it. The routing table is a map from the tuple signature to a set of *valid routing destinations*, those operators that can successfully process a tuple with that particular signature.

Query execution/tuple routing. Initially, a tuple from an input data stream (R, S, or T) flows into the eddy router. The eddy (i) adds the data to the associated SteM or SteMs, and (ii) consults the routing table to determine the set of possible destination operators. It then chooses a destination (using a *policy* to be described later) and sends the tuple to the operator. The operator then either *filters* the tuple, or *produces* one or more output tuples, as a result of applying selection conditions or joining with the data within a SteM. Output tuples are marked as having been processed by the operator that produced them. If they have been processed by all operators, they will be sent to the query output, and if not, they will be sent back to the eddy's router and to one of the remaining operators.

Routing policies. The problem of choosing among alternate routing destinations has been addressed with a variety of strategies.

Tickets and lottery scheduling [2]. In this scheme, each operator receives a *ticket* for each tuple it receives from the router, and it returns the ticket each time it outputs a tuple to the router. Over time, each operator is expected to have a number of tickets proportional to $(1 - p)$ where p is the operator's selectivity. The router holds a *lottery* among valid routing destinations, where each operator's chance of winning is proportional to its number of tickets. Additionally, as a flow control mechanism, each operator has an input queue, and if this queue fills, then the operator may not participate in the lottery.

Deterministic with batching [9]. A later scheme was developed to reduce the per-tuple overhead of eddies by choosing destinations for batches of tuples. Here, each operator's selectivity is explicitly monitored and each predicate is assumed to be independent. Periodically, a *rank ordering* algorithm is used to choose a



Select *
from R,S,T
where R.x = S.x and S.x = T.x and $\sigma_P(t)$

Contains	From	Valid routing destinations
{R}	{}	$\{\bowtie S, \bowtie T, \sigma_P\}$
{S}	{}	$\{\bowtie R, \bowtie T, \sigma_P\}$
{T}	{}	$\{\bowtie R, \bowtie S, \sigma_P\}$
{S,T}	$\{\bowtie S, \bowtie T\}$	$\{\bowtie R, \sigma_P\}$
{R}	$\{\sigma_P\}$	$\{\bowtie S, \bowtie T\}$
...	...	...

Adaptive Stream Processing. Figure 1. Illustration of eddy with SteMs.

destination for a *batch* of tuples: the rank ordering algorithm sorts predicates in decreasing order of $c_i/(1 - p_i)$, where c_i is the cost of the applying predicate σ_i and p_i is its selectivity.

Content-based routing [7]. (CBR) attempts to learn correlations between attribute values and selectivities. Using sampling, the system determines for each operator the attribute most strongly correlated with its selectivity – this is termed the *classifier attribute*. CBR then builds a table characterizing all operators' selectivities for different values of each classifier attribute. Under this policy, when the eddy needs to route a tuple, it first looks up the tuple's classifier attribute values in the table and determines the destination operators' selectivities. It routes the tuple probabilistically, choosing a next operator with probability inversely proportional to its selectivity.

Other optimization strategies. An alternative strategy that does not use the eddies framework is the *adaptive greedy* [5] (A-greedy) algorithm. A-greedy continuously monitors the selectivities of query predicates using a *sliding window profile*, a table with one Boolean attribute for each predicate in the query, and sampling. As a tuple is processed by the query, it may be chosen for sampling into the sliding window profile – if so, it is tested against every query predicate. The vector of Boolean results is added as a row to the sliding window profile. Then the sliding window profile is then used to create a *matrix view* $V[i, j]$ containing, for each predicate σ_i, the number of tuples in the profile that satisfy $\sigma_1 ... \sigma_{i-1}$ but not σ_j. From this matrix view, the reoptimizer seeks to maintain the constraint that the *i*th operation over an input tuple must have the lowest cost/selectivity ratio $c_i/(1 - p(S_i|S_1,...,S_{i-1}))$. The overall strategy has one of the few performance guarantees in the adaptive query processing space: if data properties were to converge, then performance would be within a factor of 4 of optimal [5].

Managing Resource Consumption

A common challenge in data stream management systems is limiting the use of resources – or accommodating limited resources while maintaining quality of service, in the case of bursty data. We discuss three different problems that have been studied: load shedding to ensure input data is processed by the CPU as fast as it arrives, minimizing buffering and memory consumption during data bursts, and minimizing network communication with remote streaming sites.

Load Shedding. Allows the system to selectively drop data items to ensure it can process data as it arrives. Both the Aurora and STREAM DSMSs focused heavily on adaptive load shedding.

Aurora. In the Aurora DSMS [15], load shedding for a variety of query types are supported: the main requirement is that the user has a *utility function* describing the value of output data relative to how much of it has been dropped. The system seeks to place load shedding operators in the query plan in a way that maximizes the user's utility function while the system achieves sufficient throughput. Aurora precomputes conditional load shedding plans, in the form of a *load shedding road map* (LRSM) containing a sequence of plans that shed progressively more load; this enables the runtime system to rapidly move to strategies that shed more or less load.

LRSMs are created using the following heuristics: first, load shedding points are only inserted at data input points or at points in which data is split to two or more operators. Second, for each load shedding point, a *loss/gain ratio* is computed: this is the reduction in output utility divided by the gain in cycles, $R(p \cdot L - D)$, where R is the input rate into the drop point, p is the ratio of tuples to be dropped, L is the amount of system load flowing from the drop point, and D is the cost of the drop operator. Drop operators are injected at load shedding points in decreasing order of loss/gain ratio. Two different types of drops are considered using the same framework: *random drop*, in which an operator is placed in the query plan to randomly drop some fraction p of tuples; and *semantic drop*, which drops the p tuples of lowest utility. Aurora assumes for the latter case that there exists a utility function describing the relative worth of different attribute values.

Stanford STREAM. The Stanford STREAM system [4] focuses on aggregate (particularly SUM) queries. Again the goal is to process data at the rate it arrives, while minimizing the inaccuracy in query answers: specifically, the goal is to minimize the *maximum relative error across all queries*, where the relative error of a query is the difference between actual and approximate value, divided by the actual value.

A Statistics Manager monitors computation and provides estimates of each operator's selectivity and its running time, as well as the mean value and standard deviation of each query q_i's aggregate operator. For each q_i, STREAM computes an error threshold C_i, based on the mean, standard deviation, and number of values. (The results are highly technical so the reader is referred to [4] for more details.) A sampling rate P_i is chosen for query q_i that satisfies $P_i \geq C_i/\in_i$, where $\in_i$ is the allowable relative error for the query.

As in Aurora's load shedding scheme, STREAM only inserts load shedding operators at the inputs or at the start of shared segments. Moreover, if a node has a set of children who all need to shed load, then a portion of the load shedding can be "pulled up" to the parent node, and all other nodes can be set to shed some amount of additional load *relative* to this. Based on this observation, STREAM creates a query dataflow graph in which each path from source to sink initially traverses through a load shedding operator whose sampling rate is determined by the desired error rate, followed by additional load shedding operators whose sampling rate is expressed relative to that first operator. STREAM iterates over each path, determines a sampling rate for the initial load shedding operator to satisfy the load constraint, and then computes the maximum relative error for any query. From this, it can set the load shedding rates for individual operators.

Memory Minimization. STREAM also addresses the problem of minimizing the amount of space required to buffer data in the presence of burstiness [3]. The Chain algorithm begins by defining a *progress chart* for each operator in the query plan: this chart plots the relative size of the operator output versus the time it takes to compute. A point is plotted at time 0 with the full size of the input, representing the start of the query; then each operator is given a point according to its cost and relative output size. Now a *lower envelope* is plotted on the progress chart: starting with the initial point at time 0, the *steepest* line is plotted to any operator to the right of this point; from the point at the end of the first line, the next steepest line is plotted to a successor operator; etc. Each line segment (and the operators whose points are plotted beside it) represents a chain, and operators within a chain are scheduled together. During query processing, at each time "tick," the scheduler considers all tuples that have been output by any chain. The tuple that lies on the segment with *steepest slope* is the one that is scheduled next; as a tie-breaker, the earliest such tuple is scheduled. This Chain algorithm is proven to be near-optimal (differing by at most one unit of memory per operator path for queries where selectivity is at most one).

Minimizing Communication. In some cases, the constrained resource is the network rather than CPU or memory. Olston et al. [13] develop a scheme for reducing network I/O for AVERAGE queries, by using accuracy bounds. Each remote object O is given a *bound width* w_O: the remote site will only notify the central query processor if O's value V falls outside this bound. Meanwhile, the central site maintains a *bound cache* with the last value and the bound width for every object.

If given a *precision constraint* δ_j for each query Q_j, then if the query processor is to provide query answers within δ_j, the sum of the bound widths for the data objects of Q_j must not exceed δ_j times the number of objects. The challenge lies in the selection of widths for the objects.

Periodically, the system tries to tighten all bounds, in case values have become more stable; objects whose values fall outside the new bounds get reported back to the central site. Now some of those objects' bounds must be loosened in a way that maintains the precision constraints over all queries. Each object O is given a *burden score* equal to $c_O/(p_O w_O)$, where c_O is the cost of sending the object, w_O is its bound width, and p_O is the frequency of updates since the previous width adjustment. Using an approximation method based on an iterative linear equation solver, Olston et al. compute a *burden target* for each query, i.e., the lowest overall burden score required to always meet the query's precision constraint. Next, each object is assigned a *deviation*, which is the maximum difference between the object's burden score and any query's burden target. Finally, a queried objects' bounds are adjusted in decreasing order of deviation, and each object's bound is increased by the largest amount that still conforms to the precision constraint for every query.

Key Applications

Data stream management systems have seen significant adoption in areas such as sensor monitoring and processing of financial information. When there are associated quality-of-service constraints that might require load shedding, or when the properties of the data are subject to significant change, adaptive stream processing becomes vitally important.

Future Directions

One of the most promising directions of future study is how to best use a combination of offline modeling, selective probing (in parallel with normal query execution), and feedback from query execution to find optimal strategies quickly. Algorithms with certain optimality guarantees are being explored in the online learning and theory communities (e.g., the *k*-armed bandit problem), and such work may lead to new improvements in adaptive stream processing.

Cross-references

▶ Distributed Stream
▶ Query Processor
▶ Stream Processing

Recommended Reading

1. Abadi D.J., Carney D., Cetintemel U., Cherniack M., Convey C., Lee S., Stonebraker M., Tatbul N., and Zdonik S. Aurora: a new model and architecture for data stream management. VLDB J., 12(2):120–139, 2003.
2. Avnur R. and Hellerstein J.M. Eddies: continuously adaptive query processing. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 261–272.
3. Babcock B., Babu S., Datar M., and Motwani R. Chain: operator scheduling for memory minimization in data stream systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 253–264.
4. Babcock B., Datar M., and Motwani R. Load shedding for aggregation queries over data streams. In Proc. 20th Int. Conf. on Data Engineering, 2004, p. 350.
5. Babu S., Motwani R., Munagala K., Nishizawa I., and Widom J. Adaptive ordering of pipelined stream filters. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 407–418.
6. Balazinska M., BalaKrishnan H., and Stonebraker M. Demonstration: load management and high availability in the Medusa distributed stream processing system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 929–930.
7. Bizarro P., Babu S., DeWitt D.J., and Widom J. Content-based routing: different plans for different data. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 757–768.
8. Chandrasekaran S., Cooper O., Deshpande A., Franklin M.J., Hellerstein J.M., Hong W., Krishnamurthy S., Madden S., Raman V., Reiss F., and Shah M.A. TelegraphCQ: continuous dataflow processing for an uncertain world. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
9. Deshpande A. An initial study of overheads of eddies. ACM SIGMOD Rec., 33(1):44–49, 2004.
10. Deshpande A., Ives Z., and Raman V. Adaptive query processing. Found. Trends Databases, 1(1):1–140, 2007.
11. Madden S., Shah M.A., Hellerstein J.M., and Raman V. Continuously adaptive continuous queries over streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 49–60.
12. Motwani R., Widom J., Arasu A., Babcock B., Babu S., Datar M., Manku G., Olston C., Rosenstein J., and Varma R. Query processing, resource management, and approximation in a data stream management system. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
13. Olston C.,Jiang J., and Widom J., Adaptive filters for continuous queries over distributed data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 563–574.
14. Raman V., Deshpande A., and Hellerstein J.M. Using state modules for adaptive query processing. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 353–366.
15. Tatbul N., Cetintemel U., Zdonik S.B., Cherniack M., and Stonebraker M. Load shedding in a data stream manager. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 309–320.

Adaptive Workflow/Process Management

▶ Workflow Evolution

ADBMS

▶ Active Database Management System Architecture

ADBMS Framework

▶ Active Database Management System Architecture

ADBMS Infrastructure

▶ Active Database Management System Architecture

Adding Noise

▶ Matrix Masking

Additive Noise

▶ Noise Addition

Administration Model for RBAC

Yue Zhang, James B. D. Joshi
University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

ARBAC97; SARBAC

Definition

The central ideal of administration model for RBAC is to use the role itself to manage roles. There are two well-known families of administration RBAC models.

Administrative RBAC

The Administrative RBAC family of models known as ARBAC97 [3] introduces administrative roles that are used to manage the regular roles. These roles can form a role hierarchy and may have constraints. ARBAC97 consists of three administrative models, the user-role assignment (URA97) model, the permission-role assignment (PRA97) model, and the role-role administration (RRA97) model. URA97 defines which administrative roles can assign which users to which regular roles by means of the relation: *can_assign*. Similarly, PRA97 defines which administrative roles can assign which permissions to which regular roles by means of the relation: *can_assignp*. Each of these relations also has a counterpart for revoking the assignment (e.g., *can_revoke*). RRA97 defines which administrative roles can change the structure (add roles, delete roles, add edges, etc.) of which range of the regular roles using the notion of *encapsulated range* and the relation: *can_modify*.

Scoped Administrative RBAC

The SARBAC model uses the notion of *administrative scope* to ensure that any operations executed by a role *r* will not affect other roles due to the hierarchical relations among them [1]. There are no special administrative roles in SARBAC, and each regular role has a *scope* of other regular roles called *administrative scope* that can be managed by it. Each role can only be managed by its administrators. For example, a senior-most role should be able to manage all its junior roles.

Key Points

ARBAC model is the first known role-based administration model and uses the notion of *range* and *encapsulated range*. Role *range* is essentially a set of regular roles. To avoid undesirable side effects, RRA97 requires that all role ranges in the *can_modify* relation be encapsulated, which means the range should have exactly one senior-most role and one junior-most role. Sandhu et al. later extended the ARBAC97 model into ARBAC99 model where the notion of mobile and immobile user/permission was introduced [4]. Oh et al. later extended ARBAC99 to ARBAC02 by adding the notion of organizational structure to redefine the user-role assignment and the role-permission assignment [2]. Recently, Zhang et al. have proposed an ARBAC07 model that extends the family of ARBAC models to deal with an RBAC model that allows hybrid hierarchies to co-exit [6].

SARBAC

The most important notion in SARBAC is that of the *administrative scope*, which is similar to the notion of *encapsulated range* in ARBAC97. A role *r* is said to be within to be the administrative scope of another role *a* if every path upwards from *r* goes through *a*, and *a* is said to be the administrator of *r*. SARBAC also consists of three models: SARBAC-RHA, SARBAC-URA, and SARBAC-PRA. In SARBAC-RHA, each role can only administer the roles that are within its own administrative scope. The operations include adding roles, deleting roles, adding permissions, and deleting permissions. The semantics for SARBAC-URA and SARBAC-PRA is similar to URA97 and PRA97. The administrative scope can change dynamically. Zhang et al. have extended SARBAC to also deal with hybrid hierarchy [5].

Cross-references

▶ Role Based Access Control

Recommended Reading

1. Crampton J. and Loizou G. Administrative scope: a foundation for role-based administrative models. ACM Trans. Inf. Syst. Secur., 6(2):201–231, 2003.
2. Oh S. and Sandhu R. A model for role administration using organization structure. In Proc. 7th ACM Symp. on Access Control Models and Technologies, 2002, pp. 155–162.

3. Sandhu R., Bhamidipati V., and Munawer Q. The ARBAC97 model for role-based administration of roles. ACM Trans. Inf. Syst. Secur., 2(1):105–135, 1999.
4. Sandhu R. and Munawer Q. The ARBAC99 model for administration of roles (1999). In Proc. 15th Computer Security Applications Conf. Arizona, 1999, pp. 229.
5. Zhang Y., James B., and Joshi D. "SARBAC07: scoped administration model for RBAC with hybrid hierarchy. In Proc. 3rd Int. Symp. on Information Assurance and Security, 2007, pp. 149–154.
6. Zhang Y. and Joshi J.B.D. ARBAC07: a role based administration model for RBAC with hybrid hierarchy. In Proc. IEEE Int. Conf. Information Reuse and Integration, 2007, pp. 196–202.

Administration Wizards

Philippe Bonnet[1], Dennis Shasha[2]
[1]University of Copenhagen, Copenhagen, Denmark
[2]New York University, New York, NY, USA

Definition

Modern database systems provide a collection of utilities and programs to assist a database administrator with tasks such as database installation and configuration, import/export, indexing (index wizards are covered in the self-management entry), and backup/restore.

Historical Background

Database Administrators have been skeptical of any form of automation as long as they could control the performance and security of a relatively straightforward installation. The advent of enterprise data management towards the end of the 1990s, where few administrators became responsible for many, possibly diverse database servers, has led to the use of graphical automation tools. In the mid-1990s, third party vendors introduced such tools. With SQL Server 6.5, Microsoft was the first constructor to provide an administration wizard.

Foundations

Installation and Configuration

Database servers are configured using hundreds of parameters that control everything buffer size, file layout, concurrency control options and so on. They are either set statically in a configuration file before the server is started, or dynamically while the server is running. Out-of-the-box database servers are equipped with a limited set of typical configurations.

The installation/configuration wizard is a graphical user interface that guides the administrator through the initial server configuration. The interface provides high-level choices (e.g., OLTP vs. OLAP workload), or simple questions (e.g., number of concurrent users) that are mapped onto database configuration values (log buffer size and thread pool size respectively).

Data Import/Export

Import/export wizards are graphical tools that help database administrators map a database schema with an external data format (e.g., XML, CSV, PDF), or generate scripts that automate the transfer of data between a database and an external data source (possibly another database server).

Back-up/Restore

Back-up/restore wizards automate the back-up procedure given a few input arguments: complete/incremental backup, scope of the back-up/restore operations (file, tablespace, database), target directory.

Key Applications

Automation of the central database administration tasks.

Cross-references

► Self-Management

Recommended Reading

1. Bersinic D. and Gile S. Portable DBA: SQL Server. McGraw Hill, New York, 2004.
2. Schumacher R. DBA Tools Today. DBMS Magazine, January 1997.

Advanced Transaction Models

► Extended Transaction Models and the ACTA Framework

► Generalization of ACID Properties

► Open Nested Transaction Models

Adversarial Information Retrieval

► Web Spam Detection

Affix Removal

▶ Stemming

AFI

▶ Approximation of Frequent Itemsets

Aggregate Queries in P2P Systems

▶ Approximate Queries in Peer-to-Peer Systems

Aggregation

▶ Abstraction

Aggregation Algorithms for Middleware Systems

▶ Top-k Selection Queries on Multimedia Datasets

Aggregation and Threshold Algorithms for XML

▶ Ranked XML Processing

Aggregation: Expressiveness and Containment

Sara Cohen
The Hebrew University of Jerusalem, Jerusalem, Israel

Definition

An *aggregate function* is a function that receives as input a multiset of values, and returns a single value. For example, the aggregate function `count` returns the number of input values. An *aggregate query* is simply a query that mentions an aggregate function, usually as part of its output. Aggregate queries are commonly used to retrieve concise information from a database, since they can cover many data items, while returning few. Aggregation is allowed in SQL, and the addition of aggregation to other query languages, such as relational algebra and datalog, has been studied.

The problem of determining *query expressiveness* is to characterize the types of queries that can be expressed in a given query language. The study of query expressiveness for languages with aggregation is often focused on determining how aggregation increases the ability to formulate queries. It has been shown that relational algebra with aggregation (which models SQL) has a *locality property.*

Query containment is the problem of determining, for any two given queries q and q', whether $q(D) \subseteq q'(D)$, for all databases D, where $q(D)$ is the result of applying q to D. Similarly, the *query equivalence problem* is to determine whether $q(D) = q'(D)$ for all databases D. For aggregate queries, it seems that characterizing query equivalence may be easier than characterizing query containment. In particular, almost all known results on query containment for aggregate queries are derived by a reduction from query equivalence.

Historical Background

The SQL standard defines five aggregate functions, namely, `count`, `sum`, `min`, `max` and `avg` (average). Over time, it has become apparent that users would like to aggregate data in additional ways. Therefore, major database systems have added new built-in aggregate functions to meet this need. In addition, many database systems now allow the user to extend the set of available aggregate functions by defining his own aggregate functions.

Aggregate queries are typically used to summarize detailed information. For example, consider a database with the relations `Dept(deptId, deptName)` and `Emp(empId, deptId, salary)`. The following SQL query returns the number of employees, and the total department expenditure on salaries, for each department which has an average salary above $10,000.

```
(Q1) SELECT deptID, count(empID),
            sum(salary)
     FROM Dept, Emp
     WHERE Dept.deptID = Emp.deptID
     GROUP BY Dept.deptID
     HAVING avg(salary) > 10000
```

Typically, aggregate queries have three special components. First, the `GROUP BY` clause is used to state how intermediate tuples should be grouped before applying aggregation. In this example, tuples are grouped by their value of `deptID`, i.e., all tuples with the same value for this attribute form a single group. Second, a `HAVING` clause can be used to determine which groups are of interest, e.g., those with average salary above $10,000. Finally, the outputted aggregate functions are specified in the `SELECT` clause, e.g., the number of employees and the sum of salaries.

The inclusion of aggregation in SQL has motivated the study of aggregation in relational algebra, as an abstract modeling of SQL. One of the earliest studies of aggregation was by Klug [11], who extended relational algebra and relational calculus to allow aggregate functions and showed the equivalence of these two languages. Aggregation has also been added to Datalog. This has proved challenging since it is not obvious what semantics should be adopted in the presence of recursion [15].

Foundations

Expressiveness

The study of query expressiveness deals with determining what can be expressed in a given query language. The expressiveness of query languages with aggregation has been studied both for the language of relational algebra, as well as for datalog, which may have recursion.

Various papers have studied the expressive power of nonrecursive languages, extended with aggregation, e.g., [7,9,13]. The focus here will be on [12], which has the cleanest, general proofs for the expressive power of languages modeling SQL.

In [12], the expressiveness of variants of relational algebra, extended with aggregation, was studied. First, [12] observes that the addition of aggregation to relational algebra strictly increases its expressiveness. This is witnessed by the query `Q2`:

```
(Q2) SELECT 1
     FROM R1
     WHERE (SELECT COUNT(*) FROM R) >
               (SELECT COUNT(*) FROM S)
```

Observe that `Q2` returns 1 if `R` contains more tuples than `S`, and otherwise an empty answer. It is known that first-order logic cannot compare cardinalities, and hence neither can relational algebra. Therefore, SQL with aggregation is strictly more expressive than SQL without aggregation.

The language ALG_{aggr} is presented in [12]. Basically, ALG_{aggr} is relational algebra, extended by arbitrary aggregation and arithmetic functions. In ALG_{aggr}, non-numerical selection predicates are restricted to using only the equality relation (and not order comparisons). A *purely relational query* is one which is applied only to non-numerical data. It is shown that all purely relational queries in ALG_{aggr} are *local*. Intuitively, the answers to local queries are determined by looking at small portions of the input.

The formal definition of local queries follows. Let D be a database. The *Gaifman graph* $G(D)$ of D is the undirected graph on the values appearing in D, with $(a,b) \in G(D)$ if a and b belong to the same tuple of some relation in D. Let $\vec{a} = (a_1,\ldots,a_k)$ be a tuple of values, each of which appears in D. Let r be an integer, and let $S_r^D(\vec{a})$ be the set of values b such that $dist\ (a_i, b) \leq r$ in $G(D)$, for some i. The *r-neighborhood* $N_r^D(\vec{a})$ of $\vec{a}$ is a new database in which the relations of D are restricted to contain only the values in $S_r^D(\vec{a})$. Then, $\vec{a}$ and $\vec{b}$ are *(D, r)-equivalent* if there is an isomorphism $h : N_r^D(\vec{a}) \rightarrow N_r^D(\vec{b})$ such that $h(\vec{a}) = \vec{b}$. Finally, a q is *local* if there exists a number r such that for all D, if $(\vec{a})$ and $(\vec{b})$ are (D, r)-equivalent, then $\vec{a} \in q(D)$ if and only if $\vec{b} \in q(D)$.

There are natural queries that are not local. For example, transitive closure (also called reachability) is not local. Since all queries in ALG_{aggr} are local, this implies that transitive closure cannot be expressed in ALG_{aggr}.

In addition to ALG_{aggr}, [12] introduces the languages $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{N}}$ and $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{Q}}$. $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{N}}$ and $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{Q}}$ are the extensions of ALG_{aggr} which allow order comparisons in the selection predicates, and allow natural numbers and rational numbers, respectively, in the database. It is not known whether transitive closure can be expressed in $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{N}}$. More precisely, [12] shows that if transitive closure is not expressible in $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{N}}$, then the complexity class Uniform TC^0 is properly contained in the complexity class NLOGSPACE. Since the latter problem (i.e., determining strict containment of TC^0 in NLOGSPACE) is believed to be very difficult to prove, so is the former. Moreover, this result holds even if the arithmetic functions are restricted to $\{+,\cdot,<,0,1\}$ and the aggregate functions are restricted to $\{\texttt{sum}\}$. On the other hand, $\mathrm{ALG}^{\leq,\mathbb{Q}}_{\mathrm{aggr}}$ extended by arbitrary aggregation and arithmetic functions, can express all computable queries.

The languages $\mathrm{ALG}_{\mathrm{aggr}}$, $\mathrm{ALG}^{\leq,\mathbb{N}}_{\mathrm{aggr}}$ and $\mathrm{ALG}^{\leq,\mathbb{Q}}_{\mathrm{aggr}}$ are based on relational algebra, and therefore, do not allow recursion. The Datalog language allows queries to be defined as programs, containing recursion. The meaning of an aggregate function within a recursive program, is not always well-defined. One solution is to restrict the program to have only *stratified aggregation.* Stratification means that if a derived predicate p is defined by applying an aggregate function on a derived predicate q, then the definition of q does not depend, syntactically, upon the definition of p. For example, consider the following Datalog program, P_1.

$$\begin{aligned} p(X,\ \texttt{sum}(Y)) &\leftarrow q(X,Y)\\ q(X,Y) &\leftarrow a(X,Y)\\ q(X,Z) &\leftarrow q(X,Y), q(Y,Z) \end{aligned}$$

The program P_1 is stratified. Replacing the final rule in P_1 with

$$q(X,Z) \leftarrow q(X,Y), p(Y,Z)$$

would yield a program with nonstratified aggregation.

The expressiveness of stratified aggregation was studied in [14]. Only the aggregate functions `sum`, `avg`, `min`, `max` and `count` were allowed. It is shown that that stratified aggregation cannot express *summarized explosion* (i.e., the number of instances of a part needed to construct a bigger part). On the other hand, if the language is extended to allow the function +, as well as the constants 0 and 1, then all computable queries on the integer domain can be expressed. This is correct even if the only aggregate function allowed is `max`. Additional results of this type, i.e., expressibility of other fragments of stratified Datalog, also appear in [14].

Query Containment

The equivalence and containment problems for aggregate queries have been studied for nonrecursive Datalog programs. A survey of the containment and equivalence problems for aggregate queries, containing references to most works on this topic, appears in [2].

Deriving general characterizations of containment (or equivalence) for aggregate queries is difficult, since each aggregate function tends to have its own idiosyncrasies. For example, `count` is sensitive to the number of occurrences of each value, but not to the values themselves, whereas `max` ignores repeated values, but is sensitive to the exact values appearing. As another example, `sum` ignores the value 0, whereas `prod` ignores 1. In addition, `prod` always returns 0 if it is applied to a bag containing 0.

Due to aggregate function quirks, it is often the case that equivalent queries are no longer so, if the aggregate function appearing in their head changes. To demonstrate, consider the two pairs of queries q_1, q_1' and q_2, q_2'.

$$\begin{aligned} q_1(X,\texttt{count}) &\leftarrow a(X,Y)\\ q_1'(X,\texttt{count}) &\leftarrow a(X,Y), a(X,Z)\\ q_2(X,\texttt{max}(Y)) &\leftarrow a(X,Y)\\ q_2'(X,\texttt{max}(Y)) &\leftarrow a(X,Y), a(X,Z) \end{aligned}$$

The queries q_1 and q_2 (and similarly q_1' and q_2') have the same conditions in their body, and differ only on the output aggregate function. One may show that q_1 is not equivalent to q_1' (nor is there containment in either direction), as witnessed by the database

$$D_1 = \{a(\texttt{c},0), a(\texttt{c},1), a(\texttt{d},0)\}$$

over which $q_1(D_1) = \{(\texttt{c},2),(\texttt{d},1)\}$ and $q_1'(D_1) = \{(\texttt{c},4),(\texttt{d},1)\}$. On the other hand, $q_2 \equiv q_2'$ does hold.

The different oddities of aggregate functions make finding a general solution for the equivalence and containment problems very difficult. Thus, characterizations for equivalence of aggregate queries often are defined separately for each aggregate function. Most known characterizations for equivalence are based on checking for the existence of special types of mappings between the queries. For example, conjunctive queries (i.e., Datalog programs consisting of a single rule, and no negation) with the aggregate function `count`, are equivalent if and only if they are isomorphic [1,4].

For other types of aggregate functions, as well as for `count` queries with comparisons or disjunctions, isomorphism is not a necessary condition for equivalence. To demonstrate, each pair of queries q_i, q_i' below is equivalent, yet not isomorphic:

$$q_3(\texttt{count}) \leftarrow b(X), b(Y), b(Z), X < Y, X < Z$$
$$q_3'(\texttt{count}) \leftarrow b(X), b(Y), b(Z), X < Z, Y < Z$$

$$q_4(\texttt{sum}(Y)) \leftarrow b(Y), b(Z), Y > 0, Z > 0$$
$$q_4'(\texttt{sum}(Y)) \leftarrow b(Y), b(Z), Y \geq 0, Z > 0$$

$$q_5(\texttt{avg}(Y)) \leftarrow b(Y)$$
$$q_5'(\texttt{avg}(Y)) \leftarrow b(Y), b(Z)$$

$$q_6(\texttt{max}(Y)) \leftarrow b(Y), b(Z_1), b(Z_2), Z_1 < Z_2$$
$$q_6'(\texttt{max}(Y)) \leftarrow b(Y), b(Z), Z < Y$$

Characterizations for equivalence are known for queries of the above types. Specifically, characterizations have been presented for equivalence of conjunctive queries with the aggregate functions count, sum, max and count-distinct [4] and these were extended in [5] to queries with disjunctive bodies. Equivalence of conjunctive queries with avgand with percent were characterized in [8].

It is sometimes possible to define classes of aggregate functions and then present general characterizations for equivalence of queries with any aggregate function within the class of functions. Such characterizations are often quite intricate since they must deal with many different aggregate functions. A characterization of this type was given in [6] to decide equivalence of aggregate queries with *decomposable* aggregate functions, even if the queries contain negation. Intuitively, an aggregate function is decomposable if partially computed values can easily be combined together to return the result of aggregating an entire multiset of values, e.g., as is the case for count, sum and max.

Interestingly, when dealing with aggregate queries it seems that the containment problem is more elusive than the equivalence problem. In fact, for aggregate queries, containment is decided by reducing to the equivalence problem. A reduction of containment to equivalence is presented for queries with *expandable* aggregate functions in [3]. Intuitively, for expandable aggregate functions, changing the number of occurrences of values in bags B and B' does not affect the correctness of the formula $\alpha(B)=\alpha(B')$, as long as the proportion of each value in each bag remains the same, e.g., as is the case for count, sum, max, count-distinct and avg.

The study of aggregate queries using the count function is closely related to the study of nonaggregate queries evaluated under *bag-set semantics*. Most past research on query containment and equivalence for nonaggregate queries assumed that queries are evaluated under *set semantics*. In set semantics, the output of a query does not contain duplicated tuples. (This corresponds to SQL queries with the DISTINCT operator.) Under *bag-set semantics* the result of a query is a multiset of values, i.e., the same value may appear many times. A related semantics is *bag semantics* in which both the database and the query results may contain duplication.

To demonstrate the different semantics, recall the database D_1 defined above. Consider evaluating, over D_1, the following variation of q_1:

$$q_1''(X) \leftarrow a(X, Y)$$

Under set-semantics $q''_1(D_1)$ ={ (c),(d)}, and under bag-set semantics $q''_1(D_1)$ ={{ (c),(c),(d)}}. Note the correspondence between bag-set semantics and using the count function, as in q_1, where count returns exactly the number of duplicates of each value. Due to this correspondence, solutions for the query containment problem for queries with the countfunction immediately give rise to solutions for the query containment problem for nonaggregate queries evaluated under bag-set semantics, and vice-versa.

The first paper to directly study containment and equivalence for nonaggregate queries under bag-set semantics was [1], which characterized equivalence for conjunctive queries. This was extended in [4] to queries with comparisons, in [5] to queries with disjunctions and in [6] to queries with negation.

Key Applications

Query Optimization

The ability to decide query containment and equivalence is believed to be a key component in query optimization. When optimizing a query, the database can use equivalence characterizations to remove redundant portions of the query, or to find an equivalent, yet cheaper, alternative query.

Query Rewriting

Given a user query q, and previously computed queries $v_1,\ldots,v_n$, the query rewriting problem is to find a query r that (i) is equivalent to q, and (ii) uses the queries

$v_1,\ldots,v_n$ instead of accessing the base relations. (Other variants of the query rewriting problem have also been studied.) Due to Condition (i), equivalence characterizations are needed to solve the query rewriting problem. Query rewriting is useful as an optimization technique, since it can be cheaper to use past results, instead of evaluating a query from scratch. Integrating information sources is another problem that can be reduced to the query rewriting problem.

Future Directions

Previous work on query containment does not consider queries with HAVING clauses. Another open problem is containment for queries evaluated under bag-set semantics. In this problem, one wishes to determine if the bag returned by q is always sub-bag of that returned by q'. (Note that this is different from the corresponding problem of determining containment of queries with count, which has been solved.) It has shown [10] that bag-set containment is undecidable for conjunctive queries containing inequalities. However, for conjunctive queries without any order comparisons, determining bag-set containment is still an open problem.

Cross-references

- ▶ Answering Queries using Views
- ▶ Bag Semantics
- ▶ Data Aggregation in Sensor Networks
- ▶ Expressive Power of Query Languages
- ▶ Locality
- ▶ Query Containment
- ▶ Query Optimization (in Relational Databases)
- ▶ Query Rewriting using Views

Recommended Reading

1. Chaudhuri S. and Vardi M.Y. Optimization of *real* conjunctive queries. In Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1993, pp. 59–70.
2. Cohen S. Containment of aggregate queries. ACM SIGMOD Rec., 34(1):77–85, 2005.
3. Cohen S., Nutt W., and Sagiv Y. Containment of aggregate queries. In Proc. 9th Int. Conf. on Database Theory, 2003, pp. 111–125.
4. Cohen S., Nutt W., and Sagiv Y. Deciding equivalences among conjunctive aggregate queries. J. ACM, 54(2), 2007.
5. Cohen S., Nutt W., and Serebrenik A. Rewriting aggregate queries using views. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 155–166.
6. Cohen S., Sagiv Y., and Nutt W. Equivalences among aggregate queries with negation. ACM Trans. Comput. Log., 6(2):328–360, 2005.
7. Consens M.P. and Mendelzon A.O. Low complexity aggregation in graphlog and datalog. Theor. Comput. Sci., 116(1 and 2): 95–116, 1993.
8. Grumbach S., Rafanelli M., and Tininini L. On the equivalence and rewriting of aggregate queries. Acta Inf., 40(8):529–584, 2004.
9. Hella L., Libkin L., Nurmonen J., and Wong L. Logics with aggregate operators. J. ACM, 48(4):880–907, 2001.
10. Jayram T.S., Kolaitis P.G., and Vee E. The containment problem for real conjunctive queries with inequalities. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 80–89.
11. Klug A.C. Equivalence of relational algebra and relational calculus query languages having aggregate functions. J. ACM, 29(3):699–717, 1982.
12. Libkin L. Expressive power of SQL. Theor. Comput. Sci., 3(296):379–404, 2003.
13. Libkin L. and Wong L. Query languages for bags and aggregate functions. J. Comput. Syst. Sci., 55(2):241–272, 1997.
14. Mumick I.S. and Shmueli O. How expressive is statified aggregation? Ann. Math. Artif. Intell., 15(3–4):407–434, 1995.
15. Ross K.A. and Sagiv Y. Monotonic aggregation in deductive database. J. Comput. Syst. Sci., 54(1):79–97, 1997.

Aggregation-Based Structured Text Retrieval

THEODORA TSIKRIKA
Center for Mathematics and Computer Science,
Amsterdam, The Netherlands

Definition

Text retrieval is concerned with the retrieval of documents in response to user queries. This is achieved by (i) representing documents and queries with indexing features that provide a characterisation of their information content, and (ii) defining a function that uses these representations to perform retrieval. *Structured text retrieval* introduces a finer-grained retrieval paradigm that supports the representation and subsequent retrieval of the individual document components defined by the document's *logical structure*. *Aggregation-based structured text retrieval* defines (i) the representation of each document component as the aggregation of the representation of its own information content and the representations of information content of its structurally related components, and

(ii) retrieval of document components based on these (aggregated) representations.

The aim of aggregation-based approaches is to improve retrieval effectiveness by capturing and exploiting the interrelations among the components of structured text documents. The representation of each component's own information content is generated at indexing time. The recursive aggregation of these representations, which takes place at the level of their indexing features, leads to the generation, either at indexing or at query time, of the representations of those components that are structurally related with other components.

Aggregation can be defined in numerous ways; it is typically defined so that it enables retrieval to focus on those document components more specific to the query or to each document's best entry points, i.e., document components that contain relevant information and from which users can browse to further relevant components.

Historical Background

A well-established Information Retrieval (IR) technique for improving the effectiveness of *text retrieval* (i.e., retrieval at the document level) has been the generation and subsequent combination of multiple representations for each document [3]. To apply this useful technique to the *text retrieval* of structured text documents, the typical approach has been to exploit their *logical structure* and consider that the individual representations of their components can act as the different representations to be combined [11]. This definition of the representation of a structured text document as the combination of the representations of its components was also based on the intuitive idea that the information content of each document consists of the information content of its sub-parts [2,6].

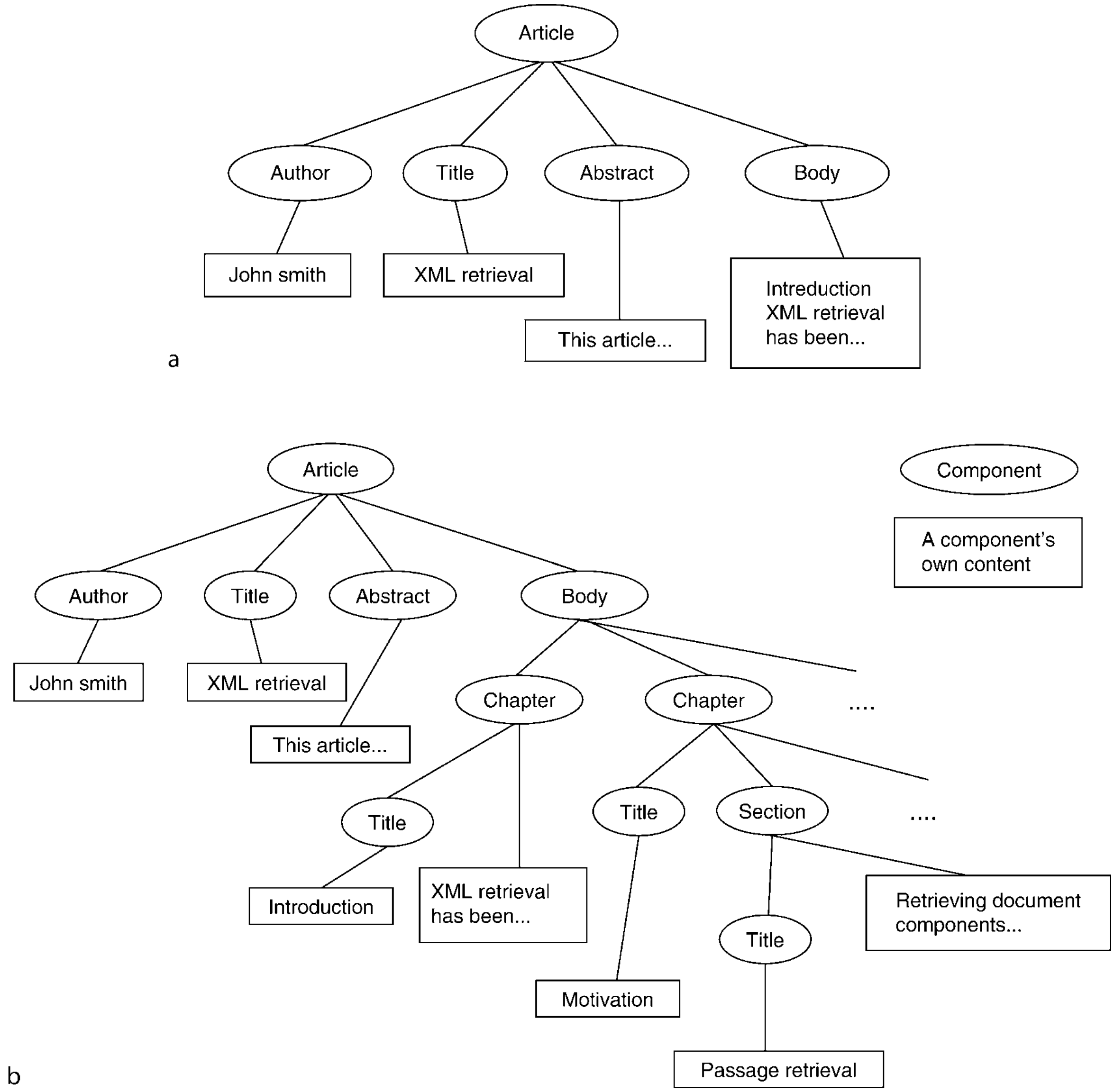
As the above description suggests, these combination-based approaches, despite restricting retrieval only at the document level, assign representations not only to documents, but also to individual document components. To generate these representations, structured text documents can simply be viewed as series of non-overlapping components (Figure 1a), such as title, author, abstract, body, etc. [13]. The proliferation of *SGML* and *XML* documents, however, has led to the consideration of hierarchical components (Figure 1b), and their interrelated representations [1]. For these (disjoint or nested) document components, the combination of their representations can take place (i) directly at the level of their indexing features, which typically correspond to terms and their statistics (e.g., [13]), or (ii) at the level of retrieval scores computed independently for each component (e.g., [15]). Overall, these combination-based approaches have proven effective for the *text retrieval* of structured text documents [11,13,15].

Following the recent shift towards the *structured text retrieval* paradigm [2], which supports the retrieval of document components (including whole documents), it was only natural to try to adapt these combination-based approaches to this new requirement for retrieval at the sub-document level. Here, the focus is on each document component: its representation corresponds to the combination of its own representation with the representations of its structurally related components, and its retrieval is based on this combined representation. Similarly to the case of combination-based approaches for *text retrieval*, two strands of research can be identified: (i) approaches that operate at the level of the components' indexing features (e.g., [12]), referred to as *aggregation-based structured text retrieval* (described in this entry), and (ii) approaches that operate at the level of retrieval scores computed independently for each component (e.g., [14]), referred to as *propagation*-based structured text retrieval.

Figure 2b illustrates the premise of aggregation- and propagated-based approaches for the simple structured text document depicted in Figure 2a. Since these approaches share some of their underlying motivations and assumptions, there has been a cross-fertilisation of ideas between the two. This also implies that this entry is closely related to the entry on *propagation*-based structured text retrieval.

Foundations

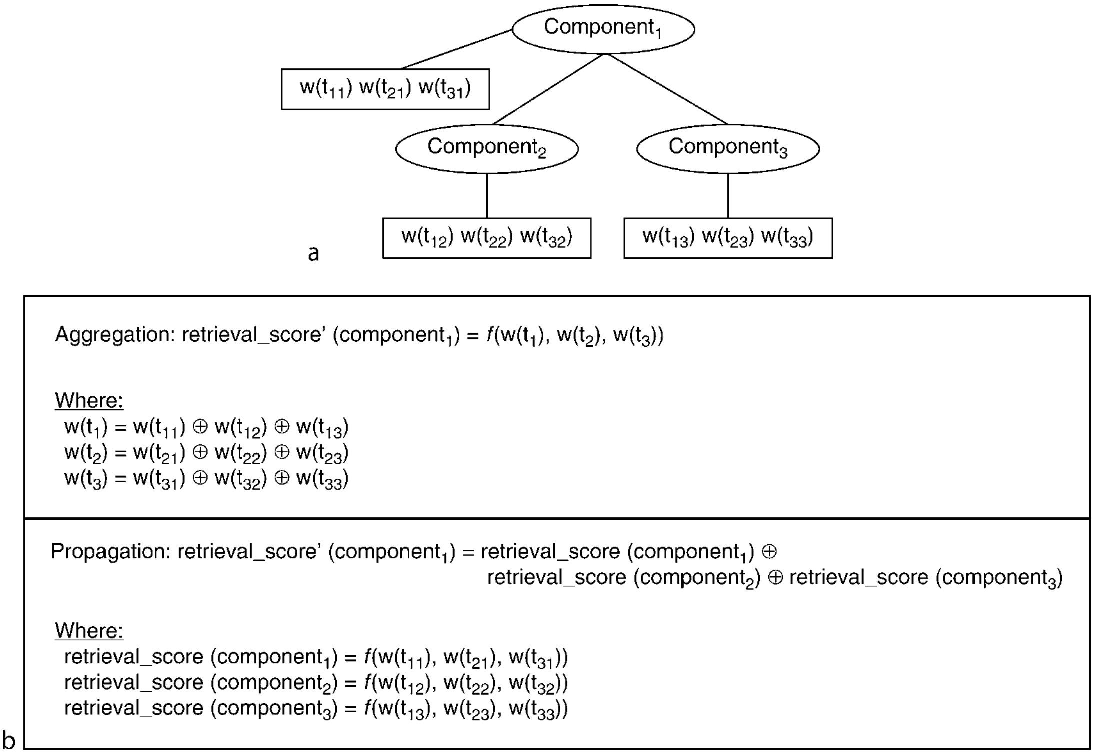
Structured text retrieval supports, in principle, the representation and subsequent retrieval of document components of any granularity; in practice, however, it is desirable to take into account only document components that users would find informative in response to their queries [1,2,4,6]. Such document components are referred to as *indexing units* and are usually chosen (manually or automatically) with respect to the requirements of each application. Once the *indexing*



Aggregation-Based Structured Text Retrieval. Figure 1. Two views on the logical structure of a structured text document.

units have been determined, each can be assigned a representation of its information content, and, hence, become individually retrievable.

Aggregation-based structured text retrieval approaches distinguish two types of *indexing units*: *atomic* and *composite*. Atomic components correspond to *indexing units* that cannot be further decomposed, i.e., the leaf components in Figure 1b. The representation of an atomic component is generated by considering only its own information content. Composite components, on the other hand, i.e., the non-leaf nodes in Figure 1b, correspond to *indexing units* which are related to other components, e.g., consist of sub-components. In addition to its own information content, a composite component is also dependent on the information content of its structurally related components. Therefore, its representation can be derived via the *aggregation* of the representation of its


Component$_1$
w(t$_{11}$) w(t$_{21}$) w(t$_{31}$)
Component$_2$
Component$_3$
w(t$_{12}$) w(t$_{22}$) w(t$_{32}$)
w(t$_{13}$) w(t$_{23}$) w(t$_{33}$)
a
b

Aggregation: retrieval_score' (component$_1$) = f(w(t$_1$), w(t$_2$), w(t$_3$))

Where:
w(t$_1$) = w(t$_{11}$) ⊕ w(t$_{12}$) ⊕ w(t$_{13}$)
w(t$_2$) = w(t$_{21}$) ⊕ w(t$_{22}$) ⊕ w(t$_{23}$)
w(t$_3$) = w(t$_{31}$) ⊕ w(t$_{32}$) ⊕ w(t$_{33}$)

Propagation: retrieval_score' (component$_1$) = retrieval_score (component$_1$) ⊕ retrieval_score (component$_2$) ⊕ retrieval_score (component$_3$)

Where:
retrieval_score (component$_1$) = f(w(t$_{11}$), w(t$_{21}$), w(t$_{31}$))
retrieval_score (component$_2$) = f(w(t$_{12}$), w(t$_{22}$), w(t$_{32}$))
retrieval_score (component$_3$) = f(w(t$_{13}$), w(t$_{23}$), w(t$_{33}$))

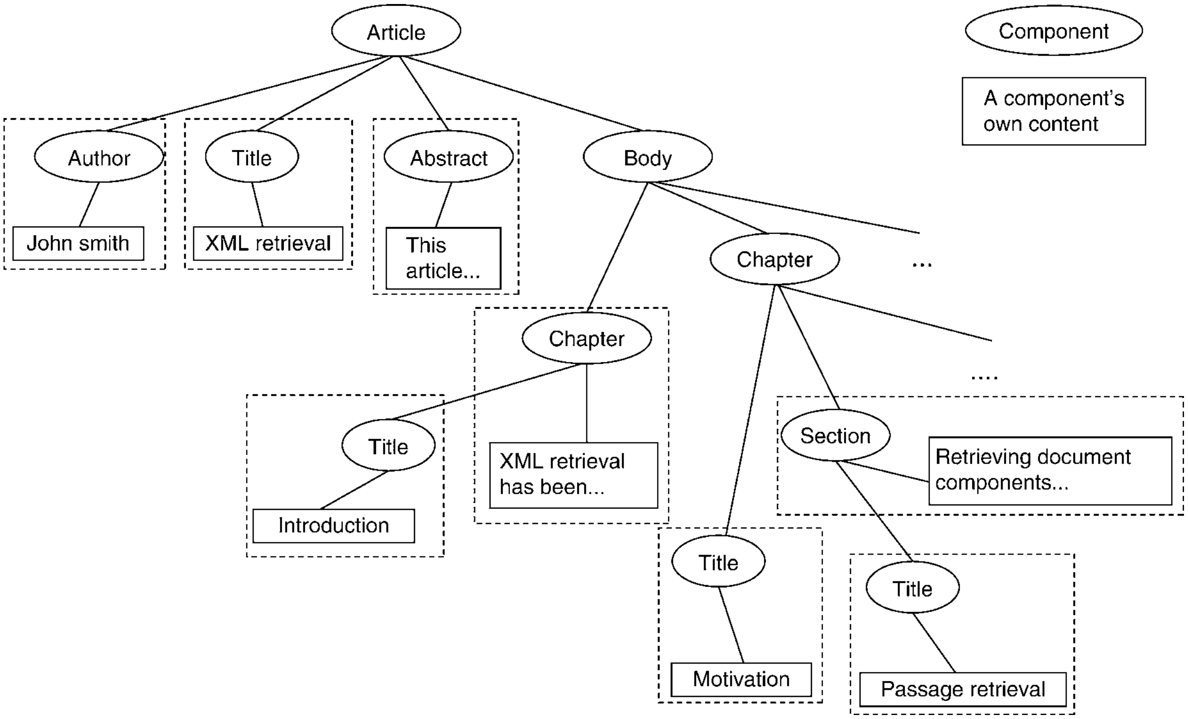
Aggregation-Based Structured Text Retrieval. Figure 2. Simple example illustrating the differences between aggregation- and propagation-based approaches.

own information content with the representations of the information content of its structurally related components; this aggregation takes place at the level of their indexing features. Given the representations of atomic components and of composite components' own information content, aggregation-based approaches recursively generate the aggregated representations of composite components and, based on them, perform retrieval of document components of varying granularity.

In summary, each aggregation-based approach needs to define the following: (i) the representation of each component's own information content, (ii) the aggregated representations of composite components, and (iii) the retrieval function that uses these representations. Although these three steps are clearly interdependent, the major issues addressed in each step need to be outlined first, before proceeding with the description of the key aggregation-based approaches in the field of *structured text retrieval.*

1. *Representing each component's own information content*: In the field of *text retrieval*, the issue of representing documents with indexing features that provide a characterisation of their information content has been extensively studied in the context of several *IR retrieval models* (e.g., Boolean, vector space, probabilistic, language models, etc.). For text documents, these indexing features typically correspond to term statistics. Retrieval functions produce a ranking in response to a user's query, by taking into account the statistics of query terms together with each document's length. The term statistics most commonly used correspond to the *term frequency tf* (t, d) of term t in document d and to the *document frequency df* (t, C) of term t in the document collection C, leading to standard $tf \times idf$ weighting schemes.

Structured text retrieval approaches need to generate representations for all components corresponding to *indexing units*. Since these components are nested, it is not straightforward to adapt these *term statistics* (particularly *document frequency*) at the component level [10]. Aggregation-based approaches, on the other hand, directly generate representations only for components that have their own information content,



Aggregation-Based Structured Text Retrieval. Figure 3. Representing the components that contain their own information.

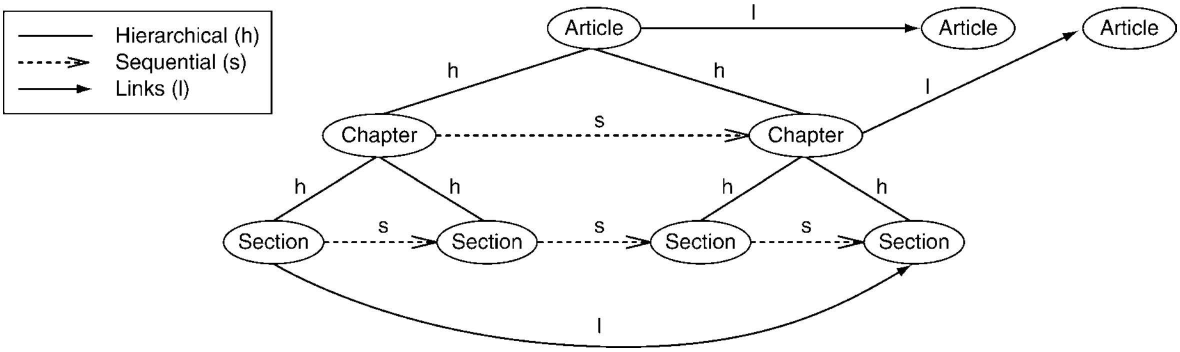
while the representations of the remaining components are obtained via the aggregation process. Therefore, the first step is to generate the representations of atomic components and of the composite components' own information content, i.e., the content not contained in any of their structurally related components. This simplifies the process, since only *disjoint* units need to be represented [6], as illustrated in Figure 3 where the dashed boxes enclose the components to be represented (cf. [5]).

Text retrieval approaches usually consider that the information content of a document corresponds only to its *textual content*, and possibly its metadata (also referred to as *attributes*). In addition to that, *structured text retrieval* approaches also aim at representing the information encoded in the logical structure of documents. Representing this *structural information*, i.e., the interrelations among the documents and their components, enables retrieval in response to both *content-only queries* and *content-and-structure queries.*

Aggregation-based approaches that only represent the textual content typically adapt standard representation formalisms widely employed in *text retrieval* approaches to their requirements for representation at the component level (e.g., [9,11]). Those that consider richer representations of information content apply more expressive formalisms (e.g., various logics [2,4]).

2. *Aggregating the representations*: The concept underlying aggregation-based approaches is that of *augmentation* [4]: the information content of a document component can be *augmented* with that of its structurally related components. Given the already generated representations (i.e., the representations of atomic components and of composite components' own information content), the augmentation of composite components is performed by the aggregation process.

The first step in the aggregation process is the identification of the structurally related components of each composite component. Three basic types of structural relationships (Figure 4) can be distinguished: hierarchical (*h*), sequential (*s*), and links (*l*). Hierarchical connections express the *composition* relationship among components, and induce the tree representing the *logical structure* of a structured



Aggregation-Based Structured Text Retrieval. Figure 4. Different types of structural relationships between the components of a structured text document.

document. Sequential connections capture the order imposed by the document's author(s), whereas links to components of the same or different documents reference (internal or external) sources that offer similar information. In principle, all these types of structural relationships between components can be taken into account by the aggregation process (and some aggregation-based approaches are generic enough to accommodate them, e.g., [7]). In practice, however, the hierarchical structural relations are the only ones usually considered. This leads to the aggregated representations of composite components being recursively generated in an ascending manner.

The next step is to define the *aggregation operator* (or *aggregation function*). Since the aggregation of the textual content of related components is defined at the level of the indexing features of their representations, the aggregation function is highly dependent on the model (formalism) chosen to represent each component's own content. This aggregation results in an (aggregated) representation modeled in the same formalism, and can be seen as being performed at two stages (although these are usually combined into one step): the aggregation of *index expressions* [2] (e.g., terms, conjunctions of terms, etc.), and of the *uncertainty* assigned to them (derived mainly by their statistics).

An aggregation function could also take into account: (i) *augmentation factors* [6], which capture the fact that the textual content of the structurally related components of a composite component is not included in that components own content and has to be "propagated" in order to become part of it, (ii) *accessibility factors* [4], which specify how the representation of a component is influenced by its connected components (a measure of the contribution of, say, a section to its embedding chapter [2]), and (iii) the overall importance of a component in a document's structure [7] (e.g., it can be assumed that a title contains more informative content than a small subsection [13]). Finally, the issue of the possible aggregation of the attributes assigned to related components needs to be addressed [2].

The above aggregation process can take place either at indexing time (*global aggregation*) or at query time (*local aggregation*). Global aggregation is performed for all composite *indexing units* and considers all indexing features involved. Since this strategy does not scale well and can quickly become highly inefficient, local aggregation strategies are primarily used. These restrict the aggregation only to indexing features present in the query (i.e., query terms), and, starting from components retrieved in terms of their own information content, perform the aggregation only for these components' ancestors.

3. *Retrieval*: The retrieval function operates both on the representations of atomic components and on the aggregated representations of composite components. Its definition is highly dependent on the formalism employed in modeling these representations. In conjunction with the definition of the aggregation function, the retrieval function operationalizes the notion of *relevance* for a *structured text retrieval* system. It can, therefore, determine whether retrieval focuses on those document components more specific to the query [2], or whether the aim is to support the users'

browsing activities by identifying each documents best entry points [7] (i.e., document components that contain relevant information which users can browse to further relevant components).

Aggregation-based Approaches

One of the most influential aggregation-based approaches has been developed by Chiaramella et al. [2] in the context of the FERMI project (http://www.dcs.gla.ac.uk/fermi/). Aiming at supporting the integration of IR, hypermedia, and database systems, the FERMI model introduced some of the founding principles of *structured text retrieval* (including the notion of retrieval focussed to the most specific components). It follows the logical view on IR, i.e., it models the retrieval process as inference, and it employs predicate logic as its underlying formalism. The model defines a generic representation of content, attributes, and structural information associated with the *indexing units*. This allows for rich querying capabilities, including support for both *content-only queries* and *content-and-structured queries*. The indexing features of structured text documents can be defined in various ways, e.g., as sets of terms or as logical expressions of terms, while the semantics of the aggregation function depend on this definition. Retrieval can then be performed by a function of the *specificity* of each component with respect to the query.

The major limitation of the FERMI model is that it does not incorporate the uncertainty inherent to the representations of content and structure. To address this issue, Lalmas [8] adapted the FERMI model by using propositional logic as its basis, and extended it by modeling the uncertain representation of the textual content of components (estimated by a *tf* × *idf* weighting scheme) using Dempster-Shafer's theory of evidence. The structural information is not explicitly captured by the formalism; therefore, the model does not provide support for *content-and-structured queries*. The aggregation is performed by Dempster's combination rule, while retrieval is based on the belief values of the query terms.

Fuhr, Gövert, and Rölleke [4] also extended the FERMI model using a combination of (a restricted form of) predicate logic with probabilistic inference. Their model captures the uncertainty in the representations of content, structure, and attributes. Aggregation of index expressions is based on a four-valued logic, allowing for the handling of incomplete information and of inconsistencies arising by the aggregation (e.g., when two components containing contradictory information are aggregated). Aggregation of term weights is performed according to the rules of probability theory, typically by adopting term independence assumptions. This approach introduced the notion of *accessibility factor* being taken into account. Document components are retrieved based on the computed probabilities of query terms occurring in their (aggregated) representations.

Following its initial development in [4], Fuhr and his colleagues investigated further this logic-based probabilistic aggregation model in [5,6]. They experimented with modeling aggregation by different Boolean operators; for instance, they noted that, given terms propagating in the document tree in a bottom-up fashion, a probabilistic-OR function would always result in higher weights for components further up the hierarchy. As this would lead (in contrast to the objectives of *specificity*-oriented retrieval) to the more general components being always retrieved, they introduced the notion of *augmentation factors*. These could be used to "downweight" the weights of terms (estimated by a *tf* × *idf* scheme) that are aggregated in an ascending manner. The effectiveness of their approach has been assessed in the context of the *Initiative for the Evaluation of XML retrieval (INEX)* [6].

Myaeng et al. [11] also developed an aggregation-based approach based on probabilistic inference. They employ Bayesian networks as the underlying formalism for explicitly modeling the (hierarchical) structural relations between components. The document components are represented as nodes in the network and their relations as (directed) edges. They also capture the uncertainty associated with both textual content (again estimated by *tf* × *idf* term statistics) and structure. Aggregation is performed by probabilistic inference, and retrieval is based on the computed beliefs. Although this model allows for document component scoring, in its original publication [11] it is evaluated in the context of *text retrieval* at the document level.

Following the recent widespread application of statistical language models in the field of *text retrieval*, Ogilvie and Callan [8] adapted them to the requirements of *structured text retrieval*. To this end, each document component is modeled by a language model; a unigram language model estimates the probability of a term given some text. For atomic components, the language model is estimated by their own text by employing a maximum likelihood estimate (MLE). For instance, the probability of term *t* given the language model θ_T of text *T* in a component can be estimated by: $P(t|\theta_T) = (1 - \omega)P_{MLE}(t|\theta_T) + \omega P_{MLE}(t|\theta_{collection})$, where ω is a parameter controlling the amount of smoothing of the background collection model. For composite components *comp$_i$*, the aggregation of language models is modeled as a linear interpolation: $P(t|\theta'_{comp_i}) = \lambda^{c'}_{comp_i} P(t|\theta_{comp_i}) + \sum_{j \in childern(comp_i)} \lambda^c_j$ $P(t|\theta_j)$, where $\lambda^{c'}_{comp_i} + \sum_{j \in childern(comp_i)} \lambda^c_j = 1$. These λs model the contribution of each language model (i.e., document component) in the aggregation, while their estimation is a non-trivial issue. Ranking is typically produced by estimating the probability that each component generated the query string (assuming an underlying multinomial model). The major advantage of the language modeling approach is that it provides guidance in performing the aggregation and in estimating the term weights.

A more recent research study has attempted to apply BM25 (one of the most successful *text retrieval* term weighting schemes) to *structured text retrieval*. Robertson et al. [13] initially adapted BM25 to structured text documents with non-hierarchical components (see Figure 1a), while investigating the effectiveness of retrieval at the document level. Next, they [9] adapted BM25 to deal with nested components (see Figure 1b), and evaluated it in the context of the *INitiative for the Evaluation of XML retrieval (INEX)*.

A final note on these aggregation-based approaches is that most aim at focusing retrieval on those document components more specific to the query. However, there are approaches that aim at modeling the criteria determining what constitutes a best entry point. For instance, Kazai et al. [7] model aggregation as a fuzzy formalisation of linguistic quantifiers. This means that an indexing feature (term) is considered in an aggregated representation of a composite component, if it represents *LQ* of its structurally related components, where *LQ* a linguistic quantifier, such as "at least one," "all," "most," etc. By using these aggregated representations, the retrieval function determines that a component is relevant to a query if *LQ* of its structurally related components are relevant, in essence implementing different criteria of what can be regarded as a best entry point.

Key Applications

Aggregation-based approaches can be used in any application requiring retrieval according to the structured text retrieval paradigm. In addition, such approaches are also well suited to the retrieval of multimedia documents. These documents can be viewed as consisting of (disjoint or nested) components each containing one or more media. Aggregation can be performed by considering atomic components to only contain a single medium, leading to retrieval of components of varying granularity. This was recognized early in the field of structured text retrieval and some of the initial aggregation-based approaches, e.g., [2,4], were developed for multimedia environments.

Experimental Results

For most of the presented approaches, particularly for research conducted in the context of the *INitiative for the Evaluation of XML retrieval (INEX)*, there is an accompanying experimental evaluation in the corresponding reference.

Data Sets

A testbed for the evaluation of structured text retrieval approaches has been developed as part of the efforts of the *INitiative for the Evaluation of XML retrieval (INEX)* (http://inex.is.informatik.uni-duisburg.de/).

URL to Code

The aggregation-based approach developed in [8] has been implemented as part of the open source *Lemur* toolkit (for language modeling and IR), available at: http://www.lemurproject.org/.

Cross-references

- ▶ Content-and-Structure Query
- ▶ Content-Only Query
- ▶ Indexing Units
- ▶ Information Retrieval Models
- ▶ INitiative for the Evaluation of XML Retrieval
- ▶ Logical Structure
- ▶ Propagation-based Structured Text Retrieval
- ▶ Relevance
- ▶ Specificity
- ▶ Structured Document Retrieval
- ▶ Text Indexing and Retrieval

Recommended Reading

1. Chiaramella Y. Information retrieval and structured documents. In Lectures on Information Retrieval, Third European Summer-School, Revised Lectures, LNCS, Vol. 1980. M. Agosti, F. Crestani, and G. Pasi (eds.). Springer, 2001, pp. 286–309.
2. Chiaramella Y., Mulhem P., and Fourel F. A model for multimedia information retrieval. Technical Report FERMI, ESPRIT BRA 8134, University of Glasgow, Scotland, 1996.
3. Croft W.B. Combining approaches to information retrieval. In Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval, Vol. 7. W.B. Croft (ed.). The Information Retrieval Series, Kluwer Academic, Dordrecht, 2000, pp. 1–36.
4. Fuhr N., Gövert N., and Rölleke T. DOLORES: A system for logic-based retrieval of multimedia objects. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 257–265.
5. Fuhr N. and Großjohann K. XIRQL: A query language for information retrieval in XML documents. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 172–180.
6. Gövert N., Abolhassani M., Fuhr N., and Großjohann K. Content-oriented XML retrieval with HyREX. In Proc. 1st Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2003, pp. 26–32.
7. Kazai G., Lalmas M., and Rölleke T. A model for the representation and focussed retrieval of structured documents based on fuzzy aggregation. In Proc. 8th Int. Symp. on String Processing and Information Retrieval, 2001, pp. 123–135.
8. Lalmas M. Dempster-Shafer's theory of evidence applied to structured documents: Modelling uncertainty. In Proc. 20th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1997, pp. 110–118.
9. Lu W., Robertson S.E., and MacFarlane A. Field-weighted XML retrieval based on BM25. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, Revised Selected Papers, LNCS, Vol. 3977, Springer, 2006, pp. 161–171.
10. Mass Y. and Mandelbrod M. Retrieving the most relevant XML components. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, pp. 53–58.
11. Myaeng S.-H., Jang D.-H., Kim M.-S., and Zhoo Z.-C. A flexible model for retrieval of SGML documents. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 138–145.
12. Ogilvie P. and Callan J. Hierarchical language models for retrieval of XML components. In Advances in XML Information Retrieval and Evaluation. In Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, Revised Selected Papers, LNCS, Vol. 3493, Springer, 2005, pp. 224–237.
13. Robertson S.E., Zaragoza H., and Taylor M. Simple BM25 extension to multiple weighted fields. In Proc. Int. Conf. on Information and Knowledge Management, 2004, pp. 42–49.
14. Sauvagnat K., Boughanem M., and Chrisment C. Searching XML documents using relevance propagation. In Proc. 11th Int. Symp. on String Processing and Information Retrieval, 2004, pp. 242–254.
15. Wilkinson R. Effective retrieval of structured documents. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1994, pp. 311–317.

AGMS Sketch

► AMS Sketch

Air Indexes for Spatial Databases

Baihua Zheng
Singapore Management University, Singapore, Singapore

Definition

Air indexes refer to indexes employed in wireless broadcast environments to address scalability issue and to facilitate power saving on mobile devices [4]. To retrieve a data object in wireless broadcast systems, a mobile client has to continuously monitor the broadcast channel until the data arrives. This will consume a lot of energy since the client has to remain active during its waiting time. The basic idea of air indexes is that by including index information about the arrival times of data items on the broadcast channel, mobile clients are able to predict the arrivals of their desired data. Thus, they can stay in power saving mode during waiting time and switch to active mode only when the data of their interests arrives.

Historical Background

In spatial databases, clients are assumed to be interested in data objects having spatial features (e.g., hotels, ATM, gas stations). "Find me the nearest restaurant" and "locate all the ATMs that are within 100 miles of my current location" are two examples. A central server is allocated to keep all the data, based on which the queries issued by the clients are answered. There are basically two approaches to disseminating spatial data to clients: (i) *on-demand access*: a mobile client submits a request, which consists of a query and the query's issuing location, to the server. The server returns the result to the mobile client via a dedicated point-to-point channel. (ii) *periodic broadcast*: data are periodically broadcast on a wireless channel open to the public. After a mobile client receives a query from its user, it tunes into the broadcast channel to receive the data of interest based on the query and its current location.

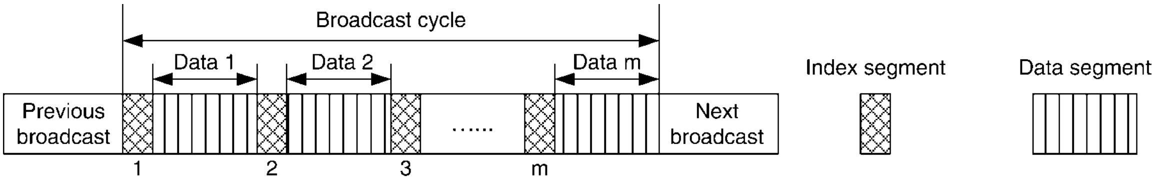
On-demand access is particularly suitable for light-loaded systems when contention for wireless channels and server processing is not severe. However, as the number of users increases, the system performance deteriorates rapidly. Compared with on-demand access, broadcast is a more scalable approach since it allows simultaneous access by an arbitrary number of mobile clients. Meanwhile, clients can access spatial data without reporting to the server their current location and hence the private location information is not disclosed.

In the literature, two performance metrics, namely *access latency* and *tuning time*, are used to measure access efficiency and energy conservation, respectively [4]. The former means the time elapsed between the moment when a query is issued and the moment when it is satisfied, and the latter represents the time a mobile client stays active to receive the requested data. As energy conservation is very critical due to the limited battery capacity on mobile clients, a mobile device typically supports two operation modes: *active mode* and *doze mode*. The device normally operates in active mode; it can switch to doze mode to save energy when the system becomes idle.

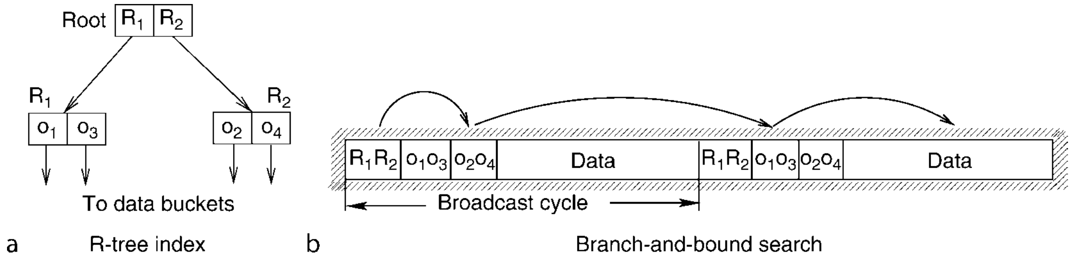
With data broadcast, clients listen to a broadcast channel to retrieve data based on their queries and hence are responsible for query processing. Without any index information, a client has to download all data objects to process spatial search, which will consume a lot of energy since the client needs to remain active during a whole broadcast cycle. A broadcast cycle means the minimal duration within which all the data objects are broadcast at least once. A solution to this problem is *air indexes* [4]. The basic idea is to broadcast an index before data objects (see Fig. 1 for an example). Thus, query processing can be performed over the index instead of actual data objects. As the index is much smaller than the data objects and is selectively accessed to perform a query, the client is expected to download less data (hence incurring less tuning time and energy consumption) to find the answers. The disadvantage of air indexing, however, is that the broadcast cycle is lengthened (to broadcast additional index information). As a result, the access latency would be worsen. It is obvious that the larger the index size, the higher the overhead in access latency.

An important issue in air indexes is how to multiplex data and index on the sequential-access broadcast channel. Figure 1 shows the well-known (1, *m*) scheme [4], where the index is broadcast in front of every 1/m fraction of the dataset. To facilitate the access of index, each data page includes an offset to the beginning of the next index. The general access protocol for processing spatial search involves following three steps: (i) *initial probe*: the client tunes into the broadcast channel and determines when the next index is broadcast; (ii) *index search*: The client tunes into the broadcast channel again when the index is broadcast. It selectively accesses a number of index pages to find out the spatial data object and when to download it; and (iii) *data retrieval*: when the packet containing the qualified object arrives, the client downloads it and retrieves the object.

To disseminate spatial data on wireless channels, well-known spatial indexes (e.g., R-trees) are candidates for air indexes. However, unique characteristics of wireless data broadcast make the adoption of existing spatial indexes inefficient (if not impossible). Specifically, traditional spatial indexes are designed to cluster data objects with spatial locality. They usually assume a resident storage (such as disk and memory) and adopt search strategies that minimize I/O cost. This is achieved by *backtracking* index nodes during search. However, the broadcast order (and thus the access order) of



Air Indexes for Spatial Databases. Figure 1. Air indexes in wireless broadcast environments.



Air Indexes for Spatial Databases. Figure 2. Linear access on wireless broadcast channel.

index nodes is extremely important in wireless broadcast systems because data and index are only available to the client when they are broadcast on air. Clients cannot randomly access a specific data object or index node but have to wait until the next time it is broadcast. As a result, each backtracking operation extends the access latency by one more cycle and hence becomes a constraint in wireless broadcast scenarios.

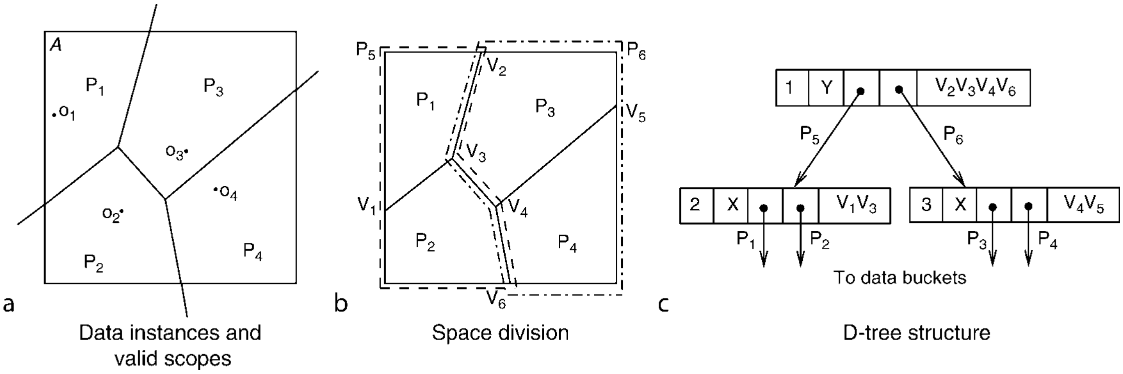
Figure 2 depicts an example of spatial query. Assume that an algorithm based on R-tree first visits root node, then the node R_2, and finally R_1, while the server broadcasts nodes in the order of root, R_1, and R_2. If a client wants to backtrack to node R_1 after it retrieves R_2, it will have to wait until the next cycle because R_1 has already been broadcast. This significantly extends the access latency and it occurs every time a navigation order is different from the broadcast order. As a result, new air indexes which consider both the constraints of the broadcast systems and features of spatial queries are desired.

Foundations

Several air indexes have been recently proposed to support broadcast of spatial data. These studies can be classified into two categories, according to the nature of the queries supported. The first category focuses on retrieving data associated with some specified geographical range, such as "Starbucks Coffee in New York City's Times Square" and "Gas stations along Highway 515." A representative is the index structure designed for *DAYS* project [1]. It proposes a location hierarchy and associates data with locations. The index structure is designed to support query on various types of data with different location granularity. The authors intelligently exploit an important property of the locations, i.e., containment relationship among the objects, to determine the relative location of an object with respect to its parent that contains the object. The containment relationship limits the search range of available data and thus facilitates efficient processing of the supported queries. In brief, a broadcast cycle consists of several sub-cycles, with each containing data belonging to the same type. A major index (one type of index buckets) is placed at the beginning of each sub-cycle. It provides information related to the types of data broadcasted, and enables clients to quickly jump into the right sub-cycle which contains her interested data. Inside a sub-cycle, minor indexes (another type of index buckets) are interleaved with data buckets. Each minor index contains multiple pointers pointing to the data buckets with different locations. Consequently, a search for a data object involves accessing a major index and several minor indexes.

The second category focuses on retrieving data according to specified distance metric, based on client's current location. An example is nearest neighbor (NN) search based on Euclidian distance. According to the index structure, indexes of this category can be further clustered into two groups, i.e., *central tree-based* structure and *distributed* structure. In the following, we review some of the representative indexes of both groups.

D-tree is a paged binary search tree to index a given solution space in support of planar point queries [6]. It assumes a data type has multiple data instances, and each instance has a certain *valid scope* within which this instance is the only correct answer. For example, restaurant is a data type, and each individual restaurant represents an instance. Take NN search as an example, Fig. 3a illustrates four restaurants, namely o_1, o_2, o_3, and o_4, and their corresponding valid scopes p_1, p_2, p_3, and p_4. Given any query location q in, say, p_3, o_3 is the restaurant to which q is nearest. D-tree assumes the valid scopes of different data instances are known and it focuses only on planar point queries which locate the



Air Indexes for Spatial Databases. Figure 3. Index construction using the D-tree.

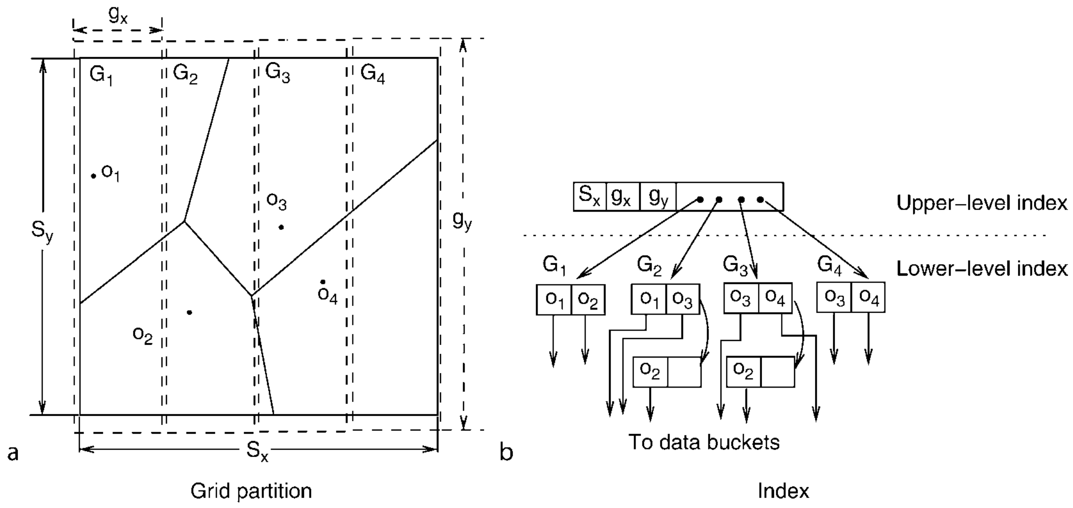
query point into a valid scope and return the client the corresponding data instance.

The D-tree is a binary tree built based on the divisions between data regions (e.g., valid scopes). A space consisting of a set of data regions is recursively partitioned into two complementary subspaces containing about the same number of regions until each subspace has one region only. The partition between two subspaces is represented by one or more polylines. The overall orientation of the partition can be either x-dimensional or y-dimensional, which is obtained, respectively, by sorting the data regions based on their lowest/uppermost y-coordinates, or leftmost/rightmost x-coordinates. Figure 3b shows the partitions for the running example. The polyline $pl(v_2, v_3, v_4, v_6)$ partitions the original space into p_5 and p_6, and polylines $pl(v_1,v_3)$ and $pl(v_4,v_5)$ further partition p_5 into p_1 and p_2, and p_6 into p_3 and p_4, respectively. The first polyline is y-dimensional and the remaining two are x-dimensional. Given a query point q, the search algorithm works as follows. It starts from the root and recursively follows either the left subtree or the right subtree that bounds the query point until a leaf node is reached. The associated data instance is then returned as the final answer.

Grid-partition index is specialized for NN problem [9]. It is motivated by the observation that an object is the NN only to the query points located inside its Voronoi Cell. Let $O = \{o_1, o_2,...,o_n\}$ be a set of points. $V(o_i)$, the *Voronoi cell* (VC) for o_i, is defined as the set of points q in the space such that $dist(q,o_i) < dist(q,o_j)$, $\forall j \neq i$. That is, $V(o_i)$ consists of the set of points for which o_i is the NN. As illustrated in Fig. 3a, p_1, p_2, p_3, and p_4 denote the VCs for four objects, o_1, o_2, o_3, and o_4, respectively. Grid-partition index tries to reduce the search space for a query at the very beginning by partitioning the space into disjoint grid cells. For each grid cell, all the objects that could be NNs of at least one query point inside the grid cell are indexed, i.e., those objects whose VCs overlap with the grid cell are associated with that grid cell.

Figure 4a shows a possible grid partition for the running example, and the index structure is depicted in Fig. 4b. The whole space is divided into four grid cells; i.e., G_1, G_2, G_3, and G_4. Grid cell G_1 is associated with objects o_1 and o_2, since their VCs, p_1 and p_2, overlap with G_1; likewise, grid cell G_2 is associated with objects o_1, o_2, o_3, and so on. If a given query point is in grid cell G_1, the NN can be found among the objects associated with G_1 (i.e., o_1 and o_2), instead of among the whole set of objects. Efficient search algorithms and partition approaches have been proposed to speed up the performance.

Conventional spatial index R-tree has also been adapted to support *k*NN search in broadcast environments [2]. For R-tree index, the *k*NN search algorithm would visit index nodes and objects sequentially as backtracking is not feasible on the broadcast. This certainly results in a considerably long tuning time especially when the result objects are located in later part of the broadcast. However, if clients know that there are at least k objects in the later part of the broadcast that are closer to the query point than the currently found ones, they can safely skip the downloading of the intermediate objects currently located. This observation motivates the design of the enhanced *k*NN search algorithm which caters for the constraints of wireless broadcast. It requires each index node to

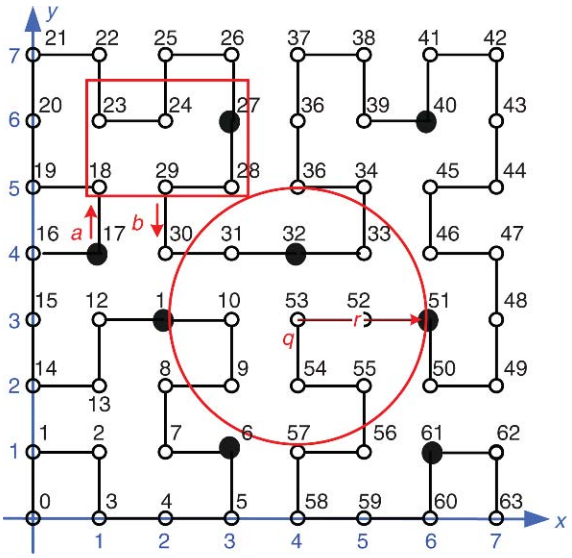


Air Indexes for Spatial Databases. Figure 4. Index construction using the grid-partition.

carry a count of the underlying objects (object count) referenced by the current node. Thus, clients do not blindly download intermediate objects.

Hilbert Curve Index (HCI) is designed to support general spatial queries, including window queries, *k*NN queries, and continuous nearest-neighbor (CNN) queries in wireless broadcast environments. Motivated by the linear streaming property of the wireless data broadcast channel and the optimal spatial locality of the Hilbert Curve (HC), HCI organizes data according to Hilbert Curve order [7,8], and adopts B^+-tree as the index structure. Figure 5 depicts a 8 × 8 grid, with solid dots representing data objects. The numbers next to the data points, namely *index value*, represent the visiting orders of different points at Hilbert Curve. For instance, data point with (1,1) as the coordinates has the index value of 2, and it will be visited before data point with (2,2) as the coordinates because of the smaller index value.

The *filtering and refining* strategy is adopted to answer all the queries. For window query, the basic idea is to decide a candidate set of points along the Hilbert curve which includes all the points within the query window and later to filter out those outside the window. Suppose the rectangle shown in Fig. 5 is a query window. Among all the points within the search range, the first point is point *a* and the last is *b*, sorted according to their occurring orders on the Hilbert curve, and both of them are lying on the boundary of the search range. Therefore, all the points inside this query window should lie on the Hilbert curve segmented by points *a* and *b*. In other words, data points with index values between 18 and 29, but not the others, are the candidates. During the access, the client can derive the coordinates of data points based on the index values and then retrieve those within the query window.



Air Indexes for Spatial Databases. Figure 5. Hilbert curve index.

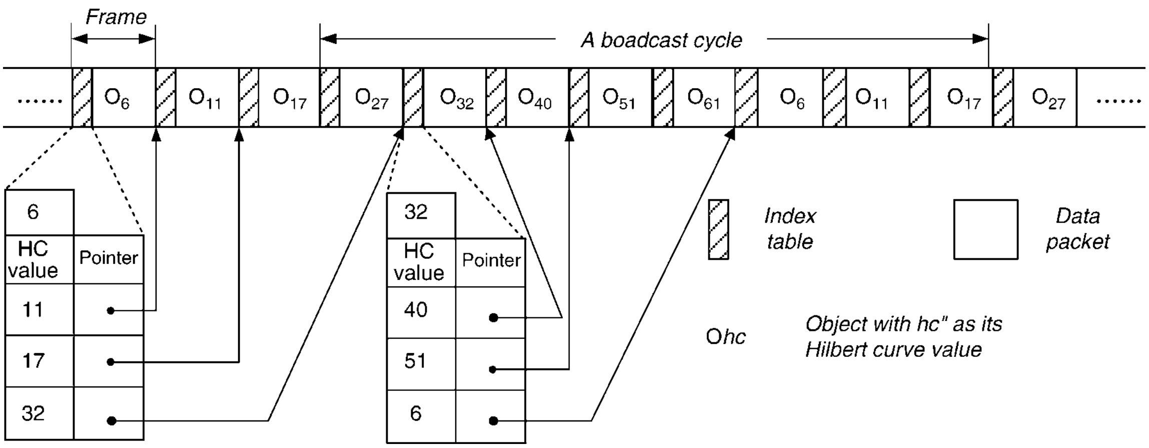
For *k*NN query, the client first retrieves those *k* nearest objects to the query point along the Hilbert curve and then derives a range which for sure bounds at least *k* objects. In the filtering phase, a window query which bounds the search range is issued to filter out those unqualified. Later in the refinement phase, *k* nearest objects are identified according to their distance to the query point. Suppose an NN query at point *q* (i.e., index value 53) is issued. First, the client finds its nearest neighbor (i.e., point with index value 51) along the curve and derives a circle centered at *q* with *r* as the radius (i.e., the green circle depicted in Fig. 5). Since the circle bounds point 51, it is certain to contain the nearest neighbor to point *q*. Second, a window query is issued to retrieve all the data points inside the circle, i.e., points with index values 11, 32, and 51. Finally, the point 32 is identified as the nearest neighbor. The search algorithm for CNN adopts a similar approach. It approximates a search range which is guaranteed to bound all the answer objects, issues a window query to retrieve all the objects inside the search range, and finally filters out those unqualified.

All the indexes mentioned above are based on a central tree-based structure, like R-tree and B-tree. However, employing a tree-based index on a linear broadcast channel to support spatial queries results in several deficiencies. First, clients can only start the search when they retrieve the root node in the channel. Replicating the index tree in multiple places in the broadcast channel provides multiple search starting points, shortening the initial root-probing time. However, a prolonged broadcast cycle leads to a long access latency experienced by the clients. Second, wireless broadcast media is not error-free. In case of losing intermediate nodes during the search process, the clients are forced to either restart the search upon an upcoming root node or scan the subsequential broadcast for other possible nodes in order to resume the search, thus extending the tuning time. *Distributed spatial index (DSI)*, a fully distributed spatial index structure, is motivated by these observations [5]. A similar distributed structure was proposed in [3] as well to support access to spatial data on air.

DSI is very different from tree-based indexes, and is not a hierarchical structure. Index information of spatial objects is fully distributed in DSI, instead of simply replicated in the broadcast. With DSI, the clients do not need to wait for a root node to start the search. The search process launches immediately after a client tunes into the broadcast channel and hence the initial probe time for index information is minimized. Furthermore, in the event of data loss, clients resume the search quickly.

Like HCI, DSI also adopts Hilbert curve to determine broadcast order of data objects. Data objects, mapped to point locations in a 2-D space, are broadcast in the ascending order of their HC index values. Suppose there are *N* objects in total, DSI chunks them into n_F *frames*, with each having n_o objects ($n_F = \lceil N/n_o \rceil$). The space covered by Hilbert Curve shown in Fig. 5 is used as a running example, with solid dots representing the locations of data objects (i.e., $N = 8$). Figure 6 demonstrates a DSI structure with n_o set to 1, i.e., each frame contains only one object.

In addition to objects, each frame also has an index table as its header, which maintains information



Air Indexes for Spatial Databases. Figure 6. Distributed spatial index.

regarding to the HC values of data objects to be broadcast with specific waiting interval from the current frame. This waiting interval can be denoted by delivery time difference or number of data frames apart, with respect to the current frame. Every index table keeps n_i entries, each of which, τ_j, is expressed in the form of $\langle HC'_j, P_j \rangle$, $j \in [0, n_i)$. P_j is a pointer to the r^j-th frame after the current frame, where r (> 1) is an exponential base (i.e., a system-wide parameter), and HC'_j is the HC value of the first object inside the frame pointed by P_j. In addition to τ_j, an index table also keeps the HC values HC_k ($k \in [1, n_o]$) of all the objects obj_k that are contained in the current frame. This extra information, although occupying litter extra bandwidth, can provide a more precise image of all the objects inside current frame. During the retrieval, a client can compare HC_ks of the objects against the one she has interest in, so the retrieval of unnecessary object whose size is much larger than an HC value can be avoided.

Refer to the example shown in Fig. 5, with corresponding DSI depicted in Fig. 6. Suppose $r = 2$, $n_o = 1$, $n_F = 8$, and $n_i = 3$. The index tables corresponding to frames of data objects O_6 and O_{32} are shown in the figure. Take the index table for frame O_6 as an example: τ_0 contains a pointer to the next upcoming (2^0-th) frame whose first object's HC value is 11, τ_1 contains a pointer to the second (2^1-th) frame with HC value for the first object (the only object) 17, and the last entry τ_2 points to the fourth (2^2-th) frame. It also keeps the HC value 6 of the object O_6 in the current frame. Search algorithm for window queries and kNN searches are proposed.

Key Applications

Location-based Service

Wireless broadcast systems, because of the scalability, provide an alternative to disseminate location-based information to a large number of users. Efficient air indexes enable clients to selectively tune into the channel and hence the power consumption is reduced.

Moving Objects Monitoring

Many moving objects monitoring applications are interested in finding out all the objects that currently satisfy certain conditions specified by the users. In many cases, the number of moving objects is much larger than the number of submitted queries. As a result, wireless broadcast provides an ideal way to deliver subscribed queries to the objects, and those objects that might affect the queries can then report their current locations.

Cross-references

- ▶ Nearest Neighbor Query
- ▶ Space-Filling Curves for Query Processing
- ▶ Spatial Indexing Techniques
- ▶ Voronoi Diagrams

Recommended Reading

1. Acharya D. and Kumar V. Location based indexing scheme for days. In Proc. 4th ACM Int. Workshop on Data Eng. for Wireless and Mobile Access, 2005, pp. 17–24.
2. Gedik B., Singh A., and Liu L. Energy efficient exact knn search in wireless broadcast environments. In Proc. 12th ACM Int. Symp. on Geographic Inf. Syst., 2004, pp. 137–146.
3. Im S., Song M., and Hwang C. An error-resilient cell-based distributed index for location-based wireless broadcast services. In Proc. 5th ACM Int. Workshop on Data Eng. for Wireless and Mobile Access, 2006, pp. 59–66.
4. Imielinski T., Viswanathan S., and Badrinath B.R. Data on air – organization and access. IEEE Trans. Knowl. Data Eng., 9(3):1997.
5. Lee W.-C. and Zheng B. Dsi: a fully distributed spatial index for wireless data broadcast. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2005, pp. 349– 358.
6. Xu J., Zheng B., Lee W.-C., and Lee D.L. The d-tree: an index structure for location-dependent data in wireless services. IEEE Trans. Knowl. Data Eng., 16(12):1526–1542, 2002.
7. Zheng B., Lee W.-C., and Lee D.L. Spatial queries in wireless broadcast systems. ACM/Kluwer J. Wireless Networks, 10 (6):723–736, 2004.
8. Zheng B., Lee W.-C., and Lee D.L. On searching continuous k nearest neighbors in wireless data broadcast systems. IEEE Trans. Mobile Comput., 6(7):748–761, 2007.
9. Zheng B., Xu J., Lee W.-C., and Lee L. Grid-partition index: a hybrid method for nearest-neighbor queries in wireless location-based services. VLDB J., 15(1):21–39, 2006.

AJAX

Alex Wun
University of Toronto, Toronto, ON, Canada

Definition

AJAX is an acronym for "Asynchronous JavaScript and XML" and refers to a collection of web development technologies used together to create highly dynamic web applications.

Key Points

AJAX does not refer to a specific technology, but instead refers to a collection of technologies used in conjunction to develop dynamic and interactive web applications. The two main technologies comprising AJAX are the JavaScript scripting language and the W3C open standard XMLHttpRequest object API. While the use of XML and DOM are important for standardized data representation, using neither XML nor DOM is required for an application to be considered AJAX-enabled since the XMLHttpRequest API actually supports any text format.

Using the XMLHttpRequest API, web applications can fetch data asynchronously while registering a callback function to be invoked once the fetched data is available. More concretely, the XMLHttpRequest object issues a standard HTTP POST or GET request to a web server but returns control to the calling application immediately after issuing the request. The calling application is then free to continue execution while the HTTP request is being handled on the server. When the HTTP response is received, the XMLHttpRequest object calls back into the function that was supplied by the calling application so that the response can be processed. The asynchronous callback model used in AJAX applications is analogous to the Operating System technique of using interrupt handlers to avoid blocking on I/O. As such, development using AJAX necessarily requires an understanding of multi-threaded programming.

There are three main benefits to using AJAX in web applications:

1. *Performance*: Since XMLHttpRequest calls are asynchronous, client-side scripts can continue execution after issuing a request without being blocked by potentially lengthy data transfers. Consequently, web pages can be easily populated with data fetched in small increments in the background.
2. *Interactivity*: By maintaining long-lived data transfer requests, an application can closely approximate real-time event-driven behavior without resorting to periodic polling, which can only be as responsive as the polling frequency.
3. *Data Composition*: Web applications can easily pull data from multiple sources for aggregation and processing on the client-side without any dependence on HTML form elements. Data composition is also facilitated by having data adhere to standard XML and DOM formats.

The functionality provided by AJAX allows web applications to appear and behave much more like traditional desktop applications. The main difference is that data consumed by the application resides primarily out on the Internet – one of the concepts behind applications that are labeled as being representative "Web 2.0" applications.

Cross-references

▶ JavaScript
▶ MashUp
▶ Web 2.0/3.0
▶ XML

Recommended Reading

1. The Document Object Model: W3C Working Draft. Available at: http://www.w3.org/DOM/
2. The XMLHttpRequest Object: W3C Working Draft. Available at: http://www.w3.org/TR/XMLHttpRequest/

Allen's Relations

Peter Revesz[1], Paolo Terenziani[2]
[1]University of Nebraska-Lincoln, Lincoln, NE, USA
[2]University of Turin, Turin, Italy

Synonyms

Qualitative relations between time intervals; Qualitative temporal constraints between time intervals

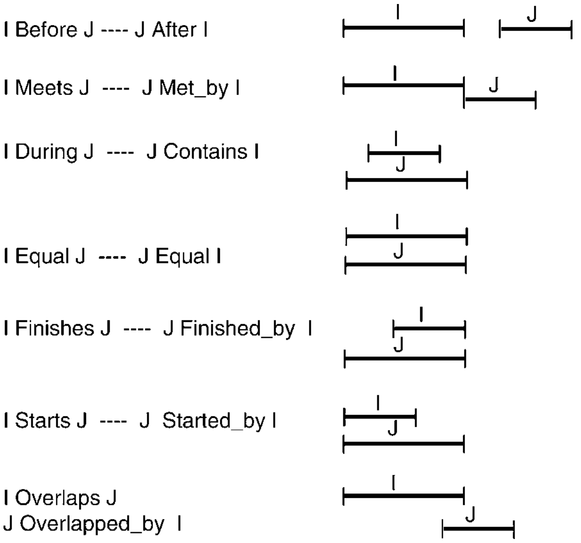
Definition

A (convex) *time interval* I is the set of all time points between a starting point (usually denoted by I^-) and an ending point (I^+). Allen's relations model all possible relative positions between two time intervals [1]. There are 13 different possibilities, depending on the relative positions of the endpoints of the intervals (Table 1).

For example, "There will be a guest speaker during the Database System class" can be represented by Allen's relation I_{Guest} *During* $I_{Database}$ (or by $I^-_{\text{Guest}} > I^-_{\text{Database}} \wedge I^+_{\text{Guest}} < I^+_{\text{Database}}$ considering the relative

Allen's Relations. Table 1. Translation of Allen's interval relations between two intervals I and J into conjunctions of point relations between I−, I+, J−, J+.

	I^-J^-	I^-J^+	I^+J^-	I^+J^+
After		>		
Before			<	
Meets			=	
Met_by		=		
During	>			<
Contains	<			>
Equal	=			=
Finishes	>			=
Finished_by	<			=
Starts	=			<
Started_by	=			>
Overlaps	<		>	<
Overlapped_by	>	<		>


I Before J ---- J After I
I
J
I Meets J ---- J Met_by I
I
J
I During J ---- J Contains I
I
J
I Equal J ---- J Equal I
I
J
I Finishes J ---- J Finished_by I
I
J
I Starts J ---- J Started_by I
I
J
I Overlaps J
J Overlapped_by I
I
J

Allen's Relations. Figure 1. Visualization of Allen's interval relations.

position of the endpoints. Moreover, any subset of the 13 relations, excluding the empty subset, is a relation in Allen's Interval Algebra (therefore, there are 2^{13}-1 relations in Allen's Algebra). Such subsets are used in order to denote ambiguous cases, in which the relative position of two intervals is only partially known. For instance, I_1 (Before, Meets, Overlaps) I_2 represents the fact that I_1 is before *or* meets *or* overlaps I_2.

Key Points

In many cases, the exact time interval when facts occur is not known, but (possibly imprecise) information on the relative temporal location of facts is available. Allen's relations allow one to represent such cases of *temporal indeterminacy*. For instance, planning in Artificial Intelligence is the first application of Allen's relations. A graphical representation of the basic 13 Allen's relations is shown in Fig. 1.

Allen's relations are specific cases of *temporal constraints*. Namely, they are qualitative temporal constraints between time intervals. Given a set of such constraints, *qualitative temporal reasoning* can be used in order to make inferences (e.g., to check whether the set of constraints is consistent).

Finally, notice that, in many entries of this Encyclopedia, the term *(time) period* has been used with the same meaning of *(time) interval* in this entry.

Cross-references

▶ Qualitative Temporal Reasoning
▶ Temporal Constraints
▶ Temporal Indeterminacy

Recommended Reading

1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832–843, 1983.

AMOSQL

Peter M. D. Gray
University of Aberdeen, Aberdeen, UK

Definition

AMOSQL [2] is a functional language having its roots in the functional query languages OSQL [1] and DAPLEX [5] with extensions of mediation primitives, multi-directional foreign functions, late binding, *active rules*, etc. Queries are specified using the select–from–where construct as in SQL. Furthermore, AMOSQL has aggregation operators, nested subqueries, disjunctive queries, quantifiers, and is relationally complete.

AMOSQL is a functional query language operating within the environment of Amos II , which is an open, light-weight, and extensible database management system (DBMS) with a functional data model. Each Amos II server contains all the traditional database facilities, such as a storage manager, a recovery manager, a transaction manager, and a query language. The system can be used as a single-user database or as a multi-user server to applications and to other Amos II peers. It has mainly been used for experiments with Mediators [4].

Key Points

AMOSQL is often used within a distributed mediator system where several mediator peers communicate over the Internet. Functional views provide transparent access to data sources from clients and other mediator peers. Amos II mediators are composable since a mediator peer can regard other mediator peers as data sources. In AMOSQL [2] the top-level language uses function application within an SQL-like syntax, for example to list children of a particular parent who like sailing:

```
create function sailch(person p) ->
string as
select name(c) from person c
where parent(c) = p and hobby(c) =
'sailing';
```

This is turned internally into a typed object comprehension.

```
sailch(p) == [name(c) | c <- person; par-
ent(c)=p; hobby(c) = 'sailing']
```

Some functions may be defined as views on other databases accessed through mediators. In the comprehension form, such functions may easily be substituted because of referential transparency, leading to a longer conjunction with extra clauses [3].

Suppose information about `hobby(c)` was held in connection with `sports-person`, a subclass of person with a `surname` attribute:

```
hobby(c) == [recreation(x) | x <- sports-
person; surname(x) = name(c)]
```

These comprehensions merge into the following which can be further simplified:

```
sailch(p) == [name(c) | c <- person; par-
ent(c)=p; x <- sportsperson;
surname(x) = name(c); recreation(x) =
'sailing']
```

Note that the internal form of comprehension does not explicitly distinguish the use of generators but simply use equality as in a filter. The conceptual advantage of this is that generators are not always a fixed role and some optimizations may reverse the role of filter and generator (systems with explicit generators use rewrite rules to do this).

Cross-references

► Comprehensions
► Functional Query Language

Recommended Reading

1. Beech D. A foundation of evolution from relational to object databases. In Advances in Database Technology, Proc. 1st Int. Conf. on Extending Database Technology, 1988, pp 251–270.
2. Fahl G., Risch T., and Sköld M. 1AMOS – an architecture for active mediators. In Proc. Workshop on Next Generation Information Technologies and Systems, 1993.
3. Josifovski V. and Risch T. Functional query optimization over object-oriented views for data integration. J. Intell. Inf. Syst., 12 (2–3):165–190, 1999.
4. Risch T., Josifovski V., and Katchaounov. T. Functional Data Integration in a Distributed Mediator System. In The Functional Approach to Data Management, chapter 9, P.M.D. Gray, L. Kerschberg, P.J.H. King, and A. Poulovassilis (eds.). Springer, Berlin Heidelberg New York, 2004.
5. Shipman D.W. The functional data model and the data language DAPLEX. ACM Trans. Database Syst., 6(1):140–173, 1981.

AMS Sketch

ALIN DOBRA
University of Florida, Gainesville FL, USA

Synonyms

AGMS sketch; Sketch; Tug-of-war sketch

Definition

AMS sketches are randomized summaries of the data that can be used to compute aggregates such as the second frequency moment (the self-join size) and sizes of joins. AMS sketches can be viewed as random projections of the data in the frequency domain on ± 1 pseudo-random vectors. The key property of AMS sketches is that the product of projections on the same random vector of frequencies of the join attribute of two relations is an unbiased estimate of the size of join of the relations. While a single AMS sketch is inaccurate, multiple such sketches can be computed and combined using averages and medians to obtain an estimate of any desired precision.

Historical Background

The AMS sketches were introduced in 1996 by Noga Alon, Yossi Matias, and Mario Szegedy as part of a suit of randomized algorithms for approximate computation of frequency moments. The same authors, together with Phillip Gibbons, extended the second frequency moment application of AMS sketches to the computation of the size of join of two relations, a more relevant database application. The initial work on AMS sketches fostered a large amount of subsequent work on data streaming algorithms including generalizations and extensions of AMS sketches. Alon, Matias, and Szegedy received the Gödel Prize in 2005 for their work on AMS sketches.

Foundations

While the AMS sketches were initially introduced to compute the second frequency moment, since the reader might be more familiar with database terminology, the problem of estimating the size of join of two relations will be considered here instead. Notice that the size of the self join size of a relation coincides with the second frequency moment of the relation thus the treatment here is slightly more general but not more complicated.

Problem Setup

To set up the problem, assume access is provided to two relations F and G each with a single attribute a. Since it is convenient, denote with f_i and g_i the frequency of value i of attribute a in relation F and G, respectively. Assume that elements of F and G are streamed the result of the query: COUNT$(F \bowtie_a G)$ needs to be computed or estimated. Consider the following example:

$$\text{Stream } F\colon \boxed{\begin{array}{c|cccccc} \text{a} & 1 & 1 & 2 & 3 & 1 & 3 \end{array}},$$

$$\text{frequency vector } \mathbf{f}\colon \begin{array}{c|ccc} i & 1 & 2 & 3 \\ \hline f_i & 3 & 1 & 2 \end{array}$$

$$\text{Stream } G\colon \boxed{\begin{array}{c|ccccc} \text{a} & 3 & 1 & 3 & 1 & 1 \end{array}},$$

$$\text{frequency vector } \mathbf{g}\colon \begin{array}{c|ccc} i & 1 & 2 & 3 \\ \hline g_i & 3 & 0 & 2 \end{array}$$

Elements of F and G are assumed to arrive one by one (i.e., are streamed). If the frequency vectors **f** and **g** can be maintained, than the result of the query COUNT $(F \bowtie\ _aG)$ can be computed clearly in the following example:

$$\begin{aligned} \text{COUNT}(F \bowtie_\alpha G) &= \mathbf{f}\mathbf{g}^T \\ &= [3 \quad 1 \quad 2][3 \quad 0 \quad 2]^T \\ &= 3 \cdot 3 + 1 \cdot 0 + 2 \cdot 2 \\ &= 13 \end{aligned}$$

Observe that the size of join can be written as the dot product of the frequency vectors of the two relations. Expressing the size of the join in terms of frequencies of the join attribute is key for AMS sketch based approximation.

Main Idea

Assume now that the estimate COUNT$(F \bowtie\ _aG)$ needs to be computed but only less than linear space, in terms of the size of the frequency vectors, is available. As it turns out, exact computation is not possible with less space (in an asymptotic sense), but approximate computation is possible. The AMS sketches prove that they allow the approximation of the size of join using sub-linear space.

The main idea behind AMS sketches is to summarize the entire frequency table by projecting it on a random vector. The value thus obtained will be referred to as an elementary sketch. Then, use the two elementary sketches, one for each relation, to recover approximately the result of the query. Interestingly, a random vector $\xi = [\xi_1 \ldots \xi_n]$ of ± 1 values suffices to obtain projections with the desired properties. For simplicity, random vectors for which $\forall i$, $E[\xi_i] = 0$ are preferred. With this:

Sketch of F, $X_F = \text{f}\xi^T$

- Sketch of G, $X_G = \text{g}\xi^T$
- $X = X_F X_G$ estimates COUNT$(F \bowtie_a G)$ since

$$E[X] = E[\mathbf{f}\xi^T\xi\mathbf{g}^T] = \mathbf{f}E[\xi^T\xi]\mathbf{g}^T = \mathbf{f}I\mathbf{g}^T = \mathbf{f}\mathbf{g}^T$$

if $E[\xi^T \xi] = I$. To ensure this, property distinct elements of ξ must be pair-wise independent, i.e., $\forall i \neq i', \ \xi_i^2 = 1, \ E[\xi_i \xi_{i'}] = 0$

For the particular random vector $\xi = [\xi_1 \ \ \xi_2 \ \ \xi_3] = [-1 + 1 - 1]$, the value of the elementary sketches and the overall estimate will be:

$$X_F = \mathbf{f}\xi^T = -4$$
$$X_G = \mathbf{g}\xi^T = -5$$
$$X = X_F X_G = (-4)(-5) = 20 \approx 13$$

The error of the estimate X is due to its variance that can be shown to have the property:

$$\mathrm{Var(X)} \leq 2\mathbf{ff}^{\mathrm{T}}\mathbf{gg}^{\mathrm{T}} = 2 \ \mathrm{SJ(F)} \ \mathrm{SJ(G)}$$

as long as the random vector ξ is 4-wise independent, i.e., $\forall i_1 \neq i_2 \neq i_3 \neq i_4, \ E[\xi_{i_1} \xi_{i_2}] = 0, \ E[\xi_{i_1} \xi_{i_2} \xi_{i_3} \xi_{i_4}] = 0$

Since a higher degree of independence would not make the sketch more precise, 4-wise independence suffices. This is important since 4-wise independent ± 1 random vectors can be generated *on the fly* by combining a small seed s and the index of the entry using $\xi_i(s) = h(s,i)$ with h a special hash function that guarantees the 4-wise independence of the components of ξ. The fact that elements of ξ can be generated on the fly is important since space can be saved and, more importantly, because sketches X_F and X_G can be computed using constant storage. This is how this can be accomplished using the previous example:

$$X_F = \mathbf{f}\xi^T = \sum_i f_i \xi_i$$
$$= \sum_{t \in F} \xi_{t.a} = \xi_1 + \xi_1 + \xi_2 + \xi_3 + \xi_1 + \xi_3$$
$$= h(s,1) + h(s,1) + h(s,2) + h(s,3) + h(s,1) + h(s,3)$$

$$X_G = \mathbf{g}\xi^T = \sum_i g_i \xi_i$$
$$= \sum_{t \in G} \xi_{t.a} = \xi_3 + \xi_1 + \xi_3 + \xi_1 + \xi_1$$
$$= h(s,3) + h(s,1) + h(s,3) + h(s,1) + h(s,1)$$

From this example, it can be observed that, to maintain the elementary sketches over the streams F and G, the only operation needed is to increment X_F and X_G by the value of $\xi_{t.a}$ using the function $h(\cdot)$ and the seed s where $t.a$ is the value of attribute a of the current tuple t arriving on the data stream. The fact that the elementary sketches can be computed so easily by considering one element at the time in an arbitrary order is what makes the AMS sketches appealing as an approximation technique.

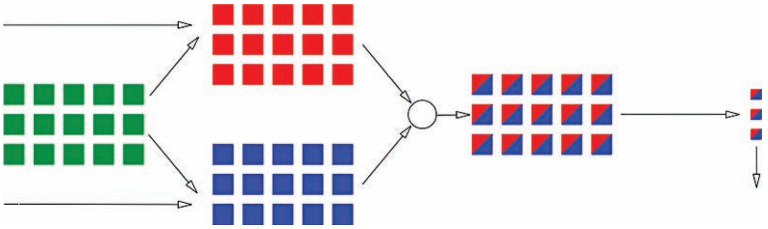
Improving the Basic Schema

Since the streams F and G are summarized by a single number X_F and X_G, respectively, it is not expected that the estimate will be very precise (this is suggested as well by the above example). In order to improve the accuracy of X, a standard technique in randomized algorithms can be used (i.e., generating multiple independent copies of random variable X). Copies of X are averaged in order to decrease the variance (thus the error). The median of such averaged values of X is used to estimate $\mathrm{COUNT}(F \bowtie_a G)$ since medians improve confidence. Multiple copies of X can be obtained using multiple seeds as depicted in Fig. 1.

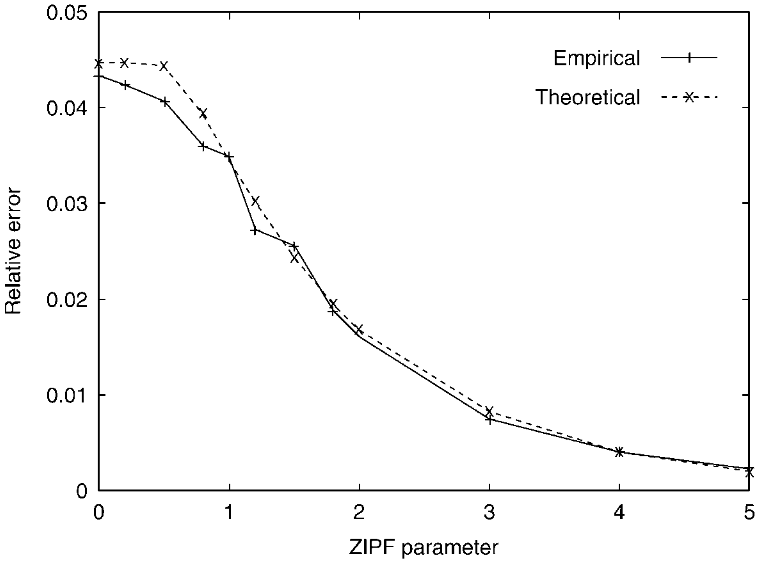
It can be shown that:

Average $\frac{8\mathrm{Var(X)}}{\epsilon^2 E^2[X]}$ independent copies of X to reduce error to $\in$

- Median of 2 log $1/\delta$ such averages increases the confidence to $1 - \delta$



AMS Sketch. Figure 1. Combining elementary sketches to estimate $\mathrm{COUNT}(F \bowtie_a G)$ with relative error at most $\in$ with probability at least $1 - \delta$.



AMS Sketch. Figure 2. Relative error of AMS sketches as a function of Zipf coefficient.

Key Applications

AMS sketches are particularly well suited for computing aggregates when data is either streamed (or a single pass over the data is allowed/desirable) or distributed at multiple sites. Thus, AMS sketches are relevant for processing large amount of data, as is the case in data warehousing, or processing distributed/streaming data, as is the case for computing networking statistics.

Experimental Results

To get an understanding of how the AMS sketches perform in the problem of estimating the self join size of a relation, consider the following setup. The domain of the attribute on which the self join size is computed is set to 16,384. The seize of the relation is fixed at 100,000 tuples. The distribution of the frequencies of join attribute values are generated according to a Zipf distribution with a varying Zipf coefficient. The number of medians is set to 1 (no median computation) and the number of elementary sketches averaged is set to 1,024.

The relative error, both theoretical and empirical, of AMS sketches is depicted in Fig. 2. The following observations confirm the intuition based on theory for the behavior of AMS sketches: (i) on the self join size problem the error is acceptable for sketches of size in the order of 2,000 words, (ii) the error decreases somewhat as the skew increases, and (iii) the theoretical prediction follows entirely the empirical behavior.

URL to Code

http://www.cs.rutgers.edu/~muthu/massdal-code-index.html http://www.cise.ufl.edu/~adobra/AQP/code.html

Cross-references

▶ Approximate Query Processing
▶ Data Stream

Recommended Reading

1. Alon N., Gibbons P.B., Matias Y., and Szegedy M. Tracking join and self-join sizes in limited storage. J. Comput. Syst. Sci., 64 (3):719–747, 2002.
2. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. In Proc. 28th Annual ACM Symp. on Theory of Computing, 1996, pp. 20–29.
3. Charikar M., Chen K., and Farach-Colton M. Finding frequent items in data streams. In Proc. 29th Int. Colloquium on Automata, Languages and Programming, 2002, pp. 693–703.
4. Cormode G. and Garofalakis M. Sketching streams through the net: distributed approximate query tracking. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 13–24.
5. Das A., Gehrke J., and Riedewald M. Approximation techniques for spatial data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 695–706.
6. Dobra A., Garofalakis M., Gehrke J., and Rastogi R. Processing complex aggregate queries over data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 61–72.
7. Rusu F. and Dobra A. Pseudo-random number generation for sketch-based estimations. ACM Trans. Database Syst., 32(2):11, 2007.
8. Rusu F. and Dobra A. Statistical Analysis of Sketch Estimators. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 187–198.

Analogy

▶ Visual Metaphor

Anchor

▶ Anchor Text

Anchor Text

Vassilis Plachouras
Yahoo! Research, Barcelona, Spain

Synonyms

Anchor; Anchor text surrogate

Definition

Anchor text is the text associated with a hyperlink pointing from one Web document to another. Anchor text provides a concise description of a document, not necessarily written by the author of the document. It is very effective in Web search retrieval tasks, and in particular, in tasks where the aim is to find the home page of a given website.

Key Points

Hyperlink analysis algorithms explicitly employ the hyperlinks between Web documents to find high quality or authoritative Web documents. A form of implicit use of the hyperlinks in combination with content analysis is the use anchor text associated with the incoming hyperlinks of documents. Web documents can be represented by an anchor text surrogate, which is formed by collecting the anchor text associated with the hyperlinks pointing to the document.

The anchor text of the incoming hyperlinks provides a concise description for a Web document. The used terms in the anchor text may be different from the ones which occur in the document itself, because the author of the anchor text is not necessarily the author of the document. Eiron and McCurley [2] found similarities in the distribution of terms between the anchor text of Web documents and the queries submitted to an intranet search engine by users. Similarities were also found in the use of abbreviations and technical terms.

Craswell et al. [1] show that anchor text is effective for navigational search tasks and more specifically for finding home pages of Web sites. They report that searching for home pages of websites using an index of anchor text performs better than using an index of the document contents. Upstill et al. [3] also suggest that the anchor text of the incoming hyperlinks from documents outside a corpus of documents enhances the retrieval effectiveness for homepage finding.

Cross-references

▶ Document Links and Hyperlinks
▶ Field-Based Information Retrieval Models

Recommended Reading

1. Craswell N., Hawking D., and Robertson S. Effective site finding using link anchor information. In Proc. 24th Annu. Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 250–257.
2. Eiron N. and McCurley K.S. Analysis of anchor text for web search. In Proc. 26th Annu. Int. ACM SIGIR Conf. on Research and Development in Informaion Retrieval, 2003, pp. 459–460.
3. Upstill T., Craswell N., and Hawking D. Query-independent evidence in home page finding. ACM Trans. Inform. Syst., 21(3):286–313, 2003.

Anchor Text Surrogate

▶ Anchor Text

AND-Join

▶ Workflow Join

AND-Split

▶ Split

Animation

▶ Dynamic Graphics

Annotation

Amarnath Gupta
University of California, San Diego, La Jolla, CA, USA

Definition

An annotation is any form of additional information "superposed" on any existing data or document.

Example: If a scientist records her experimental data in a relational database and then marks some "cells" of a table with the comment "consistent with previous findings," this additionally "marked" information is an annotation.

Key Points

Often annotations are not originally intended to be part of the collected data, and hence no data or schema structure was designed to hold it. Annotating data is a very common practice in science, where scientists would literally "mark" experimental observation with comments, and often use annotations to share their opinions in a collaborative study. As larger scale experiments are conducted and larger collaborations are formed, management of the annotated data becomes a serious challenge. In recent times, the emerging importance of annotation in scientific data management has been recognized by the Information Management community, leading to a variety of research in annotation management.

Cross-references

▶ Annotation-Based Image Retrieval
▶ Biomedical Scientific Textual Data Types and Processing
▶ Provenance

Recommended Reading

1. Bhagwat D., Chiticariu L., Tan W.C., and Vijayvargiya G. An annotation management system for relational databases. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 900–911.
2. Buneman P., Khanna S., and Tan W.-C. On propagation of deletions and annotations through views. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 150–158.
3. Geerts F., Kementsiesidis A., and Milano D. MONDRIAN: annotating and querying databases through colors and blocks. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 82.
4. Gertz M. and Sattler K.-U. Integrating scientific data through external, concept-based annotations. In Proc. Workshop on Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web, LNCS, Vol. 2590, Springer, 2002, pp. 220–240.
5. Murthy S., Maier D., and Delcambre L.M.L. Querying bi-level information. In Proc. 7th Int. Workshop on the World Wide Web and Databases, 2004, pp. 7–12.
6. Srivastava D. and Velegrakis Y. Intensional associations between data and metadata. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 401–412.

Annotation-based Image Retrieval

Xin-Jing Wang, Lei Zhang
Microsoft Research Asia, Beijing, China

Synonyms

Semantic image retrieval; Text-based image retrieval; Tag-based image search; Tag-based image retrieval

Definition

Given (i) a textual query, and (ii) a set of images and their annotations (phrases or keywords), annotation-based image retrieval systems retrieve images according to the matching score of the query and the corresponding annotations. There are three levels of queries according to Eakins [7]:

- Level 1: Retrieval by primitive features such as color, texture, shape or the spatial location of image elements, typically querying by an example, i.e., "find pictures like this."
- Level 2: Retrieval by derived features, with some degree of logical inference. For example, "find a picture of a flower."
- Level 3: Retrieval by abstract attributes, involving a significant amount of high-level reasoning about the purpose of the objects or scenes depicted. This includes retrieval of named events, of pictures with emotional or religious significance, etc., e.g., "find pictures of a joyful crowd."

Together, levels 2 and 3 are referred to as semantic image retrieval, which can also be regarded as annotation-based image retrieval.

Historical BackGround

There are two frameworks of image retrieval [6]: annotation-based (or more popularly, text-based) and content-based. The annotation-based approach can be tracked back to the 1970s. In such systems, the images are manually annotated by text descriptors, which are used by a database management system (DBMS) to perform image retrieval. There are two disadvantages with this approach. The first is that a considerable level of human labor is required for manual annotation. The second is that because of the subjectivity of human perception, the manually labeled annotations may not converge. To overcome the above disadvantages, content-based image retrieval (CBIR) was introduced in the early 1980s. In CBIR, images are indexed by their visual content, such as color, texture, shapes. In the past decade, several commercial products and experimental prototype systems were developed, such as QBIC, Photobook, Virage, VisualSEEK, Netra, SIMPLIcity. Comprehensive surveys in CBIR can be found in [7,8].

However, the discrepancy between the limited descriptive power of low-level image features and the richness of user semantics, which is referred to as the "semantic gap" bounds the performance of CBIR. On the other hand, due to the explosive growth of visual data (both online and offline) and the phenomenal success in Web search, there has been increasing expectation for image search technologies. Because of these reasons, the main challenge of image retrieval is understanding media by bridging the semantic gap between the bit stream and the visual content interpretation by humans [3]. Hence, the focus is on automatic image annotation techniques.

Foundations

The state-of-the-art image auto-annotation techniques include four main categories [3,6]: (i) using machine learning tools to map low-level features to concepts, (ii) exploring the relations among image content and the textual terms in the associated metadata, (iii) generating semantic template (ST) to support high-level image retrieval, (iv) making use of both the visual content of images and the textual information obtained from the Web to learn the annotations.

Machine Learning Approaches

A typical approach is using Support Vector Machine (SVM) as a discriminative classifier over image low-level features. Though straightforward, it has been shown effective in detecting a number of visual concepts.

Recently there is a surge of interest in leveraging and handling relational data, e.g., images and their surrounding texts. Blei et al. [1] extends the Latent Dirichlet Allocation (LDA) model to the mix of words and images and proposed a Correlation LDA model. This model assumes that there is a hidden layer of topics, which are a set of latent factors and obey the Dirichlet distribution, and words and regions are conditionally independent on the topics, i.e., generated by the topics. This work used 7,000 Corel photos and a vocabulary of 168 words for annotation.

Relation Exploring Approaches

Another notable direction for annotating image visual content is exploring the relations among image content and the textual terms in the associated metadata. Such metadata are abundant, but are often incomplete and noisy. By exploring the co-occurrence relations among the images and the words, the initial labels may be filtered and propagated from initial labeled images to additional relevant ones in the same collection [3].

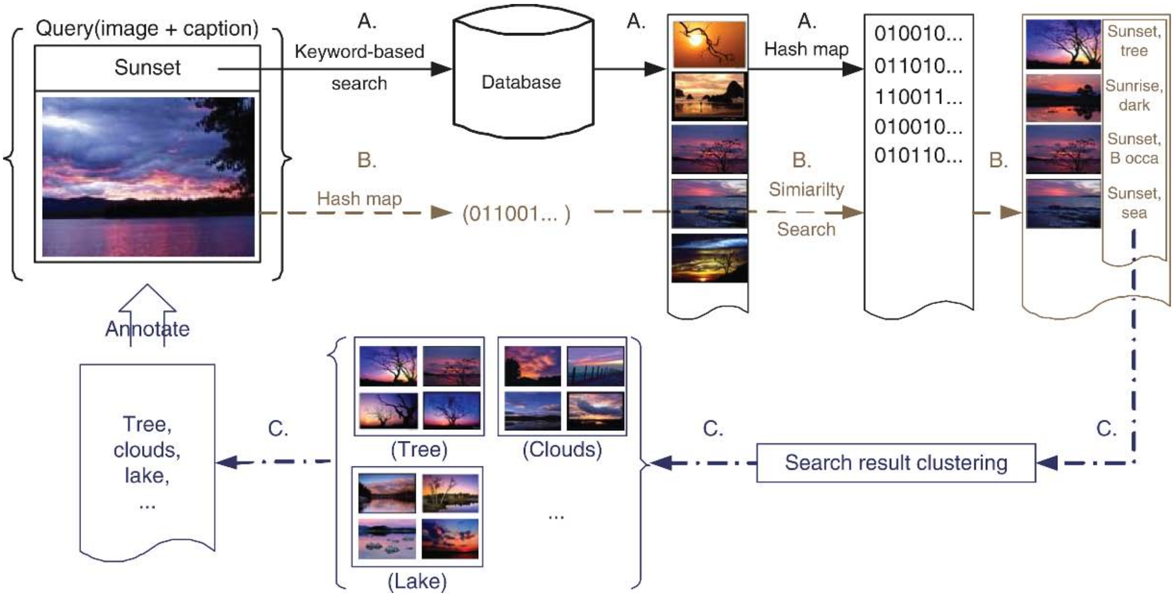
Jeon et al. [5] proposed a cross-media relevance model to learn the joint probabilistic distributions of the words and the visual tokens in each image, which are then used to estimate the likelihood of detecting a specific semantic concept in a new image.

Semantic Template Approaches

Though it is not yet widely used in the above mentioned techniques, Semantic Template (ST) is a promising approach in annotation-based image retrieval (a map between high-level concept and low-level visual features).

Chang and Chen [2] show a typical example of ST, in which a visual template is a set of icons or example scenes/objects denoting a personalized view of concepts such as meetings, sunset. The generation of a ST is based on user definition. For a concept, the objects, their spatial and temporal constraints, and the weights of each feature of each object are specified. This initial query scenario is provided to the system, and then through the interaction with users, the system finally converges to a small set of exemplar queries that "best" match (maximize the recall) the concept in the user's mind.

In contrast, Zhuang et al. [10] generates ST automatically in the process of Relevance Feedback, whose basic idea is to refine retrieval outputs based on interactions with the user. A semantic lexicon called WordNet is used in this system to construct a network of ST. During the retrieval process, once the user submits a query concept (keyword), the system can find a corresponding ST, and thus target similar images.



Annotation-based Image Retrieval. Figure 1. Framework of the search-based annotation system.

Large-Scale Web Data Supported Approaches

Good scalability to a large set of concepts is required in ensuring the practicability of image annotation. On the other hand, images from the Web repositories, e.g., Web search engines or photo sharing sites, come with free but less reliable labels. In [9], a novel search-based annotation framework was proposed to explore such Web-based resources. Fundamentally, it is to automatically expand the text labels of an image of interest, using its initial keyword and image content.

The process of [9] is shown in Fig. 1. It contains three stages: the text-based search stage, the content-based search stage, and the annotation learning stage, which are differentiated using different colors (black, brown, blue) and labels (A., B., C.). When a user submits a query image as well as a query keyword, the system first uses the keyword to search a large-scale Web image database (2.4 million images crawled from several Web photo forums), in which images are associated with meaningful but noisy descriptions, as tagged by “A.” in Fig. 1. The intention of this step is to select a semantically relevant image subset from the original pool. Visual feature-based search is then applied to further filter the subset and save only those visually similar images (the path labeled by “B.” in Fig. 1). By these means, a group of image search results which are both semantically and visually similar to the query image are obtained. To speed-up the visual feature-based search procedure, a hash encoding algorithm is adopted to map the visual features into hash codes, by which inverted indexing technique in text retrieval area can be applied for fast retrieval. At last, based on the search results, the system collects their associated textual descriptions and applies the Search Result Clustering (SRC) algorithm to group the images into clusters. The reason of using SRC algorithm is that (i) it is proved to be significantly effective in grouping documents semantically; and (ii) more attractively, it is capable of learning a name for each cluster that best represents the common topics of a clusters member documents. By ranking these clusters according to a ranking function, and setting a certain threshold, the system selects a group of clusters and merges their names as the final learnt annotations for the query image, which ends the entire process (C. in Fig. 1).

Key Applications

Due to the explosive growth of visual data (both online and offline), effective annotation-based image search becomes a highly-expected technique which facilitate human’s lives.

Cross-references

- ▶ Cross-Modal Multimedia Information Retrieval
- ▶ Hash-Based Indexing
- ▶ Image Querying
- ▶ Image Retrieval
- ▶ Indexing and Similarity Search

- ▶ Information Information Retrieval
- ▶ Object Recognition
- ▶ Multimedia Information Retrieval
- ▶ Web Search and Crawling

Recommended Reading

1. Blei D. and Jordan M.I. Modeling Annotated Data. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 127–134.
2. Chang S.-F., Chen W., and Sundaram H. Semantic Visual Templates: Linking Visual Features to Semantics. In Proc. Int. Conf. on Image Processing, Vol. 3. 1998, pp. 531–534.
3. Chang S.-F., Ma W.-Y., and Smeulders A. Recent Advances and Challenges of Semantic Image/Video Search. In Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, 2007, pp. 1205–1208.
4. Eakins J. and Graham M. Content-based image retrieval, Technical Report, University of Northumbria at Newcastle, 1999.
5. Jeon J., Lavrenko V., and Manmatha R. Automatic Image Annotation and Retrieval Using Cross-Media Relevance Models, In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 119–126.
6. Liu Y., Zhang D., Lu G., and Ma W.-Y. A survey of content-based image retrieval with high-level semantics. Pattern Recognit., 40(1):262 –282, 2007.
7. Long F., Zhang H.J., and Feng D.D. Fundamentals of content-based image retrieval. In Multimedia Information Retrieval and Management, D. Feng (eds.). Springer, 2003.
8. Rui Y., Huang T.S., and Chang S.-F. Image retrieval: current techniques, promising directions, and open issues, J. Visual Commun. Image Represent. 10(4):39–62, 1999.
9. Wang X.-J., Zhang L., Jing F., and Ma W.-Y. AnnoSearch: Image Auto-Annotation by Search. In Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition, 2006, pp. 1483–1490.
10. Zhuang Y., Liu X., and Pan Y. Apply Semantic Template to Support Content-based Image Retrieval. In Proc. SPIE, Storage and Retrieval for Media Databases, vol. 3972, December 1999, pp. 442–449.

Anomaly Detection on Streams

Spiros Papadimitriou
IBM T.J. Watson Research Center, Hawthorne,
NY, USA

Definition

Anomaly detection generally refers to the process of automatically detecting events or behaviors which deviate from those considered normal. It is an unsupervised process, and can thus detect anomalies which have not been previously encountered. It is based on estimating a model of typical behavior from past observations and consequently comparing current observations against this model. It can be performed either on a single stream or among multiple streams. Anomaly detection encompasses outlier detection as well as change detection and therefore is closely related to forecasting and clustering methods.

Historical Background

Anomaly detection in streams has close connections to traditional outlier detection, as well as to change detection. The former is a common and widely studied topic in statistics [11]. The latter emerged in the context of statistical monitoring and control for continuous processes and the widely used CUSUM algorithm was proposed as early as 1954 [9]. With the emergence of data stream management systems, anomaly detection in this setting has received significant attention, with applications in network management and intrusion detection, environmental monitoring, and surveillance, to mention a few.

Foundations

Anomaly detection is closely related to outlier detection and change detection. After a review of the main ideas, the streaming case is presented.

Outlier Detection

The existing approaches to outlier detection can be broadly classified into the following categories. Typically, outlier detection relies on a model for the data. Model parameters are estimated based on appropriately chosen historical data. As new observations arrive, they are either compared directly against the model and are declared outliers if the fit is poor. Alternatively, a second set of model parameters may be estimated from recent observations. If there is a statistically significant difference among the two sets of parameters, the new observations are declared as outliers.

Clustering-Based and Forecasting-Based Approaches

Many clustering and forecasting algorithms detect outliers as by-products. However, not all clustering or forecasting procedures can be easily turned into outlier detection procedures.

Distribution-Based Approaches Methods in this category are typically found in statistics textbooks. They deploy some standard distribution model (e.g., Gaussian) and flag as outliers those objects which deviate from the model. These work well in many occasions, but may be unsuitable for high-dimensional data sets, or when reasonable assumptions about the distribution of data points cannot be made.

Distance-Based and Density-Based Approaches A point in a data set is a distance-based outlier if at least a fraction β of all other points are further than r from it. This outlier definition is based on a single, global criterion determined by the parameters r and β. This can lead to problems when the data set has both dense and sparse regions. Density-based approaches aim to remedy this problem, by relying on the local density of each point's neighborhood.

Change Detection

Sequential hypothesis testing and sequential change detection arose out of problems in statistical process control. Assume a collected sequence of observations, modeled as random variables $X_1, X_2,..., X_t,...$ Additionally, assume that X_t are drawn from a distribution with parameter θ and that a test of whether the true parameter is θ_0 or θ_1 is desired.

The Sequential Likelihood Ratio Test (SLRT) relies on the logarithm of likelihood ratios $z_t := \log(p(x_t;\ \theta_0)/p(x_t;\ \theta_1))$ and tests the cumulative sum $z_1 + ... + z_t$ to decide upon the true parameter.

This can be extended to other settings, such as detecting changes in other distribution parameters. For example, in its simplest form, CUSUM tests for a shift in the mean by essentially applying SLRT, assuming points independently drawn from a Gaussian distribution with known variance. Many other versions have appeared since the CUSUM test was first proposed [9], relaxing or modifying some of these assumptions.

In general, change detection is closely related to outlier detection; in fact, change detection may also be viewed as outlier detection along the time axis.

Streaming Algorithms

In a streaming setting, there are two key challenges that need to be addressed:

1. *Limited resources.* In a streaming setting, a large number of observations arrives over time and the total volume of data grows indefinitely. However processing and storage capacity are limited, in comparison to the amount of data. Therefore, data summarization or *sketching* techniques need to be applied, in order to extract a few, relevant features from the raw data.
2. *Concept drift.* In an indefinitely growing collection of observations, changes in the underlying features (e.g., distribution parameters) may not necessarily correspond to anomalies, but rather be part of normal changes in the behavior of the system. Thus, mechanisms to handle such non-stationarity or *concept drift* and adapt to changing behavior are necessary [12].

Next, several of the approaches that have been studied in the literature are reviewed.

Sketching techniques In the past several years, a number of techniques for *sketch* or *synopsis* construction have appeared, with applications to many stream processing problems. Some examples include CM sketches, AMS sketches, FM sketches, and Bloom filters [2]. Other summarization techniques specifically for data clustering on streams have appeared, such as those in [1,4], which can be easily extended for outlier detection on streams.

Burst Detection In many applications, the appearance of sudden bursts in the data often signifies an anomaly. For example, in a network monitoring application, a burst in the traffic volume to a particular destination may signify a denial of service (DoS) attack. Thus, *burst detection* on streams has received significant attention. Examples of such work include [7] and [14].

Correlation Dnalysis Often a collection of multiple streams is available and measurements from different streams may be highly correlated with each other. If the strength of correlations changes over time [13] or the number of correlated components varies [10], this often signifies changes in the underlying data-generating process that may be due to anomalies.

Change Analysis More generally, detecting significant changes has been studied in the context of stream processing [3].

Key Applications

Intrusion Detection With the widespread adoption of the internet, various forms of malware (e.g., viruses, worms, trojans, and botnets) have become a serious and costly issue. Most intrusion detection systems (IDS) rely on known signatures to identify malicious payloads or behaviors. However, there are several efforts underway for automatic detection of suspicious activity on the fly, as well as for automating the signature extraction process.

System Monitoring Maintenance costs for large computer clusters or networks is traditionally labor-intensive and contributes a large fraction of total cost of ownership. Hence, autonomic computing initiatives aim at automating this process. An important first step is the automatic, unsupervised detection of abnormal events (e.g., node or link failures) based on continuously collected system metrics. Streaming anomaly detection methods are used to address this problem.

Process Control Applications in quality control and industrial process control have traditionally provided much of the impetus for the development of change detection methods. Machinery used in a production chain (e.g., food preparation or chip fabrication) typically monitor a large number of process parameters at each step. Early detection of sudden changes in those parameters is important to identify potential flaws in the process which can severely affect end product quality.

Pervasive Healthcare Small and cheap sensors which can continuously monitor patient physiological data (e.g., temperature, blood pressure, heart rate, ECG measurements, glucose levels, etc.) are becoming widely available. Anomaly detection methods can prove essential in enabling early diagnosis of potential life-threatening conditions, as well as preventive healthcare.

Civil Infrastructure Early detection of faults by continuously monitoring civil infrastructure components (e.g., bridges, buildings, and roadways) can reduce maintenance costs and increase safety. Similarly, surveillance systems on urban environments rely on anomaly detection methods to spot suspicious activities and increase security.

Future Directions

Certain anomalies can be detected only by taking into account information collected from a large number of different sources. Even if data ownership issues are resolved, collecting all this information at a central site is often infeasible due to its large volume. A number of efforts have tackled this problem in the past few years, but much remains to be done, especially as the scale of information collected increases. Also related to this trend is anomaly detection on more complex data, such as time-evolving graphs.

Cross-references

▶ Change Detection
▶ Clustering
▶ Forecasting
▶ Outlier Detection

Recommended Reading

1. Aggarwal C.C., Han J., Wang J., and Yu P.S. A Framework for clustering evolving data streams. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 81–92.
2. Aggarwal C.C. and Yu P.S. A survey of synopsis construction in data streams. In Data Streams: Models and Algorithms. Springer, 2007.
3. Cormode G. and Muthukrishnan S. What's new: finding significant differences in network data streams. IEEE/ACM Trans. Netw., 13(6):1219–1232, 2005.
4. Guha S., Meyerson A., Mishra N., Motwani R., and O'Callaghan L. Clustering data streams: theory and practice. IEEE Trans. Knowl. Data Eng., 15(3):515–528, 2003.
5. Hulten G., Spencer L., and Domingos P. Mining time-changing data streams. In Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2001, pp. 97–106.
6. Jain A.K., Narasimha Murty M. and Flynn P.J. Data clustering: a review. ACM Comput. Surv., 31(3):264–323, 1999.
7. Kleinberg J. Bursty and hierarchical structure in streams. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 91–101.
8. Lee W., Stolfo S.J., and Mok K.W. Adaptive intrusion detection: a data mining approach. Artif. Intell. Rev. 14(6):533–567, 2000.
9. Page E.S. Continuous inspection schemes. Biometrika, 41(1):100–115, 1954.
10. Papadimitriou S., Sun J., and Faloutsos C. Streaming pattern discovery in multiple time-series. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 697–708.
11. Peter J.R. and Annick M.L. Robust Regression and Outlier Detection. Wiley, New York, 1987.
12. Wang H., Fan W., Yu P.S., and Han J. Mining concept-drifting data streams using ensemble classifiers. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 226–235.
13. Zhu Y. and Shasha D. StatStream: statistical monitoring of thousands of data streams in real time. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 358–369.
14. Zhu Y. and Shasha D. Efficient elastic burst detection in data streams. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 336–345.

Anonymity

Simone Fischer-Hübner
Karlstad University, Karlstad, Sweden

Synonyms

Namelessness; Nonidentifiability

Definition

The term anonymity originates from the Greek word “anonymia,” which means “without a name.”

In the context of computing, anonymity has been defined in [2] as follows: “Anonymity of a subject means that the subject is not identifiable within a set of subjects, the anonymity set.” The anonymity set is the set of all possible subjects, e.g., the set of all possible senders of a message or the set of all possible recipients of a message (dependent on the knowledge of an attacker). *Sender anonymity* means that a message cannot be linked to the sender, while *receiver anonymity* implies that a certain message cannot be linked to the receiver of that message. Relationship anonymity of a sender and recipient means that even though a sender and a recipient can be identified as participating in some communication, they cannot be identified as communicating with each other, i.e., sender and recipient are unlinkable. The definition above corresponds to the definition of anonymity in [1]: “Anonymity of a user means that the user may use a resource or service without disclosing the user’s identity.”

To reflect the possibility to quantify anonymity, a slightly modified definition has also provided by [2]: “Anonymity of a subject from the attacker’s perspective means that the attacker cannot sufficiently identify the subject within a set of subjects, the anonymity set.”

Data protection legislation usually defines data as anonymous if information concerning personal or material circumstances can no longer (or with disappropriate amount of time, expense, and labor) be attributed to an individual.

Key Points

Providing anonymity is the best strategy for achieving privacy. If individuals can act anonymously and if their data are kept in an anonymous form, their privacy is not affected (and consequently, data protection legislation is not applicable).

For providing anonymity, however, it has to be guaranteed that potential attackers cannot or not sufficiently identify the individuals. Proposals for measuring the degree of anonymity are usually based on Shannon’s entropy.

Cross-references

▶ Privacy
▶ Privacy-Enhancing Technologies
▶ Privacy Metrics

Recommended Reading

1. Common Criteria Project, Common criteria for information technology security evaluation, Version 3.1, Part 2: Security functional requirements, September 2006, www.commoncriteriaportal.org
2. Pfitzmann A. and Hansen M. “Anonymity, unlinkability, unobservability, pseudonymity, and identity management – a consolidated proposal for terminology,” Version 0.29, http://dud.inf.tu-dresden.de/Anon_Terminology.shtml, July, 2007.

Anonymity in Location-based Services

▶ Spatial Anonymity

ANSI/INCITS RBAC Standard

YUE ZHANG, JAMES B. D. JOSHI
University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

RBAC standard

Definition

The ANSI/INCITS RBAC standard includes the *core*, *hierarchical*, and the *constraint RBAC* models [1]. The *core RBAC* includes the following entities:

- $USERS$, $ROLES$, OPS, and OBS are the sets of users, roles, operations and objects, respectively.
- $UA \subseteq USERS \times ROLES$ is a many-to-many mapping from $USERS$ to $ROLES$.
- $assigned\_users$: $(r{:}ROLES) \rightarrow 2^{USERS}$, is mapping of role r onto a set of users. Formally: $assigned\_users\ (r) = \{u \epsilon USERS \mid (u, r)\ \epsilon\ UA\}$
- $PRMS = 2^{(OBS\ \times\ OPS)}$ is the set of permissions.
- $PA \subseteq PERMS \times ROLES$ is a many-to-many mapping from $PERMISSIONS$ to $ROLES$.
- $assigned\_permissions(r{:}ROLES) \rightarrow 2^{\mathrm{PRMS}}$, is mapping of role r onto a set of permissions. Formally: $assigned\_permissions(r) = \{p \epsilon PRMS \mid (p, r)\ \epsilon\ PA\}$
- $Op(p{:}\ PRMS) \rightarrow \{op \subseteq OPS\}$, is the permission to operation mapping.
- $Ob(p{:}\ PRMS) \rightarrow \{ob \subseteq OBS\}$, is the permission to object mapping.
- $SESSIONS$ = the set of sessions
- $session\_users\ (s{:}SESSIONS) \rightarrow USERS$ is mapping of session s onto the corresponding user.

– *session_roles* (*s*:*SESSIONS*) → 2^{ROLES} is mapping of session *s* onto a set of roles.

– *avail_session_perms*(*s*:*SESSIONS*) → 2^{PRMS} gives the permissions available to a user in a session =

$$\left[\bigcup_{r\in session\_roles(s)} assigned\_permissions(r)\right]$$

The *hierarchical RBAC* model extends the core RBAC model with the *general* and *restricted* role hierarchies. The *general role hierarchy* is defined below:

- $RH \subseteq ROLES \times ROLES$ is a partial order on *ROLES*, written as ≥, where $r_1 \geq r_2$ only if all permissions of r_2 are also permissions of r_1, and all users of r_1 are also users of r_2.
- *authorized_users*(*r*: *ROLES*) → 2^{USERS}, is mapping of role *r* onto a set of users in the presence of a role hierarchy. Formally: $authorized\_users(r) = \{u \in USERS \mid r' \geq r, (u, r') \in UA\}$.
- *authorized_permissions*(*r*: *ROLES*) → 2^{PRMS}, is mapping of role *r* onto a set of permissions in the presence of a role hierarchy. Formally: $authorized\_permissions(r) = \{p \in PRMS \mid r' \geq r, (p, r') \in PA\}$

The *restricted hierarchy* is similar to the *general role hierarchies* with the following limitation:

$$\forall r,\ r_1,\ r_2 \in ROLES, \quad r \geq r_1 \wedge r \ \geq r_2 \Rightarrow r_1 = r_2$$

The *constraint RBAC* model extends the hierarchical RBAC with *Static Separation of Duty* (SSoD) and *Dynamic SoD* (DSoD) constraints.

SSoD Constraint: $SSD \subseteq (2^{ROLES} \times N)$ is a collection of pairs (*rs*, *n*) in *Static Separation of Duty*, where each *rs* is a role set, and *n* is a natural number ≥ 2, with the property that no user is assigned to *n* or more roles from the set *rs* in each (*rs*, *n*) *SSD*.

DSoD Constraint: $DSD \subseteq (2^{ROLES} \times N)$ is collection of pairs (*rs*, *n*) in *Dynamic Separation of Duty*, where each *rs* is a role set and *n* is a natural number ≥2, with the property that no subject may activate *n* or more roles from the set *rs* in each *dsd DSD*.

Key Points

The ANSI/INCTIS RBAC standard essentially evolved from the original RBAC96 model [2]. Several advanced features of RBAC such as cardinality constraints, hybrid hierarchy, much fine-grained SoD constraints, features related to the administration of RBAC, etc., are not covered in the standard. The link (http://csrc.nist.gov/groups/SNS/rbac/standards.html) includes the most updated information on ANSI RBAC.

Cross-references

▶ Role Based Access Control

Recommended Reading

1. ANSI. American national standard for information technology – role based access control. ANSI INCITS, 359–2004, February 2004.
2. Sandhu R.S., Coyne E.J., Feinstein H.L., and Youman C.E. Role-based access control models. IEEE Comput., 29(2):38–47, 1996.

Answering Queries Using Views

Vasilis Vassalos
Athens University of Economics and Business, Athens, Greece

Synonyms

Query rewriting using views

Definition

Answering queries using views refers to a data management problem and the set of related techniques, algorithms and other results to address it. The basic formulation of the problem is the following: Given a query *Q* over a database schema Σ, expressed in a query language L_Q and a set of views $V_1, V_2,..., V_n$ over the same schema, expressed in a query language L_V, is it possible to answer the query *Q* using (only) the views $V_1, V_2,..., V_n$?

The problem has a number of related formulations: What is the *maximal set* of tuples in the answer of *Q* that can be obtained from the views? If it is possible to access both the views and the database relations, what is the cheapest query execution plan for answering *Q*?

From the above, it is clear that this is more generally a family of problems. Problems of different complexity, often admitting different (or no) solutions and solution techniques, result from making choices about various "parameters" of the original problem. For example, one can choose the languages L_Q and L_V, the language L_R used to answer *Q* from the views, the ability to access the relations of the schema Σ, the possible existence of constraints, e.g., equality-generating or tuple-generating dependencies, as well as the semantics of the views (sound, complete, or both).

The problem is also referred to as query rewriting using views.

Historical Background

The problem of putting together information from multiple sources has a relatively long history within the database community. In the 1970s, distributed databases offered a controlled solution to the problem: the data are structured with a single schema and are put under the control of a single, albeit distributed, database system. In the early 1980s, the challenges of integrating full-fledged relational databases was studied in the context of multidatabases. An important direction of multidatabase research was static schema integration, i.e., creating in advance a single new schema with as much of the information of the original schemas as possible. Query processing could then be performed on the integrated schema. "Reusing" the original database tables also received some attention in the multidatabase context. In the early 1990s, techniques were developed in order to make use of existing materialized views to speed up the processing of queries that did not mention them explicitly. These techniques were precursors to the more general techniques developed a few years later for the problem of answering queries using views. The problem was cast in its current form in a seminal paper by Levy, Mendelzon, Sagiv, and Srivastava in 1995.

Foundations

Preliminaries

To fully specify the problem of answering queries using views one needs to decide on the languages used for the queries, the views and, in some cases, the rewritings. This entry presents the basic case of answering *conjunctive queries* using *conjunctive views*, possibly in the presence of constraints. The relevant definitions are presented next. Techniques developed for different languages or data models, such as bag queries (and views), XML queries, queries with aggregates, etc., can be found in the Recommended Reading list. Moreover, detailed presentations of the techniques discussed in the article, including exceptions, optimizations, and other technical issues, can be found in the research articles on the Recommended Reading list.

Conjunctive Queries A *conjunctive* query can be represented as:

$$q(\bar{X}) \leftarrow s_1(\bar{X}_1) \wedge \ldots \wedge s_n(\bar{X}_n)$$

where q and $s_1,\ldots,s_n$ are predicate names. In general, $s_1,\ldots,s_n$ refer to database relations. The atoms $s_1(\bar{X}_1),\ldots,$ $s_n(\bar{X}_n)$ that appear in the body of the query are called *subgoals* of the query, The atom $q(\bar{X})$ is called the *head* of the query and defines the answer relation. The tuples $\bar{X}$, $\bar{X}_1,\ldots,\bar{X}_n$ contain either variables or constants. The variables in $\bar{X}$ are the *distinguished* variables of the query, the rest of the variables are the *existential* variables. An important condition for a conjunctive query is *safety*: A query is safe if $\bar{X} \subseteq \bar{X}_1 \cup \ldots \cup \bar{X}_n$, i.e., every distinguished variable must also appear in a query subgoal.

Use $Vars(Q)$ and $Subg(Q)$ to refer to the set of variables (and constants) in Q and subgoals of Q respectively. $Q(D)$ refers to the result of evaluating the query Q over the database D.

Views A *view* is a query whose head defines a new database relation. If this relation is not stored, the view is called a *virtual* view. If the results of executing the view are stored, it is called a *materialized* view and the relation is the *extension* of the view. Denote by D_V the database D extended with the extensions of the views belonging to a view set V.

Containment & Equivalence The concepts of query containment and equivalence are central to query rewriting theory [ref Encyclopedia article Query Rewriting] as they are used to test the correctness of a rewriting of a query using a set of views.

Definition 1 *A query Q_1 is contained in a query Q_2, denoted $Q_1 \sqsubseteq Q_2$, if for all database instances D, the set of tuples computed for Q_1 is a subset of those computed for Q_2, i.e., $Q_1(D) \subseteq Q_2(D)$. The two queries are equivalent if $Q_1 \sqsubseteq Q_2$* and $Q_2 \sqsubseteq Q_1$.

Containment mappings provide a necessary and sufficient syntactic condition for testing query containment of conjunctive queries.

Definition 2 *A mapping τ from $Vars(Q_2)$* to $Vars(Q_1)$ is a *containment mapping if*

- *It is the identity on constants*
- *It maps every subgoal in$Subg(Q_2)$ to a subgoal in$Subg(Q_1)$, and*

- *It maps the head of Q_2 to the head of Q_1.*

A seminal result in semantic query optimization is that a query Q_1 is contained in Q_2 if and only if there is a containment mapping from Q_2 to Q_1.

Rewritings

Given a query Q and a set of views $V_1, V_2,..., V_n$ over the same database schema, the goal is to find an *equivalent rewriting* Q' of the query, i.e., a query Q' that uses one or more of the views in the body.

Definition 3 *A query Q' is an equivalent rewriting of query Q that uses the set of views $V = \{V_1, V_2,..., V_n\}$* if

- *Q and Q' are equivalent, and*
- *Q' refers to the views in V.*

For every database D, Q' is evaluated over D_V. A rewriting Q' is *locally minimal* if no literals from Q' can be removed while retaining equivalence to Q. A rewriting is *globally minimal* if there is no other rewriting with fewer literals. If the rewriting refers *only* to the views in V and to no other relations then the rewriting is called *complete*. When looking for rewritings, it is preferable to find those that are cheaper to evaluate than the original query. Below is an example of a query and a rewriting using views.

Example 1: Consider the following query Q and view V

$$Q : q(X, U) \leftarrow p(X, Y) \wedge p_0(Y, Z) \wedge p_1(X, W) \wedge p_2(W, U).$$

$$V : v(A, B) \leftarrow p(A, C) \wedge p_0(C, B) \wedge p_1(A, D).$$

Q can be rewritten using V as follows:

$$Q' : q(X, U) \leftarrow v(X, Z) \wedge p_1(X, W) \wedge p_2(W, U).$$

By substituting the view, the first two literals of the query can be removed. However, although the third literal in Q is guaranteed to be satisfied by V, it cannot be removed from the query, as the variable D is projected out in the head of V. Hence, if p_1 were removed from the query, the join condition between p_1 and p_2 could not be enforced.

In several settings, *maximally contained* rewritings need to be considered. Unlike equivalent rewritings, maximally contained rewritings may differ depending on the language used to express the rewritings. Therefore, the following definition depends on a particular query language:

Definition 3 *Let Q be a query and $V = \{V_1,..., V_m\}$ be a set of views over the same database schema, and L be a query language. The query Q' is a maximally contained rewriting of Q* using *V with respect to L* if:

- *Q' is a query in L that refers only to the views in V,*
- *$Q' \sqsubseteq Q$, and*
- *for every rewriting $Q_1 \in L$, such that $Q_1 \sqsubseteq Q$, $Q_1 \sqsubseteq Q'$.*

Characterizing Rewritings

Answering queries using views is closely related to the problem of query containment. The following proposition provides a necessary and sufficient condition for the existence of a rewriting of Q that includes a view V.

Proposition 1 *Let Q* and *V be conjunctive queries with built-in predicates. There is a rewriting of Q using V* if and only if $\pi_\emptyset(Q) \sqsubseteq \pi_\emptyset(V)$*, i.e., the projection of Q onto the empty set of columns is contained in the projection of V onto the empty set of columns.*

This proposition provides a complete characterization of the problem of using views for query answering. Two other important characteristics of the problem are that, for conjunctive queries and views, a rewriting that does not introduce new variables and does not include database relations that do not appear in the original query, can always be found. These characteristics allow significant pruning of the search space for a minimal rewriting of Q, but do not always hold for more expressive settings of the problem. Finally, a minimal rewriting of a query Q using a set of views V without built-in predicates does not need more than the number of literals in the query. In particular, *if the body of Q has p literals and Q' is a locally minimal and complete rewriting of Q using V, then Q' has at most p literals.*

The bound provided above does not hold when the database relations have functional dependencies. In such a case the size of a minimal rewriting is at most $p + d$ literals, where d is the sum of the arities of the literals in Q. In the presence of built-in predicates, the size of the rewritten query is at most exponential in the size of Q.

The above properties determine the complexity of the problem. In particular, if Q is a conjunctive query with built-in predicates and V is a set of conjunctive views *without* built-in predicates, then the problem of determining whether there exists a rewriting of Q that uses V is NP-complete. If the views in V have built-in predicates, the problem is Π_2^p-complete. If neither the query nor the views have built-in predicates, then finding a rewriting with at most k literals, where k is the number of literals in the body of Q, is NP-complete. Finally, if the query and the views have built-in predicates, then finding a rewriting with at most k literals is in Σ_3^p.

Techniques for Answering Queries Using Views

The above characterization suggests a two-step algorithm for finding rewritings of a query Q using a set of views V. At first, find some containment mapping from V to Q and add to Q the appropriate atoms of V, resulting in a new query Q'. Then, minimize Q' by removing literals from Q that are redundant. An algorithm needs to consider every possible conjunction of k or fewer view atoms, where k is the number of subgoals in the query. Algorithms that attempt to explore more effectively the search space to produce maximally contained rewritings include the Bucket Algorithm, the Inverse-Rules Algorithm and the MiniCon Algorithm.

The Bucket Algorithm The main idea underlying the algorithm is that the query rewritings that need to be considered can be significantly reduced if for each subgoal in the query the relevant views are identified.

To demonstrate the algorithm, the following query and views are used (The example used is from [10]):

$$\begin{aligned} Q_1(x) &\leftarrow \mathit{cites}(x,y) \wedge \mathit{cites}(y,x) \wedge \mathit{sameTopic}(x,y) \\ V_1(a) &\leftarrow \mathit{cites}(a,b) \wedge \mathit{cites}(b,a) \\ V_2(c,d) &\leftarrow \mathit{sameTopic}(c,d) \\ V_3(f,h) &\leftarrow \mathit{cites}(f,\ g) \wedge \mathit{cites}(g,\ h) \wedge \mathit{sameTopic}(f,\ g) \end{aligned}$$

Given a query Q, the Bucket Algorithm proceeds in two steps. In the first step, the algorithm creates a bucket for each subgoal in Q that is not in $C(Q)$, where by $C(Q)$ refers to the subgoals of comparison predicates of Q. Each bucket contains the views that are relevant to answering the particular subgoal. For the example, the following buckets are created:

cites(*x*, *y*)	*cites*(*y*, *x*)	*sameTopic*(*x*, *y*)
$V_1(x)$	$V_1(x)$	$V_2(x, y)$
$V_3(x, y)$	$V_3(x, y)$	$V_3(x, y)$

The algorithm also requires that every distinguished variable in the query should be mapped to a distinguished variable in the view. Hence, the algorithm does not include the entry $V_1(y)$ even though it is possible to map the subgoal $\mathit{cites}(x,y)$ in the query to the subgoal $\mathit{cites}(b,a)$ in V_1.

In the second step, for each element of the Cartesian product of the buckets, the algorithm constructs a conjunctive rewriting and checks whether it is contained (or can be made to be contained) in the query. If so, the rewriting is added to the answer. Therefore, the result of the Bucket Algorithm is a union of conjunctive rewritings.

In the example, the algorithm will try to combine V_1 with the other views and fail. Then it will consider the rewritings involving V_3 and V_2, and discover a contained rewriting. Finally, it will consider the rewritings involving only V_3, and find a contained rewriting by equating the variables in the head of V_3. It is possible to add an additional check that will determine whether a resulting rewriting will be redundant, as is the case here with the rewriting combining V_3 and V_2. Therefore, the only minimal rewriting in this example is: $Q_1'(x) \leftarrow V_3(x,\ x)$. This rewriting is in fact an equivalent one.

The Bucket Algorithm misses important interactions between view subgoals by considering each subgoal separately. Consequently, the buckets can contain unusable views and, as a result, the second step of the algorithm can become very expensive.

The Inverse Rules Algorithm The main idea of the Inverse-Rules Algorithm is to construct a set of rules that *invert* the view definitions, i.e., rules that compute tuples for the database relations from tuples of the views. The inverse rules together with a conjunctive query Q constitute a maximally contained rewriting for Q that is represented as a union of conjunctive queries. Considering the views of the previous example, the algorithm would construct the following inverse rules:

$$\begin{aligned} &R_1 : \mathit{cites}(a, f_{V_1}(a)) \leftarrow V_1(a) \\ &R_2 : \mathit{cites}(\ f_{V_1}(a), a) \leftarrow V_1(a) \\ &R_3 : \mathit{sameTopic}(c,d)) \leftarrow V_2(c,d) \\ &R_4 : \mathit{cites}(\ f, f_{V_3}(f,h)) \leftarrow V_3(f,h) \\ &R_5 : \mathit{cites}(\ f_{V_3}(f,h), h) \leftarrow V_3(f,h) \\ &R_6 : \mathit{sameTopic}(f, f_{V_3}(f,h)) \leftarrow V_3(f,h) \end{aligned}$$

For every view relation V with an existential variable Z_i, a function symbol $f_{V,i}$ is introduced. The above rules provide all the information about the database relations that can be extracted from the view extension. Intuitively, a tuple of the form (A) in the extension of the view V_1 is a *witness* of two tuples in the relation *cites*: a tuple of the form (A, Z), for some value of Z, and a tuple of the form (Z, A), for the same value of Z. The rules will generate from a tuple $V_1(A)$ two such tuples that will contain the unique functional term $f_{V_1}(A)$: it is a syntactic stand-in for the unknown (but known to exist) common value in the two tuples.

The rewriting of a query Q using the set of views V is simply the query consisting of Q and the inverse rules for V (There is a systematic way to eliminate the function symbols). Two important advantages of the algorithm are that the inverse rules can be constructed ahead of time, independent of a particular query, in polynomial time, and that the technique is also applicable virtually without change to recursive queries.

The rewritings produced by the Inverse-Rules Algorithm have some shortcomings when used for query evaluation. First, applying the inverse rules to the extension of the views may invert some of the computation done to produce the extent of the view. Second, they may access views that have no relevance to the query.

The MiniCon Algorithm The MiniCon Algorithm starts by considering, for each subgoal in the query, which view specializations contain subgoals that can "cover" it. A view specialization is created from a view by possibly equating some head variables. Once the algorithm finds a partial mapping φ from a subgoal g in the query to a subgoal g_1 in the view specialization V, it extends it to a minimal additional set of subgoals G of the query that must also be mapped to V. In particular, G includes all subgoals of the query that contain some variable of g mapped by ϕ to an existential variable of V. This set of subgoals and mapping information is call a *MiniCon Description* (MCD), and can be viewed as a generalized bucket. Having considered in advance how each of the variables in the query can interact with the available views, the second phase of the MiniCon Algorithm needs to consider significantly fewer combinations of these generalized buckets.

Using the query and views of the previous example, the MCDs formed during the first phase of the MiniCon Algorithm are:

$V(Y)$	h	φ	G
$V_2(c, d)$	$c \to c, d \to d$	$x \to c, y \to d$	3
$V_3(f, f)$	$f \to f, h \to f$	$x \to f, y \to f$	1, 2, 3

where h is a head homomorphism on V_i, $V(\bar{Y})$ is the result of applying h to V_i to create a view specialization, φ is a partial mapping from $Vars(Q_1)$ to $h(Vars(V_i))$ and G is a subset of the subgoals in Q_1 that are covered by some subgoal in $h(V_i)$ under the mapping φ.

In the second phase, the MCDs are combined to produce the query rewritings. In this phase the algorithm considers combinations of MCDs, and for each valid combination it creates a conjunctive rewriting of the query. There is no need for containment testing, as opposed to the Bucket algorithm. The combinations of MCDs considered by the MiniCon Algorithm are those that cover all the subgoals of the query with pairwise disjoint subsets of subgoals. The final maximally contained rewriting is a union of conjunctive queries.

As of 2008, MiniCon has been shown to be the most efficient current algorithm for answering conjunctive queries using conjunctive views.

Chase and Backchase In the presence of a set C of constraints on the relations and the views, Chase and Backchase is a powerful technique for finding *equivalent* rewritings using the views.

The chase [ref Encyclopedia entry Chase] is a well-known technique that can be used to decide containment of queries under constraints. If the chase of Q_1 with C terminates producing a query Q_c then $Q_1 \sqsubseteq_C Q_2$ if and only if $Q_C \sqsubseteq Q_2$. $Q_1 \sqsubseteq_C Q_2$ means that $Q_1 \sqsubseteq Q_2$ on every database instance that satisfies the constraint set C.

For the equivalent rewriting problem under constraints C, Q_1 is given and must effectively find Q_2 such that $Q_1 \equiv_C Q_2$. The algorithm proceeds in two steps. In the *chase* step, it chases Q_1 with C until no more chase steps are possible. This results in a query U called the *universal plan*. The universal plan is a query over the database relations and the views that conceptually incorporate all possible alternative ways to answer Q_1 in the presence of the constraints C. In particular, any minimal conjunctive query equivalent to Q_1 under C is isomorphic to a subquery of U. Hence, to find all minimal rewritings of Q_1, the backchase step of the algorithm searches the finite space of subqueries of U. Specifically, for each subquery Q_s it checks for equivalence with Q_1 by chasing Q_s with C. Q_s is equivalent to Q_1 if and only if there is a containment mapping from Q_1 into an intermediate chase result of Q_s.

Chase and Backchase applies if the constraint set C includes tuple-generating and equality-generating dependencies. It is sound and complete if C is *weakly acyclic*.

Key Applications

The conceptual framework and techniques developed for query rewriting using views have had significant impact in a number of areas of information systems, especially (but not exclusively) in the areas of information integration and data integration [ref Encyclopedia article Information Integration, ref Encyclopedia article Data Integration]. In particular, as of 2007, data integration theory and systems follow two main architectural approaches for defining and processing queries in a virtual (In a virtual integration system, as opposed to a data warehouse [ref entry Data Warehouse], data resides only in their original locations/sources. The integration system accesses them every time it needs to answer a query) integration system. In *global-as-view* the integrated *global view* (or views) of the disparate data sources is defined (using a standard query language or an ad hoc specification system) in terms of the *local* data sources, and queries are directly expressed in terms only of the global view(s). In this case, processing a query over this global view involves *view expansion*. In *local-as-view* data integration on the other hand, local data sources are expressed in terms of an a priori given *global schema*, i.e., they are expressed as views over the relations of the global schema. Queries are also expressed in terms of the global schema. Since in an integration system the only available data reside in the local sources, answering a query requires rewriting it in terms of the available sources, i.e., rewriting it using the views.

In the above context, sources describe their contents in terms of the local schema. In addition, for sources with limited processing power, their capabilities need to also be taken into account. Since integrated queries are answered by retrieving data from local sources, the data retrieval requests (i.e., the queries) submitted to the sources need to conform to their capabilities. Query rewriting using views has provided the conceptual framework and tools to address the *capability-based rewriting* problem.

Another area of application of the ideas of answering queries using views is query optimization. When the set of relations mentioned in queries overlaps with those mentioned in one or more views, and the views select one or more attributes selected by the query, the view may be *usable* by the query. Moreover, certain indices can also be modeled as materialized views. Using the materialized views may lower the cost of query processing, depending on the available access methods and the selectivity of the involved predicates. Answering queries using views is used to decide view usability and generate the appropriate query plans.

Finally, answering queries using views, especially in the presence of constraints, has influenced the theory of *data exchange*. Data exchange, or data translation, involves taking data described in one, *source*, schema and translating, without loss of information, into data structured under a different, *target*, schema.

Future Directions

Even though significant progress has been made in this area, the applicability and/or efficiency of the developed techniques is often limited. Future work is needed on rewriting techniques that can perform correctly and efficiently for "real" SQL views on real SQL queries, including queries with aggregates, nesting, bag semantics, and user-defined functions. Moreover, the conceptual framework of rewriting using views applies to problems in graph querying, Web service composition, and rewriting of XQuery.

Cross-references

- ▶ Data Integration
- ▶ Global as Views
- ▶ Information Integration
- ▶ Local as views
- ▶ Query Containment
- ▶ View Definition
- ▶ Views

Recommended Reading

1. Abiteboul S. and Duschka O. Complexity of answering queries using materialized views. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 254–263.
2. Cohen S., Nutt W., and Sagiv Y. Containment of aggregate queries. In Proc. 9th Int. Conf. on Database Theory, 2003, pp. 111–125.
3. Deutsch A., Popa L., and Tannen V. Query reformulation with constraints. ACM SIGMOD Rec., 35(4), 2006.

4. Duschka O. and Geneserth M. Answering recursive queries using views. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997, pp. 109–116.
5. Halevy A. Answering queries using views: A survey. VLDB J., 10(4):270–294, 2001.
6. Lenzerini M. Data Integration: A theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
7. Levy A.Y., Mendelzon A.O., Sagiv Y., and Srivastava D. Answering queries using views. In Proc. 14th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1995, pp. 95–104.
8. Lin V., Vassalos V., and Malakasiotis P. MiniCount: Efficient rewriting of COUNT-Queries using views. In Proc. 22nd Int. Conf. on Data Engineering, 2006, pp. 1.
9. Papakonstantinou Y. and Vassalos V. Query Rewriting using semistructured views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 455–466.
10. Pottinger R. and Halevy A. Minicon: A scalable algorithm for answering queries using views. VLDB J., 10(2–3):182–198, 2001.
11. Xu A. and Meral Ozsoyoglu Z. Rewriting XPath queries using materialized views. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 121–132.

Anti-monotone Constraints

Carson Kai-Sang Leung
University of Manitoba, Winnipeg, MB, Canada

Definition

A constraint C is *anti-monotone* if and only if for all itemsets S and S':

$$\text{if } S \supseteq S' \text{ and } S \text{ satisfies } C, \text{ then } S' \text{ satisfies } C.$$

Key Points

Anti-monotone constraints [1,2] possess the following nice property. If an itemset S satisfies an anti-monotone constraint C, then all of its subsets also satisfy C (i.e., C is downward closed). Equivalently, any superset of an itemset violating an anti-monotone constraint C also violates C. By exploiting this property, anti-monotone constraints can be used for pruning in frequent itemset mining with constraints. As frequent itemset mining with constraints aims to find itemsets that are frequent and satisfy the constraints, if an itemset violates an anti-monotone constraint C, all its supersets (which would also violate C) can be pruned away and their frequencies do not need to be counted. Examples of anti-monotone constraints include $min(S.Price) \geq \$20$ (which expresses that the minimum price of all items in an itemset S is at least \$20) and the usual frequency constraint $support(S) \geq minsup$ (i.e., $frequency(S) \geq minsup$). For the former, if the minimum price of all items in S is less than \$20, adding more items to S would not increase its minimum price (i.e., supersets of S would not satisfy such an anti-monotone constraint). For the latter, it is widely used in frequent itemset mining, with or without constraints. It states that (i) all subsets of a frequent itemset are frequent and (ii) any superset of an infrequent itemset is also infrequent. This is also known as the *Apriori property*.

Cross-references

▶ Frequent Itemset Mining with Constraints

Recommended Reading

1. Lakshmanan L.V.S., Leung C.K.-S., and Ng R.T. Efficient dynamic mining of constrained frequent sets. ACM Trans. Database Syst. 28(4):337–389, 2003.
2. Ng R.T., Lakshmanan L.V.S., Han J., and Pang A. Exploratory mining and pruning optimizations of constrained associations rules. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 13–24.

AP@n

▶ Average Precision at n

Applicability Period

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

The *applicability period* (or *period of applicability*) for a modification (generally an insertion, deletion, or update) is the time period that modification is to be applied. Generally the modification is a sequenced modification and the period applies to valid time. This period should be distinguished from *lifespan*.

Key Points

The applicability period is specified within a modification statement. In constrast, the lifespan is an aspect of a stored fact.

This illustration uses the TSQL2 language, which has an explicit VALID clause to specify the applicability period within an INSERT, DELETE, or UPDATE statement.

For insertions, the applicability period is the valid time of the fact being inserted. The following states that Ben is in the book department for 1 month in 2007.

```
INSERT INTO EMPLOYEE
VALUES (‘Ben’, ‘Book’)
VALID PERIOD ‘[15 Feb 2007, 15 Mar 2007]’
```

For a deletion, the applicability period states for what period of time the deletion is to be applied. The following modification states that Ben in fact was not in the book department during March.

```
DELETE FROM EMPLOYEE
WHERE Name = ‘Ben’
VALID PERIOD ‘[1 Mar 2007, 31 Mar 2007]’
```

After this modification, the lifespan would be February 15 through February 28.

Similarly, the applicability period for an UPDATE statement would affect the stored state just for the applicability period.

A *current modification* has a default applicability period that either extends from the time the statement is executed to forever, or when now-relative time is supported from the time of execution to the ever-increasing current time.

Cross-references

- ▶ Current Semantics
- ▶ Lifespan
- ▶ Now in Temporal Databases
- ▶ Sequenced Semantics
- ▶ Temporal Database
- ▶ Time Period
- ▶ TSQL2
- ▶ Valid Time

Recommended Reading

1. Snodgrass R.T. (ed) The TSQL2 Temporal Query Language. Kluwer Academic, 1995.
2. Snodgrass R.T., Developing Time-Oriented Database Applications in SQL, Morgan Kaufmann, 1999.

Application Benchmark

Denilson Barbosa[1], Ioana Manolescu[2], Jeffrey Xu Yu[3]
[1]University of Alberta, Edmonton, AL, Canada
[2]INRIA Saclay–Île de France, Orsay, France
[3]The Chinese University of Hong Kong, Hong Kong, China

Synonyms

Benchmark; Performance benchmark

Definition

An application benchmark is a suite of tasks that are representative of typical workloads in an application domain.

Key Points

Unlike a MICROBENCHMARK, an application benchmark specifies broader tasks that are aimed at exercising most components of a system or tool. Each individual task in the benchmark is assigned a relative *weight*, usually reflecting its frequency or importance in the application being modeled. A meaningful interpretation of the benchmark results has to take these weights into account.

The Transaction Processing Performance Council (TPC) is a body with a long history of defining and published benchmarks for database systems. For instance, it has defined benchmarks for Online Transaction Processing applications (TPC-C and TPC-E), Decision Support applications (TPC-H), and for an Application Server setting (TPC-App).

Other examples of application benchmarks are: the OO1 and OO7 benchmarks, developed for object-oriented databases, and XMark, XBench and TPoX, developed for XML applications.

Cross-references

- ▶ Micro-Benchmarks
- ▶ XML Benchmarks

Recommended Reading

1. Carey M.J., DeWitt D.J., and Naughton J.F. The OO7 benchmark. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 12–21.
2. Gray J. (ed.). The Benchmark Handbook for Database and Transaction Systems, (2nd edn.). Morgan Kaufmann, 1993.

3. Nicola M., Kogan I., and Schiefer B. An XML transaction processing benchmark. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 937–948.
4. Schmidt A., Waas F., Kersten M.L., Carey M.J., Manolescu I., and Busse R. XMark: a benchmark for XML data management. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 974–985.
5. Transaction Processing Performance Council. Available at: http://www.tpc.org/default.asp
6. Yao B.B., Özsu M.T., and Khandelwal N. XBench benchmark and performance testing of XML DBMSs. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 621–633.

Application Recovery

David Lomet
Microsoft Research, Redmond, WA, USA

Synonyms

Persistent applications; Fault tolerant applications; Transaction processing; Recovery guarantees; Exactly once execution

Definition

Systems implement application recovery to enable applications to survive system crashes and provide "exactly once execution" in which the result of executing the application is equivalent to a single execution where no system crashes or failures occur.

Historical Background

Application recovery was first commercially provided by IBM's CICS (Customer Information Control System). Generically, these kinds of systems became known as transaction processing monitors (TP monitors) [5,9]. With a TP monitor, applications are decomposed into a series of steps. Each step is executed within a transaction. A step typically consists of reading input state from a database or transactional queue, executing some business logic, perhaps processing user input or reading and writing to a database, and finally, writing state for the next step into database or queue [4]. If a step failure occurs, its transaction is aborted. Since the prior step results are stably stored, the step can be re-executed after system recovery.

Application recovery does not always involve transactions, however, as the early work by Borg et al. demonstrates [6]. More recently, in the web context, TP monitors have been renamed as application servers (app servers). App servers are similar to TP monitors, where state is explicitly managed, frequently by using transactions. Ongoing interest in application recovery is illustrated by the "recovery oriented computing" (ROC) project [3] and the Phoenix project [1].

Foundations

Introduction

Exactly-Once Applications: While application programmers are usually familiar with the problem area for which an application is written, they are frequently also faced with having to deal with "system problems" of reliability and availability. The system goal is to permit applications to achieve "exactly once execution" [2]. For example, an airline reservation system wants to issue exactly the number of tickets a customer requests, instead of no tickets or twice the number.

Types of Failures: Applications cannot survive all forms of failures. If an exactly-once execution produces a "deterministic" failure, then every execution of the application will lead to that same failure. Such deterministic failures are called "hard failures." The failures that software can deal with are called "soft failures." A subsequent execution of the system will usually avoid the state leading to the system failure. Soft failures arise in a number of ways.

1. *Software non-determinism*: A software system is non-deterministic if, when re-executed, it results in a different execution path than a prior execution. Non-determinism can arise when, for example, paths are determined by relative processor speed or the sequence of external events. Such software bugs have been called "Heisenbugs" (hard failures being "Bohrbugs").
2. *Soft hardware failures*: Hardware can also suffer from "Heisenbugs." For example, a transient hardware failure may be triggered by an environmental cause, such as a cosmic ray changing a memory bit, etc.
3. *Operator failures*: Systems occasionally require operator intervention. Operators, being human, make mistakes. An operator is unlikely to make the same mistake at the same point in a subsequent execution.

Recovery is effective because failures are usually "soft" [3,9], which is why database recovery is so successful.

Support for Application Persistence: Applications which guarantee "exactly once" execution are called "persistent applications" because application state persists across or despite system failures. The traditional method for providing application persistence has been to use a TP monitor. However, new approaches permit implicit state management, in which the application programmer delegates the problem of managing application state to the system infrastructure.

Implicit state management does not require application steps to execute within transactions. It permits applications to be coded more "naturally" (though restrictions exist). Application state is made stable "under the covers." Phoenix [1] does this via logging application non-determinism and replaying the captured non-determinism after a failure to recover application state. CORBA's approach [11], having shorter down time but more costly normal execution, replicates application state so that if part of the system fails, a copy of the state is available elsewhere for continued execution. Implicit state management is discussed here.

Application persistence (recovery) requires different techniques than database recovery: (i) Applications are usually single-threaded and "piece-wise deterministic," while databases execute highly concurrent code. (ii) Application state is frequently distributed, so dealing with distribution is essential. (iii) Databases change state entirely within a transaction, while application state may frequently change outside of a transaction.

Persisting Application State

Implicit state management requires the infrastructure supporting an application to capture application state and make it stable in some way, relieving the programmer of this task. Two approaches have been developed. The approaches are not as different as they appear. Both require that applications be piece-wise deterministic, with clearly identified non-deterministic events that can be "applied" at an application instance to be used in deterministic re-execution that can generate the same state as the original execution.

Recovery: Recovery technology usually assumes that a failing application will be re-activated after a system crash and its activities recovered on the original system. Thus recovery oriented approaches usually capture application state stably (e.g., "on disk") so that it survives a system failure. There are two parts to this:

1. Writing a "snapshot" of the state to stable storage from time to time. It matters when this snapshot, also called a checkpoint, occurs.
2. Stably logging non-deterministic events encountered by the application in arrival order.

Recovery, in which the state is re-created following a system failure, consists of re-installing the latest captured state (checkpoint) followed by re-executing the application by "feeding" events from the log to re-create application state as of the last logged event.

Logging can be either optimistic or pessimistic [7]. Pessimistic logging eagerly makes log records stable, usually as events occur in order to ensure that execution is "exactly once." Optimistic logging defers for a time making events stable, frequently sacrificing exactly once execution. Pessimistic logging can be greatly optimized. Optimistic logging can be constrained so that exactly once execution is assured. This brings these techniques toward some middle ground in performance, though important differences remain.

Replication: Replication technology usually assumes that an application executing on a failing system A will continue execution on a separate system B where a replica of its state has been maintained. The problem for replica oriented approaches is to capture and keep in sync an application's state on these separate systems. There are two generic approaches to this:

1. Designate one system as the primary system, the others as secondaries. Non-deterministic events are sent to the primary system, which then relays them to the secondary in the order that it received them. Replicas thus execute in response to the same sequence of events. A secondary becomes the primary on failure of the primary. If a secondary fails, the primary needs to know this and re-create another secondary.
2. Use atomic broadcast to send non-deterministic events to all systems maintaining replicas "simultaneously" [1]. Atomic broadcast guarantees that replicas receive all messages and in the same order. If there is a failure, then the remaining replicas participating in the atomic broadcast need to know the members of the new group so that the atomic broadcast protocol continues to work correctly, possibly including the creation of a new replica.

Intermediate Approaches: The line between replication and recovery approaches is not always crisp: (i) A recovery system can maintain its log by sending the log records to another system, hence using a second system as stable storage instead of a disk. This second site might then become the site where the application is subsequently executed should the original site fail. (ii) A second system for a replica might not execute the events forwarded to it immediately, but rather simply "log" them. Then, only if a primary replica fails would it perhaps then "catch up" by executing the stored events.

Thus a second system might serve as a cold standby system (retaining a log of events) or as a hot standby, immediately executing the application in response to the events. Warm standbys are also possible in which an application is executed at a secondary site in a lazy fashion, lagging the state of a primary site, but not by too much. Whether this is should be viewed as recovery or replication is not really important.

Distributed Applications

Many applications are distributed, e.g., a web application might consist of a client component providing the user interface, one or more middle tier components executing business logic, and back end components that typically provide transactions and database functionality. The new problem is to coordinate the states of multiple software components executing different parts of the application [6,7].

1. The state of the set of application components needs to be "causal," i.e., every message receive reflected in the state of some component must always be accompanied by a sender in a state where the message has been sent.
2. Messages (non-deterministic events in a distributed system) must result in an "exactly once" execution by the receiving component.
3. The application must be able to interact with users or other elements outside of the persistence infrastructure, which may not obey the required protocols.
4. The application must be able to interact with transactional elements like databases.
5. Different strategies have different costs and impact the balance between normal run time costs and recovery costs and time-to-recovery.

Contracts for Persistent Components: Providing persistence for components of a distributed application requires an agreed upon set of protocols or "contracts" [1,2] involving component state and message stability, repeated sending of messages, eliminating duplicate messages, etc. The basic contract between persistent components, called the "committed interaction contract" or CIC, places burdens on both sender and receiver of a message at the time it is sent.

The *sender of a message* ensures causality. The sender ensures that it's state as of the time of message send and the message will be persistent earlier than the receiver's state is persisted. Further, the sender must continue to send the message until the receiver acknowledges the message, in order to deal with unreliable networks and crashed receivers, etc. The CIC does not specify how to do this, but the recovery approach usually writes non-deterministic events to a log, and flushes the log when a message is sent should the state include previously received messages since the last log flush.

The *receiver of the message* ensures exactly once execution. The receiver eliminates duplicate messages the sender may send in its effort to provide reliable delivery. The receiver executes in response to a message only if receiver state does not already reflect having received the message and bypasses execution otherwise, returning the same result as produced by the original execution. Finally, the receiver ensures that the state resulting from the message receipt is stable at an appropriate moment. The CIC contract does not specify how to do this, but the receiver might use a table of messages received, or some high water mark for messages when the sender is known, to which it compares each incoming message.

External and Transactional Components: Many web applications involve users entering information at a keyboard, and a database that stores the results of business dealings, e.g., the purchase of a plane ticket. Hence, persistent components must interact with elements outside of the boundaries of the application and its supporting system. This requires new forms of "contracts" in which the main burden is placed on the persistent components to enable an ensemble of elements to achieve exactly once execution, e.g., one purchase of the plane ticket.

External components, including users, may not obey CIC requirements. Hence, external interactions must be limited to ensure exactly once execution. For users, both reads and writes might be exposed as having multiple occurrences. A failure in the middle of a user interaction may require entering data more than once or repeating an output if the system fails between the event and its stable logging. Systems typically minimize the problem window by immediate logging, with a log flush, in response to external events.

A transactional component only occurs at the edge of an application system, responding to requests for and updates to its data. Transactions can abort, with the transactional component's state reset to remove all effects of the aborted transaction. The persistent component must be prepared to handle transaction aborts at any time. Transactions actually reduce the burden for a persistent component as it need not ensure that its state is stable at each interaction within a transaction. A system crash will eventually lead to the abort of any interrupted transaction. However, at the point where the persistent component requests a transaction commit, it must obey the usual requirements for a sender in a CIC.

Optimizations: Optimizations can reduce the normal runtime cost of providing persistent applications. These exploit additional information known to the application programmer when the application is either written or being deployed.

1. Some components may be read-only, producing no external side effects. So a read only component need do no logging itself, while a calling persistent component can log lazily because the read can be repeated if needed. A functional call component in which the result depends only on the arguments of the call requires no logging as the call can be replayed idempotently.
2. If called components extend the time they stably remember prior calls and results for idempotence, then the calling component need not log to make its state stable prior to each call. It can depend upon the called components in a sequence of calls to capture the call results to enable its deterministic replay.
3. When a component is a "server" for exactly one client, the client can capture what would be nondeterministic calls and their order for the server component, relieving the server from needing to log calls messages. Combining this with item 2 enables persistent components without logging [8,10]. A logless component's state is regenerated by its client replaying the sequence of calls, and it re-executing its own series of calls to other components that have captured the call results. Logless components are easily deployed anywhere, can be freely replicated, and hence make persistence simple, flexible, and low cost.

Checkpoints: Recovery time is shortened via checkpoints. Component state consists of its variables and its execution time stack. By checkpointing component state when there is no execution within the component, the checkpoint can be accomplished solely by capturing its variable values, e.g., checkpoint might be taken transparently during the midst of a return from a call to a component. The cost of the checkpoint and the need for fast recovery time dictate checkpoint frequency.

Discussion

Making declarative the requirements for application persistence, as represented by interaction contracts, makes it easier to provide and optimize exactly once execution by expanding the range of implementation options, including interesting optimizations.

The level of the contract is also important. A CIC could have been derived from reliable messaging instead of more explicitly as repeated sending of messages with duplicate elimination. However, reliable messaging does not describe what happens to messages once they are delivered. By describing the requirements for state stability, etc., it becomes clear that delivery is not, by itself, sufficient. Further, were "persistent reliable messaging" used, this would requiring extra log forces that, can frequently be avoided with the "end-to-end" application persistence protocol.

System provided transparent application persistence is an example of delegating a serious problem to the "system," similar to how transactions delegate concurrency control and recovery to database systems. There is no need for special application programmer consideration, simplifying the application logic, and improving programmer productivity.

Key Applications

Application recovery (persistence) is used wherever exactly once semantics is required. Traditionally, this has been within transaction processing systems, but now more commonly, this involves web applications for e-business, whether for end user customers or business to business dealings. If there are financial or legal requirements for applications, they will be built using some form of application recovery to ensure exactly once execution.

Cross-references

► Transaction

Recommended Reading

1. Barga R., Chen S., and Lomet D. Improving Logging and Recovery Performance in Phoenix/App. In Proc. 20th Int. Conf. on Data Engineering, 2004.
2. Barga R., Lomet D., Shegalov G., and Weikum G. Recovery Guarantees for Internet Applications. ACM Trans. internet Tech., 4(3):289–328, 2004.
3. Berkeley/Stanford Recovery-Oriented Computing (ROC) Project. http://roc.cs.berkeley.edu. October 10, 2008.
4. Bernstein P., Hsu M., and Mann B. Implementing Recoverable Requests Using Queues. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp.112–122.
5. Bernstein P. and Newcomer E. Principles of Transaction Processing. Morgan Kaufmann, 1997.
6. Borg A., Baumbach J., and Glazer S. A message system supporting fault tolerance. In Proc. 9th ACM Symp. on Operating System Principles, 1983, pp. 90–99.
7. Elnozahy E.N., Alvisi L., Wang Y., and Johnson D.B. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. ACM Comp. Surv., 34(3), 2002, pp. 375–408.
8. Frølund S. and Guerraoui R. A Pragmatic Implementation of e-Transactions. In Proc. 19th Symp. on Reliable Distributed Syst., 2000, pp. 186–195.
9. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.
10. Lomet D. Persistent Middle Tier Components without Logging. In Proc. Int. Conf. on Database Eng. and Applications, 2005, pp. 37–46.
11. Narasimhan P., Moser L., and Melliar-Smith P.M. Lessons Learned in Building a Fault-Tolerant CORBA System. DSN, 2002, pp. 39–44.

Application Server

Heiko Schuldt
University of Basel, Basel, Switzerland

Synonyms

Web application server; Java application server

Definition

An *Application Server* is a dedicated software component in a three-tier or multi-tier architecture which provides application logic (business logic) and which allows for the separation of application logic from user interface functionality (client layer), delivery of data (web server), and data management (database server).

Key Points

Modern information systems, especially information systems on the web, follow an architectural paradigm that is based on a separation of concerns. In contrast to monolithic (single tier) architectures or two-tier client/server architectures where business logic is bundled with other functionality, three-tier or multi-tier architectures consider dedicated application servers which exclusively focus on providing business logic.

In three-tier or multi-tier architectures, application servers typically make use of several middleware services which enable the communication within and between layers. Application servers usually provide the basis for the execution of distributed applications with transactional guarantees on top of persistent data. In large-scale deployments, systems might encompass several instances of application servers (application server clusters). This allows for the distribution of client requests across application server instances for the purpose of load balancing.

Early application servers evolved from distributed TP Monitors. Over time, a large variety of application servers has emerged. The most prominent class consists of Java application servers, either as stand-alone servers or embedded in large software systems.

Cross-references

▶ Database Middleware
▶ Enterprise Application Integration
▶ Java EE
▶ Middleware Support for Database Replication and Caching
▶ Multi-Tier Architecture
▶ Replication in Multi-Tier Architectures
▶ Transactional Middleware

Recommended Reading

1. Burke B. and Monson-Haefel R. Enterprise JavaBeans 3.0. O'Reilly, 5th edn., 2006.
2. Jacobs D. Data management in application servers. Datenbank-Spektrum, 8:5–11, 2004.
3. Raghavachari M., Reimer D., and Johnson R.D. The Deployer's problem: configuring application servers for performance and reliability. In Proc. 25th Int. Conf. on Software Eng, May 2003, pp. 484–489.

Application Server Clustering

▶ Replication in Multi-Tier Architectures

Application-Centric Interfacing

▶ Enterprise Application Integration

Application-Level Tuning

Philippe Bonnet[1], Dennis Shasha[2]
[1]University of Copenhagen, Copenhagen, Denmark
[2]New York University, Newyork, NY, USA

Synonyms

Query tuning; Tuning the application interface

Definition

An under-appreciated tuning principle asserts *start-up costs are high; running costs are low*. When applied to the application level, this principle suggests performance of a few bulk operations that manipulate and transport a lot of data rather than many small operations that act on small amounts of data. To make this concrete, this entry discusses several examples and draws lessons from each.

Historical Background

Application-level tuning is about changing the way a task is performed. This entails finding a better algorithm or finding a better way to handle the database. The first is difficult to automate, but the latter goes back to the very first use of the relational databases. Whether on disk or in main memory, databases have generally always performed best when a single statement accesses all and exactly the data needed for a task.

Foundations

Application-level tuning has the nice property that it often is a pure win. Whereas adding or removing indexes often entails a trade-off between insert/delete/update performance and query performance, application-level tuning for the most part improves the performance of certain queries without hurting the performance of others. Rewriting queries to use resources more efficiently also often gives a greater benefit than physical changes.

Assemble Object Collections in Bulk

Object-oriented encapsulation is the principle of shielding the user of an object from the object's implementation. Encapsulation sometimes is interpreted as *the specification is all that counts*. That interpretation can, unfortunately, lead to horrible performance.

The reason is simple. The first design that seems to occur to object-oriented implementers is to make relational records (or sometimes fields) into objects. This has the virtue of generality. Fetching one of these objects then translates to a fetch of a record or a field. So far, so good. But then the temptation is to build bulk fetches from fetches on little objects (the so-called "encapsulation imperative"). The net result is a proliferation of small queries instead of one large query.

Consider for example a system that delivers and stores trade information. Each document type (e.g., a report on a customer account) is produced according to a certain schedule that may differ from one trade type to another. "Focus" information relates trade types to risk analysts. That is, risk analysts may focus on one trade type or another. This gives a pair of tables of the form:

```
Focus(analyst, tradetype)
Tradeinstance(id, tradetype, tradedetail)
```

When an analyst logs in, the system gives information about the trade instances in which he or she would be interested. This can easily be done with the join:

```
select tradeinstance.id, tradeinstance.
tradedetail
from tradeinstance, focus
where tradeinstance.tradetype=focus.
tradetype
and focus.analyst={input analyst name}
```

But if each trade type is an object and each trade instance is another object, then one may be tempted to write the following code:

```
Focus focustypes=new Focus();
Focus.init({input analyst name});
for (Enumeration e=focustypes.elements(); e.hasMore-
Elements();)
{
TradeInstance tradeinst=new TradeInstance();
tradeinst.init(e.nextElement());
tradeinst.print();
}
```

This application program will first issue one query to find all the trade types for the analyst (within the init method of Focus class):

```
select tradeinstance.tradetype
from focus
where focus.analyst={input analyst name}
```

and then for each such type *t* to issue the query (within the init method of TradeInstance class):

```
select tradeinstance.id, tradeinstance.
tradedetail
from tradeinstance
where tradeinstance.tradetype=t
```

This is much slower than the previous SQL formulation. The join is performed in the application and not in the database server.

The point is not that object-orientation is bad–encapsulation contributes to maintainability. The point is that programmers should keep their minds open to the possibility that accessing a bulk object (e.g., a collection of documents) should be done directly rather than by forming the member objects individually (incurring a start-up cost each time) and then grouping them into a bulk object on the application side.

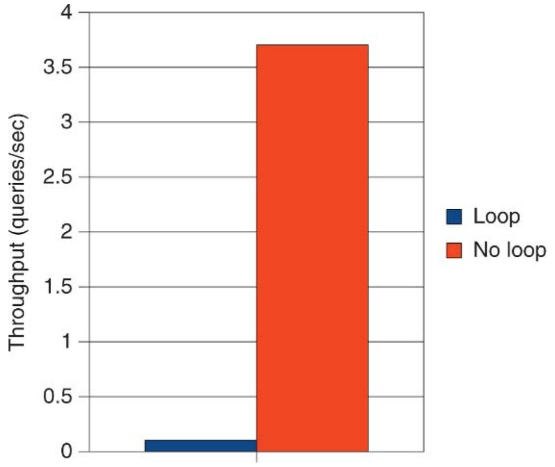
The Art of Insertion

This entry has discussed retrieving data so far. Inserting data rapidly requires understanding the sources of overhead of putting a record into the database:

- As in the retrieval case, the first source of overhead is an excessive number of round trips across the database interface. This occurs if the batch size of inserts is too small. A more radical approach is to assemble all the data to be inserted into a file, load that file into a temporary table, and then insert from that temporary table into the target table. This can improve performance by a factor of 100 or more when tables are large.
- The second issue has to do with the ancillary overhead that an insert causes: updating all the indexes on the table. Even a single index can hurt performance. For this reason, it is often a good idea to add indexes after loading the data.

Key Applications

Application-level tuning has proved useful in every setting where performance is an issue. Whereas index and physical layer tuning can be more easily automated, the benefits of application-level tuning are often greater.



Application-Level Tuning. Figure 1

Experimental Results

Looping hurts

This experiment illustrates the overhead of crossing the application interface. Two hundred tuples are fetched (these are small tuples composed of two integers) using a range query (no loop), or looping on the client-side to fetch one tuple at a time.

Figure 1 traces the throughput observed for this experiment on MySQL 6.0 with a warm buffer. There is an almost two orders of magnitude penalty when looping through the tuples. This is due to the overhead of crossing the application interface 200 times.

Url to Code and Data Sets

Loop experiment: http://www.databasetuning.org/?sec=loop

Cross-references

▶ Application server
▶ DBMS Interface
▶ JDBC
▶ ODBC
▶ SQL

Recommended Reading

1. Celko J. Joe Celko's SQL for Smarties: Advanced SQL Programming, 3rd edn. Morgan Kaufmann, San Francisco, CA, 2005.
2. Shasha D. and Bonnet P. Database Tuning: Principles, Experiments and Troubleshooting Techniques. Morgan Kaufmann, San Francisco, CA, 2002.
3. Tow D. SQL Tuning. O'Reilly, 2003.

Applications of Emerging Patterns for Microarray Gene Expression Data Analysis

Guozhu Dong[1], Jinyan Li[2]
[1]Wright State University, Dayton, OH, USA
[2]Nanyang Technological University, Singapore, Singapore

Definition

This topic is related to applications of emerging patterns in the bioinformatics field, in particular for in-silico cancer diagnosis by mining emerging patterns from large scale microarray gene expression data.

Key Points

The contemporary gene expression profiling technologies such as cDNA microarray chips and Affymetrics DNA microarry chips can measure the expression levels of thousands even tens of thousands of genes simultaneously. This provides a great opportunity to identify specific genes or gene groups that are responsible for a particular disease, for example, the subtypes of childhood leukemia disease. Reference [3] proposed to use emerging patterns to capture the signature patterns between the gene expression profiles of colon tumor cells and normal cells. This was the first bioinformatics work studying how gene groups and their expression intervals signify the difference between diseased and normal cells. That paper also proposed to design treatment plans to cure the diseased cells by adjusting certain genes' expression level based on the discovered emerging patterns. Reference [2] reported simple rules underlying gene expression profiles of more than six subtypes of acute lymphoblastic leukemia (ALL) patients. The rules are converted from the mined emerging patterns. The rules are also used to construct a classifier (PCL) that reached a benchmark diagnosis accuracy of 96% on an independent test data set. As gene expression data sets contain many attributes, the discovery of emerging patterns by border-differential based algorithms is sometimes slow. To tackle this problem, reference [1] proposed a CART-based approach to discover a proportion of emerging patterns from the high-dimensional gene expression profiling data with a high speed. The ZBDD based approach discussed in the emerging patterns entry of this volume is also fast and can handle large number of genes.

Cross-references

- ▶ Emerging Patterns
- ▶ Emerging Pattern Based Classification

Recommended Reading

1. Boulesteix A.-L., Tutz G., and Strimmer K. A CART-based approach to discover emerging patterns in microarray data. Bioinformatics, 19(18):2465–2472, 2003.
2. Li J., Liu H., Downing J.R., Eng-Juh Yeoh A., and Wong L. Simple rules underlying gene expression profiles of more than six subtypes of acute lymphoblastic leukemia (ALL) patients. Bioinformatics, 19:71–78, 2003.
3. Li J. and Wong L. Identifying good diagnostic genes or genes groups from gene expression data by using the concept of emerging patterns. Bioinformatics, 18:725–734, 2002.

Applications of Sensor Network Data Management

Farnoush Banaei-Kashani, Cyrus Shahabi
University of Southern California, Los Angeles, CA, USA

Synonyms

Applications of Sensor Networks, Applications of Sensor Databases, Applications of Sensor Network Databases

Definition

Sensor networks allow for micro-monitoring of different phenomena of interest in arbitrary physical environments. With this unique capability, sensor networks can capture the events in the real world as they happen in the form of high-resolution spatiotemporal data of various modalities, and provide the opportunity for real-time querying and analysis of the data for immediate response and control. Such functionality is desirable in many classic applications while enabling numerous other novel applications. From the data management perspective, there is a consensus among database researchers that management and analysis of the massive, dynamic, distributed and uncertain data in sensor network applications is going to be one of the new grand challenges for the database community:

> ▶ *Sensor information processing will raise many of the most interesting database issues in a new environment, with a new set of constraints and opportunities.*
>
> – Excerpt from the Lowell Database Research Self-Assessment (By a group of thirty senior

database researchers, *"The Lowell database research self-assessment"*, Communications of the ACM, Volume 48, Issue 5, May 2005.)

Historical Background

Shortly after the introduction of the sensor networks and their potential applications about a decade ago [6], management of the sensor data was recognized as one of the main challenges in realizing the sensor network applications [1].

Foundations

Sensor Databases

One can think of a sensor network as a distributed database that collects, stores and indexes the sensor data to answer the queries received from external users/applications as well as internal system entities. By considering a sensor network as a database, one envisions some of the benefits of the traditional databases potentially for sensor databases; e.g., reduced application development time, convenient multi-user data access and querying with a well-defined generic interface, efficient data reuse, and most importantly data independence. Physical data independence is a particularly beneficial advantage of the sensor database approach, because as compared to the physical layer of the traditional databases the physical infrastructure of the sensor networks is much more sophisticated. With physical data independence in sensor databases, the logical schema of the data exposed to the users is separated from the physical schema that defines the complex and probably changing implementation of the data structures and operations on the physical network. By separating the logical and physical schemas, users/applications are isolated from the typical complications of the distributed data processing in the volatile sensor networks and can focus on designing the logical structure of their queries. Hence, the use of the sensor data is significantly facilitated.

Sensor Database Distinctions

Sensor databases are different from traditional distributed databases in both physical specifications and data characteristics. At the physical level, nodes of the database (i.e., sensor nodes) are severely constrained in resources, such as memory space, storage space, CPU power, and most importantly energy. Moreover, in sensor databases nodes and links of the network are both highly volatile. On the other hand, with sensor devices continuously collecting measurements from the environment, sensor data is naturally very dynamic. Besides, due to inaccuracy of the sensor devices, signal interference, noise, etc., uncertainty is also an inherent characteristic of the sensor data. With such physical and data characteristics, maintaining the illusion of a database is arguably a more difficult objective with sensor databases as compared to that of the traditional distributed databases, and accordingly, requires new data management solutions:

- Database operators should be delay-tolerant, and tolerant to frequent updates of the data
- Query execution should be performed *in-network* for energy efficiency; similarly, data storage and access should be designed for energy efficiency
- Data acquisition plan is required to determine what data to collect
- Sensor query language should be augmented with new operators to specify duration and sampling rate of the data acquisition
- Query execution plan should be dynamically optimized to account for variable access delay and uncertain data availability
- Data uncertainty should be accounted for
- Volatility of the sensor network should be hidden to provide the illusion of a stable database
- Continuous queries should be supported, as sensor networks are primarily used for long-term monitoring
- Meaningful data digests should be maintained to allow for answering historical queries, since data is continuously collected despite the limited space for storage
- Aggregate spatiotemporal queries and range queries should be supported, for energy efficiency [13, 12]
- Approximate queries should be supported, as they are more meaningful with sensor data
- Triggers should be supported for the event-driven monitoring applications

One approach to implement sensor databases is to transfer all data to one or a small number of external base stations, where a traditional database system can be exploited. Alternatively, the data can be stored within the network itself with a balanced and optimal data storage plan. Although with the first approach one can more conveniently employ and extend the data management solutions applicable with the traditional databases, the second approach, termed *in-network storage*, allows for tighter coupling between query processing, on the one hand, and networking and application semantics, on the other hand. Tight coupling can potentially enable more energy efficient query processing in sensor databases. To evaluate the query processing performance with a particular sensor database implemented with either of these approaches, one can use the standard distributed database performance metrics such as incurred communication cost, query time, indexing time, throughput, load balance among nodes, data update overhead and storage requirements.

Key Applications

As compared to the traditional wireline sensor networks that have been in use for decades, the more recent wireless sensor networks enable low cost and rapid deployment of the sensing network while supporting mobility. With these desirable characteristics due to the wireless technology, recently the standard applications of the sensing networks are revived and new applications that were otherwise unthinkable are identified. The key classes of applications for sensor databases/networks are discussed below.

Environmental Monitoring

Environmental monitoring applications, specifically habitat monitoring [3], are among the earliest applications of the sensor networks. With the habitat monitoring applications, sensors are deployed to monitor animals or plants in their original habitats with most convenience for the scientists and least disturbance for the wildlife. With other environmental applications, sensors can be used to collect earth-science and atmospheric data for environmental explorations, such as the study of the air pollution, global warming, etc., and also early detection and prediction of the natural and man-made disasters, such as hurricanes, wildfires, earthquakes and biological hazards.

Military Intelligence

With rapid deployment, and inexpensive and untethered sensors, wireless sensor networks are well positioned as the tool to collect battlefield data for real-time battlefield intelligence [9]. For instance, wireless sensors can be utilized for geofencing (i.e., deploying a sensor network as a transparent fence to protect an area against unauthorized trespassing), enemy tracking, and battlefield exploration and condition assessment particularly in hazardous environments. In military intelligence applications, the small form-factor, reliability, interoperability and durability of the sensor nodes under severe environmental conditions are particularly critical requirements.

Asset Management

Businesses with large and high-turnover inventories of assets (such as construction companies, utility companies and trucking companies) can benefit from automated asset management systems in improving the utilization of their resources [10]. With automated asset management, sensor networks are deployed to collect real-time data about exact location and condition of an inventory of assets automatically. The collected data provides the opportunity for real-time analysis of the resource usage, which in turn enables timely and optimal decision-making on handling, supply, delivery, storage and other asset management tasks. Various types of sensing devices, such as GPS devices and passive RFID (Radio-Frequency Identification) tags, are applicable with the sensor networks used for asset monitoring.

Building Monitoring

The recent attempt aiming at optimizing the energy performance of the buildings by deep sensing of the building conditions, dubbed BIM (Building Information Modeling) (Federal BIM Program. URL: http://www.gsa.gov/bim/), heavily relies on the sensor network technology. With BIM energy tools enabled by sensing networks, one can monitor, e.g., the temperature and lighting conditions in the building, and accordingly regulate the heating and cooling systems, ventilators and lights dynamically for best energy performance [11]. Also as a safety tool, building sensor networks can detect and report threats, such as existence of the biological agents in the environment as well as physical intrusions.

Automotive

With the new standards such as Dedicated Short-Range Communication (DSRC) designated for vehicle communications, in the near future, cars will be able to communicate information to each other and to the roadside infrastructures. With this capability, while in traffic cars can form a so-called vehicular sensor network, where each car equipped with sensing devices (e.g., camera, thermometer, etc.) acts as a mobile sensor node. In a vehicular sensor network, cars can share information and analyze the aggregate information about the road conditions, congestions, nearby emergencies, etc., for applications such as collision prevention, congestion avoidance and flow optimization [8].

Healthcare

Sensor networks can effectively improve the accuracy of the patient care and, consequently, the safety of the patients when they become physically incapacitated and require immediate medical attention [7]. Sensor networks allow this by enabling close and automated monitoring of the patient's vital signs. When monitoring is coupled with real-time analysis of the signs, the sensor-enabled healthcare system can alert the right person at the right time to attend to the patient. Such healthcare systems are applicable both at homes of the elderly and at the hospitals.

Industrial Monitoring

Sensors can be used to monitor industrial processes for safety as well as manufacturing optimization [4]. One can also deploy sensors to monitor the condition of the industrial equipments for preventative maintenance and also safety of the operators. Wireline sensors have been in use for a long time in various industry sectors such as oil companies (both upstream and downstream) and chemical plants. Wireless technology and inexpensive sensors has greatly facilitated and extended the use of the sensing networks for process and equipment monitoring, encouraging oil companies, e.g., to develop smart oilfields by equipping the oil wells and other assets with wireless sensors (see e.g., [2]).

Future Directions

With the current trend, sensor networks are being applied with increasingly more complex, large-scale and distributed systems (e.g., the federal intelligent transportation system (Federal ITS Program. URL: http://www.its.dot.gov/)). Such applications demand deployment of large-scale sensor networks and, accordingly, require fully decentralized solutions for sensor data management to achieve scalability.

Data Sets

- CENS data sets. URL: http://research.cens.ucla.edu/portal/page?_pageid = 59,54414&_dad = portal&_schema = PORTAL
- Intel lab data set. URL: http://db.csail.mit.edu/labdata/labdata.html
- Precipitation data set. URL: http://www.jisao.washington.edu/data_sets/widmann/
- IHOP data set. URL: http://www.eol.ucar.edu/rtf/projects/ihop_2002/spol/

Cross-references

▶ Sensor Networks
▶ Continuous Queries in Sensor Networks
▶ Ad-hoc Queries in Sensor Networks
▶ In-Network Query Processing
▶ Data Acquisition and Dissemination in Sensor Networks
▶ Data Aggregation in Sensor Networks
▶ Data Storage and Indexing in Sensor Networks
▶ Data Uncertainty Management in Sensor Networks
▶ Database Languages for Sensor Networks

Recommended Reading

1. Bonnet P., Gehrke J., and Seshadri P. Towards sensor database systems. In Proc. 2nd Int. Conf. on Mobile Data Management, 2001, pp. 3–14.
2. The Center for Interactive Smart Oilfield Technologies. URL: http://cisoft.usc.edu/
3. Cerpa A., Elson J., Estrin D., Girod L., Hamilton M., and Zhao J. Habitat monitoring: Application driver for wireless communications technology. In Proc. SIGCOMM Workshop on Data Communications in Latin America and the Caribbean, 2001.
4. Chong C. and Kumar S.P. Sensor networks: Evolution, opportunities, and challenges. In Proc. IEEE. 91(8), 2003.
5. Culler D., Estrin D., and Srivastava M. Sensor network applications (Cover Feature). IEEE Computer, 37(8), 2004.
6. Estrin D., Govindan R., Heidemann J., and Kumar S. Next century challenges: Scalable coordination in sensor networks. In Proc. 5th Annual Int. Conf. on Mobile Computing and Networking (MOBICOM), 1999, pp. 263–270.
7. Ho L., Moh M., Walker Z., Hamada T., and Su C. A prototype on RFID and sensor networks for elder healthcare. In Proc. 2005 ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis. Pennsylvania, 2005, pp. 70–75.
8. Lee U., Magistretti E., Zhou B., Gerla M., Bellavista P., and Corradi A. MobEyes: Smart mobs for urban monitoring with a vehicular sensor network. IEEE Wireless Commun, 13(5), 2006.
9. Nemeroff J., Garcia L., Hampel D., and DiPierro S. Application of sensor network communications. In Proc. Military Communications Conference, 2001, pp. 336–341.
10. RFID Journal. URL: http://www.rfidjournal.com/
11. Schmid T, Dubois-Ferrière H., and Vetterli M. SensorScope: Experiences with a wireless building monitoring sensor network.

In Proc. Workshop on Real-World Wireless Sensor Networks, June 2005.

12. Sharifzadeh M. and Shahabi C. Utilizing Voronoi cells of location data streams for accurate computation of aggregate functions in sensor networks. GeoInformatica, 10(1), 2006.
13. Yoon S. and Shahabi C. The Clustered AGgregation (CAG) technique leveraging spatial and temporal correlations in wireless sensor networks. ACM Trans Sensor Netw, 3(1), 2007.
14. Zhao F. and Guibas L. Wireless Sensor Networks: An information processing approach. First Edition, Morgan Kaufmann, San Francisco, CA., July 2004.
15. Center for Embedded Networked Sensing. URL: http://www.cens.ucla.edu/.

Application-to-Application Integration

▶ Enterprise Application Integration

Approximate Queries in Peer-to-Peer Systems

Wolf Siberski, Wolfgang Nejdl
L3S Research Center, University of Hannover,
Hannover, Germany

Synonyms

Top-*k* queries in P2P systems; Aggregate queries in P2P systems

Definition

Peer-to-peer (P2P) networks enable the interconnection of a huge amount of information sources without imposing costs for a central coordination infrastructure. Due to the dynamic and self-organizing nature of such networks, it is not feasible to guarantee completeness and correctness as in traditional distributed databases. Therefore, P2P systems are usually applied in areas where approximate query evaluation, i.e., the computation of a nearly complete and correct answer set, is sufficient. As the most frequent application of querying in P2P is search, many of these algorithms fall into the class of top-*k* query algorithms. Another important case is the approximation of aggregate query results.

Historical Background

P2P networks use approximate querying from the outset. In Gnutella, an unstructured network, the query is distributed in a limited neighborhood only, thus the result is usually not complete. The early top-*k* query algorithms for P2P are based on such unstructured networks. PlanetP [6], a P2P network for information retrieval, employs gossiping to replicate index information among all peers and sends queries to the best peers according to this replicated index. This approach has been extended to a super-peer network in [11]. While the first Distributed Hash Table (DHT) systems such as CHORD and Pastry aim at complete and correct answers, later structured networks approximate the result set, especially P2P networks for information retrieval [2,15]. Frequently, approximate P2P networks build upon algorithms for distributed databases or distributed IR. For example, for source selection PlanetP relies on an extended version of GlOSS [7], Minerva [15] on CORI [4]. Odissea [13] uses either Fagins Algorithm (FA) or a Threshold Algorithm (TA) as top-*k* algorithm, KLEE [9] extends TA, etc. (→*Top-k Selection Queries on Multimedia Datasets*).

Foundations

The main challenge in P2P approximate query processing is to select the optimal subset of peers to which the query is forwarded. However, the selection criteria for this subset are completely different for top-*k* and aggregate queries. While in top-*k* the goal is to identify the peers holding top objects, approximate aggregate querying algorithms need to find a representative sample of peers. In both cases, each selected peer evaluates the query locally; the respective responses are collected and merged to compute the final result set.

Approximation for Top-*k* Queries

Regardless of the chosen network topology, all distributed top-*k* algorithms consist of the following elements [14]:

Indexing. Determines what is indexed and how index information is collected within the network.

Source selection. Determines the peers a query is sent to.

Result merging. Determines how local result lists are merged to form the final result set.

Unstructured Networks In an unstructured P2P network peers form a random graph. In these networks, the only available routing strategy is *filtered flooding*, i.e., forwarding the query to selected neighbors. For effective filtering, the peers maintain content indexes. For each neighbor peer, the index allows to look up which kind of content is reachable via this peer. In the case of information retrieval, the index holds term frequencies of these subnets. This index information is built by gossiping: each peer periodically sends the content summaries it holds to its neighbors, where they are merged with the index. Thus, over time each peer gathers more and more accurate information about the whole network. Frequently, bloom filters or hash sketches are used to represent the content summaries. The typical representative for this approach is PlanetP [6]. Evaluations have shown that this algorithm type only scales to several thousands of peers, due to the limitations of filtered flooding. Even with complete index information, the query still usually has to be sent to a high fraction of all peers to reach the peers holding the top-*k* objects. Also, gossiping induces rather high index maintenance costs.

Hierarchical Networks Hierarchical topologies can overcome some of these limitations. In these topologies, particularly powerful peers form a super-peer backbone. Information sources are not connected with each other, but always assigned to one of the super-peers. This topology is especially suited for adaptation of traditional distributed top-*k* algorithms: each super-peer acts as coordinating node for its peers. In some systems the peers form a tree-shaped network; this has the advantage that the same aggregation algorithm can be used up to the root peer, but at the price of extremely uneven load distribution. Therefore, the usual approach is to restrict the hierarchy to two levels. In this case, filtered flooding is used to distribute queries within the super-peer backbone. Maintaining the index independently from the actual queries can impose a high overhead; this can be avoided by building a *query-driven index* [2].

Structured Networks Arguably the most efficient peer-to-peer networks are DHTs, where peers form highly structured network topologies. However, they only provide the usual hash table feature, storage and retrieval by key. The DHT maintains lists of top peers for each feature [3,13]. As in the case of hierarchical peers, the index can be improved by considering query statistics [12]. For a query, first these lists are retrieved, and then an established algorithm such as CORI or GlOSS is used for source selection. To retrieve the top-*k* objects, the TA family of algorithms are the state of the art. TPUT is especially suitable because it limits the retrieval process to three phases: first, the query initiator determines a lower bound for the object score by requesting scores of the top-*k* objects at each peer. Second, the initiator requests all objects having at least the threshold score. It is guaranteed that all top-*k* objects are in the returned sets. Finally, the initiator determines these objects and requests the actual content. TPUT has been evolved to an approximate algorithm with probabilistic guarantees in [9].

Approximation for Aggregate Queries

Efficient approximation of aggregate queries in unstructured P2P networks can be done by sampling. Starting at the peer issuing the query, the query travels along a random path to gather the sample. The challenge is to choose this path such that the query is indeed received by a uniform sample of the network. Note that the standard approach (*Markov-Chain random walk*) of selecting each outgoing edge with equal probability does not yield a uniform sample, but favors nodes with high degrees. This can be approximately compensated by scaling down the local peer value with the probability of this peer being selected. To reduce errors due to clustering within the network, the random walk can be modified such that only each *i*th peer on the path is considered for the sample. [1] shows how usual aggregate queries (COUNT, SUM, AVG) can be computed in this way with low error rates. While gossiping also has be proposed to compute aggregates in unstructured networks, this method does not scale to large networks [10].

An efficient method to compute COUNT queries in DHT networks has been proposed in [10]. This approach relies on locally computed hash sketches which are inserted into the DHT. For hash sketches of length k the actual counting requires $O(k)$ DHT lookups, resulting in $O(k \cdot logn)$ messages.

Key Applications

Top-*k* queries are used in distributed information retrieval scenarios, such as digital library networks.

An important application for aggregate queries in massively distributed networks is the gathering of network statistics, e.g., to identify security risks or to monitor performance [8]. Approximate aggregate queries are also gaining importance in the area of sensor network [5], where limitations of the sensor hardware (processor, memory, power supply) are key factors for query algorithm design.

Cross-references

▶ Approximate XML Querying
▶ Peer Data Management System
▶ Top-K Selection Queries on Multimedia Datasets

Recommended Reading

1. Arai B., Das G., Gunopulos D., and Kalogeraki V. Efficient approximate query processing in peer-to-peer networks. IEEE Trans. Knowl. Data Eng., 19(7):919–933, 2007.
2. Balke W.T., Nejdl W., Siberski W., and Thaden U. Progressive distributed top-*k* retrieval in peer-to-peer networks. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 174–185.
3. Bender M., Michel S., Triantafillou P., Weikum G., and Zimmer C. MINERVA: collaborative P2P search. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 1263–1266.
4. Callan J.P., Lu Z., and Croft W.B. Searching distributed collections with inference networks. In Proc. 18th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 21–28.
5. Chu D., Deshpande A., Hellerstein J.M., and Hong W. Approximate data collection in sensor networks using probabilistic models. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 48.
6. Cuenca-Acuna F.M., Peery C., Martin R.P., and Nguyen R.D. Planet P: using gossiping to build content addressable peer-to-peer information sharing communities. In Proc. 12th IEEE Int. Symp. on High Performance Distributed Computing, 2003, pp. 236–246.
7. Gravano L., Garcia-Molina H., and Tomasic A. GlOSS: Text-source discovery over the internet. ACM Trans. Database Syst., 24(2):229–264, 1999.
8. Hellerstein J.M., Condie T., Garofalakis M.N., Loo B.T., Maniatis P., Roscoe T., and Taft N. Public health for the internet (PHI). In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 332–340.
9. Michel S., Triantafillou P., and Weikum G. Klee: A framework for distributed top-k query algorithms. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 637–648.
10. Ntarmos N., Triantafillou P., and Weikum G. Counting at large: Efficient cardinality estimation in internet-scale data networks. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 40.
11. Seshadri S. and Cooper B.F. Routing queries through a peer-to-peer infobeacons network using information retrieval techniques. IEEE Trans. Parallel Distrib. Syst., 18(12):1754–1765, 2007.
12. Skobeltsyn G., Luu T., Podnar Z.I., Rajman M., and Aberer K. Web text retrieval with a P2P query-driven index. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 679–686.
13. Suel T., Mathur C., Wu J., Zhang J., Delis A., Kharrazi M., Long X., and Shanmugasundaram K. ODISSEA: A peer-to-peer architecture for scalable web search and information retrieval. In Proc. 6th Int. Workshop on the World Wide Web and Databases, 2003, pp. 67–72.
14. Yu C., Philip G., and Meng W. Distributed top-n query processing with possibly uncooperative local systems. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 117–128.
15. Zimmer C., Tryfonopoulos C., and Weikum G. MinervaDL: An architecture for information retrieval and filtering in distributed digital libraries. In Proc. 11th European Conf. on Research and Advanced Technology for Digital Libraries, 2007, pp. 148–160.

Approximate Query Answering

▶ Approximate Query Processing

Approximate Query Processing

Qing Liu
CSIRO, Hobart, TAS, Australia

Synonyms

Approximate query answering

Definition

Query processing in a database context is the process that deduces information that is available in the database. Due to the huge amount of data available, one of the main issues of query processing is how to process queries efficiently. In many cases, it is impossible or too expensive for users to get exact answers in the short query response time. Approximate query processing (AQP) is an alternative way that returns approximate answer using information which is similar to the one from which the query would be answered. It is designed primarily for aggregate queries such as count, sum and avg, etc. Given a SQL aggregate query Q, the accurate answer is y while the approximate answer is y'. The relative error of query Q can be quantified as:

$$Error(Q) = |\frac{y - y'}{y}|. \qquad (1)$$

The goal of approximate query processing is to provide approximate answers with acceptable accuracy in orders of magnitude less query response time than that for the exact query processing.

Historical Background

The earliest work on approximate answers to decision support queries appears in Morgenstein's dissertation from Berkeley [8]. And the approximate query processing problem has been studied extensively in the last 10 years. The main motivations [3] which drive the techniques being developed are summarized as follows.

First, with the advanced data collection and management technologies, nowadays there are a large number of applications with data sets about gigabytes, terabytes or even petabytes. Such massive data sets necessarily reside on disks or tapes, making even a few accesses of the base data sets comparably slow. In many cases, precision to "last decimal" is not required for a query answer. Quick approximation with some error guarantee (e.g., the resident population in Australia 21,126,700 +/− 200) is adequate to provide insights about the data.

Second, decision support system (DSS) and data mining are popular approaches to analyzing large databases for decision making. The main characteristic of the DSS is that aggregation queries (e.g., count, sum, avg, etc.) are executed on large portion of the databases, which can be very expensive and resource intensive even for a single analysis query. Due to the exploratory nature of decision making, iterative process involves multiple query attempts. Approximate answers with fast response time gives users the ability to focus on their explorations and quickly identify truly interesting data. It provides a great scalability of the decision support applications.

Third, approximate query processing is also used to provide query preview. In most cases, users are only interested in a subset of the entire database. Given a trial query, query preview provides an overview about the data distribution. The users can preview the number of hits and refine the queries accordingly. This prevents users from fruitless queries such as zero-hits or mega-hits. Figure 1 shows an example of query preview interface of NASA EOSDIS (Earth Observing System Data and Information System) project, which is attempting to provide online access to a rapidly growing archive of scientific earth data about the earth's land, water, and air. In the query preview, users select rough ranges for three attributes: area, topic (a menu list of parameters such as atmosphere, land surface, or oceans) and temporal coverage. The number of data sets for each topic, year, and area is shown on preview bars. The result preview bar, at the bottom of the interface, displays the total approximate number of data sets which satisfy the query.

Finally, sometimes network limitation or disk storage failure would cause the exact answers unaffordable or unavailable. An alternative solution is to provide an approximate answer based on the local cached data synopsis.

Foundations

Due to the acceptability of approximate answers coupled with the necessity for quick query response time, approximate query processing has emerged as a cost effective approach for dealing with the large amount of data. This speed-up is achieved by answering queries based on samples or other synopses (summary) of data whose size is orders of magnitude smaller than that of the original data sets.

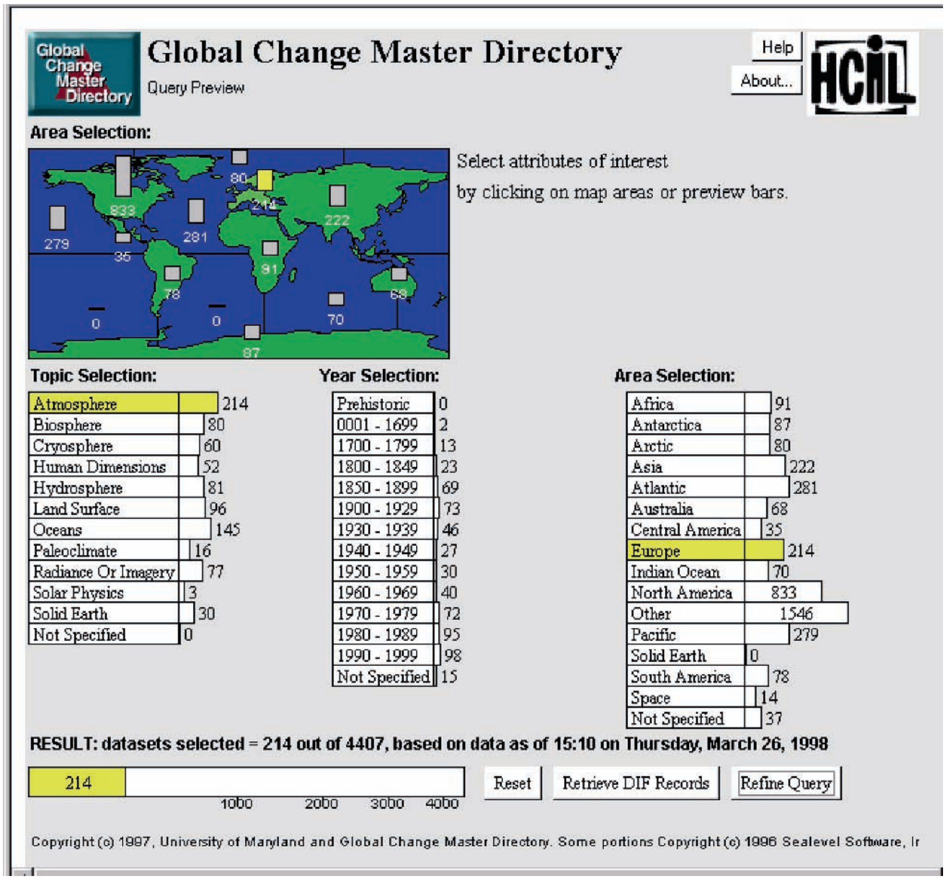
Ioannidis presented the generic flow of approximate query process (Fig. 2) in [5]. Here, data analysis is the overall approach which derives the synopsis from original data to approximate the underlying data distribution. Typically, the *algorithm* partitions the data based on the *distance* function into groups of similar elements, called *buckets*, clusters, patterns, or several other names. The *data elements* that fall in each bucket are then represented by the synopses for *approximation use* which corresponds to the actual purpose of the whole data approximation process. The quality of approximation result can be measured by the *distance* between the synopses and the original data.

There are two basic approaches to achieve approximate query processing: pre-computed synopsis and online query processing.

Pre-Computed Synopsis

Approximate query processing using pre-computed synopsis includes two steps: construct synopsis prior to query time, answer query approximately using synopsis at query time.

To provide high accurate query answer, the key issue to construct synopsis is how to represent the underlying data distribution precisely and compactly. Generally, the data distribution can be classified into two groups: uniform distribution and non-uniform distribution. The synopsis for uniform data distribution assumes the objects are distributed uniformly in the data space. For point objects locating in two-dimensional



Global Change Master Directory

Query Preview

Area Selection:

Select attributes of interest by clicking on map areas or preview bars.

Topic Selection:

Atmosphere	214
Biosphere	80
Cryosphere	60
Human Dimensions	52
Hydrosphere	81
Land Surface	96
Oceans	145
Paleoclimate	16
Radiance Or Imagery	77
Solar Physics	3
Solid Earth	30
Not Specified	0

Year Selection:

Prehistoric	0
0001 - 1699	2
1700 - 1799	13
1800 - 1849	23
1850 - 1899	69
1900 - 1929	73
1930 - 1939	46
1940 - 1949	27
1950 - 1959	30
1960 - 1969	40
1970 - 1979	72
1980 - 1989	95
1990 - 1999	98
Not Specified	15

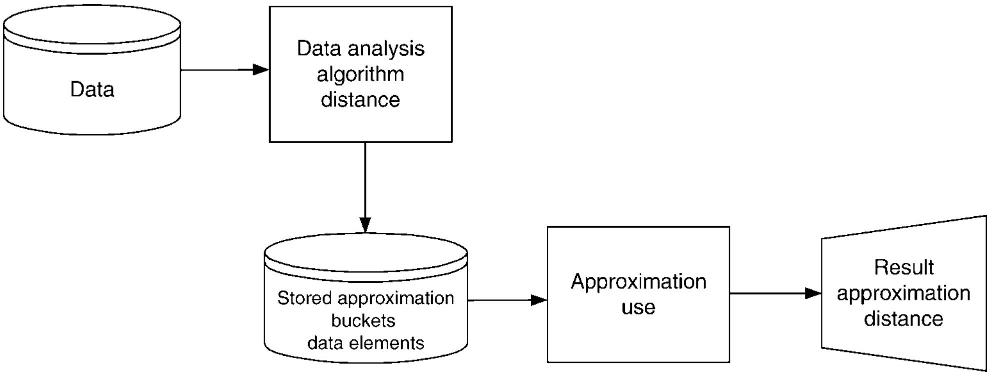
Area Selection:

Africa	91
Antarctica	87
Arctic	80
Asia	222
Atlantic	281
Australia	68
Central America	35
Europe	214
Indian Ocean	70
North America	833
Other	1546
Pacific	279
Solid Earth	0
South America	78
Space	14
Not Specified	37

RESULT: datasets selected = 214 out of 4407, based on data as of 15:10 on Thursday, March 26, 1998

Copyright (c) 1997, University of Maryland and Global Change Master Directory. Some portions Copyright (c) 1996 Sealevel Software, Ir

Approximate Query Processing. Figure 1. NASA EOSDIS interface of query preview.



Data
Data analysis algorithm distance
Stored approximation buckets data elements
Approximation use
Result approximation distance

Approximate Query Processing. Figure 2. Generic flow of approximation process.

space $[{U^x}_{min}, {U^y}_{min}] \times [{U^x}_{max}, {U^y}_{max}]$, the query result size is estimated as $N \times area(Q)/(({U^x}_{max} - {U^x}_{min}) \times ({U^y}_{max} - {U^y}_{min}))$, where N is the data set size and $area(Q)$ is the area of window query Q.

There are various techniques developed for non-uniform data distribution. They can also be divided into two groups: parametric and non-parametric. The parametric techniques try to use parameters to catch the original data distributions. Although the models can summarize data distributions with a few descriptive parameters, if the underlying data do not follow any known distributions, or their linear combinations, the model fitting techniques produce inferior results.

The non-parametric techniques use different approaches to summarize the data distributions. Generally, it is possible to classify these techniques into three categories according to the strategies adopted:

1. Sampling techniques
2. Histogram techniques
3. Wavelet techniques

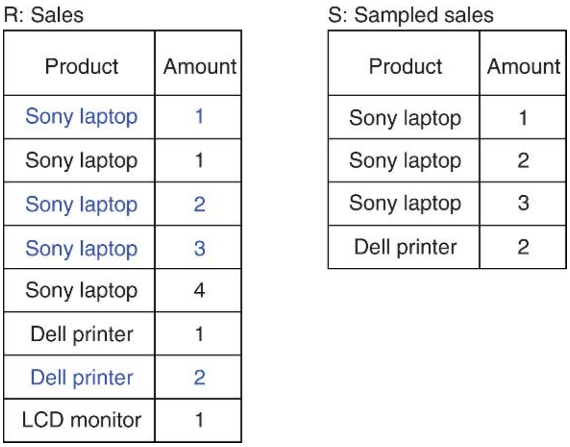
Sampling The basic idea of sampling is that a small random sample of the data often well-represent all the data. Therefore, query would be answered based on the pre-sampled small amount of data and then scaled up based on the sample rate. Figure 3 shows an example where 50% of data are sampled during the pre-computed stage. Given a query "how many Sony laptops are sold in *R*", the approximate result is "select 2 * sum(*) from *S* where *S.product* = '*SonyLaptop*'", which is 12. In *R*, the exact answer is 11.

The main issue of sampling method is to decide what sample criteria should be used to select data. The sampling techniques are classified into the following groups [1]:

1. *Uniform sampling*. Data is sampled uniformly
2. *Biased sampling*. A non-uniform random sample is pre-computed such that parts of the database deemed "more important" than the rest
3. *Icicles*. A biased sampling technique that is based on known workload information
4. *Outlier indexing*. Indexing outliers and biased sampling the remaining data
5. *Congressional sampling*. Targeting group by queries with aggregation and trying to maximize the accuracy for all groups (large or small) in each group-by query
6. *Stratified sampling*. Generalization of outlier indexing, Icicles and congressional sampling. It targets minimizing error in estimation of aggregates for the given workload

Sample-based procedures are robust in the presence of correlated and nonuniform data. Most importantly, sampling-based procedures permit both assessment and control of estimation errors. The main disadvantage of this approach is the overhead it adds to query optimization. Furthermore, join operation could lead to significant quality degradations because join operator applied on two uniform random sample can result in a non-uniform sample of the join result which contains very few tuples.

Histograms Histogram techniques are the most commonly used form of statistics in practice (e.g., they are used in DB2, Oracle and Microsoft SQL Server). This is because they incur almost no run-time overhead and



R: Sales

Product	Amount
Sony laptop	1
Sony laptop	1
Sony laptop	2
Sony laptop	3
Sony laptop	4
Dell printer	1
Dell printer	2
LCD monitor	1

S: Sampled sales

Product	Amount
Sony laptop	1
Sony laptop	2
Sony laptop	3
Dell printer	2

Approximate Query Processing. Figure 3. Example of sampling.

produce low-error estimates while occupying reasonably small space.

The basic idea is to partition attribute value domain into a set of buckets and query is answered based on the buckets. The main issues of histogram construction and query are as follows:

1. How to partition data into bucket
2. How to represent data in each bucket
3. How to estimate answer using the histogram

For one-dimensional space, a histogram on an attribute X is constructed by partitioning the data distribution of X into B ($B \geq 1$) buckets and approximating the frequencies and values in each bucket. Figure 4a is an example of original data set and Figure 4b shows its data distribution. Figure 4c is an example of histogram constructed accordingly, where $B = 3$.

If there are several attributes involved in a query, a multi-dimensional histogram is needed to approximate the data distribution and answer such a query. A multi-dimensional histogram on a set of attributes is constructed by partitioning the joint data distribution of the attributes. They have the exact same characteristics as one-dimensional histograms, except that the partition rule needs to be more intricate and cannot always be clearly analyzed because there cannot be ordering in multiple dimensions [9].

To represent data in each bucket, it includes value approximation and frequency approximation. Value approximation captures how attribute values are approximated within a bucket. And frequency approximation captures how frequencies are approximated within a bucket.

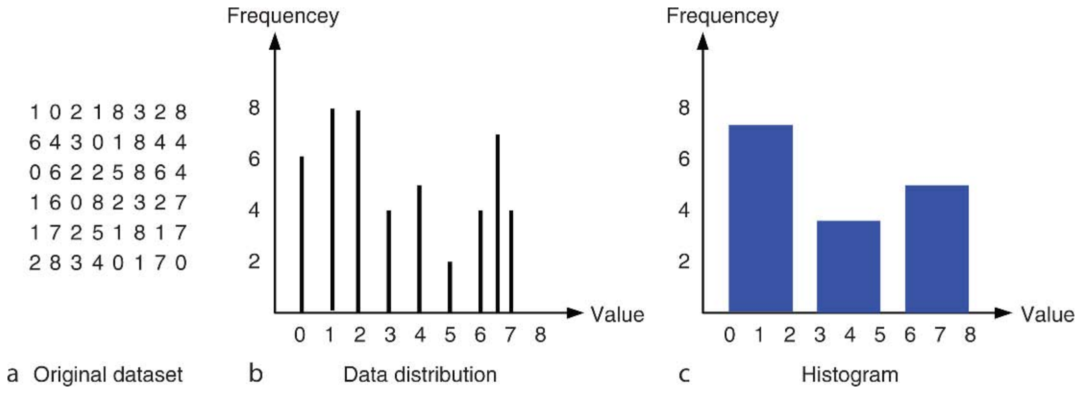
The two main approaches for value approximation are *continuous value assumption* and *uniform spread assumption* [10]. Continuous value assumption only maintains min and max value without indication of how many values there are or where they might be. Under the uniform spread assumption, one also maintain the number of values within each bucket and approximates the actual value set by the set that is formed by (virtually) placing the same number of values at equal distances between the min and max value in multi-dimensional space [6].

With respect to frequency approximation, almost all work deal with *uniform distribution assumption*.

The benefit of a histogram synopsis is that it can be easily used to answer many query types, including the aggregate and non-aggregate queries. However, one of the issues of histogram approach is it is hard to calculate a theoretical error bound. Thus the evaluations on the histogram synopsis usually rely heavily on the experiment results. Further more, histogram-based approaches become problematic when dealing with the high-dimensional data sets that are typical for modern decision support applications. This is because as the dimensionality of the data increases, both the storage overhead (i.e., number of buckets) and the construction cost of histograms that can achieve reasonable error rates increase in an explosive manner.

Wavelet Wavelet is a mathematical tool for hierarchical decomposition of functions using recursive pairwise averaging and differencing at different resolutions. It represents a function in terms of a coarse overall shape, plus details that range from broad to narrow. It is widely used in the signal and image processing.

Matias et al. [7] first proposed the use of Haar-wavelet coefficients as synopsis for estimating the selectivity of window queries. The basic idea is to apply wavelet decomposition to the input data collection to



1 0 2 1 8 3 2 8
6 4 3 0 1 8 4 4
0 6 2 2 5 8 6 4
1 6 0 8 2 3 2 7
1 7 2 5 1 8 1 7
2 8 3 4 0 1 7 0

Approximate Query Processing. Figure 4. Example of histogram.

obtain a compact data synopsis that comprises a select small collection of wavelet coefficients. Figure 5 shows an example of hierarchical decomposition tree of Haar-wavelet. The leaf nodes are original data and non-leaf nodes are wavelet coefficients generated by averaging and differencing from their two children.

Later, the wavelet concept was extended to answer more general approximate queries. The results of recent studies have clearly shown that wavelets can be very effective in handling aggregates over high-dimensional online analytical processing (OLAP) cubes, while avoiding the high construction costs and storage overheads of histogram techniques.

Another important part of the above three technologies for approximate query processing is synopsis maintenance. If the data distribution is not changed significantly, the data synopsis would be updated accordingly to reflect such change. Otherwise, a new data synopsis will be constructed and discard the old one. Refer to the specific techniques for more details.

There are a few other work that do not belong to the above three categories to approximate the underlying data distribution. For example, recently Das et al. [2] present a framework that is based on randomized projections. This is the first work in the context of spatial database which provides probability quality guarantees with respect to query result size approximation.

Online Query Processing

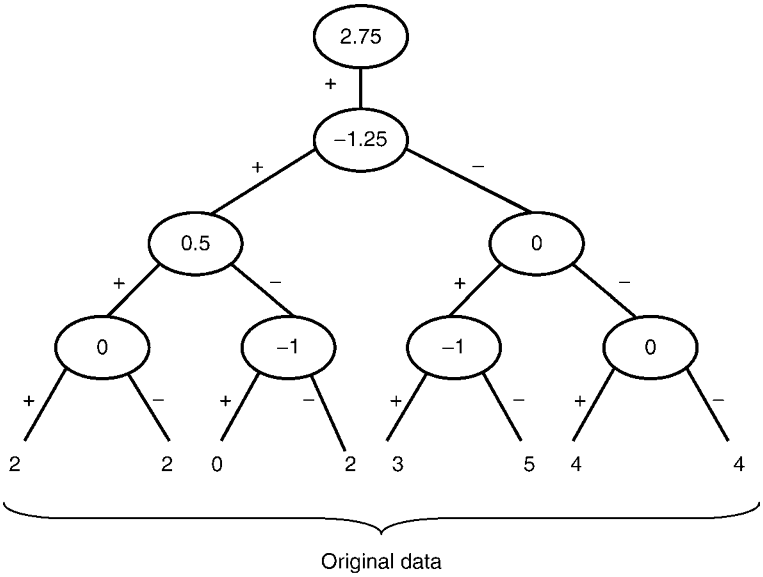
The motivation of online query processing is that the data analysis is to extract unknown information from data. It is an iterative process starting by asking broad questions and continually refining them based on approximate or partial results. Therefore, instead of optimizing for query response time, it needs to balance two conflicting performance goals: minimizing uneventful "dead time" between updates for the users, while simultaneously maximizing the rate at which partial or approximate answers approach a correct answer. Refer to [4] for details.

Compared with pre-computed synopses approach, the advantages of online query processing approach is it does not require pre-processing and progressive refinement of approximate results at runtime can quickly lead to a satisfied results. However, the main obstacles for this technique to be practical is it significantly changes the query processor of current commercial query processing system which is not desirable.

Key Applications

AQP in Relational Data Management

The AQUA system proposed by the Bell lab can support various kinds of approximate queries over the relational database.



Approximate Query Processing. Figure 5. Example of Haar-wavelet hierarchical decomposition.

AQP in Spatial Data Management

Alexandria Digital Library allows user to define spatial queries and returns the approximate number of hits quickly as the initial result and then user can refine the query accordingly.

AQP in Stream Data Management

MIT proposed Aurora as a data stream management system, which virtually supports answering approximate queries over the data stream.

AQP in Sensor Network

The techniques of TinyDB system, proposed by Massachusetts Institute of Technology and UC Berkeley, can lead to orders of magnitude improvements in power consumption and increased accuracy of query results over non-acquisitional systems that do not actively control when and where data is collected.

AQP in Semantic Web Search

For semantic web search, the viewpoints of users performing a web search, ontology designers and annotation designers do not match. This leads to missed answers. Research studies have shown AQP is of prime importance for efficiently searching the Semantic Web.

Cross-references

▶ Aggregate Queries in P2P Systems
▶ Data Mining
▶ Decision Support
▶ Histogram
▶ On-Line Analytical Processing
▶ Query Optimization
▶ Sampling
▶ Selectivity
▶ Wavelets on Streams

Recommended Reading

1. Das G. Sampling methods in approximate query answering systems. In Invited Book Chapter, Encyclopedia of Data Warehousing and Mining, John Wang (ed.). Information Science Publishing, 2005.
2. Das A., Gehrke J., and Riedewald M. Approximation Techniques for Spatial Data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 695–700.
3. Garofalakis M. and Gibbons P. Approximate Query Processing: Taming the TeraBytes: A Tutorial. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
4. Hellerstein J., Haas P., and Wang H. Online Aggregation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 171–182.
5. Ioannidis Y. Approximation in Database Systems. In Proc. 9th Int. Conf. on Database Theory, 2003, pp. 16–30.
6. Ioannidis Y. The History of Histograms (abridged). In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 19–30.
7. Matias Y., Vitter J., and Wang M. Wavelet Based Histograms for Selectivity Estimation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 448–459.
8. Morgenstein J. Computer Based Management Information Systems Embodying Answer Accuracy as a User Parameter. PhD Thesis, U.C. Berkeley, 1980.
9. Poosala V. and Ioannidis Y. Selectivity Estimation Without the Attribute Value Independence Assumption. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 466–475.
10. Poosala V., Ioannidis Y., Haas P., and Shekita E. Improved Histograms for Selectivity Estimation of Range Predicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 294–305.

Approximate Querying

▶ Model-Based Querying in Sensor Networks

Approximate Reasoning

Vilém Novák
University of Ostrava, Ostrava, Czech Republic

Definition

Approximate reasoning is a deduction method which makes it possible to derive a conclusion on the basis of imprecisely characterized situation (quite often using linguistically specified fuzzy IF-THEN rules) and a new information that can also be imprecise. The basic scheme of approximate reasoning is the following:

$$\begin{array}{lcl} \text{Condition} & : & \text{IF } X \text{ is } \mathcal{A}_1 \text{ THEN } Y \text{ is } \mathcal{B}_1 \\ & & \dots\dots\dots\dots\dots\dots\dots\dots \\ & & \text{IF } X \text{ is } \mathcal{A}_m \text{ THEN } Y \text{ is } \mathcal{B}_m \\ \text{Premise} & : & X \text{ is } \mathcal{A}' \\ \hline \text{Conclusion} & : & Y \text{ is } \mathcal{B}' \end{array} \tag{1}$$

where "Condition" is a linguistic description consisting of a set of fuzzy/linguistic IF-THEN rules and $\mathcal{A}'$ is a possible modification of antecedent of some of the former rules. For example, "X is small" can be replaced by "X is very small."

Key Points

The mathematical model of approximate reasoning depends on the way how the linguistic description forming the condition is interpreted (see FUZZY/LINGUISTIC IF-THEN RULES AND LINGUISTIC DESCRIPTIONS).

Let X is $\mathcal{A}'$ be interpreted by a fuzzy set $A' \underset{\sim}{\subseteq} U$. If "Condition" is interpreted by a fuzzy relation $R \underset{\sim}{\subseteq} U \times V$ then the result of approximate reasoning is a fuzzy set $B' \underset{\sim}{\subset} V$ which interprets "Y *is* $\mathcal{B}'$" and which is obtained using the formula

$$B'(y) = \bigvee_{x \epsilon U} (A'(x) \otimes R(x,y)) \qquad (2)$$

where $\otimes$ is a t-norm (product in residuated lattice).

Alternative approximate reasoning method is *perception-based logical deduction*. Its idea consists of finding *perception* of the given measured value of the input $X = x_0$. The perception is an evaluative expression occurring among $\mathcal{A}_1, \ldots, \mathcal{A}_m$ that fits x_0 in the best way. Then the corresponding fuzzy IF-THEN rule is fired and the proper output is derived. More details can be found in [2,3].

Cross-references

► Fuzzy/Linguistic IF-THEN Rules and Linguistic Descriptions
► Fuzzy Relation
► Fuzzy Set
► Triangular Norms

Recommended Reading

1. Klir G.J. and Yuan B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice-Hall, New York, 1995.
2. Novák V. and Lehmke S. Logical structure of fuzzy IF-THEN rules. Fuzzy Sets Syst., 157:2003–2029, 2006.
3. Novák V. and Perfilieva I. On the semantics of perception-based fuzzy logic deduction. Int. J. Intell. Syst., 19:1007–1031, 2004.
4. Novák V., Perfilieva I., and Močkoř J . Mathematical Principles of Fuzzy Logic. Kluwer, Boston/Dordrecht, 1999.

Approximate XML Querying

► Ranked XML Processing

Approximation of Frequent Itemsets

JINZE LIU
University of Kentucky, Lexington, KY, USA

Synonyms

AFI

Definition

Consider an $n \times m$ binary matrix D. Each row of D corresponds to a transaction t and each column of D corresponds to an item i. The (t, i)-element of D, denoted $D(t, i)$, is 1 if transaction t contains item i, and 0 otherwise. Let $T_0 = \{t_1, t_2, \ldots, t_n\}$ and $I_0 = \{i_1, i_2, \ldots, i_m\}$ be the set of transactions and items associated with D, respectively.

Let D be as above, and let $\varepsilon_r, \varepsilon_c \in [0,1]$. An itemset $I \subseteq I_0$ is an approximate frequent itemset AFI(ε_r, ε_c), if there exists a set of transactions $T \subseteq T_0$ with $|T| \geq minsup|T_0|$ such that the following two conditions hold:

1. $\forall i \in T, \dfrac{1}{|I|} \sum_{j \in I} D(i,j) \geq (1 - \epsilon_r);$
2. $\forall j \in I, \dfrac{1}{|T|} \sum_{i \in T} D(i,j) \geq (1 - \epsilon_c);$

Historical Background

Relational databases are ubiquitous, cataloging everything from market-basket data [1] to genomic data collected in biological experiments [2]. A binary matrix is one common representation of relational databases. Rows in the matrix often correspond to the objects, while columns represent attributes of the objects. The binary value of each matrix entry indicates the presence (1) or absence (0) of an attribute in the object. For example, in a market-basket database, rows represent transactions, columns represent product items, and a binary entry indicates whether an item is contained in the transaction [1]. Frequent itemset mining [1] is a key technique in the analysis of such data. In the binary representation, a *frequent itemset* corresponds to a sub-matrix of 1s, where the itemset (the set of columns) are supported by a sufficiently large number of transactions (set of rows).

While frequent itemsets and the algorithms to generate them have been well studied, the problem is that the data in real application is often imperfect. In a transaction database, frequent itemsets might be obscured by the vagaries of the market and human behaviors. Items expected to be purchased together by a customer might not appear together in some transactions when one of them is out of stock in the market or overstocked by the customer. In addition, empirical data is subject to measurement noise. For example, Microarray data is often error-prone due to variations in the experimental technology and the stochastic nature of biological processes.

The noise recorded in real applications undermines the ultimate goal of the classical frequent itemset algorithms, i.e., revealing the itemset that is present in a sufficient fraction of transactions (Fig. 1). In fact, when noise is present, the classical frequent itemset algorithms may discover multiple small fragments of the true itemset while missing the true itemset itself. The problem worsens for the most interesting large itemsets since they are more vulnerable to noise.

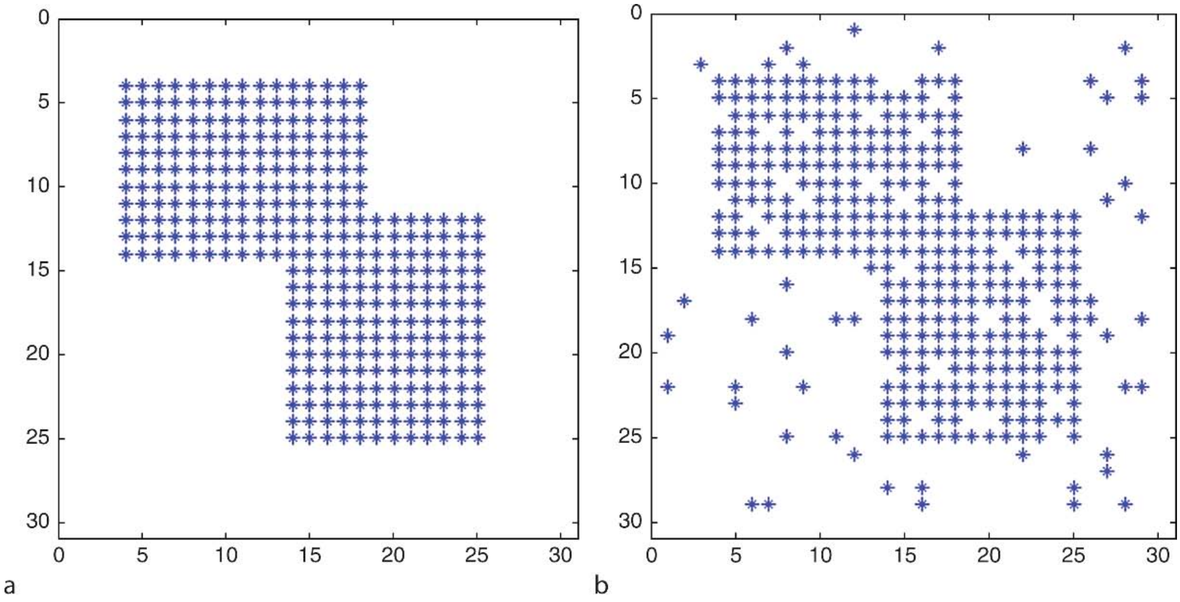
The approximate frequent itemset AFI(ε_r, ε_c) denotes a collection of submatrices of D where the row-wise and column-wise noise levels within each submatrix are below ε_r and ε_c respectively. The classical exact frequent itemset (EFI) is a member of AFI(ε_r, ε_c) where ε_r and ε_c are set to be 0 (Fig. 2). The noise thresholds ε_r and ε_c are usually below 30%. In cases when the noise in either row or column is not restricted, AFI(ε_r, $*$) or AFI($*$, ε_c) will be used to denote the corresponding families. AFI(ε_r, $*$) corresponds to the same family of itemsets, namely *Error Tolerant Itemset (ETI)*, defined by Yang et al. [8].

Foundations

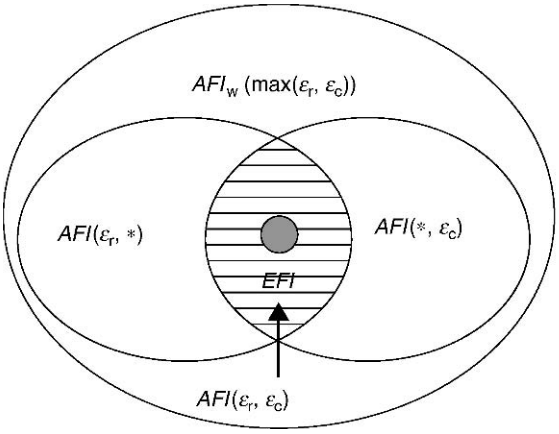
The AFI-mining algorithm [3,4] generalizes the framework of level-wise itemset enumeration. First, the Apriori property of exact frequent itemset mining doesn't hold for AFI when noise is allowed. Instead, conditions under which candidate itemsets can be pruned are established and employed in the AFI algorithm. Secondly, methods that systematically enumerate candidate itemsets without multiple scans of the database are also developed.

Noise-Tolerant Support Pruning

The anti-monotone property of exact frequent itemsets is the key to eliminating the exponential search space in frequent itemset mining [1]. In particular, the anti-monotone property ensures that a (k+1) exact itemset can be pruned if anyone of its k sub-itemsets is not sufficiently supported. However, the allowance of noise may lower the support necessary for the sub-itemsets of a noise-tolerant itemset. The following theorem suggests a lower bound of support for pruning the candidate itemsets in generating AFIs.



Approximation of Frequent Itemsets. Figure 1. Patterns with and without noise.


AFI_w (max(ε_r, ε_c))
$AFI(\varepsilon_r, *)$
$AFI(*, \varepsilon_c)$
EFI
$AFI(\varepsilon_r, \varepsilon_c)$

Approximation of Frequent Itemsets. Figure 2. Relationships of various AFI criteria.

Theorem 1. *Given a minsup threshold, If a length* $(k+1)$*-itemset* I' *is an AFI*(ε_r, ε_c), *for any of its k-sub-itemset* $I \subseteq I'$, *the number of transactions containing no more than* ε_r *fraction of noise must be at least*

$$n \cdot minsup \cdot (1 - \frac{k\epsilon_c}{\lfloor k\epsilon_r \rfloor + 1}) \qquad (1)$$

The noise-tolerant pruning support is defined as the following:

1 Definition

Given ε_c, ε_r and *minsup*, the **noise-tolerant support** for a length-k itemset.

$$minsup_I^k = minsup \cdot (1 - \frac{k\epsilon_c}{\lfloor k\epsilon_r \rfloor + 1})+ \qquad (2)$$

Here $(a)_+ = \max\{a, 0\}$.

The noise-tolerant support threshold is used as the basis of a pruning strategy for AFI mining. The strategy removes supersets of a given I from further consideration when I has support less than $minsup_I{}^k$. In the special case that $\varepsilon_r = \varepsilon_c = 0$, $minsup_I{}^k = minsup$, which is consistent with the anti-monotone property of exact frequent itemsets [1].

0/1 Extensions

A transaction t supports a k-itemset I if t contains at least a fraction $1 - \varepsilon_r$ of the items in I. The transaction set of I consists of all the transactions supporting I. Starting with singleton itemsets, the AFI algorithm generates $(k + 1)$-itemsets from k-itemsets in a breadth-first fashion. For each candidate itemset I, transactions t supporting I are generated. The transaction set of a $(k + 1)$ itemset is constructed from the transaction sets of its k-item subsets in one of two different ways, depending on the value of k and ε_r.

0-extension and 1-extension are the two basic steps to be taken for the efficient collection of the supporting transactions. They obtain the supports based on the support of its sub-itemset while avoiding the repeated database scans plaguing the algorithms proposed by [6,8].

Lemma 1 *(1-Extension) If* $\lfloor k \cdot \varepsilon_r \rfloor = \lfloor (k + 1) \cdot \varepsilon_r \rfloor$ *then any transaction that does not support a k-itemset will not support its* $(k + 1)$ *superset.*

The Lemma is based on the fact that if no additional noise is allowed when generating $(k + 1)$ itemset, a transaction does not support a (k-itemset won't support its $(k + 1)$ superset since the number of 1s it contains is always smaller than or equal or $\frac{\lfloor k_* \epsilon \rfloor - 1 + 1}{k+1} < \epsilon$. Thus if $\lfloor k \cdot \varepsilon_r \rfloor = \lfloor (k + 1) \cdot \varepsilon_r \rfloor$ then the transaction set of a $(k + 1)$ itemset I is the intersection of the transaction sets of its length k subsets. This is called a *1-extension.*

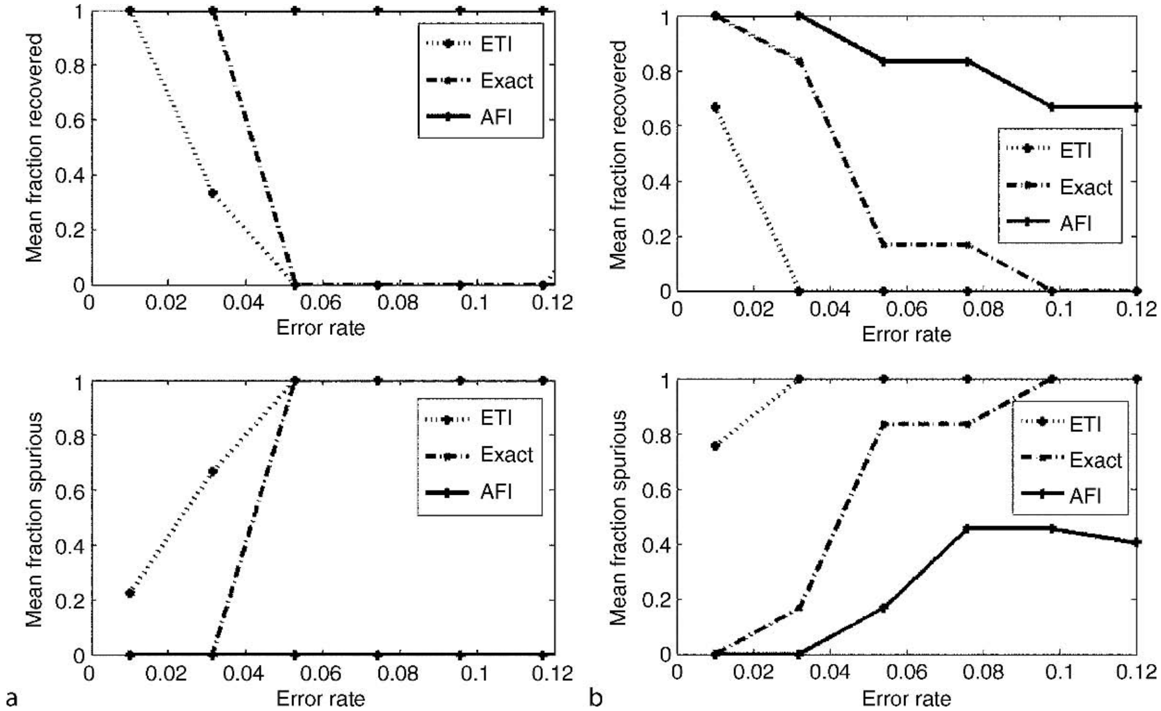
Lemma 2 *(0-Extension) If* $\lfloor k \cdot \varepsilon_r \rfloor + 1 = \lfloor (k + 1) \cdot \varepsilon_r \rfloor$ *then any transaction supporting a (k-itemset also supports its* $(k{+}1)$ *supersets.*

The procedure of 0-extension embodies how noise can be encompassed into a frequent itemset. If additional noise is allowed in a $(k + 1)$-itemset, it is intuitive that a transaction that supports a k-itemset will also support its $k + 1$-item superset, no matter whether the $k + 1$ entry is 1 or 0. To reflect this property, if $\lfloor k \cdot \varepsilon_r \rfloor + 1 = \lfloor (k + 1) \cdot \varepsilon_r \rfloor$, the transaction set of a $(k + 1)$ itemset I is the union of the transaction sets of its length k subsets. This is called a *0-extension.*

Experimental Results

In order to test the quality of the algorithm, data with an embedded pattern and overlaid random errors are generated. The discovered patterns are evaluated against the true patterns which are known in apriori. The methods, exact frequent itemset (EFI), ETI and AFI, are compared in terms of their capabilities in discovering the true patterns in the presence of noise.

Two measures jointly describing the quality are employed. They are "recoverability" and "spuriousness." Recoverability is the fraction of the embedded patterns recovered by an algorithm, while spuriousness is the fraction of the mined results that fail to



Approximation of Frequent Itemsets. Figure 3. Algorithm quality versus noise level.

correspond to any planted cluster. A truly useful data mining algorithm should achieve high recoverability with little spuriousness to dilute the results. A detailed description of the two measures is given in [3].

Multiple data sets were created and analyzed to explore the relationship between increasing noise levels and the quality of the result. Noise was introduced by bit-flipping each entry of the full matrix with a probability equal to p. The probability p was varied over different runs from 0.01 to 0.2. The number of pattern blocks embedded also varied, but the results were consistent across this parameter. The results when one or three blocks were embedded in the data matrix are presented in Fig. 3a,b, respectively.

In both cases, the exact method performed poorly as noise increased. Beyond $p = 0.05$ the original pattern could not be recovered, and all of the discovered patterns were spurious. In contrast, the error-tolerant algorithms, ETI and AFI, were much better at recovering the embedded matrices at the higher error rates. However, the ETI algorithm reported many more spurious results than AFI. Although it may discover the embedded patterns, ETI generates many more patterns that are not of interest, which may overshadow the real patterns of interest. The AFI algorithm consistently demonstrated higher recoverability of the embedded pattern while maintaining a lower level of spuriousness.

Application

AFI mining algorithm can be generally used to find dense 1s blocks in a large binary matrix. The following are example applications.

- E-commerce application: discover approximate frequent itemset in large transactional databases [3,4].
- Biogeographic application: discover regions associated with migration in biogeographic research [3].
- Microarray analysis: find noise tolerant co-expressed patterns.

Cross-references

▶ Association Rule Mining on Streams
▶ Data Mining

Recommended Reading

1. Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 207–216.
2. Creighton C. and Hanash S. Mining gene expression databases for association rules. Bioinformatics, 19(1):79–86, 2003.

3. Liu J., Paulsen S., Wang W., Nobel A., and Prins J. Mining approximate frequent itemset from noisy data. In Proc. 2005 IEEE Int. Conf. on Data Mining, 2005, pp. 721–724.
4. Liu J., Paulsen S., Sun X., Wang W., Nobel A., and Prins J. Mining Approximate frequent itemset in the presence of noise: algorithm and analysis. In Proc. SIAM International Conference on Data Mining, 2006, pp. 405–411.
5. Pei J., Tung A.K., and Han J. Fault-tolerant frequent pattern mining: problems and challenges. In Proc. Workshop on Research Issues in Data Mining and Knowledge Discovery, 2001.
6. Seppanen J.K. and Mannila H. Dense itemsets. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery, and Data Mining, 2004, pp. 683–688.
7. UCI machine learning repository. (http://www.ics.uci.edu/mlearn/MLSummary.html).
8. Yang C., Fayyad U., and Bradley P.S. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2001, pp. 194–203.

Apriori Property and Breadth-First Search Algorithms

BART GOETHALS
University of Antwerp, Antwerp, Belgium

Synonyms

Monotonicity property; Downward closure property; Levelwise search

Definition

Given a large database of sets of items, called transactions, the goal of frequent itemset mining is to find all subsets of items, called itemsets, occurring frequently in the database, i.e., occurring in a given minimum number of transactions.

The search space of all itemsets is exponential in the number of different items occurring in the database. Hence, the naive approach to generate and count the frequency of all itemsets over the database can not be achieved within reasonable time. Also, the given databases could be massive, containing millions of transactions, making frequency counting a tough problem in itself.

Therefore, numerous solutions have been proposed to perform a more directed search through the search space, almost all relying on the well known Apriori-property. These solutions can be divided into breadth-first search and depth-first search, of which the first is discussed here.

Historical Background

The original motivation for searching frequent itemsets came from the need to analyze so called supermarket transaction data, that is, to examine customer behavior in terms of the purchased products. Frequent sets of products describe how often items are purchased together. In 1993, Agrawal, Imilienski, and Swami introduced this problem, and proposed a first algorithm to solve it. Shortly after that, in 1994, the algorithm was improved and named *Apriori*. The main improvement was to exploit the monotonicity property of the frequency of itemsets, later referred to as the *Apriori property*. The same technique was independently proposed by Mannila, Toivonen, and Verkamo, after which both works were combined in one book chapter [1]. Since then, hundreds of improvements and new algorithms have been developed, many of them relying on the breadth-first search strategy as proposed in the Apriori algorithm.

Foundations

A set of items $\mathcal{I}$ and a database $\mathcal{D}$ of subsets of $\mathcal{I}$, called *transactions*, is given. An *itemset* $I \subseteq \mathcal{I}$ is some set of items; its *support* in $\mathcal{D}$ is defined as the number of transactions in $\mathcal{D}$ that contain all items in I. An itemset is called *frequent* in $\mathcal{D}$ if its support in $\mathcal{D}$ is greater than or equal to a given minimum support threshold σ. The goal is now, given a minimal support threshold σ and a database $\mathcal{D}$, to find all frequent itemsets in $\mathcal{D}$.

Instead of naively generating and counting all possible itemsets, several collections of *candidate itemsets* are generated iteratively, and their supports computed until all frequent itemsets have been generated. Obviously, the size of a collection of candidate itemsets must not exceed the size of available main memory. Moreover, it is important to generate as few candidate itemsets as possible, since computing the supports of a collection of itemsets is a time consuming procedure. In the best case, only the frequent itemsets are generated and counted. Unfortunately, this ideal is impossible generally, which will be shown later in this section.

The main underlying property exploited by most algorithms is that support is monotone decreasing with respect to extension of an itemset.

Property 1. (Apriori Property) *Given a transaction database $\mathcal{D}$ over $\mathcal{I}$, let $X, Y \subseteq \mathcal{I}$ be two itemsets. Then,*

$$X \subseteq Y \Rightarrow support(Y) \leq support(X).$$

Hence, if an itemset is infrequent, all of its supersets must be infrequent. In the literature, this property is also called the monotonicity property, or also the downward closure property, since the set of frequent itemsets is closed with respect to set inclusion. This property is of crucial importance for all frequent itemset mining algorithms as it allows for pruning large parts of the search space. As soon as an itemset is known to be infrequent, none of its supersets has to be considered anymore.

The Apriori Algorithm

For simplicity, assume that items in transactions and itemsets are kept sorted in their lexicographic order unless stated otherwise.

The itemset mining phase of the Apriori algorithm is given in Figure 1. The notation *X[i]* is used to represent the *i*th item in *X*; the k-*prefix* of a set *X* is the *k*-set $\{X[1],\ldots,X[k]\}$, and $\mathcal{F}_k$ denotes the frequent *k*-sets.

The algorithm performs a breadth-first (levelwise) search through the search space of all sets by iteratively generating and counting a collection of candidate sets. More specifically, a set is candidate if all of its subsets are counted and frequent. In each iteration, the collection C_{k+1} of candidate sets of size $k + 1$ is generated, starting with $k = 0$. Obviously, the initial set C_1 consists of all items in $\mathcal{I}$ (line 1). At a certain level *k*, all candidate sets of size $k + 1$ are generated. This is done in two steps. First, in the *join* step, the union $X \cup Y$ of sets $X, Y \in \mathcal{F}_k$ is generated if they have the same $k - 1$-prefix (lines 13–15). In the *prune* step, $X \cup Y$ is inserted into C_{k+1} only if all of its *k*-subsets are frequent and thus, occur in $\mathcal{F}_k$ (lines 16–17).

```
Input: $\mathcal{D}, \sigma$
Output: $\mathcal{F}(\mathcal{D}, \sigma)$
 1: $C_1 := \{\{i\} \mid i \in \mathcal{I}\}$
 2: $k := 1$
 3: while $C_k \neq \{\}$ do
 4:   for all transactions $(tid, I) \in \mathcal{D} \leftarrow$ do
 5:     for all candidate sets $X \in C_k$ do
 6:       if $X \subseteq I$ then
 7:         Increment $X.support$ by 1
 8:       end if
 9:   end for
10: end for
11: $\mathcal{F}_k := \{X \in C_k \mid X.support \geq \sigma\}$
12: $C_{k+1} := \{\}$
13: for all $X, Y \in \mathcal{F}_k$, such that $X[i] = Y[i]$
14:    for $1 \leq i \leq k - 1$, and $X[k] < Y[k]$ do
15:    $I := X \cup \{Y[k]\}$
16:    if $\forall J \subset I, |J| = k: J \in \mathcal{F}_k$ then
17:       Add $I$ to $C_{k+1}$
18:   end if
19:  end for
20:  Increment $k$ by 1
21: end while
```

Apriori Property and Breadth-First Search Algorithms. Figure 1. Apriori.

To count the supports of all candidate k-sets, the database is scanned one transaction at a time, and the supports of all candidate sets that are included in that transaction are incremented (lines 4–7). All sets that turn out to be frequent are inserted into $\mathcal{F}_k$ (line 11).

If the number of candidate sets is too large to remain in main memory, the algorithm can be easily modified as follows. The candidate generation procedure stops and the supports of all generated candidates are counted. In the next iteration, instead of generating candidate sets of size $k + 2$, the remaining candidate $k + 1$-sets are generated and counted repeatedly until all frequent sets of size $k + 1$ are generated and counted.

Although this is a very efficient and robust algorithm, its main drawback lies in its inefficient support counting mechanism. Fortunately, a lot of counting optimizations have been proposed for many different situations.

Optimizations

A lot of other algorithms proposed after the introduction of Apriori retain the same general structure, adding several techniques to optimize certain steps within the algorithm. Since the performance of the Apriori algorithm is almost completely dictated by its support counting procedure, most research has focused on that aspect of the Apriori algorithm. Here, only four out of more than hundreds of improvement proposals are outlined, but at least, these four represent the most influential and largest jumps forward.

Item Reordering One of the most important optimizations which can be effectively exploited by almost any frequent set mining algorithm, is the reordering of items.

The underlying intuition is to assume statistical independence of all items. Then, items with high frequency tend to occur in more frequent sets, while low frequent items are more likely to occur in only very few sets.

For example, in the case of Apriori, sorting the items in support ascending order improves the distribution of the candidate sets within the used data structure [4]. Also, the number of candidate sets generated during the join step can be reduced in this way. Unfortunately, until now, no results have been presented on an optimal ordering of all items for any given algorithm and only vague intuitions and heuristics are given, supported by practical experiments.

Partition As the main drawback of Apriori is its slow and iterative support counting mechanism, Savasere et al. [12] proposed the Partition algorithm.

The main novelty in the Partition algorithm, compared to Apriori, is that the database is partitioned into several disjoint parts and the algorithm generates for every part all sets that are relatively frequent within that part. The parts of the database are chosen in such a way that each part fits into main memory, allowing for much more efficient counting mechanisms. Then, the algorithm merges all relatively frequent sets of every part together. This results in a superset of all frequent sets over the complete database, since a set that is frequent in the complete database must be relatively frequent in one of the parts. Finally, the actual supports of all sets are computed during a second scan through the complete database.

Sampling Another technique to solve Apriori's slow counting is to use sampling as proposed by Toivonen [13].

The presented Sampling algorithm picks a random sample from the database that fits in main memory, then finds all relatively frequent patterns in that sample, and finally verifies the results with the rest of the database. In the cases where the sampling method does not produce all frequent sets, the missing sets can be found by generating all remaining potentially frequent sets and verifying their supports during a second pass through the database. The probability of such a failure can be kept small by decreasing the minimal support threshold. However, for a reasonably small probability of failure, the threshold must be drastically decreased, which can cause a combinatorial explosion of the number of candidate patterns. Nevertheless, in practice, finding all frequent patterns within a small sample of the database can be done very fast using fast in-memory support counting techniques. In the next step, all true supports of these patterns must be counted after which the standard levelwise algorithm could finish finding all other frequent patterns by generating and counting all candidate patterns iteratively. It has been shown that this technique usually needs only one more scan resulting in a significant performance improvement [13].

Concise Representations If the number of frequent sets for a given database is large, it could become infeasible to generate them all. Moreover, if the database is dense, or the minimal support threshold is set too low, then there could exist a lot of very large frequent sets, which would make sending them all to the output infeasible to begin with. Indeed, a frequent set of size k includes the existence of at least $2^k - 1$ frequent sets, i.e., all of its subsets. To overcome this problem, several proposals have been made to generate only a concise representation of all frequent sets for a given database such that, if necessary, the frequency of a set, or the support of a set not in that representation can be efficiently determined or estimated [2,5–11].

Key Applications

The Apriori property and the breadth-first search algorithms have broad applications in mining frequent itemsets and association rules. Please refer to the entries of frequent itemset mining and association rules. The Apriori property and the breadth-first search algorithms can be extended to mine sequential patterns. Refer to the entry of sequential patterns. Moreover, they can also be used to tackle other data mining problems such as density-based subspace clustering.

Experimental Results

So far, several hundreds of scientific papers present different techniques and optimizations for frequent set mining and it seems that this trend is keeping its pace. For a fair comparison of some of these algorithms, a contest was organized to find the best implementations in order to understand precisely why and under what conditions one algorithm would outperform another [3]. Although there were no clear winners, one of the rather surprising results of that contest was that the original Apriori algorithm often still performs among the best.

Cross-references

- ▶ Association Rule Mining on Streams
- ▶ Closed Itemset Mining and Non-Redundant Association Rule Mining
- ▶ Frequent Itemsets and Association Rules
- ▶ Frequent Itemset Mining with Constraints

Recommended Reading

1. Agrawal R., Mannila H., Srikant R., Toivonen H., and Verkamo A. Fast discovery of association rules. In Advances in Knowledge Discovery and Data Mining. U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (eds.). MIT, Cambridge, MA, USA, 1996, pp. 307–328.
2. Bayardo J. and Roberto J. Efficiently mining long patterns from databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 85–93.
3. Bodon F. A fast apriori implementation. In Proc. ICDM Workshop on Frequent Itemset Mining Implementations, vol. 90 of CEUR Workshp Proceedings, 2003.
4. Borgelt C. and Kruse R. Induction of association rules: apriori implementation. In Proc. 15th Conference on Computational Statistics, 2002, pp. 395–400.
5. Boulicaut J.F., Bykowski A., and Rigotti C. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. Data Min. Knowl. Discov., 7(1):5–22, 2003.
6. Bykowski A. and Rigitti C. A condensed representation to find frequent patterns. In Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2001, pp. 267–273
7. Calders T. and Goethals B. Mining all non-derivable frequent itemsets. In Principles of Data Mining and Knowledge Discovery, 6th European Conf., 2002, pp. 74–85.
8. Calders T. and Goethals B. Minimal k-free representations of frequent sets. In Principles of Data Mining and Knowledge Discovery, 7th European Conf., 2003, pp. 71–82
9. Gunopulos D., Khardon R., Mannila H., Saluja S., Toivonen H., and Sharma R. Discovering all most specific sentences. ACM Trans. Database Syst., 28(2):140–174, 2003.
10. Mannila H. Inductive databases and condensed representations for data mining. In Proc. 14th Int. Conf. Logic Programming, 1997, pp. 21–30.
11. Pasquier N., Bastide Y., Taouil R., and Lakhal L. Discovering frequent closed itemsets for association rules. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 398–416.
12. Savasere A., Omiecinski E., and Navathe S. An efficient algorithm for mining association rules in large databases. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 432–444.
13. Toivonen H. Sampling large databases for association rules. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 134–145.

ARBAC97

► Administration Model for RBAC

Architecture-aware Database System

► Architecture-Conscious Database System

Architecture-Conscious Database System

John Cieslewicz, Kenneth A. Ross
Columbia University, New York, NY, USA

Synonyms

Architecture-sensitive database system; Architecture-aware database system; Hardware-conscious database system

Definition

Database systems designed with awareness of and a sensitivity to the underlying computer hardware are "architecture-conscious." In an architecture-conscious database system implementation, the performance characteristics of computer hardware guide algorithm and system design.

Historical Background

Database system implementation has been, in varying ways, architecture conscious from the advent of the relational database. For instance, System R [2], an early relational database system prototype included the number of I/Os as a cost metric in its optimizer. At a very high level, the implementers of System R included the characteristics of the underlying hardware in their analysis. This trend has continued with growing attention paid by the database research community to the effects of hardware technology on database performance. Architecture-conscious design took on greater importance as processor speeds improved by four orders of magnitude between 1980 and 2005, while memory latency improved by less than a single order of magnitude. Because of this performance gap, memory accesses, which are central to any database workload, became relatively expensive, requiring database researchers to design database algorithms with this hardware limitation in mind. New computational features have been added to microprocessors, including SIMD instructions, branch prediction, and memory prefetching. These techniques help to improve single threaded performance and instruction level parallelism (ILP), which is the simultaneous processing of instructions from the same thread. In order to take maximal advantage of many of these features, databases must be designed with them in mind.

The introduction of chip multiprocessors (CMP), in particular, has created new challenges and opportunities for improving database performance. Due to problems with power usage, heat dissipation, and diminishing single threaded performance returns from increasingly complex logic to exploit additional ILP, beginning early in the 2000s chip architects switched their design emphasis from faster clock rates to increased on chip parallelism. It is expected that the degree of parallelism supported by CMPs will continue to increase, making it imperative to design database operators for effective on-chip parallelism. In addition to single threaded performance, achieving good performance on chip multiprocessors requires awareness of thread level parallelism (TLP), the simultaneous processing of instructions from multiple instruction streams. All of these features provide additional opportunities for architecture-conscious database system design.

Foundations

The underlying hardware architecture of which an architecture-conscious database system must be aware includes all computer components. For database systems, the most critical components are persistent storage, the memory hierarchy, and the microprocessor. Research in this field is conducted using architecture simulators and with real hardware. Both are valuable tools, as simulators can test new techniques on hypothetical future hardware designs and using real hardware yields results that are applicable to systems in production now. This article focuses on identifying and explaining the architectural features that have performance implications for databases, and providing high-level design guidelines for database systems implementers using these architectures. A more detailed description of specific database implementation techniques is given in [6].

Persistent Storage

As mentioned above, the impact of storage subsystems has been part of database implementation from the beginning, and many database textbooks describe the costs of various relational operators in terms of the number of I/Os required. Accessing secondary storage incurs significant latency compared with accessing data in memory. For magnetic disk based storage systems, sequential access is favored because each read and write requires a large, fixed cost (rotational and seek latency) before the relatively fast reading or writing. Because of this property, databases are often optimized for sequential reads and writes to storage, e.g., reading entire buffer pages rather than individual records. Magnetic disks, however, are not the only type of persistent storage. Flash memory storage devices also provide high density persistent storage, but have different properties than magnetic disks. As of 2007, flash memory does not have density comparable to magnetic disks, but the technology is improving. Flash memory, unlike a magnetic disk, has no mechanical parts, uses less power, and supports a higher number of random I/Os per second. These different characteristics require new thinking about the way a database system uses persistent storage. For instance, the read, write, and erase properties of flash memory make is less suitable for update-in-place database operations than magnetic disks. Changes to page layout and logging have been introduced to overcome this difference [11]. Flash memory's support for more random I/Os per second, also leads to changes in cost-performance metrics such as the five-min rule [7].

Main Memory and Cache Optimizations

Memory density has increased while prices have fallen, resulting in affordable systems in which a database's entire working set can be held in memory. For such systems, I/O latency no longer dominates. At the same time, processor speeds have increased at a much faster rate than memory latency, making a data load from main memory relatively more expensive. Computer architects have combated this problem by adding caches to processors. A cache is a small, but very fast memory on the same chip as the processor. The cache provides a processor with fast access to frequently used data (temporal locality) or data that resides near recently used data (spatial locality). When data is found in the cache, it is called a *cache hit*, but when data is not found in the cache this is a *cache miss* and main memory must be accessed. Accessing main memory is a longer latency operation that can stall the processor's pipeline. On database workloads, cache misses for data and instructions have been shown to account for a significant portion of execution time [1]. Therefore, data structures and query execution techniques that make more efficient use of limited cache resources are complementary improvements that reduce cache misses and improve performance.

In addition to improving cache use, the challenge posed by the memory bottleneck can be overcome by overlapping memory latency with computation. Prefetching facilitates that overlapping by attempting to load data into the cache before it is needed. Prefetching can be controlled both in hardware and software. In hardware prefetching, the processor attempts to identify access patters, such as a sequential scan, and load data ahead of when it is needed. In software prefetching, the programmer inserts prefetch instructions into a program to provide the hardware with hints about what data should be loaded into the cache. Software prefetching must be used when there is no pattern to the memory accesses. Chen et al. have improved the performance of tree-based index searches and hash joins by using prefetch instructions [3,4].

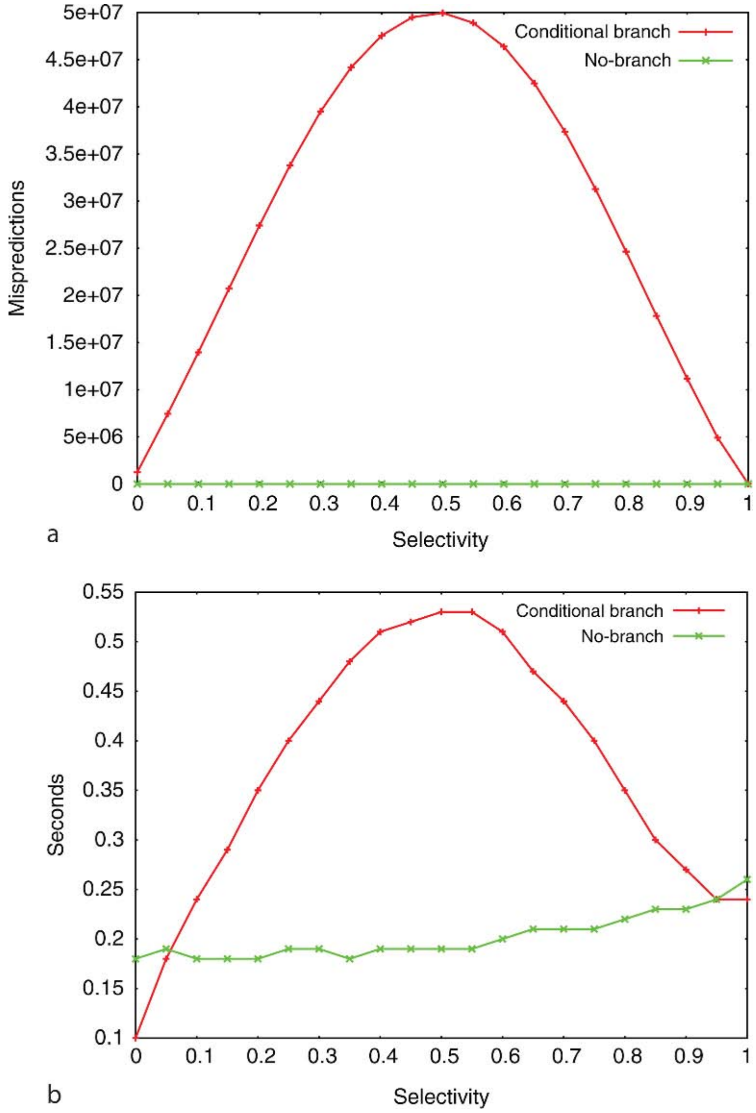
Microarchitecture Optimizations

Conditional branches are a performance hazard on many microarchitectures. A conditional branch represents a control dependency, after which the processor does not immediately know which instruction to execute. On architectures with long pipelines, the outcome of a conditional test is not known for many cycles. In the interim, the processor must either stop issuing instructions or guess the outcome of the condition. This control dependency can limit instruction level parallelism (ILP) because the compiler is limited in its ability to reorder instructions around a branch and the processor is unsure which instructions to issue after a branch instruction. Many modern processors attempt to guess the outcome of the branch using hardware branch prediction, so that instructions continue to be issued in spite of the branch. Branch prediction attempts to identify patterns in the recent history of branch outcomes in order to guess the present branch's outcome correctly. If the branch is predicted correctly, the branch causes no delay in the pipeline. When the prediction is incorrect, the pipeline must be flushed and restarted because invalid instructions have been issued. This cost is the "branch misprediction penalty" and is related to the depth of the pipeline. When there is no pattern to the branching, branch prediction fails up to half the time, and the misprediction penalty degrades performance. In database workloads, where branches are often data dependent, branch misprediction has a noticeable impact on performance [1]. Figure 1a shows the number of mispredictions of performing a simple range selection on an array of 100 million uniformly distributed four byte integers on a 2.4 GHz Intel Core 2 Duo processor. The graphs in Fig. 1 show data for a selection implementation using a conditional branch and a "no-branch" implementation in which the conditional branch's control dependency has been changed to a data dependency [12]. The maximum number of mispredictions occur in the conditional branch implementation when the selectivity is 0.5 because the branch prediction is wrong half of the time. This high rate of branch misprediction impacts performance, as shown in Fig. 1b. In contrast, the implementation with no conditional branch is less sensitive to the selectivity, with its modest rise in execution time (Fig. 1b) attributed to the cost of writing more output as the selectivity increases. Ross extends the query optimizer's cost model to take the cost of branch misprediction into account and demonstrates techniques for evaluating selection conditions using varying numbers of conditional branches [12].

Modern microarchitectures feature special vector operations, also known as *single instruction multiple data* (SIMD) instructions. These instructions operate on wide registers containing multiple variables, enabling data parallelism. For example, a 128 bit SIMD register can contain 16 `char` data types or four 32-bit integers. A wide range of mathematical, logical, and data movement operations can be applied in parallel to those variables using a single instruction. SIMD instructions can be used to improve the data parallelism present in database execution, resulting in higher performance [14]. For some architectures, such as the Cell Broadband Engine, which are highly optimized for SIMD processing, SIMD-izing database operations is crucial to achieving good performance [9]. SIMD instructions can be used to transform control dependencies into data dependencies. For instance, when using scalar instructions, selection is commonly performed with a comparison operation followed by a conditional branch. In contrast, with SIMD instructions, a selection is performed by applying a SIMD comparison operation to multiple elements in parallel. The result of this operation is a bit-mask that can be used to select the elements that pass the condition using other instructions. Transforming a control dependency into a data dependency can also be accomplished with predicated scalar instructions such as a *conditional move* instruction. Regardless of how the control dependency is eliminated, doing so will improve ILP, particularly if the branch is difficult to predict, as is the case in the example shown in Fig. 1.

On-Chip Parallelism

A chip multiprocessor (CMP) provides multiple on chip hardware thread contexts, often in the form of multiple



Architecture-Conscious Database System. Figure 1. Number of branch mispredictions and execution time for a one-sided range selection with varying selectivity. The red, ''Conditional Branch,'' line is an implementation using a conditional branch, while the green, ''No-Branch,'' line is an implementation where the conditional branch's control dependency has been replaced with a data dependency [8].

processor cores on a single die. A processor core's pipeline may be shared by different instruction streams, a design known as simultaneous multithreading (SMT). Some CMPs have both multiple cores and multiple threads per core. As of 2007, the most cores available on a commodity processor was eight, with each core supporting four threads for a total of 32 threads per CMP.

Hardware support for thread level parallelism (TLP) in a CMP differs from a symmetric multiprocessor (SMP) in that a CMP is one processor die with multiple processor cores and an SMP is a system composed of multiple processors sharing one main memory. This difference has two important implications. First, when data is shared among two or more processors in an SMP system, modifications to that data by any processor is coordinated by a cache coherency protocol, which uses the system bus to communicate information about changes to shared data. The system bus is significantly slower than the processors and if a large amount of data is shared among the processors or it is frequently updated, cache coherency overhead can significantly impact performance. In contrast, because all of the cores of a CMP are on the same die, communication between the cores can occur at chip speeds, greatly reducing the coherency overhead of shared data. Second, in a SMP system, although the main memory is shared, each processors' cache hierarchy is separate. In contrast, many CMPs share some cache levels among the processor cores or threads. Because some cache resources may be shared, designing threads to share data in the cache can be beneficial provided that coordinating the sharing does no introduce large amounts of overhead. These differences have implications for architecture-conscious database systems such as the challenge of designing core database operations that take advantage of the on-chip parallelism afforded by chip multiprocessors in a manner that will scale as CMPs provide more hardware thread contexts.

Chip multiprocessors present opportunities and challenges for improving database performance [8]. Finding parallelism in database systems is not difficult. Most database systems have long supported concurrent queries from different users. Other forms of parallelism can be found within a single query or within a single operator. Parallelism alone does not guarantee optimal performance on a CMP. An architecture-conscious database system on a CMP must exploit the on-chip parallelism in a manner that uses the parallel resources to maximum effect. For instance, Cieslewicz and Ross find that balancing parallelism, cache sharing, and thread communication is key to achieving good performance when computing aggregates using a CMP [5]. Chip multiprocessors can also help alleviate some of the problems associated with the memory bottleneck described previously. By having multiple concurrently executing thread contexts on one chip, a stall due to a cache miss in one thread does not stall the entire processor, leading to higher overall processor utilization. Similar to techniques using prefetch instructions described earlier, an on-chip thread can also be used to explicitly load data into the cache for other another thread [13].

Techniques that perform well on a uniprocessor may not be appropriate for a CMP. For instance, Johnson et al. found that work-sharing schemes for in-memory workloads on CMPs actually reduced performance [10]. This was because work-sharing created a bottleneck at the shared work, limiting the total parallelism. An important lesson is that in order to achieve optimal database performance on a CMP, database implementers must carefully evaluate design decisions that may limit parallelism.

Key Applications

Databases are crucial to a wide range of data storage and analysis activities. Optimizing core database operations to take advantage of all of the features and resources available on modern hardware will result in better performance, which in turn has the ability to directly improve the experience of all users of databases.

Future Directions

Adapting database operators to increased on-chip parallelism afforded by CMPs is one open problem. As the amount of on-chip parallelism increases, new bottlenecks and scaling challenges will emerge. Another area for future research includes using non-traditional architectures, such as the Cell Broadband Engine [9] and other future heterogeneous processors, for database operations. As long as computer architecture continues to evolve, architecture-conscious databases will need to adapt to new technologies.

Cross-references

► Cache-Conscious Query Processing

Recommended Reading

1. Ailamaki A., DeWitt D.J., Hill M.D., and Wood D.A. DBMSs on a modern processor: Where does time go? In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 266–277.
2. Chamberlin D.D., Astrahan M.M., Blasgen M.W., Gray J.N., King W.F., Lindsay B.G., Lorie R., Mehl J.W., Price T.G., Putzolu F., Selinger P.G., Schkolnick M., Slutz D.R., Traiger I.L., Wade B.W., and Yost R.A. A history and evaluation of System R. Commun. ACM, 24(10):632–646, 1981.
3. Chen S., Ailamaki A., Gibbons P.B., and Mowry T.C. Improving hash join performance through prefetching. ACM Trans. Database Syst., 32(3):17, 2007.
4. Chen S., Gibbons P.B., Mowry T.C., and Valentin G. Fractal prefetching B+ trees: Optimizing both cache and disk performance. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 157–168.
5. Cieslewicz J. and Ross K.A. Adaptive aggregation on chip multiprocessors. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 339–350.
6. Cieslewicz J. and Ross K.A. Database optimizations for modern hardware. Proc. IEEE, 96(5):863–878, May 2008.
7. Graefe G. The five-minute rule twenty years later, and how flash memory changes the rules. In Proc. Workshop on Data Management on New Hardware, 2007.
8. Hardavellas N., Pandis I., Johnson R., Mancheril N., Ailamaki A., and Falsafi B. Database servers on chip multiprocessors: Limitations and opportunities. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 79–87.

9. Héman S., Nes N., Zukowski M., and Boncz P. Vectorized data processing on the Cell Broadband Engine. In Proc. Workshop on Data Management on New Hardware, 2007.
10. Johnson R., Hardavellas N., Pandis I., Mancheril N., Harizopoulos S., Sabirli K., Ailamaki A., and Falsafi B. To share or not to share? In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 351–362.
11. Lee S.-W. and Moon B. Design of flash-based DBMS: an in-page logging approach. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 55–66.
12. Ross K.A. Selection conditions in main memory. ACM Trans. Database Syst., 29:132–161, 2004.
13. Zhou J., Cieslewicz J., Ross K.A., and Shah M. Improving database performance on simultaneous multithreading processors. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 49–60.
14. Zhou J. and Ross K.A. Implementing database operations using SIMD instructions. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 145–156.

Architecture-Sensitive Database System

► Architecture-Conscious Database System

Archiving Experimental Data

REAGAN W. MOORE
University of California, San Diego, La Jolla, CA, USA

Synonyms

Preservation; Reference collections

Definition

The archiving of data is the process by which a project preserves both data and representation information that is needed to interpret the data.

Historical Background

An increasing fraction of science research is based on the analysis of large data collections. Experimental data are collected from measurements by sensors on repeatable experiments conducted in laboratories. Observational data are collected from sensors that measure properties of the natural environment. Simulation output files are generated by applications that run on supercomputers. The size of each of these types of research data can be massive, measured in Petabytes (thousands of Terabytes), such that there is physically not enough room to keep the data on high-speed disk file systems.

The archiving of experimental data traditionally focused on the migration of data from high-performance (more costly) storage systems to the lowest cost media that were available (originally tape-based systems). Files were manually copied from disk onto the tape archive. The IEEE Mass Storage System Reference Model version 5 defines the properties that a tape archive should provide [4]. The properties include the ability to automate retrieval of the file, manage the name space used to identify the file, and manage migration of files between tapes. While these capabilities enable the long-term management of the files, each project was required to manage independently any representation information needed to interpret the file and the meaning of the data. The OAIS model defines representation information that should be provided for preservation of data [6]. This includes information about the source of the data, descriptions of the structures present within the data file, identification of the application that can manipulate the data structure, descriptions of the meaning of the data, and identification of a knowledge community that can interpret the meaning. Archiving of experimental data is successful when the data can be retrieved, interpreted, and manipulated by the project at an unspecified future time.

Foundations

The archiving of experimental data is facilitated through the development of standards for the descriptive terms used to describe meaning and provenance, standards for describing the structures present in the file, and standards for the services that manipulate the data structures.

- Standard semantics. The terms used to describe physical phenomena are created by each discipline. The meaning of these terms evolves over time as a better understanding is developed of the physical phenomena. A preservation environment will need to support evolution of the semantic meaning, through the mapping of prior vocabulary sets to the new vocabulary. This is typically done through use of ontologies that define logical mappings between terms, spatial correlation of terms to maps, assignment of temporal coordinates to processes, and specification of functional relationships between physical quantities.

- Standard formats. Each file is a linear string of bits, on which structure is imposed. A file format describes how to interpret the bit string into structures that can be named. Since the types of observational or experimental or simulation data vary extensively, each community chooses a standard format for specific physical phenomena. As new phenomena are measured, new data formats are created. Thus a preservation environment needs to manage either characterization of the file formats or migration of the file formats onto new standards. A promising approach to handle data format migration is based on Data Format Description Languages, that enable the description of the structure of the file using an XML syntax.
- Standard access methods. Each scientific community defines standard processing steps for manipulating the structures present within the standard data formats, using the standard semantic terms. These standard processing mechanisms can be ported on top of data grid technology to enable their use within the preservation environment [5]. An alternate approach is to port the required display application to each new operating system technology. This assumes the data can be retrieved from the preservation environment, and a copy is placed on the computer where the emulated application is executed.

The size of science data collections poses a major scalability challenge, and strongly impacts the implementation of the data archiving process. The large number of files and the large size of an individual file require support mechanisms that may not be provided by a specific storage system. The science research projects listed below use data grid technology to overcome limitations in current storage technology, and to simplify incorporation of new technology within the preservation environment. Data grids are software middleware that insulate the scientific collections from dependencies on current storage technology through the implementation of infrastructure independence [2]. The names used to describe the science data are managed by the data grid independently of the choice of storage system. This makes it possible to ensure consistent naming even when data is distributed across multiple storage systems or migrated from old technology to new technology.

Data grids provide standard operations for manipulating science data. The operations are executed at the remote storage system through interoperability mechanisms that can be applied on any choice of storage. Data grids also provide standard interfaces that enable the use of a preferred access mechanism. The result is the ability to use a particular research group's access interface to manipulate data stored on multiple types of remote storage systems.

The mechanisms provided by data grids to manage science data explicitly handle issues of scale:

- Large file size. An individual file can be multiple gigabytes in size. The time required to move such a file over a network to a storage system can be excessive. By using parallel I/O streams, the time can be decreased by multiple factors of two. Effectively, the data file is separated into multiple segments, each of which is transmitted over the network in parallel with the other segments. Large files are typically sent using four to eight parallel I/O streams.
- Large file size. The retrieval of a large file, even with parallel I/O streams may take multiple minutes. For cases where a small subset of the large file is needed, it is more efficient to filter the file at the remote storage system, and send only the desired subset back over the network. Data grids support remote manipulation of files through the execution of remote procedures at the location where the file resides.
- Large number of files. Each storage system has a maximum number of files that it is designed to handle. Large collections can exceed the number of files that can be written to a single storage system. To overcome this limitation, data grids aggregate small files into a single larger file called a container. The container is written to the storage system. The data grid maintains information that allows it to track the location in the container of each file. For example, the 2-Micron All Sky Survey [8] astronomy image collection containing 5 million images was archived on tape by aggregating the images into 147,000 containers. If the images in each container are from the same area on the sky, then retrieval of the container can result in multiple related images becoming accessible, improving bulk access to the collection.
- Large number of files. For the archived files to be useful, descriptive information is needed about

each file to support discovery. The descriptive information may be extractable from the file. However, if a discovery request is issued that requires the parsing of every file in the collection, the time to satisfy the request may be exceptionally long. Data grids manage metadata for each file in a metadata catalog. The metadata catalog is stored as tables in a relational database, enabling efficient searches. Data grids can use remote procedures to parse the descriptive metadata from the file, and bulk load the metadata into the metadata catalog.

- Large size of collections. The management of integrity across large collections may be viewed as an intractable problem. File integrity can be verified by reading each file, calculating a checksum, and then comparing the checksum with a previous value that was stored in the metadata catalog. If the file has become corrupted, the two checksums will not be equal. If a single tape drive is used to read a Petabyte collection, a sustained data transfer rate of 33 MB/s is needed to read the entire collection in a year. Thus integrity checking of large collections can require the dedication of significant hardware resources. If a problem is detected, a second copy is required to be able to repair the corruption. Data grids support the replication of data onto multiple storage systems that may be located at geographically remote locations. The geographic separation is needed to ensure recovery from natural disasters. Synchronization of replicas is done to verify integrity of the files.

Key Applications

Science disciplines are generating massive collections of experimental, observational, and simulation data. Single projects such as the Southern California Earthquake Center [7] plan to generate over 1.5 Petabytes of simulation data of seismic wave propagation from earthquakes on the San Andreas Fault. The Large Synoptic Survey Telescope (LSST) plans to capture more than 130 Petabytes of observational data [3]. The LSST project takes photographs of the sky to track near earth objects, supernovae, and micro-lensing events that can provide information on the structure of the Universe. The BaBar high-energy physics experiment has moved more than 400 Terabytes of experimental data from the Stanford Linear Accelerator in Palo Alto, California to Lyon, France for analysis by collaborating physicists [1]. In each case, the data are archived for comparison with future research results.

Future Directions

The ability to validate the trustworthiness of a digital repository is becoming an essential requirement for the archiving of experimental data. When scientific collections are archived, each community defines assessment criteria that they expect the preservation environment to maintain. The assessment criteria may be related to retention and disposition policies, or to time-dependent access controls, or to specifications of required descriptive metadata, or to required access mechanisms for data display and manipulation. An emerging requirement for scientific collections is the characterization of the management policies under which the desired collection properties are enforced.

Rule based data grids provide the mechanisms needed not only to enforce the application of the collection management policies, but also to automate the execution of data management policies. As scientific collections grow to the Petabyte size, the labor required to administer the collections can become onerous. This is driven by the use of distributed storage systems to manage the collections. Once the collection resides on multiple types of storage systems, located on multiple administrative domains at geographically remote sites, it becomes very hard to control what is happening. A network router or storage system may be taken down for maintenance, or an operational procedure may change, causing an unexpected result. Data grids provide mechanisms to recover from such problems through the use of replicas, checksums, and synchronization. A problem can be detected and repaired through an administrator-initiated action. When the number of detected problems becomes too large, the administrator is no longer able to keep up with the workload.

Rule based data grids minimize the labor required by the administrator by automating execution of management policies. Management policies are defined that control the preservation processes that are applied to the collection (e.g., validate checksum, verify presence of required metadata, implement the retention policy). Assessment criteria are specified that evaluate whether the management policies have been correctly applied. In a rule-based data grid, the preservation processes are expressed as sets of micro-services that are executed at the remote storage system. Each micro-service in turn is composed from standard operations that the data grid implements for each type of storage system. Management policies are expressed as sets of rules that control the execution of the micro-services. A rule engine is installed at each remote storage system to ensure that the policies are enforced, no matter which access mechanism is used to interact with the data grid. The assessment criteria are mapped to queries on persistent state information that is generated after the execution of each micro-service. Such a rule based data grid is capable of monitoring its own operations and verifying the trustworthiness of the digital repository that holds the archived scientific collection [2].

Cross-references

▶ Data Warehouse
▶ Disaster Recovery
▶ Information Lifecycle Management
▶ Meta data Repository
▶ Provenance
▶ Replication

Recommended Reading

1. BaBar B meson high energy physics project. Available at: http://www.slac.stanford.edu/BFROOT/
2. Integrated Rule-Oriented Data System (iRODS). Available at: http://irods.sdsc.edu/
3. Large Synoptic Survey Telescope (LSST). Available at: http://www.lsst.org/
4. Miller S.W. Mass storage reference model special topics. In Proc. 9th IEEE Symp. on Mass Storage Systems, Monterey, CA, November 1988, pp. 3–7.
5. Moore R. Building preservation environments with data grid technology. Am. Arch., 69(1):139–158, July 2006.
6. OAIS, reference model for an open archival information system, ISO standard ISO 14721:2003. Available at: http://nost.gsfc.nasa.gov/isoas/ref_model.html
7. Southern California Earthquake Center (SCEC). Available at: http://www.scec.org/
8. 2-micron all sky survey. Available at: http://www.ipac.caltech.edu/2mass/

Armstrong Axioms

Solmaz Kolahi
University of British Columbia, Vancouver,
BC, Canada

Definition

The term *Armstrong axioms* refers to the sound and complete set of inference rules or axioms, introduced by William W. Armstrong [2], that is used to test logical implication of functional dependencies.

Given a relation schema $R[U]$ and a set of functional dependencies Σ over attributes in U, a functional dependency f is logically implied by Σ, denoted by $\Sigma \models f$, if for every instance I of R satisfying all functional dependencies in Σ, I satisfies f. The set of all functional dependencies implied by Σ is called the *closure* of Σ, denoted by Σ^+.

Key Points

Armstrong axioms consist of the following three rules:

Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$.
Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$.
Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Note that in the above rules XZ refers to the union of two attribute sets X and Z. Armstrong axioms are *sound* and *complete*: a functional dependency f is derivable from a set of functional dependencies Σ by applying the axioms if and only if $\Sigma \models f$ (refer to [1] for more information).

Cross-references

▶ Functional Dependency
▶ Implication of Constraints

Recommended Reading

1. Abiteboul S., Hull R. and Vianu V. Foundations of Databases. Addison-Wesley, Reading, MA, 1995.
2. Armstrong W. Dependency structures of data base relationships. In IFIP Congress. 1974.

Array

▶ Redundant Array of Independent Disks (RAID)

Array Databases

▶ Raster Data Management and Multi-Dimensional Arrays

Association

▶ Abstraction
▶ Similarity and Ranking Operations

Association Rule Mining on Streams

Philip S. Yu[1], Yun Chi[2]
[1]IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
[2]NEC Laboratories America, Cupertino, CA, USA

Definition

Let $I = \{ i_1,\ldots,i_m\}$ be a set of items. Let S be a stream of transactions in a sequential order, where each transaction is a subset of I. For an *itemsetX*, which is a subset of I, a transaction T in S is said to *contain* the itemset X if $X \subseteq T$. The *support* of X is defined as the fraction of transactions in S that contain X. For a given support threshold s, X is *frequent* if the support of X is greater than or equal to $s\%$, i.e., if at least $s\%$ transactions in S contain X. For a given *confidence* threshold c, an *association ruleX* $\Rightarrow Y$ holds if $X \cup Y$ is frequent and at least $c\%$ of transactions in S that contain X also contain Y. The problem of association rule mining on streams is to discover all association rules that hold in a stream of transactions.

Historical Background

In 1993, Rakesh Agrawal et al. [1] proposed the framework for association rule mining. Since this seminal work, a lot of researches have been done to improve the efficiency of association rule mining algorithms, to extend the definition of associations rule, and to apply association rule mining to other types of data such as time sequence data and structured data such as graphs. On the other hand, research on data streams started around 2000 when several data stream management systems were originated (e.g., the Brandeis AURORA project, the Cornell COUGAR project, and the Stanford STREAM project) to solve new challenges in applications such as network traffic monitoring, transaction data management, Web click streams monitoring, sensor networks, etc. Because association rule mining plays an important role in these data stream applications (e.g., the motivation in the first association rule paper [1] is to mine patterns from retail store transaction data), along with the development of data stream management systems, developing association rule mining algorithms for data streams has become an important research topic.

Foundations

Two Sub-problems

Algorithms for association rule mining usually consist of two steps. The first step is to discover frequent itemsets. In this step, all frequent itemsets that meet the support threshold are discovered. The second step is to derive association rules. In this step, based on the frequent itemsets discovered in the first step, the association rules that meet the confidence criterion are derived. Because the second step, deriving association rules, can be solved efficiently in a straightforward manner, most of researches focus mainly on the first step, i.e., how to efficiently discover all frequent itemsets in data streams. Therefore, in the rest of this article, the focus will be on frequent itemset mining in data streams.

Key Challenges

Frequent itemset mining in general is already a challenging problem. For example, due to combinatorial explosion, there may be huge number of frequent itemsets, and a main challenge is how to efficiently enumerate, discover, and store frequent itemsets. Data streams, because of their unique features, have further posed many new challenges to frequent itemset mining. Some of these new challenges are described as the following.

Single access of data In data streams, data are arriving continuously with high speed and in large volume. As a consequence, in many cases it is impractical to store all data in persistent media and in other cases, it is too expensive to (randomly) access data multiple times. The challenge is to discover frequent itemsets while the data can only be assessed once.

Unbounded data Another feature of data streams is that data are unbounded. In comparison, storage that can be used to discover or maintain the frequent itemsets is limited. Another consequence of unbounded data is that whether an itemset is frequent depends on time. The challenge is to use limited storage to discover dynamic frequent itemsets from unbounded data.

Real-time response Because data stream applications are usually time-critical, there are requirements on response time. For some restricted scenarios, algorithms that are slower than the data arriving rate are useless. The challenge is therefore to efficiently mine frequent itemsets in real time.

Data Models

Compared with finite data in traditional databases, data streams are unbounded and the number of transactions increase with time. Due to these characteristics, different data models have been proposed in different mining algorithms.

The first data model is an accumulative model. In this model, all data in the range from the beginning to the current time are considered in an equal fashion. Therefore, frequent itemsets are defined on the accumulated data where new data are appended continuously as time grows.

The second data model is based on a sliding window. That is, although the whole data stream is unbounded, the frequent itemsets are defined based on the most recent data that fall within a temporal sliding window whose ending time is the current time. One justification for such a sliding-window model is that due to concept drifts, the data distribution in streams is usually changing with time, and very often people are interested in the most recent patterns.

The third data model falls in between the first two models – while all data are considered in frequent itemset mining, they are weighted differently according to a predefined weighting function. A very commonly used weighting function is the exponentially decaying function, that is, for a transaction at time τ, its weight is $\alpha(\tau) = \exp(\tau - t)$, where t is the current time.

Algorithm Types

Based on the mining results, existing frequent itemsets mining algorithms on data streams can be roughly divided into two categories: the exact mining algorithms and the approximate mining algorithms.

Exact mining algorithms provide as results all the frequent itemsets in the data streams together with their accurate supports. Usually the focus of exact algorithms is to efficiently update frequent itemsets when new transactions arrive and (in the sliding-window data model) when old transactions expire.

Approximate mining algorithms, on the other hand, focus more on finite memory usage and single access of data, at the cost of the accuracy of the mining results. Approximate algorithms target low false positive rate, zero, or low false negative rate, and tight error bounds on the estimation for the supports of the frequent itemsets.

Representative Algorithms

Cheung et al. [6,7] proposed algorithms *FUP* and *FUP*$_2$ for incrementally updating frequent itemsets. Thomas et al. [13] presented a similar algorithm. Both Cheung's and Thomas's algorithms assume batch updates and take advantage of the relationship between the original database (*DB*) and the incrementally changed transactions (*db*). *FUP* is similar to the well-known Apriori algorithm [1], which is a multiple-step algorithm. The key observation of *FUP* is that by adding *db* to *DB*, some previously frequent itemsets will remain frequent and some previously infrequent itemsets will become frequent (these itemsets are called *winners*); at the same time, some previously frequent itemsets will become infrequent (these itemsets are called *losers*). The key technique of *FUP* is to use information in *db* to filter out some winners and losers, and therefore reduce the size of candidate set in the Apriori algorithm. Because the performance of the Apriori algorithm relies heavily on the size of candidate set, *FUP* improves the performance of Apriori greatly. *FUP*$_2$ extended *FUP* by allowing deleting old transactions from a database as well. Therefore *FUP* is restricted to the accumulative data model while *FUP*$_2$ can be used on the sliding-window data model as well. The algorithm proposed by Thomas et al. is similar to *FUP*$_2$ except that in addition to frequent itemsets, a negative border is maintained. In the algorithm, the frequent itemsets in *db* are mined first. At the same time, the counts of frequent itemsets (and itemsets on the negative border) in *DB* are updated. Then based on the change of the frequent itemsets in *DB*, the negative border in *DB*, and the frequent itemsets in *db*, the frequent itemsets in the updated database are computed with a possible scan of the updated database. Because the updated database is scanned at most once, Thomas's algorithm has very good performance. Thomas's algorithm can be used for both the accumulative and the sliding-window data models. In addition, *FUP*, *FUP*$_2$, and Thomas's algorithm all fall into the category of exact mining algorithms.

Veloso et al. [14] proposed an algorithm *ZIGZAG* for mining frequent itemsets in evolving databases. Later, Otey et al. [11] extended *ZIGZAG* into parallel and distributed algorithms. *ZIGZAG* is similar to Cheung's and Thomas's algorithms in that it achieves its speedup by using the relationship between *DB* and *db*. However, *ZIGZAG* has many distinct features. First, *ZIGZAG* mainly used *db* to speedup the support counting of frequent itemsets in the updated database and it does not discover the frequent itemsets in *db* itself. As a result, for a given minimum support, *ZIGZAG* can handle batch update with arbitrary block size. Second, *ZIGZAG* adapts the techniques proposed in the *GENMAX* algorithm [9] and in each update only maintains *maximal* frequent itemsets. Because the information on maximal frequent itemsets and their supports is not enough to generate association rules (because the support information of some non-maximal frequent itemsets may be missing), a second step is used in *ZIGZAG* in which the updated database is scanned to discover all frequent itemsets and their supports.

Chi et al. [5] developed an algorithm, *Moment*, to mine *closed* frequent itemsets over data stream sliding windows. In this work the authors introduced a compact data structure, the *closed enumeration tree* (CET), to maintain a dynamically selected set of itemsets over a sliding window. The selected itemsets contain a boundary between closed frequent itemsets and the rest of the itemsets. Concept drifts in a data stream are reflected by boundary movements in the CET, and can be efficiently captured. Both *ZIGZAG* and *Moment* are exact mining algorithms that use the sliding-window data model.

Charikar et al. [3] presented a one-pass algorithm, *Count Sketch*, that returns most frequent *items* whose frequencies satisfy a threshold with high probabilities. Manku et al. [10] developed a randomized algorithm, the *Sticky Sampling* algorithm, and a deterministic algorithm, the *Lossy Counting* algorithm, for maintaining frequent *items* over a data stream where for a given time *t*, the frequent items are defined over the *entire* data stream up to *t*. The algorithms guarantee no false negative and a bound on the error of estimated frequency (the guarantees are in a probabilistic sense for the randomized algorithm). The *Lossy Counting* algorithm is extended to handle frequent *itemsets*, where a trie is used to maintain all frequent itemsets and the trie is updated by batches of transactions in the data stream. The algorithms of Manku et al. strive for a tunable compromise between memory usage and error bounds. *Count Sketch*, *Sticky Sampling*, and *Lossy Counting* are all approximate mining algorithms that use the accumulative data model.

Teng et al. [12] presented an algorithm, *FTP-DS*, that mines frequent temporal patterns from data streams of itemsets. *FTP-DS* is an approximate mining algorithm that uses the sliding-window data model. Chang et al. [2] presented an algorithm, *estDec*, that mines recent frequent itemsets where the frequency is defined by an aging function. Giannella et al. [8] proposed an approximate algorithm for mining frequent itemsets in data streams during arbitrary time intervals. An in-memory data structure, *FP-stream*, is used to store and update historic information about frequent itemsets and their frequency over time and an aging function is used to update the entries so that more recent entries are weighted more. Both *estDec* and Giannella's algorithm are approximate mining algorithms on weighted transactions. However, Giannella's algorithm is a little different in that it can provide different error levels for data at multiple time granularities.

The above representative algorithms are summarized in Table 1. For a more detailed survey on algorithms for frequent itemset mining over data streams, refer to a recent survey by Cheng et al. [4].

Key Applications

For most data stream applications, there are needs for mining frequent patterns and association rules from data streams. Some key applications in various areas are listed in the following.

Performance monitoring Monitor network traffic and performance, detect abnormality and intrusion

Association Rule Mining on Streams. Table 1. The categorization of the representative algorithms according to their data models and algorithm types

	Data Model		
	Accumulative	Sliding-window	Weighted
Exact Mining Algorithm	*FUP* [2]	*FUP*$_2$ [3] Thomas's [4] *ZIGZAG* [5, 6] *Moment* [8	
Algorithm Mining Algorithm	*Count Sketch* [9] *Sticky Sampline* [10] *Lossy Counting* [10]	*FTP-DS* [11	*estDec* [12] Giannella's [13]

Transaction monitoring Monitor transactions in retail stores, ATM machines, and financial markets

Log record mining Mine patterns from telecommunication calling records, Web server log, etc.

Sensor network mining Mine patterns in streams coming from sensor networks or surveillance cameras

Experimental Results

In general, for each of the presented algorithms, there are supporting experimental studies in the corresponding references. The commonly compared performance metrics include memory usage, speed of the mining process, and (for the approximate algorithms) errors such as false positive rate, false negative rate, and support errors for the frequent itemsets.

Data Sets

A Linux version of the synthetic data generator originally developed by Agrawal et al. [1] is available at http://miles.cnuce.cnr.it/~palmeri/datam/DCI/datasets.php

Some other synthetic and real-life data sets are available from the Frequent Itemset Mining Dataset Repository at http://fimi.cs.helsinki.fi/data/

Furthermore, pointers to some related data sets are available at the KDnuggets Web site http://www.kdnuggets.com/datasets/

Cross-references

▶ Approximation of Frequent Itemsets
▶ Association Rule Mining
▶ Change Detection on Streams
▶ Closed Itemset Mining and Nonredundant Association Rule Mining
▶ Continuous Queries in Sensor Networks
▶ Data Aggregation in Sensor Networks
▶ Data Estimation in Sensor Networks
▶ Data Mining
▶ Data Sketch/Synopsis
▶ Data Streams
▶ Frequent Items on Streams
▶ Frequent Itemsets and Association Rules
▶ Incremental Computation of Queries
▶ Internet and Web Transactions
▶ One-Pass Algorithm
▶ Pattern-Growth Methods
▶ Randomization Methods
▶ Real-Time Transactions
▶ Sensor Network
▶ Stream Mining
▶ Stream Models
▶ Streaming Applications
▶ Tries
▶ Web Services

Recommended Reading

1. Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 207–216.
2. Chang J.H. and Lee W.S. Finding recent frequent itemsets adaptively over online data streams. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 487–492.
3. Charikar M., Chen K., and Farach-Colton M. Finding frequent items in data streams. In Proc. 29th Int. Colloquium on Automata, Languages and Programming, 2002, pp. 693–703.
4. Cheng J., Ke Y., and Ng W. A survey on algorithms for mining frequent itemsets over data streams. Knowledge and Int. Syst., 16(1):1–27, 2008.
5. Chi Y., Wang H., Yu P.S., and Muntz R.R. Catch the moment: maintaining closed frequent itemsets in a data stream sliding window. Knowl. Inf. Syst., 10(3):265–294, 2006.
6. Cheung D.W., Han J., Ng V., and Wong C.Y. Maintenance of discovered association rules in large databases: an incremental updating technique. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 106–114.
7. Cheung D.W., Lee S.D., and Kao B. A general incremental technique for maintaining discovered association rules. In Proc. 5th Int. Conf. on Database Systems for Advanced Applications, 1997, pp. 185–194.
8. Giannella C., Han J., Pei J., Yan X., and Yu P.S. Mining frequent patterns in data streams at multiple time granularities. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.). Data Mining: Next Generation Challenges and Future Directions. AAAI, 2004.
9. Gouda K. and Zaki M.J. Efficiently mining maximal frequent itemsets. In Proc. 2001 IEEE Int. Conf. on Data Mining, 2001, pp. 163–170.
10. Manku G. and Motwani R. Approximate frequency counts over data streams. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 346–357.
11. Otey M.E., Parthasarathy S., Wang C., Veloso A., and Meira W. Parallel and distributed methods for incremental frequent itemset mining. IEEE Trans. Syst. Man Cybern. B, 34(6):2439–2450, 2004.
12. Teng W.-G., Chen M.-S., and Yu P.S. A regression-based temporal Pattern Mining Scheme for Data Streams. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 98–104.
13. Thomas S., Bodagala S., Alsabti K., and Ranka S. An efficient algorithm for the incremental updation of association rules in large databases. In Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining, 1997, pp. 263–266.
14. Veloso A., Meira Jr. W., and de Carvalho M., Pôssas B., Parthasarathy S., and Zaki M.J. Mining frequent itemsets in evolving databases. In Proc. SIAM International Conference on Data Mining, 2002.

Association Rule Visualization

▶ Visual Association Rules

Association Rules

Jian Pei
Simon Fraser University, Burnaby, BC, Canada

Definition

Let $\mathcal{I}$ be a set of *items*, where each item is a literal. A *transaction* $T \subseteq \mathcal{I}$ is a subset of $\mathcal{I}$. Association rules are defined on a set of transactions $\mathcal{T}$.

An association rule R is in the form of $X \rightarrow Y$, where X and $Y$are two sets of items, that is, $X, Y \subseteq \mathcal{I}$. R is associated with two measures, the *support* $sup(R)$ and the confidence $conf(R)$. The support $sup(R)$ is the probability that X appears in a transaction in $\mathcal{T}$. The confidence $conf(R)$ is the conditional probability that when X appears in a transaction, Yalso appears.

Historical Background

The concept of association rules were firstly proposed by Agrawal et al. [1] for market basket analysis. A well known illustrative example of association rules is "Diaper → Beer" which can be explained by the fact that, when dads buy diapers for their babies, they also buy beer at the same time for their weekends game watching.

Apriori, an efficient algorithm for mining association rules, was developed by Agrawal and Srikant [3], while the similar idea was explored by Mannila et al. [13]. There have been many studies trying to improve the efficiency of Apriori (refer to entry "Apriori property and breadth-first search algorithms"). A pattern-growth approach for mining frequent itemsets without candidate generation was developed by Han et al. [8], and has been further improved by many studies since 2001 (see entry "Pattern-growth methods").

To remove redundancy in association rules, Pasquier et al. [15] proposed the notion of frequent closed itemsets using formal concept analysis. Several efficient algorithms have been developed (refer to entry "Closed itemset mining and non-redundant association rule mining").

Association rules have been extended in several ways, such as sequential patterns and sequential association rules (refer to entry "Sequential patterns"), spatial association rules [9], cyclic association rules [14], negative association rules [19], intertransaction association rules [12], multilevel generalized association rules [7,20], and quantitative association rules (refer to entry "Quantitative association rules").

In addition to support and confidence, some other interestingness measures for association rules were explored, such as [5,18].

Foundations

Let $\mathcal{I}$ be a set of *items*, where each item is a literal. An *itemset* X is a subset of items, that is, $X \subseteq \mathcal{I}$. A *transaction* $T \subseteq \mathcal{I}$ consists of a *transaction id* and an itemset. A *transaction database* $\mathcal{T}$ is a multiset of transactions.

An association rule R is in the form of $X \rightarrow Y$, where X and $Y$are two itemsets, that is, $X, Y \subseteq \mathcal{I}$. R is associated with two measures, the *support* $sup(R)$ and the confidence $conf(R)$. The support $sup(R)$, given by $sup(R) = Pr(X)$, is the probability that X appears in a transaction in $\mathcal{T}$. The confidence $conf(R)$, given by $conf(R) = \frac{sup(X \cup Y)}{sup(X)} = Pr(Y|X)$, is the conditional probability that when X appears in a transaction, Y also appears.

Given a transaction database $\mathcal{T}$, a minimum support threshold *minsup*, and a minimum confidence threshold *minconf*, the problem of *association rule mining* is to find the complete set of association rules whose supports are at least *minsup* and whose confidences are at least *minconf*.

Association rules can be mined in two steps. In the first step, the complete set of frequent itemsets are identified. An itemset is called *frequent* if $Pr(X) \geq minsup$. In the second step, frequent itemsets are used to generate association rules.

More often than not, to make association rule mining interesting, a user may specify a *minimum support threshold min_sup* and a *minimum confidence threshold min_conf*. Then, only the association rules whose supports and confidence pass those thresholds, respectively, should be returned. Alternatively, in some situations, a user may want to find the top-*k* association rules with the largest support and/or confidence. Some other kinds of constraints can also be specified. Such thresholds and constraints may be used by some association rule mining methods to speed up the mining procedure.

Key Applications

Association rules have been extensively mined and used in many applications. For example, mining association rules about customers' market baskets helps to identify the products that customers like to purchase together, or some products that may trigger the purchases of some other products. Such information can help business in one way or another. For example, knowing that the purchase of diapers may potentially lead to the purchase of beer, a store can put beer beside diapers so that the sales of beer can be boosted.

Association rules are also mined on biological data and clinic data. For example, mining the association rules among symptoms and disease can help to diagnose diseases.

An important application of association rules is to construct classifiers using association rules. Technically, association rules among features and the target class labels can be mined and the rules can be used to make prediction on cases with unknown class labels. Research has found that associative classifiers, classifiers using association rules, are accurate and highly understandable in a few applications such as those with many features [6,10,11].

Cross-references

▶ Approximation of Frequent Itemsets
▶ Apriori Property and Breadth-First Search Algorithms
▶ Associative Classifiers
▶ Closed Itemset Mining and Non-Redundant Association Rule Mining
▶ Data Mining
▶ Emerging Patterns
▶ Frequent Itemset Mining with Constraints
▶ Frequent Itemsets and Association Rules
▶ Pattern-Growth Methods
▶ Quantitative Association Rules
▶ Sequential Patterns
▶ Sequential Patterns with Constraints

Recommended Reading

1. Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 207–216.
2. Agrawal R., Mannila H., Srikant R., Toivonen H., and Verkamo A.I. Fast discovery of association rules. In Advances in Knowledge Discovery and Data Mining, U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (eds.). AAAI/MIT, Menlo Park, CA/Cambridge, MA, 1996, pp. 307–328.
3. Agrawal R. and Srikant R. Fast algorithms for mining association rules. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
4. Agrawal R. and Srikant R. Mining sequential patterns. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 3–14.
5. Brin S., Motwani R., and Silverstein C. Beyond market basket: generalizing association rules to correlations. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 265–276.
6. Dong G. and Li J. Efficient mining of emerging patterns: discovering trends and differences. In Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 43–52.
7. Han J. and Fu Y. Discovery of multiple-level association rules from large databases. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 420–431.
8. Han J., Pei J., and Yin Y. Mining frequent patterns without candidate generation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 1–12.
9. Koperski K. and Han J. Discovery of spatial association rules in geographic information databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 47–66.
10. Li W., Han J., and pei J. CMAR: accurate and efficient classification based on multiple class-association rules. In Proc. 2001 IEEE Int. Conf. on Data Mining, 2001, pp. 369–376.
11. Liu B., Hsu W., and Ma Y. Discovering the set of fundamental rule changes. In Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2001, pp. 335–340.
12. Lu H., Han J., and Feng L. Stock movement and n-dimensional inter-transaction association rules. In Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 1998, pp. 1201–1217.
13. Mannila H., Toivonen H., and Verkamo A.I. Efficient algorithms for discovering association rules. In Proc. AAAI 1994 Workshop Knowledge Discovery in Databases, 1994, pp. 181–192.
14. Özden B., Ramaswamy S., and Silberschatz A. Cyclic association rules. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 412–421.
15. Pasquier N., Bastide Y., Taouil R., and Lakhal L. Discovering frequent closed itemsets for association rules. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 398–416.
16. Pei J., Han J., Lu H., Nishio S., Tang S., and Yang D. H-Mine: hyper-structure mining of frequent patterns in large databases. In Proc. 2001 IEEE Int. Conf. on Data Mining, 2001, pp. 441–448.
17. Pei J., Han J., and Mao R. CLOSET: an efficient algorithm for mining frequent closed itemsets. In Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp. 11–20.
18. Piatetsky-Shapiro G. Discovery, analysis, and presentation of strong rules. In Knowledge Discovery in Databases, G. Piatetsky-Shapiro and W. Frawley (eds.). AAAI/MIT, Menlo Park, CA/Cambridge, MA, 1991, pp. 229–238.
19. Savasere A., Omiecinski E., and Navathe S. Mining for strong negative associations in a large database of customer transactions. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 494–502.
20. Srikant R. and Agrawal R. Mining generalized association rules. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 407–419.

Associative Classification

► Classification by Association Rule Analysis

Asymmetric Encryption

Ninghui Li
Purdue University, West Lafayette, IN, USA

Synonyms

Public-key encryption

Definition

Asymmetric encryption, also known as public-key encryption, is a form of data encryption where the encryption key (also called the public key) and the corresponding decryption key (also called the private key) are different. A message encrypted with the public key can be decrypted only with the corresponding private key. The public key and the private key are related mathematically, but it is computationally infeasible to derive the private key from the public key. Therefore, a recipient could distribute the public key widely. Anyone can use the public key to encrypt messages for the recipient and only the recipient can decrypt them.

Key Points

A public-key encryption algorithm requires a trapdoor one-way function, i.e., a function that is easy to compute but hard to invert unless one knows some secret trapdoor (i.e., the private key). Existing public-key encryption algorithms are based on computational problems in number theory.

The most well-known public-key encryption algorithms include RSA and El Gamal. RSA uses exponentiation modulo a product of two large primes to encrypt and decrypt, and its security is connected to the presumed difficulty of factoring large integers. The El Gamal cryptosystem relies on the difficulty of the discrete logarithm problem. The introduction of elliptic curve cryptography in the mid 1980s has yielded a new family of analogous public-key algorithms. Elliptic curves appear to provide a more efficient way to leverage the discrete logarithm problem, particularly with respect to key size.

Cross-references

► Data Encryption
► Symmetric Encryption

Recommended Reading

1. Diffie W. and Hellman M.E. New directions in cryptography. IEEE Trans. Inf. Theory, 22:644–654, 1976.
2. El Gamal T. A public key cryptosystem and a signature scheme based on discrete logarithms. In Advances in Cryptology: Proc. CRYPTO ’84, LNCS, vol. 196, Springer, 1985, pp. 10–18
3. Rivest R.L., Shamir A., and Adleman L.M. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21:120–126, 1978.

ATA

► Storage Protocols

Atelic Data

Vijay Khatri[1], Richard T. Snodgrass[2], Paolo Terenziani[3]
[1]Indiana University, Bloomington, IN, USA
[2]University of Arizona, Tucson, AZ, USA
[3]University of Turin, Turin, Italy

Synonyms

Snapshot data; Point-based temporal data

Definition

Atelic data is temporal data describing facts that do not involve a goal or culmination. In ancient Greek, *telos* means “goal,” and α is used as prefix to denote negation. In the context of temporal databases, atelic data is that data for which both *upward* and *downward* (temporal) *inheritance* holds. Specifically,

- *Downward inheritance*. The *downward inheritance* property implies that one can infer from temporal data *d* that holds at valid time *t* (where t is a time period) that *d* holds in any sub-period (and sub-point) of *t*.
- *Upward inheritance*. The *u pward inheritance* property implies that one can infer from temporal data *d* that holds at two consecutive or overlapping time

periods t_1 and t_2 that d holds in the union time period $t_1 \cup t_2$.

Atelic data is differentiated from telic data, in which neither upward nor downward inheritance holds.

Key Points

Starting from Aristotle [1], researchers in areas such as philosophy, linguistics, cognitive science and computer science have noticed that different types of facts can be distinguished according to their temporal behavior. Specifically, since atelic facts do not have any specific goal or culmination, they can be seen as "temporally homogeneous" facts, so that both upward and downward inheritance holds for them. For example, the fact that an employee (say, John) works for a company (say, ACME) would be considered atelic because of lack of goal or accomplishment. As a consequence, if John has worked for ACME from January 20, 2007 to September 23, 2007, it can be correctly inferred that John was working for ACME in May 2007 (or at any specific time point in May; therefore, downward inheritance does hold); furthermore, from the additional fact that John has also worked for ACME from September 23, 2007 to February 2, 2008, it can be correctly inferred that John worked for ACME from January 20, 2007 to February 2, 2008 (therefore, upward inheritance does hold).

In Aristotle's categorization, all possible facts are divided into two categories, telic and atelic; a telic fact, e.g., "John built a house" has goal or culmination [1].

Since both upward and downward inheritance properties hold for atelic data, such data supports the conventional "snapshot-by-snapshot" (i.e., point-based) viewpoint: the intended semantics of a temporal database is the set of conventional (atemporal) databases holding at each time point. As a consequence, most approaches to temporal databases support (only) atelic facts even if, in several cases, the representation allows, as a syntactic sugar, the use of periods to denote convex sets of time points. An integrated temporal database model that supports both telic and atelic data semantics has been developed [2].

Cross-references

- ► Period-Stamped Temporal Models
- ► Point-Stamped Temporal Models
- ► Telic Distinction in Temporal Databases

Recommended Reading

1. Aristotle. The Categories. On Interpretation. Prior Analytics. Harvard University Press, Cambridge, MA, 2002.
2. Terenziani P. and Snodgrass R.T. Reconciling point-based and interval-based semantics in temporal relational databases: a proper treatment of the telic/atelic distinction. IEEE Trans. Knowl. Data Eng., 16(4):540–551, 2004.

Atomic Event

JONAS MELLIN, MIKAEL BERNDTSSON
University of Skövde, Skövde, Sweden

Synonyms

Primitive event

Definition

An atomic event is considered to be indivisible and instantaneous.

Key Points

If an event is non-instantaneous, then it is possible to divide into a beginning of this event (an initiator) and an ending of this event (a terminator). Therefore, atomic events must be instantaneous. In active database literature, the concept primitive event is typically used instead of atomic event. The major reason is probably that system primitives and application primitives are not distinguished.

Cross-references

- ► Composite Event
- ► Event
- ► Event Detection
- ► Event Specification

Atomicity

GERHARD WEIKUM
Max-Planck Institute for Informatics, Saarbruecken, Germany

Definition

The *atomicity of actions* on a database is a fundamental guarantee that database systems provide to application programs. Whatever state modifications an atomic action may perform are guaranteed to be executed in an *all-or-nothing* manner: either all state changes caused by the action will be installed in the database or none. This property is important in the potential presence of failures that could interrupt the atomic action. The database system prepares itself for this case by *logging* state modifications and providing automated *recovery* as part of the failure handling or system restart. These implementation aspects are transparent to the application program and are thus a major relief for the programs' failure handling and boost the application development productivity.

Historical Background

Since the early 1970s (or even earlier), transaction processing systems for airline reservations and debit/credit banking had means for recovery and concurrency control that were similar to atomic actions. However, these implementation techniques were hardly documented in publicly available literature and still far from a principled, universal solution. The major credit for the modern concept of atomicity (and transactions) belongs to the 1998 Turing Award winner Jim Gray [6–8]. Closely related notions of atomic actions have been proposed by other authors at around the same time, including [11–13], which in turn were inspired by the informal work on "spheres of control" by Bjork and Davies [3,4].

Foundations

As an example for the importance of atomic actions, consider a sequence of steps that read and write two bank-account records, x and y, in order to transfer some amount of money from account x to account y:

$$R(x)W(x)R(y)W(y).$$

If there is a system failure after writing x but before writing y, the underlying database becomes inconsistent, with the transferred money seemingly lost in "mid-flight." Even worse, the application program may not even know if this problem actually occurred or not (the system may have succeeded in writing y just before it crashed but could not send a return code anymore). If, on the other hand, the database system executes the entire step sequence as an atomic action, the failure handling for the application program becomes much simpler as it can always restart on a clean, consistent database.

In database systems the atomic actions themselves can be flexibly defined by the application programs, by demarcating the begin and end of a *transaction* with explicit interface calls. For example, an entire sequence of SQL command invocations can be made atomic. By default, usually every individual SQL operation is guaranteed to be atomic. These guarantees are part of the *transactional ACID properites*: atomicity, consistency-preservation, isolation, durability [8]. In some scientific communities outside of database systems research, atomicity is meant to include the *isolation* guarantee. Atomic actions are then (alternatively) defined to be state-modification sequences whose effects are ensured (by the underlying run-time environment) to be equivalent to *indivisible actions* with all effect appearing to be instantaneous upon the completion of the entire sequence. In the presence of concurrent accesses to the same shared data, this combined atomicity/isolation property provides the illusion, despite the fact that in reality accesses by different programs are interleaved. Defining this principle in formal terms leads to the concept of *serializability* and methods for *concurrency control* as part of the database (or other run-time) system [2,6,14].

As an example for the importance of isolation (as an additional property of atomic actions), consider an extended variant of the earlier example. Assume to the the money-transfer process – now viewed as a transaction T1 – , a second transaction T2 reads both bank-account records x and y in order to analyze financial portfolios and perform some kind of risk assessment. The following concurrent execution, with time proceeding from left to right, could be possible:

T1:	R(x) W(x)		R(y) W(y)
T2:		R(x) R(y)	

With this step interleaving, transaction T2 would see an inconsistent database, namely, the state of x after money is withdrawn from x and the state of y before money is deposited there. This may lead to a distorted analysis and false conclusions in the decision making of a financial broker. Running both T1 and T2 as atomic and isolated transactions would prevent this particular interleaving and guarantees that only such executions are allowed that are provably equivalent to a sequential execution where such an anomaly is impossible.

The atomicity guarantee includes all "side effects" of an action as far as the database state is concerned. For example, the effects of a database trigger are covered by the guarantee, but effects outside of the database such as sending a message are outside the scope of the database system guarantees. Modern application servers and message brokers, on the other hand, may provide such guarantees about messages (e.g., atomic multicasts) and application state (e.g., specific program variables) beyond the database. A traditional implementation technique to this end is to support failure-resilient queues. Modern database systems have integrated such message queues and application state management and extend their atomic actions to them, providing more comprehensive *application recovery*. A new research trend in programming languages is to provide atomicity guarantees to arbitrary programs, not just database applications, in order to simplify exception handling and generally ease programmers' work. For example, method invocations in an object-oriented language could be made atomic by means of an underlying *transactional memory* as part of the language's run-time system (and possibly even hardware architecture).

The atomicity concept simplifies failure handling at the application program level, but it does not mask failures. Rather a typical approach is that the program notices the failing of an atomic action by a corresponding system return code (for the atomic action invocation itself, for the end-of-action demarcation call, or upon the next interaction with the database system if the previous call simply timed out without any response), and then has to retry the action. This paradigm still requires explicit coding for the retrying, and this may require special care about non-idempotent effects or additional system guarantees and state testing for ensuring *idempotence*. Some advanced methods for application recovery can automate these re-trials and testing for non-idempotence, thus strengthening the all-or-nothing guarantee for atomicity into an *exactly-once execution* guarantee with complete failure masking [1].

On the other hand, for some data-intensive applications outside of database systems, atomicity may be an overly strong property if applied to entire processes; this holds particularly for long-lived workflows and cooperative work. Although these applications still benefit from atomic actions for smaller-grained operations, additional forms of *relaxed atomicity* or extended atomicity would be desirable. The database research community has developed a variety of such models, most notably, the model of open nested transactions and the ACTA framework [5,9].

Future Directions

Atomicity is a ground-breaking, fundamental contribution that first emerged in database systems, but is increasingly pursued also by other research communities like programming languages, operating systems, dependable system design, and also formal reasoning and program verification [10]. There are several strategic reasons for this growing interest and extended application of atomic actions:

- Web Services, long-running workflows across organizations, large scale peer-to-peer platforms, and ambient-intelligence environments with huge numbers of mobile and embedded sensor/actor devices critically need support for handling or even masking concurrency and component failures, and may mandate rethinking the traditional atomicity concept.
- There is a proliferation of open systems where applications are constructed from pre-existing components. The components and their configurations are not known in advance and they can change on the fly. Thus, it is crucial that atomicity properties of components are composable and that one can predict and reason about the behavior of the composite system.
- Modern applications and languages like Java lead millions of developers into concurrent programming. This is a drastic change from the classical situation where only a few hundred "five-star wizard" system programmers and a few thousand programmers working in scientific computing on parallel supercomputers would have to cope with the inherently complex issues of concurrency and advanced failure handling.
- On an even broader scale, the drastically increasing complexity of the new and anticipated applications will require enormous efforts and care towards dependable systems or it may lead into a major "dependability crisis." Atomicity is an elegant basic asset to build on in the design, implementation, and composition of complex systems and the reasoning about system behavior and guaranteed properties.

Cross-references

- ▶ ACID Properties
- ▶ Concurrency Control
- ▶ Open Nested Transactions
- ▶ Software Transactional Memory
- ▶ System Recovery
- ▶ Transaction

Recommended Reading

1. Barga R.S. and Lomet D.B. German Shegalov, Gerhard Weikum: Recovery guarantees for Internet applications. ACM Trans. Internet Technol., 4(3):289–328, 2004.
2. Bernstein P.A. and Hadzilacos V. Nathan Goodman: Concurrency Control and Recovery in Database Systems. Addison-Wesley, MS, 1987.
3. Bjork L.A. Recovery scenario for a DB/DC system. In Proc. 1st ACM Annual Conference, 1973, pp. 142–146.
4. Davies C.T. Recovery semantics for a DB/DC system. In Proc. First ACM Annual Conference, 1973, pp. 136–141.
5. Elmagarmid A.K. (ed.). Database Transaction Models for Advanced Applications. Morgan Kaufmann, San Fransisco, CA, 1992.
6. Eswaran K.P., Gray J., Lorie R.A., and Traiger I.L. The Notions of consistency and predicate locks in a database system. Commun. ACM 19(11):624–633, 1976.
7. Gray J. Notes on Database Operating Systems. In Operating Systems – An Advanced Course, Springer, London, UK, 1978.
8. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Fransisco, CA, 1993.
9. Jajodia S. and Kerschberg L. (eds.). Advanced Transaction Models and Architectures Kluwer, Noewell, MA, 1997.
10. Jones C.B., Lomet D.B., Romanovsky A.B., Weikum G., Fekete A., Gaudel M.-C., Korth H.F., Rogério de Lemos, J., Moss E.B., Rajwar R., Ramamritham K., Randell B., and Rodrigues L. The atomic manifesto: a story in four quarks. ACM SIGMOD Rec., 34(1):63–69, 2005.
11. Lampson B. Atomic Transactions. In: Distributed Systems – \Architecture and Implementation, Springer, New York, NJ, 1981.
12. Lomet D.B. Process structuring, synchronization, and recovery using atomic actions. In Proc. ACM Conf. on Language Design for Reliable Software, 1977, pp. 128–137.
13. Randell B. System structure for software fault-tolerance. IEEE Trans. Softw. Eng., 1(2):221–232, 1975.
14. Weikum G. and Vossen G. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Fransisco, CA, 2002.

Attribute or Value Correspondence

▶ Schema Matching

Audible Sound

▶ Audio

Audio

Lie Lu[1], Alan Hanjalic[2]
[1]Microsoft Research Asia, Beijing, China
[2]Delft University of Technology, Delft,
The Netherlands

Synonyms

Audible sound

Definition

Audio refers to audible sound – the sound perceivable by the human hearing system, or the sound of a frequency belonging to the *audible frequency range* (20-20,000 Hz). Audio can be generated from various sources and perceived as speech, music, voices, noise, or any combinations of these. The perception of an audible sound starts by the sound pressure waves hitting the eardrum of the outer ear. The generated vibrations are transmitted to the cochlea of the inner ear to produce mechanical displacements along the basilar membrane. These displacements are further transduced into electrical activity along the auditory nerve fibers, and finally "analyzed" and "understood" in the central auditory system [4,7].

Historical Background

The step from the fundamental definition of audio towards the concept of *audio signal* can be seen as a step towards the birth of the modern consumer electronics. An audio signal is a signal that contains audio information in the audible frequency range. The technology for generating, processing, recording, broadcasting and retrieving audio signals, first *analog* and later on *digital* ones, has rapidly grown for over a century, from the pioneering radio broadcasting and telephony systems to advanced mobile communication infrastructures, music players, speech recognition and synthesis tools, and audio content analysis, indexing and retrieval solutions. This growth may have been initiated by the research in the field of signal processing, but it has been maintained and has continuously gained in strength through an extensive interdisciplinary effort involving signal processing, information theory, human-computer interaction, psychoacoustics, psychology, natural language processing, network and wireless technology, and information retrieval.

Foundations

Digital Audio

An audio signal is an analog signal, which can be represented as a one-dimensional function $x(t)$, where t is a continuous variable representing time. To facilitate storage and processing of such signals in computers, they can be transformed into *digital signals* by *sampling* and *quantization*.

Sampling is the process in which one audio signal value (*sample*) is taken for each time interval (*sampling period*) T. This results in a *discrete audio* signal $x(n) = x(nT)$, where n is a numeric sequence. The sampling period T determines the *sampling frequency* that can be defined as $f = 1/T$. Typical sampling frequencies of digital audio are 8, 16, 32, 48, 11.025, 22.05, and 44.1 kHz (Hz represents the number of samples per second). Based on the Nyquist-Shannon sampling theorem, the sampling frequency must be at least 2 times larger than the band limit of the audio signal in order to be able to reconstruct the original analog signal back from its discrete representation. In the next step, each sample in the discrete audio signal is *quantized* with a bit resolution, which makes each sample be represented by a fixed limited number of bits. Common bit resolution is 8-bit or 16-bit per sample. The overall result is a digital representation of the original audio signal, that is referred to as *digital audio signal* or, if it is just considered as a set of bits, for instance for the purpose of storage and compression, as *digital audio data*.

Audio Coding and Compression

The digitization process described above leads to the basic standard of digital audio representation or *coding* named *Pulse Code Modulation* (PCM), which was developed in 1930-1940s. PCM is also the standard digital audio format in computers and Compact Disc (CD). PCM can be integrated into a widely used WAV format, which consists of the digital audio data and a *header* specifying the sampling frequency, bits per sample, and the number of audio channels.

As a basic audio coding format, PCM keeps all samples obtained from the original audio signal and all bits representing the samples. This format is therefore also referred to as *raw* or *uncompressed*. While it preserves all the information contained in the original analog signal, it is also rather expensive to store. For example, a one-hour *stereo* (A Cambridge Dictionary definition of stereo: a way of recording or playing sound so that it is separated into two signals and produces more natural sound) audio signal with 44.1 kHz sampling rate and 16 bits per sample requires 635MB of digital storage space. To save storage in computers and improve the efficiency of audio transmission, processing and management, *compression* theory and algorithms can be applied to decrease the size of a digital audio signal while still keeping the quality of the signal and communicated information at the acceptable level.

Starting with the variants of PCM, such as *Differential Pulse Code Modulation* (DPCM) and *Adaptive Differential Pulse Code Modulation* (ADPCM), a large number of audio compression approaches have been developed [5]. Some most commonly used approaches include MP3/ACC defined in the MPEG-1/2 standard [2,3], Windows Media Audio (WMA) developed by Microsoft, and RealAudio (RA) developed by RealNetworks. These approaches typically lead to a compressed audio signal being about 1/5 to 1/10 of the size of the PCM format.

Audio Content Analysis

Audio content analysis aims at extracting descriptors or *metadata* related to audio content and allowing content-based search, retrieval, management and other user actions performed on audio data. The research in the field of audio content analysis has built on the synergy of many scientific disciplines, such as signal processing, pattern recognition, machine learning, information retrieval, and information theory, and has been conducted in three main directions, namely *audio representation*, *audio segmentation*, and *audio classification*.

Audio representation refers to the extraction of audio signal properties, or *features*, that are representative of the audio signal composition (both in temporal and spectral domain) and audio signal behavior over time. The extracted features then serve as input into audio segmentation and audio classification. Audio segmentation aims at automatically revealing semantically meaningful temporal segments in an audio signal, which can then be grouped together (using e.g., a *clustering* algorithm) to facilitate search and browsing. Finally, an audio classification algorithm classifies a piece of audio signal into a predefined semantic class, and assigns the corresponding label (e.g., "applause," "action," "highlight," "music") to it for the purpose of text-based search and retrieval.

Audio Retrieval

Audio retrieval aims at retrieving sound samples from a large corpus based on their relation to an input query. Here, the query can be of different types and the expected results may vary depending on the application context. For example, in the *content-based retrieval* scenario, a user may use the text term "applause" to search for the audio clips containing the audio effect "applause." Clearly, the results obtained from audio classification can help annotate the corresponding audio samples, audio segments or audio tracks, and thus facilitate this search and retrieval strategy. However, audio retrieval can also be done by using an audio data stream as a query, i.e., by performing *query-by-example* [6]. For instance, one could aim at retrieving a song and all its variants by simply singing or humming its melody line.

In another retrieval scenario, the user may want to retrieve the exact match to the query or some information related to it. This typically falls into the application domain of *audio fingerprinting* [1]. An audio fingerprint is a highly compact feature-based representation of an audio signal enabling extremely fast search for a match between the signal and a large-scale audio database for the purpose of audio signal identification.

Key Applications

Audio technology is widely used in diverse applications areas, such as broadcasting, telephony, mobile communications, entertainment, gaming, hearing aids, and the management of large-scale audio/music collections.

Cross-references

- ▶ Audio Classification
- ▶ Audio Content Analysis
- ▶ Audio Representation
- ▶ Audio Segmentation
- ▶ Multimedia Data
- ▶ Video

Recommended Reading

1. Haitsma J. and Kalker T. A highly robust audio fingerprinting system with an efficient search strategy. J. New Music Res., 32 (2):211–221, 2003.
2. ISO/IEC 11172-3:1993. Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3: Audio, 1993.
3. ISO/IEC 13818-3:1998. Information technology - Generic coding of moving pictures and associated audio information – Part 3: Audio, 1998.
4. Pickles J.O. An Introduction to the Physiology of Hearing. Academic Press, London, UK, 1988.
5. Spanias A., Painter T., and Atti V. Audio Signal Processing and Coding. Wiley, NJ, 2007.
6. Wold E., Blum T., and Wheaton J. Content-based classification, search and retrieval of audio. IEEE Multimed., 3(3):27–36, 1996.
7. Yang X., Wang K., and Shamma S.A.Auditory representations of acoustic signals. IEEE Trans. Inform. Theory, 38:824–839, 1992.

Audio Categorization

▶ Audio Classification

Audio Characterization

▶ Audio Representation

Audio Classification

Lie Lu[1], Alan Hanjalic[2]
[1]Microsoft Research Asia, Beijing, China
[2]Delft University of Technology, Delft, The Netherlands

Synonyms

Audio categorization; Audio indexing; Audio recognition

Definition

Audio classification aims at classifying a piece of audio signal into one of the pre-defined *semantic classes*. It is typically realized as a combination of a *learning* step to learn a statistical model of each semantic class, and an *inference* step to estimate which semantic class is closest to the given piece of audio signal.

Historical Background

Audio classification associates *semantic labels* with audio signals, and can also be referred to as *audio indexing, audio categorization* or *audio recognition.* As such, audio classification plays an important role in facilitating search and retrieval in large-scale audio collections (databases). Semantic labels are used to represent semantic classes or *semantic concepts*, which can be defined at different abstraction and complexity levels. Typical examples of basic semantic audio classes are *speech*, *music*, *environmental sounds*, and *silence*, which can be detected rather effectively using the methods like [8,9,10,17]. Examples of mid-level semantic concepts are *key audio effects*, like *applause*, *cheer*, *ball-hit*, *whistling*, *car-racing*, *siren*, *gun-shot*, and *explosion*. Finally, the detection of higher-level semantic concepts, such as sport highlights [1,14–16] and action scenes in movies [3,11], is usually performed by analyzing the sequence of detected key audio effects.

Foundations

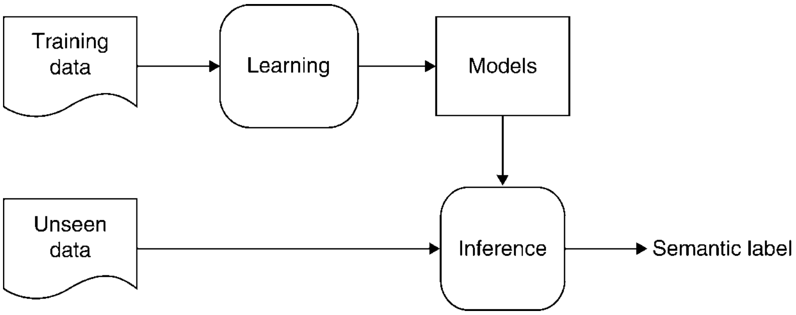
Figure 1 shows a general classification scheme, which is typically composed of two main steps: *learning* and *inference*. In the learning step, a model of each semantic class is built based on a set of training data, and with a specific learning scheme. Then, in the inference step, a new, unseen collection of data is associated with a semantic label, the model of which best resembles the properties of the data. Various schemes have been employed so far for realizing both the learning and inference steps. These schemes include sets of heuristic rules, vector Quantization (VQ), k-nearest neighbor (kNN), decision tree, Gaussian mixture model (GMM), support vector machine (SVM), boosting, Bayesian decision, hidden Markov model (HMM), and neural network. More information about these schemes can be found in [4,5].

While directly applying the previously mentioned learning and inference algorithms works well for straightforward classification tasks, there are a number of issues which should be taken into account when applying this scheme for audio classification. In the following sections, some critical issues are addressed that need to be considered when designing audio classification algorithms, and it is shown on the example of an existing approach how these issues can effectively be resolved. The classification cases discussed in the sections below concern the mid-level semantic concepts (key audio effects) and hierarchies of higher-level semantic concepts.

Key Audio Effect Detection

Several issues play a role in key audio effect detection in a continuous audio signal, and need to be resolved in order to secure reliable classification. The most important issues can be described as follows:

1. Key audio effect detection in a long, continuous audio signal, is typically approached by applying a sliding window of a given length (e.g., 0.5 s) to the signal. The audio segment captured by the window at a given time stamp is then used as the basic unit to associate with a key audio effect. An important implicit assumption here is that each segment corresponds to one and only one semantic class. However, a sliding window is often too short to capture one complete effect, which leads to over-segmentation. The sliding window could also be

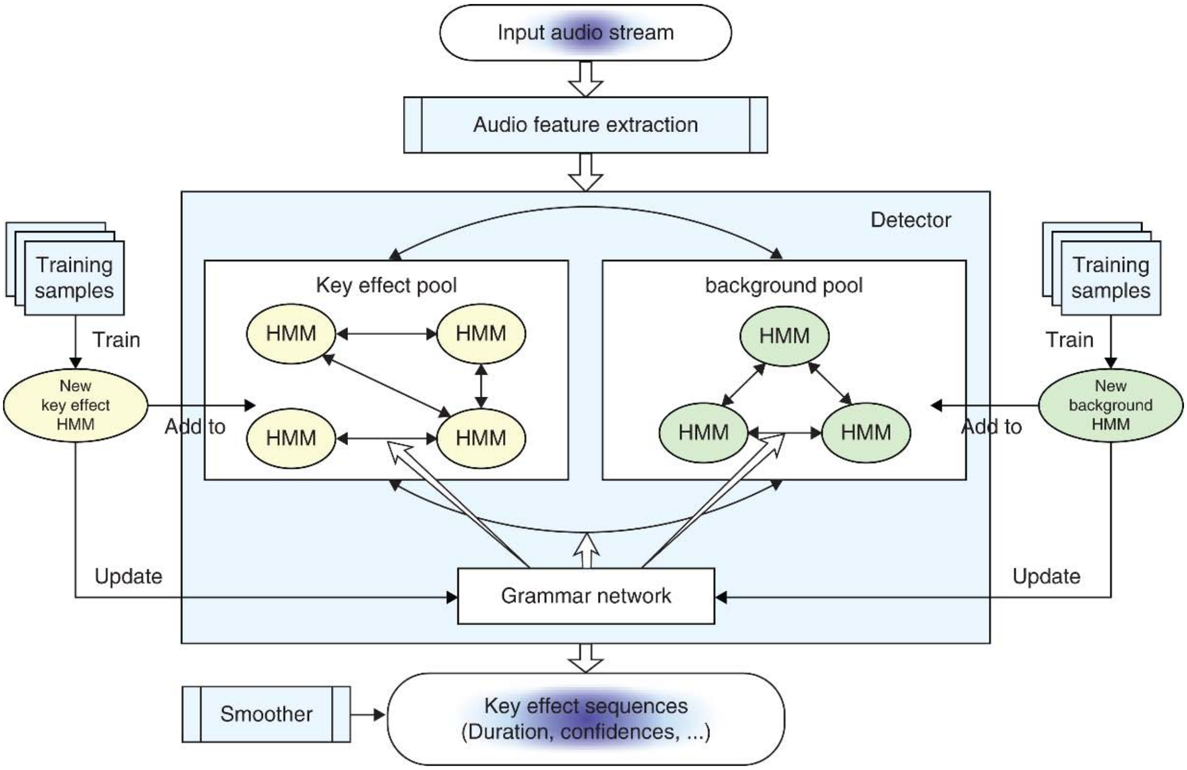


Audio Classification. Figure 1. An illustration of a general classification scheme.

too long and capture several effects within one segment.

2. The targeted audio effects are usually sparsely distributed over the signal, and there are plenty of non-target sounds that are to be rejected. Many existing approaches assume having a complete set of semantic classes available, and classify any audio segment into one of these semantic classes. Other methods use thresholds to discard the sounds with low classification confidence [3]. However, the threshold setting becomes troublesome for a large number of key effects.
3. Audio effects are usually related to each other. For example, some audio effects such as *applause* and *laughter* are likely to happen together, while others are not. Taking into account the transition relationships between audio effects is therefore likely to improve the detection of each individual effect.

To investigate the possibilities for effectively resolving the abovementioned issues when designing algorithms for key audio effect detection, the approach proposed in [2] will be discussed as an example. In this hierarchical, probabilistic framework, as illustrated in Fig. 2, an HMM model is first built for each key audio effect based on the complete set of audio samples, and the defined models are then used to compose the *Key Audio Effect Pool*. Then, comprehensive *background* models are also established to cover all non-target sounds that complement the targeted key effects. Thus, the non-target sounds would be detected as background sounds and excluded from the target audio effect sequence. Moreover, a higher-level probabilistic model is used to connect these individual models with a *Grammar Network*, in which the transition probabilities among various audio effects and background sounds are taken into account for finding the optimal audio effect sequence. Then, for a given input audio stream, the optimal audio effect sequence is found among the candidate paths using the Viterbi algorithm, and the location and duration of each key audio effect in the stream are determined simultaneously, without the need to pre-segment the audio



Audio Classification. Figure 2. The framework for audio effect detection [13], consisting of three main parts: key audio effect pool, background sound pool, and grammar network.

stream into audio segments. In the following both the learning and inference step from [2] are discussed in more detail.

Classifier Learning In the approach from [2], each key audio effect and background sound are modeled using HMMs, since HMM provides a natural and flexible way for modeling time-varying process [12]. The main issue that needs to be resolved for an HMM is the parameter selection, which includes (i) the optimal model size (the number of states); (ii) the number of Gaussian mixtures for each state, and (iii) the topology of the model.

To select the model size for a key audio effect category, one needs to balance the number of hidden states in the HMM and the computational complexity in the learning and inference processes. In general, a sufficient number of states are required to describe all the significant behavioral characteristics of a signal over time. However, when the number of states increases, the computational complexity grows dramatically and more training samples are required. Unlike speech modeling, in which the basic units such as tri-phones could be adopted to specify the number of states, general key audio effects lack such basic units, and thus make the choice of the state numbers difficult. As an example of an approach in this direction, a clustering-based method was proposed in [2,13,17] to estimate a reasonable number of states (model size) per audio effect. The clustering step was realized through an improved, unsupervised *k*-means algorithm, and the resulting number of clusters is taken as the model size.

The number of Gaussian mixtures per state is usually determined experimentally. For instance, the method from [2] adopts 32 Gaussian mixtures for each state in the HMM. This number is larger than those used in other related methods in order to secure a sufficient discriminative ability of the models to identify a large diversity of audio effects in general audio streams.

The most popular HMM topology is the left-to-right or the fully connected one. The left-to-right structure only permits transitions between adjacent states; while the fully connected structure allows transitions between any two states in the model. Different topologies can be used to model audio effects with different properties. For instance, for key audio effects with obvious time-progressive signal behavior, such as *car-crash* and *explosion*, the left-to-right structure should be adopted, while for audio effects without distinct evolution phases, such as *applause* and *cheer*, the fully connected structure is more suitable.

Regarding the background sound modeling, a straightforward approach is to build a large HMM, and train it with as many samples as possible. However, background sounds are very complex and diverse, and their feature vectors are typically widely scattered in the feature space, so that both the number of states and the Gaussian mixtures per state of such a HMM must be particularly large to secure a representation of all possible background sounds. As an alternative, the method from [2] modeled the background sounds as a set of subsets of basic audio classes. It is namely so that in most practical applications the background sounds can be further classified into a few basic categories, such as *speech*, *music*, and other *noise*. Thus, if background models could be trained from all these respective subsets, the training data would be relatively concentrated, and the training time would be reduced. Another advantage of building these subset models is that they could provide additional useful information in high-level semantic inference. For example, *music* is usually used in the background of movies, and *speech* is the most dominant component in talk shows. Following the discussion from above, three background models are built in [2] using the fully connected HMMs for *speech*, *music*, and *noise*. Here, *noise* is referred to as all background sounds except *speech* and *music*. To provide comprehensive descriptions of the background, 10 states and 128 Gaussian mixtures in each state are used in modeling.

The Grammar Network in Figure 2 is an analogy to a language model in speech processing. It organizes all the HMM models for continuous recognition. Two models are connected in the Grammar Network if the corresponding sounds are likely to occur after each other, both within and between the key audio effect pool and the background sound pool. For each connection, the corresponding transition probability is set and taken into account when finding the optimal effect sequence from the input stream.

The transition probabilities between two models can be statistically learned from a set of training data. If no sufficient training data are available, a heuristic approach can be deployed. For instance, the approach presented in [2] is based on the concept of *Audio Effect Groups*, and assumes that (i) only audio effects in the same group can happen subsequently, (ii) there should be background sounds between any two key audio effects belonging to different groups, and (iii) the transition probability is uniformly distributed per group. An audio effect group can be seen as a set of audio effects that usually occur together. An example Grammar Network with audio effect groups indicated as G_1-G_k is illustrated in Figure 3.

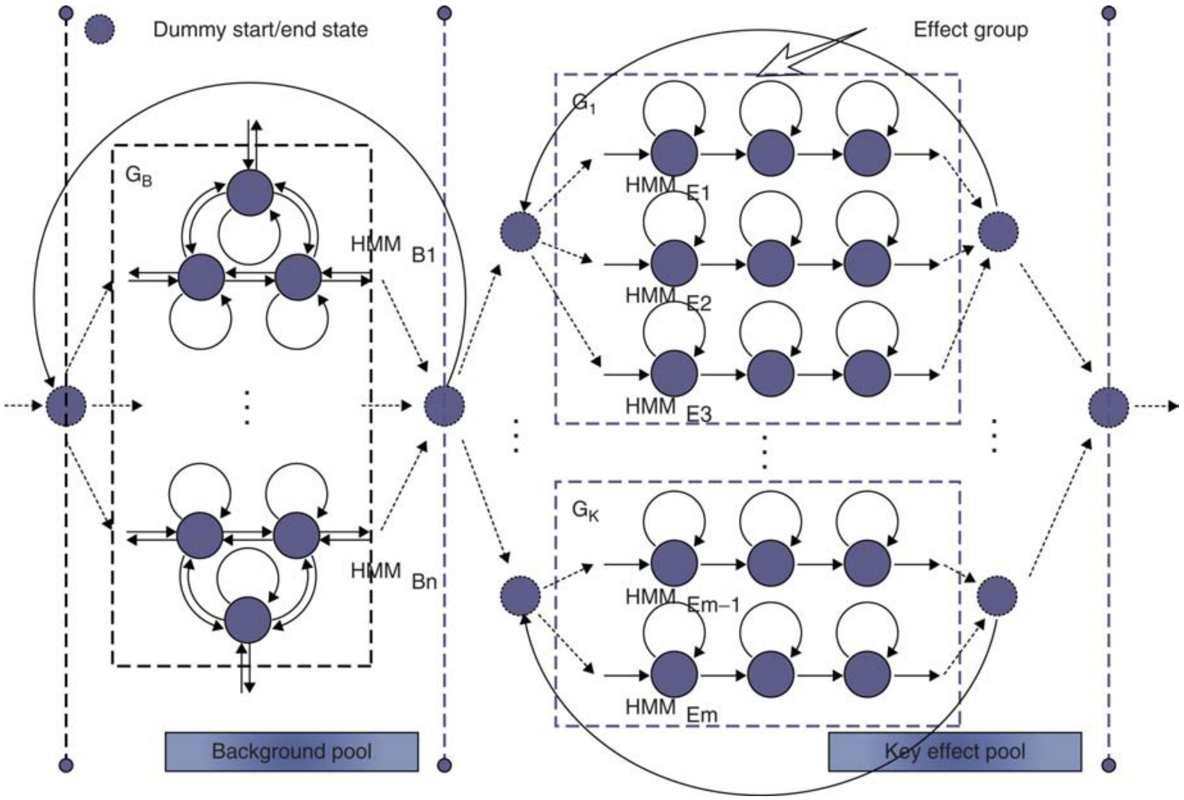
Probabilistic Inference Using the learned classification framework setup described above, the *Viterbi* algorithm can be used to choose the optimal state sequence from the continuous audio stream, as:

$$S_{optimal} = \arg\max_{s} \Pr(s|M, O). \qquad (1)$$

Here, s is the candidate state sequence, M represents the hierarchical structure, and O is the observation vector sequence of the input audio stream. In terms of practical realization of this classification scheme, the corresponding state and log-probability are obtained first for each audio frame. Then, a complete audio effect or background sound can be detected by merging adjacent frames belonging to the same sound model. Before this merging step, a smoothing filter can be applied to remove the classification outliers in the sequences of consecutive frames. The final classification confidence can be measured by averaging the log-probabilities of the classified audio frames. In addition, the starting time stamp and duration of each sound occurrence can be obtained simultaneously.

From Key Audio Effects to a Hierarchy of Semantic Concepts

Based on the obtained key audio effect sequence, methods can be developed to perform audio classification at a higher level, such as the level of audio events and scenes. While high-level semantic concepts can generally also be detected using the general scheme from Figure 1, using key audio effects as an



Audio Classification. Figure 3. An illustration of the *Grammar Network* with *Audio Effect Groups*, where G_k is the kth *Effect Group* and G_B is the *Background Sound Pool*. For convenience, all the key audio effect models are presented as 3-state left-to-right HMMs, and all the background models are denoted as 3-state fully connected HMMs. The dummy start and end states are used to link models, and make the structure more clear [2].

intermediate classification level has proved to be a more effective way to perform indexing at this level.

To infer high-level semantics from audio effects, most existing methods are rule-based [1,16], or employ a statistical classification [3,11]. Heuristic inference is straightforward and can be easily applied in practice. However, it is usually laborious to find a proper rule set if the situation is complex. For example, the rules usually involve many thresholds which are difficult to set; some rules may be in conflict with others, and some cases may not be well-covered. People are used to designing rules from a positive view but ignoring those negative instances, thus many false alarms are introduced although high recall can be achieved. Classification-based methods provide solutions from the view of statistical learning. However, the inference performance relies highly on the completeness and the size of the training samples. Without sufficient data, a positive instance not included in the training set will usually be misclassified. Thus these approaches are usually prone to high precision but low recall. Furthermore, it is inconvenient to combine prior knowledge into the classification process in these algorithms.

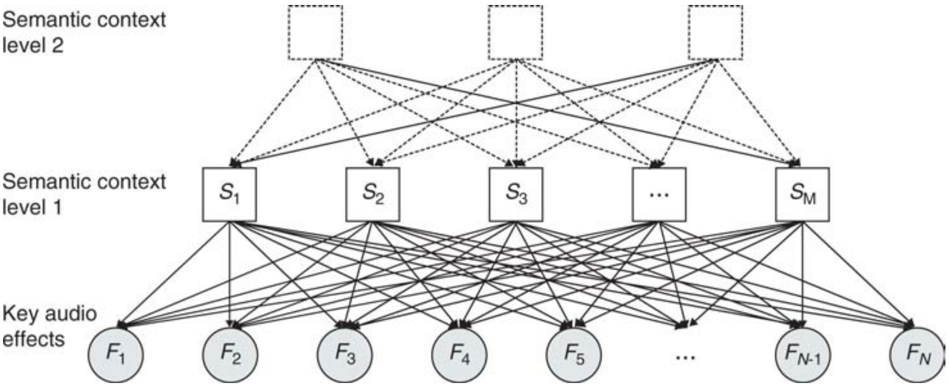
To integrate the advantages of heuristic and statistical learning methods, a Bayesian network-based approach is proposed in [2]. A Bayesian network [6] is a directed acyclic graphical model that encodes probabilistic relationships among nodes which denote random variables related to semantic concepts. A Bayesian network can handle situations where some data entries are missing, as well as avoid the overfitting of training data [6]. Thus, it weakens the influence from unbalanced training samples. Furthermore, a Bayesian network can also integrate prior knowledge by specifying its graphic structure.

Figure 4 illustrates the topology of an example Bayesian network with three layers. Nodes in the bottom layer are the observed audio effects. Nodes in the second layer denote high-level semantic categories such as scenes, while those in the top layer denote much higher semantic concepts. In Figure 4, the nodes in adjacent layers can be fully connected, or partially connected based on the prior knowledge of the application domain. For instance, if it is known *a priori* that some key effects have no relationships with a semantic category, the arcs from that category node to those key effect nodes could be removed. A Bayesian network with a manually specified topology utilizes human knowledge in representing the conditional dependencies among nodes, thus it can describe some cases which are not covered in the training samples.

The nodes in the upper layers are usually assumed to be discrete binaries which represent the presence or absence of a corresponding semantic category, while the nodes in the bottom layer produce continuous values of a Gaussian distribution

$$P(F_i|\mathbf{pa}_i) \sim N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)(1 \leq i \leq N) \tag{2}$$

where F_i is a 2-dimensional observation vector of the ith audio effect and is composed of the normalized duration of an effect and its detection confidence in a given semantic segment. The conditional argument $\mathbf{pa}_i$ denotes a possible assignment of values to the parent nodes of F_i, while $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ are the mean and



Audio Classification. Figure 4. An example of a Bayesian network for inference of a hierarchy of semantic concepts. Arcs are drawn from cause to effect. Following the convention, discrete variables are represented as squares while continuous variables are indicated as circles. Furthermore, observed variables are shaded, while hidden variables are not.

covariance of the corresponding Gaussian distribution respectively. In the training phase, all these conditional probability distributions are uniformly initialized and then updated by maximum likelihood estimation using the EM algorithm. In the inference process, the junction tree algorithm [7] can be used to calculate the occurrence probability of each semantic category. Thus, given the information on the audio effects, the semantics in each layer can be inferred based on the posterior probabilities. A semantic segment can be classified into the *c*th semantic category with the maximum marginal posterior probability:

$$c = \arg\max_{j} \Pr(S_j|F) \quad 1 \leq j \leq M$$
$$where \quad \mathbf{F} = \{F_1, F_2, \cdots, F_N\} \tag{3}$$

With this scheme, human knowledge and machine learning are effectively combined to perform high-level semantic inference. In other words, the topology of the network can be designed according to the prior knowledge of an application domain, and the optimized model parameters can then be estimated by statistical learning.

Key Applications

Audio classification is typically applied for content-based search and retrieval in and management of large-scale audio collections (databases).

Cross-references

- ▶ Audio Content Analysis
- ▶ Audio Representation
- ▶ Audio Segmentation

Recommended Reading

1. Baillie M., and Jose J.M. Audio-based event detection for sports video. In Proc. 2nd International Conference on Image and Video Retrieval, 2003, pp. 300–309.
2. Cai R., Lu L., Hanjalic A., Zhang H.-J., and Cai L.-H. A flexible framework for key audio effects detection and Auditory context inference. IEEE Trans. Audio Speech Lang. Process., 14 (3):1026–1039, 2006.
3. Cheng W.-H., Chu W.-T., and Wu J.-L. Semantic context detection based on hierarchical audio models. In Proc. 5th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, 2003, pp. 109–115.
4. Duda R.O., Hart P.E., and Stork D.G., Pattern Classification (2nd edn.), Wiley, New York, 2000.
5. Hastie T., Tibshirani R., and Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, New York, NY, 2001.
6. Heckerman D. A tutorial on learning with Bayesian networks. Microsoft Research, Redmond, Washington, Tech. Rep. MSR-TR-95–06, 1995.
7. Huang C. and Darwiche A. Inference in belief networks: a procedural guide. Int. J. Approx. Reason., 15(3):225–263, 1996.
8. Liu Z., Wang Y., and Chen T. Audio feature extraction and analysis for scene segmentation and classification. J. VLSI Signal Process. Syst. Signal Image Video Technol., 20(1–2):61–79, 1998.
9. Lu L., Zhang H.-J., and Jiang H. Content analysis for audio classification and segmentation. IEEE Trans. Speech Audio Process., 10(7):504–516, 2002.
10. Lu L., Zhang H.-J., and Li S. Content-based audio classification and segmentation by using support vector machines. ACM Multimed. Syst. J., 8(6):482–492, 2003.
11. Moncrieff S., Dorai C., and Venkatesh S. Detecting indexical signs in film audio for scene interpretation, In Proc. IEEE Int. Conf. on Multimedia and Expo, 2001, pp. 1192–1195.
12. Rabiner L.R. A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE, 77(2): 257–286, 1989.
13. Reyes-Gomez M.J., and Ellis D.P.W. Selection, parameter estimation, and discriminative training of hidden Markov models for general audio modeling, In Proc. IEEE Int. Conf. on Multimedia and Expo, 2003, pp. 73–76.
14. Rui Y., Gupta A., and Acero A. Automatically extracting highlights for TV baseball programs, Proc. 8th ACM Int. Conf. on Multimedia, 2000, pp. 105–115.
15. Xiong Z., Radhakrishnan R., Divakaran A., and Huang T.S. Audio events detection based highlights extraction from baseball, golf and soccer games in a unified framework, In Proc. IEEE Int. Conf. on Multimedia and Expo, 2003, vol. 3, pp. 401–404.
16. Xu M., Maddage N., Xu C.-S., Kankanhalli M., and Tian Q. Creating audio keywords for event detection in soccer video. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2003, pp. 281–284.
17. Zhang T. and Jay Kuo C.C. Hierarchical system for content-based audio classification and retrieval, In Proc. SPIE: Multimedia Storage and Archiving Systems III, vol. 3527, 1998, pp. 398–409.

Audio Content Analysis

Lie Lu[1], Alan Hanjalic[2]
[1]Microsoft Research Asia, Beijing, China
[2]Delft University of Technology, Delft,
The Netherlands

Synonyms

Audio information retrieval; Semantic inference in audio

Definition

An *audio signal* is a signal that contains information in the audible frequency range. Audio content analysis refers to a set of theories, algorithms and systems that aim at extracting descriptors or metadata related to audio content and allowing search, retrieval and other user actions performed on audio signals.

Historical Background

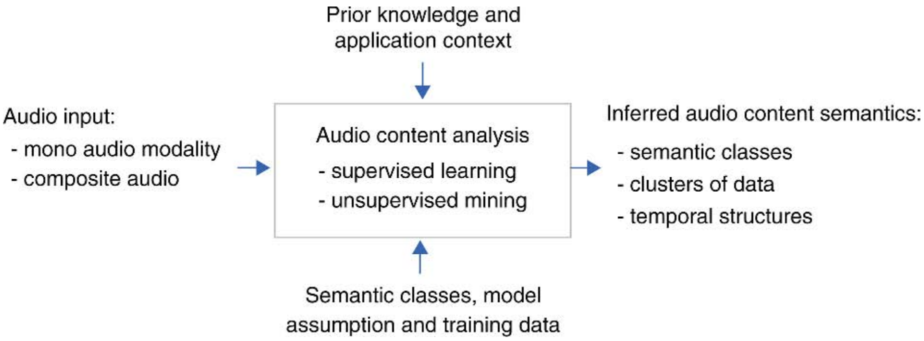
Multimedia content analysis has been one of the most booming research directions in the past years. With the objective of providing fast, natural, intuitive and personalized content-based access to vast multimedia data collections, and building on the synergy of many scientific disciplines, such as signal processing, pattern recognition, machine learning, information retrieval, information theory, natural language processing and psychology, the research initiative born around the end of the 1980s has succeeded in inspiring and mobilizing enormous number of researchers worldwide. This initiative turned into a broad research effort that has continuously gained in strength ever since. As an integrated part of multimedia (multimodal) documents, audio plays an important role in multimedia content analysis. In particular, audio content analysis can be combined with content analysis of visual information to jointly address semantic inference from *multimedia data* streams [5]. However, also taken separately, audio content analysis has been recognized as the key technology for providing easy access to rapidly growing audio archives, and in particular to music collections [3].

Foundations

The underlying problem of audio content analysis is to infer the information about the semantics of the content carried by an audio signal. Here, the audio signal can be a rather simple one consisting of one audio type only (e.g., pure speech, music), or a more complex audio signal (*composite audio*) resulting from a superposition of several audio types, like music, speech, audio effects, and noise. The audio content semantics includes the information on audio types (e.g., speech, music, noise, or any combination of these), on audio semantic classes (e.g., applause, solo instrument, guitar, speaker X or Y), and on audio content structure (e.g., the breaks between semantically coherent audio segments, clusters of *auditory scenes*).

The structure of a general audio content analysis system is illustrated in Figure 1. Depending on the levels at which prior knowledge is specified and the system is trained, this inference system can be realized by employing various approaches ranging from purely supervised to fully unsupervised ones. For example, if the scheme in Figure 1 is seen as a speech recognition system, the prior knowledge, such as the labeled audio data, dictionary and grammar, need to be pre-collected to train both an acoustic model and a language model in a supervised fashion [6]. Similarly, trained models of semantic classes such as car-racing, siren, gun-shot, and explosion can be used to detect the occurrences of these sounds in movie soundtracks [2,4]. Compared to these supervised realizations, an unsupervised approach can be employed to find clusters in audio data corresponding to auditory scenes [1,7], or to find "unusual" events in the sound track of a surveillance signal [9].

While the realizations of the Scheme in Figure 1 can be very different with respect to the types of input audio signals they handle, the prior knowledge and trained models they use, the types of inference techniques they employ (e.g., supervised versus unsupervised), and the inference results they are expected to provide (e.g., semantic categories versus clusters of data), the research targeting these realizations can be



Audio Content Analysis. Figure 1. A general audio content analysis scheme, specifying various possible types of input audio signals, prior knowledge and application context, the types of inference techniques and inference results.

said to follow three main directions that also roughly define the scope of the audio content analysis research field. These directions are *audio representation*, *audio segmentation*, and *audio classification*.

- Audio representation refers to the extraction of audio signal properties, or *features*, that are representative of the audio signal, such as short time energy, zero-crossing rate, pitch, or spectrum. Obtaining a compact feature-based representation of an audio signal can improve the efficiency of audio processing and benefit many applications based on such processing (e.g., audio retrieval).
- Audio segmentation aims to automatically reveal coherent and semantically meaningful temporal segments in an audio signal. Examples of such segments are those containing pure speech or music, or those corresponding to *auditory scenes* [7]. The segments discovered through segmentation can also be clustered together into semantically coherent groups [1] to provide the possibility for an easy content access (e.g., through browsing).
- While the abovementioned segmentation and clustering processes are typically *unsupervised*, audio classification classifies a piece of audio signal into one of the pre-defined semantic classes using *supervised* methods of machine learning and pattern classification. The semantic classes can be defined at the level of basic audio types (e.g., speech, music) [8], audio effects (e.g., applause, gun-shot, car chasing) [2,4], or auditory scenes (e.g., action, highlights, romance).

The term “audio” in the context of audio content analysis typically stands for general or *composite audio* signals [1]. When dealing with specific audio types, such as speech and music, and related applications, dedicated research has been deployed and led to a number of specific research directions like music information retrieval [3] and speech recognition [6].

Key Applications

Audio content analysis is typically applied for content-based search and retrieval in the management of large-scale audio (or multimedia) collections (databases).

Cross-references

▶ Audio Classification
▶ Audio Content Analysis
▶ Audio Representation
▶ Audio Segmentation
▶ Multimedia Data
▶ Video Content Analysis
▶ Video Scene and Event Detection

Recommended Reading

1. Cai R., Lu L., and Hanjalic A. Unsupervised Content Discovery in Composite Audio, Proc. IEEE Int. Conf. on Multimedia and Expo, 2005, pp. 628–637.
2. Cai R., Lu L., Hanjalic A, Zhang H.-J., and Cai L.-H. A Flexible Framework for Key Audio Effects Detection and Auditory Context Inference. IEEE Trans. Audio Speech Lang. Process., 14(3):1026–1039, 2006.
3. Casey M., et al. Content-Based Music Information Retrieval: Current Directions and Future Challenges. In Proc. IEEE, Special Issue on Advances in Multimedia Information Retrieval, 96(4): 668–696, 2008.
4. Cheng W.-H., Chu W.-T., and Wu J.-L. Semantic context detection based on hierarchical audio models. In Proc. 5th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, 2003, pp. 109–115.
5. Hanjalic A. Content-Based Analysis of Digital Video. Kluwer Academic, Norwell, MA, 2004.
6. Huang X, Acero A., and Hon H.W. Spoken Language Processing: A Guide to Theory, Algorithm, and System Development. Prentice, Upper Saddle River, NJ, 2001.
7. Lu L., Cai R., and Hanjalic A. Audio Elements based Auditory Scene Segmentation. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Vol. 5, 2006, pp. 17–20.
8. Lu L., Zhang H.-J., and Jiang H. Content analysis for audio classification and segmentation. IEEE Trans. Speech Audio Process., 10(7):504–516, 2002.
9. Radhakrishnan R., Divakaran A., and Xiong Z. A time series clustering based framework for multimedia mining and summarization using audio features. In Proc. 6th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, 2004, pp. 157–164.

Audio Feature Extraction

▶ Audio Representation

Audio Indexing

▶ Audio Classification

Audio Information Retrieval

▶ Audio Content Analysis

Audio Metadata

Werner Kriechbaum
IBM Development Lab, Böblingen, Germany

Synonyms

Music metadata

Definition

Audio, first used in 1934 refers to "Sound, esp. recorded or transmitted sound . . . and signals representing this" [Oxford English Dictionary, Oxford 2005, Vol. 1, p, 780].

Metadata is data about data of any sort in any media, describing an individual datum, content item, or a collection of data including multiple content items. In that way metadata facilitates the understanding, characterization, use and management of data.

Audio metadata is structured, encoded data that describes content and representation characteristics of audio entities to facilitate the automatic or semiautomatic identification, discovery, assessment, interpretation, and management of the described entities, as well as their generation, manipulation, and distribution.

Historical Background

Audio metadata predate audio data by centuries. Since antiquity artists and theoreticians alike were interested to classify the effects that could be produced by the combination of different tones and to devise rules for the proper composition of music [12]. The baroque *Affektenlehre* (doctrine of the affections) provided the composers of music with rules and schemes to express affects like love, fear, or hate, and on the other hand gave the listeners a reference system to decode the emotional content of a piece of music. Falling into oblivion with the end of the baroque area similar classification schemes resurfaced with the advent of silent films. Score snippets classified this way enabled the pianist accompanying the film to select music appropriate to the mood of the film scene on the fly.

Like many other technological innovations the modern history of audio metadata started in Bell Labs with the vocoder, developed by Homer Dudley during the late 20s and early 30s of the last century [2]. As part of a speech analysis/synthesis system the vocoder was used to reduce the amount of storage needed to store human speech. The analysis part of the system computed a spectrogram (a variant of a short-term power spectrum) and extracted the fundamental frequency of the speech signal – the first low-level audio metadata. This paved the way for the development of an ever increasing stream of audio analysis techniques that can be traced for example in the "Transactions of the IRE Professional Group on Audio" and its IEEE follow-on and spin-off journals (Available at: http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber = 8340).

MPEG-7 Audio standard [7], finalized in 2002, selected and standardized a set of low- and high-level descriptors from the plethora of available signal processing techniques and tools. MPEG-7 low-level audio descriptors are derived from the time-frequency analysis of the audio signal and include:

- Basic descriptors: e.g., the audio waveform envelope
- Basic spectral descriptors: e.g., the centroid of the audio spectrum
- Signal parameters: e.g., the fundamental frequency
- Timbral temporal descriptors: e.g., the logarithmic attack time
- Timbral spectral descriptors: e.g., the harmonic spectral centroid
- Spectral basis descriptors: which are low-dimensional projections of the spectrum
- Silence

The MPEG-7 high-level descriptors, which are usually collections of further low-level descriptors with additional context, comprise:

- Audio signature
- Musical instrument timbre
- Monophonic melody
- HMM sound model
- Spoken content model

Markup languages and among those most notably HyTime [6], provided the second major force that influenced the development of audio metadata. The linking concepts introduced by HyTime provided mechanisms to connect (textual) symbolic representations and (non-textual) realizations of a media entity without embedding the link in either. The Standard Music Description Language (SMDL) [5] – developed as a HyTime application and up to now not promoted from a draft standard to an international standard – applied these techniques to standardize architecture for the representation of music. From an SMDL point of view a single musical work is comprised of four domains:

- Logical domain: "The logical domain is the basic musical content – the essence from which all performances and editions of the work are derived, including virtual time values, nominal pitches, etc. The logical domain is describable as 'the composer's intentions with respect to pitches, rhythms, harmonies, dynamics, tempi, articulations, accents, etc.,'" [5, p. 5]
- Gestural domain: Any number of performances of the logical domain, e.g., a digital audio recording capturing a concert
- Visual domain: Any visual rendering of the logical domain, e.g., a score
- Analytical domain: Any number of music-theoretical analyses like e.g., Schenkerian analysis [A. Cadwallader/D. Gagné, Analysis of Tonal Music: A Schenkerian Approach, Oxford 1998]

There is only a faint echo of this concept in the MPEG-7 Audio standard but the ideas are taken up again in the IEEE P1599 Recommended Practice for Definition of a Commonly Acceptable Musical Application using the XML Language (MX) which is as the time of this writing still a draft but should undergo ballot soon. In the terminology of MX, the SMDL domains are called layers and the aspects of music addressed are refined to six:

- General: Information that applies to the piece of music as a whole, e.g., the opus number
- Logic: The symbolic description of the music (equivalent to the SMDL logical domain)
- Structural: Description of music objects and their causal relationship
- Notational: The score (similar to the SMDL visual domain)
- Performance: An audible rendering of the piece of music like e.g., MIDI, CSound, etc
- Audio: Links to and description of digital audio

The great success of the compact disc and the almost ubiquitous availability of tools to create own copies or compilations from digital audio in MP3 format spawned a further community-based audio metadata "standardization" initiative. Since according to the original Red Book specification [IEC 60908 Ed. 2.0 b:1999 Audio recording – Compact disc digital audio system] the compact discs did not include metadata like disc name, track names or author information the need arose to add this information from a supplemental database and to embed at least part of this information in an MP3 file generated from the audio disc. The basic idea to bind the audio disc to its database entry used by all such systems is to compute a hash based on the track information available in the CD's table of contents. This hash is used as key for the metadata stored in the database. An entry in the open-source GPL-licensed audio metadata database freedb [http://www.freedb.org/] contains the following information:

- DISCID: the hash key
- DTITLE: Artist and disc title separated by "/"
- DYEAR: Year the compact disc was released
- DGENRE: The genre in textual form. In addition to this genre entry the database is split into eleven different categories (blues, classical, country, data, folk, jazz, newage, reggae, rock, soundtrack, misc) to minimize hash collisions. The information in the genre field can conflict with the category and will do so when two discs from the same category produce the same hash key. The recommended resolution for the key collision is to put the second disc in another category.
- TTITLEN: The title of track N
- EXTD: Extended data (i.e., any interesting information) for the audio disc
- EXTTN: Extended data for track N

Like the original audio disc the MP3 standard [ISO/IEC 11172–3:1993 Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3: Audio] did not provide for the inclusion of textual material as part of the encoded audio. This did not deter the user community from enhancing MP3 files with textual information. One result of these efforts is the (informal) ID3 standard [http://www.id3.org/id3v2.4.0-structure and http://www.id3.org/id3v2.4.0-frames] which specifies how to prepend metadata to an MP3 file. There is a rich set of predefined ID3 frames for example:

- Text information frames: Information like album, author, artist(s), etc.
- URL link frames: Links to e.g., the web page of a performer.
- Event timing codes: Time stamps for events in the audio like e.g., verse start, refrain start, theme start, key change, theme end, profanity, profanity end.

- Unsynchronized lyrics or text transcription
- Synchronized lyrics or text.
- Equalization: Equalizer settings for the encoded audio.
- Attached picture

The tag set can be extended since ID3 parsers – like HTML parsers – have to ignore unknown tags.

Foundations

From a metadata point of view, audio is rather ill-defined. All but the basic technical metadata describing the recording setup and the physical metadata characterizing the recorded signal, are specific for the recorded material and require content specific expertise. What makes sense for the description of the recording of a starting airplane is quite different from the data needed to describe birdsong or an opera recording. And even the description of the audio signal on the physical level is not without problems when the metadata are collected for human usage. On their way from the eardrum to the brain, auditory signals undergo a non-linear transformation caused by one's sound perception system [3,9] and all the concepts our cognition forms about audio events are based on this transformed signal. The perceived intensity (loudness) of a signal, for example, is quite different from the physical intensity of the signal. Low-level descriptors of audio signals have to take these differences into account, especially when they are used to derive more complex cognitive descriptors.

Since the domain of the recorded material has a marked influence on the metadata useful and necessary to describe the audio recording, the following discussion is restricted to the recording of music. Almost all music, except monophonic works, is realized by the co-operation of musicians that perform in parallel different subsets of the music. But in many cases, grouping mechanisms in one's auditory perception transform even monophonic music in two or more perceived separate streams [1]. Therefore even pure music not accompanying a stage play can be understood and described as multimedia data and the scientific fundamentals discussed in the entry on multimedia metadata apply to music as well. Furthermore all music can exist in two different forms: a symbolic representation (the score), and a realization (the audio recording). Information like metrum or key that is hard to derive from the analysis of the recorded audio is readily available from the score. In addition a score is accessible for music theoretical analysis that leads to further metadata [8]. Whenever one of the two, score or realization, is missing, it can, at least in principle, be derived from the other. The realizations generated by performing a score vary to a considerable degree. First of all most scores do not give an absolute reference point for the frequency. To map the note A4 (the A above the middle C) to a frequency of 440 Hertz, now an ISO standard [ISO 16:1975 Acoustics – Standard tuning frequency (Standard musical pitch)], is a rather new convention established by an international conference in 1936. Throughout history a variety of reference frequencies for A4 have been used, varying from as low as 392 Hertz up to 466 Hertz. Prior to the acceptance of equal temperament as tuning standard in the second half of the eighteenth century, a variety of tuning standards were in use and rendering baroque music on historic instruments with a historic temperament leads to a quite different performance than the one produced by a modern orchestra with well-tempered tuning. Similar variability exists for the global tempo (at least prior to the use of the metronome ticks to specify absolute time values) and the dynamic. And of course each individual performer or orchestra has its personal style of phrasings and embellishments. Therefore one score gives rise to a variety of realizations and at least when one is interested in identifying music all metadata derived from the realization of the score should in the end lead to the same piece of music. But scores themselves are by no means static; throughout history they have been transformed to adapt them to different needs. Examples of such transformations are transpositions (shifts in pitch) to adapt the score to the ambitus of an instrument, or piano reductions where an orchestral piece is simplified in such a way that its "essence" can be rendered on a piano. Like with the variation in realizations metadata for the description of music should be able to cope with this variability. Both types of variation exist not only in classical western music but in popular music or non-western music like Indonesian gamelan or Indian ragas as well.

Music has a complex temporal organisation and a rich semantic structure that defines a natural segmentation for the audio stream. Like in images, in music many features are characteristic for segments, vary from segment to segment, and become meaningless when averaged over all segments. Musical structure is not arbitrary but conforms to a set of possible patterns: The sonata form [10] for example is one of the most influential structural patterns during the classical era of western music. The structure of the sonata form is built from three to five pieces: an optional introduction, an exposition, a middle part (*Durchführung*), a repeat, and an optional coda. Usually these five elements are not atomic but further structured and some segments of this substructure are derived from each other by transformations like e.g., transposition, inversion, reflection, or tempo changes. Besides being metadata in its own right, structural information linked with the audio material allows a natural navigation of the recorded performance. As outlined above, the approach to document structure is prescriptive. As in nineteenth century music theory it is assumed that there is an ideal architecture for a sonata form, and that a piece of music not conforming to these rules is in error. In the twentieth century this concept has come under considerable criticism [e.g., 11] and a descriptive approach has been advocated. As a consequence, each musical work is likely to have more than one semantic segmentation, depending on the analysis approaches chosen. To be of any use, both a controlled vocabulary describing the structures and an ontology describing the relationships among them are needed. But this is by no means specific to the structure of music. For many other audio metadata standardized controlled vocabularies and ontologies are still lacking. For example without amendment the Dublin Core [http://www.dublincore.org/] term creator, “An entity primarily responsible for making the resource” [http://purl.org/dc/terms/creator], attached to a piece of music makes it hard if not impossible to recognize the specific role of the creator. Felix Weingartner as creator could refer to his role as conductor (he conducted what is believed to be the first complete recording of Beethoven’s symphonies), or his role as composer (symphonies, string quartets, operas) or his role as editor (he edited the complete works of Berlioz).

Key Applications

Audio metadata are essential for any search for audio data. In addition music metadata help to reveal similarities in style or structure between different compositions or different realisations of a piece of music. There are many frameworks for the analysis and manipulation of music, two GPLed packages that allow easy experimentation with different metadata concepts for music are CLAM [http://www.clam.iua.upf.edu/] and RUBATO® [http://www.rubato.org/].

Cross-references

▶ Image Metadata
▶ Multimedia Metadata
▶ Video Metadata

Recommended Reading

1. Deutsch D. (ed.) The Psychology of Music, 2nd edn. Academic, San Diego, CA, 1999.
2. Dudley H.W. The vocoder. Bell Labs Rec., 18:122–126, 1939.
3. Handel S. Listening. MIT, Cambridge, MA, 1989.
4. IEEE P1599/D5.0. Draft recommended practice for definition of a commonly acceptable musical application using the XML Language. NY, 2008.
5. ISO/IEC DIS 10743. Standard music description language (SMDL). July, 1995.
6. ISO/IEC 10744:1997. Information technology – Hypermedia/Time-based structuring language (HyTime). Geneva, 1997.
7. ISO/IEC 15938–4:2002. Information technology – Multimedia content description interface – Part 4: Audio. Geneva, 2002.
8. Mazzola G. The Topos of Music. Birkhäuser, Basel, 2002.
9. Moore B. (ed.) Hearing. Academic, San Diego, CA, 1995.
10. Mauser, S. (ed.) Handbuch der musikalischen Gattungen, Laaber 1993 ff.
11. Rosen C. Sonata Forms, 2nd edn. Norton, NY, 1980.
12. Zaminer F. Geschichte der Musiktheorie, Darmstadt 1984 ff.

Audio Parsing

▶ Audio Segmentation

Audio Recognition

▶ Audio Classification

Audio Representation

Lie Lu[1], Alan Hanjalic[2]
[1]Microsoft Research Asia, Beijing, China
[2]Delft University of Technology, Delft,
The Netherlands

Synonyms

Audio feature extraction; Audio characterization

Definition

An *audio signal* is a signal that contains information in the audible frequency range. Audio representation refers to the extraction of audio signal properties, or *features*, that are representative of the audio signal composition (both in temporal and spectral domain) and audio signal behavior over time. *Feature extraction* is typically combined with *feature selection*, through which the best set of features for the intended operation on the audio signal is defined.

Historical Background

Audio feature extraction typically leads to a strongly reduced audio signal representation. Obtaining such representation can improve the efficiency of audio processing and benefit many applications based on such processing. For example, a compact representation of an audio signal in the form of a *fingerprint* can enable extremely fast search for a match between this signal and a large-scale audio database for the purpose of audio signal identification. Further, if audio features are carefully chosen, they can capture the information from the original data that is relevant to subsequent audio signal analysis and processing steps, while leaving out the redundant and irrelevant (noisy) information parts. This possibility to simultaneously improve the efficiency and robustness of audio signal analysis and processing indicates the importance of the *feature selection* step as the basis step in *audio content analysis*, and in particular in *audio segmentation* and *audio classification*.

Foundations

Audio features can be divided into *temporal* and *spectral* features that capture the temporal and spectral characteristics of an audio signal, respectively. In terms of the way the features are extracted, a division into *frame-level* and *window-level* features can be made. An audio *frame* is the elementary temporal segment of the signal, from which features are extracted. The length of an audio frame typically varies between 10 and 50 ms. Due to its short duration, a frame can be said to contain (close-to) stationary signal behavior. The window-level features are extracted from a longer audio segment, comprising a number of consecutive frames, and are typically marked by applying a sliding window to the signal. While most audio features are extracted at the frame level, window-level features are mainly derived from the frame-level features by investigating their variation along the frames within the window, e.g., the mean and standard deviation of frame-level features. This expansion of the frame-level feature consideration from an individual frame to a series of consecutive frames proved to be useful in many applications, which indicates the importance of window-level features.

Table 1 gives an overview of the typical features proposed in literature to perform various operations on audio, and in particular, the audio segmentation and classification. The features in the table are ranked according to the frequency of their usage in literature. Multiple names indicated per row of the table stand for one and the same feature and/or its variants. Also, the notation ***t***, ***s***, ***fl*** and ***wl*** is used to indicate whether a feature is temporal, spectral, frame-level or a window-level feature. The table is followed by detailed descriptions of a subset of the most prominent frame-level and window-level features. Finally, information is provided about the processes of feature normalization and selection to generate optimal feature sets (vectors) for audio representation.

Zero-Crossing Rate

Zero-crossing rate (*ZCR*) is defined as the relative number of times the audio signal crosses the zero-line within a frame. It can be computed using the following expression:

$$ZCR = \frac{1}{2(N-1)} \sum_{m=1}^{N-1} |\mathrm{sgn}[x(m+1)] - \mathrm{sgn}[x(m)]| \qquad (1)$$

Here, sgn[] is a sign function, *x(m)* is the discrete audio signal, *m* = *1...N*, and *N* is the frame length.

The *ZCR* is a computationally simple measure of the general frequency content of a signal, and as such it is particularly useful in characterizing audio signals in terms of the *voiced* and *unvoiced* sound categories. As speech signals are generally composed of alternating voiced and unvoiced sounds, which is not the case in music signals, the variation in the *ZCR* values is expected to be larger for speech signals than for music signals. Due to its discriminative power in separating speech, music and various audio effects, *ZCR* is often employed in audio content analysis algorithms. An illustration of its practical usage can be found in [7,8,10–12,15].

Short Time Energy

Short Time Energy (*STE*) is the total spectral power of a frame. It can be computed from the audio signal directly, as

Audio Representation. Table 1. An overview of audio features most frequently used in literature for the purpose of audio segmentation and classification

Features	Level	Temporal/spectral
Short time energy, root mean square (RMS), spectrum power, volume, loudness	*fl*	*t, s*
Zero crossing rate (ZCR)	*fl*	*t*
Mel-frequency cepstral coefficient (MFCC)	*fl*	*s*
Spectral centroid, brightness, frequency centroid	*fl*	*s*
Bandwidth	*fl*	*s*
Sub-band energy (distribution), sub-band power, band-energy ratio	*fl*	*s*
Short time fundamental frequency, pitch, harmonic frequency	*fl*	*s*
LPC-derived cepstral coefficients (LPCC)	*fl*	*s*
Linear predictive coding (LPC)	*fl*	*s*
Spectral rolloff	*fl*	*s*
Spectral peak	*fl*	*s*
Spectral moments	*fl*	*s*
Spectral flatness	*fl*	*s*
Harmonicity	*fl*	*s*
Harmonicity prominence	*fl*	*s*
Sub-band partial prominence	*fl*	*s*
Wavelet decomposition	*fl*	*s*
MPEG-7 audio features	*fl*	*s*
Spectrum flux	*wl*	*s*
Percentage of low-energy frames, low short-time energy ratio (LSTER), non-silence ratio	*wl*	*t, s*
High ZCR ratio (HZCRR)	*wl*	*t*
Noise frame ratio, noise or non-voice ratio	*wl*	*t*
4 Hz modulation energy	*wl*	*t, s*
Pulse metric	*wl*	*t*

$$STE = \frac{1}{N}\sum_{m=1}^{N} |x(m)|^2, \qquad (2)$$

or from its *Discrete Fourier Transform* (*DFT*) coefficients, as

$$STE = \sum_{k=0}^{K/2} |F(k)|^2. \qquad (3)$$

Here, $F(k)$ denotes the *DFT* coefficients, $|F(k)|^2$ is the signal power at the discrete frequency k, and K is the order of *DFT*. In [7,14], this energy is computed using the logarithmic expression, to get a measure in (or similar to) decibels.

Similar to *ZCR*, *STE* is also an effective feature for discriminating between speech and music signals. For example, there are more silence (or unvoiced) frames in speech than in music. As a result, the variation of *STE* in speech is in general much higher than in music. An illustration of the practical usage of this feature can be found in [7,10,12,14,15].

Sub-Band Energy Distribution

To further exploit the energy information based on the *STE* feature defined above, the *sub-band energy distribution* (*SBED*) can be computed. This distribution can be obtained by dividing the frequency spectrum into sub-bands, and by computing for each sub-band j the ratio D_j between the energy contained in that sub-band and the total spectral power of the frame:

$$D_j = \frac{1}{STE}\sum_{L_j}^{H_j} |F(k)|^2 \qquad (4)$$

Here, L_j and H_j are the lower and upper bound of sub-band j respectively. The sub-band division can be done in various ways, such as in octave-scale [7] or in mel-scale [1].

Since the spectrum characteristics are rather different for sounds produced by different sources (e.g., human voice, music, environmental noise) the *SBED* feature has often been used for general audio classification [5,10], and, in particular, for discriminating between different sound effects [1,14].

Brightness and Bandwidth

Brightness and *bandwidth* are related to the first- and second-order statistics of the spectrum, respectively. The brightness is the centroid of the spectrum of a frame, and can be defined as:

$$w_c = \frac{\sum_{k=0}^{K/2} k|F(k)|^2}{\sum_{k=0}^{K/2} |F(k)|^2} \tag{5}$$

Bandwidth is the square root of the power-weighted average of the squared difference between the spectral components and the centroid:

$$B = \sqrt{\frac{\sum_{k=0}^{K/2} (k - w_c)^2 |F(k)|^2}{\sum_{k=0}^{K/2} |F(k)|^2}} \tag{6}$$

Brightness and Bandwidth characterize the shape of the spectrum, and roughly indicate the timbre quality of a sound. From this perspective, brightness and bandwidth can provide useful information for audio classification processes [11,14].

Mel-Frequency Cepstral Coefficient (MFCC)

The set of *Mel-Frequency Cepstral Coefficients* [10] is a cepstral representation of the audio signal obtained based on the mel-scaled spectrum. The log spectral amplitudes are first mapped onto the perceptual, logarithmic *mel-scale*, using a triangular band-pass filter bank. Then, the mel-scaled spectrum is transformed into MFCC using the Discrete Cosine Transform (*DCT*).

$$c_x = \sqrt{\frac{2}{K}} \sum_{k=1}^{K} (\log S_k) \cos[n(k - 0.5)\pi/K] \qquad n = 1, 2, \ldots, L \tag{7}$$

Here, c_n is the n-th MFCC, K is the number of band-pass filters, S_k is the mel-scaled spectrum after passing the k-th triangular band-pass filter, and L is the order of the cepstrum.

MFCC is commonly used in speech recognition and speaker recognition systems. However, *MFCC* also proved to be useful in discriminating between speech and other sound classes, such as music, which explains its wide usage in the audio analysis and processing literature [3,6,12].

Sub-Band Partial Prominence and Harmonicity Prominence

The *Sub-Band Partial Prominence* (*SBPP*) is used to measure whether there are salient frequency components (i.e., *partials*) in a sub-band. In other words, the *SBPP* estimates the existence of prominent partials in sub-bands [1]. It is computed by accumulating the variation between adjacent frequency bins in each sub-band, that is

$$S_p(i) = \frac{1}{H_i - L_i} \sum_{j=L_i}^{H_i - 1} \left| \hat{F}(k+1) - \hat{F}(k) \right| \tag{8}$$

Here, L_i and H_i are the lower and upper boundaries of the ith sub-band respectively, and the value of $S_p(i)$ indicates the corresponding prominence of salient partial components. The *SBPP* value $S_p(i)$ for sub-bands containing salient components is expected to be large. In order to reduce the impact induced by the energy variation over time, the original *DFT* spectral coefficient vector $\boldsymbol{F}$ is first converted to the decibel scale and constrained to the unit L_2-norm [2] to yield the new spectral coefficient vector used on (8):

$$\overset{\wedge}{\boldsymbol{F}} = \frac{10 \log_{10}(\boldsymbol{F})}{\left\| 10 \log_{10}(\boldsymbol{F}) \right\|} \tag{9}$$

If now the property of an ideally harmonic sound (with one dominant fundamental frequency f_0) is considered, its spectral energy is highly concentrated and precisely located at those predicted harmonic positions which are the multiples of the fundamental frequency f_0. To detect this situation, the following three factors could be measured: (i) the energy ratio between the detected harmonics and the whole spectrum; (ii) the deviation between the detected harmonics and predicted positions; and (iii) the concentration degree of the harmonic energy. Based on the above, the *Harmonicity Prominence* (*HP*) was defined in [14] to estimate the harmonic degree of a sound. The *HP* measure takes into account the above three factors and can be defined as

$$H_p = \frac{\sum_{n=1}^{N} E^{(n)} \left(1 - |B_r^{(n)} - f_n| \middle/ 0.5 f_0\right)\left(1 - B_w^{(n)} \middle/ B\right)}{E} \qquad (10)$$

Here, $E^{(n)}$ is the energy of the detected nth harmonic contour in the range of $[f_n - f_0/2, f_n + f_0/2]$ and the denominator E is the total spectral energy. The ratio between $E^{(n)}$ and E stands for the first of the three factors identified above. Further, f_n is the nth predicted harmonic position and is defined as

$$f_n = n f_0 \sqrt{1 + \beta(n^2 - 1)} \qquad (11)$$

where β is the *inharmonicity modification factor*, and $B_y^{(n)}$and $B_w^{(n)}$ are the brightness and bandwidth of the nth harmonic contour, respectively. The brightness $B_y^{(n)}$ is used instead of the detected harmonic peak in order to estimate a more accurate frequency center. The bandwidth $B_y^{(n)}$ describes the concentration degree of the nth harmonic. It is normalized by a constant B, which is defined as the bandwidth of an instance where the energy is uniformly distributed in the search range. Thus, the components $\left(1 - |B_y^{(n)} - f_n|/0.5 f_0\right)$ and $\left(1 - B_w^{(n)}/B\right)$ in the numerator of (10) represent the second and the third factor defined above.

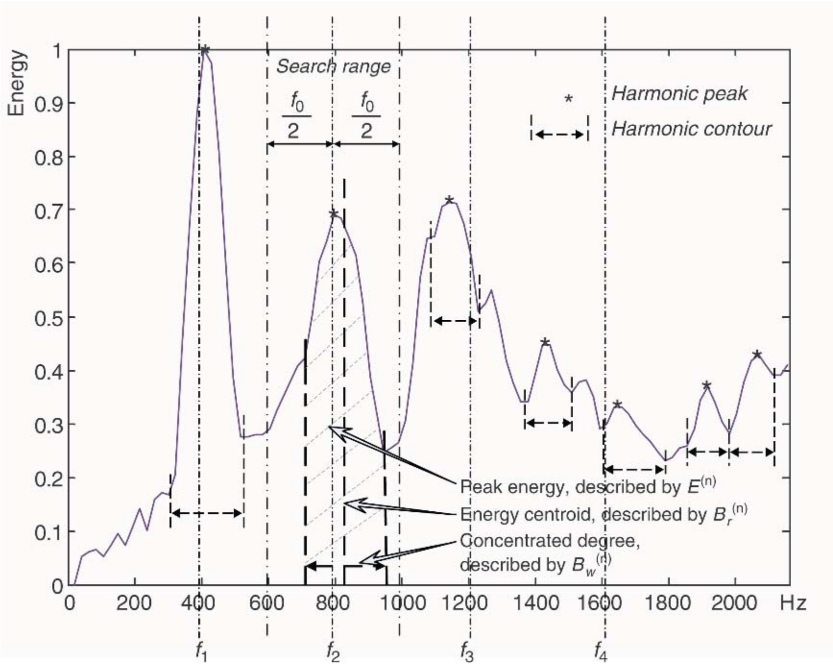
An illustration of the definition of *harmonicity prominence* is shown in Fig. 1. Detailed explanation of the computation and usage of this feature can be found in [1]. The harmonic audio analysis using the *SBPP* and *HP* features enables sophisticated audio classification, like for instance, the discrimination between *cheer* and *laughter*. This is possible because *laughter*, as opposed to *cheer*, usually contains prominent harmonic partials.

High ZCR Ratio

High ZCR Ratio (*HZCRR*) [6] is defined as the fraction of frames in the analysis window, whose *ZCR* values are 50% higher than the average *ZCR* computed in the window, that is

$$HZCRR = \frac{1}{2N} \sum_{n=0}^{N-1} [\mathrm{sgn}(ZCR(n) - 1.5 avZCR) + 1] \qquad (12)$$

Here, n is the frame index, $ZCR(n)$ is the zero-crossing rate at the n-th frame, N is the total number of frames,



Audio Representation. Figure 1. Definition of *harmonicity prominence*. The horizontal axis represents the frequency, and the vertical axis denotes the energy. The harmonic contour is the segment between the adjacent valleys separating the harmonic peaks. Based on the harmonic contour, three factors, that is, the peak energy, energy centroid (*brightness*) and degree of concentration (*bandwidth*), are computed to estimate the *harmonicity prominence*, as illustrated at the second harmonic in this example.

avZCR is the average *ZCR* in the analysis window, and sgn[] is a sign function. The variation of *HZCRR* is expected to be higher in speech signals than in music. Fig. 2(I) shows the probability distribution curves of *HZCRR* computed for a large number of speech and music signals. Using the cross-point of two displayed *HZCRR* curves as the threshold to discriminate speech and music leads to the classification error of 19.36%, as shown in [6].

Low Short-Time Energy Ratio

As an analogy for selecting *HZCRR* to model the variations of the ZCR within the analysis window, the *low short-time energy ratio* (*LSTER*) [6,11] can be defined to model the variation of the *short-time energy* (*STE*) in this window. *LSTER* is defined as the fraction of the frames within the analysis window, whose *STE* values are less than a half of the average *STE* measured in the window, that is,

$$LSTER = \frac{1}{2N}\sum_{n=0}^{N-1}[\mathrm{sgn}(0.5avSTE - STE(n)) + 1] \tag{13}$$

Here, N is the total number of frames in the analysis window, $STE(n)$ is the short time energy at the n-th frame, and *avSTE* is the average *STE* in the window. The *LSTER* measure of speech is expected to be much higher than that of music. This can be seen clearly from the probability distribution curves of *LSTER* obtained for a large number of speech and music signals, as illustrated in the Fig. 2(II). Using the cross-point of two displayed *LSTER* curves as the threshold to discriminate speech and music leads to the classification error of 8.27%, as presented in [6].

Spectrum Flux

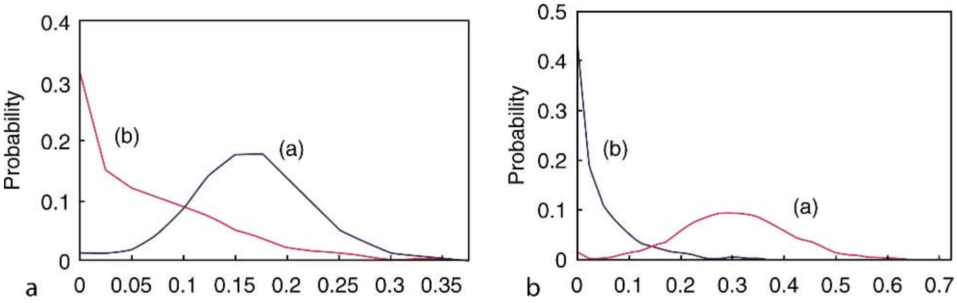
Spectrum Flux (*SF*) is defined as the average variation of the spectrum between adjacent two frames in the analysis window, that is

$$SF = \frac{1}{(N-1)(K-1)}\sum_{n=1}^{N-1}\sum_{k=1}^{K-1}[\log(A(n,k)+\delta) - \log(A(n-1,k)+\delta]^2 \tag{14}$$

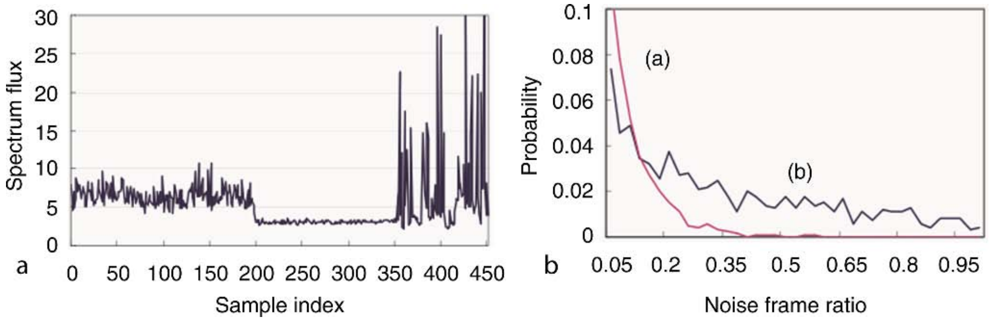
Here, $A(n, k)$ is the absolute value of the k-th *DFT* coefficient of the n-th frame, K is the order of DFT, δ is a very small value used to avoid computation overflow, and N is the number of frames in the analysis window. The *SF* of speech is expected to be larger than that of music. It was also found that the spectrum flux of environmental sounds is generally very high, and that it changes more dynamically than for speech and music [6]. To illustrate this, Fig. 3(I) shows the *SF* computed for an audio segment consisting of speech (0–200 s), music (201–350 s) and environmental sounds (351–450 s). Usage examples of this feature are provided in [8,6,11].

Noise Frame Ratio

Noise frame ratio (*NFR*) is defined as the ratio of noise frames in a given audio clip. A frame is considered as a noise frame if the maximum local peak of its normalized correlation function is lower than a pre-set threshold. The *NFR* is usually used to discriminate environmental sound from music and speech, and to detect noisy sounds. For example, the *NFR* value of a noise-like environmental sound is higher than that of music, because it contains many more noise frames. As can be observed in Fig. 3(II), considering higher NFR values can prove quite discriminative



Audio Representation. Figure 2. An illustration of probability distribution curves of (I) *HZCRR* (a) speech and (b) music, and (II) *LSTER* (a) speech and (b) music.



Audio Representation. Figure 3. (I) The spectrum flux curve of speech (0-200 s), music (201-350 s) and environmental sounds (351-450 s); (II) The probability distribution curves of *NFR*: (a) music and (b) environmental sound.

in separating these two types of audio. An illustration of the usage of this feature can be found in [5,6].

Feature Vector Generation

After they are extracted, features need to be combined together to form a complete audio representation and to provide input into audio analysis and processing steps. Since the values and dynamics of these features may vary considerably over the feature set, simply concatenating them all into a long feature vector is not likely to lead to good results. Therefore, a *normalization* needs to be performed on the features first to equalize their scales. The normalization is usually performed using the mean and standard deviation per feature, namely as $x_i^{'} = (x_i - \mu_i)/\sigma_i$, where x_i is the *i*-th feature, and where the corresponding mean μ_i and standard deviation σ_i can be obtained from the analyzed data set. Next to the normalization, *feature selection* is usually performed to improve the effectiveness of the feature vector while minimizing its dimension. While feature selection can be realized in many ways [4], a typical approach involves the *principle component analysis* (*PCA*) [13]. Technically, *PCA* is an orthogonal linear transformation that transforms the data to a new coordinate system to reveal the main characteristics (*principal components*) of the data that contribute most to the variance in data, and therefore best explain the data. The principal components can be obtained by performing a covariance analysis or *singular value decomposition* (*SVD*) [13]. If X' is a set of *N*-dimensional normalized feature vectors from *M* segments (usually $M >> N$), then X' can be written as an $M \times N$ matrix, where each row corresponds to a feature vector of one audio segment. By applying the *SVD*, the matrix X' can be written as

$$X' = USV^T \tag{15}$$

In terms of *SVD*, *V* and *U* are, respectively, an $N \times N$ and $M \times N$ matrix containing the right and left singular vectors, while the diagonal $N \times N$ matrix $S = \text{diag}\{\lambda_1,\ldots,\lambda_N,\}$ contains singular values, with $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$. In terms of *PCA*, singular vectors (columns) of the matrix *V* can be seen as principal components of X', each of which has its corresponding singular value. The larger this singular value is, the more principal (or more important) the component is. Assuming that V_m is a matrix keeping the first *m* principal components (by keeping the first *m* columns from *V*), the original feature set X' can be replaced by a reduced, PCA-transformed feature set

$$X'' = X'V_m \tag{16}$$

which only preserves those features that are relevant to subsequent audio signal analysis and processing steps, while leaving out the redundant and irrelevant (noisy) features.

Key Applications

Audio representation provides fundamentals for audio classification and audio segmentation.

Cross-references

- ▶ Audio Classification
- ▶ Audio Content Analysis
- ▶ Audio Segmentation

Recommended Reading

1. Cai R., Lu L., Hanjalic A., Zhang H.-J., and Cai L.-H. A flexible framework for key audio effects detection and auditory context

inference. IEEE Trans. Audio, Speech Lang. Process., 14 (3):1026–1039, 2006.

2. Casey M.A. MPEG-7 sound-recognition tools. IEEE Trans. Circuits and Syst. for Video Tech., 11(6):737–747, 1997.
3. Foote J. Content-based retrieval of music and audio. In Proc. SPIE Multimedia Storage and Archiving Systems II. 1997, pp. 138–147.
4. Guyon I. and Elisseeff A. An introduction to variable and feature selection. J. Mach. Learn. Res., 3:1157–1182, 2003.
5. Liu Z., Wang Y., and Chen T. Audio feature extraction and analysis for scene segmentation and classification. J. VLSI Signal Process. Sys., 20(1–2):61–79, 1998.
6. Lu L., Zhang H.-J., and Jiang H. Content analysis for audio classification and segmentation. IEEE Trans. Speech Audio Process., 10(7):504–516, 2002.
7. Lu L., Zhang H.-J., and Li S. Content-based audio classification and segmentation by using support vector machines. ACM Multimedia Sys. J., 8(6):482–492, March, 2003.
8. Peltonen V., Tuomi J., Klapuri A.P., Huopaniemi J., and Sorsa T. Computational auditory scene recognition. In Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing, Vol. 2, 2002, pp. 1941–1944.
9. Rabiner L. and Juang B.H. Fundamentals of Speech Recognition, Prentice-Hall, Englewood Cliffs, New Jersey, 1993.
10. Saunders J. Real-time discrimination of broadcast speech/music. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Vol. 2, 1996, pp. 993–996.
11. Scheirer E. and Slaney M. Construction and evaluation of a robust multifeature music/speech discriminator. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Vol. 2, 1997, pp. 1331–1334.
12. Tzanetakis G. and Cook P. Marsyas: A framework for audio analysis. Organized Sound, 4(3):2000.
13. Wall M.E., Rechtsteiner A., and Rocha L.M. Singular value decomposition and principal component analysis. In A Practical Approach to Microarray Data Analysis, D.P. Berrar, W. Dubitzky, M. Granzow (eds.). Kluwer, Norwell, MA (2003). pp. 91–109, LANL LA-UR-02-4001.
14. Wold E., Blum T. and Wheaton J. Content-based classification, search and retrieval of audio. IEEE Multimedia, 3(3):27–36, 1996.
15. Zhang T. and Kuo C.-C.J. Video content parsing based on combined audio and visual information. In Proc. SPIE: Multimedia Storage and Archiving Systems, IV, 1999, pp. 78–89.

Audio Segmentation

Lie Lu[1], Alan Hanjalic[2]
[1]Microsoft Research Asia, Beijing, China
[2]Delft University of Technology, Delft,
The Netherlands

Synonyms

Audio parsing; Auditory scene detection

Definition

Audio segmentation refers to the class of theories and algorithms designed to automatically reveal semantically meaningful temporal segments in an audio signal, also referred to as *auditory scenes* [7]. These scenes can be seen as equivalents of paragraphs in text, and can serve as input into audio categorization processes, either supervised (audio classification) or unsupervised (audio clustering). Through these processes, semantically similar auditory scenes can be grouped together and/or labeled using semantic indexes to provide multi-level, non-linear content-based access to large audio documents and collections.

Historical Background

Automatic detection of *auditory scenes* is an important step in enabling high-level semantic inference from general audio signals, and can benefit various content-based applications involving both audio and multimodal (multimedia) data sets. Traditional approaches to audio segmentation usually rely on a direct analysis of low-level audio features, that is, the targeted audio segments were often defined to coincide with a consistent low-level feature behavior [2,10,11]. This idea served as a basis for numerous approaches for audio segmentation. For example, in [10], a method for scene segmentation was presented that uses low-level features, such as cepstral and cochlear decomposition, combined with the listener model and various time scales. Motivated by the known limitations of traditional low-level feature based approaches, an approach was proposed in [7] to discover auditory scenes based on an analysis of *audio elements*, which can be seen as equivalents to the words in a text document. In this approach that draws an analogy to text document analysis, an audio track is described as a sequence of audio elements, and auditory scenes are segmented based on the semantic affinity among these audio elements and their co-occurrence.

Foundations

Traditional approaches to audio parsing relying directly on audio features have proved effective for many applications, and in particular for those where knowledge on the basic audio modalities (speech, music, and noise) is critical. However, for other applications, like those where higher-level content categories, e.g., semantic concepts, become interesting, the low-level feature based approaches have shown deficiencies due to their incapability of capturing the entire content diversity of a typical semantic concept. The audio segments obtained by typical feature-based approaches are short and of no higher semantic meaning, if compared to the true semantic segments, like, for instance, *logical story units* targeted by the algorithms of high-level video parsing [3], or the paragraphs in a text document.

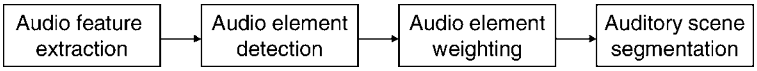
To come closer to the level of auditory scenes, a promising alternative is to design and employ suitable mid-level audio content representations. Figure 1 shows the framework for audio segmentation [6] where the input audio is first decomposed into various *audio elements* such as speech, music, various audio effects and any combination of these. Then, *audio element weighting* is performed to reveal the importance of an audio element to represent an audio document or any of its parts. The audio elements with highest weights can be adopted as the *key audio elements*, being the most characteristic for the semantics of the analyzed audio data [1,4,5,8,9,12–14]. Finally, auditory scenes can be characterized and detected based on the audio elements they contain, just as the paragraphs of a text document can be characterized and detected using a vector of words and their weights. As shown in [7], introducing the mid-level audio content representation in the form of audio elements enables splitting the semantics inference process into two steps, which leads to more robustness compared to inferring the semantics from low-level features directly.

The usefulness of audio elements for audio content analysis was already recognized, e.g., in [13], where the audio elements such as *applause*, *ball-hit*, and *whistling*, are extracted and used to detect the highlights in sports videos. However, this and similar methods usually adopted supervised data analysis and classification methods. There, the scene categories and the corresponding audio elements need to be predefined, which is usually difficult to do for general audio documents. Further, the effectiveness of supervised approaches relies heavily on the quality and quantity of the training data. This makes such approaches difficult to generalize. In view of this, a number of unsupervised approaches were proposed, including the approach to audio element discovery [1], and an approach to auditory scene segmentation [7]. The latter exploits the co-occurrence phenomena among audio elements to realize the segmentation scheme from Fig. 1. This is based on the rationale that, in general, some audio elements will rarely occur together in the same semantic context. On the other hand, the auditory scenes with similar semantics usually contain similar sets of typical audio elements. For example, many action scenes may contain *gunshots* and *explosions*, while a typical scene in a situation comedy may be characterized by a combination of *applause*, *laughter*, *speech*, and *light music*.

Audio Elements Detection and Weighting

As proposed in [1], an iterative *spectral clustering* method can be used to decompose an audio document into audio elements. Spectral clustering can be seen as an optimization problem of grouping similar data based on eigenvectors of a (possibly normalized) affinity matrix. Ng et al. [9] proposed a method to use k eigenvectors simultaneously to partition the data into k clusters, and successfully applied this to a number of complicated clustering problems. To further improve the robustness of the clustering process, the self-tuning strategy [14] can be adopted to set the context-based scaling factors for different data densities. This removes the need for the assumption that each cluster in the input data has a similar distribution density in the feature space, which is inherent in the standard spectral clustering algorithm, but usually not satisfied in complex audio data. Using this clustering method, short audio segments (e.g., one second in length [1]) can be grouped into natural semantic clusters that can then be adopted as audio elements.

In the next step, the obtained audio elements are assigned the importance weights to indicate their prominence in characterizing the content of audio data. Here, two cases can be considered. The first case assumes that only one audio document is available for weight computation. Then, a number of heuristic importance indicators, including *Element Frequency, Element Duration*, and *Average Element Length*, are



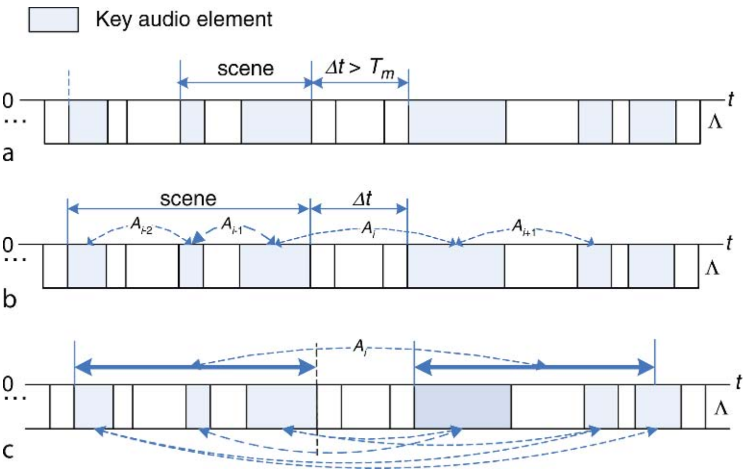
Audio Segmentation. Figure 1. The framework for audio segmentation based on audio elements [6].

proposed in [1] to compute the weight. The second case assumes that multiple audio documents are available to learn the weights. In this case, inspired by the effectiveness of *term frequency* (*TF*) and *inverse document frequency* (*IDF*) used for word weighting in text document analysis, the equivalents of these measures can be defined and employed for the case of audio segmentation. As described in [1,4,5,8,9,12–14], four factors, including *expected term frequency* (*ETF*), *expected inverse document frequency* (*EIDF*), *expected term duration* (*ETD*), and *expected inverse document duration* (*EIDD*), can be defined and combined together to give the importance weight of each audio element. These factors take into account the discriminative power of the occurrence frequency and the duration of a particular audio element to characterize the semantics of an audio document.

Auditory Scene Segmentation

In view of the way it is defined, an auditory scene may consist of multiple, concatenated and semantically related audio elements. An example of such an auditory scene is a *humor* scene consisting of several interleaved segments of *speech*, *laughter*, *cheer*, and possibly also some *light music*. In [6], a simple segmentation scheme was presented that employs crisply defined key audio elements. As shown in Figure 2a, two adjacent key audio elements are assumed to be in the same auditory scene if the time interval between them is sufficiently short. Then the scene boundaries are aligned to the key audio elements, while the background audio elements between two scenes are discarded. Clearly, the algorithm is quite naive and does not fully exploit the relationship between audio elements and auditory scenes. To improve the detection performance, the notion of *semantic affinity* between two contiguous key audio elements was introduced in [1]. This affinity takes into account both the co-occurrence statistics of these key audio elements and the time interval between them, and was employed in [1] to locate the auditory scene boundaries. As shown in Figure 2b, auditory scene boundaries are found between two key audio elements if their semantic affinity is low.

The performance of the segmentation methods discussed above strongly depends on the definition of a key audio element and the reliability of its detection. Crisply defining key audio elements and detecting them in composite audio documents may be rather difficult due to multiple superimposed audio modalities. Therefore, a more reliable solution would be to work with all audio elements instead, and rely on their importance weights. This idea also follows the analogy to the classical text [4] and video scene segmentation approaches [3,5]. As illustrated in Figure 2c, an approach in this direction would decide about the presence of a scene boundary at the observed time stamp



Audio Segmentation. Figure 2. An illustration of previous approaches to auditory scene segmentation, where a vertical line indicates a detected scene boundary: (a) using time interval between key audio elements; (b) using semantic affinity between neighboring key audio elements; and (c) investigating the relationship of (key) audio elements on a large temporal scale.

based on an investigation of the semantic affinity between audio elements taken from a broader range and surrounding this time stamp.

The possibilities for realizing the audio segmentation idea from Figure 2c are now illustrated on the example of the method proposed in [7]. Here, just like in text document analysis, the measure for semantic affinity is not based on the feature-based similarity between two audio segments, but on their joint ability to represent a semantically coherent piece of audio. With this in mind, the definition of semantic affinity in [7] is based on the following intuitive assumptions:

- Affinity between two audio segments is high if the corresponding audio elements usually occur together.
- The larger the time interval between two audio segments, the lower their affinity.
- The higher the importance weights of the corresponding audio elements, the more important is the role these elements will play in the auditory scene segmentation process, and therefore the more significant the computed semantic affinity value will be.

Figure 3 shows an example audio element sequence, where each temporal block belongs to an audio element and where different classes of audio elements are represented by different colors/grayscales. The semantic affinity between the segments s_i and s_j can now be computed as a function consisting of three components, each of which reflects one of the assumptions stated above. The following measure is proposed in [1]:

$$A\left(S_i, S_j\right) = Co\left(e_i, e_j\right) e^{-T\left(S_i, S_j\right)/T_m} p_{e_i} p_{e_j} \qquad (1)$$

Here, the notation e_i and e_j is used to indicate the audio element identities of the segments s_i and s_j, that is, to describe their content (e.g., speech, music, noise, or any combination of these). P_{ei} and P_{ej} are the importance weights of audio elements e_i and e_j, while $T(s_i,s_j)$ is the time interval between the audio segments s_i and s_j. Further, T_m is a scaling factor which can be set to 16 s, following the discussions on human memory limit [3]. The exponential expression in (1) is inspired by the content coherence computation formula introduced in [12]. Further, $Co(e_i, e_j)$ stands for the co-occurrence between two audio elements, e_i and e_j, in the entire observed audio document, and measures their joint ability to characterize a semantically coherent auditory scene.

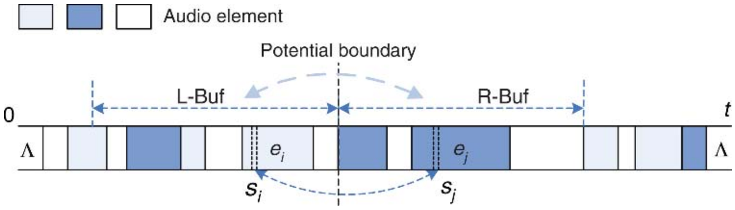
To estimate the co-occurrence between two audio elements, one can rely on the average time interval between two audio elements. The shorter the time interval, the higher the co-occurrence probability is. The procedure for estimating the value $Co(e_i, e_j)$ can then be summarized in the following three steps [1]:

1. First, compute D_{ij}, the average time interval between audio elements e_i and e_j, which is obtained by investigating the co-occurrences of the observed audio elements in the audio signal. For each segment belonging to audio element e_i, the nearest segment corresponding to e_j is located, and then D_{ij} is obtained as the average temporal distance between e_i and e_j.
2. D_{ji} is computed as an analogy to D_{ij}. One should note that D_{ij} is not always equal to D_{ji}.
3. The co-occurrence value can now be found as

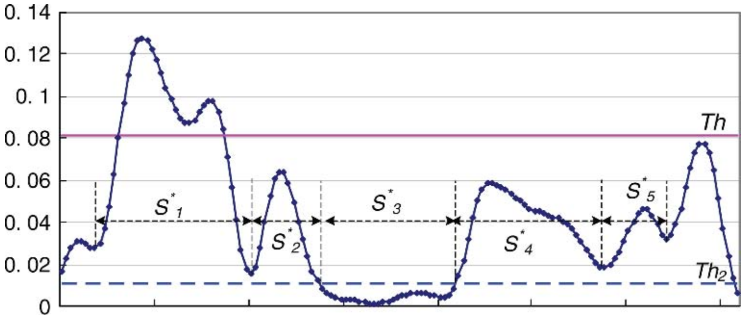
$$Co\left(e_i, e_j\right) = e^{-\frac{D_{ij} + D_{ji}}{2\mu_D}} \qquad (2)$$

where μ_D is the average of all D_{ij} and D_{ji} values. The choice for an exponential formula in (2) is made to keep the influence of audio element co-occurrence on the overall semantic affinity comparable with the influence of the time interval between the audio segments (1).

Based on the semantic affinity (1), the confidence of being within an auditory scene at the time stamp t can now be computed by averaging the affinity values



Audio Segmentation. Figure 3. An illustration of an approach to audio segmentation [7], where s_i and s_j are two audio segments, and e_i and e_j are their corresponding audio element identities.



Audio Segmentation. Figure 4. An example of the smoothed confidence curve and the auditory scene segmentation scheme, where S_1^*~S_5^* are five obtained auditory scenes and *Th* and Th_2 are two thresholds.

obtained for all pairs of segments s_i and s_j surrounding the *t*, that is,

$$C(t) = \frac{1}{N_l N_r} \sum_{i=1}^{N_l} \sum_{j=1}^{N_r} A(S_i, S_j)$$
$$= \frac{1}{N_l N_r} \sum_{i=1}^{N_l} \sum_{j=1}^{N_r} Co(e_1, e_j)\, e^{-T(S_i, S_j)/T_m} P_{e_i} P_{e_j} \qquad (3)$$

where N_l and N_r are the numbers of audio segments considered left and right from the potential boundary (as captured by the intervals *L-Buf* and *R-Buf* in Figure 3). Using this expression, a confidence curve can be obtained over the timeslots of potential boundaries, as illustrated in Figure 4. The boundaries of auditory scenes can now be detected simply by searching for local minima of the curve. In the approach from [1], the curve is first smoothed by using a median filter and then the auditory scene boundaries are found at places at which the following criteria are fulfilled:

$$C(t) < C(t+1);\;\; C(t) < C(t-1);\;\; C(t) < Th \quad (4)$$

Here, the first two conditions secure a local valley, while the last condition prevents high valleys from being detected. The threshold *Th* is set experimentally as $\mu_a + \sigma_a$, where μ_a and σ_a are the mean and standard deviation of the curve, respectively.

The obtained confidence curve is likely to contain long sequences of low confidence values, as shown by the segment S^*_3 in Figure 4. These sequences typically consist of the background audio elements which are weakly related to each other and also have low importance weights. Since it is not reasonable to divide such a sequence into smaller segments, or to merge them into neighboring auditory scenes, one could choose to isolate these sequences by including all consecutive audio segments with low affinity values into a separate auditory scene. Detecting such scenes is an analogy to detecting pauses in speech. Inspired by this, the corresponding threshold (Th_2 in Figure 4) can be set by using an approach similar to background noise level detection in speech analysis [12].

Key Applications

Audio segmentation is typically applied for content-based search and retrieval in and management of large-scale audio collections (databases).

Cross-references

► Audio Classification
► Audio Representation
► Video Content Structure

Recommended Reading

1. Cai R., Lu L., and Hanjalic A. Unsupervised content discovery in composite audio. In Proc. 13th ACM Int. Conf. on Multimedia, 2005, pp. 628–637.
2. Foote J. Automatic audio segmentation using a measure of audio novelty. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2000, pp. 452–455.
3. Hanjalic A., Lagendijk R.L., and Biemond J. Automated high-level movie segmentation for advanced video-retrieval systems. IEEE Trans. Circuits Syst. Video Technol., 9(4):580–588, 1999.
4. Kozima H. Text segmentation based on similarity between words. In Proc. 31st Annual Meeting on Association for Computational Linguistics, 1993, pp. 286–288.
5. Kender J.R. and Yeo B.-L. Video scene segmentation via continuous video coherence. In Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition, 1998, pp. 367–373.
6. Lu L., Cai R., and Hanjalic A. Towards a unified framework for content-based audio analysis. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, 2005, pp. 1069–1072.

7. Lu L., Cai R., and Hanjalic A. Audio elements based auditory scene segmentation. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, 2006, pp. 17–20.
8. Lu L. and Hanjalic A. Towards optimal audio keywords detection for audio content analysis and discovery. In Proc. 14th ACM Int. Conf. on Multimedia, 2006, pp. 825–834.
9. Ng A.Y., Jordan M.I., and Weis Y. On spectral clustering: analysis and an algorithm. In Proc. Advances in Neural Information Processing Systems, 2001, pp. 849–856.
10. Sundaram H. and Chang S.-F. Audio scene segmentation using multiple features, models and timescales. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, 2000, pp. 2441–2444.
11. Tzanetakis G. and Cook P. Multifeature audio segmentation for browsing and annotation. In Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, 1999, pp. 103–106.
12. Wang D., Lu L., and Zhang H.-J. Speech segmentation without speech recognition. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, 2003, pp. 468–471.
13. Xu M., Maddage N., Xu C.-S., Kankanhalli M., and Tian Q. Creating audio keywords for event detection in soccer video. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2003, pp. 281–284.
14. Zelnik-Manor L. and Perona P. Self-tuning spectral clustering. In Proc. Advances in Neural Information Processing Systems, 2004, pp. 1601–1608.

Audit Trail

► Logging/Recovery Subsystem

Auditing and Forensic Analysis

Brian Levine, Gerome Miklau
University of Massachusetts, Amherst, MA, USA

Synonyms

Accountability; Monitoring

Definition

The goal of *database auditing* is to retain a secure record of database operations that can be used to verify compliance with desired security policies, to trace policy violations, or to detect anomalous patterns of access. An audit log can contain the authorization ID and time stamp of read and write operations in the database, as well as a record of server connections, login attempts and authorization changes. Government and institutional regulations for the management of sensitive information often require auditing of data disclosure and data modification.

Database forensics is the analysis of the state of a database system to validate hypotheses about past events that are relevant to an alleged crime or violation of policy. Evidence supporting a forensic analysis may be found in an audit log (if available) but may also be recovered from any other component of a database system including table storage, the transaction log, temporary caches, or backup media. A challenge of working with forensically recovered evidence is that it is typically provides only a partial record of an event, possibly based on remnants of deleted information or on inference from incomplete information. On the other hand, it can be difficult to completely eradicate digital evidence from databases, which may be critical for preventing disclosure and complying with policy mandating limited data retention.

While auditing is focused on the preservation and analysis of specific data as required by law or internal corporate policy, digital forensics is broader in scope, potentially relating to any criminal or civil proceeding. Auditors and forensic analysts share some common goals, however the auditor usually relies on information retained intentionally by the system. The forensic analyst is more likely to rely on unintended remnants and inference about past events.

Historical Background

Auditing has been a common practice in settings where sensitive data is managed by computer systems, such as financial and military applications. Auditing has grown in importance as institutions are increasingly required to prove compliance with privacy regulations, or are mandated to discover and publicly respond to exploited vulnerabilities in their systems. Database forensics is an emerging subfield of digital forensics. Computer crime and investigations began receiving attention the late 1970s, but the realization that digital evidence is relevant to a spectrum of crimes has occurred within the last decade. Databases are common components of operating systems, web browers, and email programs and are an important focus of digital investigations.

Foundations

Database Auditing

The auditing component of a database system must support the collection, storage, and protection of sufficient historical data to enable desired auditing inquiries. Typical auditing queries might include the following:

- Display the query expression for all operations which modified more than ten rows.
- List the authorization IDs of users or client programs who performed SELECT queries on the Patients table between 10 P.M. and 5 A.M. last week.
- List any records in the Employee table whose salary field has been modified more than twice in the past 12 months.

In misuse detection or intrusion detection, the audit log may be used to assess more complex behavior such as: *Is today's workload of update operations similar to "normal" patterns of database usage?*

To support such inquiries, an audit log records the client programs and users who are executing operations, the data objects modified or disclosed, and the context of those operations. For each operation performed, an audit log could contain the SQL query string along with contextual information such as time of day, authorization ID, and network connection information. For update and deletion operations, the audit log may contain the removed values, and the previous values of modified data.

Analyzing database disclosure resulting from a sequence of SELECT queries can be more complex than analyzing the history of database modifications (inserts, updates, and deletes). Auditing disclosure has been the subject of intense research [1,2,4], but faces subtle challenges because a user may be able to infer information not directly released through an executed query. For example, a user's access to an individual database record can be hidden in a sequence of aggregate queries.

In establishing auditing policies, the credibility and completeness of the audit log must be carefully considered to ensure that all relevant events in the system are preserved. For example, the effects of an aborted transaction will be removed from the system for database consistency and atomicity. But a record of aborted transactions, and the reason for abort, could be important to an audit analysis [4].

Naturally, data that will never be relevant to the audit queries under consideration need not be recorded. Further, the retention of recorded audit data must be carefully determined and enforced. The log should be available for appropriate audit inquiries when they arise, but it also contains highly sensitive information and should be destroyed once the period of legitimate inquiry has passed.

Systems Issues and Performance Considerations To support auditing, database systems must efficiently collect required data and permit analysis. Modern commercial databases contain a range of native auditing features which usually include more than one type of log. Common features include a system log to record all connections to the database, and an query log to record each query expression submitted to the database. An auditing policy can be chosen to specify the level of detail that should be logged (e.g., all accesses to relations, or individual tuples; logging of first access in a session or all accesses, etc.).

Logged data may be written to files outside database storage, or to system tables within the database. In the former case, protection depends on operating system access control, while in the latter case protection of audit data depends on the access controls of the database. In particular, the DBA often has privileges to read or alter system tables. When audit data is stored in tables it can have a substantial performance impact on normal database operations.

Database users can implement their own auditing through user-level triggers. A trigger is a user-defined procedure that executes before or after designated events in the database. The triggering event can be an insert, delete, or update command, and most systems allow for tuple-level execution (in which the rule executes once for each tuple affected) or statement level (in which rule executes once for each statement). User-level triggers can be inefficient (especially tuple-level triggers) and the scope of events that can act as trigger events may be limited.

The database transaction log is designed to support critical ACID properties for the concurrent execution of transactions. The transaction log typically includes the before and after images of all database modifications to allow for rollback of aborted transactions and the redo of changes lost due to system failure. While the transaction log contains a wealth of information relevant to auditing, it has a number of limitations when used as an auditing mechanism. First, key data is missing from the transaction log: namely a record of read accesses to the database, as well as some operational context information. In addition, performing an audit analysis using the transaction log could require recovering the state of the database as of a past moment in time. Although a number of current systems provide such point-in-time recovery by reinstating backups and rolling transactions forward, using this mechanism for audit analysis is very inefficient. Finally, the retention period of transaction log data and audit log data may be substantially different. Transaction logs are often implemented as circular files in which old log records no longer needed for recovery are overwritten. Retention periods for auditing data may be much longer.

In a persistent database (also known as an archiving, or transaction-time database) the historical state of the database is purposely retained as modifications are applied. In such systems a deletion never destroys data and an update merely creates a new version of a tuple. It is possible to pose queries "as-of" any past point in time. Persistent databases have received considerable attention from the research community [5], motivated by both auditing and other applications. Combined with query logs and system logs, a persistent database can offer the most complete audit collection along with efficient audit analysis since the historical state of the database can be queried.

Support for persistence has not been widely implemented, and is not usually used to support auditing in commercial systems. A number of research projects have built persistent, temporal or transaction-time databases, many as extensions to existing systems like MySQL [8], BerkeleyDB [7], and SQL Server.

Protecting the Audit Log It is essential that the audit log be protected from unauthorized modification so that it accurately reflects history. The database administrator, and other privileged parties, should not be capable of tampering with the audit log. Typically there is no legitimate reason for records in an audit log to be modified. The log should be append-only and may be implemented using write-once media. Deletion of the audit log should occur only when it is clear that the log is no longer needed for audit inquiries. Cryptographic techniques have been proposed for detecting tampering of database audit logs and for efficiently tracing the location of illegally modified records once tampering has been discovered.

It is equally critical that the confidentiality of the audit log be protected. The audit log poses multiple privacy threats: the audit log contains records from database that may be sensitive, as well as a history of how the database was used (the users of the database, the queries that were executed, and times of day can all violate the privacy of individuals). Viewing audit logs is a highly privileged operation in most systems. Recent research has investigated the use of cryptography to permit searching over encrypted audit logs to minimize the disclosed data during an investigation [10].

Database Forensics

The goal of database forensics is the analysis of a database system's contents to validate hypotheses about past events that are relevant to an alleged crime or violation of policy. This is a challenge since recovered evidence is typically only a partial record of past events. The goal of the analysis is a formal presentation of recovered data in a court of law, and therefore it is critical for the investigator to also understand legal concepts, including evidence handling. Unlike auditing, there is no limitation on the type of data that can be of interest. Broadly, there are two types of evidence.

- Database evidence can be the direct subject of a crime. For example, records can store contraband, such as images of child pornography, copyrighted media, and stolen intellectual property.
- Evidence can be indirectly related to a crime, for example data from a log that verifies that a relationship exists between two users or computers. Or a log of database query terms can corroborate the notion that a user had knowledge of and intent to possess particular contraband content found in their file system.

Typically investigations cover not only databases but other computer systems (e.g., file systems, email stores, web history) as well as aspects of a crime scene beyond the computer. Evidence from the entire scope of a crime scene must be synthesized and reported as testimony that is persuasive to an adjudicator.

Harvesting Database Evidence Databases are among the most complicated systems found in modern computers. Investigators can harvest evidence from table storage, indexes, transaction and audit logs, temporary caches, database catalogs, or archived copies of records on backed up media.

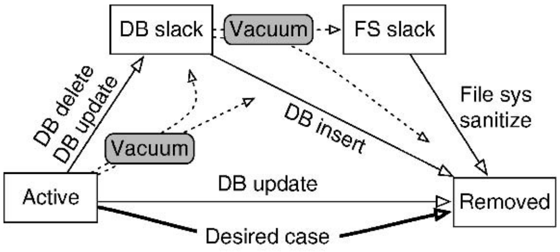
Data values have a complex lifetime with a database system. When input to a database, records begin in an active state, which means a database and its services need the record in order to function properly. Database operations can change or create active records or remove the purpose of active records. In the latter case, records become expired. For example, a record becomes expired when it is deleted and when there is no combination of operations left that would ever use the record again. Forensic investigators are interested in both active and recoverable expired records.

The lifetime of data is illustrated in Fig. 1 for the simple case of records in table storage. An insertion creates an active record, a deletion changes a record from being active to being expired and eventually the data may be overwritten by another database operation. After expiration, a tuple can be either removed, or it can continue to exist as recoverable slack data. Methods of removal are discussed below.

In the case of table storage, each paged file of storage is shared by many records. The database API enables users or investigators to retrieve all active values. When data is deleted by users, typically a single bit is flipped in the page file to indicate removal of the data. However, the record can be retrieved outside the mechanisms of the API.

Other database mechanisms can also leave recoverable records. Updates to records with variable lengths can replace one or more attribute values with smaller attribute values; the tail-end of the old record will remain partially recoverable until it is overwritten. Or an administrator may initiate a *vacuum* command to improve storage performance. When vacuum executes, on many systems, the reorganization is not performed completely in place. In addition to reorganizing records within and across pages, the size of the file used for table storage may be reduced, returning space to the file system, creating the possibility that the database records can be recovered through file system forensics.

Expired data can also be found in stored indexes if, for example, entries in B+tree nodes are deleted but not overwritten immediately. Temporary relations, materialized for improved query processing or used for external sorting, also contain data recoverable as filesystem slack. Data can be recovered from transaction logs, which are typically implemented sequentially written circular files where the newest data overwrites the oldest portion of the log. The amount of recoverable data is dependent on the file system space allocated to the log and characteristics of the database, including the rate and size of updates and checkpoints. Similarly, the amount of data stored in backups varies with policy, storage capacity, and use.



Auditing and Forensic Analysis. Figure 1. The flow of data during its lifetime. It begins in the active state. Before it is deleted and becomes expired it will often be retained as database slack or filesystem slack [9].

System Transparency and Privacy Forensic analysis is not restricted to active tuples because database designs do not strive to eliminate unintended retention of data accessible through interfaces that are not controlled by the database. This incongruence between what the database presents to users and what is actually stored represents a threat to privacy and confidentiality.

For example, as stated above, businesses can unintentionally violate privacy regulations when deleted data is left in table or file storage. Adversaries that investigate databases recovered from lost or stolen computers can reveal sensitive information that was thought to be deleted.

From this point of view, it is desirable for a database to operate in a forensically *transparent way* [9]. Stahlberg et al. have proposed a set of desiderata to determine the extent to which a database system is forensically transparent. A database system is forensically transparent if it satisfies all three desiderata.

Clarity: The impact of each operation on the state of records, whether active or expired, is clear to the user.
Purposeful retention: Only *active* records should be retained by the database.
Complete removal: Expired records must be *removed* by the system within a short, fixed time from when they become expired. In other words, there must be a small upper bound on the time that slack data exists in the database.

Databases that satisfy these desiderata provide a reliable interface to the user in terms of what data is actually stored in the system.

Removal of Data Ensuring that removed data is unrecoverable from a database system is difficult. Records can be removed by overwriting storage with a standard pattern (e.g., all zeros) or through the use of encryption.

Deletion through overwriting can be costly in terms of performance and therefore should be limited to small records or files. Asynchronous overwriting during idle periods can ameliorate performance degradation while opening a window of opportunity for recovering data. Secure removal of data can also be accomplished by storing data in encrypted form and using overwriting to remove the decryption key. This technique was first proposed for efficient simultaneous removal of data from files and backup logs. Stahlberg et al. [9] present an extended discussion of the use of encryption keys and overwriting in the different internal database mechanisms.

Key Applications

Auditing and forensic analysis are critical operations in any setting where sensitive or high-value data is exposed to untrusted users. For example, database applications that manage financial, national intelligence, or medical information must maintain audit records of past data and operations, and may be the subject of forensic analysis if policy violations occur.

Future Directions

A frequent outcome of auditing and forensic analysis is the detection of a malicious or mistaken operation in the past. Correcting the corrupted database, especially after many valid transactions have been applied, is a significant challenge receiving recent attention by the research community [3,6].

Also note that the above discussion has focused on a single database server. In modern applications, a comprehensive audit or forensic analysis is likely to involve many integrated systems where data objects are derived from diverse sources and follow a complex workflow. In such settings auditing inquiries share some similarities with data provenance (or lineage), which is a record of how a data item has come to exist in a database or view.

Cross-references

- ▶ Data Quality (Lineage, Provenance, Curation, Incomplete and Imprecise Data)
- ▶ Database Security and Privacy
- ▶ Inference Control in Statistical Databases
- ▶ Intrusion Detection

Recommended Reading

1. Adam N.R. and Wortmann J.C. Security-control methods for statistical databases: A comparative study. ACM Comput. Surv., 21(4):515–556, 1989.
2. Agrawal R., Bayardo R.J., Faloutsos C., Kiernan J., Rantzau R., and Srikant R. Auditing compliance with a hippocratic database. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 516–527.
3. Ammann P., Jajodia S., and Liu P. Recovery from malicious transactions. IEEE Trans, Knowl. Data Eng., 14(5):1167–1185, 2002.
4. Castano S., Fugini M.G., Martella G., and Samarati P. Database security. ACM/Addison-Wesley, New York, NY, USA, 1994.
5. Jensen C.S., Mark L., and Roussopoulos N. Incremental implementation model for relational databases with transaction time. IEEE Trans. Knowl. Data Eng., 3(4):461–473, 1991.
6. Lomet D., Vagena Z., and Barga R. Recovery from "bad" user transactions. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 337–346.
7. Snodgrass R.T. and Collberg C.S. The τ-BerkeleyDB temporal subsystem. Available at www.cs.arizona.edu/tau/tbdb/.
8. Snodgrass R.T. and Collberg C.S. The τ-MySQL transaction time support. Available at www.cs.arizona.edu/tau/tmysql.
9. Stahlberg P., Miklau G., and Levine B. Threats to privacy in the forensic analysis of database systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 91–102.
10. Waters B., Balfanz D., Durfee G., and Smetters D. Building an encrypted and searchable audit log. In Proc. Network and Dist. Syst. Security Symp., 2004, pp. 91–102.

Auditory Scene Detection

▶ Audio Segmentation

Authentication

Marina Blanton
University of Notre Dame, Notre Dame, IN, USA

Definition

Authentication is a broad term, which is normally referred to mechanisms of ensuring that entities are who they claim to be or that data has not been manipulated by unauthorized parties. Thus, *entity authentication* or *identification* refers to the means of verifying user identity, after which the user will be granted appropriate privileges. *Data origin authentication* refers to the means of ensuring that the data comes from an authentic source and has not been tampered with during the transmission.

Historical Background

The need for user authentication in early computer systems arose once it became possible to support multi-user environments. Similarly, data base systems that can be accessed by multiple users with different privileges have to rely on user authentication to enforce proper access control. There is a variety of mechanisms that allow users to authenticate themselves, but password-based authentication is currently the most widely used form of identification.

Data origin authentication (or *data authentication* for short) is also an old concept which gained importance with the adoption of inter-computer communications. With respect to data base systems, data authentication is crucial in distributed environments and when data bases are disseminated to other entities.

Foundations

Identification

The purpose of *entity authentication* or *identification* is to allow one party (the *verifier*) to gather evidence that the identity of another party (the *claimant*) is as claimed. Thus, authentication protocols should permit honest parties to successfully finish the protocol with the claimant's identity being accepted as authentic and make it difficult for dishonest parties to impersonate an identity of another user. Impersonation must remain difficult even for an adversary who can observe a large number of successful executions of the authentication protocol by another entity.

Identification mechanisms can normally be divided into the following types depending on how the identity evidence is gathered:

1. *The user knows a secret.* Such types of identification include passwords, personal identification numbers (PINs), or secret keys.
2. *The user possesses a token.* This is normally based on a hardware token such as magnetic-striped cards (or smartcards) or other custom-designed devices that generate time-variant passwords.
3. *The user has a physical characteristic.* Identification can be based on characteristics inherent to the user being authenticated such as biometrics, handwritten signatures, keystroke dynamics, facial and hand geometries, voice, and others.

Note that for added security often different mechanisms can be combined together. For example, PIN-based authentication is almost always used in conjunction with a physical device that stores information about its owner (i.e., user ID, credit card number, etc.). Likewise, biometric-based recognition can be used in combination with a password or a physical token.

Before a user will be able to engage in an authentication protocol, she needs to *register* with the system and store the data (e.g., a password, keys, biometric data) that will thereafter aid the system in the authentication process. Password-based authentication is treated next, followed by stronger forms of entity authentication.

Password-Based Authentication

Identification based on conventional time-invariant passwords is the most widely used form of identification even though such approaches do not provide strong authentication. A password is a string of (normally 8 or more) characters associated with a certain user, which serves the purpose of a shared secret between the user and the system. When a user initiates the identification process, she supplies the system with the pair (*userid, password*), where *userid* identifies the user and *password* provides the necessary evidence that the user possesses the secret.

The most straightforward approach for the system to store passwords is in the clear text. This, however, allows the system administrator, or an adversary in case of system compromise, to recover passwords of individual users leading to security concerns. Thus, most systems first apply a one-way *hash function* to each password and store the output in the system. Then when a user supplies the password during the identification process, the password is first hashed and then compared to the string stored in the system. This security measure no longer allows cleartext passwords to be recovered, but other attacks on passwords are still possible. In particular, the following attacks can be carried out:

- *Replay of passwords.* Since passwords are reusable, an adversary who obtains password information (by either seeing the user type the password, using a keylogger program, or capturing the password in transit from the user to the system) will be able to reuse it and successfully impersonate the user.
- *Exhaustive search.* An adversary might attempt to guess user passwords by trying all possible strings as potential passwords (on the verifier itself or by obtaining a copy of the password file and performing this attack off-line). Generally it is infeasible for an adversary to try all passwords if they are chosen from a sufficiently large space, but it is

possible to exhaust short passwords (e.g., of length 6 characters or less).

- *Dictionary attack.* It is well known that users tend to choose passwords that they can easily remember but which are considered weak from the security point of view. Thus, an adversary might try to guess a user password using words from a dictionary and variations thereof. If a user makes a poor password choice, such an attack can have a high probability of success. Dictionary attacks become increasingly complicated, testing for combinations of words, common substitutions and misspellings, insertion of additional symbols, words from foreign languages, etc.

To decrease the vulnerability of the system to these attacks, additional measures are normally employed, some of which are:

- *Salting passwords.* In order to make guessing attacks less effective, many systems use an additional random string, called *salt*, with each password. Before a password is stored, it is augmented with a random salt, hashed, and then stored in the system along with the salt. This prevents an attacker who is in possession of a file with many user passwords to launch a dictionary attack against all of them at the same time, and requires each user's password to be tested individually.
- *Slowing down password verification.* To defeat against attacks that perform trials of a large number of passwords, the function that computes the hash of the password can be made more computationally extensive. This, for example, can be done by iterating the computation n times. When increasing the computation for password mapping, a care, however, must be taken not to impose a burden on legitimate users.
- *Limiting the number of unsuccessful password guesses.* It is common for a user account to be locked after the number of unsuccessful authentication attempts exceeds a certain threshold. The owner of a locked account must then contact an administrator and have the account activated.
- *Password rules.* To protect against guessing attacks, often certain rules are imposed on user choice of passwords such as the minimal password length and/or usage of capital letters, numbers, and special symbols. Such rules normally strengthen the password choices but they also limit the password search space. Also, a technique called *password aging* is often employed to force users to choose a new password after a certain period of time. If such a period is rather short, however, users will be unable to remember a newly chosen strong password, thus weakening the security of the system with bad password choices.

It is always a challenge to find a balance between memorability of passwords (passwords that are hard to remember tend to be written down) and their resistance to dictionary attacks (i.e., passwords with enough randomness in them). Thus to aid users in choosing less predictable passwords which they can remember, techniques exist to create computer-generated *pronounceable passwords* which are based on mnemonics. Also, recently various solutions have been developed to use images for authentication (a user is given a number of images and is asked to identify the set of pre-selected images), graphical interfaces (a user draws a pattern on a grid that has to match a previously chosen pattern), etc. Such systems, however, are not widely deployed and have not been thoroughly evaluated to determine the level of security they provide.

Since a major security concern with fixed passwords is the possibility of replaying them, a natural way to improve their security is to consider *one-time passwords.* As the name suggests, in such systems each password is used only once and there are different ways to realize them, which is briefly outlined next.

- The user and the system initially agree on a sequence of secret passwords. Each time the user authenticates to the system, a new password is used. This solution is simple but requires maintenance of the shared list.
- The user updates her password with each instance of the authentication protocol. For instance, the user might send the new password encrypted under a key derived from her current password. This method crucially relies on the correct communication of the new password to the system.
- The new password is derived with each instance of the authentication protocol using a one-way *hash function.* As an example, consider the scheme called S/Key due to Lamport [2]. The user begins with a secret k and applies a one-way *hash function* h to produce a sequence of values $k, h(k), h(h(k)), \ldots, h^t(k)$. The password for ith identification session

is $k_i = h^{t-i}(k)$. When the user authenticates $(i+1)$ st time with k_{i+1}, the server (which has k_i for that user stored) checks whether $h(k_{i+1}) = k_i$ and, if so, accepts the authentication and replaces k_i with k_{i+1}. This check convinces the server because the function h is considered to be infeasible to invert and only the legitimate user will be able to construct k_{i+1} that passes the check.

Challenge-Response Identification

Challenge-response techniques provide a strong form of entity authentication as they are not vulnerable to replay attacks. The main idea behind such protocols is that the claimant possesses a secret. During an identification session, the server sends to the claimant a unique randomly chosen *challenge*. The claimant computes a *response* which is a function of her secret and the server's challenge and sends it to the server. It is important to note that the response does not provide information about the user secret, and cannot be used to successfully compute responses to server's challenges in the future by someone who monitors the message exchange.

A variety of cryptographic challenge-response techniques exist which could be based on (i) symmetric encryption, (ii) one-way *hash functions*, or (iii) public-key encryption. A more detailed explanation of such techniques is beyond the scope of this article and can be found in standard textbooks on cryptography such as [3,7].

Data Origin Authentication

The purpose of *data origin authentication* is to ensure that the data comes from a trusted source and has not been tampered with during the transmission. Techniques that permit verifying data authenticity can be divided in two categories: (i) the communicating parties *share a common secret* and (ii) the communicating parties *do not share a secret.*

Consider that in a distributed database environment two systems communicate often and there is a need to ensure data integrity. Then such systems can share a secret S that permits them to use *message authentication codes* (MAC) for data authentication. That is, prior to transmitting the data, the sender constructs the MAC using S and sends it along with the data. After obtaining the data, the receiver uses the shared secret to reconstruct the MAC and compare it with the MAC received. If the check succeeds, the data is accepted as authentic, and it is discarded otherwise.

In cases when the sender and the receiver do not already have a secret which is known to both of them, *digital signatures* can be used to verify the authenticity of the sender and integrity of the data. This mechanism assumes that the sender has a public-private key pair, which is used to sign the data. Then the sender uses her private key to sign the message and the receiver uses the corresponding public key to verify the fact that the message arrived intact.

The standard way of producing a digital signature on data is first to apply a one-way *hash function* on the data to compute its digest and then sign the digest. The purpose of computing the digest first is to compress the data to a short string, which then can be efficiently signed to produce a fixed-size signature. In cases when integrity of a database needs to be verified with some regularity while only parts of it change, more advanced techniques can be used. In particular, Merkle hash tree is commonly used to produce a digital signature on a hierarchically structured set of documents (e.g., an XML tree of documents). In such a tree, digests of individual nodes are computed and then combined in a bottom-up fashion to result in a single short digest of the tree. The owner of the data produces a signature on the root node only. Verification of data integrity in such trees can normally be done faster than recomputing digests of the entire tree if the user would like to verify the integrity of only a part of the tree.

Key Applications

The main application of authentication is access control. Namely, in multi-user systems a user authenticates to the system and is granted access to specific resources determined by her access privileges. The mechanisms used to determine user access rights vary drastically from one system to another and are based on the type of access control (e.g., role-based, discretionary, etc.,) and access control policies.

Also, authentication applications and services such as Kerberos, X.509 Authentication Service, or Public-Key Infrastructure (PKI) can be used to aid in the authentication process.

In case of data authentication, verification of data integrity and authenticity is essential in determining trustworthiness of the data. For example, updated database records that are coming from a trusted source can be safely used to modify the current contents of the database. If, on the other hand, data integrity and authenticity of the modifications cannot be verified, in many cases such data will not be trusted.

Cross-references

▶ Access Control
▶ Digital Signatures
▶ Hash Functions
▶ Merkle Hash Trees
▶ Message Authentication Codes
▶ Security Services
▶ Storage Security

Recommended Reading

1. Bishop M. Computer Security: Art and Science. Addison Wesley Professional, 2002.
2. Haller N. The S/Key one-time password system. In Proc. Symp. on Network and Distributed System Security, 1994, pp. 151–157.
3. Menezes A., van Oorschot P., and Vanstone S. Handbook of Applied Cryptography. CRC, 1996.
4. Pfleeger C. and Pfleeger S. Security in Computing, 3rd edn. Prentice-Hall, 2003.
5. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edn. Wiley, 1996.
6. Stallings W. Cryptography and Network Security: Principles and Practices, 4th edn. Pearson Prentice Hall, 2006.
7. Stinson D. Cryptography: Theory and Practice, 3rd edn. Chapman & Hall/CRC, 2006.

Authentication Trees

▶ Merkle Trees

Authorization Administration Policies

▶ Access Control Administration Policies

Authorization Administration Privileges

▶ Access Control Administration Policies

Authorization Policy Languages

▶ Access Control Policy Languages

Authorization Verification

▶ Access Control

Auto-administration and Auto-Tuning of Database Systems

▶ Self-Management Technology in Databases

Auto-Annotation

▶ Automatic Image Annotation

Automata Induction

▶ Grammar Inference

Automatic Abstracting

▶ Summarization

Automatic Image Annotation

Nicolas Hervé, Nozha Boujemaa
INRIA Paris-Rocquencourt, Le Chesnay Cedex, France

Synonyms

Multimedia content enrichment; Image classification; Object detection and recognition;Auto-annotation

Definition

The widespread search engines, in the professional as well as the personal context, used to work on the basis of textual information associated or extracted from indexed documents. Nowadays, most of the exchanged or stored documents have multimedia content. To reduce the technological gap so that these engines still can work on multimedia content, it is very convenient developing methods capable to generate automatically textual annotations and metadata. These methods will then allow to enrich the upcoming new content or to post-annotate the existing content with additional information extracted automatically if ever this existing content is partly or not annotated.

A broad diversity in the typology of manual annotation is usually found in image databases. Part of them is representing contextual information. The author, date, place or technical shooting conditions are quite frequent. Some semantic or subjective annotations, like emotions that flow out from images, can be found. Some other annotations could be related to the visual content of images. They provide information on a given image such as indicating whether it is a drawing, a map or a photograph... For photographs, the global aspect is often specified (vertical/horizontal, color/black and white, indoor/outdoor, landscape, portrait...), as well as the presence of remarkable objects or persons.

The aim of automatic image annotation approaches is to provide efficient methods that extract automatically the visual content of pictures allowing semantic labeling of images. This is generally achieved by learning algorithms that, once being trained on annotated sub-corpora, are able to suggest keywords to the archivist through object detection/recognition and image classification methods.

Historical Background

The exploration of visual content databases and their querying to retrieve some specific content usually rely on textual annotations that have been previously provided manually by human operators. The outcome of the tremendous improvements in digitization and acquisition devices is the availability of exponentially growing content. Usual annotation techniques then became more and more difficult to apply because they are time and cost consuming. Moreover, manual annotations are far from being perfect. They are often focused on the context, subjective, partial and driven by the needs of the end-users at the time they are produced. As these needs are evolving, part of the existing annotations becomes irrelevant and others are missing. This is especially true with the arising of Internet and the availability of all kind of databases online. An other issue lies in the lack of controlled vocabularies for most of the databases making difficult for the end-user to guess what query words he has to use in order to retrieve the content he has in mind.

Visual content indexing and retrieval community have achieved significant progress in the recent years [13] toward efficient approaches for visual features extraction and visual appearance modeling together with developing advanced mechanism for interactive visual information retrieval. One of the major issues was and remains the semantic gap [4,8].

Two main types of images databases could be distinguished. Specific databases are focused on a given restricted field. In the scientific domain, one can cite satellite images for weather forecast or cultivation study, medical images or botanical databases for species recognition. They are also found in the cultural heritage domain (e.g., paintings databases) or the military and security domain (e.g., fingerprints and faces databases). On the other side, generic databases contain very different images, without any *a priori* on their content. This is usually the case for professional news agencies, illustration photo stock collections and personal family and holiday photo albums. Only methods for generic content databases labeling will be addressed.

By analyzing automatically images and characterizing them with low-level features (mainly colors, textures and shapes) CBIR systems [3] provided new query paradigms that enable users to express their needs. The main one is "query by example" where the system retrieves images of the database that are the most similar to a given example. The scientific community has been facing the well known semantic gap problem for a while which remain the major concern of the research community. Since the late 90s, relevance feedback mechanism is one of possible solutions to this difficult problem.

The early papers on automatic annotation that have been published tackled image orientation detection or the classical indoor versus outdoor and city versus landscape classifications of photographs [14,15]. Recently, relevance feedback allows moreover helping for interactive mass-annotation of image collections. This approach is often referred to as semi-automatic image annotation.

Foundations

Despite some of its drawbacks, the query by keyword is still very useful and quite natural for the end-user [6]. Automatic annotation generates such keywords to enrich the images semantic descriptions and ease further querying. Because of the computational costs of all current approaches, the existing systems are always composed of two parts. An offline part is in charge of indexing the visual content and generating the annotations. Eventually, a human operator can help the system during the process or after it to validate/invalidate the produced annotations. In such cases, one talks of semi-automatic annotation systems. The second part, online and real-time, is a query by keywords module.

As the main purpose is to describe the visual content, the term "visual concept" is preferred over keyword to describe the labels a system has to discover in images. As previously mentioned, these visual concepts could be related to either global appearance of the image or presence of some objects. Objects detection can also be refined in generic object class detection or specific object instance detection. For example, one can ask a system to label only the "vehicle" concept, or more precisely to distinguish cars, motorbikes, boats and airplanes, and, at a very specific level, being able to recognize different makes of cars. This is the same problem with annotating persons. Being able to detect the presence of a person in an image is a different procedure and result than recognizing him. As face recognition is a well studied problem which is tackled by a specific research community. The ability to generalize from a few examples and to reach higher abstraction levels is natural for humans but it is very challenging task to achieve with current state-of-the-art's annotation systems [5,7,10,12].

One of the fundamental hypotheses of automatic annotation is that what looks similar is probably semantically similar. Most of the approaches rely on this assumption. The main generic steps of automatic annotation are described below. First, visual features are extracted automatically from images in order to obtain representations in a visual space. The second step is to build models that will link the visual concepts to the relevant information in the visual space. When new content is proposed, models are then able to predict the corresponding visual concepts.

The performances evaluation of such methods may rely on the usage of the annotations by the final users. As in most of information retrieval systems, precision and recall measures are used. Precision emphasizes the retrieval of relevant documents earlier and recall focuses on the retrieval of the full set of relevant documents. Precision and recall are complementary to judge the quality of a system. But for some applications, precision is the only important measure. This is especially the case when a huge image database is available (like Internet). When doing a query, a user is more interested in the first satisfying results than in the complete relevant result set.

Images Description with Low-Level Features

The visual description of images is of great importance as it is the raw material on which further models are built. There is not a universally good low-level features extractor. In specific databases, *a priori* on the content of images can be used to extract specialized features that will better describe their special nature. For example, numerous features can be found in the literature for faces or fingerprints description. In generic databases, compromises have to be made between exhaustiveness, fidelity to the content, ability to generalize and different invariance degrees (illumination changes, rotations, scales, occlusions. . .). The use of inappropriate features leading to poor performances of a system has often been described as semantic gap. In this case, one rather faces the numerical gap, meaning that the visual information is present in images but it has not been extracted correctly.

Due to their ability to generalize to content in different conditions, statistical features are often used. They gather color, shape and texture information in histograms, separately or jointly. Color histograms are among the first features used to describe images. They vary depending on the underlying color space that is used, the quantization parameters, different weighting schemes or the use of co-occurrences of colors. Shapes can be described by properties of edges found in images like their types, orientations or lengths. Textures are focusing on the analysis of frequencies in images. They often rely on Fourier transform, Gabor filter banks or wavelets. Some features also combine different types of information, mixing for example color and texture in a single representation. Typically, these visual features are represented by vectors in high-dimensional spaces (generally between a few tens and a few hundreds dimensions) [9,15].

Initially, the features were extracted over the full image. This approach is well suited to describe the global aspect of the content but is too coarse to represent small details and objects. Features need to be extracted locally. First, a support region has to be determined. Once its location, shape and size are known, features are computed on this small portion of the image. These features can be of the same types as those extracted at a global level or they can be specialized according to the nature of the support regions. Several strategies are used to select the support regions. Segmentation algorithms try to find the boundaries between homogeneous regions in images [2]. Segmentation is a difficult problem in itself that is not well defined. Unfortunately, the general trend has always been to focus on segmentation that detects objects, which is already a highly semantic task and, thus, not really achievable through automatic processes. Alternative approaches consist on sliding windows and fixed grid, with varying sizes and spacing, are common ways of obtaining dense sampling of the visual content [9,14]. Another popular region selection approach is based on local features detectors via point-of-interest. They were originally designed for image registration. These detectors are generally attracted to specific areas of images that have high variation in the visual signal, such as the vicinity of edges and corners of regions. They allow the selection of a very small proportion of image locations having the highest visual variance [11,16]. Typically, when using dense sampling, point-of-interest or when mixing them, between a few hundreds and a few thousands features are extracted per image. The computational cost is then much higher than with global features. Some representations also try to carry other information, like geometrical relations between features locations or contextual information [1].

With global features, the image representation is straightforward. However, even when local features are to be used, learning algorithms may sometimes require a global image representation that encompasses all the local visual information. The bag-of-visual-words representation, very much inspired by the classical bag-of-words representation for text, is one of the most popular for images. A visual vocabulary composed of visual words (some representative features) is generated. An image is then represented by a coordinate vector, each value of which expresses the degree of importance of a feature with respect to the image and/or the database as a whole. The creation of a visual vocabulary is an important step in the full process. The selected visual words (a few hundreds to a few thousands) have to be representative of the database content as they will serve as a basis for further representation. Creating a good vocabulary will avoid the loss of too much local information. Generally, clustering algorithms either supervised or not, are used with a sample of the database. This step can be seen as a quantization of the local features.

Whatever the selected representation, the similarity between images is measured by a distance functional in the visual space. A broad variety has been developed: classical Euclidean distance (L2), L1, earth mover distance (EMD), chi-squared (χ^2), vector angle, histogram intersection,. . .

Learning and Models

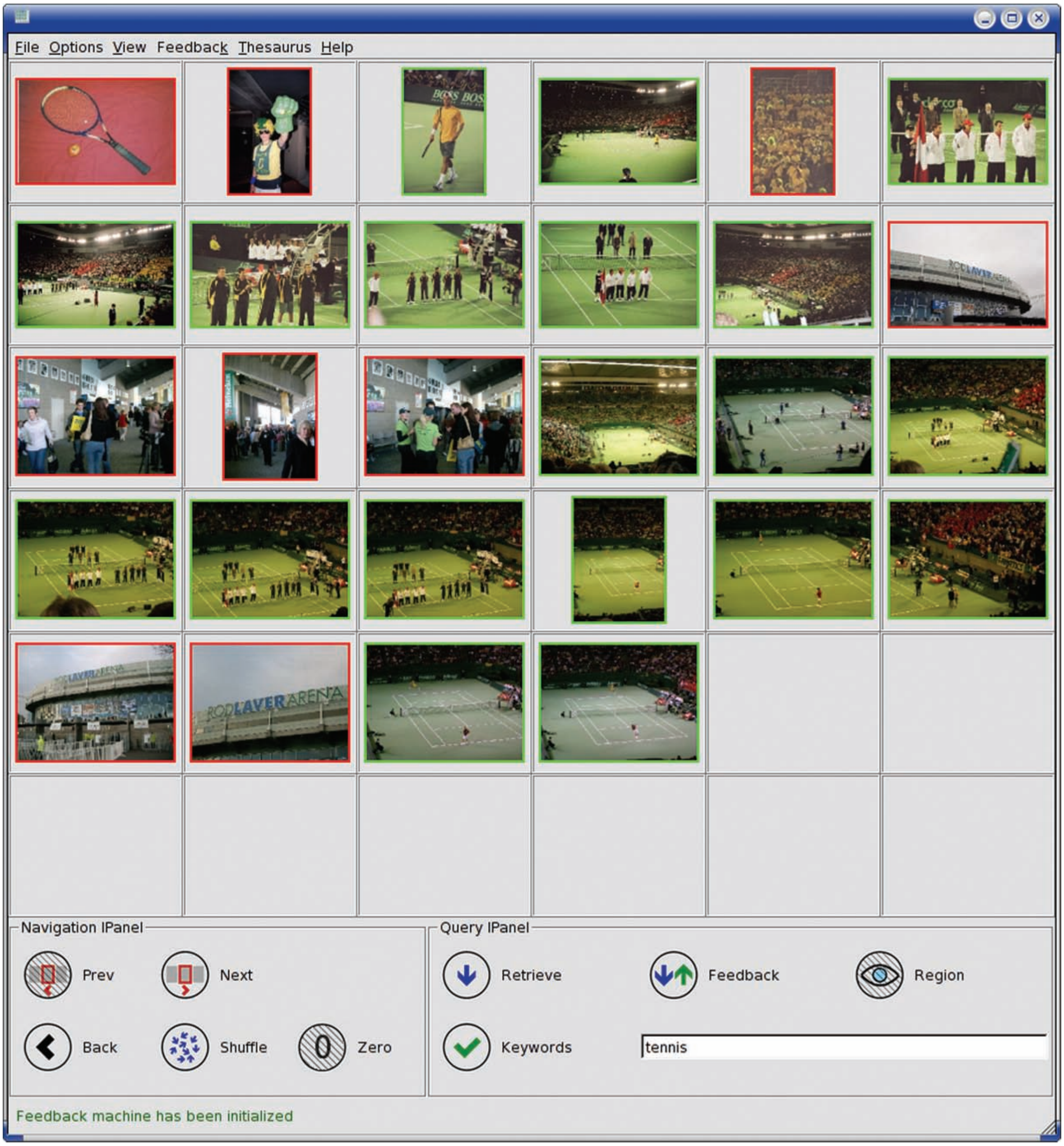
Although several formulations have been proposed, the main purpose of building models for visual concepts is to associate them with the visual space regions that best represent them. This problem is at the crossroads of computer vision, data mining and machine learning. Usually, the models are built through a supervised learning process. For given visual concepts, an algorithm is fed with a training dataset containing both positive and negative images regarding the concepts to learn. This algorithm has to find the discriminant information from the visual space that best models the concepts. Generally, the available annotations for the training set are not localized. The presence of a visual concept for an image is known, but its exact location is not provided. This is the case for almost all professional and personal databases. In the same way, annotating new images does not require to locate exactly the visual concept, but only to predict its presence. This is the main distinction that can be made with object detection tasks.

Two main learning algorithm families are used:

- *Generative*: the system tries to estimate density distribution of concepts in the visual space or other hidden variables [2]. Popular examples include Gaussian Mixture Models, Hidden Markov Models, Bayesian networks, Latent Semantic Analysis and translation models from the text processing community. The Expectation-Maximization algorithm is often used to train these models.
- *Discriminative*: instead of trying to model the distributions, discriminative approaches are focusing on detecting the boundaries between classes. For each visual concept, the annotation process is then often formalized as a two-class classification problem (present/not present). Although appearing to be a little bit more effective, discriminative approaches do not have the elegance of generative ones. They act more as black boxes and relationships between the different variables are not explicit, and thus difficult to analyze. The most famous algorithms are Support Vector Machine (SVM) [9,16], boosting (e.g., adaboost) [11] and all flavors of discriminant analysis (linear – LDA, biased – BDA, multiple – MDA, Fisher – FDA).

Both approaches may use global or local representations. Methods using bag-of-words representations are also called "multiple instance learning." Sometimes, pre-processes may also be used to prepare the data in order to enhance the performances or to reduce the computational costs: feature selection, dimensionality reduction, scaling or normalization. Current systems are often composed of several components, using different low-level features, combining them according to different schemes and training models with multiple learning strategies.

Once the models have been learned, they can be used to predict the visual concepts. Two types of predictions are possible. Hard decision simply indicates the presence or absence of the concept. Soft decision also provides a degree of confidence in the prediction, allowing ranking more easily the results when answering an end-user query and thus improving the retrieval of pertinent images earlier.



tennis

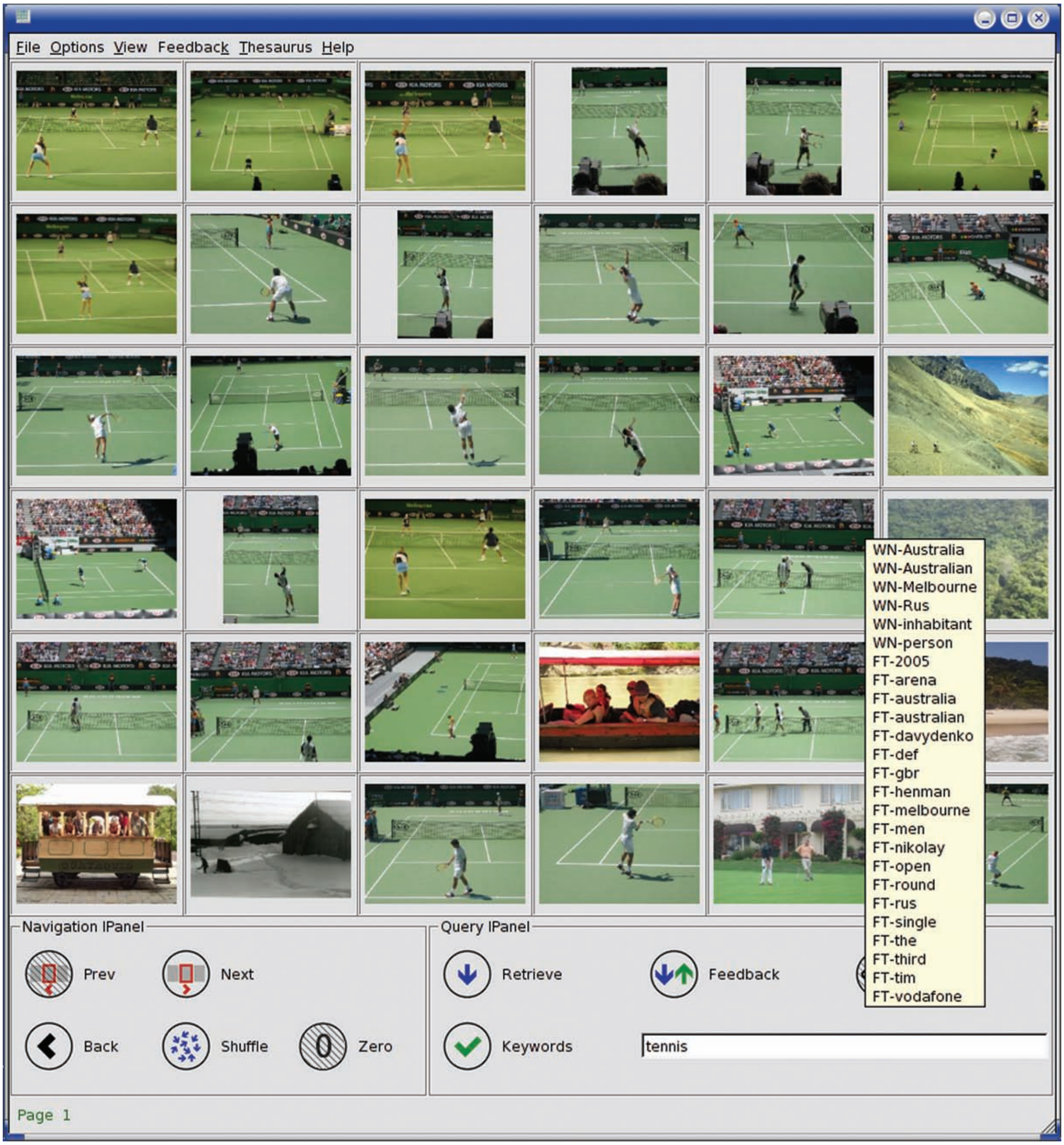
Feedback machine has been initialized

Automatic Image Annotation. Figure 1. Pictures annotated with the "tennis" keyword.

Current Results

The different methods are actually mature enough to predict global visual concepts like image types and scene categories. Regarding local concepts, huge improvements still need to be made in order to provide useful applications to real users. Both dense sampling and point-of-interest have shown to perform quite well on research databases, but results on real databases are quite poor [9,12]. A good indication of state-of-the-art performances can be obtained in the results of official benchmark campaigns like ImagEval, Pascal VOC, Imageclef or Trecvid. In all cases, the contextual information (visual or from the existing metadata) has shown to be of great importance in the results.

When the availability of correctly annotated images for a given visual concept is not guaranteed, the offline



Automatic Image Annotation. Figure 2. Pictures displayed after two iterations.

learning approach is not possible. One of the solutions is then to interact with a user through relevance feedback, also called interactive learning. In a few iterations, the user will provide the system positive and negative examples and guide it to recognize the visual concept. Part of the mechanism involved are the same as offline learning, but the training labels are provided online by a user. As an example, the following screen captures from Ikona [3] are showing the IAPR-TC12 database, used for the Imageclef benchmark. The first screen displays all the images annotated with the "tennis" keyword. The user is only interested in pictures where a tennis court is visible. He indicates positive (green border) and negative (red border) examples. After two iterations, one can see on the second screen that a lot of tennis court pictures have been retrieved. None of them was annotated with the "tennis" keyword. After a few more iterations, when no more correct pictures are retrieved from the database, the user is able to annotate massively all the pictures gathered through the iterations that were kept in a specific basket (third screen) (Figs. 1–3.)

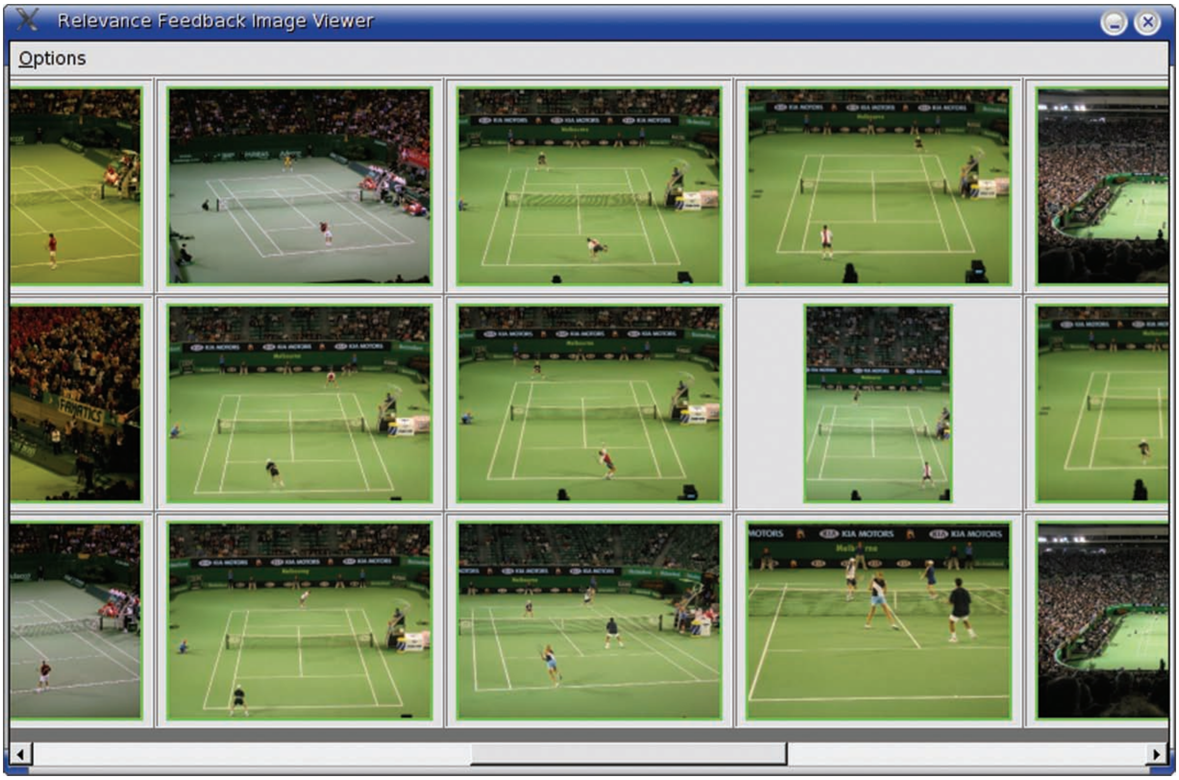
Key Issues and Future Research

The lack of generalization ability for both visual features and learning algorithms has to be compensated by a huge number of training examples. Depending on the complexity of the visual concept, a good estimation is around a hundred positive examples and ten times more negatives examples for the training set. Paradoxically, despite the tremendous amount of images available nowadays, finding content that has been reliably annotated for training dataset is hard.

The computational complexity is also still too high for real-time annotation when dealing with several thousands of visual concepts. Research is made on scalability issues in machine learning and is linked to existing high-dimensional data indexing structures.

Progresses for better description and integration of all types of available information need to be achieved.

There are also some questions arising: will the problem be solved with more computational power when one is able to process images at every scale and location in real-time? Are massive collaborative annotation websites, like Flickr, going to change the annotation paradigm



Automatic Image Annotation. Figure 3. All positive pictures basket allowing mass-annotation.

by transforming Internet in a giant common repository? What is the impact of GPS metadata, and more generally all the new information captured directly when a photograph is taken?

Key Applications

- Professional content owners: post-editing
- Personal family and holiday photo albums
- Web image search
- Searching into poorly human-made annotated corpora enhancing the quality of search results

Cross-references

▶ Annotation-Based Image Retrieval
▶ Boosting
▶ Content-based Image Retrieval (CBIR)
▶ Image Database
▶ Image Retrieval and Relevance Feedback
▶ Object Detection and Recognition
▶ Object Recognition
▶ Support Vector Machine
▶ Visual Content Analysis

Recommended Reading

1. Amores J., Sebe N., and Radeva P. Context-based object-class recognition and retrieval by generalized correlograms. IEEE Trans. Pattern Anal. Mach. Intell., 29(10):1818–1833, 2007.
2. Barnard K., Duygulu P., Forsyth D., de Freitas N., Blei D.M., and Jordan M.I. Matching words and pictures. J. Mach. Learn. Res., 3:1107–1135, 2003.
3. Boujemaa N., Fauqueur J., Ferecatu M., Fleuret F., Gouet V., Le Saux B., and Sahbi H. Ikona: interactive specific and generic image retrieval. In Proc. Int. Workshop on Multimedia Content-Based Indexing and Retrieval, 2001. Available at: http://www-rocg.inria.fr/imedia/mmcbirzod.html.
4. Boujemaa N., Fauqueur J., and Gouet V. What's beyond query by example? Technical report, INRIA, 2003.
5. Datta R., Li J., and Wang J.Z. Content-based image retrieval – approaches and trends of the new age. In Proc. 7th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, 2005, pp. 253–262.
6. Enser P.G.B., Sandom C.J., and Lewis P.H. Automatic annotation of images from the practitioner perspective. In Proc. 4th Int. Conf. Image and Video Retrieval, 2005, pp. 497–506.
7. Hanjalic A., Sebe N., and Chang E. Multimedia content analysis, management and retrieval: trends and challenges. In Proc. SPIE: Multimedia Content Analysis, Management, and Retrieval, 2006.
8. Hare J.S., Lewis P.H., Enser P.G.B., and Sandom C.J. Mind the gap: another look at the problem of the semantic gap in image retrieval. In Proc. SPIE: Multimedia Content Analysis, Management, and Retrieval, 2006.
9. Hervé N. and Boujemaa N. Image annotation: which approach for realistic databases? In Proc. 6th ACM Int. Conf. Image and Video Retrieval, 2007, pp. 170–177.
10. Lew M.S., Sebe N., Djeraba C., and Jain R. Content-based multimedia information retrieval: State of the art and Challenges. ACM Trans. Multimedia Comp., Comm., and Appl., 2(1):1–19, 2006.
11. Opelt A., Pinz A., Fussenegger M., and Auer P. Generic object recognition with boosting. Pattern Anal. Mach. Intell., 28 (3):416–431, 2006.
12. Ponce J., Hebert M., Schmid C., and Zisserman A (eds.). Toward category-level object recognition. Springer-Verlag Lecture Notes in Computer Science, 2006.
13. Smeulders A.W.M., Worring M., Santini S., Gupta A., and Jain R. Content-based image retrieval at the end of the early years. IEEE Trans. Pattern Anal. Mach. Intell., 22(12):1349–1380, 2000.
14. Szummer M. and Picard R.W. Indoor-outdoor image classification. In Proc. Workshop on Content-based Access to Image and Video Databases, Bombay, 1998.
15. Vailaya A., Jain A., and Zhang H-J. On image classification: city images vs. landscapes. Pattern Recognit. J., 31(12):1921–1935, 1998.
16. Zhang J., Marszalek M., Lazebnik S., and Schmid C. Local features and kernels for classification of texture and object categories: a comprehensive study. Int. J. Comput. Vis., 73(2):213–238, 2007.

Automatic Induction

▶ Grammar Inference

Automatic Language Induction

▶ Grammar Inference

Automatic Wrapper Induction

▶ Fully Automatic Web Data Extraction

Autonomic Database Replica Allocation

▶ Autonomous Replication

Autonomic Database Systems

▶ Self-Management Technology in Databases

Autonomic Query Processing

▶ Adaptive Query Processing

Autonomous Message Queuing Systems

▶ Adaptive Middleware for Message Queuing Systems

Autonomous Replication

Cristiana Amza
University of Toronto, Toronto, ON, Canada

Synonyms

Database provisioning; Adaptive database replication; Autonomic database replica allocation

Definition

Autonomous database replication refers to dynamic allocation of servers to applications in shared server clusters, in such a way to meet per-application performance requirements. Autonomous database replication enables the service provider to efficiently multiplex data center resources across applications in order to save per-server costs related to human management, power and cooling.

Historical Background

The concept of Autonomic Computing and the associated research area of automated, adaptive self-management in data centers was introduced by IBM as a grand-challenge project in the early 2000s. Other companies, which have responded or have had similar proposals of their own include Microsoft, Intel, Sun and HP. Related industry efforts in this area have been on developing open standards for resource monitoring tools e.g., as available on IBM's Alphaworks, (http://www.alphaworks.ibm.com.) and academic or collaborative industry-academia efforts related to applying machine learning techniques for automated adaptation in cluster systems [6,8,13].

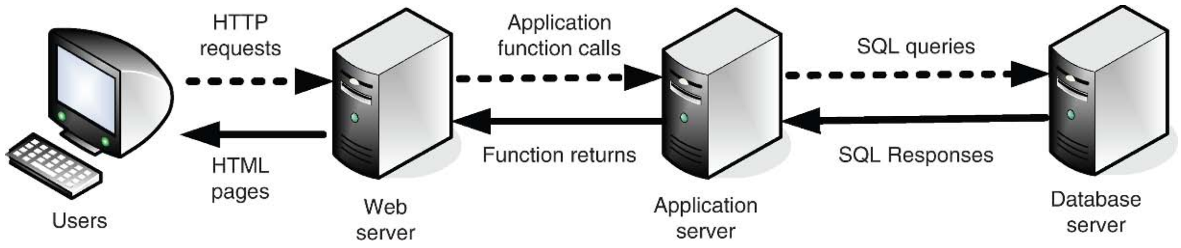
Foundations

Dynamic content servers, such as Amazon.com and eBay.com, commonly use a three-tier architecture (see Fig. 1) that consists of a front-end web server tier, an application server tier that implements the business logic, and a back-end database tier that stores the dynamic content of the site.

Large data centers may host multiple applications concurrently, such as e-commerce, auctions, news and games. The cooling and power costs of gross hardware over-provisioning for each application's estimated peak load are making efficient resource usage crucial. Furthermore, the excessive personnel costs involved in server management motivate an automated approach to resource allocation for applications in large sites.

Dynamic resource allocation techniques, i.e., dynamic provisioning of servers to multiple applications in each tier of the dynamic content site, have been recently introduced to address the increasing costs of ownership for large dynamic content server clusters. These automatic solutions add servers to an application's allocation based on perceived or predicted performance bottlenecks caused by either load spikes or component failures; they remove resources from a application's allocation when in underload.

If services experience daily patterns with peak loads for each service type at a different time (e.g., daytime for e-commerce, evening for auctions, morning for news sites, night for gaming), there are opportunities for re-assigning hardware resources from one service



Autonomous Replication. Figure 1. Three-tier architecture.

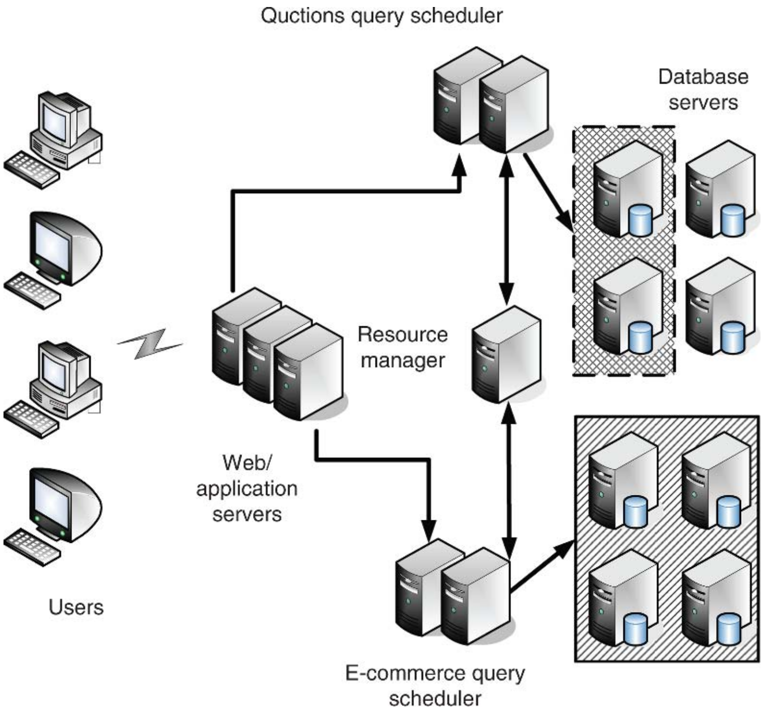
to another. Thus, instead of gross hardware overprovisioning for each application's estimated peak load, dynamic resource provisioning techniques enable the service provider to efficiently multiplex data center resources across applications.

Common Architecture for Database Replica Provisioning

Figure 2 shows the common architecture of sites with dynamic provisioning in the database server tier. A resource manager makes the replica allocation decisions for each application hosted on the site based on the application requirements and the current system state. The requirements are expressed in terms of a service level agreement (SLA) that consists of a latency requirement on the application's queries. The current system state includes the current performance of this application and the system capacity. The resource manager operates in two modes, underload and overload. During underload, the number of replicas exceeds overall demand and allocation decisions per application are made independently. During overload, the manager uses either a utility-based scheme, e.g., profit-based, or a fairness scheme, e.g., equal share, to allocate replicas to applications.

The allocation decisions are communicated to a set of schedulers, one per application, interposed between the application and the database tiers. Each scheduler fulfills the following functions. (i) It keeps track of the current *database set* allocated to its application and allocates or removes replicas from its managed set according to the resource manager's decisions. (ii) It distributes the corresponding incoming requests onto the respective database replicas. (iii) It periodically samples various system and application metrics, e.g., the average application latency from its *database set* in order to perceive or predict resource bottlenecks. (iv) It provides consistent replication, e.g., one-copy serializability [5], at all the replicas allocated to the application it manages.

Strong consistency is desirable for dynamic provisioning in the database tier of a multi-tier data center, for transparency reasons; the replicated nature of the database back-end and its configuration adaptations are thereby hidden from the application server, which interacts with the replicated back-end as with a single database. Any eager replication scheme with strong consistency guarantees, such as, 1-copy-serializability or 1-copy-snapshot-isolation



Autonomous Replication. Figure 2. Cluster architecture.

[5] can be used within each application's replica set. However, existing dynamic provisioning schemes typically leverage the presence of the scheduler and the synchronous, one query at a time, nature of the communication between the application and database tiers in a data center to implement a middleware replication solution in the scheduler itself. Upon receiving a query from the application server, the scheduler sends the query using a read-one, write-all replication scheme to the replica set allocated to the application. The scheduler assigns a global serialization order to all transactions pertaining to its application, e.g., based on conservatively-perceived table-level conflicts between transactions, and ensures that transactions execute in this order at all the corresponding database replicas. Table-level concurrency control affords optimizations based on conflict awareness in the scheduler [2,3], which offset any penalties due to the coarse-grain control for read-intensive e-commerce applications [11].

The scheduler is also in charge of bringing a new replica up to date by a process called *data migration*, during which all missing updates are applied on that replica. Integrating a stale replica into an application's allocation occurs through an on-line reconfiguration technique [10,11] without stalling on-going transactions on already active replicas for that application. Finally, each scheduler may itself be replicated for availability [2,3].

Overview of Dynamic Provisioning Solutions

Several fully-transparent provisioning solutions [4,9,12] have been recently introduced to address the increasing cost of management problem. Many of these approaches [4,9,12] investigate dynamic provisioning of resources within the (mostly) stateless web server and application server tiers. This entry focuses on the dynamic resource allocation solutions within the stateful database tier, which commonly becomes the bottleneck [1].

Regardless of the tier it applies to, a dynamic provisioning solution can be either *proactive* or *reactive* depending on its capabilities for predicting performance bottlenecks in the dynamic content server. *Proactive* solutions use sophisticated system models for prediction, such as, queuing models [4], utility models [12], machine learning models [6,8,13] or marketplace approaches [7]. *Proactive* provisioning techniques use the system model predictions for triggering allocations in advance of expected need. This is especially important for tiers with a higher adaptation delay, such as the database tier. In contrast, *reactive* approaches do not use prediction, but rather detect and react to existing resource bottlenecks. They rely on predefined thresholds for application-level or system metrics, such as latency or CPU usage for triggering changes in application allocation.

Challenges for Database Replica Provisioning

Adapting to a bottleneck caused by either a workload spike or a failure, by allocating additional replicas to the application, poses a set of challenges, which is summarized next.

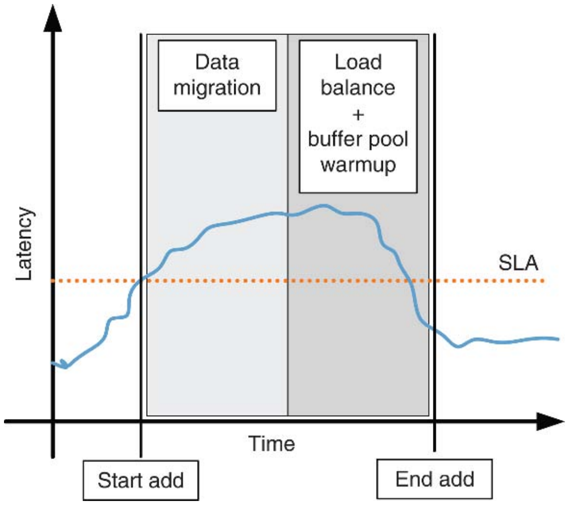
Adaptation Delay

Adding a database replica to an application's allocation is not an immediate process. The database state of the new replica(s) for that application will be stale and must be brought up-to-date, or a new instance of that application may need to be installed before it can be used. Furthermore, load balancing for the old and new replicas needs to occur and the buffer pool at the new replica(s) needs to be warm before the new replica(s) can be used effectively.

Oscillations in Allocation

Oscillations in database allocations to applications may occur during system instability induced by adaptations. As discussed earlier, the replica addition process can be long. *During* the adaptation phases, i.e., data migration, buffer pool warmup and load stabilization, the latency will remain high or may even temporarily continue to increase as shown in Figure 3. Latency sampling during this potentially long time is thus not necessarily reflective of a continued increase in load, but of system instability after an adaptation is triggered. If the system takes further decisions based on sampling latency during the stabilization time, it may continue to add further replicas which are unnecessary, hence will need to be removed later. This is an oscillation in allocation which carries performance penalties for other applications running on the system due to potential interference.

Both reactive and proactive policies that measure application-level metrics periodically, including during the replica addition process, can suffer from allocation instability. Allocation oscillations, in their turn, cause cross-application interference due to the



Autonomous Replication. Figure 3. Latency instability during replica addition.

price paid for warming up the buffer pool as part of the "context-switch" between applications on the machines involved.

While rapid load fluctuations may induce similar behavior, simple smoothing or filtering techniques can offer some protection to very brief load spikes. All existing dynamic provisioning schemes use some form of smoothing or filtering, to dampen brief load fluctuations.

Accurate and Lightweight Modeling

As previously mentioned proactive provisioning is desirable in the database back-end tier due to the higher adaptation delay inherent in this tier. The challenge for proactive database tier provisioning lies in accurately modeling the behavior of a replicated database back-end tier. Many factors, such as the wide range of query execution times typical in e-commerce workloads, the load balancing policy, caching effects, the replica consistency maintenance algorithm, etc. may influence performance. The derivation of a classic analytical model taking into account even a subset of these factors can be time consuming, hence unsuitable for on-line modeling and adaptation. Therefore, typically, pro-active provisioning schemes use machine learning approaches, such as K-Nearest-Neighbors (KNN) [6], or Support Vector Machine Regression (SVM) [8] to estimate performance models used for on-line adaptation. KNN-based approaches, or similar table-driven provisioning approaches [14] use off-line measurements and simple interpolation of response time values for various database server configurations and workloads. SVM-based approaches collect similar measurements[8] on-line to derive a regression function approximating the performance model dynamically.

Design Choices and Trade-offs for Database Provisioning

In designing a dynamic provisioning solution for the database back-end, it is necessary to consider the design trade-off between allocating replicas to applications in a *disjoint* versus *overlapping* manner. Consider the case where a *disjoint* set of machines is allocated to host the replicas of each application and dynamically adjusted to each application's cluster partition. When an application requires an additional replica, it must use an unallocated machine or a machine allocated to another application. In either case, the full *adaptation delay* for replica addition is incurred to the application.

Replica addition delay can be avoided altogether with *fully-overlapped* replicas, where all the database applications are replicated across all the available cluster machines. In this case, there is no replica addition delay because replicas do not have to be added or removed. However, this approach causes interference due to resource sharing. For example, when multiple database applications run on the same machine their performance can degrade due to buffer pool interference. This discussion shows that there is a trade-off between using disjoint and fully-overlapped replica allocation strategies. Disjoint allocation reduces interference and thus improves steady-state performance. Fully-overlapped allocation avoids replica addition delay and thus can speed up the system's response to load spikes and failures.

A compromise solution is to use a *partial overlap* strategy [11], where the application allocations are disjoint, but, for each application, batched updates are periodically executed on a set of database machines outside of that application's allocation, thus keeping them partially up-to-date.

Key Applications

Autonomic database replication is used for dynamic resource allocation in the database back-end of dynamic content web sites hosting e-commerce applications. Write queries in dynamic content applications are typically more lightweight and have a much lower memory footprint compared to read queries [2]. For instance, in e-commerce applications, an update query typically updates only the record pertaining to a particular customer or product, while read queries caused by browsing involve expensive database joins as a result of complex search criteria. Moreover, read queries are much more frequent than write queries. Hence, a *partial overlap* replication solution, which causes minimal resource interference, is typically used in order to reduce the adaptation delay [11].

Cross-references

▶ Database Replication
▶ Replication for Scalability

Recommended Reading

1. Amza C., Chanda A., Cox A.L., Elnikety S., Gil R., Rajamani K., Zwaenepoel W., Cecchet E., and Marguerite J. Specification and implementation of dynamic web site benchmarks. In Proc. 5th IEEE Workshop on Workload Characterization, 2002, pp. 3–13.
2. Amza C., Cox A.L., and Zwaenepoel W. Conflict-aware scheduling for dynamic content applications. In Proc. 4th USENIX Symp. on Internet Tech. and Syst., 2003, pp. 6–6.
3. Amza C., Cox A.L., and Zwaenepoel W. Distributed versioning: consistent replication for scaling back-end databases of dynamic content web sites. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2003, pp. 282–304.
4. Bennani M.N. and Menasce D.A. Resource allocation for autonomic data centers using analytic performance models. In Proc. 2nd Int. Conf. on Autonomic Computing, 2005, pp. 229–240.
5. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, Massachusetts, 1987.
6. Chen J., Soundararajan G., and Amza C. Autonomic provisioning of backend databases in dynamic content web servers. In Proc. 3rd Int. Conf. on Autonomic Computing, 2006, pp. 123–133.
7. Coleman K., Norris J., Candea G., and Fox A. Oncall: defeating spikes with a free-market server cluster. In Proc. 1st Int. Conf. on Autonomic Computing, 2004, pp. 198–205.
8. Ghanbari S., Soundararajan G., Chen J., and Amza C. Adaptive learning of metric correlations for temperature-aware database provisioning. In Proc. 4th Int. Conf. on Autonomic Computing, 2007, pp. 26.
9. IBM Corporation. Automated provisioning of resources for data center environments. http://www-306.ibm.com/software/tivoli/solutions/provisioning/, 2003.
10. Liang W. and Kemme B. Online recovery in cluster databases. In Advances in Database Technology, Proc. 11th Int. Conf. on Extending Database Technology, 2008, pp. 121–132.
11. Soundararajan G. and Amza C. Reactive provisioning of backend databases in shared dynamic content server clusters. ACM Trans. Auton. Adapt. Syst, 1(2):151–188, 2006.
12. Tesauro G., Das R., Walsh W.E., and Kephart J.O. Utility-function-driven resource allocation in autonomic systems. In Proc. 2nd Int. Conf. on Autonomic Computing, 2005, pp. 342–343.
13. Tesauro G., Jong N.K., Das R., and Bennani M.N. On the use of hybrid reinforcement learning for autonomic resource allocation. Cluster Computing, 10(3):287–299, 2007.
14. Walsh W.E., Tesauro G., Kephart J.O., and Das R. Utility functions in autonomic systems. In Proc. 1st Int. Conf. on Autonomic Computing, 2004, pp. 70–77.

Autonomous Message-oriented Middleware

▶ Adaptive Middleware for Message Queuing Systems

Average Precision

Ethan Zhang[1,2], Yi Zhang[1]
[1]University of California, Santa Cruz, CA, USA
[2]Yahoo! Inc., Santa Clara, CA, USA

Definition

Average precision is a measure that combines recall and precision for ranked retrieval results. For one information need, the average precision is the mean of the precision scores after each relevant document is retrieved.

$$Average\ Precision = \frac{\sum_r P@r}{R}$$

where r is the rank of each relevant document, R is the total number of relevant documents, and $P@r$ is the precision of the top-r retrieved documents.

Key Points

The average precision is very sensitive to the ranking of retrieval results. The relevant documents that are ranked higher contribute more to the average than the relevant documents that are ranked lower. Changes to the ranking of relevant documents have a significant impact on the average precision score. Average precision is considered a reasonable evaluation measure for emphasizing returning more relevant documents earlier.

Cross-references

▶ MAP

▶ Mean Average Precision

▶ Precision

▶ Precision at n

▶ P@n

▶ Recall

▶ Standard Effectiveness Measures

Average Precision at n

Nick Craswell, Stephen Robertson
Microsoft Research Cambridge, Cambridge, UK

Synonyms

AP@n

Definition

Average Precision at n is a variant of Average Precision (AP) where only the top n ranked documents are considered (please see the entry on Average Precision for its definition). AP is already a top-heavy measure, but has a recall component because it is normalized according to R, the number of relevant documents for a query. In AP@n there are a number of options for normalization, for example, normalize by n or normalize by min(n,R).

Key Points

The well-known measure Average Precision has a number of lesser-known variants, used in TREC [3] and elsewhere. Before and during TREC-1, it was usual to calculate an 11-point interpolated Precision-Recall curve, and take the average of these 11 precision values, giving an "interpolated AP." In TREC-2 and beyond, the modern non-interpolated AP was introduced. It calculates precision at each relevant document.

A number of other AP variants arise in a precision-oriented setting, where it is possible to calculate Average Precision at n. AP@n takes into account both the number of relevant documents in the top n and the positions of those documents. This is in contrast to Precision at n (P@n), which ignores position. It is defined as:

$$AP@n = \sum_{i=1}^{n} \frac{rel(i) \times P@i}{NF}$$

where rel(i) = 1 if the ith retrieved document is relevant and rel(i) = 0 otherwise, and NF is the normalization factor. In Average Precision it is usual to normalize by the number of relevant documents, i.e., NF = R. In a precision-oriented setting, this presents two problems. First, in precision-oriented evaluation one may not have judged enough documents to know R, or estimate it accurately. Second, if R is greater than n, a ceiling of n/R is imposed on the measure. For example, if one knows R = 100 relevant documents, then the best possible top-20, containing 20 relevant documents, will score an AP@20 of 0.2.

Three alternate normalization factors (NF) have been considered, only one of which has been used in TREC. Here, r is the number of relevant documents retrieved and n is the number of documents retrieved.

- *Normalize by r:* Baeza-Yates and Ribeiro-Neto [1] includes this variant, calling it "Average Precision at Seen Relevant Documents." This measure has the property that it may decrease when a relevant document is promoted into the top-n, because this increases the normalization factor. This seems counter-intuitive.
- *Normalize by n:* This variant was introduced for use in Web search evaluation [2]. In cases where R is less than n, this variant has a ceiling of less than 1. The scale of the measure is 0 ≤ AP@n ≤ min (R/n,1). It may be considered undesirable that the scale of the measure varies from query to query, for example when measuring the mean.
- *Normalize by min(n,R):* This is the "modified average precision" from the TREC-7 Very Large Collection track [3]. The ceiling is always 1 for this variant.

The normalization NF = min(n,R) allows AP@n scores to have the full range of 0.1, while retaining the property that promoting a relevant document always increases the score.

The arithmetic mean of AP@n over a set of queries can be called Mean Average Precision at n (MAP@n), in the same way that Average Precision relates to Mean Average Precision.

Cross-references

▶ Average Precision
▶ Mean Average Precision
▶ Precision at n
▶ Precision-Oriented Effectiveness Measures

Recommended Reading

1. Baeza-Yates R.A. and Ribeiro-Neto B. Modern Information Retrieval. Addison-Wesley, Reading, MA, 1999.
2. Hawking D., Craswell N., Bailey P., and Griffiths K. Measuring search engine quality. Inf. Retr., 4(1):33–59, 2001.
3. Voorhees E.M. and Harman D.K. TREC: Experiment and Evaluation in Information Retrieval. MIT Press, Cambridge, MA, 2005.

Average Precision Histogram

Steven M. Beitzel[1], Eric C. Jensen[2], Ophir Frieder[3]
[1]Telcordia Technologies, Piscataway, NJ, USA
[2]Twitter, Inc., San Fransisco, CA, USA
[3]Georgetown University, Washington, DC, USA

Definition

The average precision histogram plots the performance of a single run produced by an information retrieval system on a per-query basis. Each data point on the abscissa represents a query used in the evaluation process. The corresponding points on the ordinate measures the difference in this run's performance on the given topic relative to the median average precision of other runs on that topic.

Key Points

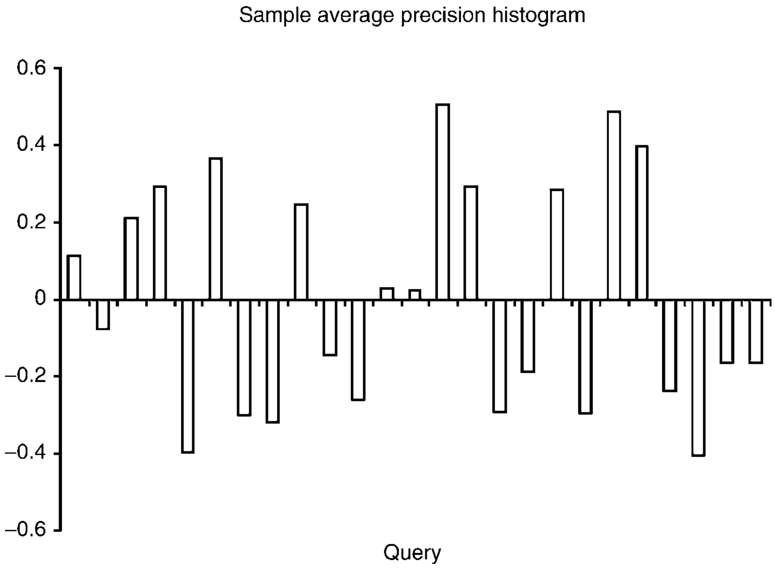
Average precision histograms are often used to illustrate the performance differences that an information retrieval system may exhibit across different queries in an evaluation set in relation to other systems or runs that are evaluated on the same set of queries. These histograms can be used to identify the queries in an evaluation set which pose the most difficulty for information retrieval systems. Various tracks in the NIST Text Retrieval Conference (TREC) have used average precision histograms in the post-competition analysis of submitted runs [1]. An example average precision histogram for 25 queries is shown in Fig. 1.

Cross-references

▶ Average Precision
▶ Effectiveness Involving Multiple Queries
▶ Geometric Mean Average Precision
▶ Mean Average Precision

Recommended Reading

1. National Institute of Standards and Technology. TREC-2003 Common Evaluation Metrics. Available online at: http://trec.nist.gov/pubs/trec12/appendices/measures.ps (retrieved on August 27, 2007), 2003.



Average Precision Histogram. Figure 1. Example average precision histogram.

Average R-Precision

Steven M. Beitzel[1], Eric C. Jensen[2], Ophir Frieder[3]
[1]Telcordia Technologies, Piscataway, NJ, USA
[2]Twitter, Inc., San Fransisco, CA, USA
[3]Georgetown University, Washington, DC, USA

Definition

The Average R-precision is the arithmetic mean of the R-precision values for an information retrieval system over a set of n query topics. It can be expressed as follows:

$$ARP = \frac{1}{n}\sum_{n} RP_n$$

where RP represents the R-Precision value for a given topic from the evaluation set of n topics. R-Precision is defined as the precision after R documents have been retrieved by the system, where R is also the total number of judged relevant documents for the given topic. Precision is defined as the portion of retrieved documents that are truly relevant to the given query topic.

Key Points

R-precision places lower emphasis on the exact ranking of the relevant documents returned by an information retrieval system. This can be useful when a topic has a large number of judged relevant documents, or when an evaluator is more interested in measuring aggregate performance as opposed to the fine-grained quality of the ranking provided by the system.

As an example, consider two query topics: topic A has ten relevant documents, and topic B has six relevant documents. Suppose further that an information retrieval system returns five relevant documents in the top ten retrieved for topic A, two relevant documents in the top six retrieved for topic B. For this case, Average R-precision for this run would be:

$$ARP = \frac{\frac{5}{10} + \frac{2}{6}}{2} \approx 0.4167$$

Cross-references

▶ Effectiveness Involving Multiple Queries
▶ Mean Average Precision
▶ R-Precision

Recommended Reading

1. National Institute of Standards and Technology. TREC-2004 Common Evaluation Measures. Available online at: http://trec.nist.gov/pubs/trec14/appendices/CE.MEASURES05.pdf (retrieved on August 27, 2007), 2005.

AXML

▶ Active XML

B+-Tree

Donghui Zhang[1], Kenneth Paul Baclawski[1],
Vassilis J. Tsotras[2]
[1]Northeastern University, Boston, MA, USA
[2]University of California-Riverside, Riverside,
CA, USA

Synonyms

B-tree

Definition

The B+-tree is a disk-based, paginated, dynamically updateable, balanced, and tree-like index structure. It supports the exact match query as well as insertion/deletion operations in $O(\log_p n)$ I/Os, where n is the number of records in the tree and p is the page capacity in number of records. It also supports the range searches in $O(\log_p n + t/p)$ I/Os, where t is the number of records in the query result.

Historical Background

The binary search tree is a well-known data structure. When the data volume is so large that the tree does not fit in main memory, a disk-based search tree is necessary. The most commonly used disk-based search trees are the B-tree and its variations. Originally invented by Bayer and McCreight [2], the B-tree may be regarded as an extension of the balanced binary tree, since a B-tree is always balanced (i.e., all leaf nodes are on the same level). Since each disk access retrieves or updates an entire block of information between memory and disk rather than a few bytes, a node of the B-tree is expanded to hold more than two child pointers, up to the block capacity. To guarantee worst-case performance, the B-tree requires that every node (except the root) has to be at least half full. Because of this requirement, an exact match query, insertion or deletion operation must access at most $O(\log_p n)$ nodes, where p is the page capacity in number of child pointers, and n is the number of objects. The most popular variation of the B-tree is the B+-tree [3,4]. In a B+-tree, objects are stored only at the leaf level, and the leaf nodes are organized into a double linked list. As such, the B+-tree can be seen as an extension of the Indexed Sequential Access Method (ISAM), a static (and thus possibly unbalanced if updates take place) disk-based search tree proposed by IBM in the mid 1960's.

Foundations

Structure

The B+-tree is a tree structure where every node corresponds to a disk block and which satisfies the following properties:

- The tree is balanced, i.e., every leaf node has the same depth.
- An internal node stores a list of keys and a list of pointers. The number of pointers is one more than the number of keys. Every node corresponds to a key range. The key range of an internal node with k keys is partitioned into k+1 sub-ranges, one for each child node. For instance, suppose that the root node has exactly two keys, 100 and 200. The key range of the root node is divided into three sub-ranges $(-\infty, 100)$, $(100, 200)$ and $(200, +\infty)$. Note that a key in an internal node does not need to occur as the key of any leaf record. Such a key serves only as a means of defining a sub-range.
- A leaf node stores a list of records, each having a key and some value.
- Every node except the root node is at least half full. For example suppose that an internal node can hold up to p child pointers (and p-1 keys, of course) and a leaf node can hold up to r records. The half full requirement says any internal node (except the root) must contain at least $\lceil p/2 \rceil$ child pointers and any leaf node (except the root) must contain at least $\lceil r/2 \rceil$ records.
- If the root node is an internal node, it must have at least two child pointers.

- All the leaf nodes are organized, in increasing key order, into a double linked list.

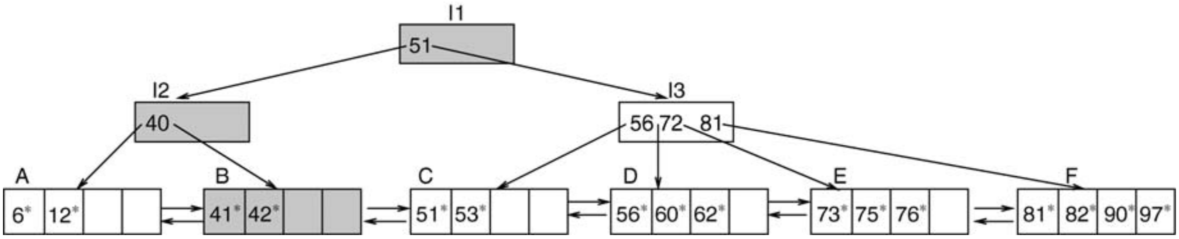
An example B+-tree is given in Fig. 1. It is assumed that every node has between two and four entries. In a leaf node, an entry is simply a record. In an internal node, an entry is a pair of (key, child pointer), where the key for the first entry is NULL. To differentiate a leaf entry (which corresponds to an actual record) from a key in an index entry, each leaf entry is followed by a "*".

Query Processing

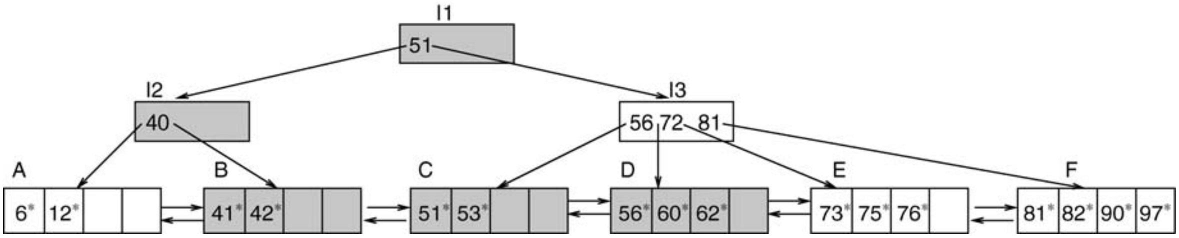
The B+-tree efficiently supports not only *exact-match queries*, which find the record with a given key, but also *range queries*, which find the records whose keys are in a given range. To perform an exact-match query, the B+-tree follows a single path from the root to a leaf. In the root node, there is a single child pointer whose key range contains the specified key. If one follows the child pointer to the corresponding child node, inside the child node there is also a single child pointer whose key range contains the desired key. Eventually, one reaches a leaf node. The desired record, if it exists, must be located in this node. As an example, Fig. 1 shows the search path if one searches for the record with key = 41. Besides exact-match queries, the B+-tree also supports range queries. That is, one can efficiently find all records whose keys belong to a range *R*. In order to do so, all the leaf nodes of a B+-tree are linked together. To search for all records whose keys are in the range *R* = [*low*, *high*], one performs an exact match query for key = *low*. This leads to a leaf node. One examines all records in this leaf node, and then follows the sibling link to the next leaf node, and so on. The algorithm stops when a record with key > *high* is encountered. An example is shown in Fig. 2.

Insertion

To insert a new record, the B+-tree first performs an exact-match query to locate the leaf node where the record should be stored, then the record is stored in the leaf node if there is enough space available. If there is not enough space, the leaf node is split. A new node is allocated, and half of the records, the ones with the larger keys in the overflowing node, are moved to the new node. A new index entry (the smallest key in the new node and a pointer to the new node) is inserted into the parent node. This may, in turn, cause



B+-Tree. Figure 1. Illustration of the B+-tree and exact-match query processing. To search for a record with key = 41, nodes I_1, I_2 and *B* are examined.



B+-Tree. Figure 2. Illustration of the range query algorithm in the B+-tree. To search for all records with keys in the range [41,60], the first step is to find the leaf node containing 41* (I_1, I_2 and *B* are examined). The second step is to follow the right-sibling pointers between leaf nodes and examine nodes *C* and *D*. The algorithm stops at *D* because a record with key >60 is found.

the parent node to overflow, and so on. In the worst case, all nodes along the insertion path are split. If the root node is split into two, a new root node is allocated and therefore the height of the tree increases by one.

As an example, Fig. 3 shows an intermediate result of inserting record 92* into Fig. 1. In particular, the example illustrates that splitting a leaf node results in a *"copy up"* operation. The result is intermediate because the parent node I_3 will also be split.

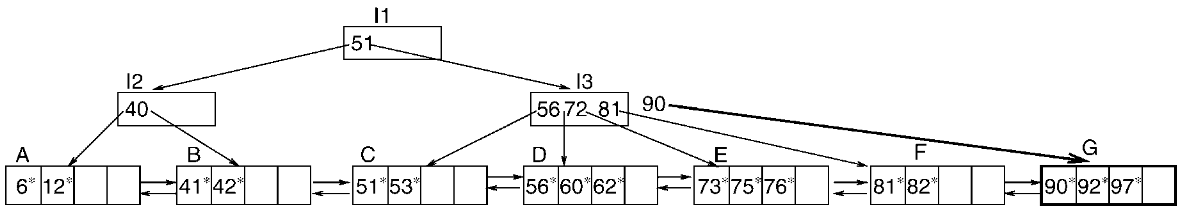
The complete result after inserting 92* is shown in Fig. 4. Here the overflowing internal node I_3 is split. In particular, the example illustrates that splitting an internal node can result in a *"push up"* operation.

Deletion

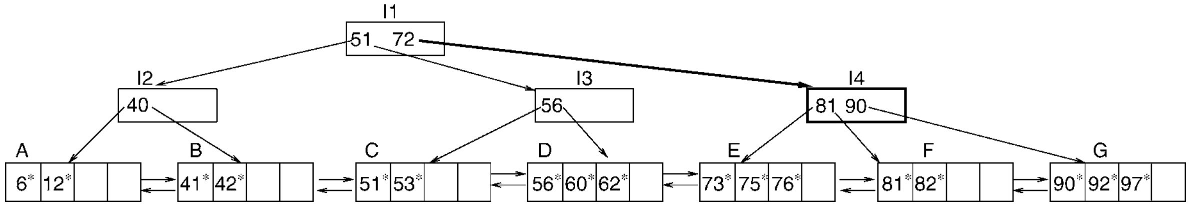
To delete a record from the B+-tree, one first uses the exact-match query algorithm to locate the leaf node that contains the record, and then the record is removed from the leaf node. If the node is at least half full, the algorithm finishes. Otherwise, the algorithm tries to re-distribute records between an immediate sibling node and the underflowing node. If redistribution is not possible, the underflowing node is merged with an immediate sibling node. Note that this merge is always possible.

As an example, Fig. 5 shows the intermediate result of deleting record 41* from the B+-tree shown in Fig. 4. Note that when merging two leaf nodes, a key in the parent node is *discarded*, which is the reverse operation of *copy up*. The result is intermediate because node I_2 is also underflowing.

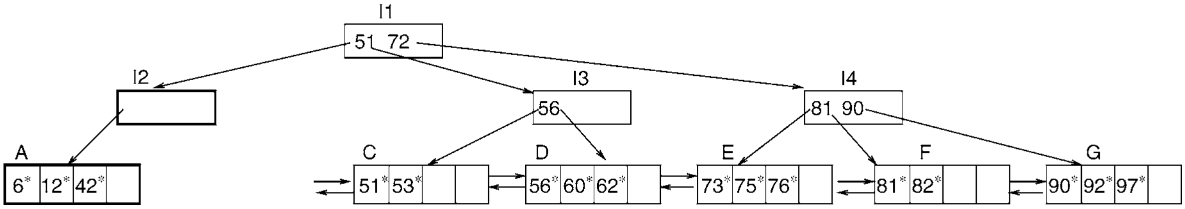
Figure 6 illustrates the final result of deleting 41*. The underflow of node I_2 is handled by merging I_2 with I_3. This merge causes key 51 from the parent node to be *dragged down* to I_2. It is the reverse operation of *push up*.



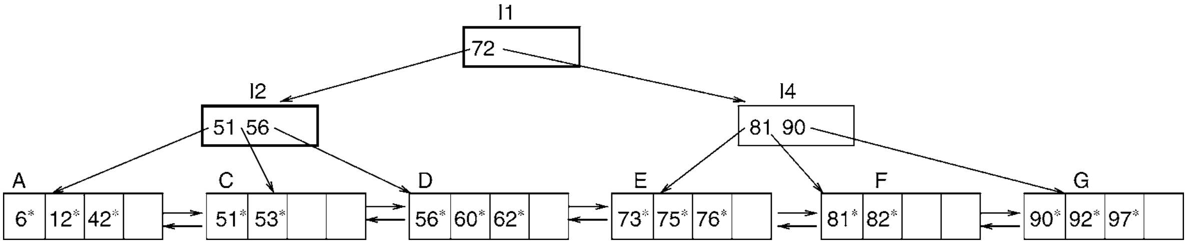
B+-Tree. Figure 3. Intermediate result of inserting record 92* into Fig. 1. The leaf node *F* is split. The smallest key in the new node *G*, which is 90, is *copied up* to the parent node.



B+-Tree. Figure 4. Continued from Fig. 3. Final result of inserting record 92*. The overflowing internal node I_3 is split. The middle key, 72, is *pushed up* to the parent node.



B+-Tree. Figure 5. Intermediate result of deleting 41* from Fig. 4. Node *B* is merged with node *A*. Key 40 (as well as the pointer to node *B*) is *discarded* from the parent node.



B+-Tree. Figure 6. Continued from Fig. 5. Final result of deleting 41*. Node I_2 is merged with node I_3. Key 51 from the parent node is *dragged down.*

Comparison with Some Other Index Structures

Compared with the B-tree, the B+-tree stores all records at the leaf level, and organizes the leaf nodes into a double linked list. This enables efficient range queries. Compared with ISAM, the B+-tree is a fully dynamic structure that balances itself nicely as records are inserted or deleted, without the need for overflow pages. Compared with external hashing schemes such as Linear Hashing and Extendible Hashing, the B+-tree can guarantee logarithmic query cost in the worst case (while hashing schemes have linear worst-case cost, although this is very unlikely), and can efficiently support range queries (whereas hashing schemes do not support range queries).

Key Applications

The B+-tree index has been implemented in most, if not all, relational database management systems such as Oracle, Microsoft SQL Server, IBM DB2, Informix, Sybase, and MySQL. Further, it is implemented in many filesystems including the NTFS filesystem (for Microsoft Windows), ReiserFS filesystem (for Unix and Linux), XFS filesystem (for IRIX and Linux), and the JFS2 filesystem (for AIX, OS/2 and Linux).

Future Directions

The impact of the B+-tree index is very significant. Many disk-based index structures, such as the R-tree [4] and k-d-B-tree [5] or their variants, are extensions of the B+-tree. Concurrency in B+-trees was studied in [6]. The Universal B-tree, which extends the B+-tree to index multi-dimensional objects, was studied in [1].

Cross-references

- ▶ Extendible Hashing
- ▶ Index Sequential Access Method (ISAM)
- ▶ k-d-B-Tree
- ▶ Linear Hashing
- ▶ Rtree
- ▶ Tree-based Indexing

Recommended Reading

1. Bayer R. The universal B-tree for multidimensional indexing: general concepts. In Proc. Int. Conf. on Worldwide Computing and Its Applications (WWCA), 1997, pp. 198–209.
2. Bayer R. and McCreight E.M. Organization and maintenance of large ordered indices. Acta Inf., 1, 1972.
3. Comer D. The ubiquitous B-tree. ACM Comput. Surv., 11(2), 1979.
4. Knuth D. The Art of Computer Programming, Vol. 3: Sorting and Searching. Addison Wesley, MA, USA, 1973.
5. Robinson J. The K-D-B tree: a search structure for large multidimensional dynamic indexes. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1981, pp. 10–18.
6. Srinivasan V. and Carey M.J. Performance of B+ tree concurrency algorithms. VLDB J., 2(4):361–406, 1993.
7. Theodoridis Y. The R-tree-portal. http://www.rtreeportal.org, 2003.

Backup

▶ Logging and Recovery

Backup and Restore

Kenichi Wada
Hitachi Limited, Tokyo, Japan

Synonyms

Backup copy

Definition

Backup is the action of collecting data stored on non-volatile storage media to aid recovery in case the original data are lost or becomes inaccessible due to corruption or damage caused by, mainly, a failure in storage component hardware (such as a hard disk drive or controller), a disastrous event, an operator's mistake, or intentional alteration or erasure of the data. In addition, the data collected by this action are called a backup. Restore is the action of copying a backup to on-line storage for use by applications in a recovery. The term "back up and restore" indicates both actions.

Key Points

The purpose of backup includes recovering data from storage media failure. For this reason, it is common to make backup to different storage media. Removable storage media is used because it is relatively less expensive than online storage media like hard disk drive, and it is easy to bring it offsite in case of site failure. Hard disk drives are becoming popular as backup storage media, because of its decreasing bit cost and quick restore capability.

There are many ways to take backup. A full backup is the way to make a full copy of data. It requires equal capacity of original data, unless data compression is not used. If full backup is taken daily from Monday to Friday and they are kept for the following week, five times capacity is required for backup.

The following two kinds of backup are used to reduce the size of backup data:

Incremental backup: A backup that copies all data modified since the last full backup. Modified data can be copied many times until next full backup is taken. To restore data when incremental backups are in use, the latest full backup and the latest incremental backup are required.

Differential backup: A backup that copies all data modified since the last backup. Modified data are not copied more than once. To restore data when differential backups are used, the latest full backup and all differential backups or all backups are required.

Cross-references

- ▶ Disaster Recovery
- ▶ Replication

Backup Copy

▶ Backup and Restore

Backup Mechanisms

▶ Replication for High Availability

Backward Recovery

▶ Crash Recovery

Bag Semantics

Todd J. Green
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

Duplicate Semantics; Multiset Semantics

Definition

In the ordinary relational model, relations are sets of tuples, which by definition do not contain "duplicate" entries. However, RDBMSs typically implement a variation of this model where relations are *bags* (or *multisets*) of tuples, with duplicates allowed. Formally, a bag is a mapping of tuples to natural number *multiplicities*; a set can be viewed as a special case of a bag where all tuple multiplicities are 0 or 1. The operations of the relational algebra are extended to operate on bags by defining their action on tuple multiplicities. RDBMSs based on bags rather than sets are said to implement *bag semantics* (rather than *set semantics*). Duplicates may occur at multiple levels: in source relations, in materialized views, or in query answers. A variation of bag semantics called *bag-set semantics* is obtained by requiring source relations to be sets, while allowing views and query answers to contain duplicates. Bag-set semantics represents a practical compromise between bag semantics and set semantics.

Historical Background

In the ordinary relational model, relations are sets of tuples, which by definition do not contain "duplicate" entries. However, RDBMS implementations, beginning with System R and INGRES, deviated from the "pure" relational model by allowing duplicate tuples in query answers, thus making bags (multisets), rather than sets, the primary collection type in query processing. The primary motivation for allowing duplicate tuples was the need to avoid expensive duplicate elimination in query processing. Instead, duplicate elimination would only be performed if explicitly requested by the user, e.g., via the SQL `DISTINCT` keyword. In their use of bag relations, RDBMS implementations were ahead of theory, and the precise semantics of relational algebra on bags was not studied until the 1980s [6,11,12]. Klug [12] pointed out the failure of certain algebraic identities under bag semantics, and also gave a precise semantics of aggregate functions that did not depend on bag semantics. Dayal et al. initiated the study of query optimization under bag semantics in [6]. Bag containment of unions of conjunctive queries (UCQs) was shown to be undecidable by Ioannidis and Ramakrishan [9], while bag equivalence of conjunctive queries (CQs) was shown to be decidable, in fact the same as isomorphism, by Chaudhuri and Vardi [3]. The $\Pi_2{}^p$-hardness of checking bag containment of CQs was also established in [3], but the decidability of the problem remains open (see Future Directions). The same paper introduced the terminology "bag-set semantics" to describe the semantics obtained by requiring source relations to be sets while allowing views and query answers to contain duplicates, and established the decidability of bag-set equivalence of CQs. For conjunctive queries with inequalities ($\mathrm{CQ}^{\neq}$), bag containment and bag-set containment were both shown to be undecidable by Jayram et al. [10]. Bag equivalence of UCQs was shown to be decidable (and like CQs, the same as isomorphism) by Cohen et al. [5]. Cohen [4] studied query equivalence for a generalization of bag-semantics called *combined semantics*, which captures user-specified elimination of duplicates at intermediate stages of query processing. A bag semantics for Datalog was proposed by Mumick et al. [14], and Mumick and Shmueli [15] studied computational problems related to infinite multiplicities (which may occur in query answers because of recursion). Going beyond the relational model, several papers have studied bag semantics in the context of query languages for nested relations; see [2,8] and references therein.

Foundations

Bag relational algebra. For simplicity, the definitions here assume the unnamed perspective [1] of the relational model. Fix a countable domain $\mathbb{D}$ of database values. A *bag relation* of arity k is a mapping R: $\mathbb{D}^k \to \mathbb{N}$ associating *tuples* with their *multiplicities.* Conceptually, a tuple not present in the bag relation is mapped to zero. It is typically required that bag relations have *finite support*, i.e., R is zero on all but finitely many tuples. A *duplicate tuple* is a tuple whose multiplicity is greater than 1. From the perspective of bag semantics, an ordinary set relation of arity k is a mapping $R : \mathbb{D}^k \to \{0,1\}$ (again of finite support), in other words, a bag relation with no duplicate tuples. The relational algebra can be extended to bag relations by defining the action of the relational operators on tuple multiplicities:

Selection. If R is a bag relation of arity k and the selection predicate P maps each k-tuple to either 0 or 1 then $\sigma_P R$ is the bag relation of arity k defined by

$$(\sigma_P R)(t) \overset{\text{def}}{=} R(t) \cdot P(t).$$

Projection. If R is a bag relation of arity k and $V = (v_1,...,v_n)$ is a list of indices, $1 \leq v_i \leq k$, then π_V R is a bag relation of arity n, defined by

$$(\pi_V R)(t) \overset{\text{def}}{=} \sum_{t'\text{s.t.}t'=\pi_V(t)} R(t').$$

Cross product. If R_1 is a bag relation of arity k_1 and R_2 is a bag relation of arity k_2, then $R_1 \times R_2$ is a bag relation of arity k_1+k_2, defined by

$$(R_1 \times R_2)(t) \overset{\text{def}}{=} R_1(t_1) \cdot R_2(t_2),$$

where t is a (k_1+k_2)-tuple obtained by concatenating t_1 and t_2.

Union. If R_1 and R_2 are bag relations of arity k, then $R_1 \cup R_2$ is a bag relation of arity k, defined by

$$(R_1 \cup R_2)(t) \overset{\text{def}}{=} R_1(t) + R_2(t).$$

Intersection. If R_1 and R_2 are bag relations of arity k, then $R_1 \cap R_2$ is a bag relation of arity k, defined by

$$(R_1 \cap R_2)(t) \overset{\text{def}}{=} \min(R_1(t), R_2(t)).$$

Difference. If R_1 and R_2 are bag relations of arity k, then $R_1 - R_2$ is a bag relation of arity k, defined by

$$(R_1 - R_2)(t) \overset{\text{def}}{=} \max(R_1(t) - R_2(t), 0),$$

in other words, the calculation uses *proper subtraction* of natural numbers.

Example 1.

$$R = \begin{array}{|cc|c|}\hline a & b & 2 \\ c & b & 3 \\ c & d & 1 \\ \hline\end{array}\,, \qquad S = \begin{array}{|cc|c|}\hline b & c & 5 \\ b & d & 1 \\ d & d & 2 \\ \hline\end{array}\,,$$

$$R \circ S = \begin{array}{|cc|c|}\hline a & c & 2 \cdot 5 = 10 \\ a & d & 2 \cdot 1 = 2 \\ c & c & 1 \cdot 5 = 5 \\ c & d & 3 \cdot 1 + 1 \cdot 2 = 5 \\ \hline\end{array}$$

where $R \circ S$ *denotes relational composition, i.e.,* $R \circ S \overset{\text{def}}{=} \pi_{1,4}\, \sigma_{2=3}(R \times S)$.

Example 2.

$$R = \begin{array}{|cc|c|}\hline a & b & 1 \\ c & b & 1 \\ c & d & 1 \\ \hline\end{array}\,, \qquad S = \begin{array}{|cc|c|}\hline b & c & 1 \\ b & d & 1 \\ d & d & 1 \\ \hline\end{array}\,,$$

$$\pi_{1,3}\ (R \times S) = \begin{array}{|cc|c|}\hline a & b & 1 \cdot 1 + 1 \cdot 1 = 2 \\ a & d & 1 \cdot 1 = 1 \\ c & b & 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 4 \\ c & d & 1 \cdot 1 + 1 \cdot 1 = 2 \\ \hline\end{array}$$

Note that the use of projection results in duplicate tuples in the query output, even though the source tables R and S are duplicate-free. Duplicates may also be introduced by the union operator.

Bags and SQL. Practical query languages such as SQL are based on bag semantics rather than set semantics, since eliminating duplicates typically requires an expensive sort. However, SQL provides a way to emulate set semantics by the use of an explicit *duplicate elimination* operator, which is specified using the `DISTINCT` keyword. This operator converts a bag relation into the corresponding set relation. For example, to execute the SQL query

```
SELECT DISTINCT R.A, S.C
FROM R, S
WHERE R.B = S.B
```

the RDBMS will compute the bag join of R and S, then eliminate duplicates to produce a set relation. The result is the same as the join of R and S under set semantics.

SQL also provides alternate versions of the union, intersection, and difference operators which differ in their handling of duplicates. In particular, `UNION ALL`, `INTERSECT ALL`, and `EXCEPT ALL` correspond to the bag operations as defined above, i.e., they retain duplicates and produce bag relations as output. Meanwhile, `UNION`, `INTERSECT`, and `EXCEPT` perform duplicate elimination and produce set relations as output. For `UNION`, duplicate elimination is performed after the bag union, while for `INTERSECT` and `EXCEPT`, the duplicate elimination is performed beforehand on the operands.

Query reformulation with bag semantics. The context of bag semantics poses unique challenges in query reformulation and optimization, as classical optimization techniques such as query minimization do not necessarily transfer to bag semantics. For example, under set semantics, the query $R \bowtie R$ can be minimized by removing the redundant self-join to produce the equivalent query R. However, under bag semantics, the resulting query is not equivalent to the original, as the "redundant" self-join may increase the multiplicity of some output tuples. The rest of this section summarizes some of the theoretical results which are known regarding containment and equivalence of queries under bag semantics. In studying query reformulation, it is convenient to assume queries are expressed in a Datalog-style syntax. Thus, the algebraic query $R \bowtie R$ corresponds to the CQ

$$Q(x, y){:}\ -R(x, y), R(x, y),$$

while the query R corresponds to the CQ

$$Q'(x, y){:}\ -R(x, y).$$

As the example illustrates, repetitions of an atom in the body of a CQ are significant. As with set semantics, the bag semantics of CQs is defined in terms of *valuations* which assign the variables in the CQ values from the domain $\mathbb{D}$. The multiplicity of an output tuple is computed by summing over all valuations which produce that output tuple, and for each valuation, taking the product of the source multiplicities of the tuples corresponding to the body. Thus, for a CQ

$$Q(\bar{x}){:}\ -A_1(\bar{z}_1), \ldots, A_n(\bar{z}_n),$$

the multiplicity of an output tuple t is given by

$$Q(t) \overset{\text{def}}{=} \sum_{v} \prod_{i=1}^{n} A_i(v(\bar{z}_i)), \tag{1}$$

where the sum is over valuations $v : \text{vars}(Q) \rightarrow \mathbb{D}$ such that $v(\bar{x}) = t$. Note that this definition agrees with the bag relational algebra definition for SPJ queries (using selection, projection, and cross product).

Example 3. *The relational algebra query from Example 1 corresponds to the CQ*

$$Q(x,z) : -R(x,y), S(y,z).$$

Applying Q to the bag relations R and S from Example 1 yields $Q(c,d) = 3 \cdot 1 + 1 \cdot 2 = 5$, *where the* $3 \cdot 1$ *is produced by the valuation v which sends* $x \mapsto c, y \mapsto b$, *and* $z \mapsto d$, *and the* $1 \cdot 2$ *is produced by the valuation* v' *which sends* $x \mapsto c, y \mapsto d$, *and* $z \mapsto d$.

For a UCQ $\bar{Q} = (Q_1,\ldots,Q_n)$ the multiplicity of an output tuple is computed by summing over all the CQs in $\bar{Q}$:

$$\bar{Q}(t) \overset{\text{def}}{=} \sum_i^n Q_i(t). \tag{2}$$

This definition agrees with the bag relational algebra definition for SPJU queries (using selection, projection, cross product, and union).

Example 4. *Applying the UCQ* Q' *defined by the CQs*

$$Q'(x,z): -R(x,y), S(y,z)$$

$$Q'(z,y): -R(x,y), S(x,z)$$

to the set relations R and S from Example 2, the multiplicity of (c,d) *in the output is* $1 \cdot 1 + 1 \cdot 1 = 2$, *with each of the CQs contributing a term to the sum.*

A query P is *bag-contained* in a query Q, denoted $P \sqsubseteq_b Q$, if for all bag instances I, for every output tuple t, $P(I)(t) \leq Q(I)(t)$. When $P \sqsubseteq_b Q$ and $Q \sqsubseteq_b P$, P and Q are said to be *bag-equivalent*, denoted $P \equiv_b Q$. The analogous notions of set-containment and set-equivalence are denoted $P \sqsubseteq_s Q$ and $P \equiv_s Q$, respectively. With set semantics, it is well-known that a CQ Q can be optimized by deleting atoms in the body of Q, producing a minimal *core* of Q which is set-equivalent to Q. However, under bag semantics, this optimization no longer works, as deleting an atom in the body of a CQ always produces an inequivalent query, in fact:

Theorem 1 ([3]). *For CQs* P, Q, $P \equiv_b Q$ *iff* $P \cong Q$.

Here, $P \cong Q$ denotes that P and Q are *isomorphic*. Two CQs P and Q are isomorphic if there is a (bijective) renaming of variables $\theta : \text{vars}(P) \rightarrow \text{vars}(Q)$ which transforms P into Q. Checking isomorphism of directed graphs (i.e., Boolean CQs over a single binary predicate), is known to be in *NP*, but is not known or believed to be either in *P* or *NP*-complete. This holds also for general relational structures (and arbitrary CQs). Thus, although there are fewer opportunities for optimization of CQs under bag semantics, checking bag-equivalence is presumably computationally easier than checking set-equivalence (which is known to be *NP*-complete). It turns out that Theorem 1 can be generalized to UCQs:

Theorem 2 ([5]). *For UCQs* $\bar{P}$, $\bar{Q}$, $\bar{P} \equiv_b \bar{Q}$ *iff* $\bar{P} \cong \bar{Q}$.

Bag-equivalence of unions of conjunctive queries with inequalities (UCQ$^{\neq}$s) is also known to be decidable:

Theorem 3 ([5]). For *UCQ$^{\neq}$ s* P, Q, *checking* $P \equiv_b Q$ *is in PSPACE.*

The same result was obtained earlier for CQ$^{\neq}$s by Nutt et al. [16]. The exact complexity in each case (CQ$^{\neq}$s or UCQ$^{\neq}$s) seems to be open. (Theorem 1 implies a graph isomorphism-hard lower bound).

In contrast to equivalence, which seems to only become easier to check for bag semantics than for set semantics, containment can become much harder. For CQs, the following lower bound on the complexity of bag containment is known:

Theorem 4 ([3]). For *CQs* P, Q, *checking* $P \sqsubseteq_b Q$ *is* $\Pi_2{}^P$-*hard.*

However, this is only a lower bound, and as of 2008, it is not known whether the problem is even decidable. For UCQs, the question has been resolved negatively:

Theorem 5 ([9]). For *UCQs* $\bar{P}, \bar{Q}$, *checking* $\bar{P} \sqsubseteq_b \bar{Q}$ *is undecidable.*

The reduction used to prove this result highlights a close connection between checking bag containment of UCQs and Hilbert's Tenth Problem, which concerns checking for the existence of solutions to Diophantine equations. Another (non-trivial) reduction from the same problem was used to establish a similar result for CQ$^{\neq}$s:

Theorem 6 ([10]). *For CQ$^{\neq}$ s* P, Q, *checking* $P \sqsubseteq_b Q$ *is undecidable.*

Query reformulation with bag-set semantics. Recall that with bag-set semantics, all source tuples are assumed to have cardinality 1 (or 0, if not present). For query optimization, the main ramification compared to bag semantics is that redundant (repeated) atoms in the body of a CQ are immaterial under bag-set semantics and can be simply deleted from the body, producing an *irredundant* CQ. For example, the CQs

$$Q_1(x,z){:}-R(x,y),S(y,z)$$

$$Q_2(x,z){:}-R(x,y),R(x,y),S(y,z)$$

are bag-set equivalent, denoted $Q_1 \equiv_{bs} Q_2$, and Q_1 is irredundant. The following result states that eliminating repeated atoms is essentially the only optimization possible for CQs under bag-set semantics:

Theorem 7 ([3]). *For irredundant CQs* P, Q, $P \equiv_{bs} Q$ *iff* $P \equiv_b Q$ *(hence* $P \equiv_{bs} Q$ *iff* $P \cong Q$*).*

This result was essentially a rediscovery of a well-known result in graph theory due to Lovász [13], who showed that for finite relational structures F, G, if $|\mathrm{Hom}(F,H)| = |\mathrm{Hom}(G,H)|$ for all finite relational structures H, where $\mathrm{Hom}(A,B)$ is the set of homomorphisms $h: A \to B$, then $F \cong G$. In database terminology, this says that bag-set equivalence of irredundant Boolean CQs is the same as isomorphism.

Theorem 7 was extended to UCQs in [5], and the PSPACE upper bound of Theorem 3 was also shown to hold for bag-set equivalence of UCQ$^{\neq}$s in [5].

In general, results on bag-set semantics correspond to results on bag semantics via the following *transfer lemma*:

Lemma 1 ([7]). *There exists a mapping* $\varphi : CQ \to CQ$ *(which extends to UCQs by applying it componentwise on CQs), a mapping f from bag instances to set instances, and a mapping g from set instances to bag instances, such that for any UCQ* $\bar{Q}$*, bag instance I, and set instance J:*

1. $\bar{Q}(I) = \varphi(\bar{Q})(f(I))$
2. $\varphi(\bar{Q})(J) = \bar{P}(g(J))$

In particular this lemma shows that bag-containment of CQs (UCQs) is polynomial time reducible to bag-set containment, as shown for CQs in [3]. Thus Theorem 5 also implies that bag-set containment of UCQs is undecidable. Also, Lemma 1 generalizes to queries with inequality predicates, hence Theorem [10] also implies also the undecidability of bag-set containment of CQ$^{\neq}$s, as shown in [10]. Finally, note that for UCQs $\bar{P}, \bar{Q}$ the transformation ϕ can be defined such that $\bar{P} \cong \bar{Q}$ iff $\varphi(\bar{P}) \cong \varphi(\bar{Q})$. Theorem 2 thus follows from a similar result for bag-set equivalence, also shown in [5]:

Theorem 8 ([5]). *For unions of irredundant CQs* $\bar{P}, \bar{Q}$*, we have* $\bar{P} \equiv_{bs} \bar{Q}$ *iff* $\bar{P} \cong \bar{Q}$.

Bag semantics of datalog queries. As with the relational algebra, the semantics of Datalog queries can be extended operate on bag instances. This is done by defining how *derivation trees* for an output tuple contribute to the multiplicity of the tuple. The *fringe* of a derivation tree τ of an output tuple t, denoted $\mathrm{fringe}(\tau)$, is the bag of leaves (source tuples) in the derivation tree. The count of an output tuple is computed by summing over all derivation trees, and taking the product of the multiplicities of the source tuples in the fringe. If a source tuple appears several times in the fringe, it is counted that many times in the product. Formally, for a Datalog program Q and source bag instance I, the multiplicity of a tuple t in the output is defined by

$$Q(t) \overset{\mathrm{def}}{=} \sum_{\tau} \prod_{A(t') \in \mathrm{fringe}(\tau)} A(t') \qquad (3)$$

where the sum is over all derivation trees for t. Note that when Q is a CQ or UCQ, the above definition agrees with definitions (1.1) and (1.2). However, a basic difference is that for recursive queries, there may be *infinitely many* derivation trees for an output tuple. In this case, its count is defined to be ∞.

Example 5. *For the transitive closure query TC and bag relation R defined by*

$$\begin{array}{lcl} TC(x,y) & :- & R(x,y) \\ TC(x,z) & :- & TC(x,y), R(y,z) \end{array} \qquad R = \begin{array}{ccc} a & b & 1 \\ a & c & 2 \\ c & b & 1 \\ c & d & 1 \\ d & d & 3 \end{array}$$

evaluating the query yields $TC(a,\, b) = 1 + 2 \cdot 1 = 3$ *but* $TC(c,\, d) = \infty$.

Computational problems related to infinite counts are therefore of central interest for Datalog programs with bag semantics. The most basic problem is to check whether or not a count for a given output tuple is ∞. Clearly, computing the set of all derivation trees, and then checking whether or not the set is finite, is not feasible. However, it turns out that the problem is decidable in polynomial time (data complexity), even for Datalog extended with safe stratified negation [15]. The related problem of checking statically whether answer counts are finite for all possible source bag instances is also decidable, even for Datalog extended with and negation on unary edb's [15]. However, checking this property is undecidable for Datalog extended with safe stratified negation [15].

Key Applications

The use of bag semantics in practical RDBMS implementations has led to a reexamination of fundamental issues in query optimization, namely, query containment and query equivalence. The switch from classical set semantics to bag semantics turns out to have a radical impact on these issues. Bag semantics also poses challenges for processing of recursive Datalog programs, where infinite multiplicities may arise in the output.

Future Directions

The most salient open problem involving bag semantics concerns containment of conjunctive queries. This was shown to be Π_2^p-hard in [3], but the decidability of the problem remains open. In contrast, for set semantics, the problem is known to be *NP*-complete.

Cross-references

- ▶ Bag Semantics
- ▶ Data Models
- ▶ Duplicate Elimination
- ▶ Multiset Semantics

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases. Addison-Wesley, Reading, MA, 1995.
2. Buneman P., Naqvi S., Tannen V., and Wong L. Principles of programming with complex objects and collection types. Theor. Comput. Sci., 149(1):3–48, 1995.
3. Chaudhuri S. and Vardi M.Y. Optimization of *real* conjunctive queries. In Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1993.
4. Cohen S. Equivalence of queries combining set and bag-set semantics. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 70–79.
5. Cohen S., Nutt W., and Serebrenik A. Rewriting aggregate queries using views. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999.
6. Dayal U., Goodman N., and Katz R.H. An extended relational algebra with control over duplicate elimination. In Proc. 1st ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1982, pp. 117–123.
7. Green T.J. Containment of conjunctive queries on annotated relations. In Proc. 12th Int. Conf. on Database Theory, 2009.
8. Grumbach S., Libkin L., Milo T., and Wong L. Query languages for bags: Expressive power and complexity. SIGACT News, 1996, p. 27.
9. Ioannidis Y.E. and Ramakrishnan R. Containment of Conjunctive Queries: Beyond Relations as Sets. ACM TODS, 20 (3):288–324, 1995.
10. Jayram T.S., Kolaitis P.G., and Vee E. The containment problem for *real* conjunctive queries with inequalities. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006.
11. Klausner A. and Goodman N. Multirelations – semantics and languages. In VLDB, 1985.
12. Klug A.C. Equivalence of relational algebra and relational calculus query languages having aggregate functions. J. ACM, 29(3):699–717, 1982.
13. Lovász L. Operations with structures. Acta Math. Hungarica, 18(3–4):321–328, 1967.
14. Mumick I.S., Pirahesh H., and Ramakrishnan R. The magic of duplicates and aggregates. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 264–277.
15. Mumick I.S. and Shmueli O. Finiteness properties of database queries. In Proc. 4th Australian Database Conf. Brisbane, Australia, February 1993.
16. Nutt W., Sagiv Y., and Shurin S. Deciding equivalences among aggregate queries. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 214–223.

Bagging

Wei Fan[1], Kun Zhang[2]
[1]IBM T.J. Watson Research, Hawthorne, NY, USA
[2]Xavier University of Louisiana, New Orleans, LA, USA

Synonyms

Bootstrap aggregating

Definition

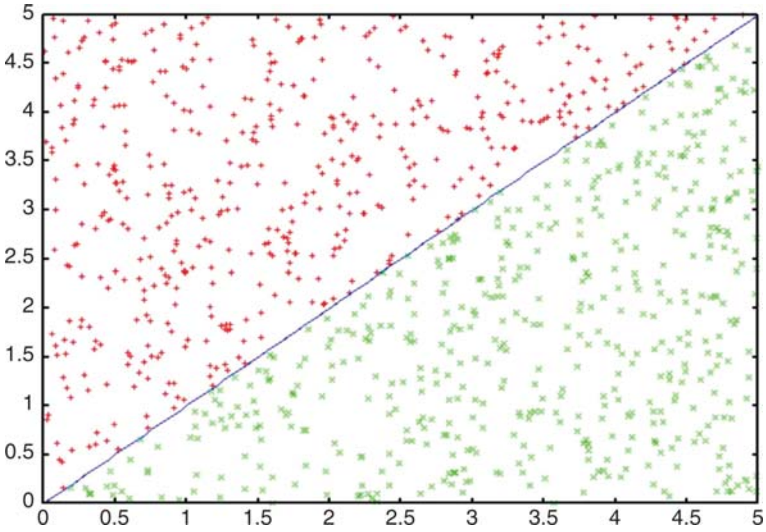
Bagging (**B**ootstrap **Agg**regating) uses "majority voting" to combine the output of different inductive models, constructed from bootstrap samples of the same training set. A bootstrap has the same size as the training data, and is uniformly sampled from the original training set with replacement. That is, after an example is selected from the training set, it is still kept in the training set for subsequent sampling and the same example could be selected multiple times into the same bootstrap sample. When the training set is sufficiently large, on average, a bootstrap sample has 63.2% unique examples from the original training set, and the rest are duplicates. In order to make full use of bagging, typically, one need to generate at least 50 bootstrap samples and construct 50 classifiers using these samples. During prediction, the class label receiving the most votes or most predictions from the base level 50 classifiers will be the final prediction. Normally, Bagging is more accurate than a single classifier trained from the original training set. Statistically, this is due to multiple classifiers' power to reduce the effect of overfitting on the training data and statistical variance. Bagging works the best for non-stable inductive models, such as decision trees, and its advantage is limited for stable methods such as logistic regression, SVM, naive Bayes, etc.

Historical Background

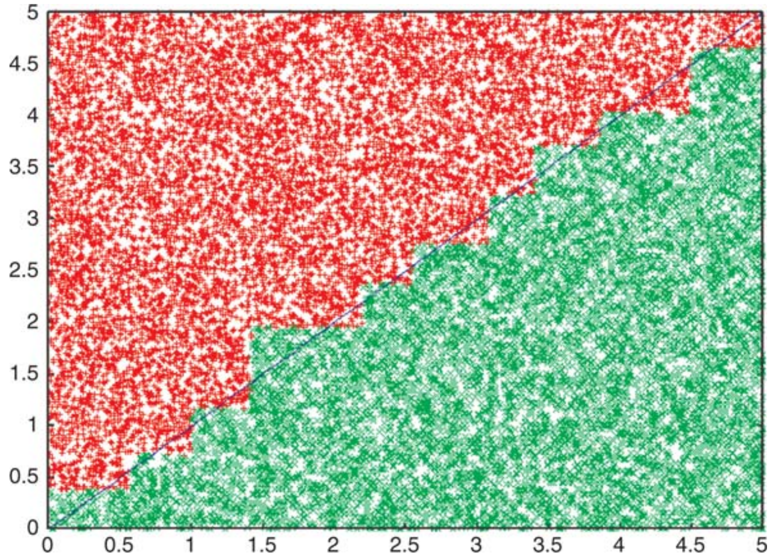
Bagging was originally proposed by Leo Breiman in 1996 [3]. The motivation comes from the fact that unstable inductive learners such as decision trees tend to generate very different models with just a slight change in the training data, for example, the inclusion or exclusion of one example. This is due to the "greedy-based" search heuristics of these methods to find the best hypothesis to fit the labeled training data. Statistically, the effect is that the variance in bias and variance decomposition of error is large. Bias is the systematic error of the chosen learning technique, and variance is due to variations in the training data, i.e., different training data produce different models. Bagging was proposed to remedy these problems. In order to overcome variations of a single training set, multiple bootstrap samples are used to replace the single original training set. Since each bootstrap sample is randomly selected from the original training data, it still "preserves" the main concept to be modeled. However, each bootstrap sample is different, thus offsetting the "individuality" of any single training set that contributes to variance. In the same time, any possible over-sample or "non-sample" of particular labeled examples from the training data (recall that 63.2% are unique examples and the rest are duplicates) that could reduce learning accuracy is resolved by majority voting or choosing the predicted labels with the most number of votes from base line models. The simple observation is that the probability for most models to make the same mistakes is much less than any single model itself.

Foundations

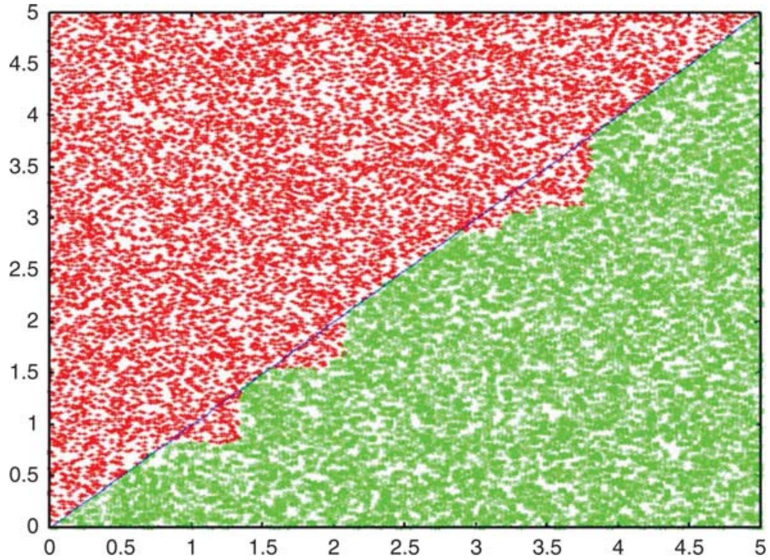
The key idea of Bagging can be illustrated by a simple synthetic example. In Fig. 1, the true decision boundary that separates two classes of examples is a simple linear function, $y = x$, where examples above the line belongs to one class and examples below the line belongs to another class. The labeled training examples are randomly sampled from the universe of examples and is obviously not exhaustive. When a decision tree algorithm is applied to the sampled training data, it will construct a tree model particular to that sample set. Figure 2 shows the decision boundary of a single tree constructed by C4.5 algorithm [12]. As observed, the "stair-step" shape of the decision boundary is due to "bias" or systematic error of decision tree algorithm, where the splits are always perpendicular to the axes. Once an algorithm is chosen, its bias is hard to avoid. At the same time, the exact position and size of each step is due to variations in individually sampled training data, and it contributes to "variance" term in the prediction error. Next, this entry studies how "variance" can be effectively reduced by applying Bagging. Figure 3 demonstrated the decision boundary of Bagging generated from 50 bagged C4.5 trees. It is clear



Bagging. Figure 1. Training data for a simple linear function.



Bagging. Figure 2. Decision boundary constructed by a single tree.



Bagging. Figure 3. Decision boundary constructed by bagging.

that the decision boundary is much closer to the perfect "$y = x$" line, and only three steps are left.

Formally, the reduction in variance can be approximated by the following equation:

$$\epsilon = \beta + \frac{\sigma}{\sqrt{n}}, \tag{1}$$

where the error ε is decomposed into bias β and variance σ. The reduction in variance is scaled by $\sqrt{n}$ where n is the number of bagging models. Additionally, the idea of Bagging can be applied to both classification problem or discrete variable prediction (such as the synthetic example given above) and regression problems or continuous variable prediction. In regression problems, the estimated value of different models are averaged as the final prediction.

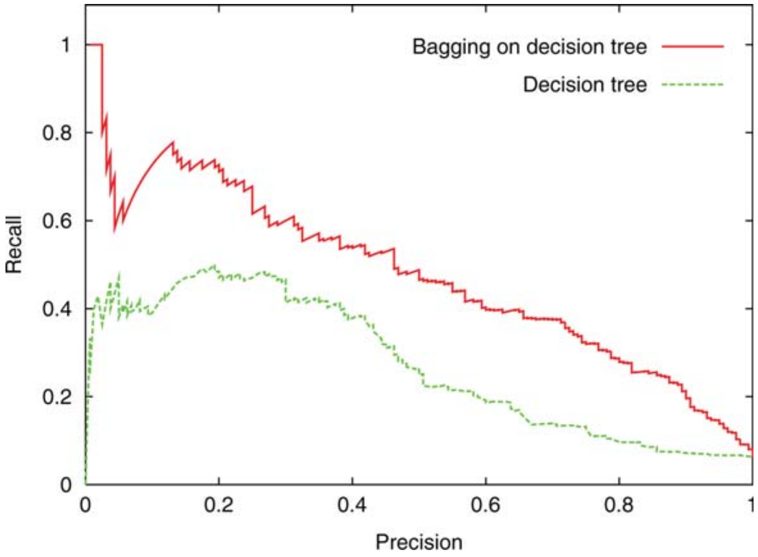
For further reading and understanding of Bagging and its various applications, refer to the 15 papers in the "Recommended Reading" section. For theories to explain how and why Bagging works, the best source is the original paper by Leo Breiman [3], that explains the statistical fundamentals that Bagging is built upon. Additionally, [5] explains the appropriate understanding of overfitting and how overfitting plays a role in classifers' generalization power. In essence, Bagging helps to correct the overfitting problem of unstable learners, such as decision trees. For various information on how to build an accurate decision tree, the base model where Bagging normally combines, the following list of papers contain solid information [2,4,9,11,12,13]. For state-of-the-art and future works of Bagging, one of the most novel ideas is "randomization" and different treatment on this subject can be found in various papers [1,6,7,14]. During the time when Bagging was proposed, an alternative method called "Boosting" was proposed to resolve the instability of inductive learners and some papers that theoretically describe its motivation and practice can be found in [8,10]. For a detailed and systematic comparison of Bagging and many other state-of-the art algorithms on a difficult application problem "skewed and stochastic ozone day forecasting", the works in [15] contain useful information.

Key Applications

Bagging is most suitable for applications where both the training set is not too big, i.e., the number of training examples can fit into main memory of the machine, and the chosen inductive learner is unstable, such as decision trees. This is mainly because generating multiple bootstrap sample involves multiple scans of the training data and can be a bottleneck if most operations rely on disk. At the same time, if the number of examples does not fit into main memory of the machine, learning can incur swapping cost and take a long time to complete. As discussed previously, bagging's success is mainly limited to unstable learners that normally have large variance.

As a good example, "ozone day prediction" [15] illustrates how well Bagging can perform in practice. What makes this problem interesting is that there are 72 continuous features in the collected dataset, only about 3% of 2,500+ collected days are positive. It is important to understand that for a problem with 72 dimensions, 2,500 training examples are trivial in size. Even if these 72 features are binary in values, the total problem size is an astronomical number 2^{72}, and obviously, 2,500 samples from this space is "really nothing".

Applying non-stable inductive learning method, such as decision tree, on this problem is unlikely to obtain satisfactory result. The simple reason is that these methods tend to "overfit" or build unnecessarily detailed models to fit well on the given 2,500 examples, but do not generalize well on unseen testing data in the problem space of 2^{72}. As illustrated in a precision-recall plot in Fig. 4, Bagging consistently obtains higher "precision" than its comparable single tree methods. Recall is the percentage of "true ozone days" that are correctly predicted as ozone days, and precision is the



Bagging. Figure 4. Bagging versus single decision tree.

percentage of "predicted ozone days" that are actually ozone days. By reading the consistently higher precision numbers by Bagging on the same recall number, it clearly demonstrates that Bagging is more accurate than single decision trees.

Future Directions

Most recently, a completely counter-intuitive method called "Random Decision Tree" [7,6,14] has been proposed to reduce both bias and variance in inductive learning. Each tree in a random tree ensemble is used as a structure to summarize the training data, and importantly, the structure of the tree is semi-randomly constructed, and data are "filled-in" the tree structure to summarize statistics of the training data. During prediction, the estimated probability from each tree is averaged as the final conditional probability $P(\mathbf{y}|\mathbf{x})$ and the class label with the highest probability is chosen as the predicted label. Since it does not incur any overhead of producing bootstraps or using information gain to perform feature selection as Bagging does, random decision tree can be highly efficient. One application on difficult skewed and high dimensional ozone day forecast can be found in [15]. Similar to Bagging, Random Decision Tree is applicable to both classification and regression problems.

Cross-references

▶ Boosting
▶ Decision Tree

Recommended Reading

1. Amit Y. and Geman D. Shape quantization and recognition with randomized trees. Neural Comput., 9(7):1545–1588, 1997.
2. Bradford J.P., Kunz C., Kohavi R., Brunk C., and Brodley C.E. Pruning decision trees with misclassification costs. In Proc. Eur. Conf. Mach. Learn., 131–136, 1998.
3. Breiman L. Bagging predictors. Mach. Learn., 24(2):123–140, 1996.
4. Buntine W. Learning classification trees. In Artificial Intelligence frontiers in statistics, D.J. Hand (ed.). Chapman & Hall, London, 1993, pp. 182–201.
5. Domingos P. Occam's two razors: The sharp and the blunt. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998.
6. Fan W., Greengrass E., McCloskey J., Yu P.S., and Drummey K. Effective estimation of posterior probabilities: Explaining the accuracy of randomized decision tree approaches. In Proc. IEEE Int. Conf. on Data Mining, 2005, pp. 154–161.
7. Fan W., Wang H., Yu P.S., and Ma S. Is random model better? on its accuracy and efficiency. In Proc. 19th Int. Conf. on Data Engineering, 2003.
8. Freund Y. and Schapire R. A decision-theoretic generalization of on-line learning and an application to boosting. Comput. Syst. Sci., 55(1):119–139, 1997.
9. Gehrke J., Ganti V., Ramakrishnan R., and Loh W.-Y. BOAT-optimistic decision tree construction. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
10. Kearns M. and Mansour Y. On the boosting ability of top-down decision tree learning algorithms. In Proc. Annual ACM Symp. on the Theory of Computing, 1996, pp. 459–468.
11. Mehta M., Rissanen J., and Agrawal R. MDL-based decision tree pruning. In Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining, 1995, pp. 216–221.
12. Quinlan R. C4.5: Programs for Machine Learning. Morgan Kaufmann, Los Altos, CA, 1993.
13. Shawe-Taylor J. and Cristianini N. Data-dependent structural risk minimisation for perceptron decision trees. In Advances in Neural Information Processing Systems 10, M. Jordan, M. Kearns, and S. Solla (eds.). MIT Press, Cambridge, MA, 1998, pp. 336–342.
14. Zhang K., Xu Z., Peng J., and Buckles B.P. Learning through changes: An empirical study of dynamic behaviors of probability estimation trees. In Proc. IEEE Int. Conf. on Data Mining, 2005, pp. 817–820.
15. Zhang K. and Fan W. Forecasting skewed biased stochastic ozone days: analyses, solutions and beyond. Know. Inf. Syst., 14(3), 2008.

Base-line Clock

▶ Time-Line Clock

Bayes Classifier

▶ Bayesian Classification

Bayesian Classification

Wynne Hsu
National University of Singapore, Singapore, Singapore

Synonyms

Bayes classifier

Definition

In classification, the objective is to build a classifier that takes an unlabeled example and assigns it to a class. Bayesian classification does this by modeling the probabilistic relationships between the attribute set and the class variable. Based on the modeled relationships, it estimates the class membership probability of the unseen example.

Historical Background

The foundation of Bayesian classification goes back to Reverend Bayes himself [2]. The origin of Bayesian belief nets can be traced back to [15]. In 1965, Good [4] combined the independence assumption with the Bayes formula to define the Naïve Bayes Classifier. Duda and Hart [14] introduced the basic notion of Bayesian classification and the naïve Bayes representation of joint distribution. The modern treatment and development of Bayesian belief networks is attributed to Pearl [8]. Heckerman [13] later reformulated the Bayes results and defined the probabilistic similarity networks that demonstrated the practicality of Bayesian classification in complex diagnostic problems.

Foundations

Bayesian classification is based on Bayes Theorem. It provides the basis for probabilistic learning that accommodates prior knowledge and takes into account the observed data.

Let X be a data sample whose class label is unknown. Suppose H is a hypothesis that X belongs to class Y. The goal is to estimate the probability that hypothesis H is true given the observed data sample X, that is, P(Y|X).

Consider the example of a dataset with the following attributes: Home Owner, Marital Status, and Annual Income as shown in Fig. 1. Credit Risks are Low for those who have never defaulted on their payments and credit risks are High for those who have previously defaulted on their payments.

Assume that a new data arrives with the following attribute set: X = (Home Owner = Yes, Marital Status = Married, Annual Income = High). To determine the credit risk of this record, it is noted that the Bayes classifier combines the predictions of all alternative hypotheses to determine the most probable classification of a new instance. In the example, this involves computing P(High|X) and P(Low|X) and to determine whether P(High|X) > P(Low|X)?

However, estimating these probabilities is difficult, since it requires a very large training set that covers every possible combination of the class label and attribute values. Instead, Bayes theorem is applied and it resulted in the following equations:

$$\mathrm{P(High|X) = P(X|High)P(High)/P(X)} \text{ and}$$

$$\mathrm{P(Low|X) = P(X|Low)P(Low)/P(X)}$$

P(High), P(Low), and P(X) can be estimated from the given dataset and prior knowledge. To estimate the class-conditional probabilities P(X|High), P(X|Low), there are two implementations: the Naïve Bayesian classifier and the Bayesian Belief Networks.

In the Naïve Bayesian classifier [13], the attributes are assumed to be conditionally independent given the class label y. In other words, for an n-attribute set X = (X_1, X_2,...,X_n), the class-conditional probability can be estimated as follows:

$$P(Y|X) = \alpha P(Y)\prod_i P(X_i|Y)$$

Customer ID	Home owner	Marital status	Annual income	Credit risks
1	Yes	Married	High	Low
2	No	Single	Medium	High
3	No	Married	Medium	High
4	Yes	Divorced	Low	High
5	No	Married	High	Low
6	No	Divorced	High	Low
7	Yes	Single	Low	Low
8	No	Single	High	Low
9	No	Married	Medium	High
10	Yes	Single	Medium	Low

Bayesian Classification. Figure 1. Dataset example.

In the example,

$$\begin{aligned}&(\mathrm{X}|\mathrm{Low}) = \mathrm{P}(\text{Home Owner}\\&\qquad = \text{Yes}|\text{Credit Risk} = \text{Low})\times\\&\mathrm{P}(\text{Marital Status} = \text{Married}|\text{Credit Risk}\\&\qquad = \text{Low})\times\\&\mathrm{P}(\text{Annual Income} = \text{High}|\text{Credit Risk}\\&\quad = \text{Low} = 3/4 \times 2/4 \times 4/4 = 3/8\\&\mathrm{P}(\mathrm{X}|\text{High}) = \mathrm{P}(\text{Home Owner}\\&\qquad = \text{Yes}|\text{Credit Risk} = \text{High})\times\\&\mathrm{P}(\text{Marital Status}\\&\quad = \text{High})\times\\&\mathrm{P}(\text{Annual Income} = \text{High}|\text{Credit Risk}\\&\qquad = \text{High}) = 0\end{aligned}$$

Putting them together, $\mathrm{P(High|X)} = \mathrm{P(X|High)P(High)/P(X)} = 0$

$$\mathrm{P(Low|X) = P(X|Low)P(Low)/P(X)} > 0$$

Since $\mathrm{P(Low|X)} > \mathrm{P(High|X)}$, X is classified as having Credit Risk = Low.

In other words,

$$\mathit{Classifiy}(X) = \arg\max_{y} P(Y = y)\prod_{i} P(X_i|Y = y)$$

In general, Naïve Bayes classifiers are robust to isolated noise points and irrelevant attributes. However, the presence of correlated attributes can degrade the performance of naïve Bayes classifiers as they violate the conditional independence assumption. Fortunately, Domingos and Pazzani [3] showed that even when the independence assumption is violated in some situations, the naïve Bayesian classifier can still be optimal. This has led to a wide spread use of naïve Bayesian classifiers in many applications. Jaeger [9] also further clarifies and distinguishes the concepts that can be recognized by naïve Bayes classifiers and the theoretical limits on learning the concepts from data.

There are many extensions to the naïve Bayes classifier that impose limited dependencies among the feature/attribute nodes, such as tree-augmented naïve Bayes [6], and forest-augmented naïve Bayes [11].

Bayesian belief network [10] overcomes the rigidity imposed by this assumption by allowing the dependence relationships among a set of attributes to be modeled as a directed acyclic graph. Associated with each node in the directed acyclic graph is a probability table. Note that a node in the Bayesian network is conditionally independent of its non-descendants, if its parents are known.

Refer to the running example. Suppose the probabilistic relationships among Home Owner, Marital Status, Annual Income, and Credit Risks are shown in Fig. 2. Associated with each node is the corresponding conditional probability table relating the node to its parent node(s).

In the Bayesian belief network, the probabilities are estimated as follows:

$$P(Risk|X) = \alpha \sum_{I} P(Risk|Income) \sum_{O} P(Owner) \sum_{S} P(Status)P(Income|Owner, Status)$$

In the above example,

$$\begin{aligned}\mathrm{P(Low|X)} =& \mathrm{P(Low|Income = High)}\\&*\mathrm{P(Income = High|Owner}\\&= \mathrm{Yes, Status = Married)} = 1 * 1 = 1\end{aligned}$$

Alternatively, by recognizing that given Annual Income, Credit Risks is conditionally independent of Home Owner and Marital status, then

$$\mathrm{P(Low|X) = P(Low|Income = High)} = 1$$

This example illustrates that the classification problem in Bayesian networks is a special case of belief updating for any node (target class) Y in the network, given evidence X.

While Bayesian belief network provides an approach to capture dependencies among variables using a graphical model, constructing the network is time consuming and costly. Substantial research has been and is still continuing to address the inference as well as the automated construction of Bayesian networks by learning from data. Much progress has been made recently. So applying Bayesian networks for classification is no longer as time consuming and costly as before, and the approach is gaining headway into the mainstream applications.

Key Applications

Bayesian classification techniques have been applied in many applications. Here, a few more common applications of Bayesian classifiers are mentioned.

Text Document Classification

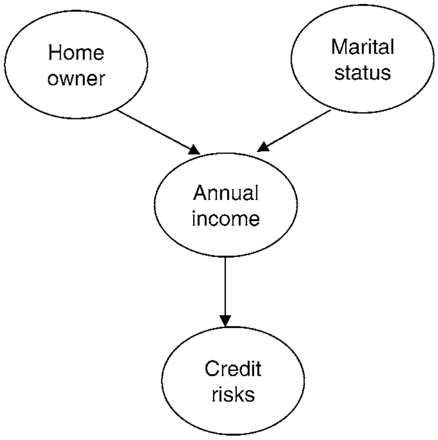
Text classification refers to the grouping of texts into several clusters so as to improve the efficiency and effectiveness of text retrieval. Typically, the text documents are pre-processed and the key words chosen. Based on the selected keywords of the documents, probabilistic classifiers are built. Dumais et al. [5] show naïve Bayes classifier yields surprisingly good classifications for text documents.

Image Pattern Recognition

In image pattern recognition, a set of elementary or low level image features are selected which describe some characteristics of the objects. Data extracted based on this feature set are used to train Bayesian classifiers for subsequent object recognition. Aggarwal et al. [1] did a comparative study of three paradigms for object recognition – Bayesian Statistics, Neural Networks and Expert Systems.

Medical Diagnostic and Decision Support Systems

Large amounts of medical data are available for analysis. Knowledge derived from analyzing these data can be used to assist the physician in subsequent diagnosis. In this area, naïve Bayesian classifiers performed exceptionally well. Kononenko et al. [12] showed that



Owner = Yes
0.4

(Table associated with node owner)

Status	
Married	0.4
Single	0.4
Divorced	0.2

(Table associated with node status)

	Income = High	Income = Medium	Income = Low
HO = Yes, MS = Married	1	0	0
HO = Yes, MS = Divorced	0	0	1
HO = Yes, MS = Single	0	0.5	0.5
HO = No, MS = Married	0.33	0.67	0
HO = No, MS = Divorced	1	0	0
HO = No, MS = Single	0.5	0.5	0

(Table associated with node income)

	Risk = High	Risk = Low
Income = High	0	1
Income = Medium	0.75	0.25
Income = Low	0.5	0.5

(Table associated with node credit risks)

Bayesian Classification. Figure 2. Bayesian belief network of the credit risks dataset.

the naïve Bayesian classifier outperformed other classification algorithms on five out of the eight medical diagnostic problems.

Email Spam Filtering

With the growing problem of junk email, it is desirable to have an automatic email spam filters to eliminate unwanted messages from a user's mail stream. Bayesian classifiers that take into consideration domain-specific features for classifying emails are now accurate enough for real world usage.

Data Sets

http://archive.ics.uci.edu/beta/datasets.html
http://spamassassin.apache.org/publiccorpus/

URL to Code

More recent lists of Bayesian Networks software can be found at:

Kevin Murphy's website:
http://www.cs.ubc.ca/~murphyk/Bayes/bnsoft.html
Google directory:
http://directory.google.com/Top/Computers/Artificial_Intelligence/Belief_Networks/Software/
Specialized naïve Bayes classification software:
jBNC – a java toolkit for variants of naïve Bayesian classifiers, with WEKA interface

Cross-references

▶ Belief Networks
▶ Probabilistic Model
▶ Supervised Classification
▶ UnSupervised Classification

Recommended Reading

1. Aggarwal J.K., Ghosh J., Nair D., and Taha I. A comparative study of three paradigms for object recognition – Bayesian statistics, neural networks, and expert systems. Advances In Image Understanding: A Festschrift for Azriel Rosenfeld. IEEE Computer Society Press, Washington, DC, 1996, pp. 241–262.
2. Bayes T. An essay towards solving a problem in the doctrine of chances. Philos. Trans. R. Soc., 53:370–418, 1763.
3. Domingos P. and Pazzani M. Beyond independence: conditions for the optimality of the simple Bayesian classifier. In Proc. 13th Int. Conf. on Machine Learning, 1996, pp. 105–112.
4. Duda R.O. and Hart P.E. Pattern Classification and Scene Analysis. Wiley, New York, 1973.
5. Dumais S., Platt J., Heckerman D., and Sahami M. Inductive learning algorithms and representations for text categorization. In Proc. Int. Conf. on Information and Knowledge Management, 1998.
6. Friedman N., Geiger D., and Goldszmidt M. Bayesian network classifiers. Mach. Learn., 29:131–163, 1997.
7. Good I.J. The Estimation of Probabilities: An Essay on Modern Bayesian Methods. MIT Press, Cambridge, MA, 1965.
8. Heckerman D. Probabilistic Similarity Networks. ACM Doctoral Dissertation Award Series. MIT Press, Cambridge, MA, 1991.
9. Jaeger M. Probabilistic classifiers and the concepts they recognize. In Proc. 20th Int. Conf. on Machine Learning, 2003, pp. 266–273.
10. Jensen F.V. An introduction to Bayesian networks. Springer, New York, 1996.
11. Keogh E. and Pazzani M. Learning augmented Bayesian classifiers: A comparison of distribution-based and classification-based approaches. In Proc. 7th Int. Workshop on Artificial Intelligence and Statistics, 1999.
12. Kononenko I., Bratko I., and Kukar M. Application of Machine Learning to Medical Diagnosis. Machine Learning, Data Mining and Knowledge Discovery: Methods and Applications. Wiley, New York, 1998.
13. Langley P., Iba W., and Thompson K. An analysis of Bayesian classifiers. In Proc. 10th National Conf. on Artificial Intelligence, 1992, pp. 223–228.
14. Pearl J. Probabilistic Reasoning in Intelligenet Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo, CA, 1988.
15. Wright S. Correlation and causation. J. Agric. Res., 20(7): 557–585, 1921.

BCNF

▶ Boyce-Codd Normal Form

Belief Time

▶ Transaction Time

Benchmark

▶ Application Benchmark

Biased Distribution

▶ Data Skew

Bibliography

▶ Citation

Bi-clustering

▶ Subspace Clustering Techniques

Bioinformatics

▶ Implications of Genomics for Clinical Informatics

Biological Data Retrieval, Integration, and Transformation

▶ Query Languages for the Life Sciences

Biological Metadata Management

Zoé Lacroix[1], Cartik R. Kothari[2], Peter Mork[3], Mark Wilkinson[2], Sarah Cohen-Boulakia[4]
[1]Arizona State University, Tempe, AZ, USA
[2]University of British Columbia, Vancouver, BC, Canada
[3]The MITRE Corporation, McLean, VA, USA
[4]University Paris-Sud, Orsay Cedex, France

Definition

Metadata characterize biological resources by core information including a name, a description of its input and its output (parameters or format), its address, and various additional properties. Resources are organized with respect to metadata that characterize their content (for data sources), their semantics (in terms of ontological classes and relationships), their characteristics (syntactical properties), their performance (with metrics and benchmarks), their quality (curation, reliability, trust), etc.

Historical Background

Digital resources for the Life Sciences include a variety of data sources and applications whose number increases dramatically every year [4]. Although this rich and valuable offering provides scientists with multiple options to implement and execute their scientific protocols (i.e., pipelines, dataflows, workflows), selecting the resources suitable for implementing each scientific step remains a difficult task. Scientific protocols are typically implemented using the resources a scientist is most familiar with, instead of the resources that may best meet the protocol's needs. The number of resources a scientist uses regularly, knowing their structure, the quality of data and annotations they offer, the capabilities made available by the provider to access, analyze, and display the data are ridiculously small compared to the thousands of resources made available on the Web [3]. Metadata are not only critical in selecting suitable resources for implement scientific protocols, but they are essential to the proper composition and integration of resources in a platform such as a workflow system or wrapped into a database mediation system. They also play a decisive role in the analysis of data, in particular to track data provenance. Finally, they contribute to data curation and the management of Life Sciences resources in a global and linked digital biological maze.

Foundations

Metadata management relies on the description of resources including the resource name, identification, and all additional information that may be relevant to locating, evaluating, and using the resource. A *resource identifier* is a sequence of characters that uniquely identifies a resource and is globally shared and understood over a network. A resource is analogous to a node on the Web. The ubiquitous Uniform Resource Locator (URL) is an example of a resource identifier, which uses the location, the local directory path, and the local file name of the resource to locate it on the Web. Unique Resource Identifiers (URIs) include URLs that not only identify the resource but describe its primary access mechanism or network location, and Uniform Resource Names (URN) that identify a resource by name in a particular namespace. The unique identification of resources is an unresolved problem in the life sciences community. Different protein, gene, and molecular interaction databases often assign separate identifiers to the same resource, a phenomenon known as *coreference*. Leveraging the information from all these databases becomes problematic, leading to duplicate records and inconsistency. To alleviate this problem, many Life Sciences databases cross reference their identifiers.

Metadata are data that describe a resource. Metadata include a wide range of information from attribution metadata, such as those attributes defined in the Dublin Core, to detailed policy metadata indicating who can access the resource under what conditions. *Semantic metadata* include the description of a resource with respect to the domain knowledge (e.g., a data source provides information about proteins, a tool computes the translation of a RNA sequence into an AA sequence). *Syntactic metadata* provide the description of the resource interface. *Summary metadata* describe the actual contents of the resource. These metadata include free text summaries and statistical summaries of the instances (values) contained in the database. Summary metadata can be classified along several axes: (i) textual versus quantitative, (ii) structured versus unstructured, and (iii) manually generated versus automatically generated. By far the most common type of summary metadata are textual. For example, NAR [4] maintains a listing of hundreds of biomedical resources. For each resource, they provide a brief description of the contents of that resource. Textual metadata allow an application developer or end-user to search for resources using keywords or phrases. The success of existing approaches seems to show that it is a familiar and intuitive operation, which works well when searching for reasonably well-defined concepts. Textual metadata are unstructured (i.e., free text) and manually curated. Alternatively, summary metadata can take the form of keywords drawn from a *controlled vocabulary*, such as Medical Subject Headings (MeSH) terms. A controlled vocabulary makes it easier to search for resources, assuming the vocabulary is sufficiently expressive and used consistently to annotate the resources. In most cases, textual metadata are generated manually, although there is some research in automatically extracting keywords from a resource for its annotation.

Quantitative metadata describe resources in terms of numeric datatypes. In the simplest case, these metadata specify the range of values that can be found in the resource. For example, all of the subjects in a pediatric database would be younger than 18. More detailed summaries are also possible. In the case of quantitative metadata, unstructured metadata make little sense. The end user needs to know what a given number represents, including relevant units. Quantitative metadata can still be generated manually or automatically. The former is required if the resource does not contain the necessary raw data. For example, if a pediatric database does not contain the ages of its subjects, the relevant age range must be specified manually. However, when the resource does contain the necessary raw data, quantitative metadata can be generated automatically. Moreover, the amount of detail in the metadata can vary depending on the needs of the resource owners and community members searching for resources. A current research challenge is determining the appropriate granularity for quantitative metadata and using these metadata to estimate the extent to which a given resource matches the end user's search criteria. *Statistical metadata* and benchmarks provide an additional layer exploited when the scientist wishes to predict the outcome of an execution. These metadata are particularly useful when several resources are combined to evaluate alternative evaluation strategies and select the most efficient one with respect to the protocols' aim [5]. The domain and range of resources, as well as resource overlaps contribute to the statistical description of resources. Similarly, information related to the quality of the resource (e.g., curation) may be exploited to optimize the quality of the execution.

Structural metadata describe the resource interface and the intention of the resource provider. These metadata can take on many forms including database schemata, Unified Modeling Language (UML) diagrams or Web service descriptions. What structural metadata provide are a description of how the resource provider intends to organize and deliver data. Controlled vocabularies capture domain knowledge and clarify resource descriptions (e.g., identical concepts). Controlled vocabularies are naturally extended by logical or conceptual representations such as expressed in a domain ontology. More generally, a metadata registry containing structural metadata allows an application developer to search for resources that are intended to contain particular types of data. Moreover, once a developer discovers a useful resource, he has a good idea of how to interact with that resource, both in terms of formulating queries and processing results. Structural metadata only indicate what sorts of queries can be posed, not whether those queries will return meaningful results. For example, the structural metadata for a card catalog might indicate that it includes, for each entry, a list of authors. Thus, one could reasonably query the card catalog for all books authored by John Grisham. However, if the card catalog supports a medical (non-fiction) library, it is unlikely that this query will return any record.

Bioinformatics resources may be represented with formats and standards developed by various communities driven by disparate motivations including business, library, Web, etc. The Resource Description Framework (RDF) and the RDF Schema (RDFS) were the earliest adopted standards for representing metadata about Web resources. The Dublin Core Metadata Elements Set (DCMES) is a standard set of metadata elements that can be used to describe a generic resource to facilitate its discovery and use. RDF specifically provides a very simple "triples" syntax or Subject-Predicate-Object syntax to capture resource metadata. Universal Description, Discovery and Integration (UDDI) is the XML-based format to register businesses on the Web proposed by OASIS. Dublin Core is a standard (NISO Standard Z39.85-2007) for cross-domain information resource description. The Web Ontology Language OWL, based on earlier languages OIL and DAML+OIL, is a W3C recommendation that extends XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics with descriptions of classes, along with their related properties and instances. The Web Service Description Language (WSDL) and its extension WSDL-S are used respectively as resource description and semantic annotation. Resource description and registration developed for the life science include BioMoby, PISE/Mobyle, caBIG, and SOAPlab.

The Life Sciences community has a number of metadata annotation standards. The Darwin Core (DwC) is a metadata standard from the National Biological Information Infrastructure (NBII) for annotating the objects contained within natural history specimen collections and species observation databases. These annotations are used to retrieve records of natural history specimens and observation records from local libraries, integrating them with other collections across the United States and making them available on the Web. The Access to Biological Collections Data Schema (ABCD Schema) is a complementary, hierarchical metadata standard for the annotation of biological specimens. Mappings exist from the terms of Darwin Core to the ABCD Schema that illustrate the overlap between the two standards and the few differences. Organizations such as NBII, the Integrated Taxonomic Information System and the Global Biodiversity Information Facility leverage metadata standards such as ABCD Schema and DwC to discover and utilize information pertaining to species and natural history specimens that is distributed around the world.

The Minimum Information About a Microarray Experiment (MIAME) [1] from the Microarray Gene Expression Data (MGED) Society, is used to describe sufficient information about a microarray experiment to reproduce it unambiguously. An increasing number of data providers are embracing the MIAME standard and several journals including NAR, Cell, and Nature require MIAME compliant data as a condition for publishing microarray based papers. The Minimum Information About a Proteomics Experiment (MIAPE) [7] is a minimum information reporting requirement for proteomics experiments. It is analogous to the MIAME standard for transcriptomics data. MIAPE is distributed across several modules, each of which is useful to describe a different proteomics experiment. Example modules are the MIAPE Gel Electrophoresis module, the MIAPE Mass Spectrometry module and the MIAPE Column Chromatography module. The Minimum Information required for reporting a Molecular Interaction Experiment (MIMIx) [6] has been developed as a framework to capture metadata about molecular interaction experiments. MIAME, MIAPE and MIMIx are being developed under the auspices of the Proteomics Standards Initiative at the Human Proteomics Organization.

Metadata standards vary in complexity from the simple format of the Dublin Core to the complex requirements of MIAME. A number of organizations are currently involved in the development of metadata standards in various fields of study. The Federal Geographic Data Committee (FGDC) is involved in the creation of several metadata standards such as the Spatial Data Transfer Standard for the exchange of spatial data, the National Vegetation Classification Standard to support a national vegetation classification system, the Biological Data Profile for the documentation of biological data, and the Utilities Data Content Standard to standardize geospatial information for utility systems. In addition the FGDC publishes the Content Standard for Digital Geospatial Metadata (CSDGM) for the annotation of geospatial data in the form of maps, atlases and satellite images. The Ocean Biogeographic Information System as an initiative to make marine biogeographic data available to a worldwide audience. OBIS has developed a taxonomic hierarchy of metadata elements, called the OBIS taxonomy for annotation of marine species observations. GeoConnections is a Canadian initiative to make geospatial data in the form of maps and satellite images easily available to a worldwide audience. GeoConnections uses the CSDGM published by the FGDC, to annotate its geospatial data.

Key Applications

Resource discovery systems exploit metadata in order to map the user requirements to the resource characteristics. For example, caCORE is an n-tier data management and integration infrastructure that combines several interconnected software and services. The Cancer Bioinformatics Infrastructure Objects (caBIO) model is at the heart of caCORE. The caBIO model contains definitions of concepts and inter-concept relationships that are common to biomedical research. These concept definitions in caBIO are the basis upon which data from distributed repositories are integrated. These repositories include gene and homolog databases (e.g., UniGene and Homologene), pathway databases (e.g., BioCarta), vocabulary and terminology repositories (e.g., the National Cancer Institute (NCI) Thesaurus, NCI Metathesaurus, and the Gene Ontology). The PathPort framework developed at The Virginia Bioinformatics Institute, presents a Web based interface that makes it possible for end users to invoke local and distributed biological Web services that are described in WSDL, in a location and platform independent manner.

Metadata offer metrics that may be used to predict the outcome of the execution of a scientific protocol on selected resources [5]. Metadata provide resource characterization that allows their comparison with similar resources, thus addressing the need of combining multiple complementary resources to implement completely a single task (e.g., data coverage). Path-based systems such as BioNavigation exploit statistical metadata to predict the resources that are likely to return the most entries, the best characterized (most attributes) entries, etc. [3].

Although some degree of transparency is often needed in queries, scientists also expect to be aware of the provenance of the answers. In order to analyze the results obtained from the execution of their scientific protocols, they often need to understand the process that produced the dataset. In particular, they need to know which resources have been used, and how entries have been linked to generate the answer to their question. Data traceability is related to the degree of information pertaining to the resources used to implement the process as well as the integration used to combine them. Because of this, reasoning on data provenance may exploit scientific resource metadata repositories. For example, BioGuideSRS allows the user to visualize the correspondence between the graph of entities and the graph of sources-entities [2]. By selecting an entity, the user visualizes the sources which provide information about this entity; similarly, by selecting a relationship, the user visualizes the links between sources which achieve this relationship. Second, the data obtained as a result yielded by BioGuideSRS to the user is systematically associated with the path which has been used to obtain it. In this way, the user knows the exact sequence of sources and links used. This approach was demonstrated with the ZOOM*UserViews system.

Future Directions

Metadata management remains a critical issue for the Life Sciences. First, the community has not agreed on common metadata to publish together with a resource so that it is properly identified, located, and used by scientists. Multiple discussions related to the representation of scientific objects generate the design of a large number of ontologies as published by the Open Biomedical Ontology (OBO) group. Although this effort contributes significantly to the better understanding of scientific information, it produces ontologies that may overlap and that are difficult to integrate. This semantic gap is aggravated by the diversity of models and formats used by biological data providers. Moreover, the community shows reluctance to adopting recommendations from the W3C Semantic Web for the specification of resources. The lack of a common publishing process for resources affects their impact significantly. In particular, it challenges the development of resource repositories to support resource discovery. Consequently, it affects the ability for scientists to select resources suitable to implement the scientific tasks involved in their protocols. The development of adequate technology, still in its infancy, is rather limited by the lack of a *franca lingua* for metadata. Future developments include the identification of metrics that adequalety capture the characteristics of resources, the design of benchmarks to evaluate and compare similar resources, automated data curation approaches that exploit and update resource metadata, automated classification of resources, data provenance analysis, etc.

Data Sets

NAR http://nar.oxfordjournals.org/

BMC Source Code for Biology and Medicine http://www.scfbm.org/home

Bioinformatics Links Directory http://bioinformatics.ca/links_directory/

Open Biomedical Ontologies (OBO) http://obofoundry.org/

Next Generation Biology Workbench (Swami) http://www.ngbw.org

URL to Code

Medical Subjects Headings (MeSH) http://www.nlm.nih.gov/mesh

UDDI http://www.uddi.org/

OWL Web Ontology Language http://www.w3.org/TR/owl-features/

DAML+OIL http://www.w3.org/TR/daml+oil-reference

BioMOBY http://biomoby.org/

PISE http://www.pasteur.fr/recherche/unites/sis/Pise/

Mobyle http://www.pasteur.fr/recherche/unites/sis/Pise/mobyle.html

caBIG http://cabig.nci.nih.gov/

SOAPlab http://www.ebi.ac.uk/Tools/webservices/soaplab/overview

National Biological Information Infrastructure (NBII) http://www.nbii.gov/

Microarray Gene Expression Data (MGED) Society http://www.mged.org

Federal Geographic Data Committee (FGDC) http://www.fgdc.gov

Ocean Biogeographic Information System (OBIS) http://www.iobis.org

GeoConnections http://www.geoconnections.org

caCORE http://ncicb.nci.nih.gov/infrastructure/cacoresdk

ZOOM*UserViews system http://db.cis.upenn.edu/research/provwf.html

BioNavigation http://bioinformatics.eas.asu.edu/

BioGuide http://bioguide-project.net/

Cross-references

▶ Benchmark
▶ Biological Resource Discovery
▶ Dublin Core
▶ Graph Management in the Life Sciences
▶ HTTP
▶ Metadata
▶ Ontology
▶ RDF
▶ UML
▶ URI
▶ URL
▶ URN
▶ Web Services
▶ Web Services and the Semantic Web for Life Science Data
▶ XML

Recommended Reading

1. Brazama A., Hingamp P., Quackenbush J., Sherlock G., Spellman P., Stoeckert C., Aach J., Ansorge W., Ball C.A., Causton H.C., Gaasterland T., Holstege F.C.P., Kim I.F., Markowitz V., Matese J.C., Parkinson H., Robinson A., Sarkans U., Schulze-Kremer S., Stewart J., Taylor R., Vilo J., and Vingron M. Minimum information about a microarray experiment (MIAME) – toward standards for microarray data. Nat. Genet., 29:365–371, 2001.
2. Cohen-Boulakia S., Biton O., Davidson S., Froidevaux C. BioGuideSRS: querying multiple sources with a user-centric perspective. Bioinformatics, 23(10):1301–1303, 2007.
3. Cohen-Boulakia S., Davidson S., Froidevaux C., Lacroix Z, and Vidal M.E. Path-based systems to guide scientists in the maze of biological resources. J. Bioinf. Comput. Biol., 4(5):1069–1095, 2006.
4. Galperin M.Y. The molecular biology database collection: 2007 update. Nucl. Acids Res., 35:D3–D4, 2007.
5. Lacroix Z., Raschid L., and Eckman B. Techniques for optimization of queries on integrated biological resources. J. Bioinf. Comput. Biol., 2(2):375–411, 2004.
6. Orchard S., Salwinski L., Kerrien S., Montecchi-Palazzi L., Oesterheld M., Stmpflen V., Ceol A., Chatr-aryamontri A., Armstrong J., Woollard P., Salama J.J., Moore S., Wojcik J., Bader G.D., Vidal M., Cusick M.E., Gerstein M., Gavin A.C., Superti-Furga G., Greenblatt J., Bader J., Uetz P., Tyers M., Legrain P., Fields S., Mulder N., Gilson M., Niepmann M., Burgoon L., De Las Rivas J., Prieto C., Perreau V.M., Hogue C., Mewes H.W., Apweiler R., Xenarios I., Eisenberg D., Cesareni G., and Hermjakob H. The minimum information required for reporting a molecular interaction experiment (MIMIx). Nat. Biotechnol., 25:894–898, 2007.
7. Taylor C.F., Paton N.W., Lilley K.S., Binz P.A., Julian R.K., Jones A.R., Zhu W., Apweiler R., Aebersold R., Deutsch E.W., Dunn M.J., Heck A.J.R., Leitner A., Macht M., Mann M., Martens L., Neubert T.A., Patterson1 S.D., Ping P., Seymour S.L., Souda P., Tsugita A., Vandekerckhove J., Vondriska T.M., Whitelegge J.P., Wilkins M.R., Xenarios I., Yates J.R., and Hermjakob H. The minimum information about a proteomics experiment (MIAPE). Nat. Biotechnol., 25:887–893, 2007.

Biological Networks

AMARNATH GUPTA
University of California-San Diego, La Jolla, CA, USA

Synonyms

Biological pathways; Molecular interaction graphs; Signal transduction networks; Transcriptional networks; Protein-protein interaction networks

Definition

A biological network is a graph-structured representation of binarized interactions among biological objects. Typically, the nodes in such a graph represent biological molecules, and the edges are labeled to represent different forms of interactions between molecules.

Example: A transcriptional network is a directed graph where a node represents either a protein (a transcription factor) or a region of the chromosome such that the edges can be constructed from the protein node to the chromosomal region. The edge in the graph represents that the protein can initiate the transcription (production of messenger RNA) process.

Key Points

A biological network is typically a node and edge attributed graph, where the edges can have different semantics depending on the kind of network. In some networks, the edges may be weighted, denoting, for instance, the probability of the interaction taking place. In some networks, like the protein-protein interaction graph, the edges are undirected. In some cases, like signal transduction networks, the edges represent the flow of time. Querying, integrating, and simulating are typical operations performed on biological networks.

Cross-references

▶ Graph Data Management in Scientific Applications

Recommended Reading

1. Baitaluk M., Qian X., Godbole S., Raval A., Ray A., and Gupta A. PathSys: integrating molecular interaction graphs for systems biology. BMC Bioinformatics, 7:55, 2006.
2. Eckman B.A. and Brown P.G. Graph data management for molecular and cell biology. IBM J. Res. Dev., 50(6):545–560, 2006.
3. Leser U. A query language for biological networks. Bioinformatics, 21(Suppl 2):ii33–ii39, 2005.

Biological Pathways

▶ Biological Networks

Biological Query Languages

▶ Query Languages for the Life Sciences

Biological Resource Discovery

ZOÉ LACROIX[1], CARTIK R. KOTHARI[2], PETER MORK[3], RAMI RIFAIEH[4], MARK WILKINSON[2], JULIANA FREIRE[5], SARAH COHEN-BOULAKIA[6]
[1]Arizona State University, Tempe, AZ, USA
[2]University of British Columbia, Vancouver, BC, Canada
[3]The MITRE Corporation, McLean, VA, USA
[4]University of California-San Diego, San Diego, CA, USA
[5]University of Utah, Salt Lake City, UT, USA
[6]University Paris-Sud, Orsay, France

Definition

Resources for the Life Sciences include various expedients including (access to) data stored in flat files or databases (e.g., a query form or a textual search engine), links between resources (index or hyperlink), or services such as applications or tools. *Resource discovery* is the process of identifying and locating existing resources that have a particular property. Machine-based resource discovery relies on crawling, clustering, and classifying resources discovered on the Web automatically. Resource discovery systems allow the expression of queries to identify and locate resources that implement scientific tasks and have properties of interest.

Historical Background

Resource selection relies on the identification of the resources suitable to achieve each task and the ability to compose the selected resources into a meaningful and efficient executable protocol. Metadata constitute the core information requisite to evaluate the suitability of Life Sciences resources to achieve a scientific task. Metadata critical to resource discovery include (i) resource publication, identification, and location, and

(ii) semantic and (iii) syntactic descriptions. First scientists need to be aware of existing resources. If academic publications such as Nucleic Acids Research (NAR) [7] or BMC Source Code for Biology and Medicine have provided valuable media where bioinformaticians may publish their resources, they require significant manpower to identify and evaluate the potential of each resource and compile and record their location and description for future use. Core resource description in a unified format accessible to scientists and machines alike and resource repositories contribute greatly to ease the problem of resource identification and location.

Foundations

Resource discovery relates to the activity of identifying a resource suitable to implement a particular task. Resource discovery relies on various metadata that specify the characteristics of resources thus allowing the mapping of the requirements to the resource specifications. The type of metadata chosen to represent resources will constrain resource discovery. For example, textual metadata drawn from a controlled, hierarchical vocabulary, support resource discovery by automatically expanding search terms to include more-specific terms. However, textual metadata are not sufficient when searching on specific criteria; for example, consider a researcher interested in finding datasets of *"MRI images for subjects between the ages of 18 and 24"*. Syntactic (formats) and semantic (concepts) metadata describe how a resource is organized and provide some insight into what type of information might be found in the resource while summary metadata specify the content of the resource. Although structural metadata are normally generated to help application developers understand how to interact with the resource, they can also be collected to support resource discovery. For example, in the caBIG framework, structural metadata are represented as common data elements. Each common data element references a common terminology (the NCI thesaurus in this case) and may also contain free text documentation describing that data element both providing a semantic representation of the resource. The metadata registry also maps common data elements to resources that provide instances of that element. An application developer searches the metadata registry by providing a collection of keywords; the registry returns a list of data elements that contain those keywords.

Resource discovery is the interface between resource metadata on one side and resource integration to implement complex scientific protocols (or workflows, queries, pipelines) on the other. Indeed the motivation for discovering a resource is drawn from the need to implement a scientific task. Most approaches to support resource discovery only locate one resource at a time, regardless of their future composition to implement complex workflows. In contrast, path-based guiding systems such as BioNavigation and BioGuide provide the ability to express resource discovery queries to identify resources that can be composed to express scientific protocols expressed as connected scientific tasks [4].

Key Applications

BioMoby is an open source, extensible framework that enables the representation, discovery, retrieval, and integration of biological data from distributed data repositories and analysis services. By registering their analysis and data access services with BioMoby, service providers agree to use and provide service specifications in a shared semantic space. The BioMoby Central registry now hosts more than a thousand services in the United States, Canada, and several other countries across the world. BioMoby uses a datatype hierarchy to facilitate the automated discovery of Web services capable of handling specific input datatypes. As a minimal Web based interface, the Gbrowse Moby service browser can be used by biologists to discover and invoke biological Web services from the Moby registry and seamlessly chain these services to compose multi-step analytical workflows. The process is data centric, relying on input and output datatype specifications of the services. The Seahawk client interface can infer the datatype of the input data files directly and immediately presents the biologist with a list of Web services that can process the input file. Users of the Seahawk interface are relieved of the necessity to familiarize themselves with datatype hierarchies, and instead are free to concentrate on the analytical aspects of their work. The BioMoby service encyclopedia provides a query interface to the repository of services. MOBY-S Web Service Browser retrieves bioinformatics resources with respect to a data type. Additional interfaces to BioMoby services include registry browsers that provide access to the complete list of registered BioMoby services organized in a HTML page.

These interfaces are convenient when searching for services with respect to a specific data format (input), but they are not suitable when searching for services with respect to their scientific meaning rather than their format. Another critical limitation of the approaches occurs when no single service achieves the task. In order to allow the discovery of the services that can be used to express scientific protocols, combinations of services must be retrieved. Scientific data integration systems such as workflow systems [6] enable the composition and execution of bioinformatics services. Combining a workflow approach with a service representation that guarantees compatibility of data formats offers a great value to the scientist who has selected the services to use and wishes to combine them in an executable workflow. A resource is selected because it uses or produces the expected format (e.g., `FASTA`) rather than because it implements the expected scientific aim or because it is efficient. This is illustrated by the BioMoby plugin in Taverna. When a BioMoby service, e.g., '`DragonDB_TBlastN`', is included in a workflow its output format, e.g., `NCBI_BLAST_TEXT`, can be searched (brief search) against available formats to determine if it is an input to any other service registered in BioMoby [9]. The characterization of a resource provided by existing formats such as Web services does not include the level of metadata necessary to evaluate the suitability of resources beyond the description of its input and output. More advanced resource formats such as OWL-S, WSDL-S, SAWSDL, and BioMoby aim at providing a semantic layer to capture better what the resource does in addition to its input and output data formats. The semantic part of the resource registration allows the classification of resources into a hierarchy of classes thus enhancing resource discovery. For example, the semantic search method of the BioMoby plugin in Taverna traverses the object ontology and recursively extracts the parent nodes of that particular output object [9]. However existing approaches do not offer an interface that allows the discovery of services with respect to their scientific meaning expressed in an ontology. To overcome this difficulty, path-based guiding systems such as SemanticMap and BioGuide can be combined with integration platforms to allow the discovery of resources suitable to implement scientific workflows. For example, BioGuide extends SRS [5] to allow a unique interface to discover resources and express queries over integrated data sources [3].

Resource discovery can exploit further resource metadata to predict the outcome of a workflow execution and select the resource more likely to produce the expected output. For example, BioNavigation [4] exploits various statistical metadata combined with semantic data to rank resources with respect to the users' criteria. Resources are ranked with respect to their cardinality, the characterization of their entries (number of attributes), etc. A user interested in *retrieving as many genes involved in a particular disease* selects a path in a domain ontology together with the corresponding ranking criteria. BioNavigation returns a ranking of all implementations of the conceptual path [10].

BioSpider is a system that integrates biological and chemical online databases. Given a biological or chemical identifier, BioSpider produces a report containing physico-chemical, biochemical and genetic information about the identifier. Ngu et al. [11] proposed an approach to classify search interfaces by probing these interfaces and trying to match the control flow of the interface against a standard control flow. InfoSpiders is a multi-agent focused crawler specialized for biomedical information whose goal is to fetch information about diseases when given information about genes. The Adaptive Crawler for Hidden-Web Entry Points (ACHE) is a focused crawler specialized for locating searchable Web forms that serve as entry points to online databases and Web services. Context-Aware Form Clustering (CAFC) is a clustering approach that models Web forms as a set of hyperlinked objects and considers visible information in the form context – both within and in the neighborhood of forms – as the basis for similarity comparison. A repository of scientific resources was automatically compiled using the approach [1].

Future Directions

Biological resource discovery remains a critical issue for the Life Sciences. The development of a system to support resource discovery is directly constrained by the information pertaining to scientific resources made available to the users as well as the formats designed to represent these rich metadata. For these reasons research on resource discovery for Life Sciences still is in its infancy. Scientists dramatically need assistance at each level of the process from the identification of the resources that best would meet the experimental requirements to the actual composition of the resources in an executable workflow. Future developments include the design of systems that combine various orthogonal aims for resource selection such as semantics (what the resource does), statistics (prediction of the result), syntax (schema mapping for resource compsition), performance (efficiency), quality, etc.

Data Sets

NAR http://nar.oxfordjournals.org/

BMC Source Code for Biology and Medicine http://www.scfbm.org/home

Bioinformatics Links Directory http://bioinformatics.ca/links_directory/

Semantic Map for Structural Bioinformatics http://bioserv.rpbs.jussieu.fr/SBMap/

Automatically compiled list of biological resources http://formsearch.cs.utah.edu

Open Biomedical Ontologies (OBO) http://obofoundry.org/

Next Generation Biology Workbench (Swami) http://www.ngbw.org

URL to Code

caBIG http://cabig.nci.nih.gov/

BioMOBY http://biomoby.org/

SOAPlab http://www.ebi.ac.uk/Tools/webservices/soaplab/overview

SemanticMap http://bioinformatics.eas.asu.edu/

myGRID http://www.mygrid.org.uk/

Taverna http://taverna.sourceforge.net/

MOBY-S http://mobycentral.icapture.ubc.ca/

BioNavigation http://bioinformatics.eas.asu.edu/

BioGuide http://bioguide-project.net/

Seahawk http://biomoby.open-bio.org/

Remora http://lipm-bioinfo.toulouse.inra.fr/remora/cgi/remora.cgi

Kepler http://www.kepler-project.org/

BioSpider http://biospider.ca

InfoSpiders http://www.informatics.indiana.edu/fil/IS/

Cross-references

▶ Benchmark
▶ Biological Metadata Management
▶ Dublin Core
▶ Graph Management in the Life Sciences
▶ HTTP
▶ Metadata
▶ Ontology
▶ RDF
▶ UML
▶ URI
▶ URL
▶ URN
▶ Web Services
▶ Web Services and the Semantic Web for Life Science Data
▶ XML

Recommended Reading

1. Barbosa L., Tandon S., and Freire J. Automatically Constructing a Directory of Molecular Biology Databases. In Data Integration in the Life Sciences, LNCS, Vol. 4544, 2007, pp. 6–16.
2. Clark T., Martin S., and Liefeld T. Graphically Distributed Object Identification for Biological Knowledge Bases. Brief. Bioinformat., 5(1):59–70, 2004.
3. Cohen-Boulakia S., Biton O., Davidson S., and Froidevaux C. BioGuideSRS: q uerying multiple sources with a user-centric perspective. Bioinformatics, 23(10):1301–1303.
4. Cohen-Boulakia S., Davidson S., Froidevaux C., Lacroix Z., and Vidal M.E. Path-based systems to guide scientists in the maze of biological resources. J. Bioinformat. Comput. Biol., 4(5):1069–1095, 2006.
5. Etsold T., Harris H., and Beaulah S. Bioinformatics: Managing Scientific Data. Chapter 5 – SRS: An Integration Platform for Databanks and Analysis Tools in Bioinformatics, pp. 109–146. Z. Lacroix and T. Critchlow (eds.). Morgan Kaufmann, Los Altos, CA, 2003.
6. Fox G.C. and Gannon D. (eds). Concurrency and Computation: Practice and Experience, Special Issue: Workflow in Grid Systems, 2006.
7. Galperin M.Y. The Molecular Biology Database Collection: 2007 update. Nucl. Acids Res., 35:D3–D4, 2007.
8. Good B.M. and Wilkinson M.D. The Life Sciences Semantic Web is Full of Creeps! Brief. Bioinformat., 7(3):275–286, 2006.
9. Kawa, E.A., Senger M., and Wilkinson M.D. BioMoby extensions to the Taverna workflow management and enactment software BMC Bioinformat., 7:523, 2006.
10. Lacroix Z., Raschid L., and Eckman B. Techniques for optimization of queries on integrated biological resources. J. Bioinformat. Computat. Biol., 2(2):375–411, 2004.
11. Ngu A.H.H., Rocco D., Critchlow T., and Buttler D. Automatic discovery and inferencing of complex bioinformatics web interfaces. World Wide Web, 8(4):463–493, 2005.
12. Wolstencroft K., Alper P., Hull D., Wroe C., Lord P., Stevens R., and Goble C. The myGrid Ontology: Bioinformatics Service Discovery. Int. J. Bioinformat. Res. Appl., 3(3):303–325, 2007.

Biological Sequences

Amarnath Gupta
University of California-San Digeo, La Jolla, CA, USA

Synonyms

DNA sequences; Protein sequence

Definition

A biological sequence is a sequence with a small fixed alphabet, and represents a naturally occurring or experimental generated fragment of genetic or protein material or any intermediate product (like the messenger RNA).

Example: A DNA fragment has the 4 character alphabet 'A', 'C', 'T', 'G'. Chromosomes are long strings over this alphabet.

Key Points

Biological sequences can be long. A full chromosome may have millions of characters. Therefore development of proper storing and indexing strategies is very important for fast retrieval. Suffix tree based indexes have been used successfully for long biological sequences. Further, approximate string matching techniques with potential deletions and insertions are important for biological sequences. BLAST is a well known algorithm used for approximate matching and ranking of biological sequences.

Cross-references

▶ Index Structures for Biological Sequences

▶ Query Languages and Evaluation Techniques for Biological Sequence Data

▶ Query languages for the life sciences

Recommended Reading

1. Brown A.L. Constructing genome scale suffix trees. In Proc. 2nd Asia-Pacific Bioinformatics Conference, 2004.
2. Hunt E., Atkinson M.P., and Irving R.W. A database index to large biological sequences. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 139–148.
3. Phoophakdee B. and Zaki M.J. TRELLIS +: an effective approach for indexing genome-scale sequences using suffix trees. In Proceedings of the Pacific Symposium on Biocomputing (online proceedings), 2008, pp. 90–101.
4. Tian Y., Tata S., Hankins R.A., and Patel J.M. Practical methods for constructing suffix trees. VLDB J., 14(3): 281–299, 2005.

Biomedical Data Annotation

▶ Biomedical Data/Content Acquisition, Curation

Biomedical Data/Content Acquisition, Curation

Nigam Shah
Stanford University, Stanford, CA, USA

Synonyms

Biomedical data annotation

Definition

The largest source of biomedical knowledge is the published literature, where results of experimental studies are reported in natural language. Published literature is hard to query, integrate computationally or to reason over. The task of reading published papers (or other forms of experimental results such as pharmacogenomics datasets) and distilling them down into structured knowledge that can be stored in databases as well as knowledgebases is called curation. The statements comprising the structured knowledge are called annotations. The level of structure in annotation statements can vary from loose declarations of "associations" between concepts (such as associating a paper with the concept "colon cancer") to statements that declare a precisely defined relationship between concepts with explicit semantics. There is an inherent tradeoff between the level of detail of the structured annotations and the time and effort required to create them. Curation to create highly structured and computable annotations requires PhD level individuals to curate the literature. In the molecular biology research community, this task is performed primarily by curators employed by genome databases such as the saccharomyces genome database [7]. In the biomedical research community this task is performed by curators employed by community portals such as AlzForum for Alzheimer's research [9] and PharmGKB for pharmacogenomics [31]. In the medical community such curation is still an ignored task, with some groups, such as RCTBank [26], pioneering the effort to curate clinical trial reports.

Historical Background

In the biomedical domain, curation began with the formation of cDNA, EST and gene sequence databases such as GenBank. Initially, curation was restricted to the task of assigning a functional annotation (usually in free text) to a sequence being submitted to GenBank. Scientists performing the experiments and submitting the data performed the task on their own. With the rise in the amount of sequence data and subsequently data on the function, structure and cellular locations of gene products along with the formation of communities of researchers around specific model organisms, the task of curation gradually became centralized in the role of a curator at model organism databases. Interaction amongst the curators and leading scientists led to the creation of projects such as the gene ontology project [1] in 1998, which led to a systematic basis for creating annotations about the molecular function, biological process and cellular locations of gene products. In subsequent years, user groups formed around other kinds of data, such as microarray gene expression data, resulting in the creation of information models for structuring the metadata pertaining to high throughput experiments. Individual research groups, such as Ecocyc, have already maintained a high level of curation effort, particularly for Information about biological pathways; although it was the success of the gene ontology project that resulted in the widespread appreciation for the need of curated content. With the continued rise in the amount and diversity of biomedical data, the need for curation continues to increase; both in terms of the number of man-hours required and in the level detail desired in the resulting annotations.

Foundations

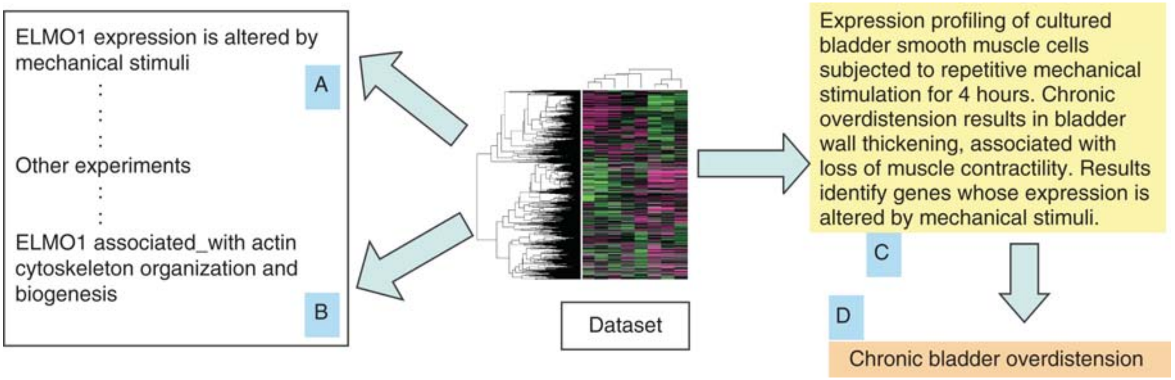
In the course of their work, biomedical investigators must integrate a growing amount of diverse information. It is not possible for scientists to bring together this large amount of information without the aid of computers. Researchers have turned to ontologies – which allow representation of experimental results in a structured form – to facilitate interoperability among databases by indexing them with standard terms as well as to create knowledge bases that store large amounts of knowledge in a structured manner. Ontologies provide researchers with both the structure into which experimental results, facts and findings have to be put into as well as the words (or terms) to be used in populating the structure with instances [8,14]. If the ontologies are well-designed, then the resulting knowledge bases can be used to retrieve relevant facts, to organize and interpret disparate knowledge, to infer non-obvious relationships, and to evaluate hypotheses posited by scientists.

Assertion annotations – assertions or statements about the relationships among biological entities and the processes in which they participate – are a crucial link between abstractions of experimental results and the theory (or theories) that explain the underlying results. The national center for biomedical ontology develops methods and tools that enable the easy creation of such assertion annotations [11]. However, even with tool support, the creation of assertion annotations is manual, hard and expensive. In addition to this annotation as assertion viewpoint, another predominant use of annotations is to provide metadata for datasets stored in databases. In this case, these metadata-annotations are not assertions about any biological entity but instead provide additional information about the experiment or dataset examining the biological entity. Such metadata-annotations provide information about experimental conditions, the disease that the dataset pertains to, the perturbation applied during the experiment, and so on. Metadata-annotations do not state a biological fact like an assertion based on interpretation of experimental results. These two kinds of annotations: (i) assertion annotations and (ii) metadata annotations, are highlighted in Fig. 1.

Curation is the process by which annotations (either assertions or metadata) are created. (The word "annotation" is also used by some as a verb to describe this process of curation to create annotations; this has lead to wide-spread confusion in the community about the meaning of annotation.) Until now curation has been largely a manual process requiring highly qualified individuals to read and interpret the text in published papers to create the annotations. It is important to note that even though curation is carried out by skilled personnel, different curators have different opinions on what "knowledge" is being reported in the paper. Increasingly, automated methods are being employed to assist in the curation task because of the fact that manual curation is unlikely to scale and keep pace with the growth of biomedical data and literature [3]. Curation is typically carried out using a tool that allows the curator to select relevant ontology terms and associate them with the entity being annotated. The same tool writes out the resulting annotations in a custom format.

Technical Issues

The Different Types of Expressivity of Ontologies/Vocabularies Used to Create the Annotations As discussed, annotations can range from simple terms that are "associated" with a particular resource to structured


ELMO1 expression is altered by mechanical stimuli
A
Other experiments
ELMO1 associated_with actin cytoskeleton organization and biogenesis
B
Dataset
Expression profiling of cultured bladder smooth muscle cells subjected to repetitive mechanical stimulation for 4 hours. Chronic overdistension results in bladder wall thickening, associated with loss of muscle contractility. Results identify genes whose expression is altered by mechanical stimuli.
C
D
Chronic bladder overdistension

Biomedical Data/Content Acquisition, Curation. Figure1. Shows the relationship of the assertion-annotations and metadata-annotations with datasets. The dataset is analyzed by a researcher to make a fine grained statement (a), which states the particular observation made in the dataset and reported in a publication. Several such statements get published in scientific papers. A curator, after reading a multitude of these papers creates an *assertion-annotation* (such as a GO annotation) as a summary statement (b) based on the fine grained statements. The dataset is also described by the researcher in terms of the disease studied, the cell lines used, the experimental conditions that existed etc. This description (c), comprises the *metadata-annotation* of the dataset, and is usually in natural language; although at times it is done using a CV. (d) shows the "tag" from a controlled vocabulary or ontology that can be assigned to this dataset upon processing the text description computationally.

assertions that use explicit logical relationships. Depending on the required use – that of creating assertion-annotations or creating metadata-annotations – the ontologies used in the annotation process need to have adequate expressivity in terms of the different relationships the user can use during the curation process. A detailed discussion on the kinds of relationships that are available in biomedical ontologies is can be found in [27]. The most widely used artifacts for annotation are controlled vocabularies (CVs). A CV provides a list of terms whose meanings are specifically defined. Terms from a CV are usually used for indexing records in a database. The Gene Ontology (GO) is the most widely used CV in databases serving biomedical researchers [1]. The GO provides terms that are "associated" with particular gene products for describing their molecular function (MF), biological process (BP) and cellular component (CC). Arguably, CVs provide the most return-on-effort in terms of facilitating database search and interoperability.

Storage Schemes and Data Models to Store These Annotations in Underlying Databases Most annotations when created initially are stored as flat text files. However in order for the annotations to be useful to researchers, they need to be stored in database systems that support efficient storage and querying. Naturally the database schema and the data model to which the annotations conform to becomes an important issue. Until recently the trend was to create a relational schema corresponding to the annotation model used for a particular curation workflow and each group created its own annotation model as well as schema. This lead to various "silo" databases that need to be mapped to one another. The need for such mapping lead to the creation of groups, such as BioPAX, which proposed "exchange formats" to map silo databases to one another [4]. Recently, semantic web technology is receiving a lot of attention in the biomedical community because of the promise of "automatic" interoperability if different groups use consistent identifier (URIs) as well as the Resource Description Framework (RDF) format to describe entities and resources in their annotations [22].

Techniques for Indexing the Curated Annotation for Retrieval Assuming the issue of creating annotations (manually or computationally) is adequately addressed, special attention needs to be paid to the appropriate indexing of annotations. For example, once a publication is annotated by associating it with the term *melanoma*, in order to ensure appropriate retrieval when someone searches for *skin neoplasms* it is essential to index the same paper with terms such as *skin neoplasm* (because *melanoma* is a kind of *skin neoplasm*). This can be accomplished by pre-computing all such inferred annotations or by real time query expansion using the hierarchy among the terms *melanoma* and *skin neoplasm*.

Workflow Aspects of the Curation Process As noted before, different curators can have different opinions on what "knowledge" is being reported in the paper. The level of this agreement is quantified by calculating inter-curator agreement using a variety of methods [6]. Using detailed curation guidelines, many projects achieve inter-curator agreement in the range of 85–90% and some as high as 94% [6]. Currently, curation is typically carried out using a tool, such as Phenote, (www.phenote.org.) that allows the curator to select relevant ontology terms and associate them with the entity being annotated. The same tool writes out the resulting annotations in a custom format. Usually the workflow for curation differs by organization and the kinds of source (such as published papers or medical records or clinical trials) being curated. Currently, there are no off-shelf workflow systems that provide a generic curation workflow. Increasingly curation is becoming web based and the tools used for curation are tied to a database which stores the annotations (See the Alzforum and SWAN projects for an example). There are also efforts to make curation collaborative, and several wiki-based projects such as wikipathways (www.wikipathways.org) are underway in the field.

Key Applications

The discovery process in biomedical research is cyclical; Scientists examine existing data to formulate models that explain the data, design experiments to test the hypotheses and develop new hypotheses that incorporate the data generated during experimentation. Currently, in order to advance this cycle, the experimentalist must perform several tasks: (i) gather information of many different types about the biological entities that participate in a BP, (ii) formulate hypotheses (or models) about the relationships among these entities, (iii) examine the different data to evaluate the degree to which his/her hypothesis is supported, and (iv) refine the hypotheses to achieve the best possible match with the data. In today's data-rich environment, this is a very difficult, time-consuming, and tedious task.

If existing data, information and knowledge are curated to create knowledge bases that store large amounts of knowledge in a structured manner [14,15] the resulting knowledge bases can be used to retrieve relevant facts, to organize and interpret disparate knowledge and to computationally evaluate hypotheses and model posited by scientists [2,18,19]. For example, EcoCyc is a comprehensive source of structured knowledge on metabolic pathways in E. Coli and can be used to reason about E. Coli metabolism. Reactome is a source of structured knowledge on BPs related to signal transduction, gene regulation and metabolism in eukaryotic organisms [13]. The creation of such knowledgebases requires that the task of curation be carried out with great detail and that the tradeoffs between the complexity of the annotation structure required and the curation overhead entailed by that be balanced. Understanding the curation cost is a significant factor in determining the feasibility of proposed knowledge-driven applications [12].

There are several public as well as private groups that curate biomedical literature and other data to create highly structured knowledge bases. A majority of these knowledge bases are centered on biological pathways and Ecocyc and Reactome are the leading examples. In Pharmacogenomics, PharmGKB is a resource that provides curated knowledge on the interactions between genotype and pharmacological effects of drugs [31]. The Semantic Web Applications in Neuroscience (SWAN) project is a resource providing curated knowledge on Alzheimer's research with a focus on capturing the evolving scientific discourse as the research progresses [16]. In the commercial sector, companies such as Ingenuity offer subscription access to curated literature content as well as curation-for-fee services.

Such curated and structured content is primarily used to interpret the results of high-throughput datasets in the light of prior knowledge. At the simplest level, coloring nodes of a pathway according the increase or decrease in their expression level in a particular assay is a widely used approach. Another widely used approach is that of counting the annotations, such as the association with a particular BP, assigned to a set of biological entities, such as genes deemed significant for a particular cancer, and analyzing for a statistically significant difference in the distribution of the annotation counts as compared to a reference such as the set of all the genes assayed.

The other key use of curated and structured content is to support computer aided reasoning; with the goal of inferring possible explanations for biological phenomena [25], for evaluating alternative explanations for biological phenomena[19], for automated question answering [29] and automatically constructing as well as extending existing structured descriptions of BPs such as pathways [23].

Future Directions

As the amount and diversity of data, information and knowledge rise in the biomedical domain, there is a recognized need to be able to compute with the existing knowledge [5]. As the use of ontology rises in the biomedical domain [30], the appreciation for the need of curated content is also rapidly increasing; along with the realization that manual curation is unlikely to keep pace with the needs of the community [3].

These trends have led several groups, such as the BioAI group at Arizona State University and the SWAN group, to propose the use of distributed and collaborative curation in an attempt to leverage the "wisdom of the masses" [10,17]. Collaborative curation holds tremendous promise for the field if the community can arrive at an agreed upon platform and formalism using which researchers can contribute structured content. The other clear future direction is the use of text-mining in the curation pipeline as a "force multiplier" to increase the productivity of existing curation efforts. The computational pharmacology group at University of Colorado is conducting exciting research in this direction. Both community-based collaborative curation as well as the use of text-mining to increase the efficiency of curation tools will be activities to follow closely for those interested in biomedical data acquisition and curation.

Cross-references

▶ Annotation
▶ Biological Metadata Management
▶ Curation

Recommended Reading

1. Ashburner M., et al. Gene ontology: tool for the unification of biology. Nat. Genet., 25(1):25–29, 2000.
2. Baral C., et al. A knowledge based approach for representing and reasoning about signaling networks. Bioinformatics, 20 (1):15–22, 2004.
3. Baumgartner Jr, et al. Manual curation is not sufficient for annotation of genomic databases. Bioinformatics, 23 (13):41–48, 2007.
4. BioPax-Consortium. BioPAX: Biological Pathways Exchange. Available from: http://www.biopax.org/ 2006.
5. Bodenreider O. and Stevens R. Bio-ontologies: current trends and future directions. Brief. Bioinform., 7(3):256–274, 2006.
6. Camon E.B., et al. An evaluation of GO annotation retrieval for BioCreAtIvE and GOA. BMC Bioinform., 6(Suppl 1):S17, 2005.
7. Cherry J.M., et al. SGD: saccharomyceas genome database. Nucleic Acids Res., 26(1):73–79, 1998.
8. Ciccaresse P., Wu E., and Clark T. An overview of the SWAN 1.0 ontology of scientific discourse. In Proc. 16th Int. World Wide Web Conference, 2007.
9. Clark T. and Kinoshita J. Alzforum and SWAN: the present and future of scientific web communities. Brief. Bioinform., 8(3):163–171, 2007.
10. Gao Y., et al. SWAN: a distributed knowledge infrastructure for Alzheimer disease research. J. Web Semantics, 4(3):222–228, 2006.
11. Gibson M. Phenote. Berkeley Bioinformatics and Ontology Project (BBOP), National Center for Biomedical Ontology, Lawrence Berkeley National Laboratory, 2007.
12. Hunter L. and Cohen K.B. Biomedical language processing: what's beyond PubMed? Mol. Cell., 21(5):589–594, 2006.
13. Joshi-Tope G., et al. Reactome: a knowledge base of biological pathways. Nucleic Acids Res., 33(Database Issue): D428–432, 2005.
14. Karp P.D. An ontology for biological function based on molecular interactions. Bioinformatics, 16(3):269–285, 2000.
15. Karp P.D. Pathway databases: a case study in computational symbolic theories. Science, 293(5537):2040–2044, 2001.
16. Katz A.E., et al. Molecular staging of genitourinary malignancies. Urology, 47(6):948–958, 1996.
17. Leslie M. Netwatch. Science, 312:1721, 2006.
18. Massar J.P., et al. BioLingua: a programmable knowledge environment for biologists. Bioinformatics, 21(2):199–207, 2004.
19. Racunas S.A., et al. HyBrow: a prototype system for computer-aided hypothesis evaluation. Bioinformatics, 20 (Suppl 1):257–264, 2004.
20. Reactome Curator Guide. http://wiki.reactome.org/index.php/Reactome_Curator_Guide
21. Rise of the Bio-Librarian – the field of biocuration expands as the data grow. http://www.the-scientist.com/article/display/23316/.
22. Ruttenberg A., et al. Advancing translational research with the Semantic Web. BMC Bioinform., 8(Suppl 3):S2, 2007.
23. Rzhetsky A., et al. GeneWays: a system for extracting, analyzing, visualizing, and integrating molecular pathway data. J. Biomed. Inform., 37(1):43–53, 2004.
24. Second International Biocuration Meeting, San Jose, CA, October 25–28, 2007. http://biocurator.org/Mtg2007/index.html.
25. Shrager J., et al. Deductive biocomputing. PLoS ONE, 2(4): e339, 2007.
26. Sim I., Olasov B., and Carini S. The Trial Bank system: capturing randomized trials for evidence-based medicine. AMIA Annu. Symp. Proc., 2003:1076, 2003.

27. Smith B., et al. Relations in biomedical ontologies. Genome Biol., 6(5):R46, 2005.
28. Spasic I., Ananiadou S., McNaught J., and Kumar A. Text mining and ontologies in biomedicine: making sense of raw text. Brief. Bioinform. 6(3):239–251, 2005.
29. Tari L., et al. BioQA. http://cbioc.eas.asu.edu/bioQA/v2/index.html, 2007.
30. The National Center for Biomedical Ontology. Available at: www.biontology.org, 2006.
31. Thorn C.F., Klein T.E., and Altman R.B. PharmGKB: the pharmacogenetics and pharmacogenomics knowledge base. Meth. Mol. Biol., 311:179–91, 2005.

Biomedical Image Data Types and Processing

Sameer Antani
National Institutes of Health, Bethesda, MD, USA

Synonyms

Data Types: Image, Video, Pixel, Voxel, Frame; Conceptual data types: Pixel, Point, Edge, Volume, Region of interest, Shape, Color, Texture, Feature; Format: Joint photographic experts group (JPEG), Digital imaging and communications in medicine (DICOM), JPEG2000, Imaging Technique: X-Ray, Magnetic resonance imaging (MRI), Computerized tomography (CT), Ultrasound, Positron emission tomography (PET), Nuclear magnetic resonance (NMR), Microscopy, Single photon emission computerized tomography (SPECT), Fluoroscopy; Image Processing: Compression, Wavelet compression, Functional mapping, Image reconstruction, 2D image processing, Texture analysis, Edge detection, 3D image processing, Surface detection, Image content analysis; Storage and Retrieval: Image databases, Content-based image retrieval (CBIR), Visual similarity, Feature indexing, Multimedia information retrieval

Definition

The entry term describes biomedical image types (X-Ray, CT, MR, PET) stored in a particular format (DICOM, JPEG) that can be processed for visual enhancement (windowing, leveling) or extraction of features for further processing as needed in specific applications (generate 3D volumes from 2D slices, Content-Based Image Retrieval (CBIR)).

Historical Background

Both image processing and databases have been studied for over four decades. Biomedical processing and storage systems have received significant attention within the last two decades. Imaging and image processing has gained significant importance in clinical medicine, biomedical research and education and correspondingly, biomedical image databases have also found increasing use in recent years. Images are still largely stored as flat files on file servers and made accessible via links stored in revelant database records. Significant progress has been made in image types, formats, and content being computed and stored in these databases. This information can help in processing and further use of these data. Image and image feature indexing is a topic of significant research interest with some specialized types are already in practical use.

Foundations

Imaging has taken on a very important role in clinical medicine, biomedical research, and education. Biomedical visual data are acquired using a variety of techniques: single frame images; 3D volumes composed of single frame images; and made and as time-synchronized multiple frames as video data. In addition, these data are acquired at varying scales ranging from gross anatomy to the cellular level. Each image data type has specific acquisition methods, set of image processing methods for feature extraction that aid in analysis for targeted purposes, compression and storage methods, and particular data handling methods [1]. The image database primarily serves as a file storage mechanism with various processes for analysis and retrieval traditionally included in utility applications. Image databases are typically found in practical use as “multimedia databases” or “multimedia information systems” in the form of Radiological Information Systems (RIS), Hospital Information Systems (HIS), and Picture Archiving and Communication Systems (PACS). Such systems link textual data to image data through file links stored in database records. Image databases imply use of image feature indexing strategies such as metric index trees, multidimensional data trees, spatial databases, and R-trees, for specialized use such as Content-Based Image Retrieval (CBIR) [8].

Image data types are challenging to define in standard terms such as integers, characters, strings, etc. An image is composed of pixels or in case of 3D images may be considered to composed of a set of elements of conceptual data type called voxels. Wikipedia (http://www.wikipedia.org) defines voxel as a portmanteau of words volumetric and pixel representing a unit element on a 3D image. Each such element (pixel or voxel) can be considered a complex data type as its content may be expressed using *n*-bits where *n* may be 8, 12, 16, 24, or 32. Typically 8-, 12-, and 16-bit images are gray scale images. A color pixel is typically 24-bits in depth comprising of three 8-bit channels for the additive color primaries (RED, GREEN, BLUE), though it is possible to have color images with other bit depths. This information is not natively stored in the image but needs to be exposed to the application through image metadata that may be stored in particular formats, such as a DICOM (http://dicom.nema.org/) on JPEG image header [6].

The images can be generated using a variety of techniques. Radiographic or X-Ray images, Computerized Tomography (CT), Magnetic Resonance images (MRI), Positron Emission Tomography (PET) images are examples of various imaging techniques. Techniques such as CT and MRI image the desired anatomical region in closely spaced sections. These sectional images can then be processed to create views along desired axes (axial, coronal, sagittal, oblique) as well as generate 3D volumetric data rendering.

Image processing is a term that includes functions and methods that focus on enhancement of images for improved human visualization or computer analysis, such as windowing, leveling, object edge detection, among others [3,9]. It also is synonymous with application of methods whereby features such as edges, textures, and surfaces, among others, can be computed for making measurements, computer-aided diagnosis, visual enhancement, identifying anatomical structures, determining unique image content signature, etc. For instance, using the above example of generating 3D volumetric data from 2D image slices, for a data set of MRI slices of the brain, it would be necessary to segment the edges from each 2D MRI image slice and register them with corresponding edges from the same anatomy in other slices. The next step would be to convert these edges into surfaces formed across these slices in order to generate 3D volumetric data.

With the increasing use of images in medical care and research, it becomes necessary make this data connect with other image and non-image data. The resulting database systems have evolved as PACS, RIS, and HIS and are commonly found in modern hospitals and medical centers. These database systems are capable of storing and retrieving text data, for example, a patient record containing test results and other medical history, along with image data. The systems may exist on a single computer, a local network of computers, or distributed over a wide area network. Variants of these systems developed for medical research studies can also correlate between different study participant information and keep track of longitudinal information.

In database processing it is often necessary to define the image to be of a particular type. This can assist in data and type verification as well as communication of semantics to other applications that may be using the data. Some database systems require storage of images in their native form as undefined BLOB data types while others, including most PACS, prefer to maintain references to image data files that are stored in traditional directory (folder) file structures. The choice between these approaches is largely determined by storage and computational efficiency and dependent on particular applications and solutions. In either scenario it is efficient to store the image metadata as database records. While this information may be available in image header files, it requires the additional step of accessing and opening each image file for any database operation involving images.

Image processing steps often result in features extracted from images. These features could be regions of pixels, measurements of color, texture, edges forming a shape, surfaces, etc. Each of these features could be standardized to be a data type or could use standard data types, e.g., a 3-channel color histogram could be represented using 3D arrays that could represent 3D histograms. Each such conceptual type may be used as a predefined data type. Other operations could include transforming the image or extracted characteristics from the spatial domain several into other domains through Fourier analysis or Wavelet transforms. Selecting these or other image processing methods is heavily dependent on the nature of the images and several methods are covered in [1,3,9]. Further, it may be necessary to compress the images in order to minimize data storage requirements or improve transmission efficiency over networks. These decisions must be made carefully in light of possible data loss found in typical implementations of common image compression methods such as JPEG or JPEG2000 [6].

In summary, biomedical image processing is critical to analysis and use of biomedical images for clinical medicine, research and education. These images may also have other associated images as well as text data. All this information is stored in biomedical databases that use a combination of image types, header information, image units, and content through the extracted features as data types. Images may be indexed through multidimensional indexing trees or be linked to flat files stored in a folder or accessed via a file server.

Key Applications

Multimedia Medical Information Systems: Medical Information Systems, like the PACS, hold medical data about patients, medical research study participants, etc. This medical data is typically heterogeneous comprising of electronic medical records containing the medical history, clinical notes, lab reports, and any acquired images. The text data can be fielded or exist as a block of free text. The image data can be from various sources and in a variety of formats. As such, a PACS can be considered a special type of a Multimedia Medical Information System. Another example of a Multimedia Medical Information System is the Multimedia Database Tool (MDT) being developed at the National Library of Medicine (NLM), part of the National Institutes of Health (NIH). The MDT is a Web-based system [5] with a MySQL back-end database that enables retrieval of text and image data in response to specific queries. For example, from a database of a cervical cancer study being conducted by the National Cancer Institute (NCI) one could query for "all women with cervical intraepithelial neoplasia 3 whose cytologic results are atypical squamous cells of undetermined significance."

The MDT was developed to access, evaluate, and collect information from thousands of uterine cervix images or cervigrams for cancer studies. It is one of two systems developed to work with these images: the boundary marking tool (BMT) for marking areas of particular importance in the images and the MDT for Internet dissemination of the images and to relate them with text data and information collected with the BMT. The MDT has system architecture capable of deploying color images and related information on the Web with minimal reprogramming. It has the flexibility to accept new datasets with the required customization performed at the level of a database administrator, rather than a programmer. It also has the capability of querying a database of text and images over the Web, of showing the query results consisting of multiple images and text data, and of exporting these results for statistical analysis. Additionally, the MDT is designed for data collection from remote users; given adequate password protection and anonymized data to assure patient privacy, experts worldwide will be able to access the data resource. Through the Internet, they will evaluate images and test data, perform analysis of the information, and record their evaluations in a central database. Additionally, because of its architecture, the MDT system can support a broad class of text and image databases. Therefore, the MDT is designed to grow if additional groups wish to merge their data on cervical cancer or to use it to manage their own multimedia collections.

Content-Based Image Retrieval (CBIR): While the MDT allows querying of text and images using text keywords and structured SQL-like queries, an alternative complementary form of image-based querying has gained significant research interest in recent years. This approach, called Content-Based Image Retrieval (CBIR), uses distinguishing features extracted from images to serve as indices. These features are then used to find images similar to an image query. For example, in the Spine Pathology Image Retrieval System (SPIRS) [4], developed at NLM, the boundary edges of individual vertebra are used to query the X-ray images of the spine captured as a part of the Second National Health and Nutrition Examination Survey (NHANES II). This shape feature was chosen because the pathology of interest is expressed along the vertebral boundary seen in the spine imaged on the sagittal plane. Perturbations along the boundary are indicative of pathology. In contrast, the set of cervigrams from the NCI cancer study is an example of an image class where color, texture, and location information are much more important than edge information. In this latter case, the pathology of interest is acetowhitened regions [10] on the cervical wall.

Finding similar images tends to be a very subjective matter and is heavily dependent on the extracted features from the images. Additionally, it is also dependent on the level of detail extracted. For example, one measure of image similarity is overall appearance of the image, which in case of X-Ray images, can be characterized by a histogram of pixel intensity levels in the image. While this *global* measure may be sufficient for overall similarity, it is insufficient in expressing *local* pathology that can only be captured by feature extraction within the region of interest. An intelligently implemented hierarchical strategy works better in a heterogeneous collection of images.

Given the subjective nature of image content and human perception, it is challenging to evaluate systems through system characteristics or reported performance measures alone. These results are sensitive to the kinds of image data that the system is operating on, extracted features, query capability, and several other gaps that need to be overcome for developing an "ideal" system. A framework for these gaps is discussed in [2]. Other medical systems are reviewed in [7]. The Cross Language Evaluation Forum (CLEF) benchmarking competition has evaluates image classification and image retrieval in a biomedical setting on an annual basis (CLEF-Campaign, http://www.clef-campaign.org) and permits use of text data commonly found with medical images to improve usability. Due to the transient nature of academic systems, and lack of detail available in commercial systems, such a venue provides valuable metrics for comparing various systems and assessing the state-of-the-art.

Cross-references

▶ Annotation-Based Image Retrieval
▶ 2D Shape Retrieval
▶ Feature-Based 3D Object Retrieval
▶ Feature Extraction for Content-Based Image Retrieval
▶ Image
▶ Image Content
▶ Image Database
▶ Image Management for Biological Data
▶ Image Metadata
▶ Image Representation
▶ Image Retrieval
▶ Image Retrieval and Relevance Feedback
▶ Image Salient Points and Features
▶ Image Segmentation
▶ Indexing and Similarity Search
▶ Indexing Metric Spaces
▶ Lossless Data Compression
▶ Low Level Image Content Analysis (Color, Texture Shape)
▶ Multimedia Data
▶ Multimedia Databases
▶ Multimedia Data Indexing
▶ Multimedia Data Storage
▶ Radiology Information Systems
▶ Relevance Feedback for Content-Based Information Retrieval
▶ Spatial Data Types
▶ Visual Content Analysis

Recommended Reading

1. Beutel J., Kundel H.L., and Van Metter R.L. (eds.). Handbook of Medical Imaging. Vols. 1, 2, and 3. SPIE Press, Bellingham, WA.
2. Deserno T.M., Antani S., Long R. Ontology of Gaps in Content-Based Image Retrieval. J Digital Imaging, February 2008.
3. Gonzales R.C. and Woods R.E. (eds.). Digital Image Processing (2nd edn.). Prentice Hall, Upper Saddle River, NJ.
4. Hsu W., Antani S., Long LR. SPIRS: a framework for content-based image retrieval from large biomedical databases. Medinfo, 12 (Pt 1):188–92, 2007.
5. Jeronimo J., Long L.R., Neve L., Bopf M., Antani S., Schiffman M. Digital tools for collecting data from cervigrams for research and training in colposcopy. J. Lower Genital Tract Dis., 10(1):16–25, 2006
6. Joint Photographic Experts Group (JPEG) http://www.jpeg.org/. American Medical Information Association (AMIA 2007), Chicago, November 2007, pp. 826–830.
7. Müller H., Michoux N., Bandon D., Geissbuhler A. A Review of Content-Based Image Retrieval Systems in Medical Applications – Clinical Benefits and Future directions. Int J Med Inform., 73(1):1–23, 2004.
8. Samet H. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufman. San Francisco, CA, 2006.
9. Sonka M., Hlavac V., and Boyle R. (eds.). Image Processing, Analysis, and Machine Vision (2nd edn.). PWS Publishing, Washington, DC.
10. Xue Z., Antani S.K., Long L.R., Jeronimo J., Thoma G.R. Investigating CBIR techniques for cervicographic images. In Proc. 2007 Annual Symposium of the American Medical Information Association, 2007, pp. 826–830.

Biomedical Informatics

▶ Implications of Genomics for Clinical Informatics
▶ Taxonomy: Biomedical Health Informatics

Biomedical Literature

▶ Biomedical Scientific Textual Data Types and Processing

Biomedical Scientific Textual Data Types and Processing

Li Zhou[1], Hua Xu[2]
[1]Partners HealthCare System Inc., Boston, MA, USA
Harvard Medical School, Boston, MA, USA
[2]Columbia University, New York, NY, USA

Synonyms

Scientific knowledge bases; Biomedical literature; MEDLINE/PubMed; Curation; Annotation; Information retrieval; Information retrieval models/metrics/operations; Indexing; Semi-structured text retrieval; Text extraction; Text mining; Web search and crawling

Definition

Vast amounts of biomedical scientific information and knowledge are recorded in text [1,7]. Various scientific textual data in the biomedical domain may generally be disseminated through the following resources [7,11]: biomedical literature (e.g., original reports and summaries of research in journals, books, reports, and guidelines), biological databases (e.g., annotations in gene/protein databases), patient records (e.g., clinical narrative reports), and web content.

A variety of techniques have been applied to identify, extract, manage, integrate and exploit knowledge from biomedical text. Some researchers [11] divide biomedical scientific textual data processing into three major activities as shown in Figure 1: information retrieval (IR), information extraction (IE), and text mining (TM).

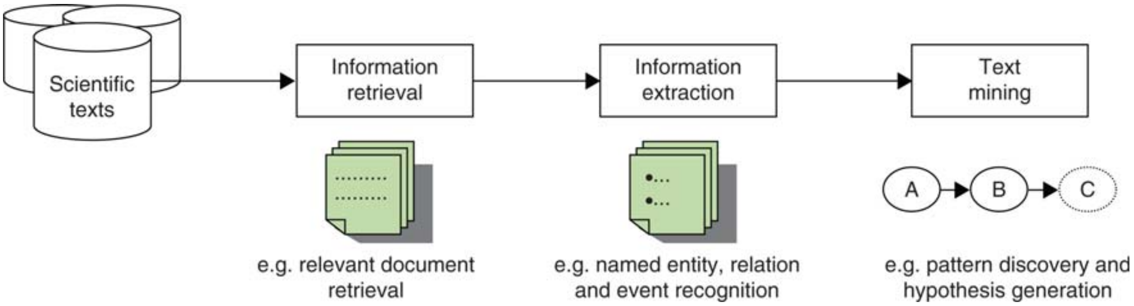
Information retrieval [2,11] is the science of indexing and searching for information particularly in text or other unstructured forms. The aim of IR is to identify relevant documents in response to a particular query, which forms the basis of any knowledge discovery process.

Information extraction [11] aims to identify and extract categorized or semantically well-defined data (entities, relations or events) from text documents in a certain domain, as well as to create structured knowledge bases that can be accessed by other informatics applications. Typical subtasks of IE are named entity, relation and event recognition (e.g., recognition of protein names and interactions between proteins), coreference (e.g., identifying whether a chain of noun phrases refer to the same object), and terminology extraction (e.g., finding the relevant terms for a given corpus).

Text mining (TM) [2,6] is the process of discovering and extracting interesting and non-trivial patterns and knowledge from unstructured text data. The primary goal of TM is to retrieve knowledge that is hidden in text, and to present the distilled knowledge to users in a concise form. Typical subtasks of TM may include pattern discovery, hypothesis generation, correlation discovery, etc. However, some researchers give a broader definition of TM which overlaps with IR and IE on certain tasks such as text classification, text clustering and named entity recognition.

Historical Background

Before the invention of computers, results of biomedical research have been published as journal or conference prints for a long time. Bibliographic databases that typically contained references to literature on library shelves was the first application of using computers to improve library service. MEDLINE (Medical Literature Analysis and Retrieval System Online) is the U.S. National Library of Medicine's (NLM) premier bibliographic database that contains over 16 million references to journal articles in life sciences with a concentration on biomedicine. As an online interactive searchable bibliographic database, MEDLINE was introduced in 1971 by NLM, to replace its previous version called MEDLARS (Medical Literature Analysis and Retrieval System). In 1997, PubMed was developed by the National Center for Biotechnology Information (NCBI) at the NLM, to provide free and efficient access to MEDLINE through the World Wide Web. Currently, MEDLINE/PubMed is probably the best-known biomedical literature reference database. The use of high-throughput experimental technologies has dramatically increased the pace of biomedical knowledge discovery. In 2006, over 623,000 references to published articles were added to the MEDLINE database. A large amount of effort has been spent on improving the performance of IR on the MEDLINE database. A distinctive feature of MEDLINE is that the records are indexed with NLM's Medical Subject Headings (MeSH).


Scientific texts
Information retrieval
Information extraction
Text mining
A
B
C
e.g. relevant document retrieval
e.g. named entity, relation and event recognition
e.g. pattern discovery and hypothesis generation

Biomedical Scientific Textual Data Types and Processing. Figure 1. Major stages of processing biomedical scientific textual data and relevant subtasks [11].

During the past decade, more and more full-text biomedical publications have become available on the Internet, though most of them have restricted access. In 1999, a bold new initiative called PubMed Central (PMC) was designed at the U.S. National Institutes of Health (NIH) to provide a central repository for literature in the life sciences with open access. To date, there are more than 300 journals that have joined PMC and they provide free access to their publications. BioMed Central, a commercial publisher, also provides free access to papers published in their journals.

Various text processing methods, such as natural language processing (NLP) and machine learning (ML) technologies, have been extensively studied in the domain of computer and information science. However, they have not been widely applied to biomedical text before the 1990s, largely due to the lack of available biomedical text. Starting at the mid 1990s, text processing technologies have been gradually applied to biomedical text on different tasks, such as information retrieval, biomedical entity recognition, text clustering and classification, and knowledge discovery.

Foundations

As mentioned above, biomedical scientific textual data processing applies methods and technologies from multiple disciplines, including linguistics, computer science, statistics, and so on. In general, the major stages of processing scientific textual data to exploit rich knowledge include retrieval of relevant documents, extraction of named entities and relations, and discovery of new knowledge. However, some processes may not follow the exact steps. This entry adopts a classification by Natarajan et al. [11] on major constituent technologies for knowledge discovery in text (see Fig. 1). Scientific fundamentals for each stage will be discussed in the following sections.

Information Retrieval

Conventional IR methods are often based on keyword queries, using Boolean logic, vector space models or probabilistic models [1,7]. One of the simplest forms of IR is to search keywords in documents that are indexed by a set of keywords. Search algorithms are used to identify the relevant documents based on the number of index keywords that match query keywords. One disadvantage of this approach is that the documents are determined either relevant or irrelevant. There is no further ranking. Vector space model is an algebraic model for representing text documents. When applying the model to IR, both the query and documents are represented as vectors, whose dimensions correspond to different terms in the query or documents. There are different methods to compute the weight of terms in the vectors and the tf-idf weighting is one of the best known schemes. Relevancy rankings of documents to a query can be determined by calculating the document similarities between the query and documents, via measurements such as cosine similarity of two vectors. Probabilistic models treat the process of document retrieval as a probabilistic inference and similarities are computed as probabilities that a document is relevant for a given query. An advantage of probabilistic model is that documents are ranked in decreasing order of their probability of being relevant to the query.

Information Extraction

Approaches to named entity recognition generally fall into three categories: lexicon based, rule based and statistically-based [1]. For example, part-of-speech tagging, inductive rule learning, decision trees, Bayesian model, support vector machines, as well as combined methods have been applied to this problem. For discovering relationships among entities, variant techniques have been used. Shallow parsing is often used to focus on specific parts of the text to analyze predefined words such as verbs and nouns. Some systems combine natural language processing and co-occurrence techniques, while others apply machine learning techniques.

Text Mining

Text classification and clustering are the most widely used techniques in biomedical text mining. While text classification is a form of learning from pre-classified examples, text clustering is referred to as unsupervised learning. Bayesian models were widely used in the early days. In recent years, more advanced machine learning methods, such as k-nearest neighbors, artificial neural networks, support vector machines, expectation maximization, and fuzzy clustering have been used. Logical inference models [13] have been applied to hypothesis generation which attempts to uncover relationships that are not present in the text but instead are inferred by the other existing relationships. There are a variety of techniques for knowledge discovery from biomedical text using graphs and knowledge models.

Key Applications

Information retrieval technologies have been used extensively to help users to find relevant articles that they are interested in. MEDLINE/PubMed has used various methods to improve the performance of searches. It provides keyword-based Boolean search to allow users to search by keywords, as well as document-based search, which implements the vector space model and could find documents for similar topics. With the availability of full-text articles online, more IR applications have tried to search full-text articles for detailed information. For example, the focus of the 2006 genomics track of the Text Retrieval Conference (TREC) [8] was to retrieve answers for biological questions from full text articles. The European Bioinformatics Institute (EBI) at the European Molecular Biology Laboratory (EMBL) has developed a biomedical information retrieval system called "CiteXplore," which combines literature search with text mining tools for biology. It also links biomedical literature sources to existing bioinformatics databases, such as SwissProt.

Although most of biomedical text processing tools are still in the research stage, some of them have shown potential uses. Many information extraction systems have been used to build knowledge bases from biomedical literature. Different approaches have been reported to extract relations among biomedical entities of interest (e.g., gene/protein). GENIES (GENomic Information Extraction System) [4] is an NLP-based system that extracts molecular pathways from literature. It semantically parses sentences into a structured form for relation extraction. PASTA (Protein Active Site Template Acquisition) [5] is a system that uses manually created templates to extract relationships between amino acid residues and their functions within a protein. The PreBIND [3] system uses Support Vector Machine (SVM) technology to locate protein-protein interaction data in the literature, thus to facilitate the curation process for protein databases. IR and IE systems are also combined to build more sophisticated systems to help specific tasks in biology, such as biological database curation tools that can help curators find related articles and identify critical findings from biological articles [14]. The iHOP [9] system extracts protein-relationship from the literature. It also includes advanced search modes for discovery and visualization of protein-protein-interaction network [9].

Another potential application of text mining tools is to discover new knowledge from literature, for example, helping biomedical researchers to generate new research hypotheses. ARROWSMITH and BITOLA [9,14] are two online tools that provide the function of literature-based knowledge discovery. ARROWSMITH detects indirect associations between concepts that are not directly linked in the literature. BITOLA is designed for disease candidate gene discovery by mining the bibliographic database MEDLINE.

Cross-references

- ▶ Data Mining
- ▶ Data, Text, and Web Mining in Healthcare
- ▶ Information Retrieval
- ▶ Text Mining of Biological Resources
- ▶ Text Mining

Recommended Reading

1. Chen H., Friedman C., Hersh W., and Fuller S.S. (eds.) Medical Informatics: Knowledge Management and Data Mining in Biomedicine. Springer, Secaucus, NJ, 2005.

2. Cohen A.M. and Hersh W.R. A survey of current work in biomedical text mining. Brief Bioinform., 6(1):57–71, 2005.
3. Donaldson I., Martin J., deBruijn B., Wolting C., Lay V., Tuekam B., Zhang S., Baskin B., Bader G., Michalickova K., et al. PreBIND and Textomy – mining the biomedical literature for protein-protein interactions using a support vector machine. BMC Bioinformatics, 4:11, 2003.
4. Friedman C., Kra P., Yu H., Krauthammer M., and Rzhetsky A. GENIES: a natural-language processing system for the extraction of molecular pathways from journal articles. Bioinformatics, 17 (Suppl 1):S74–S82, 2001.
5. Gaizauskas R., Demetriou G., Artymiuk P.J., and Willett P. Protein structures and information extraction from biological texts: the PASTA system. Bioinformatics, 19(1):135–143, 2003.
6. Hearst M. Untangling text data mining. In Proc. 27th Annual Meeting of the Assoc. for Computational Linguistics, 1999.
7. Hersh W. Information Retrieval: A Health and Biomedical Perspective. Springer, NY, 2003.
8. Hersh W., Cohen A., Roberts P., and Rekapalli H.K. TREC 2006 genomics track overview. In Proc. TREC 2006. Available at: http://trec.nist.gov/pubs/trec15/papers/GEO06. OVERVIEW. pdf
9. Hoffmann R. and Valencia A. A gene network for navigating the literature. Nat. Genet., 36(7):664, July 2004.
10. Hristovski D. and Peterlin B. Literature-based disease candidate gene discovery. In Proc. Medinfo. American Medical Informatics Association, Bethesda, 2004, p. 1649.
11. Natarajan J., Berrar D., Hack C.J., and Dubizky W. Knowledge discovery in biology and biotechnology texts: a review of techniques, evaluation strategies, and applications. Crit. Rev. Biotechnol., (25):31–52, 2005.
12. Smalheiser N. and Swanson D. Using ARROWSMITH: a computer-assisted approach to formulating and assessing scientific hypotheses. Comput. Methods Programs Biomed., 57:149–153, 1998.
13. Swanson D.R. Complementary structure in disjoint science literatures. In Proc. 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1990, pp. 280–289.
14. Yeh A.S., Hirschman L., and Morgan A.A. Evaluation of text data mining for database curation: lessons learned from the KDD Challenge Cup. Bioinformatics, 19 (Suppl 1):i331–i339, 2003.

Biostatistics and Data Analysis

MEHMET M. DALKILIÇ
Indiana University, Bloomington, IN, USA

Definition

Biostatistics is the application of probability and statistical techniques to the biological sciences. Probability has played a significant role in areas like genetics where combinatorics validate conjectures about the relationships of genes and the environment. Recently, combinatorics has become one of the main approaches to solving problems *e.g.*, motif discovery in bioinformatics. In the nineteenth century, well-known biologists like Herman von Helmholtz advocated that biological phenomenon could be understood using techniques in the physical sciences (remnants of this view still are present today). That approach, together with the "vitalism" movement [3] impeded the use of statistics. By the beginning of this century, however, statistics has become *de rigueur* in virtually all biological publications.

Historical Background

Although statistics as a mathematical area can be traced further, biostatistics is often associated with the work of Francis Galton (1822–1911). His major contribution was demonstrating that statistical methods could be beneficial in biology. Carrying on this tradition were Karl Pearson (1857–1936) and R.A. Fisher (1890–1962). While a number of standard applications and techniques have not changed for more than half-a-century, the availability of computing has made some (heretofore infeasible) techniques available. The classic text in this area is by Sokal and Rohlf, "Biometry" [5] now in its third edition. It should be pointed out that the demarcation often cited between "frequentists" [6] and "Bayesians" [1] is fairly well evident in biostatistics. In the former, physical, repeatable, identical, and random experiments can be associated to a mathematical limit *e.g.*, $P(heads) = \lim_{flips\rightarrow\infty} \frac{\#heads}{flips} = 1/2$ is the probability of getting heads when flipping a coin. Bayesians, on the other hand presume probability to be a degree of belief (or subjective probability). This can be written as *posterior* $\propto$ *prior* $\times$ *likelihood*, in symbols $P(H|E) \propto P(H)$ $P(E|H)$. The distinctive feature is that Bayesian statisticians will associate values with the *posterior*, *hypothesis*, and *prior* (likelihood), whereas non-Bayesian statisticians will only consider the hypotheses in constrained settings. Classical statistics is often the primary choice. However, Bayesian techniques are becoming increasingly popular in systems biology–biology that integrates and evaluates disparate data usually in the form of very large graphs. In these graphs, nodes are typically genes and edges evidence of relationship.

Foundations

As explained eloquently in [5], there is a deeper philosophical debate centered on whether biological phenomena can ultimately be modeled using deterministic approaches. That debate aside, the use of statistics is continuing to grow as the amounts of biological data grow. Indeed, recent technologies (high-throughput) are producing several orders of magnitude more data in comparison to traditional approaches. Statistical tools, from this perspective, becomes a necessity. Furthermore, a growing number of biologists now believe that there is benefit in examining data normally not studied within one's own specialized domain–this is called a systems biology. While the foundations for systems biology are still under development it is clear that it will rely heavily on Bayesian reasoning. Typically the disparate experimental, textual, *etc.* data are structured as a directed, acyclic graph where nodes are random variables and edges are effects. The basic attempt is to discover sets of independent variables and form a better understanding of the joint probability. For example, one might take the some 14,000 *Drosophila m.* genes and presume a joint distribution $P(X_1,X_2,...,X_{14000})$ where X_i represents the probability that gene i has some expression level. Bayesians build large graphs relying on conditional probabilities and so-called "separations" that expose which random variables are independent of one another. They then examine the behavior of the graph under particular conditions. The interested reader is guided to [4]. Topics studied in biostatistics are too numerous to list (for example multivariate regression, analysis of covariance, linear discriminant analysis, principal component analysis, and so forth; therefore, a sample that reflects the kind of tools that are used and most prevalent techniques will be given. Most of the biostatistics used is parameteric; the models result from human expertise. This is opposed to nonparametric models (or data-driven) that rely on the data itself. With the availability of cheap, fast computation, however, the use of nonparametric models has exploded. Given a set of data $\mathbf{x} = x_1, x_2,...,x_n$, the probability $P(\mathbf{x}|\theta_1,\theta_2,...,\theta_m)$ is parametric if m is not dependent on n; otherwise it is non-parametric. Linear Regression is a parametric model that presumes a linear relationship between two random variables (rv) both having Gaussian distributed noise. It is presumed the variables are real valued. If rv Y is a function of rv X, then the phrase, "A regression of Y on X," is used. One is determining the optimal coefficients β_0,β_1 given data $D = \{\langle x_1,y_1\rangle,...,\langle x_n,y_n\rangle\}$ on function $Y = \beta_1 X + \beta_0$. Regression, though simple, provides a first step in understanding the relationship between to rvs. Regression can be used to predict, adjust, explain, *etc.* and is common in biostatistics. Regression produces an optimal linear relationship between rvs Y and X. Typically, one minimizes the squares of the residuals–the difference between the hypothetical point and observed point. The use is so ubiquitous that virtually every mathematical or statistical package has this available. As pointed out many times in the area, because two rvs are *not* linearly related, does not mean they are *not* functionally related.

Correlation, related to Linear Regression, is as often used. It is so similar in so many ways to Linear Regression that the literature will often have the use of one, when in fact, it is the other that is warranted or even makes sense. Succinctly, Regression examines how one rv depends on another; Correlation examines how two rvs behave together–in concert or more formally *covary*. Correlation is related to moments and is often called product-moments. One of the most popular is the Pearson product-moment $\rho = \frac{\sigma_{XY}}{\sigma_X\sigma_Y}$ where σ_i is the variance of the joint (numerator) and individual rvs (denominator). The value $\rho \in [-1,1]$ where as $|\rho| \approx 1$, the variables appear to have an association. As ρ approaches 0, this indicates they do not. The positive and negative score reflects the direction of the association. Another popular correlation examines how pairs of pairs of values behave called ranks. Kendall's τ examines two pairs of pairs $\langle x_i, y_y\rangle,\langle x_j, y_j\rangle$ observing if *both* values of one pair are greater or smaller than the other pair; if so, then the pair of pairs is called concordant. If this is not satisfied, then the pair of pairs is discordant. The formula for Kendall's $\tau = \frac{N}{n(n-1)}$ where N is the count of ranks and n the sample size. It should be pointed out this is a nonparametric statistic and value $\tau \in (-1,1)$.

A good deal of analysis is done with simple hypothesis testing of sample statistics. Because an entire population can be seldom checked, to establish a degree of certainty about a property we use sampling techniques (statistics). Sampling is done randomly to hopefully reflect the underlying distribution, especially if it is not known. If the sample is consistent with the conjecture about the existence of a property (called a hypothesis), then the hypothesis is said to be *accepted*; if not, it is *rejected*. There exists two disparate hypotheses that are simultaneously proposed when doing this analysis. The Null hypothesis, typically denoted H_0, is that the property was observed likely through chance. The Alternative hypothesis, typically denoted by H_a, is that the property is *not* due to chance. While recapitulating this process in its entirety is not appropriate here, we summarize the following sequence of standard steps that are typically followed for such analysis: **(0)** State the two hypotheses H_0 and H_a. It must be the case that only one can be true; **(1)** Establish the protocol for acquiring and using sample data. The decision usually depends on a chosen test statistic (a real-valued function of the sample); **(3)** Calculate the test statistic's value; **(4)** Check whether the statistic's value is likely to have occurred by chance or not typically using tables [2].

There are two types of errors that can result in this procedure. A Type I error will reject a null hypothesis when it is actually true. The significance level is the probability of committing this error and is often denoted by α. A Type II error is accepting a null hypothesis when it is actually false. The power of the test is the probability of not committing a Type II error. A P-value is, in a sense, an extreme case in examining H_0. It will reflect the strength of the evidence. P-values are very useful, since one can apply any number of significance levels reflecting confidence in the data.

ANOVA (analysis of variance), developed by Fisher almost a century ago remains a popular tool that examines differences in populations means $\mu = \Sigma_{i=1}^{N} \frac{X_i}{N}$, for values $X_1, X_2, \ldots, X_N$ one of the measures of central tendency of a population (*e.g.*, median, mode). Population variance, $\sigma^2 = \Sigma_{i=1}^{N} \frac{(X_i - \mu)^2}{N}$, measures the spread of values. There are two models of ANOVA, Model I and Model II, which form H_0 and H_a based on mean and variance, respectively. The interested reader is guided to [5] for an in-depth presentation.

Key Applications

The initial application of biostatistics was to study evolution and natural selection. It now plays an essential role in sequence and genome analysis, protein structure prediction, proteomics, phylogenetics, *etc.* A current challenge for data analysis is to consider extensions to relational data, to include probabilistic data and uncertainty, as they their roles in biological inquiry. Database researchers, one of the challenges is to move in thinking from a Boolean model (relational) to one that involves probability and uncertainty, conflicting data, non-replicable data, and data that has orders of magnitude more attributes than tuples. Data in biology is often un-normalized and may lack primary key identifiers normalization.

Future Directions

Biologists and computer scientists use different paradigms when considering data and analysis. Biologists are reductionists and focus on tightly constrained problems, whereas computer scientists pursue generic technological solutions that can be applied to multiple problems. Researchers in data management and biostatistics must keep these differences in mind as they move towards developing useful yet generic tools to serve biologists into the future. The future of biostatistics will depend directly on the computational resources available. Typically, two different approaches to problems are taken: combinatorial (or enumerative) or statistical. The former is usually very fast, but lacks the ability to discover nuanced relationships. Statistical approaches are computationally expensive, *e.g.*, Expectation-Maximization, but with increasingly powerful computers and clusters, this computational impediment is slowly eroding.

Cross-reference

▶ Annotation
▶ Biomedical Data/Content Acquisition, Curation
▶ Clustering
▶ Curation
▶ Data Quality Assessment
▶ F-measure
▶ Metadata-based Query Processing for Statistical Data
▶ Principal Component Analysis
▶ Probabilistic Databases
▶ Spectral Clustering
▶ Taxonomy: Biomedical Health Informatics
▶ Term Weighting
▶ Text Analytics
▶ Text Compression
▶ Two-Poisson Model
▶ Uncertainty in Events

Recommended Reading

1. Lee P.M. Bayesian Statistics, 2nd Ed. Arnold, 2003.
2. Lindley D.V. and Scott W.F. New Cambridge Statistics Tables 2nd Ed. Cambridge University Press, 1995.

3. Myers C.S. Vitalism: A Brief Historical and Critical Review. Mind, 9(35):319–331, 1900.
4. Neapolitan R.E. Learning Bayesian Networks. Prentice Hall, 2003.
5. Sokal R. and Rohlf F.J. *Biometry.* W.H. Freeman and Company, NY, 3rd edition, 1995.
6. von Mises R. Probability, Statistics, and Truth. 1939. Translated by J. Neyman and D. Scholl and E. Rabinowitsch.

BIR Model

▶ Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model

Bit Vector Join

▶ Semijoin

Bi-temporal Access Methods

▶ Bi-Temporal Indexing

Bitemporal Algebras

▶ Temporal Algebras

Bitemporal Data Model

▶ Temporal Data Models

Bi-Temporal Indexing

Mirella M. Moro[1], Vassilis J. Tsotras[2]
[1]The Federal University of Rio Grande dosul, Porto Alegre, Brazil
[2]University of California-Riverside, Riverside, CA, USA

Synonyms

Bi-temporal access methods

Definition

A bi-temporal index is a data structure that supports both temporal time dimensions, namely, transaction-time (the time when a fact is stored in the database) and valid-time (the time when a fact becomes valid in reality). The characteristics of the time dimensions supported imply various properties that the bi-temporal index should have to be efficient. As traditional indices, the performance of a temporal index is described by three costs: (i) storage cost (i.e., the number of pages the index occupies on the disk), (ii) update cost (the number of pages accessed to perform an update on the index; for example when adding, deleting or updating a record), and (iii) query cost (the number of pages accessed for the index to answer a query).

Historical Background

Most of the early work on temporal indexing has concentrated on providing solutions for transaction-time databases. A basic property of transaction-time is that it always *increases*. Each newly recorded piece of data are time-stamped with a new, larger, transaction time. The immediate implication of this property is that previous transaction times *cannot* be changed. Hence, a transaction-time database can “rollback” to, or answer queries for, any of its previous states.

On the other hand, a valid-time database maintains the entire temporal behavior of an enterprise as best known now. It stores the current knowledge about the enterprise’s past, current or even future behavior. If errors are discovered about this temporal behavior, they are corrected by modifying the database. In general, if the knowledge about the enterprise is updated, the new knowledge modifies the existing one. When a correction or an update is applied, previous values are not retained. It is thus not possible to view the database as it was before the correction/update.

By supporting both valid and transaction time, a bi-temporal database combines the features of the other temporal database types. While it keeps its past states, it also supports changes anywhere in the valid time domain. Hence, the overlapping and persistent methodologies proposed for transaction-time indexing can be applied [5,6,9]. The difference with transaction-time indexing is that the underlying access method should be able to dynamically manage intervals (like an R-tree, a quad-tree etc.). For a worst-case comparison of temporal access methods, the reader is referred to [7].

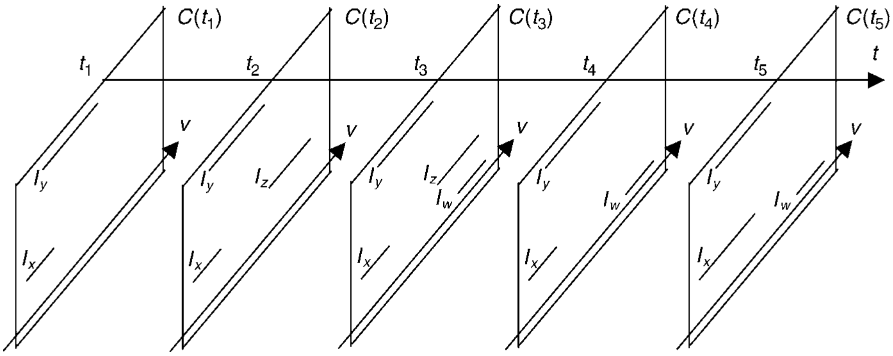
Foundations

When considering temporal indexing, it is important to realize that the valid and transaction time dimensions are *orthogonal* [3]. While in various scenarios it may be assumed that data about a fact is entered in the database at the same time as when it happens in the real world (i.e., valid and transaction time coincide), in practice, there are many applications where this assumption does not hold. For example, data records about the sales that occurred during a given day are recorded in the database at the end of the day (when batch processing of all data collected during the day is performed). Moreover, a recorded valid time may represent a later time instant than the transaction time when it was recorded. For example, a contract may be valid for an interval that is later than the (transaction) time when this information was entered in the database. The above properties are critical in the design of a bi-temporal access method since the support of both valid and transaction time affects directly the way records are created or updated. Note that the term "interval" is used here to mean a "convex subset of the time domain" (and not a "directed duration"). This concept has also been named a "period"; in this discussion however, only the term "interval" is used.

The reader is referred to the entry on Transaction-Time Indexing, in which a transaction time database was abstracted as an evolving collection of objects; updates arrive in increasing transaction-time order and are always applied on the latest state of this set. In other words, previous states cannot be changed. Thus a transaction-time database represents and stores the database activity; objects are associated with intervals based on this database activity. In contrast, in the chapter on Valid-Time Indexing, a valid-time database was abstracted as an evolving collection of interval-objects, where each interval represents the validity interval of an object. The allowable changes in this environment are the addition/deletion/modification of an interval-object. A difference with the transaction-time abstraction is that the collection's evolution (past states) is *not* kept. Note that when considering the valid time dimension, changes do not necessarily come in increasing time order; rather they can affect *any* interval in the collection. This implies that a valid-time database can correct errors in previously recorded data. However, only a single data state is kept, the one resulting after the correction is applied.

A bi-temporal database has the characteristics of both approaches. Its abstraction maintains the evolution (through the support of transaction-time) of a dynamic collection of (valid-time) interval-objects. Figure 1 offers a conceptual view of a bi-temporal database. Instead of maintaining a single collection of interval-objects (as a valid-time database does) a bi-temporal database maintains a sequence of such collections $C(t_i)$ indexed by transaction-time. Assume that each interval I represents the validity interval of a contract in a company. In this environment, the user can represent how the knowledge about company contracts evolved. In Fig. 1, the t-axis (v-axis) corresponds to transaction (valid) times. At transaction time t_1, the database starts with interval-objects I_x and I_y. At t_2, a new interval-object I_z is recorded, etc. At t_5 the valid-time interval of object I_x is modified to a new length.

When an interval-object I_j is inserted in the database at transaction-time t, a record is created with the object's surrogate (contract_no I_j), a valid-time interval (contract duration), and an initial transaction-time interval $[t, UC]$. When an object is inserted, it is not yet known if it (ever) will be updated. Therefore, the right endpoint of the transaction-time interval is filled with



Bi-Temporal Indexing. Figure 1. A bi-temporal database.

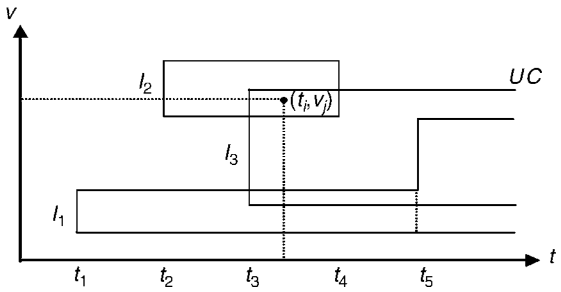
the variable *UC* (Until Changed), which will be changed to another transaction time if this object is later updated. For example, the record for interval-object I_z has transaction-time interval $[t_2, t_4]$, because it was inserted in the database at transaction-time t_2 and was "deleted" at t_4. Note that the collections $C(t_3)$ and $C(t_4)$ correspond to the collections C_a and C_b of Fig. 1 in the Valid-Time Indexing chapter, assuming that at transaction-time t_4 the erroneous contract I_z was deleted from the database.

Based on the above discussion, an index for a bi-temporal database should: (i) store past states, (ii) support addition/deletion/modification changes on the interval-objects of its current logical state, and (iii) efficiently access and query the interval-objects on any state.

Figure 1 summarizes the differences among the various database types. Each collection $C(t_i)$ can be thought of on its own, as a separate valid-time database. A valid-time database differs from a bi-temporal database since it keeps *only one* collection of interval-objects (the latest). A transaction-time database differs from a bi-temporal database in that it maintains the history of an evolving set of *plain*-objects instead of *interval*-objects. A transaction-time database differs from a conventional (non-temporal) database in that it also keeps its *past* states instead of only the latest state. Finally, the difference between a valid-time and a conventional database is that the former keeps *interval*-objects (and these intervals can be queried).

There are three approaches that can be used for indexing bi-temporal databases.

Approach 1: The first one is to have each bi-temporal object represented by a "bounding rectangle" created by the object's valid and transaction-time intervals, and to store it in a conventional multi-dimensional structure like the R-tree. While this approach has the advantage of using a single index to support both time dimensions, the characteristics of transaction-time create a serious overlapping problem [5]. A bi-temporal object with valid-time interval *I* that is inserted in the database at transaction time *t*, is represented by a rectangle with a transaction-time interval of the form $[t, UC]$. All bi-temporal objects that have not been deleted (in the transaction sense) will share the common transaction-time endpoint *UC* (which in a typical implementation, could be represented by the largest possible transaction time). Furthermore, intervals that remain unchanged will create long (in the transaction-time axis) rectangles, a reason for further overlapping. A simple bi-temporal query that asks for all valid time intervals that at transaction time t_i contained valid time v_j, corresponds to finding all rectangles that contain point (t_i, v_j).

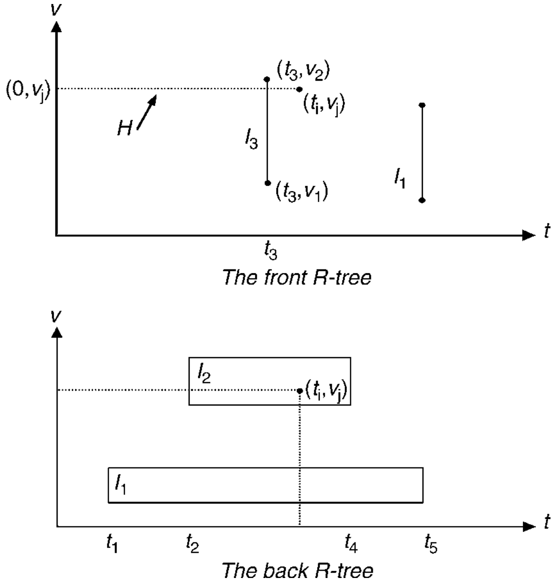


Bi-Temporal Indexing. Figure 2. The bounding-rectangle approach for bi-temporal objects.

Figure 2 illustrates the bounding-rectangle approach; only the valid and transaction axis are shown. At t_5, the valid-time interval I_1 is modified (enlarged). As a result, the initial rectangle for I_1 ends at t_5, and a new enlarged rectangle is inserted ranging from t_5 to *UC*.

Approach 2: To avoid overlapping, the use of two R-trees has also been proposed [5]. When a bi-temporal object with valid-time interval *I* is added in the database at transaction-time *t*, it is inserted at the *front* R-tree. This tree keeps bi-temporal objects whose right transaction endpoint is unknown. If a bi-temporal object is later deleted at some time $t' > t$, it is physically deleted from the front R-tree and inserted as a rectangle of height *I* and width from *t* to *t'* in the *back* R-tree. The back R-tree keeps bi-temporal objects with known transaction-time interval. At any given time, all bi-temporal objects stored in the front R-tree share the property that they are alive in the transaction-time sense. The temporal information of every such object is thus represented simply by a vertical (valid-time) interval that "cuts" the transaction axis at the transaction-time when this object was inserted in the database. Insertions in the front R-tree objects are in increasing transaction time while physical deletions can happen anywhere on the transaction axis.

In Fig. 3, the two R-trees methodology for bi-temporal data are divided according to whether their right transaction endpoint is known. The scenario of Fig. 2 is presented here (i.e., after time t_5 has elapsed). The query is then translated into an interval intersection and a point enclosure problem. A simple



Bi-Temporal Indexing. Figure 3. The two R-tree methodology for bi-temporal data.

bi-temporal query that asks for all valid time intervals which contained valid time v_j at transaction time t_i, is answered with two searches. The back R-tree is searched for all rectangles that contain point (t_i, v_j). The front R-tree is searched for all vertical intervals that intersect a horizontal interval H that starts from the beginning of transaction time and extends until point t_i at height v_j.

When an R-tree is used to index bi-temporal data, overlapping may also incur if the valid-time intervals extend to the ever-increasing *now*. One approach could be to use the largest possible valid-time timestamp to represent the variable *now*. In [2] the problem of addressing both the *now* and *UC* variables is addressed by using bounding rectangles/regions that increase as the time proceeds. A variation of the R-tree, the GR-tree is presented. The index leaf nodes capture the exact geometry of the bi-temporal regions of data. Bi-temporal regions can be static or growing, rectangles or stair-shapes. Two versions of the GR-tree are explored, one using minimum bounding rectangles in non-leaf nodes, and one using minimum bounding regions in non-leaf nodes. Details appear in [2].

Approach 3: Another approach to address bi-temporal problems is to use the notion of partial persistence [1,4]. This solution emanates from the abstraction of a bi-temporal database as a sequence of collections $C(t)$ (in Fig. 1) and has two steps. First, a good index is chosen to represent each $C(t)$. This index must support dynamic addition/deletion of (valid-time) interval-objects. Second, this index is made partially persistent. The collection of queries supported by the interval index structure implies which queries are answered by the bi-temporal structure. Using this approach, the Bi-temporal R-tree that takes an R-tree and makes it partially persistent was introduced in [5].

Similar to the transaction-time databases, one can use the "overlapping" approach [3] to create an index for bi-temporal databases. It is necessary to use an index that can handle the valid-time intervals and an overlapping approach to provide the transaction-time support. Multi-dimensional indexes can be used for supporting intervals. For example, an R-tree or a quad-tree. The Overlapping-R-tree was proposed in [6], where an R-tree maintains the valid time intervals at each transaction time instant. As intervals are added/deleted or updated, overlapping is used to share common paths in the relevant R-trees. Likewise, [9] proposes the use of quad-trees (which can also be used for spatiotemporal queries).

There are two advantages in "viewing" a bi-temporal query as a "partial persistence" or "overlapping" problem. First, the valid-time requirements are disassociated from the transaction-time ones. More specifically, the valid time support is provided from the properties of the R-tree while the transaction time support is achieved by making this structure "partially persistent" or "overlapping." Conceptually, this methodology provides fast access to the $C(t)$ of interest on which the valid-time query is then performed. Second, changes are always applied to the most current state of the structure and last until updated (if ever) at a later transaction time, thus avoiding the explicit representation of variable *UC*. Considering the two approaches, overlapping has the advantage of simpler implementation, while the partial-persistence approach avoids the possible logarithmic space overhead.

Key Applications

The importance of temporal indexing emanates from the many applications that maintain temporal data. The ever increasing nature of time imposes the need for many applications to store large amounts of temporal data. Accessing such data specialized indexing techniques is necessary. Temporal indexing has offered many such solutions that enable fast access.

Cross-references

- ▶ B+-Tree
- ▶ Rtree
- ▶ Temporal Database
- ▶ Transaction-Time Indexing
- ▶ Valid-Time Indexing

Recommended Reading

1. Becker B., Gschwind S., Ohler T., Seeger B., and Widmayer P. An asymptotically optimal multiversion B-tree. VLDB J., 5 (4):264–275, 1996.
2. Bliujute R., Jensen C.S., Saltenis S., and Slivinskas G. R-tree based indexing of now-relative bitemporal data. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 345–356.
3. Burton F.W., Huntbach M.M., and Kollias J.G. Multiple generation text files using overlapping tree structures. Comput. J., 28(4):414–416, 1985.
4. Driscoll J.R., Sarnak N., Sleator D.D., and Tarjan R.E. Making data structures persistent. J. Comput. Syst. Sci., 38(1):86–124, 1989.
5. Kumar A., Tsotras V.J., and Faloutsos C. Designing access methods for bitemporal databases. IEEE Trans. Knowl. Data Eng., 10(1):1–20, 1998.
6. Nascimento M.A. and Silva J.R.O. Towards historical R-trees. In Proc. 1998 ACM Symp. on Applied Computing, 1998, pp. 235–240.
7. Salzberg B. and Tsotras V.J. A comparison of access methods for time-evolving data. ACM Comput. Surv., 31(2):158–221, 1999.
8. Snodgrass R.T. and Ahn I. Temporal databases. IEEE Comput., 19(9):35–42, 1986.
9. Tzouramanis T., Vassilakopoulos M., and Manolopoulos Y. Overlapping linear quadtrees and spatio-temporal query processing. Comput. J., 43(4):325–343, 2000.

Bitemporal Interval

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

Facts in a bitemporal database may be timestamped by time values that are products of time intervals drawn from two orthogonal time domains that model valid time and transaction time, respectively. A *bitemporal interval* then is given by an interval from the valid-time domain and an interval from the transaction-time domain, and denotes a rectangle in the two-dimensional space spanned by valid and transaction time.

When associated with a fact, a bitemporal interval then identifies an interval (valid time) during which that fact held (or holds or will hold) true in reality, as well as identifies an interval (transaction time) when that belief (that the fact was true during the specified valid-time interval) was held, i.e., was part of the current database state.

Key Points

In this definition, a time interval denotes a convex subset of the time domain. Assuming a discrete time domain, a bitemporal interval can be represented with a non-empty set of bitemporal chronons or granules.

Cross-references

- ▶ Bitemporal Relation
- ▶ Chronon
- ▶ Temporal Database
- ▶ Temporal Granularity
- ▶ Time
- ▶ Time Domain
- ▶ Time Interval
- ▶ Transaction Time

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399. Springer, Berlin Heidelberg New York, 1998, pp. 406–413.
2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts–February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399. Springer, Berlin Heidelberg New York, 1998, pp. 367–405.

Bitemporal Relation

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Temporal relation; Fully temporal relation; Valid-time and transaction-time relation

Definition

A *bitemporal relation* captures exactly one valid time aspect and one transaction time aspect of the data it contains. This relation inherits its properties from valid-time relations and transaction-time relations. There are no restrictions as to how either of these temporal aspects may be incorporated into the tuples.

Key Points

In this definition, "bi" refers to the capture of exactly two temporal aspects. An alternative definition states that a bitemporal relation captures one or more valid times and one or more transaction times. In this definition, "bi" refers to the existence of exactly two types of times.

One may adopt the view that the data in a relation represents a collection of logical statements, i.e., statements that can be assigned a truth values. The valid times of these so-called facts are the times when these are true in the reality modeled by the relation. In cases where multiple realities are perceived, a single fact may have multiple, different valid times. This might occur in a relation capturing archaeological facts for which there no agreements among the archaeologists. In effect, different archaeologists perceive different realities.

Transaction times capture when database objects are current in a database. In case an object migrates from one database to another, the object may carry along its transaction times from the predecessor databases, termed *temporal generalization*. This then calls for relations that capture multiple transaction times.

The definition of bitemporal is used as the basis for applying bitemporal as a modifier to other concepts such as "query language." A query language is bitemporal if and only if it supports any bitemporal relation. Hence, most query languages involving both valid and transaction time may be characterized as bitemporal.

Relations are named as opposed to databases because a database may contain several types of relations. Most relations involving both valid and transaction time are bitemporal according to both definitions.

Concerning synonyms, the term "temporal relation" is commonly used. However, it is also used in a generic and less strict sense, simply meaning any relation with time-referenced data.

Next, the term "fully temporal relation" was originally proposed because a bitemporal relation is capable of modeling both the intrinsic and the extrinsic time aspects of facts, thus providing the "full story." However, this term is no longer used.

The term "valid-time and transaction-time relation" is precise and consistent with the other terms, but is also lengthy.

Cross-references

- ▶ Bitemporal Interval
- ▶ Temporal Database
- ▶ Temporal Generalization
- ▶ Transaction Time
- ▶ Valid Time

Recommended Reading

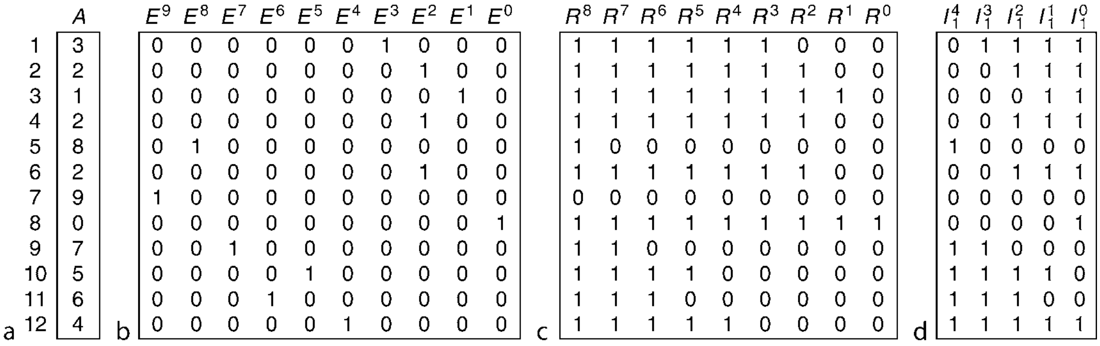
1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts–February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399. Springer, Berlin Heidelberg New York, 1998, pp. 367–405.

Bitmap Index

Chee Yong Chan
National University of Singapore, Singapore,
Singapore

Definition

An index on an attribute provides an efficient way to access data records associated with a given range of values for the indexed attribute. Typically, an index stores a list of RIDs (called a RID-list) of all the records associated with each distinct value v of the indexed attribute. In a *bitmap index*, each RID-list is represented in the form of a bit vector (i.e., bitmap) where the size of each bitmap is equal to the cardinality of the indexed relation, and the ith bit in each bitmap corresponds to the ith record in the indexed relation. The simplest bitmap index design is the *Value-List index*, which is illustrated in Fig. 1b for an attribute A of a 12-record relation R in Fig. 1a. In this bitmap index, there is one bitmap E^v associated with each attribute value $v \in [0,9]$ such that the ith bit of E^v is set to 1 if and only if the ith record has a value v for the indexed attribute.



	A	E^9	E^8	E^7	E^6	E^5	E^4	E^3	E^2	E^1	E^0	R^8	R^7	R^6	R^5	R^4	R^3	R^2	R^1	R^0	I_1^4	I_1^3	I_1^2	I_1^1	I_1^0
1	3	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1
2	2	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	0	0	0	0	1	1	1
3	1	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1
4	2	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	0	0	0	0	1	1	1
5	8	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
6	2	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	0	0	0	0	1	1	1
7	9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1
9	7	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	0
10	5	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	1	0
11	6	0	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0
12	4	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0	0	0	0	1	1	1	1	1

Bitmap Index. Figure 1. Examples of bitmap indexes. (a) indexed attribute *A* (b) equality-encoded index (or value-list index) (c) range-encoded index (or base-10 bit-sliced index) (d) interval-encoded index.

Historical Background

The idea of using bitmap indexes to speed up selection predicate evaluation has been recognized since the early 1970s [4]. Some early implementations of bitmap processing techniques include PC DBMSs (e.g., FoxPro, Interbase), a scientific/statistical database application developed at Lawrence Berkeley Laboratory [11], and Model 204, which is a commercial DBMS for the IBM mainframe [7].

The main advantage of using a bitmap index is the CPU efficiency of bitmap operations (AND, OR, XOR, NOT). Furthermore, compared to RID-based indexes, bitmap indexes are more space-efficient for attributes with low cardinality and more I/O-efficient for evaluating selection predicates with low selectivities. For example, assuming each RID requires four bytes of storage and ignoring any compression, bitmap indexes are more space-efficient if the attribute cardinality is less than 32, and reading a bitmap is more I/O-efficient than reading a RID-list if the selectivity factor of the selection predicate is more than $\frac{1}{32}(\approx 3.2\%)$. Another advantage of bitmap indexes is that they are very amenable to parallelization due to the equal-sized bitmaps and the nature of the bitwise operations.

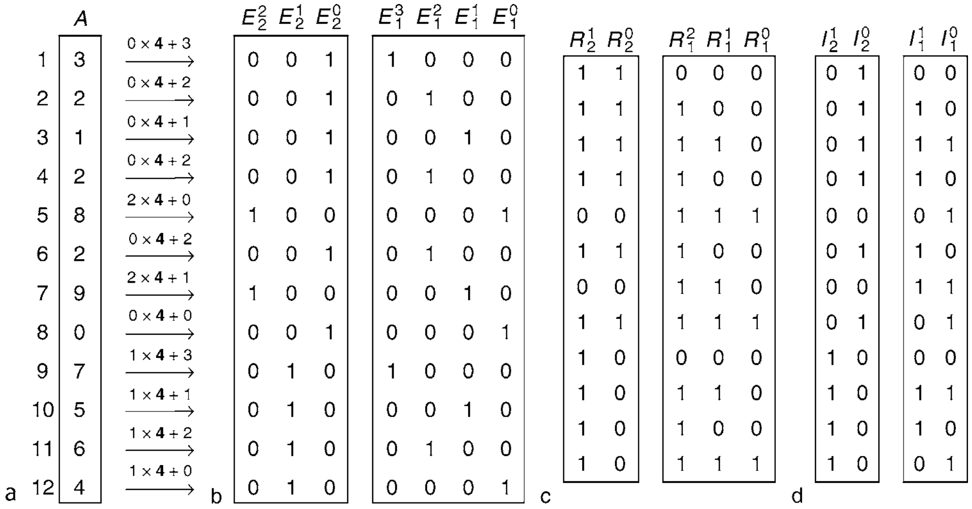
A variety of bitmap index designs have been proposed since the early days. Besides the simple Value-List index illustrated in Fig. 1b, another early bitmap index design is the *Bit-Sliced index (BSI)* which is implemented in Model 204 and Sybase IQ [7]. A BSI for an attribute with a cardinality of C consists of $k = \lceil log_2(C) \rceil$ bitmaps, with one bitmap associated with each bit in the binary representation of C. Compared to the Value-List index, the BSI is more space-efficient with an attribute value v being encoded by a string of k bits corresponding to its binary representation. The BSI design can be generalized to use a non-binary base b such that it consists of $k(b - 1)$ bitmaps $\{B_i^j : 1 \leq i \leq k, 0 \leq j < b\}$, where $k = \lceil log_b(C) \rceil$. Each attribute value v is expressed in base b as a sequence of k base-b digits $v_k v_{k-1} ... v_2 v_1$, and each bitmap B_i^j represents the set of records with $v_i \leq j$. Using a larger base number improves the index's performance for evaluating range predicates at the cost of an increased space cost. An example of a base-10 BSI is shown in Fig. 1c. Both Model 204 and Sybase IQ implemented base-10 Bit-Sliced indexes [7].

Several bitmap index designs have also been implemented in a scientific/statistical database application at Lawrence Berkeley Laboratory [11]. These bitmap indexes include binary encoded indexes (equivalent to binary BSI), unary encoded indexes (equivalent to non-binary BSI), K-of-N encoded indexes (generalizations of Value-List indexes where each attribute value is encoded by a N-bit string with exactly K bits set to 1), and superimposed encoded indexes based on superimposed encoding which is useful for indexing set-valued attributes.

Interest in bitmap indexes was revived in the mid 1990s due to the emergence of data warehousing applications which are characterized by read-mostly query workloads dominated by large, complex ad hoc queries [7]. All the major DBMS vendors (IBM, Microsoft, Oracle, and Sybase) also started to support bitmap indexes in their products around this time.

Foundations

Chan and Ioannidis propose a two-dimensional framework to characterize the design space of bitmap indexes [2]. The two orthogonal parameters identified for bitmap indexes (with an attribute cardinality of C) are (i) the arithmetic used to represent attribute values; i.e., how an attribute value is decomposed into digits according to some base (e.g., base-C arithmetic is used in a Value-List index); and (ii) the encoding scheme of each decomposed digit in bits (e.g., each attribute value in a Value-List index is encoded by turning on exactly one out of C bits). Consider an attribute value $v \in [0,C)$ and a sequence of n base numbers $\mathcal{B} = <b_n, b_{n-1},\ldots,b_1>$, where $b_n = \left\lceil \frac{C}{\prod_{i=1}^{n-1} b_i} \right\rceil$ and $b_i \geq 2$, $i \in [1,n]$. Using $\mathcal{B}$, v can be decomposed into a unique sequence of n digits $<v_n,v_{n-1},\ldots,v_1>$ as follows: $v_i = V_i \; mod \; b_i$, where $V_1 = v$ and $V_i = \left\lfloor \frac{V_{i-1}}{b_{i-1}} \right\rfloor$, for $1 < i \leq n$. Thus, $v = v_n\left(\prod_{j=1}^{n-1} b_j\right) + \ldots + v_i\left(\prod_{j=1}^{i-1} b_j\right) + \ldots + v_2 b_1 + v_1$. Note that each v_i is a base-b_i digit (i.e., $0 \leq v_i < b_i$). Each choice of n and base-sequence $\mathcal{B}$ gives a different representation of attribute values, and therefore a different index (known as a Base-$\mathcal{B}$ index). The index consists of n *components* (i.e., one component per digit) where each component individually is now a collection of bitmaps. Figure 2b shows a base-<3,4> Value-List index that consists of two components: the first component has four bitmaps $\{E_1^3, E_1^2, E_1^1, E_1^0\}$, and the second component has three bitmaps $\{E_2^2, E_2^1, E_2^0\}$. Note that the kth bit in each bitmap E_i^j is set to 1 if and only if $v_i = j$, where $<v_2,v_1>$ is the <3,4>-decomposition of the kth record's indexed attribute value. For the encoding scheme dimension, there are two basic encoding schemes: equality encoding and range encoding. Consider the ith component of an index with base b_i, and a value $v_i \in [0,b_i - 1]$. In the *equality encoding scheme*, v_i is encoded by b_i bits, where all the bits are set to 0 except for the bit corresponding to v_i, which is set to 1. Thus, an equality-encoded component (with base b_i) consists of b_i bitmaps $\{E_i^{b_i-1},\ldots,E_i^0\}$ such that the kth bit in each bitmap E_i^j is set to 1 if and only if $v_i = j$, where v_i is the ith digit of the decomposition of the kth record's indexed attribute value. In the *range encoding scheme*, v_i is encoded again by b_i bits, with the v_i rightmost bits set to 0 and the remaining bits (starting from the one corresponding to v_i and to the left) set to 1. The kth bit in each bitmap R_i^j is set to 1 if and only if $v_i \leq j$, where v_i is the ith digit of the decomposition of the kth record's indexed attribute value. Since the bitmap $R_i^{b_i-1}$ has all bits set to 1, it does not need to be stored, so a range-encoded component consists of $(b_i - 1)$ bitmaps $\{R_i^{b_i-2},\ldots,R_i^0\}$. Value-List and Bit-Sliced indexes therefore correspond to equality-encoded and range-encoded indexes, respectively. Figures 1c and 2c show the range-encoded indexes corresponding to the equality-encoded indexes in Figs. 1b and 2b. Details of



	A		E_2^2	E_2^1	E_2^0	E_1^3	E_1^2	E_1^1	E_1^0	R_2^1	R_2^0	R_1^2	R_1^1	R_1^0	I_2^1	I_2^0	I_1^1	I_1^0
1	3	$\xrightarrow{0 \times 4 + 3}$	0	0	1	1	0	0	0	1	1	0	0	0	0	1	0	0
2	2	$\xrightarrow{0 \times 4 + 2}$	0	0	1	0	1	0	0	1	1	1	0	0	0	1	1	0
3	1	$\xrightarrow{0 \times 4 + 1}$	0	0	1	0	0	1	0	1	1	1	1	0	0	1	1	1
4	2	$\xrightarrow{0 \times 4 + 2}$	0	0	1	0	1	0	0	1	1	1	0	0	0	1	1	0
5	8	$\xrightarrow{2 \times 4 + 0}$	1	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1
6	2	$\xrightarrow{0 \times 4 + 2}$	0	0	1	0	1	0	0	1	1	1	0	0	0	1	1	0
7	9	$\xrightarrow{2 \times 4 + 1}$	1	0	0	0	0	1	0	0	0	1	1	0	0	0	1	1
8	0	$\xrightarrow{0 \times 4 + 0}$	0	0	1	0	0	0	1	1	1	1	1	1	0	1	0	1
9	7	$\xrightarrow{1 \times 4 + 3}$	0	1	0	1	0	0	0	1	0	0	0	0	1	0	0	0
10	5	$\xrightarrow{1 \times 4 + 1}$	0	1	0	0	0	1	0	1	0	1	1	0	1	0	1	1
11	6	$\xrightarrow{1 \times 4 + 2}$	0	1	0	0	1	0	0	1	0	1	0	0	1	0	1	0
12	4	$\xrightarrow{1 \times 4 + 0}$	0	1	0	0	0	0	1	1	0	1	1	1	1	0	0	1

Bitmap Index. Figure 2. Example of base- <3,4> indexes. (a) indexed attribute A (b) equality-encoded index (c) range-encoded index (d) interval-encoded index.

query processing algorithms and space-time tradeoffs of equality/range-encoded, multi-component bitmap indexes are given in [4].

A new encoding scheme, called *interval encoding*, was proposed in [5]. For an attribute with a cardinality of C, a value $v \in [0,C)$ is encoded using $\lceil \frac{C}{2} \rceil$ bits such that if $v < \lceil \frac{C}{2} \rceil$, then v is encoded by setting the $(v+1)$ rightmost bits to 1 and the remaining bits to 0; otherwise, v is encoded by setting the $(C-1-v)$ leftmost bits to 1 and remaining bits to 0. Thus, the interval encoding scheme consists of $\lceil \frac{C}{2} \rceil$ bitmaps $\{I^{\lceil \frac{C}{2} \rceil - 1},\ldots,I^0\}$, where each bitmap I^j is associated with a range of $(m+1)$ values $[j,j+m]$, $m = \lfloor \frac{C}{2} \rfloor - 1$, such that the kth bit in a bitmap I^j is set to 1 if and only if the kth record's indexed attribute value is in $[j,j+m]$. Figures 1d and 2d show the interval-encoded indexes corresponding to the equality-encoded indexes in Figs. 1b and 2b. Note that interval encoding has better space-time tradeoff than range encoding: it has the same worst-case evaluation cost of two bitmap scans as range encoding but its space requirement is about half that of range encoding.

Wu and Bachmann proposed a variant of binary BSI called *encoded bitmap index (EBI)* [12,13]. Instead of encoding each attribute value simply in terms of its binary representation, an EBI uses a lookup table to map each attribute value to a distinct bit string; this flexibility enables optimization of the value-to-bit-string mapping, by exploiting knowledge of the query workload, to reduce the number of bitmap scans for query evaluation. Thus, binary BSI is a special case of EBI. Another similar index design called *Encoded-Vector index (EVI)* is used in IBM DB2. Instead of storing the index as a collection of $\lceil log_b(C) \rceil$ bitmaps as in EBI, EVI is organized as a single vector of $\lceil log_b(C) \rceil$-bit strings, and the purpose of the lookup table optimization is to reduce the CPU cost of bit string comparisons when evaluating selection queries of the form "$A \in \{v_1,v_2,\ldots,v_n\}$." Another related index is the *Projection index* [7], which is implemented in Sybase IQ.

Complex, multi-table join queries (such as star-join queries) can also be evaluated very efficiently using *bitmapped join indexes* [6], which are indexes that combine the advantages of *join indexes* and bitmap representation. A *join index* for the join between two relations R and S is a precomputation of their join result defined by $\Pi_{R.rid,S.rid}(R \bowtie_p S)$, where p is the join predicate between R and S. Thus, a join index on $R \bowtie S$ can be thought of as a conventional index on the table R, where the attribute being indexed is the "virtual" attribute $S.rid$; i.e., each distinct $S.rid$ value v is associated with a list of all $R.rid$ values that are related to v via the join. A *bitmapped join index* [6] is simply a join index with the RID-lists represented using bitmaps. Bitmapped join indexes are implemented in Informix Red Brick Warehouse and Oracle. Bitmap indexes can also be applied to evaluate queries that involve aggregate functions (e.g., SUM, MIN/MAX, MEDIAN); evaluation algorithms for Value-List and Bit-Sliced indexes are discussed in a paper by O'Neil and Quass [7]. Efficient algorithms for performing arithmetic operations (addition and subtraction) on binary BSIs are proposed in [8].

As bitmap indexes become less efficient for larger attribute cardinality, a number of approaches have been developed to reduce their space requirement. Besides using multi-component bitmap indexes [2,11], another common space-reduction technique is to apply compression. In Model 204 [7], the bitmaps are compressed by using a hybrid representation; specifically, each individual bitmap is partitioned into a number of fixed-size segments, and segments that are dense are stored as verbatim bitmaps while sparse segments are converted into RID-lists. While generic compression techniques (e.g., LZ77) are effective in reducing both the disk storage and retrieval cost of bitmap indexes, the savings in I/O cost can be offset by the high CPU cost incurred for decompressing the compressed bitmaps before they can be operated with other bitmaps. A number of specialized compression techniques that enable bitmaps to be operated on without a complete decompression have been proposed: Byte-aligned Bitmap Code (BBC) (which is used by Oracle), and Word-Aligned Hybrid code (WAH) [14]. Some performance study of compressed bitmap indexes are reported in [1–3,14].

A different approach proposed to reduce the size of bitmaps is to use *range-based bitmaps (RBB)* [15], which have been applied to index large data sets in tertiary storage systems as well as large, multidimensional data sets in scientific applications [14]. Unlike Value-List indexes where there is one bitmap for each distinct attribute value, the RBB approach partitions the attribute domain into a number of disjoint ranges and constructs one bitmap for each range of values (this is also known as *binning*). Thus, RBB provides a form of lossy compression which requires additional post-processing to filter out false positives.

Koudas [5] has examined space-optimal RBBs for equality queries when both the attribute and query distributions are known; these results have been extended for range queries [9]. More recently, Sinha and Winslett have proposed *multi-resolution bitmap indexes* to avoid the cost of filtering out false positives for RBB [10]. For example, in a two-resolution bitmap index, it has a lower resolution index consisting of RBB (i.e., with each bitmap representing a range of attribute values) and a higher resolution index consisting of one bitmap for each distinct attribute value. By combining the efficiency of lower resolution indexes and the precision of higher resolution indexes, queries can be evaluated efficiently without false positive filtering.

Key Applications

Today, bitmap indexes are supported by all major database systems, and they are particularly suitable for data warehousing applications [7]. Bitmap indexes have also been used in scientific databases (e.g., [10,11,14]), indexing data on tertiary storage systems (e.g., [1]), and data mining applications.

Future Directions

The design space for bitmap indexes is characterized by four key parameters: levels of resolution (which affects the number of levels of bitmap indexes and the index granularity at each level), attribute value representation (which affects the number and size of the index components at each level) encoding scheme (which affects how the bitmaps in each component are encoded), and storage format (i.e., uncompressed, compressed, or a combination of compressed and uncompressed). While there are several performance studies that have examined various combinations of the above parameter space, a comprehensive investigation into the space-time tradeoffs of the entire design space is, however, still lacking and deserves to be further explored. The result of such a study can be applied to further enhance automated physical database tuning tools.

Cross-references

- ► Access Path
- ► Bitmap-based Index Structures
- ► Data Warehouse

Recommended Reading

1. Amer-Yahia S. and Johnson T. Optimizing queries on compressed bitmaps. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 329–338.
2. Chan C.Y. and Ioannidis Y.E. Bitmap index design and evaluation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 355–366.
3. Chan C.Y. and Ioannidis Y.E. An efficient bitmap encoding scheme for selection queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 215–226.
4. Knuth D.E. Retrieval on secondary keys. In The Art of Computer Programming: Sorting and Searching, vol. 3. Chap. 6. Addison-Wesley, Reading, Mass., 1973, pp. 550–567.
5. Koudas N. Space efficient bitmap indexing. In Proc. Int. Conf. on Information and Knowledge Management, 2000, pp. 194–201.
6. O'Neil P. and Graefe G. Multi-table joins through bitmapped join indices. ACM SIGMOD Record. September 1995, pp. 8–11.
7. O'Neil P. and Quass D. Improved query performance with variant indexes. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 38–49.
8. Reinfret D., O'Neil P., and O'Neil E. Bit-sliced index arithmetic. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 2001, pp. 47–57.
9. Rotem D., Stockinger K., and Wu K. Optimizing candidate check cost for bitmap indices. In Proc. Int. Conf. on Information and Knowledge Management, 2005, pp. 648–655.
10. Sinha R. and Winslett M. Multi-resolution bitmap indexes for scientific data. ACM Trans. Database Syst., 32(3):1–39, 2007.
11. Wong H.K.T., Liu H-F., Olken F., Rotem D., and Wong L. Bit transposed files. In Proc. 11th Int. Conf. on Very Large Data Bases, 1985, pp. 448–457.
12. Wu M.C. Query optimization for selections using nitmaps. In Proc. ACM SIGMOD Int. Conf. on Management of Data 1999, pp. 227–238, 1999.
13. Wu M.C. and Buchmann A.P. Encoded bitmap indexing for data warehouses. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 220–230.
14. Wu K., Otoo E.J., and Shoshani A. Optimizing bitmap indices with efficient compression. ACM Trans. Database Syst., 31(1):1–38, 2006.
15. Wu K.L. and Yu P.S. Range-based bitmap indexing for high cardinality attributes with skew. Technical report, IBM Watson Research Center, May 1996.

Bitmap-based Index Structures

Guadalupe Canahuate, Hakan Ferhatosmanoglu
The Ohio State University, Columbus, OH, USA

Synonyms

Bitmap Index; Projection Index

Definition

A bitmap-based index is a binary vector that represents an interesting property and indicates which objects in the dataset satisfy the given property. The vector has a 1 in position *i* if the *i*-th data object satisfies the property, and 0 otherwise. Queries are executed using fast bitwise logical operations supported by hardware over the binary vectors.t

Historical Background

Bitmap-based indexing was first implemented in Computer Corporation of America's Model 204 in the mid-1980s by Dr. Patrick O'Neil. The bitmap index from Model 204 was a hybrid between verbatim (uncompressed) bitmaps and RID lists. Originally, a bitmap was created for each value in the attribute domain. The entire bitmap index is smaller than the original data as long as the number of distinct values is less than the number of bits used to represent the attribute in the original data. For example, if an integer attribute has cardinality 10 and integers are stored using 32 bits, then the bitmap index for such an attribute, which only has 10 bit vectors, is 3.2 times smaller than the original data. For floating point attributes, the bitsliced index (BSI) [8] stores each bit of the binary representation of the attribute independently. Bitsliced indexes are never more than the size of the original data. However, in general, all bitslices need to be accessed to answer a query.

In order to reduce the number of bit vectors that needed to be read to answer a query, more complex encodings for bitmap-based indexes have been proposed, such as range encoded bitmaps [15,8], encoded bitmaps [16], and interval encoded bitmaps [4,5]. For attributes with high cardinality including floating point attributes, binning [7,13] was proposed to reduce the number of values in the domain, and therefore the number of bit vectors in the index. In addition, special compression techniques were developed to improve the performance of the bitmap indexes. The two most popular techniques are Byte-aligned Bitmap Code (BBC) [1], and Word Aligned Hybrid (WAH) [18] compression method. These compression techniques allow query execution over the compressed bitmaps. With compression, bitmap indexes have been proven to perform well with high cardinality attributes [17].

Foundations

Bitmap tables are a special type of binary matrices. Each binary row in the bitmap table represents one tuple in the database. The bitmap columns are produced by quantizing the attributes in the database into categories or bins. Each tuple in the database is then encoded based on which bin each attribute value falls into. Bitmap index encoding is based on the properties of physical row identifiers and there is a one-to-one correspondence between the data objects and the bits in the bitmap vector. Therefore, given a bit position, the location of its corresponding table row can be computed by simple arithmetic operations. For tables with a clustered-index or index-ordered tables, a mapping table is used to map bit positions to row locations.

Bitmap Encoding

For equality encoded bitmaps (also called simple encoding or projection index) [8], if a value falls into a bin, this bin is marked "1", otherwise "0". Since a value can only fall into a single bin, only a *single* "1" can exist for each row of each attribute. After binning, the whole database is converted into a large 0–1 bitmap, where rows correspond to tuples and columns correspond to bins. Figure 1 shows an example using a table with one attribute with cardinality 4. Columns

Tuple	Value	Equality				Eq w/binning		Range			Interval			Bitsliced		
		= 1	= 2	= 3	= 4	{1,2}	{3,4}	≤3	≤2	≤1	[1,2]	[2,3]	[3,4]	2^2	2^1	2^0
t_1	1	1	0	0	0	1	0	1	1	1	1	0	0	0	0	1
t_2	2	0	1	0	0	1	0	1	1	0	1	1	0	0	1	0
t_3	3	0	0	1	0	0	1	1	0	0	0	1	1	0	1	1
t_4	1	1	0	0	0	1	0	1	1	1	1	0	0	0	0	1
t_5	2	0	1	0	0	1	0	0	1	1	1	1	0	0	1	0
t_6	4	0	0	0	1	0	1	0	0	0	0	0	1	1	0	0
t_7	1	1	0	0	0	1	0	1	1	1	1	0	0	0	0	1
t_8	3	0	0	1	0	0	1	1	0	0	0	1	1	0	1	1

Bitmap-based Index Structures. Figure 1. Bitmap index examples for a table with one attribute with cardinality 4.

3–6 of Fig. 1 show the equality encoded bitmap for this table. The first tuple t_1 has value 1, therefore only the corresponding bit in the bitmap =1 is set. Columns 7 and 8, show the bitmaps for equality encoding with binning. In this example, the attribute was quantized into two bins.

For range encoded bitmaps [4], a bin is marked "1" if the value falls into it or a smaller bin, and "0" otherwise. Using this encoding, the last bin for each attribute is all 1s. Thus, this column is not explicitly stored. Columns 9–11 in Fig. 1 show the range encoding for the attribute. The first tuple t_1 has the smallest value 1, therefore all the bitmaps have the first bit set. For interval encoded bitmaps, every bitmap represents a range of $\lceil \frac{C}{2} \rceil$ values, where C is the cardinality of the attribute. This encoding allows to answer any range or point query on one attribute by reading at most two bit vectors [4,5]. Columns 12–14 in Fig. 1 show the interval encoded bitmap. The fifth tuple t_5 has value 2, therefore the fifth bit is set for the [1,2] and [2,3] interval bitmaps. Bit-sliced indexes (BSI) [8] can be considered as a special case of the encoded bitmaps [16]. With the bit-sliced index the bitmaps encode the binary representation of the attribute value. Columns 15–17 of Fig. 1 show the bit-sliced index of the table. The third tuple t_3 has value 3 which is represented by the binary number 011, therefore the corresponding bit in slices 2^1 and 2^0 are set. Encoded bitmaps encode the binary representation of the attribute bins. Therefore, only $\lceil \log_2 bins \rceil$ bitmaps are needed to represent all values.

With binning, several values are encoded in the same bitmap. However, the results from the queries are supersets of the actual results, therefore additional disk access may be needed to evaluate the candidates and retrieve the exact results. Several binning strategies, based on data distribution and query workloads, and several strategies for candidate evaluation have been proposed [10,11]. A binned bitmap index augmented with an auxiliary order-preserving bin-based Clustering (OrBiC) [19] can significantly reduce the I/O cost of candidate evaluation.

Query Execution

Bitmap indexes can provide efficient performance for point and range queries thanks to fast bitwise operations (AND, OR, NOT) which are supported by hardware. With equality encoded bitmaps a point query is executed by ANDing together the bit vectors corresponding to the values specified in the query conditions. For example, finding the data points that correspond to a query where Attribute 1 is equal to 3 and Attribute 2 is equal to 5 is only a matter of ANDing the two bitmaps together. Equality Encoded Bitmaps are optimal for point queries. Range queries are executed by first ORing together all the bit vectors specified by each range in the query conditions and then ANDing the answers together. If the query range for an attribute queried includes more than half of the cardinality then one executes the query by taking the complement of the ORed bitmaps that are not included in the query condition. The performance of queries with large-range query conditions can be improved by using multi-resolution bitmap indexes [12], i.e., bitmaps with different interval size: at the highest level one bit vector corresponds to an individual value and at lower levels each bit vector corresponds to a bin of values. Queries are executed using combined resolution bitmaps that minimize both the number of bitwise operations and the number of candidate points. With more complex encodings, the query evaluation strategy depends on the encoding and the range being queried.

Bitmap Compression

The goal of the bitmap index compression is twofold: to reduce the space requirement of the index and to maintain query execution performance by executing the query over the compressed index. The two most popular run-length compression techniques are the Byte-aligned Bitmap Code (BBC) [1] and the Word-Aligned Hybrid (WAH) compression method [18]. BBC stores the compressed data in bytes while WAH stores it in words. WAH is simpler because it only has two types of words, while BBC has four. Both techniques are based on the idea of run length encoding that represents consecutive bits of the same symbol (also called a fill or a gap) by their bit value and their length. The bit value of a fill is called the fill bit. BBC first divides the bit sequence into bytes and then group bytes into runs. A run consists of a fill word followed by a tail of literal bytes. A run always contains a number of whole bytes as it represents the fill length as number of bytes rather than number of bits. The byte alignment property limits a fill length to be an integer multiple of bytes. This ensures that during any bitwise logical operation a tail byte is never broken into individual bits. Similar to BBC, WAH is a hybrid between the run length encoding and the literal scheme. WAH stores compressed data in words rather than in bytes. The most significant bit of a word distinguishes between a literal word(0) and a fill word(1). Lower bits of a literal word contain the bit values from the bit sequence. The second most significant bit of a fill word is the fill bit and the lower (w-2) bits store the fill length. Imposing word alignment ensures that the logical operation functions only need to access words not bytes or bits. In general, bit operations over the compressed WAH bitmap file are faster than BBC while BBC gives better compression ratio.

Typically, complex encoded bitmaps do not compress well as the bit density of such bitmaps is relatively high. Recently, reordering has been proposed as a preprocessing step for improving the compression of bitmaps. The objective with reordering is to increase the performance of run length encoding. By reordering the data, the compression ratio of large boolean matrices can be improved [6]. However, optimal matrix reordering is NP-hard and the authors use traveling salesman heuristics to compute the new order. The idea of reordering is also applied to compression of bitmap indexes [9]. The authors show that the tuple reordering problem is NP-complete and propose a Gray code ordering heuristic.

Bitmap update is an expensive operation. The common practice is to drop the index, do a batch update, and recreate the index. For improved update performance, it is possible to add to each compressed bitmap a pad-word that encodes all possible rows using non-set bits [3]. For insertions, only the updated column needs to be accessed as opposed to all the bitmap columns. This technique is suitable for maintaining an online index when the update rate is not very high. An alternative to run-length compression proposed in the literature is the Approximate Bitmap Encoding [2], where hashing is used to encode the data objects in a bloom filter. This technique can offer improved performance for selection queries over small number of rows.

Key Applications

Bitmap indexing has been traditionally used in data warehouses to index large amount of data that are infrequently updated. More recently, bitmaps have found many other applications such as visualization and indexing of scientific data. Other works have applied bitmaps for term matching and similarity searches. In general, one could claim that bitmaps would outperform most approaches in the domains where full scan of the data are unavoidable and computations can be done using parallel bitwise operations.

Bitmap-based indexes are successfully implemented in commercial Database Management Systems. Oracle implements a version of the simple bitmap encoding compressed using BBC. Sybase IQ has two different types of bitwise indexes. IBM DB2 dynamically builds bitmaps from a single-column B-tree to join tables. Informix uses bitmapped indexes since version 8.21 released in 1998. PostgreSQL supports bitmap index scans since version 8.1 where dynamic bitmaps are created to combine the results of multiple index conditions using bitwise operations.

Cross-references

► Approximation and Data Reduction Techniques
► Indexing
► Query Processing and Optimization in Object Relational Databases

Recommended Reading

1. Antoshenkov, G. Byte-aligned bitmap compression. In Data Compression Conference, 1995. Oracle Corp.
2. Apaydin, T., Canahuate, G., Ferhatosmanoglu, H., and Tosun, A. Approximate encoding for direct access and query processing over compressed bitmaps. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.
3. Canahuate, G., Gibas, M., Ferhatosmanoglu, H. Update conscious bitmap indices. In Proc. 19th Int. Conf. on Scientific and Statistical Database Management, 2007.
4. Chan, C.Y. and Ioannidis, Y.E. Bitmap index design and evaluation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998.
5. Chan, C.Y. and Ioannidis, Y.E. An efficient bitmap encoding scheme for selection queries. ACM SIGMOD Rec., 1999.
6. Johnson, D., Krishnan, S., Chhugani, J., Kumar, S., and Venkatasubramanian, S. Compressing large boolean matrices using reordering techniques. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
7. Koudas, N. Space efficient bitmap indexing. In Proc. Int. Conf. on Information and Knowledge Management, 2000.
8. O'Neil, P. and Quass, D. Improved query performance with variant indexes. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997.
9. Pinar, A., Tao, T., and Ferhatosmanoglu, H. Compressing bitmap indices by data reorganization. In Proc. 21st Int. Conf. on Data Engineering, 2005.
10. Rotem, D., Stockinger, K., and Wu, K. Optimizing candidate check costs for bitmap indices. In Proc. Int. on Information and Knowledge Management, 2005.
11. Rotem, D., Stockinger, K., and Wu, K. Minimizing I/O costs of multi-dimensional queries with bitmap indices. In Proc. 18th Int. Conf. on Scientific and Statistical Database Management, 2006.

12. Sinha, R., Winslett, M. Multi-resolution bitmap indexes for scientific data. ACM Trans. Database Syst., 2007.
13. Stockinger, K. Design and implementation of bitmap indices for scientific data. In Proc. Int. Conf. on Database Eng. and Applications, 2001.
14. Stockinger, K., Wu, K., and Shoshani, A. Evaluation strategies for bitmap indices with binning. In Proc. 15th Int. Conf. Database and Expert Syst. Appl., 2004.
15. Wong, H.K., Liu, H., Olken, F., Rotem, D., and Wong, L. Bit transposed files. In Proc. 11th Int. Conf. on Very Large Data Bases, 1985.
16. Wu, M-C. and Buchmann, A. Encoded bitmap indexing for data warehouses. In Proc. 14th Int. Conf. on Data Engineering, 1998.
17. Wu, K., Otoo, E.J., and Shoshani, A. On the performance of bitmap indices for high cardinality attributes. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
18. Wu, K., Otoo, E.J., and Shoshani, A. Optimizing bitmap indexes with efficient compression. ACM Trans Database Syst., 2006.
19. Wu, K., Stockinger, K., Shoshani, A. Breaking the curse of cardinality on bitmap indexes. In Proc. 20th Int. Conf. on Scientific and Statistical Database Management, 2008.

Blind Signatures

Barbara Carminati
University of Insubria, Varese, Italy

Definition

In blind signature schemes, the sender and the signer of the message are two distinguished entities. Given a message m, blind signatures have the property that the signer digitally signs a blinded version of m, i.e., m', without the disclosure of any information about the original message. The obtained blind signature can be verified by using the public key of the signer and the original message m, instead of m'.

Key Points

In 1982, David Chaum introduced the concept of blind signatures for protecting user privacy during electronic payment transactions [1]. This scheme has been devised for scenarios where the sender and the signer of the message are two distinguished entities, with the aim of preventing the signer from observing the message he or she signs. More precisely, given a message m generated by A, a signer B is able to digitally sign a blinded version of m', i.e., $DS_B(m')$, without the disclosure of any information about the original message. The key property of blind signatures is that the obtained signature $DS_B(m')$ can be verified by using the public key of B and the original message m, instead of m'. This property makes blind signatures particularly suitable for applications where sender privacy is the main concern, like for instance, in digital cash protocols and electronic voting systems.

Several digital signature schemes can be used to obtain a blind signature, like for instance RSA and DSA schemes. In the following to explain the generation and verification of a blind signature, the RSA algorithm is considered. Let $PK_B = (n, e)$ and $SK_B = (n, d)$ be the public and private keys of signer B.

Blinding. Let k be a random integer chosen by the sender A, such that $0 \leq k \leq n - 1$ and $gcd(n, k) = 1$. Thus, given a message m, the sender generates the blinded version of it, i.e., m', by combing m with k. According to RSA, the blinded version of m is computed as $m' = (mk^e) \bmod n$.

Blind signature generation. The signer digitally signs the blinded version m' using the RSA digital signature algorithm; that is, $DS_B(m') = (m')^d \bmod n$.

Signature unblinding. The intended verifier can obtain the signature of m, i.e., $DS_B(m)$, as follows $DS_B(m) = k^{-1}DS_B(m')$. This signature can be validated by the RSA verification algorithm, according to the standard process.

Cross-references

▶ Digital Signatures
▶ Privacy Enhancing Technologies

Recommended Reading

1. Chaum D. Blind signatures for untraceable payments, advances in cryptology. In Proceedings of Crypto ’82, 1983, pp. 199–203.

Bloom Filter Join

▶ Semijoin

Bloom Filters

Michael Mitzenmacher
Harvard University, Boston, MA, USA

Synonyms

Hash filter

Definition

A Bloom filter is a simple, space-efficient randomized data structure based on hashing that represents a set in a way that allows membership queries to determine whether an element is a member of the set. False positives are possible, but not false negatives. In many applications, the space savings afforded by Bloom filters outweigh the drawbacks of a small probability for a false positive. Various extensions of Bloom filters can be used to handle alternative settings, such as when elements can be inserted and deleted from the set, and more complex queries, such as when each element has an associated function value that should be returned.

Historical Background

Burton Bloom introduced what is now called a Bloom filter in his 1970 paper [2], where he described the technique as an extension of hash-coding methods for applications where error-free methods require too much space and were not strictly necessary. The specific application he considered involved hyphenation: a subset of words from a standard dictionary require specialized hyphenation, while the rest could be handled by a few simple and standard rules. Keeping a Bloom filter of the words requiring specialized hyphenation dramatically cut down disk accesses. Here, false positives caused words that could be handled by the simple rules to be treated as special cases. Bloom filters were also used in early UNIX spell-checkers [12,13], where a false positive would allow a misspelled word to be ignored, and were suggested as a way of succinctly storing a dictionary of insecure passwords, where a false positive would disallow a potentially secure password [16].

Bloom filters were applied in databases to reduce the amount of communication and computation for join operations, especially distributed join operations [1,4,11]. The term Bloomjoin is sometimes used to describe a semijoin operation that utilizes Bloom filters. The Bloom filter is used to represent join column values, so that matching values can be found by querying the Bloom filter. False positives cause false matches of tuples that must later be removed. However, the communication and computation gains from using the filter can yield significant advantages even with these false positives.

Foundations

A Bloom filter for representing a set $S = \{x_1,x_2,...,x_n\}$ of n elements from a universe U consists of an array of m bits, initially all set to 0. The filter uses k independent hash functions $h_1,...,h_k$ with range $\{1,...,m\}$. For each element $x \in S$, the bits $h_i(x)$ are set to 1 for $1 \leq i \leq k$. (A location can be set to 1 multiple times.) To check if an item y is in S, one checks whether all $h_i(y)$ are set to 1. If not, then clearly y is not a member of S. If all $h_i(y)$ are set to 1, the data structure returns that y is in S. There is no possibility of a false negative, but it is possible that for $y \notin S$, all $h_i(y)$ are set to 1, in which case the data structure gives a false positive.

The fundamental issue in the analysis of the Bloom filter is the *false positive probability* for an element not in the set. In the analysis, generally it is assumed that the hash functions map each element in the universe to a random number independently and uniformly over the range. While this is clearly an optimistic assumption, it appears to be suitable for practical implementations. With this assumption, after all the elements of S are hashed into the Bloom filter, the probability that a specific bit is still 0 is

$$p' = (1 - 1/m)^{kn} \approx \mathrm{e}^{-kn/m}.$$

It is convenient to use the approximation $p = \mathrm{e}^{-kn/m}$ in place of p'. If ρ is the proportion of 0 bits after all the n elements are inserted in the table, then conditioned on ρ the probability f of a false positive is

$$f = (1-\rho)^k \approx (1-p')^k \approx (1-p)^k = \left(1 - \mathrm{e}^{-kn/m}\right)^k.$$

These approximations follow since $\mathrm{E}[\rho] = p'$, and ρ can be shown to be highly concentrated around p' using standard techniques [8].

The optimal number of hash function can be found by finding where $(1 - \mathrm{e}^{-kn/m})^k$ is minimized as a function of k. Simple calculus reveals the optimum occurs when $k = \ln 2 \cdot (m/n)$, giving a false positive probability f of

$$f = \left(1 - \mathrm{e}^{-kn/m}\right)^k = (1/2)^k \approx (0.6185)^{m/n}.$$

In practice, k must be an integer, and the choice of k might depend on application-specific questions.

Key Applications

Essentially any application that requires membership checks against a list or set of objects, and for which space is at a premium, is a candidate for a Bloom filter. In some applications, Bloom filters may also save computational resources. The consequences of false positives, however, need to be carefully considered in all circumstances.

Bloom filters enjoyed a recent resurgence in the networking community after their use in a paper by Fan et al. on distributed Web caches [9]. Instead of having caches distribute lists of URLs (or lists of their 16-byte MD5 hashes) corresponding to the cache contents, the authors demonstrate a system that saves in communication costs by sharing Bloom filters of URLs. False positives may cause a server to request a page from another nearby server, even when that server does not hold the page. In this case, the page must subsequently be retrieved from the Web. If the false positive probability is sufficiently low, the penalty from these occasional failures is dominated by the improvement in overall network traffic. Countless further applications in databases, peer-to-peer networks, overlay networks, and router architectures have arisen, and the use of Bloom filters and their many variants and extensions has expanded dramatically [5].

In databases, applications include the previously mentioned Bloomjoin variation of the semijoin. In a similar manner, Bloom filters can be effectively used to estimate the size of semijoin operations, using the fact that Bloom filters can be used to estimate the size of a set (and the size of set unions and intersections) [14]. Another early database application utilizes differential files [10]. In this setting, all changes to a database that occur during a given time period are processed as a batch job, and a differential file tracks changes that occur until the batched update occurs. To read a record then requires determining if the record has been changed by some transaction in the differential file. If not, the record can be read directly from the database, which is generally much quicker and more efficient than performing the necessary processing on the differential file. Instead of keeping a list of all records that have changed since the last update, a Bloom filter of the records that have been changed can be kept. Here, a false positive would force the database to check both the differential file and the database on a read even when a record has not been changed. If false positives are sufficiently rare, this cost is minimal.

Future Directions

Recent research related to Bloom filters has followed three major directions: alternative constructions, improved analysis, and extensions to more challenging questions. All three of these directions are likely to continue to grow.

Research in alternative constructions has focused on building data structures with the same capabilities as a Bloom filter or related data structures with improved efficiency, sometimes particularly for a specialized domain, such as router hardware. Improved analysis has considered reducing for example the amount of randomness required for a Bloom filter, or its performance under specific classes of hash functions.

The largest area of research has come from extending the basic idea of the Bloom filter to more general and more difficult problems. Arguably, the success of Bloom filters relies on their flexibility, and numerous variations on the theme have arisen. For example, counting Bloom filters extend Bloom filters by allowing sets to change dynamically via insertions and deletions of elements [9,15]. Spectral Bloom filters extend Bloom filters to handle multi-sets [7]. Count-min sketches extend Bloom filters to approximately track counts associated to items in a data stream, such as byte-counts for network flows [8]. Bloomier filters extend Bloom filters to the situation where each element of S is associated with a function value from a discrete, finite set of values, and the data structure should return that value [6]. Approximate concurrent state machines further extend Bloom filter to the setting where both the set can change due to insertions and deletions and the function values associated with set elements can change dynamically [3]. The list of variations continues to grow steadily as more applications are found.

URL to Code

- en.wikipedia.org/wiki/Bloom_filter
- www.patrow.net/programming/hashfunctions/index.html
- search.cpan.org/~mceglows/Bloom-Filter-1.0/Filter.pm

Cross-references

▶ Join
▶ Semijoin

Recommended Reading

1. Babb E. Implementing a relational database by means of specialized hardware. ACM Trans. Database Syst., 4(1):1–29, 1979.
2. Bloom B. Space/time tradeoffs in hash coding with allowable errors. Commun. ACM, 13(7):422–426, 1970.
3. Bonomi F., Mitzenmacher M., Panigrahy R., Singh S., and Varghese G. Beyond Bloom filters: from approximate membership checks to approximate state machines. Comput. Commun. Rev., 36(4):315–326, 2006.
4. Bratbergsengen K. Hashing methods and relational algebra operations. In Proc. 10th Int. Conf. on Very Large Data Bases, 1984, pp. 323–333.
5. Broder A. and Mitzenmacher M. Network applications of Bloom filters: a survey. Internet Math., (4):485–509, 2005.
6. Chazelle B., Kilian J., Rubinfeld R., and Tal A. The Bloomier filter: an efficient data structure for static support lookup tables. In Proc. 15th Annual ACM-SIAM Symp. on Discrete Algorithms, 2004, pp. 30–39.
7. Cohen S. and Matias Y. Spectral Bloom filters. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 241–252.
8. Cormode G. and Muthukrishnan S. An improved data stream summary: the count-min sketch and its applications. J. Algorithms, 55(1):58–75, 2003.
9. Fan L., Cao P., Almeida J., and Broder A.Z. Summary cache: a scalable wide-area Web cache sharing protocol. IEEE/ACM Trans. Network., 8(3):281–293, 2000.
10. Gremilion L.L. Designing a Bloom filter for differential file access. Commun. ACM, 25:600–604, 1982.
11. Mackett L.F. and Lohman G.M. R* optimizer validation and performance evaluation for distributed queries. In Proc. 27th Int. Conf. on Very Large Data Bases, 1986, pp. 149–159.
12. McIlroy M.D. Development of a spelling list. IEEE Trans. Commun., 30(1):91–99, January 1982.
13. Mullin J.K. and Margoliash D.J. A tale of three spelling checkers. Software Pract. Exp., 20(6):625–630, June 1990.
14. Mullin J.K. Estimating the size of a relational join. Inf. Syst., 18(3):189–196, 1993.
15. Mitzenmacher M. Compressed Bloom filters. IEEE/ACM Trans. Network., 10(5):604–612, October 2002.
16. Spafford E.H. Opus: preventing weak password choices. Comp. Sec., 11:273–278, 1992.

Bloom Join

► Semijoin

BM25

Giambattista Amati

Ugo Bordoni Foundation, Rome, Italy

Synonyms

OKAPI retrieval function; Probabilistic model

Definition

BM25 is a ranking function that ranks a set of documents based on the query terms appearing in each document, regardless of the inter-relationship between the query terms within a document (e.g., their relative proximity). It is not a single function, but actually a whole family of scoring functions, with slightly different components and parameters. It is used by search engines to rank matching documents according to their relevance to a given search query and is often referred to as "Okapi BM25," since the Okapi information retrieval system was the first system implementing this function. The BM25 retrieval formula belongs to the BM family of retrieval models (BM stands for Best Match), that is the weight of a term **t** in a document **d** is:

$$\frac{\mathbf{tf}}{k+\mathbf{tf}}\ln\frac{(r_{\mathbf{t}}+0.5)\cdot(\mathbf{N}-R-\mathbf{n_t}+r_{\mathbf{t}}+0.5)}{(\mathbf{n_t}-r_{\mathbf{t}}+0.5)\cdot(R-r_{\mathbf{t}}+0.5)} \qquad [BM\text{'s family}]$$

where

- R is the number of documents known to be relevant to a specific topic,
- $r_{\mathbf{t}}$ is the number of relevant documents containing the term,
- **N** is the number of documents of the collection,
- $\mathbf{n_t}$ is the document frequency of the term,
- **tf** is the frequency of the term in the document,
- k is a parameter.

The BM25 document-query matching function is:

$$\sum_{\mathbf{t}\in\mathbf{q}} w_{\mathbf{t}}\cdot \mathrm{In}\frac{(r_{\mathbf{t}}+0.5)\cdot(\mathbf{N}-R-\mathbf{n_t}+r_{\mathbf{t}}+0.5)}{(\mathbf{n_t}-r_{\mathbf{t}}+0.5)\cdot(R-r_{\mathbf{t}}+0.5)} \; [BM25] \tag{1}$$

where

- **q** is the query,
- $w_{\mathbf{t}}=(k_1+1)\frac{\mathbf{tf}}{k+\mathbf{tf}}\cdot(k_3+1)\frac{\mathbf{tf}_q}{(k_3+\mathbf{tf_q})}$
- $\mathbf{tf}_q$ is the frequency of the term within the topic from which **q** was derived
- l and $\bar{\mathbf{l}}$ are respectively the document length and average document length.

- k is $k_1\left((1-b)+b(\frac{l}{\bar{\mathbf{l}}})\right)$.
- k_1, b and k_3 are parameters which depend on the nature of the queries and possibly on the database.
- k_1 and b are set by default to 1.2 and 0.75 respectively, k_3 to 1000.

By using these default parameters, the unexpanded BM25 ranking function, that is the BM25 applied in the absence of information about relevance ($R=r=0$), is:

$$\sum_{\mathbf{t}\in\mathbf{q}} \frac{2.2\cdot\mathbf{tf}}{0.3+0.9\frac{l}{\bar{\mathbf{l}}}+\mathbf{tf}}\cdot\frac{1001\cdot\mathbf{tf}_q}{1000+\mathbf{tf}_\mathbf{q}}\ln\frac{N-\mathbf{n_t}+0.5}{\mathbf{n_t}+0.5} \tag{2}$$

Historical Background

The successful formula of the BM family, the BM25, was introduced by Robertson and Walker in 1994 [1,3] and it has its root in Harter's 2-Poisson model of litheness for indexing. Before BM25, a direct exploitation of itheness for document retrieval was explored by Robertson, Van Rijsbergen, Porter, Williams and Walker [3,4] who plugged the Harter 2-Poisson model into the standard probabilistic model of relevance o Robertson and Spark Jones [2]. The evolution of the 2-Poisson model as designed by Robertson, Van Rijsbergen and Porter has thus motivated the birth of the BM family of term-weighting forms. The BM25 formula is the matching function of the Okapi information retrieval system of City University in London.

Foundations

BM25 derives from both the 2-Poisson model and the probabilistic binary independence model of relevance, that is as a combination of the probabilistic model

$$\mathrm{Prob}(rel,\mathbf{d}|\mathbf{q}) \propto \prod_{\mathbf{t}\in\mathbf{q}\cap\mathbf{d}} \frac{\mathrm{Prob}(\mathbf{t}|rel)\cdot\mathrm{Prob}(\bar{\mathbf{t}}|\overline{rel})}{\mathrm{Prob}(\mathbf{t}|\overline{rel})\cdot\mathrm{Prob}(\bar{\mathbf{t}}|\overline{rel})} \tag{3}$$

and the 2-Poisson model

$$\mathrm{Prob}(X=\mathbf{tf}) = p\cdot\frac{e^{-\lambda_{\mathbf{E_t}}}\lambda_{\mathbf{E_t}}{}^{\mathbf{tf}}}{\mathbf{tf}!}+(1-p)\cdot\frac{e^{-\lambda_{\overline{\mathbf{E_t}}}}\lambda_{\overline{\mathbf{E_t}}}{}^{\mathbf{tf}}}{\mathbf{tf}!} \tag{4}$$

where p is the probability of a document to belong to the Elite set.

However, the 2-Poisson has three parameters: the mean term frequency of the term in the elite set $\lambda_{\overline{\mathbf{E_t}}}$ (a set of documents with a large number of occurrences of the term), the mean term frequency of the term in the rest of the collection, $\lambda_{\overline{\mathbf{E_t}}}$, and the mixing parameter p. Therefore, the 2-Poisson model needs reasonable approximations in order to make the probabilistic mixture a workable retrieval model.

The combination of the notion of eliteness with that of relevance generates the Robertson, Van Rijsbergen and Porter's query term-document matching function:

$$w = \ln\frac{(p_1\lambda_{\mathbf{E_t}}{}^{\mathbf{tf}}e^{-\lambda_{\mathbf{E_t}}}+(1-p_1)\lambda_{\overline{\mathbf{E_t}}}{}^{\mathbf{tf}}e^{-\lambda_{\overline{\mathbf{E_t}}}})}{(p_2\lambda_{\mathbf{E_t}}{}^{\mathbf{tf}}e^{-\lambda_{\mathbf{E_t}}}+(1-p_2)\lambda_{\overline{\mathbf{E_t}}}{}^{\mathbf{tf}}e^{-\lambda_{\overline{\mathbf{E_t}}}})} \frac{(p_2e^{-\lambda_{\mathbf{E_t}}}+(1-p_2)e^{-\lambda_{\overline{\mathbf{E_t}}}})}{(p_1e^{-\lambda_{\mathbf{E_t}}}+(1-p_1)e^{-\lambda_{\overline{\mathbf{E_t}}}})} \tag{5}$$

where p_1 and p_2 are the conditional probabilities of a document to be or not in the elite set respectively, given the set of relevant documents. Since elite set (documents with high query-term frequency) and relevance (documents relevant to the query) are highly correlated then it can be assumed that $p_1 > p_2$.

With little algebra, (5) is equivalent to

$$w = \ln\frac{\left(p_1+(1-p_1)\left(\frac{\lambda_{\overline{\mathbf{E_t}}}}{\lambda_{\mathbf{E_t}}}\right)^{\mathbf{tf}}e^{\lambda_{\mathbf{E_t}}-\lambda_{\overline{\mathbf{E_t}}}}\right)}{\left(p_2+(1-p_2)\left(\frac{\lambda_{\overline{\mathbf{E_t}}}}{\lambda_{\mathbf{E_t}}}\right)^{\mathbf{tf}}e^{\lambda_{\mathbf{E_t}}-\lambda_{\overline{\mathbf{E_t}}}}\right)} \frac{\left(p_2e^{-\lambda_{\mathbf{E_t}}+\lambda_{\overline{\mathbf{E_t}}}}+(1-p_2)\right)}{\left(p_1e^{-\lambda_{\mathbf{E_t}}+\lambda_{\overline{\mathbf{E_t}}}}+(1-p_1)\right)} \tag{6}$$

Equation (6) can be rewritten as

$$w(\mathbf{tf}) = \ln C(\mathbf{tf}) + \ln C_0$$

where C_0 is the ratio of the two components of the cross-product ratio not containing the variable **tf**. The first derivative with respect to the variable **tf** is

$$\frac{dw}{d\mathbf{tf}} = \frac{(p_1-p_2)\cdot e^{\lambda_{\mathbf{E_t}}-\lambda_{\overline{\mathbf{E_t}}}}\cdot\left(\frac{\lambda_{\overline{\mathbf{E_t}}}}{\lambda_{\mathbf{E_t}}}\right)^{\mathbf{tf}}\cdot\ln\left(\frac{\lambda_{\mathbf{E_t}}}{\lambda_{\overline{\mathbf{E_t}}}}\right)}{C(\mathbf{tf})}$$

The derivative is always positive because $p_1 > p_2$, $\lambda_{\overline{\mathbf{E_t}}} \ll \lambda_{\mathbf{E_t}}$ and thus $\ln\left(\frac{\lambda_{\mathbf{E_t}}}{\lambda_{\overline{\mathbf{E_t}}}}\right) > 0$ in the 2-Poisson model. Therefore $w(\mathbf{tf})$ is a monotonically increasing function. The limiting form of (6) for **tf** $\rightarrow\infty$ is

$$w = \ln \frac{p_1\left(p_2 e^{-\lambda_{\mathbf{E_t}}+\lambda_{\overline{\mathbf{E_t}}}} + (1-p_2)\right)}{p_2\left(p_1 e^{-\lambda_{\mathbf{E_t}}+\lambda_{\overline{\mathbf{E_t}}}} + (1-p_1)\right)}$$

Since $e^{-\lambda_{\mathbf{E_t}}+\lambda_{\overline{\mathbf{E_t}}}} \sim 0$, this limit is very close to

$$w \sim \mathrm{In} \frac{p_1(1-p2)}{p_2(1-p1)} \tag{7}$$

Because (6) is monotonic with respect to the within–document term-frequency **tf**, document ranking is obtained by decreasing order of the **tf** value. Hence because for the highest values of the term frequency **tf** (i.e., for **tf** → ∞), the limiting form of (7) can be taken as the actual score of the topmost documents.

In 1994, Robertson and Walker defined as an approximation of w of (7) the product

$$w = \frac{\mathbf{tf}}{\mathbf{tf}+K} \cdot \mathrm{In} \frac{\mathrm{Prob}(\mathbf{t}|rel)\mathrm{Prob}(\bar{\mathbf{t}}|\overline{rel})}{\mathrm{Prob}(\mathbf{t}|\overline{rec})\mathrm{Prob}(\bar{\mathbf{t}}|\overline{rel})} \tag{8}$$

Indeed, both (6) and (8) have Formula 7 as their limit for large **tf**. Varying the parameter K and using equation

$$\frac{\mathrm{Prob}(\mathbf{t}|rec)\mathrm{Prob}(\bar{\mathbf{t}}|\overline{rec})}{\mathrm{Prob}(\mathbf{t}|\overline{rec})\mathrm{Prob}(\bar{\mathbf{t}}|rec)} = \frac{(r_{\mathbf{t}}+0.5)\cdot(\mathbf{N}-R-\mathbf{n_t}+r_{\mathbf{t}}+0.5)}{(\mathbf{n_t}-r_{\mathbf{t}}+0.5)\cdot(R-r_{\mathbf{t}}+0.5)} \tag{9}$$

the BM weighting formulas are derived.

Cross-references

- ▶ Divergence from Randomness Models
- ▶ Information Retrieval
- ▶ Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model
- ▶ Relevance
- ▶ Term Weighting
- ▶ TF*IDF
- ▶ Two-Poisson Model

Recommended Reading

1. Robertson S.E., Walker S., Beaulieu M.M., Gatford M., and Payne A. Okapi at trec-4. In Harman D.K. (ed.). NIST Special Publication 500-236: In Proc. The 4th Text Retrieval Conference. 1996.
2. Robertson S.E. and Sparck-Jones K. Relevance weighting of search terms. J. Am. Soc. Inform. Sci., 27:129–146, 1976.
3. Robertson S.E. and Walker S. Some simple approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, June 1994, pp. 232–241.
4. Robertson S.E., Van Rijsbergen C.J., and Porter M. Probabilistic models of indexing and searching. In Robertson S.E., Van Rijsbergen C.J., and Williams P.W. (eds.). Information retrieval Research, Chap. 4, Butterworths, 1981, pp. 35–56.

Boolean Model

Massimo Melucci
University of Padua, Padua, Italy

Definition

In the Boolean Model for Information Retrieval, a document collection is a set of documents and an index term is the subset of documents indexed by the term itself. An index term can also be seen as a proposition which asserts whether the term is a property of a document, that is, if the term occurs in the document or, in other words, if the document is about the concept represented by the term.

The interpretation of a query is set-theoretical. In practice, a query is a Boolean expression where the set operators are the usual intersection, union and complement, and the operands are index terms. The document subsets which corresponds to the index terms of the query are combined through the set operators. The system returns the documents which belong to the subset expressed by the query.

Historical Background

The Boolean model for Information Retrieval was proposed as a paradigm for accessing large scale systems since the 1950s. The idea of composing queries as Boolean propositions was at that time considered advantageous for the end user and simple to implement, so that its use rapidly spread in the 1960s. The main reason for the rapid advent of the Boolean model was the presence of experienced end users who also were expert of the domain of the searched documents. These users were expected to be able to quite effectively express their own information needs by composing index terms and operators as propositions. More recently, some user studies suggested that even experienced users tended to employ the less Boolean operators as possible and to submit quite simple queries.

A possible reason for the potential success of the Boolean model was the controlled, coordinated and very often manual indexing of the collections; these terms were often quite homogeneous terms, namely, referred to a restricted domain, therefore, indexing carefully avoided ambiguous terms and correctly associated related terms. Since then, the Boolean model was adopted by various other information management systems, e.g., office information systems, library catalogue systems, and database systems.

Currently, the Boolean model is quite ubiquitous since the advanced search functions constitute a constant of almost every information management systems, Web search engines included. However, it is a matter of fact that the success of this model decreased over time because of the decreasing proportion of experienced end users, the increasing heterogeneity of the document collections and the lack of a support for ranking documents. The most glaring example is the World Wide Web, which fuels the search engine indexes with more and more heterogenous Web pages accessed by an increasing number of unexperienced end users. Various research works showed that the percentage of users who use advanced search functions is very limited, even when they are quite expert of the domain or in using the interface.

Foundations

A parallel can be established between database design and Boolean model. As far as database systems are concerned, the design process aims at structuring entities, relationships and attributes accessed through relational query languages. When Boolean Information Retrieval systems are considered, the document properties are index terms and the design process aims at selecting the index terms of the document through an indexing process. Once every document has been indexed, the document collection is described by an index whose terms are associated to the documents in which they occur. The basic information retained by the system is the membership of the document to the index term.

Suppose, for example, some authors are writing their own documents, say, d_1,d_2,d_3 using, say, three terms, namely, A,B,C. The terms *unambiguously* describe one concept each – the absence of ambiguity of the natural language is an important assumption so as to make this model effective. As a document may be about one or two concepts, a document may belong to more subsets. Suppose, for example, that the collection stores the documents d_1,d_2,d_3, which respectively contains the following index terms: $\{A,C\}$, $\{A,B\}$, $\{B\}$. According to the Boolean model, three subsets $A = \{d_1,d_2\}$, $B = \{d_2,d_3\}$, $C = \{d_1\}$ are defined after indexing the collection. (It is customary to label the subset with the index term.)

The example shows that a concept is addressed by one or more documents and a term is associated to a single document subset. These document subsets are the extensional expressions of the concept described by an index term. In other words, the enumeration of the document subset is the description of the concept labeled by the index term. Indeed, the Boolean model was thought as a means for describing concepts without recurring to ontological description, but to a simple, yet powerful logical language.

The index terms and the associated document subsets are decided at indexing time by a human indexer or an automated process driven by an indexing algorithm. If the set operators were not available, the end user could use one index term at a time for formulating his own query thus having a very few degrees of freedom for formulating his own information need. Thanks to the set-based view imposed by the Boolean model, the end user can utilize the Boolean algebra for constructing new subsets. These new subsets are what results from Boolean expressions based on the classical operators, namely, intersection, union and complement. Suppose, for example, A,B,C are the document subsets associated to three index terms; let $A = \{d_1,d_2\}$, $B = \{d_2,d_3\}$, $C = \{d_1\}$ and A AND NOT C be a new subset. This subset contains d_2 only thus expressing the fact that d_2 is about A AND NOT C.

What is important here is that the construction of document subsets through the Boolean algebra allows the end user for expressing new concepts which were not explicitly thought by the authors. Indeed, the author of d_2 did not think that d_2 was about A AND NOT C. With this respect, the Boolean model is a powerful language because would permit to represent the knowledge stored in a document collection by means of an algebra. A system based on the Boolean model can efficiently answer these queries by performing some simple algorithms which implement set operations and process sets.

Despite its strengths and simplicity, the Boolean model has some weaknesses. Understanding the main weaknesses of the Boolean model is crucial for making the application of this model to realistic contexts effective. The weaknesses of the Boolean model can be summarized as follows (see also Cooper, 1988):

- Set operator confusion
- Expressiveness gap
- Index term ambiguity
- Null output
- Output overload

Null output and output overload are due to the lack of support for ranking documents.

Set operator confusion occurs whenever intersection and union, namely, `AND` and `OR` are exchanged when composing a query. Because of the imprecision of the natural language, humans often use "or" for saying "and," and viceversa. This confusion causes a similar exchange when using the artificial Boolean language. Suppose, for example, the collection is $\{d_4,d_5\}$, d_4 is about *A* and d_5 is about *B*. Using the natural language, end user's information need might be "the documents about *A and B*" to mean the whole document collection. To obtain the whole collection as a result, the user should use *A* `OR` *B*. However, if the user translated the "and" of his own natural language request into an `AND` operator, the translation of the request into an artificial language would be *A* `AND` *B*, which returns the empty set. Things are more difficult as the query is more complex.

Expressiveness gap is the loss of expressiveness encountered whenever an information need has to be translated to an artificial language like the Boolean language from a more expressive natural language. Suppose, for example, the end user wants to retrieve documents about the Boolean language in Information Retrieval or Database Systems. The use of the Boolean query language requires that each word or group of words is corresponded to an index term, namely, to a document subsets, and that some set operators are applied to these sets; for example, a query might be `(boolean AND information retrieval) OR database systems`. The problem of the expressiveness gap is amplified by the multiplicity of queries which can be formulated as a representation of the same information need. As a consequence, a variety of document subsets not all being the same can be retrieved. Suppose, for example, the end user wants to retrieve documents about the Boolean language in Information Retrieval or Database Systems. Potential Boolean queries are, for example:

- `(boolean AND information retrieval) OR database systems`
- `boolean AND (information retrieval OR database systems)`

The expressiveness gap occurs because the request expressed in natural language does not indicate if and where the parentheses are located, whereas the location of the parentheses is crucial when using the Boolean algebra. Although both are valid expressions, they provide different results.

Index term ambiguity occurs whenever a term has two distinct meanings (polysemy) or two terms have the same meaning (synonymy). When a term is polysemous, documents about two distinct concepts co-occur in the same subset. As a consequence, irrelevant documents may be retrieved when the end user employs the term in his own query. Suppose, for example, `bank` = $\{d_6,d_7\}$, but d_6 is about bank as river bank and d_7 is about bank as bank branch. When the user expresses his own Boolean query using `bank` as term, both documents are retrieved. If, however, the user needs information about river banks, one irrelevant document, i.e., d_7, is retrieved. In this event, precision is lower than the precision measured if ambiguity does not occur. When two terms are synonyms, two distinct document subsets are defined, yet the documents are about the same concept. As a consequence, relevant documents may be missed when the end user employs only one of the two terms in his own query. Suppose, for example, `personalcomputer` = $\{d_8,d_9\}$ and `PC` = $\{d_{10}\}$, but `PC` is the acronym of `personal computer` and therefore d_{10} should be retrieved, but it is not. When the user expresses his own Boolean query using `PC` as term, only d_{10} is retrieved. If, however, the use needs information about personal computers, two relevant documents, i.e., d_8,d_9, are missed. In this event, recall decreases.

Null output occurs whenever the end user submits a very restrictive query to the system to an extent to make the returned document subset very small, if not even empty. Limit examples are terms being absent from the index, which are associated to the empty document set or two index terms whose intersection is empty because there are no common documents. The event of intersecting disjoint document subsets is very probable since the document subsets are usually small if compared with the document collection. Moreover, Boolean systems do not usually keep track of the semantic relationships between terms, and therefore connecting by AND two semantically related terms whose document subsets are disjoint would give the empty set as result. To reduce null output, some query terms can be related by OR or some AND should be removed by the user.

Output overload occurs whenever the result document subset is too large for being effectively browsed by the end user. This drawback often happens when too many ORs or too few ANDs are utilized for formulating the query. Even though a document subset is small if compared with the document collection, a subset of, say, 1,000 documents is very large for the end user who is required to inspect all of them. A strategy for avoiding output overload is reducing the number of OR or adding some AND.

Overall, the side effects of null output and output overload are hardly controlled by the end user who is requested to add and delete terms or Boolean operators so as to make the query an effective description of his own information need. This task, which is called Query Expansion, would overload the cognitive effort of the end user, if performed using a Boolean systems.

To overcome the problems of null output and output overload, the notion of level of coordination was introduced. The level of coordination is a measure of the degree to which each returned document matches the query. In this way, the level of coordination provides a score for ranking the documents. This ranking allows the user for deciding how many documents to inspect and the system for cutting the bottom ranked documents off the list.

The level of coordination is calculated as follows. A Boolean query is transcribed in conjunctive normal form, namely, as a list of propositions related by AND – each proposition is a disjunction of terms. The level of coordination of a document is the number of propositions satisfied by the document. Suppose, for example, A,B,C are the document subsets associated to three index terms; let $A = \{d_1,d_2\}$, $B = \{d_2,d_3\}$, $C = \{d_3\}$ and A AND (C OR B) be the query; the level of coordination of d_1 is 1 because only the first proposition is satisfied by d_1, the level of d_2 is 2 and that of d_3 is 1 – d_1,d_3 would have not been retrieved if the level of coordination were not been calculated because they do not satisfy the query.

A variation of the level of coordination was introduced for taking the variable size of document subsets into account – indeed, the document subsets are of arbitrary size and therefore a small subset may be treated as a large subset. This variation has been called weighted level of coordination: instead of assigning a constant weight to each proposition made true by a document, a different weighted is assigned depending on the proposition; for example, an IDF may be used. The weighed level of coordination is then the sum of the weights assigned to every proposition.

Key Applications

The research papers and articles on the Boolean model flourished until when other models, e.g., the probabilistic models and the vector-space model, appeared on the scene of the Information Retrieval theater and the Web search engines drastically changed the audience and the document collections. The knowledge of this model, however, is a basic element of an Information Retrieval system because the Boolean query language is provided by many if not all information management systems. Of course, this model is the starting point for facing the non-classical logic models for Information Retrieval.

Cross-references

► Indexing Units
► Logical Models of Information Retrieval
► Precision
► Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model
► Query Expansion for Information Retrieval
► Query Expansion Models
► Recall
► Relevance Feedback
► Vector-Space Model

Recommended Reading

1. Bar-Hillel Y. Language and information. Addison-Wesley, Reading, MA, USA, 1964.
2. Belkin N.J., Cool C., Croft W.B., and Callan J.P. The Effect of Multiple Query Representations on Information Retrieval System Performance. In Proc. 16th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 339–346.
3. Blair D. Language and representation in information retrieval. Elsevier, Amsterdam, 1990.
4. Cooper W. Getting beyond Boole. Inform. Process. Manage., 24:243–248, 1988.
5. Croft W., Turtle H., and Lewis D. The Use of phrases and structured queries in information retrieval. In Proc. 14th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1991, pp. 32–45.

6. Grefenstette G. (ed.), Cross-Language Information Retrieval, International Series on Information Retrieval, Kluwer Academic, Dordecht, 1998.
7. Hersh W. and Hickam D. An evaluation of interactive Boolean and natural language searching with an online medical textbook. J. Am. Soc. Inform. Sci., 46(7):478–489, 1995.
8. Hull D. A weighted Boolean model for Cross Language Text Retrieval. In Grefenstette [6], pp. 119–136.
9. Korfhage R. Information Storage and Retrieval. Wiley, New York, 1997.
10. Kowalski G. and Maybury M. Information retrieval systems: Theory and implementation. Kluwer, Dordecht, 2000.
11. Lancaster F. and Warner A. Information retrieval today. Information Resources, Arlington, VA, 1993.
12. Lee J. Properties of Extended Boolean Models in Information Retrieval. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1994, pp. 182–190.
13. Lee J., Kim W., Kim M., and Lee Y. On the evaluation of Boolean operators in the extended Boolean retrieval framework. In Proc. 16th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 291–297.
14. Radecki T. Generalized boolean methods of information Retrieval. Int. J. Man–Machine Studies, 18(5):407–439, 1983.
15. van Rijsbergen C. The Geometry of Information Retrieval. Cambridge University Press, UK, 2004.
16. Wong S., Ziarko W., Raghavan V., and Wong P. Extended boolean query processing in the generalized vector space model. Inform. Syst., 14(1):47–63, 1989.

Boosting

Zhi-Hua Zhou

Nanjing University, Nanjing, China

Definition

Boosting is a kind of ensemble method which produces a strong learner that is capable of making very accurate predictions by combining rough and moderately inaccurate learners (which are called as *base learners* or *weak learners*). In particular, Boosting sequentially trains a series of base learners by using a *base learning algorithm*, where the training examples wrongly predicted by a base learner will receive more attention from the successive base learner. After that, it generates a final strong learner through a weighted combination of these base learners.

Historical Background

In 1988, Kearns and Valiant posed an interesting question for the research of computational learning theory, i.e., whether a *weak* learning algorithm that performs just slightly better than random guess can be "boosted" into an arbitrarily accurate *strong* learning algorithm. In other words, whether two complexity classes, *weakly learnable* and *strongly learnable* problems, are equal. In 1989, Schapire [9] proved that the answer to the question is "yes", and the proof he gave is a construction, which is the first Boosting algorithm. One year later, Freund developed a more efficient algorithm. However, both algorithms suffered from some practical drawbacks. Later, in 1995, Freund and Schapire [4] developed the AdaBoost algorithm

```
Input: Data set $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$;
       Base learning algorithm $\mathcal{L}$;
       Number of learning rounds $T$.
Process:
  $D_1(i) = 1/m$.        % Initialize the weight distribution
  for $t = 1,\cdots,T$:
      $h_t = \mathcal{L}(\mathcal{D}, D_t)$;        % Train a base learner $h_t$ from $\mathcal{D}$ using distribution $D_t$
      $\epsilon_t = \Pr_{i \sim D_i}[h_t(x_i \neq y_i)]$;      % Measure the error of $h_t$
      $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$; % Determine the weight of $h_t$
      $D_{t+1}(\mathrm{i}) = \frac{D_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(x_i) = y_i \\ \exp(\alpha_t) & \text{if } h_t(x_i) \neq y_i \end{cases}$
               $= \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$   % Update the distribution, where $Z_t$ is
                     % a normalization factor which enables $D_{t+1}$ be a distribution
  end.
Output: $H(x) = \mathrm{sign}\,(f(x)) = \mathrm{sign}\left(\Sigma_{t=1}^{T} \alpha_t h_t(x)\right)$
```

Boosting. Figure 1. The AdaBoost algorithm.

which is effective and efficient in practice, and then a hot wave of research on Boosting arose.

Foundations

AdaBoost is the most influential Boosting algorithm. Let $\mathcal{X}$ and $\mathcal{Y}$ denote the instance space and the set of class labels, respectively, and assume $\mathcal{Y} = \{-1, +1\}$. Given a training set $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ $(i = 1, \ldots, m)$, and a base learning algorithm which can be decision tree, neural networks or any other learning algorithms, the AdaBoost algorithm works as follows.

First, it assigns equal weights to all the training examples (x_i, y_i) $(i \in \{1, \ldots, m\})$. Denote the distribution of the weights at the tth learning round as D_t. From the training set and D_t the algorithm generates a base learner $h_t : \mathcal{X} \to \mathcal{Y}$ by calling the base learning algorithm. Then, it uses the training examples to test h_t, and the weights of the incorrectly classified examples will be increased. Thus, an updated weight distribution D_{t+1} is obtained. From the training set and D_{t+1} AdaBoost generates another base learner by calling the base learning algorithm again. Such a process is repeated for T times, each of which is called a *round*, and the final learner is derived by weighted majority voting of the T base learners, where the weights of the learners are determined during the training process. In practice, the base learning algorithm may be a learning algorithm which can use weighted training examples directly; otherwise the weights can be exploited by sampling the training examples according to the weight distribution D_t. The pseudo-code of AdaBoost is shown in Fig. 1.

Freund and Schapire [4] proved that the training error of the final learner H is upper-bounded by

$$\epsilon_D = \Pr_{i \sim \mathcal{D}}[H(x_i) \neq y_i] \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t (1 - \epsilon_t)},$$

which can be written as

$$\epsilon_D \leq \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2 \leq \exp} \left(-2 \sum_{t=1}^{T} \gamma_t^2 \right),$$

where $\gamma_t = 1/2 - \epsilon_t$. Thus, if each base learner is slightly better than random so that $\gamma_t \geq \gamma$ for some $\gamma > 0$, the training error will drop exponentially fast in T since the upper bound is at most $e^{-2T\gamma^2}$.

Freund and Schapire [4] also gave a generalization error bound of H in terms of its training error $\epsilon_{\mathcal{D}}$, the size m of the training set, the VC-dimension d of the base learner space, and the number of rounds T, by

$$\epsilon \leq \epsilon_{\mathcal{D}} + \tilde{O}\left(\sqrt{\frac{Td}{m}} \right)$$

with high probability, where $\tilde{O}(\cdot)$ is used to hide all logarithmic and constant factors [10] instead of using $O(\cdot)$ which hides only constant factors.

The above generalization error bound suggests that AdaBoost will overfit if it runs for many rounds since T is in the numerator. However, empirical observations show that AdaBoost often does *not* overfit even after a large number of rounds, and sometimes it is even able to reduce the generalization error after the training error has already reached zero. Thus, later, Schapire et al. [11] presented another generalization error bound,

$$\epsilon \leq \Pr_{i \sim \mathcal{D}}\left[\text{margin}_f (x_i, y_i) \leq \theta \right] + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}} \right)$$

for any $\theta > 0$ with high probability, where the *margin* of f on (x_i, y_i) was defined as

$$\text{margin}_f(x_i, y_i) = \frac{y_i f(x_i)}{\sum_{t=1}^{T} |\alpha_t|} = \frac{y_i \sum_{i=1}^{T} \alpha_t h_t(x)}{\sum_{t=1}^{T} |\alpha_t|},$$

whose value is in $[-1, +1]$ and is positive only if H classifies (x_i, y_i) correctly. In fact, the magnitude of margin can be explained as a measure of confidence in prediction. The larger the magnitude of margin, the higher confidence of prediction. Note that when $H(x_i) = y_i$, $\text{margin}_f(x_i, y_i)$ can still be increased as t increases. Thus, the above margin-based generalization bound gives an answer to the question of why AdaBoost is able to reduce the generalization error even after the training error reaches zero, that is, the confidence in prediction can be increased further. However, Breiman [2] showed that improving the margin does not necessarily lead to the improvement of generalization, which doubted the above margin-based explanation to AdaBoost.

In addition to the previously mentioned studies, the behavior of AdaBoost has been explained from the views of game theory [2,3], additive model [5], etc. Many variants or extensions of AdaBoost have been developed [6,10], which makes Boosting become a big family of ensemble methods.

In contrast to another famous ensemble method, Bagging (which reduces *variance* significantly but has little effect on *bias*), Boosting can significantly reduce bias in addition to reducing variance. So, on weak learners such as decision stumps, which are one-level decision trees, Boosting is usually more effective.

Key Applications

The first application of Boosting was on Optical Character Recognization by Drucker et al. Later, Boosting was applied to diverse tasks such as text categorization, speech recognization, image retrieval, medical diagnosis, etc. [6,10]. It is worth mentioning that AdaBoost has been combined with a cascade process for face detection [12], and the resulting face detector was 15 times faster than state-of-the-art face detectors at that time but with comparable accuracy, which was recognized as one of the major breakthroughs in computer vision (in particular, face detection) during the past decade.

Future Directions

The margin-based explanation to why Boosting often does not overfit was seriously challenged by Breiman's indication that larger margin does not necessarily mean better generalization [2]. Recently, Reyzin and Schapire [8] found that Breiman considered minimum margin instead of average or median margin. If the margin-based explanation can survive, it may be possible to establish a unified theoretical framework for the two powerful learning approaches, i.e., Boosting and support vector machine, since it is well-known that support vector machine works by maximizing the margin in a feature space.

It has been observed that Boosting performs poorly when abundant noise exists. Making Boosting more robust to noise is an important task. Moreover, Boosting suffers from some general deficiencies of ensemble methods, such as the lack of comprehensibility, i.e., the knowledge learned by Boosting is not understandable to the user. Trying to overcome those deficiencies is an important future direction.

Experimental Results

Empirical studies on Boosting have been reported in many papers, such as [1,7].

Data Sets

A large collection of datasets commonly used for experiments can be found at http://www.ics.uci.edu/∼mlearn/MLRepository.html.

URL to Code

The code of an extended AdaBoost algorithm, BoosTexter, which was designed for multi-class text categorization, can be found at http://www.cs.princeton.edu/ schapire/boostexter.html.

Cross-references

▶ Bagging
▶ Decision Tree
▶ Ensemble
▶ Neural Networks
▶ Support Vector Machine

Recommended Reading

1. Bauer E. and Kohavi R. An empirical comparison of voting classification algorithms: Bagging, Boosting, and variants. Mach. Learn., 36(1–2):105–139, 1999.
2. Breiman L. Prediction games and arcing classifiers. Neural Comput., 11(7):1493–1517, 1999.
3. Freund Y. and Schapire R.E. Game theory, on-line prediction and Boosting. In Proc. Ninth Annual Conf. on Computational Learning Theory, 1996, pp. 325–332.
4. Freund Y. and Schapire R.E. A decision-theoretic generalization of on-line learning and an application to Boosting. J. Comput. Syst. Sci., 55(1):119–139, 1997. (A short version appeared in the Proceedings of EuroCOLT'95).
5. Friedman J., Hastie T., and Tibshirani R. Additive logistic regression: A statistical view of Boosting with discussions. Ann. Stat., 28(2):337–407, 2000.
6. Meir R. and Rätsch G. An introduction to Boosting and leveraging. In Advanced Lectures in Machine Learning, S. Mendelson and A.J. Smola (eds.). LNCS, Vol. 2600, Springer, Berlin, 2003, pp. 118–183.
7. Opitz D. and Maclin R. Popular ensemble methods: An empirical study. J. Artif. Intell. Res., 11:169–198, 1999.
8. Reyzin L. and Schapire R.E. How boosting the margin can also boost classifier complexity. In Proc. 23rd Int. Conf. on Machine Learning, 2006, pp. 753–760.
9. Schapire R.E. The strength of weak learnability. Mach. Learn., 5(2):197–227, 1990.
10. Schapire R.E. The Boosting approach to machine learning: An overview. In Nonlinear Estimation and Classification. D.D. Denison, M.H. Hansen, C. Holmes, B. Mallick, and B. Yu (eds.). Springer, Berlin, 2003.
11. Schapire R.E., Freund Y., Bartlett P., and Lee W.S. Boosting the margin: A new explanation for the effectiveness of voting methods. Ann. Stat., 26(5):1651–1686, 1998.

12. Viola P. and Jones M. Rapid object detection using a boosted cascade of simple features. In Proc. IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition, 2001, pp. 511–518.

Bootstrap

Hwanjo Yu
University of Iowa, Iowa City, IA, USA

Synonyms

Bootstrap sampling; Bootstrap estimation

Definition

The bootstrap is a statistical method for estimating the performance (e.g., accuracy) of classification or regression methods. The bootstrap is based on the statistical procedure of sampling *with replacement*. Unlike other estimation methods such as cross-validation, the same object or tuple can be selected for the training set more than once in the boostrap. That is, each time a tuple is selected, it is equally likely to be selected again and re-added to the training set.

Historical Background

The bootstrap sampling was developed by Bradley Efron in 1979, and mainly used for estimating the statistical parameters such as mean, standard errors, etc. [2]. A meta-classification method using the bootstrap called *bootstrap aggregating* (or *bagging*) was proposed by Leo Breiman in 1994 to improve the classification by combining classifications of randomly generated training sets [1].

Foundations

This section discusses a commonly used bootstrap method, *.632 bootstrap*. Given a dataset of N tuples, the dataset is sampled N times, with replacement, resulting in a *bootstrap sample* or training set of N tuples. It is very likely that some of the original data tuples will occur more than once in the training set. The data tuples that were not sampled into the training set end up forming the test set. If this process is repeated multiple times, on average 63.2% of the original data tuples will end up in the training set and the remaining 36.8% will form the test set (hence, the name, .632 bootstrap).

The figure, 63.2%, comes from the fact that a tuple will not be chosen with probability of 36.8%. Each tuple has a probability of $1/N$ of being selected, so the probability of not being chosen is $(1 - 1/N)$. The selection is done N times, so the probability that a tuple will not be chosen during the whole time is $(1 - 1/N)^N$. If N is large, the probability approaches $e^{-1} = 0.368$. Thus, 36.8% of tuples will not be selected for training and thereby end up in the test set, and the remaining 63.2% will form the training set.

The above procedure can be repeated k times, where in each iteration, the current test set is used to obtain an accuracy estimate of the model obtained from the current bootstrap sample. The overall accuracy of the model is then estimated as

$$Acc(M) = \sum_{i=1}^{k} (0.632 \times Acc(M_i)_{test\_set} + 0.368 \times Acc(M_i)_{train\_set}), \tag{1}$$

where $Acc(M_i)_{train\_set}$ and $Acc(M_i)_{test\_set}$ are the accuracy of the model obtained with bootstrap sample i when it is applied to training set and test set respectively in sample i.

Key Applications

The bootstrap method is preferably used for estimating the performance when the size of dataset is relatively small.

Cross-references

- ▶ Cross-Validation
- ▶ Holdout Test
- ▶ Sampling
- ▶ Validation

Recommended Reading

1. Breiman L. Bagging predictors. Machine Learning, 1996.
2. Efron B. and Tibshirani R.J. An Introduction to the Bootstrap. CRC Press, Boca Raton, 1994.

Bootstrap Aggregating

▶ Bagging

Bootstrap Estimation

▶ Bootstrap

Bootstrap Sampling

▶ Bootstrap

Bottom-up Semantics

▶ Emergent Semantics

Boyce-Codd Normal Form

Marcelo Arenas
Pontifical Catholic University of Chile, Santiago, Chile

Synonyms

BCNF

Definition

Let $R(A_1,\ldots,A_n)$ be a relation schema and Σ a set of functional dependencies over $R(A_1,\ldots,A_n)$. Then (R, Σ) is said to be in Boyce-Codd Normal Form (BCNF) if for every nontrivial functional dependency $X \rightarrow A$ implied by Σ, it holds that X is a superkey for R.

Key Points

In order to avoid update anomalies in database schemas containing functional dependencies, BCNF was introduced by Codd (In [2], Codd pointed out that this normal form was developed by Raymond F. Boyce and himself.) in [2]. This normal form is defined in terms of the notion of superkey as shown above. For example, given a relation schema $R(A, B, C)$ and a set of functional dependencies $\Sigma = \{AB \rightarrow C, C \rightarrow B\}$, it does not hold that $(R(A, B, C), \Sigma)$ is in BCNF since C is not a superkey for R. On the other hand, $(S\ (A, B, C), \Gamma)$ is in BCNF if $\Gamma = \{A \rightarrow BC\}$, since A is a superkey for S in this case.

It should be noticed that relation schema $R(A, B, C)$ above is in 3NF if $\Sigma = \{AB \rightarrow C, C \rightarrow B\}$, although this schema is not in BCNF. In fact, BCNF is strictly stronger than 3NF; every schema in BCNF is in 3NF, but there exist schemas (as the one shown above) that are in 3NF but not in BCNF.

For every normal form two problems have to be addressed how to decide whether a schema is in that normal form, and how to transform a schema into an equivalent one in that normal form. As opposed to the case of 3NF, it can be tested efficiently whether a relation schema is in BCNF. A relation schema (R, Σ) is in BCNF if and only if for every nontrivial functional dependency $X \rightarrow Y \in \Sigma$, it holds that X is a superkey. Thus, it is possible to check efficiently whether (R, Σ) is in BCNF by using the linear-time algorithm for functional dependency implication developed by Beeri and Bernstein [1]. On the negative side, given a relation schema S, it is not always possible to find a database schema S' such that S' is in BCNF and S' is a lossless and dependency preserving decomposition of S. In fact, relation schema $R(A, B, C)$ and set of functional dependencies $\{AB \rightarrow C, C \rightarrow B\}$ does not admit a dependency preserving decomposition in BCNF.

Cross-references

▶ Fourth Normal Form (4NF)
▶ Normal Forms and Normalization
▶ Second Normal Form (2NF)
▶ Third Normal Form (3NF)

Recommended Reading

1. Beeri C. and Bernstein P. Computational Problems Related to the Design of Normal Form Relational Schemas. ACM Trans. Database Sys., 4(1):30–59, 1979.
2. Codd E.F. Recent Investigations in Relational Data Base Systems. In Proc. IFIP Congress. 1974, pp. 1017–1021.

BP-Completeness

Dirk Van Gucht
Indiana University, Bloomington, IN, USA

Synonyms

Instance-completeness; Relation-completeness

Definition

A relational query language Q is BP-complete if for each relational database D, the set of all relations defined by the queries of Q on D is equal to the set of all first-order definable relations over D. More formally, fix some infinite universe **U** of atomic data elements. A *relational database schema* S is a finite set of relation names, each with an associated arity. A *relational database* D with schema S assigns to each relation name of S a *finite* relation over **U** of its arity. The domain of D, $dom(D)$, is the set of all atomic data elements occurring in the tuples of its relations. Let FO^S be the set of first-order formulas over signature S and the equality predicate, and let $FO^S(D) = \{\varphi(D) | \varphi \in FO^S\}$. (For a formula $\varphi \in FO^S$ with free variables $(x_1,...,x_m)$, $\varphi(D)$ denotes the m-ary relation over $dom(D)$ defined by φ, where the variables in φ are assumed to range over $dom(D)$.) Let Q^S denote those queries of Q defined over schema S, and let $Q^S(D) = \{q(D)|q \in Q^S\}$, i.e., the set of relations defined by queries of Q applied to D. Then, Q is *BP- complete* if for each relational database D over schema S,

$$Q^S(D) = FO^S(D).$$

In the words of Chandra and Harel, "BP-completeness can be seen to be a measure of the power of a language to express relations and *not* of its power to express functions having relations as outputs, i.e., queries." In fact, there exist BP-complete languages that do not express the same queries.

Key Points

Chandra and Harel introduced the concept of BP-completeness and attributed it to Bancilhon and Paredaens who were the first to study it. Bancilhon and Paredaens considered the following decision problem: given a relational database D and a relation R defined over $dom(D)$, does there exists a first-order formula φ such that $\varphi(D) = R$? (Paredaens considered this problem for the relational algebra, but by Codd's theorem on the equivalence of first-order logic and the relational algebra, these decision problems are the same.) They gave an algebraic, language-independent characterization of this problem by showing that such a first-order formula exists if and only if for each bijection $h : dom(D) \rightarrow dom(D)$, if $h(D) = D$ then $h(R) = R$. Equivalently, if h is an automorphism of D then it is also an automorphism of R. (Here, $h(D)$ and $h(R)$ are the natural extensions of h to D and R, respectively.) For a relational database D over schema S and a relation R, denote by $Aut(D)$ and $Aut(R)$ the sets of automorphisms of D and R, respectively, and let $R^S(D) = \{R \mid R$ is a relation over $dom(D)$ such that $Aut(D) \subseteq Aut(R)\}$. Then, an alternative characterization for the BP-completeness of Q is to require that for each relational database D over schema S,

$$Q^S(D) = R^S(D).$$

Van den Bussche showed that this characterization follows from Beth's Theorem about the explicit and implicit definability of first-order logic. The concept of BP-completeness has been generalized as well as specialized to query languages over other database models.

Cross-references

▶ Complete Query Languages
▶ Query Language
▶ Relational Calculus and Algebra

Recommended Reading

1. Bancilhon F. On the completeness of query languages for relational databases. In Proc. Mathematical Foundations of Computer Science, 1978.
2. Chandra A. and Harel D. Computable Queries for relational databases. J. Comput. Syst. Sci., 25:99–128, 1982.
3. Paredaens J. On the expressive power of the relational algebra. Inform. Process. Lett., 7(2):107–111, 1978.
4. Van den Bussche. J. Applications of Alfred Tarski's Ideas in Database Theory. In Computer Science Logic, Lect. Notes Comput. Sci., 2142:20–37, 2001.

BPEL

▶ Business Process Execution Language

BPEL4WS

▶ Business Process Execution Language

BPMN

▶ Business Process Modeling Notation

Bpref

Nick Craswell
Microsoft Research Cambridge, Cambridge, UK

Definition

Bpref is a preference-based information retrieval measure that considers whether relevant documents are ranked above irrelevant ones. It is designed to be robust to missing relevance judgments, such that it gives the same experimental outcome with incomplete judgments that Mean Average Precision would with complete judgments.

Key Points

In a test collection where all relevant documents have been identified, experiments using bpref and MAP should give the same outcome, for example both systems should agree that system A is better than system B. However, if the relevance judgments are incomplete, for example where only half the pool has been judged, MAP becomes unstable and may incorrectly show that system B is better than system A. The bpref measure was developed to maintain the correct ordering of systems (A better than B) even with incomplete judgments.

Given a ranked list of search results and a set of R known relevant documents and N known irrelevant documents, bpref first identifies the top-R irrelevant documents in the list. For these irrelevant documents n and relevant documents r the measure is:

$$bpref = \frac{1}{R}\sum_{r}\left(1 - \frac{|n\ ranked\ higher\ than\ r|}{\min(R,N)}\right)$$

The bpref paper [1] found good agreement between full-judgment MAP and reduced-judgment bpref even if the judged set was reduced by 80%. Later work has introduced other measures with similar or better properties, notably Inferred Average Precision [2].

Cross-references

▶ Mean Average Precision
▶ Precision-Oriented Effectiveness Measures

Recommended Reading

1. Buckley C. and Voorhees E.M. Retrieval evaluation with incomplete information. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 25–32.
2. Yilmaz E. and Aslam J.A. Estimating average precision with incomplete and imperfect judgments. In Proc. Int. Conf. on Information and Knowledge Management, 2006, pp. 102–111.

Branch

▶ OR-Split

Bridging

▶ Mediation

Browsing

Kent Wittenburg
Mitsubishi Electric Research Laboratories, Inc., Cambridge, MA, USA

Synonyms

Perusal; Scanning

Definition

"Browsing" has two definitions in the context of visual interfaces for database systems:

1. The human activity of visual perception and interpretation of electronic content when there is no specific target object being sought.
2. The human activity of clicking or tapping on a sequence of elements in an information display that results in a sequence of screens of information state.

Definition (1) implies something about the mental intent of the user performing the act of information processing. The intent can be to learn the gist of "what is there." It may be to take in information in order to be entertained or informed. Or it may simply be a part of involuntary visual scanning within an environment. In the physical world, examples include flipping through the pages of a book, scanning quickly through a menu, or glancing down a store aisle.

Definition (2) is used in the context of contrasting the human-computer interaction paradigm of link-following in World Wide Web applications versus searching, which entails forming queries. "Browsing" in this context refers to a sequence of clicking or tapping behaviors on highlighted elements (hyperlinks) in order to go directly to a subsequent state of the information display. In contrast, "searching" entails specifying and then executing a query. The interaction paradigms of browsing and searching are independent of the intentional state of the user – the individual may or may not be looking for something specific in either case.

Key Points

A branch of research in visual interfaces for browsing in the first sense is Rapid Serial Visual Presentation (RSVP) [2, Sect. 4.2]. The human psychology of visual perception, specifically models of short-term visual memory, can inform the design of visual interfaces for browsing. Visual information can be presented in space or in time or in some combination thereof. For image presentation specifically, is it better to utilize screen real estate to present multiple images concurrently or to present them over time? If over time, should images move in some specified path or remain motionless? What are the best controls to give to the user in order to support the tasks of rapid perusal in order to get a gist of the content within that information space or to help in making a selection for further detailed investigation? These are some of the questions being asked by researchers and designers in the context of RSVP interfaces.

An important aspect of browsing as link-following has to do with information "scent." Just as animals' scent can be used as an indicator of their territory or former presence, the presentation of information links provides hints of what a user would find were he or she to follow the link. The theory of information foraging [1] provides insight into human behavior in information seeking that can help inform the design for interfaces and electronic documents. According to Pirolli, humans tend to make decisions about what links to follow and how long to continue along certain paths based on resource-limited assessments of the costs of such activities relative to a judgment of payoff.

Cross-references

- ▶ Browsing in Digital Libraries
- ▶ Discovery
- ▶ Human-Computer Interaction
- ▶ Information Foraging
- ▶ Information Navigation
- ▶ Information Retrieval
- ▶ Searching Digital Libraries
- ▶ Visual Interaction
- ▶ Visual Interfaces
- ▶ Web Information Retrieval Models

Recommended Reading

1. Pirolli P. Information Foraging Theory: Adaptive Interaction with Information. Oxford University Press, New York, NY, 2007.
2. Spence R. Information Visualization: Design for Interaction (2nd edn.). Pearson/Prentice Hall, Upper Saddle River, NJ, 2007.

Browsing in Digital Libraries

RAO SHEN[1], EDWARD A. FOX[2]
[1]Yahoo!, Sunnyvale, CA, USA
[2]Virginia Tech, Blacksburg, VA, USA

Synonyms

Exploring; Surfing; Looking over/through

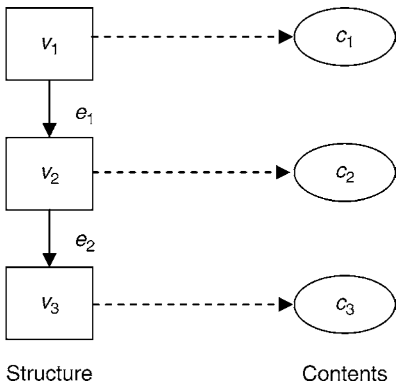
Definition

Informally, browsing is a process that involves looking through a collection of information. Thus, in a traditional library, one may wander about the stacks, glancing at titles of works, in regions where one expects to find interesting material. In the broadest sense, browsing is considered as one type of exploration, typically less directed or purposeful than searching, in that the goal or result is not always precisely known in advance. In the World Wide Web, this is very common, making use of "browsers" like Firefox or Internet Explorer. Sometimes the colorful term "surfing" is used when it seems appropriate to emphasize the excitement that some feel when exploring new Web content, or the thrill of getting closer to some desirable result. Yet, the WWW is but one example of a hypertext (or, if multiple media types are considered, hypermedia) environment. Accordingly, in a digital library, which many believe must include a hypertext [4], browsing involves moving from one information object to another according to some connections or links or structures.

Historical Background

A hypertext is a high level interactive navigational structure which allows non-sequential exploration of and access to information [3]. In the most general case, it consists of nodes connected by directed links in a graph structure. A directed link in the hypertext graph is called a hyperlink. Today's most famous hypertext system is the World Wide Web. The Web's structure contains hyperlinks connecting nodes that each can be associated with either a complete document (e.g., HTML page) or with a location in a document (e.g., a start tag in an HTML page).

Three models for Web browsing defined by Baeza-Yates and Ribeiro-Neto [1] are flat browsing, structure guided browsing, and utilizing the hypertext model. If a document collection is organized as a one dimensional list, exploring the document space is an example of flat browsing; if the organization is hierarchical instead of flat, then browsing is structure guided. The first two models have a hypertext graph that is made up of a disconnected bipartite graph and a tree, respectively, so they are special cases of the third model.



Browsing in Digital Libraries. Figure 1. A simple hypertext.

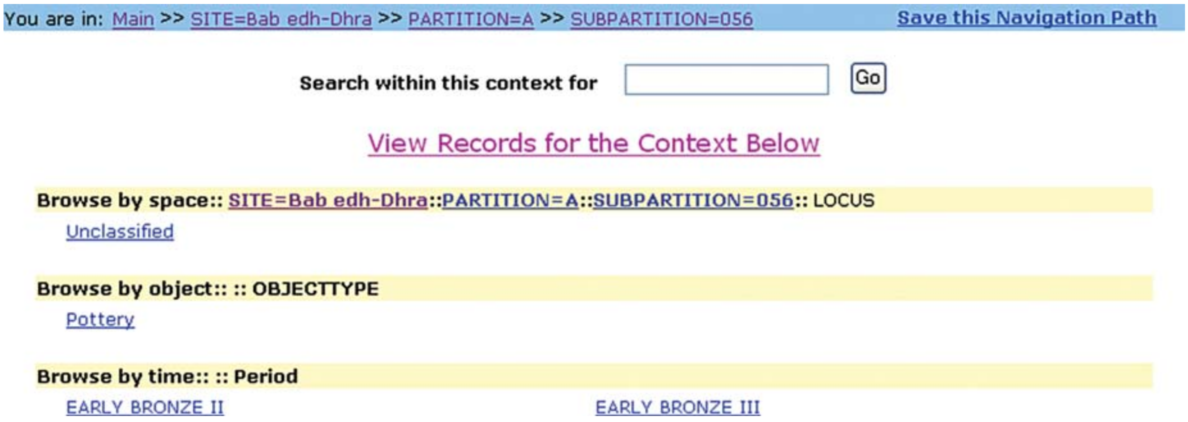
Browsing and searching are two paradigms for finding things on the Web. Some Web search engines provide Web directories used for browsing (and also for searching). Web directories are hierarchical taxonomies that classify human knowledge. Although a taxonomy can be considered as a tree, there can be cross references. One of the advantages of browsing a Web directory, in most cases, is that if a user finds what she is looking for in the taxonomy, then the answer that is found classified under that category almost certainly will be useful. In Web directories, a search can be reduced to a sub-tree of the taxonomy. However, searching may miss related pages that are not in that part of the taxonomy, if a user cannot formulate her information need sufficiently broadly.

Foundations

Browsing and searching are two methods of pursuing information on the Web; they also are two major paradigms for exploring digital libraries. The Web has no maintenance organization. Individuals add and delete pages at will. On the other hand, digital libraries require proper collection maintenance, and they will succeed only if their content is well organized [2].

The 5S (Streams, Structures, Scenarios, Spaces, and Societies) framework [4], which is used as a formal base upon which to describe digital libraries, defines structures to specify the way in which parts of a whole are managed or organized. In such digital libraries, structures can represent hypertexts.

For example, Fig. 1 illustrates a simple hypertext designed to provide access to objects in a digital library, in chronological order. It is made up of structural hyperlinks that follow chronological order (see solid



You are in: Main >> SITE=Bab edh-Dhra >> PARTITION=A >> SUBPARTITION=056 Save this Navigation Path

Search within this context for [ ] Go

View Records for the Context Below

Browse by space:: SITE=Bab edh-Dhra::PARTITION=A::SUBPARTITION=056:: LOCUS

Unclassified

Browse by object:: :: OBJECTTYPE

Pottery

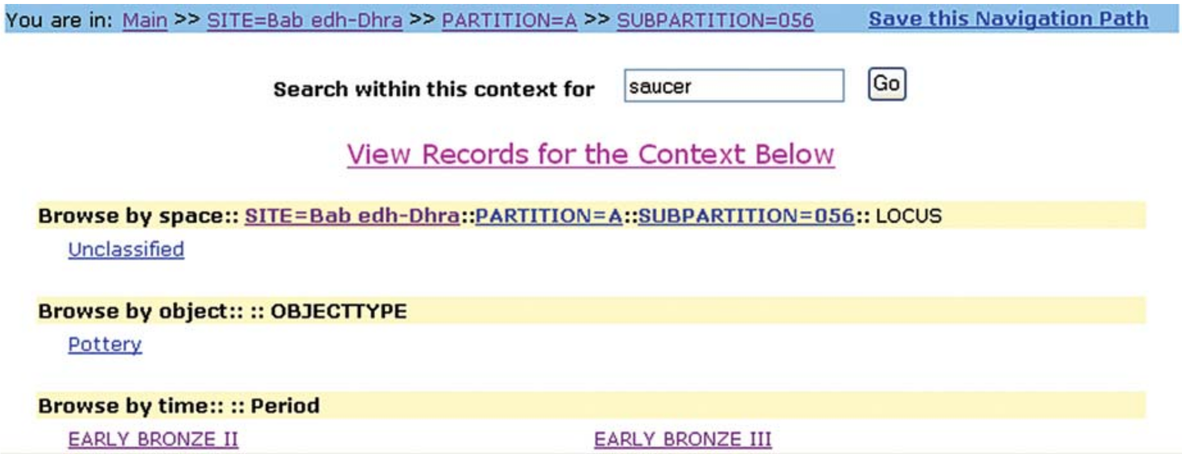
Browse by time:: :: Period

EARLY BRONZE II EARLY BRONZE III

Browsing in Digital Libraries. Figure 2. Multi-dimensional browsing.

arrows in Fig. 1) and external reference links (see dashed arrows in Fig. 1).

Thus, a formal and precise definition of browsing is illustrated in Fig. 1: Given a hypertext of a digital library, with vertex set *V* and edge set *E*, browsing is a set of sequences of traverse link events over the hypertext, such that event *e* of traversing from node v_k to v_t is associated with a function which retrieves the contents of node v_t. Hence, browsing a digital library is the


You are in: Main >> SITE=Bab edh-Dhra >> PARTITION=A >> SUBPARTITION=056 Save this Navigation Path

Search within this context for saucer Go

View Records for the Context Below

Browse by space:: SITE=Bab edh-Dhra::PARTITION=A::SUBPARTITION=056:: LOCUS

Unclassified

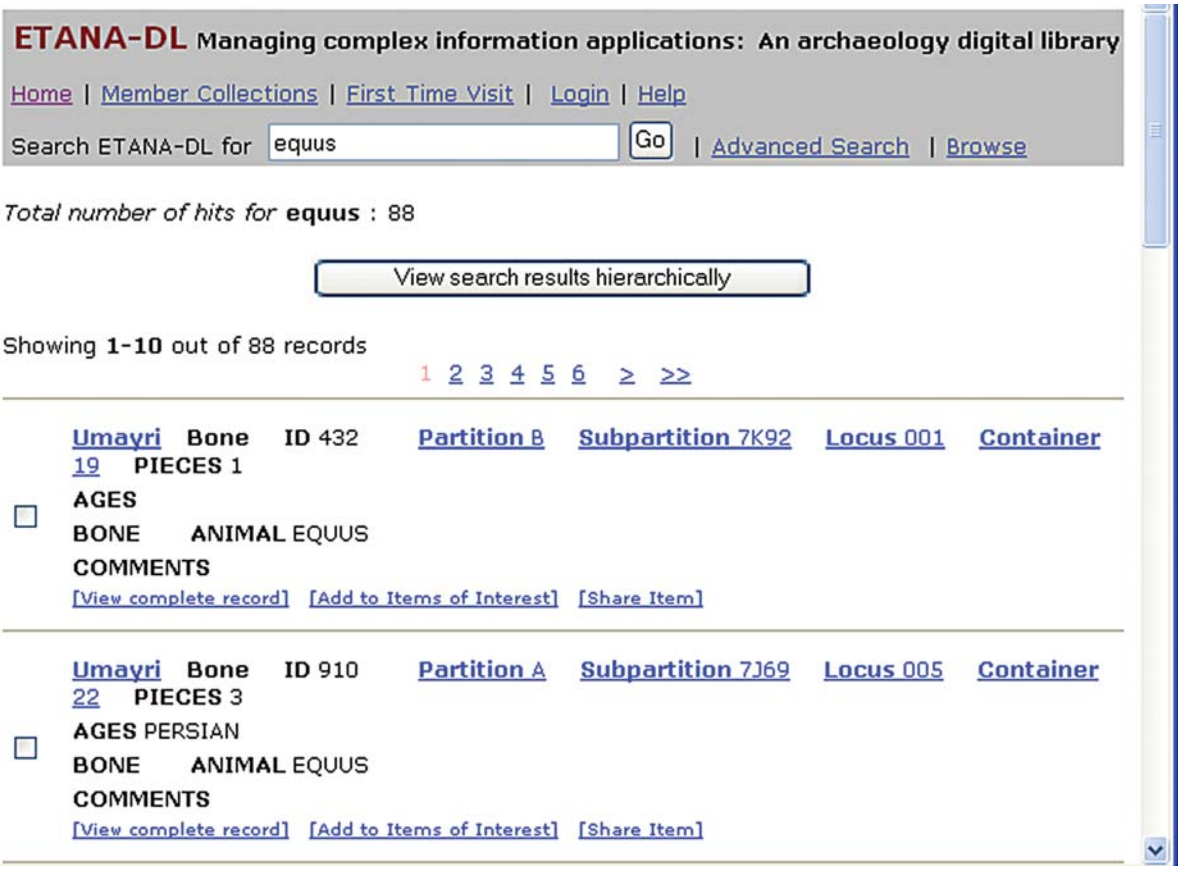
Browse by object:: :: OBJECTTYPE

Pottery

Browse by time:: :: Period

EARLY BRONZE II EARLY BRONZE III

Browsing in Digital Libraries. Figure 3. Search saucer records.


ETANA-DL Managing complex information applications: An archaeology digital library

Home | Member Collections | First Time Visit | Login | Help

Search ETANA-DL for equus Go | Advanced Search | Browse

Total number of hits for equus : 88

View search results hierarchically

Showing 1-10 out of 88 records

1 2 3 4 5 6 > >>

□ Umayri Bone ID 432 Partition B Subpartition 7K92 Locus 001 Container 19 PIECES 1
AGES
BONE ANIMAL EQUUS
COMMENTS
[View complete record] [Add to Items of Interest] [Share Item]

□ Umayri Bone ID 910 Partition A Subpartition 7J69 Locus 005 Container 22 PIECES 3
AGES PERSIAN
BONE ANIMAL EQUUS
COMMENTS
[View complete record] [Add to Items of Interest] [Share Item]

Browsing in Digital Libraries. Figure 4. Equus records are retrieved through basic searching.

procedure of navigating the hypertext, and it also can be understood as a traversal of a directed linked graph.

Returning to the previous example, one can observe that the left part of Fig. 1 is a directed linked graph (V, E), where each the right part is a set of contents $C = \{c_1, c_2, c_3\}$, and where c_i $(1 <= i <= 3)$ is the contents of a Web page associated with node v_i. An event of traversing along edge $e_1 = (v_1, v_2)$ is associated with a function: $E \rightarrow C$, which retrieves the contents of node v_2, i.e., c_2.

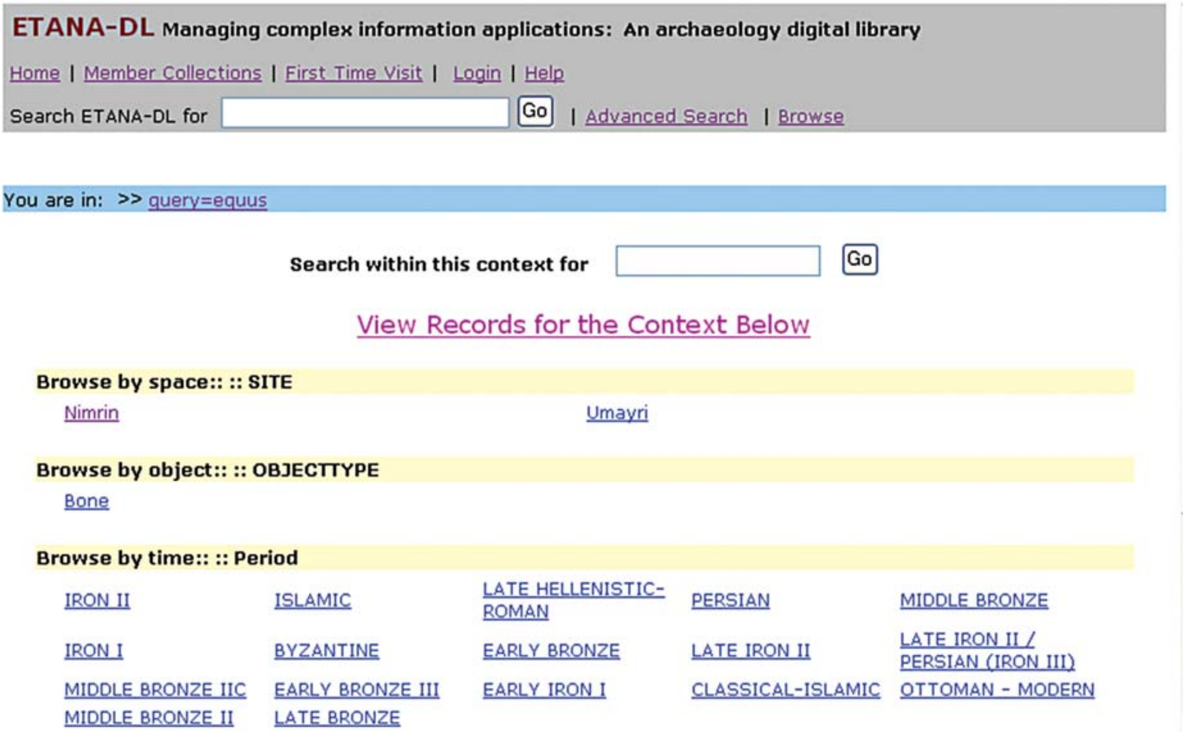
Note that (sorted) lists and hierarchical trees can be represented as hypertexts, so browsing can be over common organizations, such as alphabetical lists of authors, or indented lists of categories and subcategories; these are familiar from books with author indexes or tables of contents.

Key Applications

Modeling and analyzing browsing in digital libraries helps ease development and maintenance in those digital libraries. Some domain specific digital libraries have heterogeneous data that should be organized using several schemes. Thus, digital objects in an archaeological digital library may fit into various categories of archaeological data such as figurine images, bone records, locus sheets, and site plans. They can be organized according to different hierarchical structures (e.g., animal bone records are organized based on sites where they are excavated, temporal sequence, and animal names). These hierarchical structures contain one or more hierarchically arranged categories. In addition, they can be refined based on taxonomies existing in botany and zoology, or through classification and description of artifacts by archaeologists.

Thus, an archaeological digital library may provide multi-dimensional browsing (see Fig. 2) to allow users to move along any of the navigational dimensions, or a combination thereof. Navigational dimensions correspond to hierarchical structures used to browse digital objects, as mentioned above.

For example, an archaeologist might browse through three dimensions: space, object, and time. She can start from any of these dimensions and move along by clicking. The scenario shown in Fig. 2 tells that she is interested in the artifact records from the tomb numbered 056 in area A of the Bab edh-Dhra site. The clickstream representing her navigation path is


ETANA-DL Managing complex information applications: An archaeology digital library
Home | Member Collections | First Time Visit | Login | Help
Search ETANA-DL for
Go
| Advanced Search | Browse
You are in: >> query=equus
Search within this context for
Go
View Records for the Context Below

Browse by space:: :: SITE

Nimrin Umayri

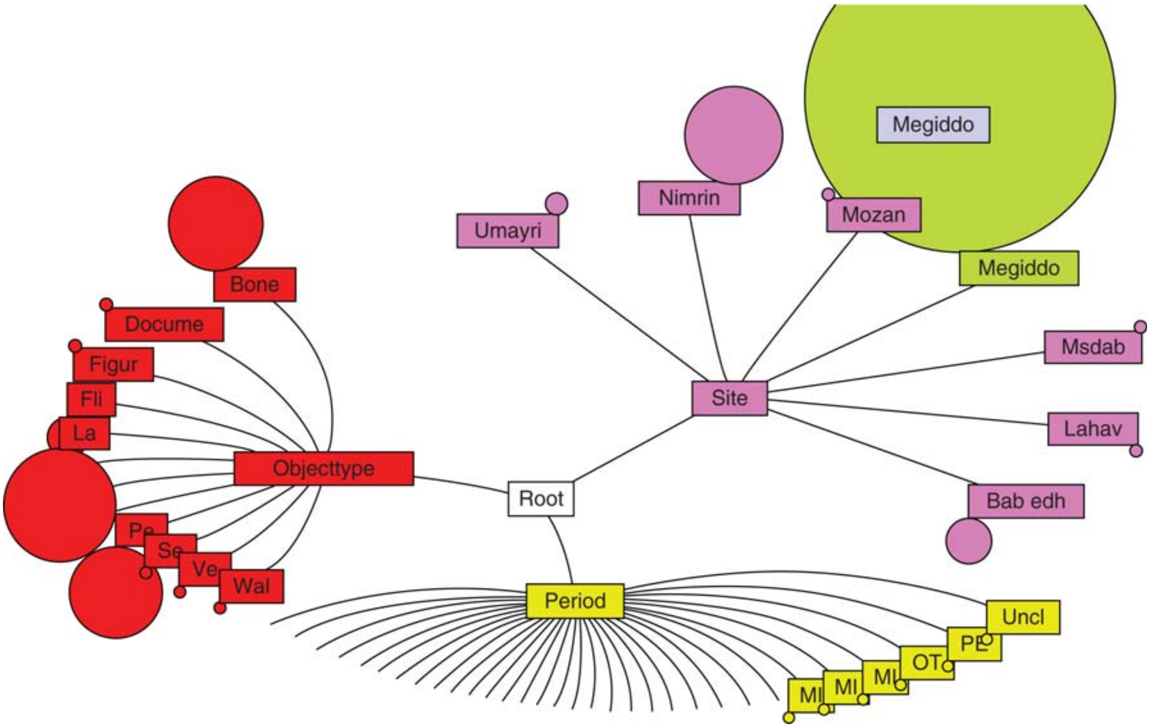
Browse by object:: :: OBJECTTYPE

Bone

Browse by time:: :: Period

IRON II	ISLAMIC	LATE HELLENISTIC-ROMAN	PERSIAN	MIDDLE BRONZE
IRON I	BYZANTINE	EARLY BRONZE	LATE IRON II	LATE IRON II / PERSIAN (IRON III)
MIDDLE BRONZE IIC	EARLY BRONZE III	EARLY IRON I	CLASSICAL-ISLAMIC	OTTOMAN - MODERN
MIDDLE BRONZE II	LATE BRONZE			

Browsing in Digital Libraries. Figure 5. Retrieved equus records are organized into 3 dimensions.



Browsing in Digital Libraries. Figure 6. Overview of an archaeological digital library.

denoted "Site=Bab edh-Dhra>>PARTITION=A>> SUBPARTITION=056." While this navigation path is within the first dimension, it also is associated with the other dimensions. The second dimension shows there is only one type of object, i.e., pottery, from that particular location. The third dimension presents the two time periods associated with those pottery records. Hence, the dynamic coverage and hierarchical structure of those dimensions yields a tool supporting learning and exploring. The user can navigate across dimensions. By clicking "EARLY BRONZE II" in the third dimension, she can view all of the interesting artifact records from the EARLY BRONZE II period.

Browsing may present a useful starting point for active exploration of an answer space. Subsequent browsing and searching, in any combination, also can be employed to refine or enhance users' initial, possibly under-specified, information needs.

Browsing context is associated with a user's navigation path. Browsing results within a certain browsing context typically are a set of records (web pages), e.g., there are 35 pottery records within the browsing context represented by the navigation path denoted "Site=Bab edh-Dhra>>PARTITION=A>>SUBPARTITION=056." For example, assume a user wants to find saucer records in the set of thirty five pottery records. She types "saucer" in the search box as shown in Fig. 3. She switches from browsing to searching, so searching then is a natural extension of browsing.

Browsing may be provided as a post-retrieval service to organize searching results hierarchically in digital libraries. For example, in Fig. 4 one can see that eighty eight equus records are retrieved through the basic searching service, in response to a query "equus". They are organized into three dimensions after the user clicks the button "View search results hierarchically" (see Fig. 5).

Browsing may be supported by visualization to provide a starting point for users. Graphic overviews of a digital library collection can display category labels hierarchically based on the facets. Categories can be visualized as a hyperbolic tree [5] as well as through a traditional node-link representation of a tree.

A hyperbolic tree in Fig. 6 shows hierarchical relationships among excavation data in an archaeological digital library based on spatial, temporal, and artifact-related taxonomies. A node name represents a category, and a bubble attached to a node represents a set of archaeological records. The size of a bubble attached to a node reflects the number of records belonging to that category. The hyperbolic tree supports "focus + context" navigation; it also provides an overview of records organized in the archaeological digital library. It shows that the records are from seven archaeological sites (the Megiddo site has the most) and are of twelve different types.

Future Directions

Browsing and searching are often provided by digital libraries as separate services. Developers commonly see these functions as having different underlying mechanisms, and they follow a functional, rather than a task-oriented, approach to interaction design. While exhibiting complementary advantages, neither paradigm alone is adequate for complex information needs. Searching is popular because of its ability to identify information quickly. On the other hand, browsing is useful when appropriate search keywords are unknown or unavailable to users. Browsing also is appropriate when a great deal of contextual information is obtained along the navigation path. Therefore, a synergy between searching and browsing is required to support users' information seeking goals. Browsing and searching can be converted and switched to each other under certain conditions [5]. This suggests some new possibilities for blurring the dividing line between browsing and searching. If these two services are not considered to have different underlying mechanisms, they will not be provided as separated functions in digital libraries, and may be better integrated.

Text mining and visualization techniques provide digital libraries additional powerful exploring services, with possible beneficial effects on browsing and searching. Digital library exploring services such as browsing, searching, clustering, and visualization can be generalized in the context of a formal digital library framework [5]. The theoretical approach may provide a systematic and functional method to design and implement exploring services for domain focused digital libraries.

Experimental Results

See Fig. 2 – Fig. 6 above, and the corresponding explanation.

Data Sets

See Fig. 2 – Fig. 6 above, and the ETANA Digital Library [5].

Cross-references

▶ Digital Libraries

Recommended Reading

1. Baeza-Yates R. and Ribeiro-Neto B. Modern Information Retrieval. Addison-Wesley, Reading, MA, 1999.
2. Fox E.A. and Urs S.R. Digital libraries, chap. 12. In Annual Review of Information Science and Technology, Vol. 36, B. Cronin (ed.); Medford, NJ, Information Today, Inc. 2002, pp. 503–589.
3. Fox E.A., Rous B., and Marchionini G. ACM's hypertext and hypermedia publishing projects. In Hypertext/Hypermedia Handbook, E. Berk, J. Devlin (eds.). McGraw-Hill, NY, 1991, pp. 465–467.
4. Goncalves M., Fox E.A., Watson L., and Kipp N. Streams, structures, spaces, scenarios, societies (5S): a formal model for digital libraries. ACM Trans. Inf. Syst., 22(2):270–312, 2004.
5. Shen R., Vemuri N., Fan W., Torres R., and Fox E.A. Exploring digital libraries: integrating browsing, searching, and visualization. In Proc. 6th ACM/IEEE-CS joint Conference on Digital Libraries, 2006, pp. 1–10.

B-Tree

▶ B+-Tree

B-Tree Concurrency Control

▶ B-Tree Locking

B-Tree Locking

GOETZ GRAEFE
Hewlett-Packard Laboratories, Palo Alto, CA, USA

Synonyms

B-tree concurrency control; Row-level locking; Key value locking; Key range locking; Lock coupling; Latching; Latch coupling; Crabbing

Definition

B-tree locking controls concurrent searches and updates in B-trees. It separates transactions in order to protect the B-tree contents and it separates threads in order to protect the B-tree data structure. Nowadays, the latter is usually called latching rather than locking.

Historical Background

Bayer and Schkolnick [1] presented multiple locking (latching) protocols for B^*-trees (all data records in the leaves, merely separator keys or "reference keys" in upper nodes) that combined high concurrency with deadlock avoidance. Their approach for insertion and deletion is based on deciding during a root-to-leaf traversal whether a node is "safe" from splitting (during an insertion) or merging (during a deletion), and on reserving appropriate locks (latches) for ancestors of unsafe nodes.

Lehman and Yao defined B^{link}-trees by relaxing the B-tree structure in favor of higher concurrency [8]. Srinivasan and Carey demonstrated their high performance using detailed simulations [13]. Jaluta et al. recently presented a detailed design for latching in B^{link}-trees, including a technique to avoid excessive link chains and thus poor search performance [7].

IBM's System R project explored multiple transaction management techniques, including transaction isolation levels and lock duration, predicate locking and key value locking, multi-granularity and hierarchical locking, etc. These techniques have been adapted and refined in many research and product efforts since then. Research into multi-level transactions [14] and into open nested transactions [3,12] enables crisp separation of locks and latches – the former protecting database contents against conflicts among transactions and the latter protecting in-memory data structures against conflicts among concurrent threads.

Mohan's ARIES/KVL design [10,11] explicitly separates locks and latches, i.e., logical database contents versus "structure maintenance" in a B-tree. A key value lock covers both a gap between two B-tree keys and the upper boundary key. In non-unique indexes, an intention lock on a key value permits operations on separate rows with the same value in the indexed column. In contrast, other designs include the row identifier in the unique lock identifier and thus do not need to distinguish between unique and non-unique indexes.

Lomet's design for key range locking [4] attempts to adapt hierarchical and multi-granularity locking to keys and half-open intervals but requires additional lock modes, e.g., a "range insert" mode, to achieve the desired concurrency. Graefe's design [9] applies traditional hierarchical locking to keys and gaps (open intervals) between keys, employs ghost (pseudo-deleted) records during insertion as well as during deletion, and permits more concurrency with fewer special cases. The same paper also outlines hierarchical locking exploiting B-trees' hierarchical structure or multi-field B-tree keys.

Foundations

The foundations of B-tree locking are the well-known transaction concepts, including multi-level transactions and open nested transactions, and pessimistic concurrency control, i.e., locking. Multiple locking concepts and techniques are employed, including two-phase locking, phantom protection, predicate locks, precision locks, key value locking, key range locking, multi-granularity locking, hierarchical locking, and intention locks.

Preliminaries

Most work on concurrency control and recovery in B-trees assumes what Bayer and Schkolnick call B^*-trees [1] and what Comer calls B^+-trees [2], i.e., all data records are in leaf nodes and keys in non-leaf or "interior" nodes act merely as separators enabling search and other operations but not carrying logical database contents. Following this tradition, this entry ignores the original design of B-trees with data records in interior nodes.

Also ignored are many other variations of B-trees here. This includes what Comer, following Knuth, calls B^*-trees, i.e., attempting to merge an overflowing node with a sibling rather than splitting it immediately. Among the ignored techniques are whether or not underflow is recognized and acted upon by load balancing and merging nodes, whether or not empty nodes are removed immediately or ever, whether or not leaf nodes form a singly or doubly linked list using physical pointers (page identifiers) or logical boundaries (fence keys equal to separators posted in the parent node during a split), whether suffix truncation is employed when posting a separator key, whether prefix truncation or any other compression is employed on each page, and the type of information associated with B-tree keys. Most of these issues have little or no bearing on locking in B-trees, with the exception of sibling pointers, as indicated below where appropriate.

Two Forms of B-Tree Locking

B-tree locking, or locking in B-tree indexes, means two things. First, it means concurrency control among concurrent database transactions querying or modifying database contents and its representation in B-tree indexes. Second, it means concurrency control among concurrent threads modifying the B-tree data structure in memory, including in particular images of disk-based B-tree nodes in the buffer pool.

These two aspects have not always been separated cleanly. Their difference becomes very apparent when a single database request is processed by multiple parallel threads. Specifically, two threads within the same transaction must "see" the same database contents, the same count of rows in a table, etc. This includes one thread "seeing" updates applied by the other thread. While one thread splits a B-tree node, however, the other thread should not observe intermediate and incomplete data structures. The difference also becomes apparent in the opposite case when a single operating system thread is multiplexed to serve all user transactions.

These two purposes are usually accomplished by two different mechanisms, locks and latches. Unfortunately, the literature on operating systems and programming environments usually uses the term locks for the mechanisms that in database systems are called latches, which can be confusing.

Locks separate transactions using read and write locks on pages, on B-tree keys, or even gaps (open intervals) between keys. The latter two methods are called key value locking and key range locking. Key range locking is a form of predicate locking that uses actual key values in the B-tree and the B-tree's sort order to define predicates. By default, locks participate in deadlock detection and are held until end-of-transaction. Locks also support sophisticated scheduling, e.g., using queues for pending lock requests and delaying new lock acquisitions for lock conversions, e.g., an existing shared lock to an exclusive lock. This level of sophistication makes lock acquisition and release fairly expensive, often thousands of CPU cycles, some of those due to cache faults in the lock manager's hash table.

Latches separate threads accessing B-tree pages, the buffer pool's management tables, and all other in-memory data structures shared among multiple threads. Since the lock manager's hash table is one of the data structures shared by many threads, latches are required while inspecting or modifying a database system's lock information. With respect to shared data structures, even threads of the same user transaction conflict if one thread requires a write latch. Latches are held only during a critical section, i.e., while a data structure is read or updated. Deadlocks are avoided by appropriate coding disciplines, e.g., requesting multiple latches in carefully designed sequences. Deadlock resolution requires a facility to roll back prior actions, whereas deadlock avoidance does not. Thus, deadlock avoidance is more appropriate for latches, which are designed for minimal overhead and maximal performance and scalability. Latch acquisition and release may require tens of instructions only, usually with no additional cache faults since a latch can be embedded in the data structure it protects.

Latch Coupling and B^{link}-Trees

Latches coordinate multiple concurrent threads accessing shared in-memory data structures, including images of on-disk storage structures while in the buffer pool. In the context of B-trees, latches solve several problems that are similar to each other but nonetheless lend themselves to different solutions.

First, a page image in the buffer pool must not be modified (written) by one thread while it is interpreted (read) by another thread. For this issue, database systems employ latches that differ from the simplest implementations of critical sections and mutual exclusion only by the distinction between read-only latches and read-write latches, i.e., shared or exclusive access. Latches are useful not only for pages in the buffer pool but also for the buffer pool's table of contents or the lock manager's hash table.

Second, while following a pointer (page identifier) from one page to another, e.g., from a parent node to a child node in a B-tree index, the pointer must not be invalidated by another thread, e.g., by deallocating a child page or balancing the load among neighboring pages. This issue requires retaining the latch on the parent node until the child node is latched. This technique is traditionally called "lock coupling" or better "latch coupling."

Third, "pointer chasing" applies not only to parent-child pointers but also to neighbor pointers, e.g., in a chain of leaf pages during a scan. This issue is similar to the previous, with two differences. On the positive side, asynchronous read-ahead may alleviate the frequency of buffer faults. On the negative side, deadlock avoidance among scans in opposite directions requires that latch acquisition code provides an immediate failure mode.

Fourth, during a B-tree insertion, a child node may overflow and require an insertion into its parent node, which may thereupon also overflow and require an insertion into the child's grandparent node. In the most extreme case, the B-tree's root node splits and a new root node is added. Going back from the leaf towards the B-tree root works well in single-threaded B-tree implementations, but in multi-threaded code it introduces the danger of deadlocks. This issue affects all updates, including insertion, deletion, and even record updates, the latter if length changes in variable-length records can lead to nodes splitting or merging. The most naïve approach, latching an entire B-tree with a single exclusive latch, is obviously not practical in multi-threaded servers.

One approach latches all nodes in exclusive mode during the root-to-leaf traversal. The obvious problem in this approach is the potential concurrency bottleneck, particularly at a B-tree's root. Another approach performs the root-to-leaf search using shared latches and attempts an upgrade to an exclusive latch when necessary. A third approach reserves nodes using "update" or "upgrade" latches. A refinement of these three approaches retains latches on nodes along its root-to-leaf search only until a lower, less-than-full node guarantees that split operations will not propagate up the tree beyond the lower node. Since most nodes are less than full, most insertion operations will latch no nodes in addition to the current one.

A fourth approach splits nodes proactively during a root-to-leaf traversal for an insertion. This method avoids both the bottleneck of the first approach and the failure point (upgrading a latch) of the second approach. Its disadvantage is that it wastes some space by splitting earlier than truly required. A fifth approach protects its initial root-to-leaf search with shared latches, aborts this search when a node requires splitting, restarts a new one, and upon reaching the node requiring a split, acquires an exclusive latch and performs the split.

An entirely different approach relaxes the data structure constraints of B-tress and divides a node split into two independent steps. Each node has a high fence key and a pointer to its right neighbor, thus the name B^{link}-trees. The right neighbor might not yet be referenced in the node's parent and a root-to-leaf search might need to proceed to the node's right neighbor. The first step of splitting a node creates the high fence key and a new right neighbor. The second, independent step posts the high fence key in the parent. The second step should happen as soon as possible yet it may be delayed beyond a system reboot or even a crash. The advantage of B^{link}-trees is that allocation of a new node and its initial introduction into the B-tree is a local step, affecting only one preexisting node. The disadvantages are that search may be a bit less efficient, a solution is needed to prevent long linked lists among neighbor nodes during periods of high insertion rates, and verification of a B-tree's structural consistency is more complex and perhaps less efficient.

Key Range Locking

Locks separate transactions reading and modifying database contents. For serializability, read locks are retained until end-of-transaction. Write locks are always retained until end-of-transaction in order to ensure the ability to roll back all changes if the transaction aborts. High concurrency requires a fine granularity of locking, e.g., locking individual keys in B-tree indexes. The terms key value locking and key range locking are often used interchangeably.

Key range locking is a special form of predicate locking. The predicates are defined by intervals in the sort order of the B-tree. Interval boundaries are the key values currently existing in the B-tree, which form half-open intervals including the gap between two neighboring keys and one of the end points.

In the simplest form of key range locking, a key and the gap to the neighbor are locked as a unit. An exclusive lock is required for any form of update of this unit, including modifying non-key fields of the record, deletion of the key, insertion of a new key into the gap, etc. Deletion of a key requires a lock on both the old key and its neighbor; the latter is required to ensure the ability to re-insert the key in case of transaction rollback.

High rates of insertion can create a hotspot at the "right edge" of a B-tree index on an attribute correlated with time. With next-key locking, one solution verifies the ability to acquire a lock on $+\infty$ but does not actually retain it. Such "instant locks" violate two-phase locking but work correctly if a single acquisition of the page latch protects both verification of the lock and creation of the new key on the page.

In those B-tree implementations in which a deletion does not actually erase the record and instead merely marks the record as invalid, "pseudo-deleted," or a "ghost" record, each ghost record's key participates in key range locking just like a valid record's key. Another technique to increase concurrency models a key, the appropriate neighboring open interval, and the combination of key and open interval as three separate items [9]. These items form a hierarchy amenable to multi-granularity locking. Moreover, since key, open interval, and their combination are all identified by the key value, additional lock modes can replace multiple invocations of the lock manager by a single one, thus eliminating the execution costs of this hierarchy.

Multi-granularity locking also applies keys and individual rows in a non-unique index, whether such rows are represented using multiple copies of the key, a list of row identifiers associated with a single copy of the key, or even a bitmap. Multi-granularity locking techniques exploiting a B-tree's tree structure or a B-tree's compound (multi-column) key have also been proposed. Finally, "increment" locks may be very beneficial for B-tree indexes on materialized summary views [5].

Both proposals need many details worked out, e.g., appropriate organization of the lock manager's hash table to ensure efficient search for conflicting locks and adaptation during structure changes in the B-tree (node splits, load balancing among neighboring nodes, etc.).

Key Applications

B-tree indexes have been called ubiquitous more than a quarter of a century ago [2], and they have become ever more ubiquitous since. Even for single-threaded applications, concurrent threads for maintenance and tuning require concurrency control in B-tree indexes, not to mention online utilities such as online backup. The applications of B-trees and B-tree locking are simply too numerous to enumerate them.

Future Directions

Perhaps the most urgently needed future direction is simplification – concurrency control and recovery functionality and code are too complex to design, implement, test, tune, explain, and maintain. Elimination of any special cases without a severe drop in performance or scalability would be welcome to all database development and test teams.

At the same time, B-trees are employed in new areas, e.g., Z-order UB-trees for spatial and temporal information, various indexes for unstructured data and XML documents, in-memory and on-disk indexes for data streams and as caches of reusable intermediate query results. It is unclear whether these application areas require new concepts or techniques in B-tree concurrency control.

Online operations – load and query, incremental online index creation, reorganization & optimization, consistency check, trickle load and zero latency in data warehousing including specialized B-tree structures.

Scalability – granularities of locking between page and index based on compound keys or on B-tree structure; shared scans and sort-based operations including "group by," merge join, and poor man's merge join (index nested loops join); delegate locking (e.g., locks on orders cover order details) including hierarchical delegate locking.

B-tree underpinnings for non-traditional database indexes, e.g., blobs, column stores, bitmap indexes, and master-detail clustering.

Confusion about transaction isolation levels in plans with multiple tables, indexes, materialized and indexed views, replicas, etc.

URL to Code

Gray and Reuter's book [6] shows various examples of sample code. In addition, the source of various open-source database systems is readily available.

Cross-references

▶ Concurrency Control and Recovery
▶ Database benchmarks – online transaction processing
▶ Locking Granularity and Lock Types
▶ pessimistic concurrency control
▶ phantoms
▶ precision locks
▶ predicate locks
▶ relational data warehousing
▶ System Recovery
▶ two-phase commit
▶ two-phase locking
▶ write-ahead logging

Recommended Reading

1. Bayer R. and Schkolnick M. Concurrency of operations on B-trees. Acta Inf., 9:1–21, 1977.
2. Comer D. The ubiquitous B-tree. ACM Comput. Surv., 11(2):121–137, 1979.
3. Eliot J. and Moss B. Open Nested Transactions: Semantics and Support. In Proc. Workshop on Memory Performance Issues 2006.

4. Graefe G. Hierarchical locking in B-tree indexes. BTW Conf., 2007, pp. 18–42.
5. Graefe G. and Zwilling M.J. Transaction support for indexed views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
6. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1993.
7. Jaluta I., Sippu S., and Soisalon-Soininen E. Concurrency control and recovery for balanced B-link trees. VLDB J., 14(2):257–277, 2005.
8. Lehman P.L. and Yao S.B. Efficient locking for concurrent operations on B-trees. ACM Trans. Database Syst., 6(4):650–670, 1981.
9. Lomet D.B. Key range locking strategies for improved concurrency. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 655–664.
10. Mohan C. ARIES/KVL: A key-value locking method for concurrency control of multiaction transactions operating on B-tree indexes. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 392–405.
11. Mohan C., Haderle D.J., Lindsay B.G., Pirahesh H., and Schwarz P.M. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. Database Syst., 17(1):94–162, 1992.
12. Ni Y., Menon V., Adl-Tabatabai A-R., Hosking AL., Hudson RL., Moss JEB., Saha B., and Shpeisman T. Open nesting in software transactional memory. In Proc. 12th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, 2007, pp. 68–78.
13. Srinivasan V. and Carey M.J. Performance of B-tree concurrency algorithms. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991, pp. 416–425.
14. Weikum G. Principles and realization strategies of multilevel transaction management. ACM Trans. Database Syst., 16(1):132–180, 1991.

Buffer Management

GIOVANNI MARIA SACCO
University of Torino, Torino, Italy

Definition

The database buffer is a main-memory area used to cache database pages. Database processes request pages from the buffer manager, whose responsibility is to minimize the number of secondary memory accesses by keeping needed pages in the buffer. Because typical database workloads are I/O-bound, the effectiveness of buffer management is critical for system performance.

Historical Background

Buffer management was initially introduced in the 1970s, following the results in virtual memory systems. One of the first systems to implement it was IBM System-R. The high cost of main-memory in the early days forced the use of very small buffers, and consequently moderate performance improvements.

Foundations

The buffer is a main-memory area subdivided into frames, and each frame can contain a page from a secondary storage database file. Database pages are requested from the buffer manager. If the requested page is in the buffer, it is immediately returned to the requesting process with no secondary memory access. Otherwise, a fault occurs and the page is read into a free frame. If no free frames are available, a "victim" page is selected and its frame is freed by clearing its content, after writing it to secondary storage if the page was modified. Usually any page can be selected as a victim, but some systems allow processes actively using a page to fix or pin it, in order to prevent the buffer manager from discarding it [5]. Asynchronous buffered write operations have an impact on the recovery subsystem and require specific protocols not discussed here.

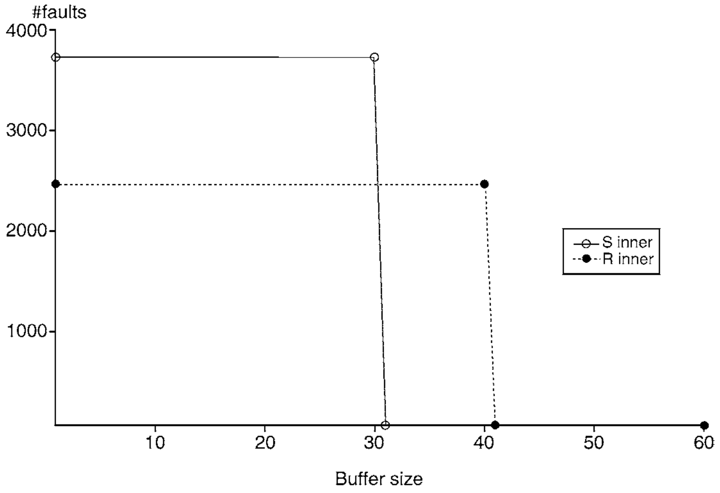
There are obvious similarities between buffer management and virtual memory (VM) systems [3]. In both cases, the caching system tries to keep needed pages in main-memory in order to minimize secondary memory accesses and hence speed up execution. As in VM systems, buffer management is characterized by two policies: the *admission policy*, which determines when pages are loaded into the buffer, and the *replacement policy*, which selects the page to be replaced when no empty frames are available. The admission policy normally used is demand paging (i.e., a missing page is read into the buffer when requested by a process), although prefetching (pages are read before processes request them) was studied (e.g., [1]). Since the interaction with the caching system is orders of magnitude less frequent in database systems than in VM systems, "intelligent" replacement policies such as LRU [3] (the Least Recently Used page is selected for replacement) can be implemented in software, with no performance degradation. Inverted page tables are used because their space requirement is proportional to the buffer size rather than to the entire database space as in normal page tables. Finally, database pages in the buffer can be shared among different processes, whereas the amount of sharing in VM systems is usually negligible.

The VM Approach

Besides minor architectural differences, the buffer manager can be used exactly as a virtual memory system for database pages, so that buffer management is transparent for the database system. In this VM approach, research is focused on effective replacement policies, which include GCLOCK and LRD (Least Reference Density) [5], LRU, and, more recently, efficient policies that account for page popularity, such as LRU-K [11].

LRU does not discriminate between frequently and infrequently referenced pages, and once a page is admitted in a buffer of B frames, it will stay there for at least B-1 references, even if not referenced again. Therefore, a potentially large portion of the buffer can be wasted by caching useless (i.e., infrequent) pages. *LRU-K* dynamically estimates the interreference distance for each page in the buffer by keeping the history of the last K references for each page in the buffer. A shorter interreference distance means a more frequently accessed page. Consequently, the page selected for replacement is the one with the largest interreference distance, i.e., the one with the maximum backward K-distance. The backward K-distance $b_t(p, K)$ at time t is defined as the distance backward to the Kth most recent reference the page p. When K = 1, only the last reference is considered, and LRU-1 is equivalent to LRU. As K grows, so do space and time overheads, because longer histories must be stored and kept ordered. At the same time, larger values of K improve the estimate of the interreference distance, but make the algorithm less responsive to dynamic changes in page popularity, so that, in practice, LRU-2 is normally used. The *2Q* (two queues) [7] algorithm provides an efficient, constant-time algorithm equivalent to LRU-2 replacement.

Although LRU-K improves LRU replacement through additional information about page access frequency, it is fundamentally different from the Least Frequently Used (LFU) replacement, because LFU does not adapt its estimate to evolving access patterns. The idea of exploiting both recency and frequency of access (also present in LRD replacement [5]) is extended by *LRFU* (least recently/frequently used) replacement [9], to model a parametric continuum of replacement policies ranging from LRU to LFU. The Combined Recency and Frequency (CRF) information is associated to each page in the buffer. CRF weights page references giving higher weights to more recent references, and combines frequency and recency information through a parameter λ ($0 \leq \lambda \leq 1$). The page to be replaced is the one with minimum CRF. When $\lambda = 0$, LRFU becomes LFU; when $\lambda = 1$, it becomes LRU. The optimal λ depends on the actual workload, but a self-tuning strategy can be used. An efficient implementation of LRFU exists, and experiments show this replacement strategy to outperform LRU-K with small buffer sizes.



Buffer Management. Figure 1. Number of faults as a function of the available buffer space, under LRU replacement.

A known problem in the VM approach is that processes with fast sequential scans over large relations tend to fill the buffer with useless pages accessed in the scan and flush the active pages of other processes out of memory, thereby significantly increasing the overall fault rate [12]. In a sequential scan, the current page is (i) rapidly accessed several times in order to read all the tuples on it, (ii) once the last tuple in the page is read, the page will never be reaccessed again. Reaccess in (i) makes VM strategies to incorrectly estimate that the current page is likely to be reaccessed in the future, and therefore to keep it in the buffer at the expense of other pages. In order to avoid this problem, some strategies use parametric correctives. LRU-K, for instance, uses the Correlated Reference Period. During this period, a page freshly admitted to the buffer cannot be replaced because additional references are expected. At the same time, subsequent references during this period are not tracked because they are expected to be correlated and do not give a reliable indication of future behavior. The length of such period is at the same time critical and difficult to estimate.

The Predictive Approach

A completely different, predictive approach was proposed with the *hot set model* [12,13]. In database systems with non-procedural data manipulation languages (such as relational or object-relational database systems), it is not the programmer but the system query optimizer that determines the access plan for a query. Such access plan is based on a small number of access primitives, such as sequential scans, nested loop joins, etc., whose reference string is known or can be estimated before actual execution. As an example, consider the execution of a tuplewise nested loop join between relation R (3 tuples/page, 40 pages) and relation S (2 tuples/page, 30 pages). Assuming S inner and indicating by Xi the i-th page of relation X, and by $\{a\}^N$ N repetitions of string a, the reference string is exactly $\{R1, \{S1\}^2,\ldots,\{S30\}^2\}^3, \{R2, \{S1\}^2,\ldots,\{S30\}^2\}^3,\ldots,\{R40, \{S1\}^2,\ldots,\{S30\}^2\}^3$. Figure 1 plots the number of faults as a function of the available buffer space, under LRU replacement.

Figure 1 shows that, differently from what happens in VM systems where fault curves tend to be smooth, the fault curve here is discontinuous. If the buffer is not large enough to contain all the pages in the inner relation plus one frame for the current page in the outer one, no reusal occurs and the fault rate is the same as for 1 frame. If sufficient space exists, all the needed pages are kept in the buffer, and the cost becomes linear in the number of pages of the two relations. In this case, the entire behavior of the plan in which S is inner is completely characterized by two *hot points* and their corresponding fault rate:

- hp1 = 1, faults(hp1) = |R|(1 + pages(S))
- hp2 = 1 + pages(S), faults(hp2) = pages (R) + pages(S)

The number of frames needed by a plan is called its hot set size, and usually is the largest hot point no larger than the total buffer size. Note that the length of the loop on the inner relation is exactly known at the database level, but it is very hard to discover in the VM approach.

The basic idea of the hot set model is that, given an access plan and a replacement policy, the number of faults as a function of the available buffer space can be predicted. Three main types of reusal, modeled upon primitive strategies, are considered:

1. Simple reusal. It models the sequential scan of a relation: the current page is accessed several times in order to read all the tuples on it, but once the last tuple in the page is read, the page will never be reaccessed again. There is only one hot point at 1 frame.
2. Loop reusal. Used in the example in Fig. 1. There are as many hot points as there are loops.
3. Index reusal. It includes both clustered and unclustered index access. Reusal can occur when indices are used in a join, or when an index is accessed by several processes. In unclustered index access, data pages are accessed randomly, and the hot set is estimated through Yao's function.

It must be stressed that LRU replacement is not required, and that other replacement strategies can be adopted [2,13]. As a matter of fact, both simple and loop reusal do not benefit from LRU replacement at all. The *query locality set model* (QLSM) [2] extends the hot set model by determining the hot set size on a file-instance basis rather than on a primitive operation basis, and by adopting replacement policies appropriate for each type of reusal. As an example, a nested loop join between R and S (S inner) is characterized by two different access patterns: a simple reusal (sequential scan) on R, requiring 1 frame, and a loop reusal on S, requiring 1 to pages(S) frames. MRU (most recently used) replacement can be used for loop reusal, since it is optimal for this type of reference string.

Since the reference string of the application is known, optimal replacement policies can be used. This opportunity is exploited in *OLRU* [14], which derives an optimal replacement policy for clustered index reusal, e.g., repeated access to clustered B+ trees. In this case, the Independent Reference Model (IRM) [3] is used. IRM was originally proposed as a theoretical evaluation model for VM systems, and assumes, in extreme synthesis, that the probability of reaccess for each page is known and stationary. Under this assumption, it can be proven that the optimal buffering strategy for a buffer of B frames consists of ordering the pages by decreasing reaccess probabilities, fixing the first B-1 high-probability pages in the buffer and using the remaining frame to access all the other low-probability pages. Since the optimal strategy only requires a ranking among pages, it can be straightforwardly applied to a B+ tree of order m and height h by assuming a uniform distribution of access to the leaf level. In this case, the probability of reaccessing a page at level j ($0 \leq j < h$) is $1/m^j$. The optimal policy is then an allocation by levels, i.e., fixing the first levels in the tree in the buffer. LRU replacement allocates the buffer by traversal stacks and is therefore suboptimal, unless severe deviations from uniformity in data page access occur, in which case OLRU and LRU are similar in performance. LRU-K, which assumes IRM as well, can be seen as a run-time approximation of OLRU.

A complete characterization of query buffer requirements and corresponding access costs is required for two major reasons. First, query optimizers need to have a precise *cost estimate* as a function of the available buffer size, in order to discriminate execution plans. This is especially important because (i) cost curves can intersect as shown in Fig. 1: R inner is cheaper in the interval [1,30], whereas S inner is better or no worse for buffer sizes of at least 31 frames; (ii) many query evaluation strategies (e.g., nested loops) do not explicitly account for memory, and they must be compared with other strategies (such as fragmentation/recursive hash partitioning join) in which available buffer space is fully accounted for and directly managed.

Second, *thrashing* phenomena [3] can occur in database buffers as in virtual memory systems, and become potentially more frequent as the number of concurrent active users increases. Thrashing occurs when the available memory is insufficient to keep all the pages each active process needs. Consequently, processes steal pages from each other and, if the available memory is severely overcommitted, the system collapses because all activity is devoted to swapping pages to and from main-memory. In VM systems, thrashing can be avoided by (i) monitoring the pagination device for excessive utilization or (ii) using the working set model [3] in order to estimate the memory requirements of active processes. In buffer management, there is no pagination device and accesses may be directed to a high number of secondary devices. In addition, the working set model, which is an expensive run-time estimator, cannot be efficiently used in database systems [13].

Thrashing avoidance, and the more general problem of *scheduling* queries for execution in order to optimally use buffer resources, is considerably simplified in the predictive approach because buffer requirements for each query are known before execution. The simplest policy [13] schedules queries for execution in such a way that the sum of their hot set sizes does not exceed the total buffer space. Each query can then be run in isolation in its own buffer partition, and this, by definition, avoids thrashing. However, page sharing, which is relatively frequent in database systems, is not accounted for. If two different processes request the same page, two faults occur. If sharing is considered these additional faults can be avoided, and the actual required buffer size can decrease because a page shared by several processes requires only one frame. For these reasons, hot set scheduling [12,13] maintains an additional global LRU chain to manage free pages, and, on a local page fault, scans the entire buffer for potential shared pages. In addition, a measure of buffer consumption is used in order to avoid unnecessarily inflating hot set sizes, which would result in serializing small queries behind queries with high buffer requirements. Variations of this scheduling algorithm include DBMIN [2] (where different local replacement policies can be used), scheduling with marginal gains [10], and scheduling with prefetching [1].

The VM Versus the Predictive Approach

When compared to the predictive approach, the VM approach has the advantage of placing all concerns on buffering into a single system component that can be seen as a black box from the rest of the system. In addition, the VM approach is more generally applicable since it does not require non-procedural interactions, and inherently implements page sharing.

However, for non-procedural systems, the predictive approach solves a number of important problems:

- Uniformity of estimated costs for query optimization. Query plans can be compared, regardless of whether methods directly manage memory or not.
- Thrashing avoidance, and efficient, low cost query scheduling. The predictive approach characterizes requirements before execution and does not require expensive run-time estimators.
- Resource planning for self-tuning databases. Predictive characterization of buffer requirements can be used to determine the optimal buffer size for actual workloads.
- Better performance. Since reference strings are known, optimal replacement policies can be used and buffer requirements carefully tuned.

There is a duality between detecting sequential scans and detecting page sharing. The VM approach has no problems in implementing page sharing. VM strategies work reasonably well for index access, but, despite correctives, they tend to break on simple and loop reusal. Conversely, sequential scan detection and inner loop size detection is trivial in the predictive approach, but, since prediction is based on a query run in isolation, run-time corrections are required for page sharing. This duality suggests that a combination of the two approaches might be beneficial.

Key Applications

The trend towards cheaper and larger main-memories does not make buffer management less important. In fact, the increase in available main-memory has been so far matched by a larger increase in secondary storage capacity and in the amount of data to be managed, in the complexity of queries, and in the number of users. Consequently, buffer management continues to be a fundamental topic for database systems, and indeed its results carry over to different areas, such as web caching.

Current research areas in database systems include automatic buffer sizing in self-tuning databases [15], extendibility of buffer management strategies [6], specific buffering strategies for object databases [8], XML databases and P2P data architectures, multidimensional databases, real-time databases where process priorities must be considered in scheduling, and, of course, new index structures or evaluation strategies for which buffer analysis is required [4]. In addition, challenging applications managing very large amounts of data such as sensor data, search engines, large digital libraries, etc. require high-performance buffer management.

Cross-references

▶ Evaluation of Relational Operators
▶ Indexing
▶ Recovery Techniques

Recommended Reading

1. Cai F.F., Hull M.E.C., and Bell D.A. Buffer management for high performance database systems. In Proc. High-Performance Computing on the Information Superhighway (HPC-Asia'97). Seoul, Korea, 1997, pp. 633–638.
2. Chou H.-T. and DeWitt D.J. An evaluation of buffer management strategies for relational database systems. In Proc. 11th Int. Conf. on Very Large Data Bases, 1985, pp. 174–188.
3. Coffman Jr. and Denning P.J. Operating Systems Theory. Prentice-Hall, Englewood Cliffs, NJ, 1973.
4. Corral A., Vassilakopoulos M., and Manolopoulos Y. The impact of buffering on closest pairs queries using R-trees. In Proc. Fifth East European Conference on Advances in Databases and Information Systems, 2001, pp. 41–54.
5. Effelsberg W. and Haerder T. Principles of database buffer management. ACM Trans. Database Syst., 9(4):560–595, 1984.
6. Goh L., Shu Y., Huang Z., and Ooi C. Dynamic buffer management with extensible replacement policies. VLDB J., 15(2): 99–120, 2006.
7. Johnson T. and Shasha D. 2Q: a low overhead high performance buffer management replacement algorithm. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 439–450.
8. Kemper A. and Kossmann D. Dual-buffering strategies in object bases. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 427–438.
9. Lee D., Choi J., Kim J.-H., Noh S.H., Min S.L., Cho Y., and Kim C.S. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies. In Proc. Int. Conf. on Measurement and Modeling of Computer Systems, 1999, pp. 134–143.
10. Ng R., Faloutsos C., and Sellis T. Flexible buffer allocation based on marginal gains. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991, pp. 387–396.
11. O'Neil E.J., O'Neil P.E., and Weikum G. The LRU-K page replacement algorithm for database disk buffering. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 297–306.
12. Sacco G.M. and Schkolnick M. A mechanism for managing the buffer pool in a relational database system using the hot set model. In Proc. 8th Int. Conf. on Very Data Bases, 1982, pp. 257–262.
13. Sacco G.M. and Schkolnick M. Buffer management in relational database systems. ACM Trans. Database Syst., 11(4):473–498, 1986.

14. Sacco G.M. Index access with a finite buffer. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 301–309.
15. Storm A.J., Garcia-Arellano C., Lightstone S.S., Diao Y., and Surendra M. Adaptive self-tuning memory in DB2. In Proc. 12th Int. Conf. on Very Large Data Bases, 2006, pp. 1081–1092.

Buffer Manager

Goetz Graefe
Hewlett-Packard Laboratories, Palo Alto, CA, USA

Synonyms

Cache manager

Definition

If a buffer pool is employed in a database management system, the associated software must provide appropriate services for efficient query processing, correct transaction execution, and effective sharing and reuse of database pages. It must provide interfaces for page access including pinning and latching pages, and it must invoke primitives for disk I/O and synchronization.

Historical Background

Database buffer pool management was studied heavily in the 1970s and 1980s as the new relational database management system posed new challenges, in particular non-procedural queries with range scans. Reliance of virtual memory and file system buffer pool was investigated but rejected due to performance issues (read-ahead, prefetch) and correctness issues (transaction management, write-behind, write-through).

Buffer pool management is currently not a very active research area. It may be revived in order to serve deep storage hierarchies, e.g., slow disk, fast disk, flash memory, main memory, and CPU cache. It may also be revived in the context of very large memories, e.g., query optimization that considers residence in the buffer pool or physical database design that includes temporary or partial indexes that exist only in the buffer pool.

Foundations

This section describes a database system's buffer pool and its management by focusing on management of individual pages in the buffer pool, replacement policies, asynchronous I/O, and the requirements imposed by concurrency control and recovery.

Buffer Pool Interfaces

The principal methods provided by the buffer pool manager request and release a page. A page request fixes or pins a page, i.e., it protects it from replacement as well as movement within the buffer pool, thus permitting access to the page by in-memory pointers. Variants of requesting and releasing a page apply to disk pages immediately after allocation (no need to read the page contents from disk) and immediately after deallocation (no need to save the page contents).

In addition, a buffer pool may provide methods for concurrency control among threads in order to protect page contents, also known as latching. Pragmatically, the methods to pin and to release a page include parameters that control latching.

Concurrency control among transactions is usually not provided by the buffer pool. In aid of logging and recovery, a buffer pool must support a method to force a page to disk and may support a method to retain one page until another page has been written.

For performance, a buffer pool may support methods to hint asynchronous prefetch, read-ahead, and write-behind. If the buffer pool serves a memory pool for other software layers, it must support methods for memory allocation and deallocation.

The principal methods upon which a buffer pool manager relies are disk reading and writing, including asynchronous operations, and scheduling primitives to implement latching. If the buffer pool size is dynamic, memory allocation and deallocation are also required. If the buffer pool supports multiple page sizes, it requires fast methods for moving (copying) page contents from one memory address to another.

Replacement Policies

The goal of the replacement policy (or retention policy) is to speed up future page accesses. Prediction of future accesses can be based on past accesses (how recent, how frequent, whether read or write) or on hints from higher software layers within the database management system. Standard policies include LRU (least recently used, implemented using a doubly-linked list), LRU-K (least recent K uses), LFU (least frequently used), second chance (usually implemented following a clock metaphor), generalized clock (using counters instead of a single bit per page frame), and combinations of those. Many combinations have been proposed, including the hot set model, the query locality set model, adaptive replacement cache, etc.

They differ in their heuristics to separate pages used only once, e.g., in a large sequential scan, from pages likely to be reused, e.g., pages containing the database catalog or root pages of B-tree indexes. Alternative designs let higher software layers hint the likelihood of reuse, e.g., love/hate hints or keep/toss hints.

Dirty pages (containing recent updates) may be retained longer than clean ones because their replacement cost and delay are twice as high (write plus read instead of merely a read operation) and because correct preparation for recovery may impose restrictions on the order in which pages are written. For example, write-ahead logging requires writing the relevant log page to stable storage before overwriting old database contents. On the other hand, non-logged operations (e.g., index creation) require flushing dirty pages as part of transaction commit.

Asynchronous I/O

Rather than merely responding to requests from higher software layers, a buffer pool may employ or enable asynchronous I/O, in three forms:

Write-behind cleans the buffer pool of dirty pages in order to complete update transactions as fast as possible yet enable quick page replacement without needing a write operation prior to a read operation. However, most write-behind is driven by checkpoints rather than page faults, based on typical checkpoint intervals (very few minutes) and retention intervals as calculated or optimized using the five-minute rule (many minutes or even hours). A write-behind operation may leverage a disk seek forced by another read or write operation. Alternatively, a write-behind operation may move the data to achieve this effect, e.g., in log-structured file systems and write-optimized database indexes.

Read-ahead speeds up large scans, e.g., a range scan in a B-tree or a complete scan of a heap structure. The appropriate amount of read-ahead is the product of bandwidth and latency, i.e., the smallest of I/O bandwidth and processing bandwidth multiplied with the delay from initiation to completion of a read operation. Read-ahead may be triggered by observation of the access pattern or by a hint from a higher software layer, e.g., a table scan in a query execution plan.

Prefetch accelerates fetch operations by loading into the buffer pool precisely those pages that contain needed data records. Prefetch in heap structures is quite straightforward. In B-tree indexes, prefetch may apply to all tree levels or merely to the leaf level, combined with synchronous read operations or large read-ahead for interior B-tree nodes. Prefetch speeds up not only to ordinary forward processing but also to transaction rollback as well as system recovery.

Concurrency Control and Recovery

The buffer pool may participate in concurrency control, e.g., if multi-version concurrency control requires multiple versions of individual pages. The mechanisms for pinning and latching are essential for coordination of multiple threads accessing the same in-memory data structures, including in-memory images of on-disk pages.

The buffer pool always participates in the preparation for recovery including transaction rollback, media recovery, and system recovery. For example, by retaining all modified pages in the buffer pool until transaction commit, one can avoid logging *undo* information in the persistent log – this is called a “no steal” policy. By forcing merely log pages to stable storage, one can avoid writing all modified database pages back to disk – this is called a “no force” policy. Most database management systems use “steal – no force” by default. The log volume of index creation and similar operations is often minimized using a “force” policy.

Logging volume can be reduced by ensuring specific write sequences. For example, when a B-tree node is split and some records are moved to a new node, one can avoid logging the moved records by writing the new page before writing the modified old page.

The buffer pool also participates in checkpoint processing. In addition to recording active transactions, a checkpoint must write all dirty database pages to the log or to the database. Proactive asynchronous write-behind may lessen the number of write operations during the checkpoint.

Cooperative Buffer Pool Management

Multiple buffer pools may cooperate. Prototypical examples include client-server operation (in which the buffer pools form a hierarchy) and shared-disk database systems (in which the buffer pools are peers). Those environments require optimizations for both data traffic among buffer pools and control messages, in particular for concurrency control and lock management.

Key Applications

A buffer pool and its management software are required in any database management system that employ multiple levels in a memory hierarchy, process and store data at different levels, and do not rely other means for moving data between those levels. The main example is main memory and disks – the buffer pool manager manages which data pages are immediately available for access, e.g., from the query execution engine. A CPU cache is a level in the memory hierarchy above the main memory, but data movement between main memory and CPU cache are automatic. A database management system could rely on a file system and its buffer pool manager but usually does not due to performance issues (e.g., read-ahead, prefetch) and due to correctness issues (write-behind, write-through). A database management system could rely on virtual memory provided by the operating system but typically does not for the same reasons.

Future Directions

While basic buffer pool management in traditional database management systems is well understood, there are many developments that build upon it. For example, integration of database cache, mid-tier cache, and web cache may become imperative in order to maximize efficiency and thus minimize costs for hardware, management, power, and cooling.

Buffer pools and buffer management will become more pervasive with the increased virtualization of storage and processing. At the same time, it will become more complex due to missing information on the true cost and location of data. It will also become more pervasive and complex due to increasing use of peer-to-peer storage, communication, and processing.

Any buffer pool becomes more effective with data compression and co-location. Compression reduces the space required locally (in the buffer pool) and remotely. Co-location techniques such as master-detail clustering enable access to multiple related records or pieces of information within a single frame in the buffer pool and with a single access to the remote location.

In a very large buffer pool, one might create temporary on-disk structures that never even exist on disk. For example, a temporary index on a permanent table may be created yet retained in the buffer pool. During contention in the buffer pool, the index is dropped. Ideally, such an index is partitioned, created and dropped incrementally, and left behind as a free side effect of query execution.

The data structures that manage a buffer pool, both its contents descriptors and its replacement policy, can be complex yet require very high concurrency, in particular in forthcoming many-core processors. Hardware-assisted transactional memory may simplify the software implementation effort as well as increase concurrency and performance.

In deep memory hierarchies, e.g., a three-level hierarchy of traditional memory, flash memory, and disk, contents descriptors and data structures in aid of the replacement policy may be separate. For example, the contents descriptors may need to be persistent if the flash memory is part of the persistent database, but all data structures for the replacement policy (between flash memory and disk) might be in the traditional memory.

Cross-references

- ▶ Buffer pool
- ▶ B-tree locking
- ▶ Concurrency control and recovery
- ▶ Flash memory
- ▶ Lock manager
- ▶ Storage hierarchy
- ▶ Storage layer
- ▶ Storage manager

Recommended Reading

1. Bansal S. and Modha D.S. CAR: Clock with adaptive replacement. In Proc. 3rd USENIX Conf. on File and Storage Technologies, 2004, pp. 187–200.
2. Chou H-T. and DeWitt D.J. An evaluation of buffer management strategies for relational database systems. Algorithmica, 1(3):311–336, 1986.
3. Effelsberg W. and Härder T. Principles of database buffer management. ACM Trans. Database Syst., 9(4):560–595, 1984.
4. Gray J. and Putzolu G.R. The 5 minute rule for trading memory for disk accesses and the 10 byte rule for trading memory for CPU Time. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 395–398.
5. Ramamurthy R. and DeWitt D.J. Buffer-pool Aware Query Optimization. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 250–261.

6. Stonebraker M. Operating System Support for Database Management. Commun. ACM, 24(7):412–418, 1981.

Buffer Pool

GOETZ GRAEFE
Hewlett-Packard Laboratories, Palo Alto, CA, USA

Synonyms

I/O cache, Page cache

Definition

Cost constraints (dollars per gigabyte) prohibit in-memory databases in most cases, but processors can access and manipulate data only while it is in memory. The in-memory buffer pool holds database pages currently in use and retains those deemed likely to be used again soon.

The buffer pool and the buffer management component within the storage layer of a database management system provide fast access and fast recall of on-disk pages using in-memory images of those pages. In addition to the size of individual pages and of the entire buffer pool, key issues are (i) page replacement in response to buffer faults, and (ii) page retention and update in aid of database recovery.

A database buffer pool differs from virtual memory as it contributes to correctness and efficiency of query and update processing, e.g., by pinning pages while in use and by ensuring a write sequence that guarantees the ability to recover. In some systems, the buffer pool permits "stealing" memory for query processing operations such as sorting and hash join, for utilities such as reorganization and consistency checks, etc.

Historical Background

Gray and Putzolo's paper introducing the five-minute rule makes a strong case for using multiple levels of the memory hierarchy for data collections that include both hot and cold data, i.e., data that are accessed with different frequencies. Hot data are kept at a high level of the memory hierarchy, whereas cold data are fetched as needed and buffered temporarily.

A buffer pool is also needed if the provided granularity of access is too coarse. Disks permit random page accesses, whereas query execution, predicate evaluation, etc. require accesses to individual records, fields, and bytes.

Early research demonstrated that virtual memory is not appropriate for use in database management systems, as observed by Härder, Stonebraker, Traiger, and their research teams for early relational database management systems. Reasons include both correctness, specifically with respect to recovery, and performance, specifically asynchronous read-ahead and write-behind.

In order to avoid double page faults, a buffer pool should not be subject to page replacement by virtual memory provided by the operating system. One means to achieve this is to vary the size of the buffer pool in response to paging rates of the virtual memory. The five-minute rule gives guidance for the appropriate sizing of memory and buffer pool.

Techniques to map on-disk database contents into virtual memory advanced in the context of object-oriented databases and persistent programming languages but did not result in wide adoption.

Flash memory still requires an in-memory buffer pool for access to bytes within pages, yet it can also serve as a buffer pool for pages that require faster access than rotating disks can provide. In fact, buffer pool management techniques such as replacement policies apply to all levels in a multi-level memory hierarchy, e.g., CPU caches, RAM, flash memory, disk caches, rotating disk, tape media, etc. In an extreme case, a fast disk may serve a buffer pool for a slower disk; alternatively, both disks may serve as permanent storage and buffer replacement policies may be adapted for page placement on those disks, possibly including frequent page migration.

Foundations

This section describes a database system's buffer pool, replacement policies, and the requirements imposed by concurrency control and recovery.

Buffer Frames

A buffer pool contains many frames, each capable of holding the image of an on-disk page. Pages can be fixed-length or variable-length, i.e., multiple base pages, typically a power of 2. Space management is complex for variable-length pages; one technique employed commercially relies on multiple buffer pools with a single page size in each, i.e., a fixed-length page frame in each buffer pool.

In addition to being idle or unused, buffer frames can be pinned to protect the page from replacement, latched (locked) to protect against concurrent readers or writers, or in transit from or to disk. Pinning and latching are important for performance; they enable the database's query execution software to inspect or update records directly in the buffer pool.

A buffer pool may hold multiple versions of the same on-disk page in aid of transaction isolation, compression, asynchronous write-behind, or protection from partial writes. Write-behind with a single copy inhibits further updates until the write operation completes. The copy step may compact free space between variable-length records.

A small descriptor data structure is used for each frame in the buffer pool. It identifies the on-disk page and its status with respect to pinning, latching, versioning, etc. The descriptor also participates in a look-up scheme, typically a hash table, and in data structures used for page replacement.

Buffer Pool Data Structures

In addition to page frames and their descriptors, a buffer pool needs data structures to locate buffered pages and to manage page replacement including page frames currently unused. Locating a buffered page, i.e., mapping from a disk address (page number) to an in-memory address, usually is implemented with a hash table.

The data structures needed for page replacement depend on the page replacement policy – one method that is simple but not ideal is to employ a doubly linked list of page descriptors with pages ordered the time since they were last used. When a page is pinned, it is removed from the list. When it is unpinned, it is inserted at the head of the list. When a page is needed for replacement, the page at the tail of the list is chosen.

Replacement Policies

The goal of the replacement policy (or retention policy) is to speed up future page accesses. Prediction of future accesses can be based on past accesses (how recent, how frequent, whether read or write) or on hints from higher software layers within the database management system. Standard policies include LRU (least recently used, implemented using a doubly-linked list), LRU-K (least recent K uses), LFU (least frequently used), second chance (usually implemented following a clock metaphor), generalized clock (using counters instead of a single bit per page frame), and combinations of those. Many combinations have been proposed, including the hot set model, the query locality set model, adaptive replacement cache, etc.

They differ in their heuristics to separate pages used only once, e.g., in a large sequential scan, from pages likely to be reused, e.g., pages containing the database catalog or root pages of B-tree indexes. Alternative designs let higher software layers hint the likelihood of reuse, e.g., love/hate hints or keep/toss hints.

Dirty pages (containing recent updates) may be retained longer than clean ones because their replacement cost and delay are twice as high (write plus read instead of merely a read operation) and because correct preparation for recovery may impose restrictions on the order in which pages are written. For example, write-ahead logging requires writing the relevant log page to stable storage before overwriting old database contents. On the other hand, non-logged operations (e.g., index creation) require flushing dirty pages as part of transaction commit.

Key Applications

A buffer pool and its management software are required in any database management system that employ multiple levels in a memory hierarchy, process and store data at different levels, and do not rely other means for moving data between those levels. The main example is main memory and disks – the buffer pool manager manages which data pages are immediately available for access, e.g., from the query execution engine. A CPU cache is a level in the memory hierarchy above the main memory, but data movement between main memory and CPU cache are automatic. A database management system could rely on a file system and its buffer pool manager but usually does not due to performance issues (e.g., read-ahead, prefetch) and due to correctness issues (write-behind, write-through). A database management system could rely on virtual memory provided by the operating system but typically does not for the same reasons.

Future Directions

In deep memory hierarchies, e.g., a three-level hierarchy of traditional memory, flash memory, and disk, contents descriptors and data structures in aid of the replacement policy may be separate. For example, the contents descriptors may need to be persistent if the flash memory is part of the persistent database, but all data structures for the replacement policy (between flash memory and disk) might be in the traditional memory.

Cross-references

- ▶ B-tree locking
- ▶ Concurrency control and recovery
- ▶ Lock manager

- ▶ Flash memory
- ▶ Storage hierarchy
- ▶ Storage layer
- ▶ Storage manager

Recommended Reading

1. Bansal S. and Modha D.S. CAR: Clock with adaptive replacement. In Proc. 3rd USENIX Conf. on File and Storage Technologies, 2004, pp. 187–200.
2. Chou H-T. and DeWitt D.J. An evaluation of buffer management strategies for relational database systems. Algorithmica, 1(3):311–336, 1986.
3. Effelsberg W. and Härder T. Principles of database buffer management. ACM Trans. Database Syst., 9(4):560–595, 1984.
4. Gray J. and Putzolu G.R. The 5 minute rule for trading memory for disk accesses and the 10 byte rule for trading memory for CPU time. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 395–398.
5. Ramamurthy R. and DeWitt D.J. Buffer-pool aware query optimization. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 250–261.
6. Stonebraker M. Operating system support for database management. Commun. ACM, 24(7):412–418, 1981.

Business Intelligence

Stefano Rizzi
University of Bologna, Bologna, Italy

Definition

Business intelligence is a business management term that indicates the capability of adding more intelligence to the way business is done by companies. More precisely, it refers to a set of tools and techniques that enable a company to transform its business data into timely and accurate information for the decisional process, to be made available to the right persons in the most suitable form. Business intelligence systems are used by decision makers to get a comprehensive knowledge of the business and of the factors that affect it, as well as to define and support their business strategies. The goal is to enable data-based decisions aimed at gaining competitive advantage, improving operative performance, responding more quickly to changes, increasing profitability and, in general, creating added value for the company.

Key Points

Though business intelligence has its roots in reporting systems, it was born as a term within the industrial world in the early 1990's, to indicate a set of technologies aimed at satisfying the managers' request for efficiently and effectively analyzing the enterprise data in order to better understand the situation of their business and improving the decision process. In the mid-1990's business intelligence became an object of interest for the academic world, and ten years of research managed to transform a bundle of naive techniques into a well-founded approach to information extraction and processing that led to defining the modern architectures of data warehousing systems. Currently, business intelligence includes not only the tools to gather, provide access to, and analyze data and information about company operations, but also a wide array of technologies used to support a closed decisional loop (known as *Business Performance Management*) where the company performance is measured by a set of indicators (commonly called *Key Performance Indicators, KPIs*) whose target values are determined by the company strategy, and where the actions taken are aimed at matching current and target values for these indicators.

From an architectural point of view, the core of a business intelligence system is usually a data warehouse that stores the corporate historical data in a consistent and integrated form. A number of applications may be built around the data warehouse, for instance aimed at supporting OLAP analysis, data mining, what-if analysis, forecasting, balanced scorecards preparation, geospatial analysis, click-stream analysis. The architecture may be completed by a reactive data flow, more suited for monitoring the time-critical operational processes by supporting real-time applications.

Cross-references

- ▶ Data Mining
- ▶ Data Warehouse Applications
- ▶ Data Warehousing Systems: Foundations and Architectures
- ▶ On-Line Analytical Processing
- ▶ What-if Analysis

Recommended Reading

1. Eckerson W. Performance dashboards: Measuring, monitoring, and managing your business. Wiley, 2005.
2. Golfarelli M., Rizzi S., Cella I. Beyond data warehousing: What's next in business intelligence? In Proc. ACM 7th Int. Workshop on Data Warehousing and OLAP, 2004, pp. 1–6.
3. Moss L.T. and Atre S. Business Intelligence Roadmap: The complete project lifecycle for decision-support applications. Addison-Wesley Information Technology Series, 2003.

Business Process Execution Language

W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven,
The Netherlands

Synonyms

BPEL; BPEL4WS

Definition

The *Business Process Execution Language for Web Services* (BPEL) has emerged as a standard for specifying and executing processes. It is supported by many vendors and positioned as the "process language of the Internet." BPEL is XML based and aims to enable "programming in the large," i.e., using BPEL new services can be composed from other services.

Key Points

BPEL [2,3] supports the modeling of two types of processes: executable and abstract processes. An *abstract*, (not executable) *process* is a business protocol, specifying the message exchange behavior between different parties without revealing the internal behavior for any one of them. This abstract process views the outside world from the perspective of a single organization or (composite) service. An *executable process* views the world in a similar manner. However, things are specified in more detail such that the process becomes executable, i.e., an executable BPEL process specifies the execution order of a number of *activities* constituting the process, the *partners* involved in the process, the *messages* exchanged between these partners, and the *fault* and *exception handling* required in cases of errors and exceptions.

A BPEL process itself is a kind of flow-chart, where each element in the process is called an *activity*. An activity is either a primitive or a structured activity. The set of *primitive activities* contains: `invoke`, invoking an operation on a web service; `receive`, waiting for a message from an external source; `reply`, replying to an external source; `wait`, pausing for a specified time; `assign`, copying data from one place to another; `throw`, indicating errors in the execution; `terminate`, terminating the entire service instance; and `empty`, doing nothing.

To enable the presentation of complex structures the following *structured activities* are defined: `sequence`, for defining an execution order; `switch`, for conditional routing; `while`, for looping; `pick`, for race conditions based on timing or external triggers; `flow`, for parallel routing; and `scope`, for grouping activities to be treated by the same fault-handler. Structured activities can be nested and combined in arbitrary ways. Within activities executed in parallel the execution order can further be controlled by the usage of `links` (sometimes also called control links, or guarded links), which allows the definition of directed graphs. The graphs too can be nested but must be acyclic.

The terminology above is based on BPEL 1.1 which was introduced in 2003 [3]. A new version of the standard [2] was published in 2007. This version has been approved as an OASIS Standard. This new version resolves many semantical issues [1,5]. Moreover, new activity types such as `repeatUntil`, `validate`, `forEach` (parallel and sequential), `rethrow`, `extensionActivity`, and `compensateScope`, have been added and some of the existing activities have been renamed (`switch/case` renamed to `if/else` and `terminate` renamed to `exit`). Currently, many extensions are under development, including BPEL4People which enables BPEL activities to be executed by human resources [4].

Cross-references

- ▶ BPMN
- ▶ Business Process Management
- ▶ Choreography
- ▶ Composition
- ▶ Orchestration
- ▶ Web Services
- ▶ Workflow Management
- ▶ Workflow Patterns

Recommended Reading

1. Aalst van der W.M.P., Dumas M., ter Hofstede A.H.M., Russell N., Verbeek H.M.W., and Wohed P. Life after BPEL? In WS-FM, 2005, pp. 35–50.
2. Alves A., Arkin A., Askary S., Barreto C., Bloch B., Curbera F., Ford M., Goland Y., Guzar A., Kartha N., Liu C.K., Khalaf R., Koenig D., Marin M., Mehta V., Thatte S., Rijn D., Yendluri P., and Yiu A. Web services business process execution language, version 2.0 (OASIS Standard). WS-BPEL TC OASIS. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007.
3. Andrews T., Curbera F., Dholakia H., Goland Y., Klein J., Leymann F., Liu K., Roller D., Smith D., Thatte S., Trickovic I., and Weerawarana S. Business process execution language for web services, version 1.1. Standards Proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.

4. Kloppmann M., Koenig D., Leymann F., Pfau G., Rickayzen A., von Riegen C., Schmidt P., and Trickovic I. WS-BPEL extension for people BPEL4People. In Proc. 22nd Int. Conf. on Conceptual Modeling, 2005.
5. Wohed P., van der Aalst W.M.P., Dumas M., and ter Hofstede A.H.M. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In I.Y. Song, S.W. Liddle, T.W. Ling, and P. Scheuermann, editors. In Proc. 22nd Int. Conf. on Conceptual Modeling, 2003, pp. 200–215.

Business Process Management

W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven,
The Netherlands

Synonyms

Workflow management; Process management; Case handling

Definition

Information technology has changed business processes within and between enterprises. Traditionally, information technology was mainly used to support individual tasks ("type a letter") and to store information. However, today business processes and their information systems are intertwined. Processes heavily depend on information systems and information systems are driven by the processes they support [1]. *Business Process Management* (BPM) is concerned with the interactions between processes and information systems. An important element of BPM is the modeling and analysis of processes. Processes can be designed using a wide variety of languages ranging from BPMN to Petri nets. Some of these languages allow for analysis techniques (e.g., model checking and simulation) to answer questions related to correctness and performance. Models can be used to configure generic software tools, e.g., middleware, workflow management systems, ERP systems, etc. These systems, also referred to as *Business Process Management Systems* (BPMS), are used to enact relevant business processes. A business process management system can be defined as: *a generic software system that is driven by explicit process designs to enact and manage operational business processes.* The system should be "process-aware" and "generic" in the sense that it is possible to modify the processes it supports. The process designs are often graphical and the focus is on structured processes that need to handle many cases. Workflow management systems are typical examples of such "process-aware" systems. An important technological enabler for business process management systems is the Service Oriented Architecture (SOA). The partitioning of processes into services makes it easier to isolate the process-logic.

Historical Background

Traditionally, information systems are viewed from either a *process-centric* or an *information-centric* perspective. The information-centric view focuses on the information managed by the system. Database management systems provide the functionality required to store and retrieve data. Since the 1970's, there have been consensus on the modeling of data. Although there are different languages and different types of database management systems, the fundamental concepts are quite stable for the information-centric view of information systems. The process-centric view on information systems on the other hand can be characterized by the term "divergence." There is little consensus on the fundamental concepts. Despite the availability of established formal languages (e.g., Petri nets and process calculi) industry has been pushing ad-hoc/domain-specific languages. As a result there is a plethora of systems and languages available today.

An good starting point from a scientific perspective is the early work on office information systems. In the 1970's, people like Skip Ellis, Anatol Holt, and Michael Zisman already worked on so-called office information systems, which were driven by explicit process models [6]. It is interesting to see that the three pioneers in this area independently used Petri-net variants to model office procedures. During the 1970's and 1980's, there was great optimism about the applicability of office information systems. Unfortunately, few applications succeeded. As a result of these experiences, both the application of this technology and research almost stopped for a decade. Consequently, hardly any advances were made in the 1980's. In the 1990's, there again was a huge interest in these systems [7]. The number of workflow management systems developed in the period 1995–2005 and the many papers on workflow technology illustrate the revival of office information systems. Today workflow management systems are readily available. However, their application is still limited to specific industries such as banking and insurance. In fact, workflow technology is often hidden inside other systems. For example, ERP systems like SAP and Oracle provide workflow engines. Many other platforms include workflow-like functionality. For example, integration and application infrastructure software such as IBM's Websphere provides extensive process support.

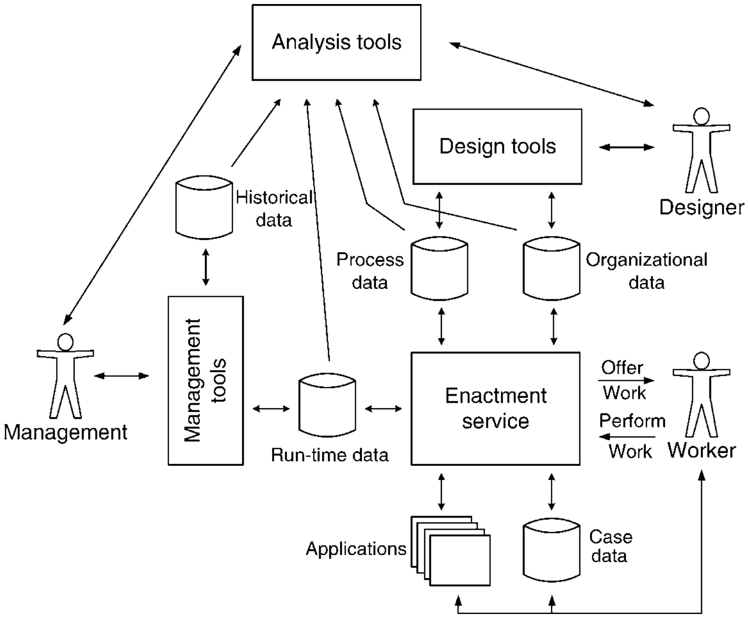
When comparing today's business process management systems to the workflow management systems of the nineties two things can be noted. First of all, the focus is no longer exclusively on automation and enactment, e.g., process analysis (simulation, process mining, verification, etc.) is increasingly important. Second, the use of web technology makes it easier to realize such systems even if processes are scattered over multiple organizations.

Foundations

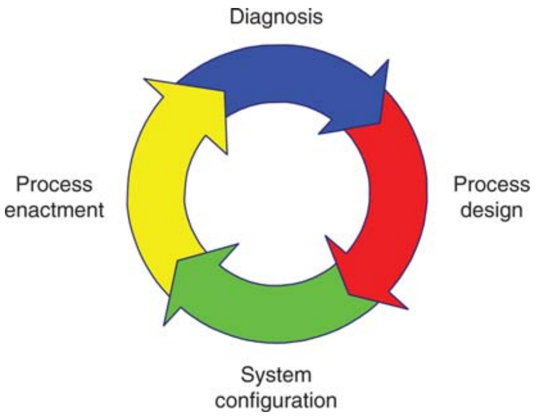
Business process management looks at the relationships between business processes and information systems. Using information systems in an innovative way enables new types of business processes. For example, making paper documents electronic may enable the concurrent execution of tasks thus shortening flow times. Moreover, characteristics of business processes lead to requirements for business process management systems (cf. workflow patterns). Business process management is *not* limited to the automation of business processes. For example, it is vital to *analyze* processes before and after they are enacted. During the design phase it is vital to use verification techniques to assess the correctness of the process design. Moreover, simulation techniques can be used to estimate the performance of the process once it is realized. While the processes is running, the information system needs to record information about actual events and realized performance. Using process mining techniques and other types of business intelligence, the event logs of systems can be analyzed. Based on such a diagnosis, the process can be improved.

The modeling and analysis of processes plays a central role in business process management. Therefore, the choice of language to represent an organization's processes is essential. Three types of languages can be identified:

1. *Formal languages*: Processes have been studied using theoretical models. Mathematicians have been using Markov chains, queueing networks, etc. to model processes. Computer scientists have been using Turing machines, transition systems, Petri nets, and process algebras to model processes.



Business Process Management. Figure 1. Architecture of a business process management system.



Business Process Management. Figure 2. BPM life-cycle.

All of these languages have in common that they have *unambiguous semantics* and allow for *analysis*.

2. *Conceptual languages*: Users in practice have problems using formal languages. They prefer to use higher-level languages. Examples are BPMN (Business Process Modeling Notation), EPCs (Event-Driven Process Chains), UML activity diagrams, etc. These language are typically *informal*, i.e., they do not have a well-defined semantics and do not allow for analysis. Moreover, the lack of semantics makes it impossible to directly execute them.
3. *Execution languages*: Formal language typically abstract from "implementation details" (e.g., data structures) and conceptual languages only provide an approximate description of the desired behavior. Therefore, more technical languages are needed for enactment. An example is the BPEL (Business Process Execution Language) language. Most vendors provide a proprietary execution language.

The existence and parallel use of these three types of languages causes many problems. The lack of consensus makes it difficult to exchange models. The gap between conceptual languages and execution languages leads to re-work and a disconnect between users and implementers. Moreover, both types of languages are not supported by advanced analysis tools.

Figure 1 shows the typical architecture of a business process management system. The figure also shows three roles of people involved: management, designer, and worker. The "heart" of the business process management system is the enactment service also known as "workflow engine" [4,7]. This engine is offering the right pieces of work (work-items) to workers at the right point in time. In order to do this, it needs to have detailed descriptions of the processes and organizations involved. Using design tools one can model processes and organizations. Note that Fig. 1 presents an idealized view. As indicated before there may be different languages (formal, conceptual, and execution languages) involved. The designer may first model the process in an informal manner. This model is then converted into a model that can be enacted or analyzed. The enactment service is driven by models in order to offer the right piece of work to the right persons at the right time. Moreover, the enactment service is starting applications and provides access to case data. During execution all kinds of information are recorded and at any time there is a (partial) history (i.e., audit trails, event logs, etc.) and a current state (run-time data). This information can be used for all kinds of analysis. Some types of analysis focus on the process design (e.g., verification and simulation). Other types of analysis focus on the actual behavior of the process (e.g., process mining).

As indicated before, different types of people are involved (management, designers, and workers). Moreover, business process management systems have a characteristic *life-cycle*. Figure 2 shows the four phases of such a life-cycle [7]. In the *design phase*, the processes are (re)designed. In the *configuration phase*, designs are implemented by configuring a process aware information system (e.g., a BPMS). After configuration, the *enactment phase* starts where the operational business processes are executed using the system configured. In the *diagnosis phase*, the operational processes are analyzed to identify problems and to find things that can be improved. The focus of traditional workflow management (systems) is on the lower half of the BPM life-cycle. As a result there is little support for the diagnosis phase. Moreover, support in the design phase is limited to providing an editor while analysis and real design support are missing. It is remarkable that few systems provide good support simulation, verification, and validation of process designs. Another problem of conventional workflow systems is the lack of flexibility. Fortunately, the emphasis is shifting from automation of highly structured processes to issues such as flexibility and analysis. Case handling systems such as FLOWer and academic prototypes such as DECLARE, YAWL/worklets, and ADEPT offer innovative ways of supporting flexible processes. Process mining tools such as ProM, ARIS PPM, etc. allow for the analysis of actual behavior. This supports the diagnosis phase and triggers process improvement.

The architecture demonstrated in Fig. 1 does not show any organizational boundaries. In the traditional setting it was very difficult to support inter-organizational processes. Web services and the Service Oriented Architecture (SOA) simplify the distribution of processes over different organizations [8]. Moreover, the paradigm shift towards services has also changed the architecture within a single organization. When focusing on processes, two terms are important: (i) choreography and (ii) orchestration.

Choreography is concerned with the exchange of messages between those services. *Orchestration* is concerned with the interactions of a single service with its environment. While choreography can be characterized by reaching an agreement and the monitoring of the overall progress, the focus of orchestration is more on the implementation of a particular service by describing the process logic and linking this to neighboring services. Orchestration languages are close to traditional workflow languages (BPMN, BPEL, Petri nets, etc.). An important characteristic of such languages is the ability to compose a service by using other services. The role of choreography languages (e.g., WS-CDL) is less clear.

Business process management is clearly related to management science. For example, topics such as operations research, operations management, business process re-engineering are highly relevant [9]. There are also clear links with coordination languages and theory. Coordination can be defined as "managing dependencies between activities." Process modeling languages and concepts such as choreography and orchestration are obviously related to coordination.

Key Applications

Banking

The financial industry has changed dramatically using both business process re-engineering and workflow-like technologies. Many processes have been rationalized using business process management techniques. The rise of e-banking led to a dramatic reduction of people and offices. Banking processes are supported and monitored by business process management systems.

Government

Government organizations need to react quickly to new legislation, i.e., the corresponding processes need to be modified based on changes in tax laws, customs procedures, immigration laws, corporate governance, safety regulations, etc. Business process management assists in dealing with these changes and further improving the processes.

Business-to-Business

Mergers and virtual enterprises trigger the need for cross-organizational workflows. Web services and business process management techniques can assist in connecting process fragments from different organizations. The SOA combined with languages like BPEL provides a good basis for cross-organizational workflows.

Health-care

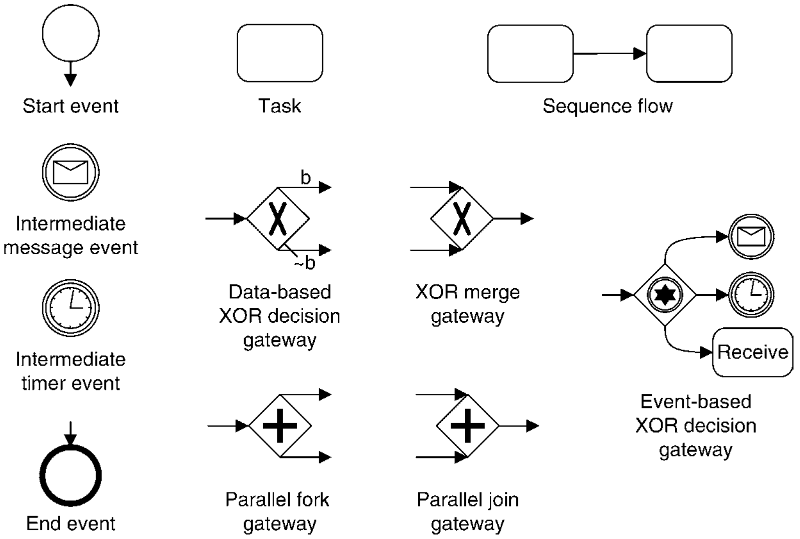
Business process management techniques have mainly been applied to structured processes. Given the nature of care processes, it is not easy to streamline these processes and to support them with workflow-like systems. However, they only way to reduce costs and improve effectively is to provide better support for such processes. Hence, the health-care domain poses an interesting and relevant challenge for business process management.

Cross-references

▶ BPEL
▶ BPMN
▶ Composition
▶ Choreography
▶ Orchestration
▶ Process Mining
▶ Web Services
▶ Workflow Management
▶ Workflow Management and Workflow Management Systems
▶ Workflow Model Analysis
▶ Workflow Patterns

Recommended Reading

1. Dumas M., van der Aalst W.M.P., and ter Hofstede A.H.M. Process-Aware Information Systems: Bridging People and Software Through Process Technology. Wiley, New York, NY, USA, 2005.
2. Georgakopoulos D., Hornick M., and Sheth A. An Overview of Workflow Management: From Process Modeling to Workflow



Business Process Modeling Notation. Figure 1. BPMN notation.

Automation Infrastructure. Distrib. Parallel Databases, 3:119–153, 1995.

3. Jablonski S. and Bussler C. Workflow Management: Modeling Concepts, Architecture, and Implementation. International Thomson Computer, London, UK, 1996.
4. Leymann F. and Roller D. Production Workflow: Concepts and Techniques. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.
5. Reijers H. Design and Control of Workflow Processes: Business Process Management for the Service Industry. LNCS 2617. Springer, Berlin Heidelberg New York, 2003.
6. van der Aalst W.M.P. Business process management demystified: a tutorial on models, systems and standards for workflow management. In Lectures on Concurrency and Petri Nets, J. Desel, W. Reisig, G. Rozenberg (eds.). LNCS 3098. Springer, Berlin Heidelberg New York, 2004, pp. 1–65.
7. van der Aalst W.M.P. and van Hee K.M. Workflow Management: Models, Methods, and Systems. MIT, Cambridge, MA, 2004.
8. Weske M. Business Process Management: Concepts, Languages, Architectures. Springer, Berlin Heidelberg New York, 2007.
9. zur Muehlen M. Workflow-Based Process Controlling: Foundation, Design and Application of Workflow-Driven Process Information Systems. Logos, Berlin, 2004.

Business Process Model

► Workflow Model

Business Process Modeling

► Workflow Modeling

Business Process Modeling Notation

W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven,
The Netherlands

Synonyms

BPMN

Definition

The Business Process Modeling Notation (BPMN) is a graphical notation for drawing business processes. It is proposed as a standard notation for drawing models understandable by different business users. BPMN aims to bridge the communication gap that frequently occurs between business process design and implementation. The language is similar to other informal notations such as UML activity diagrams and extended event-driven process chains.

Key Points

BPMN was initially developed by Business Process Management Initiative (BPMI). It is now being maintained by the Object Management Group (OMG) who released a “Final Adopted Specification” in 2006 [3]. The intent of BPMN is to standardize a business process modeling notation in the face of many different modeling notations and viewpoints.

A model expressed in terms of BPMN is called a Business Process Diagram (BPD). A BPD is essentially a flowchart composed of different elements. There are four basic categories of elements: (i) *Flow Objects*, (ii) *Connecting Objects*, (iii) *Swimlanes*, and (iv) *Artifacts* [3]. Flow objects are the main graphical elements to define the behavior of a process. There are three types of flow objects: *Events*, *Activities*, and *Gateways*. Events are comparable to places in a Petri net, i.e., they are used to trigger and/or connect activities. There are different types of activities. Atomic activities are referred to as tasks. Gateways are used to model splits and joins. The flow objects can be connected to establish a control-flow. Swimlanes are just a means to structure processes. Artifacts are used to add data or to further annotate process models.

Figure 1 shows the basic set of symbols used by BPMN. These symbols can be combined to construct a BPD. On the left hand side, four types of events are shown. Three type of splits are shown. The data-based XOR gateway split passes control to exactly one of its output arcs. The parallel fork gateway passes control to all output arcs. The event-based XOR gateway selects one output arc based on the occurrence of the corresponding event. Note that Fig. 1 shows only a subset of all possible notations. The complete language is rather complex. The specification itself [3] is 308 pages without providing any formal semantics.

BPMN is an informal language aiming at the communication and not directly at execution. Therefore, the language is positioned as the design language for BPEL, i.e., BPMN diagrams are gradually refined into BPEL specifications. Given the lack of formal semantics this is not a trivial task [2,4]. Therefore, several attempts have been made to provide semantics for a subset of BPMN [1].

Cross-references

- ▶ BPEL
- ▶ Business Process Management
- ▶ Composition
- ▶ Orchestration
- ▶ Petri Nets
- ▶ Web Services
- ▶ Workflow Management
- ▶ Workflow Patterns

Recommended Reading

1. Weske M. Business Process Management: Concepts, Languages, Architectures. Springer, Berlin Heidelberg New York, 2007.
2. White S. Using BPMN to model a BPEL process. BPTrends, 3(3):1–18, March 2005.
3. White S.A. et al. Business Process Modeling Notation Specification (Version 1.0, OMG Final Adopted Specification), 2006.
4. Wohed P., van der Aalst W.W.P., Dumas M., ter Hofstede A.H.M., and Russell N. On the Suitability of BPMN for Business Process Modelling. In Proc. Int. Conf. Business Process Management 2006, pp. 161–176.

Business Process Monitoring

▶ Event-Driven Business Process Management

Business Process Optimization

▶ Process Optimization

Business Process Redesign

▶ Business Process Reengineering

Business Process Reengineering

CHIARA FRANCALANCI
Politecnico di Milano University, Milan, Italy

Synonyms

Business Process Redesign

Definition

Business process reengineering refers to a substantial change of a company's organizational processes that (i) is enabled by the implementation of new information technologies that were not previously used by the company, (ii) takes an interfunctional (or interorganizational) perspective, i.e., involves multiple organizational functions (or organizations) that cooperate along processes (iii) considers end-to-end processes, i. e., processes that deliver a service to a company's customers, (iv) emphasizes the integration of information and related information technologies to obtain seamless technological support along processes.

Historical Background

In 1993, Hammer and Champy [3] introduced the concept of business process reengineering as a radical and fast change of organizational processes that leverages information technology. Their work was the first of a wave of research contributions that analyzed the role and impact of information technologies in business process reengineering. This research was accompanied by a widespread use of the term business process reengineering in the industry to indicate large projects that involved the implementation of client-server architectures and the concurrent redesign of core business processes. Client-server enabled a more extensive sharing of organizational data within organizations by making server data accessible to personal computers through more user friendly client applications. In turn, this enabled a redesign of end-to-end business processes towards greater interfunctional cooperation. These changes primarily involved service companies, such as insurance companies, banks, research institutions, and so on, since client server had much broader application in service industries due to a more widespread use of personal computers by all employees and in all activities, both clerical and operating. More recently, the term has been extended to indicate any substantial redesign of business processes that is enabled by information technologies.

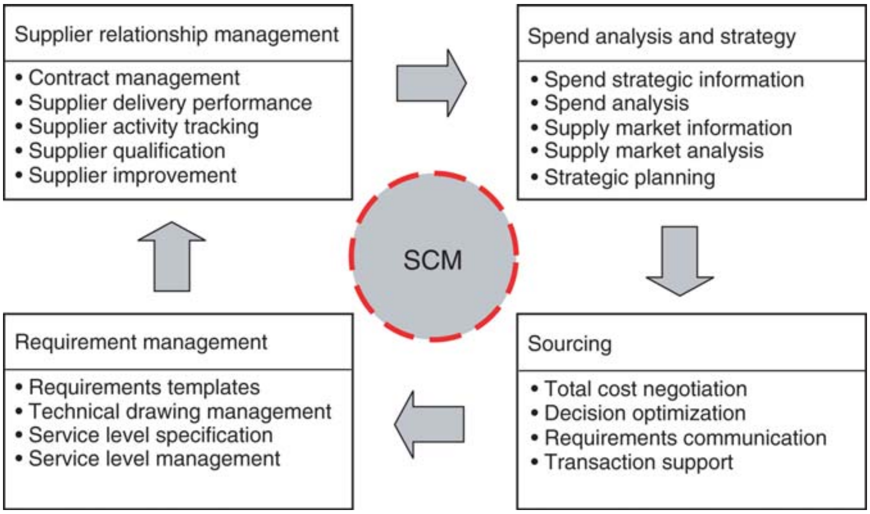
Foundations

The interfunctional integration of business processes has been recognized as a fundamental lever to improve organizational performance ever since the early studies within the information perspective of organizational theory. Going back to Galbraith's analysis [5], two complementary methods for increasing the information processing capacity of an organization are proposed. The first improves the communication of information within organizations along hierarchies of authority through *vertical* information systems. Information is gathered from lower hierarchical levels, consolidated, and conveyed to higher decision-making centers. While very effective for routine decision making, vertical information systems can quickly become overloaded in conditions of increasing uncertainty. In this case, firms can resort to direct *lateral* communication, Galbraith's second method. Liaison roles, task forces, project teams, and matrix structures are increasingly powerful mechanisms to facilitate this lateral interchange of information. While the first method reinforces the communication of information across levels of authority, the second consists of a set of organizational solutions allowing information exchanges orthogonal to the hierarchy.

Solutions for direct lateral communication permit higher organizational efficiency when hierarchies become inadequate information processors because of increasing coordination needs. This happens when dependencies between agents are difficult to foresee, as they continually change over time. In principle, within a hierarchy, lateral communication can take place only through levels of authority. Any two agents communicating through hierarchical levels experience an efficiency that is inversely proportional to their distance in the hierarchy. If dependencies between agents are stable and cause recurring paths of lateral communication, hierarchies of authority can be built by grouping agents who require most frequent interaction. By minimizing the distance between interacting agents, hierarchical coordination can be efficient. On the contrary, when organizations operate in conditions of high uncertainty, lateral information exchange becomes essential. Since demands for responsiveness and flexibility in today's business environment are raised as a result of increased uncertainty, the lateral exchange of information becomes critical to support overall information processing needs. Business process reengineering involves the redesign of organizational processes towards a higher degree of lateral communication.

Traditional Intra-Organizational Reengineering

Historically, mainframe-based architectures could support the implementation of vertical information systems, but were inherently inadequate for lateral communication. In centralized architectures, no distinction was made between data and applications. Data belonged to the application creating them and could be accessed only through that same application. Consistent with Galbraith's recommendations for the use of IT to support vertical communication, design methodologies witnessed a focus on the hierarchical conception and implementation of information systems. For example, the Normative Application Portfolio [9]


Supplier relationship management
• Contract management
• Supplier delivery performance
• Supplier activity tracking
• Supplier qualification
• Supplier improvement
Spend analysis and strategy
• Spend strategic information
• Spend analysis
• Supply market information
• Supply market analysis
• Strategic planning
SCM
Requirement management
• Requirements templates
• Technical drawing management
• Service level specification
• Service level management
Sourcing
• Total cost negotiation
• Decision optimization
• Requirements communication
• Transaction support

Business Process Reengineering. Figure 1. The supply chain management (SCM) learning cycle.

adopted a layered view of organizations, following Anthony's hierarchical framework of planning and control systems. Three main classes of applications were distinguished, supporting operations, management, and strategy, respectively. The application portfolio was created by building vertically from one level to the next, in order to guarantee that activities on lower levels supported higher level activities.

In the mid-1970s, databases and database management systems (DBMS) introduced significant changes in the design of IT architectures. Based on database technology, a new type of IT architecture was implemented, which can be broadly categorized as *centralized*. Centralized architectures allow the *logical separation* between data and applications. The layer of software services constituting the DBMS logically separated data and applications and permitted their independent design. Data common to different applications could be designed and managed as a unified resource. Database management systems exported data manipulation services that could be accessed by any application.

The management of data as a unified resource favored information sharing by integrating data common to different applications. Users running different applications could exchange information by storing and retrieving data in the central database. Conversely, designers could conceive new applications taking advantage of previously gathered and integrated information. A classical example is the use of accounting data by financial applications, allowing more precise financial analyses through detailed information on a firm's cash flows. Likewise, replenishment could be optimized through integration with order fulfillment data.

In the 1990s, client-server architectures allowed the *physical separation* between data and applications. Unlike the logical separation provided by databases in centralized architectures, the physical separation allowed the storage of data and the execution of applications on any computer. Within a client-server environment, shared data were typically stored on the server, but applications could be stored and run on local servers or personal computers.

The DBMS layer of centralized architectures was complemented by an additional layer of network services achieving the physical separation between data and applications. This made the number and the location of resources transparent to individual nodes in the architecture. By relying on peripheral processing and storage capacities of individual nodes, distributed architectures could grow incrementally. This provided the flexibility to continuously adjust to requirements and to implement a variety of applications according to individual needs. This greater flexibility enabled lateral organizational solutions that allowed different functions to take full advantage of organizational information with a variety of functionalities that could be more easily adjusted to changing requirements. These functionalities are now incorporated inside ERP (Enterprise Resource Planning) systems representing fully integrated software solutions that embed laterally integrated organizational processes.

Supply Chain Management Process Reengineering

More recently, the Web service paradigm is shifting reengineering activities towards interorganizational and interpersonal processes. Web service platforms are designed to wrap a company's information system and make selected functionalities available as web services to both internal and external users. This opens up a number of new opportunities. First, the Web service paradigm is causing a radical redesign of supply chain management processes (see Fig. 1). Supply chain management applications (SCM) support the integration of suppliers into a company's information system. This integration allows concurrent and efficient planning of production activities along the value chain. SCM involve a learning process, as shown in Fig. 1. This learning process starts from the monitoring of suppliers to measure their performance and, hence, optimize procurement activities. Then, a subset of efficient and reliable suppliers is selected for tighter integration, ranging from electronic orders to requirement management and electronic requirement management and codesign. If the supplier management process is effective, a company can build and evaluation and qualification system that can lead to official certifications that suppliers themselves can leverage as part of their brand equity. Overall, SCM involves a deep reengineering of supply management processes that represents the objective of a number of current projects.

Knowledge Management Process Reengineering

Knowledge management processes constitute a second important objective of current business process reengineering activities enabled by the Web and by the more recent Web service paradigm. Knowledge management systems (KMS) are "IT-based systems developed to support and enhance the organizational processes of knowledge creation, storage/retrieval, transfer, and application" ([1]: 114). KMS span a large and complex spectrum from help desk and customer care applications to those designed to develop employee skills. Virtual communities and collaborative environments are forms of KMS and KMS can also serve as corporate knowledge repositories and maps of expertise.

The literature on Knowledge Management Systems (KMS) largely assumes that an individual's knowledge can be captured and converted into group or organization-available knowledge. When individuals do not contribute to such systems, the knowledge creation capability of the firm is adversely affected. However, there is little clarity in the information systems literature on which mechanisms are necessary for conversion to take place. Many in the information systems literature (e.g., [6]) have argued for social influences (such as culture), hierarchical authority, and/or economic incentives, all of which are external influences that rely on the broader social context outside the KMS system. An alternative view to these external influences is internal or intrinsic motivation. This view assumes that there is little that a broader context outside of the person and the person's interactions with KMS can do to enhance contributions. "Creating and sharing knowledge are intangible activities that can neither be supervised nor forced out of people. They only happen when people cooperate voluntarily" [6]. Intrinsic motivation implies that the activity is performed because of the immediate satisfaction it provides in terms of flow, self-defined goal, or obligations of personal and social identity rather than some external factor or goal. The external factors can even undermine knowledge contributions if they interfere with intrinsic motivation. Although both views are likely to play a role in knowledge sharing, the internal view is important in organizations that take on characteristics of knowledge era (or postmodern) organizations. Such organizations require a more participative and self-managing knowledge worker compared to industrial era organizations. Knowledge-era organizations are associated with increased egalitarianism among positions, increased availability of information and knowledge resources, and increased self-management of knowledge workers. Wikipedia.org represents a typical example of postmodern self-organizing KMS, in which everyone in the world can contribute spontaneously with personal expertise and where control over contributions appropriateness is not based on any hierarchical structure, but is completely peer-based.

In a postmodern organization, an employee's commitment to his or her organization results from autonomous forms of organizing and the resulting self-expression and feelings of responsibility and control of the outputs of work. With knowledge workers, where work is associated with flows of knowledge and information, rather than flows of materials, self-expression, responsibility, and control are often targeted to knowledge outputs. Knowledge workers want to share their knowledge across the organization while preserving their association with these contributions. This type of psychological attachment, or emotional connection between the person and knowledge is well documented in open source initiatives. When given a choice to require or not require attribution to one's creative works (permit others to copy, distribute, display, modify the work but only if given credit without any economic implications), individuals invariably choose the requirement of future users of their knowledge to attribute knowledge to them (http://Creative commons.org/). The importance of psychological attachment to knowledge is no less important in commercial contexts.

KMS do not necessarily harness psychological attachment between knowledge embedded in the system and the individual who is the source of that knowledge. On one side, it is risky for the organization to let knowledge reside within the minds of individuals, in a form that can easily leak across the firm's boundaries and lead to a loss of competitive advantage. On the other hand, individual employees may perceive their personal goals to be poorly served by sharing knowledge unless the system helps to manage the knowledge worker's personal attachment to knowledge and make this attachment known in relevant organizational communities. In knowledge intensive organizations, people's distinctiveness depends upon their possessed knowledge. KMS that do not help construct, communicate, and defend the psychological attachment between the knowledge and the knowledge worker, and the rest of the organization can reduce the motivation to contribute knowledge to KMS. Fostering psychological attachment is likely to increase the knowledge workers quality of contributions, not only the quantity. In fact, the more complex and tacit the knowledge contributed to KMS, the higher the effort that knowledge workers are likely to put in their contributing activities. When the KMS fosters psychological attachment to contributions, knowledge workers increase their likelihood to engage also those knowledge sharing activities that require greater effort in order to be carried out.

Process Modeling Languages and Techniques

Reengineering initiatives typically involve a process modeling phase that supports the analysis of existing processes and the specification of new processes. UML (www.uml.org) represents a quasi-standard for process modeling and is widely used in business process reengineering.

Over the years, the scope of business processes and BPM has broadened. Initially, BPM, or workflow, was a technique that helped design largely human-based, paper-driven processes within a corporate department. For example, to handle a claim, an insurance claims process, taking as input a scanned image of a paper claims form, would pass the form electronically from the mailbox (or worklist) of one claims specialist to that of another, mimicking the traditional movement of interoffice mail from desk to desk. The contemporary process orchestrates complex system interactions, and is itself a service capable of communicating and conversing with the processes of other companies according to well-defined technical contracts. A retailer's process to handle a purchase order, for example, is a service that uses XML messages to converse with the service-based processes of consumers and warehouses.

A number of new modeling languages have recently been proposed to accommodate this complexity. For example, the i* model and its subsequent developments within the TROPOS project allow the representation of strategic relationships between organizations and their relationships with goals, resources and system components (cf.[10]). The impact of the structure of cooperation forms on the interactions among actors are modeled in the field of Multi-Agent Systems to determine an architectural solution based on typical software non-functional requirements (i.e., security, integrity, modularity, etc.). Moreover, the i* model supports the analysis of high-level goals together with non-functional requirements. The ability to move from high-level goals to sub-goals is also provided by KAOS (cf.[4]). KAOS is a formal approach for analyzing goals and for transforming goals into requirements for the software system. Finally, GBRAM (Goal Based Requirements Analysis Method, cf.[2]) proposes a method supporting the initial identification of high-level goals. Different from i* and KAOS, GBRAM does not assume that high-level goals are previously identified and provides a set of strategies to elicit goals from all available sources of information.

Key Applications

Enterprise Resource Planning (ERP), Web Services (WS), Knowledge Management Systems (KMS).

Cross-references

▶ Data architectures
▶ DBMS

- ► ERP
- ► IT architectures
- ► UML
- ► Web services

Recommended Reading

1. Alavi M. and Leidner D.E. Knowledge management and knowledge management systems: Conceptual foundations and research issues. MIS Q, 25(1):107–136, 2001.
2. Anton A. Goal Identification and Refinement in the Specification of Software-based Information Systems, Ph.D. Dissertation. Georgia Institute of Technology, Atlanta, 1997.
3. Champy J. and Hammer M. Reengineering the Corporation. Harper Collins, New York, NY, 1993.
4. Dardenne A., Lamsweerde vanA., and Fickas S. Goal-directed requirements acquisition. Sci. Comput. Program., 20(1–2):3–50, 1993.
5. Galbraith J.R. Organization Design. Addison-Wesley Publishing Company, Reading, MA, 1977.
6. Jarvenpaa S.L. and Staples D.S. The use of collaborative electronic media for information sharing: An exploratory study of determinants. J. Strateg. Inform. Syst., 9(2–3):129–154, 2000.
7. Kim W.C. and Mauborgne R. Procedural justice, strategic decision making, and the knowledge economy. Strateg. Manage. J., 19:323–338, 1988.
8. Malone T.W. and Crowston K. The Interdisciplinary study of coordination. ACM Comput. Surv., 26(1):87–119, 1994.
9. Nolan R.L. Managing the Data Resource Function. West Publishing Company, St. Paul, Minnesota, MN, 1982.
10. Yu E. and Mylopoulos J. Using goal, rules and methods to support reasoning in business process reengineering. Int. J. Intell. Syst. Account. Finance Manage., 5(1):1–13, 1996.

Cache Manager

► Buffer Manager

Cache Performance

► Performance Analysis of Transaction Processing Systems

Cache-Aware Query Processing

► Cache-Conscious Query Processing

Cache-Conscious Query Processing

Kenneth A. Ross
Columbia University, New York, NY, USA

Synonyms

Cache-aware query processing; Cache-sensitive query processing

Definition

Query processing algorithms are designed to efficiently exploit the available cache units in the memory hierarchy. Cache-conscious algorithms typically employ knowledge of architectural parameters such as cache size and latency. This knowledge can be used to ensure that the algorithms have good temporal and/or spatial locality on the target platform.

Historical Background

Between 1980 and 2005, processing speeds improved by roughly four orders of magnitude, while memory speeds improved by less than a single order of magnitude. As a result, it is common (at the time of writing) for data accesses to RAM to require several hundred CPU cycles to resolve. Many database workloads have shifted from being I/O bound to being memory/CPU-bound as the amount of memory per machine has been increasing. For such workloads, improving the locality of data-intensive operations can have a direct impact on the system's overall performance.

Foundations

A cache is a hardware unit that speeds up access to data. Several cache units may be present at various levels of the memory hierarchy, depending on the processor architecture. For example, a processor may have a small but fast Level-1 (L1) cache for data, and another L1 cache for instructions. The same processor may have a larger but slower L2 cache storing both data and instructions. Some processors may even have an L3 cache. On multicore processors, the lower level caches may be shared among groups of cores.

Some initial analysis would typically be performed to determine the performance characteristics of a workload. For example, Ailamaki et al. [1] used hardware performance counters to demonstrate that several commercial systems were, at that time, suffering many L2 data cache misses and L1 instruction cache misses. Based on such observations, one can determine that the L2 data cache and L1 instruction cache are targets for performance tuning.

If the operating system does not provide direct access to system parameters such as the cache size, a database system can run a calibration test to estimate the relevant parameters [13].

To get good cache performance, algorithm designers typically utilize one or more of the following general approaches:

- Improve spatial locality, so that data items that are often accessed together are in the same cache lines.
- Improve temporal locality for data, so that after an initial cache miss, subsequent data item accesses occur while the item is still cache-resident.

- Improve temporal locality for instructions, so that code that needs to be applied to many data items is executed many times while the instructions reside in the cache.
- Hide the latency. Latency can be hidden in several ways. If the data access pattern is predictable, prefetching data elements into the cache can overlap the memory latency with other work. On architectures that support multiple simultaneous outstanding memory requests, cache miss latencies can be overlapped with one another.
- Sample the data to predict the cache behavior, and choose an algorithm accordingly.

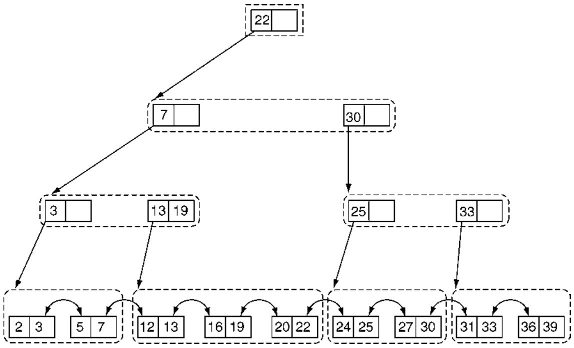
Examples of each of these approaches are given below.

Spatial Locality

In many query-processing contexts, only a few columns for a table are needed. In such cases, it pays to organize the table column-wise, so that column values from consecutive records are contiguous. Cache lines then contain many useful data elements. A row-wise organization would require more cache-line accesses, since each cache line would contain some data from unneeded columns. Examples of systems with column-wise storage are Sybase IQ [12], MonetDB [3], and C-Store [19]. The PAX storage model [6] allows for column-wise storage within each disk page. The main advantage of such an approach is that existing page-oriented database systems can improve cache behavior with limited changes to the whole system.

Chilimbi et al. [7] advocate placing multiple levels of a binary tree within a cache line, to reduce the number of cache misses per traversal. Chilimbi et al. also use cache coloring to place frequently accessed items in certain ranges of physical memory. The idea is to reduce the number of conflict misses by making sure that the low order bits of the addresses of certain items cannot be the same.

To reduce the number of cache lines needed to search for an item in an index, Rao and Ross proposed CSS-trees [16] and CSB+-trees [17]. CSS-trees eliminate pointers; a node contains only keys. Nodes are of fixed size, typically one cache line, aligned to the cache line boundaries. Nodes are stored in an array in such a way that the children of a node can be determined using simple arithmetic operations, making pointers unnecessary. CSB+-trees extend this idea, allowing just one pointer per node and requiring that all sibling nodes be contiguous. CSB+-trees have better update performance than CSS-trees, while retaining almost all of the cache-efficiency. The diagram below shows a CSB+-Tree of Order 1. Note that each node has only one child pointer and that each node's children are allocated contiguously.



Several other ways to compress B+-tree nodes for cache performance, such as key compression and key-prefix truncation, are discussed in [11].

Temporal Locality

Blocking is a general technique for ensuring temporal locality. Data is processed in cache-sized units, so that all data within the block stays cache-resident. The blocks are then combined in a later phase. Alpha-Sort [14] is an example of such a method: cache-sized units of input data are read and quick-sorted into runs. These runs are merged in a later phase. Padmanabhan et al. [15] modified a commercial database system to pass data between certain operators in blocks, and demonstrated improved cache behavior.

Buffering is a related strategy to improve temporal locality. Zhou and Ross [20] propose buffering to speed up bulk B-tree index lookups. By sending probes only one level at a time through the tree, in batches, one can amortize the cost of reading an index node over many probes. The savings in data cache misses usually outweigh the extra cost of reading and writing to intermediate buffers. Zhou and Ross also examine the code size of database operators, and propose to buffer data to ensure that the footprint of the active code is smaller than the size of the L1 instruction cache [21]. Again, the savings in instruction cache misses usually outweigh the cost of buffering.

Partitioning the data into cache-sized units for later processing is the dual of blocking. Examples include the partitioned hash join [18] and radix-join [3].

When multiple processors, or multiple threads within a single processor access a shared cache, cache interference can result. Even if each individual thread is cache-conscious, the total cache resources may be insufficient for all threads, and cache thrashing may result. To counter this interference, one could design cooperative threads that work together on a common task using common cache-conscious data structures. Multithreaded join operators [10,22] and aggregation operators [8] have been proposed.

Many divide-and-conquer style algorithms generate temporal locality at recursively smaller granularities. Such algorithms have been termed *cache oblivious* because they can achieve locality at multiple levels of the memory hierarchy without explicit knowledge of the cache parameters [9].

Prefetching

Prefetching involves reading data into the cache ahead of when it is to be used. When access patterns can be predicted in advance, and when memory bandwidth is not saturated, prefetching can effectively hide the memory latency. Some hardware platforms automatically recognize certain access patterns, such as regular fixed-stride access to memory. The hardware then automatically prefetches ahead in the access sequence. For access patterns that are not so easily recognized, or for hardware platforms that do not support hardware prefetching, one can explicitly prefetch memory locations using software.

Chen et al. [6] show how to prefetch parts of a B+-tree node or CSB+-tree node to get a bigger effective node size. For example, if the memory system can support n outstanding memory requests, then a node consisting of n cache lines could be retrieved in only slightly more time than a single cache line. Since wider nodes result in shallower trees, the optimal node size might be several cache lines wide.

Chen et al. [4] use prefetching to speed up hash joins. The internal steps for processing records are divided into stages. A memory access is typically required between stages. Stages for multiple records are scheduled so that while data for a forthcoming operation is being prefetched, useful work is being performed on other records.

Zhou et al. [22] define the notion of a work-ahead set, a data structure that describes a memory location and a computation stage for some data-intensive operation. One thread of a two-threaded system is devoted purely to prefetching the data into the cache, while the other thread does the algorithmic work.

Sampling

Inspector joins sample the data during an initial partitioning phase [5]. This information is used to accelerate a cache-optimized join algorithm for processing the partitions. Cieslewicz et al. [8] sample a stream of tuples for aggregation to estimate (among other things) the locality of reference of group-by values. Based on that information, an appropriate aggregation algorithm is chosen for the remainder of the stream.

Key Applications

Data intensive operators such as sorts, joins, aggregates, and index lookups, form the "assembly language" into which complex queries are compiled. By making these operators as efficient as possible on modern hardware, all database system users can effectively exploit the available resources.

Future Directions

Future processors are likely to scale by placing many cores on a chip, with only a modest increase in clock frequency. As a result, the amount of cache memory per processor may actually decrease over time, making cache optimization even more critical. For chips with shared caches, interference between cores will be a significant performance hazard. While locality is good for cache behavior, it can be bad for concurrency due to hot-spots of contention [8]. Cache performance will need to be considered together with parallelism to find appropriate performance trade-offs.

Cross-references

- ► Architecture-Conscious Database System
- ► Cache-Conscious Transaction Processing

Recommended Reading

1. Ailamaki A., Dewitt D.J., Hill M.D., and Wood D.A. DBMSs on a modern processor: where does time go? In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 266–277.
2. Ailamaki A., DeWitt D.J., Hill M.D., and Skounakis M. Weaving relations for cache performance. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 169–180.
3. Boncz P.A., Manegold S., and Kersten M.L. Database architecture optimized for the new bottleneck: memory access. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 54–65.
4. Chen S., Ailamaki A., Gibbons P.B., and Mowry T.C. Improving hash join performance through prefetching. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 116–127.
5. Chen S. et al. Inspector joins. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 817–828.
6. Chen S., Gibbons P.B., and Mowry T.C. Improving index performance through prefetching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 235–246.
7. Chilimbi T.M., Hill M.D., and Larus J.R. Cache-conscious structure layout. In Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation, 1999, pp. 1–12.
8. Cieslewicz J. and Ross K.A. Adaptive aggregation on chip multiprocessors. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 339–350.

9. Frigo M., Leiserson C.E., Prokop H., and Ramachandran S. Cache-oblivious algorithms. In Proc. 40th Annual Symp. on Foundations of Computer Science, 1999, pp. 285–298.
10. Garcia P. and Korth H. Database hash-join algorithms on multithreaded computer architectures. In Proc. 3rd Conf. on Computing Frontiers, 2006, pp. 241–251.
11. Graefe G. and Larson P. B-tree indexes and CPU caches. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 349–358.
12. MacNicol R. and French B. Sybase IQ multiplex – designed for analytics. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 1227–1230.
13. Manegold S., Boncz P.A., and Kersten M.L. What happens during a join? dissecting CPU and memory optimization Effects. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 339–350.
14. Nyberg C., Barclay T., Cvetanovic Z., Gray J., and Lomet D.B. AlphaSort: a cache-sensitive parallel external sort. VLDB J., 4 (4):603–627, 1995.
15. Padmanabhan S., Malkemus T., Agarwal R., and Jhingran A. Block oriented processing of relational database operations in modern computer architectures. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 567–574.
16. Rao J. and Ross K.A. Cache conscious indexing for decision-support in main memory. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 78–89.
17. Rao J. and Ross K.A. Making B+ trees cache conscious in main memory. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 475–486.
18. Shatdal A., Kant C., and Naughton J.F. Cache conscious algorithms for relational query processing. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 510–521.
19. Stonebraker M., Abadi D.J., Batkin A., Chen X., Cherniack M., Ferreira M., Lau E., Lin A., Madden S., O'Neil E.J., O'Neil P.E., Rasin A., Tran N., and Zdonik S.B. C-Store: a column-oriented DBMS. In Proc. 31th Int. Conf. on Very Large Data Bases, 2005, pp. 553–654.
20. Zhou J. and Ross K.A. Buffering accesses to memory-resident index structures. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 405–416.
21. Zhou J. and Ross K.A. Buffering database operations for enhanced instruction cache performance. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 191–202.
22. Zhou J., Cieslewicz J., Ross K., and Shah M. Improving database performance on simultaneous multithreading processors. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 49–60.

Cache-Sensitive Query Processing

▶ Cache-Conscious Query Processing

Calculus Expression

▶ Comprehensions

Calendar

Christian S. Jensen[1], Richard Snodgrass[2]

[1]Aalborg University, Aalborg, Denmark

[2]University of Arizona, Tucson, AZ, USA

Definition

A *calendar* provides a human interpretation of time. As such, calendars ascribe meaning to temporal values such that the particular meaning or interpretation provided is relevant to its users. In particular, calendars determine the mapping between human-meaningful time values and an underlying time line.

Key Points

Calendars are most often cyclic, allowing human-meaningful time values to be expressed succinctly. For example, dates in the common Gregorian calendar may be expressed in the form <*month*, *day*, *year*> where the month and day fields cycle as time passes.

The concept of calendar defined here subsumes commonly used calendars such as the Gregorian calendar, the Hebrew calendar, and the Lunar calendar, though the given definition is much more general. This usage is consistent with the conventional English meaning of the word.

Dershowitz and Reingold's book presents complete algorithms for fourteen prominent calendars: the present civil calendar (Gregorian), the recent ISO commercial calendar, the old civil calendar (Julian), the Coptic an Ethiopic calendars, the Islamic (Muslim) calendar, the modern Persian (solar) calendar, the Bahá'í calendar, the Hebrew (Jewish) calendar, the Mayan calendars, the French Revolutionary calendar, the Chinese calendar, and both the old (mean) and new (true) Hindu (Indian) calendars. One could also envision more specific calendars, such as an academic calendar particular to a school, or a fiscal calendar particular to a company.

Cross-references

▶ Calendric System

▶ SQL

▶ Temporal Database

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A Glossary of Time Granularity Concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS, vol. 1399, Springer, Berlin, pp. 406–413, 1998.

2. Dershowitz N. and Reingold E.M. Calendrical Calculations, Cambridge, 1977.
3. Jensen C.S. and Dyreson C.E. (eds.). Böhlen M., Clifford J., Elmasri R., Gadia S.K., Grandi F., Hayes P., Jajodia S., Käfer W., Kline N., Lorentzos N., Mitsopoulos Y., Montanari A., Nonen D., Peressi E., Pernici B., Roddick J.F., Sarda N.L., Scalas M.R., Segev A., Snodgrass R.T., Soo M.D., Tansel A., Tiberio R. and Wiederhold G. A Consensus Glossary of Temporal Database Concepts – February 1998 Version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS, vol. 1399, Springer, Berlin, 1998, pp. 367–405.
4. Urgun B., Dyreson C.E., Snodgrass R.T., Miller J.K., Kline N., Soo M.D., and Jensen C.S. Integrating Multiple Calendars using τZaman. Software Pract. Exper., 37(3):267–308, 2007.

Calendric System

Curtis E. Dyreson[1], Christian S. Jensen[2], Richard Snodgrass[3]
[1]Utah State University, Logan, UT, USA
[2]Aalborg University, Aalborg, Denmark
[3]University of Arizona, Tucson, AZ, USA

Definition

A calendric system is a collection of calendars. The calendars in a calendric system are defined over contiguous and non-overlapping intervals of an underlying time-line. Calendric systems define the human interpretation of time for a particular locale as different calendars may be employed during different intervals.

Key Points

A calendric system is the abstraction of time available at the conceptual and logical (query language) levels. As an example, a Russian calendric system could be constructed by considering the sequence of six different calendars used in that region of the world. In prehistoric epochs, the Geologic calendar and Carbon-14 dating (another form of calendar) are used to measure time. Later, during the Roman empire, the lunar calendar developed by the Roman republic was used. Pope Julius, in the first century B.C., introduced a solar calendar, the Julian calendar. This calendar was in use until the 1917 Bolshevik revolution when the Gregorian calendar, first introduced by Pope Gregory XIII in 1572, was adopted. In 1929, the Soviets introduced a continuous schedule work week based on 4 days of work followed by 1 day of rest, in an attempt to break tradition with the 7-day week. This new calendar, the Communist calendar, had the failing that only eighty percent of the work force was active on any day, and it was abandoned after only 2 years in favor of the Gregorian calendar, which is still in use today.

The term "calendric system" has been used to describe the calculation of events within a single calendar. However, the given definition generalizes that usage to multiple calendars in a very natural way.

Cross-references

- ► Calendar
- ► Temporal Database
- ► Time Interval

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). Böhlen M., Clifford J., Elmasri R., Gadia S.K., Grandi F., Hayes P., Jajodia S., Käfer W., Kline N., Lorentzos N., Mitsopoulos Y., Montanari A., Nonen D., Peressi E., Pernici B., Roddick J.F., Sarda N.L., Scalas M.R., Segev A., Snodgrass R.T., Soo M.D., Tansel A., Tiberio R. and Wiederhold G., A Consensus Glossary of Temporal Database Concepts – February 1998 Version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS, vol. 1399, Springer, Berlin, 1998, pp. 367–405.

Camera Break Detection

► Video Shot Detection

Capsule

► Snippet

Cardinal Direction Relationships

Spiros Skiadopoulos
University of Peloponnese, Tripoli, Greece

Synonyms

Orientation relationships; Directional relationships

Definition

Cardinal direction relationships are qualitative spatial relations that describe how an object is placed relative to other objects utilizing a co-ordinate system. This knowledge is expressed using symbolic (qualitative) and not numerical (quantitative) methods. For instance, *north* and *southeast* are cardinal direction relationships. Such relationships are used to describe and constrain the relative positions of objects and can be used to pose queries such as "Find all objects *a*, *b* and *c* such that *a* is *north of b* and *b* is *southeast of c*".

Historical Background

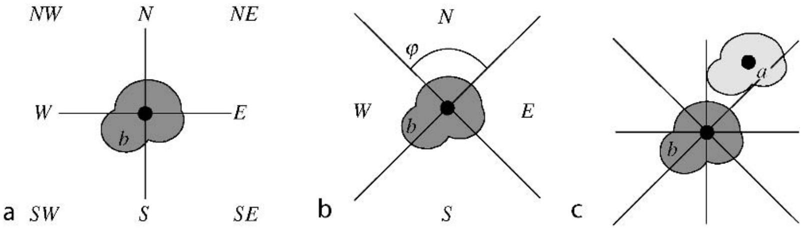
Qualitative spatial relationships (QSR) approach common sense knowledge and reasoning about space using symbolic rather than numerical methods [5]. QSR has found applications in many diverse scientific areas such as geographic information systems, artificial intelligence, databases, and multimedia. Most researchers in QSR have concentrated on the three main aspects of space, namely topology, distance and direction. The uttermost aim in these lines of research is to define new and more expressive categories of spatial operators, as well as to build efficient algorithms for the automatic processing of queries involving these operators.

Foundations

Several models capturing cardinal direction relationships have been proposed in the literature. Typically, a cardinal direction relationship is a binary relation that describes how a *primary object a* is placed relative to a *reference object b* utilizing a co-ordinate system (e.g., object *a* is *north of* object *b*). Early models for cardinal direction relationships approximate an extended spatial object by a representative point [3,6], e.g., objects in Fig. 1 are approximated by their centroid. Typically, such models partition the space around the reference object *b* into a number of mutually exclusive areas. For instance, the *projection* model partitions the space using lines parallel to the axes (Fig. 1a) while the *cone* model partitions the space using lines with an origin angle ϕ (Fig. 1b). Depending on the adopted model, the relation between two objects may change. For instance, consider Fig. 1c. According to the projection model, *a* is northeast of *b* while according to the cone model, *a* is north of *b*. Point based approximations may be crude [4], thus, later models more finely approximate an object using a representative area (most commonly the minimum bounding box (The minimum bounding box of an object *a* is the smallest rectangle, aligned with the axis, that encloses *a*.)) and express directions on these approximations [7,9]. Unfortunately, even with finer approximations, models that approximate both the primary and the reference object may give misleading directional relations when objects are overlapping, intertwined, or horseshoe-shaped [4].

Recently, more precise models for cardinal direction relationships have been proposed. Such models define directions on the exact shape of the primary object and only approximate the reference object (using its minimum bounding box). The *projection-based directional relations* ($\mathcal{PDR}$) model is the first model of this category [4,11,12]. The $\mathcal{PDR}$ model partitions the plane around the reference object into nine areas similarly to the projection model (Fig. 2a). These areas correspond to the minimum bounding box (*B*) and the eight cardinal directions. Intuitively, the cardinal direction relationship is characterized by the names of the reference areas occupied by the primary object. For instance, in Fig. 2b, object *a* is partly *NE* and partly *E* of object *b*. This is denoted by *a NE*:*E b*. Similarly in Fig. 2c, *a B*:*S*:*SW*:*W*:*NW*:*N*:*E*:*SE b* holds. In total, the $\mathcal{PDR}$ model identifies 511 (= $2^9 - 1$) relationships.

Clearly, the $\mathcal{PDR}$ model offers a more precise and expressive model than previous approaches that approximate objects using points or rectangles [4]. The $\mathcal{PDR}$ model adopts a projection-based partition using lines parallel to the axes (Fig. 2a). Typically, most people find it more natural to organize the surrounding space using lines with an origin angle similarly to the cone model (Fig. 3a). This partition of space is adopted by the *cone-based directional relations* ($\mathcal{CDR}$) model. Similarly to the $\mathcal{PDR}$ model, the $\mathcal{CDR}$ model uses the exact shape of the primary object and only approximates the reference object using its minimum bounding box

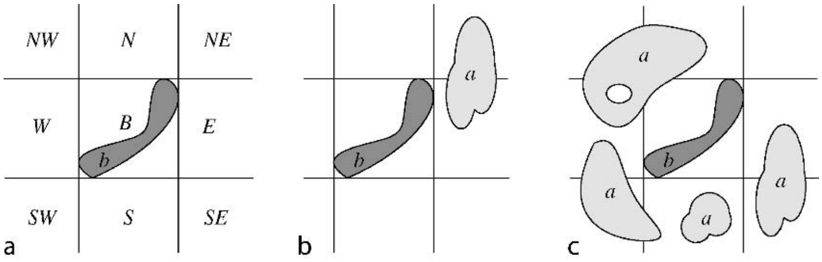


Cardinal Direction Relationships. Figure 1. Projection-based and cone-based point models.

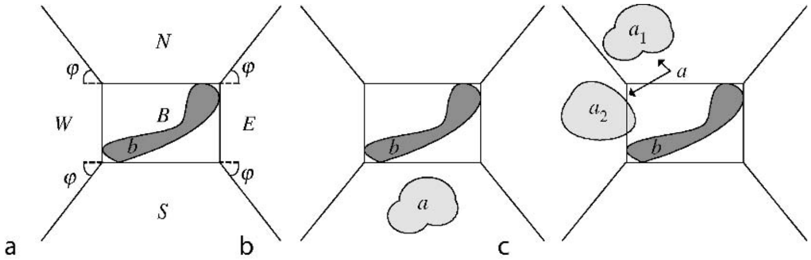
[14]. But, for the $\mathcal{CDR}$ model the space around the reference object is partitioned into five areas (Fig. 3a). The cardinal direction relationship is formed by the areas that the primary object falls in. For instance, in Fig. 3b, *a* is south of *b*. This is denoted by *a S b*. Similarly in Fig. 3c, *a B*:*W*:*N b* holds. In total, the $\mathcal{CDR}$ model identifies 31 ($= 2^5 - 1$) relationships.

In another line of research, cardinal direction relationships are modeled as ternary relationships [2]. Given three objects *a*, *b* and *c*, the ternary model expresses the direction relation of the primary object *a* with respect to a reference frame constructed by objects *b* and *c*. Specifically, the convex-hull, the internal and the external tangents of objects *b* and *c* divide the space into five areas as in Fig. 4a. These areas correspond to the following directions: right-side (*RS*), before (*BF*), left-side (*LS*), after (*AF*) and between (*BT*). Similarly to $\mathcal{PDR}$ and $\mathcal{CDR}$, the name of the areas that *a* falls into, determines the relation. For instance, in Fig. 4b, *a* is before and to the left-side of *b* and *c*. This is denoted by *LS*:*BF*(*a*,*b*,*c*). Notice that, if the order of the reference objects changes, the relationship also changes. For instance, in Fig. 4b, *RS*:*AF*(*a*,*c*,*b*) also holds.

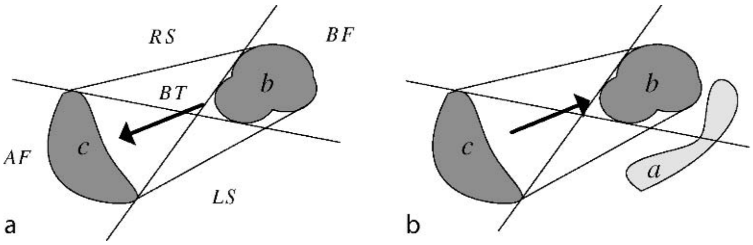
For all the above models of cardinal direction relationships, research has focused on four interesting operators: (i) efficiently determining the relationships that hold between a set of objects, (ii) calculating the inverse of a relationship, (iii) computing the composition of two relationships and (iv) checking the consistency of a set of relationships. These operators are used as mechanisms that compute and infer cardinal direction relations. Such mechanisms are important as they are in the heart of any system that retrieves collections of objects similarly related to each other using spatial relations. Table 1 summarizes current research on the aforementioned problems.



Cardinal Direction Relationships. Figure 2. Extending the projection model.



Cardinal Direction Relationships. Figure 3. Extending the cone model.



Cardinal Direction Relationships. Figure 4. Ternary cardinal direction relationships.

Cardinal Direction Relationships. Table 1. Operations for cardinal direction relationships

Model	Computation	Inverse	Composition	Consistency
Point approximations	[10]	[6]	[6]	[6]
Rectangle approximations	[9]	[9]	[9]	[9]
$\mathcal{PDR}$	[13]	[1]	[11]	[8, 12]
$\mathcal{CDR}$	Open problem	[14]	[14]	Open problem
Ternary	[2]	[2]	Open problem	Open problem

Key Applications

Cardinal direction relationships intuitively describe the relative position of objects and can be used to constrain and query spatial configurations. This information is very useful in several applications like geographic information systems, spatial databases, spatial arrangement and planning, etc.

Future Directions

There are several open and important problems concerning cardinal direction relations. For the models discussed in the previous section, as presented in Table 1, there are four operators that have not been studied (two for the $\mathcal{CDR}$ and two for the ternary model). Another open issue is the integration of cardinal direction relationships with existing spatial query answering algorithms and data indexing structures (like the R-tree). Finally, with respect to the modeling aspect, even the most expressive cardinal direction relationships models define directions by approximating the reference objects. Currently, there is not a simple and easy to use model that defines cardinal direction relationships on the exact shape of the involved objects.

Cross-references

▶ Geographic Information System
▶ Spatial Operations and Map Operations
▶ Topological Relationships

Recommended Reading

1. Cicerone S. and Di Felice P. Cardinal directions between spatial objects: the pairwise-consistency problem. Inf. Sci., 164 (1–4):165–188, 2004.
2. Clementini E. and Billen R. Modeling and computing ternary projective relations between regions. IEEE Trans. Knowl. Data Eng., 18(6):799–814, 2006.
3. Freksa C. Using orientation information for qualitative spatial reasoning. In Proceedings of COSIT'92, LNCS, vol. 639, 1992, pp. 162–178.
4. Goyal R. Similarity Assessment for Cardinal Directions Between Extended Spatial Objects. PhD Thesis, Department of Spatial Information Science and Engineering, University of Maine, April 2000.
5. Hernández D. Qualitative Representation of Spatial Knowledge, LNCS, vol. 804. Springer, Berlin, 1994.
6. Ligozat G. Reasoning about cardinal directions. J. Visual Lang. Comput., 9:23–44, 1998.
7. Mukerjee A. and Joe G. A qualitative model for space. In Proc. 7th National Conf. on AI, 1990, pp. 721–727.
8. Navarrete I., Morales A., and Sciavicco G. Consistency checking of basic cardinal constraints over connected regions. In Proc. 20th Int. Joint Conf. on AI, 2007, pp. 495–500.
9. Papadias D. Relation-based representation of spatial knowledge. PhD Thesis, Department of Electrical and Computer Engineering, National Technical University of Athens, 1994.
10. Peuquet D.J. and Ci-Xiang Z. An algorithm to determine the directional relationship between arbitrarily-shaped polygons in the plane. Pattern Recognit, 20(1):65–74, 1987.
11. Skiadopoulos S. and Koubarakis M. Composing cardinal direction relations. Artif. Intell., 152(2):143–171, 2004.
12. Skiadopoulos S. and Koubarakis M. On the consistency of cardinal directions constraints. Artif. Intell., 163(1):91–135, 2005.
13. Skiadopoulos S., Giannoukos C., Sarkas N., Vassiliadis P., Sellis T., and Koubarakis M. Computing and managing cardinal direction relations. IEEE Trans. Knowl. Data Eng., 17 (12):1610–1623, 2005.
14. Skiadopoulos S., Sarkas N., Sellis T., and Koubarakis M. A family of directional relation models for extended objects. IEEE Trans. Knowl. Data Eng., 19(8):1116–1130, 2007.

Cartesian Product

Cristina Sirangelo
University of Edinburgh, Edinburgh, UK

Synonyms

Cross product

Definition

Given two relation instances R_1, over set of attributes U_1, and R_2, over set of attributes U_2 – with U_1 and U_2 disjoint – the cartesian product $R_1 \times R_2$ returns a new relation, over set of attributes $U_1 \cup U_2$, consisting of tuples $\{t | t(U_1) \in R_1 \text{ and } t(U_2) \in R_2\}$. Here $t(U)$ denotes the restriction of the tuple t to attributes in the set U.

Key Points

The cartesian product is an operator of the relational algebra which extends to relations the usual notion of cartesian product of sets.

Since the sets of attributes of the input relations are disjoint, in $R_1 \times R_2$ each tuple of R_1 is combined with each tuple of R_2; moreover the arity of the output relation is the sum of the arities of R_1 and R_2.

As an example, consider a relation *Students* over attributes (*student-number, student-name*), containing tuples {(1001, *Black*), (1002, *White*)}, and a relation *Courses* over attributes (*course-number, course-name*), containing tuples {(*EH*1, *Databases*), (*GH*5, *Logic*)}. Then *Students* × *Courses* is a relation over attributes (*student-number, student-name, course-number, course-name*) containing tuples (1001, *Black*, *EH*1, *Databases*), (1001, *Black*, *GH*5, *Logic*), (1002, *White*, *EH*1, *Databases*), (1002, *White*, *GH*5, *Logic*).

The cartesian product can also be viewed as a special case of natural join, arising when the set of attributes of the operands are disjoint. However, relations over non-disjoint sets of attributes can also be combined by the cartesian product, provided that the renaming operator is used to rename common attributes in one of the two relations.

In the presence of attribute names, the cartesian product is commutative. In the case that relation schemas do not come with attribute names, but are specified by a relation name and arity, the cartesian product $R_1 \times R_2$ returns the concatenation $t_1 t_2$ of all pairs of tuples such that $t_1 \in R_1$ and $t_2 \in R_2$. Moreover the output schema is specified by the sum of the arities of the input schemas. In this case the cartesian product is a non-commutative operator.

Cross-references

- ▶ Join
- ▶ Relation
- ▶ Relational Algebra
- ▶ Renaming

Cartography

▶ Visual Interfaces for Geographic Data

CAS

▶ Storage Security

CAS Query

▶ Content-and-Structure Query

Case Handling

▶ Business Process Management

Case Management

▶ Workflow Management

Case Report Forms

▶ Clinical Data Acquisition, Storage and Management

Cataloging

▶ Cataloging in Digital Libraries

Cataloging in Digital Libraries

Mary Lynette Larsgaard
University of California-Santa Barbara, Santa Barbara, CA, USA

Synonyms

Cataloging; Classification

Definition

Cataloging is using standard rules to create a mainly text surrogate that describes an object sufficiently in detail so that the object is uniquely differentiated from all other objects. Without looking at the object, a user may know enough about the object to know if it suits the user's needs. It is generally considered to include bibliographic description, and the application of subjects, both as words and as classification.

Historical Background

Devising and using methods of arranging and describing information – respectively termed, within the standard library world, classification and cataloging – have been primary concerns of libraries ever since libraries began, in the ancient world of the Greeks and the Romans. A collection of information without classification and cataloging is not a library. The whole point of classification and cataloging is to make access quick and easy for users; it was discovered very early that putting like objects together (classification) and creating text or relatively speaking much smaller surrogates to describe an information object (cataloging) made finding information much quicker for the user.

Experiments in using digital records in libraries started in approximately the late 1960s. But it was only in the mid-1970s, with the success of what is called "shared cataloging" – many libraries using a catalog record contributed as "original cataloging" by the first library to catalog the item – that using digital systems for cataloging came into its own. This sharing of bibliographic records in online form began with OCLC, initially as a consortium of college libraries in Ohio (starting in 1967), but growing rapidly to become the most successful such library utility, currently with about 60,000 participating libraries in 112 countries and territories (http://www.oclc.org). The development of integrated library systems (ILS) or library management systems (LMS) – software, or a combination of software and hardware, that permits a library to perform multiple functions, such as acquisitions/ordering, cataloging and classification, circulation, preservation, and reference, using digital files in large databases with many tables – has continued to the present in the "Library of Congress Authorities" online authority system, http://authorities.loc.gov).

The inception and speedy growth of the World Wide Web (Web) since the mid-1990s has spread this interest in arrangement and description of, and access to, information objects to non-library communities, and within the library world to how specifically to arrange and describe digital objects made available over the Web in digital libraries. There have been many standards for the description of information objects. They are most often intended either to apply at least in theory to all materials (e.g., Anglo-American Cataloging Rules, hereafter referred to as AACR; Dublin Core, which began in 1995, http://www.dublincore.org/documents/dces/) or to apply to the description of a specific body of information (e.g., for digital geospatial data, "Content Standard for Digital Geospatial Metadata"; 2nd edition, 1998, http://www.fgdc.gov/standards/projects/FGDC-standards-projects/metadata/base-metadata/v2_0698.pdf and ISO Standard 19115, "Geographic Information, metadata; Information géographique, métadonnées," 2003).

Foundations

Classification

Classification and cataloging complement each other. Classification is a form of subject cataloging, which is where the major overlap between the two occurs. Classification tends to be hierarchical, breaking a given world of information or knowledge into broad divisions (e.g., Law) and then breaking that into smaller divisions (e.g., education for law; law of various countries; etc.). Classification is most often placing like items about like subjects (e.g., works by Shakespeare) and like genres or formats (e.g., maps) together. It also provides a physical location within a library for each object, be it digital or hardcopy. The most prominent systems used are the Dewey Decimal Classification (often used by public libraries and smaller libraries generally), and the Library of Congress Classification (most often used by university and other research libraries in the United States). There are many more systems, such as the Bliss Classification and the Colon Classification, and special and research libraries (such as the New York Public Library) devise and maintain their own systems. Devising a classification system is easy, but maintaining it is very difficult and time-consuming. Often a library maintaining a classification system unique to itself will find that it is far less work to convert to a rigorously maintained system that is used by many libraries than it is to maintain a unique system not used by any other collections.

Classification of digital objects at first glance seems unnecessary. Why not just assign an arbitrary number (e.g., a unique identifier that database software assigns to each separate catalog record) and be done with it? Libraries of digital objects have found that, for several reasons, it is very practical to assign classification to digital objects just as one would to hard-copy objects. The main one is that very often one needs to move around, or to perform the same operation (e.g., create distribution forms) on large numbers of digital objects in groups of items that are, e.g., the same file type. For example, the Alexandria Digital Library (ADL), a collection of digital geospatial data, with a catalog that includes catalog records both for digital and hard-copy geospatial data (http://webclient.alexandria.ucsb.edu), always gives a digital geospatial data object (e.g., a scan of a paper map; a born-digital object) the same classification number as the hardcopy geospatial data equivalent (e.g., the paper map that has been scanned). Classification of digital objects allows one file to be accessible from multiple classification numbers, an action that few libraries of hardcopy items have ever been able to afford. For example, a digital map of California and Nevada may have two classification numbers (either pointing toward one file, or storing the same file two separate places), one for California and one for Nevada. Few libraries of hardcopy items have financial resources available to buy multiple copies of an item and store a copy at each applicable classification number.

Cataloging

A catalog record (whether in hardcopy or in digital form) provides information on the thematic and physical nature of an item (whether hard-copy or digital) being cataloged. Libraries first used hard-copy catalogs, generally book-format catalogs, then cards, and then beginning in the late 1960s the use of databases as catalogs. This started with in 1969 the Library of Congress' MARC, MAchine-Readable Catalog format for the transmission of catalog-card information in digital form (http://www.loc.gov/marc/). While MARC may be used as the machine format for bibliographic records formulated using any set of cataloging rules, it is most often used for records based on the cataloging rules of the Anglo-American library community, AACR (Anglo-American Cataloging Rules) in its various editions, first issued in 1967. AACR itself is based on the International Standard Bibliographic Descriptions (ISBDs) as far as content of fields, and field order, are concerned.

The purpose of ISBDs is to standardize the form and content – including in what order information appears – for the bibliographic description of an object. ISBDs specifically do not include rules for fields for subject cataloging, added entries for additional authors, or classification. The idea of ISBDs arose at a conference of the International Meeting of Cataloging Experts held in 1969, with the first ISBD (for monographic publications) being issued in 1971. ISBDs were issued for numerous categories of publications (e.g., computer files; cartographic materials; serials; etc.; for full list, see http://www.ifla.org/VI/3/nd1/isbd-list.htm). In 2002, work began on a single consolidated ISBD, to supersede all existing ISBDs. The consolidated edition (available online at http://www.ifla.org/VII/s13/pubs/cat-isbd.htm) was published in 2007 (International standard bibliographic description (ISBD), 2007).

The same organization that issues ISBDS – the International Federation of Library Associations (IFLA) – has been a prime agent in the move toward an international cataloging standard. In the early 1990s, IFLA's Study Group on the Functional Requirements for Bibliographic Records (FRBR) began work on its report to recommend basic functionalities of a catalog and bibliographic-record data requirements. The Group's final report was issued in 1998 (Functional Requirements for Bibliographic Records (FRBR)).

While FRBR put forward several ideas, the one that most engaged the interest and discussion of the library community was the importance of incorporating into the bibliographic description the concept of the relationship between the work, the expression, the manifestation, and the item (called Group 1 Entities). A work is a distinct intellectual or artistic creation but a concept rather than an actual physical object. An expression is the intellectual or artistic realization of a work but still not generally an actual physical object. The manifestation is all copies of a published object (e.g., all copies of a printed map; all copies of a DVD of a specific piece of music), and the item is one copy of a manifestation. For example, all digital versions of one given map (one could be raster and the other could be vector; there could be more than one level of resolution of raster and of vector images), and all hard-copy versions of the same map (e.g., paper; microfiche; microfilm) are two expressions' the basic map itself is the work and the 1973 edition of all copies of the paper map is a specific manifestation, with each one of those printed maps being an item.

Key Applications

Key applications of classification and cataloging:

Classification: Arrangement of digital objects in digital libraries. See previous section

Cataloging: Creation of metadata for digital objects in digital libraries

While full-text searching is extremely useful and a major step forward in the history of information retrieval, it does have the following main problems: it may give the user very large numbers of hits; it is not as efficient as searching well-constructed text surrogates; and it does not work for what are primarily non-text materials (e.g., music; maps; etc.). The reason for creating metadata records for digital objects is the same as that for performing standard cataloging – constructing a surrogate for the item so that users may quickly and efficiently find resources that suit the users' needs. Metadata is constructed by non-library entities (e.g., federal government agencies) and by libraries. Metadata records include but are not limited to the information contained in what the standard library cataloging world terms "bibliographic records."

Metadata records tend to be considerably longer than catalog records, because they contain far more and generally more detailed technical information than a catalog record would. The latter is much more likely to simply include a URL that points to an online version of that technical information, quite possibly to a metadata record. The reasons for this are that metadata records tend to be constructed for very focused, often technically skilled audiences, and a geographic digital dataset (such as a geographic information system, more commonly known as a GIS) is often quite large and complicated, with many layers of information, and therefore is by no means as easily browsed – in order to determine its suitability for use – as is a hard-copy map. For example, the catalog record for Digital Orthophoto Quarter Quadrangles (DOQQs; mosaics of rectified aerial photographs) is relatively brief – about one standard printed page – when compared with the metadata record for DOQQs, which is seven pages. For example, see http://fisher.lib.virginia.edu/collections/gis/doq/helps/doqq_meta.html (Fig. 1).

A library generating metadata records has two major options: load the records into the library's ILS (integrated library system) online catalog; or create what is in effect another ILS, or at the very least an online catalog for the metadata records. The first technique generally requires that the records be in MARC format, since the alternative is that the online-catalog software must be capable of searching over multiple catalog-record databases in different formats.

For the second technique, the following is required: software (UNIX; a database manager; a user interface; and middleware to connect inquiries on the user interface with the data and return results); hardware; computer technical staff/programmers (for a digital library of any size, a minimum of three computer programmers to deal with adding new data and metadata and maintaining and improving the system, including the interface, plus one more programmer to deal with the operating system, disk storage, and manipulation and maintenance of the directories of digital material).

An example of this is the ADL (Alexandria Digital Library) Catalog, http://webclient.alexandria.ucsb.edu. The ADL webpage (http://www.alexandria.ucsb.edu) provides an outline as to what kind and how much work is required to start up, develop, and maintain such a library; software is free for download, and general instructions are given as to what work should be done in order to get the software working.

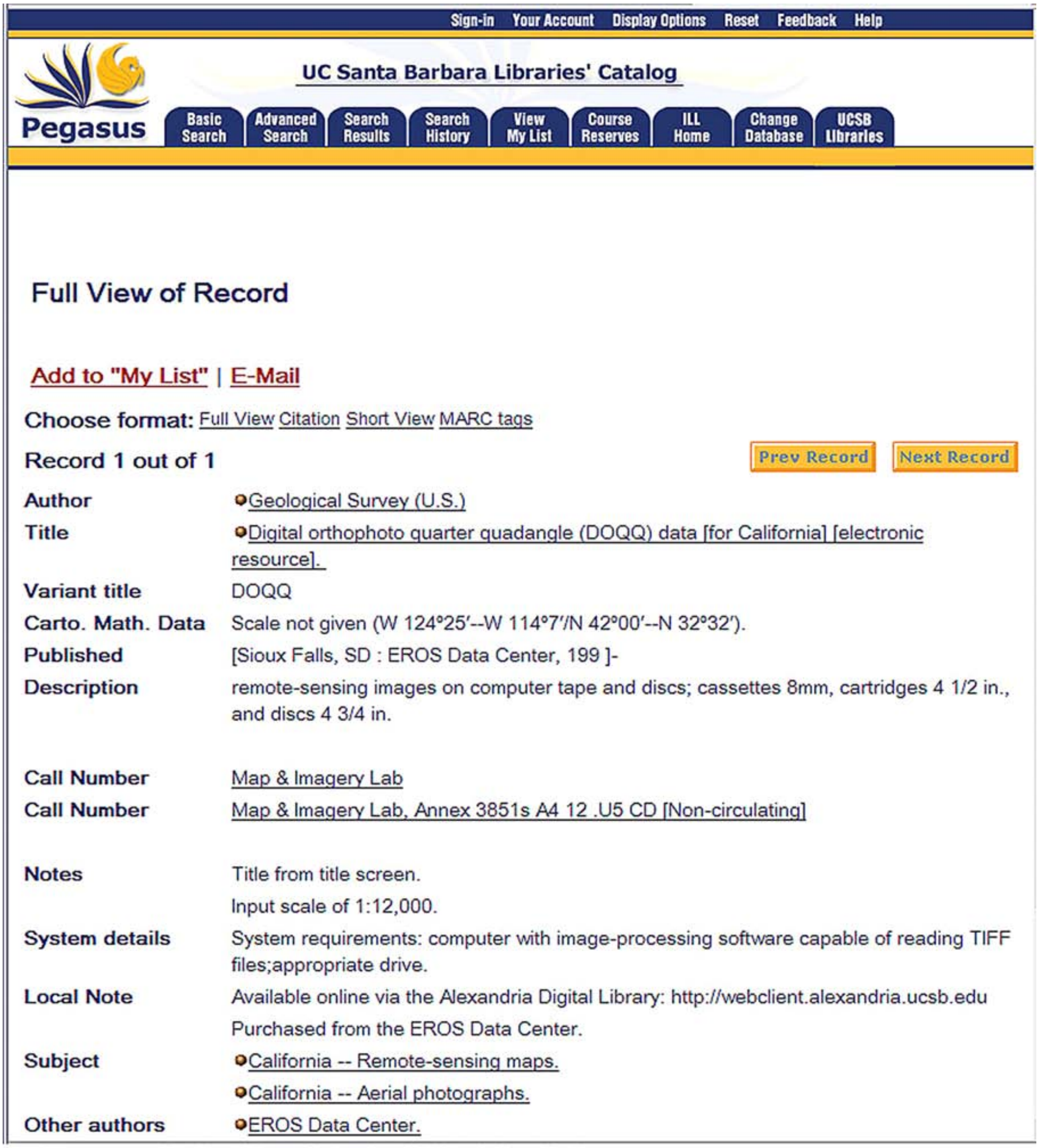
As previously indicated, there are numerous metadata standards. The following are major standards that digital libraries creating metadata records will probably need to deal with, at least in the United States: Dublin Core; XML; METS; and MODS.

Dublin Core

Dublin Core (DC) (http://dublincore.org/) is extremely heavily used by libraries cataloging digital content. Its adaptability to any form or type of digital data and its brevity (15 fields, what libraries term minimal-level cataloging) with no fields required and all fields repeatable, makes it very flexible. While DC may be used either as "qualified" (each of the 15 elements may be qualified in some way to make the information clear, e.g., for Coverage, one might state that the geographic area is given in decimal degrees), the experience in libraries over the nearly 15 years since DC was made available for use is that it is strongly advised only qualified DC be used. This is because unqualified DC results in metadata records that are so unstructured as to be nearly useless [12]. For digital libraries needing keep at least one foot solidly in the traditional library world, there is a DC-to-MARC2 crosswalk at http://www.loc.gov/marc/dccross.html, and also one the other direction.

XML

XML has achieved primacy as the format of choice for metadata for digital libraries and is of considerable importance to the standard library world, as evidenced by the Library of Congress having MARC21 in XML available over the Web at http://www.loc.gov/standards/marcxml/Sandburg/sandburg.xml. It was announced in April 2005 that the ISO



Full View of Record

Add to "My List" | E-Mail

Choose format: Full View Citation Short View MARC tags

Record 1 out of 1

Author	Geological Survey (U.S.)
Title	Digital orthophoto quarter quadangle (DOQQ) data [for California] [electronic resource].
Variant title	DOQQ
Carto. Math. Data	Scale not given (W 124°25′--W 114°7′/N 42°00′--N 32°32′).
Published	[Sioux Falls, SD : EROS Data Center, 199]-
Description	remote-sensing images on computer tape and discs; cassettes 8mm, cartridges 4 1/2 in., and discs 4 3/4 in.
Call Number	Map & Imagery Lab
Call Number	Map & Imagery Lab, Annex 3851s A4 12 .U5 CD [Non-circulating]
Notes	Title from title screen.
	Input scale of 1:12,000.
System details	System requirements: computer with image-processing software capable of reading TIFF files;appropriate drive.
Local Note	Available online via the Alexandria Digital Library: http://webclient.alexandria.ucsb.edu
	Purchased from the EROS Data Center.
Subject	California -- Remote-sensing maps.
	California -- Aerial photographs.
Other authors	EROS Data Center.

Image courtesy Davidson Library, University of California, Santa Barbara.
Copyright 2007 The Regents of theUniversity of California; all rights reserved.

Cataloging in Digital Libraries. Figure 1. Record from a library online catalog.

committee for Technical Interoperability – Information and Documentation was to vote on a proposal for a New Work Item concerning an XML schema to wrap MARC records. ISO 2709 had been used for many years and has worked well, but the library community needed a standard exclusively for MARC records (of which there are over 100 million worldw ide) in XML. The standard is to be published as ISO 25577 with the short name of MarcXchange; the temporary Webpage for the standard is http://www.bs.dk/marcxchange/.

METS and MODS

METS (Metadata Encoding and Transmission Standard) is a standard for encoding descriptive, administrative, and structural metadata of objects in a digital library (http://www.loc.gov/standards/mets/).

The "Metadata Object Description Schema" (MODS) is intended both to carry metadata from existing MARC21 records, and to be used to create new catalog records. It has a subset of MARC fields, and – unlike MARC21. It uses language-based tags rather than numeric tags. It occasionally regroups elements from the MARC21 bibliographic format (http://www.loc.gov/standards/mods/). METS and MODS are both expressed using XML, and are maintained by the Library of Congress's Network Development and MARC Standards Office.

Conclusion

Organizations creating metadata records and arranging digital files are best advised to follow well-maintained national or international standards. In no case should organizations just starting out on this work create their own standards. Instead, use of sturdy standards – some of which have an extensions feature, to enable customization of the records to the library users' needs – is recommended.

Creating metadata records is relatively easy compared with the difficult and expensive work of setting up what is in effect an ILS online catalog. Libraries need to consider this very carefully. If the library cannot sustain the programming effort required to develop and then to maintain and add to the catalog, then the library should not begin the project. If anything, digital libraries and their catalogs are, at the moment, at least as expensive and time-consuming to develop and maintain as are hard-copy libraries.

Cross-references

▶ Annotation
▶ Audio Metadata
▶ Biomedical Data/Content Acquisitions
▶ Biomedical Metadata Management and Resource Discovery
▶ Browsing in Digital Libraries
▶ Classification by Association Rule Analysis
▶ Classification on Streams
▶ Clinical Data/Content Acquisition
▶ Cross-Modal Information Retrieval
▶ Curation
▶ Data Warehouse Metadata
▶ Digital Libraries
▶ Discovery
▶ Dublin Core (DC)
▶ Field Based Information Retrieval Models
▶ Geographic Information Retrieval
▶ Image Metadata
▶ Indexing Historical Spatio-Temporal Data
▶ Information Retrieval
▶ Information Retrieval Model
▶ ISO/IEC 11179
▶ Knowledge Discovery Metamodel
▶ Metadata
▶ Metadata Interchange Specification(MDIS)
▶ Metadata Registry
▶ Metadata Repository
▶ Metasearch Engines
▶ METS
▶ Multimedia Metadata
▶ Ontologies
▶ Ontology
▶ Schema Mapping
▶ Searching Digital Libraries
▶ Text Indexing and Retrieval
▶ XML
▶ XML Metadata Interchange

Recommended Reading

1. Anglo-American Cataloging Rules, American Library Association, Chicago, 2005.
2. Borgman C.L. From Gutenberg to the Global Information Infrastructure: Access to Information in the Networked World. MIT Press, Cambridge, MA, 2000.
3. Chan L.M. Cataloging and Classification: An Introduction. Scarecrow Press, Blue Ridge Summit, PA, 2007.
4. IFLA Study Group. Functional Requirements for Bibliographic Records (FRBR). K.G. Saur, Munchen, 1998, (UBCIM publications, new series; vol. 19). Available online at: http://www.ifla.org/VII/s13/frbr/frbr.htm.
5. IFLA Study Group. International Standard Bibliographic Description (ISBD), (Preliminary consolidated edn.). K.G. Saur, München, 2007 (IFLA series on bibliographic control; vol. 31).
6. Kochtanek T.R. Library Information Systems, From Library Automation to Distributed Information Access Solutions. Libraries Unlimited, Westport, CT, 2002.
7. Libraries. Encyclopedia Britannica, Micropedia 7:333–334; Macropedia 22:947–963. Encyclopedia Britannica, Chicago, 2002. Available online at: http://search.eb.com/.
8. Library of Congress. 1969? MARC21 Concise Bibliographic, Library of Congress, Washington, DC. Available online at http://www.loc.gov/marc/.
9. Linton J. Beyond Schemas, Planning Your XML Model. O'Reilly, Sebastopol, CA, 2007.
10. Reitz J.M. Dictionary for Library and Information Science. Libraries Unlimited, Westport, CT, 2004.

11. Svenonius E. The Intellectual Foundation of Information Organization. MIT Press, Cambridge, MA, 2000.
12. Tennant R. Bitter Harvest: Problems and Suggested Solutions for OAI-PMH Data and Service Providers. California Digital Library, Oakland, CA, 2004. Available online at:http://www.cdlib.org/inside/projects/harvesting/bitter_harvest.html.

CDA

▶ Clinical Document Architecture

CDA R1

▶ Clinical Document Architecture

CDA R2

▶ Clinical Document Architecture

CDP

▶ Continuous Data Protection

CDs

▶ Storage Devices

CDS

▶ Clinical Decision Support

Cell Complex

▶ Simplicial Complex

Certain (and Possible) Answers

Gösta Grahne
Concordia University, Montreal, QC, Canada

Synonyms

True answer (Maybe answer); Validity (Satisfiability)

Definition

Let T be a finite theory expressed in a language L, and ϕ an L-sentence. Then T finitely entails ϕ, in notation $T \vDash \phi$, if all finite models of T also are models of ϕ. A theory T is said to be *complete in the finite* if for each L-sentence ϕ either $T \vDash \phi$ or $T \vDash \neg\phi$. In particular, if T is incomplete (not complete in the finite), then there is an L-sentence ϕ, such that $T \nvDash \phi$ and $T \nvDash \neg\phi$. It follows from classical logic that a first order theory is complete in the finite if and only if all its finite models are isomorphic. Consider now a theory

$$T_1 = \begin{cases} R(a,b) \wedge R(a,c), \\ \forall x, y : R(x,y) \rightarrow (x,y) = (a,b) \vee (a,c), \\ a \neq b, a \neq c, b \neq c. \end{cases}$$

where a, b, and c are constants. This theory is complete, and clearly for instance $T \vDash R(a, b)$, $T \vDash R(a, c)$, and $T \nvDash R(d, c)$, for all constants d different from a and b. Consider then the theory

$$T_2 = \begin{cases} R(a,b) \vee R(a,c), \\ \forall x, y : R(x,y) \rightarrow (x,y) = (a,b) \vee (a,c), \\ a \neq b, a \neq c, b \neq c. \end{cases}$$

This theory is incomplete, since for instance $T_2 \nvDash R(a, b)$, and $T_2 \nvDash \neg R(a, b)$. If "finitely entails" is equated with "certainly holds," it is possible to say that $R(a, b)$ and $R(a, c)$ *certainly* hold in T_1. Dually, it is possible to say that $R(a, b)$ *possibly* holds in T_2, since $T_2 \nvDash \neg R(a, b)$, and similarly that $R(a, c)$ possibly holds in T_2.

Key Points

An *incomplete database* is similar to a logical theory: it is defined using a finite specification, usually a *table* T (relation with nulls and conditions) of some sort, and a function *Rep* that associates a set of complete (ordinary, finite) databases $Rep(T)$ with T. Then each instance $I \in Rep(T)$ represents one isomorphism class (isomorphism up to renaming of the constants) of the finite models of the table T regarded as a logical theory. Depending on the interpretation of facts missing from T, either the *closed world assumption* is made [9], which postulates or axiomatizes (as in the middle "row" in T_1 and T_2) that any facts not deducible from T are *false*, or the *open world assumption* (omit the middle rows), in which there are certain and possible facts, but no false ones. There is actually a spectrum of closed world assumptions, ranging up to semantics best axiomatized in third order logic [4].

Having settled on a representation T, and an interpretation *Rep*, the *certain answer* to a query Q on an incomplete database T, is now defined as $\bigcap_{I \in Rep(T)} Q(I)$, sometimes also denoted $\bigcap Q(Rep(T))$. In database parlance the certain answer consists of those facts that are true in *every* possible database instance I that T represents. Likewise, the *possible answer* to a query Q on an incomplete database T, consists of those facts which are true in *some* possible database, i.e., $\bigcup_{I \in Rep(T)} Q(I)$. Needless to say, the possible answer $\bigcup Q(Rep(T))$ is interesting only under a closed world assumption, since otherwise every fact is possible.

These definitions are clear and crisp, but unfortunately it doesn't mean that they always have tractable computational properties. Consider the membership problem for the set

$$\textsc{Cert}(Q) = \{(t,T) : t \in \bigcap Q(Rep(T))\}.$$

If T actually is a complete instance I, it is well known that $\textsc{Cert}(Q)$ has polynomial time complexity, for any first order (relational algebra) or datalog query Q. Likewise, the set

$$\textsc{Poss}(Q) = \{(t,T) : t \in \bigcup Q(Rep(T)).$$

has PTIME complexity for first order and datalog queries Q, and tables T that actually are complete databases.

A table T with *unmarked nulls* is a classical instance containing existentially quantified variables (nulls), such that each existential quantifier has only one variable in its scope. This means that each occurrence of a null can be independently substituted by a constant for obtaining one possible database in $Rep(T)$.

If only simple incomplete databases with unmarked nulls are allowed, only existential first order queries Q need to be admitted, or alternatively algebraic expressions with operators from $\{\pi, \sigma, \cup, \bowtie\}$, in order for $\textsc{Cert}(Q)$ to become coNP-complete, and $\textsc{Poss}(Q)$ to become NP-complete [2]. The use of inequalities $\neq$ or disjunctions $\vee$ in Q is essential. If the use of inequalities and disjunctions is denied, $\textsc{Cert}$ (Q) and $\textsc{Poss}(Q)$ remain in PTIME. If one admits arbitrary first order or full relational queries Q, along with an open world assumption, $\textsc{Cert}(Q)$ and $\textsc{Poss}(Q)$ become undecidable. This follows from validity and satisfiability of a variant of first order logic known to be undecidable [3]. (Note that under the open world assumption $\textsc{Poss}(Q)$ equals all possible databases, assuming $\textsc{Poss}(Q) \neq \emptyset$. The problem then becomes to decide whether $\textsc{Poss}(Q)$ is non-empty or not.)

On the other hand, if the representation mechanism allowed for T is more powerful that the simple incomplete databases above, $\textsc{Cert}(Q)$ and $\textsc{Poss}(Q)$ again become coNP and NP complete, respectively, already with Q being the identity query. For this, the *conditional tables* of [6] are needed. As observed in [5], conditional tables can be obtained as a closure of simple incomplete databases by requiring that the exact result $\{q(I) : I \in Rep(T)\}$ of any relational algebra query on a any table T is representable by conditional table. In other words, for each T conditional table and Q relational algebra query, there exists a conditional table U, such that $Rep(U) = Q(Rep(T))$.

Another way of representing incomplete databases, is to consider an *information integration* scenario, where the basic facts are stored in *views* of a virtual *global schema*. For instance, in the integration scenario

$$T_3 = \begin{cases} V(a), \\ \forall x, y : R(x,y) \rightarrow V(x), \\ \forall x : V(x) \rightarrow x = a, \\ \forall x, y : R(x,y) \rightarrow x = a \end{cases}$$

gives (closed world) $Rep(T_3) = \{I : V^I = \pi_1(R^I) = \{(a)\}\}$. ($R^I$ means the value (interpretation) of predicate symbol R in instance/model I. The meaning of V^I is similar.) Open world (omit in T_3 the third and fourth rows) $Rep(T_3)$ would be defined as $\{I : V^I \supseteq \{(a)\}, \pi_1(R^I) \supseteq \{(a)\}$. If T is allowed to use conjunctive queries (such as the second row of T_3) to express the views V in terms of the global relations R, then $\textsc{Cert}(Q)$ is in PTIME for existential first order and datalog queries under the open world assumption, and coNP complete under the closed world assumption [1]. The latter is due to the negation implicit in the closed world assumption. If one allows inequalities $\neq$ in the query, $\textsc{Cert}(Q)$ is coNP complete also under the open world assumption. Undecidability of $\textsc{Cert}(Q)$ is achieved by allowing negation in the query or the view definitions, or, under the open world assumption by allowing view definitions in (recursive) datalog.

Finally, one can see the *data exchange* problem [7,8] as a variation of the integration problem. The data exchange problem consists of importing the data from a source database R_s to a target database R_t, using data dependencies (implicational sentences) to express the translation. For example, a (closed world) exchange scenario could be

$$T_4 = \{R_s(a), \forall x : [R_s(x) \leftrightarrow \exists y : R_t(x, y)].$$

The base facts are in a source database R_s, and the user query is expressed against the target database R_t. As T_4 obviously is incomplete, the certain answer of Q on T_4 is defined as $\cap Q(Rep(T_4)$, and the possible answer as $\cup Q(Rep(T_4)$. It is perhaps no big surprise that essentially the same complexity landscape for $\text{Cert}(Q)$ and $\text{Poss}(Q)$ as in the previous table- and integration-scenarios emerges: the boundaries between undecidability, intractability, and polynomial time depend on similar restrictions on the use of negation, of inequalities or unions in the exchange mappings, and on the open or closed world assumptions.

Cross-references

- ▶ Conditional Tables
- ▶ Incomplete Information
- ▶ Naive Tables
- ▶ Null Values

Recommended Reading

1. Abiteboul S. and Duschka O.M. Complexity of answering queries using materialized views. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 254–263.
2. Abiteboul S., Kanellakis P.C., and Grahne G. On the representation and querying of sets of possible worlds. Theor. Comput. Sci., 78(1):158–187, 1991.
3. Di Paola R.A. The recursive unsolvability of the decision problem for the class of definite formulas. J. ACM, 16(2):324–327, 1969.
4. Eiter T., Gottlob G., Gurevich Y. Curb your theory! a circumspective approach for inclusive interpretation of disjunctive information. In Proc. 13th Int. Joint Conf. on AI, 1993, pp. 634–639.
5. Green T.J. and Tannen V. Models for incomplete and probabilistic information. In Proc. EDBT 2006 workshops LNCS Vol. 4251, 2006.
6. Imielinski T. and Lipski W. Incomplete information in relational databases. J. ACM, 31(4):761–791, 1984.
7. Kolaitis P.G. Schema mappings, data exchange, and metadata management. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 61–75.
8. Libkin L. Data exchange and incomplete information. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 60–69.
9. Reiter R. On closed world data bases. In Logic and Data Bases, 1977, pp. 55–76.

Chandra and Harel Complete Query Languages

▶ Complete query languages

Change Detection and Explanation on Streams

▶ Change Detection on Streams

Change Detection on Streams

Daniel Kifer
Yahoo! Research, Santa Clara, CA, USA

Synonyms

Change detection and explanation on streams

Definition

A *data stream* is a (potentially infinite) sequence of data items $x_1, x_2, \ldots$. As opposed to traditional data analysis, it is not assumed that the data items are generated independently from the same probability distribution. Thus *change detection* is an important part of data stream mining. It consists of two tasks: determining when there is a change in the characteristics of the data stream (preferably as quickly as possible) and explaining what is the nature of the change.

The nature of the data stream model means that it may be infeasible to store all of the data or to make several passes over it. For this reason, change detection algorithms should satisfy the following desiderata: the memory requirements should be constant or increase logarithmically, and the algorithm should require only one pass over the data.

Historical Background

There has been a lot of work on detecting change in time series data *after* all of the data has been collected. Change point analysis [5] is a statistical field devoted to detecting the point in time where the distribution of the data has changed. The description of the change is often concerned with how the parameters of the distributions (such as the mean) have changed. Scan statistics [11] can be used to detect and describe changes (not just along the time dimension) by identifying regions where the probability mass has changed the most. For examples of offline analysis of change in terms of data mining models see [6] for itemset mining, [10] for itemset mining and decision trees, and [14] for burst detection. Offline algorithms for describing change include [1] for hierarchical numerical data and [7,8] for semi-structured data.

The offline methods are useful for data analysis, but as data acquisition becomes easier and easier, the data stream model becomes more and more relevant. The assumptions behind this model are that there is so much data that it cannot all be stored on disk (let alone kept in memory) and that the data arrives at such a rate that expensive online computations are impractical. Often results are expected in real-time – for example, notification of change should occur as soon as possible. The data stream model is discussed in detail in [2].

In the data stream model, one can predefine a certain set of stochastic processes. One of these processes is initially active and the goal is to determine when a different process in that set becomes active [3]. The description of the change is then the identity of the new process. Alternative approaches [9,13] avoid making distributional assumptions by using ideas from nonparametric statistics. In these approaches the main idea is to divide the domain of the data into (possibly overlapping) regions, estimate the probability of the regions, and then determine whether the changes in probability in any of the regions are statistically significant. Alternate approaches for handling change include testing if the data are exchangeable (i.e., any permutation of the data is equally likely) [15] and developing data mining algorithms (such as decision tree construction) that adapt to change [12].

Foundations

Let $x_1,x_2,...,$ be a potentially infinite sequence of data points. To detect changes in the data stream, one first has to determine a plausible framework that describes how the data can be generated. In the simplest case, data are generated from one of two probability distributions S_1 and S_2 (for example a Gaussian with mean 0 and variance 1, and a Gaussian with mean 10 and variance 1). Initially, the data points are generated independently from S_1 and after some time the data are generated independently from S_2. The celebrated CUSUM algorithm by Page [4] can be used to detect that a change from S_1 to S_2 occurred by comparing the likelihoods that parts of the data were generated by S_1 or S_2. Suppose S_1 has density f_1 and S_2 has density f_2, and let $\delta > 0$ be a threshold.

A user of the change-detection system is interested in the first time k where. $\sum_{i=k}^{\text{now}} \log(f_2(x_i)/f_1(x_i)) > \delta$. When this happens, the system signals that a change has occurred and can return k as a plausible estimate of the change point. This test can be done in an online fashion by defining $T_0 = 0$ and $T_k = \max(T_{k-1} + \log(f_2(x_k)/f_1(x_k)),0)$ and signaling a change if $T_{\text{now}} > \delta$. Typically S_1 is chosen based on an initial sample of the data stream and S_2 is then chosen to represent the smallest change whose detection is desired. For example, suppose that S_1 is chosen to be a Gaussian with mean 0 and variance 1, and suppose that for the current application it is desirable to detect any change in mean greater than 10. Then a natural choice for S_2 is a Gaussian with mean 10 and variance 1.

This framework has been generalized by Bansal and Papantoni-Kazakos [3] to the case where S_1 and S_2 are stochastic processes that need not generate each point independently. Additional generalizations of the CUSUM algorithm, including the case of multiple data generating distributions, are discussed in [4].

The framework of the CUSUM algorithm is an example of a *parametric* framework: the set of possible data-generating distributions has been prespecified and elements in that set can be identified using a small number of parameters. Parametric approaches are powerful in cases where they can accurately model the data. In cases where the data is not well modeled by a parametric framework, performance may deteriorate in terms of more false change reports and/or fewer detection of changes.

Kifer, Ben-David, and Gehrke [13] showed how to avoid problems with parametric approaches by using a nonparametric framework. In this framework the data points $x_1,x_2,...,$ are k-dimensional vectors of real numbers. Point x_1 is generated by some (arbitrary) probability distribution F_1, x_2 is generated by F_2 (independently of x_1), x_3 is generated by probability distribution F_3 (independently of x_1 and x_2), etc. A change is defined as a change in the data-generating distribution; if the first change occurs at time n_1 then $F_1 = F_2 = ... = F_{n_1-1} \neq F_{n_1}$; if the second change occurs at time n_2 then $F_{n_1} = F_{n_1+1} = ... = F_{n_2-1} \neq F_{n_2}$, etc. This framework for detecting change consists of three parts: a collection of regions of interests, a method for estimating probabilities, and a statistical test.

Regions of Interest

A collection of regions of interest serves two purposes: to restrict attention to changes that are considered meaningful, and to provide a means for describing the change.

Ideally a change is said to have occurred whenever the data-generating distribution changes. However, for many practical applications not all changes are meaningful. For example, consider the case where F_1 is a probability distribution that assigns probability 1 to the set of real numbers between 0 and 1 whose seventeenth significant digit is odd and furthermore suppose that F_1 is uniform over this set. From time 1 up to $n - 1$ the data are generated independently from the distribution F_1. At time n a change occurs and from that point the data are generated independently from the distribution F_n defined as follows: F_n assigns probability 1 to the set of real numbers between 0 and 1 whose seventeenth significant digit is even and is uniform over this set. Letting f_1 be the probability density function for F_1 and f_n be the probability density function for F_n it can be seen that f_1 and f_n are very different according to some common similarity measures. Indeed, the L^1 distance between f_1 and f_n (i.e., $\int_0^1 |f_1(x) - f_n(x)|\, dx$), is as large as possible. However, in many applications it is of no practical consequence whether the true distribution is F_1 or F_n. This is because one may be interested only in questions such as "what is the probability that the next point is larger than 0.75" or "what is the probability that the next point is within the safety range of 0.14–0.95." For these types of questions one would only be interested in the probabilities of various intervals, so instead of receiving notification of arbitrary types of change, one would be happy to know only when some interval has become more or less probable. In this case, the intervals are said to be the *regions of interest*. In general, if the domain of each data element is $\mathcal{D}$ then the set of regions of interest is a collection of subsets of $\mathcal{D}$. Note that regions of interest can be overlapping, as in the case of intervals, or they can form a partition of the domain. Dasu et al. [9] also proposed to partition the domain based on an initial sample of the data.

Once the regions of interest have been specified, the goal is to report a change whenever the system detects that the probability of a region has changed. The region (or regions) with the largest change in probability are then given as the description of the change.

Estimating Probabilities

Since the true distributions F_i are unknown, it is necessary to estimate them. A *window of size m* is a set of m consecutive data points. The initial distribution is estimated using a window that contains the first m data points and the most recent distribution is estimated using a window containing the most recent m data points (in practice, several change detection algorithms can be run in parallel, each using a different value of m). In each window W_i, the probability of a particular region of interest R can be estimated by $\frac{|W_i \cap R|}{|W_i|}$, the fraction of points in the window that occur in the region. Alternatively, if the regions of interest form a partition of the domain into k regions, then a Bayesian-style correction $\frac{|W_i \cap R| + \alpha}{|W_i| + \alpha k}$ can also be used [9].

Statistical Testing

In this setting, a *statistic f* is a function that assigns a number to a pair of windows (W_1, W_2). The larger this number is, the more likely it is that the points in one window were generated from one distribution and the points in the other window were generated from a different distribution. A *statistical test* is a statistic f and a real number τ that serves as a threshold; when $f(W_1, W_2) \geq \tau$ then one can conclude that the points in W_1 were generated from a different distribution than the points in W_2.

For each $i \geq 1$ let W_i be the window that contains the points $x_i, \ldots, x_{i+m-1}$, so that W_1 is the set of the first m data points. To use a statistical test f with threshold τ, one computes the values $f(W_1, W_m)$, $f(W_1, W_{m+2})$, $f(W_1, W_{m+3}), \ldots$, and signals a change the first time i such that $f(W_1, W_i) \geq \tau$. At this time, the current window W_i is considered to be a set of m points generated from the new distribution. The distribution may change again in the future, so one proceeds by computing the values of $f(W_i, W_{i+m})$, $f(W_i, W_{i+m+1})$, $f(W_{i+m+2})$, etc., until another change is detected.

Note that in order for it to be useful, a statistic should be easy to compute since a new value must be computed every time a new data point arrives. The value of the threshold τ should also be carefully chosen to avoid incorrect detections of change. A *false positive* is said to have occurred if the algorithm reports a change when the underlying distribution has not changed. Since a stream is a potentially unbounded source of data, false positives will occur and so instead of bounding the probability of a false positive, the goal is to choose a value of τ that bounds the *expected rate* of false positives. Several statistics for detecting change and a method for choosing τ are presented next.

Let $\mathcal{A}$ be the collection of regions of interest. Let W and W' be two windows and let P and P' be the corresponding probability estimates: for any $A \in \mathcal{A}$, $P(A)$ is the estimated probability of region A based on window W and $P'(A)$ is the estimated probability of A based on window W'. The following statistics can be used in the change detection framework [13]:

$$d_{\mathcal{A}}(W, W') = \sup_{A \in \mathcal{A}} |P(A) - P'(A)|$$

$$\phi_{\mathcal{A}}(W, W') = \sup_{A \in \mathcal{A}} \frac{|P(A) - P'(A)|}{\sqrt{\min\left\{\frac{P(A)+P'(A)}{2}, 1 - \frac{P(A)+P'(A)}{2}\right\}}}$$

$$\Xi_{\mathcal{A}}(W, W') = \sup_{A \in \mathcal{A}} \frac{|P(A) - P'(A)|}{\sqrt{\frac{P(A)+P'(A)}{2}\left(1 - \frac{P(A)+P'(A)}{2}\right)}}$$

Note that when $\mathcal{A}$ is the set of intervals of the form $(-\infty, b)$ then $d_{\mathcal{A}}$ is also known as the Kolmogorov-Smirnov statistic. For any one of these statistics, the region $A \in \mathcal{A}$ which maximizes the statistic is the region where the change in observed probability is the most statistically significant; this region (or the ℓ most significant regions, depending on user preferences) and its change in probability is therefore the description of the change.

To use these statistics, one must determine the value of the threshold τ and the corresponding expected rate of false positives. To do this one can take advantage of the fact that for one-dimensional data, the worst-case behavior of the $d_{\mathcal{A}}$, $\phi_{\mathcal{A}}$, and $\Xi_{\mathcal{A}}$ statistics occur when the data are generated by continuous distributions and that the statistics behave in the same way for all continuous distributions [13]. This means that one can perform an offline computationally-intensive simulation to determine τ and then use this value for *any* one-dimensional stream afterwards.

To perform the simulation, a user must specify a test statistic f and four parameters: a window size m, a real number p between 0 and $\frac{1}{2}$, a large integer $q >= 2m$ (e.g., 1 million), and the number of repetitions B. For each repetition i, generate q points independently from any continuous distribution (e.g., from a Gaussian distribution with mean 0 and variance 1). Compute the value $\tau_i \equiv \max_{m \leq j \leq q-m+1} f(W_1, W_j)$ (this represents the largest value of the statistic f that would have encountered if this were the real data). After B repetitions, choose a value for the threshold τ such that τ is greater than $(1 - p)B$ of the values $\tau_1, \ldots, \tau_B$. This value of τ guarantees that the probability of a false positive in the first q points is approximately p.

To compute the expected rate of false positives corresponding to τ, one first notes that false reports of change should occur in pairs for the following reason. Once a false positive has occurred, one has a window with points that are considered anomalous (since they caused a change to be reported); as new data points arrive, one compares the m most recent points (which are still generated from the original distribution) with this anomalous window using the chosen test statistic and therefore a second report of change should soon occur. Thus one can upper bound the expected number H of false positives in the first q points using the following probability distribution: $P(H = 2) = p$, $P(H = 4) = p^2$, etc., and $P(H = 0) = \frac{1-2p}{1-p}$. The expected value is $\frac{2p}{(1-p)^2}$ and one can use this as an upper bound on the number of false positives in the first q points. One can approximate the expected number of errors in the next q points also by $\frac{2p}{(1-p)^2}$ so that the expected rate of false positives is approximated by $\frac{2p}{q(1-p)^2}$.

When the regions of interest form a partition of the domain into k regions, other statistics, such as the KL-distance can be used [9]:

$$KL_{\mathcal{A}}(W, W') = \sum_{A \in \mathcal{A}} P(A) \log \frac{P(A)}{P'(A)}$$

where the probabilities $P(A)$ and $P'(A)$ are estimated using the Bayesian correction (i.e., $P(A) = \frac{|W \cap A| + \alpha}{|W| + \alpha k}$). Dasu et al. propose using the KL-distance with the following scheme (which uses a user-defined parameter γ): initially collect m data points for the window W_1 and use these points to create the regions of interest which partition the (possibly high-dimensional) domain; then compute $KL_{\mathcal{A}}(W_1, W_m), KL_{\mathcal{A}}(W_1, W_{m+1}), KL_{\mathcal{A}}(W_1, W_{m+2})$, etc., and report a change whenever $n\gamma$ consecutive values of the statistic exceed a threshold τ. The value of τ depends on the points in W_1 and must be recomputed every time a change is detected. As before, τ is estimated via simulation.

To determine the value of τ, choose a parameter p $(0 < p < 1/2)$ and number of repetitions B. For each repetition i, use the probability distribution P estimated from W_1 to generate two windows V_1 and V_2 of m points each. Define τ_i to be $KL_{\mathcal{A}}(V_1, V_2)$. After B

repetitions, choose τ so that it is greater than $(1 - p)B$ of the $\tau_1,\ldots,\tau_B$.

Key Applications

Data mining, network monitoring.

Future Directions

Key open problems include efficiently detecting change in high-dimensional spaces (see also [9]) and detecting change in streams where data points are not generated independently.

Cross-references

▶ Stream Data Management
▶ Stream Mining

Recommended Reading

1. Agarwal D., Barman D., Gunopulos D., Korn F., Srivastava D., and Young N. Efficient and effective explanation of change in hierarchical summaries. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007, pp. 6–15.
2. Babcock B., Babu S., Datar M., Motwani R., and Wisdom J. Models and issues in data stream systems. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 1–16.
3. Bansal R.K. and Papantoni-Kazakos P. An algorithm for detecting a change in a stochastic process. IEEE Trans. Inf. Theor., 32(2):227–235, 1986.
4. Basseville M. and Nikiforov I.V. Detection of Abrupt Changes: Theory and Application. Prentice-Hall, Englewood Cliffs, NJ, 1993.
5. Carlstein E., Müller H.-G., and Siegmund D. (eds.) Change-point problems. Institute of Mathematical Statistics, Hayward, CA, USA, 1994.
6. Chahrabarti S., Sarawagi S., and Dom B. Mining surprising patterns using temporal description length. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 606–617.
7. Chawathe S.S., Abiteboul S., and Widom J. Representing and querying changes in semi-structured data. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 4–13.
8. Chawathe S.S. and Garcia-Molina H. Meaningful change detection in structured data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 26–37.
9. Dasu T., Krishnan S., Venkatasubramanian S., and Yi K. An information-theoretic approach to detecting changes in multi-dimensional data streams. In Proc. 38th Symp. on the Interface of Statistics, Computing Science, and Applications, 2006.
10. Ganti V., Gehrke J., and Ramakrishnan R. Mining data streams under block evolution. SIGKDD Explorations, 3(2):1–10, 2002.
11. Glaz J. and Balakrishnan N. (eds.) Scan Statistics and Applications. Birkhäuser, Boston, USA, 1999.
12. Hulten G., Spencer L., and Domingos P. Mining time-changing data streams. In Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2001, pp. 97–106.
13. Kifer D., Ben-David S., and Gehrke J. Detecting change in data streams. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 180–191.
14. Kleinberg J.M. Bursty and hierarchical structure in streams. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 91–101.
15. Vovk V., Nouretdinov I., and Gammerman A. Testing exchangeability on-line. In Proc. 20th Int. Conf. on Machine Learning, 2003, pp. 768–775.

Channel-Based Publish/Subscribe

Hans-Arno Jacobsen
University of Toronto, Toronto, ON, Canada

Synonyms

Event channel; Event service

Definition

Channel-based publish/subscribe is a communication abstraction that supports data dissemination among many sources and many sinks. It is an instance of the more general publish/subscribe concept. The communication channel mediates between publishing data sources and subscribing data sinks and decouples their interaction.

Key Points

Publishing data sources submit messages to the channel and subscribing data sinks listen to the channel. All messages published to the channel are received by all subscribers listening on the channel. The channel broadcasts a publication message to all listening subscribers.

The channel decouples the interaction among publishing data sources and subscribing data sinks. The same decoupling characteristics as discussed under the general publish/subscribe concept apply here as well. Realizations of this model found in practice vary in the exact decoupling offered. To properly qualify as publish/subscribe, at least the anonymous communication style must exist. That is publishing clients must not be aware of who the subscribing clients are and how many subscribing clients exist, and vice versa. Thus, channel-based publish/subscribe enables the decoupled interaction of n sources with m sinks for $n, m \geq 1$.

Channel-based publish/subscribe systems often allow the application developer to create multiple logical channels, where each channel can be configured to offer different qualities-of-service to an application. Furthermore, a channel can be dedicated to the dissemination of messages pertaining to a specific subject or type. The channel-based publish/subscribe model does not support message filtering, except through the use of various channels to partition the publication space. It is the clients' responsibility to select the right channel for the dissemination of messages, which are sent to all listeners on the channel. This enables a limited form of filtering by constraining messages to be disseminated on one channel to a given message type. Finally, channel-based publish/subscribe is often coupled with client-side filtering, where messages are still broadcast throughout the channel, but filtered upon arrival at the data sink before passing to the application. More fine-grained filtering functionalities are provided by the other publish/subscribe models, such as the topic-based model and the content-based model.

In channel-based publish/subscribe, the publication data model is defined by the type of message the channel-based communication abstraction supports. This is often closely tied to the programming language or the library that implements the model.

Similarly, the subscription language model is defined by the programming language or library that allows the application developer to select channels to listen to, unless special provisions for subscriber-side filtering are offered. If supported, subscriber-side filtering can be arbitrarily complex, even selecting messages based on their content, as offered by the content-based publish/subscribe model.

Matching in the sense of evaluating a publication message against a set of subscriptions, as is common in the other publish/subscribe instantiations, does not occur in channel-based publish/subscribe.

Channel-based publish/subscribe systems are often coupled with different client interaction styles. These are the *push-style* and the *pull-style*. In the push-style, data sources initiate the transfer of messages to the channel, which delivers the messages to all listening data sinks. In the pull-style, data sinks initiate the message transfer by requesting messages from the channel, which requests any available messages from all connected data sources. Both interaction styles can also be combined. That is on one channel some clients can connect to the channel through the push-style, while others connect via the pull-style.

Channel-based publish/subscribe systems are distinguished by the qualities-of-service the channel offers to its clients, such as various degrees of reliability, persistence, real-time constraints, and message delivery guarantees. Channel-based publish/subscribe relates to topic-based publish/subscribe in that publishing a message to a channel is similar to associating a message with a topic, which could be the name or identity of the channel. However, in topic-based publish/subscribe this association is reflected in the message itself, while in channel-based publish/subscribe the association is indirect, reflected by selecting a channel, not part of the message. Also, topics can go far beyond channel identities, as discussed under the topic-based publish/subscribe concept. Examples that follow the channel-based publish/subscribe model are the CORBA Event Service [2], IP multicast [?], Usenet newsgroups [?], mailing lists, and group communication [?]. Elements of channel-based publish/subscribe can also be found in the Java Messaging Service [1], the OMG Data Dissemination Service [3], and other messaging middleware. However, these approaches are not directly following the channel-based model as described above; rather these approaches are enriched with elements of message queuing, topic-based publish/subscribe, and content-based publish/subscribe.

There are many applications of channel-based publish/subscribe. Examples include change notification, update propagation, information dissemination, newsgroups, email lists, and system management. Channel-based publish/subscribe serves well, if one or more entities have to communicate date to an anonymous group of receivers that may change over time, without the need of filtering messages within the channel.

In the literature the term channel-based publish/subscribe is not used uniformly. Abstractions that exhibit the above described functionality are also often referred to as event services, event channels, and simply channels. Messages disseminated to listeners are also often referred to as events. Publishing data sources are often referred to as publishers, producers or suppliers, and subscribing data sinks are often referred to as subscribers, consumers or listeners.

Cross-references

- ▶ Content-Based Publish/Subscribe
- ▶ Publish/Subscribe
- ▶ Topic-Based Publish/Subscribe

Recommended Reading

1. Hapner M., Burridge R., and Sharma R. Java Message Service. Sun Microsystems, version 1.0.2 edition, Nov 9, 1999.
2. OMG. Event Service Specification, version 1.2, formal/04–10–02 edition, October 2004.
3. OMG. Data Distribution Service for Real-time Systems, version 1.2, formal/07–01–01 edition, January 2007.

Chart

Hans Hinterberger
ETH Zurich, Zurich, Switzerland

Synonyms

Chart; Map; Diagram; Information graphic; Graph

Definition

A chart is an instrument to consolidate and display information.

The term is applied to virtually any graphic that displays information, be it a map used for navigation, a plan for military operations, a musical arrangement, barometric pressure, genealogical data, and even lists of tunes that are most popular at a given time.

Definitions of specialized charts typically include the graphical method on which the chart is based (e.g., bar chart) and or its application area (e.g., CPM chart) but it does not specify design principles. Tufte [2] introduces the notion of "chartjunk" and defines it to be that part of the chart which functions only as decoration, all of which is considered to redundant data ink.

Sometimes charts are reduced to refer to maps and diagrams, excluding graphs and tables.

Key Points

Because charts are used for different purposes there exist almost as many different types of charts as there are applications. Some maps, however, are more general in their use and therefore assigned to one of the following four categories: Graphs, Maps, Diagrams, and Tables. Each of these categories is broken down further into subcategories. In the category Diagrams, one therefore finds pie charts as well as flow charts or organization charts. A detailed categorization can be found in [1].

Charts are not only used to visualize data. They serve as a useful tool for many tasks where information is the main ingredient such as planning, presentation, analysis, monitoring.

Cross-references

- ▶ Data Visualization
- ▶ Diagram
- ▶ Graph
- ▶ Map
- ▶ Table
- ▶ Thematic Map

Recommended Reading

1. Harris R.L. Information Graphics: A Comprehensive Illustrated Reference, Oxford University Press, New York/Oxford, 1999.
2. Tufte E.R. The Visual Display of Quantitative Information. Graphics Press, Cheshire, CT, 1983.

Chase

Alin Deutsch[1], Alan Nash[2]
[1]University of California-San Diego, La Jolla, CA, USA
[2]Aleph One LLC, La Jolla, CA, USA

Definition

The chase is a procedure that takes as input a set Σ of constraints and an instance I. The chase does not always terminate, but if it does it produces as output an instance U with the following properties:

1. $U \models \Sigma$; that is, U satisfies Σ.
2. $I \rightarrow U$; that is, there is a homomorphism from I to U.
3. For every instance J (finite or infinite), if $J \models \Sigma$ and $I \rightarrow J$, then $U \rightarrow J$.

In [7], an instance that satisfies (1) and (2) above is called a *model of* Σ *and* I and an instance that satisfies (3) above is called *strongly universal.*

In summary, the chase is a procedure which – whenever it terminates – yields a strongly-universal model.

Comments

1. The set Σ of constraints is usually a set of tuple-generating dependencies (tgds) and equality-generating dependencies (egds) [5], or, equivalently, embedded dependencies [5,10]. However, the chase has been extended to wider classes of constraints and to universality under functions other than homomorphisms [6,7,9]. In this case, the chase often produces a strongly-universal model *set* (see below), instead of a single model.

2. It was noted in [7] that in database applications, *weak universality* (condition 3 above restricted to finite instances) would suffice. Nevertheless, the chase gives strong universality.

Historical Background

The term "chase" was coined in [14], where it was used to test the logical implication of dependencies (i.e., whether all databases satisfying a set Σ of dependencies must also satisfy a given dependency σ). The implication problem was one of the key concerns of dependency theory, with applications to automatic schema design. The chase was defined in [14] for the classes of functional, join and multivalued dependencies. Related chase formulations for various kinds of dependencies were introduced in [15,17]. The work [5] unified the treatment of the implication problem for various dependency classes by introducing and defining the chase for tuple-generating and equality-generating dependencies (sufficiently expressive to capture all prior dependencies).

Ancestors of the chase (introduced as unnamed algorithms) appear in [2–4]. [4] introduces tableaux, a pattern-based representation for relational queries, and shows how to check the equivalence of tableau queries in the presence of functional dependencies, with applications to query optimization. To this end, the tableaux are modified using an algorithm that coincides with the chase with functional dependencies. The same algorithm is used in [3] for minimization of tableaux under functional dependencies. This algorithm is extended in [2] to include also multivalued dependencies, for the purpose of checking whether the join of several relations is lossless (i.e., the original relations can be retrieved as projections of the join result).

The chase was extended to include disjunction and inequality in [9], and to arbitrary ∀∃-sentences in [6]. Independently, [13] extended the chase to a particular case of disjunctive dependencies incorporating disjunctions of equalities between variables and constants (see also [12]). There are also extensions of the chase to deal with more complex data models beyond relational. The chase (and the language of embedded dependencies) is extended in [16] to work over complex values and dictionaries. For an excellent survey of the history of the chase prior to 1995, consult [1].

Foundations

A *tuple-generating dependency (tgd)* is a constraint σ of the form

$$\forall \bar{x}, \bar{y}\ (\alpha(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \beta(\bar{x}, \bar{z}))$$

where α and β are conjunctions of relational atoms. Furthermore, every variable in $\bar{x}$ appears in both α and β. The $\forall \bar{x}, \bar{y}$ prefix of universal quantifiers is usually omitted. If $\bar{z}$ is empty, then σ is *full*.

An *equality-generating dependency (egd)* is a constraint ϕ of the form

$$\forall x_1, x_2, \bar{y}\ (\alpha(x_1, x_2, \bar{y}) \rightarrow x_1 = x_2)$$

where α is a conjunction of relational atoms.

The chase is used on instances whose active domain consists of constants and *labeled nulls.* A *homomorphism* from A to B is denoted $A \rightarrow B$. It is a mapping h on the constants and nulls in A that (i) preserves constants (i.e., $h(c) = c$ for every constant c) and preserves relationships (i.e., for every tuple $R(x_1,...,x_n) \in A$, that is $R(h(x_1),...,h(x_n)) \in B$). Two instances A and B are *homomorphically equivalent* if $A \rightarrow B$ and $B \rightarrow A$.

The chase is a natural procedure for building strong universal models. Indeed, it turns out that checking for strong universality is undecidable as shown in [7]). In contrast, checking whether an instance is a model can be done efficiently. Therefore, it is natural to define any procedure for constructing strong universal models by steps which always preserve strong universality while attempting to obtain a model and then to check whether a model was indeed obtained. This is precisely what the chase does.

A tgd $\sigma \in \Sigma$ *fails* (or *applies*) on instance A and tuple $\bar{a}$ if there is tuple $\bar{b}$ in A such that the premise α of σ satisfies $A \vDash \alpha(\bar{a}, \bar{b})$, yet there is no tuple $\bar{c}$ in A such that the conclusion β of σ satisfies $A \vDash \beta(\bar{a}, \bar{c})$. Assume that the instance A' is obtained by adding to A the tuples in $\beta(\bar{a}, \bar{n})$ where $\bar{n}$ is a tuple of new nulls. Then A' is the result of *firing* σ on A, $\bar{a}$. Notice that $A \subseteq A'$ and that σ does not fail on A', $\bar{a}$. It is easy to verify that if A is strongly universal for Σ and I, then so is A' (towards this, it is essential that all the nulls in $\bar{n}$ be new and distinct).

An egd $\sigma \in \Sigma$ *fails* (or *applies*) on instance A and values a_1, a_2 if there is tuple $\bar{b}$ in A such that the premise α of σ satisfies $A \vDash \alpha(a_1, a_2, \bar{b})$, yet $a_1 \neq a_2$. If a_2 is a null a_2 is replaced everywhere in A with a_1

to obtain A', then say that A' is the result of *firing* σ on A, a_1, a_2. Notice that $A \rightarrow A'$ and that σ does not fail on A', a_1, a_1. It is easy to verify that if A is strongly universal for Σ and I, then so is A'. If a_2 is a constant, but a_1 is null, then it is possible to replace a_1 everywhere in A with a_2 instead. However, if both a_1 and a_2 are constants, then it is not possible to satisfy σ and preserve strong universality and the chase *fails.*

The standard chase procedure proceeds as follows.

1. Set $A_0 = I$.
2. Repeat the following:
   a. If A_n is a model of Σ and I, stop and return A_n.
   b. Otherwise, there must be either
      i. a tgd σ and $\bar{a}$ such that σ fails on A, $\bar{a}$, or
      ii. an egd σ' and a_1, a_2 such that σ' fails on A, a_1, a_2.

   Obtain A_{n+1} by picking one such σ and $\bar{a}$ and firing σ on A_n, $\bar{a}$, or by picking one such σ' and a_1, a_2 and firing σ' on A, a_1, a_2. (This is one *chase step* of the standard chase.)

Notice that, at every chase step, there may be a choice of σ and $\bar{a}$, respectively σ' and a_1, a_2. How these choices are picked is often left unspecified and in that case the standard chase is non-deterministic. The chase *terminates* if A_n is a model of Σ and I for some n.

The chase of instance I with tgds Σ produces a sequence of instances $I = A_0 \subseteq A_1 \subseteq A_2 \subseteq \ldots$ such that every A_i is strongly universal for Σ and I. The chase with tgds and egds produces a sequence $I = A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow \ldots$ such that every A_i is strongly universal for Σ and I. In the presence of egds, it is no longer the case that $A_i \subseteq A_j$ for $i \leq j$ and there is the additional complication that a chase step may fail. The chase for tgds and egds is described in more detail in [1].

Example 1 Consider the schema consisting of two relations:

1. employee Emp(ss#, name, dept#), with social security, name, and dept. number, and
2. department Dept(dept#, name, location, mgr#), with dept. number, name, location, and its manager's social security number.

Assume that Σ consists of the constraints

σ_1: dept# is a foreign key in Emp,
σ_2: mgr# is a foreign key in Dept, and
σ_3: every manager manages his own department.

(This omits the constraints that say that ss# is a key for Emp and that dept# is a key for Dept to keep the example simple.) These constraints can be written as follows (where σ_1 and σ_2 are tgds and σ_3 is an egd):

σ_1: $\mathrm{Dept}(d, e, \ell, m) \rightarrow \exists n, d' \mathrm{Emp}(m, n, d')$,
σ_2: $\mathrm{Emp}(s, n, d) \rightarrow \exists e, \ell, m \mathrm{Dept}(d, e, \ell, m)$, and
σ_3: $\mathrm{Dept}(d, e, \ell, m), \mathrm{Emp}(m, n, d') \rightarrow d = d'$.

Consider the initial instance

$$I_0 = \mathrm{Dept}(1, ''\mathrm{HR}'', ''\mathrm{somewhere}'', 333 - 33 - 3333)$$

containing a single tuple. Then in the first step of the chase, σ_1 fires, giving

$$I_1 = \{\mathrm{Dept}(1,\}\mathrm{HR}\},\}\mathrm{somewhere}\}, 33), \\ \mathrm{Emp}(33, \alpha, \beta)\}$$

where α and β are labeled nulls. In the second step, both σ_2 and σ_3 apply. If σ_3 fires, then β is set to 1 and yields

$$I_2 = \{\mathrm{Dept}(1, ''\mathrm{HR}'', ''\mathrm{somewhere}'', 33), \mathrm{Emp}(33, \alpha, 1)\}.$$

Since I_2 satisfies Σ, the chase terminates. However, if instead at the second step σ_2 fires, then it gives

$$I'_2 = \{\mathrm{Dept}(1, ''\mathrm{HR}'', ''\mathrm{somewhere}'', 33), \\ \mathrm{Emp}(33, \alpha, \beta),\ \mathrm{Dept}(\beta, \gamma, \delta, \epsilon\}$$

where γ, δ, and ε are new nulls. In this case, it is possible to continue firing σ_1, σ_2, and σ_3 in such a way as to obtain a chase that does not terminate, perpetually introducing new nulls.

If the standard chase (or any other chase listed below) terminates, it yields a strongly-universal model of Σ and I and it is straightforward to verify that all such models are homomorphically equivalent. Therefore the result of the standard chase is unique up to homomorphic equivalence. However, the choice of what constraint to fire and on what tuple may affect whether the chase terminates or not.

There are several variations of the chase, which shall be called here the *standard* chase, the *parallel* chase, and the *core* chase. The standard chase was described above. In the *parallel chase*, at every chase step σ is fired on A_n, $\bar{a}$ for all pairs $(\sigma, \bar{a})$ such that σ fails on A, $\bar{a}$.

One writes I^{Σ} for the result of the chase on Σ and I, if the chase terminates. In that case, one says that I^{Σ} is defined. In general, it holds that if $A \rightarrow B$, then $A^{\Sigma} \rightarrow B^{\Sigma}$, whenever the latter are defined.

It was shown in [7] that the standard chase is incomplete, in the following sense: it may be that Σ and I have a strongly-universal model, yet the standard chase does not terminate. The parallel chase is also incomplete in this sense. In contrast, the *core chase* introduced in [7] is complete: if a strongly universal-model exists, the core chase terminates and yields such a model. A chase step of the core chase consists of one chase step of the parallel chase, followed by computing the core of the resulting instance.

Any of the above mentioned variations of the chase can be applied to sets of constraints which consist of

1. tgds only
2. tgds and egds
3. tgds and egds with disjunctions
4. tgds and egds with disjunctions and negation which are equivalent to general $\forall\exists$ sentences

The chase with tgds and egds has been described above. The chase has been extended to handle disjunction and negation. In this case, it gives not a single model, but a set S of models which is strongly universal, in the sense that for any model J (finite or infinite) of Σ and I, there is a model $A \in S$ such that $A \rightarrow J$. Such a set arises from a single initial model by branching due to disjunction. For example, consider the set Σ with the single disjunctive tgd

$$\sigma : R(x) \rightarrow S(x) \vee T(x)$$

and the instance I containing the single fact $R(1)$. Clearly every model of Σ and I, must contain either $S(1)$ or $T(1)$. It is easy to verify that the set $S = \{I_1, I_2\}$ where $I_1 = \{R(1), S(1)\}$ and $I_2 = \{R(1), T(1)\}$ is strongly universal for I and Σ, but no proper subset of S is. The disjunctive chase with Σ on I consists of a single step, which produces not a single model, but the set S of models. Intuitively, whenever a disjunctive tgd fires on a set W of models, it produces, for every instance $A \in W$, one instance for every disjunct in its conclusion. For details, to see how negation is handled, and to see how universality for functions other than homomorphisms is achieved, see [6,7].

It was shown in [7] that it is undecidable whether the standard, parallel, or core chase with a set of tgds terminates. A widely-applicable, efficiently-checkable condition on a set Σ of tgds, which is sufficient to guarantee that the chase with Σ on any instance I terminates, was introduced in [9,11]. A set of tgds satisfying this condition is called *weakly acyclic* in [11] and is said to have *stratified witnesses* in [9]. A more widely-applicable condition, also sufficient for chase termination, was introduced in [7], where a set of tgds satisfying this condition is called *stratified*.

Key Applications

The chase has been used in many applications, including

- Checking containment of queries under constraints (which in turn is used in such query rewriting tasks as minimization, rewriting using views, and semantic optimization)
- Rewriting queries using views
- Checking implication of constraints
- Computing solutions to data exchange problems
- Computing certain answers in data integration settings

To check whether a query P is contained in a query Q under constraints Σ, written $P \sqsubseteq_\Sigma Q$, it is sufficient to (1) treat P as if it was an instance in which the free variables are constants and the bound variables are nulls (this is known as the "frozen instance" or "canonical database" [1] corresponding to P) (2) chase it with Σ, and if this chase terminates to yield P^Σ (3) check whether the result of this chase is contained in Q, written $P^\Sigma \sqsubseteq Q$. In symbols, if the chase described above terminates, then

$$P \sqsubseteq_\Sigma Q \text{ iff } P^\Sigma \sqsubseteq Q.$$

That is, the chase reduces the problem of query containment under constraints to one of query containment without constraints.

To check whether a set Σ of tgds implies a tgd σ of the form

$$\forall \bar{x}, \bar{y}(\alpha(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \beta(\bar{x}, \bar{z}))$$

which is logically equivalent to

$$\forall \bar{x}\,(\underbrace{\exists \bar{y} \alpha(\bar{x}, \bar{y})}_{Q_\alpha(\bar{x})} \rightarrow \underbrace{\exists \bar{z} \beta(\bar{x}, \bar{z})}_{Q_\beta(\bar{x})}),$$

it suffices to check whether the query Q_α in the premise of σ is contained under the constraints Σ in the query Q_β in the conclusion of σ. That is, if $Q_\alpha{}^\Sigma$ is defined, then

$$\Sigma \models \sigma \text{ iff } Q_\alpha \sqsubseteq_\Sigma Q_\beta \text{ iff } Q_\alpha^\Sigma \sqsubseteq Q_\beta.$$

The chase was also employed to find equivalent rewritings of conjunctive queries using conjunctive query views, in the presence of constraints. Given a set $\mathcal{V}$ of conjunctive query views and a conjunctive query Q, one can construct, using the chase, a query R expressed in terms of $\mathcal{V}$, such that Q has some equivalent rewriting using $\mathcal{V}$ if and only if R is itself such a rewriting. Moreover, every minimal rewriting of Q is guaranteed to be a sub-query of R. The algorithm for constructing R and exploring all its sub-queries is called the *Chase&Backchase (CB)* [8], and it is sound and complete for finding all minimal rewritings under a set Σ of embedded dependencies, provided the chase with Σ terminates [9]. The CB algorithm constructs R by simply (i) constructing a set $\Sigma_{\mathcal{V}}$ of tgds extracted from the view definitions, and (ii) chasing Q with $\Sigma \cup \Sigma_{\mathcal{V}}$ and restricting the resulting query to only the atoms using views in $\mathcal{V}$.

In [11] is was shown that the certain answers to a union Q of conjunctive queries on a ground instance I under a set Σ of source-to-target tgds and target tgds and egds can be obtained by computing $Q(U)$ – where U is a *universal solution* for I under Σ – then discarding any tuples with nulls. Universal solutions, which are the preferred solutions to materialize in data exchange, are closely related to strongly-universal models [7] and it was shown in [11] that they can be obtained using the chase.

Cross-references

- ▶ Data Exchange
- ▶ Data Integration
- ▶ Database Dependencies
- ▶ Equality-Generating Dependencies
- ▶ Query Containment
- ▶ Query Optimization
- ▶ Query Rewriting
- ▶ Query Rewriting Using Views
- ▶ Tuple-Generating Dependencies

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases. Addison Wesley, Reading, MA, 1995.
2. Aho A.V., Beeri C., and Ullman J.D. The theory of joins in relational databases. ACM Trans. Database Syst., 4(3):297–314, 1979.
3. Aho A.V., Sagiv Y., and Ullman J.D. Efficient optimization of a class of relational expressions. ACM Trans. Database Syst. 4(4):435–454, 1979.
4. Aho A.V., Sagiv Y., and Ullman J.D. Equivalence of relational expressions. SIAM J. Comput., 8(2):218–246, 1979.
5. Beeri C. and Vardi M.Y. A proof procedure for data dependencies. J. ACM, 31(4):718–741, 1984.
6. Deutsch A., Ludaescher B., and Nash A. Rewriting queries using views with access patterns under integrity constraints. In Proc. 10th Int. Conf. on Database Theory, 2005, pp. 352–367.
7. Deutsch A., Nash A., and Remmel J. The chase revisited. In Proc. 27th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2008, pp. 149–158.
8. Deutsch A., Popa L., and Tannen V. Physical Data Independence, Constraints, and Optimization with Universal Plans. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 459–470.
9. Deutsch A. and Tannen V. XML queries and constraints, containment and reformulation. Theor. Comput. Sci. 336(1): 57–87, 2005, preliminary version in ICDT 2003.
10. Fagin R. Horn clauses and database dependencies. J. ACM, 29(4):952–985, 1982.
11. Fagin R., Kolaitis P.G., Miller R.J., and Popa L. Data Exchange: Semantics and Query Answering. Theor. Comput. Sci., 336 (1):89–124, 2005, preliminary version in PODS 2005.
12. Fuxman A., Kolaitis P.G., Miller R.J., and Tan W.C. Peer Data Exchange. ACM Trans. Database Syst., 31(4):1454–1498, 2006, preliminary version in PODS 2005.
13. Grahne G. and Mendelzon A.O. Tableau Techniques for Querying Information Sources through Global Schemas. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 332–347.
14. Maier D., Mendelzon A.O., and Sagiv Y. Testing implications of data dependencies. ACM Trans. Database Syst., 4(4):455–469, 1979.
15. Maier D., Sagiv Y., and Yannakakis M. On the complexity of testing implication of functional and join dependencies. J. ACM, 28(4):680–695, 1981.
16. Popa L. and Tannen V. An Equational Chase for Path-Conjunctive Queries, Constraints, and Views. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 39–57.
17. Vardi M. Inferring multivalued dependencies from functional and join dependencies. Acta Informatica, 19:305–324, 1983.

Checkpoint

▶ Logging and Recovery

Checksum and Cyclic Redundancy Check Mechanism

KENICHI WADA
Hitachi, Ltd, Tokyo, Japan

Synonyms

Cyclic Redundancy Check (CRC)

Definition

Checksum and CRC are schemes for detecting the errors of data which occur during transmission or storage. The data computed and appended to original data in order to detect errors are also referred as checksum and CRC.

A checksum consists of a fixed number of bits computed as a function of the data to be protected, and is appended to the data. To detect errors, the function is recomputed, and the result is compared to that appended to the data. Simple implementation of checksum is to divide the data into same length bits chunk and to make exclusive-or of all chunks. Cyclic redundancy check mechanism exploits mathematical properties of cyclic codes. Specifically, CRC uses polynomial devisor circuits with a given generator polynomial so as to obtain the remainder polynomial. The remainder is similarly appended to the original data for transmission and storage, and then utilized for error detection. CRC can be used as a kind of checksum.

Key Points

CRC is usually expressed by the use of binary polynomials due to mathematical convenience. When original data $M(x)$ is given, basic CRC mechanism calculates redundancy data $R(x)$ by using a pre-defined generator polynomial $G(x)$. That is, supposing the degree of $G(x)$ is m, a polynomial $M(x) * x^m$ is divided by $G(x)$ and the remainder is used for $R(x)$ such that a concatenated polynomial $T(x) = M(x) * x^m + R(x)$ is divisible by $G(x)$. The obtained $T(x)$ is used for transmission or storage. For error detection, CRC mechanism similarly checks the divisibility of $T(x)$ by $G(x)$. These encoding and detection processes can be implemented by using multi-level shift register circuits.

Given below is an example of CRC calculation. Assume that a generator polynomial and original data are given as follows.

$$\mathrm{G}(x) = x^3 + x + 1 \quad (\textit{binary expression}: 1011)$$

$$M(x) = x^4 + 1 \quad (10001)$$

In this case, a remainder polynomial $R(x)$ can be obtained by dividing $M(x) * x^3$ by $G(x)$.

$$R(x) = x \quad (010)$$

Therefore, the resulting data $T(x)$ can be obtained as follows.

$$T(x) = x^7 + x^3 + x \quad (10001010)$$

Theoretically, CRC is capable of detecting m-bit long or shorter bust errors. This property is suitable for communication infrastructure and storage infrastructure, which often introduce burst errors rather than random errors.

Cross-references

► Disk

Recommended Reading

1. Houghton A. Error Coding for Engineers. Kluwer Academic Publishers, Dordrecht, 2001.
2. Sweeney P. Error Control Coding from Theory to Practice. Wiley, NY, 2002.

Choreography

W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven, The Netherlands

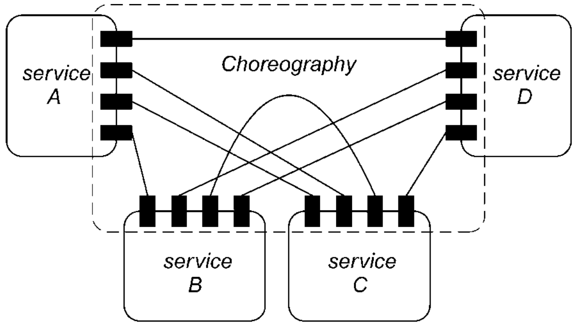
Definition

In a service oriented architecture (SOA) services are interacting by exchanging messages, i.e., by combining services more complex services are created. Choreography is concerned with the composition of such services seen from a global viewpoint focusing on the common and complementary observable behavior. Choreography is particulary relevant in a setting where there is not a single coordinator.

Key Points

The terms orchestration and choreography describe two aspects of integrating services to create business processes [1,3]. The two terms overlap somewhat and the distinction is subject to discussion. Orchestration and choreography can be seen as different "perspectives." Choreography is concerned with the exchange of messages between those services. Orchestration is concerned with the interactions of a single service with its environment.

Figure 1 illustrates the notion of choreography. The dashed area shows the focal point of choreography,



Choreography. Figure 1. Choreography.

i.e., the aim is to establish a “contract” containing a “global” definition of the constraints under which messages are exchanged. Unlike orchestration, the viewpoint is not limited to a single service. The Web Services Choreography Description Language (WS-CDL, cf. [2]) and the Web Service Choreography Interface (WSCI) are two languages aiming at choreography. Since the focus is on agreement rather than enactment, choreography is quite different from traditional workflow languages. The goal is not to control and enact but to coordinate autonomous parties. Some characterize choreography as “Dancers dance following a global scenario without a single point of control” to emphasize this distinction.

Cross-references

- ▶ BPEL
- ▶ Business Process Management
- ▶ Orchestration
- ▶ Web Services
- ▶ Workflow Management

Recommended Reading

1. Dumas M., van der Aalst W.M.P., and ter Hofstede A.H.M. Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley, New York, 2005.
2. Kavantzas N., Burdett D., Ritzinger G., Fletcher T., and Lafon Y. Web Services Choreography Description Language Version 1.0 (W3C Candidate Recommendation). http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/, 2005.
3. Weske M. Business Process Management: Concepts, Languages, Architectures. Springer, Berlin, 2007.

Chronicle Recognition

▶ Event Detection

Chronon

Curtis Dyreson
Utah State University, Logan, UT, USA

Synonyms

Instant; Moment; Time quantum; Time unit

Definition

A chronon is the smallest, discrete, non-decomposable unit of time in a temporal data model. In a one-dimensional model, a chronon is a *time interval* or *period*, while in an *n*-dimensional model it is a non-decomposable region in *n*-dimensional time. Important special types of chronons include valid-time, transaction-time, and bitemporal chronons.

Key Points

Data models often represent a time line by a sequence of non-decomposable, consecutive time periods of identical duration. These periods are termed chronons. A data model will typically leave the size of each particular chronon unspecified. The size (e.g., one microsecond) will be fixed later by an individual application or by a database management system, within the restrictions posed by the implementation of the data model. The number of chronons is finite in a bounded model (i.e., a model with a minimum and maximum chronon), or countably infinite otherwise. Consecutive chronons may be grouped into larger intervals or segments, termed *granules*; a chronon is a granule at the lowest possible granularity.

Cross-references

- ▶ Temporal Granularity
- ▶ Time Domain
- ▶ Time Instant
- ▶ Time Interval

Recommended Reading

1. Dyreson C.E. and Snodgrass R.T. The base-line clock. In The TSQLZ temporal query language, Kluwer, pp. 73–92, 1987.
2. Dyreson C.E. and Snodgrass R.T. Timestamp Semantics and Representation. Inf. Syst., 18(3):143–166, 1993.

CIFS

▶ Storage Protocols

Cipher

► Data Encryption

Citation

Prasenjit Mitra
The Pennsylvania State University, University Park, PA, USA

Synonyms

Reference; Bibliography

Definition

A citation is a reference from one article to another article. A citation is a record that consists of the names of the authors, the title of the referred article, the time and place of publication, as well as various other fields. The fields in the citation should collectively specify unambiguously where the full text of the referred article could be obtained. Typically, all citations are presented at the end of the referring article. However, articles in certain domains list the citations as footnotes in the pages where the reference occurs. Citations can range from references to be to single articles or to entire books.

Key Points

Often, authors have to refer to knowledge that is derived from another work. For example, when quoting text from another article or book, the author must specify from which article or book the quotation is obtained. Authors need to refer to other works in order to point out preliminary information on the shoulders of which the current treatise stands, to refer to related work and contrast the current work with previous works, etc. A citation is used for primarily two purposes: (i) to provide a reference to an article or book such that the reader can retrieve the article or book easily and read the article to gain additional knowledge, and (ii) to provide credit (or discredit for "negative" citations) to the authors of the works that are being cited.

There are various widely used formats for citations. Citation formats vary by discipline; typically a discipline adheres to one (or a few) "style-guide" that indicates what fields should be mentioned in a citation and how the fields should be formatted and presented. Recently, with the proliferation of electronic documents published over the World-Wide-Web, citations to Uniform Resource Locators (URLs) of websites are increasingly common. Unlike printed articles and books, websites are dynamic and can change frequently. Therefore, in order to specify precisely which version of the webpage was being referred, apart from the publication date, authors usually provide the date on which the website was accessed.

Citations analysis has been performed to identify the impact of published articles. Because different authors use different formats, automatic analysis of citations requires *citation matching*. Citation matching helps identify which different citations formatted differently refer to the same article or book. The term *bibliometrics* is used to refer to metrics designed based on citation analysis. Citation indexing for academic journals was popularized by Eugene Garfield [1,2]. A citation index contains the information about which document cites which. The term *co-citation* refers to the frequency with which two documents are cited together [3]. Today, Google Scholar (http://scholar.google.com) provides a readily-available collection of indexed citations on the web.

Cross-references

► Digital Library

Recommended Reading

1. Garfield E. Citation Indexing: Its Theory and Application in Science, Technology, and Humanities. Wiley, New York, NY, USA, 1979.
2. Garfield E. Citation analysis as a tool in journal evaluation: journals can be ranked by frequency and impact of citations for science policy studies. Science, 178(4060):471–479, 1972.
3. Small H. Co-citation in the scientific literature: a new measure of the relationship between two documents. J. Am. Soc. Inf. Sci., Wiley Periodicals, 24(4):265–269, 1973.

CLARA (Clustering LARge Applications)

► K-Means and K-Medoids

CLARANS (Clustering Large Applications Based Upon Randomized Search)

► K-Means and K-Medoids

Classification

Ian H. Witten
University of Waikato, Hamilton, New Zealand

Synonyms

Classification learning; Supervised learning; Learning with a teacher, Concept learning; Statistical decision techniques

Definition

In *Classification learning*, an algorithm is presented with a set of classified examples or "instances" from which it is expected to infer a way of classifying unseen instances into one of several "classes". Instances have a set of features or "attributes" whose values define that particular instance. Numeric prediction, or "regression," is a variant of classification learning in which the class attribute is numeric rather than categorical. Classification learning is sometimes called *supervised* because the method operates under supervision by being provided with the actual outcome for each of the training instances. This contrasts with clustering where the classes are not given, and with association learning which seeks any association – not just one that predicts the class.

Historical Background

Classification learning grew out of two strands of work that began in the 1950s and were actively pursued throughout the 1960s: statistical decision techniques and the Perceptron model of neural networks. In 1955, statisticians Bush and Mosteller published a seminal book *Stochastic Models for Learning* which modeled in mathematical terms the psychologist B. F. Skinner's experimental analyses of animal behavior using reinforcement learning [2]. The "perceptron" was a one-level linear classification scheme developed by Rosenblatt around 1957 and published in his book *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms* [10]. In a response published in 1969, Minsky and Papert argued that perceptrons were simplistic in terms of their representational capability and had been greatly over-hyped as potentially universal learning machines [6]. This scathing response by widely-respected artificial intelligence pioneers dampened research in neural nets and machine learning in general. Meanwhile, in 1957 others were investigating the application of Bayesian decision schemes to pattern recognition; the general conclusion was that full Bayesian models were prohibitively expensive. In 1960 Maron investigated in the context of information retrieval what has since become known as the "naïve Bayes" approach, which assumes independence between attributes notwithstanding overwhelming evidence to the contrary [5]. Other early machine learning work was buried in cybernetics, the study of feedback and derived concepts such as communication and control in living and artificial organisms. Throughout the 1960s classification learning applied to pattern recognition was the central thread of the embryo field of machine learning, as underlined by the subtitle of Nilsson's 1965 landmark book *Learning Machines – Foundations of Trainable Pattern-Classifying Systems* [7].

Symbolic learning techniques began to recover from the doldrums in the late 1970s, with influential and almost simultaneous publications by Breiman et al. on *classification and regression trees* (the CART system) [1] and Quinlan on *decision tree induction* (the ID3 and later C4.5 systems) [8,9]. Whereas Breiman was a statistician, Quinlan was an experimental computer scientist who first used decision trees not to generalize but to condense large collections of chess end-games. Their work proceeded independently, and the similarities remained unnoticed until years later. CART (by default) produces multivariate trees whose tests can involve more than one attribute: these are more accurate and smaller than the univariate trees produced by Quinlan's systems, but take longer to generate.

The first workshop devoted to machine learning was held in 1980 at Carnegie-Mellon University. Further workshops followed in 1983 and 1985. These invitation-only events became an open conference in 1988. Meanwhile the journal *Machine Learning* was established in 1986. By the 1990s the subject had become the poster child of artificial intelligence – a successful, burgeoning, practical technology that eschewed the classical topics of general knowledge representation, logical deduction, theorem proving, search techniques, computational linguistics, expert systems and philosophical foundations that still characterize the field today. Classification learning, which forms the core of machine learning, outgrew its behaviorist and neurological roots and moved into the practical realm of database systems.

Early work focused on the *process* of learning – learning curves, the possibility of sustained learning, and the like – rather than the *results* of learning. However, with the new emphasis on applications, objective techniques of empirical testing began to supplant the scenario-based style of evaluation that characterized the early days. A major breakthrough came during the 1980s, when researchers finally realized that evaluating a learning system on its training data gave misleading results, and instead put the subject on a secure statistical footing.

Foundations

One of the most instructive lessons learned since the renaissance of classification in the 1980s is that simple schemes often work very well. Today, practitioners strongly recommend the adoption of a "simplicity-first" methodology when analyzing practical datasets. There are many different kinds of simple structure that datasets can exhibit. One dataset might have a single attribute that does all the work, the others being irrelevant or redundant. Alternatively, the attributes might contribute independently and equally to the final outcome. Underlying a third dataset might be a simple contingent structure involving just a few attributes. In a fourth, a few independent rules may govern the assignment of instances to classes. In a fifth, classifications appropriate to particular regions of instance space might depend on the distance between the instances themselves. A sixth might exhibit dependence among numeric attributes, determined by a sum of attribute values with appropriately chosen weights. This sum might represent the final output for numeric prediction, or be compared to a fixed threshold in a binary decision setting. Each of these examples leads to a different style of method suited to discovering that kind of structure.

Rules Based on a Single Attribute

Even when instances have several attributes, the classification decision may rest on the value of just one of them. Such a structure constitutes a set of rules that all test the same attribute (or, equivalently, a one-level decision tree). It can be found by evaluating the success, in terms of the total number of errors on the training data, of testing each attribute in turn, predicting the most prevalent class for each value of that attribute. If an attribute has many possible values – and particularly if it has numeric values – this may "overfit" the training data by generating a rule that has almost as many branches as there are instances. Minor modifications to the scheme overcome this problem.

A startling discovery published in 1993 was that "very simple classification rules perform well on most commonly used datasets" [3]. In an empirical investigation of the accuracy of rules that classify instances on the basis of a single attribute, on most standard datasets the resulting rule was found to be as accurate as the structures induced by the majority of machine learning systems – which are far more complicated. The moral? – always compare new methods with simple baseline schemes.

Statistical Modeling (see entry Bayesian Classification)

Another simple technique is to use all attributes and allow them to make contributions to the decision that are *equally important* and *independent* of one another, given the class. Although grossly unrealistic – what makes real-life datasets interesting is that the attributes are certainly not equally important or independent – it leads to a statistically-based scheme that works surprisingly well in practice. Employed in information retrieval as early as 1960 [5], the idea was rediscovered, dubbed "naïve Bayes," and introduced into machine learning 30 years later [4]. Despite the disparaging moniker it works well on many actual datasets. Over-reliance on the independence of attributes can be countered by applying attribute selection techniques.

Divide and Conquer Technique (see entry Decision Tree Classification)

The process of constructing a decision tree can be expressed recursively. First, select an attribute to use at the root, and make a branch for each possible value. This splits the instance set into subsets, one for every value of the attribute. Now repeat the process recursively for each branch, using only those instances that actually reach the branch. If all instances at a node have the same classification, stop developing that part of the tree. This method of "top-down induction of decision trees" was explored and popularized by Quinlan [8,9]. The nub of the problem is to select an appropriate attribute at each stage. Of many heuristics that have been investigated, the dominant one is to measure the expected amount of information gained by knowing that attribute's actual value. Having generated the tree, it is selectively pruned back from the leaves to avoid over-fitting. A series of improvements include ways of dealing with numeric attributes, missing values, and noisy data; and generating rules from trees.

Covering Algorithms (see entry Rule-Based Classification)

Classification rules can be produced by taking each class in turn and seeking a rule that covers all its instances, at the same time excluding instances not in the class. This bottom-up approach is called *covering* because at each stage a rule is identified that "covers" some of the instances. Although trees can always be converted into an equivalent rule set, and vice versa, the perspicuity of the representation often differs. Rules can be symmetric whereas trees must select one attribute to split on first, which can produce trees that are much larger than an equivalent set of rules. In the multiclass case a decision tree split takes account of all classes and maximizes the information gained, whereas many rule generation methods concentrate on one class at a time, disregarding what happens to the others.

Instance-Based Learning (see entry Nearest Neighbor Classification)

Another approach is to store training instances verbatim and, given an unknown test instance, use a distance function to determine the closest training instance and predict its class for the test instance. Suitable distance functions are the Euclidean or Manhattan (city-block) metric; attributes should be normalized to lie between 0 and 1 to compensate for scaling effects. For nominal attributes that assume symbolic rather than numeric values, the distance between two values is 1 if they are not the same and 0 otherwise. In the *k*-nearest neighbor strategy, some fixed number of nearest neighbors – say five – are located and used together to determine the class of the test instance by majority vote. Another way of proofing the database against noise is to selectively and judiciously choose the exemplars that are added. Nearest-neighbor classification was notoriously slow until advanced data structures like *k*D-trees were applied in the early 1990s.

Linear Models (see entry Linear Regression)

When the outcome and all attributes are numeric, linear regression can be used. This expresses the class as a linear combination of the attributes, with weights that are calculated from the training data. Linear regression has been popular in statistical applications for decades. If the data exhibits a nonlinear dependency, the best-fitting straight line will be found, where "best" is interpreted in the least-mean-squared-difference sense. Although this line may fit poorly, linear models can serve as building blocks for more complex learning schemes.

Linear Classification (see entry Neural Networks, Support Vector Machine)

The idea of linear classification is to find a hyperplane in instance space that separates two classes. (In the multiclass case, a binary decision can be learned for each pair of classes.) If the linear sum exceeds zero the first class is predicted; otherwise the second is predicted. If the data is linearly separable – that is, it can be separated perfectly using a hyperplane – the perceptron learning rule espoused by Rosenblatt is guaranteed to find a separating hyperplane [10]. This rule adjusts the weight vector whenever the prediction for a particular instance is erroneous: if the first class is predicted the instance (expressed as a vector) is added to the weight vector (making it more likely that the result will be positive next time around); otherwise the instance is subtracted.

There have been many powerful extensions of this basic idea. Support vector machines use linear decisions to implement nonlinear class boundaries by transforming the input using a nonlinear mapping. Multilayer perceptrons connect many linear models in a hierarchical arrangement that can represent nonlinear decision boundaries, and use a technique called "back-propagation" to distribute the effect of errors through this hierarchy during training.

Missing Values

Most datasets encountered in practice contain missing values. Sometimes different kinds are distinguished (e.g., unknown vs. unrecorded vs. irrelevant values). They may occur for a variety of reasons. There may be some significance in the fact that a certain instance has an attribute value missing – perhaps a decision was taken not to perform some test – and that might convey information about the instance other than the mere absence of the value. If this is the case, *not tested* should be recorded as another possible value for this attribute. Only someone familiar with the data can make an informed judgment as to whether a particular value being missing has some significance or should simply be coded as an ordinary missing value. For example, researchers analyzing medical databases have noticed that cases may, in some circumstances, be diagnosable strictly from the tests that a doctor decides to make, regardless of the outcome of the tests. Then a record of which values are "missing" is all that is needed for a complete diagnosis – the actual measurements can be ignored entirely!

Meta-Learning

Decisions can often be improved by combining the output of several different models. Over the past decade or so the techniques of *bagging*, *boosting*, and *stacking* have been developed that learn an ensemble of models and deploy them together. Their performance is often astonishingly good. Researchers have struggled to understand why, and during that struggle new methods have emerged that are sometimes even better. For example, whereas human committees rarely benefit from noisy distractions, shaking up bagging by adding random variants of classifiers can improve performance. Boosting – perhaps the most powerful of the three methods – is related to the established statistical technique of additive models, and this realization has led to improved procedures.

Combined models share the disadvantage of being rather hard to analyze: they can comprise dozens or even hundreds of individual learners and it is not easy to understand in intuitive terms what factors are contributing to the improved decisions. In the last few years methods have been developed that combine the performance benefits of committees with comprehensible models. Some produce standard decision tree models; others introduce new variants of trees that provide optional paths.

Evaluation

For classification problems, performance is naturally measured in terms of the *error rate*. The classifier predicts the class of each test instance: if it is correct, that is counted as a success; if not, it is an error. The error rate is the proportion of errors made over a whole set of instances, and reflects the overall performance of the classifier. Performance on the training set is definitely *not* a good indicator of expected performance on an independent test set. A classifier is *overfitted* to a dataset if its structure reflects that particular set to an excessive degree. For example, the classifier might be generated by rote learning without any generalization whatsoever. An overfitted classifier usually exhibits performance on the training set which is excellent but far from representative of performance on other datasets from the same source.

In practice, one must predict performance bounds based on experiments with whatever data is available. Labeled data is required for both training and testing, and is often hard to obtain. A single data set can be partitioned for training and testing in various different ways. In a popular statistical technique called *cross-validation* the experimenter first decides on a fixed number of "folds," or partitions of the data – say three. The data is split into three approximately equal portions, and each in turn is used for testing while the remainder serves for training. The procedure is repeated three times so that in the end every instance has been used exactly once for testing. This is called *threefold cross-validation*. "Stratification" is the idea of ensuring that all classes are represented in all folds in approximately the right proportions. *Stratified tenfold cross-validation* has become a common standard for estimating the error rate of a classification learning scheme. Alternatives include *leave-one-out* cross-validation, which is effectively *n*-fold cross-validation where *n* is the size of the data set; and the *bootstrap*, which takes a carefully-judged number of random samples from the data with replacement and uses these for training, combining the error rate on the training data (an optimistic estimate) with that on the test data (a pessimistic estimate, since the classifier has only been trained on a subset of the full data) to get an overall estimate.

Key Applications

Classification learning is one of the flagship triumphs of research in artificial intelligence. It has been used for problems that range from selecting promising embryos to implant in a human womb during in vitro fertilization to the selection of which cows in a herd to sell off to an abattoir. Fielded applications are legion. They include decisions involving judgment, such as whether a credit company should make a loan to a particular person; screening images, such as the detection of oil slicks from satellite images; load forecasting, such as combining historical load information with current weather conditions and other events to predict hourly demand for electricity; diagnosis, such as fault finding and preventative maintenance of electromechanical devices; marketing and sales, such as detecting customers who are likely to switch to a competitor.

URL to Code

The Weka machine learning workbench is a popular tool for experimental investigation and comparison of classification learning techniques, as well as other machine learning methods. It is described in [11] and available for download from http://www.cs.waikato.ac.nz/ml/weka.

Cross-references

- ▶ Abstraction
- ▶ Association Rules
- ▶ Bagging
- ▶ Bayesian Classification
- ▶ Boosting
- ▶ Bootstrap
- ▶ Cataloging in Digital Libraries
- ▶ Classification by Association Rule Analysis
- ▶ Clustering Overview and Applications
- ▶ Cross-Validation
- ▶ Data Mining
- ▶ Decision Rule Mining in Rough Set Theory
- ▶ Decision Tree Classification
- ▶ Fuzzy Set Approach
- ▶ Genetic Algorithms
- ▶ Linear Regression
- ▶ Log-Linear Regression
- ▶ Nearest Neighbor Classification
- ▶ Neural Networks
- ▶ Receiver Operating Characteristic
- ▶ Rule-Based classification
- ▶ Support Vector Machine

Recommended Reading

1. Breiman L., Friedman J.H., Olshen R.A., and Stone C.J. Classification and Regression Trees. Wadsworth, Pacific Grove, CA, 1984.
2. Bush R.R. and Mosteller F. Stochastic Models for Learning. Wiley, New York, 1955.
3. Holte R.C. Very simple classification rules perform well on most commonly used datasets. Mach. Learn., 11:63–91, 1993.
4. Kononebko I. ID3, sequential Bayes, naïve Bayes and Bayesian neural networks. In Proc. 4th European Working Session on Learning, 1989, pp. 91–98.
5. Maron M.E. and Kuhns J.L. On relevance, probabilistic indexing and information retrieval. J. ACM, 7(3):216–244, 1960.
6. Minsky M.L. and Papert S. Perceptrons. Cambridge, MIT Press, 1969.
7. Nilsson N.J. Learning Machines. McGraw-Hill, New York, 1965.
8. Quinlan J.R. Induction of decision trees. Mach. Learn., 1(1):81–106, 1986.
9. Quinlan J.R. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco, CA, 1993.
10. Rosenblatt F. Principles of Neurodynamics. Spartan, Washington, DC, 1961.
11. Witten I.H. and Frank E. Data Mining: Practical Machine Learning Tools and Techniques (2nd edn.). Morgan Kaufmann, San Francisco, CA, 2003.

Classification by Association Rule Analysis

Bing Liu
University of Illinois at Chicago, Chicago, IL, USA

Synonyms

Associative classification

Definition

Given a training dataset *D*, build a classifier (or a classification model) from *D* using an association rule mining algorithm. The model can be used to classify future or test cases.

Historical Background

In the previous section, it was shown that a list of rules can be induced or mined from the data for classification. A decision tree may also be converted to a set of rules. It is thus only natural to expect that association rules [1] be used for classification as well. Yes, indeed! Since the first classification system (called CBA) that used association rules was reported in [10], many techniques and systems have been proposed by researchers [2–4,6–8,13,15,16]. CBA is based on class association rules (CAR), which are a special type of association rules with only a class label on the right-hand-side of each rule. Thus, syntactically or semantically there is no difference between a rule generated by a class association rule miner and a rule generated by a rule induction system (or a decision tree system for that matter). However, class association rule mining inherits the completeness property of association rule mining [1]. That is, all rules that satisfy the user-specified minimum support and minimum conference are generated. Other classification algorithms only generate a small subset of rules existing in data for classification [9,10].

Most existing classification systems based on association rules (also called *associative classifiers*) employ CARs directly for classification, although their ways of using CARs can be quite different [3,7,8,10,15,16].

To deal with unbalanced class distributions, the multiple minimum class supports approach is proposed in [9,11], which gives each class a different minimum support based on its relative frequency in the data. In [2,4,6,13], the authors also proposed to use rules as features or attributes to augment the original data or even to replace the original data. That is, in these techniques, CARs are not directly used for classification, but are used only to expand or to replace the original data. Any classification technique can be used subsequently to build the final classifier based on the expanded data, e.g., naïve Bayesian and SVM. Since the number of class association rules can be huge, closed rule sets have been proposed for classification in [3]. This approach helps solve the problem that in many data sets the complete sets of CARs cannot be generated due to combinatorial explosion. The closed rule set is a smaller, lossless and concise representation of all rules. Thus, long rules (rules with many conditions) may be used in classification, which otherwise may not be generated but can be crucial for accurate classification. Finally, normal association rules may be used for prediction or classification as well.

This section thus introduces the following three approaches to using association rules for classification:

1. Using class association rules for classification
2. Using class association rules as features or attributes
3. Using normal association rules for classification

The first two approaches can be applied to tabular data or transactional data. The last approach is usually employed for transactional data only. Transactional data sets are difficult to handle by traditional classification techniques, but are very natural for association rules. Below, the three approaches are described in turn. Note that various sequential rules can be used for classification in similar ways as well if sequential data sets are involved [6].

Foundations

Classification Using Class Association Rules

As mentioned above, a class association rule (CAR) is an association rule with only a class label on the right-hand side of the rule. Any association rule mining algorithm can be adapted for mining CARs. For example, the Apriori algorithm [1] for association rule mining was adapted to mine CARs in [10].

There is basically no difference between rules generated from a decision tree (or a rule induction system) and CARs if only categorical (or discrete) attributes (more on this later) are considered. The differences are in the mining processes and the final rule sets. CAR mining finds all rules in data that satisfy the user-specified minimum support (minsup) and minimum confidence (minconf) constraints. A decision tree or a rule induction system finds only a subset of the rules (expressed as a tree or a list of rules) for classification. In many cases, rules that are not in the decision tree (or the rule list) may be able to perform the classification more accurately. Empirical comparisons reported by several researchers have shown that classification using CARs can perform more accurately on many data sets than decision trees and rule induction systems [7,8,10,15,16].

The complete set of rules from CAR mining is also beneficial from the rule usage point of view. In many applications, the user wants to act on some interesting rules. For example, in an application for finding causes of product problems in a manufacturing company, more rules are preferred to fewer rules because with more rules, the user is more likely to find rules that indicate causes of problems. Such rules may not be found by a decision tree or a rule induction system. A deployed data mining system based on CARs is reported in [12] for finding actionable knowledge from manufacturing and engineering data sets.

One should, however, also bear in mind of the following differences between CAR mining and decision tree construction (or rule induction):

1. Decision tree learning and rule induction do not use the minsup or minconf constraint. Thus, some rules that they find can have very low supports, which, of course, are likely to be pruned because the chance that they overfit the training data is high. Although a low minsup for CAR mining can be used, it may cause combinatorial explosion. In practice, in addition to minsup and minconf, a limit on the total number of rules to be generated may be used to further control the CAR generation process. When the number of generated rules reaches the limit, the algorithm stops. However, with this limit, long rules (with many conditions) may not be generated. Recall that the Apriori algorithm works in a level-wise fashion, i.e., short

rules are generated before long rules. In some applications, this may not be an issue as short rules are often preferred and are sufficient for classification or for action. Long rules normally have very low supports and tend to overfit the data. However, in some other applications, long rules can be useful.

2. CAR mining does not use continuous (numeric) attributes, while decision trees deal with continuous attributes naturally. Rule induction can use continuous attributes as well. There is still no satisfactory method to deal with such attributes directly in association rule mining. Fortunately, many attribute discretization algorithms exist that can automatically discretize the value range of a continuous attribute into suitable intervals [e.g., [5]], which are then considered as discrete values.

Mining Class Association Rules for Classification There are many techniques that use CARs to build classifiers. Before describing them, it is useful to first discuss some issues related to CAR mining for classification.

Rule pruning: CAR rules are highly redundant, and many of them are not statistically significant (which can cause overfitting). Rule pruning is thus needed. The idea of pruning CARs is basically the same as tree pruning in decision tree building or rule pruning in rule induction. Thus, it will not be discussed further (see [8,10] for some of the pruning methods).

Multiple minimum class supports: A single minsup may be inadequate for mining CARs because many practical classification data sets have uneven class distributions, i.e., some classes cover a large proportion of the data, while others cover only a very small proportion (which are called *rare* or *infrequent classes*).

For example, there is a data set with two classes, *Y* and *N*. 99% of the data belong to the *Y* class, and only 1% of the data belong to the *N* class. If the minsup is set to 1.5%, no rule for class *N* will be found. To solve the problem, the minsup needs to be lowered. Suppose the minsup is set to 0.2%. Then, a huge number of overfitting rules for class *Y* may be found because minsup = 0.2% is too low for class *Y*.

Multiple minimum class supports can be applied to deal with the problem. A different *minimum class support* $minsup_i$ for each class c_i can be assigned, i.e., all the rules of class c_i must satisfy $minsup_i$. Alternatively, one single total minsup can be provided, denoted by *t_minsup*, which is then distributed to each class according to the class distribution:

$$minsup_i = t\_minsup \times sup(c_i)$$

where $sup(c_i)$ is the support of class c_i in the training data. The formula gives frequent classes higher minsups and infrequent classes lower minsups. There is also a general algorithm for mining normal association rules using multiple minimum supports in [9,11].

Parameter selection: The parameters used in CAR mining are the minimum supports and the minimum confidences. Note that a different minimum confidence may also be used for each class. However, minimum confidences do not affect the classification much because classifiers tend to use high confidence rules. One minimum confidence is sufficient as long as it is not set too high. To determine the best $minsup_i$ for each class c_i, a range of values can be tried to build classifiers and then use a validation set to select the final value. Cross-validation may be used as well.

Classifier Building After all CAR rules are found, a classifier is built using the rules. There are many existing approaches, which can be grouped into three categories.

Use the strongest rule: This is perhaps the simplest strategy. It simply uses CARs directly for classification. For each test instance, it finds the strongest rule that covers the instance. A rule *covers* an instance if the instance satisfies the conditions of the rule. The class of the strongest rule is then assigned as the class of the test instance. The strength of a rule can be measured in various ways, e.g., based on confidence, χ^2 test, or a combination of both support and confidence values.

Select a subset of the rules to build a classifier: The representative method of this category is the one used in the CBA system [10]. The method is similar to the sequential covering method, but applied to class association rules with additional enhancements as discussed above.

Let the set of all discovered CARs be *S*. Let the training data set be *D*. The basic idea is to select a subset L ($\subseteq S$) of high confidence rules to cover *D*. The set of selected rules, including a default class, is then used as the classifier. The selection of rules is based on a total order defined on the rules in *S*.

Definition: *Given two rules, r_i and r_j, $r_i \succ r_j$(called r_i precedes r_j, or r_i has a higher precedence than r_j) if*

1. *The confidence of $r_i$is greater than that of r_j, or*
2. *Their confidences are the same, but the support of $r_i$is greater than that of r_j, or*
3. *Both the confidences and supports of $r_i$and r_j are the same, but r_i is generated earlier than r_j.*

A CBA classifier L is of the form:

$L = < r_1, r_2, \ldots, r_k, \textit{default-class}>$

where $r_i \in S$, $r_a \succ r_b$ if $b > a$. In classifying a test case, the first rule that satisfies the case classifies it. If no rule applies to the case, it takes the default class (*default-class*). A simplified version of the algorithm for building such a classifier is given in Fig. 1. The classifier is the *RuleList*.

This algorithm can be easily implemented by making one pass through the training data for every rule. However, this is extremely inefficient for large data sets. An efficient algorithm that makes at most two passes over the data is given in [10].

Combine multiple rules: Like the first approach, this approach does not have an additional step to build a classifier. At the classification time, for each test instance, the system first finds the subset of rules that covers the instance. If all the rules in the subset have the same class, the class is assigned to the test instance. If the rules have different classes, the system divides the rules into groups according to their classes, i.e., all rules of the same class are in the same group. The system then compares the aggregated effects of the rule groups and finds the strongest group. The class label of the strongest group is assigned to the test instance. To measure the strength of a rule group, there again can be many possible techniques. For example, the CMAR system uses a weighted χ^2 measure [8].

Class Association Rules as Features

In the above two approaches, rules are directly used for classification. In this approach, rules are used as features to augment the original data or simply form a new data set, which is then fed to a traditional classification algorithm, e.g., decision trees or the naïve Bayesian algorithm.

To use CARs as features, only the conditional part of each rule is needed, and it is often treated as a Boolean feature/attribute. If a data instance in the original data contains the conditional part, the value of the feature/attribute is set to 1, and 0 otherwise. Several applications of this method have been reported [2,4,6,13]. The reason that this approach is helpful is that CARs capture multi-attribute or multi-item correlations with class labels. Many classification algorithms do not find such correlations (e.g., the naïve Bayesian method), but they can be quite useful.

Classification Using Normal Association Rules

Not only can class association rules be used for classification, but also normal association rules. For example, association rules are commonly used in e-commerce Web sites for product recommendations, which work as follows: When a customer purchases some products, the system recommends him/her some other related products based on what he/she has already purchased.

Recommendation is essentially a prediction problem. It predicts what a customer is likely to buy. Association rules are naturally applicable to such applications. The classification process is as follows:

1. The system first uses previous purchase transactions (the same as market basket transactions) to mine association rules. In this case, there are no fixed classes. Any item can appear on the left-hand side or the right-hand side of a rule. For recommendation purposes, usually only one item appears on the right-hand side of a rule.
2. At the prediction (e.g., recommendation) time, given a transaction (e.g., a set of items already

```
Algorithm CBA(S, D)
S = sort(S);                       // sorting is done according to the precedence>
RuleList = ∅;                      // the rule list classifier
for each rule r ∈ S in sequence do
    if D ≠ ∅ AND r classifies at least one example in D correctly then
        delete from D all training examples covered by r;
        add r at the end of RuleList
    endif
endfor
add the majority class as the default class at the end of RuleList
```

Classification by Association Rule Analysis. Figure 1. A simple classifier building algorithm.

purchased by a customer), all the rules that cover the transaction are selected. The strongest rule is chosen and the item on the right-hand side of the rule (i.e., the consequent) is the predicted item and is recommended to the user. If multiple rules are very strong, multiple items can be recommended.

This method is basically the same as the "use the strongest rule" method described earlier. Again, the rule strength can be measured in various ways, e.g., confidence, χ^2 test, or a combination of both support and confidence. Clearly, the other two classification methods discussed earlier can be applied here as well.

The key advantage of using association rules for recommendation is that they can predict any item since any item can be the class item on the right-hand side. Traditional classification algorithms only work with a single fixed class attribute, and are not easily applicable to recommendations.

Finally, it should be noted that multiple minimum supports in rule mining [11] can be of significant help. Otherwise, *rare items* will never be recommended, which is called the *coverage* problem [14]. It is shown in [14] that using multiple minimum supports can dramatically increase the coverage.

Key Applications

The applications of associative classifiers are very wide. Three main scenarios are briefly described below.

1. Since classification using class association rules is a supervised learning technique, it can be (and has been) used as a classification algorithm just like any other classification algorithm from machine learning, e.g., decision trees, naïve Bayesian classifiers, SVM, and rule induction. In many cases, an associative classifier performs better than these classic machine learning techniques.
2. Apart from classification, individual class association rules themselves are very useful in practice due to the completeness property. In many practical applications (especially diagnostic data mining applications), the user wants to find interesting rules that are actionable. As discussed earlier, traditional classification algorithms (e.g., rule induction or any other technique) are not suitable for such applications because they only find a small subset of rules that exist in data. Many interesting or actionable rules are not discovered. A deployed data mining system, called Opportunity Map, for Motorola Corporation was based on class association rules [12]. When this entry was written, the system had been in use in Motorola for more than 2 years and further improvements were still being made. Although the system was originally designed for finding rules that indicate causes of phone call failures, it had been used in a variety of other applications in Motorola.
3. Using normal association rules for classification or prediction is also very common, especially for the transaction type of data. For such kind of data, as described above, traditional classification techniques are not easily applicable because they can only predict some fixed class items (or labels).

Cross-references

▶ Association Rule Mining on Streams
▶ Decision Trees
▶ Rule Induction

Recommended Reading

1. Agrawal R. and Srikant R. Fast algorithms for mining association rules. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
2. Antonie M.L. and Zaiane O. Text document categorization by term association. In Proc. 2002 IEEE Int. Conf. on Data Mining, 2002, pp. 19–26.
3. Baralis E. and Chiusano S. Essential classification rule sets. ACM Trans. Database Syst, 29(4):635–674, 2004.
4. Cheng H., Yan X., Han J., and Hsu C.-W. Discriminative frequent pattern analysis for effective classification. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 706–715.
5. Dougherty J., Kohavi R., and Sahami M. Supervised and unsupervised discretization of continuous features. In Proc. 12th Int. Conf. on Machine Learning, 1995, pp. 194–202.
6. Jindal N. and Liu B. Identifying comparative sentences in text documents. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 244–251.
7. Li J., Dong G., and Ramamohanarao K. Making use of the most expressive jumping emerging patterns for classification. In Advances in Knowledge Discovery and Data Mining, 4th Pacific-Asia Conf., 2000, pp. 220–232.
8. Li W., Han J., and Pei J. CMAR: Accurate and efficient classification based on multiple class-association rules. In Proc. 2001 IEEE Int. Conf. on Data Mining, 2001, pp. 369–376.
9. Liu B. Web data mining: exploring hyperlinks, contents and usage data. Springer, Berlin, 2007.
10. Liu B., Hsu W., and Ma Y. Integrating classification and association rule mining. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998, pp. 80–86.
11. Liu B., Hsu W., and Ma Y. Mining association rules with multiple minimum supports. In Proc. 5th ACM SIGKDD

Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 337–341.

12. Liu B., Zhao K., Benkler J., and Xiao W. Rule interestingness analysis using OLAP operations. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006, pp. 297–306.
13. Meretakis D. and Wüthrich B. Extending naïve bayes classifiers using long itemsets. In Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 165–174.
14. Mobasher B., Dai H., Luo T., and Nakagawa N. Effective personalization based on association rule discovery from web usage data. In Proc. 3rd ACM Workshop on Web Information and Data Management, 2001, pp. 9–15.
15. Wang K., Zhou S., and He Y. Growing decision trees on supportless association rules. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 265–269.
16. Yin X. and Han J. CPAR: classification based on predictive association rules. In Proc. SIAM International Conference on Data Mining, 2003.

Classification Learning

► Classification

Classification in Streams

Charu C. Aggarwal
IBM T. J. Watson Research Center, Yorktown Heights, NY, USA

Synonyms

Learning in streams; Knowledge discovery in streams

Definition

The classification problem is a well defined problem in the data mining domain, in which a training data set is supplied, which contains several feature attributes, and a special attribute known as the class attribute. The class attribute is specified in the training data, which is used to model the relationship between the feature attributes and the class attribute. This model is used in order to predict the unknown class label value for the test instance.

A data stream is defined as a large volume of continuously incoming data. The classification problem has traditionally been defined on a static training or test data set, but in the stream scenario, either the training or test data may be in the form of a stream.

Historical Background

The problem of classification has been studied so widely in the classification literature, that a single source for the problem cannot be identified. Most likely, the problem was frequently encountered in practical commercial scenarios as a statistical problem, long before the field of machine learning was defined. With advances in hardware technology, data streams became more common, and most data mining problems such as clustering and association rule mining were applied to the data stream domain. Domingos and Hulten [2] were the first to model the problem in the context of data streams.

Foundations

There are numerous techniques available for classification in the classical literature [3]. However, most of these techniques cannot be used directly for the stream scenario. This is because the stream scenario creates a number of special constraints which are as follows:

- The data stream typically contains a large volume of continuously incoming data. Therefore the techniques for training or testing need to be very efficient. Furthermore, a data point may be examined only once over the course of the entire computation. This imposes hard constraints on the nature of the algorithms which may be used for stream classification. This constraint is generally true of almost all data mining algorithms.
- Often the patterns in the underlying data may evolve continuously over time. As a result, the model may soon become stale for data mining purposes. It is therefore important to keep the models current even when the patterns in the underlying data may change. This issue is known as concept drift.
- Many stream classification methods have considerable memory requirements in order to improve computational efficiency. The stream case is particularly resource constrained, since the memory may sometimes be limited, while the computational efficiency requirements continue to be very high.
- In many cases, the rate of incoming data cannot be controlled easily. Therefore, the classification process needs to be nimble enough in order to provide effective tradeoffs between accuracy and efficiency.

Most of the known classification methods can be made to work in the data stream scenario with a few modifications. These modifications are generally designed to deal with either the one-pass constraint, or the stream evolution scenario. The different types of classifiers which can be modified for the data stream scenario are as follows:

- Nearest Neighbor Classifiers: In these techniques, the class label of the nearest neighbor to the target record is used in order to perform the classification. Since the nearest neighbor cannot be defined easily over the entire stream, a stream sample is used in order to perform the classification. This stream sample can be dynamically maintained with the one-pass constraint with the use of a technique called *reservoir sampling*. In order to deal with issues of stream evolution, one can used *biased* reservoir sampling. In biased sampling, a time decay function is used in order to maintain a sample which is biased towards more recent data points.
- Decision Tree Classifiers: In this techniques, decision trees need to be built in one pass of the stream. A method known as Very Fast Decision Trees (VFDT) was proposed in [2] which uses probabilistic split methods in order to create decision trees with predictable accuracy. In particular, the Hoeffding inequality is used in order to ensure that the generated tree produces the same tree as a conventional learner. Several other techniques were proposed by the same authors subsequently, which deal with the time-changing aspect of the data streams.
- Cluster-based Classifiers: An on-demand stream classification model was proposed which uses clustering techniques in order to build the optimal model for a classifier on demand. In this technique, a micro-clustering technique is used in order to compress the underlying data into clusters. The data belonging to different classes are compressed into different clusters. For a given test example, the class of the closest cluster is used in order to predict the class label. One key aspect of this classifier is that it assumes that both the training and the test data are in the form of a stream. The technique calculates the optimal horizon for using the cluster statistics.
- Ensemble Classifiers: In this case, a combination of different models is used in order to deal with the issue of concept drift. This is because different kinds of models work better with different kinds of data patterns. Therefore, an optimal model is picked depending upon the current data pattern. The idea is that different classifiers are more effective for different kinds of data patterns. Therefore, by making an optimal choice of the classifier from the ensemble, it is possible to improve the classification accuracy significantly.
- Bayes Classifiers: The naive Bayes classifier computes the Bayes a-posteriori probabilities of a test instance belonging to a particular class using the inter-attribute independence assumption. The key in adapting such classifiers is to be able to effectively maintain the statistics used to compute conditional probabilities in one pass. In the case of an evolving data stream, the statistics need to be maintained over particular user-specific horizons.

A number of other methods for stream classification also exist which cannot be discussed within the scope of this entry. A detailed survey on classification methods may be found in [3].

Key Applications

Stream classification finds application to numerous data domains such as network intrusion detection, target marketing and credit card fraud detection. In many of these cases, the incoming data clearly has very large volume. For example, typical intrusion scenarios have a large volume of incoming data. Similarly, in the case of target-marketing, super-store transactions may have very large volumes of incoming data.

Many of the traditional classification applications are still used in the batch mode, since the stream technology is still in its infancy, and it is sometimes simpler to collect a sample of the data set and run a batch process on it. Most of the traditional problems for the classification domain will eventually be transformed to the data stream scenario. This is because more and more data domains are being converted to the stream scenario with advances in hardware technology.

Cross-references

▶ Association Rule Mining on Streams
▶ Clustering on Streams
▶ Data Stream

Recommended Reading

1. Aggarwal C.C. (ed.). Data Streams: Models and Algorithms. Springer, Berlin Heidelberg, New York, 2007.
2. Domingos P. and Hulten G. Mining high speed data streams. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 71–80.
3. James M. Classification Algorithms. Wiley, New York, 1985.

Classification Tree

▶ Decision Tree Classification

Classification Trees

▶ Decision Trees

Classifier Combination

▶ Ensemble

Client-Server DBMS

M. Tamer Özsu
University of Waterloo, Waterloo, ON, Canada

Definition

Client-server DBMS (database management system) refers to an architectural paradigm that separates database functionality between client machines and servers.

Historical Background

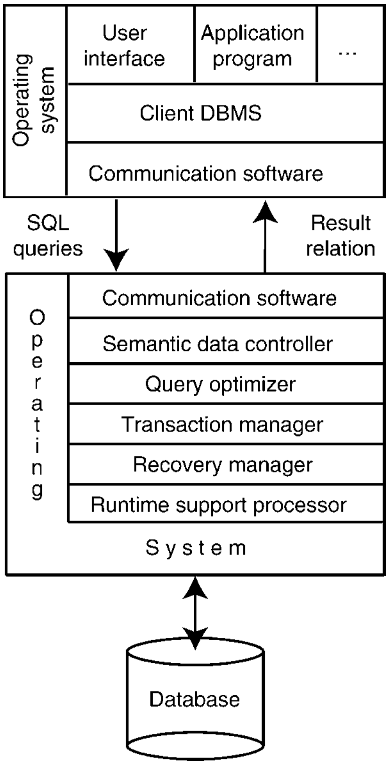
The original idea, which is to offload the database management functions to a special server, dates back to the early 1970s [1]. At the time, the computer on which the database system was run was called the *database machine*, *database computer*, or *backend computer*, while the computer that ran the applications was called the *host computer*. More recent terms for these are the *database server* and *application server*, respectively.

The client-server architecture, as it appears today, has become a popular architecture around the beginning of 1990s [2]. Prior to that, the distribution of database functionality assumed that there was no functional difference between the client machines and servers (i.e., an earlier form of today's peer-to-peer architecture). Client-server architectures are believed to be easier to manage than peer-to-peer systems, which has increased their popularity.

Foundations

Client-server DBMS architecture involves a number of database client machines accessing one or more database server machines. The general idea is very simple and elegant: distinguish the functionality that needs to be provided and divide these functions into two classes: server functions and client functions. This provides a *two-level architecture* that makes it easier to manage the complexity of modern DBMSs and the complexity of distribution.

In client-server DBMSs, the database management functionality is shared between the clients and the server(s) (Fig. 1). The server is responsible for the bulk of the data management tasks as it handles the storage, query optimization, and transaction management (locking and recovery). The client, in addition

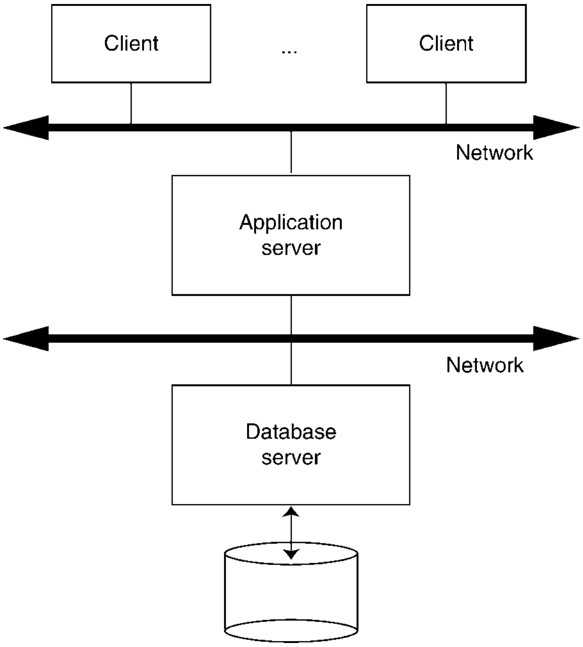


Client-Server DBMS. Figure 1. Client-server reference architecture.

to the application and the user interface, has a *DBMS client* module that is responsible for managing the data that are cached to the client, and (sometimes) managing the transaction locks that may have been cached as well. It is also possible to place consistency checking of user queries at the client side, but this is not common since it requires the replication of the system catalog at the client machines. The communication between the clients and the server(s) is at the level of SQL statements: the clients pass SQL queries to the server without trying to understand or optimize them; the server executes these queries and returns the result relation to the client. The communication between clients and servers are typically over a computer network.

In the model discussed above, there is only one server which is accessed by multiple clients. This is referred to as *multiple client-single server* architecture [3]. There are a number of advantages of this model. As indicated above, they are simple; the simplicity is primarily due to the fact that data management responsibility is delegated to one server. Therefore, from a data management perspective, this architecture is similar to centralized databases although there are some (important) differences from centralized systems in the way transactions are executed and caches are managed. A second advantage is that they provide predictable performance. This is due to the movement of non-database functions to the clients, allowing the server to focus entirely on data management. This, however, is also the cause of the major disadvantage of client-server systems. Since the data management functionality is centralized at one server, the server becomes a bottleneck and these systems cannot scale very well.

The disadvantage of the simple client-server systems are partially alleviated by a more sophisticated architecture where there are multiple servers in the system (the so-called *multiple client-multiple server* approach). In this case, two alternative management strategies are possible: either each client manages its own connection to the appropriate server or each client knows of only its "home server", which then communicates with other servers as required. The former approach simplifies server code, but loads the client machines with additional responsibilities, leading to what has been called "heavy client" systems. The latter approach, on the other hand, concentrates the data management functionality at the servers. Thus, the transparency of data access is provided at the server interface, leading to "light clients."



Client-Server DBMS. Figure 2. Database server approach.

The integration of workstations in a distributed environment enables an extension of the client-server architecture and provides for a more efficient function distribution. Application programs run on workstations, called *application servers*, while database functions are handled by dedicated computers, called *database servers*. The *clients* run the user interface. This leads to the present trend in three-tier distributed system architecture, where sites are organized as specialized servers rather than as general-purpose computers (Fig. 2).

The application server approach (indeed, a n-tier distributed approach) can be extended by the introduction of multiple database servers and multiple application servers, as can be done in client-server architectures. In this case, it is common for each application server to be dedicated to one or a few applications, while database servers operate in the multiple server fashion discussed above.

Key Applications

Many of the current database applications employ either a two-layer client-server architecture of the three-layer application-server approach.

Cross-references

► DBMS Architecture

Recommended Reading

1. Canaday R.H., Harrisson R.D., Ivie E.L., Rydery J.L., and Wehr L.A. A back-end computer for data base management. Commun. ACM, 17(10):575–582, 1974.
2. Orfali R., Harkey D., and Edwards J. Essential Client/Server Survival Guide, Wiley, New York, 1994.
3. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 2nd edn., Prentice-Hall, Englewood Cliffs, NJ, 1999.

Clinical Classifications

► Clinical Ontologies

Clinical Content Database

► Clinical Knowledge Repository

Clinical Content Registry

► Clinical Knowledge Repository

Clinical Content Repository

► Clinical Knowledge Repository

Clinical Data Acquisition, Storage and Management

Chimezie Ogbuji
Cleveland Clinic Foundation, Cleveland, OH, USA

Synonyms

Electronic data capture; Case report forms; Clinical data management systems

Definition

The management of clinical data for supporting patient care and for supporting retrospective clinical research requires a means to acquire the clinical data and a repository that stores the data and provides the functions necessary for managing them over their lifetime. Typically, patient data are collected "at the point of care" (i.e., onsite where health care is being provided) and entered into a patient record system [2]. Data entry is typically the first line of precaution for maintaining a certain amount of quality on the data collected. Subsequently, a representative from an externally sponsoring organization or authorized personnel from within the health care institution then extracts a select set of medical record data into a Clinical Data Management System (*CDMS*). The entries in such systems are often referred to as secondary patient records since they are derived from a primary patient record and are used by personnel who are not involved in direct patient care [2]. These systems are also often referred to as *patient registries.*

The study committee of the Institute of Medicine (IOM) defined [2] a computer-based patient record (*CPR*) as

► an electronic patient record that resides in a system specifically designed to support users by providing accessibility to complete and accurate data, alerts, reminders, clinical decision support systems, links to medical knowledge, and other aids.

Such systems are also often referred to as Electronic Health Records (*EHR*s). The committee also defined a *primary* patient record as one used by health care professionals while providing patient care services to review patient data or document their own observations, actions, or instructions [2]. Finally, the committee emphasized the distinction between clinical data and the systems that capture and process this data by defining a patient record *system* as

► the set of components that form the mechanism by which patient records are created, used, stored, and retrieved.

A CDMS is the repository for the management of the data used for clinical studies or trials. One of the core services provided by a CDMS is to facilitate the identification (and correction) of errors due to human entry as well as errors that existed in the source from which the data was gathered. Data completeness implies that CDMS will accommodate an expected range and complexity for the data in the system [2]. In addition, the CDMS can employ the use and enforcement of one or more vocabulary standards.

Finally, a CDMS will also provide services for querying the data as well as generate reports from the data. The generated reports and results of such queries are typically transmitted to a centralized authority or normalized for use by statistical analysis tools.

Historical Background

Virtually every person in the United States who has received health care in the United States since 1918 has a patient record [2]. Most of these records consist of structured paper forms with sections that consist solely of narrative text. However, conventional patient records can also appear in other forms such as scanned media, microfilm, optical disk, etc.

They are created and used most frequently in health care provider settings. However, their use also extends to other facilities such as correctional institutions, the armed forces, occupational health programs, and universities [2].

The process of recording patient care information has primarily consisted of entry into a paper patient record. For the purpose of a clinical trial or study, data are manually transcribed into a paper Case Report Form (*CRF*). CRFs are typically in the form of a questionnaire formulated to collect information specific to a particular clinical trial. The ICH Guidelines for Good Clinical Practice define [4] the CRF as:

- ▶ A printed, optical, or electronic document designed to record all of the protocol required information to be reported to the sponsor on each trial subject.

CRFs are then collected by a representative from the sponsoring organization and manually entered into a CDMS. This secondary transcription is often called *double data entry*. In some cases, Optical Character Recognition (*OCR*) is used to semi-automate the transcription from a CRF into a CDMS [1].

Traditionally, clinical data management systems consist of infrastructure built on top of relational database management systems. Depending on the nature of the requirements for the creation of analysis data sets for biostatisticians, accuracy of the data, and speed of data entry, a wide spectrum of database management or spreadsheet systems are used as the underlying medium of storage for the CDMS.

Good database design and proper application of relational model theory for normalizing the data is typically used to ensure data accuracy. Traditional relational query languages such as SQL are used to identify and extract relevant variables for subsequent analysis or reporting purposes.

Foundations

Electronic Data Capture

There is a slow, but steady move by pharmaceutical companies towards the adoption of an electronic means of capturing patient record information directly from the source and at the point of care into an electronic system that submits the data relevant to the trial to the sponsor or to other consumers of electronic patient record data. This new shift of emphasis from paper to a direct electronic system is often referred to (in the health care industry) as Electronic Data Capture (*EDC*) [3].

Infrastructure and Standards for Data Exchange

Once patient record data are collected and stored in an electronic information system, the increasing need to transfer the machine-readable data to external systems emphasizes the importance of standardized formats for communication between these disparate systems [2]. Efforts to standardize a common format for communication between health care systems and other external consumers of health care data have settled on the adoption of Extensible Markup Language (*XML*) as the primary data format for the transmission of Health Level 7 (*HL7*) messages.

HL7 is an organization with a mission to develop standards that improve the delivery of care, optimize the management of workflow, reduce ambiguity in clinical terminology and facilitate efficient transfer of knowledge between the major stakeholders.

Document Models and Management Systems

As its name implies, XML is a markup language for describing structured data (or *documents*) in a manner that is both human- and machine-readable. It can be extended to support users who wish to define their own vocabularies. It is meant to be highly reusable across different information systems for a variety of purposes. It is recommended by the World Wide Web Consortium (*W3C*) and is a free and open standard.

XML is at the core of an entire suite of technologies produced by the W3C that includes languages for querying XML documents as well as describing their structure for the purpose of validating their content. This suite of technologies is meant to serve as infrastructure for a contemporary set of information systems each known more broadly as a Document Management System (*DMS*).

Common Components of Information Systems

Like most information systems, document management systems are comprised of a particular data model (XML in this case), one or more query languages, and a formal processing model for systems that wish to support queries written in language against the underlying data. Document management systems (and information systems in general) typically also offer security services that ensure limited access to the data. This is particularly important for clinical data management systems.

With respect to the kind of services they provide for the systems that are built on top of them (such as clinical data management systems), document management systems are very much like relational database management systems. However, whereas relational database management systems have an underlying relational model that is tabular and rigid, document management systems have a data model that is hierarchical with data elements that can be extended to support new terminology over the life of the data.

Text-Oriented Information Systems

Most modern computer-based patient record systems mainly adopt information systems with hierarchical, relational, or text-oriented data models. Text-based information systems typically store their content primarily as narrative text and often employ natural language processing for extracting structured data for transcription into a clinical data management system. Querying such systems usually involves keyword-based searches that use text indexes that are used to associate words with the sections of narrative in which they can be found.

Key Applications

Electronic data capture and clinical data management systems constitute the majority of the infrastructure necessary in the overall process of clinical research from the point of interaction with primary patient records all the way to the analysis of the curated clinical research data. The sections below describe the major areas where their application makes a significant difference.

Electronic Data Collection Options

There are a variety of ways in which data can be acquired electronically for transcription into a clinical data management system. The most desired means is one where the data are directly retrieved electronically from an existing source such as the primary patient record. This method is often referred to as *single entry* [1]. It requires that the primary patient record adopt or align with a set of consistent format standards such that they can facilitate the support of primary care as well as reuse for the purpose of (unanticipated) clinical research. Unfortunately the lack of adoption of computer-based patient records remains a primary impediment to this more direct means of acquiring clinical data [2,3].

Alternatively, clinical data can be transcribed from a primary patient record into a secondary patient record using some form of an electronic user interface on a particular device. Typically, such user interfaces are either web browser-based (i.e., they are built on top of an existing web browser such as Internet Explorer or Firefox) or they are written as independent applications that are launched separately. The latter approach is often referred to as a *thick-client system* [6].

Patient Registries

The set of functions associated with a secondary computer-based patient record system is often adopted from the underlying information system. Modern document and relational database management systems are equipped with a wide spectrum of capabilities each of which is directly relevant to the needs of users of these systems. This includes: content organization, archival, creation of documents, security, query services, disaster recovery, and support for web-based user interfaces.

Clinical Workflow Management

Equally important to the clinical data is the management of the pattern of activity, responsibilities, and resources associated with a particular clinical study or trial. These patterns are often referred to as *workflow*. Orchestrating the overall process can also have a significant impact on the success of a clinical study. Clinical data management systems sometimes have off-the-shelf capabilities for managing workflow. These usually support some amount of automation of the workflow process. Document management systems that include capabilities for building customized application are well-suited for supporting workflows that are either specific to a particular study protocol or capable of supporting multiple (or arbitrary) protocols.

Quality Management, Report Generation, and Analysis

Finally, document management and relational database management systems include capabilities for monitoring error in the data collected. This is often supported through the application of a set of common constraints that are relevant to the research protocol. Typically, these systems have an automated mechanism for indicating when the underlying data does not adhere to the constraints specified.

In addition, document management and relational database management systems include services for generating reports and extracting variables for statistical analysis.

Future Directions

Modern information management systems are adopting standards for representation formats that push the envelope of machine-readability. In particular, the W3C has recently been developing a suite of technologies that build on the standards associated with the World Wide Web and introduce a formal model for capturing knowledge in a manner that emphasizes the meaning of terms rather than their structure. Such approaches to modeling information are often referred to as knowledge representation or conceptual models. This particular collection of standards is commonly referred to as *semantic web technologies* [5].

Semantic web technologies are built on a graph-based data model known as the Resource Description Framework (*RDF*) as well as a language for describing conceptual models for RDF data known as Ontology Web Language (*OWL*). RDF leverages a highly-distributable addressing and naming mechanism known as Uniform Resource Identifiers (*URIs*) that is the foundation of the current web infrastructure.

Semantic web technologies also include a formal mechanism for rendering or transforming XML document dialects into RDF known as Gleaning Resources Descriptions from Dialects of Languages (*GRDDL*). Finally a common query language has been defined for accessing data expressed in RDF known as *SPARQL*.

The Institute of Medicine has indicated [2] that the flexibility of computer-based patient records is primarily due to their adoption of a data dictionary that can be expanded to accommodate new elements. In addition, the IOM has identified [2] the following as crucial to the evolution of content and standard formats in computer-based patient record systems:

- The content of CPRs must be defined and contain a uniform core set of data elements.
- Data elements must be named consistently via the enforcement of some form of vocabulary control.
- Format standards for data exchange must be developed and used.

In addition, the IOM's study committee identified the ability for CPRs to be linked with other clinical records as a critical attribute of a comprehensive computer-based patient record. The combination of these observations is a strong indication that in the future, clinical data management systems will be built on information management systems that adopt semantic web technologies in order to better meet the growing needs of the management of clinical research data.

Finally, a new generation of technologies for building declarative web applications will lower the technological barrier associated with the kind of user interfaces necessary for the adoption of electronic data capture methods at health care institutions. In particular, an XML-based technology known as *XForms* is well positioned to have a significant impact on the front end of the clinical data pipeline (data collection).

XForm applications are web form-based, independent of the device on which they are deployed and use XML as the data model of the underlying content. This approach has strong correspondence with the current direction of clinical data exchange standards with the adoption of XML as the format for communication between health care systems.

In the near future, lightweight devices (such as Tablet PCs) will connect to remote, distributed computer-based patient record systems over a secure web-based network protocol. Electronic data capture will be implemented by XForm applications that run in a browser and compose XML documents that represent sections of a computer-based patient record. These documents will adhere to a standard document format for the exchange of medical records such as the HL7 Clinical Document Architecture (*CDA*). The HL7 CDA is an XML-based document markup standard that specifies the structure and semantics of clinical documents for the purpose of exchange.

These documents will be securely transmitted directly into a primary computer-based patient record which employs XML as its core data model and uses GRDDL to also store an RDF representation of the document that conforms to a formal, standard ontology (expressed in OWL) that describes the meaning of the terms. This ontology provides a certain degree of logical consistency that facilitates ad hoc analysis through the use of logical inference.

Patients that meet the criteria for a particular research protocol will be identified by a SPARQL query that is dispatched against the patient record system, which uses terminology easily understood by the investigators themselves (rather than an intermediary database administrator). These patient records will then be transmitted directly into a clinical data management system (or patient registry) that will include the facilities for managing the workflow associated with the relevant research protocol. These facilities will be implemented as web applications built on the same underlying information management systems as those used by the primary computer-based patient records.

Cross-references

▶ Clinical Content Management
▶ Clinical Data Quality and Validation
▶ Data warehousing and Quality Management for Clinical Practice
▶ Electronic Health Record
▶ Life Cycles and Provenance
▶ Versioning

Recommended Reading

1. Anisfeld M.H. and Prokscha S. Practical Guide to Clinical Data Management. CRC Press, Boca Raton, FL, 1999.
2. Committee on Improving the Patient Record, Institute of Medicine. The computer-based patient record: an essential technology for health care (revised edition). National Academies Press, 1997.
3. Lori A. and Nesbitt. Clinical Research: What It Is and How It Works. Jones and Bartlett Publishers, Sudbury, MA, 2003.
4. Rondel R.K., Varley S.A., and Webb C.F. Clinical Data Management. Wiley, Chichester, 2000.
5. Ruttenberg A., Clark T., Bug W., Samwald M., Bodenreider O., Chen H., Doherty D., Forsberg K., Gao Y., Kashyap V., Kinoshita J., Luciano J., Marshall M.S., Ogbuji C., Rees J., Stephens S., Wong G.T., Elizabeth Wu, Davide Zaccagnini, Tonya Hongsermeier, Neumann E., Herman I., and Cheung K.-H. Advancing translational research with the Semantic Web. BMC Bioinformatics, 8(Suppl. 3), 2007.
6. Wilson D., Pace M.D., Elizabeth W., and Staton, M. S. T. C. Electronic Data Collection Options for Practice-Based Research Networks. Ann. Fam. Med., 3:S2–S4, 2005.

Clinical Data and Information Models

Chintan Patel, Chunhua Weng
Columbia University, New York, NY, USA

Definition

A formal representation of the clinical data using entities, types, relationships and attributes. The abstraction of clinical data into an information model enables reusability and extensibility of the database to satisfy different application needs and accommodate changes in the underlying data.

Key Points

The clinical domain is a data rich environment with multitude of different data entities ranging from several thousands of laboratory tests, procedures or medications that change often with new ones getting added almost every day. Furthermore these data are generated from different information systems or devices (often from different vendors) in the hospital. Integrating such wide variety of data streams into a common information model is a challenging task.

Most healthcare databases use generic information models [3,4] such as event-component models with an Entity-Attribute-Value [5] (EAV) schema to represent the data (see Fig. 1). The advantage of using a generic information model is to accommodate the data heterogeneity and extensibility. Generally, an external terminology or vocabulary is used in conjunction with a generic information model to represent the clinical domain (laboratory tests, medications and so on) and the healthcare activities, for example, LOINC is a

```
Event (Chest X-Ray*)  -> Attribute1 = Value1 (on-date* = mm/dd/yyyy)
                      -> Attribute2 = Value2 (finding-site* = breast*)
                      -> Attribute3 = Value3 (has-morphology* = neoplasm*)
                                 ...
*coded using an external terminology
```

Clinical Data and Information Models. Figure 1. The event component information model.

standard vocabulary for representing laboratory data or SNOMED CT for healthcare activities.

Various information models have been proposed towards standardizing the representation of clinical data. The goal of standardizing the information model is to facilitate exchange, sharing and reuse of clinical data by different systems locally as well as nationally. Following are some current standardized models:

HL7 Reference Information Model: The HL7 standards organization [2] has developed a Reference Information Model (RIM) to share consistent meaning of healthcare data beyond local context. The RIM specifies a set of abstract bases classes Entity, Role, Participation and Act, which contain specific classes/attributes such as Person, Organization, Patient, Provider, Intent, Observation and so on. This model is used to create concrete concepts by combining the RIM types, for example, *elevated blood pressure* would be represented in RIM as class = Observation with code = Finding of increased blood pressure (SNOMED#241842005), mood = Event, interpretation code = abnormal (HL7#A), target site = heart (LOINC#LP7289). Note that standardized terminology codes (SNOMED CT and LOINC) are used to represent specific findings and modifiers. An implementation of HL7 RIM based model over a relational database schema is described here [1].

openEHR Reference Model: The openEHR specification [6] (developed largely by the institutions in EU and Australia) provides information models for the electronic health record (EHR), demographics, data structures, integration and so on. The openEHR EHR model represents various facets of EHR such as clinician/patient interaction, audit-trailing, technology/data format independence and supporting secondary uses. The openEHR project uses the notion of archetypes that enable domain experts to formally model a domain concept (or an aggregation of concepts), corresponding constraints and other compositions, for example, an archetype on *blood pressure measurement* consists of systolic, diastolic measurements and units with other clinically relevant information such as history.

Recommended Reading

1. Eggebraaten T.J., Tenner J.W., and Dubbels J.C. A health-care data model based on the HL7 reference information model. IBM Syst. J., 46(1):5–18, 2007.
2. HL7 Reference Information Model. Available at: http://www.hl7.org/ (Accessed April 18, 2008).
3. Huff S., Rocha R., Bray B., Warner H., and Haug P. An event model of medical information representation. J. Am. Med. Inform. Assoc., 2(2):116–134, 1995.
4. Johnson S. Generic data modeling for clinical repositories. J. Am. Med. Inform Assoc., 3(5):328–367, 1996.
5. Nadkarni P., Marenco L., Chen R., Skoufos E., Shepherd G., and Miller P. Organization of heterogeneous scientific data using the EAV/CR representation. J. Am. Med. Inform. Assoc., 6(6):478–571, 1999.
6. openEHR Reference Information Model. Available at: http://www.openehr.org/ (Accessed April 18, 2008).

Clinical Data Management Systems

► Clinical Data Acquisition, Storage and Management

Clinical Data Quality and Validation

Chintan Patel, Chunhua Weng
Columbia University, New York, NY, USA

Definition

Clinical data quality is defined as the accuracy and completeness of the clinical data for the purposes of clinical care, health services and other secondary uses such as decision support and clinical research. The quality of clinical data can be achieved by the standardization, inspection and evaluation of the data generating processes and tools [2].

Key Points

The term data quality can potentially have different meanings or interpretation based on the domain or the application using the data [1]. Even within the context of clinical databases, there exists a multitude of different data types (administrative data, procedure data, laboratory data and so on) that may be used for several different applications such as clinical report generation, billing or research. The major components of clinical data quality can be broadly characterized as follows:

Accuracy

Clinical data are often generated by automated systems (such as lab equipment) or manually entered by clinicians (notes). These data generating processes are prone to errors that result in incorrect data being stored in the database. The severity of errors can vary significantly, for example, a minor misspelling in patient history note versus a prescription error in drug dosage order can lead to drastically different outcomes in terms of patient care. The accuracy of clinical data is defined as the proportion of correct data (truly representing the actual patient condition or measurement) in the clinical database. The accuracy of clinical data depends on the enforcement of well-defined data entry standards and protocols.

Completeness

It is defined as the availability of data elements in a clinical database that are necessary to accomplish a given task, for example, a clinical trial recruitment application with detailed eligibility criteria would require information from the clinical notes in addition to coded problem list data. The completeness of a patient record is critical for a clinician to choose a most appropriate treatment plan for the patient. The availability of complete patient information is critical during an emergency condition. In the case of unavailability of data elements, some applications tend to substitute data sources, which can lead to sub-optimal results. Consider for example, a clinical decision support application reusing coarse ICD (International Classification of Disease) classification to generate decisions.

Reliability

The notion of "repeatability" – to determine whether the clinical data generation processes produce consistent data at different times or settings. Hospitals are a chaotic environment with multiple care providers taking care of a single patient. It becomes critical to develop data entry protocols to ensure consistent representation of patient information in the clinical database. Often to eliminate the variations across different users the data entry software systems such as the EMR (electronic medical record) contain various checks to ensure the correctness and completeness of the data elements [3]. The coding of clinical data using terminologies such as ICD has to be done in a consistent fashion to facilitate applications that require data integration or comparative analysis.

Maintaining quality in clinical databases is a continuous process requiring strong commitment from different stakeholders involved. The amount of electronic biomedical data generated is growing at an exponential rate. Developing high quality clinical databases can have significant implications for the applications reusing the data.

Cross-references

▶ Quality and Trust of Information Content and Credentialing

Recommended Reading

1. Arts D., De Keizer N., and Scheffer G. Defining and improving data quality in medical registries: a literature review, case study, and generic framework. J. Am. Med. Inform. Assoc., 9(6): 600–611, 2002.
2. Black N. High-quality clinical databases: breaking down barriers. Lancet, 353(9160):1205–1211, 2006.
3. Hogan W. and Wagner M. Accuracy of data in computer-based patient records. J. Am. Med. Inform. Assoc., 4(5): 342–397, 1997.

Clinical Decision Support

Adam Wright
Partners HealthCare, Boston, MA, USA

Synonyms

CDS; Decision support

Definition

Clinical Decision Support systems are computer systems which assist humans in making optimal clinical decisions. While clinical decision support systems are most often designed for clinicians, they can also be developed to assist patients or caregivers. Common examples of clinical decision support systems include drug-drug interaction checks, dose range checking for medication and preventive care reminders.

Historical Background

The first clinical decision support system was described in 1959 by Robert Ledley and Lee Lusted [6] in their paper "Reasoning foundations of medical diagnosis; symbolic logic, probability, and value theory aid our understanding of how physicians reason." Ledley and Lusted described an analog computer used to sort cards containing a diagnosis and a series of punches which represented symptoms. By selecting the cards which matched the symptoms present in a given case a clinician could develop a possible set of diagnosis.

In 1961, Homer Warner [15] described a clinical decision support system for diagnosing congenital heart defects. The system was developed around a contingency table that mapped clinical symptoms to forms of congenital heart disease. A physician would input the patient's symptoms and findings from the clinical exam and other studies into the system, which would then proceed to suggest the most probable diagnoses based on the contingency table.

In the 1970s, Edward Shortliffe developed the well-known MYCIN system for antibiotic therapy. MYCIN was an expert system with a large knowledge base of clinical rules [12]. Users of MYCIN would input known facts about their patient, and MYCIN would apply them to the rule base using backward chaining to yield a probable causative agent for infections as well as suggestions for antibiotic therapy.

While the systems described so far all focused on a specific area of medicine, the INTERNIST-I system, developed by Randy Miller, Harry Pople and Jack Myers [8] targeted the broad domain of diagnosis in internal medicine. The INTERNIST-I knowledge base consisted of a large set of mappings between symptoms and diagnoses. These links were scored along three axes: evoking strength (the likelihood that a patient has a diagnosis given a particular symptom), frequency (how often a symptom is present given a particular diagnosis) and import (how critical it is that a particular diagnosis be considered given that it is possible or probable based on a set of symptoms). Octo Barnett's DXplain system for diagnostic decision support was developed around the same time as INTERNIST-I.

The earliest decision support systems were standalone, but the second wave in clinical decision support, beginning in the 1970s, was the integration of decision support systems into broader clinical information systems. The first two examples of this integration were the Health Evaluation through Logical Processing (HELP) system at the University of Utah and LDS Hospital, and the Regenstrief Medical Records System (RMRS) developed at the Regenstrief Institute in Indianapolis. The HELP system, which was used for many facets of patient care, had support for the development of a variety of kinds of decision support, and was especially well known for its Bayesian reasoning modules. The RMRS was developed, from the ground up, with a large knowledge base of clinical care rules. Both HELP and RMRS are in active use today.

Most current commercially available clinical information systems have some support for clinical decision support, and efforts to standardize representation and enable the sharing of decision support content are ongoing.

Foundations

Development of clinical decision support systems entails a variety of issues. The first step in developing any clinical information system is to identify an important clinical target, and then consider interventions. The most critical database systems related issues are knowledge representation, storage and standards.

Issues of Knowledge Representation

Once a desired clinical decision support target has been identified and relevant medical knowledge has been collected, the knowledge must somehow be represented. Knowledge in clinical decision support systems has been represented in a variety of ways, the most common being if-then rules, expert systems, probabilistic and Bayesian systems and reference content.

Perhaps the simplest form of knowledge is if-then rules. Much of clinical decision support content can be represented this way (for example "if the acetaminophen dose is 10 g per day, alert the user that this is too high" or "if the patient is over 50 years of age and has not had a sigmoidoscopy, recommend one"). These rules are frequently designed to be chained together, although generally in a fixed and predetermined pattern.

More complex than simple if-then rules are expert systems. These systems are composed of large knowledge bases which contain many intermediate states and assertions. Like if-then rules, these rules are composed of an antecedent, a consequent and an implication. However, expert systems are generally designed to elicit emergent behavior from extensive chaining including, in many cases, goal-directed backward chaining.

Probabilistic and Bayesian systems share much in common with if-then rules. However, instead of modeling knowledge and clinical states as deterministic values, they use probabilities. By combining these probabilities with knowledge provided by the user, these systems can estimate the likelihood of various diagnostic possibilities, or the relative utility of different therapeutic modalities. It is important to note that many expert systems employ probabilistic or Bayesian reasoning.

A simpler form of knowledge representation is reference knowledge designed to be read by a human. This form of decision support provides information to the user but expects him or her to formulate a plan of action on his or her own. In many cases knowledge, such as clinical guidelines, can be equivalently modeled as rules or as reference content. Reference content is simpler to construct, but it sometimes can not be as proactive as rule-based content.

Storage of Clinical Knowledge in Database Systems

A key challenge for developers of database systems for clinical decision support is selecting the optimal strategy for storing clinical knowledge in a database. This selection has many tradeoffs among performance, space, maintainability and human readability.

Rule based decision support content is often stored as compiled or interpreted code and, when properly integrated into clinical information support systems, can be very efficient. However, many systems instead choose to store rules in some intermediate form, often indexed according to their trigger (a clinical event, such as a new prescription, which causes decision support rules to fire). A chained hash table with these triggers as its keys and decision support rules to invoke as values can be a particularly efficient representation.

In cases of particularly high transaction volume, where performance is important and the number of rules to evaluate is large, more sophisticated storage and processing mechanisms can be used. One of the most effective in terms of performance (although not necessarily in terms of space) is Charles Fogarty's Rete algorithm. The Rete algorithm is an efficient network-based method for pattern matching in rule-based systems.

Because it is not rule based, reference knowledge requires a different set of storage and retrieval strategies, based largely on the principles of information retrieval. In general, these strategies employ one or some combination of full-text search and metadata queries.

Standards for Sharing Clinical Decision Support Content between Database Systems

In addition to the aforementioned issues of internal representation of clinical knowledge, there are also issues relating to the sharing of clinical decision support content between systems. Several standards for sharing such content have been proposed, beginning with Arden Syntax, a standard for event-driven rule-based decision support content. Other standards, such as Guideline Interchange Format (GLIF) and the related expression language GELLO exist to represent more complex forms of clinical knowledge. While construction of standards for representing clinical knowledge may seem straightforward, issues relating to terminology and a reference model for patient information have proven formidable.

An alternate approach to strict structured knowledge representation formalisms for sharing clinical decision support content is the use of services. Several recent efforts, including SEBASTIAN and SANDS have defined a set of interfaces and, in the case of SANDS, patient data models to help overcome prior difficulties in sharing decision support content.

Key Applications

Applications of clinical decision support can be categorized along a variety of axes, including intervention type (alert, reminder, reference information, etc.), clinical purpose (diagnosis, therapy, prevention), disease target (diabetes, hypertension, cancer, etc.) and user (physician, nurse, patient, etc.).

Several clinical decision support systems have been described in the historical background section. Additional significant systems include:

- Morris Collen's system for "Automated Multiphasic Screening And Diagnosis."
- Howard Bleich's system for diagnosis and treatment of acid-base disorders.
- A system for the diagnosis and management of abdominal complaints developed by F.T. de Dombal.
- The ATTENDING system developed by Perry Miller and designed to critique and suggest improvements to anesthesia plans.
- A system for ventilator management by Dean Sittig.

- A blood product ordering critiquing system by Reed Gardner.
- An antibiotic advising system by Scott Evans.

Experimental Results

There is a long experimental tradition in the field of clinical decision support, and many systems have shown strong results, even for the earliest systems. Warner's system for congenital heart defects was compared favorably to experienced cardiologists, MYCIN proposed clinically appropriate antibiotic therapy 75% of the time (and got better as more rules were added) and INTERNIST performed about as well as average doctors at diagnosis.

Just as significant is the effect that such systems have on physician practice. In a landmark paper, Clem McDonald described the results of an experimental trial performed within the RMRS system. In the trial, half of the physician users of RMRS received patient care suggestions based on the knowledge base of rules, while half did not. Physicians who received the suggestions carried them out 51% of the time, while physicians who did not receive suggestions performed the actions that would have been suggested only 22% of the time. When the reminder system was turned off, physician performance returned almost immediately to baseline.

There have been several significant systematic reviews of clinical decision support systems. A 2005 review by Amit Garg [2] found that decision support systems were associated with improved provider performance in 64% of the controlled trials reviewed. Another systematic review by Ken Kawamoto found that decision support systems improved performance in 68% of trials, and that systems designed to the highest criteria improved performance in 94% of trials.

Cross-references

► Clinical Data and Information Models
► Clinical Prediction Rule

Recommended Reading

1. Bates D.W., Kuperman G.J., and Wang S., et al. Ten commandments for effective clinical decision support: making the practice of evidence-based medicine a reality. J. Am. Med. Inform. Assoc., 10(6):523–530, 2003.
2. Garg A.X., Adhikari N.K., and McDonald H., et al. Effects of computerized clinical decision support systems on practitioner performance and patient outcomes: a systematic review. Jama, 293(10):1223–1238, 2005.
3. Kawamoto K., Houlihan C.A., Balas E.A., and Lobach D.F. Improving clinical practice using clinical decision support systems: a systematic review of trials to identify features critical to success. BMJ, 330(7494):765, 2005.
4. Kawamoto K. and Lobach D.F. Design, implementation, use, and preliminary evaluation of SEBASTIAN, a standards-based web service for clinical decision support. In Proc. AMIA Symposium, 2005, pp. 380–384.
5. Kuperman G.J., Gardner R.M., and Pryor T.A. HELP: A Dynamic Hospital Information System. Springer, New York, 1991.
6. Ledley R.S. and Lusted L.B. Reasoning foundations of medical diagnosis; symbolic logic, probability, and value theory aid our understanding of how physicians reason. Science, 130(3366):9–21, 1959.
7. McDonald C.J. Protocol-based computer reminders, the quality of care and the non-perfectability of man. N. Engl. J. Med., 295(24):1351–1355, 1976.
8. Miller R.A., Pople H.E. Myers J.D. Internist-1, an experimental computer-based diagnostic consultant for general internal medicine. N. Engl. J. Med., 307(8):468–476, 1982.
9. Osheroff J.A., Pifer E.A., Sittig D.F., Jenders R.A., and Teich J.M. Improving Outcomes with Clinical Decision Support: an Implementers' Guide. HIMSS, Chicago, 2005.
10. Osheroff J.A., Teich J.M., Middleton B., Steen E.B., Wright A., and Detmer D.E. A roadmap for national action on clinical decision support. J. Am. Med. Inform. Assoc., 14(2):141–145, 2007.
11. Sittig D.F., Wright A., and Osheroff J.A., et al. Grand challenges in clinical decision support. J. Biomed. Inform., 41(2): 387–392, 2007.
12. Shortliffe E.H., Davis R., Axline S.G., Buchanan B.G., Green C.C., and Cohen SN. Computer-based consultations in clinical therapeutics: explanation and rule acquisition capabilities of the MYCIN system. Comput. Biomed. Res., 8(4):303–320, 1975.
13. Wright A., Goldberg H., Hongsermeier T., and Middleton B. A description and functional taxonomy of rule-based decision support content at a large integrated delivery network. J. Am. Med. Inform. Assoc., 14(4):489–496, 2007.
14. Wright A., Sittig D.F., SANDS: A service-oriented architecture for clinical decision support in a National Health Information Network. J. Biomed. Inform. (2008), doi:10.1016/j.jbi.2008.03.001.
15. Warner H.R., Toronto A.F., Veasey L.G., and Stephenson R. A mathematical approach to medical diagnosis. Application to congenital heart disease. Jama, 177:177–183, 1961.

Clinical Document Architecture

Amnon Shabo (Shvo)
IBM Research Lab-Haifa, Haifa, Israel

Synonyms

CDA; CDA R1; CDA R2

Definition

The Clinical Document Architecture (CDA) is a document markup standard that specifies the structure and semantics of clinical documents for the purpose of exchange and share of patient data. The standard is developed by Health Level Seven (HL7) – a Standards Development Organization [2] focused on the area of healthcare. At the time of writing this entry, two releases of CDA were approved: CDA R1 was approved in 2000 and CDA R2 in 2005. Both releases are part of the HL7 new generation of standards (V3), all derived from a core reference information model (RIM) that assures semantic consistency across the various standards such as laboratory, medications, care provision and so forth. The RIM is based on common data types and vocabularies, and together these components constitute the HL7 V3 Foundation that is an inherent part of the CDA standard specification.

Key Points

Clinical documents such as discharge summaries, operative notes and referral letters are ubiquitous in healthcare and currently exist mostly in paper. The computerized clinical document is similar in purpose to its paper counterpart and the clinician's narratives are a key component of both versions. Narratives are compositions based on the natural language of the writer, while computerized structuring of a document is limited to some computer language. The design of the CDA standard strives to bridge the gap between these "languages" especially when it comes to the mixture of structured and unstructured data intertwined to describe the same phenomena, while addressing two important goals: human readability and machine-processability. The drive to structure medical narratives is also challenging the thin line between art and craftsmanship in the medical practice [3].

The basic structure of a CDA document consists of a header and a body. The header represents an extensive set of metadata about the document such as time stamps, the type of document, encounter details and of course the identification of the patient and those who participated in the documented encounter or service. While the header is a structured part of the document and is similar in the two releases of CDA, the body consists of clinical data organized in sections and only in CDA R2 it enables the formal representation of structured data along with narratives [1]. Data is structured in clinical statements based on entries such as observations, medication administrations, or adverse events where several entries are associated into a compound clinical statement. Nevertheless, only the narrative parts of the CDA body are mandatory, which makes CDA easy to adopt if structured data is not yet available. It is even possible to simply wrap a non-XML document with the CDA header or create a document with a structured header and sections containing only narrative content. The purpose of this design is to encourage widespread adoption, while providing an information infrastructure to incrementally move toward structured documents, serving the goal of semantic interoperability between disparate health information systems.

Beside text, CDA can also accommodate images, sounds, and other multimedia content. It can be transferred within a message and can be understood independently, outside the relaying message and its sending and receiving systems. CDA documents are encoded in Extensible Markup Language (XML), and they derive their machine processable meaning from the RIM, coupled with specific vocabularies.

A CDA document is a collection of information that is intended to be legally authenticated and has to be maintained by an organization entrusted with its care (stewardship). Inherent in the HL7 CDA standard are mechanisms for dealing with the authentication and versioning of documents so that it can be used in medical records enterprise repositories as well as in cross-institutional sharing of personal health information to facilitate continuity of care.

Cross-references

► Electronic Health Record

Recommended Reading

1. Dolin R.H., Alschuler L, Boyer S, Beebe C, Behlen FM, Biron PV, Shabo A. HL7 Clinical Document Architecture, Release 2. J. Am. Med. Inform. Assoc., 13(1):30–39, 2006.
2. Health Level Seven (HL7) – http://www.hl7.org.
3. Shabo A. Synopsis of the Patient Records Section: Structuring the Medical Narrative in Patient Records – A Further Step towards a Multi-Accessible EHR. The IMIA 2004 Yearbook of Medical Informatics: Towards Clinical Bioinformatics, 2004.

Clinical Event

Dan Russler
Oracle Health Sciences, Redwood Shores, CA, USA

Definition

Vernacular Definition

1. In event planning circles, a "clinical event" is an event, e.g., meeting or party, attended by clinicians as opposed to administrative or financial personnel.

Technical Definitions

1. A state transition, normally a "create" or "update" state transition, targeting a record in an electronic medical record system or one of the systems associated with an electronic medical record system.
2. A report generated within a clinical trial that is subsequently evaluated for the presence of an adverse event by a clinical trial Clinical Event Committee.

Words often confused by use of the term "Clinical Event" include: Clinical Event (multiple definitions); Adverse Event; Clinical Act; Patient Event, Information Event.

The primary technical definition of "clinical event" includes the kind of "events" that are monitored by a "clinical event monitor"[1–3] used in synchronous or asynchronous decision support functions. Examples of these events include clinical orders, electronic medical record entries, admission, transfer and discharge notifications, lab results, and patient safety reports. These events trigger state transitions in an electronic medical record system or related system.

Typically, once the clinical event monitoring system, such as an HL7 Arden Syntax-based system, discovers a state transition, in the electronic medical record system, a decision support rule is applied to the clinical event and related data in order to determine whether a notification of a person or another system is required.

Key Points

"Event" or "Action" analysis traces its roots to the work of Aristotle on propositions. Propositions usually follow the form of Subject-Predicate and, upon analysis, may be found to be "true" or "false." The classic example of a proposition is "Socrates is a man." "Socrates" is the "Subject" and "is a man" is the "Predicate." An analogous proposition in healthcare is "Peter has a potassium level of 5.5 mg/dl." Clinical Events are propositions in healthcare that may be evaluated themselves by clinicians as "true" or "false" or may be applied in rules that evaluate to true or false.

For example, the creation of a record asserting that "Peter has a potassium level of 5.5 mg/dl" might trigger a clinical event monitoring system to implement the rule: "If potassium level record created, then evaluate if ("record value" >5.0); if "true," then notify Dr. X."

"Event-driven programming" as opposed to "procedural programming" utilizes the same kinds of predicate logic in evaluating state transitions or triggers to state transitions in a modern computer-programming environment. Consequently, Clinical Events drive programming logic in many modern systems.

The HL7 Reference Information Model (RIM) describes clinical events; the term "Act" in the RIM identifies objects that are instantiated in XML communications between systems or in records within the electronic healthcare systems themselves. These "Acts" correspond to "clinical events" used for monitoring systems in healthcare. However, in the RIM, "Event" is defined narrowly as an instance of an Act that has been completed or is in the process of being completed. Clinical event monitoring systems may also evaluate HL7 "Orders or Requests" or other kinds of "Act" instances as events of interest (www.hl7.org).

Cross-references

- ▶ Clinical Observation
- ▶ Clinical Order
- ▶ Interface Engines in Healthcare
- ▶ Event Driven Architecture
- ▶ HL7 Reference Information Model
- ▶ Predicate Logic
- ▶ Propositions

Recommended Reading

1. Glaser J., et al. Impact of information events on medical care. HIMSS, 1996.
2. Hripisak G., et al. Design of a clinical event monitor. Comp. Biomed. Res., 29:194–221, 1996.
3. McDonald C. Action-oriented Decisions in Ambulatory Medicine. Yearbook Medical Publishers, Chicago, IL, 1981.

Clinical Genetics

▶ Implications of Genomics for Clinical Informatics

Clinical Genomics

▶ Implications of Genomics for Clinical Informatics

Clinical Judgment

▶ Clinical Observation

Clinical Knowledge Base

▶ Clinical Knowledge Repository

Clinical Knowledge Directory

▶ Clinical Knowledge Repository

Clinical Knowledge Management Repository

▶ Clinical Knowledge Repository

Clinical Knowledge Repository

Roberto A. Rocha
Partners Healthcare System, Inc., Boston, MA, USA

Synonyms

Clinical knowledge base; Clinical content repository; Clinical content database; Clinical knowledge management repository; Clinical content registry; Clinical knowledge directory

Definition

A clinical knowledge repository (CKR) is a multipurpose storehouse for clinical knowledge assets. "Clinical knowledge asset" is a generic term that describes any type of human or machine-readable electronic content used for computerized clinical decision support. A CKR is normally implemented as an enterprise resource that centralizes a large quantity and wide variety of clinical knowledge assets. A CKR provides integrated support to all asset lifecycle phases such as authoring, review, activation, revision, and eventual inactivation. A CKR routinely provides services to search, retrieve, transform, merge, upload, and download clinical knowledge assets. From a content curation perspective, a CKR has to ensure proper asset provenance, integrity, and versioning, along with effective access and utilization constraints compatible with collaborative development and deployment activities. A CKR can be considered a specialized content management system, designed specifically to support clinical information systems. Within the context of clinical decision support systems, a CKR can be considered a special kind of knowledge base – one specially designed to manage multiple types of human and machine-readable clinical knowledge assets.

Key Points

In recent years, multiple initiatives have attempted to better organize, filter, and apply the ever-growing biomedical knowledge. Among these initiatives, one of the most promising is the utilization of computerized clinical decision support systems. Computerized clinical decision support can be defined as computer systems that provide the correct amount of relevant knowledge at the appropriate time and context, contributing to improved clinical care and outcomes. A wide variety of knowledge-driven tools and methods have resulted in multiple modalities of clinical decision support, including information selection and retrieval, information aggregation and presentation, data entry assistance, event monitors, care workflow assistance, and descriptive or predictive modeling. A CKR provides an integrated storage platform that enables the creation and maintenance of multiple types of knowledge assets. A CKR ensures that different modalities of decision support can be combined to properly support the activities of clinical workers. Core requirements guiding the implementation of a CKR include clinical knowledge asset provenance (metadata), versioning, and integrity. Other essential requirements include the proper representation of access and utilization constraints, taking into account the collaborative nature of asset development processes and deployment environments. Another fundamental requirement is to aptly represent multiple types of knowledge assets, where each type might require specialized storage and handling. The CKR core requirements are generally similar to those specified for other types of repositories used for storage and management of machine-readable assets.

Historical Background

Biomedical knowledge has always been in constant expansion, but unprecedented growth is being observed during the last decade. Over 30% of the 16.8 million citations accumulated by MEDLINE until December of 2007 were created in the last 10 years, with an average of over 525,000 new citations per year [5]. The number of articles published each year is commonly used as an indicator of how much new knowledge the scientific community is creating. However, from a clinical perspective, particularly for those involved with direct patient care, the vast amount of new knowledge represents an ever-growing gap between what is known and what is routinely practiced. Multiple initiatives in recent years have attempted to better organize, filter, and apply the knowledge being generated. Among these various initiatives, one of the most promising is the utilization of computerized clinical decision support systems [6]. In fact, some authors avow that clinical care currently mandates a degree of individualization that is inconceivable without computerized decision support [1].

Computerized clinical decision support can be defined as computer systems that provide the correct amount of relevant knowledge at the appropriate time and context, ultimately contributing to improved clinical care and outcomes [3]. Computerized clinical decision support has been an active area of informatics research and development for the last three decades [2]. A wide variety of knowledge-driven tools and methods have resulted in multiple modalities of clinical decision support, including information selection and retrieval (e.g., infobuttons, crawlers), information aggregation and presentation (e.g., summaries, reports, dashboards), data entry assistance (e.g., forcing functions, calculations, evidence-based templates for ordering and documentation), event monitors (e.g., alerts, reminders, alarms), care workflow assistance (e.g., protocols, care pathways, practice guidelines), and descriptive or predictive modeling (e.g., diagnosis, prognosis, treatment planning, treatment outcomes). Each modality requires specific types of knowledge assets, ranging from production rules to mathematical formulas, and from automated workflows to machine learning models. A CKR provides an integrated storage platform that enables the creation and maintenance of multiple types of assets using knowledge management best practices [4].

The systematic application of knowledge management processes and best practices to the biomedical domain is a relatively recent endeavor [2]. Consequently, a CKR should be seen as a new and evolving concept that is only now being recognized as a fundamental component for the acquisition, storage, and maintenance of clinical knowledge assets. Most clinical decision support systems currently in use still rely on traditional knowledge bases that handle a single type of knowledge asset and do not provide direct support for a complete lifecycle management process. Another relatively recent principle is the recognition that different modalities of decision support have to be combined and subsequently integrated with information systems to properly support the activities of clinical workers. The premise of integrating multiple modalities of clinical decision support reinforces the need for knowledge management processes supported by a CKR.

Foundations

Core requirements guiding the implementation of a CKR include clinical knowledge asset provenance (metadata), versioning, and integrity. Requirements associated with proper access and utilization constraints are also essential, particularly considering the collaborative nature of most asset development processes and deployment environments. Another fundamental requirement is to aptly represent multiple types of knowledge assets, where each type might require specialized storage and handling. The CKR core requirements are generally similar to those specified for other types of repositories used for storage and management of machine-readable assets (e.g., "ebXML Registry" (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev = regrep)).

Requirements associated with asset provenance can be implemented using a rich set of metadata properties that describe the origin, purpose, evolution, and status of each clinical knowledge asset. The metadata properties should reflect the information that needs to be captured during each phase of the knowledge asset lifecycle process, taking into account multiple iterative authoring and review cycles, followed by a possibly long period of clinical use that might require multiple periodic revisions (updates). Despite the diversity of asset types, each with a potentially distinct lifecycle process, a portion of the metadata properties should be consistently implemented, enabling basic searching and retrieval services across asset types. Ideally, the shared metadata should be based on metadata standards (e.g., "Dublin Core Metadata Element Set" (http://dublincore.org/documents/dces/)). The adoption of standard metadata properties also simplifies the integration of external collections of clinical knowledge assets in a CKR. In addition to a shared set of properties, a CKR should also accommodate extended sets of properties specific for each clinical knowledge asset type and its respective lifecycle process. Discrete namespaces are commonly used to represent type-specific extended metadata properties.

Asset version and status, along with detailed change tracking, are vital requirements for a CKR. Different versioning strategies can be used, but as a general rule there should be only one clinically active version of any given knowledge asset. This general rule is easily observed if the type and purpose of the clinical knowledge asset remains the same throughout its lifecycle. However, a competing goal is created with the very desirable evolution of human-readable assets to become machine-readable. Such evolution invariably requires the creation of new knowledge assets of different types and potentially narrower purposes. In order to support this "natural" evolution, a CKR should implement the concept of asset generations, while preserving the change history that links one generation to the next. Also within a clinical setting, it is not uncommon to have to ensure that knowledge assets comply with, or directly implement, different norms and regulations. As a result, the change history of a clinical knowledge asset should identify the standardization and compliance aspects considered, enabling subsequent auditing and/or eventual certification.

Ensuring the integrity of clinical knowledge assets is yet another vital requirement for a CKR. Proper integrity guarantees that each asset is unique within a specific type and purpose, and that all its required properties are accurately defined. Integrity requirements also take into account the definition and preservation of dependencies between clinical knowledge assets. These dependencies can be manifested as simple hyperlinks, or as integral content defined as another independent asset. Creating clinical knowledge assets from separate components or modules (i.e., modularity) is a very desirable feature in a CKR – one that ultimately contributes to the overall maintainability of the various asset collections. However, modularity introduces important integrity challenges, particularly when a new knowledge asset is being activated for clinical use. Activation for clinical use requires a close examination of all separate components, sometimes triggering unplanned revisions of components already in routine use. Another important integrity requirement is the ability to validate the structure and the content of a clinical knowledge asset against predefined templates (schemas) and dictionaries (ontologies). Asset content validation is essential for optimal integration with clinical information systems. Ideally, within a given healthcare organization all clinical information systems and the CKR should utilize the same standardized ontologies.

Contextual characteristics of the care delivery process establish the requirements associated with proper access, utilization, and presentation of the clinical knowledge assets. The care delivery context is a multidimensional constraint that includes characteristics of the patient (e.g., gender, age group, language, clinical condition), the clinical worker (e.g., discipline, specialty, role), the clinical setting (e.g., inpatient, outpatient, ICU, Emergency Department), and the information system being used (e.g., order entry, documentation, monitoring), among others. The care delivery context normally applies to the entire clinical knowledge asset, directly influencing search, retrieval, and presentation services. The care delivery context can also be used to constrain specific portions of a knowledge asset, including links to other embedded assets, making them accessible only if the constraints are satisfied. An important integrity challenge created by the systematic use of the care delivery context is the need for reconciling conflicts caused by incompatible asset constraints, particularly when different teams maintain the assets being combined. In this scenario, competing requirements are frequently present, namely the intention to maximize modularity and reusability versus the need to maximize clinical specificity and ease or use.

The accurate selection, retrieval, and presentation of unstructured assets is generally perceived as a simple but very useful modality of clinical decision support, particularly if the information presented to the clinical worker is concise and appropriate to the care being delivered. However, the appropriateness of the information is largely defined by the constraints imposed by the aforementioned care delivery context. Moreover, the extent of indexing ("retrievability") of most collections of unstructured clinical knowledge assets is not sufficient to fully recognize detailed care delivery context expressions. Ultimately, the care delivery context provides an extensible mechanism for defining the appropriateness of a given clinical knowledge asset in response to a wide variety of CKR service requests.

The requirements just described are totally or partially implemented as part of general-purpose (enterprise) content management systems. However, content management systems have been traditionally constructed for managing primarily human-readable electronic content. Human-readable content, more properly characterized as unstructured knowledge assets, include narrative text, diagrams, and multimedia objects. When combined, these unstructured assets likely represent the largest portion of the inventory of clinical knowledge assets of any healthcare institution. As a result, in recent years different healthcare organizations have deployed CKRs using enterprise content management systems, despite their inability to manage machine-readable content.

Key Applications

Computerized Clinical Decision Support, Clinical Knowledge Engineering, Clinical Information Systems.

Cross-references

- ▶ Biomedical Data/Content Acquisition, Curation
- ▶ Clinical Data Acquisition, Storage and Management
- ▶ Clinical Decision Support
- ▶ Dublin Core
- ▶ Evidence Based Medicine
- ▶ Executable Knowledge
- ▶ Metadata
- ▶ Reference Knowledge

Recommended Reading

1. Bates D.W. and Gawande A.A. Improving safety with information technology. N. Engl. J. Med. 348(25):2526–2534, 2003.
2. Greenes R.A (ed.). Clinical Decision Support: The road ahead. Academic Press, Boston, 2007, pp. 544.
3. Osheroff J.A., Teich J.M., Middleton B., Steen E.B., Wright A., and Detmer D.E. A roadmap for national action on clinical decision support. J. Am. Med. Inform. Assoc., 14(2):141–145, 2007.
4. Rocha R.A., Bradshaw R.L., Hulse N.C., and Rocha B.H.S.C. The clinical knowledge management infrastructure of Intermountain Healthcare. In: Clinical Decision Support: The road ahead, RA.Greenes (ed.). Academic Press, Boston, 2007, pp. 469–502.
5. Statistical Reports on MEDLINE®/PubMed® Baseline Data, National Library of Medicine, Department of Health and Human Services [Online]. Available at: http://www.nlm.nih.gov/bsd/licensee/baselinestats.html. Accessed 8 Feb 2008.
6. Wyatt J.C. Decision support systems. J. R. Soc. Med., 93(12): 629–633, 2000.

Clinical Nomenclatures

▶ Clinical Ontologies

Clinical Observation

Dan Russler
Oracle Health Sciences, Redwood Shores, CA, USA

Synonyms

Clinical result; Clinical judgment; Clinical test; Finding of observation

Definition

1. The act of measuring, questioning, evaluating, or otherwise observing a patient or a specimen from a patient in healthcare; the act of making a clinical judgment.
2. The result, answer, judgment, or knowledge gained from the act of observing a patient or a specimen from a patient in healthcare.

These two definitions of "observation" have caused confusion in clinical communications, especially when applying the term to the rigor of standardized terminologies. When developing a list of observations, the terminologists have differed on whether the list of terms should refer to the "act of observing" or the "result of the observation."

Logical Observation Identifiers Names and Codes (LOINC) (www.loinc.org) focus on observation as the "act of observing." Systematized Nomenclature of Medicine (SNOMED) (www.ihtsdo.org) asserts that "General finding of observation of patient" is a synonym for "General observation of patient." Of note is the analysis in HL7 that identifies many shared attributes between descriptions of the act of observing and the result obtained. As a consequence, in HL7 Reference Information Model (RIM), both the act of observing and the result of the observation are contained in the same Observation Class (www.hl7.org).

Key Points

The topic of clinical observation has been central to the study of medicine since medicine began. Early physicians focused on the use of all five senses in order to make judgments about the current condition of the patient, i.e., diagnosis, or to make judgments about the future of patients, i.e., prognosis. Physical exam included sight, touch, listening, and smell. Physicians diagnosed diabetes by tasting the urine for sweetness.

As more tests on bodily fluids and tissues were discovered and used, the opportunity for better diagnosis and prognosis increased. Philosophy of science through the centuries often included the study of clinical observation in addition to the study of other observations in nature.

During the last century, the study of rigorous testing techniques that improve the reproducibility and interpretation of results has included the development of extensive nomenclatures for naming the acts of observation and observation results, e.g., LOINC and SNOMED. These terminologies were developed in part to support the safe application of expert system rules to information recorded in the electronic health care record.

The development of the HL7 Reference Information Model (RIM) was based on analysis of the "act of observing" and the "result of the act of observing" [1]. Today, new Entity attributes proposed for the HL7 RIM are evaluated for inclusion based partly on whether the information is best communicated in a new attribute for an HL7 Entity or best communicated in an HL7 Observation Act.

Improved standardization of clinical observation techniques, both in the practice of bedside care and the recording of clinical observations in electronic healthcare systems is thought to be essential to the continuing improvement of healthcare and patient safety.

Cross-references

▶ Clinical Event

▶ Clinical Order

▶ Interface Engines in Healthcare

Recommended Reading

1. Russler D., et al. Influences of the unified service action model on the HL7 reference information model. In JAMIA Symposium Supplement, Proceedings SCAMC, 1999, pp. 930–934.

Clinical Ontologies

Yves A. Lussier, James L. Chen
University of Chicago, Chicago, IL, USA

Synonyms

Clinical terminologies; Clinical nomenclatures; Clinical classifications

Definition

An ontology is a formal representation of a set of heterogeneous concepts. However, in the life sciences, the term clinical ontology has also been more broadly defined as also comprising all forms of classified terminologies, including classifications and nomenclatures. Clinical ontologies provide not only a controlled vocabulary but also relationships among concepts allowing computer reasoning such that different parties, like physicians and insurers, can efficiently answer complex queries.

Historical Background

As the life sciences integrates increasingly sophisticated systems of patient management, different means of data representation have had to keep pace to support user systems. Simultaneously, the explosion of genetic information from breakthroughs from the Human Genome Project and gene chip technology have further expedited the need for robust, scalable platforms for handling heterogeneous data. Multiple solutions have been developed by the scientific community to answer these challenges at all different levels of biology.

This growing field of "systems medicine" starts humbly at the question – how can one best capture and represent complex data in a means that can be understood globally without ambiguity? In other words, does the data captured have the same semantic validity after retrieval as it did prior? These knowledgebases are in of themselves organic. They need to be able to expand, shrink, and rearrange themselves based on user or system needs. This entry will touch upon existing clinical ontologies used in a variety of applications.

Foundations

The complexity of biological data cannot be understated. Issues generally fall into challenges with (i) definition, (ii) context, (iii) composition, and (iv) scale. One cannot even take for granted that the term "genome" is well-understood. Mahner found five different characterizations for the term "genome" [8]. Ontologies then provide a means of providing representational consistency through their structure and equally important provide the ability to connect these terms together in a semantically informative and computationally elegant manner [9]. This has led to their ubiquity in the life sciences. Formal ontologies are designated using frames or description logics [5]. However, few life science knowledgebases are represented completely in this manner due to difficulties with achieving consensus on definitions regarding the terms and the effort required to give context to the terms. Thus, this article defines well-organized nomenclatures and terminologies as clinical ontologies – regardless if their terms adhere to strict formalism.

Looking at elevations in gene expression, it matters what organism and under what experimental conditions the experiment was conducted. Clinical context changes the meaning of terms. The term "cortex" can either indicate a part of the kidney or that of the brain. Generalized or "essential hypertension" can be what is known colloquially as "high blood pressure" or localized to the lungs as "pulmonary hypertension." One can have pulmonary hypertension but not essential hypertension. This leads to the next representational challenge – that of composition. Should hypertension be represented implicitly as "essential hypertension" and as "pulmonary hypertension"? Or should it be stored explicitly as "hypertension" with a location attribute? These representational decisions are driven by the queries that may be asked. The difficulty arises in anticipating the queries and in post-processing of the query to split the terminological components of the overall concept. Finally, the knowledge model needs to be able to scale upward. The same decision logic that was relevant when the knowledgebase contained 100 concepts needs to still be relevant at 1,000,000 concepts.

Properties of Clinical Ontologies

Ontologies vary widely in their degree of formalism and design. With this comes differing computability. In 1998, Cimino proposed desirable properties for purposes of clinical computation [3,4]. Table 1 summarizes the overall properties of the commonly used clinical ontologies.

1. Concept-oriented: a single concept is the preferred unit
2. Formal semantic definition: well-defined terms
3. Nonredundancy: each concept needs to be unique
4. Nonambiguity: different concepts should not overlap or be conflated
5. Relationships: the structure of connections between concepts differentiate ontologies:

- Monohierarchy (tree): each concept only has one parent concept
- Polyhierarchy: each concept may multiply inherit from multiple parents
- Directed Acycle Graph (DAG): there are no cycles in the graph – in other words, children concepts may not point to parent terms

Key Applications

This section reviews different, well-used life science ontologies used to annotate datasets. First, this discussion summarizes a select number of archetypal clinical

Clinical Ontologies. Table 1. Properties of clinical ontologies

	Architecture					
Ontology	Concept oriented	Formal semantic definition	Concept permanence	Nonredundancy	Uniqueness	Relationship
ICD-9	+		±	+	+	M
LOINC	±		+	+		P
CPT	±		±			M
SNOMED	+	+	+	+	+	DAG
UMLS	+		+	+	+	CG

M = Monohierarchy/tree, P = Polyhierarchy, DAG = Directed Acyclic Graph, CG = cyclic graph

Clinical Ontologies. Table 2. Coverage of classification, nomenclatures and ontologies

Ontology	Content						Number of concepts (order of magnitude)
	Diseases	Anatomy	Morphology	Labs	Procedures	Drugs	
ICD-9	X						10^4
LOINC				X			10^5
CPT					X		10^4
SNOMED	X	X	X	X	X	X	10^5
UMLS	X	X	X	X	X	X	10^6

ontologies that comprise one or several types of clinical entities such as diseases, clinical findings, procedures, laboratory measurements, and medications. Table 2 below summarizes the content coverage of each of archetypal health ontologies.

Prototypical Clinical Ontologies

a. The Systematized Nomenclature of Medicine (SNOMED CT) SNOMED CT is the most extensive set of publically available collection of clinical concepts. It is organized as a directed acyclic graph (DAG) and contains class/subclass relationships and partonomy relationships. It is maintained by the College of American Pathologists and is available in the United States through a license from the National Library of Medicine in perpetuity. SNOMED CT is one of the designated data standards for use in U.S. Federal Government systems for the electronic exchange of clinical health information. SNOMED CT is now owned by the International Healthcare Terminology Standards Development Organization [6].

b. International Statistical Classification of Diseases (ICD-9, ICD-10, ICD-CM) ICD-9 and ICD 10 are detailed ontologies of disease and symptomatology used ubiquitously for reimbursement systems (i.e., Medicare/Medicaid) and automated decision support in medicine. ICD-10 is used worldwide for morbidity and mortality statistics. Owned by the World Health Organization (WHO), licenses are available generally free for research. ICD-9 CM is a subtype of ICD-9 with clinical modifiers for billing purposes [11].

c. Medical Subject Headings (MeSH) MeSH grew out of an effort by the NLM for indexing life science journal articles and books. {Nelson S.J., 2001 #6}. The extensive controlled vocabulary MeSH serves as the backbone of the MEDLINE/PubMed article database. MeSH can be browsed and downloaded free of charge on the Internet [10].

d. International Classification of Primary Care (ICPC-2, ICPC-2-E) ICPC is a primary care encounter classification system [12]. It has a biaxial structure of 17 clinical systems and 7 types of data. It allows for the classification of the patient's reason for encounter (RFE), the problems/diagnosis managed, primary care interventions, and the ordering of the data. of the primary care session in an episode of care structure. ICPC-2-E refers to a revised electronic version.

e. Diagnostic and Statistical Manual of Mental Disorders (DSM-IV, DSM-V) The DSM is edited and published by the American Psychiatric Association provides categories of and diagnosis criteria for mental disorders [2]. It is used extensively by clinicians, policy makers and insurers. The original version of the DSM was published in 1962. DSM-V is due for publication in May 2012. The diagnosis codes are developed to be compatible with ICD-9.

f. Logical Observation Identifiers Names and Codes (LOINC) LOINC is a database protocol aimed at standardizing laboratory and clinical codes. The Regenstrief Institute, Inc, maintains the LOINC database and supporting documentation. LOINC is endorsed by the American Clinical Laboratory Association and College of American Pathologist and is one of the accepted standards by the US Federal Government for information exchange [7].

g. Current Procedural Terminology (CPT) The CPT code set is owned and maintained by the American Medical Association through the CPT Editorial Panel [1]. The CPT code set is used extensively to communicate medical and diagnostic services that were rendered among physicians and payers. The current version is the CPT 2008.

Cross-references

- ▶ Anchor text
- ▶ Annotation
- ▶ Archiving Experimental Data
- ▶ Biomedical Data/Content Acquisition, Curation
- ▶ Classification
- ▶ Clinical Data Acquisition, Storage and Management
- ▶ Clinical Data and Information Models
- ▶ Clinical Decision Support
- ▶ Data Integration Architectures and Methodology for the Life Sciences
- ▶ Data Types in Scientific Data Management
- ▶ Data Warehousing for Clinical Research
- ▶ Digital Curation
- ▶ Electronic Health Record
- ▶ Fully-Automatic Web Data Extraction
- ▶ Information Integration Techniques for Scientific Data
- ▶ Integration of Rules and Ontologies
- ▶ Logical Models of Information Retrieval
- ▶ Ontologies
- ▶ Ontologies and Life Science Data Management
- ▶ Ontology
- ▶ Ontology Elicitation
- ▶ Ontology Engineering
- ▶ Ontology Visual Querying
- ▶ OWL: Web Ontology Language
- ▶ Query Processing Techniques for Ontological Information
- ▶ Semantic Data Integration for Life Science Entities
- ▶ Semantic Web
- ▶ Storage Management
- ▶ Taxonomy: Biomedical Health Informatics
- ▶ Web Information Extraction

Recommended reading

1. American Medical Association [cited; Available at: http://www.cptnetwork.com].
2. American Psychiatric Association [cited; Available at: http://www.psych.org/MainMenu/Research/DSMIV.aspx].
3. Cimino J.J. Desiderata for controlled medical vocabularies in the twenty-first century. Methods Inf. Med., 37(4–5):394–403, 1998.
4. Cimino J.J. In defense of the Desiderata. [comment]. J. Biomed. Inform., 39(3):299–306, 2006.
5. Gruber T.R. Toward principles for the design of ontologies used for knowledge sharing. Int. J. Hum. Comput. Stud., 43(4–5): 907–928, 1995.
6. I.H.T.S.D. [cited; Available from: http://www.ihtsdo.org/our-standards/snomed-ct].
7. Khan A.N. et al. Standardizing laboratory data by mapping to LOINC. J Am Med Inform Assoc, 13(3):353–355, 2006.
8. Mahner M. and Kary M. What exactly are genomes, genotypes and phenotypes? And what about phenomes? J. Theor. Biol., 186(1):55–63, 1997.
9. Musen M.A. et al. PROTEGE-II: computer support for development of intelligent systems from libraries of components. Medinfo, 8 (Pt 1):766–770, 1995.
10. Nelson S.J., Johnston D., and Humphreys. B.L Relationships in medicical subject headings. In Relationships in the Organization of Knowledge, A.B. Carol, G. Rebecca (eds.). Kluwer, Dordecht, 2001, pp. 171–184.
11. World Health Organization [cited; Available at: http://www.who.int/classifications/icd/en/].
12. World Organization of National Colleges, Academies, and Academic Associations of General Practitioners/Family Physicians, ICPC. International Classification of Primary Care. Oxford University Press, Oxford, 1987.

Clinical Order

Dan Russler
Oracle Health Sciences, Redwood Shores, CA, USA

Synonyms

Order item; Service order; Service request; Service item; Procedure order; Procedure request

Definition

The act of requesting that a service be performed for a patient.

Clinical orders in healthcare share many characteristics with purchase orders in other industries. Both clinical orders and purchase orders establish a customer-provider relationship between the person placing the request for a service to be provided and the person or organization filling the request. In both cases, the clinical order and purchase order are followed by either a promise or intent to fill the request, a decline to fill the request, or a counter-proposal to provide an alternate service. In both scenarios, an authorization step such as an insurance company authorization or a credit company authorization may be required. Therefore, the dynamic flow of communications between a placer and filler in a clinical order management system and a purchase order management system are very similar.

Both clinical order and purchase order management systems maintain a catalog of items that may be requested. These items in both kinds of systems may represent physical items from supply or services from a service provider. Each of these items in both kinds of systems is associated with an internally unique identifier, a text description, and often a code. Dates, status codes, delivery locations, and other attributes of a clinical order and purchase order are also similar. Therefore, in addition to similarities in the dynamic flow of order communications, the structure of the content in clinical orders and purchase orders is similar.

Logical Observation Identifiers Names and Codes (LOINC) (www.loinc.org) describe many of the requested services in healthcare, especially in laboratory systems. Other procedural terminologies exist for healthcare, either independently in terminologies like LOINC or included in more comprehensive terminologies such as Systematized Nomenclature of Medicine (SNOMED) (www.ihtsdo.org).

Key Points

Clinical orders exist in the context of a larger clinical management, process. The order management business process of an organization, that includes defining a catalog of services to be provided and then allowing people to select from the catalog of services, is common in many industries. However, the decision support opportunities for helping providers select the optimum set of services for a patient are often more complex in healthcare than occurs in other industries. The outcomes of this selection process are studied in clinical research, clinical trials on medications and devices, and in organizational quality improvement initiatives. Finally, the outcomes of the service selection process are used to improve the clinical decision support processes utilized by providers selecting services for patients. This business process in healthcare as well as in many other industries describes a circular feedback loop defined by the offering of services, the selection of services, the delivery of services, the outcome of services, and finally, the modification of service selection opportunities and decision support.

In the HL7 Reference Information Model (RIM), "ACT" classes sub-typed with the moodCode attribute support the healthcare improvement process (www.hl7.org). These objects with process "moods" support the sequence of objects created during the execution of a process defined in Business Process Execution Language (BPEL) in a service oriented architecture that begins with an "order", evolves into an "appointment", which then is completed as an "event". The reason the term "mood" is used is that the values of the moodcode attribute are analogous to the models of verbs in many languages, e.g., the "Definition mood" used to define service catalogs corresponds to the "infinitive" verbal mood, i.e., a possible action; the "Request or Order mood" corresponds to the "imperative" verbal mood; the "Event mood" corresponds to the "indicative" verbal mood; and the "Goal mood," which describes the desired outcome of the selected service, corresponds to the "subjunctive" verbal mood.

Cross-references

- ▶ Clinical Event
- ▶ Clinical Observation
- ▶ Interface Engines in Healthcare

Clinical Research Chart

- ▶ Data Warehousing for Clinical Research

Clinical Result

- ▶ Clinical Observation

Clinical Terminologies

- ▶ Clinical Ontologies

Clinical Test

- ▶ Clinical Observation

Clock

- ▶ Physical Clock
- ▶ Time-Line Clock

Closed Itemset Mining and Non-redundant Association Rule Mining

Mohammed J. Zaki
Rensselaer Polytechnic Institute, Troy, NY, USA

Synonyms

Frequent concepts; Rule bases

Definition

Let I be a set of binary-valued attributes, called *items*. A set $X \subseteq I$ is called an *itemset*. A transaction database D is a multiset of itemsets, where each itemset, called a transaction, has a unique identifier, called a tid. The *support* of an itemset X in a dataset D, denoted $sup(X)$, is the fraction of transactions in D where X appears as a subset. X is said to be a *frequent* itemset in D if $sup(X) \geq minsup$, where $minsup$ is a user defined minimum support threshold. An (frequent) itemset is called *closed* if it has no (frequent) superset having the same support.

An *association rule* is an expression $A \Rightarrow B$, where A and B are itemsets, and $A \cap B = \emptyset$. The *support* of the rule is the joint probability of a transaction containing both A and B, given as $sup(A \Rightarrow B) = P(A \wedge B) = sup(A \cup B)$. The *confidence* of a rule is the conditional probability that a transaction contains B, given that it contains A, given as: $conf(A \Rightarrow B) = P(B|A) = \frac{P(A \wedge B)}{P(A)} = \frac{sup(A \cup B)}{sup(A)}$. A rule is frequent if the itemset $A \cup B$ is frequent. A rule is confident if $conf \geq minconf$, where $minconf$ is a user-specified minimum threshold. The aim of non-redundant association rule mining is to generate a *rule basis*, a small, non-redundant set of rules, from which all other association rules can be derived.

Historical Background

The notion of closed itemsets has its origins in the elegant mathematical framework of Formal Concept Analysis (FCA) [3], where they are called *concepts*. The task of mining frequent closed itemsets was independently proposed in [7,11]. Approaches for non-redundant association rule mining were also independently proposed in [1,9]. These approaches rely heavily on the seminal work on rule bases in [5,6]. Efficient algorithms for mining frequent closed itemsets include CHARM [10], CLOSET [8] and several new approaches described in the Frequent Itemset Mining Implementations workshops [4].

Foundations

Let $I = \{i_1, i_2, \ldots, i_m\}$ be the set of items, and let $T = \{t_1, t_2, \ldots, t_n\}$ be the set of tids, the transaction identifiers. Just as a subset of items is called an itemset, a subset of tids is called a tidset. Let $\mathbf{t} : 2^I \rightarrow 2^T$ be a function, defined as follows:

$$\mathbf{t}(X) = \{t \in T \mid X \subseteq \mathbf{i}(t)\}$$

That is, $\mathbf{t}(X)$ is the set of transactions that contain *all* the items in the itemset X. Let $\mathbf{i} : 2^T \rightarrow 2^I$ be a function, defined as follows:

$$\mathbf{i}(Y) = \{i \in I \mid \forall t \in Y,\ t \text{ contains x}\}$$

That is, $\mathbf{i}(T)$ is the set of items that are contained in *all* the tids in the tidset Y. Formally, an itemset X is closed if $\mathbf{i} \circ \mathbf{t}(X) = X$, i.e., if X is a fixed-point of the closure operator $\mathbf{c} = \mathbf{i} \circ \mathbf{t}$. From the properties of the closure operator, one can derive that X is the maximal itemset that is contained in all the transactions $\mathbf{t}(X)$, which gives the simple definition of a closed itemset, namely, a closed itemset is one that has no superset that has the same support.

Based on the discussion above, three main families of itemsets can be distinguished. Let $\mathcal{F}$ denote the set of all frequent itemsets, given as

$$\mathcal{F} = \{X \mid X \subseteq I \text{ and } sup(\mathrm{X}) \geq minsup\}$$

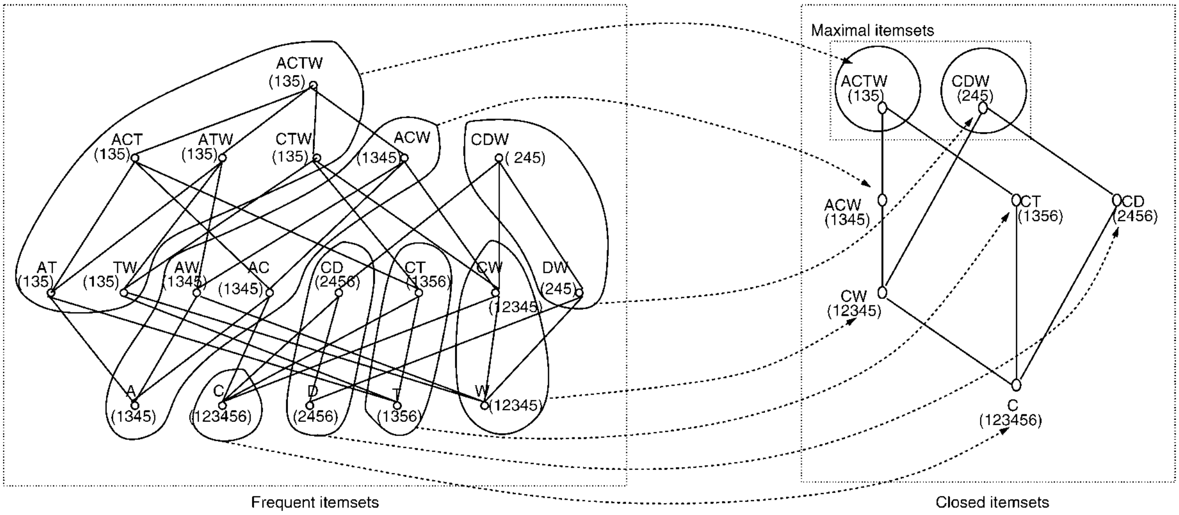
Let $\mathcal{C}$ denote the set of all closed frequent itemsets, given as

$$\mathcal{C} = \{X|X \in \mathcal{F} \text{ and } \nexists \mathrm{Y} \supset \mathrm{X} \text{ with } sup(\mathrm{X}) = sup(\mathrm{Y})\}$$

Finally, let $\mathcal{M}$ denote the set of all *maximal* frequent itemsets, given as

$$\mathcal{M} = \{X|X \in \mathcal{F} \text{ and } \nexists \mathrm{Y} \supset \mathrm{X}, \text{ such that } \mathrm{Y} \in \mathcal{F}\}$$

The following relationship holds between these sets: $\mathcal{M} \subseteq \mathcal{C} \subseteq \mathcal{F}$, which is illustrated in Fig. 1, based on the example dataset shown in Table 1 and using minimum support $minsup = 3$. The *equivalence classes* of itemsets that have the same tidsets have been shown clearly; the largest itemset in each equivalence class is a closed itemset. The figure also shows that the maximal itemsets are a subset of the closed itemsets.



Closed Itemset Mining and Non-redundant Association Rule Mining. Figure 1. Frequent, closed frequent and maximal frequent itemsets.

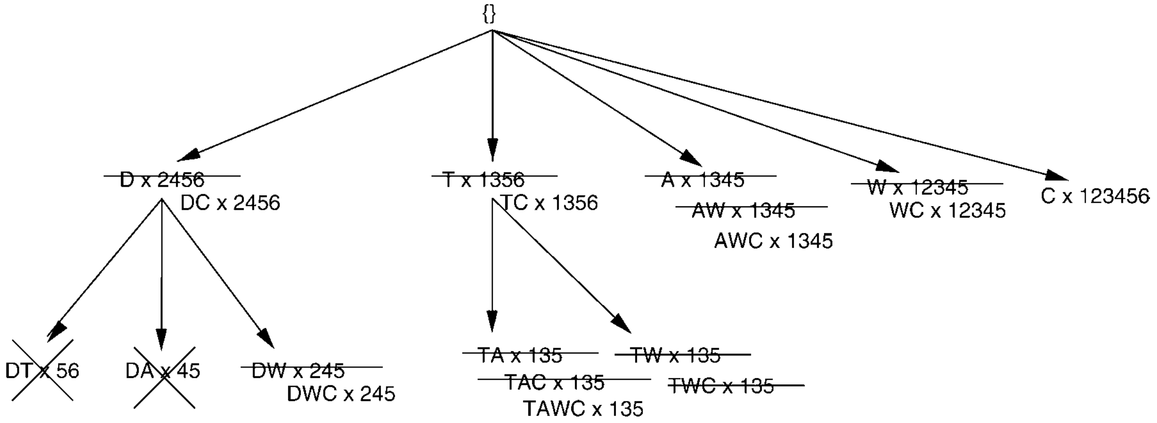
Closed Itemset Mining and Non-redundant Association Rule Mining. Table 1. Example transaction dataset

	i(*t*)
1	ACTW
2	CDW
3	ACTW
4	ACDW
5	ACDTW
6	CDT

Mining Closed Frequent Itemsets

CHARM [8] is an efficient algorithm for mining closed itemsets. Define two itemsets X, Y of length k as belonging to the same *prefix equivalence class*, $[P]$, if they share the $k - 1$ length prefix P, i.e., $X = Px$ and $Y = Py$, where $x,y \in I$. More formally, $[P] = \{Px_i \mid x_i \in I\}$, is the class of all itemsets sharing P as a common prefix. In CHARM there is no distinct candidate generation and support counting phase. Rather, counting is simultaneous with candidate generation. For a given prefix class, one performs intersections of the tidsets ofall pairs of itemsets in the class, and checks if the resulting tidsets have cardinality at least *minsup*. Each resulting frequent itemset generates a new class which will be expanded in the next step. That is, for a given class of itemsets with prefix P, $[P] = \{Px_1, Px_2, ..., Px_n\}$, one performs the intersection of Px_i with all Px_j with $j > i$ to obtain a new class $[Px_i] = [P']$ with elements $P'x_j$ provided the itemset Px_ix_j is frequent. The computation progresses recursively until no more frequent itemsets are produced. The initial invocation is with the class of frequent single items (the class $[\emptyset]$). All tidset intersections for pairs of class elements are computed. However in addition to checking for frequency, CHARM eliminates branches that cannot lead to closed sets, and grows closed itemsets using subset relationships among tidsets. There are four cases: if $\mathbf{t}(X_i) \subset \mathbf{t}(X_j)$ or if $\mathbf{t}(X_i) = \mathbf{t}(X_j)$, then replace every occurrence of X_i with $X_i \cup X_j$, since whenever X_i occurs X_j also occurs, which implies that $\mathbf{c}(X_i) \subseteq \mathbf{c}(X_i \cup X_j)$. If $\mathbf{t}(X_i) \supset \mathbf{t}(X_j)$ then replace X_j for the same reason. Finally, further recursion is required if $\mathbf{t}(X_i) \neq \mathbf{t}(X_j)$. These four properties allow CHARM to efficiently prune the search tree (for additional details see [10]).

Figure 2 shows how CHARM works on the example database shown in Table 1. First, CHARM sorts the items in increasing order of support, and initializes the root class as $[\emptyset] = \{D \times 2456,\ T \times 1356,\ A \times 1345,\ W \times 12345,\ C \times 123456\}$. The notation $D \times 2456$ stands for the itemset D and its tidset $\mathbf{t}(D) = \{2,4,5,6\}$. CHARM first processes the node $D \times 2456$; it will be combined with the sibling elements. DT and DA are not frequent and are thus pruned. Looking at W, since $\mathbf{t}(D) \neq \mathbf{t}(W)$, W is inserted in the new equivalence class $[D]$. For C, since $\mathbf{t}(D) \subset \mathbf{t}(C)$, all occurrences of D are replaced with DC, which means that $[D]$ is also changed to $[DC]$, and the element DW to DWC. A recursive call with class $[DC]$ is then made and since



Closed Itemset Mining and Non-redundant Association Rule Mining. Figure 2. CHARM: mining closed frequent itemsets.

there is only a single itemset *DWC*, it is added to the set of closed itemsets $\mathcal{C}$. When the call returns to *D* (i.e., *DC*) all elements in the class have been processed, so *DC* itself is added to $\mathcal{C}$.

When processing *T*, $\mathbf{t}(T) \neq \mathbf{t}(A)$, and thus CHARM inserts *A* in the new class $[T]$. Next it finds that $\mathbf{t}(T) \neq \mathbf{t}(W)$ and updates $[T] = \{A,W\}$. When it finds $\mathbf{t}(T) \subset \mathbf{t}(C)$ it updates all occurrences of *T*with *TC*. The class $[T]$ becomes $[TC] = \{A,W\}$. CHARM then makes a recursive call to process $[TC]$. When combining *TAC* with *TWC* it finds $\mathbf{t}(TAC) = \mathbf{t}(TWC)$, and thus replaces *TAC* with *TACW*, deleting *TWC* at the same time. Since *TACW* cannot be extended further, it is inserted in $\mathcal{C}$. Finally, when it is done processing the branch *TC*, it too is added to $\mathcal{C}$. Since $\mathbf{t}(A) \subset \mathbf{t}(W) \subset \mathbf{t}(C)$ no new recursion is made; the final set of closed itemsets $\mathcal{C}$ consists of the uncrossed itemsets shown in Fig. 2.

Non-redundant Association Rules

Given the set of closed frequent itemsets $\mathcal{C}$, one can generate all non-redundant association rules. There are two main classes of rules: (i) those that have 100% confidence, and (ii) those that have less than 100% confidence [9]. Let X_1 and X_2 be closed frequent itemsets. The 100% confidence rules are equivalent to those directed from X_1 to X_2, where $X_2 \subseteq X_1$, i.e., from a superset to a subset (not necessarily proper subset). For example, the rule $C \Rightarrow W$ is equivalent to the rule between the closed itemsets $\mathbf{c}(W) \Rightarrow \mathbf{c}(C) \equiv CW \Rightarrow C$. Its support is $sup(CW) = 5/6$, and its confidence is $\frac{sup(CW)}{sup(W)} = 5/5 = 1$, i.e., 100%. The less than 100% confidence rules are equivalent to those from X_1 to X_2 where $X_1 \subset X_2$, i.e., from a subset to a proper superset. For example, the rule $W \Rightarrow T$ is equivalent to the rule $\mathbf{c}(W) \Rightarrow \mathbf{c}(W \cup T) \equiv CW \Rightarrow ACTW$. Its support is $sup(TW) = 3/6 = 0.5$, and its confidence is $\frac{sup(TW)}{sup(W)} = 3/5 = 0.6$ or 60%. More details on how to generate these non-redundant rules appears in [9].

Key Applications

Closed itemsets provide a loss-less representation of the set of all frequent itemsets; they allow one to determine not only the frequent sets but also their exact support. At the same time they can be orders of magnitude fewer. Likewise, the non-redundant rules provide a much smaller, and manageable, set of rules, from which all other rules can be derived. There are numerous applications of these methods, such as market basket analysis, web usage mining, gene expression pattern mining, and so on.

Future Directions

Closed itemset mining has inspired a lot of subsequent research in mining compressed representations or summaries of the set of frequent patterns; see [2] for a survey of these approaches. Mining compressed pattern bases remains an active area of study.

Experimental Results

A number of algorithms have been proposed to mine frequent closed itemsets, and to extract non-redundant rule bases. The Frequent Itemset Mining Implementations (FIMI) Repository contains links to many of the latest implementations for mining closed itemsets. A report on the comparison of these methods also appears in [4]. Other implementations can be obtained from individual author's websites.

Data Sets

The FIMI repository has a number of real and synthetic datasets used in various studies on closed itemset mining.

Url to Code

The main FIMI website is at http://fimi.cs.helsinki.fi/, which is also mirrored at: http://www.cs.rpi.edu/~zaki/FIMI/

Cross-references

▶ Association Rule Mining on Streams
▶ Data Mining

Recommended Reading

1. Bastide Y., Pasquier N., Taouil R., Stumme G., and Lakhal L. Mining minimal non-redundant association rules using frequent closed itemsets. In Proc. 1st Int. Conf. Computational Logic, 2000, pp. 972–986.
2. Calders T., Rigotti C., and Boulicaut J.-F. A Survey on Condensed Representation for Frequent Sets. In Constraint-based Mining and Inductive Databases, LNCS, Vol. 3848, J-F. Boulicaut, L. De Raedt, and H. Mannila (eds.). Springer, 2005, pp. 64–80.
3. Ganter B. and Wille R. Formal Concept Analysis: Mathematical Foundations. Springer, Berlin Heidelberg New York, 1999.
4. Goethals B. and Zaki M.J. Advances in frequent itemset mining implementations: report on FIMI'03. SIGKDD Explor., 6(1): 109–117, June 2003.
5. Guigues J.L. and Duquenne V. Familles minimales d'implications informatives resultant d'un tableau de donnees binaires. Math. Sci. hum., 24(95):5–18, 1986.
6. Luxenburger M. Implications partielles dans un contexte. Math. Inf. Sci. hum., 29(113):35–55, 1991.
7. Pasquier N., Bastide Y., Taouil R., and Lakhal L. Discovering frequent closed itemsets for association rules. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 398–416.
8. Pei J., Han J., and Mao R. Closet: An efficient algorithm for mining frequent closed itemsets. In Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp. 21–30.
9. Zaki M.J. Generating non-redundant association rules. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 34–43.
10. Zaki M.J. and Hsiao C.-J. CHARM: An efficient algorithm for closed itemset mining. In Proc. SIAM International Conference on Data Mining, 2002, pp. 457–473.
11. Zaki M.J. and Ogihara M. Theoretical foundations of association rules. In Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 1998.

Closest Pairs

▶ Closest-Pair Query

Closest-Pair Query

Antonio Corral[1], Michael Vassilakopoulos[2]
[1]University of Almeria, Almeria, Spain
[2]University of Central Greece, Lamia, Greece

Synonyms

Closest pairs; k-Closest pair query; k-Distance join; Incremental k-distance join; k-Closest pair join

Definition

Given two sets P and Q of objects, a closest pair (CP) query discovers the pair of objects (p, q) with a distance that is the smallest among all object pairs in the Cartesian product P×Q. Similarly, a k closest pair query (k-CPQ) retrieves k pairs of objects from P and Q with the minimum distances among all the object pairs. In spatial databases, the distance is usually defined according to the Euclidean metric, and the set of objects P and Q are disk-resident. Query algorithms aim at minimizing the processing cost and the number of I/O operations, by using several optimization techniques for pruning the search space.

Historical Background

The closest pair query, has been widely studied in computational geometry. More recently, this problem has been approached in the context of spatial databases [4,8,12,14]. In spatial databases, existing algorithms assume that P and Q are indexed by a spatial access method (usually an R-tree [1]) and utilize some pruning bounds and heuristics to restrict the search space.

[8] was the first to address this issue, and proposed the following distance-based algorithms: incremental distance join, k distance join and k distance semijoin between two R-tree indices. The incremental processing reports one-by-one the desired elements of the result in ascending order of distance (k is unknown in advance and the user can stop when he/she is satisfied by the result). The algorithms follow the Best-First (BF) traversal policy, which keeps a heap with the entries of the nodes visited so far (it maintains a priority queue which contains pairs of index entries

and objects, and pop out the closest pair and process it). BF is near-optimal for CP queries; i.e., it only visits the pairs of nodes necessary for obtaining the result with a high probability. In [12] several modifications to the algorithms of [8] had been proposed in order to improve performance. Mainly, a method was proposed for selecting the sweep axis and direction for the plane sweep technique in bidirectional node expansion which minimizes the computational overhead of [8].

Later, an improved version of BF and several algorithms that follow Depth-First (DF) traversal ordering from the non-incremental point of view (which assumes that k is known in advance and reports the k elements of the result all together at the end of the algorithm) was proposed in [4]. In general, a DF algorithm visit the roots of the two R-trees and recursively follows the pair of entries $< E_P, E_Q >$, $E_P \in R_P$ and $E_Q \in R_Q$, whose MINMINDIST is the minimum distance among all pairs. At the opposite of BF, DF is sub-optimal, i.e., it accesses more nodes than necessary. The main disadvantage of BF with respect to DF is that it may suffer from buffer thrashing if the available memory is not enough for the heap (it is space-consuming), when a great quantity of elements of the result is required. In this case, part of the heap must be migrated to disk, incurring frequent I/O accesses. The implementation of DF is by recursion, which is available in most of the programming languages, and linear-space consuming with respect to the height of the R-trees. Moreover, BF is not favored by page replacement policies (e.g., LRU), as it does not exhibit locality between I/O accesses.

Another interesting contribution to the CP query was proposed by [14], in which a new structure called the b-Rdnn tree was presented, along with a better solution to the k-CP query when there is high overlap between the two datasets. The main idea is to find k objects from each dataset which are the closest to the other dataset.

There are a lot of papers related to k-CP query, like buffer query [3], iceberg distance join query [13], multi-way distance join query [6], k-nearest neighbor join [2], closest pair query with spatial constraints [11], etc. For example, a buffer query [3] involves two spatial datasets and a distance threshold ρ; the answer to this query is a set of pairs of spatial objects, one from each input dataset, that are within distance ρ of each other.

Foundations

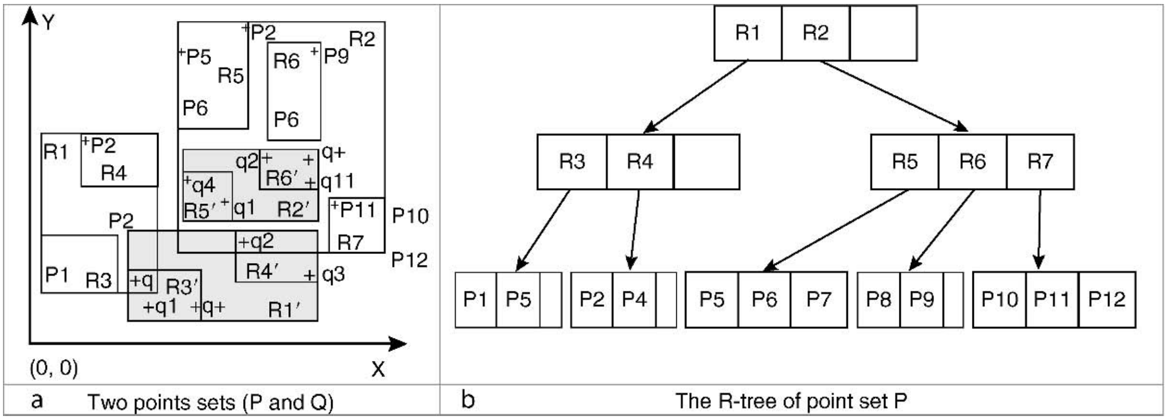
In spatial databases, existing algorithms assume that sets of spatial objects are indexed by a spatial access method (usually an R-tree [1]) and utilize some pruning bounds to restrict the search space. An R-tree is a hierarchical, height balanced multidimensional data structure, designed to be used in secondary storage based on B-trees. The R-trees are considered as excellent choices for indexing various kinds of spatial data (points, rectangles, line-segments, polygons, etc.). They are used for the dynamic organization of a set of spatial objects approximated by their Minimum Bounding Rectangles (MBRs). These MBRs are characterized by *min* and *max* points of rectangles with faces parallel to the coordinate axis. Using the MBR instead of the exact geometrical representation of the object, its representational complexity is reduced to two points where the most important features of the spatial object (position and extension) are maintained. The R-trees belong to the category of data-driven access methods, since their structure adapts itself to the MBRs distribution in the space (i.e., the partitioning adapts to the object distribution in the embedding space). Figure 1a shows two points sets P and Q (and the node extents), where the closest pair is (p_8, q_8), and Fig. 1b is the R-tree for the point set $P = \{p_1, p_2, \ldots, p_{12}\}$ with a capacity of three entries per node (branching factor or fan-out).

Assuming that the spatial datasets are indexed on any spatial tree-like structure belonging to the R-tree family, then the main objective while answering these types of spatial queries is to reduce the search space. In [5], three MBR-based distance functions to be used in algorithms for CP queries were formally defined, as an extension of the work presented in [4]. These metrics are MINMINDIST, MINMAXDIST and MAXMAXDIST. MINMINDIST (M_1, M_2) between two MBRs is the minimum possible distance between any point in the first MBR and any point in the second MBR. Maxmaxdist between two MBRs (M_1, M_2) is the maximum possible distance between any point in the first MBR and any point in the second MBR. Finally, MINMAXDIST between two MBRs (M_1, M_2) is the minimum of the maximum distance values of all the pairs of orthogonal faces to each dimension. Formally, they are defined as follows:

Given two MBRs $M_1 = (a, b)$ and $M_2 = (c, d)$, in the d-dimensional Euclidean space,

$M_1 = (a, b)$, where $a = (a_1, a_2, \ldots, a_d)$ and $b = (b_1, b_2, \ldots, b_d)$ such that $a_i \leq b_i$ $1 \leq i \leq d$

$M_2 = (c, d)$, where $c = (c_1, c_2, \ldots, c_d)$ and $d = (d_1, d_2, \ldots, d_d)$ such that $a_i \leq b_i$ $1 \leq i \leq d$



a Two points sets (P and Q)

b The R-tree of point set P

Closest-Pair Query. Figure 1. Example of an R-tree and a point CP query.

the MBR-based distance functions are defined as follows:

$$MINMINDIST(M_1, M_2) = \sqrt{\sum_{i=1}^{d} \begin{cases} (c_i - b_i)^2, & c_i > b_i \\ (a_i - d_i)^2, & a_i > d_i \\ 0, & otherwise \end{cases}}$$

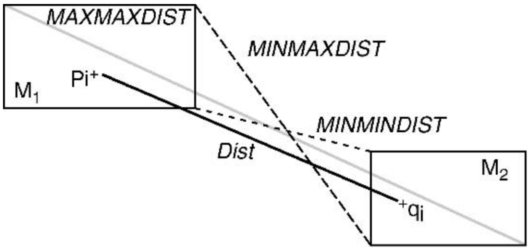
$$MAXMAXDIST(M_1, M_2) = \sqrt{\sum_{i=1}^{d} \begin{cases} (d_i - a_i)^2, & c_i > b_i \\ (b_i - c_i)^2, & a_i > d_i \\ \max\{(d_i - a_i)^2, (b_i - c_i)^2\}, & otherwise \end{cases}} .$$

$$MINMAXDIST(M_1, M_2) = \sqrt{\min_{1 \le j \le d} \left\{ x_j^2 + \sum_{i=1, i \ne j}^{d} y_i^2 \right\}}$$

where

$$x_j = \min\{|a_j - c_j|, |a_j - d_j|, |b_j - c_j|, |b_j - d_j|\} \text{and}$$
$$y_i = \max\{|a_i - d_i|, |b_i - c_i|\}$$

To illustrate the distance functions MINMINDIST, MINMAXDIST and MAXMAXDIST which are the basis of query algorithms for CPQ, in Fig. 2, two MBRs and their MBR-based distance functions and their relation with the distance (dist) between two points (p_i, q_j) are depicted in 2-dimensional Euclidean space.



Closest-Pair Query. Figure 2. MBR-based distance functions in 2-dimensional Euclidean space.

According to [5], MINMINDIST(M_1,M_2) is monotonically non-decreasing with the R-tree heights. MINMINDIST(M_1,M_2) and MAXMAXDIST(M_1, M_2) serve respectively as lower and upper bounding functions of the Euclidean distance from the k closest pairs of spatial objects within the MBRs M_1, and M_2. In the same sense, MINMAXDIST(M_1, M_2) serves as an upper bounding function of the Euclidean distance from the closest pair of spatial objects enclosed by the MBRs M_1 and M_2. As long as the distance functions are consistent, the branch-bound algorithms based on them will work correctly [5].

Moreover, the general pruning mechanism for k-CP queries over R-tree nodes using branch-and-bound algorithms is the following: *if MINMINDIST (M_1, M_2) > z, then the pair of MBRs (M_1, M_2) will be discarded*, where *z* is the distance value of the k-th closest pair that has been found so far (during the processing of the algorithm), or the distance value of the k-th largest MAXMAXDIST found so far (*z* is also called as the pruning distance).

Branch-and-bound algorithms can be designed following DF or BF traversal ordering (Breadth-First traversal order (level-by-level) can also be implemented, but the processing of each level must follow a BF order) to report k closest pairs in non-incremental way (for incremental processing the ordering of traversal must be BF [8]).

As an example, Fig. 3 shows the BF k-CPQ algorithm for two R-trees, for the non-incremental processing version. This algorithm needs to keep a minimum binary heap (H) with the references to pairs of internal nodes (characterized by their MBRs) accessed so far from the two different R-trees and their minimum distance (<MINMINDIST, $\mathrm{Addr}_{\mathrm{MPi}}$, $\mathrm{Addr}_{\mathrm{MQj}}$>). It visits the pair of MBRs (nodes) with the minimum MINMINDIST in H, until it becomes empty or the MINMINDIST value of the pair of MBRs located in the root of H is larger than the distance value of the k-th closest pair that has been found so far (z). To keep track of z, an additional data structure that stores the k closest pairs discovered during the processing of the algorithm is needed. This data structure is organized as a maximum binary heap (k-heap) and holds pairs of objects according to their minimum distance (the pair with the largest distance resides in the root). In the implementation of k-CPQ algorithm, the following cases must be considered: (i) initially the k-heap is empty (z is initialized to ∞), (ii) the pairs of objects reached at the leaf level are inserted in the k-heap until it gets full (z keeps the value of ∞), (iii) if the distance of a new pair of objects discovered at the leaf level is smaller than the distance of the pair residing in the k-heap root, then the root is extracted and the new pair is inserted in the k-heap, updating this data structure and z (distance of the pair of objects residing in the k-heap root).

Several optimizations had been proposed in order to improve performance, mainly with respect to the CPU cost. For instance, a method for selecting the sweep axis and direction for the plane sweep technique has been proposed [12]. But the most important optimization is the use of the plane-sweep technique for k-CPQ [5,12], which is a common technique for computing intersections. The basic idea is to move a sweep-line perpendicular to one of the dimensions, so-called the sweeping dimension, from left to right. This technique is applied for restricting all possible combinations of pairs of MBRs from two R-tree nodes from R_P

```
Create and initialize H, k-heap, z = ∞;
for each pair of MBRs <$M_{Pi}$, $M_{Qj}$> in <$root_{RP}$, $root_{RQ}$> do
      H.insert(MINMINDIST($M_{Pi}$, $M_{Qj}$), $Addr_{MPi}$, $Addr_{MQj}$);
enddo
While (not H.isEmpty()) and (H.MinimumDistance() ≤ z) do
      minimum = H.deleteMinimum();
      $node_{RP}$ = RP.readNode(minimum.$Addr_{MP}$);
      $node_{RQ}$ = RQ.readNode(minimum.$Addr_{MQ}$);
      if ($node_{RP}$ and $node_{RQ}$ are internal nodes) then
           for each pair of MBRs <$M_{Pi}$, $M_{Qj}$> in < $node_{RP}$, $node_{RQ}$> do
                if (MINMINDIST($M_{Pi}$, $M_{Qj}$) ≤ z) then
                  H.insert(MINMINDIST($M_{Pi}$, $M_{Qj}$), $Addr_{MPi}$, $Addr_{MQj}$);
                endif
           enddo
      else
         for each pair of points <$p_i$, $q_j$> in <$node_{RP}$, $node_{RQ}$> do
           distance = dist($p_i$, $q_j$);
           if (k-heap.isFull()) then
              if (distance ≤ z) then
                 k-heap.delete Maximum();
                 k-heap.insert(distance, $p_i$, $q_j$);
                 z = k-heap.get Maximum();
              endif
           else
              k-heap.insert(distance, $p_i$, $q_j$);
           endif
         enddo
      endif
endod
Destroy H;
```

Closest-Pair Query. Figure 3. Best-First k-CPQ Algorithm using R–trees.

and R_Q. If this technique is not used, then a set with all possible combinations of pairs of MBRs from two R-tree nodes must be created. In general, the technique consists in sorting the MBRs of the two current R-tree nodes, based on the coordinates of one of the lower left corners of the MBRs in increasing order. Each MBR encountered during a plane sweep is selected as a pivot, and it is paired up with the non-processed MBRs in the other R-tree node from left to right. The pairs of MBRs with MINMINDIST on the sweeping dimension that are less than or equal to z (pruning distance) are selected for processing. After all possible pairs of MBRs that contain the pivot have been found, the pivot is updated with the MBR of the next smallest value of a lower left corner of MBRs on the sweeping dimension, and the process is repeated. In summary, the application of this technique can be viewed as a *sliding window* on the sweeping dimension with a width of z starting in the lower end of the pivot MBR, where all possible pairs of MBRs that can be formed using the MBR of the pivot and the other MBRs from the remainder entries of the other R-tree node that fall into the current sliding window are chosen. For example, in Fig. 4, a set of MBRs from two R-tree nodes ({M_{P1}, M_{P2}, M_{P3}, M_{P4}, M_{P5}, M_{P6}} and {M_{Q1}, M_{Q2}, M_{Q3}, M_{Q4}, M_{Q5}, M_{Q6}, M_{Q7}}) is shown. Without plane-sweep, 6*7 = 42 pairs of MBRs must be generated. If the plane-sweep technique is applied over the X axis (sweeping dimension) and taking into account the distance value of z (pruning distance), this number of possible pairs will reduced considerably (the number of selected pairs of MBRs using the plane sweep technique is only 29).

Key Applications

Geographical Information Systems

Closest pair is a common distance-based query in the spatial database context, and it has only recently received special attention. Efficient algorithms are important for dealing with the large amount of spatial data in several GIS applications. For example, k-CPQ can discover the K closest pairs of cities and cultural landmarks providing an increase order based on its distances.

Data Analysis

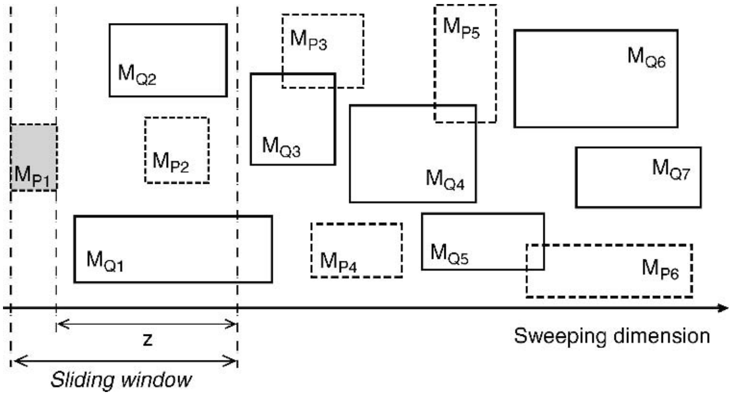
Closest pair queries have been considered as a core module of clustering. For example, a proposed clustering algorithm [10] owes its efficiency to the use of closest pair query, as opposed to previous quadratic-cost approaches.

Decision Making

A number of decision support tasks can be modeled as closest pairs query. For instance, find the top k factory-house pairs ordered by the closeness to one another. This gives us a measure of the effect of individual factory on individual household, and can give workers a priority to which factory to address first.

Future Directions

k-closest pair query is a useful type of query in many practical applications involving spatial data, and the traditional technique to handle this spatial query generally assumes that the objects are static. Objects represented as a function of time have been studied in other domains, as in spatial semijoin [9]. For this reason,



Closest-Pair Query. Figure 4. Using plane-sweep technique over MBRs from two R-tree nodes.

closest pair query in spatio-temporal databases could be an interesting line of research.

Another interesting problem to study is the monitoring of k-closest pairs over moving objects. It aims at maintaining closest pairs results while the underlying objects change the positions [15]. For example, return k pairs of taxi stands and taxies that have the smallest distances.

Other interesting topics to consider (from the static point of view) are to study k-CPQ between different spatial data structures (Linear Region Quadtrees for raster and R-trees for vector data), and to investigate k-CPQ in non-Euclidean spaces (e.g., road networks).

Experimental Results

In general, for every presented method, there is an accompanying experimental evaluation in the corresponding reference. [4,5,8] compare BF and DF traversal order for conventional k-CPQ (from the incremental and non-incremental point of view). In [7], a cost model for k-CPQ using R-trees was proposed, evaluating their accuracy. Moreover, experimental results on k-closest pair queries to support the fact that b-Rdnn tree is a better alternative with respect to the R*-trees, when there is high overlap between the two datasets, were presented in [14].

Data Sets

A large collection of real datasets, commonly used for experiments, can be found at: http://www.rtreeportal.org/

URL to Code

R-tree portal (see above) contains the code for most common spatial access methods (mainly R-tree and variations), as well as data generators and several useful links for researchers and practitioners in spatial databases.

The sources in C + + of k-CPQ are in: http://www.ual.es/~acorral/DescripcionTesis.htm

Cross-references

▶ Multi-Step Query Processing
▶ Nearest Neighbor Query
▶ R-Tree (and family)
▶ Spatial Indexing Techniques
▶ Spatial Join

Recommended Reading

1. Beckmann N., Kriegel H.P., Schneider R., and Seeger B. The R*-tree: an efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 322–331.
2. Böhm C. and Krebs F. The k-nearest neighbour join: Turbo charging the KDD process. Knowl. Inform. Syst., 6(6):728–749, 2004.
3. Chan E.P.F. Buffer queries. IEEE Trans. Knowl. Data Eng., 15(4):895–910, 2003.
4. Corral A., Manolopoulos Y., Theodoridis Y., and Vassilakopoulos M. Closest pair queries in spatial databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 189–200.
5. Corral A., Manolopoulos Y., Theodoridis Y., and Vassilakopoulos M. Algorithms for processing K-closest-pair queries in spatial databases. Data Knowl. Eng., 49(1):67–104, 2004.
6. Corral A., Manolopoulos Y., Theodoridis Y., and Vassilakopoulos M. Multi-way distance join queries in spatial databases. GeoInformatica, 8(4):373–402, 2004.
7. Corral A., Manolopoulos Y., Theodoridis Y., and Vassilakopoulos M. Cost models for distance joins queries using R-trees. Data Knowl. Eng., 57(1):1–36, 2006.
8. Hjaltason G.R. and Samet H. Incremental distance join algorithms for spatial databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 237–248.
9. Iwerks G.S., Samet H., and Smith K. Maintenance of spatial semijoin queries on moving points. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 828–839.
10. Nanopoulos A., Theodoridis Y., and Manolopoulos Y. C^2P: clustering based on closest pairs. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 331–340.
11. Papadopoulos A.N., Nanopoulos A., and Manolopoulos Y. Processing distance join queries with constraints. Comput. J., 49(3):281–296, 2006.
12. Shin H., Moon B., and Lee S. Adaptive multi-stage distance join processing. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 343–354.
13. Shou Y., Mamoulis N., Cao H., Papadias D., and Cheung D.W. Evaluation of iceberg distance joins. In Proc. 8th Int. Symp. Advances in Spatial and Temporal Databases, 2003, pp. 270–288.
14. Yang C. and Lin K. An index structure for improving closest pairs and related join queries in spatial databases. In Proc. Int. Conf. on Database Eng. and Applications, 2002, pp. 140–149.
15. Zhu M., Lee D.L., and Zhang J. k-closest pair query monitoring over moving objects. In Proc. 3rd Int. Conf. on Mobile Data Management, 2002, pp. 14–14.

Cloud Computing

▶ Replication in Multi-Tier Architectures
▶ Storage Grid

Cluster and Distance Measure

Dimitrios Gunopulos[1,2]
[1]Computer Science and Eng. Dept., Univ. of California Riverside, Riverside, CA 92521, USA
[2]Dept. of Informatics and Telecommunications, University of Athens, Athens, Greece

Synonyms

Unsupervised learning; Segmentation

Definition

Clustering

Clustering is the assignment of objects to groups of similar objects (clusters). The objects are typically described as vectors of features (also called attributes). So if one has *n* attributes, object **x** is described as a vector $(x_1,..,x_n)$. Attributes can be numerical (scalar) or categorical. The assignment can be hard, where each object belongs to one cluster, or fuzzy, where an object can belong to several clusters with a probability. The clusters can be overlapping, though typically they are disjoint. Fundamental in the clustering process is the use of a distance measure.

Distance Measure

In the clustering setting, a distance (or equivalently a similarity) measure is a function that quantifies the similarity between two objects.

Key Points

The choice of a distance measure depends on the nature of the data, and the expected outcome of the clustering process. The most important consideration is the type of the features of the objects. One first focuses on distance measures when the features are all numerical. This includes features with continuous values (real numbers) or discrete values (integers). In this case, typical choices include:

1. The L_p norm. It is defined as $D(\mathbf{x},\mathbf{y}) = \left(\sum_{1\leq i\leq n}(X_1 - Y_1)^p\right)^{1/p}$. Typically p is 2 (the intuitive and therefore widely used Euclidean distance), or 1 (the Manhattan or city block distance), or infinity (the Maximum distance).
2. The Mahalanobis distance. It is defined as $D(\mathbf{x},\mathbf{y}) = (\mathbf{x}-\mathbf{y})\sum^{-1}(\mathbf{x}-\mathbf{y})^T$which generalizes the Euclidean and allows the assignment of different weights to different features.
3. The angle between two vectors, computed using the inner product of two vectors $\mathbf{x}\cdot\mathbf{y}$.
4. The Hamming distance, which measures the number of disagreements between two binary vectors.

In different settings different distance measures can be used. The edit, or Levenshtein, distance, is an extension of the Hamming distance, and is typically used for measuring the distance between two strings of characters. The edit distance is defined as the minimum number of insertions, deletions or substitutions that it takes to transform one sting to another.

When two time series are compared, the Dynamic Time Warping distance measure is often used to quantify their distance. The length of the Longest Common Subsequence (LCSS) of two time series is also frequently used to provide a similarity measure between the time series. LCSS is a similarity measure because the longest common subsequence becomes longer when two time series are more similar. To create a distance measure, LCSS is typically normalized by dividing by the length of the longest of the two sequences, and then subtracting the ratio from one.

Finally, when sets of objects are compared, the Jaccard coefficient is typically used to compute their distance. The Jaccard coefficient of sets A and B is defined as $J(A,B) = |A \cap B| / |A \cup B|$, that is, the fraction of the common elements over the union of the two sets.

The majority of the distance measures used in practice, and indeed most of the ones described above are metrics. Formally, a distance measure D is a metric if it obeys the following properties:

For objects A, B, (i) D(A,B)\geq 0, (ii) D(A,B) = 0 if and only if A = B, and (iii) D(A,B) = D(B,A), and (iv) for any objects A,B,C, D(A,B) + D(B,C)\geq D(A,C) (triangle inequality).

Most distance measures can be trivially shown to observe the first three properties, but do not necessarily observe the triangle inequality. For example, the constrained Dynamic Time Warping distance, a typically used measure to compute the similarity between time series which does not allow arbitrary stretching of a time series, is not a metric because it does not satisfy the triangle inequality. Experimental results have shown that the constrained Dynamic Time Warping distance performs at least as good as the unconstrained one and it is also faster to compute, thus justifying its use although it is not a metric. Note however that, if it is so required, any distance measure can be converted into a metric by taking the shortest path between objects A and B in the complete graph where each object is a node and each edge is weighted by the distance between the two nodes.

Cross-references

▶ Clustering Overview and Applications
▶ Data Mining

Recommended Reading

1. Everitt B.S., Landau S., Leese M. Cluster Analysis. Wiley, 2001.
2. Jain A.K., Murty M.N., and Flyn P.J. Data Clustering: A Review. ACM Comput Surv, 31(3):1999.
3. Theodoridis S. and Koutroubas K. Pattern recognition. Academic Press, 1999.

Cluster Database Replication

▶ Replica Control

Cluster Databases

▶ Process Structure of a DBMS

Cluster Replication

▶ Replication for Scalability
▶ Replication in Multi-Tier Architectures

Cluster Stability

▶ Clustering Validity

Cluster Validation

▶ Clustering Validity

Clustering

▶ Deduplication in Data Cleaning
▶ Physical Database Design for Relational Databases

Clustering for Post Hoc Information Retrieval

DIETMAR WOLFRAM
University of Wisconsin-Milwaukee, Milwaukee, WI, USA

Synonyms

Document clustering

Definition

Clustering is a technique that allows similar objects to be grouped together based on common attributes. It has been used in information retrieval for different retrieval process tasks and objects of interest (e.g., documents, authors, index terms). Attributes used for clustering may include assigned terms within documents and their co-occurrences, the documents themselves if the focus is on index terms, or linkages (e.g., hypertext links of Web documents, citations or co-citations within documents, documents accessed). Clustering in IR facilitates browsing and assessment of retrieved documents for relevance and may reveal unexpected relationships among the clustered objects.

Historical Background

A fundamental challenge of information retrieval (IR) that continues today is how to best match user queries with documents in a queried collection. Many mathematical models have been developed over the years to facilitate the matching process. The details of the matching process are usually hidden from the user, who is only presented with an outcome. Once a set of candidate documents has been identified, they are presented to the user for perusal. Traditional approaches have relied on ordered linear lists of documents based on calculated relevance or another sequencing criterion (e.g., date, alphabetical by title or author). The resulting linear list addresses the assessed relationship of documents to queries, but not the relationships of the documents themselves. Clustering techniques can reduce this limitation by creating groups of documents (or other objects of interest) to facilitate more efficient retrieval or perusal and evaluation of retrieved sets.

The application of clustering techniques to IR extends back to some of the earliest experimental IR systems including Gerard Salton's SMART system, which relied on document cluster identification within a vector space as a means of quickly identifying sets of relevant documents. The rationale for applying clustering was formalized as the “cluster hypothesis,” proposed by Jardine and van Rijsbergen [6]. This hypothesis proposes that documents that are relevant to a query are more similar to each other than to documents that are not relevant to the query. The manifestation of this relationship can be represented in different ways by grouping like documents or, more recently, visualizing the relationships and resulting proximities in a multidimensional space.

Early applications of clustering emphasized its use to more efficiently identify groups of related, relevant documents and to improve search techniques. The computational burden associated with real-time cluster identification during searches on increasingly larger data corpora and the resulting lackluster performance improvements have caused clustering to lose favor as a primary mechanism for retrieval. However, clustering methods continue to be studied and used today (see, for example, [7]). Much of the recent research into clustering for information retrieval has focused on other areas that support the retrieval process. For instance, clustering has been used to assist in query expansion, where additional terms for retrieval may be identified. Clustering of similar terms can be used to construct thesauri, which can be used to index documents [3].

Recent research on clustering has highlighted its benefits for post hoc retrieval tasks, in particular for the presentation of search results to better model user and usage behavior. The focus of applications presented here is on these post hoc IR tasks, dealing with effective representation of groups of objects once identified to support exploratory browsing and to provide a greater understanding of users and system usage for future IR system development.

Foundations

Methods used to identify clusters are based on cluster analysis, a multivariate exploratory statistical technique. Cluster analysis relies on similarities or differences in object attributes and their values. The granularity of the analysis and the validity of the resulting groups are dependent on the range of attributes and values associated with objects of interest. For IR applications, clusters are based on common occurrences and weights of assigned terms for documents, the use of query terms, or linkages between objects of interest represented as hypertext linkages or citations/co-citations.

Clustering techniques can be divided into hierarchical and non-hierarchical approaches. Non-hierarchical clustering methods require that a priori assumptions be made about the nature and number of clusters, but can be useful if specific cluster parameters are sought. Hierarchical clustering, which is more commonly used, begins with many small groups of objects that serve as initial clusters. Existing groups are clustered into larger groups until only one cluster remains. Visually, the structure and relationship of clusters may be represented as a dendrogram, with different cluster agglomerations at different levels on the dendrogram representing the strength of relationship between clusters. Other visualization techniques may be applied and are covered elsewhere. In hierarchical methods, the shorter the agglomerative distance, the closer the relationship and the more similar the clusters are. As an exploratory technique, there is no universally accepted algorithm to conduct the analysis, but the general steps for conducting the analysis are similar. First, a similarity measure is applied to the object attributes, which serves as the basis for pairwise comparisons. Standard similarity or distance measures applied in IR research such as the Euclidean distance, cosine measure, Jaccard coefficient, and Dice coefficient can be used. Next, a method for cluster determination is selected. Common methods include: single complete linkage, average linkage, nearest neighbor, furthest neighbor, centroid clustering (representing the average characteristics of objects within a cluster), and Ward’s method. Each method uses a different algorithm to assess cluster membership and may be found to be more appropriate in given circumstances. Outcomes can vary significantly depending on the method used. This flexibility underscores one of the challenges for effectively implementing cluster analysis. With no one correct or accepted way to conduct the analysis, outcomes are open to interpretation, but may be viewed as equally valid. For example, single linkage clustering, which links pairs of objects that most closely resemble one another, is comparatively simple to implement and has been widely used, but can result in lengthy linear chains of clusters. Parameters may need to be specified that dictate the minimum size of clusters to avoid situations where there are large orders of difference in cluster membership. Another challenge inherent in clustering is that different clustering algorithms can produce similar numbers of clusters, but if some clusters contain few members, this does little to disambiguate the members within large clusters. The number of clusters that partition the object set can be variable in hierarchical clustering. More clusters result in fewer objects per cluster with greater inter-object similarity, but with potentially more groups to assess. It is possible to test for an optimal number of clusters using various measures that calculate how differing numbers of clusters affect cluster cohesiveness.

Clustering may be implemented in dynamic environments by referencing routines based on specific clustering algorithms developed by researchers or through specialty clustering packages. Details on clustering algorithms for information retrieval can be found in Rasmussen [8]. Standard statistical and mathematical software packages such as SAS and SPSS also support a range of clustering algorithms. Special algorithms may need to be applied to very large datasets to reduce computational overhead, which can be substantial for some algorithms.

Key Applications

In addition to early applications of clustering for improving retrieval efficiency, clustering techniques in IR have included retrieval results presentation, and modeling of IR user and usage characteristics based on transactions logs. Although largely a topic of research interest, some applications have found their way into commercial systems. Clustering of search results has been applied by several Web-based search services since the late 1990s, some of which are no longer available. Most notable of the current generation is Clusty (clusty.com), which organizes retrieval results from several search services around topical themes.

The application of clustering to support interactive browsing has been an active area of investigation in recent years. Among the earliest demonstrations for this purpose was the Scatter/Gather method outlined by Cutting et al. [4], in which the authors demonstrated how clustering of retrieved items can facilitate browsing for vaguely defined information needs. This approach was developed to serve as a complement to more focused techniques for retrieval assessment. In application, the method presents users with a set of clusters that serves as the starting point for browsing. The user selects the clusters of greatest interest. The contents of those clusters are then gathered into a single cluster, which now serves as the corpus for a new round of clustering, into which the new smaller corpus of items is scattered. The process continues until the user's information need is met or the user abandons the search. To support real time clustering of datasets, the authors developed an efficient clustering algorithm, called buckshot, plus a more accurate algorithm, called fractionation, to permit more detailed clustering in offline environments where a timely response is less critical. Another algorithm, called cluster digest, was used to encapsulate the topicality of a given cluster based on the highest weighted terms within the cluster. Hearst and Pedersen [5] evaluated the efficacy of Scatter/Gather on the top-ranked retrieval outcomes of a large dataset, and tested the validity of the cluster hypothesis. The authors compared the number of known relevant items to those appearing in the generated clusters. A user study was also conducted, which demonstrated that participants were able to effectively navigate and interact with the system incorporating Scatter/Gather.

Increasingly, IR systems provide access to heterogeneous collections of documents. The question arises whether the cluster hypothesis, and the benefits of capitalizing on its attributes, extends to the distributed IR environment, where additional challenges include the merger of different representations of documents and identification of multiple occurrences of documents across the federated datasets. Crestani and Wu [2] conducted an experimental study to determine whether the cluster hypothesis holds in a distributed environment. They simulated a distributed environment by using different combinations of retrieval environments and document representation heterogeneity, with the most sophisticated implementation representing three different IR environments with three different collections. Results of the different collections and systems were clustered and compared. The authors concluded that the cluster hypothesis largely holds true in distributed environments, but fails when brief surrogates of full text documents are used.

With the growing availability of large IR system transaction logs, clustering methods have been used to identify user and usage patterns. By better understanding patterns in usage behavior, IR systems may be able to identify types of behaviors and accommodate those behaviors through context-sensitive assistance or through integration of system features that accommodate identified behaviors. Chen and Cooper [1] relied on a rich dataset of user sessions collected from the University of California MELVYL online public access catalog system. Based on 47 variables associated with each user session (e.g., session length in seconds, average number of items retrieved, average number of search modifications), their analysis identified six clusters representing different types of user behaviors during search sessions. These included help-intensive searching, knowledgeable usage, and known-item searching. Similarly, Wen et al. [9] focused on clustering of user queries in an online encyclopedia environment to determine whether queries could be effectively clustered to direct users to appropriate frequently asked questions topics. IR environments that cater to a broad range of users are well-known for short query submissions by users, which make clustering applications based solely on query term co-occurrence unreliable. In addition to the query content, the authors based their analysis on common retrieved documents viewed by users. By combining query content with common document selections, a link was established between queries that might not share search terms. The authors demonstrated how the application of their clustering method, which was reportedly adopted by the encyclopedia studied, could effectively guide users to appropriate frequently asked questions.

The previous examples represent only a sample of clustering applications in an IR context. Additional recent research developments and applications using clustering may be found in Wu et al. [10].

Cross-references

- ▶ Data Mining
- ▶ Text Mining
- ▶ Visualization for Information Retrieval

Recommended Reading

1. Chen H.M. and Cooper M.D. Using clustering techniques to detect usage patterns in a web-based information system. J. Am. Soc. Inf. Sci. Technol., 52(11):888–904, 2001.
2. Crestani F. and Wu S. Testing the cluster hypothesis in distributed information retrieval. Inf. Process. Manage., 42(5):1137–1150, 2006.
3. Crouch C.J. A cluster-based approach to thesaurus construction. In Proc. 11th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1988, pp. 309–320.
4. Cutting D.R., Karger D.R., Pedersen J.O., and Tukey J.W. Scatter/Gather: a cluster-based approach to browsing large document collections. In Proc. 15th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1992, pp. 318–329.
5. Hearst M.A. and Pedersen J.O. Reexamining the cluster hypothesis: Scatter/Gather on retrieval results. In Proc. 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1996, pp. 76–84.
6. Jardine N. and van Rijsbergen C. The use of hierarchic clustering in information retrieval. Inf. Storage Retr., 7(5):217–240, 1971.
7. Liu X. and Croft W.B. Cluster-based retrieval using language models. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 186–193.
8. Rasmussen E. Clustering algorithms. In Information Retrieval Data Structures & Algorithms, W.B. Frakes, R. Baeza-Yates (eds.). Prentice Hall, Englewood Cliffs, NJ, 1992, pp. 419–442.
9. Wen J.R., Nie J.Y., and Zhang H.J. Query clustering using user logs. ACM Trans. Inf. Syst., 20(1):59–81, 2002.
10. Wu W., Xiong H., and Shekhar S. (eds.) Clustering and Information Retrieval. Kluwer, Norwell, MA, 2004.

Clustering Index

▶ Primary Index

Clustering on Streams

Suresh Venkatasubramanian
University of Utah, Salt Lake City, UT, USA

Definition

An instance of a clustering problem (see clustering) consists of a collection of points in a distance space, a measure of the *cost* of a clustering, and a measure of the *size* of a clustering. The goal is to compute a partitioning of the points into clusters such that the cost of this clustering is minimized, while the size is kept under some predefined threshold. Less commonly, a threshold for the cost is specified, while the goal is to minimize the size of the clustering.

A data stream (see data streams) is a sequence of data presented to an algorithm one item at a time. A stream algorithm, upon reading an item, must perform some action based on this item and the contents of its working space, which is sublinear in the size of the data sequence. After this action is performed (which might include copying the item to its working space), the item is discarded.

Clustering on streams refers to the problem of clustering a data set presented as a data stream.

Historical Background

Clustering (see clustering) algorithms typically require access to the entire data to produce an effective clustering. This is a problem for large data sets, where random access to the data, or repeated access to the entire data set, is a costly operation. For example, the well-known k-means heuristic is an iterative procedure that in each iteration must read the entire data set twice. One set of approaches to performing clustering on large data involves sampling: a small subset of data is extracted from the input and clustered, and then this clustering is extrapolated to the entire data set.

The *data stream* paradigm [14] came about in two ways: first, as a way to model access to large streaming sources (network traffic, satellite imagery) that by virtue of their sheer volume, cannot be archived for offline processing and need to be aggregated, summarized and then discarded in real time. Second, the streaming paradigm has shown itself to be the most effective way of accessing large databases: Google's Map Reduce [9] computational framework is one example of the efficacy of stream processing.

Designing clustering algorithms for stream data requires different algorithmic ideas than those useful for traditional clustering algorithms. The online computational paradigm [4] is a potential solution: in this paradigm, an algorithm is presented with items one by one, and using only information learned up to the current time, must make a prediction or estimate on the new item being presented. Although the online computing paradigm captures the sequential aspect of stream processing, it does not capture the additional constraint that only a small portion of the history may be stored. In fact, an online algorithm is permitted to use the entirety of the history of the stream, and is usually not limited computationally in any way. Thus, new ideas are needed to perform clustering in a stream setting.

Foundations

Preliminaries

Let X be a domain and d be a distance function defined between pairs of elements in X. Typically, it is assumed that d is a metric (i.e., it satisfies the triangle inequality $d(x, y) + d(y, z) \geq d(x, z) \forall x,y,z \in X$). One of the more common measures of the cost of a cluster is the so-called *median cost*: the cost of a cluster $C \subseteq X$ is the function

$$\mathrm{cost}(C) = \sum_{x \in C} d(x, c^*)$$

where $c^* \in X$, the *cluster center*, is the point that minimizes cost(C). The *k-median problem* is to find a collection of k disjoint clusters, the sum of whose costs is minimized.

An equally important cost function is the *mean cost*: the cost of a cluster $C \subseteq X$ is the function

$$\mathrm{cost}(C) = \sum_{x \in C} d^2(x, c^*)$$

where c^* is defined as before. The *k-means problem* is to find a collection of clusters whose total *mean cost* is minimized. It is useful to note that the median cost is more robust to outliers in the data; however, the mean cost function, especially for points in Euclidean spaces, yields a very simple definition for c^*: it is merely the centroid of the set of points in the cluster. Other measures that are often considered are the k-center cost, where the goal is to minimize the maximum radius of a cluster, and the diameter cost, where the goal is to minimize the maximum diameter of a cluster (note that the diameter measure does not require one to define a cluster center).

A data stream problem consists of a sequence of items $x_1, x_2,\ldots,x_n$, and a function $f(x_1,\ldots,x_n)$ that one wishes to compute. The limitation here is that the algorithm is only permitted to store a *sublinear* number of items in memory, because n is typically too large for all the items to fit in memory. Further, even random access to the data is prohibitive, and so the algorithm is limited to accessing the data in sequential order.

Since most standard clustering problems (including the ones described above) are NP-hard in general, one cannot expect solutions that minimize the cost of a clustering. However, one can often show that an algorithm comes close to being optimal: formally, one can show that the cost achieved by an algorithm is within some multiplicative factor c of the optimal solution. Such an algorithm is said to be a c-approximation algorithm. Many of the methods presented here will provide such guarantees on the quality of their output. As usual, one should keep in mind that these guarantees are *worst-case*, and thus apply to any possible input the algorithm may encounter. In practice, these algorithms will often perform far better than promised.

General Principles

Stream clustering is a relatively new topic within the larger area of stream algorithms and data analysis. However, there are some general techniques that have proven their usefulness both theoretically as well as practically, and are good starting points for the design and analysis of stream clustering methods. This section reviews these ideas, as well as pointing to examples of how they have been used in various settings.

Incremental Clustering The simplest way to think about a clustering algorithm on stream data is to imagine the stream data arriving in chunks of elements. Prior to the arrival of the current chunk, the clustering algorithm has computed a set of clusters for all the data seen so far. Upon encountering the new chunk, the algorithm must update the clusters, possibly expanding some and contracting others, merging some clusters and splitting others. It then requests the next chunk, discarding the current one. Thus, a core component of any stream clustering algorithm is a routine to incrementally update a clustering when new data arrives. Such an approach was developed by Charikar et al. [6] for maintaining clusterings of data in a metric space using a diameter cost function. Although their scheme was phrased in terms of incremental clusterings, rather than stream clusterings, their approach generalizes well to streams. They show that their scheme yields a provable approximation to the optimal diameter of a k-clustering.

Representations One of the problems with clustering data streams is choosing a representation for a cluster. At the very least, any stream clustering algorithm stores the location of a cluster center, and possibly the number of items currently associated with this cluster. This representation can be viewed as a *weighted point*, and can be treated as a single point in further iterations of the clustering process. However, this representation loses information about the geometric size and distribution of a cluster. Thus, another standard representation of a cluster consists of the center and the number of points augmented with the sum of squared distances from the points in the cluster to the center. This last term informally measures the variation of points within a cluster, and when viewed in the context of density estimation via Gaussians, is in fact the sample variance of the cluster.

Clusters reduced in this way can be treated as weighted points (or weighted balls), and clustering algorithms should be able to handle such generalized points. One notable example of the use of such a representation is the one-pass clustering algorithm of Bradley et al. [5], which was simplified and improved by Farnstrom et al. [11]. Built around the well known k-means algorithm (that iteratively seeks to minimize the k-means measure described above), this technique proceeds as follows.

Algorithm 1: Clustering with representations

```
Initialize cluster centers randomly
**While** chunk of data remains to be read **do**
    Read a chunk of data (as much as will fit in memory),
    and cluster it using the k-means algorithm.
    For each cluster, divide the points contained within it
    into the core (points that are very close to the center
    under various measures), and the periphery.
    Replace the set of points in the core by a summary as
    described above. Discard all remaining points.
    Use the current cluster list as the set of centers for the
    next chunk.
```

It is important that representations be *linear*. Specifically, given two chunks of data c,c', and their representations r,r', it should be the case that the representation of $c \cup c'$ be formed from a linear combination of r and r'. This relates to the idea of *sketching* in stream algorithms, and is important because it allows the clustering algorithm to work in the (reduced) space of representations, rather than in the original space of data. Representations like the one described above are linear, and this is a crucial factor in the effectiveness of these algorithms.

Hierarchical Clustering Viewing a cluster as a weighted point in a new clustering problem quickly leads to the idea of *hierarchical clustering*: by thinking of a point as a single-element cluster, and connecting a cluster and its elements in a parent-child relationship, a clustering algorithm can represent multiple levels of merges as a tree of clusters, with the root node being a single cluster containing all the data, and each leaf being a single item. Such a tree is called a Hierarchical Agglomerative Clustering (HAC), since it can be viewed bottom-up as a series of agglomerations. Building such a hierarchy yields more general information about the relationship between clusters, and the ability to make better judgments about how to merge clusters.

The well-known clustering algorithm BIRCH [15] makes use of a hierarchy of cluster representations to cluster a large database in a few passes. In a first pass, a tree called the CF-tree is constructed, where each internal node represents a cluster of clusters, and each leaf represents a cluster of items. This tree is controlled by two parameters: B, the branching factor, and T, a diameter threshold that limits the size of leaf clusters.

In further passes, more analysis is performed on the CF-tree to compress clusters further. The tree is built much in the way a B+-tree is built: new items are inserted in the deepest cluster possible, and if the threshold constraint is violated, the cluster is split, and updates are propagated up the tree.

BIRCH is one of the best-known large-data clustering algorithms, and is generally viewed as a benchmark to compare other clustering algorithms against. However, BIRCH does not provide formal guarantees on the quality of the clusterings thus produced. The first algorithm that computes a hierarchical clustering on a stream while providing formal performance guarantees is a method for solving the k-median problem developed by Guha et al. [12,13]. This algorithm is best described by first presenting it in a non-streaming context:

Algorithm 2: Small space

```
Divide the input into $l$ disjoint parts.
Cluster each part into $O(k)$ clusters. Assign each point to its nearest cluster center.
cluster the $O(lk)$ cluster centers, where each center is weighted by the number of points assigned to it.
```

Note that the total space required by this algorithm is $O(\ell k + n/\ell)$. The value of this algorithm is that it propagates good clusterings: specifically, if the intermediate clusterings are computed by algorithms that yield constant-factor approximations to the best clustering (under the k-median cost measure), then the final output will also be a (larger) constant factor approximation to the best clustering. Also note that the final clustering step may itself be replaced by a recursive call to the algorithm, yielding a hierarchical scheme.

Converting this to a stream algorithm is not too difficult. Consider each chunk of data as one of the disjoint parts the input is broken into. Suppose each part is of size m, and there exists a clustering procedure that can cluster these points into $2k$ centers with reasonable accuracy. The algorithm reads enough data to obtain m centers ($m^2/2k$ points). These m "points" can be viewed as the input to a second level streaming process, which performs the same operations. In general, the ith-level stream process takes $m^2/2k$ points from the $(i-1)$th-level stream process and clusters them into m points, which are appended to the stream for the next level.

The guarantees provided by the method rely on having accurate clustering algorithms for the intermediate steps. However, the general paradigm itself is useful as a heuristic: the authors show that using the k-means algorithm as the intermediate clustering step yields reasonable clustering results in practice, even though the method comes with no formal guarantees.

On Relaxing the Number of Clusters If one wishes to obtain guarantees on the quality of a clustering, using at least k clusters is critical; it is easy to design examples where the cost of a $(k-1)$-clustering is much larger than the cost of a k-clustering. One interesting aspect of the above scheme is how it uses weaker clustering algorithms (that output $O(k)$ rather than k clusters) as intermediate steps on the way to computing a k-clustering. In fact, this idea has been shown to be useful in a formal sense: subsequent work by Charikar et al. [7] showed that if one were to use an extremely weak clustering algorithm (in fact, one that produces $O(k \log n)$ clusters), then this output can be fed into a clustering algorithm that produces k clusters, while maintaining overall quality bounds that are better than those described above. This idea is useful especially if one has a fast algorithm that produces a larger number of clusters, and a more expensive algorithm that produces k clusters: the expensive algorithm can be run on the (small) output of the fast algorithm to produce the desired answer.

Clustering Evolving Data

Stream data is often temporal. Typical data analysis questions are therefore often limited to ranges of time ("in the last three days," "over the past week," "for the period between Jan 1 and Feb 1," and so on). All of the above methods for clustering streams assume that the goal is to cluster the entire data stream, and the only constraint is the space needed to store the data. Although they are almost always incremental, in that the stream can be stopped at any time and the resulting clustering will be accurate *for all data seen upto that point*, they cannot correctly output clusterings on *windows* of data, or allow the influence of past data to gradually wane over time. Even with non-temporal data, it may be important to allow the data analysis to operate on a subset of the data to capture the notion of *concept drift* [10], a term that is used to describe a scenario when natural data characteristics change as the stream evolves.

Sliding Windows A popular model of stream analysis is the *sliding window* model, which introduces a new parameter W. The goal of the stream analysis is to produce summary statistics (a clustering, variance estimates or other statistics), on the *most recent W items only*, while using space that is sublinear in W. This model can be thought of as represented by a *sliding window* of length W with one end (the sliding end) anchored to the current element being read. The challenge of dealing with sliding windows is the problem of deletion. Although not as general as a fully dynamic data model where arbitrary elements can be inserted and deleted, the sliding window model introduces with the problem of updating a cluster representation under deletions, and requires new ideas.

One such idea is the *exponential histogram*, first introduced by Datar et al. [8] to estimate certain statistical properties of sliding windows on streams, and used by Babcock et al. [3] to compute an approximate k-median clustering in the sliding window model. The idea here is to maintain a set of buckets that together partition all data in the current window. For each bucket, relevant summary statistics are maintained. Intuitively, the smaller the number of items assigned to a bucket, the more accurate the summary statistics (in the limit, the trivial histogram has one bucket for each of the W items in the window). The larger this number, the fewer the number of buckets needed. Balancing these two conflicting requirements yields a scheme where each bucket stores the items between two timestamps, and the bucket sizes increase exponentially as they store items further in the past. It requires more detailed analysis to demonstrate that such a scheme will provide accurate answers to queries over windows, but the use of such exponentially increasing bucket sizes allows the algorithm to use a few buckets, while still maintaining a reasonable approximation to the desired estimate.

Hierarchies of Windows The sliding window model introduces an extra parameter W whose value must be justified by external considerations. One way of getting around this problem is to maintain statistics for multiple values of W (typically an exponentially increasing family). Another approach, used by Aggarwal et al. [1] is to maintain *snapshots* (summary representations of the clusterings) at time steps at different levels of resolution. For example, a simple two level snapshot scheme might store the cluster representations computed after times t, $t + 1$,...$t + W$, as well as t, $t + 2$, $t + 4$,...$t + 2W$ (eliminating duplicate summaries as necessary). Using the linear structure of representations will allow the algorithm to extract summaries for time intervals: they show that such a scheme uses space efficiently while still being able to detect evolution in data streams at different scales.

Decaying Data For scenarios where such a justification might be elusive, another model of evolving data is the *decay model*, in which one can think of a data item's influence waning (typically exponentially) with time. In other words, the value of the ith item, instead of being fixed at x_i, is a function of time $x_i(t) = x_i(0)\exp(-c(t - i))$. This reduces the problem to the standard setting of computing statistics over the entire stream, while using the decay function to decide which items to remove from the limited local storage when computing statistics. The use of exponentially decaying data is quite common in temporal data analysis: one specific example of its application in the clustering of data streams is the work on HPStream by Aggarwal et al. [2].

Key Applications

Systems that manage large data sets and perform data analysis will require stream clustering methods. Many modern *data cleaning systems* require such tools, as well as large scientific databases. Another application of stream clustering is for network traffic analysis: such algorithms might be situated at routers, operating on packet streams.

Experimental Results

Most of the papers cited above are accompanied by experimental evaluations and comparisons to prior work. BIRCH, as mentioned before, is a common benchmarking tool.

Cross-references

- ▶ Data Clustering
- ▶ Information Retrieval
- ▶ Visualization

Recommended Reading

1. Aggarwal C.C., Han J., Wang J., and Yu P.S. A framework for clustering evolving data streams. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 81–92.
2. Aggarwal C.C., Han J., Wang J., and Yu P.S. A framework for projected clustering of high dimensional data streams. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 852–863.

3. Babcock B., Datar M., Motwani R., and O'Callaghan L. Maintaining variance and k-medians over data stream windows. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2003, pp. 234–243.
4. Borodin A. and El-Yaniv R. Online computation and competitive analysis. Cambridge University Press, New York, NY, USA, 1998.
5. Bradley P.S., Fayyad U.M., and Reina C. Scaling Clustering Algorithms to Large Databases. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998, pp. 9–15.
6. Charikar M., Chekuri C., Feder T., and Motwani R. Incremental Clustering and Dynamic Information Retrieval. SIAM J. Comput., 33(6):1417–1440, 2004.
7. Charikar M., O'Callaghan L., and Panigrahy R. Better streaming algorithms for clustering problems. In Proc. 35th Annual ACM Symp. on Theory of Computing, 2003, pp. 30–39.
8. Datar M., Gionis A., Indyk P., and Motwani R. Maintaining stream statistics over sliding windows: (extended abstract). In Proc. 13th Annual ACM -SIAM Symp. on Discrete Algorithms, 2002, pp. 635–644.
9. Dean J. and Ghemaway S. MapReduce: simplified data processing on large clusters. In Proc. 6th USENIX Symp. on Operating System Design and Implementation, 2004, pp. 137–150.
10. Domingos P. and Hulten G. Mining high-speed data streams. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 71–80.
11. Farnstrom F., Lewis J., and Elkan C. Scalability for clustering algorithms revisited. SIGKDD Explor., 2(1):51–57, 2000.
12. Guha S., Meyerson A., Mishra N., Motwani R., and O'Callaghan L. Clustering Data Streams: Theory and practice. IEEE Trans. Knowl. Data Eng., 15(3):515–528, 2003.
13. Guha S., Mishra N., Motwani R., and O'Callaghan L. Clustering data streams. In Proc. 41st Annual Symp. on Foundations of Computer Science, 2000, p. 359.
14. Muthukrishnan S. Data streams: algorithms and applications. Found. Trend Theor. Comput. Sci., 1(2), 2005.
15. Zhang T., Ramakrishnan R., and Livny M. BIRCH: A New Data Clustering Algorithm and Its Applications. Data Min. Knowl. Discov., 1(2):141–182, 1997.

Clustering Overview and Applications

Dimitrios Gunopulos[1,2]
[1]Computer Science and Eng. Dept., Univ. of California
Riverside, Riverside, CA 92521, USA
[2]Dept. of Informatics and Telecommunications,
University of Athens, Athens, Greece

Synonyms

Unsupervised learning

Definition

Clustering is the assignment of objects to groups of similar objects (clusters). The objects are typically described as vectors of features (also called attributes). Attributes can be numerical (scalar) or categorical. The assignment can be hard, where each object belongs to one cluster, or fuzzy, where an object can belong to several clusters with a probability. The clusters can be overlapping, though typically they are disjoint. A distance measure is a function that quantifies the similarity of two objects.

Historical Background

Clustering is one of the most useful tasks in data analysis. The goal of clustering is to discover groups of similar objects and to identify interesting patterns in the data. Typically, the clustering problem is about partitioning a given data set into groups (clusters) such that the data points in a cluster are more similar to each other than points in different clusters [4,8]. For example, consider a retail database where each record contains items purchased at the same time by a customer. A clustering procedure could group the customers in such a way that customers with similar buying patterns are in the same cluster. Thus, the main concern in the clustering process is to reveal the organization of patterns into "sensible" groups, which allow one to discover similarities and differences, as well as to derive useful conclusions about them. This idea is applicable to many fields, such as life sciences, medical sciences and engineering. Clustering may be found under different names in different contexts, such as unsupervised learning (in pattern recognition), numerical taxonomy (in biology, ecology), typology (in social sciences) and partition (in graph theory) [13].

The clustering problem comes up in so many domains due to the prevalence of large datasets for which labels are not available. In one or two dimensions, humans can perform clustering very effectively visually, however in higher dimensions automated procedures are necessary. The lack of training examples makes it very difficult to evaluate the results of the clustering process. In fact, the clustering process may result in different partitioning of a data set, depending on the specific algorithm, criterion, or choice of parameters used for clustering.

Foundations

The Clustering Process

In the clustering process, there are no predefined classes and no examples that would show what kind of desirable relations should be valid among the data. That is the main difference from the task of classification: Classification is the procedure of assigning an object to a predefined set of categories [FSSU96]. Clustering produces initial categories in which values of a data set are classified during the classification process. For this reason, clustering is described as "unsupervised learning"; in contrast to classification, which is considered as "supervised learning." Typically, the clustering process will include at least the following steps:

1. Feature selection: Typically, the objects or observations to be clustered are described using a set of features. The goal is to appropriately select the features on which clustering is to be performed so as to encode as much information as possible concerning the task of interest. Thus, a pre-processing step may be necessary before using the data.
2. Choice of the clustering algorithm. In this step the user chooses the algorithm that is more appropriate for the data at hand, and therefore is more likely to result to a good clustering scheme. In addition, a similarity (or distance) measure and a clustering criterion are selected in tandem
   – The distance measure is a function that quantifies how "similar" two objects are. In most of the cases, one has to ensure that all selected features contribute equally to the computation of the proximity measure and there are no features that dominate others.
   – The clustering criterion is typically a cost function that the clustering algorithm has to optimize. The choice of clustering criterion has to take into account the type of clusters that are expected to occur.
3. Validation and interpretation of the results. The correctness of the results of the clustering algorithm is verified using appropriate criteria and techniques. Since clustering algorithms define clusters that are not known a priori, irrespective of the clustering methods, the final partition of the data typically requires some kind of evaluation. In many cases, the experts in the application area have to integrate the clustering results with other experimental evidence and analysis in order to draw the right conclusion.

After the third phase the user may elect to use the clustering results obtained, or may start the process from the beginning, perhaps using different clustering algorithms or parameters.

Clustering Algorithms Taxonomy

With clustering being a useful tool in diverse research communities, a multitude of clustering methods has been proposed in the literature. Occasionally similar techniques have been proposed and used in different communities. Clustering algorithms can be classified according to:

1. The type of data input to the algorithm (for example, objects described with numerical features or categorical features) and the choice of similarity function between two objects.
2. The clustering criterion optimized by the algorithm.
3. The theory and fundamental concepts on which clustering analysis techniques are based (e.g., fuzzy theory, statistics).

A broad classification of clustering algorithms is the following [8,14]:

1. *Partitional clustering algorithms*: here the algorithm attempts to directly decompose the data set into a set of (typically) disjoint clusters. More specifically, the algorithm attempts to determine an integer number of partitions that optimize a certain criterion function.
2. *Hierarchical clustering algorithms*: here the algorithm proceeds successively by either merging smaller clusters into larger ones, or by splitting larger clusters. The result of the algorithm is a tree of clusters, called dendrogram, which shows how the clusters are related. By cutting the dendrogram at a desired level, a clustering of the data items into disjoint groups is obtained.
3. *Density-based clustering*: The key idea of this type of clustering is to group neighbouring objects of a data set into clusters based on density conditions. This includes grid-based algorithms that quantise the space into a finite number of cells and then do operations in the quantised space.

For each of above categories there is a wealth of subtypes and different algorithms for finding the clusters. Thus, according to the type of variables allowed in the data set additional categorizations include [14]: (i) Statistical algorithms, which are based on statistical analysis concepts and use similarity measures to partition objects and they are limited to numeric data.

(ii) Conceptual algorithms that are used to cluster categorical data. (iii) Fuzzy clustering algorithms, which use fuzzy techniques to cluster data and allow objects to be classified into more than one clusters. Such algorithms lead to clustering schemes that are compatible with everyday life experience as they handle the uncertainty of real data. (iv) Crisp clustering techniques, that consider non-overlapping partitions so that a data point either belongs to a class or not. Most of the clustering algorithms result in crisp clusters, and thus can be categorized in crisp clustering. (v) Kohonen net clustering, which is based on the concepts of neural networks.

In the remaining discussion, partitional clustering algorithms will be described in more detail; other techniques will be dealt with separately.

Partitional Algorithms

In general terms, the clustering algorithms are based on a criterion for assessing the quality of a given partitioning. More specifically, they take as input some parameters (e.g., number of clusters, density of clusters) and attempt to define the best partitioning of a data set for the given parameters. Thus, they define a partitioning of a data set based on certain assumptions and not necessarily the "best" one that fits the data set.

In this category, K-Means is a commonly used algorithm [10]. The aim of K-Means clustering is the optimisation of an objective function that is described by the equation:

$$E = \sum_{i=1}^{c} \sum_{x \in C_i} d(x, m_i)$$

In the above equation, m_i is the center of cluster C_i, while $d(x, m_i)$ is the Euclidean distance between a point x and m_i. Thus, the criterion function E attempts to minimize the distance of every point from the center of the cluster to which the point belongs.

It should be noted that optimizing E is a combinatorial problem that is NP-Complete and thus any practical algorithm to optimize it cannot guarantee optimality. The K-means algorithm is the first practical and effective heuristic that was suggested to optimize this criterion, and owes its popularity to its good performance in practice. The K-means algorithm begins by initialising a set of c cluster centers. Then, it assigns each object of the dataset to the cluster whose center is the nearest, and re-computes the centers. The process continues until the centers of the clusters stop changing.

Another algorithm of this category is PAM (Partitioning Around Medoids). The objective of PAM is to determine a representative object (medoid) for each cluster, that is, to find the most centrally located objects within the clusters. The algorithm begins by selecting an object as medoid for each of c clusters. Then, each of the non-selected objects is grouped with the medoid to which it is the most similar. PAM swaps medoids with other non-selected objects until all objects qualify as medoid. It is clear that PAM is an expensive algorithm with respect to finding the medoids, as it compares an object with the entire dataset [12].

CLARA (Clustering Large Applications), is an implementation of PAM in a subset of the dataset. It draws multiple samples of the dataset, applies PAM on samples, and then outputs the best clustering out of these samples [12]. CLARANS (Clustering Large Applications based on Randomized Search), combines the sampling techniques with PAM. The clustering process can be presented as searching a graph where every node is a potential solution, that is, a set of k medoids. The clustering obtained after replacing a medoid is called the neighbour of the current clustering. CLARANS selects a node and compares it to a user-defined number of their neighbours searching for a local minimum. If a better neighbor is found (i.e., having lower-square error), CLARANS moves to the neighbour's node and the process starts again; otherwise the current clustering is a local optimum. If the local optimum is found, CLARANS starts with a new randomly selected node in search for a new local optimum.

The algorithms described above result in crisp clusters, meaning that a data point either belongs to a cluster or not. The clusters are non-overlapping and this kind of partitioning is further called crisp clustering. The issue of uncertainty support in the clustering task leads to the introduction of algorithms that use fuzzy logic concepts in their procedure. A common fuzzy clustering algorithm is the Fuzzy C-Means (FCM), an extension of classical C-Means algorithm for fuzzy applications [2]. FCM attempts to find the most characteristic point in each cluster, which can be considered as the "center" of the cluster and, then, the grade of membership for each object in the clusters.

Another approach proposed in the literature to solve the problems of crisp clustering is based on probabilistic models. The basis of this type of clustering algorithms is the EM algorithm, which provides a quite general approach to learning in presence of unobservable variables [11]. A common algorithm is the probabilistic variant of K-Means, which is based on the mixture of Gaussian distributions. This approach of K-Means uses probability density rather than distance to associate records with clusters. More specifically, it regards the centers of clusters as means of Gaussian distributions. Then, it estimates the probability that a data point is generated by the jth Gaussian (i.e., belongs to jth cluster). This approach is based on Gaussian model to extract clusters and assigns the data points to clusters assuming that they are generated by normal distribution. Also, this approach is implemented only in the case of algorithms based on the EM (Expectation Maximization) algorithm.

Another type of clustering algorithms combine graph partitioning and hierarchical clustering algorithms characteristics. Such algorithms include CHAMELEON [9], which measures the similarity among clusters based on a dynamic model contrary to the clustering algorithms discussed above. Moreover in the clustering process both the inter-connectivity and closeness between two clusters are taken into account to decide how to merge the clusters. The merge process based on the dynamic model facilitates the discovery of natural and homogeneous clusters. Also it is applicable to all types of data as long as a similarity function is specified. Finally, BIRCH [ZRL99] uses a data structure called CF-Tree for partitioning the incoming data points in an incremental and dynamic way, thus providing an effective way to cluster very large datasets.

Partitional algorithms are applicable mainly to numerical data sets. However, there are some variants of K-Means such as K-prototypes, and K-mode [7] that are based on the K-Means algorithm, but they aim at clustering categorical data. K-mode discovers clusters while it adopts new concepts in order to handle categorical data. Thus, the cluster centers are replaced with "modes," a new dissimilarity measure used to deal with categorical objects.

The K-means algorithm and related techniques tend to produce spherical clusters due to the use of a symmetric objective function. They require the user to set only one parameter, the desirable number of clusters K. However, since the objective function gets smaller monotonically as K increases, it is not clear how to define what is the best number of clusters for a given dataset. Although several approaches have been proposed to address this shortcoming [14], this is one of the main disadvantages of partitional algorithms. Another characteristic of the partitional algorithms is that they are unable to handle noise and outliers and they are not suitable to discover clusters with non-convex shapes. Another characteristic of K-means is that the algorithm does not display a monotone behavior with respect to K. For example, if a dataset is clustered into M and 2M clusters, it is intuitive to expect that the smaller clusters in the second clustering will be subsets of the larger clusters in the first; however this is typically not the case.

Key Applications

Cluster analysis is very useful task in exploratory data analysis and a major tool in a very wide spectrum of applications in many fields of business and science. Clustering applications include:

1. *Data reduction.* Cluster analysis can contribute to the compression of the information included in the data. In several cases, the amount of the available data is very large and its processing becomes very demanding. Clustering can be used to partition the data set into a number of "interesting" clusters. Then, instead of processing the data set as an entity, the representatives of the defined clusters are adopted in the process. Thus, data compression is achieved.
2. *Hypothesis generation.* Cluster analysis is used here in order to infer some hypotheses concerning the data. For instance, one may find in a retail database that there are two significant groups of customers based on their age and the time of purchases. Then, one may infer some hypotheses for the data, that it, "young people go shopping in the evening," "old people go shopping in the morning."
3. *Hypothesis testing.* In this case, the cluster analysis is used for the verification of the validity of a specific hypothesis. For example, consider the following hypothesis: "Young people go shopping in the evening." One way to verify whether this is true is to apply cluster analysis to a representative set of stores. Suppose that each store is represented by its customer's details (age, job, etc.) and the time of transactions. If, after applying cluster analysis, a cluster that corresponds to "young people buy in the evening" is formed, then the hypothesis is supported by cluster analysis.
4. *Prediction based on groups.* Cluster analysis is applied to the data set and the resulting clusters are characterized by the features of the patterns that

belong to these clusters. Then, unknown patterns can be classified into specified clusters based on their similarity to the clusters' features. In such cases, useful knowledge related to this data can be extracted. Assume, for example, that the cluster analysis is applied to a data set concerning patients infected by the same disease. The result is a number of clusters of patients, according to their reaction to specific drugs. Then, for a new patient, one identifies the cluster in which he/she can be classified and based on this decision his/her medication can be made.

5. *Business Applications and Market Research.* In business, clustering may help marketers discover significant groups in their customers' database and characterize them based on purchasing patterns.
6. *Biology and Bioinformatics.* In biology, it can be used to define taxonomies, categorize genes with similar functionality and gain insights into structures inherent in populations.
7. *Spatial data analysis.* Due to the huge amounts of spatial data that may be obtained from satellite images, medical equipment, Geographical Information Systems (GIS), image database exploration etc., it is expensive and difficult for the users to examine spatial data in detail. Clustering may help to automate the process of analysing and understanding spatial data. It is used to identify and extract interesting characteristics and patterns that may exist in large spatial databases.
8. *Web mining.* Clustering is used to discover significant groups of documents on the Web huge collection of semi-structured documents. This classification of Web documents assists in information discovery. Another application of clustering is discovering groups in social networks.

In addition, clustering can be used as a pre-processing step for other algorithms, such as classification, which would then operate on the detected clusters.

Cross-references

▶ Cluster and Distance Measure
▶ Clustering for Post Hoc Information Retrieval
▶ Clustering on Streams
▶ Clustering Validity
▶ Clustering with Constraints
▶ Data Mining
▶ Data Reduction
▶ Density-Based Clustering
▶ Dimension Reduction Techniques for Clustering
▶ Document Clustering
▶ Feature Selection for Clustering
▶ Hierarchial Clustering
▶ Semi-Supervised Learning
▶ Spectral Clustering
▶ Subspace Clustering Techniques
▶ Text Clustering
▶ Visual Clustering
▶ Visualizing Clustering Results

Recommended Reading

1. Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 94–105.
2. Bezdeck J.C., Ehrlich R., and Full W. FCM: Fuzzy C-Means algorithm. Comput. Geosci., 10(2–3):191–203, 1984.
3. Ester M., Kriegel H.-Peter., Sander J., and Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. In Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, 1996, pp. 226–231.
4. Everitt B.S., Landau S., and Leese M. Cluster Analysis. Hodder Arnold, London, UK, 2001.
5. Fayyad U.M., Piatesky-Shapiro G., Smuth P., and Uthurusamy R. Advances in Knowledge Discovery and Data Mining. AAAI Press, Menlo Park, CA, 1996.
6. Han J. and Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, San Fransisco, CA, 2001.
7. Huang Z. A fast clustering algorithm to cluster very large categorical data sets in data mining. In Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 1997.
8. Jain A.K., Murty M.N., and Flyn P.J. Data clustering: a review. ACM Comput. Surv., 31(3):264–323, 1999.
9. Karypis G., Han E.-H., and Kumar V. CHAMELEON: a hierarchical clustering algorithm using dynamic modeling. IEEE Computer., 32(8):68–75, 1999.
10. MacQueen J.B. Some methods for classification and analysis of multivariate observations. In Proc. 5th Berkeley Symp. on Mathematical Statistics and Probability, vol. 1, 1967, pp. 281–297.
11. Mitchell T. Machine Learning. McGraw-Hill, New York, 1997.
12. Ng R. and Han J. Efficient and effective clustering methods for spatial data mining. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 144–155.
13. Theodoridis S. and Koutroubas K. Pattern Recognition. Academic Press, New York, 1999.
14. Vazirgiannis M., Halkidi M., and Gunopulos D. Uncertainty Handling and Quality Assessment in Data Mining. Springer, New York, 2003.
15. Wang W., Yang J., and Muntz R. STING: A statistical information grid approach to spatial data mining. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 186–195.
16. Zhang T., Ramakrishnman R., and Linvy M. BIRCH: an efficient method for very large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 103–114.

Clustering Validity

Michalis Vazirgiannis
Athens University of Economics & Business, Athens, Greece

Synonyms

Cluster validation; Cluster stability; Quality assessment; Stability-based validation of clustering

Definition

A problem one faces in clustering is to decide the optimal partitioning of the data into clusters. In this context visualization of the data set is a crucial verification of the clustering results. In the case of large multidimensional data sets (e.g., more than three dimensions) effective visualization of the data set is cumbersome. Moreover the perception of clusters using available visualization tools is a difficult task for humans that are not accustomed to higher dimensional spaces. The procedure of evaluating the results of a clustering algorithm is known under the term *cluster validity*. Cluster validity consists of a set of techniques for finding a set of clusters that best fits natural partitions (of given datasets) without any a priori class information. The outcome of the clustering process is validated by a cluster validity index.

Historical Background

Clustering is a major task in the data mining process for discovering groups and identifying interesting distributions and patterns in the underlying data. In the literature a wide variety of algorithms for different applications and sizes of data sets. The application of an algorithm to a data set, assuming that the data set offers a clustering tendency, aims at discovering its inherent partitions. However, the clustering process is an unsupervised process, since there are no predefined classes or examples. Then, the various clustering algorithms are based on some assumptions in order to define a partitioning of a data set. As a consequence, they may behave in a different way depending on: i. the features of the data set (geometry and density distribution of clusters) and ii. the input parameter values.

One of the most important issues in cluster analysis is the evaluation of clustering results to find the partitioning that best fits the underlying data. This is the main subject of cluster validity. If clustering algorithm parameters are assigned an improper value, the clustering method results in a partitioning scheme that is not optimal for the specific data set leading to wrong decisions. The problems of deciding the number of clusters better fitting a data set as well as the evaluation of the clustering results has been subject of several research efforts. The procedure of evaluating the results of a clustering algorithm is known under the term cluster validity. In general terms, there are three approaches to investigate cluster validity. The first is based on *external criteria*. This implies that the results of a clustering algorithm are evaluated based on a prespecified structure, which is imposed on a data set and reflects one's intuition about the clustering structure of the data set. The second approach is based on *internal criteria*. The results of a clustering algorithm may be evaluated in terms of quantities that involve the vectors of the data set themselves (e.g., proximity matrix). The third approach of clustering validity is based on *relative criteria*. Here the basic idea is the evaluation of a clustering structure by comparing it to other clustering schemes, resulting by the same algorithm but with different parameter values. There are two criteria proposed for clustering evaluation and selection of an optimal clustering scheme: (i) Compactness, the members of each cluster should be as close to each other as possible. A common measure of compactness is the variance, which should be minimized. (ii) Separation, the clusters themselves should be widely spaced.

Foundations

This section discusses methods suitable for the quantitative evaluation of the clustering results, known as cluster validity methods. However, these methods give an indication of the quality of the resulting partitioning and thus they can only be considered as a tool at the disposal of the experts in order to evaluate the clustering results. The cluster validity approaches based on external and internal criteria rely on statistical hypothesis testing. In the following section, an introduction to the fundamental concepts of hypothesis testing in cluster validity is presented.

In cluster validity the basic idea is to test whether the points of a data set are randomly structured or not. This analysis is based on the *Null Hypothesis*, denoted as *Ho*, expressed as a statement of random structure of a data set X. To test this hypothesis, statistical tests are used, which lead to a computationally complex procedure.

Monte Carlo techniques are used as a solution to this problem.

External Criteria

Based on external criteria, one can work in two different ways. First, one can evaluate the resulting clustering structure **C**, by comparing it to an independent partition of the data ***P*** built according to one's intuition about the clustering structure of the data set. Second, one can compare the proximity matrix P to the partition ***P***.

Comparison of C with Partition P (Non-hierarchical Clustering) Let ***C*** = {C_1...C_m} be a clustering structure of a data set X and ***P*** = {P_1...P_s} be a defined partition of the data. Refer to a pair of points ($\mathbf{x_v}$, $\mathbf{x_u}$) from the data set using the following terms:

- ***SS***: if both points belong to the same cluster of the clustering structure **C** and to the same group of partition ***P***.
- ***SD***: if points belong to the same cluster of **C** and to different groups of ***P***.
- ***DS***: if points belong to different clusters of **C** and to the same group of ***P***.
- ***DD***: if both points belong to different clusters of **C** and to different groups of ***P***.

Assuming now that **a**, **b**, **c** and **d** are the number of SS, SD, DS and DD pairs respectively, then $a+b+c+d=\mathbf{M}$ which is the maximum number of all pairs in the data set (meaning, M = N(N−1)/2 where N is the total number of points in the data set).

Now define the following indices to measure the degree of similarity between ***C*** and ***P***:

1. *Rand Statistic*: R = (a + d)/M
2. *Jaccard Coefficient*: J = a/(a + b + c)
   The above two indices range between 0 and 1, and are maximized when m=s. Another known index is the:
3. *Folkes and Mallows index*:

$$FM = a/\sqrt{m_1 m_2} = \sqrt{\frac{a}{a+b}\cdot\frac{a}{a+c}} \qquad [1]$$

   where m_1 = (a + b), m_2= (a + c).
   For the previous three indices it has been proven that the higher the values of these indices are the more similar ***C*** and ***P*** are. Other indices are:
4. *Huberts Γ statistic*:

$$\Gamma = (1/M)\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} X(i,j)Y(i,j) \qquad [2]$$

   High values of this index indicate a strong similarity between the matrices X and Y.
5. *Normalized Γ statistic*:

$$\hat{\Gamma} = \frac{\left[(1/M)\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}(X(i,j)-\mu_X)(Y(i,j)-\mu_Y)\right]}{\sigma_X\sigma_Y} \qquad [3]$$

   where X(i, j) and Y(i, j) are the (i, j) element of the matrices X, Y respectively that one wants to compare. Also μ_x, μ_y, σ_x, σ_y are the respective means and variances of X, Y matrices. This index takes values between –1 and 1.

All these statistics have right-tailed probability density functions, under the random hypothesis. In order to use these indices in statistical tests, one must know their respective probability density function under the Null Hypothesis, *Ho*, which is the hypothesis of random structure of the data set. Thus, if one accepts the Null Hypothesis, the data are randomly distributed. However, the computation of the probability density function of these indices is computationally expensive. A solution to this problem is to use Monte Carlo techniques.

After having plotted the approximation of the probability density function of the defined statistic index, its value, denoted by q, is compared to the q(C_i) values, further referred to as q_i. The indices R, J, FM, Γdefined previously are used as the q index mentioned in the above procedure.

Internal Criteria

Using this approach of cluster validity the goal is to evaluate the clustering result of an algorithm using only quantities and features inherited from the data set. There are two cases in which one applies internal criteria of cluster validity depending on the clustering structure: (i) hierarchy of clustering schemes, and (ii) single clustering scheme.

Validating Hierarchy of Clustering Schemes A matrix called cophenetic matrix, P_c, can represent the hierarchy diagram that is produced by a hierarchical algorithm. The element $P_c(i, j)$ of cophenetic matrix represents the proximity level at which the two vectors x_i and x_j are found in the same cluster for the first time. A statistical index can be defined to measure the degree of similarity between P_c and P (proximity matrix) matrices. This index is called *Cophenetic Correlation Coefficient* and defined as:

$$\mathrm{CPCC} = \frac{(1/M)\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} d_{ij}c_{ij} - \mu_P\mu_C}{\sqrt{\left[(1/M)\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} d_{ij}^2 - \mu_P^2\right]\left[(1/M)\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} c_{ij}^2 - \mu_C^2\right]}}, \quad [4]$$

where $M = N\times(N-1)/2$ and N is the number of points in a data set. Also, μ_p and μ_c are the means of matrices P and P_c respectively, and are defined in the (Eq. 5):

$$\mu_P = (1/M)\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} P(i,j), \quad [5]$$
$$\mu_C = (1/M)\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} P_c(i,j)$$

Moreover, d_{ij}, c_{ij} are the (i, j) elements of P and P_c matrices respectively. The CPCC values range in [–1, 1]. A value of the index close to 1 is an indication of a significant similarity between the two matrices.

Validating a Single Clustering Scheme The goal here is to find the degree of match between a given clustering scheme C, consisting of n_c clusters, and the proximity matrix P. The defined index for this approach is Hubert's Γstatistic (or normalized Γstatistic). An additional matrix for the computation of the index is used, that is

$$Y(i,j) = \begin{cases} 1, \text{ if } x_i \text{ and } x_j \text{ belong to different clusters} \\ 0, \text{ otherwise } \textit{array}. \text{ where } i,j = 1, 1/4, N. \end{cases}$$

The application of Monte Carlo techniques is also a means to test the random hypothesis in a given data set.

Relative Criteria

The major drawback of techniques based on internal or external criteria is their high computational complexity. A different validation approach is discussed in this section. The fundamental idea of the relative criteria is to choose the best clustering scheme of a set of defined schemes according to a pre-specified criterion. More specifically, the problem can be stated as follows:

Let P_{alg} be the set of parameters associated with a specific clustering algorithm (e.g., the number of clusters n_c). Among the clustering schemes C_i, i = 1,...,n_c, is defined by a specific algorithm. For different values of the parameters in P_{alg}, choose the one that best fits the data set.

Then, consider the following cases of the problem:

1. *P_{alg} does not contain the number of clusters, n_c, as a parameter.* In this case, the choice of the optimal parameter values are described as follows: The algorithm runs for a wide range of its parameters' values and the largest range for which n_c remains constant is selected (usually $n_c << N$ (number of tuples)). Then the values that correspond to the middle of this range are chosen as appropriate values of the P_{alg} parameters. Also, this procedure identifies the number of clusters that underlie the data set.
2. *P_{alg} contains n_c as a parameter.* The procedure of identifying the best clustering scheme is based on a validity index. Selecting a suitable performance index, q, one proceeds with the following steps:
   - clustering runs for all values of n_c between n_{cmin} and n_{cmax} defined a priori by the user.
   - For each of n_c values, the algorithm runs r times, using different sets of values for the other parameters of the algorithm (e.g., different initial conditions).
   - The best values of the index q obtained by each n_c are plotted as the function of n_c.

Based on this plot, the best clustering schemes are identified. There are two approaches for defining the best clustering depending on the behavior of q with respect to n_c. Thus, if the validity index does not exhibit an increasing or decreasing trend as n_c increases, one seeks the max (min) of the plot. On the other hand, for indices that increase (decrease) as the number of clusters increase, one searches for the values of n_c at which a significant local change in value of the index occurs. This change appears as a "knee" in the plot and it is an indication of the number of clusters underlying the data set. The absence of a knee is an indication that the data set possesses no clustering structure. Below, some representative relative validity indices are presented.

The Modified Hubert Γ Statistic

The definition of the modified Hubert Γstatistic is given by the equation

$$\Gamma = (1/\mathrm{M}) \sum_{\mathrm{i}=1}^{\mathrm{N}-1} \sum_{\mathrm{j}=\mathrm{i}+1}^{\mathrm{N}} \mathrm{P}(\mathrm{i},\mathrm{j}) \cdot \mathrm{Q}(\mathrm{i},\mathrm{j}) \qquad [6]$$

where N is the number of objects in a data set, M = N (N−1)/2, P is the proximity matrix of the data set and Q is an N × N matrix whose (i, j) element is equal to the distance between the representative points (v_{ci}, v_{cj}) of the clusters where the objects x_i and x_j belong.

Similarly, one can define the normalized Hubert Γstatistic, given by equation

$$\hat{\Gamma} = \frac{\left[(1/\mathrm{M}) \sum_{\mathrm{i}=1}^{\mathrm{N}-1} \sum_{\mathrm{j}=\mathrm{i}+1}^{\mathrm{N}} (\mathrm{P}(\mathrm{i},\mathrm{j}) - \mu_{\mathrm{P}})(\mathrm{Q}(\mathrm{i},\mathrm{j}) - \mu_{\mathrm{Q}})\right]}{\sigma_{\mathrm{P}}\sigma_{\mathrm{Q}}}. \qquad [7]$$

where μ_P, μ_Q, $\sigma_{P,}$ σ_Q are the respective means and variances of P, Q matrices.

If the d(v_{ci}, v_{cj}) is close to d(x_i, x_j) for i, j =1, 2,...,N, P and Q will be in close agreement and the valuesof Γ and $\hat{\Gamma}$(normalized Γ) will be high. Conversely, a high value of Γ ($\hat{\Gamma}$) indicates the existence of compact clusters. Thus, in the plot of normalized Γ versus n_c, one seeks a significant knee that corresponds to a significant increase of normalized Γ. The number of clusters at which the knee occurs is an indication of the number of clusters that occurs in the data. Note that for $n_c = 1$ and $n_c = N$, the index is not defined.

Dunn Family of Indices

A cluster validity index for crisp clustering proposed by Dunn (1974), aims at the identification of "compact and well separated clusters". The index is defined in the following equation for a specific number of clusters

$$D_{n_c} = \min_{i=1,\ldots,n_c} \left\{ \min_{j=i+1,\ldots,n_c} \left(\frac{d(c_i, c_j)}{\max_{k=1,\ldots,n_c} (\mathrm{diam}(\mathrm{c_k}))} \right) \right\} \qquad [8]$$

where d(c_i, c_j) is the dissimilarity function between two clusters c_i and c_j defined as $d(c_i, c_j) = \min_{x \in C_i, y \in C_j} d(x, y)$, and diam(c) is the diameter of a cluster, which may be considered as a measure of clusters' dispersion. The diameter of a cluster *C* can be defined as follows:

$$\mathrm{diam(C)} = \max_{\mathrm{x,y} \in \mathrm{C}} \{\mathrm{d(x, y)}\} \qquad [9]$$

If the data set contains compact and well-separated clusters, the distance between the clusters is expected large and the diameter of the clusters is expected small. Based on the Dunn's index definition, one concludes that large values of the index indicate the presence of compact and well-separated clusters.

The problems of the Dunn index are: (i) its considerable time complexity, and (ii) its sensitivity to the presence of noise in data sets, since these are likely to increase the values of the diameter.

RMSSDT, SPR, RS, CD

This family of validity indices is applicable in the cases that hierarchical algorithms are used to cluster the data sets. Hereafter the discussion refers to the definitions of four validity indices, which have to be used simultaneously to determine the number of clusters existing in the data set. These four indices are applied to each step of a *hierarchical* clustering algorithm and they are known as:

- *Root-mean-square standard deviation (RMSSTD) of the new cluster,*
- *Semi-partial R-squared (SPR),*
- *R-squared (RS),*
- *Distance between two clusters (CD).*

Getting into a more detailed description of them, one can say that:

RMSSTD of a new clustering scheme defined at a level of a clustering hierarchy is the square root of the variance of all the variables (attributes used in the clustering process). This index measures the homogeneity of the formed clusters at each step of the hierarchical algorithm. Since the objective of cluster analysis is to form homogeneous groups the RMSSTD of a cluster should be as small as possible. Where the values of RMSSTD are higher than the ones of the previous step, one has an indication that the new clustering scheme is worse.

In the following definitions, the term SS is used, which means *Sum of Squares* and refers to the equation:

$$\mathrm{SS} = \sum_{i=1}^{n} (X_i - \overline{X})^2 \qquad [10]$$

Along with this, additional terms will be used, such as:

1. SS_w referring to the sum of squares within group,
2. SS_b referring to the sum of squares between groups,
3. SS_t referring to the total sum of squares, of the whole data set.

In the case cluster join to form a new one, *SPR*- for the new cluster – is defined as the difference between SS_w of the new cluster and the sum of the SS_w's values of clusters joined to obtain the new cluster (*loss of homogeneity*), divided by the SS_t for the whole data set. This index measures the loss of homogeneity after merging the two clusters of a single algorithm step. If the index value is zero then the new cluster is obtained by merging two perfectly homogeneous clusters. If its value is high then the new cluster is obtained by merging two heterogeneous clusters.

RS of the new cluster is the ratio of SS_b over SS_t. SS_b is a measure of difference between groups. Since $SS_t = SS_b + SS_{w,}$ the greater the SS_b the smaller the SS_w and vice versa. As a result, the greater the differences between groups, the more homogenous each group is and vice versa. Thus, RS may be considered as a measure of dissimilarity between clusters. Furthermore, it measures the degree of homogeneity between groups. The values of RS range between 0 and 1. Where the value of RS is zero, there is an indication that no difference exists among groups. On the other hand, when RS equals 1 there is an indication of significant difference among groups.

Key Applications

There is a certain cross disciplinary interest for clustering validity method and indices. A prominent area where cluster validity measures apply is the area of biological data [2,6]. Patterns hidden in gene expression data offer a tremendous opportunity for an enhanced understanding of functional genomics. However, the large number of genes and the complexity of biological networks greatly increase the challenges of comprehending and interpreting the resulting mass of data, which often consists of millions of measurements. The data mining process aims to reveal natural structures and identify interesting patterns in the underlying data. Clustering techniques constitute a first essential step toward addressing this challenge. Moreover recent research effort papers in the area of image segmentation [13,3].

The area is fertile as the clustering issue is a fundamental problem and the application domains are still widening. Challenging relevant research directions [9] follow:

- Is there a principled way to measure the quality of a clustering on particular data set?
- Can every clustering task be expressed as an optimization of some explicit, readily computable, objective cost function?
- Can stability be considered a first principle for meaningful clustering?
- How should the similarity between different clusterings be measured?
- Can one distinguish clusterable data from structureless data?
- What are the tools that should be imported from other relevant areas of research?

Cross-references

▶ Cluster and Distance Measure
▶ Clustering on Streams
▶ Clustering with Constraints
▶ Density-Based Clustering
▶ Document Clustering
▶ Feature Selection for Clustering
▶ Hierarchial Clustering
▶ Semi-Supervised Learning
▶ Spectral Clustering
▶ Subspace Clustering Techniques
▶ Text Clustering
▶ Visual Clustering
▶ Visualizing Clustering Results

Recommended Reading

1. Bezdek J.C. and Pal N.R. Some new indexes of cluster validity, IEEE Trans., Systems, Man, and Cybernetics, Part B. 28 (3):301–315, 1998.
2. Datta S. and Datta S. Comparisons and validation of statistical clustering techniques for microarray gene expression data. Bioinformatics, 19(4):459–466, 2003.
3. El-Melegy M.T., Zanaty E.A., Abd-Elhafiez W.M., and Farag A.A. On cluster validity indexes in fuzzy and hard clustering algorithms for image segmentation. In Proc. Int. Conf. Image Processing, 2007, pp. 5–8.
4. Halkidi M., Batistakis Y., and Vazirgiannis M. On clustering validation techniques. J. Intell. Inf. Syst., 17(2–3):107–145, 2001.
5. Halkidi M., Gunopulos D., Vazirgiannis M., Kumar N., and Domeniconi C. A clustering framework based on subjective and objective validity criteria. ACM Trans. Knowl. Discov. Data, 1(4), 2008.

6. Jiang D., Tang C., and Zhang A. Cluster Analysis for Gene Expression Data: A Survey. IEEE Trans. Knowl. Data Eng., 16(11):1370–1386, 2004.
7. Kim M. and Ramakrishna R.S. New indices for cluster validity assessment. Pattern Recogn. Lett., 26(15):2353–2363, 2005.
8. Maulik U. and Bandyopadhyay S. Performance evaluation of some clustering algorithms and validity indices. IEEE Trans. Pattern Anal. Mach. Intell., 24(12):1650–1654, 2002.
9. NIPS 2005 workshop on theoretical foundations of clustering, Saturday, December 10th, 2005. Available at: (http://www.kyb.tuebingen.mpg.de/bs/people/ule/clustering_workshop_nips05/clustering_workshop_nips05.htm_).
10. Pal N.R. and Bezdek J.C. On cluster validity for the fuzzy c-means model, IEEE Trans. Fuzzy Systems., 3(3):370–379, 1995.
11. Rand W.M. Objective criteria for the evaluation of clustering methods. J. Am. Stat. Assoc., 66(336):846–850, 1971.
12. Wang J.-S. and Chiang J.-C. A cluster validity measure with a hybrid parameter search method for the support vector clustering algorithm. Pattern Recognit., 41(2):506–520, 2008.
13. Zhang J. and Modestino J.W. A model-fitting approach to cluster validation with application to stochastic model-based image segmentation. IEEE Trans. Pattern Anal. Mach. Intell., 12(10):1009–1017, 1990.

Clustering with Constraints

Ian Davidson
University of California-Davis, Davis, CA, USA

Synonyms

Semi-supervised clustering

Definition

The area of clustering with constraints makes use of hints or advice in the form of constraints to aid or bias the clustering process. The most prevalent form of advice are conjunctions of pair-wise instance level constraints of the form must-link (ML) and cannot-link (CL) which state that pairs of instances should be in the same or different clusters respectively. Given a set of points P to cluster and a set of constraints C, the aim of clustering with constraints is to use the constraints to improve the clustering results. Constraints have so far being used in two main ways: (i) Writing algorithms that use a standard distance metric but attempt to satisfy all or as many constraints as possible and (ii) Using the constraints to learn a distance function that is then used in the clustering algorithm.

Historical Background

The idea of using constraints to guide clustering was first introduced by Wagstaff and Cardie in their seminal paper ICML 2000 [13] with a modified COBWEB-style algorithm that attempts to satisfy all constraints. Later [14] they introduced constraints to the *k*-means algorithms. Their algorithms (as most algorithms now do) look at satisfying a conjunction of must-link and cannot-link constraints. Independently, Cohn, Caruana and McCallum [3,4] introduced constraints as a user feedback mechanism to guide the clustering algorithm to a more useful result.

In 2002 Xing and collaborators [15] (NIPS 2002) and Klein and collaborators (ICML 2002) [12] explored making use of constraints by learning a distance function for non-hierarchical clustering and a distance matrix for hierarchical clustering respectively.

Basu and collaborators more recently have looked at key issues such as which are the most informative sets of constraints [2] and seeding algorithms using constraints [1]. Gondek has explored using constraints to find orthogonal/alternative clusterings of data [3,11]. Davidson and Ravi explored the intractability issues of clustering under constraints for non-hierarchical clustering [6], hierarchical clustering [5] and non-hierarchical clustering with feedback [9].

Foundations

Clustering has many successful applications in a variety of domains where the objective function of the clustering algorithm finds a novel and useful clustering. However, in some application domains the typical objective functions may lead to well-known or non-actionable clusterings of the data. This could be overcome by an ad hoc approach such as manipulating the data. The introduction of constraints into clustering allows a principled approach to incorporate user preferences or domain expertise into the clustering process so as to guide the algorithm to a desirable solution or away from an undesirable solution. The typical semi-supervised learning situations involves having a label associated with a subset of the available instances. However in many domains, knowledge of the relevant categories is incomplete and it is easier to obtain pairwise constraints either automatically or from domain experts.

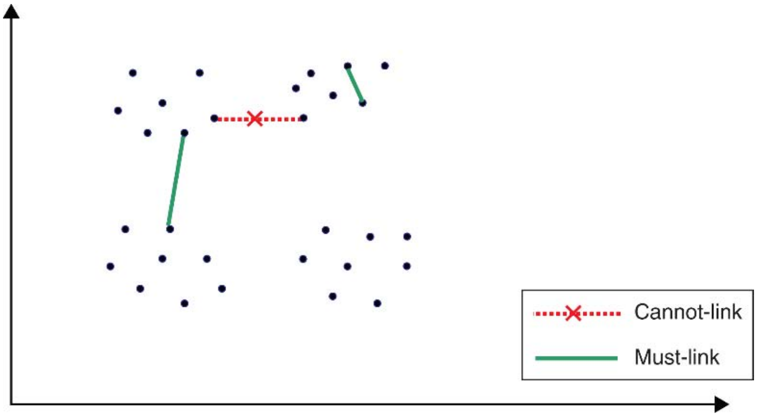
Types of Constraints. Must-link and cannot-link constraints are typically used since they can be easily generated from small amounts of labeled data (generate a must-link between two instances if the labels agree, cannot-link if they disagree) or from domain experts. They can be used to represent geometric properties [6,14] by noting that for instance, making the maximum cluster diameter be α is equivalent to enforcing a conjunction of cannot-link constraints between all points whose distance is greater than α. Similarly, clusters can be separated by distance at at least δ by enforcing a conjunction of must-link constraints between all points whose distance is less than δ. Both types of instance-level constraints have interesting properties that can be used to effectively generate many additional constraints. Must-link constraints are transitive: $ML(x,y)$, $ML(y,z) \rightarrow ML(x,z)$ and cannot link constraints have an entailment property: $ML(a,b)$, $ML(x,y)$, $CL(a,x) \rightarrow CL(a,y)$, $CL(b,x)$, $CL(b,y)$.

How Constraints Are Used. Constraints have typically been used in clustering algorithms in two ways. Constraints can be used to modify the cluster assignment stage of the cluster algorithm [4,14], to enforce satisfaction of the constraints or as many as possible [2,6]. These approaches typically use a standard distance or likelihood function. Alternatively, the distance function of the clustering algorithm can also be trained either before or after the clustering actually occurs using the constraints [12,15]. The former are called constraint-based approaches and the later distance based approaches.

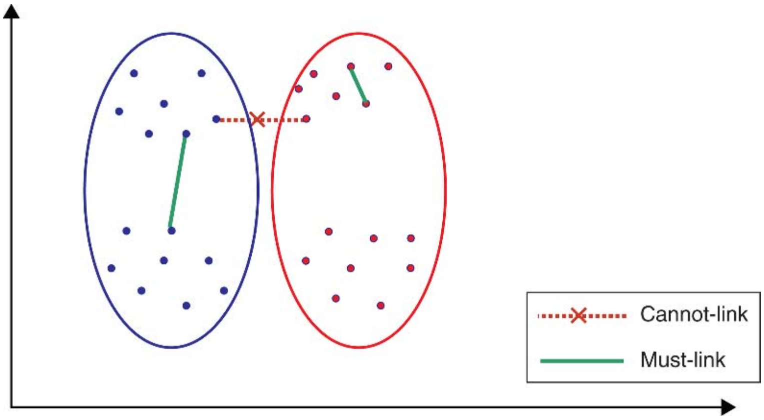
Constraint-Based Methods. In constraint-based approaches, the clustering algorithm itself (typically the assignment step) is modified so that the available constraints are used to bias the search for an appropriate clustering of the data. Fig. 2 shows how though two clusterings exist (a horizontal and vertical clustering) just three constraints can rule out the former.

Constraint-based clustering is typically achieved using one of the following approaches:

1. Enforcing constraints to be satisfied during the cluster assignment in the clustering algorithm [5,13].



Clustering with Constraints. Figure 1. Input instances and constraints.



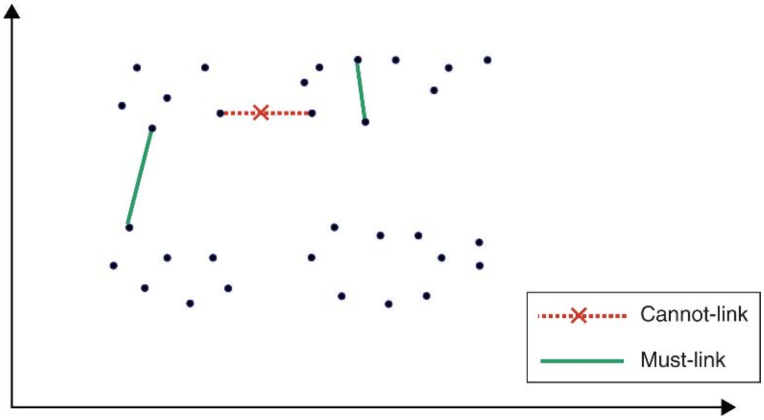
Clustering with Constraints. Figure 2. A clustering that satisfies all constraints.

2. Modifying the clustering objective function so that it includes a term for satisfying specified constraints. Penalties for violating constraints have been explored in the maximum likelihood framework [2] and distance framework [6].
3. Initializing clusters and inferring clustering constraints based on neighborhoods derived from labeled examples [1].

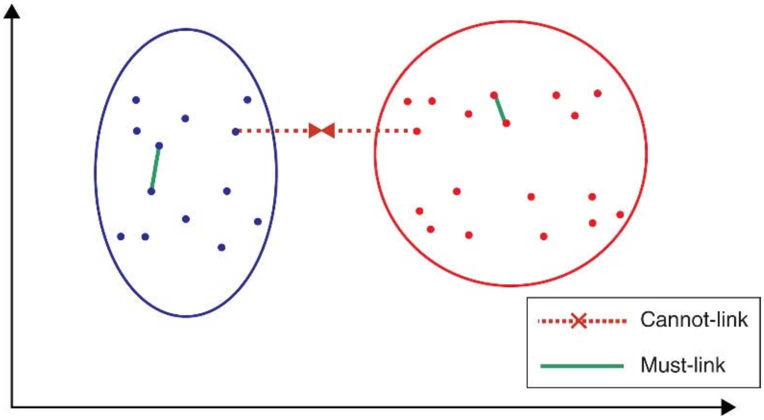
Each of the above approaches provides a simple method of modifying existing partitional and agglomerative style hierarchical algorithms to incorporate constraints. For more recent advances in algorithm design such as the use of variational techniques for constrained clustering see [3].

Distance-Based Methods. In distance-based approaches, an existing clustering algorithm that uses a distance measure is employed. However, rather than use the Euclidean distance metric, the distance measure is first trained to "satisfy" the given constraints. The approach of Xing and collaborators [15] casts the problem of learning a distance *metric* from the constraints so that the points (and surrounding points) that are part of the must-link (cannot-link) constraints are close together (far apart). They consider two formulations: firstly learning a generalized Mahanabolis distance metric which essentially stretches or compresses each axis as appropriate. Figure 4 gives an example where the constraints can be satisfied by stretching the *x*-axis and compressing the *y*-axis and then applying a clustering algorithm to the new data space. The second formulation allows a more complex transformation on the space of points.

Klein and collaborators [12] explore learning a distance *matrix* from constraints for agglomerative clustering. Only points that are directly involved in the constraints are brought closer together or far apart



Clustering with Constraints. Figure 3. Input instances and constraints.



Clustering with Constraints. Figure 4. A learnt distance space respective of the constraints.

using a multi-step approach of making must-linked points have a distance of 0 and cannot-linked points having the greatest distance.

There have been some algorithms that try to both enforce constraints and learn distance functions from constraints [2].

Key Applications

Key application areas include images, video, biology, text, web pages, audio (speaker identification) [3] and GPS trace information [14].

URL to Code

http://www.constrained-clustering.org

Cross-references

- ▶ Clustering
- ▶ Semi-Supervised Learning

Recommended Reading

1. Basu S., Banerjee A., and Mooney R. Semi-supervised clustering by seeding. In Proc. 19th Int. Conf. on Machine Learning, 2002, pp. 27–34.
2. Basu S., Banerjee A., and Mooney R.J. Active semi-supervision for pairwise constrained clustering. In Proc. SIAM International Conference on Data Mining, 2004.
3. Basu S., Davidson I., and Wagstaff K. (eds.). Constrained Clustering: Advances in Algorithms, Theory and Applications. Chapman & Hall, CRC Press, 2008.
4. Cohn D., Caruana R., and McCallum A. Semi-Supervised Clustering with User Feedback. Technical Report 2003–1892. Cornell University, 2003.
5. Davidson I. and Ravi S.S. Agglomerative hierarchical clustering with constraints: theoretical and empirical results. In Principles of Data Mining and Knowledge Discovery, 9th European Conf., 2005, pp. 59–70.
6. Davidson I. and Ravi S.S. Clustering with constraints: feasibility issues and the *k*-means algorithm. In Proc. SIAM International Conference on Data Mining, 2005.
7. Davidson I. and Ravi S.S. Identifying and generating easy sets of constraints for clustering. In Proc. 15th National Conf. on AI, 2006.
8. Davidson I., Ester M., and Ravi S.S. Efficient incremental clustering with constraints. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007, pp. 204–249.
9. Davidson I. and Ravi S.S. Intractability and clustering with constraints. In Proc. 24th Int. Conf. on Machine Learning, 2007, pp. 201–208.
10. Davidson I. and Ravi S.S. The complexity of non-hierarchical clustering with instance and cluster level constraints. Data Mining Know. Discov., 14(1):25–61, 2007.
11. Gondek D. and Hofmann T. Non-redundant data clustering. In Proc. 2004 IEEE Int. Conf. on Data Mining, 2004, pp. 75–82.
12. Klein D., Kamvar S.D., and Manning C.D. From instance-level constraints to space-level constraints: making the most of prior knowledge in data clustering. In Proc. 19th Int. Conf. on Machine Learning, 2002, pp. 307–314.
13. Wagstaff K. and Cardie C. Clustering with instance-level constraints. In Proc. 17th Int. Conf. on Machine Learning, 2000, pp. 1103–1110.
14. Wagstaff K., Cardie C., Rogers S., and Schroedl S. Constrained K-means clustering with background knowledge. In Proc. 18th Int. Conf. on Machine Learning, 2001, pp. 577–584.
15. Xing E., Ng A., Jordan M., and Russell S. Distance metric learning, with application to clustering with side-information. Adv. Neural Inf. Process. Syst. 15, 2002.

CM Sketch

▶ Count-Min Sketch

CMA

▶ Computational Media Aesthetics

CO Query, Content-Only Query

▶ Content-Only Query

CO+S Query

▶ Content-and-Structure Query

Co-clustering

▶ Subspace Clustering Techniques

CODASYL Data Model

▶ Network Data Model

Collaborative Software

▶ Social Applications

Co-locations

▶ Spatial Data Mining

Colored Nets

▶ Petri Nets

Column Segmentation

Sunita Sarawagi
IIT Bombay, Mumbai, India

Synonyms

Text segmentation; Record extraction; Information extraction

Definition

The term column segmentation refers to the segmentation of an unstructured text string into segments such that each segment is a column of a structured record.

As an example, consider a text string S=``18100 New Hampshire Ave. Silver Spring, MD 20861'' representing an unstructured form of an Address record. Let the columns of this record be House number, Street name, City name, State, Zip and Country. In column segmentation, the goal is to segment S and assign a column label to each segment so as to get an output of the form:

```
House Number   :  18100
Street Name    :  New Hampshire Ave.
City           :  Silver Spring
State          :  MD
Zip            :  20861
Country        :  --
```

Historical Background

The column segmentation problem is a special case of a more general problem of Information Extraction (IE) that refers to the extraction of structure from unstructured text. Column segmentation is typically performed on short text strings where most of the tokens belong to one of a fixed set of columns. In the more general IE problem, the unstructured text could be an arbitrary paragraph or an HTML document where the structured entities of interest form a small part of the entire string.

There is a long history of work on information extraction [5]. Most of the early work in the area was in the context of natural language processing, for example for extracting named entities like people names, organization names, and location names from news articles. The early systems were based on hand-coded set of rules and relied heavily on dictionaries of known records. Later systems were based on statistical methods like maximum entropy taggers [9], Hidden Markov Models [11] and Conditional Random Fields (CRFs) [7].

In the database research community, interest in column segmentation arose in the late nineties as a step in the process of cleaning text data for data warehousing. Many commercial tools were developed purely for the purposes of cleaning names and addresses. These were based on hand-coded, rule-based, database driven methods that work only for the region that they are developed for and do not extend to other domains. Much manual work has to be done to rewrite these rules when shifting the domain from one locality to another. This led to the adoption of statistical techniques [1,3] which proved to be more robust to noisy inputs.

Foundations

A formal definition of column segmentation follows. Let $Y = \{y_1,\ldots,y_m\}$ denote the set of column types of the structured record. Given any unstructured text string $\mathbf{x}$, column segmentation finds segments of $\mathbf{x}$ and labels each with one of the columns in Y. The input $\mathbf{x}$ is typically treated as a sequence of tokens obtained by splitting $\mathbf{x}$ along a set of delimiters. Let $x_1,\ldots,x_n$ denote such a sequence of tokens. A segmentation of $\mathbf{x}$ is a sequence of segments $s_1\ldots s_p$. Each segment s_j consists of a *start position* t_j, an *end position* u_j, and a *label* $y_j \in Y \cup \{\text{“Other”}\}$. The special label “Other” is used to label tokens not belonging to any of the columns. The segments are assumed to be contiguous, that is, segment s_{j+1} begins right after segment s_j ends. Also, the last segment ends at n and the first segment starts at 1.

As a second example consider a citation String T=P.P.Wangikar, T.P. Graycar, D.A. Estell, D.S. Clark, J.S. Dordick (1993) Protein and Solvent Engineering of Subtilising BPN' in Nearly Anhydrous Organic Media J.Amer. Chem. Soc. 115, 12231-12237. and a set of columns: Author names, title, year, publication venue, volume, number. A segmentation of this string is:

```
Title     :  Protein and Solvent Engineering
             of Subtilising BPN
Authors   :  P.P.Wangikar, T.P.Graycar,
             D.A. Estell, D.S. Clark, J.S. Dordick
Year      :  1993
Venue     :  Nearly Anhydrous Organic Media
             J.Amer. Chem. Soc.
Volume    :  115
Number    :  --
Pages     :  12231-12237
```

In this example, the tokens "in", "(" and ")" of the input have been assigned label "Other".

Challenges

The problem of column segmentation is challenging because of the presence of various kinds of noise in the unstructured string.

- The same column might be represented in many different forms, for example "Street" might be abbreviated as "St." or "st".
- The order in which columns appear might be different in different strings: for example, in some citations authors could be before title, and after title in others.
- Columns might be missing: some addresses might contain a country name, others may not.
- Strings from different sources might be formatted differently: for example, some citations might use a comma to separate fields whereas others might have no regular delimiter between fields.

Main Techniques

A column segmentation technique needs to combine information from multiple different sources of evidence to be able to correctly recognize segmentations in noisy strings. One source is the characteristic words in each elements, for example the word "street" appears in road-names. A second source is the limited partial ordering between its element. Often the first element is a house number, then a possible building name and so on and the last few elements are zipcode and state-name. A third source is the typical number of words in each element. For example, state names usually have one or two words whereas road names are longer. Even within a field, some words are more likely to appear in the beginning of the field rather than towards its end. Often, there is a pre-existing database of known values of columns. Match of a substring of the text to an existing database column, can be a valuable clue for segmentation. The format of the entry, presence of certain regular expression, capitalization, and punctuation patterns can be useful when word-level matches are absent. A good column segmentation technique would combine evidence from all of these clues in performing the final segmentation.

The three main types of column segmentation techniques are:

Rule-Based Systems

A rule-based technique, as the name suggests, encodes one or more of the above clues as rules. These are applied in a specified order and when more than two rules conflict, another set of rule resolution mechanisms are used to decide which one wins.

Here are some examples of rules that can be used to extract columns from citation records:

Punctuation CapsWord{2–10} Dot → Title
CapsWord Comma Initial Dot Initial Dot → Author name
Initial Dot CapsWord Comma → Author name
AllCaps Words{1–2} Journal → Journal

For example, the first rule marks as title any substring of two to ten capitalized words appearing between a punctuation and a full-stop. The second rule marks as an author name any substring consisting of a capitalized word followed by comma and two initials. This would identify strings of the form "Gandhi, M. K." as author names. Whereas the third rule would mark strings like "V. Ganti," as author names. The fourth rule would mark string like "ACM computing Journal" as journal names.

Such rules could be either hand-coded or learnt from example datasets [2,6]. Existing rule-based techniques are able to concentrate only on a subset of the above mentioned clues to limit the complexity of the learnt rules. They provide high precision segmentation in uniform settings where the amount of noise is limited. When the input becomes noisy, rule-based systems tend to lose on recall.

Hidden Markov Models

Hidden Markov Models (HMMs) provide an intuitive statistical method for combining many of the above clues in a unified model. A HMM is a probabilistic finite state automata where the states represent the fields to be extracted, directed edges between edges are attached with probability values indicating probability of transitioning from one state to another, and states are attached with a distribution over the words that can be generated from the state. A segmentation of a string *S* is achieved by finding the sequence of states for which the product of the probability of generating the words in *S* and following the transitions in state sequences is maximized. Such a path can be found efficiently using a dynamic programming algorithm. The parameters controlling the transition and word distributions of states are learnt using examples of correctly segmented strings.

An example, of a Hidden Markov Model trained to recognize Indian addresses appears in Fig. 1. The number of states is 10 and the edge labels depict the state transition probabilities. For example, the probability of an address beginning with House Number is 0.92 and that of seeing a City after Road is 0.22. The dictionary and the emission probabilities are not shown for compactness.

For more details on the use of HMMs in column segmentation see [1,3,11].

Conditional Models

A limitation of HMMs is that the distribution that controls the generation of words within a state is generative, and can therefore capture only a limited set of properties of the words it can generate. For example, it is complicated to account for various orthographic properties of words, like its capitalization pattern, or the delimiter following that word. These limitations are removed by recently proposed formalisms like Conditional Random Fields (CRFs) that capture the conditional distribution of column sequence *given* the sequence of words in a string *S*. This enables the incorporation of any arbitrary set of clues derived from a word and the words in its neighborhood. Also it becomes easy to incorporate clues derived from the degree of match of a proposed column with pre-existing values in the database.

A CRF models the conditional probability distribution over segmentations **s** for a given input sequence **x** as follows:

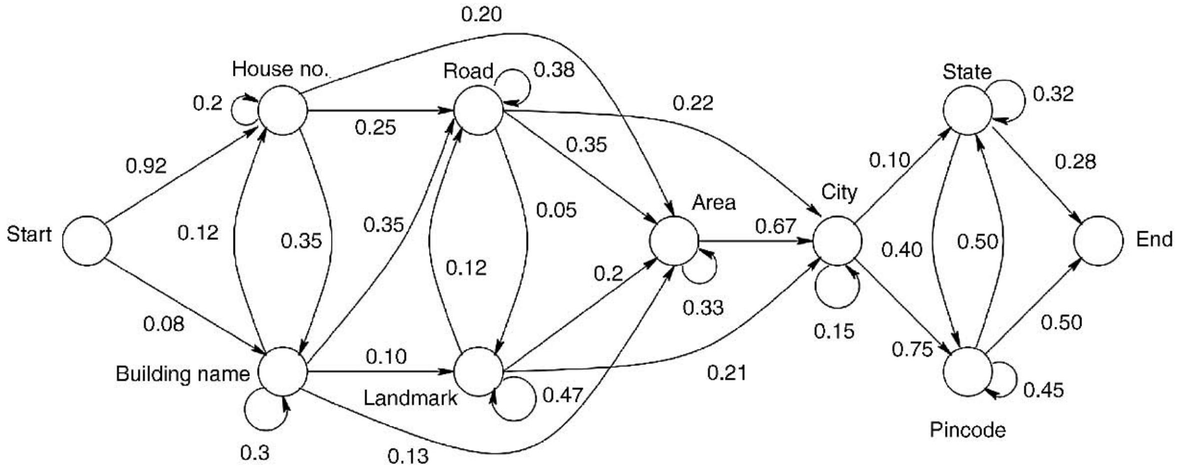
$$\Pr(\mathbf{s}|\mathbf{x},\mathbf{W}) = \frac{1}{Z(\mathrm{x})}\exp\left(\mathbf{W}.\sum_{\mathrm{j}}\mathbf{f}(j,\mathbf{x},\mathbf{s})\right) \qquad (1)$$

where $\mathbf{f}(j,\mathbf{x},\mathbf{s})$ is a vector of local feature functions $f_1 \ldots f_N$ of **s** at the jth segment and $\mathbf{W} = (W_1, W_2, \ldots, W_N)$ is a weight vector that encodes the importance of each feature function in **f**. $Z(\mathbf{x}) = \sum_{\mathbf{s}'} \mathbf{exp}(\mathbf{W} \cdot \sum_{\mathrm{j}} \mathbf{f}(j, \mathbf{x},\mathbf{s}'))$ is a normalization factor. The label of a segment depends on the label of the previous segment and the properties of the tokens comprising this segment and the neighboring tokens. Thus a feature for segment $s_j = (t_j, u_j, y_j)$ is a function of the form $f(y_j, y_{j-1}, \mathbf{x}, t_j, u_j)$ that returns a numeric value. Example of such features are:

$$f_8(y_i, y_{i-1}, \mathbf{x}, 3, 5) = [\![x_3x_4x_5 \text{ appears in a journal list}]\!].[\![y_i = \text{journal}]\!]$$

$$f_{12}(y_i, y_{i-1}, \mathbf{x}, 19, 19) = [\![x_{19} \text{ is an integer}]\!].[\![y_i = \text{year}]\!].[\![y_{i-1} = \text{month}]\!]$$

The weight vector **W** is learnt during training via a variety of methods, such as likelihood maximization [7]. During *segmentation*, the goal is to find a



Column Segmentation. Figure 1. An example of trained HMM for segmenting addresses.

$\mathbf{s} = s_1 \ldots s_p$ for the input sequence $\mathbf{x} = x_1 \ldots x_n$ such that $\Pr(\mathbf{s}|\mathbf{x}, \mathbf{W})$ (as defined by (1) is maximized.

$$\arg\max_{s} \Pr(\mathbf{s}|\mathbf{x}, \mathbf{W}) = \arg\max_{\mathbf{s}} \mathbf{W}. \sum_j \mathbf{f}(y_i, y_{j-1}, \mathbf{x}, t_j, u_j)$$

The right hand side can be efficiently computed using dynamic programming. Let L be an upper bound on segment length. Let $\mathbf{s}_{i:y}$ denote set of all partial segmentation starting from 1 (the first index of the sequence) to i, such that the last segment has the label y and ending position i. Let $V(i, y)$ denote the largest value of $\mathbf{W} \cdot \sum_j \mathbf{f}(j, \mathbf{x}, \mathbf{s}')$ for any $\mathbf{s}' \in \mathbf{s}_{i:y}$. The following recursive calculation finds the best segmentation:

$$V(i,y) = \begin{cases} \max_{y', i'=i-L \ldots i-1} V(i', y') \\ \qquad + \mathbf{W} \cdot \mathbf{f}(y, y', \mathbf{x}, i'+1, i) & \text{if } i > 0 \\ 0 & \text{if } i = 0 \\ -\infty & \text{if } i < 0 \end{cases}$$

The best segmentation then corresponds to the path traced by $\max_y V(|\mathbf{x}|, y)$.

More details on CRFs can be found in [7] and the extension of CRFs for segmentation can be found in [10]. [8] reports an empirical evaluation of CRFs with HMMs for segmenting paper citations. [4] shows how to perform efficient segmentation using CRFs in the presence of a large pre-existing database of known values.

Key Applications

Column segmentation has many applications, including,

Cleaning of text fields during warehouse construction: In operational datasets, text fields like addresses are often recorded as single strings. When warehousing such datasets for decision support, it is often useful to identify structured elements of the address. This not only allows for richer structured queries, it also serves as a useful pre-processing step for duplicate elimination.

Creation of citation databases: A key step in the creation of citation databases like Citeseer and Google Scholar, is to resolve for each citation, which paper it refers to in the database. Citations as extracted from papers are unstructured text strings. These have to be segmented into component author names, titles, years, and publication venue before they can be correctly resolved to a paper entry in the database.

Extraction of product information from product descriptions: Comparison shopping websites often need to parse structured fields representing various attributes of product from unstructured HTML sources.

URL to Code

Java packages for column segmentation using conditional random fields are available via Source Forge at http://crf.sf.net and as part of the Mallet package at http://mallet.cs.umass.edu

Cross-references

▶ Data Cleaning

Recommended Reading

1. Agichtein E. and Ganti V. Mining reference tables for automatic text segmentation. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 20–29.
2. Aldelberg B. Nodose: a tool for semi-automatically extracting structured and semi-structured data from text documents. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 283–294.
3. Borkar V.R., Deshmukh K., and Sarawagi S. Automatic text segmentation for extracting structured records. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 175–186.
4. Chandel A., Nagesh P.C., and Sarawagi S. Efficient batch top-k search for dictionary-based entity recognition. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
5. Cunningham H. Information Extraction, Automatic. Encyclopedia of Language and Linguistics, 2nd edn., 2005.
6. Kushmerick N., Weld D.S., and Doorenbos R. Wrapper induction for information extraction. In Proc. 15th Int. Joint Conf. on AI., 1997, pp. 729–737.
7. Lafferty J., McCallum A., and Pereira F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proc. 18th Int. Conf. on Machine Learning, 2001, pp. 282–289.
8. Peng F. and McCallum A. Accurate information extraction from research papers using conditional random fields. In HLT-NAACL. 2004, pp. 329–336.
9. Ratnaparkhi A. Learning to parse natural language with maximum entropy models. Mach. Learn., 34, 1999.
10. Sarawagi S. and Cohen W.W. Semi-markov conditional random fields for information extraction. In Advances in Neural Inf. Proc. Syst. 17, 2004.
11. Seymore K., McCallum A., and Rosenfeld R. Learning Hidden Markov Model structure for information extraction. In Papers from the AAAI-99 Workshop on Machine Learning for Information Extraction. 1999, pp. 37–42.

Committee-based Learning

▶ Ensemble

Common Object Request Broker Architecture

▶ CORBA

Common Subexpression Elimination

▶ Multi-Query Optimization

Common Warehouse Metadata Interchange (CWMI)

▶ Common Warehouse Metamodel (CWM™)

Common Warehouse Metamodel

Liam Peyton
University of Ottawa, Ottawa, ON, Canada

Synonyms

Common Warehouse Metadata Interchange (CWMI); CWM

Definition

The Common Warehouse Metamodel (CWM™) is an adopted specification from the OMG (Object Management Group) standards body. It defines standard interfaces that can be used to enable easy interchange of data warehouse and business intelligence metadata between data warehouse tools, data warehouse platforms and data warehouse metadata repositories in distributed heterogeneous environments. It supports relational, non-relational, multi-dimensional, and most other objects found in a data warehousing environment. It leverages three other standards from OMG:

- UML – Unified Modeling Language
- MOF – Meta Object Facility
- XMI – XML Metadata Interchange

The Object Management Group has been an international, open membership, not-for-profit computer industry consortium since 1989 with over 700 member organizations.

Historical Background

An initial Request For Proposal (RFP) for a common warehouse metadata interchange (CWMI) was issued by the OMG (Object Management Group) in 1998. A joint submission was received by the OMG in 1999 from Dimension EDI, Genesis Development Corporation, Hyperion Solutions, International Business Machines, NCR, Oracle, UBS AG, and Unisys.

At the time, there was a competing initiative from the Meta Data Coalition (MDC) which was supported by Microsoft and others. In 2000, however the two initiatives merged when the MDC joined OMG [5]. In 2001, version 1.0 of the specification was adopted with the name: Common Warehouse Metamodel (CWM™). The currently adopted version is 1.1 [1] (March, 2003).

The purpose of the Common Warehouse Metamodel specification was to make it possible for large organizations to have a metadata repository with a single metamodel. In practice this was not possible to achieve, since every data management and analysis tool requires different metadata and different metadata models [3]. Instead, the CWM specification defines interfaces that facilitate the interchange of data warehouse metadata between tools. In particular, the OMG Meta-Object Facility (MOF™) bridges the gap between dissimilar meta-models by providing a common basis for meta-models. If two different meta-models are both MOF-conformant, then models based on them can reside in the same repository.

However, compliance with the CWM specification does not guarantee tools from different vendors will integrate well, even when they are "CWM-compliant." The OMG addressed some of these issues by releasing patterns and best practices to correct these problems in a supplementary specification, the Common Warehouse Metamodel (CWM™)) Metadata Interchange Patterns (MIP) Specification. Version 1.0 [2] was released in March 2004.

Foundations

The Common Warehouse Metamodel enables organizations and tool vendors to define and represent their metadata, metadata models and the processes which manipulate them in a common format so that the information can be streamed between tools and accessed programmatically [4].

The basic architecture and key technologies supporting the Common Warehouse Metamodel are shown in Fig. 1, on the next page. Metadata in a variety of formats, and from a variety of sources (Tools, Repositories, Databases, Files, etc.) is defined and represented in UML notation, based on the objects and classes that are defined in the Common Warehouse Metamodel. That representation is persisted in an XML notation that can be streamed to other tools, repositories, databases or files based on the XMI protocol. Finally, MOF is used to provide a broker facility that supports the ability to define and manipulate metamodels programmatically using fine grained CORBA interfaces. Using this architecture, organizations can create a single common repository which stores all the CWM-modeled descriptions of metadata and metamodels.

An example of a CWM description of a table from a relational database is shown below, along with the metadata description of the type of one of its columns (type="22");

```
<CWMRDB:Table xmi.id="_15" name="MyTableName">
 <CWM:Classifier.feature>
  <CWMRDB:Column xmi.id="_16" name="myPrimaryKeyID" precision="4" type="_17"/>
  <CWMRDB:Column xmi.id="_18" name="myForeignKey1ID" precision="4" type="_17"/>
  <CWMRDB:Column xmi.id="_19" name="myForeignKey2ID" precision="4" type="_17"/>
 <CWMRDB:Column xmi.id="_20" name=" myForeignKey3ID" precision="4" type="_17"/>
 <CWMRDB:Column xmi.id="_21" name="description" length="200" type="_22" />
 </CWM:Classifier.feature>
 <CWM:Namespace.ownedElement>
  <CWMRDB:ForeignKey xmi.id="_23" name="unnamed_23" namespace="_15" feature="_19" uniqueKey="_24"/>
 </CWM:Namespace.ownedElement>
</CWMRDB:Table>
<CWMRDB:SQLSimpleType
  xmi.id="_22"
  name="VARCHAR2"
  visibility="public"characterMaximumLength="200"
  characterOctetLength="1" type
  Number="12"/>
```

The CWM specifications consists of a collection of metamodels (defined in UML) that capture all the elements of metadata, metamodels, and their processing that can be expressed when exchanging information between tools. These are what is identified in Fig. 1 as the Common Warehouse Metamodel. It is organized into five layers of abstraction.

Object Model

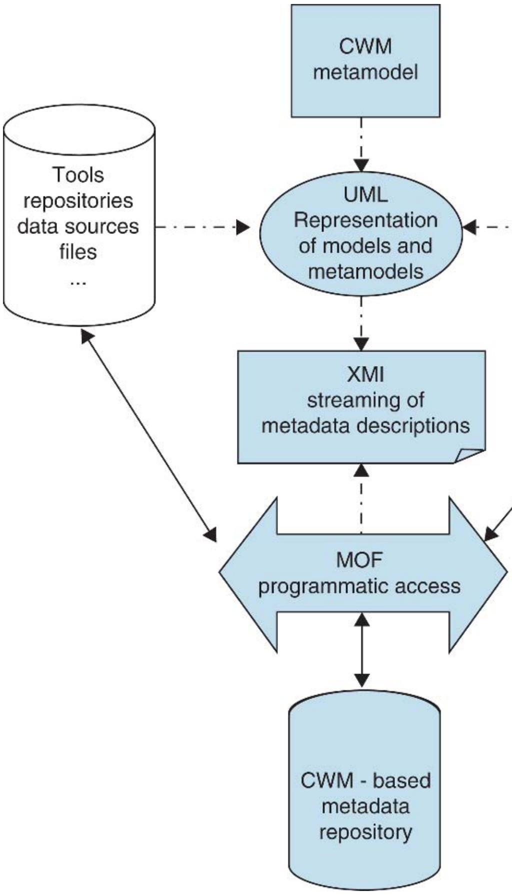
The Object Model layer is the base layer of the Common Warehouse Metamodel. The metamodels in the Object Model layer define the subset of UML that is used for creating and describing the CWM. They are the building blocks used by all the metamodels in the upper layers. This enables CWM to leverage UML's concepts without requiring implementations to support all full of UML's capabilities.

- Core
  The Core metamodel contains basic classes and associations used by all other CWM metamodels like Namespace, Constraint, Attribute, ModeledElement etc.
- Behavioral
  The Behavioral metamodel describe the behavior of CWM types and how that behavior is invoked with classes like Event, Parameter, CallAction etc.
- Relationships
  The Relationships metamodel describes two types of relationships between object within a CWM information store: generalizations (for parent-child relationships) and associations (for links between objects).
- Instance
  The Instance metamodel contains classes to support the inclusion of data instances with the metadata.

Foundation

The metamodels in the Foundation layer contain general model elements that represent concepts and structures shared by other CWM packages. Metamodels in this layer are not necessarily complete, but serve as a common basis that can be shared with other metamodels.

- Data Types
  The DataTypes metamodel supports definition of metamodel constructs that modelers can use to create the specific data types they need with classes like Enumeration, Union, EnumerationLiteral, UnionMember, etc.
- Expression
  The Expressions metamodel supports the definition of expression trees.
- Keys and Indexes
  This metamodel defines the basic concepts of Index, IndexedFeature, UniqueKey, and KeyRelationship.


CWM metamodel
Tools repositories data sources files ...
UML Representation of models and metamodels
Tools repositories data sources files ...
XMI streaming of metadata descriptions
MOF programmatic access
CWM - based metadata repository

Common Warehouse Metamodel. Figure 1. Common warehouse metamodel architecture.

- Type Mapping
  This metamodel is used to support the mapping of types between different tools or data sources.
- Business Information
  The Business Information metamodel supports business-oriented information about model elements with classes like ResponsibleParty, Contact, ResourceLocater etc.
- Software Deployment
  The Software Deployment metamodel contains classes like SoftwareSystem, Component, Site to record how the software in a data warehouse is used.

Resource

The metamodels in the resource layer define the type of data sources and formats that are supported.

- Relational
  The Relational metamodel describes relational data this is accessed through an interface like ODBC, JDBC or the native interface of a relational database.
- Record
  The Record metamodel describes the concept of a record and its structure that can be applied to data records stored in files and databases, or to programming language structured data types.
- Multi-Dimensional
  The Multi-Dimensional metamodel describes a generic representation of a multidimensional database using classes like Schema, Dimension, Member etc.
- XML
  The XML metamodel describes the metadata of XML data with classes like ElementType, Attribute etc.

Analysis

The metamodels in the analysis layer define the types of interaction with metadata that are supported.

- Transformation
  The Transformation metamodel contains classes to describe common transformation metadata used

in Extract, Transform, Load (ETL) tools and processes.

- OLAP
  The OLAP metamodel contains classes to describe common analysis metadata using in OLAP processing with classes like MemberSelection and CubeDeployment.
- Data Mining
  The Data Mining metamodel provide the necessary abstractions to model generic representations of both data mining tasks and models (i.e., mathematical models produced or generated by the execution of data mining algorithms).
- Information Visualization
  The Information Visualization metamodel contains classes like Rendering, RenderedObject, to describe metadata associated with the display of data.
- Business Nomenclature
  The Business Nomenclature metamodel supports the definition of terms used in capturing business requirements with classes like Nomenclature, BusinesDomain, Taxonomy, Glossary, Term, Concept, etc.

Management

The metamodels in the management layer define two aspects of warehouse management.

- Warehouse Process
  The Warehouse Process metamodel supports the documentation of process flows used to execute transformations. A process flow can associate a transformation with a set of events, which will be used to trigger the execution of the transformation.
- Warehouse Operation
  The Warehouse Operation metamodel contains classes for the day-to-day operation and maintenance of the warehouse including scheduled activities, measurements, and change requests.

Key Applications

Vendors of data warehouse tools, conform to the CWM specification to ensure that the metadata in their tools is open and accessible to any CWM compliant tool.

Large organizations leverage the specification in order to be able to manage and maintain there data warehouses in a common metadata repository. By using the CWM specification IT administrators and system integrators can extract and link metadata from different vendors tools.

Oracle, IBM, SA, Informatica, Meta Integration Technology Incorporated are among several industry leaders who have data warehouse tools that are CWM compliant to facilitate interoperability.

Cross-references

▶ Data Warehouse Metadata
▶ Metadata
▶ Metadata Interchange Specification
▶ Metadata Registry, ISO/IEC 11179
▶ Meta Object Facility
▶ Metamodel
▶ Unified Modelling Language
▶ XML Metadata Interchange

Recommended Reading

1. Common Warehouse Model (CWM) Specification, Version 1.1, Object Management Group. Needham, MA, March 2, 2003. http://www.omg.org/technology/documents/formal/cwm.htm.
2. CWM Metadata Interchange Patterns Specification, Version 1.0, Object Management Group. Needham, MA, March 25, 2004.
3. Grossman R.L., Hornick M.F., and Meyer G. Data mining standards initiatives. Commun. ACM., 45(8):59–61, 2002.
4. Poole J., Chang D., Tolbert D., and Mellor D. Common Warehouse Metamodel: An Introduction to the Standard for Data Warehouse Integration. Wiley, 2002.
5. Vaduva A. and Dittrich K.R. Metadata Management for Data Warehousing: Between Vision and Reality. In Proc. Int. Conf. on Database Eng. and Applications, 2001, p. 0129.

Communication Boundary of a DBMS

▶ DBMS Interface

Compact Suffix Tries

▶ Suffix Trees

Comparative Analysis

▶ Comparative Visualization

Comparative Visualization

Hans Hinterberger
ETH Zurich, Zurich, Switzerland

Synonyms

Comparative analysis

Definition

Comparative visualization refers to

1. Methods that support the process of understanding in what way different datasets are similar or different.
2. Methods that allow comparing different characteristics of a given dataset.
3. Methods that allow a comparison of different types of (linked) data graphics.

Key Points

Comparisons of datasets may occur in different ways. Data value to data value: entries of different datasets are compared to one another based on their values; derived quantity to derived quantity: these could be statistical moments of data fields or topological characteristics; methodology to methodology: comparisons of methodologies involve quantifying differences in experiment or simulation parameters; and, if the data are visualized, image to image: such comparisons quantify the differences in the visualizations produced by a given graphical method.

Comparative visualization methods have been developed as an enabling technology for computational and experimental scientists whose ability to collect and generate data far outpaces their ability to analyze and understand such data. The Visualization and Analysis Center for Enabling Technologies (http://www.vacet.org), for example, provides (publicly available) comparative data visualization software [1] for the scientists at the various research labs associated with the US Department of Energy.

Graphical displays that readily allow simultaneous comparisons of several characteristics of multivariate datasets – the parallel coordinate display for example – are sometimes referred to as "comparative graphs."

There is evidence that visual explorations into a dataset's structure are particularly effective when the data can be compared by simultaneously observing different visualizations of the same data. Today's data visualization packages routinely include several different graphic methods to allow such comparisons. To be truly effective, the different graphics should be operationally linked. See [2] and [3] for two examples among others.

Cross-references

- ► Exploratory Data Analysis
- ► Parallel Coordinates

Recommended Reading

1. Bavoil L., Callahan S.P., Crossno P.J., Freire J., Scheidegger C.E., Silva C.T., and Vo H.T. VisTrails: enabling interactive multiple-view visualizations. In Proc. IEEE Visualization, 2005.
2. Schmid C. and Hinterberger H. Comparative multivariate visualization across conceptually different graphic displays. In Proc. 7th Int. Working Conf. on Scientific and Statistical Database Management, 1994.
3. Siirtola H. Combining parallel coordinates with the reorderable matrix. In Proc. Int. Conf. on Coordinated & Multiple Views in Exploratory Visualization, 2003.

Compensating Transactions

Greg Speegle
Baylor University, Waco, TX, USA

Definition

Given a transaction *T*, and its compensating transaction *C*, then for any set of transactions *H* executing concurrently with *T*, the database state *D* resulting from executing *THC* is equivalent to the database state D' resulting from executing *H* alone. Typically, equivalent means both *D* and D' satisfy all database consistency constraints, but *D* and D' do not have to be identical.

A compensating transaction is defined in terms of its corresponding failed transaction, and once started, must be completed. This may involve re-executing the compensating transaction multiple times. The result of compensation is application dependent.

Key Points

A *compensating transaction* is a set of database operations that perform a logical undo of a failed transaction. The goal of the compensating transaction is to restore any database consistency constraints violated by the failed transaction without adversely affecting other concurrent transactions (e.g., *cascading aborts*). However, it does not require the database to be in the exact same state as if the transaction had never executed as with traditional *ACID* properties. A compensating transaction also removes the externalized affects of a failed transaction [2].

Compensating transactions can best be understood by comparing them to traditional atomicity requirements. Under traditional atomicity, either all effects of a transaction are present in the database, or none of them are. Thus, if a transaction T_1 updates a data item and transactions T_2 reads that update, in order to remove all of the effects of T_1, T_2 must also be removed. With compensating transactions, the abort of T_2 is not be required.

Consider an example application of a company manufacturing widgets. The transaction for buying widgets consists of two subtransactions, one to order the widgets and another to pay for them. Since this business is very efficient, as soon as the widgets are ordered, another transaction starts executing the desired widgets. It is possible to compensate for the ordered widgets by simply removing the order from the system. The extra widgets would be produced, but they will be consumed by later orders. Under traditional atomicity requirements, the production transaction would have to be aborted if the buying transaction failed after the order was placed (for example, if the customer could not pay for the widgets).

Compensating transactions are used in long duration transactions called *Sagas* [1], and other applications that require *semantic atomicity*. Unfortunately, compensation is not universally possible – the common example of an externalized event that cannot be undone is the launching of a missile – or may be very complex. Thus, compensating transactions are used when the benefits of avoiding cascading aborts and early externalization of results outweigh the difficulty in determining the compensation.

Cross-references

- ▶ ACID Properties
- ▶ Sagas
- ▶ Semantic Atomicity

Recommended Reading

1. Garcia-Molina H. and Salem K. SAGAS. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 249–259.
2. Korth H.F., Levy E., and Silberschatz A. A formal approach of recovery by compensating transactions. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 95–106.

Computationally Complete Relational Query Languages

Victor Vianu[1], Dirk Van Gucht[2]
[1]University of California-San Diego, La Jolla, CA, USA
[2]Indiana University, Bloomington, IN, USA

Synonyms

Complete query languages; Chandra and Harel complete query languages

Definition

A *relational query language* (or query language) is a set of expressions (or programs). The semantics of a query language defines for each of these expressions a corresponding query which is a generic, computable function from finite relation instances to finite relation instances over fixed schemas. A query language is *computationally complete* (or complete) if it defines all queries.

The genericity condition is a consistency criterion requiring that a query commute with isomorphisms of the database domain. Thus, when applied to isomorphic input relation instances, a query returns isomorphic output relation instances. The concept of genericity is based on the well-accepted idea that the result of a query should be independent of the representation of data in a database, and should treat the elements of the database as uninterpreted objects [4]. The computability condition requires that the query can be effectively computed, in other words it must be *implementable* by a program of a Turing-complete programming language under some suitable encoding of relation instances into objects of that language.

Historical Background

The search for an appropriate notion of "complete" query language began soon after the introduction of the relational model by Codd, with its accompanying query languages relational algebra (RA) and relational calculus (RC) [9]. Initially, RA was proposed as a yardstick for query expressiveness. A language was called by Codd "relationally complete" if it was able to simulate RA [10]. Bancilhon and Paredaens independently proposed the notion of BP-completeness of a language, using an instance-based approach: a language is BP-complete if for every pair of input and output instances satisfying a consistency criterion (The criterion requires that every automorphism of the input be also an automorphism of the output) there exists a query in the language mapping the input instance to the output instance [5,13]. The notion of genericity was first articulated in the database context by Aho and Ullman [4], although its roots can already be found in the consistency criterion used in the definition of BP-completeness, and an idea similar to genericity underlies Tarski's concept of "logical notion," introduced in a series of lectures in the mid 1960s [15]. The modern notion of computationally complete query language is due to Chandra and Harel, who also defined the first such language, QL [7].

Foundations

Codd introduced the relational model and its query languages, relational algebra (RA) and relational calculus (RC). These query languages are equivalent, i.e., they define the same set of queries. For example, assuming that **R** and **S** are relation schemas both of arity 2, then the RA-expression $\pi_{1,4}(\sigma_{2=3}(\mathbf{R} \times \mathbf{S})) \cup (\mathbf{R} - \mathbf{S})$, and the RC-expression $\{(x, y) \mid \exists z : (\mathbf{R}(x, z) \wedge \mathbf{S}(z, y)) \vee (\mathbf{R}(x, y) \wedge \neg \mathbf{S}(x, y))\}$ define the same computable query Q which maps each relation instance R over **R** and each relation instance S over **S** to their join unioned with their set difference. Notice that Q is also generic: consider, for example, the input relation instances

$$R_1 = \boxed{\begin{array}{cc} a & b \\ a & c \end{array}} \quad S_1 = \boxed{\begin{array}{cc} a & b \\ b & d \end{array}}$$

and the isomorphic input relation instances

$$R_2 = \boxed{\begin{array}{cc} e & f \\ e & g \end{array}} \quad S_2 = \boxed{\begin{array}{cc} e & f \\ f & h \end{array}}$$

then $Q(R_1, S_1)$ and $Q(R_2, S_2)$ are the isomorphic output relation instances $\boxed{a \quad c}$ and $\boxed{e \quad g}$, respectively. (As a caveat to genericity, consider the query C defined by the RA-expression $\sigma_{1=\mathbf{a}}(\mathbf{R})$, where **a** is some *constant* interpreted as a. Then, $C(R_1) = R_1$, but $C(R_2) = \emptyset$. The difficulty is that, though R_1 and R_2 are isomorphic, they are not isomorphic by an isomorphism that *fixes* a. If however, the value of e in R_2 is replaced by a, then $C(R_2^{e \leftarrow a}) = R_2^{e \leftarrow a}$. This suggests that when constants are involved, genericity should be modified to isomorphisms that fix these constants. In the literature, this is referred to as **C**-genericity.)

The development of the relational model led to the introduction of the query language SQL which has, at its logical core, a sub-language pure-SQL, that is equivalent with RA and RC. For example, in pure-SQL, the query Q can be defined by the expression (Here A, B, C, and D are attribute names referring to the first and second columns of **R**, and the first and second columns of **S**, respectively.)

```
(SELECT R.A, S.D AS B FROM R, S WHERE R.B = S.C)
  UNION
  (
  (SELECT R.A, R.B FROM R)
  EXCEPT
  (SELECT S.C AS A, S.D AS B FROM S)
  );
```

A natural question is now "Are RA, RC, and pure-SQL complete query languages?" The answer is no. To this end, consider the following three queries, which are easily seen to be generic and computable:

1. TC maps each binary relation instance R to its transitive closure R^*.
2. EVEN maps each unary relation instance R to $\{()\}$ (true) if $|R|$ is even, and to $\emptyset$ (false), otherwise.
3. EVEN$^<$ maps each pair of a unary relation instance R and a binary relation instance O to EVEN(R) if O defines an ordering on $dom(R)$, and is undefined otherwise. (Here, $dom(R)$ denotes the set of values that occur in the tuples of R.)

It turns out that none of the above queries is expressible in RA (or RC, or pure-SQL). Consider (1). Fagin showed in [11] that the TC query cannot be defined by any RC expression (the result was later re-proven for RA by Aho and Ullman [4]). Intuitively, the difficulty in computing TC is the following. For each $i \geq 0$, there exists an RA-expression $\mathbf{E}^i$ that defines the pairs of elements in R at distance i in the directed graph represented by R. However, there does not exist a *single* RA-expression that defines the union of all these pairs, as needed for computing TC. A solution to this problem is to augment RA with an *iteration* construct. This led Chandra and Harel to define the language While (initially introduced as RQ in [8] and LE in [6]). The language uses, in addition to database relations, typed relational variables (of fixed arity) initialized to ∅, to which RA expressions can be assigned. Iteration is provided by a construct "while change to **R** do ⟨program⟩ od" whose semantics is to iterate ⟨*program*⟩ as long as the value of the relational variable **R** changes. For example, the following While program defines TC:

```
$\mathbf{TC} := \mathbf{R}$;
while change to $\mathbf{TC}$
    do $\mathbf{TC} := \mathbf{TC} \cup \pi_{1,4}(\sigma_{3=4}(\mathbf{TC} \times \mathbf{R}))$ od.
```

Here **TC** is a binary relation variable (initialized to ∅). The program first assigns R to **TC**, then loops as long as **TC** changes. Upon termination, this value is R^*. One might hope that While is a complete query language. However, this is not the case. Indeed, even though it is easy to write a While program that defines the $\text{EVEN}^<$ query, Chandra showed that no such program can define the simple linear-time computable EVEN query [6]. Thus, While is not computationally complete. Intuitively, While programs do not have the ability to compute with natural numbers, unless such computations can be simulated by utilizing an ordering on the elements of its input. With such orderings available, While can define precisely the PSPACE-computable queries [17]. The PSPACE upper bound (that holds with or without order) is a consequence of the fact that a program's finite set of variables are of fixed arity and can only hold relation instances built from the elements of its inputs.

To overcome these problems, it appears natural to embed RA into a language that can perform arbitrarily powerful computations. This is in the spirit of "embedded SQL" languages, in which a computationally complete programming language such as C or Java accesses the database using SQL queries. A language called LC (for *Looping+Counters*), abstracting the "embedded SQL" paradigm, was introduced by Chandra [6] (with a variant called WhileN later defined in [1]). The language LC extends While by allowing integer variables (initialized to zero) that can be incremented or decremented. Iteration for computation on integers is provided by an additional while loop of the form "while i > 0 do ⟨program⟩ od" which causes ⟨program⟩ to iterate as long as the value of the integer variable i is positive. For example, consider the following program in an LC-like syntax: (The "if-else" statement is a macro that can be easily written using just the "while-change" construct.)

```
$\mathbf{TC} := \mathbf{R}$;
$n := 0$;
while change to $\mathbf{TC}$
    do
      $n := n + 1$;
      $\mathbf{TC} := \mathbf{TC} \cup \pi_{1,4}(\sigma_{2=3}(\mathbf{TC} \times \mathbf{R}))$
    od;
if $n \leq 1$ return $\{()\}$ else return $\emptyset$
```

At the end of the computation, n contains the number of times the body of the *while* loop was executed. Thus, if R is non-empty, the final value of n is the diameter of the directed graph represented by R (here the diameter means the maximum finite distance between two nodes). The program returns {()} (true) if n ≤ 1 and ∅ (false) otherwise. Note that, since LC is computationally complete on the integers, the condition "n ≤ 1" could be replaced by *any* computable property of n. Thus, LC can test any computable property of the diameter of R. Clearly, LC can define strictly more queries than While, since all computable functions on natural numbers can be defined and used. This leads to the next question: "Is LC a complete query language?" Again, the answer is no. Indeed, Chandra showed that the EVEN query can still not be defined in this language. This time, the difficulty stems from the fact that, even though the values of the relation and natural number variables can depend on each other, LC programs lack the ability to *explicitly coerce* (encode) these values into each other. However, when input relation instances are accompanied by an ordering of the domain, such coercions can be simulated, and Abiteboul and Vianu showed that then LC *is* complete [1].

To obtain a complete language without order, several solutions are possible. A brute force approach to the coercion problem is to augment LC with an encoding function *enc* mapping relations to integers, and a decoding function *dec* returning query answers from their encodings and the original input database. Since LC is computationally complete on integers, it can compute the integer encoding of the answer from that of the input. Although this theoretically produces a complete language, manipulating integer encodings of databases is not a satisfying solution, so further discussion of this approach is omitted. Instead, two more appealing alternatives are described, that both go back to While as a starting point and extend it in different ways. Recall that While is limited to PSPACE computations, as a consequence of two facts: (i) only relations of fixed arity are used, and (ii) the relations can be populated by tuples using only elements occurring in the input. The first approach, proposed by Chandra and Harel, breaks the PSPACE space barrier by relaxing (i) it allows untyped relational variables, whose arity can grow arbitrarily. The other approach, introduced by Abiteboul and Vianu, relaxes (ii) it keeps typed relational variables but allows the introduction of new domain values in the course of the computation. These languages are described next.

The complete language proposed by Chandra and Harel was called QL [6]. Up to minor syntactic differences, QL is very similar to While, only with untyped relation variables. Consider the following QL program (For simplicity, the syntax used here differs slightly from the original QL syntax.) which is strikingly similar to the LC program shown above:

```
$\mathbf{TC} := \mathbf{R}$;
$\mathbf{ONE} := \pi_1(\mathbf{R}) \cup \pi_2(\mathbf{R})$;
$\mathbf{N} := \{()\}$;
while change to $\mathbf{TC}$
    do
      $\mathbf{N} := \mathbf{N} \times \mathbf{ONE}$;
      $\mathbf{TC} := \mathbf{TC} \cup \pi_{1,4}(\sigma_{2=3}(\mathbf{TC} \times \mathbf{R}))$
    od;
if $(\mathbf{N} = \{()\})$ or $(\mathbf{N} = \mathbf{ONE})$ then return $\{()\}$ else
return $\emptyset$.
```

In this program, **R** is a binary relation input variable, and **TC**, **ONE**, and **N** are relation variables. Note that the arities of **TC** and **ONE** remain fixed throughout the execution of the program, while the arity of **N** changes. Integers can be easily simulated using the arity of relations. Thus, starting with relation instance R, the variable **ONE** is initialized to dom(R), which plays the role of the natural number 1. The variable **N** plays the role of a natural number variable n. The statement **N** :=$\{()\}$ corresponds the statement n = 0, and the statement **N** := **N** ×**ONE** serves to increment n by 1; notice that the × operator plays the role of the addition operator + over natural numbers, and the decrement operator can be simulated by projection. Similarly to the earlier LC program, the final arity of **N** is the diameter of the directed graph represented by **R**. The final "if" statement compares **N** to $\{()\}$ or **ONE**, and the program returns $\{()\}$ (true) if the diameter of **R** is at most 1, and $\emptyset$ (false) otherwise. Observe therefore that this QL program defines the same query as its corresponding LC program.

The above example illustrates how arithmetic on natural numbers can be simulated in QL. So far, this allows simulating LC. Recall that LC is not complete, but becomes so if an ordering of the domain is provided. QL is however complete even if an ordering is not provided, because it can construct its own orderings! Indeed, such orderings of the domain can simply be constructed in QL by building one relation whose arity equals the size of the input domain. In such a relation, any tuple that does not contain repeated elements provides a successor relation on the domain, which in turns induces an ordering. The completeness of QL now follows from the completeness of LC on ordered domains. Thus, QL can express all computable queries. But is everything it expresses a query? It is easy to see that all mappings defined by QL programs are computable and generic. The difficulty is to guarantee that a QL program always produces answers of the desired arity. In fact, this property is undecidable for QL programs. Fortunately, there is an effective syntactic restriction guaranteeing that QL programs are "well behaved," i.e., always produce answers of fixed arity. Moreover, all computable queries can be expressed by QL programs satisfying the syntactic restriction.

The language WhileNew, introduced by Abiteboul and Vianu in [3], extends While by allowing the creation of new values throughout the computation. This is achieved by an instruction of the form **S** := new(**R**), where **R** and **S** are relational variables and arity(**S**) = *arity*(**R**) + 1. The semantics is the following. Given a relation instance R over **R**, the relation instance S over **S** is obtained by extending each tuple of R by one distinct new value not occurring in the input, the current state, or in the program. For example, if R is the relation instance in Fig. 1 then S is of the form shown in the same figure. The values α,β,γ are distinct new values. Note that the new construct is, strictly speaking, nondeterministic. Indeed, the new values are arbitrary, so several outcomes are possible depending on the choice of values. However, the different outcomes differ *only* in the choice of new values.

The ability to successively introduce new values throughout the computation easily allows simulating integers and arithmetic, yielding the power of LC. Moreover, orderings of the input domain can also be constructed and marked by distinct new values. Since LC is complete on ordered domains, this shows that

WhileNew can express all computable queries. As in the case of QL, one must ask whether *all* mappings expressed by WhileNew are in fact queries according to the definition. The difficulty arises from the presence of new values. Indeed, if new values may appear in the outputs of a WhileNew program, the mapping it defines is non-deterministic. Moreover, it is undecidable whether a WhileNew program never contains new values in its output. The solution to this problem is similar to the one for QL: one can impose a syntactic restriction on WhileNew programs guaranteeing that no new value appears in their answers. All generic computable queries can be expressed by WhileNew programs satisfying the syntactic restriction.

As an aside, suppose the definition of query is extended by allowing new values in query answers. This arises naturally in some contexts such as object-oriented databases, where outputs to queries may contain new objects with their own fresh identifiers. One might hope that WhileNew remains complete for this extension. Surprisingly, it was shown by Abiteboul and Kanellakis that the answer is negative [2]. Indeed, WhileNew cannot express the query containing the input/output pair shown in Fig. 2, where $\psi_0,\ldots,\psi_3$ are new values. As shown by Abiteboul and Kanellakis, completeness with new values can be achieved by adding to WhileNew a construct called *duplicate elimination*. This is however a rather complex construct, that encapsulates a test for isomorphism of relations. The search for a language using more natural primitives and complete for queries with new values in the answer remains open.

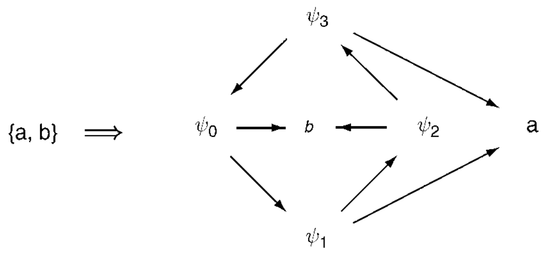
R =

a	b
a	c
c	a

S =

a	b	α
a	c	β
c	a	γ

Computationally Complete Relational Query Languages. Figure 1. An application of new. Here, $S = \text{new}(R)$.



Computationally Complete Relational Query Languages. Figure 2. A query with new values not expressible in WhileNew.

The relational model, though very simple, is not always the most natural model for databases in certain application domains. In the late 1980s and early 1990s database researchers considered object-oriented databases as an alternative to the relational model, and a significant amount of theory was developed around the model and its query languages, including the completeness of object-oriented query languages (see [1,16]). Finally, consistency notions other than genericity can be considered for specialized application domains. This was done, for example, in the context of spatial databases [12,14].

Key Applications

The theoretical query languages discussed here are closely related to various practical languages. Thus, RA and RC correspond to pure-SQL. The language While corresponds to wrapping programming constructs such as loops around SQL, as done in PL/SQL (Oracle); assignment statements can be implemented using SQL insert and delete operations.

The language LC (or WhileN) can again be simulated in PL/SQL augmented with natural number variables (with no coercion allowed). Another approach is to embed SQL in a programming language such as C or Java. In such languages, relational variables must be statically defined and so have fixed arity. One significant feature of the embedded SQL languages that sets them apart from LC is that they allow accessing tuples in relations one at a time, using looping over cursors. In particular, this allows coercing the entire database into a native data structure, and yields computational completeness. However, there is a catch: programs using cursors are generally non-deterministic, in the sense that running the same program on the same database content may yield different results. Unfortunately, it is undecidable whether a given embedded SQL program is deterministic, and no natural syntactic restriction is known that ensures determinism while preserving completeness. Thus, completeness is achieved at the cost of losing the guarantee of determinism.

The computationally complete language QL can be simulated in Dynamic SQL. In this language one can dynamically create relation variables whose schemas depend on the data in the database. This can be used to support untyped relational variables. The language WhileNew allowing the introduction of new domain values is akin to object-oriented languages that allow the creation of new object identifiers.

Future Directions

The database area is undergoing tremendous expansion and diversification under the impetus of the Web and a host of specialized applications. Consequently, new structures and objects have to be modeled and manipulated. For example, in biological and scientific applications, sequences and matrices occur prominently; in XML databases, text and tree-structured documents are the main objects. This has led to new database models and query languages. Their formal foundations are fast developing, but are not yet as mature as for the relational data model. Notions of computationally complete languages for the new models are still emerging, and are likely to build upon the theory developed for relational databases.

Cross-references

▶ BP-completeness
▶ Constraint Query Languages
▶ Data Models with Nested Collections and Classes
▶ Ehrenfeucht-Fraïssé Games
▶ Expressive Power of Query Languages
▶ Object Data Models
▶ Query Language
▶ Relational Calculus
▶ Relational Model
▶ Semantic Web Query Languages
▶ Semi-Structured Query Languages
▶ SQL
▶ XML
▶ XPath/XQuery

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases. Addison-Wesley, Reading, MA, 1995.
2. Abiteboul S. and Kanellakis P.C. Object identity as a query language primitive. J. ACM, 45(5):798–842, 1998.
3. Abiteboul S. and Vianu V. Procedural languages for database queries and updates. J. Comput. Syst. Sci., 41(2):181–229, 1990.
4. Aho A.V. and Ullman J.D. Universality of data retrieval languages. In Proc. 6th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages, 1979, pp. 110–120.
5. Bancilhon F. On the completeness of query languages for relational data bases. In Proc. 7th Symp. on the Mathematical Foundations of Computer Science, 1978, pp. 112–123.
6. Chandra A. Programming primitives for database languages. In Proc. 8th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages, 1981, pp. 50–62.
7. Chandra A. and Harel D. Computable queries for relational data bases. J. Comput. Syst. Sci., 21(2):156–178, 1980.
8. Chandra A. and Harel D. Structure and complexity of relational queries. J. Comput. Syst. Sci., 25:99–128, 1982.
9. Codd E. A relational model for large shared databanks. Commun. ACM, 13(6):377–387, 1970.
10. Codd E. Relational completeness of data base sublanguages. In Data Base Systems, R. Rustin (ed.). Prentice-Hall, Englewood, Cliffs, NJ, 1972, pp. 65–98.
11. Fagin R. Monadic generalized spectra. Zeitschrift für Math. Logik Grundlagen d. Math, 2189–96, 1975.
12. Gyssens M., Van den Bussche J., and Van Gucht D. Complete geometric query languages. J. Comput. Syst. Sci., 58(3):483–511, 1999.
13. Paredaens J. On the expressive power of the relational algebra. Inf. Process. Lett., 7(2):107–111, 1978.
14. Paredaens J. Spatial databases, a new frontier. In Proc. 5th Int. Conf. on Database Theory. 1995, pp. 14–32.
15. Tarski A. What are logical notions? History Phil. Logic, 7:154, 1986. J. Corcoran (ed.).
16. Van den Bussche J., Van Gucht D., Andries M., and Gyssens M. On the completeness of object-creating database transformation languages. J. ACM, 44(2):272–319, 1997.
17. Vardi M.Y. The complexity of relational query languages. In Proc. 14th Annual ACM Symp. on Theory of Computing, 1982, pp. 137–146.

Complex Event

Opher Etzion
IBM Research Lab in Haifa, Haifa, Israel

Synonyms

Composite event; Derived event

Definition

A complex event is an event derived from a collection of events by either aggregation or derivation function [3].

Key Points

A complex event [2], [1] is a derived event; it can be derived by various means:

1. Explicit concatenation of a collection of events,
   - Example: Create an event that contains all the events that are related to the 2008 USA presidential elections.

2. Derivation of an aggregated value from a collection of events from the same type.
   - Example: Create an event that contains the average, maximal and minimal value of a certain stock during a single trade day.
3. Derivation [4] of an event as a function of other events that is a result of event pattern detection.
   - Example: Whenever a sequence of three complain-events from the same customer occurs within a single week, create an event "angry customer" with the customer-id.

     Note that this event may or may not contain the raw complain events.

Cross-references

► Complex Event Processing

► Event Pattern Detection

Recommended Reading

1. Ericsson A.M., Pettersson P., Berndtsson M., Seiriö M. Seamless formal verification of complex event processing applications. In Proc. Inaugural Int. Conf. Distributed Event-Based Systems, 2007, pp. 50–61.
2. Luckham D. The Power of Events. Addison-Wesley, 2002.
3. Luckham D., and Schulte R. (eds.) - EPTS Event Processing Glossary version 1.1. http://complexevents.com/?p=409.
4. Zimmer D., and Unland R. On the Semantics of Complex Events in Active Database Management Systems. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 392–399.

Complex Event Processing

Opher Etzion
IBM Research Lab in Haifa, Haifa, Israel

Synonyms

Event processing; Event stream processing

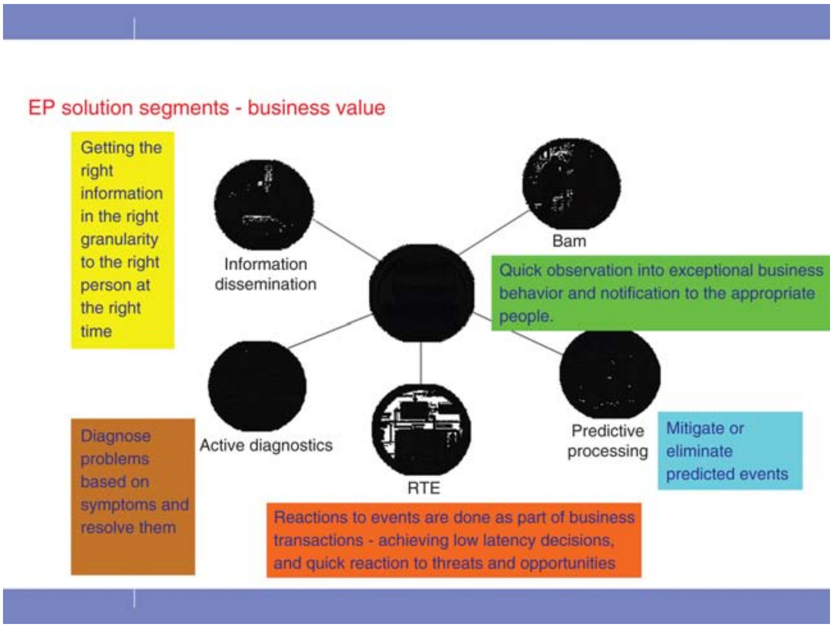
Definition

Complex event processing deals with various types of processing complex events.

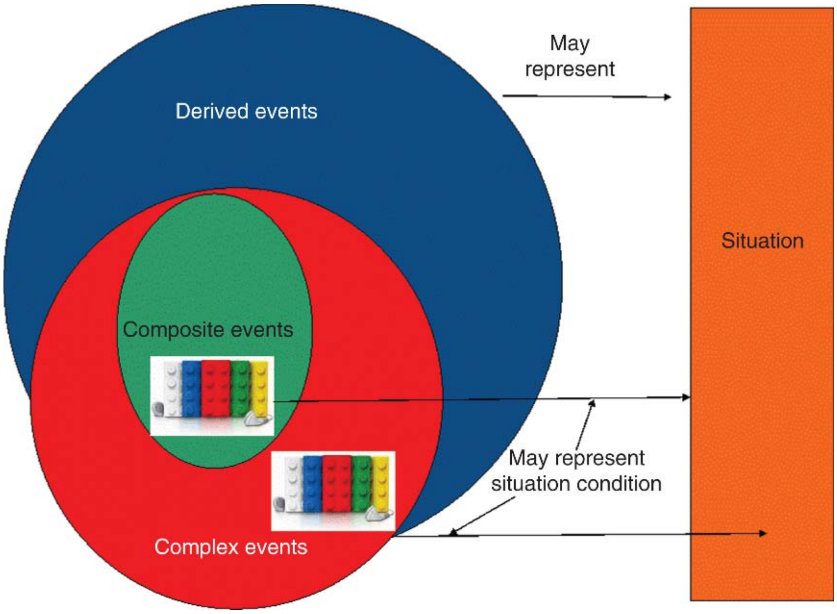
Key Points

Figure 1 shows that the different applications of the CEP technology are not monolithic, and can be classified into five different solution segments, which differ in their motivation, from the user's perspective, they are:

- *RTE (Real-Time Enterprise)*: The processing should affect business processes while they are still running. For example, stop an instance of a workflow that deals with trading a certain stock, if the trade request has been withdrawn.
- *Active Diagnostics*: Finding the root-cause of a problem based on events that are symptoms.
- *Information Dissemination*: A personalized subscription that enables subscriptions in lower granularity, where the subscription does not match the published


EP solution segments - business value
Getting the right information in the right granularity to the right person at the right time
Information dissemination
Bam
Quick observation into exceptional business behavior and notification to the appropriate people.
Active diagnostics
RTE
Predictive processing
Mitigate or eliminate predicted events
Diagnose problems based on symptoms and resolve them
Reactions to events are done as part of business transactions - achieving low latency decisions, and quick reaction to threats and opportunities

Complex Event Processing. Figure 1. Various CEP solution semgents.



Complex Event Processing. Figure 2. Relationships among major complex event processing terms.

event, but a combination of event. For example, notify me when IBM stock has gone up 2% within 1 hour.

- *BAM (Business Activity Management)*: Monitor for exceptional behavior, by defining KPI (Key Performance indicators) and other exceptional behavioral constraints. For example, the delivery has not been shipped by the deadline.
- *Prediction*: Mitigate or eliminate future predicted events.

Figure 2 shows the relations among the different terms around complex event processing. Complex event may be a derived event, but the overlapping among them is partial, the complex event processing is materialized by detecting patterns which may correspond to situations (cases that require action). The exact definitions of terms can be found in the EPTS glossary.

Cross-references

► Complex Event
► Event Pattern Detection

Recommended Reading

1. Etzion O. Event processing, architecture and patterns, Tutorial. In Proc. 2nd Int. Conf. Distributed Event-Based Systems, 2008.
2. Event processing glossary. Available at: http://www.epts.com
3. Luckham D. The Power of Events. Addison-Wesley, Reading, MA, 2002.
4. Sharon G. and Etzion O. Event processing networks – model and implementation. IBM Syst. J., 47(2):321–334, 2008.
5. Zimmer D. and Unland R. On the semantics of complex events in active database management systems. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 392–399.

Complex Event Processing (CEP)

► Event and Pattern Detection over Streams
► Stream Processing

Compliance

► Storage Security

Component Abstraction

► Abstraction

Composed Services and WS-BPEL

Francisco Curbera
IBM Research, T.J. Watson Research Center,
Hawthorne, NY, USA

Synonyms

Service orchestration; Service choreography; WS-BPEL; Web services business process execution language

Definition

Service oriented architectures (SOAs) are models of distributed software components where business or scientific functions are delivered by a network of distributed services. Services can be classified into atomic services and composed services, based on how they are created and run. Atomic services are those that do not depend on other services for their operation, and are typically built on technologies native to a specific platform, such as COBOL or Enterprise Java Beans. Composed services are created by composing the function provided by one or more external services into a new service. There is no restriction implied as to the programming model used to create composed services; platform specific programming models such as Enterprise Java Beans or C# have been extended to support the creation of composed services. In addition to that approach, service centric programming models have been defined to support the development of composed services. Foremost among those is the Web Services Execution Language for Web Services (WS-BPEL or BPEL), a service composition language based on the workflow programming model.

Historical Background

The service oriented architecture model [1] evolved early on to include aspects of a component oriented architecture, based on the similarity between the reuse of services and of traditional software components. Component oriented software development has been a constant reference point in the development of SOA. Service composition also draws from the experience of workflow programming and business process reengineering [5] as it developed throughout the 1990s. The workflow programming model relies on a two level programming paradigm in which applications are combined to accomplish a business goal by means of a graph oriented programming approach. The term "two level programming" derives from the differentiation between the programming task whose goal is to create the individual applications and the creation of the workflow control and data graphs whose goal is the use of those applications to achieve a particular business goal. Because of the close alignment between two level programming and SOA service reuse through composition, workflow programming became a second reference point in the early development of the SOA model, leading to process oriented service composition models and eventually the WS-BPEL language [10].

Foundations

Services and Components

The component oriented software development model (COSD) assumes that software applications and systems can be more efficiently developed and managed when created through the aggregation (composition) of a set of software building blocks (components) independently produced by third parties. Szyperski [8] restricts components to binary code and associated resources, to differentiate components from other software abstractions. More importantly, components are units of deployment (can be independently deployed) and composition (can be independently integrated in composed applications). Components are endowed with well defined interfaces through which they interact with other components.

Service oriented architectures build on this same paradigm. Services in SOA are nothing but the interfaces of SOA components. SOAs add an important new perspective to the usual COSD approach. Components in SOAs are not only units of independent deployment, but also units of independent ownership and management by third parties. The implication is that when creating a composite application two related perspectives are possible. In a traditional COSD approach ("component composition"), a single party deploys and maintains control and management rights over all components. The subjects of the composition task are individual software (binary) components. At runtime they are managed as parts of the composed application to which they belong. By contrast, in a service oriented approach, the subjects of the composition are the services exposed by software components, which are in principle deployed and managed independently of the composite application itself. The term "service composition" is used to refer to a new SOA application created by composing services. The services exposed by a service composition are called "composite services."

The traditional COSD approach is followed by the Service Component Architecture (SCA, see [2]), which defines a straightforward model for the deployment and composition of service oriented components. In SCA, components are "wired" to each other by connecting their interfaces, to create new SCA applications or "composites." SCA composites are component compositions, and they can provide services by exposing one or more service interfaces. Composites are deployed by deploying and configuring their constituent component's binary implementations. SCA composites are potentially components themselves, thus supporting a recursive composition model.

Service Composition Models

Focusing on both the development time and runtime natures of a service composition, it is possible to distinguish two types of service composition strategies.

"Localized" compositions specify the composite's application internal operation and its dependencies on a set of external services, including in particular the expected behavior of those services. Localized compositions are concerned with applications executing in a single logical location of control. The composition relies on a single logical node where the application logic is executed and from which the interactions with the composed services are controlled. The services used by the composite are naturally executed in separate nodes, but their operation is not the concern of the composition except for its externally visible behavior. The term "service orchestration" is sometimes used to refer to localized service compositions. WS-BPEL is the model for this type of service composition.

"Distributed" service compositions specify the behavior of a set of independent SOA applications interacting through service interfaces as part of a distributed composite application. In its purest form, a distributed composition doesn't specify the internal behavior of any of the participating applications, but only the interactions between the participating services. The execution of the composite is assumed to be distributed among a set of independent nodes, which coordinate their operation according to the composition's specification. The Web Services Choreography Description Language [7] is the prime example of this type of composition. It is also important to mention in this category of service composition the use of SCA as a service wiring specification, in particular in the context of Enterprise Service Bus runtimes; this usage of SCA is discussed later in this entry, under "Key Applications."

In spite of the difference in approach, all service composition models share a set of common characteristics. Both types of service composition require the specification of the expected behavior from the composed services. Expected behavior refers here to the message exchange sequence between the services, which is frequently referred to as a "service conversation." A service conversation is the realization of a business protocol between two parties, over a SOA infrastructure, and is the basis for business level interoperability between services.

Together with a particular service conversation, the relationship between two services is characterized by a particular interaction pattern. Several of these patterns are possible. In a hierarchical organization, the lifecycle of one service is subordinated to another's, such that the later is responsible for the creation and termination of runtime instances of the former. This is the relationship between a process and a sub-process. In a peer-to-peer interaction, both services maintain independent lifecycles and interact as peers. Again, both relationships are possible in both localized and distributed compositions.

It is clear that maintaining the appropriate level of distributed coordination between participating services at runtime is a non-trivial problem. Hierarchical lifecycle dependencies as well as distributed transactional behavior must rely on coordination middleware to ensure agreement on the outcome and synchronization of the interaction. These "coordinated behaviors" of distributed systems are supported in the Web services specification stack by the Web Services Coordination specification, which defines a framework for the creation of distributed protocols [11].

Finally, "recursive composition" is a common characteristic of all service composition models, and the basis for the creation of composite services through composition. In a recursive model, the service composition becomes a service itself, available for further invocation and composition. WS-BPEL, SCA and WS-CDL all allow the creation of new services through composition. A survey of different methodologies for development of service compositions can be found in [4].

Workflow Oriented Composition in WS-BPEL

WS-BPEL is the model and industry standard for localized service composition. WS-BPEL is also the prototype for workflow (or process) oriented composition, a form of service composition that follows the workflow programming model. The WS-BPEL specification adapts the workflow two level programming model to a service centric environment.

The WS-BPEL language consists of two main parts, one dealing with the representation of service interaction requirements and a second one dealing with the specification of the control and data logic of the workflow.

Service Interaction in Processes A WS-BPEL composition is called a "process model." Service interaction requirements are defined in terms of "abstract" Web services interfaces, that is, XML interface definitions containing no references to service access details such as interaction protocol or endpoint address. Abstract interfaces are defined using the Web Services Description Language (WSDL, see [11]); abstract interfaces are called portTypes in WSDL 1.1. The interaction between the process model and each of the composed services requires in the general case two such abstract interfaces, one used by the process to call the service and one for the service to call back on the process. This pair of interfaces characterizes the interaction and is called a "partner link" in WS-BPEL. The actual ways in which these interfaces are exercised is defined by the business logic of the process definition, which is described later in this section.

WS-BPEL service composition takes place at the interface type level ("service types composition") instead at the instance level ("service instance composition"): abstract service interfaces are referenced by the process, instead of actually deployed services. The goal of this approach to composition is to expand the reusability of WS-BPEL processes, since it allows the same process composition to be used with different services and using different access protocols, as long as the correct portTypes are supported. One particular consequence of this approach is the absence of quality of service specifications in WS-BPEL processes.

A second aspect of the interaction between services is the identification of dynamic correlation data fields for process instance identification and message routing. A WS-BPEL process specifies a model for the execution of individual "process instances." A new process instance is started any time a "start" message (as defined by the process model business logic) is received by the process engine. At any point in time, many instances of the each process model are executing the same process engine. In traditional workflow infrastructures, each instance is identified by a unique identifier, which is carried by all messages sent to the process instance. By carrying this identifier, messages can be routed to the correct instance.

WS-BPEL takes a different approach to the routing problem. The correlation between messages and process instances is done using business information fields, grouped in data sets that the WS-BPEL specification calls "correlation sets." A correlation set is a group of message fields that collectively identify the executing process instance. Correlation fields are identified using XPath expressions to point to individual data fields within the messages received by the process, allowing content based message routing to instances. Correlation information is also specified in outgoing messages, when the values of these fields need to be communicated to an external service. The main benefit of the WS-BPEL correlation approach is to replace the use of platform specific artifacts by business meaningful information for the purpose of interoperable message routing and transaction identification. Correlation values are a key element of any message oriented business protocol.

Specification of Business Logic in Process Compositions Business logic (the control and data graphs of a workflow) is specified in WS-BPEL by means of a set of "atomic" and "structured activities." Atomic activities represent individual steps in the computation of the process graph, and they stand for external service interaction steps (calling or being called by a service), or data manipulation primitives (assignment of data fields). Atomic activities are combined according to a particular sequence of execution using structured activities. WS-BPEL provides structured activities for sequential, parallel as well as conditional and iterative execution. These activities allow the creation of "structured" process graphs, those in which the programming style is similar to that of structured programming languages (albeit with intrinsic parallel capabilities).

WS-BPEL also supports a graph oriented process modeling approach, where atomic activities are combined as nodes of an explicit graph. Edges of the graph are called "control links," and represent explicit transfer of control between a source and a target activity (as opposed to the implicit dependency defined by a sequence activity for example). The execution of the control link graph follows the "dead path elimination" operational semantics where those graph branches not followed in the execution of a process instance (typically because of conditional statements are not satisfied) are transitively eliminated to ensure that every activity in the graph is eventually executed or marked as part of an eliminated "dead path." Dead path elimination and the rule that prevents cyclic control graphs ensure the termination of every valid WS-BPEL graph's execution. The graph and structured styles are however not strictly separated and can be combined in a rich but at times challenging authoring style. Dead path elimination semantics are embedded in the structured execution model through exception handling, see [3].

Data flow in WS-BPEL is not explicitly modeled, but implied by the use of a set of process variables as inputs and outputs of atomic activities. Data variables in WS-BPEL contain XML data, which is typed according to the XML Schema language.

WS-BPEL contains error handling and recovery primitives to support business interaction in loosely coupled environment. Error handling is supported through the introduction of "fault handlers" which are charged with recovering from errors generated in the course of process execution: faults generated in the course of a service call, errors explicitly raised by the process when certain business conditions are detected, and system generated faults raised when the underlying execution runtime cannot comply with the requirements of process logic (such as errors accessing message data and other error conditions).

Fault handlers are associated with sections of the process called "scopes," the complete process being the outermost scope. Faults originated within a scope are handled by the fault handlers attached to the scope. In the course of recovering from a fault, it may be determined that a particular action already competed must be "undone." A "compensation handler" may be associated with that action (atomic activity) or with a collection of actions (a scope) to indicate the steps required to "undo" that activity or scope, and would then be executed by the fault handler. A compensation handler defines a set of business level actions that provide a logical reversal (backward execution path) of the action in question. Compensation recovery represents an alternative to automatic rollback, and is necessary in service oriented scenarios where loosely coupled services cannot participate in transactions with atomic semantics and automatic rollback recovery. Business transaction in SOA environments require looser transactional semantics (see [7] for details) where recovery is typically specified at the application level.

Key Applications

Current practice of service composition is closely tied to two types of SOA runtimes available in the industry: SOA-enabled business process management (BPM) platforms and enterprise service bus (ESB) infrastructures.

WS-BPEL service composition on BPM platforms represents is by far the most extensive application of the service composition model in enterprise computing. Its success is due to two factors: the increased focus of enterprises on end-to-end business automation, and the fact that service composition builds on the well known business process integration paradigm.

There are important differences between platforms for process oriented service composition and traditional BPM, appearing in two main areas: the reach of the process integration capabilities and the approach to process management. Service oriented process composition adds a new perspective typically absent from traditional BPM platforms, namely, a uniform model for representing internal and external business function based on the service paradigm. The result is the ability to seamlessly incorporate cross departmental and cross organizational services to capture and automate end-to-end business requirements. End-to-end business automation is the main driver of SOA adoption by businesses today.

The management capabilities of the BPM platform are also significantly affected by the service oriented model. Full management is now limited to the process itself and any services deployed locally within the BPM platform. Unlike in traditional BPM, the ability to manage other services is limited (or missing altogether) because they are typically run and managed by different parties (other departmental organizations or different enterprises). BPM platforms supporting service composition must rely on service management standards (such as the Web Services Distributed Management specification [7]) and service level agreements (such as the Web Services Agreement specification, [7]) to provide visibility and limited control over the execution of remote services.

The enterprise service bus architecture (ESB, see [7]) is quickly gaining widespread adoption because it enables simplified service access and reuse across the enterprise. Services are plugged to the ESB to make them available for access by other enterprise applications. ESBs are usually built as service extensions to traditional messaging backbones ("messaging clouds"). Services and applications are connected across the ESB by creating "wires" that declaratively create a logical communication channel between the two. The SCA component and wiring model is used in this context both to drive deployment of SOA components and also to wire existing services, thus exposing its ability to function as both a component and a service composition model.

Scientific computing middleware software has, independently of its commercial counterpart, identified the need for an architectural model in which computing resources are consumed following the service model. The Open Grid Services Architecture (see [6,7]) shows how the Grid application model is supported by SOA. In this context, scientific workflows have been characterized as compositions of scientific services and specialized languages have been developed to enable the composition of complex scientific computing experiments as process oriented service compositions [9]. The Grid Process Execution Language (GPEL) in particular adapts WS-BPEL to deal with scientific and Grid computing requirements such as processing extremely large data sets, allocating dynamic resources from a Grid infrastructure, and supporting the integration with legacy scientific code among several others (see Chapter 15 in [9]).

Cross-references

▶ Workflow Management and Workflow Management System

Recommended Reading

1. Burbeck S. The Tao of e-business services. Available at http://www.ibm.com/developerworks/webservices/library/ws-tao/, October 2000.
2. Curbera F., Ferguson D., Nally M., and Stockton M. Toward a Programming Model for Service-Oriented Computing. In Proc. 3rd Int. Conf. Service-Oriented Computing. 2005, pp. 33–47.
3. Curbera F., Khalaf R., Leymann F., and Weerawarana S. Exception Handling in the BPEL4WS Language, In Proc. Int. Conf. Business Process Management, 2003, pp. 276–290.
4. Dustdar S. and Schreiner W. A survey on web services composition. Int. J. Web Grid Serv.,1(1):1–30, 2005.
5. Leymann F. and Roller D. Production Workflow. Prentice Hall, Englewood Cliffs, NJ, 1999.
6. Open Grid Services Architecture, Version 1.5. Available at http://www.ggf.org/documents/GFD.80.pdf, July 2006.
7. Papazoglou M. Web Services: Principles and Technology. Prentice Hall, Englewood Cliffs, NJ, 2007.
8. Szyperski C. Component Software. Addison Wesley, Reading, MA, 2002.
9. Taylor I.J., Deelman E., Gannon D.B., Shields M., (eds.). Workflows for e-Science. Scientific Workflows for Grids. Springer, Berlin, 2007.
10. Web Services Business Process Execution Language Version 2.0. Available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, April 2007.
11. Weerawarana S., Curbera F., Leymann F., Storey T., and Ferguson D. Web Services Platform Architecture. Prentice Hall, Englewood Cliffs, NJ, 2005.

Composite Event

ANNMARIE ERICSSON, MIKAEL BERNDTSSON,
JONAS MELLIN
University of Skövde, Skövde, Sweden

Definition

A composite event is a set of events matching an event specification.

Key Points

Pioneering work on composite events was done in the HiPAC project [3] and the ideas were extended and refined in most proposals for active object-oriented databases during the early 1990s.

A composite event is composed according to an event specification (in an event algebra), where the composition is performed using a set of event operators such as disjunction, conjunction and sequence. More advanced event operators have been suggested in literature, e.g., [2,4,5].

The *initiator* of a composite event is the event initiating the composite event occurrence and the *terminator* is the event terminating the composite event occurrence.

Events contributing to composite events may carry parameters (e.g., temporal) in which the event is said to occur. Events contributing to composite events are also referred to as constituent events.

Composite events need to be composed according to some event context that define which event that can participate in the detection of a composite event. The event context is an interpretation of the streams of contributing events. The seminal work by Chakravarthy et al. [1,2], defines four different event contexts (or consumption policies): recent, chronicle, continuous, and cumulative.

In the recent event context, only the most recent constituent events will be used to form composite events. The recent event context is, for example, useful if calculations must be performed on combinations of the last measured values of temperature and pressure in a tank [1,2].

In the chronicle event context, events are consumed in chronicle order. The earliest unused initiator/terminator pair are used to form the composite event. The chronicle event context is, for example, useful if sensors are placed along a conveyor-belt monitoring objects traveling along the belt and combinations of sensor events triggered by the same object is needed. In that case events must be combined in occurrence order since the first event from the first sensor and the first event from the second sensor are likely triggered by the same object [1,2].

In the continuous event context, each initiator starts the detection of a new composite event and a terminator may terminate one or more composite event occurrences. The difference between continuous and chronicle event contexts is that in the continuous event context, one terminator can detect more than one occurrence of the composite event.

In the cumulative event context, all events contributing to a composite event are accumulated until the composite event is detected. When the composite event is detected, all contributing events are consumed [1,2].

Cross-references

▶ Active Database (aDB)
▶ Active Database Execution Model
▶ Active Database Knowledge Model
▶ Active Database (Management) System (aDBS/aDBMS)
▶ ECA Rules
▶ Event
▶ Event Detection
▶ Event Specification

Recommended Reading

1. Chakravarthy S., Krishnaprasad V., Anwar E., and Kim S.K. Composite events for active databases: semantics contexts and detection. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 606–617.
2. Chakravarthy S. and Mishra D. Snoop: an expressive event specification language for active databases. Data Knowl. Eng., 14(1):1–26, 1994.
3. Dayal U., Blaustein B., Buchmann A., and Chakravarthyand S. et al. HiPAC: A Research Project in Active, Time-Constrained Database Management. Tech. Rep. CCA-88-02, Xerox Advanced Information Technology, Cambridge, 1988.
4. Gatziu S. Events in an Active Object-Oriented Database System. Ph.D. thesis, University of Zurich, Switzerland, 1994.
5. Gehani N., Jagadish H.V., and Smueli O. Event Specification in an Active Object-Oriented Database. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1992, pp. 81–90.

Composite Event Query

▶ Event Specification

Composite Web Applications

▶ Web Mashups

Composition

W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven,
The Netherlands

Synonyms

Service composition; Process composition

Definition

In computer science, *composition* is the act or mechanism to combine simple components to build more complicated ones. Composition exists at different levels. For example, one can think of the usual composition of functions in mathematics, i.e., the result of the composed function is passed to the composing one via a parameter. If one has to functions f and g, these can be combined into a new function $h = f.g$, i.e., $h(x) = f(g(x))$. Another level of abstraction is the level of activities. Here all kinds of process modeling languages can be used to compose activities into processes (e.g., Petri nets, BPMN, etc.). Typical composition operators are sequential composition, parallel composition, etc. Process composition is related to business process management, workflow management, etc. Yet another level of abstraction is provided by services, i.e., more complex services can be composed from simpler ones even when they do not reside in the same organization. Service composition is sometimes also referred to as orchestration and a typical language used for this purpose is BPEL.

Key Points

The composition of more complex components from simpler components has been common practice in computer science right from the start. It is clear that composition is needed to allow for "divide and conquer" strategies and reuse. One of the most complex issues is the composition of processes. There are basically two types of composition approaches: graphs-based languages and process algebras. Examples of graph-based languages are Petri nets, state charts, BPMN, EPCs, etc. In these languages activities and subprocesses are connected to impose some ordering relations. For example two transitions in a Petri net can be connected by a place such that the first one triggers the second one [3]. Process algebras enforce a more structured way of modeling processes. Typical operations are sequential composition ($x.y$, i.e., x is followed by y), alternative composition ($x + y$, i.e., there is a choice between x and y), and parallel composition ($x||y$, i.e., x and y are executed in parallel) [1,2]. Languages like BPEL provide a mixture of both styles, e.g., operators such as `sequence`, `switch`, `while` and `pick` correspond to the typical process-algebraic operators while the `flow` construct defines in essence an acyclic graph.

The principle of compositionality states that the meaning of a composite is determined by the meanings of its constituent parts and the rules used to combine them. For example, if a process is composed of parts that have certain properties, then these properties should be preserved by the composition and should not depend on lower-level interactions. Such properties can be obtained by simplifying the language used or restricting the compositions allowed.

Cross-references

- ▶ Abstraction
- ▶ BPEL
- ▶ BPMN
- ▶ Business Process Management
- ▶ Orchestration
- ▶ Petri Nets
- ▶ Web Services
- ▶ Workflow Management
- ▶ Workflow Patterns

Recommended Reading

1. Baeten J.C.M. and Weijland W.P. Process Algebra. Cambridge Tracts in Theoretical Computer Science, vol. 18. Cambridge University Press, Cambridge, 1990.
2. Milner R. Communicating and Mobile Systems: The Pi-Calculus. Cambridge University Press, Cambridge, UK, 1999.
3. van der Aalst W.M.P. Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management. In J. Desel, W. Reisig, G. Rozenberg (eds.). Lectures on Concurrency and Petri Nets. LNCS, vol. 3098. Springer, Berlin, 2004, pp. 1–65.

Comprehensions

Peter M.D. Gray
University of Aberdeen, Aberdeen, UK

Synonyms

Calculus expression; List comprehension; Set abstraction; ZF-expression

Definition

The comprehension comes from ideas of mathematical set theory. It originated as a way of defining sets of values so as to avoid the famous paradoxes of early set theory, by starting from other well-defined sets and using some carefully chosen constructors and filters. The values in the sets could be tuples of basic values, which suits the *relational model*, or they could be object identifiers, which fits with *ODMG object data models* [2], or they could be tagged variant records which fit well with *semi-structured data*. They could even be sets, lists or bags defined by other comprehensions.

The abstract structure of a comprehension precisely describes almost all the computations done in functional query languages, despite their very different surface syntax. Better still, it allows many optimizations to be expressed as well defined mathematical transformations.

Key Points

Consider an example, using SQL syntax, to find the set of surnames of persons whose forename is "Jim":

```
SELECT surname FROM person WHERE forename = "Jim"
```

Using a list comprehension this can be written as:

```
[surname(p) | p <- person; f <- forename
(p); f = "Jim"]
```

This denotes the list of values of the expression to the left of the vertical bar. This expression usually includes variables such as p which are instantiated by generators to the right of the bar. It can be transliterated as:

The set of values of the surname of p *such that* p *is in the set* person *and* f *is in the set* of forenames of p *and* f is equal to "Jim". Here *forename(p)* could alternatively be written *p.forename* or *(forename p)*.

Thus, the vertical bar can be read as *such that* and the semicolons as conjunctions (*and*). The arrows act as *generators*, supplying alternative possible values, subject to restrictions by predicate terms to the right, acting as *filters*. Thus p is generated from the set of persons but is only chosen where the forename of p satisfies the test of equalling "Jim".

In the above syntax the arrow operator is overloaded , so that if a function such as `forename` delivers a single value instead of a set then the arrow just assigns that single value to the variable on its left. Strictly, one should make a singleton set containing this value, and then extract it:

```
[surname(p) | p <- person; f <- [fore-
name(p)]; f = "Jim"]
```

This wasteful operation would be compiled away to give this equivalent form:

```
[surname(p) | p <- person; forename
(p) = "Jim"]
```

The term "list comprehension" is commonly used, but one should really distinguish between lists, sets and bags [1]. Thus comprehensions are usually represented internally as lists, but often the order is ignored, as in sets, and sometimes it is necessary to keep duplicates and form a bag, especially when totaling up the contents! Particular classes of operator used in comprehensions give rise to *monad comprehensions* and *monoid comprehensions* with valuable mathematical properties.

Cross-references

▶ OQL

Recommended Reading

1. Buneman P., Libkin L., Suciu D., Tannen V., and Wong L. Comprehension syntax. ACM SIGMOD Rec., 23(1):87–96, 1994.
2. Fegaras L. and Maier D. Towards an effective calculus for Object Query Languages. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 47–58.

Compressed and Searchable Data Format

▶ Indexing Compressed Text

Compressed Full-Text Indexing

▶ Indexing Compressed Text

Compressed Suffix Array

▶ Indexing Compressed Text

Compressed Suffix Tree

▶ Indexing Compressed Text

Compressing XML

▶ Managing Compressed Structured Text

Compression of Mobile Location Data

Goce Trajcevski[1], Ouri Wolfson[2], Peter Scheuermann[1]
[1]Northwestern University, Evanston, IL, USA
[2]University of Illinois at Chicago, Chicago, IL, USA

Synonyms

Spatio-temporal data reduction

Definition

In moving objects databases (MOD) [8], the data pertaining to the whereabouts-in-time of a given mobile object is commonly represented as a sequence of *(location, time)* points, ordered by the temporal dimension. Depending on the application's settings, such points may be obtained by different means, e.g., an on-board GPS-based system, RFID sensors, road-network sensors, base stations in a cellular architecture, etc. The main motivation for compressing the location data of the moving objects is twofold: (i) Reducing the storage requirements: for example, maintaining the information about the daily routes of a few million vehicles, even if the GPS samples are taken once every 30 s, can still generate Terra-Bytes (TB) of data. In addition, with the increase in the number of cellular phones and personal digital assistants that are location aware, the volume of the data corresponding all the mobile entities in a given region will even further increase. However, if a given point, say, (x, y, t) can be eliminated from the representation of the particular trajectory without prohibitively sacrificing the accuracy of its representation, then the space required for that point's storage can be saved; (ii) If a particular point along a given trajectory can be eliminated as soon as it is "generated" (i.e., when the *location* value is obtained by the on-board GPS at a given *time*), yet another type of savings can be achieved – that object need not transmit the *(location, time)* value to a given server, thus reducing the bandwidth consumption. This entry explains the basic problems involved in compressing spatio-temporal data corresponding to trajectories of mobile objects, and outlines the foundations of the approaches that have addressed some of those problems.

Historical Background

The field of *data compression* originated in the works of Shannon, Fano, and Huffman in the 1940s [11], and its main goal is to represent information in as compact form as possible. Some popular forms of data compression have, historically, been around even earlier, for instance, the Morse code has been used in telegraphy since the mid-nineteenth century. Based on the observation that some letters occur more frequently than others, the code assigns shorter sequences of (combinations of) "·" and "−" to such letters. Thus, for example, "*e*" → " · ", "*a*" → " ·−". On the other hand, the letters which occur less frequently, are assigned longer sequences like, for example, "*q*" → "−−·−". In this setting, the frequency of the occurrence of single letters provided statistical structure that was exploited to *reduce the average time* to transmit a particular message since, in practice, the duration of the symbol " − " is (approximately) three times longer than the duration of the "·" symbol. A natural extension is to use frequency of the *words* over a given alphabet, in order to further compress the encoding of a given text, which is used in the Grad-2 Braille coding. When the probability model of the source is known, the popular approach for encoding a collection of *letters* of a given *alphabet* is the Huffman coding [11]. Contrary to the ASCII/EBDCIC which are *fixed-length* codes, in the sense that every symbol is assigned same number of bits, Huffman code is a *variable-length* one, which assigns shorter codewords to symbols occurring less frequently, in an optimal manner with respect to the entropy of the source. When dealing with texts, some statistical correlations can be detected in terms of the occurrences of *words*. Taking this into consideration the, so called, *dictionary techniques* for data compression have been obtained, an example of which is the UNIX `compress` command. In computer science, the need for compression techniques was mainly motivated by the reduction of the size of the data for storage and transmission purposes.

Different kind of data may exhibit different kinds of structures that can be exploited for compression, provided a proper model is developed. For example, given the sequence of numbers {9,11,11,11,14,13, 15,17,16,17,20,21}, let x_n denote its nth element. If one transmits the binary representation of each x_i ($i \in \{1,2,...,12\}$), 5 bits-per-sample are needed, for a total of 60 bits transmitted. However, if one provides a model represented by the equation $\overline{x}_n = n + 8$, then the difference-sequence (i.e., the residual) of the initial sequence, represented as $e_n = x_n - \overline{x}_n$ becomes: {0, 1, 0, −1, 1, −1, 0, 1, −1, −1, 1, 1}. This sequence consists of only three different numbers {− 1, 0, 1} Using the mapping " − 1" → "00"; "0" → " − 1"; "1" → "10", each e_i can be encoded with only 2 bits. Hence, the sequence can be transmitted with a total of 24 bits, achieving 60% compression ratio and, consequently, savings in the transmission. Such intrinsic properties of the underlying domain have been heavily exploited in the areas of speech compression, image compression, etc. [11].

There are several classification of compression techniques. One example, as mentioned above, is *fixed* vs. *variable* length, however, one may also need to distinguish between *static* (the codewords are fixed, say, before the transmission) and *dynamic/adaptive*. The classification that is most relevant for this article is *lossless* vs. *lossy* compression. With lossless compression, the original data can be *exactly* recovered from the compressed one, which it is not the case for the lossy compression.

There are several different measures regarding the quality of a given compression method: (i) the complexity of the algorithms; (ii) the memory footprint required; (iii) the amount of compression; (iv) the quality of the data (in lossy compression). The main goal of the methods for compressing *spatio-temporal* data is to strike a good balance between the complexity of the algorithm and the error-bound on the compressed data with respect to the original one.

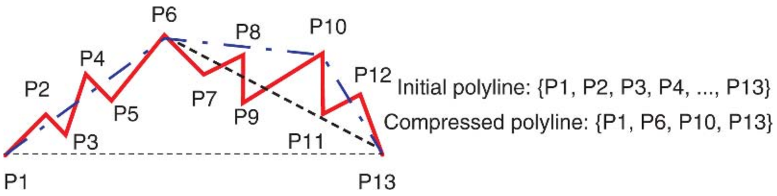
There are two research fields that have addressed problems similar in spirit to the ones of compressing mobile location data:

1. *Cartography*. The goal of the *map generalization* in cartography is to reduce the size/complexity of a given map for the purpose of simplified representation of the details appropriate to a given scale [16].
2. *Computational geometry* (CG). In particular, the problem of *polyline* (which is, a sequence of nodes specifying a chain of line segments) simplification [3], that can be described as follows. Given a polyline PL_1 with vertices $\{v_1,v_2,...,v_n\}$ in a respective k-dimensional Euclidean space, and a tolerance ε, construct another polyline PL_1' with vertices $\{v_1',v_2',...,v_m'\}$ in the same space, such that $m \leq n$ and for every point $P \in PL_1$ its distance from PL_1' is smaller than a given threshold: $dist(P, PL_1') \leq \varepsilon$. In case $\{v_1',v_2',...,v_m'\} \subseteq \{v_1, v_2,...,v_n\}$, PL_1' is a *strong* simplification of PL_1; otherwise PL_1' is called a *weak* simplification. There are two distinct facets of the minimal line simplification problem: (i) Given PL and ε, minimize the number of points m in PL' (known as min-# problem) [5], and (ii) Given PL and the "budget" m of the vertices in PL', minimize the error ε (known as min-ε problem).

A popular heuristic for polyline simplification in the context of map generalization was proposed by Douglas and Peucker in [6]. Essentially, it recursively approximates a given polyline in a "divide and conquer" manner, where the farthest vertex, according to the distance used, is selected as the "divide" point. Given a *begin_vertex* p_i and an *end_vertex* p_j, if the greatest distance from some vertex p_k to the straight line segment $\overline{p_ip_j}$ is greater than the tolerance ε, then the trajectory is broken into two parts at p_k and the procedure is recursively called on each of the sub-polylines $\{p_i,...,p_k\}$ and $\{p_k,...,p_j\}$; Otherwise, the vertices between p_i and p_j are removed from trajectory and this segment is simplified as a straight line $\overline{p_ip_j}$. An illustration of the DP heuristic is given in Fig. 1. Although the original version of the algorithm, as presented in [6], has a running time $O(n^2)$, an $O(n \log n)$ algorithm was presented in [9]. However, none of these algorithms can ensure an optimality, in terms of the size of the compression (alternatively, in terms of a minimal ε-error for a fixed reduction factor). An optimal algorithm was presented in [5], with a complexity of $O(n^2)$, subsequently extended for 3D and higher dimensions in [3].

Foundations

Assuming that the objects are moving in a 2D space with respect to a given coordinate system, a *trajectory*, which is often used in the MOD literature [8,13,15] to describe the motion of the moving objects, is defined as a function $F_t : T \rightarrow \mathcal{R}^2$ which maps a given (temporal) interval $[t_b, t_e]$ into a one-dimensional subset of $\mathcal{R}^2$. It is represented as a sequence of 3D points (2D geography + time) (x_1,y_1,t_1), (x_2,y_2,t_2),...,(x_n,y_n,t_n), where $t_b = t_1$ and $t_e = t_n$ and $t_1 \leq t_2 \leq ... \leq t_n$. Each point (x_i,y_i,t_i) in the sequence represents the 2D location (x_i,y_i) of the object, at the time t_i. For every $t \in (t_i,t_{i+1})$, the *location* of the object is obtained by a *linear interpolation* between (x_i,y_i) and (x_{i+1},y_{i+1}) with the ratio $(t - t_i)/(t_{i+1} - t_i)$, which is, in between two points the object is assumed to move along a straight line-segment and with a constant speed. The 2D


P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
Initial polyline: {P1, P2, P3, P4, ..., P13}
Compressed polyline: {P1, P6, P10, P13}

Compression of Mobile Location Data. Figure 1. Douglas–Peucker heuristic.

projection of the trajectory is a polygonal chain with vertices (x_1, y_1), (x_2, y_2)...(x_n, y_n), called a *route*.

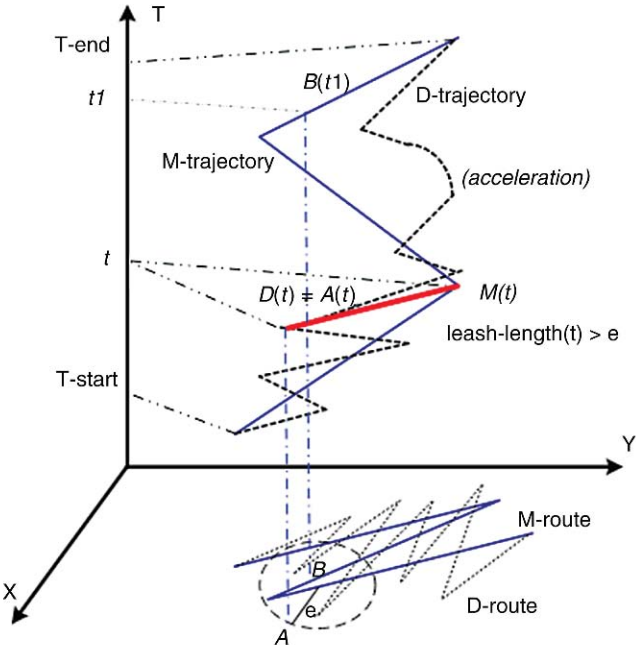
Observe that a trajectory may represent both the *past* and the *future* motion, i.e., the *motion plan* of a given object (c.f. [8]). Typically, for future trajectories, the user provides the starting location, starting time and the destination (plus, possibly, a set of to-be-visited) points, and the MOD server uses these, along with the distribution of the speed-patterns on the road segments as inputs to a dynamic extension of the Dijkstra's algorithm [13], to generate the shortest travel-time trajectory.

One may be tempted to straightforwardly apply the existing results on polyline simplification from the CG literature (e.g., the DP [6,9] or the optimal algorithm [3,5]), in order to compress a given trajectory. However, as pointed out in [4], the semantics of the spatial + temporal domain combined, raises two major concerns:

1. What is the *function* used to measure the *distance between points* along trajectories?
2. How does the choice of that function affect the *error* that the compressed trajectory introduces in the answers of the popular spatio-temporal queries? In the sequel, each of these questions is addressed in a greater detail.

Distance Function

A popular distance function between two curves, often used in CG applications is the, so called, Hausdorff distance [1]. Essentially, two curves C_1 and C_2, their Hausdorff distance simply looks for the smallest ε such that C_1 is completely contained in the ε-neighborhood of C_2 (i.e., C_1 is completely contained in the Minkowski Sum of C_2 and a disk with radius ε) and vice versa. Although it is arguably a very natural distance measure between curves and/or compact sets, the Hausdorff distance is too "static", in the sense that it neither considers any direction nor any dynamics of the motion along the curves. A classical example of the inadequacy of the Hausdorff distance, often used in the CG literature [1,2] is the *"man walking the dog"*. Figure 2 illustrates the corresponding routes of the man *(M-route)* and the dog *(D-route)*, as well as their trajectories *M-trajectory* and *D-trajectory*. Observe that, ignoring the temporal aspect of their motions, the D-route and the M-route are within Hausdorff distance of *e*, as exemplified by the points *A* and *B* in the *XY* plane. However, their actual *temporally aware* distance corresponds to the minimal length of the leash that the man needs to hold. The 3D part of Fig. 2 illustrates the discrepancy between the distances among the points along the *routes*, and their corresponding counterparts along *trajectories*: when the dog is at the point



Compression of Mobile Location Data. Figure 2. Hausdorff vs. Fréchet distance.

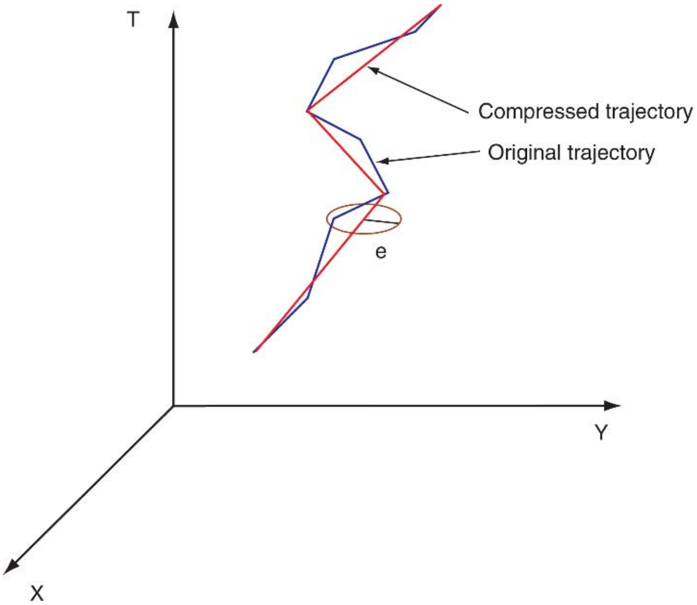
A, which is at time *t*, the man is actually at *M*(*t*), and their distance is much greater then *e* (the man is at the geo-location *B* at the time $t_1 > t$). The Fréchet distance [2] is more general than the Hausdorff one, in the sense that it allows for a variety of possible motion-patterns along the given route-segments. As an illustration, observe that on the portion of the *D-trajectory*, the dog may be moving non-uniformly (i.e., accelerating) along a route segment.

The discussion above illustrates two extreme points along the spectrum of distance functions for moving objects. Although the Fréchet distance is the most general one, regarding the possible dynamics of motions, it is unnecessarily complex for the common trajectory model in MOD settings. The inadequacy of the L_2 norm as a distance function for spatio-temporal trajectories was pointed out in [4] where, in order to properly capture the semantics of the problem domain, alternative distance functions were introduced. Given a spatio-temporal point $p_m = (x_m, y_m, t_m)$ and a trajectory segment $\overline{p_i, p_j}$ between the vertices $p_i = (x_i, y_i, t_i)$ and $p_j = (x_j, y_j, t_j)$, [4] proposed the E_u and E_t distance functions between the p_m and $\overline{p_i, p_j}$, which are explained next

1. E_u – The three dimensional time_uniform distance, which is defined when $t_m \in [t_i, t_j]$, as follows: $E_u(p_m, \overline{p_i p_j}) = \sqrt{(x_m - x_c)^2 + (y_m - y_c)^2}$ where $p_c = (x_c, y_c, t_c)$ is the unique point on $\overline{p_i p_j}$ which has the same *time* value as p_m (i.e., $t_c = t_m$). An illustration of using the E_u distance function for reducing the size of a given trajectory is presented in Fig. 3. Intuitivelly, the distance is measured at equal *horizontal* planes, for the respective values of the temporal dimension. One can "visually" think of the relationship between the original trajectory and the compressed trajectory as follows: the original trajectory is contained inside the sheared cylinder obtained by sweeping (the center of) a horizontal disk with radius ε along the compressed trajectory.
2. E_t – The time distance is defined as: $E_t(p_m, \overline{p_i p_j}) = |t_m - t_c|$, where t_c is the time of the point on the *XY* projection $\overline{p'_i p'_j}$ of $\overline{p_i p_j}$, which is closest (in terms of the 2D Euclidean distance) to the *XY* projection p'_m of p_m. Intuitively, to find the time distance between p_m and $\overline{p_i p_j}$, one needs to:

   1. Project each of them on the *XY* plane;
   2. Find the point $p'_c \in \overline{p'_i, p'_j}$ that is closest to p'_m;
   3. Find the difference between the corresponding times of p_c and p_m.

An important observation regarding the *computation* of the compressed version of a given original trajectory as an input, is that both the DP [6] and the



Compression of Mobile Location Data. Figure 3. E_u distance function for trajectory compression.

optimal algorithm [5] can be used, provided they are appropriately modified to reflect the distance function used. Experimental results in [4] demonstrated that the DP heuristics yields a compression factor that is very comparable to the one obtained by the optimal algorithm, however, its execution is much faster.

Spatio-Temporal Queries and Trajectory Compression

The most popular categories of spatio-temporal queries, whose efficient processing has been investigated by many MOD researchers [8] are:

1. *where_at*(T,t) – returns the expected location at time t.
2. *when_at*(T,x,y) – returns the time t at which a moving object on trajectory T is expected to be at location (x,y).
3. *intersect*(T,P,t_1,t_2) – is *true* if the trajectory T intersects the polygon P between the times t_1 and t_2. This is an instance of the, so called, spatio-temporal range query).
4. *nearest_neighbor*(T,O,t) – The operator is defined for an arbitrary set of trajectories O, and it returns a trajectory T' of O. The object moving according to T', at time t, is closest than any other object of O to the object moving according to T.
5. *join*(O, Θ) – O is a set of trajectories and the operator returns the pairs (T_1,T_2) such that their distance, according to the distance function used, is less than a given threshold Θ.

An important practical consideration for compressing trajectory data is how the (im)precision generated by the compression, affects the answers of the spatio-temporal queries. As it turns out, the distance function used in the compression process plays an important role and, towards this, the concept of *soundness* [4] of a distance function with respect to a particular query was introduced in [4]. A pair *(distance_function, query)* is called *sound* if the error of the *query*-answer, when processed over the compressed trajectory is bounded. In case the error is *unbounded*, which is, although the compression itself guarantees a distance-error of ε between the points on the compressed trajectory with respect to the original one, the error of the answer to the query can grow arbitrarily large, the pair is called *unsound*. Table 1 below (adapted from [4]) summarizes the soundness properties of three distance functions with respect to five categories of spatio-temporal queries.

Compression of Mobile Location Data. Table 1. Distance soundness and error-bound on spatio-temporal query answers

	Where_at	*When_at*	*Intersect*	*Nearest neighbor*
E_2 (L_2*over routes)*	Unsound	Unsound	Unsound	Unsound
E_u	Sound (ε)	Unsound	Sound (ε)	Sound (2ε)
E_t	Unsound	Sound (ε)	Unsound	Unsound

As one can see, there is no single distance function that is sound for all the possible spatio-temporal queries.

The compression techniques for spatio-temporal data presented thus far, implicitly assumed that the trajectories are available in their entirety, i.e., they are past-motion trajectories. However, in practice, it is often the case that the *(location, time)* data is generated on-board mobile units, and is transmitted to the MOD server in real time [7,17]. *Dead-reckoning* is a policy which essentially represents an agreement between a given moving object and the MOD server regarding the updates transmitted by that particular object. The main idea is that the *communication* between them can be reduced (consequently, network bandwidth can be spared) at the expense of the *imprecision* of the data in the MOD representing the object's motion. In order to avoid an unbounded error of the object's location data, the agreement specifies a threshold δ that is a parameter of the policy, which can be explained as follows:

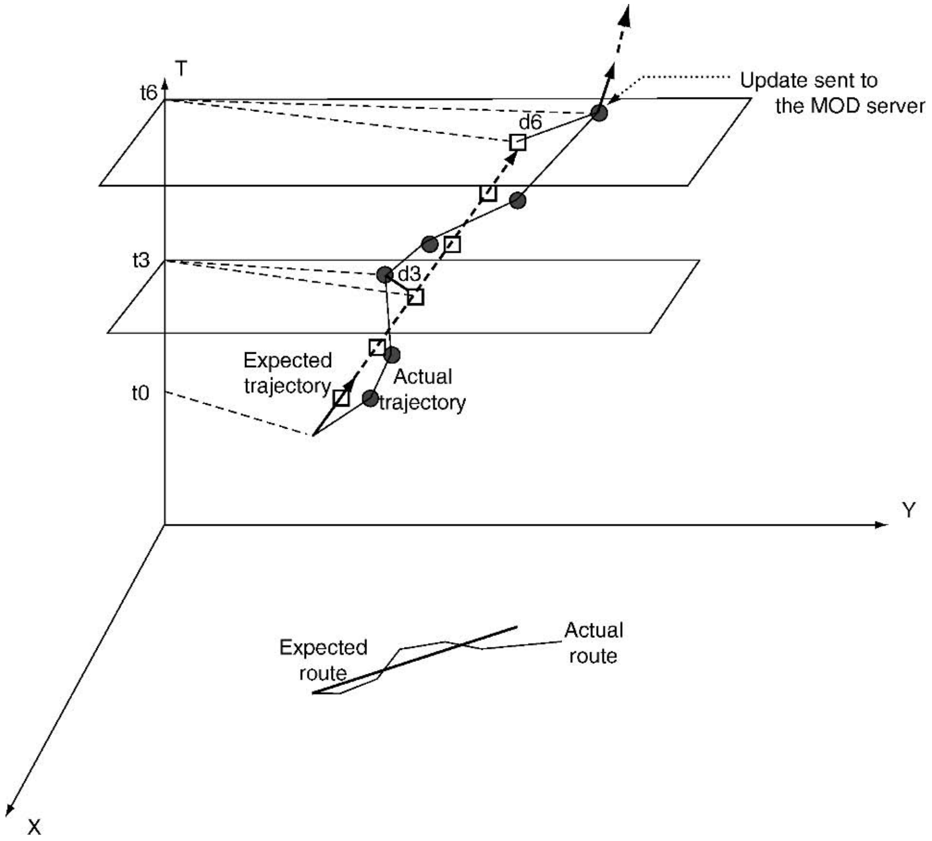
1. The object sends its *location* and the *expected velocity* to the MOD server and, as far as the MOD server is concerned, the future trajectory of that object is an infinite ray originating at the update point and obtained by extrapolation, using the velocity vector.
2. The information that the MOD server has is the *expected trajectory* of the moving object. However, each moving object is aware of its *actual location*, by periodically sampling it, e.g., using an on-board GPS.
3. For as long as its actual location at a given time t_i does not deviate by more than δ from the location that the MOD estimates at t_i using the information previously transmitted, the object does not

transmit any new updates. When the actual distance deviates by more then δ from its location on the expected trajectory, the object will send another *(location, time, velocity)* update.

The policy described above is commonly known as a *distance-based* dead reckoning, and an illustration is given in Fig. 4. At time t_0 the object sent its location and the predicted velocity (arrowed line) to the MOD server. The dashed line extending the vector indicate the expected trajectory of the moving object and the squares along it indicate the object's positions at six time instances, as estimated by the MOD, while the shaded circles indicate the actual positions of the object. Typically, the actual trajectory is obtained by connecting the GPS points with straight line-segments, assuming that in-between two updates, the object was moving with a constant speed. As illustrated, at t_6 the distance between the actual position and the MOD-estimated one exceeds the threshold agreed upon ($d_6 > \delta$) and the object sends a new update, at which point the MOD changes the expected trajectory, based on that update. Thus, at t_6, the MOD server actually performs two tasks:

1. Corrects its own "knowledge" about the recent past and approximates the actual trajectory between t_0 and t_6 with a straight line-segment, which defines the *actual simplification* of the near-past trajectory;
2. generates another infinite ray corresponding to the future-expected trajectory, starting at the last update-point, and using the newly received velocity vector for extrapolation.

Various trade-offs between the update costs and the (impacts on the) imprecision of the MOD data for several different variants of dead reckoning are investigated in [17]. The dead-reckoning, in a sense, achieves in real-time both of the goals of compression: – reduces the communication, and enables the MOD server to store only a subset of the actual trajectory. Assuming that a dead-reckoning policy with threshold δ was used in real-time, clearly, the MOD has obtained a compressed past-trajectory, say $Tr_m{}^c$, of a given mobile



Compression of Mobile Location Data. Figure 4. Distance-based dead-reckoning policy.

object o_m. If o_m was to transmit every single GPS-based update, i.e., no dead-reckoning applied, the MOD would have an uncompressed trajectory Tr_m available. The results in [14] have established that ${Tr_m}^c$ is a strong simplification of Tr_m, with an error-bound 2δ.

Key Applications

The compression of moving objects trajectories data is of interest in several scientific and application domains.

Wireless Sensor Networks (WSN)

Wireless sensor networks consist of a large number of *sensors* – devices that are capable of measuring various phenomena; performing elementary calculations; and communicating with each other, organizing themselves in an ad hoc network [19]. A particularly critical aspect of the WSN is the efficient management of the energy-reserves, given that the communication between two nodes drains a lot more battery-power than the operations of sensing and (local) computing. Consequently, in many tracking applications that can tolerate delays and imprecision in the *(location, time)* data, performing local compression of the trajectory data, before it is sent to a particular sink, can yield substantial increase in the networks' lifetime. Different policies for such compressions are presented in [18].

Location-Based Services (LBS)

A variety of applications in LBS depend on the data for mobile objects with different mobility properties (e.g., pedestrians, private vehicles, taxis, public transportation, etc.). Typically, LBS are concerned with a context-aware delivery of the data which matches the preferences of users based on their locations [12]. In order to provide faster response time, and more relevant information, the LBS should be able to predict, based on the motion patterns, what kind of data will be relevant/requested in a near future by given users. This, in turn, implies some accumulated knowledge about the mobility patterns of the users in the (near) past. However, keeping such data in its entirety can impose prohibitively high storage requirements.

Geographic Information Systems (GIS)

Recently, a plethora of services and devices has emerged for providing path planning and navigation for the mobile users: from MapQuest and Google-maps, through Garmin and iPaq Travel Companion. Each of these services relies on some traffic-based information in order generate the optimal (in distance or travel-time) path for their users. However, as the traffic conditions fluctuate, the future-portions of the routes may need to be recalculated. In order to better estimate the impact of the traffic fluctuations, some knowledge from the past is needed which, ultimately means storing some past information about trajectories. However, as observed in the literature [4], storing the uncompressed trajectory data corresponding to daily driving activities of few millions of users, could require TBs of data.

Spatio-Temporal Data Mining

Clustering is a process of grouping a set of (physical or abstract) objects into classes of similar objects, and its purpose is to facilitate faster data analysis in a given domain of interest. With the recent advances in miniaturization of computing devices and communications technologies, the sheer volume makes it very costly to apply clustering to the original trajectories' data. Compressing such data, especially if one can guarantee a bounded error for the queries of interest, can significantly improve the processing time for many algorithms for trajectories clustering [10].

Future Directions

Any problem-domain that depends on storing large volumes of trajectories' data, in one way and level or another, needs some sort of data compression in order to reduce the storage requirements and to speed up processing of spatio-temporal queries of interest. Clearly, a desirable property of the compression techniques is to ensure a bound on the errors of the answers to the queries.

There are several directions of interest for the future research on mobile data compression. In applications like GIS and LBS, it is a paramount to add some context-awareness to the compression techniques. For example, combining the mobile location data compression with the particular tourists attractions and the season/time, could provide a speed-up in algorithms which are used for generating real-time advertisements, while ensuring that the error (in terms of users that received particular ad) is bounded. An interesting aspect that has been presented in [4] is the, so-called, *aging* of the trajectories: a trajectory that is 1-week old could have higher impact on the traffic-impact analysis, than a trajectory that was recorded

5weeks ago. Consequently, one may reduce the older trajectory with a higher error-bound, thus further reducing the storage requirements. Automatizing this process in a manner that reflects the specifics of a given problem-domain (e.g., context-aware information delivery) is an open question. Despite the large body of works on OLAP and warehousing of traditional data, very little has been done on spatio-temporal OLAP. It is likely that the process of mobile data compression will play an important role in these directions.

Cross-references

▶ Data Compression
▶ Data Mining
▶ Moving Objects Databases

Recommended Reading

1. Alt H. and Guibas L. Discrete geometric shapes: matching, interpolation, and approximation. In Handbook of Computational Geometry. Elsevier, Amsterdam, 1999.
2. Alt A., Knauer C., and Wenk C. Comparison of distance measures for planar curves. Algorithmica, 38(1):45–58, 2004.
3. Barequet G., Chen D.Z., Deascu O., Goodrich M.T., and Snoeyink J. Efficiently approximating polygonal path in three and higher dimensions. Algorithmica, 33(2):150–167, 2002.
4. Cao H., Wolfson O., and Trajcevski G. Spatio-temporal data reduction with deterministic error bounds. VLDB J., 15(3):211–228, 2006.
5. Chan W. and Chin F. Approximation of polygonal curves with minimum number of line segments or minimal error. Int. J. Computat. Geometry Appl., 6(1):59–77, 1996.
6. Douglas D. and Peucker T. Algorithms for the reduction of the number of points required to represent a digitised line or its caricature. Can. Cartographer, 10(2):112–122, 1973.
7. Gedik B. and Liu L. Mobieyes: a distributed location monitoring service using moving location queries. IEEE Trans. Mobile Comput., 5(10):1384–1402, 2006.
8. Güting R.H. and Schneider M. Moving objects databases. Morgan Kaufmann, Los Altos, CA, 2005.
9. Hershberger J. and Snoeyink J. Speeding up the douglas-peucker line-simplification algorithm. In Proc. 5th Int. Symp. on Spatial Data Handling, 1992, pp. 134–143.
10. Jensen C.S., Lin D., and Ooi B.C. Continuous clustering of moving objects. IEEE Trans. Knowl. Data Eng., 19(9):1161–1174, 2007.
11. Sayood K. Introduction to Data Compression. Morgan Kaufmann, Los Altos, CA, 1996.
12. Schiller J. and Voisard A. Location-Based Services. Morgan Kaufmann, Los Altos, CA, 2004.
13. Trajcevski G., Wolfson O., Hinrichs K., and Chamberlain K. Managing uncertainty in moving objects databases. ACM Trans. Database Syst., 29(3):463–507, 2004.
14. Trajcevski G., Cao H., Wolfson H., Scheuermann P., and Vaccaro D. On-line data reduction and the quality of history in moving objects databases. In Proc. 5th ACM Int. Workshop on Data Eng. for Wireless and Mobile Access, 2006, pp. 19–26.
15. Vlachos M., Hadjielefteriou M., Gunopulos D., and Keogh E. Indexing multidimensional time-series. VLDB J., 15(1):1–20, 2006.
16. Weibel R. Generalization of spatial data: Principles and selected algorithms. In Algorithmic Foundations of Geographic Information Systems. Van Kreveld M. Nievergelt J., Roos T., and Widmayer P. (eds.). LNCS Tutorial Springer, Berlin, 1998.
17. Wolfson O., Sistla A.P., Chamberlain S., and Yesha Y. Updating and querying databases that track mobile units. Distrib. Parallel Databases, 7(3):257–387, 1999.
18. Xu Y. and Lee W.-C. Compressing moving object trajectory in wireless sensor networks. Int. J. Distrib. Sensor Netw. 3(2): 151–174, 2007.
19. Zhao F. and Guibas L. Wireless sensor networks: an information processing approach. Morgan Kaufmann, Los Altos, CA, 2004.

Computational Media Aesthetics

Chitra Dorai
IBM T. J. Watson Research Center, Hawthorne, NY, USA

Synonyms

CMA; Media semantics; Production-based approach to media analysis

Definition

Computational media aesthetics is defined as the algorithmic study of a variety of image and aural elements in media founded on their patterns of use in film grammar, and the computational analysis of the principles that have emerged underlying their manipulation, individually or jointly, in the creative art of clarifying, intensifying, and interpreting some event for the audience [3]. It is a computational framework to establish semantic relationships between the various elements of sight, sound, and motion in the depicted content of a video and to enable deriving reliable, high-level concept-oriented content annotations as opposed to verbose low-level features computed today in video processing for search and retrieval, and nonlinear browsing of video. This media production knowledge-guided semantic analysis has led to a shift away from a focus on low level features that cannot answer high level queries for all types of users, to applying the principled approach of computational media aesthetics to analyzing and interpreting diverse video domains such as movies, instructional media, broadcast video, etc.

Historical Background

With the explosive growth of media available on the Web, especially on hugely popular video sharing websites such as YouTube, managing the digital media collections effectively and leveraging the media content in the archives in new and profitable ways continues to be a challenge to enterprises, big and small. Multimedia content management refers to everything from ingesting, archival and storage of media to indexing, annotation and tagging of content for easy access, search and retrieval, and browsing of images, video and audio. One of the fundamental research problems in multimedia content management is the semantic gap – that renders all automatic content annotation systems of today brittle and ineffective – between the shallowness of features in their descriptive power that can be currently computed automatically and the richness of meaning and interpretation that users desire search algorithms to associate with their queries for easy searching and browsing of media. Smeulders et al. [8] describe that while "the user seeks semantic similarity, the database can only provide similarity on data processing." This semantic gap is a crucial obstacle that content management systems have to overcome in order to provide reliable media descriptions to drive search, retrieval, and browsing services that can gain widespread user acceptance and adoption. There is a lack of framework to establish semantic relationships between the various elements in the content since current features are frame/shot-representational and far too simple in their expressive power.

Addressing the semantic gap problem in video processing will enable innovative media management, annotation, delivery and navigational services for enrichment of online shopping, help desk services, and anytime-anywhere training over wireless devices. Creating technologies to annotate content with deep semantics results in an ability to establish semantic relationships between the form and the function in the media, thus for the first time enabling user access to stored media not only in predicted manner but also in unforeseeable ways of navigating and accessing media elements. Semantics-based media annotations will break the traditional linear manner of accessing and browsing media, and support vignette-oriented viewing of audio and video as intended by the content creators. This can lead to new offerings of customized media management utilities for various market segments such as education and training video archives, advertisement houses, news networks, broadcasting studios, etc.

Foundations

Computational Media Aesthetics advocates an approach that markedly departs from existing methods for deriving video content descriptions by analyzing audio and visual features (for a survey of representative work, see [8]). It proposes that to go beyond describing just what is seen in a video, the visual and emotional impact of how the content is depicted needs to be understood. Both media compositional and aesthetic principles need to guide media analysis for richer, more expressive descriptions of the content depicted and seen.

What are the methodologies for analyzing and interpreting media? Structuralism, in film studies for example, proposes film segmentation followed by an analysis of the parts or sections. Structural elements or portions of a video, when separated from cultural and social connotations can be treated as plain data and therefore, can be studied using statistical and computational tools. Another rich source is production knowledge or film grammar. Directors regularly use accepted rules and techniques to solve problems presented by the task of transforming a story from a written script to a captivating visual and aural narration [2]. These rules encompass a wide spectrum of cinematic aspects ranging from shot arrangements, editing patterns and the triangular camera placement principle to norms for camera motion and action scenes. Codes and conventions used in narrating a story with a certain organization of a series of images have become so standardized and pervasive over time that they appear natural to modern day film production and viewing. However, video production mores are found more in history of use, than in an abstract predefined set of regulations, are descriptive rather than prescriptive, and elucidate on ways in which basic visual and aural elements can be synthesized into larger structures and on the relationships that exist between the many cinematic techniques employed worldwide and their intended meaning to a movie audience.

Media aesthetics is both a process of examination of media elements such as lighting, picture composition, and sound by themselves, and a study of their role in manipulating the viewer's perceptual reactions, in communicating messages artistically, and in synthesizing effective media productions [10]. Inspired by it, Dorai and Venkatesh defined Computational media aesthetics [3] as the algorithmic study of a variety of image and aural elements in media guided by the patterns of their use, and the computational analysis of the principles for manipulating these elements to facilitate high-level content annotations.

Computational media aesthetics provides a handle on interpreting and evaluating relative communication effectiveness of media elements in productions through knowledge of film codes that mediate perception of the content shown in the video. It exposes the semantic and semiotic information embedded in the media production by focusing not merely on the representation of perceived content in digital video, but on the semantic connections between the elements and the emotional, visual appeal of the content seen and remembered. It advocates a study of mappings between specific cinematic elements and narrative forms, and their intended visual and emotional import.

In multimedia processing, many research efforts have sought to model and describe specific events occurring in a particular video domain in detail for providing high-level descriptions; computational media aesthetics, on the other hand enables development of video analysis techniques founded upon production knowledge for film/video understanding, for the extraction of high-level semantics associated with the expressive elements and narrative forms synthesized from the cinematic elements, and for the detection of high-level mappings through the use of software models. It highlights the systematic use of film grammar, as motivation and also as foundation in the automated process of analyzing, characterizing, and structuring of produced videos for media search, segment location, and navigational functions.

Computational media aesthetics provides a framework to computationally determine elements of form and narrative structure in videos from the basic units of film grammar namely, the shot, the motion, the recording distances, and from the practices of combination that are commonly followed during the audiovisual narration of a story. At first, primitive computable aspects of cinematographic techniques are extracted. New expressive elements (higher order semantic entities) can then be defined and constructed from these primitive aspects. Both the definition and extraction of these semantic entities are based on film grammar, and these entities are formulated only if directors purposefully design them and manipulate them. The primitive features and the higher order semantic notions thus form the vocabulary of content description language for media.

Key Applications

In seeking to create tools for the automatic understanding of media, computational media aesthetics states the problem as one of faithfully reflecting the forces at play in media production, and interpreting the data with its maker's eye. Several studies have explored the workings of Computational Media Aesthetics when applied to extraction of meaning using many of the aesthetic elements introduced by Zettl [10]: Time, sound and color. Adams et al. [1] took an example of carrying one aspect of film grammar all the way from literature to computable entity, namely tempo and pace for higher level analysis of movies. Adams et al. [1] showed that although descriptive and sometimes fuzzy in scope, film grammar provides rich insights into the perception of subjective time as tempo and pace and its manipulation by the makers of film for drama. Further research [9,6,5,7,4] has applied this approach pervasively from extracting mood in music, emotion in movies, to adding musical accompaniment to videos and extracting semantic metadata for mobile images at the time of image capture.

Film is not the only domain with a grammar to leverage in analysis. News, sitcoms, educational video, etc., all have more or less complex grammars that may be used to capture their crafted structure and to derive semantic descriptions with automated techniques following the framework of computational media aesthetics.

Cross-references

▶ Media Semantics
▶ Multimedia Processing
▶ Video Analysis

Recommended Reading

1. Adams B., Dorai C., and Venkatesh S. Towards automatic extraction of expressive elements from motion pictures: tempo. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2000, pp. 641–645.
2. Arijon D. Grammar of the film language. Silman-James Press, Los Angeles, CA, 1976.
3. Dorai C. and Venkatesh S. Computational media aesthetics: finding meaning beautiful. IEEE Multimed., 8(4):10–12, 2001.

4. Marc Davis. Editing out video editing. IEEE Multimed., 10(2):2–12, 2003.
5. Mulhem P., Kankanhalli M.S., Ji Yi., and Hassan H. Pivot vector space approach for audio-video mixing. IEEE Multimed., 10 (2):28–40, 2003.
6. Salway A. and Graham M. Extracting information about emotions in films, In Proc. 9th Int. Conf. on Multimedia Modeling, 2003, pp. 299–302.
7. Sarvas R., Herrarte E., Wilhelm A., and Davis M. Metadata creation system for mobile images. In Proc. 2nd Int. Conf. Mobile Systems, Applications and Services, 2004, pp. 36–48.
8. Smeulders A., Worring M., Santini S., and Gupta A. Content based image retrieval at the end of the early years. IEEE Trans. Pattern Anal. Mach. Intell., 22(12):1349–1380, 2000.
9. Yazhong Feng, Yueting Zhuang, and Yunhe Pan. Music information retrieval by detecting mood via computational media aesthetics. In Proc. IEEE/WIC Int. Conf. on Web Intelligence, 2003, pp. 235–241.
10. Zettl H. Sight, Sound, Motion: Applied Media Aesthetics Wadsworth Publishing, Belmont, CA, 1999.

Computational Ontology

▶ Ontology

Computer Human Interaction (CHI)

▶ Human-Computer Interaction

Computer-based Physician Order Entry

▶ Computerized Physician Order Entry

Computer-based Provider Order Entry

▶ Computerized Physician Order Entry

Computer-Interpretable Formalism

▶ Executable Knowledge

Computerized Order Entry (COE)

▶ Computerized Physician Order Entry

Computerized Physician Order Entry

Michael Weiner
Indiana University School of Medicine, Indianapolis, IN, USA

Synonyms

Computer-based physician order entry; Computer-based provider order entry; Computerized provider order entry; Computerized order entry (COE); Physician order entry

Definition

In daily medical practice, physicians routinely create plans of diagnosis and treatment for their patients. These plans typically contain specific, formal orders – directives – that are expected to be implemented by other medical professionals, such as nurses or personnel at laboratories or pharmacies. When such personnel are expected to implement part of a physician's diagnosis or treatment plan, corresponding orders must be created and documented in the patient's medical record. Physicians have traditionally used paper-based charting systems to record medical orders.

Computerized physician order entry (CPOE) is a process by which physicians directly enter medical orders into a computer. CPOE is typically done when the computer is being used to access an electronic health record (EHR), and the physician is creating a treatment plan for a specific patient in a clinical setting.

In many medical institutions, non-physicians such as nurses, dieticians, social workers, pharmacists, therapists, or advanced nurse practitioners can also enter certain types of orders, hence the broader, useful term "*computerized provider order entry*."

Historical Background

CPOE was first implemented and described in the latter half of the twentieth century. In the US, early reports came from several institutions, including Harvard Medical School and Brigham and Women's Hospital, the US Veterans Health Administration [16,20], Vanderbilt University [8], University of Virginia [12], Indiana University, and Regenstrief Institute for Health Care [14].

In 1994, Sittig and Stead published "Computer-based physician order entry: the state of the art" [17], summarizing many of the early results. Many difficulties were reported regarding leadership, delays, cultural resistance, high costs, technical support, workflow, and other operational difficulties for end users.

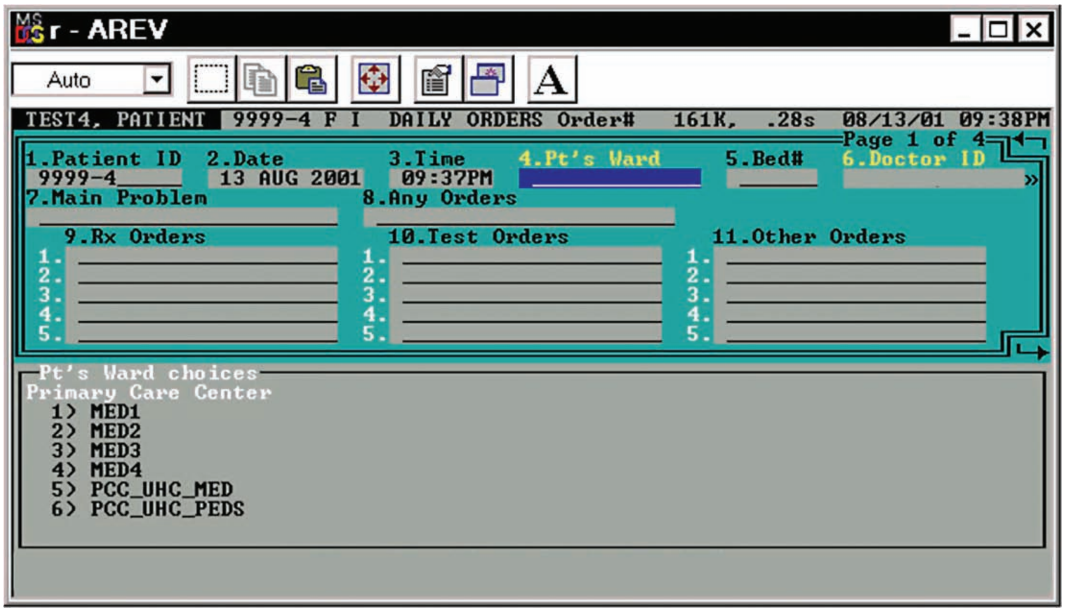
By the end of 2006, CPOE was on the rise, though adoption rates – often correlated with adoption of EHRs – varied widely throughout the world. Most modern EHR systems, whether developed by public, private, or academic institutions, would be expected to include at least some form of CPOE. In the US, only 10% of hospitals had complete availability of CPOE in 2002 [2]. In the United Kingdom, Australia, and New Zealand [21], the fraction is much higher, since use of EHRs exceeds 80% and is approaching 99% among general practitioners in ambulatory practice.

Foundations

CPOE can be used to order a variety of medical services. In some medical institutions with EHRs, CPOE can be used to order any type of medical service and may be required to generate any order. When not required, providers may have the opportunity to select between writing orders in a paper-based chart or an EHR, or clerks or other professionals may perform CPOE on a provider's behalf or direction.

CPOE is performed via a user interface of some kind, though this may occur on a desktop computer, terminal or thin client, personal digital assistant, other portable computer, or other form of computer. A user would typically authenticate himself or herself, identify a patient, and proceed to enter orders, often by navigating through a series of forms, each of which might facilitate a certain type of order, such as for radiology, laboratory, pharmacy, nursing, or referral (see Fig. 1). The layout or interface seen by the user is highly variable and may depend on the developer, personal preferences, underlying database structures, or medical or administrative processes generated in response to specific orders.

Many EHRs allow providers to generate orders and non-order documentation in a single computer session. Non-order documentation may include clinical details such as historical information, measurements, other observations, patient's preferences or directives, or narrative notes or reports. The workflow imposed by a CPOE system should be considered carefully in conjunction with the user's baseline workflow. Greatest success with implementation can often be found when the system does not disrupt the user's own pattern of work.



```
TEST4, PATIENT  9999-4 F I  DAILY ORDERS Order#   161K,  .28s  08/13/01 09:38PM
                                                              Page 1 of 4
1.Patient ID  2.Date        3.Time      4.Pt's Ward      5.Bed#   6.Doctor ID
 9999-4_____   13 AUG 2001   09:37PM
7.Main Problem              8.Any Orders

   9.Rx Orders               10.Test Orders            11.Other Orders
1.                        1.                        1.
2.                        2.                        2.
3.                        3.                        3.
4.                        4.                        4.
5.                        5.                        5.

 Pt's Ward choices
Primary Care Center
  1> MED1
  2> MED2
  3> MED3
  4> MED4
  5> PCC_UHC_MED
  6> PCC_UHC_PEDS
```

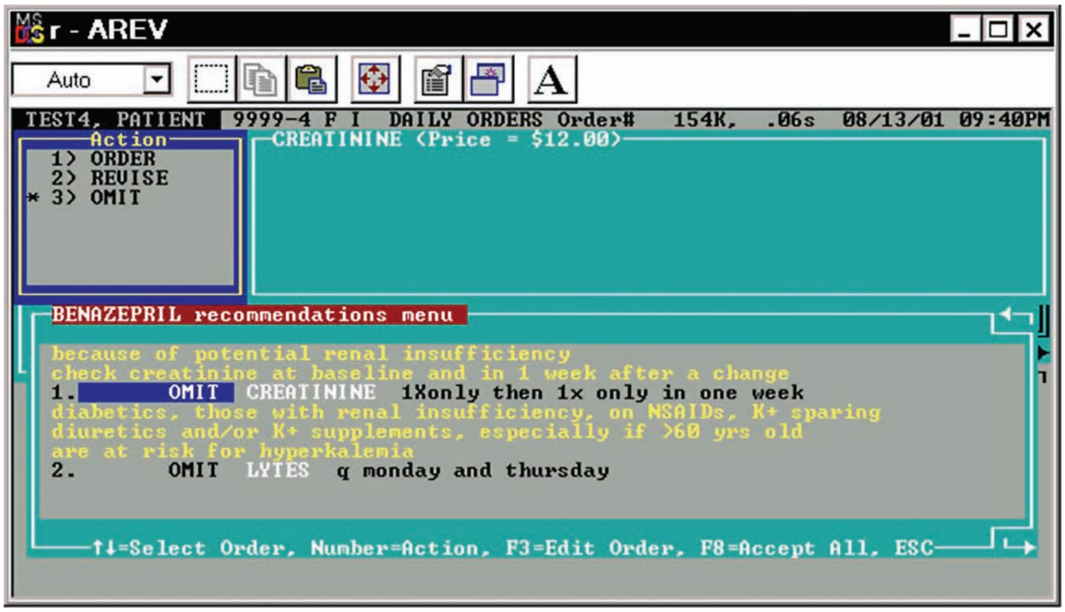
Computerized Physician Order Entry. Figure 1. A user interface for computerized physician order entry. A user would typically authenticate himself or herself, identify a patient, and proceed to enter orders, often by navigating through a series of forms, each of which might facilitate a certain type of order. This screenshot shows form fields that allow entry of orders for drugs, diagnostic tests, and other types of orders (e.g., nursing). The visual separation of types of orders into categories is done primarily for the user's convenience and organization; it may or may not reflect underlying data models.

Once a session is completed, the orders generated lead to action. This may occur via simple printing of the orders or through electronic delivery to remote locations, such as a radiology department, consultant's office, laboratory, or pharmacy. Some orders, such as for prescriptions for drugs or lifestyle changes, may be provided to patients for direct implementation or delivery elsewhere. A session's orders are then typically archived in the EHR. Consistent with traditional medical documentation, orders from CPOE are generated once and cannot be modified or deleted once finalized, though what is being ordered can often later be modified or discontinued with a subsequent order.

CPOE has been developed and implemented for a variety of reasons. Many advantages have been postulated, including rectification of substantial legibility problems with handwritten orders. CPOE systems have the potential to refer to all of a patient's medical history as well as all available medical knowledge, to improve the quality of medical care in real time, at the point of care. One of the most important potentials of CPOE is inclusion of *clinical decision support*, by which the computer can be programmed to suggest tailored orders *de novo* (e.g., for a vaccination recommended by clinical guidelines) or respond to specific orders, such as in the event of a possible drug reaction [11] or contraindication to a procedure (see Fig. 2). CPOE can reduce the rate of certain medication errors by more than half [4]. Removing the healthcare provider from electronic order entry, or moving CPOE outside the point of care, could negate these large potential benefits.

Some institutions are starting with, focus, or limit their computer-based development to electronic prescribing, or "e-prescribing." This is a form of CPOE. E-prescribing is targeted especially because prescribing is frequent, can be targeted by CPOE algorithms, and represents a most common form of medical error [13,18]. In the US, the Institute of Medicine has recommended e-prescribing of drugs, in conjunction with clinical decision support [9].

Medical orders symbolize but also allow and direct the operations behind medical care. By encoding and electronically documenting orders, CPOE improves capabilities to assess and improve quality of care and to conduct clinical research related to diagnosis and treatment. Medical practices, governments, and other authorities can gather and study data from CPOE systems to improve knowledge about how medical



```
TEST4, PATIENT  9999-4 F I   DAILY ORDERS Order#   154K,  .06s  08/13/01 09:40PM
  Action          CREATININE (Price = $12.00)
   1) ORDER
   2) REVISE
 * 3) OMIT

 BENAZEPRIL recommendations menu
 because of potential renal insufficiency
 check creatinine at baseline and in 1 week after a change
 1.          OMIT  CREATININE  1Xonly then 1x only in one week
 diabetics, those with renal insufficiency, on NSAIDs, K+ sparing
 diuretics and/or K+ supplements, especially if >60 yrs old
 are at risk for hyperkalemia
 2.          OMIT  LYTES   q monday and thursday

 ↑↓=Select Order, Number=Action, F3=Edit Order, F8=Accept All, ESC
```

Computerized Physician Order Entry. Figure 2. Example of clinical decision support. In this instance, the user has prescribed benazepril. The software responds, as shown, by prompting the user to decide about ordering blood tests, to monitor for possible side effects from the drug. To enable this capability, the application has been programmed with clinical rules that combine guidelines or medical knowledge with this patient's medical history, evaluation, or treatment.

care is formulated and delivered. CPOE can also facilitate billing processes that depend on orders, such as for certain procedures or drugs. Query languages or systems must be designed to accommodate data models used for CPOE.

Customization of CPOE systems may allow individual providers or groups of providers to create templates or order sets, which are groups of orders often used or often used together. This could save time, improve standardization, and improve care.

CPOE does have costs, risks, and unintended consequences [5]. A CPOE system must be developed thoughtfully and be maintained frequently and regularly, to ensure that it accommodates the latest tests, treatments, and guidelines, both locally and more broadly. If an institution does not stock a particular drug, the system might not allow that drug to be ordered or might at least alert the provider about the issue. Institutional changes and policies that can affect CPOE are frequent and so must lead to corresponding modifications to the CPOE system.

CPOE can increase the time required to generate a medical order [3,15]. Studies of this have reported mixed findings, with increases in some and decreases in others. Increased time for initial learning and ongoing use can cause dissatisfaction among providers and even complete failures of systems. Increasing time to generate or implement orders can have adverse clinical effects. Adverse effects might be expected especially in emergencies or acute care, when life-saving drugs may be needed rapidly. Errors in processing electronic orders could also be expected to lead to adverse effects in at least some cases. In 2005, Koppel et al. reported that one CPOE system design often facilitated medication-related errors, such as by providing inadequate views of medications and increasing inappropriate dosing and incompatible orders [10].

If not implemented effectively, increased use of computers in healthcare might distract providers, causing them to spend less time with patients or decrease patient's satisfaction [7,19]. This could have an adverse effect on patient-provider relationship or patient's health. Provider-to-provider communication might also suffer if appropriate internal communications systems are not used.

Clinical decision support, a key feature of CPOE systems, can backfire by presenting too many or inappropriate alerts. Effective solutions to "alert overload" are not yet well developed or widespread, though some solutions have been discussed [18]. Several recent studies of interventions in decision support have had negative results and require further investigation.

Financial costs of implementing CPOE can be high, especially initially. Developing cost-effectiveness analyses of EHRs and CPOE systems are thus complex, because benefits or harms can occur much later than implementation of a system and later than the time of initial care or clinical presentation.

Moving orders from paper to computers has created situations that require new handling. Computer programs, for example, must know the authority of the authenticated user and whether the user has permission to generate the requested orders. This need also exists with paper systems but is handled in those environments by people, rather than computers. In addition, the use of templates or order sets has not been heavily studied. Templates may in some cases decrease quality of care if they are adopted hastily or used in the wrong setting. In ultimately pooling or sharing data across institutions, it will be important to use standards and customary terminologies to represent orders.

Technical Issues

"Prescription" is another term for medical order, though the term is used conventionally to refer to providing instructions to patients. The structure of a traditional drug prescription provides a useful framework for understanding the primary components of medical orders. Drug orders have a superscription (including timestamp and patient's identifier), inscription (name and amount or strength of ingredient), subscription (formulation or method to prepare), and *signa* ("sig," or directions including route and frequency). Orders of any other type have analogous components, though some may have additional or somewhat different components. CPOE systems should handle components of orders with agility. Below are discussed a few key technical issues that present themselves in the design, study, and implementation of CPOE systems.

Data Models for CPOE. One must consider what data model would best support CPOE. For example, should orders be categorized and, if so, how? Many institutions have found that categories of orders, such as laboratory, consultative, pharmacy, nursing, and radiological, are clinically and informationally logical but may also be necessary from the standpoint of linking disparate data systems. A key goal in the design is the ability to accommodate future expansion of order types and categories even before those types are developed or identified. This can prompt a somewhat "flat" model, in which the nature, type, or category of an order is a value of a database field, rather than a field or variable itself. Other important aspects of orders that may have implications for the data model are the indication for the order, urgency (i.e., when it should be implemented), and who is expected to implement it.

Standardization of text in an order can be helpful for both accuracy of implementation and research. For example, an order that can have multiple forms, such as "take two 40-mg tablets by mouth twice daily" and "take one 80-mg tablet by mouth every 12 hours" can complicate both clinical care and research. Allowing providers to add narratives or free text is essential for tailoring to patient's needs, but this demands effective handling in data storage and clinical decision support. A system that can standardize the order accurately without hindering the user's experience is desirable but challenging to create. Standardization of terminology used in orders should accommodate query systems. One difficulty is that each type of order – such as for a drug, diet, radiological procedure, or laboratory test – may have unique "domains" or components. For example, only drugs have a dose, yet the dose may need to be a discrete, searchable component of a query. Therefore, an ideal data model can handle all types of orders, as well as new types, but it can also identify and distinguish between various values of key components of orders, regardless of how widespread those components are across various types of orders.

Order sets can often translate directly into a group of individual orders, but users often desire the ability to customize order sets. This may mean maintaining a base of order sets but also the customizations that are unique to each provider or role. Order sets also need to be integrated with any available decision support systems, and many providers seek to share customized order sets with each other. There are thus aspects of order sets that pertain to the system itself and to particular patients, groups of providers, and individual providers. Associated with each order set is also generally information about conditions under which the order set applies, such as a diagnosis, age group, or other criteria found in a clinical guideline – hence the possible need for order sets to be linked to ontologies or terminology systems that in turn link to such guidelines.

Specific types of orders often require further processing or delivery to specific clinical departments. Therefore, the processing needed must be encoded into the system, though it might be a part of the main data engine more than a core part of the data model or record. In any case, if laboratory orders, for example, need to be delivered to the laboratory, then the data system must support this well enough so that all laboratory orders – and only laboratory orders – are processed in this way. Similarly, systems that notify particular professionals or departments should be modular enough that those systems can be updated readily as personnel or even departments change.

An audit trail is important for documentation, accreditation, and quality and safety of care. Included in the data system should be a method to indicate formally not just who created a record and when, but what happened to the record: where it was sent and who accessed it later. Whether this is part of CPOE or the larger records system may depend on the circumstances and design of a system. Many alert systems that stem from orders do not currently provide effective prioritization, so this is an area of important research.

Key Applications

CPOE continues to undergo development and will for the foreseeable future. Due to the difficulty and time required to generate electronic orders, various forms of data entry are being explored. These include transcription with or without voice recognition and input using portable devices, digital pens, or tablets. Due to capability for electronic communication of orders, development is also occurring in remote areas or environments with limited access to healthcare, such as for rural or homebound patients.

CPOE is undergoing significant development especially in the US, the United Kingdom and other parts of Europe, Asia, Australia, and New Zealand. It can be expected to grow throughout the world, even as developing countries create EHR systems.

Future Directions

The largest looming issues for CPOE are how to maximize efficiency of data entry and effectiveness of decision support in the complex environment. There are also unmet needs for CPOE to be linked to access to general medical knowledge. Health policy, attention to quality, patient safety, and reimbursement will likely gain importance in driving uses of CPOE, especially in areas where its use is currently low. The precise roles, usefulness, impact, and specifications of incentives for healthcare providers to adopt health information technologies such as CPOE are not yet clear.

Cross-references

▶ Clinical Decision Support
▶ Data Acquisition
▶ Electronic Health Record

Recommended Reading

1. Agency for Healthcare Research and Quality. AHRQ National Resource Center for Health Information Technology. 2007. Available online at: http://healthit.ahrq.gov/ (accessed on August 29, 2007).
2. Ash J.S., Gorman P.N., Seshadri V., and Hersh W.R. Computerized physician order entry in U.S. hospitals: results of a 2002 survey. J. Am. Med. Inform. Assoc., 11(2):95–99, 2004.
3. Bates D.W., Boyle D.L., and Teich J.M. Impact of computerized physician order entry on physician time. In Proc. Annual Symp. on Computer Applications in Medical Care, 1994, p. 996.
4. Bates D.W., Leape L.L., Cullen D.J., et al. Effect of computerized physician order entry and a team intervention on prevention of serious medication errors. JAMA, 280(15):1311–1316, 1998.
5. Campbell E.M., Sittig D.F., Ash J.S., Guappone K.P., and Dykstra R.H. Types of unintended consequences related to computerized provider order entry. J. Am. Med. Inform. Assoc., 13(5): 547–556, 2006.
6. Certification Commission for Healthcare Information Technology. 2007. Available online at: http://www.cchit.org/ (accessed on August 29, 2007).
7. Frankel R., Altschuler A., George S., et al. Effects of exam-room computing on clinician-patient communication: a longitudinal qualitative study. J. Gen. Intern. Med., 20(8):677–682, 2005.
8. Geissbuhler A. and Miller R.A. A new approach to the implementation of direct care-provider order entry. In Proc. AMIA Annual Fall Symposium, 1996, pp. 689–693.
9. Institute of Medicine. Crossing the Quality Chasm: A New Health System for the 21st Century. The National Academies Press, Washington, DC, 2001.
10. Koppel R., Metlay J.P., Cohen A., et al. Role of computerized physician order entry systems in facilitating medication errors. JAMA, 293(10):1197–1203, 2005.
11. Kuperman G.J., Bobb A., Payne T.H., et al. Medication-related clinical decision support in computerized provider order entry systems: a review. J. Am. Med. Inform. Assoc., 14(1):29–40, 2007.
12. Massaro T.A. Introducing physician order entry at a major academic medical center: I. Impact on organizational culture and behavior. Acad. Med., 68(1):20–25, 1993.
13. Miller R.A., Gardner R.M., Johnson K.B., and Hripcsak G. Clinical decision support and electronic prescribing systems: a time for responsible thought and action. J. Am. Med. Inform. Assoc., 12(4):403–409, 2005.
14. Overhage J.M., Mamlin B., Warvel J., Warvel J., Tierney W., and McDonald C.J. A tool for provider interaction during patient care: G-CARE. In Proc. Annual Symp. on Computer Applications in Medical Care, 1995, pp. 178–182.
15. Overhage J.M., Perkins S., Tierney W.M., and McDonald C.J. Controlled trial of direct physician order entry: effects on physicians' time utilization in ambulatory primary care internal medicine practices. J. Am. Med. Inform. Assoc., 8(4):361–371, 2001.
16. Payne T.H. The transition to automated practitioner order entry in a teaching hospital: the VA Puget Sound experience. In Proc. AMIA Annual Symposium, 1999, pp. 589–593.
17. Sittig D.F. and Stead W.W. Computer-based physician order entry: the state of the art. J. Am. Med. Inform. Assoc., 1(2): 108–123, 1994.
18. Teich J.M., Osheroff J.A., Pifer E.A., Sittig D.F., and Jenders R.A. Clinical decision support in electronic prescribing: recommendations and an action plan: report of the joint clinical decision support workgroup. J. Am. Med. Inform. Assoc., 12(4):365–376, 2005.
19. Weiner M. and Biondich P. The influence of information technology on patient-physician relationships. J. Gen. Intern. Med., 21(Suppl 1):S35–S39, 2006.
20. Weir C, Lincoln M, Roscoe D, Turner C, and Moreshead G. Dimensions associated with successful implementation of a hospital based integrated order entry system. In Proc. Annual Symp. on Computer Applications in Medical Care, 1994, pp. 653–657.
21. Wells S., Ashton T., and Jackson R. Electronic clinical decision support. 2005. Updated October 2005. Available via Internet at: http://www.hpm.org/survey/nz/a6/2 (accessed on August 29, 2007).

Computerized Provider Order Entry

▶ Computerized Physician Order Entry

Concept Languages

▶ Description Logics

Conceptual Data Model

▶ Semantic Data Model

Conceptual Image Data Model

▶ Image Content Modeling

Conceptual Model

▶ Semantic Data Model

Conceptual Modeling

▶ Semantic Modeling for Geographic Information Systems

Conceptual Modeling for Geographic Information System

▶ Semantic Modeling for Geographic Information Systems

Conceptual Modeling for Spatio-Temporal Applications

▶ Semantic Modeling for Geographic Information Systems

Conceptual Schema Design

Alexander Borgida[1], John Mylopoulos[2]
[1]Rutgers University, New Brunswick, NJ, USA
[2]University of Trento, Trento, Italy

Definition

Conceptual schema design is the process of generating a description of the contents of a database in high-level terms that are natural and direct for users of the database. The process takes as input information requirements for the applications that will use the database, and produces a schema expressed in a conceptual modeling notation, such as the Extended Entity-Relationship (EER) Data Model or UML class diagrams. The challenges in designing a conceptual schema include: (i) turning informal information requirements into a *cognitive model* that describes unambiguously and completely the contents of the database-to-be; and (ii) using the constructs of a data modeling language appropriately to generate from the cognitive model a conceptual schema that reflects it as accurately as possible.

Historical Background

The history of conceptual schema design is intimately intertwined with that of conceptual data models (aka semantic data models). In fact, for many years researchers focused on the design of suitable *languages* for conceptual schemas, paying little attention to the design process itself. Jean-Raymond Abrial proposed the *binary semantic model* in 1974 [1], shortly followed by Peter Chen's *entity-relationship model* (ER for short) [4]. Both were intended as advances over logical data models proposed only a few years earlier, and both emphasized the need to model more naturally the contents of a database. The ER model and its extensions were relatively easy to map to logical schemas for relational databases, making EER [9] the first conceptual modeling notation to be used widely by practitioners. On the other hand, Abrial's semantic model was more akin to object-oriented data models that became popular more than a decade later.

The advent of object-oriented software analysis techniques in the late 1980s revived interest in object-oriented data modeling and led to a number of proposals. Some of these, notably OMT [7] adopted many ideas from the EER model. These ideas were consolidated into the *Unified Modeling Language* (UML), specifically UML class diagrams.

The process of designing conceptual schemas by using such modeling languages was not studied until the late 1970s, see for instance [8]. In the early 1980s, the DATAID project proposed a state-of-the-art design process for databases, including conceptual schema design [2].

Throughout this history, research on knowledge representation in Artificial Intelligence (AI) has advanced a set of concepts that overlaps with those of conceptual data models. Notably, semantic networks, first proposed in the 1960s, were founded on the notions of *concept*, *link* and *isA* hierarchy (analogously to *entity*, *relationship* and *generalization* for the EER model). The formal treatment of these notations has led to modern modeling languages such as Description Logics, including OWL, for capturing the semantics of web data. In fact, Description Logics have been shown to be able to capture the precise semantics of conceptual modeling notations such as EER diagrams and UML class diagrams.

Another important recent development has been the rise of *ontological analysis*. An ontology is a specification of a conceptualization of a domain. As such, an ontology offers a set of concepts for modeling an application, and foundational ontologies strive to uncover appropriate cognitive primitives with which to describe and critique conceptual modeling notations [5]. Foundational ontologies have been used to analyze the appropriate use of EER constructs [10] and UML [6]. Based on this work, a two-phase perspective is adopted here by distinguishing between the design of a *cognitive model* based on cognitive primitives, and the design of a corresponding *conceptual schema* that is based on the constructs of a conceptual model such as the EER or UML class diagrams.

Foundations

Building a cognitive model. Information requirements for a database-to-be are generally expressed informally, based on multiple sources (e.g., applications/queries that need to be supported, existing paper and computerized systems). Information requirements describe some part of the world, hereafter the *application domain* (or *universe of discourse*). A *cognitive model* is a human conceptualization of this domain, described in terms of cognitive primitives that underlie human cognition. The following are some of the most important among these primitives.

An *object* is anything one may want to talk about, and often represents an *individual* ("my dog," "math422"). Usually, individual objects persist over many states of the application domain. Moreover, they have *qualities* (such as size, weight), and can be related to other individuals by *relations* (e.g., "friendOf," "part of," "between"). Individuals may be concrete (such as "Janet," or "that tree"), abstract (e.g., "the number 12," "cs422"), or even hypothetical (e.g., "Santa," "the king of the USA"). Individuals have a notion of *identity*, allowing them, for example, to be distinguished and counted. For some individuals such as "Janet," identity is an intrinsic notion that distinguishes her from all other objects. *Values* are special individuals whose identity is determined by their structure and properties (e.g., numbers, sets, lists and tuples). The number "7," for example, is the unique number that comes after "6," while "{a, b, c}" is the unique set with elements "a," "b" and "c."

Individuals can be grouped into *categories* (also called *concepts*, *types*), where each category (e.g., "Book") captures common properties of all its instances (e.g., "my DB textbook," "Ivanhoe"). Categories themselves are usefully structured into taxonomies according to their generality/specificity. For instance, "Book" is a specialization of "LibraryMaterial," along with "Journal" and "DVD." Moreover, "Book" has its own specializations, such as "Hardback," "Paperback."

Many categories (e.g., "Person") are *primitive*, in the sense that they don't have a definition. By implication, there is no algorithm to determine whether a given individual object is an instance of such a category – one must be told explicitly such facts. Other categories are *defined*, in the sense that instances can be recognized based on some rule. For example, an instance of "Teenager" can be recognized given an instance of "Person" and its age.

Relations relate two or more objects, for example "book45 is on loan to Lynn," or "Trento is between Bolzano and Verona" and can represent among others a dependence of some sort between these objects. Each relation is characterized by a positive integer n–its *arity*–representing the number of objects being related. In the example above, "onLoanTo" is binary, while "between" is ternary. Predicate logic notation is used to express specific relations between individuals, e.g., "between(Trento,Bolazano,Verona)." Like their individual counterparts, relations can be grouped into relation categories. In turn, relation categories can be organized into subcategory hierarchies: "brotherOf" and "sisterOf" are subcategories of "siblingOf," which in turn is a subcategory of "relativeOf." A binary relation category has a *domain* and a *range*. A cognitive model can specify arbitrary constraints between relations and categories, which describe the valid states ("semantics") of the application domain. Cardinality constraints, for instance, specify upper/lower bounds on how many instances of the range of a relation can be associated to an instance of the domain, and vice versa.

A subtle complexity arises when one wants to describe information *about* a relation, for example when did it become or ceased to be true. In such cases the modeler can resort to *reification*. For example, the reification of "Trento is between Bolzano and Verona" consists of creating a new individual, "btw73" which is an instance of the category "Between" and is related to its three arguments via three functional relations: "refersTo (btw73,Trento)," "source(btw73,Bolzano)," "destination (btw73,Verona)." Note that this representation allows another instance of "Between," say "btw22," with functions to the same three individuals. To avoid this redundancy, one needs suitable constraints on "Between." One can now model other information about such reified relations, e.g., "believes(yannis,btw73)."

There are several categories of relations that deserve special consideration.

PartOf (with inverse *hasPart*) represents the part-whole relation that allows composite conceptualizations consisting of simpler parts. PartOf actually represents several distinct relations with different formal properties [5]. Most prominent among them are the relations *componentOf* (e.g., cover is a component of book) and *memberOf* (e.g., player member of a team). PartOf is frequently confused with other relations, such as *containment, connectedness, hasLocation.* A useful diagnostic test for this confusion is to check that if A is part of B, and B is damaged, then A is also considered damaged. Note how "love-note placed inside book" fails this test.

The relation between an object and a category is often called *instanceOf*, and the set of all instances of a category are called its *extension*. The *isA* (*subcategory*) relation represents a taxonomic ordering between categories, e.g., "isA(HardcoverBook,Book)." The isA relation is transitive and anti-symmetric and interacts with instanceOf in an important way: if "isA(A, B)" and "instanceOf(x,A)" then "instanceOf(x,B)." Any general statements one associates with a category, apply to all its specializations. For example, "every book has a title" will automatically apply to subcategories ("every HardcoverBook has a title"); this is called *inheritance* of constraints. In many cases, a group of subcategories are mutually *disjoint* (e.g., "Hardcover," "Paperback," are disjoint subcategories of the category "Book"). Sometimes, a set of subcategories *covers* their common parent category in the sense that every instance of the latter is also an instance of at least one subcategory (for example, "Male," "Female" cover "Person"). When a collection of subcategories partitions a parent category, it is often because some relation (e.g., "gender") takes on a single value from an enumerated set (e.g., {"M," "F"}).

When building an isA hierarchy, it is useful to start by building a backbone consisting of primitive, disjoint concepts that describe the basic categories of individuals in the application domain. Some categories can be distinguished from others by their *rigidity* property: an instance of the category remains an instance throughout its lifetime. For example, "Person" is such a category (in the sense of "once a person, always a person"), but "Student" is not. Once a backbone taxonomy has been constructed, other categories, such as role categories, can be added to the hierarchy as specializations of the categories. Welty and Guarino [11] present a principled approach to the construction of taxonomies. A useful rule for building meaningful isA hierarchies is to ensure that the children of any category are all at the same level of granularity. A leaf category in an isA hierarchy should be further refined if it has instances that can be usefully grouped into sub-categories, which participate in new relations or which are subject to new constraints.

Categories may also be seen as objects, and can then be grouped into meta-categories that capture common meta-properties of their instances. For example, the meta-category "LibraryMaterialType" has instances such as "Book," "Hardback" and "DVD," and may have an integer-valued meta-property such as "number-on-order."

In most worlds there are not just enduring objects but also occurrences of events, activities, processes, etc., such as borrowing, renewing or returning a book. These phenomena are called *events*. An event can be described from a number of perspectives: First, there are the *participants* in an event: the material borrowed, the patron who did the borrowing, the library from which the material was borrowed, the date when the material is required to be returned, etc. Often these participants are given names describing the *role* they play in the event: "borrower," "lender," "due date." Second, every event takes place in time, so its temporal aspects can be represented: starting and possibly ending time if it is of a long duration, cyclic nature, etc. Third, events may also have parts – the sub-events that need to take place (e.g., taking the book to the counter, proffering the library card,...). There may also be special relations, such as causality or temporal precedence that hold among events.

In database modeling, one often ignores events because they are transient, while the database is supposed to capture persistence. However, events are in fact present in the background: relations between objects other than "partOf" are usually established or terminated by events. The "onLoan" relation, for example, is created by a "borrow" event and terminated by a "return" event. And values for many qualities (e.g., the size or weight of an object) are established through events representing acts of observation. As a result, relations often carry information about the events. Thus information about the "borrow" activity's participants is present in the arguments of the "onLoan" relation. And since "renew" shares the main participants of "borrow," its traces can also be attached to "onLoan," through an additional temporal argument, such as "lastRenewedOn."

Note that it is application requirements that determine the level of detail to be maintained in a cognitive model. For example, whether or not one records the time when the borrowing and renewal occurred, or whether it is sufficient to have a "dueDate" attribute on the "onLoan" relation. In addition, the details of the subparts of "borrow" will very likely be suppressed. On the other hand, semantic relations between events, such as the fact that a book renewal can only occur after the book has been borrowed, do need to be captured.

Because databases often have multiple sets of users, there may be several conflicting interpretations of terms and information needs. The important process of reconciling such conflicts, known in part as "view integration" is not addressed in this entry.

From a cognitive model to a conceptual schema. The above account has focused on the construction of a model that captures information requirements in terms of cognitive primitives, with constraints expressed in a possibly very rich language. Every effort was made to keep the modeling and methodology independent of a particular modeling language. Next, one must tackle the problem of producing a conceptual schema expressed in some particular formal notation, in this case the EER. Comparable discussions apply if the target was UML class diagrams, or even OWL ontologies.

The basic mapping is quite straightforward: *categories of individuals* in the conceptual model are mapped to ER *entity sets*; relation categories in the conceptual model are modeled directly as relationship sets, with the participating entities playing (potentially named) roles in the relationship set. Qualities and values related to individuals by binary relations, or appearing as arguments of relations become *attributes.* Cardinality constraints of the cognitive model are mapped directly to the conceptual schema.

One complex aspect of EER schema development is the definition of *keys* consisting of one or more attributes that uniquely distinguish each individual instance of an entity set. Moreover, the values of these attributes must be stable/unchanging over time. For example, "isbnNr" or "callNumber" would be a natural key attributes for "Book." Globally unique identifiers are actually relatively rare. Instead, entities are often identified with the help of intermediate relationships (and attributes). For example, "BookCopy," has attribute "copyNr," which surely does not identify a particular book copy; but if "BookCopy" is represented as a *weak entity*, related to "Book" via the "copyOf" identifying relationship then "BookCopy" will have a composite identifier. In other situations where one would need a large set of attributes to identify an entity, and especially if these attributes are not under the control of database administrators, the designer may introduce a *new* surrogate entity attribute specifically for identification purposes (e.g., "studentId" for the entity set "Student").

Sub-categories, with possible disjoint and coverage constraints, are represented in a direct manner in EER since it supports these notions. Significantly, in most version of EER key attribute(s) can only be specified for the top-most entity in a hierarchy, and must then be inherited by all sub-entities. Therefore one cannot have "Employee" with key "ssn," while sub-class "TeachingAssistant" has key "studentId." The reason for this is to avoid multiple ways of referring to what would otherwise be the same individual.

Since the EER model supports n-ary relationships, reified relationships are normally only required in case the model needs to make "meta" statements about relationships, e.g., recording that a particular loan was verified by a clerk. In variants of the EER that allow aggregate relationships, this is modeled by relating entity "Clerk" via relationship "hasVerified" to the aggregate representing the "lentTo" relationship. In impoverished variants of EER that do not support aggregates, this can be encoded using weak entities that reify the relationship.

The EER notation (in constrast to UML, say), does not provide support for distinguishing *partOf* relationships, nor for relationship hierarchies. However, the designer may encode these using reified relationships.

Conceptual Schema Design in Practice

There is a plethora of commercial tools for conceptual schema design based on the EER model or UML class diagrams. These support the drawing, documenting and layout of conceptual schemas, and even the automatic generation of standard logical (relational) schemas to support database design.

More advanced tools, such as icom (http://www.inf.unibz.it/~franconi/icom/), allow not just the drawing of conceptual schemas, but their translation to formal logic. Advantages of such tools include (i) precise, formal semantics for all the constructs of the conceptual model; (ii) the ability to check the conceptual schema for consistency–an issue which is particularly interesting in the case of *finite models*; (iii) the ability to augment a conceptual schema with constraints expressed in the underlying logic.

Key Applications

Conceptual schema design is the first–and for many the most important–step in the design of any database.

Future Directions

One of the major research challenges of the new century is data integration, and the semantics of data captured in conceptual models has been shown to form a useful foundation for merging multiple information sources. In this context, one can imagine using a conceptual schema as the mediating schema to access different database sources.

The advent of the Web has made databases a public resource that can be shared world-wide. In such a setting, where the user may know nothing about a database she is accessing, the issues that dominate are (i) encapsulating the semantics of data along with the data, so that the user can interpret the data; (ii) ensuring data quality, so that the user can determine whether the data are suitable for her purposes; (iii) ensuring compliance with privacy, security and governance regulations. In turn, these new requirements are going to redefine the scope of database design in general and conceptual schema design in particular.

Specifically, extended conceptual modeling languages and conceptual schema design techniques are envisioned where quality, privacy, security and governance policies can be expressed explicitly and can be accommodated during the design process to produce conceptual schemas that are well-suited to address such concerns. We also envision extensions to conceptual modeling languages that introduce primitive concepts for modeling dynamic, intentional and social aspects of an application domain. These aspects constitute an important component of the semantics of any database and a prerequisite for dealing with data quality and regulation compliance.

Cross-references

▶ Conceptual Data Model
▶ Description Logics
▶ Logical Schema Design
▶ Schema Integration
▶ Semantic Data Model

Recommended Reading

1. Abrial J-R. Data Semantics. In Data Management Systems, K. Koffeman North-Holland, Amsterdam, 1974.
2. Atzeni P., Ceri S., Paraboschi S., and Torlone R. Database Systems: Concepts, Languages & Architectures. McGraw Hill, New York, 1999.
3. Batini C., Ceri S., and Navathe S. Conceptual Database Design. Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1991.
4. Chen P.P. The entity-relationship model – toward a unified view of data. ACM Trans. Database Syst., 1(1):9–36, 1976.
5. Gangemi A., Guarino A., Masolo C., Oltramari A., and Schneider L. Sweetening ontologies with DOLCE. In Proc. 12th Int. Conf. Knowledge Eng. and Knowledge Management: Ontologies and the Semantic Web, 2002, pp. 166–181.
6. Guizzardi G., Herre H., and Wagner G. Towards Ontological Foundations for UML Conceptual Models. In Proc. Confederated Int. Conf. DOA, CoopIS and ODBASE, 2002, pp. 1100–1117.
7. Rumbaugh J., Blaha M., Premerlani W., Eddy F., and Lorensen W. Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, NJ, 1991.
8. Sakai H. Entity-relationship approach to the conceptual schema design. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1980, pp. 1–8.
9. Teorey T., Yang D., and Fry J. A logical design methodology for relational databases using the extended entity-relationship model. ACM Comput. Surv., 18(2):197–222, 1986.
10. Wand Y., Storey V.C., and Weber R. An ontological analysis of the relationship construct in conceptual modeling. ACM Trans. Database Syst., 24(4):494–528, 1999.
11. Welty C. and Guarino N. Supporting ontological analysis of taxonomic relationships. Data Knowl. Eng., 39:51–74, 2001.

Conceptual Schemas

▶ Resource Description Framework (RDF) Schema (RDFS)

Concurrency Control

▶ Concurrency Control – Traditional Approaches
▶ Correctness Criteria Beyond Serializability
▶ Performance Analysis of Transaction Processing Systems

Concurrency Control – Traditional Approaches

Gottfried Vossen
University of Münster, Münster, Germany

Synonyms

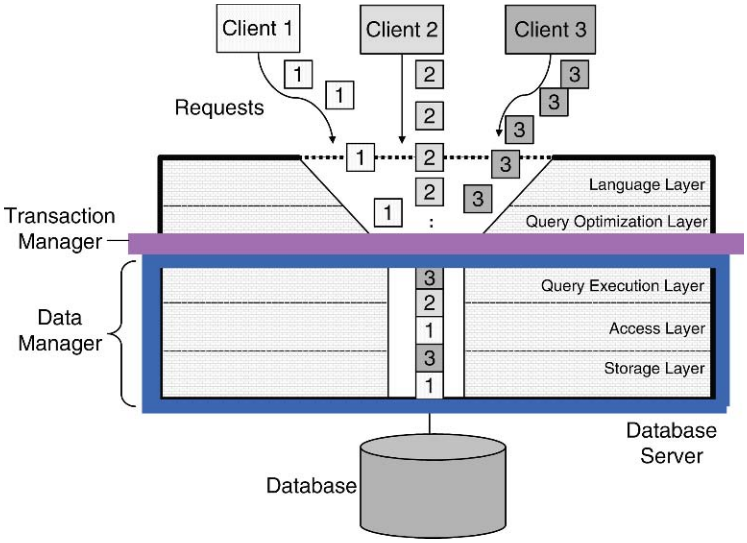
Concurrency control; Transaction execution; Scheduling; Page locking; Two-phase-locking

Definition

The core requirement on a transactional server is to provide the ACID properties for transactions, which requires that the server includes a *concurrency control* component as well as a *recovery component*. Concurrency control essentially guarantees the isolation properties of transactions, by giving each transaction the impression that it operates alone on the underlying database and, more technically, by providing serializable executions. To achieve serializability, a number of algorithms have been proposed. Traditional approaches focus on the read-write model of transactions and devise numerous ways for correctly scheduling read-write transactions. Most practical solutions employ a variant of the two-phase locking protocol.

Key Points

Concurrency control for transactions is done by the *transaction manager* of a database system and within that component by a *scheduler*. The scheduler implements an algorithm that takes operations from active transactions and places them in an interleaving or *schedule* that must obey the correctness criterion of serializability. As illustrated in Fig. 1, which indicates the "positioning" of a transaction manager within the multi-layer architecture of a database system, the scheduler receives steps from multiple transactions, one after the other, and tries to attach them to the schedule already output. This is possible if the new schedule is still serializable; otherwise, the scheduler can block or even reject a step (thereby causing the respective transaction to abort) [1,5,6]. The algorithm a scheduler follows must be such that serializability can be tested



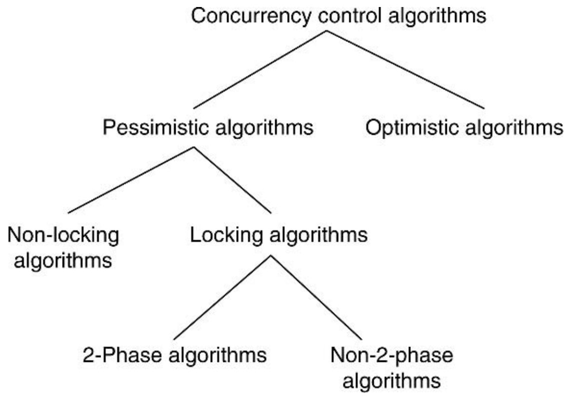
Concurrency Control – Traditional Approaches. Figure 1. Illustration of a transaction scheduler.

on the fly; moreover, it must be very efficient so that high throughput rates (typically measured in [committed] transactions per minute) can not only be achieved, but even be guaranteed.

Classification of Approaches

Scheduling algorithms for database transactions can be classified as "traditional" if they concentrate on scheduling read-write transactions; non-traditional schedulers takes semantic information into account which is not available at the syntactic layer of read and write page operations. Traditional schedulers generally fall into two categories: A scheduler is *optimistic* or aggressive if it mostly lets steps pass and rarely blocks; clearly, this bears the danger of "getting stuck" eventually when the serializability of the output can no longer be guaranteed. An optimistic scheduler is based on the assumption that conflicts between concurrent transactions are rare; it will only test from time to time whether the schedule produced so far is still serializable, and it takes appropriate measures if the schedule is not.

On the other hand, a scheduler is *pessimistic* or conservative if it mostly blocks (upon recognizing conflicts); in the extreme yet unlikely case that all transactions but one have been blocked, the output would become a serial schedule. This type of scheduler is based on the assumption that conflicts between transactions are frequent and therefore need to be constantly observed. Pessimistic schedulers can be *locking* or *non-locking* schedulers, where the idea of the former is to synchronize read or write access to shared data by using locks which can be set on and removed from data items on behalf of transactions. The intuitive meaning is that if a transaction holds a lock on a data object, the object is not available to transactions that execute concurrently. Non-locking schedulers replace locks, for example, by timestamps that are attached to transactions. Among locking schedulers, the most prominent protocol is based on a *two-phase* approach (and hence abbreviated two-phase locking or 2PL) in which, for each transaction, a first phase during which locks are obtained is strictly separated from a second phase where locks can only be released. Non-two-phase schedulers replace the two-phase property, for example, by an order in which transactions may access data objects. Figure 2 summarizes the major classes of concurrency control protocols. Beyond the approaches shown in Fig. 2, the concurrency control problem can even be broken into two subproblems, which could then be solved individually by possibly distinct protocols: (i) read operations are synchronized against write operations or vice versa; (ii) write operations are synchronized against other write operations, but not against reads. If these synchronization tasks are distinguished, a scheduler can be thought of as consisting of two components, one for each of the respective synchronization tasks. Since the two components need proper integration, such a scheduler is called a hybrid scheduler. From an application point of view, most classes of protocols surveyed above, including the hybrid ones, have not achieved great relevance in practice. Indeed, 2PL is by far the most important concurrency control protocol, since it can be implemented with low overhead, it can be extended to abstraction levels beyond pure page operations, and it has always outperformed any competing approaches [2–4].



Concurrency Control – Traditional Approaches. Figure 2. Overview of concurrency control protocol classes.

Cross-references

- ▶ B-Tree Locking
- ▶ Distributed Concurrency Control
- ▶ Locking Granularity and Locks Types
- ▶ Performance Analysis of Transaction Processing Systems
- ▶ Serializability
- ▶ Snapshot Isolation
- ▶ Two-Phase Locking

Recommended Reading

1. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, MA, 1987.

2. Bernstein P.A. and Newcomer E. Principles of Transaction Processing for the Systems Professional. Morgan Kaufmann, San Francisco, CA, 1997.
3. Claybrook B. OLTP – Online Transaction Processing Systems. Wiley, New York, 1992.
4. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1993.
5. Papadimitriou C.H. The Theory of Database Concurrency Control. Computer Science, Rockville, MD, 1986.
6. Weikum G. and Vossen G. Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2002.

Concurrency Control and Recovery

► Transaction Management

Concurrency Control Manager

Andreas Reuter[1,2]
[1]EML Research GmbH Villa Bosch, Heidelberg, Germany
[2]Technical University Kaiserslautern, Kaiserslautern, Germany

Synonyms

Concurrency control manager; Lock manager; Synchronization component

Definition

The concurrency control manager (CCM) synchronizes the concurrent access of database transactions to shared objects in the database. It is responsible for maintaining the guarantees regarding the effects of concurrent access to the shared database, i.e., it will protect each transaction from anomalies that can result from the fact that other transactions are accessing the same data at the same time. Ideally, it will make sure that the result of transactions running in parallel is identical to the result of some serial execution of the same transactions. In real applications, however, some transactions may opt for lower levels of synchronization, thus trading protection from side effects of other transactions for performance. The CCM is responsible for orchestrating all access requests issued by the transactions such that each transaction receives the level of protection it has asked for. The CCM essentially implements one of a number of different synchronization protocols, each of which ensures the correct execution of parallel transactions, while making different assumptions regarding prevalent access patterns, frequency of conflicts among concurrent transactions, percentage of aborts, etc. The protocol that most CCMs are based on is using locks for protecting database objects against (inconsistent) parallel accesses; for that reason, the CCM is often referred to as the "lock manager." Some CCMs distinguish between multiple versions of a data object (e.g., current version, previous version) in order to increase the level of parallelism [2].

Key Points

The CCM monitors all access requests issued by higher-level components, be it to the primary data (tuples, records), or to access paths, directory data, etc. on behalf of the database transactions. This information (transaction **X** accesses object **O** in order to perform action **A** at time **T**) is employed in different ways, depending on the synchronization protocol used. In case of locking protocols, each access request has to be explicitly granted by the CCM. If no conflict will arise by performing action A on object O, the access request is granted. If, however, a conflict is detected (e.g., A is an update request, and some other transaction Y is already updating object O), the request is not granted, and the CCM will record the fact that X has to wait for the completion of transaction Y. In case of optimistic protocols, the requests are granted right away, but when X wants to commit, all its accesses are checked for conflicts with accesses performed by other transactions that are either still running or have committed while X was active. In those situations the time T of the access request is relevant. So the basic data structure maintained by the CCM is a table of accesses/access requests. For locking protocols, the CCM will also maintain a list of which transaction waits for the completion of which other transaction(s). That list is tested by the CCM for deadlocks. If a deadlock or, in case of optimistic protocols, a conflict is detected, the CCM decides which transaction will be aborted. It then informs the transaction manager, who will initiate the abort; rollback of the operations is performed by the recovery manager [1].

Cross-references

► Degrees of Consistency
► Dependency

- ► Isolation
- ► Locking
- ► Scheduling
- ► Synchronization

Recommended Reading

1. Gray J. and Reuter A. Transaction Processing – Concepts and Techniques. Morgan Kaufmann, San Mateo, 1993.
2. Weikum G. and Vossen G. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control. Morgan Kaufmann, San Mateo, 2001.

Condition Event Nets

- ► Petri Nets

Conditional Branching

- ► OR-Split

Conditional Routing

- ► OR-Split

Conditional Tables

Gösta Grahne
Concordia University, Montreal, QC, Canada

Synonyms

C-tables; Extended relations

Definition

A conditional table [4] generalizes relations in two ways. First, in the entries in the columns, variables, representing unknown values, are allowed in addition to the usual constants. The second generalization is that each tuple is associated with a condition, which is a Boolean combination of atoms of the form $x = y$, $x = a$, $a = b$, for x, y null values (variables), and a, b constants. A conditional table essentially represents an existentially quantified function free first order theory.

Formally, let *con* be a countably infinite set of constants, and *var* be a countably infinite set of variables, disjoint from *con*. Let U be a finite set of attributes, and $R \subseteq U$ a relational schema. A tuple in a c-table over R is a mapping from R, and a special attribute, denoted φ, to $con \cup var \cup \beta$, where β is the set of all Boolean combinations of equality atoms, as above. Every attribute in R maps to a variable or a constant, and φ maps to β. In a multirelational database schema, there are multi-tables, meaning in effect that variables can be shared between tables (just as constants are). An example of a 2-multitable is shown below. The conditions φ are all *true*, and it so happens that the two tables do not share any variables.

T_1	A	B	C	φ
	a	*b*	*x*	*true*
	e	*f*	*g*	*true*

T_2	B	C	D	φ
	y	*c*	*d*	*true*

Key Points

It is now possible to extend the complete set of regular relational operators $\{\pi,\sigma,\bowtie,\cup,-,\rho\}$ to work on c-tables. To distinguish the operators that apply to tables from the regular ones, the extended operators are accented by a dot. For instance, c-table join is denoted $\dot{\bowtie}$. The extended operators work as follows: projection $\dot{\pi}$ is the same as relational projection, except that the condition column φ can never be projected out. Selection $\dot{\sigma}_{A=a}(T)$ retains all tuples t in table T, and conjugates the condition $t(A) = a$ to $t(\varphi)$. A join $T \dot{\bowtie} T'$ is obtained by composing each tuple $t \in T$ with each tuple $t' \in T'$. The new tuple $t \cdot t'$ has condition $t(\varphi) \wedge t'(\varphi) \wedge \delta(t,t')$, where condition $\delta(t,t')$ states that the two tuples agree on the values of the join attributes. The example below serves as an illustration of this definition. The union $\dot{\cup}$ is the same as relational union, and so is renaming $\dot{\rho}$, except that the φ-column cannot be renamed. Finally, the set difference, say $T \dot{-} T'$ is obtained by retaining all tuples $t \in T$ and conjugating to them the condition stating that the tuple t differs from each tuple t' in T'. A tuple t differs from a tuple t', if it differs from t' in at least one column.

Let T_1 and T_2 be as in the figure above. The three c-tables in the figure below, illustrate the result of evaluating $\dot{\sigma}_{C=g}(T_1)$, $T_1 \dot{\bowtie} T_2$, and $\dot{\pi}_{BC}(T_2) \dot{-} \dot{\pi}_{BC}(T_1)$, respectively.

A	B	C	φ
a	b	x	$x = g$
e	f	g	$g = g$

A	B	C	D	φ
a	y	x	d	$b = y \wedge x = c$
e	y	g	d	$f = y \wedge g = c$

B	C	φ
y	c	$(y \neq b \vee c \neq x) \wedge (y \neq f \vee c \neq g)$

Note that the second tuple in the first c-table has a tautological condition. Likewise, since any two constants differ, the condition of the second tuple in the middle c-table is contradictory.

So far, nothing has been said about what the c-tables mean. In the *possible worlds* interpretation, an incomplete database is a usually infinite *set* of ordinary databases, one of which corresponds to the actual (unknown) database. Considering c-tables, they serve as finite representations of sets of possible databases. One (arbitrary) such database is obtained by instantiating the variables in the c-table to constants. Each occurrence of a particular variable is instantiated to the same constant. Formally, the instantiation is a valuation $v : con \cup var \rightarrow con$, that is identity on the constants. Valuations are extended to tuples and conditions tables in the obvious way, with the caveat that given a particular valuation v, only those tuples t for which $v(t(\varphi)) \equiv true$, are retained in $v(T)$. Consider for instance the first table above. For those valuations v, for which $v(x) = g$, there will be two tuples in $v(T)$, namely (a,b,g) and (e,f,g). For valuations v', for which $v(x) \neq g$, there will only be the tuple (e,f,g) in $v'(T)$.

The remarkable property of c-tables is that for all c-tables T and relational expressions E, it holds that $v(\dot{E}(T)) = E(v(T))$ for all valuations v. In other words, the extended algebra is a Codd sound and complete inference mechanism for c-tables. Furthermore, c-tables are closed under relational algebra, meaning that the result of applying any relational expression on any (schema-wise appropriate) c-table can be represented as another c-table. The extended algebra actually computes this representation, as was seen in the example above.

Needless to say, all of this comes with a price. Testing whether a c-table is satisfiable, that is, whether there exists at least one valuation v, such that $v(T) \neq \emptyset$ is an NP-complete problem [2]. Furthermore, even if one starts with a simple c-table where all variables are distinct, and all conditions are *true*, applying even a monotone relational expression to such a c-table can result in quite a complex table, so here again [2] satisfiability of the resulting table is NP-complete [2]. To make matters even worse, testing containment of c-tables is Π_2^p-complete. A c-table T is contained in a c-table T', if every for every valuation v of T, there exists a valuation v' of T', such that $v(T) = v(T')$. Nonetheless, c-tables possess a natural robustness. For instance, it has been shown [1,3] that the set of of possible databases defined by a set of materialized views, can be represented as a c-table.

Cross-references

- ▶ Certain (and Possible) Answers
- ▶ Incomplete Information
- ▶ Maybe answer
- ▶ Naive tables

Recommended Reading

1. Abiteboul S., Duschka O.M. Complexity of Answering Queries Using Materialized Views. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 254–263.
2. Abiteboul S., Kanellakis P.C., Grahne G. On the Representation and Querying of Sets of Possible Worlds. Theor. Comput. Sci., 78(1):158–187, 1991.
3. Grahne G., Mendelzon A.O. Tableau Techniques for Querying Information Sources through Global Schemas. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 332–347.
4. Imielinski T., Lipski W. Jr. Incomplete Information in Relational Databases. J. ACM, 31(4):761–791, 1984.

Confidentiality Protection

▶ Statistical Disclosure Limitation For Data Access

Conflict Serializability

▶ Two-Phase Locking

Conjunctive Query

Val Tannen
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

SPC query; Horn clause query

Definition

Conjunctive queries are first-order queries that both practically expressive and algorithmically relatively tractable. They were studied first in [2] and they have played an important role in database systems since then.

As a subset of the relational calculus, conjunctive queries are defined by formulae that make only use of atoms, conjunction, and existential quantification. As such they are closely related to Horn clauses and hence to logic programming. A single Datalog rule can be seen as a conjunctive query [1].

Optimization and reformulation for various purposes is quite feasible for conjunctive queries, as opposed to general relational calculus/algebra queries. The equivalence (and indeed the *containment*) of conjunctive queries is decidable, albeit NP-complete [1].

Key Points

This entry uses terminology defined in the entry Relational Calculus.

Conjunctive queries are first-order queries of a particular form: $\{\langle e_1,\ldots,e_n\rangle \mid \exists x_1\ldots x_m \psi\}$ where ψ is an (equality-free) conjunction of relational atoms, i.e., atoms of the form $R(d_1,\ldots,d_k)$ (where $d_1,\ldots,d_k$ are variables of constants). In addition, it is required that any variable among $e_1,\ldots,e_n$ must also occur in one of the relational atoms of ψ. This last condition, called *range restriction* [3] is necessary for *domain independence*, e.g., consider $\{(x, y) | R(x)\}$, and, in fact, it is also sufficient.

The semantics of a conjunctive query in an instance $\mathcal{I}$, as a particular case of first-order queries, involves assignments μ defined on the variables among $e_1,\ldots,e_n$ such that $\mathcal{I}, \mu \models \exists\, x_1..x_m\psi$. Note however that this is the same as extending μ to a *valuation* ν defined in addition on $x_1\ldots x_m$ and such that $\mathcal{I}, \nu \models \psi$. Since ψ is a conjunction of relational atoms, this amounts to ν being a *homomorphism* from ψ seen as a relational instance (the *canonical instance* associated to the query) into $\mathcal{I}$. This simple observation has many useful consequences, including some that lead to the decidability of equivalence (containment). It also leads to an alternative way of looking at conjunctive queries, related to logic programming. Here is an example of a conjunctive query in both relational calculus form and in a Prolog-like, or "rule-based," formalism, also known as a *Datalog rule* [1,3]:

$$\{(x, c, x) \mid \exists y\ R(c, y) \wedge S(c, x, y)\}\mathit{ans}(x, c, x) :$$
$$- R(c, y), S(c, x, y)$$

In the spirit of rule-based/logic programming, the output tuple of a conjunctive query is sometimes called the "head" of the query and the atom conjunction part the "body" of the query. So far, this discussion has considered only the class CQ of conjunctive without equality in the body. The class $\mathrm{CQ}^=$ which allows equalities in the body defines essentially the same queries but there are a couple of technical complications. First, the range restriction condition must be strengthened since, for example, $\{(x, y)|\exists z\ R(x)\ \wedge\ y = z\}$ is domain dependent. Therefore, for $\mathrm{CQ}^=$ it is required that any variable among $e_1,\ldots,e_n$ must equal, as a consequence of the atomic equalities in ψ, some constant, or some variable that occurs in one of the relational atoms of ψ.

$\mathrm{CQ}^=$ has the additional pleasant property (shared, in fact, with the full relational calculus) that query heads can be restricted to consist of just distinct variables.

Clearly $\mathrm{CQ} \subseteq \mathrm{CQ}^=$. The converse is "almost true." It is possible to get rid of equality atoms in a conjunctive query iff the query is *satisfiable* i.e., there exists some instance on which the query returns a nonempty answer. All the queries in CQ are satisfiable (take the canonical instance). Queries in $\mathrm{CQ}^=$ are satisfiable iff the equalities in their body do not imply the equality of distinct constants. Thus, for conjunctive queries (of both kinds) satisfiability is decidable, as opposed to general first-order queries. Now, any satisfiable query in $\mathrm{CQ}^=$ can be effectively translated into an equivalent query in CQ.

The conjunctive queries correspond to a specific fragment of the relational algebra, namely the fragment that uses only the selection, projection, and cartesian product operations. This fragment is called the *SPC algebra*. There is an effective translation that takes every conjunctive query into an equivalent SPC algebra expression. There is also an effective translation that takes every SPC algebra expression into an equivalent conjunctive query.

Via the translation to the SPC algebra it can be seen that conjunctive queries correspond closely to certain SQL programs. For example, the CQ$^{=}$ query $ans(x, y) : -R(x, z), x = c, S(x, y, z)$ corresponds to

```
select r.1, s.2
from R r, S s
where r.1=c and s.1=r.1 and r.3=s.2
```

Such SQL programs, in which the "where" clause is a conjunction of equalities arise often in practice. So, although restricted, conjunctive queries are important.

Cross-references

- ▶ Datalog
- ▶ Relational Algebra
- ▶ Relational Calculus

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases: The Logical Level. Addison Wesley, Reading, MA, 1994.
2. Chandra A.K. and Merlin P.M. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In Proc. 9th Annual ACM Symp. on Theory of Computing, 1977, pp. 77–90.
3. Ullman J.D. Principles of Database and Knowledge-Base Systems, Vol. I. Computer Science, Rockville, MD, 1988.

Connection

Sameh Elnikety
Microsoft Research, Cambridge, UK

Synonyms

Database socket

Definition

A connection is a mechanism that allows a client to issue SQL commands to a database server. In a typical usage, the client software opens a connection to the database server, and then sends SQL commands and receives responses from the server.

To open a connection, the client specifies the database server, database name, as well as the client's credentials. Opening the connection includes a handshake between the client software and the database server. The client sends its credentials, for example in the simplest form a user name and password. The server examines the credentials to authorize the connection. Further information may also be negotiated such as the specific protocol and data encoding.

Key Points

Handling and servicing connections is an important part of database servers because connections are the main source of concurrency.

Database servers limit the number of connections they can accept and may provide differentiated service to connections from high priority clients (e.g., from database administrators).

Connections are implemented using inter-process (e.g. pipes) or remote (e.g., TPC sockets) communication mechanisms. Database vendors and third-party providers supply libraries that client programs use to open connections to database servers. Several standards have emerged such as ODBC (Open Database Connectivity) [2], JDBC (Java Database Connectivity) [3], and ADO.NET (data access classes in Microsoft.NET platform) [1].

When client software uses a database system extensively, it employs a connection pool to reuse a group of open connections, allowing multiple concurrent SQL commands. Using a connection pool avoids closing and reopening connections, as well as opening too many connections that tie up resources at both ends of the connection.

Cross-references

- ▶ Session

Recommended Reading

1. Adya A., Blakeley J., Melnik S., and Muralidhar S. Anatomy of the ADO.NET entity framework. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 877–888.
2. Data Management: SQL Call Level Interface (CLI), Technical Standard C451–15/10/1993, The Open Group.
3. Sun Microsystems, Java Database Connectivity. Available at: http://java.sun.com/javase/technologies/database/

Connectionist Model

▶ Neural Networks

Consistency in Peer-to-Peer Systems

► Updates and Transactions in Peer-to-Peer Systems

Consistency Models For Replicated Data

Alan Fekete
University of Sydney, Sydney, NSW, Australia

Synonyms

Replica consistency; Memory consistency

Definition

When a distributed database system keeps several copies or replicas for a data item, at different sites, then the system may ensure that the copies are always consistent (that is, they have the same value), or the system may allow temporary discrepancy between the copies. Even if the copies are not the same, the algorithms that manage the data may be able to hide the discrepancies from clients. A consistency model defines the extent to which discrepancies can exist or be observed, between the copies. If the system offers a strong consistency model, then clients will not be aware of the fact that the system has replicated data, while a weak consistency model requires more careful programming of the clients, so they can cope with the discrepancies they observe.

Historical Background

Most replication research in the 1970s aimed to provide the illusion of an unreplicated database offering serializability. In the early 1980s, Bernstein and colleagues formalized this notion as a strong consistency model [2].

The late 1980s and early 1990s focused on systems that offered weak consistency in various definitions. Eventual consistency was introduced in the work of Demers et al. [3], while consistency models in which reads might see stale values were also explored by several groups [1,5].

Since 2000, a new strong consistency model, one-copy SI, was introduced [4], and it has attracted much attention.

Foundations

There are many different system architectures that can be used for a distributed database with replicated data. For example, clients may submit operations directly to the different databases, or instead requests may all go through a middleware layer; the local databases may communicate directly with one another, or only with the clients or middleware; the requests that arrive at a local database may be a read or write on the local replica of an item, or they may be a SQL statement (which might involve many reads and/or writes), or indeed a whole transaction may form a single request; a read may be performed on one replica and writes on all replicas (read-one-write-all), or else a complex quorum rule may determine where reads and writes are performed; and each site may perform the operations once only, or else there may be possibilities for operations to be done tentatively, then (after conflicting information is received) the system might be able to roll back some operations and then replay them in a different order. Sometimes several system designs offer clients the same functionality, so the choice would be based only on performance, or the validity of assumptions in the design (such as the ability to know in advance which transactions access which items). If the client cannot learn, by the values returned or the operations which are allowed, which system design is used, then one can say that the different designs offer the same consistency model. However, sometimes a difference in design does change the functionality of the whole system, in ways that clients can detect. If one abstracts away the details of the system design, and instead focuses on what the essential features are that distinguish between the properties, then one is describing the consistency model offered by the system. For example, some system designs allow clients to learn about the existence of several copies. Perhaps one client might read the same item several times and see different values in each read. These values may have been taken from different older transactions. This can't happen in a system where all the data is at one site, with one copy for each item. Thus this system provides a consistency model which reveals the existence of copies to the client.

A system provides a strong consistency model if it provides clients with the illusion that there is a single copy of each piece of data, hiding all evidence of the replication. There are in fact several variants among strong models, because there are several different isolation models used by different DBMS platforms, and because the formal definition of isolation doesn't always capture exactly the properties of an implementation. For example, one-copy serializability (q.v.) was defined in [2] as a consistency model in which clients have the illusion of working with a single-site unreplicated database which uses a concurrency control algorithm that offer serializability (q.v.) as the isolation level. In contrast, one-copy SI [4] is a different strong consistency model, where clients see the same behaviors as in an unreplicated system where concurrency control is done by Snapshot Isolation (q.v.).

In contrast to strong consistency models which maintain an illusion of a single-site system, in weaker models the clients are able to see that the system has replicas. Different models are characterized by the ways in which the divergence between replicas is revealed. The best-known weak consistency model is eventual consistency (q.v.) which is suitable for replicated databases where an updating transaction can operate at any replica, and the changes are then propagated lazily to other replicas through an epidemic mechanism. Eventually, each replica learns about the updates, and this consistency model ensures that a reconciliation mechanism resolves conflicting information, so that when the system quiesces, the values in all the replicas of a logical item eventually converge to the same value. In a system providing eventual consistency, there is not much that can be said about the value seen by a read, before convergence has been reached.

A different weak consistency model is common in systems where there is a single master copy for each item, and all updates are done first at the master, before being propagated in order, to the replicas. In this model, writes happen in a well-defined order, and each read sees a value from some prefix of this order; however, a read can see a value that is stale, that is, it does not include the most recent updates to the item.

Key Applications

The commercial DBMS vendors all offer replication mechanisms with their products. The performance impact of strong consistency models is usually seen as high, and these are typically provided only within a single cluster. For replication across dispersed machines, most platforms offer some form of weak consistency. There are also a range of research prototypes which give the user a choice between several consistency models; in general the user sees a tradeoff, where improved performance comes from accepting weaker consistency models.

Future Directions

Effective database replication is not yet a solved problem; the existing proposals compromise somehow among many desired properties, such as scalability for read-heavy workloads, scalability for update-heavy workloads, availability in face of failures or partitions, generality of the clients supported, ease of system programming, capacity to use varied local databases as black boxes, and the consistency provided. Thus the design space of possible systems is still being actively explored, and sometimes a new design achieves a consistency model different from those previously seen. One topic for ongoing research is how users can express their requirements for performance and for different levels of consistency, and how a system can then choose the appropriate replica control mechanism to provide the user with what they need. Research is also likely in consistency models that deal, to some extent, with malicious (often called "Byzantine") sites.

Cross-references

► Data Replication
► Strong Consistency Models for Replicated Data
► Weak Consistency Models for Replicated Data

Recommended Reading

1. Alonso R., Barbará D., and Garcia-Molina H. Data caching issues in an information retrieval system. ACM Trans. Database Syst., 15(3):359–384, 1990.
2. Attar R., Bernstein P.A., and Goodman N. Site initialization, recovery, and backup in a distributed database system. IEEE Trans. Software Eng., 10(6):645–650, 1984.
3. Demers A.J., Greene D.H., Hauser C., Irish W., Larson J., Shenker S., Sturgis H.E., Swinehart D.C., and Terry D.B. Epidemic algorithms for replicated database maintenance. In Proc. ACM SIGACT-SIGOPS 6th Symp. on the Principles of Dist. Comp., 1987, pp. 1–12.
4. Plattner C. and Ganymed G.A. Scalable replication for transactional web applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004, pp. 155–174.
5. Sheth A.P. and Rusinkiewicz M. Management of interdependent data: specifying dependency and consistency requirements. In Proc. Workshop on the Management of Replicated Data, 1990, pp. 133–136.

Consistency Preservation

► ACID Properties

Consistent Facts

▶ Possible Answers

Consistent Query Answering

LEOPOLDO BERTOSSI
Carleton University, Ottawa, ON, Canada

Definition

Consistent query answering (CQA) is the problem of querying a database that is inconsistent, i.e., that fails to satisfy certain integrity constraints, in such a way that the answers returned by the database are consistent with those integrity constraints. This problem involves a characterization of the semantically correct or consistent answers to queries in an inconsistent database.

Key Points

Databases may be inconsistent in the sense that certain desirable integrity constraints (ICs) are not satisfied. However, it may be necessary to still use the database, because it contains useful information, and, most likely, most of the data is still consistent, in some sense. CQA, as introduced in [1], deals with two problems. First, with the logical characterization of the portions of data that are consistent in the inconsistent database. Secondly, with developing computational mechanisms for retrieving the consistent data. In particular, when queries are posed to the database, one would expect to obtain as answers only those answers that are semantically correct, i.e., that are consistent with the ICs that are violated by the database as a whole.

The consistent data in the database is characterized [1] as the data that is invariant under all the database instances that can be obtained after making minimal changes in the original instance with the purpose of restoring consistency. These instances are the so-called (minimal) *repairs*. In consequence, what is consistently true in the database is what is *certain*, i.e., true in the collection of possible worlds formed by the repairs. Depending on the queries and ICs, there are different algorithms for computing consistent answers. Usually, the original query is transformed into a new query, possibly written in a different language, to be posed to the database at hand, in such a way that the usual answers to the latter are the consistent answers to the former [1]. For surveys of CQA and specific references, c.f. [2,3].

Cross-references

▶ Database Repairs
▶ Inconsistent Databases

Recommended Reading

1. Arenas M., Bertossi L., and Chomicki J. Consistent query answers in inconsistent databases. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 68–79.
2. Bertossi L. Consistent query answering in databases. ACM SIGMOD Rec., 35(2):68–76, 2006.
3. Chomicki J. Consistent query answering: five easy pieces. In Proc. 11th Int. Conf. on Database Theory, 2007, pp. 1–17.

Constant Span

▶ Fixed Time Span

Constrained Frequent Itemset Mining

▶ Frequent Itemset Mining with Constraints

Constraint Databases

FLORIS GEERTS
University of Edinburgh, Edinburgh, UK

Definition

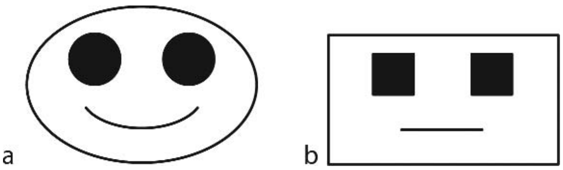
Constraint databases are a generalization of relational databases aimed to store possibly infinite-sized sets of data by means of a finite representation (*constraints*) of that data. In general, constraints are expressed by *quantifier-free first-order formulas* over some fixed vocabulary Ω and are interpreted in some Ω-structure $\mathcal{M} = \langle \mathbb{U}, \Omega \rangle$. By varying Ω and $\mathcal{M}$, constraint databases can model a variety of data models found in practice including traditional relational databases, spatial and spatio-temporal databases, and databases with text fields (strings). More formally, let Ω be a fixed vocabulary consisting of function, predicate and constant symbols, and let $\mathcal{R} = \{R_1,\ldots,R_\ell.\}$ be a relational schema, where each relation name R_i is of arity

$n_i > 0$. An Ω*-constraint database* $\mathbf{D}$ with schema $\mathcal{R}$ maps each relation $R_i \in \mathcal{R}$ to a quantifier-free formula $\varphi^{\mathbf{D}}_{R_i}(x_1,\ldots,x_{n_i})$ (with n_i free variables $x_1,\ldots,x_{n_i}$) in *first-order logic* over Ω. When interpreted over an Ω-structure $\mathcal{M} = \langle\mathbb{U},\Omega\rangle$, an Ω-constraint database $\mathbf{D}$ with schema $\mathcal{R}$ corresponds to the collection of the $\mathcal{M}$*-definable sets* $[\![R_i]\!]^{\mathbf{D}}_{\mathcal{M}} = \{(a_1,\ldots,a_{n_i}) \in \mathbb{U}^{n_i} \mid \mathcal{M} \models \varphi^{\mathbf{D}}_{R_i}(a_1,\ldots,a_{n_i})\}$, for $R_i \in \mathcal{R}$. Constraint query languages have been devised to manipulate and query constraint databases.

Key Points

The primary motivation for constraint databases comes from the field of spatial and spatio-temporal databases where one wants to store an infinite set of points in the real Euclidean space and query it as if all (infinitely) many points are present [3,4,5]. In the spatial context, the constraints used to finitely represent data are *Boolean combinations of polynomial inequalities.* For instance, the infinite set of points in the real plane $\mathbb{R}^2$ depicted in Fig. 1(a) can be described by means of a disjunction of polynomial inequalities with integer coefficients as follows: $\varphi(x, y) = (x^2/25 + y^2/16 = 1) \vee (x^2 + 4x + y^2 - 2y \leq 4) \vee (x^2 - 4x + y^2 - 2y \leq -4) \vee (x^2 + y^2 - 2y = 8 \wedge y < -1)$. In the language of constraint databases, $\varphi(x, y)$ is a quantifier-free first-order formula over $\Omega = (+,\cdot,0,1,<)$ and Fig. 1(a) represents the $\mathcal{M}$-definable set in $\mathbb{R}^2$ corresponding to the formula φ for the Ω-structure $\mathcal{M} = \langle\mathbb{R},\Omega\rangle$. If $\mathcal{R}$ is a relational schema consisting of a binary relation R, then the Ω-constraint database $\mathbf{D}$ with schema $\mathcal{R}$ defined by $R \mapsto \varphi(x, y)$ "stores" the set in Fig. 1(a). In this case, the $\mathcal{M}$-definable sets are also known as semi-algebraic sets [2].

When *Boolean combination of linear inequalities* suffice, such as in geographical information systems (GIS), one considers constraint databases over $\Omega = (+,0,1,<)$ and $\mathcal{M} = \langle\mathbb{R},\Omega\rangle$. Fig. 1(b) shows an example of a set defined by means of a first-order formula over $\Omega = (+,0,1,<)$. The advantage of the constraint approach to represent spatial data is the uniform representation of the various spatial entities. Whereas in GIS one normally defines a special data-type for each spatial object such as line, poly-line, circle,..., each of those are now represented by constraints in the same constraint language.



Constraint Databases. Figure 1. Example of set definable by (a) polynomial constraints and (b) linear constraints.

Other common scenarios of constraint databases include: *dense order constraints over the rationals*, where $\Omega = (<,(c)_{c\in\mathbb{Q}})$ and $\mathcal{M} = \langle\mathbb{Q},\Omega\rangle$. That is, rational numbers with order and constants for every $c \in \mathbb{Q}$; and *constraints over strings*, where $\Omega = ((f_a)_{a\in\Sigma},\prec,\text{el})$ and $\mathcal{M} = \langle\Sigma^*,\Omega\rangle$ [1]. Here, Σ is a finite alphabet, f_a is a function that adds a at the end of its argument, $\prec$ is the prefix relation and $\text{el}(x, y)$ is a binary predicate that holds if $|x| = |y|$, where $|\cdot|$ stands for the length of a finite string. In the latter case, the $\mathcal{M}$-definable sets are precisely the regular languages over Σ.

Finally, standard relational databases with schema $\mathcal{R}$ can be considered as constraint databases over *equality constraints over an arbitrary infinite domain* $\mathbb{U}$, where $\Omega = ((c)_{c\in\mathbb{U}})$ and $\mathcal{M} = \langle\mathbb{U},\Omega\rangle$. Indeed, consider a tuple $t = (a_1,\ldots,a_n)$ consisting of some constants $a_i \in \mathbb{U}$, for $i \in [1,n]$. The tuple t can be expressed by the formula $\varphi_t(x_1,\ldots,x_n) = (x_1 = a_1) \wedge\ldots\wedge (x_n = a_n)$ over the signature $\Omega = ((c)_{c\in\mathbb{U}})$. More generally, an instance $I = \{t_1,\ldots,t_N\}$ over $R \in \mathcal{R}$ corresponds to $\varphi_I = \bigvee_{i=1}^{N} \varphi_{t_i}$. Therefore, a relational instance $(I_1,\ldots,I_\ell)$ over $\mathcal{R}$ can be represented as the constraint database $\mathbf{D}$ defined by $R_i \mapsto \varphi_{I_i}(x_1,\ldots,x_{n_i})$, for $i \in [1,\ell]$. This shows that constraint databases indeed generalize standard relational databases.

Cross-references

- ▶ Constraint query languages
- ▶ Geographic information system
- ▶ Relational model
- ▶ Spatial Data Types

Recommended Reading

1. Benedikt M., Libkin L., Schwentick T., and Segoufin L. Definable relations and first-order query languages over strings. J. ACM, 50(5):694–751, 2003.
2. Bochnak J., Coste M., and Roy M.F. Real Algebraic Geometry. Springer, Berlin, 1998.
3. Kanellakis P.C., Kuper G.M., and Revesz P.Z. Constraint query languages. J. Comput. Syst. Sci., 51(1):26–52, 1995.
4. Kuper G.M., Libkin L., and Paredaens J. (eds.) Constraint databases. Springer, Berlin, 2000.
5. Revesz P.Z. Introduction to Constraint Databases. Springer, Berlin, 2002.

Constraint Query Languages

FLORIS GEERTS
University of Edinburgh, Edinburgh, UK

Definition

A constraint query language is a query language for constraint databases.

Historical Background

The field of constraint databases was initiated in 1990 in a paper by Kanellakis, Kuper and Revesz [9]. The goal was to obtain a database-style, optimizable version of constraint logic programming. It grew out of the research on DATALOG and constraint logic programming. The key idea was that the notion of tuple in a relational database could be replaced by a conjunction of constraints from an appropriate language, and that many of the features of the relational model could then be extended in an appropriate way. In particular, standard query languages such as those based on first-order logic and DATALOG could be extended to such a model.

It soon became clear, however, that recursive constraint query languages led to non-effective languages. The focus therefore shifted to non-recursive constraint query languages. The standard query language is the constraint relational calculus (or equivalently, the constraint relational algebra). The study of this query language turned out to lead to many interesting research problems. During the period from 1990 to 2000, the constraint setting has been studied in great generality which led to deep connections between constraint databases and embedded finite model theory. Also, the potential application of constraint databases in the spatial context led to numerous theoretical results and concrete implementations such as the DEDALE and the DISCO systems. The connection with so-called o-minimal geometry underlies many of the results in the spatial setting. The success of this research led to the publication of a comprehensive survey of the area in 2000 [11] and a textbook in 2002 [13].

In recent years, constraint query languages have been studied in new application domains such a strings, spatio-temporal and moving objects.

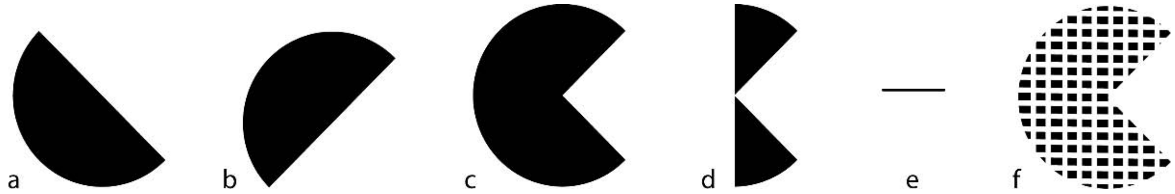
Foundations

In the constraint model, a database is viewed as a collection of constraints specified by quantifier-free first-order logic formulas over some fixed vocabulary $\mathbf{\Omega}$. When interpreted over an $\mathbf{\Omega}$-structure $\mathcal{M} = \langle \mathbb{U}, \mathbf{\Omega} \rangle$, each constraint corresponds to an $\mathcal{M}$-definable set. Consequently, when interpreted over $\mathcal{M}$, an $\mathbf{\Omega}$-constraint database corresponds to a collection of $\mathcal{M}$-definable sets. For instance, consider the vocabulary $\mathbf{\Omega} = (+, \cdot, 0, 1, <)$ and $\mathcal{M} = \langle \mathbb{R}, \mathbf{\Omega} \rangle$. Constraints in first-order logic over $\mathbf{\Omega}$, denoted by FO($\mathbf{\Omega}$), correspond to Boolean combinations of polynomial inequalities with integer coefficients. The corresponding $\mathcal{M}$-definable sets are better known as semi-algebraic sets. Let $\mathcal{R} = \{R, S\}$ be a relational schema consisting of two binary relations R and S and let $\mathbf{D}$ be the constraint database that maps $R \mapsto \varphi_R(x, y) = (x^2 + y^2 \leq 1) \wedge (y - x \geq 0)$ and $S \mapsto \varphi_S(x, y) = (x^2 + y^2 \leq 1) \wedge (-y - x \geq 0)$. The two $\mathcal{M}$-definable sets in $\mathbb{R}^2$ corresponding to φ_R and φ_S are shown in Figs. 1(a) and (b) respectively.

A constraint database can therefore be viewed from two different perspectives: First, one can simply look at the finite representations (constraints) stored in them; Second, one can regard them as a set of definable sets. Whereas in traditional relational databases, a query is simply a mapping that associates with each database an answer relation, in the constraint setting the two different perspectives give rise to two different notions of queries.

Indeed, for a fixed vocabulary $\mathbf{\Omega}$, relational schema $\mathcal{R}$ consisting of relation names $R_1,\ldots,R_\ell$, where each relation R_i is of arity $n_i > 0$, and natural number k, a *k-ary constraint query* with schema $\mathcal{R}$ over $\mathbf{\Omega}$, is a (partial) function Q that maps each $\mathbf{\Omega}$-constraint databases $\mathbf{D}$ with schema $\mathcal{R}$ to a k-ary $\mathbf{\Omega}$-constraint relation $Q(\mathbf{D})$. That is, a constraint query works entirely on the representational (constraint) level. On the other hand, given an additional $\mathbf{\Omega}$-structure $\mathcal{M} = \langle \mathbb{U}, \mathbf{\Omega} \rangle$, a *k-ary unrestricted query* with schema $\mathcal{R}$ over $\mathcal{M}$ is a (partial) function Q that maps each collection $\mathbf{D}$ of sets in $\mathbb{U}^{n_i}$, for $i \in [1, \ell]$, to a set $Q(\mathbf{D})$ in $\mathbb{U}^k$. Such a collection of sets $\mathbb{U}^{n_i}$, for $i \in [1, \ell]$, is called an *unrestricted database* with schema $\mathcal{R}$ over $\mathcal{M}$.

For instance, consider again $\mathbf{\Omega} = (+, \cdot, 0, 1, <)$ and $\mathcal{R} = \{R, S\}$. The mapping Q_1 that associates each $\mathbf{\Omega}$-constraint database $\mathbf{D}$ over $\mathcal{R}$ with the binary $\mathbf{\Omega}$-constraint relation defined by taking the disjunction of the constraints in R and S, is an example of a 2-ary constraint query over $\mathbf{\Omega}$. When applied on the database $\mathbf{D}$ given above, $Q_1(\mathbf{D})$ is mapped to $\varphi_R(x, y) \vee \varphi_S(x, y)$. Similarly, the mapping Q_2 that maps $\mathbf{D}$ to the



Constraint Query Languages. Figure 1. Sets in $\mathbb{R}^2$ defined by $\varphi_R(x, y)$ (a); by $\varphi_S(x, y)$ (b); by $\varphi_R(x, y) \vee \varphi_S(x, y)$ (c); and by $\varphi_1(x, y)$ (d). The set in $\mathbb{R}$ defined by $\varphi_2(x)$ (e). An example of a non-definable set in $\mathbb{R}^2$ (f).

constraint in R or S that contains the polynomial with the largest coefficient (if there is no such unique constraint then Q_2 is undefined) is also a constraint query. It will be undefined on the example database $\mathbf{D}$ since both R and S consist of a polynomial with coefficient one.

So far, only constraint queries have been considered. To relate constraint and unrestricted queries requires some care. Clearly, a constraint query only makes sense if it corresponds to an unrestricted query. In this case, a constraint query is called *consistent*. More formally, a constraint query Q is called consistent if there exists an unrestricted query Q_0 such that for any constraint database $\mathbf{D}$ and any unrestricted database $\mathbf{D}_0$, if $\mathbf{D}$ represents $\mathbf{D}_0$, then $Q(\mathbf{D})$ is defined if and only if $Q_0(\mathbf{D}_0)$ is defined and furthermore, $Q(\mathbf{D})$ represent $Q_0(\mathbf{D}_0)$. One also says that Q *represents* Q_0.

For instance, consider again $\Omega = (+, \cdot, 0, 1, <)$, $\mathcal{M} = \langle \mathbb{R}, \Omega \rangle$ and $\mathcal{R} = \{R, S\}$. The mapping $\tilde{Q}_1$ that assigns to any two sets $A \subseteq \mathbb{R}^2$ and $B \subseteq \mathbb{R}^2$, corresponding to R and S, respectively, their union $A \cup B \subseteq \mathbb{R}^2$ is an unrestricted query. It is clear that Q_1 and $\tilde{Q}_1$ satisfy the condition of consistency and therefore Q_1 is consistent. Fig. 1(c) shows $\tilde{Q}_1(\mathbf{D}_0)$ for the unrestricted database $\mathbf{D}_0$ shown in Fig. 1(a) and (b). This set is indeed represented by the constraint relation $Q_1(\mathbf{D}) \mapsto \varphi_R(x, y) \vee \varphi_S(x, y)$. On the other hand, it is easily verified that Q_2 is not consistent. Indeed, it suffices to consider the behavior of Q_2 on $\mathbf{D}$ defined above and $\mathbf{D}'$ defined by $R \mapsto \varphi'_R(x, y) = (x^2 + y^2 \leq 1) \wedge (6(y - x) \geq 0)$ and $S \mapsto \varphi'_S(x, y) = (x^2 + y^2 \leq 1) \wedge (-y - x \geq 0)$. While both $\mathbf{D}$ and $\mathbf{D}'$ represent the same unrestricted database, note that $Q_2(\mathbf{D})$ is undefined while $Q_2(\mathbf{D}') \mapsto \varphi_R$. Hence, no unrestricted query that is consistent with Q_2 can exist.

Finally, unrestricted queries are defined without any reference to the class of $\mathcal{M}$-definable sets. A desirable property, however, is that when an unrestricted query Q is defined on an unrestricted database $\mathbf{D}$ that consists of $\mathcal{M}$-definable sets, then also $Q(\mathbf{D})$ is an $\mathcal{M}$-definable set. Such unrestricted queries are called *closed*. Note that an unrestricted query that is represented by a consistent constraint query is uniquely defined and moreover is trivially closed. An example of an unrestricted query for $\Omega = (+, \cdot, 0, 1, <)$ and $\mathcal{M} = \langle \mathbb{R}, \Omega \rangle$ that is not closed is the query Q that maps any $\mathcal{M}$-definable set A in $\mathbb{R}^2$ to its intersection $A \cap \mathbb{Q}^2$. Fig. 1(f) shows (approximately) the result of this query on $\tilde{Q}_1(\mathbf{D}_0)$ (i.e., Fig. 1(c)). Since this is not a semi-algebraic set in $\mathbb{R}^2$, it cannot be defined by means of a quantifier-free FO(Ω)-formula. As a consequence, Q is not closed.

Now that the notion of query is defined in the setting of constraint databases, the basic *constraint query language* is introduced. This language, in the same spirit as the relational calculus for traditional relational databases, is the *relational calculus* or first-order logic of the given class of constraints. More specifically, given a vocabulary Ω and relational schema $\mathcal{R}$, a *relational calculus formula* over Ω is a first-order logic formula over the expanded vocabulary $(\Omega, \mathcal{R})$ obtained by expanding Ω with the relation names (viewed as predicate symbols) of the schema $\mathcal{R}$. This class of queries is denoted by FO($\Omega, \mathcal{R}$), or simply FO(Ω) when $\mathcal{R}$ is understood from the context.

For instance, for $\Omega = (+, \cdot, 0, 1, <)$ and $\mathcal{R} = \{R, S\}$, the expressions $\varphi_1(x, y) = (R(x, y) \vee S(x, y)) \wedge x > 0$ and $\varphi_2(x) = \exists y \varphi_1(x, y)$ are formulas in FO($+, \cdot, 0, 1, <, R, S$).

Given an Ω-structure $\mathcal{M} = \langle \mathbb{U}, \Omega \rangle$, formulas in FO($\Omega, \mathcal{R}$) express (everywhere defined) unrestricted queries with schema $\mathcal{R}$ over $\mathcal{M}$. Indeed, a formula $\varphi(x_1, \ldots, x_k) \in$ FO($\Omega, \mathcal{R}$) defines the k-ary unrestricted query Q over $\mathcal{M}$ as follows: consider the expansion of $\mathcal{M}$ to a structure $\langle \mathcal{M}, \mathbf{D} \rangle = \langle \mathbb{U}, \Omega, \mathbf{D} \rangle$ over the expanded vocabulary $(\Omega, \mathcal{R})$ by adding the sets in the unrestricted database $\mathbf{D}$ to $\mathcal{M}$ for each $R_i \in \mathcal{R}$. Then, $Q(\mathbf{D}) = \{(a_1, \ldots, a_k) \in \mathbb{U}^k \mid 0\langle \mathcal{M}, \mathbf{D} \rangle \models \varphi(a_1, \ldots, a_k)\}$.

For instance, for $\Omega = (+, \cdot, 0, 1, <)$ and $\mathcal{M} = \langle \mathbb{R}, \Omega \rangle$, the formula $\varphi_1(x, y)$ defined above corresponds to the unrestricted query Q_1 that takes the union of the two sets in $\mathbb{R}^2$ corresponding to R and S, respectively, restricted to those points in $\mathbb{R}^2$ with strictly positive x-coordinate. Similarly for φ_2, but with an additional projection on the x-axis. The results of these two unrestricted queries have been shown in Figs. 1(d), (e), respectively.

The previous example raises the following two questions: (i) are the unrestricted queries expressed by formulas in first-order logic closed, and (ii) if so, can one find a corresponding constraint query that is effectively computable? The fundamental mechanism underlying the use of first-order logic as a constraint query language is the following observation that provides an answer to both questions:

- Every relational calculus formula φ expresses a consistent, effectively computable, total constraint query that represents the unrestricted query expressed by φ, if and only if $\mathcal{M}$ admits *effective quantifier elimination*.

Here, an Ω-structure $\mathcal{M}$ admits effective quantifier elimination if there exists an effective algorithm that transforms any first-order formula in FO(Ω) to an equivalent (in the structure $\mathcal{M}$) *quantifier-free* first-order formula in FO(Ω).

Consider the two FO($+,\cdot,0,1,<, R, S$)-formulas φ_1 and φ_2 given above. It is known that the structure $\langle\mathbb{R}, +, \cdot, 0, 1, <\rangle$ admits effective quantifier-elimination. In case of φ_1 it is easy to see that the result of corresponding constraint query is obtained by "plugging" in the constraints for R (resp. S) as given by the constraint database into the expression for φ_1. That is, on the example database **D**, φ_1 corresponds to the constraint query that maps **D** to $(\varphi_R(x, y) \vee \varphi_S(x, y)) \wedge (x > 0)$, which is a 2-ary Ω-constraint relation. In case of φ_2, however, first plug in the descriptions of the constraints as before, resulting in $\exists y\, (\varphi_R(x, y) \vee \varphi_S(x, y)) \wedge (x > 0)$. In order to obtain an Ω-constraint relation, one needs perform quantifier-elimination. It is easily verified that in this example, a corresponding constraint query is one that maps **D** to $(0 < x) \wedge (x \leq 1)$ which is consistent with Fig. 1(e).

For Ω-structures $\mathcal{M}$ that admit effective quantifier-elimination, this suggests the following effective evaluation mechanism for constraint relational calculus queries φ on a constraint database **D**: (i) plug in the contents of **D** in the appropriate slots (relations). Denote the resulting formula by plug(φ, **D**); and (ii) eliminate the quantifiers in plug(φ, **D**). Since **D** consists of quantifier-free formulas, the number of quantifiers that need to be eliminated is the same as in φ and is therefore independent of **D**. For many structures $\mathcal{M}$ this implies that the evaluation of constraint queries can be done in polynomial data complexity, which is a desirable property for any query language.

It is important to point out that the classical equivalence between the relational calculus and the relational algebra can be easily extended to the constraint setting. That is, for a fixed Ω and schema $\mathcal{R}$, one can define a *constraint relational algebra* and show that every constraint relational calculus formula can be effectively converted to an equivalent constraint relational algebra expression, and vice versa. This equivalence is useful for concrete implementations of constraint database systems.

The study of expressivity of FO($\Omega, \mathcal{R}$) for various Ω-structures $\mathcal{M}$ has led to many interesting results. In particular, the impact of the presence of the "extra" structure on the domain elements in $\mathbb{U}$ has been addressed when **D** consists of an ordinary finite relational database that takes values from $\mathbb{U}$ [3]. In particular, the correspondence between natural and active-domain semantics has been revisited. That is, conditions are identified for $\mathcal{M} = \langle\mathbb{U}, \Omega\rangle$ such that the language FO($\Omega, \mathcal{R}$) is equal to $\text{FO}_{\text{act}}(\Omega, \mathcal{R})$, the query language obtained by interpreting $\forall x$ and $\exists x$ over the active domain of **D** instead of over $\mathbb{U}$. Such structures are said to admit the *natural-active collapse*. Similarly, ordered structures $\mathcal{M}$ are identified that admit the *active-generic collapse*. That is, $\text{FO}_{\text{act}}(\Omega, \mathcal{R})$ is equal to $\text{FO}_{\text{act}}(<, \mathcal{R})$ with respect to the class of generic queries. In other words, every generic query definable under active domain semantics with Ω-constraints is already definable with just order constraints. Finally, structures $\mathcal{M}$ are considered that allow the *natural-generic collapse*. This is the same as the active-generic collapse but with natural domain semantics instead of active domain semantics. The study of these collapse properties for various structures not only sheds light on the interaction of the structure on $\mathbb{U}$ and the query language, it is also helpful to understand the expressiveness of constraint query languages [3,11].

Indeed, let $\Omega = (+, \times, 0, 1, <)$ and $\mathcal{M} = \langle\mathbb{R}, \Omega\rangle$. It can be shown that $\mathcal{M}$ admits all three collapses because it is a so-called o-minimal structure. As a consequence, the query EVEN that returns **yes** if the cardinality of **D** is even and **no** otherwise, is not expressible in $\mathrm{FO}(\Omega, \mathcal{R})$. Indeed, if it would be expressible by a query φ in $\mathrm{FO}(\Omega, \mathcal{R})$ it would already have been expressible by a query in $\mathrm{FO}_{\mathrm{act}}(<, \mathcal{R})$, which is known not to be true in the traditional database setting.

The expressivity of $\mathrm{FO}(\Omega, \mathcal{R})$ has been studied extensively as well when **D** corresponds to sets of infinite size. In particular, expressiveness questions have been addressed in the spatial setting where $\Omega = (+, \cdot, 0, 1, <)$ and $\mathcal{M} = \langle \mathbb{R}, \Omega \rangle$ (polynomial constraints); and $\Omega' = (+, 0, 1, <)$ and $\mathcal{M} = \langle \mathbb{R}, \Omega' \rangle$ (linear constraints). In this setting, many reductions are presented in [7] to expressiveness questions in the finite case. Combined with the collapse results mentioned above, these reductions were used to show that, for example, topological connectivity of Ω- (resp. Ω'-) constraint databases is not expressible in first-order logic. Indeed, a proof of this results relies on the fact that the EVEN-query is not expressible in $\mathrm{FO}(\Omega, \mathcal{R})$ (resp. $\mathrm{FO}(\Omega', \mathcal{R})$) [7].

An interesting line of work in the spatial context concerns the expressive power of $\mathrm{FO}(\Omega, \mathcal{R})$ with respect to queries that preserve certain geometrical properties. More formally, let $\mathcal{G}$ be a group of transformations of $\mathbb{R}^k$. A query Q is called $\mathcal{G}$-generic if, for every transformation $g \in \mathcal{G}$, and for any two databases **D** and **D**′, $g(\mathbf{D}) = \mathbf{D}'$ implies $g(Q(\mathbf{D})) = Q(\mathbf{D}')$. Transformation groups and properties of the corresponding generic queries have been studied for the group of homeomorphisms, affinities, similarities, isometries, among others [8]. Especially the study of the topologically queries (those that are generic under homeomorphisms) has received considerable attention [10,2].

To conclude, both for the historical reasons mentioned above and in view of the limited expressive power of $\mathrm{FO}(\Omega, \mathcal{R})$, various recursive extensions of $\mathrm{FO}(\Omega, \mathcal{R})$ have been proposed such as: constraint transitive-closure logic [5], constraint DATALOG [9], and $\mathrm{FO}(\Omega, \mathcal{R})$ extended with a WHILE-loop [8]. The interaction of recursion with the structure on $\mathbb{U}$ imposed by Ω leads in most cases to computationally complete query languages. Worse still, queries defined in these languages may not be closed or even terminate. To remedy this, special-purpose extensions of $\mathrm{FO}(\Omega, \mathcal{R})$ have been proposed that guarantee both termination and closure. Characteristic examples include $\mathrm{FO}(\Omega, \mathcal{R}) + \mathrm{AVG}$ and $\mathrm{FO}(\Omega, \mathcal{R}) + \mathrm{SUM}$ in the context of aggregation [4]. In the spatial setting, extensions of $\mathrm{FO}(\Omega, \mathcal{R})$ with various connectivity operators have been proposed [1].

Results concerning constraint query languages have been both extended to great generality and applied to concrete settings. Refer to [14] for a gentle introduction and to [11] for a more detailed survey of this research area up to 2000. Some more recent results are included in Chapter 5 of [12] for the general constraint setting and Chapter 12 in [6] for the spatial setting.

Key Applications

Manipulation and querying of constraint databases, querying of spatial data.

Cross-references

- ▶ Computationally Complete Relational Query Languages
- ▶ Constraint Databases
- ▶ FOL
- ▶ FOL Modeling of Integrity Constraints (Dependencies)
- ▶ Query Language
- ▶ Relational Calculus
- ▶ Relational Model

Recommended Reading

1. Benedikt M., Grohe M., Libkin L., and Segoufin L. Reachability and connectivity queries in constraint databases. J. Comput. Syst. Sci., 66(1):169–206, 2003.
2. Benedikt M., Kuijpers B., Christ of Löding, Van den Bussche J., and Wilke T. A characterization of first-order topological properties of planar spatial data. J. ACM, 53(2):273–305, 2006.
3. Benedikt M. and Libkin L. Relational queries over interpreted structures. J. ACM, 47(4):644–680, 2000.
4. Benedikt M. and Libkin L. Aggregate operators in constraint query languages. J. Comput. Syst. Sci., 64(3):628–654, 2002.
5. Geerts F., Kuijpers B., and Van den Bussche J. Linearization and completeness results for terminating transitive closure queries on spatial databases. SIAM J. Comput., 35(6):1386–1439, 2006.
6. Geerts F. and Kuijpers B. Real algebraic geometry and constraint databases. In M. Aiello, I. Pratt-Hartmann, and J. Van Benthem, editors, Handbook of Spatial Logics. Springer 2007.
7. Grumbach S. and Su J. Queries with arithmetical constraints. Theor. Comput. Sci., 173(1):151–181, 1997.
8. Gyssens M., Van den Bussche J., and Van Gucht D. Complete geometric query languages. J. Comput. Syst. Sci., 58(3):483–511, 1999.
9. Kanellakis P.C., Kuper G.M., and Revesz P.Z. Constraint Query Languages. J. Comput. Syst. Sci., 51(1):26–52, 1995.
10. Kuijpers B., Paredaens J., and Van den Bussche J. Topological elementary equivalence of closed semi-algebraic sets in the real plane. J. Symb. Log., 65(4):1530–1555, 2000.
11. Kuper G.M., Libkin L., and Paredaens J. Constraint Databases. editors. Springer, 2000.

12. Libkin L. Embedded finite models and constraint databases. In Grädel E., Kolaitis P.G., Libkin L., Marx M., Spencer J., Vardi M.Y., Venema Y. and Weinstein S., editors, Finite Model Theory and Its Applications. Springer, 2007.
13. Revesz P.Z. Introduction to Constraint Databases. Springer, 2002.
14. Van den Bussche J. Constraint databases. A tutorial introduction. ACM SIGMOD Record, 29(3):44–51, 2000.

Constraint-Driven Database Repair

Wenfei Fan[1,2]
[1]University of Edinburgh, Edinburgh, UK
[2]Bell Laboratories, Murray Hill, NJ, USA

Synonyms

Data reconciliation; Minimal-change integrity maintenance; Data standardization

Definition

Given a set Σ of integrity constraints and a database instance D of a schema R, the problem of *constraint-driven database repair* is to find an instance D' of the same schema R such that (i) D' is *consistent*, i.e., D' satisfies Σ, and moreover, (ii) D' *minimally differs* from the original database D, i.e., it takes a minimal number of repair operations or incurs minimal cost to obtain D' by updating D.

Historical Background

Real life data is often dirty, i.e., inconsistent, inaccurate, stale or deliberately falsified. While the prevalent use of the Web has made it possible, on an unprecedented scale, to extract and integrate data from diverse sources, it has also increased the risks of creating and propagating dirty data. Dirty data routinely leads to misleading or biased analytical results and decisions, and incurs loss of revenue, credibility and customers. With this comes the need for finding repairs of dirty data, and editing the data to make it consistent. This is the data cleaning approach that US national statistical agencies, among others, has been practicing for decades [10].

The notion of constraint-based database repairs is introduced in [1], highlighting the use of integrity constraints for characterizing the consistency of the data. In other words, constraints are used as data quality rules, which detect inconsistencies as violations of the constraints. Prior work on constraint-based database repairs has mostly focused on the following issues. (i) Integrity constraints used for repair. Earlier work considers traditional functional dependencies, inclusion dependencies and denial constraints [1,2,4,6,12]. Extensions of functional and inclusion dependencies, referred to as conditional functional and inclusion dependencies, are recently proposed in [3,9] for data cleaning. (ii) Repair semantics. Tuple deletion is the only repair operation used in [6], for databases in which the information is complete but not necessarily consistent. Tuple deletion and insertion are considered in [1,4] for databases in which the information may be neither consistent nor complete. Updates, i.e., attribute-value modification are proposed as repair operations in [12]. Cost models for repairs are studied in [2,8]. (iii) Algorithms. The first algorithms for finding repairs are developed in [2], based on traditional functional and inclusion dependencies. Algorithms for repairing and incrementally repairing databases are studied in [8], using conditional functional dependencies. The repair model adopted by these algorithms supports updates as repair operations. (iv) Fundamental issues associated with constraint-based repairs. One issue concerns the complexity bounds on the database repair problem [2,6]. Another issue concerns the static analysis of constraint consistency [3,9] for determining whether a given set of integrity constraints is dirty or not itself.

Constraint-based database repairs are one of the two topics studied for constraint-based data cleaning. The other topic, also introduced in [1], is *consistent query answers*. Given a query Q posed on an inconsistent database D, it is to find tuples that are in the answer of Q over every repair of D. There has been a host of work on consistent query answers [1,4,6,11,12] (see [5,7] for comprehensive surveys).

Foundations

The complexity of the constraint-based database repair problem is highly dependent upon what integrity constraints and repair model are considered.

Integrity Constraints for Characterizing Data Consistency

A central technical issue for data cleaning concerns how to tell whether the data is dirty or clean. Constraint-based database repair characterizes inconsistencies in terms of violations of integrity constraints. Constraints employed for data cleaning include functional dependencies, inclusion dependencies, denial constraints, conditional functional dependencies and conditional inclusion dependencies. To illustrate these constraints, consider the following relational schema R, which consists of three relation schemas:

customer(name, country-code, area-code, phone, city, street, zip)
order(name, country-code, area-code, phone, item-id, title, price, item-type)
book(isbn, title, price, format)

Traditional functional dependencies and inclusion dependencies defined on the schema R include:

FD: customer(country-code, area-code, phone → city, street, zip)
IND: order(name, country-code, area-code, phone) $\subseteq$ customer(name, country-code, area-code, phone)

The functional dependency FD asserts that the phone number (country-code, area-code and phone) of a customer uniquely determines her address (state, city, street, zip). That is, for any two customer tuples, if they have the same country-code, area-code and phone number, then they must have the same state, city, street, zip code. The inclusion dependency IND asserts that for any order tuple t, there must exist a customer tuple t' such that t and t' match on their name, country-code, area-code and phone attributes. In other words, an item cannot be ordered by a customer who does not exist.

Consider a set Σ consisting of FD and IND . One may want to use Σ to specify the consistency of database instances of R. An example instance D of R is shown in Fig. 1. This database is inconsistent, because tuples t_3 and t_4 in D violate the functional dependency FD . Indeed, while t_3 and t_4 have the same country-code, area-code and phone number, they differ in their street attributes. In other words, t_3, t_4 or both of them may be dirty.

One may also want to add a denial constraint to the set Σ:

DC : $\forall$nm, cc, ac, ph, id, tl, tp, pr$\neg$ (order(nm, cc, ac, ph, id, tl, pr, tp) $\wedge$pr $>$ 100)

Here nm, cc, ac, ph, id, tl, tp, pr stand for name, country-code, area-code, phone number, item-id, title, item type and price, respectively. This constraint says that no items in the order table may have a price higher than 100. In the database D of Fig. 1, tuple t_6 violates the constraint DD : the price of the CD is too high to be true. In general denial constraints can be expressed as universally quantified first-order logic sentences of the form:

$$\forall \bar{x}_1,\ldots,\bar{x}_m \neg(R_1(\bar{x}_1) \wedge \ldots \wedge R_m(\bar{x}_m) \wedge \varphi(\bar{x}_1,\ldots,\bar{x}_m)),$$

where R_i is a relation symbol for $i \in [1,m]$, and φ is a conjunction of built-in predicates.

Now consider an instance D' of D by removing t_3 from D and changing t_6[pr] to, e.g., 7.99. Then the database D' satisfies Σ. However, D' is not quite clean: it violates each of the following constraints. In other

	Name	Country-code	Area-code	Phone	City	Street	Zip
t_1:	M. Smith	44	131	1233444	NYC	Mayfield	EH8 8LE
t_2:	L. Webber	44	131	2344455	NYC	Crichton	EH8 8LE
t_3:	M. Hull	01	908	3456788	NYC	Mountain Ave	07974
t_4:	R. Xiong	01	908	3456788	NYC	Main St	07974

a Example customer data

	Name	Country-code	Area-code	Phone	Item-id	Title	Price	Item-type
t_5:	M.Smith	44	131	1233444	b23	H. Porter	17.99	book
t_6:	M.Hull	01	908	3456788	c68	John Denver	2000	CD

b Example order data

	isbn	Title	Price	Format
t_7:	b23	Harry porter	17.99	hard-cover
t_8:	b88	Snow white	7.94	audio

c Example book data

Constraint-Driven Database Repair. Figure 1. Example database instance.

words, if one further extends Σ by including the following constraints, then D' no longer satisfies Σ.

CFD1 : customer(country-code = 44, zip → street)
CFD2 : customer(country-code = 44, area-code = 131, phone → city = EDI, street, zip)
CFD3 : customer(country-code = 01, area-code = 908, phone → city = MH, street, zip)
CIND1 : order(id, title, price; item-type = book) ⊆ book(isbn, title, price)

Here CFD1, CFD2 and CFD3 are conditional functional dependencies defined on the customer relation. The constraint CFD1 asserts that for each customer in the UK, i.e., when the country code is 44, her zip code uniquely determines her street. In contrast to traditional functional dependencies, CFD1 does not hold on the entire customer relation. Indeed, it does not hold on customer tuples with, e.g., country-code = 01. Instead, it is applicable only to the set of customer tuples with country-code = 44. Constraints CFD2 and CFD3 refine the traditional functional dependency FD given earlier: CFD2 requires that when the country-code is 44 and area-code is 131, the city must be Edinburgh (EDI); similarly for CFD3. None of these can be expressed as traditional functional dependencies.

The constraint CIND1 is a conditional inclusion dependency, asserting that when the type of an item t in the order table is book, there must exist a corresponding tuple t' in the book table such that t and t' match on their id, title and price. Again this is a constraint that only holds conditionally. Indeed, without the condition item-type = book, a traditional inclusion dependency order(id, title, price) ⊆ book (isbn, title, price) does not make sense since, among other things, it is unreasonable to require each CD item in the order table to match a tuple in the book table.

These conditional dependencies tell us that the database D' is not clean after all. Indeed, tuples t_1,t_2 in the customer table violate CFD1: while they both represent customers in the UK and have the same zip code, they differ in their street attributes. Furthermore, each of t_1 and t_2 violates CFD2: while its area-code is 131, its city is NYC instead of EDI. Similarly, t_4 violates CFD3. From these one can see that while it takes two tuples to violate a traditional functional dependency, a single tuple may violate a conditional functional dependency. The inconsistencies in D' are not limited to the customer table: while tuple t_5 in the order table has item-type = book, there exists no tuple t' in the book table such that t_5 and t' match on their id, title and price attributes. Thus either t_5 in the order table is not error-free, or the book table is incomplete or inconsistent.

Conditional functional and inclusion dependencies are extensions of traditional functional and inclusion dependencies, respectively. In their general form each conditional functional (resp. inclusion) dependency is a pair comprising of (i) a traditional functional (resp. inclusion) dependency and (ii) a pattern tableau consisting of tuples that enforce binding of semantically related data values. Traditional functional (resp. inclusion) dependencies are a special case of conditional functional (resp. inclusion) dependencies, in which the tableaux do not include tuples with patterns of data values. As opposed to traditional functional and inclusion dependencies that were developed mainly for schema design, conditional functional and inclusion dependencies aim to capture the consistency of the data, for data cleaning. As shown by the example above, conditional dependencies are capable of detecting more errors and inconsistencies than what their traditional counterparts can find.

In summary, integrity constraints specify a fundamental part of the semantics of the data. Indeed, errors and inconsistencies in real-world data often emerge as violations of integrity constraints. The more expressive the constraints are, the more errors and inconsistencies can be caught. On the other hand, as will be seen shortly, the expressive power of the constraints often comes with the price of extra complexity for finding database repairs.

Repair Models

Consider functional dependencies, inclusion dependencies, denial constraints, conditional functional dependencies and inclusion dependencies. Given a database instance D of a schema R, if D violates a set Σ consisting of these constraints, one can always edit D to obtain a consistent instance D' of R, such that D' satisfies Σ. An extreme case is to delete all tuples from D and thus get an empty D'. Such a fix is obviously impractical: it removes inconsistencies as well as correct information. Apparently database repairs should not be conducted with the price of losing information of the original data. This motivates the criterion for database repairs to minimally differ from the original data.

Several repair models have been proposed [1,6]. One model allows tuple deletions only, assuming that the information in the database D is inconsistent but is complete. In this model, a repair D' is a maximal subset of D that satisfies Σ. For example, consider Σ consisting of FD given above, and the database D shown in Fig. 1. Then a repair of D can be obtained by removing either t_3 or t_4 from the customer table.

Another model allows both tuple deletions and insertions. In this model, a repair D' is an instance of R such that (i) D' satisfies Σ, and (ii) the difference between D and D', i.e., $(D \setminus D') \cup (D' \setminus D)$, is minimal when D' ranges over all instances of R that satisfy Σ. As an example, let Σ consist of FD and CIND given above, and D be the database of Fig. 1. Then one can obtain a repair of D either by removing both t_3 and t_5, or by removing t_3 but inserting a tuple t' to the book table such that t_5 and t' agree on their id, title and price attributes.

A more practical model is based on updates, i.e., attribute-value modifications. To illustrate this, let us consider Σ consisting of all the constraints that have been encountered, i.e., FD, IND, DD, CFD1, CFD2, CFD3 and CIND given above, and the database D of Fig. 1. Observe that every tuple in the customer relation violates at least one of the (conditional) functional dependencies in Σ. In the two models mentioned above, the only way to find a repair is by removing all tuples from the customer table. However, it is possible that only some fields in a customer tuple are not correct, and thus it is an overkill to remove the entire tuple. A more reasonable fix is to update the tuples by, e.g., changing t_1[city] and t_2[city] to EDI (for CFD2), t_1[street] to Crichton (forCFD1), t_3[city] and t_4[city] to MH (for CFD3), t_3[street] to Mountain Ave (for FD), t_6[price] to 7.99 (for DD), and t_5[title] to Harry Porter (for CIND). This yields a repair in the update model.

An immediate question about the update model concerns what values should be changed and what values should be chosen to replace the old values. One should make the decisions based on both the accuracy of the attribute values to be modified, and the "closeness" of the new value to the original value. Following the practice of US national statistical agencies [10], one can define a cost metric as follows [2,8]. Assuming that a *weight* in the range [0,1] is associated with each attribute A of each tuple t in D, denoted by $w(t,A)$ (if $w(t,A)$ is not available, a default weight can be used instead). The weight reflects the confidence of the *accuracy* placed by the user in the attribute $t[A]$, and can be propagated via data provenance analysis in data transformations. For two values v,v' in the same domain, assume that a *distance function* $\text{dis}(v,v')$ is in place, with lower values indicating greater similarity. The cost of changing the value of an attribute $t[A]$ from v to v' can be defined to be:

$$\text{cost }(v, v') = w(t, A) \cdot \text{dis }(v, v')/\max\ (|v|, |v'|),$$

Intuitively, the more accurate the original $t[A]$ value v is and more distant the new value v' is from v, the higher the cost of this change. The similarity of v and v' is measured by $\text{dis}(v,v')/\max(|v|,|v'|)$, where $|v|$ is the length of v, such that longer strings with 1-character difference are closer than shorter strings with 1-character difference. The cost of changing the value of a tuple t to t' is the sum of $\text{cost}(t[A],t'[A])$ when A ranges over all attributes in t for which the value of $t[A]$ is modified. The cost of changing D to D', denoted by $\text{cost}(D',D)$, is the sum of the costs of modifying tuples in D. A repair of D in the update model is an instance D' of R such that (i) D' satisfies Σ, and (ii) $\text{cost}(D',D)$ is minimal when D' ranges over all instances of R that satisfy Σ.

The accuracy of a repair can be measured by precision and recall metrics, which are the ratio of the number of errors correctly fixed to the total number of changes made, and the ratio of the number of errors correctly fixed to the total number of errors in the database, respectively.

Methods for Finding Database Repairs

It is prohibitively expensive to find a repair of a dirty database D by manual effort. The objective of constraint-based database repair is to automatically find candidate repairs of D. These candidate repairs are subject to inspection and change by human experts. There have only been preliminary results on methods for finding quality candidate repairs, as outlined below.

Given a set Σ of integrity constraints, either defined on a schema R or discovered from sample instances of R, one first wants to determine whether or not Σ is dirty itself. That is, before Σ is used to find repairs of D, one has to check, at compile time, whether or not Σ is consistent or makes sense, i.e., whether or not there exists a nonempty database instance of R that satisfies Σ. For traditional functional and inclusion dependencies, this is not an issue: one can specify arbitrary functional and inclusion dependencies without worrying about their consistency. While conditional inclusion dependencies alone retain this nice property, it is no longer the case when it comes to conditional functional dependencies. For example, consider the following conditional functional dependencies: $\psi_1 = R(A = \text{true} \rightarrow B = b_1)$, $\psi_2 = R(A = \text{false} \rightarrow B = b_2)$, $\psi_3 = R(B = b_1 \rightarrow A = \text{false})$, and $\psi_4 = R(B = b_2 \rightarrow A = \text{true})$, where the domain of attribute A is Boolean. While each of these constraints can be separately satisfied by a nonempty database instance, there exists no nonempty instance that satisfies all of these constraints. Indeed, for any tuple t in an instance, no matter what Boolean value $t[A]$ has, these constraints force $t[A]$ to take the other value from the Boolean domain.

The consistency problem is already NP-complete for conditional functional dependencies alone [9], and it becomes undecidable for conditional functional and inclusion dependencies taken together [3]. In light of the complexity, the consistency analysis is necessarily conducted by effective heuristic methods, ideally with performance guarantee. There has been approximate algorithms developed for checking the consistency of conditional functional dependencies, and heuristic algorithms for conditional functional and inclusion dependencies taken together.

After Σ is confirmed consistent, one needs to detect the inconsistencies in the database D, i.e., to find all tuples in D that violate one or more constraints in Σ. It has been shown that it is possible to automatically generate a fixed number of SQL queries from Σ, such that these queries can find all violations in D. This strategy works when Σ consists of functional dependencies, inclusion dependencies, conditional functional and inclusion dependencies [9]. Better yet, the size of the queries is dependent upon neither the number of constraints in Σ nor the pattern tableaux in the conditional dependencies in Σ.

After all violations are identified, the next step is to find an accurate repair of D by fixing these violations. This is challenging: in the repair model based on attribute-value updates, the problem of finding a database repair is already NP-complete even when the database schema is fixed and only a fixed number of traditional functional (or inclusion) dependencies are considered [2].

To cope with the tractability of the problem, several heuristic algorithms have been developed, based on the cost model given above. A central idea of these algorithms is to separate the decision of which attribute values should be equal from the decision of what value should be assigned to these attributes. Delaying value assignment allows a poor local decision to be improved in a later stage of the repairing process, and also allows a user to inspect and modify a repair. To this end an equivalence class eq(t,A) can be associated with each tuple t in the dirty database D and each attribute A in t. The repairing is conducted by merging and modifying the equivalence classes of attributes in D. For example, if tuples t_1,t_2 in D violate a functional dependency $R(X \rightarrow A)$, one may want to fix the inconsistency by merging eq(t_1,A) and eq(t_2,A) into one, i.e., by forcing t_1 and t_2 to match on their A attributes. If a tuple t_1 violates an inclusion dependency $R_1[X] \subseteq R_2[Y]$, one may want to resolve the conflict by finding an existing tuple t_2 in the R_2 relation or inserting a new tuple t_2 into the R_2 table, such that for each corresponding attribute pair (A,B) in $[X]$ and $[Y]$, $t_1[A] = t_2[B]$ by merging eq(t_1,A) and eq(t_2,B) into one. A target value is assigned to each equivalence class when no more merging is possible.

Based on this idea, effective heuristic algorithms have been developed for repairing databases using traditional functional and inclusion dependencies (e.g., [8]). The algorithms modify tuple attributes in the right-hand side of a functional or inclusion dependency in the presence of a violation. This strategy, however, no longer works for conditional functional dependencies: the process may not even terminate if only tuple attributes in the right-hand side of a conditional functional dependency can be modified. Heuristic algorithms for repairing conditional functional dependencies have been developed [8], which are also based on the idea of equivalence classes but may modify tuple attributes in either the left-hand side or right-hand side of a conditional functional dependency in the presence of a violation.

Key Applications

Constraint-based database repairs have a wide range of applications in, e.g., data standardization, data quality tools, data integration systems, master data management, and credit-card fraud detection.

Future Directions

The study of constraint-based database repair is still in its infancy. There is naturally much more to be done. One topic for future research is to identify new integrity constraints that are capable of detecting inconsistencies and errors commonly found in practice, without incurring extra complexity. The second topic is to develop more accurate and practical repair models. The third topic is to find heuristic methods, with performance guarantees, for reasoning about integrity constraints used for data cleaning, such as their consistency and implication analyses. The fourth yet the most challenging topic is to develop scalable algorithms for finding database repairs with performance guarantee, such as to guarantee that the precision and recall of the repairs found are above a predefined bound with a high confidence.

Cross-references

▶ Data Cleaning
▶ Data Quality Models
▶ Database Dependencies
▶ Database Repair
▶ Functional Dependency
▶ Inconsistent Databases
▶ Record Linkage
▶ Relational Integrity Constraints
▶ Uncertain Databases

Recommended Reading

1. Arenas M., Bertossi L.E., and Chomicki J. Consistent query answers in inconsistent databases. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 68–79.
2. Bohannon P., Fan W., Flaster M., and Rastogi R. A cost-based model and effective heuristic for repairing constraints by value modification. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
3. Bravo L., Fan W., and Ma S. Extending dependencies with conditions. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 243–254.
4. Calì A., Lembo D., and Rosati R. On the decidability and complexity of query answering over inconsistent and incomplete databases. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2003, pp. 260–271.
5. Chomicki J. Consistent query answering: Five easy pieces. In Proc. 11th Int. Conf. on Database Theory, 2007, pp. 1–17.
6. Chomicki J. and Marcinkowski J. Minimal-change integrity maintenance using tuple deletions. Inf. Comput., 197(1–2): 90–121, 2005.
7. Chomicki J. and Marcinkowski J. On the computational complexity of minimal-change integrity maintenance in relational databases. Inconsistency Tolerance :119–150, 2005.
8. Cong G., Fan W., Geerts F., Jia X., and Ma S. Improving data quality: Consistency and accuracy. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 315–326.
9. Fan W., Geerts F., Jia X., and Kementsietsidis A. Conditional functional dependencies for capturing data inconsistencies. ACM Trans. Database Syst., 33(2), 2008.
10. Fellegi I. and Holt D. A. systematic approach to automatic edit and imputation. J. Am. Stat. Assoc., 71(353):17–35, 1976.
11. Lopatenko A. and Bertossi L.E. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In Proc. 11th Int. Conf. on Database Theory, 2007, pp. 179–193.
12. Wijsen J. Database repairing using updates. ACM Trans. Database Syst., 30(3):722–768, 2005.

Content Delivery Networks

▶ Storage Grid

Content-and-Structure Query

Thijs Westerveld[1,2]
[1]Teezir Search Solutions, Ede, Netherlands
[2]CWI, Amsterdam, Netherlands

Synonyms

CAS query; CO+S query

Definition

A content-and-structure query is a formulation of an information need in XML retrieval or, more generally, in semi-structured text retrieval that includes explicit information about the structure of the desired result.

Key Points

Content-and-structure query is a term from semi-structured text retrieval, used predominantly for XML retrieval. The term refers to a specific way of querying a structured document collection. In addition to describing the (topical) content of the desired result, content-and-structure queries include explicit hints about the structure of the desired result or the structure of the context it appears in. Content-and-structure queries are useful for users who have knowledge about the collection structure and want to express the precise structure of the information they are after. For example, they can express the granularity of the desired results, e.g., return *sections* about architecture, or they can express the structural context of the information they are looking for, e.g., return *sections* about architecture within documents about *Berlin*. It is up to the retrieval system to decide how to use the structural hints in locating the most relevant information. In INEX, the Initiative for the Evaluation of XML Retrieval [1], content-and-structure queries are known as *CAS* queries or CO+S queries (*Content-Only queries* with structural hints) and expressed in the NEXI language [2]. More information on query languages, including content-only and content-and-structured queries in the field of XML search can be found in [1].

Cross-references

▶ Content-Only Query
▶ NEXI
▶ XML Retrieval

Recommended Reading

1. Amer-Yahia S. and Lalmas M. XML search: languages, INEX and scoring. ACM SIGMOD Rec., 35(4):16–23, 2006.
2. Trotman A. and Sigurbjörnsson B. Narrowed extended xpath i (NEXI). In Advances in XML Information Retrieval: Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004. N. Fuhr, M. Lalmas, S. Malik, and Z. Szlavik (eds). Dagstuhl Castle, Germany, December 6–8, 2004, Revised Selected Papers , Vol. 3493. Springer, Berlin Heidelberg New York, GmbH, May 2005. http://www.springeronline.com/3-540-26166-4.

Content-based Image Retrieval (CBIR)

▶ Image Database

Content-Based Publish/Subscribe

Hans-Arno Jacobsen
University of Toronto, Toronto, ON, Canada

Definition

Content-based publish/subscribe is a communication abstraction that supports selective message dissemination among many sources and many sinks. The publication message content is used to make notification decisions. Subscribers express interest in receiving messages based on specified filter criteria that are evaluated against publication messages. Content-based publish/subscribe is an instance of the publish/subscribe concept.

Key Points

Content-based publish/subscribe is an instance of the publish/subscribe concept. In the content-based model clients interact by publishing messages and subscribing to messages through the publish/subscribe system. The key difference between the content-based and other publish/subscribe models is that the content of publication messages is used as the basis for disseminating publications to subscribers. Subscriptions consist of filters that specify subscriber interests making reference to publication message content. The publish/subscribe system matches publications against subscriptions by evaluating the publication message content against the filters expressed in subscriptions. The kind of content to subscribe to that exists in content-based publish/subscribe systems is either out-of-band information and must be know to clients, or is dynamically discoverable by clients based on additional support provided by the system.

A publication message published to the content-based publish/subscribe system is delivered to all subscribers with matching subscriptions. A subscription is a Boolean function over predicates. A publication matches a subscription, if the Boolean function representing it evaluates to true, otherwise the publication does not match the subscription.

As in the other publish/subscribe models, the content-based publish/subscribe model decouples the interaction among publishing data sources and subscribing data sinks. The same decoupling characteristics as discussed under the general publish/subscribe concept apply here as well. Specific realizations of this model found in practice vary in the exact decoupling offered. To properly qualify as publish/subscribe, at least the anonymous communication style must exist. That is publishing clients must not be aware of who the subscribing clients are and how many subscribing clients exist, and vice versa. Thus, content-based publish/subscribe enables the decoupled interaction of n sources with m sinks for $n, m \geq 1$.

In content-based publish/subscribe, the publication data model is defined by the data model underlying the definition of publication messages. The publication data model defines the structure and the type of publication messages processed by the system. In many approaches, a publication is a set of attribute-value pairs, where values are explicitly or implicitly typed. In explicit typing, each attribute-value pair has an additional type component specifying the type of the value. In implicit typing, no type is specified, and the type interpretation for matching is conveyed by the operator specified in the subscription referencing the attribute. The type-based publish/subscribe concept is a refinement of this based on programming language type theory.

Some content-based publish/subscribe approaches exist that define multi-valued attributes, where an attribute may have more than one value. Also, publication schemas and patterns have been introduced that specify certain attributes as required, while others are optional. Besides representing publications as attribute-value pairs, many other representations of publications have been introduced in the literature, such as XML, RDF, and strings.

The subscription language model is closely tied to the publication data model and defines the subscriptions the publish/subscribe system processes. The subscription language model that corresponds to the above described attribute-value pair-based publication data model, represents subscriptions as Boolean functions over predicates. Most content-based publish/subscribe systems process conjunctions of predicates only. In these systems, more general subscriptions must be represented as separate conjunctive subscriptions. A predicate is an attribute- operator-value triple that evaluates to true or false. Besides representing subscriptions as Boolean functions over predicates, many other representations of publications have been introduced in the literature, such as XPath, RQL, regular expressions, and keywords.

Subscription language and publication data model define subscriptions and publications processed by the publish/subscribe system. The matching semantic defines when a publication matches a subscription. Commonly the matching semantic is crisp; that is a publication matches a subscription or it does not. However, other semantics have been explored, such as an approximate match, a similarity-based match, or even a probabilistic match.

The publish/subscribe matching problem is stated as follows: Given a set of subscriptions and a publication, determine the subscriptions that match for the given publication. The publish/subscribe system can be interpreted as a filter that based on the subscriptions it stores, publications that do not match are filtered out, while those that match are forwarded to subscribers that have expressed interest in receiving information by registering subscriptions. The challenge is to efficiently solve this problem without computing a separate match between all subscriptions and the given publication. This is possible since in many applications, subscriptions share predicates, subsumption relationships exist among different subscriptions, and the evaluation of one predicate may allow to determine the result of other predicates without requiring explicit computation.

Content-based publish/subscribe systems differ in the publication data model, the subscription language model, the matching semantic, and the system architecture. In a system based on a centralized architecture, all publishers and subscribers connect to one and the same publish/subscribe system. In a system based on a distributed architecture, publishers and subscribers connect to one of many publish/subscribe systems that are interconnected in a federation. The federated publish/subscribe system offers the same functionality and solves the same matching problem as the centralized one.

Content-based publish/subscribe differs from topic-based publish/subscribe in that the entire message content is used for matching, while in a topic-based approach only the topic associated with a message is used.

Content-based publish/subscribe differs from database stream processing in that the publish/subscribe system processes publications of widely varying schemas. In the extreme case, every publication processed by the system could be based on a different schema. In stream processing, each data stream follows one and the same schema, which is an important assumption in the design of the stream query engine.

Content-based publish/subscribe has been an active area of research, since at least the late 1990s. The early work in the area was influenced from approaches in active databases, network management, and distributed systems. Many academic and industry research projects have developed content-based publish/subscribe systems. Various standards exhibit elements of the above described content-based publish/subscribe model. These standards are the CORBA Notification Service [3], the OMG Data Dissemination Service [4], the OGF's Info-D specification [2], and the Advanced Message Queuing Protocol [1].

Content-based publish/subscribe intends to support applications that need to highly selectively disseminate messages from one or more data sources to several data sinks. The mapping of sources to sinks can change dynamically with every message published and is fully determined by the publication content and the at publication time existing subscriptions. Given no change in the subscription set, one and the same message published twice, is sent to the same recipient set. Applications that require fine-grained filtering capabilities are ideally suited for realization with content-based publish/subscribe. Most existing publish/subscribe systems allow the application to dynamically change subscriptions at run-time. There are many applications that follow these characteristics. Examples include selective information dissemination, information filtering, database trigger processing, application-level firewalls, intrusion detection systems, and notification and altering services. Furthermore, recently it was demonstrated how higher-level applications can be build effectively with content-based publish/subscribe. Scenarios in this category are business activity monitoring, business process execution, monitoring and control of service level agreements, and automatic service discovery.

In the literature, the term content-based publish/subscribe refers to the above-described model and encompasses the centralized as well as the distributed realization of the publish/subscribe concept. In this context the terms matching and filtering are used interchangeably. The term content-based routing is reserved for the distributed realization of the model, where the publish/subscribe system is also referred to as a router, a broker, or a publish/subscribe message broker. In information retrieval, the publish/subscribe matching problem is referred to as information filtering. Subscriptions are then referred to as profiles or filters.

Cross-references

▶ Content-Based Routing
▶ Publish/Subscribe
▶ Type-Based Publish/Subscribe

Recommended Reading

1. AMQP Consortium. Advanced Message Queuing Protocol Specification, version 0–10 edition, 2008.
2. OGF. Information Dissemination in the Grid Environment Base Specifications, 2007.
3. OMG. Notification Service Specification, version 1.1, formal/04–10–11 edition, October 2004.
4. OMG. Data Distribution Service for Real-time Systems, version 1.2, formal/07–01–01 edition, January 2007.

Content-based Retrieval

▶ Multimedia Information Retrieval Model

Content-Based Video Retrieval

Cathal Gurrin
Dublin City University, Dublin, Ireland

Synonyms

Digital video search; Digital video retrieval

Definition

Content-based Video Retrieval refers to the provision of search facilities over archives of digital video content, where these search facilities are based on the outcome of an analysis of digital video content to extract indexable data for the search process.

Historical Background

As the volume of digital video data in existence constantly increases, the resulting vast archives of professional video content and UCC (User Created Content) are presenting an opportunity for the development of content-based video retrieval systems. Content-based video retrieval system development was initially lead by academic research such as the Informedia Digital Video Library [3] from CMU and the Físchlár Digital Video Suite [6] from DCU (Dublin City University). Both of these systems operated over thousands of hours of content, however digital video search has now become an everyday WWW phenomenon, with millions of items of digital video being indexed by the major WWW search engines and video upload sites. The early research focused content-based digital video systems, such as the offerings from CMU and DCU, exploited aspects of text search and content-based image search in order to provide intelligent indexing, retrieval, summarization and visualization of digital video content. In recent years, the emerging WWW search engines have focused on the textual indexing of large quantities of digital video, at the expense of performing complex and time-consuming visual content analysis.

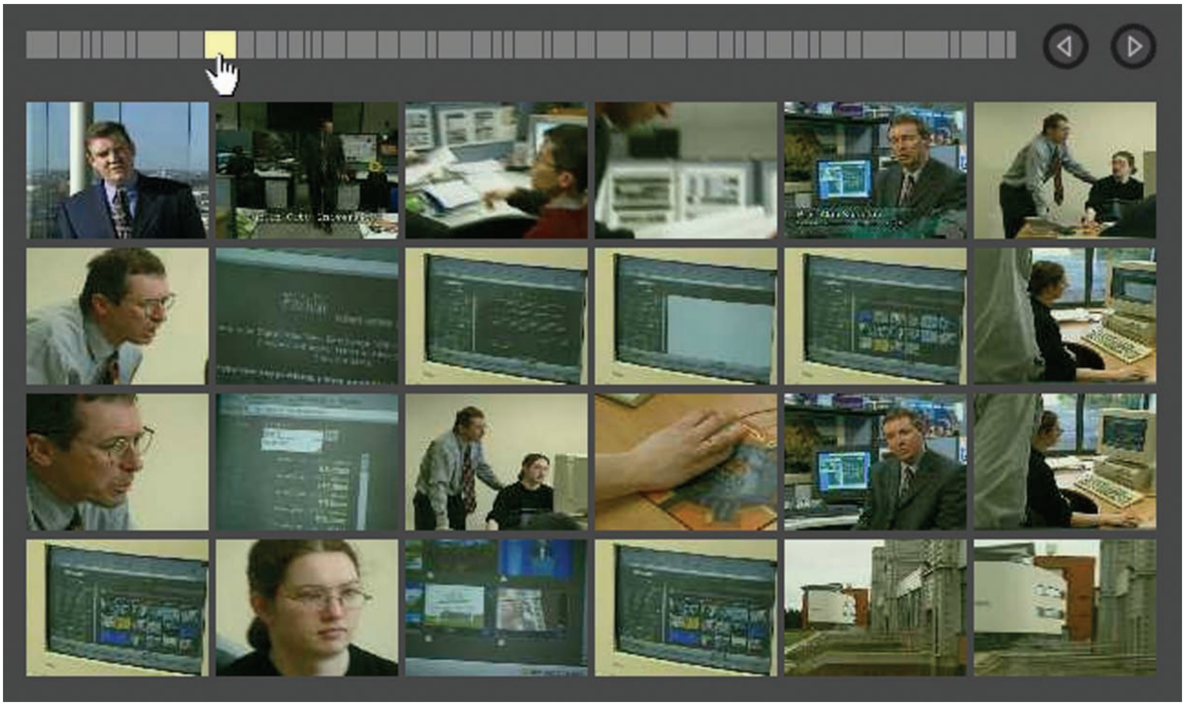
Foundations

The aim of a content-based video search system is to answer user queries with a (ranked) list of appropriate video content. Many different sources of video content exist and each needs to be treated differently. Firstly there is professional created content such as TV news content, documentaries, TV programmes, sports video and many others. Professional content is directed and polished content with many visual effects, such as a movie, music video or TV programme. Secondly there is the increasing quantities of UCC (User Created Content), such as home movie content or amateur/semi-professional content and finally there is security video content, which is increasingly being captured as the number of surveillance cameras in use increases. In addition to the type of content, another factor of key importance for content-based video search is the unit of retrieval.

Unit of Retrieval

Different content types will require different units of retrieval. In most WWW video search systems such as YouTube or Google Video, the unit of retrieval is the entire video content, which is sensible because most of the video content is short UCC or UUC (User Uploaded Content) clips. Retrieval of entire video units is not ideal for other types of content, for example TV news video, where the logical unit of retrieval would be a news story. There are a number of units of retrieval that are typically employed in addition to the entire video unit.

A *shot* in digital video is a sequence of continuous images (frames) from a single camera. A shot boundary is crossed when a recording instance ends and a new one begins. In many content-based video search systems the (automatically segmented) shot is the preferred unit of retrieval due to the fact that it is relatively easy to split a video file into its constituent shot in an automatic process called shot boundary detection [1]. Once a video stream has been segmented into shots, it can be browsed or indexed for subsequent search and retrieval, as shown in Fig. 1. A *scene* in digital video is a logical combination of video shots that together comprise some meaningful semantic unit. A *news story* is a special type of scene that is found in the context of news video. News stories can be automatically segmented from news video in a process called story-segmentation. This, like scene segmentation, is not a simple process, though reasonable accuracy can be achieved by replying on a number of individual cues from the video content and can be improved by exploiting the unique video production techniques of a particular broadcaster. For some video content, generating a *summary* or a *video skim* is a logical unit of retrieval. These summaries can be independent of any user need (context or query) or can be generated in response to a user need. Summaries have been successfully employed in the domain of field



Content-Based Video Retrieval. Figure 1. StoryBoard Interface from the Físchlár Video Retrieval System [2].

sports [4] or news summaries of reoccurring news topics [2]. Finally, as mentioned earlier, the entire video content may be returned in response to a user query, as is the case on many WWW video search engines in 2008.

Representing Video Content Visually on Screen

The quality of the interface to a content-based video retrieval system has a great effect on the usefulness of the system. In order to represent digital video visually, one or more keyframes (usually JPEG images) are usually extracted from the video to represent the content. These keyframes can then be employed for visual analysis of the video content by representing a video clip (typically a shot) by its keyframe and applying visual analysis techniques to the keyframe. In addition, these keyframes may be employed for visual presentation of video contents to support a degree of random access into the content. By processing video into a sequence of shots/scenes, and representing each shot/scene with one or more keyframes allows for the display of an entire video as a sequence of keyframes in what is called a StoryBoard interface, as shown in Fig. 1. Clicking on any keyframe would typically begin video playback from that point.

However, relying on simply presenting keyframes in screen can still require browsing through a very large information entity for long videos, maybe having over a hundred keyframes (shots) per hour. Therefore the ability to search within video content to locate a desired section of the video is desirable.

Searching Archives of Digital Video

The goals of supporting search through digital video archives are to (i) understand video content and (ii) understand how relevant content is likely to be to a user's query and to (iii) present the most highest ranked content for user consideration. We try to achieve these goals by indexing video content utilizing a number of sources (textual, audio and visual), either alone or in any combination. The unit of retrieval can be shots, scenes, stories, entire video units or any other unit of retrieval required.

Content-Based Retrieval using Text Sources The most common searching technology used for video retrieval in the WWW is content searching using proven text search techniques. This implies that it is possible to generate textual content (often referred to as a text surrogate) for the video. There are a number of sources of text content that can be employed to generate these textual surrogates, for example sources based on analyzing the digital video or the broadcast video stream:

- *Spoken words*, generated by utilizing a speech-to-text tool. The spoken words will provide an indication of the content of the video.
- *Written words*, extracted using a process of OCR (OCR – Optical Character Recognition) from the actual visual content of the video frames.
- *Professional closed caption annotation,* which are the closed caption (teletext) transcripts of video that accompanies much broadcast video content.

In addition there are many sources of textual evidence that can be employed that do not directly rely on the content of the digital video stream, and typically, these would be available for publicly available WWW digital video content:

- *Professionally annotated metadata* from the content provider which, if available, provides a valuable source of content for the textual surrogate.
- *Community annotated metadata* from general users of the content. On WWW video sharing sites users are encouraged to annotate comments about the video content and these annotations can be a valuable source of indexable content.

All of these sources of textual data can be employed alone, or in any combination to generate textual surrogates for video content (shots, scenes, stories or entire videos). Users can query such systems with conventional text queries and this is the way that most WWW video search engines operate. Text search through video archives is a very effective way to support search and retrieval and relies on well-known and proven text search techniques.

Content-Based Retrieval using Visual Sources

Digital Video, being a visual medium, can also be analyzed using visual analysis tools, which typically operate over individual keyframes to visually index each clip (typically a shot or scene). The visual content analysis tools are often borrowed from the domain of visual image analysis. The first generation of video analysis systems relied on modeling video with easily extractable low-level visual features such as color, texture and edge detection. However a significant 'semantic gap' exists between these low-level visual features and the semantic meaning of the video content, which is how a typical user would like to query a video search system. To help bridge this semantic gap, video content in the current generation of video search systems is processed to seek more complex semantic (higher-level or derived) visual concepts, such as people (faces, newsreaders), location (indoor/outdoor, cityscape/landscape), objects (buildings, cars, airplanes), events (explosions, violence) and production techniques such as camera motion. The output of these higher-level concept detectors can, with sufficient development and training, be successfully integrated (mainly as filters) into content-based video retrieval systems. However, the development of these concept detectors can be a difficult process and it is not reasonable to assume the development of tens of thousands of concept detectors to cover all concepts for the video archive. Research carried out by the Informedia team at CMU suggest that "concept-based" video retrieval with fewer than 5,000 concepts, detected with minimal accuracy of 10% mean average precision is likely to provide high accuracy results, comparable to text retrieval on the web, for a typical broadcast news video archive. Extending into other domains besides broadcast news may require some additional concepts. A review of image analysis techniques will provide more details of these semantic visual concept detectors and how they can be developed.

The output of easily extracted low-level feature analysis can be also employed in a content-based video retrieval system to support linking between visually similar content, though it is unlikely to be used to support direct user querying. Semantic features can form part of a user query, whereby a user, knowing the semantic factures extracted from a video archive, can specify semantic features that are required/not required in the result of a video search. For example, a user may request video content concerning forest fires, that also contains the feature 'fire'.

Content-Based Retrieval using Audio Sources

Apart from the speech-to-text there are other uses of audio sources for content-based video retrieval. For example, security video to identify non-standard audio events, such as a window breaking to provide a special access point to security video at this point. Key events in sports video can be identified using visual analysis (e.g., goal-mouth detection, or onscreen scoreboard changing) but also using audio analysis, for example crowd noise level or commentator excitement level.

Effective Retrieval

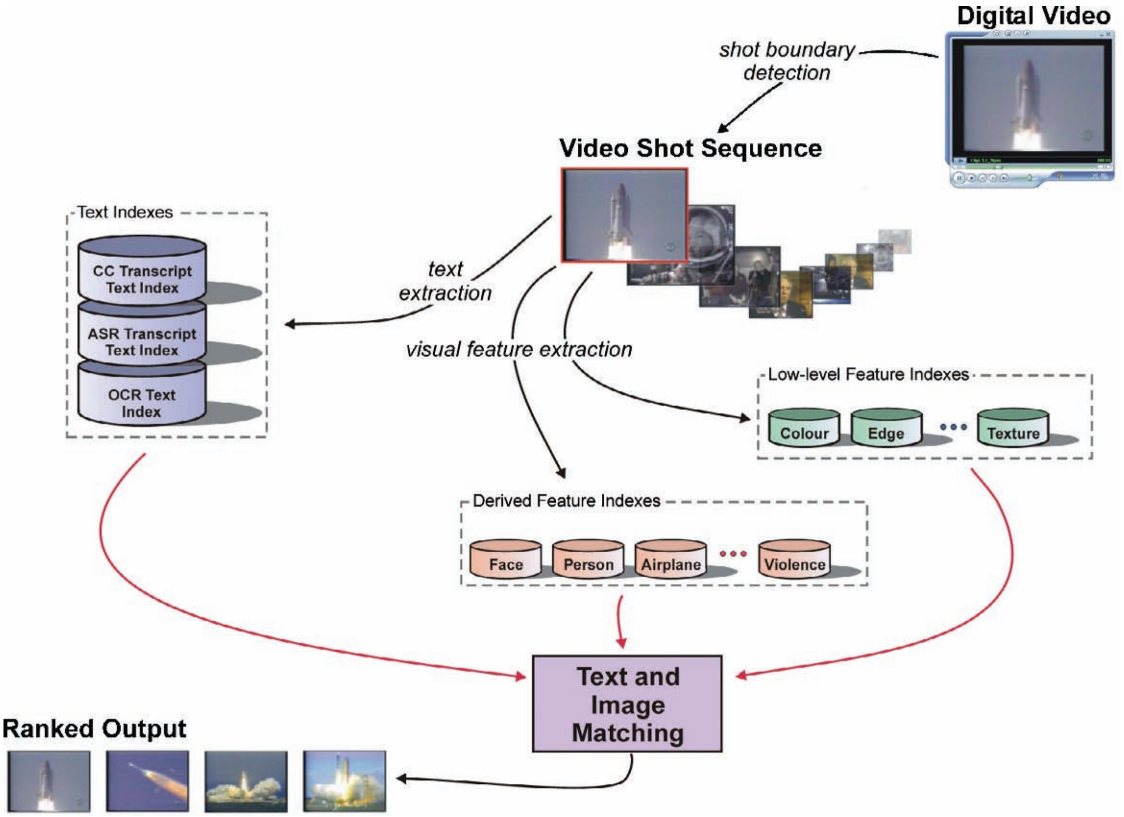
As can be seen from the current generation of WWW video search engines, most content-based video retrieval relies on user text queries to operate over text surrogates of video content. In typical situations the use of visual sources does not achieve any noticeable improvement in performance over using textual sources (such as CC text or ASR text). However, combining both sources of evidence can lead to higher performance than using either source alone. Figure 2 summarizes a typical shot-level content-based indexing process for digital video and illustrates some of the indexing options available.

Often successful academic video search systems allow the user to search to find the location in a piece of video which is likely to be of interest, with the user being encouraged to browse this area of the video by presenting keyframes from shots in the general video area (before and after).

Key Applications

The key application areas can be broadly divided into two categories; domain dependent video retrieval and generic (non-domain) video retrieval. In *domain dependent video retrieval*, the domain of the search system is limited, thereby allowing the search tool to exploit any domain dependent knowledge to develop a tailored and more effective content-based video retrieval system. Typical domains include news video where the unit of retrieval would a news story. Domain dependent additions for news video retrieval include anchor person detection to aid in the identification of news story bounds, inter-story linkage, story trails and timeline story progression. An example of a typical news story retrieval system is the Físchlár-News Digital Video Library that was operational from 2001 to 2004 at Dublin City University, and shown in Fig. 3.

Another example domain dependent application area is sports video, where research has been progressing on generating automatic summaries of field sports events and a third example is security video where research is ongoing into the automatic analysis of security footage to identify events of interest or even to identify and track individuals and objects in the video streams from many cameras in a given location.

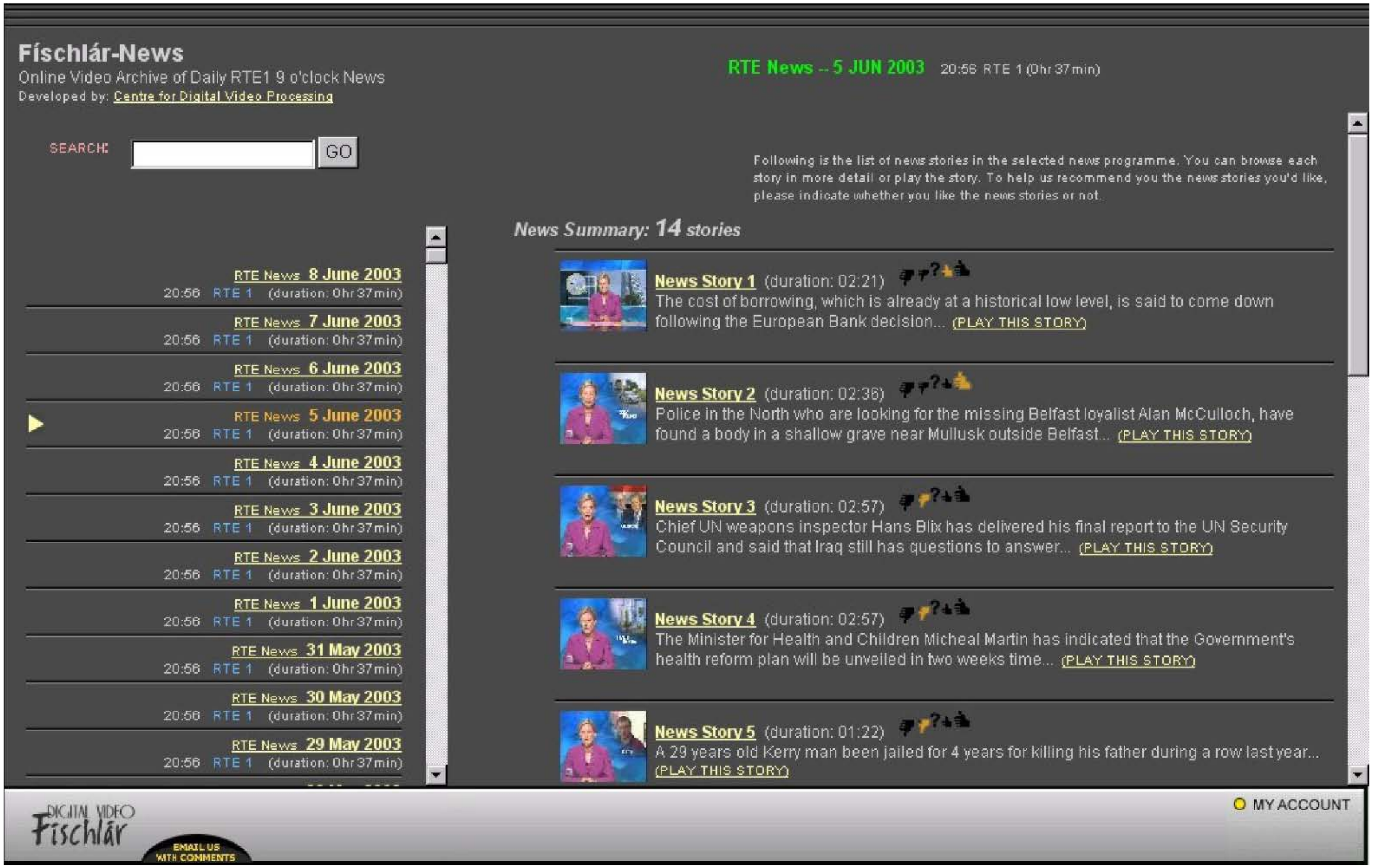


Content-Based Video Retrieval. Figure 2. The content-based indexing process for digital video (from Físchlár system at TRECVid in 2004) showing some text extraction, some low-level features and some higher level (derived) features (concepts).

Domain independent video retrieval attempts to index all types of content, such as general TV programmes or generic UCC. Given that there are no domain specific clues to exploit, retrieval is usually on textual indexing of a textual surrogate or extracted visual concepts, such as objects [5], locations, people, etc. The unit of retrieval would typically be a shot or an entire video clip, but could also be a non-shot unit that matches the user request. Domain independent video retrieval is most commonly seen in WWW video search engines, which index content based on text surrogates and returns entire videos in the result set.

Future Directions Future applications of, and research into content-based video search will likely focus on developing techniques for providing access to large archives of digital video content as broadcasters continue the process of digitizing their huge archives of programmes and the raw video content (rushes) that is used in the making of TV programmes. For rushes content especially, one will not be able to rely on text transcripts for indexing purposes. In addition, the ever increasing volume of UCC requires content-based retrieval techniques to be developed that will provide an improved semantic search facility over this content. Finally, the third point of research into the future will likely be in migrating content-based retrieval tools onto consumer devices (PVRs for example), which themselves are becoming capable of storing hundreds of hours of recorded video and UCC.

Experimental Results In the field of content-based video retrieval there exists an annual, worldwide forum for the evaluation of techniques for video search, called TRECVid [7] which began in 2001. The TRECVid workshop is (2007) part of the TREC [8] conference series is sponsored by the National Institute of Standards and Technology (NIST). In 2007, 54 teams from Europe, the Americas, Asia, and Australia participated in TRECVid. Over the course of the TRECVid evaluations, data employed has been either TV news, documentaries, educational video and rushes



Físchlár-News
Online Video Archive of Daily RTE1 9 o'clock News
Developed by: Centre for Digital Video Processing

SEARCH: GO

RTE News -- 5 JUN 2003 20:56 RTE 1 (0hr 37min)

RTE News 8 June 2003
20:56 RTE 1 (duration: 0hr 37min)

RTE News 7 June 2003
20:56 RTE 1 (duration: 0hr 37min)

RTE News 6 June 2003
20:56 RTE 1 (duration: 0hr 37min)

RTE News 5 June 2003
20:56 RTE 1 (duration: 0hr 37min)

RTE News 4 June 2003
20:56 RTE 1 (duration: 0hr 37min)

RTE News 3 June 2003
20:56 RTE 1 (duration: 0hr 37min)

RTE News 2 June 2003
20:56 RTE 1 (duration: 0hr 37min)

RTE News 1 June 2003
20:56 RTE 1 (duration: 0hr 37min)

RTE News 31 May 2003
20:56 RTE 1 (duration: 0hr 37min)

RTE News 30 May 2003
20:56 RTE 1 (duration: 0hr 37min)

RTE News 29 May 2003
20:56 RTE 1 (duration: 0hr 37min)

Following is the list of news stories in the selected news programme. You can browse each story in more detail or play the story. To help us recommend you the news stories you'd like, please indicate whether you like the news stories or not.

News Summary: 14 stories

News Story 1 (duration: 02:21)
The cost of borrowing, which is already at a historical low level, is said to come down following the European Bank decision... (PLAY THIS STORY)

News Story 2 (duration: 02:36)
Police in the North who are looking for the missing Belfast loyalist Alan McCulloch, have found a body in a shallow grave near Mullusk outside Belfast... (PLAY THIS STORY)

News Story 3 (duration: 02:57)
Chief UN weapons inspector Hans Blix has delivered his final report to the UN Security Council and said that Iraq still has questions to answer... (PLAY THIS STORY)

News Story 4 (duration: 02:57)
The Minister for Health and Children Micheal Martin has indicated that the Government's health reform plan will be unveiled in two weeks time... (PLAY THIS STORY)

News Story 5 (duration: 01:22)
A 29 years old Kerry man been jailed for 4 years for killing his father during a row last year... (PLAY THIS STORY)

Content-Based Video Retrieval. Figure 3. Físchlár-News, a domain dependent content-based video retrieval system.

(Rushes content, is the unproduced content that is used to prepare TV programming.) content. TRECVid has organized a number of tasks for the annual evaluations which may change each year. The tasks evaluated, 2001, have included shot boundary determination, interactive and automatic (no query modification or browsing) video search, high-level concept detection, story boundary determination for TV news and camera motion analysis, among others.

In addition to TRECVid, other evaluation forums also exist such as Video Analysis and Content Extraction (VACE) which is a US program that addresses the lack of tools to assist human analysts monitor and annotate video for indexing. The video data used in VACE is broadcast TV news, surveillance, Unmanned Aerial Vehicle, meetings, and ground reconnaissance video. Other evaluation forums such as the French ETISEO and EU PETS evaluations have evaluated content-based retrieval (event detection and object detection) from surveillance video. ARGOS, sponsored by the French government, evaluated tasks similar to TRECVid and employed video data from TV news, scientific documentaries and surveillance video archives.

Some summary findings from content-based video retrieval research are that employing visual analysis of the video content does not provide a significant increase in search performance over using text transcripts, that text transcripts provide the single most important clue for searching content, that employing as many text sources as possible aids text search quality, and finally that, incorporating visual content search can improve retrieval over that of text indexing alone. The performance of visual indexing tools suggests that this is an unsolved problem with much research needed. As an example, the highest accuracy attained (in terms of Inferred Average Precision) for the twenty visual concepts evaluated at TRECVid in 2007 are shown in Table 1.

Finally, it should be noted that the interface to an interactive video search system (for example [3,6]) can make a huge difference for effective content-based video search and retrieval. Content searching through text transcripts can locate the area of the video, but a good storyboard interface to find the exact video shot of interest is a valuable addition.

Data Sets The TRECVid evaluation framework provides a number of datasets to support the comparative and repeatable evaluation of TREC. Since 2001, these datasets, along with the associated queries and relevance judgements are available. The video data employed in these datasets comes from various sources, such as the video from the Movie Archive of the Internet Archive, news video data in a number of languages (English, Arabic and Chinese) and rushes content. Datasets used in other evaluation forums are also available

Content-Based Video Retrieval. Table 1. Inferred Average Precision (infAP (In terms of infAP, a value of 1.0 infers that the technique locates only correct examples of the concept, whereas a value of 0.0 infers that the technique only locates incorrect examples)) measurement for the top performing techniques for visual concept detection at the TRECVid workshop in 2007

CONCEPT	infAP	CONCEPT	infAP
Sports	0.144	Computer/TV screen	0.209
Weather	0.062	US flag	0.041
Office	0.222	Airplane	0.226
Meeting	0.279	Car	0.265
Desert	0.155	Truck	0.108
Mountain	0.12	Boat/ship	0.212
Waterscape/ waterfront	0.374	People marching	0.104
Police/security	0.046	Explosion/fire	0.069
Military personnel	0.081	Maps	0.236
Animal	0.249	Charts	0.225

Cross-references

▶ Video Abstraction
▶ Video Content Analysis
▶ Video Content Modeling
▶ Video Metadata
▶ Video Representation
▶ Video Scene and Event Detection
▶ Video Segmentation
▶ Video Shot Detection
▶ Video Skimming
▶ Video Summarization

Recommended Reading

1. Browne P., Smeaton A.F., Murphy N., O'Connor N., Marlow S., and Berrut C. Evaluating and combining digital video shot boundary detection algorithms. In Proc. IMVIP 2000 – Irish Machine Vision and Image Processing Conference, 2000, pp. 93–100.

2. Christel M.G., Hauptmann A.G., Wactlar H.D., and Ng T.D., Collages as dynamic summaries for news video. In Proc. 10th ACM Int. Conf. on Multimedia, 2002, pp. 561–569.
3. Hauptmann A. lessons for the future from a decade of informedia video analysis research, image and video retrieval. In Proc. 4th Int. Conf. Image and Video Retrieval, 2005, pp. 1–10.
4. Sadlier D. and O'Connor N. Event detection in field sports video using audio-visual features and a support vector machine. IEEE Trans. Circuits Syst. Video Technol., 15(10):1225–1233, 2005.
5. Sivic J. AND Zisserman A. Video Google: a text retrieval approach to object matching in videos. In Proc. 9th IEEE Conf. Computer Vision, Vol. 2, 2003, pp. 1470–1477.
6. Smeaton A.F., Lee H., and Mc Donald K. Experiences of creating four video library collections with the Físchlár system. Int. J. Digit. Libr., 4(1):42–44, 2004.
7. Smeaton A.F., Over P., and Kraaij W. Evaluation campaigns and TRECVid. In Proc. 8th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, 2006, pp. 321–330.
8. http://trec.nist.gov Last visited June '08.

Content-Only Query

Thijs Westerveld[1,2]
[1]Teezir Search Solutions, Ede, The Netherlands
[2]CWI, Amsterdam, The Netherlands

Synonyms

Content-only query; CO query

Definition

A content-only query is a formulation of an information need in XML retrieval or, more generally, in semi-structured text retrieval that does not contain information regarding the structure of the desired result.

Key Points

Content-only query or CO query is a term from semi-structured text retrieval, used predominantly for XML retrieval. The term refers to a specific way of querying a semi-structured document collection. Content-only queries ignore the structure of the collection and only refer to the (topical) content of the desired result. In that sense, they are similar to the keyword queries typically used in traditional information retrieval systems or in web search engines. The fact that structural information is lacking from the query formulation does not mean structure does not play a role. When a content-only query is posed, it is up to the retrieval system to decide the appropriate level of granularity to satisfy the information need. This contrasts so-called *content-and-structure queries* where the user specifies structural clues regarding the desired result. More information on query languages, including content-only and content-and-structured queries in the field of XML search can be found in [1].

Cross-references

- ▶ Content-and-structure query
- ▶ NEXI
- ▶ Xml Retrieval

Recommended Reading

1. Amer-Yahia S. and Lalmas M. XML search: languages, INEX and scoring. ACM SIGMOD Rec., 35(4):16–23, 2006.

Content-oriented XML Retrieval

▶ XML Retrieval

Context

Opher Etzion
IBM Research Lab in Haifa, Haifa, Israel

Synonyms

Life-span (in part); Space-span (in part)

Definition

A context is a collection of semantic dimensions within which the event occurs. These dimensions may include: temporal context, spatial context, state-related context and reference-related context.

Key Points

Event processing is being done within context, which means that an event is interpreted differently in different contexts, and may trigger different actions in different contexts, or be irrelevant in certain context. In the event processing network, each agents operates within a single context. While the term context has been associated with the spatial dimension, in event processing it is most strongly associated with the temporal dimension.

Each context-dimension may be specified either explicitly, or by using higher level abstractions.

Examples are:

- Temporal context:
  - Explicit: Everyday between 8AM–5PM EST.
  - Implicit: From sunrise to sunset.
  - Mixed: Within two hours from admission to the hospital.
- Spatial context:
  - Explicit: Within 1 KM from coordinate $+ 51^\circ\ 3'\ 45.71''$, $-1^\circ\ 18'\ 25.56''$.
  - Implicit: Within the borders of the city of Winchester.
  - Mixed: Within 1 KM north of the border between Thailand and Laos.
- State-oriented context:
  - Explicit: When "red alert" is present.
  - Implicit: During traffic jam in the area.
- Reference-oriented context:
  - Explicit: Context-instance for each platinum-customer with credit-limit > $1M.
  - Implicit: Context-instance for each "angry customer."

Note that the state-oriented dimension is different, since it does not relate to the event itself, and is global in nature. A context may consist of one dimension only or combination of dimensions. The reference-oriented context is mainly used to partition the event space. Context instances may or may not cover the entire space of possibilities, a context can also be created from binary operations on contexts (union, intersection, difference).

Cross-references

▶ Complex Event Processing
▶ Event Processing Network
▶ Retrospective Event Processing

Recommended Reading

1. Adi A., Biger A., Botzer D., Etzion O., and Sommer Z. Context awareness in Amit. In Proc. 5th Annual Workshop on Active Middleware Services, 2003, pp. 160–167.
2. Barghouti N.S. and Krishnamurthy B. Using event contexts and matching constraints to monitor software processes. In Proc. 17th Int. Conf. on Software Eng., 1995, pp. 83–92.
3. Buvac S. Quantificational logic of context. In Proc. 10th National Conf. on AI, 1996, pp. 600–606.
4. Hong C., Lee K., Suh Y., Kim H., Kim H., and Lee H. Developing context-aware system using the conceptual context model. In Proc. 6th IEEE Int. Conf. on Information Technology, 2006, pp. 238.
5. Rakotonirainy A., Indulska J., Loke S.W., and Zaslavsky A. Middleware for reactive components: An integrated use of context, roles, and event based coordination. In Proc. IFIP/ACM Int. Conf. on Dist. Syst. Platforms, 2001, pp. 77–98.

Context-aware Interfaces

▶ Adaptive Interfaces

Contextual Advertising

▶ Web Advertising

Contextualization

Jaana Kekäläinen, Paavo Arvola, Marko Junkkari
University of Tampere, Tampere, Finland

Definition

In relation to structured text retrieval, contextualization means estimating the relevance of a given structural text unit with information obtainable from – besides the unit itself – the surrounding structural text units, that is, from the context of the unit. From now on, structural text units are referred to as elements in accordance with [4]. In other words, in contextualization it is assumed that the context of an element gives hints about the relevance of the element.

Historical Background

Structured information retrieval typically addresses documents marked-up with, for instance, SGML or XML. In this article, XML documents are used as a sample case of structured documents. These documents have a hierarchical structure, which is often represented as a tree. In structured text retrieval, like in information retrieval (IR) in general, querying is based on words representing the information needed. Structural conditions, concerning the tree, may or may not be added to the query. An information retrieval system (IRS) returns a list of elements ranked by their retrieval status values (RSV), which are scores given by the IRS. Typically, RSVs are based on the statistics of words (cf. definitional entry *Term statistics*) appearing in the element and the query, although other features of the document or its structure may be used in addition.

The idea of XML retrieval is not to return whole documents but those elements that are best matches to the query–relevant elements with the least irrelevant content. The length of the textual content varies as the size of the elements varies in the hierarchy, so that descendant elements often have less text than their ancestors. As a consequence, small elements down in the hierarchy may have too few words in common with the query, that is, too little evidence to be matched with the query, although they might be more exact matches than their ancestors. This problem, known as the vocabulary mismatch, is typical for text retrieval, and is caused by natural language allowing several ways to refer to objects. However, elements are often dependent on each other because of textual cohesion. Thus, one solution is to use the context of the element to give more evidence about the subject of the element. One could say that "good elements" appear in "good company." This approach was first proposed by Sigurbjörnsson, Kamps, and de Rijke in [8].

Another problem related to the nested structure of structured texts is the calculation of word statistics. There are no obvious indexing units like in nonstructured (flat) text retrieval (cf. entry *Indexing units*). The calculation of word weights is challenging since the length of the elements vary, which has effects on word frequencies. Moreover, inverted element frequencies, corresponding to inverted document frequencies (*idf*s) in weight calculation, vary depending on the indexing unit. As a solution for this, the concept of *document pivot* factor (originally introduced in [9] for classical document retrieval) is suggested by Mass and Mandelbrod [6] to scale the final relevance status value of an element in XML IR. In [6] the scaling is based on the document pivot factor, RSV of the topmost ancestor (the root element), and the RSV of the element. This can be regarded as contextualization though for different reasons than in the first mentioned case.

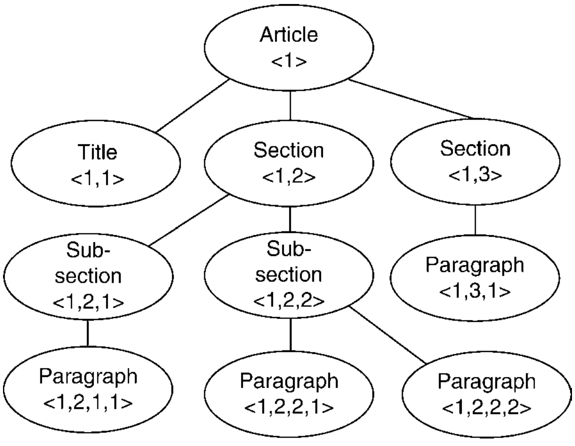
Foundations

An XML document consists of elements, which in turn may contain smaller elements. If an element x contains immediately another element y, then x is called the parent of y, whereas y is called a child of x. Any element containing x is an ancestor of y, and y is a descendant of those elements. A sample XML document is represented as a tree in Fig. 1. The document is an article consisting of a title, sections, subsections, and paragraphs. All elements are labeled with Dewey indices for reference.

Depending on how a collection is organized, an element may be viewed at various levels of context. For example, assuming that documents follow an article-section-subsection-paragraph division as in the sample, then the article, the section and the subsection form different levels of context for a paragraph. Further, a subsection can be viewed in the contexts of the section or article. The length of the path from the context element to the element at hand determines the level of context. For example, the parent of an element determines the first level context; the ancestor with the path length 2 determines the second level context, etc. The root element forms the topmost context.

As an example, the paragraph labeled <1,2,2,1> in Fig. 1 is examined. Now Subsection <1,2,2> forms the first level context and Section <1,2> the second level context of this paragraph. The article is the root element, or it determines the topmost context of this paragraph. In turn, Section <1,2> forms the first level context, and the article the second level (or topmost) context of Subsection<1,2,2>. The article possesses no context.

The idea of contextualization is based on the assumption that an element in a relevant context should be ranked higher than a similar element in a nonrelevant context. In contextualization, the RSV of an element is tuned by the RSV of its context element(s). If the RSV of the context is low (predicting nonrelevance), the RSV



Contextualization. Figure 1. A sample document tree.

of the element should be decreased; if the RSV of the context is high (predicting relevance), the RSV of the element should be increased. Here low and high are relative to the element RSV and the RSVs of other contexts.

As an example three special contextualization cases are considered – parent, root or all ancestors as a context. In defining contextualization function the following notations, presented in [1], are used:

parent(e) yields the parent element of the element e,
root(e) yields the root element of the element e,
$\langle e_1, e_2, \ldots, e_n \rangle$ means the path from the root element e_1 to its descendant e_n such that $\forall i \in \{1,\ldots,n-1\}$ holds $e_i = parent(e_{i+1})$, and
$w(q, e)$ denotes the RSV of the element e with respect to the query q.

With relation to the query expression q, contextualization based on the nearest context (parent) of the element e can be calculated by averaging the RSVs of the element e and its parent element. Averaging is applied as an example because it increases the RSV of the element whose context's RSV is higher, and decreases the RSV of such elements whose context's RSV is lower. The function for this is denoted by the symbol c_p:

$$c_p(q, e) = \frac{w(q, e) + w(q, parent(e))}{2}.$$

The contextualization by the topmost context is denoted by the function symbol c_r and it can be calculated by averaging the RSVs of the element e and its root element.

$$c_r(q, e) = \frac{w(q, e) + w(q, root(e))}{2}.$$

The contextualization function c_t is called tower contextualization and it yields the average of the RSVs of all the elements within the path from the root element to the element e, that is, all ancestors.

$$c_t(q, e) = \frac{\sum_{i=1}^{n} w(q, e_i)}{n},$$

when $e_1 = root(e)$ and $e_n = e$ in the path $\langle e_1, \ldots, e_n \rangle$.

Now, Dewey indices are utilized as a method for handling the XML tree structure. In the following, the contextualization for any path between the element and the root is generalized. The notations used are as follows:

- The symbol ξ is used for denoting a Dewey index labeling an element. An element possessing the index ξ is called the ξ *element*.
- The set of indices related to the XML collection at hand is denoted by *IS*.
- The length of an index ξ is denoted by $len(\xi)$. For example $len(\langle 1,2,2,3 \rangle)$ is 4.
- The index $\langle i \rangle$ consisting of an integer i (i.e., its length is 1) is called *root index* and it is associated with the whole document.
- Let ξ be an index and i a positive integer, then the *cutting operation* $\delta_i(\xi)$ selects the sub-index of the index ξ consisting of its i first integers. For example if $\xi = \langle a,b,c \rangle$ then $\delta_2(\xi) = \langle a,b \rangle$. In terms of the cutting operation the root index at hand is denoted by $\delta_1(\xi)$ whereas the index of the parent element can be denoted by $\delta_{len(\xi)-1}(\xi)$.

A general contextualization function C has the following arguments: q, ξ, and g. The arguments q (query) and ξ (index) are defined above. The argument g is called *contextualization vector* and is set-theoretically represented as a tuple, consisting of values by which elements between the root element and ξ element are weighted in contextualization. The length of g is $len(\xi)$. When referring to the ith position of the contextualization vector g, the notation $g[i]$ is used. For example, if $g = \langle a,b,c \rangle$ then $g[2] = b$. The value in $g[i]$ relates to the element with the index $\delta_i(\xi)$. For example, if $\xi = \langle 1,2,2 \rangle$ then g is the 3-tuple $\langle a,b,c \rangle$ where a is the contextualization weight of the root element $\langle 1 \rangle$ (i.e., the element with index $\delta_1(\xi)$), b is the contextualization weight of the $\langle 1,2 \rangle$ element (i.e., the element with index $\delta_2(\xi)$), and c is the weight of the $\langle 1,2,2 \rangle$ element (i.e., the element with the index $\delta_{len\,(\xi)}(\xi)$). The contextualized RSVs of elements are calculated by weighted average based on the contextualization vector and the index at hand. In the sample case above the contextualized RSV is calculated as $(a * w(q, \langle 1 \rangle) + b * w(q, \langle 1,2 \rangle) + c * w(q, \langle 1,2,2 \rangle))/(a + b + c)$. Contextualization is applied only to those elements whose basic RSV is not zero. The general contextualization function C is formally defined:

$$C(q, \xi, g) = \begin{cases} 0, \text{ if } w(q, \xi) = 0 \\ \dfrac{\sum_{i=1}^{len(\xi)} g[i] \cdot w(q, \delta_i(\xi))}{\sum_{i=1}^{len(\xi)} g[i]}, \text{ otherwise.} \end{cases}$$

The values in g are not bound to any range. This means that in terms of g, different levels of the context can be weighted in various ways. For example, weighting may increase or decrease toward the topmost context (root element).

Now, parent and root contextualization with the given generalized notation are considered. For simplicity, binary contextualization weights are used, that is, only such cases where the values of *g* are either 1 or 0. Zero value means that the corresponding element is not taken into account in the contextualization. With relation to a query expression *q*, the contextualization based on the first level (parent) context of the ξ element is calculated using the contextualization vector where two last elements have the value 1 and the others zero value. This function is denoted $c_p(q, \xi)$ and defined as follows:

$$c_{\mathrm{p}}(q, \xi) = C(q, \xi, g)$$

where g =

$$\begin{cases} g[len(\xi)] = 1 \\ g[len(\xi) - 1] = 1, \text{when } \mathrm{len}(\xi) > 1 \\ g \overset{len(\xi)-2}{\underset{i=1}{[i]}} = 0, \text{when } \mathrm{len}(\xi) > 2 \end{cases}$$

The contextualization by the topmost context (or by the root element) is denoted by the function symbol c_r. In this case the weights for the first and the last element are 1 and the other weights are 0 in the contextualization vector.

$$c_r(q, \xi) = C(q, \xi, g) \text{ where } g = \begin{cases} g[len(\xi)] = 1 \\ g[1] = 1 \\ g \overset{len(\xi)-1}{\underset{i=2}{[i]}} = 0, \\ \text{when } len(\xi) > 2. \end{cases}$$

There are alternative interpretations and presentations of the idea of contextualization, for example [8,6,10]. The idea of mixing evidence from the element itself and its surrounding elements was presented for the first time by Sigurbjörnsson, Kamps, and de Rijke [8]. They apply language modeling based on a mixture model. The final RSV for the element is combined from the RSV of the element itself and the RSV of the root element as follows:

$$w_{mix}(q, e) = lp(e) + \alpha \cdot w(q, root(e)) + (1 - \alpha) \cdot w(q, e),$$

where the notation is as given above; $w_{mix}(q,e)$ is the combined RSV for the element, $lp(e)$ is the length prior, and α is a tuning parameter.

In [6] independent indices are created for different – selected – element types. Some indices have sparse data compared with others, that is, all the words of root elements are not contained in lower level elements. This has effects on inverted element frequencies and comparability of the word weights across the indices. To resolve the problem the final element RSV is tuned by a scaling factor and the RSV of the root element. With the notation explained above this can be represented as follows:

$$w_p(q, e) = DocPivot \cdot w(q, root(e)) + (1 - DocPivot) \cdot w(q, e)$$

where $w_p(q,e)$ is the pivoted RSV of the element, and *DocPivot* is the scaling factor.

In [10], the structural context of the element is utilized for scoring elements. Original RSVs are obtained with the Okapi model [7]. For this, word frequencies are calculated for elements rather than documents, and normalized for each element type. The combined RSV for the element is calculated as a sum of the RSV of the context and the original RSV for the element, each score multiplied by a parameter. Machine learning approach is applied for learning the parameters. Let $x = \{t_1, t_2, \ldots, t_d\}$ be a vector of features representing the element *e*. The features are RSVs of the element *e* and its different contexts. Then the combined RSV is

$$f\omega(x) = \sum_{j=1}^{d} \omega_j t_j,$$

where $\omega = \{\omega_1, \omega_2, \ldots, \omega_d\}$ are the parameters to be learned. The approach was tested with the parent and root as the contexts.

Key Applications

Contextualization and similar approached have been applied in XML retrieval with different retrieval models: tf-idf based ranking and structural indices [1], the vector space model [6], and the language modeling [8]. The key application is element ranking. The results obtained with different retrieval systems seem to indicate that the root contextualization is the best alternative.

In traditional text retrieval a similar approach has been applied to text passages [3,5]. The idea of contextualization is applicable to all structured text documents; yet the type of the documents to be retrieved has effects on contextualization. Lengthy documents with one or few subjects are more amenable for the method than short documents, or documents with diverse subjects; contextualization might not work in an encyclopedia with short entries, but can improve effectiveness, say, in the retrieval of the elements of scientific articles.

Experimental Results

The effectiveness of contextualization in XML retrieval has been experimented with the INEX test collection consisting of IEEE articles [1,2]. All three contextualization types mentioned above (parent, root, and tower contextualization) were tested. The results show clear improvement over a non-contextualized baseline; the best results were obtained with the root and tower contextualization. Additionally, approaches suggested in [8,6,10] were tested with the same INEX collection and reported to be effective compared with non-contextualized baselines, that is, compared with ranking based on elements' basic RSVs.

Cross-references

► Indexing Units
► Term Statistics for Structured Text Retrieval
► XML Retrieval

Recommended Reading

1. Arvola P., Junkkari M., and Kekäläinen J. Generalized contextualization method for XML information retrieval. In Proc. Int. Conf. on Information and Knowledge Management, 2005, pp. 20–27.
2. Arvola P., Junkkari M., and Kekäläinen J. Query evaluation with structural indices. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 134–145.
3. Callan J.P. Passage-level evidence in document retrieval. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 302–310.
4. Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C Recommendation 16 August 2006. Available at: http://www.w3.org/TR/xml/[retrieved 17.8.2007].
5. Kaszkiel M., Zobel J., and Sacks-Davis R. Efficient passage ranking for document databases. ACM Trans. Infor. Syst., 17(4):406–439, 1999.
6. Mass Y. and Mandelbrod M. Component ranking and automatic query refinement for XML retrieval. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 73–84.
7. Robertson S.E., Walker S., Jones S., Hancock-Beaulieu M.M., and Gatford M. Okapi at TREC-3. In Proc. The 3rd Text Retrieval Conf., 1994, pp. 500–226.
8. Sigurbjörnsson B, Kamps J., and De Rijke M. An element-based approach to XML retrieval. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2003, 19–26. Available at: http://inex.is.informatik.uni-duisburg.de:2003/proceedings.pdf [retrieved 29.8.2007].
9. Singhal A., Buckley C., and Mitra M. Pivoted document length normalization. In Proc. 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1996, 21–29.
10. Vittaut J.-N. and Gallinari P. Machine learning ranking for structured information retrieval. In Proc. 28th European Conf. on IR Research, 2006, 338–349.

Continuous Backup

► Continuous Data Protection (CDP)

Continuous Data Feed

► Data Stream

Continuous Data Protection

Kenichi Wada
Hitachi, Ltd, Tokyo, Japan

Synonyms

Continuous backup
CDP

Definition

CDP is a data protection service capturing data changes to storage, often providing the capability of restoring any point in time copies.

Key Points

CDP differs from usual backups in that users do not need to specify the point in time until they recover data from backups. From an application point of view, every time when it updates data in an original volume, CDP keeps updates. In case of recovery, when users specify the point in time, CDP creates the point in time copy from an original volume and updates.

In several CDP implementations, users can specify the granularities of restorable objects which help them to specify the point in time easily. For example, restorable objects range from crash-consistent images to logical objects such as files, mail boxes, messages, database files, or logs.

Cross-references

► Backup and Restore

Recommended Reading

1. Laden G., et al. Architectures for Controller Based CDP. In Proc. 5th USENIX conf. on File and Storage Technologies, 2007, pp. 107–121.

Continuous Monitoring of Spatial Queries

Kyriakos Mouratidis
Singapore Management University, Singapore, Singapore

Synonyms

Spatio-temporal stream processing

Definition

A continuous spatial query runs over long periods of time and requests constant reporting of its result as the data dynamically change. Typically, the query type is range or nearest neighbor (NN), and the assumed distance metric is the Euclidean one. In general, there are multiple queries being processed simultaneously. The query points and the data objects move frequently and arbitrarily, i.e., their velocity vectors and motion patterns are unknown. They issue location updates to a central server, which processes them and continuously reports the current (i.e., updated) query results. Consider, for example, that the queries correspond to vacant cabs, and that the data objects are pedestrians that ask for a taxi. As cabs and pedestrians move, each free taxi driver wishes to know his/her closest client. This is an instance of continuous NN monitoring. Spatial monitoring systems aim at minimizing the processing time at the server and/or the communication cost incurred by location updates. Due to the time-critical nature of the problem, the data are usually stored in main memory to allow fast processing.

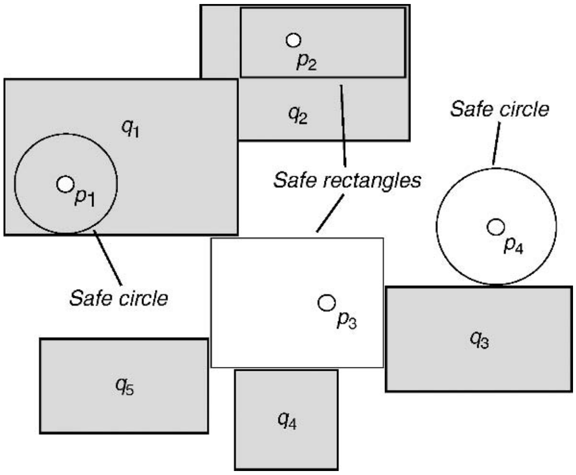
Historical Background

The first algorithms in the spatial database literature process snapshot (i.e., one-time) queries over static objects. They assume disk-resident data and utilize an index (e.g., an *R-tree*) to restrict the search space and reduce the I/O cost. Subsequent research considered spatial queries in client-server architectures. The general idea is to provide the user with extra information (along with the result at query-time) in order to reduce the number of subsequent queries as he/she moves (see entry *Nearest Neighbor Query*). These methods assume that the data objects are either static or moving linearly with known velocities. Due to the wide availability of positioning devices and the need for improved location-based services, the research focus has recently shifted to continuous spatial queries. In contrast with earlier assumed contexts, in this setting (i) there are multiple queries being evaluated simultaneously, (ii) the query results are continuously updated, and (iii) both the query points and the data objects move unpredictably.

Foundations

The first spatial monitoring method is called *Q-index* [13] and processes static range queries. Based on the observation that maintaining an index over frequently moving objects is very costly, *Q-index* indexes the queries instead of the objects. In particular, the monitored ranges are organized by an *R-tree*, and moving objects probe this tree to find the queries that they influence. Additionally, *Q-index* introduces the concept of *safe regions* to reduce the number of location updates. Specifically, each object p is assigned a circular or rectangular region, such that p needs to issue an update only if it exits this area (because, otherwise, it does not influence the result of any query). Figure 1 shows an example, where the current result of query q_1 contains object p_1, that of q_2 contains p_2, and the results of q_3, q_4, and q_5 are empty. The safe regions for p_1 and p_4 are circular, while for p_2 and p_3 they are rectangular. Note that no query result can change unless some objects fall outside their assigned safe regions. Kalashnikov et al. [4] show that a grid implementation of *Q-index* is more efficient (than *R-trees*) for main memory evaluation.

Monitoring Query Management (MQM) [1] and *MobiEyes* [2] also monitor range queries. They further exploit the computational capabilities of the objects to reduce the number of updates and the processing load of the server. In both systems, the objects store locally the queries in their vicinity and issue updates to the server only when they cross the boundary of any of these queries. To save their limited computational capabilities, the objects store and monitor only the queries they may affect when they move. MQM and *MobiEyes* employ different strategies to identify these queries. The former applies only to static queries. The latter can also handle moving ones, making however the assumption that they move linearly with fixed velocity.



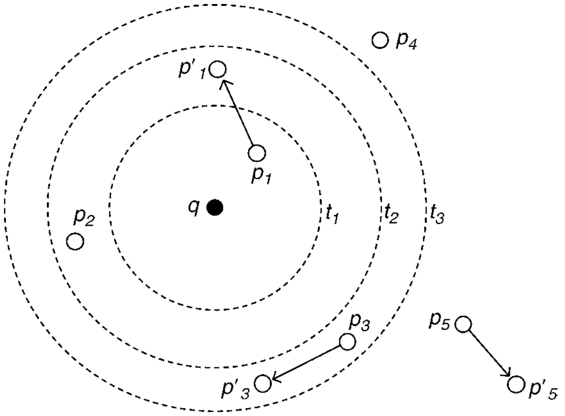
Continuous Monitoring of Spatial Queries. Figure 1. Circular and rectangular safe regions.

Mokbel et al. [7] present *Scalable INcremental hash-based Algorithm* (SINA), a system that monitors both static and moving ranges. In contrast with the aforementioned methods, in SINA the objects do not perform any local processing. Instead, they simply report their locations whenever they move, and the objective is to minimize the processing cost at the server. SINA is based on *shared execution* and *incremental evaluation*. Shared execution is achieved by implementing query evaluation as a spatial join between the objects and the queries. Incremental evaluation implies that the server computes only updates (i.e., object inclusions/exclusions) over the previously reported answers, as opposed to re-evaluating the queries from scratch.

The above algorithms focus on ranges, and their extension to NN queries is either impossible or nontrivial. The systems described in the following target NN monitoring. Hu et al. [3] extend the safe region technique to NN queries; they describe a method that computes and maintains rectangular safe regions subject to the current query locations and *k*NN results. Mouratidis et al. [11] propose *Threshold-Based algorithm* (TB), also aiming at communication cost reduction. To suppress unnecessary location updates, in TB the objects monitor their distance from the queries (instead of safe regions). Consider the example in Fig. 2, and assume that q is a continuous 3-NN query (i.e., $k = 3$). The initial result contains p_1, p_2, p_3. TB computes three thresholds (t_1, t_2, t_3) which define a range for each object. If every object's distance from q lies within its respective range, the result of the query is guaranteed to remain unchanged. Each threshold is set in the middle of the distances of two consecutive objects from the query. The distance range for p_1 is $[0, t_1)$, for p_2 is $[t_1, t_2)$, for p_3 is $[t_2, t_3)$, and for p_4, p_5 is $[t_3, \infty)$. Every object is aware of its distance range, and when there is a boundary violation, it informs the server about this event. For instance, assume that p_1, p_3, and p_5 move to positions p'_1, p'_3 and p'_5, respectively. Objects p_3 and p_5 compute their new distances from q, and avoid sending an update since they still lie in their permissible ranges. Object p_1, on the other hand, violates its threshold and updates its position to the server. Since the order between the first two NNs may have changed, the server requests for the current location of p_2, and updates accordingly the result and threshold t_1. In general, TB processes all updates issued since the last result maintenance, and (if necessary) it decides which additional object positions to request for, updates the k NNs of q, and sends new thresholds to the involved objects.

All the following methods aim at minimizing the processing time. Koudas et al. [6] describe *aDaptive Indexing on Streams by space-filling Curves* (DISC), a technique for *e*-approximate *k*NN queries over streams of multi-dimensional points. The returned (*e*-approximate) *k*th NN lies at most *e* distance units farther from q than the actual *k*th NN of q. DISC partitions the space with a regular grid of granularity such that the maximum distance between any pair of points in a cell is at most *e*. To avoid keeping all arriving data in the system, for each cell c it maintains only K points and discards the rest. It is proven that an exact *k*NN search in the retained points corresponds to a valid *ek*NN answer over the original dataset provided that $k \leq K$. DISC indexes the data points with a *B-tree* that uses a *space filling curve* mechanism to facilitate fast updates and query processing. The authors show how to adjust the index to: (i) use the minimum amount of memory in order to guarantee a given error bound *e*, or (ii) achieve the best possible accuracy, given a fixed amount of memory. DISC can process both snapshot and continuous *ek*NN queries.

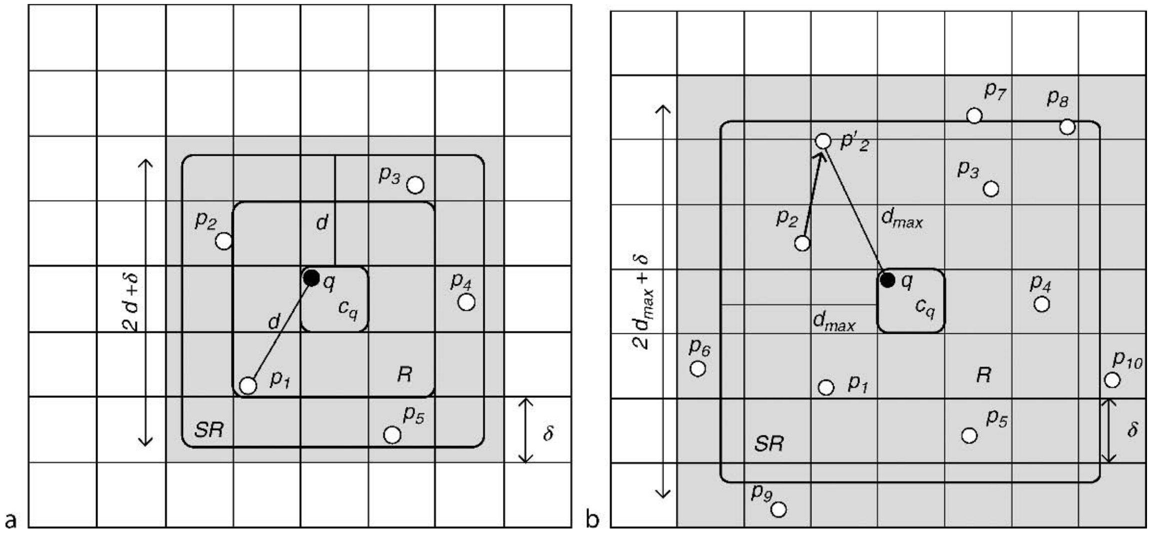
Yu et al. [17] propose a method, hereafter referred to as YPK-CNN, for continuous monitoring of exact *k*NN queries. Objects are stored in main memory and indexed with a regular grid of cells with size $\delta \times \delta$. YPK-CNN does not process updates as they arrive, but directly applies them to the grid. Each NN query installed in the system is re-evaluated every T time units. When a query q is evaluated for the first time, a two-step NN search technique retrieves its result. The first step visits the cells in an iteratively enlarged square R around the cell c_q of q until k objects are found. Figure 3a shows an example of a single NN query where the first candidate NN is p_1 with distance d from q; p_1 is not necessarily the actual NN since there may be objects (e.g., p_2) in cells outside R with distance smaller than d. To retrieve such objects, the second step searches in the cells intersecting the square SR centered at c_q with side length $2{\cdot}d + \delta$, and determines the actual kNN set of q therein. In Fig. 3a, YPK-CNN processes p_1 up to p_5 and returns p_2 as the actual NN. The accessed cells appear shaded.



Continuous Monitoring of Spatial Queries. Figure 2. TB example (k = 3).

When re-evaluating an existing query q, YPK-CNN makes use of its previous result in order to restrict the search space. In particular, it computes the maximum distance d_{max} among the current locations of the previous NNs (i.e., d_{max} is the distance of the previous neighbor that currently lies furthest from q). The new SR is a square centered at c_q with side length $2{\cdot}d_{max} + \delta$. In Fig. 3b, assume that the current NN p_2 of q moves to location p_2'. Then, the rectangle defined by $d_{max} = dist(p_2', q)$ is guaranteed to contain at least one object (i.e., p_2). YPK-CNN collects all objects (p_1 up to p_{10}) in the cells intersecting SR and identifies p_1 as the new NN. Finally, when a query q changes location, it is handled as a new one (i.e., its NN set is computed from scratch).

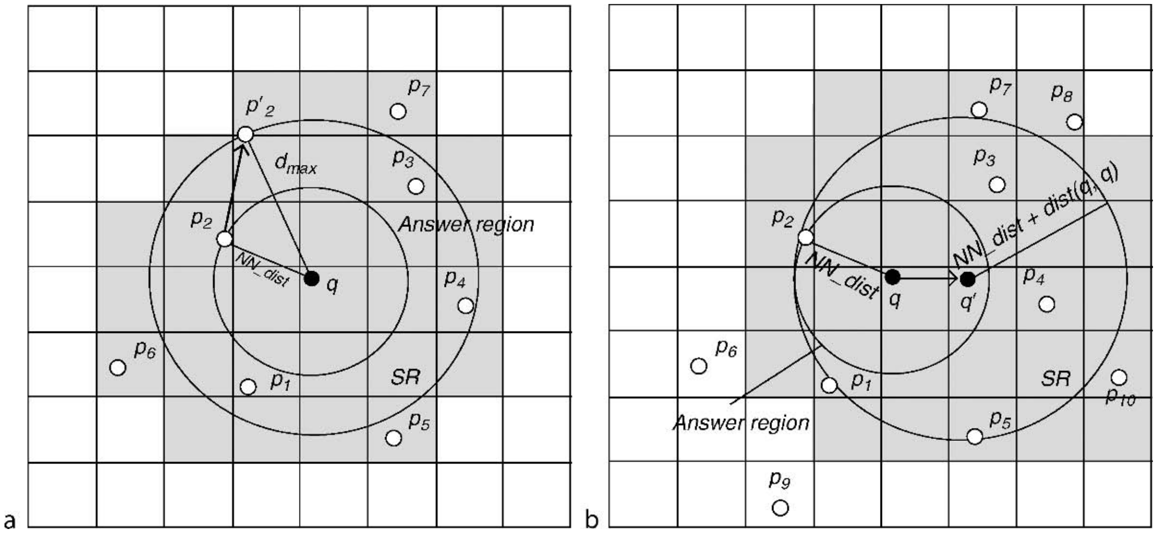
Xiong et al. [16] propose *Shared Execution Algorithm for Continuous NN queries* (SEA-CNN). SEA-CNN focuses exclusively on monitoring the NN changes, without including a module for the first-time evaluation of an arriving query q (i.e., it assumes that the initial result is available). Objects are stored in secondary memory, indexed with a regular grid. The *answer region* of a query q is defined as the circle with center q and radius NN_dist (where NN_dist is the distance of the current kth NN). Book-keeping information is stored in the cells that intersect the answer region of q to indicate this fact. When updates arrive at the system, depending on which cells they



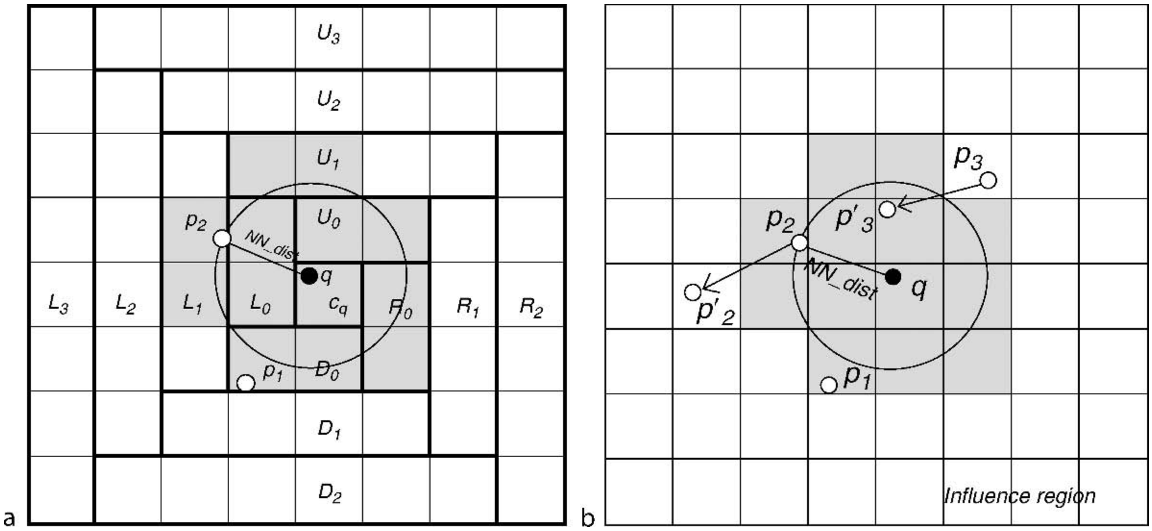
Continuous Monitoring of Spatial Queries. Figure 3. YPK-CNN examples.

affect and whether these cells intersect the answer region of the query, SEA-CNN determines a circular search region *SR* around *q*, and computes the new *k*NN set of *q* therein. To determine the radius *r* of *SR*, the algorithm distinguishes the following cases: (i) If some of the current NNs move within the answer region or some outer objects enter the answer region, SEA-CNN sets *r* = *NN_dist* and processes all objects falling in the answer region in order to retrieve the new NN set. (ii) If any of the current NNs moves out of the answer region, processing is similar to YPK-CNN; i.e., $r = d_{max}$ (where d_{max} is the distance of the previous NN that currently lies furthest from *q*), and the NN set is computed among the objects inside *SR*. Assume that in Fig. 4a the current NN p_2 issues an update reporting its new location p'_2. SEA-CNN sets $r = d_{max} = dist(p'_2, q)$, determines the cells intersecting *SR* (these cells appear shaded), collects the corresponding objects (p_1 up to p_7), and retrieves p_1 as the new NN. (iii) Finally, if the query *q* moves to a new location *q*′, then SEA-CNN sets *r* = *NN_dist* + *dist* (*q*, *q*′), and computes the new *k*NN set of *q* by processing all the objects that lie in the circle centered at *q*′ with radius *r*. For instance, in Fig. 4b the algorithm considers the objects falling in the shaded cells (i.e., objects from p_1 up to p_{10} except for p_6 and p_9) in order to retrieve the new NN (p_4).

Mouratidis et al. [9] propose another NN monitoring method, termed *Conceptual Partitioning Monitoring* (CPM). CPM assumes the same system architecture and uses similar indexing and book-keeping structures as YPK-CNN and SEA-CNN. When a query *q* arrives at the system, the server computes its initial result by organizing the cells into conceptual rectangles based on their proximity to *q*. Each rectangle *rect* is defined by a *direction* and a *level number*. The direction is U, D, L, or R (for up, down, left and right), and the level number indicates how many rectangles are between *rect* and *q*. Figure 5a illustrates the conceptual space partitioning around the cell c_q of *q*. If *mindist*(*c*,*q*) is the minimum possible distance between any object in cell *c* and *q*, the NN search considers the cells in ascending *mindist*(*c*, *q*) order. In particular, CPM initializes an empty heap *H* and inserts (i) the cell of *q* with key equal to 0, and (ii) the level zero rectangles for each direction *DIR* with key *mindist*(DIR_0, *q*). Then, it starts de-heaping entries iteratively. If the de-heaped entry is a cell, it examines the objects inside and updates accordingly the NN set (i.e., the list of the *k* closest objects found so far). If the de-heaped entry is a rectangle DIR_{lvl}, it inserts into *H* (i) each cell $c \in DIR_{lvl}$ with key *mindist*(*c*, *q*) and (ii) the next level rectangle DIR_{lvl+1} with key *mindist*(DIR_{lvl+1}, *q*). The algorithm terminates when the next entry in *H* (corresponding either to a cell or a rectangle) has key greater than the distance *NN_dist* of the *k*th NN found. It can be easily verified that the server processes only the cells that intersect the circle with center at *q* and radius equal to *NN_dist*. This is the minimal set of cells to visit in order to guarantee correctness. In Fig. 5a, the search processes the shaded cells and returns p_2 as the result.



Continuous Monitoring of Spatial Queries. Figure 4. SEA-CNN update handling examples.



Continuous Monitoring of Spatial Queries. Figure 5. CPM examples.

The encountered cells constitute the *influence region* of *q*, and only updates therein can affect the current result. When updates arrive for these cells, CPM monitors how many objects enter or leave the circle centered at *q* with radius *NN_dist*. If the *outgoing* objects are more than the *incoming* ones, the result is computed from scratch. Otherwise, the new NN set of *q* can be inferred by the previous result and the update information, without accessing the grid at all. Consider the example of Fig. 5b, where p_2 and p_3 move to positions p_2' and p_3', respectively. Object p_3 moves closer to *q* than the previous *NN_dist* and, therefore, CPM replaces the outgoing NN p_2 with the incoming p_3. The experimental evaluation in [11] shows that CPM is significantly faster than YPK-CNN and SEA-CNN.

Key Applications

Location-Based Services

The increasing trend of embedding positioning systems (e.g., GPS) in mobile phones and PDAs has given rise to a growing number of location-based services. Many of these services involve monitoring spatial relationships among mobile objects, facilities, landmarks, etc. Examples include location-aware advertising, enhanced 911 services, and mixed-reality games.

Traffic Monitoring

Continuous spatial queries find application in traffic monitoring and control systems, such as on-the-fly driver navigation, efficient congestion detection and avoidance, as well as dynamic traffic light scheduling and toll fee adjustment.

Security Systems

Intrusion detection and other security systems rely on monitoring moving objects (pedestrians, vehicles, etc.) around particular areas of interest or important people.

Future Directions

Future research directions include other types of spatial queries (e.g., *reverse nearest neighbor* monitoring [15,5]), different settings (e.g., NN monitoring over sliding windows [10]), and alternative distance metrics (e.g., NN monitoring in *road networks* [12]). Similar techniques and geometric concepts to the ones presented above also apply to problems of a non-spatial nature, such as continuous skyline [14] and top-*k* queries [8,18].

Experimental Results

The methods described above are experimentally evaluated and compared with alternative algorithms in the corresponding reference.

Cross-references

- ► B+-Tree
- ► Nearest Neighbor Query
- ► R-tree (and Family)
- ► Reverse Nearest Neighbor Query

- ▶ Road Networks
- ▶ Space-Filling Curves for Query Processing

Recommended Reading

1. Cai Y., Hua K., and Cao G. Processing range-monitoring queries on heterogeneous mobile objects. In Proc. 5th IEEE Int. Conf. on Mobile Data Management, 2004, pp. 27–38.
2. Gedik B. and Liu L. MobiEyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 67–87.
3. Hu H., Xu J., and Lee D. A generic framework for monitoring continuous spatial queries over moving objects. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 479–490.
4. Kalashnikov D., Prabhakar S., and Hambrusch S. Main memory evaluation of monitoring queries over moving objects. Distrib. Parallel Databases, 15(2):117–135, 2004.
5. Kang J., Mokbel M., Shekhar S., Xia T., and Zhang D. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 806–815.
6. Koudas N., Ooi B., Tan K., and Zhang R. Approximate NN queries on streams with guaranteed error/performance bounds. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 804–815.
7. Mokbel M., Xiong X., and Aref W. SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 623–634.
8. Mouratidis K., Bakiras S., Papadias D. Continuous monitoring of top-*k* queries over sliding windows. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 635–646.
9. Mouratidis K., Hadjieleftheriou M., and Papadias D. Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 634–645.
10. Mouratidis K. and Papadias D. Continuous nearest neighbor queries over sliding windows. IEEE Trans. Knowledge and Data Eng., 19(6):789–803, 2007.
11. Mouratidis K., Papadias D., Bakiras S., and Tao Y. A threshold-based algorithm for continuous monitoring of *k* nearest neighbors. IEEE Trans. Knowledge and Data Eng., 17(11):1451–1464, 2005.
12. Mouratidis K., Yiu M., Papadias D., and Mamoulis N. Continuous nearest neighbor monitoring in road networks. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 43–54.
13. Prabhakar S., Xia Y., Kalashnikov D., Aref W., and Hambrusch S. Query indexing and velocity constrained indexing: scalable techniques for continuous queries on moving objects. IEEE Trans. Comput., 51(10):1124–1140, 2002.
14. Tao Y. and Papadias D. Maintaining sliding window skylines on data Streams. IEEE Trans. Knowledge and Data Eng., 18(3): 377–391, 2006.
15. Xia T. and Zhang D. Continuous reverse nearest neighbor monitoring. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
16. Xiong X., Mokbel M., and Aref W. SEA-CNN: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 643–654.
17. Yu X., Pu K., and Koudas N. Monitoring k-nearest neighbor queries over moving objects. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 631–642.
18. Zhang D., Du Y., and Hu L. On monitoring the top-k unsafe places, In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 337–345.

Continuous Multimedia Data Retrieval

Jeffrey Xu Yu
Chinese University of Hong Kong, Hong Kong, China

Definition

Continuous multimedia is widely used in many applications nowadays. Continuous multimedia objects, such as audio and video streams, being stored on disks with different requirements of bandwidths, are required to be retrieved continuously without interruption. The response time is an important measurement in supporting continuous multimedia streams. Several strategies are proposed in order to satisfy the requirements of all users in a multi-user environment where multiple users are trying to retrieve different continuous multimedia streams together.

Historical Background

Several multimedia data retrieval techniques are proposed to support the real-time display of continuous multimedia objects. There are three categories [6]. The first category is to sacrifice the quality of the data in order to guarantee the required bandwidth of multimedia objects. The existing techniques either use lossy compression techniques (such as predictive [15], frequency oriented [11], and importance oriented [10]), or use a low resolution device. The second category is to use the placement techniques to satisfy the continuous requirement by arranging the data to appropriate disk locations. In other words, it is to organize multimedia data across the surface of a disk drive to maximize its bandwidth when it is retrieved [4,5,16,22,20]. The third category is to increase the bandwidth of storage device by using parallelism. The basic idea is to employ the aggregate bandwidth of several disk drives by putting an object across multiple disks, for example, a Redundant Arrays of Inexpensive Disk (RAID) [17]. The existing works [9,19] focus on this direction.

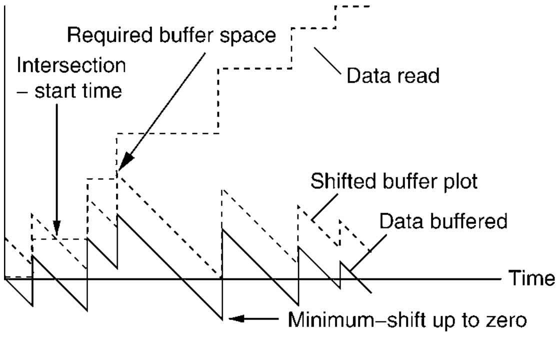
Foundations

This section focuses on the second and third categories, and discusses multimedia data retrieval regarding single/multiple stream(s) and single/multiple disk(s).

Retrieval of a Single Stream on a Single Disk

For the retrieval of a single multimedia stream on a single disk, the stream data is read into a first-in-first-out queue (FIFO) continuously first, and then is sent to the display devices, possibly via a network, at the appropriate rate. In order to satisfy the real-time requirements – to display multimedia data continuously on a display, it is required to keep the FIFO non empty. In other words, there is some multimedia data to be displayed in the FIFO in the duration of the playback. As pointed in [6], pre-fetching all the data into the FIFO before playback is not a feasible solution because the size of the stream can be very large.

Suppose that a read request of a large multimedia data is issued. The starting time and the minimum buffer space, to display the retrieved multimedia data continuously, are determined as follows, under the following conditions: (i) the timing of data retrieval is known in advance, (ii) both the transfer rate and consumption rate are constant, and (iii) the transfer rate of the storage device is at least as great as the consumption rate. Consider Fig. 1. First, the amount of data, that needs to be consumed by a display, is illustrated as the dotted line marked `data read`. The vertical line segments show the amount of data that needs to be consumed in order to continuously display, and the horizontal line shows the time periods such amount of data is consumed on a display. Second, the solid zigzag line, marked `data buffered`, shows the data to be accessed in the data buffers. The vertical line segments show the data to be read into the buffers followed by the line segments that show data is consumed in the buffer during a certain time interval. Here, in the solid zigzag line, there is a minimum point (marked `minimum-shift up to zero`), which is a possible negative value and is denoted as $z(<0)$. Third, the dotted zigzag line (marked `shifted buffer plot`) is the line by shifting the entire solid zigzag line up by the amount of $|z|$ where $z < 0$. Finally, the starting time to display is determined as the point at which the shifted-up dotted zigzag line (`shifted buffer plot`) and the dotted line (`data read`) intersect, which is indicated as `intersection - start time` in Fig. 1. Also, the minimum buffer size is the maximum value of in the line of `shifted buffer plot`, which is indicated as `required buffer space` in Fig. 1. Details are discussed in [7].

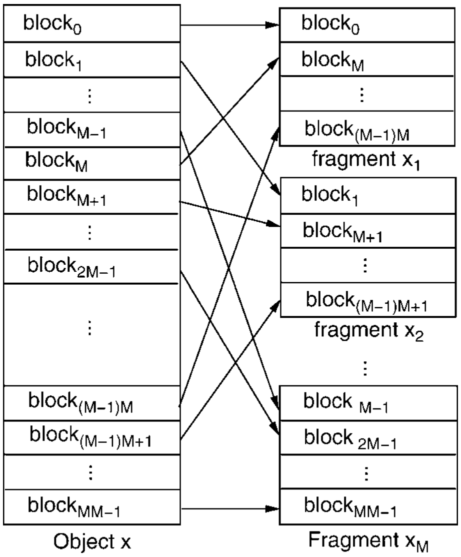


Continuous Multimedia Data Retrieval. Figure 1. Finding minimum buffer space and start time (Fig. 2 in [6]).

Retrieval of a Single Stream on Multiple Disks

The multimedia data retrieval using multiple disks is a technique to retrieve a data stream continuously at the required bandwidth. The main idea behind is to de-cluster the data stream into several fragments [14,2], and distribute these fragments across multiple processors (and disks). By combining the I/O bandwidths of several disks, a system can provide the required retrieval rate to display a continuous multimedia stream in real-time. Assume that the required retrieval rate is B and the bandwidth of each disk is B_D. The degree of de-clustering can be calculated as $M = \lceil \frac{B}{B_D} \rceil$, which implies the number of disks that is needed to satisfy the required retrieval rate.

When the degree of de-clustering is determined, the fragments can be formed using a round-robin partitioning strategy as illustrated in Fig. 2, where an object x is partitioned into M fragments stored on M disks. The round-robin partitioning is conducted as follows. First, the object x is divided in N blocks (disk pages) depending on the disk-page size allowed on disks. In Fig. 2, the number of blocks is $N = M \cdot M$. The first block$_0$ is assigned to first fragment indicated as x_1 in Fig. 2, and the second block$_1$ is assigned to the second fragment indicated as x_2. The first M blocks from block$_0$ to block$_{M-1}$ are assigned to the M fragments one by one. In next run, the next set of blocks,



Continuous Multimedia Data Retrieval. Figure 2. Round-robin partitioning of object *x* (Fig. 5 in [9]).

from block$_M$ to block$_{2M-1}$ will be assigned to the M fragments in the similar fashion. The process repeats until all data blocks are assigned to the fragments in a round-robin fashion.

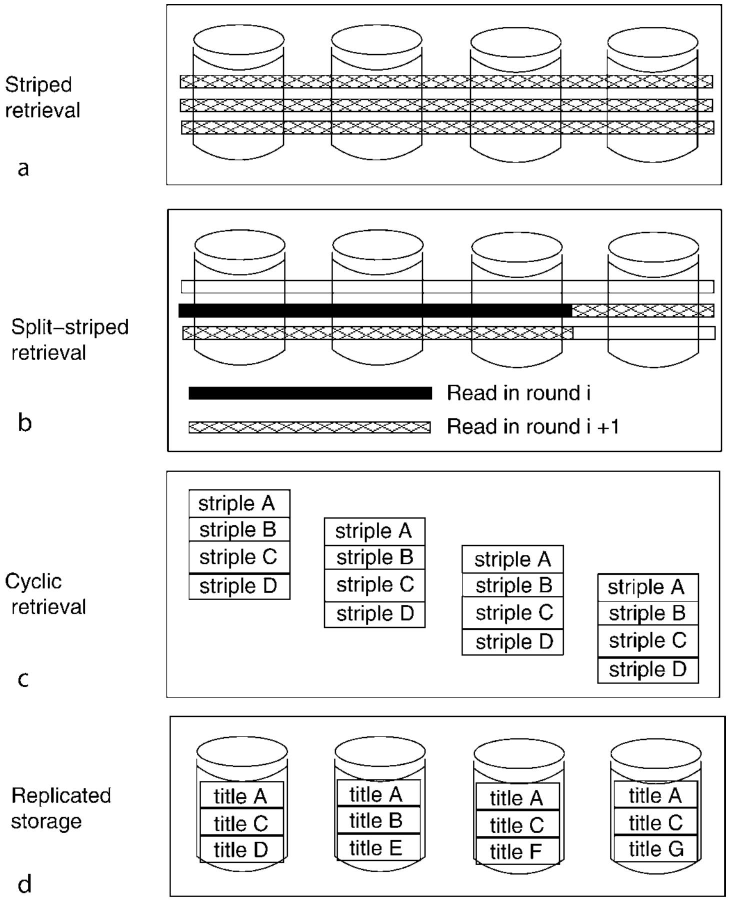
Retrieval of Multiple Streams on a Single Disk

In a multi-user environment, several users may retrieve data streams simultaneously. Therefore, there are multiple data streams requested on a single disk. The data streams are retrieved in rounds, and each stream is allowed a disk access or a fixed number of disk accesses at one time. All data retrieval requests need to be served in turn. Existing solutions include SCAN, round-robin, EDF, and Sorting-Set algorithms. The round-robin algorithm retrieves data for each data retrieval request, in turn, in a predetermined order. The SCAN algorithm moves the disk head back and forth, and retrieves the requested blocks when the disk head passes over the requested blocks [18]. The EDF (earliest-deadline-first) algorithm serves the request with the earliest deadline first, where a deadline is given to a data stream [13]. The sorting-set algorithm is designed to exploit the trade-off between the number of rounds between successive reads for a data stream and the length of the round [8,21], by assigning each data stream to a sorting set. Fixed time slots are allocated to a sorting set in a round during which its requests are possibly processed.

Retrieval of Multiple Streams on Multiple Disks

In order to support multiple stream retrieval, making use of parallel disks is an effective method, where a data stream is striped across the parallel disks. There are several approaches to retrieve data streams when they are stored on parallel disks (Fig. 3). It is important to note that the main issue here is to increase the number of data streams to be retrieved simultaneously. It is not to speed up retrieval for an individual data stream using multiple disks. Consider the striped retrieval as shown in Fig. 3a, where a data stream is striped across m parallel disks. Suppose that each disk has r_c bandwidth, m parallel disks can be together used to increase the bandwidth up to $m \cdot r_c$. However, the issue is the system capacity in terms of the number of data streams it can serve, for example, from n data streams to $m \cdot n$ data streams using m parallel disks. Suppose that each data stream will be served in turn. When it increases the number of data streams from n to $m \cdot n$, in the striped retrieval, the round length (or in other words consecutive reads for a single data stream) increases proportionally from n to $m \cdot n$. It implies that, in order to satisfy the required retrieval rate, it needs to use a larger buffer, which also implies a larger startup delay. An improvement over striped retrieval is to use split-stripe retrieval which allows partial stripes to be used (Fig. 3b), in order to reduce the buffer size required in the striped retrieval. But, it has its limit to significantly reduce startup delay and buffer space.

Observe the data transfer patterns in the striped retrieval and the split-striped retrieval, which show busty patterns for data to be read into the buffer. For instance, consider Fig. 3a, an entire strip for a single data stream will be read in and be consumed in a period which is related to the round length. It requests larger buffer sizes, because it needs to keep the data to be displayed continuously until the next read, in particular when the number of streams increases from n to $m \cdot n$. Instead, an approach is proposed to read small portion of data frequently, in order to reduce the buffer space required. The approach is called cyclic retrieval. As shown in Fig. 3c, the cyclic retrieval tries to read multiple streams rather than one stream at one time. Rather than retrieving an entire stripe at once, the cyclic retrieval retrieves each striping unit of a stripe consecutively [1,3]. Using this approach, the buffer space is significantly reduced. But the reduction comes with cost. The buffer space reduction is achieved at the expense of cuing (*a stream is said to*



Continuous Multimedia Data Retrieval. Figure 3. Retrieval of multiple streams on multiple disks [6].

be cued if it is paused and playback may be initiated instantaneously) and clock skew tolerance.

As an alternative to striped (split-striped) or cyclic retrieval, it can deal with each disk independently rather than treating them as parallel disks. Here, each disk stores a number of titles (data streams). When there is a multimedia data retrieval request, a disk that contains the data stream will respond. The data streams that are frequently requested may be kept in multiple disks using replication. The number of replications can be determined based on the retrieval frequency of data streams [12], as shown in Fig. 3d. Based on the replicated retrieval, both the startup delay time and buffer space can be reduced significantly. It is shown that it is easy to scale when the number of data streams increase at the expense of more disk space required. [9] discusses data replication techniques.

A comparison among striped-retrieval, cyclic retrieval, and replicated retrieval in supporting *n* streams is shown in Table 1.

Continuous Multimedia Data Retrieval. Table 1. A comparison of multi-disk retrieval strategies supporting *n* streams (Table 1 in [6])

	Striped	Cyclic	Replicated
Instant restart	yes	no	yes
Clock skew tolerance	yes	no	yes
Easy scaling	no	no	yes
Capacity	per-system	per-system	per-title
Startup delay	O(n)	O(n)	O(1)
Buffer space	O(n^2)	O(n)	O(n)

Key Applications

Continuous multimedia data retrieval is used in many real-time continuous multimedia streams such as audio and video through the network. Especially in a multi-user environment, the continuous multimedia data retrieval techniques are used to support simultaneous display of several multimedia objects in real-time.

Cross-references

- ▶ Buffer Management
- ▶ Buffer Manager
- ▶ Multimedia Data Buffering
- ▶ Multimedia Data Storage
- ▶ Multimedia Resource Scheduling
- ▶ Scheduling Strategies for Data Stream Processing
- ▶ Storage Management
- ▶ Storage Manager

Recommended Reading

1. Berson S., Ghandeharizadeh S., Muntz R., and Ju X. Staggered striping in multimedia information systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 79–90.
2. Carey M.J. and Livny M. Parallelism and concurrency control performance in distributed database machines. ACM SIGMOD Rec., 18(2):122–133, 1989.
3. Chen M.S., Kandlur D.D., and Yu P.S. Storage and retrieval methods to support fully interactive playout in a disk-array-based video server. Multimedia Syst., 3(3):126–135, 1995.
4. Christodoulakis S. and Ford D.A. Performance analysis and fundamental performance tradeoffs for CLV optical disks. ACM SIGMOD Rec., 17(3):286–294, 1988.
5. Ford D.A. and Christodoulakis S. Optimizing random retrievals from CLV format optical disks. In Proc. 17th Int. Conf. on Very Large Data Bases, 1991, pp. 413–422.
6. Gemmell D.J. Multimedia information storage and management, chap. 1. Disk Scheduling for Continuous Media. Kluwer, Norwell, MA, USA, 1996.
7. Gemmell J. and Christodoulakis S. Principles of delay-sensitive multimedia data storage retrieval. ACM Trans. Inf. Syst., 10(1):51–90, 1992.
8. Gemmell D.J. and Han J. Multimedia network file servers: multichannel delay-sensitive data retrieval. Multimedia Syst., 1(6):240–252, 1994.
9. Ghandeharizadeh S. and Ramos L. Continuous retrieval of multimedia data using parallelism. IEEE Trans. on Knowl. and Data Eng., 5(4):658–669, 1993.
10. Green J.L. The evolution of DVI system software. Commun. ACM, 35(1):52–67, 1992.
11. Lippman A. and Butera W. Coding image sequences for interactive retrieval. Commun. ACM, 32(7):852–860, 1989.
12. Little T.D.C. and Venkatesh D. Popularity-based assignment of movies to storage devices in a video-on-demand system. Multimedia Syst., 2(6):280–287, 1995.
13. Liu C.L. and Layland J.W. Scheduling algorithms for multiprogramming in a hard real-time environment. In Tutorial: Hard Real-Time Systems. IEEE Computer Society, Los Alamitos, CA, USA, 1989, pp. 174–189.
14. Livny M., Khoshafian S., and Boral H. Multi-disk management algorithms. SIGMETRICS Perform. Eval. Rev., 15(1):69–77, 1987.
15. Luther A.C. Digital video in the PC environment, (2nd edn.). McGraw-Hill, New York, NY, USA, 1991.
16. McKusick M.K., Joy W.N., Leffler S.J., and Fabry R.S. A fast file system for UNIX. Comput. Syst., 2(3):181–197, 1984.
17. Patterson D.A., Gibson G.A., and Katz R.H. A case for redundant arrays of inexpensive disks (RAID). In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988, pp. 109–116.
18. Teorey T.J. and Pinkerton T.B. A comparative analysis of disk scheduling policies. In Proc. 3rd ACM Symp. on Operating System Principles, 1971, pp. 114.
19. Tsai W.J. and Lee S.Y. Storage design and retrieval of continuous multimedia data using multi-disks. In Proc. 1994 Int. Conf. on Parallel and Distributed Systems, 1994, pp. 148–153.
20. Wong C.K. Minimizing expected head movement in one-dimensional and two-dimensional mass storage systems. ACM Comput. Surv., 12(2):167–178, 1980.
21. Yu P.S., Chen M.S., and Kandlur D.D. Grouped sweeping scheduling for DASD-based multimedia storage management. Multimedia Syst., 1(3):99–109, 1993.
22. Yue P.C. and Wong C.K. On the optimality of the probability ranking scheme in storage applications. J. ACM, 20(4):624–633, 1973.

Continuous Queries in Sensor Networks

Yong Yao, Johannes Gehrke
Cornell University, Ithaca, NY, USA

Synonyms

Long running queries

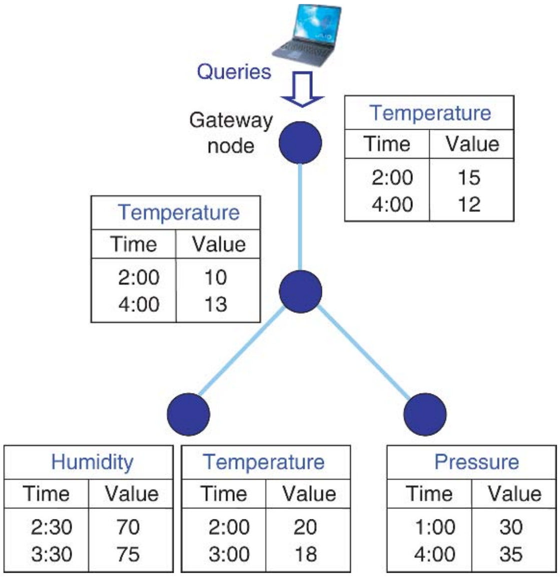
Definition

A powerful programming paradigm for data acquisition and dissemination in sensor networks is a declarative query interface. With a declarative query interface, the sensor network is programmed for long term monitoring and event detection applications through continuous queries, which specify what data to retrieve at what time or under what conditions. Unlike snapshot queries which execute only once, continuous queries are evaluated periodically until the queries expire. Continuous queries are expressed in a high-level language, and are compiled and installed on target sensor nodes, controlling when, where, and what data is sampled, possibly filtering out unqualified data through local predicates. Continuous queries can have a variety of optimization goals, from improving result quality and response time to reducing energy consumption and prolonging network lifetime.

Historical Background

In recent years sensor networks have been deployed successfully for a wide range of applications from environmental sensing to process monitoring. A database approach to programming sensor networks has gained much importance: Clients program the network through queries without knowing how the results are generated, processed, and returned to the client. Sophisticated catalog management, query optimization, and query processing techniques abstract the client from the physical details of contacting the relevant sensors, processing the sensor data, and sending the results to the client. The concept of a sensor network as a database was first introduced in [3]. A number of research projects, including TinyDB [9] and Cougar [14] have implemented continuous queries as part of their database languages for sensor networks. In these systems time is divided into epochs of equal size, and continuous queries are evaluated once per epoch during their lifetime. Figure 1 shows this database view of sensor networks.

Two properties are significant to continuous query processing in sensor networks: energy conservation and fault-tolerance in case of failures of sensors, both topics that are not of importance in traditional database systems or data stream systems. Advanced query processing techniques have been proposed to enable energy-efficient query processing in the presence of frequent node and communication failures. For example, a lot of research has been dedicated to in-network query processing [6,9,14] to reduce the amount of data to be transmitted inside the network. Another approach is to permit approximate query processing [4,5], which produces approximate query answers within a predefined accuracy range, but consumes much less energy. Sensor data is correlated in time and space. Data compression in sensor networks and probabilistic data models [1,7,8] exploit data correlation and remove redundant data from intermediate results.



Temperature	
Time	Value
2:00 4:00	15 12

Temperature	
Time	Value
2:00 4:00	10 13

Humidity	
Time	Value
2:30 3:30	70 75

Temperature	
Time	Value
2:00 3:00	20 18

Pressure	
Time	Value
1:00 4:00	30 35

Continuous Queries in Sensor Networks. Figure 1. Database view of sensor networks.

Next generation sensor network may consist of media-rich and mobile sensor nodes, which result in new challenges arise for continuous query processing such as mobility and high data rates. ICEDB [15] describes a new framework for continuous query processing in sensor networks with intermittent network connectivity and large amount of data to transfer.

Foundations

Continuous queries are a natural approach for data fusion in sensor networks for long running applications as they provide a high-level interface that abstracts the user from the physical details of the network. The design and implementation of continuous queries needs to satisfy several requirements. First, it has to preserve the scarce resources such as energy and bandwidth in battery-powered sensor networks. Thus the simple approach of transmitting all relevant data back to a central node for query evaluation is prohibitive for sensor networks of non-trivial size, as communication using the wireless medium consumes a lot of energy. Since sensor nodes have the ability to perform local computation, communication can be traded for computation by moving computation from the clients into the sensor network, aggregating partial results or eliminating irrelevant data. Second, sensor network applications usually have different QoS requirements, from accuracy, energy consumption to delay. Therefore the continuous query model needs to be flexible enough to adopt various processing techniques in different scenarios.

Sensor Data Model

In the view of a sensor network as a database, each sensor node is modeled as a separate data source that generates records with several fields such as the sensor type, location of the sensor node, a time stamp, and the value of the reading. Records of the same sensor type from different nodes have the same schema, and these records collectively form a distributed table of sensor readings. Thus the sensor network can be considered as a large distributed database system consisting of several tables of different types of sensors.

Sensor readings are samples of physical signals whose values change continuously over time. For example, in environmental monitoring applications, sensor readings are generated every few seconds (or even faster). For some sensor types (such as PIR sensors that sense the presence of objects) their readings might change rapidly and thus may be outdated rather quickly, whereas for other sensors, their value changes only slowly over time as for temperature sensors that usually have a small derivative. Continuous queries recompute query results periodically and keep query results up-to-date. For applications that require only approximate results, the system can cache previous results and lower the query update rate to save energy.

Instead of querying raw sensor data, most applications are more interested in composite data which captures high-level events monitored by sensor networks. Such composite data is produced by complex signal processing algorithms given raw sensor measurements as inputs. Composite data usually has a compact structure and is easier to query.

Continuous Query Models

In TinyDB and Cougar, continuous queries are represented as a variant of SQL with a few extensions. A simple query template in Cougar is shown in the figure below. (TinyDB uses a very similar query structure.)

```
SELECT   {attribute, aggregate}
FROM     {Sensordata S}
WHERE    {predicate}
GROUP BY {attribute}
HAVING   {predicate}
DURATION time interval
EVERY    time span e
```

The template can be extended to support nested queries, where the basic query block shown below can appear within the WHERE or HAVING clause of another query block. The query template has an obvious semantics: the SELECT clause specifies attributes and aggregates from sensor records, the FROM clause specifies the distributed relation describing the sensor type, the WHERE clause filters sensor records by a predicate, the GROUP BY clause classifies sensor records into different partitions according to some attributes, and the HAVING clause eliminates groups by a predicate. Join queries between external tables and sensor readings are constructed by including the external tables and sensor readings in the FROM clause and join predicates in the WHERE clause.

Two new clauses introduced for continuous queries are DURATION and EVERY; The DURATION clause specifies the lifetime of the continuous query, and the EVERY or clause determines the rate of query answers. TinyDB has two related clauses: LIFETIME and SAMPLE INTERVAL, specifying the lifetime of the query and the sample interval, respectively. The LIFETIME clause will be discussed in more detail a few paragraphs later.

In event detection applications, sensor data is collected only when particular events happen. The above query template can be extended with a condition clause as a prerequisite to determine when to start or stop the main query. Event-based queries have the following structure in TinyDB:

```
ON EVENT   {event(arguments)}:
  {query body}
```

Another extension to the basic query template is lifetime-based queries, which have no explicit EVERY or SAMPLE INTERVAL clause; only the query lifetime is specified through a LIFETIME clause [9]. The system automatically adjusts the sensor sampling rate to the highest rate possible with the guarantee that the sensor network can process the query for the specified lifetime. Lifetime-based queries are more intuitive in some mission critical applications where user queries have to run for a given period of time, but it is hard to predict the optimal sampling rate in advance. Since the sampling rate is adjusted continuously according to the available power and the energy consumption rate in the sensor network, lifetime-based queries are more adaptive to unpredictable changes in sensor networks deployed in a harsh environment.

Common Types of Continuous Queries in Sensor Networks Select-All Queries

Recent sensor network deployments indicate that a very common type of continuous queries is a select-all query, which extracts all relevant data from the sensor network and stores the data in a central place for further processing and analysis. Although select-all queries are simple to express, efficient processing of select-all queries is a big challenge. Without optimization, the size of the transmitted data explodes quickly, and thus the power of the network would be drained in a short time, especially for those nodes acting as bridge to the outside world; this significantly decreases the lifetime of the sensor network.

One possible approach is to apply model-based data compression at intermediate sensor nodes [7]. For many types of signals, e.g., temperature and light, sensor readings are highly correlated in both time and space. Data compression in sensor networks can significantly reduce the communication overhead and increase the network lifetime. Data compression can also improve the signal quality by removing unwanted noise from the original signal. One possible form of compression is to construct and maintain a model of the sensor data in the network; the model is stored both on the server and on sensor nodes in the network. The model on the server can be used to predicate future values within a pre-defined accuracy range. Data communication happens to synchronize the data model on the server with real sensor measurements [7].

Aggregate Queries

Aggregate queries return aggregate values for each group of sensor nodes specified by the GROUP BY clause. Below is is an example query that computes the average concentration in a region every 10 seconds for the next hour:

```
SELECT    AVG(R.concentration)
FROM      ChemicalSensor R
WHERE     R.loc IN region
HAVING    AVG(R.concentration) > T
DURATION  (now,now+3600)
EVERY     10
```

Data aggregation in sensor networks is well-studied because it scales to sensor networks with even thousands of nodes. Query processing proceeds along a spanning tree of sensor nodes towards a gateway node. During query processing, partial aggregate results are transmitted from a node to its parent in the spanning tree. Once an intermediate node in the tree has received all data from nodes below it in a round, the node compute a partial aggregate of all received data and sends that output to the next node. This solution works for aggregate operators that are incrementally computable, such as avg, max, and moments of the data. The only caveat is that this in-network computation requires synchronization between sensor nodes along the communication path, since a node has to "wait" to receive results to be aggregated. In networks with high loss rates, broken links are hard to differentiate from long delays due to high loss rates, making synchronization a non-trivial problem [13].

Join Queries

In a wide range of event detection applications, sensor readings are compared to a large number of time and location varying predicates to determine whether a user-interesting event is detected [1]. The values of these predicates are stored in a table. Continuous queries with a *join* operator between sensor readings and the predicate table are suitable for such applications. Similar join queries can be used to detect defective sensor nodes whose readings are inaccurate by checking their readings against readings from neighboring sensors (again assuming spatial correlation between sensor readings). Suitable placement of the join operator in a sensor network has also been examined [2].

Key Applications

Habitat Monitoring

In the Great Duck Island experiment, a network of sensors was deployed to monitor the microclimate in and around nesting burrows used by birds, with the goal of developing a habitat monitoring kit that would enable researchers worldwide to engage in non-intrusive and non-disruptive monitoring of sensitive wildlife and habitats [10]. In a more recent experiment, a sensor network was deployed to densely record the complex spatial variations and the temporal dynamics of the microclimate around a 70-meter tall redwood tree [12].

The Intelligent Building

Sensor networks can be deployed in intelligent buildings for the collection and analysis of structural responses to ambient or forced excitation of the building's structure, for control of light and temperature to conserve energy, and for monitoring of the flow of people in critical areas. Continuous queries are used both for data collection and for event-based monitoring of sensitive areas and to enforce security policies.

Industrial Process Control

Idustrial manufacturing processes often have strict requirements on temperature, humidity, and other environmental parameters. Sensor networks can be deployed to monitor the production environment without expensive wires to be installed. Continuous join queries compare the state of the environment to a range of values specified in advance and send an alert when an exception is detected [1].

Cross-references

- ▶ Approximate Query Processing
- ▶ Data Acquisition and Dissemination in Sensor Networks
- ▶ Data Aggregation in Sensor networks
- ▶ Data Compression in Sensor Networks
- ▶ Data Fusion in Sensor Networks
- ▶ Database Languages for Sensor Networks
- ▶ Distributed Database Systems
- ▶ In-Network Query Processing
- ▶ Sensor Networks

Recommended Reading

1. Abadi D., Madden S., and Lindner W. REED: robust, efficient filtering and event detection in sensor networks. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 768–780.
2. Bonfils B. and Bonnet P. Adaptive and decentralized operator placement for in-network query processing. In Proc. 2nd Int. Workshop Int. Proc. in Sensor Networks, 2003, pp. 47–62.
3. Bonnet P., Gehrke J., and Seshadri P. Towards sensor database systems. In Proc. 2nd Int. Conf. on Mobile Data Management, 2001, pp. 3–14.
4. Chu D., Deshpande A., Hellerstein J., and Hong W. Approximate data collection in sensor networks using probabilistic models. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
5. Considine J., Li F., Kollios G., and Byers J. Approximate aggregation techniques for sensor databases. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 449–460.
6. Deligiannakis A., Kotidis Y., and Roussopoulos N. Hierarchical in-network data aggregation with quality guarantees. In Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 658–675.
7. Deshpande A., Guestrin C., Madden S., Hellerstein J., and Hong W. Model-driven data acquisition in sensor networks. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 588–599.
8. Kanagal B. and Deshpande A. Online filtering, smoothing and probabilistic modeling of streaming data. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 1160–1169.
9. Madden S., Franklin M., Hellerstein J., and Hong W. The design of an acquisitional query processor for sensor networks. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 491–502.
10. Mainwaring A., Polastre J., Szewczyk R., Culler D., and Anderson J. Wireless sensor networks for habitat monitoring. In Proc. 1st ACM Int. Workshop on Wireless Sensor Networks and Applications, 2002, pp. 88–97.
11. Stoianov I., Nachman L., Madden S., and Tokmouline T. PIPENET: a wireless sensor network for pipeline monitoring. In Proc. 6th Int. Symp. Inf. Proc. in Sensor Networks, 2007, pp. 264–273.
12. Tolle G., Polastre J., Szewczyk R., Culler D., Turner N., Tu K., Burgess S., Dawson T., Buonadonna P., Gay D., and Hong W. A macroscope in the redwoods. In Proc. 3rd Int. Conf. on Embedded Networked Sensor Systems, 2005.
13. Trigoni N., Yao Y., Demers A.J., Gehrke J., and Rajaraman R. Wave scheduling and routing in sensor networks. ACM Trans. Sensor Netw., 3(1):2, 2007.
14. Yao Y. and Gehrke J. Query processing in sensor networks. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
15. Zhang Y., Hull B., Balakrishnan H., and Madden S. ICEDB: intermittently connected continuous query processing. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 166–175.

Continuous Query

Shivnath Babu
Duke University, Durham, NC, USA

Synonyms

Standing query

Definition

A continuous query *Q* is a query that is issued once over a database *D*, and then logically runs continuously over the data in *D* until *Q* is terminated. *Q* lets users get new results from *D* without having to issue the same query repeatedly. Continuous queries are best understood in contrast to traditional SQL queries over *D* that run once to completion over the current data in *D*.

Key Points

Traditional database systems expect all data to be managed within some form of persistent data sets. For many recent applications, where the data is changing constantly (often exclusively through insertions of new elements), the concept of a continuous data stream is more appropriate than a data set. Several applications generate data streams naturally as opposed to data sets, e.g., financial tickers, performance measurements in network monitoring, and call detail records in telecommunications. Continuous queries are a natural interface for monitoring data streams. In network monitoring, e.g., continuous queries may be used to monitor whether all routers and links are functioning efficiently.

The Tapestry system [3] for filtering streams of email and bulletin-board messages was the first to make continuous queries a core component of a database system. Continuous queries in Tapestry were expressed using a subset of SQL. Barbara [2] later formalized continuous queries for a wide spectrum of environments. With the recent emergence of general-purpose systems for processing data streams, continuous queries have become the main interface that users and applications use to query data streams [1].

Materialized views and triggers in traditional database systems can be viewed as continuous queries. A materialized view *V* is a query that needs to be reevaluated or incrementally updated whenever the base data over which *V* is defined changes. Triggers implement event-condition-action rules that enable database systems to take appropriate actions when certain events occur.

Cross-references

▶ Database Trigger
▶ ECA-Rule
▶ Materialized Views
▶ Processing

Recommended Reading

1. Babu S. and Widom J. Continuous queries over data streams. ACM SIGMOD Rec., 30(3):109–120, 2001.
2. Barbara D. The characterization of continuous queries. Int. J. Coop. Inform. Syst., 8(4):295–323, 1999.
3. Terry D., Goldberg D., Nichols D., and Oki B. Continuous queries over append-only databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1992, pp. 321–330.

Continuous Query Languages

▶ Stream-oriented Query Languages and Operators

Continuous Query Processing Applications

▶ Streaming Applications

Continuous Query Scheduling

▶ Scheduling Strategies for Data Stream Processing

ConTract

Andreas Reuter[1,2]
[1]EML Research aGmbH Villa Bosch, Heidelberg, Germany
[2]Technical University Kaiserslautern, Kaiserslautern, Germany

Definition

A ConTract is an extended transaction model that employs transactional mechanisms in order to provide a run-time environment for the reliable execution of long-lived, workflow-like computations. The focus is on durable execution and on correctness guarantees with respect to the effects of such computations on shared data.

Key Points

The notion of a ConTract (concatenated transactions) combines the principles of workflow programing with the ideas related to long-lived transactions. The ConTract model is based on a two-tier programing approach. At the top level, each ConTract is a script describing a (long-lived) computation. The script describes the order of execution of so-called steps. A step is a predefined unit of execution (e.g., a service invocation) with no visible internal structure. A step can access shared data in a database, send messages, etc.

A ConTract, once it is started, will never be lost by the system, no matter which technical problems (short of a real disaster) will occur during execution. If completion is not possible, all computations performed by a ConTract will be revoked, so in a sense ConTracts have transactional behaviour in that they will either be run to completion, or the impossibility of completion will be reflected in the invocation of appropriate recovery measures.

The ConTract model draws on the idea of Sagas, where the notion of compensation is employed as a means for revoking the results of computations beyond the boundaries of ACID transactions. In a ConTract, by default each step is an ACID transaction. But it is possible to group multiple steps (not just linear sequences) into a transaction. Compensation steps must be supplied by the application explicitly.

The ideas of the ConTract model have selectively been implemented in some academic prototypes, but a full implementation has never been attempted. It has influenced many later versions of “long-lived transaction” schemes, and a number of its aspects can be found in commercial systems such as BizTalk.

Cross-references

- ▶ Extended Transaction Models
- ▶ Persistent Execution
- ▶ Sagas
- ▶ Workflow

Recommended Reading

1. Reuter A. and Waechter H. The ConTract model. In Readings in Database in Database Systems, (2nd edn.), M. Stonebraker, J. Hellerstein, (eds.). Morgan Kaufmann, Los Altos, CA, 1992, pp. 219–263.

ConTracts

▶ Flex Transactions

Contrast Pattern

▶ Emerging Patterns

Contrast Pattern Based Classification

▶ Emerging Pattern Based Classification

Control Data

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

Workflow control data; Workflow engine state data; Workflow enactment service state data

Definition

Data that is managed by the Workflow Management System and/or a Workflow Engine. Such data is internal to the workflow management system and is not normally accessible to applications.

Key Points

Workflow control data represents the dynamic state of the workflow system and its process instances.

Workflow control data examples include:

- State information about each workflow instance.
- State information about each activity instance (active or inactive).
- Information on recovery and restart points within each process, etc.

The workflow control data may be written to persistent storage periodically to facilitate restart and recovery of the system after failure. It may also be used to derive audit data.

Cross-references

- ▶ Activity
- ▶ Process Life Cycle
- ▶ Workflow Management and Workflow Management System
- ▶ Workflow Model

Control Flow Diagrams

▶ Activity Diagrams

Controlled Vocabularies

▶ Lightweight Ontologies

Controlling Overlap

▶ Processing Overlaps

Convertible Constraints

Carson Kai-Sang Leung
University of Manitoba, Winnipeg, MB, Canada

Definition

A constraint C is *convertible* if and only if C is convertible anti-monotone or convertible monotone.

A constraint C is *convertible anti-monotone* provided there is an order $\mathcal{R}$ on items such that when an ordered itemset S satisfies constraint C, so does any prefix of S. A constraint C is *convertible monotone* provided there is an order $\mathcal{R}'$ on items such that when an ordered itemset S' violates constraint C, so does any prefix of S'.

Key Points

Although some constraints are neither anti-monotone nor monotone in general, several of them can be converted into anti-monotone or monotone ones by properly ordering the items. These *convertible constraints* [1-3] possess the following nice properties. By arranging items according to some proper order $\mathcal{R}$, if an itemset S satisfies a *convertible anti-monotone constraint* C, then all prefixes of S also satisfy C. Similarly, by arranging items according to some proper order $\mathcal{R}'$, if an itemset S violates a *convertible monotone constraint* C', then any prefix of S also violates C'. Examples of convertible constraints include $avg(S.Price) \geq 50$, which expresses that the average price of all items in an itemset S is at least \$50. By arranging items in non-ascending order $\mathcal{R}$ of price, if the average price of items in an itemset S is at least \$50, then the average price of items in any prefix of S would not be lower than that of S (i.e., all prefixes of S satisfying a convertible anti-monotone constraint C also satisfy C). Similarly, by arranging items in non-descending order $\mathcal{R}^{-1}$ of price, if the average price of items in an itemset S falls below \$50, then the average price of items in any prefix of S would not be higher than that of S (i.e., any prefix of S violating a convertible monotone constraint C also violates C). Note that (i) any *anti-monotone constraint* is also convertible anti-monotone (for any order $\mathcal{R}$) and (ii) any *monotone constraint* is also convertible monotone (for any order $\mathcal{R}'$).

Cross-references

► Frequent Itemset Mining with Constraints

Recommended Reading

1 Pei J. and Han J. Can we push more constraints into frequent pattern mining? In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 350–354.

2. Pei J., Han J., and Lakshmanan L.V.S. Mining frequent item sets with convertible constraints. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 433–442.

3. Pei J., Han J., and Lakshmanan L.V.S. Pushing convertible constraints in frequent itemset mining. Data Mining Knowl. Discov. 8(3):227–252, 2004.

Cooperative Classification

► Visual Classification

Cooperative Content Distribution

► Peer-To-Peer Content Distribution

Cooperative Storage Systems

► Peer-to-Peer Storage

Coordination

W.M.P. van der Aalst
Eindhoven University of Technology, Eindhoven,
The Netherlands

Definition

Coordination is about managing dependencies between activities, processes, and components. Unlike the classical computation models, a coordination model puts much more emphasis on communication and cooperation than computation.

Key Points

Turing machines are a nice illustration of the classical "computation-oriented" view of systems. However, this view is too limited for many applications (e.g., web services). Many systems can be viewed as a collection of interacting entities (e.g., communicating Turing machines). For example, in the context of a service oriented architecture (SOA) coordination is more important than computation. There exist many approaches to model and support coordination. Linda is an example of a language to model coordination and communication among several parallel processes operating upon objects stored in and retrieved from a shared, virtual, associative memory [1]. Linda attempts to separate coordination from computation by only allowing interaction through tuplespaces. However, one could argue that this is also possible in classical approaches such as Petri nets (e.g., connect processes through shared places), synchronized transition systems/automata, process algebra, etc. Coordination also plays an important role in agent technology [2].

Some authors emphasize the interdisciplinary nature of coordination [3]. Coordination is indeed not a pure computer science issue and other disciplines like organizational theory, economics, psychology, etc. are also relevant.

Cross-references

▶ Business Process Management
▶ Choreography
▶ Web Services
▶ Workflow Management

Recommended Reading

1. Gelernter D. and Carriero N. Coordination languages and their significance. Commun. ACM, 35(2):97–107, 1992.
2. Jennings N.R. Commitments and conventions: the foundation of coordination in multi-agent systems. Knowl. Eng. Rev., 8(3):223–250, 1993.
3. Malone T.W. and Crowston K. The interdisciplinary study of coordination. ACM Comput. Surv., 26(1):87–119, 1994.

||-Coords

▶ Parallel Coordinates

Copy Divergence

▶ Weak Consistency Models for Replicated Data

Copy Transparency

▶ Strong Consistency Models for Replicated Data

Copyright Issues in Databases

Michael W. Carroll
Villanova University School of Law, Villanova,
PA, USA

Synonyms

Intellectual property; License

Definition

Copyright is a set of exclusive rights granted by law to authors of original works of authorship. It applies automatically as soon as an original work is created and fixed in a tangible medium of expression, such as when it is stored on a hard disk. Originality requires independent creation by the author and a modicum of creativity. Copyright covers only an author's original expression. Facts and ideas are not copyrightable. Copyright usually applies only partially to databases. Copyrightable expression usually is found in database structures, such as the selection and arrangement of field names, unless these do not reflect any creativity or are standard within an area of research. Copyright will also apply to creative data, such as photographs or expressive and sufficiently long text entries. By and large, the rule on facts and ideas means that most numerical data, scientific results, other factual data, and short text entries are not covered by copyright.

Historical Background

Copyright has evolved from a limited right to control the unauthorized distribution of a limited class of works, primarily books, to a more expansive set of rights that attach automatically to any original work of authorship. Copyright law has always been national in scope, but through international treaties most nations now extend copyright to non-resident copyright owners. To comply with these treaties, copyright is now also automatic in the USA, which has abandoned requirements that a copyright owner register the work with the Copyright Office or publish the work with the copyright symbol – © – in order to retain copyright.

Foundations

Copyright

Copyright attaches to an original work of authorship that has been embodied in a fixed form. The "work" to which copyright attaches can be the structure of the database or a relatively small part of a database, including an individual data element, such as a photograph. It is therefore possible for a database to contain multiple overlapping copyrighted works or elements. To the extent that a database owner has a copyright, or multiple copyrights, in elements of a database, the rights apply only to those copyrighted elements. The rights are to reproduce, publicly distribute or communicate, publicly display, publicly perform, and prepare adaptations or derivative works.

Standards for Obtaining Copyright

Originality Copyright protects only an author's "original" expression, which means expression independently created by the author that reflects a minimal spark of creativity. A database owner may have a copyright in the database structure or in the user interface with the database, whether that be a report form or an electronic display of field names associated with data. The key is whether the judgments made by the person(s) selecting and arranging the data require the exercise of sufficient discretion to make the selection or arrangement "original." In *Feist Publications, Inc. v. Rural Telephone Service Company*, the US Supreme Court held that a white pages telephone directory could not be copyrighted. The data—the telephone numbers and addresses – were "facts" which were not original because they had no "author." Also, the selection and arrangement of the facts did not meet the originality requirement because the decision to order the entries alphabetically by name did not reflect the "minimal spark" of creativity needed.

As a practical matter, this originality standard prevents copyright from applying to complete databases – i.e., those that list all instances of a particular phenomenon – that are arranged in an unoriginal manner, such as alphabetically or by numeric value. However, courts have held that incomplete databases that reflect original selection and arrangement of data, such as a guide to the "best" restaurants in a city, are copyrightable in their selection and arrangement. Such a copyright would prohibit another from copying and posting such a guide on the Internet without permission. However, because the copyright would be limited to that particular selection and arrangement of restaurants, a user could use such a database as a reference for creating a different selection and arrangement of restaurants without violating the copyright owner's copyright.

Copyright is also limited by the merger doctrine, which appears in many database disputes. If there are only a small set of practical choices for expressing an idea, the law holds that the idea and expression merge and the result is that there is no legal liability for using the expression.

Under these principles, metadata is copyrightable only if it reflects an author's original expression. For example, a collection of simple bibliographic metadata with fields named "author," "title," "date of publication," would not be sufficiently original to be copyrightable. More complex selections and arrangements may cross the line of originality. Finally, to the extent that software is used in a databases, software is protectable as a "literary work." A discussion of copyright in executable code is beyond the scope of this entry.

Fixation A work must also be "fixed" in any medium permitting the work to be perceived, reproduced, or otherwise communicated for a period of more than a transitory duration. The structure and arrangement of a database may be fixed any time that it is written down or implemented. For works created after January 1, 1978 in the USA, exclusive rights under copyright shower down upon the creator at the moment of fixation.

The Duration of Copyright

Under international treaties, copyright must last for at least the life of the author plus 50 years. Some countries, including the USA, have extended the length to the life of the author plus 70 years. Under U.S. law, if a work was made as a "work made for hire," such as a work created by an employee within the scope of employment, the copyright lasts for 120 years from creation if the work is unpublished or 95 years from the date of publication.

Ownership and Transfer of Copyright

Copyright is owned initially by the author of the work. If the work is jointly produced by two or more authors, such as a copyrightable database compiled by two or more scholars, each has a legal interest in the copyright. When a work is produced by an employee, ownership differs by country. In the USA, the employer is treated as the author under the "work made for hire" doctrine and the employee has no rights in the resulting work. Elsewhere, the employee is treated as the author and retains certain moral rights in the work while the employer receives the economic rights in the work. Copyrights may be licensed or transferred. A non-exclusive license, or permission, may be granted orally or even by implication. A transfer or an exclusive license must be done in writing and signed by the copyright owner. Outside of the USA, some or all of the author's moral rights cannot be transferred or terminated by agreement. The law on this issue varies by jurisdiction.

The Copyright Owner's Rights

The rights of a copyright owner are similar throughout the world although the terminology differs as do the limitations and exceptions to these rights.

Reproduction As the word "copyright" implies, the owner controls the right to reproduce the work in copies. The reproduction right covers both exact duplicates of a work and works that are "substantially similar" to the copyrighted work when it can be shown that the alleged copyist had access to the copyrighted work. In the USA, some courts have extended this right to cover even a temporary copy of a copyrighted work stored in a computer's random access memory ("RAM").

Public Distribution, Performance, Display or Communication The USA divides the rights to express the work to the public into rights to distribute copies, display a copy, or publicly perform the work. In other parts of the world, these are subsumed within a right to communicate the work to the public.

Within the USA, courts have given the distribution right a broad reading. Some courts, including the appeals court in the Napster case, have held that a download of a file from a server connected to the internet is both a reproduction by the person requesting the file and a distribution by the owner of the machine that sends the file. The right of public performance applies whenever the copyrighted work can be listened to or watched by members of the public at large or a subset of the public larger than a family unit or circle of friends. Similarly, the display right covers works that can be viewed at home over a computer network as long as the work is accessible to the public at large or a subset of the public.

Right of Adaptation, Modification or Right to Prepare Derivative Works A separate copyright arises with respect to modifications or adaptations of a copyrighted work so long as these modifications or adaptations themselves are original. This separate copyright applies only to these changes. The copyright owner has the right to control such adaptations unless a statutory provision, such as fair use, applies.

Theories of Secondary Liability

Those who build or operate databases also have to be aware that copyright law holds liable certain parties that enable or assist others in infringing copyright. In the USA, these theories are known as contributory infringement or vicarious infringement.

Contributory Infringement Contributory copyright infringement requires proof that a third party intended to assist a copyright infringer in that activity. This intent can be shown when one supplies a means of infringement with the intent to induce another to infringe or with knowledge that the recipient will infringe. This principle is limited by the so-called *Sony* doctrine, by which one who supplies a service or technology that enables infringement, such as a VCR or photocopier, will be deemed not to have knowledge of infringement or intent to induce infringement so long as the service or technology is capable of substantial non-infringing uses.

Two examples illustrate the operation of this rule. In *A&M Records, Inc. v. Napster, Inc.*, the court of appeals held that peer-to-peer file sharing is infringing but that Napster's database system for connecting users for peer-to-peer file transfers was capable of substantial non-infringing uses and so it was entitled to rely on the *Sony* doctrine. (Napster was held liable on other grounds.) In contrast, in *MGM Studios, Inc. v. Grokster, Ltd.*, the Supreme Court held that Grokster was liable for inducing users to infringe by specifically advertising its database service as a substitute for Napster's.

Vicarious Liability for Copyright Infringement Vicarious liability in the USA will apply whenever (i) one has control or supervisory power over the direct infringer's infringing conduct and (ii) one receives a direct financial benefit from the infringing conduct. In the Napster case, the court held that Napster had control over its users because it could refuse them access to the Napster server and, pursuant to the Terms of Service Agreements entered into with users, could terminate access if infringing conduct was discovered. Other courts have required a greater showing of actual control over the infringing conduct.

Similarly, a direct financial benefit is not limited to a share of the infringer's profits. The Napster court held that Napster received a direct financial benefit from infringing file trading because users' ability to obtain infringing audio files drew them to use Napster's database. Additionally, Napster could potentially receive a financial benefit from having attracted a larger user base to the service.

Limitations and Exceptions

Copyrights' limitations and exceptions vary by jurisdiction. In the USA, the broad "fair use" provision is a fact-specific balancing test that permits certain uses of copyrighted works without permission. Fair use is accompanied by some specific statutory limitations that cover, for example, certain uses in the classroom use and certain uses by libraries. The factors to consider for fair use are: (i) the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes; (ii) the nature of the copyrighted work; (iii) the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and (iv) the effect of the use upon the potential market for or value of the copyrighted work. The fact that a work is unpublished shall not itself bar a finding of fair use if such finding is made upon consideration of all the above factors.

Countries whose copyright law follows that of the United Kingdom, a more limited "fair dealing" provision enumerates specific exceptions to copyright. In Europe, Japan, and elsewhere, the limitations and exceptions are specified legislatively and cover some private copying and some research or educational uses.

Remedies and Penalties

In general, a copyright owner can seek an injunction against one who is either a direct or secondary infringer of copyright. The monetary consequences of infringement differ by jurisdiction. In the USA, the copyright owner may choose between actual or statutory damages. Actual damages cover the copyright owner's lost profits as well as a right to the infringer's profits derived from infringement. The range for statutory damages is $750–$30,000 per copyrighted work infringed. If infringement is found to have been willful, the range increases to $150,000. The amount of statutory damages in a specific case is determined by the jury. There is a safe harbor from statutory damages for non-profit educational institutions if an employee reproduces a copyrighted work with a good faith belief that such reproduction is a fair use.

A separate safe harbor scheme applies to online service providers when their database is comprised of information stored at the direction of their users. An example of such a database would be YouTube's video sharing database. The service provider is immune from monetary liability unless the provider has knowledge of infringement or has control over the infringer and receives a direct financial benefit from infringement. The safe harbor is contingent on a number of requirements, including that the provider have a copyright policy that terminates repeat infringers, that the provider comply with a notice-and-takedown procedure, and that the provider have an agent designated to receive notices of copyright infringement.

Key Applications

In cases arising after the *Feist* decision, the courts have faithfully applied the core holding that facts are in the public domain and free from copyright even when substantial investments are made to gather such facts. There has been more variation in the characterization of some kinds of data as facts and in application of the modicum-of-creativity standard to the selections and arrangements in database structures.

On the question of when data is copyrightable, a court of appeals found copyrightable expression in the "Red Book" listing of used car valuations. The defendant had copied these valuations into its database, asserting that it was merely copying unprotected factual information. The court disagreed, likening the valuations to expressive opinions and finding a modicum of originality in these. In addition, the selection and arrangement of the data, which included a division of the market into geographic regions, mileage adjustments in 5,000-mile increments, a selection of optional features for inclusion, entitled the plaintiff to a thin copyright in the database structure.

Subsequently, the same court found that the prices for futures contracts traded on the New York Mercantile Exchange (NYMEX) probably were not expressive data even though a committee makes some judgments in the setting of these prices. The court concluded that even if such price data were expressive, the merger doctrine applied because there was no other practicable way of expressing the idea other than through a numerical value and a rival was free to copy price data from NYMEX's database without copyright liability.

Finally, where data are comprised of arbitrary numbers used as codes, the courts have split. One court of appeals has held that an automobile parts manufacturer owns no copyright in its parts numbers, which are generated by application of a numbering system that the company created. In contrast, another court of appeals has held that the American Dental Association owns a copyright in its codes for dental procedures.

On the question of copyright in database structures, a court of appeals found that the structure of a yellow pages directory including listing of Chinese restaurants was entitled to a "thin" copyright, but that copyright was not infringed by a rival database that included 1,500 of the listings because the rival had not copied the plaintiff's data structure. Similarly, a different court of appeals acknowledged that although a yellow pages directory was copyrightable as a compilation, a rival did not violate that copyright by copying the name, address, telephone number, business type, and unit of advertisement purchased for each listing in the original publisher's directory. Finally, a database of real estate tax assessments that arranged the data collected by the assessor into 456 fields grouped into 34 categories was sufficiently original to be copyrightable.

Cross-references

- ► European Law in Databases
- ► Licensing and Contracting Issues in Databases

Recommended Reading

1. American Dental Association v. Delta Dental Plans Ass'n, 126 F.3d 977 (7th Cir.1997).
2. Assessment Technologies of WI, LLC v. WIRE data, Inc., 350 F.3d 640 (7th Cir. 2003).
3. Bellsouth Advertising & Publishing Corp. v. Donnelly Information Publishing, Inc., 999 F.2d 1436 (11th Cir. 1993) (en banc).
4. CCC Information Services, Inc. v. MacLean Hunter Market Reports, Inc., 44 F.3d 61 (2d Cir. 1994).
5. Feist Publications, Inc. v. Rural Telephone Service Co., 499 U.S. 340 (1991).
6. Ginsburg J.C. Copyright, common law, and sui generis protection of databases in the United States and abroad, University of Cincinnati Law Rev., 66:151–176, 1997.
7. Key Publications, Inc. v. Chinatown Today Publishing Enterprises, Inc., 945 F.2d 509 (2d Cir. 1991).
8. New York Mercantile Exchange, Inc. v. Intercontinental-Exchange, Inc., 497 F.3d 109, (2d Cir. 2007).
9. Southco, Inc. v. Kanebridge Corp., 390 F.3d 276 (3d Cir. 2004) (en banc).

CORBA

Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA

Synonyms

Object request broker; Common object request broker architecture

Definition

The Common Object Request Broker Architecture (CORBA) [2,3] is standardized by the Object Management Group (OMG) for distributed object computing.

Key Points

The CORBA standard specifies a platform-independent and programming language-independent architecture and a set of APIs to simplify distributed application development. The central idea in CORBA is to decouple the interface from the implementation. Applications that provide services declare their interfaces and operations in the Interface Description Language (IDL). IDL compilers read these definitions and synthesize client-side stubs and server-side skeletons, which provide data marshaling and proxy capabilities.

CORBA provides both a type-safe RPC-style object communication paradigm called the Static Invocation Interface (SII), and a more dynamic form of communication called the Dynamic Invocation Interface (DII), which allows creation and population of requests dynamically via reflection capabilities. The DII is often used to bridge different object models. CORBA defines a binary format for on-the-wire representation of data called the Common Data Representation (CDR). CDR has been defined to enable programming language-neutrality.

The CORBA 1.0 specification (October 1991) and subsequent revisions through version 1.2 (December 1993) defined these basic capabilities, however, they lacked any support for interoperability across different CORBA implementations.

The CORBA 2.0 specification (August 1996) defined an interoperability protocol called the General Inter-ORB Protocol (GIOP), which defines the packet formats for data exchange between communicating CORBA entities. GIOP is an abstract specification and must be mapped to the underlying transport protocol. The most widely used concrete mapping of GIOP is called the Internet Inter-ORB Protocol (IIOP) used for data exchange over TCP/IP networks.

Despite these improvements, the earlier versions of CORBA focused only on the client-side portability and lacked any support for server-side portability. This limitation was addressed in the CORBA 2.2 specification (August 1996) through the Portable Object Adapter (POA) concept. The POA enables server-side transparency to applications and server-side portability. The POA provides a number of policies that can be used to manage the server-side objects.

The CORBA specification defines compliance points for implementations to ensure interoperability. The CORBA specification has also been enhanced with additional capabilities that are available beyond the basic features, such as the Real-time CORBA specification [1]. Implementations of these specifications must provide these additional capabilities.

In general, CORBA enhances conventional procedural RPC middleware by supporting object oriented language features (such as encapsulation, interface inheritance, parameterized types, and exception handling) and advanced design patterns for distributed communication. The most recent version of CORBA specification at the time of this writing is 3.3 (January 2008), which also includes support for a component architecture.

Cross-references

- ▶ Client-Server Architecture
- ▶ DCE
- ▶ DCOM
- ▶ J2EE
- ▶ Java RMI
- ▶ .NET Remoting
- ▶ Request Broker
- ▶ SOAP

Recommended Reading

1. Object Management Group, Real-Time CORBA Specification, Version 1.2, OMG Document No. formal/2005-01-04, January 2005.
2. Object Management Group, Common Object Request Broker Architecture (CORBA), Version 3.1, OMG Document No. formal/2008-01-08, January 2008.
3. Soley R.M. and Stone C.M. Object Management Architecture Guide, 3rd edn., Object Management Group, June 1995.

Corpora

▶ Document Databases

Corpus

▶ Test Collection

Correctness Criteria Beyond Serializability

Mourad Ouzzani[1], Brahim Medjahed[2],
Ahmed K. Elmagarmid[1]
[1]Purdue University, West Lafayette, IN, USA
[2]The University of Michigan – Dearborn, Dearborn, MI, USA

Synonyms

Concurrency control; Preserving database consistency

Definition

A *transaction* is a logical unit of work that includes one or more database access operations such as insertion, deletion, modification, and retrieval [8]. A *schedule* (or history) S of n transactions $T_1,...,T_n$ is an ordering of the transactions that satisfies the following two conditions: (i) the operations of T_i (i = 1,...,n) in S must occur in the same order in which they appear in T_i, and (ii) operations from T_j (j ≠ i) may be interleaved with T_i's operations in S. A schedule S is *serial* if for every two transactions T_i and T_j that appear in S, either all operations of T_i appear before all operations of T_j, or vice versa. Otherwise, the schedule is called *nonserial* or *concurrent*. Non-serial schedules of transactions may lead to concurrency problems such as lost update, dirty read, and unrepeatable read. For instance, the lost update problem occurs whenever two transactions, while attempting to modify a data item, both read the item's old value before either of them writes the item's new value [2].

The simplest way for controlling concurrency is to allow only serial schedules. However, with no concurrency, database systems may make poor use of their resources and hence, be inefficient, resulting in smaller transaction execution rate for example. To broaden the class of allowable transaction schedules, *serializability* has been proposed as the major correctness criterion for concurrency control [7,11]. Serializability ensures that a concurrent schedule of transactions is equivalent to some serial schedule of the same transactions [12]. While serializability has been successfully used in traditional database applications, e.g., airline reservations and banking, it has been proven to be restrictive and hardly applicable in advanced applications such as Computer-Aided Design (CAD), Computer-Aided Manufacturing (CAM), office automation, and multidatabases. These applications introduced new requirements that either prevent the use of serializability (e.g., violation of local autonomy in multidatabases) or make the use of serializability inefficient (e.g., long-running transactions in CAD/CAM applications). These limitations have motivated the introduction of more flexible correctness criteria that go beyond the traditional serializability.

Historical Background

Concurrency control began appearing in database systems in the early to mid 1970s. It emerged as an active database research thrust starting from 1976 as witnessed by the early influential papers published by Eswaren et al. [5] and Gray et al. [7]. A comprehensive coverage of serializability theory has been presented in 1986 by Papadimitriou in [12]. Simply put, serializability theory is a mathematical model for proving whether or not a concurrent execution of transactions is correct. It gives precise definitions and properties that non-serial schedules of transactions must satisfy to be serializable. Equivalence between a concurrent and serial schedule of transactions is at the core of the serializability theory. Two major types of equivalence have then been defined: *conflict* and *view* equivalence. If two schedules are conflict equivalent then they are view equivalent. The converse is not generally true.

Conflict equivalence has initially been introduced by Gray et al. in 1975 [7]. A concurrent schedule of transactions is *conflict equivalent* to a serial schedule of the same transactions (and hence *conflict serializable*) if they order conflicting operations in the same way, i.e., they have the same precedence relations of conflicting operations. Two operations are *conflicting* if they are from different transactions upon the same data item, and at least one of them is `write`. If two operations conflict, their execution order matters. For instance, the value returned by a `read` operation depends on whether or not that operation precedes or follows a particular `write` operation on the same data item. Conflict serializability is tested by analyzing the acyclicity of the graph derived from the execution of the different transactions in a schedule. This graph, called *serializability graph*, is a directed graph that models the precedence of conflicting operations in the transactions.

View equivalence has been proposed by Yannakakis in 1984 [15]. A concurrent schedule of transactions is *view equivalent* to a serial schedule of the same transactions (and hence *view serializable*) if the respective transactions in the two schedules read and write the same data values. View equivalence is based on the following two observations: (i) if each transaction reads each of its data items from the same `writes`, then all `writes` write the same value in both schedules; and (ii) if the final `write` on each data item is the same in both schedules, then the final value of all data items will be the same in both schedules. View serializability is usually expensive to check. One approach is to check the acyclicity of a special graph called *polygraph*. A polygraph is a generalization of the precedence graph that takes into account all precedence constraints required by view serializability.

Foundations

The limitations of the traditional serializability concept combined with the requirement of advanced database applications triggered a wave of new correctness criteria that go beyond serializability. These criteria aim at achieving one or several of the following goals: (i) accept non serializable but correct executions by exploiting the semantics of transactions, their structure, and integrity constraints (ii) allow inconsistencies to appear in a controlled manner which may be acceptable for some transactions, (iii) limit conflicts by creating a new version of the data for each update, and (iv) treat transactions accessing more than one database, in the case of multidatabases, differently from those accessing one single database and maintain overall correctness. While a large number of correctness criteria have been presented in the literature, this entry will focus on the major criteria which had a considerable impact on the field. These criteria will be presented as described in their original versions as several of these criteria have been either extended, improved, or applied to specific contexts. Table 1 summarizes the correctness criteria outlined in this section.

Multiversion Serializability

Multiversion databases aim at increasing the degree of concurrency and providing a better system recovery. In such databases, whenever a transaction writes a data

Correctness Criteria Beyond Serializability. Table 1. Representative correctness criteria for concurrency control

Correctness criterion	Basic idea	Examples of application domains	Reference
Multiversion serializability	Allows some schedules as serializable if a read is performed on some older version of a data item instead of the newer modified version.	Multiversion database systems	[1]
Semantic consistency	Uses semantic information about transactions to accept some non-serializable but correct schedules.	Applications that can provide some semantic knowledge	[6]
Predicatewise serializability	Focuses on data integrity constraints.	CAD database and office information systems	[9]
Epsilon-serializability	Allows inconsistencies to appear in a controlled manner by attaching a specification of the amount of permitted inconsistency to each transaction.	Applications that tolerate some inconsistencies	[13]
Eventual consistency	Requires that duplicate copies are consistent at certain times but may be inconsistent in the interim intervals.	Distributed databases with replicated or interdependent data	[14]
Quasi serializability	Executes global transactions in a serializable way while taking into account the effect of local transactions.	Multidatabase systems	[4]
Two-level serializability	Ensures consistency by exploiting the nature of integrity constraints and the nature of transactions in multidatabase environments.	Multidatabase systems	[10]

item, it creates a new version of this item instead of overwriting it. The basic idea of *multiversion serializability* [1] is that some schedules can be still seen as serializable if a read is performed on some older version of a data item instead of the newer modified version. Concurrency is increased by having transactions read older versions while other concurrent transactions are creating newer versions. There is only one type of conflict that is possible; when a transactions reads a version of a data item that was written by another transaction. The two other conflicts (write, write) and (read, write) are not possible since each write produces a new version and a data item cannot be read until it has been produced, respectively. Based on the assumption that users expect their transactions to behave as if there were just one copy of each data item, the notion of *one-copy serial* schedule is defined. A schedule is one-copy serial if for all i, j, and x, if a transaction T_j reads x from a transaction T_i, then either $i = j$ or T_i is the last transaction preceding t_j that writes into any version of x. Hence, a schedule is defined as *one-copy serializable* (1-SR) if it is equivalent to a 1-serial schedule. 1-SR is shown to maintain correctness by proving that a multiversion schedule behaves like a serial non-multiversion schedule (there is only one version for each data item) if the multiversion schedule is one-serializable. The one-copy serializability of a schedule can be verified by checking the acyclicity of the multiversion serialization graph of that schedule.

Semantic Consistency

Semantic consistency uses semantic information about transactions to accept some non-serializable but correct schedules [6]. To ensure that users see consistent data, the concept of *sensitive transactions* has been introduced. Sensitive transactions output only consistent data and thus must see a consistent database state. A semantically consistent schedule is one that transforms the database from a consistent state to another consistent state and where all sensitive transactions obtain a consistent view of the database with respect to the data accessed by these transactions, i.e., all data consistency constraints of the accessed data are evaluated to True. Enforcing semantic consistency requires knowledge about the application which must be provided by the user. In particular, users will need to group actions of the transactions into steps and specify which steps of a transaction of a given type can be interleaved with the steps of another type of transactions without violating consistency. Four types of semantic knowledge are defined: (i) transaction semantic types, (ii) compatibility sets associated with each type, (iii) division of transactions into steps, and (iv) counter-steps to (semantically) compensate the effect from some of the steps executed within the transaction.

Predicatewise Serializability

Predicatewise serializability (PWSR) has been introduced as a correctness criterion for CAD database and office information systems [9]. PWSR focuses solely on data integrity constraints. In a nutshell, if database consistency constraints can be expressed in a conjunctive normal form, a schedule is said to be PWSR if all projections of that schedule on each group of data items that share a disjunctive clause (of the conjunctive form representing the integrity constraints) are serializable. There are three different types of restrictions that must be enforced on PWSR schedules to preserve database consistency: (i) force the transactions to be of *fixed structure*, i.e., they are independent of the database state from which they execute, (ii) force the schedules to be *delayed read*, i.e., a transaction T_i cannot read a data item written by a transaction T_j until after T_j has completed all of its operations, or (iii) the conjuncts of the integrity constraints can be ordered in a way that no transaction reads a data item belonging to a higher numbered conjunct and writes a data item belonging to a lower numbered conjunct.

Epsilon-Serializability

Epsilon-serializability (ESR) [13] has been introduced as a generalization to serializability where a limited amount of inconsistency is permitted. The goal is to enhance concurrency by allowing some non serializable schedules. ESR introduces the notion of *epsilon transactions* (ETs) by attaching a specification of the amount of permitted inconsistency to each (standard) transaction. ESR distinguishes between transactions that contain only read operation, called query epsilon transaction or query ET, and transactions with at least one update operation, called update epsilon transaction or update ET. Query ETs may view uncommitted, possibly inconsistent, data being updated by update ETs. Thus, update ETs are seen as exporting some inconsistencies while query ETs are importing these inconsistencies. ESR aims at bounding the amount of imported and exported inconsistency for each ET. An *epsilon-serial* schedule is defined as a schedule where (i) the update ETs form a serial schedule if considered alone without the query ET and (ii) the entire schedule consisting of both query ETs and update ETs is such that the non serializable conflicts between query ETs and update ETs are less than the permitted limits specified by each ET. An epsilon-serializable schedule is one that is equivalent to an epsilon-serial schedule. If the permitted limits are set to zero, ESR corresponds to the classical notion of serializability.

Eventual Consistency

Eventual consistency has been proposed as an alternative correctness criterion for distributed databases with replicated or interdependent data [14]. This criterion is useful is several applications like mobile databases, distributed databases, and large scale distributed systems in general. Eventual consistency requires that duplicate copies are consistent at certain times but may be inconsistent in the interim intervals. The basic idea is that duplicates are allowed to diverge as long as the copies are made consistent periodically. The times where these copies are made consistent can be specified in several ways which could depend on the application, for example, at specified time intervals, when some events occur, or at some specific times. A correctness criterion that ensures eventual consistency is the *current copy serializability*. Each update occurs on a current copy and is asynchronously propagated to other replicas.

Quasi Serializability

Quasi Serializability (QSR) is a correctness criterion that has been introduced for multidatabase systems [4]. A multidatabase system allows users to access data located in multiple autonomous databases. It generally involves two kinds of transactions: (i) Local transactions that access only one database; they are usually outside the control of the multidatabase system, and (ii) global transactions that can access more than one database and are subject to control by both the multidatabase and the local databases. The basic premise is that to preserve global database consistency, global transactions should be executed in a serializable way while taking into account the effect of local transactions. The effect of local transactions appears in the form of indirect conflicts that these local transactions introduce between global transactions which may not necessarily access (conflict) the same data items. A *quasi serial* schedule is a schedule where global transactions are required to execute serially and local schedules are required to be serializable. This is in contrast to global serializability where all transactions, both local and global, need to execute in a (globally) serializable way. A global schedule is said to be quasi serializable if it is (conflict) equivalent to a quasi serial schedule. Based on this definition, a quasi serializable schedule maintains the consistency of multidatabase systems since (i) a quasi serial schedule preserves the mutual consistency of globally replicated data items, based on the assumptions that these replicated data items are updated only by global transactions, and (ii) a quasi serial schedule preserves the global transaction consistency constraints as local schedules are serializable and global transactions are executed following a schedule that is equivalent to a serial one.

Two-Level Serializability

Two-level serializability (2LSR) has been introduced to relax serializability requirements in multidatabases and allow a higher degree of concurrency while ensuring consistency [10]. Consistency is ensured by exploiting the nature of integrity constraints and the nature of transactions in multidatabase environments. A global schedule, consisting of both local and global transactions, is 2LSR if all local schedules are serializable and the projection of that schedule on global transactions is serializable. Local schedules consist of all operations, from global and local transactions, that access the same local database. Ensuring that each local schedule is serializable is already taken care of by the local database. Furthermore, ensuring that the global transactions are executed in a serializable way can be done by the global concurrency controller using any existing technique from centralized databases like the Two-phase-locking (2PL) protocol. This is possible since the global transactions are under the full control of the global transaction manager. [10] shows that under different scenarios 2LSR preserves a strong notion of correctness where the multidatabase consistency is preserved and all transactions see consistent data. These different scenarios differ depending on (i) which kind of data items, local or global, global and local transactions are reading or writing, (ii) the existence of integrity constraints between local and global data items, and (iii) whether all transaction are preserving the consistency of local databases when considered alone.

Key Applications

The major database applications behind the need for new correctness criteria include distributed databases, mobile databases, multidatabases, CAD/CAM applications, office automation, cooperative applications, and software development environments. All of these advanced applications introduced requirements and limitations that either prevent the use of serializability like the violation of local autonomy in multidatabases, or make the use of serializability inefficient like blocking long-running transactions.

Future Directions

A recent trend in transaction management focuses on adding transactional properties (e.g., isolation, atomicity) to business processes [3]. A business process (BP) is a set of tasks which are performed collaboratively to realize a business objective. Since BPs contain activities that access shared and persistent data resources, they have to be subject to transactional semantics. However, it is not adequate to treat an entire BP as a single "traditional" transaction mainly because BPs: (i) are of long duration and treating an entire process as a transaction would require locking resources for long periods of time, (ii) involve many independent database and application systems and enforcing transactional properties across the entire process would require expensive coordination among these systems, and (iii) have external effects and using conventional transactional rollback mechanisms is not feasible. These characteristics open new research issues to take the concept of correctness criterion and how it should be enforced beyond even the correctness criteria discussed here.

Cross-references

- ▶ ACID Properties
- ▶ Concurrency Control
- ▶ Distributed
- ▶ Parallel and Networked Databases
- ▶ System Recovery
- ▶ Transaction Management
- ▶ Two-Phase Commit
- ▶ Two-Phase Locking

Recommended Reading

1. Bernstein P.A. and Goodman N. Multiversion concurrency control – theory and algorithms. ACM Trans. Database Syst., 8(4):465–483, 1983.
2. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency control and recovery in database systems. Addison-Wesley, Reading, MA, 1987.
3. Dayal U., Hsu M., and Ladin R. Business process coordination: state of the art, trends, and open issues. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 3–13.

4. Du W. and Elmagarmid A.K. Quasi serializability: a correctness criterion for global concurrency control in Interbase. In Proc. 15th Int. Conf. on Very Large Data Bases, 1989, pp. 347–355.
5. Eswaran K.P., Gray J., Lorie R.A., and Traiger I.L. The notions of consistency and predicate locks in a database system. Commun. ACM, 19(11):624–633, 1976.
6. Garcia-Molina H. Using semantic knowledge for transaction processing in a distributed database. ACM Trans. Database Syst., 8(2):186–213, 1983.
7. Gray J., Lorie R.A., Putzolu G.R., and Traiger I.L. Granularity of locks in a large shared data base. In Proc. 1st Int. Conf. on Very Data Bases, 1975, pp. 428–451.
8. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, Los Altos, CA, 1993.
9. Korth H.F. and Speegle G.D. Formal model of correctness without serializability. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988, pp. 379–386.
10. Mehrotra S., Rastogi R., Korth H.F., and Silberschatz A. Ensuring consistency in multidatabases by preserving two-level serializability. ACM Trans. Database Syst., 23(2):199–230, 1998.
11. Papadimitriou C.H. The serializability of concurrent database updates. J. ACM, 26(4):631–653, 1979.
12. Papadimitriou C.H. The Theory of Database Concurrency Control. Computer Science, Rockville, MD, 1986.
13. Ramamritham K. and Pu C. A formal characterization of epsilon serializability. IEEE Trans. Knowl. Data Eng., 7(6):997–1007, 1995.
14. Sheth A., Leu Y., and Elmagarmid A. Maintaining Consistency of Interdependent Data in Multidatabase Systems. Tech. Rep. CSD-TR-91-016, Purdue University, http://www.cs.toronto.edu/georgem/ws/ws.ps, 1991.
15. Yannakakis M. Serializability by locking. J. ACM, 31(2): 227–244, 1984.

Correctness Criterion for Concurrent Executions

► Serializability

Correlated Data Collection

► Data Compression in Sensor Networks

Correlation

► Similarity and Ranking Operations

Correlation Clustering

► Subspace Clustering Techniques

Cost Estimation

Stefan Manegold
CWI, Amsterdam, The Netherlands

Definition

Execution costs, or simply *costs*, is a generic term to collectively refer to the various goals or objectives of database query optimization. Optimization aims at finding the "cheapest" ("best" or at least a "reasonably good") query execution plan (QEP) among semantically equivalent alternative plans for the given query. Cost is used as a metric to compare plans. Depending on the application different types of costs are considered. Traditional optimization goals include minimizing response time (for the first answer or the complete result), minimizing resource consumption (like CPU time, I/O, network bandwidth, or amount of memory required), or maximizing throughput, i.e., the number of queries that the system can answer per time. Other, less obvious objectives – e.g., in a mobile environment – may be to minimize the power consumption needed to answer the query or the on-line time being connected to a remote database server.

Obviously, evaluating a QEP to measure its execution cost does not make sense. *Cost estimation* refers to the task of predicting the (approximate) costs of a given QEP a priori, i.e., without actually evaluating it. For this purpose, mathematical algorithms or parametric equations, commonly referred to as *cost models*, provide a simplified "idealized" abstract description of the system, focusing on the most relevant components. In general, the following three cost components are distinguished.

1. *Logical costs* consider only the data distributions and the semantics of relational algebra operations to estimate intermediate result sizes of a given (logical) query plan.
2. *Algorithmic costs* extend logical costs by taking also the computational complexity (expressed in terms of *O*-classes) of the algorithms into account.
3. *Physical costs* finally combine algorithmic costs with system/hardware specific parameters to predict the total costs, usually in terms of execution time.

Next to query optimization, cost models can serve another purpose. Especially algorithmic and physical cost models can help database developers to understand

and/or predict the performance of existing algorithms on new hardware systems. Thus, they can improve the algorithms or even design new ones without having to run time and resource consuming experiments to evaluate their performance.

Since the quality of query optimization strongly depends on the quality of cost estimation, details of cost estimation in commercial database products are usually well kept secrets of their vendors.

Historical Background

Not all aspects of database cost estimation are treated as independent research topic of their own. Mainly selectivity estimation and intermediate result size estimation have received intensive attention yielding a plethora of techniques proposed in database literature. Discussion of algorithmic costs usually occurs with the proposal of new or modified database algorithms. Given its tight coupling with query optimization, physical cost estimation has never been an independent research topic of its own. Apart from very few exceptions, new physical cost models and estimation techniques are usually published as “by-products” in publications that mainly deal with novel optimization techniques.

The first use of (implicit) cost estimation were complexity analyses that led to heuristic optimization rules. For instance, a join is always considered cheaper than calculating first the Cartesian product, followed by a selection. Likewise, linear operations that tend to reduce the data stream (selections, projections) should be evaluated as early as data dependencies allow, followed by (potentially) quadratic operations that do not “blow-up” the intermediate results (semijoins, foreign-key joins). More complex, and hence expensive, operations (general joins, Cartesian products) should be executed as late as possible.

Since a simple complexity metric does not necessarily reflect the same ranking of plans as the actual execution costs, first explicit cost estimation in database query optimization aimed at estimating intermediate result sizes. Initial works started with simplifications such as assuming uniform data distributions and independence of attribute values. Over time, the techniques have been improved to model non-uniform data distributions. Til date, effective treatment of (hidden) correlations is still an open research topic.

The following refinement was the introduction of physical costs. With I/O being the dominating cost factor in the early days of database management systems, the first systems assessed query plans by merely estimating the number of I/O operations required. However, I/O systems exhibit quite different performance for sequential and randomly placed I/O operations. Hence, the models were soon refined to distinguish between sequential and random accesses, weighing them with their respective costs, i.e., time to execute one operation.

With main memory sizes growing, more and more query processing work is done within main memory, minimizing disk accesses. Consequently, CPU and memory access costs can no longer be ignored. Assuming uniform memory access costs, memory access has initially been covered by CPU costs. CPU costs are estimated in terms of CPU cycles. Scoring them with the CPU’s clock speed yields time, the common unit to combine CPU and I/O costs to get the overall physical costs.

Only recently with the advent of CPU caches and extended memory hierarchies, the impact of memory access costs has become so significant that it needs to be modeled separately [15,16]. Similarly to I/O costs, memory access costs are estimated in terms of number of memory accesses (or cache misses) and scored by their penalty to achieve time as common unit.

In parallel and distributed database systems, also network communication costs are considered as contributing factors to the overall execution costs.

Foundations

Different query execution plans require different amounts of effort to be evaluated. The objective function for the query optimization problems assigns every execution plan a single non-negative value. This value is commonly referred to as *costs* in the query optimization business.

Cost Components

Logical Costs/Data Volume The most important cost component is the amount of data that is to be processed. Per operator, three data volumes are distinguished: input (per operand), output, and temporary data. Data volumes are usually measured as cardinality, i.e., number of tuples. Often, other units such as number of I/O blocks, number of memory pages, or total size in bytes are required. Provided that the respective tuple sizes, page sizes, and block sizes are known, the cardinality can easily be transformed into the other units.

The amount of input data is given as follows: For the leaf nodes of the query graph, i.e., those operations that directly access base tables stored in the database, the input cardinality is given by the cardinality of the base table(s) accessed. For the remaining (inner) nodes of the query graph, the input cardinality is given by the output cardinality of the predecessor(s) in the query graph.

Estimating the output size of database operations – or more generally, their *selectivity* – is anything else but trivial. For this purpose, DBMSs usually maintain statistic about the data stored in the database. Typical statistics are

1. Cardinality of each table,
2. Number of distinct values per column,
3. Highest/lowest value per column (where applicable).

Logical cost functions use these statistics to estimate output sizes (respectively selectivities) of database operations. The simplest approach is to assume that attribute values are uniformly distributed over the attribute's domain. Obviously, this assumption virtually never holds for "real-life" data, and hence, estimations based on these assumption will never be accurate. This is especially severe, as the estimation errors compound exponentially throughout the query plan [9]. This shows, that more accurate (but compact) statistics on data distributions (of base tables as well as intermediate results) are required to estimate intermediate results sizes.

The importance of statistics management has led to a plethora of approximation techniques, for which [6] have coined the general term "*data synopses*". Such techniques range from advanced forms of *histograms* (most notably, *V-optimal histograms* including multidimensional variants) [7,10] over *spline synopses* [12,11], *sampling* [3,8], and *parametric curve-fitting techniques* [4,20] all the way to highly sophisticated methods based on *kernel estimators* [1] or *Wavelets* and other transforms [2,17].

A logical cost model is a prerequisite for the following two cost components.

Algorithmic Costs/Complexity

Logical costs only depend on the data and the query (i.e., the operators' semantics), but they do not consider the algorithms used to implement the operators' functionality. Algorithmic costs extend logical costs by taking the properties of the algorithms into account.

A first criterion is the algorithm's complexity in the classical sense of complexity theory. Most unary operator are in $O(n)$, like selections, or $O(n \log n)$, like sorting; n being the input cardinality. With proper support by access structures like indices or hash tables, the complexity of selection may drop to $O(\log n)$ or $O(1)$, respectively. Binary operators can be in $O(n)$, like a union of sets that does not eliminate duplicates, or, more often, in $O(n^2)$, as for instance join operators.

More detailed algorithmic cost functions are used to estimate, e.g., the number of I/O operations or the amount of main memory required. Though these functions require some so-called "physical" information like I/O block sizes or memory pages sizes, they are still considered algorithmic costs and not physical cost, as these informations are system specific, but not hardware specific. The standard database literature provides a large variety of cost formulas for the most frequently used operators and their algorithms. Usually, these formulas calculate the costs in terms of I/O operations as this still is the most common objective function for query optimization in database systems [5,13].

Physical Costs/Execution Time

Logical and algorithmic costs alone are not sufficient to do query optimization. For example, consider two algorithms for the same operation, where the first algorithm requires slightly more I/O operations than the second, while the second requires significantly more CPU operations than the first one. Looking only at algorithmic costs, both algorithms are not comparable. Even assuming that I/O operations are more expensive than CPU operations cannot in general answer the question which algorithm is faster. The actual execution time of both algorithms depends on the speed of the underlying hardware. The physical cost model combines the algorithmic cost model with an abstract hardware description to derive the different cost factors in terms of time, and hence the total execution time. A hardware description usually consists of information such as CPU speed, I/O latency, I/O bandwidth, and network bandwidth. The next section discusses physical cost factors on more detail.

Cost Factors

In principle, physical costs are considered to occur in two flavors, *temporal* and *spatial*. Temporal costs cover all cost factors that can easily be related to execution time, e.g., by multiplying the number of certain events with there respective cost in terms of some time unit. Spatial costs contain resource consumptions that cannot directly (or not at all) be related to time. The following briefly describes the most prominent cost factors of both categories.

Temporal Cost Factors

Disk-I/O This is the cost of searching for, reading, and writing data blocks that reside on secondary storage, mainly on disk. In addition to accessing the database files themselves, temporary intermediate files that are too large to fit in main memory buffers and hence are stored on disk also need to be accessed. The cost of searching for records in a database file or a temporary file depends on the type of access structures on that file, such as ordering, hashing, and primary or secondary indexes. I/O costs are either simply measured in terms of the number of block-I/O operations, or in terms of the time required to perform these operations. In the latter case, the number of block-I/O operations is multiplied by the time it takes to perform a single block-I/O operation. The time to perform a single block-I/O operation is made up by an initial seek time (*I/O latency*) and the time to actually transfer the data block (i.e., block size divided by *I/O bandwidth*). Factors such as whether the file blocks are allocated contiguously on the same disk cylinder or scattered across the disk affect the access cost. In the first case (also called *sequential I/O*), I/O latency has to be counted only for the first of a sequence of subsequent I/O operations. In the second case (*random I/O*), seek time has to be counted for each I/O operation, as the disk heads have to be repositioned each time.

Main-Memory Access These are the costs for reading data from or writing data to main memory. Such data may be intermediate results or any other temporary data produced/used while performing database operations.

Similar to I/O costs, memory access costs can be modeled be estimating the number of memory accesses (i.e., cache misses) and scoring them with their respective penalty (latency) [16].

Network Communication In centralized DBMSs, communication costs cover the costs of shipping the query from the client to the server and the query's result back to the client. In distributed, federated, and parallel DBMSs, communication costs additionally contain all costs for shipping (sub-)queries and/or (intermediate) results between the different hosts that are involved in evaluating the query.

Also with communication costs, there is a latency component, i.e., a delay to initiate a network connection and package transfer, and a bandwidth component, i.e., the amount of data that can be transfer through the network infrastructure per time.

CPU Processing This is the cost of performing operations such as computations on attribute values, evaluating predicates, searching and sorting tuples, and merging tuples for join. CPU costs are measured in either CPU cycles or time. When using CPU cycles, the time may be calculated by simply dividing the number of cycles by the CPU's clock speed. While allowing limited portability between CPUs of the same kind, but with different clock speeds, portability to different types of CPUs is usually not given. The reason is, that the same basic operations like adding two integers might require different amounts of CPU cycles on different types of CPUs.

Spatial Cost Factors

Usually, there is only one spatial cost factor considered in database literature: *memory size*. This cost it the amount of main memory required to store intermediate results or any other temporary data produced/used while performing database operations.

Next to not (directly) being related to execution time, there is another difference between temporal and spatial costs that stems from the way they share the respective resources. A simple example shall demonstrate the differences. Consider to operations or processes each of which consumes 50% of the available resources (i.e., CPU power, I/O-, memory-, and network bandwidth). Further, assume that when run one at a time, both tasks have equal execution time. Running both tasks concurrently on the same system (ideally) results in the same execution time, now consuming all the available resources. In case each individual process consumes 100% of the available resources, the concurrent execution time will be twice the individual execution time. In other words, if the combined resource consumption of concurrent tasks exceed 100%, the execution time extends to accommodate the excess resource requirements. With spatial cost factors, however, such "stretching" is not possible. In case two tasks together would require more than 100% of the available memory, they simply cannot be executed at the same time, but only after another.

Types of (Cost) Models

According to their degree of abstraction, (cost) models can be classified into two classes: *analytical models* and *simulation models*.

Analytical Models In some cases, the assumptions made about the real system can be translated into mathematical descriptions of the system under study. Hence, the result is a set of mathematical formulas that is called an analytical model. The advantage of an analytical model is that evaluation is rather easy and hence fast. However, analytical models are usually not very detailed (and hence not very accurate). In order to translate them into a mathematical description, the assumptions made have to be rather general, yielding a rather high degree of abstraction.

Simulation Models Simulation models provide a very detailed and hence rather accurate description of the system. They describe the system in terms of (a) simulation experiment(s) (e.g., using event simulation). The high degree of accuracy is charged at the expense of evaluation performance. It usually takes relatively long to evaluate a simulation base model, i.e., to actually perform the simulation experiment(s). It is not uncommon, that the simulation actually takes longer than the execution in the real system would take.

In database query optimization, though it would appreciate the accuracy, simulation models are not feasible, as the evaluation effort is far to high. Query optimization requires that costs of numerous alternatives are evaluated and compared as fast as possible. Hence, only analytical cost models are applicable in this scenario.

Architecture and Evaluation of Database Cost Models

The architecture and evaluation mechanism of database cost models is tightly coupled to the structure of query execution plans. Due to the strong encapsulation offered by relational algebra operators, the cost of each operator, respectively each algorithm, can be described individually. For this purpose, each algorithm is assigned a set of *cost functions* that calculate the three cost components as described above. Obviously, the physical cost functions depend on the algorithmic cost functions, which in turn depend on the logical cost functions. Algebraic cost functions use the data volume estimations of the logical cost functions as input parameters. Physical cost functions are usually specializations of algorithmic cost functions that are parametrized by the hardware characteristics.

The cost model also defines how the single operator costs within a query have to be combined to calculate the total costs of the query. In traditional sequential DBMSs, the single operators are assumed to have no performance side-effects on each other. Thus, the cost of a QEP is the cumulative cost of the operators in the QEP [18]. Since every operator in the QEP is the root of a sub-plan, its cost includes the cost of its input operators. Hence, the cost of a QEP is the cost of the topmost operator in the QEP. Likewise, the cardinality of an operator is derived from the cardinalities of its inputs, and the cardinality of the topmost operator represents the cardinality of the query result.

In non-sequential (e.g., distributed or parallel) DBMSs, this subject is much more complicated, as more issues such as scheduling, concurrency, resource contention, and data dependencies have to considered. For instance, in such environments, more than one operator may be executed at a time, either on disjoint (hardware) resources, or (partly) sharing resources. In the first case, the total cost (in terms of time) is calculated as the maximum of the costs (execution times) of all operators running concurrently. In the second case, the operators compete for the same resources, and hence mutually influence their performance and costs. More sophisticated cost function and cost models are required here to adequately model this resource contention [14,19].

Cross-references

- ▶ Distributed Query Optimization
- ▶ Multi-Query Optimization
- ▶ Optimization and Tuning in Data Warehouses
- ▶ Parallel Query Optimization
- ▶ Process Optimization
- ▶ Query Optimization
- ▶ Query Optimization (in Relational Databases)
- ▶ Query Optimization in Sensor Networks
- ▶ Query Plan
- ▶ Selectivity Estimation
- ▶ Spatio-Temporal Selectivity Estimation
- ▶ XML Selectivity Estimation

Recommended Reading

1. Blohsfeld B., Korus D., and Seeger B. A comparison of selectivity estimators for range queries on metric attributes. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 239–250.
2. Chakrabarti K., Garofalakis M.N., Rastogi R., and Shim K. Approximate query processing using wavelets. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 111–122.
3. Chaudhuri S., Motwani R., and Narasayya V.R. On random sampling over joins. In Proc. ACM SIGMOD Int. Conf.

on Management of Data, Philadephia, PA, USA, June 1999, pp. 263–274.

4. Chen C.M. and Roussopoulos N. Adaptive selectivity estimation using query feedback. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 161–172.
5. Garcia-Molina H., Ullman J.D., and Widom J. Database Systems: The Complete Book. Prentice Hall, Englewood Cliffs, NJ, USA, 2002.
6. Gibbons P.B. and Matias Y. Synopsis data structures for massive data sets. In Proc. 10th Annual ACM-SIAM Symp. on Discrete Algorithms, 1999, pp. 909–910.
7. Gibbons P.B., Matias P.B., and Poosala V. Fast incremental maintenance of approximate histograms. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 466–475.
8. Haas P.J., Naughton J.F., Seshadri S., and Swami A.N. Selectivity and cost estimation for joins based on random sampling. J. Comput. Syst. Sci., 52(3):550–569, 1996.
9. Ioannidis Y.E. and Christodoulakis S. On the propagation of errors in the size of join results. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991, pp. 268–277.
10. Ioannidis Y.E. and Poosala V. Histogram-based approximation of set-valued query-answers. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 174–185.
11. König A.C. and Weikum G. Combining histograms and parametric curve fitting for feedback-driven query result-size estimation. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 423–434.
12. König A.C. and Weikum G. Auto-tuned spline synopses for database statistics management. In Proc. Int. Conf. on Management of Data, 2000.
13. Korth H. and Silberschatz A. Database Systems Concepts. McGraw-Hill, Inc., New York, San Francisco, Washington, DC, USA, 1991.
14. Lu H., Tan K.L., and Shan M.C. Hash-based join algorithms for multiprocessor computers. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 198–209.
15. Manegold S. Understanding, Modeling, and Improving Main-Memory Database Performance. PhD Thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, December 2002.
16. Manegold S., Boncz P.A., and Kersten M.L. Generic database cost models for hierarchical memory systems. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 191–202.
17. Matias Y., Vitter J.S., and Wang M. Wavelet-based histograms for selectivity estimation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 448–459.
18. Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., and Price T.G. Access path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.
19. Spiliopoulou M. and Freytag J.-C. Modelling resource utilization in pipelined query execution. In Proc. European Conference on Parallel Processing, 1996, pp. 872–880.
20. Sun W., Ling Y., Rishe N., and Deng Y. An instant and accurate size estimation method for joins and selection in a retrieval-intensive environment. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 79–88.

Count-Min Sketch

Graham Cormode
AT&T Labs–Research, Florham Park, NJ, USA

Synonyms

CM Sketch

Definition

The Count-Min (CM) Sketch is a compact summary data structure capable of representing a high-dimensional vector and answering queries on this vector, in particular point queries and dot product queries, with strong accuracy guarantees. Such queries are at the core of many computations, so the structure can be used in order to answer a variety of other queries, such as frequent items (heavy hitters), quantile finding, join size estimation, and more. Since the data structure can easily process updates in the form of additions or subtractions to dimensions of the vector (which may correspond to insertions or deletions, or other transactions), it is capable of working over streams of updates, at high rates.

The data structure maintains the linear projection of the vector with a number of other random vectors. These vectors are defined implicitly by simple hash functions. Increasing the range of the hash functions increases the accuracy of the summary, and increasing the number of hash functions decreases the probability of a bad estimate. These tradeoffs are quantified precisely below. Because of this linearity, CM sketches can be scaled, added and subtracted, to produce summaries of the corresponding scaled and combined vectors.

Historical Background

The Count-Min sketch was first proposed in 2003 [5] as an alternative to several other sketch techniques, such as the Count sketch [3] and the AMS sketch [1]. The goal was to provide a simple sketch data structure with a precise characterization of the dependence on the input parameters. The sketch has also been viewed as a realization of a counting *Bloom filter* or Multistage-Filter [8], which requires only limited independence randomness to show strong, provable guarantees. The simplicity of creating and probing the sketch has led to its wide use in disparate areas since its initial description.

Foundations

The CM sketch is simply an array of counters of width w and depth d, $CM[1, 1] \ldots CM[d, w]$. Each entry of the array is initially zero. Additionally, d hash functions

$$h_1 \ldots h_d : \{1 \ldots n\} \rightarrow \{1 \ldots w\}$$

are chosen uniformly at random from a pairwise-independent family. Once w and d are chosen, the space required is fixed: the data structure is represented by wd counters and d hash functions (which can each be represented in $O(1)$ machine words [14]).

Update Procedure

Consider a vector a, which is presented in an implicit, incremental fashion (this abstract model captures a wide variety of data stream settings, see entries on *Data Stream Models* for more details). This vector has dimension n, and its current state at time t is $a(t) = [a_1(t), \ldots a_i(t), \ldots a_n(t)]$. Initially, $a(0)$ is the zero vector, 0, so $a_i(0)$ is 0 for all i. Updates to individual entries of the vector are presented as a stream of pairs. The tth update is (i_t, c_t), meaning that

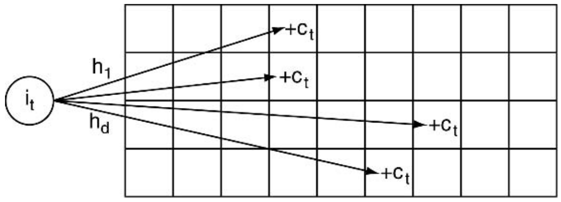
$$\begin{aligned} a_{i_t}(t) &= a_{i_t}(t-1) + c_t \\ a_{i'}(t) &= a_{i'}(t-1) \qquad i' \neq i_t \end{aligned}$$

This procedure is illustrated in Fig. 1. In the remainder of this article, t is dropped, and the current state of the vector is referred to as just a for convenience. It is assumed throughout that although values of a_i increase and decrease with updates, each $a_i \geq 0$. The Count-Min sketch also applies to the case where a_is can be less than zero, with small factor increases in space. Here, details of these extensions are omitted for simplicity of exposition (full details are in [5]).

When an update (i_t, c_t) arrives, c_t is added to one count in each row of the Count-Min sketch; the counter is determined by h_j . Formally, given (i_t, c_t), the following modifications are performed:

$$\forall 1 \leq j \leq d : CM[j, h_j(i_t)] \leftarrow CM[j, h_j(i_t)] + c_t$$

Because computing each hash function takes $O(1)$ (constant) time, the total time to perform an update is $O(d)$, independent of w. Since d is typically small in practice (often less than 10), updates can be processed at high speed.



Count-Min Sketch. Figure 1. Each item i is mapped to one cell in each row of the array of counts: when an update of c_t to item i_t arrives, c_t is added to each of these cells.

Point Queries

A *point query* is to estimate the value of an entry in the vector a_i. The point query procedure is similar to updates: given a query point i, an estimate is found as $\hat{a}_i = \min_{1 \leq j \leq d} CM[j, h_j(i)]$. Since the space used by the sketch is typically much smaller than that required to represent the vector exactly, there is necessarily some approximation in the estimate, which is quantified as follows:

Theorem 1 (Theorem 1 from [5]). *If* $w = \lceil \frac{e}{\varepsilon} \rceil$ *and* $d = \lceil \ln \frac{1}{\delta} \rceil$, *the estimate* $\hat{a}_i$ *has the following guarantees:* $a_i \leq \hat{a}_i$; *and, with probability at least* $1 - \delta$,

$$\hat{a}_i \leq a_i + \varepsilon \|a\|_1.$$

The proof follows by considering the estimate in each row, and observing that the expected error in using $CM[j, h_j(i)]$ as an estimate has expected (non-negative) error $\|a\|_1 / w$. By the Markov inequality [14], the probability that this error exceeds $\epsilon \|a\|_1$ is at most $\frac{1}{e}$ (where e is the base of the natural logarithm, i.e., 2.71828 . . ., a constant chosen to optimize the space for fixed accuracy requirements). Taking the smallest estimate gives the best estimator, and the probability that this estimate has error exceeding $\epsilon \|a\|_1$ is the probability that *all* estimates exceed this error, i.e., $e^{-d} \leq \delta$.

This analysis makes no assumption about the distribution of values in a. However, in many applications there are Zipfian, or power law, distributions of item frequencies. Here, the (relative) frequency of the ith most frequent item is proportional to i^{-z}, for some parameter z, where z is typically in the range 1–3 ($z = 0$ gives a perfectly uniform distribution). In such cases, the skew in the distribution can be used to show a stronger space/accuracy tradeoff:

Theorem 2 (Theorem 5.1 from [7]). *For a Zipf distribution with parameter z, the space required to* *answer point queries with error $\epsilon\|a\|_1$ with probability at least $1-\delta$ is given by $O\,(\varepsilon^{-\min\{1,1/z\}}\ln 1/\delta)$.*

Moreover, the dependency of the space on z is optimal:

Theorem 3 (Theorem 5.2 from [7]). *The space required to answer point queries correctly with any constant probability and error at most $\epsilon\|a\|_1$ is $\Omega(\varepsilon^{-1})$ over general distributions, and $\Omega(\varepsilon^{-1/z})$ for Zipf distributions with parameter z, assuming the dimension of a, n is $\Omega(\varepsilon^{-\min\{1,1/z\}})$.*

Range, Heavy Hitter and Quantile Queries

A range query is to estimate $\sum_{i=l}^{r} a_i$ for a range $[l...r]$. For small ranges, the range sum can be estimated as a sum of point queries; however, as the range grows, the error in this approach also grows linearly. Instead, log$n$ sketches can be kept, each of which summarizes a derived vector a^k where

$$a^k[j] = \sum_{i=j2^k}^{(j+1)2^k-1} a_i$$

for $k = 1...\log n$. A range of the form $j2^k...(j+1)2^k-1$ is called a *dyadic range*, and any arbitrary range $[l...r]$ can be partitioned into at most 2log n dyadic ranges. With appropriate rescaling of accuracy bounds, it follows that:

Theorem 4 (Theorem 4 from [5]). *Count-Min sketches can be used to find an estimate $\hat{r}$ for a range query on l...r such that*

$$\hat{r} - \epsilon\|a\|_1 \leq \sum_{i=l}^{r} a_i \leq \hat{r}$$

The right inequality holds with certainty, and the left inequality holds with probability at least $1-\delta$. The total space required is $O(\frac{\log^2 n}{\epsilon}\log\frac{1}{\delta})$.

Closely related to the range query is the *ϕ-quantile query*, which is to find a point j such that

$$\sum_{i=1}^{j} a_i \leq \phi\|a\|_1 \leq \sum_{i=1}^{j+1} a_i.$$

A natural approach is to use range queries to binary search for a j which satisfies this requirement approximately (i.e., tolerates up to $\epsilon\|a\|_1$ error in the above expression) given ϕ. In order to give the desired guarantees, the error bounds need to be adjusted to account for the number of queries that will be made.

Theorem 5 (Theorem 5 from [5]). *ε-approximate ϕ-quantiles can be found with probability at least $1-\delta$ by keeping a data structure with space $O\left(\frac{1}{\epsilon}\log^2(n)\log\left(\frac{\log n}{\delta}\right)\right)$. The time for each insert or delete operation is $O\left(\log(n)\log\left(\frac{\log n}{\delta}\right)\right)$, and the time to find each quantile on demand is $O\left(\log(n)\log\left(\frac{\log n}{\delta}\right)\right)$.*

Heavy Hitters are those points i such that $a_i \geq \phi\|a\|_1$ for some specified ϕ. The range query primitive based on Count-Min sketches can again be used to find heavy hitters, by recursively splitting dyadic ranges into two and querying each half to see if the range is still heavy, until a range of a single, heavy, item is found. Formally,

Theorem 6 (Theorem 6 from [5]). *Using space $O\left(\frac{1}{\epsilon}\log(n)\log\left(\frac{2\log(n)}{\delta\phi}\right)\right)$, and time $O\left(\log(n)\log\left(\frac{2\log n}{\delta\phi}\right)\right)$ per update, a set of approximate heavy hitters can be output so that every item with frequency at least $(\phi+\epsilon)\|a\|_1$ is output, and with probabilitye $1-\delta$ no item whose frequency is less than $\phi\|a\|_1$ is output.*

For skewed Zipfian distributions, as described above, with parameter $z > 1$, it is shown more strongly that the top-k most frequent items can be found with relative error ε using space only $\tilde{O}(\frac{k}{\epsilon})$ [7].

Inner Product Queries

The Count-Min sketch can also be used to estimate the inner product between two vectors; in database terms, this captures the (equi)join size between relations. The inner product $a \cdot b$, can be estimated by treating the Count-Min sketch as a collection of d vectors of length w, and finding the minimum inner product between corresponding rows of sketches of the two vectors. With probability $1-\delta$, this estimate is at most an additive quantity $\epsilon\|a\|_1\|b\|_1$ above the true value of $a \cdot b$. This is to be compared with AMS sketches which guarantee $\epsilon\|a\|_2\|b\|_2$ additive error, but require space proportional to $\frac{1}{\epsilon^2}$ to make this guarantee.

Interpretation as Random Linear Projection

The sketch can also be interpreted as a collection of inner-products between a vector representing the input and a collection of random vectors defined by the hash functions. Let a denote the vector representing the input, so that $a[i]$ is the sum of the updates to the ith location in the input. Let $r_{j,k}$ be the binary vector such that $r_{j,k}[i] = 1$ if and only if $h_j(i) = k$. Then it follows that $CM[j, k] = a \cdot r_{j,k}$. Because of this linearity, it follows immediately that if sketches of two vectors, a and b, are built then (i) the sketch of $a + b$ (using the same w,d,h_j) is the (componentwise) sum of the sketches (ii) the sketch of λa for any scalar λ is λ times the sketch of a. In other words, the sketch of any linear combination of vectors can be found. This property is useful in many applications which use sketches. For example, it allows distributed measurements to be taken, sketched, and combined by only sending sketches instead of the whole data.

Conservative Update

If only positive updates arrive, then an alternate update methodology may be applied, known as conservative update (due to Estan and Varghese [8]). For an update (i,c), $\hat{a}_i$ is computed, and the counts are modified according to $\forall 1 \leq j \leq d : CM[j, h_j(i)] \leftarrow \max(CM[j, h_j(i)], \hat{a}_i + c)$. It can be verified that procedure still ensures for point queries that $a_i \leq \hat{a}_i$, and that the error is no worse than in the normal update procedure; it is remarked that this can improve accuracy "up to an order of magnitude" [8]. Note however that deletions or negative updates can no longer be processed, and the additional processing that must be performed for each update could effectively halve the throughput.

Key Applications

Since its description and initial analysis, the Count-Min Sketch has been applied in a wide variety of situations. Here is a list of some of the ways in which it has been used or modified.

- Lee et al. [13] propose using least-squares optimization to produce estimates from Count-Min Sketches for point queries (instead of returning the minimum of locations where the item was mapped). It was shown that this approach can give significantly improved estimates, although at the cost of solving a convex optimization problem over n variables (where n is the size of the domain from which items are drawn, typically 2^{32} or higher).
- The "skipping" technique, proposed by Bhattacharrya et al. [2] entails avoiding adding items to the sketch (and saving the cost of the hash function computations) when this will not affect the accuracy too much, in order to further increase throughout in high-demand settings.
- Indyk [9] uses the Count-Min Sketch to estimate the residual mass after removing a subset of items. That is, given a (small) set of indices I, to estimate $\sum_{i \notin I} a_i$. This is needed in order to find clusterings of streaming data.
- The *entropy* of a data stream is a function of the relative frequencies of each item or character within the stream. Using Count-Min Sketches within a larger data structure based on additional hashing techniques, Lakshminath and Ganguly [12] showed how to estimate this entropy to within relative error.
- Sarlós et al. [17] gave approximate algorithms for personalized page rank computations which make use of Count-Min Sketches to compactly represent web-size graphs.
- In describing a system for building selectivity estimates for complex queries, Spiegel and Polyzotis [18] use Count-Min Sketches in order to allow clustering over a high-dimensional space.
- Rusu and Dobra [16] study a variety of sketches for the problem of inner-product estimation, and conclude that Count-Min sketch has a tendency to outperform its theoretical worst-case bounds by a considerable margin, and gives better results than some other sketches for this problem.
- Many applications call for tracking *distinct* counts: that is, a_i should represent the number of distinct updates to position i. This can be achieved by replacing the counters in the Count-Min sketch with approximate Count-Distinct summaries, such as the *Flajolet-Martin sketch*. This is described and evaluated in [6,10].
- Privacy preserving computations ensure that multiple parties can cooperate to compute a function of their data while only learning the answer and not anything about the inputs of the other participants. Roughan and Zhang demonstrate that the Count-Min Sketch can be used within such computations, by applying standard techniques for computing privacy preserving sums on each counter independently [15].

Related ideas to the Count-Min Sketch have also been combined with group testing to solve problems in the realm of Compressed Sensing, and finding significant changes in dynamic streams.

Future Directions

As is clear from the range of variety of applications described above, Count-Min sketch is a versatile data structure which is finding applications within Data Stream systems, but also in Sensor Networks, Matrix Algorithms, Computational Geometry and Privacy-Preserving Computations. It is helpful to think of the structure as a basic primitive which can be applied wherever approximate entries from high dimensional vectors or multisets are required, and one-sided error proportional to a small fraction of the total mass can be tolerated (just as a Bloom filter should be considered in order to represent a set wherever a list or set is used and space is at a premium). With this in mind, further applications of this synopsis can be expected to be seen in more settings.

As noted below, sample implementations are freely available in a variety of languages, and integration into standard libraries will further widen the availability of the structure. Further, since many of the applications are within high-speed data stream monitoring, it is natural to look to hardware implementations of the sketch. In particular, it will be of interest to understand how modern multi-core architectures can take advantage of the natural parallelism inherent in the Count-Min Sketch (since each of the d rows are essentially independent), and to explore the implementation choices that follow.

Experimental Results

Experiments performed in [7] analyzed the error for point queries and F_2 (self-join size) estimation, in comparison to other sketches. High accuracy was observed for both queries, for sketches ranging from a few kilobytes to a megabyte in size. The typical parameters of the sketch were a depth d of 5, and a width w of a few hundred to thousands. Implementations on desktop machines achieved between and two and three million updates per second. Other implementation have incorporated Count-Min Sketch into high speed streaming systems such as Gigascope [4], and tuned it to process packet streams of multi-gigabit speeds.

Lai and Byrd report on an implementation of Count-Min sketches on a low-power stream processor [18], capable of processing 40 byte packets at a throughput rate of up to 13 Gbps. This is equivalent to about 44 million updates per second.

URL To Code

Several example implementations of the Count-Min sketch are available. C code is given by the MassDal code bank: http://www.cs.rutgers.edu/~muthu/massdal-code-index.html. C++ code due to Marios Hadjieleftheriou is available from http://research.att.com/~marioh/sketches/index.html.

Cross-references

- ▶ AMS Sketch
- ▶ Data sketch/synopsis
- ▶ FM Synopsis
- ▶ Frequent items on streams
- ▶ Quantiles on streams

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. In Proc. 28th Annual ACM Symp. on Theory of Computing, 1996, pp. 20–29. Journal version in J. Comput. Syst. Sci., 58:137–147, 1999.
2. Bhattacharrya S., Madeira A., Muthukrishnan S., and Ye T. How to scalably skip past streams. In Proc. 1st Int. Workshop on Scalable Stream Processing Syst., 2007, pp. 654–663.
3. Charikar M., Chen K., and Farach-Colton M. Finding frequent items in data streams. In 29th Int. Colloquium on Automata, Languages, and Programming, 2002, pp. 693–703.
4. Cormode G., Korn F., Muthukrishnan S., Johnson T., Spatscheck O., and Srivastava D. Holistic UDAFs at streaming speeds. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 35–46.
5. Cormode G. and Muthukrishnan S. An improved data stream summary: the count-min sketch and its applications. J. Algorithms, 55(1):58–75, 2005.
6. Cormode G. and Muthukrishnan S. Space efficient mining of multigraph streams. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 271–282.
7. Cormode G. and Muthukrishnan S. Summarizing and mining skewed data streams. In Proc. SIAM International Conference on Data Mining, 2005.
8. Estan C. and Varghese G. New directions in traffic measurement and accounting. In Proc. ACM Int. Conf. of the on Data Communication, 2002, pp. 323–338.
9. Indyk P. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In Proceedings of ACM-SIAM Symposium on Discrete Algorithms, 2003.
10. Kollios G., Byers J., Considine J., Hadjieleftheriou M., and Li F. Robust aggregation in sensor networks. Q. Bull. IEEE TC on Data Engineering, 28(1):26–32, 2005.
11. Lai Y.-K. and Byrd G.T. High-throughput sketch update on a low-power stream processor. In Proc. ACM/IEEE Symp. on Architecture for Networking and Communications Systems, 2006, pp. 123–132.

12. Lakshminath B. and Ganguly S. Estimating entropy over data streams. In Proc. 14th European Symposium on Algorithms, 2006, pp. 148–159.
13. Lee G.M., Liu H., Yoon Y., and Zhang Y. Improving sketch reconstruction accuracy using linear least squares method. In Proc. 5th ACM SIGCOMM Conf. on Internet Measurement, 2005, pp. 273–278.
14. Motwani R. and Raghavan P. Randomized Algorithms. Cambridge University Press, 1995.
15. Roughan M. and Zhang Y. Secure distributed data mining and its application in large-scale network measurements. Computer Communication Review, 36(1):7–14, 2006.
16. Rusu F. and Dobra A. Statistical analysis of sketch estimators. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 187–198.
17. Sarlós T., Benzúr A., Csalogány K., Fogaras D., and Rácz B. To randomize or not to randomize: space optimal summaries for hyperlink analysis. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 297–306.
18. Spiegel J. and Polyzotis N. Graph-based synopses for relational selectivity estimation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 205–216.

Coupling and De-coupling

SERGUEI MANKOVSKII
CA Labs, CA, Inc., Thornhill, ON, Canada

Definition

Coupling is a measure of dependence between components of software system.

De-coupling is a design or re-engineering activity aiming to reduce coupling between system elements.

Key Points

Coupling of system components refers to a measure of dependency among them. Coupled components might depend on each other in different ways. Some examples of the dependencies are:

- One component might depend on syntax, format, or encoding of data produced by another component.
- One component might depend on the execution time within another component.
- One component might depend on state of another component.

Notion of coupling is connected to notion of cohesion. Cohesion is a measure of how related and focused are responsibilities of a software component. For example a highly cohesive component might group responsibilities

- using the same syntax, format or encoding of data.
- performed at the same time.
- executed in the same state.

Highly cohesive components lead to fewer dependencies between components and voice versa.

Notions of coupling and cohesion were studied in structured and object oriented programming. The research developed software tools to calculate coupling and cohesion metrics.

Low coupling is often desirable because it leads to reliability, easy of modification, low maintenance costs, understandability, and reusability. Low coupling can be achieved by deliberately designing system with low values of coupling metric. It can also be achieved by re-engineering of existing software system through restructuring of system into a set of more cohesive components. These activates are called de-coupling.

Cross-references

► Cohesion
► OODB (Object-Oriented Database)
► Structured Programming

Coverage

► Specificity

Covering Index

DONGHUI ZHANG
Northeastern University, Boston, MA, USA

Definition

Given an SQL query, a covering index is a composite index that includes all of the columns referenced in SELECT, JOIN, and WHERE clauses of this query. Because the index contains all the data needed by the query, to execute the query the actual data in the table does not need to be accessed.

Key Points

Covering indexes [1] support index-only execution plans. In general, having everything indexed tends to increase the query performance (in number of I/Os). However, using a covering index with too many columns can actually degrade performance. Typically, multi-dimensional index structures, e.g., the R-tree, perform poorer than linear scan with high dimensions. Some guidelines of creating a covering index are: (i) Create a covering index on frequently used queries. There are overheads in creating a covering index, which is often more significant than creating a regular index with fewer columns. Hence, if a query is seldom used, the overhead to create a covering index on it is not substantiated. This corresponds to Amdahl's law: improve the "interesting" part to receive maximum overall benefit of a system. (ii) Try to build a covering index by expanding an existing index. For instance, if there already exists an index on "age" and "salary," and one needs a covering index on "age," "salary," and "income," it is often better to expand the existing index rather than building a new index, which would share two columns with the existing index.

The term "covering index" is sometimes used to mean the collection of single-column, non-clustered indexes on all the columns in a table. This is due to the "index intersection" technique incorporated into the Microsoft SQL Server's query optimizer [1]. In particular, the query optimizer can build, at run time, a hash-based "covering index" to speedup queries on a frequently used table. This covering index is really a hash table, which is built based on multiple existing indexes. Creating single-column indexes on all columns encourages the query optimizer to perform index intersection, i.e., to build dynamic covering indexes.

Cross-references

▶ Access Methods
▶ Indexing

Recommended Reading

1. McGehee B. Tips on Optimizing Covering Indexes. http://www.sql-server-performance.com/tips/covering_indexes_p1.aspx, 2007.

Covert Communication

▶ Steganography

CPU Cache

▶ Processor Cache

Crabbing

▶ B-Tree Locking

Crash Recovery

Theo Härder
University of Kaiserslautern, Kaiserslautern, Germany

Synonyms

Failure handling; System recovery; Media recovery; Online recovery; Restart processing; Backward recovery

Definition

In contrast to transaction aborts, a crash is typically a major failure by which the state of the current database is lost or parts of storage media are unrecoverable (destroyed). Based on log data from a stable log, also called temporary log file, and the inconsistent and/or outdated state of the permanent database, system recovery has to reconstruct the most recent transaction-consistent database state. Because DBMS restart may take too long to be masked for the user, a denial of service can be observed. Recovery from media failures relies on the availability of (several) backup or archive copies of earlier DB states – organized according to the generation principle – and archive logs (often duplexed) covering the processing intervals from the points of time the backup copies were created. Archive recovery usually causes much longer outages than system recovery.

Historical Background

Log data delivering the needed redundancy to recover from failures was initially stored on nonvolatile core memory to be reclaimed at restart by a so-called log salvager [3] in the "pre-transaction area". Advances in VLSI technology enabled the use of cheaper and larger, but volatile semiconductor memory as the computers' main memory. This technology change triggered by 1971 in industry – driven by database product adjustments – the development of new and refined concepts of logging such as log sequence numbers (LSNs), write-ahead log protocol (WAL), log duplexing and more. Typically, these concepts were not published, nevertheless they paved the way towards the use of ACID transactions. As late as 1978, Jim Gray documented the design of such a logging system implemented in IMS in a widely referenced publication [5].

Many situations and dependencies related to failures and recovery from those in databases have been thoroughly explored by Lawrence Bjork and Charles Davies in their studies concerning DB/DC systems back in 1973 leading to the so-called "spheres of control" [2]. The first published implementation of the transaction concept by a full-fledged DBMS recovery manager was that of System R, started in 1976 [4]. It refined the Do-Undo-Redo protocol and enabled automatic recovery for new recoverable types and operations. In 1981, Andreas Reuter presented in his Ph.D. dissertation further investigations and refinements of concepts related to failure handling in database systems [9]. Delivering a first version of the principles of transaction-oriented database recovery [Härder and Reuter 1979], including the *Ten Commandments* [6], this classification framework, defining the paradigm of transaction-oriented recovery and coining the acronym ACID for it [7], was finally published in 1983. The most famous and most complete description of recovery methods and their implementation was presented by C. Mohan et al. in the ARIES paper [8] in 1992, while thorough treatment of all questions related to this topic appeared in many textbooks, especially those of Bernstein et al. [1], Gray and Reuter [3], and Weikum and Vossen [11]. All solutions implemented for crash recovery in industrial-strength DBMSs are primarily disk-based. Proposals to use "safe RAM", for example, were not widely accepted.

Foundations

The most difficult failure type to be recovered from is the system failure or system crash (see Logging and Recovery). Due to some (expected, but) unplanned failure event (a bug in the DBMS code, an operating system fault, a power or hardware failure, etc.), the *current database* – comprising all objects accessible to the DBMS during normal processing – is not available anymore. In particular, the in-memory state of the DBMS (lock tables, cursors and scan indicators, status of all active transactions, etc.) and the contents of the database buffer and the log buffer are lost. Furthermore, the state lost may include information about LSNs, ongoing commit processing with participating coordinators and participants as well as commit requests and votes. Therefore, restart cannot rely on such information and has to refer to the temporary log file (stable log) and the *permanent (materialized) database*, that is, the state the DBMS finds after a crash at the non-volatile storage devices (disks) without having applied any log information.

Consistency Concerns

According to the *ACID principle*, a database is consistent if and only if it contains the results of successful transactions – called transaction-consistent database. Because a DBMS application must not lose changes of committed transactions and all of them have contributed to the DB state, the goal of crash recovery is to establish the most recent transaction-consistent DB state. For this purpose, redo and undo recovery is needed, in general. Results of committed transactions may not yet be reflected in the database, because execution has been terminated in an uncontrolled manner and the corresponding pages containing such results were not propagated to the permanent DB at the time of the crash. Therefore, they must be repeated, if necessary – typically by means of log information. On the other hand, changes of incomplete transactions may have reached the permanent DB state on disk. Hence, undo recovery has to completely roll back such uncommitted changes.

Because usually many interactive users rely in their daily business on DBMS services, crash recovery is very time-critical. Therefore, crash-related interruption of DBMS processing should be masked for them as far as possible. Although today DBMS crashes are rather rare events and may occur several times a month or a year – depending on the stability of both the DBMS and its operational environment – , their recovery should take no more than a number of seconds or at most a few minutes (as opposed to archive recovery), even if GByte or TByte databases with thousands of users are involved.

Forward Recovery

Having these constraints and requirements in mind, which kind of recovery strategies can be applied? Despite the presence of so-called non-stop systems (giving the impression that they can cope with failures by forward recovery), rollforward is very difficult, if not impossible in any stateful system. To guarantee atomicity in case of a crash, rollforward recovery had to enable all transactions to resume execution so that they can either complete successfully or require to be aborted by the DBMS. Assume the DB state containing the most recent successful DB operations could be made available, that is, all updates prior to the crash have completely reached the permanent DB state. Even then rollforward would be not possible, because a transaction cannot resume in "forward direction" unless its local state is restored. Moreover in a DBMS environment, the in-memory state lost makes it entirely impossible to resume from the point at the time the crash occurred. For these reasons, a rollback strategy for active transactions is the only choice in case of crash recovery to ensure atomicity (wiping out all traces of such transactions); later on these transactions are started anew either by the user or the DBMS environment. The only opportunities for forward actions are given by redundant structures where it is immaterial for the logical DB content whether or not modifying operations are undone or completed. A typical example is the splitting operation of a B-tree node.

Logging Methods and Rules

Crash recovery – as any recovery from a failure – needs some kind of redundancy to detect invalid or missing data in the permanent database and to "repair" its state as required, i.e., removing modifications effected by uncommitted transactions from it and supplementing it with updates of complete transactions. For this task, the recovery algorithms typically rely on log data collected during normal processing. Different forms of logging are conceivable. *Logical logging* is a kind of operator logging; it collects operators and their arguments at a higher level of abstraction (e.g., for internal operations (actions) or operations of the data management language (DML)). While this method of logging may save I/O to and space in the log file during normal processing, it requires at restart time a DB state that is level-consistent w.r.t. the level of abstraction used for logging, because the logged operations have to be executed using data of the permanent database. For example, action logging and DML-operation logging require action consistency and consistency at the application programming interface (API consistency), respectively [6]. Hence, the use of this kind of methods implies the atomic propagation (see below) of all pages modified by the corresponding operation which can be implemented by shadow pages or differential files. *Physical logging* – in the simplest form collecting the before- and after-images of pages – does not expect any form of consistency at higher DB abstraction levels and, in turn, can be used in any situation, in particular, when non-atomic propagation of modified pages (*update-in-place*) is performed. However, writing before- and after-images of all modified pages to the log file, is very time-consuming (I/O) and not space-economical at all. Therefore, a combination of both kinds leads to the so-called *physiological logging*, which can be roughly characterized as "physical to a page and logical within a page". It enables compact representation of log data (logging of elementary actions confined to single pages, entry logging) and leads to the practically most important logging/recovery method; non-atomic propagation of pages to disk is sufficient for the application of the log data. Together with the use of *log sequence numbers* in the log entries and in the headers of the data pages (combined use of LSNs and PageLSNs, see ARIES Protocol), simple and efficient checks at restart detect whether or not the modifications of elementary actions have reached the permanent database, that is, whether or not undo or redo operations have to be applied.

While, in principle, crash recovery methods do not have specific requirements for forcing pages to the permanent DB, sufficient log information, however, must have reached the stable log. The following rules (for forcing of the log buffer to disk) have to be observed to guarantee recovery to the most recent transaction-consistent DB state:

- Redo log information must be written at the latest in phase 1 of commit.
- WAL (*write ahead logging*) has to be applied to enable undo operations, before uncommitted (dirty) data is propagated to the permanent database.
- Log information must not be discarded from the temporary log file, unless it is guaranteed that it will no longer be needed for recovery; that is, the corresponding data page has reached the permanent DB. Typically, sufficient log information has been written to the archive log, in addition.

Taxonomy of Crash Recovery Algorithms

Forcing log data as captured by these rules yields the necessary and sufficient condition to successfully cope with system crashes. Specific assumptions concerning page propagation to the permanent database only influence performance issues of the recovery process.

When dirty data can reach the permanent DB (steal property), recovery must be prepared to execute undo steps and, in turn, redo steps when data modified by a transaction is not forced at commit or before (no-force property). In contrast, if propagation of dirty data is prevented (no-steal property), the permanent DB only contains clean (but potentially missing or old) data, thus making undo steps unnecessary. Finally, if all transaction modifications are forced at commit (force property), redo is never needed at restart.

Hence, these properties concerning buffer replacement and update propagation are maintained by the buffer manager/transaction manager during normal processing and lead to four cases of crash recovery algorithms which cover all approaches so far proposed:

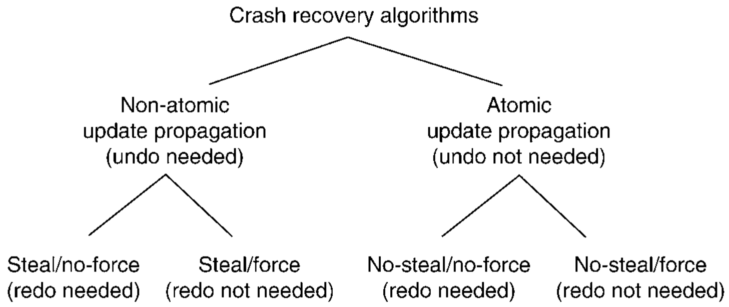
1. *Undo/Redo:* This class contains the *steal/no-force* algorithms which have to observe no other requirements than the logging rules. However, potentially undo and redo steps have to be performed during restart after a crash.
2. *Undo/NoRedo:* The so-called *steal/force* algorithms guarantee at any time that all actions of committed transactions are in the permanent DB. However, because of the steal property, dirty updates may be present, which may require undo steps, but never redo steps during restart.
3. *NoUndo/Redo:* The corresponding class members are known as *no-steal/no-force* algorithms which guarantee that dirty data never reaches the permanent DB. Dirty data pages are either never replaced from the DB buffer or, in case buffer space is in short supply, they are displaced to other storage areas outside the permanent DB. Restart after a crash may require redo steps, but never undo steps.
4. *NoUndo/NoRedo:* This "magic" class of the so-called *no-steal/force* algorithms always guarantees a state of the permanent DB that corresponds to the most recent transaction-consistent DB state. It requires that no modified data of a transaction reaches the permanent DB before commit and that all transaction updates are atomically propagated (forced) at commit. Hence, neither undo nor redo steps are ever needed during restart.

The discussion of these four cases is summarized in Fig. 1 which represents a taxonomy of crash recovery algorithms.

Implementation Implications

The latter two classes of algorithms (NoUndo) require a mechanism which can propagate a set of pages in an atomic way (with regard to the remaining DBMS processing). Such a mechanism needs to defer updates to the permanent DB until or after these updates become committed and can be implemented by various forms of shadowing concepts or differential file approaches.

Algorithms relying on redo steps, i.e., without the need to force committed updates to the permanent DB, have no control about the point of time when committed updates reach the permanent DB. While the buffer manager will propagate back most of the modified pages soon after the related update operations, a few hot-spot pages are modified again and again, and, since they are referenced so frequently, have not been written from the buffer. These pages potentially have accumulated the updates of many committed transactions, and redo recovery will therefore have to go back very far on the temporary log. As a consequence, restart becomes expensive and the DBMS's out-of-service time unacceptably long. For this reason, some form of *checkpointing* is needed to make restart costs independent of mean time between failures. Generating a checkpoint means collecting


Crash recovery algorithms
Non-atomic update propagation (undo needed)
Atomic update propagation (undo not needed)
Steal/no-force (redo needed)
Steal/force (redo not needed)
No-steal/no-force (redo needed)
No-steal/force (redo not needed)

Crash Recovery. Figure 1. Taxonomy of crash recovery algorithms.

information related to the DB state in a safe place, which is used to define and limit the amount of redo steps required after a crash. The restart logic can then return to this checkpoint state and attempt to recover the most recent transaction-consistent state.

From a conceptual point of view, the algorithms of class 4 seem to be particularly attractive, because they always preserve a transaction-consistent permanent DB. However in addition to the substantial cost of providing atomic update propagation, the need of forcing all updates at commit, necessarily in a synchronous way which may require a large amount of physical I/Os and, in turn, extend the lock duration for all affected objects, makes this approach rather expensive. Furthermore, with the typical disk-based DB architectures, pages are units of update propagation, which has the consequence that a transaction updating a record in a page cannot share this page with other updaters, because dirty updates must not leave the buffer and updates of complete transactions must be propagated to the permanent DB at commit. Hence, no-steal/force algorithms imply at least *page locking* as the smallest lock granule.

One of these cost factors – either synchronously forced updates at commit or atomic updates for NoUndo – applies to the algorithms of class 2 and 3 each. Therefore, they were not a primary choice for the DBMS vendors competing in the today's market.

Hence, the laissez-faire solution "steal, no-force" with non-atomic update propagation (update-in-place) is today's favorite solution, although it always leaves the permanent DB in a "chaotic state" containing dirty and outdated data pages and keeping the latest version of frequently used pages only in the DB buffer. Hence, with the optimistic expectation that crashes become rather rare events, it minimizes recovery provisions during normal processing. Checkpointing is necessary, but the application of direct checkpoints flushing the entire buffer at a time, is not advisable anymore, when buffers of several GByte are used. To affect normal processing as little as possible, so-called *fuzzy checkpoints* are written; only a few pages with metadata concerning the DB buffer state have to be synchronously propagated, while data pages are "gently" moved to the permanent DB in an asynchronous way.

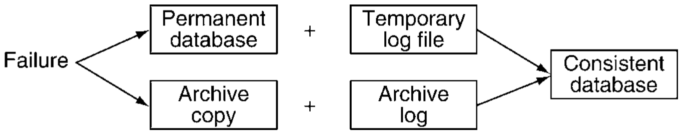
Archive Recovery

So far, data of the permanent DB was assumed to be usable or at least recoverable using the redundant data collected in the temporary log. This is illustrated by the upper path in Fig. 2. If any of the participating components is corrupted or lost because of other hardware or software failure, archive recovery – characterized by the lower path – must be tried. Successful recovery also implies independent failure modes of the components involved.

The creation of an archive copy, that is, copying the online version of the DB, is a very expensive process; for example, creating a transaction-consistent DB copy would interrupt update operation for a long time which is unacceptable for most DB applications. Therefore, two base methods – fuzzy dumping and incremental dumping – were developed to reduce the burden of normal DB operation while an archive copy is created. A fuzzy dump copies the DB on the fly in parallel with normal processing. The other method writes only the changed pages to the incremental dump. Of course, both methods usually deliver inconsistent DB copies such that log-based post-processing is needed to apply incremental modifications. In a similar way, either type of dump can be used to create a new, more up-to-date copy from the previous one, using a separate offline process such that DB operation is not affected.

Archive copies are "hopefully" never or very infrequently used. Therefore, they may be susceptible to magnetic decay. For this reason, redundancy is needed again, which is usually solved by keeping several generations of the archive copy.

So far, all log information was assumed to be written only to the temporary log file during normal processing. To create the (often duplexed) archive log, usually an independent and asynchronously running process copies the redo data from the temporary log. To guarantee successful recovery, failures when using



Crash Recovery. Figure 2. Two ways of DB crash recovery and the components involved.

the archive copies must be anticipated. Therefore, archive recovery must be prepared to start from the oldest generation and hence the archive log must span the whole distance back to this point in time.

Key Applications

Recovery algorithms, and in particular for crash recovery, are a core part of each commercial-strength DBMS and require a substantial fraction of design/implementation effort and of the code base: "A recoverable action is 30% harder and requires 20% more code than a non-recoverable action" (J. Gray). Because the occurrence of failures can not be excluded and all data driving the daily business are managed in databases, mission-critical businesses depend on the recoverability of their data. In this sense, provisions for crash recovery are indispensable in such DBMS-based applications. Another important application area of crash recovery techniques are file systems, in particular their metadata about file existence, space allocation, etc.

Future Directions

So far, crash recovery provisions are primarily disk-based. With "unlimited" memory available, main-memory DBMSs will provide efficient and robust solutions without the need of non-volatile storage for crash recovery. More and more approaches are expected to exploit specialized storage devices such as battery-backed RAM or to use replication in grid-organized memories. Executing online transaction processing sequentially, revolutionary architectural concepts are already proposed which may not require transactional facilities at all [10].

Cross-references

- ▶ ACID Properties
- ▶ Application Recovery
- ▶ B-Tree Locking
- ▶ Buffer Management
- ▶ Logging and Recovery
- ▶ Multi-Level Recovery and the ARIES Algorithm

Recommended Reading

1. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, MA, 1987.
2. Davies C.T. Data processing spheres of control. IBM Syst. J., 17(2):179–198, 1978.
3. Gray H. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1993.
4. Gray J., McJones P., Blasgen M., Lindsay B., Lorie R., Price T., Putzolu F., and Traiger I.L. The recovery manager of the System R database manager. ACM Comput. Surv., 13(2): 223–242, 1981.
5. Gray J, Michael J. Feynn, Jim Gray, Anita K. Jones, Klans Lagally, Holger Opderbeck, Gerald J. Popek, Brian Randell, Jerome H. Saltfer, Hans-Rüdiger Wiehle. Notes on database operating systems. In Operating Systems: An Advanced Course. Springer, LNCS 60, 1978, pp. 393–481.
6. Härder T. DBMS Architecture – Still an Open Problem. In Proc. German National Database Conference, 2005, pp. 2–28.
7. Härder T. and Reuter A. Principles of transaction-oriented database recovery. ACM Comput. Surv., 15(4): 287–317, 1983.
8. Mohan C., Haderle D.J., Lindsay B.G., Pirahesh H., and Schwarz P.M. ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. Database Syst., 17(1):94–162, 1992.
9. Reuter A. Fehlerbehandlung in Datenbanksystemen. Carl Hanser, Munich, 1981, p. 456.
10. Stonebraker M., Madden S., Abadi D.J., Harizopoulos S., Hachem N., and Helland P. The End of an Architectural Era (It's Time for a Complete Rewrite). In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 1150–1160.
11. Weikum G. and Vossen G. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2002.

Crawler

▶ Incremental Crawling

Credulous Reasoning

▶ Possible Answers

Cross Product

▶ Cartesian Product

Cross-language Cross-Language Mining and Retrieval C217 Informational Retrieval

▶ Cross-Language Mining and Retrieval

Cross-Language Mining and Retrieval

Wei Gao[1], Cheng Niu[2]
[1]The Chinese University of Hong Kong, Hong Kong, China
[2]Microsoft Research Asia, Beijing, China

Synonyms

Cross-language text mining; Cross-language web mining; Cross-language informational retrieval; Translingual information retrieval

Definition

Cross-language mining is a task of text mining dealing with the extraction of entities and their counterparts expressed in different languages. The interested entities may be of various granularities from acronyms, synonyms, cognates, proper names to comparable or parallel corpora. Cross-Language Information Retrieval (CLIR) is a sub-field of information retrieval dealing with the retrieval of documents across language boundaries, i.e., the language of the retrieved documents is not the same as the language of the queries. Cross-language mining usually acts as an effective means to improve the performance of CLIR by complementing the translation resources exploited by CLIR systems.

Historical Background

CLIR addresses the growing demand to access large volumes of documents across language barriers. Unlike monolingual information retrieval, CLIR requires query terms in one language to be matched with the indexed keywords in the documents of another language. Usually, the cross-language matching can be done by making use of bilingual dictionary, machine translation software, or statistical model for bilingual words association. CLIR generally takes into account but not limited to the issues like how to translate query terms, how to deal with the query terms nonexistent in a translation resource, and how to disambiguate or weight alternative translations (e.g., to decide that "traitement" in a French query means "treatment" but not "salary" in English, or how to order the French terms "aventure," "business," "affaire," and "liaison" as relevant translations of English query "affair"), etc. The performance of CLIR can be measured by the general evaluation metrics of information retrieval, such as recall precision, average precision, and mean reciprocal rank, etc.

The first workshop on CLIR was held in Zürich during the SIGIR-96 conference. Workshops have been held yearly since 2000 at the meetings of CLEF (Cross Language Evaluation Forum), following its predecessor workshops of TREC (Text Retrieval Conference) cross-language track. The NTCIR (NII Test Collection for IR Systems) workshop is also held each year in Japan for CLIR community focusing on English and Asian languages.

The study of cross-language mining appears relatively more lately than CLIR, partly due to the increasing demands on the quality of CLIR and machine translation, as well as the recent advancement of text/Web mining techniques. A typical early work on cross-lingual mining is believed to be PTMiner [14] that mines parallel text from the Web used for query translation. Other than parallel data mining, people also tried to mine the translations of Out-of-Vocabulary (OOV) terms from search results returned from search engine [5,18] or from web anchor texts and link structures [12]. Based on phonetic similarity, transliteration (the phonetic counterpart of a name in another language, e.g., "Schwarzenegger" is pronounced as "shi wa xin ge" in Chinese pinyin) of foreign names also could be extracted properly from the Web [10]. These methods are proposed to alleviate the OOV problem of CLIR since there is usually lack of appropriate translation resources for new terminologies and proper names, particularly in the scenario of cross-language web search.

Foundations

Most approaches to CLIR perform query translation followed by monolingual retrieval. So the retrieval performance is largely determined by the quality of query translation. Queries are typically translated either using a bilingual dictionary [15], a machine translation (MT) software [7], bilingual word association model learned from parallel corpus [6,14], or recently a query log of a search engine [9]. Despite the types of the resources being used, OOV translation and translation disambiguation are the two major bottlenecks for CLIR. On one hand, translation resources can never be comprehensive. Correctly translating queries, especially Web queries, is difficult since they often contain new words (e.g., new movies, brands, celebrities, etc.) occurring timely and frequently, yet being OOV to the system; On the other hand, many words are polysemous, or they do not have a unique translation, and sometimes the alternative translations have very different meanings. This is known as translation ambiguity. Selecting the correct translation is not trivial due to the shortage of context provided in a query, and effective techniques for translation disambiguation are necessary.

It should be mentioned that document translation with MT in the opposite direction is an alternative approach to CLIR. However, it is less commonly used than query translation in the literature mainly because MT is computationally expensive and costly to develop, and the document sets in IR are generally very large. For cross-language web search, it is almost impractical to translate all the web pages before indexing. Some large scale attempts to compare query translation and document translation have suggested no clear advantage for either of the approaches to CLIR [12]. But they found that compared with extremely high quality human query translations, it is advantageous to incorporate both document and query translation into a CLIR system.

Cross-Language Web Mining

Mining Parallel Data The approaches of mining parallel text make extensive use of bilingual websites where parallel web pages corresponding to the specified language pair can be identified and downloaded. Then the bilingual texts are automatically aligned in terms of sentences and words by statistical aligning tools, such as GIZA++ [21]. The word translation probabilities can be derived with the statistics of word pairs occurring in the alignments, after which one can resort to statistical machine translation models, e.g., IBM model-1 [4], for translating given queries into the target language. The typical parallel data mining tools include PTMiner [14], STRAND [16] and the DOM-tree-alignment-based system [17].

Mining OOV Term Translation Web pages also contain translations of terms in either the body texts or the anchor texts of hyper-links pointing to other pages. For example, in some language pairs, such as Chinese-English or Japanese-English, the Web contains rich body texts in a mixture of multiple languages. Many of them contain bilingual translations of proper nouns, such as company names and person names. The work of [5,16] exploits this nice characteristic to automatically extract translations from search result for a large number of unknown query terms. Using the extracted bilingual translations, the performance of CLIR between English and Chinese is effectively improved. Both methods select translations based on some variants of co-occurrence statistics.

The anchor text of web pages' hyperlinks is another source for translational knowledge acquisition. This is based on the observation that the anchor texts of hyperlinks pointing to the same URL may contain similar descriptive texts. Lu et al. [11] uses anchor text of different languages to extract the regional aliases of query terms for constructing a translation lexicon. A probabilistic inference model is exploited to estimate the similarity between query term and extracted translation candidates.

Query Translation Disambiguation

Translation disambiguation or ambiguity resolution is crucial to the query translation accuracy. Compared to the simple dictionary-based translation approach without addressing translation disambiguation, the effectiveness of CLIR can be 60% lower than that of monolingual retrieval [3]. Different disambiguation techniques have been developed using statistics obtained from document collections, all resulting in significant performance improvement. Zhang et al. [19] give concise review on three main translation disambiguation techniques. These methods include using term similarity [1], word co-occurrence statistics of the target language documents, and language modeling based approaches [20]. In this subsection, we introduce these approaches following the review of Zhang et al. [19].

Disambiguation by Term Similarity Adriani [1] proposed a disambiguation technique based on the concept of statistical term similarity. The term similarity is measured by the Dice coefficient, which uses the term-distribution statistics obtained from the corpus. The similarity between term *x* and *y*, *SIM(x, y)*, is calculated as:

$$SIM(x,y) = 2\sum_{i=1}^{n}(w_{xi}w_{yi}) \bigg/ \left(\sum_{i=1}^{n} w_{xi}^2 + \sum_{i=1}^{n} w_{yi}^2\right)$$

where w_{xi} and w_{yi} is the weights of term *x* and *y* in document *i*. This method computes the sum of maximum similarity values between each candidate translation of a term and the translations of all other terms in the query. For each query term, the translation with the highest sum is selected as its translation. The results of Indonesian-English CLIR experiments demonstrated the effectiveness of this approach. There are many variant term association measures like Jaccard, Cosine, Overlap, etc. that can be applied similarly for calculating their similarity.

Disambiguation by Term Co-occurrence Ballesteros and Croft [3] used co-occurrence statistics obtained from the target corpus for resolving disambiguation. They assume the correct translations of query terms should co-occur in target language documents and incorrect translations tend not to co-occur. Similar approach is studied by Gao et al. [8]. They observed that the correlation between two terms is stronger when the distance between them is shorter. They extended the previous co-occurrence model by incorporating a distance factor $\mathrm{D(x,y)} = \mathrm{e}^{-\alpha(Dis(x,y)-1)}$. The mutual information between term x and y, $MI(x, y)$, is calculated as:

$$MI(x,y) = \log\left(\frac{f_w(x,y)}{f_x f_y} + 1\right) \times D(x,y)$$

where $f_w(x, y)$ is the co-occurrence frequency of x and y that occur simultaneously within a window size of w in the collection, f_x is the collection frequency of x, and f_y is the collection frequency of y. $D(x, y)$ decreases exponentially when the distance between the two terms increases, where α is the decay rate, and $D(x, y)$ is the average distance between x and y in the collection. The experiments on the TREC9 Chinese collection showed that the distance factor leads to substantial improvements over the basic co-occurrence model.

Disambiguation by Language Modeling In the work of [20], a probability model based on hidden Markov model (HMM) is used to estimate the maximum likelihood of each sequence of possible translations of the original query. The highest probable translation set is selected among all the possible translation sets. HMM is a widely used for probabilistic modeling of sequence data. In their work, a smoothing technique based on absolute discounting and interpolation method is adopted to deal with the zero-frequency problem during probability estimation. See [20] for details.

Pre-/Post-Translation Expansion

Techniques of OOV term translation and translation disambiguation both aim to translate query correctly. However, it is arguable that precise translation may not be necessary for CLIR. Indeed, in many cases, it is helpful to introduce words even if they are not direct translations of any query word, but are closely related to the meaning of the query. This observation has led to the development of cross-lingual query expansion (CLQE) techniques [2,13]. [2] reported the enhancement on CLIR by post-translation expansion. [13] made performance comparison on various CLQE techniques, including pre-translation expansion, post-translation expansion and their combinations. Relevance feedback, the commonly used expansion technique in monolingual retrieval, is also widely adopted in CLQE. The basic idea is to expand original query by additional terms that are extracted from the relevant retrieval result initially returned. Amongst different relevance feedback methods, explicit feedback requires documents whose relevancy is explicitly marked by human; implicit feedback is inferred from users' behaviors that imply the relevancy of the selected documents, such as which returned documents are viewed or how long they view some of the documents; blind or "pseudo" relevance feedback is obtained by assuming that top n documents in the initial result are relevant.

Cross-Lingual Query Suggestion

Traditional query translation approaches rely on static knowledge and data resources, which cannot effectively reflect the quickly shifting interests of Web users. Moreover, the translated terms can be reasonable translations, but are not popularly used in the target language. For example, the French query "aliment biologique" is translated into "biologic food," yet the correct formulation nowadays should be "organic food." This mismatch makes the query translation in the target language ineffective. To address this problem, Gao et al. [9] proposed a principled framework called Cross-Lingual Query Suggestion (CLQS), which leverages cross-lingual mining and translation disambiguation techniques to suggest related queries found in the query log of a search engine.

CLQS aims to suggest related queries in a language different from the original query. CLQS is closely related to CLQE, but is distinct in that it suggests full queries that have been formulated by users so that the query integrity and coherence are preserved in the suggested queries. It is used as a new means of query "translation" in CLIR tasks. The use of query log for CLQS stems from the observation that in the same period of time, many search users share the same or similar interests, which can be expressed in different manners in different languages. As a result, a query written in a source language is possible to have an equivalent in the query log of the target language. Especially, if the user intends to perform CLIR, then original query is even more likely to have its correspondent included in the target language log. Therefore, if a candidate for CLQS appears often in the query log, it is more likely to be the appropriate one to suggest. CLQS is testified being able to cover more relevant documents for the CLIR task.

The key problem with CLQS is how to learn a similarity measure between two queries in different languages. They define cross-lingual query similarity based on both translation relation and monolingual similarity. The principle for learning is, for a pair of queries, their cross-lingual similarity should fit the monolingual similarity between one query and the other query's translation. There are many ways to obtain a monolingual similarity between queries, e.g., co-occurrence based mutual information and χ^2. Any of them can be used as the target for the cross-lingual similarity function to fit. In this way, cross-lingual query similarity estimation is formulated as a regression task:

$$sim_{CL}(q_f, q_e) = w \cdot \varphi(f(q_f, q_e)) = sim_{ML}(T_{q_f}, q_e)$$

where given a source language query q_f, a target language query q_e, and a monolingual query similarity between them sim_{ML}, the cross-lingual query similarity sim_{CL} can be calculated as an inner product between a weight vector and the feature vector in the kernel space, and φ is the mapping from the input feature space onto the kernel space, and w is the weight vector which can be learned by support vector regression training. The monolingual similarity is measured by combining both query content-based similarity and click-through commonality in the query log.

This discriminative modeling framework can integrate arbitrary information sources to achieve an optimal performance. Multiple feature functions can be incorporated easily into the framework based on different translation resources, such as bilingual dictionaries, parallel data, web data, and query logs. They work uses co-occurrence-based dictionary translation disambiguation, IBM translation model-1 based on parallel corpus, and Web-based query translation mining as means to discover related candidate queries in the query log. Experiments on TREC6 French-English CLIR task demonstrate that CLQS-based CLIR is significantly better than the traditional dictionary-based query translation with disambiguation and machine translation approaches.

Latent Semantic Index (LSI) for CLIR

Different from most of the alternative approaches discussed above, LSI for CLIR [6] provides a method for matching text segments in one language with the segments of similar meaning in another language without having to translate either. Using a parallel corpus, LSI can create a language-independent representation of words. The representation matrix reflects the patterns of term correspondences in the documents of two languages. The matrix is factorized by Singular Value Decomposition (SVD) for deriving a latent semantic space with a reduced dimension, where similar terms are represented by similar vectors. In latent semantic space, therefore, the monolingual similarity between synonymous terms from one language and the cross-lingual similarity between translation pairs from different languages tend to be higher than the similarity with irrelevant terms. This characteristic allows relevant documents to be retrieved even if they do not share any terms in common with the query, which makes LSI suitable for CLIR.

Key Applications

Cross-language mining and retrieval is the foundation technology for searching web information across multiple languages. It can also provide the cross-lingual functionality for the retrieval of structured, semi-structured and un-structured document databases of specific domains or in large multinational enterprises.

Experimental Results

In general, for every presented work, there is an accompanying experimental evaluation in the corresponding reference. Especially, the three influential international workshops held annually, i.e., CLEF, NTCIR and TREC, defines many evaluation tasks for CLIR, and there are a large number of experimental results being published based on these benchmark specifications.

Data Sets

Data sets for benchmarking CLIR are released to the participants of TREC, CLEF and NTCIR workshops annually with license agreements.

Cross-references

▶ Anchor Text
▶ Average Precision
▶ Document databases
▶ Document Links and Hyperlinks
▶ Evaluation Metrics for Structured Text Retrieval
▶ Information Extraction
▶ Information Retrieval
▶ MAP
▶ MRR
▶ Query Expansion for Information Retreival
▶ Query Translation
▶ Relevance Feedback
▶ Singular Value Decomposition
▶ Snippet
▶ Stemming
▶ Stoplists
▶ Term Statistics for Structuerd Text Retrieval
▶ Term Weighting
▶ Text Indexing and Retrieval
▶ Text Mining
▶ Web Information Extraction
▶ Web Search Relevance Feedback

Recommended Reading

1. Adriani M. Using statistical term similarity for sense disambiguation in cross-language information retrieval. Inform. Retr., 2(1):71–82, 2000.
2. Ballestors L.A. and Croft W.B. Phrasal translation and query expansion techniques for cross-language information retrieval. In Proc. 20th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1997, pp. 84–91.
3. Ballestors L.A. and Croft W.B. Resolving ambiguity for cross-language information retrieval. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 64–71.
4. Brown P.F., Pietra S.A.D., Pietra V.D.J., and Mercer R.L. The mathematics of machine translation: parameter estimation. Comput. Linguist., 19:263–312, 1992.
5. Cheng P.-J., Teng J.-W., Chen R.-C., Wang J.-H., Lu W.-H., and Chien L.-F. Translating unknown queries with Web corpora for cross-language information retrieval. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 146–153.
6. Dumais S.T., Landauer T.K., and Littman M.L. Automatic cross-linguistic information retrieval using latent semantic indexing. ACM SIGIR Workshop on Cross-Linguistic Information Retrieval, 1996, pp. 16–23.
7. Fujii A. and Ishikawa T. Applying machine translation to two-stage cross-language information retrieval. In Proc. 4th Conf. Association for Machine Translation in the Americas, 2000, pp. 13–24.
8. Gao J., Zhou M., Nie, J.-Y., He H., and Chen W. Resolving query translation ambiguity using a decaying co-occurrence model and syntactic dependence relations. In Proc. 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2002, pp. 183–190.
9. Gao W., Niu C., Nie J.-Y., Zhou M., Hu J., Wong K.-F., and Hon H.-W.: Cross-lingual query suggestion using query logs of different languages. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 463–470.
10. Jiang L., Zhou M., Chien L.-F., and Niu C. Named entity translation with Web mining and transliteration. In Proc. 20th Int. Joint Conf. on AI, 2007, pp. 1629–1634.
11. Lu W.-H., Chien L.-F., and Lee H.-J. Translation of Web queries using anchor text mining. ACM Trans. Asian Lang. Information Proc., 1(2):159–172, 2002.
12. McCarley J.S. Should we translate the documents or the queries in cross-language information retrieval? In Proc. 27th Annual Meeting of the Assoc. for Computational Linguistics, 1999, pp. 208–214.
13. McNamee P. and Mayfield J. Comparing cross-language query expansion techniques by degrading translation resources. In Proc. 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2002, pp. 159–166.
14. Nie J.-Y., Smard M., Isabelle P., and Durand R. Cross-language information retrieval based on parallel text and automatic mining of parallel text from the Web. In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 74–81.
15. Pirkola A., Hedlund T., Keshusalo H., and Järvelin K. Dictionary-based cross-language information retrieval: problems, methods, and research findings. Inform. Retr., 3(3–4): 209–230, 2001.
16. Resnik P. and Smith N.A. The Web as a parallel corpus. Comput. Linguist., 29(3):349–380, 2003.
17. Shi L., Niu C., Zhou M., and Gao J. A DOM Tree alignment model for mining parallel data from the Web. In Proc. 44th Annual Meeting of the Assoc. for Computational Linguistics, 2006, pp. 489–496.
18. Zhang Y. and Vines P. Using the Web for automated translation extraction in cross-language information retrieval. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 162–169.
19. Zhang Y., Vines P., and Zobel J. An empirical comparison of translation disambiguation techniques for Chinese-English Cross-Language Information Retrieval. In Proc. 3rd Asia Information Retrieval Symposium, 2006, pp. 666–672.
20. Zhang Y., Vines P., and Zobel J. Chinese OOV translation and post-translation query expansion in Chinese-English

cross-lingual information retrieval. ACM Trans. Asian Lang. Information Proc., 4(2):57–77, 2005.
21. http://www.fjoch.com/GIZA++.html

Cross-language Text Mining

▶ Cross-Language Mining and Retrieval

Cross-language Web Mining

▶ Cross-Language Mining and Retrieval

Cross-lingual Information Retrieval

▶ Cross-Language Mining and Retrieval

Cross-lingual Text Mining

▶ Cross-Language Mining and Retrieval

Cross-media Information Retrieval

▶ Cross-Modal Multimedia Information Retrieval

Cross-Modal Multimedia Information Retrieval

Qing Li, Yu Yang
City University of Hong Kong, Hong Kong, China

Synonyms

Multi-modal information retrieval; Cross-media information retrieval

Definition

Multimedia information retrieval tries to find the distinctive multimedia documents that satisfy people's needs within a huge dataset. Due to the vagueness on the representation of multimedia data, usually the user may only have some clues (e.g., a vague idea, a rough query object of the same or even different modality as that of the intended result) rather than concrete and indicative query objects. In such cases, traditional multimedia information retrieval techniques as Query-By-Example (QBE) fails to retrieve what users really want since their performance depends on a set of specifically defined features and carefully chosen query objects. The cross-modal multimedia information retrieval (CMIR) framework consists of a novel multifaceted knowledge base (which is embodied by a layered graph model) to discover the query results on multiple modalities. Such cross-modality paradigm leads to better query understanding and returns the retrieval result which meets user need better.

Historical Background

Previous works addressing multimedia information retrieval can be classified into two groups: approaches on single-modality, and those on multi-modality integration.

Retrieval Approaches on Single-Modality

The retrieval approach in this group only deals with a single type of media, so that most content-based retrieval (CBR) approaches [2,3,5,8,9] fall into this group. These approaches differ from each other in either the low-level features extracted from the data, or the distance functions used for similarity calculation. Despite the differences, all of them are similar in two fundamental aspects: (i) they all rely on low-level features; (ii) they all use the query-by-example paradigm.

Retrieval Approaches on Multi-Modality Integration

More recently there are some works that investigate the integration of multi-modality data, usually between text and image, for better retrieval performance. For example, iFind [7] proposes a unified framework under which the semantic feature (text) and low-level features are combined for image retrieval, whereas the 2M2Net [12] system extends this framework to the retrieval of video and audio. WebSEEK [9] extracts keywords from the surrounding text of image and videos, which is used as their indexes in the retrieval process. Although these systems involve more than one media, different types of media are not actually integrated but are on different levels. Usually, text is only used as the annotation (index) of other medias. In this regard, cross-modal multimedia information retrieval (CMIR) enables an extremely high degree of multi-modality integration, since it allows the interaction among objects of any modality in any possible ways (via different types of links).

MediaNet [1] and multimedia thesaurus (MMT) [10] seek to provide a multimedia representation of semantic concept – a concept described by various media objects including text, image, video, etc – and establish the relationships among these concepts. MediaNet extends the notion of relationships to include even perceptual relationships among media objects. Both approaches can be regarded as "concept-centric" approaches since they realize an organization of multi-modality objects around semantic concepts. In contrast, CMIR is "concept-less" since it makes no attempt to identify explicitly the semantics of each object.

Foundations

The cross-modality multimedia information retrieval (CMIR) mechanism shapes a novel scenario for multimedia retrieval: The user starts the search by supplying a set of seed objects as the hints of his intention, which can be of any modality (even different with the intended objects), and are not necessarily the eligible results by themselves. From the seeds, the system figures out the user's intention and returns a set of cross-modality objects that potentially satisfy this intention. The user can give further hints by identifying the results approximating his need, based on which the system improve its estimation about the user intention and refines the results towards it. This scenario can be also interpreted as a cooperative process: the user tries to focus the attention of the system to the objects by giving hints on the intended results, while the system tries to return more reasonable results that allows user to give better hints. A comparison between CMIR and the current CBR approaches is shown in Table 1.

To support all the necessary functionalities for such an ideal scenario, a suite of unique models, algorithms and strategies are developed in CMIR. As shown in Fig. 1, the foundation of the whole mechanism is a multifaceted knowledge base describing the relationships among cross-modality objects. The kernel of the knowledge base is a layered graph model, which characterizes the knowledge on (i) history of user behaviors, (ii) structural relationships among media objects, and (iii) content of media objects, at each of its layers. Link structure analysis—an established technique for web-oriented applications—is tailored to the retrieval of cross-modality data based on the layered graph model. A unique relevant feedback technique that gears with the underlying graph model is proposed, which can enrich the knowledge base by updating the links of the graph model according to user behaviors. The loop in Fig. 1 reveals the hill-climbing nature of the CMIR mechanism, i.e., it enhances its performance by learning from the previously conducted queries and feedbacks.

Layered Graph Model

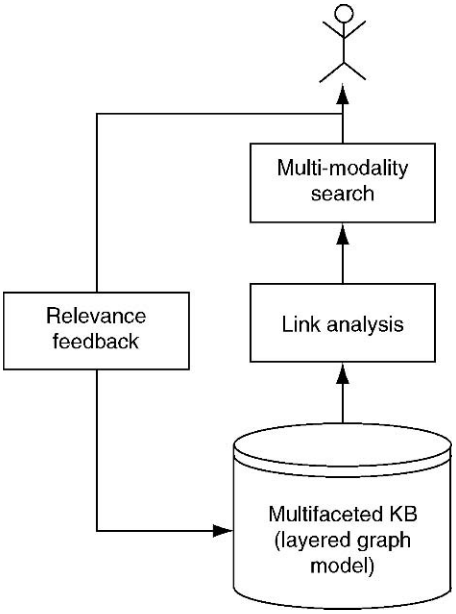
As the foundation of the retrieval capability, the multifaceted knowledge base accommodates a broad range of knowledge indicative of data semantics, mainly in three aspects: (i) user behaviors in the user-system interaction, (ii) structural relationships among media objects, and (iii) content of each media object. The kernel of the knowledge base is a three-layer graph model, with each layer describing the knowledge in one aspect, called knowledge layer. Its formal definition is given as follows.

Definition 1 *A* ***knowledge layer*** *is a undirected graph* $G = (V, E)$*, where* V *is a finite set of vertices and* E *is a finite set of edges. Each element in* V *corresponds to a media object* $O_i \in O$*, where* O *is the collection of media*

Cross-Modal Multimedia Information Retrieval. Table 1. CBR paradigms, drawbacks, and suggested remedies in CMIR

	CBR paradigms	Drawbacks	Suggested remedies in CMIR
Interaction	Highly representative sample object	Vague idea, or clear idea without appropriate samples	Cross-modality seed objects, only as hints
Data index	Low-level features	Inadequate to capture semantics	Multifaceted knowledge (user behaviors, structure, content)
Results	Single-modality, perceptually similar objects	Looks like or sounds like, but not what user actually needs	Cross-modality, semantically related objects

objects in the database. E is a ternary relation defined on $V \times V \times R$, *where R represents real numbers. Each edge in E has the form of* $<O_i, O_j, r>$, *denoting a semantic link between* O_i *and* O_j *with r as the weight of the link. The graph corresponds to a* $|V| \times |V|$ ***adjacency matrix*** (The adjacency matrix defined here is slightly different from the conventional definition in mathematics, in which each component is a binary value indicating the existence of the corresponding edge.) $M = [m_{ij}]$, *where* $m_{ij} = m_{ji}$ *always holds. Each element* $m_{ij} = r$ *if there is an edge* $<O_i, O_j, r>$, *and* $m_{ij} = 0$ *if there is no edge between* O_i *and* O_j. *The elements on the diagonal are set to zero, i.e.,* $m_{ii} = 0$.

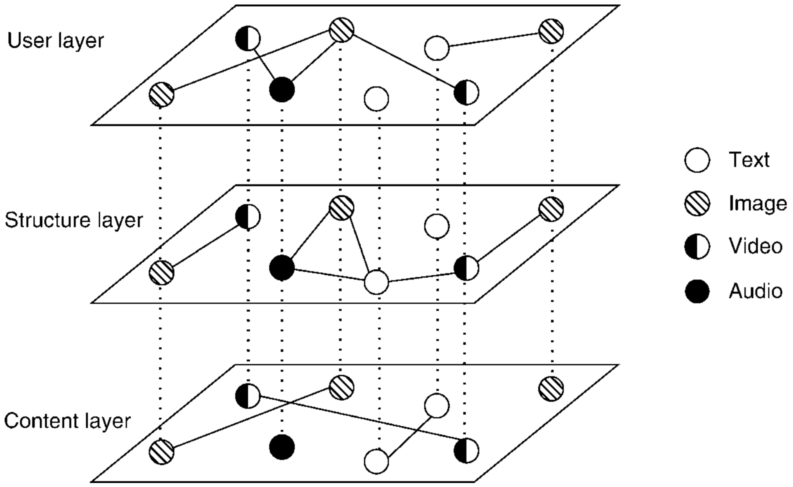


Cross-Modal Multimedia Information Retrieval. Figure 1. Overview of the CMIR mechanism.

Each semantic link between two media objects may have various interpretations, which corresponds to one of the three cases: (i) a user has implied the relevance between the two objects during the interaction, e.g., designating them as the positive example in the same query session; (ii) there is a structural relationships between them, e.g., they come from the same or linked web page(s); or (iii) they resemble each other in terms of their content. The multifaceted knowledge base seamlessly integrates all these links into the same model while preserving their mutual independence.

Definition 2 *The multifaceted knowledge base is a* ***layered graph model*** *consisting of three superimposed knowledge layers, which from top to bottom are* ***user layer***, ***structure layer***, *and* ***content layer***. *The vertices of the three layers correspond to the same set of media objects, but their edges are different either in occurrences or in interpretations.*

Figure 2 illustrates the layered graph model. Note that the ordering of the three layers is immutable, which reflects their priorities in terms of knowledge reliability. The user layer is placed uppermost since user judgment is assumed most reliable (not necessarily always reliable). Structure links is a strong indicator of relevance, but not as reliable as user links. The lowest layer is the content layer. As a generally accepted fact in



Cross-Modal Multimedia Information Retrieval. Figure 2. The Layered graph model as multifaceted knowledge base.

CBR area, content similarity does not entail any well-defined mapping with semantics.

A unique property of the layered graph model is that it stores the knowledge on the links (or relationships) among media objects, rather than on the nodes (media objects) upon which most existing retrieval systems store the data index. All the algorithms based of this model can be interpreted as manipulation of links: to serve the user query, relevant knowledge is extracted from this graph model by analyzing the link structure; meanwhile, user behaviors are studied to enrich the knowledge by updating the links. An advantage of such link-based approach is that the retrieval can be performed in a relatively small locality connected via links instead of the whole database, and therefore it can afford more sophisticated retrieval algorithms.

Link Analysis Based Retrieval

As illustrated in Fig. 3, the retrieval process can be described as a circle: the intended objects are retrieved through the upper semicircle, and the user evaluations are studied and incorporated into the knowledge base though the lower half-circle, which initiates a new circle to refine the previously retrieved results based on the updated knowledge. Consequently, it is a hill-climbing approach in that the performance is enhanced incrementally as the loop is repeated.

The retrieval process consists of five steps (as shown in Fig. 3): (i) generate the seed objects as the hints of the user's intention; (ii) span the seeds to a collection of candidate objects via the links in the layered graph model; (iii) distill the results by ranking the candidates based on link structure analysis, (iv) update the knowledge base to incorporate the user evaluation of the current results, and (v) refine the results based on user evaluations.

Key Applications

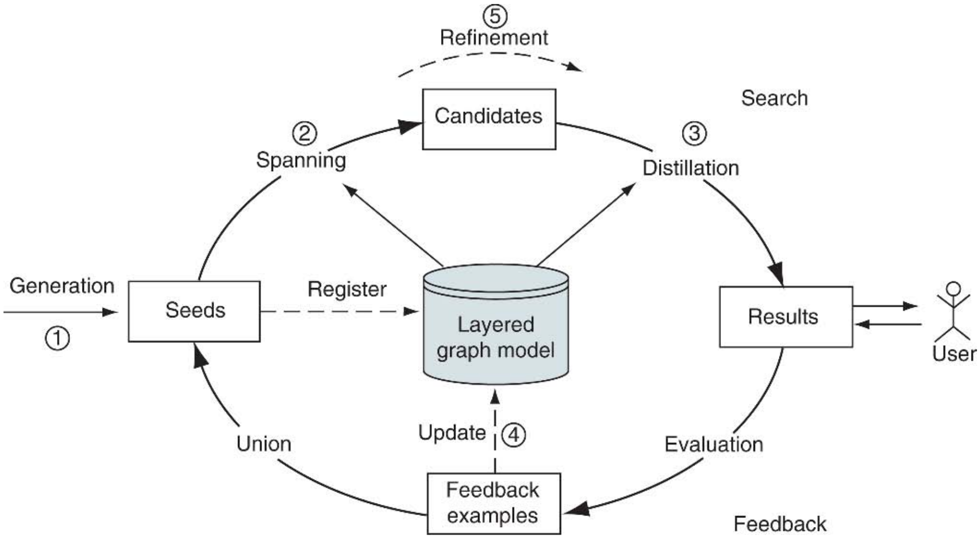
Multimedia Information Retrieval System

For multimedia data, the modalities supported can be texts (surrounding or tagged), images, videos and audios. An ongoing prototype [11] utilizes the primitive features and similarity functions for these media shown in Table 2. The experimental results prove the usefulness of the approach for better query understanding.

Future Directions

Due to the generality and extensibility of the CMIR, there are many potential directions that can be implemented on it:

Navigation. The graph model provides abundant links through which the user can traverse from an object to its related objects. An intuitive scenario for navigation is when the user is looking at a certain object, he is recommended with the objects that are linked to it in the graph model, ranked by their link weights and link types, from which he may select one as the next object he will navigate to.



Cross-Modal Multimedia Information Retrieval. Figure 3. Overview of the link analysis based retrieval algorithm.

Cross-Modal Multimedia Information Retrieval. Table 2. Primitive features and similarity function used in prototype

	Text	Image	Video
Primitive features	Keywords, weighted by TF*IDF	256-d HSV color histogram, 64-d LAB color coherence, 32-d Tamura directionality	First frame of each shot as key-frame, indexing key-frame as an image
Similarity function	Cosine distance	Euclidean distance for each feature, linear combination of different similarities	Key-frame (image) similarity as shot similarity, average pair-wise shot similarity as video similarity

Clustering. Clustering cross-modality objects into semantically meaningful groups is also an important and challenging issue, which requires an underlying *similarity function* among objects, along with a method that produces clusters based on the similarity function. The layered graph model provides knowledgeable and rich links, based on which different similarity functions can be easily formulated. Meanwhile, many existing approaches can be employed as the clustering method, such as simulated and deterministic annealing algorithm [4]. Moreover, CMIR inherently allows the clustering of cross-modality objects, rather than single-modality objects that most previous classification approaches can deal with.

Personalized retrieval. The user layer of the graph model characterizes the knowledge obtained from the behaviors of the whole population of users, and allows a query from a single user to benefit from such common knowledge. However, each user may have his/her personal interests, which may not agree with each other. The "multi-leveled user profile" mechanism [6] leads a good direction for future study.

Cross-references

▶ Multimedia Data

▶ Multimedia Information Retrieval

Recommended Reading

1. Benitez A.B., Smith J.R., and Chang S.F. MediaNet: a multimedia information network for knowledge representation. In Proc. SPIE Conf. on Internet Multimedia Management Systems, vol. 4210, 2000, pp. 1–12.
2. Chang S.F., Chen W., Meng H.J., Sundaram H., and Zhong D. VideoQ: an automated content based video search system using visual cues. In Proc. 5th ACM Int. Conf. on Multimedia, 1997.
3. Flickner M., Sawhney H., Niblack W., and Ashley J. Query by image and video content: the QBIC system. IEEE Comput., 28(9):23–32, 1995.
4. Hofmann T. and Buhmann J.M. Pairwise data clustering by deterministic annealing. IEEE Trans. Pattern Anal. Mach. Intell., 19(1):1–14, 1997.
5. Huang T.S., Mehrotra S., and Ramchandran K. Multimedia analysis and retrieval system (MARS) project. In Proc. 33rd Annual Clinic on Library Application of Data Processing-Digital Image Access and Retrieval, 1996.
6. Li Q., Yang J., and Zhuang Y.T. Web-based multimedia retrieval: balancing out between common knowledge and personalized views. In Proc. 2nd Int. Conf. on Web Information Systems Eng., 2001.
7. Lu Y., Hu C.H., Zhu X.Q., Zhang H.J., and Yang Q. A unified framework for semantics and feature based relevance feedback in image retrieval systems. In Proc. 8th ACM Int. Conf. on Multimedia, 2000, pp. 31–38.
8. Smith J.R. and Chang S.F. VisualSEEk: a fully automated content-based image query system. In Proc. 4th ACM Int. Conf. on Multimedia, 1996.
9. Smith J.R. and Chang S.F. Visually searching the web for content. IEEE Multimed. Mag., 4(3):12–20, 1997.
10. Tansley R. The Multimedia Thesaurus: An Aid for Multimedia Information Retrieval and Navigation. Master Thesis, Computer Science, University of Southampton, UK, 1998.
11. Yang J., Li Q., and Zhuang Y. Octopus: Aggressive search of multi-modality data using multifaceted knowledge base. In Proc. 11th Int. World Wide Web Conference, 2002, pp. 54–64.
12. Yang J., Zhuang Y.T., and Li Q. Search for multi-modality data in digital libraries. In Proc. Second IEEE Pacific-Rim Conference on Multimedia, 2001.

Cross-Validation

Payam Refaeilzadeh, Lei Tang, Huan Liu
Arizona State University, Tempe, AZ, USA

Synonyms

Rotation estimation

Definition

Cross-Validation is a statistical method of evaluating and comparing learning algorithms by dividing data into two segments: one used to learn or train a model and the other used to validate the model. In typical cross-validation, the training and validation sets must cross-over in successive rounds such that each data point has a chance of being validated against. The basic form of cross-validation is k-fold cross-validation. Other forms of cross-validation are special cases of *k*-fold cross-validation or involve repeated rounds of *k*-fold cross-validation.

In *k*-fold cross-validation, the data is first partitioned into *k* equally (or nearly equally) sized segments or folds. Subsequently *k* iterations of training and validation are performed such that within each iteration a different fold of the data is held-out for validation while the remaining $k - 1$ folds are used for learning. Fig. 1 demonstrates an example with $k = 3$. The darker section of the data are used for training while the lighter sections are used for validation. In data mining and machine learning 10-fold cross-validation ($k = 10$) is the most common.

Cross-validation is used to evaluate or compare learning algorithms as follows: in each iteration, one or more learning algorithms use $k - 1$ folds of data to learn one or more models, and subsequently the learned models are asked to make predictions about the data in the validation fold. The performance of each learning algorithm on each fold can be tracked using some predetermined performance metric like accuracy. Upon completion, *k* samples of the performance metric will be available for each algorithm. Different methodologies such as averaging can be used to obtain an aggregate measure from these sample, or these samples can be used in a statistical hypothesis test to show that one algorithm is superior to another.

Historical Background

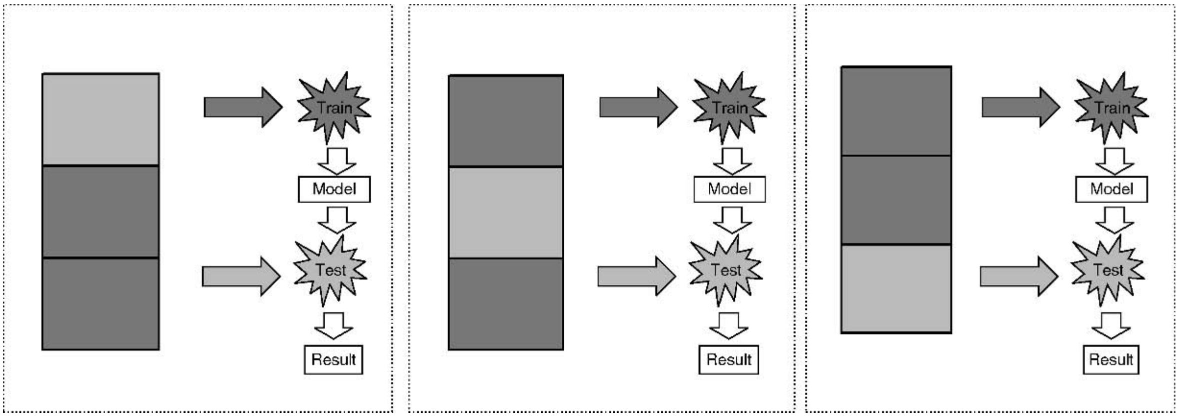
In statistics or data mining, a typical task is to learn a model from available data. Such a model may be a regression model or a classifier. The problem with evaluating such a model is that it may demonstrate adequate prediction capability on the training data, but might fail to predict future unseen data. cross-validation is a procedure for estimating the generalization performance in this context. The idea for cross-validation originated in the 1930s [6]. In the paper one sample is used for regression and a second for prediction. Mosteller and Tukey [9], and various other people further developed the idea. A clear statement of cross-validation, which is similar to current version of *k*-fold cross-validation, first appeared in [8]. In 1970s, both Stone [12] and Geisser [4] employed cross-validation as means for choosing proper model parameters, as opposed to using cross-validation purely for estimating model performance. Currently, cross-validation is widely accepted in data mining and machine learning community, and serves as a standard procedure for performance estimation and model selection.

Foundations

There are two possible goals in cross-validation:

- To estimate performance of the learned model from available data using one algorithm. In other words, to gauge the generalizability of an algorithm.
- To compare the performance of two or more different algorithms and find out the best algorithm for the available data, or alternatively to compare the performance of two or more variants of a parameterized model.

The above two goals are highly related, since the second goal is automatically achieved if one knows the accurate estimates of performance. Given a sample



Cross-Validation. Figure 1. Procedure of three-fold cross-validation.

of N data instances and a learning algorithm A, the average cross-validated accuracy of A on these N instances may be taken as an estimate for the accuracy of A on unseen data when A is trained on all N instances. Alternatively if the end goal is to compare two learning algorithms, the performance samples obtained through cross-validation can be used to perform two-sample statistical hypothesis tests, comparing a pair of learning algorithms.

Concerning these two goals, various procedures are proposed:

Resubstitution Validation

In resubstitution validation, the model is learned from all the available data and then tested on the same set of data. This validation process uses all the available data but suffers seriously from over-fitting. That is, the algorithm might perform well on the available data yet poorly on future unseen test data.

Hold-Out Validation

To avoid over-fitting, an independent test set is preferred. A natural approach is to split the available data into two non-overlapped parts: one for training and the other for testing. The test data is held out and not looked at during training. Hold-out validation avoids the overlap between training data and test data, yielding a more accurate estimate for the generalization performance of the algorithm. The downside is that this procedure does not use all the available data and the results are highly dependent on the choice for the training/test split. The instances chosen for inclusion in the test set may be too easy or too difficult to classify and this can skew the results. Furthermore, the data in the test set may be valuable for training and if it is held-out prediction performance may suffer, again leading to skewed results. These problems can be partially addressed by repeating hold-out validation multiple times and averaging the results, but unless this repetition is performed in a systematic manner, some data may be included in the test set multiple times while others are not included at all, or conversely some data may always fall in the test set and never get a chance to contribute to the learning phase. To deal with these challenges and utilize the available data to the max, k-fold cross-validation is used.

K-Fold Cross-Validation

In k-fold cross-validation the data is first partitioned into k equally (or nearly equally) sized segments or folds. Subsequently k iterations of training and validation are performed such that within each iteration a different fold of the data is held-out for validation while the remaining $k-1$ folds are used for learning. Data is commonly stratified prior to being split into k folds. Stratification is the process of rearranging the data as to ensure each fold is a good representative of the whole. For example in a binary classification problem where each class comprises 50% of the data, it is best to arrange the data such that in every fold, each class comprises around half the instances.

Leave-One-Out Cross-Validation

Leave-one-out cross-validation (LOOCV) is a special case of k-fold cross-validation where k equals the number of instances in the data. In other words in each iteration nearly all the data except for a single observation are used for training and the model is tested on that single observation. An accuracy estimate obtained using LOOCV is known to be almost unbiased but it has high variance, leading to unreliable estimates [3]. It is still widely used when the available data are very rare, especially in bioinformatics where only dozens of data samples are available.

Repeated *K*-Fold Cross-Validation

To obtain reliable performance estimation or comparison, large number of estimates are always preferred. In k-fold cross-validation, only k estimates are obtained. A commonly used method to increase the number of estimates is to run k-fold cross-validation multiple times. The data is reshuffled and re-stratified before each round.

Pros and Cons

Kohavi [5] compared several approaches to estimate accuracy: cross-validation(including regular cross-validation, leave-one-out cross-validation, stratified cross-validation) and bootstrap (sample with replacement), and recommended *stratified 10-fold cross-validation* as the best model selection method, as it tends to provide less biased estimation of the accuracy.

Salzberg [11] studies the issue of comparing two or more learning algorithms based on a performance metric, and proposes using k-fold cross-validation followed by appropriate hypothesis test rather than directly comparing the average accuracy. Paired t-test is one test which takes into consideration the variance of training and test data, and is widely used in machine learning. Dietterich [2] studied the properties of 10-fold cross-validation followed by a paired t-test in detail and found that such a test suffers from higher than expected type I error. In this study, this high type I error was attributed to high variance. To correct for this Dietterich proposed a new test: 5 × 2-fold cross-validation. In this test 2-fold cross-validation is run five times resulting in 10 accuracy values. The data is re-shuffled and re-stratified after each round. All 10 values are used for average accuracy estimation in the t-test but only values from one of the five 2-fold cross-validation rounds is used to estimate variance. In this study 5 × 2-fold cross-validation is shown to have acceptable type I error but not to be as powerful as 10-fold cross validation and has not been widely accepted in data mining community.

Bouckaert [1] also studies the problem of inflated type-I error with 10-fold cross-validation and argues that since the samples are dependent (because the training sets overlap), the actual degrees of freedom is much lower than theoretically expected. This study compared a large number of hypothesis schemes, and recommend 10 × 10 fold cross-validation to obtain 100 samples, followed with t-test with degree of freedom equal to 10 (instead of 99). However this method has not been widely adopted in data mining field either and 10-fold cross-validation remains the most widely used validation procedure.

A brief summery of the above results is presented in Table 1.

Cross-Validation. Table 1. Pros and Cons of different validation methods

Validation method	Pros	Cons
Resubstitution Validation	Simple	Over-fitting
Hold-out Validation	Independent training and test	Reduced data for training and testing; Large variance
k-fold cross validation	Accurate performance estimation	Small samples of performance estimation; Overlapped training data; Elevated Type I error for comparison; Underestimated performance variance or overestimated degree of freedom for comparison
Leave-One-Out cross-validation	Unbiased performance estimation	Very large variance
Repeated *k*-fold cross-validation	Large number of performance estimates	Overlapped training and test data between each round; Underestimated performance variance or overestimated degree of freedom for comparison

Why 10-Fold Cross-Validation: From Ideal to Reality

Whether estimating the performance of a learning algorithm or comparing two or more algorithms in terms of their ability to learn, an ideal or statistically sound experimental design must provide a *sufficiently large* number of *independent* measurements of the algorithm(s) performance.

To make independent measurements of an algorithm's performance one must ensure that the factors affecting the measurement are independent from one run to the next. These factors are: (i) the training data the algorithm learns from and, (ii) the test data one uses to measure the algorithm's performance. If some data is used for testing in more than one round, the obtained results, for example the accuracy measurements from these two rounds, will be dependent and a statistical comparison may not be valid. In fact, it has been shown that a paired t-test based on taking several random train/test splits tends to have an extremely high probability of Type I error and should never be used [2].

Not only must the datasets be independently controlled across different runs, there must not be any overlap between the data used for learning and the data used for validation in the same run. Typically, a learning algorithm can make more accurate predictions on a data that it has seen during the learning phase than those it has not. For this reason, an overlap between the training and validation set can lead to an over-estimation of the performance metric and is forbidden. To satisfy the other requirement, namely a sufficiently large sample, most statisticians call for 30+ samples.

For a truly sound experimental design, one would have to split the available data into 30 × 2 = 60 partitions to perform 30 truly independent train-test runs. However, this is not practical because the performance of learning algorithms and their ranking is generally not invariant with respect to the number of samples available for learning. In other words, an estimate of accuracy in such a case would correspond to the accuracy of the learning algorithm when it learns from just 1⁄60 of the available data (assuming training and validation sets are of the same size). However, the accuracy of the learning algorithm on unseen data when the algorithm is trained on all the currently available data is likely much higher since learning algorithms generally improve in accuracy as more data becomes available for learning. Similarly, when comparing two algorithms A and B, even if A is discovered to be the superior algorithm when using 1⁄60 of the available data, there is no guarantee that it will also be the superior algorithm when using all the available data for learning. Many high performing learning algorithms use complex models with many parameters and they simply will not perform well with a very small amount of data. But they may be exceptional when sufficient data is available to learn from.

Recall that two factors affect the performance measure: the training set, and the test set. The training set affects the measurement indirectly through the learning algorithm, whereas the composition of the test set has a direct impact on the performance measure. A reasonable experimental compromise may be to *allow for overlapping training sets, while keeping the test sets independent*. *K*-fold cross-validation does just that.

Now the issue becomes selecting an appropriate value for k. A large k is seemingly desirable, since with a larger k (i) there are more performance estimates, and (ii) the training set size is closer to the full data size, thus increasing the possibility that any conclusion made about the learning algorithm(s) under test will generalize to the case where all the data is used to train the learning model. As k increases, however, the overlap between training sets also increases. For example, with 5-fold cross-validation, each training set shares only 3⁄4 of its instances with each of the other four training sets whereas with 10-fold cross-validation, each training set shares 8/9 of its instances with each of the other nine training sets. Furthermore, increasing k shrinks the size of the test set, leading to less precise, less fine-grained measurements of the performance metric. For example, with a test set size of 10 instances, one can only measure accuracy to the nearest 10%, whereas with 20 instances the accuracy can be measured to the nearest 5%. These competing factors have all been considered and the general consensus in the data mining community seems to be that $k = 10$ is a good compromise. This value of k is particularity attractive because it makes predictions using 90% of the data, making it more likely to be generalizable to the full data.

Key Applications

Cross-validation can be applied in three contexts: performance estimation, model selection, and tuning learning model parameters.

Performance Estimation

As previously mentioned, cross-validation can be used to estimate the performance of a learning algorithm. One may be interested in obtaining an estimate for any of the many performance indicators such as accuracy, precision, recall, or F-score. Cross-validation allows for all the data to be used in obtaining an estimate. Most commonly one wishes to estimate the accuracy of a classifier in a supervised-learning environment. In such a setting, a certain amount of labeled data is available and one wishes to predict how well a certain classifier would perform if the available data is used to train the classifier and subsequently ask it to label unseen data. Using 10-fold cross-validation one repeatedly uses 90% of the data to build a model and test its accuracy on the remaining 10%. The resulting average accuracy is likely somewhat of an underestimate for the true accuracy when the model is trained on all data and tested on unseen data, but in most cases this estimate is reliable, particularly if the amount of labeled data is sufficiently large and if the unseen data follows the same distribution as the labeled examples.

Model Selection

Alternatively cross-validation may be used to compare a pair of learning algorithms. This may be done in the case of newly developed learning algorithms, in which case the designer may wish to compare the performance of the classifier with some existing baseline classifier on some benchmark dataset, or it may be done in a generalized model-selection setting. In generalized model selection one has a large library of learning algorithms or classifiers to choose from and wish to select the model that will perform best for a particular dataset. In either case the basic unit of work is pair-wise comparison of learning algorithms. For generalized model selection combining the results of many pair-wise comparisons to obtain a single *best* algorithm may be difficult, but this is beyond the scope of this article. Researchers have shown that when comparing a pair of algorithms using cross-validation it is best to employ proper two sample hypothesis testing instead of directly comparing the average accuracies. Cross-validation yields k pairs of accuracy values for the two algorithms under test. It is possible to make a null hypothesis assumption that the two algorithms perform equally well and set out to gather evidence against this null-hypothesis using a two-sample test. The most widely used test is the paired t-test. Alternatively the non-parametric sign test can be used.

A special case of model selection comes into play when dealing with non-classification model selection. For example when trying to pick a feature selection [7] algorithm that will maximize a classifier's performance on a particular dataset. Refaeilzadeh et al. [10] explore this issue in detail and explain that there are in fact two variants of cross-validation in this case: performing feature selection before splitting data into folds (OUT) or performing feature selection k times inside the cross-validation loop (IN). The paper explains that there is potential for bias in both cases: With OUT, the feature selection algorithm has looked at the test set, so the accuracy estimate is likely inflated; On the other hand with IN the feature selection algorithm is looking at less data than would be available in a real experimental setting, leading to underestimated accuracy. Experimental results confirm these hypothesis and further show that:

- In cases where the two feature selection algorithms are not statistically differentiable, IN tends to be more truthful.
- In cases where one algorithm is better than another, IN often favors one algorithm and OUT the other.

OUT can in fact be the better choice even if it demonstrates a larger bias than IN in estimating accuracy. In other words, *estimation bias is not necessarily an indication of poor pair-wise comparison*. These subtleties about the potential for bias and validity of conclusions obtained through cross-validation should always be kept in mind, particularly when the model selection task is a complicated one involving pre-processing as well as learning steps.

Tuning

Many classifiers are parameterized and their parameters can be *tuned* to achieve the best result with a particular dataset. In most cases it is easy to learn the proper value for a parameter from the available data. Suppose a Naïve Bayes classifier is being trained on a dataset with two classes: {+, –}. One of the parameters for this classifier is the prior probability p(+). The best value for this parameter according to the available data can be obtained by simply counting the number of instances that are labeled positive and dividing this number by the total number of instances. However in some cases parameters do not have such intrinsic meaning, and there is no good way to pick a best value other than trying out many values and picking the one that yields the highest performance. For example, support vector machines (SVM) use soft-margins to deal with noisy data. There is no easy way of learning the best value for the soft margin parameter for a particular dataset other than trying it out and seeing how it works. In such cases, cross-validation can be performed on the training data as to measure the performance with each value being tested. Alternatively a portion of the training set can be reserved for this purpose and not used in the rest of the learning process. But if the amount of labeled data is limited, this can significantly degrade the performance of the learned model and cross-validation may be the best option.

Cross-references

▶ Classification
▶ Evaluation Metrics for Structured Text Retrieval
▶ Feature Selection for Clustering

Recommended Reading

1. Bouckaert R.R. Choosing between two learning algorithms based on calibrated tests. In Proc. 20th Int. Conf. on Machine Learning, 2003, pp. 51–58.
2. Dietterich T.G. Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput., 10(7):1895–1923, 1998.
3. Efron B. Estimating the error rate of a prediction rule: improvement on cross-validation. J. Am. Stat. Assoc., 78: 316–331,1983.
4. Geisser S. The predictive sample reuse method with applications. J. Am. Stat. Assoc., 70(350):320–328,1975.
5. Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proc. 14th Int. Joint Conf. on AI, 1995, pp. 1137–1145.
6. Larson S. The shrinkage of the coefficient of multiple correlation. J. Educat. Psychol., 22:45–55, 1931.
7. Liu H. and Yu L. Toward integrating feature selection algorithms for classification and clustering. IEEE Trans. Knowl. Data Eng., 17(4):491–502, 2005.

8. Mosteller F. and Tukey J.W. Data analysis, including statistics. In Handbook of Social Psychology. Addison-Wesley, Reading, MA, 1968.
9. Mosteller F. and Wallace D.L. Inference in an authorship problem. J. Am. Stat. Assoc., 58:275–309, 1963.
10. Refaeilzadeh P., Tang L., and Liu H. On comparison of feature selection algorithms. In Proc. AAAI-07 Workshop on Evaluation Methods in Machine Learing II. 2007, pp. 34–39.
11. Salzberg S. On comparing classifiers: pitfalls to avoid and a recommended approach. Data Min. Knowl. Disc., 1(3):317–328, 1997.
12. Stone M. Cross-validatory choice and assessment of statistical predictions. J. Royal Stat. Soc., 36(2):111–147, 1974.

Cryptographic Hash Functions

▶ Hash Functions

C-Tables

▶ Conditional Tables

Cube

Torben Bach Pedersen
Aalborg University, Aalborg, Denmark

Synonyms

Hypercube

Definition

A cube is a data structure for storing and and analyzing large amounts of multidimensional data, often referred to as *On-Line Analytical Processing (OLAP)*. Data in a cube lives in a space spanned by a number of hierarchical *dimensions*. A single point in this space is called a *cell*. A (non-empty) cell contains the values of one or more *measures*.

Key Points

As an example, a three-dimensional cube for capturing sales may have a Product *dimension P*, a Time dimension *T*, and a Store dimension *S*, capturing the product sold, the time of sale, and the store it was sold in, for each sale, respectively. The cube has two *measures*: Dollar Sales and ItemSales, capturing the sales price and the number of items sold, respectively. In a cube, the combinations of a dimension value from each dimension define a *cell* of the cube. The measure value(s), e.g., DollarSales and ItemSales, corresponding to the particular combination of dimension values are then stored stored in the corresponding cells.

Data cubes provide true multidimensionality. They generalize spreadsheets to any number of dimensions, indeed cubes are popularly referred to as "spreadsheets on stereoids." In addition, *hierarchies* in dimensions and formulas are first-class, built-in concepts, meaning that these are supported without duplicating their definitions. A collection of related cubes is commonly referred to as a *multidimensional database* or a *multidimensional data warehouse*.

In a cube, dimensions are first-class concepts with associated domains, meaning that the addition of new dimension values is easily handled. Although the term "cube" implies three dimensions, a cube can have any number of dimensions. It turns out that most real-world cubes have 4–12 dimensions [3]. Although there is no theoretical limit to the number of dimensions, current tools often experience performance problems when the number of dimensions is more than 10–15. To better suggest the high number of dimensions, the term "hypercube" is often used instead of "cube."

Depending on the specific application, a highly varying percentage of the cells in a cube are non-empty, meaning that cubes range from *sparse* to *dense*. Cubes tend to become increasingly sparse with increasing dimensionality and with increasingly finer granularities of the dimension values. A non-empty cell is called a *fact*. The example has a fact for each combination of time, product, and store where at least one sale was made.

Generally, only two or three dimensions may be viewed at the same time, although for low-cardinality dimensions, up to four dimensions can be shown by nesting one dimension within another on the axes. Thus, the dimensionality of a cube is reduced at query time by *projecting* it down to two or three dimensions via *aggregation* of the measure values across the projected-out dimensions. For example, to view sales by Store and Time, data is aggregates over the entire Product dimension, i.e., for all products, for each combination of Store and Time.

OLAP SQL extensions for cubes were pioneered by the proposal of the data cube operators CUBE and ROLLUP [1]. The CUBE operator generalizes GROUP BY, crosstabs, and subtotals using the special "ALL" value that denotes that an aggregation has been performed over all values for one or more attributes, thus generating a subtotal, or a grand total.

Cross-references

- ▶ Cube Implementations
- ▶ Dimension
- ▶ Hierarchy
- ▶ Measure
- ▶ Multidimensional Modeling
- ▶ On-Line Analytical Processing

Recommended Reading

1. Gray J., Chaudhuri S., Bosworth A., Layman A., Venkatrao M., Reichart D., Pellow F., and Pirahesh H. Data cube: a relational aggregation operator generalizing group-by, cross-tab and sub-totals. Data Mining Knowl. Discov., 1(1):29–54, 1997.
2. Pedersen T.B., Jensen C.S., and Dyreson C.E. A foundation for capturing and querying complex multidimensional data. Inf. Syst., 26(5):383–423, 2001.
3. Thomsen E. OLAP Solutions: Building Multidimensional Information Systems. Wiley, New York, 1997.

Cube Implementations

Konstantinos Morfonios, Yannis Ioannidis
University of Athens, Athens, Greece

Synonyms

Cube materialization; Cube precomputation

Definition

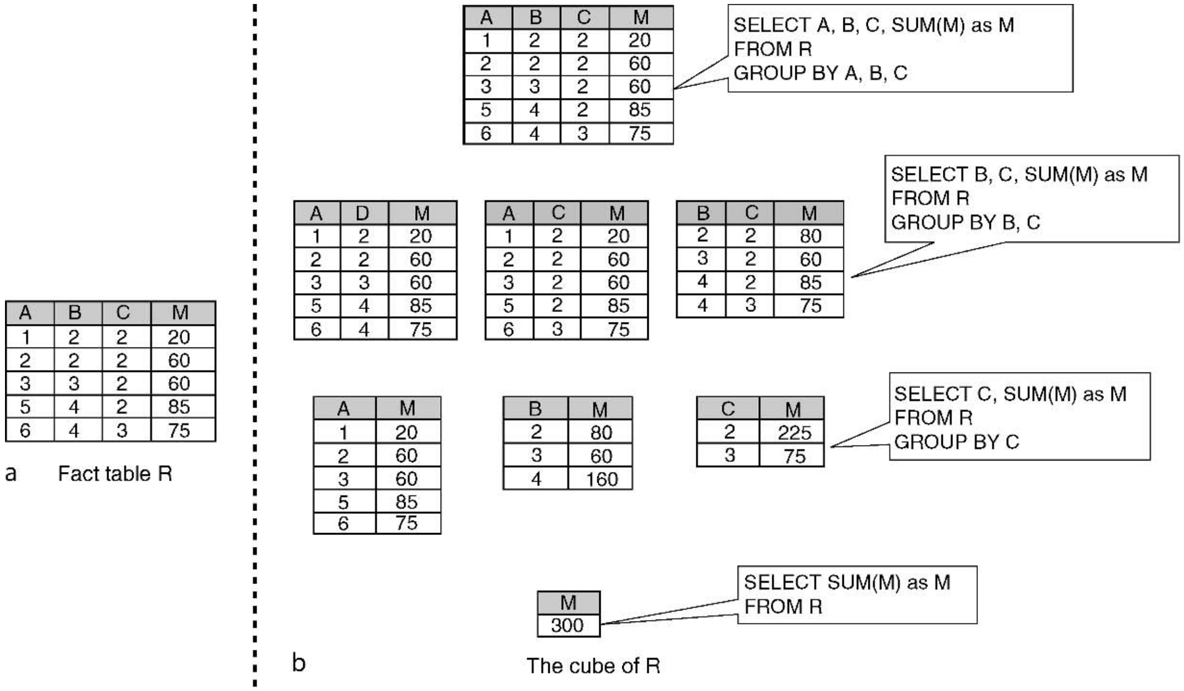
Cube implementation involves the procedures of computation, storage, and manipulation of a data cube, which is a disk structure that stores the results of the aggregate queries that group the tuples of a fact table on all possible combinations of its dimension attributes. For example in Fig. 1a, assuming that R is a fact table that consists of three dimensions (A, B, C) and one measure M (see definitional entry for *Measure*), the corresponding cube of R appears in Fig. 1b. Each cube node (i.e., view that belongs to the data cube) stores the results of a particular aggregate query as shown in Fig. 1b. Clearly, if D denotes the number of dimensions of a fact table, the number of all possible aggregate queries is 2^D; hence, in the worst case, the size of the data cube is exponentially larger with respect to D than the size of the original fact table. In typical applications, this may be in the order of gigabytes or even terabytes, implying that the development of efficient algorithms for the implementation of cubes is extremely important.

Let *grouping attributes* be the attributes of the fact table that participate in the group-by clause of an aggregate query expressed in SQL. A common representation of the data cube that captures the computational dependencies among all the aggregate queries that are necessary for its materialization is the cube lattice [6]. This is a directed acyclic graph (DAG) where each node represents an aggregate query *q* on the fact table and is connected via a directed edge with every other node whose corresponding group-by part is missing one of the grouping attributes of *q*. For example, Fig. 2 shows the cube lattice that corresponds to the fact table R (Fig. 1a).

Note that precomputing and materializing parts of the cube is crucial for the improvement of query-response times as well as for accelerating operators that are common in On-Line Analytical Processing (OLAP), such as drill-down, roll-up, pivot, and slice-and-dice, which make an extensive use of aggregation [3]. Materialization of the entire cube seems ideal for efficiently accessing aggregated data; nevertheless, in real-world applications, which typically involve large volumes of data, it may be considerably expensive in terms of storage space, as well as computation and maintenance time. In the existing literature, several efficient methods have been proposed that attempt to balance the aforementioned tradeoff between query-response times and other resource requirements. Their brief presentation is the main topic of this entry.

Historical Background

Most data analysis efforts, whether manual by analysts or automatic by specialized algorithms, manipulate the contents of database systems in order to discover trends and correlations. They typically involve complex queries that make an extensive use of aggregation in order to group together tuples that "behave in a similar fashion." The response time of such queries over extremely large data warehouses can be prohibitive. This problem inspired Gray et al. [3] to introduce the data-cube operator and propose its off-line computation and storage for efficiency at query time. The corresponding seminal publication has been the seed for a plethora of papers thereafter, which have dealt with several different aspects of the lifecycle of a data cube, from cube construction and storage to indexing, query answering, and incremental maintenance.



A	B	C	M
1	2	2	20
2	2	2	60
3	3	2	60
5	4	2	85
6	4	3	75

a Fact table R

A	B	C	M
1	2	2	20
2	2	2	60
3	3	2	60
5	4	2	85
6	4	3	75

```
SELECT A, B, C, SUM(M) as M
FROM R
GROUP BY A, B, C
```

A	D	M
1	2	20
2	2	60
3	3	60
5	4	85
6	4	75

A	C	M
1	2	20
2	2	60
3	2	60
5	2	85
6	3	75

B	C	M
2	2	80
3	2	60
4	2	85
4	3	75

```
SELECT B, C, SUM(M) as M
FROM R
GROUP BY B, C
```

A	M
1	20
2	60
3	60
5	85
6	75

B	M
2	80
3	60
4	160

C	M
2	225
3	75

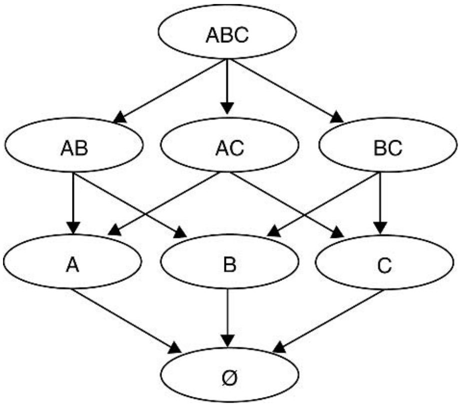
```
SELECT C, SUM(M) as M
FROM R
GROUP BY C
```

M
300

```
SELECT SUM(M) as M
FROM R
```

b The cube of R

Cube Implementations. Figure 1. Fact table R and the corresponding data cube.



Cube Implementations. Figure 2. Example of a cube lattice.

Taking into account the format used for the computation and storage of a data cube, the cube-implementation algorithms that have appeared in the literature can be partitioned into four main categories: Relational-OLAP (ROLAP) algorithms exploit traditional materialized views in RDBMSes; Multidimensional-OLAP (MOLAP) algorithms take advantage of multidimensional arrays; Graph-Based methods use specialized graph structures; finally, approximation algorithms use various in-memory representations, e.g., histograms.

The literature also deals with the rest of the cubes lifecycle [12]. Providing fast answers to OLAP aggregate queries is the main purpose of implementing data cubes to begin with, and various algorithms have been proposed to handle different types of queries on the formats above. Moreover, as data stored in the original fact table changes, data cubes must follow suit; otherwise, analysis of obsolete data may result into invalid conclusions. Periodical reconstruction of the entire cube is impractical, hence, incremental-maintenance techniques have been proposed.

The ideal implementation of a data cube must address efficiently all aspects of cube functionality in order to be viable. In the following section, each one of these aspects is further examined separately.

Foundations

In the following subsections, the main stages of the cube lifecycle are analyzed in some detail, including subcube selection, computation, query processing, and incremental maintenance. Note that the references given in this section are only indicative, since the number of related publications is actually very large. A more comprehensive survey may be found elsewhere [11].

Subcube Selection

In real-world applications, materialization of the entire cube is often extremely expensive in terms of computation, storage, and maintenance requirements, mainly because of the typically large fact-table size and the exponential number of cube nodes with respect to the number of dimensions. To overcome this drawback, several existing algorithms select an appropriate subset of the data cube for precomputation and storage [4,5,6]. Such selection algorithms try to balance the tradeoff between response times of queries (sometimes of a particular, expected workload) and resource requirements for cube construction, storage, and maintenance. It has been shown [6] that selection of the optimum subset of a cube is an NP-complete problem. Hence, the existing algorithms use heuristics in order to find near-optimal solutions.

Common constraints used during the selection process involve constraints on the time available for cube construction and maintenance, and/or on the space available for cube storage. As for the criteria that are (approximately) optimized during selection, they typically involve some form of the benefit gained from the materialization of a particular cube subset.

A particularly beneficial criterion for the selection problem that needs some more attention, since it has been integrated in some of the most efficient cube-implementation algorithms (including Dwarf [17] and CURE [10], which will be briefly presented below) is the so-called redundancy reduction. Several groups of researchers have observed that a big part of the cube data is usually redundant [7,8,10,12,17,20]. Formally, a value stored in a cube is *redundant* if it is repeated multiple times in the same attribute in the cube. For example, in Fig. 1b, tuples $\langle 1, 20\rangle$ of node A, $\langle 1, 2, 20\rangle$ of AB, and $\langle 1, 2, 20\rangle$ of AC are redundant, since they can be produced by properly projecting tuple $\langle 1, 2, 2, 20\rangle$ of node ABC. By appropriately avoiding the storage of such redundant data, several existing cube-implementation algorithms achieve the construction of compressed cubes that can still be considered as fully materialized. Typically, the decrease in the final cube size is impressive, a fact that benefits the performance of computation as well, since output costs are considerably reduced and sometimes, because early identification of redundancy allows pruning of parts of the computation. Furthermore, during query answering, aggregation and decompression are not necessary; instead, some simple operations, e.g., projections, are enough.

Finally, for some applications (e.g., for mining multidimensional association rules), accessing the tuples of the entire cube is not necessary, because they only need those group-by tuples with an aggregate value (e.g. count) above some prespecified minimum support threshold (minsup). For such cases, the concept of *iceberg cubes* has been introduced [2]. Iceberg-cube construction algorithms [2,16] take into consideration only sets of tuples that aggregate together giving a value greater than minsup. Hence, they perform some kind of subcube selection, by storing only the tuples that satisfy the aforementioned condition.

Cube Computation

Cube computation includes scanning the data of the fact table, aggregating on all grouping attributes, and generating the contents of the data cube. The main goal of this procedure is to place tuples that aggregate together (i.e., tuples with identical grouping-attribute values) in contiguous positions in main memory, in order to compute the required aggregations with as few data passes as possible. The most widely used algorithms that accomplish such clustering of tuples are sorting and hashing. Moreover, nodes connected in the cube lattice (Fig. 2) exhibit strong computational dependencies, whose exploitation is particularly beneficial for the performance of the corresponding computation algorithms. For instance, assuming that the data in the fact table R (Fig. 1a) is sorted according to the attribute combination ABC, one can infer that it is also sorted according to both AB and A as well. Hence, the overhead of sorting can be shared by the computation of multiple aggregations, since nodes ABC $\rightarrow$AB $\rightarrow$A $\rightarrow\emptyset$ can be computed with the use of pipelining without reclustering the data. Five methods that take advantage of such node computational dependencies have been presented in the existing literature [1] in order to improve the performance of computation algorithms: smallest-parent, cache-results, amortize-scans, share-shorts, and share-partitions.

Expectedly, both sort-based and hash-based aggregation methods perform more efficiently when the data they process fits in main memory; otherwise, they are forced to use external-memory algorithms, which generally increase the I/O overhead by a factor of two or three. In order to overcome such problems, most computation methods initially apply a step that partitions data into segments that fit in main memory, called *partitions* [2,10,15]. Partitioning algorithms distribute the tuples of the fact table in accordance with the principle that tuples that aggregate together must be placed in the same partition. Consequently, they can later process each partition independently of the others, since by construction, tuples that belong to different partitions do not share the same grouping-attribute values.

In addition to the above, general characteristics of cube-computation algorithms, there are some further details that are specific to each of four main categories mentioned above (i.e., ROLAP, MOLAP, Graph-Based, and Approximate), which are touched upon below.

ROLAP algorithms store a data cube as a set of materialized relational views, most commonly using either a *star* or a *snowflake schema*. Among these algorithms, algorithm CURE [10] seems to be the most promising, since it is the only solution with the following features: It is purely compatible with the ROLAP framework, hence its integration into any existing relational engine is rather straightforward. Also, it is suitable not only for "flat" datasets but also for processing datasets whose dimension values are hierarchically organized. Furthermore, it introduces an efficient algorithm for external partitioning that allows the construction of cubes over extremely large volumes of data whose size may far exceed the size of main memory. Finally, it stores cubes in a compressed form, removing all types of redundancy from the final result.

MOLAP algorithms store a data cube as a multidimensional array, thereby avoiding to store the dimension values in each array cell, since the position of the cell itself determines these values. The main drawback of this approach comes from the fact that, in practice, cubes have a large number of empty cells (i.e., cubes are sparse), rendering MOLAP algorithms inefficient with respect to their storage-space requirements. To overcome this problem, the so-called chunk-based algorithms have been introduced [21], which avoid the physical storage of most of the empty cells, storing only *chunks*, which are nonempty subarrays. ArrayCube [21] is the most widely accepted algorithm in this category. It has also served as an inspiration to algorithm MM-Cubing [16], which applies similar techniques just to the dense areas of the cube, taking into account the distribution of data in a way that avoids chunking.

Graph-Based algorithms represent a data cube as some specialized graph structure. They use such structures both in memory, for organizing data in a fashion that accelerates computation of the corresponding cube, and on disk, for compressing the final result and reducing storage-space requirements. Among the algorithms in this category, Dwarf [17] seems to be the strongest overall, since it is the only one that guarantees a polynomial time and space complexity with respect to dimensionality [18]. It is based on a highly compressed data structure that eliminates prefix and suffix redundancies efficiently. Prefix redundancy occurs when two or more tuples in the cube share the same prefix, i.e., the same values in the left dimensions; suffix redundancy, which is in some sense complementary to prefix redundancy, occurs when two or more cube tuples share the same suffix, i.e., the same values in the right dimensions and the aggregate measures. An advantage of Dwarf, as well as of the other graph-based methods, is that not only does its data structure store a data cube compactly, but it also serves as an index that can accelerate selective queries.

Approximate algorithms assume that data mining and OLAP applications do not require fine grained or absolutely precise results in order to capture trends and correlations in the data; hence, they store an approximate representation of the cube, trading accuracy for level of compression. Such algorithms exploit various techniques, inspired mainly from statistics, including histograms [14], wavelet transformations [19], and others.

Finally, note that some of the most popular industrial cube implementations include Microsoft SQL Server Analysis Services (http://www.microsoft.com/sql/technologies/analysis/default.mspx) and Hyperion Essbase, which has been bought by ORACLE in 2007 (http://www.oracle.com/hyperion).

Query Processing

The most important motivation for cube materialization is to provide low response times for OLAP queries. Clearly, construction of a highly-compressed cube is useless if the cube format inhibits good query answering performance. Therefore, efficiency during query processing should be taken into consideration as well when selecting a specific cube-construction algorithm and its corresponding storage format. Note that the latter determines to a great extent the access methods that can be used for retrieving data stored in the corresponding cube; hence, it strongly affects performance of query processing algorithms over cube data.

Intuitively, it seems that brute-force storage of an entire cube in an uncompressed format behaves best during query processing: in this case, every possible aggregation for every combination of dimensions is precomputed and the only cost required is that of retrieving the data stored in the lattice nodes participating in the query. On the other hand, query processing over compressed cubes seems to induce additional overhead for on-line computation or restoration of (possibly redundant) tuples that have not been materialized in the cube.

Nevertheless, the literature has shown that the above arguments are not always valid in practice. This is mostly due to the fact that indexing an uncompressed cube is nontrivial in real-world applications, whereas applying custom indexing techniques for some sophisticated, more compact representations has been found efficient [2]. Furthermore, storing data in specialized formats usually offers great opportunities for unique optimizations that allow a wide variety of query types to run faster over compressed cubes [2]. Finally, recall that several graph-based algorithms, e.g., Dwarf [17], store the cube in a way that is efficient with respect to both storage space and query processing time.

Incremental Maintenance

As mentioned earlier, in general, fact tables are dynamic in nature and change over time, mostly as new records are inserted in them. Aggregated data stored in a cube must follow the modifications in the corresponding fact table; otherwise, query answers over the cube will be inaccurate.

According to the most common scenario used in practice, data in a warehouse is periodically updated in a batch fashion. Clearly, the window of time that is required for the update process must be kept as narrow as possible. Hence, reconstruction of the entire cube from scratch is practically not a viable solution; techniques for incremental maintenance must be used instead.

Given a fact table, its corresponding cube, and a set of updates to the fact table that have occurred since the last cube update, let *delta cube* be the cube formed by the data corresponding to these updates. Most incremental-maintenance algorithms proposed in the literature for the cube follow a common strategy [20]: they separate the update process into the *propagation* phase, during which they construct the delta cube, and the *refresh* phase, during which they merge the delta cube and the original cube, in order to generate the new cube. Most of them identify the refresh phase as the most challenging one and use specialized techniques to accelerate it, taking into account the storage format of the underlying cube (some examples can be found in the literature [12,17]). There is at least one general algorithm, however, that tries to optimize the propagation phase [9]. It selects particular nodes of the delta cube for construction and properly uses them in order to update all nodes of the original cube.

Key Applications

Efficient implementation of the data cube is essential for OLAP applications in terms of performance, since they usually make an extensive use of aggregate queries.

Cross-references

- ▶ Data Warehouse
- ▶ Dimension
- ▶ Hierarchy
- ▶ Measure
- ▶ OLAP
- ▶ Snowflake Schema
- ▶ Star Schema

Recommended Reading

1. Agarwal S., Agrawal R., Deshpande P., Gupta A., Naughton J.F., Ramakrishnan R., and Sarawagi S. On the computation of multidimensional aggregates. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 506–521.
2. Beyer K.S. and Ramakrishnan R. Bottom-up computation of sparse and iceberg CUBEs. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 359–370.
3. Gray J., Bosworth A., Layman A., and Pirahesh H. Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 152–159.
4. Gupta H. Selection of views to materialize in a data warehouse. In Proc. 6th Int. Conf. on Database Theory, 1997, pp. 98–112.
5. Gupta H. and Mumick I.S. Selection of views to materialize under a maintenance cost constraint. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 453–470.
6. Harinarayan V., Rajaraman A., and Ullman J.D. Implementing data cubes efficiently. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 205–216.

7. Kotsis N. and McGregor D.R. Elimination of redundant views in multidimensional aggregates. In Proc. 2nd Int. Conf. Data Warehousing and Knowledge Discovery, 2000, pp. 146–161.
8. Lakshmanan L.V.S., Pei J., and Zhao Y. QC-Trees: an efficient summary structure for semantic OLAP. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 64–75.
9. Lee K.Y. and Kim M.H. Efficient incremental maintenance of data cubes. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 823–833.
10. Morfonios K. and Ioannidis Y. CURE for cubes: cubing using a ROLAP engine. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 379–390.
11. Morfonios K., Konakas S., Ioannidis Y., and Kotsis N. ROLAP implementations of the data cube. ACM Comput. Surv., 39(4), 2007.
12. Morfonios K. and Ioannidis Y. Supporting the Data cube Lifecycle: the Power of ROLAP. VLDB J., 17(4):729–764, 2008.
13. Mumick I.S., Quass D., and Mumick B.S. Maintenance of data cubes and summary tables in a warehouse. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 100–111.
14. Poosala V. and Ganti V. Fast approximate answers to aggregate queries on a data cube. In Proc. 11th Int. Conf. on Scientific and Statistical Database Management, 1999, pp. 24–33.
15. Ross K.A. and Srivastava D. Fast computation of sparse datacubes. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 116–125.
16. Shao Z., Han J., and Xin D. MM-Cubing: computing iceberg cubes by factorizing the lattice Space. In Proc. 16th Int. Conf. on Scientific and Statistical Database Management, 2004, pp. 213–222.
17. Sismanis Y., Deligiannakis A., Roussopoulos N., and Kotidis Y. Dwarf: shrinking the PetaCube. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 464–475.
18. Sismanis Y. and Roussopoulos N. The complexity of fully materialized coalesced cubes. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 540–551.
19. Vitter J.S. and Wang M. Approximate computation of multidimensional aggregates of sparse data using wavelets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 193–204.
20. Wang W., Feng J., Lu H., and Yu J.X. Condensed cube: an efficient approach to reducing data cube size. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 155–165.
21. Zhao Y., Deshpande P., and Naughton J.F. An array-based algorithm for simultaneous multidimensional aggregates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 159–170.

Cube Materialization

► Cube Implementations

Cube Precomputation

► Cube Implementations

Curation

► Biomedical Scientific Textual Data Types and Processing

Current Date

► Now in Temporal Databases[3]

Current Semantics

Michael H. Böhlen[1], Christian S. Jensen[2], Richard T. Snodgrass[3]
[1]Free University of Bozen-Bolzano, Bolzano, Italy
[2]Aalborg University, Aalborg, Denmark
[3]University of Arizona, Tucson, AZ, USA

Synonyms

Temporal upward compatibility

Definition

Current semantics constrains the semantics of non-temporal statements applied to temporal databases. Specifically, current semantics requires that non-temporal statements on a temporal database behave as if applied to the non-temporal database that is the result of taking the timeslice of the temporal database as of the current time.

Key Points

Current semantics [3] requires that queries and views on a temporal database consider the current information only and work exactly as if applied to a non-temporal database. For example, a query to determine who manages the high-salaried employees should consider the current database state only. Constraints and assertions also work exactly as before: they are applied to the current state and checked on database modification.

Database modifications are subject to the same constraint as queries: they should work exactly as if applied to a non-temporal database. Database modifications, however, also have to take into consideration that the current time is constantly moving forward. Therefore, the effects of modifications must persist into the future (until overwritten by a subsequent modification).

The definition of current semantics assumes a timeslice operator $\tau[t](D^t)$ that takes the snapshot of a temporal database D^t at time t. The timeslice operator takes the snapshot of all temporal relations in D^t and returns the set of resulting non-temporal relations.

Let *now* be the current time [2] and let t be a time point that does not exceed *now*. Let D^t be a temporal database instance at time t. Let $M_1,\ldots,M_n$, $n \geq 0$ be a sequence of non-temporal database modifications.

Let Q be a non-temporal query. Current semantics requires that for all Q, t, D^t, and $M_1,\ldots,M_n$ the following equivalence holds:

$$\begin{aligned} &Q(M_n(M_{n-1}(\ldots(M_1(D^t)\ldots)))) \\ &\quad = Q(M_n(M_{n-1}(\ldots(M_1(\tau[now](D^t)))\ldots))) \end{aligned}$$

Note that for $n = 0$ there are no modifications, and the equivalence becomes $Q(D^t) = Q(\tau[now](D^t))$, i.e., a non-temporal query applied to a temporal database must consider the current database state only.

An unfortunate ramification of the above equivalence is that temporal query languages that introduce new reserved keywords not used in the non-temporal languages they extend will violate current semantics. The reason is that the user may have previously used such a keyword as an identifier (e.g., a table name) in the database. To avoid being overly restrictive, it is reasonable to consider current semantics satisfied even when reserved words are added, as long as the semantics of all statements that do not use the new reserved words is retained by the temporal query language.

Temporal upward compatibility [1] is a synonym that focuses on settings where the original temporal database is the result of rendering a non-temporal database temporal.

Cross-references

- ▶ Nonsequenced Semantics
- ▶ Now in Temporal Databases
- ▶ Sequenced Semantics
- ▶ Snapshot Equivalence
- ▶ Temporal Database
- ▶ Temporal Data Models
- ▶ Temporal Query Languages
- ▶ Timeslice Operator

Recommended Reading

1. Bair J., Böhlen M.H., Jensen C.S., and Snodgrass R.T. Notions of upward compatibility of temporal query languages. Wirtschaftsinformatik, 39(1):25–34, February 1997.
2. Clifford J., Dyreson C., Isakowitz T., Jensen C.S., and Snodgrass R.T. On the Semantics of "NOW" in Databases. ACM Trans. Database Syst., 22:171–214, June 1997.
3. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, Los Altos, CA, 1999.

Current Time

▶ Now in Temporal Databases

Current Timestamp

▶ Now in Temporal Databases

Curse of Dimensionality

Lei Chen
Hong Kong University of Science and Technology, Hong Kong, China

Synonyms

Dimensionality curse

Definition

The *curse of dimensionality*, first introduced by Bellman [1], indicates that the number of samples needed to estimate an arbitrary function with a given level of accuracy grows exponentially with respect to the number of input variables (i.e., dimensionality) of the function.

For similarity search (e.g., nearest neighbor query or range query), the *curse of dimensionality* means that the number of objects in the data set that need to be accessed grows exponentially with the underlying dimensionality.

Key Points

The *curse of dimensionality* is an obstacle for solving dynamic optimization problems by backwards induction. Moreover, it renders machine learning problems complicated, when it is necessary to learn a state-of-nature from finite number data samples in a high dimensional feature space. Finally, the *curse of dimensionality* seriously affects the query performance for similarity search over multidimensional indexes because, in high dimensions, the distances from a query to its nearest and to its farthest neighbor are similar. This indicates that data objects tend to be close to the boundaries of the data space with the increasing dimensionality. Thus, in order to retrieve even a few answers to a nearest neighbor query, a large part of the data space should be searched, making the multi-dimensional indexes less efficient than a sequential scan of the data set, typically with dimensionality greater than 12 [2]. In order to break the *curse of dimensionality*, data objects are usually reduced to vectors in a lower dimensional space via some dimensionality reduction technique before they are indexed.

Cross-references

► Dimensionality Reduction

Recommended Reading

1. Bellman R.E. Adaptive Control Processes. Princeton University Press, Princeton, NJ, 1961.
2. Beyer K.S., Goldstein J., Ramakrishnan R., Shaft U. When is "Nearest Neighbor" Meaningful? In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 217–235.

Cursor

► Iterator

CW Complex

► Simplicial Complex

CWM

► Common Warehouse Metamodel

Cyclic Redundancy Check (CRC)

► Checksum and Cyclic Redundancy Check (CRC) Mechanism

D

DAC

► Discretionary Access Control

Daplex

TORE RISCH
Uppsala University, Uppsala, Sweden

Definition

Daplex is a query language based on a functional data model [1] with the same name. The Daplex data model represents data in terms of entities and functions. The Daplex data model is close to the entity-relationship (ER) model with the difference that relationships between entities in Daplex have a logical direction, whereas ER relationships are directionless. Unlike ER entities, relationships, and properties are all represented as functions in Daplex. Also, entity types are defined as functions without arguments returning sets of a built-in type ENTITY. The entity types are organized in a type/subtype hierarchy. Functions represent properties of entities and relationships among entities. Functions also represent derived information. Functions may be set-valued and are invertible. The database is represented as tabulated function extents. Database updates change the function extents.

The Daplex query language has been very influential for many other query languages, both relational, functional, and object oriented. Queries are expressed declaratively in an iterative fashion over sets similar to the FLWR semantics of the XQuery language. Daplex queries cannot return entities but a value returned from a query must always be a literal.

The query language further includes schema (function) definition statements, update statements, constraints, etc.

Key Points

Daplex functions are defined using a DECLARE statement, for example:

```
DECLARE name(Student) = STRING
```

Where "Student" is a user defined entity type and "STRING" is a built-in type. Set valued functions are declared by a " = > >" notation, e.g.,

```
DECLARE course(Student) = > > Course
```

Entity types are functions returning the built-in type ENTITY, for example:

```
DECLARE Person() = > > ENTITY
```

Inheritance among entity types is defined by defining entities as functions returning supertypes, for example:

```
DECLARE Student() = > > Person
```

Functions may be overloaded on different entity types.

Queries in Daplex are expressed using a FOR EACH – SUCH THAT – PRINT fashion similar to the FLWR semantics of XQuery. For example:

```
FOR EACH X IN Employee
SUCH THAT Salary(X) > Salary(Manager (X))
PRINT Name(X)
```

The PRINT statement is here not regarded as a side effect but rather as defining the result set from the query. Derived functions are defined though the DEFINE statement, e.g.,

```
DEFINE Course(Student) = > Course
SUCH THAT
    FOR SOME Enrollment
       Stud#(Student) = Stud#(Enrollment) AND
       Course#(Enrollment) = Course#(Course)
```

Daplex was first implemented in the Multibase system [2]. There, it was used as a multi-database query language to query data from several databases.

The P/FDM [1] data model and query language is close to Daplex. The query languages OSQL and AmosQL are also based on Daplex. These languages extend Daplex with object identifiers (OIDs) to represent actual entities and thus queries can there return entities as OIDs.

Cross-references

▶ AmosQL
▶ Functional Data Model
▶ OSQL
▶ P/FDM
▶ Query Language
▶ XPath/XQuery

Recommended Reading

1. Gray P.M.D., Kerschberg L., King P.J.H., and Poulovassilis A. (eds.). The Functional Approach to Data Management. Springer, Berlin, 2004.
2. Landers T. and Rosenberg R.L. An overview of Multibase. In Proc. 2nd Int. Symp. on Distributed Databases, 1982, pp. 153–184.
3. Shipman D.W. The functional data model and the data language DAPLEX. ACM Trans. Database Syst., 6(1):140–173, 1981.

DAS

▶ Direct Attached Storage

Data Acquisition

▶ Data Acquisition and Dissemination in Sensor Networks

Data Acquisition and Dissemination in Sensor Networks

Turkmen Canli, Ashfaq Khokhar
University of Illinois at Chicago, Chicago, IL, USA

Synonyms

Data gathering; Data collection; Data acquisition

Definition

Wireless sensor networks (WSNs) are deployed to monitor and subsequently communicate various aspects of physical environment, e.g., acoustics, visual, motion, vibration, heat, light, moisture, pressure, radio, magnetic, biological, etc. Data acquisition and dissemination protocols for WSNs are aimed at collecting information from sensor nodes and forwarding it to the subscribing entities such that maximum data rate is achieved while maximizing the overall network life time. The information can be simple raw data or processed using basic signal processing techniques such as filtering, aggregation/compression, event detection, etc.

Historical Background

Wireless sensor networks consist of tiny dispensable smart sensor nodes, with limited battery power and processing/communication capabilities. In addition, these networks also employ more powerful "sink" node(s) that collect information from the sensor nodes and facilitate interfacing with the outside computing and communication infrastructure. WSNs are configured to execute two fundamental tasks: information acquisition/collection at the sink nodes, and dissemination of information to the nodes across the network. Existing data acquisition and dissemination techniques have been investigated for different levels of application abstractions including: structured data collection [1,5,6,11,12] in a query-database paradigm, and raw data acquisition in for field reconstruction and event recognition at the sink nodes [2,3,4,7,9,10,13].

Foundations

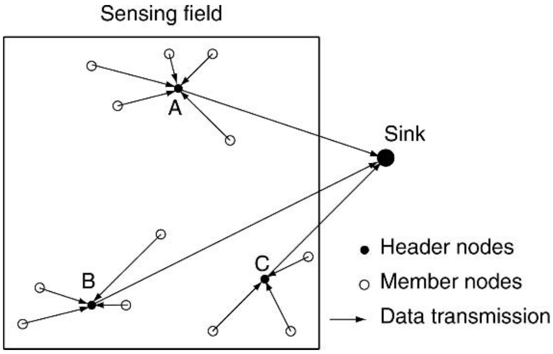
In WSNs, devices have limited battery life, which is generally considered non-replenishable, therefore the pertinent challenge is to design protocols and algorithms that maximize the network life time. In data acquisition and dissemination tasks, data volume is high therefore another optimization criteria is to increase throughput while reducing power consumption. Several data collection protocols aiming at data reduction using data transformation techniques have been suggested [2,10]. In [10], authors have proposed the use of wavelet compression to reduce data volume in structure monitoring WSN applications, thus resulting in low power consumption and reduced communication latency. Similar data transformation and compression techniques have been used to compute summary of the raw data [2].

In most of the data collection algorithms, the network nodes are organized into logical structures, and communication among the nodes and with the sink is realized using such logical structures. For example, in tree based data acquisition protocols, a collection tree is built that is rooted at the data collection center such as the sink node [8]. The dissemination of the data requests from the participating nodes and collection of data from the sensor nodes are accomplished using this tree. A cluster based data acquisition mechanism has been proposed in [3]. As shown in Fig. 1, nodes are organized into a fixed number of clusters, and nodes within each cluster dynamically elect a cluster head.

The data acquisition is carried out in two phases. In the first phase, cluster heads collect data from their cluster nodes. In the second phase, cluster heads send collected data to the nodes that have subscribed to the data. The cluster heads are re-elected to balance energy consumption among the nodes in the cluster. Zhang et al. [13] have proposed an adaptive cluster based data collection protocol that dynamically adjusts the number of cluster heads to the traffic load in the network. This dynamic traffic model is developed at the sink node.

In [7], network is divided into virtual grids and sensor nodes in each grid are classified as either gateway nodes or internal nodes. For example, in Fig. 2 nodes B, G are selected as gateway nodes that are responsible for transmitting data to nodes outside the grid. By doing so, data contention and redundant data transmission of a packet are reduced, which saves energy.

The common characteristic of all the aforementioned protocols is the pro-actively built routing infrastructure. As an alternative, authors in [4] have proposed the directed diffusion approach. The routing infrastructure is constructed on the fly. The sink node disseminates its interest to the network and gradients are set-up from nodes that match the sink's interest. There may be more than one path from a sensor node to the sink node. Sink nodes regulate data rate across all the paths.

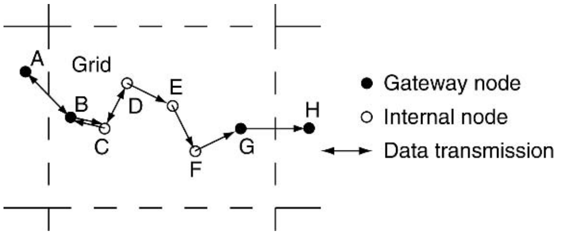


Data Acquisition and Dissemination in Sensor Networks. Figure 1. Clustering concept as proposed in LEACH [3].

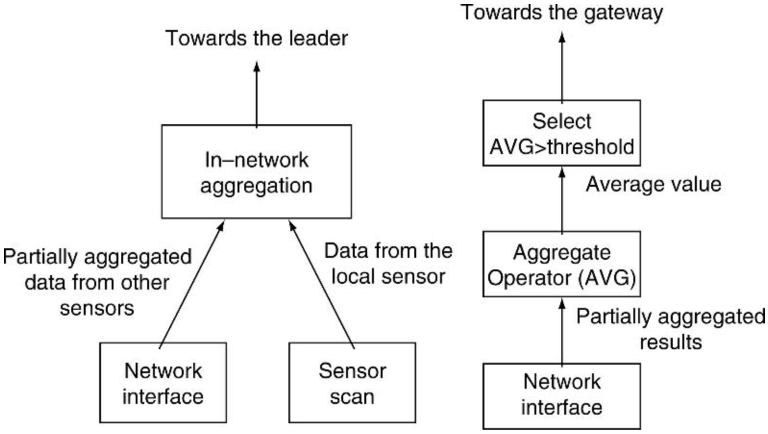
Data acquisition and dissemination techniques designed with a higher level of application abstraction model the sensor network as a distributed database system. In these techniques, data center disseminates its queries, and database operation such as "join" or "select" operations are computed distributively using sensor nodes that have the requested data. For example, in [12] a new layer, referred to as query layer, is proposed that is logically situated between the network and application layers of the network protocol stack. This layer processes descriptive queries and determines a power efficient execution plan that makes use of in-network processing and aggregation operations. An in-network aggregation is realized by packet merging or by incrementally applying aggregation operators such as min, max, count, or sum.

Cougar system [11] presents a framework for specifying a Query execution plan in WSNs. It allows specification of routing among sensor nodes and execution of aggregate operation over the collected data.

As depicted in Fig. 3, each sensor node samples the environment as specified by the query. According to the execution plan, sampled data is sent to a leader node, or together with the partially aggregated data received from other nodes, an aggregation operators is applied. The new partially aggregated value is then sent towards the leader node. Partial aggregation is possible only for the aggregation operators that can be computed incrementally. The volume of data is decreased by partial or incremental aggregation. The



Data Acquisition and Dissemination in Sensor Networks. Figure 2. Principle of LAF.



Data Acquisition and Dissemination in Sensor Networks. Figure 3. Query plan at a source and leader node [11].

responsibility of the leader node is to combine all the partially aggregated results and report it to the gateway node if the value exceed the set threshold.

In TinyDB framework [5], WSN is viewed as one big relational table. The columns of the table correspond to the type of phenomenon observed, i.e., humidity, temperature, etc. The aim is to reduce the power consumption during data acquisition by in-network processing of raw data. In other words, it addresses questions such as when data should be sampled for a particular query, which sensors should respond to a query, in which order sensors should sample the data, and how to achieve balance between in-network processing of the raw samples and collection of the raw samples at sink nodes without any processing. Moreover, the structured query language (SQL) is extended specifically for sensor network applications. Keywords such as, SAMPLE, ON EVENT and LIFETIME have been added to SQL language to facilitate realization of basic sensor network applications.

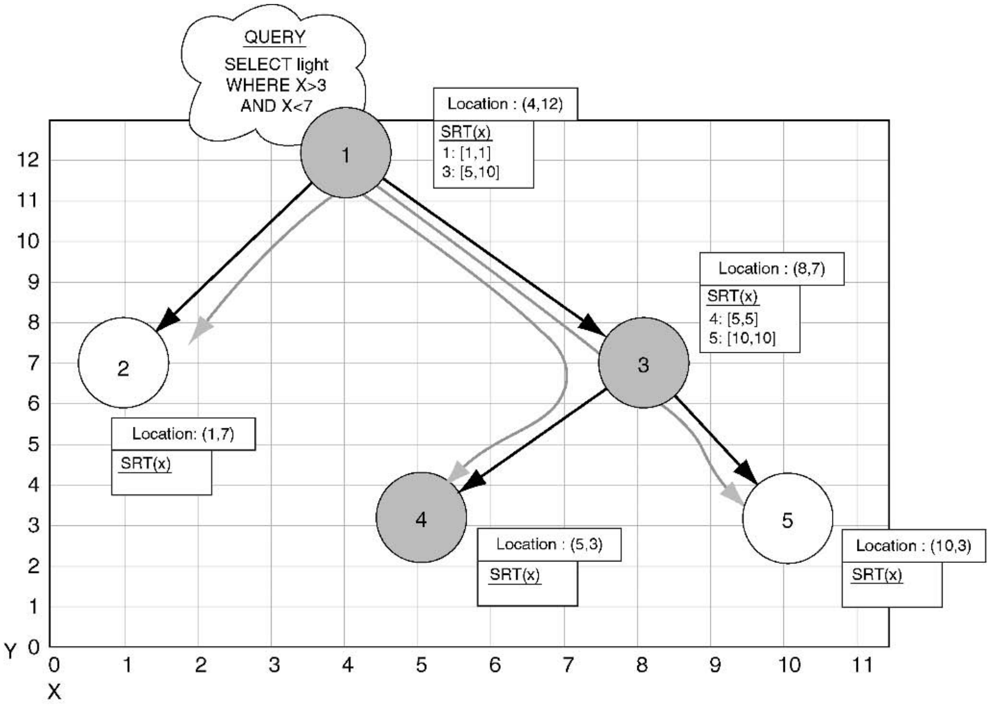
Through SAMPLE clause, the sampling rate of the sensors can be controlled. LIFETIME allows automatic sampling rate adjustment for a given lifetime. ON EVENT is used for trigger, i.e., query is executed only when the specified event occurs. A query sample shown below [5] illustrates the use of extended SQL. In this example query, the sampling period is set via introducing SAMPLE clause. The query planner needs meta data information regarding power consumption, sensing and communication costs to compute lifetime approximation and execute the LIFETIME clause. ON EVENT type queries may trigger multiple instances of same type of internal query.

```
SELECT COUNT(*)
FROM sensors AS s, recentLight AS r1
WHERE r1.nodeid=s.nodeid
AND s.light < r1.light
SAMPLE INTERVAL 10s
```

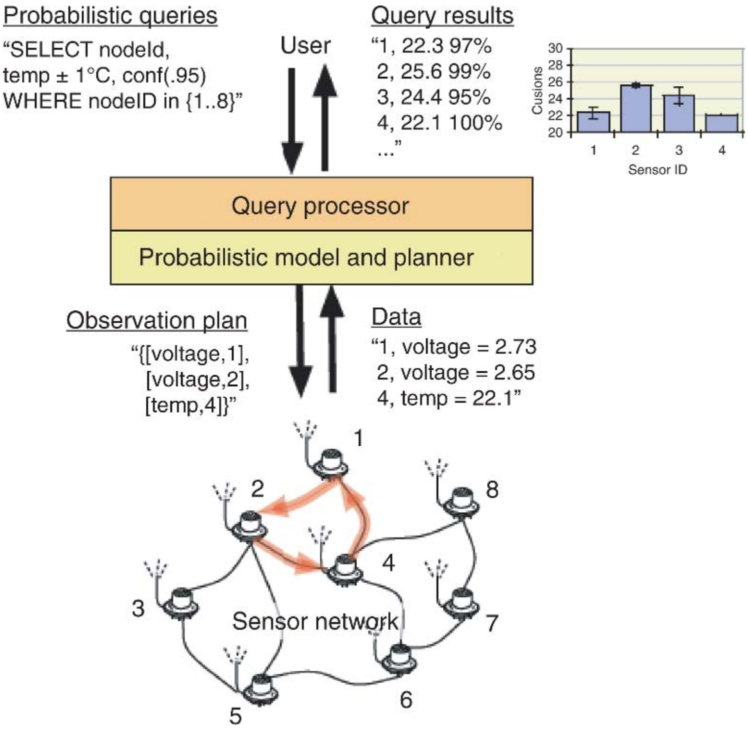
Optimization of the data sampling order can also reduce energy consumption significantly. Typically, a sensor node has more than one on-board sensor, i.e., nodes may have all temperature, humidity, pressure, etc. sensors on a single sensing platform. If in the query it is requested to report the temperature of the nodes where humidity value is greater than some threshold, it would be inefficient to simultaneously sample temperature and humidity values. The energy spent on sampling temperature values where humidity is less than the threshold could be saved by reordering the predicate evaluation [5].

The semantic tree (SRT) [5] is a mechanism that allows nodes to find out whether their children have the data for the incoming query. Every parent node stores the range of its children's values. Therefore when query arrives to a node it is not forwarded to those children that do not have the data. For instance in Fig. 4, node 1 will not send query request to node 2, similarly, node 3 will not send query to node 5.

Authors in [1] have used probabilistic models to facilitate efficient query processing and data acquisition on sensor networks. The idea is to build statistical model of the sensor readings from stored and current readings of the sensors. Whenever an SQL query is submitted to the network, constructed model is used to provide answers. Accuracy of the requested data can be specified by setting the



Data Acquisition and Dissemination in Sensor Networks. Figure 4. A semantic routing tree in use for a query. Gray arrows indicate flow of the query down the tree, gray nodes must produce or forward results in the query [5].



Data Acquisition and Dissemination in Sensor Networks. Figure 5. Model based querying in sensor networks [1].

confidence interval in the SQL statement. Figure 5 illustrates typical SQL query. Depending on error tolerance levels, response to the query may involve collecting information from every sensor node, for example, if 100% accuracy is required. On the other hand, if the error tolerance is high, query can be answered by only using the constructed model.

Key Applications

Building Health Monitoring, Micro-climate Monitoring, Habitat Monitoring, Hazardous Environment Monitoring.

Cross-references

- ▶ Ad-hoc Queries in Sensor Networks
- ▶ Continuous Queries in Sensor Networks
- ▶ Data Aggregation in Sensor Networks
- ▶ Data Compression in Sensor Networks
- ▶ Data Estimation in Sensor Networks
- ▶ Data Fusion in Sensor Networks
- ▶ Data Storage and Indexing in Sensor Networks
- ▶ Database Languages for Sensor Networks
- ▶ Model-Based Querying in Sensor Networks
- ▶ Query Optimization in Sensor Networks
- ▶ Sensor Networks

Recommended Reading

1. Deshpande A., Guestrin C., Madden S.R., Hellerstein J.M., and Hong W. Model-driven data acquisition in sensor networks. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 588–599.
2. Ganesan D., Greenstein B., Perelyubskiy D., Estrin D., and Heidemann J. An evaluation of multi-resolution storage for sensor networks. In Proc. 1st Int. Conf. on Embedded Networked Sensor Systems, 2003, pp. 89–102.
3. Heinzelman W.R., Chandrakasan A., and Balakrishnan H. Energy-efficient communication protocol for wireless microsensor networks. In Proc. 33rd Annual Hawaii Conf. on System Sciences, 2000, pp. 8020.
4. Intanagonwiwat C., Govindan R., Estrin D., Heidemann J., and Silva F. Directed diffusion for wireless sensor networking. IEEE/ACM Trans. Netw., 11(1):2–16, 2003.
5. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. The design of an acquisitional query processor for sensor networks. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 491–502.
6. Madden S.R., Franklin M.J., Hellerstein J.M., and Hong W. TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst., 30(1):122–173, 2005.
7. Sabbineni M.H. and Chakrabarty S.M.K. Location-aided flooding: an energy-efficient data dissemination protocol for wireless sensor networks. IEEE Trans. Comput., 54(1):36–46, 2005.
8. Szewczyk R., Osterweil E., Polastre J., Hamilton M., Mainwaring A., and Estrin D. Habitat monitoring with sensor networks. Commun. ACM, 47(6):34–40, 2004.
9. Xi Y., Yang W., Yamauchi N., Miyazaki Y., Baba N., and Ikeda H. Real-time data acquisition and processing in a miniature wireless monitoring system for strawberry during transportation. In Proc. Int. Technical Conf. of IEEE Region 10 (Asia Pacific Region), 2006.
10. Xu N., Rangwala S., Chintalapudi K.K., Ganesan D., Broad A., Govindan R., and Estrin D. A wireless sensor network for structural monitoring. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 13–24.
11. Yao Y. and Gehrke J. The cougar approach to in-network query processing in sensor networks. ACM SIGMOD Rec., 31 (3):9–18, 2002.
12. Yao Y. and Gehrke J. Query processing in sensor networks, 2003.
13. Zhan X., Wang H., and Khokhar A. An energy-efficient data collection protocol for mobile sensor networks. Vehicular Technology Conference, 2006, pp. 1–15.

Data Aggregation in Sensor Networks

Jun Yang[1], Kamesh Munagala[1], Adam Silberstein[2]

[1]Duke University, Durham, NC, USA

[2]Yahoo! Research Silicon Valley, Santa Clara, CA, USA

Definition

Consider a network of N sensor nodes, each responsible for taking a reading v_i $(1 \leq i \leq N)$ in a given epoch. The problem is to compute the result of an aggregate function (cf. *Aggregation*) over the collection of all readings $v_1, v_2, \ldots, v_N$ taken in the current epoch. The final result needs to be available at the base station of the sensor network. The aggregate function ranges from simple, standard SQL aggregates such as SUM and MAX, to more complex aggregates such as top-k, median, or even a contour map of the sensor field (where each value to be aggregated is a triple $\langle x_i, y_i, z_i \rangle$, with x_i and y_i denoting the location coordinates of the reading z_i).

In battery-powered wireless sensor networks, energy is the most precious resource, and radio communication is often the dominant consumer of energy. Therefore, in this setting, the main optimization objective is to minimize the total amount of communication needed in answering an aggregation query. A secondary objective is to balance the energy consumption across all sensor nodes, because the first node to run out of battery may render a large portion of the network inaccessible.

There are many variants of the above problem definition. For example, the aggregation query may be *continuous* and produce a new result for each epoch; not all nodes may participate in aggregation; the result may be needed not at the base station but at other nodes in the network. Some of these variants are further discussed below.

Historical Background

Early deployments of wireless sensor networks collect and report all data to the base station without summarization or compression. This approach severely limits the scale and longevity of sensor networks, because nodes spend most their resources forwarding data on behalf of others. However, many applications do not need the most detailed data; instead, they may be interested in obtaining a summary view of the sensor field, monitoring outlier or extreme readings, or detecting events by combining evidence from readings taken at multiple nodes. Data aggregation is a natural and powerful construct for applications to specify such tasks.

Data aggregation is supported by *directed diffusion* [9], a data-centric communication paradigm for sensor networks. In directed diffusion, a node diffuses its interests for data in the network. Then, nodes with data matching the interests return the relevant data along the reverse paths of interest propagation. Intermediate nodes on these paths can be programmed to aggregate relevant data as it converges on these nodes. Systems such as *TinyDB* and *Cougar* take a database approach, providing a powerful interface for applications to pose declarative queries including aggregation over a sensor network, and hiding the implementation and optimization details from application programmers. The seminal work by Madden et al. [12] on *TAG* (*Tiny AGgregation*) is the first systematic study of database-style aggregation in sensor networks. One of the first efforts at supporting more sophisticated aggregation queries beyond SQL aggregates is the work by Hellerstein et al. [8], which shows how to extend TAG to compute contour maps, wavelet summaries, and perform vehicle tracking. Since these early efforts, the research community has made significant progress in sensor data aggregation; some of the developments are highlighted below.

Foundations

The key to efficient sensor data aggregation is in-network processing. On a typical sensor node today, the energy cost of transmitting a byte over wireless radio is orders-of-magnitude higher than executing a CPU instruction. When evaluating an aggregate query, as data converges on an intermediate node, this node can perform aggregate computation to reduce the amount of data to be forwarded, thereby achieving a favorable tradeoff between computation and communication.

To illustrate, consider processing a simple SUM aggregate with TAG [12]. TAG uses a routing tree rooted at the base station spanning all nodes in the network. During aggregation, each node listens to messages from its children in the routing tree, computes the sum of all values in these messages and its own reading, and then transmits this result – which equals the sum of all readings in the subtree rooted at this node – to its parent. To conserve energy, each node only stays awake for a short time interval to listen, compute, and transmit. To this end, TAG coordinates the nodes' communication schedules: the beginning of a parent node's interval must overlap with the end of its children's intervals, so that the parent is awake to receive children's transmissions. Overall, to compute SUM, each node needs to send only one constant-size message, so the total amount of communication is $\Theta(N)$. In comparison, for the naïve approach, which sends all readings to the root, the amount of data that needs to be forwarded by a node increases closer to the root, and the total amount of communication can be up to $\Theta(Nd)$, where d is the depth of the routing tree. Clearly, in-network aggregation not only decreases overall energy consumption, but also balances energy consumption across nodes.

The above algorithm can be generalized to many other aggregates. Formally, an aggregate function can be implemented using three functions: an *initializer* f_i converts an input value into a *partial aggregate record; a merging function* f_m combines two partial aggregate records into one; finally, an *evaluator* f_e computes the final result from a partial aggregate record. During aggregation, each node applies f_i to its own reading; each non-leaf node invokes f_m to merge partial aggregate records received from its children with that of its own; the root uses f_e to compute the aggregate result from the final partial aggregate record. As an example, standard deviation can be computed (in theory, without regard to numerical stability) using the following functions, where the partial aggregate record is a triple $\langle s, r, n\rangle$ consisting of a sum (s), a sum of squares (r), and a count (n):

$$f_i(v) = \langle v, v^2, 1\rangle,$$

$$f_m(\langle s_1, r_1, n_1\rangle, \langle s_2, r_{2,} n_2\rangle) = \langle s_1 + s_2, r_1 + r_2, n_1 + n_2\rangle,$$

$$f_e(\langle s, r, n\rangle) = \frac{1}{n}\sqrt{nr - s^2}.$$

In general, one cannot expect the partial aggregate record to be of constant size for arbitrary aggregate functions. Consider the following examples. For a top-k aggregate, which finds the k largest values, a partial aggregate record needs to be of size $\Theta(k)$. For exact median computation, the size requirement becomes $\Theta(N)$, which is no better than the naïve approach of sending all readings to the root. For contour map construction, the size of the partial aggregate records depends on the complexity of the field being sensed, and can be $\Theta(N)$ in the worst case.

Approximate Aggregation

Approximation is a popular and effective technique for bounding the communication cost in evaluating complex aggregation functions. The basic idea is to replace an exact partial aggregate record with an approximate partial aggregate record that consumes less space and is therefore cheaper to send. An early example of applying this idea is the work by Hellerstein et al. [8]. Since then, a diverse collection of approximation methods has been developed for many aggregate functions; a few illustrative examples are given below.

To compute order-statistics (e.g., quantile queries including top-k and median), Greenwald and Khanna [7] propose a technique based on *ε-approximate quantile summaries*. An ε-approximate quantile summary for a collection S of sensor readings is an ordered subset $\{q_i\}$ of S, where each q_i is associated with a lower bound $r\min_i$ and an upper bound $r\min_i$ on q_i's rank within S, and the difference between $r\max_{i+1}$ and $r\min_i$ is no greater than $2\varepsilon|S|$. Any quantile query over S can be answered instead on this summary within an additive rank error of $\varepsilon|S|$. Specifically, a query requesting the r-th ranked reading can be answered by returning q_j from the summary, where $r - \varepsilon|S| \leq r\min_j$ and $r\max \leq r + \varepsilon|S|$. Greenwald and Khanna represent a partial aggregate record sent up from a node u by a set of quantile summaries – at most one for each class numbered 1 through $\log N$ – which together disjointly cover all readings in the subtree rooted at u; the summary for class i covers between 2^i and $2^{i+1}-1$ readings using at most $(\log N/\varepsilon + 1)$ of these readings. Each sensor node starts with an $\varepsilon/2$-approximate summary of all its local readings. Each intermediate node merges summaries from its children together with its own summary into up to $\log N$ merged summaries, prunes each of them down to the maximum size allowed, and then sends them up to the parent. Finally, the root merges all summaries into a single one and prunes it down to $(\log N/\varepsilon + 1)$ readings. Although pruning introduces additional error, the use of per-class summaries bounds the error in a class-i summary to $\varepsilon/2 + (i/(2\log N/\varepsilon))$, which in turn allows the error in the final summary to be bounded by ε. Overall, the communication cost incurred by each node during aggregation is only $o(\log^2 N/\varepsilon)$.

Silberstein et al. [15] approach the problem of computing top-k aggregates using a very different style of approximation. Instead of having each node always sending the top k readings in its subtree, a node sends only the top k' readings among its local reading and those received from its children, where $k' \leq k$. The appropriate setting of k' for each node is based on the samples of past sensor readings, or, more generally, a model capturing the expected behavior of sensor readings. Intuitively, a subtree that tends to contribute few of the top values will be allotted a smaller k'. Unlike the ε-approximate quantile summaries, which provide hard accuracy guarantees, the accuracy of this approach depends on how well the past samples or the model reflect the current behavior of readings. Nevertheless, the approach can be augmented by transmitting additional information needed to establish the correctness of some top-k answers, thereby allowing the approximation quality to be assessed.

As a third example, the contour map of a sensor field is a complex spatial aggregate defined over not only values but also locations of the sensor readings. In this case, the partial aggregate record produced by a node is a compact, usually lossy, representation of the contour map encompassing all readings in this node's subtree. In Hellerstein et al. [8], each contour in the map is represented by an orthogonal polygon whose edges follow pre-imposed 2-d rectangular grid lines. This polygon is obtained by starting with the minimum bounding rectangle of the contour, and then repeatedly subtracting the largest-area rectangle that does not contain any point in the contour, until a prescribed limit on the number of vertices is reached. Gandhi et al. [5] use general polygons instead of orthogonal ones. During aggregation, each node constructs and sends to its parent an approximate description of its contour map consisting of up to k possibly disconnected line segments. The root then connects and untangles such line segments to obtain the final contour map. It is shown that the approximation error in the k -segment representation produced by distributed aggregation is within a constant factor of the smallest possible error attainable by any k segments, and it is conjectured that the resulting contour map has size $o(k)$.

Duplicate-Insensitive Aggregation

Approximate aggregation methods based on *duplicate-insensitive synopses* are especially worth noting because of their resiliency against failures, which are common in sensor networks. Tree-based aggregation techniques are vulnerable to message failures. If a message carrying the partial aggregate record from a node fails, all information from that subtree will be lost, resulting in significant error. Sending the same message out on multiple paths towards the base station decreases the chance of losing all copies, and is a good solution for aggregation functions such as MAX. However, for other aggregation functions whose results are sensitive to duplicates in their inputs, e.g., SUM and COUNT, having multiple copies of the same partial aggregation record causes a reading to participate multiple times in aggregation, leading to incorrect results. In general, if an aggregation method is *order- and duplicate-insensitive (ODI)* [14], it can be implemented with more failure-resilient routing structures such as directed acyclic graphs, without worrying about the duplicates they introduce. The challenge, then, is in designing ODI aggregation methods for duplicate-sensitive aggregation functions.

Duplicate-insensitive synopses provide the basis for computing many duplicate-sensitive aggregation functions approximately in an ODI fashion. This approach is pioneered by Considine et al. [1] and Nath et al. [14]. To illustrate the idea, consider COUNT, which is duplicate-sensitive. Nodes in the sensor network are organized into rings centered at the base station, where the i -th ring includes all nodes at i hops away from the base station. A partial aggregate record is a Flajolet-Martin sketch, a fixed-size bit-vector for estimating the number of distinct elements in a multi-set. This sketch is duplicate-insensitive by design: conceptually, it is obtained by hashing each element to a bitmap index (using an exponential hash function) and setting that bit to one. During aggregation, each node first produces a sketch for its local sensors. A node in the i-th ring receives sketches from its neighbors (i.e., nodes within direct communication distance) in the $(i+1)$-th ring, takes the bitwise OR of all these sketches and its own, and broadcasts the result sketch to all its neighbors in the $(i-1)$-th ring. Taking advantage of broadcast communication, each node sends out only one message during aggregation, but the information therein can reach the base station via multiple paths, boosting reliability. The overall COUNT can be estimated accurately with high probability using sketches of size $\Theta(\log N)$.

The failure-resiliency feature comes with two costs in the above approach: the final answer is only approximate, and the size of each message is larger than the tree-based exact-aggregation approach. Manjhi et al. [13] have developed an adaptive, hybrid strategy that combines the advantages of the two approaches by applying them to different regions of the network, and dynamically exploring the tradeoffs between the two approaches.

Temporal Aspects of Aggregation

The preceding discussion has largely ignored the temporal aspects of aggregation. In practice, aggregation queries in sensor networks are often *continuous*. In its simplest form, such a query executes continuously over time and produces, for each epoch, an aggregate result computed over all readings acquired in this epoch. A key optimization opportunity is that sensor readings often are temporally correlated and do not change haphazardly over time. Intuitively, rather than re-aggregating from scratch in every epoch, evaluation efforts should focus only on relevant changes since the last epoch.

An effective strategy for implementing the above intuition is to install on nodes local constraints that dictate when changes in subtrees need to be reported. These constraints carry memory about past readings and filter out reports that do not affect the current aggregate result, thereby reducing communication. For example, to compute MAX continuously, Silberstein et al. [16] set a threshold at each node, which is always no less than its local reading and its children's thresholds. A node sends up the current maximum value in its subtree only if that value exceeds the threshold; the threshold is then adjusted higher. If the current global maximum falls, nodes with thresholds higher than the new candidate maximum must be visited to find the new maximum; at the same time, their thresholds are adjusted lower. Thresholds control the tradeoff between reporting and querying costs, and can be set adaptively at runtime. Alternatively, optimum settings can be found by assuming adversarial data behavior or using predictive models of data behavior.

As another example, consider continuous SUM. Unlike MAX, even a small change in one individual reading affects the final SUM result. Approximation is thus needed to do better than one message per node per epoch. Deligiannakis et al. [4] employ an interval constraint at each node, which bounds the sum of all current readings within the subtree. In each epoch, the node sends its estimate of this partial sum to its parent only if this value falls outside its interval; the interval then recenters at this value. The length of the interval controls the error allowance for the subtree. Periodically, based on statistics collected, the interval lengths are adjusted by recursively redistributing the total error allowed in the final result to all nodes.

Continuous versions of more complex queries, for which approximation is needed to reduce the size of partial aggregate records, have also been studied. For example, Cormode et al. [2] show how to continuously compute ε-approximate quantile summaries using a hierarchy of constraints in the network to filter out insignificant changes in subtree summaries. As with other techniques described above, a major technical challenge lies in allocating error tolerance to each constraint; optimum allocation can be computed for predictive models of data behavior. Xue et al. [17] consider the continuous version of the contour map query. Instead of sending up the entire partial aggregate record (in this case, a contour map for the subtree), only its difference from the last transmitted version needs to be sent.

In the continuous setting, aggregation can apply not only spatially to the collection of readings acquired in the same epoch, but also temporally over historical data (e.g., recent readings in a sliding window). Cormode et al. [3] consider the problem of continuously computing time-decaying versions of aggregation functions such as SUM, quantiles, and heavy hitters. The contribution of a reading taken at time t' to the aggregate result at the current time $t' = t' + \Delta$ is weighted by a user-defined *decay function* $f(\Delta) \leq 0$, which is non-increasing with Δ. The solution is based on duplicate-insensitive sketching techniques, and it approximates a general decay function using a collection of sliding windows of different lengths.

Other Aspects of Aggregation in Sensor Networks

Besides the above discussion, there are many other aspects of sensor data aggregation that are not covered by this entry; some of them are outlined briefly below.

Most techniques presented earlier opportunistically exploit in-network processing, whenever two partial aggregate records meet at the same node following their standard routes to the base station. More generally, routing can be made aggregation-driven [11] by encouraging convergence of data that can be aggregated more effectively (e.g., the merged partial aggregate record uses less space to achieve the required accuracy).

Oftentimes, only a sparse subset of nodes contribute inputs to aggregation, and this subset is not known a priori, e.g., when aggregation is defined over the output of a filter operation evaluated locally at each node. The challenge in this scenario is to construct an ad hoc aggregation tree of high quality in a distributed fashion [6].

Finally, for some applications, the final aggregate result is needed at all nodes in the sensor network as opposed to just the base station. *Gossiping* is an effective technique for this purpose [10], which relies only on local communication and does not assume any particular routing strategy or topology.

Key Applications

Aggregation is a fundamental query primitive indispensible to many applications of sensor networks. It is widely used in expressing and implementing common sensor network tasks such as summarization, compression, monitoring, and event detection. Even for applications that are interested in collecting all detailed sensor readings, aggregation can be used in monitoring system and data characteristics, which support maintenance of the sensor network and optimization of its operations.

Cross-references

- ▶ Ad-Hoc Queries in Sensor Networks
- ▶ Aggregation
- ▶ Continuous Queries in Sensor Networks

▶ Data Compression in Sensor Networks
▶ Data Fusion in Sensor Networks

Recommended Reading

1. Considine J., Li F., Kollios G., and Byers J. Approximate aggregation techniques for sensor databases. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 449–460.
2. Cormode G., Garofalakis M., Muthukrishnan S., and Rastogi R. Holistic aggregates in a networked world: distributed tracking of approximate quantiles. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 25–36.
3. Cormode G., Tirthapura S., and Xu B. Time-decaying sketches for sensor data aggregation. In Proc. ACM Symposium on Principles of Distributed Computing, 2007, pp. 215–224.
4. Deligiannakis A., Kotidis Y., and Roussopoulos N. Hierarchical in-network data aggregation with quality guarantees. In Advances in Database Technology, In Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 658–675.
5. Gandhi S., Hershberger J., and Suri S. Approximate isocontours and spatial summaries for sensor networks. In Proc. 6th Int. Symp. Inf. Proc. in Sensor Networks, 2007, pp. 400–409.
6. Gao J., Guibas L.J., Milosavljevic N., and Hershberger J. Sparse data aggregation in sensor networks. In Proc. 6th Int. Symp. Inf. Proc. in Sensor Networks, 2007, pp. 430–439.
7. Greenwald M. and Khanna S. Power-conserving computation of order-statistics over sensor networks. In Proc. 23rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2004, pp. 275–285.
8. Hellerstein J.M., Hong W., Madden S., and Stanek K. Beyond average: toward sophisticated sensing with queries. In Proc. 2nd Int. Workshop Int. Proc. in Sensor Networks, 2003, pp. 63–79.
9. Intanagonwiwat C., Govindan R., and Estrin D. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In Proc. 6th Annual Int. Conf. on Mobile Computing and Networking, 2000, pp. 56–67.
10. Kempe D., Dobra A., and Gehrke J. Gossip-based computation of aggregate information. In Proc. 44th Annual Symp. on Foundations of Computer Science, 2003, pp. 482–491.
11. Luo H., Y. Liu, and S. Das. Routing correlated data with fusion cost in wireless sensor networks. IEEE Transactions on Mobile Computing, 11(5):1620–1632, 2006.
12. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. TAG: a tiny aggregation service for ad-hoc sensor networks. In Proc. 5th USENIX Symp. on Operating System Design and Implementation, 2002.
13. Manjhi A., Nath S., and Gibbons P.B. Tributaries and deltas: efficient and robust aggregation in sensor network streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 287–298.
14. Nath S., Gibbons P.B., Seshan S., and Anderson Z.R. Synopsis diffusion for robust aggregation in sensor networks. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 250–262.
15. Silberstein A., Braynard R., Ellis C., and Munagala K. A sampling-based approach to optimizing top-k queries in sensor networks. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
16. Silberstein A., Munagala K., and Yang J. Energy-efficient monitoring of extreme values in sensor networks. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006.
17. Xue W., Luo Q., Chen L., and Liu Y. Contour map matching for event detection in sensor networks. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 145–156.

Data Analysis

▶ Data Mining

Data Anomalies

▶ Data Conflicts

Data Broadcasting, Caching and Replication in Mobile Computing

PANOS K. CHRYSANTHIS[1], EVAGGELIA PITOURA[2]
[1]University of Pittsburgh, Pittsburgh, PA, USA
[2]University of Ioannina, Ioannina, Greece

Synonyms

Data dissemination; Push/pull delivery; Data copy

Definition

Mobile computing devices (such as portable computers or cellular phones) have the ability to communicate while moving by being connected to the rest of the network through a wireless link. There are two general underlying infrastructures: single-hop and multi-hop ones. In *single-hop* infrastructures, each mobile device communicates with a stationary host, which corresponds to its point of attachment to the wired network. In *multi-hop* infrastructures, an ad-hoc wireless network is formed in which mobile hosts participate in routing messages among each other. In both infrastructures, the hosts between the source (or sources) and the requester of data (or data sink) form a dissemination tree. The hosts (mobile or stationary) that form the dissemination tree may store data and participate in computations towards achieving *in network* processing. Challenges include [14], (i) *intermittent connectivity*, which refers to both short and long periods of network unavailability, (ii) *scarcity of resources*, including storage and battery life, and (iii) *mobility* itself.

To handle these challenges, data items may be stored locally (cached or replicated) at the requester or at the intermediate nodes of the dissemination tree. Cache and replication aim at increasing availability in the case of network disconnections or host failures as well as at handling intermittent connectivity. Mobility introduces additional challenges in maintaining cache and replica consistency and in replica placement protocols. Wireless data delivery in both infrastructures physically supports broadcasting. This broadcast facility has been used for providing a push-mode of data dissemination where a server broadcasts data to a large client population often without an explicit request from the clients. Issues addressed by related research include broadcast scheduling and organization (i.e., which items to broadcast and in which order), indexing broadcast data, and update propagation.

Historical Background

Mobile computing can be traced back to file systems and the need for disconnected operations in the late 1980s. With the rapid growth in mobile technologies and the cost effectiveness in deploying wireless networks in the 1990s, the goal of mobile computing was the support of AAA (anytime, anywhere and any-form) access to data by users from their portable computers, mobile phones and other devices with small displays and limited resources. These advances motivated research in data management in the early 1990s.

Foundations

Data Broadcasting

Many forms of wireless network infrastructures rely on broadcast technology to deliver data to large client populations. As opposed to point-to-point data delivery, broadcast delivery is scalable, since a single broadcast response can potentially satisfy many clients simultaneously. There are two basic modes of broadcast data delivery: pull-based and push-based. With *push-based* data delivery, the server sends data to clients without an explicit request. With *pull-based* or *on-demand* broadcast delivery, data are delivered only after a specific client request. In general, access to broadcast data is sequential with clients monitoring the broadcast channel and retrieving any data items of interest as they arrive. The smallest access unit of broadcast data is commonly called a *bucket* or *page*.

Scheduling and Organization

A central issue is determining the content of the broadcast or *broadcast scheduling*. Scheduling depends on whether we have on demand, push or hybrid delivery. In on-demand broadcast, there is an up-link channel available to clients to submit requests. The item to be broadcast next is chosen among those for which there are pending requests. Common heuristics for on-demand scheduling include First Come First Served and Longest Wait First [7]. The $R \times W$ strategy selects the data item with the maximal $R \times W$ value, where R is the number of pending requests for an item and W the amount of time that the oldest pending request for that item has spent waiting to be served [3]. More recent schemes extended $R \times W$ to consider the semantics of the requested data and applications such as subsumption properties in data cubes [15]. Push-based broadcast scheduling assumes a-priori knowledge of client access distributions and prepares an off-line schedule. Push-based data delivery is often periodic. In hybrid broadcast, the set of items is partitioned, so that some items are pushed, i.e., broadcast continuously, and the rest are pulled, i.e., broadcast only after being requested [2]. Commonly, the partition between push and pull data is based on popularity with the most popular items being pushed periodically and the rest delivered on demand. One problem is that for push items, there is no way to detect any changes in their popularity. One solution is to occasionally stop broadcasting some pushed items. This forces clients to send explicit requests for them, which can be used to estimate their popularity [16]. An alternative that avoids flooding of requests requires a percentage of the clients to submit an explicit request irrespective of whether or not a data item appears on the broadcast [5].

The organization of the broadcast content is often called *broadcast program*. In general, broadcast organizations can be classified as either *flat* where each item is broadcast exactly once or *skewed* where an item may appear more than once. One can also distinguish between *clustered* organizations, where data items having the same or similar values at some attribute appear consecutively, and *non-clustered* ones, where there is no such correlation. In skewed organizations, the broadcast frequency of each item depends on its popularity. For achieving optimal access latency or response time, it was shown that (i) the relative number of appearances of items should be proportional to the square root of their access probabilities and (ii) successive broadcasts of the same item should be at equal distances [7]. It was also shown that the *Mean Aggregate Access* (MAD) policy that selects to broadcast next the item whose access probability × the interval since its last broadcast is the highest achieves close to optimal response time [17]. Along these lines, a practical skewed push broadcast organization is that of *broadcast disks* [1]. Items are assigned to virtual disks with different "speeds" based on their popularity with popular items being assigned to fast disks. The spin speed of each disk is simulated by the frequency with which the items assigned to it are broadcast. For example, the fact that a disk D_1 is three times faster than a disk D_2, means that items assigned to D_1 are broadcast three times as often as those assigned to D_2. To achieve this, each disk is split into smaller equal-sized units called chunks, where the number of chunks per disk is inversely proportional to the relative frequence of the disk. The broadcast program is generated by broadcasting one chunk from each disk and cycling through all the chunks sequentially over all the disks.

Indexing

To reduce energy consumption, a mobile device may switch to *doze* or *sleep* mode when inactive. Thus, research in wireless broadcast also considers reducing the *tuning time* defined as the amount of time a mobile client remains active listening to the broadcast. This is achieved by including index entries in the broadcast so that by reading them, the client can determine when to tune in next to access the actual data of interest. Adding index entries increases the size of the broadcast and thus may increase access time. The objective is to develop methods for allocating index entries together with data entries on the broadcast channel so that both access and tuning time are optimized. In *(1, m) indexing* [18], an index for all data items is broadcast following every fraction (1/*m*) of the broadcast data items. *Distributed indexing* [18] improves over this method by instead of replicating the whole index *m* times, each index segment describes only the data items that follow it. Following the same principles, different indexing schemes have been proposed that support different query types or offer different trade-offs between access and tuning time. Finally, instead of broadcasting an index, *hashing-based techniques* have also been applied.

Data Caching and Replication

A mobile computing device (such as a portable computer or cellular phone) is connected to the rest of the network through a wireless link. Wireless communication has a double impact on the mobile device since the limited bandwidth of wireless links increases the response times for accessing remote data from a mobile host and transmitting as well as receiving of data are high energy consumption operations. The principal goal of caching and replication is to store appropriate pieces of data locally at the mobile device so that it can operate on its own data, thus reducing the need for communication that consumes both energy and bandwidth. Several cost-based caching policies along the principles of *greedy-dual* ones have been proposed that consider energy cost.

In the case of broadcast push, the broadcast itself can be viewed as a "cache in the air." Hence, in contrast to traditional policies, performance can be improved by clients caching those items that are accessed frequently by them but are not popular enough among all clients to be broadcast frequently. For instance, a cost-based cache replacement policy selects as a victim the page with the lowest *p*/*x* value, where *p* is the local access probability of the page and *x* its broadcast frequency [1]. Prefetching can also be performed with low overhead, since data items are broadcast anyway. A simple prefetch heuristic evaluates the worth of each page on the broadcast to determine whether it is more valuable than some other page in cache and if so, it swaps the cache page with the broadcast one.

Replication is also deployed to support *disconnected operation* that refers to the autonomous operation of a mobile client, when network connectivity becomes either unavailable (for instance, due to physical constraints), or undesirable (for example, for reducing power consumption). Preloading or prefetching data to sustain a forthcoming disconnection is often termed *hoarding*. Optimistic approaches to consistency control are typically deployed that allow data to be accessed concurrently at multiple sites without a priori synchronization between the sites, potentially resulting in short term inconsistencies. At some point, operations performed at the mobile device must be synchronized with operations performed at other sites. Synchronization depends on the level at which correctness is sought. This can be roughly categorized as replica-level correctness and transaction-level correctness. At the *replica level*, correctness or coherency requirements are expressed per item in terms of the allowable divergence among the values of the copies of each item. At the *transaction level*, the strictest form of correctness is achieved through global serializability that requires the execution of all transactions running at mobile and stationary hosts to be equivalent to some serial execution of the same transactions. With regards to update propagation with *eager replication*, all copies of an item are synchronized within a single transaction, whereas with *lazy replication*, transactions for keeping replica coherent execute as separate, independent database transactions after the original transaction commits.

Common characteristics of protocols for consistency in mobile computing include:

- The propagation of updates performed at the mobile site follows in general lazy protocols.
- Reads are allowed at the local data, while updates of local data are tentative in the sense that they need to be further validated before commitment.
- For integrating operations at the mobile hosts with transactions at other sites, in the case of replica-level consistency, copies of each item are reconciled following some conflict resolution protocol. At the transaction-level, local transactions are validated against some application or system level criterion. If the criterion is met, the transaction is committed. Otherwise, the execution of the transaction is either aborted, reconciled or compensated.

Representative approaches along these lines include *isolation-only transactions* in Coda, *mobile open-nested transactions* [6], *two-tier replications* [8], *two-layer transactions* [10] and *Bayou* [9].

When local copies are read-only, a central issue is the design of efficient protocols for disseminating server updates to mobile clients. A server is called *stateful*, if it maintains information about its clients and the content of their caches and *stateless* otherwise. A server may use broadcasting to efficiently propagate update reports to all of its clients. Such update reports vary on the type of information they convey to the clients, for instance, they may include just the identifiers of the updated items or the updated values themselves. They may also provide information for individual items or aggregate information for sets of items. Update propagation may be either synchronous or asynchronous. In *asynchronous* methods, update reports are broadcast as the updates are performed. In *synchronous* methods, the server broadcasts an update report periodically. A client must listen for the report first to decide whether its cache is valid or not. This adds some latency to query processing, however, each client needs only tune in periodically to read the report. The efficiency of update dissemination protocols for clients with different connectivity behavior, such as for workaholics (i.e., often connected clients) and sleepers (i.e., often disconnected clients), is evaluated in [4].

Finally, in the case of broadcast push-data delivery, clients may read items from different broadcast programs. The *currency* of the set of data items read by each client can be characterized based on the current values of the corresponding items at the server and on the temporal discrepancy among the values of the items in the set [13]. A more strict notion of correctness may be achieved through transaction-level correctness by requiring the client read-only transactions to be serializable with the server transactions. Methods for doing so include: (i) an *invalidation* method [12], where the server broadcasts an invalidation report that includes the data items that have been updated since the broadcast of the previous report, and transactions that have read obsolete items are aborted, (ii) *serialization graph testing* (SGT) [12], where the server broadcasts control information related to conflicting operations, and (iii) *multiversion broadcast* [11], where multiple versions of each item are broadcast, so that client transactions always read a consistent database snapshot.

Key Applications

Data broadcasting, caching and replication techniques are part of the core of any application that requires data sharing and synchronization among mobile devices and data servers. Such applications include vehicle dispatching, object tracking, points of sale (e.g., ambulance and taxi services, Fedex/UPS), and collaborative applications (e.g., homecare, video gaming). They are also part of embedded or light versions of database management systems that extend enterprise applications to mobile devices. These include among others Sybase Inc.'s SQL Anywhere, IBM's DB2 Everyplace, Microsoft SQL Server Compact, Oracle9i Lite and SQL Anywhere Technologies' Ultralite.

Cross-references

▶ Concurrency Control
▶ Hash-Based Indexing

- ▶ MANET Databases
- ▶ Mobile Database
- ▶ Replicated Database Concurrency Control
- ▶ Transaction Management

Recommended Reading

1. Acharya S., Alonso R., Franklin M.J., and Zdonik S.B. Broadcast disks: data management for asymmetric communications environments. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 199–210.
2. Acharya S., Franklin M.J., and Zdonik S.B. Balancing push and pull for data broadcast. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 183–194.
3. Aksoy D. and Franklin M.J. RxW: a scheduling approach for large scale on-demand broadcast. IEEE/ACM Trans. Netw., 7(6):846–860, 1999.
4. Barbará D. and Imielinski T. Sleepers and workaholics: caching strategies in mobile environments. VLDB J., 4(4):567–602, 1995.
5. Beaver J., Chrysanthis P.K., and Pruhs K. To broadcast push or not and what? In Proc. 7th Int. Conf. on Mobile Data Management, 2006, pp. 40–45.
6. Chrysanthis P.K. Transaction processing in a mobile computing environment. In Proc. IEEE Workshop on Advances in Parallel and Distributed Systems, 1993, pp. 77–82.
7. Dykeman H.D., Ammar M.H., and Wong J.W. Scheduling algorithms for videotex systems under broadcast delivery. In Proc. IEEE Int. Conf. on Communications, 1986, pp. 1847–1851.
8. Gray J., Helland P., Neil P.O., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
9. Petersen K., Spreitzer M., Terry D.B. Theimer M., and Demers A.J. Flexible update propagation for weakly consistent replication. In Proc. 16th ACM Symp. on Operating System Principles, 1997, pp. 288–301.
10. Pitoura E. and Bhargava B. Data consistency in intermittently connected distributed systems. IEEE Trans. Knowl. Data Eng., 11(6):896–915, 1999.
11. Pitoura E. and Chrysanthis P.K. Exploiting versions for handling updates in broadcast disks. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 114–125.
12. Pitoura E. and Chrysanthis P.K. Scalable processing of read-only transactions in broadcast push. In Proc. 19th Int. Conf. on Distributed Computing Systems, 1999, pp. 432–439.
13. Pitoura E., Chrysanthis P.K., and Ramamritham K. Characterizing the temporal and semantic coherency of broadcast-based data dissemination. In Proc. 9th Int. Conf. on Database Theory, 2003, pp. 410–424.
14. Pitoura E. and Samaras G. Data Management for Mobile Computing. Kluwer, Boston, USA, 1998.
15. Sharaf MA. and Chrysanthis P.K. On-demand data broadcasting for mobile decision making. MONET, 9(6):703–714, 2004.
16. Stathatos K., Roussopoulos N., and Baras J.S. Adaptive data broadcast in hybrid networks. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 326–335.
17. Su C.J, Tassiulas L., and Tsotras V.J. Broadcast scheduling for information distribution. Wireless Netw., 5(2):137–147, 1999.
18. T I., Viswanathan S., and Badrinath B.R. Data on air: organization and access. IEEE Trans. Knowl. Data Eng., 9(3):353–372, 1997.

Data Cache

- ▶ Processor Cache

Data Cleaning

Venkatesh Ganti
Microsoft Research, Redmond, WA, USA

Definition

Owing to differences in conventions between the external sources and the target data warehouse as well as due to a variety of errors, data from external sources may not conform to the standards and requirements at the data warehouse. Therefore, data has to be transformed and cleaned before it is loaded into a data warehouse so that downstream data analysis is reliable and accurate. *Data Cleaning* is the process of standardizing data representation and eliminating errors in data. The data cleaning process often involves one or more tasks each of which is important on its own. Each of these tasks addresses a part of the overall data cleaning problem. In addition to tasks which focus on transforming and modifying data, the problem of diagnosing quality of data in a database is important. This diagnosis process, often called data profiling, can usually identify data quality issues and whether or not the data cleaning process is meeting its goals.

Historical Background

Many business intelligence applications are enabled by data warehouses. If the quality of data in a data warehouse is poor, then conclusions drawn from business data analysis could also be incorrect. Therefore, much emphasis is placed on cleaning and maintaining high quality of data in data warehouses. Consequently, the area of data cleaning received considerable attention in the database community. An early survey of automatic data cleaning techniques can be found in [14]. Several companies also started developing domain-specific data cleaning solutions (especially for the customer address domain). Over time, several generic data cleaning techniques have been also been developed (e.g., [10,5,15,8,9,1]) and, domain neutral commercial data cleaning software also started making its appearance (e.g., [13,11]).

Foundations

Main Data Cleaning Tasks

In this section, the goals of several data cleaning tasks are introduced informally. The set of tasks mentioned below consists of those addressing commonly encountered problems in data cleaning and may not be a comprehensive list. However, note that most of the tasks mentioned below are important whether one wants to clean data at the time of loading a data warehouse or at the time of querying a database [6].

Column Segmentation Consider a scenario where a customer relation is being imported to add new records to a target customer relation. Suppose the address information in the target relation is split into its constituent attributes [street address, city, state, and zip code] while in the source relation they are all concatenated into one attribute. Before the records from the source relation could be inserted in the target relation, it is essential to segment each address value in the source relation to identity the attribute values at the target. The goal of a *column segmentation* task is to split an incoming string into segments, each of which may be inserted as attribute values at the target. A significant challenge to be addressed by this task is to efficiently match sub-strings of an input string with patterns such as regular expressions and with members of potentially large reference tables in order to identify values for target attributes. Note that, in general data integration may involve more complex schema transformations than achieved by the column segmentation task.

Record Matching Consider a scenario where a new batch of customer records is being imported into a sales database. In this scenario, it is important to verify whether or not the same customer is represented in both the existing as well as the incoming sets and only retain one record in the final result. Due to representational differences and errors, records in both batches could be different and may not match exactly on their key attributes (e.g., name and address or the CustomerId). The goal of a *record matching* task is to identify record pairs, one in each of two input relations, which correspond to the same real world entity. Challenges to be addressed in this task include (i) identification of criteria under which two records represent the same real world entity, and (ii) efficient computation strategies to determine such pairs over large input relations.

Deduplication Consider a scenario where one obtains a set of customer records or product records from an external (perhaps low quality) data source. This set may contain multiple records representing the same real world (customer or product) entity. It is important to "merge" records representing the same entity into one record in the final result. The goal of a *deduplication* task is to partition a relation into disjoint sets of records such that each group consists of records which represent the same real world entity. Deduplication may (internally) rely on a record matching task but the additional responsibility of further grouping records based on pairwise matches introduces new challenges. The output of record matching may not be transitively closed. For instance, a record matching task comparing record pairs in a relation may output pairs (r_1, r_2) and (r_2, r_3) as matches, but not (r_1, r_3). Then, the problem of deriving a partitioning that respects the pairwise information returned by record matching is solved by deduplication.

Data Standardization Consider a scenario where a relation contains several customer records with missing zip code or state values, or improperly formatted street address strings. In such cases, it is important to fill in missing values and adjust, where possible, the format of the address strings so as to return correct results for analysis queries. For instance, if a business analyst wants to understand the number of customers for a specific product by zip code, it is important for all customer records to have correct zip code values. The task of improving the quality of information within a database is often called *data standardization*. Similar tasks also occur in various other domains such as product catalog databases. The data standardization task may also improve the effectiveness of record matching and deduplication tasks.

Data Profiling The process of cleansing data is often an iterative and continuous process. It is important to "evaluate" quality of data in a database before one initiates data cleansing process, and subsequently assesses its success. The process of evaluating data quality is called *data profiling*, and typically involves gathering several aggregate data statistics which constitute the *data profile*, and ensuring that the values match up with expectations. For example, one may expect the customer name and address columns together to uniquely determine each customer record in a Customer relation. In such a case, the number of unique [name, address] values must be close to that of the total number of records in the Customer relation. Note that a large subset of elements of a data profile may each be obtained using one or more SQL queries. However, because all the elements of a data profile are computed together, there is an opportunity for a more efficient computation strategy. Further, the data profile of a database may also consist of elements which may not easily be computed using SQL queries.

Besides the set of data cleaning tasks mentioned above, other data cleaning tasks such as filling in missing values, identifying incorrect attribute values and then automatically correcting them based on known attribute value distributions are also important for applications such as cleaning census data.

Data Cleaning Platforms

The above requirements for a variety of data cleaning tasks have led to the development of utilities that support data transformation and cleaning. Such software falls into two broad categories:

Vertical Solutions: The first category consists of verticals such as Trillium [15] that provide data cleaning functionality for specific domains, e.g., addresses. Since they understand the domain where the vertical is being applied, they can fine tune their software for the given domain. However, by design, these are not generic and hence cannot be applied to other domains.

Horizontal Platforms: The other approach of building data cleaning software is to define and implement basic data cleaning operators. The broad goal here is to define a set of domain neutral operators, which can significantly reduce the load of developing common data cleaning tasks such as those outlined above. An example of such a basic operator is the set similarity join which may be used for identifying pairs of highly similar records across two relations (e.g., [16,4]). The advantage is that custom solutions for a variety of data cleaning tasks may now be developed for specialized domains by composing one or more of these basic operators along with other (standard or custom) operators. These basic operators do the heavy lifting and thus make the job of developing data cleaning programs easier. Examples of such platforms include AJAX [7,8] and Data Debugger [3].

The above mentioned data cleaning operators may then be included in database platforms so as to enable programmers to easily develop custom data cleaning solutions. For instance, *ETL (extract-transform-load)* tools such as Microsoft SQL Server Integration Services (SSIS) [13] and IBM Websphere Information Integration [11] that can be characterized as "horizontal" *platforms*. These platforms are applicable across a variety of domains, and provide a set of composable operators (e.g., relational operators) enabling users to build programs involving these operators. Further, these platforms also allow users to build their own custom operators which may then be used in these programs. Hence, such ETL platforms provide a great vehicle to include core data cleaning operators.

Key Applications

Data cleaning technology is critical for several information technology initiatives (such as data warehousing and business intelligence) which consolidate, organize, and analyze structured data. Accurate data cleaning processes are typically employed during data warehouse construction and maintenance to ensure that subsequent business intelligence applications yield accurate results.

A significant amount of recent work has been focusing on extracting structured information from documents to enable structured querying and analysis over document collections [2,12]. Invariably, the extracted data is unclean and many data cleaning tasks discussed above are applicable in this context as well.

Cross-references

- ▶ Column segmentation
- ▶ Constraint-driven database repair
- ▶ Data deduplication
- ▶ Data profiling
- ▶ Record matching
- ▶ Similarity functions for data cleaning

Recommended Reading

1. Borkar V. Deshmukh V. and Sarawagi S. Automatic segmentation of text into structured records. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001.
2. Cafarella M.J. Re C. Suciu D. Etzioni O. and Banko M. Structured querying of the web text. In Proc. 3rd Biennial Conf. on Innovative Data systems Research, 2007.

3. Chaudhuri S. Ganti V. and Kaushik. R. Data debugger: an operator-centric approach for data quality solutions. IEEE Data Eng. Bull., 2006.
4. Chaudhuri S. Ganti V. and Kaushik. R. A primitive operator for similarity joins in data cleaning. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
5. Cohen. W. Integration of heterogeneous databases without common domains using queries based on textual similarity. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998.
6. Fuxman A. Fazli E. and Miller. R.J. Conquer: efficient management of inconsistent databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
7. Galhardas H. Florescu D. Shasha D. and Simon. E. An extensible framework for data cleaning. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
8. Galhardas H. Florescu D. Shasha D. Simon E. and Saita. C. Declarative data cleaning: language, model, and algorithms. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
9. Gravano L. Ipeirotis P.G. Jagadish H.V. Koudas N. Muthukrishnan S. and Srivastava. D. Approximate string joins in a database (almost) for free. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
10. Hernandez. M. and Stolfo. S. The merge/purge problem for large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995.
11. IBM Websphere information integration. http://ibm.ascential.com.
12. Ipeirotis P.G. Agichtein E. Jain P. and Gravano. L. To search or to crawl? towards a query optimizer for text-centric tasks. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006.
13. Microsoft SQL Server 2005 integration services.
14. Rahm E. and Do. H.H. Data cleaning: problems and current approaches. IEEE Data Engineering Bulletin, 2000.
15. Raman V. and Hellerstein. J. An interactive framework for data cleaning. Technical report, University of California, Berkeley, 2000.
16. Sarawagi S. and Kirpal. A. Efficient set joins on similarity predicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
17. Trillium Software. www.trilliumsoft.com/trilliumsoft.nsf.

Data Collection

► Data Acquisition and Dissemination in Sensor Networks

Data Compression in Sensor Networks

Amol Deshpande
University of Maryland, College Park, MD, USA

Synonyms

Distributed source coding; Correlated data collection; Data suppression

Definition

Data compression issues arise in a sensor network when designing protocols for efficiently collecting all data observed by the sensor nodes at an Internet-connected base station. More formally, let X_i denote an attribute being observed by a node in the sensor network – X_i may be an environmental property being sensed by the node (e.g., *temperature*), or it may be the result of an operation on the sensed values (e.g., in an anomaly-detection application, the sensor node may continuously evaluate a filter such as "*temperature* > 100" on the observed values). The goal is to design an energy-efficient protocol to periodically collect the observed values of all such attributes (denoted $X_1,...,X_n$) at the base station, at a frequency specified by the user. In many cases, a bounded-error approximation might be acceptable, ie., the reported values may only be required to be within $\pm \in$ of the observed values, for a given $\in$. The typical optimization metric is the total energy expended during the data collection process, commonly approximated by the total communication cost. However, metrics such as minimizing the maximum energy consumption across all nodes or maximizing the lifetime of the sensor network may also be appropriate in some settings.

Key Points

The key issue in designing data collection protocols is modeling and exploiting the strong spatio-temporal correlations present in most sensor networks. Let X_i^t be a random variable that denotes the value of X_i at time t (assuming time is discrete), and let $H(X_i^t)$ denote the *information entropy* of $X_i{}^t$. In most sensor network deployments, especially in environmental monitoring applications, the data generated by the sensor nodes is typically highly correlated both in time and in space — in other words, $H(X_i^{t+1}|X_i^t) \ll H(X_i^{t+1})$, and $H(X_1{}^t,...,X_n{}^t) \ll H(X_1{}^t) + ... H(X_n{}^t)$. These correlations can usually be captured quite easily by constructing predictive models using either prior domain knowledge or historical data traces. However, because of the distributed nature of data generation in sensor networks, and the resource-constrained nature of sensor nodes, traditional data compression techniques cannot be easily adapted to exploit such correlations.

The distributed nature of data generation has been well-studied in the literature under the name of *Distributed Source Coding*, whose foundations were laid almost 35 years ago by Slepian and Wolf [6]. Their seminal work proves that it is theoretically possible to encode the correlated information generated by distributed data sources at the rate of their joint entropy even if *the data sources do not communicate with each other*. However this result is non-constructive, and constructive techniques are known only for a few specific distributions [4]. More importantly, these techniques require precise and perfect knowledge of the correlations. This may not be acceptable in practical sensor networks, where deviations from the modeled correlations must be captured accurately. Pattem et al. [3] and Chu et al. [2], among others, propose practical data collection protocols that exploit the spatio-temporal correlations while guaranteeing correctness. However, these protocols may exploit only some of the correlations, and further require the sensor nodes to communicate with each other (thus increasing the overall cost).

In many cases, it may not be feasible to construct a predictive model over the sensor network attributes, as required by the above approach, because of mobility, high failure rates or inherently unpredictable nature of the monitored phenomena. Suppression-based protocols, that monitor local constraints and report to the base station only when the constraints are violated, may be used instead in such scenarios [5].

Sensor networks, especially wireless sensor networks, exhibit other significant peculiarities that make the data collection problem challenging. First, sensor nodes are typically computationally constrained and have limited memories. As a result, it may not be feasible to run sophisticated data compression algorithms on them.

Second, the communication in wireless sensor networks is typically done in a broadcast manner – when a node transmits a message, all nodes within the radio range can receive the message. This enables many optimizations that would not be possible in a one-to-one communication model.

Third, sensor networks typically exhibit an extreme asymmetry in the computation and communication capabilities of the sensor nodes compared to the base station. This motivates the design of pull-based data collection techniques where the base station takes an active role in the process. Adler [1] proposes such a technique for a one-hop sensor network. The proposed algorithm achieves the information-theoretical lower bound on the number of bits *sent* by the sensor nodes, while at the same time offloading most of the compute-intensive work to the base station. However, the number of bits *received* by the sensor nodes may be very high.

Finally, sensor networks typically exhibit high message loss and sensor failure rates. Designing robust and fault-tolerant protocols with provable guarantees is a challenge in such an environment.

Cross-references

► Continuous Queries in Sensor Networks
► Data Aggregation in Sensor Networks
► Data Fusion in Sensor Networks
► In-Network Query Processing
► Model-based Querying in Sensor Networks

Recommended Reading

1. Adler M. Collecting correlated information from a sensor network. In Proc. 16th Annual ACM -SIAM Symp. on Discrete Algorithms, 2005.
2. Chu D., Deshpande A., Hellerstein J., and Hong W. Approximate data collection in sensor networks using probabilistic models. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
3. Pattem S., Krishnamachari B., and Govindan R. The impact of spatial correlation on routing with compression in wireless sensor networks. In Proc. 3rd Int. Symp. Inf. Proc. in Sensor Networks, 2004.
4. Pradhan S. and Ramchandran K. Distributed source coding using syndromes (DISCUS): Design and construction. IEEE Trans. Inform. Theory, 49(3), 2003.
5. Silberstein A., Puggioni G., Gelfand A., Munagala K., and Yang J. Making sense of suppressions and failures in sensor data: a Bayesian approach. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
6. Slepian D. and Wolf J. Noiseless coding of correlated information sources. IEEE Trans. Inform. Theory, 19(4), 1973.

Data Confidentiality

► Security Services

Data Conflicts

HONG-HAI DO
SAP AG, Dresden, Germany

Synonyms

Data problems; Data quality problems; Data anomalies; Data inconsistencies; Data errors

Definition

Data conflicts are deviations between data intended to capture the same state of a real-world entity. Data with conflicts are often called "dirty" data and can mislead analysis performed on it. In case of data conflicts, data cleaning is needed in order to improve the data quality and to avoid wrong analysis results. With an understanding of different kinds of data conflicts and their characteristics, corresponding techniques for data cleaning can be developed.

Historical Background

Statisticians were probably the first who had to face data conflicts on a large scale. Early applications, which needed intensive resolution of data conflicts, were statistical surveys in the areas of governmental administration, public health, and scientific experiments. In 1946, Halbert L. Dunn already observed the problem of duplicates in data records of a person's life captured at different places [3]. He introduced the term Record Linkage to denote the process to resolve the problem, i.e., to obtain and link all unique data records to a consistent view on the person. In 1969, Fellegi and Sunter provided a formal mathematical model for the problem and thereby laid down the theoretical foundation for numerous record linkage applications developed later on [5].

Soon it became clear that record linkage is only the tip of the iceberg of the various problems, such as wrong, missing, inaccurate, and contradicting data, which makes it difficult for humans and applications to obtain a consistent view on real-world entities. In the late 1980s, computer scientists began to systematically investigate all problems related to data quality, increasingly from a practical perspective in the context of business applications. This was essentially pushed by the need to integrate data from heterogeneous sources for business decision making and by the emergence of enterprise data warehouses at the beginning of the 1990s. To date, various research approaches and commercial tools have been developed to deal with the different kinds of data conflicts and to improve data quality [1,2,4,7].

Foundations

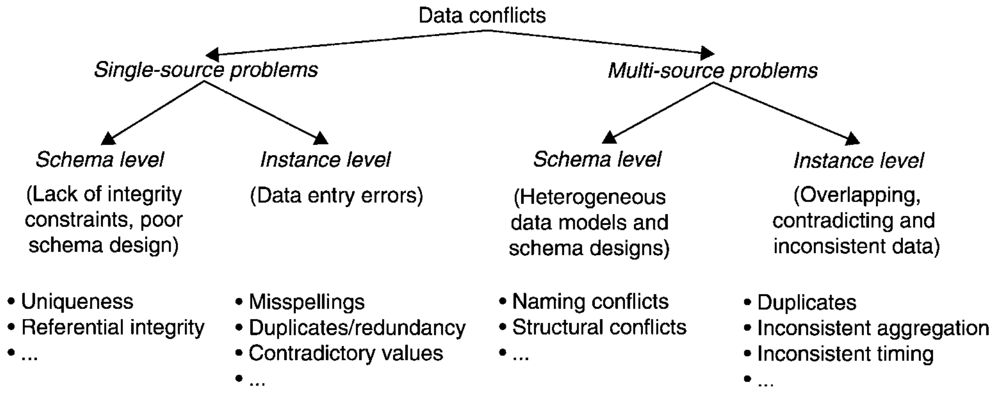
Classification of Data Conflicts

As shown in Fig. 1, data conflicts can be classified according to the following criteria:

- *Single-source* versus *multi-source*: Data conflicts can occur among data within a single source or between different sources.
- *Schema-level* versus *instance-level*: Schema-level conflicts are caused by the design of the data schemas. Instance-level conflicts, on the other hand, refer to problems and inconsistencies in the actual data contents, which are not visible at the schema level.

Figure 1 also shows typical data conflicts for the various cases. While not shown, the single-source conflicts occur (with increased likelihood) in the multi-source case, too, besides specific multi-source conflicts.

Single-Source Data Conflicts The data quality of a source largely depends on the degree to which it is governed by schema and integrity constraints controlling permissible data values. For sources without a schema,



Data Conflicts. Figure 1. Examples of multi-source problems at schema and instance level.

such as files, there are few restrictions on what data can be entered and stored, giving rise to a high probability of errors and inconsistencies. Database systems, on the other hand, enforce restrictions of a specific data model (e.g., the relational approach requires simple attribute values, referential integrity, etc.) as well as application-specific integrity constraints. Schema-related data quality problems thus occur because of the lack of appropriate model-specific or application-specific integrity constraints, e.g., due to data model limitations or poor schema design, or because only a few integrity constraints were defined to limit the overhead for integrity control. Instance-specific problems are related to errors and inconsistencies that cannot be prevented at the schema level (e.g., misspellings).

Both schema- and instance-level conflicts can be further differentiated according to the different problem scopes *attribute*, *record*, *record type*, and *source*. In particular, a data conflict can occur within an individual attribute value (*attribute*), between attributes of a record (*record*), between records of a record type (*record type*), and between records of different record types (*source*). Examples of data conflicts in each problem scope are shown and explained in Tables 1 and 2 for the schema and instance level, respectively. Note that uniqueness constraints specified at the schema level do not prevent duplicated instances, e.g., if information on the same real world entity is entered twice with different attribute values (see examples in Table 2).

Multi-Source Data Conflicts The problems present in single sources are aggravated when multiple sources need to be integrated. Each source may contain dirty data and the data in the sources may be represented differently, may overlap, or contradict. This is because the sources are typically developed, deployed and maintained independently to serve specific needs. This results in a large degree of heterogeneity with respect to database management systems, data models, schema designs, and the actual data.

At the schema level, data model and schema design differences are to be addressed by the steps of schema translation and schema integration, respectively. The main problems with respect to schema design are naming and structural conflicts. Naming conflicts arise when the same name is used for different objects (homonyms) or different names are used for the same object (synonyms). Structural conflicts occur in many variations and refer to different representations of the same object in different sources, e.g., attribute versus table representation, different component structure, different data types, different integrity constraints, etc.

In addition to schema-level conflicts, many conflicts appear only at the instance level. All problems from the single-source case can occur with different representations in different sources (e.g., duplicated records, contradicting records). Furthermore, even when there are the same attribute names and data types, there may be different value representations (e.g., M/F vs. Male/Female for marital status) or different interpretation of the values (e.g., measurement units Dollar vs. Euro) across sources. Moreover, information in the sources may be provided at different aggregation levels (e.g., sales per product vs. sales per product group) or refer to different points in time (e.g., current sales as of yesterday for Source 1 vs. as of last week for Source 2).

A main problem for cleaning data from multiple sources is to identify overlapping data, in particular matching records referring to the same real-world entity (e.g., a particular customer). This problem is

Data Conflicts. Table 1. Examples for single-source problems at schema level (violated integrity constraints)

Scope	Type of conflict	Dirty data	Reasons/Remarks
Attribute	Illegal values	birthdate = 13/30/1970	Values outside of domain range
Record	Violated attribute dependencies	city = “Redmond,” zip = 77777	City and zip code should correspond
Record type	Uniqueness violation	emp_1 = (name = “John Smith,” SSN = “123456”), emp_2 = (name = “Peter Miller,” SSN = “123456”)	Uniqueness for SSN (social security number) violated
Source	Referential integrity violation	emp = (name = “John Smith,” deptno = 127)	Referenced department (127) not defined

Data Conflicts. Table 2. Examples for single-source problems at instance level

Scope	Type of conflict	Dirty data	Reasons/Remarks
Attribute	Missing values	phone = 9999–999999	Unavailable values during data entry (dummy values or null)
	Misspellings	city = “London”	Usually typos, phonetic errors
	Cryptic values, Abbreviations	experience = “B”; occupation = “DB Prog.”	Use of code lists
	Embedded values	name = “J. Smith 02/12/70 New York”	Multiple values entered in one attribute (e.g., in a free-form field)
	Misfielded values	city = “Germany”	City field contains value of country field
Record	Violated attribute dependencies	city = “Redmond,” zip = 77777	City and zip code should correspond
Record type	Word transpositions	name_1 = “J. Smith,” name_2 = “Miller P.”	Usually in a free-form field
	Duplicated records	emp_1 = (name = “John Smith,” . . .); emp_2 = (name = “J. Smith,” . . .)	Same employee represented twice due to some data entry errors
	Contradicting records	emp_1 = (name = “John Smith,” bdate = 02/12/70); emp_2 = (name = “John Smith,” bdate = 12/12/70)	The same real world entity is described by different values
Source	Wrong references	emp = (name = “John Smith,” deptno = 17)	Referenced department (17) is defined but wrong

also referred to as the record linkage problem, the object identity problem, or the deduplication problem. Frequently, the information is only partially redundant and the sources may complement each other by providing additional information about an entity. Thus duplicate information should be purged and complementing information should be consolidated and merged in order to achieve a consistent view of real-world entities.

The two relational sources, Source 1 and Source 2, in the example of Fig. 2 exhibit several kinds of conflicts. At the schema level, there are name conflicts (synonyms *Customer* vs. *Client*, *CID* vs. *Cno*, *Sex* vs. *Gender*) and structural conflicts (different structures for names *Name* vs. {*LastName*, *FirstName*}, and for addresses {*Street*, *City*, *Zip*} vs. *Address*). At the instance level, one can see that there are different gender representations (“0”/“1” vs. “*F*”/“*M*”) and presumably a duplicate record (*Kristen Smith*). The latter observation also reveals that while *CID* and *Cno* are both source-specific identifiers, their contents are not comparable between the sources; different numbers (“11,” “49”) refer to the same person while different persons have the same number (“24”).

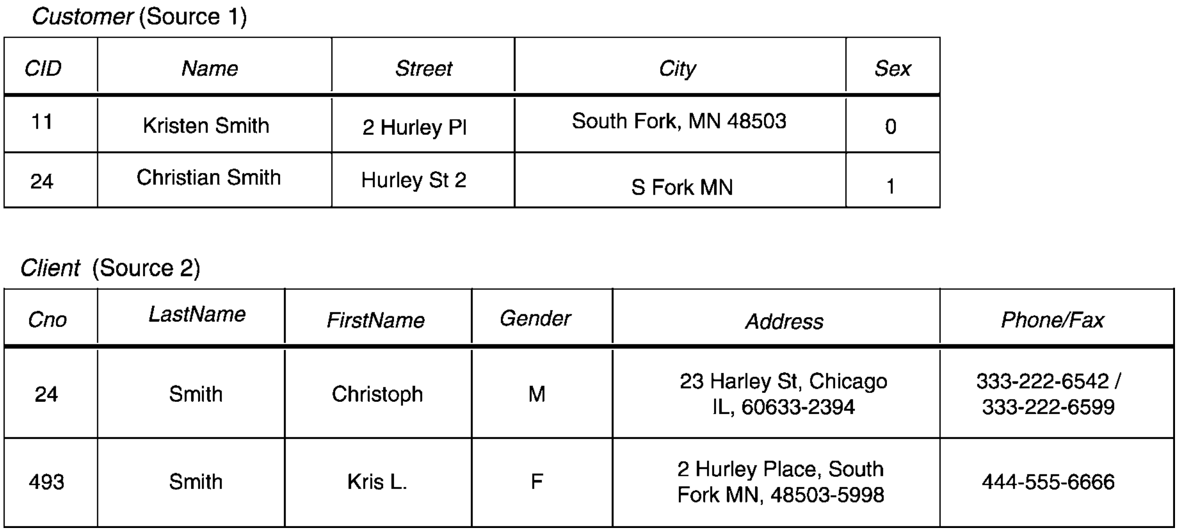
Dealing with Data Conflicts

Data conflicts can be dealt with in a preventive and/or corrective way. As resolving existing data conflicts is generally an expensive task, preventing dirty data to be entered is promising to ensure a high data quality. This requires appropriate design of the database schema with corresponding integrity constraints and strict enforcement of the constraints in the databases and data entry applications. For most applications, however, a corrective strategy, i.e., data cleaning (a.k.a. cleansing or scrubbing), is needed in order to remove conflicts from given data and make it suitable for analysis. Typically, this process involves thorough analysis of the data to detect conflicts and transformation of the data to resolve the identified conflicts.

Key Applications

Data Warehousing

Data warehousing aims at a consolidated and consistent view of enterprise data for business decision making. Transactional and non-transactional data from a variety of sources is aggregated and structured typically in a multidimensional schema to effectively support dynamic



Customer (Source 1)

CID	*Name*	*Street*	*City*	*Sex*
11	Kristen Smith	2 Hurley Pl	South Fork, MN 48503	0
24	Christian Smith	Hurley St 2	S Fork MN	1

Client (Source 2)

Cno	*LastName*	*FirstName*	*Gender*	*Address*	*Phone/Fax*
24	Smith	Christoph	M	23 Harley St, Chicago IL, 60633-2394	333-222-6542 / 333-222-6599
493	Smith	Kris L.	F	2 Hurley Place, South Fork MN, 48503-5998	444-555-6666

Data Conflicts. Figure 2. Classification of data conflicts in data sources.

querying and reporting, such as Online Analytical Processing (OLAP). As multiple data sources are considered, the probability that some of the sources contain conflicting data is high. Furthermore, the correctness of the integrated data is vital to avoid wrong conclusions. Due to the wide range of possible data inconsistencies and the sheer data volume, data cleaning is one of the biggest problems for data warehousing. Data conflicts need to be detected and resolved during the so-called ETL process (Extraction, Transformation, and Load), when source data is integrated from corresponding sources and stored into the data warehouse.

Data Mining

Data mining, or knowledge discovery, is the analysis of large data sets to extract new and useful information. Developed algorithms utilize a number of techniques, such as data visualization (charts, graphs), statistics (summarization, regression, clustering), and artificial intelligence techniques (classification, machine learning, and neural networks). As relevant data typically needs to be integrated from different sources, data warehousing represents a promising way to build a suitable data basis for data mining. However, due to performance reasons on large data sets, specialized data mining algorithms often operate directly on structured files. In either case, resolving data conflicts to obtain correct data is crucial for the success of data mining. On the other hand, the powerful data mining algorithms can also be utilized to analyze dirty data and discover data conflicts.

Cross-references

- ▶ Data Cleaning
- ▶ Data Quality
- ▶ Duplicate Detection

Recommended Reading

1. Barateiro J. and Galhardas H. A survey of data quality tools. Datenbank-Spektrum, 14:15–21, 2005.
2. Batini C. and Scannapieco M. Data Quality – Concepts, Methodologies and Techniques. Springer, Berlin, 2006.
3. Dunn H.L. Record linkage. Am. J. Public Health, 36(12):1412–1416, 1946.
4. Elmagarmid A.K., Ipeirotis P.G., and Verykios V.S. Duplicate record detection – a survey. IEEE Trans. Knowl. Data Eng., 19(1):1–16, 2007.
5. Fellegi I.P. and Sunter A.B. A theory for record linkage. J. Am. Stat. Assoc., 64(328):1183–1210, 1969.
6. Kim W., Choi B.-J., Kim S.-K., and Lee D. A taxonomy of dirty data. Data Mining Knowl. Discov., 7(1):81–99, 2003.
7. Rahm E. and Do H.-H. Data cleaning – problems and current approaches. IEEE Techn. Bull. Data Eng., 23(4):3–13, 2000.

Data Copy

▶ Data broadcasting, caching and replication

Data Corruption

▶ Storage Security

Data Deduplication

▶ Record Matching

Data Dependency

▶ Database Dependencies

Data Dictionary

James Caverlee
Texas A&M University, College Station, TX, USA

Synonyms

System catalog; Metadata repository

Definition

A data dictionary catalogs the definitions of data elements, data types, data flows and other conventions that are used in an information system. Data dictionaries have been widely adopted by both (i) the database community, where a dictionary typically describes database entities, schemas, permissions, etc.; and (ii) the software development community, where a dictionary typically describes flows of information through the system. In essence, a data dictionary is a virtual database of metadata about an information system itself. A dictionary may also be referred to as a "system catalog."

Key Points

Understanding and managing an information system – both from a design and from an implementation point-of-view – requires some documentation of the schema, capabilities, constraints, and other descriptive features of the system. This documentation is typically embodied by a data dictionary – that is, a repository of information for an information system that describes the entities represented as data, including their attributes and the relationships between them [3].

The importance of a systematic way to store and manage the metadata associated with an information system has been well known since the earliest days of database and large-scale systems development. By the time the relational model was garnering attention in the 1970s, metadata management via a system catalog was a standard feature in database management systems (DBMSs) like System R [1] and INGRES [5]. Around the same time, the structured analysis approach for large-scale systems development also advocated for the use of a data dictionary [2].

The phrase *data dictionary* has two closely related meanings: (i) as documentation primarily for consumption by human users, administrators, and designers; and (ii) as a mini-database managed by a DBMS and tightly coupled with the software components of the DBMS.

In the first meaning, a data dictionary is a document (or collection of documents) that provides a conceptual view of the structure of an information system for those developing, using, and maintaining the system. In this first meaning, a data dictionary serves to document the system design process, to identify the important characteristics of the system (e.g., schemas, constraints, data flows), and to provide the designers, users, and administrators of the system a central metadata repository [6]. A data dictionary can provide the names of tables and fields, types for data attributes, encoding information, and further details of an overall structure and usage. The owner of a database or database administrator (DBA) might provide it as a book or a document with additional descriptions and diagrams, or as generated documentation derived from a database. Database users and application developers then benefit from the data dictionary as an accepted reference, though this hardcopy version is not always provided nor required.

In the second meaning, a data dictionary is a mini-database tightly coupled and managed by an information system (typically a DBMS) for supporting query optimization, transaction processing, and other typical features of a DBMS. When used in this sense, a data dictionary is often referred to as a *catalog* or as a *system catalog*. As a software component of a database or a DBMS, a data dictionary makes up all the metadata and additional functions needed for a database manipulation language (DML) to select, insert, and generally operate on data. A database user will do this in conjunction with a high-level programming language or from a textual or graphical user interface (GUI). The data dictionary for a database or DBMS typically has these elements:

- Descriptions of tables and fields
- Permissions information, such as usernames and privileges
- How data is indexed

- Referential integrity constraints
- Definitions for database schemas
- Storage allocation parameters
- Usage statistics
- Stored procedures and database triggers

For an example, a developer unfamiliar with what tables are available within a database could query the virtual INFORMATION_SCHEMA database, which serves as the data dictionary for MySQL databases [4].

Besides this low-level version of a data dictionary, some software frameworks add another layer of abstraction to create a high-level data dictionary as well. This layer can reduce development time by providing features not supported at the lower level, such as alternative database scheme models. One example is Object-Relational Mapping (ORM), which seeks to map the data types created in the Object-Oriented Programming (OOP) paradigm to a relational database.

Cross-references

► Metadata
► Metadata Repository

Recommended Reading

1. Astrahan M. et al. (1979) System R: a relational data base management system. IEEE Comput. 12(5):42–48, 1979.
2. Demarco T. Structured Analysis and System Specification. Yourdon, 1978.
3. Elmasri R. and Navathe S. Fundamentals of database systems. Addison-Wesley, Reading, MA, 2000.
4. MySQL MySQL 5.0 Reference Manual, 2008.
5. Stonebraker M., Wong E., Kreps P., and Held G. The design and implementation of INGRES. ACM Trans. Database Syst., 1(3):189–222, 1976.
6. Yourdon E. Modern Structured Analysis. Yourdon, 1989.

Data Dissemination

► Data broadcasting, caching and replication

Data Encryption

Ninghui Li
Purdue University, West Lafayette, IN, USA

Synonyms

Encryption; Cipher

Definition

Data encryption is the process of transforming data (referred to as plaintext) to make it unreadable except to those possessing some secret knowledge, usually referred to as a key. The result of the process is encrypted data (referred to as ciphertext). Data encryption aims at preserving confidentiality of messages. The reverse process of deriving the plaintext from the ciphertext (using the key) is known as decryption. A cipher is a pair of algorithms which perform encryption and decryption. The study of data encryption is part of cryptography. The study of how to break ciphers, i.e., to obtaining the meaning of encrypted information without access to the key, is called cryptanalysis.

Historical Background

Encryption has been used to protect communications since ancient times by militaries and governments to facilitate secret communication. The earliest known usages of cryptography include a tool called Scytale, which was used by the Greeks as early as the seventh century BC, and the Caesar cipher, which was used by Julius Caesar in the first century B.C.

The main classical cipher types are transposition ciphers, which rearrange the order of letters in a message, and substitution ciphers, which systematically replace letters or groups of letters with other letters or groups of letters. Ciphertexts produced by classical ciphers always reveal statistical information about the plaintext. Frequent analysis can be used to break classical ciphers.

Early in the twentieth century, several mechanical encryption/decryption devices were invented, including rotor machines – most famously the Enigma machine used by Germany in World War II. Mechanical encryption devices, and successful attacks on them, played a vital role in World War II.

Cryptography entered modern age in the 1970s, marked by two important events: the introduction of the U.S. Data Encryption Standard and the invention of public key cryptography. The development of digital computers made possible much more complex ciphers. At the same time, computers have also assisted cryptanalysis. Nonetheless, good modern ciphers have stayed ahead of cryptanalysis; it is usually the case that use of a quality cipher is very efficient (i.e., fast and requiring few resources), while breaking it requires an effort many orders of magnitude larger, making cryptanalysis so inefficient and impractical as to be effectively impossible.

Today, strong encryption is no longer limited to secretive government agencies. Encryption is now widely used by the financial industry to protect money transfers, by merchants to protect credit-card information in electronic commerce, by corporations to secure sensitive communications of proprietary information, and by citizens to protect their private data and communications.

Foundations

Data encryption can be either secret-key based or public-key based. In secret-key encryption (also known as symmetric encryption), a single key is used for both encryption and decryption. In public-key encryption (also known as asymmetric encryption), the encryption key (also called the public key) and the corresponding decryption key (also called the private key) are different. Modern symmetric encryption algorithms are often classified into stream ciphers and block ciphers.

Stream Ciphers

In a stream cipher, the key is used to generate a pseudo-random key stream, and the ciphertext is computed by using a simple operation (e.g., bit-by-bit XOR or byte-by-byte modular addition) to combine the plaintext bits and the key stream bits. Mathematically, a stream cipher is a function $f:\{0,1\}^{\ell} \rightarrow \{0,1\}^{m}$, where ℓ is the key size, and m determines the length of the longest message that can be encrypted under one key; m is typically much larger than ℓ. To encrypt a message x using a key k, one computes $c = f(k) \oplus x$, where $\oplus$ denote bit-by-bit XOR. To decrypt a ciphertext c using key k, one computes $f(k) \oplus c$.

Many stream ciphers implemented in hardware are constructed using linear feedback shift registers (LFSRs). The use of LFSRs on their own, however, is insufficient to provide good security. Additional variations and enhancements are needed to increase the security of LFSRs.

The most widely-used software stream cipher is RC4. It was designed by Ron Rivest of RSA Security in 1987. It is used in popular protocols such as Secure Sockets Layer (SSL) (to protect Internet traffic) and WEP (to secure wireless networks).

Stream ciphers typically execute at a higher speed than block ciphers and have lower hardware complexity. However, stream ciphers can be susceptible to serious security problems if used incorrectly; in particular, the same starting state (i.e., the same generated key stream) must never be used twice.

Block Ciphers

A block cipher operates on large blocks of bits, often 64 or 128 bits. Mathematically, a block cipher is a pair of functions $\mathcal{E} : \{0,1\}^{\ell} \times \{0,1\}^{n} \rightarrow \{0,1\}^{n}$ and $\mathcal{D} : \{0,1\}^{\ell} \times \{0,1\}^{n} \rightarrow \{0,1\}^{n}$, where ℓ is the key size and n is the block size. To encrypt a message x using key k, one calculates $\mathcal{E}(k,x)$, which is often written as $\mathcal{E}_k[x]$. To decrypt a ciphertext c using key k, one calculates $\mathcal{D}(k,c)$, often written as $\mathcal{D}_k[c]$. The pair $\mathcal{E}$ and $\mathcal{D}$ must satisfy

$$\forall k \in \{0,1\}^{\ell} \;\; \forall x \in \{0,1\}^{n} \;\; \mathcal{D}_k[\mathcal{E}_k[x]] = x.$$

The two most widely used block ciphers are the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES).

DES is a block cipher selected as Federal Information Processing Standard for the United States in 1976. It has subsequently enjoyed widespread use internationally. The block size of DES is 64 bits, and the key size 56 bits. The main weakness of DES is its short key size, which makes it vulnerable to bruteforce attacks that try all possible keys.

One way to overcome the short key size of DES is to use Triple DES (3DES), which encrypts a 64-bit block by running DES three times using three DES keys. More specifically, let $(\mathcal{E}, \mathcal{D})$ be the pair of encryption and decryption functions for DES, then the encryption function for 3DES is

$$3\mathrm{DES}_{k_1,k_2,k_3}(x) = \mathcal{E}_{k_1}[\mathcal{D}_{k_2}[\mathcal{E}_{k_3}(x)]].$$

AES was announced as an U.S. Federal Information Processing Standard on November 26, 2001 after a 5-year selection process that is opened to the public. It became effective as a standard May 26, 2002. The algorithm is invented by Joan Daemen and Vincent Rijmen and is formerly known as Rijndael. AES uses a block size of 128 bits, and supports key sizes of 128 bits, 192 bits, and 256 bits.

Because messages to be encrypted may be of arbitrary length, and because encrypting the same plaintext under the same key always produces the same output, several modes of operation have been invented which allow block ciphers to provide confidentiality for messages of arbitrary length. For example, in the electronic codebook (ECB) mode, the message is divided into blocks and each block is encrypted separately. The disadvantage of this method is that identical plaintext blocks are encrypted into identical ciphertext blocks.

It is not recommended for use in cryptographic protocols. In the cipher-block chaining (CBC) mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block is dependent on all plaintext blocks processed up to that point. Also, to make each message unique, an initialization vector must be used in the first block and should be chosen randomly. More specifically, to encrypt a message x under key k, let x_1, $x_2,\ldots,x_m$ denote the message blocks, then the ciphertext is $c_0||c_1||\ldots||c_m$ where $||$ denote concatenation, c_0 = IV, the randomly chosen initial value, and $c_i = \mathcal{E}_k[x_i] \oplus c_{i-1}$ for $1 \leq i \leq m$. Other well-known modes include Cipher feedback (CFB), Output feedback (OFB), and Counter (CTR).

Public Key Encryption Algorithms

When using symmetric encryption for secure communication, the sender and the receiver must agree upon a key and the key must kept secret so that no other party knows the key. This means that the key must be distributed using a secure, but non-cryptographic, method; for example, a face-to-face meeting or a trusted courier. This is expensive and even impossible in some situations. Public key encryption was invented to solve the key distribution problem. When public key encryption is used, users can distribute public keys over insecure channels.

One of the most widely used public-key encryption algorithm is RSA. RSA was publicly described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman at MIT; the letters RSA are the initials of their surnames. To generate a pair of RSA public/private keys, one does the following: choose two distinct large prime numbers p, q, calculate $N = pq$ and $\phi(N) = (p-1)(q-1)$, choose an integer e such that $1 < e < \phi(N)$, and e and $\phi(N)$ share no factors other than 1. The public key is (N, e), and the private key is (N, d), where $ed \equiv 1(\text{mod } \phi(N))$. A message to be encrypted is encoded using a positive integer x where $x < N$. To encrypt x, compute $c = x^e \bmod N$. To decrypt a ciphertext c, compute $c^e \bmod N$. Practical RSA implementations typically embed some form of structured, randomized padding into the value x before encrypting it. Without such padding, the ciphertext leaks some information about the plaintext and is generally considered insecure for data encryption. It is generally presumed that RSA is secure if N is sufficiently large. The lengths of N are typically 1,024–4,096 bits long.

A central problem for public-key cryptography is proving that a public key is authentic and has not been tampered with or replaced by a malicious third party. The usual approach to this problem is to use a public-key infrastructure (PKI), in which one or more third parties, known as certificate authorities, certify ownership of key pairs.

Asymmetric encryption algorithms are much more computationally intensive than symmetric algorithms. In practice, public key cryptography is used in combination with secret-key methods for efficiency reasons. For encryption, the sender encrypts the message with a secret-key algorithm using a randomly generated key, and that random key is then encrypted with the recipient's public key.

Attack Models

Attack models or attack types for ciphers specify how much information a cryptanalyst has access to when cracking an encrypted message. Some common attack models are:

- *Ciphertext-only attack*: the attacker has access only to a set of ciphertexts.
- *Known-plaintext attack*: the attacker has samples of both the plaintext and its encrypted version (ciphertext).
- *Chosen-plaintext attack*: the attacker has the capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts.
- *Chosen-ciphertext attack*: the attacker has the capability to choose a number of ciphertexts and obtain the plaintexts.

Key Applications

Data encryption is provided by most database management systems. It is also used in many settings in which database is used, e.g., electronic commerce systems.

Cross-references

▶ Asymmetric Encryption
▶ Symmetric Encryption

Recommended Reading

1. Diffie W. and Hellman M.E. New directions in cryptography. IEEE Trans. Inform. Theory, 22:644–654, 1976.
2. Federal information processing standards publication 46-3: data encryption standard (DES), 1999.
3. Federal information processing standards publication 197: advanced encryption standard, Nov. 2001.

4. Kahn D. The codebreakers: the comprehensive history of secret communication from ancient times to the internet. 1996.
5. Menezes A.J., Oorschot P.C.V., and Vanstone S.A. Handbook of applied cryptography (revised reprint with updates). CRC, West Palm Beach, FL, USA, 1997.
6. Rivest R.L., Shamir A., and Adleman L.M. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21:120–126, 1978.
7. Singh S. The code book: the science of secrecy from ancient Egypt to quantum cryptography. Anchor, Garden City, NY, USA, 2000.

Data Errors

▶ Data Conflicts

Data Estimation in Sensor Networks

Le Gruenwald
University of Oklahoma, Norman, OK, USA

Synonyms

Data imputation

Definition

In wireless sensor networks, sensors typically transmit their data to servers at predefined time intervals. In this environment, data packets are very susceptible to losses, delays or corruption due to various reasons, such as power outage at the sensor's node, a higher bit error rate of the wireless radio transmissions compared to the wire communication alternative, an inefficient routing algorithm implemented in the network, or random occurrences of local interferences (e.g., mobile radio devices, microwaves or broken line-of-sight path). To process queries that need to access the missing data, if repeated requests are sent to sensors asking them to resend the missing information, this would incur power-costly communications as those sensors must be constantly in the listening mode. In addition, it is not guaranteed that those sensors would resend their missing data or would resend them in a timely manner. Alternatively, one might choose to estimate the missing data based on the underlying structure or patterns of the past reported data. Due to the low power-cost of computation, this approach represents an efficient way of answering queries that need to access the missing information. This entry discusses a number of existing data estimation approaches that one can use to estimate the value of a missing sensor reading.

Key Points

To estimate the value of a missing sensor reading, the quality of service in terms of high estimate accuracy and low estimation time needs to be observed. Data estimation algorithms can be divided into three major groups: (i) traditional statistical approaches; (ii) statistical-based sensor/stream data approaches, and (iii) association rule data mining based approaches. Many traditional statistical data approaches are not appropriate for wireless sensor networks as they require either the entire data set to be available or data to be missed at random, or do not consider relationships among sensors. Some statistical based sensor/stream data estimation algorithms include SPIRIT [3] and TinyDB [2]. SPIRIT is a pattern discovery system that uncovers key trends within data of multiple time series. These trends or correlations are summarized by a number of hidden variables. To estimate current missing values, SPIRIT applies an auto-regression forecasting model on the hidden variables. TinyDB is a sensor query processing system where missing values are estimated by taking the average of all the values reported by the other sensors in the current round. Two association rule based data estimation algorithms are WARM [1] and FARM [1]. WARM identifies sensors that are related to each other in a sliding window containing the latest w rounds using association rule mining. When the reading of one of those sensors is missing, it uses the readings of the other related sensors to estimate the missing reading. FARM is similar to WARM except that it does not use the concept of sliding window and it considers the freshness of data.

Cross-references

▶ Association Rule Mining on Streams
▶ Data Quality
▶ Sensor Networks
▶ Stream Data Analysis

Recommended Reading

1. Gruenwald L., Chok H., and Aboukhamis M. Using data mining to estimate missing sensor data. In Proc. 7th IEEE ICDM Workshop on Optimization-Based Data Mining Techniques with Applications, 2007, pp. 207–212.

2. Madden S., Franklin M., Hellerstein J., and Hong W. TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst., 30(1):122–173, 2005.

3. Papadimitriou S., Sun J., and Faloutsos C. Pattern discovery in multiple time-series. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 697–708.

Data Exchange

LUCIAN POPA
IBM Almaden Research Center, San Jose, CA, USA

Synonyms

Data translation; Data migration; Data transformation

Definition

Data exchange is the problem of materializing an instance of a target schema, given an instance of a source schema and a specification of the relationship between the source schema and the target schema. More precisely, a *data exchange setting* is a quadruple of the form $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where **S** is the source schema, **T** is the target schema, Σ_{st} is a schema mapping that expresses constraints between **S** and **T**, and Σ_t is a set of constraints on **T**. Such a setting gives rise to the following *data exchange problem*: given an instance I over the source schema **S**, find an instance J over the target schema **T** such that I and J together satisfy the schema mapping Σ_{st}, and J satisfies the target constraints Σ_t. Such an instance J is called a *solution* for I in the data exchange setting. In general, many different solutions for an instance I may exist. The main focus of the data exchange research is to study the space of all possible solutions, to identify the "best" solutions to materialize in a practical application, and to develop algorithms for computing such a best solution.

Historical Background

The first systems supporting the restructuring and translation of data were built several decades ago. An early such system was EXPRESS [21], which performed data exchange between hierarchical schemas. The need for systems supporting data exchange has persisted over the years and has become more pronounced with the proliferation of data in various formats ranging from traditional relational database schemas to semi-structured/XML schemas and scientific formats. An example of a modern data exchange system is Clio [18,20], a schema mapping prototype developed at IBM Almaden Research Center and in collaboration with University of Toronto that influenced both theoretical and practical aspects of data exchange.

The data exchange problem is related to the data integration problem [16] in the sense that both problems are concerned with management of data stored in heterogeneous formats. The two problems, however, are different for the following reasons. In data exchange, the main focus is on actually *materializing* a target instance (i.e., a solution) that reflects the source data as accurately as possible. This presents a challenge, due to the inherent under-specification of the relationship between the source and the target, which means that in general there are many different ways to materialize such a target instance. In contrast, a target instance need not be materialized in data integration. There, the main focus is on answering queries posed over the target schema using views that express the relationship between the target and source schemas.

Fagin et al. [8] were the first to formalize the data exchange problem and to embark on an in-depth investigation of the foundational and algorithmic issues that surround it. Their framework focused on data exchange settings in which **S** and **T** are relational schemas, Σ_{st} is a set of tuple-generating dependencies (tgds) between **S** and **T**, also called *source-to-target tgds*, and Σ_t is a set of tgds and equality-generating dependencies (egds) on **T**. Fagin et al. isolated a class of solutions for the data exchange problem, called *universal solutions*, and showed that they have good properties that justify selecting them as the preferred solutions in data exchange. Universal solutions are solutions that can be homomorphically mapped into every other solution; thus, intuitively, universal solutions are the most general solutions. Moreover, in a precise sense, universal solutions represent the entire space of solutions. One of the main results in [8] is that, under fairly general conditions (*weak acyclicity* of the set of target tgds), a *canonical* universal solution can be computed (if solutions exist) in polynomial time, by using the classical chase procedure [2].

In general, universal solutions need not be unique. Thus, in a data exchange setting, there may be many universal solutions for a given source instance. Fagin, Kolaitis and Popa [9] addressed the issue of further isolating a "best" universal solution, by using the concept of the *core* of a graph or a structure [14]. By definition, the core of a structure is the smallest substructure that is also a homomorphic image of that structure. Since all universal solutions for a source instance I are homomorphically equivalent, it follows that they all have the same core (up to isomorphism). It is then shown in [9] that this core is also a universal solution, and hence the smallest universal solution. The uniqueness of the core of a universal solution together with its minimality make the core an ideal solution for data exchange. In a series of papers that started with [9] and continued with [12,13], it was shown that the core of the universal solutions can be computed in polynomial time, for data exchange settings where Σ_{st} is a set of source-to-target tgds and Σ_t is the union of a weakly-acyclic set of tgds with a set of edgs. This is in contrast with the general case of computing the core of an arbitrary structure, for which it is known that, unless P=NP, there is no polynomial-time algorithm.

There are quite a few papers on data exchange and theory of schema mappings that extended or made use of the concepts and results introduced in [8,9]. Some of the more representative ones addressed: extensions to XML data exchange [1], extensions to peer data exchange [11], the study of solutions under the closed-world assumption (CWA) [17], combined complexity of data exchange [15], schema mapping composition [10,19] and schema mapping inversion [7]. The Clio system, which served as both motivation and implementation playground for data exchange, was the first to use source-to-target dependencies as a language for expressing schema mappings [20]. Mapping constraints, expressed as either embedded dependencies (which comprise tgds and egds) or as equalities between relational or SQLS expressions, also play a central role in the model management framework of Bernstein and Melnik [3].

Foundations

Given a source schema **S** and a target schema **T** that are assumed to be disjoint, a *source-to-target dependency* is, in general, a formula of the form $\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \chi_{\mathbf{T}}(\mathbf{x}))$, where $\phi_{\mathbf{S}}(\mathbf{x})$ is a formula, with free variables **x**, over the source schema **S**, and $\chi_{\mathbf{T}}(\mathbf{x})$ is a formula, with free variables **x**, over the target schema **T**. The notation **x** signifies a vector of variables $x_1,\ldots x_k$. A *target* dependency is, in general, a formula over the target schema **T** (the formalism used to express a target dependency may be different in general from those used for the source-to-target dependencies). The source schema may also have dependencies that are assumed to be satisfied by every source instance. Source dependencies do not play a direct role in data exchange, because the source instance is *given.*

The focus in [8] and in most of the subsequent papers on data exchange theory is on the case when **S** and **T** are relational schemas and when the dependencies are given as *tuple-generating dependencies (tgds) and equality-generating dependencies (egds)* [2]. More precisely, each source-to-target dependency in Σ_{st} is assumed to be a tgd of the form

$$\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x},\mathbf{y})),$$

where $\phi_{\mathbf{S}}(\mathbf{x})$ is a conjunction of atomic formulas over **S** and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over **T**. All the variables in **x** are assumed to appear in $\phi_{\mathbf{S}}(\mathbf{x})$. Moreover, each target dependency in Σ_t is either a tgd (of the form shown below left) or an egd (of the form shown below right):

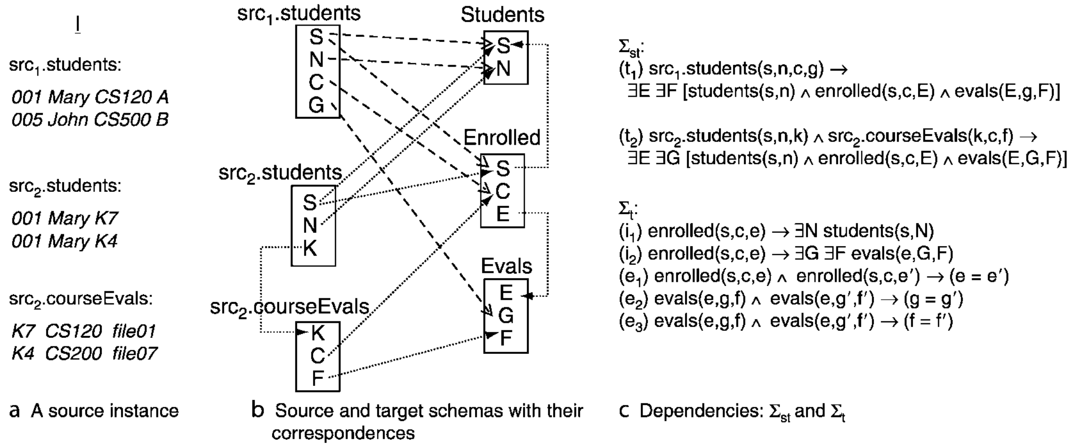
$$\forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x},\mathbf{y})) \quad \forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow (x_1 = x_2))$$

In the above, $\phi_{\mathbf{T}}(\mathbf{x})$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas over **T**, where all the variables in **x** appear in $\phi_{\mathbf{T}}(\mathbf{x})$, and x_1, x_2 are among the variables in **x**. An often used convention is to drop the universal quantifiers in front of a dependency, and implicitly assume such quantification. However, the existential quantifiers are explicitly written down.

Source-to-target tgds are a natural and expressive language for expressing the relationship between a source and a target schema. Such dependencies are semi-automatically derived in the Clio system [20] based on correspondences between the source schema and the target schema. In turn, such correspondences can either be supplied by a human expert or discovered via schema matching techniques. Source-to-target tgds are also equivalent to the language of "sound" *global-and-local-as-view (GLAV)* assertions often used in data integration systems [16].

It is natural to take the target dependencies to be tgds and egds: these two classes together comprise the (embedded) implicational dependencies [6]. However, it is somewhat surprising that tgds, which were originally "designed" for other purposes (as constraints), turn out to be ideally suited for specifying desired data transfer.

Example 1. Figure 1b shows a source schema (on the left) and a target schema (on the right) with



src$_1$.students:

001 Mary CS120 A
005 John CS500 B

src$_2$.students:

001 Mary K7
001 Mary K4

src$_2$.courseEvals:

K7 CS120 file01
K4 CS200 file07

Σ_{st}:

(t_1) src$_1$.students(s,n,c,g) $\rightarrow$
$\exists E\, \exists F$ [students(s,n) $\wedge$ enrolled(s,c,E) $\wedge$ evals(E,g,F)]

(t_2) src$_2$.students(s,n,k) $\wedge$ src$_2$.courseEvals(k,c,f) $\rightarrow$
$\exists E\, \exists G$ [students(s,n) $\wedge$ enrolled(s,c,E) $\wedge$ evals(E,G,F)]

Σ_t:

(i_1) enrolled(s,c,e) $\rightarrow \exists N$ students(s,N)
(i_2) enrolled(s,c,e) $\rightarrow \exists G\, \exists F$ evals(e,G,F)
(e_1) enrolled(s,c,e) $\wedge$ enrolled(s,c,e$'$) $\rightarrow$ (e = e$'$)
(e_2) evals(e,g,f) $\wedge$ evals(e,g$'$,f$'$) $\rightarrow$ (g = g$'$)
(e_3) evals(e,g,f) $\wedge$ evals(e,g$'$,f$'$) $\rightarrow$ (f = f$'$)

a A source instance

b Source and target schemas with their correspondences

c Dependencies: Σ_{st} and Σ_t

Data Exchange. Figure 1. A data exchange example.

correspondences between their attributes. The source schema models two different data sources or databases, src_1 and src_2, each representing data about students. The first source consists of one relation, $src_1.students$, while the second source consists of two relations, $src_2.students$ and $src_2.courseEvals$. The attributes S, N, C, G, F represent, respectively,“student id,” “student name,” “course,” “grade” (only in src_1), and “file evaluation” (a written evaluation that a student receives for a course; only in src_2). The attribute K in src_2 is used to link students with the courses they take; more concretely, K plays the role of a foreign key in $src_2.students$ and the role of a key in $src_2.courseEvals$. As seen in the instance in Fig. 1a, information in the two sources may overlap: the same student can appear in both sources, with each source providing some information that the other does not have (e.g., either grade or file evaluation).

The two data sources are mapped into a target schema with three relations: *students* (with general student information), *enrolled* (listing course entries for each student), and *evals* (with evaluation entries per student and per course). The attribute E (evaluation id) is used to link enrollment entries with the associated evaluation records (E is a foreign key in *enrolled* and a key in *evals*). Similarly, the attribute S (student id) links *enrolled* with *students*.

The relationship between the individual attributes in the schemas is described by the arrows or correspondences that “go” between the attributes. However, the more precise mapping between the schemas is given by the set $\Sigma_{st} = \{t_1, t_2\}$ of source-to-target tgds that is shown in Fig. 1c.

The first source-to-target tgd, t_1, specifies that for each tuple in $src_1.students$ there must exist three corresponding tuples in the target: one in *students*, one in *enrolled*, and one in *evals*. Moreover, t_1 specifies how the four components of the source tuple (i.e., s, n, c, g) must appear in the target tuples. The tgd also specifies the existence of “unknown” values (via the existential variables E and F) for the target attributes that do not have any corresponding attribute in the source. Note that existential variables can occur multiple times; in the example, it is essential that the same variable E is used in both *enrolled* and *evals* so that the association between students, courses and their grades is not lost in the target.

The second source-to-target tgd, t_2, illustrates a case where the source pattern (the premise of the tad) is not limited to one tuple of a relation but encodes a join between multiple relations. In general, not all variables in the source pattern must occur in the target (e.g., k does not occur in the target). In this example, t_2 plays a “complementary” role to t_1, since it maps a different source that contains file evaluations rather than grades.

The target dependencies in Σ_t are formulas expressed solely in terms of the target schema that further constrain the space of possible target instances. In this example, the tgds i_1 and i_2 are inclusion dependencies that encode referential integrity constraints from *enrolled* to *students* and *evals*, respectively. The egds e_1, e_2 and e_3 encode functional dependencies. that must be satisfied. In particular, e_1 requires that a student and a course must have a unique evaluation id, while e_2 and e_3 together specify that the evaluation id must be a key for *evals*.

Solutions In general, in a data exchange setting (**S**, **T**, Σ_{st},Σ_t), there can be zero or more *solutions* J for a given source instance I. In other words, there can be zero or more target instances J such that: (1) J satisfies the target dependencies in Σ_t, and (2) I together with J satisfy the source-to-target dependencies in Σ_{st}. The latter condition simply means that the instance $\langle I, J\rangle$ that is obtained by considering together all the relations in I and J satisfies Σ_{st}. Note that $\langle$I, $J\rangle$ is an instance over the union of the schemas **S** and **T**.

Example 2. Figure 2 illustrates three target instances that are plausible for the source instance I shown in Fig. 1a, and given the dependencies Σ_{st} and Σ_t in Fig. 1b. Consider the first instance J_0, shown in Fig. 2a. It can be seen that $\langle I, J_0\rangle$ satisfies all the source-to-target dependencies in Σ_{st}; in particular, for any combination of the source data in I that satisfies the premises of some source-to-target tad in Σ_{st}, the "required" target tuples exist in J_0. Note that in J_0, the special values E_1,...E_4, F_1, F_2, G_3 and G_4 are used to represent "unknown" values, that is, values that do not occur in the source instance. Such values are called *labeled nulls* or *nulls* and are to be distinguished from the values occurring in the source instance, which are called *constants*. (See the later definitions.) It can then be seen that J_0 fails to satisfy the set Σ_t of target dependencies; in particular, the egd e_1 is not satisfied (there are two *enrolled* tuples for student 001 and course CS120 having different evaluation ids, E_1 and E_3). Thus, J_0 is not a solution for I.

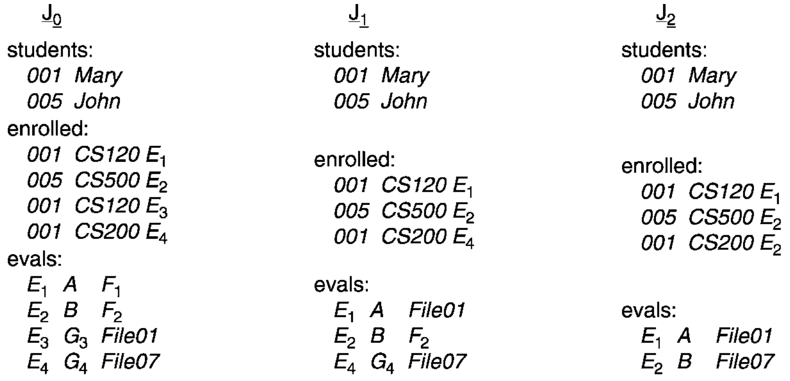
On the other hand, the two instances J_1 and J_2 shown in Fig. 2b and c, respectively, are both solutions for I. The main difference between J_1 and J_2 is that J_2 is more "compact": the same null E_2 is used as an evaluation id for two different pairs of student and course (in contrast, J_1 has different nulls, E_2 and E_4).

In this example, J_1 and J_2 illustrate two possible ways of filling in the target that both satisfy the given specification. In fact, there are infinitely many possible solutions: one could choose other nulls or even constants instead of E_1, E_2,..., or one could add "extra" target tuples, and still have all the dependencies satisfied. This raises the question of which solutions to choose in data exchange and whether some solutions are better than others.

Universal solutions. A key concept introduced in [8] is that of *universal solutions*, which are the most general among all the possible solutions.

Let $\underline{\text{Const}}$ be the set, possibly infinite, of all the values (also called *constants*) that can occur in source instances. Moreover, assume an infinite set $\underline{\text{Var}}$ of values, called *labeled nulls*, such that $\underline{\text{Var}} \cap \underline{\text{Const}} = \emptyset$. The symbols I, I', I_1, I_2,... are reserved for instances over the source schema **S** and with values in $\underline{\text{Const}}$. The symbols J, J', J_1, J_2,... are reserved for instances over the target schema **T** and with values in $\underline{\text{Const}} \cup \underline{\text{Var}}$. All the target instances considered, and in particular, the solutions of a data exchange problem, are assumed to have values in $\underline{\text{Const}} \cup \underline{\text{Var}}$. If J is a target instance, then $\underline{\text{Const}}(J)$ denotes the set of all constants occurring in J, and $\underline{\text{Var}}(J)$ denotes the set of labeled nulls occurring in J.

Let J_1 and J_2 be two instances over the target schema. A *homomorphismh* : $J_1 \to J_2$ is a mapping from $\underline{\text{Const}}(J_1) \cup \underline{\text{Var}}(J_1)$ to $\underline{\text{Const}}(J_2) \cup \underline{\text{Var}}(J_2)$ such that: (1) $h(c) = c$, for every $c \in \underline{\text{Const}}(J_1)$; (2) for every tuple



J_0
students:
001 Mary
005 John
enrolled:
001 CS120 E_1
005 CS500 E_2
001 CS120 E_3
001 CS200 E_4
evals:
E_1 *A* F_1
E_2 *B* F_2
E_3 G_3 *File01*
E_4 G_4 *File07*

J_1
students:
001 Mary
005 John
enrolled:
001 CS120 E_1
005 CS500 E_2
001 CS200 E_4
evals:
E_1 *A* *File01*
E_2 *B* F_2
E_4 G_4 *File07*

J_2
students:
001 Mary
005 John
enrolled:
001 CS120 E_1
005 CS500 E_2
001 CS200 E_2
evals:
E_1 *A* *File01*
E_2 *B* *File07*

a A target instance that is not a solution

b A universal solution

c A solution that is not universal

Data Exchange. Figure 2. Examples of target instances.

t in a relation R of J_1, the tuple $h(t)$ is in the relation R of J_2 (where, if $t=(a_1,...a_s)$, then $h(t)=(h(a_1),...,h(a_s))$). The instance J_1 is *homomorphically equivalent* to the instance J_2 if there are homomorphisms $h: J_1 \rightarrow J_2$ and $h': J_2 \rightarrow J_1$.

Consider a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$. If I is a source instance, then a *universal solution for I* is a solution J for I such that for every solution J' for I, there exists a homomorphism $h: J \rightarrow J'$.

Example 3. The solution J_2 in Fig. 2c is not universal. In particular, there is no homomorphism from J_2 to the solution J_1 in Fig. 2b. Specifically, the two *enrolled* tuples (005, $CS500$, E_2) and $(001, CS200, E_2)$ of J_2 cannot be mapped into tuples of J_1 (since E_2 cannot be mapped into both E_2 and E_4). Thus, J_2 has some "extra" information that does not appear in all solutions (namely, the two *enrolled* tuples for (005, $CS500$) and $(001, CS200)$ sharing the same evaluation id). In contrast, a universal solution has only information that can be homomorphically mapped into every possible solution. It can be shown that J_1 is such a universal solution, since it has a homomorphism to every solution (including J_2).

As the above example suggests, universal solutions are the preferred solutions in data exchange, because they are at least as general as any other solution (i.e., they do not introduce any "extra" information).

Computing universal solutions with the chase. Fagin et al. [8] addressed the question of how to check the existence of a universal solution and how to compute one, if one exists. They showed that, under a *weak acyclity* condition on the set Σ_t of target dependencies, universal solutions exist whenever solutions exist. Moreover, they showed that, under the same condition, there is a polynomial-time algorithm for computing a *canonical* universal solution, if a solution exists; this algorithm is based on the classical chase procedure.

Intuitively, the following procedure is applied to produce a universal solution from a source instance I: start with the instance $\langle I, \emptyset \rangle$ that consists of I for the source, and the empty instance for the target; then chase $\langle I, \emptyset \rangle$ with the dependencies in Σ_{st} and Σ_t in some arbitrary order and for as long as they are applicable. Each chase step either adds new tuples in the target or attempts to equate two target values (possibly failing, as explained shortly). More concretely, let $\langle I, J \rangle$ denote an intermediate instance in the chase process (initially $J = \emptyset$). Chasing with a source-to-target tgd $\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ amounts to the following: check whether there is a vector $\mathbf{a}$ of values that interprets $\mathbf{x}$ such that $I \vDash \phi_{\mathbf{S}}(\mathbf{a})$, but there is no vector $\mathbf{b}$ of values that interprets $\mathbf{y}$ such that $J \vDash \psi_{\mathbf{T}}(\mathbf{a}, \mathbf{b})$; if such $\mathbf{a}$ exists, then add new tuples to J, where fresh new nulls $\mathbf{Y}$ interpret the existential variables $\mathbf{y}$, such that the resulting target instance satisfies $\psi_{\mathbf{T}}(\mathbf{a}, \mathbf{Y})$. Chasing with a target tgd is defined similarly, except that only the target instance is involved. Chasing with a target egd $\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow (x_1 = x_2)$ amounts to the following: check whether there is a vector $\mathbf{a}$ of values that interprets $\mathbf{x}$ such that $J \vDash \phi_{\mathbf{T}}(\mathbf{a})$ and such that $a_1 \neq a_2$; if this is the case, then the chase step attempts to identify a_1 and a_2, as follows. If both a_1 and a_2 are constants then the chase fails; it can be shown that there is no solution (and no universal solution) in this case. If one of a_1 and a_2 is a null, then it is replaced with the other one (either a null or a constant); this replacement is global, throughout the instance J. If no more chase steps are applicable, then the resulting target instance J is a universal solution for I.

Example 4. Recall the earlier data exchange scenario in Fig. 1. Starting from the source instance I, the source-to-target tgds in Σ_{st} can be applied first. This process adds all the target tuples that are "required" by the tuples in I and the dependencies in Σ_{st}. The resulting target instance after this step is an instance that is identical, modulo renaming of nulls, to the instance J_0 in Fig. 2a. Assume, for simplicity, that the result *is* J_0.

The chase continues by applying the dependencies in Σ_t to J_0. The tgds (i_1) and (i_2) are already satisfied. However, the egd (e_1) is not satisfied and becomes applicable. In particular, there are two *enrolled* tuples with the same student id and course but different evaluation ids (E_1 and E_3). Since E_1 and E_3 are nulls, the chase with e_1 forces the replacement of one with the other. Assume that E_3 is replaced by E_1. After the replacement, the two *enrolled* tuples become identical (hence, one is a duplicate and is dropped). Moreover, there are now two *evals* tuples with the same evaluation id (E_1). Hence, the egds e_2 and e_3 become applicable. As a result, the null G_3 is replaced by the constant A and the null F_1 is replaced by the constant *file*01. The resulting target instance is the instance J_1 in Fig. 2b, which is a canonical universal solution for I.

As a remark on the expressive power of dependencies (and of their associated chase), note how the above process has merged, into the same tuple, information from two different sources (i.e., the grade, and respectively, the file, for student 001 and course *CS*120). Also note that the chase is, inherently, a recursive procedure.

In general, the chase with an arbitrary set of target tgds and egds may not terminate. Hence, it is natural to ask for sufficient conditions for the termination of the chase. An extensively studied condition that guarantees termination is that the target tgds in Σ_t form a *weakly acyclic set of tgds* (the latter is also known as a set of constraints with *stratified witnesses*) [8,5]. Two important classes of dependencies that are widely used in database dependency theory, namely sets of full target tgds and acyclic sets of inclusion dependencies, are special cases of weakly acyclic sets of tgds.

The following theorem summarizes the use of chase in data exchange and represents one of the main results in [8].

Theorem 1 *Assume a data exchange setting* (**S**, **T**, Σ_{st},Σ_t) where Σ_{st} *is a set of source-to-target tgds, and* Σ_t *is the union of a weakly acyclic set of tgds with a set of egds. Then: (1) The existence of a solution can be checked in polynomial time. (2) A universal solution exists if and only if a solution exists. (3) If a solution exists, then a universal solution can be produced in polynomial time using the chase.*

The weak acyclicity restriction is essential for the above theorem to hold. In fact, it was shown in [15] that if the weak-acyclicity restriction is removed, the problem of checking the existence of solutions becomes undecidable.

Multiple universal solutions and core. In general, in a data exchange setting, there can be many universal solutions for a given source instance. Nevertheless, it has been observed in [9] that all these universal solutions share one common part, which is the *core* of the universal solutions. The core of the universal solutions is arguably the "best" universal solution to materialize, since it is the unique most compact universal solution. It is worth noting that the chase procedure computes a universal solution that may not necessarily be the core. So, additional computation is needed to produce the core.

A universal solution that is not a core necessarily contains some "redundant" information which does not appear in the core. Computing the core of the universal solutions could be performed, conceptually, in two steps: first, materialize a canonical universal solution by using the chase, then remove the redundancies by taking to the core. The second step is tightly related to *conjunctive query minimization* [4], a procedure that is in general intractable. However, by exploiting the fact that in data exchange, the goal is on computing the core of a universal solution rather than that of an arbitrary instance, polynomial-time algorithms were shown to exist for certain large classes of data exchange settings. Specifically, for data exchange settings where Σ_{st} is a set of arbitrary source-to-target tgds and Σ_t is a set of egds, two polynomial-time algorithms, the *blocks* algorithm and the *greedy* algorithm, for computing the core of the universal solutions were given in [9]. By generalizing the blocks algorithm, this tractability case was further extended in [12] to the case where the target tgds are full and then in [13] to the more general case where the target tgds form a weakly acyclic set of tgds.

Further results on query answering. The semantics of data exchange problem (i.e., which solution to materialize) is one of the main issues in data exchange. Another main issue is that of answering queries formulated over the target schema. Fagin et al. [8] adopted the notion of the "certain answers" in incomplete databases for the semantics of query answering in data exchange. Furthermore, they studied the issue of when can the certain answers be computed based on the materialized solution alone; in this respect, they showed that in the important case of unions of conjunctive queries, the certain answers can be obtained simply by running the query on an arbitrary universal solution and by eliminating the tuples that contain nulls. This in itself provided another justification for the "goodness" of universal solutions. The followup paper [9] further investigated the use of a materialized solution for query answering; it showed that for the larger class of existential queries, evaluating the query on the core of the universal solutions gives the best approximation of the certain answers. In fact, if one redefines the set of certain answers to be those that occur in every universal solution (rather than in every solution), then the core gives the exact answer for existential queries.

Key Applications

Schema mappings are the fundamental building blocks in information integration. Data exchange gives theoretical foundations for schema mappings, by studying the transformation semantics associated to a schema mapping. In particular, universal solutions are the main concept behind the "correctness" of any program, query or ETL flow that implements a schema mapping specification. Data exchange concepts are also essential in the study of the operators on schema mappings such as composition of sequential schema mappings and inversion of schema mappings.

Cross-references

▶ Data Integration
▶ Schema Mapping
▶ Schema Mapping Composition

Recommended Reading

1 Arenas M. and Libkin L. XML data exchange: consistency and query answering. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 13–24.
2. Beeri C. and Vardi M.Y. A proof procedure for data dependencies. J. ACM, 31(4):718–741, 1984.
3. Bernstein P.A. and Melnik S. Model management 2.0: manipulating richer mappings. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1–12.
4. Chandra A.K. and Merlin P.M. Optimal implementation of conjunctive queries in relational data bases. In Proc. 9th Annual ACM Symp. on Theory of Computing, 1977, pp. 77–90.
5. Deutsch A. and Tannen V. XML queries and constraints, containment and reformulation. Theor. Comput. Sci., 336 (1):57–87, 2005.
6. Fagin R. Horn clauses and database dependencies. J. ACM, 29 (4):952–985, 1982.
7. Fagin R. Inverting schema mappings. ACM Trans. Database Syst., 32(4), 2007.
8. Fagin R., Kolaitis P.G., Miller R.J., and Popa L. Data exchange: semantics and query answering. Theor. Comput. Sci., 336 (1):89–124, 2005.
9. Fagin R., Kolaitis P.G., and Popa L. Data exchange: getting to the core. ACM Trans. Database Syst., 30(1):174–210, 2005.
10. Fagin R., Kolaitis P.G., Popa L., and Tan W.-C. Composing schema mappings: second-order dependencies to the rescue. ACM Trans. Database Syst., 30(4):994–1055, 2005.
11. Fuxman A., Kolaitis P.G., Miller R.J., and Tan W.-C. Peer data exchange. ACM Trans. Database Syst., 31(4): 1454–1498, 2006.
12. Gottlob G. Computing cores for data exchange: new algorithms and practical solutions. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005.
13. Gottlob G. and Nash A. Data exchange: computing cores in polynomial time. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 40–49.
14. Hell P. and Nešetřil J. The core of a graph. Discrete Math, 109:117–126, 1992.
15. Kolaitis P.G., Panttaja J., and Tan W.C. The complexity of data exchange. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 30–39.
16. Lenzerini M. Data Integration: A Theoretical Perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
17. Libkin L. Data exchange and incomplete information. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 60–69.
18. Miller R.J., Haas L.M., and Hernández M.A. Schema mapping as query discovery. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 77–88.
19. Nash A., Bernstein P.A., and Melnik S. Composition of mappings given by embedded dependencies. ACM Trans. Database Syst., 32(1):4, 2007.
20. Popa L., Velegrakis Y., Miller R.J., Hernández M.A., and Fagin R. Translating Web data. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 598–609.
21. Shu N.C., Housel B.C., Taylor R.W., Ghosh S.P., and Lum V.Y. EXPRESS: A Data EXtraction, Processing, and REStructuring System. ACM Trans. Database Syst., 2(2):134–174, 1977.

Data Expiration

▶ Temporal Vacuuming

Data Extraction

▶ Screen Scraper

Data Flow Diagrams

▶ Activity Diagrams

Data Fusion

▶ Semantic Data Integration for Life Science Entities

Data Fusion in Sensor Networks

Aman Kansal, Feng Zhao
Microsoft Research, Redmond, WA, USA

Synonyms

Distributed sensor fusion

Definition

Data fusion in sensor networks is defined as the set of algorithms, processes, and protocols that combine data from multiple sensors. The goal may be to extract information not readily apparent in an individual sensor's data, improve the quality of information compared to that provided by any individual data, or improve the operation of the network by optimizing usage of its resources.

For instance, the output of a magnetic sensor and an audio sensor may be combined to detect a vehicle (new information), outputs of multiple vibration sensors may be combined to increase the signal to noise ratio (improving quality), or a passive infrared sensor may be combined with a camera in a people detection network to reduce the frame-rate of the camera for conserving energy (improving operation).

The sensors fused may be of the same or different types. Key features of data fusion in sensor networks, that distinguish it from other methods to combine multiple sources of data, are that the former methods are designed to (i) reduce the amount of communication among and from the sensors, and (ii) increase the life-time of the sensor network when all or some of the sensors are battery operated. The methods used to combine the data may leverage past observations from the same sensors, previously known models of the sensed phenomenon, and other information in addition to the sensor data.

Historical Background

Data fusion in sensor networks is founded on the methods developed for fusion of data in several older systems that used multiple sensors but not under the same system constraints as are typical of sensor networks. The Radar systems used in World War II present one of the first notable examples of such systems. The advantages that these systems demonstrated in robustness to failure of a fraction of sensors increased quality of data due to increased dimensionality of the measurement space. Since then, better discrimination between available hypotheses have led to the use of fusion methods in many sensor systems.

Most of the algorithms for sensor data processing can be viewed as derivatives of the Kalman filter and related Bayesian methods [7,9]. Decentralized forms of these methods have been developed to take data from multiple sensors as input and produced a single higher quality fused output [15,16]. Latest advances in these areas are discussed at the IEEE Sensor Array and Multichannel Signal Processing Workshop and the International Conference on Information Fusion among other venues. Other related works on fusion of sensor data are found in computer vision, tracking, and defense applications [2,3,5,6,8].

The achievable advantage in reducing signal distortion through fusion of multiple sensor inputs has been derived using information theoretic methods for general and Gaussian models for the phenomenon and sensor noise [1,12,13].

Foundations

The basic data fusion problem may be expressed as follows. The sensor network is deployed to measure a phenomenon represented by a state vector $\mathbf{x}^{(\mathbf{t})}$. For example, if the phenomenon is an object being tracked, $\mathbf{x}^{(\mathbf{t})}$ may represent a vector of position and velocity of the object at time *t*. The observation model at sensor *i*, that relates the observation **z** to state **x**, is assumed to be Gaussian in many of the works, for computational tractability:

$$\mathbf{z}_i(t) = \mathbf{H}_i(\mathbf{x}(t)) + \mathbf{w}_i(t) \tag{1}$$

Linear models where **H** is a matrix are often used.

The *belief* about the phenomenon at time *t* is defined to be the *a posteriori* distribution of **x**:

$$p(\mathbf{x}|\mathbf{z}_1, \ldots, \mathbf{z}_n) \tag{2}$$

where *n* is the number of sensors. The belief is sufficient to characterize the phenomenon and compute typical statistics such as the expected value of **x** and its residual uncertainty after the estimation.

The centralized methods to determine this belief from the observations require a knowledge of measurements $\mathbf{z}_i$ from all the sensors. The decentralized Kalman filters, such as [15,16], typically assume that each of the *n* sensors in the sensor network is connected to every other sensor and an $O(n^2)$ communication overhead is acceptable. This design may be used for fusion in systems with a small number of sensors or when high data rate communication links are present among sensors, such as networks of defense vehicles. However, such a high overhead is not acceptable in sensor networks based on embedded and wireless platforms. In these systems, both due to the large number of sensors and the low data rates supported by their radios for battery efficiency, the fusion methods must minimize the communication requirements. Such fusion methods are said to be distributed.

One approach [17,18] to realize a distributed fusion method is to selectively use only a subset of the large number of sensors that have the most relevant information about the phenomenon being sensed. The fusion method then is also required to provide for appropriate selection of these most informative sensors and to dynamically adapt the set of selected sensor as the phenomenon evolves over time. To this end, a quantitative measure of the usefulness of a sensor for fusion is introduced, referred to as the *information content* of that sensor. An information utility function is defined:

$$\psi : \mathcal{P}(\mathcal{R}^d) \to \mathcal{R} \tag{3}$$

that acts on the class $\mathcal{P}(\mathcal{R}^d)$ of all probability distributions on $\mathcal{R}^d$ and returns a real number, with d being the dimension of $\mathbf{x}$. Specific choices of ψ are derived from information measures known from information theory, such as entropy, the Fischer information matrix, the size of the covariance ellipsoid for Gaussian phenomena models, sensor geometry based measures and others. An example form of ψ if information theoretic entropy is used, is:

$$\psi(p_{\mathbf{x}}) = \int_S p_{\mathbf{x}}(x) log(p_{\mathbf{x}}(x))dx \tag{4}$$

where S represents the domain of $\mathbf{x}$ and $p_{\mathbf{x}}$ is its probability distribution. Let $U \subset \{1,...,n\}$ be the set of sensors whose measurements have been incorporated into the belief, i.e., the current belief is:

$$p(\mathbf{x}|\{\mathbf{z}_i\}_{i\in U}) \tag{5}$$

If the measurement from sensor j is also selected for inclusion in the computation of belief, the belief becomes:

$$p(\mathbf{x}|\{\mathbf{z}_i\}_{i\in U} \cup \{\mathbf{z}_j\}) \tag{6}$$

To select the sensor that has the maximum information content, the sensor j should be selected to maximize the information utility of the belief after including $\mathbf{z}_j$. Noting that j is to be selected from set $A = \{1,...,n\} - U$, the best sensor $\hat{j}$ is:

$$\hat{j} = arg_{j\in A} max\ \psi\left(p(\mathbf{x}|\{\mathbf{z}_i\}_{i\in U} \cup \{\mathbf{z}_j\})\right) \tag{7}$$

However, in practice, the knowledge about $\mathbf{z}_j$ is not available before having selected j. The most likely best sensor j can then be selected by computing the expectation of the information utility with respect to $\mathbf{z}_j$:

$$\hat{j} = arg_{j\in A} max\ E_{\mathbf{z}_j}[\psi\left(p(\mathbf{x}|\{\mathbf{z}_i\}_{i\in U} \cup \{\mathbf{z}_j\})\right)|\{\mathbf{z}_i\}_{i\in U}] \tag{8}$$

among other options.

Also, the cost of communication from a sensor is explicitly modeled. Suppose the current belief is held at sensor l, referred to as the leader node. Suppose $M_c(l, j)$ denotes the cost of communication to sensor j. Then the sensor selection method choses the best sensor as follows:

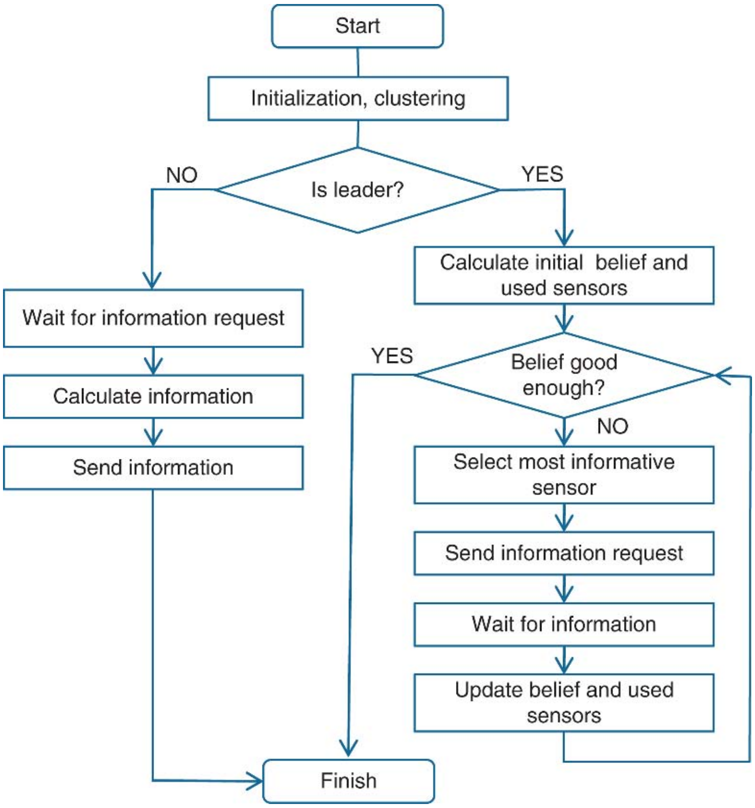
$$\hat{j} = arg_{j\in A}\max[\alpha M_u(j) - (1-\alpha)M_c(l,j)] \tag{9}$$

where $M_u(j)$ denotes the expectation of the information utility as expressed in (8), and $\alpha \in [0,1]$ balances the contribution from M_u and M_c.

These fundamentals can be used to develop a distributed data fusion algorithm for a sensor network. Suppose the sensor nodes are synchronized in time, and the phenomenon is initially detected at time $t = 0$. At this time, a known distributed leader election algorithm may be executed to select a node l as the leader, which computes the initial belief using only its own observation. It now selects the next sensor to be included in the belief calculation using (9). The process continues until the belief is known to a satisfactory quality as characterized by known statistical measures. The flowchart of the fusion algorithm followed at all the nodes is shown in Fig. 1.

The sensor selection process used in the algorithm above is a greedy one – at each selection step, it only considers a single sensor that optimizes the selection metric. It is possible that selecting multiple sensors at the same time yields a better choice. This can be achieved at the cost of higher computation complexity by selecting a set of sensors instead a single sensor j in (9), using appropriate modifications to the cost and utility metrics.

The distributed fusion method described above limits the number of sensors used and hence significantly reduces the communication overhead as compared to the $O(n^2)$ overhead of decentralized methods. It is well-suited for problems where the sensed phenomenon is localized, such as an object being tracked, since the number of most informative nodes selected can then yield fused results close to that provided by the entire network.


Start
Initialization, clustering
Is leader?
NO
YES
Wait for information request
Calculate information
Send information
Calculate initial belief and used sensors
Belief good enough?
YES
NO
Select most informative sensor
Send information request
Wait for information
Update belief and used sensors
Finish

Data Fusion in Sensor Networks. Figure 1. Distributed sensor fusion based on selecting only the most informative and cost effective sensors.

In another approach, a distributed Kalman filter is derived. The sensing model is as expressed before in (1). However, instead of using a centralized Kalman filter to estimate the state $\mathbf{x}$, n micro-Kalman filters, each executing locally at the n sensor nodes, using the measurements from only the local node, are used. In addition, two consensus problems are solved using a method that requires communication only with one hop wireless neighbors [10,11]. This approach is more relevant when the observations from all the nodes in the network are important, such as when measuring a distributed phenomenon.

Distributed fusion methods that are tolerant to communication losses and network partitioning have also been developed using message passing on junction trees [14] and techniques from assumed density filtering [4].

Additionally, distributed methods are also available for cases where the sensed phenomenon is not itself required to be reproduced but only some of its properties are to be obtained. These properties may be global and depend on the measurements of all the sensors in the network, such as the number of targets in the region, the contours of a given phenomenon value in a heat map, or tracking relations among a set of objects [17].

The field of data fusion in sensor networks is rapidly evolving with new advances being presented frequently at forums including the ACM/IEEE IPSN and ACM SenSys.

Key Applications

Infrastructure monitoring, pervasive health-care, defense, scientific experimentation, environmental sensing, urban monitoring, home automation, supply-chain control, industrial control, business process monitoring, security.

Data Sets

Several data sets collected from experimental sensor network deployments are available for researchers to test their data fusion methods:

Center for Embedded Networked Sensing, http://www.sensorbase.org/

Intel Lab Data, http://db.csail.mit.edu/labdata/labdata.html

Cross-references

- ▶ Data Aggregation in Sensor Networks
- ▶ Data Estimation in Sensor Networks
- ▶ In-Network Query Processing

Recommended Reading

1. Berger T., Zhang Z., and Vishwanathan H. The CEO problem. IEEE Trans. Inform. Theory, 42(3):887–902, 1996.
2. Brooks R.R. and Iyengar S.S. Multi-sensor fusion: Fundamentals and applications with software. Prentice-Hall, Englewood, Cliffs, NJ, 1997.
3. Crowley J.L. and Demazeau Y. Principles and techniques for sensor data fusion. Signal Process., 32(1–2):5–27, 1993.
4. Funiak S., Guestrin C., Paskin M., and Sukthankar R. Distributed inference in dynamical systems. In Advances in Neural Information Processing Systems 19, B. Scholkopf, J. Platt, and T. Hoffman (eds.). MIT, Cambridge, MA, 2006, pp. 433–440.
5. Hall D.L. and McMullen S.A.H. Mathematical Techniques in Multisensor Data Fusion. Artech House, 2004.
6. Isard M. and Blake A. Condensation – conditional density propagation for visual tracking. Int. J. Comput. Vision, 29(1):5–28, 1998.
7. Jazwinsky A. Stochastic processes and filtering theory. Academic, New York, 1970.
8. Lodaya M.D. and Bottone R. Moving target tracking using multiple sensors. In Proc. SPIE, Vol. 4048, 2000, pp. 333–344.
9. Maybeck P.S. The kalman filter: an introduction to concepts. In Autonomous Robot vehicles, I.J. Cox and G.T. Wilfong, Eds. Springer-Verlag New York, New York, NY, 1990, pp. 194–204.
10. Olfati-Saber R. Distributed kalman filtering for sensor networks. In Proc. 46th IEEE Conf. on Decision and Control. 2007.
11. Olfati-Saber R. and Shamma J.S. Consensus filters for sensor networks and distributed sensor fusion. In Proc. 44th IEEE Conf. on Decision and Control. 2005.
12. Oohama Y. The rate distortion function for the quadratic Gaussian CEO problem. IEEE Trans. Inform. Theory, 44(3), 1998.
13. Pandya A., Kansal A., Pottie G.J., and Srivastava M.B. Fidelity and resource sensitive data gathering. In 42nd Allerton Conference, 2004.
14. Paskin M., Guestrin C., and McFadden J. A robust architecture for distributed inference in sensor networks. In Proc. 4th Int. Symp. Inf. Proc. in Sensor Networks, 2005.
15. Rao B., Durrant-Whyte H., and Sheen J. A fully decentralized multi-sensor system for tracking and surveillance. Int. J. Robot. Res., 12(1):20–44, 1993.
16. Speyer J.L. Computation and Transmission requirements for a decentralized linear-quadratic-Gaussian control problem. IEEE Trans. Automat. Control, 24(2):266–269, 1979.
17. Zhao F., Liu J., Liu J., Guibas L., and Reich J. Collaborative Signal and Information Processing: An Information Directed Approach. In Proc. IEEE, 91(8):1199–1209, 2003.
18. Zhao F. and Guibas L. Wireless Sensor Networks: An Information Processing Approach. Morgan Kaufmann, 2004.

Data Gathering

▶ Data Acquisition and Dissemination in Sensor Networks

Data Grids

▶ Storage Grid

Data Imputation

▶ Data Estimation in Sensor Networks

Data Inconsistencies

▶ Data Conflicts

Data Integration

▶ Information Integration

Data Integration Architectures and Methodology for the Life Sciences

ALEXANDRA POULOVASSILIS
University of London, London, UK

Definition

Given a set of biological data sources, *data integration* is the process of creating an integrated resource combining data from the data sources, in order to allow queries and analyses that could not be supported by the individual data sources alone. Biological data sources are characterized by their high degree of heterogeneity, in terms of their data model, query interfaces and query processing capabilities, data types used, and nomenclature adopted for actual data values. Coupled with the variety, complexity and volumes of biological data that are becoming increasingly available, integrating biological data sources poses many challenges, and a number of methodologies, architectures and systems have been developed to support it.

Historical Background

If an application requires data from different data sources to be integrated in order to support users' queries and analyses, one possible solution is for the required data transformation and aggregation functionality to be encoded into the application's programs. However, this may be a complex and lengthy process, and may also affect the robustness and maintainability of the application. These problems have motivated the development of architectures and methodologies which abstract out data transformation and aggregation functionality into generic data integration software.

Much work has been done since the early 1990s in developing architectures and methodologies for integrating biological data sources in particular. Many systems have been developed which create and maintain integrated data resources: examples of significant systems are DiscoveryLink [7], K2/Kleisli [3], Tambis [6], SRS [16], Entrez [5], BioMart [4]. The main aim of such systems is to provide users with the ability to formulate queries and undertake analyses on the integrated resource which would be very complex or costly if performed directly on the individual data sources, sometimes prohibitively so.

Providing access to a set of biological data sources via one integrated resource poses several challenges, mainly arising from the large volumes, variety and complexity of othe data, and the autonomy and heterogeneity of the data sources [2,8,9]. Data sources are developed by different people in differing research environments for differing purposes. Integrating them to meet the needs of new users and applications requires the reconciliation of their different data models, data representation and exchange formats, content, query interfaces, and query processing capabilities. Data sources are in general free to change their data formats and content without considering the impact this may have on integrated resources derived from them. Integrated resources may themselves serve as data sources for higher-level integrations, resulting in a network of dependencies between biological data resources.

Foundations

Three main methodologies and architectures have been adopted for biological data integration, *materialized*, *virtual* and *link-based*:

- With *materialized integration*, data from the data sources is imported into a data warehouse and it is transformed and aggregated as necessary in order to conform to the warehouse schema. The warehouse is the integrated resource, typically a relational database. Queries can be formulated with respect to the warehouse schema and their evaluation is undertaken by the database management system (DBMS), without needing to access the original data sources.
- With *virtual integration*, a schema is again created for the integrated resource. However, the integrated resource is represented by this schema, and the schema is not populated with actual data. Additional mediator software is used to construct mappings between the data sources and the integrated schema. The mediator software coordinates the evaluation of queries that are formulated with respect to the integrated schema, utilizing the mappings and the query processing capabilities of the database or file management software at the data sources. Data sources are accessed via additional "wrapper" software for each one, which presents a uniform interface to the mediator software.
- With *link-based integration* no integrated schema is created. Users submit queries to the integration software, for example via a web-based user interface. Queries are formulated with respect to data sources, as selected by the user, and the integration software provides additional capabilities for facilitating query formulation and speeding up query evaluation. For example, SRS [16] maintains indexes supporting efficient keyword-based search over data sources, and also maintains cross-references between different data sources which are used to augment query results with links to other related data.

A link-based integration approach can be adopted if users will wish to query the data sources directly, will only need to pose keyword and navigation-style queries, and the scientific hypotheses that they will be investigating will not require any significant transformation or aggregation of the data. Otherwise, the adoption of a materialized or virtual integration approach is indicated. The link-based integration approach is discussed further in [1] and in the entry on Pathway Databases.

A key characteristic of materialized or virtual integration is that the integrated resource can be queried as though it were itself a single data source rather than an integration of other data sources: users and applications do not need to be aware of the schemas or formats of the original data sources, only the schema/format of the integrated resource. Materialized integration is usually chosen for query performance reasons: distributed access to remote data sources is avoided and sophisticated query optimization techniques can be applied to queries submitted to the data warehouse. Other advantages are that it is easier to clean and annotate the source data than by using mappings within a virtual integration approach. However, maintaining a data warehouse can be complex and costly, and virtual integration may be the preferred option if these maintenance costs are too high, or if it is not possible to extract data from the data sources, or if access to the latest versions of the data sources is required.

Examples of systems that adopt the materialized integration approach are GUS [3], BioMart [4], Atlas [14], BioMap [12]. With this approach, the standard methodology and architecture for data warehouse creation and maintenance can be applied. This consists of first extracting data from the data sources and transporting it into a "staging" area. Data from the data sources will need to be re-extracted periodically in order to identify changes in the data sources and to keep the warehouse up-to-date. Data extraction from each data source may be either full extraction or incremental extraction. With the former, the entire source data is re-extracted every time while with the latter it is only relevant data that has changed since the previous extraction. Incremental extraction is likely to be more efficient but for some data sources it may not be possible to identify the data that has changed since the last extraction, for example due to the limited functionality provided by the data sources, and full extraction may be the only option. After the source data has been brought into the staging area, the changes from the previous versions of the source data are determined using "difference" algorithms (in the case of full re-extraction) and the changed data is transformed into the format and data types specified by the warehouse schema. The data is then "cleaned" i.e., errors and inconsistencies are removed, and it is loaded into the warehouse. The warehouse is likely to contain materialized views which transform and aggregate in various ways the detailed data from the data sources. View maintenance capabilities provided by the DBMS can be used to update such materialized views following insertions, updates and deletions of the detailed data. It is also possible to create and maintain additional "data marts" each supporting a set of specialist users via a set of additional views specific to their requirements. The warehouse serves as the single data source for each data mart, and a similar process of extraction, transformation, loading and aggregating occurs to create and maintain the data mart.

One particular characteristic of biological data integration, as compared with business data integration for example, is the prevalence of both automated and manual annotation of data, either prior to its integration, or during the integration process, or both. For example, the Distributed Annotation System (DAS) (http://www.biodas.org) allows annotations to be generated and maintained by the owners of data resources, while the GUS data warehouse supports annotations that track the origins of data, information about algorithms or annotation software used to derive new inferred data, and who performed the annotation and when. Being able to find out the provenance of any data item in an integrated resource is likely to be important for users, and this is even more significant in biological data integration where multiple annotation processes may be involved.

Another characteristic of biological data integration is the wide variety of nomenclatures adopted by different data sources. This greatly increases the difficulty of aggregating their data and has led to the proposal of many standardized ontologies, taxonomies and controlled vocabularies to help alleviate this problem e.g., from the Gene Ontology (GO) Consortium, Open Biomedical Ontologies (OBO) Consortium, Microarray Gene Expression Data (MGED) Society and Proteomics Standards Initiative (PSI). The role of ontologies in scientific data integration is discussed in the entry on Ontologies in Scientific Data Integration. Another key issue is the need to resolve possible inconsistencies in the ways that biological entities are identified within the data sources. The same biological entity may be identified differently in different data sources or, conversely, the same identifier may be used for different biological entities in different data sources. There have been a number of initiatives to address the problem of inconsistent identifiers e.g., the Life Sciences Identifiers (LSID) initiative and the International Protein Index (IPI). Despite such initiatives, there is still a legacy of large numbers of non-standardized identifiers in biological datasets and therefore techniques are needed for associating biological entities independently of their identifiers. One technique is described in [12] where a clustering approach is used to identify sets of data source entities that are likely to refer to the same real-world entity.

Another area of complexity is that data sources may evolve their schemas over time to meet the needs of new applications or new experimental techniques (henceforth, the term "data source schema" is used to encompass also the data representation and exchange formats of data sources that are not databases). Changes in the data source schemas may require modification of the extraction-transformation-loading (ETL), view materialization and view maintenance procedures. Changes in the warehouse schema may impact on data marts derived from it and on the procedures for maintaining these.

Turning now to virtual data integration, architectures that support virtual data integration typically include the following components:

- A Repository for storing information about data sources, integrated schemas, and the mappings between them.
- A suite of tools for constructing integrated schemas and mappings, using a variety of automatic and interactive methods.
- A Query Processor for coordinating the evaluation of queries formulated with respect to an integrated schema; the Query Processor first reformulates such a query, using the mappings in the Repository, into an equivalent query expressed over the data source schemas; it then optimizes the query and evaluates it, submitting as necessary sub-queries to the appropriate data source Wrappers and merging the results returned by them.
- An extensible set of Wrappers, one for each type of data source being integrated; each Wrapper extracts metadata from its data source for storage in the Repository, translates sub-queries submitted to it by the Query Processor into the data source's query formalism, issues translated sub-queries to the data source, and translates sub-query results returned by the data source into the Query Processor's data model for further post-processing by the Query Processor.

An integrated schema may be defined in terms of a standard data modeling language, or it may be a source-independent ontology defined in an ontology language and serving as a "global" schema for multiple potential data sources beyond the specific ones that are being integrated (as in TAMBIS for example). The two main integration methodologies are *top-down* and *bottom-up*. With top-down integration, the integrated schema, *IS*, is first constructed, or may already exist from previous schema design, integration or standardization efforts. The set of mappings, *M*, between *IS* and the data source schemas are then defined. With bottom-up integration, an initial version of *IS* and *M* are first constructed – for example, these may be based on just one of the data source schemas. The integrated schema *IS* and the set of mappings *M* are then incrementally extended by considering in turn each of the other data source schemas: for each object *O* in each source schema, *M* is modified so as to encompass the mapping between *O* and *IS*, if it is possible to do so using the current *IS*; otherwise, *IS* extended as necessary in order to encompass the data represented by *O*, and *M* is then modified accordingly.

A mixed top-down/bottom-up approach is also possible: an initial *IS* may exist from a previous design or standardization activity, but it may need to be extended in order to encompass additional data arising from the set of data sources being integrated within it. With either top-down, bottom-up or mixed integration, it is possible that *IS* will not need to encompass all of the data of the data sources, but only a subset of the data which is sufficient for answering key queries and analyses – this avoids the possibly complex process of constructing a complete integrated schema and set of mappings.

There are a number of alternatives to defining the set of mappings *M* above, and different data integration systems typically adopt different approaches: with the *global-as-view* (GAV) approach, each mapping relates one schema object in *IS* with a view that is defined over the source schemas; with the *local-as-view* (LAV) approach, each mapping relates one schema object in one of the source schemas with a view defined over *IS*; and with the *global-local-as-view* (GLAV) approach, each mapping relates a view over a source schema with a view over *IS* [10,11]. Another approach is the *both-as-view* [13] approach supported by the AutoMed system. This provides a set of primitive transformations on schemas, each of which adds, deletes or renames a schema object. The semantic relationships between objects in the source schemas and the integrated schema are represented by reversible sequences of such transformations. The ISPIDER project [15] uses AutoMed for virtual integration of several Grid-enabled proteomics data sources.

In addition to the approach adopted for specifying mappings between source and integrated schemas, different systems may also make different assumptions about the degree of semantic overlap between the data sources: some systems assume that each data source contributes to a different part of the integrated resource (e.g., K2/Kleisli); some relax this assumption but do not undertake any aggregation of duplicate or overlapping data that may be present in the data sources (e.g., TAMBIS); and some can support aggregation at both the schema and the data levels (e.g., AutoMed). The degree of data source overlap impacts on the degree of schema and data aggregation that will need to be undertaken by the mappings, and hence on their complexity and the design effort involved in specifying them. The complexity of the mappings in turn impacts on the sophistication of the query processing mechanisms that will be needed in order to optimize and evaluate queries posed on the integrated schema.

Key Applications

- Integrating, analyzing and annotating genomic data.
- Predicting the functional role of genes and integrating function-specific information.
- Integrating organism-specific information.
- Integrating and analyzing chemical compound data and metabolic pathway data to support drug discovery.
- Integrating protein family, structure and pathway data with gene expression data, to support functional genomics data analysis.
- Integrating, analyzing and annotating proteomics data sources recording data from experiments on protein separation and identification.
- Supporting systems biology research.
- Integrating phylogenetic data sources for genealogical reconstruction.
- Integrating data about genomic variations in order to analyze the impact of genomic variations on health.
- Integrating genomic and proteomic data with clinical data to support personalized medicine.

Future Directions

Identifying semantic correspondences between different data sources is a necessary prerequisite to integrating them. This is still largely a manual and time-consuming process undertaken with significant input from domain experts. Semi-automatic techniques are being developed to alleviate this problem, for example name-based or structural comparisons of source schemas, instance-based matching at the data level to determine overlapping schema concepts, and annotation of data sources with terms from ontologies to facilitate automated reasoning over the data sources.

The transformation of source data into an integrated resource may result in loss of information, for example due to imprecise knowledge about the semantic correspondences between data sources. This is leading to research into capturing within the integrated resource incomplete and uncertain information, for example using probabilistic or logic-based representations and reasoning.

Large amounts of information are potentially available in textual form within published scientific articles. Automated techniques are being developed for extracting information from such sources using grammar and rule-based approaches, and then integrating this information with other structured or semi-structured biological data.

Traditional approaches to data integration may not be sufficiently flexible to meet the needs of distributed communities of scientists. Peer-to-peer data integration techniques are being developed in which there is no single administrative authority for the integrated resource and it is maintained instead by a community of peers who exchange, transform and integrate data in a pair-wise fashion and who cooperate in query processing over their data.

Finally, increasing numbers of web services are being made available to access biological data and computing resources - see, for example, the entry on Web Services and the Semantic Web for Life Science Data. Similar problems arise in combining such web services into larger-scale workflows as in integrating biological data sources: the necessary services are often created independently by different parties, using different technologies, formats and data types, and therefore additional code needs to be developed to transform the output of one service into a format that can be consumed by another.

Cross-references

- ▶ Data Provenance
- ▶ Ontologies and Life Science Data Management
- ▶ Pathway Databases
- ▶ Provenance in Scientific Databases
- ▶ Semantic Data Integration for Life Science Entities
- ▶ Web Services and the Semantic Web for Life Science Data

Recommended Reading

1. Cohen-Boulakia S., Davidson S., Froidevaux C., Lacroix Z., and Vidal M.E. Path-based systems to guide scientists in the maze of biological data sources. J. Bioinformatics Comput. Biol., 4(5):1069–1095, 2006.
2. Davidson S., Overton C., and Buneman P. Challenges in integrating biological data sources. J. Comput. Biol., 2(4): 557–572, 1995.
3. Davidson S.B., et al. K2/Kleisli and GUS: experiments in integrated access to genomic data sources. IBM Syst. J., 40 (2):512–531, 2001.
4. Durnick S., et al. Biomart and Bioconductor: a powerful link between biological databases and microarray data analysis. Bioinformatics, 21(16):3439–3440, 2005.
5. Entrez – the life sciences search engine. Available at: http://www.ncbi.nlm.nih.gov/Entrez
6. Goble C.A., et al. Transparent access to multiple bioinformatics information sources. IBM Syst. J., 40(2):532–551, 2001.
7. Haas L.M., et al. Discovery Link: a system for integrated access to life sciences data sources. IBM Syst. J., 40(2):489–511, 2001.
8. Hernandez T. and Kambhampati S. Integration of biological sources: current systems and challenges ahead. ACM SIGMOD Rec., 33(3):51–60, 2004.
9. Lacroix Z. and Critchlow T. Bioinformatics: Managing Scientific Data. Morgan Kaufmann, San Francisco, CA, 2004.
10. Lenzerini M. Data integration: a theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
11. Madhavan J. and Halevy A.Y. Composing mappings among data sources. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 572–583.
12. Maibaum M., et al. Cluster based integration of heterogeneous biological databases using the AutoMed toolkit. In Proc. 2nd Int. Workshop on Data Int. in the Life Sciences, 2005, pp. 191–207.
13. McBrien P. and Poulovassilis A. Data integration by bi-directional schema transformation rules. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 227–238.
14. Shah S.P., et al. Atlas – a data warehouse for integrative bioinformatics. BMC Bioinformatics, 6:34, 2005.
15. Zamboulis L., et al. Data access and integration in the ISPIDER Proteomics Grid. In Proc. 3rd Int. Workshop on Data Integration in the Life Sciences, 2006, pp. 3–18.
16. Zdobnov E.M., Lopez R., Apweiler R., and Etzold T. The EBI SRS Server – recent developments. Bioinformatics, 18(2):368–373, 2002.

Data Integration in Web Data Extraction System

Marcus Herzog[1,2]
[1]Vienna University of Technology, Vienna, Austria
[2]Lixto Software GmbH, Vienna, Austria

Synonyms

Web information integration and schema matching; Web content mining; Personalized Web

Definition

Data integration in Web data extraction systems refers to the task of providing a uniform access to multiple Web data sources. The ultimate goal of Web data integration is similar to the objective of *data integration* in database systems. However, the main difference is that Web data sources (i.e., Websites) do not feature a structured data format which can be accessed and queried by means of a query language. In contrast, *Web data extraction systems* need to provide an additional layer to transform Web pages into (semi)-structured data sources. Typically, this layer provides an extraction mechanism that exploits the inherent document structure of HTML pages (i.e., the document object model), the content of the document (i.e., text), visual cues (i.e., formatting and layout), and the inter document structure (i.e., hyperlinks) to extract data instances from the given Web pages. Due to the nature of the Web, the data instances will most often follow a semi-structured schema. Successful data integration then requires to solve the task of reconciling the syntactic and semantic heterogeneity, which evolves naturally from accessing multiple independent Web sources. Semantic heterogeneity can be typically observed both on the schema level and the data instance level. The output of the Web data integration task is a unified data schema along with consolidated data instances that can be queried in a structured way. From an operational point of view, one can distinguish between on-demand integration of Web data (also referred to as metasearch) and off-line integration of Web data similar to the ETL process in data warehouses.

Historical Background

The concept of data integration was originally conceived by the database community. Whenever data are not stored in a single database with a single data schema, data integration needs to resolve the structural and semantic heterogeneity found in databases built by different parties. This is a problem that researches have been addressing for years [8]. In the context of web data extraction systems, this issue is even more pressing due to the fact that web data extraction systems usually deal with schemas of semi-structured data, which are more flexible both from a structural and semantic perspective. The Information Manifold [12] was one of the systems that not only integrated relational databases but also took Web sources into account. However, these Web sources were structured in nature and were queried by means of a Web form. Answering a query involved a join across the relevant web sites. The main focus of the work was on providing a mechanism to describe declaratively the contents and query capabilities of the available information sources.

Some of the first research systems which covered the aspects of data integration in the context of Web data extraction systems were ANDES, InfoPipes, and a framework based on the Florid system. These systems combine languages for web data extraction with mechanisms to integrate the extracted data in a homogeneous data schema. ANDES [15] is based on the Extensible Stylesheet Language Transformations (XSLT) for both data extraction and data integration tasks. The ANDES framework merges crawler technology with XML-based extraction techniques und utilized templates, (recursive) path expressions, and regular expressions for data extraction, mapping, and aggregation. ANDES is primarily a software framework, requiring application developers to manually build a complete process from components such as Data Retriever, Data Extractor, Data Checker, and Data Exporter.

The InfoPipes system [10] features a workbench for visual composition of processing pipelines utilizing XML-based processing components. The components are defined as follows: Source, Integration, Transformation, and Deliverey. Each of those components features a configuration dialog to interactively define the configuration of the component. The components can be arranged on the canvas of the workbench and can be connected to form information processing pipelines, thus the name InfoPipes. The Source component utilized ELOG programs [13] to extract semi-structured data from Websites. All integration tasks are subsequently performed on XML data. The Integration component also features a visual dialog to specify the reconciliation of the syntactic and semantic heterogeneity in the XML documents. These specifications are then translated into appropriate XSLT programs to perform the reconciliation during runtime.

In [14] an integrated framework for Web exploration, wrapping, data integration, and querying is described. This framework is based on the Florid [13] system and utilizes a rule-based object-oriented language which is extended by Web accessing capabilities and structured document analysis. The main objective of this framework is to provide a unified framework – i.e., data model and language – in which all tasks (from Web data extraction to data integration and querying) are performed. Thus, these tasks are not necessarily separated, but can be closely intertwined. The framework allows for modeling the Web both on the page level as well as on the parse-tree level. Combined rules for wrapping, mediating, and Web exploration can be expressed in the same language and with the same data model.

More recent work can be found in the context of Web content mining. Web content mining focuses on extracting useful knowledge from the Web. In Web content mining, Web data integration is a fundamental aspect, covering both schema matching and data instance matching.

Foundations

Semi-structured Data

Web data extraction applications often utilize XML as data representation formalism. This is due to the fact that the semi-structured data format naturally matches with the HTML document structure. In fact, XHTML is an application of XML. XML provides a common syntactic format. However, it does not offer any means for addressing the semantic integration challenge. Query languages such as XQuery [5], XPath [1] or XSLT [11] provide the mechanism to manipulate the structure and the content of XML documents. These languages can be used as basis for implementing integration systems. The semantic integration aspect has to be dealt with on top of the query language.

Schema and Instance Matching

The main issue in data integration is the finding the semantic mapping between a number of data sources. In the context of Web extraction systems, these sources are web pages or more generally websites. There are three distinct approaches to the matching problem: manual, semiautomatic, or automatic matching. In the manual approach, an expert needs to define the mapping by using a toolset. This is of course time consuming. Automatic schema matching in contrast is AI-complete [3] and well researched in the database community [16], but typically still lacks reliability. In the semiautomatic approach, automatic matching algorithms suggest certain mappings which are validated by an expert. This approach saves time due to filtering out the most relevant matching candidates.

An example for manual data integration framework is given in [6]. The Harmonize framework [9] deals with business-to-business (B2B) integration on the "information" layer by means of an ontology-based mediation. It allows organizations with different data standards to exchange information seamlessly without having to change their proprietary data schemas. Part of the Harmonize framework is a mapping tool that allows for manually generating mapping rules between two XML schema documents.

In contrast to the manual mapping approach, automated schema mapping has to rely on clues that can be derived from the schema descriptions: utilizing the similarities between the names of the schema elements or taking the amount of overlap of data values or data types into account.

While matching schemas is already a time-consuming task, reconciling the data instances is even more cumbersome. Due to the fact that data instances are extracted from autonomous and heterogeneous websites, no global identifiers can be assumed. The same real world entity may have different textual representations, e.g., "CANOSCAN 3000ex 48 Bit, 1200×2400 dpi" and "Canon CanoScan 3000ex, 1200 × 2400dpi, 48Bit." Moreover, data extracted from the Web is often incomplete and noisy. In such a case, a perfect match will not be possible. Therefore, a similarity metric for text joins has to be defined. Most often the widely used and established cosine similarity metric [17] from the information retrieval field is used to identify string matches. A sample implementation of text joins for Web data integration based on an unmodified RDBMS is given in [17]. Due to the fact that the number of data instances is much higher than the number of schema elements, data instance reconciliation has to rely on automatic procedures.

Web Content Mining

Web content mining uses the techniques and principles of data mining to extract specific knowledge from Web pages. An important step in Web mining is the integration of extracted data. Due to the fact that Web mining has to work on Web-scale, a fully automated process is required. In the Web mining process, Web data records are extracted from Web pages which serve as input for the subsequent processing steps. Due to the large scale approach of Web mining it calls for novel methods that draw from a wide range of fields spanning data mining, machine learning, natural language processing, statistics, databases, and information retrieval [4].

Key Applications

Web data integration is required for all applications that draw data from multiple Web sources and need to interpret the data in a new context. The following main application areas can be identified:

Vertical Search

In contrast to web search as provided by major search engines, vertical search targets a specific domain such as e.g., travel offers, job offers, or real estate offers. Vertical search applications typically deliver more structured results than conventional web search engines. While the focus of the web search is to cover the breath of all available websites and deliver the most relevant websites for a given query, vertical search typically searches less websites, but with the objective to retrieve relevant data objects. The output of a vertical search query is a result set that contains e.g., the best air fares for a specific route. Vertical search also needs to address the challenge of searching the deep Web, i.e., extracting data by means of automatically utilizing web forms. Data integration in the context of vertical search is both important for interface matching, i.e., merge the source query interfaces and map onto a single query interface, and result data object matching, where data extracted from the individual websites is matched against a single result data model.

Web Intelligence

In Web Intelligence applications, the main objective is to gain new insights from the data extracted on the

Web. Typical application fields are market intelligence, competitive intelligence, and price comparison. Price comparison applications are probably the most well known application type in this field. In a nutshell these applications aggregate data from the Web and integrate different Web data sources according to a single data schema to allow for easy analysis and comparison of the data. Schema matching and data reconciliation are important aspects with this type of applications.

Situational Applications

Situational applications are a new type of application where people with domain knowledge can build an application in a short amount of time without the need to setup an IT project. In the context of the Web, Mashups are addressing these needs. With Mashups, readymade widgets are used to bring together content extracted from multiple websites. Additional value is derived by exploiting the relationship between the different sources, e.g., visualizing the location of offices in a mapping application. In this context, Web data integration is required to reconcile the data extracted from different Web sources and to resolve the references to real world objects.

Cross-references

- ▶ Data Integration
- ▶ Enterprise Application Integration
- ▶ Enterprise Information Integration
- ▶ Schema Matching
- ▶ Web Data Extraction

Recommended Reading

1. Baumgartner R., Flesca S., and Gottlob G. Visual web information extraction with Lixto. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 119–128.
2. Berglund A., Boag S., Chamberlin D., Rernandez M.F., Kay M., Robie J., and Simeon J. (eds.). XML XPath Language 2.0. W3C Recommendation, 2007.
3. Bernstein P.A., Melnik S., Petropoulos M., and Quix C. Industrial-strength schema matching. ACM SIGMOD Rec., 33(4):38–43, 2004.
4. Bing L. and Chen-Chuan-Chang K. Editorial: special issue on web content mining. ACM SIGKDD Explorations Newsletter, 6(2):1–4, 2004.
5. Boag S., Chamberlin D., Fernandez M.F., Florescu D., Robie J., and Simeon J. (eds.). XQuery 1.0. An XML Query Language. W3C Recommendation, 2007.
6. Fodor O. and Werthner E. Harmonise: a step toward an interoperable e-tourism marketplace. Intl. J. Electron. Commerce, 9(2):11–39, 2005.
7. Gravano L., Panagiotis G.I., Koudas N., and Srivastava D. Text joins in an RDBMS for web data integration. In Proc. 12th Int. World Wide Web Conference, 2003, pp. 90–101.
8. Halevy A., Rajaraman A., and Ordille J. Data integration: the teenage years. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 9–18.
9. Harmonise Framework. Available at: http://sourceforge.net/projects/hmafra/.
10. Herzog M. and Gottlob G. InfoPipes: a flexible framework for m-commerce applications. In Proc. 2nd Int. Workshop on Technologies for E-Services, 2001, pp. 175–186.
11. Kay M. (ed.). XSL Transformations. Version 2.0. W3C Recommendation, 2007.
12. Kirk T., Levy A.Y., Sagiv Y., and Srivastava D. The information manifold. In Proc. Working Notes of the AAAI Spring Symp. on Information Gathering from Heterogeneous, Distributed Environments. Stanford University. AAAI Press, 1995, pp. 85–91.
13. Ludäscher B., Himmeröder R., Lausen G., May W., and Schlepphorst C. Managing semistructured data with florid: a deductive object-oriented perspective. Inf. Syst., 23(9):589–613, 1998.
14. May W. and Lausen G. A uniform framework for integration of information from the web. Inf. Syst., 29:59–91, 2004.
15. Myllymaki J. Effective web data extraction with standard XML technologies. Comput. Networks, 39(5):653–644, 2002.
16. Rahm E. and Bernstein P.A. A survey of approaches to automatics schema matching. VLDB J., 10(4):334–350, 2001.
17. Salton G. and McGill M.J. Introduction to Modern Information Retrieval. McGraw-Hill, New York, NY, 1983.

Data Integrity Services

▶ Security Services

Data Lineage

▶ Data Provenance

Data Manipulation Language

▶ Query Language

Data Map

▶ Thematic Map

Data Mart

Il-Yeol Song
Drexel University, Philadelphia, PA, USA

Definition

A data mart is a small-sized data warehouse focused on a specific subject. While a data warehouse is meant for an entire enterprise, a data mart is built to address the specific analysis needs of a business unit. Hence, a data mart can be defined as "a small-sized data warehouse that contains a subset of the enterprise data warehouse or a limited volume of aggregated data for specific analysis needs of a business unit, rather than the needs of the whole enterprise." Thus, an enterprise usually ends up having many data marts.

Key Points

While a data warehouse is for a whole enterprise, a data mart focuses on a specific subject of a specific business unit. Thus, the design and management of a data warehouse must consider the needs of the whole enterprise, while those of a data mart are focused on the analysis needs of a specific business unit such as the sales department or the finance department. Thus, a data mart shares the characteristics of a data warehouse, such as being subject-oriented, integrated, nonvolatile, and a time-variant collection of data [1]. Since the scope and goal of a data mart is different from a data warehouse, however, there are some important differences between them:

- While the goal of a data warehouse is to address the needs of the whole enterprise, the goal of a data mart is to address the needs of a business unit such as a department.
- While the data of a data warehouse is fed from OLTP (Online Transaction Processing) systems, those of a data mart are fed from the enterprise data warehouse.
- While the granularity of a data warehouse is raw at its OLTP level, that of a data mart is usually lightly aggregated for optimal analysis by the users of the business unit.
- While the coverage of a data warehouse is fully historical to address the needs of the whole enterprise, that of a data mart is limited for the specific needs of a business unit.

An enterprise usually ends up having multiple data marts. Since the data to all the data marts are fed from the enterprise data warehouse, it is very important to maintain the consistency between a data mart and the data warehouse as well as among data marts themselves. A way to maintain the consistency is to use the notion of conformed dimension. A conformed dimension is a standardized dimension or a master reference dimension that are shared across multiple data marts [2].

The technology for advanced data analysis for the business intelligence in the context of data mart environment is called OLAP (Online Analytic Processing). Two OLAP technologies are ROLAP (Relational OLAP) and MOLAP (Multidimensional OLAP). In ROLAP, data are structured in the form a star schema or a dimensional model. In MOLAP, data are structured in the form of multidimensional data cubes. For MOLAP, a specialized OLAP software is used to support the creation of data cubes and OLAP operations such as drill-down and roll-up.

Cross-references

► Active and Real-Time Data Warehousing
► Business Intelligence
► Data Mining
► Data Warehouse
► Data Warehouse Life-Cycle and Design
► Data Warehouse Maintenance, Evolution and Versioning
► Data Warehouse Metadata
► Data Warehouse Security
► Data Warehousing and Quality Data Management for Clinical Practice
► Data Warehousing for Clinical Research
► Data Warehousing Systems: Foundations and Architectures
► Dimension
► Extraction, Transformation, and Loading
► Multidimensional Modeling
► On-Line Analytical Processing
► Transformation

Recommended Reading

1. Inmon W.H. Building the Data Warehouse, 3rd edn. Wiley, New York, 2002.
2. Kimball R. and Ross M. The Data Warehouse Toolkit, 2nd edn. Wiley, New York, 2002.

Data Migration

▶ Data Exchange

Data Mining

Jiawei Han
University of Illinois at Urbana-Champaign, Urbana, IL, USA

Synonyms

Knowledge discovery from data; Data analysis; Pattern discovery

Definition

Data mining is the process of discovering knowledge or patterns from massive amounts of data. As a young research field, data mining represents the confluence of a number of research fields, including database systems, machine learning, statistics, pattern recognition, high-performance computing, and specific application fields, such as WWW, multimedia, and bioinformatics, with broad applications. As an interdisciplinary field, data mining has several major research themes based on its mining tasks, including pattern-mining and analysis, classification and predictive modeling, cluster and outlier analysis, and multidimensional (OLAP) analysis. Data mining can also be categorized based on the kinds of data to be analyzed, such as multirelational data mining, text mining, stream mining, web mining, multimedia (or image, video) mining, spatiotemporal data mining, information network analysis, biological data mining, financial data mining, and so on. It can also be classified based on the mining methodology or the issues to be studied, such as privacy-preserving data mining, parallel and distributed data mining, and visual data mining.

Historical Background

Data mining activities can be traced back to the dawn of early human history when data analysis methods (e.g., statistics and mathematical computation) were needed and developed for finding knowledge from data. As a distinct but interdisciplinary field, knowledge discovery and data mining can be viewed as starting at The First International Workshop on Knowledge Discovery from Data (KDD) in 1989. The first International Conference on Knowledge Discovery and Data Mining (KDD) was held in 1995. Since then, there have been a number of international conferences and several scientific journals dedicated to the field of knowledge discovery and data mining. Many conferences on database systems, machine learning, pattern recognition, statistics, and World-Wide Web have also published influential research results on data mining. There are also many textbooks published on data mining, such as [5,7,8,11,12], or on specific aspects of data mining, such as data cleaning [4] and web mining [2,9]. Recently, there is also a trend to organize dedicated conferences and workshops on mining specific kinds of data or specific issues on data mining, such as International Conferences on Web Search and Data Mining (WSDM).

Foundations

The overall knowledge discovery process usually consists of a few steps, including (i) data preprocessing, e.g., data cleaning and data integration (and possibly building up data warehouse), (ii) data selection, data transformation (and possibly creating data cubes by multidimensional aggregation), and feature extraction, (iii) data mining, (iv) pattern or model evaluation and justification, and (v) knowledge update and application. Data mining is an essential step in the knowledge discovery process.

As a dynamic research field, many scalable and effective methods have been developed for mining patterns and knowledge from an enormous amount of data, which contributes to theories, methods, implementations, and applications of knowledge discovery and data mining. Several major themes are briefly outlined below.

Mining Interesting Patterns from Massive Amount of Data

Frequent patterns are the patterns (e.g., itemsets, subsequences, or substructures) that occur frequently in data sets. This line of research started with association rule mining [1] and has proceeded to mining sequential patterns, substructure (or subgraph) patterns, and their variants. Many scalable mining algorithms have been developed and most of them explore the Apriori (or *downward closure*) property of frequent patterns, i.e., *any subpattern of a frequent pattern is frequent.* However, to make discovered patterns truly useful in many applications, it is important to study how to mine interesting frequent patterns [6], e.g., the patterns that satisfy certain constraints, patterns that reflect strong correlation relationships, compressed patterns, and the patterns with certain distinct features. The discovered patterns can also be used for classification, clustering, outlier analysis, feature selection (e.g., for index construction), and semantic annotation.

Scalable Classification and Predictive Modeling

There are many classification methods developed in machine learning [10] and statistics [8], including decision tree induction, rule induction, naive-Bayesian, Bayesian networks, neural networks, support vector machines, regression, and many statistical and pattern analysis methods [5,7,8,11,12]. Recent data mining research has been exploring scalable algorithms for such methods as well as developing new classification methods for handling different kinds of data, such as data streams, text data, web data, multimedia data, and high-dimensional biological data. For example, a pattern-based classification method, called DDPMine [3], that first extracts multidimensional features by discriminative frequent pattern analysis and then performs classification using these features has demonstrated high classification accuracy and efficiency.

Cluster and Outlier Analysis

Data mining research has contributed a great deal to the recent development of scalable and effective cluster analysis methods. New methods have been proposed to make partitioning and hierarchical clustering methods more scalable and effective. For example, the micro-clustering idea in BIRCH [13] has been proposed that first groups objects into tight, micro-clusters based on their inherent similarity, and then performs flexible and efficient clustering on top of a relatively small number of micro-clusters. Moreover, new clustering methodologies, such as density-based clustering, link-based clustering, projection-based clustering of high-dimensional space, user-guided clustering, pattern-based clustering, and (spatial) trajectory clustering methods have been developed and various applications have been explored, such as clustering high-dimensional microarray data sets, image data sets, and interrelated multi-relational data sets. Furthermore, outlier analysis methods have been investigated, which goes beyond typical statistical distribution-based or regression deviation-based outlier analysis, and moves towards distance-based or density-based outlier analysis, local outlier analysis, and trajectory outlier analysis.

Multidimensional (OLAP) Analysis

Each object/event in a dataset usually carries multidimensional information. Mining data in multidimensional space will substantially increase the power and flexibility of data analysis. By integration of data cube and OLAP (online analytical processing) technologies with data mining, the power and flexibility of data analysis can be substantially increased. Data mining research has been moving towards this direction with the proposal of OLAP mining, regression cubes, prediction cubes, and other scalable high-dimensional data analysis methods. Such multidimensional, especially high-dimensional, analysis tools will ensure that data can be analyzed in hierarchical, multidimensional structures efficiently and flexibly at user's finger tips. OLAP mining will substantially enhance the power and flexibility of data analysis and lead to the construction of easy-to-use tools for the analysis of massive data with hierarchical structures in multidimensional space.

Mining Different Kinds of Data

Different data mining methods are often needed for different kinds of data and for various application domains. For example, mining DNA sequences, moving object trajectories, time-series sequences on stock prices, and customer shopping transaction sequences require rather different sequence mining methodology. Therefore, another active research frontier is the development of data- or domain-specific mining methods. This leads to diverse but flourishing research on mining different kinds of data, including multi-relational data, text data, web data, multimedia data, geo-spatial data, temporal data, data streams, information networks, biological data, financial data, and science and engineering data.

Key Applications

Data mining claims a very broad spectrum of applications since in almost every domain, there is a need for scalable and effective methods and tools to analyze massive amounts of data. Two applications are illustrated here as examples:

Biological Data Mining

The fast progress of biomedical and bioinformatics research has led to the accumulation of an enormous amount of biological and bioinformatics data. However, the analysis of such data poses much greater challenges than traditional data analysis methods. For example, genes and proteins are gigantic in size (e.g., a DNA sequence could be in billions of base pairs), very sophisticated in function, and the patterns of their interactions are largely unknown. Thus it is a fertile field to develop sophisticated data mining methods for in-depth bioinformatics research. Substantial research is badly needed to produce powerful mining tools for biology and bioinformatics studies, including comparative genomics, evolution and phylogeny, biological data cleaning and integration, biological sequence analysis, biological network analysis, biological image analysis, biological literature analysis (e.g., PubMed), and systems biology. From this point view, data mining is still very young with respect to biology and bioinformatics applications.

Data Mining for Software Engineering

Software program executions potentially (e.g., when program execution traces are turned on) generate huge amounts of data. However, such data sets are rather different from the datasets generated from the nature or collected from video cameras since they represent the executions of program logics coded by human programmers. It is important to mine such data to monitor program execution status, improve system performance, isolate software bugs, detect software plagiarism, analyze programming system faults, and recognize system malfunctions.

Data mining for software engineering can be partitioned into static analysis and dynamic/stream analysis, based on whether the system can collect traces beforehand for post-analysis or it must react at real time to handle online data. Different methods have been developed in this domain by integration and extension of the methods developed in machine learning, data mining, pattern recognition, and statistics. For example, statistical analysis such as hypothesis testing approach can be performed on program execution traces to isolate the locations of bugs which distinguish program success runs from failure runs. Despite of its limited success, it is still a rich domain for data miners to research and further develop sophisticated, scalable, and real-time data mining methods.

Future Directions

There are many challenging issues to be researched further, and therefore, there are great many research frontiers in data mining. Besides the mining of biological data and software engineering data, as well as the above introduced advanced mining methodologies, a few more research directions are listed here.

Mining Information Networks

Information network analysis has become an important research frontier, with broad applications, such as social network analysis, web community discovery, cyberphysical network analysis, and network intrusion detection. However, information network research should go beyond explicitly formed, homogeneous networks (e.g., web page links, computer networks, and terrorist e-connection networks) and delve deeply into implicitly formed, heterogeneous, dynamic, interdependent, and multidimensional information networks, such as gene and protein networks in biology, highway transportation networks in civil engineering, theme-author-publication-citation networks in digital libraries, wireless telecommunication networks among commanders, soldiers and supply lines in a battle field.

Invisible Data Mining

It is important to build data mining functions as an invisible process in many systems (e.g., rank search results based on the relevance and some sophisticated, preprocessed evaluation functions) so that users may not even sense that data mining has been performed beforehand or is being performed and their browsing and mouse clicking are simply using the results of or further exploration of data mining. Google has done excellent invisible data mining work for web search and certain web analysis. It is highly desirable to introduce such functionality to many other systems.

Privacy-Preserving Data Mining

Due to the security and privacy concerns, it is appealing to perform effective data mining without disclosure of private or sensitive information to outsiders. Much research has contributed to this theme and it is expected that more work in this direction will lead to powerful as well as secure data mining methods.

Experimental Results

There are many experimental results reported in numerous conference proceedings and journals.

Data Sets

There are many, many data sets (mostly accessible on the web) that can be or are being used for data mining.

University of California at Irvine has an online repository of large data sets which encompasses a wide variety of data types, analysis tasks, and application areas. The website of UCI Knowledge Discovery in Databases Archive is http://kdd.ics.uci.edu.

Researchers and practitioners should work on real data sets as much as possible to generate data mining tools for real applications.

URL to Code

Weka (http://www.cs.waikato.ac.nz/ml/weka) presents a collection of machine learning and data mining algorithms for solving real-world data mining problems.

RapidMiner (http://rapid-i.com), which was previously called YALE (Yet Another Learning Environment), is a free open-source software for knowledge discovery, data mining, and machine learning.

IlliMine (IlliMine.cs.uiuc.edu) is a collection of data mining software derived from the research of the Computer Science department at the University of Illinois at Urbana-Champaign.

For frequent pattern mining, the organizers of the FIMI (Frequent Itemset Mining Implementations) workshops provides a repository for frequent itemset mining implementations at http://fimi.cs.helsinki.fi.

There are many other websites providing source or object codes on data mining.

Cross-references

- ▶ Association Rules
- ▶ Bayesian Classification
- ▶ Classification
- ▶ Classification by Association Rule Analysis
- ▶ Clustering Overview and Applications
- ▶ Data, Text, and Web Mining in Healthcare
- ▶ Decision Rule Mining in Rough Set Theory
- ▶ Decision Tree Classification
- ▶ Decision Trees
- ▶ Dimentionality Reduction
- ▶ Event pattern detection
- ▶ Event Prediction
- ▶ Exploratory data analysis
- ▶ Frequent graph patterns
- ▶ Frequent itemset mining with constraints
- ▶ Frequent itemsets and association rules
- ▶ Machine learning in Computational Biology
- ▶ Mining of Chemical Data
- ▶ Opinion mining
- ▶ Pattern-growth methods
- ▶ Privacy-preserving data mining
- ▶ Process mining
- ▶ Semi-supervised Learning
- ▶ Sequential patterns
- ▶ Spatial Data Mining
- ▶ Spatio-temporal Data Mining
- ▶ Stream Mining
- ▶ Temporal Data Mining
- ▶ Text Mining
- ▶ Text mining of biological resources
- ▶ Visual Association Rules
- ▶ Visual Classification
- ▶ Visual Clustering
- ▶ Visual Data Mining

Recommended Reading

1. Agrawal R. and Srikant R. Fast algorithms for mining association rules. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
2. Chakrabarti S. Mining the Web: Statistical Analysis of Hypertex and Semi-Structured Data. Morgan Kaufmann, 2002.
3. Cheng H., Yan X., Han J., and Yu P.S. Direct discriminative pattern mining for effective classification. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 169–178.
4. Dasu T. and Johnson T. Exploratory Data Mining and Data Cleaning. Wiley, 2003.
5. Duda R.O., Hart P.E., and Stork D.G. Pattern Classification, 2nd edn. Wiley, New York, 2001.
6. Han J., Cheng H., Xin D., and Yan X. Frequent pattern mining: Current status and future directions. Data Min. Knowl. Disc., 15:55–86, 2007.
7. Han J. and Kamber M. Data Mining: Concepts and Techniques, 2nd edn. Morgan Kaufmann, 2006.
8. Hastie T., Tibshirani R., and Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer, 2001.
9. Liu B. Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data. Springer, 2006.
10. Mitchell T.M. Machine Learning. McGraw-Hill, 1997.
11. Tan P., Steinbach M., and Kumar V. Introduction to Data Mining. Addison Wesley, 2005.
12. Witten I.H. and Frank E. Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann, 2005.
13. Zhang T., Ramakrishnan R., and Livny M. BIRCH: an efficient data clustering method for very large databases. In Proc. ACM-SIGMOD Int. Conf. on Management of Data, 1996, pp. 103–114.

Data Mining in Bioinformatics

- ▶ Machine Learning in Computational Biology

Data Mining in Computational Biology

▶ Machine Learning in Computational Biology

Data Mining in Moving Objects Databases

▶ Spatio-Temporal Data Mining

Data Mining in Systems Biology

▶ Machine Learning in Computational Biology

Data Mining Pipeline

▶ KDD Pipeline

Data Mining Process

▶ KDD Pipeline

Data Model Mapping

▶ Logical Database Design: From Conceptual to Logical Schema

Data Organization

▶ Indexing and Similarity Search

Data Partitioning

Daniel Abadi
Yale University, New Haven, CT, USA

Definition

Data Partitioning is the technique of distributing data across multiple tables, disks, or sites in order to improve query processing performance or increase database manageability. Query processing performance can be improved in one of two ways. First, depending on how the data is partitioned, in some cases it can be determined *a priori* that a partition does not have to be accessed to process the query. Second, when data is partitioned across multiple disks or sites, *I/O parallelism* and in some cases *query parallelism* can be attained as different partitions can be accessed in parallel. Data partitioning improves database manageability by optionally allowing backup or recovery operations to be done on partition subsets rather than on the complete database, and can facilitate loading operations into rolling windows of historical data by allowing individual partitions to be added or dropped in a single operation, leaving other data untouched.

Key Points

There are two dominant approaches to data partitioning.

Horizontal partitioning divides a database table tuple-by-tuple, allocating different tuples to different partitions. This is typically done using one of five techniques:

1. *Hash partitioning* allocates tuples to partitions by applying a hash function to an attribute value (or multiple attribute values) within the tuple. Tuples with equivalent hash function values get allocated to the same partition.
2. *Range partitioning* allocates tuples to partitions by using ranges of attribute values as the partitioning criteria. For example, tuples from a customer table with last name attribute beginning with “A”–“C” are mapped to partition 1, “D”–“F” mapped to partition 2, etc.
3. *List partitioning* allocates tuples to partitions by associating a list of attribute values with each partition. Using range or list partitioning, it can be difficult to ensure that each partition contains approximately the same number of tuples.
4. *Round-robin partitioning* allocates the ith tuple from a table to the (i mod n)th partition where n is the total number of partitions.
5. *Composite partitioning* combines several of the above techniques, typically range partitioning followed by hash partitioning.

Vertical partitioning divides a table column-by-column, allocating different columns (or sets of columns) to different partitions. This approach is less frequently used relative to horizontal partitioning since it is harder to parallelize query processing over multiple vertical partitions, and merging or joining partitions is often necessary at query time. *Column-stores* are databases that specialize in vertical partitioning, usually taking the approach to the extreme, storing each column separately.

Cross-references

- ▶ Horizontally Partitioned Data
- ▶ Parallel Query Processing

Data Pedigree

▶ Data Provenance

Data Perturbation

▶ Matrix Masking

Data Privacy and Patient Consent

David Hansen[1], Christine M. O'Keefe[2]
[1]The Australian e-Health Research Centre, Brisbane, QLD, Australia
[2]CSIRO Preventative Health National Research Flagship, Acton, ACT, Australia

Synonyms

Data protection

Definition

Data privacy refers to the interest individuals and organisations have in the collection, sharing, use and disclosure of information about those individuals or organizations. Common information types raising data privacy issues include health (especially genetic), criminal justice, financial, and location. The recent rapid growth of electronic data archives and associated data technologies has increased the importance of data privacy issues, and has led to a growing body of legislation and codes of practice.

Patient consent, in relation to data, refers to a patient's act of approving the collection, sharing, use or disclosure of information about them. It is important because data with appropriate patient consent often falls into exception clauses in privacy legislation.

The challenge in data privacy is to balance the need to share and use data with the need to protect personally identifiable information and respect patient consent. For example, a person may have an expectation that their health data is being held securely but is available to clinicians involved in their care. In addition, researchers seek access to health data for medical research and to answer questions of clinical and policy relevance. Part of the challenge is that there are a range of views about, for example, what constitutes legitimate uses of data, what level of consent is required and how fine-grained that consent should be, what constitutes a clinical "care team," and what privacy legislation should cover.

The development of technologies to support data privacy and patient consent is currently attracting much attention internationally.

Historical Background

Privacy issues began to attract attention in Europe and North America in the 1960s, and shortly thereafter in Australia. Probably a large factor in the relatively late recognition of privacy as a fundamental right is that most modern invasions of privacy involve new technology. For example, before the invention of computer databases, data were stored on paper in filing cabinets which made it difficult to find and use the information. The traditional ways of addressing the harm caused by invasions of privacy through invoking trespass, assault or eavesdropping were no longer sufficient in dealing with invasions of privacy enacted with modern information technologies.

In the health care area, the notion of consent arose in the context of both treatment and clinical trials in research into new treatments. Patients undergoing a procedure or treatment would either be assumed to have given implicit consent by their cooperation, or would be required to sign an explicit statement of consent. Participants in clinical trials would be asked to give a formal consent to a trial, considered necessary because of the risk of harm to the individual due to the unknown effects of the intervention under research. The notion of consent has transferred to the context of the (primary) use of data for treatment and the (secondary) use of data for research. For example, implied or expressed consent would be required for the transfer of medical records to a specialist or may be required for the inclusion of data in a research database or register. Increasingly, and somewhat controversially, health data are being made available for research without patient consent, but under strict legal and ethical provisions. Where this is the case, ethics committees are given the responsibility to decide if the research benefits outweigh to a substantial degree the public interest in protecting privacy. Ethics committees will generally look for de-identified data to be used wherever practical.

A recent development related to the privacy of health data is the concept of participation or moral rights (http://www.privireal.org/). This can be viewed as an objection to the use of personal information on moral grounds; that individuals should have the right to know, and veto, how their data is used, even when there is no risk to the release of that personal information.

The major data privacy-related legislative provisions internationally are Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data, the Health Insurance Portability and Accountability Act (HIPAA) enacted by the US Congress in 1996, the Data Protection Act 1998 of the UK Parliament and the Australian Privacy Act 1988. There are apparent differences between these provisions in terms of both their scope and philosophy.

Recent technological approaches to data privacy include: tight governance and research approvals processes, restricted access through physical and IT security, access control mechanisms, de-identification, statistical disclosure limitation and remote server technology for remote access and remote execution. Patient consent is often built into the approvals process, but can also be a component of the access control mechanism. In all cases the purpose of access to the data is of prime importance.

Foundations

The use of sensitive personal data falls broadly into two areas: primary and secondary use.

Primary use of data refers to the primary reason the data was captured. Increasing amounts of personal health data are being stored in databases for the purpose of maintaining a life long health record of an individual. While in most cases this will provide for more appropriate care, there will be times when the individual will not want sensitive data revealed, even to a treating clinician. This may particularly be the case when the data is judged not relevant to the current medical condition.

In some electronic health record programs, patients will have the option to opt-in or opt-out of the program. This can be a controversial aspect of health record systems and often legislation is needed to support this aspect of any collection of health related data.

Once the patient data has been entered into an electronic health record system, there will generally be several layers of security protecting the data. Authentication is often performed using a two pass system – a password and a certificate are required to enter the system, to ensure that the person accessing the system is correctly authenticated. Once the person is authenticated access to appropriate data must be managed. Often a Role Based Access System (RBAC) [5] is implemented to ensure that access to the data is only granted to people who should have it. There are also now XML based mark up languages, such as XACML, which enable Role Based Access Control rules to be encoded using the language and hence shared by multiple systems. Audit of access to electronically stored health data is also important, to enable review access granted to data.

Storage and transmission of data is also an issue for privacy and confidentiality of patient data.

Encryption of the data when being transmitted is one way of ensuring the security of the data. There are also new computer systems which support Mandatory Access Control (MAC) which embed the security algorithms in the computer hardware, rather than at the software level, which are now available for the storage and access of electronic data. Security Assertion Markup Language (SAML) token are one technology which is being used to store privacy and security information with health data as they are transmitted between computers.

The USA HIPAA legislation covers the requirements of how to capture, store and transmit demographic and clinical data.

The secondary use of data is more problematic when it comes to data privacy and patient consent, since it refers to usage of the data for purposes which are not directly related to the purpose for which it was collected. This includes purposes like medical research and policy analysis, which are unlikely to have a direct affect on the treatment of the patient whose data is used.

There are a range of technological approaches to the problem of enabling the use of data for research and policy analysis while protecting privacy and confidentiality. None of these technologies provides a complete answer, for each must be implemented within an appropriate legislative and policy environment and governance structure, with appropriate management of the community of authorised users and with an appropriate level of IT security including user authentication, access control, system audit and follow-up. In addition, none of the technologies discussed here is the only solution to the problem, since there are many different scenarios for the use of data, each with a different set of requirements. It is clear that different technologies and approaches have different strengths and weaknesses, and so are suitable for different scenarios.

A high level discussion of the problem of enabling the use of data while protecting privacy and confidentiality typically discusses two broad approaches. The first is restricted access, where access is only provided to approved individuals and for approved purposes. Further restrictions can be imposed, such as access to data only given at a restricted data centre, restrictions on the types of analyses which can be conducted and restrictions on the types of outputs which can be taken out of a data centre. There can be a cost associated with access to the data. The second is restricted or altered data, where something less than the full data set is published or the data are altered in some way before publication. Restricted data might involve removing attributes, aggregating geographic classifications or aggregating small groups of data. For altered data, some technique is applied to the data so that the released dataset does not reveal private or confidential information. Common examples here include the addition of noise, data swapping or the release of synthetic data. Often these broad approaches are used in combination.

Below, three current technological approaches to the problem are reviewed. These fall into the category restricted or altered data described above, and all are used in combination with restricted access.

The first approach is to release de-identified data to researchers under strict controls. De-identification is a very complex issue surrounded by some lack of clarity and standard terminology. It is also very important as it underpins many health information privacy guidelines and legislation.

First, it is often not at all clear what is meant when the term “de-identified” is used to refer to data. Sometimes it appears to mean simply that nominated identifiers such as name, address, date of birth and unique identifying numbers have been removed from the data. At other times its use appears to imply that individuals represented in a data set cannot be identified from the data – though in turn it can be unclear what this means. Of course simply removing nominated identifiers is often insufficient to ensure that individuals represented in a data set cannot be identified – it can be a straightforward matter to match some of the available data fields with the corresponding fields from external data sets, and thereby obtain enough information to determine individuals’ names either uniquely or with a low uncertainty. In addition, sufficiently unusual records in a database without nominated identifiers can sometimes be recognized. This is particularly true of health information or of information which contains times and/or dates of events.

The second approach is statistical disclosure control where the techniques aim to provide researchers with useful statistical data at the same time as preserving privacy and confidentiality.

It is widely recognized that any release of data or statistical summaries increases the risk of identification of some individual in the relevant population, with the consequent risk of harm to that individual through inference of private information about them. On the other hand, attempts to limit such disclosures can adversely affect the outcomes or usefulness of statistical analyses conducted on the data. Statistical disclosure control theory attempts to find a balance between these opposing objectives.

Statistical disclosure control techniques can be organized into categories in several different ways. First, there are different techniques for tabular data (where data are aggregated into cells) versus microdata (individual level data). Second, techniques can be perturbative or non-perturbative. Perturbative methods operate by modifying the data, whereas non-perturbative methods do not modify the data. Perhaps the most well-known perturbative method is the addition of random “noise” to a dataset, and perhaps the most well-known non-perturbative method is cell suppression. In fact, current non-perturbative methods operate by suppressing or reducing the amount of information released, and there is much ongoing debate on whether a good perturbative method gives more useful information than a non-perturbative method. On the other hand, it has been noted that perturbative techniques which involve adding noise provide weak protection and are vulnerable to repeated queries, essentially because the noise becomes error in models of the data. There is much activity directed at developing perturbative techniques that do not suffer from this problem.

Virtually every statistical disclosure control technique can be implemented with differing degrees of intensity, and hence depends on a parameter which is usually pre-specified.

Remote analysis servers are designed to deliver useful results of user-specified statistical analyses with acceptably low risk of a breach of privacy and confidentiality.

The third approach is the technology of remote analysis servers. Such servers do not provide data to users, but rather allow statistical analysis to be carried out via a remote server. A user submits statistical queries by some means, analyses are carried out on the original data in a secure environment, and the user then receives the results of the analyses. In some cases the output is designed so that it does not reveal private information about the individuals in the database.

The approach has several advantages. First, no information is lost through confidentialization, and there is no need for special analysis techniques to deal with perturbed data. In many cases it is found to be easier to confidentialize the output of an analysis, in comparison to trying to confidentialize a dataset when it is not known which analyses will be performed.

However, analysis servers are not free from the risk of disclosure, especially in the face of multiple, interacting queries. They describe the risks and propose quantifiable measures of risk and data utility that can be used to specify which queries can be answered and with what output. The risk-utility framework is illustrated for regression models.

Each of the broad technologies is implemented within the context that the analyst is trusted to comply with legal and ethical undertakings made. However, the different approaches have been designed with different risks of disclosure of private information, and so rely more or less heavily on trust. De-identification requires the greatest trust in the researcher, while remote servers require the least. Statistical disclosure control, whether used alone or in combination with a remote analysis server, is somewhere inbetween these two extremes. De-identification provides the most detailed information to the researcher, while remote servers provide the least. Again, Statistical Disclosure Control is inbetween.

These mechanisms of maintaining data privacy are made more difficult when it is necessary to link two or more databases together for the purpose of building a linked data set for analysis. This sort of research is increasingly being used as a way of reducing the cost of collecting new data and to make better use of existing data. The difficulties are two fold. First, there is the linkage step, i.e., recognizing that patients are the same across the data sets. Recent work has included blindfolded linkage methodologies [1,2] and encryption techniques [4] as ways of linking datasets while not revealing any patient information. The second difficulty lies in the greater chance of revealing information from a linked data set than a single data set, especially when combining data from multiple modalities, e.g., health data and geographic data.

Key Applications

As discussed above, data privacy and patient consent has an impact on large sections of the health care industry and the biomedical research community. There are many applications which will need to consider data privacy and patient consent issues. Below is a discussion of Electronic Health Records, possibly the fastest growing example of data collected for primary use, and medical research, often the largest source of requests for data for secondary use.

Electronic Health Records

Electronic Health Records (EHR) are the fastest growing example of an application which concern data privacy and patient consent. Increasing amounts of personal health data are being stored in databases for the purpose of maintaining a life long health record of an individual.

The data is being stored according to a number of different international and local standards and development for the format of the data, for example openEHR (http://www.openehr.org/), and the transmission of the data, such as HL7 (http://www.hl7.org/). Some of this data is stored using codes from clinical terminologies, while some of it will be free text reports. This data are being stored so that the data are available to clinicians for the purpose of treating the individual.

While in most cases this will provide more appropriate care, there will be times when the individual will not want sensitive data revealed, even to a treating clinician. This may particularly be the case when the data is judged not relevant to the current medical condition. With a number of countries introducing Electronic Health Records, there are concerns over who will have access to the data. Generally these are governed by strict privacy policies, as well as allowing patients the opportunity to have some level of control over whether data is added to the EHR or not.

Medical Research

Secondary use of data is primarily concerned with providing health data for clinical or medical research. For most secondary data use, it is possible to use de-identified data, as described above. Increasingly, secondary use of data involves the linkage of data sets to bring different modalities of data together, which raises more concerns over the privacy of the data as described above. The publication of the Human Genome gave rise to new ways of finding relationships between clinical disease and human genetics. The increasing use and storage of genetic information also impacts the use of familial records, since the information about the patient also provides information on the patient's relatives. The issues of data privacy and patient confidentiality and the use of the data for medical research are made more difficult in this post-genomic age.

Cross-references

- ▶ Access Control
- ▶ Anonymity
- ▶ Electronic Health Record
- ▶ Exploratory Data Analysis
- ▶ Health Informatics
- ▶ Privacy Policies and Preferences
- ▶ Privacy-Enhancing Technologies
- ▶ Privacy-Preserving data mining
- ▶ Record Linkage

Recommended Reading

1. Agrawal R., Evfimievski A., and Srikant R. Information sharing across private databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 86–97.
2. Churches T. and Christen P. Some methods for blindfolded record linkage. BMC Med. Inform. Decis. Making, 4:9, 2004.
3. Domingo-Ferrer J. and Torra V. (eds.). Privacy in Statistical Databases. Lect. Notes Comput. Sci., Vol 3050. Springer Berlin Heidelberg, 2004.
4. OKeefe C.M., Yung M., and Baxter R. Privacy-preserving linkage and data extraction protocols. In Workshop on Privacy in the Electronic Society in conjunction with the 11th ACM CCS Conference, 2004.
5. Sandhu R.S., Coyne E.J., Feinstein H.L., and Youman C.E. Role-based access control models. IEEE Comput., 29(2):38–47, 1996.

Data Problems

▶ Data Conflicts

Data Profiling

Theodore Johnson
AT&T Labs – Research, Florham, Park, NJ, USA

Synonyms

Database profiling

Definition

Data profiling refers to the activity of creating small but informative summaries of a database [5]. These summaries range from simple statistics such as the number of records in a table and the number of distinct values of a field, to more complex statistics such as the distribution of n-grams in the field text, to structural properties such as keys and functional dependencies. Database profiles are useful for database exploration, detection of data quality problems [4], and for schema matching in data integration [5]. Database exploration helps a user identify important database properties, whether it is data of interest or data quality problems. Schema matching addresses the critical question, "do two fields or sets of fields or tables represent the same information?" Answers to these questions are very useful for designing data integration scripts.

Historical Background

Databases which support a complex organization tend to be quite complex also. Quite often, documentation and metadata are incomplete and outdated, no DBA understands the entire system, and the actual data fails to match documented or expected properties [2]. These problems greatly complicate already difficult tasks such as database migration and integration, and in fact database profiling was originally developed for their support. Newer developments in database profiling support database exploration, and finding and diagnosing data quality problems.

Foundations

A *database profile* is a collection of summaries about the contents of a database. These summaries are usually collected by making a scan of the database (some profiles use sampled data, and some require multiple complex queries). Many of the profile statistics are collected by the DBMS for query optimization. If the optimizer statistics are available, they can be retrieved instead of calculated – though one must ensure that the optimizer's statistics are current before using them. Profile summaries are typically stored in a database for fast interactive retrieval.

Basic Statistics

These statistics include schema information (table and field names, field types, etc.) and various types of counts, such as the number of records in a table, and the number of special values of a field (typically, the number of null values).

Distributional Statistics

These statistics summarize the *frequency distribution* of field values: for each distinct value of a field, how often does it occur. Examples include the number of distinct values of a field, the entropy of the frequency distribution [10], and the most common values and their frequency of occurrence. Another summary is the *inverse frequency distribution*, which is the distribution of the frequency distribution (e.g., the number of distinct values which occur once, occur twice, and so on). While the inverse frequency distribution is often small, it can become large and in general needs to be summarized also.

Textual Summaries

A textual summary represents the nature of the data in a field. These summaries are very useful for exploring the value and pattern distributions and for field content matching in a schema matching task, i.e., to determine whether or not two fields across tables or databases represent similar content.

Textual summaries apply to fields with numeric as well as string data types. The issue is that many identifiers are numeric, such as telephone numbers, Social Security numbers, IP addresses, and so on. These numeric identifiers might be stored as numeric or string fields in a given table – or even combined with other string fields. To ensure that field matches can be made in these cases, numeric fields should be converted to their string representation for textual matching. Patterns (say, regular expressions) which most field values conform to are very useful in identifying anomalies and data quality issues.

Minhash Signatures: One type of summary is very useful in this regard, the *minhash signature* [1]. To compute a minhash signature of a field, one starts with N hash functions from the field domain to the integers. For each hash function, compute the hash value of each field value, and collect the minimum. The collection of minimum hash values for each hash function constitutes the minhash signature.

For example, suppose our set consists of $X = \{3,7,13,15\}$ and our two hash functions are $h_1(x) = x \bmod 10$, and $h_2(x) = x \bmod 5$ (these are simple but very poor hash functions). Then, $min\{h_1(x) \mid x \text{ in } X\} = 3$, and $min\{h_2(x) \mid x \text{ in } X\} = 2$. Therefore the minhash signature of X is $\{3,2\}$.

A surprising property of the minhash signature is its ability to determine the intersection of two sets. Given two minhash signatures for sets A and B, the number of hash functions with the same minimum value divided by the number of hash functions is an estimator for the *resemblance* of two sets, which is the size of the intersection of A and B divided by the union of A and B ($\rho = |A \cap B|/|A \cup B|$). Given knowledge of $|A|$ and $|B|$ (from the distributional statistics), an estimate of the size of the intersection is $|A \cap B| = \rho(|A| + |B|)/(1 + \rho)$.

If one extends the minhash signature to include the number of times that the minimum value of a hash function occurred, one can summarize the tail of the inverse frequency distribution [3]. Augmenting the minhash signature summary with the counts of the most frequent values (from the distributional statistics), which constitute the head, completes the summary.

Substring summaries: Another type of textual summary determines if two fields have textually similar information, i.e., many common substrings. As with approximate string matching [6], these summaries rely on *q-grams* – all consecutive *q*-letter sequences in the field values. One type of approximate textual summary collects the distribution of all q-grams within a field's values, and summarizes this distribution using a *sketch* such as the minhash signature or the min-count sketch [3]. Two fields are estimated to be textually similar if their q-gram distributions, represented by the sketches, are similar.

Structural Summaries

Some summaries represent patterns among fields in a table. The two most common examples are *keys* and *functional dependencies* (FDs). Since a table might be corrupted by data quality problems, another type of structural summary are *approximate* keys and *approximate FDs* [7], which hold for most (e.g., 98%) of the records in a table.

Key and FD finding is a very expensive procedure. Verifying whether or not a set X of fields is a key can be performed using a count distinct query on X, which returns the number of distinct values of X. Now, X is an (approximate) key if the number of distinct values of X is (approximately) equal to the size of the table. And, an FD X → Y (approximately) holds if the number of distinct values of X is (approximately) equal to the number of distinct values of (X U Y). An exhaustive search of a *d* field table requires 2^d expensive count-distinct operations on the table. There are several ways to reduce this cost. For one, keys and FDs with a small number of fields are more interesting than ones with a large number of fields (large keys and FDs are likely spurious – because of the limited size of the table – and likely do not indicate structural properties of the data). If the maximum set of fields is k (e.g., $k = 3$), then the search space is limited to $O(d^k)$. The search space can be trimmed further by searching for *minimal* keys and functional dependencies [7]. Finally, one can hash string fields to reduce the cost of computing count distinct queries over them (if exact keys and FD are required, a candidate key or FD found using hashing must be verified by a query over the actual table).

Samples

A random sample of table records also serves as summary of the contents of a table. A few sampled rows of a table are surprisingly informative, and can be used to estimate a variety of distributions, e.g., identifying the most frequent values or patterns and their frequencies in a field.

Sampling can be used to accelerate the expensive computation of profile data. For example, when computing keys and FDs on a large table (e.g., one with many records), one can sample the table and compute keys and FDs over the sample. If a key or FD is (approximately) valid over the base table then it is also valid on a sample. But, it is possible that a key or FD which is valid on the sample may not be a valid on the base table. Therefore, a random sample can be used to identify candidate keys and FDs. If exact keys and FDs are needed, candidates can be verified by queries over the actual table.

A minhash signature can be computed over sampled data. Suppose that F and G are again keys with identical sets of strings and of size *S*, and that one computes minhash signatures over F' and G', which are sampled from F and G (respectively) at rate *p*. Then, the resemblance of F' and G' is

$$\begin{aligned}\rho' &= |\mathrm{F}' \cap \mathrm{G}'|/(|\mathrm{F}'| + |\mathrm{G}'| - |\mathrm{F}' \cap \mathrm{G}'|)\\ &= p^2S/(2pS - p^2S) = p/(2-p) \approx p/2\end{aligned}$$

While the resemblance decreases linearly with *p* (and experiences a larger decrease than the sample intersection), minhash signatures have the advantage of being small. A profiling system which collects minhash signatures might use a signature size of, e.g., 250 hashes – small enough that an exhaustive search for matching fields can be performed in real time. Very large tables are sampled to accelerate the computation of the minhash signatures, say $p = .1$. When comparing two fields which both represent identical key values, there will be about 13 matches on average – enough to provide a reliable signal. In contrast, small uniform random samples of similar sizes drawn from the two fields may not provide accurate estimates of the resemblance. For instance, collecting 250 samples from a table with 1,000,000 rows requires a sampling rate of $p = 0.00025$, meaning that the intersection of the samples of F and G is very likely to be empty.

While random sampling is a common data synopsis used to estimate a wide variety of data properties, its use as a database profile is limited. For one, a random sample cannot always provide an accurate estimation of the number of distinct values in a field, or of the frequency distribution of a field. Table samples are also ineffective for computing the size of the intersection of fields. Suppose that fields F and G are keys and contain the same set of strings, and suppose that they are sampled at rate *p*. Then, the size of the intersection of the sample is $p^2|F| = p^2|G|$. One can detect that F and G are identical if the size of the intersected sample is p times the size of the samples. However, if *p* is small enough for exhaustive matching (e.g., $p = 0.001$), then $p^2|F|$ is likely to be very small – and therefore an unreliable indicator of a field match.

Implementation Considerations

Profiling a very large database can be a time-consuming and computationally intensive procedure. A given DBMS might have features, such as sampling, grouping sets, stored procedures, user-defined aggregate functions, etc., which can accelerate the computation of various summaries. Many profile statistics are computed by the DBMS for the query optimizer, and might be made available to users.

However, database profiles are often used to compare data from different databases. Each of these databases likely belongs to its own administrative domain, which will enable or disable features depending on the DBA's needs and preferences. Different databases often reside in different DBMSs. Therefore, a profiling tool which enables cross-database comparisons must in general make use of generic DBMS facilities, making use of DBMS-specific features as an optimization only.

Modes of Use

The types of activities supported by database profiles can be roughly categorized into *database exploration* and *schema matching*. Database exploration means to help a user identify important database properties, whether it is data of interest, data quality problems, or properties that can be exploited to optimize database performance. For example, the user might want to know which are the important tables in a database, and how do they relate (how can they be joined). The number of records in a table is a good first indicator of the importance of a table, and a sample of records is a good first indicator of the kind of data in the table. The most frequent values of a field will often indicate the field's default values (often there is more than one). Other types of information, e.g., keys and field resemblance, help to identify join paths and intra-table relationships.

By collecting a sequence of historical snapshots of database profiles, one can extract information about how the database is updated. A comparison of profiles can indicate which tables are updated, which fields tend to change values, and even reveal changes in database maintenance procedures [3]. For example, in the two large production databases studied in [3], only 20–40% of the tables in the database changed at all from week to week. Furthermore, most of the tables which ever changed experienced only a small change. Only 13 of the 800 + tables were found to be dynamic.

A schema matching activity asks the question, "do these two instances represent the same thing?" – fields, sets of fields, tables, etc. For example, textual summaries are designed to help determine if two fields have the same (or nearly the same) contents. However, any single type of information (schema, textual, distributional) can fail or give misleading results in a large number of cases. The best approach is to use all available information [11].

Key Applications

Data profiling techniques and tools have been developed for database exploration, data quality exploration, database migration, and schema matching. Systems and products include Bellman [4], Ascential [8] and Informatica [9].

Cross-references

- ▶ Count-Min Sketch
- ▶ Data Sketch/Synopsis
- ▶ Hash Functions

Recommended Reading

1. Broder A. On the resemblance and containment of documents. In Proc. IEEE Conf. on Compression and Comparison of Sequences, 1997, pp. 21–29.
2. Dasu T. and Johnson T. Exploratory Data Mining and Data Cleaning. Wiley Interscience, New York, 2003.
3. Dasu T., Johnson T., and Marathe A. Database exploration using database dynamics. IEEE Data Eng. Bull. 29(2):43–59, 2006.
4. Dasu T., Johnson T., Muthukrishnan S., and Shkapenyuk V. Mining database structure; or, how to build a data quality browser. In Proc. ACM SIGMOD Int. Conf. on Management of data, 2002, pp. 240–251.
5. Evoke Software. Data Profiling and Mapping, The Essential First Step in Data Migration and Integration Projects. Available at: http://www.evokesoftware.com/pdf/wtpprDPM.pdf 2000.
6. Gravano L., Ipeirotis P.G., Jagadish H.V., Koudas N., Muthukrishnan S., and Srivastava D. Approximate String Joins in a Database (Almost) for Free. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 491–500.
7. Huhtala Y., Karkkainen J., Porkka P., and Toivonen H. TANE: an efficient algorithm for discovering functional and approximate dependencies. Comp. J., 42(2):100–111, 1999.
8. IBM Websphere Information Integration. Available at: http://ibm.ascential.com
9. Informatica Data Explorer. Available at: http://www.informatica.com/products_services/data_explorer
10. Kang J. and Naughton J.F. On schema matching with opaque column names and data values. In Proc. ACM SIGMOD Int.

Conf. on Management of Data, San Diego, CA, 2003, pp. 205–216.

11. Shen W., DeRose P., Vu L., Doan A.H., and Ramakrishnan R. Source-aware entity matching: a compositional approach. In Proc. 23rd Int. Conf. on Data Engineering, pp. 196–205.

Data Protection

► Data Privacy and Patient Consent
► Storage Protection

Data Provenance

Amarnath Gupta
University of California San Diego, La Jolla, CA, USA

Synonyms

Provenance metadata; Data lineage; Data tracking; Data pedigree

Definition

The term "data provenance" refers to a record trail that accounts for the origin of a piece of data (in a database, document or repository) together with an explanation of how and why it got to the present place.

Example: In an application like Molecular Biology, a lot of data is derived from public databases, which in turn might be derived from papers but after some transformations (only the most significant data were put in the public database), which are derived from experimental observations. A provenance record will keep this history for each piece of data.

Key Points

Databases today do not have a good way of managing provenance data and the subject is an active research area. One category of provenance research focuses on the case where one database derives some of its data by querying another database, and one may try to "invert" the query to determine which input data elements contribute to *this data element*. A different approach is to explicitly add annotations to data elements to capture the provenance. A related issue is to keep process provenance, especially in business applications, where an instrumented business process capturing software is used to track the data generation and transformation life cycle. While keeping a trail of provenance data is beneficial for many applications, storing, managing and searching provenance data introduces an overhead.

Cross-references

► Annotation
► Provenance

Recommended Reading

1. Bose R. and Frew J. Lineage retrieval for scientific data processing: a survey. ACM Comput. Surv., 37(1):1–28, 2005.
2. Buneman P., Khanna S., Tajima K., and Tan W.-C. Archiving scientific data. In Proc. ACM SIGMOD Conf. on Management of Data, 2002, pp. 1–12.
3. Buneman P., Khanna S., and Tan W.C. On propagation of deletions and annotations through views. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 150–158.
4. Simmhan Y.L., Plale B., and Gannon D. A Survey of Data Provenance Techniques. Technical Report TR618, Department of Computer Science, Indiana University, 2005.
5. Widom J. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. In Proc. 2nd Biennial Conference on Innovative Data Systems Research, 2005, pp. 262–276.

Data Quality

► Information Quality and Decision Making
► Information Quality Policy and Strategy

Data Quality Assessment

Carlo Batini
University of Milano – Bicocca, Milan, Italy

Synonyms

Data quality measurement; Data quality benchmarking

Definition

The goal of the assessment activity in the area of data quality methodologies is to provide a precise evaluation and diagnosis of the state of databases and data flows of an information system with regard to data quality issues. In the assessment the evaluation is performed measuring the quality of data collections along relevant quality dimensions. The term (data quality) *measurement* is used to address the issue of measuring the value of a set of data quality dimensions. The term (data quality) *assessment* is used when such measurements are analyzed in order to enable a diagnosis of the quality of the data collection. The term (data quality) *benchmarking* is used when the output of the assessment is compared against reference indices, representing average values or best practices values in similar organizations. The term (data quality) *readiness* aims at assessing the overall predisposition of the organization in accepting and taking advantages of data quality improvement programs.

The assessment activity may concern: (i) the schema of the data base (the intension), (ii) the values of data (the extension), and (iii) the costs of poor data quality to the organization. Therefore, the principal outputs of assessment methodologies are: (i) measurements of the quality of databases and data flows, both schemas and values, (ii) costs to the organization due to the present low data quality, and (iii) a comparison with data quality levels considered acceptable from experience, or else a benchmarking with best practices, together with suggestions for improvements.

Historical Background

Ever since computer applications have been used to automate more and more business and administrative activities, it has become clear that available data often result from inaccurate observations, imputation, and elaborations, resulting in data quality problems. More importantly, in the last decades, information systems have been migrating from a hierarchical/monolithic to a network-based structure; therefore, the potential sources that organizations can use for the purpose of their businesses are dramatically increased in size and scope. Data quality problems have been further worsened by this evolution, since the external sources are created and updated at different times and by different organizations or persons and are characterized by various degrees of trustworthiness and accuracy, frequently unknown a priori. As a consequence, the overall quality of the data that flow between information systems may rapidly degrade over time if both processes and their inputs are not themselves subject to quality assessment.

Foundations

The typical steps to assess data quality are:

1. *Data analysis*, which examines data schemas and performs interviews to reach a complete understanding of data and related architecture and management rules.
2. *Requirements analysis*, which surveys the opinion of data users and administrators to identify quality issues and set new quality targets.
3. *Identification of critical areas*, which selects the most relevant databases and data flows to be assessed quantitatively.
4. *Process modeling*, which provides a model of the processes producing or updating data.
5. *Measurement of quality*, which selects relevant quality dimensions, defines corresponding metrics and performs the actual measurement.

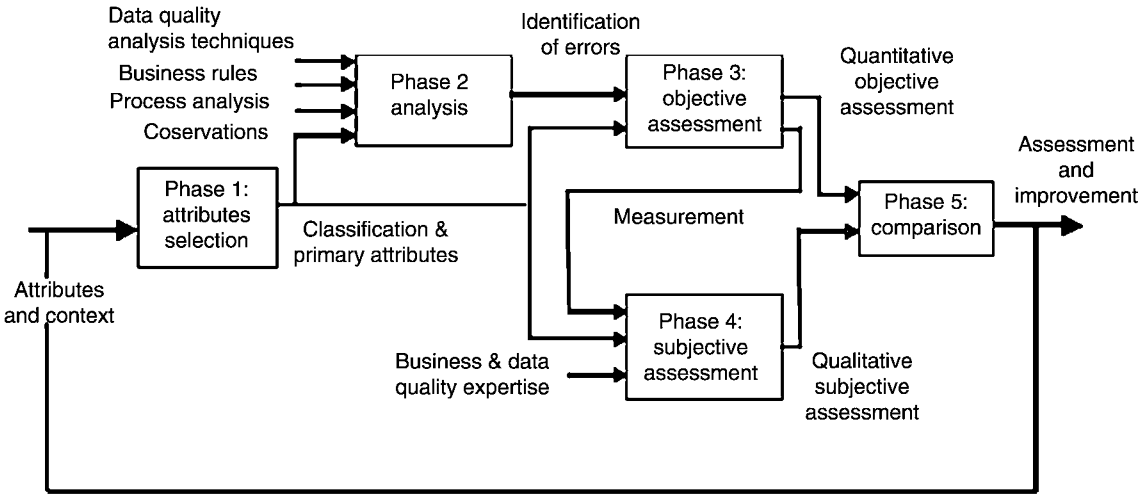
The usual process followed in the *measurement of quality* step has two main activities: qualitative assessment, based on subjective judgments of experts, and objective assessment, based on measures of data quality dimensions.

Qualitative assessment is performed through questionnaires and interviews with stakeholders and with internal and external users, with the goal of understanding the consequences and impact of poor data quality on the work of internal users and on products and services provided to information consumers, and the extent the needs of external users and customers are currently satisfied.

Quantitative assessment is based on the selection of quality dimensions and their measurement through metrics. Over 50 quality dimensions have been proposed in the literature (see the respective entry and [2] for a thorough description of dimensions and proposed classifications). The most frequently mentioned concern the values of data, and are accuracy, completeness, currency/timeliness, and inconsistency.

Examples of methodologies for the choice of dimensions and measures and for the objective vs. subjective evaluation are given in [3,7–9]. With regard to dimension classification, dimensions are classified in [7] into *sound*, *useful*, *dependable*, and *usable*, according to their positioning in quadrants related to "product quality/service quality" and "conforms to specifications/meets or exceeds consumer expectations" coordinates. The goal of the classification is to provide a context for each individual quality dimension and metric, and for consequent evaluation. In the following the five phases of the methodology proposed in [3] are described in more detail (see Fig. 1).

Phase 1, *attribute selection*, concerns the identification, description, and classification of the main data attributes to be assessed. Then, they are characterized


Data quality analysis techniques
Business rules
Process analysis
Coservations
Phase 2 analysis
Identification of errors
Phase 3: objective assessment
Quantitative objective assessment
Assessment and improvement
Phase 1: attributes selection
Classification & primary attributes
Measurement
Phase 5: comparison
Attributes and context
Phase 4: subjective assessment
Business & data quality expertise
Qualitative subjective assessment

Data Quality Assessment. Figure 1. The main phases of the assessment methodology described in [2].

according to their meaning and role. The possible characterizations are *qualitative/categorical*, *quantitative/numerical*, and *date/time*.

In Phase 2, *analysis*, data quality dimensions and integrity constraints to be measured are identified. Statistical techniques are used for the inspection of data. Selection and inspection of dimensions is related to process analysis, and has the final goal of discovering the main causes of erroneous data, such as unstructured and uncontrolled data loading and data updating processes. The result of the analysis on selected dimensions leads to a report with the identification of the errors.

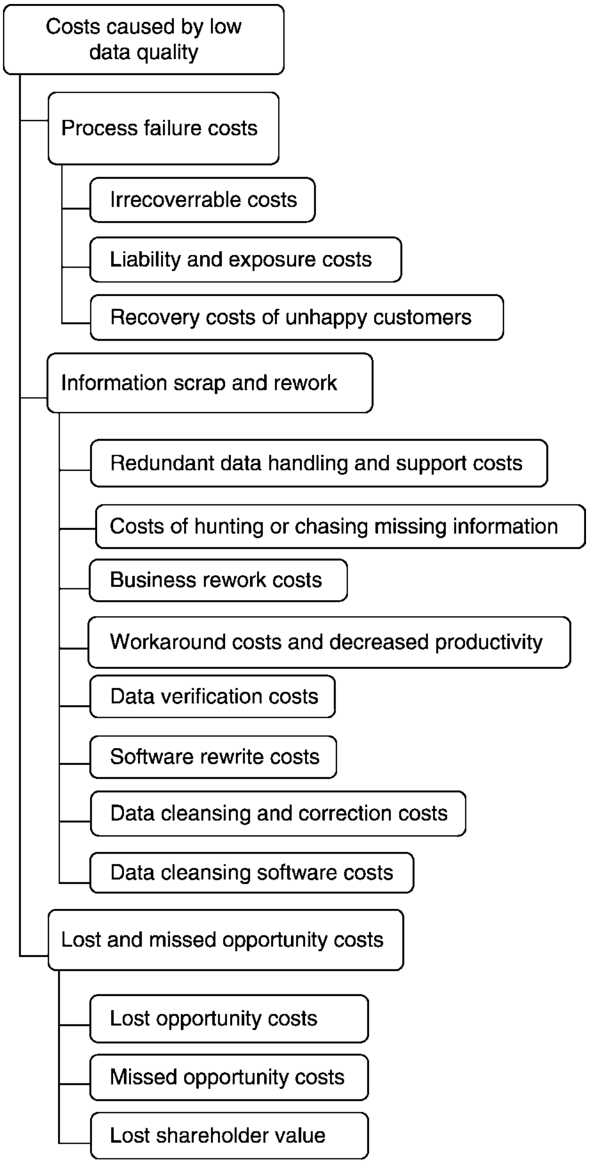
In Phase 3, *objective/quantitative assessment*, appropriate indices are defined for the evaluation and quantification of the global data quality level. The number of low quality data items for the different dimensions and the different data attributes is first evaluated with statistical and/or empirical methods, and, subsequently, normalized and summarized.

Phase 4 deals with *subjective/qualitative assessment*. The qualitative assessment is obtained by merging independent evaluations from (i) business experts, who analyze data from a business process point of view, (ii) final users (e.g., for financial data, a trader), and (iii) data quality experts, who have the role of analyzing data and examining its quality.

Finally, in the *comparison* phase objective and subjective assessments are compared. For each attribute and quality dimension, the distance between the percentages of erroneous observations obtained from quantitative analysis, mapped in a discrete domain, and the quality level defined by the judgment of the evaluations is calculated. Discrepancies are analyzed by the data quality experts, to further detect causes of errors and to find alternative solutions to correct them.

The above mentioned methodologies, although not explicitly, refer to the assessment of structured data, namely, data represented in terms of typed files or relational tables and databases. Recently, the attention in data quality assessment has moved towards semi-structured and un-structured data. Assessment methodologies for evaluating specific qualities of web sites are proposed in [1,6]. Atzeni et al. [1] is specifically focused on *accessibility*, evaluated on the basis of a mixed quantitative/qualitative assessment. The quantitative assessment activity checks the guidelines provided by the World Wide Web Consortium in (W3C. http://www.w3.org/WAI/). The qualitative assessment is based on experiments performed with disabled users. Fraternali et al. [6] focuses on the *usability* of the site and proposes an approach based on the adoption of *conceptual logs*, which are web usage logs enriched with meta-data derived from the application of conceptual specifications expressed by the conceptual schema of the web site.

Since data and information are often the most relevant resource consumed in administrative and business processes, several authors consider the evaluation of costs of poor data quality as part of the data quality assessment problem. Figure 2 shows the classification proposed in [4], for which comments follow:


Costs caused by low data quality
Process failure costs
Irrecoverrable costs
Liability and exposure costs
Recovery costs of unhappy customers
Information scrap and rework
Redundant data handling and support costs
Costs of hunting or chasing missing information
Business rework costs
Workaround costs and decreased productivity
Data verification costs
Software rewrite costs
Data cleansing and correction costs
Data cleansing software costs
Lost and missed opportunity costs
Lost opportunity costs
Missed opportunity costs
Lost shareholder value

Data Quality Assessment. Figure 2. A comprehensive classification of costs of poor data quality [4].

- *Process failure costs* result when poor quality information causes a process to not perform properly. As an example, inaccurate mailing addresses cause correspondence to be misdelivered.
- *Information scrap and rework* costs occur every time data of poor quality requires several types of defect management activities, such as reworking, cleaning, or rejecting. Examples of this category are (i) redundant data handling, if the poor quality of a source makes it useless, time and money has to be spent to collect and maintain data in another database, (ii) business rework costs, due to re-performing failed processes, such as resending correspondence, (iii) data verification costs, e.g., when data users do not trust the data, they perform their own quality inspection.
- *Loss and missed opportunity costs* correspond to the revenues and products not realized because of poor information quality. For example, due to low accuracy of customer e-mail addresses, a percentage of customers already acquired cannot be reached in periodic advertising campaigns, resulting in lower revenues, roughly proportional to the decrease of accuracy in addresses.

Data quality assessment has been investigated also under a managerial perspective. Following the results of the assessment, a managerial activity might be the analysis of the main barriers in the organization to the quality management perspective in terms of resistance to change processes, control establishment, information sharing, and quality certification.

Key Applications

Quality assessment is used in a large set of business and administrative activities, such as organization assessment, strategic planning, supply chain, marketing, selling, demographic studies, health experiments, management of health files, census applications, epidemiological analyses. The perception of the importance of quality assessment is increasing in the area of risk management, such as operational risk management related to the Basel II norms.

Future Directions

Open areas of research in data quality assessment concern quality dimensions and the relationship between data quality assessment and process quality assessment.

The first area concerns assessment of a wider set of dimensions, such as performance, availability, security, with concern also to risk management, and investigation on dependencies among dimensions. For example, a dependency among currency and accuracy is the rule “70% of all outdated data is also inaccurate.” Knowledge about dependencies can greatly aid in finding causes of low data quality, and in conceiving improvement activities.

The relationship between data quality and process quality is a wide area of investigation, due to the relevance and diversity of characteristics of business processes in organizations. The different impacts of data quality at the three typical organizational levels, namely operations, the tactical level, and the strategic level, are analyzed in [10] reporting interviews and the outcomes of several proprietary studies. Data quality and its relationship with the quality of services, products, business operations, and consumer behavior is investigated in very general terms in [9,11]. The symmetric problem of investigating how to improve information production processes positively influences data quality is analyzed in [10].

Cross-references

▶ Data Quality Dimensions
▶ Design for Data Quality
▶ Information Quality Assessment
▶ Information Quality Policy and Strategy
▶ Quality of Data Warehouses

Recommended Reading

1. Atzeni P., Merialdo P., and Sindoni G. Web site evaluation: methodology and case study. In Proc. Int. Workshop on Data Semantics in Web Information Systems, 2001.
2. Batini C. and Scannapieco M. Data Quality: Concepts, Methodologies and Techniques. Springer, 2006.
3. De Amicis F. and Batini C. A methodology for data quality assessment on financial data. Stud. Commn. Sci., 4(2):115–136, 2004.
4. English L.P. Improving Data Warehouse and Business Information Quality. Wiley, 1999.
5. English L.P. Process management and information quality: how improving information production processes improves information (product) quality. In Proc. 7th Int. Conf. on Information Quality, 2002, pp. 206–209.
6. Fraternali P., Lanzi P.L., Matera M., and Maurino A. Model-driven web usage analysis for the evaluation of web application quality. J. Web Eng., 3(2):124–152, 2004.
7. Kahn B., Strong D.M., and Wang R.Y. Information quality benchmarks: product and service performance. Commun. ACM, 45(4):184–192, 2002.
8. Lee Y.W., Strong D.M., Kahn B.K., and Wang R.Y. AIMQ: a methodology for information quality assessment. Inf. Manag., 40(2):133–146, 2001.
9. Pipino L., Lee Y.W., and Wang R.Y. Data quality assessment. Commun. ACM, 45(4):211–218, 2002.
10. Redman T.C. The impact of poor data quality on the typical enterprise. Commun. ACM, 41(2):70–82, 1998.
11. Sheng Y.H. Exploring the mediating and moderating effects of information quality on firms? Endeavor on information systems. In Proc. 8th Int. Conf. on Information Quality, 2003, pp. 344–353.

Data Quality Attributes

▶ Data Quality Dimensions

Data Quality Benchmarking

▶ Data Quality Assessment

Data Quality Criteria

▶ Data Quality Dimensions

Data Quality Dimensions

Kai-Uwe Sattler
Technical University of Ilmenau, llmenau, Germany

Synonyms

Data quality criteria; Data quality attributes; Data quality measurement

Definition

Data quality (DQ) is usually understood as a multidimensional concept. The dimensions represent the views, criteria, or measurement attributes for data quality problems that can be assessed, interpreted, and possibly improved individually. By assigning scores to these dimensions, the overall data quality can be determined as an aggregated value of individual dimensions relevant in the given application context.

Historical Background

Since the mid-1990s data quality issues have been addressed by systematic research studies. In this context, relevant dimensions of data quality have also been investigated. One of the first empirical studies by Wang and Strong [6] has identified 15 relevant dimensions out of 179 gathered criteria. This list was later supplemented by other researchers. Initially, there were proposed divergent definitions of the same dimensions, mostly due to different views, e.g., management perspectives versus data-oriented perspectives as well as application-specific views. In addition, several classifications for data quality problems and criteria were proposed.

To date there is still a different understanding of several dimensions, depending on the application scenario and its requirements. However, there exists a set of agreed upon dimensions that are relevant in most domains.

Foundations

The selection of dimensions relevant in a given scenario is mostly application-dependent. In addition, many dimensions are not independent and, therefore, should not be used together. However, because quality dimensions characterize potential data quality problems they can be classified according some important characteristics. In the following, some representative classifications are introduced followed by a discussion of the most important dimensions:

Classifications

A first approach for classifying DQ dimensions proposed by Redman [5] is based on DQ problems or conflicts by considering the different levels where they can occur:

- The *intensional level* comprises criteria concerning the content of the conceptual schema relevance, clarity of definition, the scope, the level of detail (e.g., granularity of attributes, the precision of the attribute domains) as well as consistency and flexibility.
- The *extensional level* considers the data values comprising criteria such as accuracy and correctness of values, timeliness, and completeness of data.
- The *level of data representation* addresses problems related to the data format, e.g., interpretability, portability, adequateness.

In contrast to this data-oriented approach, the classification introduced by Naumann [4] is more comprehensive. Dimensions are classified into four sets:

1. *Content-related* dimensions consider the actual data and therefore data-intrinsic properties such as accuracy, completeness, and relevance.
2. *Technical* dimensions address aspects of the hard- and software used for maintaining the data. Examples are availability, latency, response time, but also price.
3. *Intellectual* dimensions represent subjective aspects, such as trustworthiness or reputation.
4. *Instantiation-related* dimensions concern the presentation of data, e.g., the amount of data, understandability, and verifiability.

An alternative way of classifying DQ dimensions is to look at the process of data evolution by analogy of data with products. In [3] an approach is presented promoting hierarchical views on data quality following the steps of the data life cycle: collection, organization, presentation, and application. Based on an analysis of possible root causes for poor quality relevant dimensions can be identified and assigned to the different DQ views:

- *Collection quality* refers to problems during data capturing, such as observation biases or measurement errors. The relevant dimensions are, among others, accuracy, completeness, and trustworthiness of the collector.
- *Organization quality* deals with problems of data preparation and manipulation for storing it in a database. It comprises dimensions such as consistency, storage, and retrieval efficiency. Furthermore, collection quality is also a component of organization quality.
- *Presentation quality* addresses problems during processing, re-interpretation, and presentation of data. Dimensions are for example interpretability, formality as well as the organization quality component.
- *Application quality* concerns technical and social constraints preventing an efficient utilization of data and comprises dimensions like timeliness, privacy, and relevance in addition to the presentation quality component.

Among all these dimensions the most important ones in many application scenarios are completeness, accuracy, consistency, and timeliness that are now described in detail.

Completeness

Missing or incomplete data is one of the most important data quality problem in many applications. However, there are different meanings of completeness.

An obvious and often used definition is the absence of null values or more exactly the ratio of non-null values and the total number of values. This measure can be easily assessed. Given a relation $R(A_1,...,A_n)$ then N_{A_i} denotes the set of all non-null values in A_i:

$$N_{A_i} = \{t \in R | NotNull(t.A_i)\}$$

Completeness $Q_C(A_i)$ can be now defined as:

$$Q_C(A_i) = \frac{|N_{A_i}|}{|R|}$$

This can be also extended to take tuples into account instead of single values by determining the number of tuples containing no null values:

$$Q_C(R) = \frac{|N_{A_1,\ldots,A_n}|}{|R|}$$

Note that null can have different meanings which have to be treated in a special way: it could represent a missing value or simply a not-applicable case, e.g., a customer without a special delivery address.

Sometimes, not all attributes are of equal importance, e.g., whereas a customer identifier is always required, the customer's email address is optional. In this case, weights can be assigned to the individual attributes or rules of the form "if A_1 is not available (`null`) then A_2 is important, otherwise not" are used.

This notion of completeness concerns only the data inside the database. An alternative definition for completeness is the portion of real-world objects stored in the database. It addresses the case that for instance not all customers are represented in the database and therefore the data is incomplete. This is also known as *coverage*. However, assessing this completeness is often much more difficult because it requires either additional metadata (e.g., it is known that the DBLP digital library contains only computer science literature) or a (manual) checking with the real world, possibly supported by sampling.

Besides these extensional views, completeness can be also interpreted from an intensional point of view. Here, completeness (or *density*) is defined as the number of attributes represented in the database compared to the required real-world properties. Again, assessing this kind of completeness requires manual inspection.

Improvement of completeness is generally achieved by choosing better or additional data sources. In some cases, null values can be replaced with the help of dictionaries or reference sources (e.g., an address database). Depending of the usage of data missing numeric values can be sometimes also imputed based on knowledge about data characteristics, such as value distribution and variance.

Accuracy

A second data quality problem is often caused by measurement errors, observation biases or simply improper representation. Accuracy can be defined as the extent to which data are correct, reliable, and certified free of error. Note that the meaning of correctness is application-dependent: it can specify the distance to the actual real-world value or just the optimal degree of detail of an attribute value. Assuming a table representing sales volumes for products a value $10,000 could be interpreted as inaccurate if the actual value, e.g., obtained in a different way, is $10,500. However, if the user is interested only in some sales categories (low: $\leq$ 20K, high: $>$ 20K) the value is accurate.

In order to assess accuracy for a given value v the real world value $\overline{v}$ or at least a reference value is needed. Then, the distance can be easily computed for numeric values as $|v - \overline{v}|$ or – for textual values – as the syntactic distance using the edit distance measure. However, particularly for textual attributes sometimes the semantic distance has to be considered, e.g., the strings "Munich" and "München" are syntactically different but represent the same city. Solving this problem requires typically a dictionary or ontology.

Based on the distance of single attribute values, the accuracy of tuples or the whole relation can be computed as shown above for completeness, for example by determining the fraction of tuples with only correct values.

An improvement of accuracy is often possible only by removing inexact values or preferably by applying data cleaning techniques.

Consistency

Though modern database systems provide advanced support for ensuring integrity and consistency of data, there are many reasons why inconsistency is a further important data quality problem. Thus, consistency as a DQ dimension is defined as the degree to which data managed in a system satisfy specified constraints or business rules. Such rules can be classic database integrity constraints, such as uniqueness of customer identifiers or referential integrity (e.g., "for each order, a customer record must exist,") or more advanced business rules describing relationships between attributes (for instance "age = current-date − data-of-birth," "driver license number can only exist, if age $\geq$ 16.") These rules have to be specified by the user or can be derived automatically from training data by applying rule induction. Using a set **B** of such rules, the set of tuples from a relation R satisfying these rules can be determined:

$$W_{\mathbf{B}} = \{t \in R | \mathit{Satisfies}(t, \mathbf{B})\}$$

Then, the consistency measure for relation R can be computed as the fraction of tuples in $W_{\mathbf{B}}$ as shown above.

As for accuracy, an improvement of consistency can be achieved by removing or replacing inconsistent data.

Timeliness

Another reason for poor quality of data is outdated data. This problem is captured by the dimension timeliness describing the degree to which the provided data is up-to-date. Depending on the application this is not always the same as the ordinary age (time between creation of data and now). For instance, in a stock information system stock quotes data from 2000 are outdated if the user is interested in the current quotes. But if he asks for stock quotes from the time of the dot-com bubble, it would be still sufficient.

Therefore, both the age $age(v)$ of a value v (as the time between observation and now) and the update frequency $f_u(v)$ (updates per time unit) have to be considered, where $f_u(v) = 0$ means the value is never updated. Using this information, timeliness $Q_T(v)$ of v can be computed as

$$Q_T(v) = \frac{1}{f_u(v) \cdot age(v) + 1}$$

This takes into account that an object with a higher update frequency ages faster and that objects that are never updated have the same timeliness.

Further Dimensions

Finally, the following further dimensions are also important for many applications.

Relevance, also known from information retrieval, is the degree to which the provided information satisfies the users need. The problem of relevance occurs mainly if keyword-based search is used for querying data or documents. In database systems using exact queries, relevance is inherently high.

Response time measures the delay between the submission of a request (e.g., a query) and the arrival of the complete response. Though a technical criterion, response time is particularly important for users, because they usually do not want to wait more than a couple of seconds for an answer. Related to response time is *latency* defining the delay to the arrival of the first result data. Often, a small latency compensates a larger response time in user satisfaction.

Believability, trustworthiness, and reputation are dimensions which often depend on each other. Believability and trustworthiness can be understood as the degree to which data is accepted by the user as correct, accurate or complete. In contrast, reputation describes the degree to which a data (source) has a good standing by users. Reputation is based on the memory and summary of behavior from past transactions, whereas believability is more an subjective expectation.

Key Applications

DQ dimensions are primarily used for quality assessment. They define the criteria under which data quality is measured and for which quality scores can be derived. A further application are data quality models for explicitly representing data quality scores that can be used for annotating the data.

Cross-references

- ▶ Data Conflicts
- ▶ Data Quality Assessment
- ▶ Data Quality Models

Recommended Reading

1. Batini C. and Scannapieco M. Data Quality – Concepts, Methodologies and Techniques. Springer, 2006.
2. Gertz M., Özsu M.T., Saake G., and Sattler K. Report on the Dagstuhl Seminar: data quality on the Web. ACM SIGMOD Rec., 33(1):127–132, 2004.
3. Liu L. and Chi L. Evolutional data quality: a theory-specific view. In Proc. 7th Int. Conf. on Information Quality, 2002, pp. 292–304.
4. Naumann F. Quality-Driven Query Answering for Integrated Information Systems. LNCS 2261, Springer, Berlin, 2002.
5. Redman T. Data Quality for the Information Age. Artech House, Norwood, MA, USA, 1996.
6. Wang R. and Strong D. Beyond Accuracy: What Data Quality Means to Data Consumers. J. Inf. Syst., 12(4):5–34, 1996.
7. Wang R., Ziad M., and Lee Y. Data Quality. Kluwer, Boston, MA, USA, 2001.

Data Quality Measurement

- ▶ Data Quality Dimensions
- ▶ Data Quality Assessment

Data Quality Models

Monica Scannapieco
University of Rome, Rome, Italy

Synonyms

Data quality representations

Definition

Data quality models extend traditional models for databases for the purpose of representing data quality dimensions and the association of such dimensions to data. Therefore, data quality models allow analysis of a set of data quality requirements and their representation in terms of a conceptual schema, as well as accessing and querying data quality dimensions by means of a logical schema. Data quality models also include process models tailored to analysis and design of quality improvement actions. These models permit tracking data from their source, through various manipulations that data can undergo, to their final usage. In this way, they support the detection of causes of poor data quality and the design of improvement actions.

Historical Background

Among the first data quality models, in 1990 the *polygen model* [5] was proposed for explicitly tracing the origins of data and the intermediate sources used to arrive at that data. The model is targeted to heterogeneous distributed systems and is a first attempt to represent and analyze the provenance of data, which has been recently investigated in a more general context.

In the mid-1990's, there was a first proposal of extending the relational model with quality values associated to each attribute value, resulting in the *quality attribute model* [6]. An extension of the Entity Relationship model was also proposed in the same years ([4], and [7], Chapter 3), similarly focused on associating quality dimensions, such as accuracy or completeness, to attributes. More recently, models for associating quality values to data-oriented XML documents have been investigated (e.g., [2]). Such models are intended to be used in the context of distributed and cooperative systems, in which the cooperating organizations need to exchange data each other, and it is therefore critical for them to be aware of the quality of such data. These models are semi-structured, thus allowing each organization to export the quality of its data with a certain degree of flexibility; quality dimensions can be associated to various elements of the data model, ranging from the single data value to the whole data source, in this way being different from the previous attribute-based models.

The principal data quality models that are oriented towards process representation are based on the principle that data can be seen as a particular product of a manufacturing activity, and so descriptive models (and methodologies) for data quality can be based on models conceived in the last two centuries for manufacturing traditional products. The *Information Product Map (IP-MAP)* [3] is a significant example of such models and follows this view, being centered on the concept of information product. The IP-MAP model has been extended in several directions (see [1], Chap. 3). Indeed, more powerful mechanisms have been included, such as *event process chain diagrams* representing the business process overview, the interaction model (how company units interact), the organization model (who does what), the component model (what happens), and the data model (what data is needed). A further extension called *IP-UML* consists of a UML profile for data quality based on IP-MAP.

Foundations

Data quality models can be distinguished in data-oriented models, focused on the representation of data quality dimensions, and process-oriented models focused on the representation of the processes that manipulate data and on their impact on the data quality.

Data-oriented models include extensions of the Entity Relationship model, of the relational model, and of the XML data model.

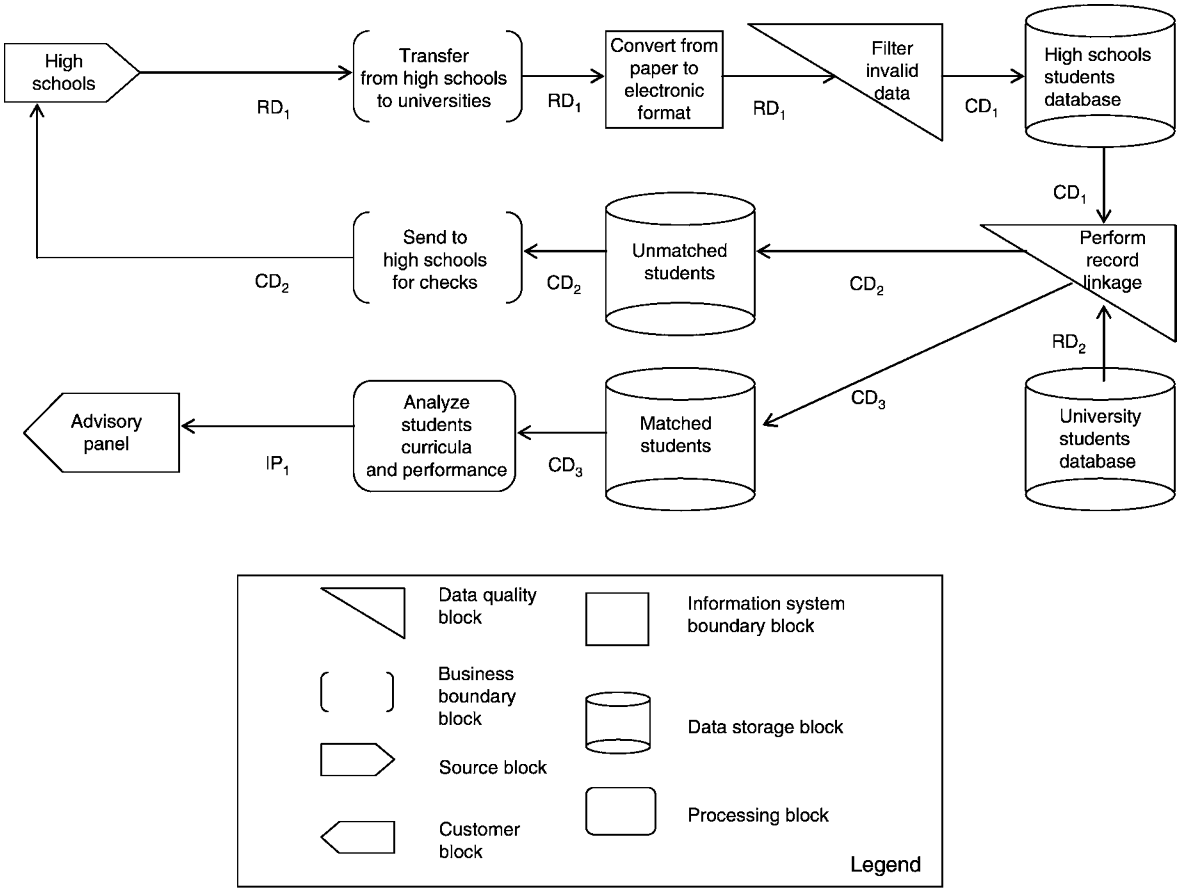
When extending the Entity Relationship model for representing data quality, one possibility is to introduce two types of entities, explicitly defined to express quality dimensions and their values: a data quality dimension entity and a data quality measure entity. The goal of the data quality dimension entity is to represent possible pairs `<DimensionName, Rating>` of dimensions and corresponding ratings resulting from measurements. The data quality dimension entity characterizes the quality of an attribute and the scale may obviously depend on the attribute. In these cases, it is necessary to extend the properties of the data quality dimension entity to include the attribute, that is `<DimensionName, Attribute, Rating>`.

In order to represent metrics for dimensions, and the relationship with entities, attributes, and dimensions, the model introduces the data quality measure entity; its attributes are Rating, the values of which depend on the specific dimension modeled, and DescriptionOfRating. The complete data quality schema, shown by means of the example in Fig. 1, is made up of:

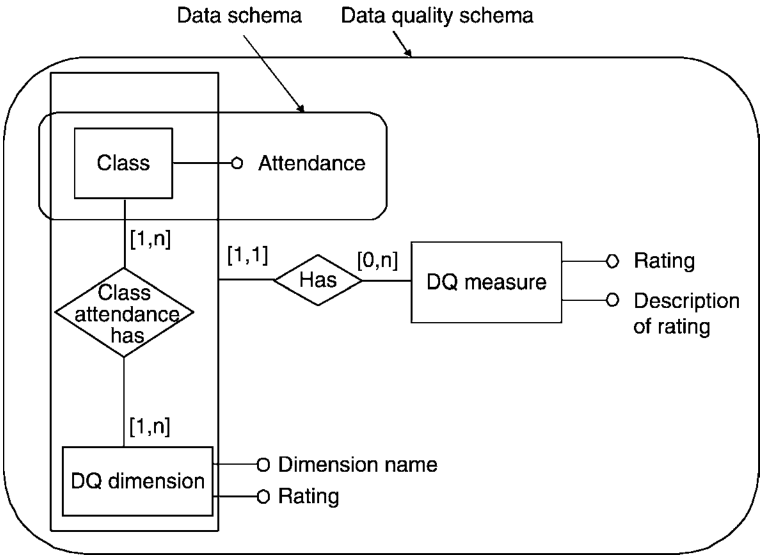
- The original data schema, in the example represented by the entity Class with the attribute Attendance.
- The DQ Dimension entity with a pair of attributes <DimensionName, Rating >.
- The relationship between the entity Class, the related attribute Attendance, and the DQ Dimension entity with a many-to-many relationship ClassAttendanceHas; a distinct relationship has to be introduced for each attribute of the entity Class.
- The relationship between the previous structure and the DQ Measure entity with a new representation structure that extends the Entity Relationship model, and relates entities and relationships.

An extension of the relational data model is provided by the quality attribute model, explained in the following by means of the example shown in Fig. 2.

The figure shows a relational schema Employee, defined on attributes EmployeeId, Address, DateofBirth, and others, and one of its tuples. Relational schemas are extended adding an arbitrary number of underlying levels of quality indicators (only one level in the figure) to the attributes of the schema, to which they are linked through a quality key. In the example, the attribute EmployeeId is extended with one quality attribute, namely accuracy, the attribute Address with two quality attributes, namely accuracy and currency, while the attribute DateofBirth is extended with accuracy and completeness. The



Data Quality Models. Figure 1. An example of IP-MAP.



Data Quality Models. Figure 2. An extension of the entity relationship model.

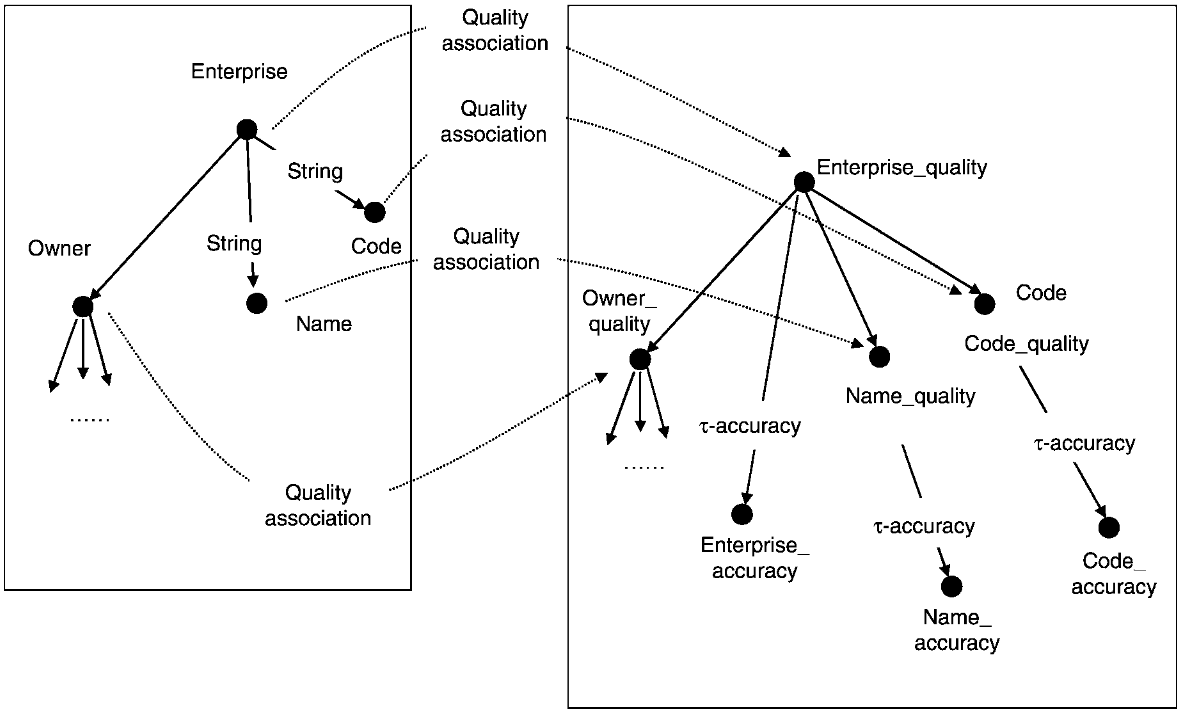
values of such quality attributes measure the quality dimensions' values associated with the whole relation instance (top part of the figure). Therefore, completeness equal to 0.8 for the attribute `DateofBirth` means that the 80% of the tuples have a non-null value for such an attribute. Similar structures are used for the instances of quality indicator relations (bottom part of the figure); if there are `n` attributes of the relational schema, `n` quality tuples will be associated to each tuple in the instance.

The model called Data and Data Quality (D^2Q) is among the first models for associating quality values to data-oriented XML documents. D^2Q can be used in order to certify dimensions like accuracy, consistency, completeness, and currency of data. The model is semi-structured, thus allowing each organization to export the quality of its data with a certain degree of flexibility. More specifically, quality dimension values can be associated with various elements of the data model, ranging from the single data value to the whole data source. The main features of the D^2Q model are summarized as follows:

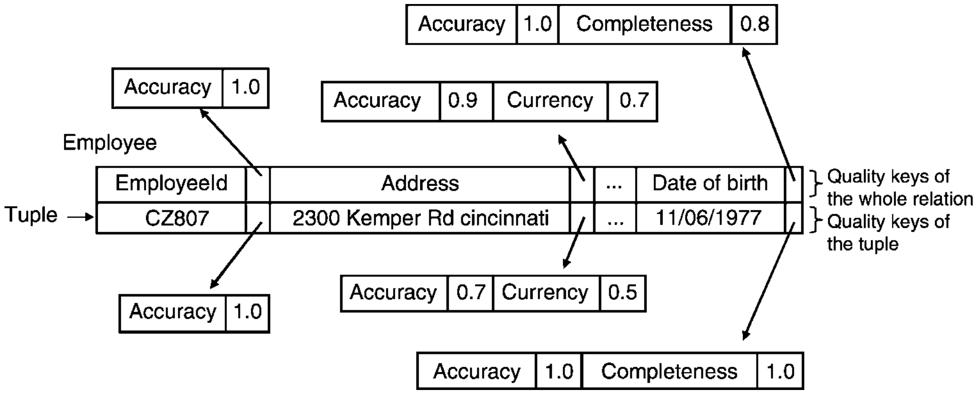
- A data class and a data schema are introduced to represent the business data portion of the D^2Q model.
- A quality class and a quality schema correspond to the quality portion of the D^2Q model.
- A quality association function that relates nodes of the graph corresponding to the data schema to nodes of the graph corresponding to the quality schema. Quality associations represent biunivocal functions among all nodes of a data schema and all non-leaf nodes of a quality schema.

In Fig. 3, an example of a D^2Q schema is depicted. On the left-hand side of the figure, a data schema is shown representing enterprises and their owners. On the right-hand side, the associated quality schema is represented. Specifically, two quality classes, `Enterprise_Quality` and `Owner_Quality` are associated with the `Enterprise` and `Owner` data classes. Accuracy nodes are shown for both data classes and related properties. For instance, `Code_accuracy` is an accuracy node (of type *τ-accuracy*) associated with the `Code` property, while `Enterprise_accuracy` is an accuracy node associated with the data class `Enterprise`. The arcs connecting the data schema and the quality schema with the quality labels represent the quality association functions. The D^2Q model is intended to be easily translated into the XML data model. This is important for meeting the interoperability requirements that are particularly stringent in cooperative systems.

Process-oriented models have their principal representative in the Information Product Map (IP-MAP) model. An information product map is a graphical model designed to help people comprehend, evaluate, and describe how an information product, such as an invoice, a customer order, or a prescription, is assembled in a business process. IP-MAPs are designed to help analysts visualize the information production process, identify ownership of process phases, understand information and organizational boundaries, and estimate time and quality metrics associated with the current production process. There are eight types of construct blocks that can be used to form the IP-MAP: source, customer, data quality, processing, data storage, decision, business boundary, and information systems boundary. An example of information product map is shown in Fig. 4. Information products (IP in the figure) are produced by means of processing activities and data quality checks on raw data (RD), and semi-processed information called component data (CD). In the example, it is assumed that high schools and universities of a district have decided to cooperate in order to improve their course offering to students, avoiding overlap and being more effective in the



Data Quality Models. Figure 3. An example of a D^2Q schema.


Accuracy 1.0 Completeness 0.8
Accuracy 1.0
Accuracy 0.9 Currency 0.7
Employee
EmployeeId
Address
...
Date of birth
Tuple
CZ807
2300 Kemper Rd cincinnati
...
11/06/1977
Quality keys of the whole relation
Quality keys of the tuple
Accuracy 0.7 Currency 0.5
Accuracy 1.0
Accuracy 1.0 Completeness 1.0

Data Quality Models. Figure 4. An extension of the relational model.

education value chain. To this end, they have to share historical data on students and their curricula. Therefore, they perform a record linkage activity that matches students in their education life cycle ("Perform Record Linkage" block). To reach this objective, high schools periodically supply relevant information on students; in case it is in paper format, the information has to be converted in electronic format. At this point, invalid data are filtered and matched with the database of university students. Unmatched students are sent back to high schools for clerical checks, and matched students are analyzed. The result of the analysis of curricula and course topics are sent to the advisory panel of the universities.

Key Applications

Data-oriented and process-oriented data quality models can be used to represent quality dimensions and quality related activities thus supporting techniques for data quality improvement. However, such techniques seldom rely on the described model extensions, with the distinctive exception of the IP-MAP model. Indeed, only a few prototypical DBMSs have experienced the adoption of some of the approaches mentioned. This is mainly due to the complexity of the representational structures proposed in the different approaches. Indeed, measuring data quality is not an easy task, hence models that impose to associate data quality dimensions values at attribute level have proven not very useful in practice. A greater flexibility is more useful in real applications, like for instance, in scenarios of e-Government or e-Commerce. In these scenarios, which involve cooperation between different organizations, a more successful case is provided by XML data exchanged with associated quality profiles, which are based on semi-structured data quality models.

Future Directions

The future of research on models appears to be in provenance and semi-structured data quality models. In open information systems and in peer-to-peer ones, knowing the provenance and having a flexible tool to associate quality to data is crucial. Indeed, such systems have to be able to trace the history of data and to certify the level of quality of the retrieved data.

Cross-references

▶ Data Quality Dimensions
▶ Entity Relationship Model
▶ Information Product Management
▶ Provenance
▶ Relational Model
▶ Semi-Structured Data
▶ XML

Recommended Reading

1. Batini C. and Scannapieco M. Data Quality: Concepts, Methodologies, and Techniques. Springer, Berlin, 2006.
2. Scannapieco M., Virgillito A., Marchetti C., Mecella M., and Baldoni R. The DaQuinCIS architecture: a platform for exchanging and improving data quality in cooperative information systems. Inf. Syst., 29(7):551–582, 2004.
3. Shankaranarayan G., Wang R.Y., and Ziad M. Modeling the manufacture of an information product with IP-MAP. In Proc. 5th Int. Conf. on Information Quality, 2000, pp. 1–16.
4. Storey V.C. and Wang R.Y. An analysis of quality requirements in database design. In Proc. 4th Int. Conf. on Information Quality, 1998, pp. 64–87.
5. Wang R.Y. and Madnick S.E. A polygen model for heterogeneous database systems: the source tagging perspective. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 519–538.
6. Wang R.Y., Reddy M.P., and Kon H. Toward data quality: an attribute-based approach. Decision Support Syst., 13(3–4):349–372, 1995.
7. Wang R.Y., Ziad M., and Lee Y.W. Data Quality. Kluwer, Boston, MA, USA, 2001.

Data Quality Problems

▶ Data Conflicts

Data Quality Representations

▶ Data Quality Models

Data Rank/Swapping

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona, Catalonia

Synonyms

Data swapping; Rank swapping

Definition

Data swapping was originally designed by Dalenius and Reiss [1] as a masking method for statistical disclosure control of databases containing only categorical attributes. The basic idea behind the method is to transform a database by exchanging values of confidential attributes among individual records. Records are exchanged in such a way that low-order frequency counts or marginals are maintained.

Rank swapping is a variant of data swapping [2,3]. First, values of an attribute X_i are ranked in ascending order, then each ranked value of X_i is swapped with another ranked value randomly chosen within a restricted range (e.g., the rank of two swapped values cannot differ by more than $p\%$ of the total number of records, where p is an input parameter). This algorithm is independently used on each original attribute in the original data set.

Key Points

It is reasonable to expect that multivariate statistics computed from data swapped with this algorithm will be less distorted than those computed after an unconstrained swap. In empirical work on SDC scores, rank swapping with small swapping range has been identified as a particularly well-performing method in terms of the trade-off between disclosure risk and information loss. Consequently, it is one of the techniques that have been implemented in the $\mu - Argus$ package [3].

Cross-references

- ▶ Data Rank/Swapping
- ▶ Disclosure Risk
- ▶ Inference Control in Statistical Databases
- ▶ Information Loss Measures
- ▶ K-anonymity
- ▶ Microaggregation
- ▶ Microdata
- ▶ Microdata rounding
- ▶ Noise Addition
- ▶ Non-perturbative masking methods
- ▶ Pram
- ▶ Record Linkage
- ▶ Synthetic Microdata
- ▶ SDC Score
- ▶ Tabular Data

Recommended Reading

1. Dalenius T. and Reiss S.P. Data-swapping: a technique for disclosure control (extended abstract). In Proc. ASA Section on Survey Research Methods. American Statistical Association, Washington DC, 1978, pp. 191–194.
2. Domingo-Ferrer J. and Torra V. A quantitative comparison of disclosure control methods for microdata. In Confidentiality, Disclosure and Data Access: Theory and Practical Applications for Statistical Agencies. P. Doyle, J.I. Lane, J.J.M. Theeuwes, and L. Zayatz (eds.). Amsterdam, North-Holland, 2001, pp. 111–134.
3. Hundepool A., Van de Wetering A., Ramaswamy R., Franconi F., Polettini S., Capobianchi A., De Wolf P.-P., Domingo-Ferrer J., Torra V., Brand R. and Giessing S. μ-Argus User's Manual version 4.1, February 2007.

Data Reconciliation

▶ Constraint-Driven Database Repair

Data Reduction

Rui Zhang
University of Melbourne, Melbourne, VIC, Australia

Definition

Data reduction means the reduction on certain aspects of data, typically the volume of data. The reduction can also be on other aspects such as the dimensionality of data when the data is multidimensional. Reduction on any aspect of data usually implies reduction on the volume of data.

Data reduction does not make sense by itself unless it is associated with a certain purpose. The purpose in turn dictates the requirements for the corresponding data reduction techniques. A naive purpose for data reduction is to reduce the storage space. This requires a technique to compress the data into a more compact format and also to restore the original data when the data needs to be examined. Nowadays, storage space may not be the primary concern and the needs for data reduction come frequently from database applications. In this case, the purpose for data reduction is to save computational cost or disk access cost in query processing.

Historical Background

The need for data reduction arises naturally. In early years (pre-1990's), storage was quite limited and expensive. It fostered the development of a class of techniques called *compression techniques* to reduce the data volume for lower consumption of resources such as storage space or bandwidth in telecommunication settings. Another requirement for a compression technique is to reproduce the original data (from the compressed data) for reading. Here "reading" has different meanings depending on the data contents. It means "listening" for audio data, "viewing" for video data, "file reading" for general contents, etc. Therefore the reproduction of the data should be either exactly the same as the original data or very close to the original data by human perception. For example, MP3 is an audio compression technique which makes a compressed audio sound almost the same to the original one. Until today, compression techniques is still a very active research topic. But, instead of concerning data size of kilobytes or megabytes as in the early years, today's compression techniques concern data size of gigabytes or even terabytes.

As the rapid advances of hardware technologies, storage limit is no longer the most critical issue in many cases. Another huge force driving the need for data reduction appears in database applications. Storing the data may not be a problem, but retrieving data from the storage (typically hard disk) is still a quite expensive operation due to the slow improvement in disk seek time. Database queries commonly need to retrieve large amount of data from the disk. Therefore data reduction is compelling for providing high performance in query processing. Different from data compression, data reduction in database applications usually do not need to generate a reproduction that is exactly the same as the original data or sounds/looks very close to the original data. Instead, an approximation of the intended answer suffices, which gives more flexibility for data reduction.

Traditionally, data reduction techniques have been used in database systems to obtain summary statistics, mainly for estimating costs of query plans in a query optimizer. Here, an approximation of the expected cost suffices as an estimate. At the same time, highly reduced data (summary statistics) is essential to make evaluation of the query plans much cheaper than evaluation of the query.

In the last two decades, there has been enormous interest in *online analytic processing (OLAP)*, which is characterized by complex queries involving group-by and aggregation operators on extremely large volume of data. OLAP is mainly performed in *decision support* applications, which analyze data and generate summaries from data. Organizations need these results to support high-level decision making. The data typically comprises of data consolidated from many sources of an organization, forming a repository called a *data warehouse*. In face of the high data volume, efficient OLAP calls for data reduction techniques. Due to the analytical and exploratory nature of the queries, approximate answers are usually acceptable and the error tolerance can sometimes be quite high.

Foundations

Compression techniques and data reduction techniques in databases are discussed separately below due to the differences in their purposes and general characteristics. Compression techniques are more often studied in the information retrieval research community while data reduction techniques in databases are studied in the database research community. Compression techniques is a subcategory of data reduction techniques, although sometimes the term *compression technique* is used in a less strict way to refer to data reduction in general.

Compression techniques involve the processes of *encoding* and *decoding*. Encoding converts the original data to a more compact format based on a mapping from *source messages* into *codewords*. Decoding performs the inverse operation to reproduce the original data. If the reproduction is exactly the same as the original data, the compression technique is *lossless*; otherwise, it is *lossy*. Lossless compression techniques are used for generally any data format without needing to know the contents or semantics of the data. Popular techniques include *ZIP* invented by Phil Katz in late 1980s and *RAR* invented by Eugene Roshal in early 1990s. If some domain knowledge on the data is available, usually *lossy* compression techniques yield better compression rates. For example, JEPG, MP3 and MPEG are popular compression techniques for audio, image and video data, respectively. Lossy compression techniques leave out the less important information and noise to achieve higher compression. More concretely, the MP3 audio encoding format removes the audio details most human beings cannot hear to make the compressed audio sound like a faithful reproduction of the original uncompressed one. Different compression techniques mainly differ in the mapping from source messages into codewords. A survey of compression techniques is given in [6]. Readers interested in recent research results in compression techniques are referred to the proceedings of the Data Compression Conference [1].

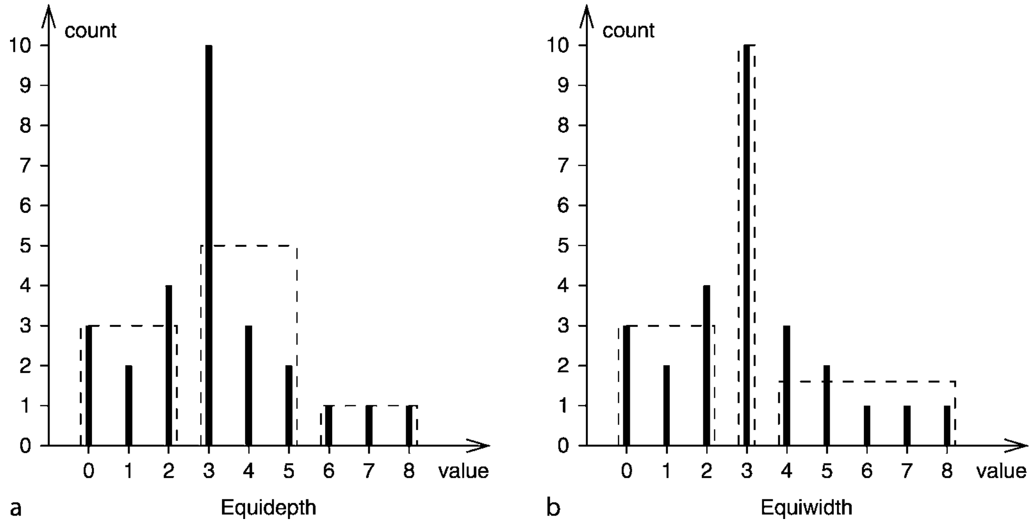
Data reduction in databases can make use of various techniques. Popular ones include histograms, clustering, singular value decomposition (SVD), discrete wavelet transform (DWT), etc. The techniques can be divided into two categories, *parametric* and *nonparametric* techniques, depending on whether the technique assumes a certain model. Histograms and clustering are nonparametric techniques while SVD and DWT are parametric techniques. A summary of data reduction techniques for databases can be found in [3].

Histograms

A histogram is a data structure used to approximate the distribution of values. The value domain is divided into subranges called *buckets*. For each bucket, a count of the data items whose values fall in the bucket is maintained. Therefore a histogram basically contains the information of the boundaries of the buckets and the counts. The data distribution can be approximated by the average values of the buckets and the counts. Commonly, two types of histograms are used, *equiwidth* and *equidepth* histograms, distinguished by how the buckets are determined. In an equiwidth histogram, the length of every bucket is the same while in an equidepth histogram, the count for every bucket is the same (Sometimes, an exact same count cannot be achieved and then the counts for the buckets are approximately the same.). Figure 1 shows an example data distribution in the value domain [0,8] and the equiwidth and equidepth histograms for the data assuming three buckets. A thick vertical line represents the count for a value; a dashed line represent a bucket range and the estimated count for the values in the bucket. The estimated count of a certain value is simply the average count in the bucket. In the equiwidth histogram (Fig. 1a), each bucket has the range of length 3. The estimated counts for most values are quite close to the actual values. Equiwidth histograms are simple and easy to maintain, but the estimate is less accurate for skewed distribution such as the count of value 3. This problem is alleviated in the equidepth histogram (Fig. 1b). Each bucket has the count of about 9. The estimate count for value 3 is very accurate. The disadvantage of equidepth histograms is that determining the boundaries of buckets is more difficult. There are other types of histograms such as *compressed* histograms and *v-optimal* histograms. A thorough classification on various histograms can be found in [7].

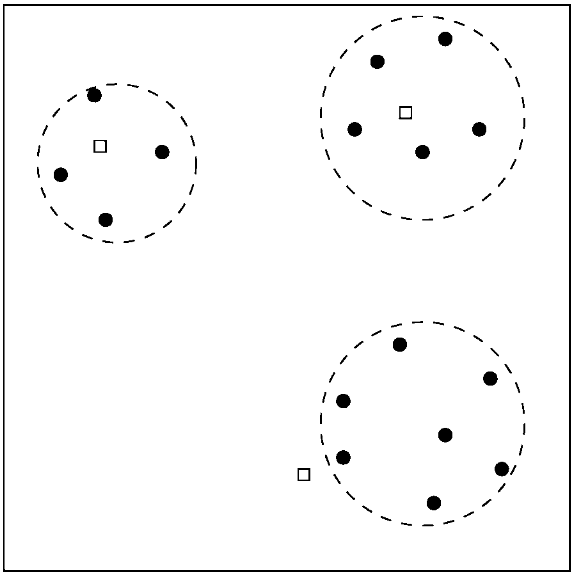
Clustering

Clustering is a technique to partition objects into groups called *clusters* such that the objects within a group are similar to each other. After clustering, operations can be performed on objects collectively as groups. The information of data can be represented at the cluster level and hence greatly reduced. The data to perform clustering on usually contain multiple attributes. Therefore each data object can be represented by a multidimensional point in space. The similarity is measured by a distance function. Typically, a metric function, such as Euclidean



Data Reduction. Figure 1. Histograms.

distance, is used as the distance function. Given a data set, there is no single universal answer for the problem of clustering. The result of clustering depends on the requirements or the algorithm used to perform clustering. A classic algorithm is the *k-means* algorithm, which partitions the data into k clusters. Given a data set and a number k, the algorithm first picks k points randomly or based on some heuristics to serve as cluster centroids. Second, every point (object) is assigned to its closest centroid. Then the centroid for each cluster is recomputed based on the current assignment of points. If the newly computed centroids are different from the previous ones, all the points are assigned to their closest centroids again and then the centroids are computed again. This process is repeated until the centroids do not change. Figure 2 shows an example where $k = 3$. The black dots are data points, squares are initial centroids and the dashed circles show the resultant clusters. In the beginning, the value of k is given by the user in a very subjective manner, usually depending on the application needs. Another algorithm called *k-medoid* works in a very similar manner but with a different way of choosing their cluster representatives, called *medoids*, and with a different stop condition. Recently, algorithms designed for large data sets were proposed in the database research community such as BIRCH [9] and CURE [4].



Data Reduction. Figure 2. *k*-means clustering.

Singular Value Decomposition (SVD)

Any $m \times n$ real matrix A can be decomposed as follows:

$$\mathbf{A} = \mathbf{USV}^t \qquad (1)$$

where $\mathbf{U}$ is a column-orthonormal $m \times r$ matrix, r is the rank of the matrix $\mathbf{A}$, $\mathbf{S}$ is a diagonal $r \times r$ matrix and $\mathbf{V}$ is a column-orthonormal $n \times r$ matrix (bold symbols are used to represent matrices and vectors). It can be further expressed in the *spectral decomposition* [5] form:

$$\mathbf{A} = \lambda_1 \mathbf{u}_1 \mathbf{v}_1^t + \lambda_2 \mathbf{u}_2 \mathbf{v}_2^t + \ldots + \lambda_r \mathbf{u}_r \mathbf{v}_r^t \qquad (2)$$

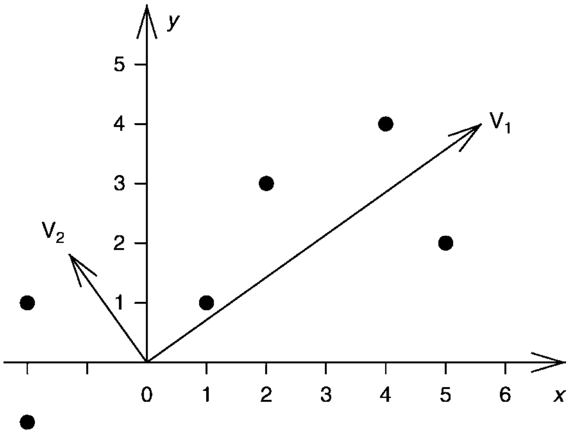
where $\mathbf{u}_i$ and $\mathbf{v}_i$ are column vectors of $\mathbf{U}$ and $\mathbf{V}$, respectively, and λ_i are the diagonal elements of $\mathbf{S}$. $\mathbf{A}$ can be viewed as m n-dimensional points (each row being a point). Because $\mathbf{v}_i$ are orthogonal vectors, they form a new basis of the space. λ_i represents the importance of the basis vector $\mathbf{v}_i$ (dimension i) and $\mathbf{u}_i$ represents the coordinates of the m points in dimension i in this new coordinate system. Assume that λ_i are sorted in descending order. Then, $\mathbf{v}_1$ is the direction (dimension) with the largest dispersion (variance) of the points; $\mathbf{v}_2$ is the direction with the second largest dispersion of the points, and so on. If the last few λ_i values are small and one omits them when calculating $\mathbf{A}$, the resulted error will be very small. Therefore SVD is widely used in dimensionality reduction and matrix approximation. The following is an example, with $\mathbf{A}$ given as

$$\mathbf{A} = \begin{bmatrix} -2 & 1 \\ -2 & -1 \\ 1 & 1 \\ 2 & 3 \\ 4 & 4 \\ 5 & 2 \end{bmatrix}$$

The SVD of $\mathbf{A}$ is

$$\mathbf{A} = \begin{bmatrix} -0.118 & 0.691 \\ -0.250 & 0.125 \\ 0.158 & 0.079 \\ 0.383 & 0.441 \\ 0.633 & 0.316 \\ 0.593 & -0.454 \end{bmatrix} \begin{bmatrix} 8.82 & 0 \\ 0 & 2.87 \end{bmatrix} \begin{bmatrix} 0.811 & 0.585 \\ -0.585 & 0.811 \end{bmatrix}$$

Here, $\lambda_1 = 8.82, \lambda_2 = 2.87, \mathbf{v}_1 = \begin{bmatrix} 0.811 \\ 0.585 \end{bmatrix}$ and $\mathbf{v}_2 = \begin{bmatrix} -0.585 \\ 0.811 \end{bmatrix}$. Figure 3 shows the data points, and the directions of $\mathbf{v}_1$ and $\mathbf{v}_2$.



Data Reduction. Figure 3. SVD.

Discrete Wavelet Transform (DWT)

Wavelet Transform is a commonly used signal processing technique like other transforms such as *Fourier Transform* or *Cosine Transform*. In databases, commonly used is the discrete version called Discrete Wavelet Transform (DWT). After applying DWT, a multi-resolution decomposition of the original signal is obtained in the form of wavelet coefficients. The wavelet coefficients are projections of the signal onto a set of orthogonal basis vectors. The choice of the basis vectors determines the type of DWT. The most popular one is the Haar transform, which is easy to implement and fast to compute. Some of the wavelet coefficients obtained may be small, therefore they can be replaced by zeros and hence the data is reduced. Inverse DWT can be applied on the reduced wavelet coefficients to get an approximation of the original signal. This is basically how DWT is used for compression. DWT based compression provides better lossy compression than Discrete Fourier Transform and Discrete Cosine Transform.

In the Haar transform, elements of a signal are processed pairwise. Specifically, the average and difference of every two neighboring elements are computed. The averages serve as a lower-resolution approximation of the signal and the differences (divided by 2) are the coefficients. For example, signal $S = \{2, 4, 5, 5, 3, 1, 2, 2\}$. Computing the average of every two neighboring elements results in a lower-resolution signal $S_1 = \{3, 5, 2, 2\}$. The coefficients are obtained by computing the difference of every two neighboring elements divided by 2, which is $D_1 = \{-1, 0, 1, 0\}$. S can be restored exactly by adding (or subtracting) the coefficient to the corresponding element in S_1. For example, $S(1) = S_1(1) + D_1(1) = 3 + (-1) = 2$; $S(2) = S_1(1) - D_1(1) = 3 - (-1) = 4$. Similarly, a even lower-resolution signal S_2 can be obtained by applying the same process on S_1. This can be done recursively until the length of the signal becomes 1. The full decomposition on S is shown in Fig. 4. The Haar transform of S is given as the average over all elements (3), and all the coefficients, $S' = \{3, 1, -1, 0, -1, 0, 1, 0\}$.

Resolution	Averages	Coeffecients
8	2, 4, 5, 5, 3, 1, 2, 2	
4	3, 5, 2, 2	–1, 0, 1, 0
2	4, 2	–1, 0
1	3	1

Data Reduction. Figure 4. Haar transform.

Key Applications

Data Storage and Transfer

Compression techniques are essential for data storage and transfer in many applications.

Database Management Systems

Histograms is a popular technique for maintaining summary information in database management systems. It is especially useful for a cost-based query optimizer.

OLAP

Due to the huge volume of data in OLAP applications, data reduction techniques such as sampling are commonly used to obtain quick approximate answers.

Multimedia Data

Multimedia data is characterized by large size. Therefore data reduction techniques are usually applied on multimedia data from storage to data processing. For example, the MP3, JPEG, MPEG formats for audio, image and video data, respectively, all use compression techniques. The new JPEG digital image standard, JPEG-2000, uses DWT for all its codecs [8]. Similarity search on multimedia data usually needs to deal with very high-dimensional point representations. SVD can be used to reduce the dimensionality to achieve better search performance. In a recent paper [2], DWT is used to represent 3D objects to obtain better data retrieval performance.

Taxonomy

Clustering is widely used in almost all taxonomy applications such as taxonomies of animals, plants, diseases and celestial bodies. It can also help visualization through a hierarchically clustered structure.

Cross-references

- ▶ Clustering
- ▶ Discrete Wavelet Transform and Wavelet Synopses
- ▶ Histogram
- ▶ Nonparametric Data Reduction Techniques
- ▶ Parametric Data Reduction Techniques
- ▶ Singular Value Decomposition

Recommended Reading

1. http://www.cs.brandeis.edu/∼dcc/index.html.
2. Ali M.E., Zhang R., Tanin E., and Kulik L. A motion-aware approach to continuous retrieval of 3D objects. In Proc. 24th Int. Conf. on Data Engineering, 2008.
3. Barbará D., DuMouchel W., Faloutsos C., Haas P.J., Hellerstein J.M., Ioannidis Y.E., Jagadish H.V., Johnson T., Ng R.T., Poosala V., Ross K.A., and Sevcik K.C. The New Jersey data reduction report. IEEE Data Eng. Bull., 20(4):3–45, 1997.
4. Guha S., Rastogi R., and Shim K. CURE: an efficient clustering algorithm for large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 73–84.
5. Jolliffe I.T. Principal component analysis. Springer, Berlin, 1986.
6. Lelewer D.A. and Hirschberg D.S. Data compression. ACM Comput. Surv., 19(3):261–296, 1987.
7. Poosala V., Ioannidis Y.E., Haas P.J., and Shekita E.J. Improved histograms for selectivity estimation of range predicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 294–305.
8. The JPEG 2000 standard. http://www.jpeg.org/jpeg2000/index.html.
9. Zhang T., Ramakrishnan R., and Livny M. BIRCH: an efficient data clustering method for very large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 103–114.

Data Replication

Bettina Kemme
McGill University, Montreal, QC, Canada

Synonyms

Database replication; Replication

Definition

Using data replication, each logical data item of a database has several physical copies, each of them located on a different machine, also referred to as site or node. Depending on the context and the type of replication architecture, the term *replica* can refer to one of the physical copies of a particular data item, or to an entire site with all its data copies. Data replication can serve different purposes. First, it can be used to increase *availability* and provide *fault-tolerance* since the data can, in principle, be accessed as long as one replica is available. Second, it can provide good *performance*. By storing replicas close to users that want to access the data, replication allows *fast local access*. Third, access requests can be distributed across the replicas. By adding more replicas to the system a higher incoming workload can be handled, and hence, a higher throughput can be achieved. Thus, replication is a means to achieve *scalability*. Finally, for some applications, replication is a natural choice, e.g., if mobile users have to access data while disconnected from their corporate data server.

The main challenge of replication is that the replicas have to be kept consistent when updates occur. This is the task of *replica control*. It has to translate the read and write operations that users submit on the logical data items into operations on the physical copies. In the most common approach, read operations are performed on one replica while write operations have to be performed on all replicas (ROWA, or read-one-write-all approach). Ideally, all copies of a data item have the same value at all times. In reality, many different *consistency models* have been developed that reflect the needs for different application domains. Additionally, *replica allocation algorithms* have the task to decide where and when to install or remove replicas. They have to find a trade-off between the performance benefits for read operations and the overhead of keeping the copies consistent.

Historical Background

Data replication has gained attention in two different communities. The database community typically considers the replication of data items of a database, e.g., records or tables of a relational database. It has to consider that transactions and queries not only access individual data items, but read and write a whole set of related data items. In contrast, the distributed systems community mainly focuses on replication techniques for objects that are typically accessed individually, such as distributed file systems and web-servers. Nevertheless, in all the application types, similar issues arise regarding replica control and allocation, and the associated coordination and communication overhead. Thus, there has always been an active exchange of research ideas, such as [1,5], and there exist several publication venues where work from both communities appears.

Work on database replication had an early peak in the 1980s, where it was first introduced for availability purposes, and most approaches provided strong consistency properties. A good overview is given in [2]. A seminal paper by Gray et al. in 1996 [6] revived research in this area. It provided a first characterization of replica control algorithms and presented an analytical model showing that existing strong consistency solutions come with a large performance penalty. Since then, replication has remained an active research area. Emphasis has been put on reducing the high communication and coordination overhead of the early solutions. One research direction aims at reducing the costs by delaying the sending of updates to remote replicas. However, in this case, replicas might have stale or even inconsistent data. Solutions have been proposed to avoid inconsistencies (e.g., [3]), to provide limits on the staleness of the data (e.g., [8]), and to detect and then reconcile inconsistencies [9]. Another research direction has developed techniques to provide strong consistency guarantees at acceptable costs, for example, by taking advantage of multicast and group maintenance primitives provided by group communication systems [14].

In the distributed systems community, early work focused on replicated file systems (e.g., [10,13]). Later, web-server replication [12] and file replication in peer-2-peer systems, (e.g., [7]) have attained considerable attention. A wide range of consistency models has been defined to accommodate application needs. Also, a large body of literature exists regarding object replication for fault-tolerance purposes [4,11].

Foundations

Replica Control

Replica control, which has the task of keeping the copies consistent despite updates, is the main issue to be tackled by any replication solution. Replica control has to decide which data copies read operations should access, and when and how to update individual data copies in case of write operations. Thus, most of the work done in the area of database replication is to some degree associated with replica control. The entry *Replica Control* provides a detailed overview of the main challenges.

Replica Control and Concurrency Control

In order to work properly with a database system providing transactional semantics, replica control has to be integrated with concurrency control. Even in a nonreplicated and nondistributed system, as soon as transactions are allowed to execute concurrently, concurrency control mechanisms restrict how the read and write operations of different transactions may interleave in order to provide each transaction a certain level of isolation. If data items are now replicated, the issue becomes more challenging. In particular, different transactions might start their execution on different replicas making it difficult to detect conflicts.

For a nonreplicated database system, the most studied isolation level is *serializability*, which indicates that the concurrent execution of transactions should be equivalent to a serial execution of the same transactions. This is typically achieved via locking, optimistic, or multi-version concurrency control. Thus, one of the first steps in the research of replicated databases was to define a corresponding correctness criterion 1-copy-serializability, which requires that the concurrent execution of transactions over all replicas is equivalent to a serial execution over a single logical copy of the database. Many replica control algorithms have been developed to provide this correctness criterion, often extending the concurrency control algorithms of non-replicated systems to work in a replicated environment.

In early solutions, replicas run some form of coordination during transaction execution to guarantee an appropriate serialization. This type of protocols is often called *eager* or *synchronous* since all replicas coordinate their operations before transaction commit. Textbooks on distributed systems typically contain a chapter on these replica control algorithms since they serve as a nice example of how to design coordination protocols in a distributed environment. A problem with most of these traditional replication solutions is that they induce a large increase in transaction response times which is often not acceptable from an application point of view.

More recent approaches addressed this issue and designed replica control algorithms providing 1-copy-serializability or other strong consistency levels that are tuned for performance. Many envision a cluster architecture, where a set of database replicas is connected via a fast local area network. In such a network, eager replication can be feasible since communication delays are short. Replication is used to provide both scalability and fault-tolerance. The entries *Replication based on Group Communication* and *Replication for Scalability* describe recent, efficient replica control algorithms providing strong consistency levels for cluster architectures.

Consistency Models and Conflict Resolution

By definition, eager replication incorporates coordination among replicas before transaction commit. Alternatively, *lazy* replication (also called asynchronous or optimistic) allows a transaction to commit data changes at one replica without coordination with other replicas. For instance, all update transactions could be executed and committed at a specific primary replica which then propagates changes to other replicas sometime after commit. Then, secondary replicas lag behind the current state at the primary. Alternatively, all replicas might accept and execute updates, and eventually propagate the changes to the rest of the replicas. In this case, replicas can become inconsistent. Conflicting updates are only detected after update propagation and inconsistent data has to be reconciled. In this context, a considerable body of research exists defining correctness criteria weaker than 1-copy-serializability. In particular, many formalisms exist that allow the defining of limits to the allowed divergence between the copies of a data item.

Availability

Replication can increase the availability of the system since, in principle, a data item is available as long as one replica is accessible. However, in practice, it is not trivial to design a high-available replication solution. As discussed before, most replication algorithms require all updates to be performed at all replicas (ROWA, or read-one-write-all). If this is taken in the strict sense, then, if one replica is update transactions cannot execute, and the availability observed by update transactions is actually lower than in a nonreplicated system. To avoid this, most replica control algorithms actually implement a read-one-write-all-available (ROWAA) strategy that only updates replicas that are actually accessible. When a replica recovers after a crash, it first has to get the current state from the available replicas. This can be a complex process. The possibility of network partitions imposes an additional challenge. Although a particular replica might be up and running, it might not be accessible because of an interruption in the network connectivity. Many commercial database systems offer specialized high-availability replication solutions. Typically, a primary database is replicated at a backup database system. All transactions are executed at the primary that sends updates to the backend. Only when the primary crashes, the backup takes over.

Quorum systems are an alternative high-availability replication approach. In quorum approaches both read and write operations have to access a quorum of data replicas. For example, a quorum could be a majority of replicas. This guarantees that any two operations overlap in at least one replica. Thus, each read operation reads at least one replica that has the most recent update, and any two concurrent write operations are serialized at least at one replica. There exist many ways to define quorums differing in the structure and the sizes of the quorums. Quorums are an elegant solution to network partitions and are attractive for write intensive applications since writes do not need to access all replicas. However, they have worse performance than ROWA for the more common read-intensive applications.

Replica Allocation

Using *full replication*, every site has copies of all existing data items. This simplifies the execution of read operations but has high update costs since all sites have to perform the updates for all data items. In contrast, in *partial replication* each site has only copies of some of the data items. The advantage is that an update on a data item does not lead to costs at all sites but only at those that contain a replica of the data item. However, read operations become more challenging. First, in a wide-area setting, if no local replica is available, read operations observe higher delays since they have to access a remote replica. Furthermore, if a request has to access several objects within a single query, the query might have to access data items on different sites, leading to distributed queries. Thus, replica allocation algorithms have to decide on the placement of replicas considering issues such as communication and update costs.

Related to replica allocation is the task to adjust the replication configuration automatically and dynamically to the needs of the application. This is particularily interesting in a cluster-based configuration where sites are located in a local area network and replication is used for load distribution and fault-tolerance. In here, configuration does not only relate to the number of replicas needed, but also to the question of how to distribute load across replicas, how to accomodate different applications on the replicated data, and how to optimally use all available resources. An important issue is that advanced information systems do not only have a database system but consist of a multi-tier architecture with web-servers, application servers, and database systems that interact with each other.

Materialized views are a special form of data replication, where the data retrieved by the most typical database queries is stored in a pre-formatted form. Typically, queries can be run over materialized views as if they were base tables. Since the data is already in the format needed by the query, query processing time can be considerably reduced. However, updates on the views are usually disallowed but have to be performed directly on the base tables. Special refresh algorithms then guarantee that materialized views are updated when the base data changes.

Replication in Various Computing Environments

Replication is a fundamental technique for data management that can be applied in various computing environments. The different purposes replication can generally have in LANs (clusters) and in wide-area environments have already been discussed above.

Peer-to-peer (P2P) networks are a special form of wide-area environment. In here, each site is both client and server. For instance, each peer can provide storage space to store documents or data items that can be queried by all peers in the system. Or it provides processing capacity that can be used by other peers to perform complex calculations. In turn, it can request data items or processing capacity from other peers. A large body of research has developed algorithms that decide where to store data items and how to find them in the P2P network. Replication plays an important task for fault-tolerance, fast access, and load distribution.

Replication also plays an important role in mobile environments that differ in some fundamental ways from wired networks. Firstly, communication between mobile units and the servers on the standard network is typically slow and unreliable. Secondly, mobile devices are usually less powerful and have considerably less storage space than standard machines leading to an asymmetric architecture. Furthermore, mobile devices, such as laptops, are often disconnected from the network and only reconnect periodically. Thus, having replicas locally on the mobile device provides increased availability during disconnection periods.

Key Applications

Data replication is widely used in practice. Basically, all database vendors offer a suite of replication solutions. Additionally, replication is often implemented ad-hoc at the application layer or as a middleware layer as the need arises. Replication is used in many application domains. Below some examples are listed.

- Companies use high-availability replication solutions for their mission critical data that has to be available 24/7. Examples are banking or trading applications.
- Companies use cluster replication, ideally with autonomic behavior, in order to provide a scalable and fault-tolerant database backend for their business applications. In particular companies that do e-business with a large number of users resort to database replication to be able to obtain the required throughput. This also includes techniques such as materialized views.
- Globally operating companies often have databases located at various sites. Parts of these databases are replicated at the other locations for fast local access. Examples are companies maintaining warehouses at many locations.
- Replication of web-sites is a common technique to achieve load-balancing and fast local access. As the information shown on the web becomes more and more dynamic (i.e., it is retrieved from the database in real-time), database replication techniques need to be applied.
- Companies that have employees working off-site, such as consulting or insurance companies, use mobile replication solutions. Data replicas are downloaded to mobile units such as laptops in order to work while disconnected. Upon reconnection to the network, the replicas are reconciled with the master replica on the database server. Typically, database vendors provide specialized software in order to allow for such types of replication.
- There exist many P2P-based document sharing systems, e.g., to share music, video files. Entire file systems can be built on P2P networks.

- Data Warehouses can be seen as a special form of replication where the transactional data is copied and reformatted to be easily processed by data analysis tools.

Future Directions

Being a fundamental data management technology, data replication solutions will need to be adjusted and revisited whenever new application domains and computing environments are developed. Thus, data replication will likely be a topic that will be further explored as our IT infrastructure and our demands change.

Cross-references

▶ Autonomous Replication
▶ Consistency Models for Replicated Data
▶ Data Broadcasting, Caching and Replication in Mobile Computing
▶ Distributed Database Design
▶ Middleware Support for Database Replication and Caching
▶ One-Copy-Serializability
▶ Optimistic Replication and Resolution
▶ Partial Replication
▶ P2P Database
▶ Quorum Systems
▶ Replication
▶ Replica Control
▶ Replica Freshness
▶ Replication based on Group Communication
▶ Replication for High Availability
▶ Replication for Scalability
▶ Replication in Multi-Tier Architectures
▶ Strong Consistency Models for Replicated Data
▶ Traditional Concurrency Control for Replicated Databases
▶ WAN Data Replication
▶ Weak Consistency Models for Replicated Data

Recommended Reading

1. Alonso G., Charron-Bost B., Pedone F., and Schiper A. (eds.), A 30-year Perspective on Replication, Monte Verita, Switzerland, 2007.
2. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison Wesley, Boston, MA, 1987.
3. Breitbart Y., Komondoor R., Rastogi R., Seshadri S., and Silberschatz A. Update Propagation Protocols For Replicated Databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 97–108.
4. Budhiraja N., Marzullo K., Schneider F.B., and Toueg S. The primary-backup approach. In Distributed Systems (2nd edn.). S. Mullender (ed.). Addison Wesley, New York, NY, pp. 199–216.
5. Cabrera L.F. and Pâris J.F. (eds.). In Proc. 1st Workshop on the Management of Replicated Data, 1990.
6. Gray J., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
7. Lv Q., Cao P., Cohen E., Li K., and Shenker S. Search and replication in unstructured peer-to-peer networks. In Proc. 16th Annual Int. Conf. on Supercomputing, 2002, pp. 84–95.
8. Röhm U., Böhm K., Schek H.J., and Schuldt H. FAS - a freshness-sensitive coordination middleware for a cluster of OLAP components. In Proc. Int. Conf. on Very Large Data Bases, 2002, pp. 754–765.
9. Saito Y. and Shapiro M. Optimistic replication. ACM Comput. Surv., 37(1):42–81, 2005.
10. Satyanarayanan M., Kistler J.J., Kumar P., Okasaki M.E., Siegel E.H., and Steere D.C. Coda: a highly available file system for a distributed workstation environment. IEEE Trans. Comput., 39 (4):447–459, 1990.
11. Schneider F.B. Replication management using the state-machine approach. In Distributed Systems (2nd edn.), S. Mullender (ed.). Addison Wesley, New York, NY, 1993, pp. 169–198.
12. Sivasubramanian S., Szymaniak M., Pierre G., and van Steen M. Replication for web hosting systems. ACM Comput. Surv., 36 (3):291–334, 2004.
13. Terry D.B., Theimer M., Petersen K., Demers A.J., Spreitzer M., and Hauser C. Managing update conflicts in Bayou, a weakly connected replicated storage system. In Proc. 15th ACM Symp. on Operating System Principles, 1995, pp. 172–183.
14. Wiesmann M. and Schiper A. Comparison of database replication techniques based on total order broadcast. IEEE Trans. Knowl. Data Eng., 17(4):551–566, 2005.

Data Replication Protocols

▶ Replica Control

Data Sampling

Qing Zhang
The Australian e-health Research Center, Brisbane, QLD, Australia

Definition

Repeatedly choosing random numbers according to a given distribution is generally referred to as *sampling*. It is a popular technique for data reduction and approximate query processing. It allows a large set of data to be summarized as a much smaller data set, the *sampling synopsis*, which usually provides an estimate of the original data with provable error guarantees. One advantage of the sampling synopsis is easy and efficient. The cost of constructing such a synopsis is only *proportional to the synopsis size*, which makes the sampling complexity potentially sublinear to the size of the original data. The other advantage is that the sampling synopsis represents parts of the original data. Thus many query processing and data manipulation techniques that are applicable to the original data, can be directly applied on the synopsis.

Historical Background

The notion of representing large data sets through small samples dates back to the end of nineteenth century and has led to the development of many techniques [5]. Over the past two decades, sampling techniques have been greatly developed in various database areas, especially on query optimization and approximate query processing.

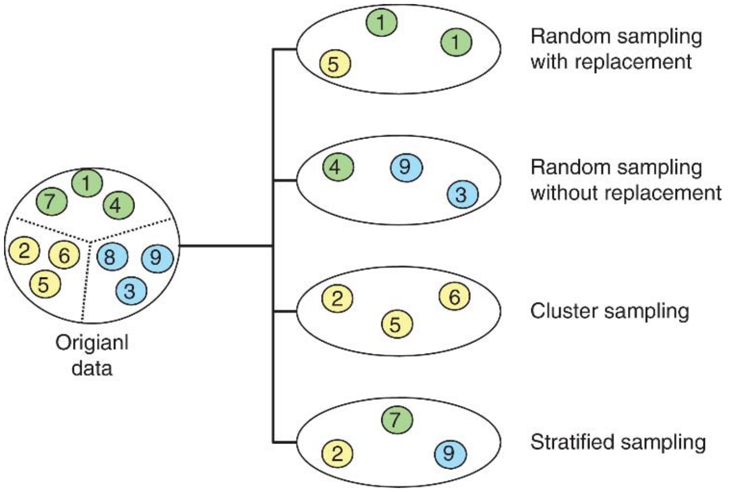
- *Query Optimization*: Query optimizer identifies an efficient execution plan for evaluating the query. The optimizer generates alternative plans and chooses the cheapest one. It uses the statistical information stored in the system catalog to estimate the cost of a plan. Sampling synopsis plays a critical role in the query optimization of a RDBMS. Some commercial products, such as DB2 and Oracle, have already adopted sampling techniques to estimate several catalog statistics. In the Heterogeneous DBMS, the global query optimizer also employs sampling techniques to estimate query plans when some local statistical information is unavailable [6].
- *Approximate Query Processing*: Sampling is mainly used to generate approximate numeric answers for aggregate queries over a set of records, such as COUNT, SUM, MAX, etc. Compared with other approximate query processing techniques, such as histogram and wavelet, sampling is easy to be implemented and efficient to generate approximate answers with error guarantees. Many prototypes on approximate query processing have adopted sampling approaches [2,3,4].

Foundations

A sampling estimation can be roughly divided into two stages. The first stage is to find a suitable sampling method to construct the sampling synopsis from the original data set. The second stage is to analyze the sampling estimator to find the characteristics (bounds and parameters) of its distribution.

Sampling Method: Existing sampling methods can be classified into two groups, the *uniform random sampling* and *biased sampling*. The uniform random sampling is a straightforward solution. Every tuple of the original data has the same probability to be sampled. Thus for aggregate queries, the estimation from samples is the expected value of the answer. Due to the usefulness of uniform random sampling, commercial DBMSs have already supports operators to collect uniform samples. However, there are some queries for which the uniform random sampling are less effective on estimation. Given a simple group-by query which intends to find the average value of different groups, a smaller group is often as important to the user as those larger groups. It is obvious that the uniform random sampling will not have enough information for the smaller group. That is why the biased sampling methods are developed in these cases. Stratified sampling, for example, is a typical biased sampling, which will be explained in detail later. The four basic sampling methods, two uniform sampling and two biased sampling, are listed below. Figure 1 shows corresponding sampling synopses generated by those methods on the sample data.

1. *Random sampling with replacement*: This method creates a synopsis by randomly drawing n of the N original data records, where the probability of drawing any record is $\frac{1}{N}$. In other words, the records that have already been drawn are not to be remembered. So the chance exists that a certain record will be repeatedly drawn in several runs.
2. *Random sampling without replacement*: This is similar to the random sampling with replacement method except that in each run the drawn record will be remembered. That is, the same record will not be chosen on subsequent runs. Although sampling without replacement appears to lead to better approximation results, sampling with replacement is significantly easier to be implemented and analyzed. Thus in practice the negligible difference between these two methods' effects makes the sampling with replacement a desirable alternative.
3. *Cluster sampling*: The N original data records are grouped into M mutually disjoint clusters. Then a


Random sampling with replacement
Random sampling without replacement
Cluster sampling
Stratified sampling
Origianl data
1
2
3
4
5
6
7
8
9

Data Sampling. Figure 1. Sampling methods (sampling synopsis size = 3).

random sampling on M clusters is obtained to form the cluster sampling synopsis. That is, the clusters are treated as sampling units so statistical analysis is done on a population of clusters.

4. *Stratified sampling*: Like the cluster sampling, the N records are grouped into M mutually disjoin clusters, called *strata*. A stratified sampling synopsis is generated through running a random sampling on each cluster. This method is especially useful when the original data has skew distribution. In this way, the cluster with smallest number of records will be sure to be represented in the synopsis.

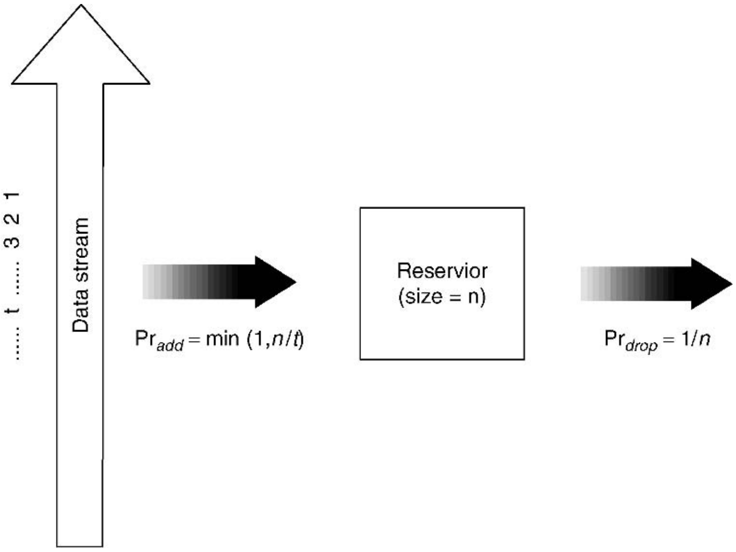
These basic sampling methods are straightforward solutions although they usually can not satisfy the error or space requirements. However, they are building blocks of more advanced methods that can either be used in certain situations or guarantee the estimation error with a confidence level.

Here is an example of a specially designed sampling method, which extends the basic random sampling method to be usable in the data stream environment. Note that the basic unbiased random sampling method requires a fixed data set with pre-defined data size. However, in a data stream environment, the size of the original data set is unknown. Thus the dynamic sampling method is required to get an unbiased sampling synopsis over the whole data stream. For this purpose, reservoir based sampling methods were originally proposed in [7].

Suppose constructing an unbiased sampling synopsis against a data stream T. A sample reservoir of n records is maintained from the stream. That is the first n records of T will be added to the reservoir for initialization. Any $t-th$ new coming record will be added to the reservoir with probability $\frac{n}{t}$. If a new record is added to the reservoir, any existing records of the reservoir will be discarded with probability $\frac{1}{n}$. Figure 2 demonstrates the construction steps of the reservoir. Finally, when all data of T has been processed, the n records of the reservoir form an unbiased random sampling synopsis of all the records of T. Similar reservoir based sampling method can also be developed for biased sampling [1].

Sampling Analysis : This stage analyzes the random variable generated by sampling methods. More specifically, it analyzes the distribution of the random variable through discovering its bound and distribution parameters. Given N records, assume that the function $f(N)$ represents an operation on the N records. Let S represent a sampling synopsis of N, and $f(S)$ is often closely related to $f(N)$ for most common operations, such as AVERAGE or MAX. Let $X = f(S)$. X is the random variable that are going to be analyzed. If $f(N)$ represents some linear aggregation functions, such as AVERAGE, X can be approximated as a normal distribution, according to the *Central Limit Theorem*. If however, $f(N)$ represents other functions, such as MAX, probabilistic bounds based on key distribution parameters, such as expectation $E(X)$ and variance $Var[X]$, need to be found. This is often quite acceptable as an alternative to characterize the entire distribution of X.

There exist a number of inequalities to estimate the probabilistic bound. These inequalities are collectively


..... t 3 2 1
Data stream
$\mathrm{Pr}_{add} = \min(1, n/t)$
Reservior
(size = n)
$\mathrm{Pr}_{drop} = 1/n$

Data Sampling. Figure 2. Random sampling with a reservoir.

known as tail bounds. Given a random variable X, if $E[X]$ is known, *Markov's Inequality* gives:

$$\forall\ a > 0, \quad \Pr(X \geq a) \leq \frac{E[X]}{a}$$

The variance of X, $Var[X]$ is defined as:

$$Var[X] = E[(X - E[X])^2]$$

A significantly stronger tail bound can be obtained by *Chebyshev's Inequality* if $Var[X]$ is known:

$$\forall\ a > 0, \quad \Pr(|X - E[X]| \geq a) \leq \frac{Var[X]}{a^2}$$

The third inequality is an extremely powerful tool called *Chernoff bounds*, which gives exponentially decreasing bounds on the tail distribution. These are derived by applying Markov's Inequality to $e^{tX}$for some well-chosen value t. Bounds derived from this approach are generally referred to collectively as Chernoff bounds. The most commonly used version of the Chernoff bound is for the tail distribution of a sum of independent 0–1 random variable. Let $X_1,\ldots,X_n$ be independent Poisson trials such that $\Pr(X_i) = p_i$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$. For $0 < \delta < 1$,

$$\Pr(|X - \mu| \geq \mu\delta) \leq 2e^{-\mu\delta^2/3}$$

Finally, an example to illustrate the different tail bounding abilities of the three inequalities is given below. Suppose estimating the synopsis size generated by a random sampling with replacement of N data records. Each record has a same probability $\frac{1}{2}$ to be sampled. Let X denote the size of the sampling synopsis. Then the size expectation is $E[X] = \frac{N}{2}$. The probabilities of the synopsis size greater than $\frac{3}{4}N$, under estimations from different inequalities, are:

$$\text{Markov's Inequality} : \Pr\left(X \geq \tfrac{3}{4}N\right) \leq \tfrac{2}{3}$$

$$\text{Chebyshev's Inequality} : \Pr\left(X \geq \tfrac{3}{4}N\right) \leq \tfrac{4}{N}$$

$$\text{Chernoff Bounds} : \Pr\left(X \geq \tfrac{3}{4}N\right) \leq 2e^{-N/24}$$

Key Applications

Query Optimization

Data sampling is one of the primary techniques used by query optimizers. In some multi-dimensional cases, it becomes the only easy and viable solution.

Approximate Query Processing

Data sampling is one of the three major data deduction techniques (the other two are histogram and wavelet) employed by approximate query processors.

Data Streaming

Sampling is a simple yet powerful method for synopsis construction in data stream.

Cross-references

- ▶ Approximate Query Processing
- ▶ Data Reduction
- ▶ Data Sketch/Synopsis
- ▶ Histogram
- ▶ Query Optimization

Recommended Reading

1. Aggarwal C.C. On biased reservoir sampling in the presence of stream evolution. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.
2. Chaudhuri S. et al. Overcoming limitations of sampling for aggregation queries. In Proc. 17th Int. Conf. on Data Engineering, 2001.
3. Ganti V., Lee M.-L., and Ramakrishnan R. ICICLES: Self-tuning samples for approximate query answering. In Proc. 28th Int. Conf. on Very Large Data Bases, 2000.
4. Gibbons P.B. and Matias Y. 1New sampling-based summary statistics for improving approximate query answers. In Proc. ACM SIGMOD int. conf. on Management of Data, 1998.
5. Kish L. Survey Sampling. Wiley, New York, xvi, 643, 1965.
6. Speegle G.D. and Donahoo M.J. Using statistical sampling for query optimization in heterogeneous library information systems. In Proc. 20th ACM Annual Conference on Computer Science, 1993.
7. Vitter J.S. Random sampling with a reservoir. ACM Trans. Math. Softw., 11(1):37–57, 1985.

Data Sketch/Synopsis

Xuemin Lin
University of New South Wales, Sydney, NSW, Australia

Synonyms

Summary

Definition

A *synopsis* of dataset D is an abstract of D. A *sketch* is also referred to an abstract of dataset D but is usually referred to an abstract in a sampling method.

Key Points

Sketch/synopsis techniques have many applications. They are mainly used for statistics estimation in query processing optimization and for supporting on-line data analysis via approximate query processing. The goal is to develop effective and efficient techniques to build a small space synopsis while achieving high precision. For instance, a key component in query processing optimization is to estimate the result sizes of queries. Many techniques [1,2] have been developed for this purpose, including histograms, wavelets, and join synopses.

In data stream applications, the space requirements of synopses/sketches are critical to keep them in memory for on-line query processing. Streams are usually massive in size and fast at arrival rates; consequently it may be infeasible to keep a whole data stream in memory. Many techniques [3] have been proposed with the aim to minimize the space requirement for a given precision guarantee. These [3] include heavy hitter, quantiles, duplicate-insensitive aggregates, joins, data distribution estimation, etc.

Cross-references

- ▶ Approximate Query Processing
- ▶ Histograms on Streams
- ▶ Wavelets on Streams

Recommended Reading

1. Alon N., Gibbons P.B., Matias Y., and Szegedy M. Tracking join and self-join sizes in limited storage. In Proc. 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 1999.
2. Gibbons P.B. and Matias Y. Synopsis data structures for massive data sets. In Proc. ACM-SIAM Symposium on Discrete Algorithms, 1999.
3. Zhang Y., Lin X., Xu J., Korn F., and Wang W. Space-efficient relative error order sketch over data streams. In Proc. 22nd Int. Conf. on Data Engineering, 2006.

Data Skew

Luc Bouganim
INRIA-Rocquencourt, Le Chesnay, France

Synonyms

Biased distribution; Non-uniform distribution

Definition

Data skew primarily refers to a non uniform distribution in a dataset. Skewed distribution can follow common distributions (e.g., Zipfian, Gaussian, Poisson), but many studies consider Zipfian [3] distribution to model skewed datasets. Using a real bibliographic database, [1] provides real-world parameters for the Zipf distribution model. The direct impact of data skew on parallel execution of complex database queries is a poor load balancing leading to high response time.

Key Points

Walton et al. [2] classify the effects of skewed data distribution on a parallel execution, distinguishing *intrinsic skew* from *partition skew*. Intrinsic skew is skew inherent in the dataset (e.g., there are more citizens in Paris than in Waterloo) and is thus called *Attribute value skew (AVS)*. Partition skew occurs on parallel implementations when the workload is not evenly distributed between nodes, even when input data is uniformly distributed. Partition skew can further be classified in four types of skew. *Tuple placement skew (TPS)* is the skew introduced when the data is initially partitioned (e.g., with range partitioning). *Selectivity skew (SS)* is introduced when there is variation in the selectivity of select predicates on each node. *Redistribution skew (RS)* occurs in the redistribution step between two operators. It is similar to TPS. Finally *join product skew (JPS)* occurs because the join selectivity may vary between nodes.

Cross-references

▶ Query Load Balancing in Parallel Database Systems

Recommended Reading

1. Lynch C. Selectivity estimation and query optimization in large databases with highly skewed distributions of column values. In Proc. 14th Int. Conf. on Very Large Data Bases, 1988, pp. 240–251.
2. Walton C.B., Dale A.G., and Jenevin R.M. A taxonomy and performance model of data skew effects in parallel joins. In Proc. 17th Int. Conf. on Very Large Data Bases, 1991, pp. 537–548.
3. Zipf G.K. Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology. Addison-Wesley, Reading, MA, 1949.

Data Sorts

▶ Data Types in Scientific Data Management

Data Standardization

▶ Constraint-Driven Database Repair

Data Storage and Indexing in Sensor Networks

Philip B. Gibbons
Intel Labs Pittsburgh, Pittsburgh, PA, USA

Definition

Sensor data can either be stored local to the sensor node that collected the data (*local storage*), transmitted to one or more collection points outside of the sensor network (*external storage*), or transmitted and stored at other nodes in the sensor network (*in-network storage*). There are trade-offs with each of these approaches, as discussed below, depending on the volume of data collected at each sensor node, the query workload, and the resource limitations of each node. Moreover, the local and in-network storage scenarios often require *in-network indexes* in order to reduce the overheads of answering queries on data stored within the sensor network. Such indexes can be classified as either *exact-match indexes* or *range indexes*.

Historical Background

External storage is in some sense the default approach for sensor networks, reflecting the common scenario in which the application is interested in all the collected sensor readings. Early work in *local storage* includes Cougar [11] and TinyDB [8]; both push SQL-style queries out to data stored at the sensor nodes. In TinyDB and in Directed Diffusion [6] (another early work) the query workload dictates which sensors are turned on. This functionality can be used for external, local, or in-network storage, depending on where these collected data get stored. Seminal work on *in-network storage* includes the work on geographic hash tables (GHTs) [10], which support exact-match indexing. The authors advocate *data-centric storage*, a class of in-network storage in which data are stored according to named attribute values. Early work on supporting range indexes for in-network storage includes DIFS [5] and DIM [7].

Foundations

External Storage, in which all the sensor readings are transmitted to collection points outside of the sensor network, has several important advantages. First, storage is plentiful outside the sensor network, so that all the data can be archived, as well as disseminated to any interested party (e.g., posted on the web). Archiving all the data is quite useful for testing out new theories and models on these historical data, and for forensic activities. Second, processing power, memory, and energy are plentiful outside the sensor network, so that queries and complex data analysis can be executed quickly and without exhausting the sensors' limited energy reserves. Finally, such data processing can be done using standard programming languages and tools (such as MatLab) that are not available on sensor nodes. On the other hand, external storage suffers the disadvantage that it incurs the costs (primarily energy, but also bandwidth and latency) of transmitting all the data to outside the network.

Local storage, in which sensor readings are stored local to the node that collected the data, avoids the costs of transmitting all the data. Instead, it incurs the costs of pushing queries into the network and returning the query answers. Queries are often flooded through the sensor network. A *collection tree* is constructed hop-by-hop from the query source (called the *root*), as follows [8]. The root broadcasts the query, and each sensor node that hears the broadcast makes the root its parent in the tree. These nodes in turn broadcast the query, and any node that hears one or more broadcasts (and is not yet in the tree) selects one of these nodes as its parent, and so on. This tree is used to collect query answers: Each leaf node sends its piece of the answer to its parent, and each internal node collects these partial answers from its children, combines them with its own piece of the answer, and sends the result to its parent. The process proceeds level-by-level up the tree to the root. Thus, the cost of pushing the query and gathering the answer can be high.

Nevertheless, the amount of data transmitted when using local storage can often be far less than when using external storage. First, queries are often long running (*continuous queries*); for such queries, the costs of query flooding and tree construction are incurred only once at the start of the query (although maintaining the tree under failures can incur some additional costs). Second, indexes can be used (as discussed below) to narrow the scope of the query to a subset of the nodes. Third, queries can be highly selective (e.g., looking for rare events such as an intruder sighting), so that most sensors transmit little or no real data. In camera sensor networks (e.g., IrisNet [4]), local filtering of images can result in query answers that are orders of magnitude smaller than the raw images. Fourth, many queries are amenable to efficient *in-network aggregation*, in which partial answers received from children can be combined into a single fixed-sized packet. For example, in a Sum query each internal node can send to its parent a single value equal to the sum of its value and the values received from its children. Finally, in sensor networks supporting queries of live data only (i.e., only the latest data are of interest), the amount of data sensed can far exceed the amount of data queried.

A key consideration when using local storage is that the amount of such storage is limited. Thus, at some point old data need to be discarded or summarized to make room for new data [2]. Moreover, the local storage is often flash memory, and hence flash-friendly techniques are needed to minimize the costs for accessing and updating locally stored data [9].

In-network storage, in which sensor readings are transmitted and stored at other nodes in the sensor network, falls in between the extremes of external storage and local storage. Caching data that passes through a sensor node during query processing is a simple form of in-network storage. As cached data become stale over time, care must be taken to ensure that the data meets the query's freshness requirements [4]. In TinyDB [8], "storage point" queries can be used to collect data satisfying a query (e.g., all temperature readings in the past 8 seconds, updated every second) at nodes within the network. In *data-centric storage* [10], data are stored according to named attribute values; all data with the same general name (e.g., intruder sightings) are stored at the same sensor node. Because data items are stored according to their names, queries can retrieve all data items associated with a target name from just a single "home" node for that name (as opposed to potentially all the nodes when using local storage). The approach relies on building and maintaining indexes on the names, so that both sensor readings and queries can be routed efficiently to the corresponding home node.

In-network indexes, in which the index is maintained inside the sensor network, are useful for both local storage and in-network storage. The goal is to map named data to the sensor node(s) that hold such data, in order to minimize the cost of answering queries. Queries using the index are guided to the sensor nodes holding the desired data. In TinyDB each internal node of a collection tree maintains a lower and upper bound on the attribute values in the subtree rooted at the node;

this index is used to restrict the query processing to only subtrees containing the value(s) of interest. In Directed Diffusion [6], queries expressing interests in some target data are initially flooded out from the query node. Paths leading to nodes that are sources of the target data are reinforced, resulting in an ad hoc routing tree from the sources back to the query node. This can be viewed as an ad hoc query-specific index.

A *geographic hash table* (GHT) [10] is an exact-match in-network indexing scheme for data-centric storage. A GHT hashes a data name (called a *key*) to a random geographic location within the sensor field. The home node for a key is the sensor node that is geographically closest to the location returned by the hash of the key. (For fault tolerance and to mitigate hot spots, data items are also replicated on nearby nodes.) *Geographic routing* is used to route sensor readings and queries to the home node. In geographic routing each node knows its geographic coordinates and the coordinates of its neighbors. Upon receiving a packet, a node forwards the packet to the neighbor closest to the home node. In this way, the packet attempts to take a shortest path to the home node. The packet can get stuck, however, in a local minimum node v such that no neighbor of v is closer to the home node than v itself. Different geographic routing schemes provide different approaches for recovering from local minima, so that the home node can always be reached. Follow-on work on GHTs (e.g., [1]) has focused on improving the practicality (efficiency, robustness, etc.) of geographic routing and hence GHTs.

In-network storage based on a GHT is less costly than local storage whenever the savings in data transmitted in querying exceed the additional costs of transmitting sensor data to home nodes. In a square sensor field of n sensors, routing to a home node takes $O(\sqrt{n})$ hops. Consider a workload where there are E event messages to be transmitted and Q queries each requesting all the events for a distinct named event type. With in-network storage based on a GHT, the total hops is $O(\sqrt{n}(Q+E))$. With local storage, the total hops is $O(Qn)$, as it is dominated by the cost to flood the query. Thus, roughly, the in-network scheme is less costly when the number of events is at most a factor of $\sqrt{n}$ larger than the number of queries. However, this is in many respects a best case scenario for in-network storage, and in general, local or external storage can often be less costly than in-network storage. For example, with a single continuous query that aggregates data from all the sensor nodes for $t \gg 1$ time periods (i.e., $E = tn$), in-network storage based on a GHT incurs $O(tn^{1.5})$ total hops while local storage with in-network aggregation incurs only $O(tn)$ total hops, as the cost is dominated by the t rounds of hops up the collection tree.

Moreover, a GHT is not well-suited to answering range queries. To remedy this, a variety of data-centric storage schemes have been proposed that provide effective range indexes [5,7,3]. DIM [7], for example, presents a technique (inspired by k-d trees) for constructing a locality-preserving geographic hash function. Combined with geographic routing, this extends the favorable scenarios for in-network storage to include also multi-dimensional range queries.

In summary, which of the external, local, or in-network storage schemes is preferred depends on the volume of data collected at each sensor node, the query workload, and the resource limitations of each node.

Key Applications

Sensor networks, Applications of sensor network data management.

Cross-references

- ▶ Ad-Hoc Queries in Sensor Networks
- ▶ Applications of Sensor Network Data Management
- ▶ Continuous Queries in Sensor Networks
- ▶ Data Acquisition and Dissemination in Sensor Networks
- ▶ Data Aggregation in Sensor Networks
- ▶ Data Compression in Sensor Networks
- ▶ Data Fusion in Sensor Networks
- ▶ In-Network Query Processing
- ▶ Sensor Networks

Recommended Reading

1. Ee C.T., Ratnasamy S., and Shenker S. Practical data-centric storage. In Proc. 3rd USENIX Symp. on Networked Systems Design & Implementation, 2006, pp. 325–338.
2. Ganesan D., Greenstein B., Estrin D., Heidemann J., and Govindan R. Multiresolution storage and search in sensor networks. ACM Trans. Storage, 1(3):277–315, 2005.
3. Gao J., Guibas L.J., Hershberger J., and Zhang L. Fractionally cascaded information in a sensor network. In Proc. 3rd Int. Symp. Inf. Proc. in Sensor Networks, 2004, pp. 311–319.
4. Gibbons P.B., Karp B., Ke Y., Nath S., and Seshan S. IrisNet: An architecture for a worldwide sensor web. IEEE Pervasive Comput, 2(4):22–33, 2003.
5. Greenstein B., Estrin D., Govindan R., Ratnasamy S., and Shenker S. DIFS: A distributed index for features in sensor

networks. In Proc. First IEEE Int. Workshop on Sensor Network Protocols and Applications, 2003, pp. 163–173.
6. Intanagonwiwat C., Govindan R., Estrin D., Heidemann J., and Silva F. Directed diffusion for wireless sensor networking. IEEE/ACM Trans. Network., 11(1):2–16, 2003.
7. Li X., Kim Y.J., Govindan R., and Hong W. Multi-dimensional range queries in sensor networks. In Proc. 1st Int. Conf. on Embedded Networked Sensor Systems, 2003, pp. 63–75.
8. Madden S.R., Franklin M.J., Hellerstein J.M., and Hong W. TinyDB: An acquisitional query processing system for sensor networks. ACM Trans Database Syst, 30(1):122–173, 2005.
9. Mathur G., Desnoyers P., Ganesan D., and Shenoy P. Capsule: An energy-optimized object storage system for memory-constrained sensor devices. In Proc. 4th Int. Conf. on Embedded Networked Sensor Systems, 2006, pp. 195–208.
10. Ratnasamy S., Karp B., Shenker S., Estrin D., Govindan R., Yin L., and Yu F. Data-centric storage in sensornets with GHT, a geographic hash table. Mobile Networks Appl., 8(4):427–442, 2003. Springer.
11. Yao Y. and Gehrke J. Query processing for sensor networks. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.

Data Stream

Lukasz Golab
AT&T Labs-Research, Florham Park, NJ, USA

Synonyms

Continuous data feed

Definition

A data stream S is an ordered collection of data items, $s_1, s_2, \ldots$, having the following properties:

- Data items are continuously generated by one or more sources and sent to one or more processing entities.
- The arrival order of data items cannot be controlled by the processing entities.

For instance, an Internet Service Provider (ISP) may be interested in monitoring the traffic on one or more of its links. In this case, the data stream consists of data packets flowing through the network. The processing entities, e.g., monitoring software, may be located directly on routers inside the ISP's network or on remote nodes.

Data streams may be classified into two types: based and derived. A base stream arrives directly from the source. A derived stream is a pre-processed base stream, e.g., an intermediate result of a query or a sub-query over one or more base streams. In the network monitoring scenario, the base stream corresponds to the actual IP packets, whereas a derived stream could contain aggregate measurements of traffic between each source and destination in a five-minute window.

Key Points

Depending upon the application, a data stream may be composed of raw data packets, relational tuples, events, pieces of text, or pieces of an XML document.

Furthermore, each data stream item may be associated with two timestamps: generation time (assigned by the source) and arrival time (assigned by the processing entity). The order in which items arrive may be different from their generation order, therefore these two timestamps may produce different orderings of the data stream.

A data stream may arrive at a very high speed (e.g., a router may process hundreds of thousands of packets per second) and its arrival rate may vary over time. Hence, a data stream may be unbounded in size. In particular, the processing entity may not know if and when the stream "ends."

Cross-references

▶ Stream-Oriented Query Languages and Operators
▶ Stream processing
▶ One-pass algorithm
▶ Stream mining
▶ Synopsis structure

Recommended Reading

1. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and issues in data streams. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 1–16.
2. Golab L. and Özsu M.T. Issues in data stream management. ACM SIGMOD Rec., 32(2):5–14, 2003.
3. Muthukrishnan S. Data streams: algorithms and applications. Found. Trends Theor. Comput. Sci., 1(2):1–67, 2005.

Data Stream Algorithm

▶ One-Pass Algorithm

Data Stream Management Architectures and Prototypes

Yanif Ahmad, Uğur Çetintemel
Brown University, Providence, RI, USA

Definition

Data stream processing architectures perform database-style query processing on a set of continuously arriving input streams. The core query executor in this type of architecture is designed to process continuous queries, rather than ad-hoc queries, by pushing inputs through a series of operators functioning in a pipelined and potentially non-blocking manner. Stream processing applications perform explicit read and write operations to access storage via asynchronous disk I/O operations. Other architectural components that differ significantly from standard database designs include the stream processor's scheduler, storage manager and queue manager.

Historical Background

Support database-style query processing for long-running applications that operate in high (data) volume environments, and impose high throughput and low latency requirements on the system. There have been several efforts from both the academic and industrial communities at developing functional prototypes of stream processing engines (SPEs), to demonstrate their usefulness and to better understand the challenges posed by data stream applications. The first general purpose relational stream processing architectures appeared from the research community in 2001–2002, while the initial industrial offerings began to appear in 2003–2004. As a historical note, non-relational approaches to stream or event processing have existed in many forms prior to the early 2000s, especially in the embedded systems and signal processing communities.

Foundations

An SPE has a radically different architecture than that of a traditional database engine. Conceptually, the architectural differences can be captured by the following key characteristics:

1. Continuous query model
2. Inbound processing model
3. Single-process model

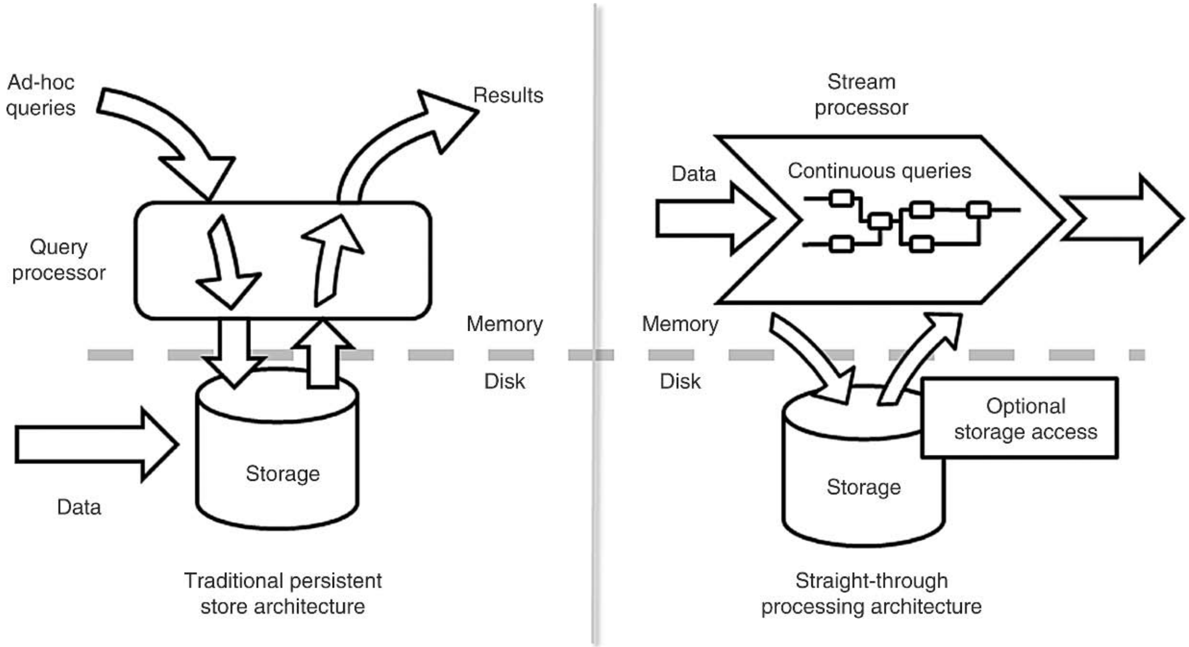
In the continuous query model, the queries execute continuously as new input data becomes available. This contrasts with the prevailing one-time query model where users (or clients) issue queries that process the available input data and produce one-time results. In other words, in the continuous model, the queries are persistent and input data is transient, whereas in the traditional model, queries are transient and input data is persistent, as illustrated in Fig. 1.

An SPE supports inbound processing, where incoming event streams immediately start to flow through the query processing operators as they enter the system. The operators process the events as they move, continuously producing results, all in main memory when possible. Read or write operations to storage are optional and can be executed asynchronously in many cases, when they are present. Inbound processing overcomes a fundamental limitation of the traditional outbound processing model, employed by all conventional database management systems, where data are always inserted into the database (usually as part of a transaction) and indexed as a first step before any processing can take place to produce results. By removing the storage from the critical path of processing, an SPE achieves significant performance gains compared to the traditional outbound processing approach.

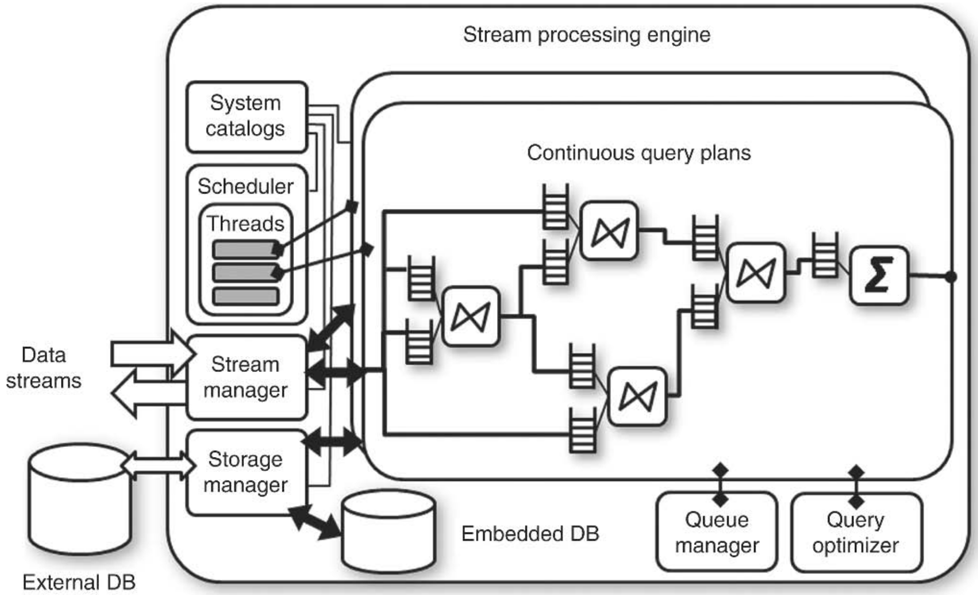
An SPE often adopts a single-process model, where all time-critical operations (including data processing, storage, and execution of custom application logic) are run in a single process space. This integrated approach eliminates high-overhead process context switches that are present in solutions that use multiple software systems to collectively provide the same set of capabilities, yielding high throughput with low latency.

The SPE prototypes developed independently in the academic community share core design principles and architectural components to implement push-based dataflows. At a high level, an SPE's core includes a query executor maintaining plans for users' queries, a queue manager and storage manager to govern memory resources and perform optional disk access and persistence, a stream manager to handle stream I/O with data sources and sinks, and a scheduler to determine an execution strategy. Figure 2 presents a diagrammatic overview of these core components.

SPEs implement continuous queries and inbound processing inside the query executor, by instantiating query plans with non-blocking operators that are capable of producing result tuples from each individual



Data Stream Management Architectures and Prototypes. Figure 1. Illustration of storage-oriented and streaming-oriented architectures. The former requires outbound processing of data, whereas the latter enables inbound (or straight-through) processing.



Data Stream Management Architectures and Prototypes. Figure 2. Architectural components of an SPE.

input tuple, or a window of tuples, depending on the operator type. This is in contrast to traditional operators that wait for relations to be scanned from disk. The query executor represents query plans as operators connected together with queues that buffer continuously-arriving inputs (as found in the Aurora and Stream prototypes [2,5]), and any pending outputs (for example Fjords in TelegraphCQ [8]), while each operator performs its processing. A queue manager is responsible for ensuring the availability of memory resources to support buffering inputs and outputs, and interacts with other system components to engage alternative processing techniques when memory availability guarantees cannot be upheld [3].

Operators may choose to access inputs, or share state with other operators, through persistent storage. Disk access is provided through a storage manager that is responsible for maintaining cursors on external tables, and for performing asynchronous read and write operations while continuing to process data streams. Advanced storage manager features include the ability to spill operator queues and state to disk under dwindling memory availability, as well as the ability to maintain approximations of streams, queues and states. SPEs typically include a stream manager component to handle interaction with the network layer as data sources transmit stream inputs typically over TCP or UDP sockets. The stream manager is additionally responsible for any data format conversions through built-in adaptors, and to indicate the arrival of new inputs to the scheduler as the new inputs are placed on the operators' queues.

An operator scheduler [5,7] is responsible for devising an execution order based on various policies to ensure efficient utilization of system resources. These policies typically gather operator cost and selectivity statistics in addition to resource utilization statistics to determine a schedule that improves throughput and latencies. While SPEs execute under a single-process model, various scheduler threading designs have been proposed to provide query processing parallelism. Finally, SPEs also include query optimizers such as load shedders [15] and adaptive plan optimizers [6], that also monitor the state of the running query in terms of statistics and other optimizer-specific monitors to dynamically and adaptively determine advantageous modifications to query plans and operator internals.

Prototypes

The key features in the architectural design of stream processors primarily arose from academic prototypes, before being extended by industrial-grade tools based on the commercialization of the academic efforts. These features are described for a subset of the prototypes below.

Aurora/Borealis: The Aurora and Borealis [2,1] projects are first- and second-generation stream processing engines built in a collaboration by Brandeis, Brown and MIT. The Aurora engine was implemented from scratch in C++, and included the basic architectural components described above to produce a single-site design. Aurora was a general-purpose engine that provided a relational operator set to be used to construct queries visually in an editor, as a workflow. This workflow-style programming paradigm (sometimes referred to as "boxes-and-arrows") differed significantly from similar research projects which focused more on providing stream-oriented language extensions to SQL.

The Aurora architecture included a multi-threaded scheduler capable of supporting tuple-trains and superbox scheduling. Aurora also supported load shedding, the concept of selectively processing inputs in the presence of excessive load due to high-rate data streams. The Aurora engine also supported embedded tables to enable operators to share state. The embedded tables were implemented as BerkeleyDB stores and the core operator set included operators capable of performing a subset of SQL queries on these tables.

The Borealis project extended the Aurora architecture for a multi-site deployment, and implemented components to address the novel challenges exposed by distributed execution. These included a decentralized catalog structure maintaining metadata for the set of deployed queries, and a distributed statistics collector and optimizer. The optimizer targeted distributed query optimizations such as spreading load across multiple machines to achieve both a balanced allocation, and resilience to changes in load, in addition to a distributed load shedding mechanism that factored in the allocation of operators to sites.

Stream: The Stream project at Stanford [5] developed a C++ implementation of a stream processing engine with a similar high-level architecture to the design described above. The novel features of the Stream architecture included its use of the Continuous Query Language (CQL) which extended SQL with DDL statements to define data streams and subsequently provided DML clauses for several types of windowing operations on these streams.

The core engine included a single-threaded scheduler that continuously executes operators based on a scheduling policy, while the operators implement non-blocking query execution through the use of queues. In addition to this basic query executor, the Stream project studied various resource management, query approximation and adaptive query processing techniques. These included memory management techniques implemented by both a scheduling policy that executes groups of operators to minimize queue sizes, and by exploiting shared synopses (window implementations) and constraints in the arrival patterns of data streams (such as bounding the arrival delay between interacting inputs). In terms of query approximation, Stream provided both a load-shedding algorithm that approximates query results to reduce CPU load, in addition to synopsis compaction techniques that reduced memory requirements at operators. Finally, Stream is capable of adapting the running query through the aid of two components, a profiler that continuously collects statistics during query execution, and a reoptimizer component that maintains both filter and join orderings based on changing selectivities.

TelegraphCQ: In contrast to the ground up design of Aurora and Stream, the TelegraphCQ project [8] at UC Berkeley developed a stream processing engine on top of the PostgreSQL open-source database. This approach allowed the reuse of several PostgreSQL components, such as the system catalogs, parser and optimizer.

TelegraphCQ is divided into two high-level components, a frontend and a backend. The frontend is responsible for client interaction such as parsing and planning queries, in addition to returning query results. The TelegraphCQ backend is a continually executing process that performs the actual query processing, and adds query plans submitted by the frontend to the set of executable objects. The backend is implemented in a multi-threaded fashion enabling processing parallelism. The query executor in the TelegraphCQ backend is principally designed to support adaptive query processing through the use of Eddies to dynamically route tuples between a set of commutative operators (thus performing run-time reordering). The executor also leans heavily on exploiting opportunities for shared processing, both in terms of the state maintained internally within operators (such as aggregates), and in terms of common expressions used by selections through grouped filters. Finally, as a result of its PostgreSQL base, TelegraphCQ investigated query processing strategies combining the use of streamed data and historical data from a persistent source.

Gigascope: The Gigascope data stream engine [10] was developed at AT&T Labs-Research to primarily study network monitoring applications, for example involving complex network and protocol analyses of BGP updates and IP packets. Gigascope supports textual queries through GSQL, a pure stream query language that is a simplified form of standard SQL. GSQL queries are internally viewed as having a two-level structure, where queries consist of high-level and low-level operators comprising a graph-structured program, depending on the optimization opportunities determined by a query optimizer. Low-level operators are extremely lightweight computations to perform preliminary filtering and aggregation prior to processing high-level operators, and in some cases these low-level operators may be performed on the network cards of the machines present in the network monitoring application. Gigascope queries are thus compiled into C and C++ modules and linked into a run-time system for highly-efficient execution. Gigascope also investigated the blocking properties of both high- and low-level operators and developed a heartbeat mechanism to effectively alleviate operators' memory requirements.

Nile: The Nile stream processing engine was developed at Purdue on top of the Predator [14] object-relational DBMS. Nile implements data streams as an enhanced datatype in Predator and performs stream processing with the aid of a stream manager component. This stream manager is responsible for buffering input streams and handing data to the execution engine for query processing. Nile uses a separate thread for its stream manager, and performs round-robin scheduling for processing new inputs on streams.

In addition to the basic stream processing engine design, the Nile project investigated various query correctness issues and optimization opportunities arising in the stream processing context. This included studying scheduling strategies to exploit resource sharing amongst queries, for example sharing windowed join operators between multiple queries, and pipelining mechanisms based on strategies to expire tuples in multiple windows.

System S: The System S [12] project is a recent endeavor at IBM Research investigating large-scale distributed stream processing systems focusing primarily on analytical streaming applications through the use of data mining techniques. System S processes data streams with a dataflow-oriented operator network consisting of processing elements (PEs) that are distributed across a set of processing nodes (PNs) and communicate through a transport component known as the data fabric. Some of the prominent architectural features of System S include the design and implementation of streaming analytic operators, including clustering and decision-tree based algorithms, and appropriate resource management algorithms to support these types of operators, such as a variety of load shedding and diffusion algorithms. System S also leverages data-parallelism through a content-based load partitioning mechanism that spreads the processing of an input or intermediate stream across multiple downstream PEs.

Key Applications

Stream processing architectures have been motivated by, and used in, several domains, including:

- Financial services: automated trading, market feed processing (cleaning, smoothing, and translation), smart order routing, real-time risk management and compliance (MiFID, RegNMS)
- Government and Military: surveillance, intrusion detection and infrastructure monitoring, battlefield command and control
- Telecommunications: network management, quality of service (QoS)/service level agreement (SLA) management, fraud detection
- Web/E-business: click-stream analysis, real-time customer experience management (CEM)

URL to Code

Borealis: http://www.cs.brown.edu/research/borealis/public/

Stream: http://infolab.stanford.edu/stream/code/

Cross-references

▶ Continuous Query
▶ Data Stream
▶ Stream-oriented Query Languages and Operators
▶ Stream Processing
▶ Streaming Applications
▶ Windows

Recommended Reading

1. Abadi D., Ahmad Y., Balazinska M., Çetintemel U., Cherniack M., Hwang J.-H., Lindner W., Maskey A.S., Rasin A., Ryvkina E., Tatbul N., Xing Y., and Zdonik S. The design of the Borealis stream processing engine. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005.
2. Abadi D.J., Carney D., Çetintemel U., Cherniack M., Convey C., Lee S., Stonebraker M., Tatbul N., and Zdonik S. Aurora: A new model and architecture for data stream management. VLDB J., 2003.
3. Arasu A., Babcock B., Babu S., McAlister J., and Widom J. Characterizing memory requirements for queries over continuous data streams. ACM Trans. Database Syst., 29 (1):162–194, 2004.
4. Babcock B., Babu S., Datar M., Motwani R., and Thomas D. Operator scheduling in data stream systems. VLDB J., 13 (4):333–353, 2004.
5. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and issues in data stream systems. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002.
6. Babu S., Motwani R., Munagala K., Nishizawa I., and Widom J. Adaptive ordering of pipelined stream filters. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 407–418.
7. Carney D., Çetintemel U., Rasin A., Zdonik S.B., Cherniack M., and Stonebraker M. Operator scheduling in a data stream manager. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 838–849.
8. Chandrasekaran S., Deshpande A., Franklin M., and Hellerstein J. TelegraphCQ: Continuous dataflow processing for an uncertain world. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
9. Chen J., DeWitt D.J., Tian F., and Wang Y. Niagaracq: A scalable continuous query system for internet databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 379–390.
10. Cranor C.D., Johnson T., Spatscheck O., and Shkapenyuk V. Gigascope: a stream database for network applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 647–651.
11. Gehrke J. (ed.) Data stream processing. IEEE Data Eng. Bull., 26 (1), 2003.
12. Gedik B., Andrade H., Wu K.-L., Yu P.S., and Doo M. SPADE: The Systems S Declarative Stream Processing Engine. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2008.
13. Golab L. and Özsu M.T. Issues in data stream management. ACM SIGMOD Rec., 32(2):5–14, 2003.
14. Hammad M.A., Mokbel M.F., Ali M.H., Aref W.G., Catlin A.C., Elmagarmid A.K, Eltabakh M.Y., Elfeky M.G., Ghanem T.M., Gwadera R., Ilyas I.F., Marzouk M.S., and Xiong X. Nile: a query processing engine for data streams. In Proc. 20th Int. Conf. on Data Engineering, 2004, p. 851.
15. Tatbul N., Çetintemel U., Zdonik S.B., Cherniack M., and Stonebraker M. Load shedding in a data stream manager. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 309–320.

Data Stream Processing

▶ Stream Processing

Data Suppression

▶ Data Compression in Sensor Networks

Data Swapping

► Data/Rank Swapping

Data Time

► Valid Time

Data Tracking

► Data Provenance

Data Transformation

► Data Exchange

Data Translation

► Data Exchange

Data Types for Moving Objects

► Spatio-Temporal Data Types

Data Types in Scientific Data Management

Amarnath Gupta
University of California San Diego, La Jolla, CA, USA

Synonyms

Data sorts; Many sorted algebra; Type theory

Definition

In mathematics, logic and computer science, the term "type" has a formal connotation. By assigning a variable to a type in a programming language, one implicitly defines constraints on the domains and operations on the variable. The term "data type" as used in data management derives from the same basic idea. A data type is a specification that concretely defines the "structure" of a data variable of that type, the operations that can be performed on that variable, and any constraints that might apply to them. For example, a "tuple" is a data type defined as a finite sequence (i.e., an ordered list) of objects, each of a specified type; it allows operations like "projection" popularly used in relational algebra.

In science, the term "data type" is sometimes used less formally to refer to a kind of scientific data. For example, one would say "gene expression" or "4D surface mesh of a beating heart" is a data type.

Foundations

Commonly Used Data Types in Science Applications

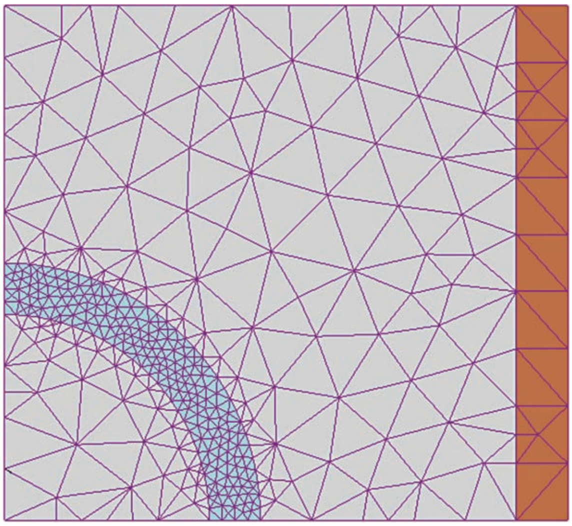
There is a very large variety of data types used in scientific domains. The following data types are commonly used in several different scientific disciplines.

Arrays Multidimensional arrays are heavily used in many scientific applications; they not only serve as natural representation for many kinds of scientific data, but they are supported by programming languages, object relational databases, many computational software libraries, as well as data visualization routines. The most common operation on arrays is index-based access to data values. However, more complex (and useful) operations can be defined on arrays. Marathe and Salem [6,7] defined an algebra on multidimensional arrays where a cell may contain a vector of values. The algebra derives from nested relational algebra, and allows one to perform value-based relational queries on arrays. Arrays are a very general data type and can be specialized with additional semantics. Baumann [1] defined a somewhat different array algebra for modeling spatiotemporal data for a system called RasDaMan. Reiner et al. [10] present a storage model for large scale arrays.

Time-Series Temporal data is a very important class of information for many scientific applications. Time-series data is a class of temporal data where the value of a variable may change with a roughly regular interval. On the other hand, the salary history of an employee is temporal data but not necessarily time-series data because the change in salary can happen at arbitrary frequencies. Data from any sensors (temperature, seismic, strain gages, electrocardiograms and so on) come in the form of a stream of explicitly or implicitly time-stamped sequence numbers (or characters). Time series data collection and storage is based on granularity, and often different data are collected at different granularity that need to be queries together [8]. All database management systems assume a discrete time line of various granularities. Granularity is a partition of a time line, for instance, years, days, hours, microseconds and so forth. Bettini et al have illustrated a formal notion of time granularities [2]. An interesting class of operations on time-series data is similarity between two time-series data. This operation can become complicated because one time series data can be recorded at a different time resolution than another and may show local variations, but still have overall similarity in the shape of the data envelope. This has prompted the investigation of efficient operators to perform this similarity. Popivanov and Miller [9] for example, has developed a measure of time-series similarity based on wavelet decomposition.

Finite Element Meshes Numerical modeling of a physical system is fundamental to many branches of science. A well known technique in this domain is called finite element analysis where a continuous domain (e.g., a 3D terrain) is partitioned into a mesh, and variables are recorded over the nodes of this mesh or regions covering multiple cells of the mesh. Assume that Fig. 1 shows the distribution of electric potential over a continuous surface. Every node of the mesh will have a (positive or negative) charge value, while a variable like "zero charge region" will be defined over regions of the mesh. Figure 1 also illustrates that mesh is not always regular – a region with a higher variation of data values will be partitioned into a finer mesh than a region will less variation.

Finite element meshes are used in many modeling as well as visualization software. In cases, where the size of the mesh is very large, and complex manipulation of data (like repartitioning based on some conditions) is needed over the elements of the mesh, the existing software do not offer robust and scalable solutions. Recently, the CORIE system [5] has developed a systematic approach to modeling a general data structure they call a *gridfield* to handle finite element meshes, and an algebra for manipulating arbitrary gridded datasets together with algebraic optimization techniques to improve efficiency of operations.



Data Types in Scientific Data Management. Figure 1. A finite element mesh.

Graphs Like arrays, graphs form a ubiquitous data type used in many scientific applications. Eckman and Brown [4] describes the use of graphs in molecular and cell biology. In their case, graphs denote relationships between biomolecular entities (A and B) that constitute molecular interaction networks, representing information like A is similar to B, A interacts with B, A regulates the expression of B, A inhibits the activity of B, A stimulates the activity of B, A binds to B and so forth. Operators on the graph data type include those that extract a subgraph from a large graph, compare one graph to another, transform one graph to another, decompose a graph into its nodes and edges, compute the intersection, union, or disjunction of two graphs, compute structural derivatives such as transitive closure and connected components and so on. In chemistry, data mining techniques are used to find most frequent subgraphs of a large number of graphs. Graphs play a dominant role in social sciences where social network analysts are interested in the analysis of the connectivity structure of the graphs. A class of operations of interest centers around the notion of aggregate properties of the graph structure. One such property is centrality, a quantity that measures for each node in a graph a value that denotes how well the node is connected to the rest of the nodes in the graph. This class of measures have been investigated in the context of data mining [11] where the task was to find the most likely subgraph "lying between" a set of query-defined nodes in the graph. While there are many alternate definitions of centrality, it should be clear that computing these aggregate values on the fly requires a different kind of data representation and operators than the previous case, where traversal and subgraph operations were dominant.

Some Basic Issues about Scientific Data Types

While database systems do not offer extensive support for scientific data types, there are many specialized software libraries that do, and hence are widely used by the scientific community. This leads to a fundamental problem as observed by the authors of [5]. On the one hand, the performance of SQL queries for manipulating large numeric datasets is not competitive with specialized tools. For example, database extensions for processing multidimensional discrete data can only model regular, rectilinear grids (i.e., arrays). On the other hand, specialized software products such as visualization software libraries are designed to process arbitrary gridded datasets efficiently. However, no algebra has been developed to simplify their use and afford optimization. In almost all cases, these libraries are data dependent – physical changes to data representation or organization break user programs. This basic observation about type specific scientific software holds for almost all of scientific data types. This calls for future research in developing storage and an algebraic manipulation for scientific data type as well as for an effort to incorporate in these techniques in scientific data management systems.

A second basic problem regarding scientific data types arises from the fact that the same data can be viewed differently for different forms of analysis and thus need to support multiple representations and storage or indexes. Consider the data type of character sequences often used in genomic studies. If S is a sequence, it is common to determine is S is "similar to" another sequence T, where T may have additional characters and missing characters with respect to S. It has been shown that a suffix tree like representation of sequences is suitable for operations of this category. However, in biology, scientists are also interested in an arbitrary number of subsequences of on the same sequences like S to which they would assign an arbitrary number of properties (called "annotations" in biology) to each subsequence. Finding similar subsequences is not a very common operation in this case. The focus is rather on interval operations like finding all subsequences overlapping a given interval that satisfies some conditions on their properties, and on finding the 1D spatial relationships among subsequences that satisfy some given properties. These operations possibly require a different storage and access structure such as an interval tree. Since a scientific application both kinds of operations would be necessary, it becomes important for the data management system to handle the multiplicity of representations and operations so that the right representations can be chosen as run time for efficient access.

Key Applications

Bioinformatics, cheminformatics, engineering databases.

Cross-references

- ► Graph Data Management in Scientific Applications
- ► Mining of Chemical Data
- ► Storage of Large Scale Multidimensional Data

Recommended Reading

1. Baumann P. A database array algebra for spatio-temporal data and beyond. In Proc. Fourth Int. Workshop on Next Generation Information Technologies and Systems, 1999, pp. 76–93.
2. Bettini C., Jajodia S., and Wang S.X. Time Granularities in Database, Data Mining, and Temporal Reasoning. Springer, 2000.
3. Borgatti S.P. and Everett M.G. A graph-theoretic perspective on centrality. Soc. Netw., 28(4):466–484, 2006.
4. Eckman B.A. and Brown P.G. Graph data management for molecular and cell biology. IBM J. Res. Dev., 50(6):545–560, 2006.
5. Howe B. and Maier D. Algebraic manipulation of scientific data sets. VLDB J., 14(4):397–416, 2005.
6. Marathe A.P. and Salem K. A language for manipulating arrays. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997, pp. 46–55.
7. Marathe A.P. and Salem K. Query processing techniques for arrays. ACM SIGMOD Rec., 28(2):323–334, 1999.
8. Merlo I., Bertino E., Ferrari E., Gadia S., and Guerrini G. Querying multiple temporal granularity data. In Proc. Seventh Int. Conf. on Temporal Representation and Reasoning, 2000, pp. 103–114.
9. Popivanov I. and Miller R.J. Similarity search over time-series data using wavelets. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 212–221.
10. Reiner B., Hahn K., Höfling G., and Baumann P. Hierarchical storage support and management for large-scale multidimensional array database management systems. In Proc. 13th Int. Conf. Database and Expert Syst. Appl., 2002, pp. 689 –700.
11. Tong H. and Faloutsos C. Center-piece subgraphs: problem definition and fast solutions. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006, pp. 404–413.

Data Types: Image, Video, Pixel, Voxel, Frame

▶ Biomedical Image Data Types and Processing

Data Uncertainty Management in Sensor Networks

Sunil Prabhakar[1], Reynold Cheng[2]
[1]Purdue University, West Lafayette, IN, USA
[2]The University of Hong Kong, Hong Kong, China

Synonyms

Imprecise data; Probabilistic data; Probabilistic querying

Definition

Data readings collected from sensors are often imprecise. The uncertainty in the data can arise from multiple sources, including measurement errors due to the sensing instrument and discrete sampling of the measurements. For some applications, ignoring the imprecision in the data is acceptable, since the range of the possible values is small enough not to significantly affect the results. However, for others it is necessary for the sensor database to record the imprecision and also to take it into account when processing the sensor data. This is a relatively new area for sensor data management. Handling the uncertainty in the data raises challenges in almost all aspects of data management. This includes modeling, semantics, query operators and types, efficient execution, and user interfaces. Probabilistic models have been proposed for handling the uncertainty. Under these models, data values that would normally be single values are transformed into groups of data values or even intervals of possible values.

Historical Background

The management of uncertain data in database management systems is a relatively new topic of research, especially for attribute-level uncertainty. Earlier work has addressed the case of tuple-level uncertainty and also node-level uncertainty for XML data. The earliest work on attribute-level uncertainty is in the area of moving object databases. In order to reduce the need for very frequent updates from moving objects, the frequency of the updates is reduced at the expense of uncertainty in the location of the object (in the database). For example, the notion of a dead-reckoning update policy allows an object to not report updates as long as it has not moved by more than a certain threshold from its last update.

In most of the earlier works, the use of probability distributions of values inside an uncertainty interval as a tool for quantifying uncertainty was not considered. Further, discussions of queries on uncertain data were often limited to the scope of aggregate functions or range queries. A model for probabilistic uncertainty was proposed for moving-objects and later extended to general numeric data for sensors in [2]. A probabilistic data model for data obtained from a sensor network was described in [6]. Past data are used to train the model through machine-learning techniques, and obtain information such as data correlation, time-varying functions of probability distributions, as well as how probability distributions are updated when new sensor values are acquired. Recently, new relational models have been proposed to manage uncertain data. These projects include MauveDB [9], Mystiq [6], Orion [15], and Trio [1]. Each of these projects aims to develop novel database systems for handling uncertain data.

Foundations

Modeling Uncertainty

Uncertainty in sensor data is often the result of either inherent limitations in the accuracy with which the sensed data is acquired or limitations imposed by concerns such as efficiency and battery life. Consider for example, a moving object application that uses GPS devices to determine the locations of people as they move about. Although GPS accuracy has improved significantly, it is well known that the location reported by a GPS sensor is really an approximation – in fact, the actual location is likely to be distributed with a Gaussian probability distribution around the reported location. This is an example of uncertainty due to the limitation of the measurement instrument.

Since most sensors are powered by batteries that can be quickly depleted, most sensor applications take great pains to conserve battery power. A common optimization is to not measure and transmit readings continuously. Instead, the data are sampled at some reasonable rate. In this case the exact values are only known at the time instances when samples are taken.

Between samples, the application can only estimate (based on the earlier samples) the values. For certain sensors, the battery overhead for taking certain types of measurements is much lower than that for others. Furthermore, the cheaper readings are correlated with more expensive reading. This allows the sensor to estimate the costlier reading by taking a cheaper reading and exploiting the correlation. However the estimate is not exact, which introduces some uncertainty.

Even when sensor readings are precise and frequently sampled, uncertainty can creep in. For example, if a given sensor is suspected of being faulty or compromised, the application may only partially trust the data provided by the sensor. In these cases, the data are not completely ignored but their reliability can be reduced. Alternatively, sensor input may be processed to generate other information – e.g., face detection on video data from a sensor. Post processing methods may not yield certain matches – the face detection algorithm may have a known degree of error or may give a degree of confidence with which it has detected a face (or a given person). In these cases, the unreliability of the raw or processed sensor data can be captured as uncertain data.

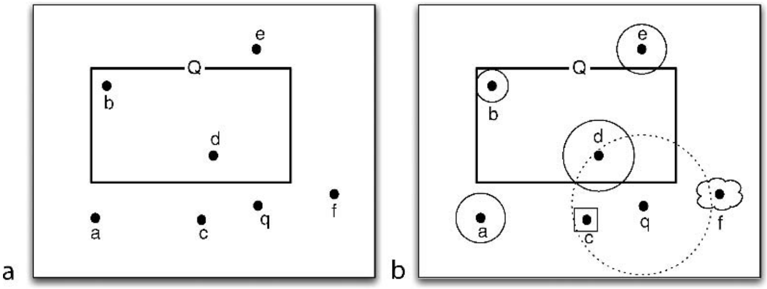
Each of these examples shows that sensor readings are not precise. Instead of data having a definite discrete value, data has numerous alternative values, possibly with associated likelihood (probabilities). The types of uncertainty in sensor data can be divided into two categories:

- *Discrete* uncertainty. Instead of a single value, a data item could take on one out of a set of alternative values. Each value in this set may further be associated with a probability indicating the likelihood of that particular value being the actual value.
- *Continuous* uncertainty. Instead of a single value, a data item can take on any one value within an interval. In addition, there may be an associated probability density function (pdf) indicating the distribution of probabilities over this interval.

In each of these cases, the total probability may or may not total to 1 for each data item. Several models for handling probabilistic data based upon the relational data model have been proposed in the literature. Most of these models can only handle discrete data wherein each alternative value for a given data time is stored in the database along with its associated probability. Extra rules are imposed over these records to indicate that only one of the alternative values for a given data time will actually occur. The Orion model is explicitly designed for handling continuous uncertainty. Under this model, uncertain attributes can be expressed as intervals with associated pdfs or as a discrete set. Representing probabilities symbolically as pdfs instead of enumerating every single alternative allows the model to handle continuous distributions.

Queries

As data becomes imprecise, there is a direct impact on the nature of query results. Figure 1 shows an example of points in two-dimensional space, a range query (Q), and a nearest-neighbor query (q) with two cases: (i) with no uncertainty; and (ii) with different types of uncertainty for different objects. Consider the two-dimensional range query Q shown in Fig. 1a. The result of the query are the identities of those points that fall within the range of the query – Points b and d in this example. If the data is imprecise (as in Fig. 1b), the data consist of regions of space (and possibly with associated probability distributions). Some of these



Data Uncertainty Management in Sensor Networks. Figure 1. A two-dimensional example: (a) exact points with no uncertainty; (b) points with uncertainty.

regions may clearly lie outside the query region and the corresponding moving objects are thus excluded from the answer (e.g., Point a). Those that lie completely within the query region are included in the answer, as in the case of precise point data (e.g., Point b). However, those objects that partially overlap the query region represent points that may or may not actually be part of the query answer (Points d and e). These points may be reported as a special subset of the answer. In [14] Future Temporal Logic (FTL) was proposed for processing location-based queries over uncertain data with no probability information. Thus an object is known to be located somewhere within a given spatial region. Queries are augmented with either MUST or MAY keywords. With the MUST keyword, objects that have even a small chance of not satisfying a query are not included in the results. On the other hand, with the MAY keyword, all objects that have even a remote chance of satisfying a query are included. FTL therefore provides some qualitative treatment for queries over uncertain data.

With the use of probability distributions, it is possible to give a more quantitative treatment to queries over uncertain data. In addition to returning the query answers, probability of each object satisfying the query can be computed and reported. In order to avoid reporting numerous low probability results, queries can be augmented with a *probability threshold*, τ. Only those objects that have a probability greater than τ of satisfying the query are reported. This notion of probabilistic queries was introduced in [2]. Most work on uncertain data management gives a quantitative treatment to queries. It should be noted that the MUST and MAY semantics can be achieved by choosing the threshold to be 1 or 0, respectively.

An important issue with regards to queries over uncertain data is the semantics of the query. What exactly does it mean to execute an arbitrary query over uncertain data? Most researchers have adopted the well-established *possible worlds* semantics (PWS) [10]. Under PWS, a database with uncertain (probabilistic) data consists of numerous probabilistic events. Depending upon the outcome of each of these events, the actual database is one out of an exponential number of possible worlds. For example, consider a single relation with two attributes: Sensor_id and reading. Assume there is a single tuple in this table, with Sensor_id S_1, and an uncertain reading which could be 1 with probability 0.3 and 2 with probability 0.7. This uncertain database consists of a single event, and there are two possible worlds: in one world (W_1), the relation consists of the single tuple $<S_1, 1>$; in world W_2, the relation consists of the single tuple $<S_1, 2>$. Furthermore, the probability of W_1 is 0.3 and that of W_2 is 0.7. In general, with multiple uncertain events, each world corresponds to a given outcome of each event and the probability of the world is given by the product of the probabilities of each event that appears in the world. It should be noted that there is no uncertainty in a given world. Each world looks like a regular database relation.

Under PWS, the semantics of a query are as follows. Executing a query over an uncertain data is conceptually composed of three steps: (i) Generate all possible worlds for the given data with associated probabilities; (ii) execute the query over each world (which has no uncertainty); and (iii) Collapse the results from all possible worlds to obtain the uncertain result to the original query. While PWS provides very clean semantics for any query over an uncertain database, it introduces challenges for efficient evaluation. First, if there is continuous uncertainty in the data, then there are an infinite number of possible worlds. Even when there are a finite number of possible worlds, the total number is exponential in the number of events. Thus it is impractical to enumerate all worlds and execute the query over each one. Techniques to avoid enumerating all worlds while computing the query correctly were proposed in [6]. They showed that there is a class of *safe* queries over uncertain data which can be computed using query plans similar to those for certain data.

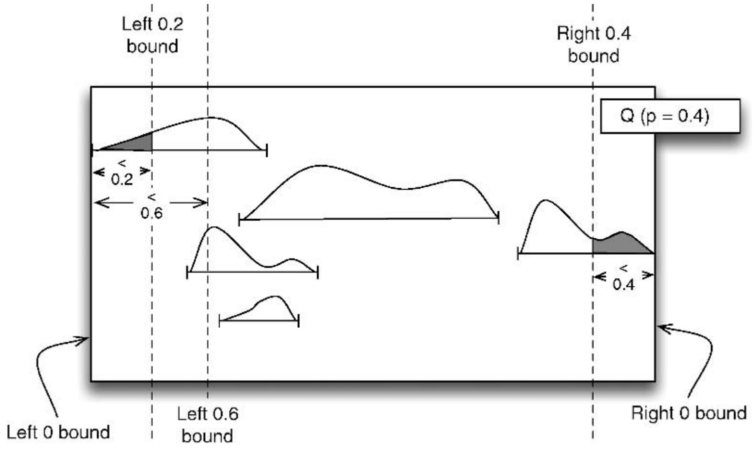
Implementation

With the goal of supporting PWS over uncertain data, systems that support uncertainty need to define probabilistic versions of database operators such as selection, projection, cross products, and comparison operators. Typically this involves operations over the probability distributions of the data, and tracking dependencies that are generated as a result of processing. Efficient management of dependencies between derived data is among the greatest challenges for uncertain data management. The base data in an uncertain database are assumed to be independent (with the exception of explicit dependencies that are expressed in the base data). However, as these data are used to produce other data, the derived data may no longer be independent of each other [6]. These dependencies affect the correct evaluation of query operators. To correctly handle dependencies, it is necessary to track them. Thus the model has to be augmented to store not only the data, but also dependencies among them. In the Trio system this information is called *Lineage*, the Orion model calls it *History*, and the MauveDB model handles dependencies using factor tables. As data is processed multiple times, the size and complexity of this dependency information can grow significantly. Efficient handling of this information is currently an active area of research.

Query processing algorithms over uncertain data have been developed for range queries [13], nearest-neighbor queries [2,11], and skyline queries [12]. Efficient join algorithms over uncertain data have been proposed in [3]. Despande et al. [7] studied the problem of answering probabilistic queries over data streams. They proposed algorithms to return results with minimum resource consumption. In [5], Cormode et al. proposed space- and time-efficient algorithms for approximating complex aggregate queries over probabilistic data streams. For queries that cannot be correctly processed using these modified operators and safe query plans, one alternative is to use approximation techniques based upon sampling. Samples of possible worlds can be drawn using the probabilities of the various events that make up the uncertain database. The query is then executed on these sample worlds and the results are aggregated to obtain an approximation of the true answer.

Indexing Indexing is a well known technique for improving query performance. Indexing uncertain data presents some novel challenges. First, uncertain data do not have a single value as is the case for traditional data. Consequently indexes such as B+-trees (and also hash indexes, since hashing requires exact matches) are inapplicable. By treating the uncertain intervals (regions) as spatial data, it is possible to use spatial indexes, such as R-trees or interval indexes, over uncertain attributes. These indexes can provide pruning based upon the extent and location of the uncertainty intervals alone. However, these index structures do not consider probability information, and are therefore incapable of exploiting probability for better evaluation. This is especially true in the case for probabilistic threshold queries.

There has been some recent work on developing index structures for uncertain data [4,11,13]. These index structures take the probability distribution of the underlying data into account. In particular, the *Probability Threshold Index* (PTI), is based on the modification of a one-dimensional R-tree. Each entry in this R-tree variant is augmented with multiple Minimum Bounding Rectangles (MBRs) to facilitate pruning. The extra MBRs are called *x*-bounds. Consider a one-dimensional data set. An MBR can been viewed as a pair of bounds: a left bound that is the right-most line that lies to the left of every object in the given node; and a right bound that is the left-most line that lies to the right of every object in the given node. The notion of *x*-bounds is similar, except that a left-x-bound is the right-most line that ensures that no object in the given node has a probability greater than *x* of lying to the left of this bound. The right-x-bound is similarly defined. Figure 2 shows an example of these bounds. Using these



Data Uncertainty Management in Sensor Networks. Figure 2. An example of *X*-Bounds for PTI.

bounds it is possible to achieve greater pruning as shown by the range query in the figure. This query has a threshold bound of 0.4. Even though the query intersects with overall MBR, using the right-0.4-bound it is clear that there is no need to visit this subtree for this query. Since the query does not cross the right-0.4-bound, there can be no objects under this node that have a probability greater than 0.4 of overlapping with the query.

Key Applications

Uncertainty in sensor data is found in virtually all applications of sensors. For many applications, however, it may be acceptable to ignore the uncertainty and treat a given value as a reasonable approximation of the sensor reading. For others, such approximations and the resulting errors in query answers are unacceptable. In order to provide correct answers for these applications it is necessary to handle the uncertainty in the data. Examples include location-based services and applications that introduce uncertainty in order to provide some degree of privacy.

Future Directions

Work on the problem of handling uncertain data in sensor databases has only just begun. Much remains to be done. A long-term goal of several current projects is the development of a full-fledged database management system with native support for uncertain data as first-class citizens. Examples of current systems include Orion, MauveDB, Mystiq, and Trio. Immediate steps in building such systems include the development of query optimization techniques. This includes cost estimation methods, query plan enumeration techniques, and approximate query evaluation methods. In addition, an important facet of system development is the user interface. Interesting issues for user interfaces include: How do users make sense of the probabilistic answers? How do they input probabilistic data and pose queries? Are new query language constructs needed? Should the probabilistic nature of the data be hidden from the user or not?

Cross-references

- ▶ Data Storage and Indexing in Sensor Networks
- ▶ Location-Based Services
- ▶ Moving Objects Databases and Tracking
- ▶ Probabilistic Databases
- ▶ R-Tree (and family)

Recommended Reading

1. Benjelloun O., Sarma A.D., Halevy A., and Widom J. ULDBs: databases with uncertainty and lineage. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 953–964.
2. Cheng R., Kalashnikov D., and Prabhakar S. Evaluating probabilistic queries over uncertain data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
3. Cheng R., Singh S., Prabhakar S., Shah R., Vitter J., and Xia Y. Efficient join processing over uncertain data. In Proc. ACM 15th Conf. on Information and Knowledge Management, 2006.
4. Cheng R., Xia Y., Prabhakar S., Shah R., and Vitter J. Efficient indexing methods for probabilistic threshold queries over uncertain data. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
5. Cormode G. and Garofalakis M. Sketching probabilistic data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 143–154.
6. Dalvi N. and Suciu D. Efficient query evaluation on probabilistic databases. In Proc. 30th Int. Conf. on Very Large Data Bases. 2004.
7. Despande A., Guestrin C., Hong W., and Madden S. Exploiting correlated attributes in acquisitional query processing. In Proc. 21st Int. Conf. on Data Engineering, 2005.
8. Deshpande A., Guestrin C., Madden S., Hellerstein J., and Hong W. Model-driven data acquisition in sensor networks. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
9. Deshpande A. and Madden S. MauveDB: supporting model-based user views in database systems. In Proc. ACM SIGMOD Int. Conf. Management of Data. 2006, pp. 73–84.
10. Halpern J.Y. Reasoning about uncertainty. MIT, Cambridge, USA, 2003.
11. Ljosa V. and Singh A. ALPA: indexing arbitrary probability distributions. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
12. Pei J., Jiang B., Lin X., and Yuan Y. Probabilistic skylines on uncertain data. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
13. Singh S., Mayfield C., Prabhakar S., Shah R., and Hambrusch S., Indexing uncertain categorical data. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
14. Sistla P.A., Wolfson O., Chamberlain S., and Dao S. Querying the uncertain positions of moving objects. Temporal databases: research and practice 1998.
15. The Orion Uncertain Database Management System. Available at: http://orion.cs.purdue.edu/

Data Utility Measures

▶ Information Loss Measures

Data Visualizations

▶ Dense Pixel Displays

Data Visualization

Hans Hinterberger
ETH Zurich, Zurich, Switzerland

Synonyms

Graphic representation of data; Information visualization

Definition

Data Visualization: (i) Interpreting information in visual terms by forming a mental picture based on data. (ii) Applying suitable methods to put data into visible form.

This definition is consistent with the Oxford English Dictionary definitions of 'data': Facts, esp. numerical facts, collected together for reference or information and of 'visualization': (i) The action or fact of visualizing; the power or process of forming a mental picture or vision of something not actually present to the sight; a picture thus formed. (ii) The action or process of rendering visible.

The first part of the definition refers to the human cognitive activity of forming a mental picture, independent of how something is represented. If this is the only activity of interest, then the term 'information visualization' is more commonly used. Similarly, 'data visualization' is often reduced to the second part of the definition.

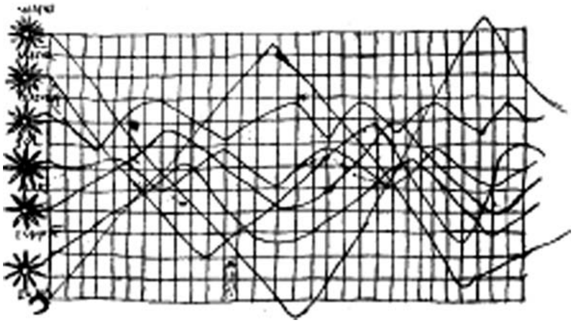
Some authors explicitly include the computer and cognition in their definition of visualization: *The use of computer supported, interactive, visual representations of data to amplify cognition* [1]. Others emphasize how data visualization differs from information visualization: *data visualization is for exploration, for uncovering information, as well as for presenting information. It is certainly a goal of data visualization to present any information in the data, but another goal is to display the raw data themselves, revealing the inherent variability and uncertainty* [16].

Historical Background

Up to the 15th Century. Over eight thousand year old maps, carved in stone, suggest that the visualization of information is as old as civilization (Fig. 1). The earliest known data visualization, a time series plot, depicting the changing values of several planets' positions, is estimated to have appeared in the tenth century (Fig. 2). In the middle of the fourteenth century, Nicole Oresme introduced an early form of coordinate graphing. He marked points in time along a horizontal line and for each of these points he drew a bar whose length represented the object's velocity at that moment.

1500–1800. Meteorological maps, showing the prevalence of winds on a geographical map, date back to 1686. In 1782, Marcellin du Carla-Boniface issues the first modern topographical maps (Fig. 3) and August Crome prints the first thematic map, showing economic production data across Europe. Also in this time period appear the first graphics used for descriptive statistics, for example Christian Huygens' plot of a function to graphically determine the years of life remaining given the current age, published in 1669. William Playfair, an English political economist, laid the ground for business graphics in 1786 with his Commercial and Political Atlas in which he documented commercial and political time series using curves, bar charts and column charts (Fig. 4). Playfair is also

Data Visualization. Figure 1. Ca. 6200 BC. The oldest known map, from a museum at Konya, Turkey.



Data Visualization. Figure 2. Ca. 950. First known graphic of a time series visualizing data of planetary orbits.

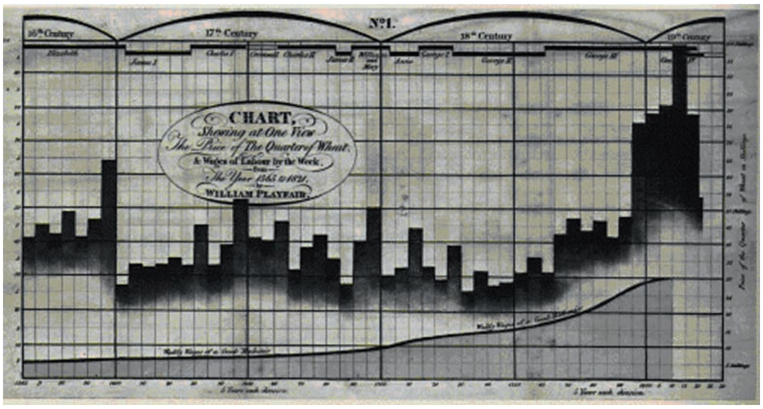
arguably the inventor of the pie chart (Statistical Breviary, 1801).

1800–1949. Scientists and civil servants are beginning to use thematic maps and statistical graphics to support their arguments. Famous examples are the dot map that Dr. John Snow drew by plotting the locations of deaths from cholera in central London during the 1854 epidemic (Fig. 5) and Florence Nightingale's comparison of deaths due to injuries in combat and deaths due to illness in an army hospital for which she invented her own graphic, the Polar-Area Diagram (Fig. 6). The second half of the nineteenth century saw a 50 year long debate on the standardization of statistical maps and diagrams which failed to produce concrete results. Early in the twentieth century there followed a 50 year period of consolidation where the accomplishments of the previous one hundred years became widely accepted.

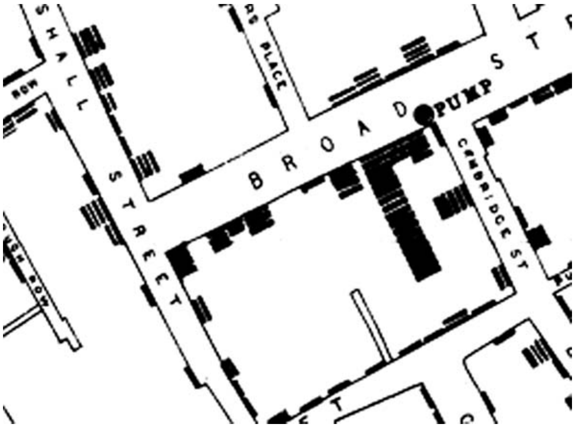
Data Visualization. Figure 3. 1782. Detail of Marcellin du Carla-Boniface's topological map.

1950–1975. Researchers started to face enormous challenges when analyzing the deluge of data produced by electronic equipment that was put in use after the Second World War. In this environment, John Tukey led the way, established the field of Exploratory Data Analysis [15] (Fig. 8) and sparked a flurry of activities that have and still are producing many novel graphical methods [5,8], including techniques that marked the beginning of dynamic statistical graphics. In the 1960s, when signals from remote sensing satellites needed to be processed graphically, geographers started to combine spatially referenced data, spatial models and map based visualizations in geographic information systems. The French cartographer Jacques Bertin worked on a theory of graphics [3] and introduced with his reorderable matrix an elegant technique to graphically process quantitative data [2] (Fig. 7).

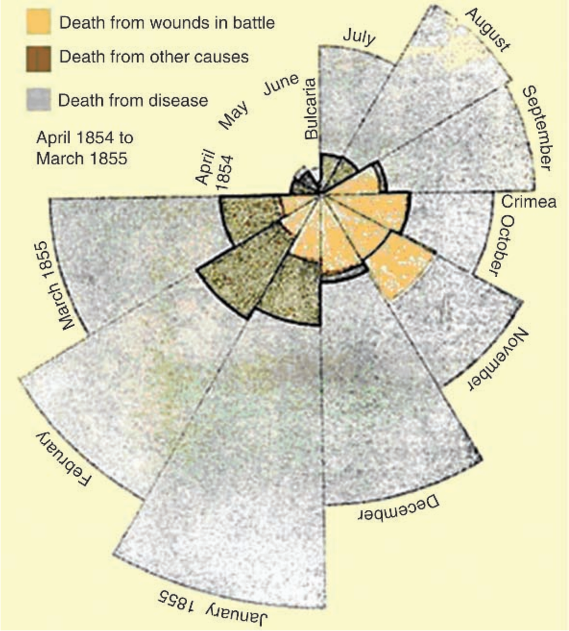
From 1975–present. Fast, interactive computers connected to a high resolution color graphic display created almost unlimited possibilities for scientific visualizations of data generated by imaging techniques, computational geometry or physics based models [10]. Event oriented programming made it easy to link different data displays, encouraging new techniques such as brushing [5]. Statisticians started to tackle high dimensional data by interactively 'touring' low dimensional projections. Large display areas encouraged graphic methods based on multiple plots [5], space filling techniques (e.g., mosaic plots) and graphics with high data densities. For an overview the reader is referred to [1,5,16]. In the early 1990s, virtual



Data Visualization. Figure 4. 1786. William Playfair' chart, depicting prices, wages, and the reigns of British kings and queens.



Data Visualization. Figure 5. 1854. Detail of the pioneering statistical map drawn by John Snow to illustrate patterns of disease.



Data Visualization. Figure 6. 1858. Polar-Area diagram, invented by Florence Nightingale to convince authorities of the need to improve sanitary conditions in hospitals.

environments were introduced as methods to immersively investigate scientific data [4]. (Fig. 10) Another way to overcome the restrictions of two dimensional displays was shown by Alfred Inselberg with his concept of parallel coordinates [9], today a ubiquitous method to visualize multidimensional data (Fig. 9).

To further explore the history of data visualization the reader is referred to [7,14].

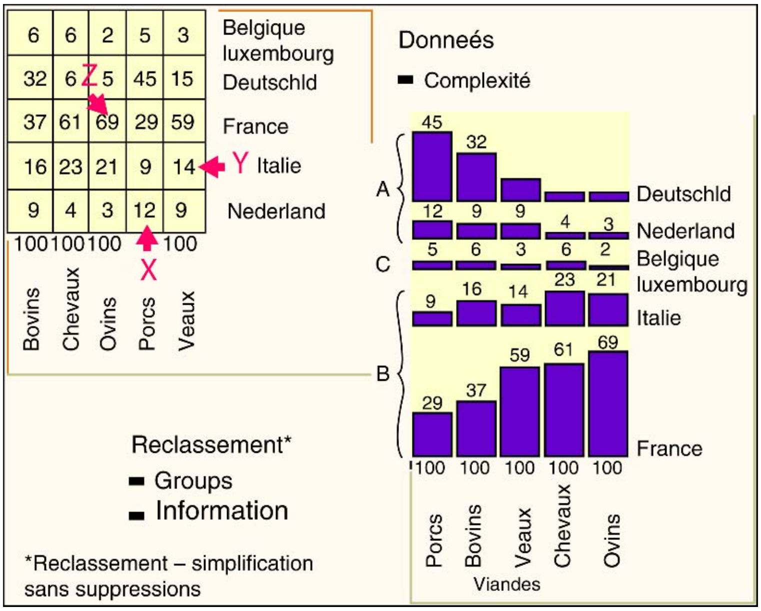
Foundations

The literature on scientific fundamentals of data visualizations fall into three independent but related fields: (i) computer graphics, (ii) presentation techniques, (iii) cognition.

Computer graphics, primarily the domain of computer scientists and mathematicians, builds on elementary principles in the following broad areas: visualization algorithms and data structures, modeling and (numerical) simulation, (volume) rendering, particle tracing, grid generation, wavelet transforms, multiscale and multiresolution methods as well as optics and color theory. A more exhaustive treatment of computer graphics fundamentals related to data visualization can be found in [10,12,17].

Most of the literature on presentation techniques can be found in statistics and computer science although economists and cartographers also made substantial contributions. The publications of John Tukey [15] and Andrew Ehrenberg [6] show how tables can be used as simple but effective presentation technique to organize data and demonstrate the method's usefulness for statistics and data analysis. In 1967 Jacques Bertin formulated a comprehensive theory for a graphical system [3] and subsequently applied parts of it to graphic information processing [2]. Other classics were published in 1983 when William Cleveland wrote an excellent methodological resource for the design of plots and Edward Tufte published his review on the graphical practice to visualize quantitative data. A reference, featuring perhaps the most complete listing of graphs, maps, tables, diagrams, and charts has been compiled by Robert Harris [8]. Parallel coordinates is one of the leading methodologies for multidimensional visualization [9]. Starting from geometric foundations, Al Inselberg explains how *n*-dimensional lines and planes can be represented in 2D through parallel coordinates. The most recent publications explain mostly dynamic, interactive methods. Antony Unwin concentrates on graphics for large datasets [16] while Robert Spence favors techniques that allow user interaction [13].

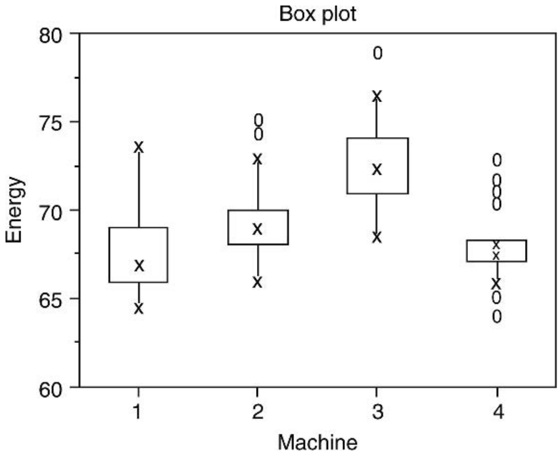
Ultimately, to be of any use, data visualization must support human cognition. The challenges this raises are of interest to cognitive scientists, psychologists, and computer scientists specializing in human-computer



Data Visualization. Figure 7. 1967. Bertin's reorderable matrix, a visualization method embedded in a comprehensive theory of graphics.

interaction. Rudolf Arnheim investigated the role of visual perception as a crucial cognitive activity of reasoning [1]. Also in the domain of 'visual thinking' is the work of Colin Ware [17] as he, among other contributions, proposes a foundation for a science of data visualization based on human visual and cognitive processing. Card et al. discuss topics of computer graphics as well as presentation techniques with a focus on how different methods support cognition. A more general approach worth mentioning is taken by Donald Norman when he argues that people deserve information appliances that fit their needs and lives [11].

To summarize, most of the literature on data visualization describes the efforts of computer scientists and cognitive scientists to develop new techniques for people to interact with data, from small statistical datasets to large information environments.



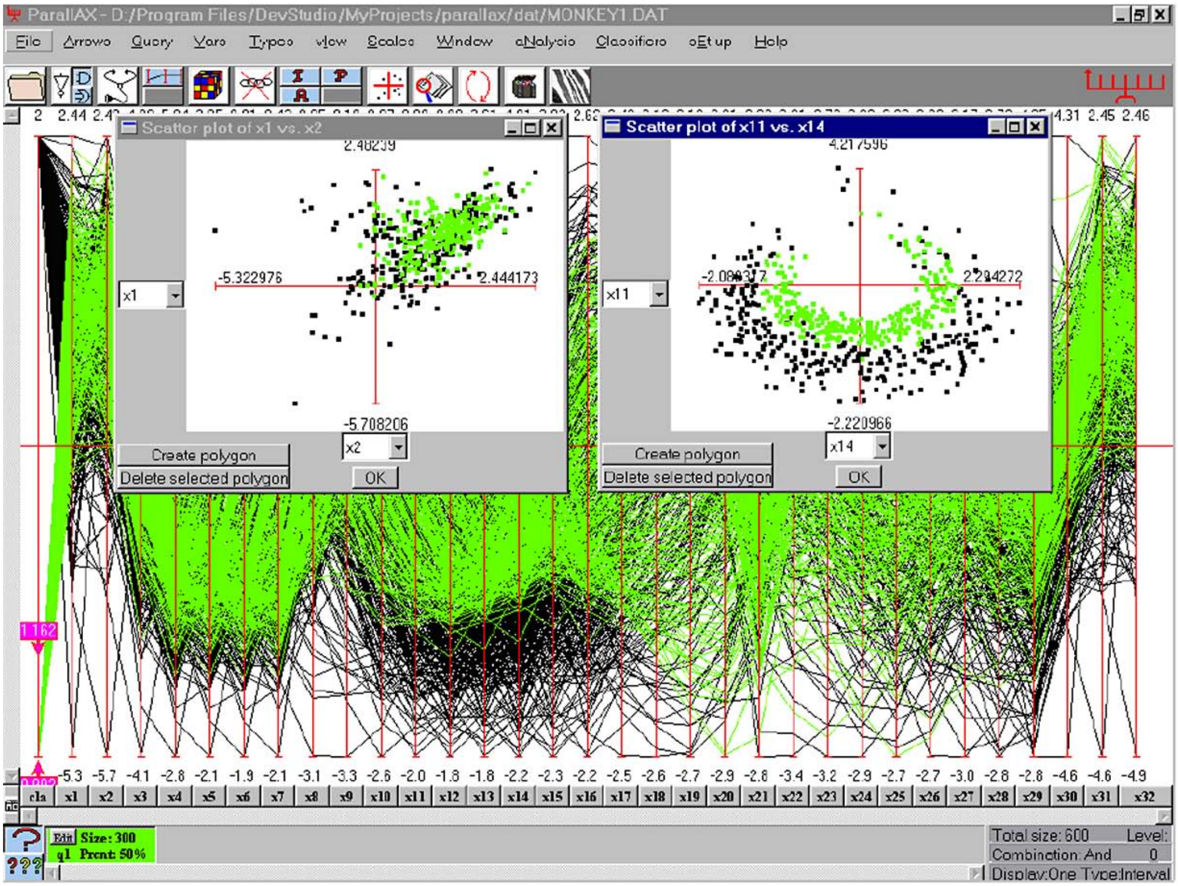
Data Visualization. Figure 8. 1977. Tukey's box-and-whisker plot graphically summarizes effectively key characteristics of the data's distribution.

Key Applications

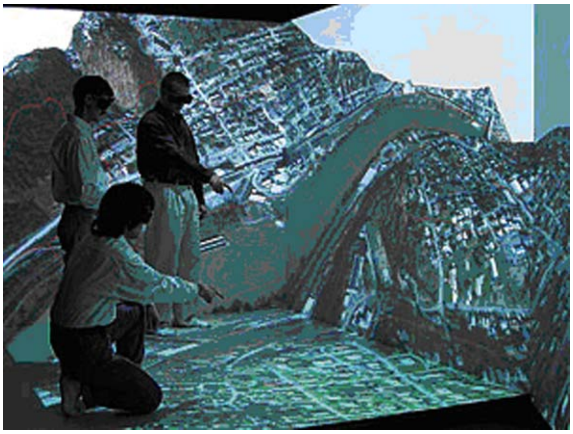
There are few – if any – application areas that do not benefit from data visualization simply because graphical methods assist the fundamental human activity of cognition and because in an increasingly digital world people are flooded with data. In the following four areas, data visualization plays a key role:

Statistics

Descriptive statistics has traditionally been the strongest customer for data visualization, primarily through its application to support exploratory data analysis. The use of data visualization as part of descriptive statistics has become a matter of fact wherever data are being collected.


ParallAX - D:/Program Files/DevStudio/MyProjects/parallax/dat/MONKEY1.DAT
File Arrows Query Vars Types View Scales Window aNalysis Classifiers sEt up Help
Scatter plot of x1 vs. x2
2.48239
-5.322976
2.444173
-5.708206
x1
x2
Create polygon
Delete selected polygon
OK
Scatter plot of x11 vs. x14
4.217596
-2.080317
2.294272
-2.220966
x11
x14
1.162
cla x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20 x21 x22 x23 x24 x25 x26 x27 x28 x29 x30 x31 x32
Edit Size: 300
q1 Prcnt: 50%
Total size: 600 Level:
Combination: And 0
Display: One Type: Interval

Data Visualization. Figure 9. 1999. A continued mathematical development of parallel coordinates led to software for 'visual data mining' in high dimensional data sets.



Data Visualization. Figure 10. 2006. Immersive Geovisualization at West Virginia University.

Information Systems

Data visualization has become an important component in the interface to information systems simply because information is stored as data. The process of recovering information from large and complex databases often depends on data mining techniques. Visual data mining – a new and rapidly growing field – supports people in their data exploration activity with graphical methods. Geographic information systems have traditionally been key applications, particularly for map-based visualizations.

Documentation

Ever since thematic maps and statistics graphics became popular with commerce, government agencies and the sciences, data visualization methods are being used routinely to illustrate that part of documents which deal with data.

Computational Science

Progress in solving scientific and engineering problems increasingly depends on powerful software for modeling and simulation. Nevertheless, success in the end often only comes with effective scientific visualizations. Computational science as a key application for data visualization is a strong driving force behind the development of graphical methods for huge amounts of high dimensional data.

Cross-references

- ▶ Chart
- ▶ Comparative Visualization
- ▶ Dynamic Graphics
- ▶ Exploratory Data Analysis
- ▶ Graph
- ▶ Methods
- ▶ Multivariate Data Visualization
- ▶ Parallel Coordinates
- ▶ Result Display
- ▶ Symbolic Representation

Recommended Reading

1. Arnheim R. Visual Thinking. University of California Press, Berkeley, CA, 1969.
2. Bertin J. Graphics and Graphic Information-Processing. Walter de Gruyter, Berlin/New York, 1981.
3. Bertin J. Semiology of Graphics (translation by W.J. Berg). University of Wisconsin Press, USA, 1983.
4. Card S.K., MacKinlay J.D., and Shneiderman B. Readings in Information Visualization: Using Vision to Think. Morgan Kaufmann, San Francisco, CA, 1999.
5. Cleveland W.S. The Elements of Graphing Data (Revised Edition). Hobart Press, Summit, NJ, 1994.
6. Ehrenberg A.S.C. A Primer in Data Reduction. Wiley, Chichester, UK, 1982.
7. Friendly M. The History of Thematic Cartography, Statistical Graphics, and Data Visualization.
8. Harris R.L. Information Graphics: A Comprehensive Illustrated Reference. Oxford University Press, New York, 1999.
9. Inselberg A. The plane with parallel coordinates. The Visual Comput., 1(2):69–91, 1985.
10. Nielson G.M., Hagen H., and Müller H. Scientific Visualization: Overviews, Methodologies, Techniques. IEEE Computer Society Press, USA, 1997.
11. Norman D.A. The Invisible Computer. The MIT Press, 1998.
12. Post F.H., Nielson G.M., and Bonneau, G.-P. (eds.). Data Visualization: The State of the Art. Kluwer Academic, 2002.
13. Spence R. Information Visualization: Design for Interaction (2nd edn.). Pearson Education, 2007.
14. Tufte E.R. The Visual Display of Quantitative Information. Graphics Press, 1983.
15. Tukey J.W. Exploratory Data Analysis. Addison-Wesley, Reading, MA, 1977.
16. Unwin A., Theus M., and Hofmann H. Graphics of Large Datasets: Visualizing a Million. Springer Series in Statistics and Computing, Berlin, 2006.
17. Ware C. Information Visualization: Perception for Design (2nd edn.). Morgan Kaufmann, 2004.

Data Warehouse

Il-Yeol Song
Drexel University, Philadelphia, PA, USA

Synonyms

Information repository; DW

Definition

A data warehouse (DW) is an integrated repository of data put into a form that can be easily understood, interpreted, and analyzed by the people who need to use it to make decisions. The most widely cited definition of a DW is from Inmon [2] who states that "a data warehouse is a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions."

The *subject-oriented* property means that the data in a DW are organized around major entities of interests of an organization. Examples of subjects are customers, products, sales, and vendors. This property allows users of a DW to analyze each subject in depth for tactical and strategic decision-making.

The *integrated* property means that the data in a DW are integrated not only from all operational database systems but also some meta-data and other related external data. When data are moved from operational databases to a DW, they are extracted, cleansed, transformed, and then loaded. This makes a DW a centralized repository of all the business data with common semantics and formats.

The *nonvolatile* property means that the data in a DW are not usually updated. Once the data are loaded into a DW, they are not deleted. Any change to the data that were already moved to a DW is recorded in the form of a snapshot. This allows a DW to keep track of the history of the data.

The *time-variant* property means that a DW usually contains multiple years of data. It is not uncommon for a DW to contain data for more than ten years. This allows users of a DW to analyze trends, patterns, correlations, rules, and exceptions from a historical perspective.

Key Points

DWs have become popular for addressing the needs of a centralized repository of business data in decision-making. An operational database system, also known as an online transaction processing (OLTP) system, supports daily business processing. On the other hand, a DW usually supports tactical or strategic business processing for business intelligence. While an OLTP system is optimized for short transactions, a DW system is optimized for complex decision-support queries. Thus, a data warehouse system is usually maintained separately from operational database systems. This distinction makes DW systems different from OLTP systems in many aspects.

The data in a DW are usually organized in formats for easy access and analysis in decision-making. The most widely used data model for DWs is called the dimensional model or the star schema [3]. A dimensional model consists of two types of entities – a fact table and many dimensions. A *fact* table stores transactional or factual data called measures that are analyzed. Examples of fact tables are order, sale, return, and claim. A dimension represents an axis that analyzes the fact data. Examples of dimensions are time, customer, product, promotion, store, and market. The dimensional model allows users of a data warehouse to analyze the fact data from any combination of dimensions. Thus, a dimensional model simplifies end-user query processing and provides a multidimensional analysis space within a relational database.

The different goals and data models of DWs need special access, implementation methods, maintenance, and analysis methods, different from those of OLTP systems [1]. Therefore, a data warehouse requires an environment that uses a blend of technologies.

Cross-references

▶ Active and Real-Time Data Warehousing
▶ Business Intelligence
▶ Data Mart
▶ Data Mining
▶ Data Warehouse Life-cycle and Design
▶ Data Warehouse Maintenance, Evolution and Versioning
▶ Data Warehouse Metadata
▶ Data Warehouse Security
▶ Data Warehousing and Quality Data Management for Clinical Practice
▶ Data Warehousing for Clinical Research
▶ Data Warehousing Systems: Foundations and Architectures
▶ Dimension
▶ Multidimensional Modeling
▶ On-Line Analytical Processing

Recommended Reading

1. Chaudhuri S. and Dayal U. An overview of data warehousing and OLAP technology. ACM SIGMOD Rec., 26(1):65–74, 1997.
2. Inmon W.H. Building the Data Warehouse, 3rd edn. Wiley, New York, 2002.
3. Kimball R. and Ross M. The Data Warehouse Toolkit, 2nd edn. Wiley, New York, 2002.

Data Warehouse Back Stage

▶ Extraction, Transformation and Loading

Data Warehouse Design Methodology

▶ Data Warehouse Life-Cycle and Design

Data Warehouse Life-Cycle and Design

Matteo Golfarelli
University of Bologna, Bologna, Italy

Synonyms

Data Warehouse design methodology

Definition

The term *data warehouse life-cycle* is used to indicate the phases (and their relationships) a data warehouse system goes through between when it is conceived and when it is no longer available for use. Apart from the type of software, life cycles typically include the following phases: requirement analysis, design (including modeling), construction, testing, deployment, operation, maintenance, and retirement. On the other hand, different life cycles differ in the relevance and priority with which the phases are carried out, which can vary according to the implementation constraints (i.e., economic constraints, time constraints, etc.) and the software specificities and complexity. In particular, the specificities in the data warehouse life-cycle derive from the presence of the operational database that feeds the system and by the extent of this kind of system that must be considered in order to keep the cost and the complexity of the project under control.

Although the design phase is only a step within the overall life cycle, the identification of a proper life-cycle model and the adoption of a correct *design methodology* are strictly related since each one influences the other.

Historical Background

The *data warehouse* (DW) is acknowledged as one of the most complex information system modules, and its design and maintenance is characterized by several complexity factors, which determined, in the early stages of this discipline, a high percentage of project failures. A clear classification of the critical factors of Data Warehousing projects was already available in 1997 when three different risk categories were identified [1]:

- *Socio-technical:* DW projects have deep impact on the decisional processes and political equilibriums, thus reducing the power of some stakeholders who will be willing to interfere with the project. For example, data ownership is power within an organization. Any attempt to share or take control over somebody else's data is equivalent to a loss of power of this particular stakeholder. Furthermore, no division or department can claim to possess 100% clean, error-free data. The possibility of revealing the quality problems of data within the information system of the department is definitely frustrating for the stakeholders affected.
- *Technological:* DW technologies are continuously evolving and their features are hard to test. As a consequence, problems related to the limited scalability of the architecture, difficulty in sharing meta-data between different components and the inadequate expertise of the programmers may hamper the projects.
- *Design:* designing a DW requires a deep knowledge of the business domain. Some recurrent errors are related to limited involvement of the user communities in the design as well as the lack of a deep analysis of the quality of the source data. In both these cases, the information extracted from the DW will have a limited value for the stakeholders since they will turn out to be unreliable and outside the user focus.

The awareness of the critical nature of the problems and the experience accumulated by practitioners determined the development of different design methodologies and the adoption of proper life cycles that can increase the probability of completing the project and fulfill the user requirements.

Foundations

The choice of a correct life cycle for the DW must take into account the specificities of this kind of systems, which according to [2], are summarized as follows:

1. DWs rely on operational databases that represent the sources of the data.
2. User requirements are difficult to collect and usually change during the project.
3. DW projects are usually huge projects: the average time for their construction is 12–36 months and their average cost ranges from 0.5 to 10 million dollars.
4. Managers are demanding users that require reliable results in a time compatible with business needs.

While there is no consensus on how to address points (i) and (ii), the DW community has agreed on an approach that cuts down cost and time to make a satisfactory solution available to the final users. Instead of approaching the DW development as a whole in a top-down fashion, it is more convenient to build it bottom-up working on single data marts [3]. A *data mart* is part of a DW with a restricted scope of content and support for analytical processing, serving a single department, part of an organization, and/or a particular data analysis problem domain. By adopting a bottom-up approach, the DW will turn out to be the union of all the data marts.

This iterative approach promises to fulfill requirement (iii) since it cuts down development costs and time by limiting the design and implementation efforts to get the first results. On the other hand, requirement (iv) will be fulfilled if the designer is able to implement first those data marts that are more relevant to the stakeholders.

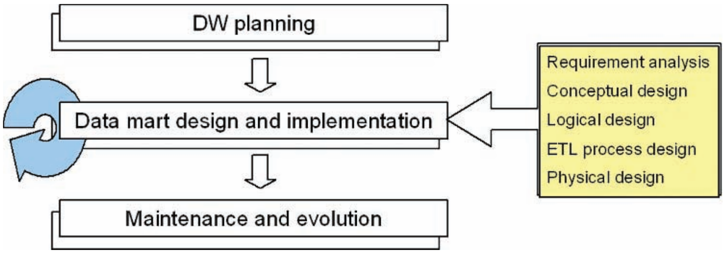
As stated by many authors, adopting a pure bottom-up approach presents many risks originating from the partial vision of the business domain that will be available at each design phase. This risk can be limited by first developing the data mart that plays a central role within the DW, so that the following can be easily integrated into the existing backbone; this kind of solution is also called *bus architecture*. The basis for designing coherent data marts and for achieving an integrated DW is the agreement of all the design teams on the classes of analysis that are relevant for the business. This is primarily obtained by the adoption of *conformed dimensions* of analysis [4]. A dimension is conformed when two copies of the dimensions are either exactly the same (including the values of the keys and all the attributes), or else one dimension is a proper subset of the other. Therefore, using the same time dimension in all the data marts implies that the data mart teams agree on a corporate calendar. All the data mart teams must use this calendar and agree on fiscal periods, holidays, and workdays. When choosing the first data mart to be implemented the designer will probably cope with the fact that the most central data mart (from a technical point of view) is not the most relevant to the user. In that case, the designer choice must be a trade-off between technical and political requirements.

Based on these considerations the main phases for the DW life-cycle can be summarized as follows:

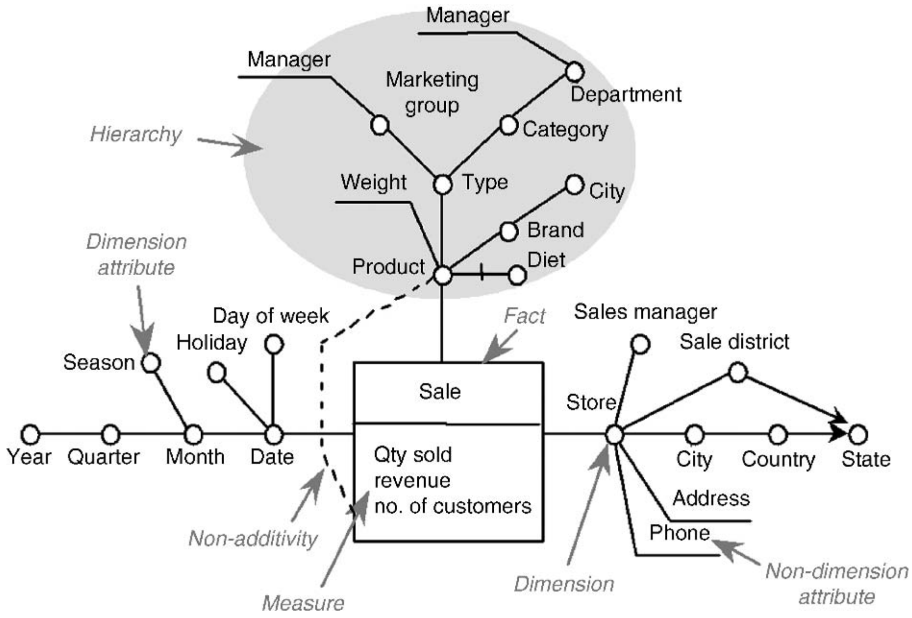
1. *DW planning*: this phase is aimed at determining the scope and the goals of the DW, and determines the number and the order in which the data marts are to be implemented according to the business priorities and the technical constraints [5]. At this stage the physical architecture of the system must be defined too: the designer carries out the sizing of the system in order to identify appropriate hardware and software platforms and evaluates the need for a reconciled data level aimed at improving data quality. Finally, during the project planning phase the staffing of the project is carried out.
2. *Data mart design and implementation*: this macro-phase will be repeated for each data mart to be implemented and will be discussed in more detail in the following. At every iteration, a new data mart is designed and deployed. Multidimensional modeling of each data mart must be carried out considering the available conformed dimensions and the constraints derived from previous implementations.
3. *DW maintenance and evolution*: DW maintenance mainly concerns performance optimization that must be periodically carried out due to user requirements that change according to the problems and the opportunities the managers run into. On the other hand, DW evolution concerns keeping the DW schema up-to-date with respect to the business domain and the business requirement changes: a manager requiring a new dimension of analysis for an existing *fact schema* or the inclusion of a new level of classification due to a change in a business process may cause the early obsolescence of the system (Fig. 1).

DW design methodologies proposed in the literature mainly concern phase 2 and thus should be better referred to as data mart design methodologies. Though a lot has been written about how a DW should be designed, there is no consensus on a design method yet. Most methods agree on the opportunity of distinguishing between the following phases:

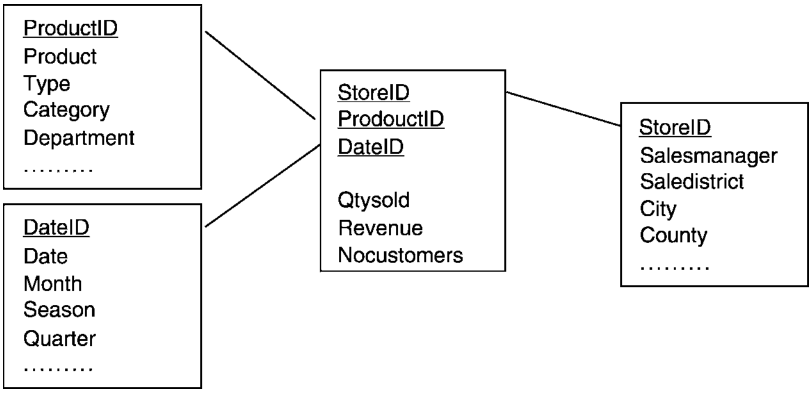
- *Requirement analysis:* identifies which information is relevant to the decisional process by either considering the user needs or the actual availability of data in the operational sources.
- *Conceptual design*: aims at deriving an implementation-independent and expressive conceptual schema for the DW, according to the conceptual model chosen (see Fig. 2).
- *Logical design*: takes the conceptual schema and creates a corresponding logical schema on the chosen logical model. While nowadays most of the DW systems are based on the relational logical model (ROLAP), an increasing number of software vendors are proposing also pure or mixed multidimensional solutions (MOLAP/HOLAP). Figure 3 reports the



Data Warehouse Life-Cycle and Design. Figure 1. The main phases for the DW life-cycle.



Data Warehouse Life-Cycle and Design. Figure 2. A conceptual representation for the SALES fact based on the DFM model [6].



Data Warehouse Life-Cycle and Design. Figure 3. A relational implementation of the SALE fact using the well-known star schema.

relational implementation of the SALE fact based on the well-known star schema [4].

- *ETL process design*: designs the mappings and the data transformations necessary to load into the logical schema of the DW the data available at the operational data source level.
- *Physical design*: addresses all the issues specifically related to the suite of tools chosen for implementation – such as indexing and allocation.

Requirement analysis and conceptual design play a crucial role in handling DW peculiarities (i) and (ii) described at the beginning of the present section. The lack of settled user requirements and the existence of operational data sources that fix the set of available information make it hard to develop appropriate multidimensional schemata that on the one hand fulfill user requirements and on the other can be fed from the operational data sources. Two different design principles can be identified: supply-driven and demand-driven [5].

- *Supply-driven* approaches [3,6] (also called *data-driven*) start with an analysis of operational data sources in order to reengineer their schemata and identify all the available data. Here user involvement is limited to select which chunks of the available data are relevant for the decision-making process. While supply-driven approaches simplify the design of the ETL because each piece of data in the DW corresponds to one or more attributes of the sources, they give user requirements a secondary role in determining the information contents for analysis as well as giving the designer little support in identifying facts, dimensions, and measures. Supply-driven approaches are feasible when all of the following are true: (i) detailed knowledge of data sources is available a priori or easily achievable; (ii) the source schemata exhibit a good degree of normalization; and (iii) the complexity of source schemata is not too high.

- *Demand-driven* approaches [7,8] start from determining the information requirements of business users. The emphasis is on the requirement analysis process and on the approaches for facilitating user participations. The problem of mapping these requirements onto the available data sources is faced only a posteriori, and may fail thus determining the users' disappointment as well as a waste of the designer's time.

Based on the previous approaches some mixed modeling solutions have been proposed in the last few years in order to overcome the weakness of each pure solution.

Conceptual design is widely recognized to be the necessary foundation for building a DW that is well-documented and fully satisfies the user requirements. The goal of this phase is to provide the designer with a high level description of the data mart possibly at different levels of detail. In particular, at the DW level it is aimed at locating the data mart within the overall DW picture, basically characterizing the class of information captured, its users, and its data sources. At the data mart level, a conceptual design should identify the set of facts to be built and their conformed dimensions. Finally, at the fact level a nonambiguous and implementation-independent representation of each fact should be provided. If a supply driven approach has been followed for requirement analysis, the conceptual model at the schema level can be semi-automatically derived from the source schemata by identifying the many-to-one relationship [3,6]. Concerning the formalism to be adopted for representing information at this level, researchers and practitioners agreed that, although the E/R model has enough expressivity to represent most necessary concepts, in its basic form, it is not able to properly emphasize the key aspects of the multidimensional model. As a consequence many ad-hoc formalisms has been proposed in the last years (e.g., [6,9]) and a comparison of the different models done by [10] pointed out that, abstracting from their graphical form, the core expressivity is similar, thus proving that the academic community reached an informal agreement on the required expressivity.

Logical design is the phase that most attracted the interest of researchers in the early stage of Data Warehousing since it strongly impacts the system performance. It is aimed at deriving out of the conceptual schemata the data structure that will actually implement the data mart by considering some sets of constraints (e.g., concerning disk space or query answering time) [11]. Logical design is more relevant when a relational DBMS is adopted (ROLAP) while in the presence of a native multidimensional DBMS (MOLAP) the logical model derivation is straightforward. On the other hand, in ROLAP system, the choices concern, for example the type of schema to be adopted (i.e., star o snowflake), the specific solution for historicization of data (i.e., slowly changing dimensions) and schema.

ETL process design is considered to be the most complex design phase and usually takes up to 70% of the overall design time. Complexity arises from the need of integrating and transforming heterogeneous and inconsistent data coming from different data sources. This phase also includes the choice of the strategy for handling wrong and incomplete data (e.g., discard, complete). Obviously, the success of this phase impacts the overall quality of DW data. Different from other design phases little efforts have been made in the literature to organize and standardize this phase [12,13], and actually none of the formalisms proposed have been widely adopted in real projects that usually rely on the graphical representation obtained from the ETL tool for documentation purposes.

Finally, during physical design, the logical structure is optimized based on the means made available by the adopted suite of tools. Specialized DBMSs usually include ad hoc index types (e.g., bitmap index and join index) and can store the meta-knowledge necessary to automatically rewrite a given query on the appropriate materialized view. In DW systems, a large part of the available disk space is devoted to optimization purposes and it is a designer task to find out its assignment to the different optimization data structures in order to maximize the overall performance [14].

Despite the basic role played by a well-structured methodological framework in ensuring that the DW designed fully meets the user expectations, only a few of the cited papers cover all the design phases [6,13]. In addition, an influential book, particularly from the practitioners' viewpoint, is the one by Kimball [4], which discusses the major issues arising in the design and implementation of data warehouses. The book presents a case-based approach to data mart design that is bottom-up oriented and adopts a mixed approach for collecting user requirements.

Finally it should be noted that, though most vendors of DW technology propose their own CASE solutions (that are very often just wizards capable of supporting the designer during the most tedious and repetitive phases of design), the only tools that currently promise to effectively automate some phases of design are research prototypes. In particular, [3,15], embracing the supply-driven philosophy, propose two approaches for automatically deriving the conceptual multidimensional schema from the relational data sources. On the contrary the CASE tool proposed in [12] follows the demand-driven approach and allows the multidimensional conceptual schemata to be drawn from scratch and to be semi-automatically translated into the target commercial tool.

Key Applications

The adoption of an appropriate methodological approach during design phases is crucial to ensure the project success. People involved in the design must be skilled on this topic, in particular.

Designers

Designers should have a deep knowledge of the pros and cons of different methodologies in order to adopt the one that best fits the project characteristics.

Business Users

Users should be aware of the design methodology adopted and their role within it in order to properly support the designer's work and to provide the correct information at the right time.

Future Directions

Research on this topic should be directed to generalizing the methodologies discussed so far in order to derive a consensus approach that, depending on the characteristics of the project, will be made up of different phases. Besides, more generally, mechanisms should appear to coordinate all DW design phases allowing the analysis, control, and traceability of data and metadata along the project life-cycle. An interesting approach in this direction consists in applying the Model Driven Architecture to automate the inter schema transformations from requirement analysis to implementation [16]. Finally, the emergence of new applications for DW such as spatial DW [17], web DW, real-time DW [18], and business performance management [19] will have their side-effects on the DW life-cycle and inevitably more general design methodologies will be devised in order to allow their correct handling.

Cross-references

▶ Cube Implementations
▶ Data Mart
▶ Data Warehouse Maintenance, evolution and versioning
▶ Data Warehousing Systems: Foundations and Architectures
▶ Multidimensional Modeling
▶ Optimization and Tuning in Data Warehouses
▶ Snowflake Schema
▶ Star Schema

Recommended Reading

1. Abello A., Samos J., and Saltor F.YAM2: a multidimensional conceptual model extending UML. Infor. Syst., 31(6):541–567, 2006.
2. Bimonte S., Towards S., and Miquel M.Towards a Spatial Multidimensional Model. In Proc. ACM 8th Int. Workshop on Data Warehousing and OLAP, 2005.
3. Demarest, M. The politics of data warehousing. Retrieved June 2007 from http://www.noumenal.com/marc/dwpoly.html.
4. Giorgini P., Rizzi S., and Garzetti M. GRAnD: A goal-oriented approach to requirement analysis in data warehouses. Decision Support System, 2008, 45(1):4–21.
5. Golfarelli M., Maio D., and Rizzi S. The dimensional fact model: a conceptual model for data warehouses. Int. J. Coop. Inf. Syst. 7(2–3): 215–247, 1998.
6. Golfarelli M. and Rizzi S. WAND: A CASE tool for data warehouse design. In Proc. 17th Int. Conf. on Data Engineering, 2001.
7. Golfarelli M., Rizzi S., and Cella I. Beyond data warehousing: What's next in business intelligence? In Proc. ACM 7th Int. Workshop on Data Warehousing and OLAP, 2004.

8. Golfarelli M., Rizzi S., and Saltarelli E. Index selection for data warehousing. In Proc. 4th Int. Workshop on Design and Management of Data Warehouses, 2002.
9. Hüsemann B., Lechtenbörger J., and Vossen G. Conceptual data warehouse design. In Proc. 2nd Int. Workshop on Design and Management of Data Warehouses, 2000.
10. Jarke M., Lenzerini M., Vassiliou Y., and Vassiliadis P. Fundamentals of Data Warehouses. Springer, 2000.
11. Jensen M., Holmgren T., and Pedersen T. Discovering Multidimensional Structure in Relational Data. In Proc. 6th Int. Conf. Data Warehousing and Knowledge Discovery, 2004.
12. Kimbal R., Reeves L., Ross M., and Thornthwaite W. The Data Warehouse Lifecycle Toolkit. Wiley, New York, 1998.
13. Laender A., Freitas G., and Campos M. MD2 – Getting users involved in the development of data warehouse applications. In Proc. 14th Int. Conf. on Advanced Information Systems Eng., 2002.
14. Mazon J., Trujillo J., Serrano M., and Piattini M. Applying MDA to the development of data warehouses. In Proc. ACM 8th Int. Workshop on Data Warehousing and OLAP, 2005.
15. Theodoratos D. and Sellis T. Designing data Data warehouses. Data & Knowl. Eng., 31(3):279–301, 1999.
16. Tho N. and Tjoa A. Grid-Based Zero-Latency Data Warehousing for continuous data streams processing. In Proc. 6th Int. Conf. Information Integration and Web Based Applications & Services, 2004.
17. Trujillo J. and Luján-Mora S.A. UML Based Approach for Modeling ETL Processes in Data Warehouses. In Proc. 22nd Int. Conf. on Conceptual Modeling, 2003.
18. Trujillo J., Luján-Mora S., and Medina E. The Gold model case tool: An environment for designing OLAP applications. In Proc. ACM 5th Int. Workshop on Data Warehousing and OLAP, 2002.
19. Vassiliadis P., Simitsis A., and Skiadopoulos S. Conceptual modeling for ETL processes. In Proc. ACM 5th Int. Workshop on Data Warehousing and OLAP, 2002.
20. Winter R. and Strauch B. A method for demand-driven information requirements analysis in data warehousing. In Proc. 36th Annual Hawaii Int. Conf. on System Sciences, 2003.

Data Warehouse Maintenance, Evolution and Versioning

Johann Eder[1], Karl Wiggisser[2]
[1]University of Vienna, Vienna, Austria
[2]University of Klagenfurt, Klagenfurt, Austria

Synonyms

Temporal data warehousing

Definition

A multidimensional data warehouse consists of three different levels: The schema level (dimensions, categories), the instance level (dimension members, master data) and the data level (data cells, transaction data). The process and methodology of performing changes on the schema and instance level to represent changes in the data warehouse's application domain or requirements is called *Data Warehouse Maintenance*. *Data Warehouse Evolution* is a form of data warehouse maintenance where only the newest data warehouse state is available. *Data Warehouse Versioning* is a form of data warehouse maintenance where all past versions of the data warehouse are kept available. Dealing with changes on the data level, mostly insertion of new data, is not part of data warehouse maintenance, but part of a data warehouse's normal operation.

Historical Background

Data warehouses are supposed to provide functionality for storing and analyzing data over a long period of time. Since the world is changing, the need for applying changes to data warehouse structures arose. Kimball [8] was probably the first to describe the problem and propose solutions. Several more sophisticated proposals followed (see below).

Foundations

A multidimensional data warehouse consists of three different levels: The schema level, the instance level, and the data level. On the schema level a data warehouse is defined by a set of dimensions and corresponding dimension categories, which build up a category hierarchy. On the instance level a data warehouse is defined by a set of dimension members for each dimension. Dimension members build up a member hierarchy which corresponds to the category hierarchy of the respective dimension. Schema and instance level together define the structure of a data warehouse. Different multidimensional models deal with measures in different ways. If no particular measure dimension is defined, measures are modeled as attributes of the fact table, thus are seen as part of the schema. If there is a measure dimension existing, measures are members of this particular dimension and therefore seen as instances. On the data level, a data warehouse consists of a set of data cells, which hold the actual values to analyze. A data cell is defined by selecting one dimension member from each dimension.

Whereas changes on the data level, most of the time data inserts, are part of the daily business in data warehouse systems, modifications of the data warehouse structure need additional effort. Structural modifications can be implied by changes in the application domain of a data warehouse system or by changes in the requirements.

Levels of the Maintenance Problem

Data warehouse maintenance systems must provide means to keep track of schema modifications as well as of instance modifications. On the schema level one needs operations for the *Insertion*, *Deletion* and *Change* of dimensions and categories. Category changes are for instance adding or deleting user defined attributes. Also the hierarchical relations between categories may be modified. On the instance level operations for the *Insertion, Deletion* and *Change* of dimension members are needed, as well as operations for changing the hierarchical relations between dimension members. Whether changing measures is a schema or instance change depends on the underlying multidimensional model. Typically, schema changes happen rarely but need much effort to be dealt with, whereas modifications of instances may happen quite often, but need fewer effort.

Keeping track of the data warehouse structure is only one aspect of data warehouse maintenance. The structure of the cell data contained in a data warehouse is determined by the data warehouse's structure. Thus, if this structure changes, existing cell data may have to be adjusted to be consistent with the new structure. Such adjustments can range from simple reaggregation to complex data transformations because for instance some unit of a measure is changed. These data adaptations must not be mistaken for data change operations as mentioned above, for instance loading new data into the data warehouse.

Figure 1 shows an example for instance and schema changes. It contains three subsequent versions of one dimension of a car dealer's data warehouse structure together with the categories for this dimension. On top, the initial version is shown. The dealer sells different car models of different brands. Each model has an attribute which denotes the engine power. For traditional German models this is given in horsepower, for English models it is given in kilowatt. The outline in the middle shows the subsequent version, where two instance changes can be seen: a new model *(BMW 1)* is introduced, and one model *(Phantom V)* is discontinued. The bottom outline shows the current structure version. Here one can see a schema change: a new category *(Company)* is inserted into the category hierarchy. On the instance level there are a number of changes: one brand *(Puch)* is removed from the product portfolio. The model *(Modell G)* attached to this brand is now sold under another brand *(Mercedes)*. Furthermore a new brand *(Chrysler)* was added to the product portfolio, together with one model assigned to it. For the newly introduced category two dimension members *(BMW&Rolls-Royce* and *Daimler-Chrysler)* are added and the brands are connected to the respective company. The attribute denoting the power of a model is unified for all models to kilowatt. All the mentioned structure modifications are due to changes in the application domain. A requirements change leading to structure updates could for instance be that besides analyzing the number of car sells, the car dealer also wants to keep track of the resulting profit (insert measure).

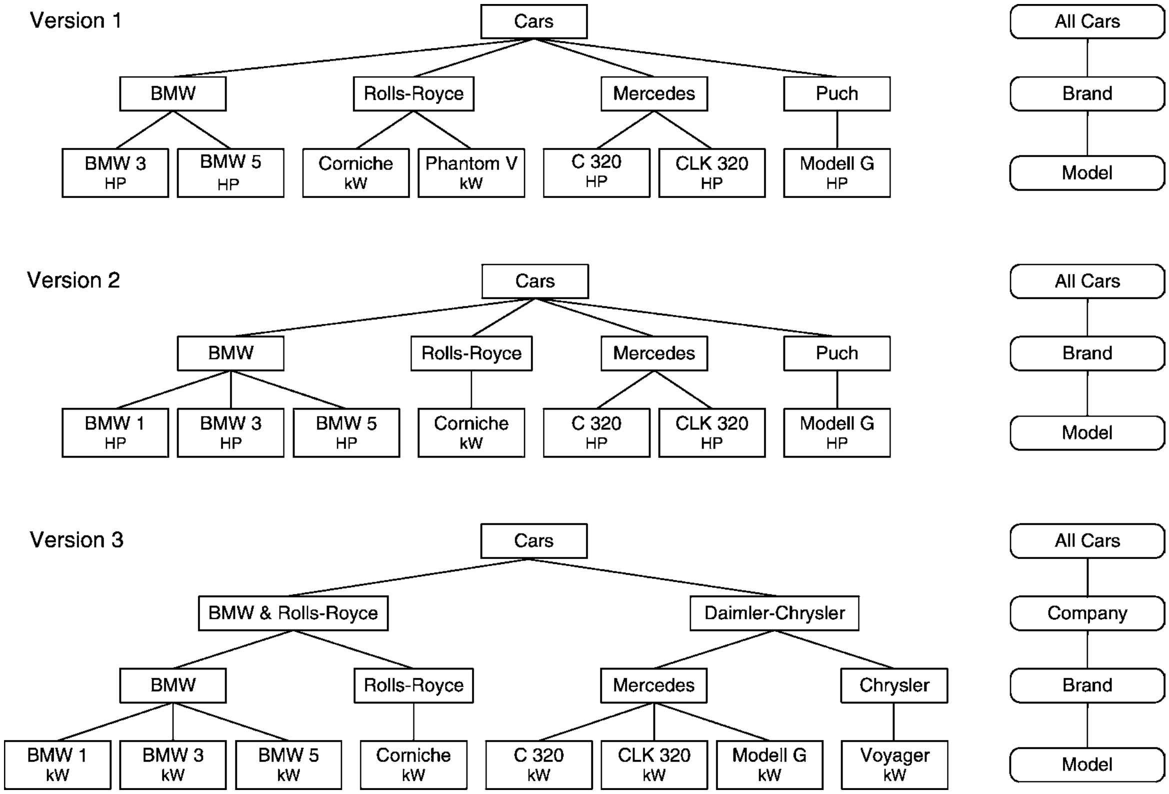
A data adjustment for this example would be the reaggregation to express that *Modell G* is now sold under the brand of *Mercedes*. Data transformation could for instance result from changing the currency from ATS to EUR, where every money-related value has to be divided by 13.7603.

Data Warehouse Versioning Versus Data Warehouse Evolution

In principle two methods of maintenance can be distinguished: Evolution and Versioning. Both of these techniques rely on the defined operations for structure changes but significantly vary in terms of query flexibility, query costs and data management effort. This distinction between versioning and evolution can be applied for both the schema and the instance level.

With *Data Warehouse Evolution*, every applied operation changes the structure of the data warehouse and the old structure is lost. The respective cell data is transformed to fit the new structure. As the old structure is lost, queries can only be done against the current structure. Queries spanning different structure versions are not possible. As the data follows one single structure, no adaptations have to be done during query runtime, which results in a better query performance compared to the versioning approach. Furthermore, no information about former versions has to be kept, which reduces the effort for data management.

With *Data Warehouse Versioning* every applied operation again leads to a new structure version. But in contrast to the evolutionary approach the old version is also kept available. Existing cell data does not need to be adapted, but can be stored further on following



Data Warehouse Maintenance, Evolution and Versioning. Figure 1. Changes in Data Warehouse Structure.

the respective structure version. This facilitates queries spanning multiple structure versions. When running such multiversion queries, data has to be either adapted in runtime, which reduces query performance, or pre-calculated and stored, which increases the required space and maintenance effort. Keeping track of structure version history is mandatory, which results in a considerable effort for the data management.

Approaches Addressing the Maintenance Problem

There are a set of approaches addressing the data warehouse maintenance problem. Kimball [8] is one of the first, discovering the need for evolving data warehouses and introducing three methods for dealing with "slowly changing dimensions". The first method proposes simply overwriting old instances with their new values. Tracking a change history is not possible. The second method consists in creating a new instance for each change. This will create a change history, but needs additional effort in data management. One has to introduce a surrogate key, because the natural primary keys may not be unique any longer. For relating the various instances for an object to each other, creating a time stamp for the validity of each version is proposed. The third method proposes creating a new attribute for the instance, such that the original and the current attribute value can be saved. This method can of course only handle two versions of an instance. All three methods are quite straightforward and only allow very basic modifications on the instance level.

With FIESTA [2] Blaschka, Sapia and Höfling present a schema design technique supporting schema evolution. Evolution for instances is not supported, but FIESTA provides an automatism to adapt existing instances after schema modification. For this adaptation two alternatives are proposed: adaption on the physical level (i.e., database changes) and adaption on the logical level (i.e., create a filter for accessing the instances). The authors define a rich set of schema changing operations, including the creation and deletion of dimensions, categories and attributes.

In [11] Ravat and Teste present their approach for dealing with changing instances. The authors define an object-oriented approach for data warehouse modeling, based on the class concept proposed by the Object Database Management Group. A warehouse object (instance) is defined by its current state and a set of historical and archived states. The difference between historical and archived states is that historical states can be exactly reestablished, whereas for archived states only aggregations are kept, for reducing data size. Mapping functions describe the building process from which the data warehouse classes are generated.

The approach of Hurtado, Mendelzon and Vaisman [14] allows data warehouse evolution on the schema and the instance level. Both schema and instances are modeled using a directed acyclic graph where the nodes represent levels and instances, respectively. The edges are labeled with their valid time intervals. Nodes connected to edges are only valid in the time interval where the edge is valid. Operations for inserting and deleting categories and instances are provided. Evolution of instances is not supported. Defining whether a specific instance is part of the current schema happens by timestamping the edge which connects the node to the graph. Additionally the temporal query language TOLAP is defined to enable queries over a set of temporal dimensions and temporal fact tables.

In [4,5] Eder and Koncilia present their COMET Metamodel for temporal data warehousing. Based on the principles of temporal databases, they introduce a system that supports data warehouse versioning on the schema and the instance level. COMET provides a rich set of maintenance operations, which comprise insertion, deletion, and update of schema elements and instances. Also the complex operations split member and merge members are defined. In contrast to other approaches these operations can also be applied on the time and fact dimensions. COMET furthermore defines so called transformation functions, which allow to transform the cell data between arbitrary versions of the data warehouse. This provides the functionality of queries spanning several structure versions.

In [6] Golfarelli et al. present their approach for schema versioning in data warehouse. Based on a graph model of the data warehouse schema they present their algebra for schema modifications. This approach supports versioning, therefore past versions are not lost. Based on those schema versions the authors describe a mechanism to execute cross-version queries, with the help of so called augmented schemas. For creating such an augmented schema, an old schema version is enriched with structure elements from a subsequent version, such that the data belonging to the old schema version can be queried as if it follows the new version.

Besides these research proposals there are also two commercial products which introduce basic means for data warehouse maintenance. SAP Inc. describes in a white paper [9] how to produces different types of reports over existing data. This can be a report using the current constellation, a report using an old constellation, a report showing the historical truth, and a report showing comparable results. This approach supports only basic operations on dimension data.

The KALIDO Dynamic Information Warehouse [7] also realizes some aspects of data warehouse maintenance. Their support for change is based on the so called generic data modeling. The data warehouse model consists of three categories of data, the transaction data (which describes the activities of the business and the measures associated with them), the business context data (which is the analog to the instances), and the metadata (which comprises among others, parts the schema). With evolving the business context data, instance evolution is supported.

There are a set of alternative approaches which have not been mentioned yet. The different techniques addressing the data warehouse maintenance problem can be classified by two features: First, by whether they support structure versioning or structure evolution, and second by the level of modifications they can handle. Table 1 shows this classification for some of the best known approaches in this area. So each of the mentioned approaches provides the features naming the respective row and column.

Besides the classical maintenance requirements of keeping track of changes in data warehouse, maintenance methodologies can also be used to facilitate so called what–if-analysis. In [1] Bebel et al. present their approach for the management of multiversion data warehouses. They differentiate between real versions and alternative versions. Real versions are used to historicize data warehouse modifications resulting from real world changes. Alternative versions provide the

Data Warehouse Maintenance, Evolution and Versioning. Table 1. Classification of data warehouse maintenance approaches

	Versioning	Evolution
Schema and instance maintenance	[4]	[14]
Schema maintenance only	[6]	[2,10]
Instance maintenance only	[9,11,13]	[7,3,8,15]

functionality to create several versions, each of them representing a possible future situation and then apply what–if-analysis on them. Additionally, alternative versions can be used to simulate data warehouse changes for optimization purposes.

Another instance of data warehouse maintenance is the so called view maintenance. Whereas the approaches presented above assume a data warehouse structure which is defined somehow independent from underlying data sources and is populated with data by ETL-processes, a data warehouse can also be seen as materialized view over a set of data sources. Such a materialized view is of course directly affected by changes in the sources. For instance, in [16] Zhuge et al. present their approach for view maintenance. But as these approaches most times only deal with data updates, they are out of scope for data warehouse maintenance. Rundensteiner et al. [12] present a view maintenance approach which can also deal with changing structures. Their evolvable view management is realized as middleware between the data sources and the data warehouse. A core feature is the so called evolvable SQL which allows to define preferences for view evolution. With these preferences it is possible to redefine the view after some source changes, such that the resulting view is possibly not equivalent the to original view any more, but still fulfills the user's needs.

Key Applications

Data warehouses are often used to efficiently support the decision making process in companies and public authorities. To fulfil this task they have to represent the application domain and users' requirements. To keep the analysis results accurate and correct over the time, data warehouse maintenance is a crucial issue. Application domains which are typically vulnerable to changing structures are among others statistic and geographic applications (for instance statistical data in the European Union), health care (for instance switching from International Classification of Deceases Version 9 to Version 10), or stock market (for instance splitting stocks). In each of these domains, traceability and comparability of data over long periods of time are very important, thus effective and efficient means to provide these capabilities have to be defined.

Future Directions

Current commercial systems assume the data warehouse structure to be constant, therefore their support for modifications is rather limited. On the other hand, in real-world applications the demand for changing structures is rather high, as the data warehouse has to be consistent with the application domain and the requirements. Despite the fact that more effort is put into integrating maintenance capabilities into commercial data warehouse systems [9,7], current products are still not well prepared for this challenge.

Whereas schema and instance maintenance is quite elaborated in current research papers, the efficient transformation of cell data between different versions is still subject to research. The main problems with data transformation are first of all defining semantically correct transformation functions, and second the oftentimes huge amount of cell data which has to be handled in an efficient way.

Related to data transformation is the problem of multiversion queries. The problem with such queries is defining the desired semantics and structure of the outcome, i.e., whether and how elements and cell values, which are not valid for all affected versions should be included in the result.

Cross-references

- ▶ Data Warehousing Systems: Foundations and Architectures
- ▶ On-line Analytical Processing
- ▶ Optimization and Tuning in Data Warehouses
- ▶ Quality of Data Warehouses
- ▶ Schema Versioning
- ▶ Temporal Database
- ▶ What-If Analysis

Recommended Reading

1. Bębel B., Eder J., Koncilia C., Morzy T., and Wrembel R. Creation and management of versions in multiversion data warehouse. In Proc. 2004 ACM Symp. on Applied computing, 2004, pp. 717–723.
2. Blaschka M., Sapia C., and Höfling G. On schema evolution in multidimensional databases. In Proc. Int. Conf. on Data Warehousing and Knowledge Discovery, 1999, pp. 153–164.
3. Chamoni P. and Stock S. Temporal structures in data warehousing. In Proc. Int. Conf. on Data Warehousing and Knowledge Discovery, 1999, pp. 353–358.
4. Eder J., Koncilia C., and Morzy T. The COMET Metamodel for Temporal Data Warehouses. In Proc. Int. Conf. on Advanced Information Systems Engineering, 2002, pp. 83–99.
5. Eder J., Koncilia C., and Wiggisser K. Maintaining temporal warehouse models. In Proc. Int. Conf. on Research and Practical Issues of Enterprise Information Systems, 2006, pp. 21–30.

6. Golfarelli M., Lechtenbörger J., Rizzi S., and Vossen G. Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation. Data & Knowledge Eng., 59:435–459, 2006.
7. KALIDO Dynamic Information Warehouse: A Technical Overview. Tech. rep., Kalido, 2004.
8. Kimball R. Slowly Changing Dimensions. DBMS Magazine, 9(4):14, 1996.
9. Multi-Dimensional Modeling with BW: ASAP for BW Accelerator. Tech. rep., SAP Inc., 2000.
10. Quix C. Repository Support for Data Warehouse Evolution. In Proc. Int. Workshop on Design and Management of Data Warehouses, 1999.
11. Ravat F. and Teste O. A Temporal Object-Oriented Data Warehouse Model. In Proc. Int. Conf. on Database and Expert Systems Applications, 2000, pp. 583–592.
12. Rundensteiner E.A., Koeller A., and Zhang X. Maintaining data warehouses over changing information sources. Commun. ACM, 43(6):57–62, 2000.
13. Sarda N.L. Temporal Issues in Data Warehouse Systems. In Proc. Int. Symp. on Database Applications in Non-Traditional Environments, 1999.
14. Vaisman A. and Mendelzon A. A Temporal Query Language for OLAP: Implementation and a Case Study. In Proc. Int. Workshop on Database Programming Languages, 2001, pp. 78–96.
15. Yang J. and Widom J. Maintaining temporal views over non-temporal information sources for data warehousing. In Proc. Int. Conf. on Extending Database Technology. 1998, pp. 389–403.
16. Zhuge Y., Garcia-Molina H., Hammer J., and Widom J. View Maintenance in a Warehousing Environment. In Proc. ACM SIGMOD Int Conf. on Management of Data, 1995, pp. 316–327.

Data Warehouse Indexing

▶ Indexing of Data Warehouses

Data Warehouse Integration

▶ Interoperability in Data Warehouses

Data Warehouse Metadata

Panos Vassiliadis
University of Ioannina, Ioannina, Greece

Definition

Data warehouse metadata are pieces of information stored in one or more special-purpose *metadata repositories* that include (i) information on the contents of the data warehouse, their location and their structure, (ii) information on the processes that take place in the data warehouse back-stage, concerning the refreshment of the warehouse with clean, up-to-date, semantically and structurally reconciled data, (iii) information on the implicit semantics of data (with respect to a common enterprise model), along with any other kind of data that aids the end-user exploit the information of the warehouse, (iv) information on the infrastructure and physical characteristics of components and the sources of the data warehouse, and, (v) information including security, authentication, and usage statistics that aids the administrator tune the operation of the data warehouse as appropriate.

Historical Background

Data warehouses are systems with significant complexity in their architecture and operation. Apart from the central data warehouse itself, which typically involves an elaborate hardware architecture, several sources of data, in different operational environments are involved, along with many clients that access the data warehouse in various ways. The infrastructure complexity is only one part of the problem; the largest part of the problem lies in the management of the data that are involved in the warehouse environment. Source data with different formats, structure, and hidden semantics are integrated in a central warehouse and then, these consolidated data are further propagated to different end-users, each with a completely different perception of the terminology and semantics behind the structure and content of the data offered to them. Thus, the administrators, designers, and application developers that cooperate towards bringing clean, up-to-date, consolidated and unambiguous data from the sources to the end-users need to have a clear understanding of the following issues (see more in the following section):

1. The location of the data
2. The structure of each involved data source
3. The operations that take place towards the propagation, cleaning, transformation and consolidation of the data towards the central warehouse
4. Any audit information concerning who has been using the warehouse and in what ways, so that its performance can be tuned

5. The way the structure (e.g., relational attributes) of each data repository is related to a common model that characterizes each module of information

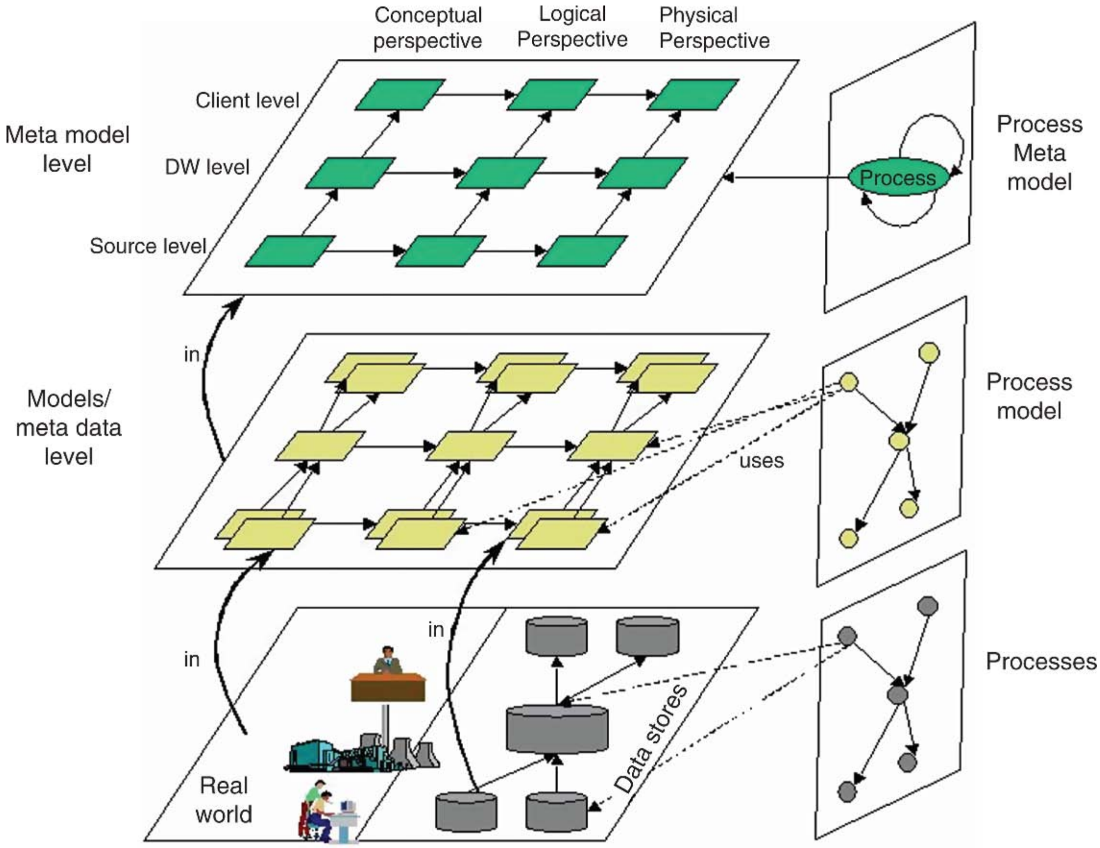
Data warehouse metadata repositories store large parts (if not all) of this kind of *data warehouse metadata* and provide a central point of reference for all the stakeholders that are involved in a data warehouse environment.

What happened was that all areas of data warehousing, ad-hoc solutions by industrial vendors and consultants were in place before the academic world provided a principled solution for the *problem of the structure and management of data warehouse metadata.* Early attempts of academic projects that related to wrapper-mediator schemes of information integration (Information Manifold, WHIPS, Squirrel, TSIMMIS – see [9] for a detailed discussion of the related literature), did not treat metadata as first-class concepts in their deliberations. At the same time, early standardization efforts from the industrial world (e.g., the MDIS standard [13]) were also poor in their treatment of the problem.

The first focused attempt towards the problem of data warehouse metadata management was made in the context of the European Project "Foundations of Data Warehouse Quality (DWQ)" [7,5]. In Fig. 1, the vertical links represent levels of abstraction: the data warehouse metadata repository, depicted in the middle layer, is an abstraction of the way the warehouse environment is structured in real life (depicted in the lowest layer of Fig. 1). At the same time, coming up with the appropriate formalism for expressing the contents of the repository (depicted in the upper layer of Fig. 1), provided an extra challenge that was tackled by [7] through the usage of the Telos language.

Foundations

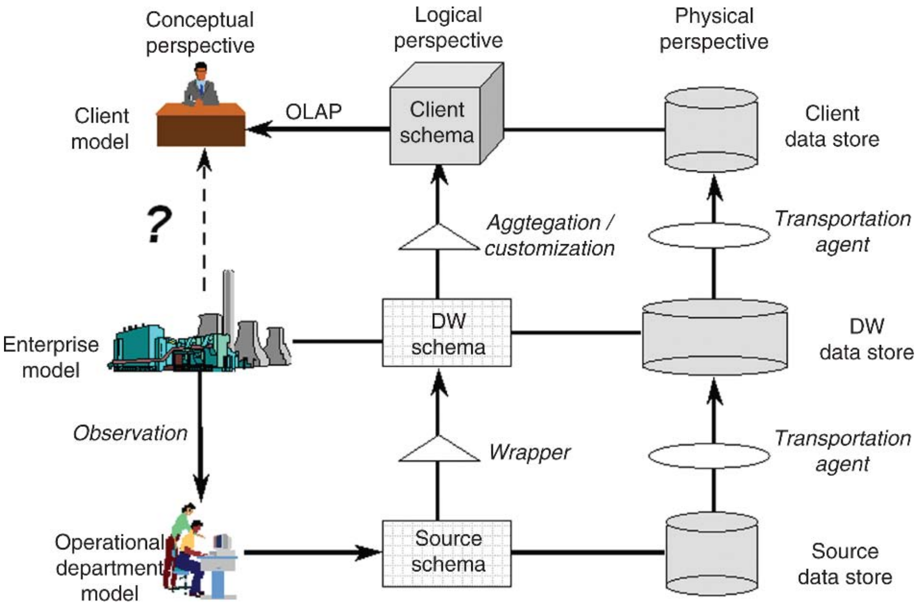
Structure of the data warehouse metadata repository. A principled approach towards organizing the structure of the data warehouse metadata repository was



Data Warehouse Metadata. Figure 1. Role and structure of a data warehouse metadata repository [12].

first offered by [7,8]. The ideas of these papers were subsequently refined in [9] and formed the basis of the DWQ methodology for the management of data warehouse metadata. The specifics of the DWQ approach are fundamentally based on the separation of data and processes and their classification in a grid which is organized in three *perspectives*, specifically the conceptual, the logical and the physical one and three *location levels*, specifically, the source, warehouse and client levels (thus the 3 × 3 contents of the middle layer of Fig. 1 and also the structure of Fig. 2). The proposal was subsequently extended to incorporate a *program versus data* classification (Fig. 1) that discriminates static architectural elements of the warehouse environment (i.e., stored data) from process models (i.e., software modules).

The *location* axis is straightforward and classifies elements as source, data warehouse and client elements. The data warehouse elements incorporate both the officially published data, contained in fact and dimension tables as well as any auxiliary data structures, concerning the Operational Data Store and the Data Staging Area. Similarly, any back-stage Extract-Transform-Clean (ETL) processes that populate the warehouse and the data marts with data are also classified according to the server in which they execute. The most interesting part of the DWQ method has to do with the management of the various *models* (a.k.a. *perspectives* in the DWQ terminology) of the system. Typically, in all DBMS's –and, thus, all deployed data warehouses- the system catalog includes both a *logical model* of the data structure (i.e., the database schema) as well as a *physical schema*, indicating the physical properties of the data (tablespaces, internal representation, indexing, statistics, etc) that are useful to the database administrator to perform his everyday maintenance and tuning tasks. The DWQ approach claimed that in a complicated and large environment like a data warehouse it is absolutely necessary to add a conceptual modeling perspective to the system that explains the role of each module of the system (be it a data or a software module). Clearly, due to the vast number of the involved information systems, each of them is accompanied by its own model, which is close enough to the perception of its users. Still, to master the complexity of all these submodels, it is possible to come up with a centralized, reference model of all the collected information (a.k.a., *enterprise model*) – exploiting, thus, the centralized nature of data warehouses. The interesting part of the method is the idea of expressing every other submodel of the warehouse as a "view" over this enterprise model. Thus, once an interested user understands the enterprise model, he/she can ultimately understand the



Data Warehouse Metadata. Figure 2. The DWQ proposal for the internal structure of the data warehouse metadata repository [4].

particularities of each submodel, independently of whether it concerns a source or client piece of data or software.

In [15], the authors discuss a coherent framework for the structuring of data warehouse metadata. The authors discriminate between back-stage *technical metadata*, concerning the structure and population of the warehouse and *semantic metadata*, concerning the front-end of the warehouse, which are used for querying purposes. Concerning the technical metadata, the proposed structure is based on (i) *entities*, comprising attributes as their structural components and (ii) an early form of schema mappings, also called *mappings* in the paper's terminology, that try to capture the semantics of the back-stage ETL process by appropriately relating the involved data stores through aggregations, joins etc. Concerning the semantic metadata, the authors treat the enterprise model as a set of *business concepts*, related to the typical OLAP metadata concerning cubes, dimensions, dimension levels and hierarchies. The overall approach is a coherent, UML-based framework for data warehouse metadata, defined at a high-level of abstraction. Specialized approaches for specific parts (like definitions of OLAP models, or ETL workflows) can easily be employed in a complementary fashion to the framework of [6] (possibly through some kind of specialization) to add more detail to the metadata representation of the warehouse. It is also noteworthy to mention that the fundamental distinction between *technical* and *business metadata* has also deeply influenced the popular, industrially related literature [11].

Contents of the data warehouse metadata repository (data warehouse metadata in detail). The variety and complexity of metadata information in a data warehouse environment are so large that giving a detailed list of all metadata classes that can be recorded is mundane. The reader who is interested in a detailed list is referred to [12] for a broader discussion of all these possibilities, and to [11] for an in depth discussion with a particular emphasis on ETL aspects (with the note that the ETL process is indeed the main provider of entries in the metadata repository concerning the technical parts of the warehouse). In the sequel, the discussion is classified in terms of data and processes.

Data. Figure 3 presents a summarized view of relevant metadata concerning the static parts of the warehouse architecture. The physical-perspective metadata are mostly related to (i) the location and naming of the information wherever data files are used and (ii) DBMS catalog metadata wherever DBMS's are used. Observe the need for efficiently supporting the end-user in his navigation through the various reports, spreadsheets and web pages (i.e., answering the question "where can I find the information I am looking for?") also observe the need to support the questions "what information is available to me anyway?" which is supported at the logical perspective for the client level. The rest of the logical perspective is also straightforward and mostly concerns the schema of data; nevertheless business rules are also part of any schema and thus data cleaning requirements and the related business rules can also be recorded at this level. The conceptual perspective involves a clear recording of the involved concepts and their intra-level mappings (source-to-DW, client-to-DW). As expected, academic efforts adopt rigorous approaches at this level [9], whereas industrial literature suggests informal, but simpler methods (e.g., see the discussion on "Business metadata" at [11]).

It is important to stress the need of tracing the mappings between the different levels and perspectives in the

Data	Source	DW	Client
Conceptual	Source model	Enterprise model	Business concepts
Logical	Schemata and/or data formats	– Schemata and/or data formats – Surrogate Key, Slowly Changing Dimension information – Data cleaning standards/specs and business rules	– Schemata of any data marts – List of available pre-canned reports and their definitions – User documentation – User profiles – Security, authentication profiles
Physical	Names of the involved data files or database installations physical properties like partitions, deployment/striping of data at disks, indexes, etc)		Map of available reports, spreadsheets, web pages

Data Warehouse Metadata. Figure 3. Metadata concerning the data of the warehouse.

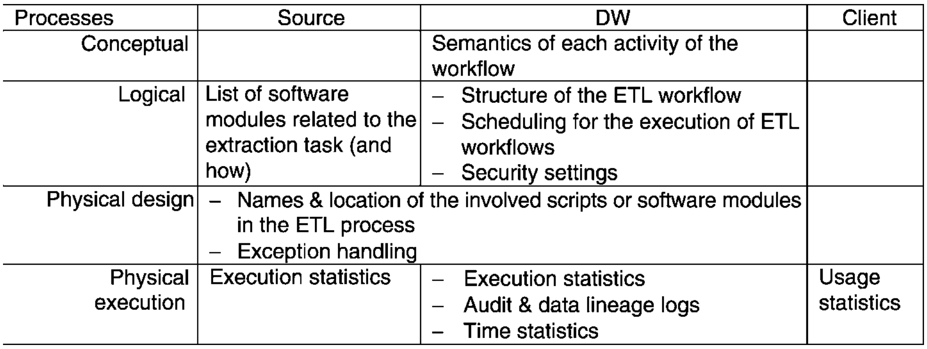
warehouse. The physical-to-logical mapping is typically performed by the DBMS's and their administrative facilities; nevertheless, the logical-to-conceptual mapping is not. Two examples are appropriate in this place: (i) the developer who constructs (or worse, maintains) a module that processes a source file of facts, has to translate cryptic code-and-value pairs (e.g., CDS_X1 = 145) to data that will be stored in the warehouse and (ii) an end-user who should see data presented with names that relate to the concepts he is familiar with (e.g., see a description "Customer name" instead of the attribute name CSTR_NAME of a dimension table). In both cases, the logical-to-conceptual mappings are of extreme importance for the appropriate construction and maintenance of code and reports.

This is also the place to *stress the importance of naming conventions* in the schema of databases and the signatures of software modules: the huge numbers of involved attributes and software modules practically enforce the necessity of appropriately naming all data and software modules in order to facilitate the maintenance process (see [11] for detailed instructions).

Processes. When the discussion comes to the metadata that concern processes, things are not very complicated again, at the high level (Fig. 4). There is a set of ETL workflows that operate at the warehouse level, and populate the warehouse along with any pre-canned reports or data marts on a regular basis. The structure of the workflow, the semantics of the activities and the regular scheduling of the process form the conceptual and logical parts of the metadata. The physical locations and names of any module, along with the management of failures form the physical part of the metadata, concerning the design level of the software. Still, it is worth noting that the physical metadata can be enriched with information concerning the execution of the back-stage processes, the failures, the volumes of processed data, clean data, cleansed or impossible-to-clean data, the error codes returned by the DBMS and the time that the different parts of the process took. This kind of metadata is of statistical importance for the tuning and maintenance of the warehouse back-stage by the administration team. At the same time, the audit information is of considerable value, since the data lineage is recorded as every step (i.e., transformation or cleaning) in the path that the data follow from the sources to their final destination can be traced.

Standards. The development of standards for data warehouse metadata has been one of the holy grails in the area of data warehousing. The standardization of data warehouse metadata allows the vendors of all kinds of warehouse-related tools to extract and retrieve metadata in a standard format. At the same time, metadata interchange among different sources and platforms –and even migration from one software configuration to another – is served by being able to export metadata from one configuration and loading it to another.

The first standardization effort came from the *MetaData Coalition (MDC)*, an industrial, non-profitable consortium. The standard was named *Meta-Data Interchange Specification (MDIS)* [13] and its structure was elementary, comprising descriptions for databases, records, dimensions and their hierarchies and relationships among them. Some years after MDIS, the *Open Information Model (OIM)* [14] followed. OIM was also developed in the context of the MetaData Coalition and significantly extends MDIS by capturing core metadata types found in the operational



Processes	Source	DW	Client
Conceptual		Semantics of each activity of the workflow	
Logical	List of software modules related to the extraction task (and how)	– Structure of the ETL workflow – Scheduling for the execution of ETL workflows – Security settings	
Physical design	– Names & location of the involved scripts or software modules in the ETL process – Exception handling		
Physical execution	Execution statistics	– Execution statistics – Audit & data lineage logs – Time statistics	Usage statistics

Data Warehouse Metadata. Figure 4. Metadata concerning the process of the warehouse.

and data warehousing environment of enterprises. The MDC OIM uses UML both as a modeling language and as the basis for its core model. The OIM is divided into sub-models, or *packages*, which extend UML in order to address different areas of information management, including database schema elements, data transformations, OLAP schema elements and data types. Some years later, in 2001, the *Object Management Group (OMG)* initiated its own standard, named *Common Warehouse Metamodel (CWM)* [4]. CWM is built on top of other standard OMG notations (UML, MOF, XMI) also with the aim to facilitate the interchange of metadata between different tools and platforms. As of 2007, CWM appears to be very popular, both due to its OMG origin and as it is quite close to the parts concerning data warehouse structure and operation. Much like OIM, CWM is built around packages, each covering a different part of the data warehouse lifecycle. Specifically, the packages defined by CWM cover metadata concerning (i) static parts of the warehouse architecture like relational, multidimensional and XML data sources, (ii) back-stage operations like data warehouse processes and operations, as well as data transformations and (iii) front-end, user-oriented concepts like business concepts, OLAP hierarchies, data mining and information visualization tasks. A detailed comparison of earlier versions of OIM and CWM can be found in [19].

Key Applications

Data Warehouse Design. Typically, the data warehouse designers both populate the repository with data and benefit from the fact that the internal structure and architecture of the warehouse is documented in the metadata repository in a principled way. [17] implements a generic graphical modeling tool operating on top of a metadata repository management system that uses the IRDS standard. Similar results can be found in [3,18].

Data Warehouse Maintenance. The same reasons with data warehouse design explain why the data warehouse administrators can effectively use the metadata repository for tuning the operation of the warehouse. In [16], there is a first proposal for the extension of the data warehouse metadata with operators characterizing the evolution of the warehouse's structure over time. A more formal approach on the problem is given by [6].

Data Warehouse Usage. Developers constructing or maintaining applications, as well as the end-users interactively exploring the contents of the warehouse can benefit from the documentation facilities that data warehouse metadata offer (refer to [11] for an example where metadata clarify semantic discrepancies for synonyms).

Data Warehouse Quality. The research on the annotation of data warehouse metadata with annotations concerning the quality of the collected data (a.k.a. *quality indicators*) is quite large. The interested reader is referred to [10,9] for detailed discussions.

Model Management. Model management was built upon the results of having a principled structure of data warehouse metadata. The early attempts in the area [1,2] were largely based on the idea of mapping source and client schemata to the data warehouse schema and tracing their attribute inter-dependencies.

Design of large Information Systems. The mental tools developed for the management of large, intra-organizational environments like data warehouses can possibly benefit other areas –even as a starting point. The most obvious candidate concerns any kind of open agoras of information systems (e.g., digital libraries) that clearly need a common agreement in the hidden semantics of exported information, before they can interchange data or services.

Cross-references

▶ CWM
▶ Data Quality
▶ Data Warehouse Life-Cycle and Design
▶ Data Warehouse
▶ MDC
▶ Metadata
▶ Metadata Repository
▶ Model Management
▶ OIM

Recommended Reading

1. Bernstein P., Levy A., and Pottinger R. A Vision for management of complex models. ACM SIGMOD Rec. 29(4):55–63, 2000.
2. Bernstein P.A. and Rahm E. Data warehouse scenarios for model management. In Proc. 19th Int. Conf. on Conceptual Modeling, 2000, pp. 1–15.
3. Carneiro L., and Brayner A. X-META: A methodology for data warehouse design with metadata management. In Proc. 4th Int. Workshop on Design and Management of Data Warehouses, 2002, pp. 13–22.
4. Common Warehouse Metamodel (CWM) Specification, version 1.1. OMG, March 2003.
5. Foundations of Data Warehouse Quality (DWQ) homepage. http://www.dblab.ece.ntua.gr/∼dwq/.

6. Golfarelli M., Lechtenbörger J., Rizzi S., and Vossen G. Schema versioning in data warehouses: enabling cross-version querying via schema augmentation. Data Knowl. Eng., 59(2):435–459, 2006.
7. Jarke M., Jeusfeld M.A., Quix C., and Vassiliadis P. 1998, Architecture and quality in data warehouses. In Proc. 10th Conf. on Advanced Information Systems Engineering, 1998. LNCS, vol. 1413, 1998, pp. 93–113.
8. Jarke M., Jeusfeld M.A., Quix C., and Vassiliadis P. Architecture and quality in data warehouses. Inf. Syst., 24(3):229–253, 1999.
9. Jarke M., Lenzerini M., Vassiliou Y., and Vassiliadis P. (eds.). Fundamentals of Data Warehouses (2nd edn.). Springer, 2003, p. 207.
10. Jeusfeld M.A., Quix C., and Jarke M. Design and analysis of quality information for data warehouses. In Proc. 17th Int. Conf. on Conceptual Modeling, 1998, pp. 349–362.
11. Kimball R. and Caserta J. The Data Warehouse ETL Toolkit. Wiley, New York, NY, 2004.
12. Kimbal R., Reeves L., Ross M., and Thornthwaite W. The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses. Wiley, 1998.
13. Metadata Coalition: Proposal for version 1.0 metadata interchange specification, 1996.
14. MetaData Coalition. Open Information Model, version 1.0 (1999).
15. Müller R., Stöhr T., and Rahm E. An integrative and uniform model for metadata management in data warehousing environments. In Proc. Int. Workshop on Design and Management of Data Warehouses, 1999.
16. Quix C. Repository support for data warehouse evolution. In Proc. Int. Workshop on Design and Management of Data Warehouses, 1999.
17. Sapia C., Blaschka M., and Höfling G. GraMMi: Using a standard repository management system to build a generic graphical modeling tool. In 33rd Annual Hawaii Int. Conf. on System Sciences, 2000.
18. Vaduva A, Kietz J-U, Zücker R. M4 - A metamodel for data preprocessing. In Proc. ACM 4th Int. Workshop on Data Warehousing and OLAP, 2001.
19. Vetterli T, Vaduva A, and Staudt M. Metadata standards for data warehousing: open information Model vs. Common warehouse metamodel. ACM SIGMOD Rec., 29(3):68–75, 2000.

Data Warehouse Query Processing

▶ Query Processing in Data Warehouses

Data Warehouse Refreshment

▶ Extraction, Transformation and Loading

Data Warehouse Security

Carlos Blanco[1], Eduardo Fernández-Medina[1], Juan Trujillo[2], Mario Piattini[1]
[1]University of Castilla-La Mancha, Ciudad Real, Spain
[2]University of Alicante, Alicante, Spain

Synonyms

Secure data warehouses; Data warehouses confidentiality

Definition

Security, as is stated in the ISO/IEC 9126 International Standard, is one of the components of software quality. Information Security can be defined as the preservation of confidentiality, integrity and availability of information [5], in which confidentiality ensures that information is accessible only to those users with authorization privileges. Integrity safeguards the accuracy and completeness of information and process methods, and availability ensures that authorized users have access to information and associated assets when required. Other modern definitions of Information Security also consider properties such as authenticity, accountability, non-repudiation, and reliability. Therefore, Data Warehouse (DW) Security is defined as the mechanisms which ensure the confidentiality, integrity and availability of the data warehouse and its components. Confidentiality is especially important once the Data Warehouse has been deployed, since the most frequent operations that users perform are SQL and OLAP queries, and therefore the most frequent security attack is against the confidentiality of data stored in the data warehouse.

Historical Background

Considering that DWs are the basis of companies' decision making processes, and due to the fact that they frequently contain crucial and sensitive internal information, and that DWs are usually managed by OLAP tools, most of the initial approaches to data warehouse security were focused on the definition and enforcement of access control policies for OLAP tools [6,10], taking into consideration one of the most traditional access control models (Discretional Access Control) and also managing the concept of role defined as subject. Other approaches dealt with real implementation in specific commercial tools by using multidimensional elements [10]. Indirect access and cover channel problems have also been detected in Statistical Databases but an entirely satisfactory solution has not yet been found.

Moreover, data stores in DWs come from heterogeneous data sources, which must be integrated, thus provoking various security problems. However, few works dealing with the security defined in data sources (which tackle the problem of merging different security measures established in each source) appear to exist. This problem has, nevertheless, been addressed in the field of Federated Databases, and some authors have used this parallelism to propose an architecture for developing Data Warehouses through the integration of Multilevel Access Control (MAC) policies defined in data sources [12]. Furthermore, ETL processes have to load the information extracted and transformed from the data sources into the Data Warehouse, but these processes do not consider security issues and must use the security defined in the data source and add new security measures for the detected lacks of security. In this field, the proposed works focus solely upon ETL model processes, and do not consider security issues.

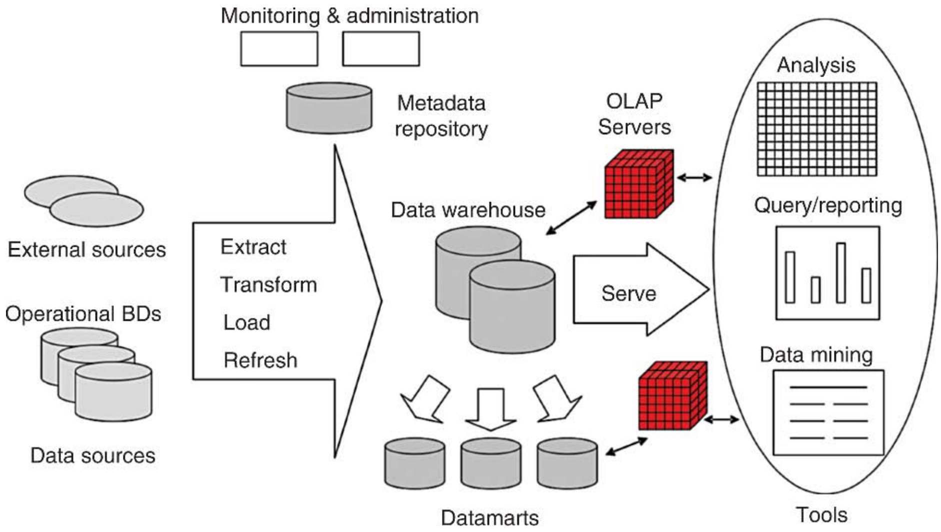
In recent decades, the development of DWs has evolved from being a handmade method, to being a more engineering-based method, and several approaches have been defined for the conceptual modeling of DWs, e.g., [4,8]. Unfortunately none of these proposals has considered security issues. However, one of these approaches has recently been extended to propose a Model Driven Multidimensional approach for developing secure DWs [1]. This approach permits the inclusion of security requirements (audit and access control) from the first stages of the DWs life cycle, and it is possible to automatically generate code for different target platforms through the use of model transformation. The scientific community demands the integration of security engineering and software engineering in order to ensure the quality and robustness of the final applications [9], and this approach fulfills this demand.

Foundations

The DW development process follows the scheme presented in Fig. 1. Therefore, security should be considered in all stages of this process by integrating the existing security measures defined in data sources, considering these measures in ETL processes, defining models that represent security constraints at a high level of abstraction and finally, enforcing these security constraints in the OLAP tools in which the DW is deployed.

Security in Data Sources

In DWs architecture, data coming from heterogeneous data sources are extracted, cleaned, debugged, and stored. Once this process is completed, the DW will be composed of these stored and integrated data, with which users will be able to discover information in strategic decision making processes. Data sources are heterogeneous, can use different representation models (relational databases, object-orientated databases, XML files, etc.), and may or may not have associated



Data Warehouse Security. Figure 1. Data warehouse architecture.

security policies. Although DW users will be different from data sources, these security policies should be considered and integrated into the DW security design.

Data source security can be defined by using various security policies such as Discretional Access Control (DAC) which restricts access to objects based on the identity of subjects with a certain access permission: Mandatory Access Control (MAC), which restricts access to objects based on the sensitivity of the information contained in the objects and the formal authorization of subjects to access information of such sensitivity; or Role-Based Access Control (RBAC), an approach which restricts system access to authorized users by assigning permissions to perform certain operations to specific roles. The integration of these policies presents a problem which has been studied in Federated Databases [12]. Some research efforts have been made to integrate different multilevel policies in a semi-automatic manner by using a schema integration process which obtains the ordered set and the translation functions between each ordered set belonging to each component database and the federated ordered set. In addition, the integration of different role-based policies has been dealt with by representing role configurations as role graphs and by applying techniques of graph integration to obtain the final role configuration. Other authors, such as Rosenthal and Sciore [11], have applied inference mechanisms to data sources in order to obtain access control policies and have used them to set up DW security.

After considering the parallelism between DW and Federated Information Systems (FIS), Saltor et al. [12] propose a seven layered architecture for preserving and integrating the multilevel security established in data sources. This architecture extends the five layered architecture developed for FIS, including two schemas: "authorization schemas" for each authorization level and "external schemas" with which to represent multilevel security information of the data sources in a Canonical Data Model (CDM). These "external schemas" with security information will later be used to obtain DW and Data Marts (DM) schemas.

Security in ETL Processes

ETL (Extraction-Transformation-Loading) processes participate in the first stage of acquisition and extract information from heterogeneous data sources, debug it, integrate it and finally, load it into data warehouses following their previously defined design.

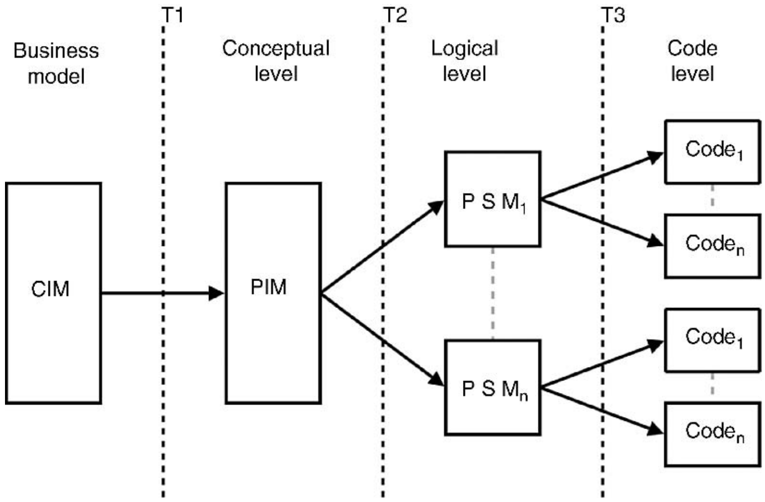
It is necessary to define security measures in ETL processes, in order to both use, adapt and integrate the security measures defined in the data sources and to add new security measures for the possibly detected lacks of security. At present, despite the existence of proposals with which to model ETL processes which can be extended to include security issues, none of the said proposals include the aforementioned concepts.

Some interesting works on the modeling of ETL processes exist, but they do not deal with security issues. Vassiliadis and Simitsis use their own graphic notation for modeling ETL processes at a conceptual level, propose how to transform these conceptual designs into logical designs [13], and define a framework for designing and maintaining ETL processes (ARKTOS). Trujillo and Luján-Mora [15] model ETL processes by using the UML notation and OCL to establish constraints. Their proposal does not take attributes into consideration but simplifies the design and maintenance processes, and the use of UML and OCL provides one with possibilities which greatly simplify the extension of this model with security.

Security in Data Warehouses Modeling

Multidimensional modeling is the foundation of DWs, Multidimensional Databases and On-Line Analytical Processing Applications (OLAP) and is different from traditional database modeling in that it is adapted to the characteristics of these approaches. Despite the quantity of interesting background on security measures and access control models specified for relational databases which is available, it cannot be directly applied as it is not appropriate for DWs. Both are models but they are based on different concepts. Relational security measures use terms of database tables, rows and columns, and DW security uses multidimensional terms of facts, dimensions or classification hierarchy. Several modeling proposals specifically created for DWs consider their properties, but none use standard notations or include security issues, e.g., [4,8].

A model driven multidimensional modeling approach for developing secure DWs has been proposed by Fernández-Medina et al. [1]. This approach proposes a Query/Views/Transformations (QVT) and Model-Driven Architecture (MDA) based approach (see Fig. 2). This aligns MDA with the DWs development process, considering multidimensional models as being PIM, logical models (such as ROLAP, MOLAP and HOLAP) as being Platform-Specific Model (PSM),



Data Warehouse Security. Figure 2. Model driven architecture.

and the DBMS and OLAP tools as being the target platforms. This proposal is made up of a security model (access control and audit model) for DW [2], an extension of UML for modeling secure multidimensional models [3] as Platform-Independent Models (PIM), and an extension of the Common Warehouse Metamodel (CWM) [14] as a Platform-Specific Model (PSM). This proposal is currently being extended within the extremes of MDA architecture: the Computational-Independent Model (CIM) level is being defined through an extension of i* which defines security goals and subgoals, and the code generation is being carried out by considering Oracle, SQL Server Analysis Services, and Pentaho as target platforms of the architecture.

Security in OLAP Tools

OLAP systems are mechanisms with which to discover business information and use a multidimensional analysis of data to make strategic decisions. This information is organized according to the business parameters, and users can discover unauthorized data by applying a set of OLAP operations to the multidimensional view. Therefore, it is of vital importance for the organization to protect its data from unauthorized accesses including security constraints in OLAP systems which take these OLAP operations (roll-up, drill-down, slice-dice and pivoting) into account, and from indirect accesses (inferences) which use parallel navigation, tracker queries, etc. The inference problem is an important security problem in OLAP which has yet to be solved and which can be studied by using the existing parallelism with Statistical Databases. Several solutions to the inference problem have been applied. Various solutions to the problem of controlling inference exist, such as the perturbation of data or the limitation of queries, but these imply a large amount of computational effort. On the other hand the establishment of security constraints at cell level allows one to control inferences without this lack of efficiency.

Several works attempting to include security issues in OLAP tools by implementing the previously defined security rules at a conceptual level have been proposed, but these works focus solely upon Discretional Access Control (DAC) and use a simplified role concept implemented as a subject. For instance, Katic et al. [6] proposed a DWs security model based on metamodels which provides one with views for each user group and uses Discretional Access Control (DAC) with classification and access rules for security objects and subjects. However, this model does not allow one to define complex confidentiality constraints. Kirkgöze et al. [7] defined a role-based security concept for OLAP by using a “constraints list” for each role, and this concept is implemented through the use of a discretional system in which roles are defined as subjects.

Priebe and Pernul later proposed a security design methodology, analyzed security requirements, classifying them into basic and advanced, and dealt with their implementation in commercial tools. First, in [10] they used adapted UML to define a Discretional Access Control (DAC) system with roles defined as subjects at a conceptual level. They then went on to implement this in Microsoft Analysis Services (SQL Server 2000) by using Multidimensional Expressions (MDX). They created a Multidimensional Security Constraint Language (MDSCL) based on MDX and put forward HIDE statements with which to represent negative authorization constraints on certain multidimensional elements: cube, measure, slice, and level.

Key Applications

DWs security is a highly important quality aspect of a DW, which must be taken into account at all stages of the development process. If security measures are not established, then unauthorized users may obtain the business information used for making strategic decisions which is vital to the survival of the organization. DWs security has to be considered in all the fields involved. These are, principally, the following: the application of techniques through which to integrate different kinds of security policies detected in the data sources; the definition of models, which permit the establishment of security constraints at upper abstraction levels; and the study of the final implementation of the defined security measures in OLAP tools in order to protect information from malicious operations such as navigations or inferences.

Cross-references

- ▶ Data Warehousing Systems: Foundations and Architectures
- ▶ Extraction
- ▶ Multidimensional Modeling
- ▶ On-Line Analytical Processing
- ▶ Transformation and Loading

Recommended Reading

1. Fernández-Medina E., Trujillo J., and Piattini M. Model driven multidimensional modeling of secure data warehouses. Eur. J. Inf. Syst., 16:374–389, 2007.
2. Fernandez-Medina E., Trujillo J., Villarroel R., and Piattini M. Access control and audit model for the multidimensional modeling of data warehouses. Decis. Support Syst., 42(3):1270–1289, 2006.
3. Fernandez-Medina E., Trujillo J., Villarroel R., and Piattini M. Developing secure data warehouses with a UML extension. Inf. Syst., 32(6):826–856, 2007.
4. Golfarelli M., Maio D., and Stefano R. The dimensional fact model: a conceptual model for data warehouses. Int. J. Coop. Inf. Syst., 7(2–3):215–247, 1998.
5. ISO27001, ISO/IEC 27001 Information technology – Security techniques – Information security management systems – Requirements, 2005.
6. Katic N., Quirchmayr G., Schiefer J., Stolba M., and Tjoa A. 1A prototype model for DW security based on metadata. In Proc. Ninth Int. Workshop on Database and Expert Systems Applications, 1998, p. 300.
7. Kirkgöze R., Katic N., Stolda M., and Tjoa A. A security concept for OLAP. In Proc. 8th Int. Workshop on Database and Expert System Applications, 1997, p. 0619.
8. Lujan-Mora S., Trujillo J., and Song I.-Y. A UML profile for multidimensional modeling in data warehouses. Data Knowl. Eng., 59(3):725–769, 2006.
9. Mouratidis H. and Giorgini P. Integrating Security and Software Engineering: Advances and Future Visions. Idea Group, Hershey, PA, 2006.
10. Priebe T. and Pernul G. A pragmatic approach to conceptual modeling of OLAP security. In Proc. 20th Int. Conf. on Conceptual Modeling, 2001, pp. 311–324.
11. Rosenthal A. and Sciore E. View security as the basis for data warehouse security. In Proc. 2nd Int. Workshop on Design and Management of Data Warehouses, 2000, p. 8.
12. Saltor F., Oliva M., Abelló A., and Samos J. Building secure data warehouse schemas from federated information systems. In Heterogeneous Information Exchange and Organizational Hubs, D.T. Bestougeff (ed.). Kluwer Academic, 2002.
13. Simitsis A. and Vassiliadis P. A method for the mapping of conceptual designs to logical blueprints for ETL processes. Decis. Support Syst., 45(1):22–40, 2007.
14. Soler E, Trujillo J., Fernández-Medina E., and Piattini M. SECRDW: an extension of the relational package from CWM for representing secure data warehouses at the logical level. In Proc. 5th Int. Workshop on Security in Information Systems. 2007, pp. 245–256.
15. Trujillo J. and Luján-Mora S. A UML based approach for modeling ETL processes in data warehouses. In Proc. 22nd Int. Conf. on Conceptual Modeling, 2003, pp. 307–320.

Data Warehousing for Clinical Research

SHAWN MURPHY
Massachusetts General Hospital, Boston, MA, USA

Synonyms

Clinical research chart

Definition

The clinical data warehouse allows rapid querying and reporting across patients. It is used to support the discovery of new relationships between the cause and effects of diseases, and to find specific patients that qualify for research studies.

Historical Background

In healthcare, the term "data warehouse" is generally reserved for those databases optimized for analysis and integrated queries across patient populations. This is as opposed to the transactional database, which is optimized for rapid updating and highly specific kinds of retrieval (like those based upon a specific patient identifier).

There appear to be three fundamentally different approaches to organizing the healthcare data warehouse. The first is to extract tables from the transaction systems of the healthcare organization and load them into the database platform of the data warehouse with minimal transformation of the data model. The codes present in the columns are usually transformed to make them compatible with codes from other systems. For example, an ICD9 diagnosis code stored as "27.60" in one system may be transformed to a common format of 02760. However, the tables are left in essentially the same schema as the transaction system [2].

The second approach is more ambitious, where not just the codes from different systems are transformed to look the same, but the data is transformed to look the same as well. The diverse data coming from different systems must be made to fit into new tables. This involves a considerable amount of data transformation, but queries against the warehouse are then much less complex [1]. This is the approach that will be described.

The third approach is to keep the data located at its source in a "distributed" data warehouse. Queries are distributed to the local databases across a network. This strategy can be successful when patients have all of their data contained within one of the local systems (such as when systems exist in distant cities). However, if a single patient's data is distributed across many of these local databases, detailed data would need to travel across the network to be accumulated in the internal processing structures of a central CPU to allow the execution of query plans. This will have a severe negative impact on the performance of these types of systems.

Foundations

Database Design for Clinical Research Data Warehouse

The clinical data warehouse allows rapid querying and reporting across patients, which unexpectedly is not available in most clinical transaction systems. Rather, transaction systems are optimized for lookups, inserts, updates, and deletes to a single patient in the database. Transactions usually occur in small packets during the day, such as when a patient's lab test is sent to the database. Transaction systems are usually updated by small bits of data at a time, but these bits come in at the rate of thousands per second. Therefore the typical clinical database used for patient care must be optimized to handle these transactions [2].

Because the clinical data warehouse does not need to handle high volumes of transactions all day long, the data warehouse can be optimized for rapid, cross patient searching. For optimal searching of a database it is best to have very large tables. These can be indexed such that a single index allows a global search. So when one designs a clinical data warehouse, one adopts a few tables that can hold nearly all the available data. The way to hold many forms of healthcare data in the same table is by the classical entity-attribute-value schema (or EAV for short) [4,5].

The EAV schema forces one to define the fundamental fact of healthcare [2]. The fundamental fact of healthcare will be the most detailed rendition possible of any healthcare observation as reported from the data warehouse. This can be defined as an observation on a patient, made at a specific time, by a specific observer, during a specific event. The fact may be accompanied by any number of values or modifiers. Each observation is tagged with a specific concept code, and each observation is entered as a row in a "fact table." This fact table can grow to billions of rows, each representing an observation on a patient. The fact table is complimented by a least an event table, a patient table, a concept table, and an observer table [4].

The Patient table is straightforward. Each row in the table represents a patient in the database. The table includes common fields such as gender, age, race, etc. Most attributes of the patient dimension table are discrete (i.e., Male/Female, Zip code, etc.) or relevant dates.

The Event table represents a "session" where observations were made. This "session" can involve a patient directly such as a visit to a doctor's office, or it can involve the patient indirectly such as running several tests on a tube of the patient's blood. Several observations can be made during a visit. Visits have a start and end date-time. The visit record also contains specifics about the location of the session, like which hospital or clinic the session occurred, and whether the patient was an inpatient or outpatient at the time of the visit.

The Observer table is a list of observers. Generally, each row in the observer dimension represents a provider at an institution, but more abstractly, it may be an observing machine, such as an Intensive Care Unit continuous blood pressure monitor.

The Concept table is the key to understanding how to search the fact table. A concept specifies exactly what observation was made on the patient and is being represented in a particular row of the fact table. A code is used to represent the concept in the fact table, and the concept table links if to a human-readable description of the code (Fig. 1).

Metadata Management in Clinical Research Data Warehouse

When looking at rows in the concept table, one is introduced to Metadata. Metadata is everywhere in a data warehouse. It represents data about the data, and is where medical knowledge is represented in the clinical data warehouse. The primary form of representation is in the groupings of terms so they can be queried as groups of similar concepts. The terms are grouped into hierarchies, each level up usually expressing a more general medical concept.

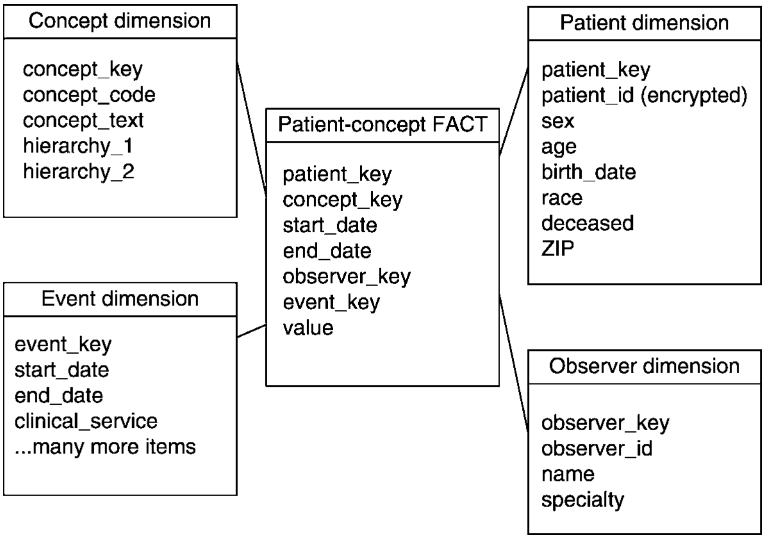
Many diverse concepts about a patient can exist in the fact table. In a clinical data warehouse, typically 100–500 thousand different concepts exist. All sorts of concepts including ICD-9 codes (International Classification of Diseases 9th Edition, most common codes used in hospitals to classify diagnoses), CPT codes (Current Procedural Terminology, most common codes used in hospitals to classify procedures), NDC codes (National Drug Codes, most common codes used in hospitals to classify medication), and LOINC codes (Logical Observation Identifiers Names and Codes, most common codes used in hospitals to classify laboratory tests) as well as numerous local coding systems are used to describe the patient. The challenge is maintaining and updating the classification of the concepts. This classification needs to seamlessly absorb new codes, and be back-compatible to old coding and classification systems.

The organization of concepts hierarchically allows the user to navigate and use the concepts in a query. Like a file path in the Windows Explorer, the path of the hierarchy indicates in which groups the concept belongs, with the most general group being listed on the far left and each group to the right of that growing more and more specific.

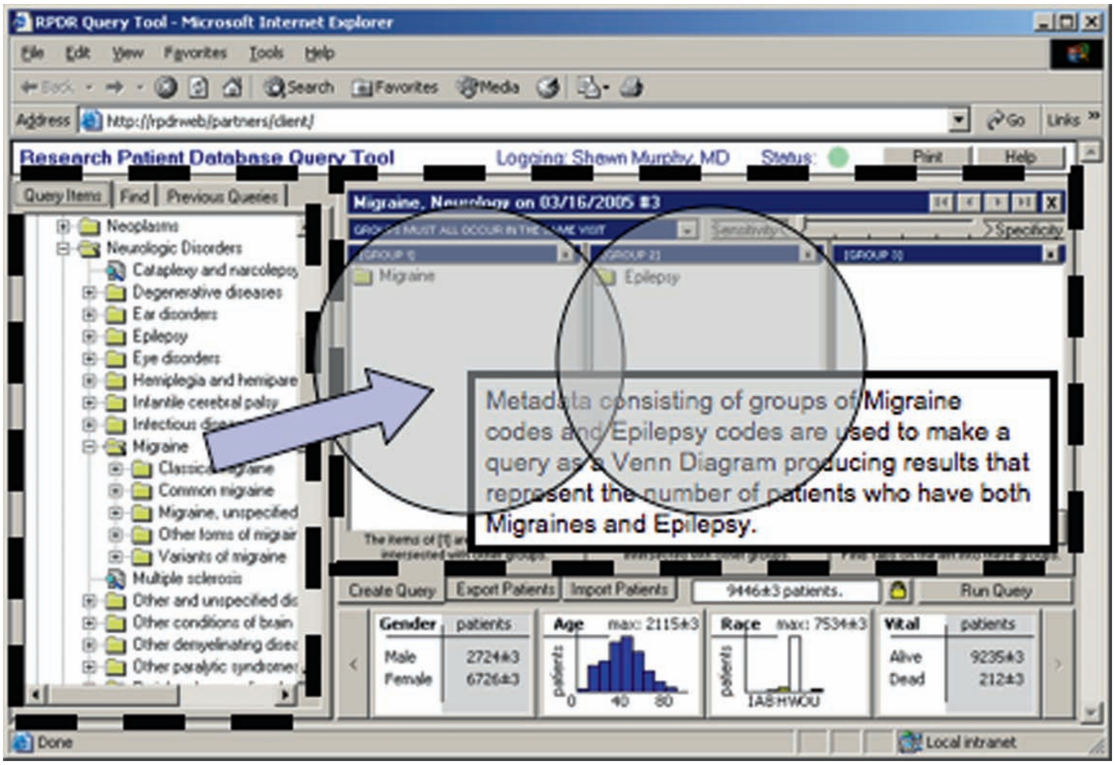
An interface to present this concept representation is shown below (Fig. 2). The use of this interface has been described in detail [3], but is essentially a way of building queries using concepts represented in the concept and provider dimension tables.

Privacy Management in the Clinical Research Data Warehouse

The clinical data warehouse should be built with patient privacy in mind. The most common strategy is to separate the data warehouse into two databases. The clinical



Data Warehousing for Clinical Research. Figure 1. Optimal star schema database design for healthcare data warehouse.



Metadata consisting of groups of Migraine codes and Epilepsy codes are used to make a query as a Venn Diagram producing results that represent the number of patients who have both Migraines and Epilepsy.

Data Warehousing for Clinical Research. Figure 2. Construction of query using the metadata from a healthcare data warehouse.

data goes into one database, and the identifiers of the patients go into a second database. Access to the second, identified, database is strictly controlled, and only accessed during data loading and the building of the data marts. The patients are given codes in the clinical database and these codes can only be looked up in the identified database. In this way, customers can use the clinical database and not have access to the patient identifiers.

Data Flow in Clinical Research Data Warehouse

Data generally flows into the data warehouse by loading it from the transaction systems, or by receiving a duplicate feed of data that are going into the transaction systems. Data are usually loaded from the transaction systems once it is provided as large "data dumps," or downloads. Transaction systems may contain many millions of records, but with current technology they can usually be written out in their entirety in just hours. Reloading all this data into the data warehouse similarly takes only a few hours, and the simplicity of this model, as opposed to the complexity of update models, often makes this a much more desirable process. The risk of an update process is that errors in update flags will cause the data warehouse to become desynchronized with the transaction system. To note many transaction systems do not have a way to provide updates and a full "data dump" is all that is possible from the transaction system.

When the data is loaded from the transaction systems, it is usually first loaded to a "staging area." As previously discussed, the data structure usually differs considerably between the transaction system and the data warehouse. Loading the transaction data into a staging area allows the data to be studied and quality assured before introducing the complexity of transforming the data into the format of the data warehouse. Because the teams from the transaction systems are usually very familiar with the data in this form, it is desirable to have the transaction team responsible for their corresponding staging area, and allow them to transfer and load the data into this area.

The data warehouse will usually distribute data back to the data consumers as a "data mart." These are subsets of the data from the data warehouse. The advantage of this approach is that the data can be prepared per request in a consumer-friendly format.

Attempting to allow customers to query the clinical data warehouse using Structured Query Language (SQL) is rarely successful. The EAV scheme is notoriously unfriendly to the causal user of data [5]. Furthermore, the metadata exists in tables that are not obviously connected to the patient data, so that tables in the data warehouse often contain no humanly readable content. Finally, the data in the data warehouse is often updated once every day and so analysis would need to go against constantly shifting data. The result is that the data is often exported into a user friendly data mart. This also limits the set of patients that a customer can view, which is important from the patient privacy point-of-view.

Key Applications

This clinical research data warehouse allows researchers to quickly obtain information that can be critical for winning corporate and government sponsored research grants, and easily gather data on patients identified for research studies. It allows clinical data to be available for research analysis where security and confidentiality are an integral part of the design, bringing clinical information to researchers' fingertips while controlling and auditing the distribution of patient data within the guidelines of the Institutional Review Boards. It also serves as a "building-block" that enables high-throughput use of patient data in some of the following applications:

1. *Bayesian inference engines.* Bayesian inference can be used to synthesize many diverse observations into fundamental atomic concepts regarding a patient. For example, a code may be assigned to a patient from several sources indicating that a patient has a disease such as diabetes. Some sources may indicate the patient has type I diabetes, while others indicate the patient has type II diabetes. Since these two types of diabetes are mutually exclusive, it is clear that one of the sources is in error. A determination of the true diagnosis can be estimated by assigning a prior probability to each source as to how often it contains correct information, and use these probabilities, to calculate the likelihood of each diagnosis.

2. *Clinical trials performed "in-silico."* Performing an observational phase IV clinical trial is an expensive and complex process that can be potentially modeled in a retrospective database using groups of patients available in the large amounts of highly organized medical data. This application would allow a formalized way of discovering new knowledge from medical databases in a manner that is well accepted by the medical community. For example, a prospective trial examining the potential harm of Vioxx would entail recruiting large numbers of patients and several years of observation. However, an in-silico clinical trial would entail setting up the database to enroll patients into a patient set automatically when they are given a prescription for Vioxx and watching them for adverse events as these events are entered in the course of clinical care. Besides requiring fewer resources, these trials could be set up for thousands of medications at a time and thereby provide a much greater scope of observational trials.

3. *Finding correlations within data.* When multiple variables are measured for each patient in a data set, there exists an underlying relationship between all pairs of variables, some highly correlated and some not. Correlations between pairs of variables may be discovered with this application, leading to new knowledge, or further insight into known relationships. Unsupervised techniques using Relevance Networks and Mutual Information algorithms can generate hypothesis from secondary observed correlations in the data. This is a way to exploit existing electronic databases for unsupervised medical knowledge discovery without a prior model for the information content. Observations collected within labs, physical examinations, medical histories, and gene expressions can be expressed as continuous variables describing human physiology at a point in time. For example, the expression of RNA found within a tumor cell may be found to correlate with the dose of effective chemotherapy for that tumor. This would allow future tumors to have their RNA expression determined and matched to various chemotherapies, and the chemotherapy found to correlate most with that gene expression would be chosen as the agent for that individual.

Cross-references

► Health Informatics
► Data Integration in Web Data Extraction System
► Data Mining
► Data Models

Recommended Reading

1. Inmon W.H. Building the Data Warehouse, 2nd edn. Wiley, NY, 1996.
2. Kimball R. The Data Warehousing Toolkit. Wiley, NY, 1997.
3. Murphy S.N., Gainer V.S., and Chueh H. A visual interface designed for novice users to find research patient cohorts in

a large biomedical database. In Proc. AMIA Annu. Fall Symp., 489–493, 2003.

4. Murphy S.N., Morgan M.M., Barnett G.O., and Chueh H.C. Optimizing healthcare research data warehouse design through past COSTAR query analysis. In Proc. AMIA Fall Symp., 892–896, 1999.
5. Nadkarni P.M. and Brandt C. Data extraction and ad hoc query of an entity-attribute-value database. J. Am. Med. Inform. Assoc., 5:511–517, 1998.

Data Warehousing Systems: Foundations and Architectures

Il-Yeol Song
Drexel University, Philadelphia, PA, USA

Definition

A data warehouse (DW) is an integrated repository of data for supporting decision-making applications of an enterprise. The most widely cited definition of a DW is from Inmon [3] who states that "a data warehouse is a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions."

Historical Background

DW systems have evolved from the needs of decision-making based on integrated data, rather than an individual data source. DW systems address the two primary needs of enterprises: data integration and decision support environments. During the 1980s, relational database technologies became popular. Many organizations built their mission-critical database systems using the relational database technologies. This trend proliferated many independent relational database systems in an enterprise. For example, different business lines in an enterprise built separate database systems at different geographical locations. These database systems improved the operational aspects of each business line significantly. Organizations, however, faced the needs of integrating the data which were distributed over different database systems and even the legacy database systems in order to create a central knowledge management repository. In addition, during the 1990s, organizations faced increasingly complex challenges in global environments. Organizations realized the need for decision support systems that can analyze historical data trends, generate sophisticated but easy-to-read reports, and react to changing business conditions in a rapid fashion. These needs resulted in the development of a new breed of database systems that can process complex decision-making queries against integrated, historical, atomic data. These new database systems are now commonly called data warehousing systems because they store a huge amount of data – much more than operational database systems – and they are kept for long periods of time. A data warehousing system these days provides an architectural framework for the flow of data from operational systems to decision-support environments. With the rapid advancement in recent computing technologies, organizations build data warehousing systems to improve business effectiveness and efficiency. In a modern business environment, a data warehousing system has emerged as a central component of an overall business intelligence solution in an enterprise.

Foundations

OLTP vs. Data Warehousing Systems

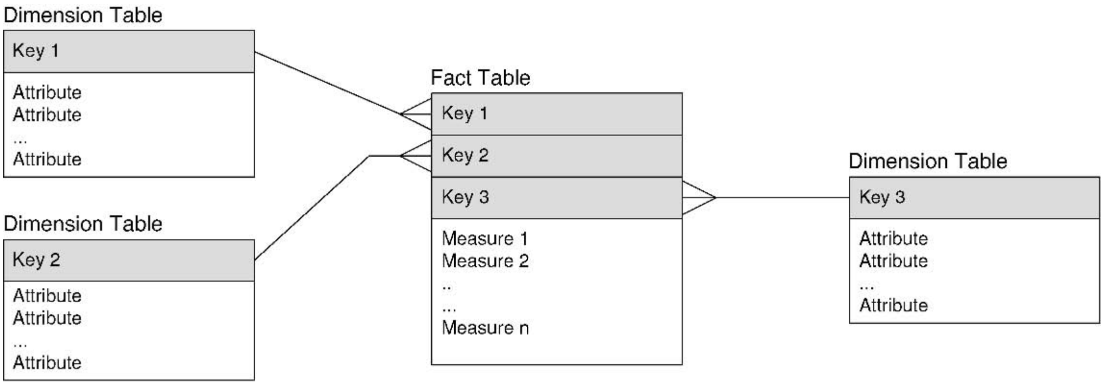
Data warehousing systems contain many years of integrated historical data, ending up storing a huge amount of data. Directly storing the voluminous data in an operational database system and processing many complex decision queries would degrade the performance of daily transaction processing. Thus, DW systems are maintained separately from operational databases, known as online transaction processing (OLTP) systems. OLTP systems support daily business operations with updatable data. In contrast, data warehousing systems provide users with an environment for the decision-making process with read-only data. Therefore, DW systems need a query-centric view of data structures, access methods, implementation methods, and analysis methods. Table 1 highlights the major differences between OLTP systems and data warehousing systems.

Rolap and Molap

The data in a DW are usually organized in formats made for easy access and analysis in decision-making. The most widely used data model for DWs is called the dimensional model or the star schema [6]. A dimensional model consists of two types of entities–a fact table and many dimensions. A *fact* table stores transactional or factual data called *measures* that get analyzed. Examples of fact tables are *Order*, *Sale*, *Return*, and *Claim*. A dimension represents an axis that analyzes the fact data. Examples of

Data Warehousing Systems: Foundations and Architectures. Table 1. A comparison between OLTP and data warehousing systems

	OLTP	Data warehouse & OLAP
Purpose	Daily business support	Decision support
	Transaction processing	Analytic processing
User	Data entry clerk, administrator, developer	Decision maker, executives
DB design	Application oriented	Subject-oriented
DB design model	ER model	Star, snowflake, Multidimensional model
Data structures	Normalized, Complex	Denormalized
		Simple
Data redundancy	Low	High
Data contents	Current, up-to-date operational data	Historical
	Atomic	Atomic and summarized
Data integration	Isolated or limited integration	Integrated
Usage	Repetitive, Routine	Ad-hoc
Queries	Predictable, predefined	Unpredictable, Complex, long queries
	Simple joins	
	Optimized for small transactions	Optimized for complex queries
Update	Transactions constantly generate new data	Data is relatively static
		Often refreshed weekly, daily
Access type	Read/update/delete/insert	Read/append mostly
Number of Records per access	Few	Many
Concurrency level	High	Low
Data retention	Usually less than a year	3–10 years or more
Response time	Subsecond to second	Seconds, minutes, worse
Systems Requirements	Transaction throughput, Data consistency	Query throughput, Data accuracy



Data Warehousing Systems: Foundations and Architectures. Figure 1. The typical structure of the star schema.

dimensions are *Time*, *Customer*, *Product*, *Promotion*, *Store*, and *Market*. Since a DW contains time-variant data, the Time dimension is always included in dimensional schemas and the data in a fact table are organized by a unit of time. An extensive list of dimensions commonly found in DWs including those dimensions used in [1,6] are presented in [4]. A typical structure of the dimensional model is illustrated in Fig. 1.

Syntactically, all the dimensions are connected with the fact table by one-to-many relationships. Thus, when a dimension has a many-to-many relationship with the fact table, a special technique such as an intersection table should be used. All the dimensions have a surrogate key, which establishes an identifying relationship with the fact table. In a star schema, all the dimensions are usually denormalized to simplify the query structure in order to minimize the number of joins. When dimensions are normalized into the third normal form, the schema is called a snowflake schema [6].

A dimensional model simplifies end-user query processing by simplifying the database structure with a few well-defined join paths. Conceptually, a dimensional model characterizes a business process with the fact table, the dimensions, and the measures involved in the business process. The dimensional model allows users of a DW to analyze the fact data from any combination of dimensions. The structure provides a multidimensional analysis space within a relational database.

Interactive data analysis of the data in a DW environment is called online analytic processing (OLAP). When the data in a dimensional model is stored in a relational database, the analysis is called relational online analytic processing (ROLAP). ROLAP engines extend SQL to support dimensional model schema and advanced OLAP functions.

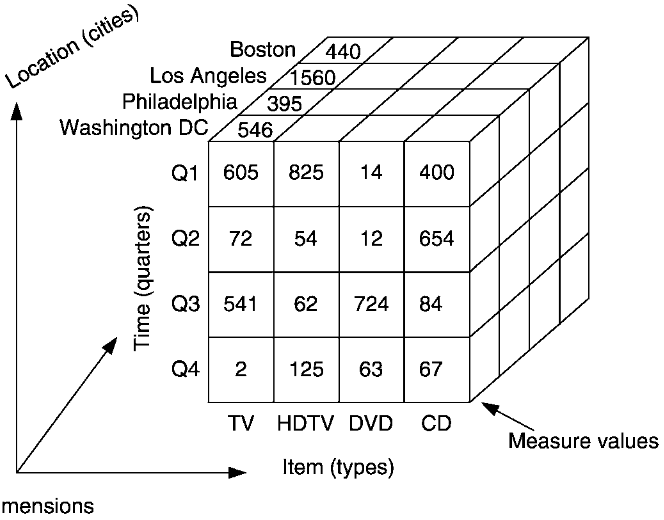
DW data can also be stored in a specialized multidimensional structure called a data cube or a hypercube. Data analysis of the data stored in a data cube is called multidimensional OLAP (MOLAP). Compared with ROLAP engines, MOLAP engines are usually limited in data storage, but provide more efficient OLAP processing by taking advantage of the multidimensional data cube structure. A typical structure of a data cube is illustrated in Fig. 2.

Hybrid OLAP (HOLAP) servers take advantage of both ROLAP and MOLAP technologies. They usually store large volumes of detailed data in a ROLAP server and store aggregated data in a MOLAP server.

Data Warehousing Architecture

A data warehousing system is an environment that integrates diverse technologies into its infrastructure. As business data and analysis requirements change, data warehousing systems need to go through an evolution process. Thus, DW design and development must take growth and constant change into account to maintain a reliable and consistent architecture. A DW architecture defines an infrastructure by which components of DW environments are organized. Figure 3 depicts the various components of a typical DW architecture that consists of five layers – data source systems, ETL management services, DW storage and metadata repository, data marts and OLAP engines, and front-end tools.

Data Source Systems The data source system layer represents data sources that feed the data into the DW. An enterprise usually maintains many different databases or information systems to serve different OLTP



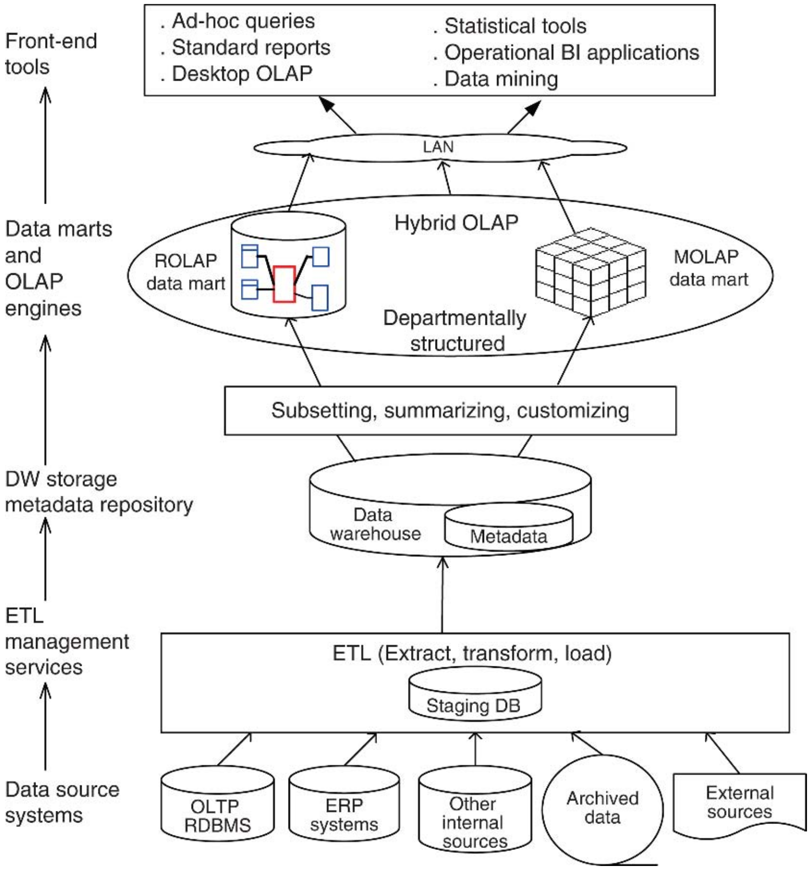
Data Warehousing Systems: Foundations and Architectures. Figure 2. A three dimensional data cube having dimensions Time, Item, and Location for MOLAP.

functions. Since a DW integrates all the important data for the analysis requirements of an enterprise, it needs to integrate data from all disparate sources. Data could include structured data, event data, semi-structured data, and unstructured data. The primary source for data is usually operational OLTP databases. A DW may also integrate data from other internal sources such as legacy databases, spreadsheets, archived storages, flat files, and XML files. Frequently, a DW system may also include any relevant data from external sources. Examples of such data are demographic data purchased from an information vendor to support sales and marketing analysis and standard reference data from the industry or the government. In order to analyze trends of data from a historical perspective, some archived data could also be selected. Thus, data warehousing systems usually end up with huge amounts of historical data.

These data are regularly fed into the second layer for processing. The interval between each feed could be monthly, weekly, daily, or even real-time, depending on the frequency of changes in the data and the importance of up-to-datedness of the data in the DW.

ETL Management Services The second layer extracts the data from disparate data sources, transforms the data into a suitable format, and finally loads them to a DW. This process is known as ETL processing.

A DW does not need all the data from the data source systems. Instead, only those data that are necessary for data analysis for tactical and strategic decision-making processes are extracted. Since these data come from many different sources, they could come in heterogeneous formats. Because a DW contains integrated data, data need to be kept in a single standard format by removing syntactic and semantic variations from different data source systems. Thus, these data are standardized for the data model used in the DW in terms of data type, format, size, unit of data, encoding of values, and semantics. This process ensures that the warehouse provides a “single version of the truth” [3].



Data Warehousing Systems: Foundations and Architectures. Figure 3. An enterprise data warehousing system architecture with ROLAP/MOLAP/Hybrid OLAP.

Only cleaned and conformed data are loaded into the DW. The storage required for ETL processing is called a staging database.

The ETL process is usually the most time-consuming phase in developing a data warehousing system [7]. It normally takes 60–80% of the whole development effort. Therefore, it is highly recommended that ETL tools and data cleansing tools be used to automate the ETL process and data loading.

Data Warehouse Storage and Metadata Repository

The third layer represents the enterprise DW and metadata repository. The enterprise DW contains all the extracted and standardized historical data at the atomic data level. A DW addresses the needs of cross-functional information requirements of an enterprise. The data will remain in the warehouse until they reach the limit specified in the retention strategy. After that period, the data are purged or archived.

Another component of this layer is the metadata repository. Metadata are data about the data. The repository contains information about the structures, operations, and contents of the warehouse. Metadata allows an organization to track, understand, and manage the population and management of the warehouse. There are three types of metadata – business metadata, technical metadata, and process metadata [7]. *Business metadata* describe the contents of the DW in business terms for easy access and understanding. They include the meaning of the data, organizational rules, policies, and constraints on the data as well as descriptive names of attributes used in reports. They help users in finding specific information from the warehouse. *Technical metadata* define the DW objects such as tables, data types, partitions, and other storage structures, as well as ETL information such as the source systems, extraction frequency, and transformation rules. *Process metadata* describe events during ETL operations and query statistics such as begin time, end time, CPU seconds, disk reads, and rows processed. These data are valuable for monitoring and troubleshooting the warehouse.

Metadata management should be carefully planned, managed, and documented. OMG's Common Warehouse Metamodel [9] provides the metadata standard.

Data Mart and OLAP Engines The fourth layer represents the data marts and OLAP engines. A data mart is a small-sized DW that contains a subset of the enterprise DW or a limited volume of aggregated data for the specific analysis needs of a business unit, rather than the needs of the whole enterprise. This definition implies three important features of a data mart, different from a DW system. First, the data for a data mart is fed from the enterprise DW when a separate enterprise DW exists. Second, a data mart could store lightly aggregated data for optimal analysis. Using aggregated data improves query response time. Third, a data mart contains limited data for the specific needs of a business unit. Conceptually, a data mart covers a business process or a group of related business processes of a business unit. Thus, in a fully-developed DW environment, end-users access data marts for daily analysis, rather than the enterprise DW.

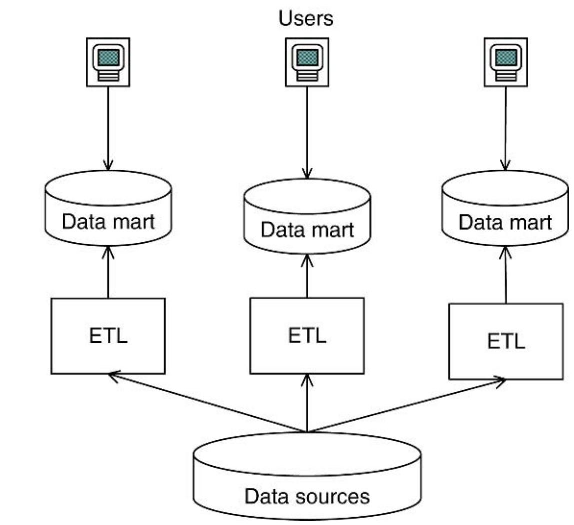
An enterprise usually ends up having multiple data marts. Since the data to all data marts are fed from the enterprise DW, it is very important to maintain the consistency between a data mart and the DW as well as among data marts themselves. A way to maintain the consistency is to use the notion of conformed dimension. A *conformed dimension* is a standardized dimension or a master reference dimension that is shared across multiple data marts [6]. Using conformed dimensions allows an organization to avoid repeating the "silos of information" problem.

Data marts are usually implemented in one or more OLAP servers. OLAP engines allow business users to perform data analysis using one the underlying implementation model – ROLAP, MOLAP, or HOLAP.

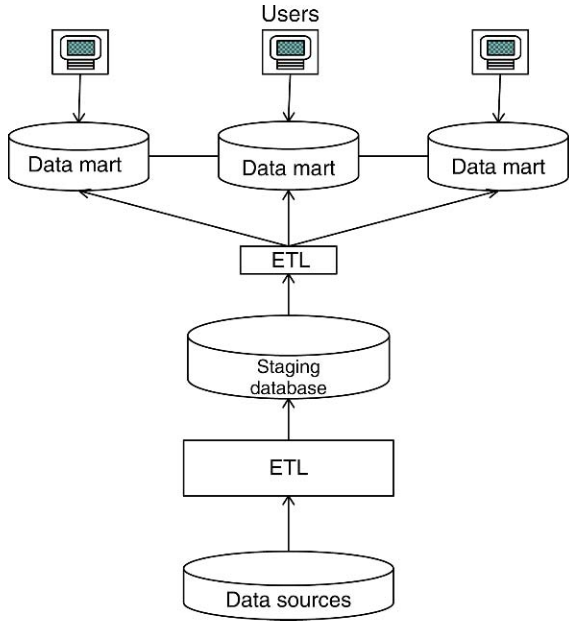
Front-end Tools The fifth layer represents the front-end tools. In this layer, end-users use various tools to explore the contents of the DW through data marts. Typical analyses include standard report generations, ad-hoc queries, desktop OLAP analysis, CRM, operational business intelligence applications such as dashboards, and data mining.

Other DW Architectures

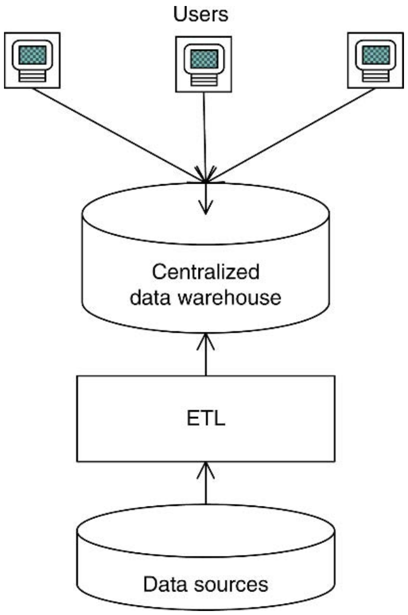
Figure 3 depicts the architecture of a typical data warehousing system with various possible components. The two primary paradigms for DW architectures are enterprise DW design in the top-down manner [3] and data mart design in the bottom-up manner [6]. A variety of architectures based on the two paradigms and other options exists [3,6,8,10,12]. In this section, seven different architectures are outlined. Figures 4–9 illustrate those architectures.



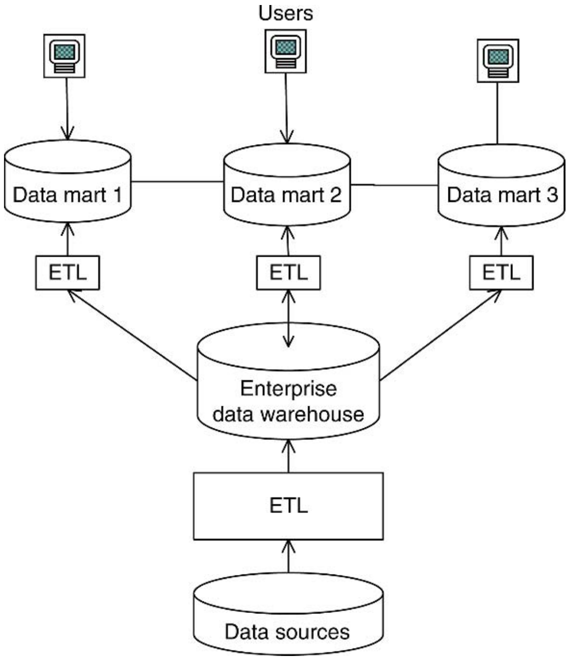
Data Warehousing Systems: Foundations and Architectures. Figure 4. Independent data marts.



Data Warehousing Systems: Foundations and Architectures. Figure 5. Data mart bus architecture with conformed dimensions.

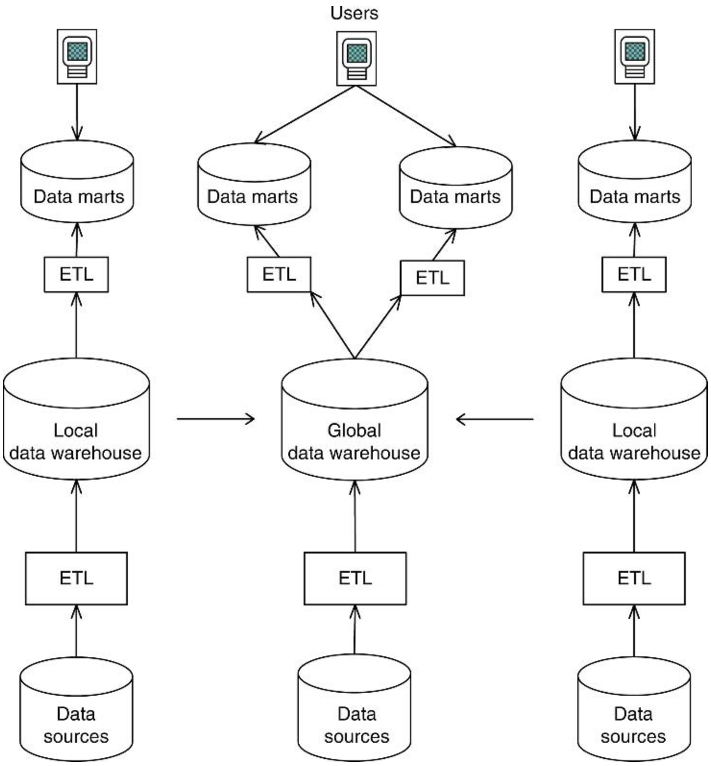


Data Warehousing Systems: Foundations and Architectures. Figure 6. Centralized DW architecture with no data marts.



Data Warehousing Systems: Foundations and Architectures. Figure 7. Hub-and-spoke architecture.

Independent Data Marts Architecture In this architecture, multiple data marts are created independently of each other. The data marts do not use conformed dimensions and measures. Thus, there is no unified view of enterprise data in this architecture. As the number of data marts grows, maintenance of consistency among data marts are difficult. In the long run, this architecture is likely to produce "silos of data marts."



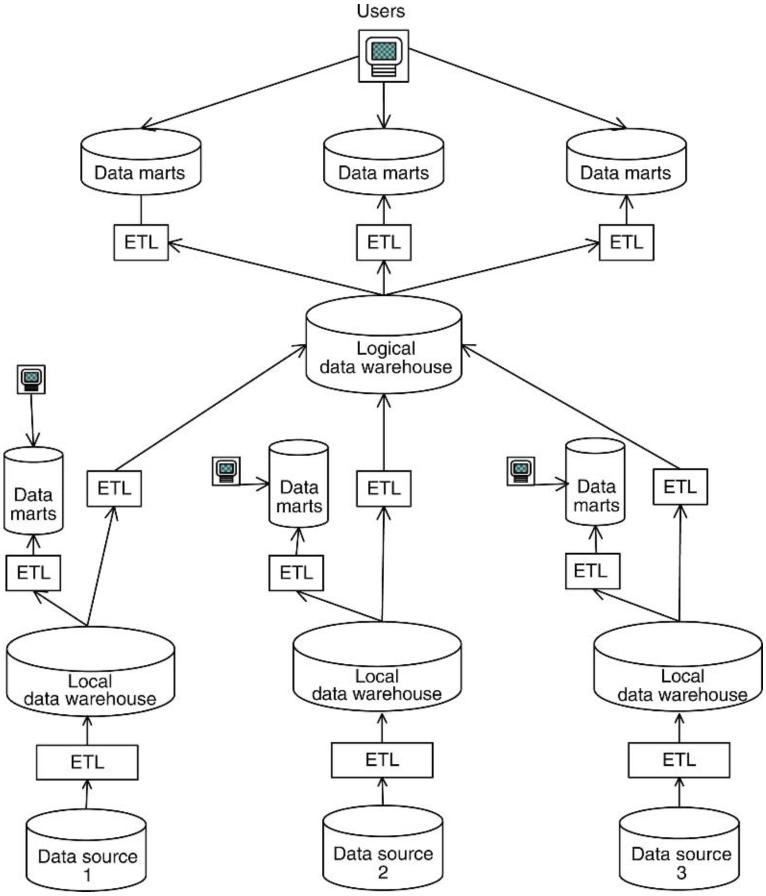
Data Warehousing Systems: Foundations and Architectures. Figure 8. Distributed DW architecture.

Data Mart Bus Architecture with Conformed Dimensions In this architecture, instead of creating a single enterprise level DW, multiple dimensional data marts are created that are linked with conformed dimensions and measures to maintain consistency among the data marts [6,7]. Here, an enterprise DW is a union of all the data marts together with their conformed dimensions. The use of the conformed dimensions and measures allows users to query all data marts together. Data marts contain either atomic data or summary data. The strength of the architecture is that data marts can be delivered quickly, and multiple data marts can be delivered incrementally. The potential weaknesses are that it does not create a single physical repository of integrated data and some data may be redundantly stored in multiple data marts.

Centralized Data Warehouse Architecture In this architecture, a single enterprise level DW is created for the entire organization without any dependent data marts. The warehouse contains detailed data for all the analytic needs of the organization. Users and applications directly access the DW for analysis.

Hub-and-Spoke Architecture (Corporate Information Factory) In this architecture, a single enterprise DW, called the hub, is created with a set of dimensional data marts, called spokes, that are dependent on the enterprise DW. The warehouse provides a single version of truth for the enterprise, and each data mart addresses the analytic needs of a business unit. This architecture is also called the corporate information factory or the enterprise DW architecture [3]. The warehouse contains data at the atomic level, and the data marts usually contain either atomic data, lightly summarized data, or both, all fed from the warehouse. The enterprise warehouse in this architecture is usually normalized for flexibility and scalability, while the data marts are structured in star schemas for performance. This top-down development methodology provides a centralized integrated repository of the enterprise data and tends to be robust against business changes. The primary weakness of this architecture is that it requires significant up-front costs and time for developing the warehouse due to its scope and scale.

Distributed Data Warehouse Architecture A distributed DW architecture consists of several local DWs



Data Warehousing Systems: Foundations and Architectures. Figure 9. Federated DW architecture.

and a global DW [3]. Here, local DWs have mutually exclusive data and are autonomous. Each local warehouse has its own ETL logic and processes its own analysis queries for a business division. The global warehouse may store corporate-wide data at the enterprise level. Thus, either corporate-level data analysis at the enterprise level or global data analyses that require data from several local DWs will be done at the global DW. For example, a financial analysis covering all the business divisions will be done at the global DW. Depending on the level of data and query flows, there could be several variations in this architecture [3]. This architecture supports multiple, geographically distributed business divisions. The architecture is especially beneficial when local DWs run on multiple vendors.

Federated Data Warehouse Architecture A federated DW architecture is a variation of a distributed DW architecture, where the global DW serves as a logical DW for all local DWs. The logical DW provides users with a single centralized DW image of the enterprise. This architecture is a practical solution when an enterprise acquires other companies that have their own DWs, which become local DWs. The primary advantage of this architecture is that existing environments of local DWs can be kept as they are without physically restructuring them into the global DW. This architecture may suffer from complexity and performance when applications require frequent distributed joins and other distributed operations. The architecture is built on an existing data environment rather than starting with a "clean slate."

Virtual Data Warehouses Architecture In a virtual DW architecture, there is no physical DW or any data mart. In this architecture, a DW structure is defined by a set of materialized views over OLTP systems. End-users directly access the data through the materialized views. The advantages of this approach are that it is easy to build and the additional storage requirement is minimal. This approach, however, has many disadvantages in that it does not allow any historical data; it does not contain a centralized metadata repository; it does not create cleansed standard data items across source systems; and it could severely affect the performance of the OLTP system.

Key Applications

Numerous business applications of data warehousing technologies to different domains are found in [1,6]. Design and development of clickstream data marts is covered in [5]. Applications of data warehousing technologies to customer relationship management (CRM) are covered in [2,11]. Extension of data warehousing technologies to spatial and temporal applications is covered in [8].

URL to Code

Two major international forums that focus on data warehousing and OLAP research are International Conferences on Data Warehousing and Knowledge Discovery (DaWaK) and ACM International Workshop on Data Warehousing and OLAP (DOLAP). DaWaK has been held since 1999, and DOLAP has been held since 1998. DOLAP papers are found at http://www.cis.drexel.edu/faculty/song/dolap.htm. A collection of articles on industrial DW experience and design tips by Kimball is listed in http://www.ralph-kimball.com/, and the one by Inmon is listed in www.inmoncif.com.

Cross-references

- ▶ Active and Real-time Data Warehousing
- ▶ Cube
- ▶ Data Mart
- ▶ Data Mining
- ▶ Data Warehouse
- ▶ Data Warehouse Life-Cycle and Design
- ▶ Data Warehouse Maintenance, Evolution and Versioning
- ▶ Data Warehouse Metadata
- ▶ Data Warehouse Security
- ▶ Dimension
- ▶ Extraction, Transformation, and Loading
- ▶ Materialized Views
- ▶ Multidimensional Modeling
- ▶ On-line analytical Processing
- ▶ Optimization and Tuning in Data Warehouses
- ▶ Transformation
- ▶ View Maintenance

Recommended Reading

1. Adamson C. and Venerable M. Data Warehouse Design Solutions. Wiley, New York, 1998.
2. Cunningham C., Song I.-Y., and Chen P.P. Data warehouse design for customer relationship management. J. Database Manage., 17(2):62–84, 2006.
3. Inmon W.H. Building the Data Warehouse, 3rd edn., Wiley, New York, 2002.
4. Jones M.E. and Song I.-Y. Dimensional modeling: identification, classification, and evaluation of patterns. Decis. Support Syst., 45(1):59–76, 2008.
5. Kimball R. and Merz R. The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse. Wiley, New York, 2000.
6. Kimball R. and Ross M. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 2nd edn., Wiley, 2002.
7. Kimball R., Ross M., Thorntwaite W., Munday J., and Becker B. 1The Data Warehouse Lifecycle Toolkit, 2nd edn., Wiley, 2008.
8. Malinowski E. and Zimanyi E. Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications. Springer, 2008.
9. Poole J., Chang D., Tolbert D., and Mellor D. Common Warehouse Metamodel: An Introduction to the Standard for Data Warehouse Integration. Wiley, 2002.
10. Sen A. and Sinha P. A comparison of data warehousing methodologies. Commun. ACM, 48(3):79–84, 2005.
11. Todman C. Designing a Data Warehouse Supporting Customer Relationship Management. Prentice-Hall, 2000.
12. Watson H.J. and Ariyachandra T. Data Warehouse Architectures: Factors in the Selection, Decision, and the Success of the Architectures. Technical Report, University of Georgia, 2005. Available from http://www.terry.uga.edu/~hwatson/DW_Architecture_Report.pdf

Data, Text, and Web Mining in Healthcare

Elizabeth S. Chen
Partners HealthCare System, Boston, MA, USA

Synonyms

Data mining; Text data mining; Web mining; Web data mining; Web content mining; Web structure mining; Web usage mining

Definition

The healthcare domain presents numerous opportunities for extracting information from heterogeneous sources ranging from structured data (e.g., laboratory results and diagnoses) to unstructured data (e.g., clinical documents such as discharge summaries) to usage data (e.g., audit logs that record user activity for clinical applications). To accommodate the unique characteristics of these disparate types of data and support the subsequent use of extracted information, several existing techniques have been adapted and applied including *Data Mining*, *Text Mining*, and *Web Mining* [7]. This entry provides an overview of each of these mining techniques (with a focus on Web usage mining) and example applications in healthcare.

Historical Background

Given the exponential growth of data in all domains, there has been an increasing amount of work focused on the development of automated methods and techniques to analyze data for extracting useful information. Data mining is generally concerned with large data sets or databases; several specialized techniques have emerged such as text mining and Web mining that are focused on text data and Web data, respectively. Early applications were in the domains of business and finance; however, the past decade has seen an increasing use of mining techniques in the life sciences, biomedicine, and healthcare. In the healthcare domain, data mining techniques have been used to discover medical knowledge and patterns from clinical databases, text mining techniques have been used to analyze unstructured data in the electronic health record, and Web mining techniques have been used for studying use of healthcare-related Web sites and systems.

Foundations

Data Mining

Knowledge Discovery in Databases (KDD) and data mining are aimed at developing methodologies and tools, which can automate the data analysis process and create useful information and knowledge from data to help in decision-making [9,11]. KDD has been defined as "the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data." This process is interactive and iterative and consists of several steps: data selection, preprocessing, transformation, data mining, and interpretation. Data mining is considered one step in the KDD process and is concerned with the exploration and analysis of large quantities of data in order to discover meaningful patterns and rules [9,11]. Two primary goals of data mining are prediction and description.

Text Mining

While data mining focuses on algorithmic and database-oriented methods that search for previously unsuspected structure and patterns in data, text mining is concerned with semi-structured or unstructured data found within text documents [5,12]. A narrower definition of text mining follows that of data mining in that it aims to extract useful information from text data or documents; a broader definition includes general text processing techniques that deal with search, extraction, and categorization [17]. Example applications include document classification, entity extraction, and summarization.

Web Mining

Web mining is the application of data mining techniques to automatically discover and extract information from data related to the World Wide Web [9,24,25]. Three categories of Web mining have been defined [18,6]:

- *Web content mining*: involves the discovery of useful information from Web content. These techniques involve examining the content of Web pages as well as results of Web searching.
- *Web structure mining*: obtains information from the organization of pages on the Web. These techniques seek to discover the model underlying link structures of the Web.
- *Web usage mining*: discovers usage patterns from Web data. These techniques involve analyzing data derived from the interactions of users while interacting with the Web.

Web usage mining seeks to understand the behavior of users by automatically discovering access patterns from their Web usage data. These data include Web server access logs, proxy server logs, browser logs, user sessions, and user queries. The typical Web usage mining process has three phases: preprocessing, pattern discovery, and pattern analysis [6,18,25].

Key Applications

Data Mining in Healthcare

Several studies have discussed the use of structured and unstructured data in the electronic health record for understanding and improving health care processes [5]. Applications of data mining techniques for structured clinical data include extracting diagnostic rules, identifying new medical knowledge, and discovering relationships between different types of clinical data. Using association rule generation, Doddi et al. discovered relationships between procedures performed on a patient and the reported diagnoses; this knowledge could be useful for identifying the effectiveness of a set of procedures for diagnosing a particular disease [8]. To identify factors that contribute to perinatal outcomes, a database of obstetrical patients was mined for the goal of improving the quality and cost effectiveness of perinatal care [23]. Mullins et al. explored a set of data mining tools to search a clinical data repository for novel disease correlations to enhance research capabilities [21].

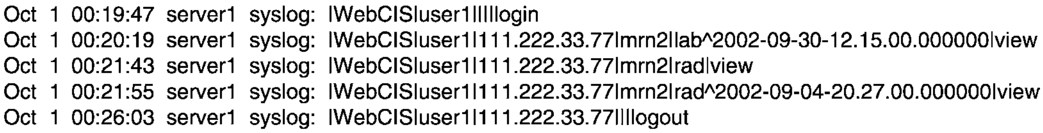
Text Mining in Healthcare

Natural language processing and text mining techniques have been applied in healthcare for a range of applications including coding and billing, tracking physician performance and resource utilization, improving provider communication, monitoring alternate courses of treatment, and detecting clinical conditions and medical errors [15]. Several studies have focused on the development of text mining approaches for identifying specific types of co-occurring concepts (e.g., concept pairs such as disease-drug or disease-finding) in clinical documents (e.g., discharge summaries) and biomedical documents (e.g., Medline articles). In one study, associations between diseases and findings (extracted from discharge summaries using a natural language processing tool) were identified and used to construct a knowledge base for supporting an automated problem list summarization system [2]. Another study discusses the mining of free-text medical records for the creation of disease profiles based on demographic information, primary diseases, and other clinical variables [14].

Web Usage Mining in Healthcare

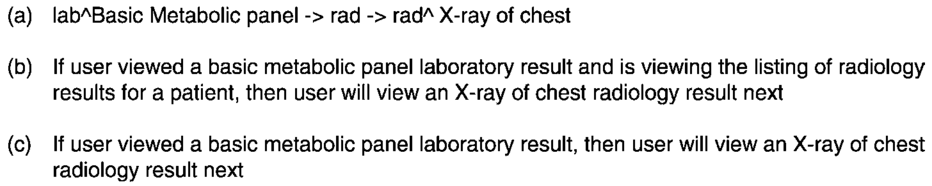
Major application areas of Web usage mining include personalization, system improvement, site modification, business intelligence, and usage characterization [25]. Web usage mining is viewed as a valuable source of ideas and methods for the implementation of personalized functionality in Web-based information systems [10,22]. Web personalization aims to make Web-based information systems adaptive for the needs and interests of individual users. The four basic classes of personalization functions are: *memorization*, *guidance*, *customization*, and *task performance support*. A number of research projects have used Web usage mining techniques to add personalization functionality in Web-based systems [20].

There are several reports of applying advanced techniques such as Web usage mining to study healthcare-related Web sites and systems. Malin has looked at correlating medical status (represented in health insurance claims as ICD-9 codes) with how information is accessed in a health information Web site [19]. The value of log data for public health surveillance has been explored for detecting possible epidemics through usage logs that record accesses to disease-specific on-line health information [16,13]. Zhang et al. used Web usage data to study users' information-seeking patterns of MyWelch, a Web-based medical library portal system [27]. Rozic-Hristovski et al. have used data warehouse and On-Line Analytical Processing (OLAP) techniques to evaluate use of the Central Medical Library (CMK) Web site. They found that existing Web log analysis tools only provided a set of predefined reports without any support for interactive data exploration, while their data warehouse and OLAP techniques would allow for dynamic generation of different user-defined reports that could be used to



```
Oct  1  00:19:47  server1  syslog:  |WebCIS|user1||||login
Oct  1  00:20:19  server1  syslog:  |WebCIS|user1|111.222.33.77|mrn2|lab^2002-09-30-12.15.00.000000|view
Oct  1  00:21:43  server1  syslog:  |WebCIS|user1|111.222.33.77|mrn2|rad|view
Oct  1  00:21:55  server1  syslog:  |WebCIS|user1|111.222.33.77|mrn2|rad^2002-09-04-20.27.00.000000|view
Oct  1  00:26:03  server1  syslog:  |WebCIS|user1|111.222.33.77|||logout
```

Data, Text, and Web Mining in Healthcare. Figure 1. WebCIS log file records. The WebCIS log files record details for users' (e.g., clinicians) interactions with patient data. Log file lines provide information on who, what, when, where, and how information was accessed in a patient's record. Each line has seven fields: timestamp, application name, userID, IP address, Medical Record Number (MRN), data type, and action. Data types may have subtypes (delimited by "^"). For example, the subtype "2002–09–30–12.15.00.000000" for the data type "lab" refers to a specific laboratory result (e.g., Basic Metabolic Panel) for the patient.



(a) lab^Basic Metabolic panel -> rad -> rad^ X-ray of chest

(b) If user viewed a basic metabolic panel laboratory result and is viewing the listing of radiology results for a patient, then user will view an X-ray of chest radiology result next

(c) If user viewed a basic metabolic panel laboratory result, then user will view an X-ray of chest radiology result next

Data, Text, and Web Mining in Healthcare. Figure 2. Transforming usage patterns to rules to shortcut rules. Each usage pattern (mined from the CIS log files (a) can be converted to a rule (b) and some patterns can be transformed to shortcut rules that exclude viewing of the department listings such as a listing of radiology results (c).

restructure the CMK Web site [26]. Another study explored regression analysis as a Web usage mining technique to analyze navigational routes used to access the gateway pages of the Arizona Health Sciences Library Web site. Bracke concluded that this technique could supplement Web log analysis for improving the design of Web sites [1].

Experimental Results

Depending on the clinical task, often only subsets of data are of interest to clinicians. Identifying these data, and the patterns in which they are accessed, can contribute to the design of efficient clinical information systems. At NewYork-Presbyterian Hospital (NYP), a study was performed to learn the patient-specific information needs (need for data in the patient record) of clinicians from the log files of WebCIS (a Web-based clinical information system at NYP) and subsequently apply this knowledge to enhance PalmCIS (a wireless handheld extension to WebCIS) [3,4].

Based on existing mining techniques (i.e., data mining and Web usage mining), "CIS Usage Mining" was developed as an automated approach for identifying patterns of usage for clinical information systems through associated log files (CIS log files). The CIS usage mining process consists of four phases: Data Collection – identify sources of CIS log files and obtain log file data (Fig. 1); Preprocessing – perform various tasks to prepare data for pattern discovery techniques including de-identification, data cleaning, data enrichment, and data transformation; Pattern Discovery – apply techniques for discovering statistics, patterns, and relationships such as descriptive statistical analysis, sequential pattern discovery, classification, and association rule generation; and, Pattern Analysis – filter out uninteresting patterns and determine how the discovered knowledge can be used through visualization techniques or query mechanisms.

The CIS usage mining techniques were applied to the log files of WebCIS to obtain usage statistics and patterns for all WebCIS users as well as particular classes of users (e.g., role-based groups such as physicians or nurses or specialty-based groups like pediatrics and surgery). A subset of the patterns were transformed into rules and stored in a knowledge base for enhancing PalmCIS with context-sensitive "shortcuts", which seek to anticipate what patient data the clinician may be interested in viewing next and provide automated links to those data (Fig. 2). Preliminary evaluation results indicated that shortcuts may have a positive impact and that CIS usage mining techniques may be valuable for detecting clinician information needs in different contexts.

Cross-references

▶ Association Rules
▶ Data Mining
▶ Text Mining
▶ Text Mining of Biological Resources
▶ Visual Data Mining

Recommended Reading

1. Bracke P.J. Web usage mining at an academic health sciences library: an exploratory study. J. Med. Libr. Assoc., 92(4): 421–428, 2004.
2. Cao H., Markatou M., Melton G.B., Chiang M.F., and Hripcsak G. Mining a clinical data warehouse to discover disease-finding associations using co-occurrence statistics. In Proc. AMIA Annual Symposium, 2005, pp. 106–110.
3. Chen E.S. and Cimino J.J. Automated discovery of patient-specific clinician information needs using clinical information system log files. In Proc. AMIA Annual Symposium, 2003, pp. 145–149.
4. Chen E.S. and Cimino J.J. Patterns of usage for a web-based clinical information system. In Proc. Medinfo, 2004, pp. 18–22.
5. Chen H., Fuller S., Friedman C., and Hersh W. Knowledge Management and Data Mining in Biomedicine. Springer, 2005.
6. Cooley R., Mobasher B., and Srivastava J. Web mining: information and pattern discovery on the World Wide Web. In Proc.

Nineth IEEE Int. Conf. on Tools with Artificial Intelligence, 1997, pp. 558–567.

7. Data Mining, Web Mining, Text Mining, and Knowledge Discovery. wwwkdnuggetscom.
8. Doddi S., Marathe A., Ravi S.S., and Torney D.C. Discovery of association rules in medical data. Med. Inform. Internet Med., 26(1):25–33, 2001.
9. Dunham M. Data Mining Introductory and Advanced Topics. Prentice-Hall, Englewood, Cliffs, NJ, 2003.
10. Eirinaki M. and Vazirgiannis M. Web mining for web personalization. ACM Trans. Internet Techn., 3(1):1–27, 2003.
11. Fayyad U., Piatetsky-Shapiro G., Smyth P., and Uthurusamy R. Advances in Knowledge Discovery and Data Mining. AAAI/MIT, 1996.
12. Hearst M. Untangling text data mining. In Proc. 27th Annual Meeting of the Assoc. for Computational Linguistics, 1999.
13. Heino J. and Toivonen H. Automated detection of epidemics from the usage logs of a physicians' reference database. In Principles of Data Mining and Knowledge Discovery, 7th European Conf, 2003, pp. 180–191.
14. Heinze D.T., Morsch M.L., and Holbrook J. Mining free-text medical records. In Proc. AMIA Symposium, 2001, pp. 254–258.
15. Hripcsak G., Bakken S., Stetson P.D., and Patel V.L. Mining complex clinical data for patient safety research: a framework for event discovery. J. Biomed. Inform. 36(1–2):120–30, 2003.
16. Johnson H.A., Wagner M.M., Hogan W.R., Chapman W., Olszewski R.T., and Dowling J. et al. Analysis of web access logs for surveillance of influenza. In Proc. Medinfo, 2004, p. 1202.
17. Konchady M. Text Mining Application Programming. Charles River Media. 2006, p. 2.
18. Kosala R. and Blockeel H. Web mining research: a survey. SIGKDD Explor., 2(1):1–15, 2000.
19. Malin B.A. Correlating web usage of health information with patient medical data. In Proc. AMIA Symposium, 2002, pp. 484–488.
20. Mobasher B., Cooley R., and Srivastava J. Automatic personalization based on web usage mining. Commun. ACM, 43(8):142–151, 2000.
21. Mullins I.M., Siadaty M.S., Lyman J., Scully K., Garrett C.T., and Greg Miller W. et al. Data mining and clinical data repositories: insights from a 667,000 patient data set. Comput. Biol. Med., 36 (12):1351–77, 2006.
22. Pierrakos D., Paliouras G., Papatheodorou C., and Spyropoulos C. Web usage mining as a tool for personalization: a survey. User Model. User-Adap., 13(4):311–372, 2003.
23. Prather J.C., Lobach D.F., Goodwin L.K., Hales J.W., Hage M.L., and Hammond W.E. Medical data mining: knowledge discovery in a clinical data warehouse. In Proc. AMIA Annual Fall Symposium, 1997, pp. 101–105.
24. Scime A. Web mining: applications and techniques. Idea Group Inc. 2005.
25. Srivastava J., Cooley R., Deshpande M., and Tan P. Web usage mining: discovery and applications of usage patterns from web data. SIGKDD Explor., 1(2):12–23, 2000.
26. Rozic-Hristovski A., Hristovski D., and Todorovski L. Users' information-seeking behavior on a medical library Website. J. Med. Libr. Assoc., 90(2):210–217, 2002.
27. Zhang D., Zambrowicz C., Zhou H., and Roderer N. User information seeking behavior in a medical web portal environment: a preliminary study. J. Am. Soc. Inform. Sci. Tech., 55(8): 670–684, 2004.

Database Adapter and Connector

Changqing Li
Duke University, Durham, NC, USA

Synonyms

Database connectivity

Definition

A database connector is a software that connects an application to any database. A database adapter is an implementation of a database connector. The connector is more at the conceptual level, while the adapter is at the implementation level, though they refer to the same thing. For simplicity, in the remaining parts of this entry, a database adapter will not be explicitly distinguished from a database connector, i.e., they are used to have the same meaning in the rest sections. Unlike the way to access data with a fixed schema, stored procedures, or queues, one can access table data directly and transparently with a database adapter.

Open Database Connectivity (ODBC) [2] and Java Database Connectivity (JDBC) [4] are two main database adapters to execute Structured Query Language (SQL) statements and retrieve results.

Historical Background

Before the universal database adapters, one has to write code that talks to a particular database using an appropriate language. For example, if a program needs to talk to an Access database and an Oracle database, the program has to be coded with two different database languages. This can be a quite daunting task, therefore uniform database adapters emerged.

Here the histories of the two main universal database adapters, i.e., ODBC and JDBC, are introduced.

ODBC enables applications connect to any database for which an ODBC driver is available. ODBC was created in 1992 by Microsoft, in partnership with Simba Technologies, by adapting the Call Level Interface (CLI) from the SQL Access Group (SAG). Later ODBC was aligned with the CLI specification making its way through X/Open (a company name) and International

Organization for Standardization (ISO), and SQL/CLI became part of the international SQL standard in 1995.

JDBC is similar to ODBC, but is designed specifically for Java programs. JDBC was firstly developed by JavaSoft, a subsidiary of Sun Microsystems, then developed under the Java Community Process. JDBC is part of the Java Standard Edition and the Java package java.sql contains the JDBC classes.

Foundations

A data architecture defines a data source interface to an application through connectors, and also by commands. Thus, a configurable request for data is issued through commands to the adapters of the data sources. This architecture provides the ability to create custom connectivity to disparate backend data sources.

Universal connectors enable rapid access to heterogeneous data and allow a broad range of seamless connectivity to file systems, databases, web applications, business applications and industry-standard protocols on numerous platforms. Business connectors allow customers to participate in collaboration, while web database adapters allow direct access to the database from web services.

Relational Database (RDB) adapters efficiently provide access to RDB data and systems. Standard SQL statements may be used to access RDB data via connectors including ODBC, OLE DB (Object Linking and Embedding, Database), JDBC, XML, iWay Business Services (Web services), MySQL Connector/ODBC and Connector/NET driver, and others.

Due to the longer history, ODBC offers connectivity to a wider variety of data sources than other new data-access Application Programming Interfaces (APIs) such as OLE DB, JDBC, and ADO.NET (ADO stands for ActiveX Data Objects).

Before the information from a database can be used by an application, an ODBC data source name must be defined, which provides information about how to connect the application server to a database, such as Microsoft SQL Server, Sybase, Oracle, or IBM DB2.

The implementations of ODBC can run on different operating systems such as Microsoft Windows, Unix, Linux, OS/2, and Mac OS X. Hundreds of ODBC drivers exist for different database products including Oracle, DB2, Microsoft SQL Server, Sybase, MySQL, PostgreSQL, Pervasive SQL, FileMaker, and Microsoft Access.

The first ODBC product was released by Microsoft as a set of Dynamic-Link Libraries (DLLs) for Microsoft Windows. In 2006, Microsoft ships its own ODBC with every supported version of Windows.

Independent Open Database Connectivity (iODBC) offers an open source, platform-independent implementation of both the ODBC and X/Open specifications. iODBC has been bundled into Darwin and Mac OS X, and it has also been ported by programmers to several other operating systems and hardware platforms, including Linux, Solaris, AIX, HP-UX, Digital UNIX, Dynix, FreeBSD, DG-UX, OpenVMS, and others.

Universal Database Connectivity (UDBC), laid the foundation for the iODBC open source project, is a cross-platform fusion of ODBC and SQL Access Group CLI, which enables non-Windows-based DBMS-independent (Database Management System independent) application development when shared-library implementations on Unix occurred only sporadically.

Headed, maintained and supported by Easysoft Director Nick Gorham, unixODBC has become the most common driver-manager for non-Microsoft Windows platforms and for one Microsoft platform, Interix. In advance of its competitors, unixODBC fully supports ODBC3 and Unicode. Most Linux distributions including Red Hat, Mandriva and Gentoo, now ship unixODBC. unixODBC is also used as the drivers by several commercial database vendors, including IBM (DB2, Informix), Oracle and SAP (Ingres). Many open source projects also make use of unixODBC. unixODBC builds on any platform that supports most of the GNU (a computer operating system composed entirely of free software) autoconf tools, and uses the LGPL (Lesser General Public License) and the GPL (General Public License) for licensing.

ODBC provides the standard of ubiquitous connectivity and platform-independence because hundreds of ODBC drivers exist for a large variety of data sources.

However, ODBC has certain drawbacks. Writing ODBC code to exploit DBMS-specific features requires more advanced programming. An application needs to use introspection to call ODBC metadata functions that return information about supported features, available types, syntax, limits, isolation levels, driver capabilities and more. Even when adaptive techniques are used, ODBC may not provide some advanced DBMS features. Important issues can also be raised by differences between drivers and driver maturity. Compared with drivers deployed and tested for years which may contain fewer bugs, newer ODBC drivers do not always have the stability.

Developers may use other SQL APIs if ODBC does not support certain features or types but these features are required by the applications. Proprietary APIs can be used if it is not aiming for platform-independence; whereas if it is aiming to produce portable, platform-independent, albeit language specific code, JDBC API is a good choice.

Sun's (a company name) Java (a programming language) 2 Enterprise Edition (J2EE) Connector Architecture (JCA) defines a standard architecture for connecting the Java 2 Platform to heterogeneous Enterprise Information Systems (EISs). The JCA enables an EIS vendor to provide a standard resource adapter (connector). The JDBC Connector is used to connect relational data sources. DataDirect technology is a pioneer in JDBC which provides resource adapters as an installable option for JDBC. The JDBC Developer Center provides the most current, developer-oriented JDBC data connectivity information available in the industry.

Multiple implementations of JDBC can exist and be used by the same application. A mechanism is provided by the API to dynamically load the correct Java packages and register them with the JDBC Driver Manager, a connection factory for creating JDBC connections.

Creating and executing statements are supported by JDBC connections. These statements may either be update statements such as SQL CREATE, INSERT, UPDATE and DELETE or query statements with SELECT.

Update statements e.g., INSERT, UPDATE and DELETE return how many rows are affected in the database, but do not return any other information.

Query statements, on the other hand, return a JDBC row result set, which can be walked over. Based on a name or a column number, an individual column in a row can be retrieved. Any number of rows may exist in the result set and the row result set has metadata to describe the names of the columns and their types.

To allow for scrollable result sets and cursor support among other things, there is an extension to the basic JDBC API in the javax.sql package.

Next the bridging configurations between ODBC and JDBC are discussed:

ODBC-JDBC bridges: an ODBC-JDBC bridge consists of an ODBC driver, but this ODBC driver uses the services of a JDBC driver to connect to a database. Based on this driver, ODBC function calls are translated into JDBC method calls. This bridge is usually used when an ODBC driver is lacked for a particular database but access to a JDBC driver is provided.

JDBC-ODBC bridges: a JDBC-ODBC bridge consists of a JDBC driver, but this JDBC driver uses the ODBC driver to connect to the database. Based on this driver, JDBC method calls are translated into ODBC function calls. This bridge is usually used when a particular database lacks a JDBC driver. One such bridge is included in the Java Virtual Machine (JVM) of Sun Microsystems. Sun generally recommends against the use of its bridge. Far outperforming the JVM built-in, independent data-access vendors now deliver JDBC-ODBC bridges which support current standards.

Furthermore, the OLE DB [1], the Oracle Adapter [3], the iWay [6] Intelligent Data Adapters, and MySQL [5] Connector/ODBC and Connector/NET are briefly introduced below:

OLE DB (Object Linking and Embedding, Database), maybe written as OLEDB or OLE-DB, is an API designed by Microsoft to replace ODBC for accessing different types of data stored in a uniform manner. While supporting traditional DBMSs, OLE DB also allows applications to share and access a wider variety of non-relational databases including object databases, file systems, spreadsheets, e-mail, and more [1].

The Oracle Adapter for Database and Files are part of the Oracle Business Process Execution Language (BPEL) Process Manager installation and is an implementation of the JCA 1.5 Resource Adapter. The Adapter is based on open standards and employs the Web Service Invocation Framework (WSIF) technology for exposing the underlying JCA Interactions as Web Services [3].

iWay Software's Data Adapter can be used for ALLBASE Database, XML, JDBC, and ODBC-Based Enterprise Integration. The Intelligent Data Adapters of iWay Work Together; each adapter contains a communication interface, a SQL translator to manage adapter operations in either SQL or iWay's universal Data Manipulation Language (DML), and a database interface to translate standard SQL into native SQL syntax [6].

MySQL supports the ODBC interface Connector/ODBC. This allows MySQL to be addressed by all the usual programming languages that run under Microsoft Windows (Delphi, Visual Basic, etc.). The ODBC interface can also be implemented under Unix, though that is seldom necessary [5]. The Microsoft .NET Framework, a software component of Microsoft Windows operating system, provides a programming interface to Windows services and APIs, and manages the execution of programs written for this framework [7].

Key Applications

Database adapters and connectors are essential for the current and future Web Services and Service Oriented Architecture, Heterogeneous Enterprise Information Systems, Data Integration and Data Interoperability, and any other applications to access any data transparently.

URL To Code

The catalog and list of ODBC Drivers can be found at: http://www.sqlsummit.com/ODBCVend.htm and http://www.unixodbc.org/drivers.html.

The guide about how to use JDBC can be found at: http://java.sun.com/javase/6/docs/technotes/guides/jdbc/.

Cross-references

- ▶ Data Integration
- ▶ Interface
- ▶ Java Database Connectivity
- ▶ .NET Remoting
- ▶ Open Database Connectivity
- ▶ Web 2.0/3.0
- ▶ Web Services

Recommended Reading

1. Blakeley J. OLE DB: a component dbms architecture. In Proc. 12th Int. Conf. on Data Engineering, 1996.
2. Geiger K. Inside ODBC. Microsoft, 1995.
3. Greenwald R., Stackowiak R., and Stern J. Oracle Essentials: Oracle Database 10g. O'Reilly, 2004.
4. Hamilton G., Cattell R., and Fisher M. JDBC Database Access with Java: A Tutorial and Annotated Reference. Addison Wesley, USA, 1997.
5. Kofler M. The Definitive Guide to MySQL5. A press, 2005.
6. Myerson J. The Complete Book of Middleware. CRC, USA, 2002.
7. Thai T., Lam H., .NET Framework Essentials. O'Reilly, 2003.

Database Clustering Methods

Xue Li
The University of Queensland, Brisbane, QLD, Australia

Synonyms

Similarity-based data partitioning

Definitions

Given a database $D = \{t_1, t_2,\ldots,t_n\}$, of tuples and a user defined similarity function s, $0 \leq s(t_i, t_j) \leq 1$, t_i, $t_j \in D$, the database clustering problem is defined as a partitioning process, such that D can be partitioned into a number of (such as k) subsets (k can be given), as C_1, $C_2,\ldots,C_k$, according to s by assigning each tuple in D to a subset C_i. C_i is called a cluster such that $C_i = \{t_i \mid s(t_i, t_r) \geq s(t_i,t_s)$, *if* $t_i,t_r \in C_j$ *and* $t_s \notin C_j\}$.

Key Points

Database clustering is a process to group data objects (referred as tuples in a database) together based on a user defined similarity function. Intuitively, a cluster is a collection of data objects that are "similar" to each other when they are in the same cluster and "dissimilar" when they are in different clusters. Similarity can be defined in many different ways such as Euclidian distance, Cosine, or the dot product. For data objects, their membership belonging to a certain cluster can be computed according to the similarity function. For example, Euclidian distance can be used to compute the similarity between the data objects with the numeric attribute values, where the geometric distance is used as a measure of the similarity. In a Euclidian space, the data objects are to each other, the more similar they are. Another example is to use the Euclidian distance to measure the similarity between a data object and a central point namely centroid of the cluster. The closer to the centroid the object is, the more likely it will belong to the cluster. So in this case, the similarity is decided by the radius of the points to their geometric centre.

For any given dataset a challenge question is how many natural clusters that can be defined. The answer to this question is generally application-dependent and can be subjective to user intentions.

In order to avoid specifying k for the number of clusters in a clustering process, a hierarchical method can be used. In this case, two different approaches, either agglomerative or divisive, can be applied. Agglomerative approach is to find the clusters step-by-step through a bottom-up stepwise merging process until the whole dataset is grouped as a single cluster. Divisive approach is to find the clusters step-by-step through a top-down stepwise split process until every data object becomes a single cluster.

Although hierarchical approaches have been widely used in many applications such as biomedical researches and experimental analysis in life science, they suffer from the problems of unable to undo the intermediate results in order to approach a global optimum solution. In an agglomerative approach, once two objects are merged, they will be together for all following merges and cannot be reassigned. In a divisive approach, once a cluster is split into two sub-clusters, they cannot be re-grouped into the same cluster for the further split.

In addition to hierarchical approaches, which do not need to specify how many clusters to be discovered, a user may specify an integer *k* for clustering data objects. In general, the task of finding a global optimal *k* partitions belongs to the class of NP-hard problem. For this reason, heuristics are used in many algorithms to achieve a balance between the efficiency and effectiveness as much as possible to close to the global optimum. Two well-known algorithms are the k-means and k-medoids.

One important feature of database clustering is that a dataset tends to be very large, high-dimensional, and coming at a high speed. By using a balanced tree structure, BIRCH algorithm [3] makes a single scan on the incoming data stream. BIRCH algorithm consists of two phases: (i) a summary of historical data is incrementally maintained in main memory as a clustering tree (CF tree). A node in CF tree gives the cardinality, centre, and radius of the cluster. Based on some heuristics, each new arriving data object is assigned to a sub-cluster, which leads to the update of its cluster feature in the CF tree. (ii) The clustering process is then applied on the leaf nodes of the CF tree. When the final cluster needs to be generated, the sub-clusters are treated as weighted data points and various traditional clustering algorithms can be applied in phase two computation without involving I/O operations.

DBSCAN [1] is a density based approach considering the coherence of the data objects. As a result, the nonconvex shapes clusters can be found based on the density that connect the data objects forming any kind of shapes in a Euclidian space. Spatial data indexes such as R* tree can be used to improve the system performance. STING [2] is another hierarchical approach that uses a grid structure to stores density information of the objects.

The key features of database clustering approaches are that (i) they are designed to deal with a large volume of data so a trade-off of accuracy and efficiency often needs to be considered. (ii) They are not able to see the complete dataset before the objects are clustered. So a progressive resolution refinement is used to approach the optimal solutions. (iii) They are designed to deal with constant data streams and so the incremental maintenances of the clustering results are required.

Cross-references

▶ Data Partitioning
▶ K-Means and K-Medoids
▶ Unsupervised Learning

Recommended Reading

1. Ester M., Kriegel H.P., Sander J., and Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise, In Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, 1996, pp. 226–231.
2. Han J., Kamber M., and Tung A.K.H. *1*Spatial clustering methods in data mining: a survey, In Geographic Data Mining and Knowledge Discovery, H. Miller, J. Han (eds.). Taylor and Francis, UK, 2001.
3. Zhang T., Ramakrishnan R., and Livny M. Birch: An efficient data clustering method for very large databases. In Proc. 1996 ACM SIGMOD Int. Conf. on Management of Data. Quebec, Canada, 1996, pp. 103–114.

Database Clusters

Marta Mattoso
Federal University of Rio de Janeiro, Rio de Janeiro, Brazil

Synonyms

DBC

Definition

A database cluster (DBC) is as a standard computer cluster (a cluster of PC nodes) running a Database Management System (DBMS) instance at each node. A DBC middleware is a software layer between a database application and the DBC. Such middleware is responsible for providing parallel query processing on top of the DBC. It intercepts queries from applications and coordinates distributed and parallel query execution by taking advantage of the DBC. The DBC term comes from an analogy with the term PC cluster, which is a solution for parallel processing by assembling sequential PCs. In a PC cluster there is no need for special hardware to provide parallelism as opposed to parallel machines or supercomputers. A DBC takes advantage of off-the-shelf sequential DBMS to run parallel queries. There is no need for special software or hardware as opposed to parallel database systems. The idea is to offer a high performance and cost-effective solution based on a PC cluster, without needing to change the DBMS or the application and its database.

Historical Background

Traditionally, high-performance of database query processing has been achieved with parallel database systems [7]. Parallel processing has been successfully used to improve performance of heavy-weight queries, typically by replacing the software and hardware platforms with higher computational capacity components (e.g., tightly-coupled multiprocessors and parallel database systems). Although quite effective, this solution requires the database system to have full control over the data, requiring an efficient database partitioning design. It also requires adapting applications from the sequential to the parallel environment. Migrating applications is complex (sometimes impossible), since it may require modifications to the source code. In addition, often it requires the expansion of the computational environment and the application modification, which can be very costly. A cheaper hardware alternative is to use parallel database systems for PC clusters. However, the costs can still be high because of a new database partitioning design and some solutions require specific software (DBMS) or hardware (e.g., SAN – Storage Area Network).

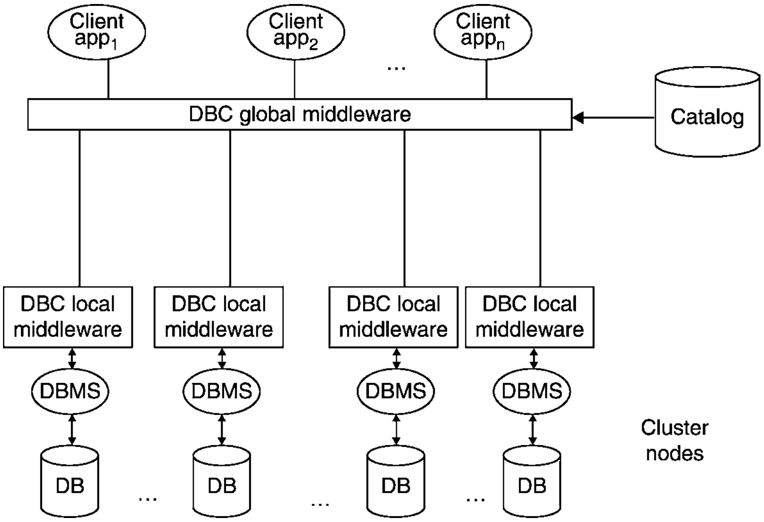
The DBC approach has been initially proposed by the database research group from ETH Zurich through the PowerDB project [10] to offer a less expensive and cost-effective alternative for high performance query processing. Thus, DBC is based on clusters of PC servers and pre-existing DBMS and applications. However, PowerDB is not open-source nor available for download. Several open-source DBC systems (e.g., RepDB*, C-JDBC, ParGRES, and Sequoia) have been proposed to support database applications by using different kinds of database replication on the DBC to obtain inter- and intra-query parallelism and fault tolerance.

Foundations

While many techniques are available for high performance query processing in parallel database systems, the main challenge of a DBC is to provide parallelism from outside the DBMS software.

A typical DBC architecture is a set of PC servers interconnected by a dedicated high-speed network, each one having its own processor(s) and hard disk (s), and running an off-the-shelf DBMS all coordinated by the DBC software middleware (Fig. 1). The DBC middleware is responsible for offering a single external view of the whole system, like a virtual DBMS. Applications need not be modified when database servers are replaced by their cluster counterparts. The DBC approach is considered to be non-intrusive since it does not require changes on the current application, its queries, its DBMS and its database.

Typically, the application is on the client side while the DBMS and the database is fully replicated at the PC cluster nodes. The DBC software middleware



Database Clusters. Figure 1. DBC architecture.

intercepts the application queries at the moment they are sent to the DBMS through the database driver. The DBC middleware then defines the best strategy to execute this query on the DBC to obtain the best performance from the DBC configuration. The DBC software middleware is typically divided on a global component which orchestrates the parallelism and a local component which tunes the local execution to participate on load balancing.

High performance in database applications can be obtained by increasing the system throughput, i.e., improving the number of transactions processed per second, and by speeding-up the query execution time for long running queries. The DBC query execution strategy varies according to the type of transactions being submitted and the DBC load. To improve system throughput, the DBC uses inter-query parallelism. To improve queries with long time execution the DBC implements intra-query parallelism. Inter- and intra-query parallelism can be combined. The query execution strategy is based on available database replicas.

Inter-query parallelism consists of executing many queries at the same time, each at a different node. Inter-query parallelism is implemented in DBC by transparently distributing queries to nodes that contain replicas of the required database. When the database is replicated at all nodes of the DBC, read-only inter-query parallelism is almost straightforward. Any read query can be sent to any replica node and execute in parallel. However, in the presence of updates the DBC must ensure the ACID transaction properties. Typically, a DBC global middleware has a component that manages a pool of connections to running DBMSs. Each request received by the DBC is submitted to a scheduler component that controls concurrent request executions and makes sure that update requests are executed in the same order by all DBMSs. Such scheduler should be able to be configured to enforce different parallel levels of concurrency.

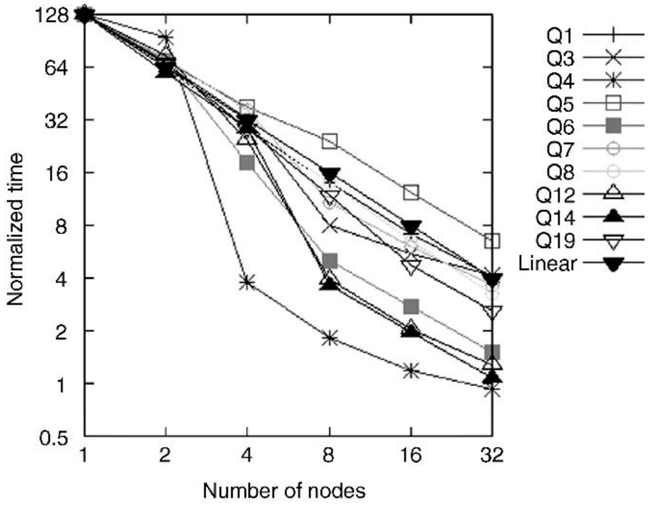
Intra-query parallelism consists of executing the same query in parallel, using sub-queries that scan different parts of the database (i.e., a partition), each at a different node. In a DBC, scanning different partitions without hurting the database autonomy is not simple to implement. In DBC, independent DBMSs are used by the middleware as "black-box" components. It is up to the middleware to implement and coordinate parallel execution. This means that query execution plans generated by such DBMSs are not parallel. Furthermore, as "black-boxes," they cannot be modified to become aware of the other DBMS and generate cooperative parallel plans. Physically partitioning the database relies on a good distribution design which may not work for several queries. An interesting solution to implement intra-query parallelism in DBC is to keep the database replicated and design partitions using *virtual partitioning* (VP) as proposed by Akal et al. [1]. VP is based on replication and dynamically designs partitions. The basic principle of VP is to take one query, rewrite it as a set of sub-queries "forcing" the execution of each one over a different subset of the table. Then the final query result is obtained through a composition of the partial results generated by the sub-queries.

Key Applications

DBC obtained much interest for various database applications like OLTP, OLAP, and e-commerce. Such applications can be easily migrated from sequential environments to the low cost DBC solution and obtain high performance in query processing. Different DBC open source solutions are available to cost-effective parallelism for various database applications. Since the high-performance requirements vary according to the typical queries of the applications, different DBC parallel techniques are provided. C-JDBC [3] and Sequoia [11] are DBC focused on e-commerce and OLTP applications. They use inter-query parallelism and are based on fault tolerance and load balancing in query distribution. RepDB* [8] is a DBC focused on throughput, which offers HPC for OLTP transactions. It uses inter-query parallelism and it is based on replica consistency techniques. ParGRES [6] is the only open-source DBC to provide for intra-query parallel processing [5], thus it is focused on OLAP applications. All these solutions have shown significant speedup through high performance query processing. Experimental results using the TPC series of benchmarks can be found for each one of the specific DBC software middlewares, for example TPC-W with C-JDBC and Sequoia, TPC-C with RepDB* and TPC-H with ParGRES.

Future Directions

Grid platforms can be considered a natural extension of PC clusters. They are also an alternative of high performance computing with large volumes of data. Several challenges in grid data management are discussed in [9]. An extension of the DBC approach to



Database Clusters. Figure 2. ParGRES DBC – TPC-H query execution times.

grids is proposed [4]. However, communication and data transfer can become a major issue.

Experimental Results

The graphic in Fig. 2 shows query execution time decreasing as more processors are included to process queries from the TPC-H benchmark. Query execution times in the graphic are normalized. These experiments have used a 32 PC cluster from Grid′5000 [2]. The graphic also shows the execution time that should be obtained if linear speedup was achieved. The speedup achieved by ParGRES while processing isolated queries with different number of nodes (from 1 to 32) is superlinear for most queries. A typical OLAP transaction is composed by a sequence of such queries, where one query depends on the result of the previous query. The user has a time frame to take his decisions after running a sequence of queries. Since OLAP queries are time consuming, running eight queries can lead to a four hour elapsed time, according to these tests using one single node for an 11 GB database. These eight queries can have their execution time reduced from four hours of elapsed time to less than one hour, just by using a small four nodes cluster configuration. With 32 nodes these queries are processed in a few minutes.

Data Sets

"TPC BenchmarkTM H – Revision 2.1.0", url: www.tpc.org.

URL to Code

url: cvs.forge.objectweb.org/cgi-bin/viewcvs.cgi/pargres/pargres/

Cross-references

- ▶ Data Partitioning
- ▶ Data Replication
- ▶ Data Warehouse Applications
- ▶ Distributed Database Design
- ▶ Grid File (and family)
- ▶ JDBC
- ▶ ODBC
- ▶ On-line Analytical Processing
- ▶ Parallel Database
- ▶ Parallel Query Processing
- ▶ Storage Area Network

Recommended Reading

1. Akal F., Böhm K., and Schek H.J. OLAP query evaluation in a database cluster: a performance study on intra-query parallelism. In Proc. Sixth East-European Conference on Advances in Databases and Information Systems, 2002, pp. 218–231.
2. Cappello F., Desprez F., and Dayde, M., et al. Grid5000: a large scale and highly reconfigurable grid experimental testbed. In International Workshop on Grid Computing, 2005, pp. 99–106.
3. Cecchet E. C-JDBC: a middleware framework for database clustering. IEEE Data Eng. Bull., 27:19–26, 2004.
4. Kotowski N., Lima A.A., Pacitti E., Valduriez P., and Mattoso M., Parallel Query Processing for OLAP in Grids. Concurrency and Computation: Practice & Experience, 20(17):2039–2048, 2008.

5. Lima A.A.B., Mattoso M., and Valduriez P. Adaptive virtual partitioning for OLAP query processing in a database cluster. In Proc. 14th Brazilian Symp. on Database Systems, 2004, pp. 92–105.
6. Mattoso M. et al. ParGRES: a middleware for executing OLAP queries in parallel. COPPE-UFRJ Technical Report, ES-690, 2005.
7. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems (2nd edn.). Prentice Hall, Englewood Cliffs, NJ, 1999.
8. Pacitti E., Coulon C., Valduriez P., and Özsu M.T. Preventive replication in a database cluster. Distribut. Parallel Databases, 18(3):223–251, 2005.
9. Pacitti E., Valduriez P., and Mattoso M. Grid data management: open problems and new issues. J. Grid Comput., 5(3):273–281, 2007.
10. Röhm U., Böhm K., Scheck H.-J., and Schuldt H. FAS - A freshness-sensitive coordination middleware for a cluster of OLAP components. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 754–768.
11. Sequoia Project, http://sequoia.continuent.org.

Database Connectivity

► Database Adapter and Connector

Database Constraints

► Database Dependencies

Database Dependencies

Marc Gyssens
University of Hasselt & Transnational University of Limburg, Diepenbeek, Belgium

Synonyms

Database constraints; Data dependency

Definition

For a relational database to be valid, it is not sufficient that the various tables of which it is composed conform to the database schema. In addition, the instance must also conform to the intended meaning of the database [15]. While many aspects of this intended meaning are inherently informal, it will generally induce certain formalizable relationships between the data in the database, in the sense that whenever a certain pattern is present among the data, this pattern can either be extended or certain data values must be equal. Such a relationship is called a database dependency. The vast majority of database dependencies in the literature are of the following form [5]:

$$(\forall x_1)\ldots(\forall x_n)\varphi(x_1,\ldots,x_n)$$
$$\Rightarrow (\exists z_1)\ldots(\exists z_k)\psi(y_1,\ldots,y_m,z_1,\ldots,z_k).$$

Here, $\{y_1,\ldots,y_m\} \subseteq \{x_1,\ldots,x_n\}$, φ is a (possibly empty) conjunction of relation atoms using all the variables $x_1,\ldots,x_n$, and ψ is either a single equality atom involving universally quantified variables only (in which case the dependency is called *equality-generating*); or ψ is a non-empty conjunction of relation atoms involving all the variables $y_1,\ldots,y_m$, $z_1,\ldots,z_k$ (in which case the dependency is called *tuple-generating*. A tuple-generating dependency is called *full* if it has no existential quantifiers: In the other case, it is called *embedded.*

Historical Background

The theory of database dependencies started with the introduction of *functional dependencies* by Codd in his seminal paper [8]. They are a generalization of (super) keys. A relation satisfies a functional dependency $X \rightarrow Y$ (where X and Y are sets of attributes) if, whenever two tuples agree on X, they also agree on Y. For example, if in a employee relation of a company database with schema

$$\Omega = \{\text{EMP-NR}, \text{ EMP-NAME}, \text{ JOB}, \text{ SALARY}\},$$

the functional dependencies

$$\{EMP\text{-}NR\} \rightarrow \{EMP\text{-}NAME, DEPT, JOB, SALARY\};$$
$$\{DEPT, JOB\} \rightarrow \{SALARY\}$$

hold, this means that *EMP-NR* is a key of this relation, i.e., uniquely determines the values of the other attributes, and that *JOB* in combination with *DEPT* uniquely determines *SALARY.*

Codd also noticed that the presence of a functional dependency $X \rightarrow Y$ also allowed a lossless decomposition of the relation into its projections onto $X \cup Y$ and $X \cup \overline{Y}$ ($\overline{Y}$ denoting the complement of Y). In the example above, the presence of $\{DEPT, JOB\} \rightarrow \{SALARY\}$ allows for the decomposition of the original relation into its projections onto $\{DEPT, JOB, SALARY\}$ and $\{EMP\text{-}NR, EMP\text{-}NAME, DEPT\}$.

Hence, the identification of constraints was not only useful for integrity checking but also for more efficient representation of the data and avoiding update anomalies through redundancy removal.

Subsequent researchers (e.g., [18]) noticed independently that the presence of the functional dependency $X \to Y$ is a sufficient condition for decomposability of the relation into its projection onto $X \cup Y$ and $X \cup \overline{Y}$, but not a necessary one. For example,

Drinker	*Beer*	*Bar*
Jones	*Tuborg*	*Far West*
Smith	*Tuborg*	*Far West*
Jones	*Tuborg*	*Tivoli*
Smith	*Tuborg*	*Tivoli*

can be decomposed losslessly into its projections onto $\{DRINKER, BEER\}$ and $\{BEER, BAR\}$, but neither $\{BEER\} \to \{DRINKER\}$ nor $\{BEER\} \to \{BAR\}$ holds. This led to the introduction of the *multivalued dependency*: a relation satisfies the multivalued dependency $X \twoheadrightarrow Y$ exactly when this relation can be decomposed losslessly into its projections onto $X \cup Y$ and $X \cup \overline{Y}$. Fagin [10] also introduced *embedded multivalued dependencies*: A relation satisfies the embedded multivalued dependency $X \twoheadrightarrow Y | Z$ if its projection onto $X \cup Y \cup Z$ can be decomposed losslessly into its projections onto $X \cup Y$ and $X \cup Z$. Sometimes, however, a relation be decomposed losslessly into three or more of its projections but not in two. This led Rissanen [17] to introduce a more general notion: a relation satisfies a *join dependency* $X_1 \bowtie \ldots \bowtie X_k$ if it can be decomposed losslessly into its projections onto $X_1,\ldots,X_k$.

Quite different considerations led to the introduction of *inclusion dependencies* [6], which are based on the concept of referential integrity, already known to the broader database community in the 1970s. As an example, consider a company database in which one relation, *MANAGERS*, contains information on department managers, in particular, *MAN-NAME*, and another, *EMPLOYEES*, contains general information on employees, in particular, *EMP-NAME*. As each manager is also an employee, every value *MAN-NAME* in *MANAGERS* must also occur as a value of *EMP-NAME* in *EMPLOYEES*. This is written as the inclusion dependency $MANAGERS[MAN\text{-}NAME] \subseteq EMPLOYEES[EMP\text{-}NAME]$. More generally, a database satisfies the inclusion dependency $R[A_1,\ldots,A_n] \subseteq S[B_1,\ldots,B_m]$ if the projection of the relation R onto the sequence of attributes $A_1,\ldots,A_n$ is contained in the projection of the relation S onto the sequence of attributes $B_1,\ldots,B_n$.

The proliferation of dependency types motivated researchers to propose subsequent generalizations, eventually leading to the tuple- and equality-generating dependencies of Beeri and Vardi [5] defined higher. For a complete overview, the reader is referred to [14] or the bibliographic sections in [1]. For the sake of completeness, it should also be mentioned that dependency types have been considered that are not captured by the formalism of Beeri and Vardi. An example is the *afunctional dependency* of De Bra and Paredaens (see, e.g., Chap. 5 of [15]).

Foundations

The development of database dependency theory has been driven mainly by two concerns. One of them is solving the inference problem, and, when decidable, developing tools for deciding it. The other is, as pointed out in the historical background, the use of database dependencies to achieve decompositions of the database contributing to more efficient data representation, redundancy removal, and avoiding update anomalies. Each of these concerns is discussed in some more detail below.

Inference

The inference problem is discussed here in the context of tuple- and equality-generating dependencies. The question that must be answered is the following: given a subtype of the tuple- and equality generating dependencies, given as input a set of constraints $\mathcal{C}$ and a single constraint c, both of the given type, is it decidable whether $\mathcal{C}$ logically implies c In other words, is it decidable if each database instance satisfying $\mathcal{C}$ also satisfies c? Given that database dependencies have been defined as first-order sentences, one might be inclined to think that the inference problem is just an instance of the implication problem in mathematical logic. However, for logical implication, one must consider all models of the given database scheme, also those containing infinite relations, while database relations are by definition finite. (In other words, the study of the inference of database dependencies lies within finite model theory.) To separate both notions of inference, a distinction is made between *unrestricted* *implication* (denoted $\mathcal{C} \models c$) and *finite implication* (denoted $\mathcal{C} \models_f c$) [5]. Since unrestricted implication is recursively enumerable and finite implication is co-recursively enumerable, their coincidence yields that the finite implication problem is decidable. The opposite, however, is not true, as is shown by the following counterexample. Consider a database consisting of a single relation R with scheme $\{A, B\}$. Let $\mathcal{C} = \{B \to A, R[B] \subseteq R[A]\}$ and let c be the inclusion dependency $R[A] \subseteq R[B]$. One can show that $\mathcal{C} \models_f c$, but $\mathcal{C} \not\models c$, as illustrated by the following, necessarily infinite, counterexample:

A	B
0	1
1	2
2	3
3	4
⋮	⋮

As will be pointed out later, the finite implication problem for functional dependencies and so-called *unary* inclusion dependencies (i.e., involving only one attribute in each side) is decidable.

An important tool for deciding (unrestricted) implication is the *chase*. In the chase, a table is created for each relation in the database. For each relation atom in the left-hand side of the dependency c to be inferred, its tuple of variables is inserted in the corresponding table. This set of tables is then *chased* with the dependencies of $\mathcal{C}$: in the case of a tuple-generating dependency, new tuples are added in a minimal way until the dependency is satisfied (in each application, new variables are substituted for existential variables); in the case of an equality-generating dependency, variables are equated until the dependency is satisfied. The result, *chase*($\mathcal{C}$), which may be infinite, can be seen as a model for $\mathcal{C}$. It is the case that $\mathcal{C} \models c$ if and only if the right-hand side of c is subsumed by some tuple of *chase*($\mathcal{C}$) (in the case of a tuple-generating dependency) or the required equality has been applied during the chase procedure.

In the case where only *full* tuple-generating dependencies and equality-generating dependencies are involved, the chase procedure is bound to end, as no existential variables occur in the dependencies, and hence no new values are introduced. In particular, the unrestricted implication problems coincides with the finite implication problem, and is therefore decidable. Deciding this inference problem is EXPTIME-complete, however.

The inference problem for all tuple- and equality-generating dependencies is undecidable, however (hence unrestricted and finite implication do not coincide). In 1992, Herrmann [13] solved a longstanding open problem by showing that the finite implication problem is already undecidable for embedded multivalued dependencies.

Another approach towards deciding inference of dependency types is trying to find an *axiomatization*: a finite set of inference rules that is both sound and complete. The existence of such an axiomatization is also a sufficient condition for the decidability of inference. Historically, Armstrong [2] was the first to propose such an axiomatization for functional dependencies. This system of inference rules was eventually extended to a sound and complete axiomatization for functional and multivalued dependencies together [3]:

(F1) $\emptyset \models X \to Y$ if Y ⊆ X (reflexivity)

(F2) $\{X \to Y\} \models XZ \to YZ$ (augmentation)

(F3) $\{X \to Y, Y \to Z\} \models X \to Z$ (transitivity)

(M1) $\{X \twoheadrightarrow Y\} \models X \twoheadrightarrow \bar{Y}$ (complementation)

(M2) $\emptyset \models X \twoheadrightarrow Y$ if Y ⊆ X (reflexivity)

(M3) $\{X \twoheadrightarrow Y\} \models XZ \twoheadrightarrow YZ$ (augmentation)

(M4) $\{X \twoheadrightarrow Y, Y \twoheadrightarrow Z\} \models X \twoheadrightarrow Z - Y$ (pseudo-transitivity)

(FM1) $\{X \to Y\} \models X \twoheadrightarrow Y$ (conversion)

(FM2) $\{X \twoheadrightarrow Y, Y \to Z\} \models X \to Z - Y$ (interaction)

Moreover, (F1)–(F3) are sound and complete for the inference of functional dependencies alone, and (M1)–(M4) are sound and complete form the inference of multivalued dependencies alone. The above axiomatization is at the basis of an algorithm to decide inference of functional and multivalued dependencies in low polynomial time.

Of course, the inference problem for join dependencies is also decidable, as they are full tuple-generating dependencies. However, there does not exist a sound and complete axiomatization for the inference of join dependencies [16], even though there does exist an axiomatization for a larger class of database dependencies.

There also exists a sound and complete axiomatization for inclusion dependencies [6]:

(I1) $\emptyset \models R[X] \subseteq R[X]$ (reflexivity)

(I2) $\{R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]\} \models R[A_{i1}, \ldots A_{ik}] \subseteq S[B_{i1}, \ldots, B_{ik}]$
if $i_1, \ldots, i_k$ is a sequence of integers in $\{1,\ldots, m\}$ (projection)

(I3) $\{R[X] \subseteq S[Y], S[Y] \subseteq T[Z]\} \models R[X] \subseteq T[Z]$ (transitivity)

Above, X, Y, and Z represent sequences rather than sets of attributes.

Consequently, the implication problem for inclusion dependencies is decidable, even though inclusion dependencies are embedded tuple-generating dependencies. However, deciding implication of inclusion dependencies is PSPACE-complete.

It has already been observed above that the unrestricted and finite implication problems for functional dependencies and unary inclusion dependencies taken together do no coincide. Nevertheless, the finite implication problem for this class of dependencies is decidable. Unfortunately, the finite implication problem for functional dependencies and general inclusion dependencies taken together is undecidable (e.g., [7]).

Decompositions

As researchers realized that the presence of functional dependencies yields the possibility to decompose the database, the question arose as to how far this decomposition process ought to be taken. This led Codd in follow-up papers to [8] to introduce several *normal forms*, the most ambitious of which is *Boyce-Codd Normal Form (BCNF)*. A database is in BCNF if, whenever one of its relations satisfies a nontrivial functional dependency $X \rightarrow Y$ (i.e., where Y is not a subset of X), X must be a superkey of the relation (i.e., the functional dependency $X \rightarrow U$ holds, where U is the set of all attributes of that relation). There exist algorithms that construct a lossless BCNF decomposition for a given relation. Unfortunately, it is not guaranteed that such a decomposition is also dependency-preserving, in the following sense: the set of functional dependencies that hold in the relations of the decomposition and that can be inferred from the given functional dependencies is in general not equivalent with the set of the given functional dependencies. Even worse, a dependency-preserving BCNF decomposition of a given relation does not always exist. For that reason, *Third Normal Form (3NF)*, historically a precursor to BCNF, is also still considered. A datatabase is in 3NF if, whenever one of its relations satisfies a nontrivial functional dependency $X \rightarrow \{A\}$ (A being a single attribute), the relation must have a minimal key containing A. Every database in BCNF is also in 3NF, but not the other way around. However, there exists an algorithm that, given a relation, produces a dependency-perserving lossless decomposition in 3NF. Several other normal forms have also been considered, taking into account multivalued dependencies or join dependencies besides functional dependencies.

However, one can argue that, by giving a join dependency, one actually already specifies how one wants to decompose a database. If one stores this decomposed database rather than the original one, the focus shifts from integrity checking to *consistency checking*: can the various relations of the decompositions be interpreted as the projections of a universal relation? Unfortunately, consistency checking is in general exponential in the number of relations. Therefore, a lot of attention has been given to so-called *acyclic join dependencies* [4]. There are many equivalent definitions of this notion, one of which is that an acyclic join dependency is equivalent to a set of multivalued dependencies. Also, global consistency of a decomposition is already implied by pairwise consistency if and only if the join dependency defining the decomposition is acyclic, which explains in part the desirability of acyclicity. Gyssens [12] generalized the notion of acyclicity to k-cyclicity, where acyclicity corresponds with the case $k = 2$. A join dependency is k-cyclic if it is equivalent to a set of join dependencies each of which has at most k components. Also, global consistency of a decomposition is already implied by k-wise consistency if and only if the join dependency defining the decomposition is k-cyclic.

Key Applications

Despite the explosion of dependency types during the latter half of the 1970s, one must realize that the dependency types most used in practice are still functional dependencies (in particular, key dependencies) and inclusion dependencies. It is therefore unfortunate that the inference problem for functional and inclusion dependencies combined is undecidable.

At a more theoretical level, the success of studying database constraints from a logical point view and the awareness that is important to distinguish between unrestricted and finite implication certainly contributed to the interest in and study and further development of finite model theory by theoretical computer scientists.

Finally, decompositions of join dependencies led to a theory of decompositions for underlying hypergraphs, which found applications in other areas as well, notably in artificial intelligence (e.g., [9,11]).

Cross-references

- ▶ Boyce-Codd Normal Form
- ▶ Chase
- ▶ Equality-Generating Dependencies
- ▶ Fourth Normal Form
- ▶ Functional Dependency
- ▶ Implication of Constraints
- ▶ Inconsistent Databases
- ▶ Join Dependency
- ▶ Multivalued Dependency
- ▶ Normal Forms and Normalization
- ▶ Relational Model
- ▶ Second Normal Form (2NF)
- ▶ Third Normal Form
- ▶ Tuple-Generating Dependencies

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of databases. Addison-Wesley, Reading, Mass., 1995. (Part C).
2. Armstrong W.W. Dependency structures of data base relationships. In Proc. IFIP Congress 74, 1974, pp. 580–583.
3. Beeri C., Fagin R., and Howard J.H. A complete axiomatization for functional and multivalued dependencies. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1978, pp. 47–61.
4. Beeri C., Fagin R., Maier D., and Yannakakis M. On the desirability of acyclic database schemes. J. ACM, 30 (3):479–513, 1983.
5. Beeri C. and Vardi M.Y. The implication problem for data dependencies. In Proc. Int. Conf. on Algorithms, Languages, and Programming, 1981. Springer, 1981, pp. 73–85.
6. Casanova M.A., Fagin R., and Papadimitriou C.H. Inclusion dependencies and their interaction with functional dependencies. J. Comput. Syst. Sci., 28(1):29–59, 1984.
7. Chandra A.K. and Vardi M.Y. The implication problem for functional and inclusion dependencies is undecidable. SIAM J. Comput., 14(3):671–677, 1985.
8. Codd E.F. A relational model of data for large shared data banks. Commun. ACM, 13(6):377–387, 1970.
9. Cohen D.A., Jeavons P., and Gyssens M. A unified theory of structural tractability for constraint satisfaction problems. J. Comput. Syst. Sci., 74(5):721–743, 2008.
10. Fagin R. Multivalued dependencies and a new normal form for relational databases. ACM Trans. Database Syst., 2(3):262–278, 1977.
11. Gottlob G., Miklós Z., and Schwentick T. Generalized hypertree decompositions: NP-hardness and tractable variants. In Proc. 26th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2007, pp. 13–22.
12. Gyssens M. On the complexity of join dependencies. Trans. Database Syst., 11(1):81–108, 1986.
13. Herrmann C. On the undecidability of implications between embedded multivalued dependencies. Inform. Comput., 122 (2):221–235, 1995.
14. Kanellakis P.C. Elements of relational database theory. In: Van Leeuwen J. (ed.). Handbook of theoretical computer science, Elsevier, 1991, pp. 1074–1156.
15. Paredaens J., De Bra P., Gyssens M., and Van Gucht D. The structure of the relational database model. In EATCS Monographs on Theoretical Computer Science, Vol. 17. Brauer W., Rozenberg G., and Salomaa A., (eds.). Springer, 1989.
16. Petrov S.V. Finite axiomatization of languages for representation of system properties. Inform. Sci., 47(3):339–372, 1989.
17. Rissanen J. Independent components of relations. ACM Trans. Database Syst., 2(4):317–325, 1977.
18. Zaniolo C. Analysis and design opf relational schemata for database systems. Ph. D. thesis, University of California at Los Angeles, 1976. Technical Report UCLA-Eng-7669.

Database Design

JOHN MYLOPOULOS
University of Trento, Trento, Italy

Definition

Database design is a process that produces a series of database schemas for a particular application. The schemas produced usually include a conceptual, logical and physical schema. Each of these is defined using a different data model. A conceptual or semantic data model is used to define the conceptual schema, while a logical data model is used for the logical schema. A physical schema is obtained from a logical schema by deciding what indexes and clustering to use, given a logical schema and an expected workload for the database under design.

Key Points

For every existing database, there is a design team and a design process that produced it. That process can make or break a database, as it determines what information it will contain and how will this information be structured.

The database design process produces a conceptual, a logical and a physical database schema. These schemas describe the contents of a database at different levels of abstraction. The conceptual schema focuses on the entities and relationships about which information is to be contained in the database. The Entity-Relationship Model is the standard model for defining conceptual schemas, though there have been many other proposals. UML class diagrams can also be used for this design phase. The logical schema describes the logical structure of the database. The Relational Model is the standard model for this phase, which views a database as a collection of tables. Alternative data models include the Hierarchical and the Network Data Models, but also object-oriented data models that view a database as a collection of inter-related objects instantiating a collection of classes.

The need to create different schemas that describe the contents of a database at different levels of abstraction was noted as far back as 1975 in a report by the American National Standards Institute (ANSI) [1], but has also evolved since. The report proposed a three-level architecture consisting of several external schemas representing alternative user views of a database, a conceptual schema whose information content subsumed that of external schemas, and an internal schema that represented database content in terms of a particular database technology (such as a relational Database Management System). For database design purposes, conceptual schemas have to be built upfront, whereas external schemas can be created dynamically according to user needs. Moreover, the notion of an internal schema has been refined to that of a logical and a physical schema.

The database design process often consists of four phases: requirements elicitation, conceptual schema design, logical schema design, and physical schema design. Requirements elicitation gathers information about the contents of the database to be designed from those who have a stake (a.k.a. *stakeholders*) This information is often expressed in natural language and may be ambiguous and/or contradictory. For example, two stakeholders may differ on what information about customers or patients is useful and should be included in the database-to-be. A conceptual schema is extracted from a given set of requirements through a series of steps that focus on noun phrases to identify entities, verb phrases to identify important relationships among entities, and other grammatical constructions to identify attributes about which information is useful to include in the database.

A conceptual schema is then transformed to a logical one through a series of well-defined transformations that map collections of entities and relationships into a relation whose attributes and keys are determined by the source entities and relationships. The logical schema design phase often includes a normalization step where an initial logical schema with associated functional dependencies is transformed into a normalized schema using one of several well-studied normal forms.

Physical schema design starts with a logical schema and determines the index to be used for each relation in the logical schema. This decision is based on the expected workload for the database-to-be, defined by the set of most important queries and updates that will be evaluated against the database. In addition, physical design determines the clustering of tuples in physical storage. This clustering plays an important role in the performance of the system as it evaluates queries that return many tuples (for example, queries that include joins). Physical schema design may dictate the revision of the logical schema by splitting/merging relations to improve performance. This step is known as *denormalization.*

As suggested by denormalization, the database design process should not be viewed as a sequential process that begins with requirements elicitation and proceeds to generate a conceptual, logical and physical schema in that order. Rather, the process consists of four linearly ordered phases and is iterative: after completing any one phase, the designer may return to earlier ones to revise the schemas that have been produced so far, and even the requirements that have been gathered.

Cross-references

▶ Conceptual Data Model
▶ Normalization Theory
▶ Physical Database Design for Relational Databases
▶ Semantic Data Model

Recommended Reading

1. American National Standards Institute. Interim Report: ANSI/X3/SPARC Study Group on Data Base Management Systems. FDT – Bull. ACM SIGMOD, 7(2):1–140, 1975.

2. Atzeni P., Ceri S., Paraboschi S., and Torlone R. Database Systems: Concepts, Languages and Architectures. McGraw Hill, New York, 1999.

Database Design Recovery

► Database Reverse Engineering

Database Engine

► Query Processor

Database Implementation

► Physical Database Design for Relational Databases

Database Interaction

► Session

Database Languages for Sensor Networks

Samuel Madden
Massachusetts Institute of Technology, Cambridge, MA, USA

Synonyms

Acquisitional query languages; TinySQL

Definition

Sensor networks – collections of small, inexpensive battery-powered, wirelessly networked devices equipped with sensors (microphones, temperature sensors, etc.) – offer the potential to monitor the world with unprecedented fidelity. Deploying software for these networks, however, is difficult, as they are complex, distributed, and failure prone. To address these complexities, several sensor network database systems, including TinyDB [7], Cougar [12], and SwissQM [8] have been proposed. These systems provide a high level SQL-like query language that allows users to specify what data they would like to capture from the network and how they would like that data processed without worrying about low-level details such power management, network formation, and time synchronization. This entry discusses the main features of these languages, and their relationship to SQL and other database languages.

Historical Background

Cougar and TinyDB were the first sensor network databases with the bulk of their development occurring between 1999 and 2003. They emerged as a result of rising interest in wireless sensor networks and other tiny, embedded, battery powered computers. TinyDB was co-developed as a part of the TinyOS operating system [2] for Berkeley Mote-based sensor networks. Initial versions of the motes used Atmel 8-bit microprocessors and 40 kbit s^{-1} radios; newer generations, developed by companies like Crossbow Technologies (http://www.xbow.com) and Moteiv Technologies (http://www.moteiv.com) use Zigbee (802.15.4) radios running at 250 kbit s^{-1} and Atmel or Texas Instruments 8 or 16 bit microprocessors running at 4–8 MHz. Nodes typically are very memory constrained (with 4–10 Kbytes of RAM and 48–128 Kbytes of nonvolatile flash-based program memory.) Most nodes can be interfaced to sensors that can capture a variety of readings, including light, temperature, humidity, vibration, acceleration, sounds, or images. The limited processing power and radio bandwidth of these devices constrains sample rates to at most a few kilosamples/s. Using such tiny devices does allow power consumption to be quite low, especially when sample rates are kept down; for example, networks that sample about once a second from each node can provide lifetimes of a month or longer on coin-cell batteries or a year or more on a pair of AA batteries [4].

The promise of sensor network databases is that they provide a very simple way to accomplish one of the most common goals of sensor networks: data collection. Using a simple, high level declarative language, users specify what data they want and how fast they want it. The challenge of building a sensor network database lies in capturing the required data in a power-efficient and reliable manner. The choice of programming language for these systems – the main topic of this entry – is essential to meeting that challenge. The language must be expressive enough to allow users to get the data they want, but also implementable in a way that is power-efficient, so that the network lasts as long as possible.

To understand how sensor network querying works, it is important to understand how sensor network databases are used. The typical usage model is as follows: a collection of static sensor nodes is placed in some remote location; each node is pre-programmed with the database software. These nodes report data wirelessly (often over multiple radio hops) to a nearby "basestation" – typically a laptop-class device with an Internet connection, which then relays data to a server where data is stored, visualized, and browsed.

Users interact with the system by issuing queries at the basestation, which in turn broadcasts queries out into the network. Queries are typically disseminated via flooding, or perhaps using some more clever gossip based dissemination scheme (e.g., Trickle [3]). As nodes receive the query, they begin processing it. The basic programming model is data-parallel: each node runs the same query over data that it locally produces or receives from its neighbors. As nodes produce query results, they send them towards the basestation.

When a node has some data to transmit, it relays it to the basestation using a so-called *tree-based routing protocol*. These protocols cause the nodes to arrange themselves into a tree rooted at the basestation. This tree is formed by having the basestation periodically broadcast a beacon message. Nodes that hear this beacon re-broadcast it, indicating that they are one hop from the basestation; nodes that hear those messages in turn re-broadcast them, indicating that they are two hops from the basestation, and so on. This process of (re)broadcasting beacons occurs continuously, such that (as long as the network is connected) all nodes will eventually hear a beacon message. When a node hears a beacon message, it chooses a node from which it heard the message to be its *parent*, sending messages through that parent when it needs to transmit data to the basestation (In general, parent selection is quite complicated, as a node may hear beacons from several candidate parents. Early papers by Woo and Culler [11] and DeCouto et al. [1] provide details.). Note that this ad hoc tree-based network topology is significantly different than the any-to-any routing networks that are used in traditional parallel and distributed database systems. As discussed below, this imposes certain limitations on the types of queries that are feasible to express efficiently in sensor network database systems.

Foundations

Most sensor network databases systems provide a SQL-like query interface. TinySQL, the query language used in TinyDB, for example, allows users to specify queries (through a GUI or command line interface) that appear as follows:

```
SELECT <select list>
  FROM <table list>
  WHERE  <condition list>
  GROUP BY  <field list>
  HAVING  <condition list>
  SAMPLE PERIOD  <duration>
   <additional clauses>
```

Most of these clauses behave just as they do in standard SQL. There are a few differences, however. First, the SAMPLE PERIOD clause requests that a data reading be produced once every <*duration*> seconds. This means that unlike most databases, where each query produces a single result set, in most sensor databases, each query produces a continuous stream of results. For example, to request the temperature from every node in a sensor network whose value is greater than 25°C once per sec, a user would write:

```
SELECT nodeid, temperature
  FROM sensors
  WHERE temperature > 25°C
  SAMPLE PERIOD 1s
```

Besides the continuous nature of sensor network query languages, this example illustrates that the data model provided by these systems is also somewhat unusual. First, the nodeid attribute is a unique identifier assigned to each sensor node and available in every query. Second, the table sensors is *virtual table* of sensor readings. Here, *virtual* means that it conceptually contains one row for every sensor type (light, temperature, etc.) from every sensor node at every possible instant, but all of those rows and columns are not actually materialized. Instead, only the sensor readings needed to answer a particular query are actually generated. The research literature refers to such languages as *acquisitional*, because they specify the rate and location where data should be acquired by the network, rather that simply querying a pre-existing table of data stored in the memory of the device [6].

Note also that although this table appears to be a single logical table its rows are actually produced by different, physically disjoint sensors. This suggests that some elements of query processing may be done inside of the network, before nodes transmit their data. For example, in the TAG system [5] a method for efficiently computing aggregates inside of a sensor network was proposed.

Restricted Expressiveness

It is important to note that sensor network query languages are less expressive than more general languages like SQL. By restricting expressiveness of their query languages, sensor network databases are able to ensure reasonably efficient query execution. For example, the TinyDB system imposes the following restrictions on queries:

- Arbitrary self-joins with the `sensors` table are not allowed. Such joins would require propagation of the entire table to all other nodes in the system. Queries with an equality predicate on `nodeid` can be evaluated efficiently and may be allowed.
- Nested queries are not allowed. Such queries potentially require disseminating query state throughout the network. For example, the query:

```
SELECT nodeid, temp
  FROM sensors
  WHERE temp >
  (SELECT AVG(temp)
  FROM sensors)
  SAMPLE PERIOD 1s
```

requires disseminating the average throughout the network in order to compute the query in a distributed fashion. Centralized implementations – where all of the data is sent to the basestation – are likely the only feasible implementation but can be quite inefficient due to the large amount of required data transmission.

Not all nested queries are inefficient to implement. For example, queries that compute aggregates over local state and then compute global aggregates over those local values (e.g., the average of the last five minutes temperatures at each node) have a natural distributed implementation. To avoid a confusing language interface where some nested queries are allowed and some are not, the designers of TinyDB chose to support certain classes of nested queries through two additional syntactic clauses: *temporal aggregates* and the *storage points*.

- *Temporal aggregates* allow users to combine a series of readings collected on a single node over time. For example, in a building monitoring system for conference rooms, users may detect occupancy by measuring the maximum sound volume over time and reporting that volume periodically; this could be done with the following query:

```
SELECT nodeid, WINAVG(volume,
30s, 5s)
  FROM sensors
  GROUP BY nodeid
  SAMPLE PERIOD 1s
```

This query will report the average volume from each sensor over the last 30 seconds once every 5 seconds, sampling once per second. The `WINAVG` aggregate is an example of a *sliding-window* operator. The final two parameters represent the window size, in seconds, and the sliding distance, in seconds, respectively.

- *Storage points* add a simple windowing mechanism to TinyDB that can be used to compute certain classes of locally nested queries. A storage point simply defines fixed-size materialization buffer of local sensor readings that additional queries can be posed over. Consider, as an example:

```
CREATE
  STORAGE POINT recentLight SIZE
  8 seconds
  AS (SELECT nodeid, light FROM
  sensors
  SAMPLE PERIOD 1s)
```

This statement provides a shared, local (i.e., single-node) location called `recentLight` to store a streaming view of recent data.

Users may then issue queries which insert into or read from storage points, for example, to insert in the `recentLight` storage point a user would write:

```
SELECT nodeid, light
  INTO recentLight
  SAMPLE PERIOD 1s
```

And to read from it, he might write:

```
SELECT AVG(light)
  FROM recentLight
  SAMPLE PERIOD 5s
```

Joins are also allowed between two storage points on the same node, or between a storage point and the `sensors` relation. When a sensors tuple arrives, it is joined with tuples in the storage point at its time of arrival.

The SwissQM [8] system does allow some forms of nested queries to be specified, but like TinyDB's STORAGE POINT syntax, these queries can operate only on a node's local state. For example, the internal query can compute each node's minimum temperature over the last 5 minutes, and then a global aggregate query can be used to compute the global minimum temperature over all nodes.

Specialized Language Constructs

Sensor network query languages usually include a number of specialized features designed to allow them to take advantage of the special hardware available on the sensor nodes they run on. For example, a user may wish to actuate some piece of attached hardware in response to a query. In TinyDB queries may specify an OUTPUT ACTION that will be executed when a tuple satisfying the query is produced. This action can take the form of a low-level operating system command (such as "Turn on the red LED"), or the instantiation of another query. For example, the query:

```
SELECT nodeid, temp
  WHERE temp > 100° F
  OUTPUT ACTION alarm()
  SAMPLE PERIOD 1 minute
```

will execute the command alarm() whenever a tuple satisfying this query is produced. This command is an arbitrary piece of C code that is written by the user and stored in a system catalog.

A related feature that is important in sensor networks is *event handling*. The idea is to initiate data collection when a particular external event occurs – this event may be outside of the database system (for example, when a switch on the physical device is triggered.) Events are important because they allow the system to be dormant until some external condition occurs, instead of continually polling or blocking, waiting for data to arrive. Since most microprocessors include external interrupt lines than can wake a sleeping device to begin processing, efficient implementations of event processing are often possible.

As an example, the TinyDB query:

```
ON EVENT switch-pressed(loc):
  SELECT AVG(light), AVG(temp),
  event.loc
  FROM sensors AS s
  WHERE dist(s.loc, event.loc) < 10m
 SAMPLE PERIOD 2 s FOR 30 s
```

reports the average light and temperature level at sensors when a switch on the device is pressed. Every time a switch-pressed event occurs, the query is issued from the detecting node and the average light and temperature are collected from nearby nodes once every 2 seconds for 30 seconds. The switch-pressed event is signaled to TinyDB by a low piece of driver-like C code that interfaces to the physical hardware.

Key Applications

Sensor network query languages have applications in any wireless sensing domain. They are particularly designed for environments where relatively simple programs that capture and process data are needed. Systems that implement such languages are often comparably efficient to hand-coded programs that require hundreds of times more code to implement.

Applications where sensor network databases have been deployed to good effect include environmental [10] and industrial [9] monitoring.

Cross-references

► In-Network Query Processing
► SQL
► Stream Processing

Recommended Reading

1. Couto D.S.J.D., Aguayo D., Bicket J., and Morris R. A high-throughput path metric for multi-hop wireless routing. In Proc. 9th Annual Int. Conf. on Mobile Computing and Networking, 2003.
2. Hill J., Szewczyk R., Woo A., Hollar S., Culler D., and Pister K. System architecture directions for networked sensors. In Proc. 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 2000.
3. Levis P., Patel N., Culler D., and Shenker S. Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor network. In Proc. 1st USENIX Symp. on Networked Systems Design & Implementation, 2004.
4. Madden S. The Design and Evaluation of a Query Processing Architecture for Sensor Networks. Ph.D. thesis, UC Berkeley, 2003.
5. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. In Proc. 5th USENIX Symp. on Operating System Design and Implementation, 2002.
6. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. The design of an acquisitional query processor for sensor networks.

In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.

7. Madden S., Hong W., Hellerstein J.M., and Franklin M. TinyDB Web Page.
8. Müller R., Alonso G., and Kossmann D. SwissQM: next generation data processing in sensor networks. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 1–9.
9. Stoianov I., Nachman L., Madden S., and Tokmouline T. PIPENET: a wireless sensor network for pipeline monitoring. In Proc. 6th Int. Symp. Inf. Proc. in Sensor Networks, 2007, pp. 264–273.
10. Tolle G., Polastre J., Szewczyk R., Culler D.E., Turner N., Tu K., Burgess S., Dawson T., Buonadonna P., Gay D., and Hong W. A macroscope in the redwoods. In Proc. 3rd Int. Conf. on Embedded Networked Sensor Systems, 2005, pp. 51–63.
11. Woo A., Tong T., and Culler D. Taming the underlying challenges of reliable multihop routing in sensor networks. In Proc. 1st Int. Conf. on Embedded Networked Sensor Systems, 2003.
12. Yao Y. and Gehrke J. Query processing in sensor networks. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.

Database Machine

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Definition

A database machine is a computer system which has special hardware designed and/or tuned for database accesses. Database machines may sometimes be coupled with a frontend server and, in this case, the database machines are called backend processors.

Key Points

The basic idea behind database machines was to put database computation closer to disk drives so as to achieve significant performance improvements. Database machines were actively studied in the 1970's and 1980's.

Early researchers explored filter processors which could efficiently examine data transferred from disk drives to a frontend server. Filter processors were categorized into four groups by D. DeWitt et al. [2]. A Processor-per-Track (PPT) machine is a system which consists of a number of cells (a set of tracks) and cell processors. As the data track rotates, the cell processor can scan the data and process search operations on the fly. A Processor-per-Head (PPH) machine is a system in which a processor is coupled with each head. Data is transferred in parallel from a set of heads and then processed in a set of processors. Thus, a whole cylinder is searched in a single rotation. In contrast to the PPT and PPH machines which need special disk hardware, a Processor-per-Disk (PPD) machine couples a processor with each disk drive. PPD can be seen as a compromising design, which has less performance advantage but can be realized at lower cost. A Multi-Processor Cache (MPC) machine is a system which couples multiple processors and multiple disk cache modules with each disk drive. The cache space is used for the processors to exchange the ability of selection operation.

When it came to the 1980s, researches of database machines were shifted to massive parallel computing. General-purpose processors and disk drives were tightly coupled into a node, and such nodes were then combined by a high-speed interconnect. Some of these types of database machines attained much success in the industry.

Cross-references

► Active Storage
► Intelligent Storage Systems

Recommended Reading

1. DeWitt D.J. and Gray J. Parallel database systems: The future of high performance database systems. Commun. ACM, 36 (6):85–98, 1992.
2. DeWitt D.J. and Hawthorn P.B. A performance evaluation of data base machine architectures. In Proc. 7th Int. Conf. on Very Data Bases, 1981, pp. 199–214.
3. Hurson A.R., Miller L.L., and Pakzad S.H. Parallel architectures for database systems. IEEE CS Press, 1989.

Database Management System

Pat Helland
Microsoft Corporation, Redmond, WA, USA

Synonyms

DBMS

Definition

A database management system is a software-based system to provide application access to data in a controlled and managed fashion. By allowing separate definition of the structure of the data, the database management system frees the application from many of the onerous details in the care and feeding of its data.

Historical Background

The first general purpose database systems emerged in the 1960s and represented their data in the *network data model* which presumes that portions of the data (typically called *records*) would explicitly reference any related data items in a graph (or network). The model was standardized as the CODASYL (Conference on Data Systems Languages) model and remained a strong influence on database systems into the 1980s.

In the late 1960s, the *hierarchical data model* emerged as exemplified by IBM's IMS (Information Management System). In the hierarchical data model, data was oriented in a hierarchy and the most natural form of navigation was via a parent-child relationship.

Starting in the 1970s, the *relational data model* emerged based on theoretical work done by Ted Codd and the System R research project at IBM's San Jose Research Lab. Out of System R emerged the SQL language. Concurrent with the System R effort, the Ingres project at UC Berkeley contributed strongly to both the development of the concepts and implementation techniques of the relational data model.

In the relational data model, data is represented in tables which can be related to each other by value in an operation called a join. Seminal to the relational data model is the absence of any explicit navigational links in the data. Relational systems continue to dominate what most people think of as a database system.

Other systems such as Object-Relational (O-R) systems and Entity-Relationship (E-R) systems either augment or compete with pure relational systems by formalizing the representation of relationships and objects (or entities) in the abstractions formally managed by the system. Industry opinions vary as to whether the O-R and E-R functionality should be included in the database management system itself or provided by the application layered on top of the database management system.

The notion of a database management system is under pressure to evolve in many different ways:

- In the 1980s, stored procedures and triggers were introduced into some system allowing the execution of application logic within the database itself. Stored procedures, combined with referential integrity and other declarative forms of business rules, pushed portions of the application into the database management system itself, blurring the traditional delineation between "app and data."
- As the single mainframe evolved into distributed systems, databases evolved to span many computers while attempting to provide the same behavior (on a larger scale) as provided by the centralized system. As the scope of the systems grows to thousands of machines, the semantics of the access to the data can no longer be identical to the smaller systems. Hence, the meaning of access to data is evolving.
- As huge numbers of devices and sources of data have arrived, it is no longer always enough to consider data as a passive collection. Consequently, innovations are seen in streaming databases wherein questions are posed about data which has not yet been completely gathered.
- With the arrival of the Internet, data is frequently sent outside of the database and then is returned back into it (potentially with changes). Database management systems have been designed to have complete control over their data and the effect of the Internet and other widely distributed systems pose challenges.
- As multiple applications (and their data) are independently created and then brought together in an increasingly connected world, the concept of a centralized definition of the data is under increasing pressure and forcing new innovation.

Database management systems focus on the data as separated from the application. While the concepts and implementations have evolved, the emphasis on data has remained at the center.

Foundations

The basic charter of a database management system is to focus on the separate management of data to reduce the costs and increase the functionality. Key to database management systems is the creation of higher level abstractions around how the application is separated from the data.

Today's database management systems are dominated by the relational data model. With the relational model, the high level abstraction is expressed as the DDL (Data Definition Language) which defines the schema for the data. In DDL, the data is laid out in tables and rows. Above the DDL abstraction, the application can manipulate the data. Below the DDL abstraction, the database management system can handle the storage, protection, high-performance reading and writing, and many other services. In so many ways, it is the existence of the DDL abstraction layer that is the essence of a database management system.

The high-level abstractions that separate the data from the application allows for a number of valuable characteristics:

- *Independent Evolution*

With the existence of the schema, the application can evolve based on the schema itself. Indeed, in most modern database management systems, the schema itself can be evolved while maintaining continuous support for existing applications.

In addition, the schema as seen by the application (the conceptual or logical schema) is typically separated from the physical schema which defines the actual layout of the storage and its associated indices. Leveraging the separation provided by the schema, the actual representation of the data may evolve to respond to changes in technology.

An essential part of a database management system is the protection of the investment in the application (which is typically very large) to allow for changes in technology. It is the higher level abstraction captured in the schema (via the DDL definition) that enable the protection of the investment in the application.

- *Multiple Applications Sharing the Same Data*

As the data is represented in a fashion based on its underlying semantics, it is possible to write new applications that modify the same shared data. By capturing the high-level conceptual schema, the underlying access to the physical storage is managed by the database management system. The combination of clearly separated meaning and the delegation of physical access to the intermediary provided by the database management system allows for new applications to be written against the same data.

- *Ad-Hoc Access to Data*

An important usage of database management systems has emerged in the form of *business intelligence*. Users are allowed to directly query and/or modify the contents of the data using direct access to the database management system itself. Ad-hoc access to data is made possible by the existence of the higher-level abstraction of the conceptual schema which describes the data independently of its physical schema and of its applications.

Business intelligence has, on its own, grown to a multi-billion dollar industry. Many enterprises find that the knowledge extracted from rapid and ad-hoc queries against their data can dramatically influence the business.

Essential to providing these abstractions are three application visible mechanisms, schema definition, data manipulation language (DML), and transactions.

- *Schema Definition*

Schema definition is typically done at two levels, the conceptual (or logical) schema and the physical schema.

The conceptual schema definition is the expression of the shape and form of the data as viewed by the application. In relational database management systems, the conceptual schema is described as tables, rows, and columns. Each column has a name and a data type. A collection of columns comprise a row, and a collection of rows, a table. The goal of the conceptual schema is to express the data organization as concisely as possible with each piece of knowledge represented in a single way.

The physical schema definition maps the conceptual schema into the underlying access methods which store the data onto disk. Both the storage of the underlying records and the capturing of various indices used for locating the records must be declared in the physical schema.

- *Data Manipulation*

Data manipulation refers to the mechanism by which the application extracts data from the database and changes data within the database. In relational systems, data manipulation is expressed as set oriented operations known as queries or updates. The SQL language, originally defined as a part of IBM's System R project in the late 1970s, has emerged as an ANSI standard and is, by far, the most commonly used DML (Data Manipulation Language) in modern relational database management systems.

The means for expressing access to and manipulation of data is one of the most important interfaces in computing. As innovations in data representation (such as XML) arrive, there are frequent debates about how to codify the access to those representations. There are competing forces in that the preservation of the existing interfaces is essential to the industry-wide investment in applications and in programmer expertise. Applying pressure to that is the desire to innovate in the representation and usage of data as system evolve. The evolution of database management systems is currently driven by the pressures of the blurring of data versus application, Internet scale distribution, streaming databases, and independently defined systems coming together.

The data manipulation portion of database management systems is a vibrant and lively space for innovation both in academia and industry.

- *Transactions*

The ability to process data using relations within the relational model of database management is dependent on *transactions* to combine the data in a meaningful and predictable way.

The transaction is a unit of work with four properties whose first letters spell ACID:

Atomic – The changes made within the transaction are all or nothing (even when failures occur).
Consistent – Rules enforced within the database (e.g., don't lose money in a bank) remain in effect when many transaction are concurrently executing.
Isolated – An ongoing transaction cannot see the effects of any other ongoing transaction.
Durable – When a transaction is committed, it remains committed.

It is the combination of Atomic, Isolated, and Durable which, when provided by the database management system, allow the application to provide its own notion of consistency.

Underlying these application visible mechanisms lays a lot of technology. Consider, in turn, query processing, access methods, and concurrency control and recovery.

- *Query Processing*

Since the advent of relational systems, set oriented expressions of data and updates to data have proven to be extraordinarily powerful. Application developers are given mechanisms for describing large amounts of data to be examined or modified and the system determines the best approach to accomplishing the intent expressed by the application. SQL DML allows for these powerful queries and updates as do some newly arriving other representations such as XQuery. There is an entire discipline within database management systems called *query processing* which applies set oriented operations to the underlying data and leverages whatever optimizations are appropriate to efficiently perform the application's work.

Efficient query processing must consider many factors over and above the expressed desires of the application. The available indices on the data, potential distribution of the data, the expected result sizes of the various sets of data that must be created as intermediaries, the performance characteristics of the processors, disks, networks, and remote computers all play a role in deciding a strategy by which the work will be accomplished.

One of the most important aspects to modern query processing is that these performance concerns are removed from the application programmer in most cases. Separating performance concerns from the application program's intent allows for an investment in applications which can survive many changes in the machines and data set sizes. The query processor may change its optimization strategies based upon new knowledge while the application program remains intact. While the ideal of performance independence is not completely realized in complex cases, it is true for a large number of applications.

Similar to the pressures on DML, query processing remains a vibrant discipline in a changing world.

- *Access Methods*

Access methods provide mechanisms for storing and retrieving the data. The dominant semantics for access methods is called a key-value pair. A key-value pair provides the ability to insert a blob of data associated with a unique key and then subsequently retrieve that blob. Some access methods allow searching for adjacent keys within a sort order of the keys. Other access methods only allow reading and writing based exclusively on exact matches of the key.

Much of the work in the 1970s and early 1980s in access method was dominated by methods for rapidly determining the disk address for the data. Initially, the records in the network and hierarchical schemes included direct disk addresses and it was straightforward for the system to retrieve the record. Soon, hashing schemes were employed wherein a single primary key, not a direct pointer, could be used locate a bucket of records and do so with high probability of accessing the record with a single disk operation. Separate indices where needed to find records based on non-primary key values.

Originally introduced in 1971 by Rudolf Bayer and Ed McCreight, B-Trees have emerged as the standard mechanism for self-organizing access methods. A B-Tree keeps an ordered list of keys in a balanced fashion which ensures a fixed depth from the base of the tree to the root even in the face of tremendous changes and churn. Since the mid-1990s, most modern database systems use a variant called a B+Tree in which the payload (the blob described above in the key-blob pair) is always kept in the collection of leaf nodes and a balanced structure of keys and pointers to other pages in the B+tree is kept in the non-leaf pages of the B+Tree.

Access methods (intertwined with the Concurrency Control and Recovery mechanism described below) are responsible for managing the storage of data within DRAM and when that data must be written out to disk. As the sizes of DRAM have increased more rapidly than most databases, there is an increasing trend towards in-memory databases. Indeed, the tipping point towards B+Trees as the dominant form of access method occurred when DRAM memories became large enough that the upper levels of the tree were essentially guaranteed to be in memory. That meant climbing the B+Tree did not require extra disk I/Os as it did when used with a smaller memory footprint.

B+Trees offer exceptional advantages for database management systems. In addition to access times that are uniform due to the uniform depth of the tree, each operation against the tree can be performed in a bounded (functional to O(log-n) of the size of the tree) time. Perhaps most importantly, B+Trees are self-managing and do not face challenges with empty space and garbage collection.

- *Concurrency Control and Recovery*

The goal of concurrency control is to provide the impression to the application developer that nothing else is happening to the contents of the database while the application's work proceeds. It should appear as if there is some serial order of execution even when lots of concurrent activity is happening. Practitioners of concurrency control speak of serializability. The effects of tightly controlling the concurrency are to make the execution behave as if it were within a serial order even when lots of work is happening concurrently. The ability to make a serial order is serializeabilty. While there are other more relaxed guarantees, serializability remains an important concept.

Recovery includes all the mechanisms used to ensure the intact recreation of the database's data even when things go wrong. These potential problems span system crashes, the destruction of the disks holding the database, and even the destruction of the datacenter holding the database.

Concurrency control and recovery systems use locking and other techniques for ensuring isolation. They are also responsible for managing the cached pages of the database as they reside in memory. Sophisticated techniques based on logging have allowed for high-performance management of caching, transactional recovery, and concurrency control.

Key Applications

Database management systems are widely in use to capture most of the data used in today's businesses. Almost all enterprise applications are written using database management systems. While many client-based applications are based on file system representations of data, most server-based applications use a DBMS.

Cross-references

- ▶ Abstraction
- ▶ Access Control
- ▶ ACID Properties
- ▶ Active and Real-Time Data Warehousing
- ▶ Atomicity
- ▶ B+-Tree
- ▶ Conceptual Schema Design
- ▶ Concurrency Control–Traditional Approaches
- ▶ Concurrency Control Manager
- ▶ Distributed DBMS
- ▶ Entity Relationship Model
- ▶ Generalization of ACID Properties
- ▶ Hierarchical Data Model
- ▶ Logical Schema Design
- ▶ Network Data Model
- ▶ QUEL
- ▶ Query Language
- ▶ Query Processing
- ▶ Query Processing (in relational databases)
- ▶ Relational Model
- ▶ Schema Evolution
- ▶ Secondary Index
- ▶ Serializability
- ▶ Stream Models
- ▶ Stream Processing
- ▶ System R(R*) Optimizer

- ▶ Transaction
- ▶ Transaction Management
- ▶ Transaction Manager
- ▶ Top-k XML Query Processing
- ▶ XML
- ▶ XPath/XQuery

Recommended Reading

1. Bayer R. Binary B-Trees for Virtual Memory. In Proc. ACM-SIGFIDET Workshop, 1971, pp. 219–235.
2. Blasgen M.W., Astrahan M.M., Chamberlin D.D., Gray J., King W.F., Lindsay B.G., Lorie R.A., Mehl J.W., Price T.G., Putzolu G.F., Schkolnick M., Slutz D.R., Selinger P.G., Strong H.R., Traiger I.L., Wade B., and Yost B. SystemR – an Architectural Update. IBM RJ IBM RJ 2581, IBM Research Center, 95193, 7/17/1979. 42 pp.
3. Gray J. Data management: past, present, and future. IEEE Comput., 29(10):38–46, 1996.
4. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques, Morgan Kaufmann, San Mateo, CA, 1992.
5. Stonebraker M., Wong E., Kreps P., and Held G. The Design and Implementation of INGRES. ACM Trans. Database Syst., 1(3):189–222, 1979.

Database Materialization

▶ Physical Database Design for Relational Databases

Database Middleware

CRISTIANA AMZA
University of Toronto, Toronto, ON, Canada

Synonyms

Database scheduling; Load balancing; Mediation and adaptation

Definition

Database middleware is a generic term used to refer to software infrastructure that supports (i) functionality, such as, interoperability between software components, or distributed transaction execution, (ii) improved database service, such as, performance scaling or fault tolerance of a database back-end in a larger system, or (iii) adaptations to workloads e.g., through the use of adaptive queuing middleware or of a scheduler component for adaptive reconfiguration of a database back-end.

Historical Background

Historically, TP Monitors were the first recognized database middleware components. TP Monitors, thus database middleware, was originally run on mainframes to connect different applications. Later, with the advent of e-business applications and modern multi-tier architectures that supported them, similar functionality as in the original TP Monitors became integrated in software components within the software stack used in these infrastructures, software components known as: "application servers," middleware components for "enterprise application integration," "enterprise service bus," and "transactional middleware." Transactional middleware supported the execution of distributed electronic transactions, and often provided much more functionality than just transactions. Modern e-business architectures consist of multiple tiers, such as client, application server, and database server. In these architectures typically replication in the database back-end is used for scaling. In these architectures, middleware components, such as, schedulers and load balancers are interposed in front of a database back-end for the purposes of scheduling database transactions, maintaining fault tolerance or providing data availability.

Foundations

Software for database middleware is very diverse, and serves a variety of purposes, e.g., integration of various data sources, load balancing for scaling, fault tolerance, etc. This entry distinguishes between the following middleware classes: *middleware for integration*, *middleware for performance scaling and availability*, *transactional and messaging middleware*, and *middleware for adaptation and reconfiguration*. This section will review these main types of database middleware and their uses, providing a brief survey of each of these areas.

Database Middleware for Integration

Database middleware, such as, Oracle9i, Informix Universal Server, MOCHA [12], DISCO [11], and Garlic [13] support integration of possibly heterogeneous, distributed data sources. Specifically, database middleware of this type may commonly consist of a data integration server, accessing the data at the remote data sources through a *gateway*, which provides client applications with a uniform view and access mechanism to the data available in each source. Several existing commercial database servers use this approach to access remote data sources through a *database gateway*, which provides an access method to the remote data, e.g., the IBM DB2 DataJoiner, and Sybases's Direct Connect, and Open Server Connect products. These products enable viewing multi-vendor, relational, nonrelational, and geographically remote data, as if it were local data.

The alternative is to use a *mediator* server for the same purpose. In this approach, the mediator server is capable of performing distributed query processing for the purposes of integration of the various data sources. Both methods superimpose a global data model on top of the local data models used at each data source. For example, the mediator uses wrappers to access and translate the information from the data sources into the global data model. Furthermore, in both of these middleware solutions, specialized libraries defining application data types and query operators are used by the clients, integration servers, gateways or wrappers deployed in the system.

Database Middleware for Scaling and Availability

Middleware components, such as, schedulers, load balancers and optimizers have been used for performance scaling of workloads on LAN-based database clusters, and/or for data availability. Schedulers and optimizers have been used for maintaining data consistency in replicated database clusters, and for minimizing the data movement in shared-nothing database clusters employing data partitioning, respectively. Shared-nothing database cluster architectures have been traditionally used for scaling classic database applications, such as, on-line transaction processing (OLTP) workloads. Data partitioning across the cluster [4,6] was absolutely necessary to alleviate the massive I/O needs of these applications through in-memory data caching. Data partitioning implied using rather complex optimizers to minimize reconfigurations and data movement between machines [5].

In contrast, more recently, due to the advent of larger memories, and the impact of modern e-commerce applications with more localized access patterns, scheduling applications for performance scaling on a cluster, using database replication has gained more attention [2,3]. For example, for the usual application sizes, there is little disk I/O in dynamic content applications [1], due to the locality exhibited by these applications. For example, in on-line shopping, bestsellers, promotional items and new products are accessed with high frequency. Similarly, the stories of the day and items on auction are hot objects in bulletin board and on-line bidding applications, respectively. This makes replication much more promising, and considerably easier to use than data partitioning.

However, replication for scaling incurs the cost of replicating the execution of update queries for maintaining the table replicas consistent. Fortunately, in dynamic content applications, queries that update the database are usually lightweight compared to read-only requests. For instance, in e-commerce, typically, only the record pertaining to a particular customer or product is updated, while any given customer may *browse* the product database using complex search criteria. More importantly, the locality in access patterns of dynamic content applications may imply higher conflict rates relative to traditional applications, given a sufficiently high fraction of writes. For instance the probability that a "best seller" book is being bought concurrently by two different customers, incurring a conflict on that item's stock is much higher than the probability that two customers access their joint account at the same time. Thus, intuitively, e-commerce applications have potentialy higher conflict rates than traditional OLTP applications. This trend has motivated more recent schemes on middleware support for scheduling transactions using a combination of load balancing and *conflict-aware replication* [2,3,10]. These techniques have shown good scaling in the tens of database engines range for the most common e-commerce workloads, e.g., on experiments using the TPC-W industry standard e-commerce benchmark. Middleware for caching query results has also been used in isolation or in combination with replication, e.g., through caching query results within the scheduler component corrdinating replication on a cluster, as an orthogonal technique for performance scaling of web sites supporting these workloads.

Finally, replication brings with it fault tolerance and high data availability as a bonus. Middleware components for fault tolerance support different degrees of failure transparency to the client. In the most common case, transaction aborts due to failures are exposed to the client when a failure of a replica occurs. More sophisticated fail-over schemes focus on precise error semantics, and on hiding failures from the client. Moreover, providing fault tolerance and data availability in a multi-tier architecture consisting of web-servers, application servers and database servers that interact with each other raises important trade-offs in terms of architecture design.

Middleware for precise failure semantics, such as exactly-once transactions [7], provide an automated end-to-end solution involving the client. Such middleware tracks client transactions through the software stack e.g., composed of client, application server, and database server, and can be used to automatically handle client duplicate requests, and reissue aborted transactions, thus seamlessly hiding failures in the database back-end from the client.

Transactional and Messaging Middleware

Transactional middleware provides control and the transfer of data between clients and a set of distributed databases. The main purpose is to run distributed transactions in an efficient manner. Transactional middleware systems, such as BEA Tuxedo, typically support a subset of the ACID properties, such as atomicity (by running a 2-phase-commit protocol over the distributed databases), durability or isolation. Transactional middleware is especially important in three-tier architectures that employ load balancing because a transaction may be forwarded to any of several servers based on their load and availability.

Message Queueing systems offer analogous functionality to TP Monitors, such as, improving reliability and scalability, although they typically offer weaker consistency guarantees than TP Monitors. Several messages may be required to complete an overall transaction, and those messages will each tend to reflect the latest system state rather than consistently looking back to the state of the system at the time the transaction started.

Messaging-oriented middleware provides an interface between client and server applications, allowing them to send data back and forth intermittently. If the target computer isn't available, the middleware stores the data in a message queue until the machine becomes available.

Middleware for Adaptation and Reconfiguration

Recent systems investigate adaptive reconfiguration in two classic middleware scenarios: database replication and message queuing systems.

In the context of database replication, dynamic adaptation has been used for reconfiguration of a database cluster to adapt to workload changes. Specifically, recent work adapts the configuration of a database cluster dynamically in response to changing demand by (i) adapting the placement of primary replicas and the degree of multi-programming at each replica [9] or by (ii) changing the number of replicas allocated to a workload [14]. For example, recent techniques for dynamic replica allocation in a database cluster employ an on-line technique based on middleware or group communication [8,14] for bringing a new replica up to date with minimal disruption of transaction processing on existing replicas in the application's allocation.

Key Applications

Database middleware is widely used in practice. All database vendors also offer a suite of middleware solutions for data integration, load balancing, scheduling, data replication, etc. Transactional middleware, message queuing systems, and middleware for fault tolerance, availbility and reconfiguration of the database back-end are commonly used in all modern e-business solutions, and in particular in multi-tier dynamic content web sites, such as, amazon.com and e-bay.com.

Cross-references

▶ Adaptive Middleware for Message Queuing Systems
▶ Mediation
▶ Message Queuing Systems
▶ Middleware Support for Database Replication and Caching
▶ Middleware Support for Precise Failure Semantics
▶ Replication in Multi-tier Architectures
▶ Transactional Middleware

Recommended Reading

1. Amza C., Cecchet E., Chanda A., Cox A., Elnikety S., Gil R., Marguerite J., Rajamani K., and Zwaenepoel W. Specification and implementation of dynamic web site benchmarks. In Proc. 5th IEEE Workshop on Workload Characterization, 2002.
2. Amza C., Cox A., and Zwaenepoel W. Conflict-aware scheduling for dynamic content applications. In Proc. 5th USENIX Symp. on Internet Technologies and Systems, 2003, pp. 71–84.
3. Amza C., Cox A.L., and Zwaenepoel W. Distributed versioning: Consistent replication for scaling back-end databases of dynamic content web sites. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2003, pp. 282–304.
4. Boral H., Alexander W., Clay L., Copeland G., Danforth S., Franklin M., Hart B., Smith M., and Valduriez P. Prototyping Bubba, a highly parallel database system. IEEE Trans. Knowl. Data Eng., 2:4–24, 1990.
5. Chaudhuri S. and Weikum G. Rethinking database system architecture: Towards a self-tuning RISC-style database system. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 1–10.
6. Copeland G., Alexander W., Boughter E., and Keller T. Data placement in Bubba. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988, pp. 99–108.

7. Frolund S. and Guerraoui R. e-transactions: End-to-end reliability for three-tier architectures. IEEE Trans. Software Eng., 28(4):378–395, 2002.
8. Liang W. and Kemme B. Online recovery in cluster databases. In Advances in Database Technology, Proc. 11th Int. Conf. on Extending Database Technology, 2008.
9. Milan-Franco J.M., Jimenez-Peris R., Patio-Martnez M., and Kemme B. Adaptive middleware for data replication. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004.
10. Plattner C. and Alonso G. Ganymed: scalable replication for transactional web applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004.
11. Rashid L., Tomasic A., and Valduriez P. Scaling heterogeneous databases and the design of DISCO. In Proc. 16th Int. Conf. on Distributed Computing Systems, 1996.
12. Rodriguez-Martinez M. and Roussopoulos N. MOCHA: a self-extensible database middleware system for distributed data sources. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000.
13. Roth M.T. and Schwarz P. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997.
14. Soundararajan G. and Amza C. Reactive provisioning of back-end databases in shared dynamic content server clusters. ACM Trans. Auto. Adapt. Syst., 1(2):151–188, 2006.

Database Physical Layer

▶ Storage Access Models

Database Profiling

▶ Data Profiling

Database Protection

▶ Database Security

Database Provisioning

▶ Autonomous Replication

Database Redocumentation

▶ Database Reverse Engineering

Database Repair

Leopoldo Bertossi
Carleton University, Ottawa, ON, Canada

Definition

Given an inconsistent database instance, i.e., that fails to satisfy a given set of integrity constraints, a repair is a new instance over the same schema that is consistent and is obtained after performing minimal changes on the original instance with the purpose of restoring consistency.

Key Points

Database instances may be inconsistent, in the sense that they may not satisfy certain desirable integrity constraints. In order to make the database consistent, certain updates can be performed on the database instance. However, it is natural to expect that any new consistent instance obtained in this way does not differ too much from the original instance. The notion of repair of the original instance captures this intuition: it is an instance of the same schema that does satisfy the integrity constraints and differs from the original instance by a minimal set of changes. Depending on what is meant by minimal set of changes, different repair semantics can be obtained.

The notion of *repair*, also called *minimal repair*, was introduced in [1]. Database instances can be seen as finite sets of ground atoms. For example, *Students(101, joe)* could be a database atom representing an entry in the relation *Students*. In order to compare two instances of the same schema, it is possible to consider their (set-theoretic) symmetric difference. A repair, as introduced in [1], will make the symmetric difference with the original instance minimal under set inclusion. That is, no other consistent instance differs from the original instance by a proper subset of database tuples. It is implicit in this notion of repair that changes on the original instance are obtained through insertions or deletions of complete database atoms. This notion of repair was used in [1] to characterize the consistent data in an inconsistent database as the data that is invariant under all possible repairs.

In the same spirit, other repairs semantics have also been investigated in the literature. For example, an alternative definition of repair might minimize the cardinality of the symmetric difference. There are also repairs that are obtained via direct updates of attribute values (as opposed to deletions followed by insertions, which might not represent a minimal change). In this case, the number of those local changes could be minimized. A different, more general aggregation function of the local changes could be minimized instead (cf. [2,3] for surveys).

Cross-references

▶ Consistent Query Answering
▶ Inconsistent Databases

Recommended Reading

1. Arenas M., Bertossi L., and Chomicki J. Consistent query answers in inconsistent databases. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 68–79.
2. Bertossi L. Consistent query answering in databases. ACM SIGMOD Rec., 35(2):68–76, 2006.
3. Chomicki J. Consistent query answering: five easy pieces. In Proc. 11th Int. Conf. on Database Theory, 2007, pp. 1–17.

Database Replication

▶ Data Replication
▶ Replica Control

Database Reverse Engineering

Jean-Luc Hainaut, Jean Henrard, Vincent Englebert, Didier Roland, Jean-Marc Hick
University of Namur, Namur, Belgium

Synonyms

Database redocumentation; Database design recovery

Definition

Database reverse engineering is the process through which the logical and conceptual schemas of a legacy database, or of a set of files, are reconstructed from various information sources such as DDL code, data dictionary contents, database contents or the source code of application programs that use the database.

Basically, database reverse engineering comprises three processes, namely physical schema extraction, logical schema reconstruction, and schema conceptualization. The first process consists in parsing the DDL code or the contents of an active data dictionary in order to extract the physical schema of the database. Reconstructing the logical schema implies analyzing additional sources such as the data and the source code of the application programs to discover implicit constraints and data structures, that is, constructs that have not been declared but that are managed by the information system or by its environment. The conceptualization process aims at recovering the conceptual schema that the logical schema implements.

Database reverse engineering is often the first step in information system maintenance, evolution, migration and integration.

Historical Background

Database reverse engineering has been recognized to be a specific problem for more than three decades, but has been formally studied since the 1980's, notably in [3,6,12]. The first approaches were based on simple rules, that work nicely with databases designed in a clean and disciplined way. A second generation of methodologies coped with physical schemas resulting from empirical design in which practitioners tend to apply non standard and undisciplined techniques. More complex design rules were identified and interpreted [2], structured and comprehensive approaches were developed [11,7] and the first industrial tools appeared (e.g., Bachman's Reengineering Tool). Many contributions were published in the 1990's, addressing practically all the legacy technologies and exploiting such sources of information as application source code, database contents or application user interfaces. Among synthesis publications, it is important to mention [5], the first tentative history of this discipline.

These second generation approaches were faced with two kinds of problems induced by empirical design [8]. The first problem is the recovery of *implicit constructs*, that is, structures and constraints that have not been explicitly declared in the DDL code. The second problem is that of the *semantic interpretation* of logical schemas that include non standard data structures.

Foundations

The ultimate goal of reverse engineering a piece of software is to recover its functional and technical specifications, starting mainly from the source code of the programs [4]. The problem is particularly complex with old and ill-designed applications. In this case, there is no documentation to rely on; moreover, the lack of systematic methodologies for designing and maintaining them have led to tricky and obscure code. Therefore, reverse engineering has long been regarded as a complex, painful and failure-prone activity, in such a way that it is simply not undertaken most of the time, leaving huge amounts of invaluable knowledge buried in legacy software, lost for all practical purposes.

In most software engineering cases, analysts have to content themselves with the extraction of abstract and/or partial information, such as call graphs, dependency graphs or program slices in order to ease the maintenance and evolution of the software. The result of reverse engineering a database is more satisfying, in that reconstructing the logical and conceptual schemas of an undocumented database is achievable with reasonable effort.

Database Design Revisited

To understand the problems, challenges and techniques specific to database reverse engineering, it is necessary to reexamine the way databases are developed, both in theory and in practice.

Standard Database Design Methodology Standard database design comprises four formal processes, namely conceptual analysis, logical design, physical design and coding.

Conceptual analysis produces the conceptual schema of the database, that is, an abstract description of the concepts that structure the application domain, of the relationships between these concepts and of the information to be collected and kept about theses classes and relationships. This schema is independent of the application programs that will use the database and is expressed in an abstract formalism such as some variant of the Entity-relationship model. It must be readable, maintainable, normalized and independent of any implementation technology.

Logical design translates the conceptual schema into data structures compliant with the data model of a family of DBMSs. This process is best described by a transformation plan, according to which the constructs (or components) of the conceptual schema that cannot be directly translated into the target DDL are first *transformed* into constructs of the DBMS model. For instance, a single-valued atomic attribute is directly translated into a column. On the contrary, a N:N relationship type cannot be expressed in the relational DDL. Therefore, it is first transformed into a relationship entity type and two N:1 relationship types, which in turn are translated into a relationship table and two foreign keys. The resulting logical schema is the basis for program development. It must be clear, simple and devoid of any performance concern. Denoting the conceptual and logical schemas respectively by `CS` and `LS`, this process can be synthesized by the functional expression `LS = logical-design(CS)`, that states that the logical schema results from the transformation of the conceptual schema.

Physical design enriches and potentially reshapes the logical schema to make it meet technical and performance requirements according to a specific technology (DBMS). Physical design can be expressed by `PS = physical-design(LS)`, where `PS` denotes the physical schema.

Coding expresses the physical schema in the DDL of the DBMS. Some of the data structures and integrity constraints can be translated into explicit DDL statements. Such is the case, in relational databases, for elementary data domains, unique keys, foreign keys and mandatory columns. However, the developer must resort to other techniques to express all the other constructs. Most relational DBMSs offer check and trigger mechanisms to control integrity, but other servers do not include such facilities, so that many constraints have to be coped with by procedural code distributed and duplicated in the application programs. The derivation of the code can be expressed by `code = coding(PS)`. The code itself can be decomposed into the DDL code in which some constructs are explicitly expressed and the external code that controls and manages all the other constructs: $\texttt{code} = \texttt{code}_{\texttt{ddl}} \cup \texttt{code}_{\texttt{ext}}$. Similarly, the coding function can be developed into a sequence of two processes ($\texttt{coding}_{\texttt{ddl}}\texttt{(PS)};\ \texttt{coding}_{\texttt{ext}}\texttt{(PS)}$).

The production of the database code from the conceptual schema (forward engineering or `FE`) can be written as `code = FE(CS)`, where function `FE` is the composition `coding o physical-design o logical-design`.

Empirical Database Design Actual database design and maintenance do not always follow a disciplined approach such as that recalled above. Many databases have been built incrementally to meet the evolving needs of application programs. Empirical design relies on the experience of self-taught database designers, who often ignore the basic database design theories and best practices. This does not mean that all these databases are badly designed, but they may include many non-standard patterns, awkward constructs and idiosyncrasies that make them difficult to understand [2]. Since no disciplined approach was adopted, such databases often include only a logical schema that integrates conceptual, logical, physical and optimization constructs. Quite often too, no up-to-date documentation, if any, is available. An important property of the functional model of database design evoked in previous section is that it is still valid for empirical design. Indeed, if empirical design rules of the designer are sorted according to the criteria of the three processes, functions `logical-design'`, `physical-design'` and `coding'` can be reconstructed into an idealized design that was never performed, but that yields the same result as the empirical design.

Database Reverse Engineering Processes

Broadly speaking, reverse engineering can be seen as the reverse of forward engineering [1], that is, considering the function $\texttt{RE} = \texttt{FE}^{-1}$, $\texttt{CS} = \texttt{RE(code)}$. Since most forward engineering processes consist of *schema transformations* [9], their reverse counterparts should be easily derivable by inverting the forward transformations.

Unfortunately, forward engineering is basically a lossy process as far as conceptual specifications are concerned. On the one hand, it is not unusual to discard bits of specifications, notably when they prove difficult to implement. On the other hand, the three processes are seldom injective functions in actual situations. Indeed, there is more than one way to transform a definite construct and several distinct constructs can be transformed into the same target construct. For instance, there are several ways to transform an is-a hierarchy into relational structures, including the use of primary-foreign keys (forward engineering). However, a primary-foreign key can also be interpreted as the implementation of a 1:1 relationship type, as the trace of entity type splitting or as the translation of an is-a relation (reverse engineering). Clearly, the transformational interpretation of these processes must be refined.

Nevertheless it is important to study and to model the reverse engineering as the inverse of `FE`, at least to identify and describe the pertinent reverse processes. Decomposing the initial relation `CS = RE(code)`, one obtains:

$$\begin{aligned}
&\texttt{CS} = \texttt{conceptualization(LS)}\\
&\texttt{LS} = \texttt{logical-reconstruction(PS, code}_{\texttt{ext}}\texttt{)}\\
&\texttt{PS} = \texttt{physical-extraction(code}_{\texttt{ddl}}\texttt{)}\\
&\texttt{RE} = \texttt{conceptualization} \circ \texttt{logical-}\\
&\texttt{reconstruction} \circ \texttt{physical-extraction}
\end{aligned}$$

where

$$\begin{aligned}
&\texttt{Conceptualization} = \texttt{logical-design}^{-1}\\
&\texttt{Logical-reconstruction} = \texttt{physical-design}^{-1}\\
&\qquad \texttt{||}\ \texttt{coding}_{\texttt{ext}}{}^{-1}\\
&\texttt{Physical-extraction} = \texttt{coding}_{\texttt{ddl}}{}^{-1}
\end{aligned}$$

This model emphasizes the role of program code as a major source of information. As explained below, other sources will be used as well.

Physical Schema Extraction

This process recovers the physical schema of the database by parsing its DDL code ($\texttt{code}_{\texttt{ddl}}$) or, equivalently, by analyzing the contents of its active data dictionary, such as the system tables in most relational systems. This extraction makes visible the *explicit constructs* of the schema, that is, the data structures and constraints that have been explicitly declared through DDL statements and clauses. Such is the case for primary keys, unique constraints, foreign keys and mandatory fields. Generally, this process is fairly straightforward. However, the analysis of sub-schemas (e.g., relational views, CODASYL sub-schemas or IMS PCBs) can be more intricate. Indeed, each sub-schema brings a partial, and often refined view of the global schema. In addition, some data managers, such as standard file managers, ignore the concept of global schema. A COBOL file for instance, is only described in the source code of the programs that use it. Each of them can perceive its data differently. Recovering the global physical schema of a COBOL file requires a potentially complex schema integration process.

Logical Schema Reconstruction

This process addresses the discovery of the *implicit constructs* of the schema. Many logical constructs have not been declared by explicit DDL statements and clauses. In some favorable situations, they have been translated into programmed database components such as SQL checks, triggers and stored procedures.

However, most of them have been translated into application program fragments (nearly) duplicated and scattered throughout millions of lines of code ($\texttt{code}_{\texttt{ext}}$). For instance, a popular way to check a referential constraint consists in accessing the target record before storing the source record in its file. Recovering these *implicit constructs*, in contrast with *explicit constructs*, which have been expressed in DDL, requires a precise analysis of various pieces of procedural code. Though program source code is the richest information source, the database contents (the *data*), screen layout, report structure, program execution, users interview and, of course, (possibly obsolete) documentation will be analyzed as well. As a final step of the reconstruction, physical components are discarded inasmuch as they are no longer useful to discover logical constructs.

Implicit Constructs All the structures and constraints that cannot be expressed in the DDL are implicit by nature. However, many database schemas include implicit constructs that could have been declared at design time but that were not, for such reasons as convenience, standardization, inheritance from older technology or simply by ignorance or bad design. Two popular examples can be mentioned. In network and hierarchical databases, some links between record types are translated into implicit foreign keys despite the fact that relationship types could have been explicitly declared through set types or parent-child relationship types. In many legacy relational databases, most foreign keys are not declared through `foreign key` clauses, but are managed by an appropriate set of triggers. The most important implicit constructs are the following [8].

Exact field and record structure. Compound and multivalued fields are often represented by the concatenation of their elementary values. Screen layout and program analysis are major techniques to discover these structures.

Unique keys of record types and multivalued fields. This property is particularly important in strongly structured record types and in sequential files.

Foreign keys. Each value of a field is processed as a reference to a record in another file. This property can be discovered by data analysis and program analysis.

Functional dependencies. The values of a field can depend on the values of other fields that have not been declared or elicited as a candidate key. This pattern is frequent in older databases and file systems for performance reasons.

Value domains. A more precise definition of the domain of a field can be discovered by data and program analysis. Identifying enumerated domains is particularly important.

Meaningful names. Proprietary naming standards (or, worse, the absence thereof) may lead to cryptic component names. However, the examination of program variables and electronic form fields in/from which field values are moved can suggest more significant names.

Sources and Techniques Analytical techniques applied to various sources can all contribute to a better knowledge of the implicit components and properties of a database schema.

Schema analysis. Spotting similarities in names, value domains and representative patterns may help identify implicit constructs such as foreign keys.

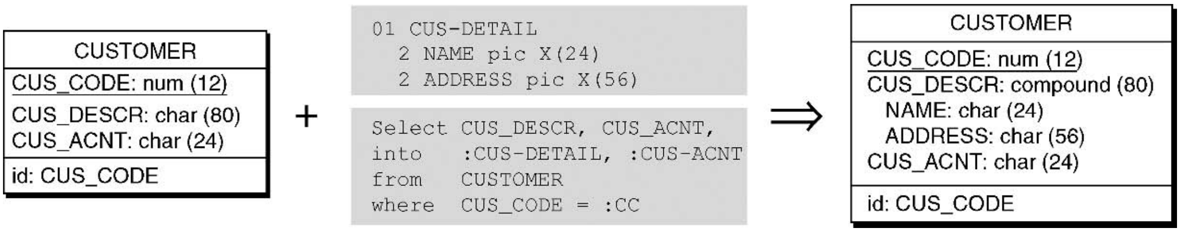
Data analysis. Mining the database contents can be used in two ways. First, to discover implicit properties, such as functional dependencies and foreign keys. Second, to check hypothetic constructs that have been suggested by other techniques. Considering the combinatorial explosion that threaten the first approach, data analysis is most often applied to check the existence of formerly identified patterns.

Program analysis. Understanding how programs use the data provides crucial information on properties of these data. Even simple analysis, such as dataflow graphs, can bring valuable information on field structure (Fig. 1) and meaningful names. More sophisticated techniques such as dependency analysis and program slicing can be used to identify complex constraint checking or foreign keys.

Screen/report layout analysis. Forms, reports and dialog boxes are user-oriented views on the database. They exhibit spatial structures (e.g., data aggregates), meaningful names, explicit usage guidelines and, at run time, data population that, combined with dataflow analysis, provide much information on implicit data structures and properties.

Schema Conceptualization

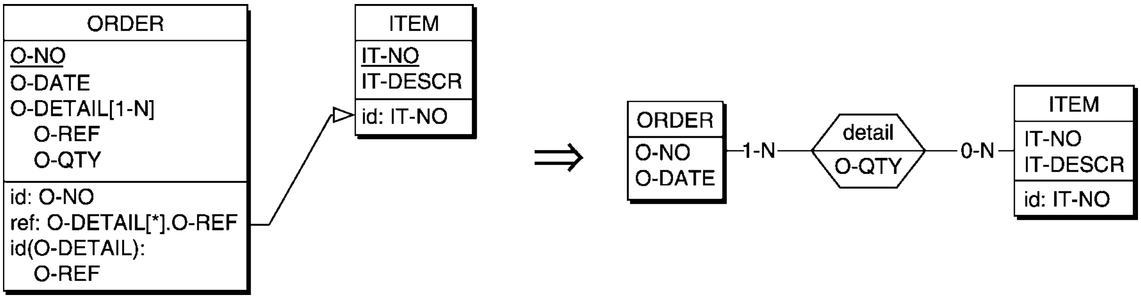
The goal of this process is to interpret the logical schema semantically by extracting a conceptual schema that represents its intended meaning. It mainly relies on transformational techniques that undo the effect of the logical design process. This complex process is decomposed in three subprocesses, namely



```
01 CUS-DETAIL
   2 NAME pic X(24)
   2 ADDRESS pic X(56)
```

```
Select CUS_DESCR, CUS_ACNT,
into   :CUS-DETAIL, :CUS-ACNT
from   CUSTOMER
where  CUS_CODE = :CC
```

Database Reverse Engineering. Figure 1. Illustration of the physical schema extraction and logical schema reconstruction processes.



Database Reverse Engineering. Figure 2. Conceptualization of a complex field.

untranslation, de-optimization and conceptual normalization. The *untranslation* process consists in reversing the transformations that have been used to draw the logical schema from the conceptual schema. For instance, each foreign key is interpreted as the implementation of a N:1 relationship type. This process relies on a solid knowledge of the rules and heuristics that have been used to design the database. Those rules can be standard, which makes the process fairly straightforward, but they can also be specific to the company or even to the developer in charge of the database (who may have left the company), in which case reverse engineering can be quite tricky. The main constructs that have to be recovered are relationship types (Fig. 2), super-type/subtype hierarchies, multivalued attributes, compound attributes and optional attributes. The *de-optimization* process removes the trace of all the optimization techniques that have been used to improve the performance of the database. Redundancies must be identified and discarded, unnormalized data structures must be decomposed and horizontal and vertical partitioning must be identified and undone. Finally, *conceptual normalization* improves the expressiveness, the simplicity, the readability and the extendability of the conceptual schema. It has the same goals and uses the same techniques as the corresponding process in Conceptual analysis.

Tools

Reverse engineering requires the precise analysis of huge documents such as programs of several millions of lines of code and schemas that include thousands of files and hundreds of thousands of fields. It also requires repeatedly applying complex rules on thousands of patterns. In addition, many reverse processes and techniques are common with those of forward engineering, such as transformations, validation and normalization. Finally, reverse engineering is only a step in larger projects, hence the need for integrated environments that combine forward and reverse tools and techniques [11].

Examples

Figure 1 illustrates the respective roles of the physical schema extraction and logical schema reconstruction processes. Parsing the DDL code identifies column CUS_DESCR as a large atomic field in the physical schema (left). Further dataflow analysis allows this column to be refined as a compound field (right).

The conceptualization of a compound field as a complex relationship type is illustrated in Figure 2. The multivalued field O-DETAIL has a component (O-REF) that serves both as an identifier for its values (the values of O-DETAIL in an ORDER record have distinct values of O-REF) and as a reference to an

ITEM record. This construct is interpreted as an N:N relationship type between ORDER and ITEM.

Key Applications

Database reverse engineering is most often the first step in information system maintenance, evolution [10], migration and integration. Indeed, such complex projects cannot be carried out when no complete, precise and up-to-date documentation of the database of the information system is available.

The scope of data reverse engineering progressively extends to other kinds of information such as web sites, electronic forms, XML data structures and any kind of semi-structured data. Though most techniques and tools specific to database reverse engineering remain valid, additional approaches are required, such as linguistic analysis and ontology alignment.

Cross-references

- ▶ Database Design
- ▶ Entity-Relationship Model
- ▶ Hierarchical Data Model
- ▶ Network Data Model
- ▶ Relational Model

Recommended Reading

1. Baxter I. and Mehlich M. Reverse engineering is reverse forward engineering. Sci. Comput. Programming, 36:131–147, 2000.
2. Blaha M.R. and Premerlani W.J. Observed idiosyncrasies of relational database designs. In Proc. 2nd IEEE Working Conf. on Reverse Engineering, 1995, p. 116.
3. Casanova M.A. and Amaral de Sa J.E. Mapping uninterpreted schemes into entity-relationship diagrams: two applications to conceptual schema design. IBM J. Res. Develop., 28(1):82–94, 1984.
4. Chikofsky E.J. and Cross J.H. Reverse engineering and design recovery: a taxonomy. IEEE Softw., 7(1):13–17, 1990.
5. Davis K.H. and Aiken P.H. Data reverse engineering: a historical view. In Proc. 7th Working Conf. on Reverse Engineering, 2000, pp. 70–78.
6. Davis K.H. and Arora A.K. A methodology for translating a conventional file system into an entity-relationship model. In Proc. 4th Int. Conf. on Entity-Relationship Approach, 1985, p. 148–159.
7. Edwards H.M. and Munro M. Deriving a logical model for a system using recast method. In Proc. 2nd IEEE Working Conf. on Reverse Engineering, 1995, pp. 126–135.
8. Hainaut J.-L. Introduction to database reverse engineering, LIBD lecture notes, Pub. University of Namur, Belgium, 2002, p. 160. Retrieved Oct. 2007 from http://www.info.fundp.ac.be/~dbm/publication/2002/DBRE-2002.pdf.
9. Hainaut J.-L. The transformational approach to database engineering. In Generative and Transformational Techniques in Software Engineering, R. Lämmel, J. Saraiva, J. Visser (eds.), LNCS 4143. Springer-Verlag, 2006, pp. 89–138.
10. Hainaut J.-L., Clève A., Henrard J., and Hick J.-M. Migration of Legacy Information Systems. In Software Evolution, T. Mens, S. Demeyer (eds.). Springer-Verlag, 2007, pp. 107–138.
11. Hainaut J.-L., Roland D., Hick J-M., Henrard J., and Englebert V. Database reverse engineering: from requirements to CARE tools. J. Automated Softw. Eng., 3(1/2):9–45, 1996.
12. Navathe S.B. and Awong A. Abstracting relational and hierarchical data with a semantic data model. In Proc. Entity-Relationship Approach: a Bridge to the User. North-Holland, 1987, pp. 305–333.

Database Scheduling

▶ Database Middleware

Database Security

Elena Ferrari
University of Insubria, Varese, Italy

Synonyms

Database protection

Definition

Database security is a discipline that seeks to protect data stored into a DBMS from intrusions, improper modifications, theft, and unauthorized disclosures. This is realized through a set of *security services*, which meet the security requirements of both the system and the data sources. Security services are implemented through particular processes, which are called *security mechanisms*.

Historical Background

Research in database security has its root in operating system security [6], whereas its developments follow those in DBMSs. Database security has many branches, whose main historical developments are summarized in what follows:

Access control. In the 1970s, as part of the research on System R at IBM Almaden Research Center, there was a lot of work on access control for relational DBMSs [3]. About the same time, some early work on Multilevel Secure Database Management Systems (MLS/DBMSs) was reported, whereas much of the development on MLS/DBMSs began [9] only after the Air Force Summer Study in 1982 [1]. This has resulted in different research prototypes, such as for instance those developed at MITRE, SRI International and Honeywell Inc. Access control models developed for relational databases have then been extended to cope with the new requirements of advanced DBMSs, such as object-oriented, object-relational, multimedia and active DBMSs [9], GIS [7], and XML DBMSs [5], and other advanced data management systems and applications, including digital libraries, data warehousing systems, and workflow management systems [9]. Role-based access control has been proposed in the 1990s [8] as a way to simplify authorization management within companies and organizations.

Privacy protection. Given the vast amount of personal data that is today collected by DBMSs, privacy is becoming a primary concern, and this has resulted in various research activities that have been started quite recently. A first research direction is related to *privacy-preserving data mining*, that is, how to apply data mining tools without compromising user privacy. All approaches developed so far are based on modifying or perturbing the data in some way. One of the issues is therefore how to maintain the quality of the modified data in such a way that they can be useful for data mining operations. Another line of research deals with the support of privacy policies, within the DBMS. In this direction, one of the most mature and promising proposal is the concept of Hippocratic database recently proposed by Agrawal et al. [3].

Auditing. Research on this issue has mainly focused on two directions: organization of the audit data and use of these data to discover possible security breaches. Another important research area is how to protect the audit data to prevent their malicious tampering.

Authentication. The simplest form of authentication is the one based on password. Throughout the years, several efforts have been made to make this scheme more robust against security threats or to develop schemes more suited to distributed and web-based environments. This is the case, for instance, of token-based schemes, that is, schemes based on biometric information, or *single sign-on* (SSO) schemes, particularly suited for distributed environments since they allow a user to authenticate once and gain access to the resources of multiple DBMSs. Recently, an innovative form of authentication has been proposed, based on user social relationships [4].

Foundations

Today data are one of the most crucial assets of any organization and, as such, their protection from intrusions, improper modifications, theft, and unauthorized disclosures is a fundamental service that any DBMS must provide [3,9,13]. Since data in a database are tightly related by semantic relationships, a damage of a data portion does not usually affect a single user or application, but the entire information system. Security breaches are typically categorized into the following categories: *unauthorized data observation*, *incorrect data modification*, and *data unavailability*. Security threats can be perpetrated either by outsiders or by users legitimately entitled to access the DBMS. The effect of unauthorized data observation is the disclosure of information to users not entitled to access such information. All organizations one may think of, ranging from commercial organizations to social or military organizations, may suffer heavy losses from both financial view and human point of view upon unauthorized data observation. Incorrect modifications of data or incorrect data deletion, either intentional or unintentional, results in an inconsistent database state. As a result, the database is not any longer correct. Any use of incorrect data may again result in heavy losses for the organization. Think, for instance, of medical data, where different observations of the same vital parameter may be used to make a clinical diagnosis. For the correct diagnosis it is crucial that each observation has not been incorrectly modified or deleted. When data are unavailable, information that are crucial for the proper functioning of the organization may not be readily accessible when needed. For instance, consider real-time systems, where the availability of some data may be crucial to immediately react to some emergency situations.

Therefore, data security requires to address three main issues:

1. *Data secrecy or confidentiality*. It prevents improper disclosure of information to unauthorized users. When data are related to personal information, *privacy* is often used as a synonym of data confidentiality. However, even if some techniques to protect confidentiality can be used to enforce privacy, protecting privacy requires some additional countermeasures. More precisely, since information privacy relates to an individual's right to determine how, when, and to what extent his or her personal

information will be released to another person or to an organization [11], protecting privacy requires to deal with additional issues with regard to confidentiality, such as, for instance, verifying that data are used only for the purposes authorized by the user and not for other purposes, or obtaining and recording the consents of users.

2. *Data integrity*. It protects data from unauthorized or improper modifications or deletions.
3. *Data availability*. It prevents and recovers data from hardware and software errors and from malicious data denials making the database or some of its portions not available.

Today, access to databases is mainly web-based and, in this context, two additional issues, besides the abovementioned one, should be addressed, in order to provide strong guarantees about data contents to users:

Data authenticity. It ensures that a user receiving some data can verify that the data have been generated by the claimed source and that they have not been maliciously modified.
Data completeness. The user can verify whether he or she has received all the data he or she requested.

A DBMS exploits the services of the underlying operating system to manage its data (for instance to store data into files). This applies also to security. However, protecting data stored into a DBMS is different from protecting data at the operating system level, and therefore it requires additional and more sophisticated mechanisms. The main reasons are the following:

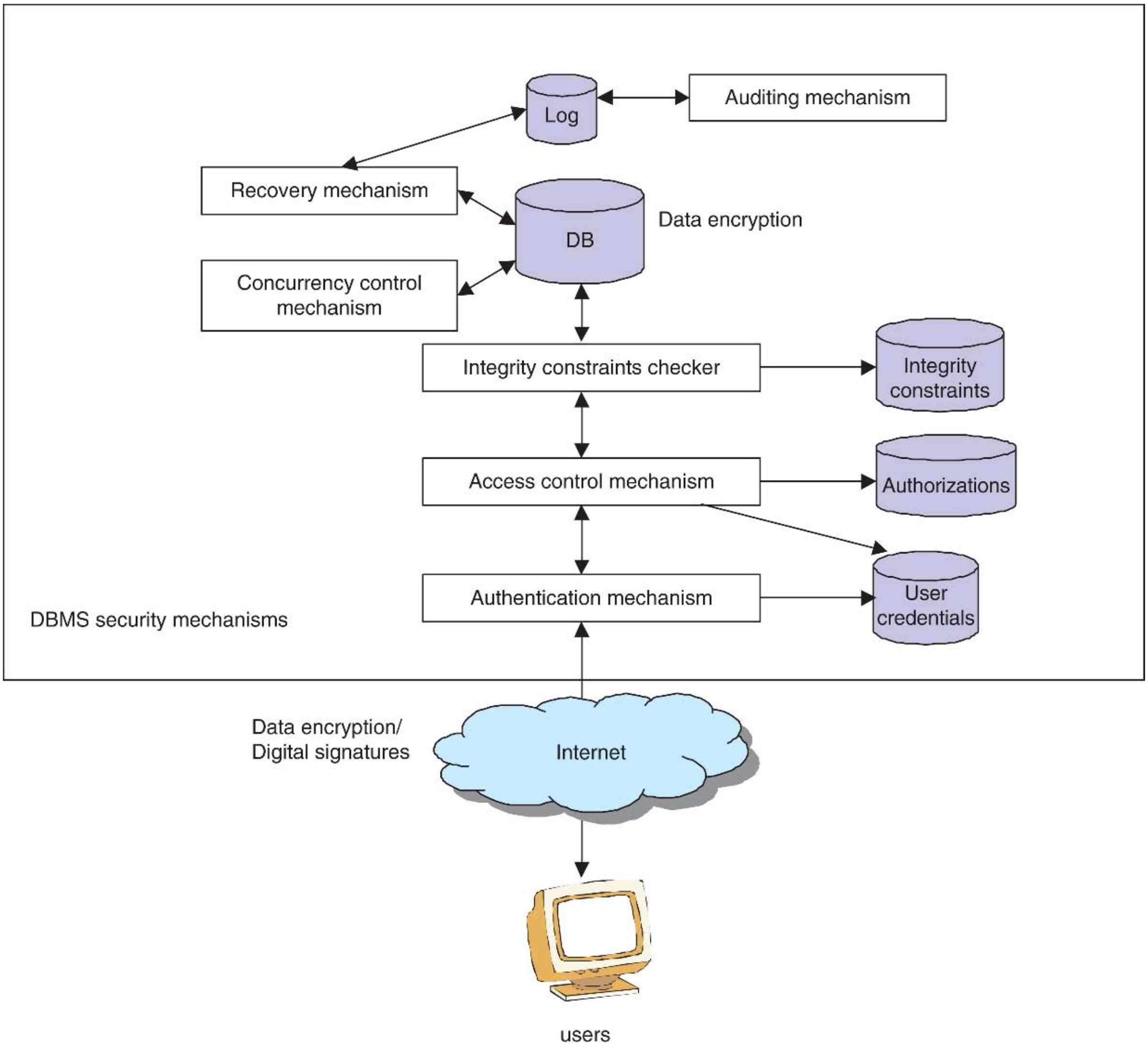
1. DBMSs and operating systems adopt different data models. In particular, data in a DBMS can be represented at different level of abstraction (physical, logical, view level), whereas an operating system adopts a unique representation of data (i.e., data are stored into files) and this simplifies data protection.
2. DBMSs are characterized by a variety of granularity levels for data protection. For instance, in a relational database, data can be protected at the relation or view level. However, sometimes finer granularity levels are needed, such as for instance selected attributes or selected tuples within a table. In contrast, in an operating system data protection is usually enforced at the file level.
3. In a database, objects at the logical level may be semantically related and these relations must be carefully protected. Moreover, several logical objects (e.g., different views) may correspond to the same physical object (e.g., the same file). These issues do not have to be considered when protecting data in an operating system.

Therefore, it is necessary that a DBMS is equipped with its own security services. Of course it can also exploit the security services provided by the underlying operating system, as well as those enforced at the hardware and network level. Generally, each security property is ensured by more than one DBMS service. In particular, the *access control mechanism* ensures data secrecy. Whenever a subject tries to access an object, the access control mechanism checks the right of the subject against a set of *authorizations*, stated usually by some Security Administrators or users. The access is granted only if it does not conflict with the stated authorizations. An *authorization* states which subject can perform which action on which object and, optionally, under which condition. Authorizations are granted according to the security policies of the organization. Data confidentiality is also obtained through the use of encryption techniques, either applied to the data stored on secondary storage or when data are transmitted on a network, to avoid that an intruder intercepts the data and accesses their contents. Data integrity is jointly ensured by the access control mechanism and by semantic integrity constraints. Similarly to confidentiality enforcement, whenever a subject tries to modify some data, the access control mechanism verifies that the subject has the right to modify the data, according to the specified authorizations and, only in this case, it authorizes the update request. Additionally, current DBMSs allow one to specify a set of *integrity constraints*, using SQL in case of an RDBMS, that expresses correctness conditions on the stored data, and therefore avoids incorrect data updates and deletions. These constraints are automatically checked by the constraint checker subsystem upon the request for each update operation. Furthermore, digital signature techniques can be applied to detect improper data modifications. They are also used to ensure data authenticity. Finally, the recovery subsystem and the concurrency control mechanism ensure that data are available and correct despite hardware and software failures and despite data accesses from concurrent application programs. In particular, to properly recover the correct state of the database after a failure, all data accesses are logged. Log files can then be further used for *auditing* activities,

Database Security. Table 1. Security requirements and enforcement techniques

Security properties	Techniques
Secrecy	Access control mechanism, data encryption
Integrity	Access control mechanism, semantic integrity constraints, digital signatures
Availability	Recovery subsystem, concurrency control mechanism,
	techniques preventing DoS attacks
Authenticity	Digital signatures
Completeness	Non standard digital signatures

that is, they can be analyzed by an *intrusion detection system* to discover possible security breaches and their causes. Data availability, especially for data that are available on the web, can be further enhanced by the use of techniques avoiding query floods [2] or other Denial-of-Service (DoS) attacks. Completeness enforcement is a quite new research direction, which is particularly relevant when data are outsourced to (untrusted) publishers for their management [12]. It can be ensured through the use of nonstandard digital signature techniques, like Merkle trees and aggregation signatures. Table 1 summarizes the security properties discussed so far and the techniques for their enforcement.



Database Security. Figure 1. Security mechanisms.

The above described security services must rely for their proper functioning on some *authentication mechanism*, which verifies whether a user wishing to connect to the DBMS has the proper credentials. Such a mechanism identifies users and confirms their identities. Commonly used authentication mechanisms are based on the use of login and password, whereas more sophisticated schemes include those using biometric information, or token-based authentication.

The security mechanisms discussed so far and their interactions are graphically summarized in Fig. 1. Additionally, security mechanisms should be devised when data are accessed through web applications. These applications can be exploited to perpetrate one of the most serious threats to web-accessible databases, that is, *SQL-Injection Attacks* (SQLIAs) [10]. In the worst case, these attacks may cause the intruder to gain access to all the data stored into the database, by by-passing the security mechanisms, and, therefore, it gives him or her the power to leak, modify, or delete the information that is stored in the database. SQLIAs are very widespread, as reported by a study by Gartner Group over 300 Internet websites, which has shown that most of them could be vulnerable to SQLIAs. SQLIAs basically exploit insufficient input validation in the application code. One of the simplest form of SQLJ attack is that of inserting SQL meta-characters into web-based input fields to manipulate the execution of the back-end SQL queries. Although several techniques to prevent such attacks have been recently proposed [10], there are so many variants of SQLIAs that finding a complete solution to these threats is still a challenging research issue.

Key Applications

Database security services are nowadays used in any application environment that exploits a DBMS to manage data, including Healthcare systems, Banking and financial applications, Workflow Management Systems, Digital Libraries, Geographical and Multimedia management systems, E-commerce services, Publish-subscribe systems, Data warehouses.

Cross-references

- ▶ Access Control
- ▶ Auditing and Forensic Analysis
- ▶ Authentication
- ▶ Concurrency Control–Traditional Approaches
- ▶ Data Encryption
- ▶ Digital Signatures
- ▶ Intrusion Detection Technologies
- ▶ Merkle Trees
- ▶ Privacy
- ▶ Privacy-Enhancing Technologies
- ▶ Privacy-Preserving Data Mining
- ▶ Secure Data Outsourcing
- ▶ Secure Database Development
- ▶ Security Services

Recommended Reading

1. Air Force Studies Board and Committee on Multilevel data management security. Multilevel data management security. National Academy, WA, USA, 1983.
2. Bertino E., Laggieri D., and Terzi E. Securing DBMS: characterizing and detecing query flood. In Proc. 9th Information Security Conference, 2004, pp. 195–206.
3. Bertino E. and Sandhu R.S. Database security: concepts, approaches, and challenges. IEEE Trans. Depend. Secure Comput., 2(1):2–19, 2005.
4. Brainard J., Juels A., Rivest R.L., Szydlo M., and Yung M. Fourth-factor authentication: somebody you know. In Proc. 13th ACM Conf. on Computer and Communications Security, 2006.
5. Carminati B., Ferrari E., and Thuraisingham B.M. Access control for web data: models and policy languages. Annals Telecomm., 61(3–4):245–266, 2006.
6. Castano S., Fugini M.G., Martella G., and Samarati P. Database security. Addison-Wesley, Reading, MA, 1995.
7. Damiani M.L. and Bertino E. Access control systems for geo-spatial data and applications. In Modelling and Management of Geographical Data over Distributed Architectures, A. Belussi, B. Catania, E. Clementini, E. Ferrari (eds.). Springer, 2007, pp. 189–214.
8. Ferraiolo D.F., Sandhu R.S., Gavrila S.I., Kuhn D.R., and Chandramouli R. Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur., 4(3):224–274, 2001.
9. Ferrari E. and Thuraisingham B.M. Secure database systems. In Advanced Databases: Technology and Design, O. Diaz, M. Piattini (eds.). Artech House, London, 2000.
10. Halfond W.G., Viegas J., and Orso A. A classification of SQL-injection attacks and countermeasures. Int. Symp. on Secure Software Engineering, 2006.
11. Leino-Kilpi H., Valimaki M., Dassen T., Gasull M., Lemonidou C., Scott A., and Arndt M. Privacy: a review of the literature. Int. J. Nurs. Stud., (38):663–671, 2001.
12. Pang H. and Tan K.L. Verifying completeness of relational query answers from online servers. ACM Trans. Inf. Syst. Secur., 11(2), 2008, article no. 5.
13. Pfleeger C.P. and Pfleeger S.L. Security in computing, 3rd edn. Prentice-Hall, Upper Saddle River, NJ, USA, 2002.

Database Socket

▶ Connection

Database Storage Layer

► Storage Access Models

Database Techniques to Improve Scientific Simulations

Biswanath Panda, Johannes Gehrke,
Mirek Riedewald
Cornell University, Ithaca, NY, USA

Synonyms

Indexing for online function approximation

Definition

Scientific simulations approximate real world physical phenomena using complex mathematical models. In most simulations, the mathematical model driving the simulation is computationally expensive to evaluate and must be repeatedly evaluated at several different parameters and settings. This makes running large-scale scientific simulations computationally expensive. A common method used by scientists to speed up simulations is to store model evaluation results at some parameter settings during the course of a simulation and reuse the stored results (instead of direct model evaluations) when similar settings are encountered in later stages of the simulation. Storing and later retrieving model evaluations in simulations can be modeled as a high dimensional indexing problem. Database techniques for improving scientific simulations focus on addressing the new challenges in the resulting indexing problem.

Historical Background

Simulations have always been an important method used by scientists to study real world phenomena. The general methodology in these application areas is similar. Scientists first understand the physical laws that govern the phenomenon. These laws then drive a mathematical model that is used in simulations as an approximation of reality. In practice scientists often face serious computational challenges. The more realistic the model, the more complex the corresponding mathematical equations. As an example, consider the simulation of a combustion process that motivated the line of work discussed in this entry. Scientists study how the composition of gases in a combustion chamber changes over time due to chemical reactions. The composition of a gas particle is described by a high-dimensional vector (10–70 dimensions). The simulation consists of a series of time steps. During each time step some particles in the chamber react, causing their compositions to change. This reaction is described by a complex high-dimensional function called the reaction function, which, given the current composition vector of a particle and other simulation properties, produces a new composition vector for the particle. Combustion simulations usually require up to 10^8–10^{10} reaction function evaluations each of which requires in the order tens of milliseconds of CPU time. As a result, even small simulations can run into days.

Due to their importance in engineering and science, many algorithms have been developed to speed up combustion simulations. The main idea is to build an approximate model of the reaction function that is cheaper to evaluate. Early approaches were offline, where function evaluations were collected from simulations and used to learn multivariate polynomials approximating the reaction function [8]. These polynomials were then used later in different simulations instead of the reaction function. Recently, more sophisticated models like neural networks and self organizing maps have also been used [3]. The offline approaches were not very successful because a single model could not generalize to a large class of simulations. In 1997, Pope developed the In Situ Adaptive Tabulation (ISAT) Algorithm [7]. ISAT was an online approach to speeding up combustion simulations. The algorithm cached reaction function evaluations at certain frequently seen regions in the composition space and then used these cached values to approximate the reaction function evaluations at compositions encountered later on in the simulation. The technique was a major breakthrough in combustion simulation because it enabled scientists to run different simulations without having to first build a model for the reaction function. Until today, the Algorithm remains the state of the art for combustion simulations. Several improvements to ISAT have been proposed. DOLFA [9] and PRISM [1] proposed alternative methods of caching reaction function evaluations. More recently, Panda et al. [6] studied the storage/retrieval problem arising out of caching/reusing reaction function evaluations in ISAT and this entry mainly discusses their observations and findings.

Foundations

Even though the ISAT algorithm was originally proposed in the context of combustion simulations the algorithm can be used for building an approximate model for any high dimensional function (f), that is expensive to compute. This section begins with a discussion of Local Models, that represent the general class of models built by ISAT (section "Local Models"). This is followed by a description of the ISAT Algorithm that uses selective evaluations of the expensive function f to build a local model (section "ISAT Algorithm"). The algorithm introduces a new storage/retrieval, and hence indexing, problem. The section then discusses the indexing problem in detail: its challenges and solutions that have been proposed (sections "Indexing Problem" and "An Example: Binary Tree") and concludes with some recent work on optimizing long running simulations (section "Long Running Simulations").

Local Models

Local Models are used in many applications to approximate complex high dimensional functions. Given a function $f : \mathbf{R}^m \rightarrow \mathbf{R}^n$; a local model defines a set of high dimensional regions in the function domain: $\mathcal{R} = \{R_1 ... R_n | R_i \subseteq \mathbf{R}^m\}$. Each region R_i is associated with a function $\hat{f}_{R_i} : R_i \rightarrow \mathbf{R}^n$; such that $\forall \mathbf{x} \in R_i : \ ||\hat{f}_{R_i}(\mathbf{x}) - f(\mathbf{x})\ || \leq \epsilon$; where ε is a specified error tolerance in the model and || is some specified error metric such as the Euclidean distance. Using a local model to evaluate f at some point $\mathbf{x}$ in the function domain involves first finding a region $(R \in \mathcal{R})$ that contains $\mathbf{x}$ and then evaluating $\hat{f}_R(\mathbf{x})$ as an approximation to $f(\mathbf{x})$.

ISAT Algorithm

Main algorithm: ISAT is an online algorithm for function approximation; its pseudocode is shown in Fig. 1. The algorithm takes as input a query point $\mathbf{x}$ at which the function value must be computed and a search structure S that stores the regions in $\mathcal{R}$. S is empty when the simulation starts. The algorithm first tries to compute the function value at $\mathbf{x}$ using the local model it has built so far (Lines 2–3). If that fails the algorithm computes $f(\mathbf{x})$ using the expensive model (Line 5) and uses $f(\mathbf{x})$ to update existing or add new regions in the current local model (Line 6). The algorithm is online because it does not have access to all query points when it builds the model.

```
ISAT Algorithm
Require: Query Point x, inde x structure $S$
if $\exists\langle R, \hat{f}_R\rangle \in S$ such that x $\in R$
    Compute y =$\hat{f}$ (x)
else
    Compute y = $f$ (x)
    Update($S$, x, $f$ (x))
end if
return y
```

Database Techniques to Improve Scientific Simulations. Figure 1. Pseudocode of the ISAT algorithm.

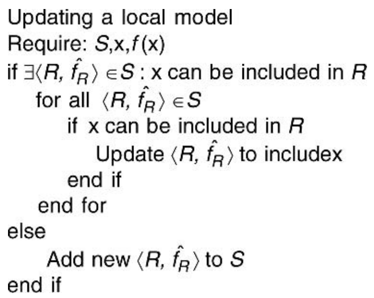
Model updating: ISAT updates the local model using a strategy outlined in Fig. 2. In general it is extremely difficult to exactly define a region R and an associated $\hat{f}_R$, such that $\hat{f}_R$ approximates f in all parts of R. ISAT proposes a two step process to discover regions. It initially starts with a region that is very small and conservative but where it is known that a particular $\hat{f}_R$ approximates f well. It then gradually grows the conservative approximations over time. More specifically, the update process first searches the index S for regions R where $\mathbf{x}$ lies outside R but $||\hat{f}_R(\mathbf{x}) - f(\mathbf{x})|| \leq \epsilon$. Such regions are grown to contain $\mathbf{x}$ (Lines 2–7). If no existing regions can be grown a new conservative region centered around $\mathbf{x}$ and associated $\hat{f}_R$ is added to the local model (Line 9). The grow process described is a heuristic that works well in practice for functions that are locally smooth. This assumption holds in combustion and in many other applications.

Instantiation: While the shape of the regions and associated functions can be arbitrary, the original ISAT algorithm proposed high dimensional ellipsoids as regions and used a linear model as the function in a region. The linear model is initialized by computing the f value and estimating the derivative of f at the center of the ellipsoidal region:

$$\hat{f}_R(\mathbf{x}) = f(\mathbf{a}) + F'_{\mathrm{a}} \times (\mathbf{x} - \mathbf{a}),$$

where $\mathbf{a}$, is the center of the region R and $F'_{\mathbf{a}}$ is the derivative at $\mathbf{a}$.

ISAT performs one of the following high level operations for each query point $\mathbf{x}$. **Retrieve**: Computing the function value at $\mathbf{x}$ using the current local model by searching for a region containing $\mathbf{x}$. **Grow**: Searching for regions that can be grown to contain $\mathbf{x}$ and updating these regions in S. **Add**: Adding a new region (R) and an associated $\hat{f}_R$ into S.



```
Updating a local model
Require: $S$,x,$f$(x)
if $\exists\langle R, \hat{f}_R\rangle \in S$ : x can be included in $R$
    for all $\langle R, \hat{f}_R\rangle \in S$
        if x can be included in $R$
            Update $\langle R, \hat{f}_R\rangle$ to includex
        end if
    end for
else
    Add new $\langle R, \hat{f}_R\rangle$ to $S$
end if
```

Database Techniques to Improve Scientific Simulations. Figure 2. Pseudocode for updating a local model.

Indexing Problem

The indexing problem in function approximation produces a challenging workload for the operations on index *S* in Figs. 1 and 2. The retrieve requires the index to support fast lookups. The grow requires both a fast lookup to find growable ellipsoids and then an efficient update process once an ellipsoid is grown. Finally, an efficient insert operation is required for the add step. There are two main observations which make this indexing problem different from traditional indexing [2,4]:

- The regions that are stored in the index are not predefined but generated by the add and grow operations. Past decisions about growing and adding affect future performance of the index, therefore the algorithm produces an uncommon query/update workload.
- The framework in which indexes must be evaluated is very different. Traditionally, the performance of index structures has been measured in terms of the cost of a search and in some cases update. There are two distinct cost factors in the function approximation problem. First, there are the costs associated with the search and update operations on the index. Second, there are costs of the function approximation application which include function evaluations and region operations. Since the goal of function approximation is to minimize the total cost of the simulation, all these costs must be accounted for when evaluating the performance of an index.

In the light of these observations a principled analysis of the various costs in the function approximation algorithm leads to the discovery of novel tradeoffs. These tradeoffs produce significant and different effects on different index structures. Due to space constraints, only a high-level discussion of the various costs in the algorithm and the associated tradeoffs is included. The remainder of this section briefly describes the tradeoffs and the tuning parameters that have been proposed to exploit the different tradeoffs. The indexing problem is studied here using the concrete instantiation of the ISAT algorithm using ellipsoidal regions with linear models. Therefore, regions are often referred to as ellipsoids in the rest of the section. However, it is important to note that the ideas discussed are applicable to any kind of regions and associated functions.

Tuning Retrieves

In most high dimensional index structures the ellipsoid containing a query point is usually not the first ellipsoid found. The index ends up looking at a number of ellipsoids before finding "the right one." The additional ellipsoids that are examined by the index are called false positives. For each false positive the algorithm pays to search and retrieve the ellipsoid from the index and to check if the ellipsoid contains the query point. In traditional indexing problems, if an object that satisfies the query condition exists in the index, then finding this object during search is mandatory. Therefore, the number of false positives is a fixed property of the index. However, the function approximation problem provides the flexibility to tune the number of false positives, because the expensive function can be evaluated if the index search was not successful. The number of false positives can be tuned by limiting the number of ellipsoids examined during the retrieve step. This parameter is denoted by Ellr. Ellr places an upper bound on the number of false positives for a query. Tuning Ellr controls several interesting effects.

- *Effect 1:* Decreasing Ellr reduces the cost of the retrieve operation as fewer ellipsoids are retrieved and examined.
- *Effect 2:* Decreasing Ellr decreases the chances of finding an ellipsoid containing the query point thereby resulting in more expensive function evaluations.
- *Effect 3:* Misses that result from decreasing Ellr can grow and add other ellipsoids. These grows and adds index new parts of the domain and also change the overall structure of the index. Both of these affect the probability of retrieves for future queries. This is a more subtle effect unique to this problem.

Tuning Grows and Adds

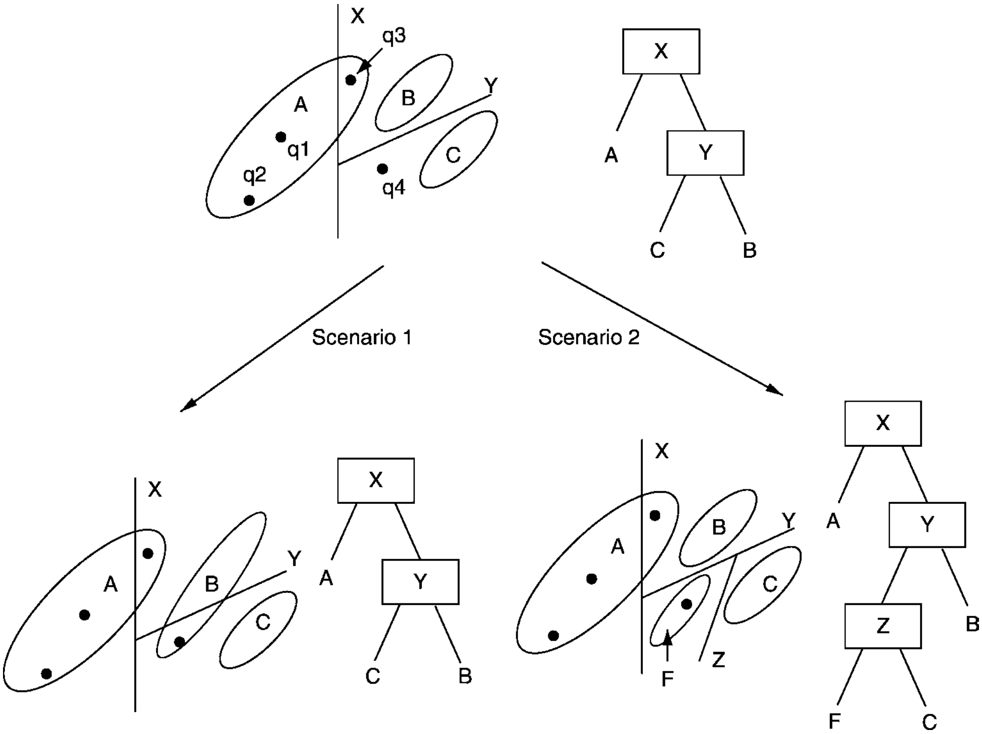
Just like the retrieve, the grow and add operations can be controlled by the number of ellipsoids examined for growing denoted as Ellg. Since an add is performed only if a grow fails, this parameter controls both the operations. Ellg provides a knob for controlling several effects that affect performance of the index and the algorithm.

- *Effect 4:* The first part of the grow process involves traversing the index to find ellipsoids that can be grown. Decreasing Ellg reduces the time spent in the traversal.
- *Effect 5:* Decreasing Ellg decreases the number of ellipsoids examined for the grow and hence the number of ellipsoids actually grown. This results in the following effects.
  - *Effect 5a:* Reducing the number of ellipsoids grown reduces index update costs that can be significant in high dimensional indexes.
  - *Effect 5b:* Growing a large number of ellipsoids on each grow operation indexes more parts of the function domain, thereby improving the probability of future retrieves.
  - *Effect 5c:* Growing a large number of ellipsoids on each grow results in significant overlap among ellipsoids. Overlap among objects being indexed reduces search selectivity in many high dimensional indexes.
- *Effect 6:* Decreasing Ellg increases the number of add operations. Creating a new region is more expensive than updating an existing region since it involves initializing the function $\hat{f}_R$ in the new region.

In summary, the two tuning parameters have many different effects on index performance and the cost of the simulation. What makes the problem interesting is that these effects often move in opposite directions. Moreover, tuning affects different types of indexes differently and to varying degrees, which makes it necessary to analyze each type of index individually.

An Example: Binary Tree

The previous section presented a qualitative discussion of the effects that tuning Ellr and Ellg can have on index performance and simulation cost. This section makes the effects outlined in the previous section more concrete using an example index structure, called the Binary Tree. The tree indexes the centers of the ellipsoids by recursively partitioning the space with cutting



Database Techniques to Improve Scientific Simulations. Figure 3. Binary tree.

planes. Leaf nodes of the tree correspond to ellipsoid centers and non-leaf nodes represent cutting planes. Figure 3 shows an example tree for three ellipsoids *A*, *B*, *C* and two cutting planes *X* and *Y*. We focus on the tree in the top part of Fig. 3 and use it to describe the operations supported by the index.

Retrieve

There are two possible traversals in the index that result in a successful retrieve.

Primary Retrieve. The first called a Primary Retrieve is illustrated with query point $\mathbf{q}_2$. The retrieve starts at the root, checking on which side of hyperplane *X* the query point lies. The search continues recursively with the corresponding subtree, the left one in the example. When a leaf node is reached, the ellipsoid in the leaf is checked for the containment of the query point. In the example, *A* contains $\mathbf{q}_2$, and hence, a successful retrieve.

Secondary Retrieve. Since the binary tree only indexes centers, ellipsoids can straddle cutting planes, e.g., *A* covers volume on both sides of cutting plane *X*. If ellipsoids are straddling planes, then the Primary Retrieve can result in a false negative. For example, $\mathbf{q}_3$ lies to the right of *X* and so the Primary Retrieve fails even though there exists an ellipsoid *A* containing it. To overcome this problem the Binary Tree performs a more expensive Secondary Retrieve if the Primary fails. The main idea of the Secondary Retrieve is to explore the "neighborhood" around the query point by examining "nearby" subtrees. In the case of $\mathbf{q}_3$, the failed Primary Retrieve ended in leaf *B*. Nearby subtrees are explored by moving up a level in the tree and exploring the other side of the cutting plane. Specifically, *C* is examined first(after moving up to *Y*, *C* is in the unexplored subtree). Then the search would continue with *A* (now moving up another level to *X* and accessing the whole left subtree). This process continues until a containing ellipsoid is found, or Ellr ellipsoids have been examined unsuccessfully.

Update

Scenario 1 (Grow) and 2 (Add) of Fig. 3 illustrate the update operations on the index.

Grow. The search for growable ellipsoids proceeds in exactly the same way as a Secondary Retrieve, starting where the failed Primary Retrieve ended. Assume that in the example in Fig. 3, ellipsoid *B* can be grown to include $\mathbf{q}_4$, but *C* and *A* cannot. After the retrieve failed, the grow operation first attempts to grow *C*. Then it continues to examine *B*, then *A* (unless Ellg < 3). *B* is grown to include $\mathbf{q}_4$, as shown on the bottom left (Scenario 1). Growing of *B* made it straddle hyperplane *Y*. Hence, for any future query point near $\mathbf{q}_4$ and "below" *Y*, a Secondary Retrieve is necessary to find containing ellipsoid *B*, which is "above" *Y*.

Add. The alternative to growing *B* is illustrated on the bottom right part of Fig. 3 (Scenario 2). Assume Ellg = 1, i.e., after examining *C*, the grow search ends unsuccessfully. Now a new ellipsoid *F* with center $\mathbf{q}_4$ is added to the index. This is done by replacing leaf *C* with an inner node , which stores the hyper-plane that best separates *C* and *F*. The add step requires the expensive computation of *f*, but it will enable future query points near $\mathbf{q}_4$ to be found by a Primary Retrieve.

Tuning parameter Ellg affects the Binary Tree in its choice of scenario 2 over 1. This choice, i.e., performing an add instead of a grow operation, reduces false positives for future queries, but adds extra-cost for the current query. Experiments on real simulation workloads have shown that this tradeoff has a profound influence on the overall simulation cost [6].

Long Running Simulations

When ISAT is used in long running combustion simulations ($\geq 10^8$ time steps), updates to the local model are unlikely after the first few million queries and the time spent in building the local model is very small compared to the total simulation time. Based on these observations, Panda et al. have modeled a long running combustion simulation as a traditional supervised learning problem [5]. They divide a combustion simulation into two phases. During the first phase, the ISAT algorithm is run and at the same time ($\mathbf{x}$,f($\mathbf{x}$)) pairs are sampled uniformly from the composition space accessed by the simulation. At the end of the first phase, the sampled ($\mathbf{x}$,f($\mathbf{x}$)) pairs are used as training data for a supervised learning algorithm that tries to find a "new" local model with lower retrieve cost than the model built using ISAT. This new model is then used for the remainder of the simulation. Their experiment shows that the algorithm adds little overhead and that the new model can reduce retrieve costs by up to 70% in the second phase of the simulation.

Key Applications

The ISAT algorithm and its optimizations have primarily been applied to combustion simulation workloads.

However, the ideas are applicable to any simulation setting that requires repeated evaluations in a fixed domain of a function that is locally smooth and expensive to compute are required.

Cross-references

► Spatial and Multidimensional Databases

Recommended Reading

1. Bell J.B., Brown N.J., Day M.S., Frenklach M., Grcar J.F., Propp R.M., and Tonse S.R. Scaling and efficiency of PRISM in adaptive simulations of turbulent premixed flames. In Proc. 28th Int. Combustion Symp., 2000.
2. Böhm C., Berchtold S., and Keim D.A. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. ACM Comput. Surv., 33(3):322–373, 2001.
3. Chen J.Y., Kollmann W., and Dibble R.W. A self-organizing-map approach to chemistry representation in combustion applications. Combust. Theory and Model., 4:61–76, 2000.
4. Gaede V. and Günther O. Multidimensional access methods. ACM Comput. Surv., 30(2):170–231, 1998.
5. Panda B., Riedewald M., Gehrke J., and Pope S.B. High speed function approximation. In Proc. 2007 IEEE Int. Conf. on Data Mining, 2007.
6. Panda B., Riedewald M., Pope S.B., Gehrke J., and Chew L.P. Indexing for function approximation. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.
7. Pope S.B. Computationally efficient implementation of combustion chemistry using in situ adaptive tabulation. Combust. Theory Model., 1:41–63, 1997.
8. Turanyi T. Application of repro-modeling for the reduction of combustion mechanisms. In Proc. 25th Symp. on Combustion, 1994, pp. 949–955.
9. Veljkovic I., Plassmann P., and Haworth D.C. A scientific on-line database for efficient function approximation. In Proc. Int. Conf. on Computational Science and its Applications, 2003, pp. 643–653.

Database Trigger

Mikael Berndtsson, Jonas Mellin
University of Skövde, Skövde, Sweden

Synonyms

Triggers

Definition

A database trigger is code that is executed in response to events that have been generated by database commands such as INSERT, DELETE, or UPDATE.

Key Points

Triggers are code that are executed in response to events that have been generated before or after a database operation. They are sometimes separated as pre- and post-triggers in the literature. A pre-trigger can be used as an extra validation check before the database command is executed, whereas a post-trigger can be used as a notification that the database command has been executed.

Triggers can be classified according to trigger granularity: *row-level triggers* or *statement-level triggers*. In case of row-level triggers, each row will generate an event, whereas statement-level triggers occur only once per database command.

Overviews of database triggers can be found in [1,2].

Cross-references

► Active Database (aDB)
► Active Database (Management) System (aDBS/aDBMS)
► ECA Rules
► Event

Recommended Reading

1. Kulkarni K.G., Mattos N.M., and Cochrane R. Active Database Features in SQL3. In Active Rules in Database Systems. 1999, pp. 197–219.
2. Sudarshan S., Silberschatz A., and Korth H. Triggers, chap. 8.6. 2006, pp. 329–334.

Database Tuning using Combinatorial Search

Surajit Chaudhuri[1], Vivek Narasayya[1],
Gerhard Weikum[2]
[1]Microsoft Corporation, Redmond, WA, USA
[2]Max-Planck Institute for Informatics, Saarbrücken, Germany

Definition

Some database tuning problems can be formulated as combinatorial search, i.e., the problem of searching over a large space of discrete system configurations to find an appropriate configuration. One tuning problem where feasibility of combinatorial search has been demonstrated is physical database design. As part of the *self-management* capabilities of a database system, it is desirable to develop techniques for automatically recommending an appropriate physical design configuration to optimize database system performance. This entry describes the application of combinatorial search techniques to the problem of physical database design.

Historical Background

Combinatorial search (also referred to as combinatorial optimization) [8] is branch of optimization where the set of feasible solutions (or configurations) to the problem is *discrete*, and the goal is to find the "best" possible solution. Several well-known problems in computer science such as the Traveling Salesman Problem, Minimum Spanning Tree Problem, Knapsack Problem etc. can be considered as examples of combinatorial search. Several combinatorial search problems have been shown to be NP-Hard, and exact algorithms that guarantee optimality are not scalable. In such cases, heuristic search algorithms such as greedy search, simulated annealing, genetic algorithms etc. are often used to ensure scalability.

In the area of database tuning problems, the combinatorial search paradigm has been successfully used for problems such as query optimization [2] and physical database design tuning [5]. These are described in more details below.

Foundations

Some of the key aspects of combinatorial search are:

- The *search space*, i.e., the space of discrete configurations from which the solution is picked.
- A *metric* for evaluating the "goodness" of a configuration in the search space. This is essential for being able to quantitatively compare different configurations.
- A *search algorithm* for efficiently searching the space to find a configuration with the minimum (or maximum) value of the goodness metric.

One early example of use of combinatorial search for tuning in database systems is query optimization (see [2] for an overview on query optimization in relational database systems). The goal of query optimization is to produce an efficient execution plan for a given query using the physical operators supported by the underlying execution engine. The above key aspects of combinatorial search are now illustrated for the problem of query optimization. The *search space* of execution plans considered by a query optimizer depends on: (i) The *set of algebraic transformations* that preserve equivalence of query expressions (e.g., commutativity and associativity of joins, commutativity of group by and join, rules for transforming a nested subquery into a single-block query, etc.) (ii) The *set of physical operators* supported in the system (e.g., three different physical operators Index Nested Loops Join, Hash Join, Merge Join for implementing a join). Query optimizers use a *cost model* that defines the goodness *metric* for evaluating the "goodness" of an execution plan. In particular for a given execution plan, the cost model computes an overall number based on estimates of the CPU, I/O and communication costs of physical operators in the execution plan. Finally, different kinds of *search algorithms* are used in today's query optimizers including bottom-up approaches (e.g., in the Starburst query optimizer [7]) as well as top-down approaches (e.g., optimizers based on the Volcano/ Cascades [6] framework).

Example: Physical Database Design using Combinatorial Search

An in-depth example is now considered, the problem of physical database design. A crucial property of a relational DBMS is that it provides physical data independence. This allows physical structures such as indexes and materialized views to be added or dropped without affecting the output of the query; but such changes do impact efficiency. Thus, together with the capabilities of the execution engine and the optimizer, the physical database design determines how efficiently a query is executed on a DBMS. Over the years, the importance of physical design has become amplified as query optimizers have become sophisticated to cope with complex decision support queries. Since query execution and optimization techniques have become more sophisticated, database administrators (DBAs) can no longer rely on a simplistic model of the engine. Thus, tools for automating physical database design can ease the DBA's burden and greatly reduce the total cost of ownership. For an overview of work in the area of physical database design, refer to [5].

The role of the *workload*, including SQL queries and updates, in physical design is widely recognized. Therefore, the problem of physical database design can be stated as: For a given workload, find a configuration, i.e., a set of indexes and materialized views that minimize the cost of the given workload. Typically, there is a constraint on the amount of storage space that the configuration is allowed to take. Since the early 1970s, there has been work on trying to automate physical database design. However, it is only in the past decade that automated physical database design tools have become part of all major commercial DBMS vendors [5].

Search Space: The search space of configurations (i.e., set of physical design structures) for a given query (and hence the given workload) can be very large. First, the set of physical design structures that are *relevant* to a query i.e., can potentially be used by the query optimizer to answer the query, is itself large. For example, consider a query with *n* selection predicates. An index defined on any subset of the columns referenced in the predicates is relevant. The order of columns in the index is also significant; thus, in principle any permutation of the columns of the subset also defines a relevant index. The space of relevant materialized views is larger than the space of indexes since materialized views can be defined on any subset of tables referenced in the query. Finally, the search space of *configurations* for the physical database design problem is the power set of all *relevant indexes and materialized views.*

Today's physical database design tools approach the issue of large search space by adopting heuristics for restricting the search space. For example in [3], a key decision is to define the search space as consisting of the union of (only) the *best* configurations for each query in the workload, where the best configuration itself is the one with lowest optimizer estimated cost for the query. Intuitively, this *candidate selection step* leverages the idea that an index (or materialized view) that is not part of an optimal (or close to optimal) configuration for at least one query, is unlikely to be optimal for the entire workload. To improve the quality of the solution in the presence of constraints (e.g., a storage bound), the above space is augmented with an additional set of indexes (and materialized views) derived by "merging" or "reducing" structures from the above space (e.g., [1]). These additional candidates exploit commonality across queries in the workload, and even though they may not be optimal for any individual query in the workload, they can be optimal for the workload as a whole in the presence of the constraint.

Metric: It is not feasible to estimate goodness of a configuration for a workload by actual creation of physical design structures and then executing the queries and updates in the workload. Early papers on physical design tuning used an external model to estimate the cost of a query for a given configuration. However, this has the fundamental problem that the decisions made by the physical design tool could be "out-of-sync" with the decisions made by the query optimizer. This can lead to a situation where the physical design tool recommends an index that is never used by the query optimizer to answer any query in the workload.

In today's commercial physical design tools, the goodness of a configuration for a query is measured by the *optimizer estimated cost* of the query for that configuration. Unlike earlier approaches that used an external cost model, this approach has the advantage that the physical design tool is "in-sync" with the query optimizer.

One approach for enabling this measure of goodness is by making the following key server-side enhancements: (i) Efficient creation of a hypothetical (or "what-if") index. This requires metadata changes to signal to the query optimizer the presence of a what-if index (or materialized view). (ii) An extension to the "Create Statistics" command to efficiently generate the statistics that describe the distribution of values of the column(s) of a what-if index via the use of sampling. (iii) A query optimization mode that enabled optimizing a query for a selected subset of indexes (hypothetical or actually materialized) and ignoring the presence of other access paths. This is important as the alternative would have been repeated creation and dropping of what-if indexes, a potentially costly solution. For more details, refer to [4].

Search algorithm: Given a workload and a set of candidate physical design structures (e.g., obtained as described above using the candidate selection step), the goal of the search algorithm is to efficiently find a configuration (i.e., subset of candidates) with the smallest total optimizer cost for the workload. Note that the problem formulation allows the specification of a set of constraints that the enumeration step must respect, (e.g., to respect a storage bound). Since the index selection problem has been shown to be NP-Hard [9], the focus of most work has been on developing heuristic solutions that give good quality recommendations and can scale well.

One important observation is that solutions that naively stage the selection of different physical design structures (e.g., select indexes first followed by materialized views, select partitioning for table first followed by indexes etc.) can result in poor recommendations. This is because: (i) The choices of these structures interact with one another (e.g., optimal choice of index can depend on how the table is partitioned and vice versa). (ii) Staged solutions can lead to redundant recommendations. (iii) It is not easy to determine a priori how to partition the storage bound across different kinds of physical design structures. Thus, there is a need for integrated recommendations that search the combined space in a scalable manner.

Broadly the search strategies explored thus far can be categorized as bottom-up (e.g., [3]) or top-down [1] search, each of which has different merits. The bottom up strategy begins with the empty (or pre-existing configuration) and adds structures in a greedy manner. This approach can be efficient when available storage is low, since the best configuration is likely to consist of only a few structures. In contrast, the top-down approach begins with a globally optimal configuration but it could be infeasible if it exceeds the storage bound. The search strategy then progressively refines the configuration until it meets the storage constraints. The top-down strategy has several key desirable properties and this strategy can be efficient in cases where the storage bound is large.

Future Directions

The paradigm of combinatorial search has been effectively used in database tuning problems such as query optimization and physical database design. It is future research to consider if this paradigm can also be effectively applied to other database tuning problems such as capacity planning and optimizing database layout.

Cross-references

- ▶ Administration Wizards
- ▶ Index Tuning
- ▶ Physical Layer Tuning
- ▶ Self-Management Technology in Databases

Recommended Reading

1. Bruno N. and Chaudhuri S. Automatic physical design tuning: a relaxation based approach. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
2. Chaudhuri S. An overview of query optimization in relational systems. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 1998.
3. Chaudhuri S. and Narasayya V. An efficient cost driven index selection tool for microsoft SQL server. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997.
4. Chaudhuri S. and Narasayya V. AutoAdmin "What-If" index analysis utility. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998.
5. Chaudhuri S. and Narasayya V. Self-tuning database systems: a decade of progress. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
6. Graefe G. The Cascades framework for query optimization. Data Eng. Bull., 18(3), 1995.
7. Haas L., Freytag C., Lohman G., and Pirahesh H. Extensible query processing in Starburst. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989.
8. Papadimitriou C.H. and Steiglitz K. Combinatorial optimization: algorithms and complexity. Dover, 1998.
9. Piatetsky-Shapiro G. The optimal selection of secondary indices is NP-complete. ACM SIGMOD Rec., 13(2):72–75, 1983.

Database Tuning using Online Algorithms

Nicolas Bruno[1], Surajit Chaudhuri[1],
Gerhard Weikum[2]
[1]Microsoft Corporation, Redmond, WA, USA
[2]Max-Planck Institute for Informatics, Saarbrücken, Germany

Definition

A self-managing database system needs to gracefully handle variations in input workloads by adapting its internal structures and representation to changes in the environment. One approach to cope with evolving workloads is to periodically obtain the best possible configuration for a hypothetical "average" scenario. Unfortunately, this approach might be arbitrarily suboptimal for instances that lie outside the previously determined average case. An alternative approach is to require the database system to continuously tune its internal parameters in response to changes in the workload. This is the *online tuning paradigm*. Although solutions for different problems share the same underlying philosophy, the specific details are usually domain-specific. In the context of database systems, online tuning has been successfully applied to issues such as buffer pool management, statistics construction and maintenance, and physical design.

Historical Background

Database applications usually exhibit varying workload characteristics over time. Moreover, changes in workloads cannot be easily modeled beforehand. As a consequence, database systems traditionally provided *offline* tools to make corrections to the system's current configuration. Examples include physical design tools that take a representative workload and return a new database design that would be beneficial in the future, or the possibility of refreshing the statistical information about values stored in database columns. These approaches allow a database administrator to react to environmental changes but only after they had happened (and potentially disrupted a previously well-tuned system).

Although offline tuning tools have been successfully introduced in several commercial database systems, there has been a growing need for additional functionality that is outside the scope of such tools. As database applications increase in size and complexity, it becomes more difficult to even decide when offline tools should be called. Additionally, offline tools are sometimes resource intensive and assume that there are idle periods of time on which they can be applied, which is not necessarily the case. To cope with these new requirements, a new set of algorithms emerged (e.g., [1,2,6,8,11,12]), which are based on a different principle. The idea is to monitor the database system as queries are processed, and in the background reorganize its internal state to cope with changes in the workload. In this way, the database system is continuously modifying itself in a closed "monitor-diagnose-tune" loop.

Foundations

A requirement towards understanding online tuning is to conceptualize the transition from scenarios in which full information is known about the system in consideration (and can therefore identify the optimal solution). Within the online tuning paradigm, only partial information is known as time passes (and therefore it is necessary to approximate optimal solutions at all times without knowing the future). For illustration purposes, some specific examples of online tuning are reviewed briefly, and one such example is provided in more detail.

An example of online tuning is that of automatically managing memory allocation across different memory pools [2,12]. Complex queries generally use memory-intensive operators (e.g., hash-joins) whose performance depends on the amount of memory that is given to each operator at runtime. However, only a finite amount of memory is available at any time, and it has to be distributed among all the competing operators. This problem is further complicated by the fact that new queries are continually being served by the database system, and therefore any static policy to distribute available memory might be inadequate as workloads change. Reference [7] presents an online algorithm to manage memory that is based on the ability of operators to dynamically grow and shrink their own memory pools. By using a feedback loop while statements are being executed, this technique is able to incrementally model the optimal resource allocation and adapt the distribution of memory to the current operators to maximize performance.

Another example of online tuning is physical database design, which is concerned with identifying the best set of redundant access paths (e.g., indexes or materialized views) that would accelerate workloads [5]. While there has been work on offline tools that assume that the representative input workload would repeat indefinitely in the future, many scenarios exhibit unpredictable changes in workload patterns. Online physical design tuning takes a different approach: it monitors incoming queries and measures the relative cost/benefit of each of the present and hypothetical (i.e., not part of the current design) structures. By carefully measuring the impact of creating and dropping structures (e.g., indexes), the system is able to change the underlying database configuration in response to changes in workload characteristics [3].

Expanded Example: Self Tuning Histograms

Consider, as an in-depth example, the problem of statistics management in database systems. Relational query optimization has traditionally relied on single- or multi-column histograms to model the distribution of values in table columns. Ideally, histogram buckets should enclose regions of the data domain with approximately uniform tuple density (i.e., roughly the same number of tuples per unit of space in a bucket), to accurately estimate the result size of range queries. At the same time, histograms (especially multi-column ones) should be sufficiently compact and efficiently computable. Typically, histogram construction strategies inspect the data sets that they characterize without considering how the histograms will be used (i.e., there is an offline algorithm that builds a given histogram, possibly as a result of bad performance of some workload query). The implicit assumption while building such histograms is that all queries are equally likely. This assumption, however, is rarely true in practice, and certain data regions might be much more heavily queried than others. By analyzing workload queries and their results, one could detect buckets that do not have uniform density and "split" them into smaller and more accurate buckets, or realize that some adjacent buckets are too similar and "merge" them, thus recuperating space for more critical regions.

In other words, rather than computing a given histogram at once, without knowledge of how it is going to be used, one can instead incrementally refine the statistical model based on workload information and query feedback. This is the essence of *self-tuning histograms*. Intuitively, self-tuning histograms exploit query workload to zoom in and spend more resources in heavily accessed areas, thus allowing some inaccuracy in the rest. These histograms also exploit query feedback as truly multidimensional information to identify promising areas to enclose in histogram buckets. As a result, the resulting histograms are more accurate for the expected workload than traditional workload-independent ones. Self-tuning histograms can also gracefully adapt to changes in the data distribution they approximate, without the need to periodically rebuild them from scratch.

Reference [1] presents STGrid, the first multidimensional histogram that uses query feedback to refine buckets. STGrid histograms greedily partition the data domain into disjoint buckets that form a grid, and refine their frequencies using query feedback by adjusting the expected cardinality of buckets based on observed values. After a predetermined number of queries have been executed, the histogram is restructured by merging and splitting rows of buckets at a time. Efficiency in histogram tuning is the main goal of this technique, at the expense of accuracy. Since STGrid histograms need to maintain the grid structure at all times, and also due to the greedy nature of the technique, some locally beneficial splits and merges have the side effect of modifying distant and unrelated regions, hence decreasing the overall accuracy of the resulting histograms.

To alleviate this problem, STHoles histograms, introduced in [4], are based on a novel partitioning strategy that is especially well suited to exploit workload information. STHoles histograms allow inclusion relationships among buckets, i.e., some buckets can be completely included inside others. Specifically, each bucket in an *STHoles* histogram identifies a rectangular range in the data domain, similar to other histogram techniques. However, unlike traditional histograms, *STHoles* histograms identify bucket sub-regions with different tuple density and "pull" them out from the corresponding buckets. Hence a bucket can have *holes*, which are themselves first-class histogram buckets. In this way, these histograms implicitly relax the requirement of rectangular regions while keeping rectangular bucket structures. By allowing bucket nesting, the resulting histograms do not suffer from the problems of STGrid histograms and can model complex shapes (not restricted to rectangles anymore); by restricting the way in which buckets may overlap, the resulting histograms can be efficiently manipulated and updated incrementally by using workload information.

STHoles histograms exploit query feedback in a truly multidimensional way to improve the quality of the resulting representation. Initially, an STHoles histogram consists of a single bucket that covers the whole data domain. For each incoming query from the workload, the query optimizer consults existing histograms and produces a query execution plan. The resulting plan is then passed to the execution engine, where it is processed. A build/refine histogram module intercepts the stream of tuples that are returned, and tunes the relevant histogram buckets so that the resulting histogram becomes more accurate for similar queries. Specifically, to refine an *STHoles* histogram, one first intercepts the stream of results from the corresponding query execution plan and counts how many tuples lie inside each histogram bucket. Next, one determines which regions in the data domain can benefit from using this new information, and refines the histogram by "drilling holes," or zooming into the buckets that cover the region identified by the query plan. Finally, to adhere to the budget constraint, it is possible to consolidate the resulting histogram by merging similar buckets.

Recently, [10] introduces ISOMER (Improved Statistics and Optimization by Maximum-Entropy Refinement), a new algorithm for feedback-driven histogram construction. ISOMER uses the same partitioning strategy as STHoles, but it is based on a more efficient algorithm to restructure the histogram, which does not require counting the number of tuples that lie within each histogram bucket. In contrast, ISOMER uses the information-theoretic principle of maximum entropy to approximate the true data distribution by a histogram distribution that is as "simple" as possible while being consistent with all the previously observed cardinalities. In this manner, ISOMER avoids incorporating extraneous (and potentially erroneous) assumptions into the histogram. ISOMER's approach amounts to imposing uniformity assumptions (as made by traditional optimizers) when, and only when no other statistical information is available. ISOMER can be seen as combining the efficient refinement characteristics from STGrid histograms with the accuracy of STHoles histograms. A related piece of work is [9], which addresses the problem of combining complementary selectivity estimations from multiple sources (which themselves can be computed using ISOMER histograms) to obtain a consistent selectivity estimation using the idea of maximum entropy. Similar to the approach in ISOMER, this work exploits all available information and avoids biasing the optimizer towards plans for which the least information is known.

Future Directions

Online tuning is dramatically gaining importance as more and more applications exhibit unpredictable workload evolution. In particular, this is a requirement for cloud-based data services. In such scenarios, the backend data services are shared by multiple applications and must be able to cope with changing access patterns. It should be noted that feedback-driven control is another paradigm that has been applied for continuous incremental tuning of systems, e.g., in the context of automated memory management [7]. A detailed discussion of feedback-driven control and its applications to computing systems can be found in [8].

Cross-references

► Histogram
► Self-Management Technology in Databases

Recommended Reading

1. Aboulnaga A. and Chaudhuri S. Self-tuning histograms: building histograms without looking at data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
2. Brown K.P., Mehta M., Carey M.J., and Livny M. Towards automated performance tuning for complex workloads. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 72–84.
3. Bruno N. and Chaudhuri S. An online approach to physical design tuning. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
4. Bruno N., Chaudhuri S., and Gravano L. STHoles: a multidimensional workload-aware histogram. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001.
5. Chaudhuri S. and Narasayya V.R. Self-tuning database systems: a decade of progress. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
6. Chen C.-M. and Roussopoulos N. Adaptive selectivity estimation using query feedback. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 161–172.
7. Dageville B. and Zait M. SQL memory management in Oracle9i. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
8. Diao Y., Hellerstein J.L., Parekh S.S., Griffith R., Kaiser G.E., and Phung D.B. Self-managing systems: a control theory foundation. In Proc. 12th IEEE Int. Conf. Engineering of Computer-Based Systems, 2005, pp. 441–448.
9. Markl V., Haas P.J., Kutsch M., Megiddo N., Srivastava U., and Tran T.M., Consistent selectivity estimation via maximum entropy. VLDB J., 16(1):55–76, 2007.
10. Srivastava U. et al. ISOMER: consistent histogram construction using query feedback. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
11. Stillger M., Lohman G.M., Markl V., and Kandil M. LEO - DB2's LEarning Optimizer. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 19–28.
12. Weikum G., König A.C., Kraiss A., and Sinnwell M. Towards self-tuning memory management for data servers. IEEE Data Eng. Bull., 22(2):3–11, 1999.

Database Tuning using Trade-off Elimination

Surajit Chaudhuri[1], Gerhard Weikum[2]
[1]Microsoft Corporation, Redmond, WA, USA
[2]Max-Planck Institute for Informatics, Saarbrücken, Germany

Definition

Database systems need to be prepared to cope with trade-offs arising from different kinds of workloads that different deployments of the same system need to support. To this end, systems offer tuning parameters that allow experienced system administrators to tune the system to the workload characteristics of the application(s) at hand. As part of the *self-management* capabilities of a database system, it is desirable to eliminate these tuning parameters and rather provide an algorithm for parameter settings such that near-optimal performance is achieved across a very wide range of workload properties. This is the trade-off elimination paradigm. The nature of the solution for trade-off elimination depends on specific tuning problems; its principal feasibility has been successfully demonstrated on issues such as file striping and cache management.

Historical Background

To cope with applications that exhibit a wide *variety of workload characteristics*, database systems have traditionally provided a repertoire of alternative algorithms for the same or overlapping functionalities, and have also exposed a rich suite of quantitative *tuning parameters.* Examples include a choice of page sizes and a choice of striping units at the underlying storage level, a choice of different data structures for single-dimensional indexing, and various tuning options for cache management. This approach prepares the system for *trade-offs* that arise across different workloads and within mixed workloads.

More than a decade ago, exposing many options and tuning parameters to system administrators had been widely accepted, but has eventually led to prohibitive costs for skilled staff. In the last 10 years, many trade-offs have become much better understood and analytically or experimentally characterized in a systematic manner. In some cases, the analysis led to the insight that a specific criteria for parameter settings could yield satisfactory performance across a wide range of workloads. Typically, this insight was derived hand in hand with progress on the underlying algorithmics or hardware technology. This can form the basis for eliminating various tuning options such as page sizes, striping units, data structures for single-dimensional indexing, and cache management strategies. Some of these research results have been adopted by commercial engines for self-management; in other cases, tuning parameters are still exposed.

Foundations

The first step towards trade-off elimination is to better understand the nature of the trade-off. The key questions to consider and analyze are the following: Why are different alternatives needed for the given parameter or function? Is the variance across (real or conceivable) workloads really so high that tuning options are justified? Do different options lead to major differences in performance? Do some options result in very poor performance? Are there any options that lead to acceptably good performance across a wide spectrum of workloads?

For illustration consider the following specific tuning issues:

1. *Page sizes:* There is a trade-off between *disk I/O efficiency* and *efficient use of memory* [4]. Larger page sizes lead to higher disk throughput because larger sequential transfers amortize the initial disk-arm seeks. Smaller page sizes can potentially make better use of memory because they contain exactly the actual requested data and there are more of such small pages that fit into memory. The impact of this trade-off was large more than 10 years ago with much smaller memory sizes. Today, memory sizes are usually at comfortable level, and disk controllers always fetch entire tracks anyway. Thus, a page size of one disk track (e.g., 100 Kilobytes) is almost always a good choice, and neither hurts disk efficiency nor memory usage too much.
2. *Striping units:* When files or tablespaces are partitioned by block or byte ranges and striped (i.e., partitions being allocated in round-robin manner) across multiple disks, the partition size, aka. striping unit, leads to a trade-off regarding I/O parallelism versus disk throughput [2,9]. Small striping units lead to high parallelism when reading large amounts of data from the disk-array, but they consume much more disk-arm time for all involved disks together, compared to larger striping units with lower degree of parallelism. Thus, larger striping units can achieve higher I/O throughput. In some applications, it may still be worthwhile to tune striping units of different files according to their request size distributions. But in database systems, there is usually only a mix of single-block random accesses and sequential scans. For such workloads, large striping units in the order of one Megabyte achieve near-optimal throughput while still allowing I/O parallelism for scans.
3. *Single-dimensional indexing:* Many commercial systems offer the (human or automated) tuning wizard a choice between a B+-tree or a hash index, for each single-attribute index that is to be created. B+-trees provide high efficiency for both random key lookups and sequential scans, and have proven their high versatility. Hash indexes offer randomization to counter access skew (with some keys being looked up much more frequently than others) and even better worst-case performance for lookups. Hashing guarantees exactly one page access, whereas B+-tree indexes have to access a logarithmic number of pages in their descent towards the requested key. This was an important trade-off to consider for tuning a decade ago. But today, the disadvantages of B+-trees for certain workloads have become minor: randomization can be achieved as well by using hash-transformed keys in the tree (at the inherent expense of penalizing range queries); and the extra costs of the tree descent

are negligible given the high fan-out and resulting low depth of index trees and the fact that all index levels other than leaf pages virtually always reside in memory. So if one were to build a lean, largely self-managing database engine today, it would have to support only B+-tree indexes (but, of course, have hash-based algorithms in its repertoire for joins and grouping).

These examples demonstrate the kinds of insights and simplifications towards self-managing systems that may be achieved by means of trade-off analysis. The analyses may be based on mathematical modeling (e.g., for estimating response times and throughput of different striping units), draw from comprehensive experimentation or simulation, or assess strengths and weaknesses qualitatively (e.g., functionality and computational costs of index implementations). Often a combination of different methodologies is needed. None of these approaches can be automated; while automatic tool support is standard (e.g., for evaluating analytic models and for sensitivity analyses) the final assessment requires human analytics and judgment. Thus, the trade-off elimination paradigm is a "thinking tool" for system-design time, providing guidance towards self-tuning systems.

Example: Cache Management with Trade-off Elimination

As a more detailed example, consider the management of a *shared page cache* in a database system and the tuning issue that underlies the page replacement strategy. It is discussed in more depth here, as it is not only another illustration of eliminating trade-offs, but also an important performance issue by itself.

For choosing a cache replacement victim, cache managers assess the "*worthiness*" of the currently cache-resident pages and drop the least worthy page. Ideally, one would predict the future access patterns: the least worthy page is the one whose next access is farthest in the future [3]. But the cache manager only knows the past accesses, and can remember only a bounded amount of information about the previous patterns. In this regard, a page shows evidence of being potentially worthy if it exhibits a history frequent accesses or recent accesses. The traditional replacement strategies give priority to either *frequency* or *recency*, but neither is optimal and the co-existence of the two criteria presents a trade-off for the cache manager.

Frequency-based worthiness would be optimal if access patterns were in steady state, i.e., if page-access frequencies had a time-invariant distribution with merely stochastic fluctuation. The algorithm of choice would then be *LFU*, which always replaces the least-frequently-used page. However, if the workload evolves and the distributions of access frequencies undergo significant changes, LFU is bound to perform poorly as it takes a long time to adjust to new load characteristics and re-estimate page-access probabilities. Therefore, the practically prevalent method is actually *LRU*, which always replaces the least-recently-used page. The LRU strategy automatically adapts itself to evolving access patterns and is thus more robust than LFU.

Despite its wide use and salient properties, LRU shows significantly sub-optimal behavior under various workloads. One issue is the co-existence of *random accesses* and *sequential scans*. Pages that are accessed only once during a scan will be considered by LRU although they may never be referenced again in the near future. This idiosyncrasy is typically fixed in industrial-strength database system by allowing the query processor to pass hints to its cache manager and advise it to give low priority to such a read-once page. However, there are further situations where the specifics of the access patterns reveal shortcomings of the LRU strategy.

Consider a workload with random accesses only, but with very high variance of the individual pages' access probabilities. Assume that there is a sequence of primary-key lookups on a database table that results in alternating accesses to uniformly selected index and data pages. As there are usually many more data pages than index pages (for the same table), the individual index pages have much higher access frequencies than each of the data pages. This indicates that index pages are more worthy for staying in the cache, but LRU considers only recency and is inherently unable to discriminate the two kinds of pages. In steady state, LRU would keep the same numbers of index pages and data pages in the cache, but the optimal behavior would prioritize index pages. Ironically, LFU would be optimal for this situation (but fails in others).

The *LRFU* algorithm [6] has addressed this issue by defining the worthiness of a page as a linear combination of its access recency and access frequency. Unfortunately, the performance of this method highly depends on proper tuning of the weighting coefficients for the two aspects and on additional parameters that govern the aging of frequency estimates. Another sophisticated approach that at least one commercial system had taken is to support multiple caches with configurable sizes and allow the administrator to assign different tablespaces to different caches. This way, the cache management for data vs. index pages or pages of different tables and indexes can be tuned extremely well. But this requires very good human skills; automating this approach would be a major challenge for databases with hundreds of tables and highly diverse workloads.

An approach that aims at eliminating all such tuning parameters is the *LRU-k algorithm* [8]. Its main principle is to dynamically estimate the access frequency of "interesting" pages by tracking the time points of the last k accesses to each page. It can be viewed as a maximum-likelihood estimator for page-access probabilities, and uses a sliding-window technique for built-in aging of the estimates. Worthiness of a page is now defined as the reciprocal of the backward distance to its kth last access, and this has been shown to be optimal among all replacement algorithms that have the same information about the access history. For $k = 1$, LRU-k behaves exactly like LRU; for $k \geq 2$, it has enough information to properly handle sequentially scanned read-once pages and to automatically discriminate page classes with very different access frequencies and patterns.

LRU-k introduces additional bookkeeping that incurs overhead compared to the best, extremely lightweight, implementations of standard LRU. In particular, it needs to track the last k accesses of more pages than the cache currently holds. There is a robust heuristics, based on the "5-min rule of thumb" [4], for deciding which pages should be tracked at all. But the bookkeeping uses a non-negligible amount of memory – the very resource whose usage the cache manager aims to optimize with so much scrutiny. Thus, LRU-k invests some memory and effectively reduces the size of the cache by that amount, in order to improve the overall caching benefit. It has been experimentally shown that this *cost/benefit ratio* is indeed profitable: even if the cache size is reduced by the bookkeeping memory, LRU-k (for $k \geq 2$) still significantly outperforms LRU (but one would typically limit k to 2 or 3).

LRU-k also needs more CPU time than LRU because its replacement decision criterion, the backward distance to the k^{th} last accesses of the currently cache-resident pages, requires a priority queue rather than a simple linked list. Even with the best possible implementation techniques, this leads to logarithmic rather than constant cost per page access. However, there are excellent ways of implementing approximate versions of the LRU-k principle without this overhead. The *2Q algorithm* [5] and the *ARC algorithm* [7] use LRU-like linked lists but separate pages in two lists: one for pages with at least k accesses and one for pages with less accesses in the current bookkeeping. By devising smart rules for migrating pages between lists, these algorithms achieve cache hit ratios that are as good as LRU-k while keeping the implementation overhead as low as that of LRU. Other extensions of the LRU-k principle are cache replacement algorithms for variable-size data items such as files or Web pages [1,10].

The best algorithms of the LRU-k family have successfully eliminated the recency-frequency trade-off and provide self-tuning cache management. The insights from this line of research provide several, more general or potentially generalizable, lessons:

- It is crucial to analyze the nature of the trade-off that led to the introduction of tuning options or alternative algorithms for the same function.
- It is beneficial to unify the modeling and treatment of different classes of access patterns (workloads), thus providing a basis for simplifying algorithms and systems.
- Additional bookkeeping to better capture the workload can be worthwhile even if it consumes the very same resource that is to be optimized. But the overhead needs to be carefully limited.
- To eliminate a "troublesome" tuning option, self-managing algorithms may introduce additional second-order parameters (e.g., for bookkeeping data structures). The art is to ensure that the new parameters must be such that it should be easy to find a robust setting that leads to near-optimal behavior under almost all workloads.

Future Directions

Trade-off elimination is a general paradigm, but not a directly applicable recipe for self-management. Thus, future research should consider studying more trade-offs and tuning problems in view of this paradigm, bearing in mind both its potential benefits and intricacies.

Cross-references

- ► Memory Hierarchy
- ► Self-Management Technology in Databases

Recommended Reading

1. Cao P. and Irani S. Cost-aware WWW proxy caching algorithms. In Proc. 1st USENIX Symp. on Internet Tech. and Syst., 1997.
2. Chen P.M., Lee E.L., Gibson G.A., Katz R.H., and Patterson D.A. RAID: high-performance, reliable secondary storage. ACM Comput. Surv., 26(2):145–185, 1994.
3. Coffman E.G. Jr. and Denning P.J. Operating Systems Theory. Prentice-Hall, Englewood, Cliffs, NJ, 1973.
4. Gray J. and Graefe G. The five-minute rule ten years later, and other computer storage rules of thumb. ACM SIGMOD Rec., 26 (4):63–68, 1997.
5. Johnson T. and Shasha D. 2Q: a low overhead high performance buffer management replacement algorithm. In Proc. 20th Int. Conf. on Very Large Data Bases. 1994, pp. 439–450.
6. Lee D., Choi J., Kim J.-H., Noh S.H., Min S.L., Cho Y., and Kim C.-S. LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Trans. Comput., 50(12):1352–1361, 2001.
7. Megiddo N. and Modha D.S. Outperforming LRU with an adaptive replacement cache algorithm. IEEE Comput., 37 (4):58–65, 2004.
8. O'Neil E.J., O'Neil P.E., and Weikum G. The LRU-K page replacement algorithm for database disk buffering. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 297–306.
9. Scheuermann P., Weikum G., and Zabback P. Data partitioning and load balancing in parallel disk systems. VLDB J., 7(1):48–66, 1998.
10. Young N.E. On-line file caching. Algorithmica, 33(3):371–383, 2002.

Database Use in Science Applications

Amarnath Gupta
University of California-San Diego, La Jolla, CA, USA

Definition

A science application is any application where a natural, social or engineering problem is investigated.

The Problem

Many science applications are data intensive. Scientific experiments produce large volumes of complex data, and have a dire need to create persistent repositories for their data and knowledge. It would seem natural that data management systems and technology will be heavily used in science. And yet, scientists traditionally do not use database management systems, and often develop home-grown solutions, or file-based software for their complex data management needs. Clearly, there is a gap between scientists' intended use of data and what current data management systems provide.

Foundations

There are many reasons, both technical and non-technical, that explain why science users do not use data management systems for their applications. A recent study [3] highlights a number of factors scientists have cited. Others [2,4,6] have analyzed different reasons why scientific studies are not effectively supported by today's database management systems. Some of the factors contributing to the problem are:

1. *Lack of Support for Scientific Data Types:* Scientists use a wide variety of data types – sequences, graphs, images, surface meshes, time-varying fields, etc., that traditional data management systems have little support for. The standard advantages of databases, including set-at-a-time processing, transaction management, automatic query optimization, efficient storage management, etc., do no work well for these data types. Although there has been significant research investigating data management issues for some of these data types (e.g., sequences), most commercial DBMS vendors and many of the open source DBMS providers do not supply adequate, out-of-the-box support for them. An important shortcoming is the lack of adequate support for time-varying versions of any data, ranging from time-series of scalar values to evolving geometric objects or changing graphs. Though one can define new data types in current data management systems, it does not help the science community, where an average user is not technically advanced enough to use these capabilities. Very often, the type extensibility provides limited flexibility because the user can define the structure and methods for a new type, but cannot associate storage management for the structure or cost-functions for the operators without going into significant high-expertise programming efforts.
2. *Lack of Support for Scientific Queries*: While business queries demonstrate a wide range of complexity, scientific queries are very often quite different and unconventional. Common computations performed by scientists in different disciplines include nearest-neighbor queries in high dimensions, dimensionality reduction and coordinate transformation operations, computations of cliques or

strong components in large graphs, eigenvalue computations on large matrices, statistical model fitting and tests of significance, all-by-all similarity computation with user-defined similarity functions, etc. In addition, specific domains require their own specialized computation. To perform these computations efficiently, the data management systems need to have special data and index structures. Again, despite academic research in these areas, most database systems do not have the ability to perform such computations. Computationally, many domains require solving differential equations, or fine-element analysis, and other forms of "modeling and simulation" support that a DBMS is not designed to provide. Consequently, scientists use different specialty software (like MATLAB) to handle their computational needs and stay within the data handling capabilities of these systems. Scientists who need to manage large data volumes and perform these complex computations typically end up writing their own custom software that fits their research needs. These software modules are usually not designed to work on top of a DBMS, and almost never designed to work from inside a DBMS.

3. *Lack of Support for Data Visualization*: In many scientific disciplines (ranging from molecular science to biodiversity modeling), visualization of data is a very important aspect of the scientific process. Scientists want to visualize data for different reasons – to roughly check for its correctness before storing it persistently, to inspect the results of a query or analysis, as a way to explore sampled contents of a data store before deciding if this is the appropriate store to query or analyze. Since scientific data is not all relational, supporting data visualization implies that the system needs to have a visualization scheme for each type and allow the users to select the type/desire of visualization they [1]. For example, if the data consists of time series of temperature and pressure over a quarter degree grid on the Pacific Ocean, it is not trivial to visualize the result of even the simplest of selection queries. In these situations, it is often easier for scientists to use domain-specific visualization (e.g., volume visualization) software that operates on the raw data and allows the user to manipulate "knobs" like the coloring scheme, the shape and sizes of icons etc. to make the output visually informative.
4. *Formulation of Queries and Analysis*: One of the key features of databases is the ability to form declarative queries that can be evaluated optimally. A large fraction of scientific data manipulation and retrieval has complex procedural logic. Although every DBMS comes with a database programming language, scientists often find it easier to use their favorite programming language and software API to implement their analysis. It has been observed that data flow systems such as Kepler [5] or Scitegic (http://www.scitegic.com/), where the user can define a graph of functions strung together with a visual interface, is a more popular tool for formulating complex analysis tasks. Lack of suitable query formulation tools that may guide scientists through the steps of formulating correct queries prevents many science users from using powerful data manipulation and query capabilities offered by a DBMS.
5. *Data Cleaning and Curation*: Scientific data often have a lot of "noise," i.e., spurious and erroneous values. The noise occurs not only because of data value entry errors but because of measurement errors, computational errors, data obsolescence, definitional errors, and so on. In addition, due to the way information is generated or acquired, data can often be "fragmented" where data elements that are supposed to belong in a single collection are in multiple locations without enough information to coherently reassemble them. Scientists often write custom tools to clean and combine the data in a way that suits their individual needs because a typical DBMS will not provide the correct tools for cleaning this kind of data, nor would it provide enough hooks so that user-defined data cleaning and curation tools can be assembled into the tool suite of an existing DBMS.
6. *Schema and Application Evolution*: The scientific enterprise works differently from a commercial enterprise. Science is often study-driven. A scientist wants to understand a natural phenomenon, or test a hypothesis, and therefore designs experiments, collects data, integrates information with collaborators' and other publicly available data, performs the analysis all geared toward the study being undertaken. As scientific questions change, study designs and all other data content and analysis requirements change, and thus a new setting is developed. Even within a study, as the investigation proceeds, there is a need to acquire new data

(or discard previously acquired data) and perform new analysis. This situation requires a lot more agility from the information system. One aspect of the agility is that the schema that was designed for a study at one point of time is very likely change in the course of the experiment. Despite significant work in schema evolution and database versioning in the research world, a standard DBMS today is not very well geared toward schema evolution. Scientists find their home-grown file based data management systems are much more amenable to their changing application requirements.

7. *Poor Provenance and Annotation Support*: The practice of science involves a significant degree of data transformation and reuse. Scientist A will use a portion of the data collected by him and another portion of the data collected by Scientist B, after changing the units of the values and shifting the origin of the data to a more appropriate coordinate system. Two years later, this combined data will be transformed in some other way and be used for a new analysis. This practice makes it important to record and track the origin and transformations performed on data as it is split, merged, changed and copied through its life cycle – an effort that is collectively known as provenance management. Although there is much recent research on provenance issues, a systematic treatment of provenance is currently not provided by today's DBMS software. A related capability, also not provided by current data management systems, is the ability to annotate data, whereby a user "marks" an existing piece of data and notes a remark on it. This remark is as important as the data itself and possibly should be carried along as the data is transformed – in a fashion similar to the propagation of provenance data. To be truly useful for scientists, the task of annotation and provenance management should be uniformly managed over any scientific data type – a capability that scientist need but DBMS's do not provide.

There are many more factors that contribute to the lack of use of databases by science applications. Non-technical reasons (purchase and maintenance cost, steep learning curve, expensive personnel cost) are equally responsible as the technical shortcomings of data management systems.

Since both the database research community and the vendor community are aware of these issues, there is now a recent trend among data management research and practices to address them directly. As a result, new modules are being added to DBMS to target specific science markets. In particular, the life science market is becoming a dominant force to demand new features and capabilities of future data management systems. The BLAST search used by molecular biologists is now "wrapped" into an IBM product for data integration. It is expected that as scientists move from science-in-the-small to large-scale science, scientists will understand and accept the value of data management systems. At the same time, as the scientific data management market becomes stronger and more demanding, commercial and open-source data management products will become more capable of handling scientific data and analyses.

Key Applications

Bioinformatics, cheminformatics, engineering databases.

Cross-references

► Data Integration Architectures and Methodology for the Life Sciences

► Data Types in Scientific Data Management

► Provenance

Recommended Reading

1. Altintas I., Berkley C., Jaeger E., Jones M., Ludäscher B., and Mock S. Kepler: an extensible system for design and execution of scientific workflows. In Proc. 16th Int. Conf. Scientific and Statistical Database Management, 2004, pp. 423–424.
2. Buneman P. Why Scientists Don't Use Databases? NeSC presentation, 2002. Available from www.nesc.ac.uk/talks/opening/no_use.pdf.
3. Gray J., Liu D.T., Nieto-Santisteban M.A., Szalay A.S., Heber G., and DeWitt D. Scientific data management in the coming decade. ACM SIGMOD Rec., 34(4): 35–41, 2005.
4. Liebman M.J. Data management systems: science versus technology? OMICS J. Integr. Biol., 7(1):67–69, 2003.
5. Livny M., Ramakrishnan R., Beyer K., Chen G., Donjerkovic D., Lawande S., Myllymaki J., and Wenger K. DEVise: Integrated Querying and Visual Exploration of Large Datasets. In Proc. 2007 ACM SIGMOD Int. Conf. on Management of Data. Tucson, AZ, 1997, pp. 301–312.
6. Maier D. Will database systems fail bioinformatics, too? OMICS J. Integr. Biol., 7(1):71–73, 2003.

Databases for Biomedical Images

► Image Management for Biological Data

Dataguide

► Structure Indexing

Datalog

Grigoris Karvounarakis
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

Deductive databases

Definition

An important limitation of relational calculus/algebra is that it cannot express queries involving "paths" through an instance, such as taking the transitive closure over a binary relation. *Datalog* extends conjunctive queries with recursion to support such queries. A Datalog program consists of a set of rules, each of which is a conjunctive query. Recursion is introduced by allowing the same relational symbols in both the heads and the bodies of the rules. A surprising and elegant property of Datalog is that there are three very different but equivalent approaches to define its semantics, namely the *model-theoretic*, *proof-theoretic*, and *fixpoint* approaches. Datalog inherits these properties from logic programming and its standard language Prolog. The main restriction that distinguishes Datalog from Prolog is that function symbols are not allowed.

Several techniques have been proposed for the efficient evaluation of Datalog programs. They are usually separated into two classes depending on whether they focus on *top-down* and *bottom-up* evaluation. The ones that have had the most impact are centered around *magic sets rewriting*, which involves an initial preprocessing of the Datalog program before following a bottom-up evaluation strategy. The addition of negation to Datalog rules yields highly expressive languages, but the semantics above do not extend naturally to them. For Datalog¬, i.e., Datalog with negated atoms in the body of the rules, *stratified* semantics, which impose syntactic restrictions on the use of negation and recursion, is natural and relatively easy to understand. The present account is based primarily on the material in [1]. Each of [1,2,9] has an excellent introduction to Datalog (Capitalization of the name follows the convention used in [2,8] (rather than [1,9]).). An informal survey can be found in [8]. The individual research contributions to Datalog are cited in the union of the bibliographies of these textbooks.

Historical Background

Datalog is a restriction of the paradigm of *logic programming (LP)* and its standard programming language, Prolog, to the field of databases. What makes logic programming attractive is its *declarative* nature, as opposed to the more operational flavor of other programming paradigms, be they imperative, object-oriented, or functional. In the late 1970's and into the 1980's, this led to much LP-related activity in Artificial Intelligence and even supercomputing (The Fifth Generation Project) which has later subsided dramatically. In databases this remains a useful paradigm, since the relational calculus is also a declarative language and LP provides a mechanism for extending its expressiveness with so-called *recursive queries*.

The name "Datalog" was coined by David Maier [1]. Research on recursive queries in databases picked up in the 1980's and eventually led to several prototype *deductive database systems* [8,9] whose data is organized into relations, but whose queries are based on Datalog.

Datalog has not quite made it as a practical query language due to the preeminence of SQL. When the need for recursive queries was recognized by RDBMS vendors, they preferred to extend SQL with some limited forms of recursion [8]. Nonetheless, more recent research on data integration has found Datalog to be a useful conceptual specification tool.

Foundations

Datalog Syntax

The syntax of Datalog follows that of the logic programming language Prolog with the proviso that only constants and relational symbols are allowed (no function symbols).

Definition 1

Fix a relational schema. (The use of the symbol :- has its roots in Prolog, but some texts, e.g., [1], use the symbol ←instead, to convey the fact that each rule is closely related to a logical implication, as explained in the discussion of model-theoretic semantics.) *A* Datalog rule *has the form:*

$$T(\mathbf{x}) :- \ q(\mathbf{x}, \mathbf{y})$$

where $\mathbf{x} = x_1,\ldots,x_n$ is a tuple of *distinguished* variables, $\mathbf{y} = y_1,\ldots,y_m$ is a tuple of "existentially quantified" variables, T is a relation and q is a conjuction of relational atoms. The left-hand side is called the head of the rule and corresponds to the output/result of the query and the right-hand side is called the body of the rule. Note that all distinguished variables in the head need to appear in at least one atom in the body, i.e., the rules are *range restricted*. A Datalog rule is identical to a conjunctive query in rule-based syntax, except that the latter does not always have a name for the head relation symbol. A Datalog program is a finite set of Datalog rules over the same schema. Relation symbols (a.k.a. predicates) that appear only in the body of the program's rules are called *edb* (extensional database) predicates, while those that appear in the head of some rule are called *idb* (intensional database) predicates. A Datalog program defines a Datalog query when one of the idb predicates is specified as the *output*.

For example, if G is a relation representing edges of a graph, the following Datalog program P_{TC} computes its transitive closure in the output predicate T:

$$T(x, y) \text{ :- } G(x, y)$$
$$T(x, y) \text{ :- } G(x, z), T(z, y)$$

Semantics

Three different but equivalent definitions can be given for the semantics of Datalog programs, namely the *model-theoretic*, *proof-theoretic* and *fixpoint* semantics.

A countably infinite set $\mathbb{D}$ of constants is fixed as the sole universe for structures/instances. Since there are no function symbols, any relational instance over $\mathbb{D}$ is an *Herbrand interpretation* in the sense used in logic programming.

In the model-theoretic semantics of Datalog, each rule is associated with a first-order sentence as follows. First recall that as a conjunctive query, $T(\mathbf{x})$:- q $(\mathbf{x}, \mathbf{y})$ corresponds to the first-order query $T \equiv \{\mathbf{x} \mid \exists \mathbf{y}\ q(\mathbf{x}, \mathbf{y})\}$. To this, one associates the sentence $\forall \mathbf{x}\ (\exists \mathbf{y}\ q(\mathbf{x}, \mathbf{y}) \rightarrow T(\mathbf{x}))$ which is clearly satisfied in a structure in which T is interpreted as the answer to the query. Note that this sentence is a *definite Horn clause*. More generally, given a Datalog program P, let Σ_P be the set of Horn clauses associated to the rules of P.

Let I be an input database instance, in this case an instance of the schema consisting only of edb predicates. A *model* of P is an instance of the entire schema (both edb and idb relation symbols) which coincides with I on the edb predicates and which satisfies Σ_P. However, there can be infinitely many instances that satisfy a given program and instance of the edb relations. Thus, logic programming, and consequently Datalog use a *minimal* model, i.e., one such that no subset of it is also a model. This is usually understood as a manifestation of the *closed world assumption*: don't assume more than you need! It can be shown that for Datalog, there is exactly one minimal model, which is also the *minimum* model.

In the *proof-theoretic* approach of defining the semantics of Datalog, note first that a tuple of constants in a relation can be seen as the head of a rule with empty body. Such rules are called *facts*. As previously seen, Datalog rules can be associated with first-order sentences. Facts correspond to just variable-free relational atoms. Now, the main idea of the proof-theoretic semantics is that the answer of a Datalog program consists of the set of facts that can be proven from the edb facts using the rules of the program as *proof rules*. More precisely, a *proof tree* of a fact A is a labeled tree where (i) each vertex of the tree is labeled by a fact; (ii) each leaf is labeled by a fact in the base data; (iii) the root is labeled by A; and (iv) for each internal vertex, there exists an instantiation A_1 :- $A_2,\ldots,A_n$ of a rule r such that the vertex is labeled A_1 and its children are respectively labeled $A_2,\ldots,A_n$ and the edges are labeled r.

Example 1. *Consider the program*:

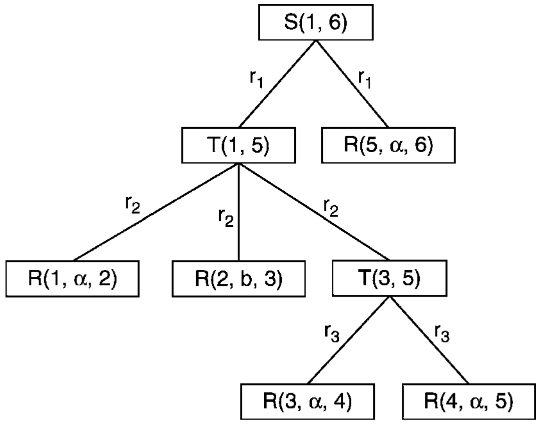
$$(r_1)\ S(x_1, x_3) \text{ :- } T\ (x_1, x_2), R(x_2, a, x_3)$$
$$(r_2)\ T(x_1, x_4) \text{ :- } R\ (x_1, a, x_2), R(x_2, b, x_3), T(x_3, x_4)$$
$$(r_3)\ T(x_1, x_4) \text{ :- } R\ (x_1, a, x_2), R(x_2, a, x_3)$$

and the instance

$$\{R(1, a, 2), R(2, b, 3), R(3, a, 4), R(4, a, 5), R(5, a, 6)\}$$

A proof tree of S(1, 6) is shown in Fig. 1.

Because rule instantiation and application correspond to standard first-order inference rules (substitution and modus ponens), the proof trees are actually rearrangements of first-order proofs. This connects Datalog, through logic programming, to automated theorem-proving. One technique for constructing proofs such as the one above in a *top-down* fashion (i.e., starting from the fact to be proven) is *SLD resolution* [1]. Alternatively, one can start from base data and apply rules on them (and subsequently on facts derived this way) to create proof trees for new facts.



Datalog. Figure 1. Proof tree.

The third approach is an operational semantics for Datalog programs stemming from *fixpoint* theory. The main idea is to use the rules of the Datalog program to define the *immediate consequence* operator, which maps idb instances to idb instances. Interestingly, the immediate consequence operator can be expressed in relational algebra, in fact, in the SPCU (no difference) fragment of the relational algebra, enriched with edb relation names. For example, the immediate consequence operator $\mathcal{F}$ for the transitive closure above is:

$$\mathcal{F}(T) = G \bowtie T \cup G$$

One way to think about this operator is that it applies rules on existing facts to get new facts according to the head of those rules. In general, for a recursive Datalog program, the same operator can be repeatedly applied on facts produced by previous applications of it. It is easy to see that the immediate consequence operator is *monotone*. Another crucial observation is that it will not introduce any constants beyond those in the edb instance or in the heads of the rules. This means that any idb instance constructed by iteration of the immediate consequence operator is over the *active domain* of the program and the edb instance. This active domain is finite, so there are only finitely many possible idb instances. They are easily seen to form a finite poset ordered by inclusion. At this point one of several technical variants of fixpoint theory can be put to work. The immediate consequence operator has a *least fixpoint* which is an idb instance and which is the semantics of the program. It can be shown that this idb instance is the same as the one in the minimal model semantics and the one in the proof tree semantics. It can also be shown that this least fixpoint can be reached after finitely many iterations of the immediate consequence operator which gives a Datalog evaluation procedure called *bottom-up*.

Evaluation and Optimization of Datalog

The simplest bottom-up evaluation strategy, also called *naive* evaluation, is based directly on fixpoint Datalog semantics. The main idea is to repeatedly apply the immediate consequence operator on results of all previous steps (starting from the base data in the first step) until some step doesn't yield any new data. It is clear that naive evaluation involves a lot of redundant computation, since every step recomputes all facts already computed in previous steps. *Seminaive* evaluation tries to overcome this deficiency, by producing at every step only facts that can be derived using at least one of the new facts produced in the last step (as opposed to all previous steps).

In some cases, bottom-up evaluation can produce a lot of "intermediate" tuples that are not used in derivations of any facts in the output relation of the query. The top-down approach avoids this problem by using heuristic techniques to focus attention on relevant facts, i.e., ones that appear in some proof tree of a query answer, especially for Datalog programs with constants appearing in some atoms. The most common approach in this direction is called the query-subquery (QSQ) framework. QSQ generalizes the SLD resolution technique, on which the proof-theoretic semantics are based, by applying it in sets, as opposed to individual tuples, as well as using constants to select only relevant tuples as early as possible. In particular, if an atom of an idb relation appears in the body of a rule with a constant for some attribute, this constant can be pushed to rules producing this idb. Similarly, "sideways information passing" is used to pass constant binding information between atoms in the body of the same rule. Such constant bindings are expressed using *adornments* or *binding patterns* on atoms in the rules, to indicate which attributes are *bound* to some constant and which are *free*.

Magic set techniques simulate the pushing of constants and selections that happens in top-down evaluation to optimize bottom-up evaluation. In particular, they rewrite the original Datalog program into a new program whose seminaive bottom-up evaluation produces the same answers as the original one, as well as producing the same intermediate tuples as the top-down approaches such as QSQ.

Datalog with Negation

The language Datalog¬ extends Datalog by allowing negated atoms in the body of the rules. Unfortunately, the semantics described above do not extend naturally to Datalog¬ programs. For example, if the fixpoint semantics are followed, there are programs that do not have a fixpoint or have multiple least fixpoints, or even if there is a least fixpoint the constructive method described above does not converge or its limit is not the least fixpoint. For model-theoretic semantics, uniqueness of the minimal model is not guaranteed. For these reasons, the common approach is to only consider a syntactically restricted use of negation in Datalog¬ programs, called *stratification*, for which natural extensions of the usual Datalog semantics do not have these problems. A stratification of Datalog¬ is a partition of its rules into subprograms that can be ordered in *strata*, so that for each relation R in the program, all rules defining R (i.e., with R in the head) are in the same stratum and for all atoms in the bodies of those rules, the definitions of those relations are in a smaller or the same stratum, if the atom is positive, or strictly in a smaller stratum, for negative atoms.

For stratified Datalog¬ programs, one can evaluate within each stratum considering atoms of relations defined in smaller strata as edbs. Then, a negated atom is satisfied in the body of the rule if the corresponding tuple does not appear in that relation (as it appears in the base data or computed for the subprograms of smaller strata).

Key Applications

Current and Potential Users and the Motivation of Studying This Area

Although Datalog was originally proposed as the foundation of deductive databases, which never succeeded in becoming part of commercial systems, it has recently seen a revival in the areas of data integration and exchange. This is due to the similarity of Datalog rules with popular *schema mapping* formalisms (*GLAV* or *tuple generating dependencies* [5]) used to describe relationships between heterogeneous schemas. In particular, [3] proposed the *inverse rules* algorithm for reformulating queries over a target schema to queries over source schemas in data integration. Other work has used Datalog rules to compute *data exchange* [7] or *update exchange* [4] solutions. In these cases, the authors employed an extension of Datalog with *Skolem* functions in the head of rules, to deal with existentially quantified variables in the target of mappings. Another extension of Datalog, *Network Datalog (NDlog)* [6] allows the declarative specification of a large variety of network protocols with a handful of lines of program code, resulting to orders of magnitude of reduction in program size.

Cross-references

- ► Conjunctive Query
- ► Relational Calculus

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases. Addison-Wesley, MA, USA, 1995.
2. Bidoit N. Bases de Données Déductives: Présentation de Datalog. Armand Colin, 1992.
3. Duschka O., Genesereth M., and Levy A. Recursive query plans for data integration. J. Logic Program., 43(1), 2000.
4. Green T.J., Karvounarakis G., Ives Z.G., and Tannen V. Update exchange with mappings and provenance. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
5. Lenzerini M. Tutorial – data integration: a theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on principles of Database Systems, 2002.
6. Loo B.T., Condie T., Garofalakis M.N., Gay D.E., Hellerstein J.M., Maniatis P., Ramakrishnan R., Roscoe T., and Stoica I. Declarative networking: language, execution and optimization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 97–108.
7. Popa L., Velegrakis Y., Miller R.J., Hernández M.A., and Fagin R. Translating web data. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
8. Ramakrishnan R. and Gehrke J. Database Management Systems, 3rd edn. McGraw-Hill, New York, 2003.
9. Ullman J.D. Principles of Database and Knowledge-Base Systems, Volume II. Addison-Wesley, MA, USA, 1989.

Datalog Query Processing and Optimization

► Query Processing in Deductive Databases

Datastream Distance

► Stream Similarity Mining

Datawarehouses Confidentiality

▶ Data Warehouse Security

DBC

▶ Database Clusters

DBMS

▶ Database Management System

DBMS Component

Johannes Gehrke
Cornell University, Ithaca, NY, USA

Synonyms

DBMS Module

Definition

A component is a self-contained piece of software in a database system. A component can be defined at different levels of coarseness. At the coarsest level, the components of a relational database management system consist of the client communications manager, the process manager, a relational query processor, a transactional storage manager, and utilities [1].

Key Points

The components of a relational database management system can be further refined into subcomponents [1]. The client communications manager consists of local client protocols and remote client protocols. The process manager consists of admission control and dispatch and scheduling. The relational query processor consists of query parsing and authorization, query rewrite, query optimization, plan execution, and DDL and utility processing. The transactional storage manager consists of access methods, a buffer manager, a lock manager, and a log manager. Sub-components that comprise the utilities component include the catalog manager, the memory manager, and replication and loading services.

This high-level division varies among commercial and open-source database products, both by level of granularity and by functionality of individual (sub-) components. Database management systems optimized for certain types of workloads (for example, decision support workloads) or database management systems with specialized architectures (for example, main-memory database management systems) may lack some of these components [2].

Cross-references

▶ Client Communications Manager
▶ Process Management
▶ Relational Query Processor
▶ Transactional Storage Manager.

Recommended Reading

1. Hellerstein J.M., Stonebraker M. and Hamilton J. Architecture of a Database System. In Foundations and Trends in Databases, 1(2):141–259, 2007.
2. Ramakrishnan R. and Gehrke J. Database Management Systems, 3rd edition. McGraw-Hill Science/Engineering/Math, 2002.

DBMS Interface

Johannes Gehrke
Cornell University, Ithaca, NY, USA

Synonyms

Communication boundary of a DBMS

Definition

A DBMS interface is the abstraction of a piece of functionality of a DBMS. It usually refers to the communication boundary between the DBMS and clients or to the abstraction provided by a component within a DBMS. A DBMS interface hides the implementation of the functionality of the component it encapsulates.

Key Points

DBMS interfaces can be external or internal [3]. The external DBMS interface is the communication boundary between the DBMS and clients. The external DBMS interface enables clients to access internal DBMS functionality without exposing the mechanisms of how this functionality is implemented. Well-known external DBMS interfaces are SQL, XPath, and XQuery. There are many different types of external DBMS interfaces, for example, stand-alone languages (such as SQL), extensions to existing languages with features from SQL (such as JDBC), and integration into middle-tier programming languages (such as PHP). The external DBMS interface can also be graphical; for example, the DBMS interface to the data data definition language is often a graphical editor enabling database designers to visualize and manipulate logical and physical database schemas in the Entity-Relationship Model or in UML.

Internal DBMS interfaces exist between different DBMS components, for example, the query processor and the storage manager [2]. Although standards for these interfaces do not exist, their design is as important as the design of the external interfaces. For example, by exposing a queue interface for internal DBMS components and arranging the DBMS components in stages, the DBMS can be designed such that queries interact with a single component at a given time, permitting enhanced processor utilization during query processing [1].

Cross-references

- ▶ DBMS Component
- ▶ Query Language
- ▶ SQL
- ▶ XPath/XQuery

Recommended Reading

1. Harizopoulos S. and Ailamaki A. StagedDB: Designing Database Servers for Modern Hardware. IEEE Data Engineering Bulletin, 28(2):11–16, June 2005.
2. Hellerstein J.M., Stonebraker M. and Hamilton J. Architecture of a Database System. In Foundations and Trends in Databases, 1(2):141–259, 2007.
3. Ramakrishnan R. and Gehrke J. Database Management Systems, 3rd edition. McGraw-Hill Science/Engineering/Math, 2002.

DBTG Data Model

▶ Network Data Model

DCE

Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA

Synonyms

Distributed computing environment

Definition

The Distributed Computing Environment (DCE) [1–3] is a technology standardized by the Open Group for client/server computing. A primary goal of DCE is interoperability using remote procedure call semantics.

Key Points

The Distributed Computing Environment (DCE) technology was developed through the efforts of the Open Software Foundation (OSF) in the late 1980s and early 1990s as an interoperable solution for client-server distributed computing. A primary objective was to overcome the heterogeneity in operating systems and network technologies. The DCE technology uses procedural programming semantics provided by languages, such as C. OSF is now part of the Open Group, which releases DCE code under the LGPL license via its DCE portal [2].

A primary feature provided by DCE is the remote procedure call [1]. DCE is, however, not only about remote procedure calls. In addition, DCE provides a number of fundamental building blocks to make applications secure, as well as provides a number of services, such as a cell directory service, a distributed file system service, a time service and a threading service.

DCE supports the notion of a cell, which is a collection of nodes that are managed by a single authority. Intra-cell communication is highly optimized and secure. Inter-cell communication requires more advanced configurations. The distributed file system service provides a high performance network file system. A POSIX-like API available locally hides the distributed aspects of the file system.

The DCE concepts and technology laid the foundation for the next generation of object-oriented client-server paradigms, such as CORBA in the mid 1990s. DCE RPC is used as the building block technology for Microsoft COM/DCOM technologies.

Cross-references

- ▶ Client-Server Architecture
- ▶ CORBA
- ▶ DCOM
- ▶ J2EE
- ▶ Java RMI
- ▶ .NET Remoting
- ▶ Request Broker
- ▶ SOAP

Recommended Reading

1. The Open Group, DCE 1.1: Remote Procedure Call, CAE Specification, Document no. C706, 1997, Published online at http://www.opengroup.org/onlinepubs/9629399.
2. The Open Group, DCE Portal, http://www.opengroup.org/dce/.
3. Software Engineering Institute, Distributed Computing Environment, http://www.sei.cmu.edu/str/descriptions/dce.html.

DCOM

Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA

Synonyms

Distributed component object model

Definition

Distributed Component Object Model (DCOM) [1,2] is a Microsoft technology for component-based distributed computing.

Key Points

Distributed Component Object Model (DCOM) is an extension of Microsoft's Component Object Model (COM) to enable distribution across address spaces and networks. By leveraging COM, which is the fundamental technology used to build many Microsoft applications, DCOM applications can derive all the power of COM applications. Distribution capabilities in DCOM are realized by leveraging DCE Remote Procedure Call mechanisms and extending them to support the notion of remote objects.

DCOM provides most of the capabilities of a Request Broker. For example, it supports location transparency, connection management, resource management, concurrency control, versioning, language-neutrality and QoS management. Since DCOM is a component model, additional capabilities for composing and deploying component-based applications also exist.

Despite its numerous benefits, DCOM has its own limitations. For example, DCOM uses its own binary protocol for communicating between applications, which restricts interoperability to communication between the same object models. Additionally, although some efforts exist at porting DCOM to other platforms, such as Linux, DCOM remains predominantly a Microsoft platform-specific technology, which limits its applicability to a wider range of applications.

These limitations have necessitated the move towards more advanced technologies, such as .NET Remoting, which provide a wider range of interoperable solutions.

Cross-references

- ▶ Client-Server Architecture
- ▶ CORBA
- ▶ DCE
- ▶ DCOM
- ▶ J2EE
- ▶ Java RMI
- ▶ .NET Remoting
- ▶ Request Broker
- ▶ SOAP

Recommended Reading

1. Brown N. and Kindel C. Distributed Component Object Model Protocol – DCOM/1.0, Internet Draft, Network Working Group, November 1996.
2. Horstmann M. and Kirtland M. DCOM Architecture, DCOM Technical Articles, Microsoft Developer Network Library, July 23, 1997.

Deadlocks in Distributed Database Systems

▶ Distributed Deadlock Management

Decay Models

Edith Cohen
AT&T Labs-Research, Florham Park, NJ, USA

Definition

Decay models are applicable on data sets where data items are associated with points in a metric space (*locations*) and there is a notion of "significance" of a data item to a location, which decays (decreases) with the distance between the item and the location. This decrease is modeled by a *decay function.*

Each location has a "view" of the data set through a different weight distribution over the items: the weight associated with each item is its *decayed weight* which is a product of its *original weight* and a decay function applied to its distance from the observing location.

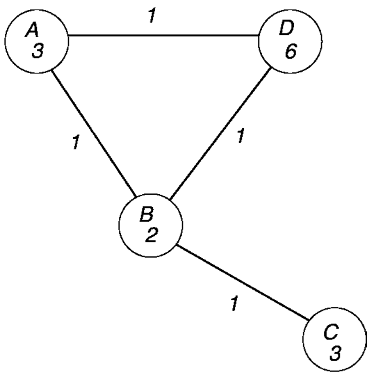
While global aggregates or statistics are computed over the set of items and their original weights, *decaying aggregates* or *decaying statistics* depend on the location with respect to which the aggregation is performed and on the decay function. Figure 1 illustrates a small network and the decaying sum with respect to all nodes.

Historical Background

Kernel Estimation

The earliest use of decay models that the author is aware of is *Kernel estimation*. Kernel estimation is a non-parametric density function estimation method [19]. The metric space is the Euclidean space and the method is applied to a set of points with weights. The decay function captures a spherically symmetric probability density (typically a Gaussian which corresponds to Exponential decay); the decaying sum at a point is the density estimate.

Recently, Hua et al. [16] proposed the use of decayed sum to assign *typicality score* to items, based on Kernel estimation. They also propose methods to efficiently compute approximate decaying sum in a high dimensional Euclidean space.



Node	Decaying sum
A	8
B	8
C	7
D	9.5

Decay Models. Figure 1. Decaying sum over a network with respect to decay function $g(d) = \frac{1}{1+d}$. Items of distance 1 have decayed weight that is 1⁄2 of their original weight. Similarly, items of distance 2 have decayed weight that is 1⁄3 of their original weight. The global (non-decaying) sum is 14. The decaying sums (listed for all nodes) are location-dependent.

Time-Decay

Time-decay, where significance of a data point decreases with elapsed time (or with number of observed items), is a natural model. Decaying aggregation on data streams predates the formal analysis of stream algorithms but was mostly limited to exponential decay. Several applications are listed next.

The random early detection (RED) protocol: RED is a popular protocol deployed in Internet routers for congestion avoidance and control. RED uses the weighted average of previous queue lengths to estimate the impending congestion at the router; the estimate is then used to determine what fraction of packets to discard [13,17]. *Holding-time policies for ATM virtual circuits:* Circuit-switched data connections have an associated cost of being kept open; data transfers are bursty and intermittent and, when a data burst arrives, it incurs smaller delay when the circuit is open. Thus, when there are multiple connections, it is important to assess the anticipated idle times (time to the next data burst) of each circuit and to close first those circuits that have longer anticipated idle times. This can be done using a time-decaying weighted average of previous idle times [18]. A similar model applies to maintaining open TCP connections at a busy Web server [7]. *Internet gateway selection products:* Multiple internet paths are available to each destination host and the product needs to assess the reliability of each path in order to facilitate a better selection of a path. A time-decaying average of previous measurements can be used as a measure of path quality [2].

Datar et al. [12] introduced the *sliding-window* model for data streams. *Sliding-window* are a threshold decay function defined over the sequence number of data items. They introduced a synopsis data structure called *Exponential Histograms* that supports approximate sum queries on sliding windows with respect to the "current" time and developed algorithms for approximately maintaining sum and average aggregates and variance [1]. Gibbons and Tirthapura [14] developed an alternative data structure that can handle multiple streams. The sliding window model was extensively studied.

Cohen and Strauss [8,9] considered decaying aggregation on data streams under *general* decay functions. They showed that general decay can be reduced to sliding windows decay. However, sliding window decay is in a sense the "hardest" decay function and other functions, such as polynomial and exponential decay can be computed more efficiently.

Network

Decaying aggregation over a network is a generalization of decaying aggregation over data streams (data streams correspond to path networks). It was first considered by Cohen in [3] as Threshold decay (aggregation over neighborhoods). Cohen proposed efficient algorithms for decaying sum. General decay functions, other aggregates, and efficient algorithms and analysis to maintain these sketches in distributed settings were considered by Cohen and Kaplan [5,6]. On networks, just like on data streams, threshold decay is the "hardest" decay function, as all-distances sketches that support sums over neighborhoods, also support decaying sums under general decay functions. It is not known, however, whether other decay functions such as exponential or polynomial decay can be computed or approximated more efficiently by a tailored approach. Further work on decayed aggregation over a networks includes [10].

Euclidean Plane

Cohen and Kaplan [5] also considered decaying sums in the Euclidean plane and proposed an efficient construction of a data structure (based on incremental construction of Voronoi diagrams [15]) that can support point queries. That is, for a query location and a query decay function, the data structure returns a respective approximate decaying sum.

Foundations

Metric Space

Decaying aggregates were considered on different metric spaces. The simplest one is one-dimensional Euclidean space such as the time dimension. Items have time stamps or sequence numbers and the relevance of an item at a certain time decays with elapsed time or with the number of items with a later time stamp. Decayed aggregates are also used on a higher dimensional Euclidean space and on a networks (directed or undirected graphs) where nonnegative lengths are associated with edges and distances correspond to shortest paths lengths.

Decay Functions

The decayed weight of an item at a location depends linearly on its original weight and decreases with distance. It is defined as a product of its original weight and the value of the decay function on the distance of the item from the location. A decay function $g(x)$ is a non-increasing function defined over the nonnegative reals. It is often convenient to define $g(0) = 1$. Some natural decay functions are *threshold functions*, where items within a certain distance from the location have their original weights and other items have 0 weight, *Exponential decay*, where the weight decreases exponentially with the distance, and *Polynomial decay*, where the weight decreases polynomially with the distance.

Threshold decay assigns equal importance to all data values within a distance of r and disregards all other data values. Exponential decay is defined with respect to a parameter $\lambda > 0$, $g(x) = \exp(-\lambda x)$. Exponential decay is convenient to compute when used for time decay as it can be maintained easily using a single register (It is not known, however, if it is simpler to compute than other decay functions on other metric spaces). Exponential-decay over time captures some natural phenomena such as radioactive decay.

Polynomial decay has the form $g(x) = 1/(1 + ax^{\alpha})$ for parameters $a > 0$ and $\alpha \geq 1$. Many natural effects (for example, electro-magnetic radiation) have polynomial decrease with distance. Polynomial decay is often a natural choice when a smooth decay is desired and when Exponential decay is too drastic [9]. In many natural graphs (like d-dimensional grids), the neighborhood size increases polynomially with the distance and exponential decay suppresses longer horizons.

Aggregate Functions

Any aggregate function over a set of weighted items can be applied to the decayed weights of the items to obtain a corresponding decaying aggregate.

The *decaying sum* is the sum of the decaying weights of the items. In the special case where the weights are binary, this aggregate is referred to as the *decaying count*. A related aggregate is the *decaying average*, defined as the ratio of the decaying sum and the decaying count of the items. Important aggregates that can be reduced to (approximate) decaying sums are (approximate) decaying variance and moments [4]. These aggregates can also be defined with respect to a subpopulation of items that is specified by a predicate.

Other aggregates are *decaying weighted random sample of a certain size* with or without replacement and derived aggregates such as quantiles (using a folklore technique, an approximate quantile with confidence $1-\delta$ can be obtained by taking the p quantile of $O(\in^{-2}\ln\delta^{-1})$ independent random samples) and heavy hitters (all distinct identifiers with total decaying weight above some threshold).

Computational Challenges

A decaying aggregate of a location with respect to a decay function can be computed by processing all items, calculating the distances and computing the decayed weights, and finally computing the desired aggregate on this set. This approach provides exact answers (up to numerical computation errors) but is highly inefficient and often infeasible.

This naive approach requires a linear pass over all items for each location and decay function pair of interest in aggregates with respect to multiple locations or multiple decay functions.

On massive data sets, this approach is infeasible: on massive data stream, one can not store the full history of observed items and in distributed data sets, one can not replicate the full information at every node.

The common approach for massive data sets is to use summaries that enable *more efficient* computation of *approximate* aggregate values. Algorithms maintain a concise data structures that "summarizes" the full information. The size of these data structure determines the amount of book keeping and/or communication required. A desirable feature of these summaries is that they support aggregations over *selected subpopulations* of the items.

Decaying aggregation, where summaries must be able to support multiple locations and decay functions, imposes greater challenges.

All Distances Sketches

Useful data structures for decayed aggregation are *all-distances sketches*. The all-distances sketch of a location is a concise encoding of sketches of the weighted sets of items that are within some distance from the location, for all possible distances. If the sketches support approximate sum, the all-distances sketch supports decaying sum with respect to any decay function.

In many applications, it is desired to efficiently obtain these sketches for multiple locations. Fortunately, all-distances sketches can be computed and maintained efficiently in many settings.

Exponential histograms [1] and the wave summaries [14] are applicable on data streams. For the more general network setting, *all-distances k-mins sketches* [3], which can be computed efficiently in a distributed setting [5]. *All-distances bottom-k sketches*, which provide tighter estimates of the decaying sum than all-distances k-mins sketches were proposed in [6].

The data structure used in [5] for the Euclidean plane essentially encodes all-distances k-mins sketch for any query point in the plane.

Key Applications

Applications of decaying aggregation can be classified into two main categories.

The first category is *prediction or estimation* of a value at a location. In this context, the data items are viewed as samples or measurements from some underlying smooth distribution and the decayed aggregate is a way to estimate or predict the value or some statistics at a query location. For example, the items are measurements of some environmental parameter (humidity, temperature, air pollution) collected by a sensor network and the decaying aggregate is an estimate of the value at other location. Application in this category are [7,13,16–18].

The second category is some measure of *influence*: the items constitute the complete data set and the decayed aggregate is a measure of influence or closeness of a property to a location. For example, items are undirected sources of electro magnetic radiation and the decaying aggregate is a measure of the radiation level at a query location. Other applications in this category are content-based routing in p2p networks [11].

Cross-references

- ▶ Approximate Query Processing
- ▶ Data Sketch/Synopsis
- ▶ Data Stream

Recommended Reading

1. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and issues in data stream systems. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002.
2. Bremler-Barr A., Cohen E., Kaplan H., and Mansour Y. Predicting and bypassing internet end-to-end service degradations. In Proc. 2nd ACM SIGCOMM Workshop on Internet Measurement, 2002.

3. Cohen E. Size-estimation framework with applications to transitive closure and reachability. J. Comput. Syst. Sci., 55:441–453, 1997.
4. Cohen E. and Kaplan H. Efficient estimation algorithms for neighborhood variance and other moments. In Proc. 15th Annual ACM-SIAM Symp. on Discrete Algorithms, 2004.
5. Cohen E. and Kaplan H. Spatially-decaying aggregation over a network: model and algorithms. J. Comput. Syst. Sci., 73:265–288, 2007.
6. Cohen E. and Kaplan H. Summarizing data using bottom-k sketches. In Proc. ACM SIGACT-SIGOPS 26th Symp. on the Principles of Dist. Comp., 2007.
7. Cohen E., Kaplan H., and Oldham J.D. Managing TCP connections under persistent HTTP. Comput. Netw., 31:1709–1723, 1999.
8. Cohen E. and Strauss M. Maintaining time-decaying stream aggregates. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2003.
9. Cohen E. and Strauss M. Maintaining time-decaying stream aggregates. J. Algorithms, 59:19–36, 2006.
10. Cormode G., Tirthapura S., and Xu B. Time-decaying sketches for sensor data aggregation. In Proc. ACM SIGACT-SIGOPS 26th Symp. on the Principles of Dist. Comp., 2007.
11. Crespo A. and Garcia-Molina H. Routing indices for peer-to-peer systems. In Proc. 18th Int. Conf. on Data Engineering, 2002.
12. Datar M., Gionis A., Indyk P., and Motwani R. Maintaining stream statistics over sliding windows. SIAM J. Comput., 31 (6):1794–1813, 2002.
13. Floyd S. and Jacobson V. Random early detection gateways for congestion avoidance. IEEE/ACM Trans. Netw., 1(4), 1993.
14. Gibbons P.B. and Tirthapura S. Distributed streams algorithms for sliding windows. In Proc. 14th Annual ACM Symp. on Parallel Algorithms and Architectures, 2002, pp. 63–72.
15. Guibas L.J., Knuth D.E., and Sharir M. Randomized incremental construction of Delaunay and Voronoi diagrams. Algorithmica, 7:381–413, 1992.
16. Hua M., Pei J., Fu A.W.C., Lin X., and Leung H.-F. Efficiently answering top-k typicality queries on large databases. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
17. Jacobson V. Congestion avoidance and control. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988.
18. Keshav S., Lund C., Phillips S., Reingold N., and Saran H. An empirical evaluation of virtual circuit holding time policies in IP-over-ATM networks. J. Select. Areas Commun., 13 (8):1371–1382, 1995.
19. Scott D.W. Multivariate Density Estimation: Theory, Practice and Visualization. Wiley, New York, 1992.

Decentralized Data Integration System

▶ Peer Data Management System

Decision Rule Mining in Rough Set Theory

Tsau Young (T.Y.) Lin
San Jose State University, San Jose, CA, USA

Synonyms

Decision rules; Classification rules; Rough Set theory (RST); Extensional relational databases (ERDB)

Definition

Rough set theory (RST) has two formats, abstract and table formats. In this entry, abstract format is hardly touched. The table format, by definition, is a theory of extensional relational databases (ERDB). However, their fundamental goals are very different. RST is on data analysis and mining, while databases are on data processing.

In RST, a relation, which is also known as information table, is called a decision table (DT), if the attributes are divided into two disjoint families, called conditional and decision attributes. *A tuple in such a DT, is interpreted as a decision rule, namely, the conditional attributes functionally determine decision attributes.* A sub-relation is called a Value Reduct, if it consists of a minimal subset of minimal length decision rules that has the same decision power as the original decision table. RST has the following distinguished theorem.

Every decision table can be reduced to a value reduct [5].

This theorem has been regarded as a data mining theorem. However, in its true natural, it is a data reduction theorem. There are some trivial points in this theorem: If there is a conditional attribute that is a candidate key, then the column is an attribute reduct, and each attribute value is a value reduct. There are more than one value reducts.

One fundamental weak point of this theorem is the assumption that every tuple is a rule. So a new theory, based on the assumption that only the high frequency tuples can be regarded as rules, has started, but not very far [3,4].

Historical Background

Rough set theory (RST) has two formats, abstract and table formats. In the abstract format, a rough set is represented by the upper and lower approximations of equivalence classes. In terms of pure math, they are the closure and interior of special topological spaces, called clopen spaces. However, this entry does not cover this aspect.

The table format of RST is a formal theory derived from the logical studies of tables (relations). So theoretically speaking, RST is a sub-theory of ERDB, however, historically, they were developed under different philosophies. RDB assumes the semantics of data is known and focuses on organizing data through its semantics. On the other hand, RST assumes data is completely defined by the given table, and focuses on data analysis, rule discovery and etc.

Foundations

Though RST has been regarded as a methodology of data mining, the core results are actually data reduction or rule reduction. Namely, RST provides a methodology to reduce a decision table to a minimal but an equivalent table. They are equivalent in the sense that both tables provide the same decision power. This minimal decision table is called value reduct. This reduction is not unique. In other words, given a decision table, there is a set of value reducts, each is equivalent to the original table.

In this section some fundamental concepts will be introduced, In addition, the procedure will be illustrated by an example. There are two major steps; one is the attribute reducts that are generalization of candidate keys. The second step is tuple reduction.

Attribute(Column) Reducts and Candidate Keys

In a relation, an attribute, or a set of attributes K is called a candidate key if all attributes of the relation is functionally depended on K (or K functionally determines every attribute), and K is a minimal set [1]. In other words, candidate key is a set of "independent variables" that all other attributes can be expressed as functions of the candidate keys. In decision table, the corresponding concept is equivalent to find a set of "independent conditions" for "if-then" rules.

Let S = $(U, \mathcal{A} = \mathcal{C} \bigcup \mathcal{D}, Dom, \rho)$ be a decision table, where

$$\mathcal{C} = A_1, A_2, \ldots, A_i, \ldots, A_n,$$

$$\mathcal{D} = B_1, B_2, \ldots, B_i, \ldots, B_m.$$

Then S is said to be a consistent decision table, if $\mathcal{C}$ functionally determine $\mathcal{D}$.

Definition 1 *$\mathcal{B}$ is called a attribute reduct of* S, *if $\mathcal{B}$ is a minimal subset of $\mathcal{C}$ such that $\mathcal{B}$ functionally determine $\mathcal{D}$.*

It is clear that such a $\mathcal{B}$ is not necessarily unique. If $\mathcal{D}$ is chosen to be the whole table $\mathcal{A}$, then the reduct is the extensional candidate key. The algorithm of finding the attribute reduct is quite straightforward; it is nearly the same as that of finding a candidate key, which can be found in almost any database text.

Value Reducts – Simplest Decision Rules

In the step of attribute reduct, the reduction algorithm find the sets of minimal columns. In this section, a similar procedure is considered, but one tuple at a time. The minimal set is called value reduct.

Illustration

Without losing the generality, the idea can be explained by the following table:

U	Location	Test	New	Case	Result
ID-1	Houston	10	92	03	10
ID-2	San Jose	10	92	03	10
ID-3	Palo Alto	10	92	04	10
ID-4	Berkeley	11	92	04	50
ID-5	New York	11	92	04	50
ID-6	Atlanta	11	92	04	50
ID-7	Chicago	11	93	70	99
ID-8	Baltimore	11	93	70	99
ID-9	Seattle	11	93	70	99
ID-10	Chicago	51	95	70	94
ID-11	Chicago	51	95	70	95

Select a Decision Table From the table given above, the following decision table will be considered: Let U be the universe, RESULT be the decision attribute, and C = TEST, NEW, CASE be the set of conditional attributes.

Each tuple in the DT is considered as an if-then rule.

Split the Decision Table Two tuples ID-10 and ID-11 form a table of inconsistent rules. Nine tuples ID-1 to ID-9 form a consistent table. From now on the term in this section "decision table" is referred to this consistent table.

Decision Classes The equivalence relation IND(RESULT) classifies entities into three equivalence classes, called decision classes

$$\text{DECISION1} = \{\text{ID-1}, \text{ID-2}, \text{ID-3}\} = [10]_{RESULT}$$
$$\text{DECISION2} = \{\text{ID-4}, \text{ID-5}, \text{ID-6}\} = [50]_{RESULT}$$
$$\text{DECISION3} = \{\text{ID-7}, \text{ID-8}, \text{ID-9}\} = [99]_{RESULT}$$

Condition Classes Let C = {TEST, NEW, CASE} be the conditional attributes. The equivalence relation IND (C) classifies entities into four equivalence classes, called condition classes:

$$\text{IND(C)-1} = \{\text{ID-1}, \text{ID-2}\},$$
$$\text{IND(C)-3} = \{\text{ID-4}, \text{ID-5}, \text{ID-6}\},$$
$$\text{IND(C)-2} = \{\text{ID-3}\},$$
$$\text{IND(C)-4} = \{\text{ID-7}, \text{ID-8}, \text{ID-9}\}$$

Knowledge Dependencies Pawlak regards a partition (classification) as a knowledge, and observe that an attribute defines a partition on the entities. So relationships between attributes are regarded as relationships between partitions, and hence between knowledges.

Observe that the entities that are indiscernible by conditional attributes are also indiscernible by decision attributes, namely the following inclusions are obtained

$$\text{IND(C)-1} \subseteq \text{DECISION1};$$
$$\text{IND(C)-2} \subseteq \text{DECISION1};$$
$$\text{IND(C)-3} \subseteq \text{DECISION2};$$
$$\text{IND(C)-4} \subseteq \text{DECISION3}.$$

These inclusions imply that the equivalence relation IND(C) is a refinement of IND(RESULT) in mathematics. In RST, they imply that the knowledge IND (RESULT) is knowledge depended on (coarser than) the knowledge IND(C). Or equivalently, RESULT are Knowledge Depended on C.

If-then Rules Knowledge dependences can be expressed by if-then rules:

1. IfTEST=10, NEW=92, CASE=03, then RESULT=10
2. IfTEST=10, NEW=92, CASE=04, then RESULT=10
3. IfTEST=11, NEW=92, CASE=04, then RESULT=50
4. IfTEST=11, NEW=93, CASE=70, then RESULT=99

Attribute (Column) Reducts It is easy to verify that {TEST, NEW} and {TEST CASE} are two attribute reducts. Note that row ID-10 and ID-11 do not contribute to the consistent decision; so they should be ignored (For clarity, these have been removed.)

The two tables can be expressed as two sets of uniformly shortened rules:

U	Test	New	Result
ID-1	10	92	10
ID-2	10	92	10
ID-3	10	92	10
ID-4	11	92	50
ID-5	11	92	50
ID-6	11	92	50
ID-7	11	93	99
ID-8	11	93	99
ID-9	11	93	99

U	Test	Case	Result
ID-1	10	03	10
ID-2	10	03	10
ID-3	10	04	10
ID-4	11	04	50
ID-5	11	04	50
ID-6	11	04	50
ID-7	11	70	99
ID-8	11	70	99
ID-9	11	70	99

Set One:

1. If TEST = 10 and NEW = 92, then RESULT = 10
2. If TEST = 11 and NEW = 92, then RESULT = 50
3. If TEST = 11 and NEW = 93, then RESULT = 99

Set Two:

1. If TEST = 10, and CASE = 3, then RESULT =10
2. If TEST = 10, and CASE = 4, then RESULT = 10
3. If TEST = 11, and CASE = 4, then RESULT = 50
4. If TEST = 11, and CASE = 70, then RESULT = 99

More plainly, the *consistent* decisions made from the original decision table can be accomplished equivalently by either one of the two sets.

U	Test	New	Result
ID-1	10	X	10
ID-2	10	X	10
ID-3	10	X	10
ID-4	11	92	50
ID-5	11	92	50
ID-6	11	92	50
ID-7	11	X	99
ID-8	11	X	99
ID-9	11	X	99
ID-10	51	95	94
ID-11	51	95	95

U	Test	Case	Result
ID-1	10	X	10
ID-1	X	03	10
ID-2	10	X	10
ID-2	X	03	10
ID-3	10	04	10
ID-4	11	04	50
ID-5	11	04	50
ID-6	11	04	50
ID-7	11/X	70	99
ID-8	11/X	70	99
ID-9	11/X	70	99
ID-10	51	70	94
ID-11	51	70	95

Value Reducts First, observe that ID-10 and ID-11 have been moved back. The reasons are that the two inconsistent rules will have impact to the final form of value reduct; see below. The first table gives

1. FIRST SET of Shortest Rules
   Rule1 If TEST = 10, then RESULT = 10
   Rule2 If TEST = 11 and NEW = 92, then RESULT = 50
   Rule3 If NEW = 93, then RESULT = 99

A casual user will not realize that the rules are derived from the consistent sub-table. So a more natural view is to look at the whole table. Fortunately these three rules are not disturbed by the inclusion of ID-10 and ID-11 tuples. In other words, The FIRST SET is a value reduct of the consistent rules.

The second table provides two sets of Shortest Rules

2. SECOND SET of Shortest Rules are: [Rule4a], [Rule6], [Rule7] in the list below.
3. THIRD SET of Shortest Rules are: [Rule4b], [Rule5], [Rule6], [Rule7] where each bracket [•] is referring to the following rules:
   Rule4a If TEST = 10, then RESULT = 10
   Rule4b If CASE = 3, then RESULT = 10
   Rule5 If TEST = 10 then RESULT = 10
   Rule6 If TEST = 11 and CASE = 4, then RESULT = 50
   Rule7 If TEST = 11 and CASE = 70, then RESULT = 99
   C-Rule8 If CASE = 70, then RESULT = 99 (This rule is valid only on consistence table.)

Note that the choice of [C-Rule8] is valid, if only the consistence table is considered. Pawlak had adopted this view [5]. However, as remarked earlier, a more natural view is: the value reduct is derived from the original whole table. In this case, [C-Rule8] is not a rule, as it is "in conflict" with ID-10 and ID-11. So Rule7 is used.

The illustrations on how to find the minimal sets of shortest rules is completed. There are three solutions (a modified view of Lin): (1) FIRST SET (Rule 1-3) (2) SECOND SET (Rule4a, 6, 7), and (3) THIRD SET (Rule4b, 6, 7). Taking the approach of Pawlak's book, the FIRST and SECOND SETS are the same as those of the author, but Third SET is (Rule4b, 6, C-Rule8).

Several comments are in order. In rough set theory, Pawlak had assumed that every tuple in a decision table is a rule; this assumption is debatable. So the view – only high frequency tuples (as in frequent itemsets) can be regarded as rules – should be explored. There are such efforts, but not very far [3,4]. Other variations also exist and should be explored.

Cross-references

▶ Data Mining
▶ Data Reduction
▶ Decision Tree Classification
▶ Decision Trees
▶ Frequent Itemsets and Association Rules
▶ Rough Set Theory, Granular Computing on Partition
▶ Rough Set Theory (RST)

Recommended Reading

1. Gracia-Molina H., Ullman J., and Windin J. Database Systems The Complete Book, Prentice-Hall, Englewood, Cliffs, NJ, 2002.
2. Lee T.T. Algebraic theory of relational databases. Bell Syst. Tech. J., 62(10):3159–3204, December 1983.
3. Lin T.Y. Rough set theory in very large databases. In Symp. in Modelling Analysis and Simulation, 1996, pp. 936–941.
4. Lin T.Y. and Han J. High frequent value reduct in very large databases. RSFDGrC, 2007, pp. 346–354.
5. Pawlak Z. Rough Sets. Theoretical Aspects of Reasoning about Data. Kluwer, Dordecht, 1991.

Decision Rules

▶ Decision Rule Mining in Rough Set Theory
▶ Deductive Data Mining Using Granular Computing

Decision Support

▶ Clinical Decision Support

Decision Tree

▶ Decision Tree Classification

Decision Tree Classification

Alin Dobra
University of Florida, Gainesville, FL, USA

Synonyms

Decision tree; Classification tree

Definition

Decision tree classifiers are decision trees used for classification. As any other classifier, the decision tree classifiers use values of attributes/features of the data to make a class label (discrete) prediction. Structurally, decision tree classifiers are organized like a *decision tree* in which simple conditions on (usually single) attributes label the edge between an intermediate node and its children. Leaves are labeled by class label predictions. A large number of learning methods have been proposed for decision tree classifiers. Most methods have a tree growing and a pruning phase. The tree growing is recursive and consists in selecting an attribute to split on and actual splitting conditions then recurring on the children until the data corresponding to that path is pure or too small in size. The pruning phase eliminates part of the bottom of the tree that learned noise from the data in order to improve the generalization of the classifier.

Historical Background

Decision tree classifiers were first introduced by Breiman and his collaborators [2] in 1984 in the statistics community. While the impact in statistics was not very significant, with their introduction in 1986 by Quinlan in machine learning literature[11], the decision tree classifiers become of the premier classification method. A large amount of research was published on the subject since in the machine learning literature. There was a renewed interest in decision tree classifiers in the 1990 in the data-mining literature due to the fact that scalable learning is nontrivial and, more importantly, the business community prefers decision tree classifiers to other classification methods due to their simplicity. A comprehensive survey of the work on decision tree classifiers can be found in [10].

Foundations

Decision tree classifiers are especially attractive in a data mining environment for several reasons. First, due to their intuitive representation, the resulting model is easy to assimilate by humans [2]. Second, decision tree classifiers are non-parametric and thus especially suited for exploratory knowledge discovery. Third, decision tree classifiers can be constructed relatively fast compared to other methods [8]. And last, the accuracy of decision tree classifiers is comparable or superior to other classification models [8].

As it is the case for most classification tasks, the kind of data that can be represented by decision tree classifiers is of tabular form, as depicted in Table 1. Each data point occupies a row in the table. The names of columns are characteristics of the data and are called *attributes*. Attributes whose domain is numerical are called *numerical attributes*, whereas attributes whose domain is not numerical are called *categorical attributes*. One of the categorical attributes is designated as the *predictive attribute*. The predictive attribute needs to be predicted from values of the other attributes. For the example in Table 1, “Car Type” is a categorical attribute, “Age” is a numerical attribute and “Lives in Suburb?” is the predictor attribute.

Figure 1 depicts a classification tree, which was built based on data in Table 1. It predicts if a person lives in a suburb based on other information about the person. The predicates, that label the edges (e.g., Age $\leq$ 30), are called *split predicates* and the attributes involved in such predicates, *split attributes*. In traditional classification and regression trees only deterministic split predicates are used (i.e., given the split predicate and the value of the the attributes, it can be determined if the attribute is true or false). Prediction with classification trees is done by navigating the tree on true predicates until a leaf is reached, when the prediction in the leaf (YES or NO in the example) is returned.

Decision Tree Classification. Table 1. Example training database

Car Type	Driver Age	Children	Lives in Suburb?
sedan	23	0	yes
sports	31	1	no
sedan	36	1	no
truck	25	2	no
sports	30	0	no
sedan	36	0	no
sedan	25	0	yes
truck	36	1	no
sedan	30	2	yes
sedan	31	1	yes
sports	25	0	no
sedan	45	1	yes
sports	23	2	no
truck	45	0	yes

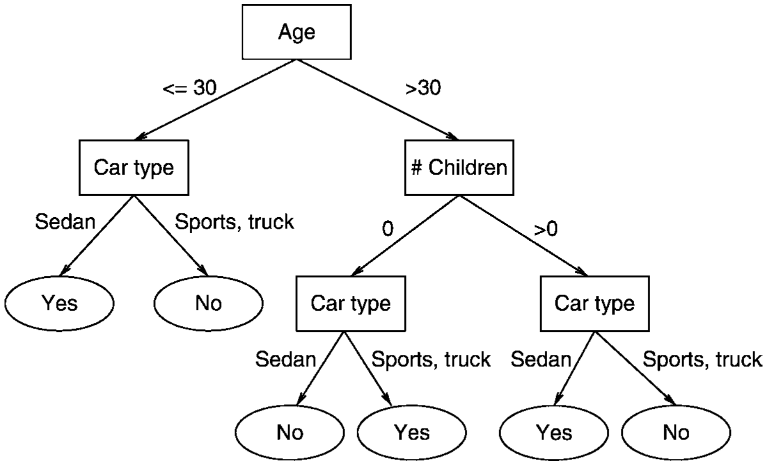
Formal Definition

A classification tree is a directed, acyclic graph $\mathcal{T}$ with tree shape. The root of the tree – denoted by Root($\mathcal{T}$) – does not have any incoming edges. Every other node has exactly one incoming edge and may have 0, 2 or more outgoing edges. A node T without outgoing edges is called *leaf node*, otherwise T is called an *internal node*. Each leaf node is labeled with one class label; each internal node T is labeled with one attribute variable $X_{T,}$ called the *split attribute*. The class label associated with a leaf node T is denoted by Label(T).

Each edge (T, T') from an internal node T to one of its children T' has a predicate $q_{(T,T')}$ associated with it where $q_{(T,T')}$ involves only the splitting attribute X_T of node T. The set of predicates Q_T on the outgoing edges of an internal node T must contain disjoint predicates involving the split attribute whose conjunction is true – for any value of the split attribute exactly one of the predicates in Q_T is true. The set of predicates in Q_T will be reffered to as *splitting predicates of T*.

Given a classification tree $\mathcal{T}$, the associated classifier is defined as $C_{\mathcal{T}}(x_1, \ldots, x_m)$ in the following recursive manner:

$$C(x_1, \ldots, x_m, T) = \begin{cases} \text{Label}(T) & \text{if } T \text{ is a leaf node} \\ C(x_1, \ldots, x_m, T_j) & \text{if } T \text{ is an internal node, } X_i \text{ is label of } T\text{, and } \text{q}(T,\ T_j)\ (xi){=}\text{true} \end{cases}$$



Decision Tree Classification. Figure 1. Example of decision tree classifier for training data in Table 1.

$$C_{\mathcal{T}}(x_1, \ldots, x_m) = C(x_1, \ldots, x_m, \mathrm{Root}(\mathcal{T}))$$

thus, to make a prediction, start at the root node and navigate the tree on true predicates until a leaf is reached, when the class label associated with it is returned as the result of the prediction.

Building Decision Tree Classifiers

Several aspects of decision tree construction have been shown to be NP-hard. Some of these are: building optimal trees from decision tables [6], constructing minimum cost classification tree to represent a simple function [4], and building optimal classification trees in terms of size to store information in a dataset [14].

In order to deal with the complexity of choosing the split attributes and split sets and points, most of the classification tree construction algorithms use the greedy induction schema in Fig. 2. It consists in deciding, at each step, upon a split attribute and split set or point, if necessary, partitioning the data according with the newly determined split predicates and recursively repeating the process on these partitions, one for each child. The construction process at a node is terminated when a termination condition is satisfied. The only difference between the two types of classification trees is the fact that for k-ary trees no split set needs to be determined for discrete attributes. By specifying the split attribute selection criteria and the split point selection criteria various decision tree classifier construction algorithms are obtained.

Once a decision is made to make a node a leaf, the majority class is used to label the leaf and will be the prediction made by the tree should the leaf be reached.

Split Attribute and Split Point Selection

At each step in the recursive construction algorithm, a decision on what attribute variable to split is made. The purpose of the split is to separate, as much as possible, the class labels from each others. To make this intuition useful, a metric that estimates how much the separation of the classes is improved when a particular split is performed is needed. Such a metric is called a *split criteria* or a *split selection method.*

There is extensive research in the machine learning and statistics literature on devising split selection criteria that produce classification trees with high predictive accuracy [10].

The most popular class of split selection methods are *impurity-based* [2,11]. The popularity is well deserved since studies have shown that this class of split selection methods have high predictive accuracy [8], and at the same time they are simple and intuitive. Each impurity-based split selection criteria is based on an impurity function $\Phi(p_1,\ldots,p_k)$, with p_j interpreted as the probability of seeing the class label c_j. Intuitively, the impurity function measures how impure the data is. It is required to have the following: (i) to be concave, (ii) to have a unique maximum at $(1/k,\ldots,1/k)$ (the most impure situation), and (iii) to achieve the minimum value for $(1,0,\ldots,0), (0,1,0,\ldots,0), \ldots, (0,\ldots,0,1)$ (i.e., when all data has the same class label). Given such an impurity measure, the impurity of a node T is $i(T) = \Phi(P[C = c_1|T],\ldots, P[C = c_k|T])$, where $P[C = c_j|T]$ is the probability that the class label is c_j given that the data reaches node T. These statistics are computed from the training data in the process of building the decision tree classifier.

```
Input: node $T$, data-partition $D$, splits election method $\mathcal{V}$
Output: classification tree $\mathcal{T}$ for $D$ rooted at $T$

Top-Down Classification Tree Induction Schema:
BuildTree(Node $T$, data-partition $D$, split attribute selection method $\mathcal{V}$)
    Apply $\mathcal{V}$ to $D$ to find the split attribute $X$ for node $T$.
    Let $n$ be the number of children of $T$.
    if ($T$ splits)
        Partition $D$ into $D_1,\ldots, D_n$ and label note $T$ with split attribute $X$
        Create children nodes $T_1,\ldots,T_n$ of $T$ and label the edge $(T, T_i)$
                with predicate $q(T, T_i)$
        foreach $i \in \{1,.., n\}$
            BuildTree($T_i$, $D_i$, $\mathcal{V}$)
        endforeach
    else
        Label $T$ with the majority class label of $D$
    endif
```

Decision Tree Classification. Figure 2. Classification tree induction schema.

Given a set Q of split predicates on attribute variable X that split a node T into nodes $T_1,\ldots,T_n$, define the *reduction in impurity* as:

$$\begin{aligned}\Delta i(T,X,Q) &= i(T) - \sum_{i=1}^{n} P[T_i|T]\,.\,i(T_i)\\ &= i(T) - \sum_{i=1}^{n} P[q_{(T,T_i)}(X)|T]\,.\,i(T_i)\end{aligned} \tag{1}$$

Intuitively, the reduction in impurity is the amount of purity gained by splitting, where the impurity after split is the weighted sum of impurities of each child node.

Two popular impurity measures are:

$$\texttt{Gini index:}\ \mathrm{gini}(p_1,\ldots,p_n) = 1 - \sum_{i=1}^{n} p_i^2$$

$$\textbf{Entropy:}\ \mathrm{entropy}(p_1,\ldots,p_n) = -\sum_{i=1}^{n} p_i \log(p_i)$$

When used in conjunction with the (1), they produce the *Gini Gain* and *Information Gain* split point selection criteria. A split attribute criteria can simply be defined as the largest value of the split point selection criteria for any predicate involving the attribute (i.e., the best split). In this way, the best split point is determined simultaneously with the evaluation of an attribute thus no other criteria is necessary.

The selection of the attribute can be dissociated from the selection of the split point. A measures that test the usefulness of a split on an attribute without considering split points is χ^2-test.

Some comments on efficiently finding the split point for the two types of attributes, categorical and numerical, are in order. For numerical attributes, only splits of the form $X > 10$ are considered thus only as many split as there are data-points are possible. For categorical attributes in the case when k-ary splits are allowed (Quinlan decision tree classifier), there is only one possible split on an attribute. The situation is more complicated for binary split trees (Breiman et al.) since there are an exponential number of possible split points (sets in this case) to consider. Fortunately, in this last situation a powerful result due to Breiman [2] leads to a linear algorithm after a sort in a specific order is performed.

Tree Pruning

An important question with respect to building decision tree classifiers is when to stop the growing of the tree. Since the estimation of probabilities that make up the formulas for the split criteria becomes less and less statistically reliable as the tree is build (data is fragmented at an exponential speed due to splits) the tree will eventually learn noise. In machine learning terminology this is called *overfitting* and should be avoided to the greatest extent possible.

Finding the point where the actual learning stops and overfitting starts is problematic. For decision tree classification there are two distinct approaches to addressing this problem: (i) detect overfitting during tree growth and stop learning, and (ii) grow a large tree and, using extra independent data, determine what part of the tree is reliable and what part should be discarded.

A statistical test like χ^2-test can be used to detect the point where overfitting occurs. This turns out to work in some circumstances but not others. This method of stopping the tree growth is preferred when the χ^2-test is used for attribute selection as well.

The most popular method to producing good quality trees is to *prune* an overly large tree. The most popular method to pruning is *Quinlan's Re-substitution Error Pruning*. Re-substitution error pruning consists of eliminating subtrees in order to obtain a tree with the smallest error on the *pruning set*, a separate part of the data used only for pruning. To achieve this, every node estimates its contribution to the error on pruning data when the majority class is used as an estimate. Then, starting from the leaves and going upward, every node compares the contribution to the error by using the local prediction with the smallest possible contribution to the error of its children (if a node is not a leaf in the final tree, it has no contribution to the error, only leaves contribute), and prunes the tree if the local error contribution is smaller – this results in the node becoming a leaf. Since, after visiting any of the nodes the tree is optimally pruned – this is the invariant maintained – when the overall process finishes, the whole tree is optimally pruned.

Other pruning techniques can be found in [9]. They tend to be significantly more complicated than re-substitution error pruning.

Key Applications

Scientific classification, medical diagnosis, fraud detection, credit approval, targeted marketing, etc.

URL to Code

http://www.dtreg.com/
http://eric.univ-lyon2.fr/~ricco/sipina_overview.html
http://www.statistics.com/resources/glossary/c/cart.php
http://www.statsoft.com/textbook/stcart.html
http://www.salfordsystems.com/

Cross-references

- ▶ Classification
- ▶ Decision Trees

Recommended Reading

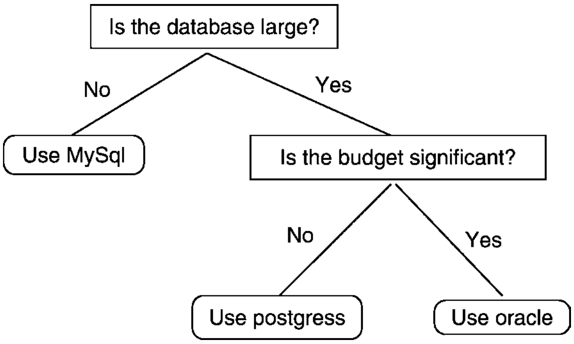
1. Agresti A. Categorical data analysis. John Wiley and Sons, 1990.
2. Breiman L., Friedman J.H., Olshen R.A., and Stone C.J. Classification and regression trees. Belmont: Wadsworth, 1984.
3. Buntine W. Learning classification trees. Artificial Intelligence frontiers in statistics Chapman & Hall, London, pp. 182–201.
4. Cox L.A., Qiu Y., and Kuehner W. Heuristic least-cost computation of discrete classification functions with uncertain argument values. Annals of Operations Research, 21: 1–30, 1989.
5. Frank E. Pruning decision trees and lists. Doctoral dissertation, Department of Computer Science, University of Waikato, Hamilton, New Zealand, 2000.
6. Hyafil L., and Rivest R.L. Constructing optimal binary decision trees is np-complete. Information Processing Letters, 5:15–17, 1976.
7. James M. Classification algorithms.Wiley, 1985.
8. Lim T.-S., Loh W.-Y., and Shih Y.-S. An empirical comparison of decision trees and other classification methods (Technical Report 979). Department of Statistics, University of Wisconsin, Madison, 1997.
9. Loh W.-Y. and Shih Y.-S. Split selection methods for classification trees. Statistica Sinica, 1997, p.7.
10. Murthy S.K. Automatic construction of decision trees from data: A multi-disciplinary survey. Data Mining and Knowledge Discovery, 1997.
11. Quinlan J.R. Induction of decision trees. Machine Learning, 1:81–106, 1986.
12. Quinlan J.R. Learning with Continuous Classes. In Proc. 5th Australian Joint Conference on Artificial Intelligence, 1992, pp. 343–348.
13. Quinlan J.R. C4.5: Programs for machine learning. Morgan Kaufman, 1993.
14. Murphy O.J. and Mccraw R.L. Designing storage efficient decision trees. IEEE Transactions on Computers, 40: 315–319, 1991.

Decision Trees

ALIN DOBRA
University of Florida, Gainesville, FL, USA

Synonyms

Classification trees



Decision Trees. Figure 1. Example of a decision tree for selecting a database.

Definition

Decision trees are compact tree like representations of conditions that specify when a decision should be applied together with the actions/decision. Decision trees consist into intermediate nodes and leaf nodes. The outgoing edges from intermediate nodes are labeled by conditions. The leaf nodes are labeled by decisions or actions. The way decision trees are used is by starting at the root then navigating down on true conditions until a leaf is reached. The action or decision in the leaf is then taken. Decision trees are just a compact representation of decision rules: the condition under which an action is taken is the conjunction of conditions starting at the root of the decision tree and leading to the leaf labeled by the action. An example of a decision tree is given in Fig. 1.

Key Points

Decision trees are an important type of representation for any kind of complex set of decisions that are conditioned on multiple factors. They are more compact than decision rules but less flexible. The main appeal of decision trees is their intuitiveness; it is very easy to understand how the decision is taken.

While they have uses in any areas that need decision support of some kind, in databases their main use is in specifying decision tree classifiers and decision tree regressors. Decision trees are mostly used in one of these two forms in databases.

Cross-references

- ▶ Decision Tree Classification
- ▶ Scalable Decision Tree Construction

Recommended Reading

1. Lindley D.V. Making Decisions. Wiley, Hoboken, NJ, USA, 1991.

Declarative Networking

Timothy Roscoe
ETH Zurich, Zurich, Switzerland

Synonyms

Declarative overlay networks

Definition

Declarative Networking refers to the technique of specifying aspects of networked systems, such as routing algorithms, in terms of declarative queries over distributed network state. These queries are then executed by a distributed query processor to obtain the same effect as executing an implementation of the algorithm in an imperative language such as C or Java. Executable descriptions of distributed algorithms as queries are typically much more concise than imperative implementations, and more amenable to automated analysis.

Historical Background

Declarative Networking emerged in about 2004 as an application of results in data management and logic programming to problems of network overlay maintenance. Its roots can be traced in several areas: attempts to describe real-world network configurations formally, e.g., [11], network management systems built over a declarative framework, such as IBM Tivoli Console and various research systems, e.g., [12,15], and perhaps most significantly the field of distributed query processing systems (e.g., [5,6,10]).

The observation that distributed query processors need to construct overlay networks of some form to route data led to the idea that the routing protocols required might be specified declaratively. The idea of describing Internet routing protocols (and new variants on them) as queries over distributed data was proposed in [9], and shortly afterwards real systems for building overlay networks from declarative specifications began to appear. P2 [8] uses a variant of the Datalog language and pushes most aspects of the distributed algorithm into the declarative realm, while Node Views [3] uses a SQL-like language [2] and builds overlay networks as views over an underlying node database.

Further work has expanded the applicability of the approach from routing algorithms networks to more general distributed algorithms such as Chandy-Lamport consistent snapshots [14], specifying security properties of networks [1], and building complete sensor network applications [4].

Foundations

The basic principle of declarative networking can be illustrated by reference to the example of routing protocols. The purpose of a routing protocol is to continously maintain a *routing table* at each node in the network. This table provides a mapping from destination addresses to "next hop" nodes – those which are directly connected to the node where the table resides. The collection of routing tables in a network can be viewed as the result of a distributed calculation whose inputs are external data such as node liveness, link load levels, user-specified policy, etc. If one represents such external data as relations, routing tables can be regarded as a distributed view over such relations, and consequently the routing – the process of maintaining the routing tables – can be regarded as an instance of distributed view maintenance.

This can be illustrated further with the example of link-state routing (as used by the widespread Internet routing protocols OSPF and IS-IS), where connectivity information is flooded globally through the network and each router performs shortest-path computations on the complete connectivity graph. Using the simplified notation of [9], based on Datalog, one can write:

```
path(S,D,P,C) :- link(S,D,C),
                 P = f concatPath(link
                  (S,D,C), nil).
path(S,D,P,C) :- link(S,Z,C1), path
                 (Z,D,P2,C2),
                 C = f_sum(C1, C2),
                 P = f concatPath
                 (link(S,Z,C1), P2).
```

These first two query rules give the standard inductive defintion of reachability in a network: there is a path P from source node S to destination D with cost C if there is a (direct) link of cost C from S to D, or if there is a path P2 to D from a node Z adjacent to S, and C is the sum of the link cost C1 to Z and the path cost C2 from Z to D.

Two further rules can compute the best path from S to D, using the standard Datalog function for aggregates:

```
bestPathCost(S,D,AGG<C>) :- path
                          (S,D,P,C).
bestPath(S,D,P,C) :- bestPathCost
                     (S,D,C),
                     path(S,D,P,C).
```

Note that the definition of "best" in this example is deliberately unbound: by changing the sum function `f_sum` and the aggregate `AGG`, a variety of network metrics can be used. This illustrates some of the key claimed benefits of declarative networking: conciseness of specification, and ease of modification.

While the four rules above specify a link-state routing algorithm in some sense, they of course say nothing about how such a specification is to executed, *where* the rules are evaluated, or what messages need to traverse the network. However, as [9] shows, each term in a rule head and body can be annotated with a "location specifier", which identifies which node the tuple resides on when the rule is executed. With such identifiers, the specification above becomes:

```
path(@S,D,P,C) :- link(@S,D,C),
                  P = f concatPath(link
                  (@S,D,C), nil).
path(@S,D,P,C) :- link(@S,Z,C1), path
                  (@Z,D,P2,C2),
                  C = f_sum(C1, C2),
                  P = f concatPath(link
                  (@S,Z,C1), P2).
bestPathCost(@S,D,AGG<C>) :- path
                  (@S,D,P,C).
bestPath(@S,D,P,C) :- bestPathCost
                     (@S,D,C),
                     path(@S,D,P,C).
```

It can be seen that all rules execute entirely locally (on whichever node corresponds to the value `S` in the operand tuples) except the second one. For this, a message must be sent from `Z` to `S`, which is conveniently directly connected to `S`. Such a specification can be trivially planned and executed independently on each node in a network, and will result in best-path routing table on each node. Furthermore, the messages that will be sent during execution correspond to those that an imperative link-state implementation would need to transmit.

Loo et al. [7] detail various evaluation strategies, and also discuss some automated checks that may be performed on such algorithm specifications. In particular, if a set of relations (such as `link` in the above example) are asserted to mean direct connectivity, the system can determine at query plan time whether or not the routing protocol is well-formed, in that it will only cause a node to send messages to its direct neighbours. It is this "global" specification of rules which sets declarative networking systems apart from purely local event-condition-action (ECA) systems.

A declarative networking system like P2 [8] can directly execute such a specification, allowing very concise programs for distributed algorithms. [8] presents an implementation of the Chord overlay in 47 lines of Datalog, considerably shorter than the several thousand lines of C++ in the original imlemenentation.

A further key benefit of declarative networking is the flexibility afforded by the query framework. Very few assumptions are made about the network in question – even the point-to-point `link` predicate in the example above could be replaced a more complex relation conveying direct connectivity, for example one giving radio strength in a wireless network. The declarativity of the specification allows external data about the network to be cleanly integrated into the algorithmic framework using familiar techniques in ways that would be tedious, cumbersome, and brittle in an imperative implementation.

Key Applications

Declarative networking is currently the domain of research rather than industrial adoption. In addition to the specification of protocols for extensible network routing and the overlay component of distributed applications, the approach has been applied to building sensor network applications over wireless networks and the implementation of fault-tolerant replication algorithms for the purposes of performance analysis [13].

Future Directions

There are a number of open questions in declarative networking. Evaluating rules concurrently on a node so as to preserve intelligible semantics to the programmer is an open area, as is the field of optimizations of such distributed queries over the kinds of sparsely-connected graphs found computer networks. The best way of integrating the declarative networking functionality of a distributed application with imperative functionality local to a node is an area of ongoing study.

URL to Code

http://p2.berkeley.intel-research.net/, mirrored at http://p2.cs.berkeley.edu/
http://developer.berlios.de/projects/slow/
http://www.dvs.tu-darmstadt.de/research/OverML/index.html

Cross-references

▶ Deductive Databases
▶ Distributed Query Processing
▶ Event-Condition-Action Systems

Recommended Reading

1. Abadi M. and Loo B.T. Towards a declarative language and system for secure networking. In Proc. 3rd Int. Workshop on Networking meets Databases, USA, April 2007.
2. Behnel S. SLOSL – a modelling language for topologies and routing in overlay networks. In Proc. 1st Int. Workshop on Modeling, Simulation and Optimization of Peer-to-Peer Environments, 2008.
3. Behnel S. and Buchmann A. Overlay networks – implementation by specification. In Proc. ACM/IFIP/USENIX 6th Int. Middleware Conf., 2005.
4. Chu D., Tavakoli A., Popa L., and Hellerstein J. Entirely declarative sensor network systems. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 1203–1206.
5. Chun B., Hellerstein J.M., Huebsch R., Jeffery S.R., Loo B.T., Mardanbeigi S., Roscoe T., Rhea S., Shenker S., and Stoica I. Querying at Internet Scale (Demo). In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
6. Huebsch R., Chun B., Hellerstein J.M., Loo B.T., Maniatis P., Roscoe T., Shenker S., Stoica I., and Yumerefendi A.R. The architecture of PIER: an Internet-scale query processor. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005.
7. Loo B.T., Condie T., Garofalakis M., Gay D.E., Hellerstein J.M., Maniatis P., Ramakrishnan R., Roscoe T., and Stoica I. Declarative networking: language, execution and optimization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 97–108.
8. Loo B.T., Condie T., Hellerstein J.M., Maniatis P., Roscoe T., and Stoica I. Implementing declarative overlays. In Proc. 20th ACM Symp. on Operating System Principles, 2005, pp. 75–90.
9. Loo B.T., Hellerstein J.M., Stoica I., and Ramakrishnan R. Declarative routing: extensible routing with declarative queries. In Proc. Int. Conf. of the on Data Communication, 2005.
10. Loo B.T., Huebsch R., Hellerstein J.M., Roscoe T., and Stoica I. Analyzing P2P overlays with recursive queries. Technical Report IRB-TR-003-045, Intel Research, November 2003.
11. Roscoe T., Hand S., Isaacs R., Mortier R., and Jardetzky P. Predicate routing: enabling controlled networking. In Proc, First Workshop on Hot Topics in Networks, 2002.
12. Roscoe T., Mortier R., Jardetzky P., and Hand S. InfoSpect: using a logic language for system health monitoring in distributed systems. In Proc. 2002 ACM SIGOPS European Workshop, 2002.
13. Singh A., Maniatis P., Druschel P., and Roscoe T. BFT protocols under fire. In Proc. 5th USENIX Symp. on Networked Systems Design and Implementation, 2008.
14. Singh A., Maniatis P., Roscoe T., and Druschel P. Using queries for distributed monitoring and forensics. In Proc. First European Systems Conference, 2006.
15. Wawrzoniak M., Peterson L., and Roscoe T. Sophia: an information plane for networked systems. In Proc. 5th USENIX Symp. on Networked Systems Design & Implementation, 2003.

Declarative Overlay Networks

▶ Declarative Networking

Deductive Data Mining using Granular Computing

Tsau Young (T. Y.) Lin
San Jose State University, San Jose, CA, USA

Synonyms

Deductive data mining, model for automated data mining; Rough set theory, granular computing on partition

Definition

What is Deductive Data Mining (DDM)? It is a methodology that derives patterns from the mathematical structure of a given *stored* data. Among three core techniques of data mining [1], *classifications and association rule mining* are deductive data mining, while clustering is not, because its algorithms often use some properties of the ambient space.

What is Granular Computing (GrC)? In general, it is a problem solving methodology deeply rooted in human thinking. For example, human body is granulated into head, neck, and etc. However, the main concerns here are on the data mining aspect of GrC. Two views are presented. One is based on current technology, and the other is on the incremental approach to the ultimate goal.

A) *Mining Relational Databases (RDB) using GrC*: In GrC, a relation K (also known as information table in rough set theory) is a knowledge representation that maps a set U of entities into a set of attribute values. The collection of entities that is mapped to the same attribute value is called a granule. By replacing each attribute value with its granule, a relation of granules is formed. Observe that the collection of granules in each column is a partition (hence an equivalence relation) on U. Hence, the relation of granules is equivalent to the pair $(U, \mathcal{R})$, called Granular Data Model (GDM), where R is the set of equivalence relations that are defined by those columns. Observe that GDM determines a relation (up to isomorphism) uniquely. See Table 1. Note that GDM has a very explicit mathematical structure of data, hence

1. Mining RDB using GrC is DDM on GDM (or relation of granules). See "Key Applications – Deductive Data Mining on GDM". Two striking results will be explained.

   B) *Mining RDB over "real world sets" using GrC*: This is a goal statement. What is a "real world set"? Strictly speaking, this can not be answered mathematically (there is no semantics in mathematics). In GrC, the "real world set" has been modeled by the Fifth GrC model (relational GrC model), (U, β), where β is a collection of n-ary relations (n is not fixed). In this approach, the focus has been on capturing the interactions among elements of the universe. For example, in human society, n person may form a committee (hence they interact). In this formulation, the committee is a tuple (as each person may play a disticnt role) in one of the relation in β [18]. Note that Fifth GrC Model (in finite U) is the relational structure of the First Order Logic (without function symbols). RDB over a model of "real world set" can be viewed as a Semantic Oriented RDB. For now, the "real world set" is Fifth GrC Model, so the corresponding GDM is *R*elational GrC Model (Fifth GrC Model) based *GDM*, denoted by *RGDM*.
2. Mining RDB over "real world sets" using GrC is DDM on RGDM. See "Future Directions".

Historical Background

How was the term GrC coined? In the academic year 1996–1997, when T. Y. Lin took his sabbatical leave at UC-Berkeley, Zadeh suggested granular mathematics to be his research area. To limit the scope, Lin proposed the term granular computing [25] to label the research area. Though the label is new, the notion is quite ancient, e.g., the infinitesimal granules led to the invention of calculus. More recent examples are Heisenberg uncertainty principle (A precise position can only determine a granule of momentum) and fuzzy set theory [24]. For database and data analysis areas, the first work may be due to Hsiao in 1970 [7]; incidentally he is also well known for database machines. His model is essentially equivalent to relational database model and was called Attribute Based File Organization (evolved into attribute based data model (ABDM)) by Eugene Wong in 1971 [2]; Wong and Stonebraker co-founded database system INGRESS.

In ABDM, Hsiao imposed equivalence relations on attribute domains to increase the precision of data access. Around 1981, Ginsburg and Hull imposed another binary relation, "partial ordering" on attribute domains [4,5]. The goal is to capture the additional information provided by the order.

About the same time, Pawlak (1982) and Lee (1983) observed that an attribute can be regarded as a partition of the universe [8,23], and developed rough set theory and algebraic theory of relational databases. Their approaches are quite different from previous theories.

In 1988–1989, for approximate retrievals, Lin generalized topological neighborhood system to Neighborhood Systems (NS) by simply dropping the axioms of topology. He imposed NS structure on attribute domains and studied the Topological Data Model [9,11] (see cross references). NS is equivalent to a set of binary relations, so it is mathematically a generalization equivalence relation (Hsiaos works) and partial ordering (Ginsburg and Hulls works). However, semantically, they are different; in NS, each neighborhood (a granule) is regarded as a unit of uncertainty. Also in 1989, Lin applied the idea of computer security [10]. This is related to the information flow on access control model (see cross references) In 1992, Lin developed a NS-version of RS theory [12]. These are pre-GrC stories.

Next, are early works. By mapping NS onto Zadehs intuitive statements, Lin used NS as his first mathematical GrC model, developed Binary Relation based Data Model [13,21], and observed "complete Pawlak theories" in symmetric binary relations and some partial covering [19]. Binary relation based Data Model has many names, such as binary knowledge based, granular table, clustered table, semantic oriented table and, etc.; now it is called a Binary Granular Data Models (BGDM). Intrinsically, BGDM is a semantic oriented data model, and has been used to mine semantic oriented decisions (classifications) and association rules; it is still an ongoing theory.

Foundations

What is Data Mining?

What is data mining? There is no universally accepted formal definition of data mining, however the following informal description from [3] is rather universal: "data mining is a non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns from data."

As having been pointed out previously in several occasions, the terms, "novel," "useful," and "understandable" represent subjective views, and hence can not be used for scientific purpose, it is paraphrased as follows:

1. Deriving useful patterns from data.

This "definition" points out four key ingredients: data, patterns, derivation methodology and the real world meaning of patterns (implied by usefulness).

Deductive Data Mining

Convention. A *symbol* is a string of "bit and bytes" that has no *formal* real world meaning, or whose real world interpretation (if there is one) does *not* participate in formal processing. A symbol is termed a *word*, if the intended real world meaning *does participate* in the formal processing.

What is the *nature* of the data in DDM? It is best to examine how the data are created: In traditional data processing: (i) first a set of attributes, called relational schema, is selected. Then (ii) tuples of words are entered the system. They represent a set of real world entities. These words are called attribute values. Each such a word is created to represent a real world fact, however, only the symbol, but not the real world semantic, is stored.

How are the data processed? In traditional data processing environment, DBMS processes these data, under *human commands*, carries out the human perceived-semantics. However, from the system point of view, each word is a pure symbol (semantics are not stored). So in an automated data mining algorithm (without the human in the loop),

1. Words are treated as symbols.

For example, association mining algorithm merely *counts* the symbols without consulting their real world meanings. So association mining algorithm transforms a table of symbols into a set of expressions (frequent itemsets) of symbols. In summary,

1. A relation is a table of stored symbols.
2. Patterns are the mathematical expressions of the stored symbols. For examples, frequency itemsets.
3. Principle of derivations is the mathematical deduction.

Definition 1 *Deductive data mining is a data mining methodology that derives (using mathematical deductions only) patterns from the mathematical structure of stored symbols.*

Among three core topics of data mining [1],

Theorem 1 *Classifications and association rule mining are deductive data mining, while clustering may be not, because its algorithms use some properties of the ambient space.*

Isomorphism – A Critical Concept

As having pointed out that a relation is a table of symbols. So the following definition of isomorphism is legitimate: Let $K = (V, \mathcal{A})$ and $H = (V, \mathcal{B})$ be two relations, where $A = \{A^1, A^2, \ldots, A^n\}$ and $B = \{B^1, B^2, \ldots, B^m\}$.

Attributes A^i and A^j are isomorphic iff there is a one-to-one and onto map, $s : Dom(A^i) \rightarrow Dom(A^j)$ such that $A^j(v) = s(A^i(v)) \forall v \in V$. The map s is called an isomorphism. Intuitively, two attributes (columns) are isomorphic if one column turns into another one by properly renaming its attribute values.

Definition 2 *Two relations K and H are said to be isomorphic if there is a one-to-one and onto correspondence between two families of attributes so that every A^i is isomorphic to the corresponding B^j, and vice versa.*

Two relational Schema K and H are said to be isomorphic if all their instances are isomorphic.

Isomorphism is an equivalence relation defined on the family of all relational tables, so it classifies the tables into isomorphic classes. Moreover, in [15], the following observation was made: *Isomorphic relations have isomorphic patterns*. Its implications are rather far reaching. It essentially declares that patterns are syntactic in nature. They are patterns of the whole isomorphic class, even though many isomorphic relations may have very different semantics. Here is the important theorem:

Theorem 2 *A pattern is a property of an isomorphic class.*

In next section, a canonical model, called Granular Data Model (GDM), for each isomorphic class will be constructed.

Granular/Relational Data Models

In RDB, a relation (called an information table in rough set theory) can be viewed as a knowledge representation of a given universe U(a set of entities) in terms of a given set of attributes $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$. A tuple(row) is a representation of an entity. An attribute (column) A_j is a mapping that maps U to its attribute domain $\mathrm{dom}(A_j)$ (another classical set). Let those entities that are mapped to the same value be called granules. By replacing each attribute value with the corresponding granule, a new *relation of granules* is formed. Observe that the collection of granules in a column are mutually disjoint So it defines a partition and hence an equivalent relation. Let $\mathcal{R}$ be the collection of such equivalence relations. So the pair $(U, \mathcal{R})$, called a granular data model (GDM), is equivalent to the relation of granules.

The details are further illustrated in Table 1. Note that the first two columns, A-partition and B-partition, is a relation of granules. It is obviously equivalence to the GDM $(U, \{A - partition, B - partition\})$.

The last two columns, A-NAME and B-NAME, is the original given relation. It can be viewed as a relation of "meaningful" names of these granules. So a relation also will be called Data Model of Names (DMN), to emphasize this contrast to the relation of granules. Table 1 also illustrates the following isomorphism theorem, which was observed by Pawlak ([23], p. 56).

Deductive Data Mining using Granular Computing. Table 1. The granular data model of a relation

		Relation of granules		Original relation	
U	→	A-partition	B-partition	A-NAME	B-NAME
*e*1	→	{*e*1, *e*2, *e*6, *e*7}	{*e*1, *e*4}	diaper	sss
*e*2	→	{*e*1, *e*2, *e*6, *e*7}	{*e*2, *e*6, *e*7}	diaper	beer
*e*3	→	{*e*3, *e*5, *e*8}	{*e*3, *e*8}	coo	ttt
*e*4	→	{*e*4}	{*e*1, *e*4}	paat	sss
*e*5	→	{*e*3, *e*5, *e*8}	{*e*5}	coo	bar
*e*6	→	{*e*1, *e*2, *e*6, *e*7}	{*e*3, *e*8}	diaper	beer
*e*7	→	{*e*1, *e*2, *e*6, *e*7}	{*e*2, *e*6, *e*7}	diaper	beer
*e*8	→	{*e*3, *e*5, *e*8}	{*e*3, *e*8}	coo	ttt

Theorem 3 *A relation (up to isomorphism) determines a GDM and vice versa. In short, DMN, up to isomorphism, is equivalent to GDM.*

So "data" processing is equivalent to "granule" processing – Granular Computing. This introduced the data mining aspect of GrC in RDB.

Key Applications

In this section, two key applications are presented. Conceptually, the results are rather striking and counterintuitive; see Theorem 4 and Theorem 5.

Feature Constructions

What is an attribute (feature)? Surprisingly, this is a thought provoking question. In database theory, an attribute (in a relation schema) is a property, a view, or a reference (e.g., a coordinate). These are the concepts in human level. Based on these concepts, database experts represent entities into relations. Unfortunately, such knowledge representations are not thorough, namely, the semantics of attributes and attribute values are not stored in computer systems. So, automated data mining algorithms cannot utilize these semantics, simply because they are not in the computer systems. So the key question is:

1. How could the concept of features be described by data?

The first two columns of the Table 1 illustrates the idea: an attribute is a partition of the universe. Since the universe is a finite set, so there are the following surprise:

Theorem 4 *The number of non-isomorphic features that can be constructed from the given relation is finite; see [15].*

This is rather counterintuitive results. The reason is largely due to the fact that most people view features from data processing point of view, not data mining point of view.

The following example may illuminate the critical concepts: Consider a numerical relation of two columns. It can be visualized as a finite set of points in Euclidean plane. In this view, attributes are coordinates. So by simply rotating the X–Y-coordinate system (from 0^o to 360^o), infinitely many relations of P_θ can be generated, where θ is the angle rotated. Now we will examine these P_θ and four points whose polar coordinates are $(1,\ 0^o)$,$(1,\ 30^o)$,$(1,\ 45^o)$,$(1,\ 90^o)$; see

Table 2. Note that the X_θ-partition and Y_θ-partition do not change most of the time, when θ moves. It only changes at $\theta = 45^o$ and its multiples, namely, $\theta = 135^o, 315^o$. Note that though X_θ-coordinate and Y_θ-coordinate do change, but most of them are isomorphic to each other. Non-isomorphic ones occurs only at $\theta = 45^o$ and its multiples.

The key point is:

1. As U is finite, both A-Partition and B-partition columns only have finitely many possible distinct choices.

High Frequency Patterns in RDB

GDM is a powerful data mining representation: Observe that the frequency of the pair, (diaper, beer), is equal to the cardinality of the intersection of two granules, the diaper granule $\{e1, e2, e6, e7\}$ and the beer granule $\{e2, e6, e7\}$.

Deductive Data Mining using Granular Computing. Table 2. The granular data model of a relation

		Relation of granules of P_0		Original relation P_0	
U	→	X_0-partition	Y_0-partition	X_0-coordinate	Y_0-coordinate
p_1	→	$\{p_1\}$	$\{p_1\}$	1	0
p_2	→	$\{p_2\}$	$\{p_2\}$	$\sqrt{3}/2$	1/2
p_3	→	$\{p_3\}$	$\{p_3\}$	$\sqrt{2}/2$	$\sqrt{2}/2$
p_4	→	$\{p_4\}$	$\{p_4\}$	1/2	$\sqrt{3}/2$
p_5	→	$\{p_5\}$	$\{p_5\}$	0	1
		Relation of granules of P_{30^o}		Original relation P_{30^o}	
U	→	X_{30^o}-partition	Y_{30^o}-partition	X_{30^o}-coordinate	Y_{30^o}-coordinate
p_1	→	$\{p_1\}$	$\{p_1\}$	$\sqrt{3}/2$	$-1/2$
p_2	→	$\{p_2\}$	$\{p_2\}$	1	0
p_3	→	$\{p_3\}$	$\{p_3\}$	$\sqrt{3}/2$	1/2
p_4	→	$\{p_4\}$	$\{p_4\}$	$\sqrt{2}/2$	$\sqrt{2}/2$
p_5	→	$\{p_5\}$	$\{p_5\}$	1/2	$\sqrt{3}/2$
		Relation of granules of P_{45^o}		Original relation P_{45^o}	
U	→	X_{45^o}-partition	Y_{45^o}-partition	X_{45^o}-coordinate	Y_{45^o}-coordinate
p_1	→	$\{p_1, p_4\}$	$\{p_1\}$	$\sqrt{2}/2$	$-\sqrt{2}/2$
p_2	→	$\{p_2\}$	$\{p_2\}$	$\sqrt{3}/2$	$-1/2$
p_3	→	$\{p_3\}$	$\{p_3\}$	1	0
p_4	→	$\{p_1, p_4\}$	$\{p_4\}$	$\sqrt{2}/2$	$\sqrt{2}/2$

This illustration immediately gives us the following generalization. Let Δ be the Boolean algebra generated by granules. Then a Boolean algebraic expression in Δ is a set theoretical expression in U, hence it has the cardinality.

Definition 3 *A Boolean algebraic expression in Δ is a high frequency pattern (generalized frequent itemsets), if the cardinality of the expression is greater than the given threshold* [14].

This idea can also be expressed by Pawlak's decision logic. A formula is a high frequency pattern, if its meaning set has adequate cardinality. Here is a striking result:

Theorem 5 *High frequency patterns can be enumerated by solving a set of linear inequalities; see [17].*

This theorem has not been useful in applications; this is due to the high volume of the patterns. Theoretically, this theorem together with the facts in "Paradoxical Phenomena in Data Mining" led to the conclusion: Semantic oriented patterns are needed.

Future Directions

There are three subsections. In the first subsection, the goal statement and incremental approaches are outlined. In the second subsection, an example is given to illustrate the inadequacy of current results. The last subsection shows the feasibility of the outline in the first subsection will work.

Relational Database Theory Over "Real World Sets"

RDB and RST are based on classical set theory, in which every set is discrete. In other words, there are no interactions among elements of any set. However, a "real world set" contains interactions. For example in human society, n persons may form a committee (hence they have interactions). In GrC modeling, Fifth GrC Model (U, β) have been proposed as a model of "real world sets", where β is a collection of n-ary relations (n is not a fixed integer) that represent the interactions among elements in U [18]. Note that the roles of each member in the committee may be all distinct, so members are not exchangeable. So *granules are tuples, not necessarily subsets.* Many authors, including (Lin), have used subsets as granules; such notions may not be general enough.

A relational database over "real world sets," as in the classical relational database, is equivalent to "real world" GDM $((U, \beta), \mathcal{R})$, where (U, β) is a model of "real world sets" and $\mathcal{R}$ is a family of "equivalence relations” on the model of a “real world sets,” (U, β). In this paper, only the idea of simplest case will be illustrated, namely, β is a single n-nary relation. By combining β and $\mathcal{R}$, the pair $((U,\beta),\mathcal{R})$ can be transformed to the pair $(U,\mathcal{S})$, where each member S_i of $\mathcal{S}$ is a new relation that combined the n-nary relation β and the “equivalence relations” R_i. Namely, each attribute value in a relation β is replaced by an equivalence class of R_i. The resulting new β is S_i.

In this entry, only the case $n = 2$ will be discussed. In other words, only the following type of GDM over “real world sets” $(U, \mathcal{B})$, called Binary GrC Data Model (BGDM), will be considered, where $\mathcal{B}$ is a collection of binary relations.

Paradoxical Phenomena in Data Mining

In this subsection, some paradoxical phenomena in data mining will be discussed. First, two semantically very different, but syntactically equivalent relations K and K’ are presented; see Table 3. Namely, two isomorphic relations K and K’. are considered (see “Isomorphism – A Critical Concept”, where K is a relation about human beings and the other K’ is about hardware. The central theme is their paradoxical phenomena of high frequency patterns. Let the threshold be 3, namely, a sub-tuple in Table 3 is regarded as a pattern (frequent itemset or undirected association rule), if its number of occurrences is greater then or equal to 3:

1. Relation K:(TWENTY, SJ) is an interesting rule; it means the amount of business in San Jose is likely to be 20 million.

1′ Relation K’ (20, BRASS) is an isomorphic pattern. However, this rule is not meaningful at all. Material, such as BRASS, has no specific weight. The schema indicates that weight 20 is referred to PIN, not BRASS.

2′ Relation K’:(SCREW, BRASS) is interesting rule; it says screws are most likely made from BRASS.

2. Relation K: (MAR, SJ) is an isomorphic pattern; it is not interesting, because MAR refers to the birth month of a supplier, not to a city. The schema indicates that totally unrelated two columns (attributes) contain the pair.

3′ Relation K’: (20, SCREW) is interesting rule; it says screws are most likely weighing 20.

3. Relation K: (TWENTY, MARCH) is an isomorphic pattern; TWENTY refers to an ability of a supplier, not to birth month.

This analysis clearly indicates that the “real word” meanings of the universe and attribute domains have to be considered. In other words, a relational database theory over “real word set” is needed; in GrC, the real world model is Fifth GrC Model (relational GrC Model).

BGDM – Mining with Constraints

In this section, the solutions for the case $n = 2$ is illustrated: The following “real word” BGDM structures are added to K and K’ ; K’ is unchanged. But, binary relations B_{Amount} and B_{City} are defined on the two attribute domains of Relation K:

1. B_{Amount} is the smallest reflexive and symmetric binary relation that has the pair (TWENTY, TEN) as its member (Recall that a binary relation is a subset of the Cartesian Product of Domains.
2. B_{City} is the smallest reflexive and symmetric binary relation, in which the three cites, SC, MV, SJ are

Deductive Data Mining using Granular Computing. Table 3. Relational table K and K′

Relation K				Relation K′			
(*S*#	Amount	Birth month	CITY)	(*P*#	Weight	Part name	Material)
(S_1	TWENTY	MAR	SC)	(P_1	20	SCREW	STEEL)
(S_2	TEN	MAR	SJ)	(P_2	10	SCREW	BRASS)
(S_3	TEN	FEB	MV)	(P_3	10	NAIL	ALUMINUM)
(S_4	TEN	FEB	MV)	(P_4	10	NAIL	ALUMINUM)
(S_5	TWENTY	MAR	SJ)	(P_5	20	SCREW	BRASS)
(S_6	TWENTY	MAR	SJ)	(P_6	20	SCREW	BRASS)
(S_7	TWENTY	APR	SJ)	(P_7	20	PIN	BRASS)
(S_8	FIFTY	NOV	LA)	(P_8	300	HAMMER	ALLOY)
(S_9	FIFTY	NOV	LA)	(P_9	300	HAMMER	ALLOY)

considered to be very near to each other and very far away from LA.

Note that B_{Amount} and B_{City} induce two binary relations B_A and B_C on U (they form the $\mathcal{S}$ in previous discussions). Again K' has no changes. Relation K with two additional binary relations and Relation K define the following BGDM

1. BGDM of K is (U, $R_{BirthMonth}$, B_A, B_C).
2. BGDM of K' has no changes, namely, (U, R_{Weight}, $R_{PartName}$, $R_{Material}$),

where R_{A_j} denotes the equivalence relation induced by the attribute A_j. So from *BGDM point of view, the two relation K and K' are not isomorphic*. and hence the paradoxical phenomena disappear. This is a formal model for data mining with *constraints*.

Various GrC models provided various degree of "real world set" structures on attribute domains. GrC may provide a right direction to "real world" deductive data mining.

Cross-references

▶ Access Control
▶ Association Rules
▶ Data Mining
▶ Data Reduction
▶ Decision Tree Classification
▶ Decision Tree
▶ Frequent Itemsets and Association Rules
▶ Frequent Itemset Mining with Constraints
▶ Information Integration
▶ Rough Set Theory, Granular Computing on Partition
▶ Rough Set Theory (RST)
▶ Search Engine Metrics
▶ Topological Data Models

Recommended Reading

1. Dunham M. Data Mining Introduction and Advanced Topics. Prentice Hall, Englewood Cliffs, NJ, 2003.
2. Eugene Wong T.C. Chiang: canonical structure in attribute based file organization. Commun. ACM 14(9):563–597, 1971.
3. Fayad U.M., Piatetsky-Sjapiro G., and Smyth P. From data mining to knowledge discovery: an overview. In Knowledge Discovery in Databases, Fayard, Piatetsky-Sjapiro, Smyth, and Uthurusamy (eds.). AAAI/MIT Press, Cambridge, MA, 1996.
4. Ginsburg S. and Hull R. Ordered attribute domains in the relational model. XP2 Workshop on Relational Database Theory, 1981.
5. Ginsburg S. and Hull R. Order dependency in the relational model. Theor. Comput. Sci., 26:149–195, 1983.
6. Gracia-Molina H., Ullman J., and Windin J. Database Systems – The Complete Book. Prentice Hall, Englewood Cliffs, NJ, 2002.
7. Hsiao D.K., and Harary F. A formal system for information retrieval from files. Commun. ACM 13(2), 1970; Corrigenda, 13(4), April 1970.
8. Lee T.T. Algebraic theory of relational databases. The Bell System Tech. J., 62:(10)3159–3204, 1983.
9. Lin T.Y. Neighborhood systems and relational database. In Proc. 15th ACM Annual Conference on Computer Science, 1988, p. 725.
10. Lin T.Y. (1989) Chinese wall security policy – an aggressive model. In Proc. 5th Aerospace Computer Security Application Conf., 1989, pp. 286–293.
11. Lin T.Y. Neighborhood systems and approximation in database and knowledge base systems. In Proc. 4th Int. Symp. on Methodologies of Intelligent Systems (Poster Session), 1989, pp. 75–86.
12. Lin T.Y. (1992) Topological and fuzzy rough sets. In Decision Support by Experience – Application of the Rough Sets Theory, R. Slowinski (ed.). Kluwer, Dordecht, 1992, pp. 287–304.
13. Lin T.Y. Granular computing on binary relations: I. Da mining and neighborhood systems. I, II: Rough set representations and belief functions. In Rough Sets in Knowledge Discovery, A. Skoworn and L. Polkowski (eds.). Physica-Verlag, Wurzburg, 1998, pp. 107–140.
14. Lin T.Y. Data mining and machine oriented modeling: a granular computing approach. Appl. Intell. 13(2)113–124, 2000.
15. Lin T.Y. Attribute (feature) completion – the theory of attributes from data mining prospect. In Proc. 2002 IEEE Int. Conf. on Data Mining, 2002:282–289.
16. Lin T.Y. Chinese wall security policy models: information flows and confining Trojan horses. DBSec 2003:282–289.
17. Lin T.Y. Mining associations by linear inequalities. In Proc. 2004 IEEE Int. Conf. on Data Mining, 2004, pp. 154–161.
18. Lin T.Y. Granular computing: examples, intuitions and modeling, In Proc. IEEE Int. Conf. on Granular Computing, 2005, pp. 40–44.
19. Lin T.Y. A roadmap from rough set theory to granular computing, In Proc. 1st Int. Conf. Rough sets and Knowledge Theory, 2006, pp. 33–41.
20. Lin T.Y. and Chiang I.J. A simplicial complex, a hypergraph, structure in the latent semantic space of document clustering. Int. J. Approx. Reason. 40(1–2):55–80, 2005.
21. Lin T.Y. and Liau C.J. Granular computing and rough sets. Data Min. Knowl. Discov. Handbook 2005:535–561, 2004.
22. Louie E. and Lin T.Y. Finding association rules using fast bit computation: machine-oriented modeling. In Proc. 12th Int. Symp. Foundations of Inteligent Systems, 2000, pp. 486–494.
23. Pawlak Z. Rough sets. Theoretical aspects of reasoning about data. Kluwer, Dordecht, 1991.
24. Zadeh L.A. Fuzzy Sets Inf. Control 8(3):338–353, 1965.
25. Zadeh L.A. Some reflections on soft computing, granular computing and their roles in the conception, design and utilization of information/ intelligent systems. Soft Comput., 2:23–25, 1998.

Deductive Data Mining, Model for Automated Data Mining

▶ Deductive Data Mining using Granular Computing

Deductive Databases

▶ Datalog

Dedup

▶ Deduplication

Deduplication

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Synonyms

Dedup; Single Instancing

Definition

The term Deduplication refers to the task of eliminating redundant data in data storage so as to reduce the required capacity. The benefits of Deduplication include saving rack space and power consumption of the data storage. Deduplication is often implemented in archival storage systems such as content-addressable storage (CAS) systems and virtual tape libraries (VTLs). The term Deduplication is sometimes shortened to Dedup.

Key Points

An address mapping table and a hash index are often used for implementing Deduplication. The address mapping table converts a logical address to a physical location for each block, and the hash index converts a hash value to a physical location for each block. When a block X is to be written to the data storage, a hash value is calculated from the content of X and then the hash index is searched. If the same hash value is not found in the hash index, a new block is allocated in the storage space, X is written to the allocated block, and then the mapping table and the hash index are updated: a pair of the logical address and the physical location of X is registered to the mapping table, and a pair of the hash value and the physical location is also registered to the hash index. In contrast, if the same hash value is found, an identical block is supposed to exist in the storage. A pair of the logical address of X and the physical location retrieved from the hash index is merely registered to the mapping table, such that the block redundancy can be avoided. Note, above, that the possibility of hash collision is ignored. In reality there is the potential that the hash function produces the same hash value for different block contents, and in such rare cases, Deduplication would lose data in the data storage. To mitigate the issue, some vendors introduce hash algorithms which have a very low possibility of hash collisions. However, this solution cannot eliminate the potential of data loss completely. Others have chosen to check block contents every time the same hash value is found in the hash index. Such careful block checking may degrade archiving throughput, but can completely avoid data loss.

Deduplication techniques may generally operate at different granularities and at different implementation levels. For example, file systems may do Deduplication at the file level, and disk array controllers may do Deduplication at the sector level.

In enterprise systems, Deduplication is often implemented at the controller of archiving storage systems such as content-addressable storage (CAS) systems and virtual tape libraries (VTLs). Intrinsically, enterprise systems often store redundant data in the data storage. Suppose that an email message which attaches a document file is forwarded to many persons in the office. A separate mailbox of each recipient thus stores the same document file. Eliminating such duplication at archival storage systems sounds a natural and effective idea.

Deduplication is recognized as a technique of energy saving in data centers. Deduplication has the benefit of reducing the number of disk drives that are required to accommodate the explosively expanding data. Consequent effects of saving electric power are expected to help the improvement of system cost efficiency and the reduction of environmental impact.

Cross-references

▶ Information Lifecycle Management
▶ Storage Management
▶ Storage Power Management
▶ Storage Virtualization

Recommended Reading

1. Diligent Technologies. Hyper Factor: a breakthrough in data reduction technology. White Paper.
2. Patterson H. Dedupe-centric storage for general applications. White Paper, Data Domain.
3. Quinlan S. and Dorward S. Venti: a new approach to archival storage. In Proc. 1st USENIX Conf. on File and Storage Technologies, 2002, pp. 89–102.

Deduplication in Data Cleaning

Raghav Kaushik
Microsoft Research, Redmond, WA, USA

Synonyms

Reference reconciliation; Record matching; Merge-purge; Clustering

Definition

Many times, the same logical real world entity has multiple representations in a relation, due to data entry errors, varying conventions, and a variety of other reasons. For example, when Lisa purchases products from SuperMart twice, she might be entered as two different customers, e.g., `[Lisa Simpson, Seattle, WA, USA, 98025]` and `[Simson Lisa, Seattle, WA, United States, 98025]`. Such duplicated information can cause significant problems for the users of the data. For example, it can lead to increased direct mailing costs because several customers may be sent multiple catalogs. Or, such duplicates could cause incorrect results in analytic queries (say, the number of SuperMart customers in Seattle), and lead to erroneous data mining models. Hence, a significant amount of time and effort are spent on the task of detecting and eliminating duplicates.

This problem of detecting and eliminating duplicated data is referred to as deduplication. It is an important problem in the broader area of data cleaning. Note that this is markedly more challenging than the standard duplicate elimination problem for answering `select distinct` queries in relational database systems, which consider two tuples to be duplicates if they match exactly.

Historical Background

The problem of deduplication has been studied for a long time under application-specific settings such as library cataloging [17] and the Census Bureau [19]. Several techniques for measuring the similarity between records for these domains have been developed. For instance, the BigMatch technique [19] is used for deduplication by the US Census Bureau.

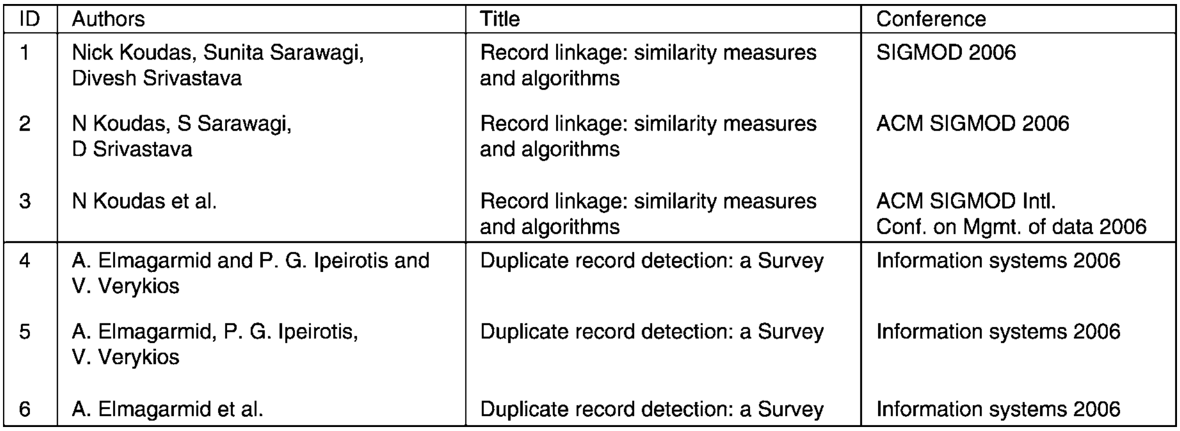
With the advent of data warehouses, data cleaning and deduplication in particular arose as a critical step in the Extract-Transform-Load (ETL) cycle of loading a data warehouse. Ensuring good quality data was essential for the data analysis and data mining steps that followed. Since the most common field of interest was the customer address field used for applications such as direct mailing, several tools such as Trillium [16] emerged that performed deduplication of addresses.

For the past several years, the database research community has addressed the deduplication problem [1,4,10,11,13]. This has been largely in a domain-independent context. The idea is to seek fundamental operations that can be customized to any specific application domain. This domain-independent approach has had a commercial presence. For instance, Microsoft SQL Server 2005 Integration Services supports data cleaning operators called *Fuzzy Lookup* and *Fuzzy Grouping* that respectively perform approximate matching and deduplication.

A new area of application for data cleaning technology and deduplication in particular has been with the data on the internet. There are web sites such as Citeseer, Google Scholar, Live Search Academic, Froogle and Live Products that integrate structured data such as citations and product catalogs from various sources of data and need to perform deduplication prior to hosting the web service.

Foundations

Deduplication can be loosely thought of as a fuzzy or approximate variant of the relational `select distinct` operation. It has as its input a table and a set of columns; the output is a partition of this table where each individual group denotes a set of records that are approximately equal on the specified columns. A large amount of information can be brought to bear in order to perform deduplication, namely the textual similarity between records, constraints that are expected to hold over clean data such as functional dependencies and attribute correlations that are known to exist. Techniques that use this wide variety of information have been explored in prior work. Figure 1 shows an example table containing citations



ID	Authors	Title	Conference
1	Nick Koudas, Sunita Sarawagi, Divesh Srivastava	Record linkage: similarity measures and algorithms	SIGMOD 2006
2	N Koudas, S Sarawagi, D Srivastava	Record linkage: similarity measures and algorithms	ACM SIGMOD 2006
3	N Koudas et al.	Record linkage: similarity measures and algorithms	ACM SIGMOD Intl. Conf. on Mgmt. of data 2006
4	A. Elmagarmid and P. G. Ipeirotis and V. Verykios	Duplicate record detection: a Survey	Information systems 2006
5	A. Elmagarmid, P. G. Ipeirotis, V. Verykios	Duplicate record detection: a Survey	Information systems 2006
6	A. Elmagarmid et al.	Duplicate record detection: a Survey	Information systems 2006

Deduplication in Data Cleaning. Figure 1. Example citation data and its deduplication.

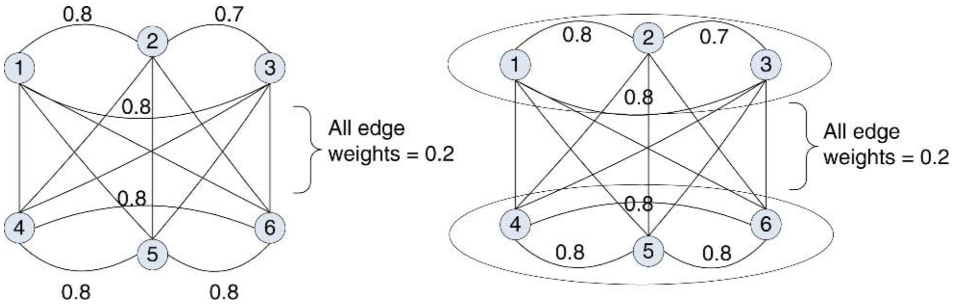
and a partition defined over all the textual columns, illustrating the output of deduplication.

At the core of the deduplication operation is a *similarity function* that measures the similarity or distance between a pair of records. It returns a similarity score which is typically a value between 0 and 1, a higher value indicating a larger similarity with 1 denoting equality. Example similarity functions are edit distance, jaccard similarity, Jaro-Winkler distance, hamming similarity and cosine similarity [12]. Given a table, the similarity function can be applied to all pairs of records to obtain a weighted similarity graph where the nodes are the tuples in the table and there is a weighted edge connecting each pair of nodes, the weight representing the similarity. Figure 2 shows the similarity graph for the data in Fig. 1 (the numbers inside the nodes correspond to the IDs). In practice, the complete graph is rarely computed since this involves a cross-product. Rather, only those edges whose weight is above a given threshold are materialized. There are well-known similarity join algorithms that perform this step efficiently as the data scales.

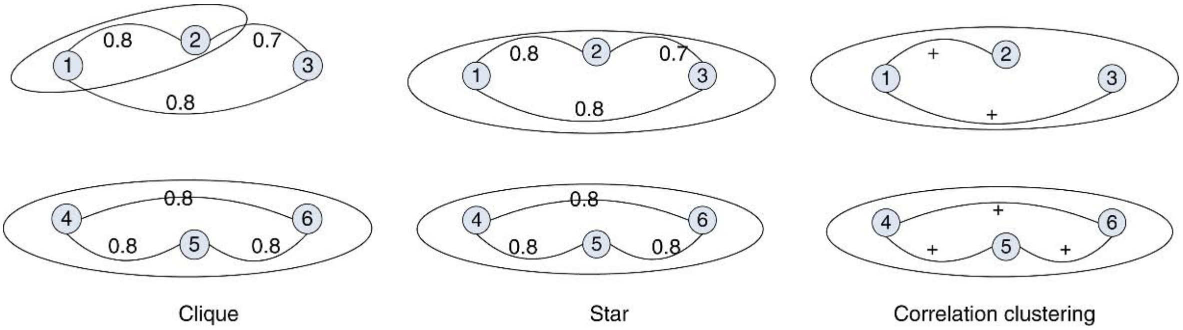
The deduplication operation is defined as a partition of the nodes in the similarity graph. Intuitively, one desires a partition where nodes that are connected with larger edge weights have a greater likelihood of being in the same group. Since similarity functions often do not satisfy properties such as triangle inequality, there are multiple ways of partitioning the similarity graph. Accordingly, several intuitive ways of partitioning the graph have been proposed in the literature. Note that this is closely related to the problem of clustering in data mining. Not surprisingly, a lot of the clustering algorithms such as K-means clustering [15] are also applicable for deduplication. While a comprehensive survey of these methods is beyond the scope of this entry, some of the key graph partitioning algorithms are reviewed.

In the *Single-Linkage Clustering* algorithm [14], at any point there is a partitioning of the nodes. At each step, the algorithm chooses a pair of groups to merge by measuring the similarity between all pairs of groups and picking the pair that has the highest similarity. The similarity between two groups of nodes is defined as the maximum similarity among all pairs of nodes, one from each group. The algorithm can be terminated in various ways, for example if there are a given number of clusters or if there are no more pairs of groups to merge which happens when all the connected components in the similarity graph have been collapsed. Figure 2 also shows the output of single-linkage when thresholds the similarity graph and retains only edges whose similarity is larger than or equal to 0.8. Notice that this partition corresponds to the one shown in Fig. 1.

One of the limitations of the single-linkage approach is that nodes two groups may be merged even if only one pair of nodes has a high similarity even if all other pairs do not. In order to address this limitation, an alternative is to decompose the graph into *cliques*. This ensures that whenever two tuples are collapsed, their similarity is large. Figure 3a shows the output of the clique decomposition for the similarity graph in Fig. 2, which only considers edges whose threshold is greater than or equal to 0.8. Since the similarity between nodes 2 and 3 is smaller than 0.8, the three nodes 1,2,3 cannot be collapsed as with single-linkage. Clique



Deduplication in Data Cleaning. Figure 2. Weighted similarity graph.



Deduplication in Data Cleaning. Figure 3. Example partitions.

partitioning can often be overly restrictive and leads to many pairs that have a large similarity getting split into multiple groups, as is the case with nodes 1 and 3 in Fig. 3a. Further, clique partitioning is known to be NP-hard.

To address this limitation, a relaxed version of the clique approach is proposed, namely *star* clustering [2]. A *star* is a pattern consisting of a single *center* node and a set of *satellite* nodes such that there is an edge with weight above a given threshold that connects the center to each of the satellite nodes. The intuition behind this choice is that for similarity functions that do satisfy the triangle inequality, the similarity between any pair of satellite nodes cannot be arbitrarily low. Aslam et al. [2] propose a greedy algorithm to compute a star partitioning of the input similarity graph. The graph is thresholded in that only edges with weight above a given threshold are retained. At any point in the algorithm, one operates with a current partitioning of the nodes. Nodes that are not assigned to any partition so far are deemed *uncovered*. At each step, one picks the uncovered node that such that the sum of the weights of its uncovered neighbors is the largest. This node and is uncovered neighbors define the next star and are grouped together. This process is repeated until no node is left uncovered. This algorithm has been empirically proven to be effective on several real-world datasets and is deployed commercially in the Fuzzy Groupby transform in Microsoft SQL Server 2005 Integration Services. Figure 3b shows the output of the star clustering algorithm over the data in Fig. 2.

One of the more recently proposed partitioning algorithms is *correlation clustering* [3]. Here, each edge in the similarity graph is viewed as a positive(+) edge when the similarity is high, say above a fixed threshold and as a negative(−) edge if it is not a positive edge. Figure 3c shows the similarity graph in which one views each edge with weight greater than or equal to 0.8 as a positive edge. Negative edges are not shown explicitly. The goal of correlation clustering is then to find a partition of the nodes that has the *least* disagreement with the pairwise edges. For a given partition of the nodes, the disagreement cost is the number of positive edges where the nodes at either end are split into different groups plus the number of negative edges where the nodes at either end are grouped together. While finding the optimal partition is NP-hard, efficient approximation algorithms have been proposed [3,7]. In fact, a simple one-pass randomized algorithm similar in spirit to the star-clustering algorithm can be shown to be a 3-approximation with high probability. The algorithm maintains a current partition of the nodes. At each step, it picks a random uncovered node. This node with all its uncovered positive neighbors define the next group to be merged. This process is repeated until no node is uncovered. Figure 3c shows the output of the correlation clustering algorithm on the data in Fig. 2.

Multi-Attribute Deduplication

Thus far, the entire discussion has focused on the computation of a single partition of the input table. However, one might often wish to partition the same input table or even multiple tables in a database in different ways. For example, the data in Fig. 1 could be partitioned by `author`, by `title` and by `conference`. These partitions of the data do not necessarily have to agree. One approach to computing these partitions is to separately run the algorithms described above for each of the chosen attributes. But clearly these attributes are not independent of one another. For example, the fact that the paper titles of papers 2 and 3 are grouped together provides additional evidence that the respective conference values must also be grouped together.

Ananthakrishna et al. [1] exploit this intuition in a multi-table setting where there is a hierarchical relationship among the different tables. For example, consider an address database with separate tables for nations, states, cities and street addresses. The idea is to proceed bottom-up in this hierarchy. The idea is to begin by deduplicating streets and then follow it up with cities, states and finally nations. At each step, the output of the previous step is used as contextual information for the current step. For example, if two states share many common cities, this provides further evidence that the two states must be collapsed.

The same intuition has also been explored in a more general setting where the data objects are arranged in a graph [9].

Constraints

A large body of work has focused on incorporating various constraints over the data into the deduplication process [3–5,8,18]. Five classes of work are identified in this overall space of incorporating constraints.

The first consists of constraints on individual tuples. These are constraints that express the condition that only some tuples (for instance "authors that have published in SIGMOD") may participate in the deduplication. Such constraints are easily enforced by pushing these filter conditions before deduplication is invoked.

The second consists of constraints that are in effect parameters input to the deduplication algorithm. For instance, the K-means clustering algorithm takes as input the number K that indicates the number of output groups desired. If the user of deduplication has some expectation for the number of groups returned after the data is partitioned, this may be used as a constraint during the deduplication itself.

Another class of constraints consists of positive and negative examples [5]. For example, for the data in Fig. 1, one might know that records 1 and 2 are indeed the same. Alternatively, one might also provide negative examples such as the fact that records 1 and 4 are *not* identical. These constraints may be used either as hard constraints to only consider partitions where the constraints are met, or to *adapt* the similarity function to accommodate the input examples.

The role of functional dependencies and inclusion dependencies in deduplication has also been explored [6]. The idea is to model the deduplication problem as a renaming problem where one is allowed to rename the attribute values of any record. At the end of the renaming, the partitions are defined by exact matching. Thus if the titles of records 1 and 2 in Fig. 1 are renamed to be the same, this signifies that these two titles have been collapsed. Every renaming has a *cost* defined by a distance function. The cost of a renaming is the distance between the new and old values. The input also consists of a set of functional and inclusion dependencies that are supposed to hold on the clean data. For example, one might require that `title` functionally determine `conference` for the data in Fig. 1. The goal is to compute the lowest cost renaming such that the output of the renaming satisfies all the constraints. Since this problem is NP-hard to solve optimally, efficient heuristics are proposed in [6].

Finally, the use of *groupwise* constraints for deduplication has also been investigated [18,8]. Here every *group* of tuples is expected to satisfy a given aggregate constraint. Such constraints arise when multiple sources of data are integrated. For instance, suppose one is integrating data from ACM and DBLP and that one is performing deduplication on the set of authors in the union of these sources. One constraint one might wish to assert on any subset of authors being considered for a collapse is that the set of papers authored in ACM and DBLP have a substantial overlap.

With an appropriate use of similarity function thresholds, constraints, and correlation attributes a user can guide the deduplication process to achieve the desired result.

Key Applications

See applications of data cleaning.

Cross-references

▶ Data Cleaning
▶ Data Integration
▶ Inconsistent Databases
▶ Probabilistic Databases
▶ Record Matching

Recommended Reading

1. Ananthakrishna R., Chaudhuri S., and Ganti V. Eliminating fuzzy duplicates in data warehouses. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
2. Aslam J.A., Pelehov K., and Rus D. A practical clustering algorithm for static and dynamic information organization. In Proc. 10th Annual ACM -SIAM Symp. on Discrete Algorithms, 1999.
3. Bansal N., Blum A., and Chawla S. Correlation clustering. Mach. Learn., 56(1–3):89–113, 2002.
4. Bhattacharya I. and Getoor L. Collective entity resolution in relational data. In Q. Bull. IEEE TC on Data Engineering, 2006.
5. Bilenko M., Basu S., and Mooney R.J. Integrating constraints and metric learning in semi-supervised clustering. In Proc. 21st Int. Conf. on Machine Learning, 2004.
6. Bohannon P., Fan W., Flaster M., and Rastogi R. A cost based model and effective heuristic for repairing constraints by value modification. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
7. Charikar M., Guruswami V., and Wirth A. Clustering with qualitative information. In J. Comp. and System Sciences, 2005.
8. Chaudhuri S., Sarma A., Ganti V., and Kaushik R. Leveraging aggregate constraints for deduplication. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 2007.
9. Dong X., Halevy A.Y., and Madhavan J. Reference reconciliation in complex information spaces. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
10. Fuxman A., Fazli E., and Miller R.J. ConQuer: efficient management of inconsistent databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
11. Galhardas H., Florescu D., Shasha D., Simon E., and Saita C. Declarative data cleaning: Language, model, and algorithms. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
12. Koudas N., Sarawagi S., and Srivastava D. Record linkage: similarity measures and algorithms. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006.
13. Sarawagi S. and Bhamidipaty A. Interactive deduplication using active learning. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002.
14. Single Linkage Clustering. http://en.wikipedia.org/wiki/Single_linkage_clustering.
15. The K-Means Clustering Algorithm. http://mathworld.wolfram.com/K-MeansClusteringAlgorithm.html.
16. Trillium Software. http://www.trilliumsoft.com/trilliumsoft.nsf.
17. Toney S. Cleanup and deduplication of an international bibliographic database. Inform. Tech. Lib., 11(1), 1992.
18. Tung A.K.H., Ng R.T., Lakshmanan L.V.S., and Han J. Constraint-based clustering in large databases. In Proc. 8th Int. Conf. on Database Theory, 2001.
19. Yancey W.E. Bigmatch: a program for extracting probable matches from a large file for record linkage. Statistical Research Report Series RRC2002/01, US Bureau of the Census, 2002.

Deep-Web Search

Kevin C. Chang
University of Illinois at Urbana-Champaign, Urbana, IL, USA

Synonyms

Hidden-Web search

Definition

With the proliferation of dynamic Web sites, whose contents are provided by online databases in response to querying, *deep-Web search* aims at finding information from this "hidden" or "deep" Web. Current search engines crawl and index pages from statically linked Web pages, or the "surface" Web. As such crawlers cannot effectively query online databases, much of the deep Web is not generally covered by current search engines, and thus remains invisible to users. A deep-Web search system queries over online databases to help users search these databases uniformly.

Historical Background

In the last few years, the Web has been rapidly "deepened" by the massive networked databases on the Internet. While the *surface Web* has linked billions of static HTML pages, a far more significant amount of information is hidden in the *deep Web*, behind the query forms of *searchable* databases. A July 2000 survey [1] claims that there were 500 billion hidden pages in 10^5 online sources, and an April 2004 study [3] reports 1.2 million query interfaces on the Web. Such information cannot be accessed directly through static URL links;

they are only available as responses to dynamic queries submitted through the query interface of a database. Because current crawlers cannot effectively query databases, such data is invisible to traditional search engines, and thus remain largely hidden from users.

With the proliferation of such dynamic sources, deep-Web search has become crucial for bridging users to the vast contents behind query forms. Unlike traditional search on the surface Web, searching over the deep Web requires integrating structured data sources at a large scale. While information integration has been investigated actively for decades, since 2000, *large scale information integration* has become the key issue for enabling deep-Web search: How to tackle the challenge of large scale? How to take advantage of large scale?

Foundations

Search Architecture

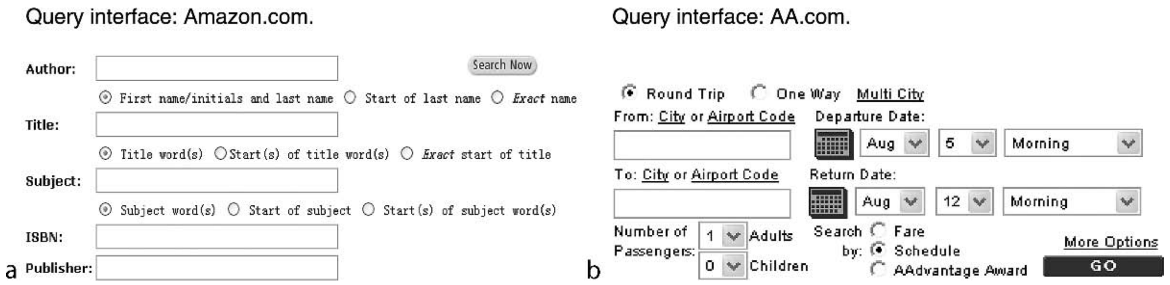
With the dynamic, structured nature of deep-Web data sources, depending on the application domains of focus, a deep-Web search system can take two forms of architecture:

1. *Crawl-and-Index*: A search system can crawl deep-Web sources regularly to build a local index for search, similar to what the current "surface-oriented" Web search engines do. This approach, by building a local index, will give fast online search performance but may risk returning stale data, if target sources are updated frequently. For comprehensive and up-to-date crawling from data sources offline to retrieve their structured data objects, it will require special "structure-aware" crawling techniques that can deal with query-driven sources.
2. *Discovery-and-Forward*: A search engine can automatically discover databases from the Web, by crawling and indexing their query interfaces (and not their data pages), and forward user queries to the right sources for the actual search of data, upon user querying. This approach, with its on-the-fly querying, gives up-to-date results, while there may be increased query latency online. As a key challenge, it needs to be able to interact with data sources to execute user queries.

Source Modeling

How does one to find sources and model what they are about? In building a deep-Web search system, one first needs to determine the set of sources to integrate, either by crawling offline or querying online. With the large scale of the Web, where numerous sources are scattered everywhere, it is necessary to crawl the Web to discover and model their query interfaces. After crawling finds a query interface, its query attributes and capabilities must be extracted. The objective of this *form extraction*, which extracts the structure of query forms, is to provide a "model" of how a source is queried, so that the search system can interact with the source.

Guarding data behind them, such query interfaces are the "entrance" to the deep Web. These interfaces, usually in HTML *query forms*, express *query conditions* for accessing objects from databases behind. Each condition, in general, specifies an *attribute*, one or more supported *operators* (or modifiers), and a *domain* of allowed values. A condition is thus a three-tuple [`attribute`; operators; domain], e.g., [`author`; {`"first name..."`, `"start..."`, `"exact name"`}; `text`] in interface shown in Fig. 1a. Users can then use the condition to formulate a specific constraint (e.g., [`author = "tom clancy"`] by selecting an operator (e.g., `"exact name"`) and filling in a value (e.g., `"tom clancy"`). For modeling and integrating Web databases, the very first step is to "understand" what a query interface says – i.e., what *query capabilities* a source supports


a Query interface: Amazon.com.

Author:
● First name/initials and last name ○ Start of last name ○ *Exact* name
Title:
● Title word(s) ○ Start(s) of title word(s) ○ *Exact* start of title
Subject:
● Subject word(s) ○ Start of subject ○ Start(s) of subject word(s)
ISBN:
Publisher:

b Query interface: AA.com.

● Round Trip ○ One Way Multi City
From: City or Airport Code
Departure Date: Aug 5 Morning
To: City or Airport Code
Return Date: Aug 12 Morning
Number of Passengers: 1 Adults 0 Children
Search by: ○ Fare ● Schedule ○ AAdvantage Award

Deep-Web Search. Figure 1. Query interfaces examples.

through its interface, in terms of specifiable conditions. For instance, Amazon.com (Fig. 1a) supports a set of five conditions (on `author`, `title`, ..., `publisher`). Such query conditions establish the *schema model* underlying the Web query interface. These extracted schema model, each of which representing the query capabilities of a source, can then be used for determining *whether* a source is relevant to the desired applications or user queries and *how* to query the source.

There are two main approaches for form extraction: The first approach, *grammar-based parsing* [9], observes that there are common visual patterns in the layout of query forms, abstracts these patterns as grammar production rules, and executes a "best-effort parsing" to extract the form structure. That is, given the set of grammar rules, it assembles the 'best-possible" parse of the query form. In this approach, it is shown that a small set of (20–30) patterns can already capture common layouts across different subject domains (books, airlines, *etc.*) of query forms.

The second approach, *classifier-based recognition*, applies learning-based or rule-based classifiers to determine the role of each form element, to formulate an overall interpretation of the form (e.g., [4]). It can first identify, for each input element (e.g., a text input box), a set of text elements as the candidates of its labels. Based on the candidates for each input element, the process then goes on to form a global assignment, which will reconcile conflicts between the candidates. The recognition of candidates can be based on features such as the proximity between elements (e.g., what are the closest text elements to this input element?) or the content of text elements (e.g., can one classify this text element as likely a label, e.g., "author"?).

Schema Matching

Schema matching, for corresponding attributes at different sources, is crucial for translating queries (as will be discussed later) to search these sources. It has been a central problem in information integration, and thus also a key challenge for deep-Web search. Because one is dealing with querying Web sources, such matching takes query forms (as extracted by source modeling) as input schemas (where attributes are specified for querying) and attempts to unify them by finding their correspondence (or by matching them to a common mediated schema). While a classic problem, however, it takes a new form in this context, with both new challenges and new insights.

As the new challenge, for searching the deep Web, in many application scenarios, such matching must deal with *large scale*, to match numerous sources (e.g., many airfare sources; book sources). Traditional schema matching targets at finding attribute correspondence between a pair of sources. Such pairwise matching, by tackling two sources at a time, does not directly handle a large number of data sources.

On the other hand, however, the new setting also hints on new approaches that can exploit the scale. Given there are many sources on the Web, instead of focusing on each pair of sources, one can observe the schemas of all the sources together, to take advantage of the "holistic" information observable only across many sources. Such holistic methods can, in particular, explore the "regularity" existing among Web sources. Note that – since one is dealing with query forms for schema matching – these query forms are "external" schemas, which are designed for users to easily understand. Unlike "internal" schemas (say, for database tables), which may use arbitrary variable names (e.g., "au_name"), these external schemas tend to use common terms (e.g., "author") as the names of attributes. As conventions naturally emerge, it has been observed [2] that certain regularities exist, in particular, for what attributes are likely to appear and what names they may be labeled with.

There are several approaches that leverage this new insight of *holistic* schema matching, to exploit the regularities across multiple sources. First, the approach of *model discovery* [2]: Schema matching may be considered as discovering a hidden generative model that dictates the occurrences of attribute names (such as when to use "author" versus "first name" and "last name"). Such a model effectively captures how attribute terms would occur – some attributes are *synonyms* that will not be used together (e.g., "author" and "name"), and some attributes are usually *grouped* with each other (e.g., "first name" and "last name"). With this conceptual view, the schema matching problem is transformed into finding an underlying model that is statistically consistent with the input schemas (to be matched). Various statistical methods can then be used for this model discovery, such as hypothesis testing or correlation mining.

Second, the approach of *attribute clustering*: In this view, one considers finding synonym attributes as clustering of those attributes into equivalent classes [8]. Given a set of query forms, where attributes are extracted with a hierarchy that indicates how they are related to each other in a form, this approach clusters attributes from different sources into a hierarchy of clusters that represent their synonym and grouping relationships. The clustering process will exploit the hierarchical relationship extracted from each individual query form as well as the similarities between attribute labels.

Furthermore, another new insight can be observed that, in the context of the deep Web, since one is dealing with matching between query interfaces, it is possible to actually try out querying, or *probing*, each source to derive their attribute correspondence. Exploiting the dynamic response of query forms has been explored for schema matching [6].

Offline Crawling

In the Crawl-and-Index architecture, like a typical search engine, data objects can be collected from various sources, index them in a local database, and provide search from the database. To realize this search framework, one needs to crawl data offline (before user searches) from various sources. The objective of this *deep Web crawling* is to collect data, as exhaustively as possible, from each source to provide comprehensive coverage. As such sources are accessed by querying through their query forms, a deep Web "crawler" [5] thus, for each source, repeatedly submitting queries and collect data from query results. There are two issues for realizing this crawling:

First, *comprehensiveness*: How to formulate queries so that the results cover as much as possible the data at the source? Each query is a way to fill in the query form of the source. For those query attributes that have a clearly defined set of input values (e.g., a selection-box of two choices {paperback, hardcover} for book format), each value can be enumerated in querying. For attributes with an open set of input values, such as a text input that takes any keywords, the crawler will need to be driven by certain "domain knowledge" that can generate a set of keyword terms relevant to the application, e.g., {computer, internet, web, databases, . . .}.

Second, *efficiency*: How to minimize the number of queries to submit to a source for crawling its content? Since different queries may retrieve overlapping results, it is possible to choose different sets of queries for crawling. For efficiency, one wants to minimize the cost of querying, such as the number of queries submitted. The problem is thus, as each data object can be reached by multiple queries, to select the smallest set of queries that together *reach* all the data objects [7].

Online Querying

In the Discover-and-Forward architecture, user queries are directed to search relevant sources, at the time of querying. Unlike offline crawling, which aims at collecting *all* the data from each source by *any* query (subject to cost minimization), this online querying must execute a specific user query at run time. Given a query (in some unified front-end language), for each source, one needs to translate the query for the source, to fill in its query form. This *query translation* will rewrite the original query into a new format that satisfies two criteria: First, it is executable at the source: The new query must contain only query conditions and operators that the source can accept in its query form. Second, it is a good approximation of the original query: The translated query, if not "equivalent" to the original query, should be as close to it as possible. In terms of techniques, query translation (e.g., [10]) can be driven by either generic type hierarchies (e.g., how to transform commonly-used data types such as date and number) or specialized domain knowledge.

Key Applications

- Enhancing General Web Search: Current search engines have started to combine specialized search responses for certain categories of queries (e.g., returning stock information for search of company names).
- Enabling Vertical Web Search: Searching data on the Web in specific domains, such as travel or real estate.

Cross-references

▶ Information Integration
▶ Query Translation
▶ Schema Matching

Recommended Reading

1. BrightPlanet.com. The Deep Web: Surfacing Hidden Value. http://www.brightplanet.com/resources/details/deepweb.html, 2000.
2. He B. and Chang K.C.C. Statistical Schema Matching across Web Query Interfaces. In Proc. ACM SIGMOD Int. Conf. on Management of Data,, 2003, pp. 217–228.
3. He H., Meng W., Lu Y., Yu C.T., and Wu Z. Towards deeper understanding of the search interfaces of the deep Web. In Proc. 16th Int. World Wide Web Conference, 2007, pp. 133–155.
4. He B., Patel M., Zhang Z., and Chang K.C.C. Accessing the deep Web: a survey. Commun ACM, 50(5): 94–101, 2007.
5. Raghavan S. and Garcia-Molina H. Crawling the hidden Web. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 129–138.

6. Wang J., Wen J.R., Lochovsky F.H., and Ma W.Y. Instance-based schema matching for web databases by domain-specific query probing. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 408–419.
7. Wu P., Wen J.R., Liu H., and Ma W.Y. Query selection techniques for efficient crawling of structured Web sources. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 47.
8. Wu W., Yu C.T., Doan A., and Meng W. An interactive clustering-based approach to integrating sourc query interfaces on the deep Web. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 95–106.
9. Zhang Z., He B., and Chang K.C.C. Understanding Web query interfaces: best-effort parsing with hidden syntax. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 117–118.
10. Zhang Z., He B., and Chang K.C.C. Light-weight domain-based form assistant: querying web databases on the fly. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 97–108.

Degrees of Consistency

▶ SQL Isolation Levels

DEMs

▶ Digital Elevation Models

Degrees of Cosistency

▶ SQL Isolation Levels

Dendrogram

▶ Visualizing Clustering Results

Dense Index

Mirella M. Moro[1], Vassilis J. Tsotras[2]
[1]Federal University of Rio Grande do Sul, Porte Alegre, Brazil
[2]University of California-Riverside, Riverside, CA, USA

Definition

Consider a tree-based index on some numeric attribute *A* of a relation *R*. This index is called *dense* if every search-key value of attribute *A* in relation *R* also appears in the index. Hence for every search-key value *x* in *A*, there is an index record of the form <*x*, *pointer*>, where *pointer* points to the first record (if many such exist) in relation *R* that has *R.A* = *x*.

Key Points

Tree-based indices are built on numeric attributes and maintain an order among the indexed search-key values. Hence, they provide efficient access to the records of a relation by attribute value. Consider for example an index built on attribute *A* of relation *R*. The leaf pages of the index contain *index-records* of the form <*search-key, pointer*>, where search-key corresponds to a value from the indexed attribute *A* and *pointer* points to the respective record in the indexed relation *R* with that attribute value. If all distinct values that appear in *R.A* also appear in index records, this index is *dense*, otherwise it is called *sparse*. If there are many records in relation *R* that have *R.A* = *x*, then the index record <*x*, *pointer*> points to the first of these records. If relation *R* is ordered on attribute *A*, the rest of these records are immediately following after the first accessed record (in the same page of the relation file). If relation *R* is not ordered according to attribute *A*, the remaining records can be accessed from a list of pointers after this first record.

Tree-indices are further categorized by whether their search-key ordering is the same with the relation file's physical order (if any). Note that a relation file may or may not be ordered. For example, if a relation *R* is ordered according to the values on the *A* attribute, the values in the other attributes will not be in order. If the search-key of a tree-based index is the same as the ordering attribute of a (ordered) file then the index is called *primary*. An index built on any non-ordering attribute of a file is called *secondary*. If an index is secondary then it should also be dense. Since the index is secondary, the relation is not ordered according to the search-key value of this index. As a result, finding a given record in the file by this index requires that all search-key values of the indexed attribute are present in the index, i.e., it is a dense index.

A dense index is typically larger than a sparse index (since all search-key values are indexed) and thus requires more space. It also needs to be updated for every relation update that involves the attribute value being indexed.

Cross-references

- ▶ Access Methods
- ▶ B+-Tree
- ▶ Indexing
- ▶ Index Sequential Access Method (ISAM)

Recommended Reading

1. Elmasri R. and Navathe S.B. Fundamentals of Database Systems, 5th edn. Addisson-Wesley, Boston, MA, 2007.
2. Manolopoulos Y., Theodoridis Y., and Tsotras V.J. Advanced Database Indexing. Kluwer, Dordecht, 1999.
3. Silberschatz A., Korth H.F., and Sudarshan S. Database System Concepts, 5th edn. McGraw-Hill, New York, 2006.

Dense Pixel Displays

Daniel A. Keim, Peter Bak, Matthias Schäfer
University of Konstanz, Konstanz, Germany

Synonyms

Data visualization; Information displays; Pixel oriented visualization techniques; Visual data exploration; Information visualization; Visualizing large data sets; Visualizing multidimensional and multivariate data; Visual data mining

Definition

Dense Pixel Displays are a visual data exploration technique. Data exploration aims at analyzing large amounts of multidimensional data for detecting patterns and extracting hidden information. Human involvement is indispensable to carry out such a task, since human's powerful perceptual abilities and domain knowledge are essential for defining interesting patterns and interpreting findings. Dense pixel displays support this task by an adequate visual representation of as much information as possible while avoiding aggregation of data-values. Data is shown using every pixel of the display for representing one data point. Attributes of the data are mapped in separate sub-windows of the display, leaving one attribute for one sub-window. The data point's value for an attribute is mapped to the pixel's color. The arrangement of pixels in sub-windows is determined by the mapping of the data-sequence into a two dimensional space preserving the pixel's neighborhood relations.

A number of different pixel-oriented visualization techniques have been proposed in recent years and shown to be useful for visually exploring data in many applications. These techniques differ mainly in their approach to arrange individual pixels in relation to each other, and in their choice of shaping the sub-windows to make maximal use of space.

Historical Background

Scientific, engineering and environmental databases containing automatically collected data have grown to an overwhelming size, which need to be analyzed and explored. The desire to augment human's perceptual abilities and understand multidimensional data inspired many scientists to develop visualization methodologies. The progress in information technology offers more and more options to visualize data, in particular multidimensional data. Detecting information in low dimensions (1D, 2D, 3D) by taking advantage of human's powerful perceptual abilities is crucial to understand the higher dimensional data set. This assumption is fundamental in all visualization techniques.

One important thing in analyzing and exploring large amounts of data is to visualize the data in a way that supports the human. Visual representations of the data are especially useful for supporting a quick analysis of large amounts of multi-modal information, providing the possibility of focusing on minor effects while ignoring known regular features of the data. Visualization of data which have some inherent two- or three-dimensional semantics has been done even before computers were used to create visualizations. In the well-known books [13,14], Edward R. Tufte provides many examples of visualization techniques that have been used for many years. Since computers are used to create visualizations, many novel visualization techniques have been developed by researchers working in the graphics field. Visualization of large amounts of arbitrary multidimensional data, however, is a fairly new research area. Early approaches are scatterplot matrices and coplots [5], Chernoff faces [4], parallel coordinates [6], and others [1,3]. Researchers in the graphics/visualization area are extending these techniques to be useful for large data sets, as well as developing new techniques and testing them in different application domains. Newer approaches for dense pixel displays are developed for several applications for visualizing data, i.e., geospatial data [10], financial data [9,15], text [11], and neuroscience data [12].

Foundations

Dense Pixel Displays as Optimization Problem – Pixel Arrangement

One of the first questions in creating a dense pixel display is how the pixels are arranged within each of the subwindows. Therefore, only a good arrangement will allow a discovery of clusters and correlations among the attributes. It is reasonable to distinguish between data sets where a natural ordering of the items is given, such as in time-series data or a production-line, and between data sets without inherent ordering, such as results of search-queries. In all cases, the arrangement has to consider the preservation of the distance of the one-dimensional ordering in the two-dimensional arrangement.

Mappings of ordered one-dimensional data sets to two dimensions have been attracted the attention of mathematicians already long before computer came into existence.

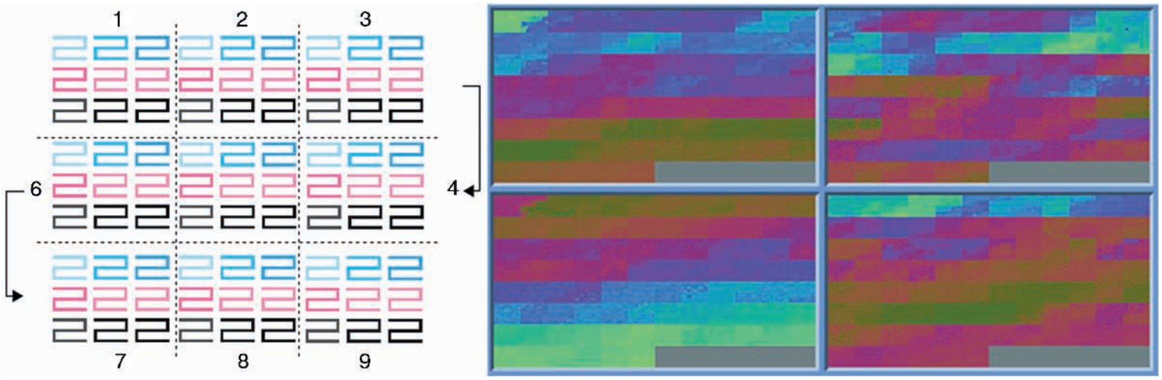
So called space-filling curves attempt to solve this problem. The Peano-Hilbert curve provides one possible solution. However, with this method it is often difficult to follow the flow of items. The Morton curve is an alternative proposed to overcome this problem, which has a simpler regularity.

The general idea of Recursive Pattern, as first proposed in [8] is based on the idea of a line and column-wise arrangement of the pixels. If simple left-right or top-down arrangements are used on pixel level, the resulting visualizations do not provide useful results, since it creates random neighborhoods that bias user's perception of correlations and clusters.

One possibility to improve the visualizations is to organize the pixels in small groups and arrange the groups to form some global pattern, as suggested by recursive pattern visualizations. The basic idea of recursive pattern visualization technique is based on a scheme which allows lower-level patterns to be used as building blocks for higher-level patterns. In the simplest case the patterns for all recursion levels are identical. In many cases, however, the data has some inherent structure which should be reflected by the pattern of the visualization. In order to create an appropriate visualization of patterns in a recursive manner, the algorithm must allow users to provide parameters for defining the structure for the low and high recursion levels.

Therefore, one major difference between the "recursive pattern" technique and other visualization techniques is that it is based on a generic algorithm allowing the users to control the arrangement of pixels. By specifying the height and width for each of the recursion levels, users may adapt the generated visualizations to their specific needs. This allows the "recursive pattern" technique to be used for a wide range of tasks and applications.

A schematic example for a highly structured arrangement is provided in the Fig. 1 The visualization shows the financial data of four assets for eight years. The subspaces are subdivided into different time intervals. A sub-window represent the whole time period, which is further divided into single years, months, weeks and days. Since days represent the lowest recursion level, they correspond to one single pixel (as schematically shown in Fig. 1 Left). This structure



Dense Pixel Displays. Figure 1. Left: Schematic representation of a highly structured arrangement. Right: Recursive arrangement of financial data (September 1987–February 1995: above left IBM, above right DOLLAR, below left: DOW JONES, below right GOLD) (adopted from [8]).

is preserved in the visual representation of the dense pixel display allowing fast extraction detailed information (Fig. 1 Right). One of the tasks that would benefit from such a representation is detecting a strong correlation for example between IBM stock-prices and the dollar's exchange rate in the first 2.5 years.

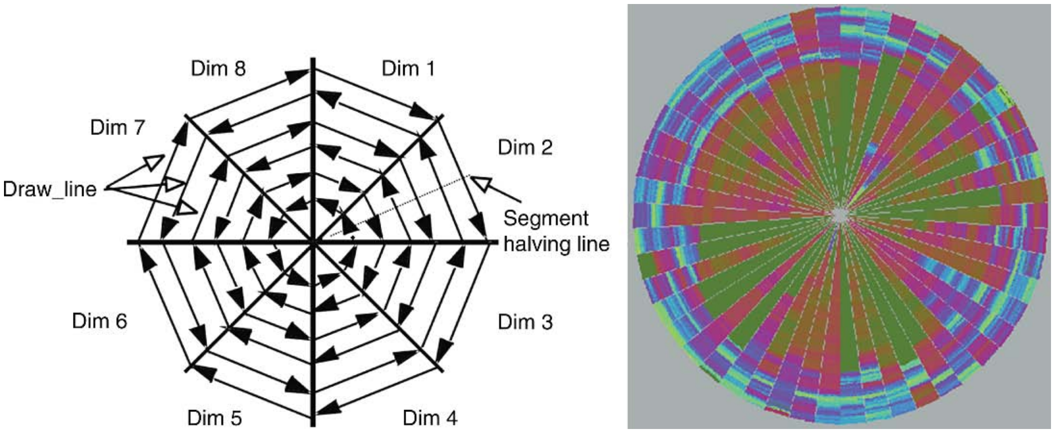
Dense Pixel Displays as Optimization Problem – Shape of Sub-windows

The second important question concerns how the shapes inside the sub-windows should be designed. Sub-windows are used to represent single attributes of the dataset. In the optimization functions used in the past, the distance between the pixel-attributes belonging to the same data object was not taken into account. This, however, is necessary in order to find appropriate shapes for the sub-windows. In many cases a rectangular partitioning of the display was suggested. The rectangular shape of the sub-windows allows a good screen usage, but at the same time it leads to a dispersal of pixels belonging to one data object over the whole screen, especially for data sets with many attributes. Because the sub-windows for the attributes are rather far apart, it might be difficult to detect correlating patterns.

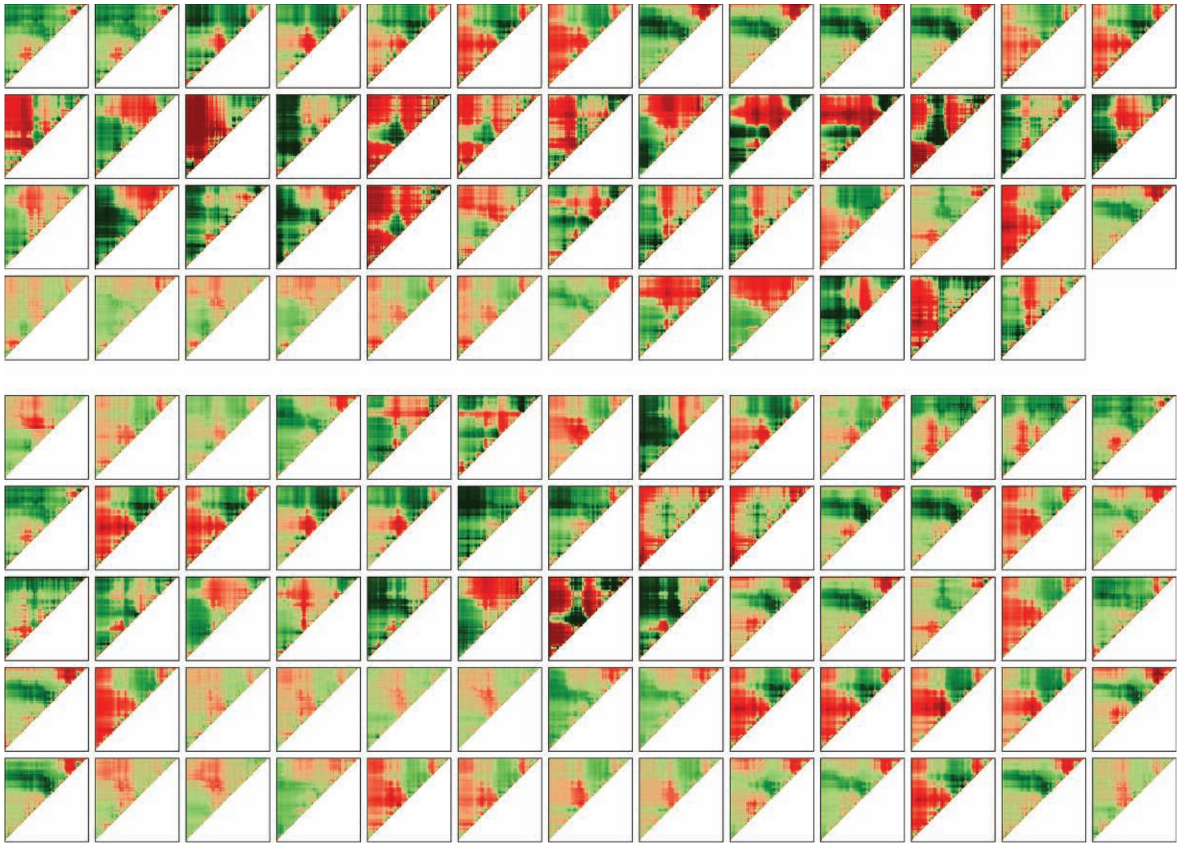
An idea for an alternative shape of the sub-windows is the circle segments technique. The technique tries to minimize the average distance between the pixels that belong to the dimensions of one data object.

The fundamental idea of the "circle segments" visualization technique is to display the data dimensions as segments of a circle (as shown in Fig. 2). If the data consists of k attribute-dimensions, the circle is partitioned into k segments, each representing one dimension. The data items within one segment are arranged in a back and forth manner orthogonal to the line that halves the two border lines of the segment (cf. Fig. 2). The drawing starts in the centre of the circle and draws the pixels from one border line of the segment to the other. Whenever the drawing hits one of the border lines, it is moved towards the outside of the circle by changing the direction. When the segment is filled out for one attribute, the next segment will start, until all dimensions are visualized. A schematic representation is shown in Fig. 2 (left). The results for such a representation is shown for 50 stock prices from the Frankfurt stock index, representing about 265,000 data values (Fig. 2 Right). The main advantage of this technique is that the overall representation of the whole data set is better perceivable – including potential dependencies, analogies, and correlations between the dimensions.

Another arrangement of sub-windows is the arrangement as bar chart, and the resulting technique is called Pixelbarcharts (see Fig. 4). The basic idea of Pixelbarcharts is to use the intuitive and widely used presentation paradigm of bar charts, but also use the available screen space to present more detailed information. By coloring the pixels within the bars according to the values of the data records, very large amounts of data can be presented to the user. To make the display more meaningful, two parameters of the data records are used to impose an ordering on the pixels in the



Dense Pixel Displays. Figure 2. Left: Circle segment technique schema for eight attribute-dimensions. Right: Twenty years of daily data of the FAZ Index (January 1974–April 1995) (adopted from [1]).



Dense Pixel Displays. Figure 3. The upper image shows 51 funds of the same financial institution and their performance over time compared to the market. Periods of time where a fund performed well compared to other funds are colored in green, and where it performed worse than others in red. The lower image shows 65 funds of a different financial institution. A comparative visual analysis reveals that the funds in the second image are more continuous and do not have as many red areas.

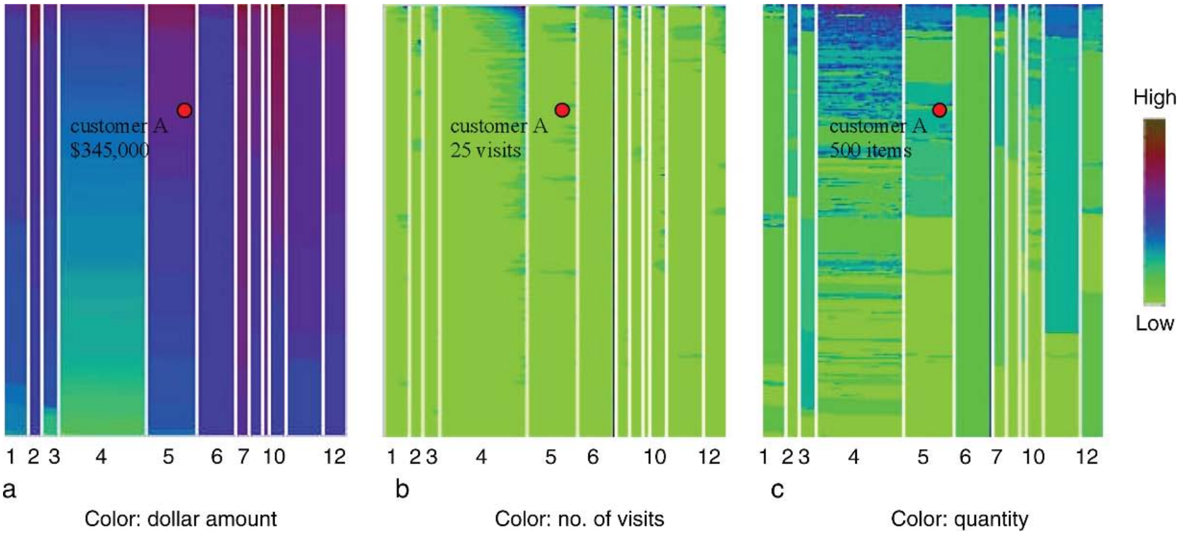
x- and y-directions. Pixel bar charts can be seen as a generalization of bar charts. They combine the general idea of xy-plots and bar charts to allow an overlap-free, non-aggregated display of multi-attribute data.

Dense Pixel Displays as Optimization Problem – Ordering of Dimensions

The final question to consider is the ordering of attributes. This problem is actually not just a problem of pixel-oriented techniques, but a more general problem which arises for a number of other techniques such as the parallel coordinates technique. The idea is that the data attributes have to be positioned in some one- or two-dimensional ordering on the screen, and this is usually done more or less by chance – namely in the order in which the dimensions happen to appear in the data set. The ordering of dimensions, however, has a major impact on the expressiveness of the visualization. One solution for finding an effective order of dimensions is to arrange the dimensions according to their similarity. For this purpose, the technique has to define measures which determine the similarity of two dimensions. Data mining algorithms, such as regression models or clustering tools, but also statistical approaches, such as inter-correlation matrixes, can provide solutions for such a problem.

Key Applications

Dense Pixel Displays have a wide range of applications. They are especially important for very large high dimensional data sets, which occur in a number of applications. Example applications include financial data analysis like



Dense Pixel Displays. Figure 4. Illustrates an example of a multi-pixel bar chart of 405,000 multi-attribute web sales transactions. The dividing attribute is product type; the ordering attributes are number of visits and dollar amount. The colors in the different bar charts represent the attributes dollar amount, number of visits, and quantity (adopted from[7]).

the “Growth Matrix” [9], business data analysis like “Pixelbarcharts” [7], text analysis like “literature fingerprinting” [11] and geospatial analysis like “PixelMaps” [10]. The following sections show a few examples of the techniques.

Growth Matrix

The “Growth Matrix” [9] is a pixel-based visualization technique that displays the return rates of assets for all possible combinations of time intervals between point of purchase and point of sale in one single view. It creates unique triangular fingerprints of the financial performance of assets, and allows a visual comparative analysis of large amounts of assets at the same time. In addition to that, it is also possible to compare the ranking of assets compared to all assets on the market for all time intervals, which allows a complete overview on the real performance of an investment. Figure 3 shows an example of this visualization technique.

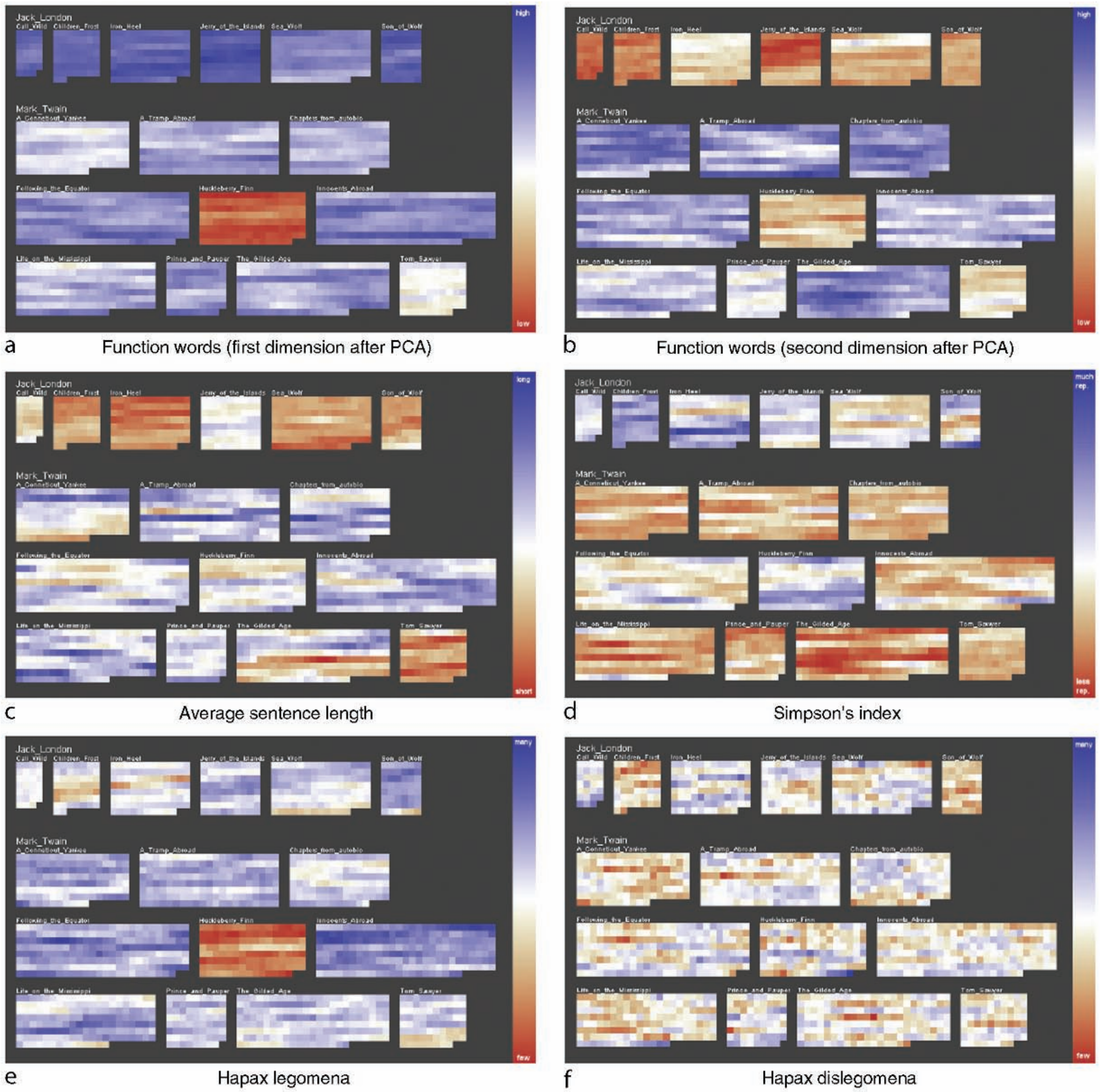
Pixelbarcharts

Pixelbarcharts retain the intuitiveness of traditional bar charts while allowing very large data sets to be visualized in an effective way. For an effective pixel placement, it solves the complex optimization problem. In Fig. 4 the following facts can be observed:

- Product type 10 and product type 7 have the top dollar amount customers (dark colors of bar 7 and 10 in Fig. 4a).
- The dollar amount spent and the number of visits are clearly correlated, especially for product type 4 (linear increase of dark colors at the top of bar 4 in Fig. 4b).
- Product types 4 and 11 have the highest quantities sold (dark colors of bar 4 and 11 in Fig. 4 c).
- By clicking on a specific pixel (A), one may find out that customer A visited 25 times, bought 500 items, and spent $345,000 on product type 5.

Literature Fingerprinting

In computer-based literary analysis different types of features are used to characterize a text. Usually, only a single feature value or vector is calculated for the whole text. “Literature fingerprinting” combines automatic literature analysis methods with an effective visualization technique to analyze the behavior of the feature values across the text. For an interactive visual analysis, a sequence of feature values per text is calculated and presented to the user as a characteristic fingerprint. The feature values may be calculated on different hierarchy levels, allowing the analysis to be done on



Dense Pixel Displays. Figure 5. Analysis of the discrimination power of several text measures for authorship attribution. Each pixel represents a text block and the pixels are grouped into books. Color is mapped to the feature value, e.g., in c) to the average sentence length. If a measure is able to discriminate between the two authors, the books in the first line (that have been written by J. London) are visually set apart from the remaining books (written by M. Twain). This is true for example for the measure that was used in c) and d) but not for the measure that was used to generate f). Outliers (such as the book Huckleberry Finn in c)) stick out immediately. The technique allows a detailed analysis of the development of the values across the text (adopted from [11]).

different resolution levels. Figure 5 gives an impression of the technique.

Cross-references

- ▶ Data Visualization
- ▶ Text Visualization
- ▶ Visual Analytics
- ▶ Visual Data Mining
- ▶ Visualizing Categorical Data
- ▶ Visualizing Clustering Results
- ▶ Visualizing Hierarchical Data
- ▶ Visualizing Network Data
- ▶ Visualizing Quantitative Data

Recommended Reading

1. Anderson E. A semigraphical method for the analysis of complex problems. In Proc. Nat. Acad. Sci. USA, 13:923–927, 1957.
2. Ankerst M., Keim D.A., and Kriegel H.-P. Circle segments: a technique for visually exploring large multidimensional data sets. In Proc. IEEE Symp. on Visualization, 1996.
3. Brissom D. Hypergraphics: Visualizing Complex Relationships in Art, Science and Technology (AAAS Selected Symposium; 24). Westview Press, 1979.
4. Chernoff H. The use of faces to represent points in k-dimensional space graphically. J. Am. Stat. Assoc., 68(342):361–368, 1973.
5. Cleveland W.S. Visualizing Data. Hobart Press, Summit, NJ, 1993.
6. Inselberg A. N-Dimensional Graphics Part I: Lines and Hyperplanes, IBM LA Science Center Report, # G320–2711, 1981.
7. Keim D.A., Hao M.C., Dayal U., and Hsu M. Pixel bar charts: a visualization technique for very large multi-attribute data sets. Inf. Visualization, 1(1):20–34, 2001.
8. Keim D.A., Kriegel H.-P., and Ankerst M. Recursive pattern: a technique for visualizing very large amounts of data. In Proc. IEEE Symp. on Visualization, 1995, pp. 279–286.
9. Keim D.A., Nietzschmann T., Schelwies N., Schneidewind J., Schreck T. and Zeigler H. A Spectral Visualization System for Analyzing Financial Time Series Data. In Euro Vis 2006: Eurographics/IEEE-VGTC Symposium on Visualization, 2006, pp. 195–202.
10. Keim D.A., North S.C., Panse C., and Sips M. PixelMaps: a new visual data mining approach for analyzing large spatial data sets. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 565–568.
11. Keim D.A. and Oelke D. Literature fingerprinting: a new method for visual literary analysis. In Proc. IEEE Symp. on Visual Analytics and Technology, 2007.
12. Langton J.T., Prinz A.A., Wittenberg D.K., and Hickey T.J. Leveraging layout with dimensional stacking and pixelization to facilitate feature discovery and directed queries. In Proc. Visual Information Expert Workshop, 2006.
13. Tufte E.R. The Visual Display of Quantitative Information. Graphics Press, Cheshire, CT, 1983.
14. Tufte E.R. Envisioning Information. Graphics Press, Cheshire, CT, 1990.
15. Ziegler H., Nietzschmann T., Keim D.A. Relevance driven visualization of financial performance measures. In Proc. Eurographics/IEEE-VGTC Symposium on Visualization, 2007, pp. 19–26.

Density-based Clustering

MARTIN ESTER
Simon Fraser University, Burnaby, BC, Canada

Definition

Density-based clusters are dense areas in the data space separated from each other by sparser areas. Furthermore, the density within the areas of noise is lower than the density in any of the clusters. Formalizing this intuition, for each *core point* the neighborhood of radius *Eps* has to contain at least *MinPts* points, i.e., the density in the neighborhood has to exceed some threshold. A point *q* is *directly-density-reachable* from a core point *p* if *q* is within the Eps-neighborhood of *p*, and *density-reachability* is given by the transitive closure of direct density-reachability. Two points *p* and *q* are called *density-connected* if there is a third point *o* from which both *p* and *q* are density-reachable. A *cluster* is then a set of density-connected points which is maximal with respect to density-reachability. *Noise* is defined as the set of points in the database not belonging to any of its clusters. The task of density-based clustering is to find all clusters with respect to parameters *Eps* and *MinPts* in a given database.

Historical Background

In the 1990s, increasingly large spatial databases became available in many applications. This drove not only the development of techniques for efficiently managing and querying these databases, it also provided great potential for data mining. The application of clustering algorithms to spatial databases raised the following requirements:

1. Discovery of clusters with arbitrary shape, because the shape of clusters in spatial databases may be spherical, drawn-out, linear, elongated etc.
2. Good efficiency on large databases, i.e., on databases of significantly more than just a few thousand objects.
3. Minimal requirements of domain knowledge to determine the input parameters, because appropriate values are often not known in advance when dealing with large databases.

Due to the local nature of density-based clusters, this new clustering paradigm promised to address all of these requirements. As required, dense connected areas in the data space can have arbitrary shape. Given a spatial index structure that supports region queries, density-based clusters can be efficiently computed by performing at most one region query per database object. Different from clustering algorithms that optimize a certain objective function, the number of clusters does not need to be specified by the user.

Foundations

The paradigm of density-based clustering was introduced in [4]. Let *D* be a database of points. The definition of density-based clusters assumes a distance function *dist*

(p, q) for pairs of points. The *Eps-neighborhood* of a point p, denoted by NEps(p), is defined by NEps $(p) = \{q \in D \mid dist(p, q) \leq Eps\}$. A point p is *directly density-reachable* from a point q w.r.t. Eps, MinPts if (1) $p \in$ NEps(q) and (2) |NEps(q)|$\geq$MinPts. A point p is *density-reachable* from a point q w.r.t. Eps and MinPts if there is a chain of points $p_1,...,p_n$, $p_1 = q$, $p_n = p$ such that $p_i + 1$ is directly density-reachable from p_i. Density-reachability is a canonical extension of direct density-reachability. Since this relation is not transitive, another relation is introduced. A point p is *density-connected* to a point q w.r.t. *Eps* and *MinPts* if there is a point o such that both, p and q are density-reachable from o w.r.t. *Eps* and *MinPts*. Figure 1 illustrates these concepts.

Intuitively, a density-based cluster is a maximal set of density-connected points. Formally, a *cluster C* wrt. *Eps* and *MinPts* is a non-empty subset of D satisfying the following two conditions:

1. $\forall p, q:$ if $p \in C$ and q is density-reachable from p w.r.t. *Eps* and *MinPts*, then $q \in C$. (maximality)
2. $\forall p, q \in C:$ p is density-connected to q w.r.t. *Eps* and *MinPts*. (connectivity)

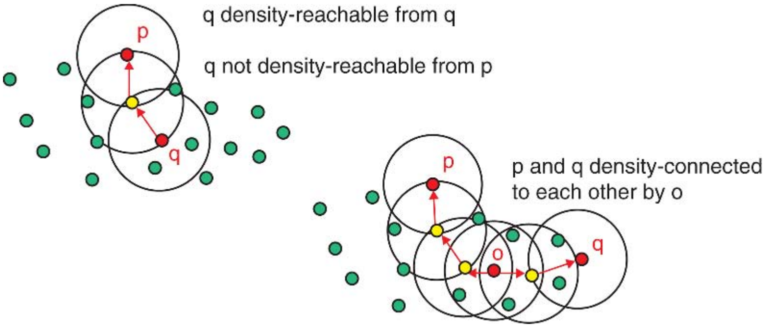
Let $C_1, ..., C_k$ be the clusters of the database D w.r.t. Eps and MinPts. The *noise* is defined as the set of points in D not belonging to any cluster C_i, i.e., $noise = \{p \in D \mid \forall i: p \notin C_i\}$.

Density-based clustering distinguishes three different types of points (see Fig. 2):

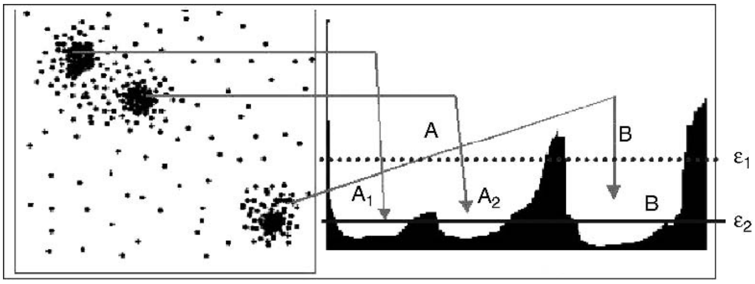
1. *Core points*, i.e., points with a dense neighborhood ($|N_{Eps}(p)| \geq MinPts$)
2. *Border points*, i.e., points that belong to a cluster, but whose neighborhood is not dense, and
3. *Noise points*, i.e., points which do not belong to any cluster

In the right part of Fig. 1, e.g., o is a core point, p and q are border points, and n is a noise point.

Density-based clusters have two important properties that allow their efficient computation. Let p be a core point in D. Then the set $O = \{$o | o$\in D$ and o is density-reachable from p wrt. *Eps* and *MinPts*$\}$ is a cluster with regard to *Eps* and *MinPts*. Let C be a cluster in D. Each point in C is density-reachable from any of the core points of C and, therefore, a cluster C contains exactly the points which are density-reachable from an arbitrary core point of C. Thus, a cluster C with regard to *Eps* and *MinPts* is uniquely determined by *any* of its core points. This is the foundation of the DBSCAN algorithm for density-based clustering [5].


q density-reachable from q
q not density-reachable from p
p
q
p and q density-connected
to each other by o
p
o
q

Density-based Clustering. Figure 1. Density-reachability and connectivity.



Density-based Clustering. Figure 2. Reachability-plot for data set with hierarchical clusters of different sizes, densities and shapes.

DBSCAN

To find a cluster, DBSCAN starts with an arbitrary database point p and retrieves all points density-reachable from p wrt. *Eps* and *MinPts*, performing region queries first for p and if necessary for p's direct and indirect neighbors. If p is a core point, this procedure yields a cluster wrt. *Eps* and *MinPts*. If p is not a core point, no points are density-reachable from p. DBSCAN assigns p to the noise and applies the same procedure to the next database point. If p is actually a border point of some cluster C, it will later be reached when collecting all the points density-reachable from some core point of C and will then be (re-)assigned to C. The algorithm terminates when all points have been assigned to a cluster or to the noise.

In the worst case, DBSCAN performs one region query (retrieving the Eps-neighborhood) per database point. Assuming efficient index support for region queries, the runtime complexity of DBSCAN is $O(n \log n)$, where n denotes the number of database points. In the absence of efficient index structures, e.g., for high-dimensional databases, the runtime complexity is $O(n^2)$. Ester et al. [4] show that a density-based clustering can be updated incrementally without having to re-run the DBSCAN algorithm on the updated database. It examines which part of an existing clustering is affected by an update of the database and presents algorithms for incremental updates of a clustering after insertions and deletions. Due to the local nature of density-based clusters, the portion of affected database objects tends to be small which makes the incremental algorithm very efficient.

GDBSCAN

The basic idea of density-based clusters can be generalized in several important ways [8]. First, any notion of a neighborhood can be used instead of a distance-based *Eps*-neighborhood as long as the definition of the neighborhood is based on a predicate $NPred(p, q)$ which is symmetric and reflexive. The neighborhood N of p is then defined as the set of all points q satisfying $NPred(p, q)$. Second, instead of simply counting the elements in a neighborhood, a more general predicate $MinWeight(N)$ can be used to determine whether the neighborhood N is "dense," if $MinWeight$ is monotone in N, i.e., if $MinWeight$ is satisfied for all supersets of sets that satisfy N. Finally, not only point-like objects but also spatially extended objects such as polygons can be clustered. When clustering polygons, for example, the following predicates are more natural than the *Eps*-neighborhood and the *MinPts* cardinality constraint: "$NPred\ (X,\ Y)$ iff $intersect\ (X,\ Y)$" and "$MinWeight(N)$ iff $\sum_{P \in N} population(P) \geq MinPop$."

The GDBSCAN algorithm [8] for finding generalized density-based clusters is a straightforward extension of the DBSCAN algorithm.

Denclue

Denclue [6] takes another approach to generalize the notion of density-based clusters, based on the concept of *influence functions* that mathematically model the influence of a data point in its neighborhood. Typical examples of influence functions are square wave functions or Gaussian functions. The overall *density* of the data space is computed as the sum of the influence functions of all data points. Clusters can then be determined by identifying *density-attractors*, i.e., local maxima of the overall density function. Most data points do not contribute to the density function at any given point of the data space. Therefore, the Denclue algorithm can be implemented efficiently by computing only a local density function, guaranteeing tight error bounds. A cell-based index structure allows the algorithm to scale to large and high-dimensional datasets.

OPTICS

In many real-life databases the intrinsic cluster structure cannot be characterized by *global* density parameters, and very different local densities may be needed to reveal clusters in different regions of the data space. In principle, one could apply a density-based clustering algorithm with different parameter settings, but there are an infinite number of possible parameter values. The basic idea of the OPTICS algorithm [2] to address this challenge is to produce a novel "cluster-ordering" of the database objects with respect to its density-based clustering structure containing the information about *every* clustering level of the data set (up to a "generating distance" *Eps*). This ordering is visualized graphically to support interactive analysis of the cluster structure.

For a constant *MinPts*-value, density-based clusters with respect to a higher density (i.e., a lower value for *Eps*) are completely contained in clusters with respect to a lower density (i.e., a higher value for *Eps*). Consequently, the DBSCAN algorithm could be extended to simultaneously cluster a database for several *Eps* values. However, objects which are density-reachable with respect to the lowest *Eps* value would always have to be processed first to guarantee that clusters with respect to higher density are finished first. OPTICS works in principle like such an extended DBSCAN algorithm for an infinite number of distance parameters Eps_i which are smaller than a "generating distance" *Eps*. The only difference is that it does not assign cluster memberships, but stores the order in which the objects are processed (the *clustering order*) and the following two pieces of information which would be used by an extended DBSCAN algorithm to assign cluster memberships. The *core-distance* of an object p is the smallest distance *Eps*' between p and an object in its *Eps*-neighborhood such that p would be a core object with respect to *Eps*' if this neighbor is contained in N_{Eps} (p). The *reachability-distance* of an object p with respect to another object o is the smallest distance such that p is directly density-reachable from o if o is a core object. The clustering structure of a data set can be visualized by a *reachability plot* (see Fig. 2) that shows the reachability-distance values r for all objects sorted according to the clustering order. "Valleys" in the reachability plot correspond to clusters, which can be hierarchically nested. In Fig. 2, e.g., cluster A can be decomposed into subclusters A_1 and A_2.

Grid-Based Methods

Sheikholeslami et al. [9] present a grid-based approach to density-based clustering, viewing the dataset as a multi-dimensional signal and applying signal processing techniques. The input data is first discretized, and then a wavelet transform is applied to convert the discretized data into the frequency space, in which the natural clusters become more distinguishable. Clusters are finally identified as dense areas in the transformed space. A strength of the WaveCluster algorithm is that, due to the multi-resolution property of wavelet transforms, clusters can be discovered at multiple resolutions. The runtime complexity of WaveCluster is $O(n \cdot m^d)$, where m denotes the number of discrete intervals per dimension and d denotes the number of dimensions, i.e., it is linear in the dataset size and exponential in the number of dimensions.

In high-dimensional datasets, clusters tend to reside in lower-dimensional subspaces rather than in the full-dimensional space, which has motivated the task of subspace clustering. As a prominent algorithm for this task, the CLIQUE algorithm [1] discretizes the data space and defines a subspace cluster as a set of neighboring dense cells in an arbitrary subspace. Such subspace clusters can be efficiently discovered in a level-wise manner, starting with 1-dimensional clusters, and extending clusters by one dimension at every level. In CLIQUE, as in all grid-based approaches, the quality of the results crucially depends on the appropriate choice of the number and width of the grid cells. To address this problem, Hinneburg and Keim [7] suggest a method of contracting projections of the data space to determine the optimal cutting hyperplanes for partitioning the data. Dimensions are only partitioned if they have a good partitioning plane. It can be shown analytically that the OptiGrid algorithm finds all center-defined clusters, which roughly correspond to clusters generated by a Gaussian distribution.

Key Applications

Geographic Information Systems (GIS)

GIS manage spatial data including points, lines and polygons and support a broad range of applications. In geo-marketing, one may want to find clusters of homes with a given characteristic, e.g., high-income homes, while in crime analysis one of the goals is to detect crime hotspots, i.e., clusters of certain types of crimes. Density-based clustering is a natural choice in these applications.

Image Data

Clustering is an important tool to discover objects in image data. Clusters in such data can have non-spherical shapes and varying density, and they can be hierarchically nested. Density-based clustering has been successfully applied to generate landuse maps from satellite data and to detect celestial sources from astronomical images.

Future Directions

While the driving applications for density-based clustering were geographic information systems, recently other applications of density-based clustering have emerged, in particular in data streams and graph data, which create interesting challenges for future research. For density-based clustering of data streams, Cao et al. [3] introduces the concepts of core-micro-clusters, potential core-micro-clusters and outlier micro-clusters to maintain the relevant statistical information. A novel pruning strategy is designed based on these concepts, which allows accurate clustering in the context of limited memory. One of the major tasks in social network analysis is the discovery of communities, which can be understood as dense subnetworks. As the first algorithm adopting the paradigm of density-based clustering to networks, SCAN [10] efficiently detects not only communities, but also hubs and outliers, which play prominent roles in social networks.

URL to Code

The open source data mining software Weka includes Java implementations of DBSCAN and OPTICS, see http://www.cs.waikato.ac.nz/~ml/weka/index.html.

Cross-references

- ▶ Clustering Overview and Applications
- ▶ Indexing and Similarity Search

Recommended Reading

1. Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 94–105.
2. Ankerst M., Breunig M.M., Kriegel H-P., and Sander J. OPTICS: Ordering Points To Identify the Clustering Structure. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 49–60.
3. Cao F., Ester M., Qian W., and Zhou A. Density-based clustering over an evolving data stream with noise. In Proc. SIAM Conf. on Data Mining, 2006.
4. Ester M., Kriegel H-P., Sander J., Wimmer M., and Xu X. Incremental Clustering for Mining in a Data Warehousing Environment. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 323–333.
5. Ester M., Kriegel H-P., Sander J., and Xu X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, 1996, pp. 226–231.
6. Hinneburg A. and Keim D.A. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998, pp. 58–65.
7. Hinneburg A. and Keim D.A. Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 506–517.
8. Sander J., Ester M., Kriegel H-P., and Xu X. Density-based clustering in spatial databases: the algorithm GDBSCAN and its applications. Data Min. Knowl. Discov., 2(2):169–194, 1998.
9. Sheikholeslami G., Chatterjee S., and Zhang A. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 428–439.
10. Xu X., Yuruk N., Feng Z., Thomas A., and Schweiger J. SCAN: a structural clustering algorithm for networks. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007, pp. 824–833.

Dependencies

▶ FOL Modeling of Integrity Constraints (Dependencies)

Derived Event

▶ Complex Event

Description Logics

Alexander Borgida
Rutgers University New Brunswick, NJ, USA

Synonyms

Terminologic languages; KL-ONE style languages; Concept languages

Definition

Description Logics (DLs) are a family of knowledge representation languages providing features for defining and describing concepts. The associated formal logics answer such questions as "*Is concept C or knowledge base T consistent?*" and "*Is concept A more specific (subsumed by) concept B* ?."

DLs view the world as being populated by individuals, grouped into classes ("concepts"), and related by binary relationships ("roles"). DLs define concepts recursively starting from atomic identifiers by using concept and role *constructors*. A key characteristic of every DL's expressiveness is therefore the set of constructors it supports. The collection of constructors considered has been determined empirically, by experience with a variety of tasks in Natural Language processing and other subfields of Artificial Intelligence. Considerable research has been devoted to finding the complexity of reasoning with various DLs. The result is a family of logics that is intermediate in expressive power between propositional logic and first order logic, with emphasis on decidable fragments.

DLs are particularly useful in the field of databases for conceptual modeling and the specification of ontologies used for information integration or in the Semantic Web.

Historical Background

Semantic Networks and Frame Systems, as originally used for representing knowledge in Artificial Intelligence, were faulted for lack of precise semantics, and Brachman [4] was the first to suggest a (graphical) notation, called KL-ONE, for structured networks based on so-called "epistemologic primitives," with a precise intended meaning. Brachman and Levesque [5] introduced the more logical compositional syntax and semantics for a restricted DL, and considered for the first time the famous expressiveness vs. complexity trade-off: more expressive concept constructors often require provably harder reasoning.

DLs have recently attracted considerable attention since they form the basis of the OWL web ontology language, and there are several widely available implementations.

The *Description Logic Handbook* [1] is the definitive reference book on this topic, covering the theory, implementation and application of Description Logics.

Foundations

Concept and Role Constructors

Consider the concept *"objects that are related by locatedIn to some City, an d by ownedBy to some Person"*. Much like one can build a complex type in a programming language by using type constructors, or a complex boolean formula using connectives that can be viewed as term constructors (e.g., $p \wedge \neg q$ written as the term **and**(p,**not**(q))), so in a DL one could specify this concept using a term-like syntax

$$\textbf{and}(\textbf{some}(locatedIn, City), \textbf{some}\ (ownedBy, Person))$$

The conventional notation used for this in the research literature is actually

$$(\exists locatedIn.City \sqcap \exists ownedBy.Person)$$

where $\exists r.\ A$ denotes all object with at least one r-role filler being in concept A, and $\sqcap$ intersects the set of instances of its argument concepts. A variety of other syntactic notations for complex concepts exist, including one for the OWL web ontology language, based on XML.

Other frequently used concept constructors include: so-called "boolean connectives" for unioning $\sqcup$, or complementing $\neg$ concepts; value restrictions $\forall p.C$, as in $\forall locatedIn.SmallTown$ denoting individuals related by *locatedIn* only to instances of *SmallTown*; cardinality constraints such as $\geq n\,p.C$, as in $\geq 2\,ownedBy.RichPerson$ denoting objects that are owned by at least two instances of concept *RichPerson*; enumerations $\{J_1,\ J_2,...\}$, as in $\{ROMULUS, REMUS\}$ denoting the set of these two individuals. Other constructors can often be defined in terms of these e.g., objects with RED hair color value ($hair\_color$: RED) is just $\exists hair\_color.\{RED\}$

Useful role constructors include: role inversion r^-, as in $ownedBy^-$ denoting what might be called the "ownerOf" relationship; role restriction $r|_C$, as in $childOf|_{Male}$ denoting what might be called "sonOf"; and role composition, as in $(childOf \circ childOf)$, denoting what might be called "grandchildOf."

Of particular utility for database applications are concept constructors that concern role fillers that are "concrete types" such as integers and strings (e.g., $\forall age.min(15)$ denotes objects whose age value is at least 15), and those that select individuals for whom combinations of roles act as unique identifiers ("keys").

Summarizing, the following properties are notable and distinguishing features of DLs as far as concept constructors:

- Although concepts resemble programming language record types, roles are multi-valued, in contrast to record fields; e.g., a house may be *ownedBy* by more than person.
- Although symbols like ∀ and ∃ are used in the mathematical notation of DLs, there are in fact *no variables* (free or bound) appearing in concept specifications.
- Complex concepts can be constructed by nesting, as in $\exists ownedBy.(Person \sqcap (\geq 3\,childOf^-.Teenager))$, with intended meaning "objects which are owned by a person that is the parent of at least three teenagers"; and they need not be given names, acting as complex noun phrases with relative clauses.

Formal Semantics

The precise meaning/denotation of concepts is usually specified by an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$, which assigns to every concept name a subset of the domain $\Delta^{\mathcal{I}}$, to every role name a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to every individual name some object in $\Delta^{\mathcal{I}}$. Table 1 shows how this is extended from atomic identifiers to complex terms for some of the concept and role constructors. (The others can be defined from these using complement.) It is often possible to achieve the same effect indirectly by providing a translation from concepts/roles to Predicate Calculus formulas with one/two free variables.

Using Concepts in Knowledge Bases

Concepts and roles do not actually make any assertions – they are like formulas with free variables. The following are some of the possible uses for them.

First, one can make assertions about *necessary properties* of atomic concepts, e.g., "every house is

Description Logics. Table 1. Semantics of composite concepts and roles

Term	Interpretation	Translation to FOPC
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	$C(x) \wedge D(x)$
$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	$\neg C(x)$
$\exists p.C$	$\{\delta \in \Delta^{\mathcal{I}} \mid p^{\mathcal{I}}(\delta) \cap C^{\mathcal{I}} \neq \emptyset\}$	$\exists y\ p(x,y) \wedge C(y)$
$\geq np.C$	$\{\delta \in \Delta^{\mathcal{I}} \mid \lvert p^{\mathcal{I}}(\delta) \cap C^{\mathcal{I}} \rvert \geq n\}$	$\exists z_1,\ldots,\exists z_n\ p(x,z_1) \wedge \ldots \wedge\ p(x,z_n)\ \wedge\ z_i \neq z_j\ \wedge\ C(z_i)$
$\{b_1,\ldots,b_m\}$	$\{b_1^{\mathcal{I}},\ldots,b_m^{\mathcal{I}}\}$	$x = b_1\ \vee \ldots \vee x = b_m$
p^{-}	$\{(\delta,\delta') \mid (\delta',\delta) \in R^{\mathcal{I}}\}$	$p(y,x)$
$p\rvert_C$	$\{(\delta,\delta') \in p^{\mathcal{I}} \mid \delta' \in C^{\mathcal{I}}\}$	$p(x,y) \wedge C(y)$
$p \circ q$	$p^{\mathcal{I}} \circ q^{\mathcal{I}}$	$\exists z\ p(x,z) \wedge q(z,y)$

located in some city and is owned by some person," by using a syntax like

$$House \sqsubseteq (\exists\ locatedIn.City \sqcap \exists\ ownedBy.Person)$$

where $\sqsubseteq$ is read as "is subsumed by." An assertion such as $A \sqsubseteq B$ is satisfied only in interpretations where $A^{\mathcal{I}}$ is a subset of $B^{\mathcal{I}}$.

Second, one can *give definitions* to atomic names; e.g., "a French house" can be defined to be a house located in a French city, using the syntax

$$FrenchHouse \equiv (House \sqcap \forall locatedIn.FrenchCity)$$

For database *cognoscenti*, definitions will be familiar as views.

If one allows *general inclusion axioms* of the form $\alpha \sqsubseteq \beta$, the definition above is equivalent to two statements

$$FrenchHouse \sqsubseteq (House \sqcap \forall locatedIn.FrenchCity)$$
$$(House \sqcap \forall locatedIn.FrenchCity) \sqsubseteq FrenchHouse$$

Such assertions about concepts form part of the *terminology* of a domain, and hence are gathered in a so-called *TBox* $\mathcal{T}$. Other assertions in a TBox may specify properties of roles, such as role inclusion (e.g., *ownedBy* $\sqsubseteq$*canSell*) or whether a role is transitive.

Given a terminology $\mathcal{T}$, one can then ask whether some subsumption $\alpha \sqsubseteq \beta$ follows from it, written as $\mathcal{T} \models \alpha \sqsubseteq \beta$, i.e., whether the subsumption holds in all interpretations satisfying $\mathcal{T}$. This is the basis of a number of standard services that are provided by a DL reasoner, such as automatically classifying all identifiers into a subsumption taxonomy, or checking whether concepts or the entire knowledge base are (in)coherent. Other, non-standard but useful reasoning services include computing the least common subsumer of a pair of concepts (a non-trivial task if one excludes concept union $\sqcup$ as a constructor), rewriting concepts using definitions in a TBox for goals such as abbreviation, and pinpointing errors/reasoning in inconsistent terminologies.

In order to describe specific states of a world, one uses formulas of two forms: (i) *LYON* : *City*, expressing that the individual *LYON* is an instance of concept *City*; and (ii) *locatedIn*(*LYON*, *EUROPE*), indicating that *LYON* and *EUROPE* are related by the *locatedIn* relationship. A collection $\mathcal{A}$ of such assertions about the state of the world is called an *ABox*. A reasoner can then determine whether a particular concept membership (resp. role relationship) follows from an (ABox, TBox)-pair: $(\mathcal{A}, \mathcal{T}) \models d : \beta$ (resp. $(\mathcal{A}, \mathcal{T}) \models r(d, e)$). For example, from the above facts about *LYON*, and the definition *EuropeanCity* $\equiv$ (*City* $\sqcap$ *locatedIn* : EUROPE), one can conclude that *LYON* : *EuropeanCity*. Based on the above judgment, DL reasoners can determine the (in)consistency of an ABox $\mathcal{A}$ with respect to a TBox $\mathcal{T}$, and can classify individuals in $\mathcal{A}$ under the most specific concept in the taxonomy. It is important to note that, unlike standard databases, DLs do *not* adopt the so-called "closed-world assumption," so that from the above facts one cannot conclude that LYON is located in only one place; i.e., to obtain this conclusion, (*LYON*:≤ 1*locatedIn*) needs to be explicitly added to the previous collection of assertions.

Mathematical Properties

The formal complexity of reasoning with a DL-based knowledge base is determined by the concept and role constructors allowed, and the form of the axioms (e.g., Are axioms permitted to create cyclic dependencies among concept names? Are there role subsumption axioms?). Also, interesting connections have been found between DLs and epistemic propositional logic, propositional dynamic logic, and various restricted subsets of First Order Logic. Many of these results have been summarized in [1].

Implementations

Unlike the earliest implemented systems (KL-ONE and LOOM), the second generation of systems (BACK, KANDOR, and CLASSIC), took seriously the notion of computing *all* inferences with respect to some semantic specification of the DL, and in fact tried to stay close to polynomial time algorithms for judgments such as subsumption, often by finding a normal form for concepts. In contrast, the DLs implemented in the past decade have been considerably more expressive (theoretically, the worst-case complexity of deciding subsumption ranges from PSPACE to non-deterministic exponential space complete), and are implemented using so-called "tableaux techniques," which have been highly optimized so that performance on practical KBs is acceptable.

Key Applications

Conceptual Modeling

As knowledge representation languages, with concept constructors that have been found to be empirically useful for capturing real-world domains of discourse, DLs are relevant to databases as conceptual modeling languages, used in the first step of database design; and for database integration, as the mediated conceptual schema seen by external users. The advantages of DLs over other notations such as Extended ER diagrams and UML is their greater expressive power and the existence of implemented reasoners that are provably correct, and that can, for example, detect inconsistencies in domain models. In fact, it has been shown that EER and UML class diagrams can be translated into the $\mathcal{SHIQ}$ DL [2,6], for which multiple implementations exist currently.

Ontology Specification

DLs have been adopted as the core (OWL-DL) of the OWL web ontology language proposed by W3C [8]. Ontologies, and reasoning with them, are predicted to play a key role in the Semantic Web, which should support not just data integration but also web service composition, orchestration, etc.

Management and Querying of Incomplete Information

Because of open-world reasoning, and the ability to make assertions about objects not explicitly present in the knowledge base (e.g., $VLDB09 : \forall takesPlaceIn.(City \sqcap locatedIn : ASIA \sqcap \forall hasPopulation.min(1000000))$ says that wherever $VLDB09$ will be held, the city will be in Asia, and it will have population over 1 million), DLs are particularly well suited to represent and reason with partial knowledge [3]. Among others, DLs face and resolve aspects of the view update problem. Industrial applications of this have been found in configuration management. Research has also addressed issues related to more complex queries, with a flurry of recent results concerning conjunctive query answering over ABoxes, starting with [7].

Query Organization and Optimization

Since a complex concept C can be used to return all individuals b classified under it (i.e., $b : C$), C can be thought of as a query, though one of somewhat limited power since it cannot express all conditions that might be stated in SQL, nor can it return sets of tuples. In exchange, query containment (which is just subsumption) is decidable, and sometimes efficiently computable. This can be used to organize queries, and help answer queries using results of previous queries.

In addition, DL-based conceptual models of database schemata can be used to enable advanced semantic query optimization, in particular when combined with DL-based description of the physical layout of data. In this setting, identification constraints, such as keys and functional dependencies, are often added to DLs [9] and subsequently used to enable rewriting of queries for purposes such as removing the need for duplicate elimination in query answers.

Future Directions

In addition to continued work on extending the expressive power of DLs, and various forms of implementations, important topics of current concern include (i) representation and reasoning about actions, (ii) extracting modules from and otherwise combining DL ontologies, (iii) restricted expressive power DLs that are tractable or of low complexity yet can handle important practical ontologies in the life sciences and conceptual models, (iv) combining rules with DLs, and, (v) conjunctive query processing over DL ABoxes.

URL to Code

http://www.dl.kr.org/ The Description Logics web site. Among others, look for links to the Description Logic Workshops (containing the most up-to-dateresearch in the field), the "navigator" for the complexity of reasoning with variousDLs, and for DL implementations.

Cross-references

▶ Conceptual Modeling
▶ Data Integration
▶ Semantic Web

Recommended Reading

1. Baader F., Calvanese D., McGuinness D.L., Nardi D., and Patel-Schneider P.F. The Description Logic Handbook: Theory, Implementation, and Applications, 2nd edn. Cambridge University Press, 2003.
2. Berardi D., Calvanese D., and De Giacomo G. Reasoning on UML class diagrams. Artif. Intell. J., 168(1-2): 70–118, 2005.
3. Borgida A. Description logics in data management. IEEE Trans. Knowl. Data Eng., 7(5): 671–682, 1995.
4. Brachman R.J. What's in a concept: structural foundations for semantic networks. Int. J. Man-Machine Studies, 9(2): 127–152, 1997.
5. Brachman R.J. and Levesque H.J. The tractability of subsumption in frame-based description languages. In Proc. 4th National Conf. on AI, 1984, pp. 34–37.
6. Calvanese D., Lenzerini M., and Nardi D. Unifying class-based representation formalisms. J. Artif. Intell. Res., 11:119–240, 1999.
7. Horrocks I. and Tessaris S. A conjunctive query language for description logic aboxes. In Proc. 12th National Conf. on AI, 2000, pp. 399–404.
8. OWL Web Ontology Language Reference, http://www.w3.org/TR/owl-ref/
9. Toman D. and Weddell G. On keys and functional dependencies as first-class citizens in description logics. J. Auto. Reason., 40(2–3): 117–132, 2008.

Design for Data Quality

Carlo Batini, Andrea Maurino
University of Milan Bicocca, Milan, Italy

Synonyms

Schema normalization; Design for quality

Definition

The design for data quality (DQ) is the process of designing data artifacts, such as information systems, databases, and data warehouses where data quality issues are considered relevant.

In information systems different types of data are managed; these may be structured such as relational tables in databases, semi-structured data such as XML documents, and unstructured data such as textual documents. Information manufacturing can be seen as the processing system acting on raw data of different types, whose aim is to produce information products. According to this approach, the design for data quality aims to design information-related processes (e.g., creation, updating, and delivering of information) taking into account data quality dimensions.

In the database field, the design for data quality has the objective of producing good (with respect to a given set of quality dimensions) conceptual and relational schemas and corresponding good data values. Concerning schema design, the most important quality dimensions are:

- *Correctness with respect to the model* concerns the correct use of the categories of the model in representing requirements
- *Correctness with respect to requirements* concerns the correct representation of the requirements in terms of the model categories
- *Minimality* requires that every part of the requirements is represented only once in the schema
- *Completeness* measures the extent to which a conceptual schema includes all the conceptual elements necessary to meet some specified requirement
- *Pertinence* measures how many unneeded conceptual elements are included in the conceptual schema
- *Readability* imposes that the schema is expressed in a clear way for the intended usage

It is worth noting that the quality of data is strongly influenced by the quality of the schema that dictates the structure of these data. Consequently, the design for data quality can be seen as the first activity in a DQ management program. In fact, let consider a schema composed of the two relations: Movie(Title, Director, Year, DirectorDateOfBirth) and Director(Name, Surname, DateOfBirth). The schema does not satisfy the minimality dimension due to the duplication of the DateOfBirth attribute. This design error will reflect on the data values that could suffer from consistency and accuracy problems.

The design for data quality is a relevant problem also for data warehouse systems. The data warehouse design process has to consider several dimensions. The *completeness* dimension is concerned with the preservation of all the entities in data sources in the data warehouse schema. The *minimality* dimension describes the degree up to which undesired redundancy is avoided during the source integration process. The *traceability* dimension is concerned with the fact that all kinds of data processing of users, designers, administrators and managers should be traceable in the data warehouse system. The *interpretability* dimension ensures that all components of the data warehouse schema are well described.

Historical Background

The problem of designing for data quality was considered relevant to the area of information systems since the late 1980s. The total data quality management methodology (TDQM) [11] introduces the information product (IP) approach, which considers information as one of the products that an organization produces. As a consequence, traditional manufacturing design for quality strategies can be applied to IPs also. Within the TDQM methodology, the IP conceptual definition phase is in charge of defining an Entity-Relationship (ER) schema enhanced with quality features, which defines the IP, its information quality (IQ) requirements, an information manufacturing system that describes how the IP is produced, and the interactions among information suppliers (vendors), manufacturers, consumers, and IP managers. In this way, many IQ requirements can be enforced into the new information manufacturing system, resulting in design for information quality procedures analogous to that of design for quality in product manufacturing.

From a historical viewpoint, early methodologies for database design did not consider data quality issues as a relevant problem. Subsequently, design methodologies incorporated DQ issues by extending conceptual and logical models with quality features. Several solutions have been proposed for extending the Entity-Relationship model with quality characteristics. In [8], a methodology for modeling the quality of attribute values as another attribute of the same entity is described. A different approach is proposed in the Quality Entity-Relationship (QER) model [9], which extends the ER model by providing a mechanism to embed quality indicators into conceptual schemas. QER introduces two generic entities, DQ_Dimension and DQ_Measure. DQ_Dimension, with attributes Name and Rating, models all possible quality dimensions (e.g., accuracy) for an attribute (e.g., Address) and its values (e.g., accuracy = "0.8"). The entity DQ_Measure is used to represent metrics for corresponding data quality measurement values (e.g., in a [0,1] scale, "0" for very inaccurate, "1" for very accurate). For what concerns logical database models, [13] presents an attribute based model that extends the relational model by adding an arbitrary number of underlying quality indicator levels to attributes. Attributes of a relational schema are expanded into ordered pairs, called quality attributes, consisting of the attribute and a quality key. The quality key is a reference to the underlying quality indicator(s). Other extensions of the relational model are presented in [14,15]. Studies on normal forms, and more in general on the normalization process provide techniques for avoiding certain anomalies such as duplication of values or updates. These techniques can be considered as evaluation techniques for measuring the quality of logical schemas.

More recently, [5] presents a new design for data quality methodology for incorporating data quality requirements into database schemas by means of goal-oriented requirement analysis techniques. The proposal extends existing approaches for addressing data quality issues during database design in two ways. First, authors consider data quality requirements (including descriptive, supportive and reflective requirements). Second, a systematic way to move from high-level, abstract quality goals into quality assurance requirements is presented.

For what concerns data warehouse design, research on designing schemas with quality properties is limited. One of the most important results in this field is the DWQ framework proposed in [3], where a general-purpose model of quality has been defined to capture the set of quality factors associated to the various data warehouse objects, and to perform their measurement for specific quality goals. In particular, it is shown that the refreshment process and the selection of the materialized views are demonstrative examples where quality can drive the design process [10].

Foundations

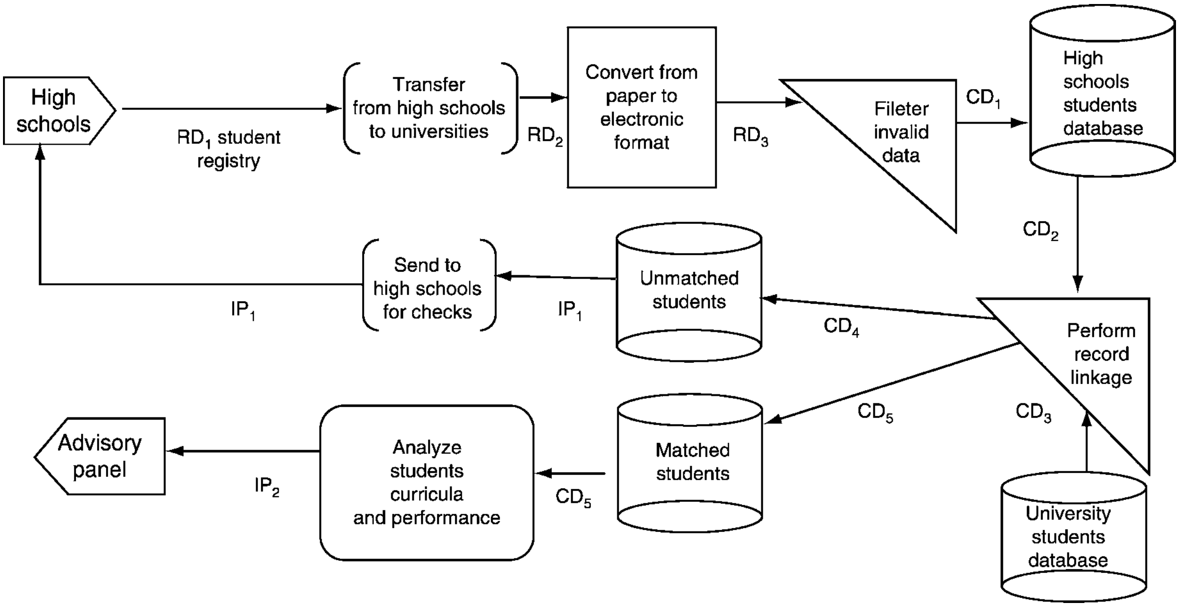
In the design of information systems the term *information quality* refers to processes involving the information life-cycle (creation, updating, and delivering of information). In particular, two important quality dimensions can be defined: *efficiency*, which measures process by comparing the production with costs and resources, and *effectiveness*, which measures the process outcome; namely the real result of the process for which the process has been conceived. The most popular model for designing quality information systems is IP-MAP [7].

IP-MAP is a graphical model designed to help people to comprehend, evaluate, and describe how an information product such as an invoice, a customer order, or a prescription is assembled in a business process. The IP-MAP is aimed at creating a systematic representation for capturing the details associated with the manufacturing of an information product. IP-MAP models (hereafter IP-MAPs) are designed to help analysts visualize the information production process, identify ownership of the process phases, understand information and organizational boundaries, and estimate time and quality metrics associated with the production process to be considered. Figure 1 shows an example of IP-MAP representing high schools and universities of a district that have agreed to cooperate in order to improve their course offering to students, avoiding overlapping and becoming more effective in the education value chain. To this end, high schools and universities have to share historical data on students and their curricula. Therefore, they perform a record linkage activity that matches students in their education life cycle. To reach this objective, high schools periodically supply relevant information on students; in case it is in paper format the information has to be converted into electronic format. At this point, invalid data are filtered and matched with the database of university students. Unmatched student records are sent back to high schools for clerical checks, and matched student records are analyzed; the results of the analysis on curricula and course topics are sent to the advisory panel of universities.

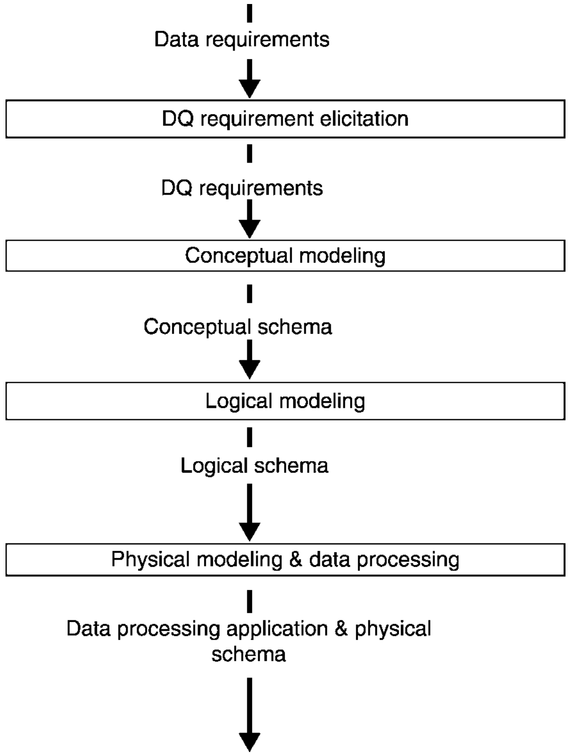
Typical activities for database design that are influenced by data quality include (see Fig. 2):

1. *DQ requirements elicitation*
2. *Conceptual modeling*
3. *Logical modeling*
4. *Data processing and Physical modeling*

The process starts with the data quality requirements elicitation. Here there is a similarity between software development processes and the design for data quality in databases. In fact, in software development processes, functional requirements describe what the software does, while the non-functional properties describe qualities that functionalities need to support. In the design for data quality process, *data requirements* describe the Universe of Discourse that the design has to represent in the database, while *data quality requirements* model quality dimensions by which a user evaluates DQ and quality processes related to data acquisition, and update. Furthermore, quality attributes are considered at two levels: quality parameters, model quality dimensions by which a user evaluates


High schools
RD1 student registry
Transfer from high schools to universities
RD2
Convert from paper to electronic format
RD3
Fileter invalid data
CD1
High schools students database
CD2
Send to high schools for checks
IP1
IP1
Unmatched students
CD4
Perform record linkage
CD5
CD3
Advisory panel
IP2
Analyze students curricula and performance
CD5
Matched students
University students database

Design for Data Quality. Figure 1. Example of IP-MAP.

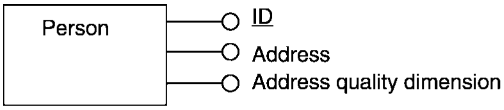

Data requirements
DQ requirement elicitation
DQ requirements
Conceptual modeling
Conceptual schema
Logical modeling
Logical schema
Physical modeling & data processing
Data processing application & physical schema

Design for Data Quality. Figure 2. Design for data quality process.

DQ (e.g., accuracy and currency); quality indicators capture aspects of the data manufacturing process (e.g., when, where, how data is produced) and provide information about the expected quality of the data produced. The data quality requirements elicitation phase produces quality requirements that are inputs to the conceptual modeling phase.

During the conceptual modeling phase, concepts and their attributes are elicited and organized into a conceptual schema. Moreover, quality parameters are identified and associated to attributes in the schema. Finally, each parameter is refined into one or more quality indicators. There are two possible design choices. A first possibility is to model the quality of attribute values as another attribute of the same entity for each attribute [8,9]. As an example, in order to add a quality dimension (e.g., accuracy or completeness) for the attribute Address of an entity Person, it is possible to add (see Fig. 3) a new attribute Address-QualityDimension to the entity.

The drawback of this solution is that now the entity is no longer normalized. Another drawback is that if there is the need to define several dimensions, a new attribute for each dimension has to be added, resulting in a proliferation of attributes. The second possibility is to add specified data quality entities, and to create a many-to-many relationship among these entities and application entities.



Design for Data Quality. Figure 3. Representing quality dimensions in the entity-relationship model.

The conceptual schema is the input of the next phase, concerned with logical modeling. In the process of translation from the conceptual schema to the logical schema, normalization has to be achieved. Normal forms guarantee the absence of data processing anomalies; as such normal forms are a relevant concept in design for data quality. The same holds true for intra-relational and inter-relational integrity constraints defined in the logical modeling phase; integrity constraints are the fundamental means to guarantee the consistency dimension for data.

The logical schema is an input for the physical modeling and data processing phase. Data processing activities are one of the most critical tasks from a data quality perspective. A careful design of the data processing procedures can significantly reduce data quality problems. For example, if the domain of an attribute is composed of a fixed set of values (e.g., the educational degrees), the use of predefined lists to insert values in attributes reduce possible typos and increases the syntactic accuracy.

The design of distributed databases especially when independently developed databases are considered, raises more challenges in the design for data quality. In this case, in fact, existing schemas cannot be modified and issues related to the quality of schema cannot be dealt with. The only possibility to partially work out the quality issues is the definition of effective schema integration techniques. Schema level conflicts include: i) *heterogeneity conflicts*, occurring when different data models are used; ii) *semantic conflicts*, regarding the relationship between model element extensions, iii) *description conflicts*, concerning concepts with different attributes, and iv) *structural conflicts*, regarding different design choices within the same data model. Instance level conflicts are another typical problem of distributed databases and occur when conflicting data values are provided by distinct sources for representing the same objects. At design time, it is possible to plan conflict resolution selecting suitable aggregation functions [2]; such functions take two (or more) conflicting values of an attribute as input and produce a value as output that must be returned as the result of the posed query. Common resolution functions are MIN, MAX, AVG. Other techniques are discussed and compared in [1]. Techniques for instance level conflict resolution at design time have a major optimization problem. Consider two relations EmployeeS1 and EmployeeS2 defined in two different databases, representing information about employees of a company. Also assume that no schema conflict is detected. Suppose that the two relations have instance level conflicts for the Salary attribute. Moreover, suppose that at design time it is specified that in case of conflicts the minimum salary must be chosen. Given the global schema, Employee (EmployeeID, Name, Surname, Salary, Email), let consider the following query:

```
SELECT EmployeeID, Email
FROM Employee
WHERE Salary < 2000
```

Since the Salary attribute is involved in the query, all employees must be retrieved in order to compute the minimum salary, not only employees with Salary < 2000, even if no conflicts on salary occur. Therefore conflict resolution at design time may be very inefficient.

Future Directions

The problem of measuring and improving the quality of data has been dealt with in the literature as an activity to be performed a posteriori, instead of during the design of the data. As a consequence, the design for data quality is still a largely unexplored area. In particular, while the data quality elicitation and the translation of DQ requirement into conceptual schemas have been investigated; there is a lack of comprehensive methodologies covering all the phases of design of information systems/databases/data warehouse systems. Moreover, easy-to-use and effective design tools could help the database designer.

Open areas of research concern the definition of quality information dimensions for semi-structured and unstructured data. In particular, while the use of XML technologies is growing within organizations, the research on design of the XML quality schemas is at an early stage.

Cross-references

▶ Data Profiling
▶ Data Provenance
▶ Data Quality Assessment
▶ Information Quality
▶ Quality Data Management
▶ Quality in Data Warehouses

Recommended Reading

1. Batini C. and Scannapieco M. Data Quality: Concepts, Methodologies and Techniques, Springer, New York, 2006.
2. Dayal U. Query processing in a multidatabase system. In Query Processing in Database Systems, W. Kim, D.S. Reiner, D.S. Batory (eds.). Springer, 1985, pp. 81–108.
3. Jarke M., Jeusfeld M.A., Quix C., and Vassiliadis P. Architecture and quality in data warehouses: an extended repository approach. Inf. Syst. 24(3):229–253, 1999.
4. Jeusfeld M.A., Quix C., and Jarke M. Design and analysis of quality information for data warehouses. In Proc. 17th Int. Conf. on Conceptual Modeling, 1998, pp. 349–362.
5. Jiang L., Borgida A., Topaloglou T., and Mylopoulos J. Data quality by design: a goal-oriented approach. In Proc. 12th Conf. on Information Quality, 2007.
6. Navathe S.B., Evolution of data modeling for databases. Commun. ACM, 35(9):112–123, 1992.
7. Shankaranarayanan G., Wang R.Y., and Ziad M. IP-MAP: representing the manufacture of an information product. In Proc. 5th Conf. on Information Quality, 2000.
8. Storey V. and Wang R.Y. Extending the ER model to represent data quality requirements. In Data Quality, R. Wang, M. Ziad, W. Lee (eds.). Kluwer Academic, Boston, MA, 2001.
9. Storey V.C. and Wang R.Y. Modeling quality requirements in conceptual database design. In Proc. 3rd Conf. on Information Quality, 1998, pp. 64–87.
10. Vassiliadis P., Bouzeghoub M., and Quix C. Towards quality-oriented data warehouse usage and evolution. In Proc. 11th Conf. on Advanced Information Systems Engineering, 1999, pp. 164–179.
11. Wang R.Y. A product perspective on total data quality management. Commun. ACM, 41(2):58–65, 1998.
12. Wang R.Y., Kon H.B., and Madnick S.E. Data quality requirements analysis and modeling. In Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 670–677.
13. Wang R.Y., Reddy M.P., and Kon H.B. Toward quality data: an attribute-based approach. Decision Support Syst., 13 (3–4):349–372, 1995.
14. Wang R.Y., Storey V.C., and Firth C.P. A framework for analysis of data quality research. IEEE Trans. Knowl. Data Eng., 7 (4):623–640, 1995.
15. Wang R.Y., Ziad M., and Lee Y.W. Data Quality. Kluwer Academic, Boston, MA, 2001.

Design for Quality

▶ Design for Data Quality

Desktop Metaphor

► Direct Manipulation

Detail-in-Context

► Distortion Techniques

Deviation from Randomness

► Divergence from Randomness Models

Dewey Decimal Classification

► Dewey Decimal System

Dewey Decimal System

Prasenjit Mitra
Pennsylvania State University, University Park,
PA, USA

Synonyms

Dewey decimal classification

Definition

The Dewey Decimal Classification (DDC) System is a system primarily used in libraries to classify books. In general, the system claims to provide a set of categories for all human knowledge. The system consists of a hierarchy of classes. At the top level, there are ten main classes, that are divided into 100 divisions which are sub-divided into 1,000 sections. The system was conceived by Melvil Dewey in 1873 and published in 1876. DDC uses Arabic numerals to number the classes and explicates the semantics of a class and its relation to other classes.

Key Points

Since its first publication in 1876 by Melvil Dewey [2], the Dewey Decimal Classification (DDC) has been updated to accommodate changes to the body of human knowledge. The current version, DDC 22, was published in mid-2003 [3] (http://www.oclc.org/dewey/versions/ddc22print/). Currently, the Online Computer Library Center (OCLC) (http://www.oclc.org) of Dublin Ohio owns the copyrights associated with the Dewey Decimal system. Each new book is assigned a DDC number by the Library of Congress. As of the date this article was written, the OCLC has accepted all assignments of DDC numbers made by the Library of Congress. The OCLC claims that libraries in more than 135 countries use the DDC and it has been translated to over 30 languages (http://www.oclc.org/dewey/versions/ddc22print/intro.pdf).

The DDC has ten main classes: *Computer Science, information and general works; Philosophy and psychology; Religion; Social Sciences; Language; Science; Technology; Arts and recreation; Literature; and History and geography* [1]. Consequently, all fiction books fall under the category Literature (800). To avoid a large number of rows in the 800 range, libraries separately stack non-fiction and fiction books in different sections. To allow for further subdivision of classes, the DDC number for a class can contain further subdivisions of ten after the three digit number. After the three digits for a class number, the classes can be further subdivided. While determining the subject of a work, the editor, at the Dewey editorial office at the Decimal Classification Division of the Library of Congress, tries to determine the author's intent. The entire content of the book is taken into account along with reviews, reference works, and opinions of subject experts.

The DDC forms the basis for the more expressive Universal Decimal Classification. Alternatives to the DDC are the Library of Congress Classification. The construction of the DDC was top down (designed by one person, Dewey) and is somewhat inflexible to accommodating changes in human knowledge. In contrast, the Library of Congress Classification has 21 classes at the top-level of the hierarchy and was developed by domain experts in a bottom-up fashion. The simple numbering system of the DDC makes it easy to use.

Cross-references

► Library of Congress METS

Recommended Reading

1. Chan L.M. Dewey Decimal Classification: A Practical Guide. Forest, 1994.
2. Dewey M. Dewey; Decimal classification and relative index for libraries, clippings, notes. Library Bureau, 1891.
3. Mitchell J.S. Summaries DDC 22: Dewey Decimal Classification. OCLC, 2003.

DHT

► Distributed Hash Table

Diagram

CARLO BATINI
University of Milan Bicocca, Milan, Italy

Synonyms

Diagram; Diagrammatic Representation; Graph; Graphic

Definition

A *diagram* is (i) a set of *symbols* selected in an alphabet of *elementary shapes*, such as rectangles, circles, and *complex shapes*, built by composing elementary shapes, and (ii) a set of *connections* linking symbols, such as straight lines, curved lines, directed lines, etc. Diagrams are used to visualize in all aspects of data base design and usage a wide set of concepts, artifacts, schemas, values, such as conceptual schemas, logical schemas, UML diagrams, database instances, queries, results of queries.

The visual representation made possible by a diagram expresses a functional relationship between concepts represented and symbols/connections representing them. e.g., in the diagrammatic representation of Entity Relationship schemas, a rectangle is associated to an entity, a diamond is associated to a relationship.

When drawing a diagram, human beings and display devices adopt suitable *drawing conventions*, *aesthetic criteria*, and *drawing constraints*. Drawing conventions express general constraints on the geometric representation of symbols and connections. The geometric representation is characterized by the number of spatial dimensions; the usual representation adopted is 2-dimensional (2D), other representations are 1D, 3D, 2–3D. Other drawing conventions refer to general rules on symbols and connections, for example, *polyline* drawing correspond to connections made of sequences of straight lines and *orthogonal* drawing use connections made of horizontal and vertical lines. Aesthetic criteria concern the shape of the diagram, independently from the meaning of symbols, and try to capture universal criteria for expressing the idea of beauty. Drawing constraints refer to the semantics of the underlying represented schema or instance.

Key Points

The above concepts are explained by means of Fig. 1. Concerning drawing conventions, both diagrams adopt a 2D representation. Diagram b adopts a further drawing convention, namely, symbols are inscribed in a rectangular grid. Drawing conventions may express some characteristic of the semantics of the diagram; for example, upward drawings visualize hierarchical relationships, such as generalization hierarchies in conceptual schemas.

Examples of aesthetic criteria are: (i) minimization of crossing between connections, (ii) minimization of the number of bends in connections, (iii) minimization of the area occupied by the diagram, (iv) maximization of the display of symmetries. The above comment to "try to capture universal criteria" is based upon recognition that an expression of beauty in Chinese culture, for example, is asymmetry, and not symmetry. Considering criterion (i), diagram a. has 6 crossings, while diagram b. has no crossing. In diagram b. Employee is symmetric with respect to Vendor, Worker, and Engineer. Note that one cannot simultaneously optimize aesthetic criteria, as intuitively understood considering, for example, previous criteria (i) and (iii).

Examples of drawing constraints are (i) place a given symbol in the centre of the drawing (e.g., Employee in Fig. 1b), keep a group of symbols close together.

Graph drawing is the research area that investigates algorithms that, having in input a nonvisual specification of a diagram, produce a corresponding diagram respecting a given set of drawing conventions, aesthetic criteria, and drawing constraints (see [1] and [2]). Diagrams can be extensively used for displaying visual information [3].

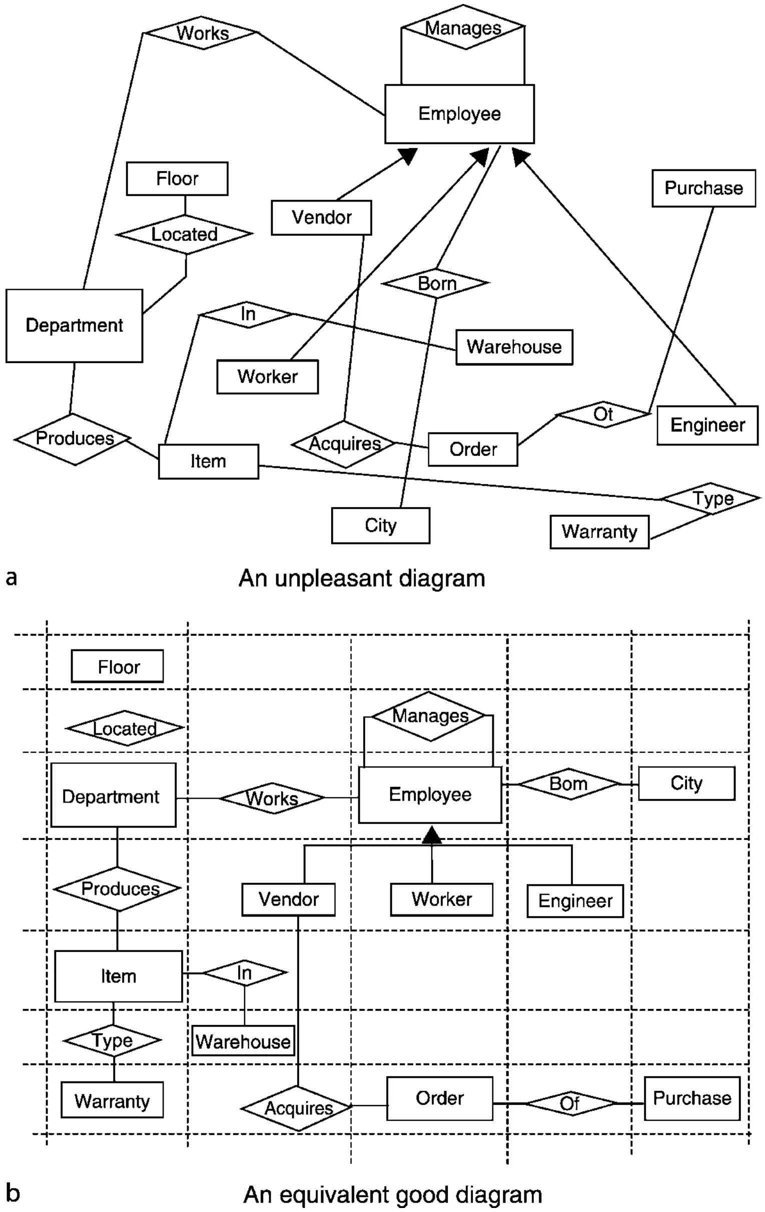


Diagram. Figure 1. Examples of unpleasant and good diagrams.

Cross-references

- ► Chart
- ► Data Visualization
- ► Graph
- ► Visual Formalisms
- ► Visual Representation

Recommended Reading

1. Cruz I.F. and Tamassia R. Graph Drawing Tutorial, http://www.cs.brown.edu/~rt/
2. Di Battista G., Eades P., Tamassia R. and Tollis I.G. Graph Drawing. Prentice-Hall, Englewood Cliffs, NJ, 1999.
3. Tufte E.R. The Visual Display of Quantitative Information. Graphic Press, Cheshire, CT, 1998.

Diagrammatic Representation

► Diagram

Difference

Cristina Sirangelo
University of Edinburgh, Edinburgh, UK

Synonyms

Set-difference

Definition

The difference of two relation instances R_1 and R_2 over the same set of attributes U, denoted by $R_1 - R_2$, is another relation instance over U containing precisely the set of tuples occurring in R_1 and not occurring in R_2.

Key Points

The difference is one of the primitive operators of the relational algebra. It coincides with the notion of set difference, with the additional restriction that it can be applied only to relations over the same set of attributes. However the difference of two arbitrary relations having the same arity can be obtained by first renaming attributes in one of the two relations.

As an example, consider a relation *Students* over attributes (*number, name*), containing tuples {(1001, *Black*),(1002,*White*)}, and a relation *Employees* over attributes (*number, name*), containing tuples {(1001, *Black*),(1003,*Brown*)}. Then the difference *Students* – *Employees* is a relation over attributes (*number, name*) containing the only tuple (1002,*White*).

In the absence of attribute names, the difference is defined on two relations with the same arity: the output is a relation with the same arity as the input, containing the difference of the the sets of tuples in the two input relations.

The number of tuples in the output relation is bounded by the the number of tuples in R_1.

Cross-references

- ▶ Relation
- ▶ Renaming
- ▶ Relational Algebra

Digital Archives and Preservation

Reagan W. Moore
University of California-San Diego, La Jolla, CA, USA

Synonyms

Persistent archives

Definition

Preservation is the set of processes that maintain the authenticity, integrity, and chain of custody of records for long periods of time. Authenticity is defined as the management of provenance information about the creation of the record. Integrity is defined as the ability to create an authentic copy. Chain of custody tracks all processing done to the record, including migration to new storage systems or to new data encoding formats. Digital preservation addresses the challenge of technology evolution by managing preservation properties independently of the choice of software and hardware systems. Preservation properties include the names used to identify archivists, records, and storage systems, and the preservation metadata which includes provenance information and administrative information about record management. The display and manipulation of digital data is maintained through the use of representation information that describes how the record can be parsed and interpreted. Digital preservation environments implement trustworthiness assessment criteria, enabling verification that the preservation properties are maintained over time.

Historical Background

The preservation community bases digital preservation upon the same concepts that are used to preserve paper records. The preservation environment manages the authenticity, integrity, and chain of custody of the digital records. At least four approaches have been implemented that define the digital preservation processes and policies that are needed to maintain authenticity and integrity. (i) Diplomatics defines the required provenance information that describes the institution, event, and ingestion information associated with the documentation of an event. Examples of events are treaties and government communiqués. The records are assumed to be held forever. (ii) The US National Archives and Records Administration associates records with a life-cycle data requirements guide. Each record is associated with a record group (institution), a record series (the set of records governed by a submission agreement), a file unit, and an entity. Each record series has a defined retention and disposition schedule. The arrangement of the records is governed by the submission order in the record series. Standard preservation processes include appraisal, accession, arrangement, description, preservation, and access. (iii) The digital library community manages preservation within the context of a collection, with the required preservation metadata and the arrangement governed by the purpose under which the collection was formed. (iv) The continuum model manages records within an active access environment. Records that are generated for one record series may be used as the basis for generating a new record series. The relationships between records within the multiple record series are tracked as part of their provenance. Each of these four communities imposes the preservation processes and preservation policies required to enforce their goals.

The OAIS standard defines a model for preservation that focuses on record submission, record preservation and record access. The information required for each process is aggregated respectively into a Submission Information Package (SIP), an Archival Information Package (AIP), and a Dissemination Information Package (DIP). An information package contains content (the digital record) and representation information (all information needed to understand the record). The OAIS representation information includes the structure and semantics of the record and links to a knowledge base of a designated community for interpreting the semantic term. The OAIS standard stores Preservation Description Information within an AIP that includes Fixity information, Provenance information, Context information, and Reference information. This approach attempts to provide the context needed both to understand how to display and manipulate a record, and to interpret the meaning of the record.

The management of technology evolution is a major concern for digital data that is addressed outside of the OAIS model [3]. The storage technology may impose a proprietary referencing mechanism on the data that locks the record onto a single vendor's system. When that vendor goes out of business, access to the records may be lost. Display applications may no longer be available for reading a record. Even though the record can be accessed, it may not be possible to interpret the internal structure correctly. A preservation environment is the interface between the fixed records and the changing external world. A digital archive must ensure that the records can be viewed in the future, while maintaining authenticity, integrity, and chain of custody.

Foundations

Preservation is an active process, starting with the extraction of records from the environment in which they were created. Each record documents an event, or the materials upon which decisions have been made, or the material that is used to support a research or business or legislative objective. The extraction processes start with appraisal, the determination of which records are worthy of preservation. A formal accession process documents the ingestion of the records into the preservation environment, along with the provenance information needed to identify the source of the records, the representation information needed to interpret and understand the records, and the administrative information needed to document the ingestion process. Once the records are under the control of the archivist, a description process organizes the provenance information and an arrangement process organizes the records. A preservation process creates archival information packages that link the preservation information to the record and stores the records. An access process provides mechanisms to discover, display, and manipulate the records.

Preservation is a form of communication with the future. Since the technology that will be used in the future is expected to be more sophisticated and more cost effective than present technology, this appears to be an intractable situation. How can records that are archived today be accessible through arbitrary choices of technology in the future? By viewing preservation as an active process, this challenge can be addressed. At the point in time when new technology is being assimilated into the preservation infrastructure, both the old and new technologies are present. Infrastructure that supports interoperability across multiple versions of software and hardware systems make it possible to evolve preservation environments over time. The approach is called infrastructure independence, and is based on data virtualization, trust virtualization, and management virtualization. The preservation environment is the set of software that enables management of persistent archives independently of the choice of storage system, information syntax, access protocol, discovery mechanism, and display service. Preservation environments insulate records from changes in the environment, while holding fixed the set of standard preservation processes and the name spaces used to identify records, archivists, and storage systems.

Data virtualization consists of two components: (i) persistent name spaces for identifying records, preservation metadata, archivists, and storage systems, (ii) standard operations for interacting with storage repository protocols. Data grid technology provides both components, enabling the integration of new storage technology into a preservation environment, the use of new access protocols for interacting with records, and the migration of records to new encoding syntax.

Trust virtualization is the management of authorization independently of the choice of preservation technology. Data grids provide access controls that enforce relationships between any pair of persistent name spaces. For example, authorization may be a constraint imposed on the archivist identity and the record identity, or a constraint imposed on preservation metadata and the archivist identity, or a constraint based on record identity and storage identity. The constraints may be applied across multiple types of data management environments (file systems, tape archives, databases). Since the name spaces are managed by the data grid, the constraints remain invariant as the records are moved between storage systems within the preservation environment.

Preservation manages communication from the past in order to make assertions about preservation properties such as authenticity, integrity, and chain of custody. For an assertion to be verifiable, a preservation environment must document the preservation processes that have been applied to each record, and the preservation policies that controlled the application of the preservation processes. Unless both processes and policies can be tracked over time, an assertion cannot be verified. The outcome of the application of each preservation process on a record should be recorded as state information that is linked to the record. Assertions can then be validated as queries on the state information that verify that the desired property has been conserved. The processes, policies, and state information constitute representation information about the preservation environment.

To manage the evolution of data formats, three approaches are pursued: Emulation in which the original display application is ported to future operating systems; transformative migration in which the encoding format of the record is changed to a future encoding format; and persistent objects in which the structure and relationships present within the record are characterized and migrated to future information and knowledge representations. Operations that can be performed upon relationships defined between structures can be applied in the future by any application that knows how to manipulate that specific relationship. An example is a query on time track change relationships embedded in a Microsoft Word document.

Key Applications

Multiple technologies can be used to implement the preservation environment. However no single technology provides all of the capabilities that are required. A preservation environment is an integration of management systems, software systems, and hardware systems that isolate records from changes in technology, while preserving authenticity, integrity, and chain of custody. Technologies that implement subsets of the capabilities needed for infrastructure independence include:

- SRB – Storage Resource Broker data grid (http://www.sdsc.edu/srb). The SRB implements the name spaces needed for infrastructure independence, and manages descriptive metadata that can be associated with each collection and file. Collections stored in the SRB include observational data, simulation output, experimental data, educational material, office products, images, web crawls, and real-time sensor data streams. The SRB data grid enables the creation of shared collections from data distributed across multiple administrative domains, institutions, and nations. Data may be stored in file systems, archives, object-relational databases, and object ring buffers. International data grids have been based on the SRB technology through which hundreds of terabytes of data have been replicated.

  The system is designed to scale to petabytes of data and hundreds of millions of files. Applications of the technology include use as data grids for sharing data, digital libraries for publishing data, and persistent archives for preserving data.

  The set of standard operations that are performed upon remote storage systems include Posix file system I/O commands (such as open, close, read, write, seek, stat, . . .) and operations needed to mask wide-area network latencies. The extended operations include support for aggregation of files into containers, bulk operations for moving and registering files, parallel I/O streams, and remote procedures that parse and subset files directly at the remote storage system.
- LOCKSS-Lots of Copies Keep Stuff Safe (http://www.lockss.org/lockss/Home). The LOCKSS system manages attributes on files that may be distributed

across multiple storage systems. The original design of the system focused on management of data distributed by publishers. The original copy of the file was downloaded from a publisher through a security module that supported the publisher authentication requirements. Attributes were then associated with each file to track the publication source. LOCKSS systems that had retrieved data from the same publisher could then provide disaster recovery copies to each other. Types of access that are supported include file retrieval. The system is designed to scale to about 20 Tera bytes of archived data.

- IBP – Internet Backplane Protocol (http://loci.cs.utk.edu/ibp/). The IBP was designed to enable applications to treat the Internet as if it were a processor backplane. The IBP replicates or caches blocks of data across distributed storage systems at multiple sites. A file system such as LSTORE (http://www.lstore.org/pwiki/pmwiki.php) is implemented on top of the IBP protocol to support the required persistent file naming.
- DSpace (http://www.dspace.org/). This is a digital library that provides standard preservation services for accessing, managing and preserving scholarly works. DSpace can store files on the local file system, or in the SRB data grid.
- FEDORA – Flexible Extensible Digital Object and Repository Architecture (http://www.fedora-commons.org/). This is digital library middleware that supports the characterization of relationships between records. The relationships may be queried to identify desired data subsets. Services support the creation, management, publication, sharing, annotation, and preservation of digital content.
- UVC-Universal Virtual Computer (http://en.wikipedia.org/wiki/Universal_Virtual_Computer). Thi s is a software system that provides standard operations that can be migrated onto future operating systems. The environment supports a Logical Data Schema for type description, a format decoder, and a Logical Data Viewer for displaying the parsed files.
- ADORE (http://african.lanl.gov/aDORe/projects/adoreArchive/). This is a write-once/read-many preservation system for digital objects. XML-based representations of digital objects are concatenated into a single, valid XML file called XMLtape. The associated data streams are aggregated into Internet Archive ARC files. Each XMLtape is accessed through the Open Archives Initiative – Protocol for Metadata Harvesting. The ARC files are accessed through OpenURL.

Most of the above systems support discovery and access to the record. A subset supports updates to the records, schema extension for provenance metadata, replicas, and bulk operations. The system managing the largest amount of material is the SRB. The SRB is used in the NARA Transcontinental Persistent Archive Prototype (http://www.sdsc.edu/NARA/) as a research tool for the investigation of properties that should be supported by a preservation environment.

Systems that manage representation information characterize records by file format type. The display and access of the record are accomplished by identifying an application that is capable of parsing the record format. Example systems range from stand alone environments to web services to highly integrated environments. Examples include:

- MVD – Multivalent Document (http://elib.cs.berkeley.edu/ib/about.html). This system presumes that a single document comprises multiple layers of related material. Media adaptors parse each layer. Behaviors that manipulate the parsed data are dynamically loaded program objects. It is possible to add new behaviors independently of the media adaptors to provide new operations for manipulating or viewing the layers.
- DFDL-Data Format Description Language (http://forge.ggf.org/sf/projects/dfdl-wg). This is an Open Grid Forum standards effort that characterizes the mapping of bit-steams to structures through creation of an associated XML file. This is the essential capability needed to interpret an arbitrary file. The structures can be named.
- EAST (http://nssdc.gsfc.nasa.gov/nssdc_news/mar02/EAST.html). EAST is a data description language that supplies information about the format of the described data. EAST is designed for building descriptions of data that are maintained separately from the data itself.
- CASPAR – Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval (http://www.casparpreserves.eu/). This is a research project to identify the representation information required to understand digital objects. This includes not only the data format types, but also the designated community that will use the data, and the knowledge base that defines the required semantic terms.

- METS–Metadata Encoding and Transmission Standard (http://www.loc.gov/standards/mets/). This is a standard for encoding descriptive, administrative, and structural metadata regarding objects within a digital library.

As pointed out by the CASPAR project, the ability to interpret the representation information can require additional representation information. In order to close this recursion, a designated community is defined that understands how to interpret the semantics of the final set of representation information by accessing a community knowledge base.

Future Directions

Rule-based data management systems have the potential to virtualize management policies by providing infrastructure that guarantees the application of the policies directly at the remote storage systems independently of the remote administrative domain. The iRODS (integrated Rule Oriented Data Systems) data grid (http://irods.sdsc.edu) installs software middleware (servers) at each remote storage system. Each server includes a rule engine that controls the execution of micro-services at that storage system. The rules are expressed as event: condition: action-sets: recovery-sets, where the condition can include operations on any of the persistent state information that is managed by the data grid. The action-sets can include micro-services or rules, enabling the aggregation of micro-services into an execution hierarchy. For each action, a recovery procedure is specified to enable the tracking of transaction semantics. Additional name spaces are required that include:

- The names of the micro-services that aggregate the standard operations into well-defined functions.
- The names of the rules that control the execution of the micro-services.
- The persistent state information that tracks the results of applying the operations (think of the location of a replica as persistent state that must be saved).

Rule-based systems support asynchronous operations. Micro-services can be queued for deferred or periodic operations. Thus a recovery-set might include the scheduling of a deferred attempt to complete the operation, followed by an e-mail message if that attempt does not succeed, or it might roll-back any changed state information and report failure.

Rule-based systems make it possible to characterize explicitly the set of management policies that control the preservation environment. This includes the rules that govern data integrity (replication, data distribution, media migration), the rules that assert the presence of required descriptive or provenance metadata (including the extraction of the required metadata from an authoritative source), and the rules that govern chain of custody (assignment of access controls and parsing of audit trails to verify the enforcement). Rule-based systems also explicitly characterize the preservation processes as sets of named micro-services that can be ported to new operating systems. In effect, a rule-based data management system is able to build an emulation environment for the set of management policies and preservation processes that the archivist wants to apply. The environment guarantees that the desired policies and processes will continue to control the preservation environment even when new technology is incorporated into the system.

The identification of standard preservation management policies is being attempted through the RLG/NARA trustworthiness assessment criteria [4]. A mapping of these assessment criteria to iRODS rules is possible, and identifies some 105 rules that are required to enforce and verify preservation criteria. The NARA Electronic Records Archive has defined the set of capabilities that they require for long-term preservation [1]. These capabilities have also been mapped to iRODS rules and micro-services. The goal is to build the set of management principles and fundamental preservation processes required for long term preservation [2].

When representation information for preservation environments is available, it may be possible to design a theory of digital preservation. The components will include:

- ► Characterization of the representation information for the preservation environment
  - Definition of the properties that the preservation environment should conserve
  - Definition of the management policies that enforce the conservation of the desired properties
  - Definition of the capabilities (preservation processes) needed to apply the management policies
- ► Analysis that the system is complete
  - Demonstration that assessment criteria can be mapped to queries on persistent state information that are managed independently of the choice of technology

  - Demonstration that these management policies can be mapped to well-defined rules
  - Demonstration that the rules control the execution of well-defined micro-services that are independent of the choice of preservation technology
- ▶ Analysis that the system is closed
  - Demonstration that the state persistent information required to validate assessment criteria are generated by each micro-service
  - Demonstration that for every micro-service the associated persistent state information is updated on each successful operation

A theory of digital preservation defines the processes required to assert that the preservation environment has been implemented correctly and will successfully enable long-term preservation.

Cross-references

▶ Archiving Experimental Data
▶ Data Warehouse
▶ Disaster Recovery
▶ Information Lifecycle Management
▶ LOC METS
▶ Metadata Repository
▶ Provenance
▶ Replication

Recommended Reading

1. Electronic Records Archive capabilities list defines a comprehensive set of capabilities needed to implement a preservation environment, and can be examined at http://www.archives.gov/era/pdf/requirements-amend0001.pdf.
2. Moore R. Building preservation environments with data grid technology. Am Archivist, 69(1):139–158, 2006.
3. OAIS, Reference Model for an Open Archival Information System, ISO 14721:2003.
4. RLG/NARA TRAC – Trustworthy Repositories Audit and Certification: Criteria and Checklist. http://wiki.digitalrepositoryauditandcertification.org/pub/Main/ReferenceInputDocuments/trac.pdf.

Digital Curation

GREG JANÉE
University of California-Santa Barbara, Santa Barbara, CA, USA

Synonyms

Stewardship

Definition

Digital curation is the activity of maintaining and adding value to a trusted body of digital information for current and future use.

Key Points

Left unattended, digital information degrades over time. Even if the information's bits are correctly preserved (a difficult task in itself) the technological context surrounding the bits – the computing platforms, programming languages, applications, file formats, and so forth – will change sufficiently over time until the information is no longer usable. Changes in the information's social context are just as significant. The communities and organizations involved in the information's initial creation and use may attach different values and interpretation to the information over time, or cease to exist altogether. And the passage of time only exacerbates contemporary problems such as establishing the authenticity, quality, and provenance of the information.

Curation is the activity of maintaining a body of information so that it remains usable over time. Curation covers the entire lifecycle of the information, from creation to contemporary use, from archival to reuse. Specific curation activities include: selection and appraisal; capture of metadata and the information's larger technological, scientific, and social contexts; conversion to archival formats; establishment and maintenance of authenticity and provenance; annotation and linkage; provisioning for secure and redundant storage; transformation, migration, and emulation as needed over time; discoverability in contemporary search systems; creation of meaningful access mechanisms; and recontextualization.

Different types of information bring different curation requirements and present different challenges. Information intended for direct human consumption, such as many textual and multimedia documents, may only need to be migrated to new formats as older formats fall out of favor. But data, particularly scientific data, may require significant reprocessing and transformation. For example, climatalogical observations may need to be periodically recalibrated to support long-term longitudinal studies, a process requiring deep understanding and emulation of the original calibration.

Cross-references

▶ Preservation

Recommended Reading

1. Beagrie N. Digital curation for science, digital libraries, and individuals. Int. J. Digital Curat., 1(1), 2006.
2. Consultative Committee for Space Data Systems. Reference Model for an Open Archival Information System (OAIS). ISO 14721:2003, 2002.
3. Trustworthy Repositories Audit and Certification: Criteria and Checklist. Center for Research Libraries, 2007.

Digital Elevation Models

LEILA DE FLORIANI, PAOLA MAGILLO
University of Genova, Genova, Italy

Synonyms

Digital Terrain Model (DTM); Digital Surface Model DEMs

Definition

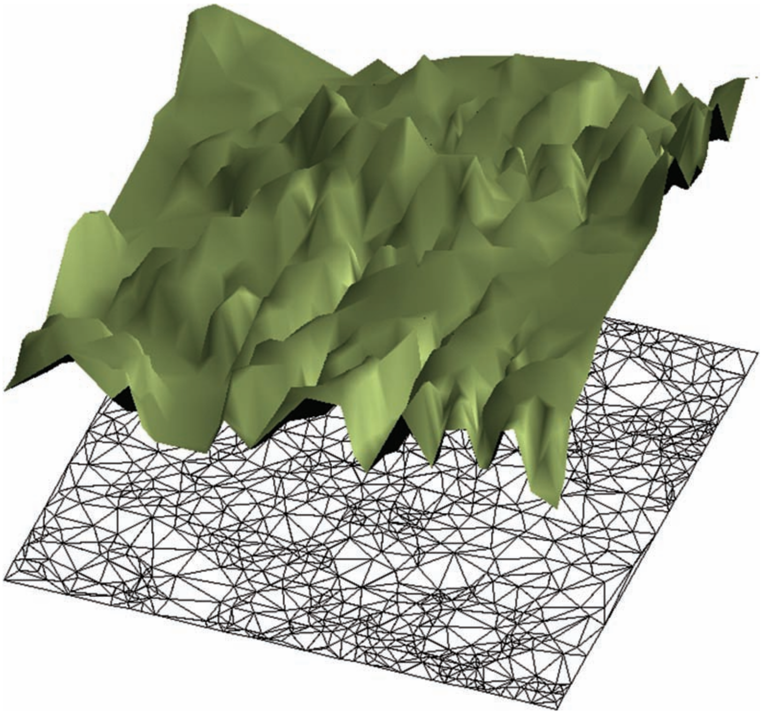
A Digital Elevation Model (DEM) represents the 3D shape of a terrain in a digital format. A terrain is mathematically modeled as a function $z = f(x, y)$ which maps each point (x, y) in a planar domain D into an elevation value $f(x, y)$. In this view, the terrain is the graph of function f over D.

In practice, a terrain is known at a finite set of points within D, which may (i) lie at the vertices of a *regular grid*, (ii) be scattered, or (iii) belong to *contour lines* (also known as *isolines*), i.e., the intersections of the terrain surface with a sequence of horizontal planes.

In case (i), the DEM consists of the grid structure plus elevation values at its vertices. This is called a *Regular Square Grid* (RSG). Within each grid cell, terrain elevation either is defined as constant, or it is modeled by a function, which can be linear (this involves cell decomposition in two triangles), or quadratic (usually, bilinear).

In case (ii), usually the DEM is defined based on a triangle mesh joining the data points in D and by a piecewise-linear function interpolating elevations at the three vertices of each triangle. This gives a *Triangulated Irregular Network* (TIN) (see Fig. 1).

In case (iii), the DEM consists of the polygonal lines forming each contour, plus the corresponding elevation, and the containment relation between contours at consecutive elevations. This provides a *contour map*.



Digital Elevation Models. Figure 1. A triangle-based terrain representation.

Historical Background

Historically, terrain models were represented as three-dimensional relief maps, generally constructed for military or educational purposes from plaster, papier-marché, or vinyl plastic. For instance, such models were used extensively by military forces during World War II. Contour maps drawn on sheets of paper have probably been the most common form of terrain model. In the early ages of computer-based Geographic Information Systems, these maps were converted into digital format through scanning devices. DEMs based on contour lines are a way for representing a terrain, but not for performing computations, or simulations. For other applications, they are usually converted into triangulated models (TINs) by connecting two consecutive contour lines through a set of triangles.

The first DEMs of the computer age were *Regular Square Grids (RSGs)*. Very large and accurate gridded DEMs are usually acquired through remote sensing techniques, and are built from aerial or satellite raster images. Thanks to the regular structure of an RSG, both storing and processing an RSG is simple from the point of view of design, but the huge size of such models may cause serious inefficiency in both storage and manipulation. This problem can be addressed by applying techniques for terrain generalization, or multi-resolution models. *Generalization* means decimating data to achieve smaller memory size and faster processing time, at the expense of less accuracy. This can be achieved through grid subsampling or selection of meaningful vertices to build a TIN. *Multi-resolution* refers to the capability of maintaining multiple accuracy levels at the same time, and selecting the most appropriate one for the current working session. This latter aspect is covered in Multiresolution Terrain Modeling by Eurico Puppo.

Triangulated Irregular Networks (TINs) are the most general model from the point of view of modeling power, since they do not assume any spatial distribution of the data. TINs can encompass morphological terrain features, *point features* (e.g., peaks, pits and passes) and *line features* (e.g., coast lines, rivers, contour lines). On the other hand, the internal representations for TINs and the algorithms for their construction are more complex than those for RSGs. These latter have been extensively studied within the field of *computational geometry*. Storing and manipulating TINs require more computational resources than RGSs, for the same number of vertices, but they may need much fewer vertices to achieve the same accuracy. This is especially true for terrains with an irregular morphology, since a TIN can adapt the density of the mesh to the local complexity of the terrain.

Foundations

There are two major categories of DEMs: Regular Square Grids (RSGs) and Triangulated Irregular Networks (TINs).

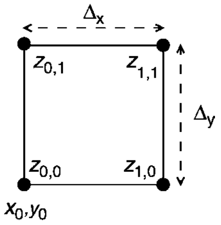
RSGs are based on a domain subdivision into a square grid. There are two major ways of approximating a terrain based on a grid. In a *stepped model*, each data point lies at the center of a grid cell, and its elevation is assigned to the whole cell. The resulting terrain model has the shape of a 2D histogram, and thus the surface presents jump discontinuities at cell edges. The second approach produces a continuous surface. The data points are the vertices of the grid cells. Within each cell, the elevation is approximated through a function that interpolates the known elevations of the four cell vertices. Let (x_0, y_0) be the coordinates of the lower left corner of the cell, (Δ_x, Δ_y) be the cell size and $z_{0,0}$,$z_{0,1}$,$z_{1,0}$ and $z_{1,1}$ be the elevations of its four corners (lower-left, upper-left, lower-right, upper-right, respectively; see Fig. 2).

A bilinear interpolant estimates the elevation at a point $P = (x, y)$ within the cell as:

$$\begin{aligned} z = {} & z_{0,0} + (z_{0,1} - z_{0,0})(y - y_0)/\Delta_y + \\ & (z_{1,0} - z_{0,0})(x - x_0)/\Delta_x + \\ & (z_{1,1} - z_{0,1} - z_{1,0} + z_{0,0})(x - x_0)(y - y_0)/\Delta_x\Delta_y \end{aligned}$$

and provides a continuous (but not differentiable) surface approximation.

RSGs are stored in a very simple data structure that encodes just a matrix of elevation values, the grid being implicitly represented. All other information (including interpolating functions and neighbor relations among cells) can be reconstructed in constant time.



Digital Elevation Models. Figure 2. A cell in an RSG.

Moreover, the regular structure of RSGs makes them well suited to parallel processing algorithms.

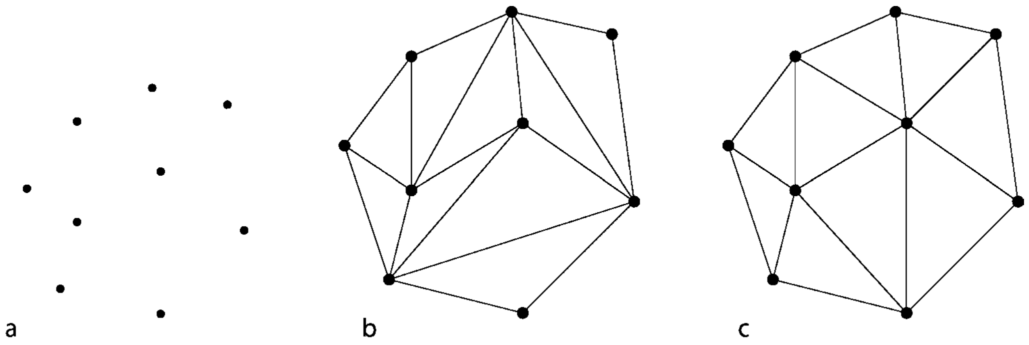
The RSG is the main format for distributing elevation data. Many existing DEMs are provided in this format, including USGS (United States Geological Survey [11]) data as well as many proprietary GIS formats (and the interchange Arc/Info ASCII grid). Usually, the file contains a header followed by the elevations values listed in either row or column order. The header contains the information needed to decode the given values and to locate the grid on the Earth surface (geo-referencing). An RGS could also be encoded in a standard image format by mapping the elevation range to a grey level value, but this format does not support geo-referencing.

The main disadvantage of an RSG is its uniform resolution (i.e., the same cell size) over the whole domain. This may imply undersampling in raw areas and oversampling in flat areas for a terrain with irregular morphology. Uniformly increasing the resolution produces huge matrices and thus high storage and processing costs. Adaptive nested grids have been proposed to overcome this problem.

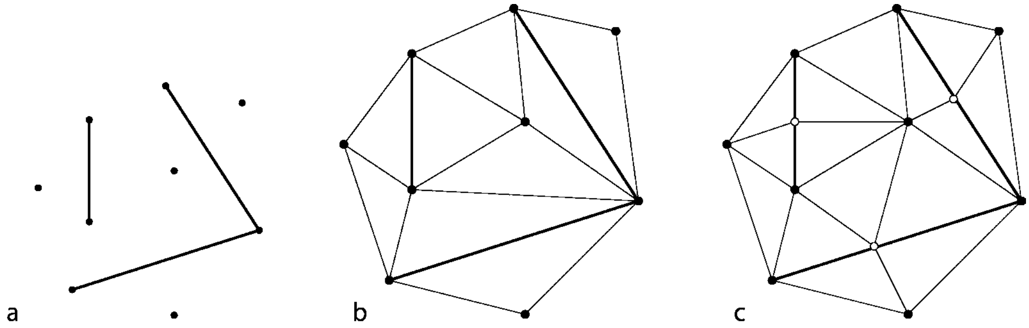
A TIN is based on an irregular domain subdivision, in which the cells are triangles, i.e., a triangle mesh with vertices at the set S of data points. Usually, a linear interpolating function is defined on the triangles of the mesh, thus providing a continuous terrain approximation. More precisely, a *triangle mesh* T consists of a set of triangles such that: (i) the set of vertices of T coincides with S, (ii) the interiors of any two triangles of T do not intersect, (iii) if the boundaries of two triangles intersect, then the intersection is either a common vertex, or a common edge (see Fig. 3). Each triangle of T is mapped in the three-dimensional space by considering the elevation values at its three vertices and the plane passing through the resulting three points in 3D space.

The quality of the terrain approximation provided by a TIN depends on the quality of the underlying triangle mesh, since the triangulation of a set of points is not unique. The most widely used triangle mesh is the *Delaunay* one. A Delaunay mesh is the one among all possible triangle meshes joining a given set of points in which the circumcircle of each triangle does not contain any data point in its interior. This means that the triangles of a Delaunay mesh are as much equiangular as possible, within the constraints imposed by the distribution of the data points [9]. It has also been proved that the use of a Delaunay mesh as the basis for a TIN improves the quality of the terrain approximation and enhances numerical stability in computations. Other triangulation criteria have been proposed which consider not only the 2D triangulation, but also the corresponding triangles in 3D space [4].

In many practical cases, a TIN must embed not only points, but also lines representing morphological terrain features (coast lines, rivers, ridges), man-made structures (roads, railways, gas lines), political or administrative boundaries, contour lines. The Delaunay criterion has been modified to deal with such lines in two different ways: (i) in the *constrained Delaunay triangulation*, the given lines appear as triangle edges [3]; (ii) in the *conforming Delaunay triangulation*, each line is discretized as a dense set of points [5] (see Fig. 4). The constrained Delaunay triangulation may present sliver triangles when segments are too long. Conforming triangulations may add a very large number of points in order to force the lines to be included in the resulting mesh.



Digital Elevation Models. Figure 3. (a) A set of data points; (b) a triangle mesh; (c) Delaunay triangle mesh.



Digital Elevation Models. Figure 4. (a) A set of data points and lines; (b) Constrained Delaunay triangulation; (c) A conforming Delaunay triangulation.

The simplest storage format for a TIN is a *triangle soup*: each triangle is represented separately by listing the nine coordinates of its three vertices. The *indexed format* has been designed to avoid replicating the coordinates of vertices shared by several triangles incident in the same vertex. It stores each vertex of the TIN, as three coordinates, in a list of vertices, and each triangle as three vertex indices within such a list. This latter format can be enriched by adding the *adjacency relation* linking each triangle with the three triangles its shares an edge with. Triangle adjacency links support efficient navigation inside a TIN. To support an efficient traversal of the mesh passing through vertices, it is convenient to attach to each vertex the index of one of its incident triangles. The encoding of a TIN in an indexed format requires one half of the space of a triangle soup. The indexed format with adjacencies requires about 2/3 of the space of a triangle soup.

Data structures and algorithms for TINs [1,3,12] are more complex than those for RSGs. But, in many cases, a TIN can reach the same approximation error as an RGS in terrain representation with a much smaller number of vertices. The main advantage of a TIN is its flexibility in adapting the density of sampling in the case of a terrain with irregular morphology, to include relevant lines or points, and fit to irregularly shaped domains. Many multi-resolution terrain models are TIN-based.

Key Applications

Visualization of terrains is needed in many fields including environmental sciences, architecture, entertainment.

Morphology analysis, which is concerned with the extraction of ridges, rivers, water basins, is important in environmental monitoring and planning.

Visibility analysis of a terrain is concerned with the computation of visible, or invisible, areas from a set of viewpoints, with the computation of hidden or scenic paths, with the extraction of networks of mutually visible points. This is useful in many applications such as communication, surveillance, visual impact of infrastructures, etc.

Applications may require computing *paths or point networks* of minimum cost according to some criteria combining visibility, length, height variation, etc. A wide range of simulations (e.g., flood, erosion, pollution, etc.) are also possible on a terrain model.

For details, see [3,7,10].

Future Directions

Some geographical applications need to represent not only the surface of the earth, but also its internal structure. This requires modeling 3D volumes as well as 2D surfaces (e.g., boundaries between two rock layers), and their adjacency relations. 3D extensions of digital elevation models, such as RSGs and TINs, leads to regular volume models and irregular tetrahedral meshes, respectively. Regular volume models are grids of hexahedral cells connecting the data points, while irregular tetrahedral meshes are meshes formed by tetrahedra joining the data points. Thus, challenging issues arise here, such as the development of compact and effective data structures for encoding digital volumetric models and of efficient algorithms for building and manipulating such models.

Another field for 3D terrain modeling is *urban terrain modeling*, which provides the integration between elevation and urban data, i.e., laying buildings and other landscape or vegetation elements over a terrain. With urban terrain models, high-quality photorealistic rendering of 3D geovirtual environments can be achieved. These are used in common products like GoogleEarth [11] or Virtual Earth [6]. Urban terrain modeling has also important applications to 3D town maps for business and entertainment, and also for city administration and urban development planning.

Another challenging field is incorporating *time* in a DEM. This will allow modeling the evolution of a terrain over time and it is relevant for both historical record and simulation. From a mathematical point of view, the reference model is no longer a 2D surface embedded in 3D space, but a 3D volume embedded in 4D space (where the last dimension is time). Here, it is necessary to define and develop digital volumetric models, like regular models and irregular tetrahedral meshes, but embedded in 4D space.

In all such applications, multi-resolution models will play an important role, because of the even larger size of the data sets and of the corresponding volumetric models.

URL to Code

United States Geological Survey (USGS) home page, http://www.usgs.gov/
Google Earth home page, http://earth.google.com/
Microsoft Virtual Earth home page, http://www.microsoft.com/virtualearth/
Spatial Data Transfer Standard (SDTS) home page, http://mcmcweb.er.usgs.gov/sdts/

Cross-references

► GIS
► Multi-Resolution Terrain Modeling
► Three-Dimensional GIS and Geological Applications
► Triangulated Irregular Network

Recommended Reading

1. de Berg M., van Kreveld M., Overmars M., and Schwarzkopf O. Computational Geometry – Algorithms and Applications. 2nd edn. Springer, Berlin, 2000.
2. De Floriani L., Magillo P., and Puppo E., Applications of computational geometry to Geographic Information Systems. In Handbook of Computational Geometry, Chap. 7, J.R. Sack, J. Urrutia (eds.). Elsevier Science, 1999, pp. 333–388.
3. De Floriani L. and Puppo E. An On-line Algorithm for Constrained Delaunay Triangulation, Graphical Models and Image Processing, 54(4):290–300, 1992.
4. Dyn N., Levin D., and Rippa S. Data dependent triangulations for piecewise linear interpolation, IMA J. Numer. Analy., 10:137–154, 1990.
5. Edelsbrunner H. and Tan T.S. An upper bound for conforming Delaunay triangulation: Discrete Comput. Geom., 10:197–213, 1993.
6. Google Earth home page, http://earth.google.com/
7. Longley P.A., Goodchild M.F., Maguire D.J., and Rhind D.W. (eds.) Geographical Information Systems, 2nd edn. Wiley, New York, 1999.
8. Microsoft Virtual Earth home page, http://www.microsoft.com/virtualearth/
9. O'Rourke J., Computational Geometry in C, 2nd edn. Cambridge University Press, Cambridge, 1998.
10. Peckham R.J. and Jordan G. (eds.). Digital Terrain Modelling – Development and Applications in a Policy Support Environment, Lecture Notes in Geoinformation and Cartography, Springer, Berlin, 2007.
11. United States Geological Survey (USGS) home page, http://www.usgs.gov/
12. van Kreveld M. Digital elevation models and TIN algorithms. In Algorithmic Foundations of Geographic Information Systems, number 1340 in Lecture Notes in Computer Science (tutorials), M. van Kreveld, J . Nievergelt, T. Roos, and P. Widmayer (eds.). Springer, Berlin, 1997, pp. 37–78.

Digital Image

► Image

Digital Libraries

Venkat Srinivasan, Seungwon Yang,
Edward A. Fox
Virginia Tech, Blacksburg, VA, USA

Synonyms

Electronic libraries

Definition

Digital libraries (DLs) are complex information systems that have facilities for storage, retrieval, delivery, and presentation of digital information. They are complex in nature because of the broad range of activities they may need to perform, and because they may need to serve multiple types of audiences. Thus, the broadest definitions include the people and agents/actors involved, as well as the software, content, structure/organization(s), services, policies, procedures, etc.

Historical Background

One of the earliest detailed works about digital libraries (DLs) was prepared by Licklider [8], who envisioned a network of computers with digitized versions of all of the literature ever published. However, the term "digital library" became widely used only around 1991, in connection with a series of workshops funded by the US National Science Foundation, which later led to significant NSF support of R&D DL projects (http://www.dli2.nsf.gov/), e.g., Informedia, which focused on digital video [2].

In a Delphi study of digital libraries, Kochtanek et al. [7] defined a digital library as an "organized collection of resources, mechanisms for browsing and searching, distributed networked environments, and sets of services objectified to meet users' needs." The President's Information Technology Advisory Committee (PITAC) report [12] mentioned "These new libraries offer digital versions of traditional library, museum, and archive holdings, including text, documents, video, sound, and images. But they also provide powerful new technological capabilities that enable users to refine their inquiries, analyze the results, and change the form of the information to interact with it...".

Starting in 2005, the European Union, through the DELOS Network of Excellence on Digital Libraries (http://www.delos.info/) worked to develop a reference model for the digital library field. Their DL manifesto [1] defines a digital library as "a (potentially virtual) organization that comprehensively collects, manages, and preserves for the long term rich digital content and offers to its user communities specialized functionality on that content, of measurable quality, and according to prescribed policies."

Foundations

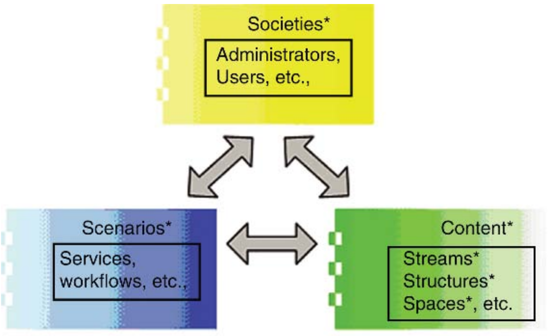
While the DELOS Reference Model [1] has aimed to identify the key constructs of a DL, in order to allow standardization and interoperability, there is one other effort that has tried to develop a formal foundation for the DL field. The 5S framework [4,5] was developed in the Digital Library Research Laboratory at Virginia Tech as a scientific base for digital libraries. There are five elements that underlie DL systems; these can be described (informally) as:

1. Streams: all types of content, as well as communications and flows over networks or into sensors, or sense perceptions. Examples include: text, video, audio, and image. These can be formalized as a sequence.
2. Structures: organizational schemes, including data structures, databases, and knowledge representations. Examples include: collection, catalog, hypertext, and document metadata. These can be formalized as a graph, with labels and a labeling function.
3. Spaces: 2D and 3D interfaces, GIS data, and representations of documents and queries. Examples include: storage spaces used in indexing, browsing, or searching services, as well as interfaces. These can be formalized as a set with operations (vector, topological, measurable, measure, and probability spaces).
4. Scenarios: system states and events, or situations of use by human users or machine processes, yielding services or transformations of data. Examples include: searching, browsing, and recommending. These can be formalized as a sequence of related transition events on a state set.
5. Societies: both software "service managers" and fairly generic "actors" who could be (collaborating) human users. Examples include: service managers (software), actors (learners, teachers, etc.) [1,4,5]. These can be formalized as a pair (i.e., a set of communities and a set of relationships).

A formal description can be found in [4].

DL systems encompass all the five Ss. The 5 Ss are also used in a formal definition of a minimal DL [5], which has the key constructs that most would agree must be found in any DL system. Of course, most real DLs are extended well beyond what constitutes a minimal DL, to better suit the needs of the users. Accordingly, the 5S framework has led to a growing set of meta-models for different types of DLs, each formally defined from a minimalist perspective: archaeological DL, image DL, personal DL, practical DL, and superimposed information DL (supporting annotations and knowledge management of the annotations and base information).

DLs also can be understood as a triad (see Fig. 1), which consists of content, societies, and scenarios. Digital libraries preserve and provide the "content" which is stored in various formats (Streams). The contents have certain "Structures" to help DLs efficiently serve their patrons and to help the administrators manage them. "Spaces" are required to store the content in a DL system. A DL system's user interface is the "Space" in which the patrons and the system



* 58 framework (streams, structures, scenarios, societies, spaces)

Digital Libraries. Figure 1. 5S framework [4,5] represented as a triad.

interact, to submit, download, share, and discuss about the digital contents.

To perform a certain task, a series of steps is needed. The interaction between the system and its patrons involves "Scenarios," which often are described as workflows. Processes to be performed by the system also belong to this category. For the "Societies" in the triad, the people involved with any kind of DL activity would fit into this category.

Building Digital Libraries

Building a DL is not an exact science. The historical absence of formal models to guide the development of DLs has led to divergence and duplication of efforts. Also, interoperability has been a problem. There have been many distributed, heterogeneous, and federated DLs, e.g., NCSTRL (http://www.ncstrl.org) or the Alexandria Digital Library [14], which were built using ad hoc principles, wherein interoperability was achieved only on a case by case basis. Standards have been defined to help achieve interoperability, for example the Open Archives Initiative (OAI), (http://www.openarchives.org/) Protocol for Metadata Harvesting. Clearly there are tradeoffs between encouraging innovation and technological improvement, between functionality enhancement and autonomy in creating DLs – and compromises made so as to achieve interoperability.

Fortunately, a number of toolkits have been developed to aid those developing DLs. One of the first was the EPrints system (http://www.eprints.org), initially supporting electronic pre-prints, but later enhanced to assist with the growing movement toward institutional repositories.

Two other popular systems used to build DLs are DSpace (http://www.dspace.org/) Greenstone (http://www.greenstone.org) and Fedora (http://www.fedora.info/). DSpace can simply be used out of the box to build a DL. Figure 2 gives a high-level summary of the parts and architecture of DSpace.

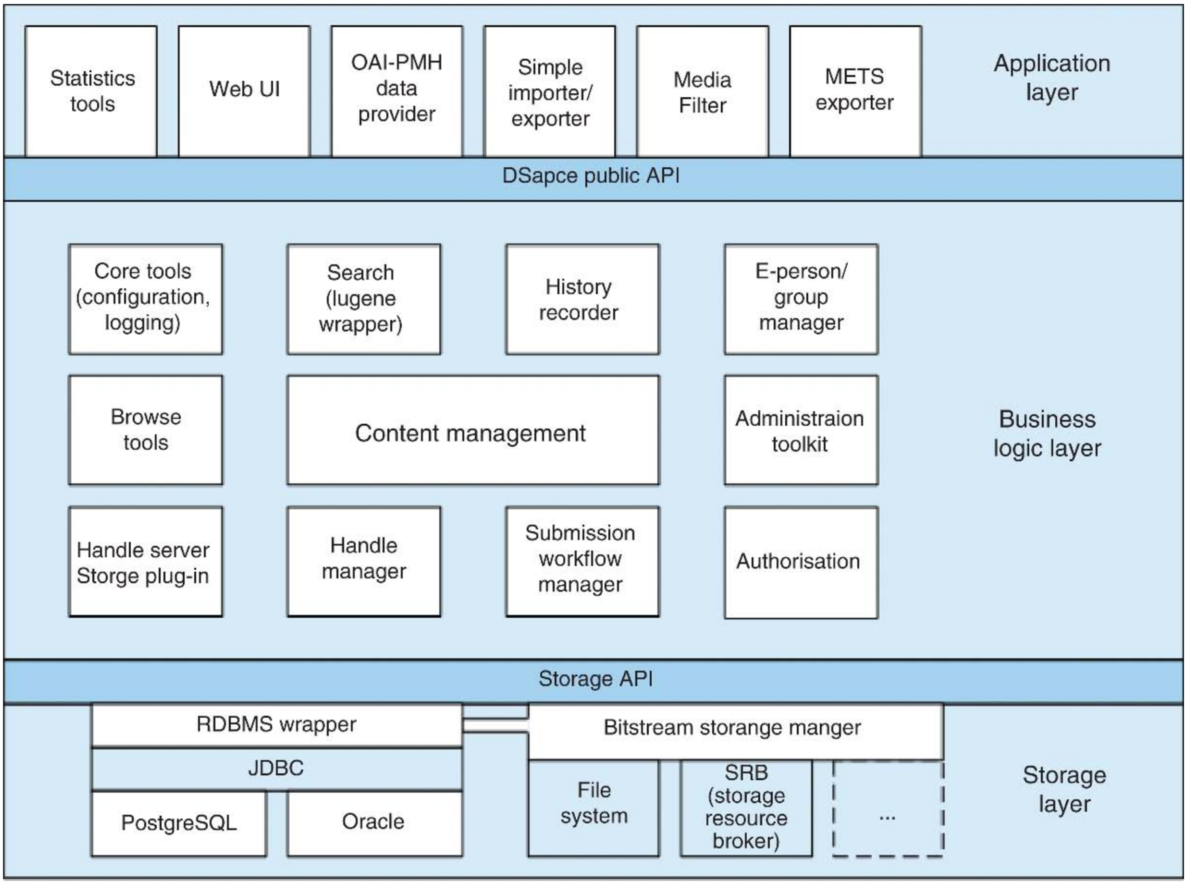
Digital library systems support access through an application layer (see top of Fig. 2), which typically includes a User Interface (UI). Also important is support to load content in batch mode, to aid the sharing of metadata (e.g., through the OAI protocols), and to facilitate access to particular digital objects by way of their unique identifiers (e.g., their handles).

The main operations of a DL can be thought of as the business logic (see middle of Fig. 2), including key operations like searching (typically, with DSpace, using Lucene, see http://lucene.apache.org/) and browsing. For more flexibility, DLs can manage complex workflows. They can support authentication and authorization, protecting privacy, management of user groups, and broad suites of services to access and preserve content. Ultimately, the content of a DL must be supported through a storage subsystem (see bottom of Fig. 2), which typically makes use of database technology, as well as handling of multimedia files.

Key Applications

DLs generally are effective content management systems, offering a broad range of services such as archiving, digital preservation, browsing, searching, and presentation. Electronic libraries, virtual libraries, institutional repositories, digital repositories, courseware management systems, and personalized information systems are all considered to be different types of DLs. Any research in the area of DLs is thus inherently interdisciplinary in nature, encompassing especially computer science (CS) areas like database management (especially for the underlying storage layer), information retrieval, multimedia, hypertext, human-computer interaction, and library and information science (LIS).

DLs can be applied to a variety of needs. Many DLs of today are suitable for personal content management systems, institutional repositories, or distributed global systems. DLs can be used to preserve documents (e.g., electronic theses and dissertations, as coordinated by NDLTD, at http://www.ndltd.org), to manage multimedia content (e.g., Informedia [2], which focuses



Digital Libraries. Figure 2. DSpace institutional repository architecture (from: http://www.dspace.org).

on video), to support content-based retrieval of images (e.g., in connection with the needs of archaeologists in the ETANA DL, at http://www.etana.org), or to handle a combination of digital formats (e.g., ADEPT [6], which connects with GIS and mapping efforts).

One example of a highly visible and successful DL initiative is the Perseus project (http://www.perseus.tufts.edu/) which digitized ancient Greek literature and makes it available as an online repository, with many added services and additional information based on careful analysis and use of powerful tools. This project has completely changed the way classics are taught in universities across the world. Thus, DLs generally go well beyond archiving, providing value added services as well.

Another example is the Traditional Knowledge Digital Library (TKDL) [13], supported by the government of India, to digitize traditional Indian medical literature (relating to Ayurveda) in order to prevent bio-piracy and patents.

Future Directions

The DL community continues to spread and grow. Work on curricular resources [3,9,10,11] will help on the education side. Publishing at conferences and online magazines (e.g., http://www.dlib.org/) will help with dissemination of findings related to research, development, practice, and policy. Work on standards and open access (http://www.openarchives.org/) will facilitate interoperability. Improvement of systems like DSpace will help with more widespread utilization of effective software solutions, including support for preservation. Additional help with archiving and preservation, beyond just preserving the bits (see, for example, the LOCKSS effort, http://www.lockss.org/), is required (see for example, http://home.pacbell.net/hgladney/ddq.htm).

It is hoped that further work on foundations, including the 5S framework and the DELOS Reference Model, will lead to a firm theoretical and practical base for the field. For example, work to apply 5S to the growing need for personal DLs seems particularly promising, supporting Personal Information Management.

Greater efficiency and effectiveness of DL systems can help address problems associated with the "information glut," and work on DL quality metrics can help those who select or maintain DL systems and installations. More tools are needed for digital librarians, to assist them as they address fundamental questions like what to store, how to preserve, how to protect intellectual property, how to display, etc. These, and many similar questions, arising from technical, economical, and sociological perspectives, also will need to be addressed, as DLs are more widely employed. Then, efforts in the research, development, deployment, and operational sectors will better support the growing community of digital librarians, who aim to provide interested societies with cyberinfrastructure, which incorporates suitable organizational structures and appropriate services.

Experimental Results

See http://www.dli2.nsf.gov/

Data Sets

See http://www.dli2.nsf.gov/

Url to Code

See http://www.eprints.org/, http://www.dspace.org/, and http://www.fedora.info/

Cross-references

▶ Browsing
▶ Searching Digital Libraries

Recommended Reading

1. Candela L., Castelli D., Ioannidis Y., Koutrika G., Pagano P., Ross S., Schek H.-J., and Schuldt H. Setting the foundations of digital libraries: the DELOS manifesto. D-Lib Mag., 13(3/4), 2007.
2. Christel M., Wactlar H., and Stevens S. Informedia digital video library. In Proc. 2nd ACM Int. Conf. on Multimedia, 1994, pp. 480–481.
3. Digital Library Curriculum Development Project homepage (2007). http://curric.dlib.vt.edu/.
4. Gonçalves M.A. Streams, Structures, Spaces, Scenarios, and Societies (5S): A Formal Digital Library Framework and Its Applications. PhD Thesis, Virginia Tech, Department of Computer Science, 2004.
5. Gonçalves M., Fox E., Watson L., and Kipp N. Streams, structures, spaces, scenarios, societies (5S): a formal model for digital libraries. ACM Trans. Inf. Syst., 22:270–312, 2004.
6. Janée G. and Frew J. The ADEPT digital library architecture. In Proc. ACM/IEEE Joint Conf. on Digital Libraries, 2002.
7. Kochtanek T. and Hein K.K. Delphi study of digital libraries. Inf. Process. Manage., 35(3):245–254, 1999.
8. Licklider J.C.R. Libraries of the Future. The MIT Press, Cambridge, MA, 1965.
9. Pomerantz J., Oh S., Wildemuth B., Yang S., and Fox E.A. Digital library education in computer science programs. In Proc. ACM/IEEE Joint Conf. on Digital Libraries, 2007.
10. Pomerantz J., Oh S., Yang S., Fox E.A., and Wildemuth B. The Core: Digital Library Education in Library and Information Science Programs. D-Lib Magazine, 12(11), 2006.
11. Pomerantz J., Wildemuth B., Oh S., Fox E.A., and Yang S. Curriculum development for digital libraries. In Proc. ACM/IEEE Joint Conf. on Digital Libraries, 2006, pp. 175–184.
12. Reddy R. and Wladawsky-Berger I. Digital Libraries: Universal Access to Human Knowledge – A Report to the President. President's Information Technology Advisory Committee (PITAC), Panel on Digital Libraries 2001.
13. Sen N. TKDL - A safeguard for Indian traditional knowledge. Curr. Sci., 82(9):1070–71, 2002.
14. Smith T.R. and Frew J. Alexandria digital library. Commun. ACM, 38(4):61–62, 1995.

Digital Rights Management

Radu Sion
Stony Brook University, Stony Brook, NY, USA

Synonyms

DRM

Definition

Digital rights management (DRM) is a term that encompasses mechanisms and protocols deployed by content publishers and rights holders to enforce access licensing terms. This entry discusses mainly DRM for relational data, specifically such methods as database watermarking. General DRM techniques are discussed elsewhere [19].

Historical Background

Historically, DRM methods have found ample application in consumer entertainment and multimedia industries since the late 1980s. More recently, with the advent of massive relational data management and warehousing systems, increasingly, valuable data has been produced, packaged and delivered in relational form. In such frameworks, DRM assurances have become an essential requirement. As traditional multimedia DRM mechanisms are ill-suited for the new data domain, starting in 2001, researchers developed mechanisms for relational data rights protection [1,5,10,11,12,13,14,15] mainly centered around the concept of deploying steganography in hiding copyright "watermarks" in the underlying data. While initial efforts focused on basic numeric data [5,11,14,6], subsequent work handled categorical [9,10,14], and streaming [16,18] data types.

Foundations

Overview

As increasing amounts of data are produced, packaged and delivered in digital form, in a fast, networked environment, one of its main features threatens to become its worst enemy: zero-cost verbatim copies. The ability to produce duplicates of digital Works at almost no cost can now be misused for illicit profit. This mandates mechanisms for effective rights assessment and protection. Different avenues are available, each with its advantages and drawbacks. Enforcement by legal means is usually ineffective, unless augmented by a digital counterpart such as steganography (information hiding). *Digital Watermarking* as a method of rights assessment deploys information hiding to conceal an indelible "rights witness" ("rights signature", watermark) within the digital Work to be protected – thus enabling ulterior court-time proofs associating particular works with their respective rights holders. The soundness of such a method relies on the assumption that altering the Work in the process of hiding the mark does not destroy the value of the Work, while it is difficult for a malicious adversary ("Mallory") to remove or alter the mark beyond detection without doing so. The ability to resist attacks from such an adversary, mostly aimed at removing the watermark, is one of the major challenges.

Watermarking for Rights Protection

But how does the ability to prove rights in court relates to the final desiderata, namely to *protect* those rights? The ability to prove/assess rights convincingly in court constitutes a deterrent to malicious Mallory. It thus becomes a tool for rights protection if counter-incentives and legal consequences are set high enough. Such a method only works however if the rightful rights-holder (Alice) actually knows about Mallory's misbehavior *and* is able to prove to the court that: (i) Mallory possesses a certain Work X and (ii) X contains a "convincing" (e.g., very rare with respect to the space of all considered similar Works) and "relevant" watermark (e.g., the string "(c) by Alice"). This illustrates the game theoretic nature at the heart of the watermarking proposition and of information security in general. Watermarking is a game with two adversaries, Mallory and Alice. At stake lies the value inherent in a certain Work X, over which Alice owns certain rights. When Alice releases X, to the public or to a licensed but potentially untrusted party, she deploys watermarking for the purpose of ensuring that one of the following holds: (i) she can always prove rights in court over any copy or valuable derivative of X (e.g., segment thereof), (ii) any existing deviate Y of X, for which she cannot prove rights, does not preserve any signiicant value (derived from the value in X), (iii) the cost to produce such an un-watermarked derived Y of X that is still valuable (with respect to X) is higher than its value.

Once outsourced, i.e., out of the control of the watermarker, data might be subjected to a set of attacks or transformations; these may be malicious – e.g., with the explicit intent of removing the watermark – or simply the result of normal use of the data. An effective watermarking technique must be able to survive such use. In a relational data framework some of the important attacks and transformations are:

1. *Sampling.* The attacker (Mallory) can randomly select and use a subset of the watermarked data set that might still provide value for its intended purpose ("subset selection"). More specifically, here the concern is with both (1a) horizontal and (1b) vertical data partitioning – in which a valuable subset of the *attributes* are selected by Mallory.
2. *Data addition.* Mallory adds a set of tuples to the watermarked set. This addition is not to significantly alter the useful properties of interest to Mallory.
3. *Alteration.* Altering a subset of the items in the watermarked data set such that there is still value associated with the result. In the case of numeric data types, a special case needs to be outlined here, namely (3a) a linear transformation performed uniformly to all of the items. This is of particular interest as it can preserve significant valuable data-mining related properties of the data.
4. *Ulterior claims of rights.* Mallory encodes an additional watermark in the already watermarked data set and claims rights based upon this second watermark.

5. *Invertibility attack*. Mallory attempts to establish a plausible (watermark,key) pair that matches the data set and then claims rights based on this found watermark [2,3].

Consumer Driven Watermarking

An important point about watermarking should be noted. By its very nature, a watermark modifies the item being watermarked: it inserts an indelible mark in the work such that (i) the insertion of the mark does not destroy the value of the work, i.e., it is still useful for the *intended purpose*; and (ii) it is difficult for an adversary to remove or alter the mark beyond detection without destroying this value. If the work to be watermarked cannot be modified without losing its value then a watermark cannot be inserted. The critical issue is not to avoid alterations, but to limit them to acceptable levels with respect to the intended use of the work. Naturally, one can always identify some use that is affected by even a minor change to any portion of the data. It is therefore important that (i) the main intended purpose and semantics that should be preserved be identified during watermarking and that (ii) *the watermarking process not interfere with the final data consumer requirements*. This paradigm is called *consumer-driven watermarking*. In consumer-driven watermarking the rights holder and Mallory play against each other within subtle trade-off rules aimed at keeping the quality of the result within acceptable bounds. The data itself (its quality requirements) acts as an impartial referee moderating each and every "move".

In [4] Gross-Amblard introduce interesting theoretical results investigating alterations to relational data (or associated XML) in a consumer-driven framework in which a set of parametric queries are to be preserved up to an acceptable level of distortion. The author first shows that the main difficulty preserving such queries "is linked to the informational complexity of sets defined by queries, rather than their computational complexity" [4]. Roughly speaking, if the family of sets defined by the queries is not *learnable* [20], no query-preserving data alteration scheme can be designed. In a second result, the author shows that under certain assumptions (i.e., query sets defined by first-order logic and monadic second order logic on restricted classes of structures – with a bounded degree for the Gaifman graph or the tree-width of the structure) a query-preserving data alteration scheme exists.

Numerical Data Types

This section explores some of the watermarking solutions in the context of relational data in which one or more of the attributes are of a numeric type. Among existing solutions one distinguishes between *single-bit* (the watermark is composed of a single bit) and *multi-bit* (the watermark is a string of bits) types. Orthogonally, the encoding methods can be categorized into two: *direct-domain* and *distribution* encodings. In a direct-domain encoding, each individual bit alteration in the process of watermarking is directly correlated to (a part of) the encoded watermark. In distribution encodings, the encoding channel lies often in higher order moments of the data (e.g., running means, hierarchy of value averages). Each individual bit alteration impacts these moments for the purpose of watermark encoding, but in itself is not directly correlated to any one portion of the encoded watermark.

Single-bit encodings. In [1,5] Kiernan, Agrawal et al. propose a direct domain encoding of a single bit watermark in a numeric relational database. Its main algorithm proceeds as follows. A subset of tuples are selected based on a secret criteria; for each tuple, a secret attribute and corresponding least significant (ξ) bit position are chosen. This bit position is then altered according to yet another secret criteria that is directly correlated to the watermark bit to be encoded. The main assumption is, that changes can be made to any attribute in a tuple at any least significant ξ bit positions. At watermark detection time, the process will re-discover the watermarked tuples and, for each detected accurate encoding, become more "confident" of a true-positive detection.

The authors discuss additional extensions and properties of the solution including incremental updatability, blind properties, optimization of parameters, as well as handling relations without primary keys. To handle the lack of primary keys, the authors propose to designate another attribute, or a number of most significant bit-portions of the currently considered one, as a primary key. This however presents a significant vulnerability due to the very likely existence of duplicates in these values. Mallory could mount a statistical attack by correlating marked bit values among tuples with the same most significant bits. This issue has been also considered in [7] where a similar solution has been adopted.

Multi-bit encodings. While there likely exist applications whose requirements are satisfied by single-bit watermarks, often it is desirable to provide for "relevance", i.e., linking the encoding to the rights holder identity. This is especially important if the watermark aims to defeat against invertibility attacks (5). In a single-bit encoding this can not be easily achieved. Additionally, while the main proposition of watermarking is not covert communication but rather rights assessment, there could be scenarios where the actual message payload is of importance. One apparent direct extension from single-bit to multi-bit watermarks would be to simply deploy a different encoding, with a separate watermark key, for each bit of the watermark to be embedded. This however, might not be possible, as it will raise significant issues of inter-encoding interference: the encoding of later bits will likely distort previous ones. This will also make it harder to handle ulterior claim of rights attacks (4).

In [6] Li et al. extend the work by Kiernan, Agrawal et al. [1,5] to provide for multi-bit watermarks in a direct domain encoding. The scheme functions as follows. The database is parsed and, at each bit-encoding step, one of the watermark bits is randomly chosen for embedding; the solution in [1,5] is then deployed to encode the selected bit in the data at the "current" point. The "strength of the robustness" of the scheme is claimed to be increased with respect to [1,5] due to the fact that the watermark now possesses an additional dimension, namely length. This should guarantee a better upper bound for the probability that a valid watermark is detected from unmarked data, as well as for the probability that a fictitious secret key is discovered from pirated data (i.e., invertibility attacks (5)). This upper bound is said to be independent of the size of database relations thus yielding robustness against attacks that change the size of database relations. In [8] the same authors propose to use the multi-bit watermarking method [6] for "fingerprinting" relational data in order to track copyright violators.

Multi-bit distribution encoding. Encoding watermarking information in resilient numeric distribution properties of data presents a set of advantages over direct domain encoding, the most important one being its increased resilience to various types of numeric attacks. In [10,11,12,13,14,15], Sion et al. introduce a multi-bit distribution encoding watermarking scheme for numeric types. The scheme was designed with both an adversary and a data consumer in mind. More specifically the main desiderata were: (i) watermarking should be consumer driven – i.e., desired semantic constraints on the data should be preserved – this is enforced by a feedback-driven rollback mechanism, and (ii) the encoding should survive important numeric attacks, such as linear transformation of the data (3.a), sampling (1) and random alterations (3).

The solution starts by receiving as user input a reference to the relational data to be protected, a watermark to be encoded as a copyright proof, a secret key used to protect the encoding and a set of data quality constraints to be preserved in the result. It then proceeds to watermark the data while continuously assessing data quality, potentially backtracking and rolling back undesirable alterations that do not preserve data quality.

Watermark *encoding* is composed of two main parts: in the first stage, the input data set is securely partitioned into (secret) subsets of items; the second stage then encodes one bit of the watermark into each subset. If more subsets (than watermark bits) are available, error correction is deployed to result in an increasingly resilient encoding. Each single bit is encoded/represented by introducing a slight skew bias in the tails of the numeric distribution of the corresponding subset. The encoding is proved to be resilient to important classes of attacks, including subset selection, linear data changes and random item(s) alterations.

In [10,14,15] the authors discuss a proof of concept implementation. It is worth mentioning here due to its consumer-driven design. In addition to a watermark to be embedded, a secret key to be used for embedding, and a set of relations/attributes to watermark, the software receives as input also a set of external *usability plugin modules*. The role of these plugins is to allow user defined query metrics to be deployed and queried at run-time without re-compilation and/or software restart. The software uses those metrics to re-evaluate data usability after each atomic watermarking step.

To validate this consumer driven design the authors perform a set of experiments showing how, for example, watermarking with classification preservation can be enforced through the usability metric plugin mechanisms. Moreover, the solution is proved experimentally on real data to be extremely resilient to random alterations and uninformed alteration attacks. This is due to its distribution-based encoding which can naturally survive such alterations. For example, altering the *entire* watermarked data set within 1% of its original values only yields a distortion of less than 5% in the detected watermark.

Categorical Data Types

Categorical data is data drawn from a discrete distribution, often with a finite domain. By definition, it is either non-ordered (nominal) such as gender or city, or ordered (ordinal) such as high, medium, or low temperatures. There are a multitude of applications that would benefit from a method of rights protection for such data.

Additional challenges in this domain derive from the fact that one cannot rely on arbitrary small (e.g., numeric) alterations to the data in the embedding process. Any alteration has the potential to be significant, e.g., changing DEPARTURE_CITY from "Chicago" to "Bucharest" is likely to affect the data quality of the result more than a simple change in a numeric domain. There are no "epsilon" changes in this domain. This completely discrete characteristic of the data requires discovery of fundamentally new bandwidth channels and associated encoding algorithms. Moreover, the ability of the adversary to simply re-map attributes values to a new domain needs to be considered.

In [9,17] Sion et al. introduce a method of watermarking relational data with categorical types, based on a set of new encoding channels and algorithms. More specifically, two domain-specific watermark embedding channels are used, namely (i) *inter-attribute associations* and (ii) *value occurrence frequency-transforms* of values. The mechanism starts with an initial user-level assessment step in which a set of attributes to be watermarked are selected. In its basic version, watermark encoding in the *inter-attribute association* channel is deployed for each attribute pair (K,A) in the considered attribute set. A subset of "fit" tuples is selected, as determined by the association between A and K. These tuples are then considered for mark encoding. Mark encoding alters the tuple's value according to secret criteria that induces a statistical bias in the distribution for that tuple's altered value. The detection process then relies on discovering this induced statistical bias. The authors validate the solution both theoretically and experimentally on real data (Wal–Mart sales). They demonstrate resilience to both alteration and data loss attacks, for example being able to recover over 75% of the watermark from under 20% of the data.

The authors further discuss additional extensions and properties of the solution, including a consumer-driven design, incremental updatability, its blind nature, optimizations for minimizing alteration distances, as well as the ability to survive extreme vertical partitioning, handle multiple data sources as well as attribute re-mapping.

Key Applications

Rights protection for relational data is important in scenarios where it is sensitive, valuable and about to be outsourced. A good example is a data mining application, where data is sold in pieces to parties specialized in mining it, e.g., sales patterns database, oil drilling data, financial data. Other scenarios involve for example online B2B interactions, e.g., airline reservation and scheduling portals, in which data is made available for direct, interactive use.

Watermarking in relational frameworks is a relatively young technology that has begun its maturity cycle towards full deployment in industry-level applications. Many of the solutions discussed above have been prototyped and validated on real data. Patents have been filed for several of them, including Agrawal et al. [1,5] and Sion et al. [9,10,11,12,13,14,15,17]. In the next few years one expects these solutions to become available commercially, tightly integrated within existing DBMS or as stand-alone packages that can be deployed simultaneously on top of multiple data types and sources. Ultimately, the process of resilient information hiding will become available as a secure mechanism for not only rights protection but also data tracing and authentication in a multitude of discrete data frameworks.

Future Directions

A multitude of associated future research avenues present themselves in a relational framework, including: the design of alternative primary or pseudo-primary key independent encoding methods, a deeper theoretical understanding of limits of watermarking for a broader class of algorithms, the ability to better defeat additive watermark attacks, an exploration of zero-knowledge watermarking, etc.

Moreover, while the concept of on-the-fly quality assessment for a consumer-driven design has the potential to function well, another interesting avenue for further research would be to augment the encoding method with direct awareness of semantic consistency (e.g., classification and association rules). This would likely result in an increase in available encoding bandwidth, thus in a higher encoding resilience. One idea would be to define a generic language (possibly subset of SQL) able to naturally express such constraints and their propagation at embedding time.

Additionally, of particular interest for future research exploration are cross-domain applications of information hiding in distributed environments such as sensor networks, with applications ranging from resilient content annotation to runtime authentication and data integrity proofs.

Cross-references

► Steganography

Recommended Reading

1. Agrawal R., Haas P.J., and Kiernan J. Watermarking relational data: framework, algorithms and analysis. VLDB J., 12 (2):157–169, 2003.
2. Craver S., Memon N., Yeo B.-L., and Yeung M.M. Resolving rightful ownerships with invisible watermarking techniques: Limitations, attacks, and implications. IEEE J. Select. Areas Commun., 16(4):573–586, 1998.
3. Cox I., Bloom J., and Miller M. Digital watermarking. In Digital Watermarking. Morgan Kaufmann, 2001.
4. Gross-Amblard D. Query-preserving watermarking of relational databases and xml documents. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems. 2003, pp. 191–201.
5. Kiernan J. and Agrawal R. Watermarking relational databases. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
6. Li Y., Swarup V., and Jajodia S. A robust watermarking scheme for relational data. In Proc. Workshop on Information Technology and Systems, 2003, pp. 195–200.
7. Li Y., Swarup V., and Jajodia S. Constructing a virtual primary key for fingerprinting relational data. In Proc. 2003 ACM Workshop on Digital Rights Management, 2003, pp. 133–141.
8. Li Y., Swarup V., and Jajodia S. Fingerprinting relational databases: Schemes and specialties. IEEE Transactions on Dependable and Secure Computing, 2(1):34–45, 2005.
9. Sion R. Proving ownership over categorical data. In Proc. 20th Int. Conf. on Data Engineering, 2004.
10. Sion R. wmdb.*: A suite for database watermarking (demo). In Proc. 20th Int. Conf. on Data Engineering, 2004.
11. Sion R., Atallah M., and Prabhakar S. On watermarking numeric sets CERIAS Tech. Rep. 2001-60, Purdue University, 2001.
12. Sion R., Atallah M., and Prabhakar S. On watermarking numeric sets. In Proc. Int. Workshop on Digital Watermarking, 2002.
13. Sion R., Atallah M., and Prabhakar S. Watermarking Databases CERIAS TR 2002-28, Purdue University, 2002.
14. Sion R., Atallah M., and Prabhakar S. Rights protection for relational data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
15. Sion R., Atallah M., and Prabhakar S. Relational data rights protection through watermarking. IEEE Trans. Knowledge and Data Eng., 16(6), 2004.
16. Sion R., Atallah M., and Prabhakar S. Resilient rights protection for sensor streams. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
17. Sion R., Atallah M., and Prabhakar S. Rights Protection for categorical data. IEEE Trans. Knowledge and Data Eng., 17 (7):912–926, 2005.
18. Sion R., Atallah M., and Prabhakar S. Rights protection for discrete numeric streams. IEEE Trans. Knowledge and Data Eng., 18(5), 2006.
19. Wikipedia. Digital Rights Management. http://en.wikipedia.org/wiki/Digital_rights_management.
20. Valiant L.G. A theory of the learnable. In Proc. Symp. on the Theory of Computing. 1984, pp. 436–445.

Digital Signatures

Barbara Carminati
University of Insubria, Varese, Italy

Synonyms

Signatures

Definition

Informally, given a message M, the digital signature of M generated by a signer S is a bit string univocally bound to M and some secret key known only by S. More precisely, since digital signature schemes are based on asymmetric cryptography, it is possible to define the digital signature of M generated by S as a bit string dependent on M and the private key of S. Digital signature schemes have the property that signatures generated with a private key can be validated only by the corresponding public key. This ensures the authenticity of the message. Moreover, any modification on the signed message will invalidate the signature itself. This means that if the signature is validated it provides an evidence that the message has not been altered after the digital signature has been applied on it. This ensures the integrity of the message.

Historical Background

The notion of digital signature appeared in 1976 in a paper by Diffie and Hellman [1]. In this paper, authors introduced for the first time the concept of asymmetric cryptography and discussed how it could be combined with one-way functions to ensure message authentication. However, the first digital signature scheme with a practical implementation appeared only in 1978, proposed by Rivest et al. [7], and still represents one of the most exploited signature technique. Later on, several other schemes have been proposed with an improved efficiency and/or offering further functionalities (see [4] for a survey). An example of these schemes are *one-time signatures*, that is, digital signature schemes where keys can be used to sign, at most, one message. These schemes have the benefit that are very efficient, making them particularly useful in applications requiring a low computational complexity. Other examples are, for instance, *arbitrated digital signatures*, where signatures are generated and verified with the cooperation of a trusted third party, or *blind signature* schemes devised in such a way that a signer is not able to observe the message it signs. This latter scheme are useful in digital cash protocols or electronic voting systems.

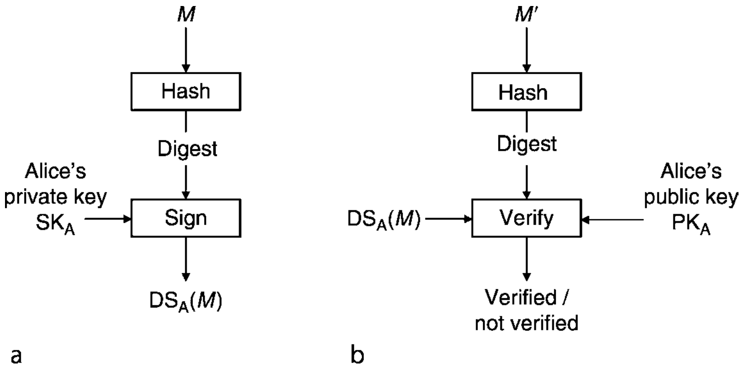
Foundations

Digital signatures have been introduced to obtain, in a digital form, the benefits of handwritten signatures. In general, by appending a handwritten signature to a document, the signer provides evidence that he or she has validated the document. This could mean that the document has been generated, modified, or at least simply read for approval by the person who claims to have done it. In a similar way, digital signatures aim to provide evidence that the signer has really elaborated the message. To achieve this result, digital signatures exploit asymmetric cryptography and collision-resistant hash functions. This implies that a user A willing to digitally sign a message has to be provided with a key pair consisting of a *private key*, SK_A, and a *public key*, PK_A. The security of the digital signature relies on the assumption that the private key is kept secret by the owner itself, whereas the public key is available to all users. Note that various digital signature schemes have been proposed, with differen signature and verification algorithms, which will be discussed later on. However, independently from the adopted algorithms, the overall process to generate an validate a digital signature is always the same. As an example, assume that Alice wishes to send Bob a messag $\mathcal{M}$ completed with its digital signature, denoted a $DS_A(\mathcal{M})$. The procedure to sign and verify the signature is described in what follows.

Digital signature generation. To sign a message $\mathcal{M}$, Alice first generates a condensed version of $\mathcal{M}$, called *digest* of the message, which is obtained by a collision resistant hash function $h()$. Then, the digest is signed using Alice's private key (cf. Fig. 1a). Notice that "signing" a digest implies to apply on it a set of transformations according to the digital signature scheme adopted.

Digital signature verification. Assuming that Bob receives the message $\mathcal{M}'$ from Alice. To validate its digital signature, i.e., to verify whether $\mathcal{M}'$ is equal to $\mathcal{M}$, he first computes the digest of $\mathcal{M}'$ by using the same hash function used for signature generation. Then, using Alice's public key and the new digest value he can verify whether the digital signature is valid or not (cf. Fig. 1b).

Digital signatures provide several benefits. The first is related to the property of asymmetric cryptography



Digital Signatures. Figure 1. (a) Digital signature generation. (b) Digital signature verification.

ensuring that it is computationally infeasible to validate a signature $DS_A(\mathcal{M})$ with a public key PK_x different from the one corresponding to the private key used for signature creation. Thus, if the signature is verified, it guarantees that the message has been really generated by the owner of the private key, that is, it ensures the *authenticity* of the message. This property entails as further benefits, the *non repudiation* of the message. Indeed, in case of a lying signer A claiming that the message was not generated by him or her, the digital signature $DS_A(\mathcal{M})$ acts like an evidence of the opposite, making thus A not able to repudiate the message. A further benefit is given by properties of hash functions, which ensures that it is computationally infeasible to find two different messages $\mathcal{M}$ and $\mathcal{M}'$such that $h(\mathcal{M}) = h(\mathcal{M})$. Thus, if the signature is validated, it means that the digest computed on the received message matches the digest extracted during the verification process. A validated signature provides, therefore, evidence that the message has not been altered after the digital signature has been applied on it. This ensures the *integrity* of the message.

In the following, two of the most used digital signature schemes, that is, the DSA and RSA algorithms, are illustrated (see [4] for a detailed description of them). However, for a more comprehensive overview of digital signatures, besides cryptographic details it is interesting to have also an example of how they can be used in real world scenarios. Given its relevance, it is considered the Web as reference scenario. For this reason, in the second part of this entry it is presented the standard proposed by W3C for digital signature encoding, called XML Signature [8].

Digital Signature Schemes

In general, a digital signature scheme is defined according to three algorithms:

1. *Key generation algorithm* that randomly computes a pair of keys (SK_A, PK_A) for the signer A.
2. *Signature generation algorithm* $S()$, which receives as input the messag $\mathcal{M}$ to be digitally signed (Hereafter, the discussion refers to a generic message M. However, it is important to recall that the signature process usually requires to digitally sign the digest of the original message.), the private key SK_A of the signer and generates the digital signature $\mathrm{DS_A}(\mathcal{M})$ of the message $\mathcal{M}$.
3. *Signature verification algorithm* $V()$, that takes as input the digital signature $\mathrm{DS_A}(\mathcal{M})$ of the message $\mathcal{M}$, the public key PK_A of the signer and, optionally, the message $\mathcal{M}$, and returns whether the verification succeeds or not.

It is possible to organize the many schemes proposed in the literature into two main classes, that is, *digital signature schemes with appendix* and *digital signature schemes with message recovery*, which are described in what follows.

Digital Signature Schemes with Appendix

The main characteristic that distinguishes these schemes with regard to those with message recovery is that they require the original message during the verification process. Thus, together with the signer's public key and the digital signature, the verification algorithm takes as input also the original message. This implies that the signer A has to send to the intended verifier B the original message complemented with its digital signature as "appendix."

In the literature, there exist several digital signature schemes with appendix. However, the most relevant and recognized is the one proposed in 1991 by the U.S. National Institute of Standards and Technology (NIST). The scheme, called Digital Signature Algorithm (DSA) [3], became an U.S. Federal Information Processing Standard (FIPS 186) in 1993, with the aim to be the first digital signature scheme recognized by any government (i.e., a Digital Signature Standard – DSS) (In 2000, NIST extended the standard to three FIPS-approved algorithms: Digital Signature Algorithm (DSA), RSA (as specified in ANSI X9.31), and Elliptic Curve DSA (ECDSA; as specified in ANSI X9.62).).

A summary of DSA algorithms is given in the following (the interested reader can refer to [6] for more details).

DSA – Key generation algorithm. Let A be the entity for which public and private keys are generated. PK_A and SK_A are calculated as follows:

1. Select a prime number p, where $2^{L-1} < p < 2^L$, L is multiple of 64, and $512 \leq L \leq 1024$;
2. Select q, a prime divisor of $p-1$, where $2^{159} < q < 2^{160}$;
3. Select g, a number whose multiplicative order modulo p is q. (This can be calculated by g =

$h^{(p-1)/q}$ mod p, where h is any integer with $1 < h < p - 1$ such that $h^{(p-1)/q}$ mod $p > 1$);

4. Generate a random or pseudorandom integer x, where $0 < x < q$;
5. Calculate $y = g^x$ mod p;
6. Set $PK_A = (p, q, g, y)$ and $SK_A = (x)$;

DSA – Signature algorithm. Let M be the message to be signed with PK_A, and $h()$ be an hash function (FIPS 186–2 uses SHA-1 as hash function [5]. The forthcoming FIPS 186–3 uses SHA-224/256/384/512.), the digital signature $DS_A(\mathcal{M})$ is generated as follows:

1. Generate a random or pseudorandom integer k, where $0 < k < q$;
2. Calculate $r = g^k$ mod p;
3. Calculate $s = (k^{-1}(h(m) + xr))$ mod q, where k^{-1} is the multiplicative inverse of k;
4. Set $DS_A(M) = (r,s)$;

DSA-Signature verification algorithm. Let M', r', and s' be the received versions of M, r, and s. To validate the digital signature, an entity B should do the following:

1. Obtain the public key of A, $PK_A = (p, q, g, y)$;
2. Verify that $0 < r' < q$ and $0 < s' < q$, if not the signature shall be rejected;
3. Compute $w = (s')^{-1}$ mod q;
4. Compute $u_1 = (h(M')w)$ mod q and $u_2 = (r'w)$ mod q;
5. Compute $v = (((g)^{u_1}(y)^{u_2})$ mod p) mod q;
6. If $v = r'$, then the signature is verified.

Digital Signature Schemes with Message Recovery

Differently to the previous signature schemes, these schemes do not require the original message during the verification process. In contrast, by taking as input only the digital signature and the public key of the signer, the verification algorithm V() recovers the original message M directly from the digital signature. The main advantage of these schemes is that they minimize the length of the message to be transmitted, in that only the digital signature has to be sent. This makes the digital signatures with message recovery particularly tailored to applications where the bandwidth is one of the main concern.

However, it is important to notice that adopting these schemes in the digital signature process depicted in Fig. 1 requires to send the original message even if $V()$ is able to recover the message. Indeed, if the message is hashed before signing it, these schemes are not able to recover the original message, rather they recover only its digest. Having the digest without the original message makes the receiver not able to verify the message integrity, in that it is not possible to match the two hash values. For this reason, when these schemes are adopted in the standard digital signature process (see Fig. 1) they are used like schemes with appendix, i.e., the digital signature is appended to the corresponding message.

One of the most known digital signature scheme with message recovery is the RSA algorithm [7], which is the first one in public-key cryptography to be suitable for signature as well as encryption. RSA algorithms are briefly illustrated in what follows, by focusing only on signatures. An interested reader should refer to [4] for a deeper discussion on RSA public key cryptography, and to PKCS#1 standard [2] for details on RSA implementation.

RSA – Key generation algorithm. Let A be the entity for which public and private keys are generated. PK_A and SK_A are calculated as follows:

1. Generate two large random or pseudorandom prime numbers p and q;
2. Compute $n = pq$ and $\phi = (p - 1)(q - 1)$;
3. Select a random integer e, $1 < e < \phi$, such that gcd $(e; \phi) = 1$;
4. Compute the unique integer d, $1 < d < \phi$, such that $ed \equiv 1($ mod $\phi)$;
5. Set $PK_A = (n, e)$ and $SK_A = (n, d)$;

RSA - Signature algorithm. Let M be the message to be signed with SK_A. The digital signature $DS_A(M)$ is generated as follows:

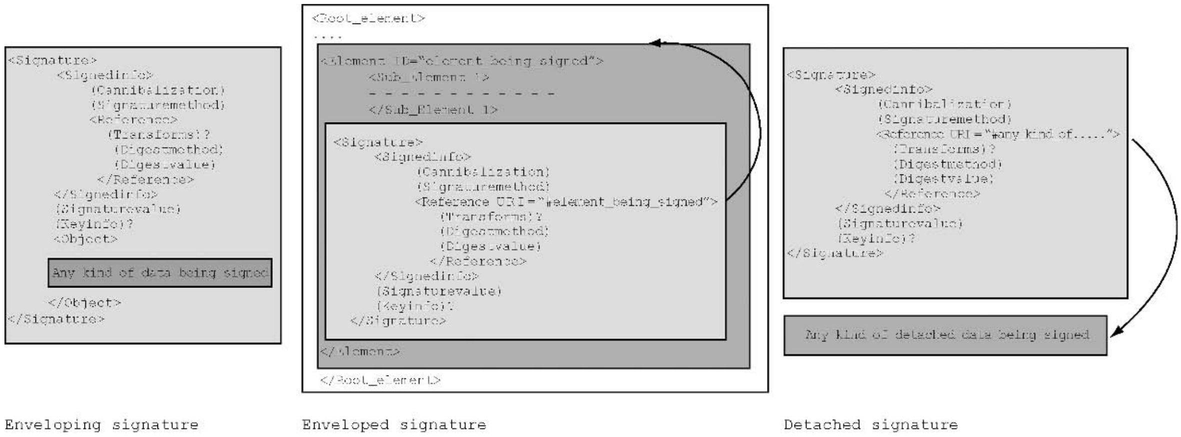
1. Compute $s = M^d$ mod n;
2. Set $DS_A(M) = (s)$.

RSA – Signature verification algorithm. Let s' be the received versions of s. To validate the digital signature an entity B should do the following:

1. Compute $M' = s'^e$ mod n;
2. If $M = M'$, then the signature is verified.

XML Signature

In conjunction with IETF, the W3C XML Signature Working Group has proposed a recommendation, called XML Signature [8], with the twofold goal of defining an XML representation for digital signatures of arbitrary data contents, as well as a description of



Digital Signatures. Figure 2. Taxonomy of XML Signatures.

the operations to be performed as part of signature generation and validation. The proposed XML syntax has been designed to be very flexible and extensible, with the result that a single XML Signature can sign more that one type of digital content. For instance, a single XML Signature can be used to sign an HTML file and all the JPEG files containing images linked to the HTML page. The overall idea is that data to be signed are digested, each digest value is placed into a distinct XML element with other additional information needed for the validation. Then, all the resulting XML elements are inserted into a parent element, which is digested and digitally signed.

Additionally, the standard supports a variety of strategies to locate the data being signed. These data can be either external or local data, that is, a portion of the XML document containing the signature itself. In particular, the XML Signature recommendation supports three different kinds of signatures, which differ for the localization of the signed data with regard to the XML element encoding its signature (see Fig. 2): the *Enveloping Signature*, where the signed data is embedded into the XML Signature; the *Enveloped Signature*, in which the signed data embeds its signature; the *Detached Signature*, where the signed data is either an external data, or a local data included as a sibling element of its signature.

In what follows, a brief overview of the process needed for XML Signature generation is given. In describing these steps the basic structure of an XML Signature, reported in Fig. 3, is referred, where symbol "?" denotes zero or one occurrences; "+" denotes one or more occurrences; and "∗" denotes zero or more occurrences.

```
<Signature>
<SignedInfo>
(CanonicalizationMethod)

  (SignatureMethod)
   (<Reference  (URI=)? >
        (Transforms)?
        (DigestMethod)
        (DigestValue)
    </Reference>)+
 </SignedInfo>
  (SignatureValue)
  (KeyInfo)?

  (<Object>
         (SignatureProperties)?
         (Manifest)?
   </Object>)*
</Signature>
```

Digital Signatures. Figure 3. Basic structure of an XML Signature.

The first step in the generation of an XML signature requires to specify which are the data to be signed. To this purpose, an XML Signature contains a `Reference` element for each signed data, whose `URI` attribute stores the address of the signed data (In the case of enveloping signatures, URI attribute is omitted since the data is contained in the signature element itself, whereas for enveloped signature the URI attribute denotes the element being signed via a fragment identifier.). Then, the digest of the data is calculated and placed into the `DigestValue` supplement. Information on the algorithm used to generate the digest are stored into the DigestMethod element. The Reference element may contain an optional Transforms subelement specifying an ordered list of transformations (such as for instance canonicalization, compression, XSLT/XPath expressions) that have been applied to the data before it was digested. The next step is to collect all the Reference elements into a SignedInfo element, which contains the information that is actually signed. Before applying the digital signature, the SignedInfo element is transformed into a standard form, called canonical form. The aim of such transformation is that of eliminating from the element additional symbols eventually introduced during the processing (for instance, spaces introduced by an XML parser and so on), that may cause mistakes during the signature validation process. After the canonical form has been generated, the digest of the whole SignedInfo element is computed and signed. The resulting value is stored into the SignatureValue element, whereas information about the algorithm used for generating the digital signature is contained in the SignatureMethod element. The Signature element can also give the recipient additional information to obtain the keys to validate the signature. Such information is stored into the optional KeyInfo subelement. The last step is to wrap the SignedInfo, SignatureValue, and KeyInfo elements into a Signature element. In the case of enveloping signature the Signature element also contains the data being signed, wrapped into the Object subelement.

Key Applications

Digital signatures are widely adopted in scenarios where assurances of message authenticity and integrity are crucial. Examples of these scenarios are email applications. Due to the relevance of these applications, two different recommendation for applying digital signatures to email have been proposed (i.e., PGP and S/MIME). Other scenarios are those requiring the authenticity and integrity of messages exchanged during protocols execution, like, for instance, the SSL protocol which exploits digital signatures to create secure Web sessions. A further relevant scenario is given by Public Key Infrastructure (PKI). PKIs have been introduced to univocally bind public keys to the respective owners. PKI assumes the existence of one or more trusted Certificate Authorities (CAs) in charge of generating public key certificates containing information about the identity of the key owner. To provide evidence that a public key certificate has been generated by a CA, PKI requires that certificates are digitally signed by CA.

Cross-references

▶ Asymmetric Cryptography
▶ Blind Signatures
▶ Hash Functions
▶ XML

Recommended Reading

1. Diffie W. and Hellman M. New directions in cryptography. IEEE Trans. Inf. Theory, IT-22(6):644–654, 1976.
2. Jonsson J. and Kaliski B. Public-Key Cryptography Standards (PKCS) No. 1: RSA Cryptography. Request for Comments 3447, February 2003.
3. Kravitz D.W. (1993) Digital Signature Algorithm. U.S. Patent No. 5, 231, 668.
4. Menezes A.J., van Oorschot P.C., and Vanstone S.A. Handbook of Applied Cryptography. CRC, 1996.
5. National Institute of Standards and Technology. Secure Hash Standard. Federal Information Processing Standards Publication, FIPS 180–1, 1995.
6. National Institute of Standards and Technology. Digital Signature Standard (DSS). Federal Information Processing Standards Publication, FIPS 186–2, 2000.
7. Rivest R.L., Shamir A., and Adleman L.M. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21:120–126,1978.
8. World Wide Web Consortium. XML-Signature Syntax and Processing. W3C Recommendation, 2002.

Digital Surface Model

▶ Digital Elevation Models (DEMs)

Digital Terrain Model (DTM)

▶ Digital Elevation Models (DEMs)

Digital Video Retrieval

▶ Content-Based Video Retrieval

Digital Video Search

▶ Content-Based Video Retrieval

Dimension

Torben Bach Pedersen
Aalborg University, Aalborg, Denmark

Definition

A *dimension* is a hierarchically organized set of *dimension values*, providing categorical information for characterizing a particular aspect of the data stored in a multidimensional *cube*.

Key Points

As an example, a three-dimensional cube for capturing sales may have a Product dimension, a Time dimension, and a Store dimension. The Product dimension captures information about the product sold, such as textual description, color, weight, etc., as well as groupings of products (product groups, product families, etc.). The Time dimension captures information about the time of the sale, at the Date level or finer, as well as groupings of time such as Week, Month, Weekday, Quarter and Year. It may also contain application-specific time-related information, e.g., what the temperature was on the particular day (interesting for ice cream sellers) or whether there was a special event in town on that day, e.g., a big sports event. The Store dimension captures information about stores (Name, Size, Layout), as well as various groupings of Stores (City, State, Sales District).

The notion of a dimension is an essential and distinguishing concept for multidimensional cubes, where dimensions are a first-class object. Dimensions are used for two purposes: the *selection* of data and the *grouping* of data at a desired level of detail. A dimension is organized into a containment-like *hierarchy* composed of a number of *levels*, each of which represents a level of detail that is of interest to the analyses to be performed. The instances of the dimension are typically called *dimension values*. Each such value belongs to a particular level.

In some multidimensional models, a dimension level may have associated with it a number of *level properties* that are used to hold simple, non-hierarchical information. For example, the Weekday of a particular date can be a level property in the Date level of the Time dimension. This information could also be captured using an extra Weekday dimension. Using the level property has the effect of not increasing the dimensionality of the cube.

Unlike the linear spaces used in matrix algebra, there is typically no ordering and/or distance metric on the dimension values in multidimensional models. Rather, the only ordering is the containment of lower-level values in higher-level values. However, for some dimensions, e.g., the Time dimension, an ordering of the dimension values is available and is used for calculating cumulative information such as "total sales in year to date."

When implemented in a relational database, a dimension is stored in one or more *dimension tables* using either a so-called *star schema* (one table per dimension, with a surrogate key and one column per dimension level or level property) or a so-called *snowflake schema* (one table per dimension level, each with a surrogate key and an attribute for the textual name of the dimension value, as well as one attribute per level property).

Cross-references

▶ Cube
▶ Hierarchy
▶ Multidimensional Modeling
▶ Snowflake Schema
▶ Star Schema

Recommended Reading

1. Kimball R., Reeves L., Ross M., and Thornthwaite W. The Data Warehouse Lifecycle Toolkit. Wiley Computer, 1998.
2. Pedersen T.B., Jensen C.S., and Dyreson C.E. A foundation for capturing and querying complex multidimensional data. Inf. Syst., 26(5):383–423, 2001.
3. Thomsen E. OLAP Solutions: Building Multidimensional Information Systems. Wiley, New York, 1997.

Dimensional Modeling

▶ Multidimensional Modeling

Dimensionality Curse

▶ Curse of Dimensionality

Dimensionality Reduction

Heng Tao Shen
The University of Queensland, Brisbane, QLD,
Australia

Definition

From database perspective, dimensionality reduction (DR) is to map the original high-dimensional data into a lower dimensional representation that captures the content in the original data, according to some criterion. Formally, given a data point P = {p_1, p_2,...,p_D} in *D*-dimensional space, DR is to find a *d*-dimensional subspace, where $d < D$, such that P is represented by a *d*-dimensional point by projecting P into the *d*-dimensional subspace.

Key Points

Advances in data collection and storage capabilities have led to an information overload in most sciences. Many new and emerging data types, such as multimedia, time series, biological sequence, have been studied extensively in the past and present new challenges in data analysis and management due to their high dimensionality of data space. One known phenomenon of "dimensionality curse" leads traditional data access methods to fail [3]. High-dimensional datasets present many mathematical challenges as well as some opportunities. In many cases, not all dimensions are equally "important" for understanding the underlying data. It is of interest in many applications to reduce the dimension of the original data prior to any modelling and indexing of the data, due to efficiency and effectiveness concerns. Generally, dimensionality reduction can be used for the following purposes:

Simplifying complex data: for many applications, particularly in database and information retrieval, high dimensionality of feature space leads to high complexity of data representation. The dimensionality has to be reduced to achieve satisfactory performance for an indexing structure. Typical methods include traditional Discrete Fourier transform (DFT) and Discrete Wavelet Transform (DWT), Adaptive Piecewise Constant Approximation (APCA), Principle Component Analysis (PCA) and its various improvements, Latent Semantic Indexing and its variants, and Locality Preservation Projection (LPP) etc. For these types of applications, the dimensionality reduction method must have an explicit mapping function to map the query points into the low-dimensional subspace for similarity search [2].

Modeling and analyzing data: for many applications, particularly in classification and pattern recognition, the underlying data structure is often embedded in a much lower-dimensional subspace. The task of recovering the meaningful low-dimensional structures hidden in high-dimensional data is also known as "manifold learning". Typical methods include Independent Component Analysis (ICA), Multidimensional Scaling (MDS), Isometric feature Mapping (Isomap) and its improvements, Locally Linear Embedding (LLE), Laplacian Eigenmaps, Semantic Subspace Projection (SSP), etc. For these types of applications, the dimensionality reduction is typically a very expensive process and performed on a small set of sample points for learning purpose. Since they are defined only on the sample/training data and have no explicit mapping function, they are not applicable to information retrieval and database applications [1].

Dimensionality reduction methods can be categorized to be linear or non-linear. Linear techniques are based on the linear combination of the original dimensions, while non-linear methods are mainly used to find an embedded non-linear manifold within the high dimensional space.

Cross-references

- ▶ Discrete Fourier Transform (Dft)
- ▶ Discrete Wavelet Transform and Wavelet Synopses
- ▶ Independent Component Analysis (Ica)
- ▶ Isometric Feature Mapping (Isomap)
- ▶ Latent Semantic Indexing
- ▶ Locality-Preserving Mapping
- ▶ Locally Linear Embedding (Lle) Laplacian Eigenmaps
- ▶ Multidimensional Scaling
- ▶ Principle Component Analysis
- ▶ Semantic Subspace Projection (Ssp)

Recommended Reading

1. Roweis S.T. and Saul L.K. Nonlinear dimensionality reduction by locally linear embedding. Science, 290(5500):2323–2326, 2000.
2. Shen H.T., Zhou X., and Zhou A. An adaptive and dynamic dimensionality reduction method for high-dimensional indexing. VLDB J, 16(2):219–234, 2007.

3. Weber R., Schek H.-J., Blott S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 194–205.

Dimension Reduction Techniques for Clustering

Chris Ding
University of Texas at Arlington, Arlington, TX, USA

Synonyms

Subspace selection; Graph embedding

Definition

High dimensional datasets is frequently encountered in data mining and statistical learning. Dimension reduction eliminates noisy data dimensions and thus and improves accuracy in classification and clustering, in addition to reduced computational cost. Here the focus is on unsupervised dimension reduction. The wide used technique is *principal component analysis* which is closely related to *K*-means cluster. Another popular method is *Laplacian embedding* which is closely related to spectral clustering.

Historical Background

Principal component analysis (PCA) was introduced by Pearson in 1901 and formalized in 1933 by Hotelling. PCA is the foundation for modern dimension reduction. A large number of linear dimension reduction techniques were developed during 1950–1970s.

Laplacian graph embedding (also called *quadratic placement*) is developed by Hall [8] in 1971. *Spectral graph partitioning* [6], is initially studied in 1970s; it is fully developed and becomes popular in 1990s for circuit layout in VLSI community (see a review [1]), graph partitioning [10] and data clustering [3,7,11]. It is now standard technique [2,9].

Foundations

High dimensional datasets is frequently encountered in applications, such as information retrieval, image processing, computational biology, global climate research, For example, in text processing, the dimension of a word vector is the size of the vocabulary of a document collection, which is typically tens of thousands. In molecular biology, human DNA gene expression profiles typically involve thousands of genes, which is the problem dimension. In image processing, a typical 2D image has $128^2 = 16,384$ pixels or dimensions.

Clustering data in such high dimension is a challenging problem. Popular clustering methods such as *K*-means and EM methods suffer from the well-known local minima problem: as iterations proceed, the system is often trapped in the local minima in the configuration space, due to the greedy nature of these algorithms. In high dimensions, the iso-surface (where the clustering cost function remains constant) is very rugged, the system almost always gets trapped somewhere close to the initial starting configuration. In other words, it is difficult to sample through a large configuration space. This is sometimes called curse of dimension. Dimension reduction is widely used to relieve the problem. In this direction, the principal component analysis (PCA) is the most widely adopted. PCA is an example of linear dimension reduction or mapping.

A related problem is graph clustering. Given a graph with n nodes (objects) and a square matrix of pairwise similarities as the edge weights, the task is to cluster nodes into disjoint clusters. The state-of-the-art algorithm is spectral clustering using graph Laplacian eigenvectors. An effective implementation is to embed the graph in metric space and use the *K*-means algorithm to cluster the data. Here the graph embedding is a key step, reducing a problem of $n(n-1)/2$ data items (pairwise similarities) into a problem of nK data items where K is the dimension of the metric space. In this direction, the Laplacian embedding is the most widely adopted. Laplacian embedding is an example of nonlinear dimension reduction or mapping.

Dimension Reduction Versus Feature Selection

Dimension reduction often finds combinations of many variables which satisfy certain global optimal conditions. Feature selection (also called variable selection) considers individual variables separately, or combinations of small number variables. Although they share similar goals for clustering and classification, their approaches differ greatly. Only dimension reduction will be discussed here.

PCA and Other Linear Dimension Reduction

Linear dimension reduction seek linear combinations of variables that optimizes certain criteria. PCA seek linear combinations that maximizes the variances.

Consider a set of input data vectors $X = (\mathbf{x}_1,\ldots,\mathbf{x}_n)$ where $\mathbf{x}_i$ is a p-dimensional vector. Let $\bar{\mathbf{x}} = \sum_{i=1}^n \mathbf{x}_i/n$ be the center of the data. The covariance matrix is $C_X = (1/n \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$. Let $U = (\mathbf{u}_1,\ldots,\mathbf{u}_k)$ be the eigenvectors of C_X associated with the k largest eigenvalues of C_X. The PCA is the linear transformation of the original p-dimensional data $(\mathbf{x}_1,\ldots,\mathbf{x}_n)$ into k-dimensional data $Y = (\mathbf{y}_1,\ldots,\mathbf{y}_n)$ with

$$\mathbf{y}_i = U^T \mathbf{x}_i. \tag{1}$$

The most important property of the transformed data Y is that they are uncorrelated:

$$C_Y = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{y}_i - \bar{\mathbf{y}})^T = U^T C_X U = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_k \end{pmatrix}. \tag{2}$$

Because of this, each dimension j of Y has a clean variance λ_j. The second important property of the transformed data is that dimensions of Y with small variances are ignored. Note that in the original data X, because different dimensions are correlated, there is no clear way to identify a dimension with small variance and eliminate it.

A third benefit is the relation to K-means clustering. The cluster centroids of the global solution of the K-means clustering form a subspace. This subspace is identical to the PCA subspace formed by transformed data $(\mathbf{u}_1,\ldots,\mathbf{u}_k)$ when k is the number of clusters, according to the theory of the equivalence [4] between PCA and K-means clustering. In other words, PCA automatically bring us to the narrow subspace containing the global solution of K-means clustering – a good starting place to find near global solutions. For this reason, PCA+K-means clustering is one of the most effective approach for clustering.

Linear Discriminant Analysis

PCA performs the unsupervised dimension reduction, without prior knowledge of class labels of each data object. If the class information is known, more appropriate dimension reduction can be devised. This is call linear discriminant analysis (LDA). In LDA, the optimal subspace $G = (\mathbf{g}_1,\ldots,\mathbf{g}_k)$ is obtained by optimizing

$$\max_U \operatorname{Tr} \frac{G^T S_b G}{G^T S_w G}, \tag{3}$$

where the between-class (S_b) and within-class (S_b) scatter matrices are defined as

$$S_b = \sum_k n_k (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T,$$
$$S_w = \sum_k \sum_{i \in C_k} (\mathbf{x}_i - \mathbf{m}_k)(\mathbf{x}_i - \mathbf{m}_k)^T, \; C_X = S_b + S_w, \tag{4}$$

where $\mathbf{m}_k$ is the mean of class C_k and $\mathbf{m}$ is the global total mean. The central idea is to separate different classes as much as possible (maximize the between-class scatter S_b) while condense each class as much as possible (minimize the within-class scatter S_w). The solution of G is given the k eigenvectors of $S_w{}^{-1}S_b$ associated with the largest eigenvalues. The dimension k of the subspace is set to $k = C - 1$ where C is the number of classes.

Once G is computed, the transformed data is $\mathbf{y}_i = G^T\mathbf{x}_i$ or $Y = G^T X$. An important property of Y is that components of Y are uncorrelated.

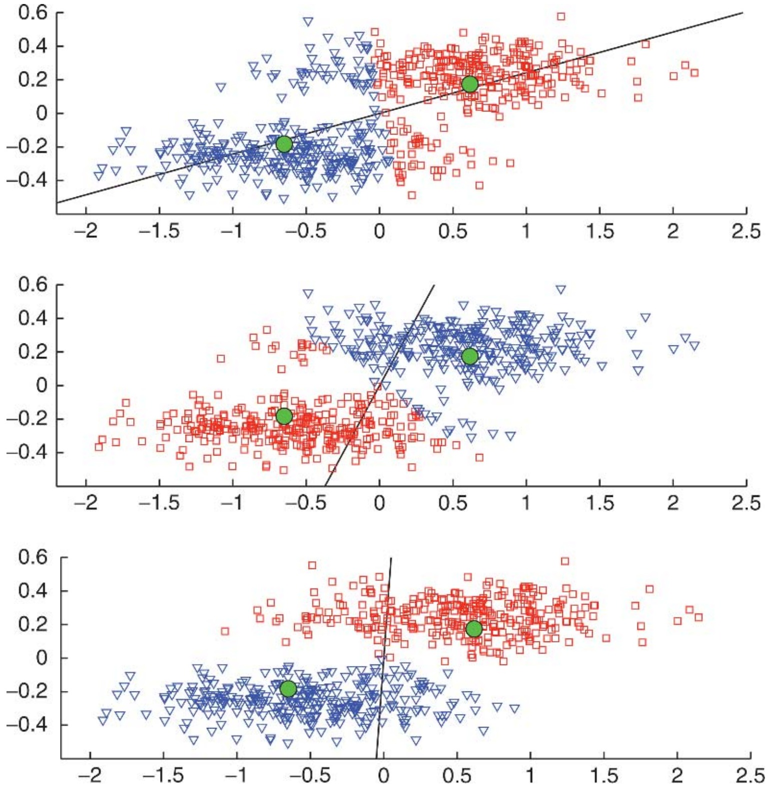
$$(C_Y)_{k\ell} = (G^T C_X G)_{k\ell} = \mathbf{g}_k^T C_X \mathbf{g}_\ell = \mathbf{g}_k^T S_w \mathbf{g}_\ell + \mathbf{g}_k^T S_b \mathbf{g}_\ell = 0, \quad k \neq \ell. \tag{5}$$

When the data dimension p is greater than n, the number of data points, S_w has zero eigenvalues and $S_w{}^{-1}$ does not exist. This problem can resolved by first project data into PCA subspace with a dimension less than $p - C$ and then perform LDA on the projected data. LDA is very popularly in image processing where the data dimension is very high.

Adaptive Dimension Reduction – Combining Dimension Reduction and Clustering

The PCA subspace is not always the best subspace for data clustering; Figure 1 shows an example. LDA subspace is more suitable for data clustering; but LDA requires the knowledge of class labels. This dilemma is resolved by the adaptive dimension reduction (ADR).

ADR start with the PCA subspace and then adaptively compute the subspace by (A) clustering to generate class labels and (B) doing LDA to obtain the most discriminant subspace. (A) and (B) are repeated until convergence. Figure 2 shows the process: the computed 1D subspace (the direction)



Dimension Reduction Techniques for Clustering. Figure 1. A 2D dataset with 600 data points. *Green dots* are the centroids of each cluster. The *line* indicates the subspace *U*. *Top*: *K*-means clustering results PCA subspace. *Middle*: After two ADR iterations. *Bottom*: ADR converges after two more iterations. The subspace is the most discriminant.

gradually moves towards the most discriminative direction. Theoretically, ADR optimizes the LDA objective function

$$\max_{G,H} \operatorname{Tr} \frac{G^T S_b G}{G^T S_w G} \tag{6}$$

to obtain simultaneously the subspace G and the clusters represented by the cluster membership indicator matrix H, where $H_{ik} = 1/|C_k|^{1/2}$ if data point $\mathbf{x}_i$ belongs to cluster C_k; $H_{ik} = 0$ otherwise. $|C_k|$ is the size of C_k.

Metric Scaling

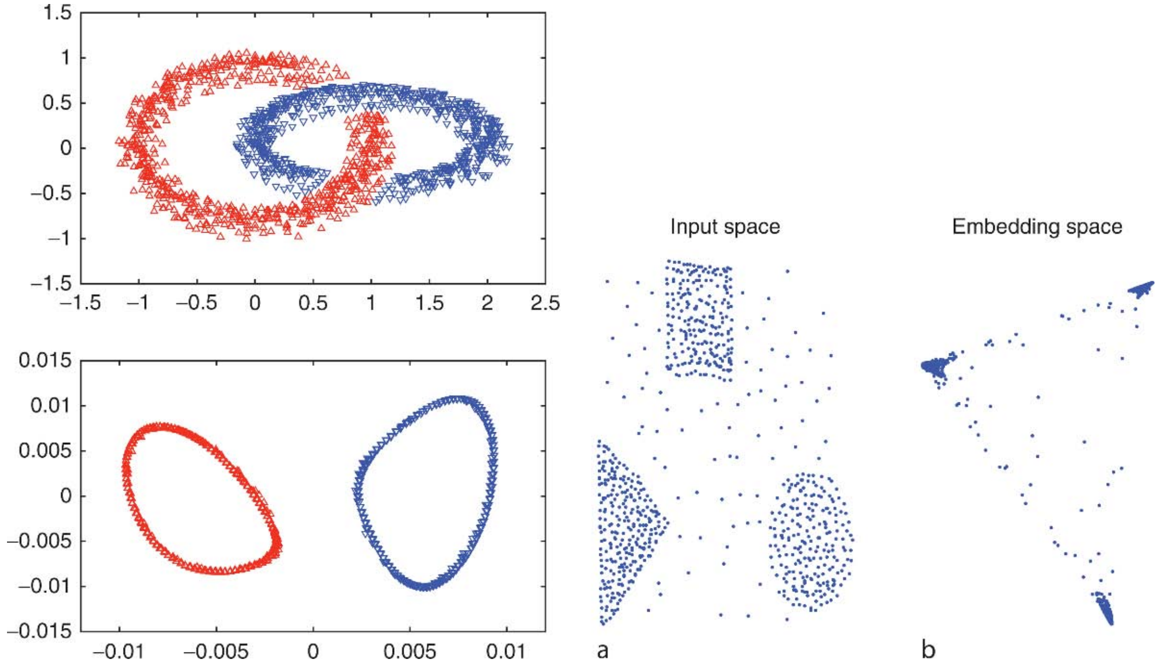
Metric scaling is the simplest form of *multidimensional scaling* and is also widely used in applications. Given a square matrix of pairwise distances (d_{ij}), the task is to embed the objects onto a lower dimensional embedding space where $||\mathbf{y}_i - \mathbf{y}_j|| \approx d_{ij}$.

Metric scaling is computed as the following. Let $A = (a_{ij}), a_{ij} = -\frac{1}{2} d_{ij}^2$. Compute B matrix: $B = (b_{ij})$, $b_{ij} = a_{ij} - \frac{1}{n}\sum_{i=1}^n a_{ij} - \frac{1}{n}\sum_{j=1}^n a_{ij} + \frac{1}{n^2}\sum_{i=1}^n \sum_{j=1}^n a_{ij}$. Compute the eigenvalues λ_k and eigenvectors $\mathbf{u}_k$, i.e., $B = \sum_{k=1}^n \lambda_k \mathbf{u}_k \mathbf{u}_k^T$. If $\lambda_k \geq 0, \forall k$, then the coordinates of object i in the embedding space is given by

$$\mathbf{y}_i = [\lambda_1^{1/2} \mathbf{u}_1(i), \ldots, \lambda_m^{1/2} \mathbf{u}_m(i)]^T$$

m is the embedding dimension. It can be verified that $||\mathbf{y}_i - \mathbf{y}_j|| = d_{ij}$. In general, only a fraction of the largest $\lambda_k \geq 0$, and only positive eigenvalue subspace is embedded into these.

Most linear dimension reduction techniques are developed in early days from 1940s to 1970s. PCA, LDA, metric scaling are the most widely techniques. Many other techniques were also developed. *Canonical correlation analysis* extract linear dimensions that best capture the correlation between two set of variables. *Independent component analysis* extract linear dimensions one after another, using non-Gaussian criteria. *Partial least squares* constructs subspace similarly to PCA, but uses class label information.



Dimension Reduction Techniques for Clustering. Figure 2. Left (*upper panel*): Dataset A in 3D space. 700 data points distributed on two interlocking rings. Left (*lower panel*): Dataset A in eigenspace ($\mathbf{f}_2$, $\mathbf{f}_3$). Middle: Dataset B as shown in 2D space. Right: Dataset B in eigenspace ($\mathbf{f}_2$, $\mathbf{f}_3$).

Laplacian Embedding and Other Nonlinear Dimension Reduction

Laplacian Embedding The input is a square matrix W of pairwise similarities among n objects. We view W as the edge weights on a graph with n nodes. The task is to embed the nodes of the graph in 1D space, with coordinates $(x_1,...,x_n)$. The objective is that if i, j are similar (i.e., w_{ij} is large), they should be adjacent in embedded space, i.e., $(x_i - x_j)^2$ should be small. This is achieved by minimizing [8]

$$\min_{\mathbf{x}} J(\mathbf{x}) = \sum_{ij}(x_i - x_j)^2 w_{ij} = 2\sum_{ij} x_i(D-W)_{ij}x_j = 2\mathbf{x}^T(D-W)\mathbf{x}, \tag{7}$$

where $D = \mathrm{diag}(d_1,...,d_n)$ and $d_i == \sum_j W_{ij}$. The minimization of $\sum_{ij}(x_i - x_j)^2 w_{ij}$ would get $x_i = 0$ if there is no constraint on the magnitude of the vector $\mathbf{x}$. Setting the normalization $\sum_i x_i^2 = 1$, and the constraint $\sum x_i = 0$ (x_i is centered around 0), the solution is given by the eigenvectors of

$$(D-W)\mathbf{f} = \lambda\mathbf{f}. \tag{8}$$

$L = D - W$ is called graph Laplacian. The solution is given by $\mathbf{f}_2$ associated with the second smallest eigenvalue (also called the Fiedler vector in recognition of his contribution to the graph connectivity).

This can be generalized to embedding in k-D space, with coordinates $\mathbf{r}_i \in \Re^k$. Let $||\mathbf{r}_i - \mathbf{r}_j||$ be the Euclidean distance between nodes i, j. The embedding is obtained by optimizing

$$\min_R J(R) = \sum_{i,j=1}^{n} ||\mathbf{r}_i - \mathbf{r}_j||^2 w_{ij} = 2\sum_{i,j=1}^{n} \mathbf{r}_i^T (D-W)_{ij}$$

$$\mathbf{r}_j = 2\,\mathrm{Tr}\,R(D-W)R^T,\quad R \equiv (\mathbf{r}_1, \ldots, \mathbf{r}_n). \tag{9}$$

With the normalization constraints $RR^T = I$, the solution is given by eigenvectors: $R = (\mathbf{f}_2,...,\mathbf{f}_{k+1})^T$.

In solving (7) for 1D embedding, we may impose the normalization $\sum_i d_i x_i^2 = 1$, where $d_i = \sum_j w_{ij}$. With this condition, the solution for x of is given by the generalized eigenvalue problem

$$(D-W)\mathbf{u} = \lambda D\mathbf{u}. \tag{10}$$

For k-dimensional embedding of (9), the normalization is $RDR^T = I$. The solution is given by the eigenvectors: $R = (\mathbf{u}_2,...,\mathbf{u}_{k+1})^T$. This approach is motivated by the normalized cut clustering (see (12) below).

Figure 2 gives two examples of the Laplacian embedding. The left panel [5] shows that the Laplacian embedding can effectively separate two interlocking data rings. It also shows that the embedding is not topology preserving. The right panel shows the self-aggregation property [5] of the embedding: data points of the same cluster self-aggregate and collapse onto a single centroid.

Relation to Spectral Clustering The most interesting aspect of Laplacian embedding is related to graph clustering, and its predecessor, the graph partitioning. In early 1990s, high performance computing is a very active research. One of the challenging task is to partition a mesh/graph into equal-size partition so that the problem could be solved on distributed processors. The popular technique of spectral graph partitioning [10] utilizes the eigenvectors of the Laplacian (the Fiedler vector $\mathbf{f}_2$). Specifically, a graph can be effectively partitioned into two equal parts depending on the sign $\mathbf{f}_2$: node i belongs to partition A if $\mathbf{f}_2(i) \geq 0$; it belongs to partition B if $\mathbf{f}_2(i) < 0$.

Soon it is recognized [7] that the 2-way Ratio Cut clustering objective function

$$J_{ratio-cut} = \frac{s(A,B)}{|A|} + \frac{s(A,B)}{|B|}, \quad s(A,B) \equiv \sum_{i\in A}\sum_{j\in B} w_{ij} \tag{11}$$

can be solved by the same Fiedler vector $\mathbf{f}_2$. This 2-way clustering is generalized to multi-way clustering [3] using $K - 1$ eigenvectors $(\mathbf{f}_2,...,\mathbf{f}_K)$. $\mathbf{f}_1 = (1,...,1)/n^{1/2}$ is a constant vector. Later it was realized that using the sum of node degree to balance clusters has some advantages. This is the normalized cut [11]

$$J_{normalized-cut} = \frac{s(A,B)}{d_A} + \frac{s(A,B)}{d_B}, \quad d_A = \sum_{i\in A} d_i,$$
$$d_B = \sum_{i\in B} d_i, \quad d_i = \sum_j w_{ij}. \tag{12}$$

The solution to this problem is given by the second eigenvector $\mathbf{u}_2$ in (10). The K-way clustering uses eigenvectors $(\mathbf{u}_2,...,\mathbf{u}_K)$. By 2001, the Laplacian based clustering methodology is fully developed.

Other Nonlinear Embedding Methods

Nonlinear embedding is a rather diverse and rapidly growing research area. They include (i) Multidimensional scaling, which embed a set of objects in in a lower-dimensional space while preserving the given pair-wise distance; (ii) Extension of PCA to nonlinear case, such as principal curves, kernel PCA, etc; (iii) Manifold learning, which uncovers a low-dimensional manifold embedded in the high-dimensional data space; isomap, local linear embedding, tangent space alignment, etc. (iv) Many other approaches, such as neuronal network based approach, called nonlinear PCA, etc.

A short description for some approaches. *Isomap* uses geodesic distances along the manifold by constructing the kNN subgraphs as the lower-dimensional manifold. It then use metric scaling to map objects into metric space. *Local Tangent Space Alignment* is a rigorous approach by building the local tangent spaces (which are local PCAs) and alignment them into a global system. *Kernel PCA* computes principal eigenvectors of the kernel matrix and embed in the eigenvector space. It should noted that even though manifold learning algorithm uncover nonlinear data structures, their primary goal is not necessarily data clustering.

Key Applications

PCA is used widely in a broad range of applications, from computer vision to text mining, gene expression profiles. Any data with high dimensions are often preprocessed with PCA. Laplacian embedding is used for network analysis, graph clustering.

Cross-references

- ▶ Dimensionality Reduction
- ▶ *K*-Means and *K*-Medoids
- ▶ Multidimensional Scaling
- ▶ Principal Component Analysis
- ▶ Social Networks

Recommended Reading

1. Alpert C.J. and Kahng A.B. Recent directions in netlist partitioning: a survey. Integ. VLSI J., 19:1–81, 1995.
2. Belkin M. and Niyogi P. Laplacian eigenmaps and spectral techniques for embedding and clustering. In Advances in Neural Information Processing Systems 14, 2001.
3. Chan P.K., Schlag M., and Zien J.Y. Spectral k-way ratio-cut partitioning and clustering. IEEE Trans. CAD-Integ. Circuit. Syst., 13:1088–1096, 1994.

4. Ding C. and He X. K-means clustering and principal component analysis. In Proc. 21st Int. Conf. on Machine Learning, 2004.
5. Ding C., He X., Zha H., and Simon H. Unsupervised learning: self-aggregation in scaled principal component space. Principles of Data Mining and Knowledge Discovery, 6th European Conf., 2002, pp. 112–124.
6. Fiedler M. Algebraic connectivity of graphs. Czech. Math. J., 23:298–305, 1973.
7. Hagen M. and Kahng A.B. New spectral methods for ratio cut partitioning and clustering. IEEE. Trans. Comput. Aided Desig., 11:1074–1085, 1992.
8. Hall K.M. R-dimensional quadratic placement algorithm. Manage. Sci., 17:219–229, 1971.
9. Ng A.Y., Jordan M.I., and Weiss Y. On spectral clustering: Analysis and an algorithm. In Advances in Neural Information Processing Systems 14, 2001.
10. Pothen A., Simon H.D., and Liou K.P. Partitioning sparse matrices with egenvectors of graph. SIAM J. Matrix Anal. Appl., 11:430–452, 1990.
11. Shi J. and Malik J. Normalized cuts and image segmentation. IEEE. Trans. Pattern Anal. Mach. Intell., 22:888–905, 2000.

Dimension-Extended Topological Relationships

ELISEO CLEMENTINI
University of L'Aquila, L'Aquila, Italy

Definition

This definition includes a group of models for topological relationships that have in common the use of two topological invariants – the set intersection empty/non empty content and the dimension – for distinguishing various relationships between spatial objects. These models had a strong impact in database technology and the standardization process.

Historical Background

Early descriptions of topological relationships (e.g., [10]) did not have enough formal basis to support a spatial query language, which needs formal definitions in order to specify exact algorithms to assess relationships. The importance of defining a sound and complete set of topological relationships was recognized in [13]. The first formal models were all based on point-set topology. In [12], the authors originally described the 4-intersection model (4IM) for classifying topological relationships between one-dimensional intervals. In [8], the authors adopted the same method for classifying topological relationships between regions. The 9-intersection model (9IM) is an extension of the 4IM based on considering the exterior of objects, besides interior and boundary [9]. In [6], the authors described the dimension-extended method (DEM), so called because they extended the 4IM with the dimension of the intersections. In the same paper, they introduced the Calculus-Based Method (CBM), made up of five relations and three boundary operators. A combination of the DEM and the 9IM was called the DE + 9IM in [1]. In this latter paper, the authors proved that the CBM was more expressive than the 4IM, 9IM, and DEM and was equivalent to the DE + 9IM. By expressive power they meant the number of topological relationships the models were able to distinguish. Later on, several extensions of all these models were developed: for example, the extension of the CBM to composite regions [5] and complex objects [2], and the extension of the 4IM to regions with holes [7].

Foundations

In the following, simple objects of the plane are considered, that is, regularly closed regions with connected interior and exterior, curved lines with only two endpoints and without self-intersections, and single points. The symbol λ will be used to denote any geometric object (simple region, simple line, or point), and $\partial\lambda$, λ°, $\overline{\lambda}$, λ^{-}will denote the boundary, the interior, the closure, and the exterior of λ, respectively. The function dim(λ)returns the dimension of λ, with possible values in the two-dimensional space of 0, 1, 2 or nil (−) for the empty set. In case the object λ consists of multiple parts, the highest dimension is returned.

To assess the relationship between two geometric objects, various point-sets can be considered, which can be empty or non-empty. This is generally called the content invariant and can be calculated for several sets: intersections, set differences, symmetric differences [11]. The most convenient one is the intersection, since it gives a comprehensive categorization of topological relationships. Six groups of relationships can be distinguished: region/region (R/R), line/region (L/R), point/region (P/R), line/line (L/L), point/line (P/L), and point/point (P/P).

Definition 1. The 4IM is a 2 × 2 matrix of the intersections of the interiors and boundaries of the two objects λ_1 and λ_2:

$$\begin{pmatrix} \lambda_1^\circ \cap \lambda_2^\circ & \lambda_1^\circ \cap \partial\lambda_2 \\ \partial\lambda_1 \cap \lambda_2^\circ & \partial\lambda_1 \cap \partial\lambda_2 \end{pmatrix}$$

Each intersection may be empty (∅) or non-empty (¬∅), resulting in a total of $2^4 = 16$ combinations. Each case is represented by a matrix of values. It is possible to apply some simple geometric constraints to assess that not all combinations are possible. For example, for the R/R group the 4IM is able to recognize eight different relationships. All 16 combinations are instead possible for the L/L group. The geometric criteria to discover real cases are discussed in [1]. Overall, 43 real cases can be identified (see Table 1).

Definition 2. The 9IM is a 3 × 3 matrix containing the empty/non-empty values for interior, boundary, and exterior intersections:

$$\begin{pmatrix} \lambda_1^\circ \cap \lambda_2^\circ & \lambda_1^\circ \cap \partial\lambda_2 & \lambda_1^\circ \cap \lambda_2^- \\ \partial\lambda_1 \cap \lambda_2^\circ & \partial\lambda_1 \cap \partial\lambda_2 & \partial\lambda_1 \cap \lambda_2^- \\ \lambda_1^- \cap \lambda_2^\circ & \lambda_1^- \cap \partial\lambda_2 & \lambda_1^- \cap \lambda_2^- \end{pmatrix}$$

By considering the empty or nonempty content of such nine sets, the total is $2^9 = 512$ theoretical combinations. Excluding the impossible cases, 68 possible cases are remaining, as shown in Table 1.

The introduction of other invariants allows finer topological distinctions. A refinement of the content invariant is given by the dimension of each intersection set.

Definition 3. The DEM is a 2 × 2 matrix containing the dimension of the intersections of the interiors and boundaries of the two objects λ_1 and λ_2:

$$\begin{pmatrix} \dim(\lambda_1^\circ \cap \lambda_2^\circ) & \dim(\lambda_1^\circ \cap \partial\lambda_2) \\ \dim(\partial\lambda_1 \cap \lambda_2^\circ) & \dim(\partial\lambda_1 \cap \partial\lambda_2) \end{pmatrix}$$

Theoretically, with the four possible values for the dimension the DEM matrix might result into $4^4 = 256$ different cases. Geometric criteria can be adopted to reduce this number of cases by referring to specific groups of relationships, for a total of 61 real cases (see Table 1).

Definition 4. The DE + 9IM is a 3 × 3 matrix containing the dimension of interior, boundary, and exterior intersections:

$$\begin{pmatrix} \dim(\lambda_1^\circ \cap \lambda_2^\circ) & \dim(\lambda_1^\circ \cap \partial\lambda_2) & \dim(\lambda_1^\circ \cap \lambda_2^-) \\ \dim(\partial\lambda_1 \cap \lambda_2^\circ) & \dim(\partial\lambda_1 \cap \partial\lambda_2) & \dim(\partial\lambda_1 \cap \lambda_2^-) \\ \dim(\lambda_1^- \cap \lambda_2^\circ) & \dim(\lambda_1^- \cap \partial\lambda_2) & \dim(\lambda_1^- \cap \lambda_2^-) \end{pmatrix}$$

There are in general for this method $4^9 = 262144$ different cases. Reducing this number with geometric criteria, 87 real topological relationships are obtained (see Table 1).

The CBM is made up of five relations and three boundary operators. In [1], the authors proved that the CBM is equivalent to the DE + 9IM regarding the number of topological relationships these two models are able to express. The model was extended for complex objects in [2]. The definitions for simple objects are the following.

Definition 5. The *touch* relationship (it applies to the R/R, L/L, L/R, P/R, P/L groups of relationships, but not to the P/P group):

$$< \lambda_1, touch, \lambda_2 > \Leftrightarrow (\lambda_1^\circ \cap \lambda_2^\circ = \emptyset) \wedge (\lambda_1 \cap \lambda_2 \neq \emptyset).$$

Definition 6. The *in* relationship (it applies to every group):

$$< \lambda_1, in, \lambda_2 > \Leftrightarrow (\lambda_1 \cap \lambda_2 = \lambda_1) \wedge (\lambda_1^\circ \cap \lambda_2^\circ \neq \emptyset).$$

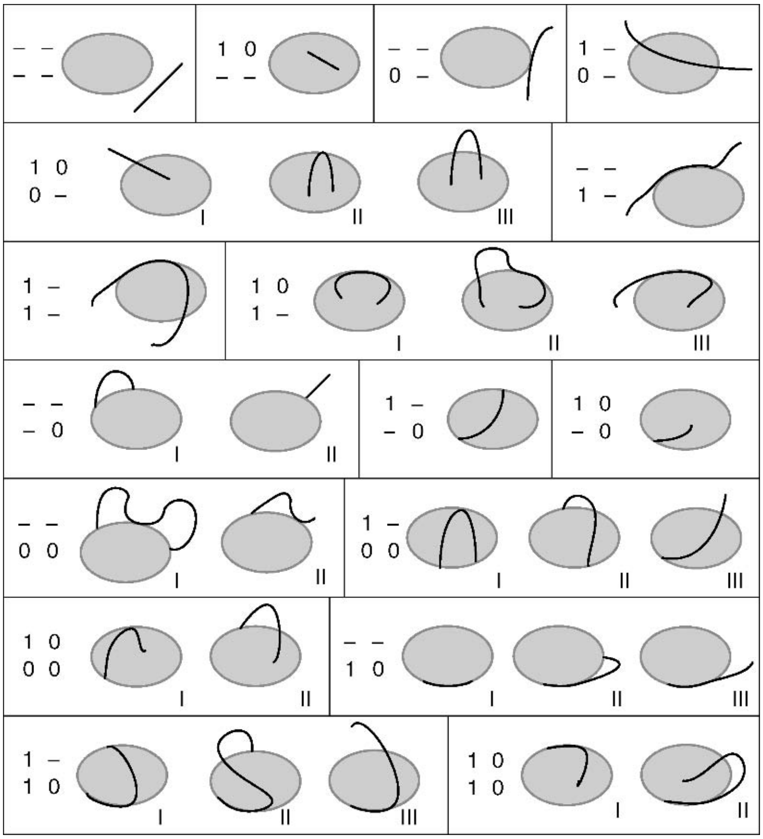
Definition 7. The *cross* relationship (it applies to the L/L and L/R groups):

$$\begin{aligned} < \lambda_1, cross, \lambda_2 > \Leftrightarrow & (\dim(\lambda_1^\circ \cap \lambda_2^\circ) \\ & < \max(\dim(\lambda_1^\circ), \dim(\lambda_2^\circ))) \\ & \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2). \end{aligned}$$

Definition 8. The *overlap* relationship (it applies to R/R and L/L groups):

Dimension-Extended Topological Relationships. Table 1. A summary of topological relationships for all models and for all groups between simple objects

Model/group	R/R	L/R	P/R	L/L	P/L	P/P	Total
4IM	8	11	3	16	3	2	43
9IM	8	19	3	33	3	2	68
DEM	12	17	3	24	3	2	61
DE + 9IM ≡ CBM	12	31	3	36	3	2	87



Dimension-Extended Topological Relationships. Figure 1. The 31 different L/R relationships of the DE + 9IM model. Each box contains cases belonging to the same DEM case.

$$
\begin{aligned}
< \lambda_1, \mathit{overlap}, \lambda_2 > \Leftrightarrow\ & (\dim(\lambda_1^\circ) = \dim(\lambda_2^\circ) \\
& = \dim(\lambda_1^\circ \cap \lambda_2^\circ)) \\
\wedge\, (\lambda_1 \cap \lambda_2 \neq \lambda_1) & \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2).
\end{aligned}
$$

Definition 9. The *disjoint* relationship (it applies to every group):

$$< \lambda_1, \mathit{disjoint}, \lambda_2 > \Leftrightarrow\ \lambda_1 \cap \lambda_2 = \emptyset.$$

Definition 10. The boundary operator b for a region A: The pair (A, b) returns the circular line ∂A.

Definition 11. The boundary operators *from* and *to* for a line L: The pairs (L, f) and (L, t) return the two endpoints of the set ∂L.

To illustrate some of the relationships, Fig. 1 shows the list of the 31 L/R cases for the DE + 9IM.

Key Applications

The various models illustrated so far have been used by the Open Geospatial Consortium (OGC) for the definition of topological relationships in spatial databases. The same definitions have been adopted by the International Organization for Standardization (ISO). Various spatial database systems (e.g., Oracle, IBM DB2, PostgreSQL) have adopted the definitions suggested by OGC or slight variations of them to define the topological operators included in their spatial query language.

Future Directions

The process of adding granularity to topological relationships could be further extended by introducing more refined topological invariants [3]. Another direction for further research on the side of adding spatial operators to query languages is to consider other categories of spatial relationships besides topological, such as projective and metric [4].

Cross-references

- ▶ Topological Relationships
- ▶ Vague Spatial Data Types

Recommended Reading

1. Clementini E. and Di Felice P. A comparison of methods for representing topological relationships. Inf. Sci., 3(3):149–178, 1995.
2. Clementini E. and Di Felice P. A model for representing topological relationships between complex geometric features in spatial databases. Inf. Sci., 90(1–4):121–136, 1996.
3. Clementini E. and Di Felice P. Topological invariants for lines. IEEE Trans. Knowl. Data Eng., 10:38–54, 1998.
4. Clementini E. and Di Felice P. Spatial operators. ACM SIGMOD Rec., 29:31–38, 2000.
5. Clementini E., Di Felice P., and Califano G. Composite regions in topological queries. Inf. Syst., 20(7):579–594, 1995.
6. Clementini E., Di Felice P., and van Oosterom P. A small set of formal topological relationships suitable for end-user interaction. In Proc. 3rd Int. Symp. Advances in Spatial Databases, 1993, pp. 277–295.
7. Egenhofer M.J., Clementini E., and Di Felice P. Topological relations between regions with holes. Int. J. Geogr. Inf. Syst., 8:129–142, 1994.
8. Egenhofer M.J. and Franzosa R.D. Point-set topological spatial relations. Int. J. Geogr. Inf. Syst., 5:161–174, 1991.
9. Egenhofer M.J. and Herring J.R. 1Categorizing Binary Topological Relationships Between Regions, Lines, and Points in Geographic Databases. Department of Surveying Engineering, University of Maine, Orono, ME, 1991.
10. Freeman J. The modelling of spatial relations. Comput. Graph. Image Process, 4:156–171, 1975.
11. Herring J.R. The mathematical modeling of spatial and non-spatial information in geographic information systems. In Cognitive and Linguistic Aspects of Geographic Space, D. Mark and A. Frank (eds.). Kluwer Academic, Dordrecht, 1991, pp. 313–350.
12. Pullar D.V. and Egenhofer M.J. Toward the definition and use of topological relations among spatial objects. In Proc. Third Int. Symp. on Spatial Data Handling, 1988, pp. 225–242.
13. Smith T. and Park K. Algebraic approach to spatial reasoning. Int. J. Geogr. Inf. Syst., 6:177–192, 1992.

Direct Attached Storage

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Synonyms

DAS

Definition

Direct Attached Storage is a dedicated storage device which is directly connected to a server. The term Direct Attached Storage is often abbreviated to DAS. Derived from the original meaning, the term sometimes refers to a conventional server-centric storage system in which DAS devices are mainly connected; this definition is used in the context of explaining and comparing storage network architectures.

Key Points

A typical DAS device is a SCSI storage device such as an internal disk drive, a disk array or a tape library that is connected only to a single server by a SCSI bus cable. Such a DAS device is accessed only by the server to which the device is directly connected. SCSI storage devices sometimes have two or more SCSI interfaces which can connect to different servers. But such a storage device is also considered a DAS device rather than a SAN device, because only one server can actually access the storage device and the other servers are merely prepared for fault tolerance. Recently small servers sometimes accommodate Advanced Technology Attachment/Integrated Drive Electronics (ATA/IDE) disk drives and Universal Serial Bus (USB) disk drives, which can be also considered DAS devices.

Cross-references

- ► Network Attached Storage
- ► Storage Area Network
- ► Storage Network Architectures

Recommended Reading

1. Storage Network Industry Association. The Dictionary of Storage Networking Terminology. Available at: http://www.snia.org/.
2. Troppens U., Erkens R., and Müller W. Storage Networks Explained. Wiley, New York, 2004.

Direct Manipulation

Alan F. Blackwell[1], Maria Francesca Costabile[2]
[1]University of Cambridge, Cambridge, UK
[2]University of Bari, Bari, Italy

Synonyms

Graphical interaction; Desktop metaphor

Definition

The term *direct manipulation* was introduced by Shneiderman [2,3] to describe user interfaces that offer the following desirable properties:

1. Visibility of the objects and actions of interest
2. Physical actions on the object of interest instead of complex syntax

3. Effects of those actions that are rapid, incremental and reversible

Direct manipulation is generally associated with Graphical User Interfaces, although the above properties are also beneficial in interfaces where the user is working with text or numeric data. Direct manipulation has been most popularly applied in the "desktop metaphor" for managing disk files, a visual representation where the objects of interest are the user's files, and they are therefore continuously presented on the screen. This was an improvement over disk operating system consoles in which file listings were only transient results of a directory listing command.

The currently most common direct manipulation interfaces are called WIMP interfaces, referring to the interaction devices they use, namely windows, icons, menus and pointers. However, direct-manipulation ideas are at the core of many advanced non-desktop interfaces. Virtual reality, augmented reality, tangible user interfaces are newer concepts that extend direct manipulation.

Key Points

The main alternatives to direct manipulation are command line interfaces and programming languages. These are characterized by abstract syntax, representation of computational processes rather than data of interest, and effects that will take place in the future rather than immediately, perhaps suddenly and irreversibly. Some GUIs (e.g., modal dialog boxes) have these latter properties, in which case they will not provide the benefits of direct manipulation. For many database users, the object of interest is their data, rather than the computational features of the database. This offers a challenge to designers of interactive database systems, for whom the system itself might be an object of interest.

Direct manipulation interfaces represent a second generation, which evolved from the first generation of command line interfaces. DM interfaces often make interaction easier for users who are not computer science experts, by allowing them to point at and move objects rather than instructing the computer by typing commands in a special syntax.

Hutchins, Hollan and Norman [1] proposed a generalization of the direct manipulation principles, in terms of minimizing the cognitive effort that is required to bridge the gulf between the user's goals and the way those goals must be specified to the system. They drew attention to a *gulf of execution*, which must be bridged by making the commands and mechanisms of the system match the thoughts and goals of the user, and a *gulf of evaluation*, bridged by making the output present a good conceptual model of the system that is readily perceived, interpreted, and evaluated. Both gulfs can be bridged to some degree by use of an appropriate visual metaphor.

Cross-references

▶ Usability
▶ Visual Interaction
▶ Visual Interfaces
▶ Visual Metaphor

Recommended Reading

1. Hutchins E.L., Hollan J.D., and Norman D.A. Direct manipulation interfaces. In User Centered System Design, New Perspectives on Human-Computer Interaction, Norman, D.A. Draper (eds.). S.W. Lawrence Erlbaum, Hillsdale, NJ, 1986.
2. Shneiderman, B. The future of interactive systems and the emergence of direct manipulation. Behav. Inf. Technol., 1:237–256, 1982.
3. Shneiderman, B. Direct manipulation: a step beyond programming languages. IEEE Comput., 16(8):57–69, 1983.

Direct Manipulation Interfaces

▶ Visual Interfaces

Directional Relationships

▶ Cardinal Direction Relationships

Dirichlet Tessellation

▶ Voronoi Diagram

Disaster Recovery

Kenichi Wada
Hitachi Limited, Tokyo, Japan

Definition

The recovery of necessary data, access to that data, and associated processing through a comprehensive process in which a redundant site (including both equipment and work space) is set up, and operations are recovered to enable business operations to continue after a loss of all or part of a data center by disaster including fire, earthquake, hurricane, flood, terrorism and power grid failure. Such a recovery involves not only an essential set of data but also all the hardware and software needed to continue processing of the data.

Key Points

Disaster recovery should be a key focus of an organization's business continuity plan in case of disaster the following are key metrics:

RPO (Recovery Point Objective): The maximum acceptable time period prior to a failure or disaster during which changes to data may be lost as consequence of recovery. At a minimum, all data changes that occur before this period preceding the failure or disaster will be available after data recovery. RTO (Recovery Time Objective): The maximum acceptable time period required to bring one or more applications and associated data back from an outage to a correct operational state.

The most common disaster recovery plans include the following strategies:

- Backups are made to tape and sent off-site at regular intervals.
- Backups are made to disk on-site and automatically copied to an off-site disk, or made directly to an off-site disk.
- Data is replicated to an off-site location, using replication method including database replication, file system replication and replication by storage system.

Cross-references

► Backup and Restore
► Replication

Disclosure Risk

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona, Catalonia

Synonyms

Re-identification risk; Attribute disclosure; Identity disclosure

Definition

In the context of statistical disclosure control, disclosure risk can be defined as the risk that a user or an intruder can use the protected dataset $\mathbf{V}'$ to derive confidential information on an individual among those in the original dataset $\mathbf{V}$. This approach to disclosure risk was formulated in Dalenius [1].

Key Points

Disclosure risk can be regarded from two different perspectives, according to Paass [2]:

1. Attribute disclosure. Attribute disclosure takes place when an attribute of an individual can be determined more accurately with access to the released statistic than it is possible without access to that statistic.

2. Identity disclosure. Identity disclosure takes place when a record in the protected dataset can be linked with a respondent's identity. Two main approaches are usually employed for measuring identity disclosure risk: uniqueness and re-identification.

2.1. Uniqueness Roughly speaking, the risk of identity disclosure is measured as the probability that rare combinations of attribute values in the released protected data are indeed rare in the original population the data come from. This approach is typically used with nonperturbative statistical disclosure control methods and, more specifically, sampling. The reason that uniqueness is not used with perturbative methods is that, when protected attribute values are perturbed versions of original attribute values, it makes no sense to investigate the probability that a rare combination of protected values is rare in the original dataset, because *that* combination is most probably *not found* in the original dataset.

2.2. Re-identification This is an empirical approach to evaluate the risk of disclosure. In this case, software is constructed to estimate the number of re-identifications that might be obtained by a specialized intruder. The use of re-identification methods as a way to measure disclosure risk goes back, at least, to Spruill [3]. Re-identification methods provide a more unified approach than uniqueness methods because the former can be applied to any kind of masking and not just to non-perturbative masking. Moreover, re-identification can also be applied to synthetic data (see e.g., [4]).

Cross-references

- ▶ Inference Control in Statistical Databases
- ▶ Information Loss Measures
- ▶ Microdata
- ▶ Record Matching
- ▶ SDC Score

Recommended Reading

1. Dalenius T. Towards a methodology for statistical disclosure control. Statistisk Tidskrift, 5:429–444, 1977.
2. Paass G. Disclosure risk and disclosure avoidance for microdata. J. Bus. Econ. Stat., 6:487–500, 1985.
3. Spruill N.L. The confidentiality and analytic usefulness of masked business microdata. In Proc. Section on Survey Research Methods, Alexandria, VA. American Statistical Association, 1983, pp. 602–607.
4. Winkler W.E. Re-identification methods for masked microdata. In Privacy in Statistical Databases, J. Domingo-Ferrer and V. Torra (eds.). LNCS, vol. 3050, Springer, Berlin Heidelberg, 2004, pp. 216–230.

DISCO

▶ Discovery

Discounted Cumulated Gain

Kalervo Järvelin, Jaana Kekäläinen
University of Tampere, Tampere, Finland

Synonyms

Normalized discounted cumulated gain (nDCG); Discounted cumulated gain (DCG)

Definition

Discounted Cumulated Gain (DCG) is an evaluation metric for information retrieval (IR). It is based on non-binary relevance assessments of documents ranked in a retrieval result. It assumes that, for a searcher, highly relevant documents are more valuable than marginally relevant documents. It further assumes, that the greater the ranked position of a relevant document (of any relevance grade), the less valuable it is for the searcher, because the less likely it is that the searcher will ever examine the document – and at least has to pay more effort to find it. DCG formalizes these assumptions by crediting a retrieval system (or a query) for retrieving relevant documents by their (possibly weighted) degree of relevance which, however, is discounted by a factor dependent on the logarithm of the document's ranked position. The steepness of the discount is controlled by the base of the logarithm and models the searcher's patience in examining the retrieval result. A small base (say, 2) models an impatient searcher while a large base (say, 10) a patient searcher.

In its normalized form, as Normalized Discounted Cumulated Gain (nDCG), the actual DCG performance for a query is divided by the ideal DCG performance for the same topic, based on the recall base of the topic in a test collection.

Historical Background

Modern large retrieval environments tend to overwhelm their users by their large output. Since all documents are not equally relevant, highly relevant documents, or document components, should be identified and ranked first for presentation to the users. In order to develop IR techniques in this direction, it is necessary to develop evaluation approaches and methods that credit IR methods for their ability to retrieve highly relevant documents and rank them higher. For this goal, non-binary or graded relevance assessments and evaluation measures are needed. Most traditional IR evaluation measures, e.g., precision and recall, are based on binary relevance. The nDCG measure family presented in this entry is one solution for measuring IR effectiveness with graded relevance.

Foundations

Ranked Retrieval Result

In an IR experiment, using a test collection, a topic set and a recall base for each topic, the retrieval system gives for each query representing a topic a ranked output, which in the DCG based evaluation is examined from ranked position 1 to position n for each query.

Document Relevance Scores and Weights

Documents have non-binary relevance scores given in the recall base. Their range may be any continuous scale of real numbers, e.g., $[-10.0, +10.0]$, $[-1.0, +1.0]$, or $[0.0, 1.0]$, or a discreet integer scale, say $\{-3,\ldots,+3\}$ or $\{0,\ldots,10\}$. In the evaluation, the relevance scores may be reweighed if one wishes to emphasize the differences of relevance scores from the user point of view. In the following it is assumed that the relevance scores 0–3 are used (3 denoting high value, 0 no value).

Gain Vector

In the DCG evaluation, the relevance score of each document, or its reweighed value, is used as a gained value measure for its ranked position in the result. Assuming relevance scores 0–3 and result lists up to rank 200, one obtains corresponding gain vectors of 200 components each having the value 0, 1, 2 or 3. For example: G' = <3, 2, 3, 0, 0, 1, 2, 2, 3, 0,. . . >.

Cumulated Gain Vector

The cumulated gain at ranked position i is computed by summing from position 1 to i when i ranges from 1 to n, e.g., n = 200. Formally, the position i in the gain vector G is denoted by G[i]. Now the cumulated gain vector CG is defined recursively as the vector CG where:

$$\mathrm{CG}[i] = \begin{cases} \mathrm{G}[1], \ \text{if } i = 1 \\ \mathrm{CG}[i-1] + \mathrm{G}[i], \ \text{otherwise} \end{cases}$$

For example, from G' the vector CG' = <3, 5, 8, 8, 8, 9, 11, 13, 16, 16, . . . > is obtained. The cumulated gain at any rank may be read directly, e.g., at rank 7 it is 11.

Discounting Principle

The discounting principle states that the greater the rank of a document in the retrieval result, the smaller share of the document score is added to the cumulated gain. A discounting function is needed which progressively reduces the document score as its rank increases but not too steeply (e.g., as division by rank) to allow for searcher persistence in examining further documents. A simple way of discounting with this requirement is to divide the document score by the log of its rank. By selecting the base of the logarithm, sharper or smoother discounts can be computed to model varying searcher behavior.

Discounted Cumulated Gain Vector

Formally, if b denotes the base of the logarithm, the cumulated gain vector with discount is defined recursively as the vector DCG where:

$$\mathrm{DCG}[i] = \begin{cases} \mathrm{CG}[i], \ \text{if } i < b \\ \mathrm{DCG}[i-1] + \mathrm{G}[i]/\log_b i, \text{if } \ i \geq b \end{cases}$$

One must not apply the logarithm-based discount at rank 1 because $\log_b 1 = 0$. Moreover, the discount is not applied for ranks less than the logarithm base (that would give them a boost). This is also realistic, since the larger the base, the smaller the discount and the more likely the searcher is to examine the results at least up to the base rank (say 10).

For example, let $b = 2$. From G' given in the preceding section one obtains DCG' = <3, 5, 6.89, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61, . . .>.

Average Vector

In an IR experiment, tests on IR methods are typically run with a set of test topics. To obtain an understanding of the average performance of an IR method, one needs average vectors across a query set. To compute the averaged vectors, one needs the vector sum operation and vector multiplication by a constant. Let V = <v_1, v_2,...,v_k> and W = <w_1, w_2,...,w_k> be two vectors. Their sum is the vector V + W = <v_1+ w_1, v_2+ w_2,...,v_k+ w_k>. For a set of vectors $\mathcal{V}$ = {V_1, V_2,...,V_n}, each of k components, the sum vector is generalised as $\Sigma_{V \in \mathcal{V}}$ V = V_1 + V_2+ ... + V_n. The multiplication of a vector V = <v_1, v_2,...,v_k> by a constant r is the vector r*V = <r*v_1, r*v_2,...,r*v_k>. The average vector **AV** based on vectors $\mathcal{V}$= {V_1, V_2,...,V_n}, is given by the function *avg-vect*($\mathcal{V}$):

$$avg\text{-}vect(v) = |\,v|^{-1}\ {}^{*}\Sigma_{V \in v}\mathrm{V}$$

Now the average CG and DCG vectors for vector sets **CG** and **DCG**, over a set of test queries, are computed by *avg-vect*(**CG**) and *avg-vect*(**DCG**).

Ideal Vector

In order to normalize the actual CG and DCG vectors one compares them to the theoretically best possible vectors for each topic. The latter vectors are constructed by arranging the recall base of each topic in descending order by relevance and then turning it into a gain vector, CG vector and DCG vector as described above. A sample ideal gain vector is: I' = <3, 3, 3, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0, . . .> and its CG vector is $\mathrm{CG_I}$' = <3, 6, 9, 11, 13, 15, 16, 17, 18, 19, 19, 19, 19, . . .>.

Normalized Vector

The (D)CG vectors for each IR technique can be normalized by dividing them by the corresponding ideal (D)CG vectors, component by component. In this way, for any vector position, the normalized value 1 represents ideal performance, and values in the range [0,1) the share of ideal performance cumulated by each technique. Given an (average) (D)CG vector

$V = <v_1, v_2,...,v_k>$ of an IR technique, and the (average) (D)CG vector $I = <i_1, i_2,...,i_k>$ of ideal performance, the normalized performance vector n(D)CG is obtained by the function:

$$\mathit{norm\text{-}vect}(V, I) = < v_1/i_1, v_2/i_2,...,v_k/i_k >$$

Normalized Discounted Cumulated Gain (nDCG) Vector

To assess whether two IR methods are significantly different in effectiveness from each other or not, when measured by DCG, one uses the normalized vectors because the (D)CG vectors are not relative to an ideal. Given an (average) DCG vector V of an IR method, and the (average) DCG vector I of ideal performance, the normalized performance vector nDCG is obtained by *norm-vect*(V, I).

For example, based on CG' and CG_I' from above, one obtains the normalized CG vector nCG' = *norm-vect*(CG', CG_I') = <1, 0.83, 0.89, 0.73, 0.62, 0.6, 0.69, 0.76, 0.89, 0.84, . . . >.

The Average Normalized Discounted Cumulated Gain Indicator

The average of a (n)(D)CG vector, up to a given ranked position, summarizes the vector (or performance) and is analogous to the non-interpolated average precision of a document cut-off value (DCV) curve up to the same given ranked position. The average of a (n)(D)CG vector V up to the position *k* is given by:

$$\mathit{avg\text{-}pos}(V, k) = k^{-1} * \sum_{i=1...k} V[i]$$

These vector averages can be used in statistical significance tests in the same way as average precision in IR evaluation.

Properties of (n)(D)CG

The strengths of the proposed CG, DCG, nCG, and nDCG measures can now be summarized as follows:

- They combine the degree of relevance of documents and their rank (affected by their probability of relevance) in a coherent way.
- At any number of retrieved documents examined (rank), CG and DCG give an estimate of the cumulated gain as a single measure no matter what is the recall base size.
- They are not heavily dependent on outliers (relevant documents found late in the ranked order) since they focus on the gain cumulated from the beginning of the result up to any point of interest.
- They are obvious to interpret, they are more direct than precision-recall curves by explicitly giving the number of documents for which each n(D)CG value holds. Precision-recall curves do not make the number of documents explicit for given performance and may therefore mask bad performance [7].

In addition, the DCG measure has the following further advantages:

- It realistically weights down the gain received through documents found later in the ranked results.
- It allows modeling searcher persistence in examining long ranked result lists by adjusting the discounting factor.

Further, the normalized nCG and nDCG measures support evaluation:

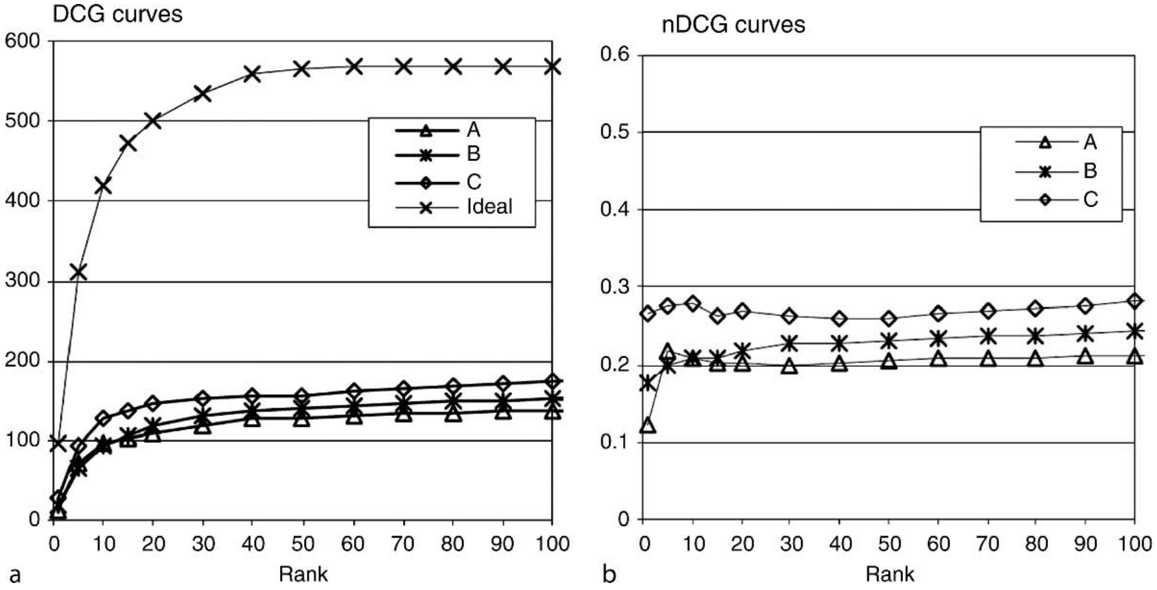
- They represent performance as relative to the ideal based on a known (possibly large) recall base of graded relevance assessments.
- The performance differences between IR techniques are also normalized in relation to the ideal thereby supporting the analysis of performance differences.

The following issues concerning the (n)(D)CG measure have been discussed in the literature:

1. Averaging over topics is not statistically reliable because the number of relevant documents (recall base size) varies by topics.
2. DCG does not discount the gain when the rank is smaller than the logarithm base.
3. Interpretation of non-normalized versions of the measure (CG, DCG) is difficult.
4. The measures should be robust when relevance judgements are incomplete.

These issues are discussed briefly below.

1. Averaging. This issue is discussed by, e.g., Hull [2] and Sakai [9]. Averaging over topics has been considered problematic when a fixed length for the result list (i.e., a fixed cut-off value, DCV) is used. Say, one evaluates effectiveness at DCV 5. One topic has five relevant documents with relevance score 3 retrieved at positions 1–5; another topic has 100 relevant documents with relevance score 3 and five of them



Discounted Cumulated Gain. Figure 1. (a) DCG curves. (b) nDCG curves.

are retrieved at positions 1–5. The (n)DCG values for the topics will be the same at DCV 5. From the point of view of searcher's effort this is correct; however, if the system-oriented effectiveness as an equal share of the relevant documents is aimed at, evaluation with fixed DCV should be avoided. With high DCVs, e.g., 500 or 1,000, the problem is less acute but the user-orientation is also lost.

A related problem is caused by the variation in the number of documents of different relevance level: As the number of most relevant documents tends to be low, weighting them strongly might lead to instability in evaluation (a loss of one highly relevant document in the top ranks hurts effectiveness badly) [11]. As a remedy large topic pools with varying recall bases could be used in evaluation.

One should also bear in mind the difference between (n)DCG curves and averaging the scores over topics at a given rank. The curves visualize the performance over a range of ranks and are useful to reveal crossovers concealed in averages. The single figure averages are needed to give concise information and for statistical testing.

2. Discounting. The original formulation of the (n)DCG, given above, does not perform discounting for ranks less than the base of the discount logarithm. When the base is relatively large, say 10, modeling a patient searcher, the 10 first ranks avoid discounting. Some scholars have been concerned about this feature [10]. It can be solved by modifying the discounting factor $\log_b i$ to the form $(1 + \log_b i)$ [8]. By doing so the first case (with the condition $i < b$) of the DCG formula can be omitted: all ranks are consistently discounted.
3. Interpretation of the scores. The CG and DCG measures are comparable only within one test setting and relevance weighting scheme. The relevance grades and their weights should be reported in order to make CG and DCG curves interpretable. Further, topics with large recall bases yield high CG and DCG figures and affect averages over all topics. The normalized versions are recommendable when averaging over topics or comparing over test settings is needed.
4. Robustness. An evaluation measure should be robust with relation to missing relevance judgements since, in practice, the whole test collection is seldom assessed for relevance. The pooling method for gathering documents for relevance judgement, and the later reuse of the test collection with new systems or methods leads unavoidably to a situation where all documents in the result lists are not assessed for relevance. The (n)DCG measure has been found robust in this respect, see [1,10].

Key Applications

The (n)(D)CG measures have been applied in traditional laboratory oriented evaluation with stable test collections when graded relevance assessment are available. They have also gained popularity in web IR evaluation, and interactive IR test settings.

Sample (n)DCG curves for three TREC 7 runs (A, B, C) are given in Fig. 1a and b. The measures use graded relevance with four relevance levels. The levels from non-relevant to the most relevant are weighted 0–1–10–100; logarithm base for discounting is 2.

Future Directions

Session-Based DCG

IR evaluation by (n)(D)CG assumes one query per topic/session. In real life however, interactive searchers often use multiple queries through reformulation and/or relevance feedback until they are satisfied or give up and move on to other means of information access. Evaluation metrics assuming one query per topic are insufficient in multiple query session evaluation, where the searcher's reformulation and feedback effort matters. Moreover, due to various reasons, the first queries often are unsuccessful. In real life, also stopping decisions are individual and variable and depend on many factors, including personal traits, task, context, and retrieval results. To overcome this limitation, it is possible to extend the (n)(D)CG into a session-based metric for multiple interactive queries. Such an extended, session-based DCG metric would incorporate query sequences as a further dimension in evaluation scenarios, allowing one to further discount any relevant documents found only after additional searcher effort, i.e., feedback or reformulation. The rationale here is that an IR system (or searcher-system combination) should be rewarded less for relevant results found by later queries.

Cross-references

- ▶ Information Retrieval
- ▶ Precision
- ▶ Recall
- ▶ Standard Effectiveness Measures

Recommended Reading

1. Bompada T., Chang C., Chen J., Kumar R., and Shenoy R. On the robustness of relevance measures with incomplete judgments. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 359–366.
2. Hull D. Using statistical testing in the evaluation of retrieval experiments. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 329–338.
3. Järvelin K. and Kekäläinen J. IR evaluation methods for retrieving highly relevant documents. In Proc. 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2000, pp. 41–48.
4. Järvelin K. and Kekäläinen J. Cumulated gain-based evaluation of IR techniques. ACM Trans. Inf. Syst., 20(4):422–446, 2002.
5. Kekäläinen J. Binary and graded relevance in IR evaluations – comparison of the effects on ranking of IR systems. Information Process. Manag., 41(5):1019–1033, 2005.
6. Kekäläinen J. and Järvelin K. Using graded relevance assessments in IR Evaluation. J. Am. Soc. Inf. Sci. Technol., 53(13):1120–1129, 2002.
7. Losee R.M. Text retrieval and filtering: Analytic models of performance. Kluwer Academic, Boston, MA, 1998.
8. Sakai T. Average gain ratio: A simple retrieval performance measure for evaluation with multiple relevance levels. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 417–418.
9. Sakai T. On the reliability of information retrieval metrics based on graded relevance. Inf. Process. Manag. 43(2):531–548, 2007.
10. A suggestion by Susan Price (Portland State University, Portland, OR, USA), May 2007, (private communication).
11. Voorhees E. Evaluation by highly relevant documents. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 74–82.

Discovery

Schahram Dustdar[1], Christian Platzer[1]
Bernd J. Krämer[2]
[1]Technical University of Vienna, Vienna, Austria
[2]University of Hagen, Hagen, Germany

Synonyms

WS-discovery; DISCO

Definition

The term discovery, as far as Web services (WS) are concerned, refers to the process of finding WS that match certain computational needs and quality requirements of service users or their software agents. More technically speaking, WS discovery mechanisms take a specification of certain functional or non-functional criteria characterizing a service and try to locate machine-readable descriptions of Web services that meet the search criteria. The services found may have been previously unknown to the requester.

Historical Background

Since Web services were introduced in the early 1990's, service-oriented architectures had to deal with the discovery problem, and it still persists. Initially, the possibilities to describe Web services properly were limited. XML remote procedure calls (XML-RPCs), which were used early on to connect WS, offered no proper way of describing a Web service's capabilities and therefore forced system designers to find other ways to publish this information. These early services were typically used to achieve a platform independent communication between remote peers, nothing more. This requirement was met with an XML-structured messaging protocol that evolved later to SOAP, a standard protocol of today's Web technology. For the act of discovering services, the actually used protocol made no difference. Services description files were propagated mostly by artificial means, by sending the file per e-mail for instance. In some cases, the developer of the Web service also worked on the client-side implementation.

A proper service description mechanism was only introduced when application developers realized that Web service technology had to be leveraged to a level that obviated the need of service consumer and provider to interact closely with each other prior to using a service. With the definition of the Web service description language (WSDL), it was finally possible to describe the interface of a WS in a standardized manner. The discovery problem, however, still persisted because no means existed to publish a service description in a widely known index or registry, once the implementation on the provider side was completed.

To overcome this barrier, a first draft of the Universal Description, Discovery and Integration, short *UDDI*, standard was released in 1999. UDDI was designed as a vendor-independent specification of an XML-based registry for service descriptions maintained in the form of WSDL documents. The standard was completed in 2002. Its purpose was to enable service providers and consumers to find each other if service request and service offer matched and to provide information about message formats and protocols accepted by a Web service. Apart from defining data models and registry structure, UDDI was also designed to offer simple search capabilities to help service consumers find Web services. Thus UDDI contributed to solve the discovery issue.

As a WSDL description of a Web service interface just lists the operations the service may perform and the messages it accepts and produces, the discovery mechanism of UDDI was constrained to match functionality only. If several candidate services could be found, the service consumer was unable to distinguish between them. Therefore people felt the need to be able to express semantic properties or quality aspects of requested services as well. But search mechanisms taking into account semantics and quality-of-service properties require richer knowledge about a registered Web service than WSDL can capture.

To complement the expressiveness of WSDL and facilitate service discovery, DAML-S, an ontology language for Web services, was proposed to associate computer-readable semantic information with service descriptions. Semantic service descriptions are seen as a potential enabler to enhance automated service discovery and matchmaking in various service oriented architectures. For the time being, however, services widely used in practice lack semantic information because no easy to use or even automated method to attach semantic information to service descriptions exists.

Foundations

The service discovery process encompasses several steps. Each step involves specific problems that have to be solved independently. The following list will discuss these steps in ascending order, beginning with the most generic step.

Enabling Discovery

At a first glance, the act of discovering a service description matching a set of terms characterizing a service, resembles a search processes for Web pages. Well-known search engines like Google or Live utilize a crawling mechanism to retrieve links to Web documents and create a index that can be searched effectively as users enter search terms. A crawler just analyzes a given Web page for hyperlinks and grinds through the tree structure generated by such hyperlinks to find other Web documents. For Web services, or more precisely Web service descriptions, the case is similar except for one major difference: WSDL files do not contain links to other services. Approaches to write crawlers that search web pages for possibly published service descriptions will produce very poor results, in general [4]. UDDI registries are just designed to eliminate the need for crawlers or alike. As a matter of fact, open UDDI registries for public Web services are increasingly loosing importance. Especially after the two largest UDDI registries from IBM and Microsoft were shut down in 2005, the vision of public services suffered immensely. Suddenly the starting point to find a public Web service was lost, leaving the possibility to query common Web search engines for Web services as the only alternative. There are, of course, some other registries but they usually do not implement the UDDi specification, which suggests that UDDI may not be an optimal solution for public service registries.

In a corporate environment, however, the initial discovery step is not a real issue. Web service descriptions can easily be published on an internal Web page or in a UDDI registry and are, therefore, easily accessible from within the institution.

Searching in Service Registries

Assuming that a comprehensive collection of service descriptions has already been established, the question is how to retrieve the closest match to a user query in an efficient manner.

As the title suggests, searching is usually performed by humans and not by software automatically. The challenge is how to create an index of services such that the addition and retrieval of service descriptions can be achieved accurately and fast. Common information retrieval methods are often used for this purpose, ranging from the vector space model for indexing and searching large repositories to graph theoretical approaches for fast processing of a rich data collection.

More or less every registry-based solution encompasses such a facility. Especially UDDI registries often come with a rudimentary interface to query the database for contained services. Unfortunately, the general structure of UDDI with its tModel component-layout, which serves as a container to store detailed service information, complicates data retrieval. Furthermore, most UDDI entries do not maintain a complete service description but include links to such descriptions kept elsewhere. But these links could be broken or inaccessible at search time. As a result, UDDI queries are usually processed on the business data related to a service and not the service description itself. This fact alone limits the usability of UDDI-based search mechanisms enormously or more precisely: it leaves most of the index quality in the hand of the users.

This area on the other hand is heavily investigated throughout the research community and several approaches have been presented that aim at improving search capabilities on service collections. Those approaches are mostly designed to handle natural language queries like “USA weather service” and are supposed to provide a user interface for various registry implementations.

Querying Repositories

A more detailed form of search in service descriptions is entitled as querying. Unlike direct search, in which a user simply provides a set of search terms, queries are formal expressions using some sort of query language. In the case of relational databases, SQL is typically used as a query language. Through a query expression it is possible to search for a specific service signature in a set of service descriptions. Assume, for example, that a user wants to find a weather service and can provide three bits of information: country, zip_code, and the desired scale for presenting the temperature. Assume further that the user wants to express certain quality requirements. Then, a query expression in a language alike SQL might read as follows:

SELECT description FROM services s WHERE

s.input.COMPOSEDOF(country AND zip_code AND useCelsiusScale)

AND s.response_time < 200ms AND s.downtime < 1%

This example also reveals a weakness of service descriptions as their signatures are usually not specified using exact type information such as *city* or *country* but rather basic types like string, integer etc. are used. Hence, it seems more appropriate to search for signatures in terms of basic data types only. But this would likely result in mismatches. Re-considering the query above, a corresponding signature using the basic types [string, integer, boolean] can easily be met by other services. There is no way to distinguish positive matches from unwanted ones without additional information or a richer index. These problems are addressed by introducing semantics and domain knowledge. For the values of response_time and downtime in this example, there already exist approaches that constantly monitor and invoke the services in a registry to create a statistical profile of the quality of such a service (*QoS*).

These approaches require a more sophisticated registry type than UDDI represents and are therefore mostly implemented in research prototypes.

Domain-Specific Knowledge in Service Descriptions

Another requirement that has to be met by more powerful discovery mechanisms is domain-specific knowledge about a service. To take on the sample above, a discovery mechanism able to match the terms city, zip_code and temperature with the semantic categories location and weather would select just the intersection of services dealing with location and temperature. Although domain information is semantic information in certain respects, it does not mean that the information has to be provided upon service registration. Certain approaches exist that are able to group service repositories according to their most probable domain. The grouping can, for instance, be achieved by using statistical cluster analysis and discover strongly related service descriptions.

On the other hand, domain-knowledge can also be gained by letting the service provider add this information. In practice however, it proved to be problematic to let users define semantic information for a service. Once, this is due to the fact that a certain amount of domain knowledge is needed by the programmer of the Web service but mostly because the categorization assigned by indexers cannot be validated and could therefore be incorrect. This field, just like the following, is still heavily investigated, e.g., under the heading "faceted search." It addresses a broad spectrum of issues but also bears a high potential for innovation.

Semantic Annotations

The next logical step towards enhancing service discovery is a complete semantic description of a Web service. A multitude of approaches exist, in which semantic annotations are used to define the concepts behind operations of Web services and their input and output messages. Those approaches use the widely known resource description framework (*RDF*) to add custom-designed semantic markup to service descriptions. A good example for such a semantic markup language is called DAML-S. It is a DAML-based Web service ontology, which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form. An example for such a markup is shown in Listing 1. This very short listing basically shows a Web service called `BookstoreBuy` and defines it as a process and therefore as a subclass of the class Process in the corresponding ontology. The second part shows an input to `BookstoreBuy` which is it is a subproperty of the property input of Process, from the process model. With this example, some of the limitations for semantic annotations become more obvious. First of all, the creator of the Web service is required to have additional knowledge about semantic annotations and ontologies. Furthermore, the appended information causes an additional amount of work. Secondly, the ontology used defines sharp boundaries for the level of detail that can be reached through the usage of such annotations. In addition, the problem of misused or erroneous annotations mentioned in the previous step persists. Put together, semantic Web services are seen as a technology with the potential to facilitate more powerful service discovery and machine-readable descriptions. Practical experience, however, showed that an exploitation of semantic information is difficult and still leaves room for further improvements.

Quality-of-Service Properties

The consideration of quality-of-service (QoS) properties in discovery attempts requires the definition of scales of measurements and metrics to qualify the properties per domain. The scales can be of different

```
<rdfs:Class rdf:ID="BookstoreBuy">
  <rdfs:subClassOf rdf:resource=
        "http://www.www.daml.org/services/daml-s/2001/05/Process#Process"/>
</rdfs:Class>

<rdf:Property rdf:ID="bookName">
  <rdfs:subPropertyOf rdf:resource=
        "http://www.www.daml.org/services/daml-s/2001/05/Process#input"/>
   <rdfs:domain rdf:resource="#BookstoreBuy"/>
   <rdfs:range rdf:resource="#rdfs:Literal"/>
</rdf:Property>
```

Discovery. Listing 1. DAML-S Sample.

kinds including nominal, ordinal, interval or ratio. They are used to assign appropriate QoS property values to a service. Here, a service provider has the choice to associate precise values or just value ranges with service property descriptions. The metrics are needed to rank services that match the functional and semantic requirement of a search according their degree of fulfillment of required QoS properties.

These issues and related modifications to service discovery schemes are still subject to a number of research projects. Wang et al. [6], for example, proposes a services discovery approach in which functional, semantic and QoS requirements are taken into account by iteratively applying related filters.

Key Applications

Some of the concepts presented above, especially the first, more general layers are already used in real world implementations. Service registries, especially those not strictly conforming to the UDDI specification are the main area of application for innovative discovery mechanisms. Search and matchmaking on the other hand is particularly required by IDEs. Especially service composition environments enormously benefit from fast and exact discovery mechanisms. Finding substitutes and alternatives for services in compositions is an important topic in service-oriented architectures.

Future Directions

Future directions show considerable tendencies towards the semantic web to enhance service discovery for Web services. This fact, however, creates a diversion among researchers on this particular area. Some argue that semantic descriptions are too complicated to be of any practical use, while others argue that they are the only way to leverage Web service discovery and search to a point where they can be processed automatically. Both views are valid and which direction proves to be the most promising will be decided by the work that is yet to come.

Some recent works have proposed a recommendation service and suggest to apply it to collaborative web service discovery. Experiments with such solutions are underway in research labs.

URL to Code

A Vector-space based search engine for Web services including a clustering algorithm: http://copenhagen.vitalab.tuwien.ac.at/VSMWeb/

A Web service registry with invocation capabilities: http://www.xmethods.net

Another Web service search engine: http://www.esynaps.com/search/default.aspx

WSBen, a Web service discovery and composition benchmark developed at Penn State: http://pike.psu.edu/sw/wsben/

Cross-references

► Business Process Management
► Publish/Subscribe
► Service Oriented Architecture
► Web Services

Recommended Reading

1. Benatallah B., Hacid M.-S., Leger A., Rey C., and Toumani F. On automating web services discovery. VLDB J., 14(1):84–96, 2005.
2. Bussler C., Fensel D., and Maedche A. A conceptual architecture for semantic web enabled web services. ACM SIGMOD Rec., 2002.
3. Kokash N., Birukou A., and D'Andrea V. Web service discovery based on past user experience. In Proc. Int. Conf. on Business Information Systems, pp. 95–107.
4. Platzer C. and Dustdar S. A vector space search engine for Web services. In Proc. 3rd European IEEE Conf. on Web Services, 2005.
5. Rosenberg F., Platzer C., and Dustdar S. Bootstrapping performance and dependability attributes of web services. In Proc. IEEE Int. Conf. on Web Services, 2006, pp. 205–212.
6. Wang X., Vitvar T., Kerrigan M., and Toma I. Synthetical evaluation of multiple qualities for service selection. In Proc. Fourth Int. Conf. on Service Oriented Computing. Springer, 2006, pp. 1–12.

Discrete Wavelet Transform and Wavelet Synopses

MINOS GAROFALAKIS
Technical University of Crete, Chania, Greece

Definition

Wavelets are a useful mathematical tool for hierarchically decomposing functions in ways that are both efficient and theoretically sound. Broadly speaking, the wavelet transform of a function consists of a coarse overall approximation together with detail coefficients that influence the function at various scales. The wavelet transform has a long history of successful applications in signal and image processing [11,12]. Several recent studies have also demonstrated the effectiveness of the wavelet transform (and *Haar wavelets*, in particular) as a tool for approximate query processing over massive relational tables [2,7,8] and continuous data streams [3,9]. Briefly, the idea is to apply wavelet transform to the input relation to obtain a compact data synopsis that comprises a select small collection of *wavelet coefficients*. The excellent energy compaction and de-correlation properties of the wavelet transform allow for concise and effective approximate representations that exploit the structure of the data. Furthermore, wavelet transforms can generally be computed in linear time, thus allowing for very efficient algorithms.

Historical Background

A growing number of database applications require on-line, interactive access to very large volumes of data to perform a variety of data-analysis tasks. As an example, large Internet Service Providers (ISPs) typically collect and store terabytes of detailed usage information (NetFlow/SNMP flow statistics, packet-header information, etc.) from the underlying network to satisfy the requirements of various network-management tasks, including billing, fraud/anomaly detection, and strategic planning. This data gives rise to massive, multi-dimensional *relational data tables* typically stored and queried/analyzed using commercial database engines (such as, Oracle, SQL Server, DB2). To handle the huge data volumes, high query complexities, and interactive response-time requirements characterizing these modern data-analysis applications, the idea of effective, easy-to-compute *approximate query answers* over precomputed, compact *data synopses* has recently emerged as a viable solution. Due to the exploratory nature of most target applications, there are a number of scenarios in which a (reasonably-accurate) fast approximate answer over a small-footprint summary of the database is actually preferable over an exact answer that takes hours or days to compute. For example, during a "drill-down" query sequence in ad-hoc data mining, initial queries in the sequence frequently have the sole purpose of determining the truly interesting queries and regions of the database. Providing fast approximate answers to these initial queries gives users the ability to focus their explorations quickly and effectively, without consuming inordinate amounts of valuable system resources.

The key behind such approximate techniques for dealing with massive data sets lies in the use of appropriate *data-reduction techniques* for constructing compact synopses that can accurately approximate the important features of the underlying data distribution. The *Haar wavelet decomposition* is one such technique with deep roots in the fields of signal and image processing, that has recently found its way into database applications as an important approximate query processing tool.

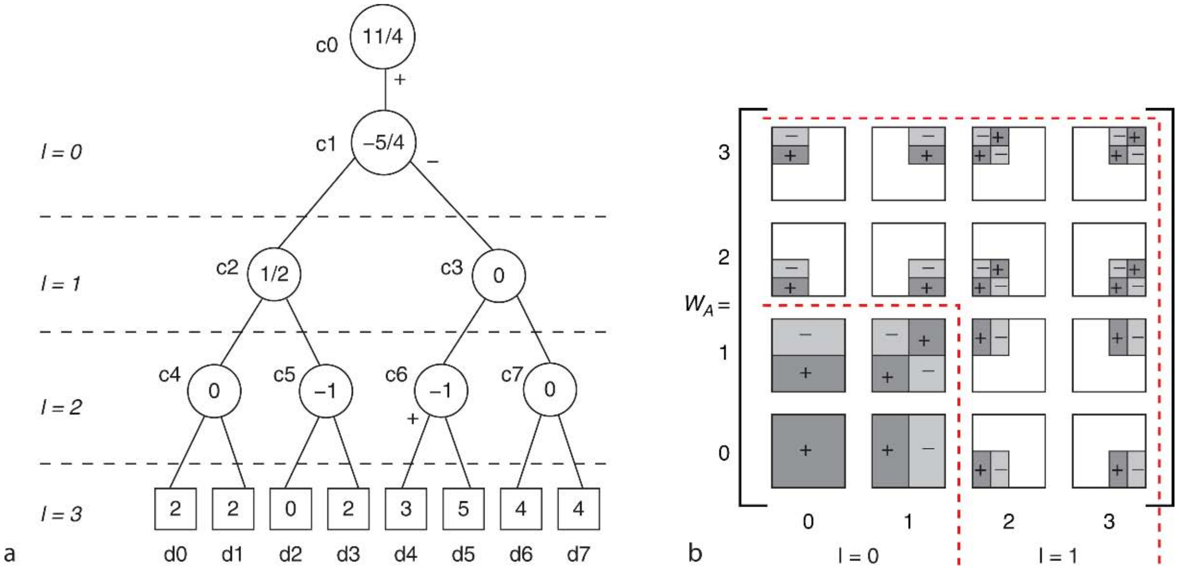
Foundations

Haar Wavelet Basics

Haar wavelets are conceptually simple, easy to compute, and have been found to perform well in practice for a variety of applications, ranging from image editing and querying to database selectivity estimation tasks. Consider a one-dimensional data vector A containing the $N=8$ data values $A=[2,2,0,2,3,5,4,4]$. The Haar wavelet transform of A can be computed as follows. The values are first averaged together pairwise to get a new "lower-resolution" representation of the data with the following average values $[2,1,4,4]$. To restore the original values of the data array, additional *detail coefficients* must be stored to capture the information lost due to this averaging. In Haar wavelets, these detail coefficients are simply the differences of the (second of the) averaged values from the computed pairwise average, that is, $[2-2,1-2,4-5,4-4]=[0,-1,-1,0]$. No information has been lost in this process – it is simple to reconstruct the eight values of the original data array from the lower-resolution array containing the four averages and the four detail coefficients. Recursively applying the above pairwise averaging and differencing process on the lower-resolution array containing the averages, gives the following full transform:

The *wavelet transform* W_A of A is the single coefficient representing the overall average of the data values followed by the detail coefficients in the order of increasing resolution, i.e., $W_A=[11/4,-5/4,1/2,0,0,-1,-1,0]$ (each entry is called a *wavelet coefficient*). For vectors containing similar values, most of the detail coefficients tend to be very small; thus, eliminating them from the wavelet transform (i.e., treating them as zeros) introduces only small errors when reconstructing the original data, resulting in a very effective form of lossy data compression [12].

A helpful tool for conceptualizing the recursive Haar wavelet transform process is the *error tree* structure (shown in Fig. 1a for the example array A). Each



Discrete Wavelet Transform and Wavelet Synopses. Figure 1. (a) Error-tree structure for the example data array A (N= 8). (b) Support regions and signs for the 16 nonstandard two-dimensional Haar basis functions.

internal node c_i (i=0,...,7) is associated with a wavelet coefficient value, and each leaf d_i (i=0,...,7) is associated with a value in the original data array; in both cases, the index i denotes the positions in the (data or wavelet transform) array. For instance, c_0 corresponds to the overall average of A. The resolution levels l for the coefficients (corresponding to levels in the tree) are also depicted.

Resolution	Averages	Detail coefficients
3	[2, 2, 0, 2, 3, 5, 4, 4]	–
2	[2, 1, 4, 4]	[0, −1, −1, 0]
1	[3/2, 4]	[1/2, 0]
0	[11/4]	[−5/4]

Given an error tree T and an internal node t of T, $t \neq c_0$, `leftleaves`(t) (`rightleaves`(t)) denotes the set of leaf (i.e., data) nodes in the subtree rooted at t's left (resp., right) child. Also, given any (internal or leaf) node u, *path*(u) is the set of all (internal) nodes in T that are proper ancestors of u (i.e., the nodes on the path from u to the root of T, including the root but not u) with non-zero coefficients. Finally, for any two leaf nodes d_l and d_h, $d(l : h)$ denotes the range sum $\sum_{i=l}^{h} d_i$. Using the error tree representation T, the following important reconstruction properties of the Haar wavelet transform can be outlined.

- (P1) The reconstruction of any data value d_i depends only on the values of the nodes in *path*(d_i). More specifically, $d_i = \sum_{c_j \in path(d_i)} \delta_{ij} \cdot c_j$, where δ_{ij}=+1 if d_i ∈`leftleaves`(c_j) or j=0, and δ_{ij}=−1 otherwise; for example, $d_4 = c_0 - c_1 + c_6 = \frac{11}{4} - (-\frac{5}{4}) + (-1) = 3$.
- (P2) An internal node c_j contributes to the range sum $d(l : h)$ *only if* $c_j \in path(d_l) \cup path(d_h)$. More specifically, $d(l : h) = \sum_{c_j \in \text{path}(d_l) \cup \text{path}(d_h)} x_j$, where

$$x_j = \begin{cases} (h-l+1) \cdot c_j, \text{if } j = 0 \\ (|\texttt{leftleaves}(c_j, l:h)| - \texttt{rightleaves}(c_j, l:h)|) \cdot c_j, \text{otherwise.} \end{cases}$$

where `leftleaves`(c_j,l : h)=`leftleaves`(c_j) ∩{ d_l, d_{l+1},..., d_h} (i.e., the intersection of `leftleaves`(c_j) with the summation range) and `rightleaves`(c_j,l : h) is defined similarly. (Clearly, coefficients whose subtree is completely contained within the summation range have a net contribution of zero, and can be safely ignored.) For example, $d(2 : 6) = 5c_0 + (2-3) c_1 - 2c_2 = 5 \times \frac{11}{4} - (-\frac{5}{4}) - 1 = 14$.

Thus, reconstructing a single data value involves summing at most log N+1 coefficients and reconstructing a range sum involves summing at most 2 log N+1 coefficients, regardless of the width of the range. The *support region* for a coefficient c_i is defined as the set of (contiguous) data values that c_i is used to reconstruct.

The Haar wavelet transform can be naturally extended to *multi-dimensional* data arrays using two distinct methods, namely the *standard* and *nonstandard* Haar transform [12]. As in the one-dimensional case, the Haar transform of a d-dimensional data array A results in a d-dimensional wavelet-coefficient array W_A with the same dimension ranges and number of entries. Consider a d-dimensional wavelet coefficient W in the (standard or nonstandard) wavelet-coefficient array W_A. W contributes to the reconstruction of a d-dimensional rectangular region of cells in the original data array A (i.e., W's support region). Further, the sign of W's contribution ($+W$ or $-W$) can vary along the quadrants of W's support region in A.

As an example, Fig. 1b depicts the support regions and signs of the sixteen nonstandard, two-dimensional Haar coefficients in the corresponding locations of a 4×4 wavelet-coefficient array W_A. The blank areas for each coefficient correspond to regions of A whose reconstruction is independent of the coefficient, i.e., the coefficient's contribution is 0. Thus, $W_A[0,0]$ is the overall average that contributes positively (i.e.,"$+ W_A[0,0]$") to the reconstruction of all values in A, whereas $W_A[3,3]$ is a detail coefficient that contributes (with the signs shown) only to values in A's upper right quadrant. Each data cell in A can be accurately reconstructed by adding up the contributions (with the appropriate signs) of those coefficients whose support regions include the cell. Error-tree structures for d-dimensional Haar coefficients are essentially *d-dimensional quadtrees*, ϵwhere each internal node t corresponds to a *set* of (at most) 2^d-1 Haar coefficients, and has 2^d children corresponding to the quadrants of the (common) support region of all coefficients in t; furthermore, properties (P1) and (P2) can also be naturally extended to the multi-dimensional case [2,7,8].

Data Reduction and Approximate Query Processing

Consider a relational table R with d data attributes X_1, X_2,...,X_d. The information in R can be represented as a d-dimensional array A_R, whose j^{th} dimension is indexed by the values of attribute X_j and whose cells contain the count of tuples in R having the corresponding combination of attribute values. A_R is essentially the *joint frequency distribution* of all the data attributes of R. Given a limited amount of storage for building a *wavelet synopsis* of an input relation R, a thresholding procedure retains a certain number $B << N$ of the coefficients in the wavelet transform of A_R as a highly-compressed approximate representation of the original data (the remaining coefficients are implicitly set to 0). (The full details as well as efficient transform algorithms can be found in [2,13].) The goal of *coefficient thresholding* is to determine the "best" subset of B coefficients to retain, so that some overall error measure in the approximation is minimized – the next subsection discusses different thresholding strategies proposed in the database literature.

The construction of wavelet synopses typically takes place during the statistics collection process, whose goal is to create concise statistical approximations for the value distributions of either individual attributes or combinations of attributes in the relations of a Database Management System (DBMS). Once created, a wavelet synopsis is typically stored (as a collection of B wavelet coefficients) as part of the *DBMS-catalog information*, and can be exploited for several different purposes. The primary (and, more conventional) use of such summaries is as a tool for enabling effective (compile-time) estimates of the result sizes of relational operators for the purpose of *cost-based query optimization.* (Accurate estimates of such result sizes play a critical role in choosing an effective physical execution plan for an input SQL query.) For instance, estimating the number of data tuples that satisfy a range-predicate selection like $l \leq X \leq h$ is equivalent to estimating the range summation $f(l:\ h)=\sum_{i=l}^{h} f_i$, where f is the frequency distribution array for attribute X. As mentioned earlier, given a B-coefficient synopsis of the f array, computing $f(l:\ h)$ only involves retained coefficients in $path(f_l) \cup path(f_h)$ and, thus, can be estimated by summing only $min\{B,\ 2\log N+1\}$ synopsis coefficients [13]. A B-coefficient wavelet synopsis can also be easily expanded (in $O(B)$ time) into an $O(B)$-bucket *histogram* (i.e., piecewise-constant) approximation of the underlying data distribution with several possible uses (e.g., as a data visualization/approximation tool).

More generally, wavelet synopses can enable very fast and accurate approximate query answers [6] during interactive data-exploration sessions. As demonstrated by Chakrabarti et al. [2], an approximate query processing algebra (which includes all conventional aggregate and non-aggregate SQL operators, such as `select`, `project`, `join`, `sum`, and `average`) can operate *directly over the wavelet synopses of relations*, while guaranteeing the correct relational operator semantics. Query processing algorithms for these operators work *entirely* in the wavelet-coefficient domain. This allows for extremely fast response times, since the approximate query execution engine can do the bulk of its processing over compact wavelet synopses, essentially postponing the (expensive) expansion step into relational tuples until the end-result of the query.

Conventional and Advanced Wavelet Thresholding Schemes

Recall that coefficient thresholding achieves data reduction by retaining $B << N$ of the coefficients in the wavelet transform of A_R as a highly-compressed, lossy representation of the original relational data. The goal, of course, is to minimize the amount of "loss" quantified through some overall approximation error metric. Conventional wavelet thresholding (the method of choice for most studies on wavelet-based data reduction) greedily retains the *B largest Haar-wavelet coefficients in absolute value* after a simple normalization step (that divides each coefficient value at resolution level l by $\sqrt{2^l}$). It is a well-known fact that this thresholding method is in fact *provably optimal* with respect to minimizing the overall root-mean-squared error (i.e., L_2*-norm error*) in the data compression [12]. More formally, letting $\hat{d}_i$ denote the (approximate) reconstructed data value for cell i, retaining the B largest normalized coefficients implies that the resulting synopsis minimizes $L_2(\hat{d}) = \sqrt{\sum_i (\hat{d}_i - d_i)^2}$ (for the given amount of space B).

Conventional wavelet synopses optimized for overall L_2 error may not always be the best choice for approximate query processing systems. The quality of the approximate answers such synopses provide can vary widely, and users have no way of knowing the accuracy of any particular answer. Even for the simplest case of approximating a value in the original data set, the absolute and relative errors can show wide variation. Consider the example depicted in Table 1. The first line shows the 16 original data values (the exact answer), whereas the second line shows the 16 approximate answers returned when using conventional wavelet synopses and storing eight coefficients. Although the first half of the values is basically a mirror image of the second half, all the approximate answers for the first half are 65, whereas all the approximate answers for the second half are exact! Similar data values have widely different approximations, e.g., 30 and 31 have approximations 30 and 65, respectively. The approximate answers make the first half appear as a uniform distribution, with widely different values, e.g., 3 and 127, having the same approximate answer 65. Moreover, the results do not improve when one considers the presumably easier problem of approximating the sum over a range of values: for *all possible* ranges within the first half involving $x = 2$ to 7 of the values, the approximate answer will be $65 \cdot x$, while the actual answers vary widely. For example, for both the range d_0 to d_2 and the range d_3 to d_5, the approximate answer is 195, while the actual answer is 285 and 93, respectively. On the other hand, *exact* answers are provided for all possible ranges within the second half.

The simple example above illustrates that conventional wavelet synopses suffer from several important problems, including the introduction of severe bias in the data reconstruction and wide variance in the quality of the data approximation, as well as the lack of nontrivial guarantees for individual approximate answers. To address these shortcomings, recent work has proposed novel thresholding schemes for building wavelet synopses that try to minimize different approximation-error metrics, such as the *maximum relative error* (with an appropriate *sanity bound* s) in the approximation of individual data values based on the synopsis; that is, minimize $\max_i \left\{ \frac{|\hat{d}_i - d_i|}{\max\{|d_i|, s\}} \right\}$. Such relative-error metrics are arguably the most important quality measures for approximate query answers. (The role of the sanity bound is to ensure that relative-error numbers are not unduly dominated by small data values.)

More specifically, Garofalakis and Gibbons [7] introduce *probabilistic* thresholding schemes based on ideas from randomized rounding, that probabilistically

Discrete Wavelet Transform and Wavelet Synopses. Table 1. Errors with conventional wavelet synopses

Original data values	127	71	87	31	59	3	43	99	100	42	0	58	30	88	72	130
Wavelet answers	65	65	65	65	65	65	65	65	100	42	0	58	30	88	72	130

round coefficients either up to a larger rounding value (to be retained in the synopsis) or down to zero. Intuitively, their probabilistic schemes assign each non-zero coefficient *fractional storage* $y \in [0,1]$ equal to its retention probability, and then flip independent, appropriately-biased coins to construct the synopsis. Their thresholding algorithms are based on *Dynamic-Programming (DP)* formulations that explicitly minimize appropriate probabilistic metrics (such as the maximum normalized standard error or the maximum normalized bias) in the randomized synopsis construction; these formulations are then combined with a *quantization* of the potential fractional-storage allotments to give combinatorial techniques [7].

In more recent work, Garofalakis and Kumar [8] show that the pitfalls of randomization can be avoided by introducing efficient schemes for *deterministic* wavelet thresholding with the objective of optimizing a *general class of error metrics* (e.g., maximum or mean relative error). Their optimal and approximate thresholding algorithms are based on novel DP techniques that take advantage of the Haar transform error-tree structure. In a nutshell, their DP algorithms tabulate the optimal solution for the subtree rooted at each error-tree node c_j *given the error contribution that "enters" that subtree through the choices made at all ancestor nodes of c_j in the tree* (i.e., the choice of coefficients on *path*(c_j)). The key observation here is that, since the depth of the error tree is $O(\log N)$, all such possible selections can be tabulated while still keeping the running-time of the thresholding algorithm in the low-polynomial range. This turns out to be a fairly powerful idea for wavelet synopsis construction that can handle a broad, natural class of *distributive error metrics* (which includes several useful error measures for approximate query answers, such as maximum or mean weighted relative error and weighted L_p-norm error) [8].

The above wavelet thresholding algorithms for non-L_2 error metrics consider only the *restricted* version of the problem, where the algorithm is forced to select values for the synopsis from the standard Haar coefficient values. As observed by Guha and Harb [10], such a restriction makes little sense when optimizing for non-L_2 error, and can, in fact, lead to sub-optimal synopses. Their work considers *unrestricted* Haar wavelets, where the values retained in the synopsis are specifically chosen to optimize a general (weighted) L_p error metric. Their proposed thresholding schemes rely on a DP over the error tree (similar to that in [8]) that *also iterates over the range of possible coefficient values for each node*. To keep time and space complexities manageable, techniques for bounding these coefficient-value ranges are also discussed [10].

Extended and Streaming Wavelet Synopses

Complex tabular data sets *with multiple measures* (multiple numeric entries for each table cell) introduce interesting challenges for wavelet-based data reduction. Such massive, multi-measure tables arise naturally in several application domains, including OLAP (On-Line Analytical Processing) environments and time-series analysis/correlation systems. As an example, a corporate sales database may tabulate, for each available product (i) the number of items sold, (ii) revenue and profit numbers for the product, and (iii) costs associated with the product, such as shipping and storage costs. Similarly, real-life applications that monitor continuous time-series typically have to deal with several readings (measures) that evolve over time; for example, a network-traffic monitoring system takes readings on each time-tick from a number of distinct elements, such as routers and switches, in the underlying network and typically several measures of interest need to be monitored (e.g., input/output traffic numbers for each router or switch interface) even for a fixed network element. Deligiannakis et al. [4] show that obvious approaches for building wavelet synopses for such multi-measure data can lead to poor synopsis-storage utilization and suboptimal solutions even in very simple cases. Instead, their proposed solution is based on (i) *extended wavelet coefficients*, the first adaptive, efficient storage scheme for multi-measure wavelet coefficients; and, (ii) novel algorithms for selecting the optimal subset of extended coefficients to retain for minimizing the weighted sum of L_2 errors across all measures under a given storage constraint.

Traditional database systems and approximation techniques are typically based on the ability to make multiple passes over *persistent data sets*, that are stored reliably in stable storage. For several emerging application domains, however, data arrives at high rates and needs to be processed on a continuous (24 × 7) basis, without the benefit of several passes over a static, persistent data image. Such *continuous data streams* arise naturally, for example, in the network installations of large Telecom and Internet service providers where detailed usage information (Call-Detail-Records (CDRs), SNMP/RMON packet-flow data, etc.) from different parts of the underlying network needs to be continuously collected and monitored for interesting trends and phenomena (e.g., fraud or Denial-of-Service attacks). Efficiently tracking an accurate wavelet synopsis over such massive streaming data, using only small space and time (per streaming update), poses a host of new challenges. Recently-proposed solutions [3, 9] rely on maintaining small-space, *pseudo-random AMS sketches* (essentially, random linear projections) over the input data stream [1]. These sketches can then be queried to efficiently recover the topmost wavelet coefficients of the underlying data distribution within provable error guarantees [3].

Key Applications

Wavelet synopses are a general data-reduction tool with several important applications, including statistics for query optimization, lossy data compression, OLAP cube summarization, and interactive data exploration, mining, and query processing.

Data Sets

Several publicly-available real-life data collections have been used in the experimental study of wavelet synopses (and other data-reduction methods); examples include the US Census Bureau data sets (http://www.census.gov/), the UCI KDD Archive (http://kdd.ics.uci.edu/), and the UW Earth Climate and Weather Data Archive (http://www-k12.atmos.washington.edu/k12/grayskies/).

Future Directions

The area of wavelet-based data reduction is still rife with interesting algorithmic questions, including, for instance (i) designing efficient methods for building wavelet synopses that optimize different error metrics under general streaming models (e.g., allowing both item insertions and deletions), and (ii) developing a sound foundation and appropriate summarization tools for approximate *set-valued* (i.e., non-aggregate) queries. Dealing with the *curse of dimensionality* that invariably haunts space-partitioning techniques (such as wavelets and histograms) is another big open issue; some initial ideas based on combining these techniques with statistical-correlation models appear in [5]. And, of course, from a systems perspective, the problem of incorporating wavelets and other approximate query processing tools in an industrial-strength database engine (that can, e.g., select and optimize the appropriate tools for each scenario) remains wide open.

Cross-references

- ▶ Approximate Query Processing
- ▶ Data Compression in Sensor Networks
- ▶ Data Reduction
- ▶ Data Sketch/Synopsis
- ▶ Synopsis Structure
- ▶ Wavelets on Streams

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. In Proc. 28th Annual ACM Symp. on Theory of Computing, 1996, pp. 20–29.
2. Chakrabarti K., Garofalakis M.N., Rastogi R., and Shim K. Approximate query processing using wavelets. VLDB J., 10(2-3): 199–223, September 2001.
3. Cormode G., Garofalakis M., and Sacharidis D. Fast approximate wavelet tracking on streams. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006.
4. Deligiannakis A., Garofalakis M., and Roussopoulos N. Extended wavelets for multiple measures. ACM Trans. Database Syst., 32(2), June 2007.
5. Deshpande A., Garofalakis M., and Rastogi R. Independence is good: dependency-based histogram synopses for high-dimensional data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001.
6. Garofalakis M. and Gibbons P.B. Approximate query processing: taming the terabytes. Tutorial in 27th International Conference on Very Large Data Bases, 2001.
7. Garofalakis M. and Gibbons P.B. Probabilistic wavelet synopses. ACM Trans. Database Syst., 29(1), March 2004.
8. Garofalakis M. and Kumar A. Wavelet synopses for general error metrics. ACM Trans. Database Syst., 30(4), December 2005.
9. Gilbert A.C., Kotidis Y., Muthukrishnan S., and Strauss M.J. One-pass wavelet decomposition of data streams. IEEE Trans. Knowl. Data Eng., 15(3): 541–554, May 2003.
10. Guha S. and Harb B. Wavelet synopsis for data streams: minimizing non-euclidean error. In Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2005, pp. 88–97.
11. Jawerth B. and Sweldens W. An overview of wavelet based multiresolution analyses. SIAM Rev., 36(3): 377–412, 1994.
12. Stollnitz E.J., DeRose T.D., and Salesin D.H. Wavelets for computer graphics – theory and applications. Morgan Kaufmann, San Francisco, CA, 1996.
13. Vitter J.S. and Wang M. Approximate computation of multidimensional aggregates of sparse data using wavelets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 193–204.

Discretionary Access Control

Gail-Joon Ahn
Arizona State University, Tempe, AZ, USA

Synonyms

DAC; Identity-based Access Control; etc.

Definition

Discretionary access control (DAC) provides for owner-controlled administration of access rights to objects. DAC, as the name implies, permits the granting and revocation of access permissions to be left to the discretion of the individual users. A DAC mechanism allows users to grant or revoke access to any of the objects under their control.

Historical Background

Trusted computer system evaluation criteria (TCSEC) published by the US Department of Defense, commonly known as the Orange Book, defined two important access control modes for information systems: discretionary access control (DAC) and mandatory access control (MAC). As the name implies, DAC allows the creators or owners of files to assign access rights. Also, a user (or subject) with discretionary access to information can pass that information on to another user (or subject). DAC has its genesis in the academic and research setting from which time-sharing systems emerged in the early 1970s.

Foundations

As defined in the TCSEC and commonly implemented, DAC policy permits system users (or subjects) to allow or disallow other users (or subjects) access to the objects under their control. The TCSEC DAC policy is defined as follows [1]:

A means of restricting access to objects based on the identity of subjects or groups, or both, to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject.

DAC is a means of restricting access to objects based on the identity of users or the groups to which they belong. The controls are discretionary in the sense that a user or subject given discretionary access to a resource is capable of passing that information on to another subject. To provide this discretionary control, DAC is based on the notion that individual users are "owners" of objects and therefore have complete discretion over who should be authorized to access the object and in which access mode [2]. Ownership is usually acquired as a consequence of creating the object [3]. In other words, DAC policies govern the access of users to the information on the basis of the user's identity and authorizations (or rules) that specify the access modes the user is allowed on the object. Each request of a user to access an object is checked against the specified authorizations. If there exists an authorization stating that the user can access the object in the specific mode, the access is granted, otherwise it is denied.

DAC mechanisms tend to be very flexible. The flexibility of discretionary policies makes them suitable for a variety of systems and applications. For these reasons, they have been widely used in a variety of implementations, especially in the commercial and industrial environments.

DAC policies based on explicitly specified authorization are said to be closed, in that the default decision of the reference monitor is denial. Similar policies, called open policies, could also be applied by specifying denials instead of permissions [4,5]. In this case, the access modes the user is forbidden on the object are specified. Each access request by a user is checked against the specified authorizations and granted only if no authorizations denying the access exist. The use of positive and negative authorizations can be combined, allowing the specification of both the accesses to be authorized as well as the accesses to be denied to the users. The combination of positive and negative authorizations can become enormously complicated. In addition, for many enterprises within industry and government, their users do not own the information to which they are allowed access as is claimed by DAC policies. For such organizations, the organization is the actual owner of system objects and it may not be appropriate to allow users to manipulate access rights on the objects.

However, even though DAC mechanisms are in wide commercial use today, they are known to be inherently weak since they do not provide real assurance on the flow of information in a system. Granting read access is transitive so nothing stops a user from copying the contents of other's file to an object that s/he controls. For example, a user who is able to read data can pass it to other users not authorized to read it without the cognizance of the owner. Therefore, it is easy to bypass the access restrictions stated through the authorizations.

The reason is that discretionary policies do not impose any restriction on the usage of information by a user once the user has got it. In other words, further dissemination of information is not governed.

In addition, access rights need to be assigned explicitly to users who need access. Programs executed by a user have the same access rights as the user who is invoking it. This means that the security of the database system depends on the applications that are being executed. That is, a security breach in an application can affect all the objects to which the user has access as well. This makes DAC mechanisms vulnerable to "Trojan Horse" attacks. Because programs inherit the identity of the users who are invoking them, a user may write a program for other user that performs some legitimate system activities, while at the same time reads the contents of other user's files and writes the contents of the files to a location that both users can access. The user may then move the contents of the files to a location not accessible to the other user [6].

In summary, DAC is very flexible and suitable for various applications but it has an inherent weakness that information can be copied from one object to another, so access to a copy is possible even if the owner of the original does not provide access to the original. Moreover, such copies can be propagated by Trojan Horse software without explicit cooperation of users who are allowed access to the original [7,8].

DAC in Relational Database

The SQL standard includes the access control facilities to support DAC features. The creator of a relation in an SQL database becomes an owner of the relation. The owner also has the fundamental ability to grant other users access to that relation. The access privileges recognized in SQL correspond explicitly to the CREATE, SELECT, INSERT, DELETE and UPDATE statements. There is also a REFERENCES privilege to control the foreign keys to a relation. SQL does not require direct privilege for a user to create a relation, unless the relation is defined to have a foreign key to another relation. For the latter case the user must have the REFERENCES privilege for the relation. To create a view a user should have the SELECT privilege on every relation mentioned in definition of the view. If a user has access privileges on these relations, corresponding privileges are obtained on the view as well.

In addition, the owner of a relation can grant one or more access privileges to another user. This can be done with or without the GRANT OPTION. If the owner grants, say, SELECT with the GRANT OPTION the user receiving this grant can further grant SELECT to other users. The general format of a grant operation in SQL is as follows.

GRANT	privileges
[ON	relation]
TO	users
[WITH	GRANT OPTION]

The GRANT command applies to base relations as well as views. The ON and WITH clauses denote that these are optional and may not be present in every GRANT command. INSERT, DELETE and SELECT privileges apply to the entire relation as a unit. INSERT and DELETE are operations on entire rows so this is appropriate. SELECT, however, allows users to select on all columns. Selection on a subset of the columns can be achieved by defining a suitable view, and granting SELECT on the view. The UPDATE privilege in general applies to a subset of the columns. For example, a user could be granted the privilege to update the ADDRESS but not the GRADE of an STUDENT.

Also, the REVOKE statement is necessary to take away a privilege that has been granted by a GRANT. It is often required that revocation should cascade. In a cascading revoke, not only is the revoked privilege taken away, but also all GRANTs based on the revoked privilege are accordingly revoked. For example say that user Alice grants Bob SELECT on relation R with the GRANT OPTION. Furthermore, Bob subsequently grants Chris SELECT on R. Now suppose Alice revokes SELECT on R from Bob. The SELECT on R privilege is taken away not only from Bob, but also from Chris. The precise methods of a cascading revoke is somewhat complicated. Suppose Bob had received the SELECT on R privilege not only from Alice, but also from David before Bob granted the SELECT to Chris. In this case Alice's revocation of the SELECT from R privilege from Bob will not cause either Bob or Chris to loose this privilege. This is because the GRANT from David remains legitimate.

Cascading revocation is not always desirable. A user's privileges to a given table are often revoked because the user's job functions and responsibilities have changed. Thus the Head of a Division may move on to a different assignment. His/her privileges to that Division's data need to be revoked. However, a cascading revoke may cause lots of employees of that Division to loose their privileges. These privileges should be reassigned to maintain the business continuity.

Several database products take the approach that a database is always created with a single user, usually called the Database Administrator (DBA). The DBA essentially has all privileges with respect to this database. The DBA is also responsible for enrolling users and creating relations. Some systems recognize a special privilege which can be granted to other users at the initial DBA's discretion and allows these users to successfully act as the DBA [9,10].

Key Applications

Database systems, operating systems, and owner-centered web applications.

Future Directions

There are many subtle issues in DAC such as multi-step grant, cascading revocation, and so on. All these subtleties of DAC are still being discussed, debated and refined in the literature. The driving principle of DAC is ownership and such an owner-based access control can be applied to preserve privacy attributes in database systems. Nevertheless DAC has the inherent weakness, DAC's flexibility and suitability are needed to be further studied to support emerging critical applications.

Cross-references

- ▶ Access Control
- ▶ Access Control Administration Policies
- ▶ Access Control Policy Languages
- ▶ Mandatory Access Control
- ▶ Role Based Access Control

Recommended Reading

1. Bertino E., Samarati P., and Jajodia S. Authorizations in relational database management systems. In Proc. 1st ACM Conf. on Computer and Communications Security, 1993, pp. 130–139.
2. Bishop M. Computer Security: Art and Science. Addison-Wesley, Reading, MA, 2003.
3. Castano S., Fugini M.G., Martella G., and Samarati P. Database Security. Addison Wesley, Reading, MA, 1994.
4. Fagin R. On an authorization mechanism. ACM Trans. Database Syst., 3(3):310–319, 1978.
5. Ferraiolo D.F., Gilbert D.M., and Lynch N. An examination of federal and commercial access control policy needs. In Proc. NIST–NCSC National Computer Security Conference, 1993, pp. 107–116.
6. Graham G.S. and Denning P.J. Protection: principles and practice. In Proc. AFIPS Spring Joint Computer Conference. 40:417–429, 1972.
7. Griffiths P.P. and Wade B.W. An authorization mechanism for a relational database system. ACM Trans. Database Syst., 1(3):242–255, 1976.
8. Lampson B.W. Protection. In Proc. 5th Princeton Symp. on Information Science and Systems, 1971, pp. 437–443. Reprinted in ACM Operat. Syst. Rev., 8(1):18–24, 1974.
9. Rabitti F., Bertino E., Kim W., and Woelk D. A model of authorization for next-generation database systems. ACM Trans. Database Syst., 16(1), 1991.
10. Sandhu R.S. and Samarati P. Access control: principles and practice. IEEE Commun., 32(9):40–48, 1994.

Disk

Peter Boncz
CWI, Amsterdam, The Netherlands

Synonyms

Hard disk; Magnetic disk; Disk drive

Definition

In disk storage, data is recorded on planar, round and rotating surfaces (disks, discs, or platters). A disk drive is a peripheral device of a computer system, connected by some communication medium to a disk controller. The disk controller is a chip, typically connected to the CPU of the computer by the internal communication bus. Main implementations are hard disks, floppy disks and optical discs, of which the first is the usual interpretation.

Recently, Solid State Disks have been introduced; though the term "disc" is a misnomer for these devices, as internally they consist of NAND Flash memory chips. Similarly, the term RAM Disk is used for a storage device consisting of volatile DRAM memory. Both offer the same data storage abstraction as a hard disk at the operating system level, though their price, size, performance and persistence characteristics are very different from a hard disk.

Key Points

The history of the hard disk started at IBM in San Jose, California, when Rey Johnson created a rotating drum that was coated in a magnetically polarizable film that

RPM	3,600	5,400	7,200	10,000	15,000	Solid state
Disk model	CDC wrenl 94145–3	Seagate ST41600	Seagate ST15150	Seagate ST39102	Seagate ST373453	Samsung MCBOE
Year	1983	1990	1994	1998	2003	2008
Capacity (GB)	0.03	1.4	4.3	9.1	73.4	32
Seq. bandwidth (MB/s)	0.6	4	9	24	86	80
Read latency (ms)	48.3	17.1	12.7	8.8	5.7	0.3
Write latency (ms)						40

could be used to store data by changing and sensing magnetic polarization.

Hard disks nowadays consist of a number of platters connected by a single axis, spinning at a fixed number of rotations per minute (rpm). Data is on a platter organized by track (distance from the center) and sector (angular region on a track). The disk head, one for each platter, mounted on a disk arm that moves in and out, therefore cover more distance per unit of time on an outer track than on an inner track. To read or write data, the head must be moved to the correct position above the track (seek time), and then wait until the region of interest spins past it (rotational delay). Therefore, the average random access latency of hard disks is seek time plus rotational delay divided by two. The bandwidth of a disk is determined by both communication infrastructure with the controller as well as rotation speed and data density, which determine the amount of bits that pass the head per second. Data density is closely related to disk capacity and has increased enormously, surpassing the historical development of any other performance factor in computer architecture (i.e., orders of magnitude faster than latency and quite a bit faster than CPU MHz as well as disk bandwidth). The historical development of disk performance parameters is shown in the below table. For comparison, the last column shows characteristics of a solid state drive.

The consequence of these developments is that relatively speaking, disk latency currently is much slower with respect to all other performance factors than it was a few decades ago, which means that fast and scalable algorithms involving I/O must focus more on sequential bulk access than fine-grained random access. Managing and optimizing I/O access is one of the primary tasks of a database system. In order to counter the trend of expensive random disk access latency, modern database systems (should) make use of asynchronous I/O to amortize disk arm movements over multiple requests, multi-disk or RAID systems to increase the I/O operation throughput, larger page sizes, as well as compression, clustered indices and efficient cooperative scan techniques to profit more from efficient sequential I/O.

Cross-references

▶ CPU
▶ Non-Volatile Memory (Flash)
▶ RAID
▶ Random Access Memory (RAM)
▶ Storage Devices
▶ Tapes

Disk Array

▶ Redundant Array of Independent Disks (RAID)

Disk Drive

▶ Disk

Disk Power Saving

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Definition

The term Disk Power Saving refers to the function of reducing the electric power that is consumed in a disk drive. Typically, Disk Power Saving cuts power supply to a particular component of the disk drive or slows down the component. In most cases, the disk drive cannot respond to input/output requests while it is in such a power saving mode.

Key Points

Power modes of modern disk drives are often classified as normal and stand-by. A disk drive in the normal mode is either processing read/write requests or ready to immediately start processing read/write requests. In this mode, all the components are almost fully operating. The disk drive thus consumes higher electric power than in the other modes. In contrast, a disk drive in the other modes cannot start processing read/write requests immediately. In the stand-by mode, the spindle motor is spun down and the head assembly is unloaded to the ramp and powered off; only the controller or a partial circuit of the controller remains operational. The disk drive thus consumes much less power than in the normal mode. Yet, when a disk drive accepts a read/write request, the controller needs to spin up the spindle motor and power on the head assembly again so as to transition to the normal mode. This process is often time- and energy-consuming.

Many commercial disk drives support stand-by commands such that operating systems and applications running on host computers can control disk power modes. In addition, some of the disk drives have the capability of threshold-based hibernation; those disk drives can automatically change to the stand-by mode when a specified time has elapsed after the last read/write access.

The difficulty of power management of disk drives is due to the significant penalties of mode changing. Many commercial disk drives need more than 10 sec and over one hundred joules to change from the stand-by mode to the normal mode.

Some manufacturers are providing more flexibility by introducing new types of power saving modes such as unloaded and low-rpm. In the unloaded mode, the head assembly is merely unloaded to the ramp and the other components are normally operating. In the low-rpm mode, the head assembly is unloaded to the ramp and powered off, but the spindle motor is spinning at lower rotational speeds. These new modes can save less power but give also smaller penalties than the traditional stand-by mode.

In another approach, some research groups are studying the possibility of multi-speed disk drives, which have the capability of dynamically changing rotational speeds in the normal mode. The use of the multi-speed disk drives sounds effective, but only several prototypes have been reported so far.

Cross-references

▶ Massive Array of Idle Disks
▶ Storage Power Management

Recommended Reading

1. Gurumurthi S., Sivasubramaniam A., Kandemir M., and Franke H. Reducing disk power consumption in servers with DRPM. IEEE Comput., 36(12):59–66, 2003.
2. HGST Inc. Quietly cool. White Paper, HGST, 2004.
3. Yada H., Ishioka H., Yamakoshi T., Onuki Y., Shimano Y., Uchida M., Kanno H., and Hayashi N. Head positioning servo and data channel for HDDs with multiple spindle speeds. IEEE Trans. Magn., 36(5):2213–2215, 2000.

Disk Process

▶ Storage Manager

Disk-based Model

▶ I/O Model of Computation

Distance between Streams

▶ Stream Similarity Mining

Distance Indexing

▶ Indexing Metric Spaces

Distance Space

▶ Metric Space

Distance-preserving Mapping

► Space Filling Curves
► Space-Filling Curves for Query Processing

Distillation

► Summarization

Distortion Techniques

Carpendale Sheelagh
University of Calgary, Calgary, AB, Canada

Synonyms

Fisheye views; Nonlinear magnification; Multiscale views; Detail-in-context; or Focus-plus-context

Definition

While the word "distortion" often has unfavorable connotations in terms of data, a *distortion technique* in digital information viewing or data exploration is the use of deformation of some aspect of the information or data in order to provide a better view or better access to some other aspect of the data.

Historical Background

The uses of distortion in digital information exploration interfaces have two independent starting points: Spence and Apperley's Bifocal Display [18] and Furnas' Generalized Fisheye Views [5]. From these origins, research initially focused on developing algorithmic solutions for distortion techniques. Well-known examples include: Sarkar and Brown's Graphical Fisheyes [16], which expand upon Furnas' approach creating spatial reorganizations of visual representations; Hyperbolic Display [9], which uses mathematical function to create detail-in-context presentations; Perspective Wall [11], and Document Lens [15], which make use of 3D perspective projection to create detail-in-context presentations; and Elastic Presentation [3], which also uses 3D manipulations and perspective projection to offer a mathematical framework that encompassed distortion techniques to date. Other methods [1,5,8,12,16] create distortion presentations by using a 2D-to-2D transformation function to spatially adjust a given two-dimensional layout (for survey see [10]).

Though many varieties exist and there continues to be interest further developing the domain within the research community, distortion techniques have not yet received widespread acceptance. This may be due to a general discomfort with the use of distortion and fears that distortion might lead to misinterpretation of data. There is still room for significant research into the relative merit among differing distortion techniques and it is likely that this kind of evaluation will be dependent on the type of task, the nature of the information, and the preferences and skills of the person using it.

Foundations

With regard to data presentation, access, navigation, and exploration, the development of distortion techniques has not been the goal in itself. Instead, these techniques are used to achieve, aid, or ease a data- or task-related goal. As such, the discussion and concepts that motivate the development of distortion techniques are important and remain relevant.

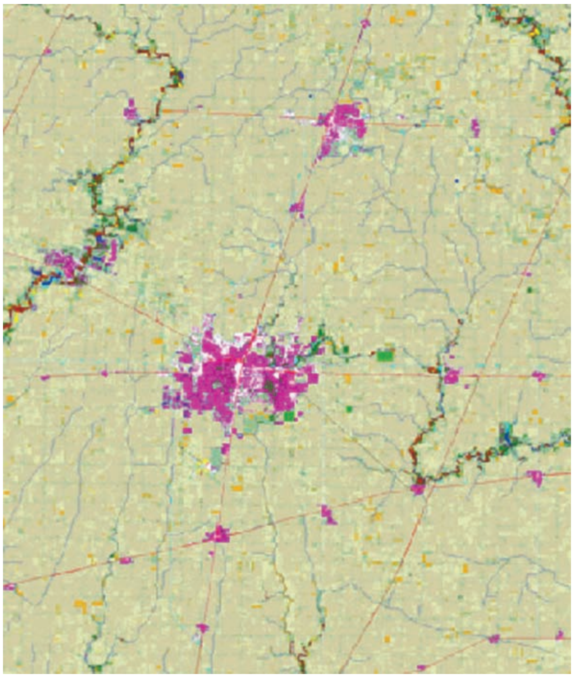
The primary motivation has always been a lack of display space. Whether it is because of the sheer size of the data to be displayed or because of the number of related windows needed to accomplish a task, it seems that there is never enough display space. While the amount of available display space is expanding – note current interest in high-resolution, large displays – it is not keeping up with either the computing power or the deluge of data. The response to this trend has been through interaction – zooming, scrolling, and panning – and various distortion techniques. Different types of distortion techniques were initially motivated independently.

Spence and Apperley [18] started by noting the frequency of both crowding and navigation problems in interfaces in general. Their response was based on an understanding of how human memory is often less than precise and can lead to fuzzy search practices. They discuss how searching in physical space is supported by several factors including spatial memory, memory of previous actions, and visual and verbal clues and can be reinforced by a reasonable degree of constancy. They suggest that interfaces might better support these factors by maintaining a full context in which spatial constancy is preserved. Spatial constancy requires ensuring that all items stay present, and that all parts or regions stay in the same positions relative to each other. This thinking led to integrated views with two distinct levels of magnification, called Bifocal Displays [18].

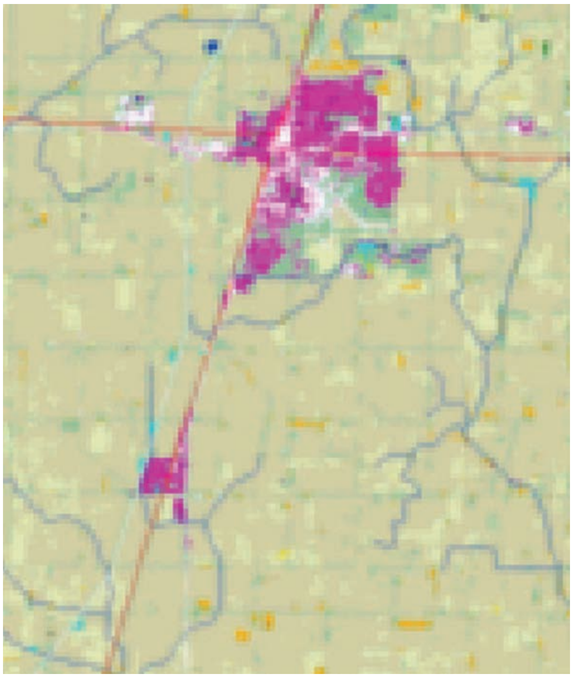
Furnas [5] studied how people retain and present information in various subject areas and workplaces such as geography, history, and newspapers and noted that people usually know the details about their own interests, set in enough domain knowledge to provide context. His widespread evidence led him to suggest that Generalized Fisheye Views may be a useful and intuitive way to present information. A Generalized Fisheye View has a focus or center of interest about which detail is displayed, gradually decreasing in detail as the distance from the focus increases, with the exception that some specific items of noted importance within the domain should be included regardless of the distance from the focus. To achieve this, Furnas proposed filtering the context by using a degree of interest (DOI) function. A DOI is based upon the distance from the current focus and a domain-specific a priori importance (API) function. These concepts led to a variety of filtered and variant magnification presentations.

Another much-discussed point is the importance of visual integration in reducing cognitive load. If a particular data presentation is too large to fit the available display space, it can be compressed uniformly to fit (Fig. 1). This can result in a presentation that is too dense to discern detail. For instance, in Fig. 1 one cannot see the two airports, which should show as white lines on a green background. To obtain a better view of these details one can zoom in, as in Fig. 2, and see one of the airports. The viewer must now either flip between views to know which airport is being displayed or view both the compressed and detailed view side by side and work out how the two views relate to each other. In this case, this is not too difficult, but does impose some additional attentional costs. Local magnification or an inset-in-place can be used, as in Fig. 3, but now the actual connections are occluded. Alternatively, an integrated detail-in-context view can be used (Fig. 4). Note that while visual integration is maintained, it makes use of a distortion technique [3].

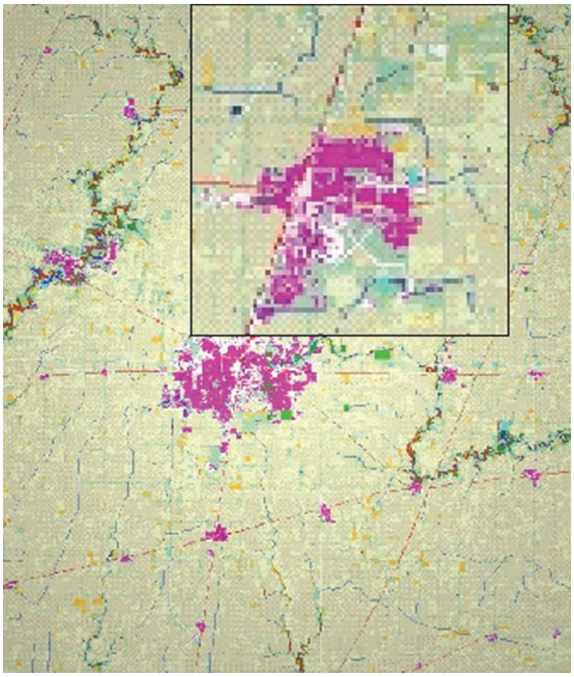
Several researchers noted that temporal continuity was also important [3,17]. Misue et al. [12] raised the issue of developing techniques that support a viewer's mental map. In particular, they note that the



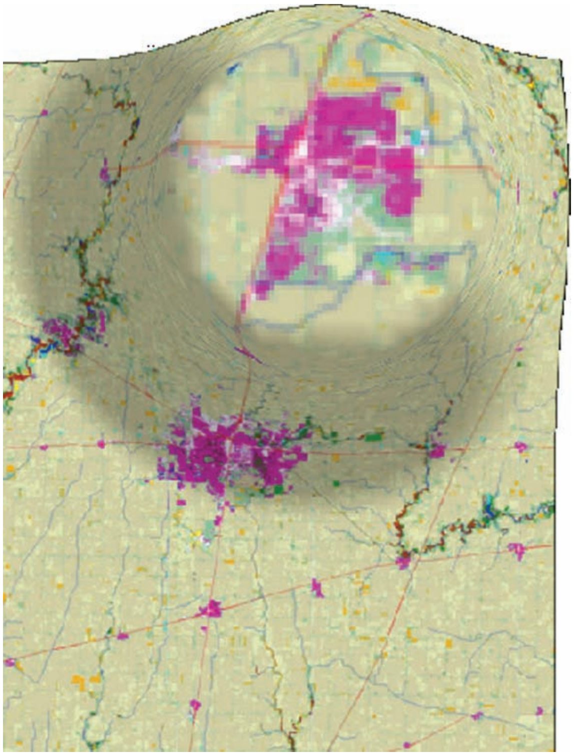
Distortion Techniques. Figure 1. This image shows a land usage map of Champaign, Illinois compressed uniformly to fit.



Distortion Techniques. Figure 2. Magnifying one region of the land use map of Champaign Illinois reveals details of airport runways not previously visible.



Distortion Techniques. Figure 3. Magnified inset occludes local context.



Distortion Techniques. Figure 4. Distortion technique provides integrated detail-in-context view.

mathematical concepts of proximity, orthogonality, and topology should be algorithmically ensured. Carpendale et al. [2] noted that viewing cues can aid in the general readability of data presentations that utilize distortion techniques.

In summary, the issues and considerations around the development of distortion techniques include:

- Maintenance of spatial constancy to support spatial memory;
- Maintenance of full context, that all items maintain some visual evidence of presence;
- Introduction of more than one magnification scale in a unified presentation;
- Reduction of cognitive effort needed for reintegration of information across separate views;
- Support for setting detail in its context, as is common practice in human memory patterns;
- Maintenance of sufficient context;
- Providing for varying needs for domain significant data within a reduced context;
- Varying the amount of detail needed around the focus;
- Exploring the possibility of more than one area of interest, or foci;
- Utilizing degree of interest (DOI) functions, as appropriate;
- Providing visual continuity;
- Providing temporal continuity; and
- Supporting the mental map by maintaining proximity, orthogonally, and topology.

Three frameworks emerged from the great variety of distortion techniques developed. Leung and Apperley [10] categorized existing methods and unified them by showing how to derive 2D-to-2D transformations for 3D-based techniques. However, these are complex formulations and have not been implemented.

There is also a family of 2D-to-2D distortion techniques. In these, a 2D data representation is transformed through use of a distortion technique to create an adjusted presentation that provides magnified detail in the foci, which are set in their context via distorted regions. Of these techniques, Magnification Fields [8] is probably the most general. It describes an approximate integration based approach that, given a pattern to strive for, can create a magnification pattern. A 2D distortion transformation function performs adjustments in x and/or y. The resulting pattern of magnification and compression is the derivative of the transformation function. Magnification Fields determines the transformation function from the magnification function [8]. This would allow a person to create a multiscale view by requesting the pattern magnification that suits their task. Their approach starts with a grid and a set of desired magnification amounts. The grid is adjusted iteratively, ensuring that no grid points overlap until the difference between the magnification provided by the adjusted grid and the desired magnification is sufficiently small [8].

Elastic Presentation Framework (EPF) [3] unifies many individual distortion techniques, allowing the seamless inclusion of more than one distortion technique in a single interface. By interpolating between the methods it describes, EPF identified new variations. EPF achieved previous 2D-to-2D approaches using an intermediate 3D step. The 3D-based approaches are quite different algorithmically than their 2D counterparts. The plane or surface that holds the two-dimensional representation is manipulated in three dimensions, and then viewed through single-point perspective projection. The transformation function results from the combination of the manipulation of the surface and the perspective projection. This combination simplifies the mathematics of the relationship between magnification and transformation to the geometry of similar triangles.

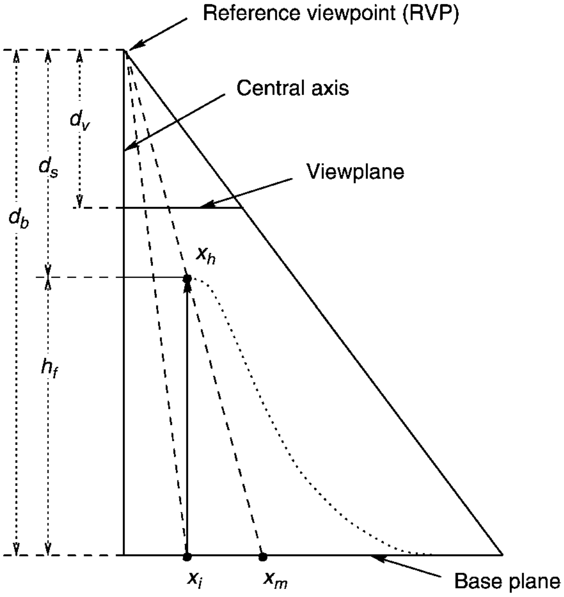
In EPF, data are placed on a 2D plane, which is then placed in a 3D viewing volume on the *baseplane*, parallel to the viewplane at a distance along the *z* axis from the viewpoint which defines unit magnification. The next step is to provide the focal regions with the intended magnification. What is needed is an asymptotic function that relates degree of magnification to *z*-translation that also guarantees fine control as the viewpoint is approached. This function can be derived from similar triangles shown in Fig. 5 as follows:

$$\frac{x_m}{x_i} = \frac{d_b}{d_s}$$

$$mag = \frac{x_m}{y_i}$$

$$h_f = d_b - d_s$$

where x_i is a point on the baseplane that is raised to a height h_f providing a magnification of *mag*. The position x_m is the apparent location after the displacement of the point x_i to a height h_f:

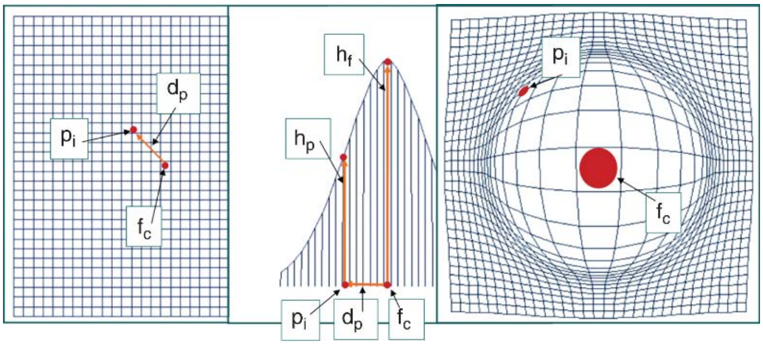


Distortion Techniques. Figure 5. The relationships between the displaced points and the apparent magnification.

$$h_f = d_b - \left(\frac{d_b}{mag}\right)$$

This function offers infinite magnification control, which is limited only by the numerical resolution of the computer. The coordinates (x_m, y_m) allow the option of performing transformations directly by translating the point in *x* and *y*, or through perspective by adjusting the height. To ensure full visibility for multiple foci, the foci are viewer-aligned. That is the focal center is aligned to the viewpoint. To ensure uniform magnification response throughout the display the translation vectors are normalized in *z* (see [3]).

Now that focal magnification is obtained, an appropriate distortion that will link the foci to the context is needed. This is achieved through a drop-off function. Points on the surface are translated depending on the value of the drop-off function when applied to the distance of the point from the focal region. The extent of the spread of the distortion into the context can be controlled by the viewer through adjustments to the domain and range of the drop-off function. In Figure 6, the focal point f_c is raised to height h_f according to the



Distortion Techniques. Figure 6. Providing an integrated context.

magnification required. The height h_p of any point p_i at a distance d_p from the focal point f_c can be calculated.

Many drop-off functions are effective. Perhaps the simplest is a linear drop-off. In an EPF linear drop-off function, the surface height h_p of a point $_p$, that is a distance $d_p$from the focal centre f_c with a lens radius l_r, and a focal height h_f is calculated by:

$$h_p = h_f * (1 - (d_p/l_r))$$

Varying the lens radius l_r affects the limits of the lens and the resulting slope. A Gaussian drop-off function offers a smooth visual integration with its bell curve.

$$h_p = h_f * \exp^{-((d_p^2/\sigma)}$$

An inverse power function offers a very good drop-off function for achieving high focal magnifications while still offering full visual integration in the distorted region.

$$h_p = h_f * (1 - (d_p/l_r))^k$$

Note that if $k = 1$, this is the equivalent to the linear drop-off function. Alternatively, setting $k = 2.7$ results in high-magnification focal regions. Having chosen the focal magnification and the drop-off, the manipulated surface is then viewed through perspective projection. Single-point perspective projection preserves angles, proximity, and parallelism on all x, y planes.

Many other possibilities exist for varying the distortion technique. For instance, different distance metrics can be used. An L_p metric offers a continuum between radial and orthogonal layouts (See Fig. 7). For 2D distances between points $p_1(x_1, y_2)$ and $p_2(x_2, y_2)$, L_p metrics are defined as:

$$L(P) = \sqrt[P]{|x_1 - x_2|^P + |y_1 - y_2|^P}$$

where $L(2)$ is Euclidean distance.
The $L(\infty)$ metric is:

$$L(\infty) = \sqrt[\infty]{|x_1 - x_2|^\infty + |y_1 - y_2|^\infty}$$

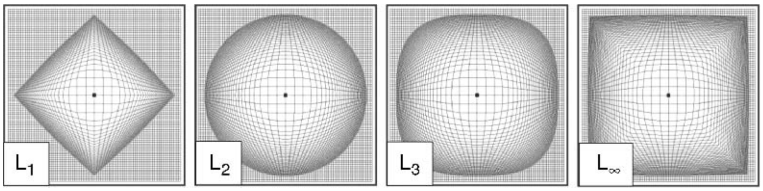
which resolves to:

$$L(\infty) = \max(|x_1 - x_2|, |y_1 - y_2|)$$

While there is continued interest in developing better distortion techniques – a recent new framework [14] offers an image-based approach and incorporates transparency blending into distortion techniques – the focus within the research community has increasingly shifted to empirical work. Distortion techniques have been shown to offer advantages for command and control [17], web navigation [13], and menu selection [1]. However, usability problems have also been shown to exist in important tasks such as targeting, steering, and layout [6,7].

Key Applications

There has not yet been widespread acceptance of distortion techniques in readily-available applications (notable exceptions include the Mac OS X Dock and www.idelix.com). In considering potential applications, it is important to return to the definition of distortion techniques, which involved a willingness to make use of distortion to achieve a data or task goal. In this light, it is probable that current distortion frameworks and lenses will in the future be considered as relatively crude functions. Consider a practical



Distortion Techniques. Figure 7. L-Metrics.

example: drawing a map in which two quite distant coastal towns are connected by two roads. One road runs along the shoreline and the other road runs high above it along a cliff top. Because of the cliff, there are no connections between these two roads except at the towns. Within the available display space, drawing the map faithfully and to scale would place the roads overtop each other at several points along the way. This is a very familiar cartographic problem and distortion has been used for centuries to solve it. Here, the data accuracy that is preserved is the fact that no connection between the roads exists. The spatial distortion is a small price to pay. However, most current distortion techniques would not provide a subtle and graceful adjustment. There is considerable open research space for more subtle and controllable distortion techniques. Also, as Furnas [4] recently noted, some of the concepts like DOIs lend themselves to integrated solutions involving data as well as spatial adjustments.

Cross-references

- ► Context
- ► Contextualization
- ► Data Transformation Methods
- ► Information Navigation
- ► Visualization for Information Retrieval
- ► Zooming Techniques

Recommended Reading

1. Bederson B. Fisheye menus. In Proc. 13th Annual ACM Symp. on User Interface Software and Technology, 2000, pp. 217–225.
2. Carpendale S., Cowperthwaite D., and Fracchia F.D. Making distortions comprehensible. In Proc. 1997 IEEE Symp. on Visual Languages, 1997, pp. 36–45.
3. Carpendale S. and Montagnese C. A Framework for unifying presentation space. In Proc. 14th Annual ACM Symp. on User Interface Software and Technology, 2001, pp. 61–70.
4. Furnas G. A fisheye follow-up: further reflections on focus+context. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2006, pp. 999–1008.
5. Furnas G.W. Generalized fisheye views. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1986, pp. 16–23.
6. Gutwin C. Improving focus targeting in interactive fisheye views. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2002, pp. 267–274.
7. Gutwin C. and Fedak C. Interacting with big interfaces on small screens: a comparison of fisheye, zoom, and panning techniques. In Proc. Graphics Interface, 2004, pp. 213–220.
8. Keahey A. The generalized detail-in-context problem. In Proc. IEEE Symp. on Information Visualization, 1998, pp. 44–51.
9. Lamping J., Rao R., and Pirolli P. A focus+context technique based on hyperbolic geometry for visualising large hierarchies. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1995, pp. 401–408.
10. Leung Y. and Apperley M. A review and taxonomy of distortion-oriented presentation techniques. ACM Transactions on Computer Human Interaction, 1(2):126–160, 1994.
11. Mackinlay J., Robertson G., and Card S. Perspective wall: detail and context smoothly integrated. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1991, pp. 173–179.
12. Misue K., Eades P., Lai W., and Sugiyama K. Layout adjustment and the mental map. J. Visual Lang. Comput., 6(2):183–210, 1995.
13. Munzner T. and Burchard P. Visualizing the structure of the world wide web in 3D hyperbolic space. In Proc. 1995 Symp. Virtual Reality Modeling Language, 1995, pp. 33–38.
14. Pietriga E. and Appert C. Sigma lenses: focus-context transitions combining space, time and translucence. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2008, pp. 1343–1352.
15. Robertson G.G. and Mackinlay J.D. The Document lens. In Proc. 6th Annual ACM Symp. on User Interface Software and Technology, 1993, pp. 101–108.
16. Sarkar M. and Brown M.H. Graphical fisheye views. Commun. ACM, 37(12):73–84, 1994.
17. Schaffer D., Zuo Z., Greenberg S., Bartram L., Dill J., Dubs S., and Roseman M. Navigating hierarchically clustered networks through fisheye and full-zoom methods. ACM Transactions on Computer Human Interaction. 3(2):162–188, 1996.
18. Spence R. and Apperley M.D. Data base navigation: an office environment for the professional. Behav. Inform. Technol., 1(1):43–54, 1982.

Distributed Architecture

Tore Risch
Uppsala University, Uppsala, Sweden

Synonyms

Parallel database; Federated database; Multi-database; Peer-to-peer database

Definition

A distributed database [3] is a database where data management is distributed over several nodes (computers) in a computer network. In a central DBMS the data is managed by one node whereas in a distributed DBMS the data is managed by several nodes. A *distributed DBMS* is a database manager consisting of several nodes distributed over a network. Each node is a database manager by itself that communicates with other nodes in the network. In a regular distributed DBMS it is up to the database administrator to manually specify how data collections (e.g., relational tables) are distributed over the nodes when a distributed database is designed. Queries and updates to the distributed relations are transparently translated by the distributed DBMS into data operations on the affected nodes giving the user the impression of using a single database, called *query* and *update transparency*. Thus the distributed DBMS provides distribution transparency for database users but not for the database administrator.

Closely related to distributed DBMSes are *parallel databases* where a parallel DBMS engine runs on usually a cluster. The parallel DBMS automatically determines how data structures are internally distributed over the nodes providing distribution transparency also for the database administrator, called *schema transparency*.

The purpose of *heterogeneous databases* is to be able to combine data from several independently developed autonomous databases. Heterogeneous databases can be divided into federated databases, mediators, and multi-databases. In a *federated database* the database administrator defines a single *global integration schema* describing how data in underlying databases are mapped to the integration schema view. This provides distribution transparency for integrated data. *Mediators* allow the definition of several views over data from different data sources. Since it may be difficult to define integration schemas and views when there are many participating autonomous databases, *multi-databases* relax the distribution transparency also for the database users who there specify queries and updates using a multi-database query language where individual data collections in the participating nodes can be explicitly referenced.

A related technology is *peer-to-peer systems* where networks of files are distributed over the Internet. Meta-data is associated with the files and the user can search for files satisfying conditions. Peer-to-peer search is usually made by propagating queries between the peers. The consistency and correctness of queries are relaxed compared to regular databases in order to provide better performance and node autonomy.

Historical Background

Distributed DBMSs were pioneered by System R and Ingres* in the beginning of the 1980s. Early distributed DBMSs assumed slow communication between nodes having limited amounts of main memory geographically distributed in a wide area network. The database administrator instructed the distributed DBMS where to place data, while the user could specify transparent queries to the distributed DBMS without detailed knowledge of where data was placed.

The evolvement of computer clusters provided hardware resources for very high performing database servers running on clusters, *parallel databases*. Since the communication between cluster nodes is very fast and not geographically distributed, the database administrator need not provide manual placement rules of distributed data, i.e., the parallel DBMS provides full distribution transparency also for the database administrator. With the evolvement of fast wide area computer networks parallel DBMS technology can be used also for some geographically distributed databases. However, it should be noted that update latency has to be taken into account for large geographical distances because of the speed of light. In general geographically distributed databases still requires manual distribution.

Not least the development of the Internet has caused the need to integrate data from many pre-existing databases. The area of *heterogeneous databases* [4,2] deals with tools and methodologies to combine data from several autonomous databases. While distributed and parallel databases assumed all data managed by one distributed DBMS, heterogeneous databases integrate databases using different DBMS and different schemas.

There are several flavors of heterogeneous databases:

1. *Federated databases* require the definition of a *global integration schema* containing mappings to the participating databases' schemas. The federated database becomes a central server on top of the participating autonomous databases.
2. As the number of databases to integrate increases it becomes very difficult or impossible to define a global integration schema over the large numbers of autonomous databases. *Multi-databases* [2] provide no global conceptual schemas and instead a *multi-database query language* allows specification of queries searching through many participating databases.
3. Mediators [5] provide a middle ground between a single integration schema and no schema at all. Instead the user can define mediator views that combine and reconcile data from different data sources. Such views require a query language that can express queries over several databases, i.e., a multi-database query language. The mediator system becomes a middleware between users and wrapped data sources.

While distributed databases could handle transparent queries and updates for a small number of nodes, the evolvement of the Internet requires technologies to deal with geographically distributed databases having 1,000s of nodes. *Peer-to-peer systems* enable such highly distributed file access where users search for data stored in peers. In a *peer-to-peer database* queries are propagated between the participating peer nodes. To improve performance at the expense of query correctness the propagation may stop after a certain number of hops. This is sufficient for many modern applications that do not have strict consistency requirements; for example Internet search engines do not guarantee the full correctness of answers.

Foundations

Autonomy and Heterogeneity

Different distributed DBMS architectures provide different levels of autonomy for the participating nodes.

A *homogeneous* distributed database is a distributed database where all nodes are managed by the same distributed DBMS. A homogeneous distributed database can be regarded as a central database distributed over many nodes where data and processing is internally transparently distributed over several nodes. By contrast, a heterogeneous database is a (distributed or central) database where data originates from participating autonomous databases possibly using different DBMSs.

Regular distributed and *parallel* databases are homogeneous. One distributed DBMS manages all data. *Distributed database design* involves designing the schema in a top-down fashion as for a conventional central database. Parallel databases provide automatic and transparent data placement without user intervention, while regular distributed databases require the database administrator to specify how data should be distributed over nodes. In regular distributed and parallel database the nodes have no autonomy at all.

Federated databases are central database servers that integrate data from several participating databases. Federated databases are thus heterogeneous. *Global integration schemas* are defined that integrate data originated in the participating databases. The design of the integrated schema needs to deal with data integration issues on how to combine the same or similar data represented differently in different participating databases. Different participating databases may use different DBMSs. The schemas of the participating databases are designed before the integrated database schema is designed. Thus the design process for heterogeneous databases becomes bottom-up, whereas homogeneous databases are usually designed top-down. The design of the integrated schema needs to deal with data integration issues on how to combine the same or similar data represented differently in different participating databases.

Both *federated databases*, *mediators*, and *multi-databases* are heterogeneous. The main difference between them is how integration schemas are defined. Federated database assume one global integration schema. If there are many different participating databases it is difficult to define such a global integration schema. This is relaxed in mediators, which allow the definition of many integration schemas as views over wrapped underlying data sources of different kinds. In multi-databases the user is given access to a *multi-database query language* where he can specify queries over many sources. A multi-database query language provides the basis for defining mediator views.

Finally, the aim of peer-to-peer databases is distributed queries in a widely distributed network of heterogeneous nodes. Unlike parallel and distributed databases the individual nodes are not managed by a single system, but independently.

Transparency

Distributed databases can be classified according to what kinds of transparency they provide w.r.t. distribution of data. Three different kinds of transparency can be identified for different kinds of services provided by the distributed DBMS, *schema transparency*, *query transparency*, and *update transparency*.

Schema transparency means that the distributed DBMS decides completely on its own where to place data on different nodes. The database administrator has the impression of using a single database and specifies the logical schema without considering any distribution at all. However, often it is desirable to allow the database administrator to specify how to distribute data, and thus relax schema transparency. For example, for performance and to allow local control, a geographically distributed database for a large enterprise may need to cluster employee data according to the countries where departments are located. Therefore full schema transparency is often provided only on local area networks or cluster computers where the communication between nodes is very fast.

With *query transparency* the distribution of data is not reflected in user queries. Queries are transparently translated by a distributed query optimizer into queries and updates to the affected nodes giving the user the impression of using a single database. By analyzing a given user query the distributed query optimizer can often statically determine which nodes to access. Query execution plans can execute in parallel on different nodes with partial results transported between nodes and combined on other nodes. Query transparency is very important for distributed databases since it is very difficult and error prone to manually implement distributed communicating execution plans.

Update transparency allows database updates to be specified without taking distribution into account. A distributed transaction manager propagates updates to affected nodes.

Distributed or Parallel DBMS Provide Update Transparency

In the classification above, only *parallel DBMSs* provide complete transparency for everyone using the database, database administrators as well as users. The term *regular distributed database* refers to a distributed DBMS with query and update transparency but without schema transparency.

With *naming transparency*, users are provided with a single name for a distributed relation defined in terms of several internal relations stored on separate nodes. Regular distributed, parallel, federated, and peer-to-peer databases provide naming transparency, which is relaxed for mediators and multi-databases.

Distributed database design involves manual specification to the distributed DBMS of the distribution of data collections. The database administrator can tune the data placement in a wide area computer network. The two fundamental methods for such manual data distribution are *fragmentation* and *replication*. Fragmentation splits a collection (e.g., Table 1) into separate non-overlapping segments on different nodes, while replication stores identical copies of a collection on different nodes. The distributed DBMS guarantees that queries and updates of fragmented or replicated collections are transparent so the user need not be aware of how data is distributed.

Fragmentation (or partitioning) allows the administrator of a distributed database to manually specify on which nodes the DBMS should place different sections of each distributed data collection. In a distributed relational database tables are fragmented. For example, the placement of employee records in a relation can be fragmented according to in which countries different employees work. Fragmentation speeds up database queries and updates since it allows parallel access to distributed fragments. Furthermore, by analyzing queries and updates the query optimizer can often determine exactly which nodes are affected and send the query/update statements only to those nodes.

Replication allows the DBA to declare to the DDBMS to place the same data collections on more than one node. For example, a relational table may be replicated on several nodes. Replication speeds up data access at the expense of update cost. However, as explained below, if consistency is relaxed the update cot may be reduced.

Federated databases also provide query and update transparency by allowing the database administrator to

Distributed Architecture. Table 1. The Architectures of DDBMSs can be Classified Along Different Dimensions. The Following Table Classifies Different Kinds of Distributed DBMS Architectures

	Autonomy	Schema transparency	Query transparency	Update transparency	Naming transparency	Central schema
Parallel	No	Yes	Yes	Yes	Yes	Yes
Regular Distributed	No	No	Yes	Yes	Yes	Yes
Federated	Yes	No	Yes	Limited	Yes	Yes
Mediators	Yes	No	Yes	Limited	No	No
Multi-databases	Yes	No	No	No	No	No
Peer-to-peer	Yes	No	Yes	Yes	Yes	No

define a *global integration schema* that hides the underlying integrated databases.

Mediators provide some query transparency by allowing users to define views over integrated databases. Update transparency is more problematic as it requires updatable views.

With *multi-databases* transparency is further relaxed so the user can reference individual databases explicitly in queries and updates.

Finally, *peer databases* [1] provide query and update transparency in widely distributed systems but do not require fully correct query answers.

Consistency

If data is widely distributed over many nodes in a network, the cost of maintaining data consistency may be very high. The transaction manager must guarantee that all transactions are atomic and updates propagated to affected nodes so that the database is kept consistent. Two and three phase commit protocols are needed when more than one node is affected by an update to guarantee full update transparency. These protocols are expensive when many nodes are involved and relaxed update transparency may suffice to enable higher transaction performance. If the same kind data is present on many nodes updates must be propagated to all replicas, which can be very expensive in a geographically distributed database.

Regular distributed databases usually provide transaction atomicity as an option. However, because of the high cost of transaction atomicity modern distributed DBMS also provide the option to propagate updates lazily, thus compromising the consistency.

In a parallel DBMS running on a cluster, the nodes inside the cluster run DBMS kernel software which is completely controlled by the parallel DBMS. From the user's point of view it looks like a central DBMS; the main difference being the higher performance provided by parallelization of DBMS kernel software.

In regular distributed and parallel DBMSs a single database kernel manages all distributed data. All individual nodes are running the same distributed DBMS software. Different nodes may have different roles, e.g., some nodes handle query processing, some nodes handle locking, some nodes handle recovery, etc. The DBMS is a monolithic systems distributed over several nodes controlling the consistency of the individual nodes.

In general, consistent updates are difficult to achieve with heterogeneous databases since the participating databases are autonomous and the integrating DBMS may not have access to transaction managers of the participating databases.

In peer-to-peer databases the data consistency is relaxed for higher update and query performance. Data can be partly replicated for efficiency but the system does not guarantee consistency among the replicas so updates need not always be propagated to all replicas at every update. This means that queries may return less reliable result, which is often acceptable in a widely distributed database. This is similar to how search engines compromise query quality for performance.

Distributed Catalog Management

A particular problem for distributed databases is how and where to handle catalog data, such as the overall schema, statistics about data collections, the location of data collections, and how data collections are replicated and partitioned. The catalog information is accessed intensively by database users in queries and updates. On the other hand, in most DBMSs it is assumed that schema and catalog information changes slowly, which, for example, permits pre-compilation of (distributed) database queries. The assumption that catalog data changes slowly but is intensively accessed is a case for replicating catalog information on many nodes, in particular on those coordinating nodes with which the users interact. On the other hand, in a heterogeneous database with many participating autonomous nodes, the assumption that schemas and data placements do not change usually does not hold.

Regular distributed and parallel databases assume few participating non-autonomous nodes and the catalog is therefore replicated. Federated databases have a central architecture where all interaction with the database is through the global schema and it contains replications of catalog information from the participating databases. For mediators, multi-databases, and peer-to-peer there is no central global schema and the query processing nodes are autonomous. Therefore the catalogue data cannot be fully replicated and it will be up to different nodes to cache catalog data when needed. The validity of cached catalog data needs to be properly handled though; otherwise queries may fail or even return the wrong data.

Cross-references

- ▶ Data partitioning
- ▶ Data dictionary
- ▶ Data replication
- ▶ Distributed Concurrecy Control
- ▶ Distributed database design
- ▶ Distributed database systems
- ▶ Distributed query optimization
- ▶ Distributed transaction management
- ▶ Distributed DBMS
- ▶ Information Integration
- ▶ Mediation
- ▶ Parallel query processing
- ▶ Parallel Database Management
- ▶ Peer Data Management System
- ▶ Shared-Nothing Architecture
- ▶ View-based data integration

Recommended Reading

1. Beng Chin Ooi and Kian-Lee Tan (guest eds.). Introduction: special section on peer-to-peer-based data management. IEEE Trans. Knowl. Data Eng., 16(7):785–786, 2004.
2. Litwin W., Mark L., and Roussopoulos N. Interoperability of multiple autonomous databases ACM Comput. Surv., 22(3):267–293, 1990.
3. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems (2nd edn.). Prentice Hall, NJ, 1999.
4. Sheth A.P. and Larson J.A. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv., 22(3):183–235, 1990.
5. Wiederhold G. Mediators in the architecture of future information systems. IEEE Comput., 25(3):38–49, 1992.

Distributed Commit Protocol

▶ Two-Phase Commit Protocol

Distributed Component Object Model

▶ DCOM

Distributed Computing Environment

▶ DCE

Distributed Concurrency Control

Mathias Weske
University of Potsdam, Potsdam, Germany

Synonyms

Synchronizing distributed transactions

Definition

Distributed concurrency control provides concepts and technologies to synchronize distributed transactions in a way that their interleaved execution does not violate the ACID properties. Distributed transactions are executed in a distributed database environment, where a set of connected data servers host related data. A distributed transaction consists of a set of subtransactions, each of which is executed by one data server. Distributed concurrency control makes sure that all subtransactions of a set of distributed transactions are serialized identically in all data servers involved. Therefore, not only local dependencies need to be taken into account, but also dependencies involving multiple data servers. Concurrency control techniques known from centralized database systems need to be extended to cope with the new requirements imposed by the distribution aspect.

Historical Background

Distributed concurrency control was an emerging research topic in the early 1980's. Based on the seminal work on concurrency control and locking protocols in centralized database systems by Eswaran et al. [3], Gray proposes implementation approaches for transactions [4], and Spector and Schwarz investigate transactions in the context of distributed computing [9].

In distributed database environments, data relevant to a specific application domain is spread across a set of data servers, each of which hosts a partition of the data items [2,8]. The data servers form a distributed database federation. Distributed database federations are homogenous if they run the same database software; they are heterogeneous if they do not.

To explain the core of distributed concurrency control, homogeneous database federations are addressed first. Since transactions are representations of application programs and data is spread across a set of data servers in a distributed environment, transactions need to access multiple data servers. Transactions with this property are called distributed transactions. Distributed transactions consist of a set of subtransactions, each of which runs at one specific data server of the federation.

Distributed transactions – just like centralized transactions – need to satisfy the ACID properties. Conflict serializability is a theoretically proven and practically relevant technique to ensure isolation of centralized transactions. Given a pair of transactions, all conflicting data access operations of these transactions occur in the same order, i.e., the serialization order of the transactions involved.

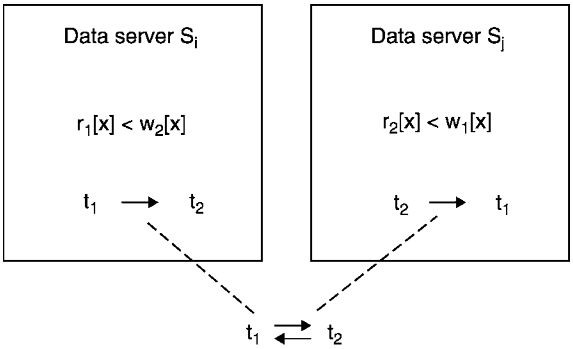
The complicating factor in the case of distribution is the lack of global knowledge. Assume transactions t_1 and t_2 running on data servers S_i and S_j. Subtransactions for t_1 and t_2 are created and started in S_i and S_j. These subtransactions are executed in the data sites according to the concurrency control technique in place. Since each data server is aware of the transactions and subtransactions that run locally, any serialization ordering between these transactions is fine, as long as no cycles in the global serialization graph are introduced.

As a result different serialization orders might emerge in the different data servers of the database federation. This is the case, if, for example, t_1 is serialized in S_i before t_2, while in S_j, t_2 is serialized before t_1. The serialization graph in S_i contains $t_1 \rightarrow t_2$, indicating that t_1 is serialized before t_2, while in S_j the serialization ordering is in opposite direction. Due to the lack of global knowledge, there is a cyclic serialization dependency, a violation of conflict serializability that none of the sites involved can detect. The distributed concurrency control problem is depicted in Fig. 1.

Foundations

In homogeneous database federations, distributed two phase locking is a variant of centralized two phase locking: All locks that a distributed transaction holds need to be held until no more locks need to be acquired by that transaction. This means, for instance, that a subtransaction of distributed transaction t_1 can only release its locks, if no subtransaction of t_1 will acquire additional locks. Local knowledge of a particular data server is insufficient to decide about unlocking data objects, because there can still be locking operations by other subtransactions of the transaction that are active in other data servers.

To provide a solution to this problem, information about distributed transaction needs to be



Distributed Concurrency Control. Figure 1. Distributed concurrency control problem: global serialization graph contains a cycle, although serialization graphs of data servers involved are acyclic.

communicated between the data servers involved. There are several approaches to providing this information. When atomic commitment protocols – such as the two phase commit protocol – are in place, then distributed two phase locking can take advantage from the commit protocols. To commit a distributed transaction *t*, the two phase commit protocol sends a prepare-to-commit message to all sites that host subtransactions of *t*. When a data server receives a prepare-to-commit message for a transaction *t*, *t* will no longer acquire new locks. When there is agreement on committing the distributed transaction, each data server commits the changes of its subtransaction of *t* and releases its locks.

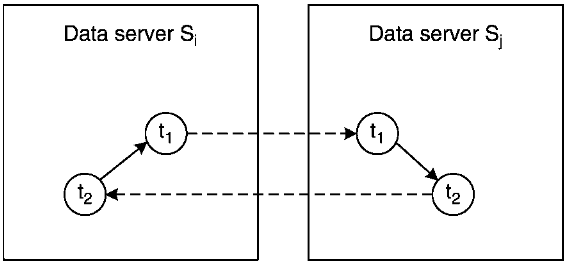
While distributed two phase locking solves the distributed concurrency control problem, the deadlock problem re-appears. Distributed deadlocks occur if conflicting locks are set by subtransactions of different transactions in a cyclic manner, and these locks are distributed among multiple sites of the database federation. As a result, no transaction can ever terminate successfully. In the example introduced above, if the subtransaction of t_1 in S_i holds a lock that the subtransaction of t_2 in S_i needs, in S_i, t_2 waits for the completion of t_1. A distributed deadlock exists if at the same time, t_1 waits for the completion of t_2 in S_j.

In this situation, there are local waiting conditions involving the subtransactions of t_1 and t_2, but there are also non-local waiting conditions: the subtransaction of t_1 in S_i waits for the completion of its subtransaction in S_j. Distributed deadlocks involve both local and non-local waiting conditions. This situation is shown in Fig. 2, marking local waiting conditions by solid edges and non-local waiting conditions by dotted edges.

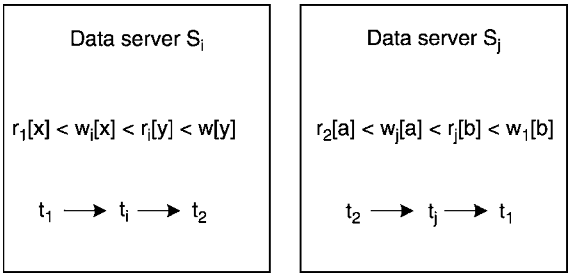
A simple approach to handling distributed deadlocks is timeout, where the time span that a lock can be held by a transaction is limited by a timeout value. If the timeout expires, the subtransaction is aborted. While distributed deadlocks are not possible using time-outs, potentially high abortion rates are introduced due to aborting transactions that are actually not involved in a deadlock cycle. More elaborate techniques are based on detecting distributed deadlocks, which is done by communicating information about waiting conditions between the data servers of a federation and aborting only those transactions that are actually involved in a distributed deadlock situation [6].

Timestamp ordering does not suffer from the deadlock problem; in this concurrency control technique, a strict ordering on the set of transactions is introduced. This ordering defines the order in which the transactions are serialized. Conflicting operations $p_i[x]$ and $q_j[x]$ are executed in the order $p_i[x]$ $q_j[x]$, if and only if $ts(t_i)$ $ts(t_j)$, where $ts(t)$ denotes the timestamp of transaction *t*. In the centralized case, assigning timestamps to transactions is performed by the scheduler that uses a logical clock to assign unique timestamps to newly arriving transactions Fig. 3.

In the distributed case, the definition of globally unique timestamps can be achieved by so called Lamport clocks, introduced in [7]. Lamport clocks use the axiom that a message always arrives after it was sent. This axiom is used to synchronize clocks in a distributed system. Each message carries a timestamp that reflects the local clock value of the sending system when the message was sent. When a message arrives with, for instance, timestamp 8, the receiving system sets its local clock to 9, should it be lower than that at the time of message arrival. To provide globally



Distributed Concurrency Control. Figure 2. Distributed deadlock situation due to waiting conditions of subtransactions involving multiple sites of a database federation.


Data server S_i
r_1[x] < w_i[x] < r_i[y] < w[y]
t_1 → t_i → t_2
Data server S_j
r_2[a] < w_j[a] < r_j[b] < w_1[b]
t_2 → t_j → t_1

Distributed Concurrency Control. Figure 3. Due to indirect conflicts with local transactions in heterogeneous database federations, global transaction manager cannot detect violations of global serialization orderings.

unique timestamps for transactions, each timestamp also contains the identifier of the data server from which the transaction originated. If these mechanisms regarding timestamps are in place in the distributed database federation, concurrency control can be done by the timestamp ordering technique, just like in the centralized case.

To summarize, distributed concurrency control in homogeneous database federations provides synchronization transparency to application programs by using variations of concurrency control techniques that are in place in centralized database systems [1]. The situation is more complex in heterogeneous federations, where data servers run different database management systems or are legacy systems that provide their functionality via application programming interfaces that allow us to access data. Regardless of the particular system in place, however, it must allow subtransactions of distributed transactions to read and write data items.

The techniques that work well in the case of homogeneous federations cannot immediately be used in the heterogeneous case. To cope with the heterogeneity issue, a software component is introduced, responsible for controlling the order in which subtransactions and their operations are issued to the data servers. This software component is called global transaction manager.

While the global transaction manager can control the ordering of subtransactions and their operations, it cannot influence the ordering of local transactions and their operations in the data servers. These local transactions can introduce serialization problems that the global transaction manager cannot take care of. These problems are due to indirect conflicts between subtransactions of distributed transactions. Indirect conflicts involve local transactions, each of which has a direct conflict with a subtransaction of a distributed transaction. These conflicts lead to serialization orderings between subtransactions that are not visible by the global transaction manager.

Assume data servers S_i and S_j are in place, global transactions t_1,t_2, that access both data servers. There are local transactions t_i in S_i and t_j in S_j. In data server S_i, the local transaction manager can serialize $t_1 t_i t_2$, while data server S_j serializes the transactions as t_2,t_j, t_1. Assuming that there are no direct conflicts between the subtransactions of t_1 and t_2 in place, the non-matching serialization cannot be detected by the global transaction manager.

There are several approaches to deal with this problem. Global serializability can be achieved by local guarantees, for instance the property of rigorousness, discussed in [10]. There are also approaches that introduce direct conflicts between distributed transactions in the data servers and, thus, make sure that the local serialization orderings of the data servers are in line.

An approach to solve the distributed concurrency control problem in heterogeneous federations is based on global transaction managers creating direct conflicts between subtransactions of distributed transactions by introducing additional data access operations. These operations force local direct conflicts that are handled by the individual data servers, using their respective concurrency control techniques. These data items are known as tickets, and the operations as take-ticket operations.

To start accessing local data items, each subtransaction of a distributed transactions first needs to take a ticket and issues the respective operation. This operation reads the current value of the ticket (a data item stored in the data server) and increments it, thus forcing direct conflicts between any two subtransactions accessing data in a particular data server of a heterogeneous federation. This technique makes sure that there are always direct conflicts between subtransactions of distributed transactions. However, the conflicts can still have different orientation, and the serialization orderings might still be different.

The information about the conflicts needs to be communicated to the global transaction manager; there are two variants to do so, an optimistic and a conservative one. In the optimistic variant, there is no restriction on the order in which subtransactions take their tickets in the data servers. The global transaction manager maintains a global ticket graph that represents the ordering in which distributed transactions take tickets. This graph contains an edge $t_i \rightarrow t_j$ if and only if both distributed transactions have subtransactions in one data server, and the ticket of t_i has a lower value than the ticket of t_j at that site. In this case t_i is serialized before t_j.

On commit of a distributed transaction t, the global transaction manager checks the global ticket graph. A loop in this graph indicates a serialization violation, leading to aborting *t*. Just like in the centralized case, if there is little contention, and few conflicts are in place, the optimistic method is advantageous. However, high contention leads to many conflicts, and the probability of transaction abortions increases.

In the conservative ticketing method, the order in which tickets are taken is restricted by the global transaction manager. A subtransaction is allowed to take a ticket only if the global serialization ordering is not violated by this taking a ticket. While at run time there is additional overhead in checking whether a take-ticket operation can be permitted, the conservative approach reduces the number of abort operations. Therefore, the conservative ticketing method is advantageous in case of high contention, while the optimistic approach is likely to perform better in low contention scenarios.

Future Directions

The internet has and is continuing to pose new requirements to existing solutions in various areas of database technology, including distributed concurrency control. If applications use resources and data provided by different organizations on the internet, transactional properties are ver/y hard, if not impossible, to satisfy. As a result, new concepts and technologies are sought to provide application level consistency. As one representative of these future directions of research, the work by Pat Helland in the context of internet scale applications and their challenges is mentioned [5].

Cross-references

- ▶ ACID Properties
- ▶ Conflict Serializability
- ▶ Timestamp Ordering
- ▶ Two-Phase Commit
- ▶ Two-Phase Locking

Recommended Reading

1. Bernstein P.A. and Goodman N. Concurrency control in distributed database systems. ACM Comput Surv., 13 (2):185–221, 1981.
2. Ceri S. and Pelagatti G. Distributed Databases: Principles and Systems. McGraw-Hill, NY, USA, 1984.
3. Eswaran K.P., Gray J.N., Lorie R.A., and Traiger I.L. The notions of consistency and predicate locks in a database system. Commun. ACM, 19(11):624–633, 1976.
4. Gray J.N. The transaction concept: virtues and limitations. In Proc. 7th Int. Conf. on Very Data Bases, 1981, pp 144–154.
5. Helland P. Life beyond distributed transactions: an Apostate's opinion In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 132–141.
6. Knapp E. Deadlock detection in distributed databases. ACM Comput. Surv., 19(4):303–328, 1987.
7. Lamport L. Time, clocks, and the ordering of events in a distributed system. Commun. ACM, 21(7):558–565, 1978.
8. Özsu M.T. and Valduriez P. Principles of distributed database systems. 2nd edn. Prentice-Hall, 1999.
9. Spector A.Z. and Schwarz P.M. Transactions: a construct for reliable distributed computing. ACM Operat. Syst. Rev., 17 (2):18–35, 1983.
10. Weikum G. and Vossen G. Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, 2002.

Distributed Data Streams

MINOS GAROFALAKIS
Technical University of Crete, Chania, Greece

Definition

A majority of today's data is constantly evolving and fundamentally distributed in nature. Data for almost any large-scale data-management task is continuously collected over a wide area, and at a much greater rate than ever before. Compared to traditional, centralized stream processing, querying such large-scale, evolving data collections poses new challenges, due mainly to the physical distribution of the streaming data and the communication constraints of the underlying network. Distributed stream processing algorithms should guarantee efficiency not only in terms of *space* and *processing time* (as conventional streaming techniques), but also in terms of the *communication load* imposed on the network infrastructure.

Historical Background

The prevailing paradigm in database systems has been understanding the management of *centralized* data: how to organize, index, access, and query data that is held centrally on a single machine or a small number of closely linked machines. Work on parallel and distributed databases has focused on different notions of consistency and methods for effectively distributing query execution plans over multi-node architectures – the issues of monitoring or querying distributed, high-speed data streams in a space-, time- and communication-efficient manner were not addressed in this realm. Similarly, the bulk of early research on data-streaming algorithms and systems has concentrated on a *centralized* model of computation, where the stream-processing engine has direct access to all the streaming data records. Centralized stream-processing models can obviously ignore communication-efficiency issues; still, such models are also painfully inadequate for many of the prototypical data-streaming applications, including IP-network and sensornet monitoring.

Foundations

Tracking and querying large-scale, evolving data collections poses a number of challenges. First, in contrast with conventional, centralized models of data-stream processing, the task is inherently *distributed*; that is, the underlying infrastructure comprises several remote sites (each with its own local data source) that can exchange information through a communication network. This also means that there typically are important *communication constraints* owing to either network-capacity restrictions (e.g., in IP-network monitoring, where the volumes of collected utilization and traffic data can be huge [7]), or power and bandwidth restrictions (e.g., in wireless sensor networks, where communication overhead is the key factor in determining sensor battery life [18]). Second, each remote site may see a *high-speed stream* of data and, thus, must solve a local (centralized) stream-processing problem within its own local resource limitations, such as *space* or *CPU-time* constraints. This is certainly true for IP routers (that cannot possibly store the log of all observed packet traffic at high network speeds), as well as wireless sensor nodes (that, even though may not observe large data volumes, typically have very little memory on-board). Finally, applications often require *continuous monitoring* of the underlying streams (i.e., real-time tracking of measurements or events), not merely one-shot responses to sporadic queries.

To summarize, the focus is on techniques for processing queries over collections of remote data streams. Such techniques have to work in a distributed setting (i.e., over a communication network), support one-shot or continuous query answers, and be space, time, and communication efficient. It is important to note that, for most realistic distributed streaming applications, the naive solution of collecting all the data in a single location is simply *not* a viable option: the volume of data collection is too high, and the capacity for data communication relatively low. Thus, it becomes critical to exploit local processing resources to effectively minimize the burden on the communication network. This establishes the fundamental concept of "*in-network processing*:" if more computational work can be done *within* the network to reduce the communication needed, then it is possible to significantly improve the value of the network, by increasing its useful life and communication capacity, and extending the range of computations possible over the network. This is a key idea that permeates the bulk of existing work on distributed data-stream processing – this work can, in general, be characterized along three (largely orthogonal) axes:

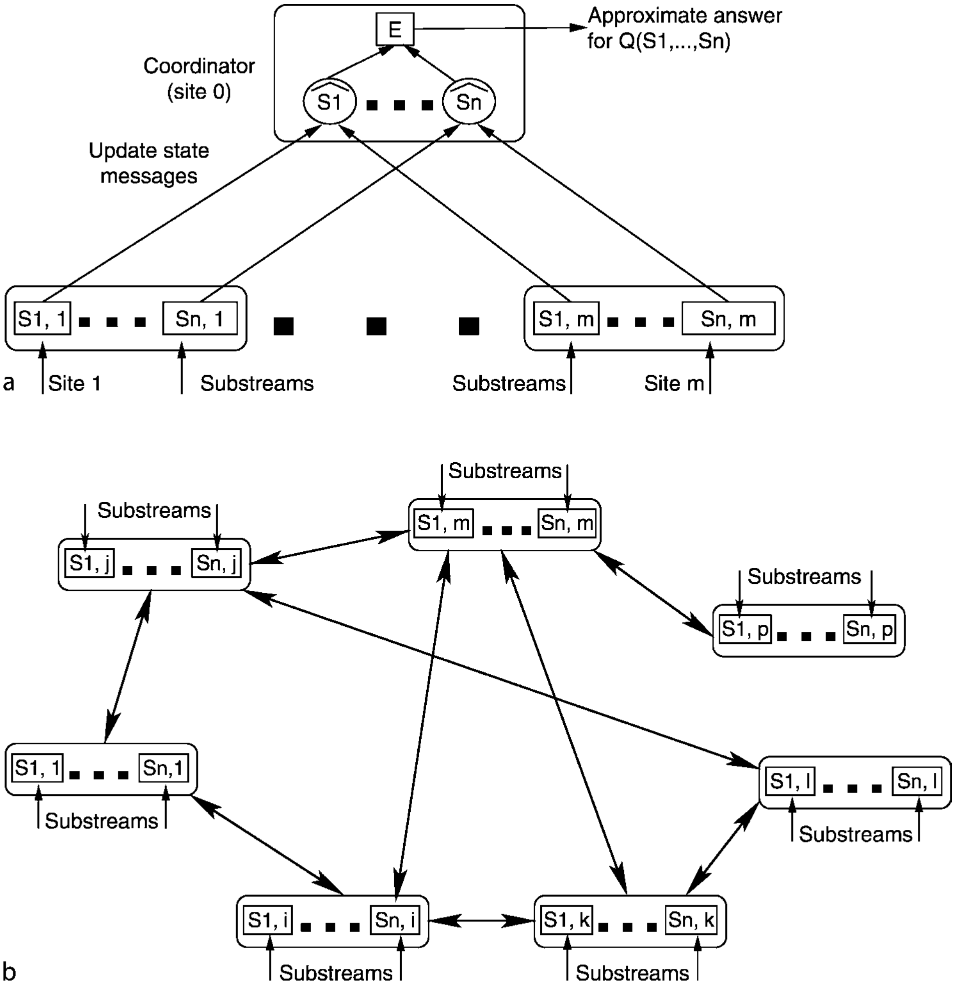
1. *Querying Model:* There are two broad classes of approaches to in-network query processing, by analogy to types of queries in traditional DBMSs. In the *one-shot* model, a query is issued by a user at some site, and must be answered by "pulling" the current state of data in the network. For simple aggregates, this can be done in a few rounds of communication where only small, partial-aggregate messages are exchanged over a spanning tree of the network. For more complex, *holistic* aggregates (that depend on the complete data distribution, such as quantiles, topk-*k*, count-distinct, and so on), simple combination of partial results is insufficient, and instead clever composable summaries give a compact way to accurately approximate query answers.

In the *continuous* model, users can register a query with the requirement that the answer be tracked continuously. For instance, a special case of such a continuous query is a *distributed trigger* that must fire in (near) real-time when an aggregate condition over a collection of distributed streams is satisfied (e.g., to catch anomalies, SLA violations, or DDoS attacks in an ISP network). This continuous monitoring requirement raises further challenges, since, even using tree computation and summarization, it is still too expensive to communicate every time new data is received by one of the remote sites. Instead, work on continuous distributed streams has focused on "push-based" techniques that tradeoff result accuracy for reduced communication cost, by apportioning the error in the query answer across *filter* conditions installed locally at the sites to reduce communication.

Approximation and *randomization* techniques are also essential components of the distributed stream querying model, and play a critical role in minimizing communication. Approximate answers are often sufficient when tracking the statistical properties of large-scale distributed systems, since the focus is typically on indicators or patterns rather than precisely-defined events. This is a key observation, allowing for techniques that effectively tradeoff efficiency and approximation accuracy.

2. *Communication Model:* The architecture and characteristics of the underlying communication network have an obvious impact on the design of effective distributed stream processing techniques. Most existing work has focused on *hierarchical* (i.e., tree) network architectures, due to both their conceptual simplicity and their importance for practical scenarios (e.g., sensornet routing trees [18]). As an example, Fig. 1a depicts a simple *single-level* hierarchical model with $m + 1$ sites and n (distributed) update streams. Stream updates arrive continuously at the remote sites 1,...,m, whereas site 0 is a special *coordinator* site that is responsible for generating answers to (one-shot or continuous) user queries Q over the n distributed streams. In this simple hierarchical model, the m remote sites do not communicate with each other; instead, as illustrated in Fig. 1a, each remote site exchanges messages only with the coordinator, providing it with state information for (sub)streams observed locally at the site.

More general, *multi-level* hierarchies have individual substream-monitoring sites at the leaves and internal nodes of a general *communication tree*, and the goal is to effectively answer or track a stream query $Q(S_1,\ldots,S_n)$ at the *root node* of the tree. The most general setting are *fully-distributed* models, where individual monitor sites are connected through an arbitrary underlying *communication network* (Fig. 1b);



Distributed Data Streams. Figure 1. (a) Single-level hierarchical stream-processing model. (b) Fully-distributed model.

this is a distinctly different distributed system architecture since, unlike hierarchical systems, no centralized authority/coordination exists and the end goal is for all the distributed monitors to efficiently reach some form of *consensus* on the answer of a distributed stream query.

Besides the connectivity model, other important network characteristics for distributed stream processing include: the potential for broadcasting or multicasting messages to sites (e.g., over a limited radio range as in wireless sensornets), and the node/link-failure and data-loss characteristics of the supporting hardware.

3. *Class of Queries:* The key dichotomy between simple, *non-holistic* aggregate queries (e.g., MIN, SUM, AVG) and *holistic* aggregates (e.g., median) has already been discussed; clearly, holistic aggregates introduce many more challenges for efficient distributed streaming computation. Another important distinction is that between *duplicate-sensitive* aggregates (that support bag/multi-set semantics, such as median, SUM, or top-k) and *duplicate-insensitive* aggregates (that support set semantics, such as MIN or count-distinct). Finally, another important class is that of complex *correlation* queries that combine/correlate streaming data across different remote sites (e.g., through a streaming *join* computation). Such correlations can be critical in understanding important trends and making informed decisions about measurement or utilization patterns. Different classes of streaming queries typically require different algorithmic machinery for efficient distributed computation.

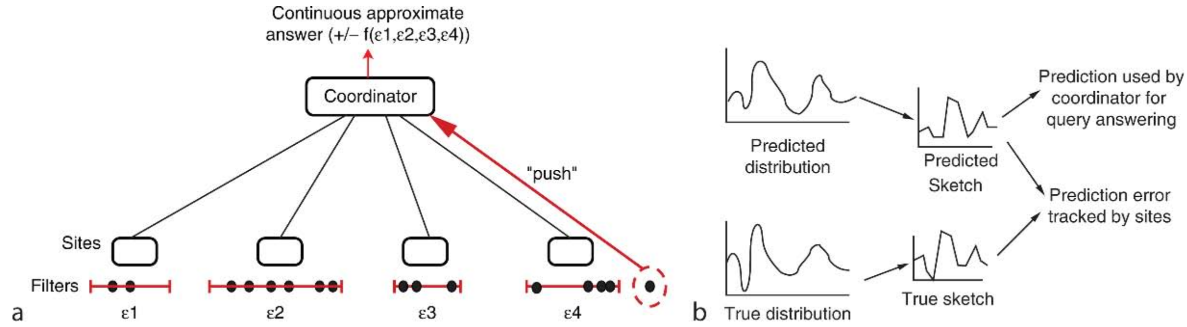
The remainder of this section provides a brief overview of some key results in distributed data streaming, for both the one-shot and continuous querying models, and concludes with a short survey of systems-related efforts in the area.

One-Shot Distributed Stream Processing. Madden et al. [18] present simple, exact *tree-based aggregation* schemes for sensor networks and propose a general framework based on *generate*, *fuse*, and *evaluate* functions for combining partial results up the aggregation tree. They also propose a classification of different aggregate queries based on different properties, such as duplicate in/sensitivity, example or summary results, monotonicity, and whether the aggregate is *algebraic* or *holistic* (which essentially translates to whether the intermediate partial state is of constant size or growing). While the exact computation of holistic aggregates requires linear communication cost, guaranteed-quality *approximate* results can be obtained at much lower cost by approximating intermediate results through *composable data synopses* [1,9].

Robustness is a key concern with such hierarchical aggregation schemes, as a single failure/loss near the root of the tree can have a dramatic effect on result accuracy. *Multi-path routing* schemes mitigate this problem by propagating partial results along multiple different paths. This obviously improves reliability and reduces the impact of potential failures; in addition, this improved reliability often comes essentially "for free" (e.g., in wireless sensornets where the network is a natural broadcast medium). Of course, multi-path routing also implies that the same partial results can be accounted for multiple times in the final aggregate. As observed by Nath et al. [20], this duplication has no effect on aggregates that are naturally *Order and Duplicate Insensitive (ODI)*, such as MIN and MAX; on the other hand, for non-ODI aggregates, such as SUM and COUNT, *duplicate-insensitive sketch synopses* (e.g., based on the Flajolet-Martin sketch [9]) can be employed to give effective, low-cost, multi-path approximations [20]. Hybrid approaches combining the simplicity of tree aggregation (away from the root node) and the robustness of multi-path routing (closer to the root) have also been explored [19].

Gossip (or, *epidemic*) protocols for spreading information offer an alternative approach for robust distributed computation in the more general, fully-distributed communication model (Fig. 1b). ODI aggregates (and sketches) naturally fit into the gossiping model, which basically guarantees that all n nodes of a network will converge to the correct global ODI aggregate/sketch after O(logn) rounds of communication. For non-ODI aggregates/sketches, Kempe et al. [15] propose a novel gossip protocol (termed *push-sum*) that also guarantees convergence in a logarithmic number of rounds, and avoids double counting by splitting up the aggregate/sketch and ensuring "conservation of mass" in each round of communication.

Continuous Distributed Stream Processing. The continuous model places a much more stringent demand on the distributed stream processing engine, since remote sites must collaborate to *continuously* maintain a query answer that is accurate (e.g., within specified error bounds) based on the current state of the stream(s). Approximation plays a critical role in the design of communication-efficient solutions for such

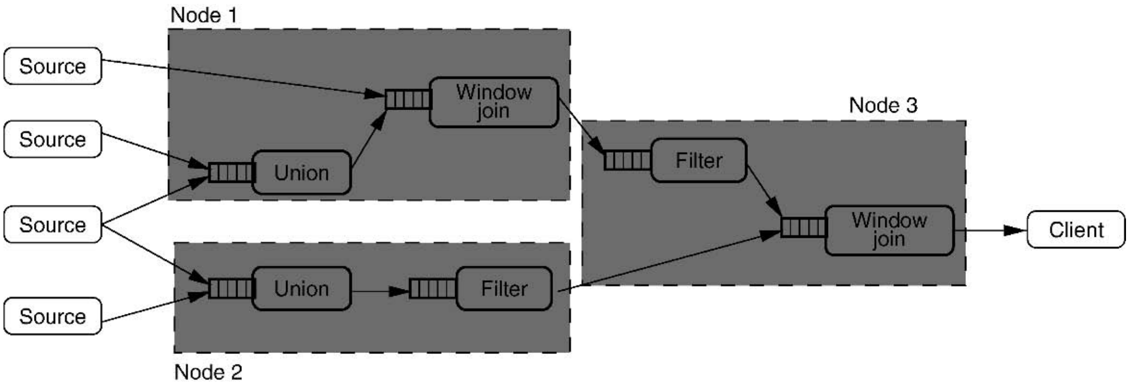

Continuous approximate answer (+/– f(ε1,ε2,ε3,ε4))
Coordinator
"push"
Sites
Filters
a
ε1
ε2
ε3
ε4
Predicted distribution
Predicted Sketch
Prediction used by coordinator for query answering
Prediction error tracked by sites
b
True distribution
True sketch

Distributed Data Streams. Figure 2. (a) Using local filters for continuous distributed query processing: Most updates fall within the local-filter ranges and require no communication with the coordinator (that can provide approximate answers with guarantees depending on the filter "widths"); only updates outside the local-filter range require new information to be "pushed" by the local site to the coordinator. (b) Prediction-based approximate query tracking: Predicted sketches are based on simple prediction models of local-stream behavior, and are kept in-sync between the coordinator (for query answering) and the remote sites (for tracking prediction error).

continuous monitoring tasks. In a nutshell, the key idea is to tradeoff result accuracy and local processing at sites for reduced communication costs, by installing *local filters* at the remote sites to allow them to only "push" significant updates to the coordinator; of course, these distributed local filters would have to be *safe*, that is, they should guarantee the overall error bound for the global query result (based on the exact current state) at the coordinator. This idea of local traffic filtering for continuous distributed queries is pictorially depicted in Fig. 2a.

A key concept underlying most continuous distributed monitoring schemes is that of *adaptive slack allocation* – that is, adaptively distributing the overall "slack" (or, error tolerance) in the query result across the local filters at different participating sites based on observed local update patterns. Obviously, the complexity of such slack-distribution mechanisms depends on the nature of the aggregate query being tracked. Olston et al. [21] consider the simpler case of algebraic aggregates (where breaking down the overall slack to safe local filters is straightforward), and discuss adaptive schemes that continuously grow/shrink local filters based on the frequency of observed local violations. As expected, the situation is more complicated in the case of holistic aggregates: Babcock and Olston [2] discuss a scheme for tracking an approximate global top-k set of items using a cleverly-built set of local constraints that essentially "align" the *local* top-k set at a site with the *global* top-k; furthermore, their algorithm also retains some amount of slack at the coordinator to allow for possible localized resolutions of constraint violations. Das et al. [8] consider the problem of monitoring distributed set-expression cardinalities and propose tracking algorithms that take advantage of the set-expression semantics to appropriately "charge" updates arriving at the local sites.

Simple slack-allocation schemes are typically based on a naive *static model* of local-site behavior; that is, the site's "value" is assumed constant since the last update to the coordinator, and communication is avoided as long as this last update value stays within the slack bounds. Cormode and Garofalakis [5] propose the use of more sophisticated, *dynamic prediction models* of temporal site dynamics in conjunction with appropriate sketching techniques for communication-efficient monitoring of complex distributed aggregate queries. Their idea is to allow each site and the coordinator to share a prediction of how the site's local stream(s) (and, their sketch synopses) evolve over time. The coordinator uses this prediction to provide continuous query answers, while the remote site checks locally that the prediction stays "close" to the actual observed streaming distribution (Fig. 2b). Of course, using a more sophisticated prediction model can also impose some additional communication to ensure that the coordinator's view is kept in-sync with the up-to-date local stream models (at the remote sites). Combined with intelligent sketching techniques and methods for bounding the overall query error, such approaches can be used to track a large class of complex, holistic queries, only requiring concise communication



Distributed Data Streams. Figure 3. Distributed stream-processing dataflow.

exchanges when prediction models are no longer accurate [5]. Furthermore, their approach can also be naturally extended to multi-level hierarchical architectures. Similar ideas are also discussed by Chu et al. [4] who consider the problem of *in-network probabilistic model maintenance* to enable communication-efficient approximate tracking of sensornet readings.

A common feature of several distributed continuous monitoring problems is continuously evaluating a condition over distributed streaming data, and *firing* when the condition is met. When tracking such *distributed triggers*, only values of the "global" continuous query that are above a certain threshold are of interest (e.g., fire when the total number of connections to an IP destination address exceeds some value) [13]. Recent work has addressed versions of this distributed triggering problem for varying levels of complexity of the global query, ranging from simple counts [16] to complex functions [26] and matrix-analysis operators [12]. Push-based processing using local-filter conditions continues to play a key role for distributed triggers as well; another basic idea here is to exploit the threshold to allow for even more effective local traffic filtering (e.g., "wider" yet safe filter ranges when the query value is well below the threshold).

Systems and Prototypes. Simple, algebraic in-network aggregation techniques have found widespread acceptance in the implementation of efficient sensornet monitoring systems (e.g., TAG/TinyDB [18]). On the other hand, more sophisticated approximate in-network processing tools have yet to gain wide adoption in system implementations. Of course, Distributed Stream-Processing Engines (DSPEs) are still a nascent area for systems research: only a few research prototypes are currently in existence (e.g., Telegraph/TelegraphCQ [25], Borealis/Medusa [3], P2 [17]). The primary focus in these early efforts has been on providing effective system support for *long-running stream-processing dataflows* (comprising connected, pipelined query operators) over a distributed architecture (Fig. 3). For instance, Balazinska et al. [3] and Shah et al. [25] discuss mechanisms and tools for supporting parallel, highly-available, fault-tolerant dataflows; Loo et al. [17] propose tools for declarative dataflow design and automated optimizations; Pietzuch et al. [22] consider the problem of distributed dataflow operator placement and propose techniques based on a cost-space representation that optimize for network-efficiency metrics (e.g., bandwidth, latency); finally, Xing et al. [27] give tools for deriving distributed dataflow schedules that are resilient to load variations in the input data streams. To deal with high stream rates and potential system overload, these early DSPEs typically employ some form of *load shedding* [3] where tuples from operators' input stream(s) are dropped (either randomly or based on different QoS metrics). Unfortunately, such load-shedding schemes cannot offer any hard guarantees on the quality of the resulting query answers. A mechanism based on *revision tuples* can be employed in the Borealis DSPE to ensure that results are *eventually correct* [3]. AT&T's Gigascope streaming DB for large-scale IP-network monitoring [7] uses approximation tools (e.g., sampling, sketches) to efficiently track "line-speed" data streams at the monitoring end-points, but has yet to explore issues related to the physical distribution of the streams and holistic queries.

Key Applications

Enterprise and ISP Network Security: The ability to efficiently track network-wide traffic patterns plays a key role in detecting anomalies and possible malicious attacks on the network infrastructure. Given the sheer volume of measurement data, continuously centralizing all network statistics is simply not a feasible option, and distributed streaming techniques are needed.

Sensornet Monitoring and Data Collection: Tools for efficiently tracking global queries or collecting all measurements from a sensornet have to employ clever in-network processing techniques to maximize the lifetime of the sensors.

Clickstream and Weblog Monitoring: Monitoring the continuous, massive streams of weblog data collected over distributed web-server collections is critical to the real-time detection of potential system abuse, fraud, and so on.

Future Directions

The key algorithmic idea underlying the more sophisticated distributed data-stream processing techniques discussed in this article is that of effectively trading off space/time *and communication* with the quality of an approximate query answer. Exploring some of the more sophisticated algorithmic tools discussed here in the context of real-life systems and applications is one important direction for future work on distributed streams; other challenging areas for future research, include:

- Extensions to other application areas and more complex communication models, e.g., monitoring P2P services over shared infrastructure (OpenDHT [23] over PlanetLab), and dealing with constrained communication models (e.g., intermittent-connectivity and delay-tolerant networks (DTNs) [14]).
- Richer classes of distributed queries, e.g., set-valued query answers, machine-learning inference models [11].
- Developing a theoretical/algorithmic foundation of distributed data-streaming models: What are fundamental lower bounds, how to apply/extend information theory, communication complexity, and distributed coding. Some initial results appear in the recent work of Cormode et al. [6].
- Richer prediction models for stream tracking: Can models effectively capture site correlations rather than just local site behavior? More generally, understand the model complexity/expressiveness tradeoff, and come up with principled techniques for capturing it in practice (e.g., using the MDL principle [24]).
- Stream computations over an *untrusted* distributed infrastructure: Coping with privacy and authentication issues in a communication/computation-efficient manner. Some initial results appear in [10].

Data Sets

Publicly-accessible network-measurement data collections can be found at the Internet Traffic Archive: (http://ita.ee.lbl.gov/), and CRAWDAD (the Community Resource for Archiving Wireless Data at Dartmouth, http://cmc.cs.dartmouth.edu/data/dartmouth.html).

Cross-references

▶ AMS Sketch
▶ Continuous Query
▶ Count-Min Sketch
▶ Data Stream
▶ Load Shedding
▶ Scheduling Strategies for Data Stream Processing
▶ Stream Models
▶ Stream Processing
▶ Stream Sampling
▶ Streaming Applications
▶ Synopsis Structure

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. In Proc. 28th Annual ACM Symp. on the Theory of Computing, 1996, pp. 20–29.
2. Babcock B. and Olston C. Distributed top-K monitoring. In Proc. 2003 ACM SIGMOD Int. Conf. on Management of Data, 2003.
3. Balazinska M., Balakrishnan H., Madden S., and Stonebraker M. Fault-tolerance in the borealis distributed stream processing system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
4. Chu D., Deshpande A., Hellerstein J.M., and Hong W. Approximate data collection in sensor networks using probabilistic models. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
5. Cormode G. and Garofalakis M. Sketching streams through the net: distributed approximate query tracking. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005.
6. Cormode G., Muthukrishnan S., and Yi K. Algorithms for distributed functional monitoring. In Proc. 19th Annual ACM-SIAM Symp. on Discrete Algorithms, 2008.
7. Cranor C., Johnson T., Spatscheck O., and Shkapenyuk V. Gigascope: a stream database for network applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
8. Das A., Ganguly S., Garofalakis M., and Rastogi R. Distributed set-expression cardinality estimation. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.

9. Flajolet P. and Nigel Martin G. Probabilistic counting algorithms for data base applications. J. Comput. Syst. Sci., 31:182–209, 1985.
10. Garofalakis M., Hellerstein J.M., and Maniatis P. Proof sketches: verifiable in-network aggregation. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
11. Guestrin C., Bodik P., Thibaux R., Paskin M., and Madden S. Distributed regression: an efficient framework for modeling sensor network data. Inform. Process. Sensor Networks, 2004.
12. Huang L., Nguyen X., Garofalakis M., Hellerstein J.M., Jordan M.I., Joseph A.D., and Taft N. Communication-efficient online detection of network-wide anomalies. In Proc. 26th Annual Joint Conf. of the IEEE Computer and Communications Societies, 2007.
13. Jain A., Hellerstein J., Ratnasamy S., and Wetherall D. A wakeup call for internet monitoring systems: The case for distributed triggers. In Proc. Third Workshop on Hot Topics in Networks, 2004.
14. Jain S., Fall K., and Patra R. Routing in a delay tolerant network. In Proc. ACM Int. Conf. of the on Data Communication, 2005.
15. Kempe D., Dobra A., and Gehrke J. Gossip-based computation of aggregate information. In Proc. 44th Annual IEEE Symp. on Foundations of Computer Science. 2003.
16. Keralapura R., Cormode G., and Ramamirtham J. Communication-efficient distributed monitoring of thresholded counts. In Proc. 2006 ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 289–300.
17. Loo B.T., Condie T., Garofalakis M., Gay D.E., Hellerstein J.M., Maniatis P., Ramakrishnan R., Roscoe T., and Stoica I. Declarative networking: language, execution, and optimization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006.
18. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. TAG: a tiny aggregation service for ad-hoc sensor networks. In Proc. 5th USENIX Symp. on Operating System Design and Implementation, 2002.
19. Manjhi A., Nath S., and Gibbons P. Tributaries and deltas: efficient and robust aggregation in sensor network streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
20. Nath S., Gibbons P.B., Seshan S., and Anderson Z.R. Synopsis diffusion for robust aggrgation in sensor networks. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems. 2004.
21. Olston C., Jiang J., and Widom J. Adaptive filters for continuous queries over distributed data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
22. Pietzuch P., Ledlie J., Schneidman J., Roussopoulos M., Welsh M., and Seltzer M. Network-aware operator placement for stream-processing systems. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
23. Rhea S., Godfrey B., Karp B., Kubiatowicz J., Ratnasamy S., Shenker S., Stoica I., and Yu H.Y. OpenDHT: a public dht service and its uses. In Proc. ACM Int. Conf. of the on Data Communication, 2005.
24. Rissanen J. Modeling by shortest data description. Automatica, 14:465–471, 1978.
25. Shah M.A., Hellerstein J.M., and Brewer E. Highly available, fault-tolerant, parallel dataflows. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
26. Sharfman I., Schuster A., and Keren D. A geometric approach to monitoring threshold functions over distributed data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 301–312.
27. Xing Y., Hwang J.-H., Cetintemel U., and Zdonik S. Providing resiliency to load variations in ditributed stream processing. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.

Distributed Database Design

Kian-Lee Tan
National University of Singapore, Singapore, Singapore

Synonyms

Horizontal fragmentation; Vertical fragmentation; Data replication

Definition

Distributed database design refers to the following problem: given a database and its workload, how should the database be split and allocated to sites so as to optimize certain objective function (e.g., to minimize the resource consumption in processing the query workload). There are two issues: (i) Data fragmentation which determines how the data should be fragmented; and (ii) Data allocation which determines how the fragments should be allocated. While these two problems are inter-related, the two issues have traditionally been studied independently, giving rise to a two-phase approach to the design problem.

The design problem is applicable when a distributed database system has to be built from scratch. In the case when multiple existing databases are to be integrated (e.g., in multi-database context), there is no design issue.

Historical Background

In a distributed database system, relations are typically fragmented and stored at multiple sites. Fragmentation of a relation is useful for several reasons. First, an application typically accesses only subsets of relations. Moreover, different subsets are naturally needed at different sites. As such, fragmenting a relation to facilitate locality of accesses of applications can improve performance (otherwise, the overhead of shipping relations from one site to another may be unnecessarily high). Second, as applications may operate on different fragments, the degree of concurrency is thus enhanced. Even within a single application, accessing multiple fragments that are located at different sites essentially facilitates parallelism. Third, for applications that require multiple fragments from different relations, these fragments can be colocated to minimize communication overhead. However, like normalization, decomposing a relation to fragments may lead to poorer performance when multiple fragments need to be joined. In addition, if two attributes with a dependency are split across fragments, then it becomes more costly to enforce the dependency. Most of the works on fragmentation were done in the early 1980s [4,5,11,12,13]. One of the fundamental task in vertical fragmentation is to identify attributes that should be grouped together. In [11,9], the bond energy algorithm (BEA) was first proposed to cluster together attributes with high affinity for each other; based on this, the relation is then partitioned accordingly.

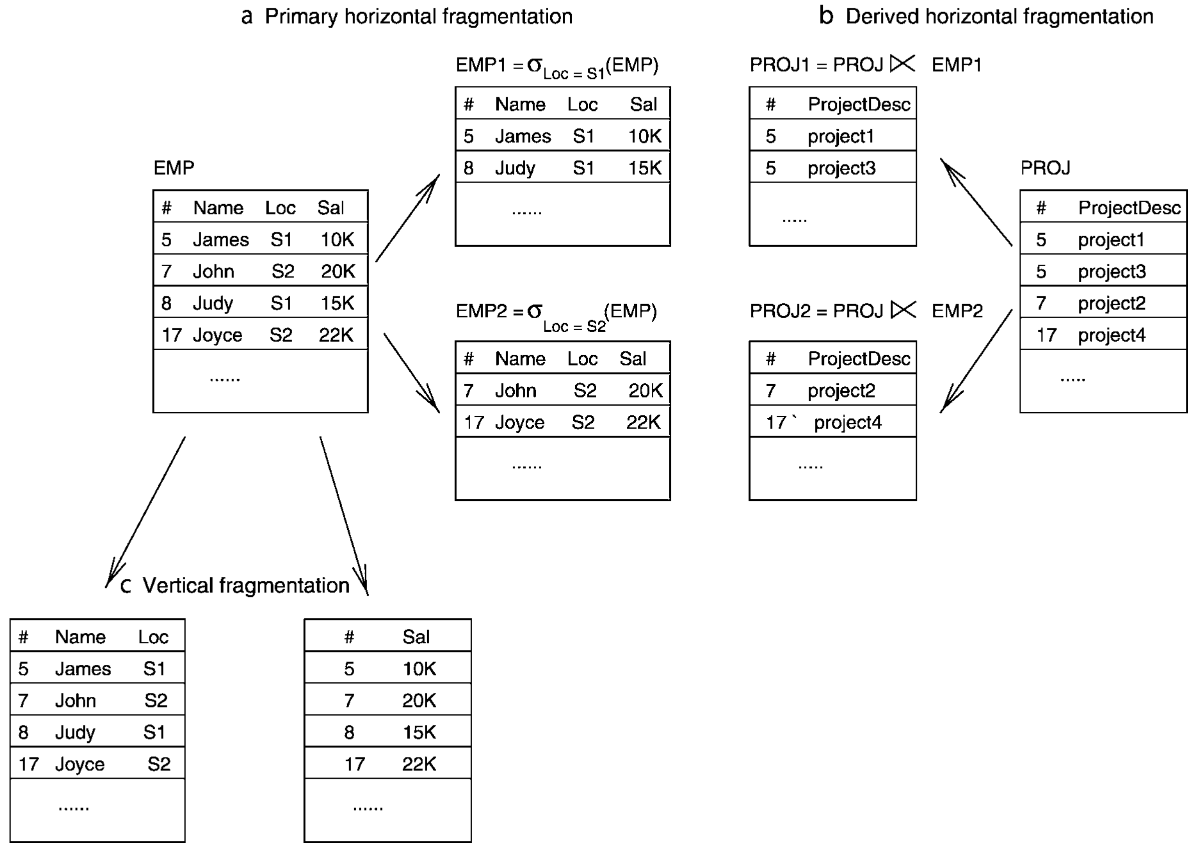
The allocation problem received much more attention. Initial work dated back to as early as 1969 where the file allocation problem was investigated [7]. In [1,2,6], the data allocation problem was shown to be NP-hard. In a dynamic environment, the workload and access pattern may change. In [3,8], dynamic data allocation algorithms were studied. These techniques change the initial data allocation to adapt to changing access patterns and workload.

There were also several works that combine both fragmentation and allocation into an integrated solution [13,14].

Foundations

Fragmentation

In a distributed database system, a relation R may be split in a number of fragments $F = \{R_1, R_2, \ldots, R_n\}$ in such a way that R can be reconstructed from them. There are essentially three fragmentation schemes.


a Primary horizontal fragmentation
b Derived horizontal fragmentation
c Vertical fragmentation

EMP

#	Name	Loc	Sal
5	James	S1	10K
7	John	S2	20K
8	Judy	S1	15K
17	Joyce	S2	22K
......			

EMP1 = $\sigma_{Loc = S1}$(EMP)

#	Name	Loc	Sal
5	James	S1	10K
8	Judy	S1	15K
......			

EMP2 = $\sigma_{Loc = S2}$(EMP)

#	Name	Loc	Sal
7	John	S2	20K
17	Joyce	S2	22K
......			

PROJ

#	ProjectDesc
5	project1
5	project3
7	project2
17	project4
.....	

PROJ1 = PROJ $\ltimes$ EMP1

#	ProjectDesc
5	project1
5	project3
.....	

PROJ2 = PROJ $\ltimes$ EMP2

#	ProjectDesc
7	project2
17 `	project4
.....	

#	Name	Loc
5	James	S1
7	John	S2
8	Judy	S1
17	Joyce	S2
......		

#	Sal
5	10K
7	20K
8	15K
17	22K
......	

Distributed Database Design. Figure 1. Fragmentation schemes.

In *primary horizontal* fragmentation, each fragment is essentially a subset of the tuples in the original relation (see Fig. 1a). In general, a fragment is defined as a selection on the original relation, i.e., $R_i = \sigma_{C_i}(R)$ where C_i is a predicate used to construct R_i. A good primary horizontal fragmentation of a relation typically has three desirable properties:

1. *Completeness:* $\forall t \in R, \exists R_i \in F$ such that $t \in R_i$. This property states that every tuple in the original relation must be assigned to a fragment.
2. *Disjointness:* $\forall t \in R_i, \nexists R_j$ such that $t \in R_j$, $i \neq j$, R_i, $R_j \in F$. This property states that all fragments are disjoint, i.e., a tuple is assigned to only one fragment. This property is useful as it leaves the decision to replicate fragments to the allocation phase.
3. *Reconstruction:* $R = \bigcup_{i=1}^{n} R_i$. This property states that the original relation can be reconstructed by simply union-ing all its fragments.

In *derived horizontal fragmentation*, a relation S is fragmented based on the fragments of another relation R. R is called the owner relation and S the member relation. Each fragment of S is obtained using a semi-join between S and a corresponding fragment of R, i.e., $\mathrm{S_i} = \mathrm{S} \ltimes_{S.k=R_i.k} R_i$ where k is the join attribute (see Fig. 1b). Derived fragmentation is useful when the fragments of R need to be combined with the records of S with matching join keys. As such, the corresponding derived fragment is the necessary subset, and can be co-located with the R fragment. In order to ensure completeness, referential integrity constraint has to be enforced, i.e., the set of distinct join attribute values of the member relation must be a subset of the set of distinct join attribute values of the owner relation. Moreover, for disjointness property to be achievable, the join attribute should be the key of the owner relation.

In *vertical fragmentation*, a relation $R[T]$ (where T is the set of attributes in R's schema) is decomposed into, say k fragments $R_1[T_1]$, $R_2[T_2]$,...$R_k[T_k]$ (where T_i is the set of attributes in fragment R_i) (see Fig. 1c). Each fragment R_i is defined as the projection of the ordinal relation on the set of attributes T_i, i.e., $R_i = \pi_{T_i}(R)$. For completeness, $T = \cup_{i=1}^{k} T_i$. In addition, to reconstruct R from its fragments, the lossless join property must be enforced, i.e., $R = \bowtie_{i=1}^{k} R_i$. One way to achieve the lossless join property is to repeat the key attributes in all the fragments, i.e., $\forall i$, $key \subseteq T_i$.

In practice, a combination of all the above fragmentation schemes can be applied on a single table. For example, the EMP table can be first horizontally partitioned based on the location, and then the fragment at location S1 (EMP1) may be further vertically partitioned into $P1 = \pi_{\#,\ Name,\ Loc} EMP1$ and $P2 = \pi_{\#,\ Sal} EMP1$. Now a query over the original relation EMP can then be evaluated as a query over the corresponding fragments (cross-reference Distributed Query Processing). For example, a query to find the names of employees in location S1, expressed as $\pi_{Name}\sigma_{Loc=S1}EMP$, can be reduced to a query over one fragment: $\pi_{Name}P1$. As can be seen, fragmentation can lead to more efficient query processing.

To generate the set of fragments, the following strategy can be adopted. Given a set of simple predicates $P = \{P_1, P_2, ..., P_m\}$ (each P_i is of the form `attribute` θ `value` where $\theta \in \{<, \leq, =, >, \geq\}$), a set of minterm predicates M is generated. M is defined as follows:

$$M = \{m | m = \wedge_{P_k \in P} P_k{}^*, 1 \leq k \leq m\}$$

where $P_k{}^*$ is P_k or $\neg P_k$. After eliminating useless minterm prediates, the resultant minterm predicates can be used to produce the set of disjoint fragments: $\sigma_m(R)$ for all $m \in M$. As an example, suppose $P = \{A < 10, A > 5, Loc = S_1, Loc = S_2\}$ is the set of simple predicates. Moreover, assume that there are only 2 locations, S_1 and S_2. Then, there will be a total of 16 minterm predicates; several of these are empty set, e.g., $\{A < 10 \wedge A > 5 \wedge Loc = S_1 \wedge Loc = S_2\}$ and $\{A < 10 \wedge \neg A > 5 \wedge Loc = S_1 \wedge Loc = S_2\}$. The resultant set of minterm predicates consists of 6 predicates: $\{5 < A < 10 \wedge Loc = S_1\}$, $\{5 < A < 10 \wedge Loc = S_2\}$, $\{A \leq 5 \wedge Loc = S_1\}$, $\{A \leq 5 \wedge Loc = S_2\}$, $\{A \geq 10 \wedge Loc = S_1\}$, $\{A \geq 10 \wedge Loc = S_2\}$. Each of these predicates will result in a fragment.

Allocation

Once a database has been fragmented, the fragments are allocated to the sites. This gives rise to the allocation problem: Given a set of fragments $F = \{F_1, F_2, ..., F_n\}$ and a number of sites $S = \{S_1, S_2, ..., S_m\}$ on which a number of applications $Q = \{Q_1, Q_2, ..., Q_p\}$ is running, allocate $F_i \in F$ to $S_j \in S$ such that some optimization criterion is met (subject to certain constraints, e.g., available storage at certain sites). Some optimization criteria include maximizing throughput, minimizing the average response time or minimizing the total cost of serving *Q*. This (allocation) problem is complex because the optimal solution depends on many factors, for example, the location in which a query originates, the query processing strategies (e.g., join methods) that are used, the hardware at the various sites and so on. As such, for the problem to be tractable, the problem is typically simplified with certain assumptions, e.g., only communication cost is considered.

As an example, consider the following simple cost models which determine the read, write and storage cost of an arbitrary fragment *f*. (Note that a more complex model will need to consider other factors like fragment size, queries involving multiple fragments, and so on.). The read cost of *f* is given by:

$$\sum_{i=1}^{m} t_i \times MIN_{j=1}^{m} C_{ij}$$

where i is the originating site of the request, t_i is the read traffic at S_i and C_{ij} is the cost to access fragment f (stored) at S_j from S_i. If only the transmission cost is considered, then $C_{ii} = 0$ if f is stored at site S_i, and $C_{ij} = \infty$ if f is not stored at site S_j. The update cost is given by:

$$\sum_{i=1}^{m} \sum_{j=1}^{m} X_j \cdot u_i \cdot C_{ij}$$

where i is the originating site of the request, j is the site being updated, $X_j = 1$ if f is stored at S_j and 0 otherwise, u_i is the write traffic at S_i and C_{ij} is the cost to update f at S_j from S_i. Finally, the storage cost is given by:

$$\sum_{i=1}^{m} X_i \cdot d_i$$

where $X_i = 1$ if f is stored at S_j and 0 otherwise, and d_i is the storage cost at S_i.

Fragments can be allocated to minimize a combination of the above costs. Based on these cost models, any optimization algorithm can be easily adapted to solve it. For example, a randomized algorithm (such as simulated annealing, iterative improvement) can allocate fragments to sites and the allocation that gives the best cost is the winner.

Heuristics have also been developed. For example, a best-fit heuristic for non-replicated allocation works as follows: For each fragment *f*, place it at the site *j* where the total cost (for this fragment only) is minimum. While this scheme is computationally efficient, it ignores the effect of other fragments which may render the allocation sub-optimal.

Future Directions

Even though the distributed database design has been extensively studied, similar data partitioning and placement (allocation) issues continue to be surfaced for new architectural design. The focus here will be more on load-balancing and dynamic data migration. For example, in a highly dynamic and loosely connected distributed systems like a peer-to-peer system, there is yet no effective solution to dynamically adapt the placement of data for optimal performance. Similarly, in a distributed publish/subscribe environment, the problem of migrating data (subscriptions) for optimal performance has not been adequately explored. Moreover, in these systems, the logical network connection between sites may be dynamically changing, and this will influence the allocation of data.

Cross-references

► Data Replication
► Parallel Database Management
► Peer Data Management System

Recommended Reading

1. Apers P.M. Data Allocation in distributed database systems. ACM Trans. Database Syst., 13(2):263–304, 1988.
2. Bell D.A. Difficult data placement problems. Comput. J., 27(4):315–320, 1984.
3. Brunstrom A., Leutenegger S.T., and Simha R. Experimental evaluation of dynamic data allocation strategies in a distributed database with changing workloads. In Proc. Int. Conf. on Information and Knowledge Management, 1995, pp. 395–402.
4. Ceri S., Negri M., and Pelagatti G. Horizontal data partitioning in database design. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1982, pp. 128–136.
5. Ceri S. and Pelagatti G. Distributed Databases: Principles and Systems. McGraw-Hill, NY, USA, 1984.
6. Chang C.C. and Shieh J.C. On the complexity of file allocation problem. In Proc. Int. Conf. on the Foundations of Data Organization, 1985, pp. 177–181.
7. Chu W.W. Optimal file allocation in a multiple computer network. IEEE Trans. Comput., C-18(10):885–889, 1969.
8. Karlapalem K. and Ng M.P. Query-driven data allocation algorithms for distributed database systems. In Proc. 8th Int. Conf. Database and Expert Syst. Appl., 1997, pp. 347–356.
9. McCormick W.T., Schweitzer P.J., and White T.W. Problem decomposition and data reorganization by a clustering techniqiue. Oper. Res., 20(5):993–1009, 1972.

10. Muri S., Ibaraki T., Miyajima H., and Hasegawa T. Evaluation of file redundancy in distributed database systems. IEEE Trans. Software Eng., 11(2):199–205, 1995.
11. Navathe S., Ceri S., Wiederhold G., and Dou J. Vertical partitioning of algorithms for database design. ACM Trans. Database Syst., 9(4):680–710, 1984.
12. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 2nd edn. Prentice-Hall, 1999.
13. Sacca D. and Wiederhold G. Database partitioning in a cluster of processors. ACM Trans. Database Syst., 10(1):29–56, 1985.
14. Yoshida M., Mizumachi K., Wakino A., Oyake I., and Matsushita Y. Time and cost evaluation schemes of multiple copies of data in distributed database systems. IEEE Trans. Software Eng., 11(9):954–958, 1985.

Distributed Database Management System (DDBMS)

► Distributed DBMS

Distributed Database Systems

Kian-Lee Tan
National University of Singapore, Singapore, Singapore

Synonyms

Homogeneous distributed database systems; Heterogeneous distributed database systems; Federated database systems; Multidatabases

Definition

A distributed database (DDB) is an integrated collection of databases that is physically distributed across sites in a computer network. A distributed database management system (DDBMS) is the software system that manages a distributed database such that the distribution aspects are transparent to the users. To form a distributed database system (DDBS), the files must be structured, logically interrelated, and physically distributed across multiple sites. In addition, there must be a common interface to access the distributed data.

Historical Background

There are many reasons that motivated DDBS. First, distributed databases reflect organizational structure. In many organizations, the data are naturally distributed across departments/branches where each department/branch maintains its own local data. Moreover, it is not always possible to build a centralized system to consolidate these data. In addition, by keeping these data at their respective remote sites facilitates autonomy where each site retains control over the data that it generates/possesses.

Next, a DDBS is expected to offer better performance – data are placed at locations where they are frequently accessed, and hence communication overhead can be minimized; moreover, parallelism can be exploited to process a query in parallel. Had data been stored in a centralized site, the centralized site may become a bottleneck, and the communication overhead may be significant.

A DDBS also offers better availability – when a site fails, the other operational sites can potentially still be available for data retrieval and query processing. A centralized site is vulnerable to single point of failure.

The concepts behind distributed DBMS were pioneered during the late 1970s through several research projects including SDD-1 [8] developed by Computer Corporation of America, Distributed INGRES [11] started at the University of California at Berkeley, and R*STAR [9] designed at IBM research lab.

The first well-publicized distributed DBMS product was INGRES/Star, announced in 1987. Oracle also announced its distributed DBMS capabilities in 1987, and the first Oracle product to reasonably support distributed database processing is Oracle 7. IBM's distributed DBMS products, based on the distributed relational data architecture, are largely systems for integrating data sets across the different versions of DB2 that run on AIX, OS/2, OS/400, VM and MVS.

More recent trends have focused on multi-databases (heterogeneous databases) and distributed systems that offer more autonomy to individual system [10,12].

Foundations

In a distributed database system, data are distributed across a number of sites [1,6]. A relation can be horizontally or vertically fragmented (cross-reference Distributed Database Design), and/or replicated. These fragments/replicas are allocated to the sites to be stored there. In a multi-database, where multiple databases are to be integrated, one can view the local database as a fragment of the integrated database.

One key consideration in the design of a DDBS is the notion of data transparency. With data transparency, the user accesses the database thinking that (s)he is working with one logical centralized database. There are several forms of transparency: (i) distribution transparency; (ii) replication transparency; (iii) Location transparency; and (d) Transaction transparency. In distribution transparency, the user is not aware of how a relation has been fragmented. In replication transparency, the user sees only one logically unique piece of data. (S)he is masked from the fact that some fragments may be replicated and that replicas reside at different locations.

In location transparency, the user is masked from the use of the location information when issuing the query. In other words, there is no need for the user to specify the location of data that (s)he is retrieving. This follows from the distribution/replication transparencies. In transaction transparency, the user is masked from coordination activities required to achieve consistency when processing queries. In fact, each transaction maintains database consistency and integrity across the multiple databases/replicas (without user knowledge). A global transaction that accesses data from multiple sites has to be divided into subtransactions that process data at one single site.

A DDBMS comprises a number of components [2]: (i) The database management component is a centralized DBMS that manages the local database; (ii) The data communication component handles the communication between sites in a DDBS; (iii) The data dictionary which is extended to represent information about the distribution of data (and sites) in the DDBS; (iv) the distributed database component (DDB) that manages the DDBS to ensure that the system functions correctly.

In particular, the DDB performs a number of tasks. First, in processing a global transaction, the DDB needs to determine a distributed query plan that is most cost-effective. Distributed query processing algorithms including semijoins are often used to reduce the communication overhead.

Second, there is a need to synchronize the accesses from multiple users to the distributed databases in order to maintain the consistency and integrity of the database. Unlike centralized database systems, the database consistency has to be guaranteed at each site as well as across all sites.

Third, as a global transaction may require accessing data from multiple sites, the competition for these data can result in deadlocks (for locking-based synchronization mechanisms, which is the most commonly used). In a DDBS, deadlocks may not occur within a site, but occur across sites. As such, having a "global" view is critical in order to detect and to recover from deadlocks.

Fourth, atomicity of global transactions is an important issue. Clearly, if a subtransaction running at one site commits while another substransaction running at another site aborts, then the database will become inconsistent. As such, distributed transaction commit protocols must ensure that such a situation cannot arise. The two-phase commit protocol is the most widely used.

Finally, as in any distributed systems, sites may fail or become inaccessible as a result of network problems. As a result, it is necessary to ensure the atomicity of global transactions in the midst of failures. Moreover, mechanisms must be provided to ensure the consistency of the database, and to bring the recovered failed nodes up-to-date.

While DDBS offers a host of promises and advantages, it is very difficult to realize a full-fledge DDBS. For example, it has been argued that full transparency makes the management of distributed data very difficult so much so that it can lead to poor manageability, poor modularity and poor message performance [3]. As another example, it is clearly more complex to design the DDB component and the respective mechanisms and protocols for the DDBS to operate as promise and to ensure consistency and reliability. Being connected to a network also means that it is more vulnerable to security threats. Nevertheless, it is an exciting field that offers many challenges for researchers to work on.

Key Applications

Many of today's applications are naturally distributed. For example, in supply-chain management, there is a need to access information concerning parts, products, suppliers which are housed by different organizations. As another example, in financial applications, a bank typically has many branches locally or overseas, and each such branch maintains its own databases to better serve their local customers; there is, however, a need to have an integrated view of the bank as well. Other applications include airline reservation, government agencies, health care, and so on.

Future Directions

Distributed databases continue to be an interesting area of research. In particular, the internet has made large scale distributed systems possible and practical. There are still many problems that have not been adequately addressed. For example, it remains a challenge to find the optimal query optimization plan especially when site autonomy are to be enforced. Some recent works have examined these using microeconomics principles [7]. Another direction is the semantic interoperability problem [4]. Yet another direction is in the design of advanced distributed data management systems, e.g., peer-based data management [5] (where peers are highly dynamic and may join and leave the network anytime), mobile data management (where nodes are mobile), and sensornet databases (where sensor nodes are battery-powered, has limited storage and processing capabilities).

Cross-references

- ► Data Replication
- ► Distributed Architecture
- ► Distributed Database Design
- ► Distributed Query Optimization
- ► Distributed Query Processing
- ► Parallel Database Management
- ► Peer Data Management System

Recommended Reading

1. Bell D. and Grimson J. Distributed Database Systems. Addison-Wesley, 1992.
2. Ceri S. and Pelagatti G. Distributed Databases: Principles and Systems. McGraw-Hill, 1984.
3. Gray J. Transparency in Its Place – The Case Against Transparent Access to Geographically Distributed Data. Technical Report TR89.1, Cupertino, Calif.: Tandem Computers Inc., 1989.
4. Halevy A.Y., Ives Z.G., Suciu D., and Tatarinov I. Schema mediation for large-scale semantic data sharing. VLDB J., 14(1):68–83 2005.
5. Ng W.S., Ooi B.C., Tan K.L., and Zhou A. PeerDB: A P2P-based System for Distributed Data Sharing. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 633–644.
6. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 2nd edn. Prentice-Hall, 1999.
7. Pentaris F. and Ioannidis Y.E. Query optimization in distributed networks of autonomous database systems. ACM Trans. Database Syst., 31(2):537–583, 2006.
8. Rothnie Jr., Bernstein P.A., Fox S., Goodman N., Hammer M., Landers T.A., Reeve C.L., Shipman D.W., and Wong E. Introduction to a system for distributed databases (SDD-1). ACM Trans. Database Syst., 5(1):1–17, 1980.
9. Selinger P.G. An architectural overview of R*: a distributed database management system. In Proc. 5th Berkeley Workshop on Distributed Data Management and Computer Networks, 1981, p. 187.
10. Sheth A.P. and Larson J.A. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv., 22(3):183–236., 1990.
11. Stonebraker M. The Design and Implementation of Distributed INGRES. The INGRES Papers, 1986, pp. 187–196.
12. Stonebraker M., Aoki P.M., Pfeffer A., Sah A., Sidell J., Staelin C., and Yu A. Mariposa: a wide-area distributed database system. VLDB J., 5(1):48–63, 1996.

Distributed Databases

► Storage Grid

Distributed DBMS

Sameh Elnikety
Microsoft Research, Cambridge, UK

Synonyms

Distributed Database Management System (DDBMS)

Definition

A distributed DBMS is a software system that manages a distributed database, which consists of data that are partitioned and replicated among interconnected server sites. The primary objective of a distributed DBMS is to hide data distribution so that it appears as one logical database system to the clients.

Historical Background

Distributed DBMS started in the late 1970s [2,10,12] with shared-nothing parallel database systems [13], which were designed for achieving higher performance by exploiting parallelism in transaction workloads. Work on distributed DBMS was mainly motivated by the need to manage data for large organizations having different offices and subsidiaries but slow computer networks hampered the adoption of DDBMS [14]. In the 1990s, advances in computer networking coupled with the growing business needs to manage distributed data fueled the work on distributed database systems.

Foundations

A distributed DBMS (DDBMS) manages a distributed database that is accessed at multiple sites, where each site contains a partition of the database. A single partition can be replicated across multiple sites. At one end of the spectrum, in a fully-replicated database, each site has a full copy of the database. At the other end of the spectrum, a fully-partitioned database is divided into disjoint partitions, also called fragments, and each is placed at only one site. Hence, a DDBMS typically maintains a data directory which maps each data item to the sites at which the data item is maintained. This design raises three key challenges for DDBMSs. The first challenge is the distribution and placement of the data. Second, the distributed DBMS must update all copies of modified data items when executing update transactions. The third challenge arises when a query needs to access data items at multiple sites, requiring coordination and communication among the sites. Despite these challenges, there is a potential gain in performance due to the proximity of data to applications and due to inter-transaction and intra-transaction parallelism in the workload.

Data Placement among Sites

As far as data placement is concerned, there is a performance tradeoff. Applications running at a site prefer to access all needed data items locally. At the same time, this naturally leads to an increasing number of data replicas, and updates become more expensive as they need to reach more replicas. This tradeoff makes deciding how the database is distributed one of the main design decisions. The database designer typically determines which relations are allocated to which sites. Data placement and distribution depend primarily on the applications that access the database as well as on the server network. More fine-grained data distribution as used in parallel database systems [3,5] is also possible. For example, a single relation can be fragmented horizontally (e.g., using a selection operator) or vertically (e.g., using a projection operator) into several sub-relations and each sub-relation is allocated to at least one site.

Data placement can also be performed dynamically [6]. Most approaches to dynamic data placement are adaptive. They keep statistics on the workload and move or copy data at different sites so as to adjust the data placement to the current workload. For example, a new copy of a set of data items could be established to help balance the load among servers, or to reduce wide-area network communication costs. When a new copy of a data item is made, the copy is designated as either a replica or a cache. A replica of a data item is long-lived and maintained by reflecting the item's modification to the copy. On the other hand, a cache is typically short-lived and invalidated on changes to the data item. Due to the complexity of the dynamic data placement problem, some research work targets using economic models [6] to optimize dynamic data placement. One can no longer discuss caching and assume that the DDBMS maintains all data item copies.

Propagating the Effects of Update Transactions

When an update transaction modifies the value of a database item, the DDBMS is responsible for reflecting this change to all copies of the data item; otherwise the copies become inconsistent and consequently the database servers may diverge, violating system correctness properties. There is a large body of work on handling updates, mainly inspired by the work in distributed systems (e.g., quorum protocols). However, few ideas have impacted commercial DDBMS products as they mainly use ROWAA (Read One Write All Available) protocols. In a ROWAA protocol, the update of a data item has to reach all copies of that data item. It is, however, sufficient to access only a single copy to read any data item. Here the discussion focuses on a key correctness property for ROWAA protocols: one-copy semantics.

When a distributed system offers one-copy semantics, clients cannot tell whether they are communicating with a single DBMS or a distributed DBMS. Without one-copy semantics, applications have to deal with inconsistencies while accessing the database. In practice, some applications require one-copy semantics (e.g., airline reservation, banking, and e-commerce systems). However, a large class of applications can tolerate inconsistencies (e.g., reporting applications).

The process of reflecting the changes of modified data items is called update propagation, and there are several protocols that implement it. For example in eager protocols, all live copies of each modified data item are changed as part of the update transaction. In contrast, lazy protocols propagate the changes after the commit of the update transaction.

One-copy semantics can be implemented using eager or lazy update propagation protocols, but it requires a total order on the propagated updates. This total order is typically the commit order of update transactions and each site applies its relevant updates in an order consistent with the total order.

Distributed Query Execution

When a distributed DBMS receives a query, it generates a query execution plan that may access data at multiple sites. Each site executes part of the plan and may need to communicate with other sites. Communication among sites is expensive and good plans generally minimize the amount of data exchanged between sites [8]. In wide-area networks, the cost of communication is generally higher than the cost of local processing. In contrast, the cost of communication is equal to or less than the cost of local processing in local-area networks. Optimizing distributed queries is challenging [6] because the search space is large. It includes the selection of data fragments to be accessed, the order of operations in the plan, and the cost of communication among the sites. The problem becomes harder when optimizing multiple queries at a time.

A DDBMS provides global transaction isolation (e.g., one-copy serializability [9] or generalized snapshot isolation [4]) through distributed concurrency control. To provide global isolation for transactions, local concurrency control protocols alone are insufficient. For example, if each site individually ensures local serializability, the global transaction execution may violate global serializability since two sites could locally serialize conflicting global transactions in different orders. Protocols that implement distributed concurrency control (e.g., distributed two-phase locking or multi-version timestamp ordering) require coordination among the sites.

For example in locking-based distributed concurrency control protocols, the primary-site [1] method uses a central lock controller to manage the locks. Alternatively, in the primary-copy [15] method each data item has one copy that is designated as the primary copy, and only the primary copy is locked. This design allows locks to be distributed with the data among several sites.

Coordination among several sites can become a significant source of overhead. Deadlocks and aborts become more frequent in a distributed DBMS compared to centralized DBMS because distributed transactions require communication among multiple sites and therefore take longer to execute.

Handling Failures

Fault-tolerance is an important aspect in any distributed system. A DDBMS contains multiple sites and each site can fail independently. The DDBMS, therefore, needs to cope with and recover from site failures. The DDBMS always guarantees the safety properties – including atomicity and durability of distributed update transactions – despite site failures [9,12]. To terminate an update transaction while ensuring the safety properties, the DDBMS employs a distributed commit protocol [1] (e.g., distributed two-phase commit, and three-phase commit) that are specifically designed to handle site failures. Liveness properties, which is concerned with the ability to process transactions, depend on which and how many sites are still connected and operating normally. Liveness properties encompass system performance and availability and usually require high degree of DDBMS customization to meet the desired targets. For example instead of using primitive techniques to recover from a crash and bring a site's database up-to-date, data streaming from multiple sources at several sites substantially reduces database recovery time and therefore improves the DDBMS availability.

Key Applications

Today, distributed DBMSs are used to manage distributed databases, such as in geographically distributed systems (e.g., hotel chains and multi-plant manufacturing systems), and in databases under several administrative or autonomous domains. Distributed DBMSs are also used to achieve fault-tolerance and scalability when a centralized DBMS is not satisfactory, for instance in financial transaction processing and airline reservation systems.

In modern data centers that host web services, DDBMS technology appears in two forms. First in fully replicated database systems, which consist of a cluster of servers interconnected through a fast local-area network. The objective of such replicated databases is to achieve both higher availability since data is available at several server nodes as well as higher performance as transactions can be processed in parallel at the database servers.

The second form is partitioned database systems in which relations are striped (fragmented) among several servers. A single query can be divided into smaller subqueries that execute on the servers, leading to a shorter response time.

Future Directions

Currently, DDMBS technology is rarely used in large scale information systems without extensive customization necessary to obtain adequate performance. At the same time, networking and storage technologies are advancing at a pace much higher than the improvement in microprocessors clock speeds. These trends suggest that DDBMS use will increase in the future. But further research is needed to ease the deployment and management of DDBMS.

Cross-references

- ▶ DBMS
- ▶ Distributed Concurrency Control
- ▶ Distributed Databases
- ▶ Distributed Database Design
- ▶ Parallel Database Management
- ▶ Replication for High Availability
- ▶ Replication for Scalability

Recommended Reading

1. Bernstein P. and Goodman N. Concurrency control in distributed database systems. ACM Comput. Surv., 13 (2):185–221, 1981.
2. Bernstein P., Shipman D., and Rothnie J. Concurrency control in a system for distributed databases (SDD-1). ACM Trans. Database Syst., 5(1):18–51, 1980.
3. DeWitt D. and Gray J. Parallel database systems: the future of high performance database systems. Commun. ACM., 35(6):85–98, 1992.
4. Elnikety S., Pedone F., and Zwaenepoel W. Database replication using generalized snapshot isolation. In Proc. 24th Symp. on Reliable Distributed Syst., 2005.
5. Ghandeharizadeh S., Gao S., Gahagan C., and Krauss R. High performance parallel database management systems. In Handbook on Data Management in Information Systems, J. Blazewicz, W. Kubiak, T. Morzy, M. Rusinkiewicz (eds.). Springer, 2003, pp. 194–220.
6. Kossmann D. The state of the art in distributed query processing. ACM Comput. Surv., 32(4):422–469, 2000.
7. Muffin S.M. A Distributed Database Machine. ERL Technical Report UCB/ERL M79/28, University of California at Berkeley, CA, 1979.
8. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems. Prentice-Hall, Englewood Cliffs, NJ, 1991.
9. Pacitti E., Coulon C., Valduriez P., and Özsu T. Preventive replication in a database cluster. Distrib. Parallel Databases., 18(3):223–251, 2005.
10. Papadimitriou C. The theory of database concurrency control. CS Press, AB, 1988.
11. Ries D. and Epstein R. Evaluation of Distribution Criteria for Distributed Database Systems. UCB/ERL Technical Report M78/22, UC Berkeley, CA, 1978.
12. Skeen D. and Stonebraker M. A formal model of crash recovery in a distributed system. IEEE Transactions on Software Engineering 9(3):219–228, 1983.
13. Stonebraker M. The case for shared nothing. IEEE Database Eng. Bull., 9: 4–9, 1986.
14. Stonebraker M. Readings in Database Systems (2nd ed.). Morgan Kaufmann Publishers, Scan Mateo, CA, 1994.
15. Zaslavsky A., Faiz M., Srinivasan B., Rasheed A., and Lai S. Primary copy method and its modifications for database replication in distributed mobile computing environment. In Proc. 15th Symp. on Reliable Distributed Syst., 1996.

Distributed Deadlock Management

WEE HYONG TOK
National University of Singapore, Singapore, Singapore

Synonyms

Deadlocks in distributed database systems

Definition

In a database that supports locking protocol, accesses to data are controlled using locks. Whenever a transaction needs to access a shared object, it will be granted a lock (and hence access) to the object if there is no other conflicting locks on the object; otherwise, the requesting transaction has to wait. A deadlock occurs when transactions accessing shared data objects are waiting *indefinitely* in a circular fashion until a special action (such as aborting one of the transactions) is taken. In a distributed database environment, deadlocks can occur locally at a single site, or across sites where a chain of transactions may be waiting for one another to release the locks over a set of shared objects.

For example, consider two data objects o_1 and o_2 stored at site 1 and site 2 respectively. Suppose two transactions, T_1 and T_2, initiated at site 1 and site 2, are updating o_1 and o_2 concurrently. As T_1 is updating o_1 at site 1, it holds a write lock on o_1. Similarly, T_2 holds a write lock on o_2 at site 2. When T_1 attempts to access o_2, it has to wait for T_2 to release the lock on o_2. Likewise, T_2 has to wait for T_1 to release the lock on o_1. This leads to a deadlock as both transactions cannot proceed.

Three techniques are commonly used to handle deadlocks: prevention, avoidance, and detection and resolution. Deadlock prevention methods ensure that deadlocks will not occur in the database system. This is achieved by preventing transactions which will cause deadlocks from being executed. Deadlock avoidance schemes preemptively detect potential deadlocks during the scheduling of multiple transactions. Deadlock detection protocols detect deadlocks in running transactions by using a transaction wait-for graph. Whenever deadlocks are detected, a resolution strategy aborts some of the transactions in order to break the deadlock.

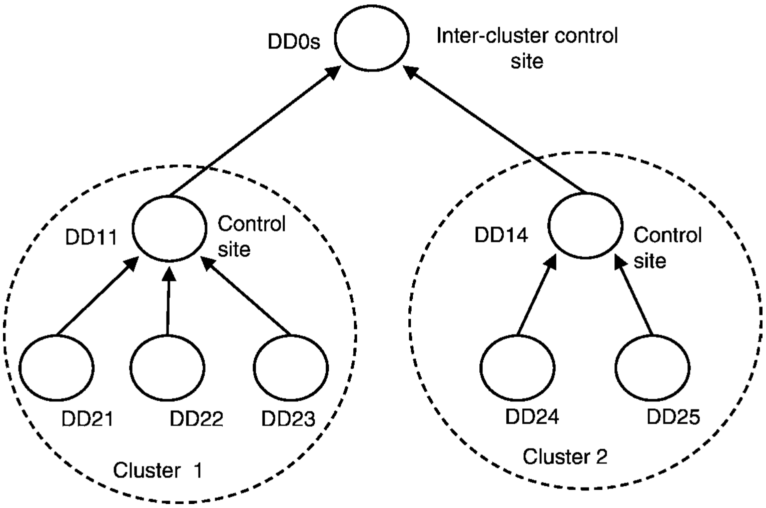
Historical Background

When transactions access shared objects, there is a dependency relationship between the transactions. The dependency relationship captures how the transactions wait for one another, and is commonly represented as a directed graph, called the wait-for graph (WFG). Each node represents a transaction. An edge (i.e., arc) in the graph is used to capture the wait-for relationship. For example, if a directed edge is found between node A and node B, then the transaction represented by node A is waiting for another transaction which is represented by node B. In distributed database systems, the WFG is local if it involves only the data at a single site, and is global if it involves data from multiple sites. When a cycle exists in a WFG, a deadlock occurs.

Deadlock detection algorithms for distributed database systems can be categorized as: centralized, hierarchical, and distributed [12].

Centralized deadlock detection algorithms [5,6] use a central site for detecting deadlocks in a distributed database system. This site is referred to as the central deadlock detection site, *C*. *C* is responsible for building the global WFG. Other sites transmit their local WFG to *C*. Subsequently, only the changes to the local WFG are transmitted to *C*. These changes include new or deleted edges in the local WFG. *C* continuously checks for cycles in the global WFG and performs deadlock resolution whenever cycles are detected. The implementation of centralized deadlock detection is simple and straightforward. However, due to the need to continuously transmit new changes in the local TWG to *C*, it can cause a high communication overhead. In addition, the use of a single site makes it susceptible to overloading and being a single point of failure. Özsu and Valduriez [12] noted that centralized two-phase locking (2PL) and deadlock detection is a good, natural combination. Centralized deadlock detection is implemented in Distributed INGRES [15].

Hierarchical deadlock detection algorithms [10,6] rely on a hierarchical organization of the sites in a distributed database system to detect deadlocks. Each internal node (site) merges the WFG from its child nodes into a WFG that captures the dependency relationship among the descendant nodes. It can thus detect deadlocks in its descendant nodes. It also transmits the combined WFG to its parent node. The key idea in hierarchical deadlock detection is to ensure that a deadlock can be detected as early as possible by a nearby site. This reduces the need to escalate the detection to the root site. As deadlock detection is spread amongst several sites, hierarchical deadlock detection algorithms incur less communication overheads compared to centralized algorithms. The implementation of hierarchical deadlock detection algorithms is more complex due to the need to coordinate between the multiple deadlock detectors. Figure 1 shows an example of how five sites in a distributed database system is organized in a hierarchy for deadlock detection. Some sites might function as the deadlock detector for multiple levels of the hierarchy. Each deadlock detector is denoted as DD_{ls}, where *l* and *s* denote the level and

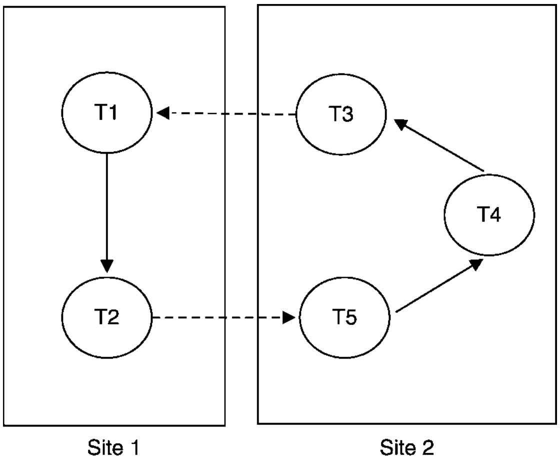


Distributed Deadlock Management. Figure 1. Hierarchical deadlock detection.

the site of the deadlock detector respectively. From the figure, observe that in cluster 1, site 1 is the control site, and detects deadlocks for sites 1, 2, and 3. Thus, sites 1, 2, and 3 will need to update the deadlock detector at the next level (i.e., DD_{11}) with their respective local WFG. If the deadlock needs to be detected between site 1 and site 5, then it will be detected by the deadlock detector at the root of the tree (i.e., DD_{0s}, where $1 \leq s \leq 5$).

Distributed deadlock detection algorithms [10,11] rely on the cooperation of all the sites in the distributed database system in order to detect deadlocks. Each site in the distributed database system consists of a deadlock detector. Potential deadlock information is transmitted from one site to another. Consequently, the deadlock detector modifies its local WFG using information about the potential deadlocks, as well as whether a local transaction is waiting for transactions at another sites. Distributed deadlock detection is implemented in System R* [11].

In distributed database systems, effective global deadlock detection rely on the the timely propagation of local information from all the sites. However, some sites might be slower in propagating local information. As a result, this might lead to the detection of phantom deadlocks. A phantom deadlock is a deadlock which does not exist. Hence, in order to break the phantom deadlocks, transactions might be aborted. Both the centralized and hierarchical algorithms are prone to the phantom deadlock problem. For example, in Fig. 2, a global deadlock exists. However, at site 2, the transaction T_5 might be aborted. T_5 can aborted due to the business logic encoded in the transaction. This changes the local WFG for site 2. Consequently, no more cycle exists in the global WFG. However, if the changes in the local WFG are not propagated in a timely manner to update the global WFG, a phantom deadlock arises.



Distributed Deadlock Management. Figure 2. Global wait-for graph.

The book [12] provides a good overview of distributed deadlock management techniques. The surveys [4,7,14,1] provide a good discussion of various distributed deadlock detection algorithms. Krivokapić et al. [8] further categorizes deadlock detection approaches as path-pushing [10], probe-based [13] or having a global state strategy [2,3]. Using the categorization, Krivokapić et al. [8] presented a detailed performance analysis of representative algorithms for each of the categories.

Foundations

A distributed database system consists of a collection of database sites. A centralized database system is located at each of the sites. The database sites communicate with each other by sending messages via a communication network. A transaction consists of a sequence of operations (e.g., read, write) that are performed on the data objects. Whenever a transaction needs to perform an operation on a data object, it sends a resource request to a transaction manager (TM). The resource request can refer to operations that are performed on local or remote data objects.

Transaction Wait-for Graph

Given N active transactions, $T_1 ... T_N$, in a database system, a directed graph, called a transaction Wait-for Graph (WFG), can be built. Each vertex of the graph corresponds to an active transaction. A wait-for relationship exists between two transactions, T_i and T_j ($i \neq j$), if T_i is waiting for T_j to release a lock. This is denoted as a directed edge between two vertices, V_i and V_j ($i \neq j$) in the graph. In most deadlock detection algorithms, the WFG is used for analyzing deadlocks. Deadlocks occur when cycles are detected in the graph.

In order to analyze deadlocks in a distributed database system, a global transaction wait-for-graph (WFG) is commonly used. The global WFG is constructed by taking the union of the local WFGs for all the sites in the distributed database system. The main difference between the local and global WFG is that the global WFG captures inter-site wait-for relationship. Inter-site waiting occurs when a transaction, executing on one site, is waiting for the release of a lock on a data object by another transaction executing on another site. For example, in a distributed database system, five transactions, T_1 to T_5 are active on two sites. Figure 2 shows the global WFG, which captures the wait-for relationship between the transactions. The dashed lines are used to denote inter-site waiting. From the figure, several observations can be made. Firstly, one can observe that at site 1, transaction T_1 is waiting for T_2. At site 2, T_5 is waiting for T_4, and T_4 is waiting for T_3. In addition, the transaction T_2 (site 1) is waiting for T_5 (site 2), and T_3 (site 2) is waiting for T_1 (site 1). As a cycle is detected in the global WFG, a deadlock is detected.

Deadlock Models

Different types of models of deadlock are presented in [7]. The models are used to capture the type of resource requests that need to be processed by application programs. The models include: one-resource, *AND*, *OR*, *AND-OR*, (nk), and the unrestricted model. In the one-resource model, a transaction can have at most one resource request. Thus, the maximum outdegree for a WFG vertex is 1. In both the *AND* and *OR* models, a transaction requests for access to a set of shared data objects (i.e., resources). The main difference between the two models is that in the *AND* model, the transaction blocks until all the requested resources are available, whereas in the *OR* model, a transaction blocks until any one of the resources is available. The *AND-OR* deadlock model is a generalization of the *AND* and *OR* models. In order to further generalize the *AND-OR* model, the $\binom{n}{k}$ model is used. In this model, a transaction can request for any k available resources from a set of n available resources. The unrestricted model does not impose any constraints on the number of resources that are requested by a transaction. In the one resource and *AND* deadlock model, a deadlock occurs whenever there is a cycle in the WFG. The detection of deadlocks in the other models require more complex computation, and is discussed in details in [7].

Static Vs Dynamic Deadlock Detection

Deadlock detection can be classified as static or dynamic. In static deadlock detection, the overall strategy in which deadlocks is detected is fixed. For example, in centralized deadlock detection, the central site is pre-determined. Similarly, in hierarchical deadlock detection, the hierarchical organization of the sites is also pre-determined. In distributed deadlock detection, all sites have individual deadlock detection mechanisms. In dynamic deadlock detection, deadlock detection agents (DDA) are dynamically created for transactions that access the same data objects. This is first proposed in [8]. This allows the DDA scheme to adapt or self-tune to the system transaction load.

Deadlock Resolution

Whenever deadlocks are detected, deadlock resolution is used to remove the deadlocks. The key idea in deadlock resolution is to minimize the cost incurred as a result of resolving the deadlock. There are two possible strategies for deadlock resolution. In the first strategy, a deadlock resolver aborts one or more transactions that caused the deadlock. In the second strategy, transactions are timed-out whenever deadlock occurs.

In the first strategy, one or more transactions are selected to be aborted. These transactions are referred to as the *victim transaction(s)*. Singhal [14] presents a general strategy for deadlock resolution in distributed database systems. A victim transaction which will optimally resolve the deadlock is selected. The victim transaction is aborted, and the locks that are held by the transaction are released. Deadlock detection information that are related to the victim transaction is removed from the system. In a distributed database system, several issues need to be considered when aborting transactions. First, whenever a deadlock is detected by a site, the site might not have access to the global deadlock information. Second, multiple sites might independently detect the same deadlock. This might cause the sites to independently resolve the deadlock. Consequently, this causes more transactions to be aborted than necessary. To solve this issue, a deadlock resolver can be selected from amongst the sites or the deadlock detection by various sites can be prioritized. In order to determine the set of victim transactions, various heuristics can be used. Intuitively, the heuristics ensure that the cost of aborting transactions is minimized. Some of these heuristics include: choosing the youngest transaction in the cycle [8] or choosing the transaction that causes the maximum number of cycles [9]. The first heuristic is motivated by the observation that the youngest transaction has just started execution. Hence, it is less costly to abort. In contrast, an older transaction has executed for some time, and will be costly to abort. The second heuristic is motivated by the observation that when deadlocks occur, it is important to break the cycles in the global WFG. If a transaction that caused the maximum number of cycles is aborted, it can potentially remove more cycles in the WFG. Hence, deadlock can be resolved faster.

In the second strategy, deadlocks are resolved using time-outs. This strategy is suitable for the case where deadlocks are infrequent. The time-out interval determines the waiting time for a transaction to be aborted. Thus, the selection of an optimal time-out interval is important. A short time-out interval will cause transactions to be aborted unnecessarily. On the other hand, a long time-out interval will result in the slow resolution of deadlocks might not be resolved quickly. Consequently, this impacts on the responsiveness of the applications.

Key Applications

Deadlocks occur whenever locking-based protocols are used to manage shared data objects or resources. Distributed deadlock management is more challenging to handle because none of the sites have global knowledge of the entire system. The techniques that are described can be applied and adapted for deadlock management for various types of distributed systems.

Future Directions

The emergence of new computing platforms (e.g., Peer-to-Peer (P2P), cloud computing) present new interaction modalities with distributed data. As applications built on these paradigms mature, a natural progression would be the need for transactions which access shared resources. This compels the need for locking-based protocols to be used for accessing the shared resources. Consequently, many open issues arise for distributed deadlock detection in these new computing platforms.

Cross-references

- ▶ Distributed Concurrency Control
- ▶ Distributed Database Design
- ▶ Distributed Database Systems
- ▶ Two-Pheese Loching

Recommended Reading

1. Abonamah A.A. and Elmagarmid A. A survey of deadlock detection algorithms in distributed database systems. Advances in Distributed and Parallel Processing (vol. one): system paradigms and methods, 1994, pp. 310–341.
2. Bracha G. and Sam T. Distributed deadlock detection. Distributed Computing, 2(3):127–138, 1985.
3. Chandy K.M and Lamport L. Distributed snapshots: Determining global states of distributed systems. ACM Trans. Comput. Syst., 3(1):63–75, 1986.
4. Elmagarmid A.K. A Survery of distsributed deadlock algorithms. ACM SIGMOD Record, 15(3):37–45, 1986.
5. Gray J. Notes on data base operating systems. In Advanced Course: Operating Systems, pages 393–481, 1978.
6. Ho Gray S. and Ramamoorthy C.V. Protocols for deadlock detection in distributed database systems. IEEE Trans. Softw. Eng., 8(6):554–557, 1982.
7. Knapp E. Deadlock detection in distributed databases. ACM Comput. Surv., 19(4):303–328, 1987.
8. Krivokapić., N. Kemper A. and Gudes E. Deadlock detection in distributed database systems: a new algorithm and a comparative performance analysis. VLDB J., 8(2):79–100, 1999.
9. Makki K. and Pissinou N. Detection and resolution of deadlocks in distributed database systems. In Proc. Int. Conf. on Information and Knowledge Management, 1995, pp. 411–416.
10. Menascé D.A. Muntz R. Locking and deadlock detection in distributed data bases. IEEE Trans. Softw. Eng., 5(3):195–202, 1997.
11. Mohan C., Lindsay., and Bruce G. Obermarck Ron Transaction management in the R* distributed database management system. ACM Trans. Database Syst., 11(4):378–396, 1986.
12. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, Second Edition. Prentice-Hall, 1999.
13. Roesler M., Burkhard W.A. and Cooper K.B. Efficient deadlock resolution for lock-based concurrency control schemes. In Proc. 18th Int. Conf. on Distributed Computing Systems, 1998, pp. 224–233.
14. Singhal M. Deadlock detection in distributed systems. Computer, 22(11):37–48, 1989.
15. Stonebraker M. The design and implementation of distributed ingres. The INGRES Papers: anatomy of a Relational Database System, 1986, pp. 187–196.

Distributed Hash Table

Wojciech Galuba, Sarunas Girdzijauskas
EPFL, Lausanne, Switzerland

Synonyms

DHT

Definition

A *Distributed Hash Table* (DHT) is a decentralized system that provides the functionality of a hash table, i.e., insertion and retrieval of key-value pairs. Each node in the system stores a part of the hash table. The nodes are interconnected in a *structured overlay network*, which enables efficient delivery of the key lookup and key insertion requests from the requestor to the node storing the key. To guarantee robustness to arrivals and departures of nodes, the overlay network topology is *maintained* and the key-value pairs are *replicated* to several nodes.

Key Points

Every DHT defines its key space. For example, in many DHTs the keys are 160-bit integers, which is the output of the popular SHA1 hash function. Each node in the system has a specific location in the key space and stores the key-value pairs that are close to that location. The different DHT systems vary in the exact algorithms for deciding which node should store which key [2].

The DHT relies on a structured overlay network for some of its functionality. The overlay network uses the DHT keys for addressing its nodes, i.e., the overlay is content (key) addressable. The structured overlay network provides the routing primitive which allows scalable and reliable delivery of key lookup and key insertion messages. The structured overlay implementations maintain the overlay topology and ensure routing efficiency as the nodes arrive and depart from the system (node churn) [1].

Not only the overlay topology but also the DHT data storage must be tolerant to churn. To achieve that, the key-value pairs are replicated across several nodes. A sufficient number of replicas needs to be maintained to prevent data loss under churn. There are a number of approaches to replication (cf. Peer-to-peer storage systems). In the most common replication strategy a node joining the DHT contacts the nodes that are close to it in the key space and replicates the key-value pairs that they store.

Cross-references

- ▶ Consistent Hashing
- ▶ Distributed Hash Table
- ▶ Overlay Network
- ▶ Peer to Peer Overlay Networks: Structure, Routing and Maintenance
- ▶ Peer-to-Peer System
- ▶ Replication in DHTs

Recommended Reading

1. Rhea S.C., Geels D., Roscoe T., and Kubiatowicz J. Handling churn in a DHT. In Proc. USENIX 2004 Annual Technical Conf., 2004, pp. 127–140.
2. Risson J. and Moors T. Survey of research towards robust peer-to-peer networks: Search methods. Comput. Networks, 50(17): 3485–3521, 2006.

Distributed Join

Kai-Uwe Sattler
Technical University of Ilmenau, Ilmenau, Germany

Synonyms

Join processing; Distributed query

Definition

The distributed join is a query operator that combines two relations stored at different sites in the following way: each tuple from the first relation is concatenated with each tuple from the second relation that satisfies a given join condition, e.g., the match in two attributes. The main characteristics of a distributed join is that at least one of the operand relations has to be transferred to another site.

Historical Background

Techniques for evaluating joins on distributed relations have already been dy discussed in the context of the first prototypes of distributed database systems such as SDD-1, Distributed INGRES and R*. In [6] the basic strategies ship whole vs. fetch matches were discussed and results of experimental evaluations were reported. Another report on an experimental comparison of distributed join strategies was given in [5].

Special strategies for distributed join evaluation that aim at reducing the transfer costs were developed by Bernstein et al. [2] (semijoin) as well as Babb [1] and Valduriez [11] (hashfilter join) respectively. The problem of delayed and bursty arrivals of tuples during join processing was addressed by particular techniques like the XJoin [10]. Moreover, a technique for dealing with limited query capabilities of wrappers in heterogeneous databases was introduced in [8] as a special variant of the fetch matches strategy.

Foundations

For evaluating a distributed join several tasks have to be addressed. First, the site where the actual join is to be processed, has to be chosen. Second, the operand relations have to be transferred to the chosen site if they are not already available there. Finally, the join between the two relations has to be computed locally at that site. In the following, these different issues are described in more details.

Site Selection

Choosing the site where the join operation is to be performed on is usually part of query optimization. For the simplest case of joining two relations R and S there are three possible join sites: the site of R, the site of S, and a third site at which the result is eventually needed. For the decision several aspects must be taken into account: the cost for transferring the operand relations, the join ordering (in case of multi-way joins) as well as subsequent operators in the query, e.g., at which site the join result is needed for further processing.

Relation Transfer

Given that one of the relations is available at the join site, there are two basic strategies to transfer the other relation. The *ship whole* approach works as follows:

1. The remote relation is shipped to the join site at a whole.
2. After the relation is received by the join site, it can be stored and used to perform the join locally.

In contrast, the *fetch matches* or *fetch as needed* strategy consists of the following steps:

1. The relation at the join site is scanned.
2. Based on the current value of the join attribute, the matching tuples from the other relation are requested.
3. If the other site has matching tuples, they are sent back and joined with the currently considered tuple.

The following example illustrates the costs (in terms of transferred data and number of messages) of these two strategies. Given two relations R (site 1) and S (site 2) with cardinalities $|R| = 2.000$ and $|S| = 5.000$, where the tuple sizes in bytes are $width(R) = width(S) = 100$ and the join attributes $A \in attr(R)$ and $C \in attr(S)$ have $width(A) = width(C) = 10$. Furthermore, a foreign key constraint $A \rightarrow C$ is assumed meaning that $|R \bowtie_{A=C} S| = 2.000$.

Using the ship whole approach the complete relation has to be shipped, i.e., in case of R the transfer volume is $|R| \cdot width(R) + |R \bowtie_{A=C} S| \cdot (width(R) + width(S))$, but only two messages are required (assuming that the relation is sent using a single message). With the fetch matches strategy the join attribute value of each tuple is shipped to the other site and for each tuple a result message is sent back. Thus, for relation R $2|R|$ messages are needed. However, in the first step $|R| \cdot width(A)$ bytes are transferred and in the second step $|R \bowtie_{A=C} S| \cdot width(S)$ bytes are sent back. The results for all strategies are shown in Table 1. Note, that only that cases are shown, where the result is sent back to the query issuer. Furthermore, for simplicity the message sizes are not considered in the transfer volume.

Distributed Join. Table 1. Transfer costs for join strategies

Strategy	Query issuing site	Transfer volume (KBytes)	Number of messages
Ship whole	Site 1	600	2
Ship whole	Site 2	900	2
Fetch matches	Site 1	220	4.000
Fetch matches	Site 2	250	10.000

The results show that:

- The ship whole approach needs a larger transfer volume.
- The fetch matches strategy requires more messages.
- The smaller relation should always be shipped.
- The transfer volume in the fetch matches strategy depends on the selectivity of the join.

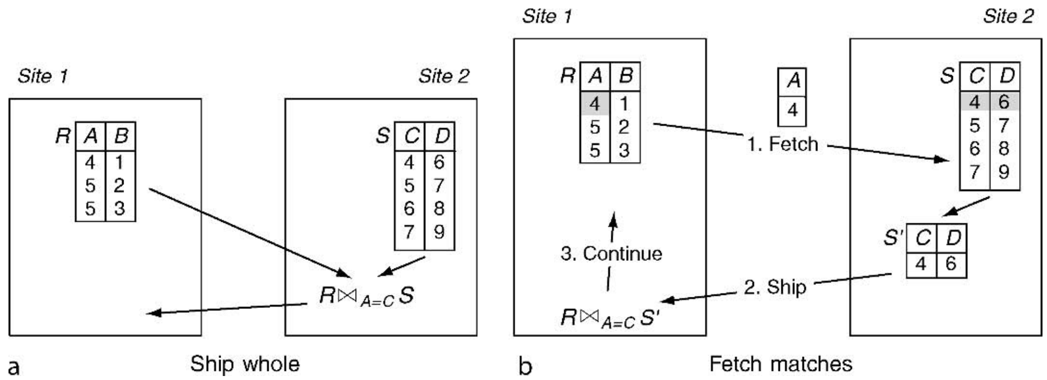
These basic strategies can further be improved by exploiting *row blocking*, i.e., sending several tuples in a block. Second, in case of the fetch matches strategy the relation can be sorted first in order to avoid fetching tuples for the same join values multiple times.

Local Join Processing

After having transferred the operand relations to the join site, the join is evaluated using any of the conventional algorithm known from centralized DBMS, e.g., nested-loops join, sort-merge join, or hash join. If local indexes on the join attributes are available, they can be exploited. Furthermore, for the ship whole strategy a temporary index on the shipped relation can be created and exploited. In any case, the decision which strategy to use for the join evaluation is made in the local optimization step.

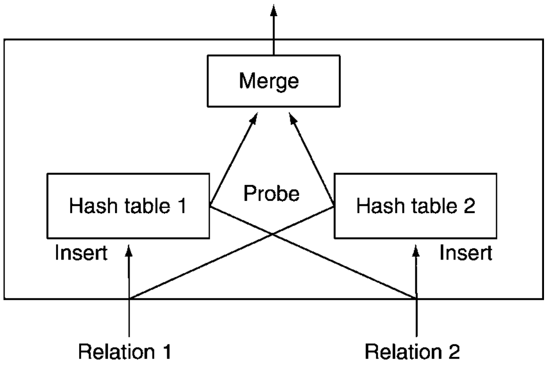
Sequential vs. Pipelined Processing

Another aspect of local processing is the strategy for dealing with delayed or bursty tuple arrivals. A first



Distributed Join. Figure 1. Basic strategies for join processing.

approach is to simply ignore this issue and assume a constant and on time arrival of tuples. Each incoming tuple is processed according to the chosen join strategy. In case of the sort-merge join this could mean to wait for all tuples of the relation. With other strategies (e.g., if the shipped relation is the outer relation of a nested-loops join) the incoming tuple is directly processed in a pipelined fashion. However, if no tuple arrives, e.g., due to a network delay, no join result can be produced. An alternative solution is to use a double pipelined hash join, which exploits the inherent parallelism of distributed processing and in this way allows to reduce the overall response time.



Distributed Join. Figure 2. Double pipelined hash join.

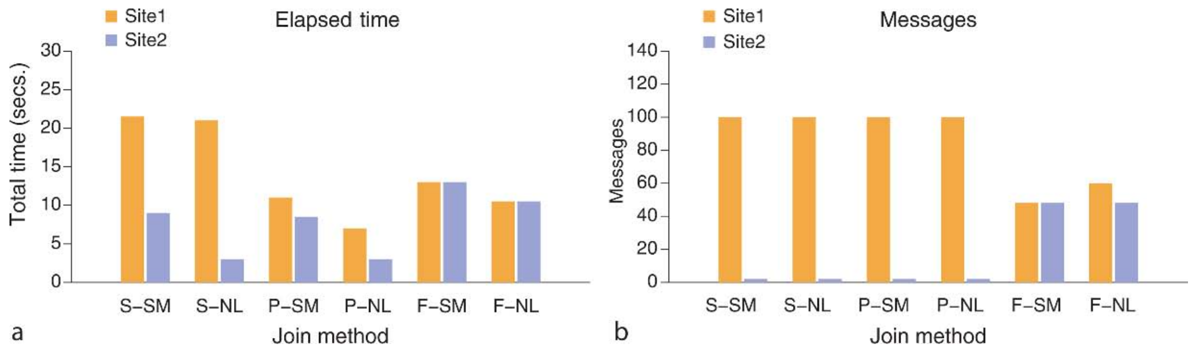
Algorithms

In the following some special approaches of distributed joins are described that extend the basic model of ship whole and fetch matches.

Semijoin/hashfilter join. This join algorithms can be regarded as a special variant of the fetch matches join with the aim to reduce transfer costs. For this purpose, only the projected join column or a compact bitmap representation (computed by applying a hash function) of that column of the first relation is sent to the second site as a whole. Next, all matching tuples are determined and sent back to the first site, where the actual join is computed.

XJoin. In wide-area networks with unpredictable response and transfer times, transmission delays may result in delayed or bursty arrivals of data. Furthermore, slow data sources can also delay or even block join processing. To address this problem and to allow to deliver join results as early as possible and continuously the XJoin [10] was proposed. It is based on the symmetric hash join, which works as follows: For each operand relation a hash table is maintained. Each incoming tuple is first inserted into the corresponding hash table. Next, it is used for probing the hash table of the other relation to find matching tuples, compute the join with them, and output the result immediately (Fig. 2). The XJoin extends this algorithm by considering the case where the hash tables exhaust the available main memory. For this purpose, a partitioning strategy is applied to swap out portions of the hash tables to disk-resident partitions. These partitions are also used to produce results when the site waits for the next tuples: in this case tuples from the disk are joined with memory-resident partitions. In order to avoid duplicates in the result special precautions are needed.

Bind join. In heterogeneous databases component databases (data sources) are usually encapsulated by wrappers responsible for query and result translation. Depending on the kind of the sources (e.g., a legacy system, a Website, or Web Service) these wrappers sometimes do not allow fetching the whole table or



Distributed Join. Figure 3. Comparison of different join algorithms.

evaluating a join. Instead they support only parameterized selections of the form

```
select * from R where A = ``?''
```

In order to still join another relation with *R* a bind join can be performed which is in fact a special fetch matches strategy. By scanning the outer relation the current value of the join column is passed on as the parameter ``?'' to the wrapper query and the results are collected and combined into the final join result. This can further be improved by precompiling the query plan, e.g., by exploiting prepared statements or cursor caching techniques.

Key Applications

The main application of distributed joins is query processing in distributed databases systems. In order to evaluate a join operation on relations stored at different sites a distributed strategy is needed.

A second area of application are heterogeneous databases, e.g., in the form of mediators or federated database systems. If legacy component databases or their wrappers provide only limited query capabilities (e.g., supporting only selections), a special strategy is required, which was introduced above as bind join. The XJoin presented above is also useful for evaluating joins in this context.

Other application examples for distributed joins include P2P systems for managing structured data, e.g., Peer Data Management Systems (PDMS) and P2P databases as well as distributed data stream processing systems.

Experimental Results

Experimental comparisons of different join strategies have been reported for example by Lu and Carey [5] as well as by Mackert and Lohman [6]. Figure 3 shows some results from [5] for the join between two relations from two different sites, each with 1,000 tuples and a result size of 100 tuples.

The join algorithms considered in this experiment are:

- A sequential combination of the ship whole approach with the nested-loops join for the local processing (S-NL) and the sort-merge join (S-SM). The shipped relation was stored in a temporary table and for the nested-loops variant an index was created before computing the join.
- A pipelined variant of this approach, where incoming tuples were processed on the fly (P-NL and P-SM).
- A strategy equivalent to the fetch matches approach using nested-loops join (F-NL) or sort-merge join (F-SM).

In Fig. 3a the elapsed time for processing the joins is shown, Fig. 3b depicts the number of messages.

Cross-references

▶ Evaluation of Relational Operators
▶ Semijoin

Recommended Reading

1. Babb E. Implementing a Relational Database by Means of Specialized Hardware. ACM Trans. Database Syst., 4(1):1–29, 1979.
2. Bernstein P.A., Goodman N., Wong E., Reeve C.L., Rothnie J.B. Query Processing in a System for Distributed Databases (SDD-1). ACM Trans. Database Syst., 6(4): 602–625, 1981.
3. Hevner A.R., Yao S.B.: Query Processing in Distributed Database Systems. IEEE Trans. on Software Eng., 5(3):177–182, 1979.
4. Kossmann D. The State of the Art in Distributed Query Processing. ACM Comput. Surv., 32(4):422–469, 2000.

5. Lu H., Carey M. Some Experimental Results on Distributed Join Algorithms in a Local Network. In Proc. 11th Int. Conf. on Very Large Data Bases, 1985, pp. 229–304.
6. Mackert L.F., Lohman G. R* Optimizer Validation and Performance Evaluation for Local Queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 84–95.
7. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 2nd Edition. Prentice Hall 1999.
8. Roth M.T., Schwarz P. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997, pp. 266–275.
9. Stonebraker M. The Design and Implementation of Distributed INGRES. In The INGRES Papers, M. Stonebraker (ed.): Addison-Wesley, Reading, MA, 1986.
10. Urhan T., Franklin M.J. XJoin: A Reactively-Scheduled Pipelined Join Operator. Bulletin of the Technical Committee on Data Engineering 23(2):27–33, 2000.
11. Valduriez P. Semi-Join Algorithms for Distributed Database Machines. In Schneider J.-J. (Ed.) Distributed Data Bases, North-Holland, 1982, pp. 23–37.
12. Williams R., Daniels D., Hass L., Lapis G., Lindsay B., Ng. P., Obermarck R., Selinger P., Walker A., Wilms P., and Yost R. R*: An overview of the Architecture. IBM Research Lab, San Jose, CA, 1981.

Distributed Query

- ▶ Distributed Join
- ▶ Distributed Query Processing

Distributed Query Optimization

Stéphane Bressan
National University of Singapore, Singapore, Singapore

Synonyms

Query optimization in distributed database systems

Definition

Distributed query optimization refers to the process of producing a plan for the processing of a query to a distributed database system. The plan is called a query execution plan. In a distributed database system, schema and queries refer to logical units of data. In a relational distributed relation database system, for instance, logical units of data are relations. These units may be be fragmented at the underlying physical level. The fragments, which can be redundant and replicated, are allocated to different database servers in the distributed system.

A query execution plan consists of operators and their allocation to servers. Standard physical operators, usually implementing the data model's algebra, are used to process data and to consolidate intermediary and final results. Communication operators realize the transfer, sending and receiving, of data from one server to another. In the case of fragmentation the plan uses fragments instead of logical data units. In the case of replication, the plan defines the choice of replicas.

Distributed query optimization, like non-distributed query optimization, involves the enumeration of candidate query execution plans and the selection of an optimal or satisfactory plan with respect to a cost model.

A cost model for distributed query optimization involves not only local processing cost, i.e., the cost of central unit processing and of input/output operations but also the cost of communications.

A distributed query optimization algorithm selects an optimal or satisfactory plan by exploring parts of the combinatorial search space defined as the set of possible query execution plans. The function of cost to be optimized is called the objective function.

Historical Background

The three reference distributed relational database management systems are SDD-1 [1], Distributed Ingres [3] and System R^* [4]. Their respective distributed query optimization algorithms are representative of the typical possible strategies.

The first distributed query optimization algorithm is Wong's "hill climbing" algorithm [10]. The algorithm greedily tries and improves an initial feasible solution and reaches, as the name indicates, a local optimum. The algorithm is further refined in the distributed query optimization algorithm of SDD-1 where it is extended to include semijoin programs. SDD-1 supports neither replication nor fragmentation. While Wong's algorithm works with a general objective function, SDD-1's implementation considers total communication cost only. Clearly, the overall focus of SDD-1's optimization approach is to reduce the volume of data transmitted. SDD-1 distributed query optimization is static.

Distributed Ingres' distributed query optimization algorithm [3] deterministically explores the search space of possible plans by making local optimization decisions at each step. Distributed Ingres supports horizontal fragmentation. The objective function is a weighted combination of total time cost and response time. It therefore includes both local and communication cost. Distributed Ingres' distributed query optimization is dynamic and therefore can benefit by the knowledge of the actual size of intermediary results.

System R*'s distributed query optimization algorithm is described in [7]. The algorithm exhaustively explores the search space of all possible query execution plans using dynamic programming to prune regions. The implementation of the algorithm in System R* supports neither replication nor fragmentation. The objective function is total cost and includes local processing and communication cost. System R*'s distributed query optimization is static.

Several approaches to dynamic query optimization have been proposed for parallel and distributed databases (see [5] for a rapid overview). Query scrambling [9], for instance, allows re-organization of the query execution plan during its execution in order to cope with unpredicted delays.

As an alternative to centralized optimization, the first system explicitly proposing an economical model for distributed query optimization is the Mariposa system [8].

While most of today's commercial database management system vendors offer distributed versions of their software, the continuous technological developments and the new application perspectives constantly compel extension and revision of existing distributed query optimization techniques in order to meet the needs of the new systems and applications.

Both textbooks [2,6] present and discuss in details the state of the art of distributed database technology in general and distributed query optimization in particular in the 1980s and 1990s, respectively. The survey [5] is a more current account of the development of these technologies and techniques.

Foundations

A distributed database management system consists of several dispersed database servers interconnected by an either local or wide area network. The database servers can be homogeneous or heterogeneous in hardware and software. The servers and network can be more or less autonomous.

Fragmentation and Replication

Data in a distributed database application may be fragmented. Logical units of data as they appear in the schema of the application may be further decomposed. Fragmentation can generally be expressed by means of logical operators. In the case of relational distributed database applications, a vertical fragment of a relation is the result of a relational projection, while a horizontal fragment is the result of a relational selection. Fragments are allocated to different servers in the distributed database management system. Fragmentation should be a lossless decomposition. However fragments can be redundant and replicated.

Fragmentation independence (often referred to as "fragmentation transparency") assumes that programmers write programs in terms of the logical schema of the application and ignore the details of fragmentation, redundancy and replication, and allocation. This property should be guaranteed for all design and programming purposes except, if necessary, for performance tuning.

The choice among possible fragments and replicas, the allocation of operations together with the cost and time needed for processing, as well as communication with local storage and network communication define the search space for the distributed query optimization problem and its objective function. Given a query to a distributed database system, its optimization, the distributed query optimization problem, is the choice of a query execution plan given fragmentation, redundancy and replication, and allocation, which is optimal or satisfactory with respect to the cost model, i.e., tries and minimizes the objective function.

Plan Enumeration

A query execution plan for a query to a distributed database management system is the decomposition of the query into local plans accessing fragments together with a global plan for the coordination of local executions, communications, and transmission, consolidation and production of results.

Since fragments and replicas are usually defined by logical operations, it is possible to rewrite the original query into a query in the same language involving only actual fragments and replicas. This decomposition is straightforwardly done by replacing each logical unit by a sub-query defining its reconstruction from fragments and replicas. For instance, in the simplest cases, a horizontally decomposed relation is the union of its fragments; a vertically decomposed relation is the natural join of its fragments. Because of redundancy and replication, there can be several possible decompositions using alternative fragments and replicas and therefore several alternative rewritings.

As with non-distributed query optimization, there may be several candidate access and execution methods, therefore several possible physical operators, for each logical operation. Furthermore, the data necessary to an operation may not be available at the server where the operation is allocated. Data needs to be transmitted from the server where it is located to the server where the operation is executed. When data transmission is a dominant cost, an extended set of operations can be considered to reduce data before transmission: this is how the semijoin operator is used in semijoin programs to favor transmission of small amounts of data and local operations to transmission of large amounts of data. One strategy consists in trying and fully reducing (when queries are acyclic) fragments before transmitting them.

A query execution plan is a tree of operators for processing, as in non-distributed database management systems, and, specifically to distributed database management systems, for communication and transmission of data. Every operator must be allocated to one database server. Non-distributed query optimizers are generally concerned with left- (or right-) deep trees of operators, i.e., sequence of operations. Distributed query optimizers need to consider bushy trees which contain opportunities for parallel execution. Indeed sibling sub-trees, when allocated to different servers, can be executed in parallel.

In summary, the plan enumeration involves the enumeration of alternative fragments and replicas, the choice of local execution methods, operations and communications, and of the choice of order and allocation of execution. As with non-distributed query optimization, the size of the search space is combinatorial. Strategies exploring the entirety of the search space are unlikely to be efficient. One can either use dynamic programming techniques to prune the search space yet finding an optimal solution, or use heuristics or randomized algorithms, such as simulated annealing, to find near or local optima. The exploration of the search space is guided by the objective function, i.e., the cost associated to each plan.

Total Cost Model and Response Time

For each query execution plan the optimizer tries and estimates its total cost, response time or a combination of both. This is the objective function of the optimization algorithm. The optimizer chooses the plan which minimizes the cost, the response time or finds the best combination or compromise.

At a coarse granularity, total cost is the sum of local cost and communication cost. At a finer granularity, the total cost model can be seen as an extension of the standard cost model for centralized databases to which is added the cost of network communications. The total cost is the sum of the unit cost of central unit processing, C_{cpu}, multiplied by the number of cycles, *#cycles*, and the unit cost an input/output operation, $C_{I/O}$, multiplied by the number of such operations, *#I/O*, with the cost of communications.

$$C_{total} = C_{cpu} \times \#cycles + C_{I/O} \times \#I/O + C_{comm}$$

It is commonly assumed that the cost of communication is linear in the number of bytes transmitted. The cost of communication combines the cost of initiating a message C_{s_mess}, the cost of receiving a messages C_{r_mess}, the number of messages *#mess*, the cost of transmitting one byte C_{tr_byte}, and the number of bytes transmitted *#bytes* as follows.

$$C_{comm} = ((C_{s\_mess} + C_{r\_mess}) \times \#mess) + (C_{tr\_byte} \times \#bytes)$$

Notice that the above formula requires the knowledge or estimation of the size of the elementary data units being processed as well as the selectivity of operations to estimate the size of intermediary results. The unit of cost is commonly time (in which case the model is referred to as the "total time cost model"). It is however probably more accurate to assume a Dollar cost since the total cost model is a measure of resource utilization.

The total cost model accounts for the usage of resources but does not account for the benefits of processing parallelism, input/output parallelism, and network parallelism naturally available in distributed database systems. Namely, processing, communication with local storage and network communications can happen in parallel since different machines interconnected by a physical network are involved.

Response time is the time, on the user's clock, needed by the system to execute a query. A simple example is a sequential scan operation (for instance in the case of a selection operation on a condition for which neither indexing nor fragmentation can be leveraged to rule out some fragments) of a relation fragmented into n fragments allocated to n different servers. The scan operation is decomposed into n scan operations: one for each individual fragments. If, for simplicity, it is assumed that all sequential scan operations start simultaneously, the response time is the time of the longest scan operation plus the time of longest request communication and the time of the last response communication. The total time is the sum of the times of individual sequential scans plus the sum of the times of all point-to-point (or broadcast, as possible in Distributed Ingres) communications.

Static versus Dynamic Distributed Query Optimization

Static query optimization is performed when compiling the application program without knowledge of the actual size of the elementary units of data. Dynamic query optimization is performed just before query execution thus allowing the optimizer to take such knowledge into account. A simple solution consists in statically generating several candidate plans for the same query and then dynamically choosing the plan with the best potential for efficient execution. Another solution, intermediary between static and dynamic query optimization prepares a query plan which is later allocated.

Furthermore modern distributed database applications leveraging the Internet are confronted with the unpredictable nature of a best effort communication network and of typically autonomously managed servers. Unpredictable delays result in queries executions being blocked sine die. Query scrambling is a dynamic approach that considers re-planning of the query during its execution. Initially a query execution plan for the distributed query is produced as described above. Re-planning takes the form of the rescheduling of transmissions and physical algebraic operators. If a plan execution is blocked because of a stalled communication, query scrambling attempts to perform other data transmission and operations initially scheduled for later in the original query execution plan. Query scrambling is a form of re-optimization at execution time (on-the-fly).

Global versus local, Centralized versus Distributed Query Optimization, andEconomical Models

The optimization process consists of both global (to the system) and local (to each database server) optimization. Global optimization is usually centralized at the server initially receiving the query. Global optimization needs to know or estimate the values required for the estimation of local cost, response time and communication cost, such as, for instance, the selectivity of sub-trees locally executed in order to estimate the number of bytes transmitted in the result. Details of local optimization, such as the choice of access methods, can be left to the component database servers.

This assumes that the database server performing the global optimization has or can collect sufficient information about the network and the other servers to properly estimate costs and to devise and choose a plan. Yet this information (including statistics about the network and local processing) may be difficult to collect and maintain if servers and network are autonomous, if the overall system is dynamic in nature, as well as if external independent factors influence its performance (e.g., general traffic of a public network).

Naïve attempts to distribute the classical distributed query optimization algorithms do not scale and rapidly lead to unsatisfactory sub-optimal solutions. One family of models for distributed optimization among autonomous agents is the one of economical models. These models try and simulate a free market regulated by supply and demand with negotiations based on bidding, auctioning or bargaining.

In a simple and simplified model with bidding for "distributed" distributed query optimization, the usage of resources is first monetized. Users (or user programs) assign a budget to their query and ask for the most economical execution available. The budget of a query is a decreasing function of response time, thus naturally expressing the acceptable compromise between resource usage and response time for the particular user. Brokers perform the (possibly partially) global by asking for and purchasing resources but remaining below the budget curve. Servers can actively try and maximize profit from selling resources by bidding for execution or buying or selling data.

Future Directions

The evolution of technology together with new applications places the optimization requirements for different new and future distributed database applications at various points on the spectrum between approaches conventionally referred to as centralized databases, parallel databases, distributed database, federated databases and peer-to-peer databases. Varying degrees of autonomy and heterogeneity, together with the relative importance of the elements defining cost change the requirements. For instance, while peer-to-peer database applications may be primarily concerned with response time and network congestion, mobile ad hoc network databases applications (distributed databases applications on mobile devices) may be primarily concerned with communication costs.

Cross-references

- ▶ Distributed Databases
- ▶ Distributed Database Design
- ▶ Distributed Query Processing
- ▶ Query Optimization
- ▶ Query Processing
- ▶ Semijoin
- ▶ SemiJoin Program

Recommended Reading

1. Bernstein P.A., Goodman N. Wong E. Reeve C.L., and Rothnie Jr. Query processing in a system for distributed databases (SDD-1). ACM Trans. Database Syst., 6(4):602–625, 1981.
2. Ceri S. and Pelagatti G. Distributed Databases Principles and Systems. McGraw-Hill, New York, NY, USA, 1984.
3. Epstein R.S., Stonebraker M., and Wong E. Distributed query processing in a relational data base system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1978, pp. 169–180.
4. Haas L.M., Selinger P.G., Bertino E., Daniels D., Lindsay B.G., Lohman G.M., Masunaga Y., Mohan C., Ng P., Wilms P.F., and Yost R.A. R*: a research project on distributed relational dbms. IEEE Database Eng. Bull., 5(4):28–32, 1982.
5. Kossmann D. The state of the art in distributed query processing. ACM Comput. Surv., 32(4):422–469, 2000.
6. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 2nd edn. 1999.
7. Selinger P.G. and Adiba M.E. Access Path Selection in Distributed Database Management Systems. In Proc. Int. Conf. on Data Bases, 1980, pp. 204–215.
8. Stonebraker M., Devine R., Kornacker M., Litwin W., Pfeffer A., Sah A., and Staelin C. An economic paradigm for query processing and data migration in mariposa. In Proc. 3rd Int. Conf. Parallel and Distributed Information Systems, 1994, pp. 58–67.
9. Urhan T., Franklin M.J., and Amsaleg L. Cost based query scrambling for initial delays. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 130–141.
10. Wong E. Retrieving dispersed data from SDD-1: a system for distributed databases. In Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, 1977, pp. 217–235.

Distributed Query Processing

Kai-Uwe Sattler
Technical University of Ilmenau, Ilmenau, Germany

Synonyms

Distributed query; Distributed query optimization

Definition

Distributed query processing is the procedure of answering queries (which means mainly read operations on large data sets) in a distributed environment where data is managed at multiple sites in a computer network. Query processing involves the transformation of a high-level query (e.g., formulated in SQL) into a query execution plan (consisting of lower-level query operators in some variation of relational algebra) as well as the execution of this plan. The goal of the transformation is to produce a plan which is equivalent to the original query (returning the same result) and efficient, i.e., to minimize resource consumption like total costs or response time.

Historical Background

Motivated by the needs of large companies and organizations that manage their data at different sites, distributed database systems are subject of research since the late 1970s. In these years, three important prototype systems were developed which already introduced fundamental techniques of distributed query processing. The first system was SDD-1 [1], developed at Computer Corporation of America between 1976 and 1980, that run on PDP-10 mainframes connected by Arpanet. SDD-1 pioneered among others optimization techniques and semijoin strategies. The two others, Distributed INGRES [8] (University of Berkeley, 1977–1981) and R* [11] (IBM Research, 1981–1985) contributed further techniques for distributed databases but none of these approaches was commercially successful. In [10], Stonebraker and Hellerstein explained this by the lack of an adequate and stable networking environment at this time and the slow market acceptance of distributed DBMS. Today, this has radically changed: the major DBMS vendors offer solutions that include query processing facilities allowing to query and combine remote tables. Typically, this is achieved by extensions of the standard query processor such as special operators for executing remote (sub-)queries, adding distributed queries to the search space of the query optimizer as well as cost functions dealing with network communication.

Besides these classic techniques, several new approaches have been developed since the early prototypes. The Mariposa system [9] was based on a micro-economic model for query processing where sites (servers) bid in an auction to execute parts of a query which is payed by the query issuer (client). This allows different bidding strategies for each server in order to maximize its own profit and may better work in a non-uniform, wide-area environment with a large number of sites. Another line of research addressed the problem of changing or unpredictable communication times and data arrival rates in a distributed environment by dynamic or adaptive approaches. Proposed solutions range from operator-level adaptation by introducing dedicated operators (e.g., join strategies) to adaptation at query level by interleaving of query optimization and execution [4].

Foundations

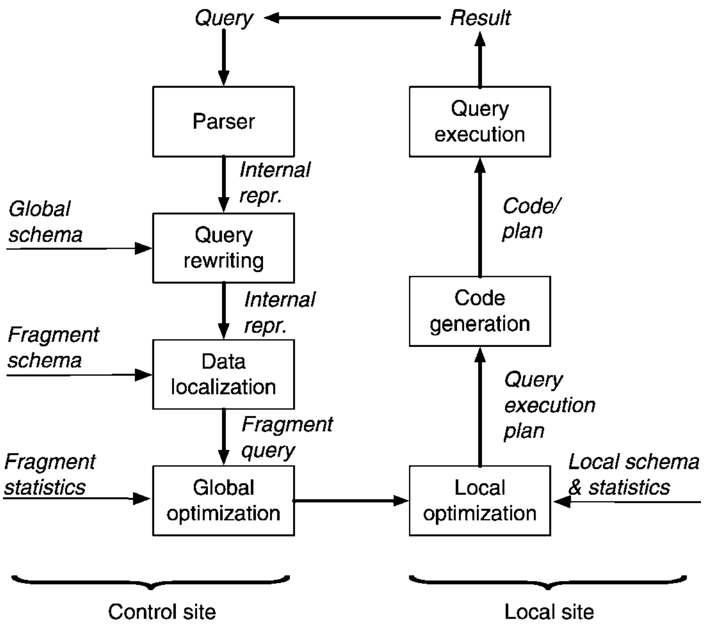
The subject of distributed query processing is to answer a query on data managed at multiple sites. This involves several steps for transforming a high-level query into an efficient query execution plan and opens various alternative ways for executing query operations of this plan. In this rest of this section, these phases are described and the most important techniques for each step are discussed.

Phases of Distributed Query Processing

The procedure of distributed query processing (Fig. 1) follows the approach of conventional query processing in database systems: In the first phase, a given query is parsed and translated into an internal representation (e.g., a query graph with nodes representing operators of an extended relational algebra). Next in the rewriting phase, the query is further transformed by applying equivalence rules in order to normalize, unnest, and simplify it without taking physical aspects such as cardinalities of relations, existence of indexes etc. into account. In the next step, the query is optimized by replacing the logical query operators by specific algorithms (plan operators) and access methods as well as by determining the order of execution. This is typically done by enumerating alternative but equivalent plans, estimating their costs and searching for the best solution. Finally, the chosen query execution plan is sent to the query execution engine, possibly after generating executable code.

In the distributed case, the phases of rewriting and optimization are extended. During query rewrite global relations referenced in the query have to be replaced by the corresponding fragmentation and reconstruction expressions resulting in fragment queries. Furthermore, reduction techniques are applied to eliminate redundant fragment queries or queries producing empty results. These steps are called *data localization.*

The optimization phase is split into a global step performed at the control site where the query was submitted and a local step which is done at each site maintaining a fragment relation referenced in this query. *Global optimization* involves the decision at which site the operation is to be executed as well inserting communication primitives into the plan. Global optimization typically ignores access methods like index usage for fragment relations – this is part of the local optimization.



Distributed Query Processing. Figure 1. Phases of Distributed Query Processing.

Data Localization

Usually, a distributed database is designed in a way that fragments of a global relation are stored at different sites. These fragments are defined by fragmentation expressions expressed for example as relational queries. In order to provide location and fragment transparency a query on a global relation R has to be transformed to a query that accesses only the fragments of R. This is achieved by replacing the global relation by the expression for reconstructing R from the fragments. Assuming a global relation $R(A, B, C)$ which is horizontally fragmented into R_1, R_2, R_3 as follows:

$$\begin{aligned} R_1 &= \sigma_{A<100}(R) \\ R_2 &= \sigma_{100\leq A\leq 200}(R) \\ R_3 &= \sigma_{A>200}(R) \end{aligned}$$

Then, R can be reconstructed by $R = R_1 \cup R_2 \cup R_3$. Using this fragmentation information, a global query $q_1 := \sigma_{A>150}(R)$ is now transformed into the query

$$q_1' := \sigma_{A>150}(R_1 \cup R_2 \cup R_3)$$

Obviously, this is not an efficient query because R_1 does not contribute to the result. Therefore, reduction techniques are applied which exploit equivalence transformations of relational algebra expressions in combination with rules for identifying fragment queries. This produces empty results due to useless selections or joins. These rules work mainly by analyzing predicates of fragment expressions and queries and finding contradictions [2].

For instance, for horizontal fragmentation the following rule can be used:

$$\sigma_{p_i}(R_j) = \emptyset \text{ if } \forall \mathrm{t} \in \mathrm{R} : \neg(\mathrm{p_i(t)} \wedge \mathrm{p_j(t)}) \tag{1}$$

Considering query q'_1, the selection operator can be pushed down to the fragments. Then, the rule finds the contradicting predicate $A < 100 \wedge A > 150$ resulting in an empty result:

$$\underbrace{\sigma_{A>150}(R_1)}_{\emptyset} \cup \sigma_{A>150}(R_2) \cup \sigma_{A>150}(R_3)$$

Thus, the fragment query $\sigma_{A>150}(R_1)$ can be eliminated.

A similar technique is used for identifying useless joins using the following rule:

$$\begin{aligned} R_i \bowtie R_j = \emptyset \text{ if } \forall \mathrm{t_1} \in \mathrm{R_i}, \forall \mathrm{t_2} \in \mathrm{R_j} : \\ \neg(p_i(t_1) \wedge p_j(t_2)) \end{aligned} \tag{2}$$

Assume a second relation $S(A, D)$ with the fragments:

$$\begin{aligned} S_1 &= \sigma_{A\leq 200}(S) \\ S_2 &= \sigma_{A>200}(S) \end{aligned}$$

A local query $q_2 := R \bowtie S$ has to be transformed into:

$$q_2' := (R_1 \cup R_2 \cup R_3) \bowtie (S_1 \cup S_2)$$

By distributing the join over the unions and applying rule (2) query q'_2 can be rewritten into the following form where three expressions can be identified as useless joins due to contradicting predicates:

$$\begin{aligned} &(R_1 \bowtie S_1) \cup (R_2 \bowtie S_1) \cup \underbrace{(R_3 \bowtie S_1)}_{\emptyset} \cup \underbrace{(R_1 \bowtie S_2)}_{\emptyset} \\ &\cup \underbrace{(R_2 \bowtie S_2)}_{\emptyset} \cup (R_3 \bowtie S_2) \end{aligned}$$

This is done by looking only at the predicates of the fragment expressions and not at the relations.

For vertical fragmentation the main goal of reduction is to identify useless projections. Assuming the following fragmentation of relation R:

$$\begin{aligned} P_1 &= \pi_{A,B}(R) \\ P_2 &= \pi_{A,C}(R) \end{aligned}$$

the reconstruction expression is $R = P_1 \bowtie P_2$. Thus, a query $q_3 := \pi_{A,C}(R)$ has to be transformed into the following query:

$$q_3' := \pi_{A,C}(P_1 \bowtie P_2) = \pi_{A,C}(P_1) \bowtie \pi_{A,C}(P_2)$$

A projection on a vertical fragment is useless, if there are no common attributes between the projection and the fragment beside the key attribute. Note, this does not results in an empty relation – however, the result is redundant. This can be formulated in the following rule. Given a fragmentation $R_i = \pi_B(R)$ of relation $R(K, A_1,\ldots,A_n)$ where $B \subseteq \{K, A_1,\ldots,A_n\}$.

$$\pi_{K,B'}(R_i) \text{ is useless, if } B' \cap B = \emptyset \tag{3}$$

Using this rule for query q'_3, the projection on fragment P_1 can be identified as useless and eliminated from the query:

$$\underbrace{\pi_{A,C}(P_1)}_{\text{useless}} \pi_{A,C}(P_2)$$

For derived and hybrid fragmentation similar techniques exist which are described e.g., in [7].

Optimization of Distributed Queries

Whereas the local optimization step is the same as in a centralized database system, the global optimization comprises some special tasks.

As in conventional query optimization the *ordering of joins* is one of the main challenges. Typically, in a distributed DBMS bushy join trees are considered by the optimizer in addition to linear (left or right deep) trees. Bushy trees are a shape of join trees where both operands can be intermediate results instead of requiring one base relation operand. These bushy trees allow better exploiting parallelism in distributed queries. Furthermore, in a distributed database the problem of *site selection* is added, i.e., at which site an operation (e.g., a distributed join) is to be executed. In the simple case of joining two relations stored at different sites this can be decided simply by comparing the size of the relations: the smaller relation is sent to the other site. However, for a multi-way join the number of possible strategies is much larger and the sizes of the intermediate join results have to be estimated and considered for finding the optimal join ordering.

Basically, there are two fundamental options in site selection: either the data is retrieved from the storing site to the site executing the query (called *data shipping*) or the evaluation of the query is delegated to the site where the data is stored (*query shipping*). These two models are shown in Fig. 2. Data shipping can avoid bottlenecks on sites with frequently used data if many queries have to be evaluated by exploiting client resources. On the other hand, query shipping is the better choice if otherwise large amount of data has to be transferred to the processing site. Thus, a hybrid strategy is sometimes the best solution [3].

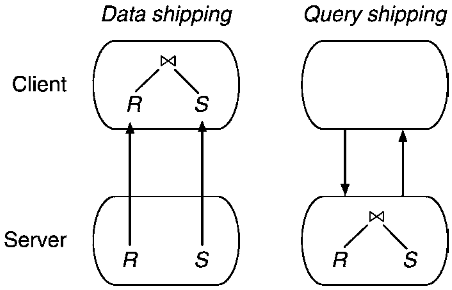
A further task in the global optimization step is to decide which strategies or algorithms are to be used to process the query operators. For example, a join between two relations stored at different sites can be processed in different ways: e.g., by shipping one relation to the site of the other relation as a whole (ship whole), by scanning one of the relations and fetching matching tuples from the other site, or by exploiting a semijoin strategy. Thus, during plan enumeration all these alternatives have to be considered by the optimizer.

Choosing the best query execution plan from the set of alternative plans requires to *estimate costs*. The conventional approach in centralized DBMS focuses on total resource consumption by estimating I/O and CPU costs. In a distributed environment, communication costs depending on the network speed have to be taken into account, too. However, the total resource consumption approach optimizes the throughput of a system, but ignores the inherent parallelism of a distributed query. For example, a scan on two fragments can be executed in parallel which reduces the query execution time to the halve. An alternative approach is the *response time model* which estimates the time between initiating the query and the receipting of the query result. Figure 3 illustrates the difference between these two models. Given two relations R and S, a query $R \bowtie S$ submitted at site 3 and assume that the costs are measured in time units.

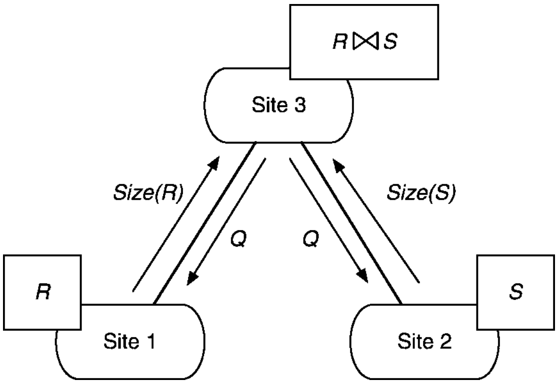
Then, the total resource consumption T_{total} of the query plan is

$$T_{total} = 2 \cdot T_Q + T_S(size(R) + size(S))$$

where T_Q is the time for sending a message (in this case the query request), T_s is the time for shipping a data



Distributed Query Processing. Figure 2. Data shipping vs. query shipping.



Distributed Query Processing. Figure 3. Total resource consumption vs. response time.

unit from one site to another, and *size*(*R*) is the size of relation *R*. In contrast, using the response time model the cost is

$$T_{response} = \max\{T_Q + T_S \cdot size(R), T_Q + T_S \cdot size(S)\}$$

This would prefer execution plans exploiting parallel processing as well as resources of other sites and therefore is better suited for distributed environments if communication is rather cheap.

Query Execution

Basically, the query execution phase of distributed processing is very similar to centralized DBMS. However, there are some specialties which are

- The need for send/receive operators for submitting fragment queries to other sites and shipping back results to the query issuer.
- Dedicated algorithms are required particularly for join processing, e.g., semijoin filtering or non-blocking hash joins.

Finally, there are some special optimization techniques which can be applied. First, in order to reduce the communication overhead caused by network latency, tuples are transferred in a block-wise manner (*row blocking*) instead of sending each tuple at a time. Second, *caching results* for reusing in future queries may help to reduce communication costs, too. Caching requires to balance the costs for loading and maintaining the data and the benefits of answering a query (partially) from the cache [6]. Furthermore, the cached data chunks have to be described in order to be able to check the containment of queries efficiently.

Key Applications

The main application of distributed query processing are classic distributed databases allowing to store and query data at different sites transparently. Another application domain are client/server database systems where the data is stored at a server and queries are submitted at client machines. Often, the computing resources of the client machines can be used for processing (portions of) the queries, too. Typical examples of such systems are object-oriented database systems but also middleware technologies such as application servers.

The existence of legacy databases of all flavors in many companies leads to the need of accessing and integrating them using queries in a standardized (e.g., relational) language on a composite schema. For this purpose, heterogeneous DBMS and mediators were developed that provide wrappers/gateways for encapsulating the system-specific transformation of queries and results between the global level and the local component system. Heterogeneous DBMS exploit distributed query processing techniques but have to deal with the heterogeneity of schema and data as well as with the possibly limited query capabilities of the component system. Commercial DBMS vendors have quickly adopted these techniques and offer now their own gateway solutions in addition to distributed query processing features, e.g., Oracle Database Gateway and IBM Information Server.

P2P systems and sensor networks are a current application area of distributed query processing. In these approaches, a network of nodes or sensors can be seen as a distributed database. Answering queries requires to retrieve and process data from different nodes using techniques described above. The challenges in these areas are mainly scalability (e.g., querying thousands of nodes) as well as dynamics and unreliability of the network.

Cross-references

▶ Distributed DBMS
▶ Distributed Join
▶ Distributed Query Optimization
▶ Query Processing

Recommended Reading

1. Bernstein P.A., Goodman N., Wong E., Reeve C.L., and Rothnie Jr., J.B. Query processing in a system for distributed databases (SDD-1). ACM Trans. Database Syst. 6(4):602–625, 1981.
2. Ceri S. and Pelagatti G. Correctness of query execution strategies in distributed databases. ACM Trans. Database Syst. 8 (4):577–607, 1983.
3. Franklin M., Jonsson B., and Kossmann D. Performance tradoffs for client-server query processing. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 149–160.
4. IEEE Computer Society. Bull. Tech. Committee Data Eng., 23 (2), June 2000.
5. Kossmann D. The state of the art in distributed query processing. ACM Comput. Surv., 32(4):422–469, 2000.
6. Kossmann D., Franklin M., Drasch G., and Ag W. Cache investment: integrating query optimization and distributed data placement. ACM Trans. Database Syst. 25(4):517–558, 2000.
7. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 2nd edn. Prentice-Hall, USA, 1999.

8. Stonebraker M. The Design and Implementation of Distributed INGRES. In stonebraker M. (ed.). The INGRES Papers, Addison-Wesley, Reading, MA, 1986.
9. Stonebraker M., Aoki P., Litwin W., Pfeffer A., Sah A., Sidell J., Staelin C., and Yu A. Mariposa: a wide-area distributed database system. VLDB J. 5(1):48–63, 1996.
10. Stonebraker M. and Hellerstein, J.M. Distributed Database Systems. In: M. Stonebraker and J.M. Hellerstein (eds.). Readings in Database Systems, 3rd edn. Morgan Kaufmann, San Francisco, CA, 1998.
11. Williams R., Daniels D., Hass L., Lapis G., Lindsay B., Ng. P., Obermarck R., Selinger P., Walker A., Wilms P., and Yost R. R*: An overview of the Architecture. Technical Report RJ3325, IBM Research Lab, San Jose, CA, 1981.
12. Yu C.T. and Chang C.C. Distributed query processing. ACM Comput. Surv., 16(4):399–433, 1984.
13. Yu C.T. and Meng W. Principles of Database Query Processing for Advanced Applications. Morgan Kaufmann, 1998.

Distributed Recovery

Kian-Lee Tan
National University of Singapore, Singapore,
Singapore

Synonyms

Recovery in distributed database systems; Recovery in distributed commit protocols; Recovery in replicated database systems

Definition

In a distributed database system (DDBS), failures in the midst of a transaction processing (such as failure of a site where a subtransaction is being processed) may lead to an inconsistent database. As such, a recovery subsystem is an essential component of a DDBS [14]. To ensure correctness, recovery mechanisms must be in place to ensure transaction atomicity and durability even in the midst of failures.

Distributed recovery is more complicated than centralized database recovery because failures can occur at the communication links or a remote site. Ideally, a recovery system should be simple, incur tolerable overhead, maintain system consistency, provide partial operability and avoid global rollback [6].

Historical Background

A DDBS must be reliable for it to be useful. In particular, a reliable DDBS must guarantee transaction atomicity and durability when failure occurs. In other words, a committed transaction's actions must persist across all the sites at which it is processed, and an aborted transaction's actions must not be allowed to persist. A transaction is aborted either explicitly (through an abort command) or implicitly as a result of a failure prior to its completion (through the recovery mechanism).

However, in a DDBS, besides traditional failures types which occur in a centralized system (such as media and power failures), new types of failures may occur, e.g., communication link failure, network partitioning, delayed/lost messages and remote site failure. While existing recovery mechanisms on centralized systems can be employed to handle the former type of failures, the latter kind is more complicated to deal with.

To ensure transaction atomicity, distributed commit protocols have been proposed. These include Two-Phase Commit and its variants [3,7,12] and Three-Phase Commit [13]. To recover from failures, a log is maintained at each site. As in centralized system, the log contains information such as the before and after view of an updated record. In addition, to facilitate distributed recovery, actions of the distributed commit protocol are also logged. In this way, each site knows the execution status of a transaction prior to the failure, and can determine if a transaction is committed or not before taking the necessary action. For example, for committed transaction, the log facilitates replaying the operations in the same order as they did before the failure; for uncommitted transactions, the operations has to be undone.

Log-based recovery protocols incur high overhead and may result in low transaction throughput. This is because log records have to be forced-written out to disks during logging, and have to be read from disks again during recovery. In order for a failed site to be recovered speedily, checkpointing techniques have also been proposed [1,10,11]. These schemes periodically maintain consistent states so that a failed site needs to rollback to a recent consistent state to reduce actions to be undone or redone.

There have been some efforts to reduce recovery overhead, and to perform online recovery so as not to disrupt transaction processing. In [15], an agent-based recovery protocol is proposed. The key idea is to buffer new database actions issued at the failed site (as it recovers using an existing log recovery scheme), and then replayed these buffered actions over the recovered state independently. In [5], distributed and redundant logging is used in which a site redundantly logs records to main memory and additionally to a remote site. In this way, the expensive forced-writes are avoided totally. In [9], HARBOR eliminates recovery logs (and hence the expensive disk write operations for log records) by exploiting replication/redundancy. This is done by keeping replicas consistent so that recovering a crashed site can be done by querying remote, online sites for missing updates.

All commercial systems today employ log-based roll-back recovery methods.

Foundations

A DDBS is vulnerable to failures that occur in centralized systems, such as power failure that may result in lost of data at a site. In addition, a DDBS is highly dependent on the availability of all sites, as well as their ability to communicate reliably with one another. When a site fails, sub-transactions running at that site may render the database inconsistent. Likewise, communication failures can result in the network becoming split into two or more partitions whose data may not be consistent. Even timeout may be deemed as a failure. As such, recovery is more complicated for DDBS.

In DDBS, log-based techniques are the pre-dominant schemes used to provide transaction atomicity and durability. Each site maintains a log which extends the widely used write ahead logging (WAL) scheme in centralized systems [3,4]. Under WAL, all modifications to records are written to a log before they are applied. Essentially, each log record contains a four-tuple $<tid, oid, v_{old}, v_{new}>$ where *tid* is the transaction identifier, *oid* is the object identifier, v_{old} and v_{new} are the old and new values of the updated object. There are also additional log records $<tid, start>$ and $<tid, commit>$, $<tid, abort>$ that capture the start of transaction *tid* and that it has committed or aborted. These log records enable the system to redo an operation if a transaction has committed (i.e., if the commit log record is in the log, then it means the transaction has committed, and the object can be updated to the new value v_{new}). Similarly, if the log commit/abort records are not found, then it means that failure has occurred before the transaction is committed/aborted. Hence, it may be necessary to undo some of the operations (by overwriting the object value with v_{old}). In this way, a database can be restored to a consistent state.

The extension arises because of the distributed commit protocols that are introduced to synchronize subtransactions of a transaction during the commit process. This calls for state transitions to be logged so that a failed site knows the status of a transaction's execution and can recover independently without communication from other sites. Consider the two-phase commit protocol which operates as follows: (i) In phase 1, the coordinator requests votes from all participants which may respond with a Yes vote (to commit the transaction) or No vote (to abort the transaction). (ii) In phase 2, the coordinator will make a global decision based on the votes – if all participants vote Yes, then the global decision is to commit the transaction; otherwise, the global decision is to abort it. Failures can occur at different state transitions.

The following log records will be generated at the coordinator: (i) before commit is initiated, the coordinator will generate a $<Tid, start-2PC>$ log which also contains the list of participants involved in processing the transaction *Tid*. (ii) When the coordinator is ready to commit the transaction (after receiving positive votes from all the participants), it will create a $<Tid, commit>$ log record; alternatively, for a global abort decision, a $<Tid, abort>$ log record is written instead. (iii) Finally, when all participants acknowledged that the commit operations have been performed, the coordinator will add a $<Tid, end-2PC>$ log.

On the other hand, at each participant, the following log records will be created: (i) a participant that is ready to commit upon receiving a request-for-vote message will log its vote, i.e., a log record $<Tid, Wait-State>$ is created; on the other hand, a participant that will not commit the transaction will create a log record $<Tid, Abort>$. (ii) Upon commit (after receiving the global commit decision), the participant will create a log commit record $<Tid, Commit>$. However, if the global decision is to abort the transaction, if the participant voted Yes, then $<Tid, Abort>$ will be created.

When there is a failure during the commit process, the operational sites will execute a termination protocol that will bring the transaction to a final state (either global abort or commit depending on the states of the operational sites). For two-phase commit, it is possible that the operational sites will need to wait for the failed sites to recover and become operational again before any further action can be taken.

For the failed site, there are two possible cases. First, the failed site is the coordinator. In this case, upon recovery, the coordinator checks its log. (i) If the failure happens prior to the initiation of the commit procedure (i.e., the failed site did not find the <*Tid*,*start* – 2*PC*> record in its log, then it can initiate the commit procedure. (ii) If the failure occurs in the waiting state (i.e., <*Tid*, *start* – 2*PC*> is found in the log but not <*Tid*, *commit*>), then it can restart the commit procedure. (iii) Finally, if the failure takes place in the commit/abort state, then if the coordinator has received all acknowledgements, it can complete successfully; otherwise, it will initiate the termination protocol as if it is an operational site.

Second, the failed site is a participant site. Again, there are several possible scenarios. (i) If the site failed before receiving any initiation request, then it can unilaterally abort the transaction. This is because the operational site would have aborted the transaction. (ii) If the site failed in the waiting state (i.e., <*Tid*, *Wait* – *State*> is found in the log but not <*Tid*, *commit*>), then the recovered site will initiate the termination protocol as an operation site. (iii) Finally, if the <*Tid*, *commit*> or <*Tid*, *abort*> or log record is found in the log, then no further action is necessary, as it means that the transaction has already committed/aborted.

Now, the recovery process for variants of the two-phase commit protocol is similar. In some case, one can optimize by allowing non-blocking of operational sites, in which case, the failed site may need to verify the status of the distributed commit operation with other operational sites upon recovery.

Key Applications

Since distributed systems will always encounter failure, distributed recovery methods are needed in all distributed systems to ensure data consistency. In particular, techniques developed for DDBS can be adapted and extended to emerging platforms like peer-to-peer (P2P) computing, mobile computing, middleware, and publish/subscribe systems.

Future Directions

Although distributed recovery is a fairly established topic, more work need to be done to support online recovery. In addition, the emergence of new distributed computing platforms (e.g., P2P computing, cloud computing, mobile computing, middleware) brings new challenges that require more effective solutions than what are currently available in the literature. For example, in P2P systems, the dynamism of node join and departure makes it difficult for a failed node to recover fully (since nodes that are operational at the time of the node's failure may no longer be in the network). It remains a challenge to device an effective recovery scheme in this context. As another example, consider mobile computing context (where nodes are mobile). Here, the base stations may be exploited to manage the logs [2]. However, because connections may be intermittent, many transactions will fail. A possible alternative is to allow a longer message delay to be tolerated. In both the P2P and mobile environments, a relax notion of serializability may be more practical. Yet another direction is to design recovery mechanisms for middleware. Initial effort has been done [7] but effective solutions are still lacking. Finally, a networked system is vulnerable to attacks that may corrupt the database. Recovering from such a state is an important subject that has not yet received much attention from the database community.

Cross-references

- ▶ Crash Recovery
- ▶ Logging and Recovery
- ▶ Three-Phase Commit
- ▶ Two-Phase Commit

Recommended Reading

1. Chrysanthis P.K., Samaras G., and Al-Houmaily Y.J. Recovery and performance of atomic commit processing in distributed database systems. In Recovery Mechanisms in Database Systems. Kumar Hsu Prentice-Hall, 1998. Chapter 13.
2. Gore M., Ghosh R.K. Recovery of Mobile Transactions. In Proc. DEXA 2000 Workshop, 23–27, 2000.
3. Gray J. Notes on data base operating systems. In Operating Systems – An Advanced Course. Bayer R., Graham R., Seegmuller G. (eds.). LNCS, Vol. 60, pp. 393–481, Springer, 1978.
4. Gray J. et al. The recovery manager of the system R database manager. ACM Comput. Surv., 3(2):223–243, 1981.
5. Hvasshovd S., Torbjornsen O., Bratsberg S., Holager P. The clustra telecom database: high availability, high throughput, and real-time response. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 469–477.
6. Isloor S.S. and Marsland T.A. System recovery in distributed databases. In Proc. 3rd Int. Computer Software Applications Conf., 1979, pp. 421–426.
7. Jimenez-Peris R., Patino-Martinez M., and Alonso G. An algorithm for non-intrusive, parallel recovery of replicated data and its correctness. In Proc. 21st Symp. on Reliable Distributed Syst., 2002, pp. 150–159.
8. Lampson, B. and Sturgis H. Crash recovery in a distributed data storage system. Technical report, Computer Science Laboratory, Xerox Palo Alto Research Center, California, 1976.

9. Lau E. and Madden S. An integrated approach to recovery and high availability in an updatable, distributed data warehouse. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 12–15.
10. Lin J. and Dunham M.H. A low-cost checkpointing technique for distributed databases. Distrib. Parall. Databases, 10(3):241–268, 2001.
11. Lomet D. Consistent timestamping for transactions in distributed systems. Tech. Report CRL90/3, Cambridge Research Laboratory, Digital Equipment Corp., 1990.
12. Mohan C., Lindsay B., and Obermarck R. Transaction management in the R* distributed data base management system. ACM Trans. Database Syst., 11(4):378–396, 1986.
13. Skeen D. Non-blocking commit protocols. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1981, pp. 133–142.
14. Özsu M.T. and Valduriez P. Principles of distributed database systems (2nd edn). Prentice-Hall, 1999.
15. Wang Y. and Liu X. Agent based dynamic recovery protocol in distributed databases. In Proc. 2nd Int. Symp. on Parallel and Distributed Computing, 2003.

Distributed Sensor Fusion

► Data Fusion in Sensor Networks

Distributed Source Coding

► Data Compression in Sensor Networks

Distributed Spatial Databases

Panos Kalnis
National University of Singapore, Singapore,
Singapore

Definition

Distributed spatial databases belong to the broad category of distributed database systems. Data reside in more than one sites interconnected by a network, and query processing may involve several sites. A site can be anything from a server to a small mobile device. The broad definition covers many research areas. This entry gives an overview of the following sub-categories: (i) Distributed spatial query processing, which focuses mainly on spatial joins. (ii) Distributed spatial indexes (e.g., a distributed version of the R-tree). (iii) Spatial queries in large distributed systems formed by devices such as PDAs, mobile phones, or even sensor networks.

Historical Background

Similar to relational databases, in spatial databases the most important operator is the spatial join. In relational databases, distributed joins are often implemented by using the semijoin operator. Let R and S be relations residing in two different sites R_{site} and S_{site}. First R_{site} calculates R' which is the projection of R on the join attribute. R' is transmitted to S_{site} and is joined with S; the result is called *semijoin*. That result is sent back to R_{site} where it is joined with R to produce the final join between R and S.

The semijoin concept can be adapted for joins between spatial datasets. However, the following characteristics of spatial datasets must be taken into account:

1. The relational semijoin is based on the assumption that the projected relation R' will be much smaller than R (since it contains fewer attributes and there are no duplicates), leading to lower transmission cost. In spatial datasets, the equivalent to the join attribute is the spatial object. The size of each spatial object (typically a polygon) may be in the order of hundreds or thousands of bytes, and usually dominates the size of other attributes. Therefore, projecting on the join attribute is not likely to reduce the transmission cost.
2. Evaluation of spatial relationships, such as containment, intersection and adjacency between two polygons, is complex and expensive (in terms of CPU and I/O cost), compared to testing the join condition in relational databases.

To address these issues, existing work [1,14] implements distributed spatial joins by using *approximations* of the spatial objects. This technique is common in spatial databases and involves two phases: the first phase operates on simple approximations of the objects (e.g., Minimum Bounding Rectangle – *MBR*) and produces a superset of the result. Since the approximations are simpler than the original objects, this phase is fast. The second phase removes the false hits of the intermediate result by operating on the exact polygons of the spatial objects.

The previous discussion assumes that the two sites are collaborating and allow access to their internal index structures. However, this is not always the case. Mamoulis et al. [10] and Kalnis et al. [5] assume that a mobile device is interested in the join of two spatial datasets whose sites do not collaborate. To avoid downloading the entire datasets, the mobile device interleaves the join with a statistics acquisition process which prunes parts of the space.

In addition to implementing distributed versions of spatial operators, several papers have focused on distributed spatial data structures (e.g., k-RP*;S [8] and hQT* [7]). A typical problem in distributed tree structures is that the server responsible for the root node is overloaded, since all queries must traverse the tree in a top-down fashion. Mouza et al. [12] proposed the SD-Rtree, which is a distributed version of the R-tree. Overloading the root is minimized by replicating a (possibly outdated) copy of the internal nodes of the tree in all clients. Several researchers [4,6,11] have also developed distributed spatial indices on top of Peer-to-Peer systems.

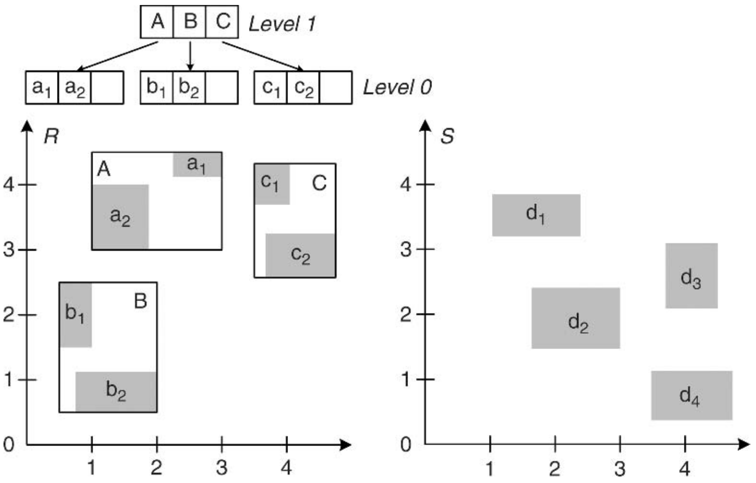
Finally, many applications assume a large number of mobile devices (e.g., PDAs, mobile phones) which store spatial data and ask spatial queries. For example, in the MobiEyes [3] system, mobile clients collaborate to answers continuous range queries. Other papers focus on broadcast-based wireless environments [15], which broadcast periodically the spatial index and data to all clients. Moreover, several sensor networks (e.g., Coman et al. [2]) optimize the query processing by exploiting the spatial properties (e.g., location, communication range and sensing range) of the sensors.

Foundations

The section on "Distributed Query Processing" discusses query processing in distributed spatial databases and focuses mainly on two cases: (i) the sites collaborate and are willing to share their internal indices (i.e., Tan et al. [14]) and (ii) the sites do not collaborate with each other (i.e., Mamoulis et al. [10]). Section "Distributed Spatial Indices" presents distributed spatial indices, the most representative being the SD-Rtree [12]. Finally, Section "Spatial Queries Involving Numerous Mobile Clients" discusses spatial queries that involve numerous and possibly mobile clients (e.g., smart phones, sensor networks, etc).

Distributed Query Processing

As mentioned above, distributed spatial joins are implemented by using approximations of the spatial objects. Tan et al. [14] investigate two approximation methods. The first one assumes that at least one of the datasets is indexed by an R-tree variant. In the example of Fig. 1, let R be the indexed dataset. The method uses the MBRs of the objects at level 0 (i.e., leaf level) or level 1 of the R-tree. Assuming that level 0 is used, R_{site} sends to S_{site} the following set of *MBRs*: $R' = \{a_1,a_2,b_1, b_2, c_1, c_2\}$. S_{site} performs window queries for each object in R' and returns to R_{site} the set $S' =\{ d_1,d_3\}$ of *objects* (i.e., polygons) which intersect with MBRs in R'. Finally R_{site} examines the polygons of the pairs (a_2, d_1) and (c_2,d_3) in order to remove any false hits. A second example assumes that level 1 of the R-tree is used. In this case R_{site} sends the MBR set $R' = \{A, B, C\}$ and S_{site} returns the set of polygons $S' = \{d_1, d_2, d_3\}$. Finally, R_{site} computes the join result from R and S'. It is noted that, if level 0 of the R-tree is used, R' typically contains many MBRs, allowing very refined search in S_{site}; consequently, S' usually contains a small number of objects. On the other hand, if level 1 is used, R' contains less MBRs

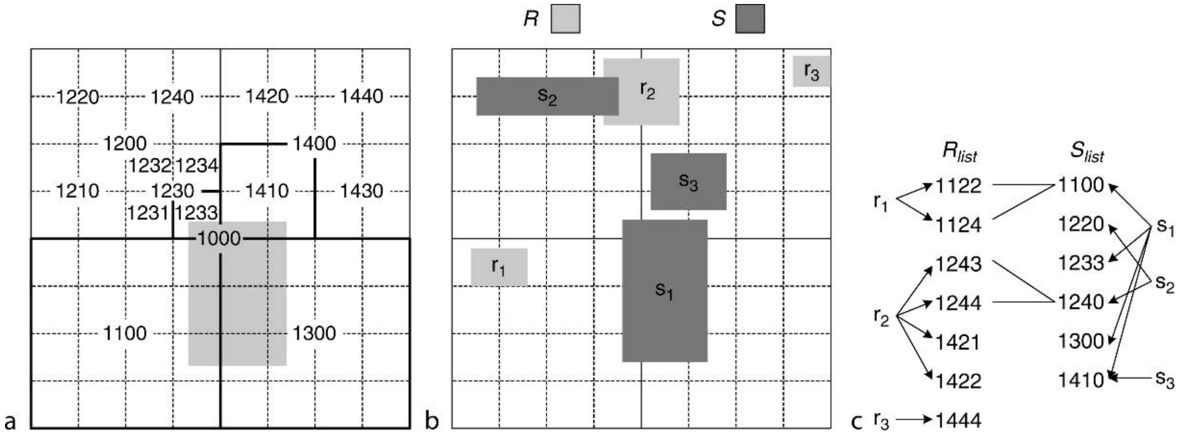


Distributed Spatial Databases. Figure 1. Distributed spatial join based on R-tree approximation.

but S' may include more objects (for instance, d_2 is a false hit). The choice of the appropriate level depends on the average size (in bytes) of the polygons in S.

The second approximation method used by Abel et al. [1] and Tan et al. [14] is similar to the one proposed by Orenstein [13]. The space is recursively divided into cells, and each cell is given a unique base-5 key. The order of the keys follows the *z*-ordering space filling curve. Each object is approximated by up to four cells at various levels of the curve. Figure 2a shows an example; the gray-shaded object is approximated by cells 1100, 1233, 1300 and 1410. The keys of all objects in R_{site} and S_{site} are sorted in ascending order, generating two lists: R_{list} and S_{list}. An example is shown in Fig. 2b and c. Each object has between one (e.g., r_3) and four keys (e.g., r_2); moreover, the keys of an object may not be consecutive in the ordered list (e.g., s_2). R_{site} transmits R_{list} to S_{site}. S_{site} uses *merge-join* to join the two lists. The join condition is cell-containment rather than key equality. For instance, the pair (1122,1100) is generated because cell 1122 is contained in cell 1100. The result may contain duplicates; for example (1122,1100) and (1124,1100) both represent the object pair (r_1,s_1). Duplicates are eliminated and S_{site} sends the semijoin result (together with the polygons of the corresponding S objects) to R_{site}; in the running example, the semijoin contains pairs (r_1, s_1) and (r_2, s_2). Finally, R_{site} eliminates the false hits (i.e., pair (r_1,s_1)) based on the exact object geometry. According to the experimental evaluation in [14], the total cost of the join (in terms of response time) is, in most cases, lower when using the *z*-ordering approximation, compared to the R-tree approach.

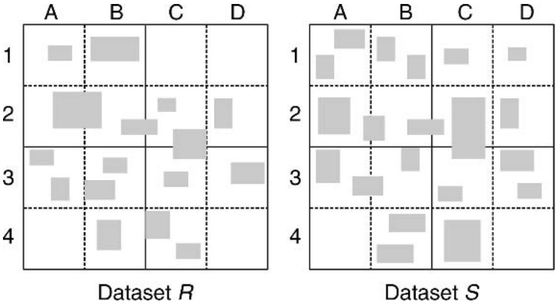
A related problem is studied by Mamoulis et al. [10]. Again, two spatial datasets R and S reside in R_{site} and S_{site}, respectively; however, the two sites do *not* collaborate. As an example, let R be a dataset which stores the location of hotels and other points of interest, whereas S stores information about the physical layers of the region (e.g., rivers, forests, urban areas, etc). Let u be a client with a mobile device (e.g., PDA, smart mobile phone) that asks the query "find all hotels which are at most 2*km* away from a forest." The query is a spatial join. However, neither R_{site} nor S_{site} can evaluate the join, because they do not collaborate. A typical solution is to use a *mediator*. Nevertheless, for ad-hoc queries, it is unlikely that there is a suitable mediator. Therefore, the query must be evaluated by the mobile device. It is assumed that the servers support a simple query interface and can answer *window* (i.e., find all objects in a region) and *count* queries (i.e., how many objects are within a region); the servers do not allow access to their internal indices. Moreover, since telecommunication providers charge by the amount of transferred data, u wants to minimize that quantity and does not consider the query cost at the servers. Figure 3 shows the two datasets. The client partitions conceptually the data by a 2 × 2 grid (i.e., *AB*12,*AB*34,*CD*12,*CD*34) and requests the *number* of objects in each quadrant. Since none of the quadrants is empty, all of them may contain joining pairs. Therefore, the client recursively partitions each quadrant and retrieves the new statistics (i.e., number of objects in each cell). Now, some cells (e.g., *C*1 in R_{site}) are empty; u eliminates these cells, since they cannot contain any solution.



Distributed Spatial Databases. Figure 2. Distributed spatial join based on *z*-ordering approximation.

For the remaining cells u may partition them again and ask for more statistics. However, if the distribution of objects in a cell is roughly uniform, further partitioning is unlikely to eliminate any cells. [3] presents a cost model to estimate when the cost of retrieving refined statistics is more than the potential savings due to pruning. At that point, u performs the spatial join. For each cell, if the number of objects is similar in both datasets, u downloads the objects of both cells and performs hash-based spatial join. On the other hand, if the number of objects differs drastically (e.g., the cell in R has much more objects than the corresponding cell in S), then u downloads only the objects from S and performs nested-loop join by sending a series of window queries to R. Kalnis et al. [5] developed more efficient algorithms for the same problem.

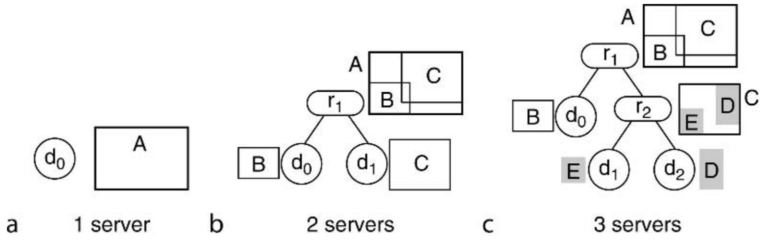
A similar method was used by Liu et al. [9] for evaluating k-nearest-neighbor queries. They assume a client-server architecture (only one server), where the server can execute only window queries. Therefore, the client must estimate the minimum window that contains the result. The authors propose a methodology that either estimates the window progressively, or approximates it using statistics about the data. Statistical information is assumed to be available at the client; hence there is no overhead for retrieving statistics.



Distributed Spatial Databases. Figure 3. MobiHook example.

Distributed Spatial Indices

Several researchers have studied distributed versions of common spatial indices. Litwin and Neimat [8] proposed a distributed version of the kd-tree, called k-RP*S, whereas Karlsson [7] developed hQT*, which is a distributed quad-tree. Both papers focus on point data. Recently, Mouza et al. [12] proposed the SD-Rtree, which is a distributed version of the R-tree capable of indexing regions. SD-Rtree is a binary balanced tree. Initially (Fig. 4a), the entire tree resides in one server S_0. There is a *data* node d_0, which contains all spatial objects and an associated MBR A, which encloses the objects of d_0. d_0 is a conceptual node and typically contains a large number of objects, limited only by the capacity of server S_0. The objects inside d_0 may be indexed by any spatial index (e.g., a local R-tree). During insertion, a split is performed if the capacity of S_0 is reached. A new server S_1 enters the system and receives approximately half of the objects; these are selected using the usual R-tree split algorithms. In Fig. 4b the objects remaining in d_0 are enclosed by MBR B, whereas a new data node d_1 is created in S_1; the corresponding MBR is C. Another node r_1 is created in S_1; r_1 has pointers to d_0 and d_1, together with their MBRs. r_1 is called *routing* node and is similar to the internal nodes of common R-trees. Figure 4c shows one more split of node d_1. A new server S_2 enters the system and creates a data node d_2 and a routing node r_2. The new MBRs of d_1 and d_2 are E and D, respectively. Finally, data is stored in three servers (i.e., S_0 stores d_0, etc) and the routing information is distributed in two servers (i.e., r_1 is in S_1 whereas r_2 is in S_2). Due to splits, the tree may become imbalanced; to achieve balance, node rotations similar to the classical AVL tree are performed.



Distributed Spatial Databases. Figure 4. SD-Rtree example.

The set of all routing nodes is called *image* of the tree. Each client application has a copy of the image. In order to query (or update) the spatial data, the client first searches the local image to identify the server that stores the required data, and contacts that server directly. This is crucial for a distributed structure, since it avoids traversing the tree through the root (i.e., it does not overload the root server). However, a client's image may be outdated. In this case, the wrongly contacted server forwards the query to the correct part of the tree and updates the client's image. Moreover, due to R-tree node overlap, a query may need to traverse multiple paths; some of them are redundant. To avoid this, if two nodes overlap, they are annotated with additional information about their intersection. In Fig. 5a, nodes *A* and *B* store the set of objects in their intersection; in this example the set is empty. The intersection set is updated only if necessary. For instance, the split of node *B* in Fig. 5b does not affect the intersection set. On the other hand, the extension of *D* in Fig. 5c changes the intersection set. If a query *q* arrives at *D*, it can be answered directly by *D* (i.e., without contacting the subtree of *F*), since the intersection set indicates that there is no matching object in *F*.

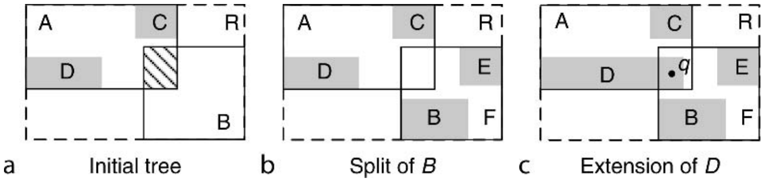
Several researchers have proposed distributed spatial indices on top of Peer-to-Peer (*P2P*) systems. For instance, Mondal et al. [11] proposed a P2P version of the R-tree, whereas Jagadish et al. [4] developed the VBI-tree, a framework for deploying multi-dimensional indices on P2P systems. Also, Kantere and Sellis [6] proposed a spatial index similar to quad-tree, on top of a structured P2P network. These approaches are based on different assumptions for the update and query frequency, and the stability (i.e., mean lifetime) of peers in the network. Therefore, it is not clear which approach is more suitable in practice.

Spatial Queries Involving Numerous Mobile Clients

There exist a variety of spatial distributed systems consisting of numerous mobile clients. Such systems are optimized for the low battery life and the scarce resources (e.g., storage, CPU) of the mobile devices. Gedik and Liu [3] proposed *MobiEyes*, a grid-based distributed system for continuous range queries. MobiEyes pushes part of the computation to the mobile clients, and the server is primarily used as a mediator. The notion of *monitoring regions* of queries was introduced to ensure that objects receive information about the query (e.g., position and velocity). When objects enter or leave the monitoring region, the server is notified. By using monitoring regions, objects only interact with queries that are relevant; hence they conserve precious resources (e.g., storage and computation).

Another architecture is based on the fact that wireless networks are typically broadcast-based. All data are broadcasted periodically and each client listens for its relevant data. In a wireless broadcast environment, an index called *air index* is commonly used to minimize power consumption. A mobile device can utilize the air index to predict the arrival time of the desired data. Therefore, it can reduce power consumption by switching to sleep mode for the time interval that no desired data objects are arriving. The intuition behind the air index is to interleave the index items with the data objects being broadcasted. Zheng et al. [15] proposed two air indexing techniques for spatial data based on the one-dimensional Hilbert Curve and the R-tree, respectively. Using those indices, they discuss how to support Continuous Nearest Neighbor queries in a wireless data broadcast environment.

A related subject is the processing of spatial queries in sensor networks. Such networks are made of a large number of autonomous devices that are able to store, process and share data with neighboring devices. The spatial properties of the sensors (e.g., location, communication range and sensing range) are typically exploited to route queries efficiently. For example, Coman et al. [2] propose a system that answers range queries (e.g., "find the temperature for each point in an area"). The system takes advantage of the fact that the



Distributed Spatial Databases. Figure 5. Example of overlap in SD-Rtree nodes.

sensing ranges of some sensors may completely cover other sensors; consequently, the latter do not need to be contacted.

Key Applications

Nowadays, the trend in databases is the separation of the location of data from the abstract concept of the database itself. Therefore a database may reside in more than one location, but can be queried as a continuous unit. This is made possible due to the increase of network speed. Numerous practical applications can benefit from distributed spatial databases. As an example, consider a spatial database that stores the location of hotels and other points of interest in an area, and a second database which stores information about the physical layers of the region (e.g., rivers, forests, urban areas, etc). Each database is useful by its own. Nevertheless, by combining the two, the value of the data increases, since the distributed system is now able to answer queries such as "find all hotels inside a forest." There are also applications where there is only one spatial dataset, but it is distributed in many servers (or peers). For example, each peer may monitor road congestion in its neighborhood. In order to find routes in a city that avoid congested roads, the spatial data in all peers must be indexed by a distributed data structure.

Cross-references

- ▶ Distributed Databases
- ▶ Distributed Join
- ▶ R-Tree (and Family)
- ▶ Spatial Indexing Techniques
- ▶ Spatial Join

Recommended Reading

1. Abel D.J., Ooi B.C., Tan K.-L., Power R., and Yu J.X. Spatial Join Strategies in Distributed Spatial DBMS. In Proc. 4th Int. Symp. Advances in Spatial Databases, 1995, pp. 348–367.
2. Coman A., Nascimento M.A., and Sander J. Exploiting Redundancy in Sensor Networks for Energy Efficient Processing of Spatiotemporal Region Queries. In Proc. Int. Conf. on Information and Knowledge Management, 2005, pp. 187–194.
3. Gedik B. and Liu L. MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 67–87.
4. Jagadish H.V., Ooi B.C., Vu Q.H., Zhang R., and Zhou A. VBI-Tree: A Peer-to-Peer Framework for Supporting Multi-Dimensional Indexing Schemes. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
5. Kalnis P., Mamoulis N., Bakiras S., and Li X. Ad-hoc Distributed Spatial Joins on Mobile Devices. In Proc. 20th Int. Parallel & Distributed Processing Symp., 2006.
6. Kantere V. and Sellis T.K. A Study for the Parameters of a Distributed Framework That Handles Spatial Areas. In Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases,. 2007, pp. 385–402.
7. Karlsson J.S. hQT*: A Scalable Distributed Data Structure for High-Performance Spatial Accesses. In Proc. Int. Conf. of Foundations of Data Organization, 1998, pp. 37–46.
8. Litwin W. and Neimat M.-A. k-RP*S: A Scalable Distributed Data Structure for High-Performance Multi-Attribute Access. In Proc. Int. Conf. on Parallel and Distributed Information Systems, 1996, pp. 120–131.
9. Liu D.-Z., Lim E.-P., and Ng W.-K. Efficient k Nearest Neighbor Queries on Remote Spatial Databases Using Range Estimation. In Proc. 14th Int. Conf. on Scientific and Statistical Database Management, 2002, pp. 121–130.
10. Mamoulis N., Kalnis P., Bakiras S., and Li X. Optimization of Spatial Joins on Mobile Devices. In Proc. 8th Int. Symp. Advances in Spatial and Temporal Databases, 2003, pp. 233–251.
11. Mondal A., Lifu Y., and Kitsuregawa M. P2PR-Tree: An R-Tree-Based Spatial Index for Peer-to-Peer Environments. In Proc. EDBT Workshops – P2P&DB, 2004, pp. 516–525.
12. du Mouza C., Litwin W., and Rigaux P. SD-Rtree: A Scalable Distributed Rtree. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 296–305.
13. Orenstein J.A. A Comparison of Spatial Query Processing Techniques for Native and Parameter Spaces. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 343–352.
14. Tan K.-L., Ooi B.C., and Abel D.J. Exploiting Spatial Indexes for Semijoin-Based Join Processing in Distributed Spatial Databases. IEEE Trans. Knowl. Data Eng., 12(6):920–937, 2000.
15. Zheng B., Lee W.-C., and Lee D.L. Search Continuous Nearest Neighbor on Air. In Proc. Int. Conf. on Mobile and Ubiquitous Systems: Networking and Services, 2004, pp. 236–245.

Distributed Storage Systems

▶ Peer-to-Peer Storage

Distributed Transaction Management

Wee Hyong Tok
National University of Singapore, Singapore, Singapore

Synonyms

Transaction management in distributed database systems

Definition

Distributed transaction management deals with the problems of always providing a consistent distributed database in the presence of a large number of transactions (local and global) and failures (communication link and/or site failures). This is accomplished through (i) distributed commit protocols that guarantee atomicity property; (ii) distributed concurrency control techniques to ensure consistency and isolation properties; and (iii) distributed recovery methods to preserve consistency and durability when failures occur.

Historical Background

A transaction is a sequence of actions on a database that forms a basic unit of reliable and consistent computing, and satisfies the ACID property. In a distributed database system (DDBS), transactions may be local or global. In local transactions, the actions access and update data in a single site only, and hence it is straightforward to ensure the ACID property. However, in global transactions, data from more than one site are accessed and updated. A global transaction typically spawns sub transactions at each of the sites in which data are accessed. When multiple global transactions are running, these sub transactions' processing may be interleaved. Thus, distributed transactions must be carefully managed to prevent errors that may corrupt the database. In addition, sites or communication link may fail and result in inconsistent database states across sites.

To ensure that a distributed database is always in a consistent state, the distributed transaction management system must guarantee the ACID property. Transaction atomicity is achieved through distributed commit protocols, e.g., Two-Phase Commit and its variants [8,10,11] and Three-Phase Commit [13]. These protocols allow sub transactions (running at different sites) of a global transaction to reach a final outcome of the execution. In this way, either all sub transactions commit (in which case the global transaction commits) or all sub transactions abort (in which case the global transaction aborts).

To ensure consistency and isolation properties, distributed concurrency control techniques have been designed. In particular, the notion of global serializability has been introduced to ensure that global transactions are serializable. These algorithms determine the management of synchronization primitives (e.g., locks, timestamps, serialization graphs), and the order in which database operations (i.e., read and write) are executed concurrently in a distributed manner. As such, anomalies such as the reading of uncommitted data, unrepeatable reads, and overwriting of uncommitted data can be prevented. Distributed concurrency control algorithms can be classified into lock-based [1,4,11,14,15], timestamp-based [12], and serialization graph testing (SGT) based [2,6,9]. Hybrid methods which use a combination of locks and timestamps are discussed in [5]. For lock-based scheme, (distributed) deadlocks have to be handled. Balter et al. [3] discusses the relationship between deadlock management and concurrency control algorithms.

Finally, when a transaction has committed, its effect must persist in the database regardless of failures. Similarly, if failure happens before a transaction commits, then none of its effects must persist. For example, a power failure that wipes out all main memory content should not nullify the effect of a committed transaction. Similarly, site failures should not affect the distributed commit protocol. To ensure durability and consistency, distributed recovery methods have been developed. These include log-based techniques [8] and checkpointing methods [7] (cross-reference to Distributed Recovery).

The two-phase commit protocol and the distributed 2PL concurrency control algorithm are implemented in the System R* [11] and NonSTOP SQL [15] systems. Distributed INGRES [14] uses the two-phase commit protocol and the primary copy 2PL for concurrency control. Log-based protocols are the main recovery methods employed in commercial systems.

Foundations

A distributed database system consists of a collection of database sites. A centralized database system is located at each of the sites. The database sites communicate with each other by sending messages via a communication network. A transaction consists of a sequence of operations that are performed on the data objects. Two operations, $o_i(x)$ and $o_j(x)$, which are accessing the shared data object x, are conflicting operations if either $o_i(x)$ or $o_j(x)$ is a write operation.

Distributed Transaction Management

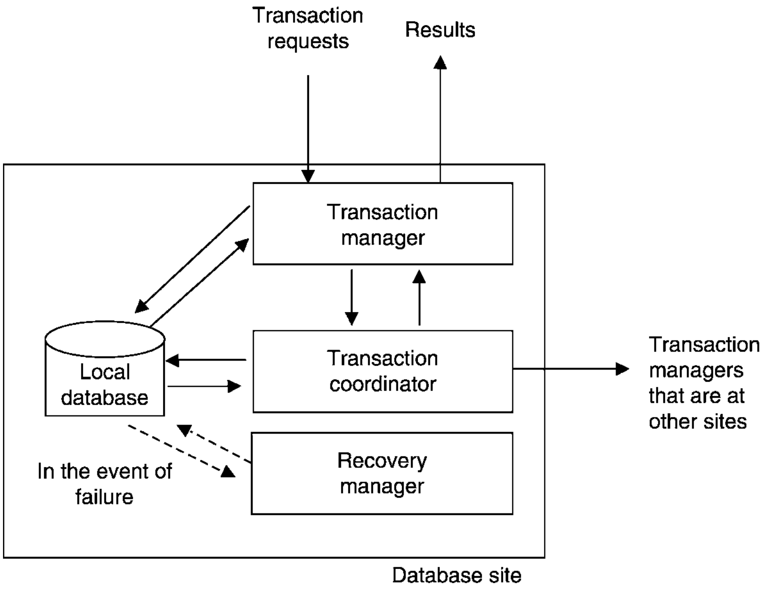
In order to handle both local and global transactions, each site of a distributed database system consists of the following components: transaction manager, transaction coordinator, and recovery manager. The transaction manager handles the execution of transactions at each site, and maintains a log to facilitate recovery. In order to ensure that the ACID properties are guaranteed when multiple transactions are concurrently executed, the transaction manager relies on concurrency control techniques. The transaction coordinator is used to plan and schedule sub transactions that are executed on multiple sites. In addition, the transaction coordinator determines whether the transactions that are executed on multiple sites are committed or aborted. The recovery manager is used to recover the database to a consistent state if failure occurs. Figure 1 shows the various components in the transaction management system at each database site.

Serializability Theory

Serializability theory provides the theoretical foundation for proving the correctness of a concurrency control algorithm, by showing that the overall results of executing a set of concurrent transactions is equivalent to executing the transactions in a serial manner. In serializability theory, this is referred to as a serializable schedule. A serializable schedule ensures the consistency of a database system.

Let T denote a set of concurrently executing transactions. The operations for the various transactions in T can be executed in an interleaved manner. A complete schedule, S, defines the execution order of all operations. A schedule, S', is a prefix of a complete schedule. Formally, $T = \{T_1, T_2,...,T_n\}$, and S is a partial order over T, with ordering relation, $<_S$, where: (i) $S = \bigcup_{i=1}^{n} T_i$, (ii) $<_S \supseteq \bigcup_{i=1}^{n} <_i$, and (iii) for any two conflicting operations, o_i and $o_j \in S$, either $o_i <_S o_j$ or $o_j <_S o_i$. The first condition states that S consists of the operations which belong to the set of transactions T. The second condition states that the ordering relation is a superset of the ordering relations of each of the transactions. The third condition states that the ordering of every pair of conflicting transaction is given by the ordering relation $<_S$. Two schedules, S_1 and S_2 defined over T are conflict equivalent if for each pair of conflicting operations o_i and o_j ($i \neq j$), if $o_i <_{S1} o_j$, then $o_i <_{S2} o_j$. A schedule is serializable if and only if it is conflict equivalent to a serial schedule.

In distributed database systems, global serializability is required. For example, consider two transactions, T_1 and T_2, that are initiated at two sites, site 1 and site 2 respectively. Suppose that both transactions access objects x and y. Moreover, suppose that x is stored at site 1, and y is stored at site 2, and that the constraint is that $x + y$ is a constant. Let $W_i(x)$ denote a write action on object x by transaction T_i. Suppose S_1 and S_2 are the local schedules at site 1 and site 2 respectively. Now, consider $S_1 = \{w_1(x), w_2(x)\}$ and $S_2 = \{w_2(y), w_1(y)\}$. Clearly, both S_1 and S_2 are locally serializable. However, while the serial schedule for S_1 is $T_1 \rightarrow T_2$, the serial schedule for S_2 is $T_2 \rightarrow T_1$. These schedules may violate the constraint and hence not globally serializable, if the



Distributed Transaction Management. Figure 1. Transaction management system at each database site.

sequence of actions happens in the following order: $w_1(x)$, $w_2(y)$, $w_2(x)$, $w_1(y)$. However, it has been shown that as long as all the local schedules at each site are serializable with the same serialization order, then the global schedule is serializable. Moreover, the serialization order of the global schedule corresponds to that of the local order. As an example, if $S_1 = \{w_1(x), w_2(x)\}$ and $S_2 = \{w_1(y), w_2(y)\}$, then the global schedule is serializable with $T_1 \rightarrow T_2$.

Concurrency control algorithms are used to ensure that the global schedule is serializable. For replicated distributed databases, more issues need to be considered before serializability theory can be applied. This is because even if the local schedules are serializable, the mutual consistency of the database needs to be considered. For replicated, distributed databases, a one-copy serializable global schedule is desired in order to ensure the mutual consistency of the replicated data. In addition to concurrency control, a replica control protocol (e.g., ROWA) is used to ensure that one-copy serializability can be achieved.

Key Applications

Distributed transaction management forms an integral part of distributed database systems. The concurrency control techniques, commit protocols, and recovery techniques can be applied and adapted for different types of distributed database systems.

Future Directions

The emergence of new computing platforms (e.g., Peer-to-Peer (P2P), and cloud computing) introduces new issues that need to be handled by distributed transaction manager. P2P database systems are inherently distributed systems, and have been studied extensively by the database community. In P2P systems, the absence of a global transaction manager introduces new challenges. In addition, nodes join or leave the P2P network. Transactions are often executed independently on each peer. The maintenance of data consistency in P2P database systems motivates the need for P2P-based concurrency control algorithms. For example, it may be necessary to maintain multiple versions of data objects and/or a weaker notion of serializability than the conventional notion of global serializability. Recovery is also complicated by the fact that a failed site may not find the operational sites (participating in its transactions) available when it recovers (these sites may have left the network). Thus, novel and online recovery mechanisms are needed. Cloud computing is an emerging computing platform for next-generation applications. It provides basic services such as storage, queuing, and computation. Hence, a data store can be easily built using these basic services. The storage services provided by the cloud can be perceived as a very large shared disk. Different applications which use the cloud will require different levels of concurrency control. The need to support concurrent access on the shared disk and different level of concurrency control motivates the need for concurrency control to be offered on a *à la carte* basis. In addition, recovery techniques for cloud computing have not been adequately studied in the literature.

Cross-references

- ▶ ACID Properties
- ▶ Distributed Architecture
- ▶ Distributed Concurrency Control
- ▶ Distributed Deadlock Management
- ▶ Distributed Recovery
- ▶ Three-Phase Commit
- ▶ Two-Phase Commit

Recommended Reading

1. Alsberg P. and Day J.D. A principle for resilient sharing of distributed resources. In Proc. 2nd Int. Conf. on Software Eng., 1976, pp. 562–570.
2. Badal D.Z. Correctness of concurrency control and implications for distributed databases. In Proc. 3rd Computer Software and Applications Conference, 1979, pp. 588–594.
3. Balter R., Berard P., and Decitre P. Why control of the concurrency level in distributed systems is more fundamental than deadlock management. In Proc. ACM SIGACT-SIGOPS 1st Symp. on the Principles of Dist. Comp., 1982, pp. 183–193.
4. Bernstein P.A. and Goodman N. Concurrency control in distributed database systems. ACM Comput. Surv., 13 (2): 185–221, 1981.
5. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Boston, MA, 1987.
6. Casanova M.A. The concurrency control problem for database systems. Lecture Notes in Computer Science, Springer Berlin 1981.
7. Chrysanthis P.K., Samaras G., and Al-Houmaily Y.J. Recovery and performance of atomic commit processing in distributed database systems, Chapter 13. In Recovery Mechanisms in Database Systems, V. Kumar, M. Hsu Prentice-Hall, Upper Saddle River, NJ, 1998.
8. Gray J. Notes on data base operating systems. In Operating Systems – An Advanced Course, R. Bayer, R. Graham, G. Seegmuller (eds.). Lecture Notes in Computer Science, vol. 60, Springer, Berlin, 1978, pp. 393–481.

9. Hadzilacos T. and Yannakakis M. Deleting completed transactions. In Proc. 5th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1986, pp. 43–46.
10. Lampson B. and Sturgis H. Crash recovery in a distributed data storage system. Technical report, Computer Science Laboratory, Xerox Palo Alto Research Center, CA, 1976.
11. Mohan C., Lindsay B.G., and Obermarck R. Transaction management in the R* distributed database management system. ACM Trans. Database Syst., 11(4):378–396, 1986.
12. Shapiro R. and Millstein R. Reliability and fault recovery in distributed processing. In OCEANS'77 Conf. Record, Vol. 9, 1977, pp. 425–429.
13. Skeen D. Non-blocking commit protocols. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 133–142.
14. Stonebraker M. and Neuhold E.J. A distributed database version of ingres. In Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, 1977, pp. 19–36.
15. The Tandem Performance Group. Tandem database group – nonstop sql: a distributed, high-performance, high-availability implementation of sql. In Proc. Second Int. Workshop on High Performance Transaction Systems, 1987, pp. 60–104.

Divergence Control

▶ Replica Freshness

Divergence from Randomness Models

GIAMBATTISTA AMATI
Ugo Bordon Foundation, Rome, Italy

Synonyms

Deviation from randomness

Definition

Divergence From Randomness (DFR) Information Retrieval models are term-document matching functions that are obtained by the product of two divergence functions. An example of DFR function is that related to Jensen's information of two probability distributions [9, pp. 26–28]:

$$\sum_i I_1(\hat{p}_i^+||\hat{p}_i) .\, I_2(\hat{p}_i^+||\hat{p}_i)$$

where $I_1(\hat{p}_i^+||\hat{p}_i) = \hat{p}_i^+ - \hat{p}_i = \Delta\hat{p}_i$ and $I_2(\hat{p}_i^+||\hat{p}_i) = \log_2 \frac{\hat{p}_i+\Delta\hat{p}_i}{\hat{p}_i}$

The DFR generalizes the Jensen's information as follows:

$$\sum_i I_1(\hat{p}_i^+||\hat{p}_i) .\, I_2(\hat{p}_i^+||p_i)$$

where

- p is a prior probability density function of terms (or documents) in the collection.
- $\hat{p}$ is the frequency of the term in a document (or in a subset of documents).
- $\hat{p}^+$ is the neighboring frequency of the term in a document (or in a subset of documents).
- $I_1(\hat{p}^+||\hat{p}_i) = \sum_i I_1(\hat{p}_i^+||\hat{p}_i) = 0$ if and only if $\hat{p}^+ = \hat{p}$.
- $I_2(\hat{p}||p) = \sum_i I_2(\hat{p}||p)$ is minimum when $\hat{p} = p$.

In a DFR model a term occurs randomly when $\hat{p} = p$, whereas a term is informative when $\hat{p} \gg p$.

Historical Background

Divergence From Randomness models were inspired by Harter's 2-Poisson model. Harter's model assumes that a word randomly distributedaccording to a Poisson distribution is not informative, whereas a word that does not follow a Poisson distribution indicates that it conveys information [7]. The Okapi retrieval function, BM25, was also inspired by Harter's 2-Poisson model, and indeed it can be derived from a DFR model [3].

Foundations

The Divergence From Randomness models have their roots in information theory. Following Shannon's theory of information a document can be seen as *a message to transmit*, where information is measured by the cost of transmission. For example, if a message $m(k)$ of length k is generated by a set $\mathbf{V}$ of n symbols $\mathbf{t}_i$, each symbol occurring with a frequency (*prior probability*) p_i, with $0 \leq i \leq n$, then the information is:

$$I_2 = -\log_2 p(m(k)) = -\log_2 \prod_{i=1}^{n} p_i^{k.pi}$$
$$= -k \sum_{i=1}^{n} .\; p_i \log_2 p_i$$

The entropy H of a the system generated by $\mathbf{V}$ is defined as the average information transmitted by its symbols

$$H = \frac{-\log_2 p(m(k))}{k} = -\sum_{0\leq i\leq n} p_i \log_2 p_i$$

The average information transmitted by an arbitrary message $m(k)$ of length k is approximated by

$$I_2(k) = -\log_2 p(m(k)) \sim k.H \tag{1}$$

A collection $\mathbf{D}$ of $\mathbf{N}$ documents can be also conceived as a set of messages that are generated by a vocabulary $\mathbf{V}$ of n words $\mathbf{t}_i$. There are two types of DFR models:

1. *The document model.* Each document $\mathbf{d}$ of length k generates a partition of its terms over n cells (terms). The partition satisfies the constraint $\sum_{i=1}^{n} \mathbf{tf}_i = k$, where $\mathbf{tf}_i$ is the term frequency of the i-th term in the document, and each term $\mathbf{t}_i$ has a prior probability p_i of occurrence in an arbitrary document of the collection.
2. *The term model.* All occurrences of a term $\mathbf{t}$ in a collection $\mathbf{D}$ of $\mathbf{N}$ documents generates a partition over $\mathbf{N}$ cells (documents). Each document has a prior probability p_i of being selected. The partition satisfies the constraint $\sum_{i=1}^{\mathrm{N}} \mathbf{tf}_i = \mathbf{TF}$, where $\mathbf{tf}_i$ is the term frequency in the i-th document and $\mathbf{TF}$ is the term frequency in the collection.

Document Models

Equation (1) provides the mean information carried by an arbitrary document $\mathbf{d}$ of length k with expected term frequencies $k \cdot p_i$. However, if the term frequency X_i in the observed document is $\mathbf{tf}_i$ and not $k \cdot p_i$, then the information of the document is given by the multinomial distribution:

$$p\left(\mathrm{X}_1 = \mathbf{tf}_1, \ldots, X_n = \mathbf{tf}_n\right) = \binom{k}{\mathbf{tf}_1 \mathbf{tf}_2 \ldots \mathrm{tf}_n} p_1^{\mathbf{tf}_1} p_2^{\mathbf{tf}_2} \cdots p_{\mathbf{n}}^{\mathbf{tf}_n}$$

Here, the term independence is assumed, according to which a document is treated as an ensemble and is said to be a *bag of words.* Setting $\hat{p}_i$ to the term frequency in the document $\frac{\mathbf{tf}_i}{k}$, it can be shown that [9, Chap. 1, Ex. 5.12] that:

$$\lim_{k \to \infty} \frac{I_2(\hat{p}||p)}{k} = D(\hat{p}||p) \tag{2}$$

where $I_2(\hat{p}||p) = -\log_2 p(X_1 = \mathbf{tf}_1, \ldots, X_n = \mathbf{tf}_n)$ and $D(\hat{p}||p) \sim \sum_{\mathrm{t}_i \in\, v} \hat{p}_i \log_2 \frac{\hat{p}_i}{p_i}$ is the *Kullback-Leibler divergence.*

The prior probability p_i can be set to $\frac{\mathbf{n}_\mathrm{i}}{\mathbf{N}}$, the ratio of the number $\mathbf{n}_\mathrm{i}$ of documents in which the term occurs and the number $\mathbf{N}$ of documents of the collection (The contribution of a term not occurring in a document is null, being $\mathbf{tf} \cdot \log_2 \mathbf{tf} \to 0$ for $\mathbf{tf} \to 0$.):

$$I_2(\hat{p}||p) \sim k.D(\hat{p}||p) = k.\sum_{\mathbf{t}_i \in \mathbf{v}} \hat{p}_i.\log_2 \frac{\hat{p}_i}{p_i} \sim \sum_{\mathbf{t}_i \in \mathbf{v}} \mathbf{tf}_i \log_2 \frac{\mathbf{tf}_i.\mathbf{N}}{k.\mathbf{n}_\mathrm{i}} \tag{3}$$

The approximation of (3) is additive on terms, and additivity is a useful property to extract the

Divergence from Randomness Models. Table 1. The notations

Notation		Document model	Term model
$\mathbf{tf}$	The term frequency in the document		
$X_i = \mathbf{tf}_i$	The frequency of the i-th term in the observed document		
$Y_j = \mathbf{tf}_j$	The frequency of the observed term in the jTtH document		
X	The vector of the X_i random variables		
Y	The vector of the Y_i random variables		
$\mathbf{TF}$	The term frequency in the collection		
$\mathbf{n_t}$	The document frequency of the term in the collection		
$\mathbf{N}$	The number of documents		
k	The document length		
$\hat{p}$	The estimate of probability density from observations (likelihood)	$\frac{\mathbf{tf}_i}{k}$	$\frac{\mathbf{tf}_i}{\mathbf{TF}}$
$\hat{p}^+$	The neighboring value of the probability density from observations	$\frac{\mathbf{tf}_i+1}{k+1}$	$\frac{\mathbf{tf}_i+1}{\mathbf{TF+1}}$
p	The prior probability density	$\frac{\mathbf{TF}_i}{\sum_i^n \mathbf{TF}_i}$ or $\frac{\mathbf{n_t}}{\mathbf{N}}$	$\frac{1}{\mathbf{N}}$
$D(\hat{p}\|\|p)$	The Kullback-Leibler divergence of $\hat{p}$ from p		

contribution of single terms to information. For example, given the query **q** the contribution of the query terms to the document is:

$$I_2(\hat{p}||p) = -\log_2 p(X_1 = \mathbf{tf}_1, \ldots, X_n = \mathbf{tf}_n|\mathbf{q}) \\ = \sum_{\mathbf{t}_i \in \mathbf{q}} \mathbf{tf}_i \log_2 \frac{\mathbf{tf}_i.\mathbf{N}}{k.\mathbf{n}_\mathrm{i}} \qquad (4)$$

Term Models

DFR term models are developed similarly as the DFR document models. The analogous formula of (3) is

$$I_2(\hat{p}||p) = -\log_2(\mathrm{p}(\mathrm{Y}_1 = \mathbf{tf}_1, \ldots, \mathrm{Y}_\mathrm{n} = \mathbf{tf}_\mathrm{N})) \\ \sim \mathbf{TF} \cdot D(\hat{p}||p) = \mathbf{TF} \cdot \sum_{i=1}^{\mathrm{N}} \hat{p}_i \cdot \log_2 \frac{\hat{p}_i}{p_i} \qquad (5) \\ = \sum_{i=1}^{\mathrm{N}} \mathbf{tf}_i \log_2 \frac{\mathbf{tf}_\mathrm{i} \cdot \mathbf{N}}{\mathbf{TF}}$$

where $\hat{p}_i = \frac{\mathbf{tf}_i}{\mathbf{TF}}$ and $p_i = \frac{1}{\mathbf{N}}$ is the prior probability of a document.

Document Length Normalization

Similarly to TF-IDF used in the vector space model, also Formula 4 can be used for retrieval (see *vector space model*). However, information (see (3)) grows approximately linearly with the document length. If Formula 4 were used as retrieval function, then long documents would be preferred to short ones, while retrieval evaluation has shown that users prefer more distilled information. Formula 4 needs to be normalized with respect to the document length. Length normalization is a critical issue in information retrieval models: for example the vector space model normalizes weights by dividing them by the norms of the query and documents vectors, whilst language models normalizes by learning the value of a parameter that combines the two term frequencies p and $\hat{p}$.

It is indeed possible to average the information-$\log(p(X_1 = \mathbf{tf}_1, \ldots, X_n = \mathbf{tf}_n))$ by the length of the document, obtaining the Kullback-Leibler divergence, but in such a case shorter documents would be preferred to longer ones, because $\hat{p}_i$ is higher in very short documents than in long documents. A way to normalize the information carried by the term is to compute the *information gain*, which corresponds to the increase of information provided by the addition of one or more occurrences of the term, for example when the term frequency increases from $\hat{p}_i = \frac{\mathbf{tf}}{k}$ to its neighboring point $\hat{p}_i^+ = \frac{\mathbf{tf}+\mathbf{1}}{k+1}$. The increment rate of information is

$$\sum_i I_1(\hat{p}_i^+||\hat{p}_i).I_2(\hat{p}_i||p_i) \sim \sum_{\mathbf{t}_i \in \mathbf{q}} \left(1 - \frac{\hat{p}_i}{\hat{p}_i^+}\right) k.\hat{p}_i. \log_2 \frac{\hat{p}_i}{p_i} \qquad (6)$$

where $I_1(\hat{p}_i^+||\hat{p}_i) = 1 - \frac{\hat{p}_i}{\hat{p}_i^+}$.

The matching Formula 6 considers only the contribution of the query terms and exploits the additivity property of information over independent terms.

Examples of Document Models

The following two models are examples of DFR models generated by (6):

- *Example of a DFR document model.* Let $\hat{p}_i = \frac{\mathbf{tf}_i}{k}$, $\hat{p}_i^+ = \frac{\mathbf{tf}_i + 1}{k+1}$. Equation (6) becomes

$$\sum_{\mathbf{t}_i \in \mathbf{q}} \frac{1 - \hat{p}_i}{k \cdot \hat{p}_i + 1} (-\log_2 p(x_1 = \mathbf{tf}_1, \ldots, x_n = \mathbf{tf}_n|\mathbf{q})) \\ \sim k \cdot \sum_{\mathbf{t}_i \in \mathbf{q}} \frac{1 - \hat{p}_i}{k \cdot \hat{p}_i + 1} \cdot \hat{p}_i \cdot \log_2 \frac{\hat{p}_i}{p_i}$$

An approximation of $-\log_2 p(X_1 = \mathbf{tf}_1, \ldots, X_n = \mathbf{tf}_\mathrm{n}|\mathbf{q})$, different from the Kullback-Leibler divergence can be obtained exploiting the Stirling formula that is used for the computation of the factorials [1]. This is a parameter free model of IR and an analogous formula is implemented in the Terrier search engine [10] (The increment rate is computed by the χ-square divergence between the two neighboring probabilities $\hat{p}_i$ and $\hat{p}_i^+$.).

- *Example of a DFR term model.* Let $\hat{p}_i = B(\mathbf{tf}_i, \mathbf{TF}, \frac{\mathbf{n}_\mathrm{t}}{\mathbf{N}})$, $\hat{p}_i^+ = B(\mathbf{tf}_i + 1, \mathbf{TF} + 1, \frac{\mathbf{n}_\mathrm{t}}{\mathbf{N}})$ where

$$B(\mathbf{tf}_i, \mathbf{TF}, p_i) = \binom{\mathbf{TF}}{\mathbf{tf}_i} \left(\frac{\mathbf{n}_\mathrm{t}}{\mathbf{N}}\right)^{\mathbf{tf}_i} \left(1 - \frac{\mathbf{n}_\mathrm{t}}{\mathbf{N}}\right)^{\mathbf{TF} - \mathbf{tf}_i}$$

Different from document model, term models do not include the document length among its observable random variables. Therefore, the term frequency **tf** is not normalized with respect to the length of the document. The following term frequency normalization was shown to be very effective

$$\mathbf{tfn} = \mathbf{tf} \cdot \ln\left(1 + \frac{c \cdot \bar{k}}{k}\right)$$

where $\bar{k}$ is the average document length and c is a parameter [8]. Equation (6) becomes

$$\sum_{\mathbf{t}_i \in q} \frac{\mathbf{TF} + 1}{\mathbf{n}_\mathbf{t} \cdot (\mathbf{tfn}_i + 1)} (-\log_2 p(Y_1 = \mathbf{tf}_1, \ldots, Y_\mathbf{N} = \mathbf{tf}_\mathbf{N}, \\ \mathrm{Z} = k|\mathbf{q})) \sim \sum_{\mathbf{t}_i \in \mathbf{q}} \frac{\mathbf{TF} + 1}{\mathbf{n}_t \cdot (\mathbf{tfn}_i + 1)} \cdot \mathbf{tfn}_i \cdot \log_2 \frac{\mathbf{tfn}_i \cdot \mathbf{N}}{\mathbf{TF}}$$

As for the previous document model, the Stripling formula can be used for the computation of the factorials to obtain a better approximation of $-\log_2 p(Y_1 = \mathbf{tf}_1, \ldots, Y_\mathrm{N} = \mathbf{tf}_\mathrm{N}, Z = k|\mathbf{q})$ [3]. Other DFR models can be built by varying the way both information and the information gain are defined [3].

Key Applications

DFR models can be also applied to query expansion, and to predict query performance [2].

Future Directions

The notion of information gain of the DFR models are strictly connected to the theory of causation and of aftereffect in future sampling (word burstiness) [4,5] [6, pp. 399–402]. The notion of neighboring points used in information gain is related to Fisher's information [9, pp. 26–28]. A deeper analysis of the relation of information gain to these concepts can lead to the discover of more performing models of IR.

URL to Code

DFR models are implemented in the search engine Terrier http://ir.dcs.gla.ac.uk/terrier/.

Cross-references

► 2-Poisson Model
► BM25
► Length Normalization
► Query Expansion Models

Recommended Reading

1. Amati G. Frequentist and Bayesian approach to Information Retrieval. In Proc. 28th European Conf. on IR Research, 2005, pp. 13–24.
2. Amati G., Carpineto C., and Romano G. Query difficulty, robustness, and selective application of query expansion. In Proc. 26th European Conf. on IR Research, 2004, pp. 127–137.
3. Amati G. and Van Rijsbergen C.J. Probabilistic models of information retrieval based on measuring the divergence from randomness. ACM Trans. Inform. Syst., 20(4):357–389, 2002.
4. Gärdenfors P. Knowledge in Flux. MIT, 1988.
5. Gaussier E. and Clinchant S. The BNB distribution for text modeling. In ECIR, Lecture Notes in Computer Science. Springer, 2008.
6. Good I.J. A casual calculus i. Br. J. Phil. Sci., 11:305–318, 1961.
7. Harter S.P. A probabilistic approach to automatic keyword indexing. PhD thesis, Graduate Library, The University of Chicago, Thesis No. T25146, 1974.
8. He I. and Ounis B. On setting the hyper-parameters of the term frequency normalisation for information retrieval. ACM Trans. Inform. Syst., 2007.
9. Kullback S. Information Theory and Statistics. Wiley, New York, 1959.
10. Ounis I., Amati G., Plachouras V., He B., Macdonald C., and Johnson D. Terrier Information Retrieval Platform. In Proc. 27th European Conf. on IR Research, 2005, pp. 517–519.

DNA Sequences

► Biological Sequence

Document

Ethan V. Munson
University of Wisconsin-Milwaukee, Milwaukee, WI, USA

Definition

A document is a representation of information designed for consumption by people. A document may contain information in any medium or in multiple media, though text is generally the dominant medium. A document may be persistent and suitable for archival uses or it may be ephemeral, lasting only for one viewing.

Key Points

Dictionary definitions of the concept of a document generally emphasize documents that have a physical form. Even when computers are considered, the definitions assume that a document is a file on a storage device.

Modern computing, thanks especially to the Web, has greatly expanded the scope of the term. Computers make documents more diverse because computers allow users to create, store and manage material from a variety of media using similar metaphors and interfaces. Computers may also create documents on the fly from fragments stored in a database, as is common in e-commerce and other Web applications. These documents may only exist long enough to be transmitted from a server to a Web browser for a single viewing by a single user.

Cross-references

► Document Representations (Inclusive Native and Relational)

Document Clustering

Ying Zhao[1], George Karypis[2]
[1]Tsinghua University, Beijing, China
[2]University of Minnesota, Minneapolis, MN, USA

Synonyms

Text clustering; High-dimensional clustering; Unsupervised learning on document datasets

Definition

At a high-level the problem of document clustering is defined as follows. Given a set S of n documents, we would like to partition them into a pre-determined number of k subsets S_1, S_2,...,S_k, such that the documents assigned to each subset are more similar to each other than the documents assigned to different subsets. Document clustering is an essential part of text mining and has many applications in information retrieval and knowledge management. Document clustering faces two big challenges: the dimensionality of the feature space tends to be high (i.e., a document collection often consists of thousands or tens of thousands unique words); the size of a document collection tends to be large.

Historical Background

Fast and high-quality document clustering algorithms play an important role in providing intuitive navigation and browsing mechanisms as well as in facilitating knowledge management. The tremendous growth in the volume of text documents available on the Internet, digital libraries, news sources, and company-wide intranets has led an increased interest in developing methods that can help users to effectively navigate, summarize, and organize this information with the ultimate goal of helping them to find what they are looking for. Fast and high-quality document clustering algorithms play an important role towards this goal as they provide both an intuitive navigation/browsing mechanism by organizing large amounts of information into a small number of meaningful clusters as well as to greatly improve the retrieval performance either via cluster-driven dimensionality reduction, term-weighting, or query expansion.

Foundations

Figure 1 shows the commonly used three-step process of transferring a document collection into clustering results that are of value to a user. Original documents are often plain text files, html files, xml files, or a mixture of them. However, most clustering algorithms cannot operate on such textual files directly. Hence, a *document representation* is needed to transform the original documents into the data model on which clustering algorithms can operate. Depending on the characteristics of the document collection and the application requirements, the actual clustering process can be performed using various types of clustering algorithms including partitional clustering, agglomerative clustering, model-based clustering, etc. Finally, the quality of the clustering results need to be properly assessed and presented to the users. Details on some of the most commonly used methods in this three-step process follows.

Document Representation

Documents are often represented using the *term frequency-inverse document frequency* (tf-idf) vector-space model [12]. In this model, each document d is considered to be a vector in the term-space and is represented by the vector

$$d_{tfidf} = (tf_1 \log(n/df_1), tf_2 \log(n/df_2), \ldots, tf_m \log(n/df_m)),$$

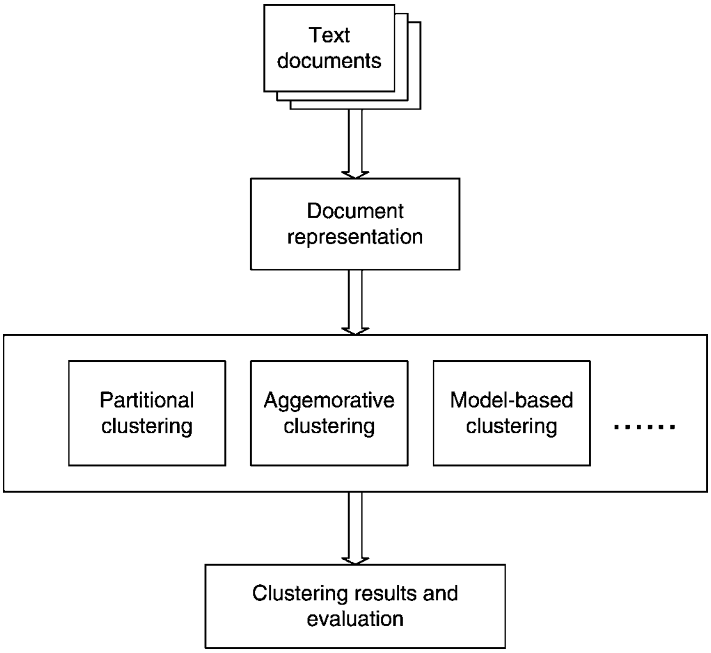
where tf_i is the frequency of the ith term (i.e., term frequency), n is the total number of documents, and df_i is the number of documents that contain the ith term (i.e., document frequency). To account for documents of different lengths, the length of each document vector is normalized so that it is of unit length.

Similarity Measures

Two prominent ways have been proposed to measure the similarity between two documents d_i and d_j when represented via their tf-idf representation. The first method is based on the commonly used [12] cosine function

$$\cos(d_{i,} d_j) = d_i^t d_j / (||d_i||\ ||d_j||),$$

and since the document vectors are of unit length, it simplifies to $d_i{}^t d_j$. The second method computes the similarity between the documents using the Euclidean distance $\mathrm{dis}(d_i, d_j) = ||d_i - d_j||$. Note that besides the fact that one measures similarity and the other measures distance, these measures are quite similar to each other because the document vectors are of unit length.



Document Clustering. Figure 1. Structure of document clustering learning system.

Partitional Document Clustering

Partitional algorithms, such as *K*-means [11], *K*-medoids [8], probabilistic [3], graph-partitioning-based [14], or spectral based [1], find the clusters by partitioning the entire dataset into either a predetermined or an automatically derived number of clusters. A key characteristic of many partitional clustering algorithms is that they use a global criterion function whose optimization drives the entire clustering process. For some of these algorithms the criterion function is implicit (e.g., PDDP [1]), whereas for other algorithms (e.g., *K*-means [11]) the criterion function is explicit and can be easily stated. This latter class of algorithms can be thought of as consisting of two key components. First is the criterion function that the clustering solution optimizes, and second is the actual algorithm that achieves this optimization.

Criterion Function Criterion functions used in the partitional clustering reflect the underlying definition of the "goodness" of clusters. The partitional clustering can be considered as an optimization procedure that tries to create high quality clusters according to a particular criterion function. Many criterion functions have been proposed and analyzed [8,6,16]. Table 1 lists a total of seven different clustering criterion functions. These functions optimize various aspects of intra-cluster similarity, inter-cluster dissimilarity, and their combinations, and represent some of the most widely used criterion functions for document clustering. These criterion functions utilize different views of the underlying collection, by either modeling the objects as vectors in a high-dimensional space, or by modeling the collection as a graph.

The $\mathcal{I}_1$ criterion function (1) maximizes the sum of the average pairwise similarities (as measured by the cosine function) between the documents assigned to each cluster weighted according to the size of each cluster. The $\mathcal{I}_2$ criterion function (2) is used by the popular vector-space variant of the *K*-means algorithm [2]. In this algorithm each cluster is represented by its centroid vector and the goal is to find the solution that maximizes the similarity between each document and the centroid of the cluster that is assigned to. Comparing $\mathcal{I}_1$ and $\mathcal{I}_2$ we see that the essential difference between them is that $\mathcal{I}_2$ scales the within-cluster similarity by the $||D_r||$ term as opposed to the n_r term used by $\mathcal{I}_1$. $||D_r||$ is the square-root of the pairwise similarity between all the document in S_r and will tend to emphasize clusters whose documents have smaller pairwise similarities compared to clusters with higher pairwise similarities.

The $\mathcal{E}_1$ criterion function (3) computes the clustering by finding a solution that separates the documents

Document Clustering. Table 1. The mathematical definition of various clustering criterion functions

Criterion function	Optimization function
$\mathcal{I}_1$	(1) maximize $\sum_{i=1}^{k} \frac{1}{n_i} \left(\sum_{v,u \in S_i} \text{sim}(v, u) \right)$
$\mathcal{I}_2$	(2) maximize $\sum_{i=1}^{k} \sqrt{\sum_{v,u \in S_i} \text{sim}(v, u)}$
$\mathcal{E}_1$	(3) minimize $\sum_{i=1}^{k} n_i \frac{\sum_{v \in S_i, u \in S} \text{sim}(v,u)}{\sqrt{\sum_{v,u \in S_i} \text{sim}(v,u)}}$
$\mathcal{G}_1$	(4) minimize $\sum_{i=1}^{k} \frac{\sum_{v \in S_i, u \in S} \text{sim}(v,u)}{\sum_{v,u \in S_i} \text{sim}(v,u)}$
$\mathcal{G}_2$	(5) minimize $\sum_{r=1}^{k} \frac{\text{cut}(V_r, V - V_r)}{W(V_r)}$
$\mathcal{H}_1$	(6) maximize $\frac{\mathcal{I}_1}{\mathcal{E}_1}$
$\mathcal{H}_2$	(7) maximize $\frac{\mathcal{I}_2}{\mathcal{E}_1}$

The notation in these equations are as follows: k is the total number of clusters, S is the total objects to be clustered, S_i is the set of objects assigned to the ith cluster, n_i is the number of objects in the ith cluster, v and u represent two objects, and sim (v,u) is the similarity between two objects.

of each cluster from the entire collection. Specifically, it tries to minimize the cosine between the centroid vector of each cluster and the centroid vector of the entire collection. The contribution of each cluster is weighted proportionally to its size so that larger clusters will be weighted higher in the overall clustering solution. $\mathcal{E}_1$ was motivated by multiple discriminant analysis and is similar to minimizing the trace of the between-cluster scatter matrix [6].

The $\mathcal{H}_1$ and $\mathcal{H}_2$ criterion functions (6) and (7) are obtained by combining criterion $\mathcal{I}_1$ with $\mathcal{E}_1$, and $\mathcal{I}_2$ with $\mathcal{E}_1$, respectively. Since $\mathcal{E}_1$ is minimized, both $\mathcal{H}_1$ and $\mathcal{H}_2$ need to be maximized as they are inversely related to $\mathcal{E}_1$.

The criterion functions that we described so far, view each document as a multidimensional vector. An alternate way of modeling the relations between documents is to use graphs. Two types of graphs are commonly used in the context of clustering. The first corresponds to the document-to-document similarity graph G_s and the second to the document-to-term bipartite graph G_b [15,4]. G_s is obtained by treating the pairwise similarity matrix of the dataset as the adjacency matrix of G_s, whereas G_b is obtained by viewing the documents and the terms as the two sets of vertices (V_d and V_t) of a bipartite graph. In this bipartite graph, if the ith document contains the jth term, then there is an edge connecting the corresponding ith vertex of V_d to the jth vertex of V_t. The weights of these edges are set using the *tf-idf* model.

Viewing the documents in this fashion, edge-cut-based criterion functions can be used to cluster document datasets. $\mathcal{G}_1$ and $\mathcal{G}_2$ (4) and (5) are two such criterion functions that are defined on the similarity and bipartite graphs, respectively. The $\mathcal{G}_1$ function [5] views the clustering process as that of partitioning the documents into groups that minimize the edge-cut of each partition. However, because this edge-cut-based criterion function may have trivial solutions the edge-cut of each cluster is scaled by the sum of the cluster's internal edges [5]. Note that cut$(S_r, S - S_r)$ in (4) is the edge-cut between the vertices in S_r and the rest of the vertices $S - S_r$, and can be re-written as $D_r^t(D - D_r)$ since the similarity between documents is measured using the cosine function. The $\mathcal{G}_2$ criterion function [15,4] views the clustering problem as a simultaneous partitioning of the documents and the terms so that it minimizes the normalized edge-cut of the partitioning. Note that V_r is the set of vertices assigned to the rth cluster and $W(V_r)$ is the sum of the weights of the adjacency lists of the vertices assigned to the rth cluster.

Optimization Method There are many techniques that can be used to optimize the criterion functions described above. They include relatively simple greedy schemes, iterative schemes with varying degree of hill-climbing capabilities, and powerful but computationally expensive spectral-based optimizers [11,1,15,4,7]. Here is a simple yet very powerful greedy strategy that has been shown to produce comparable results to those produced by more sophisticated optimization algorithms. In this greedy straggly, a k-way clustering of a set of documents can be computed either directly or via a sequence of repeated bisections. A direct k-way clustering is computed as follows. Initially, a set of k objects is selected from the datasets to act as the *seeds* of the k clusters. Then, for each object, its similarity to these k seeds is computed, and it is assigned to the cluster corresponding to its most similar seed. This forms the initial k-way clustering. This clustering is then repeatedly refined so that it optimizes a desired clustering criterion function. A k-way partitioning via repeated bisections is obtained by recursively applying the above algorithm to compute two-way clustering (i.e., bisections). Initially, the objects are partitioned into two clusters, then one of these clusters is selected and is further bisected, and so on. This process continues $k - 1$ times, leading to k clusters. Each of these bisections is performed so that the resulting two-way clustering solution optimizes a particular criterion function.

Agglomerative Document Clustering

Hierarchical agglomerative algorithms find the clusters by initially assigning each object to its own cluster and then repeatedly merging pairs of clusters until a certain stopping criterion is met. Consider an n-object dataset and the clustering solution that has been computed after performing l merging steps. This solution will contain exactly $n - l$ clusters, as each merging step reduces the number of clusters by one. Now, given this $(n - l)$-way clustering solution, the pair of clusters that is selected to be merged next, is the one that leads to an $(n - l - 1)$-way solution that optimizes a particular criterion function. That is, each one of the $(n - l) \times (n - l - 1)/$two pairs of possible merges is evaluated, and the one that leads to a clustering solution that has the maximum (or minimum) value of the particular criterion function is selected. Thus, the criterion function is *locally* optimized within each particular stage of agglomerative algorithms. Depending on the desired solution, this process continues until either there are only k clusters left, or when the entire agglomerative tree has been obtained.

The three basic criteria to determine which pair of clusters to be merged next are single-link [13], complete-link [10] and group average (i.e., unweighted Pair Group Method with Arithmetic mean (UPGMA)) [8]. The single-link criterion function measures the similarity of two clusters by the maximum similarity between any pair of objects from each cluster, whereas the complete-link uses the minimum similarity. In general, both the single- and the complete-link approaches do not work very well because they either base their decisions to a limited amount of information (single-link), or assume that all the objects in the cluster are very similar to each other (complete-link). On the other hand, the group average approach measures the similarity of two clusters by the average of the pairwise similarity of the objects from each cluster and does not suffer from the problems arising with single- and complete-link.

Evaluation of Document Clustering

Clustering results are difficult to evaluate, especially for high dimensional data and without a priori knowledge of the objects' distribution, which is quite common in practical cases. However, assessing the quality of the resulting clusters is as important as generating the clusters. Given the same dataset, different clustering algorithms with various parameters or initial conditions will give very different clusters. It is essential to know whether the resulting clusters are valid and how to compare the quality of the clustering results, so that the right clustering algorithm can be chosen and the best clustering results can be used for further analysis.

In general, there are two types of metrics for assessing clustering results: metrics that only utilize the information provided to the clustering algorithms (i.e., *internal metrics*) and metrics that utilize a priori knowledge of the classification information of the dataset (i.e., *external metrics*).

The basic idea behind internal quality measures is rooted from the definition of clusters. A meaningful clustering solution should group objects into various clusters, so that the objects within each cluster are more similar to each other than the objects from different clusters. Therefore, most of the internal quality measures evaluate the clustering solution by looking at how similar the objects are within each cluster and how well the objects of different clusters are separated. In particular, the *internal similarity* measure, *ISim*, is defined as the average similarity between the objects of each cluster, and the *external similarity* measure, *ESim*, is defined as the average similarity of the objects of each cluster and the rest of the objects in the data set. The ratio between the internal and external similarity measure is also a good indicator of the quality of the resultant clusters. The higher the ratio values, the better the clustering solution is. One of the limitations of the internal quality measures is that they often use the same information both in discovering and in evaluating the clusters.

The approaches based on external quality measures require a priori knowledge of the natural clusters that exist in the dataset, and validate a clustering result by measuring the agreement between the discovered clusters and the known information. For instance, when clustering document datasets, the known categorization of the documents can be treated as the natural clusters, and the resulting clustering solution will be considered correct, if it leads to clusters that preserve this categorization. A key aspect of the external quality measures is that they utilize information other than that used by the clustering algorithms. The *entropy* measure is one such metric that looks are how the various classes of documents are distributed within each cluster.

Given a particular cluster, S_r, of size n_r, the entropy of this cluster is defined to be

$$E(S_r) = -\frac{1}{\log q}\sum_{i=1}^{q}\frac{n_r^i}{n_r}\log\frac{n_r^i}{n_r} \qquad (8)$$

where q is the number of classes in the data set, and $n_r{}^i$ is the number of documents of the ith class that were assigned to the rth cluster. The entropy of the entire clustering solution is then defined to be the sum of the individual cluster entropy weighted according to the cluster size. That is,

$$Entropy = \sum_{r=1}^{k}\frac{n_r}{n}E(S_r). \qquad (9)$$

A perfect clustering solution will be the one that leads to clusters that contain documents from only a single class, in which case the entropy will be zero. In general, the smaller the entropy values, the better the clustering solution is.

Key Applications

Document clustering is used to organize large collections of documents into meaningful groups in order to provide intuitive navigation aids, information summarization, data compression, and dimensionality reduction.

URL to Code

An illustrative example of a software package for clustering low- and high-dimensional datasets and for analyzing the characteristics of the various clusters is Cluto [9]. Cluto has implementations of the various clustering algorithms and evaluation metrics described in previous sections. It was designed by the University of Minnesota's data mining's group and is available at http://www.cs.umn.edu/~karypis/cluto.

Data Sets

Utility tools for pre-processing documents into vector matrices and some sample document datasets are also available at http://www.cs.umn.edu/~karypis/cluto.

Cross-references

▶ Clustering Assessment
▶ Clustering for post-hoc information retrieval
▶ Information Retrieval
▶ Text Mining
▶ Unsupervised Learning

Recommended Reading

1. Boley D. Principal direction divisive partitioning. Data Mining Knowl. Discov., 2(4), 1998.
2. Cutting D.R., Pedersen J.O., Karger D.R., and Tukey J.W. Scatter/gather: A cluster-based approach to browsing large document collections. In Proc. 15th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1992, pp. 318–329.
3. Dempster A.P., Laird N.M., and Rubin D.B. Maximum likelihood from incomplete data via the em algorithm. J. R. Stat. Soc., 39, 1977.
4. Dhillon I.S. Co-clustering documents and words using bipartite spectral graph partitioning. In Knowledge Discovery and Data Mining, 2001, pp. 269–274.
5. Ding C., He X., Zha H., Gu M., and Simon H. 1Spectral min-max cut for graph partitioning and data clustering. Technical Report TR-2001-XX, Lawrence Berkeley National Laboratory, University of California, Berkeley, CA, 2001.
6. Duda R.O., Hart P.E., and Stork D.G. Pattern Classification. Wiley, New York, 2001.
7. Fisher D. Iterative optimization and simplification of hierarchical clusterings. J. Artif. Intell. Res., 4:147–180, 1996.
8. Jain A.K. and Dubes R.C. Algorithms for Clustering Data. Prentice Hall, New York, 1988.
9. Karypis G. Cluto: A clustering toolkit. Technical Report 02-017, Department of Computer Science, University of Minnesota, 2002.
10. King B. Step-wise clustering procedures. J. Am. Stat. Assoc., 69:86–101, 1967.
11. MacQueen J. Some methods for classification and analysis of multivariate observations. In Proc. 5th Symp. Math. Stat. Prob., 1967, pp. 281–297.
12. Salton G. Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley, Reading, MA, 1989.
13. Sneath P.H. and Sokal R.R. Numerical Taxonomy. Freeman, London, UK, 1973.
14. Zahn K. Graph-tehoretical methods for detecting and describing gestalt clusters. IEEE Trans. Comput., (C-20):68–86, 1971.
15. Zha H., He X., Ding C., Simon H., and Gu M. Bipartite graph partitioning and data clustering. In Proc. Int. Conf. on Information and Knowledge Management, 2001.
16. Zhao Y. and Karypis G. Criterion functions for document clustering: Experiments and analysis. Mach. Learn. 55:311–331, 2004.

Document Databases

Frank Wm. Tompa
University of Waterloo, Waterloo, ON, Canada

Synonyms

Document repositories; Text databases; Corpora

Definition

A document database is a collection of stored texts managed by a system that provides query and update facilities. Usually the database includes many documents related by their subject matter, origin, or applicability to an enterprise. The content of each document may be free text, semi-structured text including a few well-identified fields (e.g., title, author, date), or highly structured tagged text such as might be encoded using XML. Occasionally documents may also contain multimedia components.

In contrast, the term *corpus* (plural *corpora*) typically refers to a static collection of texts that have been assembled by experts to study linguistic phenomena (e.g., the Brown Corpus, created in 1964 to study American English, and the Swedish Language Bank) or to provide a rich source of text for lexicographic needs (e.g., the Dictionary of Old English Corpus, including all extant texts written in Old English in the period 600–1150 AD, and the British National Corpus). Such corpora are often distributed or licensed in the form of data only, independently of any document management system.

Historical Background

Electronic documents have been stored on computers almost as long as numeric data. Early document systems supported text editing and formatting and evolved into sophisticated document creation and publishing systems. Electronic document management became an integral part of the move towards office automation that grew substantially during the 1970s and 1980s. Holding documents in computers also made possible the growth of *hypertexts* and hypermedia more generally, starting in the 1960s and continuing today. This, in turn, formed the core of the World Wide Web.

Simultaneously the field of information retrieval developed in response to the recognition that libraries hold a substantial volume of data that is often difficult to access effectively without the intervention of professional librarians. Initially, small document collections were amassed to test the performance of various algorithms designed to locate relevant sources of data in response to users' information needs. Information retrieval has since advanced substantially to deal efficiently and effectively with multi-gigabyte collections of texts, whether the objective is to find relevant documents or to find answers to very specific factual questions.

An outgrowth of office automation was the recognition that corporate documents form a business resource that deserves management commensurate with the effort put into managing capital, human resources, and more traditionally recognized forms of data. Thus document management may be viewed as an extension of database management to handle documents and document fragments with the same care as is given to tabular and other forms of business data.

Document management systems have evolved from each of three technologies: information retrieval engines, as special applications of object management systems, and as extensions to relational database management systems. They provide facilities to define sub-collections, to load new documents and delete old ones, to update existing documents, to retrieve documents that match precise criteria exactly, and to rank documents against a set of keywords or against criteria that specify users' needs in a less precise manner. Document databases form the core of *Enterprise Content Management* systems.

Foundations

Because document database systems arise from traditional database systems and traditional information retrieval systems, the scientific fundamentals of those technologies underlie document databases as well. When the databases include semi-structured or structured documents, they often include constraints in the form of regular expressions or context-free grammars; thus the principles and practices of regular and context-free languages also underlie document databases.

Key Applications

Document databases are typically created by corporations and other enterprises to subject various documents to database management protocols. Publishers and other organizations (or organizational sub-units) for which printed or electronic documents are their products use document databases to maintain drafts and other variants of their products as well as historical materials. As consumers of such products, organizations' digital libraries use document databases to hold and manage their collections. Thus document databases form a core technology for publishers, digital libraries, e-government, and e-business more generally.

In addition, some document databases maintain materials that are internal to the enterprise. These might comprise policies, procedures, advertising, blogs, email messages, customers' comments, confidential reports, etc. The content for other document databases might be collected from external sources and may include annual reports, legal documents, suppliers' product descriptions, financial reviews, etc. Document databases may also be created to support benchmarking studies or to meet specific application needs, such as source code repositories.

Cross-references

▶ Digital Libraries
▶ Information Retrieval
▶ Semi-structured Data Model
▶ Semi-Structured Database Design
▶ XML Retrieval
▶ XML Storage

Recommended Reading

1. Bertino E., Ooi B., Sacks-Davis R., Tan K.-L., and Zobel J. Text databases. In Indexing Techniques for Advanced Database Systems. Kluwer Academic, Norwell, MA, 1997, pp. 151–184.
2. Chin A.G. (ed.). Text Databases and Document Management: Theory and Practice. Idea Group, Hershey, PA, 2001.
3. Christophides V., Abiteboul S., Cluet S., and Scholl M. From structured documents to novel query facilities. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 313–324.
4. Kilpeläinen P., Lindén G., Mannila H., and Nikunen E. 1A structured text database system. In Proc. Int. Conf. on Electronic Publishing, Document Manipulation and Typography, 1990, pp. 139–151.
5. Loeffen A. Text databases: a survey of text models and systems. ACM SIGMOD Rec., 23(1):97–106, 1994.
6. Lowe B., Zobel J., and Sacks-Davis R. A formal model for databases of structured text, In Proc. 4th Int. Conf. on Database Systems for Advanced Applications, 1995, pp. 449–456.
7. Macleod A. A data base management system for document retrieval applications. Inf. Syst., 6(2):131–137, 1981.
8. Sacks-Davis R., Arnold-Moore T., and Zobel J. Database systems for structured documents. In Proc. Int. Symp. on Advanced Database Technologies and Their Integration, 1994, pp. 272–283.
9. Salminen A. and Tompa F.W. Requirements for XML document database systems. In Proc. ACM Symp. on Document Engineering, 2001, pp. 85–94.
10. Stonebraker M., Stettner H., Lynn N., Kalash J., and Guttman A. Document processing in a relational database system. ACM Trans. Inf. Syst., 1(2):143–158, 1983.

Document Field

Vassilis Plachouras
Yahoo! Research, Barcelona, Spain

Definition

A document field is a part of a document or of the document metadata in which the text has a particular function. A document field can contain free or preformatted text. Each field, according to its function, has different characteristics, length, and term distributions.

Key Points

Textual documents have implicit structure, which aids the understanding of the text. Long textual documents are usually organized in chapters, sections, paragraphs, and each of those can have a concise description in the form of a title. In the case of hypertext documents, explicit links between documents in the form of hyperlinks are often associated with anchor text. News wire documents also have metadata such as date, or the name of the author. Efforts to standardize metadata about documents have resulted in projects such as the Dublin Core Metadata Initiative [1].

Fields are also being used to represent the annotations of text with semantic and syntactic information. For example, the semantic information may correspond to entities, or locations. The syntactic information may correspond to the part of speech of tokens or to syntactic relationships between tokens. Such information can be used to perform search tasks such as entity ranking [3].

The text and the distribution of terms in a particular field depend on the function of that field. For example, a term may occur many times in a document, because of the document's verbosity. On the other hand, the title of a document is a short and concise description of the document. Hence, terms are expected to appear only once or twice in the title of a document, and the resulting term frequency distribution is almost uniform [2]. Similarly, the anchor text of incoming hyperlinks of Web documents serves the purpose of providing a concise description of a document, and from this point of view, it is similar to the title of documents. However, a document is likely to have one title, while it is not unusual to have documents with several million incoming hyperlinks and associated anchor texts.

Cross-references

- ▶ Anchor Text
- ▶ Dublin Core
- ▶ Field-based Information Retrieval Models

Recommended Reading

1. Dublin Core Metadata Initiative. Retrieved April 15, 2008, http://dublincore.org/.
2. Jin R., Hauptmann A., and Zhai C. Title language model for information retrieval. In Proc. 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2002, pp. 42–48.
3. Zaragoza H., Rode H., Mika P., Atserias J., Ciaramita M., and Attardi G. Ranking Very Many Typed Entities on Wikipedia. In Proc. Int. Conf. on Information and Knowledge Management, 2007, pp. 1015–1018.

Document Formats

▶ Document Representations

Document Identifier

▶ Resource Identifier

Document Index and Retrieval

▶ Text Indexing and Retrieval

Document Length Normalization

Ben He
University of Glasgow, Glasgow, UK

Synonyms

Term frequency normalization; Length normalization

Definition

Document length normalization adjusts the term frequency or the relevance score in order to normalize the effect of document length on the document ranking.

Key Points

The reasons for employing a document length normalization method in an IR system are quite subtle. In general, the effect observed on the ranking by the presence of many lengthy documents in a collection is to favor their retrieval with respect to shorter documents.

Singhal, Buckley and Mitra gave the following two reasons for adopting a length normalization in the vector space model [4]:

1. The same term usually occurs repeatedly in long documents.
2. The vocabulary of a long document is usually large.

In 1994, Robertson and Walker also studied the effect of document length in the context of the probabilistic model. They observed that:

- ▶ Some documents may simply cover more material than others, [. . .], a long document covers a similar scope to a short document, but simply uses more words.

According to Robertson and Walker [2], term frequencies may also depend on author's writing style, that may describe concepts and facts either in details or concisely. Robertson and Walker called this phenomenon as the *verbosity hypothesis*.

According to the language modeling approach, the normalization of the document length is instead related to the *sparse data problem*. The sparse data problem is also the core problem in natural language processing for the estimation of the probability of string occurrences. The *smoothing technique* is usually applied to cope with the sparse data problem in the language modeling approach for IR [5].

In the context of Vector Space model, cosine normalization adjusts the effect of document length on document weights by computing the cosine similarity between the query and the document weight vectors.

Singhal et al. proposed an improvement of cosine normalization for the vector space model, called the *pivoted normalization* [4]. The basic idea of the pivoted normalization is to introduce a tunable hyper-parameter to empirically adjust the normalization factor of a given normalization method, by fitting the probability of retrieval to the probability of relevance. The probability of retrieval is computed from returned documents for each given query, and the probability of relevance is computed from the relevance information given by the human assessors.

In the context of probabilistic model, the BM25 weighting model employs a saturation function to normalize term frequency [3]. This normalization function is derived from the study of the document length effect in the 2-Poisson model.

Some of Divergence from Randomness (DFR) weighting models employ the *Normalization 2* for adjusting the relationship between term frequency and document length, that assumes a decreasing term frequency density function of document length [1].

Cross-references

▶ Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model
▶ Probability Smoothing

Recommended Reading

1. Amati G. Probabilistic models for information retrieval based on divergence from randomness. Ph.D. Thesis, Department of Computing Science, University of Glasgow, 2003.
2. Robertson S. E. and Walker S. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1994, pp. 232–241.
3. Robertson S.E., Walker S., Jones S., and Hancock-Beaulieu M. Okapi at trec-3. In Proc. The 3rd Text Retrieval Conference, 1994.
4. Singhal A., Buckley C., and Mitra M. Pivoted document length normalization. In Proc. 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1996, pp. 21–29.
5. Zhai C. and Lafferty J. A study of smoothing methods for language models applied to ad hoc information retrieval. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 334–342.

Document Links and Hyperlinks

Vassilis Plachouras
Yahoo! Research, Barcelona, Spain

Definition

Document links and hyperlinks are cross-references between different documents or between different parts of the same document. They facilitate the navigation of users in the document space. However, information seeking by only following hyperlinks is possible only for relatively small collections of hyperlinked documents. Hyperlinks can have explicit or implicit types. Two common types of links are organizational or navigational links and informational links.

Key Points

Textual documents are rich in structure, one aspect of which is the cross-references or links to different parts of the same document or to other documents. Bibliographic references is one form of links between documents for example. Bush [3] envisioned Hypertext as a natural way to organize, store and search for information, similar to the associative way in which the humans organize information.

Document links and hyperlinks are explicit cross-references between parts of the same document or different documents. Such links alter the information search process by allowing a user to navigate the document space by following hyperlinks. Navigation may be sufficient for small collections of documents. As the number of documents increases, or when navigation is allowed across heterogeneous sets of hypertext documents, however, users may not be able to locate information by merely following links. IR techniques address one aspect of this problem by allowing search for information, or locating starting points for browsing hypertext collection.

The links in hypertext systems can have explicit or implicit types. Baron et al. [2] identified two main types of links, namely the organizational and the content-based links. The former type of links was used to organize and help navigation among hypertext documents, while the latter type was used for pointing to documents on similar topics. However, as with bibliographic references in scientific publications, some hypertext systems do not provide typed links. The automatic inference of the link type is a difficult task, because it requires understanding the context of both the source and destination documents. Differently from identifying the type of hyperlinks [1], investigated the automatic typed linking of related documents. After linking all pairs of documents, the similarity of which exceeds a threshold, the resulting graph is simplified by iteratively merging links. A type is assigned to the resulting links, according to a predefined taxonomy.

Cross-references

▶ Anchor text

Recommended Reading

1. Allan J. Automatic hypertext link typing. In Proc. Seventh ACM Conf. on Hypertext, 1996, pp. 42–52.
2. Baron L., Tague-Sutcliffe J., Kinnucan M.T., and Carey T. Labeled, typed links as cues when reading hypertext documents. J. Am. Soc. Inform. Sci., 47(12):896–908, 1996.
3. Bush V. As we may think. The Atlantic Monthly, July, 1945.

Document Management

► Enterprise Content Management

Document Path Query

► Path Query

Document Repositories

► Document Databases

Document Representations (Inclusive Native and Relational)

ETHAN V. MUNSON
University of Wisconsin-Milwaukee, Milwaukee, WI, USA

Synonyms

Documents; Markup languages; Semi-structured data; Page representations

Definition

Native document representations are file formats designed for *documents*. They can be roughly divided into three types: page-oriented, stream-oriented, and tree-structured. Hybrid types can also be found. Within each type, document representations range from the simple to the complex. All native representations assume an implicit order of the document's information, reflecting the linear reading order of conventional documents. The most important document representation is the Extensible Markup Language (XML), which is tree-structured and can have any level of complexity. It is seeing widespread use on the Web and in business and is also popular for non-document applications.

Relational databases use a variety of document representations that map to a native representation. Page-oriented and stream-oriented documents are best stored in a coarse-grained manner and do not appear to have stimulated much research. In contrast, tree-structured documents are well-suited to fine-grained decomposition for storage in relational databases. As a result, XML databases are a very active research topic. The challenge for relational systems is to maintain the implicit order of the documents' elements while providing efficient access and updates.

Historical Background

Furuta et al. [6] survey document formatting systems up to 1982. The earliest document representations appear to have been created by programmers who wanted to be able to create their own documents without the aid of support staff, using readily available devices. All of the representations described in the survey are *markup languages*. The earliest markup languages, such as RUNOFF and PUB, were stream-oriented. Their markup was highly procedural, specifying changes to parameters of a simple formatter or line breaker. Later markup languages, such as Scribe and GML, supported higher levels of abstraction,were at least partially tree-structured, and were used in systems with higher-quality formatters. For the TeX system, Knuth developed advanced formatting algorithms [9] whose quality has yet to be surpassed. All markup language systems assumed that their users would edit language files with a text editor and then invoke a formatter on the command line to produce output for a printer.

The personal computer revolution in the 1980s spawned the creation of various word processing systems. These systems had user interfaces that were more accessible to non-technical workers, but their document representations were much simpler than those of the later markup-based systems. Early word processors used stream-based representations that were entirely procedural, with no facility for abstract concepts like figures or section headings. As these systems matured, they gained more abstract structural features, such as named styles for paragraphs, but their representations have remained essentially stream-oriented. In general, word processing document representations are not human-readable and are proprietary, though conversion tools between representations are widely available.

The simultaneous development of the laser printer required a means to transmit a page from computer to printer over the low-bandwidth connections then available. In response, various companies designed proprietary page description languages (PDLs) that described pages at a higher level, thus requiring substantially less bandwidth. The most important were Adobe's PostScript, used in the first personal laser printers, and Hewlett-Packard's Printer Command Language (PCL). Both are still widely used in printers, but today the most important PDL is Adobe's Portable Document Format (PDF) [1] because it is printer-independent, compact, and because Adobe distributes free viewing and printing software for all widely-used platforms.

By the mid-1980's, the diversity of incompatible markup languages and word-processing representations was making collaboration between authors quite difficult. In response, two competing document interchange formats were developed, the Standardized Generalized Markup Language (SGML) [7] and the Open Document Architecture (ODA). Only SGML was a success and its success was limited. However, SGML was the basis for the Hypertext Markup Language (HTML) used on the World Wide Web. As HTML came to be used more as a page description language than as a high-level tree-structured specification, the Web community sought a more structured solution. The result was the Extensible Markup Language (XML) [3], which is designed to allow Web documents to convey stronger semantics and to better support sophisticated, even intelligent, applications.

Foundations

Native Representations

Page-Oriented Representations There are two principal page-oriented representations: page images and page description languages (PDLs).

The simplest page-oriented document representation is a sequence of page images, usually created by scanning paper documents. While this representation may seem primitive, it is quite important because of the substantial number of documents that predate electronic representations of any kind or for which the electronic version has been lost. Often, in digital libraries, the page images will have been processed by a document analysis system in order to generate a searchable text stream or to produce an electronic version of the page that can be scaled or reformatted without producing image artifacts. The result is a hybrid representation mixing pages with a stream or tree structure. The development of efficient workflows for this analysis process has been an interesting area of research [13].

PDLs are considerably more complex. The core of any PDL is a two-dimensional vector graphics language with strong support for high-quality text rendering. This implies full support for scientific floating point computation, for conversion between various units of measure, and for specifying character fonts. PDLs must also have commands to control paper handling and common printing features like screening and halftoning. The PDLs used in printers (principally PostScript and PCL) are not suited to database applications because their documents are specific to particular printers and cannot be guaranteed to print or display correctly on all devices. In contrast, the PDF [1] representation is a generalization of PostScript that is device-independent and has evolved over time to have many of the best qualities of stream-oriented and tree-structured representations. Documents encoded by modern PDF generators typically include a complete text stream that can be indexed and searched. Both commercial and open-source tools can be found to generate and manipulate PDF. Finally, it worth mentioning that the PostScript PDL is a fully human-readable language that can be created in a standard text editor, though it also supports binary data formats.

Stream-Oriented Representations Stream-oriented representations organize documents as a sequence of characters or paragraphs. They may contain substantial amounts of formatting information, but unlike the page-oriented representations, generally do not encode the exact appearance of the document on the page or screen. The principal stream-oriented representations are raw text, the Rich Text Format, and various word processor formats.

A raw text document contains a sequence of characters. Any organization of the characters into lines, paragraphs, or pages is specified by the use of specialized characters such as the ASCII line feed and form feed characters. The most common character coding scheme is ASCII, but the more general *Unicode* format is also seen and may grow in importance over time. Raw text has the advantages of simplicity, compactness, portability, and ease of processing. Its primary disadvantage is the inability to represent almost any useful typographic, hypertext, or multimedia effect. The raw text representation is remarkably robust and remains in widespread use, especially in the software development community, where the ubiquity of programming tools makes raw text an attractive representation. It is also a common representation for e-mail.

Rich Text Format (RTF) [10] is a proprietary representation that is widely used for interchange among word processors. Its canonical form is a human-readable ASCII markup language that describes a document as a stream of paragraphs that may be divided into sections. RTF's sections and paragraphs embody regions of content with common formatting characteristics. Document content appears inside the paragraphs along with other markup.

Word processor representations resemble RTF in that they describe a sequence of paragraphs but until recently most have been proprietary, binary representations. Recently, human-readable non-proprietary formats for word processing have begun to be accepted, with the most important being the Open Document Format [11]. This format uses the tree-structured XML markup language, but its underlying structure is still a stream of paragraphs.

Tree-Structured Representations For databases, the most interesting native document representations are tree-structured markup languages. The most important such language is the Extensible Markup Language (XML) [3], which is essentially a simplification of the earlier SGML standard. Because XML is simple, general, and human-readable, it has become a standard representation for data interchange.

XML is really two languages: a markup syntax for documents and a context-free grammar meta-language for defining classes of documents that can be encoded in the markup syntax. The markup syntax primarily defines how a tree of elements with embedded content is specified by marking up the content with *tags*. The following example shows a trivial, but complete, "bookdata" document. The bookdata element is the root of the tree and contains title and editor elements. The bookdata element also has two attributes, which record the topic and year of the book. In general, elements are designed to hold content that will be shown to people and attributes are designed to hold metadata that could be processed by automated tools.

```
<? xml version="1.0" ?>
<bookdata topic="Databases" year ="2008">
   <title>Encyclopedia    of    Database
   Systems</title>
   <editor>Ling Liu</editor>
   <editor>Tamer \:{O}zsu</editor>
</bookdata>
```

XML has several important technical and philosophical differences from the page- and stream-oriented representations.

- Unlike the PDLs, XML is almost purely declarative. It is not a programming language and has no computational features. An XML document describes only a hierarchical organization of content, possibly with metadata.
- XML is designed to represent the logical organization of a document rather than its appearance. It has no predefined formatting features and does not make any assumptions about media or devices.
- While designed for representing documents, XML is not limited to this application. In fact, XML's simplicity and clean syntax have resulted in many unanticipated uses.
- XML is supported by a rich ecosystem of related languages that support tasks including document transformation (XSLT [8]) and alternative grammar systems (or *schemas*) for defining document classes (XSchema [5]). Especially important for databases is the XQuery document query language [2].

XML documents are often categorized into three classes: structured, semi-structured, and marked-up text. In a structured XML document class, all documents have the same tree structure and every element has a unique name. In semi-structured document classes, there may be variations in the tree structure at certain locations, such as alternate element types or variable repetition of one element or a group of elements. In both semi-structured and structured documents, document content is only found in the leaf elements of the tree. In contrast, marked-up text can have content at any level of the tree and may permit huge variations in tree structure. Marked-up text may have important elements of logical structure, such as sentences, that are not explicitly marked-up by elements and span multiple elements. Most database research has focused on structured and semi-structured XML.

Hybrid Representations Hybrid representations can deliver the advantages of multiple representations at the cost of increased complexity. They are most commonly seen as extensions that address the limitations of page-oriented representations.

The combination of page images with a parallel text stream has already been mentioned. This representation can be used to create document interfaces that show the scanned image, but allow indexing and searching of the content, including highlighting those portions of the original page image that match a search string.

Considerably more elaborate is Tagged PDF [1], which extends the page description core of PDF with a structural tagging system to encode the roles of text fragments (e.g., body text, footnote, etc.), adds explicit word breaks, and maps all fonts to Unicode. Used properly, Tagged PDF ensures that the content of a PDF document can be scanned in the same order that a human reader would scan it and clearly identifies elements like marginal notes and headers that are not part of the main text flow. It also supports search and indexing, as well as being able to encode some of the semantics of XML.

Relational Representations

In relational databases, documents can be represented either as atomic entities, using large objects (LOB), or decomposed into their component parts. The large object approach can be used with all native representations. Decomposition is usually called "shredding" and is only used with XML documents.

Large Object Representation LOB representation stores an entire document or medium-sized parts of an entire document as a large object in a relational table. This is the natural representation for documents whose native representation is page-oriented or stream-oriented and has some real advantages for XML documents as well. Long documents may be divided into a sequence of smaller LOBs, such as individual pages or sections.

LOB representation is useful for documents that do not need to be updated frequently and for which interesting metadata can be computed at the time of insertion into the database. In this case, the relational system provides an efficient way to find documents based on queries against the metadata. For page- and stream-oriented documents, LOB representation is a natural choice, because the internal structure of the documents (i.e., pages or sections/paragraphs) principally conveys presentation and has little semantics useful for queries and updates. In contrast, LOB representation is unlikely to be used for XML documents unless they are quite unstructured or if a description of the document class is not available.

LOB representation has the disadvantage that standard relational operations cannot be used to search or update the internal structure and content of the documents. Instead, access and update operations must be performed by other tools. While these tools may be useful and efficient for single documents, the performance and scalability benefits of the relational approach for large-scale collections are lost when using the LOB representation.

Shredded Representation *Shredding* is the process of tearing apart an XML document into its component elements for storage in database relations. There are many trade-offs in designing both relations and queries for the shredded elements. Draper [12] discusses the full range of choices. A key issue is whether the schema for the XML document class is known.

When a schema is not available for an XML document class, the *edge table* representation is used. An edge table has one tuple for each element or attribute in a document. The tuple has the following form:

`Edge(`*ID*`, parentID, name, value)`

The root element has a null parent ID and internal nodes of the tree have null values. A useful optimization is to replace the name with a `pathID` that points to another table holding the full path names of the nodes. Using pathIDs can reduce both table size and the number of joins required for common queries.

When a schema is available for the documents, *inlining* is a more efficient representation. Under inlining, elements are only placed in separate relations when they can appear multiple times. Elements that only appear once become columns in the relation for their parents. In the earlier "bookdata" example, there would be two relations: one for the bookdata element that would have columns for the two attributes and for the title; and another to hold the list of authors that would be connected to the bookdata element via a foreign key. The design of efficient queries over inlined databases is challenging. Shanmugasundaram et al.

[12] showed that a complex query structure called Sorted Outer Union provides the best combination of efficiency and generality.

A key problem when working with shredded XML documents is correctly maintaining the order of the elements. This problem arises because the order of the content in documents is usually quite important, but it is only encoded implicitly. In the earlier "bookdata" example, the order of the author's names should be preserved, but it is only apparent from the order in which the names appear in the XML source code. Relational databases do not represent order automatically, so additional information must be added to the tables. Tatarinov et al. [14] showed that the best choice of order information depends on the type of query load. When updates are rare, it is best to store a global order number (an integer representing the node's position in a preorder tree traversal). For loads that mix updates and accesses, a variable-length numbering system related to the Dewey Decimal Classification system is superior.

Key Applications

Documents are pervasive in human society, so there are many applications for document representations. The most important application is the Web, which can be viewed narrowly as a document-sharing system. Every Web page is a document written in HTML or XHTML (an adaptation of HTML to the rules of XML). A growing number of Web documents are derived from information represented in XML or from XML fragments taken from a database. Because Web browsers have only limited support for XML itself, it is primarily used as a back-end representation.

Other important applications include:

- Scanned document images are widely used to represent for historical, legal, and financial documents. Systems that support scholars typically have rich metadata attached to the page images.
- Page description languages (especially PDF) are widely used as electronic representations of the final form of documents, especially business and official documents that are also distributed in print form.
- The pervasive use of word-processing software makes stream-based representations ubiquitous for business documents. The lack of widely-adopted open standards presents a real challenge for systems that try to support them.

Cross-references

- ▶ Dewey Decimal System
- ▶ Digital Libraries
- ▶ Document
- ▶ Document Databases
- ▶ Indexing Semi-Structured Data
- ▶ Markup Language
- ▶ Meta Data
- ▶ Semantic Web
- ▶ XML
- ▶ XPath/Xquery
- ▶ XSL/XSLT

Recommended Reading

1. Adobe Systems Incorporated, PDF reference. Sixth edn., 2006.
2. Boag S., Chamberlin D., Fernández M.F., Florescu D., Robie J., and Siméon J. XQuery 1.0: an XML query language. World Wide Web Consortium (W3C), 2007.
3. Bray T., Paoli J., Sperberg-McQueen C.M., Maler E., and Yergeau F., Extensible Markup Language (XML) 1.0. World Wide Web Consortium (W3C), fourth edn., 2006.
4. Draper D. Mapping between XML and Relational Data. In XQuery from the experts: a guide to the W3C XML query language. chap. 6, Addison Wesley, 2003.
5. Fallside D.C. and Walmsley P. XML Schema Part 0: Primer. World Wide Web Consortium (W3C), second edn., 2004.
6. Furuta R., Scofield J., and Shaw A. Document formatting systems: survey, concepts, and issues. ACM Comput. Surv., 14 (3):417–472, 1982.
7. Goldfarb C.F. (ed.) Information processing – text and office systems – Standard Generalized Markup Language (SGML). International Organization for Standardization, Geneva, Switzerland, 1986, International Standard ISO 8879.
8. Kay M. XSL transformations (XSLT) version 2.0. World Wide Web Consortium (W3C), 2007.
9. Knuth D.E. and Plass M.F. Breaking paragraphs into lines. Software Prac. Exper., 11(11):1119–1184, 1982.
10. Microsoft Office Word 2007 Rich Text Format (RTF) Specification. 2007, version 1.9. Downloaded from microsoft.com, November 2007.
11. OASIS, Open Document Format for Office Applications (OpenDocument) v1.1. 2007, http://docs.oasis-open.org/office/v1.1/OS/, 2007.
12. Shanmugasundaram J., Shekita E., Barr R., Carey M., Lindsay B., Pirahesh H., and Reinwald B. Efficiently publishing relational data as XML documents. VLDB J., 10(2–3), 2001.
13. Simske S.J. and Baggs S.C. Digital capture for automated scanner workflows. In Proc. 2004 ACM Symp. on Document Engineering, 2004, pp. 171–177.
14. Tatarinov I., Viglas S.D., Beyer K., Shanmugasundaram J., Shekita E., and Zhang C. Storing and querying ordered XML using a relational database system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 204–215.

Document Retrieval

▶ Information Retrieval

Document Segmentation

▶ Text Segmentation

Document Summarization

▶ Text Summarization

Document Term Weighting

▶ Information Retrieval Models

Document Visualization

▶ Text Visualization

Documents

▶ Document representations (inclusive native and relational)

Domain Relational Calculus

▶ Relational Calculus

Downward Closure Property

▶ Apriori Property and Breadth-First Search Algorithms

DRM

▶ Digital Rights Management

Dublin Core

James Caverlee[1], Prasenjit Mitra[2],
Mary Laarsgard[3]
[1]Texas A&M University, College Station, TX, USA
[2]Pennsylvania State University, University Park, PA, USA
[3]University of California-Santa Barbara,
Santa Barbara, CA, USA

Definition

Dublin Core is a standardized metadata set for describing information resources like documents, videos, images, services, and other digital artifacts. Dublin Core is intended to provide a simple metadata model that can be adopted across a wide range of communities in an effort to enhance semantic interoperability. The Dublin Core Metadata Element Set has been formally endorsed by a number of standards bodies including ISO [5], NISO [7], and IETF [6].

Historical Background

In 1995, the Online Computer Library Center (OCLC) and the National Center for Supercomputing Applications (NCSA) co-sponsored a workshop to address the challenge of developing a common metadata set for describing networked resources [10]. The workshop was motivated in part by the explosive growth of the Web in the early 1990s and the inherent difficulty in finding Web resources. The name "Dublin" in Dublin Core derives from the location of this first workshop in Dublin, Ohio; the term "Core" refers to the basic importance of the elements defined in the metadata standard that can be applied broadly across a wide range of resources.

In the years since the first Dublin Core workshop, the community has hosted annual workshops and conferences devoted to Dublin Core and metadata applications. The continued development and organization of the metadata standard is overseen by the cross-disciplinary Dublin Core Metadata Initiative [3].

Foundations

Supporting information access, organization, and management functionalities in a massively distributed medium like the Web is a serious challenge. In an effort to provide support for these operations, Dublin Core advocates the use of metadata to provide descriptive information about information resources found on the Web, on digital libraries, in enterprize settings, and in other networked domains. In contrast to content-based features of information resources (like the text indexing of a Web document for use in a search engine), the metadata approach can rely on features that describe a resource (and that are not necessarily contained in the resource) to support information discovery, categorization, and other information management functionalities [1].

The Dublin Core metadata set provides a standardized set of metadata elements for describing a wide variety of resources – be they audio files, videos, documents, services, software packages, images, etc. By design, Dublin Core is simple so that metadata may be generated by experts and non-experts alike. The basic Dublin Core standard supports 15 different metadata elements that can be applied to a resource (as shown in Fig. 1). Each element may be used multiple times to describe a single resource, and only the necessary elements need be used in a description of a resource.

The 15 elements are intended to be core descriptors that could be applied regardless of the particular domain of interest. To illustrate, the element "Creator" could refer to the painter of a picture, the author of a book, or to an organization that publishes a software tool. Similarly, the "Description" element could refer to a free text description of a resource, the abstract of an article, a table of contents, or some other descriptive image or text. Although Dublin Core supports great flexibility in the use of these metadata elements, it is good practice to rely on some standard vocabularies for certain elements, e.g., to use standard MIME media types for the element "Format."

As a concrete example, consider a resource like an academic research paper. Figure 2 illustrates some relevant Dublin Core metadata for a sample paper.

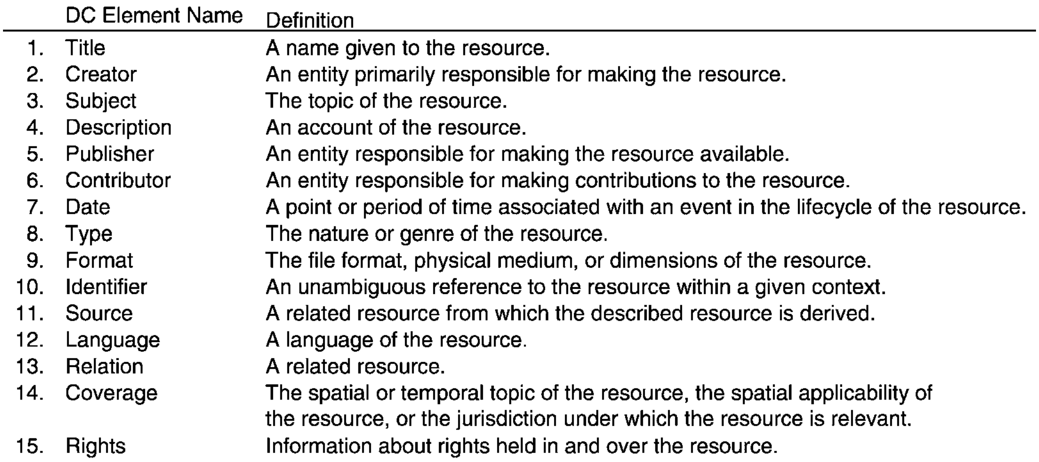
Qualified Dublin Core

Dublin Core additionally supports optional *qualifiers* that may be used to extend and refine the 15 basic Dublin Core elements. Qualifiers can be used for either (i) element refinement; or (ii) declaring an encoding scheme [2,9].

Element refinement narrows the meaning of an element. For example, the element "Date" can be refined to "Created," meaning that the metadata associated with the element "Date" refers to a creation date of the resource, and not to the date it was modified. Alternatively, "Date" could be refined to "Modified" if the semantics of "Date" are meant to convey the date the resource was modified, but not created.

Declaring an encoding scheme provides additional information about the element that can be used for interpreting the meaning of the element value. For example, the "Subject" element may be qualified with an encoding scheme for the Library of Congress Subject Headings (LCSH), a standard set of subject headings that are widely adopted in libraries. By relying on a controlled vocabulary instead of free text, the "Subject" element may provide clearer meaning to applications relying on Dublin Core.

Since Dublin Core is intended to be simple and easy-to-use, applications built to work with Dublin Core metadata should be able to ignore qualifiers entirely and still function in a useful way, albeit with some loss of expressiveness.



	DC Element Name	Definition
1.	Title	A name given to the resource.
2.	Creator	An entity primarily responsible for making the resource.
3.	Subject	The topic of the resource.
4.	Description	An account of the resource.
5.	Publisher	An entity responsible for making the resource available.
6.	Contributor	An entity responsible for making contributions to the resource.
7.	Date	A point or period of time associated with an event in the lifecycle of the resource.
8.	Type	The nature or genre of the resource.
9.	Format	The file format, physical medium, or dimensions of the resource.
10.	Identifier	An unambiguous reference to the resource within a given context.
11.	Source	A related resource from which the described resource is derived.
12.	Language	A language of the resource.
13.	Relation	A related resource.
14.	Coverage	The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant.
15.	Rights	Information about rights held in and over the resource.

Dublin Core. Figure 1. The 15 Simple Dublin Core elements [4].

DC element name	Value
Title	Mining association rules between sets of items in large databases
Creator	Rakesh agrawal and tomasz imielinski and arun swami
Subject	Databases, data mining, association rules
Description	Seminal research paper that describes an efficient algorithm for extracting association rules from a database of customer transactions
Publisher	ACM SIGMOD International Conference on Management of Data
Date	2003
Language	English
Format	Application/pdf

Dublin Core. Figure 2. Sample Dublin Core metadata for a research paper.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">

   <rdf:Description rdf:about="http://rakesh.agrawal-family.com/papers/sigmod93assoc.pdf">

      <dc:title>Mining Association Rules Between Sets of Items in Large Databases </dc:title>
      <dc:creator>Rakesh Agrawal and Tomasz Imielinski and Arun Swami</dc:creator>
      <dc:subject>Databases, data mining, association rules</dc:subject>
      <dc:description>Seminal research paper that describes an efficient algorithm
                 for extracting association rules from a database of customer
                 transactions.</dc:description>
      <dc:publisher>ACM SIGMOD International Conferenceon Management of Data</dc:publisher>
      <dc:date>2003</dc:date>
      <dc:language>English</dc:language>
      <dc:format>application/pdf</dc:format>
   </rdf:Description>
</rdf:RDF>
```

Dublin Core. Figure 3. Example RDF/XML markup using Dublin Core.

Encoding Dublin Core

Dublin Core metadata can be represented in a number of different formats, including plain text, HTML, XML, and RDF. With the rise in interest in the Semantic Web and other knowledge management activities, there has been a push to see Dublin Core widely adopted using RDF [8]. As an illustration of encoding Dublin Core in RDF, Fig. 3 shows the RDF-encoded metadata for the same resource described in Fig. 2.

Key Applications

Web, Semantic Web, digital libraries, business-to-business exchange.

Cross-references

► Metadata

► Metadata Registry

Recommended Reading

1. Cathro W. Metadata: An Overview. Standards Australia Seminar, 1997.
2. Dublin Core Metadata Initiative. Dublin Core Qualifiers, 2000.
3. Dublin Core Metadata Initiative. 2008, http://dublincore.org.
4. Dublin Core Metadata Initiative. Dublin Core Metadata Element Set, Version 1.1, 2008.
5. International Organization for Standardization. ISO 15836-2003 Information and Documentation – The Dublin Core Metadata Element Set, 2003.
6. Internet Engineering Task Force. IETF RFC 5013 – The Dublin Core Metadata Element Set, 2007.
7. National Information Standards Organization. ANSI/NISO Z39.85-2007 The Dublin Core Metadata Element Set, 2007.
8. Nilsson M., Powell A., Johnston P., and Naeve A. Expressing Dublin Core Metadata Using the Resource Description Framework (RDF), 2008.
9. Weibel S. The State of the Dublin Core Metadata Initiative. Bull. Am. Soc. Inf. Sci., 25(5):18–22, 1999.
10. Weibel S., Godby J., Miller E., and Daniel R. OCLC/NCSA Metadata Workshop Report, 1995.

Dump

► Logging and Recovery

Duplicate Detection

▶ Record Matching

▶ Semantic Data Integration for Life Science Entities

Duplicate Semantics

▶ Bag Semantics

Duplication

▶ Replication

Durability

▶ ACID Properties

Duration

▶ Time Interval

DVDs

▶ Storage Devices

DW

▶ Data Warehouse

Dynamic Graphics

Dianne Cook
Iowa State University, Ames, IA, USA

Synonyms

Multivariate data visualization; Multiple linked plots; Motion graphics; Rotation; Tour; Animation

Definition

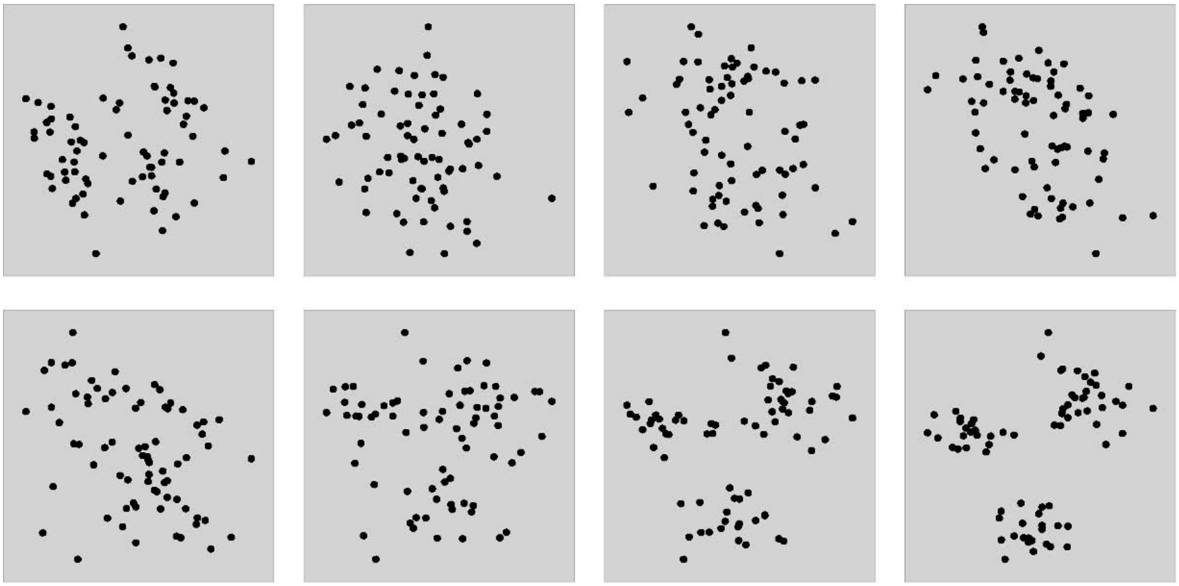
Dynamic graphics for data, means simulating motion or movement using the computer. It may also be thought of as multiple plots linked by time. Two main examples of dynamic graphics are animations, and tours. An animation, very generally defined, may be produced for time-indexed data by showing the plots in time order, for example as generated by an optimization algorithm.

A tour is designed to study the joint distribution of multivariate data, in search of relationships that may involve several variables. It is created by generating a sequence of low-dimensional projections of high-dimensional data – typically 1D or 2D – so that many different aspects of high-dimensional data can be observed. Tours are thus used to find interesting lower-dimensional projections of the data, ideally for data which contains real-valued variables. The data $\mathbf{X}_{n\times p}$ is projected into $\mathbf{A}_{p\times d}$ to produce a data projection $\mathbf{Y}_{n\times d} = \mathbf{X}_{n\times p}\mathbf{A}_{p\times d}$. The projection matrix $\mathbf{A}_{p\times d}$ is orthonormal. The coefficients in $\mathbf{A}_{p\times d}$ are generated so that all values have some given probability of being chosen and consecutive projections are close to the previous, to provide apparently smooth motion.

Historical Background

The grand tour was defined and named by Asimov [2]. It computes the projections uniformly from a $(p-1)$-D sphere. All possible projections are equally likely. To provide a smooth path, he generated sequential projections by following a path on a high-dimensional torus. Buja and Asimov [3] further developed the grand tour by using an interpolated geodesic random walk, between randomly generated basis planes. This improvement ensures that the grand tour efficiently covers the space of all projections (Grassmann manifold), and that within-plane spin is absent. The mathematics and algorithms for this approach is described in detail by Buja et al. [4]. A simpler approach to generating a grand tour is used by Tierney [13].

The grand tour was first implemented by Buja et al. [5] and the work developed to include a guided tour, where projections are chosen according to a measure of interestingness (e.g., projection pursuit) and a correlation tour, where two sets of variables are toured using 1D projections in the horizontal and vertical directions. A correlation tour is related to regression analysis, modeling one or more response variables against many explanatory variables. The guided tour was developed further by Cook et al. [8].



Dynamic Graphics. Figure 1. A sequence of projections from a tour using 2D projections of 6D data. Within seconds of watching the tour clustering of the observations can be seen.

Cook and Buja [7] developed manual controls for the tour, enabling the user to manually change the projection matrix coefficients, to assess the impact of selected variables on the visible data structure. Wegman [14] developed the full dimensional grand tour, where the projection is displayed in parallel coordinates, and also as a scatterplot matrix [6]. He further developed the image grand tour to study remote sensing data [15]. Scott [12] developed the density grand tour, where each 1D projection is shown as a density. Huh [11] developed a grand tour with a tail. Locations of points in previous projections are plotted for a given time period in future projections, giving a fuller sense of the motion of points.

Andrews curves [1] are a pre-cursor to the grand tour. A general explanation of tours suitable for a reasonably untechnical audience can be found in a book chapter by Cook et al. [9].

Foundations

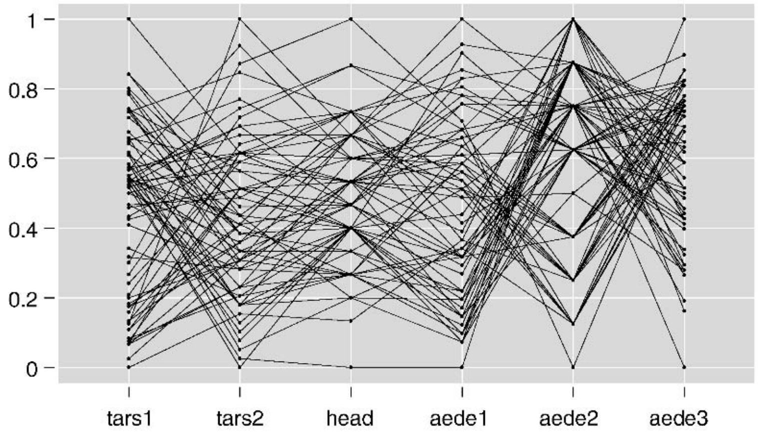
The tour is ideally suited for examining the joint distribution of multivariate data, for relatively small p (on the order of small tens rather than hundreds), where the variables take on real-values. Figures 1–3 compare structure detection in a tour with that in a parallel coordinate plot.

In Fig. 1 a set of eight 2D projections of 6D data are taken from the movie of projections shown in a grand tour. The data has three very well-separated clusters, that are elliptically shaped in the six dimensions. It should be noted that in this data, the three clusters are not perfectly visible in any pair of the six variables. Within seconds of viewing the tour, this is obvious to even the most novice audience. Viewers clue to the clusters by separations between points in certain projections and also the motion patterns of the points.

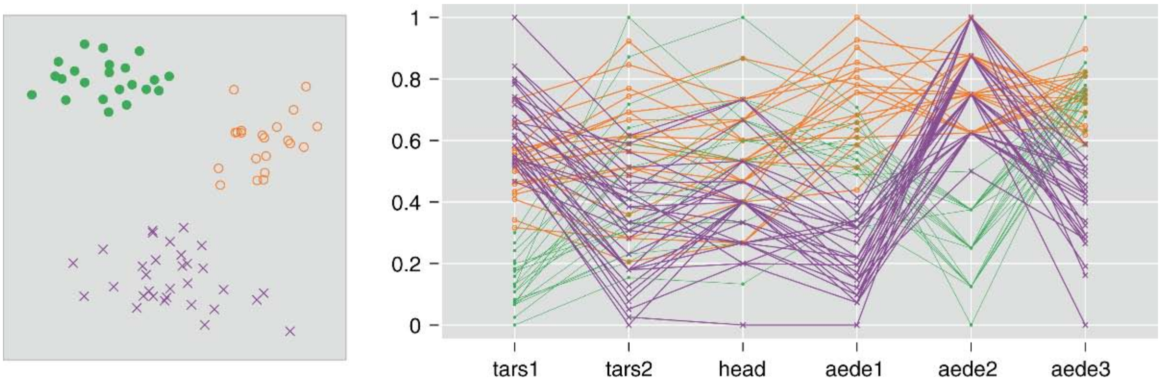
In a parallel coordinate plot (Fig. 2) the three clusters are not readily detectable. A trained eye can easily see two clusters, by recognizing two groups of similar trends in the lines. The third cluster of lines is a little more difficult to discern. If the three clusters are identified using color, then they become much more visible in the parallel coordinate plot (Fig. 3).

Generally tours are better than parallel coordinate plots for this type of data and structure such as this, clusters, outliers, linear or nonlinear dependencies, and relatively small dimension. Parallel coordinates are more appropriate when there is a mix of categorical and continuous variables, or when there are a large number of variables.

Tours are usually implemented with the inclusion of some navigation support. Figure 4 provides an example. The circle with radial line segments in the center of the plot represents the projection matrix, **A**, which for this projection is:



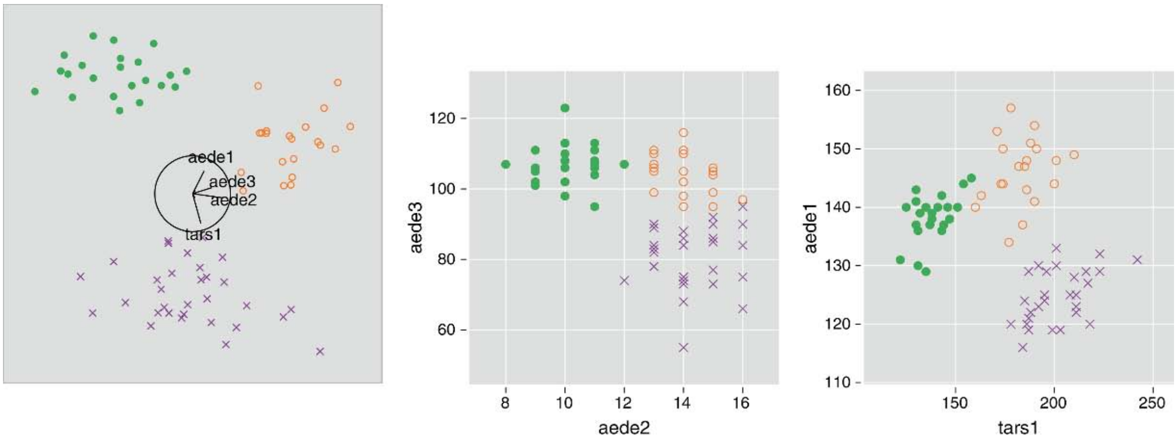
Dynamic Graphics. Figure 2. A parallel coordinate plot of the same 6D data as shown in Fig. 1. Two clusters of different line traces are readily seen but the three clusters are not so obviously recognized.



Dynamic Graphics. Figure 3. When the three clusters are colored they are more recognizably clusters in the parallel coordinates plot.

$$\mathbf{A} = \left[\begin{array}{rr|l} 0.215 & -0.775 & \mathit{tars}1 \\ 0.065 & 0.081 & \mathit{tars}2 \\ 0.053 & -0.102 & \mathit{head} \\ 0.298 & 0.589 & \mathit{aede}1 \\ 0.786 & -0.108 & \mathit{aede}2 \\ 0.490 & 0.155 & \mathit{aede}3 \end{array}\right]$$

Look at the magnitude and sign of these values to interpret the plot structure. The variables aede2 and aede3 contribute the most to the horizontal direction (first column). In this direction the orange (open circle) cluster is separated from the green (solid circle) cluster and to some extent the purple (cross) cluster. The variables tars1 and aede1 contribute the most to the vertical direction (column 2), with the contribution of tars1 being negative. In this direction the green (solid circle) cluster is separated from the purple (cross) cluster and to some extent the orange (open circle) cluster. This suggests that variables aede2, aede3 contribute to distinguishing between the orange (open circle) cluster from the other two, and variables tars1, aede1 contribute to distinguishing between green (solid circle) and purple (cross) cluster. These interpretations would be checked using pairwise scatterplots (middle, right), univariate plots, or parallel coordinates (Fig. 3). Using these additional aids one would decide that tars1 separates green (solid circle) from purple (cross), and that aede2 separates green (solid circle) from orange(open circle). It is really a combination of these variables which makes the difference between the clusters marked, but individual variables contribute in partial ways to the separation.



Dynamic Graphics. Figure 4. Circle inside the tour projection (*left*) provides navigation support for the tour. Scatterplots (*middle, right*) help to confirm the interpretation of structure.

Key Applications

Tours are a critical part of many different types of multivariate analyses: clustering, classification, multivariate tests, principal components analysis and multidimensional scaling. These types of methods are used in many, many different applications, including astronomy, biology, physics, social science, education, geology, agronomy, ecology, credit risk, defense (Wegman et al. [15] use the image grand tour to detect land mines in satellite images). The tour also can be used to study the geometry of high-dimensional spaces. Tours are an integral part of exploratory data analysis.

Cook and Swayne [10] provide many more examples where the tour might be used to gain insight into multivariate structure in data, or in the performance of multivariate methods and algorithms.

Future Directions

There is a lot of scope for research in dynamic graphics. For the tour, one might investigate different probability distributions for choosing projections, particularly for handling much larger numbers of variables, and making use of sparseness. Studies might be done to suggest the optimal viewing times for watching a tour in order to recognize different types of structure. Someone might study tours on something other than Euclidean space, to enable the study of more complex data. For example, a tour on the space of all permutations could be used to explore categorical data. Other methods for guiding the tour would be useful. Large data poses a problem, because points get over-plotted. The projections might be represented as density plots or convex hulls, if these can be computed sufficiently fast. There are interesting connections with statistical theory that might be explored. For example, most random projections of multivariate data look approximately Gaussian, is related to the Central Limit Theorem.

URL to Code

http://www.ggobi.org

Cross-references

▶ Business Intelligence
▶ Classification
▶ Clustering
▶ Cluster Visualization
▶ Curse of Dimensionality
▶ Data Mining
▶ Dimension
▶ Dimensionality Reduction
▶ Exploratory Data Analysis
▶ Feature Selection for Clustering
▶ Geographic Information System
▶ Information Extraction
▶ Machine Learning in Computational Biology
▶ Mining of Chemical Data
▶ Multidimensional Scaling
▶ Multivariate Visualization Methods
▶ Parallel Coordinates
▶ Principal Components Analysis
▶ Spatial Data Mining
▶ Visual Analytics
▶ Visual Classification
▶ Visual Clustering
▶ Visual Data Mining

Recommended Reading

1. Andrews D.F. Plots of high-dimensional data, Biometrics, 28:125–136, 1972.
2. Asimov D. The grand tour: a tool for viewing multidimensional data, SIAM J. Sci. Stat. Comput., 6(1):128–143, 1985.
3. Buja A. and Asimov D. Grand tour methods: an outline, Comput. Sci. Stat., 17:63–67, 1986.
4. Buja A., Cook D., Asimov D., and Hurley C. Computational Methods for High-Dimensional Rotations in Data Visualization, In Handbook of Statistics: Data Mining and Visualization, C.R. Rao, E.J. Wegman, J.L. Solka (eds.). Elsevier/North-Holland, 2005, pp. 391–414.
5. Buja A., Hurley C., and McDonald J.A. A data viewer for multivariate data, Comput. Sci. Stat., 17(1):171–174, 1986.
6. Carr D.B., Wegman E.J. and Luo Q. ExplorN: Design Considerations Past and Present, Technical Report 129, Center for Computational Statistics, George Mason University, Fairfax, VA, 1996.
7. Cook D. and Buja A. Manual controls for high-dimensional data projections, J. Comput. Graph. Stat., 6(4):464–480, 1997.
8. Cook D., Buja A., Cabrera J., and Hurley C. Grand tour and projection pursuit, J. Comput. Graph. Stat., 4(3):155–172, 1995.
9. Cook D., Lee E.-K., Buja A., and Wickham H. Grand tours, projection pursuit guided tours and manual controls, In Handbook of Data Visualization, C.-H. Chen, W. Härdle A. Unwin (eds.). Springer, Berlin, Germany, 2006.
10. Cook D. and Swayne D.F. Interactive and Dynamic Graphics for Data Analysis: With R and GGobi, Springer, New York, 2007.
11. Huh M.Y. and Kim K. Visualization of Multidimensional Data Using Modifications of the Grand Tour. J. Appl. Stat., 29 (5):721–728, 2002.
12. Scott D. Incorporating density estimation into other exploratory tools, In Proc. of the Section on Statistical Graphics, 1995, pp. 28–35.
13. Tierney L. LispStat: An Object-Oriented Environment for Statistical Computing and Dynamic Graphics, Wiley, New York, 1991.
14. Wegman E.J. The Grand Tour in *k*-Dimensions, Technical Report 68, Center for Computational Statistics, George Mason University, 1991.
15. Wegman E.J., Poston W.L., and Solka J.L. Image Grand Tour, In Automatic Target Recognition VIII – Proc. SPIE, 3371, SPIE, Bellingham, WA, 1998, pp. 286–294.

Dynamic Integrity Constraints

► Temporal Integrity Constraints

Dynamic Taxonomies

► Faceted Search

Dynamic Web Pages

Maristella Matera
Polytechnico di Milano University, Milan, Italy

Definition

They are Web pages that are composed at run-time, by dynamically extracting contents from a data source and composing them into pre-defined page templates.

Key Points

Historically, hypertext navigation was meant as a way to move among “static” documents, i.e., Web pages whose HTML code includes both the content to be presented, as well as the mark-up tags determining content rendition. Real-life Web applications however require the capability of serving to the users pages that dynamically publish content coming from one or more data sources. For example, the content of the home page of a news magazine is refreshed daily, by extracting the latest news from the news repository. This requirement goes beyond the original capabilities of the HTTP protocol, which is designed to exchange requests and resources between the browser and the server, and not to govern the process by which the desired resource is built.

Some server-side technologies have therefore been introduced to overcome these limitations and to enable the construction of Web pages “on the fly.” The most common solution is to adopt server-side scripting technologies (such as JSP or PHP), which enable inserting into an HTML page template some programming instructions that a server-side program execute to compute the contents to be extracted dynamically from the application data source. The result sent back to the client is then a properly formatted HTML page, including the extracted contents.

Other solutions imply the extension of the Web server, through execution engines (for example Java Servlet API) able to serve the requests for the dynamic construction of Web pages.

Cross-references

► Web Characteristics and Evolution

eAccessibility

Constantine Stephanidis[1,2]
[1]Foundation for Research and Technology – Hellas (FORTH), Heraklion, Greece
[2]University of Crete, Heraklion, Greece

Definition

eAccessibility refers to the access of Information and Communication Technologies (ICT) by people with disabilities, with particular emphasis on the World Wide Web. It is the extent to which the use of an application or service is affected by the user's particular functional limitations or abilities (permanent or temporary). eAccessibility can be considered as a fundamental prerequisite of usability.

Historical Background

The percentage of disabled citizens has increased dramatically in the last century due to life condition improvements, higher life expectancy, and population aging. This trend is anticipated to further increase in the next decades.

Traditional efforts to provide eAccessibility for users with disabilities were based on the adaptation of applications and services originally developed for able-bodied users. In the context of the Information Society, this raises the fundamental issue of granting to disabled citizens access to a variety of technologies that become progressively more entangled with all types of everyday activities.

Early technical approaches to eAccessibility mainly focused on two directions. The first treats each application separately, and takes all the necessary implementation steps to arrive at an alternative accessible version (*product-level adaptation*). Practically, product-level adaptation often implies redevelopment from scratch. The second approach "intervenes" at the level of the particular interactive application environment (e.g., MS-Windows) in order to provide appropriate software and hardware technology so as to make that environment accessible through alternative means (*environment-level adaptation*). The latter option extends the scope of eAccessibility to cover potentially all applications running under the same interactive environment, rather than a single application, and is therefore acknowledged as a more promising strategy.

The above approaches have given rise to several methods for addressing eAccessibility, including techniques for the configuration of input/output at the level of the user interface, and the provision of Assistive Technologies. Popular Assistive Technologies supporting eAccessibility include screen readers and Braille displays for blind users, screen magnifiers for users with low vision, alternative input and output devices for motor impaired users (e.g., adapted keyboards, mouse emulators, joystick, binary switches), specialized browsers (e.g., [10]), and text prediction systems (e.g., [4]). Assistive Technologies are legally defined in the US as "Any item, piece of equipment, or system, whether acquired commercially, modified, or customized, that is commonly used to increase, maintain, or improve functional capabilities of individuals with disabilities [15]."

Despite progress, Assistive Technologies and dedicated design approaches have been criticized for their essentially reactive nature [13]. Although the "reactive" approach to eAccessibility may be the only viable solution in many cases, it suffers from potentially serious shortcomings, such as limited and low quality access, as well as difficulties in development, maintenance, and keeping pace with technological evolution.

Therefore, the need for more systematic and proactive approaches to the provision of eAccessibility has emerged, leading to the concepts of Universal Access and Design for All.

Foundations

Universal Access implies the accessibility and usability of Information Society Technologies by anyone, anywhere, anytime, with the aim to enable equitable access and active participation of potentially all citizens in existing and emerging computer-mediated human activities. This can be achieved by developing universally accessible and usable products and services, which are capable of accommodating individual user requirements in different contexts of use and independently of location, target machine, or run-time environment. The discipline of Human-Computer Interaction (HCI) plays a critical role towards ensuring Universal Access to computer-based products and services, as users experience new technologies through contact with their user interfaces.

In the context of Universal Access, eAccessibility refers to the extent to which the use of an application or service is affected by the user's particular functional limitations or abilities (permanent or temporary), as well as by other contextual factors (e.g., characteristics of the environment). This implies that, for each user task of an application or service, and taking into account specific functional limitations and abilities, as well as other relevant contextual factors, there is a sequence of input actions and associated feedback, via accessible input/output devices, which leads to successful task accomplishment.

Universal Access is predominantly an issue of design. Thus, the question arises of how it is possible to design systems that permit systematic and cost-effective approaches to accommodating the requirements of potentially all users. To this effect, the concept of Design for All has been revisited in recent years in the context of HCI [14].

Design for All, or Universal Design, is well known in several engineering disciplines, such as, for example, civil engineering and architecture, with many applications in interior design, building and road construction. In the context of Universal Access, Design for All either subsumes, or is a synonym of, terms such as accessible design, inclusive design, barrier-free design, universal design, etc., each highlighting different aspects of the concept. It has a broad and multidisciplinary connotation, abstracting over different perspectives, such as:

1. Design of interactive products, services and applications, which are suitable for most of the potential users without any modifications. Related efforts mainly aim to formulate eAccessibility guidelines and standards. This is pursued in the context of international collaborative initiatives. A significant example is the effort carried out by the W3C-WAI Initiative in the area of Web accessibility guidelines [16]. Another source of web accessibility guidance is Section 508 of the US Rehabilitation Act [15].
2. Design of products which have standardized interfaces, capable of being accessed by specialized user interaction devices. An example of this approach is the Universal Remote Console, defined as a combination of hardware and software that allows a user to control and view displays of any (compatible) electronic and information technology device or service (or "target") in a way that is accessible and convenient to the user [18].
3. Design of products which are easily adaptable to different users (e.g., by incorporating adaptable or customizable user interfaces). The latter approach fosters a conscious and systematic effort to proactively apply principles and methods, and employ appropriate tools, in order to develop interactive products and services which are accessible and usable by all citizens in the Information Society, thus avoiding the need for a posteriori adaptations, or specialized design. This entails an effort to build access features into a product starting from its conception, throughout the entire development life-cycle [12].

Independently from the approach through which it may be achieved, accessibility aims to make the user experience of people with diverse functional or contextual limitations as near as possible, in terms of task accomplishment, to that of people without such limitations. Under a Universal Access perspective, accessibility has to be "designed into" the system rather than decided upon and implemented a posteriori. This raises several requirements regarding the methods, techniques and tools which can be used to integrate accessibility throughout the development lifecycle of interactive products and services. In this context, eAccessibility implies the concurrent (or adaptation driven) availability of alternative modalities, external devices and interaction styles to accommodate different needs. Therefore, multimodality plays a very important role [3].

Key Applications

eAccessibility is relevant to all interactive applications and services addressing citizens in the Information Society, in a variety of domains of everyday life, including healthcare, access to information, education, entertainment, and public administration. Some significant examples are reported below.

Web Accessibility

Access of disabled users to the World Wide Web involves the accessibility of both web content and browsing applications (web browsers). Web content accessibility is usually addressed through conformance to guidelines [16]. Various tools are available for the automatic or semiautomatic accessibility assessment and repair of web content [17]. More recent approaches go in the direction of automatic web content transformation to more accessible versions (e.g., [8]), as well as server side automatic adaptation [1]. Accessibility of web browsers is addressed through dedicated design for particular target user groups (e.g., blind users, [10]) as well as adaptation based approaches. The universally accessible AVANTI web browser [1] provides an interface to web-based information systems for a range of user categories, including: (i) "able-bodied" people, (ii) blind people, (iii) motor-impaired people with different degrees of difficulty in employing traditional input devices.

Media Accessibility

The term "rich media" indicates a broad range of digital interactive media downloadable or embedded in a web site, including video, animations, images, and sound. This type of content can be viewed or used offline with media players. Media accessibility is usually addressed through the provision of equivalent information perceivable through different senses, e.g., captioning, audio description, subtitling, and sign language translation. This requires the availability of authoring tools which support the appropriate provision of such information (e.g., [11]). Access to rich media by people with disabilities also requires the availability of accessible media players (e.g., [10]).

Accessibility in Education

eLearning systems typically rely on repositories of online materials that are made available to learners and teachers. The accessibility of eLearning content implies more than web or media accessibility, as it requires not only equivalence of information among different modalities, but also its appropriateness for the learning experience of users with diverse abilities. Meta-data classifications are under elaboration, which allow the classification of learning content according to the needs and preferences for alternative presentations of resources, methods of controlling resources, equivalent to the resources themselves and enhancements or support required by the user [9]. Access to educational material by people with disabilities also implies accessible software for the interactive delivery of such material, as well as authoring tools supporting the provision of educational content in an accessible form (e.g., [7]).

Game Accessibility

Computer games are usually quite demanding in terms of motor, sensor and mental skills needed for interaction control, and they often require mastering inflexible, quite complicated, input devices and techniques. These facts often render games inaccessible to a large percentage of people with disabilities. Furthermore, with respect to HCI, computer games have fundamental differences from all the other types of software applications for which accessibility guidelines and solutions are becoming available. Current approaches to game accessibility include the development of mainstream games compatible with the use of assistive technologies, the development of special-purpose games, optimally designed for people with disabilities, like audio-based games for blind people and switch-based games for the motor-impaired, as well as the development of universally accessible games. Examples of the latter approach are the UA-Chess web-based chess game [5] and Access Invaders ([6] – a universally accessible version of the popular classic "Space Invaders" action game), which can be played concurrently by people with different abilities and preferences, including people with disabilities (e.g., low-vision, blind and hand-motor impaired), and Access Invaders [17], a universally accessible version of the popular classic "Space Invaders" action game.

Future Directions

In the years ahead, as a result of the increasing demand for ubiquitous and continuous access to information and services, Information Society Technologies are anticipated to evolve towards a new computing paradigm referred to as Ambient Intelligence. Such an environment will be characterised by invisible (i.e., embedded) computational power in everyday appliances and other surrounding physical objects, and populated by intelligent mobile and wearable devices. Ambient Intelligence will have profound consequences on the type, content and functionality of the emerging products and services, as well as on the way people will interact with them, bringing about multiple new requirements for the development of Information Society Technologies. In such a dynamically evolving technological environment, accessibility and usability of such a complex technological environment by users with different characteristics and requirements can not be addressed through solutions introduced once the main building components of the new environment are in place. In such a context, the concepts of Universal Access and Design for All acquire critical importance towards streamlining accessibility into the new technological environment through generic solutions [2]. However, in the context of Ambient Intelligence, Universal Access will need to evolve in order to address a series of new challenges posed by the evolving technological environment. Such challenges include the distribution of interaction in the physical environment, the optimal degree of automation vs. human control, the identification of concrete human needs and requirements in such an environment, as well as issues related to health and safety, privacy and security, and social implications. In order to support the development of Universal Access solutions for Ambient Intelligence, new methodologies to capture requirements, appropriate development methods and tools, as well as design knowledge in the form of guidelines and standards will have to be provided, thus dramatically altering the current notion and practices of eAccessibility.

Cross-references

- ▶ Human-Computer Interaction
- ▶ Multimodal Interfaces
- ▶ Usability
- ▶ Visual Interfaces

Recommended Reading

1. Doulgeraki C., Partarakis N., Mourouzis A., Antona M., and Stephanidis C. Towards Unified Web-based User Interfaces. Technical Report 394, ICS-FORTH, Heraklion, Crete, Greece, 2007, p. 283. Available online at: http://www.ics.forth.gr/ftp/tech-reports/2007/2007.TR394_Towards_Unified_Web-based_UI.pdf.
2. Emiliani P.-L. and Stephanidis C. Universal access to ambient intelligence environments: opportunities and challenges for people with disabilities. IBM Syst. J. (Special Issue on Accessibility), 44(3):605–619, 2003.
3. Furner S., Schneider-Hufschmidt M., Groh L., Perrin P., and Hine N. Human factors guidelines for multimodal interaction, communication and navigation. In Proc. 19th Int. Symp. on Human Factors in Telecommunication, 2003.
4. Garay-Vitoria N. and Abascal J. Text prediction systems: a survey. Universal Access Inf. Soc., 4(3):188–203, 2006.
5. Grammenos D., Savidis A., and Stephanidis C. *1*UA-Chess: A universally accessible board game. In Universal Access in HCI: Exploring New Interaction Environments - Proc. 11th Int. Conf. on Human-Computer Interaction (HCI International 2005), vol. 7, C. Stephanidis (ed.). Lawrence Erlbaum, Mahwah, NJ, 2005.
6. Grammenos D., Savidis A., Georgalis Y., and Stephanidis C. *1*Access invaders: Developing a universally accessible action game. In Computers Helping People with Special Needs, Proc. Tenth Int. Conf. Springer, 2006, pp. 388–395.
7. Grammenos D., Savidis A., Georgalis Y., Bourdenas T., and Stephanidis C. Dual educational electronic textbooks: the starlight platform. In Proc. 9th Int. ACM SIGACCESS Conf. on Computers and Accessibility, 2007, pp. 107–114.
8. Hanson V.L. and Richards J.T. A web accessibility service: An update and findings. In Proc. 6th Int. ACM Conf. on Assistive Technologies, 2004, pp. 169–176.
9. IMS Global Learning Consortium, http://www.imsglobal.org/accessibility/index.html.
10. Miyashita H., Sato D., Takagi H., and Asakawa C. Making multimedia content accessible for screen reader users. In Proc. Int. Cross-Disciplinary Conf. on Web Accessibility, 2007, pp. 126–127.
11. National Center for Accesible Media, http://ncam.wgbh.org/webaccess/magpie/.
12. Stephanidis, C. (ed.). User Interfaces for All - Concepts, Methods, and Tools. Lawrence Erlbaum, Mahwah, NJ, 2001.
13. Stephanidis C. and Emiliani P.L. Connecting to the information society: a European perspective. Technol. Disabil. J., 10(1):21–44, 1999.
14. Stephanidis C., Salvendy G., Akoumianakis D., Bevan N., Brewer J., Emiliani P.L., Galetsas A., Haataja S., Iakovidis I., Jacko J., Jenkins P., Karshmer A., Korn P., Marcus A., Murphy H., Stary C., Vanderheiden G., Weber G., and Ziegler J. Toward an information society for all: an international R&D Agenda. Int. J. Human-Comput. Interaction, 10(2):107–134, 1998.
15. The Rehabilitation Act Amendments (Section 508). Available at http://www.section508.gov/.
16. W3C – WAI. (1999). Web Content Accessibility Guidelines 1.0. Available at http://www.w3.org/TR/WCAG10/.
17. Web Accessibility Evaluation Tools: Overview. http://www.w3.org/WAI/ER/tools/.
18. Zimmermann G., Vanderheiden G., and Gilman A. Universal remote console - prototyping for the alternate interface access standard. In Universal Access: Theoretical Perspectives, Practice and Experience - Proc. Seventh ERCIM UI4ALL Workshop, 2002, pp. 524–531.

EAI

▶ Enterprise Application Integration

EC Transactions

► e-Commerce transactions

ECA Rule Action

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Definition

An ECA rule action is typically arbitrary code invoked if the condition of a triggered rule evaluates to true.

Key Points

The ECA rule actions are executed in response to events triggering rules whose conditions evaluates to true. The action is executed as part of a transaction or as a transaction depending on the coupling mode in the system. One major problem is that several actions can be executed concurrently and these may be in conflict. Another major problem is that executing an action may results in events triggering rules, that is, cascading rule triggering.

Cross-references

► ECA Rules
► ECA Rule Condition

ECA Rule Condition

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Definition

An ECA rule condition is either a database query, a logical expression or a call to a subprogram (function or method) executing arbitrary code returning true or false. If database queries are employed, then a non-empty set is equivalent to true and an empty set is equivalent to false.

Key Points

A key issue of an ECA rule condition is that it ought to take parameters carried by the event triggering the rule that, in turn, evaluates the condition. The condition either returns a set (as a result of an SQL query) or a boolean value (as a result of evaluation of the logical expression or the execution of the subprogram). An empty set is equal to false, and a non-empty set is equal to true. Another key issue is that results of the condition evaluation can be used to optimize rule action execution in many cases. Thus, there is a need to pass parameters from the condition evaluation to the rule action execution.

Cross-references

► ECA Rules
► ECA Rule Action

ECA Rules

Mikael Berndtsson, Jonas Mellin
University of Skövde, Skövde, Sweden

Synonyms

Triggers; Event-condition-action rules; Reactive rules

Definition

An ECA rule has three parts: an event, a condition, and an action. The semantics of an ECA rule are: when the event has been detected, evaluate the condition, and if the condition is satisfied, execute the action.

Historical Background

ECA rules are used within active databases for supporting reactive behavior and were first proposed in the HiPAC project [2].

Foundations

The semantics of an ECA rule is straightforward: when an event is detected, evaluate the condition, and if the condition is true, then execute the action. There are a number of reasons why the reactive behavior is abstracted to three different parts [1]:

First of all, events, conditions, and actions have different roles. An event specifies when to trigger a rule, a condition specifies what to check, and an action specifies what to execute in response to the event. Thus, the semantics of an ECA rule is clean, and avoids ad hoc mixing of events, conditions, and actions.

Second, the separation into events, conditions, and actions is also necessary if the application requires flexible execution semantics, i.e., coupling modes.

Third, the distinct roles facilitates optimization. The event part is a prerequisite for evaluating the condition, thus conditions are evaluated when the event occurs, and not always. If the active database only allows specification of rule conditions as query statements against the database, then the condition part is a pure question on the database state and the active database can thereby utilize well known query optimization techniques.

Finally, conceptual modeling of reactive applications is made simpler if there is a corresponding notion of events both within the application domain and within the active database.

Cross-references

▶ Active Database (aDB)

▶ Active Database (Management) System (aDBS/aDBMS)

Recommended Reading

1. Dayal U. Ten years of activity in active database systems: what have we accomplished? In Proc. First Int. Workshop on Active and Real-Time Database Systems, 1995, pp. 3–22.
2. Dayal U., Blaustein B., Buchmann A., et al. S.C. HiPAC: a research project in active, time-constrained database management. Tech. Rep. CCA-88-02, Xerox Advanced Information Technology. Cambridge, MA, USA, 1988.

ECM

▶ Enterprise Content Management

e-Commerce Transactions

Jari Veijalainen
University of Jyvaskyla, Jyvaskyla, Finland

Synonyms

Electronic commerce transactions; EC transactions

Definition

"An electronic transaction is the sale or purchase of goods or services, whether between businesses, households, individuals, governments, and other public or private organizations, conducted over computer mediated networks. The goods and services are ordered over those networks, but the payment and the ultimate delivery of the good or service may be conducted on or off-line." [1]

A mobile e-Commerce transaction is an e-Commerce transaction that is initiated and performed using a *mobile device*, such as a mobile phone or a laptop, over a *wireless access network* or a *short-range wireless link*.

Key Points

E-commerce transaction was defined by OECD [1] from business and statistical perspective. The parties above can be individual customers (C), companies (B), or governments (A). One speaks accordingly about B2C, B2A, B2B, C2C e-Commerce and e-Commerce transactions. Historically, Electronic Data Interchange (EDI) was the first technology used in B2B and B2A e-commerce since 1970's. EDI is based on a number of agreed-upon (EDIFACT) message types whose instances are exchanged by the parties. Notice that B2C, B2A, B2B and C2C e-Commerce are usually governed by different laws. HTTP, HTML, and WWW server and browser technology matured during 1990's and E-commerce for individual customers (B2C, C2C) became possible. Physical (tangible) goods need to be delivered through a separate logistics channel, whereas digital information ("intangible goods") can be delivered through the same digital network as the order. Especially the global digital network ("Internet"), simultaneous emergence of suitable digital contents (software, music, videos, maps, etc.), and global payment infrastructure (credit cards, international banking) have made global digital B2C and other e-Commerce to explode since mid 1990's. Perceived as a distributed application one can require that a typical (B2C) e-Commerce transaction, during which a customer buys a good or goods, must satisfy *money atomicity, goods atomicity, and certified delivery* [2]. A more technical view is that an e-Commerce transaction is in general implemented as a distributed, hierarchical *workflow* crossing organizational borders. While performed, it must satisfy semantically defined atomicity constraints at each level [3]. Money atomicity and other desired properties follow from the constraints and thus the system implementation must enforce them. The challenges are that the steps must be performed by autonomous organizations, the workflow can last days or weeks (order book from USA to Europe) and that the originating entity (customer's terminal) cannot directly control the emerging execution or its duration, because it cannot decide or even know the emerging structure of the workflow. Rather, the latter is dynamically decided by the seller and its subcontractors (suppliers). A further challenge is that the customer can later optionally cancel the order and return the received goods to the seller. The time window typically varies from 7 to 30 days depending on the local consumer protection legislation. Conceptually, this possible phase belongs to the E-commerce transaction, but in practice this is taken care of by a distinct exception handling transaction that is often semi-automatically or manually performed.

Cross-references

▶ Atomicity
▶ Workflow Schema
▶ Workflow Transactions

Recommended Reading

1. Annex 4. The OECD definitions of Internet and E-commerce transactions. Available at: http://www.oecd.org/dataoecd/34/16/2771174.pdf
2. Tygar D. Atomicity in electronic commerce, Chapter 33. In Internet Besieged, P. Denning, D. Denning (eds.). ACM Press and Addison Wesley, USA, 1997, pp. 389– 405.
3. Veijalainen J., Terziyan V., and Tirri H. Transaction management for M-commerce at a mobile terminal. Electronic Commerce Research and Applications., 5(3):229–245, 2006.

Eddies

▶ Adaptive Query Processing

Edge Detection

▶ Image Segmentation

eDictionary

▶ Electronic Dictionary

eEncyclopedia

▶ Electronic Encyclopedia

EERM, HERM

▶ Extended Entity-Relationship Model

Effectiveness Involving Multiple Queries

Eric C. Jensen[1], Steven M. Beitzel[2], Ophir Frieder[3]
[1]Twitter, Inc., San Francisco, CA, USA
[2]Telcordia Technologies, Piscataway, NJ, USA
[3]Georgetown University, Washington, DC, USA

Synonyms

Relevance evaluation of IR systems

Definition

In information retrieval (IR), effectiveness is defined as the relevance of retrieved information to a given query. System effectiveness evaluation typically focuses on the problem of document retrieval: retrieving a ranked list of documents for each input query. Effectiveness is then measured with respect to an environment of interest consisting of the populations of documents, queries, and relevance judgments defining which of these documents are relevant to which queries. Sampling methodologies are employed for each of these populations to estimate a relevance metric, typically a function of precision (the ratio of relevant documents retrieved to the total number of documents retrieved) and recall (the ratio of relevant documents retrieved to the total number of relevant documents for that query). Conclusions about which systems outperform others are drawn from common experimental design, typically focusing on a random sample of queries, each with a corresponding value of the relevance metric. These individual measurements for each query must be aggregated across a sufficiently large sample of queries to draw conclusions with confidence.

Historical Background

Effectiveness evaluation can be divided into user-centered evaluation, where users are given access to an information retrieval system and observed while attempting a task, and system (ad hoc) evaluation where the retrieval process is evaluated over multiple queries without user interaction. The Cranfield experiments defined information retrieval experimentation in terms of a document collection, a set of queries, and a corresponding set of relevance judgments for each of those queries [3]. However, they assumed that the relevance of every document in the collection would be judged for each query. The Text REtrieval Conference (TREC) extended this methodology to construct reusable test collections without judging every document in the collection. They achieve this by holding constant the test collection and query set, pooling the top X results (typically 100) from each system and manually judging each document in the pool as relevant or irrelevant. It has been shown that if X (the judgment depth) is sufficiently large, these collections are reusable, in that the relative effectiveness of runs from new systems over the same documents and queries can be evaluated simply by applying the existing judgments and assuming documents that are not judged are irrelevant [1]. Recent work examines how to scale evaluation to collections the size of the World Wide Web [2].

Foundations

Evaluation paradigms based on Cranfield (including TREC) make several common assumptions:

- Documents are either relevant or irrelevant (relevance is binary).
- Relevance of a given document is independent of any others (relevance is independent).
- The information need for which the query was derived does not change (static users).

Although these assumptions are obviously untrue in practice, they continue to be the basis for simplifying the evaluation problem. Various tracks at TREC have investigated search tasks where these assumptions are particularly broken. For example, navigational search, where users are looking for a particular destination as opposed to all of the information about a given topic was studied for several years [4].

The majority of published work in information retrieval leverages the TREC methodology and its reusable test collections. This is due not only to the scientific virtue of repeatable experiments, but also to the extraordinary effort required to evaluate the effectiveness of information retrieval systems. Judging more and more documents as new retrieval algorithms are evaluated is often prohibitive. However, the growth of very large document collections such as the Web has called into question the reusability of test collections developed in this manner. Rather, some results indicate that simply judging each of the top ranked results to a shallow depth for a larger number of queries may be more efficient for achieving reproducibility on such large collections [6]. Most recently, methods of incorporating automatic evaluations (such as click through evidence or known results from taxonomies) with manual ones have been developed to reach reproducible conclusions with less manual effort [5].

There are many specific metrics available for combined measurement of precision and recall, and for aggregating these measurements across queries. The F-measure is often used to combine point-wise precision and recall in tasks where only a single measurement of each is available. Evaluating ranked lists, however, provides an inherent threshold mechanism for taking many such measurements, each at a different depth into that list. Average precision combines these measurements at varying depths, and is aggregated across multiple queries by either the arithmetic mean (MAP – Mean Average Precision) or geometric one (GMAP – Geometric Mean Average Precision). Let AP_n represents the Average Precision value for a query from a set of n queries, then MAP and GMAP can be computed as follows:

$$MAP = \frac{1}{n} \sum_{n} AP_n$$

$$GMAP = \sqrt[n]{\prod_{n} AP_n}$$

R-precision takes the precision and recall measurements at depth R where R is the number of relevant documents for that query, which averages well across queries with varying numbers of relevant documents. The arithmetic mean of the R-precision values for an information retrieval system over a set of n queries is called the Average R-precision.

Other information retrieval tasks, such as navigational searches focusing on one correct answer, do not involve a recall component. These are often evaluated by the reciprocal rank of the correct result in the list. Specifically, the Reciprocal Rank value of a query is the reciprocal of the rank at which the first relevant result was retrieved for the query. When the Reciprocal Rank values are averaged across multiple queries, the measure is called the Mean Reciprocal Rank (MRR).

Key Applications

As in any field, improvements in information retrieval depend on the ability to evaluate. Researchers continue to leverage these techniques to refine retrieval algorithms. Search is also one of the most used features on the web, making evaluating its effectiveness critical.

Data Sets

Text REtrieval Conference test collections: http://trec.nist.gov.

URL to Code

TREC evaluation metric calculator: http://trec.nist.gov/trec_eval/.

Cross-references

- ▶ Average Precision
- ▶ Average Precision Histogram
- ▶ Average R-Precision
- ▶ GMAP (Geometric Mean Average Precision)
- ▶ MAP (Mean Average Precision)
- ▶ MRR (Mean Reciprocal Rank)
- ▶ Precision-Oriented Effectiveness Measures
- ▶ R-Precision
- ▶ Standard Effectiveness Measures

Recommended Reading

1. Buckley C. and Voorhees E.M. Evaluating evaluation measure stability. In Proc. 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2000, pp. 33–40.
2. Clarke C.L.A., Craswell N., and Soboroff I. Overview of the TREC 2004 Terabyte track. In Proc. 15th Text Retrieval Conference, 2006, NIST Special Publication.
3. Cleverdon C.W., Mills J., and Keen E.M. Factors determining the performance of indexing systems. Cranfield CERES: Aslib Cranfield Research Project, College of Aeronautics, Cranfield. vol. 1: Design, vol. 2: Results, 1996.
4. Hawking D. and Craswell N. Overview of the TREC 2001 web track. In Proc. 10th Text Retrieval Conference, 2001. NIST Special Publication.
5. Jensen E.C., Beitzel S.M., Chowdhury A., and Frieder O. On repeatable evaluation of search services in dynamic environments. ACM Trans. Inf. Syst., 26(1), 2007.
6. Sanderson M. and Zobel J. Information retrieval system evaluation: effort, sensitivity, and reliability. In Proc. 28th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 162–169.

EF-Games

▶ Ehrenfeucht-Fraïssé Games

egd

▶ Equality-Generating Dependencies

EHR

▶ Electronic Health Record

Ehrenfeucht-Fraïssé Games

Nicole Schweikardt
Johann Wolfgang Goethe-University Frankfurt am Main, Frankfurt, Germany

Synonyms

Ehrenfeucht games; EF-games

Definition

The Ehrenfeucht-Fraïssé game (EF-game, for short) is played by two players, usually called the *spoiler* and the *duplicator* (in the literature, the two players are sometimes also called *Samson* and *Delilah* or, simply, *player I* and *player II*). The board of the game consists of two structures $\mathcal{A}$ and $\mathcal{B}$ of the same vocabulary. The spoiler's intention is to show a difference between the two structures, while the duplicator tries to make them look alike.

The rules of the classical EF-game are as follows: The players play a certain number r of rounds. Each round i consists of two steps. First, the spoiler chooses either an element a_i in the universe of $\mathcal{A}$ or an element b_i in the universe of $\mathcal{B}$. Afterwards, the duplicator chooses an element in the other structure, i.e., she chooses an element b_i in the universe of $\mathcal{B}$ if the spoiler's move was in $\mathcal{A}$, respectively, an element a_i in the universe of $\mathcal{A}$ if the spoiler's move was in $\mathcal{B}$.

After r rounds, the game finishes with elements $a_1,\ldots,a_r$ chosen in $\mathcal{A}$ and $b_1,\ldots,b_r$ chosen in $\mathcal{B}$, and exactly one of the two players has won the game. Roughly speaking, the duplicator has won if and only if the structures $\mathcal{A}$ and $\mathcal{B}$, restricted to the elements chosen during the rounds of the game, are indistinguishable. To give a precise description of the winning condition let us assume, for simplicity, that the vocabulary of the structures $\mathcal{A}$ and $\mathcal{B}$ only contains relation symbols. Precisely, the duplicator has won the game if and only if the following two conditions are met: (i) for all $i,j \in \{1,...,r\}$, $a_i = a_j$ iff $b_i = b_j$, and (ii) for each arity k, each relation symbol R of arity k in the vocabulary, and all $i_1,...,i_k \in \{1,...,r\}$, the tuple $(a_{i_1},...,a_{i_k})$ belongs to the interpretation of R in the structure $\mathcal{A}$ if and only if the tuple $(b_{i_1},...,b_{i_k})$ belongs to the interpretation of R in the structure $\mathcal{B}$.

Since the game is finite, one of the two players must have a *winning strategy*, i.e., he or she can always win the game, no matter how the other player plays.

Key Points

EF-games are a tool for proving expressivity bounds for query languages. They were introduced by Ehrenfeucht [1] and Fraïssé [3]. The fundamental use of the game comes from the fact that it characterizes first-order logic as follows: The duplicator has a winning strategy in the r-round EF-game on two structures $\mathcal{A}$ and $\mathcal{B}$ of the same vocabulary if, and only if, $\mathcal{A}$ and $\mathcal{B}$ satisfy the same first-order sentences of quantifier rank at most r (recall that the quantifier rank of a first-order formula is the maximum nesting depth of quantifiers occurring in the formula). This is known as the *Ehrenfeucht-Fraïssé Theorem*, and it gives rise to the following methodology for proving inexpressibility results, i.e., for proving that certain Boolean queries cannot be expressed in first-oder logic: To show that a Boolean query Q is *not* definable in first-order logic, it suffices to find, for each positive integer r, two structures $\mathcal{A}_r$ and $\mathcal{B}_r$ such that (i) $\mathcal{A}_r$ satisfies query Q, (ii) $\mathcal{B}_r$ does not satisfy query Q, and (iii) the duplicator has a winning strategy in the r-round EF-game on $\mathcal{A}_r$ and $\mathcal{B}_r$.

Using this methodology, one can prove, for example, that none of the following queries is definable in first-order logic: “Does the given structure’s universe have even cardinality?”, “Is the given graph connected?”, “Is the given graph a tree?” (cf., e.g., the textbook [4]).

In fact, the described methodology is the major tool available for proving inexpressibility results when restricting attention to *finite* structures. Applying it, however, requires finding a winning strategy for the duplicator in the EF-game, and this often is a non-trivial task that involves complicated combinatorial arguments. Fortunately, techniques are known that simplify this task, among them a number of sufficient conditions (e.g., *Hanf-locality* and *Gaifman-locality*) that guarantee the existence of a winning strategy for the duplicator (see e.g., the survey [2] and the textbook [4]).

Variants of EF-games exist also for other logics than first-order logic, e.g., for finite variable logics and for monadic second-order logic (details can be found in the textbook [4]).

Cross-references

▶ Expressive Power of Query Languages
▶ FOL formulae syntax
▶ Locality
▶ Logical Structure

Recommended Reading

1. Ehrenfeucht A. An application of games to the completeness problem for formalized theories. Fundamenta Mathematicae, 49:129–141, 1961.
2. Fagin R. Easier ways to win logical games. In Descriptive Complexity and Finite Models, N. Immerman, P.G. Kolaitis (eds.). DIMACS Series in Discrete Mathematics and Theoretical Computer Science 31. American Mathematical Society, 1997, pp. 1–32.
3. Fraïssé R. Sur quelques classifications des systèmes de relations. Université d’Alger, Publications Scientifiques, SérieA(1):35–182, 1954.
4. Libkin L. Elements of Finite Model Theory. Springer, Berlin, 2004.

Ehrenfeucht Games

▶ Ehrenfeucht-Fraïssé Games

Electronic Commerce Transactions

▶ e-Commerce transactions

Electronic Data Capture

▶ Clinical Data Acquisition, Storage and Management

Electronic Dictionary

Robert A. Amsler
CSC, Falls Church, VA, USA

Synonyms

eDictionary; Machine-readable dictionary (MRD); Terminological database

Definition

An electronic dictionary contains lexicographic information that is stored and accessed via a computer. The term "electronic dictionary" may refer to the data alone (e.g., a machine-readable dictionary), but more typically refers to a software or software/hardware system that provides access to dictionary data. Although a dictionary may include encyclopedic entries, it is typically distinct from an electronic encyclopedia. Terminological databases are special-purpose dictionaries that are used primarily to distinguish domain-specific terminology and choose appropriate terms when translating technical documents. Thesauri are special-purpose word books organized by relationships between words. They may contain definitions, but most famously, Roget's Thesaurus (available in electronic form from Project Gutenberg) does not.

Historical Background

The first widely available electronic data from print dictionaries were produced in 1966–1968 by John Olney at System Development Corporation. At that time, compositor tapes were created by commercial printers solely for their use in printing books under contract with publishers. Printers encoded typesetting information in proprietary formats. Even dictionary publishers had no access to the electronic form of the data. Therefore, to create an early electronic dictionary, Olney supervised the transcription of the contents of the Merriam-Webster New Pocket Dictionary (MPD) and the Merriam-Webster Seventh Collegiate Dictionary (W7) into punch-card images, which he subsequently distributed under license as multi-reel magnetic tape data sets.

As publishers began to realize the value of retaining the content of their dictionaries in electronic form, they began requesting printers to provide a digital copy of the typesetting data along with the print copies of the dictionary. The version of the Collins English Dictionary distributed by the Oxford Text Archive illustrates non-standard print formatting codes as used by commercial print houses.

Subsequently, publishers embraced text encoding standards, such as SGML. They teamed up with computer scientists to convert many of their books, including dictionaries, to use such standard formatting, and requested printers to typeset their dictionaries from these standard formats. The largest project to convert a legacy print text dictionary into contemporary markup text data was carried out by the Oxford University Press, in conjunction with IBM and the University of Waterloo, to computerize the Oxford English Dictionary [7].

The advent of the personal computer and CD-ROM readers led to the direct sale of CD-ROM versions of print dictionaries with proprietary interfaces and encodings of the text content. The World Wide Web also provided publishers with the opportunity to sell subscription-based access to electronic monolingual and bilingual dictionaries.

In 1978–1980, Amsler [2] directed a project at the Linguistics Research Center of The University of Texas at Austin to produce the first taxonomically structured dictionary from the Olney 1968 MPD. The project employed graduate students to manually "disambiguate" the kernel terms in dictionary definitions, providing the basis for connecting the dictionary entries defined by those terms into a "tangled" hierarchy. Olney had proposed such an enterprise a decade earlier, but had run out of funding before it could be undertaken.

Subsequent to Amsler's work, Roy Byrd [3] at IBM began a comprehensive program of computational lexicology work to use the contents of the W7 for computational linguistics. Using a new parsing algorithm for dictionary definitions, Byrd et al. also created a taxonomically structured version of the W7, but due to copyright restrictions neither the Amsler nor the Byrd taxonomic dictionaries were ever redistributed.

In 1986, George Miller planned the creation of the WordNet electronic dictionary to test psychological theories of semantic memory [4]. WordNet Version 1.0, released in 1991, offered alphabetic and taxonomic access to English definitions. Miller avoided copyright issues by creating his own definition texts, making it possible to widely distribute WordNet for others to use in academic and commercial ventures. WordNet has spawned the creation of other special-purpose electronic dictionaries, including monolingual and bilingual WordNets for languages other than English. It remains the major free electronic dictionary available to the research community.

Foundations

Dictionaries are among the most complex documents, owing to the large number of distinct fields they contain. The structural elements found within electronic dictionaries are typically represented by XML tags or some other form of markup, such as described by the Text Encoding Initiative [6].

A typical entry in a conventional monolingual dictionary begins with a headword, usually divided into syllables, and optionally including a homograph number to distinguish the entry from others with the same headword. This is followed by the pronunciation(s), with high and low stress marks for its syllables, followed by any variant pronunciations. Next there are part-of-speech (POS) labels, other labels that reflect general usage (e.g., whether a word is formal or colloquial, whether its use is geographically restricted, and whether it is used primarily within one discipline), and inflected forms of the headword, if any. An etymology may follow, containing various language designations together with the word forms or affixes from which the headword is believed to have been derived. The etymology might also include dates and names of specific individuals when a headword's origin is reported to come from a specific person or event.

Definitions in a major dictionary are divided into distinct senses, sub-senses, sub-sub-senses, etc., down several levels. The order of the senses in a dictionary is typically either historical, with the earliest senses first, or prioritized from most common to least common senses. Each sense may include labels that document its use as being restricted to a particular region or to a particular discipline, and it may be illustrated by one or more example sentences. In the oldest dictionaries, examples were taken from notable authors, but this practice has given way to artificially constructed examples designed to be simplified illustrations of the meaning and usage of the sense. Definitions may also contain cross-references to specific senses of related headwords.

Following the definitions may be several additional entry features. These include so-called run-on entries, which are morphologically related extended forms of the headword that do not warrant their own entries elsewhere in the dictionary. These will typically have their own syllables, parts of speech and pronunciations listed. There may also be run-on entries for common phrases. Finally, an entry may also contain special treatments of synonyms and antonyms for the headword.

Key Applications

In addition to the use of dictionaries to look up individual words and their meanings (or translations), electronic dictionaries have been used in two major application areas. The first was to better understand the nature of lexical semantics through construction of taxonomies out of dictionary definition texts [2]. The second was to utilize electronic dictionaries for computational linguistic tasks ranging from morphological analysis and speech generation through word sense disambiguation. The former purpose led to WordNet and the growth of ontologies in artificial intelligence. The latter, while initially seen as promising, led to disappointment on the part of many computational linguists when they found print dictionaries inadequate to the tasks of reliably performing all computational linguistic processing tasks. However, the introduction of large electronic dictionaries changed the nature of computational linguistic system development to step away from "toy system" lexicons of a few hundred words to forever establish that systems had to be prepared for electronic dictionary-sized lexicons. Subsequently some computational linguists have embarked on creating specialized computer-based lexical resources [5]: electronic dictionaries that exist independently of the publishing community and contain advanced syntactic and semantic information.

Cross-references

▶ Document Databases

▶ Document Representations (Inclusive Native and Relational)

▶ SGML

▶ XML

Recommended Reading

1. Amsler R.A. Machine-readable dictionaries, Chapter 6. In Annual Review of Information Science and Technology (ARIST), M.E. Williams (ed.). Knowledge Industry Publications, vol. 19, 1984.
2. Amsler R.A. The Structure of the Merriam-Webster Pocket Dictionary. Ph.D. Dissertation, The University of Texas at Austin, Austin, TX, 1980.
3. Byrd R.J., Calzolari N., Chodorow M.S., Klavans J.L., Neff M.S., and Rizk O.A. Tools and methods for computational lexicology. Comput. Linguistics, 13(3–4):219–240, 1987.

4. Fellbaum C. WordNet: An Electronic Lexical Database. MIT Press, Cambridge, MA, 1998.
5. Sérasset G. Recent trends of electronic dictionary research and development in europe. Technical Memorandum, Electronic Dictionary Research (EDR). Tokyo, Japan, 1993, p. 93.
6. TEI Consortium. Dictionaries. TEI P5: Guidelines for Electronic Text Encoding and Interchange. Available online at: http://www.tei-c.org/release/doc/tei-p5-doc/html/DI.html (accessed on February 21, 2008).
7. Weiner E.S.C. Computerizing the Oxford English Dictionary. Scholarly Publishing 16(3):239–253, 1985.

Electronic Encyclopedia

Robert A. Amsler
CSC, Falls Church, VA, USA

Synonyms

eEncyclopedia

Definition

An electronic encyclopedia, like its print counterpart, provides extensive information covering general knowledge or specific disciplines through a vast collection of small articles, typically arranged alphabetically or indexed by title. Initially, electronic encyclopedias were restricted to ASCII encodings of the text portions of entries in print encyclopedias. Advances in the representation and presentation of universal character sets, images, sound, video, and interactive graphics allow today's electronic encyclopedias to present multimedia articles that are competitive with those found in conventional print encyclopedias.

Key Points

Electronic encyclopedias were initially computer-based delivery systems for their printed counterparts, accessed primarily via optical media or over computer networks. More recently, Wikipedia has demonstrated the effectiveness of collaborative co-authoring to create an electronic encyclopedia that has no print equivalent.

Encyclopedias need to be current: if not continually updated, they become obsolete with each change in geo-politics, scientific advance, or the death of a prominent individual. Furthermore, whereas print encyclopedias have elaborate indexes, taking up one or more volumes of their printed text, they cannot support compound searches. By supporting Boolean search, an electronic encyclopedia offers the ability to combine multiple concepts unanticipated by the encyclopedia editors. For example, it can support searches for entries that mention both (Winston) Churchill and (Albert) Einstein, and reveal that both won Nobel Prizes.

Entries in an electronic encyclopedia are typically multimedia articles, having text encoded with XML. Many articles also contain hyperlinks to other entries (or even to other potential entries that do not yet exist, as in Wikipedia).

Unlike commercial encyclopedias, Wikipedia is a free Web-based encyclopedia that allows readers to create and modify its content. For example, if a Wikipedia reader follows a link to a non-existent entry, the software invites the creation of the entry and will immediately accept any text offered as the body for that entry. Readers are also encouraged to update any entry deemed to be incorrect or out of date. This form of bootstrapping provides for the rapid creation of numerous entries and the editing of information in existing entries. Each editing change becomes a new entry, but can be "reverted" back to the prior entry by a future editor. Entries also have discussion pages on which authors and potential editors can comment on their content or explain their editing changes. Modification of an entry can trigger the sending of email to prior authors if they so choose, which gives authors some control over what happens to their text. In the event there are irreconcilable differences between multiple authors, the Wikipedia core group can lock down the article text and request authors to instead debate the changes in the comments pages.

Due to its large user base and the ability to monitor and revert entries to prior states, Wikipedia has flourished with millions of entries in multiple languages. However critics have questioned how it can achieve and maintain a high level of accuracy in the absence of authoritative controls over its contributors.

As well as including conventional encyclopedic information, Wikipedia has achieved broad coverage of popular culture, much of which has traditionally been regarded as trivia and unsuitable for inclusion in a print encyclopedia. Yet, for decades there have been specialty publications on trivia, crossword puzzle terminology, and special reference works for every conceivable category of information. Thus Wikipedia is more than an alternative to traditional encyclopedias: it is a potential replacement for all reference works.

The Wikipedia software is also freely available and has spawned the growth of numerous additional Wiki-based encyclopedias on the World Wide Web. Wiki software is used behind the scenes in many informational web sites. Modifications to the permissions allow Wiki software to restrict changes in content to select groups and thus permit its use in a variety of circumstances.

Cross-references

- ► Document Databases
- ► Electronic Dictionary
- ► Hypertexts
- ► Multimedia Databases

Recommended Reading

1. Giles J. Special report: internet encyclopaedias go head to head. Nature, 438:900–901, 2005.

Electronic Health Record

Amnon Shabo (Shvo)
IBM Research Lab-Haifa, Haifa, Israel

Synonyms

Virtual health record; Shared health record; Longitudinal health record; EHR

Definition

An electronic health record (EHR) is a standard-based machine-processable information entity consisting of health data pertaining to an individual. It is a result of an exhaustive aggregation of personal health data, which is longitudinal, cross-institutional and multimodal.

Historical Background

The term "medical record" usually refers to any recording/documentation of medical care or services given to a patient. The use of computerized patient records within healthcare enterprises is already a common and well-appreciated practice that has started about four decades ago [8]. It facilitates the documentation process required for medico-legal reasons and administrative procedures [10]. It also increases the availability of clinical data at the point of care which includes medical history, lab results, diagnostic imaging and so forth. The availability of such data at the point of care could help avoiding redundant tests thus saving time and cutting costs. It could also reduce medical errors and increase the overall quality and safety of care [9].

Nevertheless, medical records are typically partial and contain incomplete and inconsistent data due to fragmentation in the healthcare arena [5] where data is created in disparate systems of various healthcare providers, from single-physician offices to expert clinics, hospitals and large HMOs (health managed organizations) to name just a few (Note that the generic term "medical record" is used to refer to records also known by the acronyms Electronic Medical Record (EMR), Computerized Patient Record (CPR), and Electronic Patient Record (EPR)). In addition, the computerization of the traditional paper-based records and charts has been slow thus far and the user interfaces of many clinical information systems have been poorly designed causing time-pressed clinicians to avoid the use of it altogether.

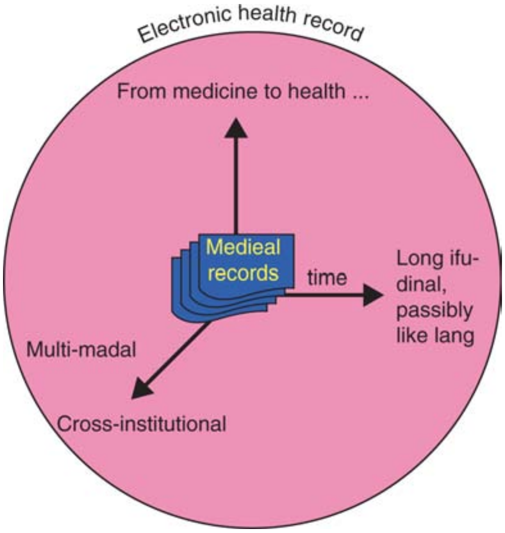
These factors and others made the benefit of computerized patient records quite limited. This situation gave rise to the emerging concept of an electronic health record (EHR). An ISO technical report [11] states that "Previous attempts to develop a definition for the Electronic Health Record have foundered due to the difficulty of encapsulating all of the many and varied facets of the EHR in a single comprehensive definition." The report lists several definitions of EHR which demonstrate the various aspects of EHRs versus medical records such as the focus on sharing data to support integrated care. In an attempt to consolidate the various definitions, the ISO report presents the following combined definition: "A repository of information regarding the health of a subject of care in computer processable form, stored and transmitted securely, and accessible by multiple authorized users. The EHR has a standardized information model which is independent of EHR systems. Its primary purpose is the support of continuing, efficient and quality integrated healthcare and it contains information which is retrospective, concurrent and prospective."

As a result of the above challenges, there have been several attempts in the past decade to realize the emerging EHR concept through national EHR efforts (e.g., in the UK [3]), regional health information exchange efforts (e.g., in the US [1]) as well as through eHealth initiatives (e.g., the Danish national eHealth portal [2]). Most of those attempts are still facing challenges due to several reasons ranging from technical issues in integrating data from dispersed sources, to the semantic differences between various formats used by healthcare providers and to socio-economic and medico-legal issues such as privacy, ownership of the EHR, access rights, governance and business models of organizations undertaking the sustainability of patient-centric EHRs.

Foundations

The term “health record” refers to an information entity which extends the traditional medical record entity described in the previous section. The extension occurs along three dimensions (Fig. 1):

1. Content: A health record contains any health-related information including self documentation, life style, environmental data, personal preferences, etc. and is not limited to medical data as in medical records.
2. Source: While a medical record is created by a healthcare provider in order to record the care or services given to a patient by that provider, a health record is a broader information entity in the sense that it aggregates recordings created by all healthcare providers regarding that patient as well as other sources of data pertaining to the health of the patient.
3. Time: A health record is a longitudinal information entity aggregating data created possibly throughout the lifetime of the individual, while a medical record is typically a recording of medical care or services given to a patient at a specific point in time.



Electronic Health Record. Figure 1. Extending the traditional enterprise medical record information entity to the all-encompassing EHR.

It is important to note that this encyclopedia entry describes the EHR information entity and does not deal with information systems that handle medical and health records. Most of these “EHR Systems” maintain medical records of a single enterprise and cannot cope with the EHR information entity as defined here.

The following sections describe each of the extension dimensions in more detail:

Content

Extending the content of medical records to any health-related data involves the integration of data types that have not been traditionally captured in medical records such as the self documentation of habits and events as experienced by the individual, whether instructed by a healthcare professional or self initiated. Other types of data include environmental and workplace related data that can provide context to clinical data and may support reasoning about possible diagnoses and treatments. A great challenge is the personal genetic data (e.g., personal DNA variations) that can assist in the realization of the personalized medicine vision [4]. While genetic data seems yet another type of lab results, there are many differences that make it distinct and rarely captured in common medical records. In particular, the notion of genomics refer to new tests and assays applied to multiple genes interacting through complex biological pathways where personal changes in the DNA sequence as well in the expression levels of coding DNA and resulting polypeptides have significant impact on the health of that individual. The challenge in this type of data is not only its complexity and rapid changing nature but also the association to phenotypic data, whether observed in the patient or scientifically known as possible interpretations of the genetic/genomic observations. The creation of genotype-phenotype associations relevant to an individual is perhaps the greatest challenge of the emerging health record concept. Only coherent and standard presentation can enable the safe and efficient use of this diverse data set to the benefit of the patient [13].

Source

Extending the typical single-enterprise source of medical records involves sharing data from multiple healthcare providers, expert clinics, community health services, clinical testing laboratories, diagnostic imaging facilities and any other institute that creates data about an individual. Sharing data for providing better care to a patient is challenging both semantically and technically wise.

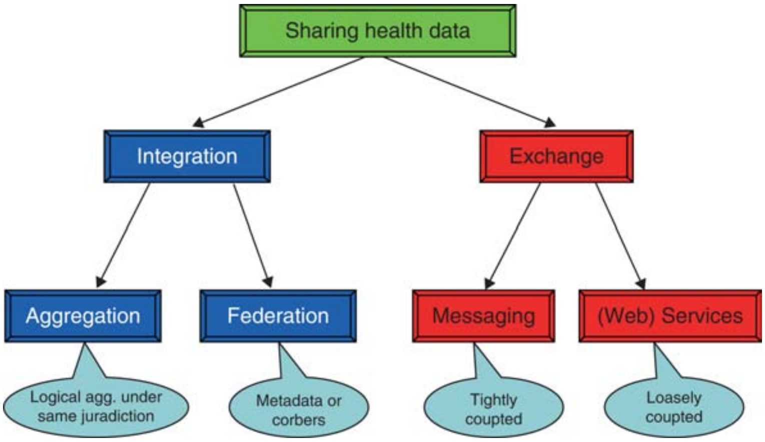
At the semantic level, the challenge is to reconcile the various formats by which the various medical records are created. The format of a medical record includes the structure (e.g., database schema, message definition, etc.) and values assigned to data items in a given structure (numeric value, medical codes, time spec, free text, etc.). To come up with a meaningful longitudinal view of the health record, it is essential to normalize the various proprietary representations of the source data; however, the lack of clear definition of the record structure could seriously hinder such effort.

At the technical level, the challenge is to find the most appropriate approach to fulfilling the requirements of information systems which handle electronic health records. The two main approaches of sharing data are exchange and integration. Data exchange is typically limited to a small number of systems exchanging information (e.g., point-to-point messaging) although the emerging service oriented architecture makes the service-based exchange more flexible.

Data integration is the preferred approach for the creation of EHRs. In principle, integration can be achieved by means of federation or aggregation (Fig. 2). In data federation (also known as "integration on the glass"), the source data is typically maintained by the enterprise where it was created and the EHR is compiled "on demand" by accessing the various sources, retrieving the data and presenting the compilation to the user. Curation and normalization are difficult to achieve at the point of care when interactive response times are expected. Efforts have been made to reach a consensus around federated formats suitable for clinical decision support and these formats are usually called Virtual Medical records (VMR). In contrast, data aggregation can be done in an on-going fashion bringing all source data into the EHR and reconciling each new item with the existing normalized record. At the point of care when the EHR is requested, an information system that aggregated the data can serve a coherent EHR in a timely fashion. The caveats of data aggregation have to do mainly with legal issues such as the release of health data by a healthcare provider and addressing privacy issues appropriately so that the EHR is highly secured and only authorized access is allowed based on proper consent of the patient. Note that the technical issues of data integartion have many governance implications and are influenced by national/regional policies as well as considerations at the socio-economic and medico-legal level [14].

Time

Medical records are typically episodic, describing visits, operations, hospitalizations, consultations and



Electronic Health Record. Figure 2. Major paradigms of data sharing through which an EHR could be created.

other services occurred in specific points in time. An EHR is a longitudinal aggregation of health data that embeds the temporal medical records along with other relevant data in a way that provides the opportunity to computationally create summaries of the data such as (i) non-redundant lists of essential data such as allergies, immunizations, diagnoses, medications etc. and (ii) useful summaries of the data such as a topical view (e.g., all data pertaining to a specific disease of the patient) or an event view (all data pertaining to a trauma the patient has gone through).

Unlike medical records, the EHR is ideally a single information entity per individual and its main value added is that it enables the creation of information layers atop of raw data (Fig. 3).

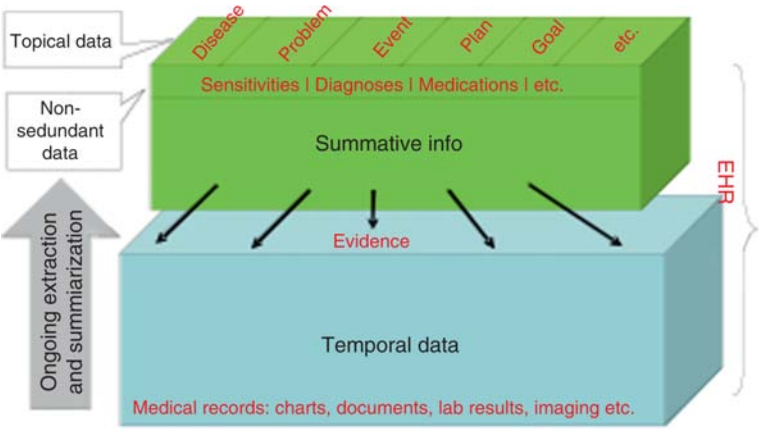
Key Applications

The main application of an electronic health record is an EHR system – a computerized information system that handles EHR information entities. EHR systems are expected to create, maintain and serve patient-centric EHRs to all authorized parties. Example of such systems are those built within the recent national and regional efforts across the globe aiming at sharing of health information in order to provide better care for the individual. Other emerging systems related to EHR are those handling the so-called Personal Health Records (PHR) – a type of EHR which is typically owned and controlled by the individual with emphases on the personal aspects in EHR such as self documentation, life style, preferences, and the like.

To overcome the diversity of formats used in the various sources integrated by an EHR system it is crucial to use standardized formats, preferably internationally-recognized EHR standards. Indeed, in the recent years, two significant standards have been developed and published by international standards development organizations. Health Level Seven coordinated an intensive international effort to publish a Functional Model Standard for EHR [7] and the European Standardization Body (CEN) has finalized an information model for EHR along with a new constraining formalism [12] to enable the standard representation of common health data sets within the EHR.

A somewhat related standard is the Clinical Document Architecture (CDA) that provides an electronic counterpart of the so common paper-based clinical documents such as operative notes, discharge summaries, referral letters and so forth [6]. These documents are a natural input to the longitudinal EHR as they typically include summary of patient's health conditions. All those standards complement each other and lay the ground for EHR Systems to finally be ready to achieve the ultimate goal of a patient-centric, standard-based, longitudinal and cross-institutional electronic health record.

A key application in the area of sharing and managing clinical documents as a step towards the EHR is an integration profile that has been developed by the IHE (Integrating the Healthcare Enterprise) and is coined XDS (Cross-enterprise Document Sharing [15]). The XDS integration profile is in fact a workflow specification that a clinical affinity domain (CAD) employs in order to share clinical documents created in disparate locations about the same patient. A CAD is established by enterprises that create and maintain


Topical data
Non-sedundant data
Ongoing extraction and summiarization
Disease
Problem
Event
Plan
Goal
etc.
Sensitivities | Diagnoses | Medications | etc.
Summative info
Evidence
Temporal data
Medical records: charts, documents, lab results, imaging etc.
EHR

Electronic Health Record. Figure 3. Layers of temporal and summative data constituting the EHR.

clinical documents and decided to share the documents through the affinity domain. Each CAD has a centralized registry where metadata about each document (e.g., the patient, author, authenticator, encounter, services, etc.) are registered by "document sources" so that queries can be posed by "document consumers." The retrieved data from the registry includes merely the metadata and once the document consumer selects documents of interest, the registry then approaches the relevant documents repositories residing in the respective premises of the CAD cooperating enterprises. Given proper authorization of the document consumer, the selected documents are then presented to the consumer. These documents could then be further processed to provide summary or analysis content per the user request. Such processes are complex to perform "on-the-fly" but the hope is that simple analyses could show the benefits of EHR to users who then might demand the creation of a coherent patient-centric EHR out of all available clinical documents and other types of medical records.

It is important to note that the XDS architecture doesn't allow for fully querying of the clinical documents stored in the enterprise repositories. The query can be done only against the metadata that was registered by the document sources. Queries by clinical data that is typically held in the body of the documents cannot be queried using the XDS integration profile. Nevertheless, the addition of an EHR data model on the top of an XDS architecture could then make the XDS repositories fully structured according to an EHR data model such as that of the CEN 13606 standard. Systems with such capabilities will also lend themselves to queries which are not patient-specific, e.g., search for data by disease, pathogens, populations and even genetic data.

Cross-references

► Clinical Document Architecture

Recommended Reading

1. Adler-Milstein J., McAfee A.P., Bates D.W., and Ashish K.J. The state of regional health information organizations: current activities and financing. Health Aff., 27(1):w60–w69, 2007.
2. Britze T.H. The Danish National e-health portal – increasing quality of treatment and patient life. Technol. Health Care, 13(5):366–367, 2005.
3. Coiera E.W. Lessons from the NHS National Programme for IT. Med. J. Aust., 186(1):3–4, 2007.
4. Davis R.L. and Khoury M.J. The journey to personalized medicine. Personalized Med., 2(1):1–4, 2005.
5. DePhillips H.A. Initiatives and barriers to adopting health information technology: a US perspective. Dis. Manag. Health Outcomes, 15(1):1–6, 2007.
6. Dolin R.H., Alschuler L., Boyer S., Beebe C., Behlen F.M., Biron P.V., and Shabo A. HL7 clinical document architecture, release 2. J. Am. Med. Inf. Assoc., 13(1):30–39, 2006.
7. Electronic Health Record System (EHR-S). Functional requirements standard. Available at: http://www.hl7.org/EHR/. Accessed on October 28, 2008.
8. Giere W. Electronic patient information – pioneers and much-More. A vision, lessons learned, and challenges. Methods Inf. Med., 43(5):543–552, 2004.
9. Haux R., Ammenwerth E., Herzog W., and Knaup P. Health care in the information society. A prognosis for the year 2013. Int. J. Med. Inf., 66(1–3):3–21, 2002.
10. Hollerbach A., Brandner R., Bess A., Schmucker R., and Bergh B. Electronically signed documents in health care – analysis and assessment of data formats and transformation. Methods Inf. Med., 44(4):520–527, 2005.
11. ISO/TC 215 technical report. Electronic health record definition, scope, and context. Second draft, August 2003.
12. Kalra D. Electronic Health Record Standards. In: IMIA Yearbook of Medical Informatics 2006, R. Haux, C. Kulikowski (eds.). Methods Inf. Med., 45:S136–S144, 2006.
13. Shabo A. The implications of electronic health records for personalized medicine. Personalized Med., 2(3):251–258, 2005.
14. Shabo A. A global socio-economic-medico-legal model for the sustainability of longitudinal electronic health records. Methods Inf. Med., 45(3 Pt 1):240–245, Methods Inf. Med., 45 (5 Pt 2):498–505, 2006.
15. XDS - cross-enterprise document sharing. Developed by the IHE (Integrating the Healthcare Enterprise) IT Infrastructure Committees. Available at: http://www.ihe.net/IT_infra/committees/index.cfm

Electronic Ink Indexing

Walid G. Aref
Purdue University, West Lafayette, IN, USA

Synonyms

Handwritten text; Online handwriting

Definition

With the proliferation of personal digital assistants (PDAs) and pen-based handheld devices, it is important that computers understand handwritten text (or electronic ink). In this pen-based environment, one can have handwritten file contents, handwritten file names, handwritten directory names, handwritten email messages, handwritten signatures, etc. With the large bodies of handwritten content, indexing techniques are essential in order to search for the relevant content. The fact that the data is handwritten makes the problem more difficult than in conventional situations. No two persons handwrite a word in exactly the same way. Even the same person cannot write a word in the same way twice. This makes the indexing and retrieval of handwritten data a hard problem.

Historical Background

Retrieval techniques for handwritten data have to be scalable so that they can handle the continually growing sizes of multimedia data, e.g., scanned documents in digital libraries and handwriting databases. An additional requirement for ink processing in a pen-based and/or a personal digital assistant environment is the need to support online retrieval and fast response time. There are two approaches to dealing with handwritten electronic data, namely, handwriting recognition and ink pattern matching.

Handwriting recognition is a procedure for converting pen strokes into strings of characters (or any other fixed character set). Once converted into characters, strings can be manipulated and searched in conventional ways. One problem with handwriting recognition is that it is prone to errors. Many critics point out that handwriting recognition does not meet the need of users. In fact, it is widely believed that this shortcoming is one of the main causes why the sales of pen-based tablets have fallen short of the expected figures. Moreover, even if handwriting recognition becomes highly accurate, it is clear that it provides a mapping from a highly expressive medium such as ink to a constrained medium, such as character strings. Handwriting recognition is not directly applicable to diagrams, drawings and many special symbols that the user can write.

Because of the drawbacks of handwriting recognition, the notions of *approximate ink matching* (AIM) and *pictograms* have emerged as an interesting alternative to handwriting recognition [11]. Pictograms are simply handwritten pictures, while AIM is the technique that evaluates how well two pictograms match. By using pattern matching techniques, AIM can take an input pictogram and evaluate how well it matches each one of the previously stored pictograms. AIM eliminates the problems of expressiveness, language and alphabet dependency and inaccuracy of handwriting recognition. The procedure simply focuses on finding a pictogram that resembles the input, without trying to "understand" or translate its meaning. AIM algorithms with high matching accuracy have been developed.

The problem with the AIM technique is that it could be computationally expensive. Without any other tool, AIM techniques are forced to sequentially search the entire pictogram repository. As the size of the pictogram repository grows, this process becomes painfully slow and impractical, especially that pattern matching techniques can be computationally expensive. This highlights the need for scalable techniques that provide fast response time for retrieval queries in handwritten databases of larger sizes.

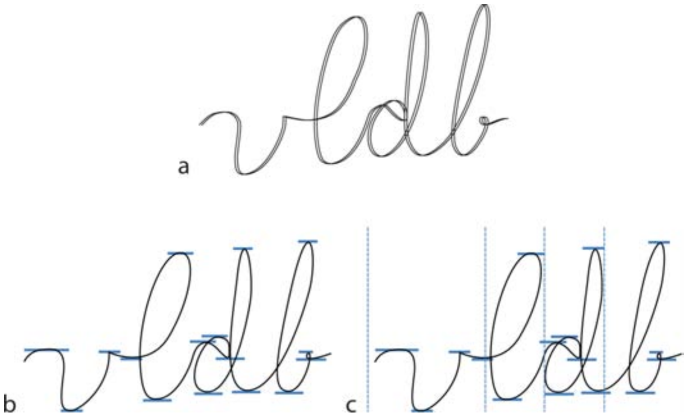
Foundations

Electronic handwritten ink in the form of a pictogram needs to be transformed into a more robust form before being passed to the database for further processing. This transformation needs to take place regardless of whether the pictrogram is to be inserted into the database, or used within a query to search the database. Electronic ink can be expressed as a sequence of time-stamped points in the plane:

$$s = (x_1, y_1, t_1), (x_2, y_2, t_2), \ldots., (X_k, Y_k, t_k),$$

where (x_i, y_i) represents the location of an electronic ink pixel at time t_i, $t_i \leq t_{\,i} + 1$, and k is the number of pixels in the pictogram.

Given a sequence S and a database of sequences S_j $(j = 1, \ldots, M)$ an important operation is to search the database for the sequences that are similar to S. Traditionally, databases use an alphanumeric representation of the data. This representation is ideally unique, precise and stable. Electronic ink data lack these qualities, making its matching a difficult problem. The data are often corrupted with noise. Even ideal ink does not provide an adequate basis for sequence identification, because a pictogram needs to be identified given slightly different variations in its shape. Different people cannot write the same word in the same way. Even for the same person, it is very difficult to generate *exactly* the same pictogram twice (it will almost always be the case that the stroke information varies each time the person writes the same word).

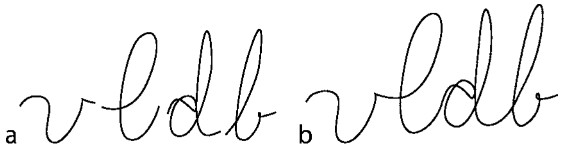


Electronic Ink Indexing. Figure 1. Example illustrating the segmentation of the pictogram in (a) into (b) strokes, and (c) alphabet symbols.

Electronic Ink Representation

In order to allow for approximate matching of handwritten electronic ink, each handwritten pictogram needs to be transformed into a more robust form. The raw electronic ink data that corresponds to points (or pixels) in the two-dimensional space is not robust as a slight shift in the input sequence may change the values of the x and y coordinates of the points representing a word. As a result, other features that are less sensitive to common deformations, e.g., translation, and rotation, are computed. Example features are the bounding box that contains all the points that form a pictogram, the average curvature, the number of points, etc.

These features can be computed at different levels of granularities. In this regard, handwritten electronic ink can be viewed at three levels of abstraction: the stroke level, the alphabet symbol level, and the pictogram level. Figure 1 illustrates a pictogram and its segmentation into pen strokes and handwritten symbols. Pictograms can be represented by any of the three granules in the figure, i.e., as one entity containing the entire pictogram (Fig. 1a), as a sequence of pen strokes (Fig. 1b), or as a sequence of alphabet symbols (Fig. 1c). For instance, in order to select the symbols as granules, a segmentation algorithm is needed to properly separate the symbols. For strokes, a simple segmentation algorithm picks local minimum (or maximum) points and uses them to segment the curve. Segmentation could be a difficult task for some types of pictograms, such as cursive handwritten words (Fig. 2b), or a simpler task as in handprinted words (see Fig. 2a). Some languages, like Japanese, lend themselves easily to symbol segmentation. In Japanese, Kanji symbols are already separated by blank spaces. The choice of granularity has an impact on the type of indexes to be built.



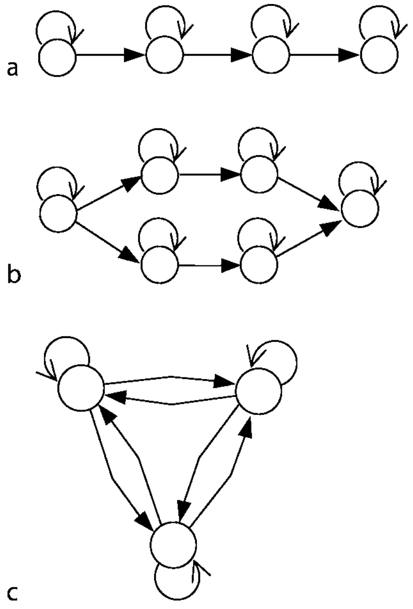
Electronic Ink Indexing. Figure 2. A handprinted word, (b) a cursive handwritten word.

The second issue is that, for matching purposes, it is better to talk about pictogram (symbols or strokes) classes instead of individual pictograms (symbols or strokes). A pictogram class is the set of pictograms that have the same semantics, according to the user. Of course, it would be impractical to store a pictogram class by storing the list of pictograms that belong to it. Alternatively, a *representative* of the class and a *distance metric* needs to be adopted. Inputs are matched against the representative (after the necessary preprocessing) and the distance metric ranks the matches. The representative is not a pictogram, but rather a model that captures the essential qualities of the pictogram class. In the next section, a sample model is presented that can be used to represent pictograms.

Hidden Markov Models (HMM)

HMMs are already used in the field of speech and handwriting recognition as a powerful tool for speech and handwritten document matching (see e.g., [11–13,15]). Each pictogram in the database can be modeled by an HMM. The HMM is constructed so that it accepts the specific pictogram with high probability (relative to the other pictograms in the database). In order to recognize a given input pictogram, each HMM in the database is executed and the one that generates the input sequence with the highest probability is selected. Each HMM in the underlying sequence database has to be tested, which results in a linear process, and the speed of execution becomes the primary difficulty.

An HMM is a doubly stochastic process, where there is a probability distribution that governs the transitions between states and an output probability distribution that identifies the distribution of output symbols for each state. An excellent coverage of HMMs can be found in [12,13]. For demonstration purposes, one type of HMM structures, termed *left-to-right HMM* [5] is illustrated in Fig. 3a. Left-to-right HMMs are useful for modeling temporal signals as in sound (see e.g., [13]) and cursive handwritten text (see e.g., [11]) because the underlying state sequence associated with the model has the property that, as time increases, the state index increases (or stays the same) – that is, the system states proceed from left to right.



Electronic Ink Indexing. Figure 3. (a) a left-to-right HMM model with four states, (b) a parallel path left-to-right HMM model with six states, and (c) the ergodic model with three states.

A left-to-right HMM can be constructed to model a handwritten word or an alphabet symbol. The HMM is constructed so that it accepts the word (or the symbol) with high probability (relative to the other words in the database). Training techniques may be applicable in the case where multiple instances of the word are available (e.g., a word can be handwritten multiple times, resulting in more than one sample). These samples can be used to train the HMM so that it can match similar words with higher probabilities (see e.g., [1,10]). The probability given by the HMM is the distance metric used for ranking purposes.

Given an HMM that models a word (or symbol), an input symbol can be run against the HMM to obtain as output a matching probability. Given a set of stored words (or symbols) with their associated HMMs, an input word (or symbol) can be searched by running each one of the corresponding HMMs against the input and choosing those with the best matching probability. In fact, the size of the answer set can be set as a parameter to choose the k best matches. To train HMMs and to use them to match a word or a symbol, a more robust form of ink representation needs to be used.

Representation Granularity

Features can be computed at the stroke level, pictogram level, symbol level, or through a collection of local features. For example, in the case of a stroke, a pictogram is segmented into a set of strokes, where each stroke is carefully described with a set of features. Thus, a pictogram can be represented by a collection of points in the feature space, one point per stroke. In contrast, at the pictogram level, a set of global features can be computed for the entire handwritten pictogram. Thus, each pictogram is described with a vector of feature values and is represented as one point in a multi-dimensional space.

Another simple, but efficient way of representing a pictogram is to pick some sample points from the pictogram (or symbol, or stroke) and compute some local features, ones that depend on a sample point and possibly the one (or two) surrounding points from each side. Depending on the application, some of these features may be more relevant than others, and hence not all of the features need be computed at a given point in the sequence. Common features are: direction, velocity, change in direction, change in velocity, accumulative angle (with respect to the initial point), accumulative length and angle of bounding box diagonal (also with respect to the initial point), and accumulative sequence length [14]. As an example of computing local features, the speed f_s at point p can be computed by: $f_s = sqrt(\Delta x_p^2 + \Delta y_p^2)/\Delta t_p$, *where* $\Delta x_p = x_{p+1} - x_p$, $\Delta y_p = y_p + 1 - y_p$, and $\Delta t_p = t_p + 1$ $t_p + 1 - t_p$. After computing the local features, the pictogram can be represented by a sequence of feature vectors (v_1, t_1), (v_2, t_2),... The dimensionality of v_i corresponds to the number of local features at each point.

Indexing Techniques

Sequential search does not scale well. The process becomes unacceptably slow as the number of pictograms stored in the database grows. Therefore, indexing techniques that help prune the choices are called for. However, indexing techniques for handwritten pictograms must exhibit two characteristics that make them different from traditional indexing techniques:

1. The structures must incorporate the underlying model that is chosen to represent ink. The model must play an active part of the index.
2. Due to the high variability of the ink data, the indices must provide approximate matching.

The following sections overview two sample indexing techniques that exhibit both of the characteristics stated above.

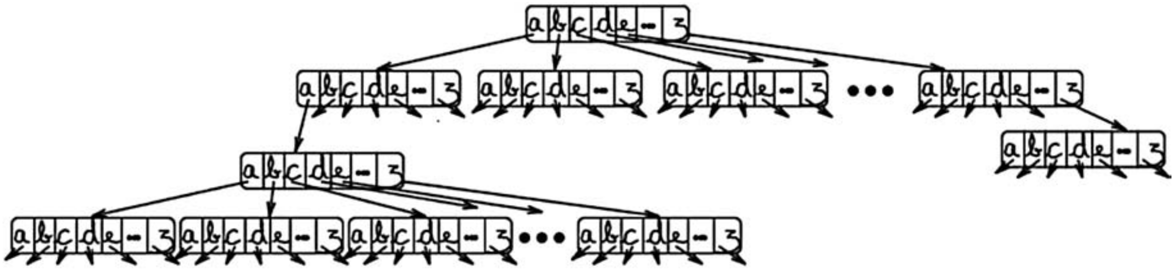
Alphabet-Level Indexing The handwritten trie [2] (Fig. 4) models ink at the alphabet symbol level and represents each of the symbol classes by using HMMs. The traditional trie data structure [4] is an *M*-ary tree whose nodes have *M* entries, and each entry corresponds to a digit or a character of the alphabet. Each node on level *l* of the trie represents the set of all keys that begin with a certain sequence of l characters; the node specifies an *M*-way branch, depending on the *(l + 1)*th character.

As pointed out in [6], the memory space of the trie structure can be significantly reduced (at the expense of running time) by using a linked list for each node, since most of the entries in the nodes tend to be empty. This results in what is termed the *forest trie.*

The handwritten trie uses a variant of the forest trie where nodes of the forest trie are packed into disk pages (a disk-based trie). Alphabet symbols of the trie are all handwritten. Since each symbol can be written differently each time, a separate HMM model per alphabet symbol is used in order to model that symbol. Given a handwritten query word, say w, a preprocessing step takes place to breakdown or segment *w* into a sequence of handwritten strokes or segments w_s. Various segmentation techniques can be used, e.g., [3]. Next, the handwritten trie is searched for an instance of w_s. Each of the HMMs at the root of the handwritten trie is executed against the first symbol in w_s and the search branches to the child node that has the best match. Mismatchings between w_s and items in handwritten trie are dealt with by backtracking and exploring different paths based on best matches between items of w_s and the corresponding items in the trie. An adapted version of the A* algorithm is used in order to prioritize which nodes of the trie to visit next.

In the case of the handwritten trie, matching is performed by using Hidden Markov Models (HMMs) [12]. The handwritten trie uses one type of HMMs, namely the *left-to-right HMM* [1] (see e.g., Fig. 3a), which is useful for modeling temporal signals as in sound (see e.g., [13]) and cursive handwritten text (see e.g., [11]).

The backtracking search mechanism in the handwritten trie takes care of 1-1 substitution errors, i.e.,



Electronic Ink Indexing. Figure 4. An example Handwritten Trie.

when a letter is mal-written, and hence is classified as another letter. However, there are other types of errors that are common in handwriting that needs to be dealt with. For example, if one of the symbols or letters in the query word is omitted (i.e., a *deletion* error), then an entire level of the handwritten trie is skipped and a node's grandchildren instead of its children are checked. Similarly, when an extraneous letter is introduced, e.g., due to an error in the cursive writing segmentation procedure (i.e., an *insertion* error), then the corresponding symbol of the input query word needs to be consumed without probing a node's children. To summarize, when deciding on how to match a symbol from the input query word against a node in the handwritten trie, all three types of error scenarios (1-1 substitution, deletion, and insertion errors) need to be investigated and prioritized using the A* search algorithm. Details of this procedure can be found in [4].

Stroke-Level Indexing

Indexing of handwritten text can be performed at the stroke level. A pictogram is segmented into a set of strokes. Each stroke is carefully described with a set of features, and, thus, can be stored as points in the feature space. Subsequently, any multi-dimensional access method, such as the R-tree, can be used. Similarity-based retrieval can be performed by executing a few range queries and then applying a voting algorithm to the output to select the handwritten words in the database that are most similar to the query [8]. The main advantage of this approach is that it is able to handle substring matching efficiently. Also, in contrast to the handwritten trie, stroke-level indexes are not order-dependent. For example, in scenarios where the order of handwriting a word may differ, the difference in the order of writing the strokes that compose Chinese characters does not adversely affect the search process. The reason is that regardless of the order, each stroke is mapped to a point into the multi-dimensional space and the neighborhood of each point is search for a best match, or best-k matches. Some rank aggregation needs to take place in order to merge all the returned best-k matches and return the best matched handwritten word in the database. Several techniques can be adopted to insure that the index is resilient to the kind of errors that result from the segmentation process, namely, stroke insertion/deletion and substitution errors [9].

Pictogram-Level Indexing

At the pictogram level, a set of global features can be computed for the entire handwritten pictogram. Thus, each pictogram is described with a vector of feature values and is represented as a point in a multi-dimensional space. In contrast to indexes that are based on stroke-level representations, no voting algorithms are needed as each pictogram is represented by only one point [9]. However, in order to increase the robustness and the matching rate of this approach, a multistage retrieval algorithm is adopted. At the first stage, the index is used to reduce the search space to a small set of candidate words that are similar to the handwritten query pictogram. The candidate set is then subjected to a pipeline of two sequential search algorithms that use a different set of extracted features and a distance ranking function to extract the most similar pictogram to the handwritten query pictogram.

Handwritten Ink as a Special Form of Time-Sequence Data

It is to be observed that handwritten text is one form of time-sequence data. As a result, all the indexing techniques applicable to handwritten text can also be applicable to the more general case of sequence data. For example, modeling sequence data using HMMs and similarity search and approximate matching techniques are also applicable to general sequence data and are not limited to only handwritten text. However, the converse is not true. The techniques used for indexing and similarity-based retrieval of time sequence data need to be adapted to make use of the special nature and features of electronic ink.

Cross-references

- ▶ Document Databases
- ▶ Feature Extraction for Content-Based Image Retrieval
- ▶ Feature Selection for Clustering
- ▶ High Dimensional Indexing
- ▶ Indexing and Similarity Search
- ▶ Indexing Metric Spaces
- ▶ Multimedia Data Indexing
- ▶ Multimedia Databases
- ▶ N-gram models
- ▶ Nearest Neighbor Query
- ▶ Object Recognition
- ▶ Similarity and Ranking Operations
- ▶ Text Indexing and Retrieval

▶ Text Indexing Techniques
▶ Tree-based Indexing
▶ Trie

Recommended Reading

1. Aref W.G., Vallabhaneni P., and Barbara D. On training hidden Markov models for recognizing handwritten text. In Proc. Int. Workshop on Frontiers of Handwriting Recognition, 1994.
2. Aref W.G., Barbará D., and Vallabhaneni P. The Handwritten Trie: Indexing Electronic Ink. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 151–162.
3. Aref W.G., Barbara D., Lopresti D.P., and Tomkins A. Ink as a first-class multimedia object. In Multimedia Database Systems: Issues and Research Directions, S. Jajodia and V.S. Subramanian (eds.). Springer, Berlin, 1995.
4. Aref W.G. and Barbará D. Supporting electronic ink databases. Inf. Syst., 24(4):303–326, 1999.
5. Bakis R. Continuous speech word recognition via centisecond acoustic states. In Proc. 91st Mts. of the Acoustical Society of America, 1976.
6. de la Briandais R. File searching using variable length keys. In Proc. Western Joint Computer Conference, 1959, pp. 295–298.
7. Fredkin E. Trie memory, Commun. ACM, 3(9):490–500, 1960.
8. Kamel I. Fast retrieval of cursive handwriting. In Proc. Int. Conf. on Information and Knowledge Management, 1996, pp. 91–98.
9. Kamel I. and Barbará D. Retrieving Electronic Ink by Content. In Proc. Int. workshop on Muldimedia Database Systems, 1996, pp. 54–61.
10. Levinson S.E., Rabiner L.R., and Sondhi M.M. An Introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. Bell Syst. Tech. J., 62(4):1035–1074, 1983.
11. Lopresti D.P. and Tomkins A. Approximate matching of hand-drawn pictograms. In Proc. 3rd Int. Workshop on Frontiers in Handwriting Recognition, 1993, pp. 102–111.
12. Rabiner. L.R. A tutorial on hidden markov models and selected applications in speech recognition. Proc. IEEE, 77(2):257–285, 1989.
13. Rabiner L.R. and Juang B.H. Fundamentals of Speech Recognition. Prentice Hall, Englewood Cliffs, NJ, 1993.
14. Rubine, D.H. The Automatic Recognition of Gestures, Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA, December, 1991. Also Technical Report Number CMU-CS-91-202.
15. Sin B.K. and Kim J.H. A statistical approach with HMMs for on-line cursive Hangul (Korean script) recognition. In Proc. 2nd Int. Conf. on Document Analysis and Recognition, 1993, pp. 147–150.

Electronic Libraries

▶ Digital Libraries

Electronic Newspapers

Robert B. Allen
Drexel University, Philadelphia, PA, USA

Definition

Newspapers collect, organize, and periodically disseminate information to a community in the form of news articles and other features. They are complex information resources which generally must also be economically viable. Electronic newspapers can provide many of the same types of services as traditional newspapers but they have many additional advantages for the readers. The electronic newspaper can be updated with the latest bulletins, tailored to the reader's interest, and take advantage of multimedia. However, increasingly, news and related services are being disaggregated and provided by separate suppliers so that in some cases, the electronic newspaper can sometimes be just a personalized web portal.

Historical Background

Newspapers arose in the seventeenth century in mercantile cities in Germany and the Netherlands. From the eighteenth to the twentieth centuries, newspapers were often the primary source of citizen's knowledge of the world beyond his immediate experience. Gradual changes in news communication technologies such as the telegraph, production technologies such as printing, delivery technologies such as trucking, increased the circulation and impact of newspapers throughout the nineteenth and early twentieth centuries. Early electronic media such as radio and television provided competition but newspapers still thrived. However, at the end of the twentieth century cable television began 24 h news coverage. The 24-h news cycle means that there is a greater need for newspapers to provide background and analysis and in many cases newspapers seem to be becoming more partisan. Interactive electronic media have amplified these trends and dramatically lowered the barriers to entry for news services. Among the notable effects has been the rise of blogging and citizen journalism. Although some electronic editions are very similar to their paper editions, in many cases it is better to consider electronic newspapers as a set of relatively loosely connected services rather than monolithic entities.

These technological upheavals have led to uncertainty as to how to maintain viable business models for newspaper publishing. As a result of digital convergence and shift to Web distribution most news content is supported by Web advertising. Web development and management also allows the multi-purposing of news content across delivery platforms. However, the Web has lowered the barriers to entry and allowed non-traditional news sources such as blogs and citizen journalism to flourish. Thus, the emerging news sources are broader but do support the depth of news gathering that has existed in the past.

While the focus of this review is on articles, electronic newspapers incorporate many other types of content such as classifieds, financial data, editorials, movie reviews, weather, sports, cartoons, and other advertising. Indeed, the distinction between news and entertainment is becoming blurred. Moreover, because of the ease of electronic publication, many sources of news are emerging such as blogs and citizen journalism. Indeed, these many be seen broadening newspapers' role in developing an informed citizenry. While dissemination of newspapers and journalism can broaden freedom of information there remains the possibility that information technology may ultimately facilitate censorship.

Foundations

An electronic newspaper is not just the presentation of a static newspaper page on a screen. Many electronic services contribute to its production, dissemination, and access and while news articles are the defining characteristic of newspapers, there are many other components which are coordinated to make up the content.

Managing Content

Metadata A comprehensive set of metadata descriptions for news articles would include date, rights, genre, and topic. An extensive set of XML-based descriptive metadata for news has been developed by the International Press and Telecommunications Council (IPTC). Beyond the general genre and topic descriptors for individual articles there are also content tags (e.g., for named entities) and structural descriptions for tabular material such as sports scores and financial data. The IPTC descriptive systems also include metadata for news images such as metadata describing the nature of the shot. For broader applications such as long term preservation and composite news objects, a multilayer metadata description framework such as the Multimedia Encoding and Transmission System (METS) can be used.

Digital Asset Management Systems Repositories extend standard Web servers by supporting content management services. The repositories used by electronic newspapers may be best characterized as digital asset management systems. Different types of digital asset management systems typically support different types of activities. Content developers such as journalists, editors, and layout designers would use a library-like service to access stored digital resources. This could provide access to stored content in a way similar to traditional photo "morgues" used by newspapers. That is they would share aspects of library services and production management services. A second major area of application for digital asset management systems in electronic newspapers could be to organize and push content to a web site. In other words, they would provide digital supply chain services.

Automated Text Processing of News Resources

To support complex activities such as composing news stories and allowing readers to interact with them, digital asset management systems need to support intermediate-level services such as summarization, search, rights management, and preservation.

Information Extraction and Text Data Mining While some news articles are now developed and disseminated with rich metadata and semantic annotation embedded, there is still a lot of untagged text. Extracting that text from unstructured news information would allow better text processing. The primary approach is based on the extraction of named-entities. There are also other extraction techniques such as template matching. Taken together, such information extraction techniques are examples of text data mining. Eventually, such text mining might extract facts in the news articles and allow them to be used in a question-answering system. In addition, it could be used to track opinions expressed on the Web in editorials, movie reviews, and blogs.

Event Detection and Tracking Of course, the main purpose of news articles is to report events so extracting, those events will be critical to any attempt to process the news articles. Topic Detection and Tracking (TDT) uses statistical approaches similar to those from information retrieval (IR). In this approach a topic is defined as a set of related stories. TDT is composed of several tasks such as new topic identification and topic tracking across a set of articles. The main technique is statistical clustering. However, this is a challenging task and the similarity is usually based on the entire description, not just the primary event. In order to improve accuracy, recent approaches have examined events at a finer-grained level. These approaches typically employ rule-based linguistic methods. For instance, they may attempt to determine the verb frames which are employed.

Commercial News Aggregators, News Summarization, and Searching Commercial news content aggregators collect articles from various news sources. In order to present them the aggregator generally clusters them. The most effective of these techniques perform the clustering on named-entities. The articles in each cluster are then ranked by attributes of their source, estimates of their scope and impact, and popularity and then presented to users.

Although they are not yet common in commercial news aggregation systems, summaries may be automatically synthesized for the clusters. Because this is done across news articles it is a type of multi-document summarization. Like most complex text, news text is composed of high-level rhetorical units. Identifying those units turns out to be helpful for generating summaries.

Individual articles and clusters of articles which are collected by news aggregators may also be indexed and searched. In addition to the usual parameters for weighting index terms such as term frequency and document frequency, terms in news articles can also be ranked on factors such as the frequencies and authority of the articles.

Additional Services and Issues

Preservation Services News has been called the "first draft of history" and, more generally, newspapers can be a significant cultural record. For instance, local libraries generally like to keep records of their local newspapers. This is an instance of keeping the record of the community. Unlike paper, electronic copies are ephemeral. However, with the decreasing cost of storage, it is now possible to collect and save almost all local news (with the possible exception of video). Challenges remain in collecting the materials. In addition to the preservation of electronic news materials, there is also now considerable interest in processing digitized historical newspapers. However, automatically extracting and processing the OCRd images has proven difficult because of variable quality.

Rights Management As with many types of commercially produced digital media, it remains unclear what is the best business model for the producers to recoup their expenses. Many publishers believe in maintaining tight control over their resources. This is certainly an issue for publishers with respect to the news aggregators. A related contentious rights issue concerns permission to store archival copies of the new items collected. With traditional newspapers, libraries could legally collect sets of old newspapers. At least in the U.S., libraries and archives are not currently allowed to made copies of digital objects for preservation.

Managing Multimedia News Content

Of course, a lot of news is on television and radio broadcasts and, increasingly, Web-based electronic newspapers include multimedia. The PRX and PBCore metadata standards have been developed for public broadcast radio and video. In addition, there are general multimedia metadata standard such as MPEG-7 and PRISM which can be applied productively to multimedia news objects.

As with data mining of text-based news articles, there could be great value in mining multimedia news objects. There has been considerable work on information extraction and summarization from television and radio news broadcasts. However, this is very challenging because it may involve combinations of different and complex processes including speech processing (unless closed captions are available), image recognition, and video segmentation. These processes tend to be error-prone leading to inaccuracies in named-entity extraction and this, in turn, affects the quality of services such as summarization and searching which use those named entities.

Presentation and Access Services

Personalized Interaction While traditional newspapers aim to provide news which is suitable for a wide range of readers, potentially news articles could be filtered to match a profile of personal preferences. These personalized articles could be combined into an entire personalized newspaper. Indeed, this sort of news interface could already be constructed from articles provided by Really Simple Syndication (RSS) services.

Recommendations Beyond personalization based on an individual's own profile, there are several ways news-related behaviors can be used to increase personalization. One technique is collaborative filtering in which suggestions are made from friends and coworkers. The ubiquity of email forwarding associated with electronic news papers shows the popularity of this type of sharing.

Automated techniques for generating recommendations are based on the pattern of news article accesses by individuals who have no particular connection to the reader. Such recommendations, which are also used for movies and music, are often calculated correlations between the user's choices and those of other individuals. When massive amounts of data are available, variations of singular-valued decomposition (SVD) have proven effective. Another way to simplify the data is by not using correlation but using simple counts such as those provided by "co-visitation." In any event, there are still substantial difficulties in generating effective recommendations such as the stability of people's interests ensuring unobtrusiveness and privacy ensuring in the collection of those interests, and the cold-start problem which occurs when there are no previous observations.

Cross-references

- ▶ Digital Archives and Preservation
- ▶ Digital Libraries
- ▶ Digital Rights Management
- ▶ Indexing the Web
- ▶ Information Extraction
- ▶ Information Retrieval
- ▶ Metadata
- ▶ Multimedia Presentation Databases
- ▶ Personalized Web Search
- ▶ Social Networks
- ▶ Text Mining
- ▶ XML

Recommended Reading

1. Hatzivassiloglou V., Gravano L., and Maganti A. An investigation of linguistic features and clustering algorithms for topical document clustering. In Proc. 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2000, pp. 224–231.
2. Linden G. People who read this article also read. . . . IEEE Spectr., 45(3):46–60, March 2008.
3. McKeown K., Barzilay R., Chen J., Elson D., Evans D., Klavans J., Nenkova A., Schiffman B., and Sigelman S. Tracking and summarizing news on a daily basis with Columbia's Newsblaster. In Proc. Human Language Technology Conf., 2002, pp. 280–285.

Eleven Point Precision-recall Curve

Ethan Zhang[1,2], Yi Zhang[1]

[1]University of California, Santa Cruz, CA, USA

[2]Yahoo! Inc., Santa Clara, CA, USA

Definition

The *11-point precision-recall curve* is a graph plotting the interpolated precision of an information retrieval (IR) system at 11 standard recall levels, that is, {0.0,0.1,0.2,...,1.0}. The method for interpolation is detailed below. The graph is widely used to evaluate IR systems that return ranked documents, which are common in modern search systems.

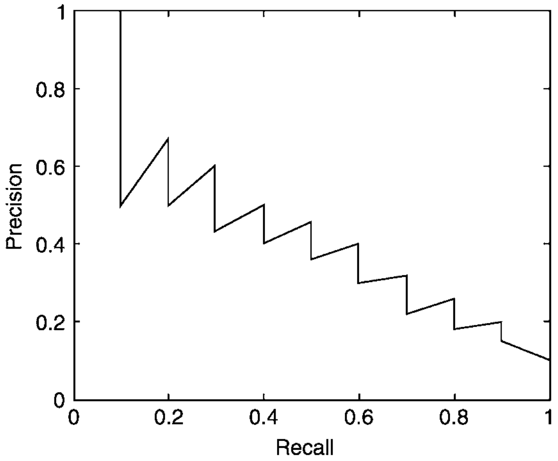
Key Points

In a precision-recall graph, precision is plotted as a function of recall. Assume a search system has retrieved n documents. Then for each $k \in [0,n]$, examine the set of top k retrieved documents and calculate the precision and recall for the set. These values are then plotted on a graph of precision versus recall. Figure 1 shows a typical precision-recall curve. It has a saw-toothed shape because the recall remains the same as k while the precision drops when the $(k + 1)$th retrieved document is not relevant.

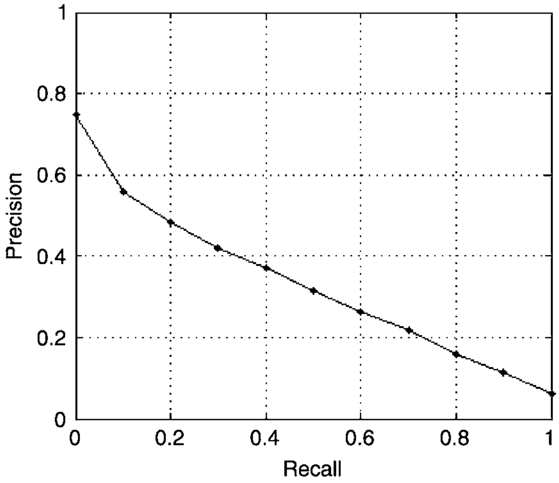
With the original precision-recall graph, averaging the curve over different queries is not straightforward. A standard approach is to plot interpolated precisions at 11 recall levels, namely, 0.0,0.1,...,1.0. The interpolated precision p_{interp} at recall level r is defined as the highest precision for any recall level $r' \geq r$:

$$p_{interp}(r) = \max_{r' \geq r} p(r')$$

Figure 2 shows the interpolated 11-point precision-recall curve for a good system in TREC 8.



Eleven Point Precision-recall Curve. Figure 1. Uninterpolated precision-recall curve.



Eleven Point Precision-recall Curve. Figure 2. Interpolated 11-point precision-recall curve.

Cross-references

- ► Precision
- ► Recall
- ► Standard Effectiveness Measures

Embedded Networked Sensing

- ► Sensor Networks

Emergent Semantics

Philippe Cudré-Mauroux
Massachusetts Institute of Technology, Cambridge,
MA, USA

Synonyms

Bottom-up semantics; Evolutionary semantics

Definition

Emergent semantics refers to a set of principles and techniques analyzing the evolution of decentralized semantic structures in large scale distributed information systems. Emergent semantics approaches model the semantics of a distributed system as an ensemble of relationships between syntactic structures. They consider both the representation of semantics and the discovery of the proper interpretation of symbols as the result of a self-organizing process performed by distributed agents exchanging symbols and having utilities dependent on the proper interpretation of the symbols. This is a complex systems perspective on the problem of dealing with semantics.

Historical Background

Syntax is classically considered as the study of the rules of symbol formation and manipulation [8]. Despite its wide usage in many contexts, the notion of *semantics* often lacks a precise definition. As a least common denominator, it can be characterized as a relationship or mapping established between a syntactic structure and some domain. The syntactic structure is a set of symbols that can be combined following syntactic rules. The possible domains that are related to the symbols may vary. In linguistics, such domains are typically considered as domains of conceptual interpretations. In mathematical logic, a semantic interpretation for a formal language is specified by defining mappings from the syntactic constructs of the language to an appropriate mathematical model. Denotational semantics applies this idea to programming languages. Natural language semantics classically concerns a triadic structure comprising a *symbol* (how some idea is expressed), an *idea* (what is abstracted from reality) and a *referent* (the particular object in reality) [9].

Emergent semantics expresses semantics through purely syntactic, recursive domains. The notions underlying emergent semantics are rooted in computational linguistics works relating semantics to the analysis of syntactic constructs. In his seminal work on syntactic semantics [10,11], William J. Rapaport defines semantic understanding as the process of understanding one domain in terms of another – antecedently understood – domain. This further raises the question of how the antecedently domain is itself understood. In the same vein, the antecedently understood domain has to be understood in terms of yet another domain, and so on and so forth recursively. To avoid to ground the recursion to a hypothetical base domain, Rapaport suggest the notions of *semantics as correspondence*, and lets the semantic interpretation function recursively map symbols to themselves or to other symbols. By considering the union of the syntactic and semantic domains, Rapaport regards semantics as syntax, i.e., turns semantics into the study of relations within a single domain of symbols and their interrelations. A dictionary is a simple example of a construct based on that paradigm, where the interpretations of symbols (i.e., words) are given by means of the same symbols, creating a closed correspondence continuum. Emergent semantics applies the conception of a closed correspondence continuum to the analysis of semantics in distributed information systems, by promoting recursive analyses of syntactic constructs – such as schemas, ontologies or mappings – in order to capture semantics.

Foundations

Beyond its implication in linguistics – where it is conjectured that human beings inevitably understand meaning in terms of syntactic domains – emergent semantics is considered as being mostly relevant to computer science. Programs, database schemas, models, or ontologies have no capacity (yet) to refer to reality. However, they have various mechanisms at their disposal for establishing relationships between internal symbols and external artifacts. In the settings where humans provide semantics, relationships among symbols – such as constraints in relational databases – are means to express semantics. In order to rectify some of the problems related to the implicit representation of semantics relying on human cognition, some have proposed the use of an explicit reference system for relating sets of symbols in a software system. Ontologies serve this purpose: an ontology vocabulary consists of formal, explicit but partial definitions of the intended meaning of a domain of discourse. In addition, formal constraints (e.g., on the mandatoriness or cardinality of relationships between concepts) are added to reduce the fuzziness of the informal definitions. Specific formal languages (such as OWL) allow to define complex notions and support inference capabilities. In that way, explicitly represented semantics of syntactic structures in an information system consist of relationships between those syntactic structures and some generally agreed-upon syntactic structure. Thus, the semantics is itself represented by a syntactic structure.

In a large scale distributed environment of information agents, such as in the Semantic Web or Peer-to-Peer systems, the aim is to have the agents interoperate irrespective of their initial vocabularies. To that aim, an agent has to map its vocabulary (carrying the meaning as initially defined in its *base* schema or ontology) to the vocabulary of other agents with which it wants to interoperate. Hence, a relationship between local and distant symbols is established. This relationship may be considered as another form of semantics, independent of the initial semantics of the symbols. Assuming that autonomous agents acquire vocabulary terms through relationships to other agents and that agents interact without human intervention, the original *human assigned* semantics would loose its relevance; from an agent's perspective, *new* semantics would then result from the relationships to its environment. This is a novel way of providing semantics to symbols of agents relative to the symbols of other agents with which they interact. Typically, this type of semantic representation is distributed such that no agent holds a complete representation of a generally agreed-upon semantics.

From a global perspective, considering a society of autonomous agents as one system, one can observe that the agents form a complex, self-referential and dynamic system. It is well-accepted that such systems often result in global states, which cannot be properly characterized at the level of local components. This phenomenon is frequently characterized by the notion of *self-organization*. Thus, emergent semantics is not only a local phenomenon, where agents obtain interpretations locally through adaptive interactions with other agents, but also a global phenomenon, where global semantics emerge from the society of agents and represent the common, current *semantic agreement* in the system. This view of semantics as the emergence of a distributed structure from a set of dynamic processes – or more specifically as some equilibrium state of such processes – is in-line with the generally accepted definitions of emergence and emergent structures in the complex systems literature [2]. In that respect, emergent semantics can be related to dynamic systems disciplines such as evolutionary game theory, semiotic dynamics [13], or graph evolution.

Key Applications

Multimedia Systems: Distributed multimedia applications were the first information systems to take advantage of emergent semantics principles in order to capture the semantics of shared objects. In this context, Santini et al. [12] argue that images do not have an intrinsic meaning, but that they are endowed with a meaning by placing them in the context of other images. From this observation, they propose a system where users are able to manipulate relations between images, and where the semantics of an image is emergent, in the sense that it is a product of the dual activities of users manipulating sets of images and of the database system. Along the same lines, Grosky et al. [5] focus on the role of context for giving meaning to a work of art. In their work, document semantics emerge through the analysis of users' browsing paths through a multimedia collection. They categorize the semantics of each item based on the collection of browsing paths.

Tagging Portals: Emergent semantics can be seen as a natural paradigm to analyze the distributed, user-driven process of giving semantics to items through the use of tags. Extreme Tagging [14] is a technique promoting the tagging of tags and the analysis of the relations between tags. Yeung et al. [15] discuss how shared documents acquire meaning through their associations with other elements. In particular, they demonstrate how different meanings of ambiguous tags can be discovered through the analysis of a tripartite graph involving users, tags, and resources. Herschel et al. [6] discuss query expansion techniques by aggregating users opinions as tags and discuss the role of pragmatics in collaboratively creating semantics.

Heterogeneous Information Systems: Emergent semantics has been suggested as a way to capture semantics in decentralized and heterogeneous information systems [4]. In contrast to mediated integration architectures, recent decentralized integration architectures – such as Peer Data Management Systems – do not require the definition of any global schema or ontology. Thus, the global semantics of such systems can only be captured by considering the collection of conceptualizations as defined by the local databases, along with their interrelations. Semantic Gossiping [1] analyzes transitive closures of schema mappings in that context in order to infer semantic agreement. Related techniques suggest the use of graph theory or probabilistic networks [3] in order to capture global semantics in similar environments. More broadly speaking, emergent semantics techniques are increasingly being seen as a way to minimize manual input and maintenance when dealing with complex and heterogeneous information systems [7].

Cross-references

▶ Data Integration
▶ Meta Data Management System
▶ Multimedia Databases
▶ Peer Data Management System
▶ Peer-to-Peer Data Integration
▶ Schema Mapping

Recommended Reading

1. Aberer K., Cudré-Mauroux P., and Hauswirth M. The Chatty Web: emergent semantics through gossiping. In Proc. 12th Int. World Wide Web Conference, 2003, pp. 197–206.
2. Bar-Yam Y. Dynamics of Complex Systems. Perseus Books Group, 1997.
3. Cudré-Mauroux P. Emergent Semantics: Rethinking Interoperability for Large-Scale Decentralized Information Systems. Ph.D. thesis, 3690, EPFL, Switzerland, 2006.
4. Cudre-Mauroux P. et al. Viewpoints on Emergent Semantics. J. Data Seman., VI:1–27, 2006.
5. Grosky W., Sreenath D., and Fotouhi F. Emergent Semantics and the Multimedia Semantic Web. ACM SIGMOD Rec., 31(4):54–58, 2002.
6. Herschel S., Heese R., and Bleiholder J. An Architecture for Emergent Semantics. In Proc. 15th Int. Conf. on Conceptual Modeling, 2006, pp. 425–434.
7. Howe B., Tanna K., Turner P., and Maier D. Emergent semantics: towards self-organizing scientific metadata. In Proc. Int. Conf. on Semantics of a Networked World, 2004, pp. 177–198.
8. Morris C. Foundations of the theory of signs. International Encyclopedia of Unified Science, 1(2), 1938.
9. Ogden C. and Richards I. The Meaning of Meaning: A Study of the Influence of Language upon Thought and of the Science of Symbolism. 10th edn., Routledge & Kegan Paul, London, 1923.
10. Rapaport W. Syntactic semantics: foundations of computational natural-language understanding. In Aspects of Artificial Intelligence, Kluwer Academic, Dordrecht, Holland, 1988, pp. 81–131.
11. Rapaport W. What Did You Mean by That? Misunderstanding, Negotiation, ans Syntactic Semantics. Minds and Mach., 13(3):397–427, 2003.

12. Santini S., Gupta A., and Jain R. Emergent Semantics through Interaction in Image Databases. IEEE Trans. Knowl. Data Eng., 13(3):337–351, 2001.
13. Steels L. Semiotic dynamics for embodied agents. IEEE Intelligent Systems, 21(3):32–38, 2006.
14. Tanasescu V. and Streibel O. Extreme tagging: emergent semantics through the tagging of tags. In Proc. Int. Workshop on Emergent Semantics and Ontology Evolution, 2007, pp. 84–94.
15. Yeung C., Gibbins N., and Shadbolt N. Understanding the semantics of ambiguous tags in folksonomies. In Proc. Int. Workshop on Emergent Semantics and Ontology Evolution, 2007, pp. 108–121.

Emerging Pattern Based Classification

Guozhu Dong[1], Jinyan Li[2]
[1]Wright State University, Dayton, OH, USA
[2]Nanyang Technological University, Singapore, Singapore

Synonyms

Contrast pattern based classification

Definition

The term “emerging-pattern based classification” refers to any classification algorithm that uses emerging patterns to directly build classifiers or to help build/improve other classifiers.

Key Points

The first approach to consider emerging-pattern based classification is CAEP (classification by aggregating emerging patterns) [2]. The main idea is to aggregate (sum) the discriminating power of many of the emerging patterns contained in a case to be classified. The discriminating power of an emerging pattern is often reflected in the support difference of the pattern in the opposing classes. For each class, the emerging patterns of that class contained in the case are aggregated to form a score; the class with the highest score is deemed to be the class of the case. Score normalization can be used to deal with data/battern imbalance between classes. This classification method can lead to high quality classifiers, comparable or better than other classifiers.

A second major direction in emerging-pattern based classification is the DeEPs method [2], which is an instance-based classification method using emerging patterns. After a case to be classified is given, DeEPs will compute the jumping emerging patterns contained in this case from the training data, and use the training data (that contain some of the discovered jumping emerging patterns) to compute the aggregated scores for the case and the classes. The DeEPs method avoids a deficiency of getting duplicate discriminating power contribution from near equivalent emerging patterns.

Recently, many other emerging-pattern based classification methods have been introduced (e.g., [1]), and several papers study the noise tolerance of emerging pattern based classifiers. More references can be found at www.cs.wright.edu/∼gdong/EPC.html.

Cross-references

- ► Applications of Emerging Patterns for Microarray Gene Expression Data Analysis
- ► Associative Classification
- ► Classification
- ► Classification by Association Rule Analysis
- ► Emerging Patterns

Recommended Reading

1. Alhammady H. and Ramamohanarao K. Using emerging patterns to construct weighted decision trees. IEEE Trans. Knowl. Data Eng., 18(7):865–876, 2006.
2. Dong G., Zhang X., Wong L., Li J. CAEP: classification by aggregating emerging patterns. Discov. Sci., 30–42, 1999.
3. Li J., Dong G., Ramamohanarao K., and Wong L. DeEPs: a new instance-based lazy discovery and classification system. Mach. Learn., 54(2):99–124, 2004.

Emerging Patterns

Guozhu Dong[1], Jinyan Li[2]
[1]Wright State University, Dayton, OH, USA
[2]Nanyang Technological University, Singapore, Singapore

Synonyms

Contrast pattern; Group difference

Definition

Roughly speaking, emerging patterns [3] are patterns whose support changes significantly from one dataset to another. The definition below captures significant change in terms of big growth rate.

Formally, let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two datasets, and let X be an itemset. For each integer $i \in \{1, 2\}$, let $supp_i(X)$ denote the support of X in $\mathcal{D}_i$. The *growth rate* of X's support from $\mathcal{D}_1$ to $\mathcal{D}_2$ is defined as

$$\text{GrowthRate}(X) = \begin{cases} 0 & \text{if } supp_1(\mathrm{X}) = 0 \text{ and} \\ & \quad supp_2(\mathrm{X}) = 0 \\ \infty & \text{if } supp_1(\mathrm{X}) = 0 \text{ and} \\ & \quad supp_2(\mathrm{X}) \neq 0 \\ \frac{supp_2(X)}{supp_1(X)} & \text{otherwise} \end{cases}$$

Observe that growth rates can also be defined in terms of counts instead of supports. One can also adjust the definition to avoid division by zero, by letting $\text{GrowthRate}(X) = \frac{supp_2(X)}{supp_1(X)+\epsilon}$ for some fixed $\epsilon > 0$.

Given a growth-rate threshold $\rho > 1$, an itemset X is called a *ρ-emerging pattern* (or simply *emerging pattern*, or *EP*) from $\mathcal{D}_1$ to $\mathcal{D}_2$ if $\text{GrowthRate}(X) \geq \rho$; the dataset $\mathcal{D}_1$ is called the *background* dataset, and $\mathcal{D}_2$ the *target* dataset, of X. Emerging patterns whose growth rate is ∞ are called *jumping emerging patterns* (JEPs).

As with frequent itemsets, the number of emerging patterns can be large. One common approach to overcome this problem is to focus on the minimal emerging patterns with respect to the set containment relation.

Historical Background

The concept of emerging patterns was motivated to capture emerging trends or change patterns over time, or significant differences/contrasts between datasets or data classes. The mining of emerging patterns can be viewed as a special case of comparative data mining, where one compares several datasets against each other in order to discover similarities and contrasts, etc. among the datasets. Emerging patterns were initially defined over relational or transactional data [3] and were recently generalized to sequences [6] and graphs [13], etc. In addition to emerging patterns, other patterns on change/difference, together with their mining, were also considered in many other studies [2,3,11,14,16].

Several algorithms have been proposed to mine emerging patterns, including the border-differential algorithm [3], a constraint-based bounding algorithm, a tree-based algorithm, a minimal hypergraph transversal based algorithm [1], a ZBDD based algorithm [12], and an incremental maintenance algorithm [9]. A theoretical analysis on the complexity of emerging pattern mining was reported in [16], and a study on the relationship between emerging patterns and rough set reducts was reported in [14]. Reference [10] proves that the space of jumping emerging patterns is convex and linked that space to version spaces. Reference [8] gives an efficient algorithm for mining minimal (with respect to set containment) emerging patterns, and linked the mining of emerging patterns to the mining of minimal generators and closed patterns. The mining of emerging patterns was also studied in several papers whose focus are on classification using emerging patterns (cf www.cs.wright.edu/∼gdong/EPC.html).

The study of emerging patterns led to two main directions of extensive and fruitful research, namely the use of emerging patterns in classification, and the application of emerging patterns for microarray gene expression data analysis.

Foundations

The border-differential algorithm [3] computes the border of certain sets of emerging patterns, by performing border differential. Borders [3] represent, in a concise manner, large convex collections of itemsets, including certain collections of emerging patterns. A border contains two boundary collections of itemsets, namely a *minB* and a *maxB*; it represents the convex set $\{z \mid \exists x \in minB, \exists y \in maxB \text{ such that } x \subseteq z \subseteq y\}$ of itemsets. For example, $<\{\{2\}, \{3, 4\}\}, \{\{2, 3, 4\}\}>$ is a border, where $\{\{2\}, \{3, 4\}\} = minB$ (containing two itemsets) and $\{\{2, 3, 4\}\} = maxB$ (containing one itemset), representing $\{\{2\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{2, 3, 4\}\}$. The main steps of the border-differential algorithm are some iterative expansion–minimization steps used to find the minimal itemsets which are contained in a given positive transaction but which are not contained in any of given negative transactions. The border-differential algorithm can directly compute the set of all jumping emerging patterns, and the set of all emerging patterns whose support in one dataset is lower than a threshold and whose support in the other dataset is higher than another threshold. Experiments show that the algorithm was fairly fast and can handle datasets of up to 75 attributes.

Reference [9] studies how to incrementally modify and maintain the concise border descriptions of the collection of all jumping emerging patterns when small changes to the original data occur. It introduces algorithms to handle four types of changes: insertion of new data, deletion of old data, addition of new attributes, and deletion of old attributes. Similar to the border-differential algorithm, the incremental algorithms obtain the new borders by manipulating the old borders. Experiments show that the incremental algorithms are much faster than the computing-from-scratch method.

Reference [1] shows that computing minimal jumping emerging patterns is equivalent to the well known problem of enumerating the minimal transversals of a hypergraph. It proposes a bottom up approach for computing minimal hypergraph transversals that is particularly suited to the types of hypergraphs arising in jumping emerging pattern mining. Earlier, the three authors of [1] also proposed a tree based algorithm to mine emerging patterns; the tree stores two (or more) counts for each node, one for each dataset; two item orderings, one based on frequency ratio and one based on frequency, are used; it was found that the algorithm was faster than earlier ones and that minimizing tree size may not lead to the shortest computation time.

Reference [12] introduces an efficient algorithm for mining emerging patterns (and generalizations). The algorithm is based on the so-called zero suppressed binary decision diagrams (ZBDDs). Binary decision diagrams (BDDs) are directed acyclic graphs which are efficient representations of boolean formulae. A ZBDD is a special type of BDD, originally introduced for set-manipulation in combinatorial problems. In [12], ZBDD is used to compress sparse high dimensional itemsets/data, and to allow the reuse of past computations. This algorithm can be hundreds of times faster than the tree-based algorithm, and can successfully mine emerging patterns from data with several hundreds (or even thousands) of attributes.

Key Applications

Emerging patterns are useful for capturing multi-dimensional contrasts between datasets/classes. They have been extensively used in classification, and for microarray gene expression data analysis (especially for cancers). Emerging patterns can also be used (i) to discover emerging trends in temporal databases by comparing datasets collected from two time intervals, (ii) to identify rare events, and (iii) to detect network intrusion. References on these can be found at www.cs.wright.edu/~gdong/EPC.html.

Cross-references

- ▶ Applications of Emerging Patterns for Microarray Gene Expression Data Analysis
- ▶ Emerging Pattern Based Classification

Recommended Reading

1. Bailey J., Manoukian T., and Ramamohanarao K. A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 485–488.
2. Bay S.D. and Pazzani M.J. Detecting change in categorical data: mining contrast sets. In Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 302–306.
3. Dong G., Li J. Efficient mining of emerging patterns: discovering trends and differences. In Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 43–52. Journal version [5].
4. Dong G., Han J., Lam J.M.W., Pei J., Wang K., and Zou W. Mining constrained gradients in large databases. IEEE Trans. Knowl. Data Eng., 16(8):922–938, 2004.
5. Dong G. Li J. Mining border descriptions of emerging patterns from dataset pairs. Knowl. Inf. Syst. 8(2):178–202, 2005.
6. Ji X., Bailey J., and Dong G. Mining minimal distinguishing subsequence patterns with gap constraints. In Proc. 2005 IEEE Int. Conf. on Data Mining, 2005, pp. 194–201. Journal version [7].
7. Ji X., Bailey J., and Dong G. Mining Distinguishing Subsequences Patterns with Gap Constraints. Knowl. Inf. Syst. 11 (3):259–289, 2007.
8. Li J., Liu G., and Wong L. Mining statistically important equivalence classes and δ-discriminative emerging patterns. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007, pp. 430–439.
9. Li J., Manoukian T., Dong G., and Ramamohanarao K. Incremental maintenance on the border of the space of emerging patterns. Data Min. Knowl. Discov. 9(1):89–116, 2004.
10. Li J., Ramamohanarao K., and Dong G. The space of jumping emerging patterns and its incremental maintenance algorithms. In Proc. 17th Int. Conf. on Machine Learning: 2000, pp. 551–558.
11. Liu B., Hsu W., and Ma Y. Discovering the set of fundamental rule changes. In Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2001, pp. 335–340.
12. Loekito E. and Bailey J. Fast Mining of High Dimensional Expressive Contrast Patterns Using Zero-Suppressed Binary Decision Diagrams. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006, pp. 307–316.
13. Ting M.H.T. and Bailey J. Mining minimal contrast subgraph patterns, In Proc. SIAM International Conference on Data Mining, 2006, pp. 638–642.
14. Terlecki P. and Walczak K. On the relation between rough set reducts and jumping emerging patterns. Inf. Sci., 177(1):74–83, 2007.
15. Vreeken J., van Leeuwen M., and Siebes A. Characterising the difference. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007, pp. 765–774.

16. Wang L., Zhao H., Dong G., and Li J. On the complexity of finding emerging patterns. Theor. Comput. Sci. 335(1):15–27, 2005.

17. Webb G.I., Butler S.M., and Newlands D.A. On detecting differences between groups. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 256–265.

Encryption

► Data Encryption

End User

► Actors/Agents/Roles

Ensemble

Zhi-Hua Zhou
Nanjing University, Nanjing, China

Synonyms

Committee-based learning; Multiple classifier system; Classifier combination

Definition

Ensemble is a machine learning paradigm where multiple learners are trained to solve the same problem. In contrast to ordinary machine learning approaches which try to learn *one* hypothesis from training data, ensemble methods try to construct a *set* of hypotheses and combine them.

Historical Background

It is difficult to trace the starting point of the history of ensemble methods since the basic idea of deploying multiple models has been in use for a long time. However, it is clear that the hot wave of research on ensemble methods since the 1990s owes much to two works. The first is an applied research conducted by Hansen and Salamon at the end of 1980s [5], where they found that predictions made by the combination of a set of neural networks are often more accurate than predictions made by the best single neural network. The second is a theoretical research conducted in 1989, where Schapire proved that *weak learners* which are slightly better than random guess can be boosted to *strong learners* that are able to make very accurate predictions, and the proof resulted in Boosting, one of the most influential ensemble methods [10].

Foundations

Terminologies

Learners composing an ensemble are usually called *base learners*. Many ensemble methods are able to boost *weak learners* which are slightly better than random guess to *strong learners* which can make very accurate predictions. So, "base learners" are also referred as "weak learners". However, it is noteworthy that although most theoretical analyses work on weak learners, base learners used in practice are not necessarily weak since using not-so-weak base learners often results in better performance.

Base learners are usually generated from training data by a *base learning algorithm* which can be decision tree, neural network or other kinds of machine learning algorithms. Most ensemble methods use a single base learning algorithm to produce *homogeneous* base learners, but there are also some methods which use multiple learning algorithms to produce *heterogeneous* learners. In the latter case there is no single base learning algorithm and thus, some people prefer calling the learners *individual learners* or *component learners* to "base learners", while the names "individual learners" and "component learners" can also be used for homogeneous base learners.

Methods

Typically, an ensemble is constructed in two steps. In the first step, a number of base learners are produced. Here the base learners can be generated in a *parallel* style, or in a *sequential* style where the generation of a base learner has influence on the generation of subsequent learners. In the second step, the base learners are combined to use. The most popular combination scheme for classification is *majority voting*, while the most popular combination scheme for regression is *weighted averaging*. The employment of different base learner generation processes and/or different combination schemes leads to different ensemble methods.

The following three paragraphs briefly describe the working routines of three representative ensemble methods, *Boosting* [10], *Bagging* [2] and *Stacking* [13], respectively. Here, binary classification is considered for simplicity. That is, let $\mathcal{X}$ and $\mathcal{Y}$ denote the instance space and the set of class labels, respectively, assuming $\mathcal{Y} = \{-1, +1\}$. A training set $\mathcal{D} = \{(x_1, y_1), (x_2, y_2),\ldots,(x_m, y_m)\}$ is given, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ ($i = 1,\ldots,m$).

For Boosting, let us consider the most famous algorithm, AdaBoost, as an example. In the first step, a number of base learners are produced by emphasizing each learner on the training examples that are wrongly predicted by preceding learners. Here, a weight distribution is maintained over the training examples, which is initialized by equal weights. In the tth learning round, a base learner $h_t : \mathcal{X} \to \mathcal{Y}$ is generated from the training set by using the current weight distribution, D_t. Then, h_t is evaluated on the training set and let ε_t denote the error. The weight distribution for the next learning round, D_{t+1}, is generated in the way that for the training example (x_i, y_i), $D_{t+1}(i) = \frac{D_t(i)}{Z_t} exp\left(\frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) y_i h_t(x_i)\right)$ where Z_t is a normalization factor which enables D_{t+1} to be a distribution. In the second step, the base learners h_t's ($t = 1,\ldots,T$) are combined by $H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.

For Bagging, in the first step a number of base learners are trained from *bootstrap samples*. A bootstrap sample is obtained by subsampling the training set with replacement, where the size of a sample is as the same as that of the training set. Thus, for a bootstrap sample, some training examples may appear but some may not, where the probability that an example appears at least once is about 0.632. On each sample a base learner $h_t : \mathcal{X} \to \mathcal{Y}$ is produced by calling a base learning algorithm. In the second step, Bagging combines the learners h_t's ($t = 1,\ldots,T$) by majority voting, i.e., $H(x) = argmax_{y \in \mathcal{Y}} \sum_{t=1}^{T} 1(y = h_t(x))$ where the value of $1(a)$ is 1 if a is *true* and 0 otherwise.

For a typical implementation of Stacking, in the first step, a number of individual learners, $h_t : \mathcal{X} \to \mathcal{Y}$ ($t = 1,\ldots,T$), are generated from the training set by employing different learning algorithms. In the second step, the individual learners are combined by using another learner. Here, for the training example (x_i, y_i), a corresponding example (z_i, y_i) is produced, where $z_i = (h_1\ (x_i),\ldots,h_T\ (x_i))$. Then, from $\{(z_i, y_i)\}$ ($i = 1,\ldots,m$) a learner $F : h_t^T \to \mathcal{Y}$ is generated by calling a learning algorithm, which is used to combine the individual learners by $H(x) = F\ (h_1\ (x),\ldots,h_T\ (x))$.

The above methods have many variants. For example, *Random Forests* [3], which has been deemed as one of the most powerful ensemble methods, is a variant of Bagging.

There are many other established ensemble methods, e.g., *Random Subspace* [6]. This method generates a number of base learners from different *subspaces* of the training set in the first step, and then combines these base learners via majority voting in the second step. Here, a subspace is actually a subset of the original attribute set.

It is worth mentioning that in addition to classification and regression, ensemble methods have also been designed for clustering [11] and other kinds of machine learning tasks.

It was thought that using more base learners will lead to a better performance, yet Zhou et al. [15] proved the "many could be better than all" theorem which indicates that this may not be the fact. It was shown that after generating a set of base learners, selecting some base learners instead of using all of them to compose an ensemble is a better choice. Such ensembles are called *selective ensembles*.

Note that usually the computational cost for building an ensemble comprising T base learners is roughly T times the cost of training a single learner. So, from the view of computational complexity, training an ensemble is almost as efficient as training a single learner.

Why Useful?

The generalization ability of an ensemble is usually much stronger than that of a single learner, which makes ensemble methods very attractive. An important question is, why can the ensembles be superior to single learners? For this, Dietterich [4] gave three reasons by viewing the nature of machine learning as searching a hypothesis space for the most accurate hypothesis. The first reason is that, the training data might not provide sufficient information for choosing a single best learner. For example, there may be many learners performing equally well on the training set. Thus, combining these learners may be a better choice. The second reason is that, the search processes of the learning algorithms might be imperfect. For example, even if there exists a unique best hypothesis, it might be difficult to achieve since running the algorithms results in sub-optimal hypotheses. Thus, ensembles can compensate for such imperfect search processes. The third reason is that, the hypothesis space being searched might not contain the true target function, while ensembles can give some good approximation. For example, it is well-known that the classification boundaries of decision trees are linear segments parallel to coordinate axes. If the target classification boundary is a smooth diagonal line, using a single decision tree cannot lead to a good result but a good approximation can be achieved by combining a set of decision trees. Although these intuitive explanations are reasonable, they lack rigorous theoretical analyses.

The *bias-variance decomposition* is often used in studying the performance of ensemble methods [1,15]. It is known that Bagging can significantly reduce the variance, and therefore it is better to be applied to learners suffered from large variance, e.g., unstable learners such as decision trees or neural networks. Boosting can significantly reduce the bias in addition to reducing the variance, and therefore, on weak learners such as decision stumps, Boosting is usually more effective.

There are many theoretical studies on famous ensemble methods such as Boosting and Bagging, yet it is far from a clear understanding of the underlying mechanism of these methods. For example, empirical observations show that Boosting often does *not* overfit even after a large number of rounds, and sometimes it is even able to reduce the generalization error after the training error has already reached zero. Although many people have studied this phenomenon, theoretical explanations are still in arguing.

Accuracy and Diversity

Generally, in order to construct a good ensemble, the base learners should be as more accurate as possible, and as more diverse as possible. This has been formally shown by Krogh and Vedelsby [7], and emphasized by many other people.

The definition of *accuracy* is clear, and there are many effective processes for estimating the accuracy of learners, such as cross-validation, hold-out test, etc. However, there is no rigorous definition on what is intuitively perceived as *diversity*. Although a number of diversity measures have been designed, Kuncheva and Whitaker [8] revealed that the usefulness of existing diversity measures in building ensembles is suspectable.

In practice, the diversity of the base learners can be introduced from different channels, such as subsampling the training examples, manipulating the attributes, manipulating the outputs, injecting randomness into learning algorithms, or even using multiple mechanisms simultaneously.

Key Applications

Ensemble methods have already been used in diverse applications such as optical character recognition, text categorization, face recognition, medical diagnosis, gene expression analysis, etc. Actually, ensemble methods can be used wherever machine learning techniques can be used.

Future Directions

A serious deficiency of ensemble methods is the lack of comprehensibility, i.e., the knowledge learned by ensembles is not understandable to the user. Improving the comprehensibility of ensembles [14] is an important yet largely understudied direction.

Diversity plays an important role in ensembles, yet currently no diversity measures is satisfying [8]. Exploring the relation between the performance of ensembles and the properties of base learners to design useful diversity measures is an interesting direction, which may lead to the development of more powerful ensemble methods.

Although there are much theoretical analyses on ensemble methods, many underlying mechanisms of successful ensemble methods are not clear. So, more theoretical studies are needed. Another ambitious attempt is to establish a general theoretical framework for ensemble methods.

Experimental Results

Empirical studies on popular ensemble methods have been reported in many papers, such as [1,9,12].

Data Sets

A large collection of datasets commonly used for experiments can be found at http://www.ics.uci.edu/~mlearn/MLRepository.html

Url To Code

The code of *Random Forests* can be found at http://www.stat.berkeley.edu/~breiman/RandomForests/

Cross-references

▶ Bagging
▶ Boosting
▶ Decision Tree

▶ Neural Networks
▶ Support Vector Machine

Recommended Reading

1. Bauer E. and Kohavi R. An empirical comparison of voting classification algorithms: bagging, Boosting, and variants. Mach. Learn., 36(1–2):105–139, 1999.
2. Breiman L. Bagging predictors. Mach. Learn., 24(2):123–140, 1996.
3. Breiman L. Random forests. Mach. Learn., 45(1):5–32, 2001.
4. Dietterich T.G. Machine learning research: Four current directions. AI Magn., 18(4):97–136, 1997.
5. Hansen L.K. and Salamon P. Neural network ensembles. IEEE Trans. Pattern Anal. Mach. Intell., 12(10):993–1001, 1990.
6. Ho T.K. The random subspace method for constructing decision forests. IEEE Trans. Pattern Anal. Mach. Intell., 20(8):832–844, 1998.
7. Krogh A. and Neural network ensembles, cross validation, and active learning. In Advances in Neural Information Processing Systems 7, G. Tesauro, D.S. Touretzky, and T.K. Leen (eds.). MIT Press, Cambridge, MA, 1995, pp. 231–238.
8. Kuncheva L.I. and Whitaker C.J. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Mach. Learn., 51(2):181–207, 2003.
9. Opitz D. and Maclin R. Popular ensemble methods: An empirical study. J. Artif. Intell. Res., 11:169–198, 1999.
10. Schapire R.E. The Boosting approach to machine learning: An overview. In Nonlinear Estimation and Classification, D.D. Denison, M.H. Hansen, C. Holmes, B. Mallick, and B. Yu (eds.). Springer, Berlin, 2003.
11. Strehl A. and Ghosh J. Cluster ensembles – a knowledge reuse framework for combining multiple partitionings. J. Mach. Learn. Res., 3:583–617, 2002.
12. Ting K.M. and Witten I.H. Issues in stacked generalization. J. Artif. Intell. Res., 10:271–289, 1999.
13. Wolpert D.H. Stacked generalization. Neural Netw., 5(2): 241–260, 1992.
14. Zhou Z.-H., Jiang Y., and Chen S.-F. Extracting symbolic rules from trained neural network ensembles. AI Commun., 16(1):3–15, 2003.
15. Zhou Z.-H., Wu J., and Tang W. Ensembling neural networks: Many could be better than all. Artif. Intell., 137(1–2):239–263, 2002.

Enterprise Application Integration

Janette Wong
IBM Canada Ltd, ON, Canada

Synonyms

EAI; Application-to-application integration; Application-centric interfacing

Definition

Enterprise Application Integration (EAI) is concerned with making applications work together seamlessly, by sharing data and functions among data sources and applications across heterogeneous platforms, facilitated by the use of common middleware. The applications can be a mixture of in-house developed, commercially packaged, and open-source applications; some are contemporary applications, some are legacy systems. Some of the applications were not designed to work together originally. The majority of the definitions set the boundary of EAI for integration within an enterprise. Integration across enterprises involves additional and sometimes different considerations that are covered by solutions that fall under the category of business-to-business (B2B) integration [10]. Variations of the definition of EAI can be found in existing literature. Some of the definitions include the idea that the ultimate goal of EAI is to enable the creation of new business solutions.

Historical Background

Integration of applications started with the sharing of data using files. An application outputs data to a file with specific file naming and placement conventions so that other applications can retrieve the contents of the file. Important decisions include deciding on the file format and how frequently to produce and consume the file. A common issue is that applications can get out of synchronization. When an application uses obsolete data, resulting consequences can be severe.

The next step in the evolution of integration came when client-server architectures became popular and gave rise to the notion of Application Programming Interfaces (APIs) through which a client can invoke the services provided by a remote server. The underlying remote invocation is handled by Remote Procedure Calls (RPC). The main disadvantage is that the API that is being invoked must be in a language and platform compatible with the calling application, which implies multiple APIs are needed to expose an application to other applications on different platforms and languages.

With the advent of object-oriented programming, distributed components in the form of objects allow applications to interact irrespective of their locations and platforms. Common Object Request Broker Architecture (CORBA) took the RPC ideas further by applying them to an object-oriented environment. Although the status of CORBA being the standard distributed object platform has been replaced by more recent platforms such as Java 2 Platform Enterprise Edition (J2EE) and .NET, the ideas from CORBA are still being used today.

When using RPC and distributed objects, applications invoke the APIs of each other directly. This is known as *point-to-point* integration and it has some drawbacks which are aggravated as the number of applications to be integrated increases. Point-to-point integration results in tightly coupled applications and is not a scalable approach.

In the 1990s, ERP (Enterprise Resource Planning) systems became popular, providing a variety of business functions such as accounting and inventory management. Enterprises want to leverage existing functions and data in the ERP systems, but it was difficult to integrate them with other applications. Vendors responded by providing *connectors* (also called *adapters*) to the ERP systems. As the number of applications to be integrated increased, it became clear that a common infrastructure which provides services to allow the applications to interact and manages those interactions would be useful. Such an infrastructure would also minimize the dependencies and assumptions applications make about each other. Industry analysts created the term EAI to represent the software in this infrastructure which enables and facilitates integration.

The success of the Internet provides new opportunities to develop strategic business solutions that bring competitive advantages and fuel the need for integration even further. Standards such as Extensible Markup Language (XML) and industry-specific business protocols provide more common basis for integration. The shift to service-oriented integration, supported by technologies such as Web Services, SOA (Service Oriented Architecture), and SCA (Service Component Architecture) have now taken integration into a new generation.

Foundations

Integrating a diverse set of applications is a big challenge. Following are typical business and technological issues involved in EAI:

- The mindset and policies of organizations need to adapt to an integrated environment. Traditionally, an IT department has almost full control over the applications for which it is responsible. In an integrated environment, an application is only one part of a bigger system. Successful integration requires the cooperation between business units and organizations in addition to cooperation between applications.
- Considerable pressure and expectation are placed on integrated systems to operate seamlessly and optimally, even under adverse and unexpected conditions. This is the result of many integrated systems providing vital business functions where impact, such as cost and reputation, to the business when a failure occurs can be severe.
- The number of applications to be integrated can range from tens to hundreds, or even thousands in a large enterprise.
- Applications to be integrated comprise of legacy systems running on mainframes (e.g., CICS and MVS applications), Commercial-Off the Shelf (COTS) applications such as SAP R/3 and PeopleSoft, in-house developed applications, database management systems, corporate directories, and open-source applications. Such a diverse set of applications often use different technologies and run on different platforms; use different architectural styles: monolithic, 2-tiers, and 3-tiers. Data format and semantics among the applications are usually different. This is known as *semantic dissonance*. Qualities of services among the applications may also be different and need reconciliation.
- Applications, particularly the older ones, were not originally designed to be integrated with.
- Applications are often autonomous: they have different owners and will continue to evolve independently.
- Applications are often geographically distributed which requires remote communication. Networks can be slow, unreliable, and insecure. The applications themselves may also be unavailable at times.
- Applications cannot be changed for the purpose of integration due to organizational and technical reasons. If changes are needed, they must be minimized.
- The breadth of business knowledge and technologies involved require a broad set of skills that is difficult to find in a small number of individuals or in a single organization.

To overcome the technical issues, EAI solutions employ a number of techniques:

- Use the design principle of *loose coupling* to minimize assumptions and dependencies applications have on each other. This will allow applications to evolve independently and if one application changes or an exception occurs, impact to other applications is minimal.
- Use connectors to adapt applications to the target integrated environment. Connectors provide a variety of services including mapping interfaces, transforming data, and exception handling. Performance and reusability are two important aspects in the design of connectors. Many reusable and configurable connectors for a variety of applications and legacy systems are available from vendors.
- Use a middleware integration layer to manage interactions among the applications. This middleware layer runs on a distributed object technology platform and uses a hub and spoke or a Message Bus [1] (also known as Enterprise Service Bus (ESB)) architecture. These architectural styles are highly scalable. Applications only need to be connected to this middleware integration layer, point-to-point integration is no longer needed.
- The middleware layer supports synchronous interactions and *asynchronous messaging*, provides message transformation and routing services, supports distributed transactions and provides security services.
- Asynchronous messaging is one of the techniques to achieve loose coupling. With asynchronous messaging, an application sends a message to a message queue or a Message Channel [1] and the application continues its own processing. There is no need for the sending application to stop and wait for the receiving application to receive the message and respond. This is known as *non-blocking* communication. The message is persisted by the Message Channel which is responsible for forwarding the message to the receiving application. The messaging system also provides transactional support for messages and handles recovery in the case of failure.
- Message transformation is another technique to achieve loose coupling. Transformation allows the format of data and messages between interacting applications to be different.
- Routing allows for the dynamic determination of the target application that is to be invoked. The source application which initiates the interaction can still have some control over which target application it interacts with, but it no longer has to bear the full responsibility of determining the address of the target application, a process known as *binding*. The middleware supports *dynamic binding* which allows the actual address of a target application to be determined based on factors such as load balancing and availability. Dynamic binding is yet another technique to achieve loose coupling.
- The middleware layer may also include sophisticated servers such as:
  - Process servers with embedded workflow engines to orchestrate or choreograph applications to carry out higher level business processes. Process servers automate the execution of tasks within a business process expressed using a standard language such as Business Process Execution Language (BPEL). Process servers often provide the capability to use business rules to control the execution of business tasks.
  - Database and directory integration servers which federates multiple databases and directory servers.
  - Portal servers which aggregate data from multiple applications to create a unified view and interaction experience for end users.

Although the middleware integration layer provides a great deal of assistance in creating an integration solution, even before using the middleware, integration architects still have much to reconcile between business and technical requirements. One important area of focus is on quality of service. Even when a participating application meets all of the functional and data requirements for integration into a target environment, that application may lack quality of service. Quality of service attributes of an application can include:

- Availability
- Reliability
- Performance
- Whether an application idempotent (i.e., no negative effects if invoked multiple times)
- Whether multiple instances of the application can be invoked concurrently
- Whether an application can participate in a transaction

Approaches to Application Integration

With such a variety of technologies, considerations that are technical and business in nature and span inter and intra enterprises, patterns are good ways of acquiring a high level understanding of application integration, its key concepts, the recurring problems, and recommended solutions. Over the years, approaches such as the ones captured by Adams [2], Trowbridge et al. [3] and Ruh et al. [4] have been documented as a set of application integration patterns.

- *Presentation integration* connects applications to give end users a unified view of the data. Among the three approaches to integration, this is the least complicated. Trowbridge [3] referred to this approach as Portal Integration and elaborated upon the different levels of sophistication, ranging from simple display of data from multiple applications to *cross-pane interactivity* where user actions in one portion of the display can affect another portion of the display. Adams [2] provided a broader coverage of presentation issues by considering access from different types of devices such as voice-enabled devices, personalized delivery of information according to user preferences, and security considerations such as single sign-on.
- *Data integration* connects multiple data stores to give applications a unified view of the data. It is responsible for resolving semantic dissonance among the data stores. The two major approaches to data integration are either physically replicating data from the multiple data stores into a central data store, or combining data from multiple data stores in real time presenting an aggregated but virtual view of the data. Sauter [5,6,7,9] identified a several patterns for data integration including Data Consolidation, Data Federation, and Data Cleansing.
- *Process integration* orchestrates interactions among multiple applications, using the services or subprocesses provided by the applications to perform a higher level business process often involving business rules. As part of process integration, business performance of processes and activities can be monitored to provide data that are vital for day to day business decisions.

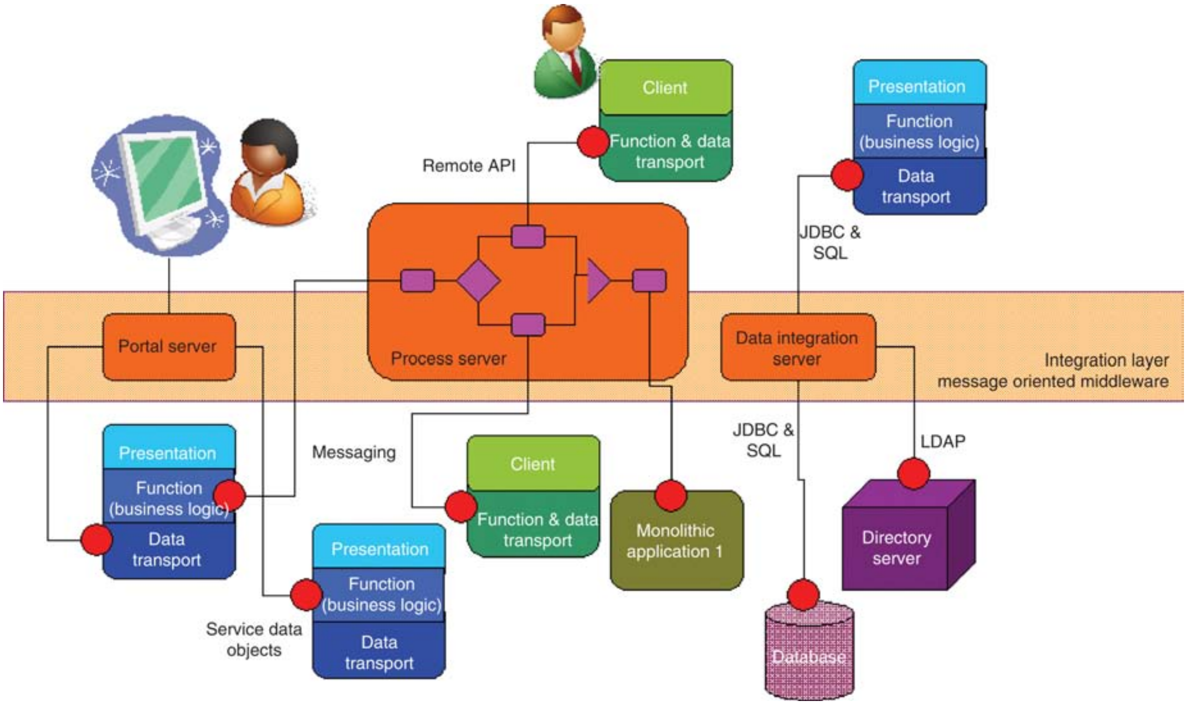
In an integration solution, it is not uncommon to see a combination of the different approaches. Adams [2] additionally provided business patterns to consider the business drivers and requirements before considering the integration approaches.

Figure 1 depicts the different approaches to integration being used together.

Connecting to Applications

Once the overall business purpose of an integration effort is clear and a logical high level understanding of the integration solution has been achieved, another important consideration is how to connect the participating applications to the middleware layer. Many contemporary business applications follow a three-tiered architecture with one layer for presentation, one layer for business logic, and a third layer for handling data. This three-tiered architecture gives rise to three common ways of connecting to an application:

- Connect an application to the middleware layer through the presentation tier of an application. This is the most limiting way of connecting to an application because it is restricted to the data and functions that an application exposes to its end users. It usually is slow in performance and is tightly coupled with the presentation tier of the application and therefore has limited potential for reuse. *Screen scraping* is a common technique to integrate an application through its presentation tier. The integration is achieved by capturing data displayed by the application and extracting relevant information from it.
- Connect an application to the middleware layer through the business logic tier of the application. Connections can be achieved by invoking APIs provided by the application, for instance, by using distributed object technologies, or by invoking the web services interfaces of the application. Asynchronous messaging is used in many cases. Hohpe and Woolf [1] provide an extensive set of asynchronous messaging patterns used for application integration.
- Connect an application to the middleware layer through the data tier of an application. This type of connection mechanism is more flexible and provides access to more data than connecting to the presentation tier. The main disadvantage is that the business logic that resides in the application that processes and manages the data is bypassed, which can lead to data integrity problems. Another disadvantage is that the integration is tied to the data



Enterprise Application Integration. Figure 1. The middleware integration layer connecting various applications (monolithic, 2-tier, 3-tier) and data sources (database, directory server). Different types of integration are supported by the integration layer.

model of the application. Some applications do not consider their data model to be part of their public interface. Directly accessing a "private" data model is a fragile approach. If the data model changes, the integration code has to change accordingly. Connecting to the data tier is more suitable if separate logic, independent of the ones in the existing applications, is intended for processing the aggregated data. A data warehouse is a typical example where logic resides to analyze the aggregated data for the purposes of creating reports and making decisions. The data warehouse typically connects to various data sources through their data tier.

- A variety of tools and data access middleware can be used to connect to an application through the data tier. Such tools and middleware connect to the databases using standard APIs such as Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC). Often, the tools and middleware provide data transformation as well.

Among the three ways of connecting to an application, connecting to the business logic tier is the most flexible and solves the widest range of integration problems, and has the greatest reuse potential. When new business logic and in particular, workflow and transactional integrity, are involved, connecting to the business logic tier is the most appropriate. However, connecting to the business logic tier is also the most complicated and requires the greatest set of skills. Connecting to the data tier directly will always be useful where the situations call for direct connection to data sources and bypassing existing business logic is not an issue.

It should not be surprising that the three major approaches to integration have affinity to the different ways of connecting to an application. The presentation integration approach is typically combined with connecting to an application via its data or presentation tier. The data integration approach is most often associated with connecting to an application via its data tier. Finally, the process integration approach is typically achieved by connecting to an application via its business logic tier.

Standards for Application Integration

While EAI is often interpreted as application integration within an enterprise, with the need for application integration outside of an enterprise, with supply chain, business partners, and regulatory agencies, many organizations want a consistent strategy and approach to integration, irrespective of whether the integration is for inter or intra enterprise. Vendors support this by having their offerings designed to support both types of integration, using the same set of tools and concepts, maximizing reuse while avoiding duplication. Many tools support standards that are applicable to integration. Chari and Seshadri [8] proposed a framework for organizing and navigating the standards according to three orthogonal dimensions. Each dimension is divided into multiple categories.

- *Application architecture*: This dimension classifies a standard according to the architectural layer of an application to which the standard applies. Three architectural layers comprise this dimension: presentation logic, business logic, and data logic.
- *Integration level*: This dimension classifies a standard according to the level of integration to which the standard applies. Three levels of integration comprise this dimension: transport, data format, and process.
- *Industry domain specificity*: This dimension classifies a standard according to whether the standard is applicable to a variety of industries or to a particular industry.

Some standards are sufficiently broad in scope that they belong to multiple categories within a dimension.

Key Applications

When EAI is employed effectively, an enterprise can leverage its existing assets:

- To provide new products and services. An example is the provisioning of an online travel reservation application to employees so that by interacting with a single application, an employee can obtain approval for a business trip, book flights, and make reservations for hotel and car rental.
- To improve its relationships with customers, suppliers, and other stakeholders. An example is when an enterprise wants to obtain a consolidated view of all its business relationships with a customer.
- To streamline and improve its operations, for example, by reducing data redundancy and overlapping functions. An example is when mergers, acquisitions, or spin-offs happen. With these business activities come duplicated and fragmented operations such as multiple payroll systems, corporate directories, and so on, which can be streamlined.
- To simplify interactions among its applications by adopting a standard approach to integration. Once an integration strategy has been created and supported by an appropriate technical infrastructure, adding new participating applications or developing new applications with integration as a key consideration will no longer be as difficult.

Cross-references

- ▶ Business Process Execution Language
- ▶ CORBA
- ▶ Enterprise Service Bus
- ▶ Java Database Connectivity
- ▶ Messaging Systems
- ▶ Multi-Tier Architecture
- ▶ Open Database Connectivity
- ▶ RMI
- ▶ Service Component Architecture (SCA)
- ▶ Service Oriented Architecture
- ▶ Web Services
- ▶ XML

Recommended Reading

1. Hohpe G. and Woolf B. Enterprise Integration Patterns – Designing, Building, and Deploying Messaging Solutions. Addison Wesley, Reading, MA, 2004.
2. Adams J. IBM Patterns for e-business: application Integration pattern. IBM developerWorks. http://www-128.ibm.com/developerworks/patterns/application/index.html
3. Trowbridge D., Roxburgh U., Hohpe G., Manolescu D., and Nadhan E.G. Integration Patterns. Microsoft Corporation, 2004.
4. Ruh W., Maginnis F., and Brown W. Enterprise Application Integration – A Wiley Tech Brief. Wiley, New York, 2001.
5. Sauter G., Mathews B., Selvage M., and Lane E. Information service patterns, part 1: Data federation pattern. IBM developerWorks, July 28, 2006. http://www.ibm.com/developerworks/webservices/library/ws-soa-infoserv1/
6. Sauter G., Mathews B., Selvage M., and Ostic E. Information service patterns, part 2: Data consolidation pattern. IBM developerWorks, Dec 5, 2006. http://www.ibm.com/developerworks/webservices/library/ws-soa-infoserv2/
7. Sauter G., Mathews B., and Ostic E. Information service patterns part 3: Data cleansing pattern. IBM developerWorks, Apr 6, 2007. http://www.ibm.com/developerworks/webservices/library/ws-soa-infoserv3/
8. Chari K. and Seshadri S. Demystifying integration. Commun. ACM, 47(7):59–63, 2004.
9. Dreibelbis A., Hechler E., Mathews B., Oberhofer M., and Sauter G. Information service patterns, Part 4: Master data management architecture patterns. IBM developerWorks,

Mar 29, 2007. http://www.ibm.com/developerworks/db2/library/techarticle/dm-0703sauter/

10. Gullege T. What is integration? Ind. Manage. & Data Syst., 106(1):5–20, 2006.

Enterprise Content Management

Frank Wm. Tompa
University of Waterloo, Waterloo, ON, Canada

Synonyms

ECM; Office automation; Records management; Document management

Definition

Enterprise content refers to the collections of records and documents that are used in support of business processes. Much of this data is unstructured or semi-structured text created by word processors and other productivity software as part of an enterprises' standard operating procedure. Enterprise content management applies database principles to collect, organize, index, and preserve such data, and ECM systems provide facilities for search, browsing, update, workflow management, web content management, collaboration support, version control, access control, record retention and destruction, legal compliance, and quality control.

Office automation typically refers to the components that create individual documents and manage the workflow. Records management refers to the components that protect and preserve the data and ensure that it is eventually archived or destroyed according to the enterprises' policies.

Key Points

Until recently, unstructured data was usually stored on the individual workstations of the office personnel who created them. This made the data inaccessible to those who wished to apply business intelligence procedures. Enterprise content management systems provide centralized mechanisms to access and manipulate the data, sometimes requiring that the data itself be centralized but often supporting distributed (federated) data management.

In essence, enterprise content management applies database principles and practices to the diverse forms of data that were traditionally outside the purview of relational database management systems.

Cross-references

- ▶ Access Control Administration Policies
- ▶ Document Databases
- ▶ Regulatory Compliance in Data Management
- ▶ Workflow Management

Recommended Reading

1. Jenkins T. Enterprise Content Management: What You Need to Know, Open Text Corporation, Waterloo, ON, Canada, 2004.

Enterprise Information Integration

▶ Information Integration

Enterprise Service Bus

Greg Flurry
IBM SOA Advanced Technology, Armonk, NY, USA

Synonyms

ESB; Service bus

Definition

The enterprise service bus (ESB) is a key architectural pattern in the larger architectural pattern called service-oriented architecture (SOA). The ESB supplies loosely coupled connectivity between service requesters and service providers in service-oriented solutions. Loose coupling permits a clean separation of concerns (temporal, technological, and organizational) between the parts in a solution, enabling flexibility and agility in both business processes and IT systems.

Historical Background

SOA is perhaps the latest stage in the continuing evolution of distributed computing. SOA inherits many principles form earlier stages of distributed computing. Client/server introduced the notion of less monolithic application architectures. Distributed objects focused on smaller-grained function and introduced encapsulated behavior and well-defined interfaces. Web services focused on the use of standards to enhance interoperability. SOA brings a strong focus on the principles of service definition according to business value instead of IT value, service composition via business processes using choreography, and separation of concerns to promote reuse, flexibility and agility.

The ESB contributes greatly to the principle of separation of concerns in SOA. Similar to SOA, the ESB is perhaps just the latest stage in the continuing evolution of the connectivity used in distributed computing and inherits principles from earlier stages. Various messaging systems (sometimes called message-oriented middleware) enabled direct point-to-point interaction. Enterprise application integration introduced centralized management of indirect connectivity and the ability to adapt disparate applications or data sources. Web services emphasized the notion of well-defined interfaces, the use of standards and a service registry to facilitate dynamic behavior. The ESB, in the context of SOA, strengthens the focus on providing loosely-coupled, highly dynamic interactions between service requesters and service providers, whether they adhere to standards or not.

Foundations

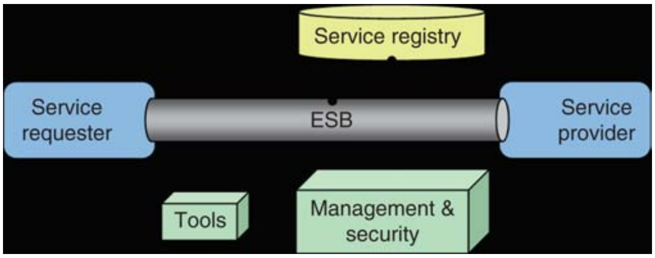
The architectural pattern called the ESB provides the connectivity that enables service interaction in SOA. As illustrated in Fig. 1, in any service interaction there are two roles, the service provider which offers a business function and the service requester that needs the business function. Service requesters and providers interact by exchanging messages. So more precisely, the ESB architectural pattern provides connectivity between service requesters and service providers. The ESB acts as a logical intermediary in the interactions; it intercepts messages from the requester, performs processing, often called mediation, on those messages, and sends the processed messages to the provider. Thus the ESB enables loosely-coupled service interactions. Loose coupling permits a clean separation of concerns (temporal, technological and organizational) between application services to enable flexibility and agility desired from SOA.

The ESB can be physically realized in different ways. The ESB appears like a centralized hub in Fig. 1, and the physical realization of the architectural pattern in many solutions in fact is a hub. The ESB, however, can be distributed so that mediation can physically take place in the service requester's environment, the service provider's environment, in one or more hub environments, or in any combination, allowing for optimization based on the requirements of the solution.

The ESB and Connectivity

Service requesters and service providers implement the application or business logic in a solution, targeted at achieving domain-specific business goals. The ESB implements the connectivity logic in a solution, targeted at achieving loosely-coupled, flexible, and on-demand interactions between service requesters and providers. In an ideal service oriented solution, separation of business logic and connectivity logic is clean, meaning that the services implement no connectivity logic, and that the ESB implements no business logic. Only by architecting this clean separation can an enterprise achieve the flexibility, agility and reuse sought from SOA.

It is sometimes difficult to distinguish between business logic and connectivity logic. There are guidelines, but the distinction may depend on the nature of the organization, or even a particular situation within the organization. One guideline leverages the distinction between semantics and syntax. A service provider performs actions which depend on the semantics, or meaning, of a service request, and so has to understand the meaning of the request; e.g., it creates, reads, updates or deletes business related



Enterprise Service Bus. Figure 1. Enterprise Service Bus.

entities. The ESB compensates for any mismatches in the syntax, or form, of the request between the service requester and provider, and has no need to understand the meaning of the request and does nothing to implement the semantics implied by the request. A related guideline emphasizes the distinction between achieving business objectives versus simply enabling interaction; since the ESB is an intermediary, removing the ESB should have no impact on logically achieving the business objectives; in other words, it adds no semantic value to a service interaction; put another way, the ESB never provides services, it always simply connects services.

Part of the difficulty distinguishing business logic and connectivity logic comes from the reality that the connectivity logic may itself be defined or driven by business considerations. For example, the ESB may route service requests to one of several candidate providers based on factors important to the proper functioning of the business, such as priority attributed to the requester or the time of day.

ESB Core Principles

Support for various forms of mediation allows the ESB to fulfill two core principles in support of separation of concerns. The first principle is service virtualization. The term refers to the ability of the ESB to compensate during service interactions for syntactic differences in:

- Protocol & pattern – Interacting services need not use the same communication protocol or interaction pattern. For example, a requester may require a single monolithic interaction via HTTP, but the provider may require interaction via some message-oriented protocol, using two correlated interactions. The ESB provides the conversion capability needed to mask the protocol and pattern switch.
- Interface – Service requesters and providers need not agree on the interface for an interaction. For example, the requester may use one form of message to retrieve customer information, and the provider may use another form. The ESB provides the transformation capability needed to reconcile the differences.
- Identity – A service requester need not know the identity (e.g., address) of the service provider, or vice versa. For example, service requesters need not be aware that a request could be serviced by any of several candidate providers at different physical locations. The actual provider is known only to, and is chosen by the ESB, and in fact, may change without impact to the requester. The ESB provides the routing capability needed to mask identity.

The second core principle is aspect-oriented connectivity. Service oriented solutions include multiple cross-cutting concerns like management and security. The ESB can implement or enforce such cross-cutting concerns on behalf of service requesters and providers, removing such concerns from of environment of the requesters and providers.

The application of these core principles allows the ESB to affect the qualities of service in interactions. Some aspects of an interaction can be abstracted away from the participants. Consider auditing, for example; if a solution requires auditing, the ESB can "add" it to interactions with no impact on the services. Similarly, the ESB can "add" or perhaps "enhance" qualities of service by retrying failed interactions, or in more sophisticated situations matching requester requirements with provider capabilities.

The ESB in an SOA Context

Figure 1 shows that the ESB performs its role in the larger context of SOA working with other parts of the architecture. The service registry (sometimes called a service repository or service registry/repository) contains and manages the metadata present in a service oriented solution; examples include interface descriptions, endpoint address and policies covering service level agreements, security relationships and so on. This service metadata contained in the registry, and thus the registry itself, have a broad scope in service oriented solutions, spanning governance, development and administration as well as runtime. The ESB can provide value using only static metadata provided during development, but realization of fully dynamic service virtualization and aspect-oriented connectivity requires the ESB to access a service registry at runtime to implement the dynamic connectivity required in a solution. Thus, the service registry can be considered the preferred way to configure the ESB, i.e., the service registry is a policy administration point for the ESB, and the ESB can be considered a policy enforcement point for the registry. As a result, the service registry, while not part of the ESB due to it broader role in SOA, is characterized as tightly coupled to the ESB.

Note that some important capabilities of any service oriented solution, specifically those related to management, are shown independent of, but accessible by the ESB. Management capabilities such as security and IT monitoring and management have a solution-wide scope, and require the coordination and cooperation of parts of a solution beyond the scope of the ESB. The ESB does not provide connectivity between services and management capabilities, and it is possible to secure and manage a solution without participation of the ESB. It is also possible, and frequently desirable, for the ESB to take an explicit, active role in helping secure and manage a solution as part of aspect-oriented connectivity. In this situation, the security and management policies are set by the policy administration points outside the ESB, and one can consider the ESB as a policy enforcement point and sometimes a policy decision point for such policies. Thus policy is set using management and security services that have no direct relationship to the ESB, but the ESB helps enforce the policy. As a result management capabilities are characterized as loosely coupled to the ESB.

An ESB requires tools for development and administration of the ESB. Developers use tools to develop the connectivity logic, or mediations, running in the ESB. Similarly, administrators use tools to deploy mediation and administer the ESB post-deployment. Ideally such tools leverage the service registry. For example, development tooling might allow the mediation developer to find the service providers required for an interaction using the registry; administration tooling might allow addition, deletion or modification of service metadata intended to impact the dynamic behavior of a solution.

Key Applications

An enterprise service bus provides the loose coupling necessary to allow service oriented applications to achieve the desired degree of separation of concerns.

Cross-references

► Service Component Architecture (SCA)
► SOA

Recommended Reading

1. An introduction to the IBM enterprise service bus. Available at: http://www-128.ibm.com/developerworks/webservices/library/ws-soa-progmodel4/index.html. July, 2005
2. Discover how an ESB can help you meet the requirements for your SOA solution. Available at: http://www-128.ibm.com/developerworks/architecture/library/ar-esbpat1/. April, 2007
3. Enterprise service bus. Available at: http://en.wikipedia.org/wiki/Enterprise_service_bus. September, 2008
4. The enterprise service bus: making service-oriented architecture real. Available at: http://researchweb.watson.ibm.com/journal/sj/444/schmidt.html. 2005

Enterprise Terminology Services

Lawrence Gerstley
PSMI Consulting, San Francisco, CA, USA

Definition

Enterprise Terminology Services refers to the complete lifecycle of vocabulary creation, publication, and supporting processes for the entire electronic medical record system. The supporting processes may include quality assurance, search and retrieval supporting other systems, and mapping for interoperability. Supporting systems, such as managing workflow and regular publication, are also discussed.

Historical Background

One repeatedly encounters the same missing processes and system components required to enable a production level terminology system. In many cases, almost all of the focus is put onto the terminology and storage system, and terminology workflow, publication control, QA, and other sub-systems are added on as afterthoughts. The terminology system itself should have supporting structures, including historical data, metadata, publishing data, and workflow data to support a system that can be fully maintained and utilized by diverse groups.

Tooling in Terminology systems may be minimal or non-existent; existing enterprise-owned applications (such as an office suite of word processing, spreadsheets, etc.) are used to fill the voids. This often results in the author, editor, and user workflows being "shoe-horned" into the tools, rather than the tools being adapted to enable maximum data quality, efficiency, and clarity. The core terminology system and system tooling should be thought of together as the entire Enterprise Terminology System, and should evolve together to fit the needs of the users with the users' direct input.

Import and export of data to other internal enterprise systems, as well as importing and mapping to and from industry standards and vendors (such as SNOMED-CT, LOINC, ICD-9/10, CPT, First Databank/Medispan, etc.) needs to also be considered as a fundamental part of a complete system. Regular maintenance of supporting data, its impact on the enterprise terminology, and required quality assurance to approve changes can range from a few hours to months of effort. Planning and system tuning can pay back any planning and expenditures many times over.

Foundations

System Architecture

1. Core terminology database. Core data model itself may use established models, or be custom developed.
2. Supporting metadata to indicate individual row authorship, last changes, metadata commentary, Dublin Core, and other institution-specific information.
3. Human Workflow, providing complete terminology lifecycle tracking for terminology requests, overall system requests, ad hoc projects, and other project tasks that require tracking.
4. QA/Metrics Reporting layer, providing insights into overall current status, historical throughput, publishing status, and other system metrics.
5. Data interchange layer, providing services for import and export of data to the system, ideally supporting HL7 messaging as a subset of interchange.
6. Centralized knowledgebase for sharing of information throughout the authoring group and end-users.

Key Applications

Case Studies and Experiences

Experience with multiple organizations shows a surprising overlap in system issues and organizational challenges, despite different company work styles, authoring teams, and end-users. In most cases, the problems were due to expediency of getting a working system in place, a philosophy of "add it later" when missing systems were suggested during design phases or identified in the normal course of work. One interesting note is that in a majority of cases, users and managers turn immediately to a spreadsheet application to fill system functionality gaps, often with disastrous consequences.

Users employ spreadsheets to manage small projects, stage data before loading into the authoring system, and other ad hoc tasks. While spreadsheets provide wide flexibility in usage, their use often has unexpected side effects. For example, groups using spreadsheet exhibit internal codes and foreign keys changing through the dropping of leading and trailing zeroes and rounding of decimal places. Intended to aid in the creation of financial information, which was the foundation of spreadsheets, these transformations have serious effects to ICD-9 codes, foreign key mapping, and other scenarios where keys must be treated as immutable. When such functional deficiencies are discovered by authoring groups, attempts are often made to make the spreadsheet application more rigid by setting software options to be standard in the group (disabling automatic formatting), developing internal templates that users should not deviate from, and potentially even developing programmatic control (such as scripting languages like VBA) behind the spreadsheet's GUI.

This is a natural outcome of finding procedural weaknesses; however, the investment of effort is wrongly placed, creating an organic system that is created one piece and a time, creating a fragile and overly complex system. The problem stems from not using the correct technology for the task. If the task is to stage data for loading into the system, experience shows that the creation of a flexible staging model and authoring tools to create terminology in the same system that will ultimately store the verified and validated data centralizes the information so multiple individuals may collaborate. Additionally, the actual tasks of verification and validation are performed instantly making the data available in the production environment.

When spreadsheets are used to perform project management, problems arise when multiple versions of spreadsheets are distributed, updated by multiple individuals, passed in emails, and propagated through a variety of other means, confusion as to which spreadsheet acts as the source of "truth" reigns. Again, this results from employing the wrong tool for the task. Groups often employ project management tools, but such tools focus on the project manager, resulting in a single user overseeing a set of tasks. Terminology authoring groups often need to see overall progress of the active project. Additionally, work may need to be performed piecemeal by team members who can self-assign individual items to themselves from a centralized pool of outstanding work. These challenges are ideally suited to a human workflow engine, such as Serena Software's TeamTrack or K2's workflow products. Such systems provide the ability to create centralized, automated workflow entry, tracking, and reporting with workflows that can be modeled to accurately represent the way that group's work. Additionally, using such systems focuses the group's attention on methods of improving workflow, gaining efficiencies and accuracy, obtaining approvals, and recording those approvals from requesters and other interested parties.

In general, turning to open-source software products to help fill in the process gaps. Firstly, it permits selection of a software tool actually geared towards the problem in question. Too often software is chosen because it is already part of a group's inventory. As with spreadsheets, its use *can* be adapted to solve the problem at hand, but results in changing the group's workflow, de-scoping the desired functionality, and other short cuts. The effects of these ostensibly expedient decisions will be encountered later, and as noted earlier will lead to subsequent additional processes, short cuts, and errors. Using open-source not only allows finding close match between requirements and desired functionality, but also permits tight integration between systems by altering or extending the code base. Such a solution is often more palatable to IT and other engineering groups, and begins to develop a completely integrated work environment. The normal purchasing process is removed from the start-up time, and approvals usually become only technical approvals.

Standardizing and expressing editorial policies formally in a computationally leverageable manner yields many benefits. Using a very simple example, expressing the standards for term string display may include capitalization rules, maximum string length for a particular string, and spaces after different punctuation marks can be expressed as part of a centralized system parameter file. Using an XML format to express such rules delivers many benefits:

1. Such a file is human-readable, providing more transparency into the systems operation.
2. It is technology-agnostic, which means that any future application can process the file for use through standard libraries, regardless of language or platform.
3. Changes may be made easily through any number of tools, ranging from the simplest of text editors (potentially dangerous) to XML editors.
4. As an XML file, it may be versioned with any in-house versioning tool already employed for software control (e.g., CVS, Subversion, SourceSafe, ClearCase).
5. Through open-source technologies like XSLT and FO, the file can be self-documenting, generating formal PDF, Word, RTF, and other test formats for system documentation.

Terminology data models often serve as the focus of development, but often this model is developed to represent the universe of active terminology in a production-ready state. Consequently, it may have little in it supporting developing terminology that must go through a verification and validation process before it is released to outside users. Metadata supporting authoring, publication control, and quality assurance is often missing from the data model, which creates the need for supplemental systems to represent and alter such information.

To illustrate, consider the need to create a new diagnosis in the system, which must be mapped to the appropriate ICD-9/10 code(s) for that diagnosis. Until such a time as a certified coder has looked at the display string, potential synonyms, and the code itself, and certified that code as accurate, such a diagnosis should not be published for outside use. However, putting the data into the authoring system itself means that either the publication process knows not to include the record until it has been verified, or the entire publication of updated terminology must be held until all records are in a satisfactory, verified state (the "all-or-nothing" method of publishing). Sometimes, groups get around this by staging data outside of the authoring system until it is ready for loading, but this brings about additional issues, including all of the previously mentioned issues surrounding use of spreadsheets and other non-authoring applications. Additionally, staging data may lead to a situation where multiple updates to the same record may be prepared outside of the system, and such work will overwrite one another. This scenario often arises during coding updates, or quality assurance and auditing tasks.

These issues necessitate an authoring model that can support records in various stages of completion. Also, it should support different levels of record locking, so that others may see a record in a "dirty" status and be able (or unable) to alter it depending upon their system role. Such a model should include metadata about the terminology records themselves, including:

1. Which user created (initiated) the record?
2. When the record was created?
3. Which user last updated the record?
4. When the last update occured?
5. Commentary about the row itself.
6. Record status (dirty, draft, awaiting publication, propagated).
7. Record publication date/time.

Once an effective set of controlling metadata is in place tracking terminology flow and validation, maintenance routines may begin updating coding and other integrated data, such as pharmaceutical information. Too often such maintenance routines grow without any control over languages used, processes requiring automation, or even documentation for completing updates and handling diverse use-cases. This is the result of different individuals creating processes as time permits, with the tools that they are already familiar with, accomplishing what needs to be done at that point in time. The same maintenance process may contain a potpourri of technologies, each employed at a different point in the process. In general, a small set of tools should be selected and standardized for maintenance, and full documentation should be an absolute requirement for system maintenance. For scripting languages, Perl, Python, and Ruby will all do the job and usually fit within any group's existing IT environment.

Defined metrics reporting should be designed on top of the terminology model, and accessible to those within the team as well as management. Effective reporting systems have been assembled from purchased packages, such as Crystal Reports, as well as open-source packages, such as BIRT, PHP and Apache Cocoon. Cocoon is intriguing in that it uses data "pipelines" which defines how the data is created, transformed, and finally delivered to the requester. Pipelines work with XML structures, so the output at any stage can be brought into other systems. Cocoon also provides a separation between the generation and presentation of data, so that the same pipeline can generate PDF, XML, Doc, HTML, and a number of other formats.

Terminology systems are often the result of evolutionary development, and can benefit greatly from redesign and standardization. Although groups often balk at the time or investment that such a redesign would take, often improvement in just one of the aforementioned areas will pay back such an investment several fold. Additionally, the system will have improvements of all other areas. Internal standardization of the terminology system also prepares it for eventual integration with outside systems, which is an industry goal.

Cross-references

▶ Business Intelligence
▶ Business Process Management
▶ Clinical Data Acquisition, Storage and Management
▶ Controlled Vocabularies
▶ Data Integration
▶ Data Transformation
▶ Database Management System
▶ Dublin Core
▶ Information Lifecycle Management
▶ Meta Data
▶ Pipeline
▶ Process Optimization
▶ Process
▶ Storage Management
▶ XML Information Integration
▶ XML
▶ XSL/XSLT

Recommended Reading

1. Apache cocoon. Available at: http://cocoon.apache.org
2. Docbook. Available at: http://www.docbook.org
3. Dublin core metadata initiative. Available at: http://dublincore.org/
4. Fogel K. and Bar M. *1*CVS: Open Source Development with CVS, 3rd edn. Paraglyph, Scottsdale, AZ, 2003.
5. Leymann F. and Roller D. Production Workflow, Concepts and Techniques. Prentice Hall PTR, Upper Saddle River, NJ, 1999.
6. Walsh N. and Muellner L. DocBook: The Definitive Guide. O'Reilly, Sebastopol, CA, 1999.

Entity Relationship Model

Il-Yeol Song[1], Peter P. Chen[2]
[1]Drexel University, Philadelphia, PA, USA
[2]Louisiana State University, Baton Rouge, LA, USA

Synonyms

ER Model; ERM; Entity-Relationship Model

Definition

The entity relationship model (ERM) is a conceptual model that represents the information structure of a problem domain in terms of entities and relationships. The result of modeling using the ERM is graphically represented as an entity relationship diagram (ERD). Thus, an ERD represents the conceptual structure of a problem domain being modeled. ERDs are widely used in database design and systems analysis to capture requirements of a system or a problem domain. In particular, when used for data modeling, the ERD assists the database designer in identifying both the data and the rules that are represented and used in a database. ERDs are readily translated into relational database schemas.

Historical Background

The ERM was introduced by Peter Chen in 1976 [2]. The ERM and its variations have been widely used in database modeling and design, systems analysis and design methodologies, computer-aided software engineering (CASE) tools, and repository systems. The popularity of the ER approach led to many extended ER and semantic data models as well as to its variations in notations of ERDs [9]. The ER model and its related research work have also laid the foundation for object-oriented analysis and design methodologies, which led to the development of the Unified Modeling Language (UML). Chen [4] presents historical events related to the ERM up to the early 2000s and roles of the ERM on the development of CASE tools.

Foundations

The three basic modeling components of the ERM are entities, relationships, and attributes.

Entities, Entity Types, and Attributes

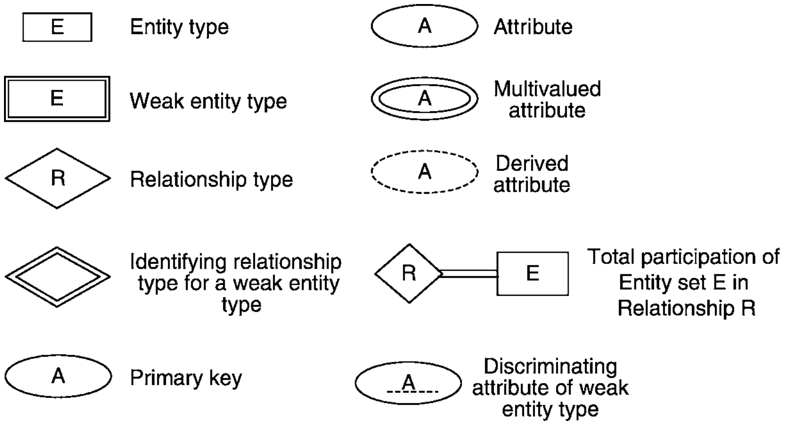
An *entity* is a primary object of a problem domain about which users need to capture information. An entity or an *entity instance* (or *entity occurrence*) is an object in the real world with an independent existence. An entity is characterized by its properties called *attributes*. Thus, an entity can be differentiated from other objects when it has at least one property different from the others. An *entity type E* defines a collection of entities that have the same attributes. For example, "John Smith" can be an entity instance of the entity type called *Employee*. Attributes of the *Employee* entity type could include *first name*, *last name*, *social security number*, *date of birth*, *address*, and *salary*. Attributes of a *Customer* entity type could include *customer number*, *first name*, *last name*, *address*, and *customer rating*. Thus, the *Employee* entity type is distinguished from the *Customer* entity type because they have different attributes. In the original ERM, all the attributes are assumed to be simple atomic values, representing numbers, character strings, and dates, not multi-valued attributes such as sets or structures.

Typical entity types usually belong to certain entity categories such as roles of people, locations, tangible things with values, organizations, events, transactions, etc. Thus, *Employee*, *Store*, *Product*, *Company*, *Reservation*, and *Sale* are all examples of entity types. In the literature, an entity and its entity type are not frequently distinguished from each other because the distinction between them is usually clear in the context.

An entity type has two types of properties: identifying attributes and descriptive attributes. An identifying attribute uniquely determines each instance of an entity type. An identifying attribute is called an entity identifier or a *key attribute*. For every entity type, there must be a designated key. For example, the attribute *social security number* would uniquely identify each instance of the entity type *Employee*. Hence, *social security number* is called the entity identifier or the primary key of the *Employee* entity type. A descriptive attribute describes a non-unique property of an entity type. Descriptive attributes of the *Employee* could include *first name*, *last name*, *date of birth*, *address*, and *salary*. Only those attributes that are meaningful in the problem under consideration are included in the ERD. For example, *eye color* would not be included in the *Employee* entity unless one needed to use *eye color* in a meaningful way, e.g., using it in an eye-color activated security system.

An attribute of an entity instance has a *value*. For example, "John" is one value of the attribute *first name*. The *domain* of an attribute is the collection of all possible values an attribute can have. For example, the domain of *first name* is a character string and that of *salary* is a number.

Figure 1 summarizes the notation used in ERDs. An entity type is depicted as a *rectangle* containing the name of the entity type. The name of an attribute is enclosed in an *oval* connected to the rectangle of the entity type they describe. An entity identifier or a key is underlined.


E
Entity type
A
Attribute
E
Weak entity type
A
Multivalued attribute
R
Relationship type
A
Derived attribute
Identifying relationship type for a weak entity type
R
E
Total participation of Entity set E in Relationship R
A
Primary key
A
Discriminating attribute of weak entity type

Entity Relationship Model. Figure 1. Symbols used in an Entity Relationship Diagram. (The notation for total participation was adopted from Elmasri and Navathe [6]).

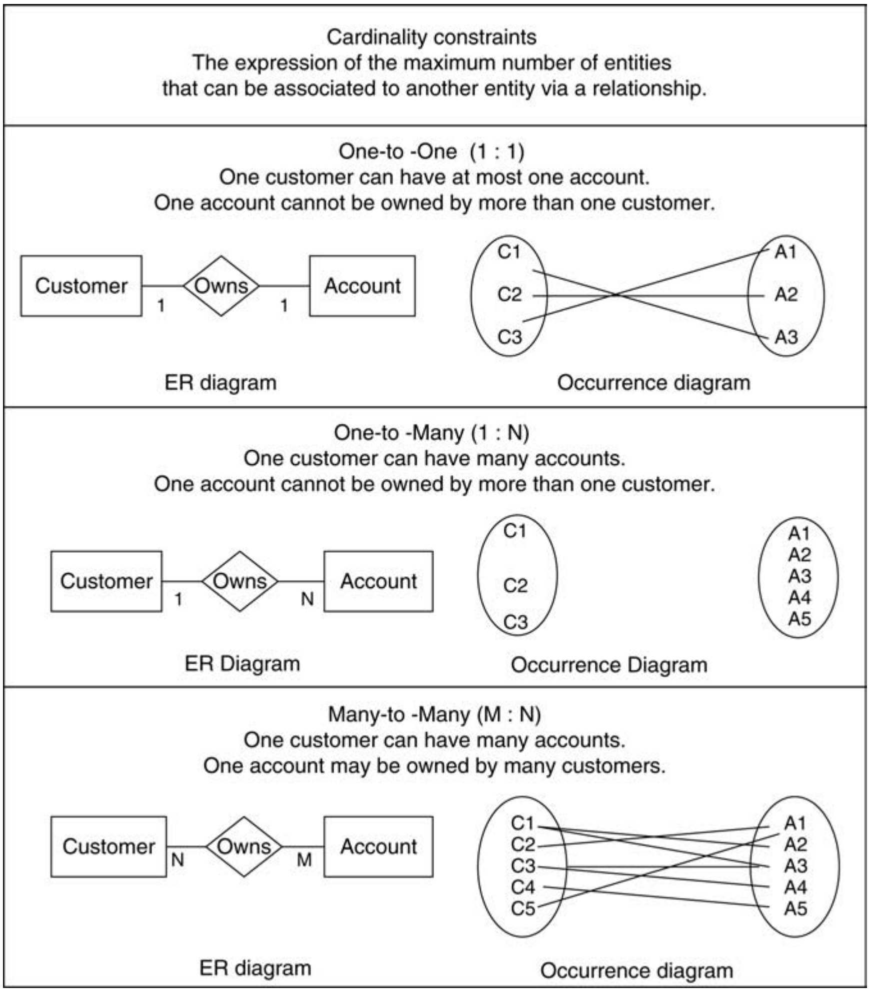
Relationships and Relationship Types

A relationship is an association between or among entities. A relationship in the ERM describes a meaningful association that needs to be remembered between or among entities. A relationship type between entity types *E* and *F* meaningfully relates some entities in *E* to some entities in *F*. Formally, a *relationship R* is a Cartesian product of $n \geq 2$ entities$\{R\ (e_1, e_2,...,e_n) \mid e_1 \in E_1, e_2 \in E_2,...,e_n \in E_n\}$, where e_i is an entity instance, E_i is an entity type, and $R\ (e_1, e_2,...,e_n)$ is a relationship. Here, n is called the *degree* of a relationship *R*, indicating how many entity types are participating in *R*. When $n = 2$, *R* is called a binary relationship; and when $n = 3$, *R* is called a ternary relationship. That is, a binary relationship is a meaningful association between two entity types, and a ternary relationship is a meaningful association among three entity types. In typical modeling, binary relationships are the most common and relationships with $n > 3$ are very rare. In a special binary relationship in which *E1* = *E2*, the relationship is called a recursive relationship because an entity is related to another entity of the same type. That is, a recursive relationship is a meaningful association between entity instances of the same entity type. As shown in Fig. 1, relationships are depicted by *diamonds* between or among entities in the ERD.

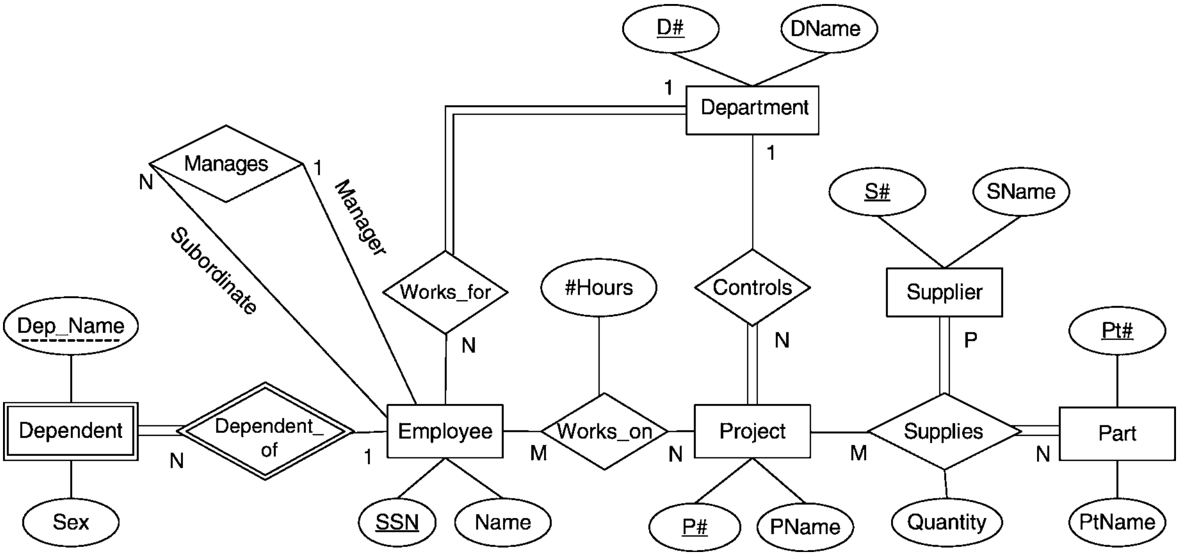
Cardinality and Participation Constraints

Cardinality is a constraint on a relationship between two entities. Specifically, the cardinality constraint expresses the maximum number of entities that can be associated with another entity via a relationship type. The values of cardinality are either "one" or "many". They are usually represented by either "1" or "N" in an ERD. A binary relationship can have three possible cardinalities: one-to-one (1:1), one-to-many (1:N), or many-to-many (M:N). Figure 2 shows the three cardinality constraints between two entity types *Customer* and *Account*. One-to-one cardinality states that one *customer* can have at most one *account* and one *account* can be owned by at most one *customer*. One-to-many cardinality says that one *customer* can have many *accounts*, but one *account* cannot be owned by more than one *customer*. Many-to-many cardinality says that one *customer* can have many *accounts* and one *account* may be owned by many *customers*. Figure 3 shows examples of a recursive relationship, binary relationships, and a ternary relationship as well as various cardinality constraints.

Participation is also a relationship constraint. Participation expresses the minimum number of entities that can be associated with another entity via a relationship type. There are two values for participation: *total* (also known as *mandatory*) participation and *partial* (also known as *optional*) participation. If every instance of an entity must participate in a given relationship, then that entity has total participation in the relationship. But if every instance need not participate in a given relationship, then the participation of that entity in the relationship is partial. In an ERD, the total participation of Entity type *E* to relationship type *R* is indicated by a double line from *E* to *R* [6], as shown in Fig. 1. For example, in Fig. 3, the participation of the *Department* entity type in the *Works_for* relationship is total, while that of the *Employee* entity type in the relationship is partial. This implies that a *Department* instance must


Cardinality constraints
The expression of the maximum number of entities
that can be associated to another entity via a relationship.
One-to -One (1 : 1)
One customer can have at most one account.
One account cannot be owned by more than one customer.
Customer
Owns
Account
1
1
C1
C2
C3
A1
A2
A3
ER diagram
Occurrence diagram
One-to -Many (1 : N)
One customer can have many accounts.
One account cannot be owned by more than one customer.
Customer
Owns
Account
1
N
C1
C2
C3
A1
A2
A3
A4
A5
ER Diagram
Occurrence Diagram
Many-to -Many (M : N)
One customer can have many accounts.
One account may be owned by many customers.
Customer
Owns
Account
N
M
C1
C2
C3
C4
C5
A1
A2
A3
A4
A5
ER diagram
Occurrence diagram

Entity Relationship Model. Figure 2. Connectivity of binary relationships.



Entity Relationship Model. Figure 3. An example ERD with different types of relationships.

have at least one *Employee*, while an *Employee* instance can exist without working for a *Department*.

Cardinality and participation constraints are business rules in the problem domain being modeled. They represent the way one entity type is associated with another entity type. They are also integrity constraints because they help ensure the accuracy of the database. These constraints limit the ways in which data from different parts of the database can be associated. For example, say the cardinality of the relationship between *Customer* and *Account* is one-to-one, as in Fig. 2. If customer C1 is associated with account A3, then C1 cannot be associated with any other accounts and A3 cannot be associated with any other customers.

Attributes of a Relationship

A relationship sometimes has attributes. A relationship attribute represents a property of the relationship, but not of any participating entity type. For example, in Fig. 3, attribute *#Hours* is a property of the relationship *Works-on*, not of *Employee* or *Project*. Another example is the *Quantity* attribute of the ternary relationship *Supplies* in Fig. 3, where *Quantity* is a property created by the interaction of three participating entity types *Supplier*, *Project*, and *Part*. Relationship attributes are most common in many-to-many relationships or ternary relationships.

Roles

The meaning of a relationship type is usually clear between two associated entity types. However, the meaning of a relationship type is not clear if there are multiple relationship types between the participating entity types. In these cases, roles are used to indicate the meaning of an entity to the associated relationship type. Roles are indicated in ERDs by labeling the relationship lines that connect diamonds to rectangles. For example, a recursive relationship always has two lines between an entity type and the recursive relationship. In Fig. 3, *Manages* is a recursive relationship; *Manager* and *Subordinate* are roles of *Employee* entity instances. Thus, the relationship can be read as "an employee with the role of a manager can manage many subordinate employees, and an employee with the role of subordinate can be managed by one manager employee." Thus, in recursive relationships, it is customary to add roles to clarify the meaning of each relationship line. In other occasions, role labels are optional.

Ternary Relationships

A ternary relationship is a meaningful association among three entity types. An important requirement of a ternary relationship is that the three entity types must always occur at the same time in the relationship. For example, in Fig. 3, *Supplies* is a ternary relationship among the *Supplier*, *Project*, and *Part* entity types. The semantics of *Supplies* can be read as follows [7,12]:

- For a given pair of Supplier and Project instance, there are many Part instances.
- For a given pair of Part and Supplier instance, there are many Project instances.
- For a given pair of Part and Project instance, there are many Supplier instances.

Weak Entity Types

An entity type that does not have its own unique identifier is referred to as a weak entity type. A weak entity type has one or more *owner* (or *strong*) entity types connected through one or more one-to-many relationships (called *weak relationships*). Therefore, the primary key of a weak entity type is always composite. The key consists of the primary keys of all the owner entity types and a *discriminator (or partial key)* of a weak entity set, where the discriminator is the set of attributes that distinguishes among all the entities of a weak entity.

As shown in Fig. 1, a weak entity type is depicted by double rectangles, a weak relationship by double diamonds, and a discriminator is underlined with a dashed line. For example, in Fig. 3, *Dependent* is a weak entity, *Dependent_of* is a weak relationship, *Employee* is the owner entity type of *Dependent*, and *Dep_Name* is the discriminator of *Dependent*. Thus, the primary key of *Dependent* is a composite of *Employee.SSN* and *Dependent.Dep_Name*.

Two important properties of a weak entity type are the *identifier dependency* (ID-dependency) and the *existence dependency*. The ID-dependency means that the primary key of the owner entity type is included in that of the weak entity type. Therefore, a weak entity always has total participation in its weak relationship, as shown in Fig. 3. Due to this property, the weak relationship is also called the identifying relationship. The existence dependency means the existence of a weak entity instance is dependent on the existence of its related owner entity instance. For example, in Fig. 3, if an *Employee* entity instance is deleted, then all of its associated *Dependent* entity instances must also be deleted.

ER Modeling Techniques

Developing a syntactically and semantically correct ER model for a domain requires both effective modeling techniques and experience. Techniques for developing a correct and complete ERD are beyond the scope of this article. An excellent starting point for understanding techniques of constructing ERDs from a problem statement is presented by Chen [3]. Chen presents a fundamental framework for developing an ERD from English sentence structure by showing the similarity between English grammar and ER modeling components. Batini et al. [1] present conceptual modeling using ERMs. Song et al. [9] present a comparison of popular ERD notations as well as a comparison between *n*-ary models and binary models. Ling [8] presents a notion of a normalized ER diagrams. Dullea et al. [5] discuss validity and redundancy in ER diagrams. Teorey et al. [10] cover in detail the database modeling and design using the ERM.

Translation of ERDs into a Relational Schema

This section briefly presents how to translate ERDs into relational schemas. There are several different methods for creating relational schemas from an ERD. See [6,10,11] for more detailed treatments of the subject. The one described here is the most popular technique [6,10]. The rules are:

1. Every entity type becomes a table.
   a. All the attributes of the entity become the attributes of the table.
2. Each 1:N relationship type is mapped into the associated N-side entity type as follows:
   a. For each N-side entity type:
      (a) Add the primary key (PK) of the 1-side entity type. This added PK becomes a foreign key.
      (b) Also add any descriptive attributes of the relationship.
   b. When this rule is applied to recursive relationships:
      (a) The PK of the 1-side entity type is annotated with its role name.
3. Each M:N relationship type becomes a separate table.
   a. Add PKs of the participating entity types to the table. They become foreign keys.
   b. Add all the descriptive attributes of the relationship type to the table.
   c. The PK of the table consists of the PKs of the two participating entity types.
   d. When this rule is applied to M:N recursive relationship type:
      (a) The two foreign keys are annotated with their role names.
4. Each 1:1 relationship type can be combined with either side of the entity type or can be treated like a 1:N relationship type.
5. A weak entity type becomes a table that includes the primary key of the identifying owner entity type.
   a. The PK of the table consists of the foreign key and the discriminator.
6. Each relationship type becomes a separate table.
   a. Add PKs of the participating entity types to the table. They become foreign keys.
   b. Add all the descriptive attributes of the relationship type to the table.
   c. The PK of the table consists of the foreign keys of the three participating entity types when the cardinality is many-many-many. When the cardinality is not many-many-many, the PK of the table consists of at least two foreign keys, where all the foreign keys coming from many-side entity types must be included in the PK of the table.

Applying the above translation rules to the ERD shown in Fig. 3 yields the following relational schema:

1. Employee (SSN, Name, D#, Manager_SSN).
2. Dependent (SSN, Dep_Name, Sex).
3. Department (D#, DName).
4. Project (P#, PName, D#).
5. Works_on (SSN, P#, #Hours).
6. Supplier (S#, SName).
7. Part (Pt#, PtName).
8. Supplies (P#, S#, Pt#, Quantity).

Key Applications

The ER model is widely used in database modeling and design and in conceptual modeling of information systems. Most commercial CASE tools support an ERM or some of its variations. The ER approach has been applied to design of object-oriented databases, data warehouses, temporal databases, spatial databases, and meta models. The ER model also laid the foundation for class modeling techniques of object-oriented analysis and design.

URL to Code

An international conference that focuses on entity-relationship approaches and conceptual modeling has been held regularly since 1979. The conference is now known as International Conferences on Conceptual Modeling and is cataloged in http://conceptualmodeling.org [7].

Cross-references

- ▶ Conceptual Schema Design
- ▶ Database Design
- ▶ Extended Entity-Relationship Model
- ▶ Logical Database Design: from Conceptual to Logical Schema
- ▶ Meta Data Repository
- ▶ Object Data Models
- ▶ Semantic Data Model
- ▶ Temporal Logical Models
- ▶ Unified Modeling Language

Recommended Reading

1. Batini C., Ceri S., and Navathe S.B. Conceptual Database Design: An Entity-Relationship Approach. Benjamin/Cummings, Reading, MA, 1991.
2. Chen P.P. The entity relationship model – toward a unified view of data. ACM Trans. Database Sys., 1(1):9–36, 1976.
3. Chen P.P. English sentence structure and entity relationship diagrams. Inf. Sci., 29(2–3):127–149, 1983.
4. Chen P.P. Entity-relationship modeling: historical events, future trends, and lessons learned. In Software Pioneers: Contributions to Software Engineering, M. Broy, E. Denert (eds.). Springer, NY, 2002, pp. 100–114.
5. Dullea J., Song I.-Y., and Lamprou I. An analysis of structural validity in entity-relationship modeling. Data Knowl. Eng., 47 (3):167–205, 2003.
6. Elmasri R. and Navathe S. Fundamentals of Database Systems, 5th edn. Benjamin/Cummings, Reading, MA, (2007).
7. ER conferences. Available at: http://conceptualmodeling.org
8. Ling T.W. A normal form for entity-relationship diagrams. In Proc. 4th Int. Conf. on Entity-Relationship Approach. IEEE Computer Society, WA, 1985, pp. 24–35.
9. Song I.-Y., Evans M., and Park E.K. A comparative analysis of entity-relationship diagrams. J. Comput. Softw. Eng., 3(4): 427–459, 1995.
10. Teorey T.J., Lightstone S.S., and Nadeau T. Database Modeling & Design: Logical Design, 4th edn. Morgan Kauffman, San Francisco, CA, 2005.
11. Teorey T.J., Yang D., and Fry J.P. A logical design methodology for relational databases using the extended entity-relationship model. Comput. Surv., 18(12):197–222, 1986.

Entity Resolution

▶ Record Matching

Entity-Relationship Model

▶ Entity Relationship Model

EPN

▶ Event Processing Network

Equality Query

▶ Membership Query

Equality Selection

▶ Membership Query

Equality-Generating Dependencies

Ronald Fagin
IBM Almaden Research Center, San Jose, CA, USA

Synonyms

egd

Definition

Equality-generating dependencies, or *egds*, are one of the two major types of *database dependencies* (the other major type consists of *tuple-generating dependencies*, or *tgds*).

To define egds, it is necessary to begin with the notion of an *atomic formula*, where an example is the formula $P(x_1,...,x_k)$, where P is a k-ary relational symbol, and $x_1,...,x_k$ are variables, not necessarily distinct.

Then egds are formulas of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow (x_1 = x_2))$, where

1. $\varphi(\mathbf{x})$ is a conjunction of atomic formulas, all with variables among the variables in **x**.
2. Every variable in **x** appears in $\varphi(\mathbf{x})$.
3. x_1 and x_2 are distinct variables in **x**.

Conditions (1) and (2) together are sometimes replaced by the weaker condition that $\varphi(\mathbf{x})$ be an arbitrary first-order formula with free variables exactly those in **x**.

Key Points

The most important example of an egd is a *functional dependency*, where an example is the formula

$$\forall x_1 \ldots \forall x_k \forall x_i'(P(x_1,\ldots,x_k) \\ \wedge P(x_1,\ldots,x_{i-1}, x_i', x_{i+1},\ldots,x_k) \rightarrow (x_i = x_i')),$$

where P is a k-ary relation symbol.

Historically, functional dependencies were the first database dependencies. They were introduced by Codd [2] for the purpose of database normalization and design. Fagin [3] defined the class of *embedded implicational dependencies*, which includes both tgds and egds, but he focused on the case where they are (i) unirelational (so that all atomic formulas involve the same relation symbol) and (ii) typed (so that no variable can appear in both the ith and jth position of an atomic formula if $i \neq j$). Beeri and Vardi [1] defined and named tgds and egds.

Cross-references

▶ Database Dependencies
▶ Functional Dependency
▶ Normal forms and Normalization
▶ Tuple-Generating Dependencies

Recommended Reading

1. Beeri C. and Vardi M.Y. A proof procedure for data dependencies. J. ACM, 31(4):718–741, 1984.
2. Codd E.F. Further normalization of the data base relational model. Courant Computer Science Series 6: Database Systems. Prentice-Hall, USA, 1972, pp. 33–64.
3. Fagin R. Horn clauses and database dependencies. J. ACM, 29(4):952–985, 1982.

ER Model

▶ Entity Relationship Model

ERM

▶ Entity Relationship Model

ESB

▶ Enterprise Service Bus

Escrow Transactions

Patrick O'Neil
University of Massachusetts, Boston, MA, USA

Synonyms

Escrow transactions

Definition

Escrow transactions [3] permit non-blocking concurrency for the most common high volume transactions, those that perform updates of hotspot data only by incrementing and decrementing aggregate data items. An example of such a transaction is one that increments dollar sale total of a customer for each purchase that decrements the quantity on hand of a product. A *hotspot* of a database under a high volume transactional workload consists of the set of data items, each one of which frequently needs to be updated by multiple concurrent transactions in order to maintain needed throughput. While this is impossible using read-write locking, since write operations do not commute, high-volume OLTP applications of this kind can be performed without blocking by a transactional system that performs increments and decrements with the appropriate protocols. Increment-decrement updates have been used for years in the IMS Fast Path product [1,2], supporting commuting high volume short-term transactions on data items in a centralized DBMS. Escrow transactions generalized this approach to support commuting long-lived transactions performing increment-decrement updates of aggregate data items, possibly in a distributed database.

Key Points

The term "escrow" is from banking: a large company entering into a business transaction in a remote location would not want to "lock" it's main bank account for this purpose, but would elect an *escrow agent*, to hold the money in trust pending fulfillment of the transaction; if the transaction succeeds, the escrow agent would transfer the money, but otherwise the money would be returned to the company. Note that this is a long-lived transaction in the sense that a business transaction using escrow might take days or weeks to complete. In the same way the Escrow transactions perform quick increments and decrements to an account, then hold no locks on what quantity remains, and so lock only the portion of an aggregate quantity needed for the business purpose of the transaction.

Escrow transactions can be illustrated with an example of a product table *Prod* with a primary key *prid* and rows containing data about warehouse storage of a product, e.g., storage_capacity, price, etc. Updates to columns of Prod are infrequent and take part in short-term update transactions, but orders changing Quantity on Hand of the product are held long-term in an Escrow data item qoh, as explained below. The qoh item can be thought of as a *composite* column for each row of the Prod table, having the appropriate prid and a number of int fields, as follows:

(qoh, prid, val, inf, sup)

The qoh item must be exempted from standard transactional recovery using before images and after images, since Escrow recover is special-purpose as described below. Note that an increment operation Inc(qoh, prid, tid, delta) by transaction tid can represent either an increment (with a positive value of delta) or a decrement (with a negative value of delta). It is assumed that product orders such as this can be held pending for an extended length of time (weeks), and either committed or aborted at the end of that time, with a fee paid by the customer for this flexibility; thus each Inc operation can be thought of as an *Option to Buy (Return)* if delta is negative (positive).

The field val represents the value of the balance if all outstanding transactions that have incremented this quantity should abort, while the field inf(sup) represent the lowest (highest) values val could attain if all transactions that have performed Inc operations with positive-valued (negative-valued) delta increments were to abort, while all increments with deltas of the opposite sign commit. Thus as new Inc operations are submitted, negative-valued deltas will lower the inf value and positive valued deltas will increase the sup value, while val will remain unchanged until one of the transactions commits. In addition, as each Inc operation causes qoh items to change, an Escrow log (Elog) is created and both the Elog and the qoh are made durable (written to mirrored disk for example), to guarantee that the Elog will be recovered in event of a system crash. Recovery from a system crash will therefore result in recovery of qoh and all Elogs, so each Escrow update is brought back to a state where it can be either committed or aborted. There are also Max and Min field values in the descriptor of the qoh data item to provide a constraint on the maximum and minimum attainable values for val. The seller will disallow an Inc(delta) with negative delta which brings inf below Min (usually zero since this would mean that some purchase order will fail if no items are returned, while there will also be a limit on how many positive delta increments can be accepted for a given product (sup will be constrained to be less than Max), since there is a limited amount of storage. Back-orders for purchase or return can be provided when circumstances permit.

Note that while Inc operations commute, they do not commute with Reads or Writes. One cannot Read or Write an Escrow item that has indeterminate value (between inf and sup), although a Read function call to return this range of possible values could be provided.

Escrow Commits and Aborts

The commutative property of Escrow transactions is based on the fact that updates by multiple concurrent transactions on any single Escrow data item can be committed or aborted in any order. This is motivated with a short example. Assume the qoh data item on a product with prid = 1234 begins with val = 1000, and that Max = 2000 and Min = 0. A sequence of transactions T_i updates this qoh creating Escrow logs, and then all T_i Commit (C_i) or Abort (A_i).

To start: (qoh, prid: 1234, val: 1000, inf: 1000, sup:1000)
Inc_1: Inc(qoh, prid: 1234, tid: 1, delta: −500)
Result: (qoh, prid: 1234, val: 1000, inf: 500, sup:1000), Elog: (tid: 1, delta:−500, qoh[1234])
Inc_2: Inc(qoh, prid: 1234, tid: 2, delta: + 500)

Result: (qoh, prid: 1234, val: 1000, inf: 500, sup:1500), Elog: (tid: 2, delta: + 500, qoh[1234]) C_2: (qoh, prid: 1234, val: 1500, inf: 1000, sup:1500), Elog for tid = 2 was used and is now invalidated.
A_1: (qoh, prid: 1234, val: 1500, inf: 1500, sup:1000), Elog for tid = 1 was used and is now invalidated.

Recall that a positive increment such as Inc_2: Inc (qoh, prid: 1234, tid: 2, delta: + 500) causes the sup value to increase by delta. Then an Abort A_2, reading off the Escrow log for Inc_2, causes the sup value to be decremented by delta again, with no other field changed, while a Commit C_2 causes the inf, and val values to be incremented by the same amount (if no other transactions remain active, the val, inf and sup must end with the same values). Similarly a negative increment Inc_1: Inc(qoh, prid: 1234, tid: 1, delta: −500) causes the inf value to be decremented by the negative delta; an Abort A_2, reading off the Escrow log for Inc_1, will cause the inf value to be incremented by delta (a positive value), while a Commit will cause the val and sup values to be decremented as well.

It should be clear from this that the minimum value possible for val at the end of a sequence of Commits and Aborts will be inf, and the maximum value will be sup. Any combination of a specific set of Aborts and Commits in any order will result in the same val value.

Distributed Escrow Transactions

A classical definition of distributed transactions has a Transactional Coordinator on one host performing updates on its own host and coordinating updates on several foreign hosts to provide ACID properties for a global transaction. The updates lock records on all hosts in the classical situation, often leading to difficulties in terms of blocking, especially if one of the non-coordinating hosts should crash at an inopportune moment during concluding Two-Phase Commit. Escrow transactions are perfect for situations like this, since they lock only the (usually small) portion of an aggregate quantity needed for the business purpose of the transaction, while original Escrow update automatically requests Phase 1 of a Two-Phase Commit. The upshot of all this is that there is no particular rush about completing Phase 2 of a Two-Phase Commit, and indeed even if the Coordinator host crashes after the final commit has started, one can afford to wait for recovery without blocking all the data items involved against further Escrow updates.

Cross-references

- ▶ Concurrency Control
- ▶ Distributed Transaction Management
- ▶ Logging and Recovery
- ▶ Transaction Model
- ▶ Transaction

Recommended Reading

1. Gawlick D. Processing "hot spots." In Digest of Papers – COMPCON, 1985, pp. 249–251.
2. Gawlick D. and Kinkade D. Varieties of concurrency control in IMS/VS fast path. IEEE Database Eng. Bull., 8(2):3–10, 1985.
3. O'Neil P.E. The escrow transactional method. ACM Trans. Database Syst., 11(4):405–430, 1986.

e-Services

▶ Web Services

ETL

▶ Extraction, Transformation and Loading

ETL Process

▶ Extraction, Transformation and Loading

ETL Tool

▶ Extraction, Transformation and Loading

ETL Using Web Data Extraction Techniques

▶ Web ETL

European Law in Databases

Michael Carroll
Villanova University, Villanova, PA, USA

Synonyms

Intellectual Property; License

Definition

European law provides three types of legal rights to database owners to control copying: (i) copyright law, (ii) the right to impose contracts or licenses on users, and (iii) a unique statutory right applicable to the non-copyrightable information – such as factual data – in European databases. This entry discusses only the third of these rights. This stand-alone or "sui generis" database right gives a database owner a claim against one who extracts or reuses the whole or a substantial part of the database contents, where "substantial part" is evaluated qualitatively or quantitatively.

Historical Background

Prior to 1996, the legal treatment of databases in Europe varied in the requirements for copyright protection. In 1992, the European Commission proposed that EU copyright law be harmonized with US law by requiring originality for copyright protection to apply. However, the European Commission also proposed that a separate legal right should be granted for unoriginal databases that require substantial investment to create or update. After further revision to ensure certain user rights, the proposal was adopted as Directive 96/9/EC of the European Parliament and of the Council on the legal protection of databases. The Directive required Member States to harmonize their copyright treatment of databases with US law and to further enact national legislation providing database owners with a new stand-alone legal right to control certain copying of non-copyrightable information in databases. All 25 countries subject to the Directive have complied with this requirement.

An analysis of the effect of the Directive conducted by the European Commission in 2005 concluded that "[w]ith respect to 'non-original' databases, the assumption that more and more layers of IP protection means more innovation and growth appears not to hold up." The report offered policymakers four options: (i) Repeal the whole Database Directive; (ii) Withdraw the sui generis right while leaving protection for creative databases unchanged; (iii) Amend the sui generis provisions in order to clarify their scope; (iv) Maintain the status quo. Comments were solicited from specific stakeholders. There was support for each option and as of January 2008 no legislation was pending.

Foundations

Article 7 of the Directive requires that: "Member States shall provide for a right for the maker of a database which shows that there has been qualitatively and/or quantitatively a substantial investment in either the obtaining, verification or presentation of the contents to prevent extraction and/or reutilization of the whole or of a substantial part, evaluated qualitatively and/or quantitatively, of the contents of that database."

This provision can be broken down into (i) the eligibility criteria and (ii) the scope of the rights granted to those deemed eligible.

Eligibility

Under the Directive, a database is "a collection of independent works, data or other materials arranged in a systematic or methodical way and individually accessible by electronic or other means." To qualify for this database right, the creator, which, unlike in European copyright law, can be a corporation, must have made qualitatively or quantitatively a substantial investment in either the obtaining, verification or presentation of the contents."

Notice that the eligibility criteria allow for multiple rightsholders in the same data. For example, Company *A* may make a substantial investment in collecting data. Company *A* may then sell the data to Company *B*, which makes a substantial investment in verifying the data. Company *B* may then sell the data to Company *C*, which makes a substantial investment in presenting the data.

This database right is initially held by the person or corporation which made the substantial investment, so long as (i) the person is a national or domiciliary of a Member State or (ii) the corporation is formed according to the laws of a Member State and has its registered office or principal place of business within the European Union. The database right lasts for 15 years from the date of publication or, in the case of unpublished databases, from the year of creation.

Scope of Rights – Extraction or Reutilization

Under the Directive, a rightsholder may bring a claim for unauthorized "extraction" or "reutilization" of a "substantial part" of a database.

Extraction is defined broadly to mean "the permanent or temporary transfer of all or a substantial part of the contents of a database to another medium by any means or in any form."

Reutilization is also defined broadly to mean "any form of making available to the public all or a substantial part of the contents of a database by the distribution of copies, by renting, by on-line or other forms of transmission."

A substantial part of the database is to be evaluated quantitatively or qualitatively, which means that even extraction or reuse of a small amount of data may infringe the right if that data has economic value to the rightholder.

Somewhat in conflict with the express language defining the right as covering only substantial parts of the database, Article 7(5) also provides that "repeated and systematic extraction and/or reutilization of insubstantial parts of the contents of the database implying acts which conflict with a normal exploitation of that database or which unreasonably prejudice the legitimate interests of the maker of the database shall not be permitted."

Exceptions and Limitations

The broad rights granted under the Directive are subject to specific exceptions and limitations.

Where copies of the database are sold in material form, such as on a CD ROM, the rightholder's reutilization right is exhausted after the first sale. This means one may resell or otherwise redistribute one's copy of the database without liability. Public lending of such a copy also is not an infringement of the reutilization right.

In addition, the Directive provides three specific exceptions under which a substantial part of a database may be extracted or reutilized without permission:

1. In the case of extraction for private purposes of the contents of a non-electronic database;
2. In the case of extraction for the purposes of illustration for teaching or scientific research, as long as the source is indicated and to the extent justified by the non-commercial purpose to be achieved; and
3. In the case of extraction and/or reutilization for the purposes of public security or an administrative or judicial procedure.

Importantly, under the Directive, the rightsholder may not supplement the protection of the database right with a contractual license that restricts the user's right to use insubstantial parts of the database without authorization.

Key Applications

At the national level, court judgments reflect a range of databases considered eligible for the sui generis right. These include listings of telephone subscribers; databases compiling case-law and legislation; websites containing classified advertisements; catalogs of various sorts; lists of headings of newspaper articles. The European Court of Justice also has accepted a broad reading of what collections of information qualify as a database under the Directive.

The eligibility requirement of "substantial investment" in the database has received varied, and arguably conflicting, interpretations in the courts. Examples of databases that courts have satisfied the criterion include a compilation of several thousand real estate listings (Belgium) and the effort to collect and verify the data for the weekly "Top 10" hit chart of music titles (Germany). Examples of databases that have failed to show substantial investment include a website containing information on building construction (Germany) and a listing of newspaper headlines that were deemed to be a "spin-off" of the primary database of news articles (Netherlands). A related area of divergent judgments concerns copying from on-line databases and Internet-related activities such as "hyper linking" or "deep-linking" using search engines.

Related differences of opinion have emerged with respect to what constitutes an extraction or reutilization of a "substantial part" of the database, particularly with regard to indexing of on-line resources such as news sites by search engines.

Disputes over sports-related data were referred to the European Court of Justice in 2004 to clarify the scope of the Directive. The ECJ rejected claims by sports bodies that organize contests to ownership over the scores or results and other data related to these contests.

Cross-references

- ▶ Copyright Issues in Databases
- ▶ Licensing and Contracting Issues in Databases

Recommended Reading

1. Comments on future of the Directive. http://circa.europa.eu/Public/irc/markt/markt_consultations/library?l=/copyright_neighbouring/database_consultation&vm=detailed&sb=Title
2. Commission of the European Communities, First evaluation of Directive 96/9/EC on the legal protection of databases, 12 Dec. 2005, at http://ec.europa.eu/internal_market/copyright/docs/databases/evaluation_report_en.pdf
3. Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, OJ L 77, 27.3.1996, pp. 20–28. http://eur-lex.europa.eu/smartapi/cgi/sga_doc?smartapi!celexapi!prod!CELEXnumdoc&lg=en&numdoc=31996L0009&model= guichett
4. Gervais, D.J. The Protection of Databases (2007): Chicago-Kent Law Review, 82:1109–1168, 2007 available at http://works.bepress.com/cgi/viewcontent.cgi?article=1011&context=daniel_gervais.
5. Judgments of the European Court of Justice in respect of the Database Directive, http://ec.europa.eu/internal_market/copyright/prot-databases/jurisprudence_en.htm
6. Maurer, S.M. Across Two Worlds: Database Protection in the United States and Europe, In Intellectual Property and Innovation in the Knowledge–Based Economy, 2001. http://strategis.ic.gc.ca/epic/site/ippd-dppi.nsf/vwapj/13-EN2%20Maurer.pdf/$file/13-EN2%20Maurer.pdf
7. Maurer, S.M., Hugenholtz, B.P., and Onsrud, Harlan J. Europe's Database Experiment, Science 2001, 294(5543): 789–790, 2001.
8. Overview of official documents. http://ec.europa.eu/internal_market/copyright/prot-databases/prot-databases_en.htm
9. Study on the Implementation and Application of Directive 96/9/EC on the Legal Protection of Databases conducted by NautaDulith. http://ec.europa.eu/internal_market/copyright/docs/databases/etd2001b53001e72_en.pdf

Evaluation Forum

▶ INitiative for the Evaluation of XML retrieval (INEX)

Evaluation in Information Retrieval

▶ Standard Effectiveness Measures

Evaluation Measures

▶ Search Engine Metrics

Evaluation Metrics for Structured Text Retrieval

Jovan Pehcevski[1], Benjamin Piwowarski[2]
[1]INRIA Paris-Rocquencourt, Le Chesnay Cedex, France
[2]University of Glasgow, Glasgow, UK

Synonyms

Performance metrics; Evaluation of XML retrieval effectiveness

Definition

An *evaluation metric* is used to evaluate the effectiveness of information retrieval systems and to justify theoretical and/or pragmatic developments of these systems. It consists of a set of measures that follow a common underlying evaluation methodology.

There are many metrics that can be used to evaluate the effectiveness of structured text retrieval systems. These metrics are based on different evaluation assumptions, incorporate different hypotheses of the expected user behavior, and implement their own evaluation methodologies to handle the level of overlap among the units of retrieval.

Historical Background

Over the past 5 years, the initiative for the evaluation of XML retrieval (INEX) has investigated various aspects of structured text retrieval, by particularly focusing on XML retrieval. Major advances, both in terms of approaches to XML retrieval and evaluation of XML retrieval, have been made in the context of INEX. The focus of this entry is on evaluation metrics for XML retrieval, as evaluation metrics for structured text retrieval have thus far been proposed in the context of XML retrieval.

Compared to traditional information retrieval, where whole documents are the retrievable units, information retrieval from XML documents creates additional evaluation challenges. By exploiting the logical document structure, XML allows for more focused retrieval by identifying information units (or XML elements) as answers to user queries. Due to the underlying XML hierarchy, in addition to finding the most specific elements that at the same time exhaustively cover the user's information need, an XML retrieval system needs to also determine the appropriate level of answer granularity to return to the user. The *overlap problem* of having multiple nested elements, each containing identical textual information, can have a huge impact on XML retrieval evaluation [6].

Traditional information retrieval evaluation measures (such as recall and precision) mostly assume that the relevance of an information unit (e.g., a document) is binary and independent of the relevance of other information units, and that the user has access to only one information unit at a time. Furthermore, they also assume that the information units are approximately equally sized.

These two assumptions do not hold in XML retrieval, where the information units are nested elements of very different sizes. As nested elements share parts of the same information, an evaluation metric for XML retrieval can no longer assume than the relevance of elements is independent. Moreover, since users can access different parts of an XML document, it also can no longer be assumed that they will have access to only one element at a time. Each of the evaluation metrics for XML retrieval supports the above assumptions to a different extent.

Another limitation of the traditional information retrieval metrics is that they are not adapted to the evaluation of specific retrieval tasks, which could use more advanced ways of presenting results that arise naturally when dealing with XML documents. For example, one task in XML retrieval is to present the retrieved elements by their containing documents, allowing for an easier identification of the relevant information within each document.

INEX has been used as an arena to investigate the behavior of a variety of evaluation metrics for XML retrieval. Most of them are extensions of traditional information retrieval metrics, namely precision-recall and cumulated gain. Precision-recall is a bi-dimensional metric that captures the concentration and the number of relevant documents retrieved by an information retrieval system. An alternative definition of this metric calculates precision at a given recall level ℓ (between 0 and 100%) as the probability that a retrieved document is relevant, provided that a user wants to see ℓ percent of the relevant documents that exist for the topic of interest [12]. The precision-recall metric was the first one extended for the purposes of XML retrieval evaluation.

The cumulated gain (CG) metric [2] relies on the idea that each retrieved document corresponds to a gain for the user, where the gain is a value between 0 and 1. The metric then simply computes the CG at a given rank k as a sum of the gains for the documents retrieved between the first rank and rank k. When normalized, the CG value is somewhat similar to recall, and it is also possible to construct an equivalent of precision for CG. The importance of extending this metric for XML retrieval lies in the fact that it allows for non-binary relevance, which means it can capture elements of varying sizes and granularity.

Foundations

A common notation is used throughout this document to describe the formulae of the different evaluation metrics for XML retrieval. The notation is presented in Table 1. It is also assumed that any XML element can be represented as a textual segment that spans the text corresponding to that XML element. This conceptual representation is practical, as it is possible to define the intersection, the union, the inclusion, and the size of any two segments.

Evaluation Metrics for Structured Text Retrieval. Table 1. Common notations used to describe formulae of XML retrieval metrics

Notation	Short description
e	An XML element
e_i	The *i*th XML element in the list
$spe(e)$	The (normalized) specificity
$exh(e)$	The (normalized) exhaustivity
$q(e)$	A quantization function
$\mathfrak{I}$	The set of ideal elements
$\mathcal{L}$	The ideal ranked list of elements
ℓ	Arbitrary recall level
size(e)	The size of element *e*, usually in number of characters
overlap(i)	The level of overlap between the *i*th element of the list and the previously returned elements

Evaluation Concepts

In XML retrieval, the commonly used ad hoc retrieval task simulates how a digital library is typically used, where information residing in a static set of XML documents is retrieved using a new set of topics. Different sub-tasks can be distinguished within the broad ad hoc retrieval task.

XML Retrieval Tasks

The main XML retrieval tasks, considered to be subtasks of the main INEX ad hoc retrieval task, are:

- Thorough, where XML retrieval systems are required to estimate the relevance of a retrieved element and return a ranked list of all the overlapping relevant elements.
- Focused, where the returned ranked list consists of non-overlapping relevant elements.
- Relevant in context (RiC), where systems are required to return a ranked list of relevant articles, where for each article a set of non-overlapping relevant elements needs to be correctly identified.
- Best in context (BiC), where the systems are required to return a ranked list of relevant articles, where for each article the best entry point for starting to read the relevant information within the article needs to be correctly identified.

User Behavior

The evaluation metrics typically model a sequential user browsing behavior: given a ranked list of answer elements, users start from the beginning of the list and inspect one element at a time, until either all the elements in the list have been inspected, or users had stopped inspecting the list since their information needs were fully satisfied. However, while inspecting a ranked list of elements, users of an XML retrieval system could also have access to other structurally related elements, or indeed could be able to inspect the *context* where the answer elements reside (which may be supported by features such as browsing, scrolling, or table of contents).

Accordingly, in addition to modeling the sequential user model, the evaluation metrics should also be able to model various user browsing behaviors.

Relevance Dimensions

The *relevance* of a retrieved XML element to a query can be described in many ways. It is therefore necessary to define a relevance scale that can be used by the evaluation metrics. Traditional information retrieval usually uses a binary relevance scale, while in XML retrieval there is a multi-graded (or continuous) relevance scale that uses the following two relevance dimensions:

- Exhaustivity (denoted exh), which shows the extent to which an XML element covers aspects of the information need.
- Specificity (denoted spe), which shows the extent to which an XML element is focused on the information need.

The two relevance dimensions have evolved over the years (readers are referred to the *relevance* definitional entry for more details). For simplicity, it will be assumed that each relevance dimension uses a continuous relevance scale with values between 0 and 1. For example, the four-graded relevance scale used by the two dimensions in INEX from 2002 until 2004 can be mapped onto the values 0, $\frac{1}{3}$, $\frac{2}{3}$ and 1.

The normalized exhaustivity and *specificity* of an XML element e are respectively denoted as $exh(e)$ and $spe(e)$. They can take values between 0 and 1.

Quantization

Quantization is the process of transforming the values obtained from the two relevance dimensions into a single normalized relevance score (which again takes values between 0 and 1). It is used to represent the extent to which the retrieved element is relevant.

For example, the strict quantization function can be used to measure the XML retrieval performance when only highly relevant elements are targets of retrieval, while the generalized quantization function can be used to measure the performance when elements with multiple degrees of relevance are targets of retrieval:

$$q_{\text{strict}}(e) = \begin{cases} 1 & \text{if } exh(e) = spe(e) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$q_{\text{gen}}(e) = exh(e) \times spe(e)$$

The strict quantization can therefore be used to reward systems that only retrieve elements that are fully exhaustive and specific, while the generalized quantization rewards systems that retrieve elements with multiple relevance degrees.

Ideality

The concept of ideality emerged in XML retrieval as a concept that is used to distinguish those among all judged relevant elements that users would prefer to see as answers. For example, in order to distinguish between the intrinsic relevance of a paragraph from the inherited relevance of its containing section, it could be said that, even though both elements are relevant, only the paragraph is *ideal*. By definition, an ideal element is always relevant but the reverse is true only in traditional information retrieval.

Ideal elements, unlike relevant elements, can be assumed to be independent. Note that this assumption is similar to the independence of document relevance in traditional information retrieval; that is, ideal elements, as documents, can overlap *conceptually* (they can contain same answers to the underlying information need) as long as they do not overlap *physically*. In XML retrieval, this assumption implies that ideal elements cannot be nested. Note that ideality can be extended to more general units than elements, namely the passages.

Construction of Ideal Sets and Lists

To construct a set $\mathfrak{I}$ of ideal elements, one has to make hypotheses about the underlying retrieval task and the expected user behavior [5].

One example of methodology for identifying the ideal elements is as follows [3]: given any two elements on a relevant path, the element with the higher quantized score is first selected. A *relevant path* is a path in the document tree that starts from the document element and ends with a relevant element that either does not contain other elements, or contains only irrelevant elements. If the two element scores are equal, the one deeper in the tree is chosen. The procedure is applied recursively to all overlapping pairs of elements along a relevant path until only one element remains. It is important that the methodology for identifying ideal elements closely reflects the expected user behavior, since it has been shown that the choice of methodology can have a dramatic impact on XML retrieval evaluation [3].

Given a set of ideal elements, and an evaluation metric that uses that set, it is then possible to construct an ideal list $\mathcal{L}$ of retrieved elements that maximises the metric score at each rank cutoff.

Near Misses and Overlap

Support of near misses is an important aspect that needs to be considered by the evaluation metrics for XML retrieval. *Near misses* are elements close to an ideal element, which act as entry points leading to one or more ideal elements. It is generally admitted that systems that retrieve near misses should be rewarded by the evaluation metrics, but to a lesser extent than when ideal elements are retrieved [4].

Early attempts that extended the traditional information retrieval metrics to support XML retrieval rewarded near misses by assigning partial scores to the elements nearby an ideal one [1,6]. However, this implies that systems that return only ideal elements will never achieve a 100% recall, since both ideal elements and near misses have to be returned to achieve this level of recall [10].

Moreover, these metric extensions are commonly considered to be “overlap positive” [14], which means that they reward systems for retrieving twice the same ideal element, either directly or indirectly, and that the total reward for retrieving that ideal element increases with the number of times it is retrieved. To cater for this problem, overlap neutral and/or negative evaluation metrics have since been developed [1,5].

It is therefore important to be able to compute the degree of overlap between an element e_i and other elements previously retrieved in the ranked list $(e_1,\ldots,e_i)$. A commonly adopted measure is the percentage of text in common between the element and the other previously retrieved elements:

$$\mathrm{overlap}(i) = \frac{\mathrm{size}\left(e_i \cap \bigcup_{j=1}^{i-1} e_j\right)}{\mathrm{size}(e_{\mathrm{i}})} \qquad (1)$$

The overlap function equals 0 when there is no overlap between the element e_i and any of the previously retrieved elements e_j, and equals 1 when there is full overlap (i.e., either all its descendants or one of its ancestors have been retrieved). A value between 0 and 1 denotes intermediate possibilities.

Metric Properties

An evaluation metric for XML retrieval should provide a support for the following properties.

- *Faithfulness*. The metric should measure what it is supposed to measure (*fidelity*) and it should be *reliable* enough so that its evaluation results can be trusted.
- *Interpretation*. The outcome of the evaluation metric should be easy to interpret.

- *Recall/precision*. The metric should capture both *recall* and *precision*, as they are complementary dimensions whose importance have been recognized in traditional information retrieval (some retrieval tasks put more focus on recall while others prefer precision).
- *Ideality*. The metric should support the notion of ideal elements.
- *Near misses*. The metric should be able to properly handle near misses.
- *Overlap*. The metric should properly handle the overlap among retrieved and judged elements.
- *Ideality graded scale*. The metric should be able to support multi-graded or continuous scales, in order to distinguish the ideality of two elements.
- *User models and retrieval tasks*. The metric should be able to model different *user behaviors* and support different *retrieval tasks*, since XML retrieval systems support a variety of features that allow information access.

Table 2 summarizes the above metric properties and provides an overview of the extent to which each of the evaluation metrics for XML retrieval (described in the next section) provides a support for them.

Evaluation Metrics

This section presents the different evaluation metrics that were proposed so far in XML retrieval.

The inex_eval Metric

For 3 years from 2002, the `inex_eval` metric [1] was used as the official INEX metric to evaluate the effectiveness of XML retrieval systems. This metric supports *weak ordering* of elements in the answer list [12], where one or more elements are assigned identical retrieval status values by an XML retrieval system. For simplicity, the discussion is restricted to the case where elements are fully ordered.

The `inex_eval` metric assumes that the degree (or probability) of relevance of an element e is directly given by the quantization function $q(e)$. Its degree

Evaluation Metrics for Structured Text Retrieval. Table 2. Metric properties, and the extent to which each of the XML retrieval metrics supports them. In the table, "y" stands for yes, "n" for no, "i" for indirect, "+" for overlap positive, "–" for overlap negative, and "=" for overlap neutral. The question mark "?" signifies unclear or not demonstrated property

Metric Property	inex_eval	inex_eval_ng	nXCG	ep/gr	T2I	GR	PRUM	EPRUM	HiXEval
Research publication	[1]	[1]	[5]	[5]	[13]	[10]	[11]	[9]	[7]
INEX metric (years)	02–04	03	05–06	05–06				06	07
Faithfulness	?	?	y	y	?	y	y	y	y
Interpretation	n[a]	n[a]	y[b]	y[b]	y	y	y	y	y[b]
Recall	y	y	y	y	y	y	y	y	y
Precision	y	y	n	y	y	y	y	y	y
Near misses	i	i	y	y	y	y	y	y	y
Overlap	+	-	=	=	=	=	=	=	=
Ideality	n	n	y	y	y	y	y	y	y[c]
Ideality graded scale	n/a	n/a	y	y	n	y	n	y	y[d]
Explicit user model	n	n	n	n	y	y	y	y	n
XML Retrieval Tasks									
Thorough	y	y	y	y	?	y	y	y	y
Focused	?	?	y	y	y	y	y	y	y
RiC	i	i	i	i	i	y	y	y	i
BiC	i	i	i	i	i	y	y	y	i

[a] But for some special cases.

[b] With parameter α set to 0 or 1.

[c] Highlighted passages are the ideal units in the case of HiXEval.

[d] The ideality of an element is fixed and directly proportional to the amount of highlighted text.

of non-relevance can be symmetrically defined as $(1 - q(e_i))$. At a given rank k, it is then possible to define the expected number of relevant (resp. non-relevant) $R(k)$ (resp. $I(k)$) elements as follows:

$$R(k) = \sum_{i \le k} q(e_i) \quad I(k) = \sum_{i \le k} (1 - q(e_i))$$

For a given recall level ℓ, the *Precall* [12] metric estimates the probability that a retrieved element is relevant to a topic (assuming that a user wants to find ℓ% of the relevant elements in the collection, or equivalently $\ell \cdot N$ relevant elements). If k_ℓ is the smallest rank k for which $R(k)$ is greater or equal to $\ell \cdot N$, then precision is defined as follows:

$$\text{Precision}(\ell) = \frac{\text{number of seen relevant units}}{\text{expected search length}} = \frac{\ell \cdot N}{k_\ell} \tag{2}$$

where N is assumed to be the expectation of the total number of relevant elements that can be found for an INEX topic, i.e., $N = \sum_e q(e)$ across all the elements of the collection.

Beyond the lack of support for various XML retrieval tasks, the main weakness of the `inex_eval` metric is that one has to choose (with the quantization function) whether the metric should allow near misses or should be overlap neutral – both are not possible. To support overlap, it is possible to compute a set of ideal elements by setting the normalized quantization scores of non-ideal elements to 0, thus not rewarding near misses. To reward near misses, the quantization function should give a non-zero values for elements nearby the ideal elements, but then the system will get fully rewarded only if it returns both the ideal and the other relevant elements. Another problematic issue is the use of non-binary relevance values inside the `inex_eval` formula shown in (2), which makes the metric ill-defined from a theoretical point of view.

The inex_eval_ng Metric

The `inex_eval_ng` metric was proposed as an alternative evaluation metric at INEX 2003 [1]. Here, the two relevance dimensions, *exhaustivity* and *specificity*, are interpreted within an *ideal concept space*, and each of the two dimensions is considered separately while calculating recall and precision scores. There are two variants of this metric, which differ depending on whether overlap among retrieved elements is penalized or not: `inex_eval_ng(o)`, which penalises overlap among retrieved elements; and `inex_eval_ng(s)`, which allows overlap among retrieved elements. Unlike the `inex_eval` metric, this metric directly incorporates element sizes in their relevance definitions.

With `inex_eval_ng(o)`, precision and recall at rank k are calculated as follows:

$$\text{Precision}(k) = \frac{\sum_{i=1}^{k} spe(e_i) \cdot \text{size}(e_i) \cdot (1 - \text{overlap}(i))}{\sum_{i=1}^{k} \text{size}(e_i) \cdot (1 - \text{overlap}(i))}$$

$$\text{Recall}(k) = \frac{\sum_{i=1}^{k} exh(e_i) \cdot (1 - \text{overlap}(i))}{\sum_{i=1}^{\text{N}} exh(e_i)}$$

With `inex_eval_ng(s)`, recall and precision are calculated in the same way as above, except that here the overlap function is replaced by the constant 0 (by which overlap among the retrieved elements is not penalized).

The `inex_eval_ng` metric has an advantage over `inex_eval`, namely the fact that it is possible to penalise overlap. However, due to the fact that it ignores the ideality concept, the metric has been shown to be very unstable if one changes the order of elements in the list, in particular the order of two nested elements [11]. Moreover, `inex_eval_ng` treats the two relevance dimensions in isolation by producing separate evaluation scores, which is of particular concern in evaluation scenarios where combinations of values from the two relevance dimensions are needed to reliably determine the preferable retrieval elements.

The XCG Metrics

In 2005 and 2006, the eXtended cumulated gain (XCG) metrics [5] were adopted as official INEX metrics. The XCG metrics are extensions of the cumulated gain metrics initially used in document retrieval [2].

Gain and Overlap

When the cumulated gain (CG) based metrics are applied to XML retrieval, they follow the assumption that the user will read the whole retrieved element, and not any of its preceding or following elements. An element is partially seen if one or more of its descendants have already been retrieved ($0 < \text{overlap}(i) < 1$), while it is completely seen if any of its ancestors have been retrieved ($\text{overlap}(i) = 1$).

To consider the level of overlap among judged relevant elements, the XCG metrics make use of an ideal set of elements, also known as the *ideal recall base*. To consider the level of overlap among the retrieved elements in the answer list, the XCG metrics implement the following result-list dependent relevance value (or gain) function:

$$\text{gain}(i) = \begin{cases} q(e_i) & \text{if overlap}(i) = 0 \\ (1-\alpha)\cdot q(e_i) & \text{if overlap}(i) = 1 \\ \alpha \cdot \dfrac{\sum_{j/e_j \subseteq e_i} \text{gain}(j)\cdot \text{size}(e_j)}{\text{size}(e_i)} & \text{otherwise} \\ \quad +(1-\alpha)\cdot q(e_i) & \end{cases} \tag{3}$$

The parameter α influences the extent to which the level of overlap among the retrieved elements is considered. For example, with α set to 1 (focused task), the gain function returns 0 for a previously fully seen element, reflecting the fact that an overlapping (and thus redundant) element does not bring any retrieval value in evaluation. Conversely, the level of overlap among the retrieved elements is ignored with α set to 0 (thorough task).

The gain formula cannot guarantee that the sum of the gain values obtained for descendants of an ideal element are smaller than the ideal element gain, and so it is necessary to "normalize" the gain value by forcing an upper gain bound [5].

XCG Metrics

Given a ranked list of elements for an INEX topic, the cumulated gain at rank k, denoted as $XCG(k)$, is computed as the sum of the normalized element gain values up to and including that rank:

$$XCG(k) = \sum_{i=1}^{k} \text{gain}(i) \tag{4}$$

Two XCG metrics used as official XML retrieval metrics at INEX in 2005 and 2006 are nXCG and ep/gr. The nXCG metric is a normalized version of $XCG(k)$, defined as the ratio between the gain values obtained for the evaluated list to the gain values obtained for the ideal list.

The ep/gr metric was defined as an extension of nXCG in order to average performances over runs and to define an equivalent of precision. It consists of two measures: effort-precision *ep*, and gain-recall *gr*.

The gain-recall *gr*, calculated at the rank *k*, is defined as:

$$gr[k] = \frac{XCG[k]}{\sum_{x \in \mathfrak{I}} gain(x)} \tag{5}$$

The effort-precision *ep* is defined as the amount of relative effort (measured as the number of visited ranks) a user is required to spend compared to the effort they could have spent while inspecting an optimal ranking. It is calculated at a cumulated gain level achieved at rank *k* and is defined as:

$$ep[k] = \frac{\min\{i | XCG_{\mathcal{L}}(i) \geq XCG(k)\}}{k} \tag{6}$$

where the indice $\mathcal{L}$ means that the score is evaluated with respect to an ideal list of relevant elements. An *ep* score of 1 reflects an ideal performance, in which case the user made the minimum necessary effort (computed in number of ranks) to reach that particular cumulated gain. An ep/gr curve can then be computed by taking pairs $(gr[k], ep[k])$ for varying rank *k* values.

The XCG metrics have advantages over inex_eval and inex_eval_ng, since its use of an ideal list ensures that the metric is overlap neutral. It also properly handles near misses by the use of an appropriate quantization function. However, the construction of the ideal set of elements relies on heuristics [3]. Other problems of the metric is that the gain is difficult to interpret for values of α other than 0 or 1, which also makes the outcome of the metric somewhat difficult to interpret.

The T2I metric

The tolerance to irrelevance (T2I) metric [13] relies on the same evaluation assumptions as inex_eval, but includes a different user model more suited to XML documents. The underlying user model is based on the intuition that a user processes the retrieved list of elements until their tolerance to irrelevance have been reached (or until they found a relevant element), at which point the user proceeds to the next system result. The T2I metric has only been theoretically proposed, and is yet to be implemented and evaluated.

The (E)PRUM and GR Metrics

The expected precision recall with user modeling (EPRUM) metric [9], which was used as an alternative evaluation metric at INEX in 2005 and as one of the official ones in 2006, extends the traditional definitions of precision and recall to model a variety of user behaviors. EPRUM is unique among all the INEX metrics in that it stochastically defines the user browsing behavior. It is the last defined within a set of three metrics, the previous one being GR (generalized recall) and PRUM (precision recall with user modeling).

The User Model

From a retrieved element, the user can navigate using the corpus structure. The *context* of a list item is defined as the set of elements that can be reached through navigation from it. This includes the pointed elements but also the context of the pointed elements (siblings, ancestors, etc.). To model the user behavior inside the context, the three metrics rely on a set of probabilities on simple events of the form "navigating from a list item to an element in the corpus". The probabilities of navigating from rank j in the list to an element x can be set to values estimated by any adequate method and is denoted $\mathrm{P}(j \rightsquigarrow x)$. When a user is over with this exploration, they *consult* the next entry of the list and repeat the process until their information needs are satisfied. Note that this user model is general enough so as to cope with all INEX tasks, since there is no constraint on how a rank is defined.

Users *see* an element when they navigate to it from another element or from the list, and they *discover* an element if they see it for the first time. The distinction between "seen" and "discovered" is important because the system is rewarded only when elements are discovered. The probability that the user discovers f ideal elements when consulting ranks between 1 and k included is then given by:

$$\mathrm{P}(F_k = f) = \sum_{\substack{A \subseteq \mathfrak{I} \\ |A| = f}} \prod_{x \in A} \mathrm{P}(x \in S_k) \prod_{x \in \mathfrak{I} \setminus A} \mathrm{P}(x \notin S_k) \quad (7)$$

where S_k is the set of all elements seen by the user who has consulted ranks 1 to k. The probability that an element was seen is computed with $P(x \in S_k) = 1 - \prod_{j=1}^{k}(1 - P(j \rightsquigarrow x))$.

GR, PRUM and EPRUM

The GR metric is a generalization of recall with the above specified user model. It simply estimates the expected number of discovered elements at a given rank k, and divides it by the expected number of ideal elements in the database in order to get a normalized value. PRUM is defined as the probability that a consulted list item leads the user to discover an ideal element. Its most important limitation is that it does not handle well non-binary assessments.

EPRUM defines precision based on the comparison of two minimum values: the minimum rank that achieves the specified recall over *all* the possible lists and over the *evaluated* list. For a given recall level ℓ, precision is thus defined as the percentage of effort (in minimum number of consulted ranks) a user would have to make when consulting an ideal list with respect to the effort when consulting the evaluated list:

$$\begin{aligned}&\mathrm{Precision}(\ell)\\&= \mathbb{E}\left[\frac{\begin{array}{c}\text{Minimum number of consulted}\\ \text{list items for achieving a recall } \ell \text{ over all lists}\end{array}}{\begin{array}{c}\text{Minimum number of consulted list items}\\ \text{for achieving a recall } \ell \text{ over the evaluated list}\end{array}}\right]\end{aligned}$$

where the assumption is that when the user cannot achieve recall ℓ in the evaluated list, then the minimum number of consulted list items for the evaluated list is infinite (this assumption is the same as in traditional information retrieval). This measure is an extension of the standard precision-recall metric.

It is similarly possible to extend the traditional definition of precision at a given rank k. If the expected recall of the evaluated list at rank k is r_k, then precision at rank k is defined as the ratio of the minimum number of consulted list items over all possible lists to achieve recall r_k to the number of consulted ranks k.

The EPRUM metric solves some problems of PRUM and substantially reduces its complexity. It also allows proper handling of graded ideality. The advantages of EPRUM metric are the fact that it handles all the INEX tasks through its user model parameters, that the user model is very flexible (for example allowing to reward near misses that are not direct ancestors or descendant of an ideal element), and that the outcome of the metric can easily be interpreted. However, like the XCG metrics, it assumes that the ideal set of elements and the ideal list of retrieved elements can easily be determined, which is shown to be not as straightforward in XML retrieval [3].

The HiXEval Metric

Since 2005, a highlighting assessment procedure is used at INEX to gather relevance assessments for the XML retrieval topics. In this procedure, assessors from the participating groups are asked to highlight sentences representing the relevant information in a pooled set of retrieved documents. To measure the extent to which an XML retrieval system returns relevant information, INEX started to employ evaluation metrics based on the `HiXEval` metric [7,8]. This is motivated by the need to directly exploit the INEX highlighting assessment procedure, and it also leads to evaluation metrics that are natural extensions of the well-established metrics used in traditional information retrieval.

`HiXEval` only considers the specificity relevance dimension, and it credits systems for retrieving elements that contain as much highlighted (relevant) text as possible, without also containing a substantial amount of non-relevant text. So, instead of counting the number of relevant elements retrieved, `HiXEval` measures the amount of relevant text retrieved. Like the XCG metrics, it makes the assumption that the user will read the whole retrieved element, and not any of its preceding or following elements. An element is partially seen by the user if one or more of its descendants have already been retrieved, while it is completely seen if any of its ancestors have been retrieved.

Let rsize(e_i) be the amount of highlighted (relevant) text contained by an element e retrieved at rank i, so that if there is no highlighted text in the element, rsize(e_i) = 0. (Note that rsize(e_i) can also be represented as: rsize(e_i) = $spe(e_i)\cdot$ size(e_i).) To measure the value of retrieving relevant text from e_i, the relevance value function rval(i) is defined as follows:

$$\mathrm{rval}(i) = \begin{cases} \mathrm{rsize}(e_i) & \text{if overlap}(i) = 0 \\ (1-\alpha)\cdot \mathrm{rsize}(e_i) & \text{if overlap}(i) = 1 \\ \mathrm{rsize}(e_i) - \alpha . \sum\limits_{j/e_j \subseteq e_i} \mathrm{rval}(j) & \text{otherwise} \end{cases} \tag{8}$$

As with the XCG metrics, the parameter α is a weighting factor that represents the importance of retrieving non-overlapping elements in the ranked list.

Precision and recall at a rank k are defined as follows:

$$\mathrm{Precision}(k) = \frac{\sum\limits_{i=1}^{k} \mathrm{rval}(i)}{\sum\limits_{i=1}^{k} \mathrm{size}(e_i)}$$

$$\mathrm{Recall}(k) = \frac{1}{Trel} \cdot \sum_{i=1}^{k} \mathrm{rval}(i)$$

In the above equation, *Trel* represents the total amount of highlighted relevant text for an INEX topic. Depending on the XML retrieval task, different *Trel* values are used by the metric. For example, for the focused task *Trel* is the total number of highlighted characters across all *documents*. This means that the total amount of highlighted relevant text for the topic represents the sum of the sizes of the (non-overlapping) highlighted passages contained by all the relevant documents. Conversely, for the thorough task *Trel* is the total number of highlighted characters across all *elements*. For this task, the total amount of highlighted relevant text for the topic represents the sum of the sizes of the (overlapping) highlighted passages contained by all the relevant elements.

The precision and recall scores can be combined in a single score using the standard F-measure (their harmonic mean). By comparing the F-measure scores obtained from different XML retrieval systems, it would be possible to see which system is more capable of retrieving as much relevant information as possible, without also retrieving a substantial amount of non-relevant information.

HiXEval has the advantage of using a naturally defined ideal unit, namely a highlighted passage, and thus overcomes the problem of defining a set of ideal elements in an arbitrary way. One shortcoming of this metric is that it makes the assumption that the degree of ideality of a passage is directly proportional to the passage size. It also shares the same issue identified with the XCG metrics that, with α values different from 0 or 1, the interpretation of the output of the rval (i) function is not very straightforward.

Key Applications

Web Search

Due to the increasing adoption of XML on the World Wide Web, information retrieval from XML document collections has the potential to be used in many Web application scenarios. Accurate and reliable evaluation of XML retrieval effectiveness is very important for improving the usability of Web search, especially if the evaluation captures the extent to which XML retrieval can be adapted to a particular retrieval task or a user model. This could certainly justify the increasing usage of XML in the ever-growing number of interactive Web search systems.

Digital Libraries

Reliable evaluation of XML retrieval effectiveness is also important for improving information retrieval from digital libraries, especially since there is a large amount of structured (XML) information that is increasingly stored in modern digital libraries.

URL to Code

EvalJ project: http://evalj.sourceforge.net

Cross-references

▶ INitiative for the Evaluation of XML Retrieval
▶ XML Retrieval

Recommended Reading

1. Gövert N., Fuhr N., Lalmas M., and Kazai G. Evaluating the effectiveness of content-oriented XML retrieval methods. Inform. Ret., 9(6):699–722, 2006.
2. Järvelin K. and Kekäläinen J. Cumulated gain-based evaluation of IR techniques. ACM Trans. Inform. Syst., 20(4):422–446, 2002.
3. Kazai G. Choosing an ideal recall-base for the evaluation of the Focused task: Sensitivity analysis of the XCG evaluation measures. In Proc. Comparative Evaluation of XML Information Retrieval Systems: Fifth Workshop of the INitiative for the Evaluation of XML Retrieval, 2007, pp. 35–44.
4. Kazai G. and Lalmas M. Notes on what to measure in INEX. In Proc. INEX 2005 Workshop on Element Retrieval Methodology, 2005, pp. 22–38.
5. Kazai G. and Lalmas M. eXtended Cumulated Gain measures for the evaluation of content-oriented XML retrieval. ACM Trans. Inform. Syst., 24(4):503–542, 2006.
6. Kazai G., Lalmas M., and de Vries A.P. The overlap problem in content-oriented XML retrieval evaluation. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 72–79.
7. Pehcevski J. Evaluation of Effective XML Information Retrieval. Ph.D. thesis, RMIT University, Melbourne, Australia, 2006.
8. Pehcevski J. and Thom J.A. HiXEval: Highlighting XML retrieval evaluation. In Advances in XML Information Retrieval and Evaluation: Proc. Fourth Workshop of the INitiative for the Evaluation of XML Retrieval, 2006, pp. 43–57.
9. Piwowarski B. and Dupret G. Evaluation in (XML) information retrieval: Expected Precision-recall with user modelling (EPRUM). In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 260–267.
10. Piwowarski B. and Gallinari P. Expected Ratio of Relevant Units: A Measure for Structured Information Retrieval. In INEX 2003 Workshop Proceedings, 2003, pp. 158–166.
11. Piwowarski B., Gallinari P., and Dupret G. Precision recall with user modelling (PRUM): Application to structured information retrieval. ACM Trans. Inform. Syst., 25(1):1–37, 2007.
12. Raghavan V., Bollmann P., and Jung G. A critical investigation of recall and precision. ACM Trans. Inform. Syst., 7(3): 205–229, 1989.
13. de Vries A., Kazai G., and Lalmas M. Tolerance to Irrelevance: A User-effort Evaluation of Retrieval Systems without Predefined Retrieval Unit. In Proceedings of RIAO 2004, 2004, pp. 463–473.
14. Woodley A. and Geva S. XCG Overlap at INEX 2004. In INEX 2005 Workshop Pre-Proceedings. 2005, pp. 25–39.

Evaluation of Fuzzy Queries Over Multimedia Systems

▶ Top-k Selection Queries on Multimedia Datasets

Evaluation of Relational Operators

Jingren Zhou
Microsoft Research, Redmond, WA, USA

Synonyms

Query evaluation; Query processing

Definition

A query usually consists of several relational operators. The corresponding physical query plan is composed of several physical operators. Generally speaking, a physical operator is an implementation of a relational operator. For each relational operator, there are several alternative algorithms (physical operators) for its implementation and there is no universally superior one. So choosing physical operators wisely is crucial for query performance. Which physical operator is the best depends on several factors, such as the sizes of the input relations, the existing sorting orders of the input relations, the existing indexes/materialized views, the buffer replacement policy, and the size of the available buffer pool, etc.

Historical Background

The relational operators serve as the building blocks for query processing. Their implementation techniques have been extensively studied ever since the first relational DBMS was built.

Foundations

Efficient evaluation of relational operators is required to provide good query performance. There are usually different evaluation algorithms for a given relational operator. Each evaluation algorithm is implemented by a physical operator. There can be more than one physical query plan for a query. The query optimizer considers various physical query plans and chooses the cheapest plan in a cost-based fashion.

Pipelined Query Execution

Database systems usually employ a demand-pull query execution model. Each query plan consists of a pipeline of physical operators. In this model, each operator supports a group of three functions that allows a parent operator to get the result *one* tuple at a time.

1. *Open*. The open() function initializes the state of the iterator by allocating buffers for its inputs and output, and is also used to pass in arguments such as selection predicates that modify the behavior of the operator.
2. *Next*. The next() function calls the next() function on each input node recursively and processes the input tuples until one output tuple is generated. The state of the operator is updated to keep track of how much input has been consumed.
3. *Close*. When all output tuples have been produced through repeated calls to the next() function, the close() function deallocates the state information and performs final housekeeping.

The iterator interface supports pipelining of results naturally. The decision to pipeline or materialize input tuples is encapsulated in the operator-specific code that processes input tuples. Pipelining requires much less space for intermediate results during execution. The demand-pull pipelined query execution model can be executed by a single process or thread. This approach has several advantages such as avoiding inter-process communication between operators, avoiding process synchronization and scheduling, minimizing data copies, keeping only the current data items in memory, and performing lazy operator evaluation.

Selection

Given a selection of the form $\sigma_{R.a\ \theta\ c}(R)$, the algorithms for selection use either scanning or indexing. The condition $R.a\ \theta\ c$ is also called the selection predicate. See Fig. 1.

Selection Based on Scanning

If there is no secondary index on $R.a$ and the relation R is not sorted on $R.a$, the only choice of evaluation is to scan the entire relation. For each tuple, the predicate is evaluated and the tuple is added to the result if the predicate is satisfied.

Selection Based on Indexing

If there is no secondary index on $R.a$ but relation R is sorted on $R.a$, relation R is said to be clustered on $R.a$ and can be viewed as a clustered index itself. A binary search can be used to locate the first tuple that satisfies the selection predicate. If the predicate is a range predicate, for example, $R.a > 1$, the rest of the tuples can be retrieved by scanning R from this location until the predicate is no longer satisfied.

If there is a secondary index (usually a B^+ tree) on $R.a$, an index lookup can be performed to locate the first index entry pointing to a qualifying tuple. Then the leaf pages of the index are scanned to retrieve all entries with the key value satisfying the predicate. Following each of these entries, the corresponding R tuple is then retrieved and added to the result.

If an index contains all the columns that are required by the query, the final R tuple retrieval can be skipped. In this scenario, the selection is done by an index only operation.

In summary, which strategy is the best depends on the index availability, whether the index is clustered or nonclustered, and the selectivity of the selection predicate.

Projection

In general, the projection is of the form $\pi_{a,\ b,\ c}(R)$. Some projected columns may be computed from attributes in R, for example, $\pi_{R.a\ +\ R.\ b,\ R.\ c}(R)$. See Fig. 2. To implement projection, unwanted attributes (i.e.,

```
SELECT *
FROM R
WHERE R.a > 1
```

Evaluation of Relational Operators. Figure 1. Selection query.

those not specified in the projection) are removed and expressions, if any, are computed on the fly.

Joins

The join operation, $R \bowtie_{R.\ r\ \theta\ S.\ s} S$, is one of the most useful relational operations and is the primary means of combining information from more than one relation. The condition $R.r\theta S.s$ is called the join predicate.

A join operation can always be implemented as a cross-product followed by selections and projections. See Fig. 3. However, a cross-product generates much larger results and thus can be quite inefficient. It is very important to recognize joins and implement them without materialized the underlying cross-product.

Join operations have received extensive research over years. There are several alternative techniques for implementing joins. Choosing the right join operation and the correct join order plays a major role in finding an optimal physical plan. The decision depends on various factors, such as the size of input relations, the amount of available memory, and available alternative access methods, etc.

Nested Loops Join

Nested loops join is the simplest join algorithm. The algorithm starts with reading the *outer* relation R, and for each tuple $\mathcal{R} \in R$, the *inner* relation S is checked and matching tuples are added to the result.

```
Algorithm 1: Nested Loops Join: $R \bowtie_{pred\ (r,s)} S$
foreach $\mathcal{R} \in R$ do
    foreach $S \in S$ do
        if pred ($\mathcal{R}$.r, S.s) then
            add {$\mathcal{R}$, S} to result
        end
    end
end
```

```
SELECT R.a + R.b, R.c
FROM R
```

Evaluation of Relational Operators. Figure 2. Projection query.

```
SELECT *
FROM R, S
WHERE R.r = S.s
```

Evaluation of Relational Operators. Figure 3. Join query.

One advantage of nested loops join is that it can handle any kind of join predicate, unlike sort-merge join and hash join which mainly deal with an equality join predicate. In order to effectively utilize buffer pages and available indexes, there are two variations of nested loops join.

Block Nested Loops Join Suppose that the memory can hold B buffer pages. If there is enough memory to hold the smaller relation, say R, with at least two extra buffer pages left, the optimal approach is to read in the smaller relation R and to use one extra page as an input buffer to read in the larger relation S and one extra buffer page as an output buffer.

If there is not enough memory to hold the smaller relation, the best approach is to break the outer relation R into *blocks* of $B - 2$ pages each and scan the whole inner relation S for each block of R. As described before, one extra page is used as an input buffer and the other as an output buffer. In this case, the outer relation R is scanned only once while the inner relation S is scanned multiple times.

Index Nested Loops Join If one of the two relations has an index on the join attribute, an index nested loops join can be used to take advantage of the index. Suppose the relation S has a suitable index on the join attribute $S.s$. For each tuple $\mathcal{R} \in R$, the index can be used to retrieve all matching tuples of S. If the number of matching S tuples for each R tuple is small, index lookup is much more efficient than scanning the whole S relation. However, index lookup usually involves random I/Os. If the number of matching S tuples is large, it might be worthwhile considering block nested loops join.

Sort-Merge Join

If the join predicate is an equality predicate, sort-merge join can be used. The basic idea of the sort-merge join algorithm is to *sort* both relations on the join attribute and to then *merge* the sorted relations by scanning them sequentially and looking for qualifying tuples.

The sorting step groups all tuples with the same value in the join attribute together. Such groups are sorted based on the value in the join attribute so that it is easy to locate groups from the two relations with the same attribute value. Sorting operation can be fairly expensive. If the size of the relation is larger than the available memory, external sorting algorithm

```
Algorithm 2: Sort-Merge Join: $R \bowtie_{R.r=S.s} S$
// sorting step
Sort the relation R on the attribute r;
Sort the relation S on the attribute s;
// merging step
$\mathcal{R}$ = first tuple in R;
S = first tuple in S;
S' = first tuple in S;
while $\mathcal{R} \neq eof$ and $S' \neq eof$ do
    while $\mathcal{R}.r < S'.s$ do
        $\mathcal{R}$ = next tuple in R after $\mathcal{R}$
    end
    while $\mathcal{R}.r > S'.s$ do
        S' = next tuple in S after S'
    end
    S = S';
    while $\mathcal{R}$.r == S'.s do
        S = S'
        while $\mathcal{R}$.r == S.s do
            add {$\mathcal{R}$,S} to result;
            S = next tuple in S after S;
        end
        $\mathcal{R}$ = next tuple in R after $\mathcal{R}$;
    end
    S' = S;
end
```

is required. However, if one input relation is already clustered (sorted) on the join attribute, sorting can be completely avoided. That is why sort-merge join looks even more attractive if any of the input relations is sorted on the join attribute.

The merging step starts with scanning the relations *R* and *S* and looking for matching groups from the two relations with the same attribute value. The two scans start at the first tuple in each relation. The algorithm advances the scan of *R* as long as the current *R* tuple has an attribute value which is less than that of the current *S* tuple. Similarly, the algorithm advances the scan of *S* as long as the current *S* tuple has an attribute value which is less than that of the current *R* tuple. The algorithm alternates between such advances until an *R* tuple $\mathcal{R}$ and an *S* tuple $\mathcal{S}$ with $\mathcal{R}.r = \mathcal{S}.s$. The join tuple $\{\mathcal{R}, \mathcal{S}\}$ is added to result.

In fact, there could be several *R* tuples and several *S* tuples with the same attribute value as the current tuples $\mathcal{R}$ and $\mathcal{S}$. That is, several *R* tuples may belong to the current $\mathcal{R}$ group since they all have the same attribute value. The same applies to the current $\mathcal{S}$ group. Every tuple in the current $\mathcal{R}$ group joins with every tuple in the current $\mathcal{S}$ group. The algorithm then resumes scanning *R* and *S*, beginning with the first tuples that follow the group of tuples that are just processed.

When the two relations are too large to be held in available memory, one improvement is to combine the merging step of external sorting with the merging step of the join if the number of buffers available is larger than the total number of sorted runs for both *R* and *S*. The idea is to allocate one buffer page for each run of *R* and one for each run of *S*. The algorithm merges the runs of *R*, merges the runs of *S*, and joins (merges) the resulting *R* and *S* streams as they are generated.

If any of the two relations has an index on the join attribute, another improvement is to merge join the index instead of the relation. Sorting, in this case, can be avoided. After the join, it may be necessary to retrieve other attributes from the relation following qualifying index entries.

Hash Join

The hash join algorithm is commonly used in database systems to implement equijoins efficiently. In its simplest form, the algorithm consists of two phases: the "build" phase and the "probe" phase. In the "build" phase, the algorithm builds a hash table on the smaller relation, say *R*. In the "probe" phase, the algorithm probes the hash table using tuples of the larger relation, say *S*, to find matches.

The algorithm is simple, but it requires that the smaller join relation fits into memory. If there is not enough memory to hold all the tuples in *R*, an additional "partition" phase is required. There are several variants of the basic hash join algorithm. They differ in terms of utilizing memory and handling overflow.

Grace Hash Join The idea behind grace hash join is to hash partition both relations on the join attribute, using the *same* hash function. As the result, each relation is hashed into *k* partitions, and these partitions are written to disk. The key observation is that *R* tuples in partition *i* can join only with *S* tuples in the same partition *i*. If any given partition of *R* can be held in memory, the algorithm can read in and build a hash table on the partition of *R*, and then probe the hash table using tuples of the corresponding partition of *S* for matches.

If one or more of the partitions still does not fit into the available memory (for instance, due to data skewness), the algorithm is recursively applied. An additional orthogonal hash function is chosen to hash the large partition into sub-partitions, which are then processed as before.

Hybrid Hash Join The hybrid hash join algorithm is a refinement of the grace hash join algorithm which takes advantage of more available memory. To partition R (S) into k partitions, the grace hash join uses one input buffer for reading in the relation and k output buffers, one for each partitions.

Suppose there is sufficient memory to hold an in-memory hash table for one partition, say the first partition, of R, the hybrid hash join does not write the partition to disk. Instead, it builds an in-memory hash table for the first partition of R during the "partition" phase. Similarly, when partitioning S, for the tuples in the first partition of S, the algorithm directly probes the in-memory hash table and writes out the results. At the end of the "partition" phase, the algorithm completes the join between the first partitions of R and S while partitioning the two relations. The algorithm then joins the remaining partitions as the grace hash join algorithm.

Compared with the grace hash join algorithm, the hybrid hash join algorithm avoids writing the first partitions of R and S to disk during the "partition" phase and reading them in again during the "build" and the "probe" phases.

```
Algorithm 3: Grace Hash Join: $R \bowtie_{r=s} S$
//partition R into k partitions
foreach $\mathcal{R} \in R$ do
    read $\mathcal{R}$ and add it to buffer page $h_1(\mathcal{R})$;
    flush the page to disk it full;
end
//partition S into k partitions
foreach $S \in S$ do
    read $S$ and add it to buffer page $h_1(S)$;
    flush the page to disk it full;
end
//"build" and "probe" phases
for $i \leftarrow 1$ to $k$ do
    foreach $\mathcal{R} \in$ partition $R_i$ do
        read $\mathcal{R}$ and insert into the hash table using $h_2(\mathcal{R})$;
    end
    foreach $S \in$ partition $S_i$ do
        read $S$ and probe the hash table using $h_2(S)$;
        for matching $\mathcal{R}$ tuples, add $\{\mathcal{R}, S\}$ to result;
    end
    clear the hash table and release the memory;
end
```

Aggregation

Simple aggregation operations include MIN, MAX, SUM, COUNT, and AVG. Aggregation operations can also be used in combination with a GROUP BY clause. See Fig. 4. DISTINCT clause that removes duplicate values can be viewed as a special aggregation operation too, which groups the input by values and outputs one row per group.

For aggregation queries without a GROUP BY clause, the algorithm is straightforward: scan the entire relation and update some state information for each scanned tuple. The state information varies for different aggregation functions. For SUM, the algorithm keeps track of the sum of the values retrieved so far. For AVG, the algorithm keeps track of the sum of the values and the total count of rows retrieved so far. For COUNT, the algorithm keeps track of the total count of values retrieved so far. For MAX (MIN), the algorithm keeps track of the largest (smallest) value retrieved so far.

For aggregation queries with a GROUP BY clause, the aggregation algorithms use either sorting or hashing.

Aggregation Based on Sorting

The sorting algorithm starts with sorting the relation on the grouping attributes and then scanning it again to compute the aggregations functions for each group. By sorting, the algorithm groups together all tuples with the same grouping attributes (therefore belong to the same group). The computation of the aggregation functions is similar to the way of aggregation computation without grouping, except that here the algorithm has to watch for group boundaries. When the algorithm sees a tuple with different grouping attributes, the current group finishes and a new group starts.

Aggregation Based on Hashing

The hashing algorithm first builds an in-memory hash table on the grouping attributes as it scans the relation. Each hash entry corresponds to a group and contains the

```
SELECT SUM (R. x)          SELECT SUM (R. x)
FROM R                     FROM R
                           GROUP NY R.r

       (1)                        (2)
```

Evaluation of Relational Operators. Figure 4. Aggregation queries.

state information required by each aggregation function for the group. For each tuple, the algorithm probes the hash table to find the entry for the group to which this tuple belongs and update the state information. If such entry does not exist, the algorithm creates a new hash entry and initiates the state information.

If the relation is so large that the hash table does not fit in memory, the algorithm hash partitions the relation on the grouping attributes. Since all tuples in a given group are in the same partition, the algorithm can then scan each partition independently and compute aggregation functions as described before.

It is also possible and sometimes preferable to compute aggregation using an index as long as the index covers all the attributes required by the aggregation query. The advantage is to read in a much smaller index compared to the entire relation. If the grouping attributes form a prefix of the indexed keys, the algorithm can avoid the sorting step and scan the index entries sequentially.

Key Applications

Each database query is composed of a few relational operators. The final query plan may implement each relation operator in different ways to achieve an optimal performance. All the implementation techniques are widely used in database systems.

Cross-references

▶ Access Methods
▶ Buffer Pool
▶ Concurrency Control
▶ Cost Model
▶ External Sorting
▶ Hashing
▶ Parallel Query Processing
▶ Query Optimization

Recommended Reading

1. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73–170, 1993.
2. Mishra P. and Eich M.H. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, 1992.
3. Ramakrishnan R. and Gehrke J. Database Management Systems. McGraw-Hill, New York, 2002.
4. Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., and Price T.G. Access path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.

Evaluation of XML Retrieval Effectiveness

▶ Evaluation Metrics for Structured Text Retrieval

Event

▶ Time Instant

Event and Pattern Detection over Streams

MINGSHENG HONG, ALAN DEMERS, JOHANNES GEHRKE, MIREK RIEDEWALD
Cornell University, Ithaca, NY, USA

Synonyms

Complex event processing (CEP); Event stream processing (ESP)

Definition

An event is a basic unit of information in streaming data. An event pattern is a combination of events correlated over time. Event pattern detection is an important activity in complex event processing. In this setting, the matches to the event patterns are referred to as complex events.

Historical Background

In the early 1990s, a set of pioneering work in *event systems*, such as SNOOP [3] and ODE [8], set out to define query languages for expressing event patterns. In these proposals, the data model for expressing events is not fixed. More recently, the approaches proposed by Cayuga [1,5,6] and SASE [14] for event pattern detection align more closely to relational query processing, in that each event is modeled by a relational schema, and some of the operators for expressing event pattern queries are drawn from relational algebra. Regardless of the data model for events, these systems all use some variant of NFA (Nondeterministic Finite state Automaton) as the processing model.

With the advent of internet-scale message brokering systems, *content based publish-subscribe systems* such as [7] emerged. They are characterized by very limited query languages, allowing simple selection predicates applied to individual events in a data stream. Such systems trade expressiveness for performance – when well engineered, they exhibit very high scalability in both the number of queries and the stream rate. However, their inability to express queries that span multiple input events makes them unsuitable for event pattern detection.

Another category of systems closely related to event pattern detection is *stream databases*, such as Aurora [2], STREAM [10], and TelegraphCQ [4]. Unlike content based publish-subscribe systems, they focus on expressiveness. Such systems have very powerful query languages, typically including a rich functionality and extending SQL with provisions for sliding window and grouping features. Though powerful, stream databases are not designed for detecting event patterns, and their query languages can be awkward for expressing event pattern queries. Moreover, there is little work on scaling up stream database engines with the number of concurrent queries that are reasonably sophisticated.

Foundations

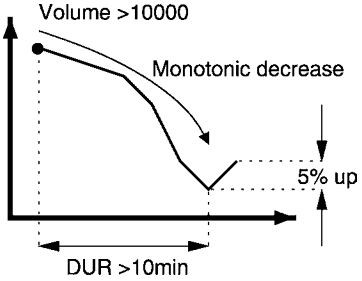
Event Pattern Query Model

An event stream is a potentially infinite sequence of events. Each event has a timestamp value, and its payload content is encoded by a relational tuple. Events are processed in timestamp order by an event processing system. They can be filtered, transformed, and correlated to form event patterns.

The computational model of matching event patterns over event streams naturally extends that of matching regular expression patterns over character streams in the following aspects. First, each stream event can contain multiple attribute-value pairs conforming to a relational schema, where the value domains are potentially infinite. In comparison, each character in a character stream provides for the same single attribute a value drawn from a finite alphabet. Second, given the multiple attributes for each stream event, event patterns can perform relational operations on the events, including filtering, projection and renaming of attributes. This provides expressive power especially to event correlation based on sequencing, where the attribute values of multiple stream events can be compared. Finally, event pattern detection involves a temporal aspect, in that stream events have timestamps, which event patterns reason about. In comparison, regular expression processing only involves character orderings in the streams, which can be viewed as a weaker notion of time.

Event patterns are usually expressed in an event algebra. Many such algebras have been proposed. One representative is the Cayuga algebra [5]. The Cayuga algebra is specifically designed for large-scale event pattern detection. In addition to unary operators for selection predicates and aggregates, the expressive power of this algebra comes from two binary operators. The first binary operator, *sequence*, can correlate two input events based on a join predicate. This join predicate involves timestamps, and can optionally involve other attributes in the stream schema. The second operator, *iteration*, is a generalization of the sequence operator. It is able to produce an event pattern involving arbitrarily many input events by iteratively concatenating input events with the pattern built so far. To allow users to interact with the system in a user-friendly way, a SQL-style query language, referred to as Cayuga Event Language (CEL) [6], has been developed. Figure 1 shows an event pattern query expressed in CEL. This event pattern query

```
SELECT Name, MaxPrice, MinPrice, Price AS FinalPrice
FROM
  FILTER{DUR > 10min}(
    (SELECT Name, Price_1 AS MaxPrice, Price AS MinPrice
    FROM FILTER{Volume > 10000}(Stock))
      FOLD{$2.Name = $.Name, $2.Price < $.Price}
    Stock)
    NEXT{$2.Name = $1.Name AND $2.Price > 1.05*$1.MinPrice}
  Stock
```



Event and Pattern Detection over Streams. Figure 1. Event pattern query to find stock price pattern.

corresponds to a particular trend in stock prices. Intuitively, this query searches for the given price pattern for any company. The pattern starts with a large trade (`Volume > 10,000`), followed by a monotonic decrease in price (`FOLD` clause), which lasts for at least 10 min (`DUR > 10min`). Then the price rebounds with a sudden increase by 5% (`NEXT` clause). The `NEXT` and `FOLD` constructs respectively correspond to the two binary operators in the Cayuga algebra introduced above. `NEXT` matches the next event in the stream that satisfies a given condition (same company name and 5% higher price in the example). `FOLD` is the iterated version of `NEXT`, i.e., it continues matching the next event that satisfies a certain property until a stopping condition is satisfied.

SASE [14] uses a query algebra similar to the Cayuga algebra, where sequence and iteration are the key primitives. SASE also supports *negation*-style event patterns, where a pattern is matched by the absence of an input event, rather than the presence. In addition to online event stream processing supported by Cayuga and SASE, work on sequence database systems has focused on matching event patterns offline over archived data [11,12,13]. Sequence is again the key primitive for expressing event patterns in the SEQ query algebra [13] and the SQL-TS query language [12].

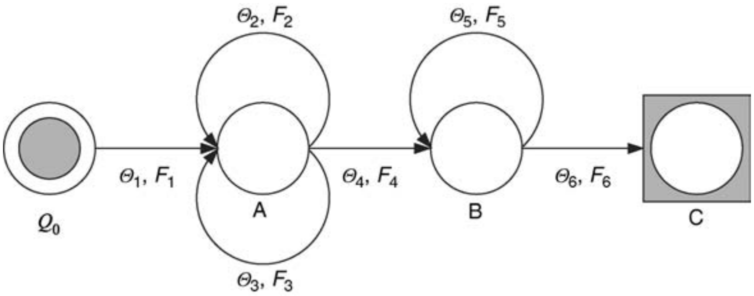
Event Pattern Query Processing

As is mentioned earlier, an event pattern query is typically implemented by a state machine. Continuing the above query example, this approach is described in the context of Cayuga. Each Cayuga automaton is an extension to the classical non-deterministic finite automaton [9] in the following aspects. First, each automaton edge is associated with a predicate, and for an incoming event, this edge is traversed if and only if the predicate is satisfied by this event. This mechanism implements the selection predicates in the event patterns. Second, when patterns are matched, to be able to generate witness events with concrete content instead of boolean answers, each automaton instance needs to store the attributes and values of those events that have contributed to the pattern instance.

Figure 2 shows the automaton for the example query in Fig. 1. The two middle states correspond to the `FOLD` and `NEXT` operators, respectively. The predicates θ_i associated with automaton edges originate either from `FILTER` conditions (θ_1, θ_6 in the example) or from join conditions of the `FOLD` and `NEXT` operators. Specifically, θ_1 implements the filter predicate `vol` $>$ 10,000. When an input stock event *e* satisfying θ_1 occurs, a new automaton instance *I* is created under state *A*, remembering the content of *e*.

Each automaton instance encodes a particular event pattern built from the prefix of the event stream that has been processed so far. When an automaton instance reaches the final state *C*, a match to the entire event pattern specified in the query has been found, and will be output by this automaton.

To continue the explanation of the example Cayuga automaton in Fig. 2, θ_2 and θ_3 respectively implement the two join predicates associated with `FOLD` in the query shown in Fig. 1. These two predicates on the two self-loop edges associated with state *A* together build a monotonically decreasing sequence in the prices of a particular stock. Specifically, after the occurrence of event *e*, when a later stock event e' together with *I* satisfies θ_2; i.e., e' and *I* have the same stock name *s* (say), θ_3 is evaluated to check whether this event pattern built so far can be extended. For this reason, θ_2 serves as a criterion which, when satisfied, concatenates the next input event from `Stock` to the event pattern built up so far, and θ_3 serves as a criterion



Event and Pattern Detection over Streams. Figure 2. Cayuga automaton example (source: [6]).

which, when satisfied, continues the extension of the event pattern being built. The event building process proceeds similarly to state *B* and *C*. The functions F_i are responsible for transforming event stream schemas and automaton state schemas.

Event pattern queries expressed in SASE and SQL-TS are processed in a similar way. In SASE, NFA is one of the run-time operators. Operator re-ordering is performed as part of the query optimization to produce efficient query plans. For example, a selection predicate above an NFA operator can be pushed inside the NFA operator to discard irrelevant input events earlier, thus improving system throughput. In SQL-TS, to optimize pattern search, the query engine exploits the inter-dependencies between the elements of a sequential pattern.

Evaluating one event pattern query efficiently is relatively easy. However, it is challenging to efficiently evaluate a large number of concurrent event patterns. Multi-Query Optimization (MQO) techniques have been developed to share the computation among concurrent event patterns. For example, in Cayuga, the event processing engine achieves this goal by exploiting the relationship of the query algebra to the automata-based query execution, and the commonality among queries. Specifically, each query is first translated into a set of automata. These automata are then "merged" with existing ones in the engine. During the merging process, two optimization techniques are used. First, two automata with the same prefix of states can merge these states, thus sharing computation and storage. This is similar to the technique of finding common subexpressions in relational query processing. Second, some of the filtering predicates on the automaton edges can be managed efficiently by indexes in a way similar to techniques for processing multiple selection operators [7]. These two techniques enable a throughput of thousands of events per second, even for tens of thousands of active event pattern queries.

Key Applications

Event pattern detection targets a large class of both well-established and emerging applications, including supply chain management for RFID (Radio Frequency Identification) tagged products, real-time stock trading, monitoring of large computing systems to detect malfunctioning or attacks, and monitoring of sensor networks, e.g., for surveillance. These *event monitoring applications* need to process massive streams of events in (near) real-time. There is great interest in these applications as indicated by the establishment of sites like http://www.complexevents.com, which bring together major industrial players like BEA, IBM, Oracle, and TIBCO.

Future Directions

One important future direction is to integrate event processing systems with stream databases. In the past they have evolved along different paths, and are designed for different query workloads, as is described in the Historical Background section. It would be beneficial to integrate these two categories of stream processing systems, for two reasons. First, there is much overlap in their functionality. For example, both categories support stateless operations such as filtering and projection, and some forms of stream join. Second, there is a class of stream applications that demand functionality from both categories. There are two major challenges in the integration. At the logical level, a unified query algebra is needed for expressing queries in both categories. At the physical level, it is desirable to evaluate both categories of queries in the same engine. This is a challenging task since current stream database engines, like relational database engines, are usually based on operator trees, while event engines are based on variants of NFAs.

Cross-references

- ▶ Complex Event Processing
- ▶ Continuous Query
- ▶ Data Stream Management Architectures and Prototypes
- ▶ Publish/Subscribe over Streams
- ▶ Stream Processing

Recommended Reading

1. Brenna L., Demers A., Gehrke J., Hong M., Ossher J., Panda B., Riedewald M., Thatte M., and White W. Cayuga: a high-performance event processing engine. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1100–1102.
2. Carney D., Çetintemel U., Cherniack M., Convey C., Lee S., Seidman G., Stonebraker M., Tatbul N., and Zdonik S. Monitoring streams – a new class of data management applications. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 215–226.
3. Chakravarthy S., Krishnaprasad V., Anwar E., and Kim S.K. Composite events for active databases: semantics, contexts and detection. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 606–617.
4. Chandrasekaran S., Cooper O., Deshpande A., Franklin M.J., Hellerstein J.M., Hong W., Krishnamurthy S., Madden S.R., Raman V., Reiss F., and Shah M.A. Telegraph CQ: continuous

dataflow processing for an uncertain world. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
5. Demers A., Gehrke J., Hong M., Riedewald M., and White W. Towards expressive publish/subscribe systems. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 627–644.
6. Demers A., Gehrke J., Panda B., Riedewald M., Sharma V., and White W. Cayuga: a general purpose event monitoring system. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 412–422.
7. Fabret F., Jacobsen H.A., Llirbat F., Pereira J., Ross K.A., and Shasha D. Filtering algorithms and implementation for very fast publish/subscribe. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 115–126.
8. Gehani N.H., Jagadish H.V., and Shmueli O. Composite event specification in active databases: model and implementation. In Proc. 18th Int. Conf. on Very Large Data Bases, 1992, pp. 327–338.
9. Hopcroft J.E., Motwani R., and Ullman J.D. Introduction to automata theory, languages, and computation. Addison-Wesley, Reading, MA, USA, 2nd ed., 2000.
10. Motwani R., Widom J., Arasu A., Babcock B., Babu S., Datar M., Manku G.S., Olston C., Rosenstein J., and Varma R. Query processing, approximation, and resource management in a data stream management system. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
11. Ramakrishnan R., Donjerkovic D., Ranganathan A., Beyer K.S., and Krishnaprasad M. SRQL: sorted relational query language. In Proc. 10th Int. Conf. on Scientific and Statistical Database Management, 1998, pp. 84–95.
12. Sadri R., Zaniolo C., Zarkesh A.M., and Adibi J. Optimization of sequence queries in database systems. In Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2001, pp. 71–81.
13. Seshadri P., Livny M., and Ramakrishnan R. SEQ: a model for sequence databases. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 232–239.
14. Wu E., Diao Y., and Rizvi S. High-performance complex event processing over streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 407–418.

Event Broker

▶ Request Broker

Event Causality

Guy Sharon
IBM Research Labs-Haifa, Haifa, Israel

Definition

The definition of an event processing network includes a relation to express un-modeled event processing logic between events.

The fact that events of type *eT1* cause events of type *eT2* is denoted by the relation *causes(eT1,eT2)*.

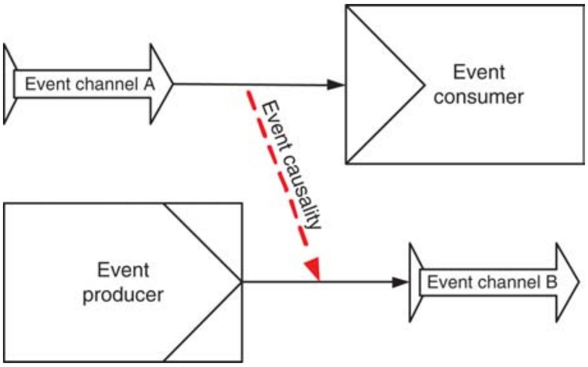
In an EPN where *E* is the set of edges representing event streams, *EC* is the set of Event Channels, *C* is the set of Event Consumers and *P* is the set of Event Producers, the relation is evaluated to be true if events of type *eT1* flow in an event stream *e1(u,v): e1*∈*E, u*∈*EC, v*∈*C* and events of type *eT2* flow in an event stream *e2(m,l): e2*∈*E, m*∈*P, l*∈*EC* and there is some un-modeled event processing logic between events of type *eT1* consumed by *v*∈*e1* and events of type *eT2* produced by *m*∈*e2*.

Key Points

The event processing intent defined by an event processing network may not cover the entire flow of events through systems as there may be cases where an event is handled by an event consumer, such as a software application, and as a result the application publishes, as an event producer, a new event to an event processing network. In essence, there is some un-modeled processing logic performed by the application that results in a new event that may be processed further on. The specifics of this logic are not important for the realization of an event processing network as this logic is implemented and executed by a consumer such as an application, however, having the relation *causes(eT1,eT2)* as part of the model enables to describe the complete event-driven interactions in systems and is used in performing model analysis such as termination analysis [1–3].

In the model, event causality is represented as a red, dashed, and directed edge from the event stream being consumed to the event stream being produced (Fig. 1).

The definition of event causality in an event processing network is meant to be theory neutral and



Event Causality. Figure 1. Event causality.

corresponds to common sense causal interpretation in the physical world. Applying the many theories about causation [4,5] to the cause of an event by another can extend the perception of the event-driven interactions and perhaps most productively enhance analysis capabilities of models, but for the purpose of the realization of an event processing network this is not necessary.

Cross-references

▶ Event Processing Network

Recommended Reading

1. Aiken A., Hellerstein J.M., and Widom J. Static analysis techniques for predicting the behavior of active database rules. ACM Trans. Database Syst., 20(1):3–41, 1995.
2. Bailey J. and Poulovassilis A. Abstract interpretation for termination analysis in functional active databases, J. Intelligent Inf. Syst., 12(2–3):243–273, 1999.
3. Baralis E., Ceri S., and Paraboschi S. Compile-time and runtime analysis of active behaviors, IEEE Trans. Knowledge Data Eng., 10(3):353–370, 1998.
4. Fisk M. A defence of the principle of event causality. Br. J. Philos. Sci., 18(2):89–108, 1967.
5. Pearl J. Causality: Models, Reasoning, and Inference. New York: Cambridge University Press, 2000.

Event Causality Graph

▶ Event Flow

Event Channel

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event connection; Event pathway; Event topic

Definition

An event channel is a routing node in the event processing network that receives events from event sources and event processing agents, and route them to event sinks and event processing agents.

Key Points

An event channel [2] is the main routing vehicle in the event processing network; it is being used to enable loosely coupled architecture, in which the event producer and event consumer does not have any dependency among them [1]. An event channel receives an event (in push or pull) from an event source or from an event processing agent, and makes routing decisions. The routing decision may be of several types:

1. Itinerary based routing: the consumers are explicitly specified within the event content.
2. Subscription based routing: using a "callback" protocol, the consumer subscribes to events on the channel.
3. Intelligent routing: a decision process is activated to decide on the event consumers.
4. Calendar based routing: Routing is controlled by a calendar function.

The routing may be partitioned according to context [3], if context is supported.

Cross-references

▶ Channel-Based Publish/Subscribe
▶ Context
▶ Event Processing Agent
▶ Event Processing Network
▶ Event Sink
▶ Event Source
▶ Event Stream

Recommended Reading

1. Crowcroft, J. Channel Islands in a Reflective Ocean: Large Scale Event Distribution in Heterogeneous networks. In Proc. 2nd Int. IFIP-TC6 Networking Conf., 2002, pp. 1–9.
2. Luckham, D., Schutle, R. (eds). EPTS Event Processing Glossary version 1.1 http://complexevents.com/?p = 409
3. Sharon, G. Etzion, O. Event processing network: Model and implementation. IBM System Journal, 47(2):321–334, 2008.

Event Cloud

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Definition

Event Cloud is a partially ordered set of events (poset), either bounded or unbounded, where the partial orderings are imposed by the causal, timing and other relationships between the events [2].

Key Points

Event cloud [1] consists of a set of events and one or more partial order relations. The following event relations are being used:

- Causality relation: Each collection of events can be considered as an *event cloud*, the causality relation creates a partial order relation among a collection of events, causality is transitive, anti-symmetric, and non-reflexive.
- Precedes relation: defined as event e1 precedes event e2, if it occurs in e1 occurs in reality before event e2. This is a partially order set – since the relative timing may not be known, or may overlap, if occur during an interval.

Event cloud can be defined over the entire collection of events that flow through some system within a certain bounded or unbounded time[3,4]. A special case of event cloud is all events that are input to a single event processing agent and are ordered as a time-series, in this case the event cloud is a collection of event streams.

Cross-references

▶ Complex Event Processing
▶ Event Stream

Recommended Reading

1. Luckham, D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, 2002.
2. Luckham, D., Shulte, R. (eds.). EPTS Event Processing Glossary version 1.1. http://complexevents.com/?p=409
3. Rozsnyai, S., Vecera, R., Schiefer, J., and Schatten, A. Event cloud – searching for correlated business events. In Proc. 9th IEEE Int. Conf. on E-Commerce Technology & 4th IEEE Int. Conf. on Enterprise Comp., E-Commerce and E-Services, 2007, pp. 409–420.
4. Widder, A., von Ammon R., Schaeffer, P., and Wolff, C. Identification of suspicious, unknown event patterns in an event cloud. In Proc. Inaugural Int. Conf. on Distributed Event-based Systems, 2007, pp. 164–170.

Event Composition

▶ Event Detection

Event Composition (Partial Overlap)

▶ Event Pattern Detection

Event Connection

▶ Event Channel

Event Consumer

▶ Event Sink

Event Control

▶ Event Detection

Event Declaration

▶ Event Specification

Event Definition

▶ Event Specification

Event Detection

JONAS MELLIN, MIKAEL BERNDTSSON
University of Skövde, Skövde, Sweden

Synonyms

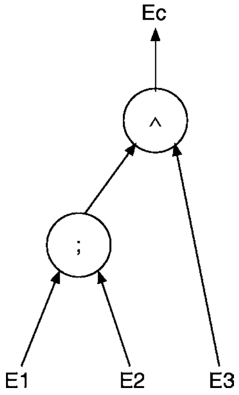
Chronicle recognition; Event trace analysis; Event control; Event composition; Monitoring of real-time logic expressions

Definition

Event detection is the process of analyzing event streams in order to discover sets of events matching patterns of events in an event context. The event patterns and the event contexts define event types. If a set of events matching the pattern of an event type is discovered during the analysis, then subscribers of the event type should be signaled. The analysis typically entails filtering and aggregation of events.

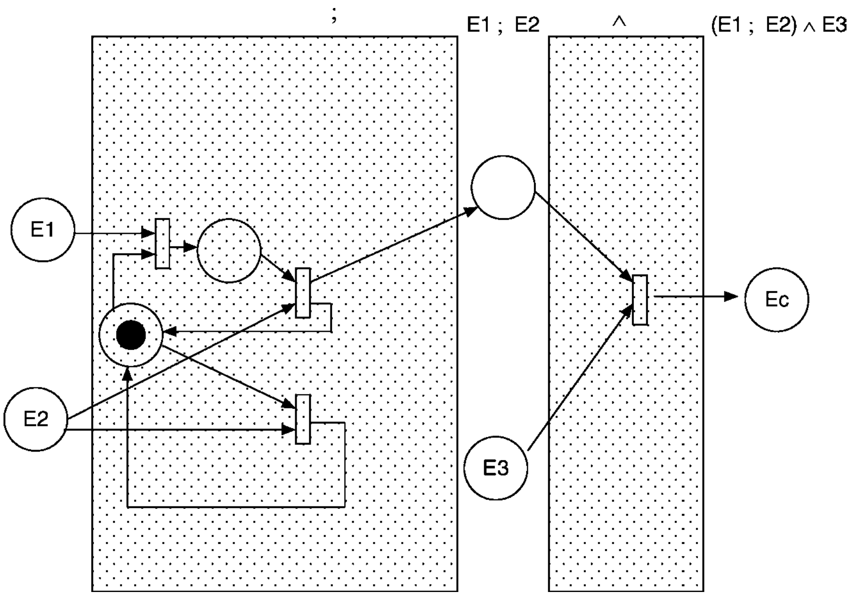
Historical Background

Seminal work on event detection was done in HiPAC [7,11] and Snoop [8,9] as well as in ODE [14] and SAMOS [13]. Essentially, in Snoop, ODE, and SAMOS different methods for realizing the matching of event detection were investigated. In Snoop, implementations of the event operators are structured according to the syntax tree of the event expression, where each node represents an event operator. The event operator in a node is the principal event operator of the event type it corresponds to and each event operator is implemented as a subprogram (e.g., procedure, method). The procedures are invoked during evaluation of the expression. In contrast, in SAMOS [13] operators expressed as Petri-nets are combined; and in ODE [14], finite state automatas built from the event specifications are employed.

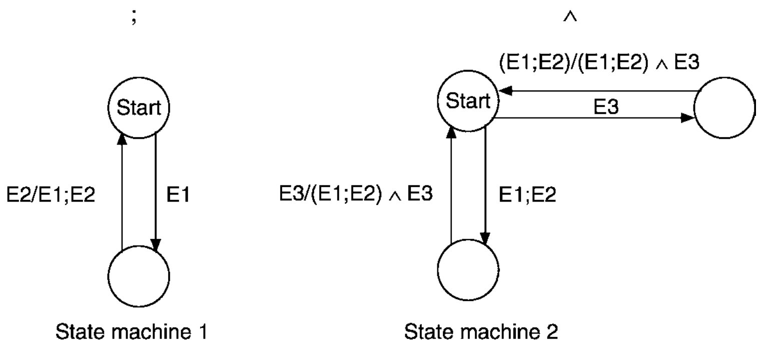


Event Detection. Figure 1. Syntax tree of $(E_1;E_2) \wedge E_3$ [18, Fig. 4.6].

For example, consider the expression E_c **defined by** $(E_1;E_2) \wedge E_3$ that means an E_c occurs if a sequence of E_1 and E_2 and an event of type E_3 occurs. This can be transformed into the syntax tree in Snoop as depicted in Fig. 1, since the conjunction '∧' is the principal event operator of E_c. As in data flow machines, the idea is that the nodes are evaluated according to some policy; for example, a common policy is that as soon as there are data available (events) on the input edges to an event, the operator is evaluated. Thus, the node ';' is evaluated if there are E_1, E_2 or both E_1 and E_2 events available. In colored Petri-nets used in SAMOS, the expression E_c is represented as depicted in Fig. 2. The events are represented as tokens in the places. Further, the event expression can be translated into a finite state automata as in ODE (depicted in Fig. 3), where the transitions are labeled with *Event/Action* where actions denoted E_x means generate a composite event occurrence E_x. A significant difference between the three different representations of event operators is whether they are capable of handling multiple situations in which events are generated concurrently. The approaches in Snoop and SAMOS can handle concurrent situations, whereas finite state automatas must be augmented with processes executing the finite state



Event Detection. Figure 2. Petri-net representation of $(E_1;E_2) \wedge E_3$ [18, Fig. 4.7].



Event Detection. Figure 3. Finite state automata representation of $(E_1;E_2) \wedge E_3$ [18, Fig. 4.8].

automatas to enable this. The rest of the historical background address the following significant issues: *(i)* management of the combinatorial explosion in the analysis of event streams; *(ii)* parameterized event detection, and *(iii)* design issues.

In Snoop [8,9], event contexts or event parameter consumption modes were introduced handle the combinatorial explosion, issue *(i)*, in a systematic way. Each event context represents some semantic or pragmatic knowledge about the situations that are monitored. For example, in recent event context only the most recent event occurrences are allowed to be used to form composite events. This context is meant to be used in, for example, real-time applications. In contrast, in chronicle event context event matching is based on historical order and is meant to be used to monitor, for example, that communication protocols are not violated.

Another way to handle the combinatorial explosion is to limit the size of the past. Essentially, this is a technique from distributed, parallel, and real-time systems, in which time intervals are used to determine the validity of events. For example, in remote procedure calls timeouts are generally used to detect failures such as server node crashes or network packet omissions – timeouts occur if there is no response within a specified time interval. The timeouts can be used to limit the past by removing events that can no longer be part of a composite event in the future. This technique has been suggested in related areas, for example, chronicle recognition [12], monitoring of real-time logic [10], and general monitoring of distributed systems [17]. For example, the technique of using time windows to limit processing has been successfully employed in event detection in real-time control [19].

One significant issue is parameterized event detection (issue *(ii)*), because it is often the case that events in event specifications must carry parameters denoting the situation in which the events were generated. In SAMOS, a limited form of parameterized event detection is allowed by either requiring that events stem from the same transaction or the same user. Bækgaard and Godskesen [2] introduced specification of parameterized events based on TCCS (temporal calculi for communicating systems). However, they have not shown how to transform the specification into code or configuration of event detection. In EPL [20] and XChange^{EQ} [4] parameterized event detection are addressed. Both EPL and XChange^{EQ} stem from databases, for example, an incremental join algorithm for parameterized event detection is proposed by Bry and Eckert [5].

Significant results concerning design issues, issue *(iii)*, are that the following features are among the most important: storage structures for event streams, indexing techniques of event streams, handling of isolation property of transaction with respect to event detection as well as memory management techniques. The BEAST benchmark [15] concluded that indexing techniques and storage structures are more important than, for example, choosing Petri-nets, finite state automata or code to perform the matching. In DeeDS [1], the isolation property is violated in a controlled way by integrating event detection in the transaction processing in such a way that events can violate isolation if explicitly allowed to. The events are cached in a separate filtered event log as well as stored as first class objects in the database. This entails that the problem of doing event detection in the scope of transaction is removed, that is, event objects are not locked as is the case when event detection is performed within transactions.

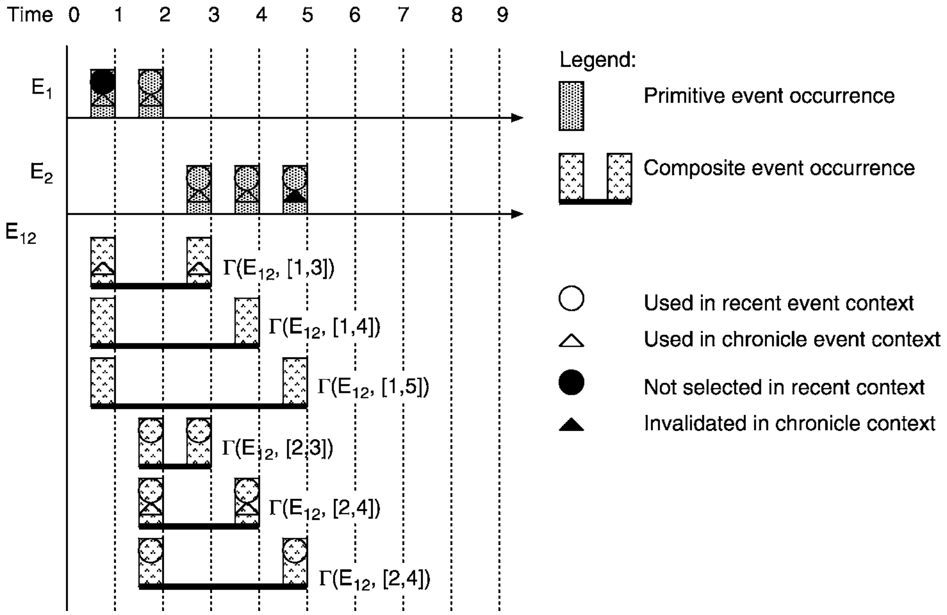
The drawback is that the filtered event log must be maintained, that is, support recovery processing and the isolation property as well as support pruning of invalid events.

Foundations

There are several issues that are significant. These issues can be categorized into a usage and a development perspective. The use of event detection entails two difficult problems: *(i)* understanding what is actually happening and *(ii)* dealing with event detection in different temporal scopes. The first problem has been considered in, for example, visualization of event detection [3]. It was observed that master's students had difficulty in understanding event detection when the topic was covered in graduate courses. The most likely explanation is that, in contrast to condition evaluation, it is also necessary to know the previous results of event detection to determine the result of the current evaluation. For example, consider Fig. 4 for E_{12} **defined by** $E_1;E_2$ where $\Gamma(E,[t_1,t_2])$ reads that "an event of type E has occurred throughout the interval $[t_1,t_2]$"; in this example, the different results of event detection can be confusing. All variations of E_{12} are discovered in general (unconstrained) event context, and the composite events marked with circles are discovered in the recent event context, whereas the composite events marked with triangles are discovered in chronicle event context. Note that, for example, to understand that $\Gamma(E,[2,4])$ has been discovered require knowledge of that $\Gamma(E[1,3])$ has been discovered and the consequences for future event detection. Thus, there is need for tools to help developing, verifying and validating event specifications. These tools require that the event detection process is instrumented in such a way that internal states can be observed.

The second problem relates to parameterized event detection. This is typically used to enable matching of events with the same temporal scope (e.g., same transaction, same process, accessing the same object). One problem is how these temporal scopes relate to each other, for example, a transaction can encompass multiple processes and these processes can serve several transactions. Sometimes it is necessary to mix temporal scopes, for example, the occurrence of an event outside a temporal scope may negate the occurrence of a composite event within the scope.

From a development perspective of ADBMS and event detection in particular, there are two issues of importance: *(i)* data structures for efficient event detection, and *(ii)* decentralized event detection. In database prototypes simple queue structures has been employed to store events, whereas, for example, some kind of a heap structure may provide better results. In particular, if event detection is explicitly invoked rather than implicitly invoked each time an event occurs, then


Time 0 1 2 3 4 5 6 7 8 9
E1
E2
E12
Γ(E12, [1,3])
Γ(E12, [1,4])
Γ(E12, [1,5])
Γ(E12, [2,3])
Γ(E12, [2,4])
Γ(E12, [2,4])
Legend:
Primitive event occurrence
Composite event occurrence
Used in recent event context
Used in chronicle event context
Not selected in recent context
Invalidated in chronicle context

Event Detection. Figure 4. Example of event detection in some event contexts [18, Fig. 6.1].

more advanced data structures can improve the performance and time complexity of event detection. Concerning issue *(ii)*, event detection is suitable for decentralized processing, since it follows the data flow machine idea. However, there are a lot of issues that must be considered in such situations. Guiding requirements are, among others, interoperability requirements and the filtering effect of event detection for a particular domain. An example of the latter requirement is network event correlation [16], in which a major part of network events filtered out and the design of event detection should be optimized for pruning events.

In terms of performance, the algorithmic time complexity depends on a number of factors: the type of event context and whether a single signaled event is evaluated (as in implicit invocation of event monitoring) or multiple signaled events are evaluated (as in explicit invocation of event monitoring). In general, the difference in terms of algorithmic time complexity between different event contexts disappears if multiple events are evaluated. Table 1 describes the algorithmic time complexity in situations where all signaled events results in composite events. The meaning of the symbols are as follows: H is the event history (set of buffered events), M is the set of event types, E is an event expression, and P is a the set of parameters carried by each event. The function f depends on how event types are indexed in the system. For example, if the red-black tree technique is employed for indexing then $\mathrm{f}(n) = \log(n)$, whereas $\mathrm{f}(x) = 1$ if hashing is employed instead. Other event contexts (e.g., [6]) may have different algorithmic time complexity characteristics.

The most important factor is the size of the event history and by keeping this as short as possible it is even possible to employ the unconstrained event context in practice (cf. [19]). Event expressions are typically short unless they are automatically generated from some other specifications. For example, if a set of event types is required to occur in a particular order and it is necessary to detect failures to do so as soon as possible, then it is necessary to monitor all permutations of the correct order and in this case the size of event expressions can become uncommonly large. The size of the parameters is typically not a problem in high bandwidth systems and is less significant compared to the event history and the size of the event expressions. Finally, the number of event types in a system is insignificant as long as proper indexing is employed.

Event Detection. Table 1. Algorithmic time complexity of event detection [18].

Event context	Single	Multiple
Recent	$O(\mathrm{f}(\|M\|) + \|E\|\|P\|)$	$O(\|H\|(\mathrm{f}(\|M\|) + \|E\|\|P\|))$
Chronicle		
Continuous	$O(\mathrm{f}(\|M\|) + \|H\|\|E\|\|P\|)$	
Cumulative		
General (unconstrained)	$O(\|H\|^{\|E\|})$	

Key Applications

The key users of this area are the researchers and developers of active functionality in, for example, active databases.

Future Directions

Some future issues in event detection is the following:

1. To develop tools and techniques for developing, understanding, verifying and validating event specifications. As addressed in this entry, the understanding of event detection is more complex than condition evaluation and this is a really crucial issue for successful use of event detection.
2. To investigate relevant data structures to improve event detection, in particular, parameterized event detection as well as decentralized event detection. If large event expressions are prevalent in a system, then this issue becomes critical.
3. To further investigate the performance characteristics in, for example, decentralized event detection. The reported study [18] has been performed for centralized event detection. Further work need to be done in other settings.
4. To investigate fault-tolerant event detection in various settings such as mobile ad-hoc networks.
5. To investigate if fuzzy reasoning inside event detection can be used. For example, in monitoring of real-world situations, it is necessary to handle uncertainties of various kinds.

Cross-references

▶ Active Database Execution Model
▶ Active Database Knowledge Model
▶ ADBMS (Active Database Management System) Architecture
▶ Atomic Event

- ▶ Composite Event
- ▶ Event
- ▶ Event specification

Recommended Reading

1. Andler S., Hansson J., Eriksson J., Mellin J., Berndtsson M., and Eftring B. DeeDS towards a distributed active and real-time database system. Special Issue on Real Time Data Base Systems. ACM SIGMOD Rec., 25(1):38–51, 1996.
2. Bækgaard L. and Godskesen J.C. Real-time event control in active databases. J. Syst. Softw., 42(3):263–271, 1997.
3. Berndtsson M., Mellin J., and Högberg U. Visualization of the composite event detection process. In Proc. Int. Workshop on User Interfaces to Data Intensive Systems, 1999.
4. Bry F. and Eckert M. Rule-based composite event queries: the language XChangeEQ and its semantics. In Proc. 1st Int. Conf. on Web Reasoning and Rule Systems, 2007, pp. 16–30.
5. Bry F. and Eckert M. Temporal order optimizations of incremental joins for composite event detection. In Proc. Inaugural Int. Conf. on Distributed Event-Based Systems, 2007, pp. 85–90.
6. Carlsson J. Event Pattern Detection for Embedded Systems. Ph.D. thesis no 44, Mälardalen University, 2007.
7. Chakravarthy S., Blaustein B., Buchmann A.P., Carey M., Dayal U., Goldhirsch D., Hsu M., Jauhuri R., Ladin R., Livny M., McCarthy D., McKee R., and Rosenthal A. HiPAC: a research project in active time-constrained database management. Tech. Rep. XAIT-89-02, Xerox Advanced Information Technology, 1989.
8. Chakravarthy S., Krishnaprasad V., Anwar E., and Kim S.K. Composite events for active database: semantics, contexts, and detection. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 606–617.
9. Chakravarthy S. and Mishra D. Snoop: an event specification language for active databases. Knowl. Data Eng., 14(1):1–26, 1994.
10. Chodrow S.E., Jahanian F., and Donner M. Run-time monitoring of real-time systems. In Proc. Real-Time Systems Symposium, 1991, pp. 74–83.
11. Dayal U., Blaustein B., Buchmann A., Chakravarthy S., Hsu M., Ladin R., McCarty D., Rosenthal A., Sarin S., Carey M.J., Livny M., and Jauharu R. The HiPAC project: combining active databases and timing constraints. ACM SIGMOD Rec., 17 (1):51–70, 1988.
12. Dousson C., Gaborit P., and Ghallab M. Situation recognition: representation and algorithms. In Proc. 13th Int. Joint Conf. on AI, 1993, pp. 166–172.
13. Gatziu S. Events in an Active Object-Oriented Database System. Ph.D. thesis, University of Zurich, Switzerland, 1994.
14. Gehani N.H., Jagadish H.V., and Schmueli O. COMPOSE – a system for composite event specification and detection. In Advanced Database Concepts and Research Issues. Springer, Berlin, 1993.
15. Geppert A., Berndtsson M., Lieuwen D., and Roncancio C. Performance evaluation of object-oriented active database management systems using the BEAST benchmark. Theor. Pract. Object Syst. 4(4):1–16, 1998.
16. Liu G., Mok A., and Yang E. Composite events for network event correlation. In Proc. IFIP/IEEE Int. Symp. on Integrated Network Management, 1999.
17. Mansouri-Samani M. and Sloman M. GEM: a generalized event monitoring language for distributed systems. IEE/IOP/BCS Distrib. Syst. Eng. J., 4(2):96–108, 1997.
18. Mellin J. Resource-Predictable and Efficient Monitoring of Events. Ph.D. thesis no 876, University of Linköping, 2004.
19. Milne R., Nicol C., Ghallab M., Trave-massuyes L., Bousson K., Dousson C., Quevedo J., Martin J.A., and Guasch A. TIGER: real-time situation assessment of dynamic systems. Intell. Syst. Eng., 3(3):103–124, 1994.
20. Motakis I. and Zaniolo C. Composite temporal events in active database rules: a logic-oriented approach. In Proc. 4th Int. Conf. on Deductive and Object-Oriented Databases, 1995, pp. 19–37.

Event Driven Architecture

K. Mani Chandy
California Institute of Technology, Pasadena, CA, USA

Synonyms

Event driven service-oriented architecture; Event processing systems; Sense and respond systems; Sensor network systems; Active databases; Streaming database systems

Definitions

An event driven architecture is a software architecture for applications that detect and respond to events. An event is a significant change in the state of a system or its environment. The change may occur rapidly or slowly. The occurrence of an event in the past, or its current unfolding, or a prediction of an event in the future is deduced from data. An event-driven architecture includes sensors and other sources of data; processors that fuse data from multiple sensors and detect patterns over time, geographical locations, and other attributes and deduce events that occurred or predict events; responders for initiating actions in response to events; communication links for transferring information between components; and administrative software for monitoring, tailoring and managing the application.

Historical Background

Event driven systems have been used in military command and control systems, and in SCADA (supervisory control and data acquisition) systems for decades. Platforms for event driven applications have been developed by several companies in the last decade; these general-purpose platforms can be tailored to obtain special-purpose event-driven applications. The development of event-driven platforms has evolved from several fields including databases, message-driven middleware, rule-based systems, business intelligence, statistics, and command and control systems.

Foundations

The metrics for evaluating event driven applications are different from those used to evaluate many other information technology systems. A discussion of the metrics helps highlight the goals of event driven systems and clarifies the differences between these systems and more traditional information technologies.

Measures for Evaluating Event Driven Applications

Event-driven applications detect and respond to events. Examples of such applications include those that: take action when earthquakes, tsunamis, or hurricanes are imminent; help manage investment portfolios by sensing risks and opportunities in markets; help disabled or aging people live independently by detecting and responding to potential problems; and prevent crime by identifying and responding to threats.

Measures by which event-driven applications are evaluated include the usual measures of total cost of ownership and measures, called the *REACTS* metrics (after the first letters of the measures), which are more specific to event driven architectures.

- *Relevance*: This measures the relevance of information generated by the system to consumers of the information. A message about a tsunami warning sent to a person's phone is highly relevant while she is on a beach where the tsunami is about to strike but is less relevant while she is hundreds of miles away. Relevance depends on the state of the receiver. Systems that send a great deal of irrelevant information (spam) are likely to be ignored.
- *Effort*: This measures the effort required by operational people to define specifications of events and responses and by IT staff to integrate event applications with existing systems. The time taken by end users, such as stock traders, to tune systems to meet their specific needs, and the time taken by IT staff to integrate event applications with existing infrastructure are major costs.
- *Accuracy*: This measures the costs of inaccurate information, such as false positives. A false tsunami warning gets beaches to be evacuated unnecessarily. High frequencies of inaccurate information result in information being ignored.
- *Completeness*: This measures the costs of incomplete information. For example, the absence of a hurricane warning can results in thousands of deaths. A single false negative has a higher cost than a single false positive in most situations; however, systems are often designed with very high rates of false positives compared to rates of false negatives.
- *Timeliness*: This measures the costs of detecting events too late. The benefit of detecting an event depends on the time available to execute an appropriate response. There is little added benefit in having millisecond responses when responses in minutes will do. There are, however, significant costs in responses initiated too late: a tsunami warning received after a tsunami strikes offers no value.
- *Security*: This measures the costs of reducing and managing intrusions and other attacks. Event driven applications often run for extended periods and manage critical infrastructure. For example, new infrastructure in the electrical power grid enables utilities to control airconditioners and other devices in homes and offices. Attacks that shut off power to homes and factories have severe consequences.

The design of event driven applications is a process of evaluating, comparing, and trading off REACTS measures for different design options.

Components of Event Driven Architectures

An event – a significant state change – is different from the data that describes the event; data that describes an event is often called an *event object*. Event objects may be stored in databases, sent as messages, or used within programs. An action, such as issuing a warning about a possible fraud, is itself an event. Information about the action is captured in an event object or is not captured at all.

Event driven architectures consist of data acquisition components such as sensors; event processors that integrate data from multiple sensors and databases; responders that initiate appropriate actions after events are detected; communication links that transmit information between components and administrative services that help in specifying and managing the operation of event driven applications. Next, each of these components is discussed briefly.

Sensors: Data acquisition components acquire data from the environment and from within the enterprise; this data is analyzed to detect patterns that signify events. Data may be pushed to these components or the components may acquire data by polling services. For example, stock ticker feeds are data acquisition components that push stock ticker information to the application without having the application make an explicit request for each stock price update. Other data acquisition components poll Web sites or other services according to fixed schedules or when requested to do so by event processors. These components send the polled information – or the difference between data obtained on successive polls – to the application, or store the data for later processing. Polling too frequently is onerous for the site being polled, while polling infrequently can increase delays in event detection.

Event Processors: The role of event processors is to integrate data from multiple sensors, add value to event objects by incorporating data from databases and business intelligence repositories, find patterns that signal events, and initiate actions by responders. Event processors may also request sensors to obtain additional data, providing closed-loop coupling between data acquisition and processing. Common functions of event processors include the following:

- Event Filtering: Data may be generated by sensors, such as stock tickers or RFID readers, at rapid rates and only a fraction of this data may be relevant to an application. Event filters are specified by the data that they pass through to later stages of processing. Filters located near sensors, physically or logically, reduce the amount of communication.
- Event Object Enrichment: Enriching an event object by adding relevant data from databases helps in later stages of processing. For example, an event object describing a customer's request for a new service can be enriched by adding data about the customer's history.
- Event Fusion: Sequences of event objects sent by multiple sensors and event processors are integrated, or "fused" by other event processors, to generate new event objects. For example, streams of event objects generated by multiple package-delivery trucks are fused into a single stream dealing with all packages.
- Event Stream Splitting: A stream of event objects can be split into multiple streams each of which deals with a separate set of attributes; processing streams of event objects all of which deal with the same topics is simpler.
- Time Series Analysis: Event processors in many applications carry out time series analyses to detect anomalous behavior or to determine if behaviors have changed significantly over time.
- Complex Event Processing and Event Prediction: The detection of complex patterns requires analytics that spans time, geographical locations, and business functions. Initiating responses in preparation for predicted events reduces reaction delay. The detection of complex patterns in the historical record and the accurate prediction of future events often requires complex processing.

Responders

A typical response is the initiation of a business process. Example responses include updating dashboards that display key performance indicators, sending alerts, setting up temporary task forces to respond to an event, and updating databases and logs for later analysis. The execution of the business process responding to an event may, in turn, generate new events.

Communication Layer

Communication among sensors and event processors, in some applications, is by means of wireless. In some applications, wireless bandwidth may be low because sensors may be required to be inexpensive, power may have to be conserved, or the environment may be noisy. The geographical layout of components communicating by wireless is critical in such applications.

The logical architecture for communication among components in many applications is publish/subscribe. Some components publish event objects, and some components subscribe for specified types of event objects with specified ranges of parameters. The pub/sub network ensures delivery of event objects to appropriate destinations. The logical communication architecture can also be point-to-point, pushing event objects asynchronously from one point in the network to the next, or a broadcast system.

Administration Layer

The administration layer is used for several purposes including the following: initially to tailor a general-purpose event driven architecture to suit a specific application; at run time to monitor an application; and, at redesign time to replay scenarios for forensics and for adapting systems to changing conditions. Since event driven applications are often used to control physical infrastructures over long periods, monitoring sensors and responders is an essential aspect of these applications. Thus, the administrative layer of an event driven application is itself an event driven application.

System Specification

The process of implementing an event driven application is, in essence, an application integration activity: sensors, processors and responders have to be integrated into operations. The specification of an event driven application consists of specifications of the sensing, processing and responding subsystems and their integration. The subsystems are quite different from each other; therefore, the notations used in specifying them are different.

Specifications of sensing subsystems include identification of what data sources and sensors to use, where they should be placed, how they should communicate, activation and polling schedules, and mappings from data models used by the external environment to data models used within the application. Specifications of event processing subsystems deal with functionalities such as fusing, filtering and splitting that also occur in database systems, rule-based systems and business intelligence systems; therefore some notations for specifying event processors use extensions of SQL, or rules, or statistical packages. Specifications of responder subsystems identify activities carried out when an event is detected; these activities are often specified using notations for business process management (BPM) and business application monitoring (BAM).

Key Applications

Many areas of science, commerce and education use event-driven applications. All systems in which data is produced continuously and which require timely response to critical events are candidates for event-driven architectures. Rapid accurate response is a competitive advantage in financial trading of stocks, commodities and foreign exchange. EDA is also used in responding to fraud and non-compliance to regulations.

Cyber infrastructure – the integration of physical infrastructure with information technology – is based on event driven applications. Supply chain applications including airline, railroad and trucking systems use EDA. Energy management in the electrical power transmission system is an increasingly important application of EDA as smart meters and sensors are deployed in homes. Many defense department and homeland security applications, such as interdiction of radiation material, use EDA. Crisis management systems, in public and private sectors, are based on EDA.

Customer relationship management (CRM) requires rapid detection and response to customer interests while customers visit Web sites or call service centers. Event driven applications are used in healthcare to monitor patients and to help an aging population live independently for longer.

Future Directions

Event driven applications will become widespread as sensors become ubiquitous and rates of data production in blogs, news, videos, and other online sources continue to increase. A subdiscipline of event driven systems will be formed by integrating concepts from disciplines such as databases, business intelligence and statistics, distributed systems and control, signal processing and state estimation, decision support and operations research, and collaboration technologies.

Cross-references

- ► Active Databases
- ► Ad-hoc Queries in Sensor Networks
- ► Business Intelligence
- ► Complex Event
- ► Continuous Query
- ► Data Mining
- ► ECA Rules
- ► Event-Driven Business Process Management
- ► Message Queuing Systems
- ► Publish/Subscribe
- ► Streaming Database Systems
- ► Text Mining

Recommended Reading

1. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and issues in data streams. In Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 2002.
2. Chandy K.M., Charpentier M., and Capponi A. Towards a theory of events. In Proc. Inaugural Int. Conf. on Distributed Event-based Systems, 2007, pp. 180–187.

3. Deshpande A., Guestrin C., Madden S.R., Hellerstein J.M., and Hong W. Model-driven data acquisition in sensor networks. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
4. Etzion O. Semantic approach to event processing. In Proc. Inaugural Int. Conf. on Distributed Event-based Systems, 2007, p. 139.
5. Luckham D. The Power of Events: The Power of Complex Event Processing Addison Wesley, Reading, MA, USA, 2002.
6. Muhl G., Fiege L., and Pietzuch P. Distributed Event Based Systems. Springer, Berlin, 2006.
7. Schulte R. The Business Impact of Event Processing: Why Mainstream Companies will soon use a lot more EDA. In Proc. IEEE Services Computing Workshop, 2006, pp. 51.
8. Zhao F. and Guibas L. Wireless sensor networks: an information processing approach. Morgan Kaufmann, Los Altos, CA, 2004.

Event Driven Service-oriented Architecture

► Event Driven Architecture

Event Emitter

► Event Source

Event Extraction

► Video Scene and Event Detection

Event Flow

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event causality graph

Definition

An event flow is a partially ordered graph that traces event instances and causality relations among them.

Key Points

When an event instance is produced by an event source [1] and transferred to the event processing network, an entry node is created into the “event flow” of the specific application, every event created by an event processing agent is added to the event flow, such that, if the input event instances to the event processing agent are <ei_1,...,ei_n> and the output event instances are <eo_1,...,eo_m> then an edge in the event flow is created for all pairs of the type.

<ei_i, eo_k> where i = 1,...,n and k = 1,...,m. This denotes that there is some causality between the input event-instance and the output event-instance. An event flow may go beyond a single event processing network [3]; an input event to an orchestration node in a sink triggers an action, which in turn, creates more events.

Event flows [2] are being used for purposes of validation for event processing applications, as well as for event lineage.

Cross-references

► Event Causality
► Event Lineage
► Event Processing Agent

Recommended Reading

1. Luckham D, Schulte R. (eds.). EPTS Event Processing Glossary version 1.1 http://complexevents.com/?p=409
2. Schwanke R.W., Toward a Real-time Event Flow Architecture Style. In Proc. 8th Annual Int. Conf. and Workshop on the Engineering of Computer-based Systems, 2001, pp. 94–102.
3. Sharon G., Etzion O. Event Processing network: Model and implementation. IBM Syst. J., 47(2):321–334, 2008.

Event in Active Databases

AnnMarie Ericsson, Mikael Berndtsson,
Jonas Mellin
University of Skövde, Skövde, Sweden

Definition

Events in active databases are typically considered to be atomic and instantaneous, i.e., they either happen or not. Events are either primitive or composite, that is, either they are part of the system (primitive) or they are specified in terms of other events (composite).

Key Points

An event is a happening of interest that can invoke execution of reactive behavior, for example, trigger an ECA rule. Events in active databases are typically considered to be atomic and instantaneous [1,3,4], that is, they either happen or not. However, this leads to problems [2] addressed in the event specification entry.

Events can be classified as primitive or composite events. Primitive events refer to elementary occurrences that are predefined in the system. Primitive events are typically further categorized as database events, time events, transaction events, method events etc. A composite event is a set of events matching an event specification.

Events can have attributes, that is, event parameters. Event parameters are usually passed on to the condition and action part of an ECA rule. In general, event parameters can be broadly classified into: (i) System level information, and (ii) Application level information. System level information is related to parameters such as transaction identity, user identity and time stamp(s). Application level information in an object-oriented environment can, for example, include object identity and parameters of the invoked method.

Events play a key role in active databases, since they specify when an ECA rule is triggered. Whenever an event is defined and used in an ECA rule, one can say that the event is explicitly defined. If the active database allows the event to be implied by the rule definition, then the event is said to be implicit. In this case, the event specification of a rule is optional.

Cross-references

- ▶ Active Database Execution Model
- ▶ Active Database Knowledge Model
- ▶ Atomic Event
- ▶ Composite Event
- ▶ ECA Rules
- ▶ Event specification

Recommended Reading

1. Chakravarthy S. and Mishra D. Snoop: an expressive event specification language for active databases. Data Knowl. Eng., 14(1):1–26, 1994.
2. Galton A. and Augusto J. Two approaches to event definition. In Proc. 13th Int. Conf. Database and Expert Syst. Appl., 2002, pp. 547–556.
3. Gatziu S. Events in an Active Object-Oriented Database System. Ph.D. thesis, University of Zurich, Switzerland, 1994, Verlag Dr. Kovac, Hamburg, Germany.
4. Gehani N., Jagadish H.V., and Smueli O. Event Specification in an active object-oriented database. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1992, pp. 81–90.

Event in Temporal Databases

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Event relation; Instant relation

Definition

An *event* is an instantaneous fact, i.e., something occurring at an instant of time.

Key Points

Some temporal data models support valid-time relations where tuples represent events and where the tuples are thus timestamped with time values that represent time instants. When granules are used as timestamps, an event timestamped with granule *t* occurs, or is valid, at some instant during *t*.

It may be observed that more general kinds of events have also been considered in the literature, including events with duration and complex and composite events.

Cross-references

- ▶ Temporal Database
- ▶ Temporal Granularity
- ▶ Time Instant
- ▶ Valid Time

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399. Springer, Berlin Heidelberg New York, 1998, pp. 406–413.
2. Events. The Internet encyclopedia of philosophy. Available at: http://www.iep.utm.edu/e/events.htm.
3. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version.

In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399. Springer, Berlin Heidelberg New York, 1998, pp. 367–405.

Event Lineage

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event pedigree

Definition

An Event lineage is a path in the event processing network that enables the tracing of the consequences of an event, or finding the reasons for a certain action.

Key Points

An event lineage is an audit feature that is determined to answer one of two questions:

1. What were all the implications of event e?
2. What were the events and processing agents that preceded action a?

The event lineage is obtained by maintaining the history of event processing networks using various techniques.

Cross-references

- ▶ Data Lineage
- ▶ Event Flow
- ▶ Event Processing Agent
- ▶ Event Processing Network

Recommended Reading

1. Benjelloun O., Sarma A.D., Halevy A.Y., Theobald M., and Widom J. Databases with uncertainty and lineage. VLDB J., 17(2):243–264, 2008.
2. Sharon G. and Etzion O. Event processing networks – model and implementation. IBM Syst. J., 47(2):321–334, 2008.

Event Mapping

▶ Event Transformation

Event Network Edge

▶ Event Stream

Event Pathway

▶ Event Channel

Event Pattern Detection

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event composition (partial overlap)

Definition

Event Pattern detection is a computational process in which a collection of events are evaluated to check whether they satisfy a pre-defined pattern. Formally an event pattern (EP) is defined as:

EP = <C, IE, PT, PRED, Policies, DER>

where:

- C = Context
- IE = List of Input Event types
- PT = Pattern Type
- Pred = Predicate
- Policies = Semantic fine-tuning policies
- Der = Derived Events

- – A context is a collection of semantic dimensions within which the event occurs. These dimensions may include: temporal context, spatial context, state-related context and reference-related context.
- – List of input event types provide the EPN edges (event pipes/event streams) that are potentially input to the pattern. Note that events that open or close contexts are indirectly also input event to the patterns within that context
- – Pattern Types:

A pattern type is a formula with variables substituted by event types, example- and (e1, e2) – where e1, e2 are variables of type event type. There are serial pattern types:

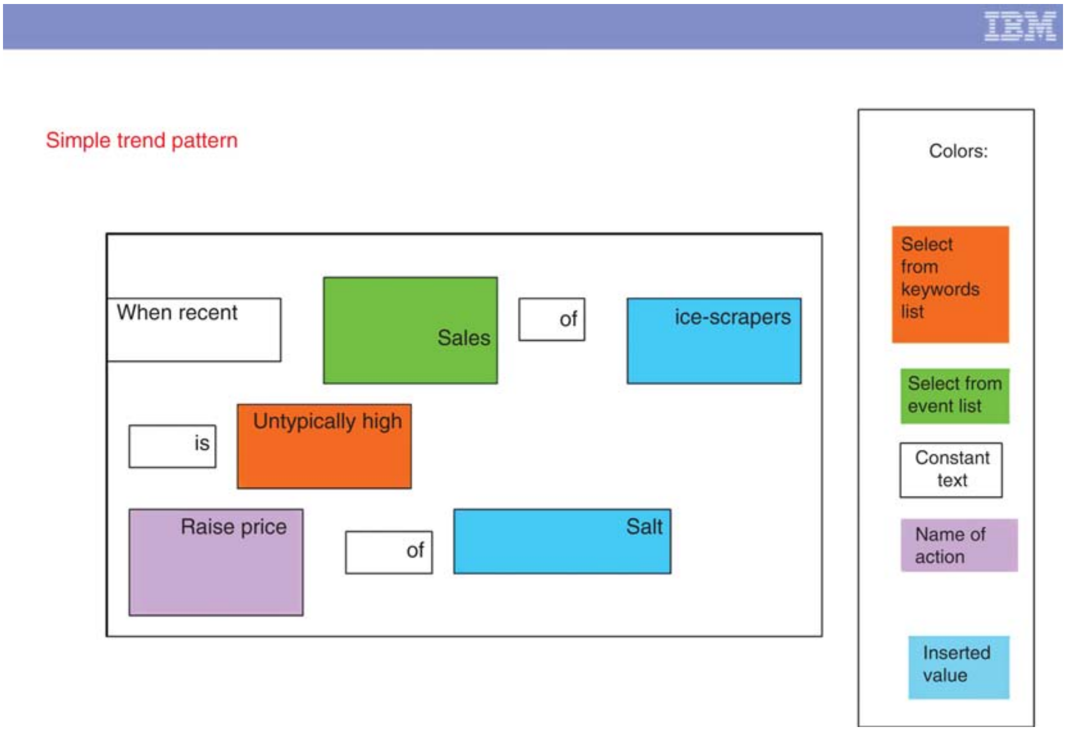
- Basic event oriented patterns, example: and (e1, e2).
- Basic set (stream) oriented patterns, example: (average reading.temperature >35).
- Dimensional patterns – temporal, spatial, spatio-temporal example: moving (e, north).
- Context termination patterns, example: absent (e1).
- Interval-oriented patterns, example: overlap context.

- Predicates: Predicates among various of the input events, example e1.a > e2. b.
- Policies: The policies have been determined to fine-tune the semantics. Example: what should be done if there is synonym events, e.g., the pattern type is and (e1, e2) and there are three instances of e1 followed by five instances of e2 – how many times this pattern will be detected and with which event instances.
- Derived Events: derive one or more events out of the input events that have participated in the pattern detection. The derived event can be a collection of events, or a single event that is derived from the content of other events.

Historical Background

Pattern detection has its roots in several different areas:

1. *Discrete event simulation*: Automatic analysis of event trends to find causality relations among the events in simulation trace has resulted in the need to find "event causality" relations, which has been the motivation for introducing patterns. David Luckham's "complex event processing" has grown up from research in that area
2. *Network and system management*: Network (and later system) management attempt to find the "root cause" of a problem, by taking as an input a collection of symptoms (e.g., device time-outs) and diagnose the core problem that has caused these symptoms. There are relatively simple patterns that belong to this family
3. *Active Databases*: The work on "composite events" in active database is one of the example of event patterns, it features some forms of patterns, and also some policies ("consumption modes")
4. *Middleware*: Publish/subscribe with content selection, and ESB mediations are sources of further (relatively basic) patterns



Simple trend pattern

When recent Sales of ice-scrapers

is Untypically high

Raise price of Salt

Colors:

Select from keywords list

Select from event list

Constant text

Name of action

Inserted value

Event Pattern Detection. Figure 1. User oriented patterns.

Foundations

The notion of pattern has originated from building architecture, as a tool to capture architectural decisions. The area of software design has borrowed this term to document a collection of activities which issue the "best practice" to solve a particular issue (Figure 1). In event processing, the notion of patterns may have several interpretations:

1. Detection of patterns over the event history – as referred to in this entry
2. User-oriented templates that may be developed into more than a single pattern.
3. Best practices pattern of design of use.

Key Applications

Patterns can be related to various types of applications:

1. Real-Time Enterprise application: in which patterns are identified to create actions that are flowing back into the business logic and effecting it. Example: algorithmic trading, active liquidity management, various applications of autonomic computing.
2. Business Activity Management: in which patters are identified to find threats, opportunities and exceptional deviations from the norm or desired behavior measured by constraints and Key Performance Indicators.
3. Diagnostics application: In this application events are symptoms and pattern detection is used in order to determine the problem root cause.
4. Information dissemination: In some cases patterns are used to enable more sophisticated subscription systems, where the user wishes to subscribe not to raw events but to cases in which patterns are satisfied, example: IBM stock has gone up in 2% within 2 h.
5. Prediction: In this case there should be a support in predicted events and the pattern can include predicted events, where the reaction intends to mitigate or eliminate some situation from occurring.

Cross-references

- ► Complex Event
- ► Complex Event Processing
- ► Composite Event
- ► Context

Recommended Reading

1. Buschmann F., Henney K., and Schmidt D.C. Pattern-Oriented Software Architecture, a Pattern Language for Distributed Computing, vol. 4. Wiley, New York, 2007.
2. Coplien J.O. and Schmidt D.C. (eds.). Pattern Languages of Program Design. ACM Press/Addison-Wesley, Reading, MA, 1995.
3. Etzion O. Event processing, architecture and patterns, tutorial. In 2nd Int. Conf. on Distributed Event-based Systems, 2008.
4. Gawlick D. and Mishra S. CEP: functionality, technology and context, Tutorial. In 2nd Int. Conf. on Distributed Event-based systems, 2008.
5. Hoope G. and Woolf B. Enterprise Integration Patterns – Designing, Building, and Deploying of messaging solutions. Addison-Wesley, Reading, MA, 2003.
6. Luckham D. Power of Events, Addison-Wesley, Reading, MA, 2002.
7. Römer K. Discovery of frequent distributed event patterns in sensor networks. In Proc. 5th European Conf. on Event Patterns in Wireless Sensor Networks, 2008, pp. 106–124.
8. Sharon G. and Etzion O. Event processing networks – model and implementation. IBM Syst. J., 47(2):321–334, 2008.
9. Widder A., von Ammon R., Schaeffer P., and Wolff C. Identification of suspicious, unknown event patterns in an event cloud. In Proc. Inaugural Int. Conf. on Distributed Event-Based Systems, 2007, pp. 164–170.

Event Pedigree

► Event Lineage

Event Pipe

► Event Stream

Event Prediction

Segev Wasserkrug
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Prediction regarding future events; Prediction of event occurrence

Definition

Event prediction is the capability to estimate the possibility that some event may occur in the future. Some applications may even be required to advance beyond simply predicting the occurrence of events and need to influence the occurrence of some future event. For example, this may involve either decreasing the chance that a non-desirable event will occur or increasing the chance that a desirable event will occur. To determine whether or not such actions should indeed be carried out requires decision-making capabilities.

Historical Background

The first event-based systems were active databases in which automatic actions were carried out as a result of database queries. This was done using the *ECA* (Event-Condition-Action) paradigm. As such applications evolved, the events to which applications were required to respond evolved beyond single queries (e.g., insertion of deletions of data) and needed to be deterministically inferred from several such queries. As the complexity of such applications continued to grow, and as event-based applications expanded into other application domains, the need to handle explicit uncertainty regarding the occurrence of events needed to be accommodated. Event-based applications needed to handle unreliable sensor readings and reason about the occurrence of events that could not be deterministically inferred.

The next evolutionary step to such event-based capabilities is to enable event-based systems to reason about the possibility that events will occur in the future and provide facilities for reasoning about possibly influencing such occurrences.

Foundations

Events that may or may not occur in the future are inherently uncertain. Therefore, in many cases, event-based applications are required to estimate the likelihood that such events will occur and attempt to influence this likelihood.

As an example, consider an e-trading web site from a commercial bank. Customers interact with this kind of site using banking transaction events, which include queries regarding the account balance, withdrawals, and deposits. Such events may serve as indications of future events. For example, there may be a number of withdrawal actions carried out by a customer, the end result of which is that there is little or no money left in the account. This set of events may be an indication of the customer's intent to close the account.

In such a case, it would be beneficial for the bank to be able to predict the likelihood that the customer will indeed close the account based on the set of withdrawal events. Furthermore, such an application should be able to consider how possible actions could potentially reduce the likelihood of such an event, while taking into account the potential costs and benefits involved.

As the above example illustrates, there are two main components to a solution that addresses event prediction:

- Component enabling prediction regarding future events
- Component that facilitates decisions regarding actions intended to impact the likelihood of future events, while accounting for the implications of these actions

Mechanisms for Event Prediction

A variety of data can be associated with the occurrence of an event (this data is called the event payload). Two examples of such data are the point in time at which an event occurred and the new stock price in an event describing a change in the price of a specific stock. Some data are common to all events (e.g., their time of occurrence), while others are specific only to some events (e.g., data describing stock prices are relevant only for stock-related events). The data items associated with an event are termed *attributes*. Because an event has attributes, the prediction of an event needs to determine the likelihood of the event occurring as well as the possible values of its attributes.

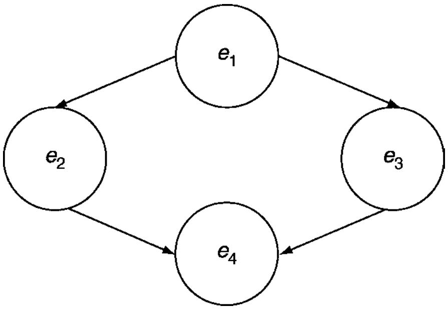
The definition of an event is very broad and may encompass a wide variety of information. Therefore, there are many relevant mechanisms from other domains that can be applied to event prediction. However, when applying these mechanisms, characteristics specific to event-based applications must be taken into account. These characteristics include the need to predict the attribute values of future events and the ability to take into account causal relations between events. This is elaborated below.

Because each event has a set of attributes associated with it, future predictions regarding an event must be able to provide some information about the possible values of the attributes, as well as some information regarding the likelihood that a specific attribute will indeed be assigned a specific value.

The need to take into account causal relations between events stems from the fact that the occurrence of some events may provide information regarding the occurrence of other events. For example, it may be the case that if an event e_1 occurs, event e_2 becomes more likely to occur as well. Such causal relations could be much more complex. For example, consider the relation depicted in Fig. 1, which states the following:

- Event e_1 occurring increases the likelihood of event e_2 occurring, as well as the likelihood of event e_3 occurring.
- Event e_2 occurring increases the likelihood of event e_4 occurring.
- Event e_3 occurring also increases the likelihood of event e_4 occurring.

These relations between events need to be taken into account in the correct manner. For example, if the likelihood of events e_2 and e_3 occurring increase independently of each other, the likelihood of event e_4 occurring may be greater than the case in which the likelihood of events e_2 and e_3 occur increases due only to the fact that event e_1 has occurred.



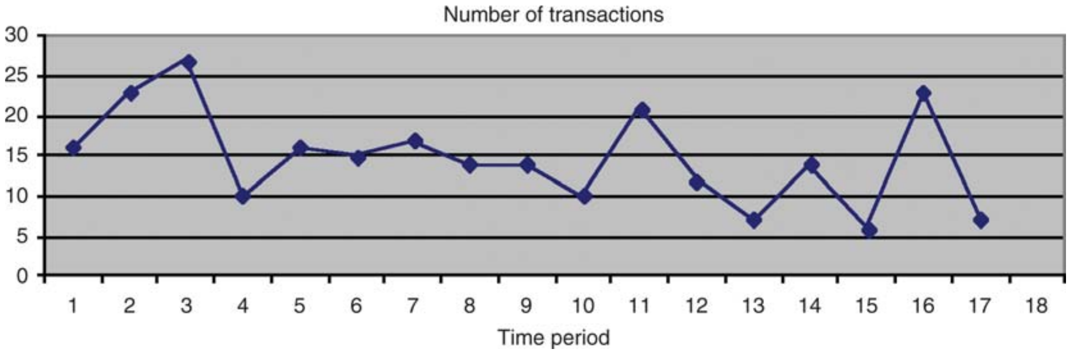
Event Prediction. Figure 1. Example of causal relations between events.

Methods for event prediction can be divided into two general types: explicit modeling methods and implicit modeling methods.

Explicit modeling methods are methods that require the ability to explicitly state the relationship between existing knowledge and events which must be predicted. Usually, such relationships take some known functional form. An example of a well known explicit modeling method is time series analysis, in which some future value is estimated, based on past values, using some function of the past values. The form of the function is defined by the person building the time series model, while the parameters of the function are derived from past data.

Figure 2 shows a plotting of past time series data in which the number of transactions is plotted for 17 time points. Based on this data, the number of transactions at the 18th time point can be predicted using time series analysis methods.

Another explicit modeling method is event composition languages. Event composition languages enable the definition of rules regarding the relationship between events. While most event composition languages are deterministic, there are event composition languages that can take uncertainty into account. Some of these languages are also suitable for defining the relationships required to enable event prediction. For example, a rule could state that if event e_1 is known to have occurred, the probability of event e_2 occurring at some future time t is 0.7. Such languages have the added benefit of naturally enabling the representation of the causal relationships between events. Furthermore, if the language is general enough, it can represent any functional form. Therefore, such languages can be seen as a generalization of all explicit modeling methods.



Event Prediction. Figure 2. Time series plot.

Implicit modeling involves models in which the relationship between the evidence and future events are not explicitly stated, but are directly derived from historical data. An example of such a modeling method is neural networks from the machine learning domain. In the neural network model, a model is constructed which may take many functional forms. The specific functional form is then derived from historical data.

The advantage of explicit models is that they can capture any inherent relationships between existing knowledge and future events, and utilize this knowledge to enable more accurate predictions. The disadvantage is that the person creating the models must possess this knowledge and be able to articulate it in a formal manner. Implicit models remove this burden from the model creator, but require large amounts of historical data. In addition, in implicit models it is more difficult to utilize inherent knowledge to guide the predictions. Consequently, there are also models that attempt to combine the two approaches by enabling the utilization of inherent knowledge when such knowledge is readily available, while deriving other parts of the model from historical data.

Influencing the Occurrence of Future Events

A solution for influencing the occurrence of future events must take into account the following:

- A set of alternative actions A, where each action $a \in A$ is intended to influence the event.
- For each action $a \in A$, the cost c_a of taking the action.
- For each action $a \in A$, the manner in which a will influence the probability of the event.
- For an undesirable event, the benefit b if the event does not occur.
- For a desirable event, the benefit b if the event occurs.
- For an undesirable event, the penalty p if the event occurs.
- For a desirable event, the penalty p if the event does not occur.

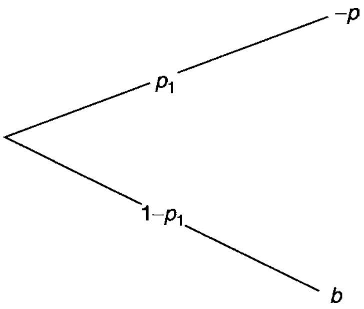
For example, consider the event of a customer closing a banking account. One possible way to prevent a customer from closing the account is to offer better terms, such as higher interest rates on savings. Offering such terms is an action $a \in A$. The loss of income due the higher interest rates is the cost c of taking action a. The benefit of the customer not closing the account is b, while the penalty p is the penalty associated with the customer closing the account.

Several methods exist for taking into account criteria such as the one above. Perhaps the most well known method is utility theory, where decisions are made based on the maximization of some utility function u. The possible decisions in utility theory are often represented by decision trees, such as the one in Fig. 3, which represents the decision tree associated with the event of a customer closing an account.

In Fig. 3, the two branches represent the two possibilities: The customer closing the account (the top branch), and the customer not closing the account (the bottom branch). The quantities at the end of the branches represent the penalties and benefits associated with each action. Therefore, the quantity at the end of the top branch is $-p$, which is the penalty when the customer closes the account, while the benefit of the bottom branch is b, i.e., the benefit when the account remains open. Note that this decision tree represents the case in which no action is taken. In this case, the expected utility can be calculated by $p_1 \cdot (-p) + (1 - p_1) \cdot b$.

Consider the case where it is expected that action a will reduce the probability of the customer closing the account from p_1 to p_2, such that $p_2 < p_1$. For this second case, the expected utility is $p_2 \cdot (-p) + (1 - p_2) \cdot (-c + b)$. In such a case, the decision may be to carry out action a whenever $p_2 \cdot (-p) + (1 - p_2) \cdot (-c + b) > p_1 \cdot (-p) + (1 - p_1) \cdot b$.

Note that the above mechanism for decision making does not explicitly represent an action $a \in A$. Rather, this representation was implicit in the different probability spaces and benefits/costs. Other decision mechanisms, such as Markov Decision Processes (MDPs), do enable an explicit representation of the actions involved, the



Event Prediction. Figure 3. Sample utility decision tree.

manner in which the actions influence the probability of an even occurring, and the associated costs and benefits.

Key Applications

Event-based applications that only react to past events are termed *reactive applications*. Applications that predict, and act upon the prediction of future events, are said to possess *proactive functionality*. Proactive functionality may be required in almost all domains. One such domain is Customer Relationship Management (CRM), in which proactive actions are often required to retain and/or grow market share and benefit. The banking example used throughout this entry is an example of an application in the CRM domain.

Future Directions

While research in event-based applications has been ongoing for several years, research in event prediction is still in its infancy. Therefore, this field is still largely unexplored and is ripe for future research.

Cross-references

▶ Active and Real-time Data Warehousing
▶ Active Database (aDB)
▶ Active Database (management) System (aDBS/aDBMS)
▶ Atomic Event
▶ Complex Event Processing
▶ Complex Event
▶ Composite Event
▶ Event
▶ Event and Pattern Detection over Streams
▶ Event Causality
▶ Event Driven Architecture
▶ Event Pattern Detection
▶ Explicit Event
▶ Implicit Event
▶ Uncertainty in Events

Recommended Reading

1. Alpaydin E. Introduction to Machine Learning. MIT Press, Cambridge, MA, 2004.
2. Brockwell P.J. and Davis R.A. Introduction to Time Series and Forecasting. Springer, New York, 2002.
3. Fishburn P.C. The Foundations of Expected Utility. Kluwer, Dordrecht, 1982.
4. Halpern J.Y. Reasoning about Uncertainty. MIT Press, Cambridge, MA, 2003.
5. Luckham D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Reading, MA, 2002.
6. Wasserkrug S., Gal A., and Etzion O. A model for reasoning with uncertain rules in event composition systems. In Proc. 21st Annual Conf. on Uncertainty in Artificial Intelligence, 2005, pp. 599–608.

Event Processing

▶ Complex Event Processing

Event Processing Agent

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event processing component; Event processing mediator

Definition

A software module and a node in the event processing network that process events [2].

Key Points

An event processing agent [3] is a node in the event processing network that receives one or more events as input, processes them, and creates one or more events as output. There are various types of event processing agents [4]:

1. Filter agent: receives a single event and decides based on a predicate expression whether to route this event further or not (in this case it is a terminal node for this specific event instance).
2. Validation agent: receives a single event and a predicate expression that serves as an integrity constraint for this event. If the integrity constraint is violated may act as a filter (reject the event), may transform the event, or may orchestrate additional actions in an event sink.
3. Transformation agent: receives a single event and transforms this event to another event format.

4. Split agent: receives a single event and creates multiple events, all are functions of the original event (may be either clones or the same events, or distinct overlapping or non-overlapping events).
5. Aggregation agent: receives multiple events and creates a derived event, whose values are aggregated from the values of the input events.
6. Pattern-detection agent: receives multiple events and based on a pre-defined pattern creates a complex event from a collection of events that satisfy the pattern.

Aggregation and pattern detection are statefull agents, and the rest are stateless agents. Aggregation agent has a small state that is incrementally updated, and the input events are not kept; Pattern-detection agent keeps some or the entire input event during the processing period, thus, it may develop large states. Agents may receive events by push or pull [3].

Cross-references

- ▶ Complex Event
- ▶ Event Pattern Detection
- ▶ Event Processing Network
- ▶ Event Routing
- ▶ Event Transformation

Recommended Reading

1. Luckham D. The Power of Events. Addison-Wesley, 2002.
2. Luckham D., Schulte R. (eds.). EPTS Event Processing Glossary version 1.1. http://complexevents.com/?p=409.
3. Loke S.W., Padovitz A., and Zaslavsky A. Context-Based Addressing: The Concept and an Implementation for Large-Scale Mobile Agent Systems Using Publish-Subscribe Event Notification. In 4th Int. Conf. on Distributed Applications and Interoperable Systems, 2003, pp. 274–284.
4. Sharon G., Etzion O. Event Processing network: Model and implementation, IBM Syst. J., 47(2):321–334, 2008.

Event Processing Component

▶ Event Processing Agent

Event Processing Mediator

▶ Event Processing Agent

Event Processing Network

Guy Sharon
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

EPN

Definition

An event processing network (EPN) is a graph $G = (V,E)$ where:

1. The set of nodes in an EPN graph G is denoted V such that $V = C \cup P \cup A \cup EC$. V is a set of nodes of four types; C denoting Event Consumer, P denoting Event Producer, A denoting Event Processing Agent and EC denoting Event Channel.
2. The set of edges in an EPN graph G is denoted E such that $E = \{(u,v) \mid (u \in (P \vee A) \rightarrow v \in EC) \wedge (u \in EC \rightarrow v \in (C \vee A))\}$. E is a set of ordered pairs of nodes representing directed edges. These edges are between either an event producer to an event channel, an event channel to an event consumer, an event processing agent to an event channel, or an event channel to an event processing agent.

The EPN model also includes a relationship definition between edges called Event Causality.

An event processing network (EPN) describes how events received from producers are directed to the consumers through agents that process these events by performing transformation, validation or enrichment. Any event flowing from one component to another must flow through a connection component. Therefore, an EPN model consists of four components: Event Producer, Event Consumer, Event Processing Agent, and a connection component called Event Channel.

Historical Background

Event driven computation is not a new concept; it is being used in user interfaces to perform a specific code when a button is pushed and when a mouse button is clicked or released. It is also being used in database systems as triggers to perform specific logic when a record is inserted, deleted or updated. Some existing approaches in configuring and expressing the event processing directives in event-driven systems are through SQL-like languages, script languages, or rule languages and are executed by standalone software, messaging systems or data stream management systems. While event driven processing is not a new concept, there is a need to provide conceptual techniques for definition and modeling of an event driven architecture, making it possible to express event processing intentions independently to the implementation models and executions. This is done through a conceptual model called Event Processing Network [3].

Foundations

As businesses are required to be more agile, to respond to the market rapidly and to be able to adapt quickly and efficiently, there is need for underlying systems and business applications to support such an environment. A sound and cost-effective way to achieve such goals is to introduce an event-driven architecture (EDA).

The producer (source) of the events only knows that the events transpired. The producer has no knowledge of the events' subsequent processing, or the interested parties, leading to entirely decoupled architecture. Such architecture enables flexible definition of the event processing logic: detection of event patterns, derivation of new events, transformation, and routing from producers to consumers based on the business logic required. Thus businesses can react to changes, execute relevant processes as a result, and influence ongoing processes based on the changes. Furthermore, such definition of event processing can be easily modified and quickly deployed in accordance with business needs such as changes to business processes and policies. The Event Processing Network (EPN) concepts model these processing directives.

An EPN model consists of four components: Event Producer, Event Consumer, Event Processing Agent, and a connection component called Event Channel.

Event Channel

An event channel [4] is a mechanism for delivering event streams from event producers and event processing agents to event consumers and event processing agents.

The event channel may receive multiple event streams from different sources and may transfer a combined event stream from all the sources to multiple sinks. It is the responsibility of the event channel to create an ordered and combined event stream from the sources and provide this stream to each target.

Another responsibility of an event channel is to retain the history of the events streaming through for retrospective event processing [6], defined as the discovery of event patterns over the history of events as opposed to online event processing, detecting predefined event patterns as new events become available.

Within a network of nodes and edges an event channel is represented as a node with edges directed to and from the node. Every incoming edge represents an event stream from an event producer or processing agent to publish events on to the channel. Every outgoing edge represents an event stream to an event consumer or processing agent subscribed to the channel. In Fig. 1 a graphical representation of an event channel is presented, where the name of the channel is within the boxed arrow and the event types are given on top of each edge to signify what type of events are provided by the event stream.

An Event Channel is defined as follows:

1. Every node in EC may have event retention policies defined for retrospective processing.
2. The edges connected to a node in EC are defined as $\forall v \in EC,\ \exists e_{in}, e_{out} \in E$: $e_{in} = (u \in (P \vee A), v)$, $e_{out} = (v, u \in (C \vee A))$.

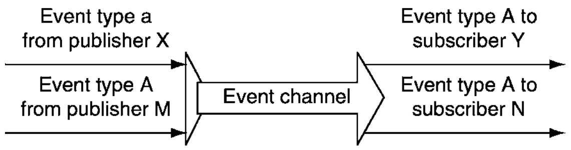
Event Producer and Consumer

An event producer, also known as an event source or event emitter, produces and publishes events through event channels for any party of interest to consume. This could be an event consumer or an event processing agent.

Within a network of nodes and edges an event producer is represented as a source node, i.e., there exist only edges directed from it. The number of edges directed from it is the number of different event channels that the producer publishes to. The edges connected to a node in P are defined as $\forall v \in P, \exists e_{out} \in E$: $e_{out} = (v, u \in EC)$ (Fig. 2).

An event consumer, also known as an event sink, is interested in events to perform its responsibilities. Once the event of interest is known to the consumer it will perform a certain task associated with this event.

Within a network of nodes and edges an event consumer is represented as a sink point, i.e., only edges directed to it. The number of edges directed to



Event Processing Network. Figure 1. Event channel.

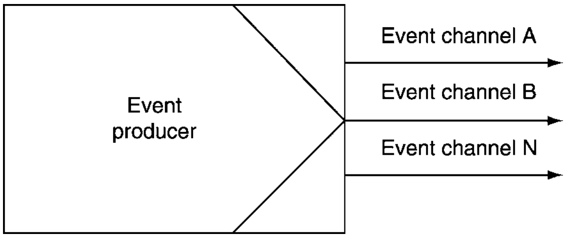
it is the number of different event channels that the consumer subscribes to. The edges connected to a node in C are defined as $\forall v \in C, \exists e_{in} \in E$: $e_{in} = (u \in EC, v)$ (Fig. 3).

Event Processing Agent

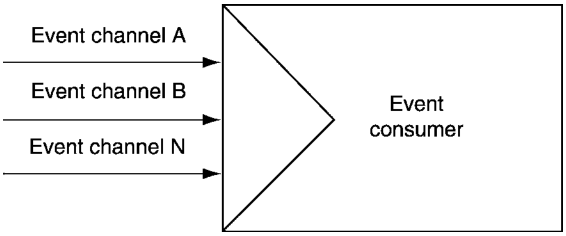
There are cases where there is a gap between the events produced by event producers, and the events required by the event consumers. These events may have different syntax (structure) and/or different semantic meaning than expected. There are also cases where a single event will not trigger an action performed by an event consumer, instead, by a complex composition of events happening at different times and in different contexts [2]. Event processing agents (EPA), also know as event mediators, are needed to detect patterns on 'raw' events, then to process these events through enrichment, transformation and validation, and finally to derive new events and publish them. An event processing agent is responsible in producing these derived events and decides where and how these events should be published.

The event processing agent is made up of three stages:

- Pattern detection – this stage is responsible in selecting events for processing according to a specified pattern.
- Processing – this stage is responsible in applying the processing functions on the selected events satisfying the pattern, resulting in derived events.



Event Processing Network. Figure 2. Event producer.



Event Processing Network. Figure 3. Event consumer.

- Emission – this stage is responsible in deciding where and how to publish the derived events.

An event processing agent subscribes to event channels for its pattern detection and publishes to event channels while emitting events.

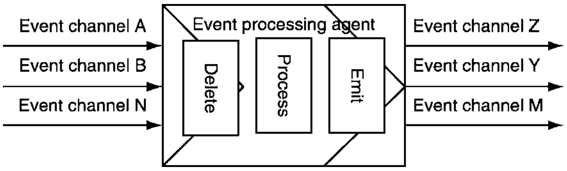
Within a network of nodes and edges an event processing agent is represented as a node with edges directed to and from the node. The number of edges directed to it is the number of different event channels that the agent subscribes to for detecting the pattern. The number of edges directed from it is the number of different event channels that the agent publishes to based on the processing and emission definitions.

The edges connected to a node in A are defined $\forall v \in A$, $\exists e_{in}, e_{out} \in E$: $e_{in} = (u, v)$, $e_{out} = (v, w)$, $u, w \in EC$. e_{in} is an event stream from an event channel the EPA is subscribed to and e_{out} is an event stream of derived events to an event channel the EPA publishes to (Fig. 4).

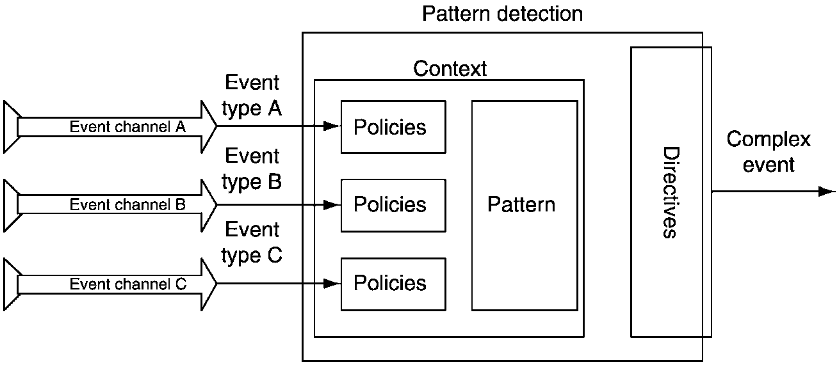
The following sub-sections discuss in detail the different stages within an event processing agent.

Pattern Detection At the pattern stage of an agent, a pattern is defined to be detected at runtime. A single event may match the pattern. In cases where more than one event is needed to detect a pattern, the collection of these events is called a complex event. The single or complex event is passed to the processing stage of an agent. The pattern description consists of four aspects and all are taken from the discipline of complex event processing [1,5] (Fig. 5).

Context – specifies the relevance of the events participating in pattern detection whether it is temporal based, spatial based, or semantic based. Temporal based means that events have to occur within a definable time frame. Spatial based means within a definable space concept such as geographical locations or regions. Semantic based context means that the events participating



Event Processing Network. Figure 4. Event processing agent.



Event Processing Network. Figure 5. Aspects of the pattern detection stage.

in pattern detection have relevance through a mutual object or entity. All types of contexts can be used in the same pattern detection and possibly more than once. Thus, an intersection or union of time frames can be specified as a context, or a context may be defined where events must occur in the same region and within the same time frame for pattern detection. Events occurring within the context defined are denoted as candidates to pattern detection.

Policies – specify how cases in which the semantics need to be fine tuned are handled, discussed in [5] as situation quantifiers. Policies include decisions such as:

- Whether to use the first, last or each of the events in the stream to detect the pattern or any other set operations
- Use only events satisfying a predicate on their attributes
- Whether each event in the stream may override the previous event and remove it from contributing to a pattern detection
- Expiry time, relative or absolute, of an event contributing in detecting the pattern
- Remove an event's contribution, due to the occurrence of a converse event, an event which conditionally implies the expiration of another event, described as contradiction elements in [1]

Choosing each event to contribute to pattern detection may result in multiple pattern detection, resulting in multiple complex events and therefore may result in multiple event publication by the event processing agent as a reaction to a single event consumed by the agent. Events occurring within the context defined and adhere to the defined policies are denoted as candidates to pattern detection.

Pattern – specifies the relationship among events complying with policies within context that is to be detected. Patterns are described in a form of event algebra discussed in [1] as situation operators. Some examples to operation semantics are

- *Any* – any event that occurs within the specified context and for which the policy holds, i.e., candidate, results in pattern detection.
- *Collect* – all events that occur within the specified context and for which the policies hold, i.e., candidates, are collected into one complex event at the end of the detection interval, for example, end of time window and exiting a geographical area.
- *Determine if* – similar to collect, however, the complex event will only include a statement of true if there were events within the detection interval and false otherwise.
- *And {eT1,...,eTn}* – one event from each event type, denoted as *eTi* must occur within the context for this pattern to be detected. There may be multiple pattern detections if there is more than one event of each type. This is determined by policies and whether the occurring events are candidates for pattern detection.

The EPN model does not restrict the set of operations to express patterns nor does it restrict the semantics given to operators.

Directives – specify directives for reporting on pattern detection to the processing stage and what to do with the events that contributed to the pattern detection, discussed in [1] as situation triggering expressions.

- Immediately as the pattern is detected.
- At the end of the detection interval.

- At time-outs associated with event attributes.
- At specifiable periods reporting on all the pattern detections since the last period.
- Allow a single or multiple activation of the processing stage when more than one pattern is detected during the detection interval.
- Consume, maintain always, or maintain an event for certain number of times after pattern detection for further detections, i.e., is the same event allowed to contribute in further pattern detection.
- Include the events that have not contributed to the pattern's detection with the report to the processing stage. When no pattern detection occurred during the reporting time such as at end of detection interval or at time-outs, all events within this period may be passed to the processing stage as well.

Processing Once the pattern detection stage has detected a pattern, it creates either a complex event, in case more than one event contributed to the detection, or passed the original event, in case only it was needed for the detection. This event is further processed based on the processing definition configured for the event processing agent. The result of the processing stage is a derived event ready for emission. There are four possible processing methods that can be performed on the result of the pattern detection stage:

- *Transformation* – specifies a transformation function to be performed on complex events to produce derived events. The transformation can cover a many-to-many relationship. Mapping is in the case of one-to-one where each output of the pattern detection is mapped to a single event structure. Splitting is in the case of one-to-many where each output of the pattern detection is split to several event structures, i.e., multiple derived events from a single complex event. Aggregation is in the case of many-to-one where an aggregation function is performed on the collection of events that are included within the complex event and a single event structure is produced with the function's results. Many-to-many transformation is achieved by applying aggregation and then splitting within the same processing.
- *Derivation* – specifies computations to apply on the collection of events reported by a complex event, which covers more than aggregation transformation. The result can be a single or a set of derived events.
- *Enrichment* – specifies directives in enriching the event with data from external sources such as a database. The result is a new event that is derived from the event received at the processing stage. The additional event data may be used to establish the connection to and retrieval of data from the external source.
- *Validation* – specifies constraints that the event must comply with (assertion) and instructions on how to handle the event in case it does not comply and violates the constraints. The decision on how to handle violation may be conditional and may differ by the repair action.
- Alert and let the processing continue.
- Reject and terminate the processing for this event. The event itself or a derived error event may be published to an event channel for error handling.
- Modify the event for the assertion to hold. When this is not possible, treat it in the same manner as rejecting the event.
- Perform change to external data if used in the assertion to make it hold. Again when this is not possible, treat it in the same manner as rejecting the event.

Emission The emission stage is responsible to emit the derived events to appropriate event channels and is configured with decisions to what event channels to emit and when to emit the events, such as immediately when received from the previous stage, periodically, at specific times or with time offsets. Thus, a single derived event may be emitted to more than one event channel.

Key Applications

Information Dissemination\Situation Awareness

Getting the right information in the right granularity to the right person\target at the right time:

Asset Tracking, Advertising context based content, Command and Control.

Active Diagnostics

Diagnose problems based on symptoms and resolve them:

System Management

Real Time Enterprise Reactions to events as part of business transactions – achieving low latency decisions and quick reaction to threat and opportunities:

Claims processing, Utilities grid management

Business Activity Monitoring

Quick observation into exceptional business behavior and notification to the appropriate people\targets:

Continuous Auditing, Fraud Detection

Predictive Processing

Mitigate or eliminate predicted events:

Epidemic control, Stock Trade

Cross-references

- ▶ Complex Event
- ▶ Complex Event Processing
- ▶ Context
- ▶ EDA
- ▶ Event Causality
- ▶ Event Channel
- ▶ Event Driven Architecture
- ▶ Event Processing Agent
- ▶ Event Sink
- ▶ Event Source
- ▶ Retrospective Event Processing

Recommended Reading

1. Adi A. A Language and an Execution Model for the Detection of Reaction Situations, PhD dissertation, Technion – Israel Institute of Technology, 2003.
2. Adi A., Biger A., Botzer D., Etzion O., and Sommer Z. Context Awareness in Amit. In Proc. 5th Annual Int. Workshop on Active Middleware Services, 2003, pp. 160–167.
3. Etzion O. and Sharon G. Event Processing Network – A Conceptual Model. In Proc. 2nd Int. Workshop on Event-driven Architecture, Processing and Systems, 2007.
4. Event Processing Glossary, http://complexevents.com/?p=195#comments
5. Luckham D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Reading, MA, 2002.
6. Yang Y., Pierce T., and Carbonell J.G. A Study on Retrospective and On-Line Event Detection. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 28–36.

Event Processing Systems

▶ Event Driven Architecture

Event Producer

▶ Event Source

Event Relation

▶ Event in Temporal Databases

Event Service

▶ Channel-Based Publish/Subscribe

Event Sink

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event consumer

Definition

An entity that receive events from other entities [1].

Key Points

Event sink are nodes in the edge of the event processing network (receiving only) that receive events from the event processing network, event sinks are typically of two types:

1. Orchestration event Sink: receives an event from the event processing network and determines what action should be triggered (e.g., ECA rule) [2]. An action can be – database update, activating a software module, activating/modifying/terminating a workflow instance [3].
2. Notification event sink: receives the event from an event processing network and sends it to a consumer – a person or a dashboard [4].

Note: the action that is triggered by the event sink may be an event source and emit events to the same event processing network or to another event processing network, this will be reflected in the event causality and event lineage.

Cross-references

- ▶ Event Causality
- ▶ Event Lineage
- ▶ Event Processing Network
- ▶ Event Source

Recommended Reading

1. Luckham D., Schulte D. (eds.). EPTS Event Processing Glossary version 1.1. http://complexevents.com/?p=409.
2. Lin Chen, Minglu Li, Jian Cao, Yi Wang. An ECA Rule-based Workflow Design Tool for Shanghai Grid. IEEE SCC 2005, 325–328.
3. Lin D., Sheng H., and Ishida T. Interorganizational Workflow Execution Based on Process Agents and ECA Rules. IEICE Transactions 90-D(9):1335–1342, 2007.
4. Ruby C.L., Green M.L., and Miller R. The Operations Dashboard: A Collaborative Environment for Monitoring Virtual Organization-specific Compute Element Operational Status. Parallel Processing Letters 16(4):485–500, 2006.

Event Source

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event producer; Event emitter

Definition

Event source is an entity that provides event for the processing cycle [3].

Key Points

Event sources are typically initial nodes (sending only) in the event processing networks, and provide events to the network for further processing. Event sources can be of various types:

1. Software module – through instrumentation
2. Sensors [1]
3. Clock/Calendar
4. State observers in workflows (BPM).

An event can be obtained from a source in various modes:

1. Push – the source initiates the event emission.
2. Periodic Pull – a calendar service pulls one or more events from the source (e.g., by query the log) [2].
3. On-Demand Pull – an event processing agent pulls the source using request/reply protocol.

Getting events from event sources to processing typically requires adapter to create the event in the desired format.

Cross-references

- ▶ Event Processing Network
- ▶ Event Sink

Recommended Reading

1. Chakraborty S., Poolsappasit N., and Ray I. Reliable Delivery of Event Data from Sensors to Actuators in Pervasive Computing Environments. In Proc. 21st Annual IFIP WG 11.3 Working Conf. on Data and Applications Security, 2007, pp. 77–92.
2. Deolasee P., Katkar A., Panchbudhe A., Ramamritham K., and Shenoy P.J. Adaptive Push-Pull: Dissemination of Dynamic Web Data. In Proc. 10th Int. World Wide Web Conference, 2001, pp. 265–274.
3. Luckham D., Schulte D. (eds.). EPTS Event Processing Glossary version 1.1. http://complexevents.com/?p=409.

Event Specification

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Synonyms

Event declaration; Event definition; Composite event query

Definition

Event specifications define event types in terms of patterns of other event types by using event operators (either expressed as operators in an operator grammar or as functions in a functional grammar) and event contexts. Event types are categorized as atomic (or primitive) or composite. An atomic event type is either a system primitive (for example, begin transaction) or it can be any defined atomic change in an activity such as a method, procedure, task etc. (e.g., method has been called). Event types defined in terms of other event types are named composite event types.

Historical Background

Essentially, whenever something has to be defined, it is necessary to have a specification. This can be found in several disciplines of computer science, for example, programming languages, compiler technology etc.

Event specifications became significant with the advent of the HiPAC project [10], in which seminal work on composite event types were done. Prior to HiPAC, mainly primitive event types were addressed in research publications and the event specifications only consisted of single atomic event types (often only system primitives).

Atomic event types in active databases can be categorized according to their sources (e.g., cf. Buchmann [4]): transactional events, temporal events, database manipulation events, and user-defined events. Transactional events are usually system primitives denoting state changes such as "transaction has begun," "transaction has committed," and "transaction has aborted." Temporal events are either absolute (a specified point in time) or relative to some other event and denotes that a time point has passed; further, temporal events can be repetitive and a special case of temporal events are periodic events. The database manipulation events depend on whether the database is relational, object-oriented or object-relational. In relational databases, there is typically only system primitives associated with the state changes corresponding to insertion, update, and deletion of entities (in particular, tuples). In object-oriented databases, system primitive events are typically associated with methods in existing classes associated with the database schema. Finally, in object-relation databases, a mix of system primitive events from both the relational aspect as well as the object-oriented aspect can be found. User-defined atomic events are either associated with methods [4] or composite event queries [3].

There are several event specification languages (either as complete specification languages in their own right or as intrinsic parts of another specification language), usually associated with the active database prototype (e.g., Snoop in Sentinel [8,9], SAMOS [12], ODE [13]), but there are several others that are not explicitly associated with active databases (e.g., GEM [16], monitoring of real-time logic [15]). All of these event specification languages in active databases share a basic set of event operators (e.g., Mellin [17], Carlsson and Lisper [6]): sequence, conjunction, disjunction and non-occurrence. The non-occurrence of events is always enclosed in an half-opened time interval, the question is what event types can be employed to delimit the interval. For example, in Snoop [8,9] and its descendants (e.g., REACH [5]) and SAMOS [12] intervals are delimited by arbitrary event types, whereas in ODE [13] the interval is from when the transaction, in which events occurs, begin. The interval is typically closed by an request to commit transaction or a relative temporal event with respect to the beginning of the interval.

Composite events are specified by event expressions in an operator grammar or in a functional grammar. For example, in Snoop [9] a sequence of events of type E_1 followed by events of type E_2 is expressed as $E_1;E_2$, whereas in EPL [18] it is expressed as $SEQ(E_1,E_2)$. Note that this specification does not dictate how many events of each type that matches the specification and it does not dictate if the events of different event types in the specification are interleaved. Most approaches (e.g., HiPAC [10], Snoop [9,8], SAMOS [12], ODE [13]) employ an operator grammar, whereas, for example, EPL (Event Pattern Language) [18] and XChange^{EQ} [3] are based on a functional grammar. Additionally, XChange^{EQ} is based on XML (e.g., Common Base Events [7]). The advantage of a functional grammar over an operator grammar is threefold:

1. There is no problem with associativity (e.g., the expression SEQ (E_1,E_2,E_3) does not have the ambiguity found in $E_1;E_2;E_3$ that either can mean $E_1;(E_2;E_3)$ or $(E_1;E_2);E_3$)
2. There is no problem with operator priority, a problem that is found in any operator grammar. For example, a conjunction should be prioritized over a disjunction so that $E_1 \& E_2|E_3$ is equivalent to $(E_1 \& E_2)|E_3$ meaning that either an E_3 occurs or a conjunction of E_1 and E_2 occurs, but how should sequences be prioritized with respect to conjunctions and disjunctions?
3. Operators in a functional grammar can be extended to be based on an arbitrary number of event types (e.g., SEQ $(E_1,E_2,...,E_n)$).

There are some pitfalls in event specifications: the definition of disjunctive event operator, and treating all event types as instantaneous. One event operator that have an unintuitive meaning in many event specification languages is disjunction (e.g., Snoop, SAMOS). In logic, disjunction can be inclusive or exclusive, but in many event specification languages it is neither, but a hybrid. Essentially, if an E_1 and an E_2 event occurs concurrently and E **defined by** $E_1|E_2$ is monitored, then the result is two event occurrences of type E with the same end time rather than one event occurrence of type E (inclusive disjunction) or no event occurrence (exclusive disjunction).

Initially, all events, both atomic and composite, were considered to be instantaneous [8,9]. However, as pointed out by Galton and Augusto [11], if all events are considered to be instantaneous, then this will introduce ambiguities in event specifications. For example, it is impossible to distinguish between E_1;(E_2;E_3) and E_2;(E_1;E_3).

Parameterized events carry parameter apart from the system default parameters of events, where the default parameters typically are event type, time stamps, and scope in which the event was generated (e.g., which transaction was the event generated in). Somehow, the event specification needs to enable specification of event parameters as, for example, by Bækgaard et al. [1]. Further, if parameterized event detection is allowed, then it is necessary to specify what parameters are part of the event type specification. Examples of specification of parameterized event detection are found in TCCS [1], XChange^{EQ} [3], and Solicitor [17].

Foundations

As always in formal specifications, there are two main approaches: either to define a minimal specification language that can handle all relevant situations or to define an expressive specification language in which the available expressions and operators are as close as possible to what is needed to express. The latter is prevalent in active database research (cf. Zimmer and Unland [19]), since it is assumed to lead to more comprehensive event specifications (e.g., the work by Liu et al. [15] illustrates the difference between Snoop and real-time logic)

There are four significant research issues that have been pursued within active databases with respect to event specifications: (i) the performance characteristics of event composition, (ii) how to introduce algebraic properties in event specification languages, (iii) how to define event specification languages that enable comprehensive specifications and, in particular, how to specify the orthogonal concepts of patterns and event contexts; and (iv) tools and techniques for understanding event specifications. Issue (i) has been dealt with within a benchmark named BEAST [14] as well as in algorithmic time complexity studies [17, ch. 9]. In the BEAST benchmark study [14] it was concluded that the underlying mechanism (e.g., finite state automatas, Petri-nets, or arbitrary code) of event composition was insignificant compared to typical information maintenance issues such as indexing of event types. It has been shown that the algorithmic time complexity of event composition is mainly based on event contexts, not the event operators per se [17, ch. 9].

The introduction of algebraic properties in event specification languages has been addressed by Carlsson and Lisper [6]. Essentially, they have introduced algebraic properties to an event specification language by defining a new event context: a variety of the recent event context. It is also an attempt to make a minimal language.

A few studies have addressed the specification events (issue (iii)), in particular how to mix event operators and event contexts. There are three approaches: (a) the event operator is locked to one event context as in, for example, ODE [13] or the work by Carlsson et al. [6]; (b) the event operators are explicitly associated with an event context as in, for example, Solicitor [17]; and (c) the different event types in the expressions are decorated with notation (For example, SEQ(*last* : E_1,*first* : E_2) is an example of using the proposed decorators *first* and *last*. This example specification is equivalent to SEQ(E_1,E_2) in recent event context.) that indicate the event context as proposed by Zimmer and Unland [19]. The most comprehensive form is approach (b), but more work is required, in particular on the topic of specifying parameterized event detection. For example, assume that there are concurrent processes accessing a shared variable and it is desirable to check that the processes behave by monitoring that processes enter and exit the critical regions associated with the shared variable within an expected time interval without a "terminate application" event in between. In this case, the matching of events representing entering and exiting the critical regions must be based on that they are generated by the same process, but the "terminate application" event is associated with all existing processes in the application. Further, parameters may express temporal scopes that subsume each other. For example, an event may be generated in a transaction by a particular process executing a specific component and events may carry the identity of transaction, process and component; in some cases, it is only desirable to only match on the transactional level, but in other cases a more precise scope such as the component level is desirable.

Concerning issue (iv), Berndtsson et al. [2] identified, in their investigation of existing visualization techniques of event composition, that event specifications are difficult to understand for M.Sc. level students. The conclusion is that tools to understand event specifications are necessary. Compared to condition evaluation, event composition is far more complicated since, in addition to condition evaluation, it is necessary to check what happened in previous evaluations of event expressions.

Key Applications

Buchmann [4] claims that there are two reasons for separating events from the condition in a rule: (i) to add the semantics of when the rule should be evaluated from what should be evaluated and (ii) efficiency of rule execution. Essentially, event specifications are employed for rule optimization in active databases. Further, if an active database is part of an event based system, then event specifications can be employed to alert other applications outside the active database.

Future Directions

There are three major unresolved threads of research on this topic: (i) specification of parameterized event types for parameterized event detection; (ii) reuse of event specifications; and (iii) usefulness of algebraic properties. The first thread addresses a problem that is similar to join in SQL queries and unification in Prolog programming language. That is, it deals with specifying that the values of certain parameters of an event should match the values of other parameters of another event in an event specification. For example, $E = E_1;E_2$ **where** $E_1.p_1 = E_2.p_2$ could be a notation stating that event occurrences of E only consists of event occurrences of E_1 and E_2 where the values of parameters p_1 and p_2 match. Making this kind of specification comprehensive, precise, unambiguous, and reusable is an open problem. (Note that the notation here is chosen for the likeness of SQL queries, it is not necessarily the best way to specify parameterized event types.)

Reuse of event specifications has not been addressed within active databases, but is a significant topic in software engineering. It would be interesting to, for example, consider if template event specifications can be useful in active databases. For example, in distributed systems call and reply semantics is employed in, among other things, remote procedure calls, that is, something is called and a reply is expected. Assume that C, R and T are template parameters to be instantiated, then a template for detecting a correct (E_c) or incorrect (E_i) sequence of call and reply within a time window could be as follows (where tps(E) and tpe(E) returns the time point of start and end of E):

$$E_c\langle C, R, T\rangle \textbf{ defined by } (C; R) \textbf{ without } E_i \textbf{ within } [\mathrm{tps(C)},\ \mathrm{tps(C)} + \mathrm{T}]$$

$$E_i\langle C, R, T\rangle \textbf{ defined by } \text{non-occurrence of } E_c \textbf{ within } [\mathrm{tps(C)},\ \mathrm{tps(C)} + \mathrm{T}]$$

Given this, a notation such as $E_c\langle A, B, 10s\rangle$ can be used to specify that $C = A$, $R = B$ and $T = 10s$. This specification should also introduce $E_i\langle A, B, 10s\rangle$ implicitly.

With respect to the usefulness of algebraic properties, the question is how useful algebraic properties are? As mentioned, Carlsson et al. [6] has shown that it is possible, but how useful is it compared to existing event contexts. This is still an open issue. For example, basing event composition on historical order, as in the chronicle event context, may become complicated, if not impossible.

Cross-references

- ▶ Active Database Coupling Modes
- ▶ Active Database Execution Model
- ▶ Active Database Knowledge Model
- ▶ Atomic Event
- ▶ Composite Event
- ▶ ECA Rules
- ▶ Event
- ▶ Event Detection

Recommended Reading

1. Bækgaard L. and Godskesen J.C. Real-time event control in active databases. J. Syst. Softw., 42(3):263–271, 1997.
2. Berndtsson M., Mellin J., and Högberg U. Visualization of the composite event detection process. In Proc. Int. Workshop on User Interfaces to Data Intensive Systems, 1999, pp. 118–127.
3. Bry F. and Eckert M. Rule-based composite event queries: the language XChangeEQ and its semantics. In Proc. First Int. Conf. on Web Reasoning and Rule Systems, 2007, pp. 16–30.
4. Buchmann A.P. Active object systems. In Advances in Object-Oriented Database Systems. A. Dogac, M.T. Ozsu, A. Biliris, and T. Sellis (eds.). Springer-Verlag, Berlin, 1994, pp. 201–224.
5. Buchmann A.P., Zimmermann J., Blakeley J.A., and Wells D.L. Building an integrated active OODBMS: requirements, architecture, and design decisions. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 117–128.
6. Carlson J. and Lisper B. An event detection algebra for reactive systems. In Proc. 4th ACM Int. Conf. on Embedded Software, 2004, pp. 147–154.

7. Common base events. URL http://www.ibm.com/developerworks/library/specification/ws-cbe/.
8. Chakravarthy S., Krishnaprasad V., Anwar E., and S.K. Kim Composite events for active database: semantics, contexts, and detection. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 606–617.
9. Chakravarthy S. and Mishra D. Snoop: an event specification language for active databases. Knowl. Data Eng., 13(3):1–26, 1994.
10. Dayal U., Blaustein B., Buchmann A., Chakravarthy S., Hsu M., Ladin R., McCarty D., Rosenthal A., Sarin S., Carey M.J., Livny M., and Jauharu R. The HiPAC Project: combining active databases and timing constraints. ACM SIGMOD Rec., 17(1):51–70, 1988.
11. Galton A. and Augusto J. Two approaches to event definition. In Proc. 13th Int. Conf. Database and Expert Syst. Appl., 2002, pp. 547–556.
12. Gatziu S. Events in an Active Object-Oriented Database System. Ph.D. thesis, University of Zurich, Switzerland, 1994.
13. Gehani N.H., Jagadish H.V., and Schmueli O. COMPOSE – a system for composite event specification and detection. In Advanced Database Concepts and Research Issues. Springer-Verlag, Berlin, 1993.
14. Geppert A., Berndtsson M., Lieuwen D., and Roncancio C. Performance evaluation of object-oriented active database management systems using the BEAST benchmark. Theor. Pract. Object Syst., 4(4):1–16, 1998.
15. Liu G., Mok A.K., and Konana P. A unified approach for specifying timing constraints and composite events in active real-time database systems. In Proc. 4th Real-Time Technology and Applications Symp., 1998, pp. 199–208.
16. Mansouri-Samani M. and Sloman M. GEM: a generalized event monitoring language for distributed systems. IEE/IOP/BCS Distrib. Syst. Eng. J., 4(2):96–108, 1997.
17. Mellin J. Resource-Predictable and Efficient Monitoring of Events. Ph.D. thesis, no. 876, University of Linköping, 2004.
18. Motakis I. and Zaniolo C. Formal semantics for composite temporal events in active database rules. J. Syst. Integrat., 7(3/4):291–325, 1997.
19. Zimmer D. and Unland R. On the semantics of complex events in active database management systems. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 392–399.

Event Stream

Opher Etzion
IBM Research Labs-Haifa, Haifa, Israel

Synonyms

Event pipe; Event network edge

Definition

An Event Stream is a collection of ordered events [4].

Key Points

An event stream is a collection of events, with one or more order relations. The naïve order relation is according to the time that the event arrives to the processing system, but this may not reflect the event order in reality [2], which may define an additional order relation. An event stream may be homogenous (all events have the same structure and type) or heterogeneous (include events that have different types). Homogenous streams are also known as *Data-Type Streams*. For practical usages, typically data-type streams are used. An edge in the event processing network, known as *event pipe* typically is a superset of *event stream* [5].

Sometimes event stream is used as a synonym to event pipe. This term has some relation to "data stream," but it is not identical [1], and does not necessarily correspond to SQL implementation. A typical case of event stream is a time-series, and there are some methods to handle out-of-order arrival in such streams [3].

Cross-references

- ▶ Data Stream
- ▶ Event Processing Network
- ▶ Event Flow

Recommended Reading

1. Albers K., Bodmann F., Slomka F. Hierarchical Event Streams and Event Dependency Graphs: A New Computational Model for Embedded Real-Time Systems. In Proc. 18th Euromicro Conf. Real-Time Systems, 2006, pp. 97–106.
2. Barga R.S., Goldstein J., Ali M.H., Hong M. Consistent Streaming Through Time: A Vision for Event Stream Processing. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 363–374.
3. Li M., Liu M., Rundensteiner E.A., and Mani M. Event Stream Processing with Out-of-Order Data Arrival. In Proc. 1st Int. Workshop on Distributed Event Processing, Systems and Applications, 2007.
4. Luckham D., Schulte D. (eds.). EPTS Event Processing Glossary version 1.1. http://complexevents.com/?p=409.
5. Sharon G., Etzion O. Event Processing network: Model and implementation, IBM System Journal, 47(2):321–334, 2008.

Event Stream Processing

▶ Complex Event Processing

Event Stream Processing (ESP)

▶ Event and Pattern Detection over Streams
▶ Stream Processing

Event Topic

▶ Event Channel

Event Trace Analysis

▶ Event Detection

Event Transformation

Peter Niblett
IBM United Kingdom Limited, Winchester, UK

Synonyms

Event mapping

Definition

In its broadest definition Event Transformation refers to any operation that takes as input a single event message or stream of event messages, and produces a single event message or stream of event messages as its output. As such, this definition encompasses many of the functions provided by an Event Processing Agent in an Event Processing Network, including agents that do complex pattern detection.

There is a somewhat narrower definition in which the operation has to be either a translation, aggregation, split or composition function.

Historical Background

Event translation (transformation of a single input event message into a single output event message) is a special case of message translation – a more general concept where the messages concerned do not necessarily represent events. Message translation has long been used to solve enterprise integration problems where the applications to be integrated have different data models. Enterprise integration sometimes also involves split operations, where input messages are split into multiple outputs. Hohpe and Woolf discuss a number of message translation and splitting patterns in [4].

Event Aggregation and Event Composition operations take streams of event messages as input and are fundamental components of all event stream processing systems. They were included in stream processing systems developed in research departments in the early part of the current century. For more information see the Aurora project [2] and Borealis project [1] from Brandeis, Brown and MIT, or the STREAM project [6] from Stanford. Further material on event processing can be found in [5].

Foundations

Transformations may be classified into the following types, depending on the nature of their input and output:

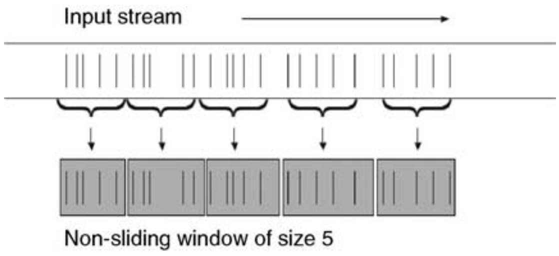
- *Translation.* Gets one event at a time, applies a translation operation and outputs the translated event. A translation operation is sometimes referred to as a *mapping.*
- *Split.* Gets one event at a time and splits the event to N different events, each of which contains a (possibly overlapping) subset of the event attributes.
- *Aggregation.* Takes a stream of events as input, and uses a windowing process to collect events from that stream into windows. The operation then processes each window to produce zero or more "derived" events as output.
- *Composition.* Takes two or more input streams and builds a window for each stream. The operation then takes this set of windows and processes them to produce zero or more derived events as output.

Aggregation and composition both operate against streams of events and since a stream is potentially unbounded they require the concept of a window defined over these input streams. A window is simply a set of event messages that is fetched from the input stream; the operation itself defines how the window is constructed from the stream and the conditions under which it is considered ready for processing. Examples include:

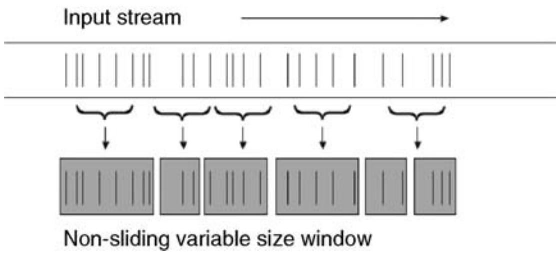
1. Non-sliding window of fixed size N. Events from the input stream are assigned to an "open window." When the specified number of events (N) has been received, the window is closed, ready to be

processed by the operation, and a new window is opened. Each input event is assigned to one and only one window (see Fig. 1).

2. Non-sliding variable size window. This is similar to the fixed-size non-sliding window, except that the closing of the window is triggered by some other condition, for example the detection of a change of type of the incoming event or the detection of a change in the value of a particular attribute on an incoming event (see Fig. 2).
3. Non-sliding time-based window. The operator specifies a time duration and a series of non-overlapping windows is created each of the specified duration. Events are assigned to a window instance based on some timestamp (for example arrival time) associated with the event. Such as if the window is specified to have a length of 2 s, then the window corresponding to time T would contain all events whose timestamp t lies between time T and T-2 seconds (see Fig. 3).
4. Sliding window of fixed size N. A new window is created when each incoming event is encountered. The incoming event and the preceding N-1 events (if any) are assigned to the window, which is then processed by the operation. Unlike the non-sliding examples, each event is assigned to multiple windows (see Fig. 4).
5. Sliding time-based window. A new window is created when each incoming event is encountered, but each window has the same fixed time duration. When a window is created, the incoming event is assigned to the window, along with all preceding events that lie within its time period, and the window is then processed. An alternative, possibly more natural, way of visualizing this is that there is a single window which moves in time; events are added to the window when they are detected, and removed from the window when they have overstayed the time limit. With either definition, if the window is specified to have a length of 2 s, then at time T the window would contain all events whose timestamp t lies between time T and T-2 s. (see Fig. 5).



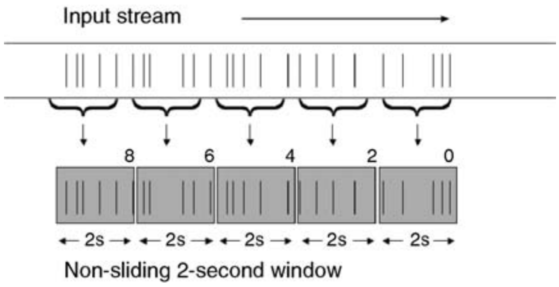
Event Transformation. Figure 1. Non-sliding window of fixed size.



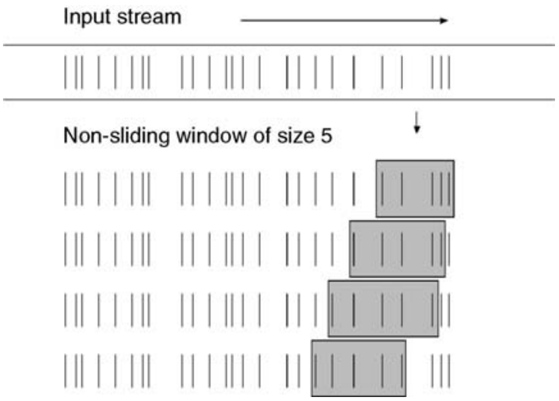
Event Transformation. Figure 2. Non-sliding variable size window.

Non-sliding windows are sometimes referred to as *tumbling* windows.

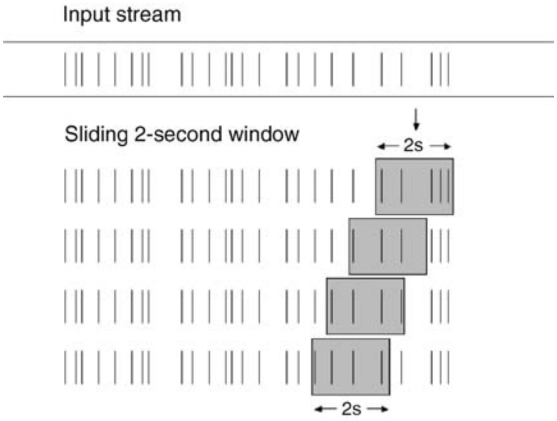
Translate and split style transformations have no requirement for a window, although they could be regarded as operating on a non-sliding window of size 1.



Event Transformation. Figure 3. Non-sliding time-based window.



Event Transformation. Figure 4. Sliding window of fixed size.



Event Transformation. Figure 5. Sliding time-based window.

A transformation where the result of processing a given event is affected by previous events from the input stream is referred to as a Stateful Transformation, whereas transformations where previous events have no effect on the processing of subsequent events are referred to as stateless transformations. Aggregation and composition are stateful by definition, whereas translate (and arguably split) can be either stateful or stateless.

Translation

A translation operation takes one event at a time and transforms it into another. Translation operations can be categorized into a number of different types depending on the amount of change they make to the incoming event.

The least intrusive type of event translation is one that changes the data representation (serialized form) of the event message, without changing its content. For example a translation might be used to convert between an XML and a Comma-Separated Value representation of the event, or to convert between different character sets.

Translation operations are frequently used to make more substantial changes to the content, and even the type, of the event:

- A translation might keep the content of the event unchanged, but modify the types used to represent the attributes of the event. For example it might convert a temperature attribute from Celsius to Fahrenheit, or replace a US state name with a two character code.
- A translation might modify the values of certain attributes of the event. This kind of translation can be used for data cleansing, for example replacing obviously incorrect date values.
- A translation can remove attributes from an event, while preserving the basic type structure of the event. Attributes could be removed if they are not needed by the event consumers, or because they contain sensitive information. A transformation that removes attributes is sometimes called a *projection*.
- Attributes can be added to the event, this is sometimes called *event enrichment*. The values assigned to the additional attributes can be:
  - Fixed values assigned by the translation, for example explicitly filling in default values for attributes omitted from the input event.
  - Values computed from other attributes in the event, for example the translation could assign a severity level to the event.
  - Values computed using an external source of information in addition to data from the event. An example of this would be a translation that uses a customer ID attribute from the input event to retrieve a customer name and address from a database and then adds these as attributes to the output event.

Translations are frequently programmed using visual editing tools, and can be represented using SQL queries, XML Style Sheets [3] that operate on an XML Infoset representation of the event, or other transformation languages.

Splitting

A splitting operation takes a single incoming event as input and produces multiple events as output. It is equivalent to taking a number of copies of the incoming event and applying a different translation to each. Splitting transformations are frequently used in combination with routing operations, with the different output events being routed to different recipients.

Some kinds of splitting transformation are particularly noteworthy:

- Splits that leave the original event untouched, but produce one or more additional events that are projections of the original event.
- Splits that break the original event apart by attribute, and produce a projected for each one of the attributes of the original event.

- Splits that iterate through lists in the original event and produce one projected event for each list item.

A split transformation can be programmed as a set of translations, this is sometimes referred to as a *static split*, since the number of output events is determined by the specification of the split operation. In contrast, in an *iterative split* the number of output events is determined by the data contained in the input event.

Aggregation

An aggregation operation is a function that operates on the events in a window to produce a single output event. The output event summarizes some aspect or aspects of the input events, by either:

- Concatenating the content of the input events, or
- Performing a function on the attributes of the input events; or

Examples of such functions include:

- Maximum, Minimum or "Top-k" values of an attribute.
- Sum or Mean values of an attribute.
- Other forms of average value of an attribute such as median or mode.
- Weighted averages, where one attribute in the event is the value to be averaged and another gives its weight. Weighted attributes are used to compute Volume-Weighted Average Price (VWAP) used in algorithmic trading applications.
- Count of the number of instances of an attribute, or the number of distinct values of an attribute.

An aggregation operation can also carry forwards a value from the previous window. For example when computing a "running maximum" of an attribute, or a VWAP, one may want to consider all events in a given time period (such as the current trading day), rather than just the events in the current window.

The choice of windowing style affects the output of the aggregation. For example an Averaging function with a non-sliding (tumbling) window will give a set of discrete averages, one for each window, whereas the use of sliding window will give a rolling average.

Composition

A composition operation takes windows from two (or more) input streams and performs an operation on them that produces zero, one or more output events. The most common type of composition operation is a *Join*.

Join operations are frequently used with sliding windows, to perform correlation between two streams. Each time an incoming event is detected on one of the streams, it is compared against the events in the window in the other stream using some kind of comparison test, and an event is output if a match is detected. This output event is produced by performing an operation on the attributes of the two matching events. In a two-way join operation both input streams are treated equally, and an incoming event on either stream is compared against the window of events on the other stream.

If the input streams are homogenous, all the events in a window have the same type, then standard relational Join operators can be used, with the two windows being treated as the two tables that are to be joined.

Key Applications

Event transformation of the types described here can be found in most event processing applications, for example in real-time monitoring, algorithmic trading, business activity monitoring.

Event translation has a particular use when integrating event producers and consumers that were not originally intended to work together.

Event aggregation and composition have a particular use in monitoring and simple pattern detection applications.

Cross-references

▶ Event Processing Network
▶ Event Routing
▶ Event Specification

Recommended Reading

1. Brandeis University, Brown University, and MIT, Borealis – Distributed Stream Processing Engine. http://www.cs.brown.edu/research/borealis/public/
2. Carney D. et al. Monitoring streams – A new class of data management applications. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 215–226.
3. Clark J. (ed.). XML Transformations (XSLT) 1.0. W3C Recommendation, http://www.w3.org/TR/1999/REC-xslt-19991116, 1999.
4. Hohpe G. and Woolf B. Enterprise Integration Patterns: Designing, Building and Deploying Messaging Solutions. Addison-Wesley, Reading, MA, 2004.

5. Luckham D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Reading, MA, 2002.
6. Stanford University, STREAM project. http://infolab.stanford.edu/stream/

Event Uncertainty

► Uncertainty in Events

Event-Condition-Action Rules

► ECA Rules

Event-Driven Business Process Management

Rainer von Ammon
Center for Information Technology Transfer GmbH (CITT), Regensburg, Germany

Synonyms

Workflow management; Business process monitoring

Definition

The term *Event-Driven Business Process Management* is a combination of actually two different disciplines: Business Process Management (BPM) and Complex Event Processing (CEP). The common understanding behind BPM is that each company's unique way of doing business is captured in its business processes. For this reason, business processes are today seen as the most valuable corporate asset. In the context of this entry, BPM means a software platform which provides companies the ability to model, manage, and optimize these processes for significant gain. As an independent system, Complex Event Processing (CEP) is a parallel running platform that analyses and processes events. The BPM- and the CEP-platform correspond via events which are produced by the BPM-workflow engine and by the IT services which are associated with the business process steps.

Historical Background

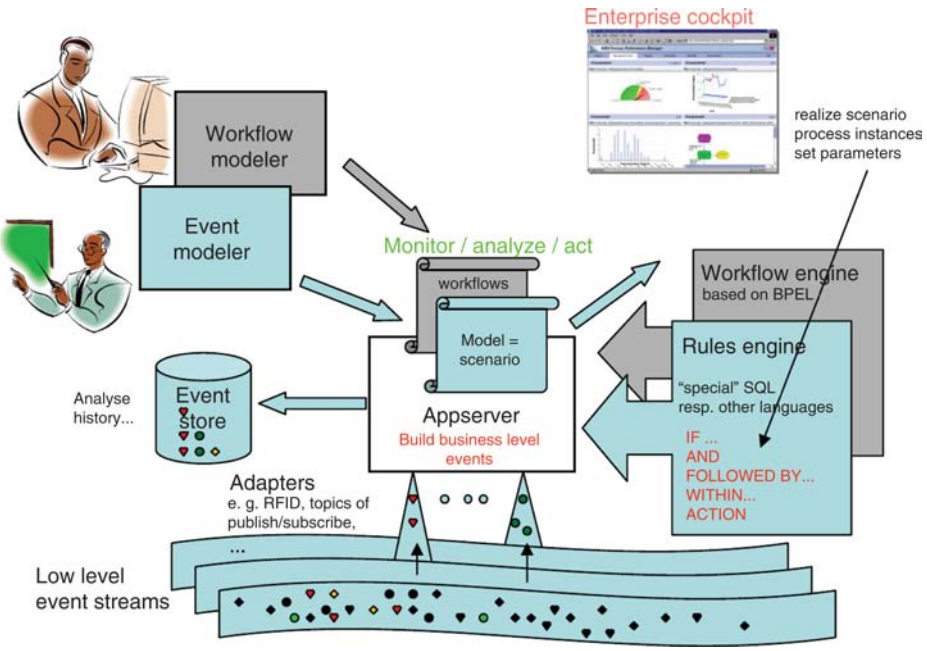
The term "Event Driven Business Process Management" was first used in June 2003 in a white paper of Bruce Silver Associates in connection with the FileNet P8-BPM platform [3]. The term was understood as a synthesis of workflow and Enterprise Application Integration (EAI). Actually, also a concept of event processing and real-time BAM was described in that white paper already, but only as single event processing without knowing anything about CEP which was not a discipline yet. Although the book "The power of events" of David Luckham [7] was already published in 2002, it wasn't well known until 2004 and there was no really working CEP platform so far. In November 2007, the term was explicitly used as a combination of BPM and CEP in an interview with Ruma Sanyal from BEA Systems [4].

Foundations

Figure 1 shows the principle of how a BPM- and BAM/CEP platform work together on the basis of events. The dark shaded components concern workflow respectively BPM, the light shaded components concern the real-time BAM/CEP components. Just started with the availability of commercial CEP platforms, there will be two different kinds of specialists in the future: the workflow modelers or business analysts and the event modelers.

The Business Process Modeler

The workflow modeler identifies business processes, starting from the value chain of an enterprise, he analyses and reengineers or optimizes the business processes by using a Business Process Analyzing-(BPA) tool based on a standard notation like the Business Process Modeling Notation (BPMN), invented 2002 by IBM and now standardized by the Object Management Group (OMG) since 2005, published February 2006 as version 1.0 [8]. A business process model is a graphically designed network based on symbols e.g., for flow objects like activities, gateways and also events. Depending on their position in the network, BPMN categorizes the events as start events, intermediate events and stop events. Each event category has different event types like timer-, message-, exception events etc. In the present release of BPMN 1.1, there are around 25 event types, the new release BPMN 2.0 is announced for Q3/2008. The modeling of business processes is a highly skilled task for a workflow modeler respectively a business analyst, but



Event-Driven Business Process Management. Figure 1. A reference model of the adoption of a CEP/BAM/BPM platform [10].

completely different from the task of an event modeler for real-time BAM/CEP. The event modeler acts with different kinds of events which are produced by the business process instances or from other event sources, like SNMP traps, RFID, log file entries etc.

[The business process models can be transformed into an executable notation of a workflow engine, respectively a BPM platform.] Executable notations are being standardized by OASIS as Business Process Execution Language (BPEL) or as a different standard XPDL by the Workflow Management Coalition (WfMC). To make a business process model really executable, it has to be remodeled more fine grained directly in the BPM platform. From the point of view of the IT department, for example, so called compensations and exceptions have to be modeled, if an IT-service cannot be executed or fails. This task cannot be done by the business analyst who does not know anything about the internals of IT services associated with the process steps. So, in the future as well as with the upcoming BPM platforms and the standards a new procedure for modeling and implementing business processes is needed. Business analysts from the operating departments must closely work with IT specialists, modeling the process and its associated IT services together. This is visualized as BPMN, but directly in an integrated modeling tool of the BPM platform. This is enhanced by a user interface for defining technical information (such as port-types, ports, partnerlinks, peoplelinks etc.), which is needed for the process execution, e.g., by a BPEL-engine. The upcoming BPM platforms are also able to produce events automatically when a service is called or cannot be executed and fails and so on. These are the event types the event modeler has to know for realizing BAM dashboards and enterprise cockpits.

The Event Modeler

By cooperating with the process owners of the operating departments or even with the C-level management of an enterprise, the event modeler has to define which BAM view has to be monitored in a dashboard, which alerts are to send to which roles in the organization and which actions shall be started automatically if a certain event pattern occurs. Derived from such BAM views, the event modeler looks for the needed event types and their instances flowing through the event streams of an enterprise or that are saved in an event store. It requires a high level of skill to define the right event patterns for a real-time BAM view. The event modeler has to know the different event sources like JMS messages realized on the basis of Publish/Subscribe topics etc. He has to install the corresponding event adapters delivered by the

CEP platform as "out of the box" prebuilt features or the event modeler has to care about the development of not yet existing adapters.

Event Processing Languages

The event modeler defines the event patterns for a BAM view on the basis of an Event Processing Language (EPL). At present, there is no standard for an EPL. The CEP community, founded as a discipline at the first CEP symposium in Hawthorne/New York in March 2006, is discussing the right standard for an EPL. It seems that there will be different EPL-approaches for different domains like Algorithmic Trading, BAM etc. At present, the CEP community is gathering use cases and is classifying them according to their corresponding domains [5]. The CEP platforms come with an SQL-like language (e.g., Coral8, Esper, Oracle, StreamBase) or provide a rule-based EPL approach (e.g., AMiT from IBM) or have an abstract user interface that hides any language and generates code like Java (e.g., Tibco, AptSoft).

The models of business processes and event scenarios are deployed into a middleware platform, e.g., into an application server, which is responsible for high availability, limitless scalability, grid computing, failover, transparency of heterogeneous infrastructures and so on.

Relation to the Database Technology

In connection with the term "Event-Driven BPM", there are two major challenges from the point of view of database technologies.

The challenge from the BPM side is storing millions of instances of long running business processes and their status. An enterprise such as a bank or the automotive industry has several thousand business processes like different types of an account opening for a depot, giro account, credit application, project budget confirmation etc. Each business process has several thousand process instances like the credit application of Betty Miller, John Smith etc. For long running processes, the BPM platform has to keep the status of the process instances, e.g., when waiting for the signature of Mrs. Miller or for the confirmation of the credit conditions by the bank. Because that can take days or weeks, there are millions of active business processes in a certain time window at the same time. That is the reason, for example, for overflows of queues in current projects.

The challenge from the BAM/CEP side is processing thousands of events per second generated by business processes and their associated IT-services by in-memory databases, realizing time windows. A memory-centric SQL engine is designed to execute queries without ever writing data to disk. Such an engine also provides a persistence feature for storing the data as "historic events" in an event store for later processing, i.e., for aggregating and correlating them with the current events of the event streams. The basic idea is to route an event through a lot of different in-memory filters, to see what queries it satisfies, rather than executing many queries in sequence against disk-based data. That is a precondition for realizing real-time BAM or the idea of "predictive business" [9].

Examples

If an online credit system receives 50,000 or 100,000 credit applications per day, the bank likes to monitor the application process by typical event patterns that signal, for example, that a credit application could be cancelled within the next moment. In this case, the system shall trigger an action to keep the customer. For this purpose, the event modeler has to identify the corresponding events. If not available from the implemented process, he has to take care of the generation of additional appropriate events. This is a demanding, challenging task.

As another example, BAM/CEP shall send an alert to an employee of the bank, because the loss by cancellations exceeds a certain threshold. However, it wouldn't make sense to send an alert each time a cancellation happens, because too many alerts may result, and the VIRT problem (valuable information at the right time [6]) arises.

Key Applications

Key applications will take place in all domains and first projects on the basis of *Event Driven BPM*-platforms just start 2008:

Logistics applications, e.g., at DHL/Deutsche Post [5].

Finance applications, e.g., Deutsche Bank, Team-Bank [2].

Telco applications, e.g., Deutsche Telekom, T-Mobile.

Current/future research projects, e.g., «Domain specific reference models for event patterns for a faster set-up of BPM/BAM applications» [1].

Cross-references

▶ Business Activity Monitoring
▶ Complex Event Processing
▶ Event Driven Architecture
▶ Event Stream Processing
▶ Service Oriented Architecture

Recommended Reading

1. Ammon R.V., Silberbauer C., and Wolff C. Domain specific reference models for event patterns – for faster developing of business activity monitoring applications. In Proc. VIP Symp. on Internet Related Research, 2007.
2. Brandl H.-M. and Guschakowski D. Complex Event Processing in the Context of Business Activity Monitoring. An Evaluation of Different Approaches and Tools Taking the Example of the Next Generation Easycredit. Diploma thesis, 2007. http://www.citt-online.de/downloads/Diplomarbeit_BaGu_Final.pdf
3. Bruce Silver Associates. FileNet P8 – event-driven business process management. Industry trend reports, June 2003. Available at: http://www.ecm-unverzichtbar.de/Repository/48/files/69_EvBPM.pdf; http://www.complexevents.com/
4. Danielsson K. and Trotta G. Key requirements for event-driven BPM and SOA. Available at: http://www.ebizq.net/topics/bpm/features/8700.html
5. Emmersberger C. and Springer F. Event Driven Business Process Management. An Evaluation of the Approach of Oracle and the Sopera Open Source Framework. Diploma thesis, 2008. http://www.citt-online.de/downloads/EmmSpr_Diplomarbeit_Final.pdf
6. Hayes-Roth F. Value information at the right time (VIRT): Why less volume is more value in hastily formed networks. Available at: www.nps.edu/cebrowski/Docs/VIRTforHFNs.pdf, 2007.
7. Luckham D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Reading, MA, 2002.
8. OMG/BPMI. Business process modeling notation (BPMN) information. Available at: http://www.bpmn.org/
9. Ranadive V. The Power to Predict: How Real Time Businesses Anticipate Customer Needs, Create Opportunities and Beat the Competition. McGraw-Hill, New York, NY, 2006.
10. Use cases of CEP applications. Available at: http://complexevents.com/?cat=16

Eventual Consistency

Marc Shapiro[1], Bettina Kemme[2]
[1]INRIA Paris-Rocquencourt and LIP6, Paris, France
[2]McGill University, Montreal, QC, Canada

Definition

In a replicated database, the consistency level defines whether and how the values of the replicas of a logical object may diverge in the presence of updates. Eventual consistency is the weakest consistency level that guarantees useful properties. Informally, it requires that all replicas of an object will eventually reach the same, correct, final value, assuming that no new updates are submitted to the object.

Key Points

Eventual consistency is an important correctness criterion in systems with a lazy, update-anywhere strategy, also referred to as *optimistic replication* (q.v.). Update operations can be submitted and executed on any node, and the propagation of updates occurs lazily after commit. Conflict resolution and reconciliation must ensure that all replicas (copies) of a logical object eventually converge to the same value. Different objects are considered independent.

In a system where updates are continuously submitted, eventual consistency can be defined by a weak form of schedule equivalence [1]. A schedule S_n^x describes the sequence of update operations node n performs on its replica of object x. An element of the schedule of the form w_i represents the execution of an update to object x, submitted by some user. S_n^x contains an element of the form $\overline{w_i}$, if the update w_i was received by n, but either not executed, or aborted due to conflict resolution.

Typically, two schedules are defined equivalent by restricting how the order of operations in the two schedules may differ. However, for eventual consistency only the convergence of object values matters. Thus, equivalence is defined by comparing the final state of the replicas. Two schedules are said *state equivalent* when, starting from the same initial state, they produce the same final state. For instance: (i) schedules $S = w_1 w_2$ and $S' = w_2 w_1$ are state-equivalent if w_1 and w_2 commute; (ii) schedules $S = w_1 w_2$ and $S' = w_2$ are state-equivalent if w_2 sets the state of the object to a completely new value (e.g., $x := 2$).

Eventual consistency of a replicated object x is defined by the following conditions, which must hold at all times, at any node n with a replica of x [1]. It is assumed that all replicas have the same initial state. There is a prefix of the schedule S_n^x that contains the same operations and is state-equivalent to a prefix of the schedule $S_{n'}^x$ of any other node n' holding a replica of x. Such a prefix is called a *committed prefix* of S_n^x.

- The committed prefix of S_n^x grows monotonically over time, i.e., the extant set of operations and their relative order remains unchanged.
- For every operation w_i submitted by a user, either w_i or $\overline{w_i}$ eventually appears in the committed prefix of S_n^x (but not both, and not more than once).
- An operation of the form w_i in the committed prefix satisfies all its preconditions (e.g., the state of the object immediately before the execution of the operation fulfills certain conditions).

As an example, assume operation w_1 sets x to 2, and w_2 sets it to 5. Operation w_1 is submitted and executed at n_1 while w_2 is first executed at n_2. At this time the local schedules are $S_1 = w_1$ and $S_2 = w_2$ and the committed prefix at both nodes is the empty schedule. Now w_1 is propagated to n_2 and w_2 is propagated to n_1. When n_1 receives w_2, it detects that w_1 and w_2 are concurrent and conflict. Say that conflict reconciliation prioritizes one of the operations, e.g., w_1. Then, w_2 is simply not executed and the new schedule is $S_1^x = w_1\overline{w_2}$. At n_2, when w_1 arrives, the conflict is also detected, w_2 is undone, w_1 is executed and the final schedule is $S_2 = \overline{w_2}w_1$. At this time, S_1 and S_2 are themselves the committed prefixes. Note that further concurrent operations on x might move the schedules further, but the extensions would still be tentative and only become committed once they are reconciled at all replicas.

Cross-references

▶ Concurrency Control – Traditional Approaches
▶ Data Replication
▶ Distributed Concurrency Control
▶ Distributed database systems
▶ Middleware Support for Database Replication and Caching
▶ One-Copy-Serializability
▶ Optimistic Replication and Resolution
▶ Replica freshness
▶ Replicated Database Concurrency Control
▶ Strong Consistency Models for Replicated Data
▶ WAN Data Replication
▶ Weak Consistency Models for Replicated Data

Recommended Reading

1. Saito Y. and Shapiro M. Optimistic replication. ACM Comput. Surv., 37(1):42–81, 2005.

Evidence Based Medicine

Michael Weiner
Indiana University School of Medicine, Indianapolis, IN, USA

Synonyms

Evidence based practice; Scientific medicine

Definition

Evidence based medicine (EBM) is the "conscientious, explicit, and judicious use of current best evidence in making decisions about the care of individual patients" [1].

Key Points

Although medical practice is historically based on observation and best attempts at scientific method, early medical practices often involved trial and error or observations of one or a few cases. In the last century, advances in research, experience, technology, and communications have moved medicine more towards practice based on established evidence.

Individuals especially responsible for promoting, developing, describing, and discussing EBM have included Archie Cochrane, David Sackett [4], and Gordon Guyatt [2]. The international Cochrane Collaboration [3] has emerged to continue to advance EBM throughout the world. This and several other organizations, such as the US Preventive Services Task Force [1], have assembled, analyzed, and graded collections of medical evidence. Specific evidence reported from such groups often has gradings that indicate the strength of evidence about a topic, based on quality or scope of methods, statistics, reproducibility of findings, etc.

The representation of medical evidence in information systems or databases can assume various forms, especially because the type of evidence is likely to vary substantially from one topic to the next. Nevertheless, the use of standard clinical research methods, such as randomized controlled trials, survival analysis, and meta-analysis, can facilitate knowledge management and representation by limiting the types of information that need to be stored or communicated. Automated extraction of medical knowledge from research reports may benefit from natural language processing when reports are not more structured.

EBM has several positive aspects. It can help to increase adherence to clinical guidelines and minimize variations in medical practice that may be viewed as too large or experimental. It can help to reduce disparities and establish a minimum standard of care for a population. To some degree, EBM seems to hold practitioners and researchers more accountable for their actions and pursuits and also distributes the knowledge among a larger group of professionals and consumers. EBM promotes the development of further research.

Overreliance on EBM may stifle creativity, or it may fail to take into account an individual's preferences or characteristics that may not "fit the mold" or may require special handling not covered by a guideline or established evidence. Thus, practicing EBM requires the ability to find, understand, interpret, and appropriately apply evidence with individual patients or groups. The Internet is increasingly a useful source of EBM knowledge, via databases, literature reviews, and communications.

The rise of EBM has been associated with corresponding local and federal health policies. For example, medical institutions may prohibit certain procedures without adequate evidence of benefit. Not only the practice of medicine, but the reimbursement of modern medical care, often depends on evidence.

Cross-references

- ▶ Classification
- ▶ Clinical Data Acquisition, Storage and Management
- ▶ Indexing of Data Warehouses
- ▶ Information Extraction
- ▶ Storage Management
- ▶ Text Mining
- ▶ Visual Content Analysis
- ▶ Web Information Extraction

Recommended Reading

1. Agency for Healthcare Research and Quality. U.S. Preventive Services Task Force (USPSTF). 2007. Available online at http://www.ahcpr.gov/clinic/uspstfix.htm (accessed Aug 30, 2007).
2. Evidence-Based Medicine Working Group. Evidence-based medicine. A new approach to teaching the practice of medicine. JAMA. 268(17):2420–2425, 1992.
3. The Cochrane Collaboration. The Cochrane Collaboration. 2007. Available online at http://www.cochrane.org/ (accessed Aug 30, 2007).
4. Sackett D.L., Rosenberg W.M., Gray J.A., Haynes R.B., and Richardson W.S. Evidence based medicine: what it is and what it isn't. BMJ. 312(7023):71–72, 1996.

Evidence Based Practice

▶ Evidence Based Medicine

Evolutionary Algorithms

▶ Genetic Algorithms

Evolutionary Computation

▶ Genetic Algorithms

Evolutionary Semantics

▶ Emergent Semantics

Exactly Once Execution

▶ Application Recovery

Executable Knowledge

Mor Peleg
University of Haifa, Haifa, Israel

Synonyms

Computer-interpretable formalism; Knowledge-based systems

Definition

Executable knowledge is represented in a symbolic formalism that can be understood by human beings and interpreted and executed by a computer program. It allows a computer program to match case data to the knowledge, reason with the knowledge, select recommended actions that are specific to the case data, and deliver them to users. Executed knowledge can be delivered in the form of advice, alerts, and reminders, and can be used in decision-support or process management.

Historical Background

Representing knowledge in a computer-interpretable format and reasoning with it so as to support humans in decision making started to be developed by the artificial intelligence community in the 1970s. According to Newell [6], knowledge is separate from its representation. At the knowledge level, an agent has as parts bodies of knowledge, actions, and goals. An agent processes its knowledge to determine the actions to take. An agent, behaving through the principal of rationality, selects those actions that attain his goals. Representing knowledge in a symbolic way (i.e., data structures and algorithms) allows the expression of knowledge in a way that would be understandable by humans and would allow people and computer programs to reason with the knowledge and select actions in the service of goals. According to Davis et al. [1], knowledge representation serves five roles: (i) a surrogate to enable an entity to determine the consequences of thinking, (ii) a set of ontological commitments about how and what to see in the world, (iii) a fragmentary theory of intelligent reasoning, (iv) a medium for efficient computation, and (v) a medium for human expression.

Research on knowledge representations started to evolve in the 1970s with rule-based systems. One of the most famous rule-based systems is Mycin [11]. This system represented clinical knowledge about diagnosis and treatment of infectious diseases and allowed clinician users to enter patient findings and, through execution of the if-then-else rules, arrive at probable diagnoses and appropriate treatments. In the 1980s decision-theoretic models, such as Bayesian Networks, influence diagrams, and decision trees [7] started to become popular ways of representing and reasoning with decision knowledge in a probabilistic way, under uncertainty.

By the late 1980s, researchers started to realize the importance of integrating knowledge-based systems with databases. In this way, the case data coming from a database could drive the execution of encoded knowledge. An example of an early formalism that addressed data integration is the Arden Syntax for Medical Logic Modules (MLMs) [4], discussed in detail below.

In the 1990, ontologies [3] were developed to formalize a shared understanding of a domain. In knowledge engineering, the term ontology is used to mean definitions of concepts in a domain of interest and the relationships among them ("a specification of a conceptualization of a domain" [3]). An ontology enables software applications and humans to share and reuse the knowledge consistently. Ontologies, as represented in a formal language such as frames or description logic, allow logical inference over the set of concepts and relationships to provide decision support and explanation facilities. Ontologies are much more maintainable than rule-based systems. In rule-based systems, the knowledge is represented as individual rules. It is difficult to foresee the affect of addition, deletion, or modification of a rule on the performance of the rule-based system. This problem is solved by ontologies. Research in the 1990s, led by Musen [5], addressed modeling of domain knowledge and problem-solving methods as ontologies that could be combined together, such that a problem solving method could be applied to several domain ontologies, and decision-support systems in a single domain could utilize several generic problem-solving methods.

In the 1990s, a new generation of executable knowledge-based systems started to be developed by two separate communities: the medical informatics community and the business process community. The common theme to the new developments by both of these communities was that the executable knowledge was no longer representing individual decisions, but rather a process that unfolds over time and includes many activities, and the fact that the knowledge-based process system had to be integrated with other systems in the organization. The process modeling languages in both communities were developed as ontologies.

Foundations

Executable knowledge formalisms were developed by several communities, including the artificial intelligence community, the medical informatics community, and the business process management community. The rest of this article reviews work done by the medical informatics and business process communities.

Following the success of rule-based decision-support systems, and at the same time, recognizing the advantages of linking a knowledge-based system to case data, a standard for encoding individual medical decisions, known as the Arden Syntax for Medical Logic Modules (MLMs) [4], was defined in 1989. Arden Syntax was developed initially under the sponsorship of the American Society for Testing and

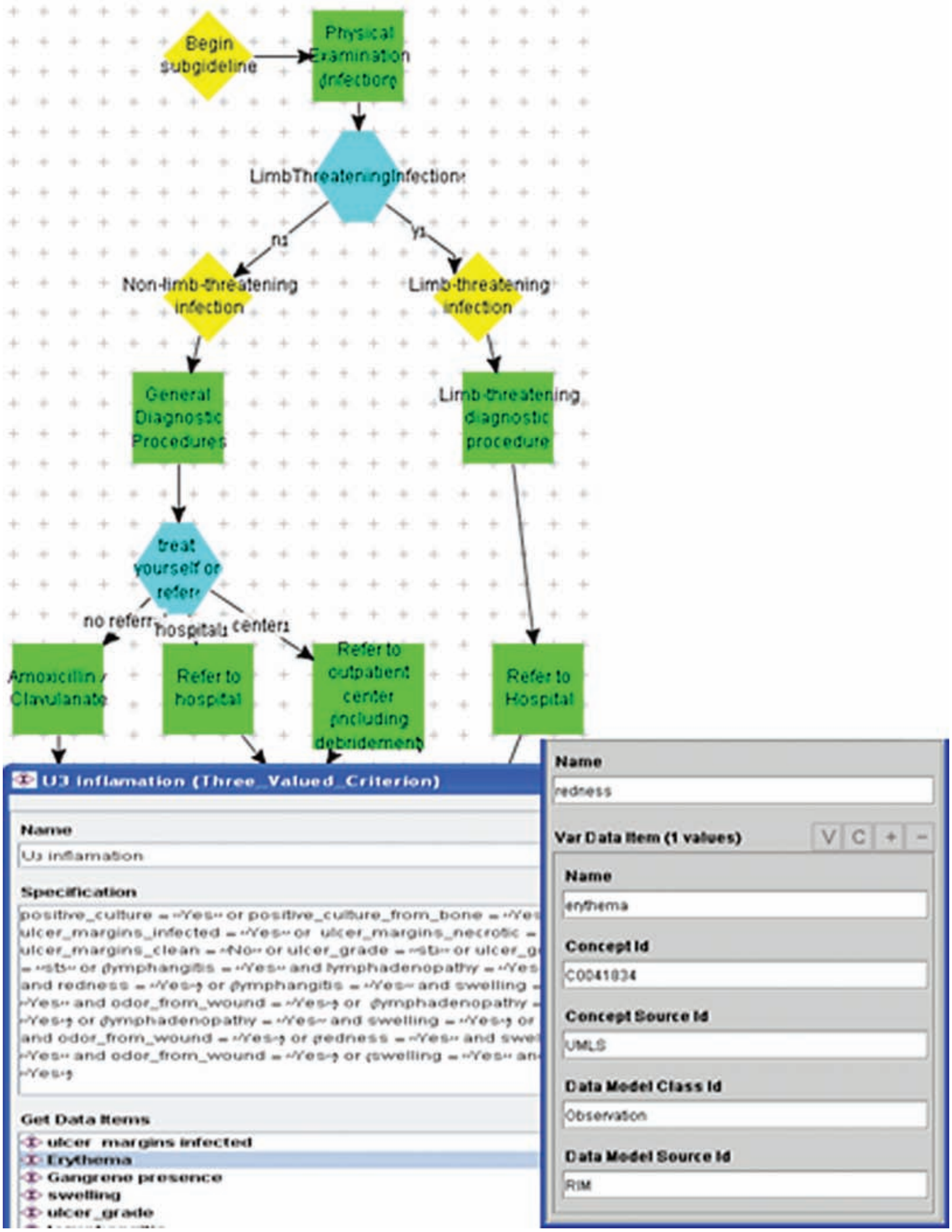
Materials and subsequently of Health Level Seven (HL7). Arden Syntax is a published as an American National Standards Institute (ANSI) standard. MLMs, in Arden Syntax, define decision logic via a knowledge category that has data, evoke, logic, and action slots. The event, logic and action slots specify respectively the events that trigger the MLM (e.g., storage of a new serum potassium test result into a database), the logical criterion that is evaluated (e.g., serum potassium value <3.5), and the action that is performed if the logical criterion holds, which is often an alert or reminder (e.g., potassium replacement therapy). The data slot specifies mappings between the specific database records and the MLM's variables. However, only part of this specification is defined by the syntax, as it does not contain standard terminology or a data model for electronic medical records.

In the 1990s, decision-support systems whose knowledge is based on evidence-based clinical guidelines started to be developed. Clinical guidelines are recommendations that are developed by healthcare organizations based on evidence from clinical trials, and are aimed at assisting practitioner and patient decisions about appropriate healthcare for specific clinical circumstances. Unlike individual clinical decisions, such as MLMs, clinical guidelines involve multi-step decisions and actions that unfold over time. Guideline-based decision-support systems help clinicians in the process of patient care, including decision making and task management. These systems are based on process-flow ontologies termed Task-Network Models (TNM) [10] – a hierarchical decomposition of guidelines into networks of component tasks that unfold over time. The task types vary in different TNMs, yet all of them support modeling of medical actions, decisions, and nested tasks. These models contain computer-interpretable specifications of decision criteria and clinical actions that enable an execution engine to interpret the guideline representation and execute it for a given patient case data. Important themes in executable guideline models include the ability of some of them to integrate with electronic databases (e.g., electronic health record systems (EHR)), using standard terminologies to express medical actions (e.g., laboratory tests, drug prescriptions) and patient data items upon which decision criteria are written, using standard expression languages for writing decision criteria, and using messaging standards for exchange of (clinical) data.

In 2003, researchers from six groups that developed TNMs participated in a study that compared their models using two clinical guidelines that served as case studies [10]. The TNMs studied were Asbru, EON, GLIF, Guide, PRODIGY, and PRO*forma*. An example of a guideline model represented in GLIF is shown in Fig. 1. Although these formalisms all depict a guideline as a TNM, they each have their own emphasis. Asbru emphasizes specifying intentions as temporal patterns; EON views the guideline model as the core of an extensible set of models, such as a model for performing temporal abstractions. EON uses a task-based approach to define decision-support services that can be implemented using alternative techniques; GLIF emphasizes the ability to share and integrate guideline specifications among software tools and implementing institutions; GUIDE focuses on integration with organizational workflow using a workflow-based model and linkage to decision-theoretic models, PRODIGY (a project that is no longer active) aimed at producing the simplest, most readily comprehensible model necessary to represent chronic disease management guidelines. It models guidelines as decision maps organized as a collection of clinical contexts; in each context, selection among relevant clinical actions is made; PRO*forma* advocates the support of safe guideline-based decision support and patient management by combining logic programming and object-oriented modeling. Its syntax and semantics are formally defined. One aim of the PRO*forma* project is to explore the expressiveness of a deliberately minimal set of modeling constructs: actions, compound plans, decisions, and inquiries of patient data from a user.

The study compared the TNM models in term of eight components that capture the structure of computerized guidelines: (i) organization of guideline plans, (ii) goals, (iii) model of guideline actions, (iv) decision model, (v) expression language, (vi) data interpretation/abstractions, (vii) medical concept model, and (viii) patient information model.

The purpose of the study was to find consensus among the different formalisms that could be a starting point for creating a standard computerized guideline formalism. Differences between the guideline modeling languages were most apparent in underlying decision models (ranging from simple switching constructs to argumentation rules for and against decision alternatives, and even use of decision-theoretic models such as influence diagrams and decision trees), goal


Begin subgideline
Physical Examination (infection)
LimbThreateningInfection
no
yes
Non-limb-threatening infection
Limb-threatening infection
General Diagnostic Procedures
Limb-threatening diagnostic procedure
treat yourself or refer
no refer
hospital
center
Amoxicillin / Clavulanate
Refer to hospital
Refer to outpatient center (including debridement)
Refer to Hospital

U3 inflamation (Three_Valued_Criterion)

Name

U3 inflamation

Specification

positive_culture = «Yes» or positive_culture_from_bone = «Yes
ulcer_margins_infected = «Yes» or ulcer_margins_necrotic =
ulcer_margins_clean = «No» or ulcer_grade = «s3» or ulcer_g
= «s5» or (lymphangitis = «Yes» and lymphadenopathy = «Yes
and redness = «Yes») or (lymphangitis = «Yes» and swelling =
«Yes» and odor_from_wound = «Yes») or (lymphadenopathy =
«Yes») or (lymphadenopathy = «Yes» and swelling = «Yes») or
and odor_from_wound = «Yes») or (redness = «Yes» and swel
«Yes» and odor_from_wound = «Yes») or (swelling = «Yes» an
«Yes»)

Get Data Items

- ulcer margins infected
- Erythema
- Gangrene presence
- swelling
- ulcer_grade

Name

redness

Var Data Item (1 values)

Name

erythema

Concept Id

C0041834

Concept Source Id

UMLS

Data Model Class Id

Observation

Data Model Source Id

RIM

Executable Knowledge. Figure 1. Part of a GLIF model for a Diabetic Foot guideline, modeled using the Protégé-200 tool. Squares denote actions, diamonds – patient state steps, hexagons – decisions. The bottom part of the figure shows the computable specification of the decision step "Limb-threatening Infection." The decision criterion is based on values of patient data items. Each patient data item is defined using a concept code taken from a controlled medical vocabulary (Unified Medical Language System (UMLS) in this case) and by a patient data model class (e.g., Observation) taken from a data model source, such as Health Level 7's Reference Information Model (RIM).

representation, use of scenarios (as a plan component that defines a particular patient management context that serves as entry points into guidelines), and structured medical actions which could be mapped into controlled vocabulary terms.

Consensus was found in plan organization (plans could be nested and structured as plan components arranged in sequence, in parallel, and in iterative and cyclic structures), expression language for specifying and sharing decision and eligibility criteria, patient state definitions, and preconditions on system actions, conceptual medical record model, medical concept model, and data abstractions (i.e., definitions of abstract terms using mathematical functions of other concepts, temporal abstractions, and concept hierarchies that allow reasoning at different levels of the hierarchy).

The HL7 Clinical Decision Support Technical Committee (CDSTC) has focused on standardization of two of the components for which consensus was found: expression language and conceptual virtual medical record (vMR) model. The object-oriented guideline expression language, GELLO [12], is an extensible guideline expression language that can be used for formally defining decision and eligibility criteria, as well as patient states. It is based on the Object Constraint Language (OCL) (http://www-306.ibm.com/software/rational/uml/resources/documentation.html). In 2004, it was established as a standard of HL7. The CDSTC started the process of standardizing a vMR, based on experiences with the patient information models of PRODIGY, EON and the HL7 RIM, which is also the basis of GLIF's default patient information model. An object oriented vMR would ease the process of mapping guideline patient data items to real EMRs, allowing decision criteria, eligibility criteria and patient states to be defined by in guideline models by reference to the VMR rather than specific EMRs.

The second community that started to develop a new generation of executable knowledge-based systems is the business process management community. Workflow management systems started to be developed based on workflow formalisms that define a model of business processes and a model of the organization, including its individual actors, roles, organizational units, and resources that participate in workflow activities. Several industrial groups defined standards for workflows. The Web Services Business Process Execution Language (BPEL) by the Advanced Open Standards for the Information Society (OASIS), is an executable formalism where the business process behavior is based on Web Services. XML Process Definition Language (XPDL) is a standard that is under development since 1998 by the Workflow Management Coalition – An industry group dedicated to creating software standards for workflow applications. The goal of XPDL is to store and exchange the process diagrams among different workflow tools, including workflow modeling tools (editors) and workflow engines. In 2004 the Workflow Management coalition endorsed the graphical workflow standard called Business Process Modeling Notation (BPMN) developed by Object Management Group, to standardize the way that process definitions are visualized.

The business process management community has developed many tools that enable modeling, analyzing, verifying, simulating, and executing a business process. While the medical informatics community devoted much research to supporting clinical decision making in addition to patient care task management, the business process community made much progress on modeling task management within an organization, where the business process often involves many departmental units and organizational roles. Workflow systems aim to help in resource management, a task that is not addressed by current executable guideline systems.

Key Applications

A generic free and open-source tool for creating ontologies, or knowledge-bases, is Protégé (http://protege.stanford.edu). The Protégé platform supports two main ways of modeling ontologies via the frames and Ontology Web Language (OWL) formalisms. Protégé is implemented in the Java programming language and is extensible. Protégé-frames knowledge bases could be reasoned with several rule-engines, such as Jess and Algernon. Additionally, the Protégé Axiom Language could be used to define first-order-logic constraints and check them for instances in the knowledge base. Several reasoners can be used with Protégé-OWL to classify instances or classes according to class definitions and class hierarchies. They include the commercial product Racer, and the commonly used free reasoners Pellet and Fact + +. The SWRL rule-based engine could also be used to reason with Protege-OWL.

The Arden Syntax for Medical Logic Modules has been applied by universities such as Columbia

University Medical School and by several health information systems vendors, such as Micromedex, Eclypsis Corporation, McKesson Information Solutions, and Siemens Medical Solutions Health Services Corporation to create MLMs that deliver clinical recommendations in the form of alerts and reminders. MLMs are being used by many clinical institutions in the United States (see http://cslxinfmtcs.csmc.edu/hl7/arden/ for a partial list). The Department of BioMedical Informatics at Columbia University developed a knowledgebase of over 240 MLMs that are now available from http://cslxinfmtcs.csmc.edu/hl7/arden/.

Many tools exist to support guideline development, modeling, verification, and execution [2,8]. Asbru modeling tools include Delt/A (http://ieg.ifs.tuwien.ac.at/projects/delta/) and URUZ, both focusing on easing the transition from narrative to formal representations via a mark-up stage, AsbruView (http://www.ifs.tuwien.ac.at/asgaard/asbru/tools.html), which focuses on visualization and user interface for authoring, and CareVis (http://ieg.ifs.tuwien.ac.at/projects/carevis/), which provides multiple simultaneous views to cover different aspects of a complex underlying data structure of treatment plans and patient data. Verification of Asbru guidelines can be done using formal verification methods [13]. Implementations in Asbru were developed for diabetes, artificial ventilation, and breast cancer guidelines.

EON guidelines can be authored in the knowledge-modeling tool Protégé-2000 and executed by an execution engine that uses a temporal data mediator to support queries involving temporal abstractions and temporal relationships. A third component provides explanation services for other components. EON has been used to create hypertension and opiods guidelines that are implemented in various hospitals and clinics of the Veteran Affairs Hospital. Protégé-2000 is also the modeling tool for GLIF guidelines. The GLIF execution engine (GLEE) has been used to implement two guidelines: diabetic foot diagnosis and management and flu vaccinations.

Guide has a new implementation of an authoring tool and an execution engine called NewGuide. Guide has been used to implement guidelines for stroke and for management of patients with heart failure.

A number of software tools have been created for creating, visualizing, and executing PRO*forma* guidelines. They include Arezzo and Tallis. Several PRO*forma* guidelines have been implemented and some have undergone clinical trials to establish their safety and utility. More information on PRO*forma* implementations as well as on implementations of other guideline formalisms can be found at the open clinical web site (www.openclinical.org) – a repository of resources about decision support, clinical workflow and other advanced knowledge management technologies for patient care and clinical research.

Key application for workflow management systems include tools for modeling and executing workflows, such as the open-source workflow tool Bonita (http://bonita.objectweb.org), FLOWer (http://www.workflow-download.com/workflow/flower.html), YAWL (http://www.yawl-system.com/), and Together Workflow Editor andserver(http://www.together.at/together/prod/twe/), and Oracle Workflow (http://www.oracle.com/technology/products/integration/workflow/workflow_fov.html). Other tools exist for verifying [15] workflows (Oracle Worklow). There are business process management engines available from several vendors including Software AG, Savvion, Lombardi, Appian, JBoss, and Tibco.

Future Directions

The challenge of creating executable knowledge that can be shared by multiple implementing institutions and mapped to their information systems started to be addressed by projects such as the Arden Syntax, GLIF, Shareable Active Guideline Environment (SAGE) [14] and Knowledge-Data Ontology Mapper [9], yet more work needs to be done in this area to define how knowledge can be authored in a way that is institution-specific and sharable.

One of the most interesting future directions of executable knowledge concerns synergetic development that draws upon developments made in the medical informatics and the business process communities. Such collaborations are emerging, as manifested by health-care related workshops that are taking place in business-process management and information systems conferences, such as the ProHealth Workshop, which is part of the Business Process Management conference.

Cross-references

- ▶ Bayesian Classification
- ▶ Ontologics
- ▶ OWL: Web Ontology Language
- ▶ Rule-Based classification

Recommended Reading

1. Davis R., Shrobe H., and Szolovits P. What is a knowledge representation? AI Magazine, 14(1):17–33, 1993.
2. de Clercq P.A., Blom J.A., Korsten H.H.M., and Hasman A. Approaches for creating computer-interpretable guidelines that facilitate decision support. Artif. Intell. Med., 31:1–27, 2004.
3. Gruber T.R. Toward principles for the design of ontologies used for knowledge sharing. Int. J. Human Comput. Stud., 43:907–928, 1995.
4. Hripcsak G., Ludemann P., Pryor T.A., Wigertz O.B., and Clayton P.D. Rationale for the arden syntax. Comput. Biomed. Res., 27(4):291–324, 1994.
5. Musen M.A., Tu S.W., Eriksson H., Gennari J.H., and Puerta A.R. PROTEGE-II: an environment for reusable problem-solving methods and domain ontologies. In Proc. 13th Int. Joint Conf. on AI, 1993.
6. Newell A. The knowledge level. AI Magazine, 2(2):1–20, 33, 1980.
7. Pearl J. Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, San Francisco, CA, 1988.
8. Peleg M. Guideline and workflow models. In Clinical Decision Support – The Road Ahead, R.A. Greenes (ed.). Elsevier/Academic Press, Orlando, FL, 2006.
9. Peleg M., Keren S., and Denekamp Y. Mapping computerized clinical guidelines to electronic medical records: knowledge-data ontological mapper (KDOM). J. Biomed. Inform., 41(1):180–201, 2008.
10. Peleg M., Tu S.W., Bury J., Ciccarese P., Fox J., Greenes R.A., et al. Comparing computer-interpretable guideline models: a case-study approach. J. Am. Med. Inform. Assoc., 10(1):52–68, 2003.
11. Shortliffe E.H. Computer-Based Medical Consultations: Mycin. Elsevier/North Holland, New York, 1976.
12. Sordo M., Ogunyemi O., Boxwala A.A., Greenes R.A., and Tu S. GELLO: a common expression language. Available online at: http://cslxinfmtcs.csmc.edu/hl7/arden/2004–09-ATL/v3ballot_gello_aug2004.zip, 2004.
13. ten Teije A., Marcos M., Balser M., van Croonenborg J., Duelli C., van Harmelen F., et al. Improving medical protocols by formal methods. Artif. Intell. Med., 36(3):193–209, 2006.
14. Tu S.W., Campbell J.R., Glasgow J., Nyman M.A., McClure R., McClay J.P.C., Hrabak K.M., Berg D., Weida T., Mansfield J.G., Musen M.A., and Abarbanel R.M. The SAGE guideline model: achievements and overview. J. Am. Med. Inform. Assoc., 14(5):589–598, 2007.
15. van der Aalst W.M.P. The application of petri nets to workflow management. J. Circuits Syst. Comput., 8(1):21–66, 1998.

Execution Skew

Nikos Hardavellas, Ippokratis Pandis
Carnegie Mellon University, Pittsburgh, PA, USA

Definition

Execution skew is a phenomenon observed during the parallel evaluation of a database query, in which the concurrent operators exhibit disparate execution times.

Key Points

Execution skew is a phenomenon observed in the parallel evaluation of a database query. It arises when there are imbalances in the execution of the operators running in parallel, resulting in some of the operators running for a longer time than others. The differences in execution times may cause some of the processors to remain idle while others still compute a part of the query. Execution skew can be a consequence of other forms of skew within a query, e.g., data skew, or arise because of temporally unavailable resources that affect the execution speed of a unit of work. The database system can minimize execution skew through the careful allocation of processors to operators and the selection of the appropriate parallel query plan. Execution skew arises in both operator-level parallelism as well as intra-operator parallelism. The interested reader is referred to [1] which presents a classification of skew effects in parallel database systems.

Cross-references

- ▶ Data Skew
- ▶ Intra-Operator Parallelism
- ▶ Operator-Level Parallelism

Recommended Reading

1. Märtens H. A Classification of Skew Effects in Parallel Database Systems. In Proc. 7th Int. Euro-Par Conference, 2001, pp. 291–300.

Exhaustivity

- ▶ Relevance

Existence Time

▶ Lifespan

Explicit Event

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Definition

In active databases, an explicit event is explicitly specified in the ECA rule definition.

Key Points

ECA rules were developed as an optimization of condition action rules. The performance of rule evaluation was improved by allowing, or even requiring, explicit definition of when rules should be triggered in the form of events. For example, (**ON** *update*$_A$ **followed_by** *update*$_B$ **IF** $A = 5 \wedge B = 3$ **DO** action) is a rule specification that is triggered by the explicit event *update*$_A$ **followed_by** *update*$_B$ (i.e., a sequence of updates).

Cross-references

▶ Atomic Event
▶ Composite Event
▶ ECA Rules
▶ Event
▶ Event Detection
▶ Event Specification
▶ Implicit Event

Exploratory Data Analysis

Hans Hinterberger
ETH Zurich, Zurich, Switzerland

Definition

Exploratory Data Analysis (EDA) is an *approach* to data analysis that employs a number of different techniques to:

1. Look at data to see what it seems to say,
2. Uncover underlying structures,
3. Isolate important variables,
4. Detect outliers and other anomalies,
5. Suggest suitable models for conventional statistics.

Key Points

The term "Exploratory Data Analysis" was introduced by John W. Tukey who in [2] shows how simple graphical and quantitative techniques can be used to open-mindedly explore data.

Typical graphical techniques are

1. Plotting the raw data (e.g., stem-and-leaf diagrams, histograms, scatter plots)
2. Plotting simple statistics (e.g., mean plots, box plots, residual plots)
3. Positioning (multiple) plots to amplify cognition

Typical quantitative techniques are

1. Interval estimation
2. Measures of location or of scale
3. Shapes of distributions

Exploratory data analysis can help to improve the results of statistical hypothesis testing by forcing one to look at data unbiased before formulating hypotheses which are subsequently tested using conventional statistics (confirmatory data analysis). Its techniques allow analysts to better assess assumptions on which statistical inference will be based and support the selection of appropriate statistical tools. EDA can also provide a background for further data collection.

For users of database systems it might be interesting to note that many EDA techniques have been adopted to support the exploration stage of data mining [1]. In particular, they are being used to identify the most relevant variables and to determine the complexity and general nature of models that can be taken into account for the model building and validation stages. Data mining and EDA are complementary in the sense that the former is more oriented towards applications and the later can help understand the basic nature of the underlying phenomena.

Cross-references

▶ Data Visualization

Recommended Reading

1. Berry M.J.A. and Linoff G.S. Mastering Data Mining. Wiley, New York, 2000.
2. Tukey J.W. Exploratory Data Analysis. Addison Wesley, Reading, MA, 1977.

Exploring

▶ Browsing in Digital Libraries

Expressive Power of Query Languages

Leonid Libkin
University of Edinburgh, Edinburgh, UK

Definition

The study of expressive power concentrates on comparing classes of queries that can be expressed in different languages, and on proving expressibility – or *in*expressibility – of certain queries in a query language.

Historical Background

Ever since Codd proposed relational calculus (first-order predicate logic) as a basic relational query language, it has been common for database query languages to have limited expressiveness. If a language cannot express everything computable, then it is natural to ask:

1. What queries cannot be expressed in a language $\mathcal{L}$?
2. Which methods are available for proving such results?

Furthermore, if there are two query languages $\mathcal{L}_1$ and $\mathcal{L}_2$, one may want to compare their expressiveness: for example, $\mathcal{L}_1 \subsetneq \mathcal{L}_2$ means that all queries expressible in $\mathcal{L}_1$ are also expressible in $\mathcal{L}_2$, but there are queries expressible in $\mathcal{L}_2$ that are not expressible in $\mathcal{L}_1$.

In 1975, Fagin [4] showed that queries such as the transitive closure of a graph and connectivity test cannot be expressed in relational calculus. The 0-1 law for first-order logic [5] implies that relational calculus furthermore lacks the ability to count, which was confirmed by the design of SQL, that explicitly includes counting primitives (aggregate functions). The field of *finite model theory* [1,9,12] provided many tools for studying the expressiveness of query languages: for example, Ehrenfeucht-Fraïssé games of different kinds, locality, and automata-based techniques for more expressive languages over special classes of structures such as trees.

In the direction of comparing power of languages, one of the most popular lines of work is comparing the power of a language with the class of all queries that have certain data complexity [3]. Classical results along these lines include extensions of relational calculus that describe familiar complexity classes such as PTIME.

Foundations

Relational calculus and algebra have precisely the power of first-order logic (FO) over *finite* relational structures. Many classical tools from logic, however, are inapplicable to finite structures. Investigation of the power of FO that applies to both finite and infinite structures was initiated by Ehrenfeucht and Fraïssé who developed the technique of Ehrenfeucht-Fraïssé games for characterizing the expressiveness of FO. With this technique it is very easy to prove, for example, that the *parity* query (is the number of elements of a set even?) is not expressible in FO. A more complicated application of Ehrenfeucht-Fraïssé games shows that the parity query is not FO-definable even over linear orders.

While parity is a nice and simple example of a query that involves counting, early questions about expressiveness of relational query languages concentrated on queries such as the transitive closure of a graph. For example, if a database represents a management hierarchy, it is natural to ask whether employee x reports to employee y. Such a query, in the absence of a priori bounds on the possible length of a management chain, involves transitive closure computation.

In 1975, Fagin [4] showed that graph connectivity is not definable in FO. In fact he proved a stronger result that it is not even definable in existential monadic second-order logic, whose formulae are of the form $\exists X_1 \ldots \exists X_k\ \varphi$, where the X_i's range over *sets of nodes* and φ is an FO formula. Inexpressibility of graph connectivity implies inexpressibility of transitive closure: had transitive closure E^* of a graph E been expressible, so would be connectivity by $\forall x \forall y E^*(x, y)$.

Early results have been proved using Ehrenfeucht-Fraïssé game techniques. Later easier techniques have been developed, such as *locality* and *zero-one laws*. Locality says that a formula $\varphi(x_1,\ldots,x_n)$ cannot "see" far beyond its free variables. For example, the transitive closure is not local because in a graph with directed edges $(0, 1),(1, 2),(2, 3),\ldots,(n - 1, n)$ a given formula $\varphi(x, y)$ will not see the difference between pairs (j, k) and (k, j) for $j < k$ if the numbers k and j as well as $k - j$ and $n - k$ are sufficiently large.

Zero-one laws say that asymptotically the truth value of a sentence is almost surely false or almost surely true. For example, *parity* violates this property, as the proportion of structures of cardinality n for which the *parity* query is true oscillates between 0 and 1. For surveys of these results, see [1,12].

Since FO cannot do nontrivial counting and cannot express fixed-point computation, database theory research looked in detail into such extensions of FO. The simplest way to add counting is by means of counting quantifiers $\exists ix\varphi(x)$ stating that there are at least i witnesses x of $\varphi(x)$. For example, one can express that the number of witnesses of $\varphi(x)$ is even as $\exists i\exists!ix\ \varphi(x)\ \wedge\exists j\ (j+j=i)$, where $\exists!ix\ \varphi(x)$ says that the number of witnesses is exactly i (it is defined as $\exists ix\ \varphi(x)\ \wedge\exists j(\exists jx\ \varphi(x)\ \wedge\ j>i)$).

One can then prove that such an extension of FO is still local, which means, in particular, that queries such as transitive closure and graph connectivity are not expressible. However, this form of counting is very different from the usual one in SQL, such as aggregate functions together with `GROUPBY` and `HAVING` clauses. These can be modeled by adding more powerful arithmetic and aggregate terms to FO. The idea is that if there are some aggregate functions $\mathcal{F}_1,\ldots,\mathcal{F}_k$ (such as `MIN`, `SUM`, `AVG`), one can have expressions of the form

$$\mathrm{AGGR}_l[i_1:\mathcal{F}_1,\ldots,i_k:\mathcal{F}_k](e)$$

whose semantics is basically that of the following SQL query:

$$\begin{array}{ll}\texttt{SELECT} & \#1,\ldots,\#m,\mathcal{F}_1(\#\texttt{i}_1),\ldots,\mathcal{F}_k(\#\texttt{i}_k)\\ \texttt{FROM} & \texttt{E}\\ \texttt{GROUPBY} & \#1,\ldots,\#\textit{l}\end{array}$$

where `E` is the result of the expression e, which is assumed to produce a relation with m attributes [10,11]. In fact even more general aggregate terms could be permitted, and even with them, the resulting queries that are expressed in such an extension remain local. Hence, even aggregate extensions of FO cannot express fixed-point computations required to compute queries such as the transitive closure [11].

However, for this result it is essential that no operations are permitted on the domain of graph nodes; for example, one cannot compare them and say $a<b$. If an ordering is available, then even with very basic aggregate functions, proving inexpressibility of transitive closure is at least as hard as solving some long-standing open problems related to separation of complexity classes.

Another well-studied extension of FO is that with fixed-point operators. For example, the following Datalog program computes the transitive closure of a graph represented by the edge relation e:

$$\begin{array}{lcl} trcl(x,y) & :- & e(x,y)\\ trcl(x,y) & :- & e(x,z),\wedge trcl(z,y)\end{array}$$

This could also be expressed as a *least fixed-point* formula $\text{LFP}_{R,x,y}\ (E(x,\ y)\ \vee\exists z(E(x,\ z)\ \wedge\ R(z,\ y)))$, which computes the least fixed-point of the operator that sends each binary relation R to the relation $F(R)=E\cup E\circ R$, where $\circ$ is the relational composition.

For such formulae, it is essential that R occur positively (under an even number of negations), but many different flavors of datalog and fixed-point logics are known, which permit more flexible syntax and fewer restrictions (often at the expense of a more complicated semantics).

It was proved that such an extension of FO with least fixed-point still has the zero-one law [2]. In particular, the *parity* query cannot be defined in it. In fact this continues to hold for more powerful fixed-point operators.

But similarly to the case of aggregates, the situation changes dramatically if a linear ordering is allowed on the domain. In that case, the Immerman-Vardi theorem states that the least-fixed-point extension of FO captures precisely the class of queries with PTIME data complexity [8,14]. And various other fixed-point logics have been proposed to capture complexity classes such as DLOGSPACE, NLOGSPACE, PSPACE over ordered structures [9].

This gives rise to a natural question whether it is possible to find a logical query language that captures polynomial-time queries over unordered structures. This question, asked for the first time in [3], remains open, despite multiple attempts to solve it. For example, adding both fixed-point operators and counting still falls short of polynomial time. See Chapts. 1 and 2 in [6] for surveys. There are some positive results when the class of structures is restricted: for example, a different (inflationary) fixed-point operator together with counting captures polynomial time on the class of all unordered planar graphs [7].

Questions related to the expressive power of query languages have been addressed for other data models as well. Results on aggregates are closely related to the expressiveness of relational languages under the bag (multiset) semantics. Many results about expressiveness of languages for nested relations can be derived from the *conservativity* theorem of [15] stating a query from relations to relations, even if it is expressed with the help of the operators of *nested* relational algebra, can be expressed in FO without any such operators. Similar in flavor conservativity results have been proved for constraint databases, and extensions of relational calculus with various constraints, such as polynomial (in) equalities over the reals (see Chapt. 5 of [6]).

Questions related to the expressive power of query languages are also actively studied in the context of XML and query languages for unranked trees (see, e.g., [13] for an overview).

Key Applications

The inability of relational calculus (and more generally its extension with aggregation) to express fixed-point queries was behind the addition of recursive constructs to SQL3. For example, to find all descendants of a node *a* in a graph represented by a relation `Graph(source, destination)`, one can write in SQL3:

```
WITH RECURSIVE TrCl(ancestor,
descendant) AS
    ( (SELECT source as ancestor, desti
      nation as descendant FROM Graph)
   UNION
     (SELECT Graph.source as ancestor,
     TrCl.descendant
     FROM Graph, TrCl
     WHERE Graph.destination = TrCl.
     ancestor)
   )
SELECT descendant FROM TrCl where
ancestor='a'
```

Often it is the understanding of the expressiveness of a language that guides the design of additional features.

Cross-references

- ▶ Aggregation
- ▶ Data Complexity
- ▶ Datalog
- ▶ Ehrenfeucht-Fraïssé Games
- ▶ Locality
- ▶ Zero-One Laws

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases, Addison-Wesley, Reading, MA, 1995.
2. Blass A., Gurevich Y., and Kozen D. A zero-one law for logic with a fixed-point operator. Inform. Control, 67:70–90, 1985.
3. Chandra A. and Harel D. Structure and complexity of relational queries. J. Comput. Syst. Sci., 25:99–128, 1982.
4. Fagin R. Monadic generalized spectra. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik, 21:89–96, 1975.
5. Fagin R. Probabilities on finite models. J. Symbolic Logic, 41:50–58, 1976.
6. Grädel E., Kolaitis Ph., Libkin L., Marx M., Spencer J., Vardi M.Y., Venema Y., and Weinstein S. Finite Model Theory and its Applications. Springer, Berlin, 2007.
7. Grohe M. Fixed-point logics on planar graphs. In Proc. IEEE Symposium on Logic in Computer Science, 1998, pp. 6–15.
8. Immerman N. Relational queries computable in polynomial time (extended abstract). In Proc. ACM Symposium on Theory of Computing, 1982, pp. 147–152.
9. Immerman N. Descriptive Complexity. Springer, Berlin, 1998.
10. Klug A. Equivalence of relational algebra and relational calculus query languages having aggregate functions. J. ACM, 29:699–717, 1982.
11. Libkin L. Expressive power of SQL. Theor. Comput. Sci., 296:379–404, 2003.
12. Libkin L. Elements of Finite Model Theory. Springer, Berlin, 2004.
13. Libkin L. Logics for unranked trees: an overview. Logic. Methods Comput. Sci., 2(3), 2006.
14. Vardi M.Y. The complexity of relational query languages. In Proc. ACM Symposium on Theory of Computing, 1982, pp. 137–146.
15. Wong L. Normal forms and conservative extension properties for query languages over collection types. J. Comput. Syst. Sci., 52:495–505, 1996.

Extended Entity-Relationship Model

Bernhard Thalheim
Christian-Albrechts University Kiel, Kiel, Germany

Synonyms

EERM, HERM; Higher-Order Entity-Relationship Model; Hierarchical Entity-Relationship Model

Definition

The extended entity-relationship (EER) model is a language for defining the structure (and functionality) of database or information systems. Its structure is developed inductively. Basic attributes are assigned to base data types. Complex attributes can be constructed by applying constructors such as tuple, list or set constructors to attributes that have already been constructed. Entity types conceptualize structuring of things of reality through attributes. Cluster types generalize types or combine types into singleton types. Relationship types associate types that have already been constructed into an association type. The types may be restricted by integrity constraints and by specification of identification of objects defined for a type. Typical integrity constraints of the extended entity-relationship model are participation, look-across, and general cardinality constraints. Entity, cluster, and relationship classes contain a finite set of objects defined on these types. The types of an EER schema are typically depicted by an EER diagram.

Historical Background

The entity-relationship (ER) model was introduced by P.P. Chen in 1976 [1]. The model conceptualizes and graphically represents the structure of the relational model. It is currently used as the main conceptual model for database and information system development. Due to its extensive usage a large number of extensions to this model were proposed in the 80s and 90s. Cardinality constraints [1,3,4,8] are the most important generalization of relational database constraints [7]. These proposals have been evaluated, integrated or explicitly discarded in an intensive research discussion. The semantic foundations proposed in [2,5,8] and the various generalizations and extensions of the entity-relationship model have led to the introduction of the higher-order or hierarchical entity-relationship model [8] that integrates most of the extensions and also supports conceptualization of functionality, distribution [9], and interactivity [6] for information systems. Class diagrams of the UML standard are a special variant of extended entity-relationship models.

The ER conferences (annually; since 1996: International Conference on Conceptual Modeling, http://www.conceptualmodeling.org/) are the main fora for conceptual models and modeling.

Foundations

The extended entity-relationship model is mainly used as a language for conceptualization of the structure of information systems applications. Conceptualization of database or information systems aims to represent the logical and physical structure of an information system. It should contain all the information required by the user and required for the efficient behavior of the whole information system for all users. Conceptualization may further target the specification of database application processes and the user interaction. Structure descriptions are currently the main use of the extended ER model.

An Example of an EER Diagram

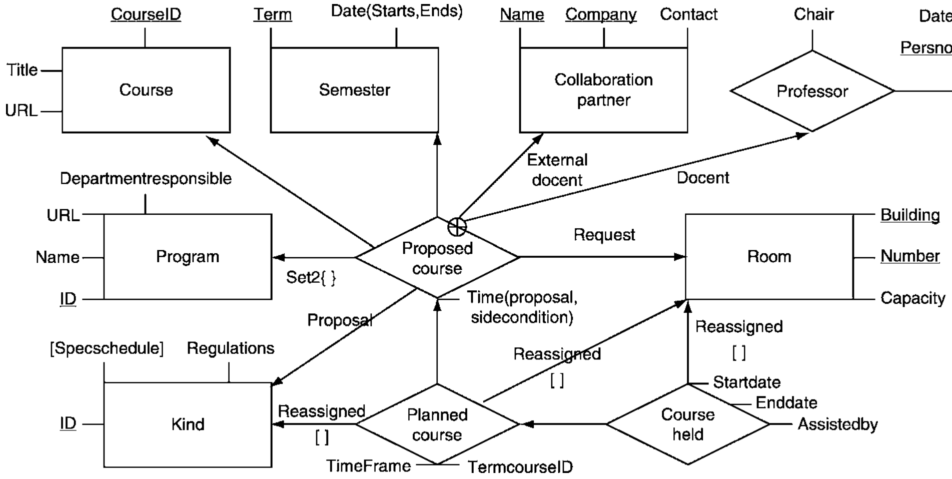
The EER model uses a formal language for schema definition and diagrams for graphical representation of the schema. Let us consider a small university application for management of *Courses*. *Proposed courses* are based on courses and taught by a docent or an external docent within a certain semester and for a set of programs. Proposals typically include a request for a room and for a time and a categorization of the kind of the course. These proposals are the basis for course planning. Planning may change time, room and kind. Planned courses are held at the university. Rooms may be changed. The example is represented by the EER diagram in Fig. 1.

Entity types are represented graphically by rectangles. Attribute types are associated with the corresponding entity or relationship type. Attributes primarily identifying a type are underlined. Relationship types are represented graphically by diamonds and associated by directed arcs to their components. A cluster type is represented by a diamond, is labeled by the disjoint union sign, and has directed arcs from the diamond to its component types. Alternatively, the disjoint union representation $\oplus$ is attached to the relationship type that uses the cluster type. In this case directed arcs associate the $\oplus$ sign with component types. An arc may be annotated with a label.

The Definition Scheme for Structures

The extended entity-relationship model uses a data type system for its attribute types. It allows the construction of entity types $E \triangleq (attr(E), \Sigma_E)$ where E is the entity type defined as a pair – the set $attr(E)$ of attribute types and the set Σ_E of integrity constraints that apply to E. The definition *def* of a type T is denoted by $T \triangleq def$.

The EER model lets users inductively build relationship types $R \triangleq (T_1,...,T_n, attr(R), \Sigma_R)$ of order i $(i \geq 1)$ through a set of (labeled) types of order less than i, a set of attribute types, and a set of integrity constraints that apply to R. The types $T_1,...,T_n$ are the components of the relationship type. Entity types are of order 0.



Extended Entity-Relationship Model. Figure 1. Extended Entity-Relationship Diagram for Course Management.

Relationship types are of order 1 if they have only entity types as component types. Relationship types are of order i if all component types are of order less than i and if one of the component types is of order $i - 1$.

Additionally, cluster types $C \overset{\circ}{=} T_1 \dot{\cup} \ldots \dot{\cup} T_n$ of order i can be defined through a disjoint union $\dot{\cup}$ of relationship types of order less than i or of entity types.

Entity/relationship/cluster classes T^C contain a set of objects of the entity/relationship/cluster type T. The EER model mainly uses set semantics, but (multi-)list or multiset semantics can also be used. Integrity constraints apply to their type and restrict the classes. Only those classes are considered for which the constraints of their types are valid. The notions of a class and of a type are distinguished. Types describe the structure and constraints. Classes contain objects.

The data type system is typically inductively constructed on a base type B by application of constructors such as the tuple or products constructor (..), set constructor {..}, and the list constructor < .. >. Types may be optional component types and are denoted by [..].

The types T can be labeled l : T. The label is used as an alias name for the type. Labels denote roles of the type. Labels must be used if the same type is used several times as a component type in the definition of a relationship or cluster type. In this case they must be unique.

An entity-relationship schema consists of a set of data, attribute, entity, relationship, and cluster types which types are inductively built on the basis of the base types.

Given a base type system B. The types of the ER schema are defined through the type equation: $T = B|(l_1 : T, ..., l_n : T) \;|\{T\}\; | < \mathrm{T} > | \; [\mathrm{T}] \;|\mathrm{T} \dot{\cup} \mathrm{T}|\mathrm{l} : T|N \overset{\circ}{=} T$

Structures in Detail

The classical four-layered approach is used for inductive specification of database structures. The first layer is the data environment, called the basic data type scheme, which is defined by the system or is the assumed set of available basic data types. The second layer is the schema of a database. The third layer is the database itself representing a state of the application's data often called micro-data. The fourth layer consists of the macro-data that are generated from the micro-data by application of view queries to the micro-data.

Attribute Types and Attribute Values The classical ER model uses basic (first normal form) attributes. Complex attributes are inductively constructed by application of type constructors such as the tuple constructor (..), set constructor {..}, and the list constructor < .. >. Typical base types are integers, real numbers, strings, and time. Given a set of names $\mathcal{N}$ and a set of base types B, a basic attribute type $A :: B$ is given by an (attribute) name $A \in \mathcal{N}$ and a base type B. The association between the attribute name and the underlying type is denoted by ::. The base type B is often called the domain of A, i.e., $dom(A) = B$. Complex attributes are constructed on base attributes by application of the type constructors. The notion of a domain is extended to complex attributes, i.e., the domain of the complex attribute A is given by $dom(A)$. Components of complex attributes may be optional, e.g., the *Title* in the attribute *Name*.

Typical examples of complex and basic attributes in Fig. 1 are

Name $\overset{\circ}{=}$ *(FirstNames <FirstName>, FamName, [AcadTitles], [FamilyTitle]),*
PersNo $\overset{\circ}{=}$ *EmplNo* $\dot{\cup}$ *SocSecNo,*
AcadTitles $\overset{\circ}{=}$ *{AcadTitle},*
Contact $\overset{\circ}{=}$ *(Phone({PhoneAtWork}, private), Email, URL, WebContact, [Fax({PhoneAtWork})]),*
PostalAddress $\overset{\circ}{=}$ *(Zip, City, Street, HouseNumber)* for *DateOfBirth :: date, AcadTitle :: acadTitleType, FamilyTitle :: familyTitleAcronym, Zip :: string7, SocSecNo :: string9, EmplNo :: int, City :: varString, Street :: varString, HouseNumber :: smallInt.*

The complex attribute *Name* is structured into a sequence of first names, a family name, an optional complex set-valued attribute for academic titles, and an optional basic attribute for family titles. Academic titles and family titles can be distinguished from each other.

Entity Types and Entity Classes Entity types are characterized by their attributes and their integrity constraints. Entity types have a subset K of the set of attributes which serve to identify the objects of the class of the type. This concept is similar to the concept of key known for relational databases. The key is denoted by ID(K). The set of integrity constraints Σ_E consists of the keys and other integrity constraints. Identifying attributes may be underlined instead of having explicit specification.

Formally, an entity type is given by a name E, a set of attributes $attr(E)$, a subset $id(E)$ of $attr(E)$, and a set Σ_E of integrity constraints, i.e.,

$E \overset{\circ}{=} (attr(E), \Sigma_E)$.

The following types are examples of entity types in Fig. 1:

Person $\overset{\circ}{=}$ *({Name, Login, URL, Address, Contact, DateOfBirth, <u>PersNo</u>})*
Course $\overset{\circ}{=}$ *({CourseID, Title, URL}, {ID({CourseID})}),*
Room $\overset{\circ}{=}$ *({Building, Number, Capacity}, {ID({Building, Number})}),*
Semester $\overset{\circ}{=}$ *({Term, Date(Starts, Ends)}, { ID({ Term })}).*

An ER schema may use the same attribute name with different entity types. For instance, the attribute *URL* in Fig. 1 is used for characterizing additional information for the type *Person* and the type *Course*. If they need to be distinguished, then complex names such as *CourseURL* and *PersonURL* are used.

Objects on type E are tuples with the components specified by a type. For instance, the object (or *entity*) *(HRS3, 408A, 15)* represents data for the *Room* entity type in Fig. 1.

An entity class E^C of type E consists of a finite *set* of objects on type E for which the set Σ_E of integrity constraints is valid.

Cluster Types and Cluster Classes A disjoint union $\dot{\cup}$ of types whose identification type is domain compatible is called a cluster. Types are domain compatible if they are subtypes of a common more general type. The union operation is restricted to disjoint unions since identification must be preserved. Otherwise, objects in a cluster class cannot be related to the component classes of the cluster type. Cluster types can be considered as a generalization of their component types.

A cluster type (or "category") $C \overset{\circ}{=} l_1 : R_1 \dot{\cup} l_2 : R_2 \dot{\cup} \ldots \dot{\cup} l_k : R_k$ is the (labeled) disjoint union of types $R_1, \ldots, R_k$. Labels can be omitted if the types can be distinguished.

The following type is an example of a cluster type:

Teacher $\overset{\circ}{=}$ *ExternalDocent : CollaborationPartner* $\dot{\cup}$ *Docent : Professor.*

The cluster class C^C is the 'disjoint' union of the sets $R_1^C, \ldots, R_k^C$. It is defined if $R_1{}^C, \ldots R_k{}^C$ are disjoint on their identification components. If the sets $R_1^C, \ldots, R_k^C$ are not disjoint then labels are used for differentiating the objects of clusters. In this case, an object uses a pair representation (l_i, o_i) for objects o_i from $R_i{}^C$.

Relationship Types and Relationship Classes First order relationship types are defined as associations between entity types or clusters of entity types. Relationship types can also be defined on the basis of relationship types that are already defined. This construction must be inductive and cannot be cyclic. Therefore, an order is introduced for relationship types. Types can only be defined on the basis of types which have a lower order. For instance, the type *Professor* in Fig. 1 is of order 1. The type *ProposedCourse* is of order 2 since all its component types are either entity types or types of order 1. A relationship type of order i is defined as an association of relationship types of order less than i or of entity types. It is additionally required that at least one of the component types is of order $i - 1$ if $i > 1$. Relationship types can also be characterized by attributes. Relationship types with one component type express a subtype or an Is-A relationship type. For instance, the type *Professor* is a subtype of the type *Person*.

Component types of a relationship type may be labeled. Label names typically provide an understanding of the role of a component type in the relationship type. Labeling uses the definition scheme *Label* : *Type*. For instance, the *Kind* entity type is labeled by *Proposal* for the relationship type *ProposedCourse* in Fig. 1.

Cluster types have the maximal order of their component types. Relationship types also may have cluster type components. The order of cluster type components of a relationship type of order i must be less than i.

Component types that are not used for identification within the relationship type can be optional. For instance, the *Room* component in Fig. 1 is optional for the type *PlannedCourse*. If the relationship object in the *PlannedCourse* class does not have a room then the proposal for rooms in *ProposedCourse* is accepted. A specific extension for translation of optional components may be used. For instance, *Room* in Fig. 1 is inherited to *PlannedCourse* from *ProposedCourse* if the *Room* component for a *PlannedCourse* is missing.

Higher order types allow a convenient description of types that are based on other types. For example, consider the course planning application in Fig. 1. Lectures are courses given by a professor or a collaboration partner within a semester for a number of programs. Proposed courses extend lectures by describing which room is requested and which time proposals and which restrictions are made. Planing of courses assigns a room to a course that has been proposed and assigns a time frame for scheduling. The kind of the course may be changed. Courses that are held are based on courses planned. The room may be changed for a course. The following types specify these assertions.

$$\begin{aligned}
&\mathit{ProposedCourse} \circeq (\underline{\mathit{Teacher}}, \underline{\mathit{Course}}, \mathit{Proposal} : \mathit{Kind}, \\
&\qquad \mathit{Request} : \mathit{Room}, \mathit{Semester}, \mathit{Set2} : \{\mathit{Program}\}, \{\mathit{Time}(\mathit{Proposal}, \mathit{SideCondition})\}, \Sigma_{\mathit{ProposedCourse}}), \\
&\mathit{PlannedCourse} \circeq (\mathit{ProposedCourse}, [\mathit{Reassigned} : \mathit{Kind}], \\
&\qquad [\mathit{Reassigned} : \mathit{Room}], \{\mathit{TimeFrame}, \mathit{TermCourseID}\}, \Sigma_{\mathit{PlannedCourse}}), \\
&\mathit{CourseHeld} \circeq (\mathit{PlannedCourse}, [\mathit{Reassigned} : \mathit{Room}], \\
&\{\mathit{StartDate}, \mathit{EndDate}, \mathit{AssistedBy}\}, \Sigma_{\mathit{CourseHeld}}).
\end{aligned}$$

The second and third types use optional components in case a proposal or a planning of rooms or kinds is changed. Typically, planned courses are identified by their own term-specific identification. Integrity constraints can be omitted until they have been defined.

Formally, a relationship type is given by a name R, a set $\mathit{compon}(R)$ of labeled components, a set of attributes $\mathit{attr}(R)$, and a set Σ_R of integrity constraints that includes the identification of the relationship type by a subset $\mathit{id}(R)$ of $\mathit{compon}(R) \cup \mathit{attr}(R)$, i.e., $R \circeq (\mathit{compon}(R), \mathit{attr}(R), \Sigma_R)$.

It is often assumed that the identification of relationship types is defined exclusively through their component types. Relationship types that have only one component type are unary types. These relationship types define subtypes. If subtypes need to be explicitly represented then binary relationship types named by *IsA* between the subtype and the supertype are used. For instance, the type *Professor* in Fig. 1 is a subtype of the type *Person*.

An object (or a "relationship") on the relationship type $R \overset{\circ}{=} (R_1,\ldots,R_n, \{B_1,\ldots,B_k\}, \mathit{id}(R), \Sigma_R)$ is an element of the Cartesian product $R_1^C \times \ldots \times R_n^C \times \mathit{dom}(B_1) \times \ldots \times \mathit{dom}(B_k)$. A relationship class R^C consists of a finite *set* $R^C \subseteq R_1^C \times \ldots \times R_n^C \times \mathit{dom}(B_1) \times \ldots \times \mathit{dom}(B_k)$ of objects on R for which $\mathit{id}(R)$ is a key of R^C and which obeys the constraints Σ_R.

Integrity Constraints Each database model also uses a set of implicit model-inherent integrity constraints. For instance, relationship types are defined over their component types, and a (relationship) object presumes the existence of corresponding component objects. Typically only finite classes are considered. The EER schema is acyclic. Often names or labels are associated with a minimal semantics that can be derived from the meaning of the words used for names or labels. This minimal semantics allows us to derive synonym, homonym, antonym, toponym, hypernym, and holynym associations among the constructs used.

The most important class of integrity constraints of the EER model is the class of cardinality constraints. Other classes of importance for the EER model are multivalued dependencies, inclusion and exclusion constraints and existence dependencies[5]. Functional dependencies, keys and referential constraints (or key-based inclusion dependencies) can be expressed through cardinality constraints.

Three main kinds of cardinality constraints are distinguished: participation constraints, look-across constraints, and general cardinality constraints. Given a relationship type $R \hat{=} (compon(R), attr(R), \Sigma_R)$, a component R' of R, the remaining substructure $R'' = R \setminus R'$ and the remaining substructure $R''' = R'' \sqcap_R compon(R)$ without attributes of R.

The participation constraint $card(R, R') = (m, n)$ restricts the number of occurrences of R' objects in the relationship class R^C by the lower bound m and the upper bound n. It holds in a relationship class R^C if for any object $o' \in R'^C$ there are at least m and at most n objects $o \in R^C$ with $\pi_{R'}(o) = o'$ for the projection function $\pi_{R'}$ that projects o to its R' components.

Participation constraints relate objects of relationship classes to objects of their component classes. For instance, the constraint *card*(*ProposedCourse*, *SemesterCourse*) = (0, 3) restricts relationship classes for proposals for courses per semester to at least 0 and at most 3, i.e., each course is proposed at most three times in a semester. There are at most three objects o in $ProposedCourse^C$ with the same course and semester objects. The integrity constraint *card*(*ProposedCourse*, *DocentSemester*) = (3, 7) requires that each docent is giving at least 3 courses and at most 7 courses. External docents may be obliged by other restrictions, e.g., *card*(*ProposedCourse*, *ExternalDocentSemester*) = (0,1).

Formally, the integrity constraint $card(R,R') = (m,n)$ is valid in R^C if $m \leq |\{o \in R^C : \pi_{R'}(o) = o'\}| \leq n$ for any $o' \in \pi_{R'}(R^C)$ and the projection $\pi_{R'}(R^C)$ of R^C to R'.

If $card(R, R') = (0, 1)$ then R' forms an identification or a key of R, i.e., $ID(R')$ for R. This identification can also be expressed by a functional dependency $R : R' \rightarrow R''$.

The lookup or look-across constraint $look(R, R') = m..n$ describes how many objects o''' from R'''^C may potentially 'see' an object o' from R'^C. It holds in a relationship class R^C if for any object $o''' \in dom(R''')$ there are at least m and at most n related objects o' with $\pi_{R'}(o) = o'$, i.e., $m \leq |\{o' \in \pi_{R'}(R^C) : o \in R^C \wedge \pi_{R'}(o) = o' \wedge \pi_{R'''}(o) = o'''\}| \leq n$ for any $o''' \in Dom(R''')$. Typically, look-across constraints are used for components consisting of one type. Look-across constraints are not defined for relationship types with one component type.

Look-across constraints are less intuitive for relationship types with more than two component types or with attribute types. For instance, the look-across constraint *look*(*ProposedCourse*, *DocentSemester*) = 0..7 specifies that for any combination of *Teacher*, *Room*, *Kind*, and *Program* objects there are between 0 and 7 *Docent* and *Semester* combinations. The lower bound expresses that there are *Teacher*, *Room*, *Kind*, and *Program* which do not have a *Docent* and *Semester* combination.

Look-across constraints for a binary relationship type whose component types form a key of the relationship type can equivalently be expressed by participation constraints, i.e., $look(R, R_1) = m_1..n_1$ if and only if $card(R, R_2) = (m_1, n_1)$. Similarly, $look(R, R_2) = m_2..n_2$ if and only if $card(R, R_1) = (m_2, n_2)$. This equivalence is neither valid for binary relationship types which cannot by identified by their components nor for relationship types with more than two components.

Participation and look-across constraints can be extended to substructures and intervals and to other types such as entity and cluster types. Given a relationship type R, a substructure R' of R, R'' and R''' as above, and given furthermore an interval $I \subseteq \mathbb{N}_0$ of natural numbers including 0, the (general) cardinality constraint $card(R, R') = I$ holds in a relationship class R^C if for any object $o' \in \pi_{R'}(R^C)$ there are $i \in I$ objects o with $\pi_{R'}(o) = o'$, i.e., $|\{o \in R^C : \pi_{R'}(o) = o'\}| \in I$ for any $o' \in \pi_{R'}(R^C)$.

The following participation, look-across and general cardinality constraints are examples in Fig. 1:

For any $R' \in$ {Semester, Course, Kind} *card*(*ProposedCourse*,R') = (0, n),
card(*ProposedCourse*, *SemesterCourseTeacher*) = (0, 1),
card(*CourseHeld*, *PlannedCourse*) = (1, 1),
card(*PlannedCourse*, *ProposedCourse*[*Semester*]*Room TimeFrame*) = (0, 1),
card(*ProposedCourse*, *DocentSemester*) = {0, 3, 4, 5, 6, 7}.

The first constraint does not restrict the database. The second constraint expresses a key or functional dependency. The types *Semester Course Teacher* identify any of the other types in the type *ProposedCourse*, i.e.,

ProposedCourse: *{Semester, Course, Teacher}* → *{Request, Time, Proposal, Set2}*.

The third constraint requires that any planned course must be given. The fourth constraint requires that rooms are not overbooked. The fifth constraint allows that docents may not teach in a semester, i.e., have a sabbatical. If a docent is teaching in a semester then at least 3 and at most 7 courses are given by the docent.

Look-across constraints were originally introduced by Chen [1] as cardinality constraints. UML uses look-across constraints. Participation and look-across constraints cannot be axiomatized through a Hilbert- or Gentzen-type logical calculus. If only upper bounds are of interest then an axiomatization can be found in [3] and [4]. General cardinality constraints combine equality-generating and object-generating constraints such as keys, functional dependencies and referential integrity constraints into a singleton construct.

Logical operators can be defined for each type. A set of logical formulas using these operators can define the integrity constraints which are valid for each object of the type.

Schemata

The schema is based on a set of base (data) types which are used as value types for attribute types.

A set $\{E_1,...E_n, C_1,...,C_l, R_1,...,R_m\}$ of entity, cluster and (higher-order) relationship types on a data scheme DD is called schema if the relationship and cluster types use only the types from $\{E_1,..., E_n, C_1,...,C_l, R_1,...,R_m\}$ as components and cluster and relationship types are properly layered.

An EER schema is defined by the pair $\mathcal{D} = (\mathcal{S}, \Sigma)$ where $\mathcal{S}$ is a schema and Σ is a set of constraints. A database $\mathcal{D}^C$ on $\mathcal{D}$ consists of classes for each type in $\mathcal{D}$ such that the constraints Σ are valid.

The classes of the extended ER model have been defined through sets of objects on the types. In addition to sets, lists, multi-sets or other collections of objects may be used. In this case, the definitions used above can easily be extended [8].

A number of domain-specific extensions have been introduced to the ER model. One of the most important is the extension of the base types by spatial data types such as: point, line, oriented line, surface, complex surface, oriented surface, line bunch, and surface bunch. These types are supported by a large variety of functions such as: meets, intersects, overlaps, contains, adjacent, planar operations, and a variety of equality predicates.

The translation of the schema to (object-)relational or XML schemata can be based on a profile [4]. Profiles define which translation choice is preferred over other choices, how hierarchies are treated, which redundancy and null-value support must be provided, which kind of constraint enforcement is preferred, which naming conventions are chosen, which alternative for representation of complex attributes is preferred for which types, and whether weak types can be used. The treatment of optional components is also specified through the translation profile of the types of the schema. A profile may require the introduction of identifier types and base the identification on the identifier. Attribute types may be translated into data formats that are supported by the target system.

The EER schema can be used to define views. The generic functions insert, delete, update, projection, union, join, selection and renaming can be defined in a way similarly to the relational model. Additionally, nesting and unnesting functions are used. These functions form the algebra of functions of the schema and are the basis for defining queries. A singleton view is defined by a query that maps the EER schema to new types. Combined views also may be considered which consist of singleton views which together form another EER schema.

A view schema is specified over an EER schema $\mathcal{D}$ by a schema $\mathcal{V} = \{S_1,...,S_m\}$, an auxiliary schema $\mathcal{A}$ and a (complex) query $q : \mathcal{D} \times \mathcal{A} \to \mathcal{V}$ defined on $\mathcal{D}$ and $\mathcal{A}$. Given a database $\mathcal{D}^C$ and the auxiliary database $\mathcal{A}^C$. The view is defined by $q(\mathcal{D}^C \times \mathcal{A}^C)$.

Graphical Representation

The schema in Fig. 1 consists of entity, cluster and relationship types. The style of drawing diagrams is one of many variants that have been considered in the literature. The main difference of representation is the style of drawing unary types. Unary relationship types are often represented by rectangles with rounded corners or by (directed) binary IsA-relationship types which associate by arcs the supertype with the subtype. Tools often do not allow cluster types and relationship types of order higher than one. In this case, those types can be objectified, i.e., represented by a new (abstract) entity type that is associated through binary relationship types to the components of the original type. In this case, identification of objects of the new type is either inherited from the component types or is provided through a new (surrogate) attribute. The first option results in the introduction of so-called weak types. The direct translation of these weak types to object-relational models must be combined with the introduction of rather complex constraint sets. Typically, this complexity can be avoided if the abstract entity type is mapped together with the new relationship types to a singleton object-relational type. This singleton type is also the result of a direct mapping of the original higher-order relationship type.

The diagram can be enhanced by an explicit representation of cardinality and other constraints. If participation constraints $card(R, R') = (m, n)$ are used for component consisting of one type R' then the arc from R to R' is labeled by (m, n). If look-across constraints $look(R, R') = m..n$ are used for binary relationship types then the arc from R to R' is labeled by $m..n$.

Key Applications

The main application area for extended ER models is the conceptualization of database applications.

Database schemata can be translated to relational, XML or other schemata based on transformation profiles that incorporate properties of the target systems.

Future Directions

The ER model has had a deep impact on the development of diagramming techniques in the past and is still influencing extensions of the unified modeling language UML. UML started with binary relationship types with look-across constraints and without relationship type attributes. Class diagrams currently allow n-ary relationship types with attributes. Relationship types may be layered. Cluster types and unary relationship types allow for distinguishing generalization from specialization.

ER models are not supported by native database management systems and are mainly used for modeling of applications at the conceptual or requirements level. ER schemata are translated to logical models such as XML schemata or relational schemata or object-relational schemata. Some of the specifics of the target models are not well supported by ER models and must be added after translating ER schemata to target schemata, e.g., specific type semantics such as list semantics (XML) or as special ordering or aggregation treatment of online analytical processing (OLAP) applications.

The ER model has attracted a lot of research over the last 30 years. Due to novel applications and to evolution of technology old problems and novel problems are challenging the research on this model. Typical old problems that are still not solved in a satisfactory manner are: development of a science of modeling, quality of ER schemata, consistent refinement of schemata, complex constraints, normalization of ER schemata, normalization of schemata in the presence of incomplete constraint sets. Novel topics for ER research are for instance: evolving schema architectures, collaboration of databases based on collaboration schemata, layered information systems and their structuring, schemata with redundant types, ER schemata for OLAP applications.

Structures of database applications are often represented through ER models. Due to the complexity of applications, a large number of extensions have recently been proposed, e.g., temporal data types, spatial data types, OLAP types and stream types. Additionally, database applications must be integrated and cooperate in a consistent form. The harmonization of extensions and the integration of schemata is therefore a never ending task for database research.

ER models are currently extended for support of (web) content management that is based on structuring of data, on aggregation of data, on extending data by concepts and on annotating data sets for simple reference and usage. These applications require novel modeling facilities and separation of syntactic, semantic and pragmatic issues. The ER model can be extended to cope with these applications.

The ER model is mainly used for conceptual specification of database structuring. It can be enhanced by operations and a query algebra. Operations and the queries can also be displayed in a graphical form, e.g., on the basis of VisualSQL. Most tools supporting ER models do not currently use this option. Enhancement of ER models by functionality is necessary if the conceptualization is used for database development. Based on functionality enhancement, view management facilities can easily be incorporated into these tools.

ER models are becoming a basis for workflow systems data. The standards that have been developed for the specification of workflows have not yet been integrated into sophisticated data and application management tools.

URL to Code

http://www.informatik.uni-kiel.de/\~thalheim/HERM.htm

http://www.is.informatik.uni-kiel.de/~thalheim/indeeerm.htm

Readings on the RADD project (Rapid Application and Database Development) Authors: M. Albrecht, M. Altus, E. Buchholz, H. Cyriaks, A. Düsterhöft,

J. Lewerenz, H. Mehlan, M. Steeg, K.D. Schewe, and B. Thalheim.

Cross-references

- ▶ Entity Relationship Model
- ▶ Relational Model
- ▶ Semantic Data Model
- ▶ Unified Modeling Language

Recommended Reading

1. Chen P.P. The entity-relationship model: toward a unified view of data. ACM Trans. Database Syst., 1(1):9–36, 1976.
2. Gogolla M. An Extended Entity-Relationship Model – Fundamentals and Pragmatics. LNCS 767. Springer, Berlin Heidelberg New York, 1994.
3. Hartmann S. Reasoning about participation constraints and Chen’s constraints. In Proc. 14th Australasian Database Conf., 2003, pp. 105–113.
4. Hartmann S., Hoffmann A., Link S., and Schewe K.-D. Axiomatizing functional dependencies in the higher-order entity-relationship model. Inf. Process. Lett., 87(3):133–137, 2003.
5. Hohenstein U. Formale Semantik eines erweiterten Entity-Relationship-Modells. Teubner, Stuttgart, 1993.
6. Schewe K.-D. and Thalheim B. Conceptual modelling of web information systems. Data Knowl. Eng., 54:147–188, 2005.
7. Thalheim B. Dependencies in Relational Databases. Teubner, Leipzig, 1991.
8. Thalheim B. Entity-Relationship Modeling – Foundations of Database Technology. Springer, Berlin Heidelberg New York, 2000.
9. Thalheim B. Codesign of structuring, functionality, distribution and interactivity. In Proc. 1st Asian-Pacific Conf. on Conceptual Modeling, 2004, pp. 3–12.

Extended Functional Dependencies

- ▶ Functional Dependencies for Semi-structured Data

Extended Relations

- ▶ Conditional Tables
- ▶ Naive Tables

Extended Transaction Models

- ▶ Generalization of ACID Properties
- ▶ Open Nested Transaction Models

Extended Transaction Models and the ACTA Framework

Panos K. Chrysanthis[1], Krithi Ramamritham[2]
[1]University of Pittsburgh, Pittsburgh, PA, USA
[2]Indian Institute of Technology Bombay, Mumbai, India

Synonyms

Advanced transaction models; Generalization of ACID properties

Definition

Although powerful, the transaction model adopted in traditional database systems is found lacking in functionality and performance when used for applications that involve reactive (endless), open-ended (long-lived) and collaborative (interactive) activities. Hence, various extensions to the traditional model have been proposed, referred to as *extended transactions*. These models are characterized by the structure of their transactions, the commit and abort dependencies and the visibility rules among transactions. *ACTA* is a comprehensive transaction framework that facilitate the specification, analysis and synthesis of extended transaction models. The name *ACTA*, meaning *actions* in Latin, was chosen given the framework’s appropriateness for expressing the properties of actions used to compose a transactional computation.

Key Points

By means of the notion of transactions, database systems offer reliability guarantees concerning the correctness of data in spite of failures and concurrent accesses by multiple users. However, the transaction model as well as the simple data model adopted in traditional database systems have been found lacking in *functionality* and *performance* in their support of the emerging advanced database applications such as design databases, computer publishing, network management, multidatabases and mobile databases. In order to deal with the inherent limitations of the traditional data and atomic transaction model, researchers have proposed semantic and object-oriented data models and extensions to the traditional transaction model. Nested transactions was the first such extension that added a hierarchical structure to the traditional flat atomic transactions. The hierarchical structure allows concurrency within a transaction and fine-grained failure and exception handling since subtransactions can abort independently without causing the abortion of the whole transaction.

The original nested transaction model was subsequently enhanced with new types of subtransactions, relaxed abort and commit dependencies and visibility rules for externalizing partial results among transactions. These extensions led to a variety of open-nested transactions models such as Sagas, Split Transactions, Flex Transactions, ConTracts and S-transactions, and of correctness criteria such as quasi serializability, epsilon-serializability, semantic atomicity, quasi failure-atomicity.

All the above extensions have been introduced with specific applications or with specific transaction properties in mind [2]. Their ad hoc character makes it difficult to identify the properties of transactions that adhere to a particular model and to ascertain in what respects an extended transaction model is similar or different from another. The need for a comprehensive transaction framework that would facilitate the precise specification of the properties of a model, vis a vis visibility, consistency, recovery and permanence, and allow the formal comparison of different models led to the development of *ACTA* [1]. ACTA is a first-order logic based formalism with a precedence relation that allows a transaction modeler to specify both the high level properties (requirements) of a model and the lower level behavioral aspects of the model in terms of axioms. Specifications include the following four components: (i) the set of transaction management events associated with the transaction model, such as *begin, commit, abort, split*, and *join*; (ii) the semantics of these significant events, characterized in terms of their effect on objects (their value and synchronization state) and other transactions (different types of dependencies, such as commit dependency and abort dependency); (iii) the view of each transaction, specifying the state of objects visible to that transaction; and (iv) the conflict set of each transaction, containing those operations with respect to which conflicts need to be considered.

Besides supporting the specification and analysis of existing transaction models, ACTA has the power to specify the requirements of new transactional applications and synthesize models that satisfy these requirements. This was demonstrated by deriving new transaction definitions either by starting from first principles or by modifying and/or combining the specifications of existing transaction models. The exercise of analyzing and synthesizing different transaction models revealed the many advantages of using a simple formalism like ACTA to deal with advanced transactions and has influenced a lot of transaction processing work in industry and academia.

Although ACTA has been developed to characterize extended transaction models, it has been extended to express the various correctness criteria beyond serializability. The use of this formalism resulted in a consolidated notion of correctness in which the different serializability-based criteria, such as predicatewise serializability and cooperative serializability, can been seen as special cases [3].

Cross-references

- ▶ ACID Properties
- ▶ Correctness Criteria beyond Serializability
- ▶ ConTract
- ▶ Compensating Transactions
- ▶ e-Commerce Transactions
- ▶ Flex Transactions
- ▶ Generalization of ACID Properties
- ▶ Internet Transactions
- ▶ Multilevel Transactions and Object-Model Transactions
- ▶ Nested Transaction Models
- ▶ Open Nested Transaction Models
- ▶ Polytransactions
- ▶ Sagas
- ▶ Semantic Atomicity
- ▶ Split Transactions
- ▶ Transaction
- ▶ Transaction Management
- ▶ Transactional Processes
- ▶ Workflow Transactions

Recommended Reading

1. Chrysanthis P.K. and Ramamritham K. Synthesis of extended transaction models using ACTA. ACM Trans. Database Syst., 19(3):450–491, 1994.
2. Elamagarmid A. K. (Ed.). Database Transaction Models for Advanced Applications, Morgan Kaufmann, Los Altos, CA, 1992.
3. Ramamritham K. and Chrysanthis P.K. A taxonomy of correctness criteria in database applications. VLDB J., 4(1):181–293, 1996.

Extendible Hashing

Donghui Zhang[1], Yannis Manolopoulos[2], Yannis Theodoridis[3], Vassilis J. Tsotras[4]
[1]Northeastern University, Boston, MA, USA
[2]Aristotle University, Thessaloniki, Greece
[3]University of Piraeus, Piraeus, Greece
[4]University of California – Riverside, Riverside, CA, USA

Definition

Extendible hashing is a dynamically updateable disk-based index structure which implements a hashing scheme utilizing a directory. The index is used to support exact match queries, i.e., find the record with a given key. Compared with the B+-tree index which also supports exact match queries (in logarithmic number of I/Os), Extendible Hashing has better expected query cost O(1) I/O. Compared with linear hashing, extendible hashing does not have any overflow page. Overflows are handled by doubling the directory which logically doubles the number of buckets. Physically, only the overflown bucket is split.

Historical Background

The extendible hashing scheme was introduced by [1]. A hash table is an in-memory data structure that associates keys with values. The primary operation it supports efficiently is a lookup: given a key, find the corresponding value. It works by transforming the key using a hash function into a hash, a number that is used as an index in an array to locate the desired location where the values should be. Multiple keys may be hashed to the same bucket, and all keys in a bucket should be searched upon a query. Hash tables are often used to implement associative arrays, sets and caches. Like arrays, hash tables have O(1) lookup cost on average.

Foundations

Structure

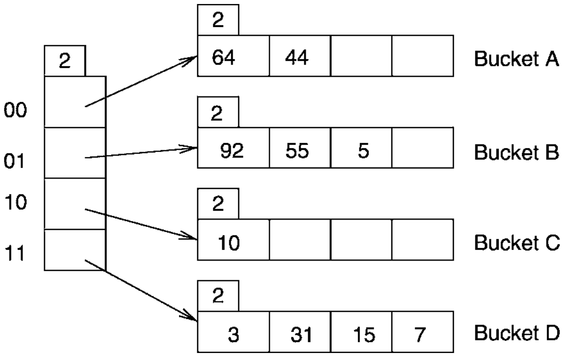
Extendible hashing uses a directory to access its buckets. This directory is usually small enough to be kept in main memory and has the form of an array with 2^d entries, each entry storing a bucket address (pointer to a bucket). The variable d is called the global depth of the directory. To decide where a key k is stored, extendible hashing uses the last d bits of some adopted hash function $h(k)$ to choose the directory entry. Multiple directory entries may point to the same bucket. Every bucket has a local depth *leqd*. The difference between local depth and global depth affects overflow handling.

An example of extendible hashing is shown in Fig. 1. Here there are four directory entries and four buckets. The global depth and all the four local depths are 2. For simplicity assume the adopted hash function is $h(k) = k$. For instance, to search for record 15, one refers to directory entry 15% 4 = 3 (or 11 in binary format), which points to bucket D.

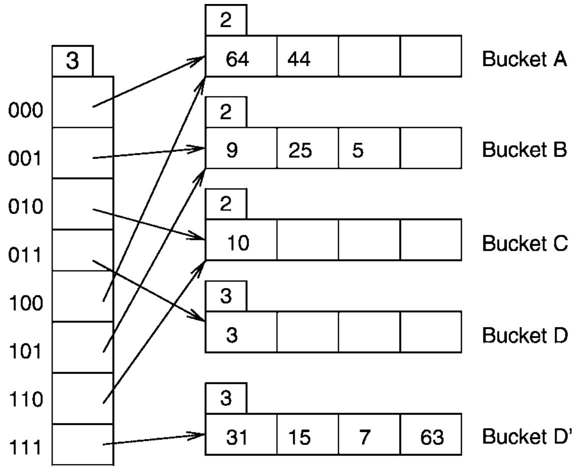
Overflow Handling

If a bucket overflow happens, the bucket is split into two. The directory may or may not double, depending on whether the local depth of the overflown bucket was equal to the global depth before split.

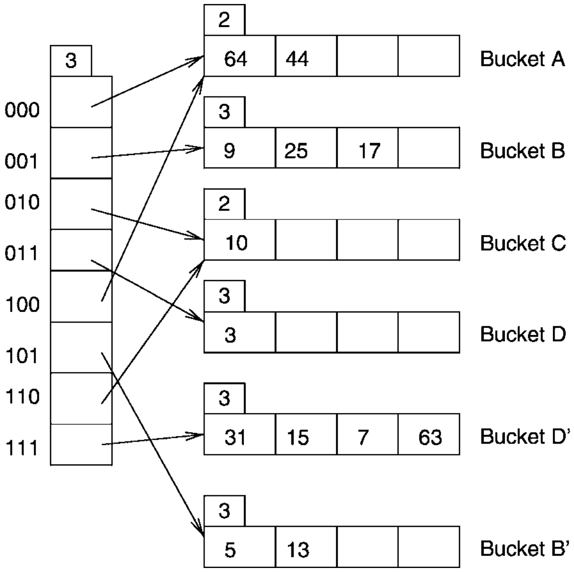
If the local depth was equal to global depth, d bits are not enough to distinguish the search values of the overflown bucket. Thus a directory *doubling* occurs, which effectively uses one more bit from the hash value. The directory size is then doubled (this does not mean that the number of buckets is doubled as buckets will share directory entries). As an example, Fig. 2 illustrates extendible hashing after inserting a new record with key 63 into Fig. 1. Bucket D overflows and the records in it are redistributed between D (where the last three bits of a record's hash value are 011) and D' (where the last three bits of a record's has value are 111). The directory doubles. The global depth is increased by one. The local depth of buckets D and D' are increased



Extendible Hashing. Figure 1. Illustration of the Extendible Hashing.



Extendible Hashing. Figure 2. The directory doubles after inserting 63 into Fig. 1.



Extendible Hashing. Figure 3. The directory does not double after inserting 17 and 13 into Fig. 2.

by one, while the local depth of the other buckets remains to be two. Except 111, which points to the new bucket D', each of the new directory entries points to the existing bucket which shares the last two bits. For instance, directory entry 101 points to the bucket referenced by directory entry 001.

In general, if the local depth of a bucket is d', the number of directory entries pointing to the bucket is $2^{d-d'}$. All these directory entries share the last d' bits.

To split an overflown bucket whose local depth is smaller than the global depth, one does not need to double the size of the directory. Instead, half of the $2^{d-d'}$ directory entries will point to the new bucket, and the local depth of both the overflown bucket and its split image are increased by one. For instance, Fig. 3 illustrates the extendible hashing after inserting 17 and 13 into Fig. 2. Bucket B overflows and a split image, bucket B', is created. There are two directory entries (001 and 101) that pointed to B before the split. Half of them (101) now points to the split image B'. The local depth of both buckets B and B' are increased by one.

Discussion

Deletion may cause a bucket to become empty, in which case it can be merged with its *buddy* bucket. The buddy bucket is referenced by the directory entry which shares the last (local depth − 1) bits. For instance, the buckets referenced by direction entries 1111 and 0111 are buddy buckets. Many deletions can cause the directory to halve its size and thus decrease its global depth. This is triggered by the bucket merging which causes all local depth to be strictly smaller than the global depth.

The fact that extendible hashing does not use any overflow page may significantly increase the directory size, as one insertion may cause the directory to double more than once. Consider the case when global depth is 3, and the bucket referenced by directory entry 001 overflows with five records 1, 17, 33, 49, 65. The directory is doubled. Directory entries 0001 and 1001 points to the overflown bucket and the split image. All the five keys will remain in the original bucket which is again overflowing. Therefore the directory has to be doubled again.

To alleviate this problem, one can allow a certain degree of overflow page links. For instance, whenever the fraction of buckets with overflow pages becomes larger than 1%, double the directory.

Key Applications

Extendible Hashing can be used in applications where exact match query is the most important query such as hash join [2].

Cross-references

- ▶ Bloom Filter
- ▶ Hashing
- ▶ Hash-based Indexing
- ▶ Linear Hashing

Recommended Reading

1. Fagin R., Nievergelt J., Pippenger N., and Strong H.R. Extendible hashing: a fast access method for dynamic files. ACM Trans. Database Syst., 4(3):315–344, 1979.
2. Schneider D.A. and DeWitt D.J. Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 469–480.

Extensible Markup Language

▶ XML

eXtensible Stylesheet Language

▶ XSL/XSLT

eXtensible Stylesheet Language Transformations

▶ XSL/XSLT

Extensional Relational Database (ERDB)

▶ Decision Rule Mining in Rough Set Theory

External Hashing

▶ Hash-based Indexing

Extraction, Transformation, and Loading

Panos Vassiliadis[1], Alkis Simitsis[2]
[1]University of Ioannina, Ioannina, Greece
[2]IBM Almaden Research Center, San Jose, CA, USA

Synonyms

ETL; ETL process; ETL tool; Data warehouse back stage; Data warehouse refreshment

Definition

Extraction, transformation, and loading (ETL) processes are responsible for the operations taking place in the background of a data warehouse architecture. In a high level description of an ETL process, first, the data are extracted from the source data stores that can be on-line transaction processing (OLTP) or legacy systems, files under any format, web pages, various kinds of documents (e.g., spreadsheets and text documents) or even data coming in a streaming fashion. Typically, only the data that are different from the previous execution of an ETL process (newly inserted, updated, and deleted information) should be extracted from the sources. Secondly, the extracted data are propagated to a special-purpose area of the warehouse, called the data staging area (DSA), where their transformation, homogenization, and cleansing take place. The most frequently used transformations include filters and checks to ensure that the data propagated to the warehouse respect business rules and integrity constraints, as well as schema transformations that ensure that data fit the target data warehouse schema. Finally, the data are loaded to the central data warehouse (DW) and all its counterparts (e.g., data marts and views). In a traditional data warehouse setting, the ETL process periodically refreshes the data warehouse during idle or low-load, periods of its operation (e.g., every night) and has a specific time-window to complete. Nowadays, business necessities and demands require near real-time data warehouse refreshment and significant attention is drawn to this kind of technological advancement.

Historical Background

Despite the fact that ETL obtained its separate existence during the first decade of the twenty-first century, ETL processes have been a companion to database technology for a longer period of time, in fact, from the beginning of its existence. During that period, ETL software was just silently hidden as a routine programming task without any particular name or individual importance. ETL was born on the first day that a programmer constructed a program that takes records from a certain persistent file and populates or enriches another file with this information. Since then, any kind of data processing software that reshapes or filters records, calculates new values, and populates another data store than the original one is a form of an ETL program.

The earliest form of ETL system goes back to the EXPRESS system [13] that was intended to act as an engine that produces data transformations given some data definition and conversion nonprocedural statements. In later years, during the early days of data integration, the driving force behind data integration were wrapper-mediator schemes; the construction of the wrappers is a primitive form of ETL scripting [12]. In the mid 1990s, data warehousing came in the central stage of database research and still, ETL was there, but hidden behind the lines. Popular books [3] do not mention ETL at all, although the different parts (transformation, cleansing, staging of intermediate data, and loading) are all covered – even if this is done very briefly at times (it is also noteworthy that the third edition of the same book in 2003 mentions ETL, although briefly). At the same time, early research literature treated data warehouses as collections of materialized views since this abstraction was simple and quite convenient for the formulation of research problems.

During the '00s, it became increasingly prevalent that ETL is really important for data integration tasks since it is costly, labor-intensive, and mission-critical – and for all these factors, important for the success of a data warehousing project. The difficulty lies in the combination of data integration and data cleaning tasks: as a typical integration problem, where data are moved and transferred from a certain data store to another, the details of schema and value mappings play a great role. At the same time, the data can be in highly irregular state (both in terms of duplicates, constraint, and business rule violations, and in terms of irregularities in the internal structure of fields – e.g., addresses). Therefore, finding effective ways for taming this nonregularity into a typical relational structure is very hard. Additionally, there are a number of persistent problems: ETL processes are hard to standardize, optimize, and execute in a failure-resilient manner (and thus the problem is hard to solve).

Foundations

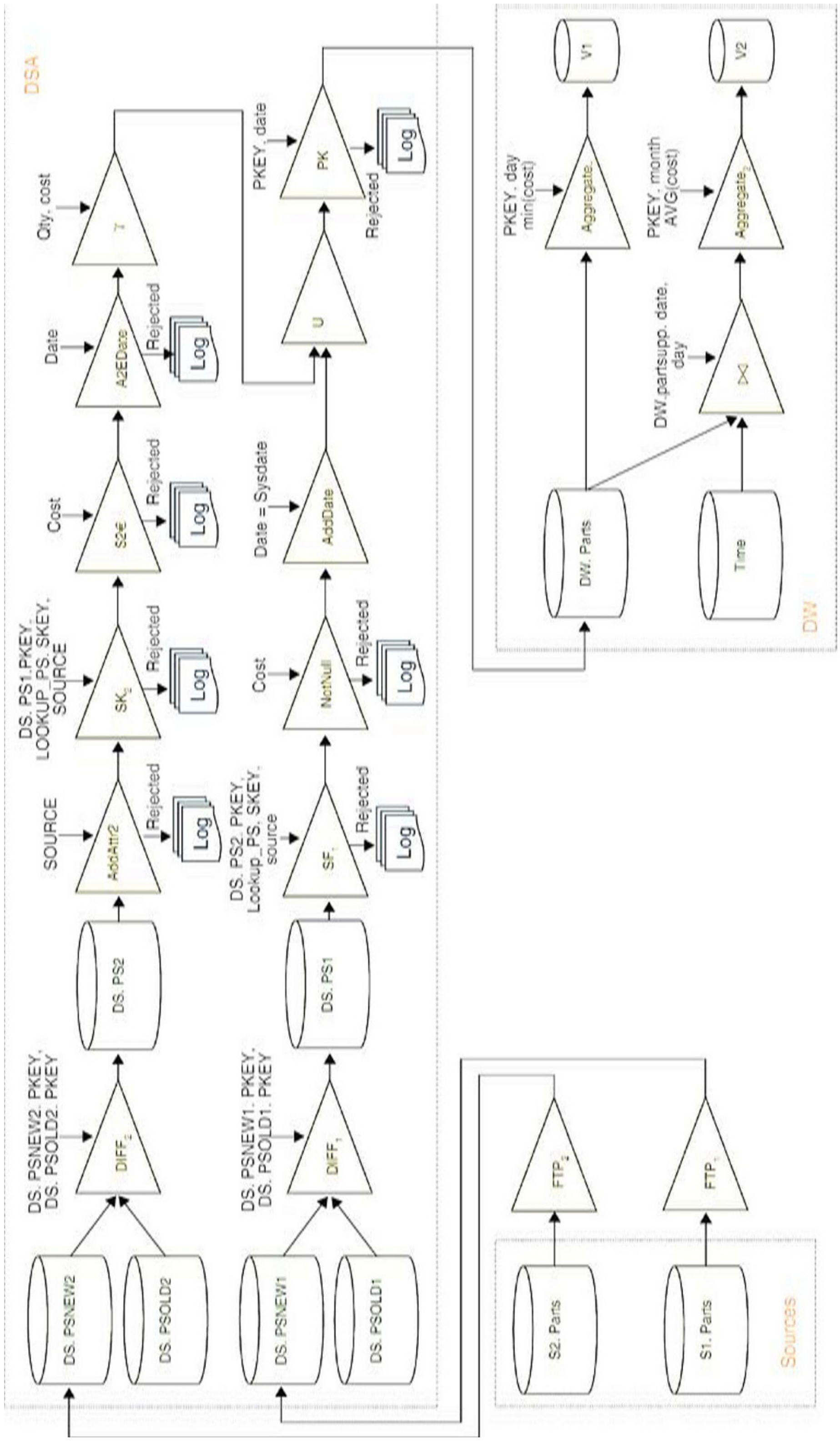
PART I. General description of an ETL process

Intuitively, an ETL process can be thought of as a directed acyclic graph, with activities and record sets forming the nodes of the graph, and input–output relationships between nodes forming its edges. Observe Figure 1, where two sources are depicted in the lower left part of the figure with the names S_1*.PARTS* and S_2*.PARTS*. The data from these sources must be propagated to the target data warehouse fact table *DW.PARTS*, depicted at the bottom right part of the figure. Moreover, the newly inserted records must be further propagated to refresh aggregate views V_1 and V_2. (In fact, the views of the example can be anything among materialized views that are populated by the ETL process, materialized views automatically refreshed by the DBMS, convenient abstractions of data marts, reports, and spreadsheets that are placed in the enterprise portal for the end-users to download, and any other form of aggregated information that is published in some form to the end-users of the warehouse.) The whole process of populating the fact table and any of its views is facilitated by a *workflow of activities* that perform all the appropriate filtering, intermediate data staging, transformations and loadings. The upper part of Figure 1 depicts how the data (which are extracted as snapshots of the sources) are transported to a special purpose intermediate area of the warehouse, called the data staging area (DSA) [4], where the transformation phase takes place. First, the snapshots are compared with their previous versions, so that newly inserted or updated data are discovered. Secondly, these new data are stored in a hard disk so that in the case of a failure, the whole process should not start from scratch. Then, the data pass through several filters or transformations and they are ultimately loaded in the data warehouse. As mentioned, the warehouse fact or dimension tables are not necessarily the end of the journey for the data: at the bottom right of Figure 1, a couple of materialized views (standing as abstractions of reports, spreadsheets, or data marts for the sake of the example) are also refreshed with newly incoming data.

PART II. Individual steps

Extraction. The extraction step is conceptually the simplest task of all, with the goal of identifying the correct subset of source data that has to be submitted to the ETL workflow for further processing. As with the rest of the ETL process, extraction also takes place at idle times of the source system – typically at night. Practically, the task is of considerable difficulty, due to two technical constraints:

- The source should incur minimum overhead during the extraction, since other administrative activities also take place during that period.
- Both for technical and political reasons, administrators are quite reluctant to accept major



Extraction, Transformation, and Loading. Figure 1. An example ETL workflow.

interventions to their system's configuration; therefore, there must be minimum interference with the software configuration at the source side.

Depending on the technological infrastructure and the nature of the source system (relational database, COBOL file, spreadsheet, web site etc.) as well as the volume of the data that has to be processed, different policies can be adopted for the extraction step, which usually is also called "change data capture." The most naïve possibility involves extracting the whole source and processing it as if the original first loading of the warehouse was conducted. A better possibility involves the extraction of a snapshot of data, which is subsequently compared with the previous snapshot of data (either at the source, or the DSA side) and insertions, deletions, and updates are detected. In this case, there is no need to further process the data that remain the same. The work presented in [5] is particularly relevant in this context. Another possibility involves the usage of triggers in the source that are activated whenever a modification takes place in the source database. Obviously, this can be done only if the source database is a relational system; most importantly though, both the interference with the source system and the runtime overhead incurred are deterring factors with respect to this option. An interesting possibility, though, involves "log sniffing," i.e., the appropriate parsing of the log file of the source. In this case, all modifications of committed transactions are detected and they can be "replayed" at the warehouse side.

A final point in the extraction step involves the necessity of encrypting and compressing the data that are transferred from the source to the warehouse, for security and network performance reasons, respectively.

Transformation. Depending on the application and the tool used, ETL processes may contain a plethora of transformations. In general, the transformation and cleaning tasks deal with classes of conflicts and problems that can be distinguished at two levels [7]: the schema and the instance level. In this article, a broader classification of the problems is presented that includes value-level problems, too.

- *Schema-level problems*. The main problems with respect to the schema level are (a) naming conflicts, where the same name is used for different objects (homonyms) or different names are used for the same object (synonyms), and (b) structural conflicts, where one must deal with different representations of the same object in different sources, or converting data types between sources and the warehouse.
- *Record-level problems*. The most typical problems at the record level concern duplicated or contradicting records. Furthermore, consistency problems concerning the granularity or timeliness of data occur, since the designer is faced with the problem of integrating data sets with different aggregation levels (e.g., sales per day vs. sales per year) or reference to different points in time (e.g., current sales as of yesterday for a certain source vs. as of last month for another source).
- *Value-level problems*. Finally, numerous low-level technical problems may be observed in different ETL scenarios. To mention a few, there may exist problems in applying format masks, like for example, different value representations (e.g., for sex: "Male," "M," "1"), or different interpretation of the values (e.g., date formats: American "mm/dd/yy" vs. European "dd/mm/yy"). Other value-level problems include assigning surrogate key management, substituting constants, setting values to NULL or DEFAULT based on a condition, or using frequent SQL operators like UPPER, TRUNC, and SUBSTR.

To deal with such issues, the integration and transformation tasks involve a wide variety of functions, such as normalizing, denormalizing, reformatting, recalculating, summarizing, merging data from multiple sources, modifying key structures, adding an element of time, identifying default values, supplying decision commands to choose between multiple sources, and so forth.

Loading. The end of the source records' journey through the ETL workflow comes with their loading to the appropriate table. A typical dilemma faced by inexperienced developers concerns the choice between bulk loading data through a DBMS-specific utility or inserting data as a sequence of rows. Clear performance reasons strongly suggest the former solution, due to the overheads of the parsing of the insert statements, the maintenance of logs and rollback-segments (or, the risks of their deactivation in the case of failures). A second issue has to do with the possibility of efficiently discriminating records that are to be inserted for the first time, from records that act as updates to previously loaded data. DBMSs typically support some declarative way to deal with this problem (e.g., Oracle's MERGE command [9]). In addition, simple SQL commands are not sufficient since the "open-loop-fetch" technique, where records are inserted one by one, is extremely slow for the vast volume of data to be loaded in the warehouse. A third performance issue that has to be taken into consideration has to do with the existence of indexes, materialized views, or both, defined over the warehouse relations. Every update to these relations automatically incurs the overhead of maintaining the indexes and the materialized views.

PART III. Global picture revisited

Design and Modeling. After the above analysis, one can understand that the design of ETL processes is crucial and complex at the same time. There exist several difficulties underlying the physical procedures already described.

At the beginning of a data warehousing project, the design team in cooperation with the business managers have to clarify the requirements, identify the sources and the target, and determine the appropriate transformations for that specific project; i.e., the first goal is to construct a *conceptual design of the ETL process* [8,18,20]. The most important part of this task is to identify the schema mappings between the source and the target schemata. This procedure is usually done through analysis of the structure and content of the existing data sources and their intentional mapping to the common data warehouse model. Possible data cleaning problems can also be detected at this early part of the design. Conceptual modeling formalisms (both ad- hoc and UML-based) have been proposed in the literature [8,18,20]. In addition, several research approaches have been proposed toward the automation of the schema mapping procedure [10]; one of the most prominent examples is CLIO, which has also been adapted by commercial solutions proposed by IBM [2]. However, so far, the automatic schema mapping supports cases of simple ETL transformations. On the contrary, there exists a semi-automatic method that uses ontologies and semantic web technology to infer the appropriate interattribute mappings along with the respective transformations needed in an ETL process [16].

When this task ends, the design involves the *specification of a primary data flow* that describes the route of data from the sources toward the data warehouse, as they pass through the transformations of the workflow. The execution sequence of the workflow's transformation is determined at this point. The data flow defines what each process does and the execution plan defines in which order and combination. The flow for the logical exceptions – either integrity or business rules violations is specified, too. Control flow operations that take care of monitoring or failure occurrences can also be specified. Most ETL tools provide the designer with the functionality to construct both the data and the control flow for an ETL process.

Optimization. Usually, an ETL workflow must be completed in a specific time window and this task is realized periodically; e.g., each night. In the case of a failure, the quick recovery of the workflow is also important [6]. Hence, for performance reasons, it is necessary to optimize the workflow's execution time. Currently, although the leading commercial tools provide advanced GUI's for the design of ETL scenarios, they do not support the designer with any technique to optimize the created scenarios. Unlike relational querying, where the user declaratively expresses a query in a high-level language and the DBMS optimizer decides the physical execution of the query automatically, in the case of ETL workflows it is the designer who must decide the order and physical implementation for the individual activities.

Practical alternatives involve (a) letting the involved DBMS (in the case of DBMS-based scripts) do the optimization and (b) treating the workflow as a big multi-query. The solution where optimization is handed over to the DBMS for execution is simply not sufficient, since DBMS optimizers can interfere only in portions of a scenario and not in its entirety. Concerning the latter solution, it should be stressed that *ETL workflows are not big queries*, since:

- It is not possible to express all ETL operations in terms of relational algebra and then optimize the resulting expression as usual. In addition, the cases of functions with unknown semantics – "black-box" operations – or with "locked" functionality – e.g., an external call to a DLL library – are quite common.
- Failures are a critical danger for an ETL workflow. The staging of intermediate results is often imposed by the need to resume a failed workflow as quickly as possible.
- ETL workflows may involve processes running in separate environments, usually not simultaneously and under time constraints; thus their cost

estimation in typical relational optimization terms is probably too simplistic.

All the aforementioned reasons can be summarized by mentioning that neither the semantics of the workflow can always be specified, nor its structure can be determined solely on these semantics; at the same time, the research community has not come-up with an accurate cost model so far. Hence, it is more realistic to consider ETL workflows as complex transactions rather than as complex queries. Despite the significance of such optimization, so far the problem has not been extensively considered in research literature, with results mainly focused on the black-box optimization at the logical level, concerning the order with which the activities are placed in the ETL workflow [14,15].

Implementation and Execution. An ETL workflow can either be hand-coded, or specified and executed via an ETL tool. The hand-coded implementation is a frequent choice, both due to the cost of ETL tools and due to the fact that developers feel comfortable to implement the scripts that manipulate their data by themselves. Typically, such an ETL workflow is built as the combination of scripts written in some procedural languages with high execution speed (for example, C or Perl) or some vendor specific database language (PL\SQL, T-SQL, and so on). Alternatively, ETL tools are employed, mainly due to the graphical programming interfaces they provide as well as for their reporting, monitoring, and recovery facilities.

Key Applications

ETL processes constitute the backbone of the data warehouse architecture. The population, maintenance, evolution, and freshness of the warehouse depends heavily on its backstage where all the ETL operations are taken place. Hence, in a corporate environment there is a necessity for a team devoted to the design and maintenance of the ETL functionality.

However, ETL is not useful only for the refreshment of large data warehouses. Nowadays, with the advent of Web 2.0 new applications have emerged. Among them, mashups are web applications that integrate data that are dynamically obtained via web-service invocations to more than one sources into an integrated experience. Example applications include Yahoo Pipes (http://pipes.yahoo.com/), Google Maps (http://maps.google.com/), IBM Damia (http://services.alphaworks.ibm.com/damia/), and Microsoft Popfly (http://www.popfly.com/). Under the hood, the philosophy for their operation is "pure" ETL. Although, the extraction phase mainly contains functionality that allows the communication over the web, the transformations that constitute the main flow resemble those which are already built-in in most ETL tools. Different applications are targeting to gather data from different users, probably in different formats, and try to integrate them into a common repository of datasets; an example application is the Swivel (http://www.swivel.com/).

Future Directions

Although the ETL logic is not novel in computer science, several issues still remain open. A main open problem in the so called traditional ETL is the agreement upon a unified algebra and/or a declarative language for the formal description of ETL processes. However, there are some first results in the context of the data exchange problem [1]. As already mentioned, the *optimization* of the whole ETL process and of any individual transformation operators pose interesting research problems. In this context, parallel processing of ETL processes is of particular importance. Finally, *standardization* is a problem that needs an extra note of attention. The convergence toward a globally accepted paradigm of thinking and educating computer scientists on the topic is a clear issue for the academic community.

However, the ETL functionality expands into new areas beyond the traditional data warehouse environment, where the ETL is executed off-line, on a regular basis. Such cases include but are not limited to: (a) *On-Demand ETL processes* that are executed sporadically (typically for Web data), and they are manually initiated by some user demand [11]; (b) *Stream ETL* that involves the possible filtering, value conversion, and transformations of incoming streaming information in a relational format [11]; (c) *(near)Real-Time ETL* that captures the need for a data warehouse containing data as fresh as possible.

Finally, with the evolution of the technology and the broader use of internet from bigger masses of users, the interest is moved also *to multiple types of data*, which do not necessarily follow the traditional relational format. Thus, modern ETL applications should also handle novel kinds of data, like XML, spatial, biomedical or multimedia data efficiently.

Data Sets

Benchmarking the ETL process is a clear problem nowadays (2007). The lack of any standard, principled experimental methodology with a clear treatment of the workflow complexity, the data volume, the amount of necessary cleaning, and the computational cost of individual activities of the ETL workflows is striking. The only available guidelines for performing a narrow set of experiments are given in the TPC-DS standard [17], and the first publicly available benchmark for ETL processes is presented in [19].

Cross-references

- ▶ Active Data Warehousing
- ▶ Data Cleaning
- ▶ Data Warehouse
- ▶ Multidimensional Modeling
- ▶ (near)Real-Time Data Warehousing

Recommended Reading

1. Fagin R., Kolaitis P.G., and Popa L. Data exchange: getting to the core. ACM Trans. Database Syst., 30(1):174–210, 2005.
2. Haas L.M., Hernández M.A., Ho H., Popa L., and Roth M. Clio grows up: from research prototype to industrial tool. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 805–810.
3. Inmon W. Building the Data Warehouse, 2nd edn. Wiley, New York, 1996.
4. Kimbal R., Reeves L., Ross M., and Thornthwaite W. The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses. Wiley, New York, 1998.
5. Labio W. and Garcia-Molina H. Efficient snapshot differential algorithms for data warehousing. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 63–74.
6. Labio W., Wiener J.L., Garcia-Molina H., and Gorelik V. Efficient resumption of interrupted warehouse loads. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 46–57.
7. Lenzerini M. Data integration: a theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
8. Luján-Mora S., Vassiliadis P., and Trujillo J. Data Mapping Diagrams for Data Warehouse Design with UML. In Proc. 23rd Int. Conf. on Conceptual Modeling, 2004, pp. 191–204.
9. Oracle , Oracle 9i SQL Reference. Release 9.2. 2002.
10. Rahm E. and Bernstein P.A. A survey of approaches to automatic schema matching. VLDB J., 10(4):334–350, 2001.
11. Rizzi S., Abelló A., Lechtenbörger J., and Trujillo J. Research in data warehouse modeling and design: dead or alive? In Proc. 9th ACM Int. Workshop on Data Warehousing and OLAP, 2006, pp. 3–10.
12. Roth M.T. and Schwarz P.M. Don't scrap it, wrap it! A wrapper architecture for legacy data sources. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 266–275.
13. Shu N.C., Housel B.C., Taylor R.W., Ghosh S.P., and Lum V.Y. EXPRESS: a data extraction, processing, and restructuring system. ACM Trans. Database Syst., 2(2):134–174, 1977.
14. Simitsis A., Vassiliadis P., and Sellis T.K. Optimizing ETL processes in data warehouses. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 564–575.
15. Simitsis A., Vassiliadis P., and Sellis T.K. State-space optimization of ETL workflows. IEEE Trans. Knowl. Data Eng., 17 (10):1404–1419, 2005.
16. Skoutas D. and Simitsis A. Designing ETL processes using semantic web technologies. In Proc. 9th ACM Int. Workshop on Data Warehousing and OLAP, 2006, pp. 67–74.
17. TPC, TPC-DS (Decision Support) specification, draft version 52. 2007.
18. Trujillo J. and Luján-Mora S. A UML based approach for modeling ETL processes in data warehouses. In Proc. 22nd Int. Conf. on Conceptual Modeling, 2003, pp. 307–320.
19. Vassiliadis P., and Karagiannis A., and Tziovara V., and Simitsis A. Towards a Benchmark for ETL Workflows. In Proc. 5th Int. Workshop on Quality in Databases, 2007, pp. 49–60.
20. Vassiliadis P., Simitsis A., and Skiadopoulos S. Conceptual modeling for ETL processes. In Proc. 5th ACM Int. Workshop on Data Warehousing and OLAP, 2002, pp. 14–21.

Extrinsic Time

▶ Transaction Time

Faceted Browsing

► Faceted Search

Faceted Classifications

► Lightweight Ontologies

Faceted Search

SUSAN DUMAIS
Microsoft Research, Redmond, WA, USA

Synonyms

Dynamic taxonomies; Faceted browsing; Hierarchical faceted metadata

Definition

The term *facet* means “little face” and is often used to describe one side of a many-sided object, especially a cut gemstone. In the context of information science, where the item being described is an information object, facets could refer to the object’s author, date, topic, etc. Facets are used to describe both the organization of information (faceted classification), and to interface techniques that provide flexible access to that information (faceted search). The motivation for faceted classification and search is that any single organizational structure is too limiting to accommodate access to complex domains. Multiple independent facets provide alternative ways of getting to the same information, thus supporting a wider range of end-user tasks and knowledge. The fields of faceted classification, information architecture, and data modeling provide theory and methods for identifying and organizing facets. The user interface challenge for faceted systems is in managing this added complexity, especially when working with very large and diverse information collections. Most interfaces to faceted information provide support for structured browsing (faceted navigation or browsing). In addition, some systems offer search capabilities and, more generally, tightly coupled views of the same information. This entry covers both the organization of information using facets, and the design of user interfaces to support searchers in accessing the information since the two aspects are closely related and should be considered together in designing information systems.

The term facet is widely used in the information science community. In other disciplines attribute, dimension, metadata, property, or taxonomy are used to refer to similar concepts. Faceted search is used in this entry to refer to flexible access to faceted information, using both browsing and search. Other terms such as hierarchical faceted metadata, faceted search and browsing, and dynamic taxonomies refer to similar concepts.

Historical Background

Shiyali Ramamrita Ranganathan, an Indian mathematician, first introduced the term “facet” in information science in the 1930s when he developed a theory of facet analysis, culminating with the publication of his book *Colon Classification* in 1933 [7]. The name comes from the use of colons to separate different facets in writing composite class numbers. Ranganathan applied the principles of faceted classification to organize all of human knowledge in libraries using five main facets – personality, matter or property, energy, space, and time. The Colon Classification system is still used in libraries in India, and the principles and techniques of facet analysis have been more widely influential as reviewed by Vickery [13]. Faceted classification is also called analytico-synthetic classification, after the two main processes involved – analysis which breaks down each subject into its basic concepts or facets, and synthesis which combines concepts to describe the subset of information of interest. Thus, in faceted classification, new classes are created by a searcher from the logical combination of facets and facet values. This is often contrasted with enumerative classification in which all classes of interest are specified by the indexer at the time of creation. Most well-known library classification systems (Dewey Decimal Classification (DDC), Universal Decimal Classification (UDC) and Library of Congress Classification (LCC)) are enumerative systems, although DDC and UDC have elements of faceted classifications as well.

In the 1950s, Calvin Mooers (who coined the term *information retrieval*) and Mortimer Taube developed new methods for searching information. Before this time, most access was based on pre-coordinated index terms. Index terms were assigned to objects and only those specific index terms could be used to retrieve items. This is very much the way that back-of-the-book indices work, with the index creator anticipating the compound subjects that people will want to search for. In contrast, in Mooers' Zatocoding system [5], there was one punch card per item and many index terms coded by notch positions on the cards. At retrieval time, a searcher used needles to isolate items sharing a common term, and multiple terms could be combined using Boolean logic. Taube, who coined the term *coordinate indexing*, developed a system in which each card represented a term (the so-called uniterm system), along with a coding of the documents to which it referred [12]. Again, flexible post-coordination of terms at retrieval time was supported. Although the early coordinate indexing systems focused on index terms, the emphasis on analyzing items of information so that retrieval could be performed by the Boolean operations on simple index entries made these ideas widely applicable to online faceted systems. The first online retrieval systems appeared in the 1960s, many containing quite sophisticated retrieval functionality. The 1980s and 1990s saw the development of much richer systems that combined browsing and searching in more flexible ways to support non-professional end users in finding information. Some early systems showing key capabilities of faceted search include Pollitt [6] and Remde et al. [8]. As described in more detail below, many of the ideas explored in these systems are seen today in faceted search systems and on some web sites, notably e-commerce sites.

Foundations

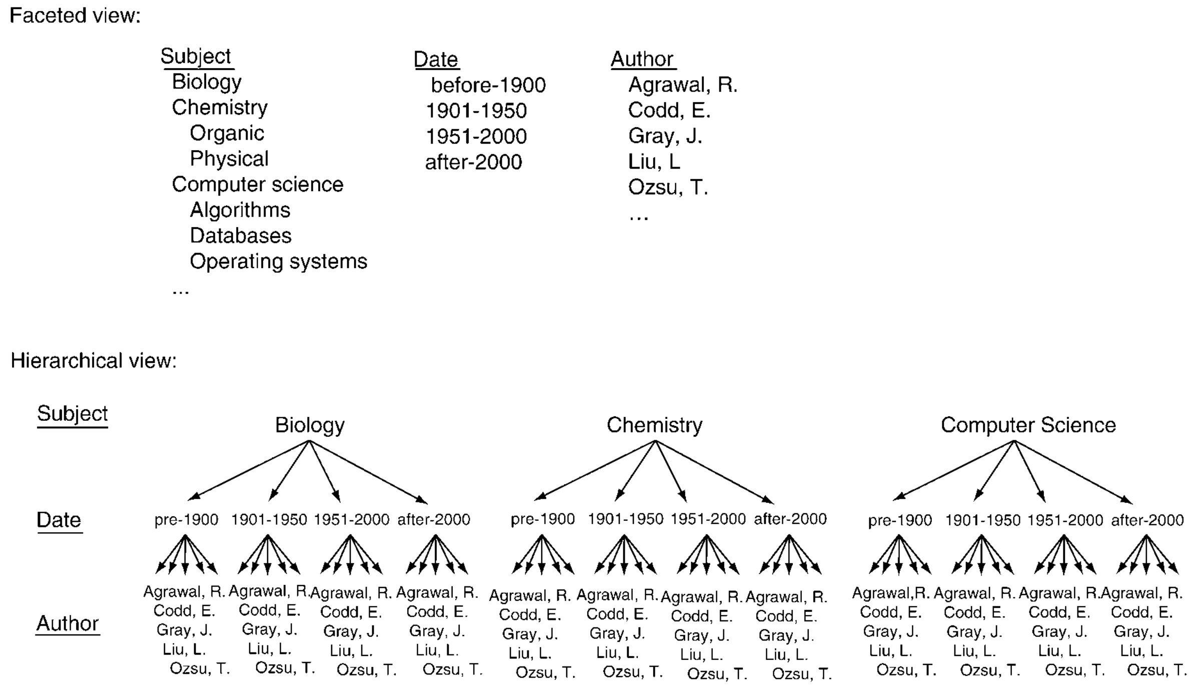
An Example

As a concrete example, consider organizing and providing access to a large collection of diverse technical publications. One way to do so is by general topic or subject area, much as the entries in this encyclopedia are organized. But there are many other facets in addition to Subject that can be used to characterize each publication – for example: Publication date, Author, Author's institution, Publication, Genre, Language, Length, Price, Tags, Download frequency, Citations by other publications, etc. In a faceted organization, each item is characterized by a value on each facet, and there are no explicit relationships among the facets. (There are a wide range of implicit relationships based on which items share facet values.) A single hierarchical structure can, of course, represent these same facets, with each level representing one of the facets. There are added storage costs since each facet needs to be replicated for every subcategory. More importantly, a single hierarchy imposes one fixed structure for navigation; each facet must be visited in the order provided by the hierarchy. Facets, on the other hand, provide many alternative ways of getting to the same information and can be combined in any order, thus providing much greater flexibility for searchers to find information. (See Fig. 1.)

Faceted Organization

Ranganathan pioneered principles and methods for identifying facets which Vickery and others extended and summarized. More recently the field of information architecture has developed these and related ideas in the context of large electronically available collections. Two important tasks are to identify the most appropriate facets for a collection and to assign a value to each item on each facet. Identifying facets is a challenging task and depends on the details of the collection, anticipated user tasks, etc. and will not be discussed in detail here. Rosenfeld and Morville [9] provide an overview of recent work on information architecture and data modeling.

The principles of faceted organization are widely applicable and flexible. Each individual *facet* can be organized in many ways, e.g., hierarchically, as a flat list, or using other relations. Each facet has a name (e.g., Subject) as well as values or labels (e.g., Biology, Chemistry, Computer Science). Each *item* in the collection is assigned one or more values on every facet.



Faceted Search. Figure 1. Comparison of faceted and hierarchical views.

For example, a technical publication will usually have only one Length or Date, but may have many Subjects, Authors or Tags. A probability or confidence can be associated with each value, as often happens when values are assigned algorithmically, although interfaces that expose this are rare. Any collection of items can be represented by a faceted organization – documents, web pages, images, videos, products, people, etc.

Once the facets for a collection have been determined, each item must be assigned values on every facet. In early library systems, all facets were assigned by trained human indexers, and many systems today still depend on human assignment of values for at least some of their facets. Machine learning techniques can also be used to provide facet values for items. Typically this involves a supervised learning phase in which items with known labels are used to build a predictive model, and an evaluation phase in which labels are assigned to new items; text classification systems are a widely-used example of this process. Sometimes values are captured implicitly via an application or use. For example, in email the sender and recipient are required for successful use of the application and are thus captured. Facets that reflect interaction with items (e.g., Download frequency, User-generated tags) are also captured implicitly as part of ongoing interactions with applications.

A key motivation for faceted systems is that any single organizational structure is too limiting to accommodate access to complex domains. Multiple facets provide many different ways of organizing information, thus enabling flexible end-user access to the information. Facets are a way of conveniently referring to a subset of items in the collection. Facets can be specified in any order, and the resulting sets can be combined using Boolean logic. The most common method for doing so (in interfaces) is to use AND to combine values from different facets and OR to combine values from within the same facet. Thus the resulting query is a conjunct of disjuncts over the selected facet labels. Other combination methods are possible, although there are interesting challenges in providing interfaces to do so. The ability to specify facets in any order and to combine them using Boolean logic illustrates the power and flexibility of faceted organization.

Compared to a single hierarchical organization of information, faceted systems provide many advantages. First, facets provide an efficient way of discriminating among sets of items. A single hierarchy with 100,000 nodes would be required to represent all the combinations that can be instantiated by 5 facets each with 10 values. Second, the ability to combine facets in any order provides users with more flexibility to specify their interests using whatever facets they are confident about for a particular search (as described in more detail below). Finally, facets are easier to manage and update, and efficient to use as input to machine learning algorithms.

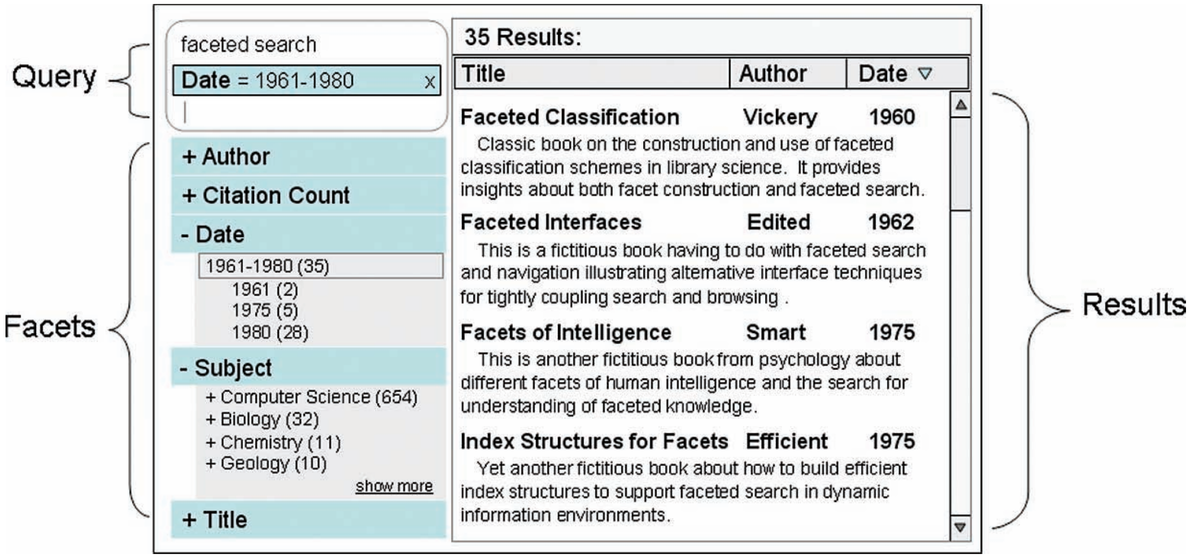
Faceted Access

Developing a faceted organization of information is only the first step in any application. Using this representation to support end-users in finding information or in understanding large multi-dimensional information spaces is critical for any successful application. In this entry, the term faceted access is used to refer to faceted browsing, search, and the integration of the two.

Several faceted search systems have been designed and deployed during the last two decades. The success of any system in supporting end-users is highly dependent on details of the domain of interest (e.g., searcher's tasks, their familiarity with the facets, etc.). A brief summary of the key components is outlined below. Most systems show the query, the facet structure, the subset of results currently specified, and sometimes a detailed view of an individual item. (See Fig. 2.)

Browsing The most common access technique supported by facets is navigation. Faceted navigation is a way to browse information, guided by the facet structure. Selecting a facet label (e.g., Subject=Computer Science) results in a list of all items described by that label. Additional facet values can be selected at any time and in any order (e.g., Date=2000; Author= Dumais; Subject=Computer Science/Databases). Faceted browsing allows searchers to refine a long list of results, using multiple dimensions, in whatever order they choose. From the user experience perspective, the big challenge is in managing the scale and complexity of multiple facets, each of which may contain thousands of values organized in either a hierarchy or flat list. Consider how cumbersome it can be to navigate using the file explorer when there are hundreds of thousands of files. And this is only a single facet (based on file location); now add other facets such as file type, size, creation date, access patterns, etc. and the design issues become apparent.

One challenge is to provide compressed views of very large information structures in ways that balance overview (context) and detailed (focus) information. A common technique is to progressively expose details for hierarchical facets, guided by the user. Initially only the top level values for each facet are shown. Once a facet is selected the next level of labels are exposed, and previous levels may be collapsed to save space with only a breadcrumb remaining. Yee et al. [14] show several examples of this. Another technique is to truncate long lists of values by showing a few labels with an option to "see more." The values can be selected in any



faceted search

Date = 1961-1980 x

+ Author
+ Citation Count
- Date
1961-1980 (35)
1961 (2)
1975 (5)
1980 (28)
- Subject
+ Computer Science (654)
+ Biology (32)
+ Chemistry (11)
+ Geology (10)
show more
+ Title

35 Results:

Title	Author	Date ▽
Faceted Classification Classic book on the construction and use of faceted classification schemes in library science. It provides insights about both facet construction and faceted search.	**Vickery**	**1960**
Faceted Interfaces This is a fictitious book having to do with faceted search and navigation illustrating alternative interface techniques for tightly coupling search and browsing .	**Edited**	**1962**
Facets of Intelligence This is another fictitious book from psychology about different facets of human intelligence and the search for understanding of faceted knowledge.	**Smart**	**1975**
Index Structures for Facets Yet another fictitious book about how to build efficient index structures to support faceted search in dynamic information environments.	**Efficient**	**1975**

Faceted Search. Figure 2. Example faceted search interface, illustrating facet browsing, searching, and tight coupling of the two.

number of ways. In some cases an ordered list may be binned (e.g., instead of showing all unique dates, the dates can be grouped: before 1960; 1961–1980; 1981–2000; after 2000). In other cases the values are chosen alphabetically or numerically, to span the space, or based on the most common user tasks. And, in some cases, search over facet values is provided as a means of quickly winnowing down a long list of possible values (e.g., [2]).

Another challenge is to select labels that are understandable to users, thus enabling them to know which facet to select to get to the desired information. This is relatively easy for some facets, like Date or Language or Author, but more difficult for others like Subject (e.g., is a paper describing a user study of faceted navigation in Subject=Computer Science or Subject=Psychology or both or neither?). In cases like this, facet labels can be expanded by including examples of labels at the next level (e.g., Subject=Computer Science (Algorithms, Databases, Human-Computer Interaction), thus improving the "information scent" of the description. Many web directories like the Open Directory (http://www.dmoz.org) label their categories in this manner.

An important technique for providing additional guidance about what will be found by selecting a facet value is called "dynamic query previews" (e.g., Greene et al. [3] Shneiderman [14]). The basic idea is to show some preview of the results that each query will generate. For facet browsing, this is commonly done by showing the number of results for each facet label next to the label (e.g., Biology (32); Chemistry (1); Computer Science (654); Geology (0)). Showing previews prevents also prevents users from selecting facets that will return no results. There are interesting challenges in efficiently providing such previews in distributed networked environments, and in cases where the set of items has been generated by some other means like a search.

Facet labels and query previews aid searchers in selecting the right facets. Interfaces should also allow for *both* specialization (narrowing a search) and generalization (broadening a search). Techniques like breadcrumbs are used to represent the current state of the "query" as it evolves over time through interaction. This provides a visible and actionable representation – searchers are able to understand the current set of items, and to quickly generalize as well as specialize.

In addition to supporting facet navigation, the interface should also show some representation of the current set of results. A simple list is one method for doing so. Richer list views in which the searcher can control the sort order along one or more facets are also used. Results sets can also be grouped by facet value to enable a greater diversity of results to be shown and to further reinforce the facet structure. Grouping is typically done using the next level for hierarchically organized facets [14], or the facet used for sorting [2]. In addition, some interfaces allow users to specify queries by example. Given a result or set of results, a new query can be generated to find other items that share one or more facets.

Searching (and Integrated Browsing and Searching)
Search is often contrasted with browsing, and suggested as a means for providing access to large, unstructured collections of information. In web search engines, for example, a query results in a long unstructured list of results. Many faceted browsing systems also provide a search capability that is largely independent of the facet structure.

An improved user experience can be achieved by tightly coupling faceted browsing and searching capabilities. The facet structure can be leveraged in several ways to do this. A simple integration is to use the facets to navigate to a subset of items of interest and then to search within that. The converse is to start with search and to show the distribution of search results in terms of their corresponding facet structure. For example, a query for "faceted search" returns results that have a distribution of values on the Subject facet – e.g., Computer Science (654); Biology (32); Chemistry (1); Geology (0).

The tightest coupling is to view both search and navigation as techniques for specifying subsets of information, and to allow either method to be used at all times. Thus search and navigation can be combined in any order at any time, while maintaining a consistent representation and interface. The Flamenco [14] and Phlat [2] systems support this kind of seamless integration between search and navigation. Effective data structures and algorithms have been developed to support quickly finding sets of results and summarizing the distributions of their facet values [1,10].

Scale and Details An important design challenge in faceted interfaces is handling scale – the overall size of the collection, the number of different facets, and the number of values per facet (especially for multi-valued facets like names or user-generated tags). Displaying all values for all facets is not possible in most realistic applications without requiring users to spend most of their time scrolling through very long lists or managing multiple windows. Methods for progressively exposing more detail for hierarchically organized facets, or for showing a limited set of key values for each facet, with more alternatives available on demand, are important techniques for managing scale. Using the facet structure to tightly couple browsing and searching is increasingly being used in e-commerce applications. This provides the searcher with tools that support a wide variety of tasks and states of knowledge.

There are still some capabilities that are difficult to implement in a usable and discoverable fashion. Searching over facet values when they are many values (e.g., Author names, Tags), multi-select operations within a facet, and using the same facet structure to support both access and tagging are all still challenging design issues. Decisions about which facets to display and how to do so depend critically on the application domain – e.g., price is very important in shopping but less so in finding technical materials; supporting range queries makes sense for Price, but probably not for Subject. Hearst et al. [4] provide a number of examples of how usability tests guided important interface design choices. These evaluations examine outcome measures such as task completion time and accuracy, subjective preferences, interaction trajectories, etc., and are used to iterate on the design until user performance goals are achieved.

Key Applications

Faceted search systems have been developed for a many applications in e-commerce (music, books, e-bay), bibliographic databases, image collections, and for finding files and email in the Windows and Macintosh operating systems. The following is a small list of applications available on the web, covering different domains.

- EBay Express, http://www.express.ebay.com/–general product information
- Flamenco, http://flamenco.berkeley.edu/demos.html – prototype applications for art images, architecture images and nobel prize winners
- NCSU Libraries http://www2.lib.ncsu.edu/catalog/–bibliographic records
- Tower Records, http://www.tower.com/–music, videos and books

These applications all in fairly narrow vertical domains and have readily available facet information. In the Flamenco prototypes, hand-generated metadata was available for each collection. In the NCSU application, the facets were populated using bibliographic records. And, in the e-commerce applications, many of the facets displayed are required for business purposes (e.g., price, manufacturer, etc.) and thus readily available.

Future Directions

New interface techniques for managing scale in existing applications will continue to be a fruitful area for innovation. Algorithms and architectures that can provide real-time dynamic previews of results in the context of the facet structure are required for distributed or networked applications. Developing methods for representing and displaying a confidence score for each facet value is another new direction.

Finally, it is interesting to consider why faceted interfaces are not widely available for general web search. There have been some attempts to structure the web using a topic hierarchy like Open Directory (http://www.dmoz.org) or Yahoo! in its early days. However this represents only a single facet (topic), and facet values are available for only a small subset of web content. Facet values could be assigned automatically using text classification techniques (e.g., topic, genre, commerciality), and Web pages already have some structure that could support faceted access (e.g., site names, creation date, modification date, language, usage history). Understanding which facets are most important to support the variety of information needs that people use the web for, and handling large-scale dynamic collections will be key in determining the success of faceted search methods for web content.

Cross-references

▶ Browsing in Digital Libraries
▶ Cataloging in Digital Libraries
▶ Cross-Modal Multimedia Information Retrieval
▶ Digital Libraries
▶ Human-Computer Interaction
▶ Information Retrieval
▶ Ontology
▶ Presenting Structured Text Retrieval Results
▶ Searching Digital Libraries
▶ Text Categorization
▶ Text Indexing and Retrieval
▶ Web Information Retrieval Models

Recommended Reading

1. Bast H. and Weber I. The complete search engine: interactive, efficient and towards IR & DB integration. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 88–95.
2. Cutrell E., Robbins D.C., Dumais S.T., and Sarin R. Fast, flexible filtering with Phlat – personal search and organization made easy. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2006, pp. 261–270.
3. Greene S., Marchionini G., Plaisant C., and Shneiderman B. Previews and overviews in digital libraries: designing surrogates to support visual information-seeking. J. Am. Soc. Inform. Sci., 51(3):380–393, 2000.
4. Hearst M., Smalley P., and Chander C. Faceted metadata for information architecture and search. Course at SIGCHI Conf. on Human Factors in Computing Systems, 2006.
5. Mooers C.N. Zatocoding applied to mechanical organization of knowledge. Am. Doc., 2(1):20–32, 1951.
6. Pollitt A.S. A common query interface using MenUSE – A menu-based user search engine. In Proc. 12th Int. Online Information Meeting, 1988, pp. 445–457.
7. Ranganathan S.R. Colon classification. The Madras Library Association, Madras, India and Goldston, London, 1933.
8. Remde J.R., Gomez L., and Landauer T.K. SuperBook: An automatic tool for information exploration. In Proc. Hypertext'87, Conf. Chapel Hill, NC, 1987, pp. 175–188.
9. Rosenfeld L. and Morville P. Information Architecture for the World Wide Web. O'Reilly Media, Sebastapol, CA, 2006.
10. Sacco G.M. Dynamic taxonomies: a model for large information bases. IEEE Trans. Knowl. Data Eng., 12(2):468–479, 2000.
11. Shneiderman B. Dynamic queries for visual information seeking. IEEE Softw., 11(6): 70–77, 1994.
12. Taube M. et al. Studies in Coordinate Indexing (vols. 1–5). Documentation Incorporation, Washington, DC, 1953–1959.
13. Vickery B.C. Faceted Classification: A Guide to the Construction and Use of Special Schemes. ASLIB, London, 1960.
14. Yee K., Swearingen K., Li K., and Hearst M. Faceted metadata for image search and browsing. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2003, pp. 401–408.

Facility-Location Problem

▶ Resource Allocation Problems in Spatial Databases

Fact-Oriented Modeling

▶ Object-Role Modeling

Failure Handling

▶ Crash Recovery
▶ Logging and Recovery

False Negative Rate

▶ Precision and Recall

Fault Tolerant Applications

▶ Application Recovery

Fault-Tolerance

▶ Replication for High Availability

Fault-Tolerance and High Availability in Data Stream Management Systems

MAGDALENA BALAZINSKA[1], JEONG-HYON HWANG[2], MEHUL A. SHAH[3]
[1]University of Washington, Seattle, WA, USA
[2]Brown University, Providence, RI, USA
[3]HP Labs, Palo Alto, CA, USA

Definition

Just like any other software system, a data stream management system (DSMS) can experience failures of its different components. Failures are especially common in *distributed* DSMSs, where query operators are spread across multiple processing nodes, i.e., independent processes typically running on different physical machines in a local-area network (LAN) or in a wide-area network (WAN). Failures of processing nodes or failures in the underlying communication network can cause continuous queries (CQ) in a DSMS to stall or produce erroneous results. These failures can adversely affect critical client applications relying on these queries.

Traditionally, availability has been defined as the fraction of time that a system remains operational and properly services requests. In DSMSs, however, availability often also incorporates end-to-end latencies as applications need to quickly react to real-time events and thus can tolerate only small delays. A DSMS can handle failures using a variety of techniques that offer different levels of availability depending on application needs.

All fault-tolerance methods rely on some form of replication, where the volatile state is stored in independent locations to protect against failures. This entry describes several such methods for DSMSs that offer different trade-offs between availability and runtime overhead while maintaining consistency. For cases of network partitions, it outlines techniques that avoid stalling the query at the cost of temporary inconsistency, thereby providing the highest availability. This entry focuses on failures within a DSMS and does not discuss failures of the data sources or client applications.

Historical Background

Recently, DSMSs have been developed to support critical applications that must quickly and continuously process data as soon as they become available. Example applications include financial stream analysis and network intrusion detection (see Key Applications for more). Fault-tolerance and high availability are important for these applications because faults can lead to quantifiable losses. To support such applications, a DSMS must be equipped with techniques to handle both node and network failures.

All basic techniques for coping with failures involve some kind of replication. Typically, a system replicates the state of its computation onto independently failing nodes. It must then coordinate the replicas in order to recover properly from failures. Fault-tolerance techniques are usually designed to tolerate up to a predefined number, *k*, of simultaneous failures. Using such methods, the system is then said to be *k*-fault tolerant.

There are two general approaches for replication and coordination. Both approaches assume that the computation can be modeled as a deterministic state-machine [4,11]. This assumption implies that two non-faulty computations that receive the same input in the same order will produce the same output in the same order. Hereafter, two computations are called *consistent* if they generate the same output in the same order.

The first approach, known as the *state-machine* approach, replicates the computation on $k + 1 \geq 2$ independent nodes and coordinates the replicas by sending the same input in the same order to all [11]. The details of how to deliver the same input define the various techniques. Later sections in this entry describe variants that are specific to DSMSs. The state-machine approach requires $k + 1$ times the resources of a single replica, but allows for quick fail-over, so a failure causes little disruption to the output stream. This property is important for critical monitoring tasks such as intrusion detection that require low-latency results at all times.

The second general approach is known as *rollback recovery* [4]. In this approach, a system periodically packages the state of its computation into a *checkpoint*, and copies the checkpoint to an independent node or a non-volatile location such as disk. Between checkpoints, the system logs the input to the computation. Since disks have high latencies, existing fault-tolerance methods for DSMSs copy the checkpointed state to other nodes and maintain logs in memory. Upon failure, the system reconstructs the state from the most recent checkpoint, and replays the log to recover the exact pre-failure state of the computation. This approach has much lower runtime overhead at the expense of longer recovery times. It is useful in situations where resources are limited, the state of the computation is small, fault-tolerance is important, but rare moderate latencies are acceptable. An example application is fabrication-line monitoring using a server cluster with limited resources.

In some cases, users are willing to tolerate temporary inconsistencies to maintain availability at all times. One example is in the wide-area where network partitions are likely (e.g., large-scale network and system monitoring). To maintain availability in face of network partitions, the system must move forward with the computation ignoring the disconnected members. In this case, however, replicas that process different inputs will have inconsistent states. There are two general approaches for recovering from such inconsistencies after the network partition heals. One approach is to propagate all updates to all members and apply various rules for reconciling conflicting updates [6,9]. The other approach is to undo all changes performed during the partition and redo the correct ones [6,15].

This entry presents how these general approaches can be adapted to distributed DSMSs. The main challenge is to ensure that applications receive low-latency results during both normal processing and failures. To do so, the methods presented leverage the structure of continuous queries (CQs) in DSMSs.

This entry makes the following assumptions. A CQ is a connected directed-acyclic graph of query operators. The operators can be distributed among many processing nodes with possibly multiple operators per node. A processing node is the unit of failure. For simplicity of exposition, this entry focuses on the case of $k = 1$ (i.e., two query replicas) although all shown techniques can handle any k. This entry describes methods to tolerate node crash failures and temporary network partitions. Roughly speaking, a node has crashed if it visibly halts or simply becomes unresponsive [12]. A network partition splits nodes into two or more groups where nodes from one group cannot communicate with nodes in another.

Foundations

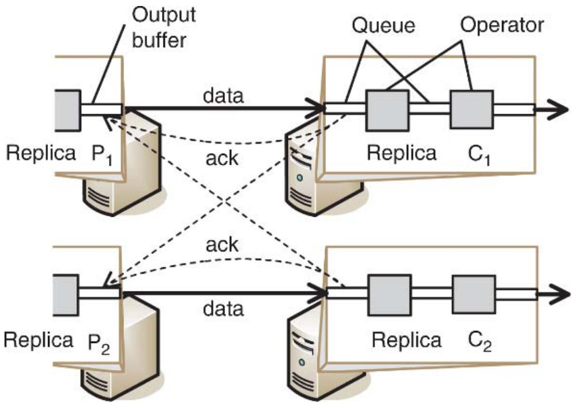
Techniques for Handling Crash Failures

This section describes new fault-tolerance techniques devised by applying the general fault-tolerance methods to continuous queries in DSMSs.

Active Replicas Active replicas are an application of the state-machine approach in which query operators are replicated and run on independently failing nodes. A simple variant of the active replicas approach uses the traditional process-pair technique to coordinate the replicas. The process-pair technique runs two copies of the query and specifies one to be the primary and the other to be the secondary. In this approach, the primary forwards all input, in the same order, to the secondary and works in lock-step with the secondary [5].

A DSMS can rely on a looser synchronization between the replicas by taking advantage of the structure of CQ dataflows. In a CQ dataflow, the operators obey a producer-consumer relationship. To provide high availability, the system replicates both the producer and consumer as illustrated in Fig. 1. In this model, there is no notion of a primary or secondary. Instead, each producer logs its output and forwards the output to its current consumer(s). Each consumer sends periodic acknowledgments to all producers to indicate that it has received the input stream up-to a certain point. An acknowledgment indicates that the input need not be resent in case of failure, so producers can truncate their output logs. Use of reliable, in-order network delivery (e.g., TCP) or checkpoints allows important optimizations where consumers send application-level acknowledgments to only a subset of producers [7,14].

The symmetric design of active replicas has some benefits. The normal-case behavior has few cases, so it is simple to implement and verify. Additionally, with sufficient buffering, each pipeline can operate at its own pace, in looser synchronization with the other.



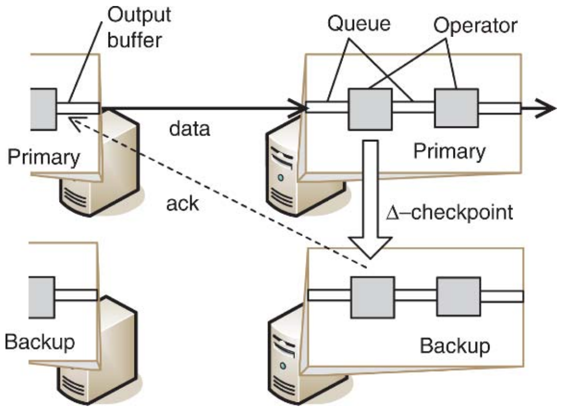
Fault-Tolerance and High Availability in Data Stream Management Systems. Figure 1. Active replicas. The operators on replicas P_1 and P_2 are the producers. The operators on C_1 and C_2 are the consumers.

The Flux [14] approach was the first to investigate this looser synchronization between replicated queries. Flux is an opaque operator that can be interposed between any two operators in a CQ. Flux implements a simple variant of this protocol and assists in recovery. The Borealis "Delay, Process, and Correct" (DPC) protocol [1,2] also uses the above coordination scheme, but differs from Flux in its recovery, as discussed later. The Flux and DPC approaches both ensure strict consistency in the face of crash failures: no duplicate output is produced and no output is lost.

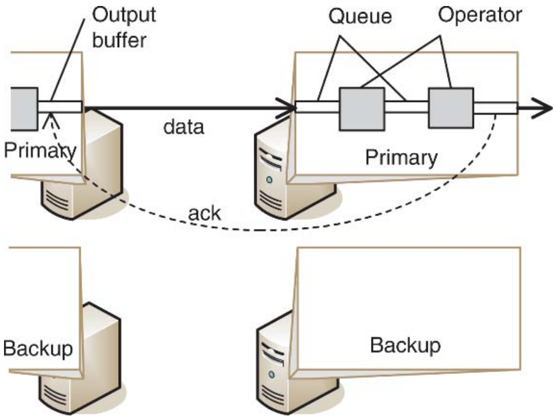
Passive Replicas There have been two applications of the rollback recovery approach to CQs [7,8]. The first, called *passive standby*, handles all types of operators. The second, called *upstream backup*, is optimized for more specific bounded-history operators that frequently arise in CQs.

In the passive standby approach, a primary node periodically checkpoints its state and sends that checkpoint to a backup. The state includes any data maintained by the operators and tuples stored in queues between operators. In practice, sending the entire state at every checkpoint is not necessary. Instead, each primary periodically performs only a *delta-checkpoint* as illustrated in Fig. 2. During a delta-checkpoint, the primary updates the backup by copying only the difference between its current state and the state at the time of the previous checkpoint.

Because of these periodic checkpoints, a backup always has its primary's state as of the last checkpoint.



Fault-Tolerance and High Availability in Data Stream Management Systems. Figure 2. Passive standby.



Fault-Tolerance and High Availability in Data Stream Management Systems. Figure 3. Upstream backup.

If the primary fails, the backup recovers by restarting from that state and reprocessing all the input tuples that the primary had processed since the last checkpoint. To enable backups to reprocess such input tuples, all primaries log their output tuples. If a downstream primary fails, each upstream primary re-sends its output tuples to the downstream backup. In a CQ, because the output of an operator can be connected to more than one downstream consumer operator, primaries discard logged output tuples only after *all* downstream backups have acknowledged a checkpoint.

For many important CQ operators, the internal state often depends only on a small amount of recent input. Examples of such operators include joins and aggregates with windows that span a short time-period or a small number of tuples. For such operators, DSMSs can use the upstream backup method to avoid any checkpointing overhead. In this approach, primaries log their output tuples, but backups remain idle as illustrated in Fig. 3. The primaries trim their logs based on notifications from operators 1-level (or more) downstream, indicating that the states of consuming operators no longer depend on the logged input. To generate these notifications, downstream operators determine, from their output, what logged input tuples can be safely discarded. If a primary fails, an empty backup rebuilds the latest state of the primary using the logs kept at upstream primaries.

Failure Recovery When a failure occurs, a DSMS must first detect and then recover from that failure. DSMSs detect failures using timeouts and, in general, rely on standard group membership mechanisms to keep consistent track of nodes entering and leaving the system [10,13]. Recovery ensues after failure detection.

There are two parts to failure recovery. The first part involves masking the failure by using the remaining replica to continue processing. For active replicas, this part is called *fail-over.* In both Flux and DPC, fail-over is straightforward. Consumers and producers adjust their connections to receive input data from or send output data to the remaining live copy of the failed node. To avoid stalls in the output stream, it is safe for active replicas to proceed with fail-over without waiting for group membership consensus [1,13]. For passive standby and upstream backup, this first part also involves bringing the state of the backup to the pre-failure state of the failed primary, as described earlier, before the backup starts sending data to downstream consumers.

The second part of recovery, called *repair*, allows the query to repair its failed pieces and regain its original level of fault-tolerance. In upstream backup, the system regains its normal fault-tolerance level when the new replica fills its output log with enough data to rebuild the states of downstream nodes.

For both active replicas and passive standby, repair can cause significant disruptions in the result stream depending on the granularity of coordination in the query. For example, if a system uses active replica coordination only at the input(s) and output(s) of a distributed query, the system must destroy the *entire* query affected by the failure, stall the *entire* remaining query, checkpoint its state, copy that state onto independent nodes, and reintegrate the new copy with the remaining query. The system must repair a query at a time because it has no control over inflight data in the network between nodes in a query. If the query state is large, e.g., tens of gigabytes, repair can take minutes, causing significant latencies in the result stream. Similarly, coarse coordination in passive standby would cause the first checkpoint after recovery to stall processing for a long time.

To remedy this problem, most high-availability CQ schemes (e.g., Flux [14,13], Borealis DPC [1,2], Active Standby [7], Passive Standby [7,8]) coordinate and repair in smaller chunks: between nodes (containing groups of operators), between operators, or even finer. Then, after failure, they can repair the lost pieces one at time, allowing the remaining pieces to continue processing and reduce the impact of stalls. In the presence of $k + 1 > 2$ replicas, DSMSs can use the extra replicas to further smooth the impact of stalls during repair. Also with finer coordination, DSMSs need to repair only the lost pieces, thereby reducing mean-time-to-recovery and improving system reliability, i.e., mean-time-to-failure [14].

Trade-Offs Among Crash Failure Techniques The above techniques provide different trade-offs between runtime overhead and recovery performance. Active replicas provide quick fail-over because replicas are always "up-to-date". With this approach, however, the runtime overhead is directly proportional to the degree of replication. Passive standby provides a flexible trade-off between runtime overhead and recovery speed through the configurable checkpoint interval. As the checkpoint interval decreases, the runtime computation and network overheads increase because the primaries copy more intermediate changes to the backups. However, recovery speed improves because the backups are in general more up-to-date when they take over. Finally, upstream backup incurs the lowest overhead because backups remain idle in the absence of failures. For upstream backup, recovery time is proportional to the size of the upstream buffers. The size of these buffers, in turn, depends on how much history is necessary to rebuild the state of downstream nodes. Thus, upstream backup is practical in small history settings.

Techniques for Handling Network Partitions

The previous techniques can mask crash failures and a limited set of network failures (by converting disconnected nodes into crashed nodes), but cannot handle network partitions in which data sources, processing nodes, and clients are split into groups that cannot communicate with each other.

In the presence of network partitions, a DSMS, like all distributed systems, has two choices. It can either suspend processing to ensure consistency, or continue processing the remaining streams with best-effort results to provide availability [3]. Existing work on fault-tolerance in distributed DSMSs has explored both options. The Flux protocol [13], originally set in the local-area where network partitions are rare, favors consistency. The Borealis's DPC protocol, designed for wide-area monitoring applications where partitions are more frequent, favors availability. During partitions, DPC generates best-effort result tuples which are labeled as *tentative*. Further, DPC allows applications to specify a maximum tolerable latency for flexibly controlling the tradeoff between consistency and availability [1,2].

Once a network partition heals, a stalled CQ node can simply resume. A node that continued processing with missing input, however, might be in a diverged state, i.e., a state different from that of a failure-free execution. To reconcile a node's diverged state, a DSMS can take two approaches. The system can revert the node to a consistent, checkpointed state and replay the subsequent, complete input, or the system can undo and redo the processing of all tuples since the network partition. To avoid stalling the output during reconciliation, a DSMS must take care not to reconcile all replicas of a node simultaneously. Moreover, nodes must correct their previous output tentative tuples to enable downstream nodes to correct their states, in turn, and to allow applications to ultimately receive the correct and complete output. The Borealis DPC protocol supports these techniques [1,2].

Optimizations

Flux: Integrating Fault Tolerance and Load Balancing A large-scale cluster is a dynamic environment in which DSMSs face not only failures but also load imbalances which reduce availability. In this setting, DSMSs typically split the state of operators into *partitions* and spread them across a cluster for scalability. A single overloaded machine, in this setup, can severely slow down an entire CQ. The Flux operator can be interposed between producer–consumer partitions to coordinate their communication. To absorb short delays and imbalances, Flux allows out-of-order processing of partition input. For long-term imbalances, it supports fine-grained, online partition state movement. The Flux operators interact with a global controller that coordinates repair and rebalancing, and uses the same state movement mechanism for both [13]. Integrating load balancing with fault tolerance allows a system to better utilize available resources as nodes enter and leave the system. These features allow smooth hardware refresh and system growth, and are essential for administering DSMSs in highly dynamic and heterogeneous environments.

Leveraging Replication for Availability and Performance in Wide-Area Networks In previous active replicas approaches, a consumer replica receives the output stream of only one of many producer replicas. In wide area networks, the connection to any single producer is likely to slow down or fail, thereby disrupting the subsequent processing until fail-over completes. To avoid such disruptions, each consumer replica in Hwang et al.'s method [7] merges streams from multiple producer replicas into a single duplicate-free stream. This scheme allows each consumer, at any instant, to use the fastest of its replicated input streams. To further reduce latency, they redesign operators to avoid blocking by processing input out-of-order when possible, while ensuring that applications receive the same results as in the non-replicated, failure-free case. Moreover, their scheme continually adjusts the replication level and placement of operator replicas to optimize global query performance in a dynamically changing environment.

Passive Standby: Distributed Checkpointing and Parallel Recovery The basic passive standby approach has two drawbacks: it introduces extra latencies due to checkpoints and has a slower recovery speed than active replicas. Hwang et al.'s distributed checkpointing technique overcomes both problems [8]. This approach groups nodes into logical clusters and backs up each node using the others in the same cluster. Because different operators on a single node are checkpointed onto separate nodes, they can be recovered in parallel. This approach dynamically assigns the backup node for each operator and schedules checkpoints in a manner that maximizes the recovery speed. To reduce disruptions, this approach checkpoints a few operators at a time. Such checkpoints also begin only at idle times.

Key Applications

There are a number of critical, online monitoring tasks that require 24×7 operation. For example, IT administrators often want to monitor their networks for intrusions. Brokerage firms want to analyze quotes from various exchanges in search for arbitrage opportunities. Phone companies want to process call-records for correct billing. Web site owners want to analyze and monitor click-streams to improve targeted advertising and to identify malicious users. These applications, and more, can benefit from fault-tolerant and highly available CQ systems.

Future Directions

Key open problems in the area of fault-tolerance and high availability in DSMSs include handling Byzantine failures, integrating different fault-tolerance mechanisms, and leveraging persistent storage. Techniques for handling failures of data sources or dirty data produced by data sources (e.g., sensors) are also areas for future work.

Experimental Results

See [1,2,7,8,13,14] for detailed evaluations of the different fault-tolerance algorithms.

URL to Code

Borealis is available at: http://www.cs.brown.edu/research/borealis/public/

Cross-references

- ► Continuous Query
- ► Data Stream
- ► Distributed Data Streams
- ► Stream Processing

Recommended Reading

1. Balazinska M. Fault-Tolerance and Load Management in a Distributed Stream Processing System. Ph.D. thesis, Massachusetts Institute of Technology, 2006.
2. Balazinska M., Balakrishnan H., Madden S., and Stonebraker M. Fault-Tolerance in the Borealis Distributed Stream Processing System. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 13–24.
3. Brewer E.A. Lessons from giant-scale services. IEEE Internet Comput., 5(4):46–55, 2001.
4. Elnozahy E.N.M., Alvisi L., Wang Y.M., and Johnson D.B. A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv., 34(3):375–408, 2002.
5. Gray J. Why do computers stop and what can be done about it? Technical Report 85.7, Tandem Computers, 1985.
6. Gray J., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.

7. Hwang J.H., Balazinska M., Rasin A., Çetintemel U., Stonebraker M., and Zdonik S. High-Availability Algorithms for Distributed Stream Processing. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 779–790.
8. Hwang J.H., Xing Y., Çetintemel U., and Zdonik S. A cooperative, self-configuring high-availability solution for stream processing. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 176–185.
9. Kawell L., Beckhardt S., Halvorsen T., Ozzie R., and Greif I. Replicated Document Management in a Group Communication System. In Proc. ACM Conf. on Computer-Supported Cooperative Work, 1988.
10. Schiper A. and Toueg S. From set membership to group membership: a separation of concerns. IEEE Trans. Dependable Secure Comput., 3(1):2–12, 2006.
11. Schneider F.B. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv., 22(4): 299–319, 1990.
12. Schneider F.B. What good are models and what models are good? In Distributed Systems. ACM/Addison-Wesley Publishing Co, 2nd edn., 1993, pp. 17–26.
13. Shah M.A. Flux: A Mechanism for Building Robust, Scalable Dataflows. Ph.D. thesis, University of California, Berkeley, 2004.
14. Shah M., Hellerstein J., and Brewer E. Highly-Available, Fault-Tolerant, Parallel Dataflows. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 827–838.
15. Terry D.B., Theimer M., Petersen K., Demers A.J., Spreitzer M., and Hauser C. Managing update conflicts in Bayou, a weakly connected replicated storage system. In Proc. 15th ACM Symp. on Operating System Principles, 1995, pp. 172–183.

FCP

▶ Storage Protocols

FD

▶ Functional Dependency

Feature Extraction for Content-Based Image Retrieval

RAIMONDO SCHETTINI[1], GIANLUIGI CIOCCA[1], ISABELLA GAGLIARDI[2]
[1]University of Milano-Bicocca, Milan, Italy
[2]National Research Council (CNR), Milan, Italy

Synonyms

Image indexing

Definition

Feature extraction for content-based image retrieval is the process of automatically computing a compact representation (numerical or alphanumerical) of some attribute of digital images, to be used to derive information about the image contents. It can be seen as a case of dimensionality reduction. A feature, or attribute, can be related to a visual characteristic, but it may also be related to an interpretative response to an image or to a spatial, symbolic, semantic, or emotional characteristic. A feature may relate to a single attribute or be a composite representation of different attributes. Features can be classified as general purpose or domain-dependent. The general purpose features can be used in any context, while the domain-dependent features are designed specifically for a given application. Every feature is intimately tied with the kind of information that it captures. The choice of a particular feature over another depends on the given application, and the kind (level) of information required.

Historical Background

The interest of the scientific community in the problem of image retrieval can be dated back to the early 1990s. One of the pioneering examples on image feature extraction is the work by Swain and Ballard [6] where the concept of color histogram is introduced for image indexing. Since then, color (in many forms and evolutions) has been one of the most widely used features exploited for indexing the image contents. This is mainly due to its power to visually capture perceptual properties, robustness with respect to many image transformations, and computation efficiency. While more complex and sophisticated general purpose and domain dependent features were being developed, two important issues became evident: the sensory gap and the semantic gap.

The sensory gap is the gap between the information of the real world, and the information in a computational description derived from a digital recording of a scene of the world. The semantic gap is the lack of coincidence between the information that one can extract from the visual data and the interpretation that the same data have for a user in a given application [5].

The problems of the sensory and semantic gaps are less relevant for very specific applications (e.g., medical imaging) where the domain may be so narrow that high knowledge can be embedded in few features. However, for general purpose applications, sensory and semantic gaps cannot be bridged by a few features alone. To cope with the sensory gap problem, current approaches try to combine different features (mostly at low or middle levels of information), and different representations of the same feature. One way to cope with the semantic gap problem is not to tackle it at the feature level but at the system-user interaction level with human-adaptive paradigms. For these paradigms, the user represents a central role for a content-based image retrieval system. User interaction allows the system to automatically adapt to the changing requests; for example, in the way the images are organized, evaluated, or retrieved [7].

Foundations

Since an image conveys information at different levels, the use of different features at the same time is a necessary requisite to develop effective indexing algorithms for content-based image retrieval. For example, any visual property can have different feature representations describing it. Visual properties can be low level properties such as color, shape or texture; middle level properties such as regions and spatial relationships; high level properties such as the localization and identification of objects or image categorization. The process of choosing the best features for a particular application is denoted as feature selection.

The selected features should ideally present the following basic properties:

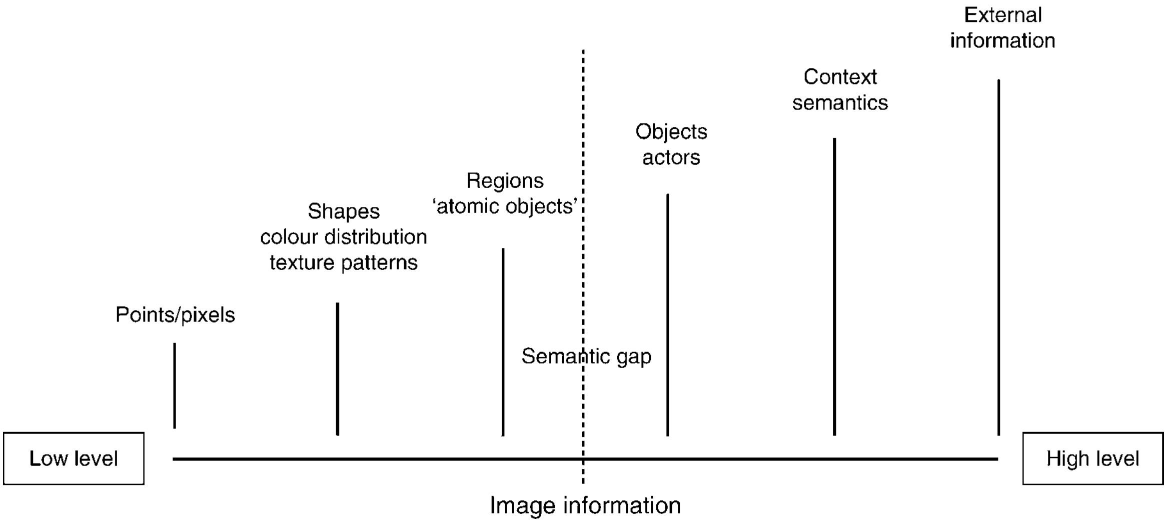
1. Perceptual similarity: the feature distance between two images is great only if the images are not “similar”;
2. Efficiency: they can be rapidly computed;
3. Economy: their dimensions are small in order not to affect retrieval efficiency;
4. Sscalability: the performance of the system is not influenced by the size of the database;
5. Rrobustness: changes in the imaging conditions (i.e., illuminations, geometric transformations,. . .) of the database images should not affect retrieval.

Once the features have been selected, a similarity measure (one for each feature and/or a global one) is chosen in order to evaluate the user query against the images in the database, and rank the retrieved images. Ideally a similarity measure should be correlated with the user’s perception of image matching or *image similarity.*

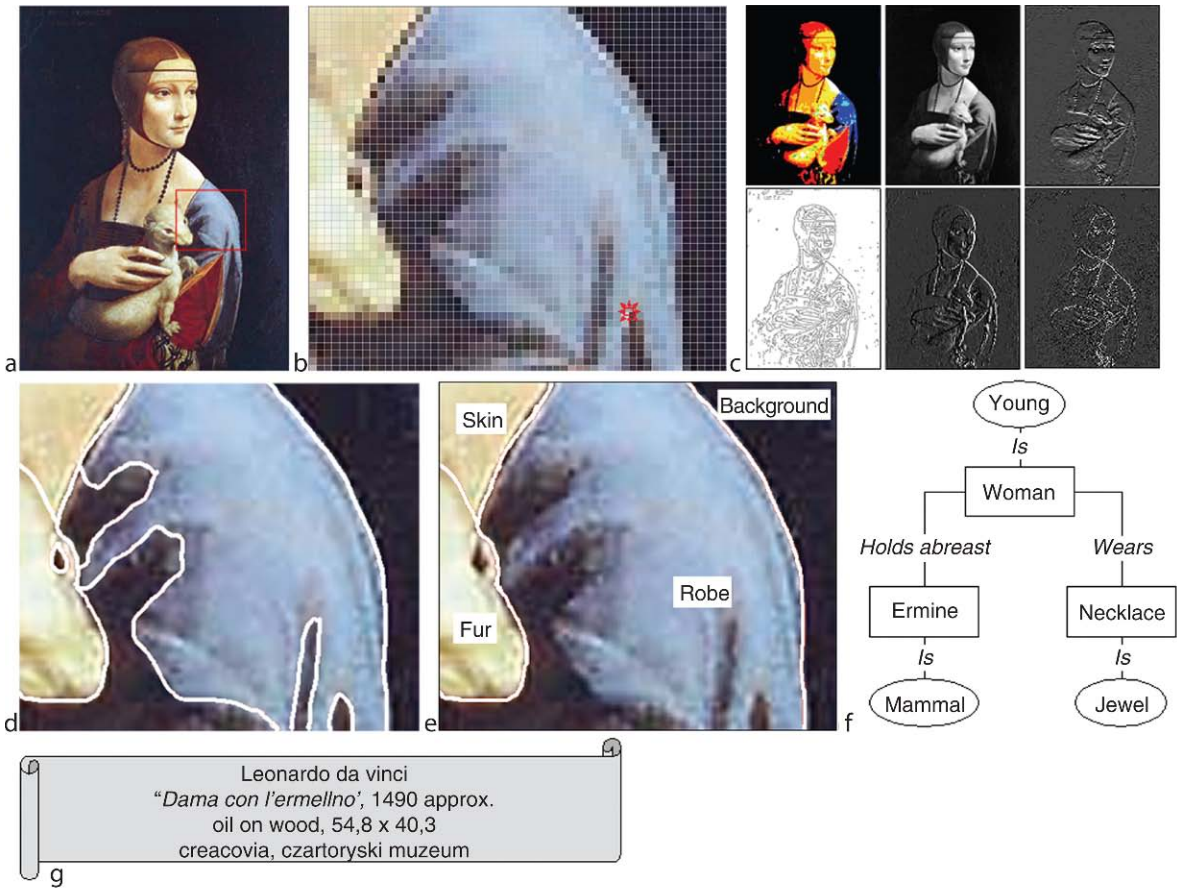
Figure 1 shows the different levels of information that can be derived from an image: the dashed line indicates where the semantic gap starts to influence the feature extraction algorithms.

Figure 2 is an example of the different information that can be obtained from an image. At each level of information more complex features are required to capture that kind of information. Details of each level are as follows:

1. Digital image of a painting.
2. Pixels. The image is coded using a numerical representation of physical properties derived from the input device with implicit spatial relationships. This representation may capture color as well as pressure, range, heath, non visible wavelengths and



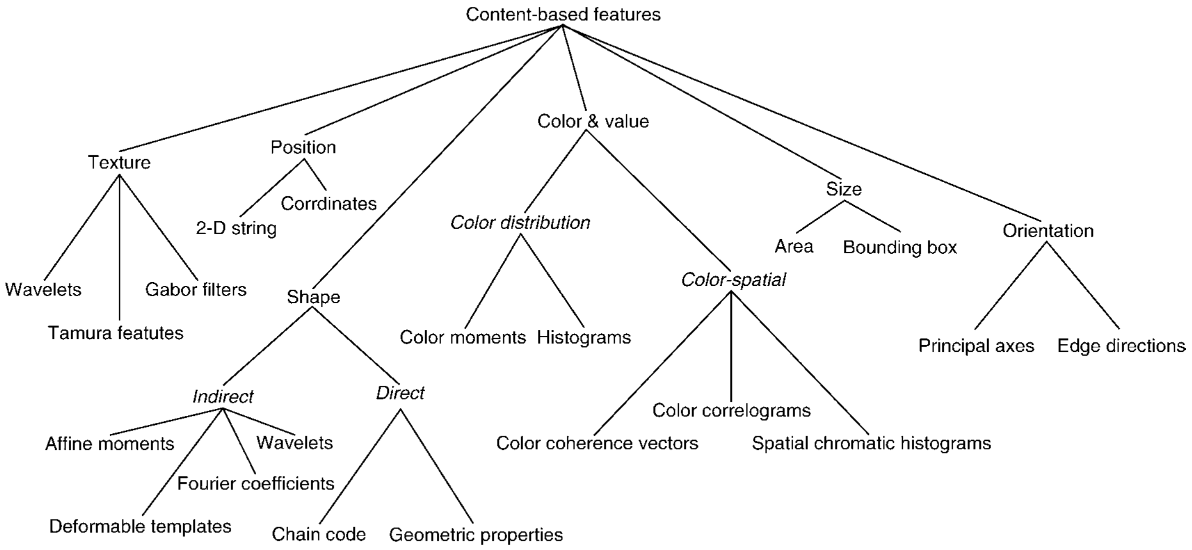
Feature Extraction for Content-Based Image Retrieval. Figure 1. The levels of information, and the semantic gap.



Feature Extraction for Content-Based Image Retrieval. Figure 2. Example of features corresponding to different levels of information. (a) the example image; (b) pixel level; (c) global low level features; (d) local low level features; (e) semantic regions; (f) semantic context; (g) external information.

so on. Each of these pieces of information physical information can be described using different numerical representations.

3. Low level information. These are global features that can describe the image contents from different perspectives in a compact way. The examples shown are: color contents in a quantized color space (top left), global edges computed with the Canny edge detector (bottom left), the gray level image (top middle), and horizontal, vertical, and diagonal edges computed from a wavelet transformation (to right, bottom middle, and bottom right respectively). This information is usually coded in terms of simple statistics (mean, variance,...), histograms, or signatures to improve their robustness to image variations, and reduce their dimensions for retrieval efficiency. The schema below shows an incomplete and non-exhaustive categorization of some low level visual properties, and their possible feature representations. Many representations and sub-representations are available for every feature (see Fig. 3). For further information interested readers may refer to [1,2,3,4].
4. Regions or "atomic objects." These are visually homogeneous regions that can be identified starting from the combination of different low level attributes (color, texture,...). *Data clustering* algorithms can be exploited for this task. Domain-dependent knowledge (for example specific spatial relationships between the areas) can be exploited to drive or refine the identification of these regions.
5. Objects/Actors. Identification of semantic objects or categories may be performed by the analysis of logically and spatially related regions. At this level features should cope with the problem of the semantic gap. The recognition of simple objects or



Feature Extraction for Content-Based Image Retrieval. Figure 3. Example of features, their corresponding representations and sub-representations.

categories is very complex, and is currently fully possible only for very narrow domains. Identification of semantic objects can be done exploiting pattern matching, *classification and decision tree* algorithms. These algorithms give a semantic tag or name to each candidate region (such as face, people, skin, sky, car, building, water, . . .). Generally one tag for each region is assigned, although multiple tags are sometimes allowed. This process is generally known as image annotation. Semantic tags can be expressed numerically or with more meaningful textual strings. Textual strings can be stored in traditional database systems to be used for image retrieval based on keywords.

6. Context semantics. This information relates to mutual spatial relationships as well as to specific actors/objects characteristics that may allow reconstruction/guessing of the overall image context. Some characteristics of the actors/objects may be derived using term taxonomies, synonyms, and dictionaries. Spatial relationships can also be described using textual strings by creating an ad-hoc language grammar such as A < B meaning that object A is at the left of object B or A ^ B meaning that object A is above object B, and so on. It should be noted that the classification and categorization of an image as a whole may involve information at the objects/actors level (the image itself is an object that can be annotated using low level features), and information at the semantic level (the presence of objects/actors, and some related characteristics can be used as clues in guessing the context of the image).
7. External information. This information does not describe some visual properties or intrinsic properties of the image. This auxiliary information cannot be derived or inferred directly from the image itself: external knowledge belonging to some entities (e.g., experts, common knowledge, etc. . .) is required. No automatic algorithm can be exploited with this kind of information: human intervention is the only viable approach in creating textual annotations usually given in free text form. The EXIF data also belong to this information category. These data, automatically embedded into the image by the hardware (e.g., camera), describe information about the device, device settings and scene information.

Key Applications

Feature extraction is a fundamental task for any application requiring the analysis of images. In particular, content-based image retrieval is important in many areas. The following is a sampling of possible content-based image retrieval applications:

1. Architecture and design (finding the right appearance)
2. Biochemical application (finding molecules)

3. Cultural heritage services (museum, art galleries)
4. Digital catalogs (browsing and searching)
5. Entertainment (image, film and video archive indexing)
6. Journalism (research and past exploration)
7. Medicine (pathology comparisons)
8. Remote sensing (localization and target finding)
9. Surveillance (identification and recognition)

Cross-references

- ▶ Automatic Image Annotation
- ▶ Decision Tree Classification
- ▶ Image Content Modeling
- ▶ Image Metadata
- ▶ Image Representation
- ▶ Indexing and Similarity Search

Recommended Reading

1. Antani S., Kasturi R., and Jain R. Survey on the use of pattern recognition methods for abstraction, indexing and retrieval of images and video. Pattern recognit., 35:945–965, 2002.
2. Eakins J.P. Towards intelligent image retrieval. Pattern Recognit., 35:3–14, 2002.
3. Schettini R., Ciocca G., and Zuffi S. Indexing and retrieval in color image databases. In Color Imaging Science: Exploiting Digital Media, R. Luo, L. MacDonald (eds.). Wiley, New york, 2002, pp. 183–211.
4. Sikora T. The MPEG-7 visual standard for content description – An overview. IEEE Trans. Circuits Syst. Video Technol., 11 (6):696–702, 2001.
5. Smeulders A.W.M., Worring M., Santini S., Gupta A., and Jain R. Content-based image retrieval at the end of the early years. IEEE Trans. Pattern Anal. Mach. Intell., 2(2):1349–1380, 2000.
6. Swain M.J. and Ballard D.H. Indexing via color histograms. In Proc. 3rd IEEE Conf. Computer Vision, 1990, pp. 390–393.
7. Zhou X.S. and Huang T.S. Relevance feedback in image retrieval: a comprehensive review. Multimed. Syst., 8(6):536–544, 2003.

Feature Selection for Clustering

Manoranjan Dash, Poon Wei Koot
Nanyang Technological University, Singapore, Singapore

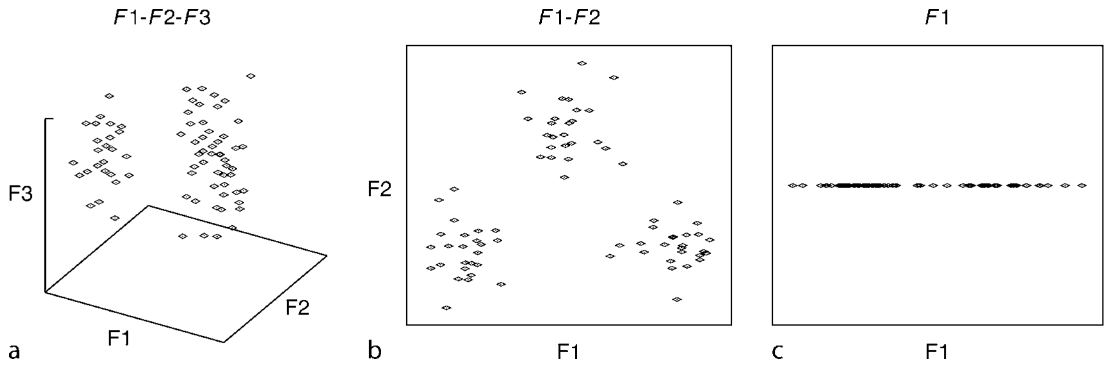
Definition

The problem of feature selection originates from the fact that while collecting data, one tends to collect all possible data. But for a specific learning task such as clustering not all the attributes or features are important. Feature selection is popular in supervised learning or for the classification task because the class labels are given and it is easier to select those features that lead to these classes. But for unsupervised data without class labels, or for the clustering task, it is not so obvious which features are to be selected. Some of the features may be redundant, some are irrelevant, and others may be "weakly relevant". The task of feature selection for clustering is to select "best" set of relevant features that helps to uncover the natural clusters from data according to the chosen criterion.

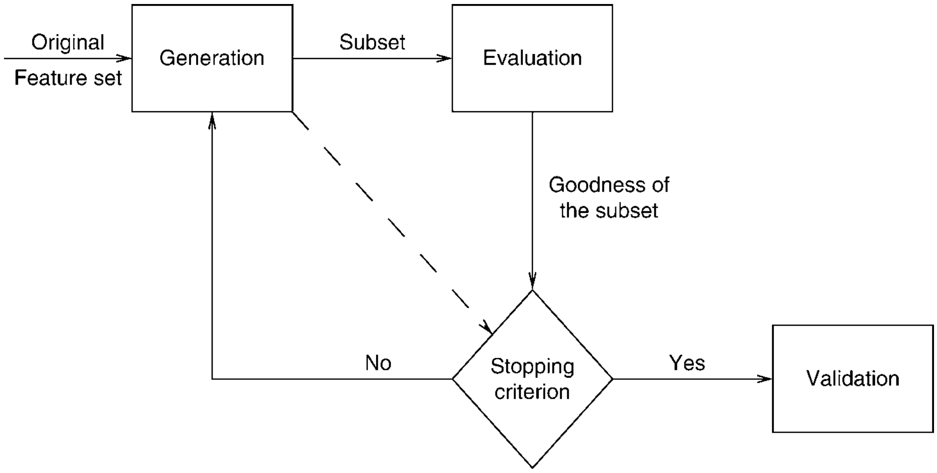
Figure 1 shows an example using a synthetic data. There are three clusters in *F*1–*F*2 dimensions which follow Gaussian distribution whereas *F*3, which does not define any cluster, follows a uniform distribution. When all three features are used the clusters are unnecessarily complex (see Fig. 1a), whereas no clusters can be found when one visualizes using only one feature *F*1 (Fig. 1c). Figure 1b with *F*1–*F*2 features shows three well-formed clusters. Selecting features *F*1 and *F*2 reduces the dimensionality of the data while forming well separated clusters. The goal of feature selection for clustering is to select important original features for clustering thus reducing the data size (and the computation time of other subsequent data mining tasks), solving the problem of curse of high-dimensionality and improving the knowledge discovery performance and comprehensibility. There are four basic steps in a typical feature selection method: a method to generate candidate subsets, an evaluation function, a stopping criterion, and a validation procedure (Fig. 2).

Historical Background

The literature for feature selection for classification, which is a supervised learning task, is very vast (see a review in [6]). On the other hand, there are only a few, mostly recent, feature selection methods for clustering, which is a unsupervised task. The reason behind this gap in research is it is easier to validate the selected features for classification (e.g., by accuracy of the classification) than to validate the selected features for clustering. Although there are many cluster validation techniques, the techniques have their limitations, and thus there is lack of unanimity among the researchers [19]. An issue that is yet to be resolved is how many clusters are there in a data set. Particularly in real-world



Feature Selection for Clustering. Figure 1. Effect of features on clustering.



Feature Selection for Clustering. Figure 2. Feature selection process with validation.

data sets often one encounters high-dimensional data sets with unknown number of cluster. It is shown that number of clusters and number of features have interrelation thus complicating the matter further [10].

Feature selection for clustering is the task of selecting important features for the underlying clusters. These methods can be divided using different categorization such as: global vs. local and wrapper (i.e., with feedback) vs. filter (i.e., without feedback – blind). Global methods select features for the whole data set whereas local methods select features for each individual cluster.

Examples of global methods are [5,12,17,22]. In [5] was proposed a filter method. A filter method is independent from the clustering algorithm to be used, whereas wrapper method uses the clustering method itself to select features. In this paper, the authors we proposed to measure entropy of the data set using point-to-point distances. Entropy is independent of number of features. Lower the entropy better the clustering. Features were ranked from most important to least important and a forward selection algorithm is used to select the relevant features. In [12] Dy and Brodley proposed a wrapper criterion for clustering. A candidate subset of features is used to cluster the data and the quality of clusters is evaluated using normalized cluster separability (for *K*-means) or normalized likelihood (for EM – expectation maximization – clustering). The bias on the feature subsets with respect to dimensionality is ameliorated by cross-projection normalization. In [11] the same authors used EM and trace measure (which is invariant for varying number of dimensions [9]) are used for evaluation. Visual aids to decide the optimal number of features were also proposed. The method described in [17] uses

K-means for evaluation of subsets of features. In [22] authors proposed an objective function for choosing the feature subset and finding the optimal number of clusters for a document clustering problem using a Bayesian statistical estimation framework.

Examples of local methods are [2,14,18,21]. In [14] authors proposed a distance measure. Using this measure with usual distance based clustering algorithms encourages the detection of subgroups of objects that preferentially cluster on subsets of features. The relevant feature subsets for each individual cluster can be different and partially or completely overlap with those of other clusters. In [18], feature saliency is integrated in EM algorithm so that feature selection is performed simultaneously with clustering process. Projected clustering (ProClus [1]) finds subsets of features defining (or important for) each cluster. ProClus first finds clusters using K-medoid considering all features and then finds the most important features for each cluster using Manhattan distance. The algorithm called CLIQUE in [2] divides each dimension into a user given divisions. It starts with finding dense regions (or clusters) in one-dimensional data and works upward to find higher-dimensional dense regions using candidate generation algorithm Apriori. In [20,21] Talavera used category utility to select features and these features are used to construct COBWEB [13] (COBWEB is a hierarchical clustering algorithm for categorical data.).

Foundations

This section first gives some insight into feature selection for clustering using an example method and then briefly discusses other methods and techniques used.

An Example of a Filter-Global Method

Figure 3 shows the histogram of point-to-point distances for two datasets: one with clusters and the other without clusters. The point-to-point distances are computed, normalized and are used to populate the bins of the histograms. An important distinction between the two histograms is that histogram for data without clusters has a predictable shape similar to bell shape. But the histogram for data with clusters has very different distribution. The smaller buckets (or distances) are mostly intra-cluster while the latter ones are mostly inter-cluster. *Typically, if the dataset consists of some clusters then, the majority of the intra-cluster distances will be smaller than the majority of the inter-cluster distances.* This observation is true for a wide range of data types. When clusters are very distinct this separation between intra-cluster and inter-cluster distances is quite distinguishable. In [5] a method is proposed which, without doing clustering, can distinguish between data with clusters and data without clusters.

Distance-based Entropy Measure From entropy theory it is known that entropy of a system can measure the amount of disorder in the system. Mathematically entropy of a dataset is given as:

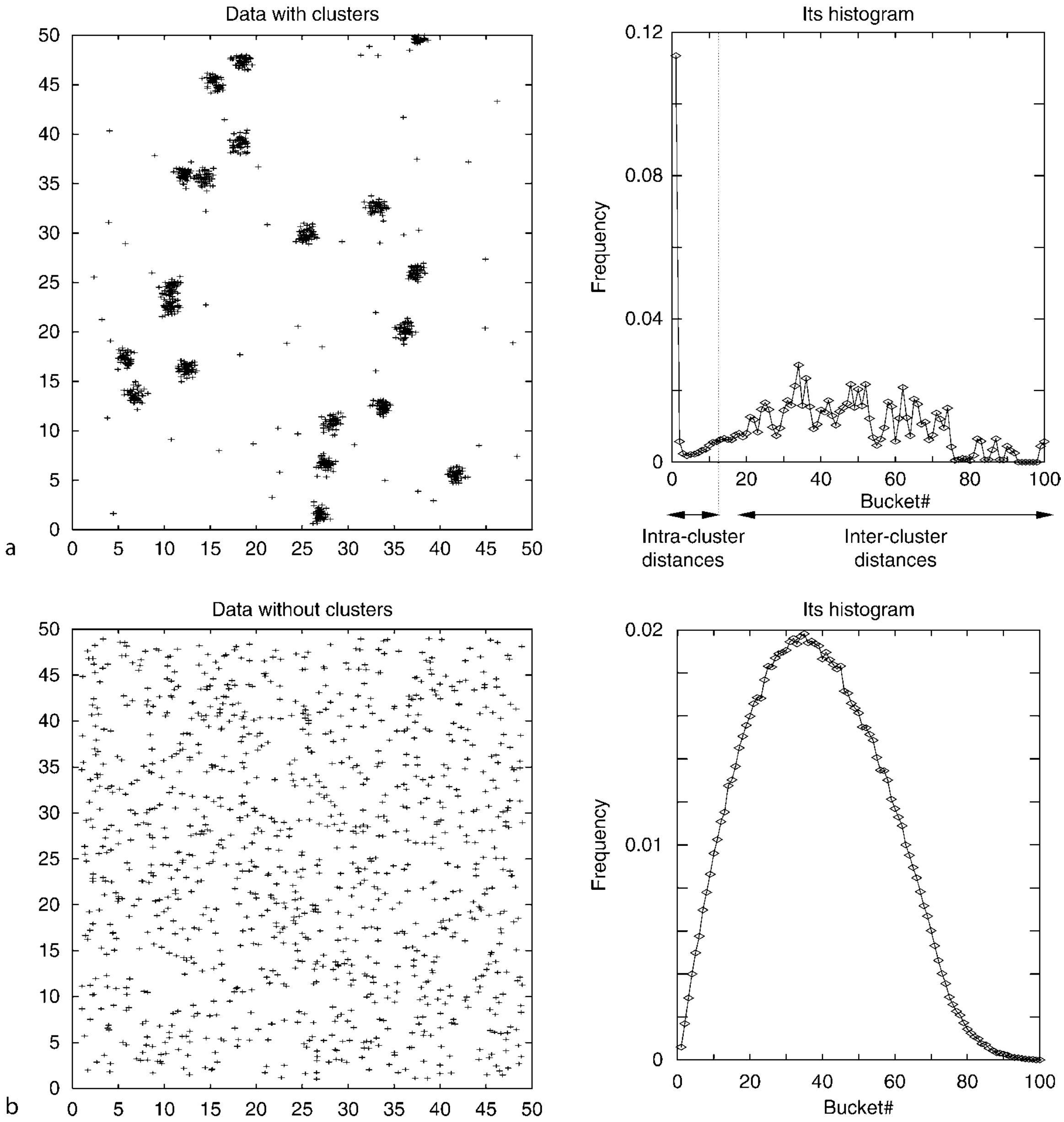
$$E = -\sum_{X_i} p(x_{i_1},\ldots,x_{i_M}) \log p(x_{i_1},\ldots,x_{i_M}) + (1 - p(x_{i_1},\ldots,x_{i_M})) \log(1 - p(x_{i_1},\ldots,x_{i_M})) \quad (1)$$

where $p(X_{i_1},\ldots,X_{i_M})$ is the probability or density at the point $(X_{i_1},\ldots,X_{i_M})$ where M is the dimensionality. The second term in the expression inside summation is used to make the expression symmetric. If the probability of each point is equal one is most uncertain about the outcome, and entropy is the maximum. This will happen when the data points are uniformly distributed in the feature space. On the other hand, when the data has well-formed clusters the uncertainty is low and so also the entropy. So, the entropy can be used to distinguish between data with clusters and data without clusters in different subspaces.

A straightforward method to compute the probability at each point is by substituting probability with distance in the following way:

$$E = -\sum_{X_i}\sum_{X_j} D_{ij} \log D_{ij} + \left(1 - D_{ij}\right) \log\left(1 - D_{ij}\right) \quad (2)$$

where D_{ij} is the normalized distance (This uses Euclidean (L_2) measure although other distances such as Manhattan (L_1) can be used.) in the range [0.0–1.0] between instances X_i and X_j. Figure 4a shows the relationship between entropy and distance after normalizing E to the range [0.0–1.0]. This measure assigns lowest entropy (0.0) for the minimum (0.0) or the maximum (1.0) distance and assigns highest entropy (1.0) for the mean distance (0.5). Although, to some extent, it works well in distinguishing data with clusters from data without clusters, it suffers from the following two drawbacks. (i) The mean distance of 0.5 can be an inter-cluster distance, but still it assigns the highest entropy. The reason is the meeting point



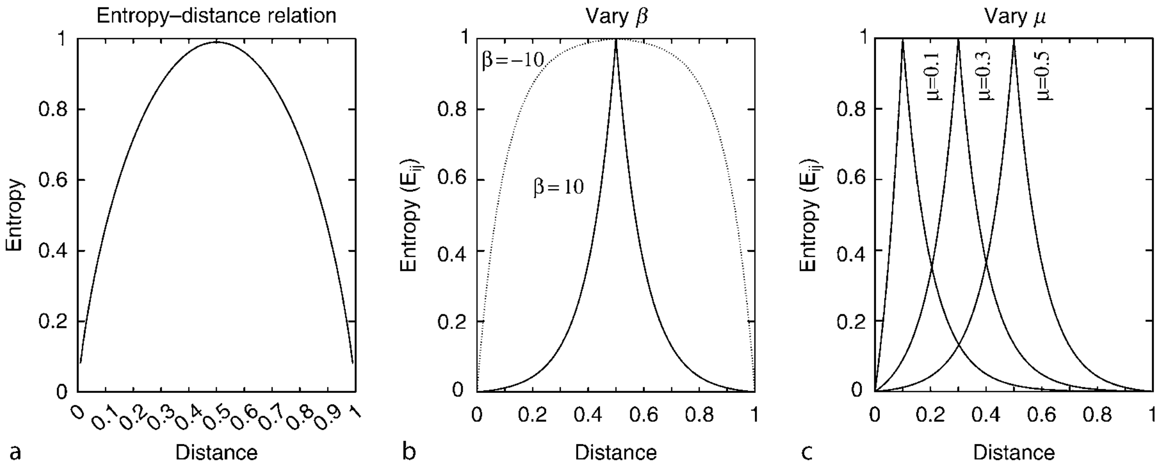
Feature Selection for Clustering. Figure 3. Distance histograms of data *with* and *without* clusters.

(μ) of the two sides (i.e., left and right) of the plot is fixed at distance 0.5. (ii) Entropy increases rapidly for very small distances thus assigning very different entropy values for intra-cluster distances. In summary, this measure does not work very well in assigning small entropy for both intra-cluster and inter-cluster distances. The second drawback can be easily overcome by incorporating a coefficient (β) in the equation and by using an exponential function in place of logarithmic function. Regarding the first drawback, the meeting point (μ) can be set so as to separate the intra-cluster and inter-cluster distances. Considering all these the following method was proposed:

$$E = \sum_{x_i} \sum_{x_j} E_{ij} \qquad (3)$$

$$E_{ij} = \begin{cases} \frac{exp\left(\beta * D_{ij}\right) - exp(0)}{exp(\beta * \mu) - exp(0)} & : \quad 0 \leq D_{ij} \leq \mu \\ \frac{\exp(\beta * (1.0 - D_{ij} - \exp(0)}{\exp(\beta * (1.0 - \mu)) - \exp(0)} & : \quad \mu \leq D_{ij} \leq 1.0 \end{cases} \qquad (4)$$

where E_{ij} is normalized to the range [0.01-1.0].



Feature Selection for Clustering. Figure 4. Relationship between entropy and distance with varying β and μ values.

Other Methods

Trace Measure In the wrapper and global methods feature subsets are evaluated by a clustering algorithm and the quality of clustering is evaluated using trace measure [12,7]. The Trace measure $(tr(S_W^{-1}S_B))$ is based on within and between cluster distances, where *tr* is *trace* os a matrix which is the sum of its diagonal elements; S_W is a within-cluster scatter matrix given as: $S_W = \sum_{i=1}^{c}\sum_{X\in\chi_i}(\mathbf{x}-\mathbf{m_i})(\mathbf{x}-\mathbf{m_i})^t$, and S_B is a bet-cluster scatter matrix given as: $S_B = \sum_{i=1}^{c}$ $(\mathbf{m_i} - \mathbf{m})(\mathbf{m_i} - \mathbf{m})^t$ where $\mathbf{x}$ is an instance vector, χ_i is the set of instances in *i*th cluster, $\mathbf{m_i}$ is the mean vector of *i*th cluster, $\mathbf{m}$ is the mean vector of the data. The higher the $tr(S_W{}^{-1}S_B)$, the higher the ratio of between-cluster to within-cluster scatter. If clustering is proper, then between cluster distance will be high and within cluster distance is low. Although there are many cluster validation techniques (see [16]), trace measure is especially selected because it is independent of number of features, i.e., irrespective of number of features in the subspace, trace measure can evaluate the clustering quality and compare among clusterings in different subspaces of equal or different dimensionality.

EM Method In [18], Law et al. cast the feature selection for clustering as an estimation problem rather than a search problem thus they avoided the combinatorial search through the feature space. Instead of selecting a subset of features they estimated a set of real-valued ([0,1]) quantities for each feature. They call it *feature saliency*. This estimation is carried out by an EM (Expectation Maximization) algorithm derived for the task. They avoided the situation where all the saliencies take the maximum possible value by adopting a minimum message length (MML) penalty. The MML criterion encourages the saliencies of the irrelevant features to go to zero. They combined the feature selection with EM clustering.

Conceptual Clustering Conceptual clustering is applicable for feature selection for clustering for data having only categorical features (e.g., COBWEB [13]). It creates a hierarchy or tree of clusters. As each data object is input to the system, the system categorizes the object by sorting it through hierarchy from the root node down to the leaves. At each level it determines whether (i) to create a new cluster, or (ii) to place the object in an existing cluster, and whether to restructure the hierarchy by (iii) merging two sibling clusters which were identified as the two best hosts for the new object or (iv) splitting the host cluster.

In [20,21] Talavera proposed two local methods for selecting features for conceptual clustering. In [21] Talavera used salience measure (this is different from the saliency measure in [18]) to select features. The higher the salience the greater is its relevance for the clustering. Salience measure originates from category utility measure.

Devaney and Ram [8] also used COBWEB to select features for clustering but they used it in a different way. The basic idea is to employ a wrapper approach with the average predictive accuracy over all features replacing the predictive accuracy of class labels. They used COBWEB as an evaluation function. The search through the feature space is either forward or backward. COBWEB is executed for each of these subsets and category utility is computed of the first partition (children of the root) of the resulting concept hierarchy, retaining the highest score.

Important Applications

Feature selection has gained a wider audience in the past few years due to the high-dimensionality of databases. Due to the high number of low level features, a lot of the current methods of clustering in low dimensionality will fail miserably. Thus the only way is to identify the most important features and reduce dimensionality. In terms of applications, there are numerous possibilities.

An important application of feature selection is in the area of bioinformatics, using gene expression microarray data. Due to the nature of high dimensionality (thousands of genes) and sparsity of the data in feature space, clustering is exceptionally difficult. In [23] this issue is addressed using CLIFF, an algorithm based on normalized cut with their feature selection process. In [24] Malik et al. paper, he proposed another method for gene expression using recursive cluster elimination (RCE) with SVM, and claimed to have improved accuracy compared to other methods. In [14] an application of local feature selection for clustering is shown over Yeast gene expression data set with 6,141 genes (features) measured on 213 samples (instances).

Bekkerman et al. [4] applied feature selection with SVM to the problem of text categorization to yield high performance accuracy. Datasets used are: 20-Newgroups, Reuters-21578 and WebKB. The 20-Newsgroups contains 19,997 articles from the Usenet newsgroup collection, while Reuters-21578 corpus contains 21,578 articles from the Reuters newswire, and WebKB is a collection 8,282 web pages from four academic domain.

Bach et al. [15] applied feature selection to the reduction of attribute in image face recognition for male/female classification. Datasets consist of 1,450 images (1,000 train, and 450 for test) with 5,100 features.

In [3], feature selection is applied to PrimeClub, a game designed to teach prime numbers to sixth and seventh grade students. The objective is to keep the students continue learning, maintaining a high level of engagement as the game progresses. Biometric devices are attached to each student to detect the mental and emotional state of their expressions. In it, 28 features are recorded, and of those, only 4 are deemed important for clustering.

While not going into details, some of the other applications of feature selection are: customer relationship management, image retrieval, text mining, protein classification and intrusion detection.

URL

While not many author published their programs or code in the internet, there exist a few that do make theirs available for others to download and compare. "RCE classification and feature selection" as described in [24] is available for download at "http://showelab.wistar.upenn.edu/"

Mark Hall wrote a feature selection program for Weka, described in his PhD dissertation "Correlation-based Feature Subset Selection for Machine Learning."

CLOP is a Matlab package developed on top of the Spider for the WCCI 2006 performance prediction challenge, and it has produced some very good result in the NIPS 2003 feature selection challenge.

RapidMiner, a freely available open source for data mining and machine learning, has functionality for feature selection. It is written in Java and works on all major operating systems, available for download at "http://rapid-i.com/content/blogcategory/38/69/"

INTERACT is a feature selection method using inconsistency and symmetrical uncertainty measurements for finding interacting features. It is describe in "http://www.public.asu.edu/∼huanliu/INTERACT/INTERACTsoftware.html," and available for download from the site.

Cross-references

▶ Cluster and Distance Measure
▶ Clustering Overview and Applications
▶ Curse of Dimensionality
▶ Data Cleaning
▶ Dimensionality Reduction
▶ Dimensionality Reduction Techniques for Clustering

Recommended Reading

1. Aggarwal C.C., Procopiuc C., Wolf J.L., Yu P.S., and Park J.S. Fast algorithms for projected clustering. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 61–72.
2. Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 94–105.
3. Amershi S., Conati C., and Maclaren H. Using feature selection and unsupervised clustering to identify affective expressions in educational games. In Proc. Workshop on Motivational and Affective Issues in ITS, 8th Int. Conf. on ITS, 2006, pp. 21–28.

4. Bekkerman R., El-Yaniv R., Tishby N., and Winter Y. Distributional Word Clusters vs Words for Text Categorization. J. Machine Learning Res. 3:1183–1208, 2008.
5. Dash M., Choi K., Scheuermann P., and Liu H. Feature selection for clustering – A filter solution. In Proc. 2002 IEEE Int. Conf. on Data Mining, 2002, pp. 115–122.
6. Dash M. and Liu H. Feature selection for classification. Int. J. Intell. Data Analy., 1(3):131–156, 1997.
7. Dash M. and Liu H. Handling large unsupervised data via dimensionality reduction. In Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 1999.
8. Devaney M. and Ram A. Efficient feature selection in conceptual clustering. In Proc. 14th Int. Conf. on Machine Learning, 1997, pp. 92–97.
9. Duda R.O. and Hart P.E. Pattern Classification and Scene Analysis, chap. Unsupervised learning and clustering. Wiley, New York, 1973.
10. Dy J.G. and Brodley C.E. Feature subset selection and order identification for unsupervised learning. In Proc. 17th Int. Conf. on Machine Learning, 2000, pp. 247–254.
11. Dy J.G. and Brodley C.E. Visualization and interactive feature selection for unsupervised data. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 360–364.
12. Dy J.G. and Brodley E. Feature Selection for Unsupervised Learning. J. of Machine Learning Res., 5:845–889, 2004.
13. Fisher D.H. Knowledge acquisition via incremental conceptual clustering. Mach. Learn., 2:139–172, 1987.
14. Friedman J. and Meulman J. Clustering objects on subsets of attributes. J. Royal Stat. Soc. B, 66(4):1–25, 2004.
15. Gilad-Bachrach R., Navot A., and Tishby N. Margin based feature selection – theory and algorithms. In Proc. 21st Int. Conf. on Machine Learning, 2004, pp. 43.
16. Jain A.K. and Dubes R.C. Algorithm for Clustering Data, chap. Clustering Methods and Algorithms. Prentice-Hall Advanced Reference Series, 1988.
17. Kim Y.S., Street W.N., and Menczer F. Feature selection in unsupervised learning via evolutionary search. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 365–369.
18. Law M.H.C., Figueiredo M.A.T., and Jain A.K. Simultaneous Feature Selection and Clustering Using Mixture Models. IEEE Trans. Pattern Analy. Mach. Intell., 26(9):1154–1166, 2004.
19. Milligan G.W. A monte carlo study of thirty internal criterion measures for cluster analysis. Psychometrika, 46(2):187–198, 1981.
20. Talavera L. Feature selection as a preprocessing step for hierarchical clustering. In Proc. 16th Int. Conf. on Machine Learning, 1999, pp 389–397.
21. Talavera L. Feature selection and incremental learning of probabilistic concept hierarchies. In Proc. 17th Int. Conf. on Machine Learning, 2000, pp 951–958.
22. Vaithyanathan S. and Dom B. Model selection in unsupervised learning with applications to document clustering. In Proc. 16th Int. Conf. on Machine Learning, 1999, pp. 433–443.
23. Xing E.P. and Karp R.M. CLIFF: clustering of high-dimensional microarray data via iterative feature filtering using normalized cuts. In Proc. 9th Int. Conf. on Intelligent Systems for Molecular Biology, 2001, pp. 306–315.
24. Yousef M., Jung S., Showe L.C., and Showe M.K. Recursive Cluster Elimination (RCE) for classification and feature selection from gene expression data. BMC Bioinformatics, 8:144, 2009.

Feature-Based 3D Object Retrieval

Benjamin Bustos[1], Tobias Schreck[2]
[1]Department of Computer Science, University of Chile, Santiago, Chile
[2]Darmstadt University of Technology, Darmstadt, Germany

Synonyms

Three-Dimensional similarity search; Shape descriptors

Definition

3D objects are an important type of data with many applications in domains such as engineering and computer aided design, science, simulation, visualization, cultural heritage, and entertainment. Technological progress in acquisition, modeling, processing, and dissemination of 3D geometry leads to the accumulation of large repositories of 3D objects. Consequently, there is a strong need to research and develop technology to support the effective retrieval of 3D object data from 3D repositories.

The feature-based approach is a prominent technique to implement content-based retrieval functionality for 3D object databases. It relies on extracting characteristic numerical attributes (so-called features) from a 3D object, usually forming high-dimensional vectors which represent the 3D object, or parts of it. The 3D feature vectors in turn are used to estimate object similarity for content-based retrieval, and can also be used for multidimensional indexing of 3D database content. There exist several degrees of freedom in obtaining 3D features. Important specifications to be made include the type of 3D characteristics or its level of detail considered, or invariance properties required, among others. Finding efficient and effective features for a given 3D repository is usually addressed by benchmarking.

Historical Background

The development of 3D object retrieval methods can be regarded as part of the larger multimedia retrieval research area. The availability of increasing volumes of multimedia data such as digital images, digital video, or digital audio induced the need to develop content-based retrieval methods supporting these data types. Image retrieval has roots in image processing and database research of the 1980s, and was joined in the 1990s by similar efforts in the video and audio domains. Roughly, beginning by 2000, 3D objects increasingly came into focus of multimedia retrieval research. Driving motivation in the research and development of 3D retrieval methods are the increasing use of 3D object data in a range of application areas.

While retrieval of 3D objects is a prominent topic in multimedia database research, the definition of similarity notions for 3D data is also considered in related disciplines. In geometry processing, the registration or alignment of geometry is of interest, using certain definitions of geometric similarity. Computer vision is concerned with the recognition of objects in images taken by a camera or other scanner device, requiring appropriate segmentation and description methods for the objects in the scene under concern. In shape analysis, shapes and scenes are often analyzed for certain structural features, supporting e.g., classification and compression.

From the multimedia database research perspective, the focus of 3D retrieval implementations not only concerns the effectiveness of the retrieval, but also, their efficiency, demanding for real-time query processing on large repositories. The feature vector approach is therefore especially suited, as it allows the efficient evaluation of object similarity, usually by calculating a Minkowski distance between feature vectors representing underlying 3D objects.

Foundations

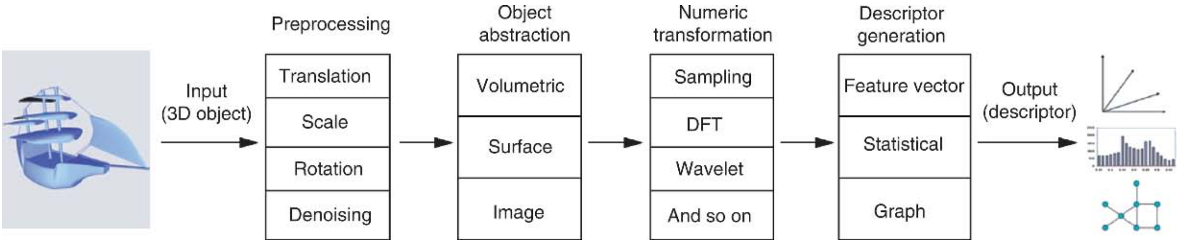
To represent 3D object data as points in feature vector space, it is necessary to find characteristics that describe the objects in a meaningful, discriminating way. A suitable feature extraction function calculates characteristic features from the 3D objects, thereby mapping them into *d*-dimensional feature vector space. With these feature vector representations, a similarity query in the original 3D object space is reduced to a search for close points in *d*-dimensional feature vector space.

Common Requirements of 3D Feature Extraction

For 3D object data, based on the given retrieval application, certain properties of the features extracted can be deemed desirable. The features may be required to be invariant with respect to changes in rotation, translation, and scale of the 3D models in their reference coordinate frame. Ideally, an arbitrary combination of translation, rotation and scale applied to one object should not affect its similarity measure with respect to another object. Another desirable property is robustness with respect to variation of the level-of-detail in which the 3D objects are given, and to small geometry and topology variations of the models. These invariance and robustness properties are especially important if the retrieval is expected to support 3D objects from heterogeneous data sources. This is because in such cases, the reference frames or levels-of-detail in which the models are represented may differ, and it cannot be assumed that respective meta data is available from all possible object sources.

3D Feature Extraction Process Model

A process model of 3D feature extraction is depicted in Fig. 1 and can be described as follows. Firstly, if required by the application, a preprocessing step normalizes the 3D object to approximate invariance to rotation, translation, scaling, and reflection [11]. A second step abstracts the 3D object according to a selected shape characteristic. For example, one can abstract a 3D object as a volume, or as an infinitely thin surface with precisely defined properties of differentiability, or as a set of 2D images formed by



Feature-Based 3D Object Retrieval. Figure 1. Feature extraction process for 3D objects.

projections from different perspectives. The third step captures the main features of the 3D object under the selected abstraction by means of a numeric transformation. As a result of this step, a numerical representation of the original 3D object is obtained. The last step of the feature extraction process model produces the final descriptor of the object from the numerical description. Generally, the descriptor may be a vector of numerical features, but it may also be a histogram of the measured characteristics, or a graph-based representation of the analyzed 3D object. Feature-based methods for 3D model retrieval usually are efficient, robust, and easy to implement. This does not imply, however, that statistical or graph-based methods should be disregarded. In fact, most of those methods have their particular strengths and may well be the ideal candidate for a specific application.

3D Feature Types

As surveys indicate [2,7,10], there is a wealth of different features that so far have been used to build 3D retrieval systems. The situation is comparable to content-based image retrieval (CBIR), where also, many different features have been proposed over the recent years. It can be stated that many of the 3D features proposed were heuristically introduced, motivated by techniques and practices from computer graphics (e.g., projection-based features), geometry processing (e.g., features based on surface curvature statistics), or Signal Processing (e.g., features obtained by representing object samples in the frequency domain). Some of the most effective 3D feature vector extractors proposed to date rely on features extracted from 2D projections of 3D objects.

Usually, it is a priori unclear which of the potentially many different features should be preferred for addressing the 3D retrieval problem. Each of the many possible descriptors captures specific model information, and their suitability for effective retrieval in a given application domain needs to be experimentally evaluated. In practice, it often shows that the effectiveness of 3D retrieval systems can benefit from using not a single, but several different types of features in combination.

Efficient 3D Object Retrieval

Similarity queries in 3D object databases may be answered by performing a sequential scan on the database, comparing the query object with all 3D objects stored in the database. This naive method might be too slow for real-world applications. In feature-based 3D object retrieval, the search system may use an index structure (e.g., spatial access methods or metric access methods) for efficient retrieval if the distance function used to compute the (dis)similarity of two 3D objects holds the properties of a metric (strict positiveness, symmetry, and the triangle inequality).

Spatial access methods [1] (also known as multidimensional indices) are index structures especially designed for vector spaces which, together with the metric properties of the distance function, use geometric information to discard points from the search space. Usually, these indices are hierarchical data structures that use a balanced tree to index the database. Metric access methods [4] (also known as metric indices) are index structures that use the metric properties of the distance function (especially the triangle inequality) to filter out certain zones of the space, thus avoiding the sequential scan.

Key Applications

Content-based 3D retrieval methods are potentially useful in all applications involving 3D object repositories, from which elements need to be retrieved based on geometric similarity. Several exemplary applications are detailed in the following, more exist.

Industrial Applications

Engineering and industrial design, the animation, and the entertainment industry heavily rely on digitized models of products or parts thereof. Computer-aided design allows the digital modeling of 3D content. Given effective retrieval capabilities, the re-usage of content from existing repositories can be supported for more efficient production processes [5].

Medicine

In medical imaging applications, often 3D volume data is generated, e.g., using MRI scans. A possible application lies in automatic diagnosis support by analysis of organ deformations, by matching actual images with medical database of known deformations.

Molecular Biology

Structural classification is a basic task in molecular biology. This classification can be supported by geometric similarity search, where proteins and molecules are modeled as 3D objects, which can be compared against bio-molecular reference databases using geometric similarity measures.

Future Directions

Feature-based 3D retrieval research is still in a rather early stage. Current approaches mostly consider features describing the geometry of whole models, that is, they support global similarity between objects. Recently, approaches also considering local features based on identification of salient object regions have been proposed. These are expected to support not only the retrieval of complete models, but also, be suited for retrieval based on local similarity. Future work will address 3D retrieval under additional similarity models, including similarity models invariant with regard to non-rigid and structural object deformations. It is also expected that application specific, specialized similarity notions will become increasingly important.

Experimental Results

The effectiveness of 3D features for retrieval is usually determined experimentally based on reference benchmarks, and measured by information retrieval metrics [3]. Well-known 3D retrieval benchmarks include the Princeton Shape Benchmark [9] and the Purdue Engineering Shape Benchmark [8]. The SHREC contest [6] is an International shape retrieval contest held regularly that involves different 3D retrieval challenges.

Cross-references

▶ Feature Extraction for Content-Based Image Retrieval
▶ Index Structures for Biological Sequences
▶ Information Retrieval
▶ Multimedia Information Retrieval Model
▶ Multimedia Retrieval Evaluation

Recommended Reading

1. Böhm C., Berchtold S., and Keim D. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. ACM Comput. Surv., 33(3):322–373, 2001.
2. Bustos B., Keim D., Saupe D., Schreck T., and Vranić D. Feature-based similarity search in 3D object databases. ACM Comput. Surv., 37(4):345–387, 2005.
3. Bustos B., Keim D., Saupe D., Schreck T., and Vranić D. An experimental effectiveness comparison of methods for 3D similarity search. Int. J. Digit. Lib., Special issue on Multimedia Contents and Management in Digital Libraries, 6(1):39–54, 2006.
4. Chávez E., Navarro G., Baeza-Yates R., and Marroquín J. Searching in metric spaces. ACM Comput. Surv., 33(3):273–321, 2001.
5. Funkhouser T., Kazhdan M., Shilane P., Min P., Kiefer W., Tal A., Rusinkiewicz S., and Dobkin D. Modeling by example. ACM Trans. on Graphics, 23(3):652–663.
6. http://www.aimatshape.net/event/SHREC S.I.S.R.C.
7. Iyer N., Jayanti S., Lou K., Kalyanaraman Y., and Ramani K. Three Dimensional Shape Searching: State-of-the-art Review and Future Trends. Comput. Aided Design, 37(5):509–530, 2005.
8. Jayanti S., Kalyanaraman Y., Iyer N., and Ramani K. Developing an engineering shape benchmark for CAD models. Comput. Aided Design, 38(9):939–953, 2006.
9. Shilane P., Min P., Kazhdan M., and Funkhouser T. The Princeton Shape Benchmark. In Proc. Int. Conf. on Shape Modeling and Applications, 2004, pp. 167–178.
10. Tangelder J. and Veltkamp R. A survey of content based 3D shape retrieval methods. In Proc. Int. Conf. on Shape Modeling and Applications, 2004, pp. 145–156.
11. Vranić D., Saupe D., and Richter J. Tools for 3D-Object Retrieval: Karhunen-Loeve Transform and Spherical Harmonics. In Proc. IEEE 4th Workshop on Multimedia Signal Processing, 2001, pp. 293–298.

Federated Database

▶ Distributed Architecture

Federated Database Systems

▶ Distributed Database Systems

Federated Search

▶ Searching Digital Libraries

Federated Search Engine

▶ Metasearch Engines

Feedback Systems

▶ Reputation and Trust
▶ Trust and Reputation in Peer-to-Peer Systems

Field-Based Information Retrieval Models

VASSILIS PLACHOURAS
Yahoo Reasearch Barcelona, Barcelona, Spain

Definition

A document *D* consists of a set of *n* document fields, and it is represented by a set of *n* vectors, where each vector corresponds to a document field. A field-based Information Retrieval (IR) model assigns a score or Retrieval Status Value (RSV) to a document *D* and a query *Q* by distinguishing the occurrences of query terms in the different field vectors, and by weighting the contribution of each field appropriately.

Historical Background

Textual documents, whether they are news wire items, scientific publications, or Web pages, are rich in structure. For example, depending on its length, a text can be organized in chapters, sections, paragraphs, and each of those can have a concise description in the form of a title. Shorter texts, such as emails, also consist of free text and formatted text. In information retrieval (IR), however, documents are usually represented as a single vector, the dimensions of which correspond to terms occurring in the document. Such a representation ignores the structure within a document because it does not distinguish between the occurrence of terms in different parts of a document, such as the document title, or the document abstract.

Field-based IR models overcome this limitation of representing the document as one vector, or as a bag-of-words, by incorporating in the document weighting process the fact that a term may appear in different document fields, and assigning different importance to the occurrences in each field. Information retrieval with document fields has a long history. Switzer [14] described a model where each index term is a vector image of the basic index terms. A document is represented by a title image, an author image, and citation images, constructed from bibliographic references. In such a representation, the title image is the average of vector images of the index terms appearing in the title. Fox [2] extended the boolean model and the vector space model with multiple concept types or fields. In the extended vector space model, a document is represented as a vector of subvectors. The ranking of documents in the extended vector space model is performed according to the weighted sum of the similarities between each of the subvectors and the queries. Fox also investigated the characteristics of several concept types related to bibliographic citations, as well as the weighting of the contribution of each of the subvectors [3]. Wilkinson studied the contribution of different fields, or document representations to retrieval effectiveness and found that improvements in early precision were obtained by using both the whole documents and their fields [16].

Foundations

There have been several proposed models to combine information from different document fields. One approach involves the use of structured document retrieval models, which allow for contained elements. Myaeng et al. [10] proposed a model in which documents and their elements or fields are represented as nodes in a network. Then, elements are ranked according to their support for the query, considering the contained elements as well. Lalmas [6] introduced a model where a document is represented as a tree. The leaf nodes of the tree correspond to the elements or fields of the document and the non-leaf nodes correspond to aggregates of its child nodes.

One other approach involves performing retrieval independently from each field and then, merging the ranked lists of results [1]. In this case, the combination of document fields takes place by merging the ranked lists of documents corresponding to each document field. This approach can be used to combine any source of evidence since it is based on the ranks of the retrieved documents, and not on their scores.

In the context of language modeling, the combination of fields, or different document representations, can be achieved with a linear combination of language models computed for each of the fields or document representations [11]. Similarly to the extended vector space model introduced by Fox [2], the combination of the different fields is also performed by computing a score or RSV for each of the document fields independently, and then performing a weighted sum of the computed scores.

The integration of information from the different fields in a retrieval model is not straightforward, because each document field may have different characteristics and the term frequency distribution can be different [5]. Thus, performing normalization and weighting independently for the various fields allows to take into account the different characteristics of the fields, and to achieve their most effective combination.

Robertson et al. [13] suggested that it is more appropriate to weight and combine the frequencies of terms from different fields in a *pseudo-frequency*, before applying a term weighting model. They argue that it is more intuitive to combine term frequencies rather than scores or RSVs assigned by scoring functions, which are not necessarily linear with respect to the term frequencies. Indeed, the linear combination of scores computed for each of the fields independently can lead to an overestimation of the importance of a term in a document. Then, Robertson et al. [13] proposed extending the BM25 weighting model to perform a weighted combination of document fields. The BM25 weighting model is given as follows:

$$w(D,Q) = \sum_{t \in D \cap Q} \frac{(k_1+1)tf}{(k_1(1-b)+b(l/\bar{l}))+tf} \cdot \log \frac{N-n_t+0.5}{n_t+0.5} \qquad (1)$$

where k_1, k_3, and b are free parameters, tf is the frequency of term t in D, l is the length of D, $\bar{l}$ is the average document length in the collection of documents, N and n are the number of documents and the document frequency of t in the collection. In the above equation, the frequency of terms in the query Q is not included. The proposed extension replaces tf in the above equation with a weighted sum of the term frequencies from each of the n document fields:

$$tf = \sum_{i=1}^{n} w_i \cdot tf_i \qquad (2)$$

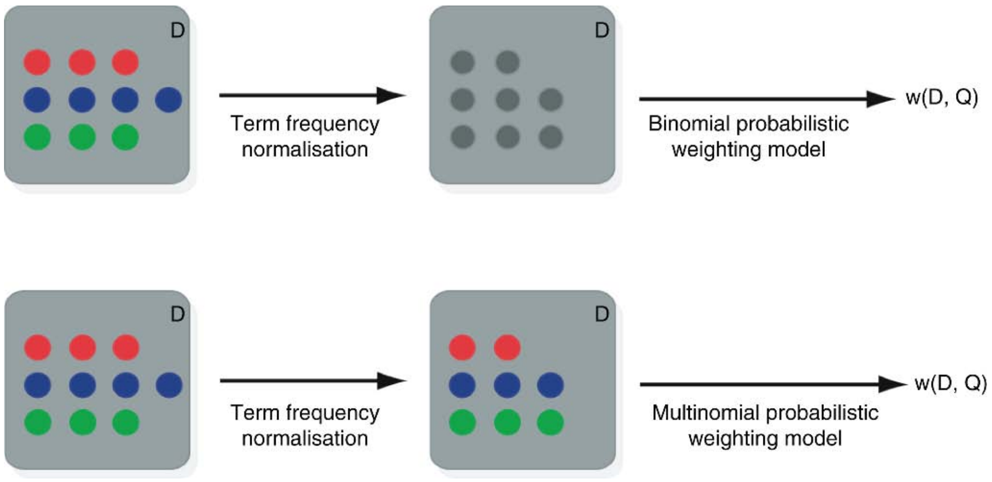
where tf_i is the frequency of term t in the i-th field and w_i is the weight of the i-th field. The described extension gives only a partial solution because it does address the combination of fields, but the term frequency normalization component is applied after computing the weighted sum of the term frequencies.

Zaragoza et al. [17] proposed BM25F, an extension of BM25 where length normalization is applied on a per-field basis. The formula of BM25F is given below:

$$\begin{aligned} w(D,Q) &= \sum_{t \in Q} \frac{tfn}{k_1 + tfn} \cdot \log \frac{N-n_t+0.5}{n_t+0.5} \\ \text{where } tfn &= \sum_{i=1}^{n} w_i \cdot \frac{tf_i}{(1+b_i(l_i/\bar{l}_i))} \end{aligned} \qquad (3)$$

In the above equation, b_f is a field-dependent normalization parameter, similar to the parameter b of BM25, k_1 is a parameter that controls the saturation of tfn, similar to the parameter k_1 of BM25; $\bar{l}_i$ is the average length of the i-th field in the document collection; and l_i is the length of the i-th field in D. The parameter w_i is the weight of the i-th field.

The Divergence From Randomness (DFR) framework has also been extended to handle multiple document fields, and to apply per-field term frequency normalization and weighting. DFR is a framework for generating families of probabilistic retrieval models consisting of three components. The weighting


D
Term frequency normalisation
D
Binomial probabilistic weighting model
w(D, Q)
D
Term frequency normalisation
D
Multinomial probabilistic weighting model
w(D, Q)

Field-Based Information Retrieval Models. Figure 1. An illustration of combination of document fields in the term frequency normalization component(top) and in the probabilistic retrieval model (bottom).

models of the Divergence From Randomness framework are based on combinations of three components: a randomness model $\mathcal{RM}$; an information gain model $\mathcal{GM}$; and a term frequency normalization model. Given a collection C of documents, the randomness model $\mathcal{RM}$ estimates the probability $P_{\mathcal{RM}}(t \in D|C)$ of having *tf* occurrences of a term t in D. The information gain model $\mathcal{GM}$ estimates the informative content $1 - P_{risk}$ of the probability P_{risk} that a term t is a good descriptor for a document. The third component of the DFR framework is the term frequency normalization model, which adjusts the frequency *tf* of the term t in d, given the length l of d and the average document length $\bar{l}$ in D.

There have been two ways proposed to extend the DFR weighting models with document fields. The first way to incorporate fields is by extending the term frequency normalization component of the DFR framework [12]. The frequency tf_i of term t in the i-th field is normalized and weighted independently of the other fields. Then, the normalized and weighted term frequencies are combined into one pseudo-frequency.

When combining the term frequencies in the length normalization component, as it happens in BM25F and in normalization 2F, it is implied that the term frequencies are drawn from the same distribution, even though the nature of each field may be different. The second way to incorporate fields in the DFR framework is by using multinomial randomness models [12]. Using the multinomial distribution, the probability that a term occurs tf_i times in the i-th field of D is given as follows:

$$P_{\mathcal{M}}(t \in D|C) = \binom{TF}{tf_1 \quad tf_2 \cdots tf_n \quad tf'} p_1^{tf_1} p_2^{tf_2} \cdots p_n^{tf_n} p'^{tf'} \qquad (4)$$

In the above equation, TF is the frequency of t in the collection, $p_i = \frac{1}{n \cdot N}$ is the prior probability that a term occurs in a particular field of D, and N is the number of documents in the collection C. The frequency $t' = TF - \sum_{i=1}^{n} tf_i$ corresponds to the number of occurrences of t in documents other than D. The probability $p' = 1 - n\frac{1}{n \cdot N} = \frac{N-1}{N}$ corresponds to the probability that t does not appear in any of the field-s of D. The use of the multinomial distribution as a randomness model requires the computation of several factorials, which can be expensive and also may introduce approximation errors. To overcome the need to compute these factorials, Plachouras and Ounis [12] also used an information theoretic approximation of the multinomial distribution.

The models described above require additional information from the index to compute efficiently scores of documents. In both the cases of BM25F and the DFR field-based weighting models, the index must contain the frequency of a term in each of the document fields, rather than just the frequency of the term in the whole document. For each document, it is also required to have the length of each field to perform length normalization on a per-field basis.

Key Applications

There is a wide range of applications for which field-based IR models enhances the retrieval effectiveness. One of the most important and widely used ones is Web search, where useful fields are the title of Web documents as well as the anchor text of incoming hyperlinks. A different application in which retrieval effectiveness increases by using document fields is email search [17] where the email from, to and date, as well as the quoted text, can be considered as a different field. In general, field-based IR models can be readily applied to search tasks in which documents have metadata elements associated or in which parts of the documents are annotated.

Future Directions

The introduction of fields increases the complexity of the retrieval models by introducing parameters related to the weighting of each field, but also to other factors such as the length normalization of each field. As an example, BM25F [17] requires the tunning of $2n + 1$ parameters when using n different document fields. Similarly, PL2F [8] introduces $2n$ parameters when using n different fields. Since the fields are not necessarily independent, setting the parameters for one field depends also on the parameter setting for the other ones. As the number of fields and the number of parameters increases, performing an exhaustive search becomes prohibitive. The application of an extension of gradient descent has been proposed to set the document field parameters among other parameters [15]. Developing parameter-free field-based IR models, however, is still an open problem.

Data Sets

There are several available data sets that allow experimentation with field-based IR models. The most commonly used ones are the standard TREC Web test collections WT10g, .GOV and .GOV2, which have been used for a range of search tasks, such as ad-hoc retrieval, named page and home page finding, as well as topic distillation. All these tasks have been evaluated in the context of Web track of Text REtrieval Conference (TREC) [4]. Another standard test collection that has been used for expert search finding and email search in mailing lists is the W3C collection, a crawl of the World Wide Web Consortium Web site available from TREC as well.

The INitiative for the Evaluation of XML Retrieval (INEX) [9], has also developed collections using XML. The first one consists of articles marked up in XML from a number of the IEEE Computer Society's publications. Other collections correspond to the contents of Wikipedia and to the contents of travel guides, marked up in XML.

Cross-references

► BM25
► Content-and-Structure-Query
► Divergence from Randomness Models
► Document Length Normalization
► Information Retrieval Model
► Vector Space Model

Recommended Reading

1. Fagin R., Kumar R., McCurley K.S., Novak J., Sivakumar D., Tomlin J.A., and Williamson D.P. Searching the workplace web. In Proc. 12th Int. World Wide Web Conference. 2003, pp. 366–375.
2. Fox E.A. Extending the Boolean and Vector Space Models of Information Retrieval with P-Norm Queries and Multiple Concept Types. Ph.D dissertation, Cornell University, 1983.
3. Fox E.A. Coefficients of combining concept classes in a collection. In Proc. 11th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1988, pp. 291–307.
4. Hawking D. and Craswell N. The very large collection and Web tracks. In TREC: Experiment and Evaluation in Information Retrieval, E. Voorhees, D. Harman (eds.). MIT, Cambridge, MA, USA, 2005, pp. 199–232.
5. Hawking D., Upstill T., and Craswell N. Toward better weighting of anchors. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 512–513.
6. Lalmas M. Uniform representation of content and structure for structured document retrieval. Technical report, Queen Mary University of London, 2000.
7. Macdonald C. and Ounis I. Combining fields in known-item email search. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 675–676.
8. Macdonald C., Plachouras V., He B., Lioma C., and Ounis I. University of Glasgow at WebCLEF 2005: experiments in per-field normalisation and language specific stemming. In Accessing Multilingual Information Repositories, Sixth Workshop of the Cross-Language Evaluation Forum, 2005, pp. 898–907.
9. Malik S., Trotman A., Lalmas M., and Fuhr N. Overview of INEX 2006. In Comparative Evaluation of XML Information Retrieval Systems. LNCS 4518, Springer, Berlin, 2007, pp. 1–11.
10. Myaeng S.H., Jang D.H., Kim M.S., and Zhoo Z.C. A flexible model for retrieval of SGML documents. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 138–145.
11. Ogilvie P. and Callan J. Combining document representations for known-item search. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 143–150.
12. Plachouras V. and Ounis I. Multinomial randomness models for retrieval with document fields. In Proc. 29th European Conf. on IR Research, 2007, pp. 28–39.
13. Robertson S., Zaragoza H., and Taylor M. Simple BM25 extension to multiple weighted fields. In Proc. Int. Conf. on Information and Knowledge Management, 2004, pp. 42–49.
14. Switzer P. Vector images in information retrieval. In Proc. Symp. on Statistical Association Methods for Mechanical Documentation,1965, pp. 163–171.
15. Taylor M., Zaragoza H., Craswell N., Robertson S., and Burges C. Optimisation methods for ranking functions with multiple parameters. In Proc. Int. Conf. on Information and Knowledge Management, 2006, pp. 585–593.
16. Wilkinson R. Effective retrieval of structured documents. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval. 1994, pp. 311–317.
17. Zaragoza H., Craswell N., Taylor M., Saria S., and Robertson S. Microsoft Cambridge at TREC-13: Web and HARD tracks. In Proc. 13th Text Retrieval Conf., 2004.

Field-Based Spatial Modeling

MICHAEL F. GOODCHILD
University of California-Santa Barbara, Santa Barbara, CA, USA

Definition

A field (or continuous field) is defined as a mapping from location **x** to a function *f*. In modeling geographic phenomena the domain of **x** is most often the two dimensions of geographic space, but may include the third spatial dimension for applications that extend above or below the Earth's surface, and may include time for dynamic phenomena. Fields can also be defined on one-dimensional networks embedded in two- or three-dimensional space. Moreover, most applications are limited to a specified sub-domain of geographic space, such as the limits of a country or county, or of a map sheet or arbitrarily defined study area. The domain of *f* includes scalar measurements on interval and ratio scales, nominal and ordinal classifications, and vectors describing such directional phenomena as wind or topographic gradient. Field-based spatial modeling can in principle be employed in the representation of any space, including the spaces of the human brain, the surfaces of other planets, or complex buildings.

Fields are one of two ways of conceptualizing the geographic world. By contrast, the discrete-object conceptualization imagines a world that is empty except where it is occupied by discrete, countable objects that maintain identity and geometric form through time. This conceptualization is more often adopted in the representation of biological organisms, manufactured objects, and man-made structures, whereas the continuous-field conceptualization is more appropriate for variables that can be defined at every location in space, such as air temperature, terrain height, soil moisture content, or wind speed.

Historical Background

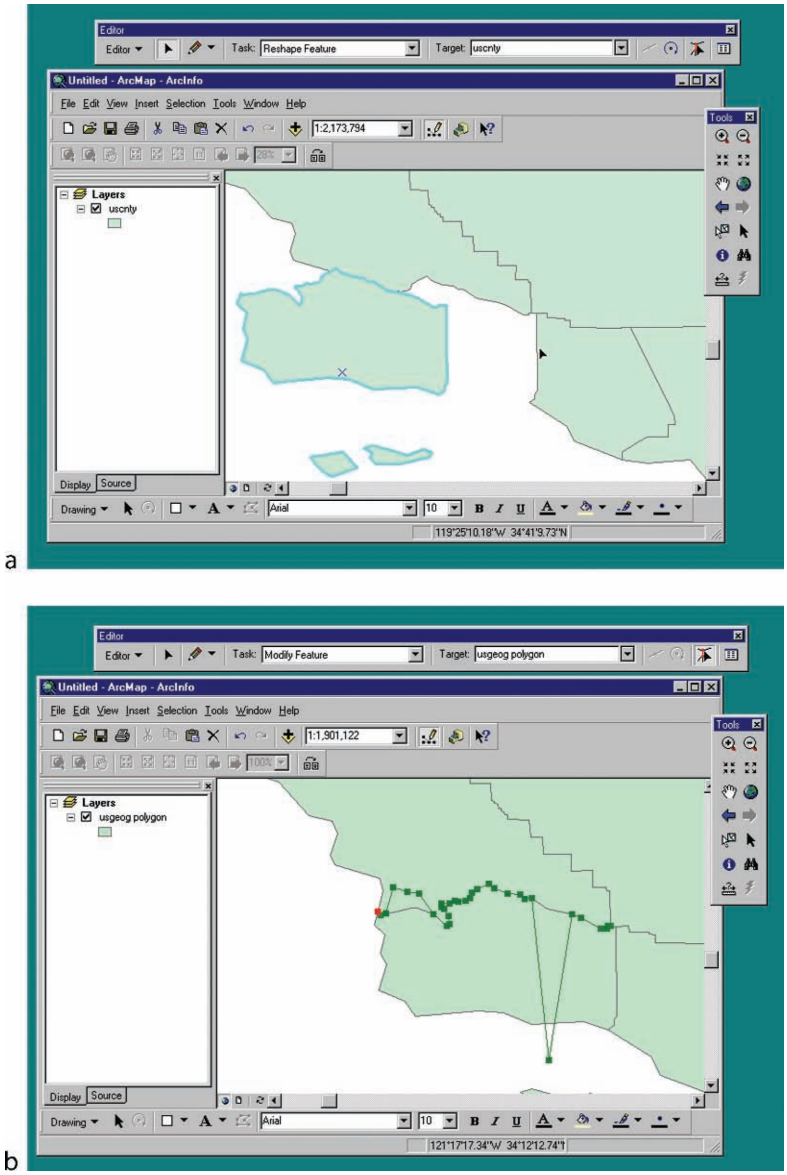
The field/object distinction has ancient roots. Its significance for geographic information science was first recognized in the late 1980s and early 1990s [1,3], and it has come to be acknowledged as the most important distinction underpinning the entire field of geographic data modeling [5,7]. Consider, for example, the section of the US county boundary map shown in Fig. 1. If counties are regarded as discrete objects then one should be able to move them around as pieces of a jigsaw, possibly overlapping and modifying the coastline of the US, as shown on the left. On the other hand if the variable *county* is regarded as a single-valued function of location **x**, defined everywhere inside the boundary of the US, then the boundaries merely indicate where the value of *county* changes. Edits that move common boundaries are feasible, but not edits that move the coastline, or edits such as that shown, which attempts to move vertices of the boundary outside the coastline. Conceptualizing the phenomenon as a field clearly results in a more consistent representation that is more appropriate to the nature of the phenomenon.

Figure 2 shows an area-class map, depicting the variation in vegetation cover class across a geographic domain. If such a map were conceptualized as a collection of discrete objects, then the issue of accuracy would come down to such questions as "Is the number of areas correct?," "Are the boundaries in the correct places?," and "Are the classes assigned to each area correct?" However uncertainty would be addressed very differently under a field conceptualization; one could ask only one form of question, "Is the value recorded at **x** correct?" It turns out that this second conceptualization is far more productive and tractable than the first [8].

Strong associations have been recognized between the two alternative conceptualizations and patterns of human thought. Many would argue that the discrete-object view is more compatible with human cognition – that human brains are in effect hard-wired to segment any visual image into a collection of discrete objects, and to track their movements through time. On the other hand the continuous-field view underlies many of the most significant advances in science, including electro-magnetism (the Maxwell equations), hydrodynamics (the Navier-Stokes equation), and quantum mechanics (the Schrödinger equation). Some of the most challenging problems in science are described in partial differential equations defined on fields, and solved using a variety of computational methods. The distinction is strongly linked to concepts of scale, as for example when the behavior of a group of ants is modeled at very detailed scale as interactions between discrete objects, or at a coarser scale as modifications to a continuous field of ant density. Very crudely, continuous-field conceptualizations tend to be more common in the natural and physical sciences, while discrete-object conceptualizations are more often found with reference to social phenomena.

Foundations

Computers are fundamentally discrete machines, founded on the representation of information in a two-valued alphabet, and as a result discrete-object conceptualizations are more readily implemented in spatial databases. There is little ambiguity, for example, in the representation of the current location of every aircraft in an airline's fleet as a point in space. However other phenomena are inherently continuous across space, including terrain, rivers, roads, and the tracks of moving objects, and their representation necessarily involves some form of discretization. A river, for example, may be broken into reaches, either at junctions or at points where the direction of the river changes significantly. Each reach will be represented using a simple mathematical function, most often a straight



Task: Reshape Feature
Target: uscnty
1:2,173,794
Layers
uscnty
Arial 10
119°25'10.18"W 34°41'9.73"N

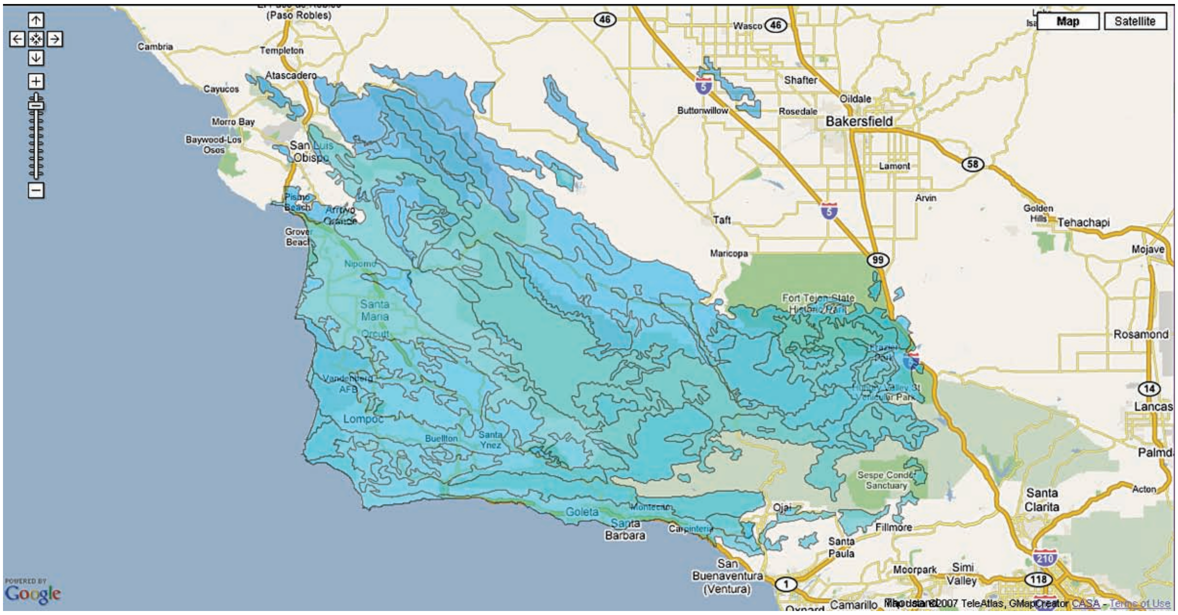
Task: Modify Feature
Target: usgeog polygon
1:1,901,122
100%
Layers
usgeog polygon
Arial 10
121°17'17.34"W 34°12'12.74"N

a
Editor
Untitled - ArcMap - ArcInfo
File Edit View Insert Selection Tools Window Help
Tools
Display Source
Drawing
b
Editor
Untitled - ArcMap - ArcInfo
File Edit View Insert Selection Tools Window Help
Tools
Display Source
Drawing

Field-Based Spatial Modeling. Figure 1. Counties conceptualized as (a) discrete objects and (b) a continuous field. As a discrete object, any county is free to move independently of its neighbors and the coastline. As a continuous field, however, only the common boundaries where county values change can be moved, and not so far as to intersect the coastline (the edit shown would not be accepted).

line but sometimes as an arc of a circle or a spline function. Roads in a road network may be broken into segments at intersections, or at other points where direction changes. These stages necessarily modify the phenomena they are used to represent, since the discretized object will in most cases differ geometrically from the original, and discretization is essential if a geometric shape that is potentially infinitely complex is to be represented in a digital store of finite capacity. Thus in almost all cases the representation of real-world geometry requires the loss of some degree of detail.

This stage of discretization is necessary whenever an arbitrarily shaped geographic feature must be represented in digital form. But a further stage is needed when the characteristics of the phenomenon also vary continuously over space, in other words in the representation of phenomena conceptualized as continuous fields. Figure 3 shows the six methods most commonly



Field-Based Spatial Modeling. Figure 2. A map of vegetation cover class conceptualized as a nominal field, superimposed on the Santa Barbara area. Each area denotes a particular type of vegetation.

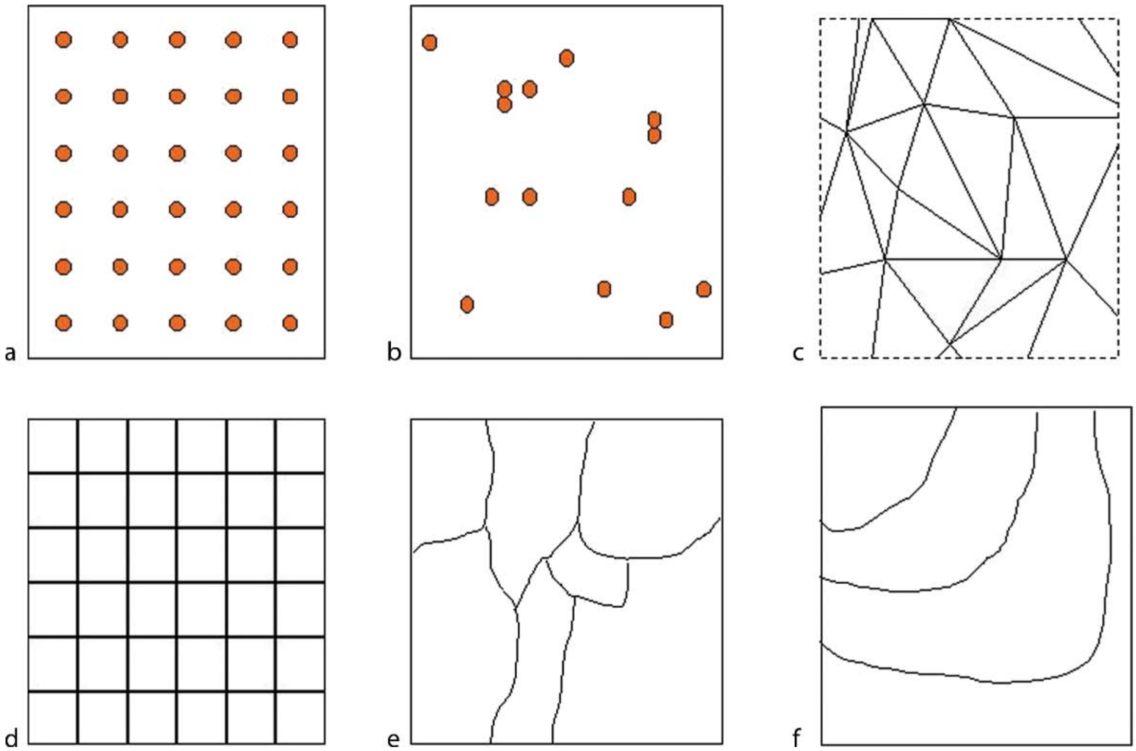
employed in spatial databases and geographic information systems. Two are based on a raster, in other words the discretization of space into a regular array with finite spacing, while the remaining four use vector methods, specifying the position of each element of the discretization as coordinates and representing volumes as collections of polygonal faces, areas as polygons, and lines as polylines.

Figure 3a shows one of the raster options, the representation of a spatially continuous phenomenon as a collection of sample values regularly spaced over a rectangular array. This is the method most commonly employed in the representation of terrain, in what are known as digital elevation models (DEMs). The spacing of sample heights implies a well-defined level of spatial resolution. Note, however, that since it is impossible to lay a rectangular grid over a curved surface, it is similarly impossible to construct a DEM of a significant part of the Earth's surface with a precisely constant spatial resolution.

Figure 3d shows the other raster option. Here the study area has been divided into rectangular cells, and a single measurement provided in each cell. This is often the mean value in the case of measurements on interval and ratio scales, but in other cases may be a modal value or some more complex function of the values of the field within the cell. Again spatial resolution is constant and well defined. This approach is most commonly encountered in the discretization of images, such as those obtained from satellites.

Figure 3b shows a vector option. In this case the field has been sampled at a set of irregularly spaced locations, and the value of f reported. Sample locations may be selected using some specified rule, or may be a historic artifact, as they are in the case of weather data, when the points correspond to weather-observation stations. Nothing is known about the values of f between sample locations, though a wide range of techniques exist for making intelligent guesses under the rubric of spatial interpolation.

Figure 3c shows another vector option involving irregularly spaced sample points, though in this case a set of triangles have been created by connecting them. The field variable is assumed to vary linearly between points, ensuring continuity of value across triangle edges. In essence this option, which is generally known as a triangulated irregular network (TIN) or triangular mesh, adds a specific method of spatial interpolation to Option 3b. It is commonly used as a method for representing terrain, where it is particularly efficient for terrains characterized by long uniform slopes and sharp ridges; and as an internal representation in contouring algorithms.



Field-Based Spatial Modeling. Figure 3. The six common methods of discretizing a field. See text for explanation.

Figure 3e shows the discretization of a field as a collection of space-exhausting, non-overlapping areas, that will themselves be discretized geometrically as polygons. The field variable is assumed uniform within each area. This is the most common discretization when the field variable is nominal or ordinal, as it is for maps of soil class and land cover class for example. It is also commonly used to represent data collected by statistical agencies and aggregated by reporting zones such as counties or census tracts. In such cases, the number of variables recorded for each reporting zone may be very large. For some of these variables, termed spatially intensive, the value reported will represent the mean within the zone of a variable such as average income, or a proportion such as percent black, or a density per unit area. In other cases the reported value will be a total, such as total population or total income, corresponding to the integration of a density over the reporting zone's area; this type of value is termed spatially extensive.

Finally, Fig. 3f shows the last example, in which the field is represented as a collection of digitized isolines. This method is commonly used to capture the contours shown on topographic maps. While it is effective for visual purposes, its value from an analytic perspective is far less than the DEM or TIN because of the very uneven sampling that is achieved.

Key Applications

Many examples of phenomena conceptualized as fields have already been cited; this section addresses the functions that are commonly applied to field representations, and the software that implements them. The functions described in this section are commonly found in GIS packages, and they and many others are reviewed by De Smith, Goodchild, and Longley [2].

A variety of functions are available to manipulate representations of topography. Most use the DEM option (3a) though very powerful algorithms have been described for similar operations on TINs. Visualization is a common requirement, and functions have been developed to compute surface gradient and hence simulated illumination; to compute and plot isolines; and to compute solar insolation as a key variable in understanding vegetation patterns on rugged topography. Another class of algorithms concern visibility, and can be used to compute the area visible by an observer positioned a given height above the terrain. Algorithms have been described for computing the most exposed and most concealed points on a landscape, as well as most exposed and most concealed routes; and to position a minimal number of observers such that the entire landscape can be observed.

Another important collection of algorithms concerns drainage. Starting with a DEM, it is possible to compute drainage directions as a vector field, and to integrate these into catchments and stream channels. DEMs are used to predict and manage flooding, and to plan modifications to the landscape such as the construction of levees.

Reference was made earlier to methods of spatial interpolation, which address the task of predicting the value of a field at locations where it has not been measured. Most often these methods are applied to the representations illustrated in Fig. 3a and 3b, but a related technique known as areal interpolation has been devised for the task of predicting the values associated with areas that do not match (cut across the boundaries of) the reported areas of 3d and 3e.

Of particular interest are algorithms that produce representations of fields from collections of discrete objects. They include density estimation, which produces a field of feature density, most often of points; and calculation of the distance from any point in the plane to the nearest of a collection of discrete objects.

The most powerful collections of field-based manipulation functions clearly exist for raster data, because of the possibility that a collection of fields can be represented using a set of co-registered rasters. This is the principle of raster GIS, most clearly illustrated by packages with firm roots in raster-based discretizations, such as Idrisi and GRASS. Most have adopted a common language for expressing instructions known as map algebra [6]. A powerful alternative geared particularly to simulation is PCRaster (http://pcraster.geo.uu.nl), developed at the University of Utrecht and having its own manipulation language that is considerably more succinct and powerful than map algebra.

Future Directions

The comparative importance of continuous fields and discrete objects is a matter of continual debate. On the one hand, the majority of GIS applications occur in worlds where the objects of interest are well-defined and often man-made. On the other hand, many phenomena in the natural world are essentially continuous, and representing them as collections of discrete objects invites error and misuse. Cognitive scientists would argue that humans are hard-wired to see the world as a collection of discrete objects, while environmental scientists might argue that continuous fields are one of the most significant breakthroughs in the history of science, lying at the root of such developments as the calculus and hydrodynamics. Many of the forms of analysis commonly used in the environmental sciences are based on fields; while most of those commonly used in the social sciences are based on discrete objects.

GIS software today embraces both conceptualizations. But it does so in a somewhat unsatisfactory manner, in which representations of continuous fields must be reduced to collections of discrete objects, without any record of that process of reduction. Thus it is impossible for the user of a GIS database to enquire whether a collection of points in the database represents a set of points sampling a field, or a set of isolated point-like objects in an otherwise empty space. Unfortunately this means that inappropriate operations can easily be performed. Nothing prevents the GIS user from applying spatial interpolation to points conceptualized as discrete objects, or from computing a density field of sample points; the former is of course more disastrous than the latter. Similarly nothing prevents a GIS user from moving an isoline so that it crosses another, or from making two polygons in a Fig. 3e representation overlap.

Several interesting and potentially powerful research directions have been pursued in the hope of eventually improving this situation. One approach has been to argue that the user should be able to interact with the concept of a field directly, rather than with the elements of one of the six representations of Fig. 3 as at present. Another has been to search for a visual paradigm for handling fields that matches and has similar power to the visual representation of sets of discrete objects that is found in UML (Unified Modeling Language) and related methods. Now that GIS technology includes the capability to design databases in UML and to create and populate the necessary tables automatically, it would make good sense to develop parallel methods for handling fields. Finally, much recent effort has gone into finding ways of reconciling and bridging the field/object dichotomy [4].

Cross-references

▶ Digital Elevation Models
▶ Geographic Information System

▶ Raster Data Management and Multi-Dimensional Arrays
▶ Spatial Data Analysis
▶ Triangulated Irregular Network
▶ Unified Modeling Language

Recommended Reading

1. Couclelis H. People manipulate objects (but cultivate fields): beyond the raster-vector debate in GIS. In Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, A.U. Frank, I. Campari, U. Formentini (eds.). Springer, Berlin, 1992, pp. 65–77.
2. De Smith M.J., Goodchild M.F., and Longley P.A. Geospatial Analysis: A Comprehensive Guide to Principles. Techniques and Software Tools. Winchelsea Press, UK, 2007.
3. Goodchild M.F. Modeling error in objects and fields. In Accuracy of Spatial Databases, M.F. Goodchild, S. Gopal (eds.). Taylor and Francis, Bristol/London, 1989, pp. 107–114.
4. Goodchild M.F., Yuan M., and Cova T.J. Towards a general theory of geographic representation in GIS. Int. J. Geogr. Inf. Sci., 21(3):239–260, 2007.
5. Longley P.A., Goodchild M.F., Maguire D.J., and Rhind D.W. Geographic Information Systems and Science, Wiley, West Sussex, 2005.
6. Tomlin C.D. Geographic Information Systems and Cartographic Modeling. Prentice Hall, Englewoods, NJ, 1990.
7. Worboys M.F. and Duckham M. GIS: A Computing Perspective, CRC Press, Boca Raton, FL, 2004.
8. Zhang J.X. and Goodchild M.F. Uncertainty in Geographical Information. Taylor and Francis, London/New York, 2002.

File Compression

▶ Text Index Compression

File Format

▶ Image Representation

Filter/Refinement Query Processing

▶ Multi-Step Query Processing

Finding of Observation

▶ Clinical Observation

Finiteness

▶ Safety and Domain Independence

First-Order Logic: Semantics

VAL TANNEN
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

Predicate calculus; Predicate logic
FOL

Definition

This entry should be read in conjunction with the companion entry *First-Order Logic: Syntax* where the terms vocabulary, variable, formula, etc. are defined.

To give semantics to first-order sentences, first-order *structures* (a.k.a. models or interpretations), and the "*holds true*" (a.k.a. satisfaction or validity) relationship is used between sentences and structures. Both are detailed below. This allows the definition of *logical consequence* (a.k.a. logical implication), $\Gamma \models \varphi$, whose meaning is that the sentence φ holds true in any structure in which the sentences in the set Γ hold true. Proof systems for first-order logic should be assessed against logical consequence whether *sound* and/or *complete*. The semantic of FOL is also used for *definability*, a concept that helps understanding the limitations of formalisms based on FOL.

As introduced by E. F. Codd into database technology, the *relational model* is based on finite first-order structures for a vocabulary that has only relational symbols.

Key Points

Fix a first-order vocabulary. A *structure* $\mathcal{A}$ for this vocabulary consists of a non-empty set A (sometimes called the "universe (of discourse)" of the structure) together with a mapping that assigns to every symbol s in the vocabulary a corresponding *meaning* $s^{\mathcal{A}}$ over A. For constants c, $c^{\mathcal{A}}$ is just an element of A, for n-ary relation symbols R, $R^{\mathcal{A}}$ is an n-ary relation over A (i.e., a subset of A^n), and for n-ary function symbols f one has a function $f^{\mathcal{A}} : A^n \to A$.

Next, the goal is to define when a sentence φ *holds true* in a structure $\mathcal{A}$, written $\mathcal{A} \models \varphi$. To do this, one should use *assignments*, which are partial functions from the set of all variables to the universe A of $\mathcal{A}$. It

$$\begin{array}{rcl} \bar{\mu}(x) & = & \mu(x) \\ \bar{\mu}(c) & = & c^{\mathcal{A}} \end{array} \qquad \bar{\mu}(f(t_1,\ldots,t_n)) \;=\; f^{\mathcal{A}}(\bar{\mu}(t_1),\ldots,\bar{\mu}(t_n))$$

$$\begin{array}{rll} \mathcal{A},\mu \models t_1 = t_n & \textit{iff} & \bar{\mu}(\mathrm{t}_1) = \bar{\mu}(\mathrm{t}_2) \\ \mathcal{A},\mu \models \textit{true} & & \\ \mathcal{A},\mu \models \varphi_1 \vee \varphi_2 & \textit{iff} & \mathcal{A},\mu \models \varphi_1 \text{ or } \mathrm{A},\mu \models \varphi_2 \\ \mathcal{A},\mu \models \neg\varphi & \textit{iff} & \mathcal{A},\mu \not\models \varphi \end{array} \qquad \begin{array}{rll} \mathcal{A},\mu \models R(t_1,\ldots,t_n) & \textit{iff} & (\bar{\mu}(t_1),\ldots,\bar{\mu}(tn)) \in R^{\mathcal{A}} \\ \mathcal{A},\mu \not\models \textit{false} & & \\ \mathcal{A},\mu \models \varphi_1 \wedge \varphi_2 & \textit{iff} & \mathcal{A},\mu \models \varphi_1 \text{ and } \mathcal{A},\mu| = \varphi_2 \\ \mathcal{A},\mu \models \varphi_1 \rightarrow \varphi_2 & \textit{iff} & \mathcal{A},\mu \models \varphi_2 \text{ whenever } \mathcal{A},\ \mu \models \varphi_1 \end{array}$$

$$\begin{array}{rll} \mathcal{A},\mu \models \exists x\,\varphi & \textit{iff} & \textit{there is } a \in A \text{ such that } \mathcal{A},\mu[x := a] \models \varphi \\ \mathcal{A},\mu \models \forall x\,\varphi & \textit{iff} & \textit{for each } a \in A,\ \mathcal{A},\mu[x := a] \models \varphi \end{array}$$

suffices to consider assignments of finite domain (defined only on finitely many variables). By extending assignments $\mu\ :Vars{\rightarrow}A$ to $\bar{\mu}\ :Terms\ \rightarrow\ A$, define $\mathcal{A},\mu \models \varphi$ where φ is a formula and such that μ is defined on (at least) all the free variables of φ:
where $\mu[x := a]$ is the same as μ except that $\mu[x := a]$ $(x) = a$. The definition above is uncomfortably "circular" if one thinks of FOL or other logical formalisms as part of a foundation for mathematics. But this definition of truth is part of a more useful point of view, due to Hilbert and Tarski, in which logic is a mathematical symbol "game" and one can use ordinary math to study it (this use of math is called "metamathematics").

If φ is a sentence it is possible to define when it holds in a structure $\mathcal{A}$: $\mathcal{A} \models \varphi$ means $\mathcal{A}, \emptyset \models \varphi$ where $\emptyset$ is the assignment defined nowhere (empty domain). A sentence is *valid* if it holds in all structures. At first validity seems computationally absurd (there isn't even a "set of all structures", because Russell's Paradox would be lurking in the fold). However, a very important result in metamathematics, Gödel's Completeness Theorem [2], shows that validity is equivalent to provability in one of the many equivalent proof systems for FOL. It follows that the set of all valid sentences is in fact recursively enumerable (r.e.), a reasonable foundation for attempting automated theorem-proving for FOL. However, Church and Turing have shown that validity is undecidable [2], an important inherent limitation that also affects applications to databases.

In fact, Gödel's Completeness Theorem shows more. If Γ is a set of sentences and φ a sentence, one defines *logical consequence* by

$$\Gamma \models \varphi \quad \textit{iff} \quad \textit{for all } \mathcal{A},\ \textit{if } \mathcal{A} \models \psi \textit{ for each } \psi \in \Gamma \textit{ then } \mathcal{A} \models \varphi$$

Fix a proof system with provability relation ⊢. The proof system is *sound* if $\Gamma \vdash \varphi$ implies $\Gamma \models \varphi$ and is *complete* if $\Gamma \models \varphi$ implies $\Gamma \vdash \varphi$. Gödel's Completeness Theorem shows that any of the equivalent proof systems for FOL is sound and complete.

A sentence is *finitely valid* if it holds in all finite structures. Finite validity is co-r.e. (just enumerate finite structures up to isomorphism). However, Trakhtenbrot's Theorem [3] shows that it is undecidable, hence it cannot be r.e. (in fact, it shows that it is co-r.e.-complete). Therefore, one cannot hope to find a sound and complete proof system for just the finitely valid sentences.

Finally, consider a structure $\mathcal{A}$ and an n-ary relation $D \subseteq A^n$. Say that D is *first-order definable in* $\mathcal{A}$ if there exists an FOL formula φ with exactly n distinct free variables $x_1,\ldots,x_n$ such that

$$\textit{for all } a_1,\ldots,a_n \in A, \qquad (a_1,\ldots,a_n) \in D \quad \textit{iff} \\ \mathcal{A}, [x_1 := a_1,\ldots,x_n := a_n] \models \varphi$$

Definability in a given structure is trivial if the structure is finite. For finite structures look at the definability of *queries*.

Cross-references

- ▶ First-Order Logic
- ▶ First-Order Logic: Syntax
- ▶ Relational Calculus

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases: The Logical Level. Addison Wesley, Reading, MA, 1994.
2. Enderton H.B. A Mathematical Introduction to Logic, 2nd edn. Academic, New York, 2000.
3. Libkin L. Elements of Finite Model Theory. Springer, 2004.

First-Order Logic: Syntax

VAL TANNEN
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

Predicate calculus; Predicate logic; FOL

Definition

First-order logic (FOL) is a formalization of the most common kind of mathematical reasoning. It is characterized by the *quantification* of *variables* that range over a "universe of discourse" (a set of values). Less complex reasoning is captured by *propositional* (a.k.a. Boolean or sentential) logic. More complex reasoning is captured by *second-order* or even *higher-order* logic.

The syntactic aspects of FOL comprise a *vocabulary* (a.k.a. language or signature), *formulae* and, in particular, *sentences* (a.k.a. assertions), and a *proof system* (one of many equivalent ones!), *structures* (a.k.a. models or interpretations), and the *satisfaction* (a.k.a. truth or validity or "holds in") relationship between sentences and structures. All are detailed below.

FOL is the source of the *relational* paradigm that was introduced by E. F. Codd in 1970 and has been dominating database technology for 30+ years.

Key Points

A first-order *vocabulary* consists of a set of *constant* symbols, for each integer $n \geq 0$ a set of n-ary *relation* (a.k.a. predicate) symbols, for each integer $n \geq 1$ a set of n-ary *function* symbols. For some authors the constants are the 0-ary function symbols. The 0-ary relation symbols are called "propositional constants" (although, in propositional logic they are called "variables"!). In addition, first-order vocabularies are often assumed to contain the equality symbol—a distinguished binary relation symbol – as well as *true* and *false* as distinguished propositional constants. For applications in Computer Science, these sets of symbols are usually finite.

Next, a vocabulary and an infinite set of *variables* are fixed. First-order *terms* are defined by the grammar

$$t ::= c \mid x \mid f(t_1,\ldots,t_n)$$

where c ranges over constants, x over variables, f over n-ary function symbols, and t, $1,\ldots,t_n$ over terms, for each integer $n \geq 1$. First-order *formulae* (sometimes called *well-formed formulae* of *wff's* are defined by the grammar

$$\begin{gathered}\varphi ::= R(t_1,\ldots,t_n) \mid t_1 = t_2 \mid \mathit{true} \mid \mathit{false} \mid \\ (\varphi_1 \vee \varphi_2) \mid (\varphi_1 \wedge \varphi_2) \mid (\neg\varphi) \mid \\ (\varphi_1 \rightarrow \varphi_2) \mid (\exists x\varphi) \mid (\forall x\varphi)\end{gathered}$$

where R ranges over n-ary relation symbols, t_1, $t_2,\ldots,t_n$ over terms (for each integer $n \geq 0$), x over variables, and φ, φ_1, φ_2 over formulae. Formulae of the form $R(t_1,\ldots,t_n)$, $t_1 = t_2$, *true* or *false* are called *atoms* (a.k.a atomic formulae). The logical connectives for disjunction, conjunction, negation, and implication are as in propositional logic. Characteristic of first-order logic is *existential* ($\exists$) and *universal* ($\forall$) quantification with variables x ranging over a "universe of discourse". The parentheses are typically omitted based on simple precedence and associativity rules.

Next, one defines when a variable x *is free* (a.k.a. "occurs free") in a formula φ (inductively on the syntax of φ): (i) if α is an atom, then x is free in it iff it is used to build one of the terms that make up α; (ii) x is free in $(\varphi_1 \vee \varphi_2)$ or in $(\varphi_1 \wedge \varphi_2)$ or in $(\varphi_1 \rightarrow \varphi_2)$ iff it is free in φ_1 or in φ_2; (iii) x is free in (φ) iff it is free in φ; (iv) x is free in $(\exists y\, \varphi)$ or $(\forall y, \varphi)$ iff it is free in$\varphi$and $x \neq y$. Every formula has a finite set of variables that occur syntactically in it. Some of these are free, as defined above. The others are called *bound* because they must appear in some quantified subformula: $(\exists y\, \psi)$ or $(\forall y\, \psi)$, in which case it is said that ψ is the *scope* of the (bound) quantified variable y. For clarity, it is good practice to use fresh variables for each quantification. However, this is not required by the definition, for example

$$\begin{aligned}\exists x\, E(a,x) \wedge (\exists y\, E(x,y) \wedge (\exists x\, E(y,x) \\ \wedge (\exists y\, E(x,y) \wedge E(y,b))))\end{aligned}$$

says that there exists a path of length five from vertex a to vertex b in a directed graph with edge relation E, and it does so with only two bound variables although there are four intermediate vertices. Bound variable "reusing" can be exploited, see *finite variable logics* [3].

A formula without free variables is called a *sentence*. Sentences are logical assertions that may or may not be *true* by themselves. However, to give them meaning, one needs to give meaning more generally to formulae.

Still part of the syntax of FOL are the *proof systems* which describe effective procedures for deriving sentences from other sentences. A common style of proof system, due to Hilbert, uses sentences that are asserted as *axioms*, such as *tautologies* from propositional logic (eg., De Morgan's Laws) or the *substitution* axioms ($\forall x\varphi \rightarrow \varphi[x := t]$ where $\varphi[x := t]$ is the result of substituting every free occurrence of x in φ with the term t) and *inference rules* such as *modus ponens* (from φ and $\varphi\rightarrow \psi$ infer ψ). In such a system, a proof is a list of sentences such that each of them is either an axiom or is inferred by some rule from previously listed sentences. The proof is for the last sentence in the list. The sentences proved in a proof system are called *theorems*. More generally, a proof system defines a *provability* (a.k.a. inference or derivability) relationship between sentences φ and sets Γ of sentences: $\Gamma \vdash \varphi$ iff there exists a proof of φ that can use sentences in Γ as additional axioms (in particular, φ is a theorem iff $\emptyset \vdash \varphi$.

Quite a few styles of proof systems have been proposed, for FOL and for other logics, moreover the choice of axioms/rules often varies within each style. Of course, all the proof systems for FOL have been shown to be equivalent, that is, they define the same provability relationship, in particular the same theorems. More importantly, all proof systems are characterized by the following; (i) proofs are finite objects; (ii) it is decidable whether a proof is correctly formed; (iii) the proved sentence is totally computable from the proof. It follows that the theorems of FOL form a recursively enumerable set (although, by the result of Church and Turing, not a decidable one).

Cross-references

- ▶ First-Order Logic
- ▶ Semantics

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of databases: the logical level. Addison Wesley, Reading, MA, USA, 1994.
2. Enderton H.B. 1A Mathematical Introduction to Logic, Academic, London, 2000.
3. Libkin L. Elements of Finite Model Theory. Springer, Berlin, 2004.

First-Order Query

▶ Relational Calculus

Fisheye Views

▶ Distortion Techniques

Fixed Time Span

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Constant span

Definition

A time span is *fixed* if it possesses the special property that its duration is independent of the assumed context.

Key Points

As an example of a fixed span, "one hour" always, assuming a setting without leap seconds, has a duration of 60 minutes. To see that not all spans are fixed, consider "one month," which is a prime example of a variable span in the Gregorian calendar. The duration of this span may be any of 28, 29, 30, and 31 days, depending on the context, i.e., the specific month.

Cross-references

- ▶ Calendar
- ▶ Temporal Database
- ▶ Time Interval
- ▶ Time Span
- ▶ Variable Time Span

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer, Berlin Heidelberg New York, 1998. pp. 406–413.
2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer, Berlin Heidelberg New York, 1998, pp. 367–405.

Flajolet-Martin Algorithm

▶ FM Synopsis

Flajolet-Martin Sketch

▶ FM Synopsis

Flake

▶ Snippet

Flash

▶ Storage Devices

Flex Transactions

Aidong Zhang[1], Bharat Bhargava[2]
[1]State University of New York at Buffalo, Buffalo, NY, USA
[2]Purdue University, West Lafayette, IN, USA

Synonyms

Flexible transactions; S-transactions; ConTracts

Definition

In database systems, a *transaction* is a sequence of actions performed on data items in a database. In a distributed database environment, a *global transaction* is a set of subtransactions, where each *subtransaction* is a transaction accessing the data items at a single local site. The flex transaction model supports flexible execution control flow by specifying two types of dependencies among the subtransactions of a global transaction: (i) execution ordering dependencies between two subtransactions, and (ii) alternative dependencies between two subsets of subtransactions.

Key Points

Flexible transaction models, such as ConTracts, Flex Transactions, S-transactions, and others [1–3], increase the failure resilience of global transactions by allowing alternate subtransactions to be executed when a local database site fails or a subtransaction aborts. In a non-flexible transaction, a global subtransaction abort is followed either by a global transaction abort decision or by a retry of the global subtransaction. With the flexible transaction model, there is an additional option of switching to an alternate global transaction execution. The alternative global transaction execution is achieved by executing alternative or contingency transactions.

Flexibility allows a flexible transaction to adhere to a weaker form of atomicity, termed *semi-atomicity*, while still maintaining its correct execution in the distributed database environment. Semi-atomicity allows a flexible transaction to commit as long as a subset of its subtransactions that can represents the execution of the entire flexible transaction commit. Flexible transactions, S-tractions, and ConTracts are instances of Open-nested Transaction Model. The following example is illustrative:

A client at bank b_1 wishes to withdraw \$50 from her savings account a_1 and deposit it in her friend's checking account a_2 in bank b_2. If this is not possible, she will deposit the \$50 in her own checking account a_3 in bank b_3. With flexible transactions, this is represented by the following set of subtransactions:

t_1: Withdraw \$50 from savings account a_1 in bank b_1
t_2: Deposit \$50 in checking account a_2 in bank b_2
t_3: Deposit \$50 in checking account a_3 in bank b_3

In this global transaction, either $\{t_1,t_2\}$ or $\{t_1,t_3\}$ is acceptable, with t_3 being a contingency of t_2 but $\{t_1,t_2\}$ preferred. If t_2 fails, t_3 may replace t_2. The entire global transaction thus may not have to be aborted even if t_2 fails.

Cross-references

- ▶ Atomicity
- ▶ Concurrency Control Manager
- ▶ ConTract
- ▶ Distributed transaction management

Recommended Reading

1. Wächter H. and Reuter A. The ConTract model. In Database Transaction Models for Advanced Applications, A.K. Elmagarmid (ed.). Morgan Kaufmann, Los Altos, CA, 1992.
2. Zhang A., Nodine M., and Bhargava B. Global scheduling for flexible transactions in heterogeneous distributed database systems. IEEE Trans. Knowl. Data Eng., 13(3):439–450, 2001.
3. Zhang A., Nodine M., Bhargava B., and Bukhres O. Ensuring relaxed atomicity for flexible transactions in multidatabase systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 67–78.

Flexible Metric Computation

▶ Learning Distance Measures

Flexible Transactions

▶ Flex Transactions

Flowcharts

▶ Activity Diagrams

FM Sketch

▶ FM Synopsis

FM Synopsis

Phillip B. Gibbons
Intel Labs Pittsburgh, Pittsburgh, PA, USA

Synonyms

FM sketch; Flajolet-Martin sketch; Flajolet-Martin algorithm

Definition

Given a multi-set S of values from a domain $\mathcal{D}$, the *distinct-values estimation* problem is to estimate the number of distinct values in S, using only one pass over S and only small working space. The *FM Synopsis* algorithm, developed by Flajolet and Martin in the mid-1980s [7], provides provably accurate distinct-values estimation using only $O(\log(|\mathcal{D}|))$ space. The basic technique makes use of a hash function $h()$ that maps each value in $\mathcal{D}$ to one of $m \approx \log(|\mathcal{D}|)$ bit positions, according to a geometric distribution. Specifically, $h()$ maps half the values in $\mathcal{D}$ to position 0, one-quarter of the values in $\mathcal{D}$ to position 1, one-eighth of the values in to position 2, and so on. The steps of the FM Synopsis algorithm are:

1. Initialize a bit vector M of m bits to all 0s.
2. For each item in S do: Set $M[h(v)]$ to 1, where $v \in \mathcal{D}$ is the value of the item.
3. Estimate the number of distinct values as $2^Z/.77351$, where Z is the position of the least significant unset bit in M.

To reduce the variance in this estimator, Flajolet and Martin take the average over tens of applications of this procedure, with different random hash functions $h()$.

Historical Background

Distinct-values estimation was one of the first non-trivial data stream problems studied. A decade before data stream synopsis structures and their desirata were defined and popularized, the FM Synopsis met the desirata for a data stream synopsis structure: (i) it needs only one pass through the data, (ii) it yields highly accurate answers regardless of the data distribution, (iii) it takes only logarithmic space, and (iv) it requires only constant per-item processing time and only constant time to produce an estimate given the synopsis structure.

Prior to the popularization of the FM Synopsis algorithm, a common approach for estimating the number of distinct values in a multi-set S was to collect a random sample of S and then apply sophisticated estimators based on the distribution of the values in the sample [2]. However, all known sampling-based estimators provide unsatisfactory results on some data sets of interest, and moreover, estimating the number of distinct values within a small constant factor (with probability $> \frac{1}{2}$) requires (in the worst case) that *nearly all of S be sampled* [3]. Thus, streaming approaches, such as the FM Synopsis algorithm, are preferred over sampling-based approaches, whenever feasible.

Recent work has built upon the FM Synopsis approach, improving the accuracy guarantees on the estimation, proving lower bounds, and considering other settings such as sliding windows and distributed streams.

Foundations

Figure 1 depicts two data sets, S_1 and S_2, where the number of distinct values, also called the zeroth frequency moment F_0, is 15 for S_1 and 6 for S_2. These data sets can help illustrate the challenges of sampling-based estimators for F_0. Consider the following 33% sample of a data set of 24 items:

A, T, A, R, E, T, P, H

Given this 33% sample (with its 6 distinct values), does the entire data set have 6 distinct values, 18 distinct values (i.e., the 33% sample has 33% of the distinct values), or something in between? Note that this particular sample can be obtained by taking every third item of either S_1 (where $F_0 = 15$) or S_2 (where $F_0 = 6$) from Fig. 1. Thus, despite sampling a large (33%) percentage of the data, estimating F_0 remains challenging, because the sample can be viewed as fairly representative of either S_1 or S_2 – two data sets with very different F_0s.

Data set S_1: R, X, A, D, A, T, Y, R, A, A, T, R, T, U, E, A, C, T, F, L, P, B, V, H

Data set S_2: T, A, A, R, H, T, A, T, A, E, T, R, A, T, E, P, H, T, R, P, P, E, A, H

FM Synopsis. Figure 1. Two example data sets of $n = 24$ items from a domain $\mathcal{D} = \{A, B, \ldots, Z\}$.

Now consider the FM Synopsis algorithm outlined above. Using a hash function ensures that all items with the same value will select the same bit position; thus the final bit vector M is independent of any duplications among the item values. Flajolet and Martin's analysis assumes an idealized random hash function that maps each value $v \in \mathcal{D}$ to a bit position $b \in [0..m-1]$ such that $b = i \in [0..m-2]$ with probability $2^{-(i+1)}$ and $b = m-1$ with probability $2^{-(m-1)}$. Accordingly, $M[i]$ is expected to be set if there are at least 2^{i+1} distinct values. Because bit $Z-1$ is set but not bit Z, there are likely greater than 2^Z but fewer than 2^{Z+1} distinct values. Flajolet and Martin's analysis shows that $E[Z] \approx log_2(.77351 \cdot F_0)$, so that $2^Z/.77351$ is a good choice in that range.

To reduce the variance in the estimator, Flajolet and Martin take the average over tens of applications of this procedure (with different hash functions). Specifically, they take the average, $\bar{Z}$, of the Z's for different hash functions and then compute $\lfloor 2^{\bar{Z}}/.77351 \rfloor$. An example is given in Fig. 2.

The error guarantee, space bound, and time bound are summarized in Theorem 1. The space bound does not include the space for representing the hash functions, and the time bound assumes that computing $h(v)$ is a constant time operation.

Theorem 1 [7] *Consider a multi-set* S *of* n *items with values from a domain* $\mathcal{D}$. *The FM Synopsis algorithm with* k *(idealized) hash functions produces a distinct-values estimator with standard error* $O(1/\sqrt{k})$, *using* k · m *memory bits for the bit vectors, for any* $m > \log_2(\min(n, |\mathcal{D}|)) + 4$. *For each item, the algorithm performs* O(k) *operations on memory words of at most* $\max(m, \log_2(|\mathcal{D}|))$ *bits.*

Optimizations

The space needed for the k bit vectors, $M_1[\cdot], M_2[\cdot],\ldots, M_k[\cdot]$, can be significantly reduced, using the following simple compression technique [4,7,10]. Note that at any point during the processing of the data set, each bit vector $M_j[\cdot]$ consists of three parts, where the first part is all 0s, the second part (the *fringe*) is a mix of 0s and 1s, and the third part is all 1s (as in Fig. 2). Moreover, the bit vectors are likely to share many bits in common, because each $M_j[\cdot]$ is constructed using the same algorithm on the same data. The technique is to interleave the bits, as follows:

$$M_1[m-1], M_2[m-1], \ldots, M_k[m-1],$$
$$M_1[m-2], \ldots, M_k[m-2], \ldots, M_1[0], \ldots, M_k[0],$$

and then run-length encode the resulting bit vector's prefix of all 0s and suffix of all 1s. Specifically, a simple encoding scheme is used that ignores the all 0s prefix and consists of (i) the interleaved bits between the all 0s prefix and the all 1s suffix and (ii) a count of the length of the all 1s suffix. An example is given in Fig. 3. Note that the interleaved fringe starts with the maximum bit set among the k fringes and ends with their minimum unset bit. Each fringe is likely to be tightly centered around the current $\log_2(F_0)$, e.g., for a given hash function and $c > 0$, the probability that no bit than $\log_2(F_0)$ + c or larger is set is $(1 - 2^{-(\log_2(F_0)+c)})^{F_0} \approx e^{-1/2^c}$. At any point in the processing of the data set, a similar

Item number	Item value *v*	Hash function 1 b	Hash function 1 $M_1[\cdot]$	Hash function 2 b	Hash function 2 $M_2[\cdot]$	Hash function 3 b	Hash function 3 $M_3[\cdot]$
1.	R	1	000010	1	000010	0	000001
2.	X	0	000011	2	000110	0	000001
3.	A	0	000011	1	000110	0	000001
4.	D	0	000011	0	000111	0	000001
5.	A	0	000011	1	000111	0	000001
6.	T	1	000011	0	000111	1	000011
7.	Y	3	001011	0	000111	1	000011
8.	R	1	001011	1	000111	0	000011
			$Z = 2$		$Z = 3$		$Z = 2$

The estimate for F_0 is $\left\lfloor \frac{2^{(2+3+2)/3}}{.77351} \right\rfloor = 6.$

FM Synopsis. Figure 2. Example run of the FM Synopsis algorithm on the first 8 items of S_1, using $k = 3$ hash functions. Each bit vector $M[\cdot]$ is $m = 6$ bits long, with $M[0]$ being the rightmost bit (the least significant bit). The estimate, 6, matches the number of distinct values in the first 8 items of S_1.

argument shows that the interleaved fringe is expected to be fewer than $O(\mathrm{k} \log \mathrm{k})$ bits. Also, the count of the all 1s suffix is at most $k \cdot m$. Thus, the encoding is expected to use fewer than $\log_2 m + O(k \log k)$ bits.

The number of operations per data item can be reduced from $O(k)$ to $O(1)$, using a variant of the FM Synopsis algorithm called *Probabilistic Counting with Stochastic Averaging (PCSA)* [7]. In the PCSA algorithm (see Fig. 4), k bit vectors are used (for k a power of 2) but only a single hash function g(). The (idealized) random hash function g() is assumed to map each value $v \in \mathcal{D}$ to a uniformly random value in $\mathcal{D}$. For each data item with value v, the $\log_2 k$ least significant bits of $g(v)$ are used to select a bit vector. Then the remaining $m - \log_2 k$ bits of $g(v)$ are used to select a position within that bit vector, according to a geometric distribution (as in the basic FM Synopsis algorithm). To compute an estimate, PCSA averages over the positions of the least significant 0-bits, computes 2 to the power of that average, and divides by the bias factor .77351. To compensate for the fact that each bit vector has seen only $1/k$th of the distinct items on average, the estimate is multiplied by k.

The error guarantee, space bound, and time bound are summarized in the following theorem. The space bound does not include the space for representing the hash function. The time bound assumes that computing g and b are constant time operations. Although some of the operations use only m-bit words, the word size is dominated by the $\log_2 n$ bits for the item value v. The analysis assumes that the bit vectors are compressed using the optimization in Fig. 3.

Theorem 2 *Consider a multi-set S of n items with values from a domain $\mathcal{D}$. The PCSA algorithm with* k *bit vectors and an (idealized) hash function produces a distinct-values estimator with standard error* $0.78/\sqrt{k}$, *using an expected* $\log_2 m + O(k \log k)$ *memory bits for the bit vectors, for any* $m > \log_2(\min(n, |\mathcal{D}|)/k) + 4$. *For each item, the algorithm performs* $O(1)$ *operations on memory words of at most* $\max(m, \log_2(|\mathcal{D}|))$ *bits.*

Extensions

There has been considerable follow-on work since the introduction of the FM Synopsis algorithm. Some highlights are provided below; further details can be found in [8].

Alon, Matias and Szegedy [1] showed how a variant of the FM Synopsis algorithm could obtain guaranteed accuracy without the assumption of idealized hash functions. Specifically, they consider standard *linear hash functions* of the form $g(v) = a \cdot v + b$, where a and b are chosen uniformly and independently at random from a suitable range and all arithmetic is computed modulo that range. Their variant estimates F_0 as 2^R, where R is the position of the most significant bit that is set to 1. Moreover, it directly keeps track of the maximum position set to 1, instead of maintaining a bit vector.

Durand and Flajolet [5] presented a variant that uses only $\approx k \log_2 \log_2(\min(n, |\mathcal{D}|)/k)$ bits for its synopses and provides a standard error of $1.05/\sqrt{k}$, assuming idealized hash functions.

The FM Synopsis approach can be readily extended to handle item *deletions* by replacing each bit in $M[\cdot]$ with a running counter that is incremented on

```
     M_j[·]'s:   00001111, 00001011, 00000111, 00010111, 00011111, 00001111
  Interleaved:     000000000000000000000110110011101111111111111111
  End-encode:                        11011001110, 16
```

FM Synopsis. Figure 3. Example FM Synopsis compression, for $k = 6$ and $m = 8$.

```
1. for $j := 1,\ldots, k$ and $i := 0,\ldots,m - 1$ do $M_j[i] := 0$
2. for each item in $S$ do:
      $x := g(v) \bmod k$, where $v \in \mathcal{D}$ is the value of the item // Note: $k$ is a power of 2
      $b :=$ the largest $i \geq 0$ such that the $i$ rightmost bits in $\lfloor g(v)/k \rfloor$ are all 0
      $M_x[b] := 1$
3. return $\left\lfloor \frac{k}{.77351} 2^{\bar{Z}} \right\rfloor$, where $\bar{Z} := \frac{1}{k}\sum_{j=1}^{k} \min\{i : M_j[i] = 0\}$ // least significant unset bit
```

FM Synopsis. Figure 4. The PCSA algorithm [7].

insertions and decremented on deletions. This increases the synopsis space by a logarithmic factor. Similarly, the FM Synopsis approach can be extended to handle *sliding windows*, where the problem is to estimate the number of distinct values over a sliding window of the w most recent items, for some fixed w. The idea is to keep track of the sequence number of the most recent item that set each FM bit. Then, when estimating the number of distinct values within the current sliding window, only those FM bits whose associated sequence numbers are within the window are considered to be set. This too increases the synopsis space by a logarithmic factor. See [8] for an overview of algorithms that improve upon these basic schemes for deletions and sliding windows.

Finally, note that the FM Synopsis approach can be readily adapted to a *distributed* setting in which a set of observers each processes the portion of the multi-set S that it sees. The goal is to estimate the number of distinct values in S by (i) having each observer compute a small synopsis for the values it sees, and then (ii) combining the synopses to output a highly accurate estimate. The FM Synopsis algorithm is perfectly suited for this task. It can be applied to each data set portion independently, using the exact same hash function at all observers, to generate a bit vector $M[\cdot]$ for each portion. Because the same hash function is used by all observers, the bit-wise OR of their bit vectors yields the exact bit vector that would have been produced by running the FM Synopsis algorithm on any interleaving of the data portions. Thus, estimating F_0 from this bit-wise OR provides the same error guarantees as in the original algorithm. Moreover, the per-observer space bound and the per-item time bound also match the space and time bounds in Theorems 1 and 2.

Key Applications

Estimating the number of distinct values in a data set is a well-studied problem with many applications. The statistics literature refers to this as the problem of estimating the number of *species* or *classes* in a population. The problem has been extensively studied in the database literature, as a tool for summarizing the diversity of data values in a data set, both to help guide query optimization and to provide fast approximate answers to distinct-value queries (e.g., a select count(distinct) query in SQL). Distributed distinct-values estimators are useful for network resource monitoring, in order to estimate the number of distinct destination IP addresses, source-destination pairs, requested urls, etc. In network security monitoring, determining sources that send to many distinct destinations can help detect fast-spreading worms.

Distinct-values estimation can also be used as a general tool for duplicate-insensitive counting: Each item to be counted views its unique id as its "value," so that the number of distinct values equals the number of items to be counted. Duplicate-insensitive counting can be used to avoid double-counting items that are counted while in motion, to compute the number of distinct neighbors at a given hop-count in a network, and to compute the size of the transitive closure of a graph. In a sensor network, duplicate-insensitive counting together with multi-path in-network aggregation enables robust and energy-efficient answers to count queries [9]. Moreover, duplicate-insensitive counting is a building block for duplicate-insensitive computation of other aggregates, such as sum and average.

Experimental Results

The number of bit vectors, k, in the FM Synopsis algorithm is a tunable parameter trading off space for accuracy. The relative error as a function of k has been studied empirically in [4,7,9,10] and elsewhere, with $< 15\%$ relative error reported for $k = 20$ and $<10\%$ relative error reported for $k = 64$, on a variety of data sets. These studies show a strong diminishing return for increases in k. Theorems 1 and 2 show that the standard error is $O(1/\sqrt{k})$. Thus, reducing the standard error from 10% to 1% requires increasing k by a factor of 100! In general, to obtain a standard error at most ε it is required that $k = \Theta(1/\varepsilon^2)$. Estan, Varghese and Fisk [6] present a number of techniques for further improving the constants in the space versus error trade-off, including using multi-resolution and adaptive bit vectors.

Cross-references

- ▶ Approximation and Data Reduction Techniques
- ▶ Stream Mining
- ▶ Synopsis Structure

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. J. Comput. Syst. Sci., 58:137–147, 1999.
2. Bunge J. and Fitzpatrick M. Estimating the number of species: a review. J. Am. Stat. Assoc., 88:364–373, 1993.

3. Charikar M., Chaudhuri S., Motwani R., and Narasayya V. Towards estimation error guarantees for distinct values. In Proc. 19th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2000, pp. 268–279.
4. Considine J., Li F., Kollios G., and Byers J. Approximate aggregation techniques for sensor databases. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 449–460.
5. Durand M. and Flajolet P. Loglog counting of large cardinalities. In Proc. 11th European Symposium on Algorithms. 2003, 604–617.
6. Estan C., Varghese G., and Fisk M. Bitmap algorithms for counting active flows on high speed links. In Proc. 3rd ACM SIGCOMM Conf. on Internet Measurement. October, 2003, pp. 153–166.
7. Flajolet P. and Martin G.N. Probabilistic counting algorithms for data base applications. J. Comput. Syst. Sci., 31:182–209, 1985.
8. Gibbons P.B. *1*Distinct-values estimation over data streams. In Data Stream Management: Processing High-Speed Data Streams, M. Garofalakis, J. Gehrke, R. Rastogi (eds.). Springer, Secaucus, NJ, USA, 2009.
9. Nath S., Gibbons P.B., Seshan S., and Anderson Z. Synopsis diffusion for robust aggregation in sensor networks. ACM Trans. on Sensor Networks, 4(2), 2008.
10. Palmer C.R., Gibbons P.B., and Faloutsos C. ANF: a fast and scalable tool for data mining in massive graphs. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 81–90.

F-Measure

Ethan Zhang[1,2], Yi Zhang[1]
[1]University of California-Santa Cruz, Santa Cruz, CA, USA
[2]Yahoo! Inc., Santa Clara, CA, USA

Synonyms

Harmonic mean of recall and precision

Definition

Assume an information retrieval (IR) system has recall R and precision P on a test document collection and an information need. The *F-measure* of the system is defined as the weighted harmonic mean of its precision and recall, that is, $F = \frac{1}{\alpha\frac{1}{P}+(1-\alpha)\frac{1}{R}}$, where the weight $\alpha \in [0,1]$. The balanced F-measure, commonly denoted as F_1 or just F, equally weighs precision and recall, which means $\alpha = 1/2$. The F_1 measure can be written as $F_1 = \frac{2PR}{P+R}$.

Key Points

The F-measure can be viewed as a compromise between recall and precision. It is high only when both recall and precision are high. It is equivalent to recall when $\alpha = 0$ and precision when $\alpha = 1$. The F-measure assumes values in the interval [0,1]. It is 0 when no relevant documents have been retrieved, and is 1 if all retrieved documents are relevant and all relevant documents have been retrieved.

Cross-references

- ▶ Precision
- ▶ Recall
- ▶ Standard Effectiveness Measures

Focused Retrieval

- ▶ XML Retrieval
- ▶ Structured Document Retrieval

Focused Web Crawling

Soumen Chakrabarti
Indian Institute of Technology of Bombay, Mumbai, India

Synonyms

Web resource discovery; Topic-directed Web crawling

Definition

The world-wide Web can be modeled as a very large graph with nodes representing pages and edges representing hyperlinks. Thanks to dynamically generated content, the Web graph is infinitely large. Page content and hyperlinks change continually. Any centralized Web search service must first fetch a large number of Web pages over the Internet using a Web crawler, and then subject the local copies to indexing and other analysis. At any time during its execution, a Web crawler has a set of pages that have been fetched, and a frontier of unexplored hyperlinks encountered on fetched pages. Given finite network resources, it is critical for the crawler to choose carefully the subset of frontier hyperlinks it should fetch next. Depending on the application and user group, it may be beneficial to preferentially acquire pages that are highly linked, pages that pertain to specific topics, pages that are likely to mention specific structured information, pages that score highly with respect to queries submitted frequently to the search engine, pages that change frequently, and so on. Focused Web crawling is a generic term for employing hyperlink and text mining techniques to prioritize the crawl frontier to maximize the harvest of qualified or preferred pages, while minimizing communication and computation effort on other pages. The network resources thus saved may be used, for example, to monitor crawled pages more aggressively for changes. Focused Web crawling is commonly used to build vertical search services catering to one or few topical interests.

Historical Background

Despite giant leaps in communication, storage and computing power during the last decade, crawlers have always struggled to keep up with Web content generation and modification. The Web graph is actually infinite, owing to dynamically generated hyperlinks and pages. A crawler begins with URL references to a fraction of nodes in the Web graph. Many of these URLs are stale because the pages they referred to are inaccessible. The number of pages collected by the crawler makes little sense in this context, compared to which specific pages are collected.

Around 1997, researchers began to propose objectives that a crawler may seek to optimize, and several forms of crawl prioritization. Some objectives [8,14] were related to the Web graph structure. Others were determined by textual content of pages [4,12]. The term "focused crawling" is generally used for the latter class, although later generations of focused crawlers use all available features and clues [7,10,17,] to guide themselves, including workloads collected from query logs [15]. As of late 2007, work on prioritizing crawls and crawl refreshing continues apace, with increased stress on balancing resources between monitoring crawled pages for changes and discovering new sites and pages.

The success of focused Web crawling depends on benign patterns and regularities in the Web graph, which started to get seriously investigated as social network phenomena around 1999. The Web as a whole and many major communities have a strongly connected core of authoritative and popular pages [2,11]. Well-connected subgraphs of the Web are also likely to be topically coherent [6,9]. Without these helpful properties, focused crawlers cannot succeed.

Foundations

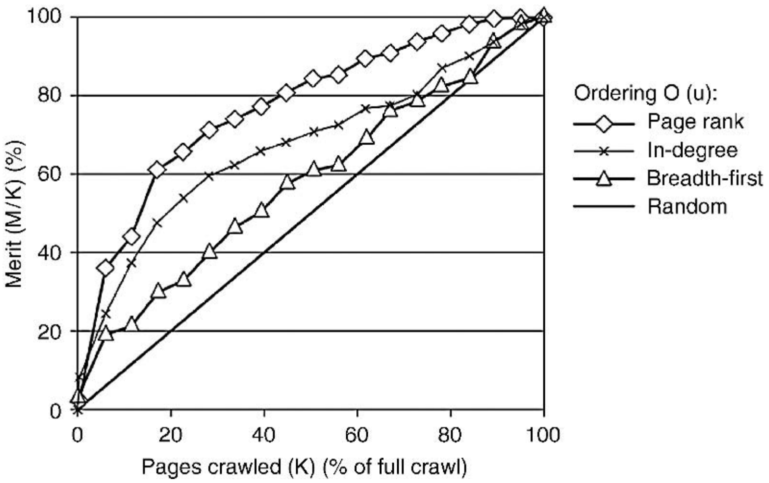
Setting Crawl Priorities Using Web Graph Properties

Initial work on focused Web crawling set reasonable fixed policies and evaluated their effect on various measures of crawl quality [8]. To characterize crawl quality, each page u is assigned an *importance* $I(u)$. For simplicity, one can initially assume that the crawler starts from just one URL u_0, and stops when K pages have been fetched. The perfect crawler collects K of the N pages reachable from u_0 by following hyperlinks with the largest possible importance. Let these "hot pages" have importance scores $I(u_1) \leq ... \leq I(u_K)$. An imperfect crawler will also fetch K pages, but only M of those will have importance at least $I(u_k)$. Then the figure of merit of the crawler will be M/K. As a baseline, if a "random" crawler could somehow collect K random samples from the N reachable pages without following links, the expected number of hot pages in the sample would be MK/N leading to a figure of merit equal to K/N.

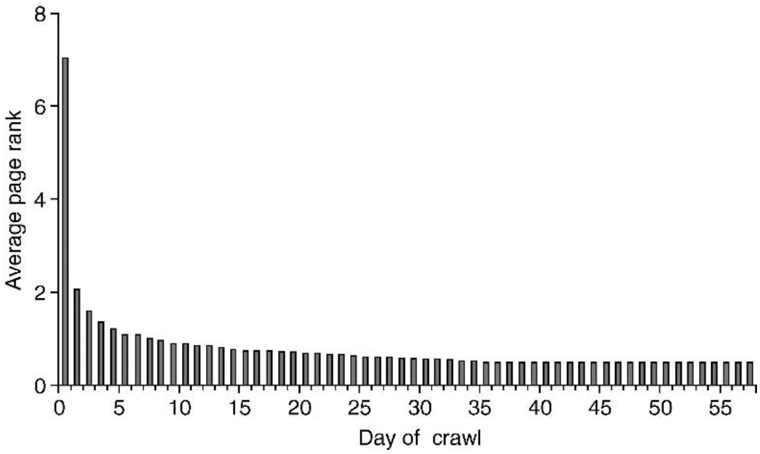
Suppose $I(u)$ is the indegree of u in the "whole" Web graph (suppose some arbitrary subset of the infinite graph is fixed a-priori as the "universe"). In this case, $I(u)$ can be known only after crawling the universe. Another global property is PageRank [16]. At any time, a real crawler only has access to the subgraph crawled thus far. It may use the indegree and PageRank computed on the crawled subgraph as approximations to the true indegree and PageRank.

Figure 1 [8] shows M/K as a percent plotted against K/N also plotted as a percent. The random crawler is understandably the worst, and all the other crawlers fare better. Somewhat surprisingly, the best crawler is the one that is guided not by indegree but PageRank computed with respect to the currently crawled subgraph. A later breadth-first crawl [14] starting from http://www.yahoo.com, covering 7 million hosts and 328 million pages, confirmed that pages with large true PageRank are acquired very quickly using the breadth-first policy (see Fig. 2).

Cho et al. [8] also considered page importance defined by the occurrence of query words on the page, and showed that favoring URLs v that were frequently referred from pages u with query words in or near the anchor text of the (u, v) link resulted in



Focused Web Crawling. Figure 1. Comparison of harvest rates of large indegree pages using various frontier prioritization policies [8].



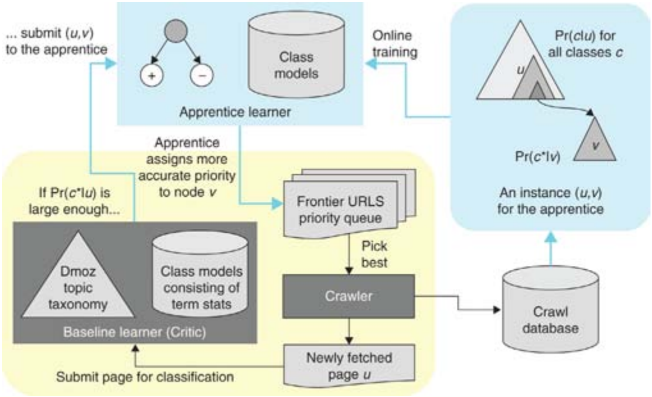
Focused Web Crawling. Figure 2. Page fetched early on in a breadth-first crawl from well-known seed URLs tend to have high PageRank [14].

improved acquisition rate for relevant pages. Similar results were reported by Hersovici et al. [12].

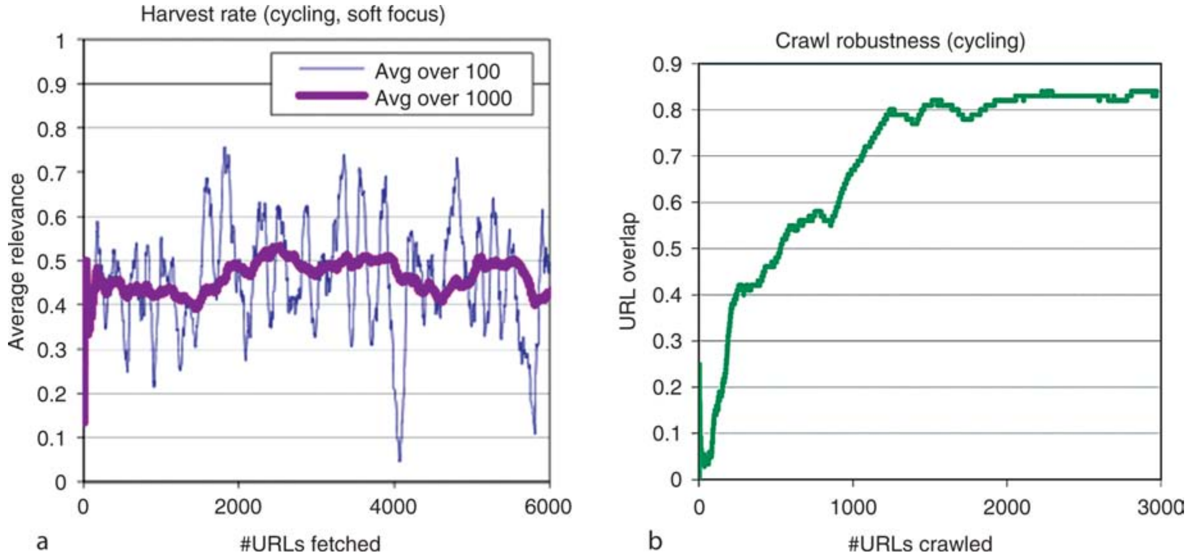
Basic Topic-Focused Crawler

During 1997–1998, document classification witnessed renewed interest and enhancements from the use of hyperlinks [5]. Rather than use occurrence of a few query words as an indicator of payoff during a crawl, Chakrabarti et al. [4] coupled a pre-trained text classifier with a crawler as shown in Fig. 3. The classifier may be trained on a topic taxonomy such as the one published by Yahoo! or the Open Directory Dmoz, and a few topics (nodes) c^* in the taxonomy (and all their descendants, by inheritance) flagged as positive or rewarding for the crawler.

This organization depends on the (verified [9,6]) premise that pages within a short link distance of each other are topically similar or related. In particular, if $\Pr(c^*|u)$ is large and hyperlink (u, v) exists, then the hypothesis is that $\Pr(c^*|v)$ is also likely to be large quite often. For reasonably broad topics about which there are sufficiently dense link communities on the Web, this hypothesis holds up well and leads to reliable rates of page collection. Some sample results are shown in Fig. 4.


... submit (u,v)
to the apprentice
+
−
Class models
Apprentice learner
Online training
Pr(c|u) for all classes c
u
v
Pr(c*|v)
An instance (u,v) for the apprentice
Apprentice assigns more accurate priority to node v
If Pr(c*|u) is large enough...
Frontier URLS priority queue
Pick best
Dmoz topic taxonomy
Class models consisting of term stats
Baseline learner (Critic)
Crawler
Crawl database
Newly fetched page u
Submit page for classification

Focused Web Crawling. Figure 3. Basic topic-focused crawler [3].

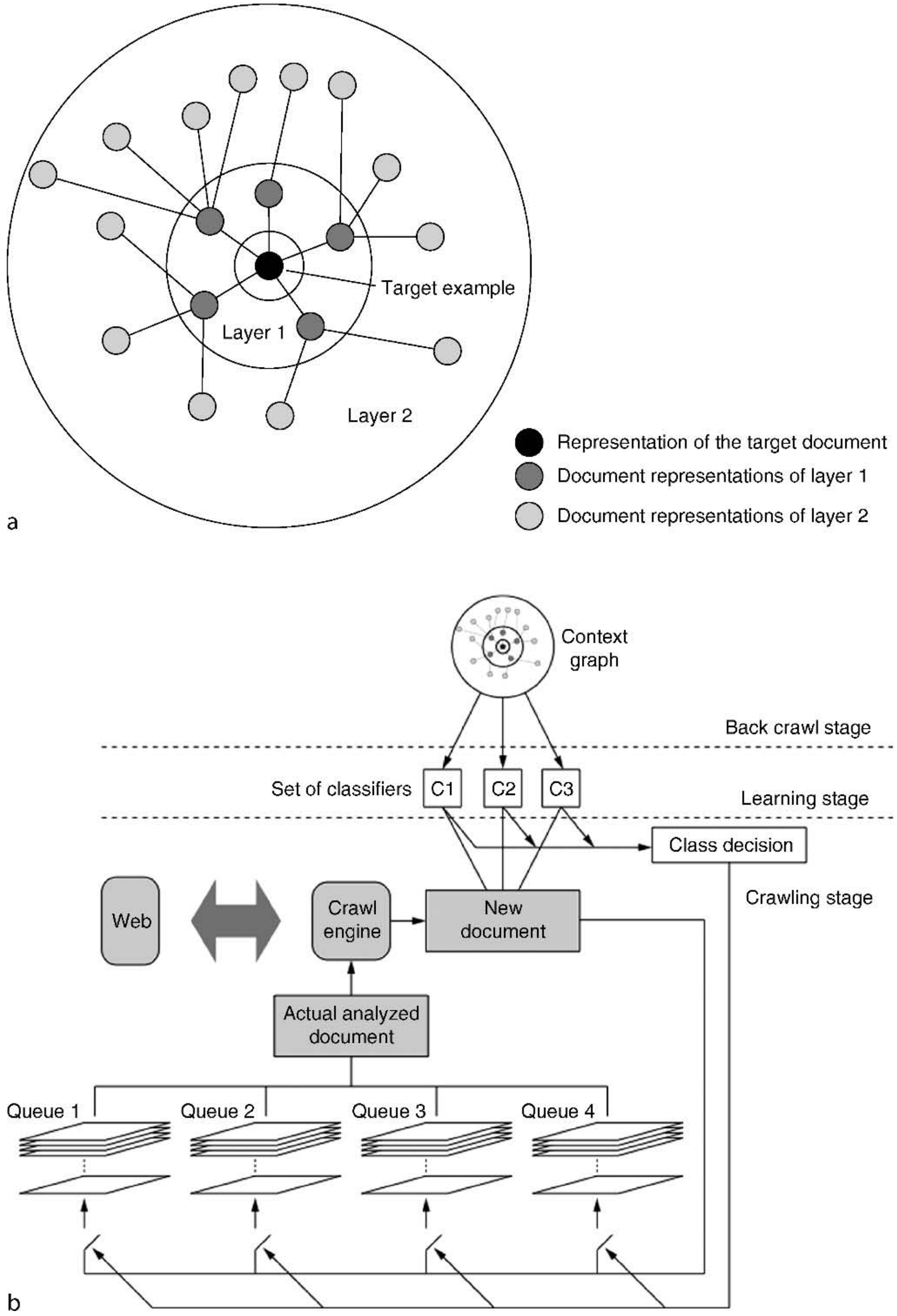


Focused Web Crawling. Figure 4. (a) The basic focused crawler manages to keep up a reasonable "harvest rate" of collecting relevant pages. (b) Started from different seed URLs, the focused crawler navigates to the dominant communities on the focus topic and visits largely overlapping sites and pages [4].

Focused Crawling Using Context Graphs

The next enhancement to focused crawling was the use of *context graphs* [10]. The first generation of topic-focused crawlers was myopic: the relevance of page u was used as a surrogate for the benefits of crawling outlink (u, v), and this prediction was limited to only one hop. In Reinforcement Learning [18], an area of Machine Learning, techniques have been evolved to anticipate more distant payoffs. In focused crawling with context graphs, during the training phase, the trainer presents to the system paths $u_1,...,u_g$ of various lengths (up to some modest limit, say, four hops) leading to goal nodes. The goal node u_g is a node on the topic/s of focus. The system includes a supervised learning module that then learns to predict, given a node u, the (minimum) distance to a goal node. When the trained crawler is deployed, after fetching page/node u, it estimates the shortest link distance from u to a goal node, and uses this as the priority for expanding outlinks of u; see Fig. 5.

Figure 6 shows the benefits from going beyond a greedy one-hop prediction paradigm. A similar technique

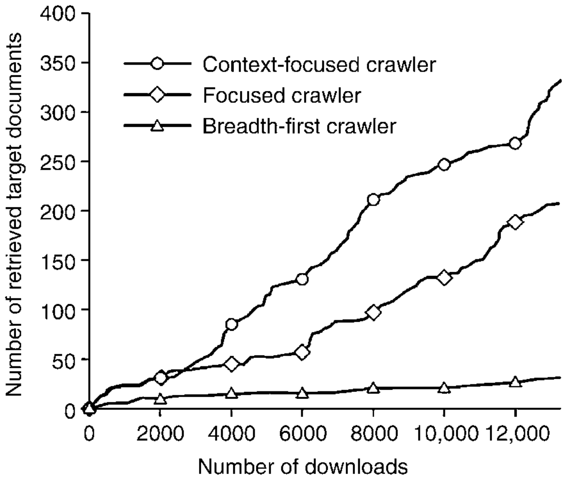


Focused Web Crawling. Figure 5. Focused crawling using context graph [10].

has been implemented by McCallum et al. [17]. More recently, Baberia et al. [1] have cast the problem of estimating the distance to a goal node as an ordinal regression, and given scalable and effective algorithms.

Training a Focused Crawler Online

The focused crawlers discussed thus far are all trained once, off-line, and then start harvesting the Web. The next generation of focused crawlers [7] "learn on the job." Compare Fig. 7 with the basic focused crawler shown in Fig. 3. In the new edition, once a page *u* is approved by the baseline topic classifier for getting its outlinks visited, each outlink (*u*, *v*) is submitted to an *apprentice*. Initially, the apprentice is inexperienced, and might assign the same priority to each outlink (*u*, *v*) that the baseline learner does. As pages continue to be fetched, the baseline classifier classifies each of them, and the apprentice "watches." The baseline classifier acts as a *critic*, pointing out where the apprentice was right and wrong. The apprentice uses features not only from the whole of *u*, but also features specific to the outlink (*u*,*v*), exploiting the DOM tree of *u*.

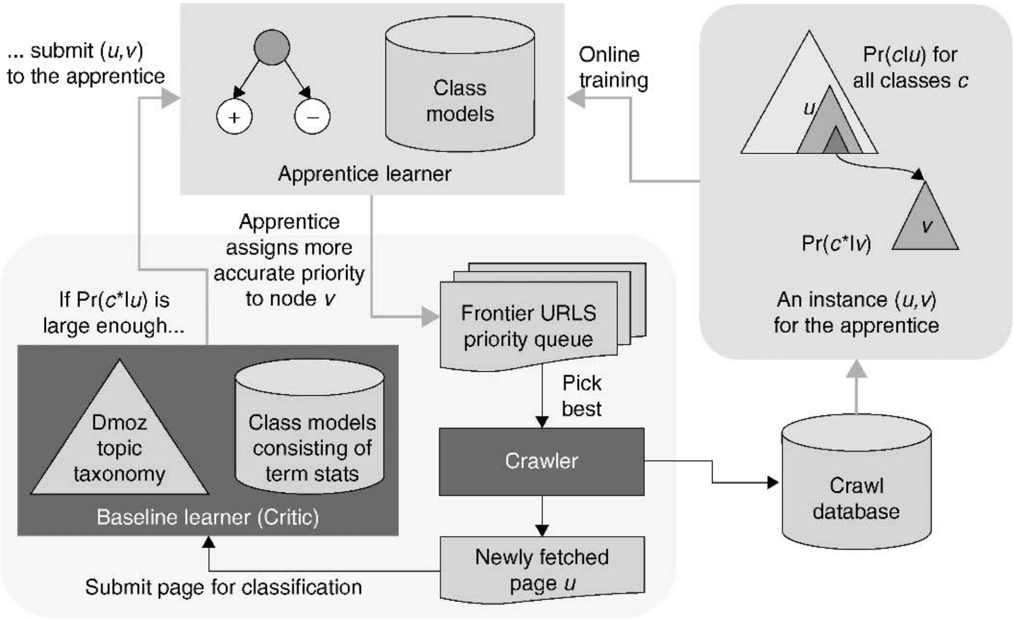


Focused Web Crawling. Figure 6. Context graphs increase harvest of relevant pages [10].

Gradually, the apprentice learns to predict, over and above the critic's relevance judgment of *u*, the expected profit from traversing link (*u*,*v*). Continually training the apprentice is much easier than training the baseline classifier. The apprentice can be a simple regression algorithm, whereas the baseline classifier has to handle much more data and many topics. As the apprentice is switched into the decision process, the number of fetched pages that must be discarded because they are off-topic, also called the "loss rate," drops dramatically. This is shown in Fig. 8a. Overall, as Fig. 8b shows, a focused crawler that learns online can significantly reduce the loss rate and thus boost the harvest rate.

Reinforcement Learning Using Markov Models

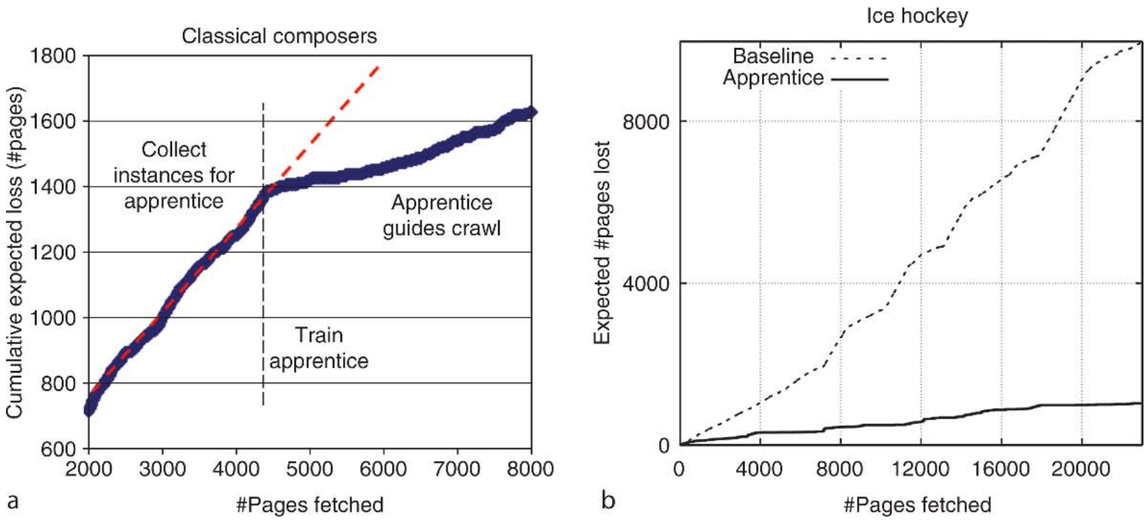
The next step in improving the ability of focused crawlers to spot distant rewards was taken by learning a more detailed representation of the path from the current node to a goal node, rather than just the estimated distance [19]. This is expected to be effective especially for crawling tasks where paths leading into goal nodes have some regularity. For example, paths leading from the root page of a Computer Science



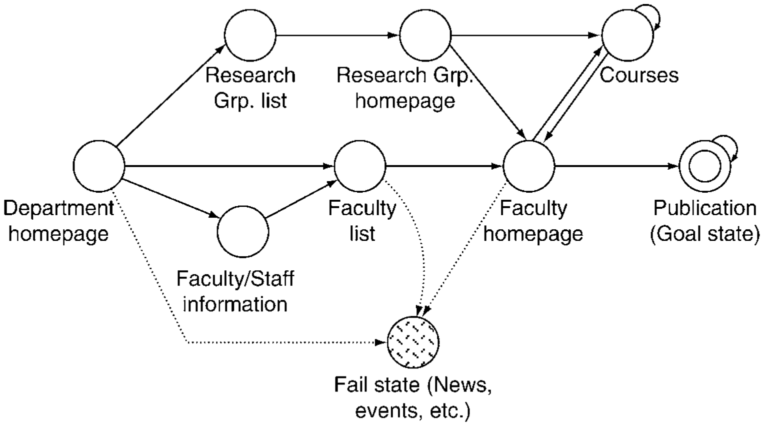
Focused Web Crawling. Figure 7. Topic-focused crawler that learns online during a crawl [7].

Department to research papers about computer vision, or paths leading from the root page of a company to a set of profiles of its top officers, are fairly regular. In fact, one can even give informal but crisp descriptions of the intermediate pages. In the first example, the intermediate pages will most likely be a roster of faculty members with research interests, followed by a faculty homepage, which either has links to PDF files, or links to a page listing publications, which then links to PDF files. In the second example, a surfer is most likely to navigate through an "about us" page, which may list the top office-bearers, each with a link to a profile page, or the profiles may be inlined. An example state transition diagram is given in Fig. 9.

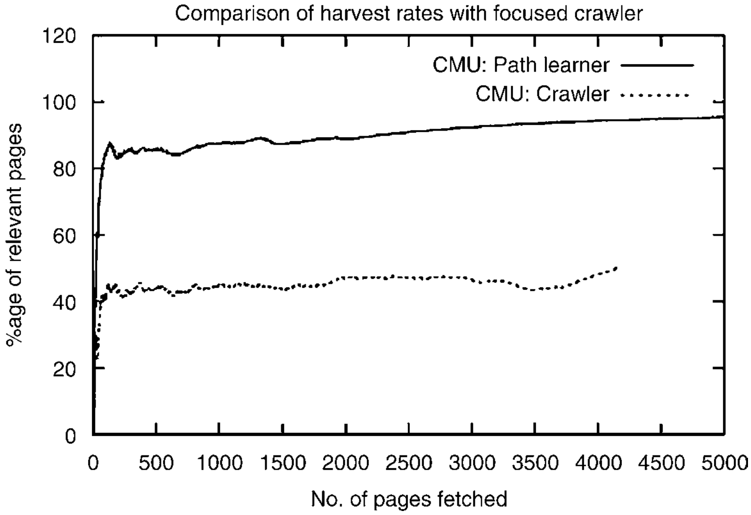
A hidden Markov model (HMM) is ideal for capturing this form of regularity. For label prediction in HMMs, Conditional Random Fields (CRFs) [13] are the best-known technique. Training the CRF involves slightly more work than in the case of context graphs: here, intermediate nodes must be assigned labels. Once this is done, the problem of learning a transition model is very similar to other sequence-labeling applications, like part-of-speech tagging (nodes are words, labels are parts of speech) and named-entity recognition (nodes are words, labels are named entity types like *person* or *location*). Over and above the general framework, smart state and feature design are a must for good accuracy. Also, during crawling, the crawl path up to



Focused Web Crawling. Figure 8. A focused crawler that "learns on the job" improves harvest rate significantly [7].



Focused Web Crawling. Figure 9. Example state transitions modeling paths from a department homepage to research papers [19].



Focused Web Crawling. Figure 10. Using a CRF to predict the expected reward continuing from the current path leads to significantly improved harvest [8].

the current node is assigned various label sequences with various probabilities, and these must contribute to a distance-discounted reward function as in reinforcement learning [18]. For crawling tasks with path regularities, this form of reward prediction is more accurate than the standard topic-focused crawler, leading to substantially improved page harvest rates; see Fig. 10.

Key Applications

Focused crawling, broadly interpreted as goal-directed crawl prioritization with resource constraints, is of interest to almost anyone that has to run a crawler of any substantial scale. All crawlers include a module for frontier prioritization, but only some degree of machine learning usually qualifies it to be explicitly called a "focused crawler." Focused crawling is especially beneficial in applications where content pertaining to a limited topic area must be collected from a much larger Web collection, possibly the whole Web, and subjected to topic-specific post-processing, such as information extraction specific to particular people, regions, languages, or products and services.

Future Directions

Focused crawlers are now available in the public domain (see http://ivia.ucr.edu/ and http://combine.it.lth.se/). Many companies in the vertical Web search space implement focused crawlers. While many of the technical issues around focused crawling are now reasonably solved, the operational details of starting, monitoring, and maintaining a focused crawler remain relatively unknown in the public domain.

URL to Code

A public domain focused crawler implementation is available from http://ivia.ucr.edu/ under the Limited Gnu Public License. Another public domain version is available from the ALVIS project, see http://combine.it.lth.se/. Also see the Wikipedia page on Focused Crawling at http://en.wikipedia.org/wiki/Focused_crawler.

Cross-references

- ▶ Classification
- ▶ Digital Libraries
- ▶ Document Databases
- ▶ Document Links and Hyperlinks
- ▶ Graph Database
- ▶ Incremental
- ▶ Indexing the Web
- ▶ Information Retrieval
- ▶ Personalized Web Search
- ▶ Relevance Feedback
- ▶ Social Networks
- ▶ Text Categorization
- ▶ Text Mining
- ▶ Web Characteristics and Evolution
- ▶ Web Crawler Architecture
- ▶ Web Crawler
- ▶ Web Harvesting

Recommended Reading

1. Babaria R., Saketha Nath J., Krishnan S., Sivaramakrishnan K.R., Bhattacharyya C., and Murty M.N. Focused crawling with scalable ordinal regression solvers. In Proc. 24th Int. Conf. on Machine Learning, 2007, pp. 57–64.
2. Broder A. et al. Graph structure in the Web: experiments and models. In Proc. 9th Int. World Wide Web Conference, 2000, pp. 309–320.
3. Chakrabarti S. Mining the Web: Discovering Knowledge from Hypertext Data, Morgan-Kauffman, 2002.
4. Chakrabarti S., van den Berg M., and Dom B. Focused crawling: a new approach to topic-specific Web resource discovery. Comput. Netw., 31:1623–1640, 1999.
5. Chakrabarti S., Dom B., and Indyk P. Enhanced hypertext categorization using hyperlinks. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 307–318.
6. Chakrabarti S., Joshi M.M., Punera K., and Pennock D.M. The structure of broad topics on the Web. In Proc. 11th Int. World Wide Web Conference, 2002, pp. 251–262.
7. Chakrabarti S., Punera K., and Subramanyam M. Accelerated focused crawling through online relevance feedback. In Proc. 11th Int. World Wide Web Conference, 2002, pp. 148–159.
8. Cho J., Garcia-Molina H., and Page L. Efficient crawling through URL ordering. In Proc. 7th Int. World Wide Web Conference, 1998, pp. 161–172.
9. Davison B.D. Topical locality in the Web. In Proc. 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2000, pp. 272–279.
10. Diligenti M., Coetzee F., Lawrence S., Giles C.L., and Gori M. Focused crawling using context graphs. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 527–534.
11. Dill S., Ravi Kumar S., McCurley K.S., Rajagopalan S., Sivakumar D., and Tomkins A. Self-similarity in the Web. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 69–78.
12. Herseovici M., Jacovi M., Maarek Y.S., Pelleg D., Shtalhaim M., and Ur S. The shark-search algorithm – an application: tailored Web site mapping. In Proc. 7th Int. World Wide Web Conference, 1998, pp. 317–326.
13. Lafferty J., McCallum A., and Pereira F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proc. 18th Int. Conf. on Machine Learning, 2001, pp. 282–289.
14. Najork M. and Weiner J. Breadth-first search crawling yields high-quality pages. In Proc. 10th Int. World Wide Web Conference, 2001, pp. 114–118.
15. Pandey S. and Olston C. User-centric Web crawling. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 401–411.
16. Page L., Brin S., Motwani R., and Winograd T. The PageRank citation ranking: bringing order to the Web. Manuscript, Stanford University, 1998.
17. Rennie J. and McCallum A. Using reinforcement learning to spider the web efficiently. In Proc. 16th Int. Conf. on Machine Learning, 1999, pp. 335–343.
18. Sutton R.S. and Barto A.G. Reinforcement Learning: An Introduction. MIT, March 1998.
19. Vinod Vydiswaran V.G. and Sarawagi S. Learning to extract information from large Websites using sequential models. In Proc. 11th Int. Conf. on Management of Data, 2005, pp. 3–14.

Focus-Plus-Context

▶ Distortion Techniques

FOL

▶ First-Order Logic: Semantics
▶ First-Order Logic: Syntax

FOL Modeling of Integrity Constraints (Dependencies)

Alin Deutsch
University of California-San Diego, La Jolla, CA, USA

Synonyms

Relational integrity constraints; Dependencies

Definition

Integrity constraints (also known as *dependencies* in the relational model) are domain-specific declarations which indicate the intended meaning of the data stored in a database. They complement the description of the structure of the data (e.g., in the relational model the structure is given by listing the names of tables and the names and types of their attributes). Integrity constraints express properties that must be satisfied by all instances of a database schema that can arise in the intended application domain (e.g., "no two distinct employees may have the same ssn value", "departments have a single manager", etc.).

Historical Background

The reference textbook [1] provides a comprehensive, unifying overview of the many special classes of relational dependencies, their modeling in first-order logic (FOL), and the key problems in the study of dependencies. This entry is a condensed form of Chap. 10 in [1] (excluding the material on reasoning about dependencies). For more detail on the motivation and history of relational dependency theory, see the excellent survey papers [10,14,22].

Historically, *functional* dependencies were the first class to be introduced (by Codd [3]). *Multi-valued* dependencies were discovered independently in [5,9,24]. These were followed by a proliferation of dependency classes, typically expressed using ad hoc syntax. These include *join* dependencies, and *inclusion* dependencies (a.k.a. referential integrity constraints) [4]. Fagin [9] first introduced *embedded multi-valued* dependencies (MVDs that hold in the projection of a relation), while [15] introduced the distinct class of *projected* JDs. Other classes include *subset* dependencies [20], *mutual* dependencies [17], *generalized mutual* dependencies [16], *transitive* dependencies [18], *extended transitive* dependencies [19], and *implied* dependencies [12].

The movement towards ever-refined classifications of dependencies was prompted mainly by research on automatic reasoning about dependencies, in particular their axiomatization. This movement was soon countered by the drive to unify the treatment of the various classes by finding a formalism that subsumes all of them. Nicolas [17] is credited with first observing that FDs, MVDs and others have a natural representation in first-order logic. In parallel, Beeri and Vardi [2] introduced *tuple-generating* and *equality-generating* dependencies expressed in a tableaux-based notation, shown to be equivalent in expressive power to Fagin's *typed embedded dependencies* [8], which were expressed in first-order logic. The class DED$^{\neq}$ was introduced in [7] as an extension of embedded dependencies with disjunction and non-equalities (see also [13] for a particular case of DED$^{\neq}$s). Deutsch et al. [6] further extends the DED$^{\neq}$ class to allow negated relational atoms. Research on using arbitrary first-order logic sentences to specify constraints includes [11,17,21].

In contrast, *algebraic dependencies* are an alternative unifying framework developed by Yannakakis and Papadimitriou [23]. Algebraic dependencies are statements of containment between queries written in relational algebra, and have the same expressive power as first-order logic.

Foundations

This section lists a few fundamental classes of relational dependencies, subsequently showing how they can be expressed in first-order logic. In the following, $R(U)$ denotes the schema of a relation with name R and set of attributes U.

Functional Dependencies

A *functional dependency (FD)* on relations of schema $R(U)$ is an expression of the form

$$R : X \rightarrow Y, \tag{1}$$

where $X \subseteq U$ and $Y \subseteq U$ are subsets of R's attributes. Instance r of schema $R(U)$ is said to *satisfy* FD fd, denoted $r \models fd$, if whenever tuples $t_1 \in r$ and $t_2 \in r$ agree on all attributes in X, they also agree on all attributes in Y:

$$r \models fd \iff \text{for every } t_1, t_2 \in r \text{ if } \pi_X(t_1) = \pi_X(t_2) \text{ then } \pi_Y(t_1) = \pi_Y(t_2).$$

Here, $\pi_X(t)$ denotes the projection of tuple t on the attributes in X.

For instance, consider a relation of schema

review(paper, reviewer, track)

listing a conference track a paper was submitted to, and a reviewer it was assigned to. The fact that every paper can be submitted to a single track is stated by the functional dependency

$$review : \ paper \longrightarrow track.$$

Key Dependencies

In the particular case when $Y = U$, a functional dependency of form (1) is called a *key dependency*, and the set of attributes X is a called a *key for R*.

Join Dependencies

A *join dependency (JD)* on relations of schema $R(U)$ is an expression of the form

$$R : \bowtie [X_1, X_2, \ldots, X_n], \tag{2}$$

where for each $1 \leq i \leq n$, $X_i \subseteq U$, and $\bigcup_{1 \leq i \leq n} X_i = U$. Instance r of schema $R(U)$ *satisfies* JD jd, denoted $r \models jd$, if the n-way natural join of the projections of r on each of the attribute sets X_i yields r:

$$r \models jd \Leftrightarrow r = \Pi_{X_1}(r) \bowtie \Pi_{X_2}(r) \bowtie \ldots \bowtie \Pi_{X_n}(r).$$

Here, $\Pi_X(r)$ denotes the projection of relation r on the attributes in X.

In the example, assume that a paper may be submitted for consideration by various tracks (e.g., poster or full paper), and that reviewers are not tied to the tracks. It makes sense to expect that for any given paper p, any track information listed with a reviewer of p is also listed with all other reviewers of p, since track and reviewer information are not correlated. This is expressed by requiring that the join of the projection of *review* on *paper, track* and of the projection on *paper, reviewer* yields the *review* table:

$$review : \bowtie [\{paper,\ track\}, \{paper,\ reviewer\}].$$

Multi-Valued Dependencies

In the particular case when $n = 2$, a join dependency of form (2) is called a *multi-valued dependency (MVD)*. Because MVDs were historically introduced and studied before JDs, they have their own notation: an MVD $R : \bowtie [X_1, X_2]$ is denoted

$$R: \ X \twoheadrightarrow Y, \tag{3}$$

where $X = X_1 \cap X_2$ and $Y = X_1 \setminus X_2$.

In the running example, the join dependency turns out to be a multi-valued dependency, which can be expressed using the following MVD-specific syntax:

$$review: \ paper \twoheadrightarrow track.$$

Inclusion Dependencies

Functional and join dependencies and their special-case subclasses each pertain to single relations. The following class of dependencies can express connections between relations. An *inclusion dependency (IND)* on pairs of relations of schemas $R(U)$ and $S(V)$ (with R and S not necessarily distinct) is an expression of the form

$$R[X] \subseteq S[Y], \tag{4}$$

where $X \subseteq U$ and $Y \subseteq V$. Inclusion dependencies are also known as *referential constraints*. Relations r and s of schemas $R(U)$, respectively $S(V)$ satisfy inclusion dependency id, denoted $r, s \vDash id$, if the projection of r on X is included in the projection of s on Y:

$$r, s \models id \Leftrightarrow \ \Pi_X(r) \subseteq \Pi_Y(s).$$

When R and S refer to the same relation name, then $r = s$ in the above definition of satisfaction.

In the running example, assume that the database contains also a relation of schema

$$PC(member, \ affiliation)$$

listing the affiliation of every program committee member. Then one can require that papers be reviewed only by PC members (no external reviews allowed) using the following IND:

$$review[reviewer] \subseteq PC[member].$$

Foreign Key Dependencies

In the particular case when Y is a key for relations of schema S ($S : Y \rightarrow V$), INDs of form (4) are called *foreign key dependencies*. Intuitively, in this case the projection on X of every tuple t in r contains the key of a tuple from the "foreign" table s.

In the running example, assuming that every PC member is listed with only one primary affiliation, *member* is a key for PC, so the IND above is really a foreign key dependency.

Expressing Dependencies in First-Order Logic

Embedded Dependencies Despite their independent introduction and widely different syntax, it turns out that all classes of dependencies illustrated above (and many more, including the ones mentioned in the historical background section) can be expressed using a fragment of the language of first-order logic. This fragment is known as the class of *embedded dependencies*, which are formulas of the form

$$\begin{aligned}&\forall x_1 \ldots \forall x_n \ \varphi(x_1, \ldots, x_n) \rightarrow \\ &\exists z_1 \ldots \exists z_k \ \psi(y_1, \ldots, y_m),\end{aligned} \tag{5}$$

where $\{z_1,\ldots,z_k\} = \{y_1,\ldots,y_m\} \setminus \{x_1,\ldots,x_n\}$, and φ is a possibly empty and ψ is a non-empty conjunction of relational and equality atoms. A *relational atom* has form $R(w_1,\ldots,w_l)$, and an *equality atom* has form $w = w'$, where each of $w,w',w_1,\ldots,w_l$ are variables or constants. The particular case when all atoms in ψ are equalities yields the class known as *equality-generating dependencies (EGD)*, while the case when only relational atoms occur in ψ defines the class of *tuple-generating dependencies (TGD)*.

The power of embedded dependencies is illustrated next by expressing the classes of dependencies described above.

Functional Dependencies

Assume without loss of generality that in (1), $|X| = k$, $|Y| = l$, and $|U \setminus (X \cup Y)| = m$, and that the ordering of attributes in U is $U = X, Y, Z$. Then any functional dependency of form (1) is expressible as the embedded dependency (actually an EGD):

$$\begin{aligned}&\forall x_1 \ldots \forall x_k \ \forall y_1 \ldots \forall y_l \ \forall y'_1 \ldots \forall y'_l \ \forall z_1 \ldots \forall z_m \\ &\forall z'_1 \ldots \forall z'_m \\ &R(x_1, \ldots, x_k, y_1, \ldots, y_l, z_1, \ldots, z_m) \ \wedge \\ &\quad R(x_1, \ldots, x_k, y'_1, \ldots, y'_l, z'_1, \ldots, z'_m) \rightarrow \\ &\quad y_1 = y'_1 \wedge \ldots \wedge y_l = y'_l\end{aligned}$$

In the particular case when X is a key, $m = 0$ and there are no z_i, z_i' variables in the above embedded dependency.

The functional dependency $review : paper \rightarrow track$ in the running example, can be expressed as the following embedded dependency:

$$\forall p \forall r \forall t \forall r' \forall t' \; review(p, r, t) \wedge review(p, r', t') \rightarrow t = t'.$$

Join Dependencies

Join dependencies of form (2), are expressed by observing that for every relation r of schema $R(U)$, the inclusion

$$r \subseteq \Pi_{X_1}(r) \bowtie \Pi_{X_2}(r) \bowtie \ldots \bowtie \Pi_{X_n}(r)$$

holds trivially. Therefore only the opposite inclusion needs to be expressed,

$$\Pi_{X_1}(r) \bowtie \Pi_{X_2}(r) \bowtie \ldots \bowtie \Pi_{X_n}(r) \subseteq r.$$

This end requires the following notation. Recalling that the set of attributes U in the schema of R is ordered, let $pos(A)$ denote the position of an attribute $A \in X$ in the ordered set U. For a set of attributes $X \subseteq U$, $pos(X)$ denotes the set $\{pos(A) | A \in X\}$. In the following, given a tuple of variables $\bar{u}$, $\bar{u}[k]$ denotes the kth variable in $\bar{u}$.

1. Let $\{\bar{u}_i\}_{1 \leq i \leq n}$ be a family of tuples of $|U|$ variables each, such that for every $1 \leq i < j \leq n$ and every $1 \leq k \leq |U|$, $\bar{u}_i[k] = \bar{u}_j[k]$ if and only if $k \in pos(X_i \cap X_j)$.
2. Let $\bar{w}$ be a tuple of $|U|$ variables, such that for every $1 \leq k \leq |U|$ and every $1 \leq i \leq n$, if $k \in pos(X_i)$ then $\bar{w}[k] = \bar{u}_i[k]$. It is easy to check that $\bar{w}$ is well-defined: indeed, since $\bigcup_i X_i = U$, for each k there is at least one i with $k \in pos(X_i)$. Moreover, by definition of the family $\{\bar{u}_i\}_i$, $\bar{u}_i[k] = \bar{u}_j[k]$ whenever $k \in pos(X_i)$ and $k \in pos(X_j)$.
3. Finally, let $V = \{v_1, \ldots v_m\} = \bigcup_{i=1}^{n} \bar{u}_i$ (with each tuple $\bar{u}_i$ viewed as a set of variables). Notice that variables occurring in several $\bar{u}_i$ tuples appear just once among V.

Then the join dependency of form (2) is given by the embedded dependency (a TGD, really):

$$\forall v_1 \ldots \forall v_m \; R(\bar{u}_1) \wedge \ldots \wedge R(\bar{u}_n) \rightarrow R(\bar{w}).$$

Join dependency $review : \bowtie[\{paper, track\}, \{paper, reviewer\}]$ is expressed as the following embedded dependency:

$$\forall p \forall r_1 \forall t_1 \forall r_2 \forall t_2 \; review(p, r_1, t_1) \wedge$$
$$review(p, r_2, t_2) \rightarrow review(p, r_2, t_1).$$

Inclusion Dependencies

To express inclusion dependencies, assume without loss of generality that in (4) $R(U) = R(Z, X)$ and $S(V) = S(Y, W)$. Then the inclusion dependency (4) is captured by the following embedded dependency (TGD):

$$\forall z_1 \ldots \forall z_{|Z|} \; \forall x_1 \ldots \forall x_{|X|} \; R(z_1, \ldots, z_{|Z|}, x_1, \ldots, x_{|X|})$$
$$\rightarrow \exists w_1 \ldots \exists w_{|W|} \; S(x_1, \ldots, x_{|X|}, w_1, \ldots, w_{|W|}).$$

In the running example, the inclusion dependency $review[reviewer] \subseteq PC[member]$ is expressible as the embedded dependency

$$\forall p \forall r \forall t \; review(p, r, t) \rightarrow \exists a \; PC(r, a).$$

Other Classes of Dependencies

Embedded dependencies turn out to be sufficiently expressive to capture virtually all other classes of dependencies studied in the literature.

Other Constraints

By employing more expressive sub-languages of first-order logic, one can capture additional, naturally occurring constraints. For instance, extending embedded dependencies with disjunction, one obtains the language of *disjunctive embedded dependencies (DED)* of form

$$\forall \bar{x} \; \varphi(\bar{x}) \rightarrow \bigvee_{i=1}^{l} \exists \bar{z}_i \; \psi_i(\bar{y}_i), \tag{6}$$

where $\bar{x}$ is a tuple of variables, and so are $\bar{z}_i$, $\bar{y}_i$ for every $1 \leq i \leq l$. Analogously to (5), $\bar{z}_i = \bar{y}_i \setminus \bar{x}$. φ and each ψ_i are conjunctions of relational and equality atoms as in (5). If in addition one allows *non-equality atoms* of the form $w \neq w'$, one obtains the class of DEDs with non-equality, DED$^{\neq}$, which is in turn a fragment of first-order logic.

Cardinality Constraints

The language DED$^{\neq}$ can express cardinality constraints. In the running example, $\delta_1 \in$ DED$^{\neq}$ states that every paper has at least two reviews:

$$(\delta_1) \quad \forall p \forall r_1 \forall t \; review(p, r_1, t) \rightarrow$$
$$\exists r_2 \; review(p, r_2, t) \wedge r_1 \neq r_2.$$

δ_2 below states that every paper receives at most two reviews:

$$(\delta_2)\quad \forall p \forall r_1 \forall r_2 \forall r_3 \forall t\ review(p, r_1, t) \wedge review(p, r_2, t) \wedge review(p, r_3, t) \wedge r_1 \neq r_2 \rightarrow r_3 = r_1 \vee r_3 = r_2.$$

Note that the conjunction of δ_1 and δ_2 requires each paper to receive precisely two reviews.

Domain Constraints

The language $\text{DED}^{\neq}$ can be employed to restrict the domain of an attribute. Such restrictions are commonly known as *domain constraints*. For instance, in the running example, this is how to specify that the conference has only three kinds of tracks: "research", "industrial" and "demo":

$$(\delta_3)\ \forall_p \forall_r \forall_t\ review(p, r, t) \rightarrow t = \text{"}research\text{"} \vee t = \text{"}industrial\text{"} \vee t = \text{"}demo\text{"}.$$

Representational Constraints

Many application are based on data models that are richer than the relational model (e.g., object-oriented, object-relational, XML, RDF models). However, they often leverage the mature relational technology by supporting the storage of their data in a relational database. The resulting relations satisfy certain constraints that stem from the original data model. This entry refers to them as *representational constraints*. In order to maintain the relational storage and to efficiently process queries over it, it is imperative to exploit these representational constraints. An obstacle to doing so is the fact that, depending on the original model they encode relationally, representational constraints tend to not fit neatly into any of the classes of relational integrity constraints devised for native relational data. Again, first-order logic comes to the rescue.

Representational constraints for the relational representation of XML are illustrated next. While there are many possible representations, they are all equivalent to the following simple one [7] which captures the fact that in the XML data model, elements are the tagged nodes of a tree. The tree is represented using the following relations (among others):

$$elem(node, tag)$$
$$child(source, target)$$
$$desc(source, target)$$

where the *elem* relation lists for every element *e*, the identifier of the tree node modeling *e*, and the tag of *e*; the *child* table is the edge relation of the XML tree, according to which *source* is the identifier of the parent node and *target* the identifier of the child node; *desc* is the descendant relation in the tree, whose *target* node is a descendant of the *source* node. Any instance storing an actual XML tree in these tables must satisfy, among others, the following constraints, all expressible in $\text{DED}^{\neq}$: every element has at most one tag (expressed in (7) below); every element has at most one parent (8); children of a node are also descendants of this node (9); the descendant relation is transitive (10); if two elements have a common descendant, then they either coincide or one is the descendant of the other (11).

$$\forall n \forall t_1 \forall t_2\ elem(n, t_1) \wedge elem(n, t_2) \rightarrow t_1 = t_2, \quad (7)$$

$$\forall n \forall p_1 \forall p_2\ child(p_1, n) \wedge child(p_2, n) \rightarrow p_1 = p_2, \quad (8)$$

$$\forall s \forall t\ child(s, t) \rightarrow desc(s, t), \quad (9)$$

$$\forall s \forall u \forall t\ desc(s, u) \wedge desc(u, t), \rightarrow desc(s, t), \quad (10)$$

$$\forall n_1 \forall n_2 \forall d\ desc(n_1, d) \wedge desc(n_2, d), \rightarrow n_1 = n_2 \vee desc(n_1, n_2) \vee desc(n_2, n_1). \quad (11)$$

Key Applications

The role of integrity constraints is to incorporate more semantics into the data model. This in turn enables an improved schema design, as well as the delegation to the database management system (DBMS) of the task of enforcing and exploiting this semantics.

Schema Design

Integrity constraints are useful for selecting the most appropriate schema for a particular database application domain. It turns out that the same information can be stored in tables in many ways, some more efficient than others with respect to avoiding redundant storage of data, improved update and query performance, and better readability. While in the early days of database application development, appropriate schema selection started out as an art, it quickly evolved into a science enabling automatic schema design tools (also known as "wizards"). Such tools are based on database theory research that proposes schema design methodology starting from a single, "universal relation", which is then decomposed into new relations that satisfy desirable normal forms that take advantage of the known integrity constraints [1]. For instance, in the running example, both the FD *review* : *paper* → *track* and the MVD *review* : *paper* →→ *track* suggest decomposing relation *review* into two tables, one associating papers with their track, and one associating papers with their reviewers, to avoid the redundant listing of track information with every reviewer.

Automatic Integrity Enforcement

For the purpose of integrity enforcement, the DBMS automatically checks every update operation for compliance with the declared integrity constraints, automatically rejecting non-compliant updates. The database administrator can therefore rest assured that the integrity of her data will be preserved despite any bugs in the applications accessing the database. This guarantee is all the more important when considering that several applications are usually running against the same database. These applications are not always under the control of the database administrator, and are often developed by third parties, which renders their code unavailable for verification. Even with full access to the application code, verification (like all software verification) is technically challenging and does not scale well with increasing number of applications or modifications to their code.

Query Optimization

The query optimizer can exploit its constraint-derived understanding of the data to automatically rewrite queries for more efficient execution. Delegating this task to the DBMS ensures that queries are efficiently executed even when they are issued by applications whose developers write the queries in sub-optimal forms due to insufficient insight into the integrity constraints. More importantly, automatic optimization inside the DBMS handles the queries generated by tools rather than human developers. These queries are usually quite far from optimal. This problem is especially prevalent when queries over a rich data model *M* are translated to relational queries over the relational representation of *M* (e.g., XQuery queries over relational storage of XML). The *chase* is a very powerful tool for reasoning about (and exploiting in optimization) dependencies specified in first-order logic (see [1] for references to the papers that independently discovered the chase, for various classes of dependencies).

A Unified View of Dependencies

Given the plethora of classes of dependencies formulated and studied independently in the literature, the task of specifying the meaning of an application domain via integrity constraints would be daunting if the developer had to fit them into these classes. In addition, tailoring the tasks of schema design, optimization and integrity enforcement to every class of dependencies (and every combination of such classes) is impractical. First-order logic provides a formalism for the simple specification of integrity constraints, with well-understood semantics and sufficient expressive power to capture all common classes of dependencies, and beyond. The insight that virtually all dependency classes are expressible in the same formalism set the foundation for their uniform treatment in research and applications.

Cross-references

- ► Chase
- ► Equality-Generating Dependencies
- ► Tuple-Generating Dependencies

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of databases. Addison-Wesley, Reading, MA, 1995.
2. Beeri C. and Vardi M.Y. The implication problem for data dependencies. In Proc. Int. Conf. on Algorithms, Languages and Programming, 1981, pp. 73–85.
3. Codd E.F. Relational completeness of database sublanguages. In Courant Computer Science Symposium 6: Data Base Systems. R. Rustin (ed.). Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 65–98.
4. Date C.J. Referential integrity. In Proc. 7th Int. Conf. on Very Data Bases, 1981, pp. 2–12.
5. Delobel C. Normalization and hierarchical dependencies in the relational data model. ACM Trans. Database Syst., 3(3):201–222, 1978.
6. Deutsch A., Ludäscher B., and Nash A. Rewriting queries using views with access patterns under integrity constraints. Theor. Comput. Sci., 371(3):200–226, 2007.
7. Deutsch A. and Tannen V. XML queries and constraints, containment and reformulation. Theor. Comput. Sci., 336 (1):57–87, 2005.
8. Fagin R. Horn clauses and database dependencies. J ACM, 29(4):952–985, 1982.
9. Fagin R. Multivalued dependencies and a new normal form for relational databases. ACM Trans. Database Syst., 2(3):262–278, 1977.
10. Fagin R. and Vardi M.Y. The theory of data dependencies: A survey. In Mathematics of Information Processing: Proceedings of Symposia in Applied Mathematics, vol. 34, M. Anshel and W. Gewirtz (eds.). American Mathematical Society, Providence, RI, 1986, pp. 19–27.
11. Gallaire H. and Minker J. Logic and databases. Plenum Press, New York, 1978.
12. Ginsburg S. and Zaiddan S.M. Properties of functional dependency families. JACM, 29(4):678–698, 1982.

13. Grahne G. and Mendelzon A.O. Tableau techniques for querying information sources through global schemas. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 332–347.
14. Kannellakis P.C. Elements of relational database theory. In Handbook of Theoretical Computer Science. J. Van Leeuwen (ed.). Elsevier, Amsterdam, 1991, pp. 1074–1156.
15. Maier D., Ullman J.D., and Vardi M.Y. On the foundations of the universal relation model. ACM Trans. Database Syst., 9(2):283–308, 1984.
16. Mendelzon A.O. and Maier D. Generalized mutual dependencies and the decomposition of database relations. In Proc. 5th Int. Conf. on Very Data Bases, 1979, pp. 75–82.
17. Nicolas J.-M. First order logic formalization for functional, multivalued, and mutual dependencies. Acta Inf., 18(3):227–253, 1982.
18. Paredaens J. Transitive dependencies in a database scheme. Technical Report R387, MBLE, Brussels, 1979.
19. Parker D.S. and Parsaye-Ghomi K. Inference involving embedded multivalued dependencies and transitive dependencies. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1980, pp. 52–57.
20. Sagiv Y. and Walecka S.F. Subset dependencies and a completeness result for a subclass of embedded multivalued dependencies. J ACM, 29(1):103–117, 1982.
21. Vardi M.Y. On decomposition of relational databases. In Proc. 23rd Annual Symp. on Foundations of Computer Science, pp. 176–185, 1982.
22. Vardi M.Y. Trends in theoretical computer science. In Fundamentals of dependency theory. E. Borger Computer Science Press, Rockville, MD, 1987, pp. 171–224.
23. Yannakakis M. and Papadimitriou C. Algebraic dependencies. J Comput. Syst. Sci., 25(2):3–41, 1982.
24. Zaniolo C. Analysis and Design of Relational Schemata for Database Systems. PhD Thesis, University of California, Los Angeles, 1976. Technical Report UCLA-Eng-7669, Department of Computer Science.

Forever

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Infinity; Positive infinity

Definition

The distinguished value *forever* is a special valid-time instant, namely the one following the largest granule in the valid-time domain. Forever is specific to valid time and has no transaction-time semantics.

Key Points

The value *forever* is often used as the valid end time for currently-valid facts. However, this practice is semanticallys incorrect, as such an end time implies that the facts are true during all future times. This usage occurs because database management systems do not offer means of storing the ever-changing current time, *now*, as an attribute value. To fix the incorrect semantics, the applications that manipulate and query the facts may interpret the semantics specially. However, the better solution is to extend the database management system to support the use of the variable *now* as an attribute value.

Concerning the synonyms, "infinity" and "positive infinity" both appear to be more straightforward, but have conflicting mathematical meanings. In addition, the time domain used for valid time may be finite, making the use of "infinity" inappropriate. Furthermore, the term positive infinity is longer and would imply the use of "negative infinity" for its opposite. Forever is intuitive and does not have conflicting meanings.

Cross-references

- ▶ Now in Temporal Databases
- ▶ Temporal Database
- ▶ Time Instant
- ▶ Until Changed
- ▶ Valid Time

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer, Berlin Heidelberg New York, 1998, pp. 367–405.
2. Torp K., Jensen C.S., and Snodgrass R.T. Effective timestamping in databases. VLDB J., 8(3–4):267–288, February 2000.

Form

Jessie Kennedy, Alan Cannon
Napier University, Edinburgh, UK

Synonyms

Forms-based interfaces

Definition

Paper-based forms are structured documents with empty slots that can be filled in by a user. In database systems, a form is an interface containing interaction elements that allow a user to input, update or query data in a database. Forms may also contain static displays of data taken from the database. Documents containing only structured query results and no interaction elements are generally referred to as reports rather than forms. Database interfaces comprising of a series of on-screen forms are referred to as forms-based interfaces.

Commonly a single form corresponds with one table in relational databases and one form instance corresponds with one record. More complex forms may contain or update data from multiple tables. Forms are also used to interact with other databases such as object-oriented or XML databases.

Forms can be designed and created using various visual programming tools such as: Forms specification languages; GUI Builders; Automatic GUI Generation Systems; or by using third and fourth generation programming languages directly. Such tools may be included in the DBMS itself, in a related suite of proprietary programs or be independent applications.

Historical Background

With the advent of more general-purpose commercial database systems in the mid-1960s, the provision of forms-based interfaces for business applications became an issue for developers. Early applications were however still being custom built using various programming languages which was a very time intensive development effort. By the late 1970s/early 1980s a number of tools were becoming available to alleviate the load on the developer, for example, the Forms Application Development System (FADS) developed at U.C. Berkeley between 1979 and 1982. Later the ideas from FADS would develop into the widely used and more sophisticated Application-By-Forms (ABF) tool developed by Ingres Corporation between 1983 and 1986 [11]. Equally, the Oracle database system also introduced a tool for developing forms called Fast Forms (later called SQL*Forms) in 1984 [8]. Other examples include Control Center Forms module in dBASE IV, Forms EXPRESS in R:BASE, Paradox 3.0, Formanager, NFQL [3]. These forms were designed to be used with the alphanumerical terminals then in common usage and accordingly, the forms were character-based with some menus. These forms also tended to be relatively simple, conforming to the underlying database structure with one form usually representing one table and one record per screen.

As Graphical User Interfaces (GUIs) became available and popular for business applications in the late 1980s/1990s, forms took advantage of the new medium. Standard GUI techniques such as multiple windows, checkboxes and point and click selection were employed (e.g., Figure1). Both DBMS and stand alone tools continued to provide support, with updated visual programming tools for GUI based forms (e.g., Oracle-Forms, Ingres's W4GL, Microsoft Access's Form Tool/Wizard). The more powerful DBMS independent development tools had facilities for supporting skilled designers to build forms-based interfaces. While this sped development and empowered designers, it did not support inexperienced designers in avoiding bad designs and still required substantial developer resources. The trend of including forms generation as part of a DBMS or a related application suite that aimed to empower the end-user in customizing their own relatively simple forms has thus continued to date. DBMS provided tools, while increasing in sophistication, are however still restricted to either relatively simple applications or require an expert application developer. Attempts to reduce the reliance on developers by making greater use of automatic GUI creators (including the DBMS tools) have not been generally successful in generating complex forms-based interfaces.

In the last decade, the emphasis has shifted from a traditional client-server delivery method for forms, to a web-based one. Web forms are now prevalent across the internet (e.g., Figure 2). These web forms use HTML with various types of scripting or more recent alternatives such as XForms, XQForms [9].

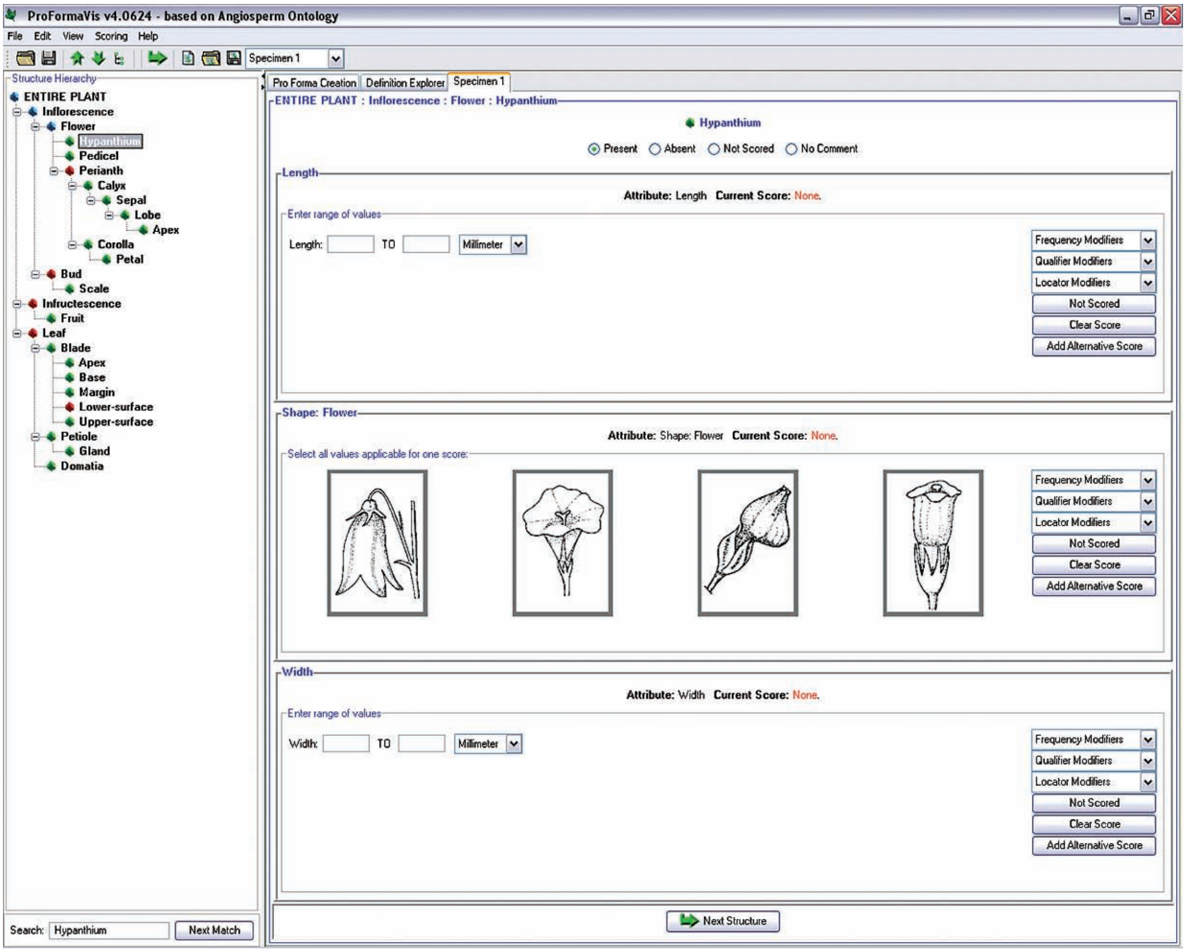
Foundations

Form Composition

In database systems, a form is an interface containing interaction elements that allow a user to input, update or query data in a database. Each form instance normally equates with a single database record, although exceptions exist and one database record may be spread over multiple forms.

Forms can be character-based, GUI-based (Fig. 1) or web-based (Fig. 2), but all basically comprise of four main elements that are tied together within logical containers such as a windows or frames:

1. Data entry interaction objects
2. Static labels

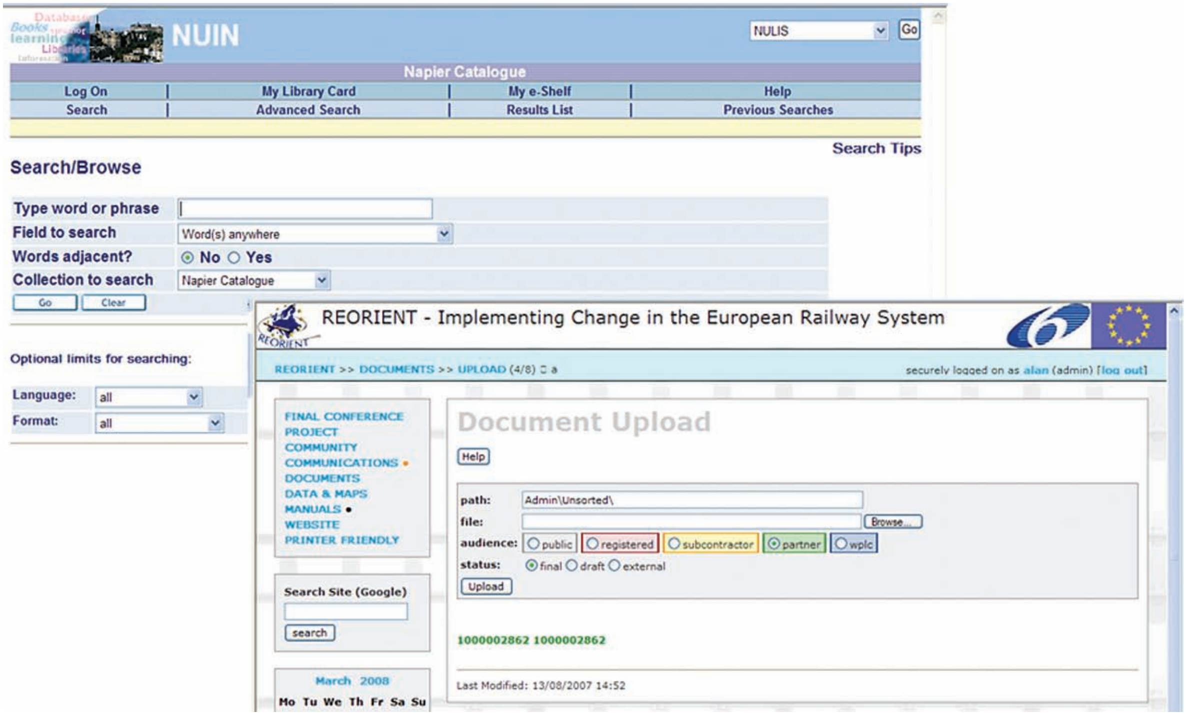


Form. Figure 1. Example of a GUI form for data entry [8].

3. Displayed data
4. Control elements

The data entry interaction objects are the slots into which data can be entered or altered by users. These elements are each bound to an associated attribute of a database table (or equivalent in non-relational databases). These comprise of such elements as text boxes, checkboxes, calendar objects, or any other of the interaction widgets that can be used to enter data. They may be initially blank or contain data, either taken from the existing record in the database (i.e., the current values of the bound attribute), from another section of the database (e.g., a range of possible values from a related table), from default attribute values specified in the database or from default values programmed into the form. The entered data can be used to fill a new record, update an existing one or query the database according to the underlying application logic. Each of the data entry interaction objects is usually associated with a static label specifying to the form user what data should be inserted. These are not usually bound to an attribute of the database. Other static data displayed in a form include existing data from the database of relevance to the form (e.g., other identifying data from the record being edited) or the results of user queries. This data may be a direct representation or it may be data manipulated by calculations programmed into the form. The last main elements of a form are control elements such as menus or buttons. These provide the form user with navigational and other controls over form operations. The most obvious examples of these are submit controls, that commit changes to the database.

Underlying business logic is programmed into a form to provide relevant database interaction, navigation controls, error checking and data validation. SQL or other query language code is normally embedded into



Search/Browse

Type word or phrase:
Field to search: Word(s) anywhere
Words adjacent? ● No ○ Yes
Collection to search: Napier Catalogue

Optional limits for searching:
Language: all
Format: all

Document Upload

path: Admin\Unsorted\
file:
audience: ○ public ○ registered ○ subcontractor ● partner ○ wplc
status: ● final ○ draft ○ external

1000002862 1000002862

Last Modified: 13/08/2007 14:52

Form. Figure 2. Example of a two web-based forms for querying Napier University Library Catalog and for uploading a document to an on-line document repository.

form fields to support data updates or queries. A form can also contain hidden fields that are also sent to a database along with any other update or query (e.g., user id). This coding may be implemented directly by a developer using a third or fourth generation programming language or semi-automatically by development tools based on user interaction or totally automatically by forms generation tools directly from database schema.

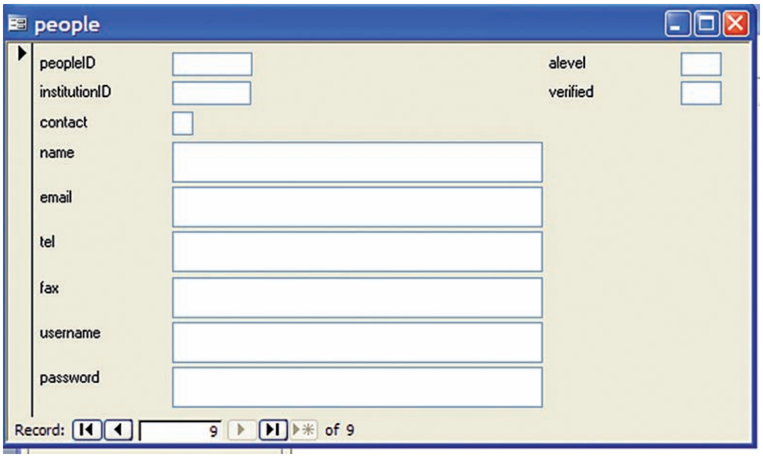
Generating Forms

Forms-based interfaces include sets of forms and the application logic that ties them together. Traditionally their generation has been a task for an expert application developer. To decrease that reliance, there have been attempts to allow end-users to develop their own forms-based interfaces. The main methods of generation are similar to any other user interface.

Traditional programming with languages that support database calls remain a common method for generation by application developers, particularly with fourth Generation languages and their associated Interface Development Environments. They permit powerful customized applications to be built.

Other visual development tools are also used both by application developers and more causal users. These can be stand-alone tools or linked to proprietary DBMS. The latter option is increasingly common, providing specific support for forms-based database interfaces. These all generally work on similar principles where the forms are designed visually with tools that allow fields, labels and other elements to be placed on the screen, using a WISYWIG system. The underlying application logic is then either selected with menus (often point and select) or by adding code in a database query language, programming or scripting language. Web based development for a forms-based interface is similar in nature (e.g., Macromedia's ColdFusion, Adobe's Dreamweaver).

Some DBMS visual development tools (e.g., MS Access, OracleForms, Paradox) have support for automatically generating forms from the database schema (Fig. 3), though the resulting interfaces are certainly at the simpler end of the forms and applications, being tied to a simple view of the database schema (such as in Figure 3), with only basic validation by data type for example being possible (for example by dragging the table attributes to XHTML form controls [10]).



Form. Figure 3. Example of a simple form created using Microsoft Access Form Wizard.

Other non-DBMS tools support automatic generation of the forms from various abstractions, where a presentation model controls the selection and layout of user interfaces, based on the modeled tasks and/or domain. These are known as model based approaches. The various approaches use different models, but usually focus on a task or domain/data model. Some approaches support completely automatic interface generation (e.g., MECANO, JANUS) and others provide support and guidance to designers to finalize an interface (e.g., MASTERMIND). Most of the approaches claim to connect to relational databases and one, Teallach, claims to connect to any object-orientated database [1]. These approaches have not, however, been widely adopted despite a strong research base [12], partly since they are generic systems and it is difficult to get good abstract domain and presentation models for specific applications. A related ontology based approach exists where an end-user edits a lightweight ontology to define the data model for a particular application. The system can then produce a forms-based data entry interface for that data model, based on a domain specific presentation model determined by the application developer [2].

Key Applications

Business Database Applications

The primary use for forms-based interfaces remains business applications, although as database technology continues to become more accessible, an increasing number of other casual users are utilizing databases and attendant forms-based interfaces for hobby and other leisure uses.

End-User Programming and Domain Specific Application Development

Empowering domain users to take over some of the application developer workload in creating forms for their own applications promises gains in development efficiency by reducing the need for a developer to learn about a domain and potentially in the applicability of the resulting forms. Improving the usability and utility of the various Integrated Development Environments for end-users is one way that database researchers can achieve this.

Ubiquitous Computing

A variety of new interface challenges must be met on different types of displays such as PDAs, mobile phones, and wall screens. It seems likely that the forms metaphor will continue to be used on these new devices and means of effectively tailoring forms to the requirements of the various displays and interaction methods must be found.

Data Quality

The phrase 'Garbage-In, Garbage-Out' remains as true today as at the dawn of computing. With forms being used constantly for data entry and updates, the control of entered data remains an important consideration. The design and data validation constraints placed on forms will continue to affect the quality of the data entered by users.

Meta-Data Capture

Forms-based interfaces are the norm for many systems such as data repositories and content management systems, which require to capture meta-data. The design and applicability of the forms used in these applications are vital to encourage users to enter meta-data, a task they are generally reluctant to undertake.

Knowledge Bases

Forms-based interfaces are also used by various ontological systems to populate knowledge bases (e.g., Protégé Frames). Existing ontology based approaches for data entry are, however, generally still limited to using automatically generated forms-based data entry interfaces unless manual editing is used. A form tends to be generated for each class instance with the choice of interaction widgets derived from the slot data types. Links between forms are based on class subsumption relationships.

Cross-references

- ▶ Database Management System
- ▶ Data Visualization
- ▶ DBMS Interface
- ▶ Direct Manipulation
- ▶ GUIs for Web Data Extraction
- ▶ Interface
- ▶ Multimodal Interfaces
- ▶ Ontology-Based Data Models
- ▶ Query language
- ▶ Table
- ▶ Views
- ▶ Visual Interfaces
- ▶ Web Services
- ▶ XML

Recommended Reading

1. Barclay P., Griffiths T., McKirdy J., Kennedy J., Cooper R., Paton N., and Gray P. Teallach – a flexible user-interface development environment for object database applications. J. Vis. Lang. Comput., 14(1):47–77, 2003.
2. Cannon A., Kennedy J., Paterson T., and Watson M. Ontology-driven automated generation of data entry interfaces. In Key Technologies for Data Management: 21st British National Conf. on Databases, 2004, pp. 150–164.
3. Embley D.W. NFQL: the natural forms query language. ACM Trans. Database Syst., 14(2):168–211, 1989.
4. Fry J.P and Sibley E.H. Evolution of data-base management systems. ACM Comput. Surv., 8(1):7–42, 1976.
5. Jagadish H.V., Chapman A., Elkiss A., Jayapandian M., Li Y., Nandi A., and Yu C. Making database systems usable. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 13–24.
6. Molina P.A. Review to model-based user interface development technology. In Proc. 1st Int. Workshop on Making Model-Based User Interface Design Practical: Usable and Open Methods and Tools. CEUR Workshop Proceedings 103 CEUR-WS.org, 2004.
7. Myers B., Hudson S.E., Pausch R. Past, present, and future of user interface software tools. ACM Trans. Comput. Hum. Interact., 7:(1)3–28, 2000.
8. Oracle Corporation. Oracle Forms. 2008. Available at: http://www.oracle.com/technology/products/forms/index.html
9. Petropoulos M., Vassalos V., and Papakonstantinou Y. XML query forms (XQForms): declarative specification of XML query interfaces. In Proc. 10th Int. World Wide Web Conference, 2001, pp. 642–651.
10. Raggett D., Le Hors A., and Jacobs I. XHTML HTML 4.01 Specification W3C Recommendation, 1999. Available at: URL: http://www.w3.org/TR/html4/
11. Rowe L.A. A retrospective on database application development frameworks. ACM SIGMOD Rec., 21(1):5–10, 1992.
12. Trætteberg H., Molina P.J., and Nunes N.J. Making model-based UI design practical: usable and open methods and tools. In Proc. 9th Int. Conf. on Intelligent User interface, 2004, pp. 376–377.

Forms-based Interfaces

▶ Form

Fourth Normal Form

Marcelo Arenas
Pontifical Catholic University of Chile, Santiago, Chile

Synonyms

4NF

Definition

Let $R(A_1,...,A_n)$ be a relation schema and Σ a set of unctional and multivalued dependencies over $R(A_1,...,A_n)$. Then (R, Σ) is said to be in Fourth Normal Form (4NF) if for every nontrivial multivalued dependency $X \twoheadrightarrow Y$ implied by Σ, it holds that X is a superkey for R.

Key Points

In order to avoid update anomalies in database schemas containing functional and multivalued dependencies, 4NF was introduced by Fagin in [2]. As for the case of BCNF, this normal form is defined in terms of the notion of superkey as shown above. For example, given a relation schema $R(A, B, C)$ and a set of functional dependencies $\Sigma = \{A \rightarrow B\}$, it does not hold that $(R\ (A, B, C), \Sigma)$ is in 4NF since $A \twoheadrightarrow B$ is a nontrivial multivalued dependency implied by Σ and A is not a superkey for R. Similarly, relation schema $S\ (A, B, C)$ and set of multivalued dependencies $\Sigma = \{A \twoheadrightarrow C\}$ is not in 4NF since A is not a superkey for S. On the other hand, relation schema $T(A,B,C)$ and set of functional and multivalued dependencies $\Sigma = \{AB \rightarrow C, AC \rightarrow B, A \twoheadrightarrow B\}$ is in 4NF since AB, AC and A are all superkeys for T.

It should be noticed that relation schema $S\ (A, B, C)$ above is in BCNF if $\Sigma = \{A \twoheadrightarrow C\}$ since no nontrivial functional dependency can be inferred from Σ. In fact, 4NF is strictly stronger than BCNF; every schema in 4NF is in BCNF, but there exist schemas (as the one shown above) that are in BCNF but not in 4NF.

For every normal form two problems have to be addressed: How to decide whether a schema is in that normal form, and how to transform a schema into an equivalent one in that normal form. As for the case of BCNF, it can be tested efficiently whether a relation schema is in 4NF. A relation schema (R, Σ) is in 4NF if and only if for every nontrivial functional dependency $X \rightarrow Y \in \Sigma$, it holds that X is a superkey, and for every nontrivial multivalued dependency $X \twoheadrightarrow Y \in \Sigma$, it holds that X is a superkey. Thus, it is possible to check efficiently whether (R, Σ) is in 4NF by using a polynomial time algorithm for functional and multivalued dependency implication [1]. On the negative side, given a relation schema S, it is not always possible to find a database schema S' such that S' is in 4NF and S' is a lossless and dependency preserving decomposition of S.

Cross-references

- ▶ Boyce-Codd Normal Form
- ▶ Normal Forms and Normalization
- ▶ Second Normal Form (2NF)
- ▶ Third Normal Form

Recommended Reading

1. Beeri C. On the membership problem for functional and multivalued dependencies in relational databases. ACM Trans. Database Syst., 5(3):241–259, 1980.
2. Fagin R. Multivalued dependencies and a new normal form for relational databases. ACM Trans. Database Syst., 2(3):262 –278, 1977.

FQL

Peter M.D. Gray
University of Aberdeen, Aberdeen, UK

Definition

At about the same time that Shipman was developing DAPLEX, Buneman and Frankel proposed the highly influential FQL functional query language [1], based on Backus's FP functional programming paradigm. A major motivation for this work was that it is in principle possible to combine arbitrary *computable functions* with *stored database functions* into a functional query language which is not limited to a predefined set of operators.

FQL was intended for implementation over existing DBMSs and an abstract implementation based on the lazy evaluation of stream data was described. In later publications [2,3], FQL was extended with features from the functional programming language ML. In this version, function definition is simpler than in the variable-free FP-like syntax and new higher-order functions can be defined in addition to the built-in ones. Also a set of built-in metalevel functions is provided, as in EFDM. Thus, object-level querying and meta level querying can be carried out in the same functional syntax.

Key Points

FP includes a built-in set of *functionals*, like higher-order functions, including function composition, $\circ$, and sequence construction. In addition to these, FQL provides the functionals !, * and ^.

Given an abstract entity type A, !A is a 0-argument function which returns a *stream* of all entities of type A. For example, given the abstract entity type COURSE, !COURSE returns a stream of all the entities of type COURSE.

Given any type A, *A is the type consisting of streams of entities of type A. For example, given the abstract entity types STUDENT and COURSE, the function `ATTENDS : STUDENT -> *COURSE` takes a STUDENT and returns a stream of the COURSEs attended by the STUDENT.

Given a function f: A → B, *f is the function of type *A → *B which takes as an argument a stream of entities of type A, a_1, a_2,... say, and returns the stream f(a_1), f(a_2),.... For example, given the function `CNAME : COURSE -> STRING` the query `COURSE o *CNAME` returns the name of every COURSE.

Given a function f: A → B or f: A → *B, the function ^f: B → *A is the inverse (or converse) of f. In a later version of FQL [2], the functionals ! and ^ are dropped.

Cross-references

► Functional Query Language

Recommended Reading

1. Buneman P. and Frankel R.E. FQL – A Functional Query Language. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 52–58.
2. Nikhil R. An incremental, strongly-typed database language. Technical Report PhD Thesis, University of Pennsylvania, 1984.
3. Nikhil R. Practical polymorphism. In Proceedings of Functional Programming and Computer Architecture'85, LNCS 201. 1985, pp. 319–333.

Fractal

Ke Deng
University of Queensland, Brisbane, OLD, Australia

Synonyms

Koch snowflake; Space-filling curve

Definition

A fractal is "a rough or fragmented geometric shape that can be subdivided in parts, each of which is (at least approximately) a reduced-size copy of the whole" [2]. This term is introduced by French mathematician Benoit Mandelbrot (born 1924) in 1975 and was derived from the Latin fractus meaning "broken" or "fractured". A simple fractal example is Koch snowflake. The method of creating this shape is to recursively replace each line segment with 4 line segments in a finer scale as shown in Fig. 1.

Key Points

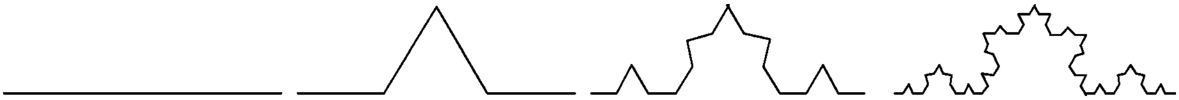
Fractals are self-similar structures (at least approximately or stochastically) which are appropriate for representing the geometry in nature. Some examples include clouds, snow flakes, coastlines, crystals, cauliflower or broccoli, and mountain ranges. In fractal geometry, the fractal dimension, D, is a statistical quantity that gives an indication of how completely a fractal appears to fill space, as one zooms down to finer and finer scales. Among several choices, Hausdorff dimension (also known as the Hausdorff-Besicovitch dimension) is a popular definition of fractal dimension and it is:

$$D = \lim_{\varepsilon \to 0} \frac{\log N(\varepsilon)}{\log(1/\varepsilon)}.$$

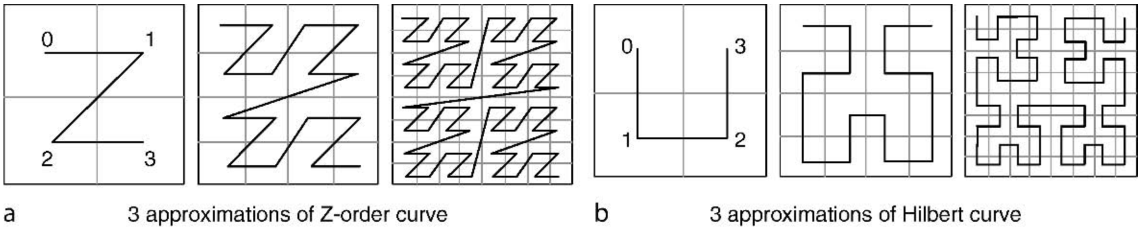
Where $N(\varepsilon)$ is the number of balls of radius at most ε to cover the fractal. The Hausdorff dimension of Koch snowflake is:

$$D = \frac{\log 4^n}{\log 3^n} = \frac{\log 4}{\log 3} = 1.2618...$$

Another case of fractals is the space-filling curves which is first described by Giuseppe Peano (1858–1932) and has Hausdroff dimension $D = 2$. Space-filling curve is useful in computer science in two general situations, managing multi-dimensional points and storing k-dimensional array on the disk [1]. This attributes to two features. (i) Space-filling curve is a continuous curve capable of mapping from d-dimensional space to 1-dimensional space. Using this mapping, a point in the cube can be described by its spatial coordinates, or by the length along the curve, measured from one its ends. Any one dimensional data structure can be used such as binary search trees, B-trees or Hash tables. (ii) Space-filling curve has good locality preserving behaviour such that points



Fractal. Figure 1. Recursively replace each line segment with 4 line segments in a finer scale in Koch snowflake.


0
1
2
3
a
3 approximations of Z-order curve
0
3
1
2
b
3 approximations of Hilbert curve

Fractal. Figure 2. Two space filling curves.

are close together in the 1-dimensional space are mapped from points that are close together in the d-dimensional space. The reverse property is not true, i.e., not all adjacent cells in the d-dimensional space are adjacent or even close on the curve. A group of contiguous cells in d-dimensional space will typically be mapped to a collection of segments on the space-filling curve. There segments are called clusters. Z-order curve and Hilbert curve are space-filling curves as shown in Fig. 2a,b.

Cross-references

▶ Space-Filling Curve

Recommended Reading

1. Faloutsos C., and Roseman S., Fractals for secondary key retrieval. In Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1989, pp. 247–252.
2. Mandelbrot B. B., Fractal Geometry of Nature, W. H. Freeman, San Francisco, CA,1977.

Frequency Moments

David Woodruff
IBM Almaden Research Center, San Jose, CA, USA

Synonyms

L_p norms; L_p distances

Definition

Consider a stream (i.e., an ordered list) $\mathcal{S} = a_1, a_2,\ldots, a_n$ of elements $a_i \in [m] \overset{\text{def}}{=} \{1,2,\ldots,m\}$. For $i \in [m]$, its frequency f_i is the number of times it occurs in $\mathcal{S}$. The k th frequency moment F_k of $\mathcal{S}$, for real $k > 0$, is defined to be $F_k(\mathcal{S}) = \sum_{i\in[m]} f_i^k$. Interpreting 0^0 as 0, one can also define F_0 this way, so that it equals the number of distinct elements in $\mathcal{S}$. Observe that $F_1 = n$ is the length of $\mathcal{S}$. In the database community, F_2 is known as the repeat rate or Gini's index of homogeneity. It is also natural to define $F_\infty = max_{1\leq i\leq m} f_i$.

It is usually assumed that n is very large and that algorithms which compute the frequency moments do not have enough storage to keep the entire stream in memory. It is also common to assume that they are only given a constant (usually one) number of passes over the data. It is further assumed that the stream is presented in an arbitrary, possibly worst-case order. This necessitates the use of extremely efficient randomized approximation algorithms. An algorithm A (ϵ, δ)-approximates the k th frequency moment F_k if for any input stream $\mathcal{S}$, $\Pr[|A(\mathcal{S}) - F_k(\mathcal{S})| \leq \epsilon F_k(\mathcal{S})] \geq 1 - \delta$, where the probability is over the coin tosses of A. Here, $A(\mathcal{S})$ means that A is presented items in $\mathcal{S}$ one-by-one. Efficiency is measured in terms of the amount of memory and update time of the algorithm.

Key Points

The frequency moments were introduced by Alon, Matias, and Szegedy [1] in their seminal paper in 1996, and are important statistics for massive databases. Indeed, efficient algorithms for estimating F_0 can be used by query optimizers for finding the number of unique values of an attribute without having to perform an expensive sort on the entire column. Efficient algorithms for F_2 are useful for determining the output size of self-joins and for error estimation in the context of query result sizes and access plan costs. Moreover, F_k for $k \geq 2$ can indicate the amount of skew of a data stream, and this can determine which algorithms to use for data partitioning. These values can also be used to detect denial-of-service attacks. In general, F_k for large k can be used to approximate the most frequent value, potentially more efficiently than computing this value directly.

There is a large body of work on bounding the memory required of F_k-approximation algorithms. In their original work, Alon, Matias, and Szegedy surprisingly showed that F_2 can be (ϵ, δ)-approximated using only $O(\frac{\lg 1/\delta}{\epsilon^2}(\lg n + \lg m))$ bits of memory. It is now known that F_k for $k \leq 2$ can be (ϵ, δ)-approximated in 1-pass using $O(\frac{\lg 1/\delta}{\epsilon^2})$ bits of space, up to a *polylog nm* factor, and there is an almost matching $\Omega(1/\epsilon^2)$ lower bound for 1-pass algorithms. For $k > 2$, a sequence of work showed that F_k can be approximated in $O(m^{1-2/k} log 1/\delta)$ space, up to a *poly* $(lognm, 1/\epsilon)$ factor, and there is an almost matching $\Omega(m^{1-2/k})$ bound. The memory required for approximating F_∞ is $\Theta(m)$. Note that for $k > 2$ the memory required depends polynomially on m, whereas for $k \leq 2$ the dependence is logarithmic. The known algorithms use a clever combination of hashing (with limited independence), sketching, and bucketing ideas. The corresponding lower bounds come from reductions from problems in communication complexity, and draw from tools in extremal combinatorics and information theory.

There are several natural questions about the computational complexity of frequency moments which remain unanswered. For instance, for constant δ, it is unknown if F_k for $0 \leq k \leq 2$ can be approximated more efficiently if more than one pass (but still a constant number) is allowed. Also, it is unknown how efficiently F_k can be approximated if the stream elements arrive in a random order.

Cross-references

- ▶ Data Mining
- ▶ Data Stream
- ▶ Stream Mining
- ▶ Stream Processing
- ▶ Stream Sampling
- ▶ Streaming Applications

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments, J. Comput. Syst. Sci., 58:137–147, 1999.
2. Bar-Yossef Z., Jayram T., Kumar R., and Sivakumar D. An information statistics approach to data stream and communication complexity. In Proc. 43rd Annual Symp. on Foundations of Computer Science, 2002, pp. 209–218.
3. Indyk P. and Woodruff D. Optimal approximations of the frequency moments of data streams. In Proc. 37th Annual ACM Symp. on Theory of Computing, 2005, pp. 202–208.

Frequent Concepts

▶ Closed Itemset Mining and Nonredundant Association Rule Mining

Frequent Elements

▶ Frequent Items on Streams

Frequent Graph Patterns

Jun Huan
University of Kansas, Lawrence, KS, USA

Synonyms

Graph database mining

Definition

There are three key concepts in mining graph databases: (i) labeled graph, (ii) subgraph isomorphism, and (iii) graph support value. Based on the concepts, the problem of frequent subgraph mining could be defined in the following discussion.

Definition 1. *A **labeled graph** G is a quadruple G = (V, E, Σ, λ) where V is a set of vertices or nodes and $E \subseteq V \times V$ is a set of undirected edges. Σ is a set of (disjoint) vertex and edge labels, and λ: $V \cup E \rightarrow \Sigma$ is a function that assigns labels to vertices and edges. Typically a total ordering is defined on the labels in Σ.*

With the previous definition, a *graph database* is a set of labeled graphs.

Definition 2. *A graph $G' = (V', E', \Sigma', \lambda')$ is **subgraph isomorphic** to $G = (V, E, \Sigma, \lambda)$, denoted by $G' \subseteq G$, if there exists a 1-1 mapping $f: V' \rightarrow V$ such that*

- $\forall v \in V', \lambda'(v) = \lambda(f(v))$
- $\forall (u,v) \in E', (f(u),f(v)) \in E$, *and*

- $\forall(u,v) \in E', \lambda'(u,v) = \lambda(f(u), f(v))$.

The function f is a *subgraph isomorphism* from graph G' to graph G. For simplicity, G' *occurs* in G if $G' \subseteq G$. Given a subgraph isomorphism f, the image of the domain V' ($f(V')$) is an embedding of G' in G.

Example 1. *Figure 1 shows a graph database of three labeled graphs. The mapping (isomorphism) $q_1 \rightarrow p_3$, $q_2 \rightarrow p_1$, and $q_3 \rightarrow p_2$ demonstrates that graph Q is isomorphic to P and so Q occurs in P. Set $\{p_1, p_2, p_3\}$ is an embedding of Q in P. Similarly, graph S occurs in graph P but not Q.*

Definition 3. *Given a graph database $\mathcal{G}$, the* support *of a graph C, denoted by sup(C), is the fraction of graphs in $\mathcal{G}$ in which C occurs.*

Problem Statement: Given a graph database $\mathcal{G}$ and a support threshold $0 < \sigma \leq 1$, the *frequent subgraph mining* problem is to identify all graphs whose support value is at least σ.

There are several possible extensions to the problem of frequent subgraph mining. For example, the node may contain multiple label rather than one [2] and the identified patterns may contain node with or without labels [6]. Special cases for frequent subgraph mining include frequent tree mining, frequent path mining, and frequent cycle mining from graphs. The current introduction emphasizes the fundamental problem frequent subgraph mining and covers little about the mentioned extensions.

Historical Background

Graph database mining is an active research field. Recent graph mining algorithms can be roughly divided into three categories. The first category uses a level-wise search strategy, including AGM [6] and FSG [7]. The second category takes a depth-first search strategy, including gSpan [11] and FFSM [4]. The third category works by first mining frequent trees and then construct cyclic graphs based on tree pattern. This category includes SPIN [5] and GASTON [8]. See [1] for a recent survey.

The following description of frequent subgraph mining is mainly based on the algorithms FFSM and SPIN. Both algorithms are built on top of a data structure termed CAM tree (graph Canonical Adjacency Matrix tree), which is a compact representation of a space of graphs. Using CAM tree, FFSM (Fast Frequent Subgraph Mining, [4] supports an incremental subgraph isomorphism check and hence is scalable for large graphs. SPIN (SPanning tree based mINing, [5]) reduces the total number of mined patterns by identifying only the *maximal* ones, i.e., the set of frequent subgraphs such that none of their supergraphs is frequent.

Foundations

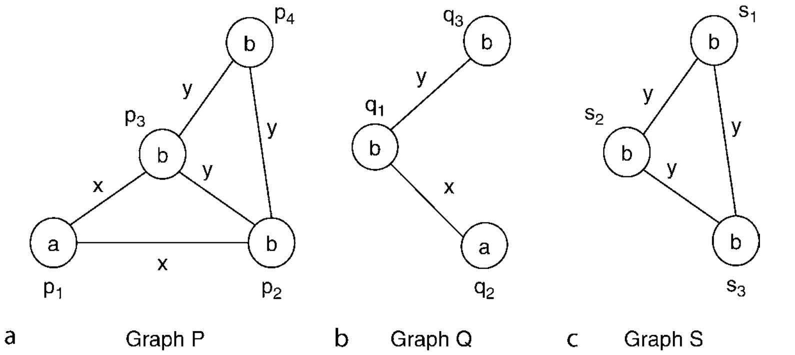
Three topics are important for designing efficient subgraph mining algorithms:

1. Graph Canonical Form.
2. Candidate Graph Proposing.
3. Candidate Support Value Computation.

In the following, the three topics are discussed in details.

Canonical Adjacency Matrix

For labeled graph, a natural way to present a graph is using an modified adjacency matrix. In this representation, every diagonal entry of the modified adjacency matrix is filled with the label of the corresponding node and every off-diagonal entry is filled with the label of the corresponding edge, or zero if there is no



Frequent Graph Patterns. Figure 1. A database of three labeled graphs.

edge. This is slightly different from the widely used adjacency matrix representation for unlabeled graphs.

One of the critical problems in graph mining is the graph isomorphic problem: given two graphs P and Q, determine whether P is isomorphic to Q. This problem may be solved following a common theme such that a graph is first transformed to a unique presentation (referred to as the *canonical form* of the graph) and secondly, two graphs are compared using the transformed presentations. A graph canonical form is specially designed such that if two graphs are isomorphic to each other, their canonical forms are the same and vice versa.

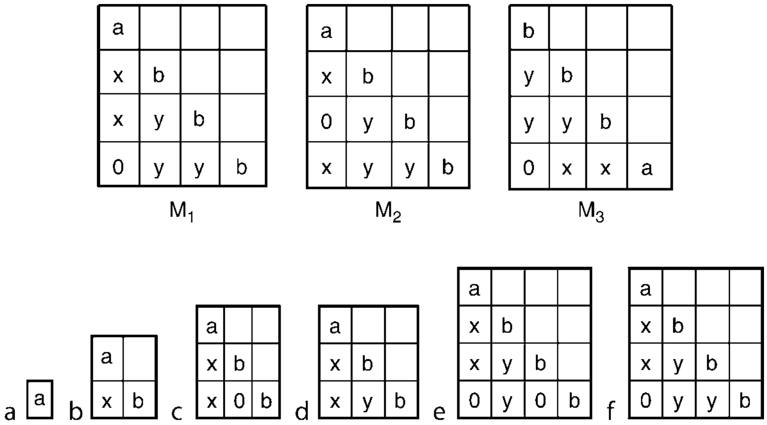
Following convention, in the following a capital letter is used to denote an adjacency matrix and the corresponding lower case letter (augmented with subscripts) is used to denote an individual entry of the adjacency matrix. For instance, $m_{i,j}$ denotes the entry on the ith row and jth column of an $n \times n$ adjacency matrix M, $0 < j \leq i \leq n$. Note that the adjacency matrix is not unique for a given graph: a permutation of the set of vertices in a graph may generate a different adjacency matrix. There is a total of $n!$ different permutations for a graph vertex set with size n, there may be up to $n!$ adjacency matrices for the graph. Figure 2 (top) shows three adjacency matrices for the labeled graph P in Fig. 1 (Only the lower triangular part of an adjacency matrix is drawn since the upper half is a mirror of the lower half.). In order to enable a unique representation for each graph, the code of adjacency matrices is defined below. Graph code helps in defining a total order among all adjacency matrices.

Definition 4. *Given an $n \times n$ adjacency matrix M of a graph G with n vertices, the* **code** *of M, denoted by code (M), is the sequence formed by concatenating lower triangular entries of M (including diagonal entries) in the order: $m_{1,1}m_{2,1}m_{2,2}...m_{n,1}m_{n,2}...m_{n,n-1}m_{n,n}$ $(0 < j \leq i \leq n)$.*

For an adjacency matrix M, each diagonal entry of M is a *node entry* and each off-diagonal none-zero entry in the lower triangular part of M is an *edge entry*. Edge entries are ordered according to their relative positions in the code of the matrix. For example the *first* edge entry of M is $m_{2,1}$ and the *last* edge entry is the one appears rightmost in *code*(M).

Example 2. *Figure 2 shows codes and edge entries for the labeled graph P showing in Fig. 1. For example for adjacency matrix M_1, the edge entry set is $\{m_{2,1}, m_{3,1}, m_{3,2}, m_{4,2}, m_{4,3}\}$ where $m_{2,1}$, $m_{4,3}$, and $m_{4,2}$ are the first, last, second-to-last edge entries of M, respectively. After applying the total ordering, the relationships are: code (M_1) = "axbxyb0yyb" $\geq$ code(M_2) = "axb0ybxyyb" $\geq$ code(M_3) = "bybyyb0xxa." Submatrices of CAM are shown at the bottom. Matrix 3.a is the proper maximal submatrix of matrix 3.b, which itself is the proper maximal submatrix of 3.c and so on so forth.*

Standard lexicographic order on sequences is usually used to define a total order of two arbitrary codes p and q. Given a graph G, its *canonical form*, denoted by $\varphi(G)$, is the maximal code among all its possible codes. The adjacency matrix M that produces the canonical form is the G's *canonical adjacency matrix* (CAM). For example, the adjacency matrix M_1 shown in Fig. 2 is the CAM of the graph P in Fig. 1, and *code*(M_1) is the canonical form of the graph.



$$M_1 = \begin{array}{|c|c|c|c|}\hline a & & & \\ \hline x & b & & \\ \hline x & y & b & \\ \hline 0 & y & y & b \\ \hline\end{array} \quad M_2 = \begin{array}{|c|c|c|c|}\hline a & & & \\ \hline x & b & & \\ \hline 0 & y & b & \\ \hline x & y & y & b \\ \hline\end{array} \quad M_3 = \begin{array}{|c|c|c|c|}\hline b & & & \\ \hline y & b & & \\ \hline y & y & b & \\ \hline 0 & x & x & a \\ \hline\end{array}$$

$$a \ \begin{array}{|c|}\hline a \\ \hline\end{array} \quad b \ \begin{array}{|c|c|}\hline a & \\ \hline x & b \\ \hline\end{array} \quad c \ \begin{array}{|c|c|c|}\hline a & & \\ \hline x & b & \\ \hline x & 0 & b \\ \hline\end{array} \quad d \ \begin{array}{|c|c|c|}\hline a & & \\ \hline x & b & \\ \hline x & y & b \\ \hline\end{array} \quad e \ \begin{array}{|c|c|c|c|}\hline a & & & \\ \hline x & b & & \\ \hline x & y & b & \\ \hline 0 & y & 0 & b \\ \hline\end{array} \quad f \ \begin{array}{|c|c|c|c|}\hline a & & & \\ \hline x & b & & \\ \hline x & y & b & \\ \hline 0 & y & y & b \\ \hline\end{array}$$

Frequent Graph Patterns. Figure 2. Top: three adjacency matrices for the graph P in Fig. 1. Bottom: examples of maximal proper submatrices.

CAM Tree

Efficient frequent subgraph mining algorithm relies on efficient data structure. Below a widely data structure called CAM tree is discussed. CAM tree utilizes a key property of the previous canonical form, i.e., a “prefix” of the canonical form is also maximal, which is stated in the following theorem.

Definition 5. *Given an $n \times n$ matrix N and a $m \times m$ matrix M, let $m_{l,k}$ be the last edge entry of M, a matrix N is the* ***maximal proper submatrix*** *of M if N is obtained by removing $m_{l,k}$ from M*

Several examples of the maximal proper submatrices are given at the bottom of Fig. 2.

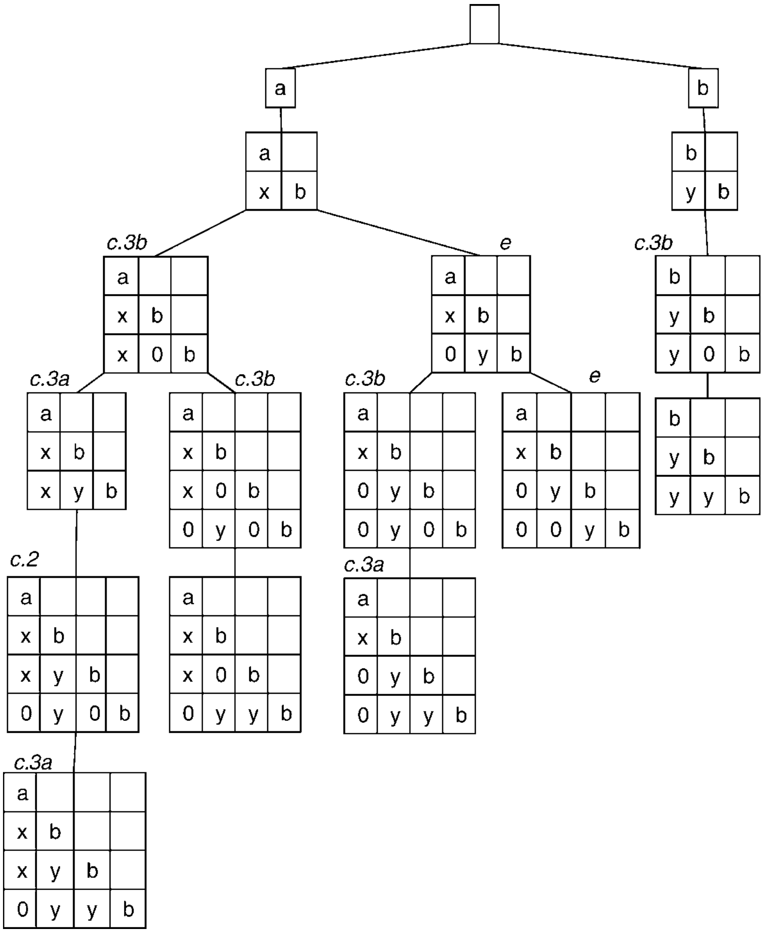
Corollary 1 *Given a CAM M of a connected graph G and a submatrix N of M, N represents a connected subgraph of G.*

Proof 1 *Since N must represent a subgraph of G, it is sufficient to show that the subgraph N represents is connected. To prove this, it is sufficient to show that in N there is no such row i (with the exception of the first row) that i doesn't contains any edge entry. The rest of the proof is trivial and could be found in* [6].

Corollary 2 *Given a connected graph G with CAM M, a submatrix N of M, and a graph H which N represents, N is the CAM of H.*

Proof 2 *For simplicity, M is assumed to have only one edge entry in the last row. It is trivial to show that code (N) is a prefix of code(M). This suggests that code$(N) \geq$ code$(CAM(H))$, where $CAM(H)$ stands for the canonical adjacency matrix of graph H. Therefore N must be the CAM of the graph H.*

For a CAM with at least two edge entries in the last row, a similar proof could be used.



Frequent Graph Patterns. Figure 3. The CAM Tree of the graph *P* in Fig. 1. Every matrix obtained by a join operation is specified by a label starting with c. and then the type of the join operation e.g., c. 3a stands for join case3a. A CAM obtained by an extension operation is labeled with e.

Usually an empty matrix is considered as the maximal proper submatrix of any matrix with size 1. With this assumption, all the CAMs of connected subgraphs of a graph *G* may be organized as a rooted tree as follows:

- The root of the tree is an empty matrix;
- Each node in the tree is a distinct connected subgraph of *G*, represented by its CAM;
- For a given none-root node (with CAM *M*), its parent is the graph represented by *M*'s maximal proper submatrix;

The tree obtained in this fashion is denoted as the *CAM tree* of the graph *G*. Figure 3 shows the CAM tree of the graph *P* from Fig. 1.

Frequent Subgraph Mining of a Graph Database

With the CAM tree data structure, the space of subgraphs could be organized in a single CAM tree. If such a tree is built in advance (regardless of the required space and computational capacity), any traversal of the tree (depth-first, level-wise, random) reveals the set of distinct subgraphs of the graph database. For each of such subgraph, its support value may be determined by a linear scan of the graph database and finally frequent ones can be reported. This method clearly suffers from the huge number of available subgraphs in a graph database and hence has poor scalability to large databases.

In the following pseudo code, an algorithm which takes advantage of the following simple fact is presented: if a subgraph *G* is infrequent (support of *G* is less than a user posted threshold), none of its supergraphs are frequent. This suggests that an algorithm can stop building a branch of the tree as early as its finds out that the current node has insufficient support.

In the pseudo code below, symbol *CAM*(*G*) denotes the CAM of the graph *G*. *is CAM* is a function computes whether the matrix *X* is the CAM of the graph it represents or not.

FFSM

input: a graph database *GD* and a support threshold $f\,(0 < f \leq 1)$

output: set S of all *G*'s connected subgraphs.

```
$C \leftarrow$ {the CAMs of the frequent edges}
$F \leftarrow$ {the CAMs of the frequent nodes and edges}
FFSM-Explore ($C$,$F$);
```

FFSM-Explore

input: *C*, a suboptimal CAM list and *F*, a set of frequent connected subgraphs' CAMs

output: set *F* contains CAMs of all frequent subgraphs searched so far.

```
for $X \in C$ do
  if ($isCAM(X)$) then
    $F \leftarrow F \cup \{X\}$
    $C' \leftarrow \emptyset$
    for $Y \in C$ do
      $C' \leftarrow C' \cup$ FFSM-Join($X$,$Y$)
    end for
    $C' \leftarrow C' \cup$ FFSM-Extension ($X$)
    remove CAM(s) from $C'$ that is either
    infrequent or not suboptimial
    FFSM-Explore($C'$,$F$)
  end if
end for
```

Key Applications

Frequent subgraph mining has many applications. Below two applications in the emergent area of bioinformatics and cheminformatics are reviewed. Other applications of frequent subgraph mining could be found in recent reviews.

Pattern Discovery from Chemical Structures

Several investigators have applied the idea of frequent subgraph mining in cheminformatics. In these applications, a graph is used to model a chemical structure where a node represents an atom and an edge represents a chemical bond in the chemical structure. Frequent subgraph mining is then utilized to discover task-relevant and problem-specific features (*descriptors* as usually called in cheminformatics literature) in the chemical structures. Finally the features are utilized to build a predictive model to map chemical structures to their target properties [9].

Pattern Discovery from Protein Structures

Huan et al. have applied data mining algorithms to analyze 3D structures of biomoelcules including proteins and chemicals. Their approach adopted geometric graph representations of protein 3D structure using a geometric technique called Delaunay Tessellation and its recent extension to almost-Delaunay. These techniques produce sparser but information-preserving graph representations than conventional distance-based methods. With the graph database mining techniques, they have identified structure patterns that occur frequently in a family of proteins but rarely in other families, or *protein family-specific fingerprints*. It has been demonstrated that patterns obtained from comparing multiple structures have clear linkages to known functional features such as the catalytic sites in enzymes [3].

Cross-references

- ▶ Data Mining
- ▶ Graph Database
- ▶ Semi-Structured Data

Recommended Reading

1. Han J., Cheng H., Xin D., and Yan X. Frequent pattern mining: current status and future directions. Data Min. Knowl. Discov., 14, 2007.
2. Huan J., Prins J., Wang W., Carter C., and Dokholyan N.V. Coordinated evolution of protein sequences and structures with structure entropy. Tech. Rep. Computer Science Department, 2006.
3. Huan J., Wang W., Bandyopadhyay D., Snoeyink J., Prins J., and Tropsha A. Mining protein family specific residue packing patterns from protein structure graphs. In Proc. 8th Annual Int. Conf. on Research in Computational Molecular Biology, 2004, pp. 308–315.
4. Huan J., Wang W., and Prins J. Efficient mining of frequent subgraph in the presence of isomorphism. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 549–552.
5. Huan J., Wang W., Prins J., and Yang J. SPIN: mining maximal frequent subgraphs from graph databases. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 581–586.
6. Inokuchi A., Washio T., and Motoda H. An apriori-based algorithm for mining frequent substructures from graph data. In Principles of Data Mining and Knowledge Discovery, 4th European Conf., 2000, pp. 13–23.
7. Kuramochi M. and Karypis G. Frequent subgraph discovery. In Proc. 2001 IEEE Int. Conf. on Data Mining, 2001, pp. 313–320.
8. Nijssen S. and Kok J. A quickstart in frequent structure mining can make a difference. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 647–652.
9. Smalter A., Huan J., and Lushington G. Structure-based pattern mining for chemical compound classification. In Proc. 6th Asia Pacific Bioinformatics Conf, 2008, pp. 39–48.
10. Vanetik N. and Gudes E. Mining frequent labeled and partially labeled graph patterns. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 91–102.
11. Yan X. and Han J. gSpan: graph-based substructure pattern mining. In Proc. 2002 IEEE Int. Conf. on Data Mining, 2002, pp. 721–724.

Frequent Items on Streams

AHMED METWALLY
Google Inc., Mountain View, CA, USA

Synonyms

Frequent elements; Heavy hitters; Hot items

Definition

Frequent items are the items that mostly represent the stream, since these are the items that occur more than a given user threshold. Formally, given a stream, S, of size N from an alphabet, A, a *frequent item*, $E_i \in A$, is an item whose *frequency*, or number of occurrences, F_i exceeds a specific user support ϕN, where $0 \leq \phi \leq 1$. There cannot be more than $\lfloor \frac{1}{\phi} \rfloor - 1$ such items. Finding the frequent items exactly in one pass requires $O(min(A,N))$ in-memory space [6]. Frequent items can be defined on the entire stream or on a sliding window of fixed or variable size (see Stream Models). Similarly, frequent items can be defined on append-only streams as well as streams with item deletions (see Stream Mining).

Historical Background

Even before the stream processing model was proposed, the early work in [3], and [9] searches for a majority item that occur more than $\frac{N}{2}$ times. This work was extended in [15] to search for $\lfloor \frac{1}{\phi} \rfloor - 1$ items, each occurring more than ϕN times.

Foundations

Due to the high cost of the exact solution of the frequent items problem, researchers proposed approximate problem definitions. This section discusses the problem approximations and their proposed algorithms.

Problem Approximations

The approximate problem definition in [3,9,15] is to report $\lfloor \frac{1}{\phi} \rfloor - 1$ items regardless of the number of items that really satisfy the frequency constraint. However, all the really frequent items should be among the reported set of items. This introduces *false positives*, which are reported items whose frequencies do not really exceed ϕN.

The *hot items* problem [6], is to report the frequent items with high probability. This raises the issue of having both false positives and *false negatives*, where a false negative is a frequent item that is not reported.

Finding the ε-*deficient* frequent items was proposed by Manku and Motwani [13] to report all the items with frequencies exceeding $\lceil \phi N \rceil$. In addition, no reported item can have a frequency less than $\lceil (\phi - \varepsilon) N \rceil$, where $\varepsilon \in [0, \phi]$ is a user-defined error. The estimated frequencies of the reported items also have to be reported.

Proposed Algorithms

Several algorithms [6,7,8,10,11,13,14] were proposed to solve the approximate frequent items problems. In [14], these algorithms are classified into *Counter-based* and *Sketch-based* algorithms.

Counter-Based Techniques

Counter-based algorithms monitor the frequency of a subset of A. They keep a counter for each monitored item. The counter of a monitored item, E_i, is updated whenever E_i is observed in the stream. The algorithms mainly differ in how they handle non-monitored items.

The *Sticky Sampling* algorithm [13] solves the ε-deficient frequent items probabilistically. It slices S into rounds of exponentially increasing length. The probability an item is sampled, i.e., starts to be monitored, at any round, r, is $\frac{1}{r}$. At rounds' boundaries, for every monitored item, a coin is tossed until a success occurs. The counter is decremented for every unsuccessful toss, and is deleted if it reaches 0. Since decrementing counters follow a geometric distribution, the probability of sampling an item is constant throughout S. *Sticky Sampling* has a space bound of $O(\frac{2}{\epsilon}\ln(\phi^{-1}\delta^{-1}))$, where δ is the failure probability.

The *Lossy Counting* algorithm [13] solves the ε-deficient frequent items with success probability of 1, though with a space bound greater than that of *Sticky Sampling*. S is broken up into rounds of equal length, $\frac{1}{\epsilon}$. Throughout every round r, non-monitored items are added to the monitored set. When a new item is added in round r, it is given the benefit of doubt. Its initial count is set to $r - 1$, and the maximum possible overestimation, $r - 1$, is recorded for the new item. Thus, *Lossy Counting* guarantees that monitored items can have their frequencies overestimated by no more than εN, and never underestimated. At the end of each ound, r, every item, E_i, whose estimated frequency s less than r is evicted in order to reduce the space footprint. *Lossy Counting* [13] has a space bound of $O(\frac{1}{\epsilon}\ln(\epsilon N))$.

The *Frequent* algorithm [7] solves the problem of [15], which asks for a maximum of $\lfloor \frac{1}{\phi} \rfloor - 1$ items, each of which has frequency exceeding ϕN. *Frequent*, a rediscovery of the algorithm proposed by [15], outputs a list of exactly $\frac{1}{\phi} - 1$ items with no guarantee on which items, if any, have frequency more than ϕN. This algorithm extends the early work done in [3,9] for finding a majority item using only one counter. The algorithm in [3,9] monitors the first item in the stream. For each observation, the counter is incremented if the observed item is the monitored one, and is decremented otherwise. If the counter reaches 0, it is assigned the next observed item, and the algorithm is then repeated. When the algorithm terminates, the monitored item is the candidate majority item. *Frequent* [7] keeps $\frac{1}{\phi} - 1$ counters to monitor items. If a monitored item is observed, its counter is incremented, else all counters are decremented in an $O(1)$ operation, using a lightweight data structure. In case any counter reaches 0, it is assigned the next observed item. The discussion assumes hashing takes constant time. If the data structure is not used, decrementing all counters has an $O(1)$ amortized cost. The algorithm uses only $\frac{1}{\phi} - 1$ counters, but reports back all the monitored items, which could yield a high ratio of false positives. The same algorithm and data structure were proposed independently in [11].

The *Space-Saving* algorithm [14] solves the ε-deficient frequent items with success probability of 1 with a space bound of $\frac{1}{\epsilon}$, and gives the same guarantees on overestimation error of *Lossy Counting*. Moreover, the guaranteed error rate, ε, provably decreases with the increase in data skew. If the observed item, x, is monitored, its counter is incremented. If x is not monitored, i.e., no counter is assigned to x, then the item that currently has the least estimated hits, *min*, is evicted and the counter is assigned to x. The intuition behind replacing the item with the least hits is to sacrifice the least amount of information about the stream history. Since the actual hits of x can be any number between 1 and $min + 1$, *Space-Saving* assumes x has been observed $min + 1$ times, since it gives items the benefit of doubt not to underestimate frequencies. Like *Lossy Counting*, for each monitored item the algorithm keeps track of its maximum possible over-estimation resulting from the initialization of its counter when starting monitoring it. To guarantee constant processing time per observation, the item with the least estimated hits has to be identified in

$O(1)$ time. To do that, *Space-Saving* uses a data structure similar to that in [7] in order to keep the counters always sorted by their estimated hits. However, not using the data structure still guarantees an amortized constant time per stream observation.

In general, counter-based techniques have fast per-item processing since counters can be stored in hash tables.

Sketch-Based Techniques

Sketch-based techniques do not monitor a subset of items, but rather provide, with less stringent guarantees, frequency estimation for all items by using arrays of counters. Usually, each counter monitors the frequencies of a subset of the item domains. Similarly, each item is hashed into the space of counters using a family of hash functions, and the hashed-to counters are updated for every hit of this item. These "representative" counters are then queried for the item frequency.

The *GroupTest* algorithm [6] solves the hot items problem, with a probability of failure of δ. The *FindMajority* algorithm was first devised to detect the majority item, by keeping a system of a global counter and $\lceil log(|A|) \rceil$ counters. Assuming that items' IDs are in the range $1...|A|$. A hit to item E is handled by updating the global counter, and all counters whose index corresponds to a 1 in the binary representation of E. At any time, counters whose value are more than half the global counter correspond to the 1s in the binary representation of the candidate majority item, if it exists. *GroupTest* is a simple generalization of *FindMajority*, which keeps only $O(\frac{\frac{1}{\phi}-1}{\delta}\log(\frac{1}{\phi}-1))$ of such systems, and uses a family of universal hash functions to map each item to $O(\frac{\log(\frac{1}{\phi}-1)}{\delta})$ *FindMajority* systems that monitor the occurrences of the item. When queried, the algorithm discards systems with more than one, or with no hot items. Also proposed is an elegant scheme for suppressing false positives by checking that all the systems a hot item belongs to report this item as being hot. *GroupTest* is generally accurate and can handle streams with both item insertions and deletions. However, it offers no information about items' frequencies or order.

The *Multistage Filters* approach, proposed by [8], which was also independently proposed by [10], is similar to *GroupTest*. The *Multistage Filters* algorithm hashes every item to a number of counters that are updated every time the item is observed in the stream. The item is considered to be frequent if the smallest of its representative counters satisfies the user required support. The algorithm by [8] judges an item to be frequent or not while updating its counters. If a counter is estimated to be frequent, it is added to a specialized set of counters for monitoring frequent items, the *flow memory*. To decrease the false positives, [8] proposed some techniques to reduce the over-estimation errors in counters. First, once an item is added to the flow memory, its counters are not monitored anymore by the *Multistage Filters*. Second, the algorithm increments only the counter(s) of the minimum value. Without the second error-reduction technique, the algorithm can be used for streams of both item insertions and deletions.

The *hCount* algorithm [10], does not employ the error reduction techniques employed by [8]. However, it keeps a number of imaginary items, which have no hits. At the end of the stream, all the items in the alphabet are checked for being frequent, and the over-estimation error for each of the items is estimated to be the average number of hits for the imaginary items. Both the *hCount* algorithm [10], and the *Multistage Filters* [8] require a number of counters bounded by $\frac{e}{\epsilon} * \ln\left(\frac{-|A|}{\ln\delta}\right)$.

In general, sketch-based techniques are not affected by the ordering of items in the stream, and are usually capable of handling streams with item insertions and deletions. On the other hand, they are probabilistic, and an item observation or a query about its frequency entails calculations across several counters.

Generalization for Sliding Windows

In [1], generalizations of the algorithm in [15], and *Sticky Sampling* were proposed for sliding windows of fixed and variable sizes (see Stream Models). Limiting the discussion to the deterministic [15] algorithm and the fixed size sliding window, the generalization monitors the stream at several levels of exponentially increasing granularities. That is, the finest granularity level, *level*-0, keeps copies of the algorithm that tracks the frequent items in sub-windows of size $f(W, \varepsilon)$, where W is the fixed size of the sliding window. These sub-windows are non-overlapping and contiguous such that their union is equivalent to the whole sliding window, modulo the oldest sub-window that contains expired items. The second finest granularity, *level*-1, keeps copies of the algorithm that tracks the frequent items in sub-windows of size $2 \times f(W, \varepsilon)$, and so forth. A boundary of a sub-window at any level has to cooccur with a boundary at the next finer level. Due to keeping several levels, and keeping several algorithm copies at each level, the space requirement is $O(\frac{1}{\epsilon}(\log(\epsilon N))^2)$.

The case for variable size sliding window was handled essentially by forming new levels of coarser granularity from existing levels when the window size multiplies by a factor of 2. The requirement here is to be able to merge two copies of the algorithm, each has observed n items, to form a copy of the algorithm that is equivalent to observing the entire 2 × n items. If the window shrinks to be a smaller size than the granularity of the coarsest level, the coarsest level is dropped.

At query time, the results of the algorithm copies at all levels are combined together to discover the frequent items in the entire sliding window. This requires the ability to combine the output of several algorithm copies while maintaining the error guarantees of the algorithm. The combining process does not include algorithm copies monitoring sub-windows containing expired items, does not include algorithm copies monitoring items that are already combined, and favors copies of the algorithm at the coarser levels.

The generalization of [15] to sliding windows at [12] does not keep several copies of the algorithm. Instead, the counters are incremented in a way that expires occurrences of the item that are older than the sliding window boundary. The space requirement is still $O(\frac{1}{\epsilon})$, thought the time for processing each element is also $O(\frac{1}{\epsilon})$. The case for variable size sliding window was handled essentially by adding new levels when the window size multiplies by a factor of 2. The requirement here is to be able to from a copy of a sliding window algorithm with error $2 \times \varepsilon n$ from another copy with error εn, where n is the number of items observed by the original copy. Like [1], if the window shrinks to be a smaller size than the granularity of the coarsest level, the coarsest level is dropped. The work done by Lee and Ting [12] was recently improved further in [16] to require $O(1)$ for processing each stream element using a doubly circularly-linked list to link the occurrences of the items sorted by their expiration time. Hence, the algorithm can expire occurrences of the items more efficiently.

Key Applications

Current motivation of the stream frequent items problem include network traffic analysis which is important for caching, routing, accounting, detecting network-level attacks, and maintenance [7,8]. In [14], the problem was applied for the application of increasing the revenue of Internet advertising networks. Motivated by these networks applications, [2] implemented *Space-Saving* and *Lossy Counting* on network processing units (NPUs), a special networking architecture with associative memories. In addition, [13] considered using frequent items queries to alleviate finding frequent itemsets (see Association rules), as well as iceberg queries and iceberg cubes. In specific, [5] generalized the problem to the case of multi-dimensional items in the presence of domain hierarchies (see Hierarchical heavy hitter mining on streams).

Experimental Results

Recently, Cormode and Hadjieleftheriou [4] conducted experiments comparing the most promising algorithms on both real and synthetic data, and noticed the competitive benefits of *Space-Saving* when processing append-only streams. In the case of streams with deletions, different sketch-based algorithms had different advantages.

In [13], the two proposed algorithms, *Lossy Counting* and *Sticky Sampling*, were compared on synthetic Zipfian data. *Lossy Counting* perform bettered even though the theoretical space bound of *Sticky Sampling* is better. However, the experiments were run with relatively small alphabets.

In [6], the algorithms *GroupTest*, *Lossy Counting*, and *Frequent* were compared on both real telephone connection and synthetic data. The data had both item insertions and deletions. Both *Lossy Counting* and *Frequent* were made to handle item deletions by decrementing the counters of the deleted items if such counters exist. Under these conditions, on synthetic data *GroupTest* performed best, followed by *Frequent*, and then *Lossy Counting* had the least precision and recall (the highest false positives and negatives). However, *GroupTest* used approximately 3× the space used by *Lossy Counting*, and 6× the space used by *Frequent*. On real data, *Lossy Counting* performed better than *Frequent*, but *GroupTest* was more accurate than both.

In [8], the proposed techniques were compared against commercial sampling-based algorithms for traffic measurement on real Internet backbone traffic traces. The results favored the *Multistage Filters* technique.

In [14], the algorithms *Space-Saving*, *GroupTest*, and *Frequent* were compared on real Internet advertising traffic as well as synthetic data, both with large alphabets. Throughout the experiments, *Frequent* attained a precision of less than 0.2. *Space-Saving* was the only algorithm that neither reported false positives nor reported false negatives, even though it consumed roughly the same space of *Frequent*. On real data, *Space-Saving* consumed approximately one fifth of the space used by *GroupTest*, and on synthetic data, *Space-Saving* consumed at most one eighth of the space used by *GroupTest*.

Nagender Bandi et al. [2] compared how the hardware and software implementations of *Space-Saving* and *Lossy Counting* perform. The results of *Lossy Counting* conformed with those in [13]. *Lossy Counting* can use space much smaller than its bounds under small alphabets. *Lossy Counting* used less space than *Space-Saving* when the data was skewed and the skew was not estimated. However, *Space-Saving* used less space when the data skew was estimated since it monitored less items. The hardware implementation of *Space-Saving* ran significantly faster than *Lossy Counting*, while the software implementation ran only slightly faster.

Cross-references

▶ Hierarchical Heavy Hitter Mining on Streams
▶ Histograms on Streams
▶ Quantiles on Streams
▶ Stream Mining
▶ Stream Models
▶ Stream Sampling

Recommended Reading

1. Arasu A. and Manku G. Approximate counts and quantiles over sliding windows. In Proc. 23rd ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2004, pp. 286–296.
2. Bandi N., Metwally A., Agrawal D., and Abbadi A.E. Fast data stream algorithms using associative memories. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 247–256.
3. Boyer R. and Moore J. A Fast Majority Vote Algorithm. Tech. Rep. 1981-32, Institute for Computing Science, University of Texas, Austin, 1981.
4. Cormode G. and Hadjieleftherion M. Finding Frequent Items in Data Streams. Proc. VLDB, 1(2):1530–1541, 2008.
5. Cormode G., Korn F., Muthukrishnan S., and Srivastava D. Diamond in the rough: finding hierarchical heavy hitters in multi-dimensional data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 155–166.
6. Cormode G. and Muthukrishnan S. What's Hot and What's Not: Tracking Most Frequent Items Dynamically. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2003, pp. 296–306, an extended version appeared in ACM Trans. on Comput. Syst., 30(1):249–278, 2005.
7. Demaine E., López-Ortiz A., and Munro J. Frequency estimation of internet packet streams with limited space. In Proc. 10th ESA European Symposium on Algorithms, 2002, pp. 348–360.
8. Estan C. and Varghese G. New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice. ACM Trans. Comput. Syst., 21(3):270–313, 2003.
9. Fischer M. and Salzberg S. Finding a Majority Among N Votes: Solution to Problem 81-5. J. Algorithms, 3:376–379, 1982.
10. Jin C., Qian W., Sha C., Yu J., and Zhou A. Dynamically maintaining frequent items over a data stream. In Proc. Int. Conf. on Information and Knowledge Management, 2003, pp. 287–294.
11. Karp R., Shenker S., and Papadimitriou C. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. ACM Trans. Database Syst., 28(1):51–55, 2003.
12. Lee L. and Ting H. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 290–297.
13. Manku G. and Motwani R. Approximate frequency counts over data streams. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 346–357.
14. Metwally A., Agrawal D., and El Abbadi A. Efficient computation of frequent and top-k elements in data streams. In Proc. 10th Int. Conf. on Database Theory, 2005, pp. 398–412, an extended version appeared in ACM Trans Database Syst., 31(3):1095–1133, 2006.
15. Misra J. and Gries D. Finding repeated elements. Sci. Comput. Program., 2:143–152, 1982.
16. Zhang L. and Guan Y. Frequency estimation over sliding windows. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 1385–1387.

Frequent Itemset Mining with Constraints

CARSON KAI-SANG LEUNG
University of Manitoba, Winnipeg, MB, Canada

Synonyms

Constrained frequent itemset mining; Frequent pattern mining with constraints; Frequent set mining with constraints

Definition

Let $\mathtt{Item} = \{ item_1, item_2, \ldots, item_m\}$ be a set of domain items, where each item represents an object in a specific domain. Each object is associated with some attributes or auxiliary information about the object. A transaction $t_i = \langle tID, I_i \rangle$ is a tuple, where *tID* is a unique identifier and $I_i \subseteq \mathtt{Item}$ is a set of items. A set of items is also known as an *itemset*. A transaction database *TDB* is a collection of transactions. An itemset *S* is *contained* in a transaction $t_i = \langle tID, I_i \rangle$ if $S \subseteq I_i$.

The *support* (or *frequency*) of an itemset *S* in a database *TDB* is the number (or percentage) of transactions in *TDB* containing *S*. An itemset is *frequent* if its support exceeds or equals a user-specified support threshold *minsup*. A user-specified constraint *C* is a predicate on the powerset of `Item` (i.e., $C: 2^{\texttt{Item}} \mapsto \{\text{true, false}\}$). An itemset *S satisfies* a constraint *C* if *C*(*S*) evaluates to true. Hence, given (i) a transaction database *TDB*, (ii) a user-specified support threshold *minsup* and (iii) user-specified constraints *C*, the problem of *frequent itemset mining with constraints* is to find from *TDB* a set of frequent itemsets that satisfy *C*.

Historical Background

The research problem of frequent itemset mining was first introduced by Agrawal et al. [1] for market basket analysis in the context of association rule mining. Specifically, frequent itemset mining, which aims to find frequent itemsets from a transaction database, is an important first step of association rule mining. Once the frequent itemsets are found, they are used in the second step to generate association rules. The rules reveal the buying patterns in consumer behavior. For instance, they tell how the presence of some itemsets is associated with the presence of some other itemsets in the shopping baskets of consumers. This information is useful in making decisions in applications such as customer targeting, shelving, and sales promotion.

Besides the mining of association rules, frequent itemset mining also plays an essential role in many important data mining tasks – such as the mining of maximal itemsets, closed itemsets, correlation, causality, and sequential patterns. This explains why frequent itemset mining has been the subject of numerous studies since its introduction. Most of these studies focused either on improving efficiency of frequent itemset mining (e.g., the Apriori frequent itemset mining framework [2] and its tree-based counterpart [7]) or on extending the initial notion of frequent itemsets to other patterns such as maximal itemsets, closed itemsets, and correlated itemsets. Regardless whether they focused on these performance or functionality issues, these studies basically considered the data mining exercise in isolation. They relied on a computational model in which the computer does almost everything and the user is *not* engaged in the mining process. In other words, this model does not explore how data mining can interact with the human user. Consequently, the model provides little or no support for user focus.

However, in many applications, it is not uncommon for the user to have certain broad phenomena in mind, on which to focus mining. Without user focus, the mining process is treated as an impenetrable black-box – only allowing the user to set the support threshold *minsup* at the beginning and to get all the frequent itemsets at the end. The user does not have the opportunity to specify his interest via the use of constraints. As a result, the user often needs to wait for a long period of time for numerous frequent itemsets, out of which only a tiny fraction may be interesting to the user. This motivates the introduction of the research problem of *frequent itemset mining with constraints*, which gives the user opportunities to express his focus in mining by means of a rich set of constraints that captures application semantics.

Foundations

When compared with its traditional unconstrained counterpart, *frequent itemset mining with constraints* provides user focus in the sense that the user has opportunities to express his interest – via the use of constraints. By using the constraints, mining can be focused and computation can be saved. Over the past decade, several studies on frequent itemset mining with constraints have been proposed. For example, Ng, Lakshmanan and their colleagues [8,9,11] proposed a framework and algorithms for mining frequent itemsets with constraints. Within their framework, the user can use a rich set of constraints – which captures the semantics of itemsets – to guide the mining process for finding only those frequent itemsets that satisfy the constraints. These constraints include SQL-style aggregate constraints as well as domain constraints. The *SQL-style aggregate constraints* are of the following form:

$$agg(S.attribute)\ \theta\ constant,$$

where *agg* is an SQL-style aggregate function (e.g., *min*, *max*,*sum*,*avg*) and θ is a Boolean comparison operator (e.g., $=,\neq,<,\leq,\geq,>$). For example, by specifying the aggregate constraint $min(S.Price) \geq \$20$, the user expresses his interest in finding every itemset *S* such that the minimum price of all items in *S* is at least $20. Other examples of aggregate constraints include $avg(S.Temperature) < -18^{\circ}\mathrm{C}$ (which expresses that the average temperature of items in *S* is below $-18^{\circ}\mathrm{C}$) and $max(S.Qty) > 15$ (which expresses that the maximum quantity of items in *S* is more than 15). *Domain* *constraints*, which are non-aggregate constraints, can be of the following forms:

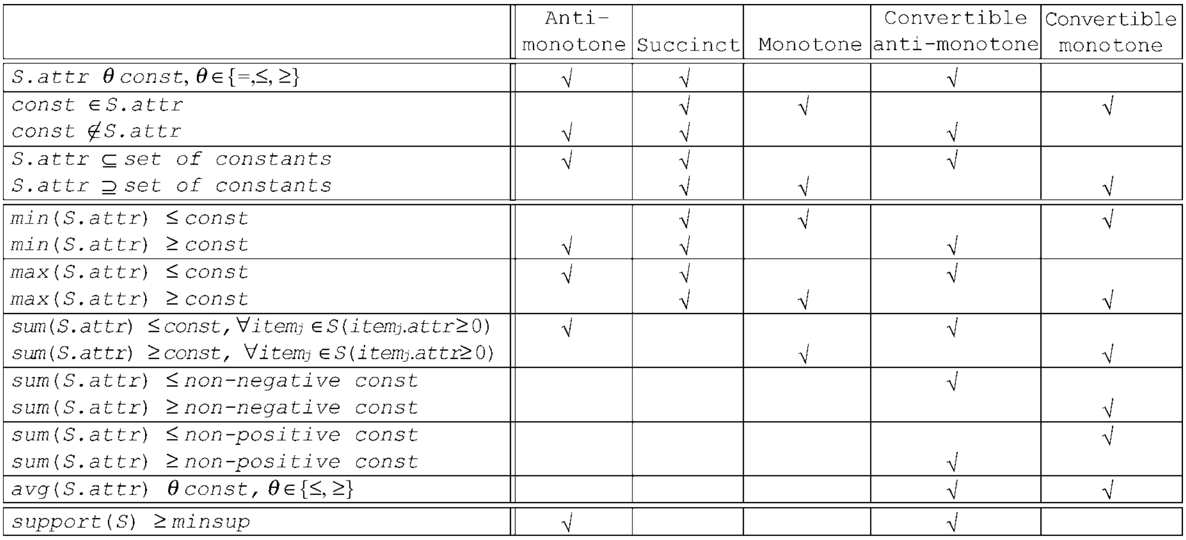
1. *S.attribute* θ *constant*, where θ is a Boolean comparison operator (e.g., $=,\neq,<,\leq,\geq,>$);
2. *constant* $\in$*S.attribute*;
3. *constant* $\notin$ *S.attribute*; or
4. *S.attribute* φ *set of constants*, where φ is a set comparison operator (e.g., $\subseteq,\not\supseteq,\subset,\not\subset,=,\neq,\supseteq,\not\supseteq,\supset,\not\supset$).

Examples of domain constraints include *S.ManufactureYear* $\neq$ 2008 (which expresses that each item in *S* is not manufactured in 2008), 2kg $\in$*S.Weight* (which expresses that a 2kg-item must be in *S*) and *S.Type* $\supseteq$ {*snack*, *soda*} (which expresses that *S* must contain some snacks and soda). These aggregate constraints and domain constraints can be categorized into several overlapping classes – such as *anti-monotone constraints*, *succinct constraints*, *monotone constraints*, and *convertible constraints* – based on properties of constraints. Figure 1 shows the characterization of some commonly used constraints.

To find frequent itemsets that satisfy the aforementioned constraints, Ng, Lakshmanan and their colleagues proposed algorithms – called *CAP* (Constrained Apriori) [11] and *DCF* (Dynamic Constrained Frequent-set computation) [8] – that exploit properties of these constraints. By doing so, both CAP and DCF ensure that the computational effort in mining frequent itemsets satisfying the constraints is proportional to the selectivity of the constraints. For instance, both CAP and DCF algorithms exploit a nice property of *anti-monotone constraints*: For an anti-monotone constraint C_{am} (e.g., *min*(*S.Price*) $\geq$ \$20, *S.ManufactureYear* $\neq$ 2008), if an itemset *S* does not satisfy C_{am} then supersets of *S* are guaranteed not to satisfy C_{am}. Hence, whenever *S* does not satisfy C_{am}, both CAP and DCF do not need to generate any superset of *S* as a candidate and thus do not need to count its support (or frequency). Consequently, by pushing C_{am} into the mining process, the mining algorithms save computation.

Note that many algorithms for mining frequent itemsets with or without constraints (e.g., Apriori [2], CAP [11] and DCF [8]) have also been exploiting a special anti-monotone constraint – namely, the *frequency constraint support*(*S*) $\geq$*minsup*, which states that the support (or frequency) of a frequent itemset *S* should equal or exceed the user-specified support threshold *minsup*. If *S* is infrequent, then the mining algorithms do not consider supersets of *S* because they are guaranteed to be infrequent. This property is commonly known as the Apriori property.

In addition to anti-monotone constraints, both CAP and DCF algorithms also exploit *succinct constraints* (e.g., *min*(*S.Price*) $\geq$ \$20, 2kg $\in$ *S.Weight*). For frequent itemset mining without constraint, it is well known that mining algorithms based on the Frequent-Pattern tree (FP-tree) [7] outperform their



	Anti-monotone	Succinct	Monotone	Convertible anti-monotone	Convertible monotone
$S.attr\ \theta\ const, \theta \in \{=,\leq,\geq\}$	√	√		√	
$const \in S.attr$		√	√		√
$const \notin S.attr$	√	√		√	
$S.attr \subseteq set\ of\ constants$	√	√		√	
$S.attr \supseteq set\ of\ constants$		√	√		√
$min(S.attr) \leq const$		√	√		√
$min(S.attr) \geq const$	√	√		√	
$max(S.attr) \leq const$	√	√		√	
$max(S.attr) \geq const$		√	√		√
$sum(S.attr) \leq const, \forall item_j \in S(item_j.attr \geq 0)$	√			√	
$sum(S.attr) \geq const, \forall item_j \in S(item_j.attr \geq 0)$			√		√
$sum(S.attr) \leq non\text{-}negative\ const$				√	
$sum(S.attr) \geq non\text{-}negative\ const$					√
$sum(S.attr) \leq non\text{-}positive\ const$					√
$sum(S.attr) \geq non\text{-}positive\ const$				√	
$avg(S.attr)\ \theta\ const, \theta \in \{\leq,\geq\}$				√	√
$support(S) \geq minsup$	√			√	

Frequent Itemset Mining with Constraints. Figure 1. Characterization of commonly used domain, aggregate and frequency constraints.

Apriori-based counterparts. As both CAP and DCF are Apriori-based, Leung et al. [10] proposed an FP-tree based algorithm – called *FPS* (*FP*-tree based mining of *S*uccinct constraints) – for mining frequent itemsets that satisfy succinct constraints. By pushing the succinct constraint C_{suc} into the mining process, all three algorithms (CAP, DCF and FPS) directly generate precisely all and only those itemsets that satisfy C_{suc} by using a precise "formula" called a *member generating function* (*MGF*). As a result, there is no need to generate and then exclude itemsets not satisfying C_{suc}. For example, itemsets satisfying the succinct constraint $min(S.Price) \geq \$20$ can be generated by first selecting items with price at least \$20 from the domain `Item` and then combining these selected items:

$$\{X | X \subseteq \sigma_{Price \geq \$20}(\text{Item}),\ X \neq \emptyset\}.$$

As another example, itemsets satisfying the succinct constraint 2kg $\in S.Weight$ can be generated by combining at least one 2kg-item (i.e., at least one mandatory item) with some optional items of any weight:

$$\{Y \cup Z | Y \subseteq \sigma_{Weight=2\text{kg}}(\text{Item}),\\ Y \neq \emptyset,\ \ Z \subseteq \sigma_{Weight \neq 2\text{kg}}(\text{Item})\}.$$

As the third example, itemsets satisfying the succinct constraint $S.Type \supseteq \{snack, soda\}$ can be generated by combining some snacks and some soda (which are mandatory items) with some optional items of other types:

$$\{X \cup Y \cup Z | X \subseteq \sigma_{Type=snack}(\text{Item}),\ X \neq \emptyset,\\ Y \subseteq \sigma_{Type=soda}(\text{Item}),\ Y \neq \emptyset,\\ Z \subseteq \sigma_{Type \neq snack \wedge Type \neq soda}(\text{Item})\}.$$

Among the succinct constraints in these three examples, the first one is also anti-monotone but the last two are not. In general, itemsets satisfying a succinct and anti-monotone constraint can be generated using only mandatory items (e.g., those with price $\geq$ \$20), whereas itemsets satisfying succinct but not anti-monotone constraints require both mandatory items and optional items.

Besides anti-monotone constraints and succinct constraints, there have been studies that handle other classes of constraints. For instance, Grahne et al. [6] exploited *monotone constraints* when finding correlated frequent itemsets. Since supersets of any itemset S satisfying a monotone constraint C_m (e.g., 2kg $\in$ *S. Weight*) are guaranteed to satisfy C_m, Grahne et al. pushed C_m into the mining process so that they do not need to perform further constraint checking on any superset of S once S satisfies C_m. Hence, computation is saved. Bucila et al. [4] proposed a dual mining algorithm – called *DualMiner* – that exploits both anti-monotone constraints and monotone constraints simultaneously to find itemsets satisfying the constraints.

Knowing that some constraints such as avg(*S. Temperature*) $< -18°$C are not anti-monotone or monotone in general, Pei and his colleagues [12,13] converted these "tough" constraints into anti-monotone constraints or monotone constraints by sorting items in each transaction in some order. They proposed the $\mathcal{FIC}$ algorithms (mining *F*requent *I*temsets with *C*onvertible constraints) to handle these *convertible constraints*. Specifically, the $\mathcal{FIC}^{\mathcal{A}}$ algorithm deals with *convertible anti-monotone constraints*, and the $\mathcal{FIC}^{\mathcal{M}}$ algorithm deals with *convertible monotone constraints*. For example, by arranging items in non-descending order of temperature, $\mathcal{FIC}^{\mathcal{A}}$ does not need to consider an itemset S if its prefix does not satisfy a convertible anti-monotone constraint C_{cam}. This is because S is guaranteed not to satisfy C_{cam} whenever the prefix of S does not. Similarly, by arranging items in non-ascending order of temperature, $\mathcal{FIC}^{\mathcal{M}}$ does not need to perform further constraint checking on an itemset S' if its prefix satisfies a convertible monotone constraint C_{cm}. This is because S' is guaranteed to satisfy C_{cm} whenever the prefix of S' does. Along this research direction, Bonchi and Lucchese [3] exploited "tough" constraints involving the aggregate functions variance and standard deviation.

Besides the aforementioned domain and aggregate constraints, the user can also express his interest by specifying other constraints. For example, Srikant et al. [14] considered *item constraints* that allow the user to impose a Boolean expression over the presence or absence of items in the itemset. Gade et al. [5] mined closed frequent itemsets that satisfy *block constraints*, which determine the significance of an itemset S by considering the dense blocks formed by items within S and by transactions associating with S. Yun and Leggett [15] proposed an algorithm for mining *weighted* frequent itemsets with *length decreasing support constraints*.

Key Applications

Frequent itemset mining – with or without constraints – plays an essential role in the mining of various patterns and relationships, which include maximal itemsets, closed itemsets, association rules, correlation, causality, sequential patterns, episodes, partial periodicity, emerging patterns, as well as frequent structures and trends. Moreover, frequent itemset mining is also useful in many data mining tasks such as associative classification, outlier detection, iceberg-cube computation, and stream mining. The knowledge discovered from frequent itemset mining can reveal important information in many real-life applications. Examples of these applications include market basket analysis (e.g., modeling of customer purchase behaviors, customer targeting, shelving, sales promotion), bioinformatics (e.g., order-preserving clustering of microarray data), Web mining (e.g., mining of Web contents, Web structures, or Web usages), mining for software reliability (e.g., software bug mining), as well as network management and intrusion detection (e.g., finding frequent routing paths, detecting signatures for intrusions).

Data Sets

Data sets commonly used for experimental evaluation for frequent itemset mining with constraints are similar to those used for frequent itemset mining *without* constraints. These data sets include the following:

1. IBM synthetic data generated by a data generator program developed at the IBM Almaden Center [2] (www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html)
2. Data from the UCI Machine Learning Repository (archive.ics.uci.edu/ml/)
3. Data from the Frequent Itemset Mining Dataset Repository (fimi.cs.helsinki.fi/data/)
4. Real-world data sets from the KDD Cup 2000, where the data sets contain purchase data from a real online retailer (www.sigkdd.org/kddcup/index.php?section=2000&method=task).

Cross-references

▶ Approximation of Frequent Itemsets
▶ Association Rule Mining on Streams
▶ Classification by Association Rule Analysis
▶ Closed Itemset Mining and Non-redundant Association Rule Mining
▶ Data Mining
▶ Frequent Itemsets and Association Rules
▶ Sequential Patterns
▶ Stream Mining

Recommended Reading

1. Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 207–216.
2. Agrawal R. and Srikant R. Fast algorithms for mining association rules. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
3. Bonchi F. and Lucchese C. Pushing tougher constraints in frequent pattern mining. In Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conf., 2005, pp. 114–124.
4. Bucila C., Gehrke J., Kifer D., and White W. DualMiner: a dual-pruning algorithm for itemsets with constraints. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 42–51.
5. Gade K., Wang J., and Karypis G. Efficient closed pattern mining in the presence of tough block constraints. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 138–147.
6. Grahne G., Lakshmanan L.V.S., and Wang X. Efficient mining of constrained correlated sets. In Proc. 16th Int. Conf. on Data Engineering, 2000, pp. 512–521.
7. Han J., Pei J., and Yin Y. Mining frequent patterns without candidate generation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 1–12.
8. Lakshmanan L.V.S., Leung C.K.-S., and Ng R.T. Efficient dynamic mining of constrained frequent sets. ACM Trans. Database Syst., 28(4):337–389, 2003.
9. Lakshmanan L.V.S., Ng R., Han J., and Pang A. Optimization of constrained frequent set queries with 2-variable constraints. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 157–168.
10. Leung C.K.-S., Lakshmanan L.V.S., and Ng R.T. Exploiting succinct constraints using FP-trees. ACM SIGKDD Explor., 4(1):40–49, 2002.
11. Ng R.T., Lakshmanan L.V.S., Han J., and Pang A. Exploratory mining and pruning optimizations of constrained associations rules. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 13–24.
12. Pei J. and Han J. Can we push more constraints into frequent pattern mining? In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 350–354.
13. Pei J., Han J., and Lakshmanan L.V.S. Mining frequent item sets with convertible constraints. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 433–442.
14. Srikant R., Vu Q., and Agrawal R. Mining association rules with item constraints. In Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining, 1997, pp. 67–73.
15. Yun U. and Leggett J. WLPMiner: weighted frequent pattern mining with length-decreasing support constraints. In Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conf., 2005, pp. 555–567.

Frequent Itemsets and Association Rules

Hong Cheng, Jiawei Han
University of Illinois at Urbana-Champaign, Urbana, IL, USA

Synonyms

Frequent Patterns; Large Itemsets

Definition

Let $I = \{ i_1, i_2, \ldots, i_n\}$ be a set of items, and $DB = \{ T_1, T_2, \ldots, T_m\}$ be a transaction database, where T_i ($i \in [1\ldots m]$) is a transaction containing a set of items in I. The *support* (or occurrence frequency) of an itemset A, where A is a set of items from I, is the number of transactions containing A in DB. An itemset A is *frequent* if A's support is no less than a user-specified *minimum support threshold* θ. An itemset A which contains k items is called a k-itemset.

Historical Background

Frequent itemset mining was first proposed by Agrawal et al. [2] for market basket analysis in the context of association rule mining. It analyzes customer buying habits by finding associations between the different items that customers place in their "shopping baskets." For instance, if customers are buying milk, how likely are they going to also buy cereal (and what kind of cereal) on the same trip to the supermarket? Such information can lead to increased sales and maximize the profit by helping retailers do selective marketing and arrange their shelf space.

Since there are usually a large number of distinct single items in a typical transaction database, and their combinations may form a very huge number of itemsets, it is challenging to develop scalable methods for mining frequent itemsets in a large transaction database. The first frequent itemset mining algorithm was Apriori, proposed by Agrawal and Srikant [3]. An interesting *downward closure* property, called Apriori property, was observed: A *k-itemset is frequent only if all of its sub-itemsets are frequent.* This implies that frequent itemsets can be mined by first scanning the database to find the frequent 1-itemsets, then using the frequent 1-itemsets to generate candidate frequent 2-itemsets, and check against the database to obtain the frequent 2-itemsets. This process iterates until no more frequent k-itemsets can be generated for some k. This is the essence of the Apriori algorithm.

Scientific Fundamentals

In many cases, the Apriori algorithm significantly reduces the size of candidate sets using the Apriori principle. However, it can suffer from two nontrivial costs: (i) generating a huge number of candidate sets, and (ii) repeatedly scanning the database and checking the candidates by pattern matching.

Han et al. [10] devised an FP-growth method that mines the complete set of frequent itemsets without candidate generation. FP-growth works in a *divide-and-conquer* way. The first scan of the database derives a list of frequent items in which items are ordered in frequency-descending order. According to the frequency-descending list, the database is compressed into a frequent-pattern tree, or *FP-tree*, which retains the itemset association information.

An example database from [10] is shown in Table 1 and the corresponding FP-tree is shown in Fig. 1. In this way, the problem of mining frequent patterns in databases is transformed to that of mining the FP-tree.

The FP-tree is mined by starting from each frequent length-1 pattern (as an initial suffix pattern), constructing its *conditional pattern base* (a "subdatabase," which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern), then constructing its conditional FP-tree, and performing mining recursively on such a tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree. Continue with the example in [10], Fig. 2 shows the recursive mining process on the conditional FP-tree of item m. It derives three frequent patterns $(am : 3)$, $(cm : 3)$ and $(fm : 3)$. Recursive calls of the FP-growth algorithm construct the conditional FP-trees of am, cm and fm (the conditional FP-tree of fm is empty in this example) respectively. The recursive mining process on these conditional FP-trees is shown in Fig. 2.

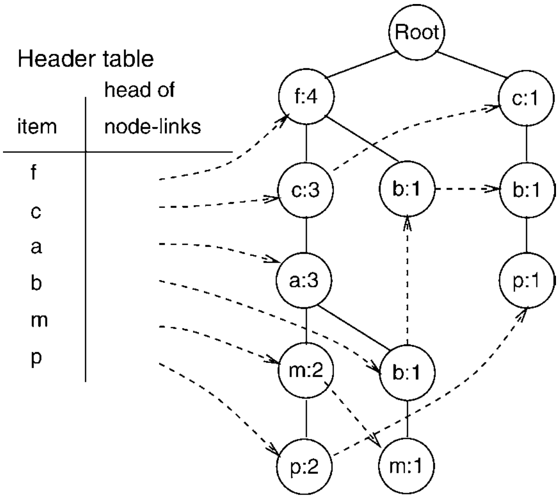
The FP-growth algorithm transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. Performance studies demonstrate that the method substantially reduces search time.

Both the Apriori and FP-growth methods mine frequent patterns from a set of transactions in *horizontal data format* (*i.e.*, {*TID: itemset*}), where TID is a transaction-id and *itemset* is the set of items bought in the transaction TID. Alternatively, mining can also be performed with data presented in *vertical data format* (i.e., {*item: TID_set*}).

Zaki [20] proposed Eclat (Equivalence CLASS Transformation) algorithm by exploring the vertical data format. The first scan of the database builds the TID_set of each single item. Starting with a single item ($k = 1$), the frequent ($k + 1$)-itemsets grown from a previous k-itemset can be generated according to the Apriori property, with a depth-first computation order similar to FP-growth [10]. The computation is done by intersection of the TID_sets of the frequent k-itemsets to compute the TID_sets of the corresponding ($k + 1$)-itemsets. This process repeats, until no frequent itemsets or no candidate itemsets can be found.

Frequent Itemsets and Association Rules. Table 1. Example transaction database *D*, $\theta = 3$

TID	Items bought	(Ordered) frequent items
100	*f,a,c,d,g,i,m,p*	*f,c,a,m,p*
200	*a,b,c,f,l,m,o*	*f,c,a,b,m*
300	*b,f,h,j,o*	*f,b*
400	*b,c,k,s,p*	*c,b,p*
500	*a,f,c,e,l,p,m,n*	*f,c,a,m,p*



Frequent Itemsets and Association Rules. Figure 1. The FP-tree for database in Table 1.

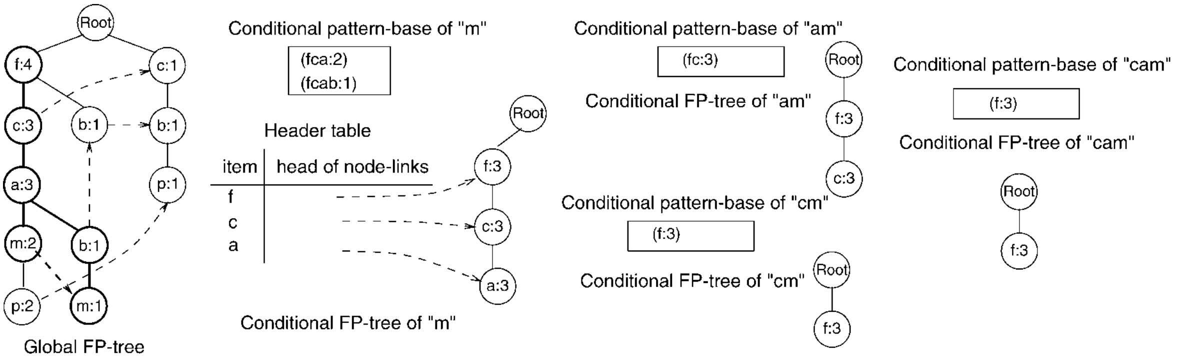
Besides taking advantage of the Apriori property in the generation of candidate ($k + 1$)-itemset from frequent k-itemsets, another merit of this method is that there is no need to scan the database to find the support of ($k + 1$)-itemsets (for $k \geq 1$). This is because the TID_set of each k-itemset carries the complete information required for counting such support.

There are many alternatives and extensions on frequent itemset mining, e.g., hashing technique [14], partitioning technique [16], sampling approach [18], dynamic itemset counting [4], and so on. A FIMI (Frequent Itemset Mining Implementation) workshop dedicated to the implementation methods of frequent itemset mining was reported by Goethals and Zaki [9].

Key Applications

Association and Correlation Analysis

Frequent itemset mining naturally leads to the discovery of associations and correlations among items in large transaction data sets. The concept of association rule was introduced together with that of frequent itemset [2]. An association rule r takes the form of $\alpha \rightarrow \beta$, where α and β are itemsets, and $\alpha \cap \beta = \phi$. support and confidence are two measures of rule


Root
f:4
c:1
c:3
b:1
b:1
a:3
p:1
m:2
b:1
p:2
m:1
Global FP-tree
Conditional pattern-base of "m"
(fca:2)
(fcab:1)
Header table
item
head of node-links
f
c
a
Root
f:3
c:3
a:3
Conditional FP-tree of "m"
Conditional pattern-base of "am"
(fc:3)
Root
f:3
c:3
Conditional FP-tree of "am"
Conditional pattern-base of "cm"
(f:3)
Conditional FP-tree of "cm"
Root
f:3
Conditional pattern-base of "cam"
(f:3)
Conditional FP-tree of "cam"
Root
f:3

Frequent Itemsets and Association Rules. Figure 2. Mining the conditional FP-tree for item *m*.

interestingness, where $support(r) = support(\alpha \cup \beta)$ and $confidence(r) = \frac{support(\alpha \cup \beta)}{support(\alpha)}$ for a rule r.

Sometimes, an association rule may not be interesting, especially when mining at a low support threshold or mining for long patterns. To mine interesting rules, a correlation measure has been used to augment the support-confidence framework of association rules. This leads to the correlation rules of the form $\alpha \Rightarrow \beta[support, confidence, correlation]$. There are various correlation measures including lift, χ^2, *cosine* and *all_confidence*.

Frequent Pattern-based Classification and Clustering

Frequent itemsets have been demonstrated to be useful for classification, where association rules are generated and analyzed for use in classification [6,11,13]. The general idea is that strong associations between frequent patterns and class labels can be discovered. Then the association rules are used for prediction. In many studies, associative classification has been found to be more accurate than some traditional classification methods, such as C4.5.

Cluster analysis in high-dimensional space is a challenging problem. Since it is easy to compute frequent patterns in subsets of high dimensions, it provides a promising direction for high-dimensional subspace clustering. Two algorithms CLIQUE [1] and ENCLUS [5] were proposed, both of which used the Apriori property to mine interesting subspaces.

Biological Data Analysis

An important experimental application of frequent itemsets is the exploration of gene expression data, where the joint discovery of both the set of conditions that significantly affect gene regulation and the set of co-regulated genes is of great interest. Wang et al. proposed pCluster [15], a pattern similarity-based clustering method for microarray data analysis, and demonstrated its effectiveness and efficiency for finding subspace clusters in high-dimensional space.

Cong et al. [7] proposed to discover top-k covering rule groups for each row of gene expression profiles. It uses a row enumeration technique and introduces several pruning strategies to make the rule mining process very efficient. A classifier is constructed from the top-k covering rule groups.

Web Mining and Software Bug Mining

Frequent itemset mining can also be applied to other application domains like Web and software debugging. Association rules discovered for pages that are often visited together can reveal user groups [8] and cluster web pages. Web access patterns via association rule mining in web logs were proposed in [15,17].

Frequent pattern mining has started playing an important role in software bug detection and analysis. PR-Miner [12] uses frequent itemset mining to extract application-specific programming rules from source code. A violation of these rules might indicate a potential software bug.

Future Directions

First, the set of frequent itemsets derived by most of the current mining methods is too huge for effective usage. The bottleneck of frequent itemset mining is not on whether users can derive the complete set of frequent patterns under certain constraints efficiently but on whether they can derive a compact but high quality set of patterns that are most useful in applications. There are proposals on reduction of such a huge set, including closed patterns, maximal patterns, approximate patterns, condensed pattern bases, representative patterns, etc. However, it is still not clear what kind of patterns will give satisfactory pattern sets in both compactness and representative quality for a particular application. Much research is still needed to substantially reduce the size of derived pattern sets and enhance the quality of retained patterns.

Second, although there are efficient methods for mining precise and complete set of frequent itemsets, approximate frequent patterns could be the best choice to handle noise or variations in many applications such as bioinformatics. How to define the approximate constraint and design efficient mining algorithms is an open question. Much research is still needed to make such mining effective and efficient.

Third, to make frequent itemset mining an essential task in data mining, much research is needed to further develop pattern-based mining methods. For example, classification is an essential task in data mining. How to construct a better classification model using frequent patterns than most other classification methods? What kind of frequent patterns are more effective and discriminative than other patterns? How to mine such patterns directly from data? These questions need to be answered before frequent patterns can play an essential role in several major data mining tasks, such as classification.

Experimental Results

In general, for every proposed method, there is an accompanying experimental evaluation in the corresponding reference. In addition, [9] in FIMI workshop provided a detailed and comprehensive experimental evaluation of many mining methods on a large set of benchmark data.

Data Sets

An IBM Quest synthetic data generator is available at http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html.

A large collection of real datasets can be found at http://fimi.cs.helsinki.fi/data/.

URL to Code

FIMI workshop website (http://fimi.cs.helsinki.fi/src/) contains the code for many different frequent itemset mining methods.

Cross-references

▶ Approximation of Frequent Itemsets
▶ Association Rule Mining on Streams
▶ Closed Itemset Mining and Non-redundant Association Rule Mining
▶ Frequent Graph Patterns
▶ Mase-pattern Mining
▶ Sequential Patterns

Recommended Reading

1. Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 94–105.
2. Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 207–216.
3. Agrawal R. and Srikant R. Fast algorithms for mining association rules. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
4. Brin S., Motwani R., Ullman J.D., and Tsur S. Dynamic itemset counting and implication rules for market basket analysis. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 255–264.
5. Cheng C.-H., Fu A.W., and Zhang Y. Entropy-based subspace clustering for mining numerical data. In Proc. 5th ACM SIGKOD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 84–93.
6. Cheng H., Yan X., Han J., and Hsu C. Discriminative frequent pattern analysis for effective classification. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 716–725.
7. Cong G., Tan K.-L., Tung A.K.H., and Xu X. Mining top-k covering rule groups for gene expression data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 670–681.
8. Eirinaki M. and Vazirgiannis M. Web mining for web personalization ACM Trans. Int. Tech., 3:1–27, 2003.
9. Goethals B. and Zaki M. An introduction to workshop on frequent itemset mining implementations. In Proc. ICDM'03 International Workshop on Frequent Itemset Mining Implementations, 2003, pp. 1–13.
10. Han J., Pei J., and Yin Y. Mining frequent patterns without candidate generation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 1–12.
11. Li W., Han J., and Pei J. CMAR: Accurate and efficient classification based on multiple class-association rules. In Proc. 2001 IEEE Int. Conf. on Data Mining, 2001, pp. 369–376.
12. Li Z. and Zhou Y. PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code. In Proc. ACM SIGSOFT Symp. on Foundations Software Eng., 2005, pp. 306–315.
13. Liu B., Hsu W., and Ma Y. Integrating classification and association rule mining. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998, pp. 80–86.
14. Park J.S., Chen M.S., and Yu P.S. An effective hash-based algorithm for mining association rules. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 175–186.
15. Pei J., Han J., Mortazavi B.-A., and Zhu H. Mining access patterns efficiently from web logs. In Advances in Knowledge Discovery and Data Mining, 4th Pacific-Asia Conf., 2000, pp. 396–407.
16. Savasere A., Omiecinski E., and Navathe S. An efficient algorithm for mining association rules in large databases. In Proc. 21st Int. Conf. on Very Large Data Bases, 1995, pp. 432–443.
17. Srivastava J., Cooley R., Deshpande M., and Tan P. Web usage mining: discovery and applications of usage patterns from web data. SIGKDD Explor., 1:12–23, 2000.
18. Toivonen H. Sampling large databases for association rules. In Proc. 22nd Int. Conf. on Very Large Data Bases, 1996, pp. 134–145.
19. Wang H., Wang W., Yang J., and Yu P.S. Clustering by pattern similarity in large data sets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 418–427.
20. Zaki M.J. Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng., 12:372–390, 2000.

Frequent Partial Orders

Antti Ukkonen
Helsinki University of Technology, Helsinki, Finland

Definition

Given a set D of n *partial orders* on S, and a threshold $\sigma \leq n$, a partial order P is a *frequent partial order* (FPO) if it is compatible with more than σ partial in D. Typically D contains total orders either on S or arbitrary subsets of S.

Historical Background

A natural extension of association rule mining is to make use of temporal information. This was first done in [1], where the authors present algorithms for mining *frequently occurring sequences* of sets of items in a database of transactions. Each of such sequences can be seen as a partial order on the complete set of items. For more recent work on the same topic please see [13,812]. The slightly different problem of mining *frequent episodes* from a sequence of events is presented in [7]. In this case an episode is a partial order over the set of all possible events. The problem differs from the one of [1] by considering a stream of events (for example notifications and alerts generated by devices in a telecommunications network) instead of a stationary set of transactions. The same problem is also discussed e.g., in [4,5].

Common for the references above is that they do not explicitly call the mined patterns partial orders. One of the first papers to do so is [6], where the authors discuss the problem of finding a number of partial orders that are a good compact description for a set of input sequences. The approach taken differs from the traditional setting of frequent pattern mining, however. A similar problem is addressed recently in [3], albeit in a more theoretical setting.

The problem of finding frequent partial orders as defined in this article was first addressed in [9,10]. The authors present an efficient algorithm for finding frequent closed partial orders (The term closed appears in the same meaning as in the context of *frequent closed itemsets*.) in a database of strings with the restriction that each symbol of the alphabet may occur only once or not at all in a string.

Foundations

The basic idea of frequent pattern mining can be formalized as follows: given a database D, a pattern class $\mathcal{P}$ and a threshold σ, find all instances of $\mathcal{P}$ that are supported by more than σ rows of D. The precise definition of a support depends on the contents of D and the pattern class $\mathcal{P}$. In case of frequent partial orders the pattern class is the set of all partial orders on some fixed set S, and the database can contain either total or partial orders on S. All orders in D that are compatible with the partial order P form the *support* of P, denoted $s(P)$. Given D and a threshold σ, the problem is to find all partial orders P so that $|s(P)| \geq \sigma$.

A pattern is *closed* if it can not be augmented without decreasing the size of its support. According to this definition the partial order P is closed in the database D if $|s(P \cup (u, v))| < |s(P)|$ for all $u, v \in S$. It can be argued that finding only the frequent closed patterns is of interest, as the non-closed ones contain less information but can be considered only equally "reliable," as they have the same support as a closed pattern. In the remainder of this section two methods for finding *frequent closed partial orders* are discussed. The first one is based on using existing algorithms for mining *frequent closed itemsets*, while the second one is a dedicated method for finding frequent partial orders.

Finding Frequent Closed Partial Orders Using Frequent Itemset Mining Algorithms

Since its original development in the early 1990s, association rule mining and especially the discovery of frequent itemsets has been a widely studied topic. As a result, there currently exist a myriad of efficient algorithms for mining frequent closed itemsets. Turns out that any of these can be used to find frequent closed partial orders if the input is in a suitable format.

Let D contain total orders on S. Usually each total order $T \in D$ is given as a list of symbols. For example, let $T = \langle a, b, c, d, e \rangle$, meaning that a comes first in T, b comes second, and so on. This is the *list representation* of T, which is a compact and intuitive way of representing total orders, but can not as such be used with frequent itemset mining algorithms, because they only consider the occurrence of a symbol and not their position relative to the other symbols in the list.

Any order relation can also be expressed as a set of ordered pairs (u, v). The pair (u, v) belongs to T when u appears before v in the list representation of T. Returning to the example, the *set representation* of T is

$$T = \{(a,b),(a,c),(a,d),(a,e),(b,c),(b,d),(b,e),\\ (c,d),(c,e),(d,e)\}.$$

This differs from the list representation by having pairs of symbols instead of single symbols as "items." Thus, if two total orders have the same pair (u, v), they must agree on the order between u and v.

Given a database D of total orders in the list representation, each member of D is converted to the set representation. The resulting database is denoted $\hat{D}$. Each row of $\hat{D}$ is a set of (u, v) pairs that form a total order $T \in D$. This representation of the input has the consequence that every closed itemset (set of (u, v) pairs) that is frequent in $\hat{D}$ given the threshold σ can in fact be interpreted as a frequent closed partial order. An itemset I is closed if no more items can be added to I without decreasing the size of its support.

Note that it is important to find the frequent closed itemsets, as a frequent itemset might not correspond to a partial order. The closedness guarantees that the resulting sets of (u, v) pairs form a transitive relation, which is required of a partial order.

Using the set representation for finding frequent partial orders works, but it has some drawbacks. First of all, the storage requirements for $\hat{D}$ are much larger than for D, as each total order with a list representation of l symbols must be replaced with a set containing $\frac{1}{2}l(l-1)$ elements. In addition, $\hat{D}$ might not be sparse, meaning the number of items in each row (total order) is not necessarily small when compared to the total number of items. This in turn may lead to poor performance of the frequent itemset mining algorithms.

The most important problem with the above approach is, however, that in general a complete partial order P is not required. Very often it is sufficient to find its *transitive reduction* $tr(p)$. All of the ordering information contained in P is retained in $tr(p)$, which is better suited for analysis purposes. For example, when visualizing partial orders as directed acyclic graphs, it is in general better to use $tr(p)$. Obviously the transitive reduction can be computed afterwards given the frequent itemsets, but this is another computationally intensive step. Also, frequent partial orders can be found more efficiently when the transitive reduction is mined directly.

Finding Transitive Reductions of Frequent Closed Partial Orders Directly

Details of the approach are given in [10], this is only a summarization of the key ideas. This algorithm (called Frecpo in [10]) operates directly on the list representations in D and returns the set of the transitive reductions of all closed partial orders that are frequent in D given the threshold σ.

The basic framework is based on enumerating all representations of transitive reductions of partial orders in a depth-first fashion using a recursive algorithm. All closed partial orders that can not be frequent are pruned as soon as possible. The pruning is based on the observation that the frequency of the partial order $P \cup (u, v)$ is upper bounded by the frequency of P. In general this is called *antimonotonicity* of a pattern.

When entering a recursive call the algorithm has already constructed a frequent partial order P in the previous steps. This is called the *current pattern*, which is initially Ø. The algorithm first computes a list L of pairs that are added to P one after the other to create a new frequent partial order. For a pair (u, v) to be included in this list it must (i) belong to a transitive reduction and (ii) the frequency of the pattern $P \cup (u, v)$ must be above the threshold σ. When computing this list the algorithm only considers total orders that belong to the support of the current pattern P, denote this with D_P. Initially $D_P = D$.

To construct the list L, the algorithm computes a table called the *detection matrix*. In this matrix for each pair (u, v) is stored the number of times u precedes v in D_P, and the set of items that appear between u and v in every total order where u precedes v. These items are called *anchors*. The frequency information can be used to prune infrequent pairs. If the frequency of the pair (u, v) is below σ in D_P, the partial order $P \cup (u, v)$ can not be frequent either. However, if the frequency of (u, v) is above σ in D_P, then $P \cup (u, v)$ must be a frequent partial order as well.

The set of anchors is used to identify forbidden pairs. A pair is forbidden if it's set of anchors is not empty. This is because a pair (u, v) can not belong to the transitive reduction if an item (the anchor) occurs between u and v in every total order where u and v occur. All pairs that are not infrequent or forbidden are added to the list L.

For details of the algorithm the reader is referred to [10], where it is also shown experimentally that Frecpo outperforms the algorithm based on frequent itemset mining by a considerable margin.

Key Applications

Frequent partial orders can be of interest in any application where the data can be viewed as a set of orders (or rankings) of some finite set.

For example, in certain voting systems the voters do not only place a single vote on one candidate, but are expected to rank the alternatives (or a subset thereof) according to their preferences. This voting mechanism is employed for instance in the general election of Ireland. Finding FPOs of the candidates from this data can give more insight to the behavior of voters than traditional opinion polls. Preference data in general is a natural application for FPOs. Such data can be based on questionnaires, but also other sources.

A related application is clickstream analysis. Based on server log files it is possible to reconstruct the sequence in which a user visited different pages of a website. Finding FPOs from these sequences can give information about user preferences (if the pages correspond to different products, for example) or potential usability problems associated with navigation on the website in question.

Another promising application is in bioinformatics in the context of gene expression analysis. A gene expression data set usually contains expression levels of several genes in a number of conditions or tissues. Instead of looking at the actual expression value, which can be very noisy, the conditions or tissues can be ranked in decreasing order of expression and use the resulting rankings for further analysis. In this case the FPOs can be seen as a generalization of the order preserving submatrices, originally proposed in [2].

Cross-references

▶ Frequent Itemsets and Association Rules
▶ Sequential Patterns

Recommended Reading

1. Agrawal R. and Srikant R. Mining sequential patterns. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 3–14.
2. Ben-Dor A., Chor B., Karp R., and Yakhini Z. Discovering Local Structure in Gene-Expression Data: The Order Preserving Submatrix Problem. In Proc. 6th Annual Int. Conf. on Computational Biology, 2002, pp. 49–57.
3. Fernandez P.L., Heath L.S., Ramakrishnan N., and Vergara J.P. Reconstructing Partial Orders from Linear Extensions. In Proc. 4th SIGKDD Workshop on Temporal Data Mining: Network Reconstruction from Dynamic Data, 2006.
4. Gwadera R., Atallah M.J., and Szpankowski W. Reliable Detection of Episodes in Event Sequences. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 67–74.
5. Laxman S., Sastry P.S., and Unnikrishnan K.P. A fast algorithm for finding frequent episodes in event streams. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007, pp. 410–419.
6. Mannila H. and Meek C. Global Partial Orders from Sequential Data. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 161–168.
7. Mannila H., Toivonen H., and Verkamo I. Discovering frequent episodes in sequences. In Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining, 1995, pp. 210–215.
8. Pei J., Han J., Mortazavi-Asl B., Pinto H., Chen Q., Dayal U., and Hsu M.-C. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 215–224.
9. Pei J., Liu J., Wang H., Wang K., Yu P.S., and Wang J. Efficiently Mining Frequent Closed Partial Orders. In Proc. 2005 IEEE Int. Conf. on Data Mining, 2005, pp. 753–756.
10. Pei J., Wang H., Liu J., Wang K., Wang J., and Yu P.S. Discovering frequent closed partial orders from strings. IEEE Trans. Knowl. Data Eng., 18(11):1467–1481, 2006.
11. Wang J. and Han J. BIDE: Efficient Mining of Frequent Closed Sequences. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 79–90.
12. Yan X., Han J., and Afshar R. CloSpan: Mining Closed Sequential Patterns in Large Datasets. In Proc. SIAM International Conference on Data Mining, 2003, pp. 166–177.
13. Zaki M. SPADE: an efficient algorithm for mining frequent sequences. Mach. Learn. J., 42(1/2):31–60, 2000.

Frequent Pattern Mining with Constraints

▶ Frequent Itemset Mining with Constraints

Frequent Patterns

▶ Frequent Itemsets and Association Rules

Frequent Set Mining with Constraints

▶ Frequent Itemset Mining with Constraints

Frequent Subsequences

▶ Sequential Patterns

Freshness Control

▶ Replica Freshness

Full Text Inverted Index

▶ Inverted Files

Fully-Automatic Web Data Extraction

Cai-Nicolas Ziegler
Siemens AG, Munich, Germany

Synonyms

Web content extraction; Automatic wrapper induction; Web information extraction

Definition

Web documents contain abundant hypertext markup information, both for indicating structure as well as for giving page rendering hints, next to informative textual content. Fully-automatic Web data extraction is geared towards extracting all relevant textual information from HTML documents, without requiring human intervention throughout the process. Commonly, two types of automatic Web extraction paradigms are distinguished in this vein. First, the extraction of one single block of informative content, e.g., in case of news pages, which is also referred to as page cleaning [4]. Second, the extraction of recurring patterns across multiple blocks, typically the case for the extraction of search engine results. In the latter case, the extraction system will commonly also assign *labels* to the single atoms of each identified recurring block, such as the search result record's title, snippet, and URL.

Historical Background

Systems for extracting information from Web pages date back to the late mid-nineties. First approaches, coined wrapper induction systems [6], extracted structured information from HTML documents in a semi-automated fashion, see, e.g., [1] and [9]. A wrapper hereby refers to a set of learned rules that extract structured information records from HTML pages of like style, i.e., dynamically generated pages for which the same template has been used. One may think of two product pages from Amazon.com as an example. The rules have been learned inductively, by means of humans labeling a sufficient number of pages.

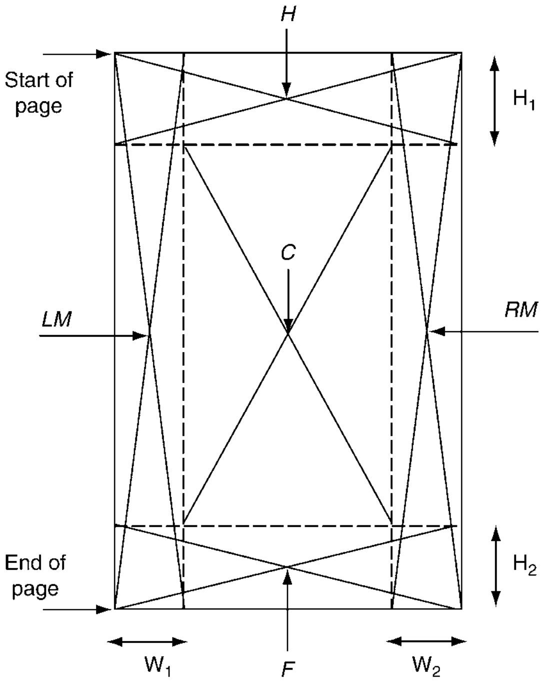
While these wrapper induction systems have been continuously improved to require less human intervention and less time for designing wrappers, they still do not scale to efforts that require the extraction of massive amounts of information from diverse types of Web document sources: Even with an effort of only five minutes per site template, coping with several thousands of those requires too many human resources for most project budgets.

Two areas have been identified where extraction works in a fully-automated fashion, the first referring to the extraction of results from any type of search engine, such as Web search engines like Google and Yahoo, product finders as they are found on Amazon and friends, and so forth. These systems are able to automatically generate wrappers as they exploit the fact that search result pages commonly feature large numbers of recurring patterns. Moreover, these automated search engine result wrappers also assign labels to the components of each entry found, such as the title of the search result snippet, the URL, and body text.

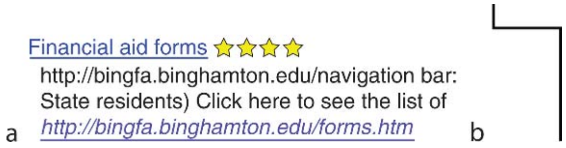
A second breed of automated page wrappers has evolved at the end of the nineties. Theses wrappers aim at extracting the purely informative content from Web pages, particularly *news* Web pages. As opposed to the automated search engine result wrappers, no labels are assigned to the different blocks of information. Hence, the generated output is commonly one single plain text document which contains the informative textual content only. Extractors of informative textual content have also been referred to as "page cleaners" [1].

Foundations

For the extraction of informative content from Web pages, also referred to as Web page cleaning [1], a broad range of diverse techniques has been investigated. Some systems use simple heuristics (see, e.g., [1] and [5]) and obtain accuracy scores that are reasonably good. For example, such heuristics comprise of rules that take into account the number of characters and text tokens which occur in one cohesive textual block [1]. Other approaches compute a set of features for each text block and implement a decision function which tells whether to discard or keep the block at hand. Such approaches are depicted in [1,6,7]. The respective decision functions are learned by means of non-linear optimization methods or common classifier training [7].



Fully-Automatic Web Data Extraction. Figure 1. The approach of Kovacevic et al. [5] segments an HTML page's visual representation into five regions.


Financial aid forms ☆☆☆☆
http://bingfa.binghamton.edu/navigation bar:
State residents) Click here to see the list of
http://bingfa.binghamton.edu/forms.htm

Fully-Automatic Web Data Extraction. Figure 2. Sample search result entry (a) and its abstract shape (b), as used in ViNTS [13].

Next to approaches that operate either on the text level or logical structure of a Web page, e.g., its DOM tree, the exploitation of visual cues has been likewise researched: The location-based segmentation approach described in [8] constructs an M-tree to represent the HTML document's physical representation as seen in the browser window. To this end, the browsers screen coordinates of block segments are taken into account. The screen is divided into five areas: header, left and right menu, footer, and the center of the page (see Fig. 1). Heuristics are used to assign HTML blocks to the defined areas, the informative content is assumed to reside in the center of the page.

For extracting *recurring* information blocks from Web pages, in particular Web search engine results, state-of-the-art wrapper generators also exploit visual cues by analyzing an HTML page's graphical representation as rendered by a Web browser engine [9,10]. ViNTs [10] automatically generates search result record extraction rules using visual context features and tag structure information. To this end, ViNTS first analyzes the graphical representation without considering the tag structure to identify content regularities by means of so-called content lines (see Fig. 2 for a content line typical of search results on Amazon). Next, structural regularities among HTML blocks are combined with these visual features to generate wrappers. To weight the relevance of different extraction rules, visual and non-visual features are considered.

The ViPER system [9] builds on similar techniques as ViNTS, but extends its capabilities by not only allowing to identify recurring blocks that are aligned vertically, but also those aligned in a horizontal fashion. Horizontally aligned search results are becoming increasingly popular among online retailers, such as Overstock (http://www.overstock.com). Next to the exploitation of visual cues, ViPER used multiple sequence alignment techniques, known from bio-informatics, to identify structure and patterns in HTML tag sequences.

Key Applications

The extraction and automatic labeling of search results sees its application in various domains, primarily product-related data integration. For instance, price robots access large numbers of online retail shops and need to interpret search results for product searches so as to digest and incorporate the found information into their own databases. Moreover, meta search engines also have an increasing demand for the automated extraction of search results, as their operation is based on the merging of search results from several hundreds of search engines, for which the manual design of extraction wrappers is enormously time-consuming. Moreover, manually crafted wrappers may break easily, when the structural template of presented results changes. Automatically generated search engine wrappers are less prone to these deficiencies.

The extraction of informative content from arbitrary HTML documents serves many purposes. Clearly, when used as processing step for Web search engines, it can help to improve the precision of search results dramatically, as only an HTML document's informative content is considered for inclusion in the search index. Pages where the search terms only occur in non-informative content, such as an advertisement or as part of a link list, are not considered as hits anymore.

Next to general-purpose search engines, the extraction of informative page content is likewise essential for special-purpose applications based on retrieval, such as reputation monitoring platforms [3,11]: These systems record the number of mentions of monitored keywords, such as company, brand, or product names, in order to allow for timeline-based trend analysis. Citation count numbers hence become more reliable, for the same reasons as those stated for Web search engines.

Automated Web content extraction for page cleaning is also at the heart of the CLEANEVAL competition (See *http://cleaneval.sigwac.org.uk/* for details.), which has become part of the "Web as Corpus" initiative (WAC) as of May 2007. The objective of WAC is to collect massive textual information from the Web in order to use it for natural language processing (NLP) and linguistic research, forming representative background corpora and language models.

Data Sets

For the CLEANEVAL competition, a dataset containing both unlabeled documents (for testing purposes) and labeled documents (for classifier training) can be downloaded from the indicated URL. The relatively new dataset is expected to serve as publicly accepted benchmark in the future.

For automatic extraction of search engine results, several smaller datasets are available, among those the Omini dataset (Available from Sourceforge via *http://sourceforge.net/projects/omini/*.) and MDR collection [8]. None of them may count as standard dataset, though.

Cross-references

- ▶ GUIs for Web Data Extraction
- ▶ Information Extraction
- ▶ Information Filtering
- ▶ Wrapper Induction

Recommended Reading

1. Crescenzi V., Mecca G., and Merialdo P. RoadRunner: towards automatic data extraction from large web sites. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 109–118.
2. Debnath S., Mitra P., and Giles C.L. Automatic extraction of informative blocks from webpages. In Proc. ACM Symp. on Applied Computing, 2005, pp. 1722–1726.
3. Glance N., Hurst M., Nigam K., Siegler M., Stockton R., and Tomokiyo T. Deriving marketing intelligence from online discussion. In Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2005, pp. 419–428.
4. Hofmann K. and Weerkamp W. Web corpus cleaning using content and structure. In Building and Exploring Web Corpora, C. Fairon, H. Naerts, A. Kilgarrif, and G. de Schryver (eds.). vol. 4, UCL, 2007, pp. 145–154.
5. Kovacevic M., Dilligenti M., Gori M., and Milutinovic V. Recognition of common areas in a web page using a visualization approach. In Proc. 10th Int. Conf. on Artificial Intelligence: Methodology, Systems, and Applications, 2002, pp. 203–212.
6. Kushmerick N., Weld D., and Doorenbos R. Wrapper induction for information extraction. In Proc. 15th Int. Joint Conf. on AI, 1997, pp. 119–128.
7. Lin S.H. and Ho J.M. Discovering informative content blocks from web documents. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 588–593.
8. Liu B., Grossman R., and Zhai Y. Mining data records in web pages. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 601–606.
9. Muslea I., Minton S., and Knoblock C. Hierarchical wrapper induction for semistructured information sources. Auton. Agent. Multi Agent Syst., 4(1–2):93–114, 2001.
10. Simon K. and Lausen G. ViPER: augmenting automatic information extraction with visual perceptions. In Proc. Int. Conf. on Information and Knowledge Management, 2005, pp. 381–388.
11. Ziegler C.N. and Skubacz M. Towards automated reputation and brand monitoring on the web. In Proc. IEEE/WIC/ACM Int. Conf. on Web Intelligence, 2006, pp. 1066–1070.
12. Ziegler C.N. and Skubacz M. Content extraction from news pages using particle swarm optimization on an linguistic and structural features. In Proc. IEEE/WIC/ACM Int. Conf. on Web Intelligence, 2007, pp. 242–249.
13. Zhao H., Meng W., Wu Z., Raghavan V., and Yu C. Fully automatic wrapper generation for search engines. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 66–75.

Fully Temporal Relation

▶ Bitemporal Relation

Functional Data Model

PETER M. D. GRAY
University of Aberdeen, Aberdeen, UK

Definition

Functional Data Models are a form of *Semantic Data Model* which appeared early in database history. They use the mathematical formalism of *function application* to represent and follow associations between data items. Functions are usually applied to variables whose values may be object identifiers or record instances. Thus if P represents an entity instance of type Person, then *forename(P)* would return a string (e.g., "Peter"). (Note however that different systems may write this in a LISP style as (forename P) or in JAVA style as P. forename.) The function *town(P)* could be used to represent an *association* by returning the identifier for Peter's home town. This allows *function composition* so that *name(town(P)) = P.town.name = "Aberdeen".*

Using functions gives several obvious advantages. Firstly the syntax is universally understood, and provides a means of expressing schemas, queries and constraints independently of any supplier-dependent language. This is very handy for integrating data held in heterogeneous databases – one of the earliest applications [11]. Secondly, functional expressions follow the rule of *Referential transparency* – expressions of equal value can be substituted for variables without altering the sense or value of the expression. This avoids the problems of side-effects in nested *procedure calls* found in programming languages. In consequence, optimizing functional expressions is much easier, as is parallel evaluation. In particular, the whole mathematical theory of *Comprehensions* worked out for functional programming can be applied to *Functional Query Languages.*

Historical Background

Functions provided an underlying formalism for data models from as early as Abrial's *access functions* for representing binary relationships between entities [1] and Florentin's *property functions* for representing the attributes of entities [4].

Kerschberg and Pacheco's *Functional Model of Data* [8] integrated these two uses for functions, modeling the universe of discourse by means of *entity sets* and *total functions.* Entity sets are represented by labeled nodes and functions by labeled arcs in a directed graph.

The main motivation for the Functional Model of Data was ease of conceptual modeling, and [8] showed how functional data models can be automatically transformed into relational and CODASYL data models for implementation within a DBMS.

The Functional Model of Data was further developed by Sibley and Kerschberg in [14]. In this development, the universe of discourse is again represented by labeled nodes and arcs. The arcs are again total functions, but the nodes are either entity sets or *value sets* (as opposed to a single set of character strings, C). A major motivation for this work was to derive a general, unifying conceptual data model which could then be specialized into either relational or CODASYL data models.

IS-EMPLOYEE	HAS-SALARY		
emp-id	name	emp-id	salary
Paul	"Paul"	Paul	19,000
Mary	"Mary"	Mary	22,000
John	"John"	John	19,000
...	...	...	...

At about the same time, Hammer and McLeod proposed the Semantic Data Model [7], as a higher-level model better suited to conceptual modelling of an application domain. A key innovation of the Semantic Data Model was the recognition of the importance of entity attributes based on *derived information.* Shipman used functions to implement a Semantic Data Model representing both data and derived information in the very influential DAPLEX language [13]. It was used to integrate a heterogeneous database network MULTIBASE [11].

Following this came fore-runners of *Object-Oriented Databases.* The GENESIS query language [2] supported a functional data model and was intended as a front end for DBMSs supporting either a relational or nested relational data model. The PROBE database system [3] supported the representation and manipulation of arbitrarily complex objects by means of a functional data model.

Foundations

The earliest functional data models were based on binary relational data, where every relational table had only two columns. The name of the table, considered as a verb relating the two columns, became the function name. One column was chosen to be the argument and the other to be the result. Where possible, this is done to make the function single-valued, as usual.

Consider two such tables representing facts such as

```
Paul is-employee ``Paul''
Mary is-employee ``Mary''
```

```
...
Paul has-salary 19000
Mary has-salary 22000
```

where each row refers to an individual employee and the right-hand column entries for that employee give their name and current salary respectively. Note that Paul and Mary are surrogates for object-identifiers, in order to make the example more readable.

A functional view could specify this information as the functions

```
is-employee :: emp-id -> name
has-salary :: emp-id -> salary
```

Where the relationship is one-to-many, as for example with

```
Paul has-child Jane
Paul has-child Sue
Paul has-child Norman
Mary has-child James
```

Then the function must return a *set of* emp-id. Thus *children*(*Paul*) = {*Jane*,*Sue*,*Norman*} and *children*(*Mary*) = {*James*}. Some values will be empty sets e.g., *children*(*John*) ={}. Historically, only garbage-collected languages (such as LISP and FP and Prolog) could deal with such variable-length lists, and they did not suit analysts used to the conventions of fixed-length tables in Codd's normal form! It was only gradually that people realized that it was perfectly straightforward to hold data in relational tables and to use the Functional Data Model as a high-level *View* on the same data. With the coming of Java, one did not even have to use a special purpose list processing language in order to handle or print such values. Likewise, although some implementations stored facts in a specially designed triple store [9], it was not essential to do so. Interestingly, such storage techniques are becoming favored again for storing RDF triples of web data (see below).

Note that the functional approach is specifying information in smaller units, that is with a finer semantic granularity than with a single n-ary table. Thus further information can readily be added, such as the function:

```
line-manager :: emp-id -> emp-id
```

A powerful argument for the functional approach, in addition to its greater flexibility, is that it facilitates the incremental development of systems where the schema evolves as more data is collected. Note also the convenience with which one can ask who is the line manager of a particular employee's line manager, by contrast with the SQL approach:

```
line-manager (line-manager(Paul))
```

instead of

```
  SELECT L2.line-manager FROM line-manager
L1 L2
  WHERE L1.emp-id = Paul AND L2.emp-id = L1.
line-manager
```

In the DAPLEX language, new abstract entities are created by means of the directive A NEW and values of base functions are assigned by means of the directive LET. For example, to create a new Student of name "Fred Jones" who attends the Biology and Biochemistry courses:

```
FOR A NEW Student
  BEGIN
    LET Name(Student) = "Fred Jones"
LET Attends(Student) =
      {THE CourseOfName("Biology"),
THE CourseOfName("Biochemistry")}
END
```

Note here the use of CourseOf Name(Biology) instead of just "Biology" as in a relational database. This is because variables in DAPLEX can denote actual *object identifiers* (as in Java, for example) whereas the analogous values in relational database columns will be *foreign key* values in order to identify the object. This is a big difference from the relational model and is common to most functional data models.

Semantic Web Vision and RDFS

The *semantic web* vision is to enable rich machine processing of web information sources. RDF stands for Resource Description FrameWork Model, which was first accepted by W3C as a Data Model described in *RDF Schema* in February 1999. (http://www.w3.org/TR/REC-rdf-syntax) A data interchange format is defined using XML syntax with tags starting < *rdf* : to encode *subject–predicate–object triples*. The *predicate* is just a function name, for example has-salary.

Thus, RDF is not unlike the Entity-Relational data model in its use of Entity identifiers as *subject*, and Property or Relationship names as *predicate* in RDF triples. However, it also includes features of object data models in its use of object identifiers and subclasses. This makes it very similar to the Functional Data Model.

The comparison below is illuminating and shows how schemas can be mapped between the two models.

Mapping a Functional Model to RDFS

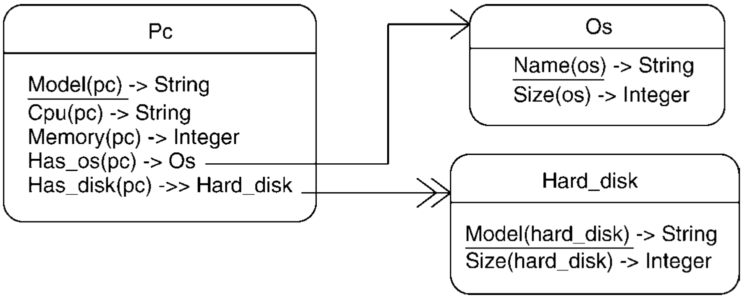
The RDFS Data Model abstracts over relational storage, flat files and object-oriented storage, following the principle of data independence. Thus, it shares with the Functional Data Model the advantage of not tieing one to any particular storage system. This is a great advantage to the programmer. The mapping to a particular knowledge source or data source can then take place separately through a wrapper. This makes it very much easier to integrate data from different sources, as is often required over the Web.

Consider the following example [5], where the functional data model schema of Fig. 1 is used to describe the pc and os classes and their relationship has_os in an application domain where components are put together to configure a workable PC.

The Functional Data Model is, of course, an extended ER model and it can be automatically mapped into an RDFS specification. A mapping program reads metadata from the database and *generates* the corresponding RDFS, as in Fig. 2, making this knowledge web-accessible. Related work by Risch [12] also shows how RDFS resources can be integrated and accessed by a functional query language. The basic rules used when mapping the schema declarations to RDFS are as follows:

- A class *c* defined as an entity (declared as *c* ->> entity) maps to an RDF resource of type rdfs: Class (where rdfs is the namespace prefix for the RDFS descriptions).
- A class *c* declared to be a subtype of another class *s* (declared as *c* ->>*s*) maps to an RDF resource of type rdfs:Class, with an rdfs:subClassOf property the value of which is the class named *s*.
- A function *f* declared on entities of class *c*, with result type *r* (declared as *f*(*c*) ->*r*) maps to an RDF resource of type rdf:Property with an rdfs: domain of *c* and an rdfs:range of *r*.

Mapping a functional schema into RDFS has the advantage of making the domain model available to



Functional Data Model. Figure 1. This functional schema shows three entity classes. The single arrow means that each pc may have only one os installed. A double arrow means that a pc can have multiple hard-disk.

```
<rdfs : Class rdf : ID= "pc">
    <rdfs : subClassOf rdf : resource = "#Resource"/>
</rdfs : Class>

<rdfs : Class rdf : ID = "os">
    <rdfs : subClassOf rdf : resource = "#Resource"/>
</rdfs : Class>

<rdf : Description rdf : ID = "has_os">
    <rdf : type rdf : Property/>
    <rdfs : domain rdf : resource = "#pc"/>
    <rdfs : range rdf : resource = "#os"/>
</rdf : Description>
```

Functional Data Model. Figure 2. RDFS (RDF Schema) representation of schema of Fig. 1.

RDFS-ready software. Some semantic information is lost, because the cardinality of each attribute is not expressed in RDFS. Also information on the *key* of each entity class is omitted. However, this information could be represented by an extra metadata class declared in RDFS.

Constraints

Integrity Constraints are a very important part of data modelling. Sometimes this is just considered as part of type checking or range checks on values, but in a *Semantic Data Model* one can express constraints that represent semantics applying to data in a particular domain, that may not be obvious from a casual inspection of tables, or even from applying Data Mining techniques. For example, in DAPLEX one can enforce that all Students attend at least four courses:

```
   DEFINE   CONSTRAINT   Number_of_courses
(Student) =>
   COUNT(Attends(Student)) > 3
```

Comprehensions in *P/FDM* are also used to describe the semantics of *Integrity Constraints*, representing invariants that must be held true under updates. For this purpose the nested loop syntax used in queries is adapted, similarly to constraints in EFDM [10]:

```
   constrain each t in seniortutor
   each s in advisees(t) to have grade(s) > 60;
```

This constrains each person in the class `seniortutor` to have only advisees with grades over 60. Both universal(each) and existential(some) quantifiers are allowed, in any combination. Such integrity constraints are particularly important in Functional Data Models, since they provide a way to *extend the data model* with rich semantics defined with the full power of a Comprehension (like range-restricted FOL). By contrast, SQL and relational languages are often restricted to just providing range checks on data items.

Updating Functional Data

In DAPLEX, the value of a multi-valued function for a particular argument can be modified by using the built-in operators INCLUDE and EXCLUDE. For example, to modify one of the courses attended by Fred Jones from BioChemistry to Physiology:

```
   FOR  THE  Student  SUCH  THAT  Name
   (Student) = "Fred Jones"
     BEGIN
       EXCLUDE  Attends(Student)  =  THE
CourseOfName("BioChemistry")
       INCLUDE  Attends(Student)  =  THE
CourseOfName("Physiology")
     END
```

It is immediately clear that the *updating* of functions defining a functional database contradicts the basic assumption of referential transparency. In a functional database, the essence of a function is conveyed by its name and its type, with which is associated its real-world semantics, and not by its current mapping, which will change as a result of database updates. The function `has-salary` specifying a person's salary as used earlier is a case in point.

The partial loss of referential transparency due to updates does not, however, alter the other considerable advantages of functional programs: that they represent a high-level, non-procedural but executable specification of what is required, and enable programs to be created in a top-down fashion with local detail encapsulated within the specification of the functions to which these details relate. Moreover a functional program that does not involve updates but accesses a database in read-only mode will be referentially transparent. Even with updates there is what might be termed local referential transparency between such updates; thus some advantages can nonetheless be gained [15].

Key Applications

The integration of data held in heterogeneous databases continues to be a very challenging problem. At one time people thought that all data could be forced into relational storage through the dominance of SQL. However, the emergence of semi-structured data, much of it held on the Web, has changed all that.

Future Directions

Functional data models have the right abstractions for dealing with heterogeneous data, but they need to be packaged for more convenient use with Web data. Interestingly, functional languages such as Python and Jython have a secure following among scientific users and others wanting to advance beyond Visual Basic. Once again, if they could be combined with a functional data modelling package that provides a view over data stored in different kinds of databases then the idea could very likely catch on.

URL to Code

http://www.csd.abdn.ac.uk/~pgray/FDMdownload.html
http://user.it.uuse/~udbl/amos/download.html http://www.dcs.bbk.ac.uk/~ap/pfl/html

Cross-references

- ► Comprehensions (IN Functional Query Languages)
- ► Functional Query Languages
- ► Introductory article of [6], from where much of this is taken

Recommended Reading

1. Abrial J.R. Data Semantics. In Data Base Management. Klimbie. J.W. and K.L. Koffeman (eds.). North Holland, 1974.
2. Batory D.S., Leung T.Y., and Wise T.E. Implementation concepts for an extensible data model and data language. ACM Trans. Database Syst., 13(3):231–262, 1988.
3. Dayal U. et al. Simplifying complex objects: the PROBE approach to modelling and querying them. In Proc. Workshop on the Theory and Applications of Nested Relations and Complex Objects, 1987, pp. 17–37.
4. Florentin J.J. Consistency auditing of databases. Computer J., 17 (1):52–28, 1974.
5. Gray P.M.D., Embury S.M., Hui K.Y., and Kemp G.J.L. The evolving role of constraints in the functional data model. J. Intell. Inf. Syst., 12:113–137, 1999.
6. Gray P.M.D., Kerschberg L., King P.J.H., and Poulovassilis A., (eds.). The Functional Approach to Data Management. Springer, Berlin Heidelberg New York, 2004.
7. Hammer M.M. and McLeod D.J. The Semantic Data Model: a modelling mechanism for database applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1978, pp. 26–35.
8. Kerschberg L. and Pacheco J.E.S. A Functional Data Base Model. Technical Report 2/76, Departmento de Informatica, Pontificia Universidade de Sao Vincente, Rio de Janeiro, 1976.
9. King P.J.H., Derakhshan M., Poulovassilis A., and Small C. TriStarp – an investigation into the Implementation and Exploitation of Binary Relational Storage Structures. In Proc. British National Conf. on Databases, 1990, pp. 64–84.
10. Kulkarni K.G. and Atkinson M.P. EFDM: Extended Functional Data Model. Computer J., 29(1):38–46, 1986.
11. Landers T. and Rosenberg R.L. An overview of Multibase. Distributed Databases, H.-J. Schneider (ed.). North Holland, 1982, pp. 153–184.
12. Risch T. Functional queries to wrapped educational semantic Web meta-data. In The Functional Approach to Data Management, Chap. 19. P.M.D. Gray, L. Kerschberg, P.J.H. King, and A. Poulovassilis, (eds.). Springer, 2004.
13. Shipman D.W. The functional data model and the data language DAPLEX. ACM Trans. Database Syst., 6(1):140–173, 1981.
14. Sibley E.H. and Kerschberg L. Data architecture and data model considerations. In Proc. AFIPS National Computer Con., 1977, pp. 85–96.
15. Sutton D.R. and Small C. Extending functional database languages to update completeness. In Proc. British National Conf. on Databases, 1995, pp. 47–63.

Functional Dependencies for Semi-Structured Data

Gillian Dobbie[1], Tok Wang Ling[2]
[1]University of Auckland, New Zealand
[2]National University of Singapore, Sigapore, Singapore

Synonyms

Path functional dependencies; Extended functional dependencies

Definition

Functional dependencies are used in relational database design to show that the value of a set of attributes depends on the value of another set of attributes. Theory has been developed to manipulate a set of functional dependencies to describe equivalences of sets of functional dependencies. Semi-structured data differs from relational data in two important ways: semi-structured data is hierarchical and the structure of the data is less consistent. Traditional functional dependencies do not capture these differences so new functional dependencies with associated theory has been defined for semi-structured data.

Key Points

Functional dependencies for semi-structured data have been defined in the three recommended readings. While the syntax of functional dependencies defined over semi-structured data varies, the semantics is similar. In this article the syntax of Arenas and Libkin [1] is used but the notation differs a little since there is no distinction between attributes and subelements. There is a notion of an XML tree and a path in an XML tree. Consider the XML document in Fig. 1a, and the XML tree for that document in Fig. 1b. The internal nodes of an XML tree are labelled with tag-name:node-ID, e.g., the node label *student:v5* means this node represents an element *student* with node ID *v5*.

A path through this XML tree is something like *department.course.code*. A tree tuple is a subtree containing at most one occurence of each path. Note that there are two occurences of paths such as *department.course.student* in Fig. 1a, so there are two tree tuples, one including the paths for the student with student number *123456* and the other including the paths for the student with student number *234567*. The tree tuples will be called t_1 and t_2 respectively. Below it is shown how the tree tuples t_1 and t_2 assign values to the paths in the XML tree in Fig. 1b:

The symbol, *S*, is a reserved symbol, and for a path *p.S*, if $t(p)=v$ then $t(p.S)$ is the value of the element or attribute at *v*. A functional dependency is an expression of the form $X \rightarrow Y$ where both *X* and *Y* are sets of paths. An XML tree satisfies a functional dependency $X \rightarrow Y$ if for any two tree tuples t_1 and t_2 in the tree $t_1(X)=t_2(X)$ then $t_1(Y)=t_2(Y)$. Note that $t_1(X)=t_2(X)$ means that $t_1(p)=t_2(p)$ for all $p \in X$. Using functional dependencies, key constraints, dependencies among attributes, and dependencies among object classes can be expressed.

The key constraint, that no two course elements will have the same code, is expressed as:

department.course.code.S → *department.course*

If there were another course in the document shown in Fig. 1a with code *"CS101"*, then this functional dependency would no longer hold.

A functional dependency is used to show that there is only one record of a student taking a course. The functional dependency represents that two student subelements of the same course cannot have the same student number. This constraint is represented as:

{*department.course,department.course.student.stuNo.S*} → *department.course.student*

In contrast, the following functional dependency expresses that no two student elements will have the same student number:

{*department.course.student.stuNo.S*} → *department.course.student*

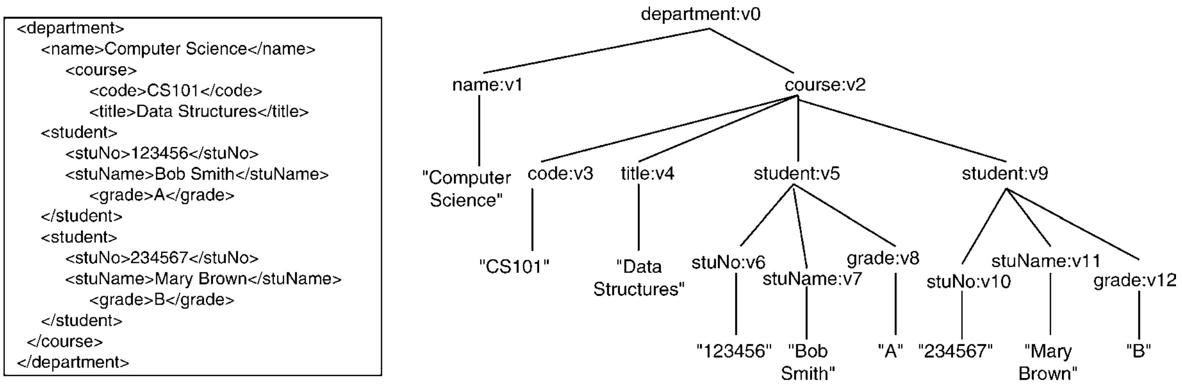
The constraint, that two student elements with the same value for *stuNo* must have the same value for *stuName*, is written:

department.course.student.stuNo.S → *department.course.student.stuName.S*

Note that this definition of functional dependencies combines node and value equality. This allows functional dependencies to be used to express not

t_1 (*department*) = υ0
t_1 (*department.name*) = υ1
t_1 (*department.name.S*) = "*Computer Science*"
t_1 (*department.course*) = υ2
t_1 (*department.course.code*) = υ3
t_1 (*department.course.code.S*) = "*CS*101"
t_1 (*department.course.title*) = υ4
t_1 (*department.course.title.S*) = "*Data Structures*"
t_1 (*department.course.student*) = υ5
t_1 (*department.course.student.stuNo*) = υ6
t_1 (*department.course.student.stuNo.S*) = "123456"
t_1 (*department.course.student.stuName*) = υ7
t_1 (*department.course.student.stuName.S*) = "*Bob Smith*"
t_1 (*department.course.student.grade*) = υ8
t_1 (*department.course.student.grade.S*) = "*A*"

t_2 (*department*) = υ0
t_2 (*department.name*) = υ1
t_2 (*department.name.S*) = "*Computer Science*"
t_2 (*department.course*) = υ2
t_2 (*department.course.code*) = υ3
t_2 (*department.course.code.S*) = "*CS*101"
t_2 (*department.course.title*) = υ4
t_2 (*department.course.title.S*) = "*Data Structures*"
t_2 (*department.course.student*) = υ9
t_2 (*department.course.student.stuNo*) = υ10
t_2 (*department.course.student.stuNo.S*) = "234567"
t_2 (*department.course.student.stuName*) = υ11
t_2 (*department.course.student.stuName.S*) = "*Mary Brown*"
t_2 (*department.course.student.grade*) = υ12
t_2 (*department.course.student.grade.S*) = "*B*"



```
<department>
    <name>Computer Science</name>
        <course>
            <code>CS101</code>
            <title>Data Structures</title>
    <student>
        <stuNo>123456</stuNo>
        <stuName>Bob Smith</stuName>
            <grade>A</grade>
    </student>
    <student>
        <stuNo>234567</stuNo>
        <stuName>Mary Brown</stuName>
            <grade>B</grade>
    </student>
  </course>
</department>
```

a An XML document

b An XML tree for the document in Figure 1(a)

Functional Dependencies for Semi-Structured Data. Figure 1. XML document and its corresponding XML tree.

only what the keys are but also to differentiate between absolute and relative constraints. That is, all students in a document have a unique student number can be expressed using an absolute constraint, and all students in a course have a unique student number can be expressed using a relative constraint. Note that the relative constraint is relative to another element or object class, in this case *course*, while the absolute constraint applies to the whole document. However, the richness of expressibility also means that functional dependencies in the semi-structured setting are more complicated than functional dependencies in the relational setting.

Cross-references

► Semi-structured Database Design

Recommended Reading

1. Arenas M. and Libkin L. A normal form for XML documents. ACM Trans. Database Syst., 29(1):195–232, 2004.
2. Lee M.L., Ling T.W., and Low W.L. Designing functional dependencies for XML. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 124–141.
3. Wu X., Ling T.W., Lee S.Y., Lee M.L., and Dobbie G. NF-SS: a normal form for semistructured schema. In Proceedings of ER Workshops LNCS, Vol. 2465/2002. 2001, pp. 292–305.

Functional Dependency

Solmaz Kolahi
University of British Columbia, Vancouver, BC, Canada

Synonyms

FD

Definition

Given a relation schema $R[U]$, a *functional dependency (FD)* is an expression of the form $X \to Y$, where $X,Y \subseteq U$. An instance I of $R[U]$ satisfies $X \to Y$, denoted by $I \models X \to Y$, if for every two tuples t_1,t_2 in I, $t_1[X] = t_2[X]$ implies $t_1[Y] = t_2[Y]$. That is, whenever two tuples contain the same values for attributes in X, they must have the same values for attributes in Y. A functional dependency $X \to Y$ is called *trivial* if $Y \subseteq X$.

A *key dependency* is a functional dependency of the form $X \to U$. Then X is called a *superkey* for relation R. If there is no proper subset Y of X such that Y is a superkey, then X is called a *key*.

Movies			
Title	*Director*	*Actor*	*Year*
The Godfather	Francis F. Coppola	Marlon Brando	1972
The Godfather	Francis F. Coppola	Al Pacino	1972
The Godfather	Francis F. Coppola	James Caan	1972
The Shining	Stanley Kubrick	Jack Nicholson	1980
The Shining	Stanley Kubrick	Shelley Duvall	1980

Key Points

Functional dependencies form an important class of integrity constraints that play a critical role in maintaining the integrity of data, query optimization and indexing, and especially schema design. The focus of the *normalization* technique in schema design is on avoiding redundancies caused by functional and other types of dependencies in relational databases. An example of such redundancies can be seen in the *year* column of the following table, where the functional dependency *title* →*year* holds.

In database design theory, there are normal forms that put restrictive conditions on data dependencies to control this kind of redundancy. Well-known normal forms that deal with FDs are second normal form (2NF), third normal form (3NF), and Boyce-Codd Normal Form (BCNF).

The implication problem for FDs can be solved in linear time. That is, given a set Σ of FDs, it is possible to check whether an FD $X \to Y$ is logically implied by Σ, denoted by $\Sigma \models X \to Y$, in the time that is linear in the size of Σ and $X \to Y$. A key concept for solving the implication problem is the *closure* of a set of attributes X, which is the set of attributes $A \in U$, denoted by X^+, such that $\Sigma \models X \to A$. There are efficient algorithms for finding the closure of a set of attributes [1].

The implication problem of functional dependencies can also be axiomatized. That is, there is a finite sound and complete set of inference rules, called *Armstrong axioms*, that can be used to solve the implication problem:

Reflexivity: If $Y \subseteq X$, then $X \to Y$.
Augmentation: If $X \to Y$, then $XZ \to YZ$.
Transitivity: If $X \to Y$ and $Y \to Z$, then $X \to Z$.

Cross-references

- ▶ Boyce-Codd Normal Form (BCNF)
- ▶ Key
- ▶ Normal Forms and Normalization
- ▶ Second Normal Form (2NF)
- ▶ Third Normal Form (3NF)

Recommended Reading

1. Abiteboul S., Hull R., Vianu V. Foundations of Databases. Addison-Wesley, Reading, MA, USA, 1995.

Functional Query Language

Peter M. D. Gray
University of Aberdeen, Aberdeen, UK

Definition

Functional Query Languages came from two lines of development:

1. From new functional programming languages such as FP that showed the value of *referential transparency*; this ensures that complex nested functional expressions can be evaluated inside-out (bottom-up) or top-down or even split up and done in parallel, with the same result. For optimization purposes this is vastly better than state-altering algorithms used in early CODASYL systems, or even code used today with embedded SQL (as in ODBC).

2. From requirements to provide a single query language and a single integrated schema over multiple autonomous, heterogeneous, distributed databases. This happened in the MULTIBASE project and resulted in the DAPLEX language [10]. It was the first functional query language to compute over instances of a *Functional Data Model*, for the purpose of abstracting away details of different storage schemas in a distributed DB.

Historical Background

No full implementation of DAPLEX was undertaken. However, later systems Adaplex, EFDM and *P/FDM* implemented large parts of the language. These showed how to integrate base functions (from the data model), derived functions, views, and integrity constraints into a single functional framework. The main drawback of DAPLEX was that for general computation it required either calling out to foreign functions, or embedding in a host programming language.

ADAPLEX was an embedding of a subset of DAPLEX into the programming language ADA. The data types of DAPLEX are reconciled with those of ADA by associating base functions with abstract entity types when these are declared. Also, derived functions are defined procedurally in ADA rather than in DAPLEX (this is an example of calling out to foreign functions).

EFDM [8] extended DAPLEX with procedural computation, recursive functions over abstract types and scalar types, built-in metalevel functions for interrogating the meta data, and a construct which can store EFDM statements in the database (like stored SQL procedures). The underlying database used persistent object storage techniques, similar to commercial object-oriented databases. The facilities for semantic integrity constraints considerably extend those of DAPLEX.

Following *FQL*, The functional database languages FDL [9] and PFL concentrated on a more general, computationally complete, functional language based on the lambda calculus.

This work was followed by the development of several functional languages, such as FAD [1] and FUGUE, for complex object and *object-oriented databases*. These capitalized on the fact that DAPLEX already had a built-in notion of *object identifier* for *entity* instances in its data model, but without committing to any particular implementation. This also inspired the Iris object-oriented database system, which used a LISP-like functional syntax. Iris in turned influenced the development of the original *AMOSQL* language used in active mediation of distributed data sources [5].

The OSQL query language [2] was intended for use with object-oriented databases. It unified functional and relational modeling by extending SQL to include entities and functions. It included primitives for creation of entities, for assignment of base function values, and for updating multi-valued base functions. Derived functions were defined by means of a SELECT clause and queries had an SQL-like syntax.

Foundations

One common thread running through almost all Functional Query Languages is the use of the functional programming concept of a *Comprehension* (q.v.) which deserves to be much more widely known. Buneman et al. [3] have generalized it for database use to include set and bag comprehensions. Fegaras [6] has related it to the algebraic structure of monoids which underlies *OQL*. This section makes general use of the notion of a Comprehension, because it describes precisely almost all the computations done in functional query languages, despite their very different surface syntax!

The comprehension crosses the borders between the lambda calculus and the predicate calculus since functions, just like predicates, can be used either as generators or as filters in collecting up sets of values. However, what matters is the mathematical theory worked out in connection with functional programming, which can be used, for example, to develop and prove correct new analysis and optimization techniques for queries. Generators may just be based on finite sets (or subranges) of integers, and filters can also do calculations. For example, suppose one wants the set of all right-angled triangles with whole-number sides less than 50. The comprehension for this neatly expresses the mathematical requirements:

```
[(x,y,z) | x <- [1..50]; y <- [1..50];
z <- isqrt(x*x + y*y); z*z = x*x + y*y;
z <50]
```

Here for each value of `x` between 1 and 50, one must explore all `y` values between 1 and 50, generate a value for the longest side `z` using a function to calculate the integer part of the square root, and then test that `z` is less than 50. The results are returned in the form of triples, such as `(3,4,5)`. This is a typical use of functions in numerical calculation. It also introduces the subject of *nested loops*. Notice that where one generator appears to the right of another, then its values have to be considered in combination with all values from the previous one, which is just like using nested loops in an ordinary programming language.

Now consider a query involving nested generators, which corresponds to the use of joins in SQL, but which appears in the OQL object database query language [4] as:

```
select x.name
from x in students, y in x.takes, z in y.
taught_by
where z.rank = "full prof"
```

This gives the names of students who take courses taught by full professors. The equivalent comprehension is

```
[name(x) | x <- students; y <- takes(x);
z <- taught_by(y);rank(z) = "full
prof"]
```

In a programming language supporting assignment and `for` loops, one would write the computation using nested loops, which show more clearly the role of the generators:

```
result := [];
for x in students()
  for y in takes(x)
    for z in taught_by(y)
      if rank(z) = "full prof"
      then result := result ++ [name(x)];
```

Note that the variables are now treated as holding object identifiers. More significantly, each variable is only introduced once with an arrow, and this must come before it is used in a predicate or as a parameter of another generator. Apart from this, generators and predicates can be reordered without altering the value of the comprehension, since it behaves like a conjunction of booleans.

This reordering is made easier by using *converse functions* of the form `f_inv` (equivalent to `Conv f` used in Functional Programming); just like an *inverted index*, this may enable one to move selections nearer to generators, so as to gain efficiency. For example, the following comprehension produces the same result more quickly by applying the filter on full professor much earlier:

```
[name(x) | z <- lecturers; rank(z) =
"full prof";
  y <- taught_by_inv(z); x <- takes_inv
  (y)]
```

This technique is widely used. Note that it is not necessary for the end users themselves to use `f_inv`; this need only happen within the query optimizer module of the DBMS.

In order to show that list comprehensions can be more complex, consider another query which returns the codes of all courses that have some section taught by a senior lecturer:

```
  [code(c) | c <- course;
  some([t | s <- sections(c); t <-
  lecturer(s);
    p <- position(t); p="SL"]) ]
```

Here, the inner comprehension depends on the variable `c` representing a course, which is bound by the outer comprehension. The function `some` is a predicate that tests whether its parameter (the inner comprehension) returns a non-empty list. It corresponds to the EXISTS predicate in SQL. In fact, the whole query can be translated into SQL automatically.

The following subsections show how comprehensions are used in a number of different query languages. Despite variations in syntax and in emphasis, all these systems make use of the same mathematical theory for their transformation and optimization.

Comprehensions in P/FDM and Constraints

P/FDM, like the examples in *AMOSQL* [5], uses list comprehensions internally but provides an equivalent query language syntax which suits regular programmers. In P/FDM the above comprehension is rendered as

```
for each c in course such that
  some t in lecturer(sections(c)) has
position(t)="SL"
    print(code(c));
```

This syntax is very close to Shipman's original DAPLEX language.

Comprehensions in P/FDM are also used to describe the semantics of integrity constraints, representing invariants that must be held true under updates. For this purpose the nested loop syntax used in queries is adapted, similarly to constraints in EFDM:

```
constrain each t in seniortutor
  each s in advisees(t)
    to have grade(s) > 60;
```

This constrains each person in the class `seniortutor` to have only advisees with grades over 60. It requires that the following list comprehension always computes an empty list:

```
[ t | t <- seniortutor; s <- advisees
(t); not (grade(s) > 60)]
```

The constraint is equivalent to the following formula in predicate logic, which shows the correspondence between the generators and the nested quantifiers and also the similarity in use of conjunctions. This demonstrates a fascinating connection between comprehensions and the predicate calculus:

$$(\forall t)\ seniortutor(t) \Rightarrow ((\forall s,g)\ advisee(t,s) \wedge grade(s,g) \Rightarrow g>60)$$

Comprehensions in Kleisli – Records, Lists and Mixed Types

In Kleisli [11], comprehensions are written with a comma as separator for conjunctions in place of a semicolon. Also, the first time that a variable iterates over a set of values (in some generator), its name needs to be prefixed with a backslash. Subsequent uses of the unprefixed variable, either in filters or in generators, make use of these bindings. Lastly, record field names are prefixed by a `#`. Thus, the initial example would read:

```
{p.#surname | \p <- person, \f <- p.
#forename, f = "Jim"}
```

If one wishes to create a *list* comprehension or *bag* comprehension, one simply encloses the above expression in different brackets `[ ... | ...]` or `{| ... | ...|}` respectively.

Thus the end user basically works with comprehensions, while the full power of functional programming is reserved for the implementers, working in Standard ML. This is an elegant use of functional programming but note that it is not essential to use a functional language for optimising comprehensions; for example, P/FDM uses Prolog while AMOS uses a version of LISP. Any good list processing language will do but functional languages are, of course, better at type checking.

Comprehensions in FDL and PFL

The functional database languages FDL [9] and PFL incorporate list comprehensions as part of a more general, computationally complete, functional language based on the lambda calculus. In such languages, comprehensions can be formalised as successive applications of a higher order function `flatmap`. Thus, for example, the following comprehension from earlier in this section:

```
[surname(p) | p <- person; f <- forename
(p); f = "Jim"]
```

would translate into the following expression:

```
flatmap (lambda p.
  flatmap (lambda f.if f = "Jim"
      then [surname(p)]
      else [])
    forename(p))
  person
```

`flatmap` is also an example of a *parametric polymorphism*, in that its type can be inferred to be `(a->[b])->[a]->[b]`, where `a` and `b` are *type variables* each of which can be replaced by *any* type. For example, the first occurrence of `flatmap` above has type `(Person->[String])->[Person]->[String]` while the second occurrence has type `(String->[String])->[String]->[String]`.

PFL has a similar type system to FDL, but uses a class of functions called *selectors* which allow storage, querying and update of sets of values of the same type.

Key Applications

The fundamental advantage of list comprehensions as a database abstraction is that they preserve *Data Independence* in a very clean and simple way. They do it by working entirely in terms of sets and functional relationships, regardless of how they are stored. This will seem strange to programmers who are used to carefully choosing between arrays of records or parallel arrays or linked lists or B-trees etc.

Database people know that large collections of data may exist in different forms on different computers, and may need to change form by restructuring on a single computer. Thus it is necessary to do the translation from a list comprehension expressed against a conceptual schema into a specific storage schema at compile time (often close to run time). Likewise it may be necessary to send part of a query to a remote server, since data is increasingly distributed in different forms. Thus the list comprehension is a good choice for passing a complex computational request between computers, free from assumptions about data storage or whether functions are computed or based on stored values.

Future Directions

About a decade ago, Stonebraker was predicting that SQL would become *Galactic Dataspeak*. With the widespread adoption of JDBC and PHP to access MySQL or much bigger commercial SQL databases, this would appear to be so. However, on the Internet there is increasing pressure to provide functionality as *Web Services* which can work on various encapsulated forms of structured and semi-structured data. Functional Query languages may well find a niche as convenient scripting languages for implementing such Web services. They allow a much richer variety of computation and can easily evolve to cope with new storage schemas, because they adhere to the Principle of Data Independence. As such, they should prove much easier to maintain in the fast-changing world of the Web. The functional language *Python* and its JAVA version *Jython* are increasingly popular, and have an explicit syntax for *Comprehensions.*

URL to Code

http://www.csd.abdn.ac.uk/˜pgray/FDMDownload.html
http://user.it.uu.se/˜udbl/amos/download.html
http://www.dcs.bbk.ac.uk/˜ap/pfl.html

Cross-references

- ► AMOSQL
- ► FQL
- ► Functional Data Model
- ► OQL
- ► P/FDM

For a general overview on the Functional Approach to modeling, integrating,querying and analyzing data see [7] from where much of this section is taken.

Recommended Reading

1. Bancilhon F., Briggs T., Khoshafian S., and Valduriez P. FAD, a powerful and simple database language. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 97–105.
2. Beech D. A foundation of evolution from relational to object databases. In Advances in Database Technology, Proc. 1st Int. Conf. on Extending Database Technology, 1988, pp. 251–270.
3. Buneman P., Libkin L., Suciu D., Tannen V., and Wong L. Comprehension syntax. ACM SIGMOD Rec., 23(1):87–96,1994.
4. Cattell R.G.G. (ed). The Object Data Standard: ODMG 3.0. Morgan Kaufmann, Los Altos, CA, 2000.
5. Fahl G., Risch T., and Sköld M. AMOS – an architecture for active mediators. In Proc. Workshop on Next Generation Information Technologies and Systems, 1993, pp. 47–53.
6. Fegaras L. and Maier D. Towards an effective calculus for Object Query Languages. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 47–58.
7. Gray P.M.D., Kerschberg L., King P.J.H., and Poulovassilis A. The Functional Approach to Data Management. Springer, Berlin, 2004.
8. Kulkarni K.G. and Atkinson M.P. EFDM : Extended Functional Data Model. Comput. J., 29(1):38–46, 1986.
9. Poulovassilis A. and King P.J.H. Extending the functional data model to computational completeness. In Advances in Database Technology, Proc. 2nd Int. Conf. on Extending Database Technology, 1990, pp. 75–91.
10. Shipman D.W. The functional data model and the data language DAPLEX. ACM Trans. Database Syst., 6(1):140 –173, 1981.
11. Wong L. Kleisli, a functional query system. J. Funct. Program., 10:19–56, 2000.

Fuzzy Information Retrieval

► Fuzzy Models

Fuzzy MCDM

▶ Fuzzy Set Approach

Fuzzy Models

Gabriella Pasi
University of Milano-Bicocca, Milan, Italy

Synonyms

Fuzzy information retrieval

Definition

The application of Fuzzy Set Theory to Information Retrieval is aimed to the definition of retrieval techniques capable of modeling, at least to some extent, the subjectivity, vagueness and imprecision that is intrinsic to the process of locating information relevant to some user's needs. In particular, Fuzzy Set Theory has been applied in the context of IR to the following main aims:

- To define generalizations of the Boolean retrieval model
- To deal with the imprecision and subjectivity that characterize the document indexing process
- To manage the user's vagueness in query formulation
- To soften the associative mechanisms, such as thesauri and documents' clustering algorithms, which are often employed to extend the functionalities of the basic IR scheme
- To define flexible aggregation strategies in meta search engines and to define flexible approaches to distributed IR
- To represent and inquiry semi-structured information (XML documents)

Historical Background

In 1979 one of the first proposals of application of Fuzzy Set Theory to IR was presented by Tadeusz Radecki [7]. His seminal paper, titled "Fuzzy set theoretical approach to document retrieval" constituted a first generalization of the Boolean retrieval model, followed by several subsequent proposals.

Fuzzy Set Theory was defined as a generalization of classical Set Theory, on which the Boolean information retrieval model is based [8]. A fuzzy set is a class of elements with unsharp boundaries suitable to represent vague concepts. Formally, a fuzzy subset A of a universe of discourse U is defined through a membership function μ_A: U → [0,1]; the value 1 indicates full membership of an element of U to A, the value 0 no membership, and a value between 0 and 1 partial membership. A finite fuzzy subset A of a set U is denoted by A = $\{\mu_A(u)/u\}$ with $u \in U$, in which the notation $\mu_A(u)/u$ indicates that with each element u of the universe U a membership degree $\mu_A(u)$ (in [0,1]) is associated, which denotes the membership of u to the subset.

The main aim of fuzzy generalizations of the Boolean IR model was to overcome its limitations, and to model relevance as a gradual property of documents with respect to a user's query, with the consequence of making an Information Retrieval System able to produce a document ranking [1,4,6]. In a fuzzy IR model a document is formally represented as a fuzzy subset of index terms (the membership value associated with a term represents its index term weight), and the Boolean query language can be extended by allowing the association of weights with query terms. In a generalized Boolean query, each term weight is interpreted as a specification of term importance, and it is formally defined as a flexible constraint (that is a constraint satisfied up to a partial extent) on the document representation. For example, a query term weight may be interpreted as a threshold, i.e., as a constraint satisfied by the documents having the indexing weight higher than the specified query term weight [1,4]. The exact matching of the Boolean model is in this way relaxed to a partial matching in fuzzy IR models, where the matching degree of a document to a query (the so called Retrieval Status Value of a document) is computed on the basis of an evaluation of the satisfaction of the flexible constraints specified in the query (by weighted terms).

Since Radecki's seminal paper, several new applications of Fuzzy Set Theory to IR have been proposed; some of them are described in the following section.

Foundations

In this section a more extensive explanation of Fuzzy IR models is presented, related in particular to the definition of flexible query languages (the key point of the above mentioned fuzzy generalizations of the Boolean model). Some other key applications of Fuzzy Set Theory to IR are also presented in distinct subsections.

Definition of Generalizations of the Boolean Query Language

By means of Fuzzy Set Theory two main generalizations of the Boolean query language have been proposed: the introduction of query term weights, and a generalization of the aggregation operators (the connectives AND and OR in the Boolean query language).

The first fuzzy models proposed in the literature introduced query term weights, by allowing the association of a weight with each query term. In this way the basic constraint specified in a query is a weighted term, identified by a pair <term, weight>. Query term weights express the "importance" of query terms as descriptors of users' needs, and are interpreted as flexible constraints on the index terms weights in each document representation. By such an extension, the structure of a Boolean query is maintained, by allowing weighted query terms to be aggregated by the AND, OR connectives and negated by the NOT operator. In the first fuzzy models query term weights were defined as numeric values in the range [0,1]. A numeric query term weight identifies a flexible constraint on the weighted document representations; such a constraint depends on the semantics of the query weight. Distinct semantics have been proposed for query weights, corresponding to distinct fuzzy generalizations of the Boolean model (distinct fuzzy IR models) [1,4,6]. The three main semantics for query term weights are: the relative importance semantics (query weights express the relative importance of pairs of terms in a query), the threshold semantics (a query weight expresses a threshold on index term weights), and the ideal index term weight semantics (a query weight expresses the "perfect" index term weight).

The choice of one out of the three proposed query weight semantics implies a distinct definition of the partial matching function which evaluates a weighted query term. A flexible constraint imposed by a weighted query term <t, *weight*> is formally defined as a fuzzy subset of the set [0,1] of the index term weights. For a given document d, the membership value μ_{weight}(F(d,t)) is interpreted as the degree of satisfaction of the flexible constraint imposed by the *weight* associated with query term t by the index term weight of t in document d (the value F(d,t)). This means that the partial evaluation function is the membership function of the fuzzy subset identified by the query term weight (i.e., the function μ_{weight}). The result of the evaluation of a weighted query term is a fuzzy set: {μ_{weight}(F(d,t))/d)}, in which the membership degree of a document is interpreted as the degree of relevance of the document to the query (the Retrieval Status Value), and is used to rank the documents.

In structured Boolean queries, the evaluation of the AND and OR connectives is applied to the fuzzy subsets returned by the weighted query terms evaluations.

In the context of Fuzzy Set Theory the connectives AND and OR are defined as aggregation operators belonging to the class of T-norms and T-conorms respectively. Usually, the AND is defined as the *min* (minimum) aggregation operator, and the OR as the *max* (maximum) aggregation operator. A Boolean expression on weighted query terms is usually evaluated by a bottom-up evaluation procedure (except in the case of relative importance semantics): first, each atomic selection condition (flexible constraint. i.e., each weighted term) in the query is evaluated for a given document, and then the aggregation operators are applied to the obtained values starting from the inmost operator in the query to the outermost operator.

As the association of a numeric value forces the user to quantify the qualitative concept of importance of query terms, some late models proposed in the literature have formalized linguistic extensions of the Boolean query language, based on the concept of linguistic variable [1,4]. The values of a linguistic variable are linguistic terms formally defined as fuzzy subsets of a given reference domain; for example in the case one wants to define *age* as a linguistic variable, some possible linguistic values can be *young*, *old*, formally defined as fuzzy subsets of the set [0,130] of the possible numeric values for the age of a human being. By using linguistic query weights, query terms can then be labeled by the words "*important*," "*very important*" or "*fairly important*." Similarly to the evaluation of numeric query term weights, a pair <t, *important*> expresses a flexible constraint evaluated by the function $\mu_{important}$ on the index term weights (the F(d,t) values). The evaluation of the relevance of a given document d to a query consisting of the pair <t, *important*> is then computed by applying the function $\mu_{important}$ to the value F(d,t). Also in this case the membership function of the fuzzy set associated with a linguistic weight depends on the weight semantics.

A second kind of generalization of the Boolean query language has concerned the definition of aggregation operators (AND and OR in the Boolean query language) [1,4,6]. When employing the Boolean retrieval model, if the AND is used for aggregating *M* keywords in a user query, a document indexed by all keywords but one is not retrieved, thus causing the possible rejection of useful items. The opposite behavior characterizes the aggregation by OR. To express more flexible aggregations, the use of linguistic quantifiers (formally defined within Fuzzy Set Theory) has been proposed. Linguistic quantifiers, such as *at least 2* and *most*, specify more flexible document selection strategies. Linguistic quantifiers have been formally defined as averaging aggregation operators, the behavior of which lies between the behavior of the AND and the OR connectives, which correspond to the *all* and the *at least one* linguistic quantifiers.

Another recent research direction is aimed at the definition of flexible query languages to inquire XML documents. Fuzzy set theory is being applied to define extensions of XML query languages so as to make possible the expression of flexible selection conditions on both the documents' structure and contents.

Flexible Indexing of Semi-Structured Documents

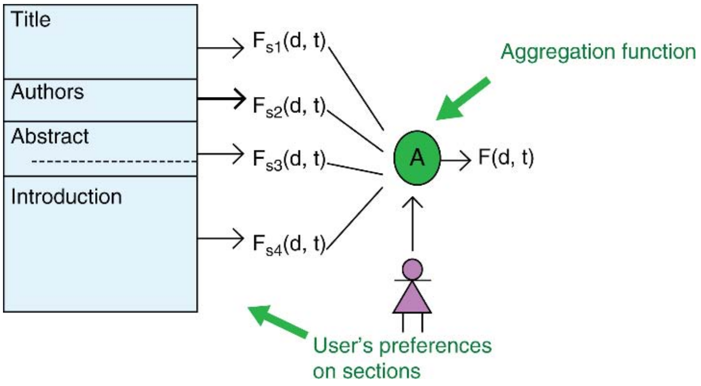
The diffusion of semi-structured documents has encouraged the definition of indexing models which take into account the information conveyed by the "structure" of the documents.

The usual tf*idf indexing schema adopted by IRSs does not take into account the distinct informative role that a term occurrence may have in distinct document sections. For example, considering the structure of a scientific paper, usually organized in sections such as *title, authors, abstract, introduction, references*, an occurrence of a term in the *title* has a distinct informative role than its occurrence in the *references* section. Moreover the usual indexing functions produce the same document representation to all users; this enhances the system's efficiency but implies a loss of effectiveness. In fact, when examining a structured document, users have their personal views of the document's information content. Users would naturally privilege the search in some subparts of the documents' structure, depending on their preferences. This observation has supported the idea of flexible and personalized indexing, first proposed in 1995 [2]. Such an indexing model is constituted by a static component and by an adaptive query-evaluation component; the static component provides an a priori computation of an index term weight for each logical section of the document.

The adaptive component may be activated by a user interaction during query formulation and provides an aggregation strategy of the *n* index term weights (where *n* is the number of sections) into an overall index term weight. The user is allowed to express preferences on the document sections, outlining those that the system should more heavily take into account in computing the overall index term weight. This user preference on the document structure is exploited to enhance the computation of index term weights: the importance of index terms is strictly related to the importance to the user of the logical sections in which they appear. The user may also decide which kind of aggregation to apply for producing the overall significance degree (see Fig. 1). This can be done by the specification of a linguistic quantifier such as *at least k* (with k an integer number) and *most*.

By adopting such an indexing model a same query may produce different document rankings if formulated by distinct users expressing distinct preferences on the documents sections.

More recently, an increasing number of approaches have proposed IR models which are based on concepts rather than keywords, thus modeling document representations at a higher level of granularity, trying to



Fuzzy Models. Figure 1. Sketch of the flexible indexing model.

describe the topical content and structure of documents. These efforts gave raise to the so called concept-based Information Retrieval, which aims at retrieving relevant documents on the basis of their meaning rather than their keywords. In this context some fuzzy set approaches to concept-based Information Retrieval have been proposed.

Fuzzy Associative Mechanisms

Associative retrieval mechanisms are defined to enhance the retrieval capability of traditional IRSs. They work by retrieving additional documents not directly indexed by the terms in a given query but indexed by terms associated to those specified in the query. The most common type of associative retrieval mechanism is based on the use of a thesaurus to associate entry terms with related terms. In traditional associative retrieval the associations are crisp.

The fuzzy associative retrieval mechanisms (first proposed in 1976) are based on the concept of fuzzy association [1,4,5]. A fuzzy association between two sets $X = \{x_1,...,x_m\}$ and $Y = \{y_1,...,y_n\}$ is formally defined as a fuzzy relation $f: X \times Y \rightarrow [0,1]$: the value $f(x, y)$ represents the degree of strength of the association existing between the values $x \in X$ and $y \in Y$.

In Information Retrieval, different kinds of fuzzy associations can be modeled depending on the semantics of the sets X and Y. Fuzzy associative mechanisms are represented by fuzzy thesauri, fuzzy pseudo-thesauri, and fuzzy clustering techniques.

Some authors have proposed the definition of fuzzy thesauri, where the links between terms are weighted to indicate strength of association. Moreover, this notion includes generalizations such as fuzzy pseudo-thesauri, and fuzzy associations based on a citation index [5].

Fuzzy associative mechanisms based on thesauri or clustering techniques have been defined in order to cope with the incompleteness characterizing either the representation of documents or the users' queries. Fuzzy thesauri and pseudo-thesauri can be used to expand the set of index terms of documents with new terms by taking into account their varying significance in representing the topics dealt with in the documents; the degree of significance of the associated terms depends on the strength of the associations with the documents' descriptors. An alternative use of fuzzy thesauri and pseudo-thesauri is to expand each of the search terms in the query with associated terms, by taking into account their distinct importance in representing the concepts of interest; the varying importance is dependent on the associations' strength with the search terms.

Fuzzy Approaches to Distributed Information Retrieval

In distributed information retrieval, there are two main models. In the first model, the information is considered as belonging to a unique, huge, centralized database which is distributed but centrally indexed for retrieval purposes. A second model is based on the distribution of the information on distinct repositories, independently indexed, and thus constituting distinct information sources. In this second case, the repositories reside on distinct servers, each of which can be provided with its own search engine (IRS).

The multi-source information retrieval paradigm is more complex than the centralized model. This paradigm presents additional problems, such as the selection of an appropriate information source for a given information need. This task of distributing retrieval is affected by uncertainty, since a decision must be taken based on an incomplete description of the information source. Furthermore, a common problem in both models is the list fusion task.

Some fuzzy methods have been defined to address the above mentioned problems of source selection and ranked list fusion [3]. In particular, a meta-search model has been recently proposed where the fusion of overlapping ordered lists into an overall ordered list is regarded as a group decision making activity in which the search engines play the role of the experts, the documents are the alternatives that are evaluated based on a set of criteria expressed in a user query, and the decision function is a soft aggregation operator which allows to take into account the lists priority, allowing to model a specific user retrieval attitude [3].

Key Applications

The main applications of Fuzzy Models in IR are aimed at defining new Information Retrieval Systems, to enhance query languages, to define new indexing algorithms, and to define associative mechanisms.

Cross-references

- ▶ Digital Libraries
- ▶ Text Indexing Techniques
- ▶ Text Retrieval

Recommended Reading

1. Bordogna G. and Pasi G. Modelling vagueness in information retrieval. In Lectures in Information Retrieval, M. Agosti, F. Crestani, G. Pasi (eds.). Springer, Berlin, 2001.
2. Bordogna G. and Pasi G. Personalized indexing and retrieval of heterogeneous structured documents. Inf. Retrieval, 8 (2):301–318, 2005.
3. Bordogna G., Pasi G., and Yager R.R. Soft approaches to distributed information retrieval. Int. J. Approx. Reasoning, 34 (2–3):105–120, 2003.
4. Kraft D.H., Bordogna G., and Pasi G. Fuzzy set techniques in information retrieval. In Fuzzy Sets in Approximate Reasoning and Information Systems, Series: The Handbooks of Fuzzy Sets Series. J.C. Bezdek, D. Dubois, H. Prade (eds.). Kluwer Academic, Norwell, MA, 1999, pp. 469–510.
5. Miyamoto S. Fuzzy Sets in Information Retrieval and Cluster Analysis. Kluwer Academic, Dordrecht, 1990.
6. Pasi G. Fuzzy Sets in Information Retrieval: State of the Art and Research trends. In Fuzzy Sets and Their Extensions: Representation, Aggregation and Models. Intelligent Systems from Decision Making to Data Mining, Web Intelligence and Computer Vision, Series: Studies in Fuzziness and Soft Computing. H. Bustince, F. Herrera, and J. Montero (eds.). Springer, Berlin, 2008, pp. 517–535.
7. Radecki T. Fuzzy set theoretical approach to document retrieval. Inf. Process. Manag., 15(5):247–260, 1979.
8. Zadeh L. Fuzzy Sets. Inf. Control, 8:338–353, 1965.

Fuzzy Multicriteria Decision Making

▶ Fuzzy Set Approach

Fuzzy Relation

Vilém Novák
University of Ostrava, Ostrava, Czech Republic

Definition

An n-ary fuzzy relation R is a fuzzy set (see FUZZY SET) in the universe $U_1 \times \ldots \times U_n$, i.e., $R \underset{\sim}{\subseteq} U_1 \times \ldots \times U_n$. A special case is a binary fuzzy relation in the universe $U \times V$, i.e., $R \underset{\sim}{\subseteq} U \times V$.

Key Points

Mathematically, an n-ary fuzzy relation is a function R: $U_1 \times \ldots \times U_n \rightarrow L$ where L is a residuated lattice.

Let $R \underset{\sim}{\subseteq} U \times V$ and $S \underset{\sim}{\subseteq} V \times W$ be two binary fuzzy relations. The *composition* of R and S is a fuzzy relation $R \circ S$ determined by the membership function

$$(R \circ S)(x, z) = \bigvee_{y \in V} (R(x, y) \wedge S(y, z)), \qquad x \in U, z \in W. \tag{1}$$

The operation $\wedge$ in ((1)) can be replaced by arbitrary t-norm.

Cross-references

▶ Fuzzy Set
▶ Residuated Lattice
▶ Triangular Norms
▶ t-Norm

Recommended Reading

1. Klement E.P., Mesiar R., and Pap E. Triangular Norms. Kluwer, Dordrecht, 2000.
2. Klir G.J. and Yuan B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice-Hall, New York, 1995.
3. Novák V. Fuzzy Sets and Their Applications. Adam Hilger, Bristol, 1989.
4. Novák V., Perfilieva I., and Močkoř J. Mathematical Principles of Fuzzy Logic. Kluwer, Boston/Dordrecht, 1999.

Fuzzy Set

Vilém Novák
University of Ostrava, Ostrava, Czech Republic

Definition

A fuzzy set is a function $A : U \rightarrow L$ where U is an ordinary set of elements called *universe* and L is a scale which is usually supposed to have the structure of a residuated lattice (see RESIDUATED LATTICE). The function A is at the same time called also *membership function*, i.e., fuzzy set is identified with its membership function.

If $x \in U$ is an element then $A(x) \in L$ is a *membership degree* of x in the fuzzy set A. It can also be interpreted as a *degree of truth* of the fact that the element x belongs to the fuzzy set A. If U is a universe and A a fuzzy set then it is convenient to write $A \underset{\sim}{\subseteq} U$.

Key Points

Fuzzy sets can be explicitly written in the form

$$\{a/u \mid a \in L, u \in U\}$$

where $a \in L$ is a membership degree of an element $u \in U$.

Given a residuated lattice (see RESIDUATED LATTICE) as the structure of truth values, the basic operations with fuzzy sets can be defined as follows:

$$\begin{aligned} &(\mathrm{union}) && (\mathrm{A} \cup \mathrm{B})(\mathrm{x}) = \mathrm{A}(\mathrm{x}) \vee \mathrm{B}(\mathrm{x}), \\ &(\mathrm{intersection}) && (\mathrm{A} \cap \mathrm{B})(\mathrm{x}) = \mathrm{A}(\mathrm{x}) \wedge \mathrm{B}(\mathrm{x}), \\ &(\mathrm{complement}) && \overline{\mathrm{A}}(\mathrm{x}) = \neg \mathrm{A}(\mathrm{x}) \end{aligned}$$

for all $x \in U$ where $\neg$ is a negation in residuated lattice. If the latter is a standard Łukasiewicz MV-algebra then

$$\overline{A}(x) = 1 - A(x).$$

Many other possible operations with fuzzy sets can be defined on the basis of operations in residuated lattice. In case that $L = [0,1]$, the union can be more generally defined using a t-conorm and the intersection using a t-norm.

Note that $\langle \{0,1\}, \vee, \wedge, \wedge, \rightarrow, 0, 1\rangle$ is also a residuated lattice because in this case $\otimes = \wedge$ is the ordinary conjunction and $\rightarrow$ is the classical boolean (material) implication. Therefore, the above operations naturally generalize classical set operations.

The *support* of a fuzzy set $A \underset{\sim}{\subset} U$ is a classical set $\mathrm{Supp}(A) = \{x \mid A(x) > 0\}$. The *kernel* of a fuzzy set is a set $\mathrm{Ker}(A) = \{x \mid A(x) = 1\}$. A fuzzy set A is *normal* if $\mathrm{Ker}(A) \neq \emptyset$.

A fuzzy set $A \subseteq \mathbb{R}$ is called a *fuzzy number* if it has one element kernel $\mathrm{Ker}(A) = \{x_0\}$, bounded support, and $A(x) \geq A(y) \wedge A(z)$ holds for all $y \leq x \leq z$. Such a fuzzy set can thus be taken as interpretation of the expression "approximately x_0."

The above notions can be illustrated on a simple example. Let the universe be a finite set $U = \{a, b, c, d, e\}$ and let L be the standard Łukasiewicz MV-algebra $\langle [0, 1], \max, \min, \otimes, \rightarrow, 0, 1\rangle$. (see RESIDUATED LATTICE) Let

$$\begin{aligned} A &= \{^{0.1}/a,^{1}/b,^{0.8}/c\}, \\ B &= \{^{0.7}/b,^{0.3}/c,^{0.9}/d,^{0.2}/e\} \end{aligned}$$

(The omitted elements have membership degree equal to 0.). Then

$$\begin{aligned} A \cap B &= \{^{0.7}/b,^{0.3}/c\}, \\ A \cap B &= \{^{0.1}/a,^{1}/b,^{0.8}/c,^{0.9}/d,^{0.2}/e\}, \\ \overline{A}(x) &= \{^{0.9}/a,^{0.2}/c,^{1}/d,^{1}/e\}, \\ \mathrm{Supp}(A) &= \{a, b, c\}, \\ \mathrm{Ker}(A) &= \{b\}. \end{aligned}$$

Example of other operation can be bold intersection

$$A \boxtimes B = \{^{0.7}/b,^{0.1}/c\}$$

using a Łukasiewicz product (this is also a t-norm) where, e.g., $(A \boxtimes B)(c) = \max\{0, 0.8 + 0.3 - 1\} = 0.1$.

Cross-references

▶ Fuzzy Relation
▶ Residuation
▶ Residuated Lattice
▶ Triangular Norms

Recommended Reading

1. Klir G.J. and Yuan B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice-Hall, New York, 1995.
2. Novák V. Fuzzy Sets and Their Applications. Adam Hilger, Bristol, 1989.
3. Novák V., Perfilieva I., and Močkoř J. Mathematical Principles of Fuzzy Logic. Kluwer, Boston/Dordrecht, 1999.

Fuzzy Set Approach

VILÉM NOVÁK
University of Ostrava, Ostrava, Czech Republic

Synonyms

Fuzzy MCDM; Fuzzy multicriteria decision making

Definition

The classical multicriteria decision making theory is based on the assumption that all the criteria can be characterized precisely so that it is possible to decide unambiguously, whether each alternative fulfils the given criterion or not. However, this is rarely the case in practice and so, the fuzzy set approach has been proposed which makes it possible to assume that the criteria can be evaluated imprecisely, for example "high quality, low reliability, very low weight," etc. Unlike classical approach which first dissolves imprecision and then constructs a decision model, the fuzzy set approach dissolves imprecision only at the very end, if necessary.

The basic concepts of fuzzy decision making are the following:

1. Decision based on the imprecisely defined set of alternatives, i.e., a fuzzy set of alternatives. This is called *decision in a fuzzy environment*.

2. Decision based on aggregation of imprecisely defined (fuzzy) preferences among alternatives.
3. Decision based on evaluation of alternatives using linguistic description of the decision situation.

The linguistic description in Case 3 consists of a set of fuzzy/linguistic IF-THEN rules, i.e., special expressions of the form

$$\text{IF } X \text{ is } \mathcal{A} \text{ THEN } Y \text{ is } \mathcal{B}$$

where X, Y are criteria and $\mathcal{A}, \mathcal{B}$ are imprecise characterizations (for example, evaluative expressions of natural language) of how well the given alternative meets the criteria. The linguistic expressions are often interpreted using fuzzy sets in various ways. The most advanced possibility takes fuzzy IF-THEN rules as conditional sentences of natural language.

A problem related to decision making is *classification*. The main task is to assign the given alternative to one of several classes. In practice, this may be quite difficult since the classes themselves can also be imprecise. For example, shoe numbers represent imprecise classes of foot sizes. The most advanced fuzzy classification methods are based on application of fuzzy IF-THEN rules.

Historical Background

The first paper on fuzzy multicriteria decision making has been written by Bellman and Zadeh in 1970 [1]. Their idea was very simple but at the same time quite powerful and it gave inspiration to many authors of the subsequent papers on fuzzy decision making. The paper started research leading to various kinds of approaches that improved the proposed method on one side and brought a lot of other new ideas and methods on the other side. After period of spontaneous development that lasted until mid of 1980s, the first systematic works appeared, namely the books [4,8].

A significant step forwards is the book by J. Fodor and M. Roubens [6] which is the first sound introduction to valued preference modeling using systematic use of fuzzy set theory and functional equations. Since then, many other papers and books have been published, e.g., [3,13]. Notable is the book by T. J. Saaty [11] which describes the, so called, *Analytic Hierarchy Process* that now belongs to one of the most widely used decision methods in practice.

The number of works on fuzzy multicriteria decision making now counts to thousands. There is also a working group EUROFUSE which regularly organizes various kinds of meetings or co-organizes conferences.

Foundations

Decision making in a fuzzy environment is realized as follows: Let a finite set of alternatives $A = \{a_1,\ldots,a_n\}$ be given. The goal is to choose the best alternative according to m criteria that, however, may be delineated only imprecisely, for example *reasonable price, nice view, great reliability, high safety*, etc.

The criteria are specified by means of defining a fuzzy set of alternatives that fulfil the given criterion in various degrees. Hence, each criterion G_j, $j = 1,\ldots,m$ is identified with a fuzzy set $G_j \underset{\sim}{\subset} A$ of alternatives fulfilling it. The membership degree $G_j(a_i)$ expresses the degree, in which the given alternative $a_i \in A$ fulfils the criterion G_j.

At the same time also constraints must be specified with respect to each criterion, for example *affordable price, view to the forest, acceptable reliability, acceptable safety*, etc. Similarly as the criteria, the constraints are also specified as fuzzy sets of alternatives. This means that each constraint C_j is defined as a fuzzy set $C_j \underset{\sim}{\subset} A$ of alternatives fulfilling C_j in various degrees. Consequently, for each j, $j = 1,\ldots,m$, two fuzzy sets of alternatives are specified: a fuzzy set G_j representing criterion and a fuzzy set C_j representing constraint.

For example, the criterion G_j can be *reasonable price* while the constraint C_j can be *affordable price*. The latter means that not each reasonable price can be for the decision-maker at the same time also affordable. If a_i is a certain house that a decision-maker wants to buy then the membership degree $G_j(a_i)$ represents the degree in which the price of a_i is reasonable (e.g., $G_j(a_i) = 0.8$ means that it is in 80% true that the price of a_i is reasonable). Quite analogously, the membership degree $C_j(a_i)$ is a degree in which the price of a_i is affordable for the decision-maker. For example, the set A of houses (alternatives) may contain also a castle a_k at a very reasonable price – several millions of dollars – but this price can be completely non-affordable for the decision-maker because he/she has, say, less than one million at disposal. In this case, set $C_j(a_k) = 0$.

The final decision is a fuzzy set of alternatives obtained by composition

$$D = (G_1 \alpha C_1)\beta \ldots \beta(G_m \alpha C_m) \tag{1}$$

where α, β are suitable fuzzy set operations. The best alternative is an alternative $a_k \in A$ with the highest membership degree $D(a_k)$.

Setting $\alpha = \beta = \cap$ (intersection) leads to *pessimistic decision*. With respect to the above example, $G_j(a_j) \wedge C_j(a_j)$ is a minimum of the degrees in which the house a_j has a reasonable and affordable price, i.e., the worse of both. Hence, for the castle a_k it is immediately obtained that $G_j(a_k) \wedge C_j(a_k) = 0$ and so $D(a_k) = 0$ which means that the castle is surely not chosen as the best alternative.

Setting $\beta = \cup$ (union) leads to *optimistic decision*. The best, however, seems compensatory decision which combines both intersection as well as union in various degrees. More details can be found in [19,14].

It is also possible to omit constraints from (1) and to consider the criteria $G_1,\ldots,G_m$ only. Then, the problem is raised how all the membership degrees $G_j(a_i)$ should be aggregated to obtain the global satisfaction degree $D(a_i)$ of the given alternative a_i. This requires the use of a specific *aggregation operator* $\mathcal{H}^m$ so that

$$D(a_i) = \mathcal{H}^m(G_1(a_i),\ldots,G_m(a_i)).$$

The symbol $\mathcal{H}^m$ denotes actual number of arguments of $\mathcal{H}$ (their number can vary).

Basic properties of aggregation operators are the following:

$$\begin{aligned}
&\mathcal{H}^1(a) = a,\\
&\mathcal{H}^m(0,\ldots,0) = 0 \text{ and } \mathcal{H}^m(1,\ldots,1) = 1,\\
&\text{if}\,(a_1,\ldots,a_m) \le (b_1,\ldots,b_m) \text{ then}\\
&\quad \mathcal{H}^m(a_1,\ldots,a_m) \ \le\ \mathcal{H}^m(b_1,\ldots,b_m).
\end{aligned}$$

Many other conditions can be imposed on aggregation operators, for example continuity, symmetry, associativity, idempotency, compensation, etc. The possible operators are classified as *conjunctive operators* (e.g., t-norms), *disjunctive operators* (e.g., t-conorms), *mean operators* (e.g., arithmetic mean), or more specific ones, such as *compensative operators* or *order weighted averaging operators* (OWA operators). Many details can be found in [2,7,13].

The criteria, however, are usually not equally important. Their relative importance is expressed using weights that can be numbers assigned to each criterion or/and constraint. The most popular method for weights assignment is *Analytical Hierarchy Process* (AHP) that itself can be used as a specific fuzzy decision method. It involves structuring multiple choice criteria into a hierarchy, assessing the relative importance of these criteria, comparing alternatives for each criterion, and determining an overall ranking of the alternatives. Thus, it can be divided into four phases:

1. *Decomposing*, i.e., the problem is structured into humanly-manageable sub-problems.
2. *Weighing*, i.e., a relative weight is assigned to each criterion, based on its importance within the node to which it belongs. A global priority is computed that quantifies the relative importance of a criterion within the overall decision model.
3. *Evaluating*, i.e., alternatives are scored and compared each one to others.
4. *Selecting*, i.e., an alternative fitting best the requirements is selected.

For the details about AHP, see [11].

Imprecisely defined preference is mathematically modeled using a binary *fuzzy preference relation* $R \underset{\sim}{\subseteq} A \times A$. This is determined by triple of binary fuzzy relations $\langle P, I, J \rangle$ where $P \underset{\sim}{\subseteq} A \times A$ is a strict fuzzy preference, $I \underset{\sim}{\subseteq} A \times A$ is a fuzzy indifference, and $J \underset{\sim}{\subseteq} A \times A$ is a fuzzy incomparability. This means, that $P(a_i, a_j)$ is a degree, in which the alternative a_i is strictly preferred to a_j. Similarly, $I(a_i, a_j)$ is the degree in which a_i is indifferent to a_j, and $J(a_i, a_j)$ is the degree in which a_i is incomparable with a_j. In other words, the given relation need not hold in full but only partially. This corresponds well with the real situations in which it needs not be fully convincing that, for example, "high building" is strictly nicer than "low" one, etc.

The definition of P, I, J and their further properties depend on the choice of the structure of truth values which is a specific residuated lattice. The solution is not unique and depends on other conditions that can be imposed on these relations. One possible solution is to assume that truth values form the standard Łukasiewicz MV-algebra and to put

$$P(a_i, a_j) = R(a_i, a_j) \wedge \neg R(a_j, a_i), \tag{2}$$

$$I(a_i, a_j) = R(a_i, a_j) \otimes R(a_j, a_i), \tag{3}$$

$$J(a_i, a_j) = \neg R(a_i, a_j) \otimes \neg R(a_j, a_i) \tag{4}$$

where $\bigotimes$ is the Łukasiewicz conjunction $a \bigotimes b = 0 \vee (a + b - 1)$. The detailed analysis and justification can be found in [6,13].

A very powerful general technique suitable also for applications in decision making as well as in classification are fuzzy/linguistic IF-THEN rules. The given decision situation is described using linguistic description and then, each alternative is judged using it on the basis of the measured characteristics for each criterion. The linguistic form of fuzzy/linguistic IF-THEN rules makes it possible to distinguish sufficiently subtly and, at the same time, aptly, various degrees of fulfilment of the respective criteria, their various importance and, moreover, it may also overcome possible discrepancies. Hence, the problem of assignment of weights to the criteria disappears. This makes fuzzy/linguistic IF-THEN rules very attractive for decision-making because the problem of weights assignment belongs to the most controversial problem in its applications. Another advantage is the possibility to include also information that can be quantified with great difficulties. For example "aesthetic quality," "overall impression," etc. are criteria which people can evaluate using expressions of natural language such as "very high, quite low, great, medium," etc. Such expressions (they are called *evaluative linguistic expressions*), however, can be used in fuzzy/linguistic IF-THEN rules without problems.

The procedure for decision support using fuzzy/linguistic IF-THEN rules is analogous to the AHP procedure mentioned above with the exception that the weighing phase is omitted. The first phase consists in decomposition of the decision problem into subproblems, each of which being characterized by a specific linguistic description. Hence, the decision situation is described using a hierarchical system of linguistic descriptions which, at the same time, renders evaluation:

$$\mathcal{R}_{1k} : \text{IF } G_{k1} \text{ is } \mathcal{A}_{11} \text{ AND } \dots$$
$$\text{AND } G_{kn(k)} \text{ is } \mathcal{A}_{1n(k)} \text{ THEN } H_k \text{ is } \mathcal{B}_1$$

....

$$\mathcal{R}_{1p(k)} : \text{IF } G_{k1} \text{ is } \mathcal{A}_{p(k)1} \text{ AND } \dots$$
$$\text{AND } G_{kn(k)} \text{ is } \mathcal{A}_{p(k)n(k)} \text{ THEN } H_k \text{ is } \mathcal{B}_{p(k)}$$

$k = 1,\dots,r$ and

$$\mathcal{R}_1 : \text{IF } H_1 \text{ is } \mathcal{A}_{11} \text{ AND } \dots$$
$$\text{AND } H_r \text{ is } \mathcal{A}_{1r} \text{ THEN } H \text{ is } \mathcal{B}_1$$

....

$$\mathcal{R}_s : \text{IF } H_1 \text{ is } \mathcal{A}_{s1} \text{ AND } \dots$$
$$\text{AND } H_r \text{ is } \mathcal{A}_{sr} \text{ THEN } H \text{ is } \mathcal{B}_s$$

where $H_1,\dots,H_r$ are local evaluations, H is a global evaluation and $\mathcal{A}$'s and $\mathcal{B}$'s are evaluative linguistic expressions. The final decision is made on the basis of the values of the global evaluation H (usually, the higher, the better). A typical example of the above rules is, for example, "IF price is small AND maintenance is more or less medium THEN economical conditions are good," or "IF house is stylish AND place is very nice THEN aesthetic quality is significantly high," etc. On the basis of initial information about each alternative for all aspects, the inference proceeds. The most convenient inference method at this step is *perception-based logical deduction*. Note that other methods can be used as well but one must be careful about defining the shapes of fuzzy sets and so, the advantage of using natural language is limited. The best alternative is selected on the basis of the highest (or, possibly, the lowest) value of the global evaluation H.

A specific task belonging also to the realm of decision making is *classification*. In general this is a procedure that assigns an element from a finite set $\Omega = \{\omega_1,\dots,\omega_c\}$ to a vector $\mathbf{x} \in S$, $S \subset \mathbb{R}$ of numbers. The set S is a *feature space* and Ω is a set of classes that can be, e.g., pieces of a figure, psychological categories, shoe numbers, diseases, etc. The need for fuzzy classification is raised in the moment when there is not a sufficient or precise information available or the character of classes is imprecise so that membership in them can be unclear and only some degree can be provided, etc. Such situations are very usual in the real life.

A specific case are fuzzy IF-THEN classifiers that are linguistic descriptions with clearly distinguished consequences. The latter can be either crisp or fuzzy numbers. Further elaboration is the same as above.

Key Applications

There are several thousands of real applications of fuzzy multicriteria decision making. One of the earliest of them was evaluation of the credit-worthiness of credit applicants developed in Germany (see [14]).

Other essential application is Yamaichi Fuzzy Fund which handles 65 industries and a majority of the stocks listed on Nikkei Dow. It is based on the application of fuzzy IF-THEN rules which are determined monthly by a group of experts and modified by senior business analysts when necessary. The system was tested for 2 years, and its performance in terms of the return and growth exceeds the Nikkei Avarage by over 20%. For comparison, the system recommended "sell" 18 days before the Black Monday in 1987. The system went to commercial operations in 1988.

Some other ones are, evaluation of weapon systems, technology transfer strategy selection in biotechnology, aggregation of market research data and thousands others. Evaluation of weapon systems has been provided using the Analytical Hierarchy Process (AHP) based on fuzzy scales.

Typical features of the weapon system evaluation were the following: the objectives of the evaluations are generally multiple and generally in conflict, and the descriptions of the weapon systems are usually linguistic and vague. The details are in [5]. It should be stressed that such features are typical for most practical decision problems and this is the main reason for using fuzzy set theory. In case of linguistically provided information, especially fuzzy IF-THEN rules taken as conditional statements of natural language are convenient tool since they make possible to include also non-quantifiable information. Such application was described in [10].

Cross-references

- ▶ Classification
- ▶ Classification by Association Rule Analysis
- ▶ Clustering
- ▶ Decision Tree Classification
- ▶ Decision Trees
- ▶ Fuzzy Models
- ▶ Hierarchial Clustering
- ▶ Rule-Based Classification

Recommended Reading

1. Bellman R. and Zadeh L.A. Decision making in a fuzzy environment. Manage. Sci., 17:140–164, 1970.
2. Calvo T., Mayor G., and Mesiar R. (eds.). Aggregation Operators: New Trends and Applications, Physica-Verlag, Heidelberg 2002.
3. Carlsson C. and Fuller R. Fuzzy Reasoning in Decision Making and Optimization. Springer, Berlin, 2002.
4. Chen S.J. and Hwang C.L. Fuzzy multiple attribute decision-making, methods and applications. Lecture Notes in Economics and Mathematical Systems, Springer, Heildelberg, 1993.
5. Cheng C.H. and Mon D.-L. Evaluating weapon system by analitical hierarchy process based on fuzzy scales. Fuzzy Sets Syst., 63:1–10, 1994.
6. Fodor J. and Roubens M. Fuzzy Preference Modelling and Multicriteria Decision Support. Kluwer Academic, Dordrecht, 1994.
7. Grabisch M., Nguyen H., and Walker E. Fundamentals of Uncertainty Calculi, with Applications to Fuzzy Inference. Kluwer Academic, Dordrecht, 1995.
8. Kacprzyk J. and Yager R.R. Management Decision Support Systems Using Fuzzy Sets and Possibility Theory. Springer, Berlin, 1985.
9. Novák V. Fuzzy Sets and Their Applications. Adam Hilger, Bristol, 1989.
10. Novák V. Soft computing methods in managerial decision making. In Proc. 7th Czech-Japanese Seminar on Data Analysis and Decision Making under Uncertainty, 2004, pp. 63–68.
11. Saaty T.J. Fundamentals of Decision Making and Priority Theory With the Analytic Hierarchy Process. RWS Publications, 2000.
12. Sakawa M. Fuzzy Sets and Interactive Multiobjective Optimization, Applied Information Technology. Plenum, New York, 1993.
13. Slowinski R. (ed.). Fuzzy Sets in Decision Analysis, Operations Research and Statistics. Handbook of Fuzzy Sets Series. Kluwer Academic, Dordrecht, 1998.
14. Zimmermann H.-J. Fuzzy Set Theory and Its Applications. Dordrecht, Boston, 1985.
15. Zopounidis C., Pardalos P.M., and Baourakis G. Fuzzy Sets in Management, Economics and Marketing. World Scientific, 2001.

Fuzzy Time

▶ Temporal Indeterminacy

Fuzzy/Linguistic IF-THEN Rules and Linguistic Descriptions

VILÉM NOVÁK
University of Ostrava, Ostrava, Czech, Republic

Definition

Fuzzy/linguistic IF-THEN rules are structured expressions of natural language having the form

$$\mathcal{R} : \text{IF } X \text{ is } \mathcal{A} \text{ THEN } Y \text{ is } \mathcal{B} \tag{1}$$

where X, Y are variables and $\mathcal{A}, \mathcal{B}$ are expressions such as *small, very small, medium, roughly medium, more or less big, big*, etc. The latter are called *evaluative linguistic expressions*. Modeling their meaning in fuzzy set theory makes it possible to model the meaning of the whole rule. The part before THEN is called *antecedent*, the part after it is called *consequent*.

A *linguistic description* is a finite set of fuzzy/linguistic IF-THEN rules

$$\begin{array}{rl} \mathcal{R}_1 : & \text{IF } X \text{ is } \mathcal{A}_1 \text{ THEN } Y \text{ is } \mathcal{B}_1 \\ & \dotfill \\ \mathcal{R}_m : & \text{IF } X \text{ is } \mathcal{A}_m \text{ THEN } Y \text{ is } \mathcal{B}_m. \end{array} \tag{2}$$

Linguistic description can be taken as a special structured text in natural language which describes some situation.

Key Points

There are two possible ways how fuzzy/linguistic IF-THEN rules can be interpreted in fuzzy set theory:

(a) IF-THEN rule is assigned a fuzzy relation,

(b) IF-THEN rule is assigned a function from the set of contexts to the set of fuzzy relations.

Case (b) is more complicated but more realistic as a model of the meaning of linguistic expressions since (1) can be taken in this case as a conditional expression of natural language.

In case (a), the whole linguistic description (2) is assigned a fuzzy relation constructed using one of two possible formulas: let each of the expressions of the form “X is $\mathcal{A}_i$” be assigned a fuzzy set $A_i \underset{\sim}{\subset} U$ and “Y *is* $\mathcal{B}_i$,” $i = 1,\ldots m$ a fuzzy set $B_i \underset{\sim}{\subset} V$. Two possible fuzzy relations can be considered:

$$R^A(x, y) = \bigvee_{i=1}^{m} (A_i(x) \otimes B_i(y)), \qquad (3)$$

$$R^I(x, y) = \bigwedge_{i=1}^{m} (A_i(x) \rightarrow B_i(y)), \qquad (4)$$

where $\otimes$ is a t-norm and $\rightarrow$ is a residuation. Then (3) is called *disjunctive normal form* in which each IF-THEN rule is interpreted as *conjunction*; (4) is called *conjunctive normal form* in which each IF-THEN rule is interpreted as *implication*. Both forms (3) and (4) are two possible interpretations of the linguistic description (2).

The second interpretation of (2) is a set of functions $I(\mathcal{R}_i)$ assigned to the rules $\mathcal{R}_i$ in (2), $i = 1,\ldots,m$. Each function has the form

$$I(\mathcal{R}_i) : C_X \times C_Y \longrightarrow L^{U \times V} \qquad (5)$$

where C_X, C_Y are sets of *contexts* for the variable X and Y, respectively. Each context $w_X \in C_X$ and $w_Y \in C_Y$ is a certain interval of elements in U and V, respectively. The couple of contexts $\langle w_X, w_Y \rangle$ is assigned via (5) a fuzzy relation of the form $A_i(x) \rightarrow B_i(y)$, $x \in U$, $y \in V$. This approach leads to a mathematical model of the meaning of the text (2). Human understanding to such expressions and deriving conclusions on the basis of them can be mimicked. The details can be found in [2].

Cross-references

▶ Approximate Reasoning
▶ Fuzzy Relation
▶ Fuzzy Set

Recommended Reading

1. Klir G.J. and Yuan B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice-Hall, New York, 1995.
2. Novák V. and Lehmke S. Logical structure of fuzzy IF-THEN rules. Fuzzy Sets Syst., 157:2003–2029, 2006.
3. Novák V. and Perfilieva I. On the semantics of perception-based fuzzy logic deduction. Int. J. Intell. Syst., 19:1007–1031, 2004.
4. Novák V., Perfilieva I., and Močkoř J. Mathematical Principles of Fuzzy Logic. Kluwer, Boston/Dordrecht, 1999.

G

Gaifman-Locality

► Locality of Queries

Gazetteers

Linda L. Hill
University of California-Santa Barbara, Santa Barbara, CA, USA

Synonyms

Place names; Toponyms; Knowledge organization systems; Ontologies

Definition

A simple definition is that gazetteers are dictionaries of placenames. The digital gazetteer as a component of georeferenced information systems, however, is more formally modeled. A gazetteer is defined as a collection of gazetteer entries, each of which contains, at a minimum, the tuple *N, F, T* where *N* is a place name, *F* is a formal expression of geographic location – a *footprint*, and *T* is a place type expressed with a term (or code) from a typing scheme. Applications often require, in addition, relationships between gazetteer entries, documentation of time frames, and additional information (as described below). The gazetteer model is a type of knowledge organization system (KOS) – or ontology – which can be modified to represent other classes of spatial-temporal information, such as named time periods and named events [3].

Key Points

Gazetteers support bidirectional translation between *informal* georeferencing using names (e.g., Las Vegas) and *formal* georeferencing using coordinates (e.g., 36°10′30″N, 115°08′11″W) or similar mathematical reference within a geospatial framework. The need for such translation is common, For example, to calculate routes and distances and to support information retrieval by either place names or coordinates. A common use for gazetteers is to answer the "where is" question: "where is Baltimore?" can be answered by a map display showing the locations of places called "Baltimore" in the world.

In addition to place names and geospatial location, a third basic component of a gazetteer entry is a classification according to a typing scheme; that is, assigning a type term to a place to indicate that it is a populated place, a river, an island, a country, a bank, etc. There is no single typing scheme for named places. Instead, there are multiple typing schemes, usually unique for a particular application.

All of the descriptive data associated with named places have time dimensions. To deal with these temporal dimensions adequately, a gazetteer data structure must incorporate time ranges for places as well as for most of the elements of description.

Since different names can be used for the same place (e.g., in multiple languages), gazetteers must support multiple names and, ideally, document the source and time frame of each name. Supporting multiple footprints and information about a place from multiple sources is also needed for some applications.

A gazetteer is a collection of gazetteer entries [5]. Inherent spatial relationships exist between gazetteer entries. It can be determined which places are located within the footprint of another place – at least, theoretically. Actually, the ability to do this adequately depends on the quality of the footprints. Explicit relationships between gazetteer entries can also be added to the descriptive information. It can be documented, for example, that Chicago *is part of* Illinois and that Illinois *is part of* the US or that Topeka *is capital of* Kansas.

The International Organization for Standardization (ISO), the Open Geospatial Consortium, and the Alexandria Digital Library at the University of

California, Santa Barbara, have published gazetteer standards, protocols, and place typing schemes; see references [2,4–6].

Cross-references

- ▶ Browsing in Digital Libraries
- ▶ Georeferencing
- ▶ GIS
- ▶ Searching Digital Libraries

Recommended Reading

1. Alexandria Digital Library. Guide to the ADL Gazetteer Content Standard. University of California, Santa Barbara, CA, 2004.
2. Hill L.L. Feature Type Thesaurus. Alexandria Digital Library, 2002.
3. Hill L.L. Georeferencing: The Geographic Associations of Information. MIT Press, Cambridge, MA, 2006.
4. International Organization for Standardization (ISO) – Technical Committee 211. Geographic Information – Spatial Referencing by Geographic Identifiers (ISO 19112:2003), 2003.
5. Janée G. and Hill L.L. ADL Gazetteer Protocol, Version 1.2. Alexandria Digital Library, University of California, Santa Barbara, CA, 2003.
6. Open Geospatial Consortium Inc. Gazetteer Service Profile of the Web Feature Service Implementation Specification. OGC 05–035rl, version 0.9.1, 2006.

Gene Expression Arrays

Mehmet M. Dalkiliç
Indiana University, Bloomington, IN, USA

Definition

A gene expression array (or DNA microarray) is a miniature, massively parallel, and semi-automated process that measures levels of gene products (or mRNA or transcripts). The DNA of an organism (cell or cells) is a set of instructions used to direct its existence. A genetic message is transcribed from DNA into mRNA (messenger RNA), then typically translated into protein. By measuring the levels of mRNA, an indirect glimpse can be taken into the functioning of a cell. The relative and absolute levels of the transcripts are called expressions, and this instantaneous set of expressions is called the expression profile. Profiles can be used, for example, to indicate a disease state.

Historical Background

DNA and RNA are bio-molecules that *hybridize* bonding due to complementary structure. Hybridization has been used for several decades as means of identifying DNA and RNA present in an organism (cell or cells). Until microarray technology was invented, utilizing hybridization was slow and could only be perfomed on a relatively small scale. The microarry process is often associated with the Pat Brown laboratory in 1995 [6] and by Affemetrix [5].

Foundations

The fundamental purpose of gene expression arrays (or DNA microarrays) is to understand how an organism functions through measuring levels of proteins – molecules that perform a vast array of functions necessary for the life of the organism. Together with its genome (the DNA of an organism), DNA microarrays provide a global perspective into how regulation takes place, in other words, how the organism makes use of its DNA. The measurements are not directly made on the protein levels themselves, but intermediaries between the DNA and protein called RNA. Specifically, messenger RNA (mRNA) is first transcribed from the DNA, then translated into protein. The basic mechanism underlying microarray technology is *hybridization* – the chemical pairing of complementary molecules. DNA can hybridize to itself and to RNA, and this property has been used for decades in Southern and Northern blotting that measures amounts of hybridized DNA and RNA, respectively, typically tagged through radioactive labeling. The weakness of these two techniques is scale and time: only a small number of proteins can be measured requiring the period of a day or so. These blotting techniques have been miniaturized, made massively parallel, and semi-automated, so that $O(10^5)$ to $O(10^6)$ experiments can be conducted at a time. In DNA microarrays single stranded fragments of DNA are affixed to a substrate (called probes) in a grid and washed with a solution containing fluorescently tagged fragments of DNA (called targets). The concentration of DNA labeled fragments affects the likelihood of hybridizing to the probes. The intensities of each point on the grid are measured and result in a table of floating point numbers. It is from the intensity of labels that the levels of proteins are to be inferred. There are essentially two kinds of DNA microarrays: cDNA (spotted arrays) and high-density oligonucleotide arrays (oligos). In the former, fragments of a genome are used as probes. Oligos are synthesized sequences of nucleotides. cDNA arrays allow for matching against the genome, and Oligos for particular sequences of interest. Because the company Affemetrix developed the oligos, these are often referred to as Affy chips (or Affy arrays). The data produced from arrays is a very large set of intensities of electromagnetic radiation. Each intensity is associated with a two-dimensional location on the chip is called a spot. Microarrays are noisy, low resolution, and inconsistent; however, as the technology matures, there have been marked improvements. The data must be heavily pre-processed before analysis. This includes correcting for experimental design, biological factors, and specific mechanical idiosyncrasies themselves. One of the most significant pre-processing steps is called "normalization" which is an attempt to make different microarray data comparable. Currently, a consensus on how to achieve normalization does not exist, and a number of approaches can be taken. Extensive libraries exist in **R** (http://www.r-project.org) a feely available computing environment for statistical analysis; in particular, the OLIN [2] and Bioconductor packages [4] include normalization and dye correction. The main objective in microarray anaylsis is discovery patterns and outliers. Patterns are, crudely, sets of gene products that possess some discernable functional relationship. Typically, the relationship is between two expressions and linearity is examined. In particular, Pearson's correlation is often used to gauge the linearity of genes. Pearson's correction denoted $\rho(\cdot,\cdot)$ is a real-valued measurement on the interval $[-1,1]$ over two sets of data, X,Y. The score $|\rho(X, Y)| \approx 1$ if there is either a positive or negative unity if a strong linear relationship exists, tending toward zero otherwise. A zero value for Pearson's only indicates that linearity does not exist. Database practitioners should realize the microarray is significant when it can be thoroughly studied with extensive biological annotations – it is much more than a table of spots.

Key Applications

There are key applications for both bioinformatics and database. To give some indication of the amount of interest in the area, the number of publications found in PubMed (http://www.pubmed.gov) by searching with terms "`microarray`" and "`2007`" yields well over 10,000 entries (executed 01/20/2008). For bioinformatics, microarrays promise to yield cell states that can help determine function, disease states (as mentioned previously), guide areas of discovery (e.g., drugs). From a database perspective, there is an incredible wealth of data that needs to be managed. However, the empirical and complex nature of the processes underlying the microarray belie its apparent simplicity – usually rendered pictorially as an attractive rectangle grid of colored circular sports that range from green to yellow to red. Although no ANSI standard exists for microarrays, like much of bioinformatics there are different popular standards vying for dominance. Of these, the most well-known is MIAME (the minimum information about a microarray experiment) that specifies elements of experiments. Since this specification is at least recognized in name, it will briefly highlight its components: (i) Array information including platform (spotted, etc.), size and number of spots, protocols; (ii) Reporter information (sequences on the slide itself); (iii) Gene information; (iv) Information about control array control elements. Often cluster analysis is done on microarrays, rows are gene products and columns are experiment conditions. The clustering results in a tree (dendogram) that depicts the most similar genes on the set of conditions. A formal language for accessing, managing, creating these large data sets does not exist.

Queries associated with arrays vary with the intent of the user. This is significant because a good number of these queries rely on data completely external to the system. For example, a typical query might be, "What are the sequence annotations of gene g_1, g_2, g_3?" For this query, the system must link to sequence annotation databases to retrieve the information. The challenge is that there are (again) no official standards and that the data, contained in flat files or even XML, must be parsed to extract the meaningful information desired. To answer the growing needs to understand a group was formed near 2000 called the Microarray and Gene Expression Database (http://www.mged.org). The Microarray and Gene Expression Data (MGED) Society is a global consortium of scientists who are striving to provide some coherence in how to make data exchange easier. The scope of MGED includes other genome-wide data and so, provides additional information particular to data like binding. Although the MGED is working to establish standards for management and so forth, it is unclear how widespread the recognition and adoption is. As mentioned with the example query, one unusual aspect of managing gene expression data is associating it with other data housed and managed in other systems. The MGED provides a few standards: MIAME, MAGE-TAB, MAGE-ML, MGED, Ontology; it also provides information about the current status of these representations.

One of the most significant elements of database design for array databases is the ability to exchange and retrieve "foreign" data – data housed in different forms in different repositories likely due to the nature of the biological questions being asked. To this end, MAGE and MAGE-ML, are means of facilitating data exchange, the latter XML in spirit. MAGE is more popular and does associate with the MIAME standard, but the associations are not one-to-one. One significant difference between MIAME and MAGE is the latter provides elements of provenance and security, while MIAME has no mechanism at all.

ArrayExpress, a public European repository for array data, is housed at the European Bioinformatics Institute (EBI) and seems to be aiming more toward a data warehousing approach, archiving data, but allowing limited keyword searches, etc. accession number or author.

A significant challenge to management of array data is the dynamic nature of information. Since array data maps back to genomes, as genome drafts are updated, array data must reflect these changes. There is genomic draft versioning to help remap gene products to the updated genomic locations. To give an idea of the complexity, often drafts result in genes being split, coalesced, removed, or changed in length. This directly affects the interpretation of the results of array.

Semantics of terms in array database are, as described above, often imprecise. One approach that biologists have begun adopting is the use of ontologies. The most well known example in biology is the Gene Ontology [3] (GO) (*http://www.geneontology.org/*). By fixing a common vocabulary and set of relationships GO has become a common tool in analysis.

Future Directions

Microarrays are only one of many so-called high-throughput technologies that produce enormous amounts of data – these can easily range in size to terabytes. Many problems biologists have been able to effectively solve on small scales, do not scale with the data. There is much work to be done in how to analyze such a large and growing corpus of data. Another aspect is how to integrate the different types of high-throughput. Systems biology is a recent discipline that attempts to semantically merge disparate types of data to form global perspectives that no one piece of data would contain. One of the standard structures used in microarrays is the directed graph. Again, polynomial complexity algorithms become infeasible with such a large amount of data. Bayesian Belief Nets have recently become popular in analyzing these large graphs (or networks).

Data Sets

There are public data sets available through the internet. Some of the better known include:

NAME	URL
ArrayExpress	http://www.ebi.ac.uk/microarray-as/aer/
GEO	http://www.ncbi.nlm.nih.gov/geo/
Stanford Microarray Database	http://genome-www5.stanford.edu/

Cross-references

- ▶ Annotation
- ▶ Approximate XML Querying
- ▶ Biological Networks
- ▶ Biological Sequence
- ▶ Data Mining
- ▶ Data Provenance
- ▶ Data Structures and Models for Biological Data Management
- ▶ Data Visualization
- ▶ Graph Database
- ▶ Graph Management in the Life Sciences
- ▶ Information Integration
- ▶ Ontologies
- ▶ Ontologies and Life Science Data Management
- ▶ Ontologies in Scientific Data Integration
- ▶ Ontology
- ▶ Semantic Data Integration for Life Science Entities
- ▶ Text Mining of Biological Resources
- ▶ Uncertainty and Data Quality Management
- ▶ Visual Data Mining
- ▶ XML

Recommended Reading

1. Drăghici S. Data Analysis Tools for DNA Microarrays. Chapman & Hall/CRC, London, 2003.

2. Futschik M.E. and Crompton T. OLIN: optimized normalization, visualization and quality testing of two-channel microarray data. Bioinformatics, 21(8):1724–1726, 2005.
3. Gene Ontology Consortium. Creating the gene ontology resource: Design and implementation. Genome Res., 11:1425–1433, 2001.
4. Gentleman R., Carey V., Bates D., Bolstad B., Dettling M., Dudoit S., Gautier L., Ge Y., Gentry J., Hornik K., Hothorn T., Huber W., Iacus S., Irizarry R., Leisch F., Li C., Maechler M., Rossini A., Sawitzki G., Smith C., Smyth G., Tierney L., Yang J., and Zhang J. Bioconductor: open software development for computational biology and bioinformatics. Genome Biol., 5(10):R80,2004.
5. Lockhart D.J., Dong H., Byrne M.C., Follettie M.T., Gallo M.V., Chee M.S., Mittmann M., Wang C., Kobayashi M., Horton H., and Brown E.L. Expression monitoring by hybridization to high-density oligonucleotide arrays. Nat. Biotechnol., 14:1675–1680, 1996.
6. Schena M., Shalon D., Davis R.W., and Brown P.O. Quantitative monitoring of gene expression patterns with complementary DNA microarray. Science, 270:467–460, 1995.

Generalisation

► Abstraction

Generalization of ACID Properties

Brahim Medjahed[1], Mourad Ouzzani[2],
Ahmed K. Elmagarmid[2]
[1]The University of Michigan–Dearborn, Dearborn, MI, USA
[2]Purdue University, West Lafayette, IN, USA

Synonyms

Advanced transaction models; Extended transaction models

Definition

ACID (Atomicity, Consistency, Isolation, and Durability) is a set of properties that guarantee the reliability of database transactions [2]. ACID properties were initially developed with traditional, business-oriented applications (e.g., banking) in mind. Hence, they do not fully support the functional and performance requirements of advanced database applications such as computer-aided design, computer-aided manufacturing, office automation, network management, multidatabases, and mobile databases. For instance, transactions in computer-aided design applications are generally of long duration and preserving the traditional ACID properties in such transactions would require locking resources for long periods of time. This has lead to the generalization of ACID properties as Recovery, Consistency, Visibility, and Permanence. The aim of such generalization is to relax some of the constraints and restrictions imposed by the ACID properties. For example, visibility relaxes the isolation property by enabling the sharing of partial results and therefore promoting cooperation among concurrent transactions. Hence, the more generalized ACID properties are, the more flexible the corresponding transaction model will be.

Key Points

ACID is an important concept of database theory. It defines four properties that traditional database transactions must display: Atomicity, Consistency, Isolation, and Durability. *Atomicity* states that transactions must follow an "all or nothing" rule. Either all of the changes made by a transaction occur, or none of them do. The two-phase commit protocol (2PC) is generally used in distributed databases to ensure that each participant in a transaction agrees on whether or not the transaction should be committed. *Consistency* means that transactions always operate on a consistent view of the database and leave the database in a consistent state. A database is said to be consistent as long as it conforms to a set of invariants, called *integrity constraints*. *Isolation* gives the illusion that each transaction is executed alone. It ensures that the effects of a transaction are invisible to other concurrent transactions until that transaction is committed. Concurrency control protocol are generally implemented in database systems to preserve this property. *Durability* states that once a transaction is committed, its effects are guaranteed to persist even in the event of subsequent failures. This is usually achieved using database backups and transaction logs.

The limitations inherent to the original ACID properties and the peculiarities of advanced database applications has lead to the generalization of ACID properties Recovery, Consistency, Visibility and Permanence. *Recovery* refers to the ability to take the database to a state that is considered correct in case of failure. *Consistency* refers to the correctness of the state of the database that a committed transaction produces. *Visibility* refers to the ability of one transaction to see the results of another running transaction.

Permanence refers to the ability of a transaction to record its results in the database. The flexibility of a transaction model depends on the way ACID properties are generalized. An extensive coverage of advanced transaction models is presented in [1]. *Sagas*, *Nested Transactions*, and *Flex* are representative examples of models that generalize ACID properties while *ACTA* is an example of a formal framework to express these models and reason about them.

A *Saga* is a chain of transactions that is itself atomic. Isolation is relaxed at the level of a Saga and visibility is permitted at the component transaction boundaries. A saga itself is atomic but because of the relaxed visibility, it supports semantic consistency. That is, each transaction in the chain is assumed to have a semantic inverse, or compensation, transaction associated with it. If one of the transactions in the saga fails, the transactions are rolled back in the reverse order of their execution. Committed transactions are rolled back by executing their corresponding compensation transactions.

Nesting allows concurrency within a transaction and provides fine-grained and hierarchical control for failure handling. The original nested transaction model, which supports only closed sub-transactions, was extended to include open sub-transactions. Closed sub-transactions may not support consistency and durability. A closed sub-transaction commits its results to its parent. These partial results are externalized only after the top (root) transaction commits, thus ensuring atomicity and isolation of the whole transaction. Because of its relaxed visibility, open sub-transactions directly externalize their results and expose them to other transactions.

The *Flex Transaction Model* has been proposed to generalize ACID properties in multidatabase systems. It relaxes the atomicity and isolation properties of nested transactions to provide users increased flexibility in specifying their transactions. A Flex transaction may proceed and commit even if some of its sub-transactions fail. It also allows the specification of dependencies on sub-transactions as internal or external dependencies. Internal dependencies define the execution order of sub-transactions, while external dependencies define the dependency of a sub-transaction execution on events (such as the start/end execution times) that do not belong to the transaction. The Flex model also enables users to control the isolation granularity of a transaction through the use of compensating sub-transactions.

ACTA is a framework that facilitates the formal description of properties of extended transaction models. It defines constructs that facilitate the synthesis of extended transaction models by tailoring/combining existing models or starting from first principles. Different notions are introduced in ACTA to enable the generalization of ACID properties from different perspectives. For instance, the notion of delegation allows transactions to selectively abort some of the operations it has performed and yet commit.

Cross-references

▶ ACID Properties
▶ Concurrency Control
▶ Distributed
▶ Extended Transaction Models
▶ Extended Transaction Models and the ACTA Framework
▶ Parallel and Networked Databases
▶ Serializability
▶ System Recovery
▶ Transaction Management
▶ Two-Phase Commit

Recommended Reading

1. Elmagarmid A.K. (ed.). Database Transaction Models for Advanced Applications. Morgan Kaufmann, Los Altos, CA, 1992.
2. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, Los Altos, CA, 1993.

Generalized Search Tree

Joseph M. Hellerstein
University of California-Berkeley, Berkeley, CA, USA

Synonyms

GiST; GIST

Definition

The Generalized Search Tree (GiST) is an extensible, disk-based index structure for large data sets, enabling the easy design and implementation of domain-specific index structures. The GiST provides a general-purpose implementation of many of the difficult systems issues inherent in indexing (data storage and access, search, concurrency and recovery), with a compact extensibility interface sufficient for the specification of the domain-specific, algorithmic aspects of indexing (clustering of data into pages, labeling of subtrees, and prioritization of the search frontier).

Historical Background

A key research challenge in database systems in the 1980s and 1990s was to support an *extensible* set of abstract data types, beyond the alphanumeric types typically used in business data processing. One critical component of database extensibility is the ability to easily add new *access methods* customized to specific data types and query operators. Ideally, extensibility interfaces should enable the programmer extending the DBMS to focus on their domain expertise (e.g., geographic information, bioinformatics, etc.), and not have to reason about database systems issues like storage, buffering, query optimization, concurrency or recovery.

The Postgres system provided one of the first truly extensible access method interfaces, which allows new access methods to be registered with a query planner and executor. However, this interface leaves the implementation of data storage, concurrency and recovery in the hands of the extension programmer, which makes the task of writing a new access method extremely difficult, and results in inelegant software that replicates complex functionality across different index implementations.

The GiST design was intended to take care of as many generic database systems issues as possible, while still enabling a wide range of indexes to be easily written within its framework. At heart, the GiST is a generalization of the B+-tree, and one of its contributions is to demonstrate the degree to which many subsequent index inventions are specific variations of the basic B+-tree ideas.

In principle, the GiST can be used for any indexing *workload*, as defined by a family of query predicates over a particular data domain. This raises theoretical questions of both lower and upper bounds: which workloads are efficiently indexable, and how close to optimal is the performance of a particular index design on a given workload? The first of these questions inspired the *Theory of Indexability*; the latter led to a set of empirical performance metrics, which have been embodied in the *Access Method DeBugger (amdb)* toolkit.

Foundations

Like the B+-tree, the GiST is a height-balanced tree of variable fan out, with data stored at the leaves, and *search keys* associated with pointers in the internal nodes. The GiST search keys are the center of its extensibility API: a new index is implemented via a new search key class and associated methods. Search keys are parameterized by four user-defined methods:

1. The `consistent` method of a search key *k* takes a query predicate *p*, and returns *false* if it is not possible for *k* and *p* to describe the same data items. The index search algorithm only traverses a subtree if the consistent method returns true for the search keys above it.
2. The `union` method takes a set of search keys from a child node, and produces a single search key to be placed in the parent. This is used to update search keys whenever the contents of the tree change.
3. The `penalty` method of a search key takes a new data item to be inserted, and returns a cost for the insertion of that item into the subtree labeled by that search key. This guides the index insertion algorithm's descent to a leaf, affecting the clustering of data in the tree.
4. The `pickSplit` method takes a set of search keys, and partitions them into two subsets; this is used for choosing how to split an overflowing page during insertion into the tree. This method also affects the clustering of data in the tree.

In its original conception, the GiST was designed to support selection predicates, via search keys forming a set-containment hierarchy. Subsequently, it was observed that the GiST can naturally support richer queries, including near-neighbor search, as well as statistical methods for approximate query answering that treat the search keys as a multi-resolution synopsis of the data.

Key Applications

GiST has been implemented in a number of database systems and projects, most notably in the Informix Universal Server, and the open-source PostgreSQL system, both of which implement the full GiST concurrency and recovery algorithms.

Currently, GiST serves as the default framework for spatial and text indexing in PostgreSQL, via extensions corresponding to R-trees for spatial data, and Russian-Doll (RD) Trees for text data. The open-source

PostGIS Geographic Information System relies upon the PostgreSQL GiST implementation for its spatial indexing.

In addition to their widespread use for spatial and text workloads, GiST-based indexes have been used for applications in image retrieval, astronomy data, and genomic data.

URL to Code

1. http://www.postgresql.org
2. http://gist.cs.berkeley.edu

Cross-references

- ▶ B-Tree
- ▶ B+-Tree
- ▶ Rtree
- ▶ M-Tree
- ▶ Extensional Relational Database (ERDB)
- ▶ Postgres
- ▶ PostgreSQL
- ▶ Geographic Information System
- ▶ Spatial Indexing Techniques
- ▶ Informix Indexability Theory
- ▶ Index Concurrency
- ▶ Index Recovery

Recommended Reading

1. Aoki P.M. Generalizing 'search' in generalized search trees. In Proc. 14th Int. Conf. on Data Engineering, 1998.
2. Aoki P.M. How to avoid building dataBlades that know the value of everything and the cost of nothing. In Proc. 11th Int. Conf. on Scientific and Statistical Database Management, 1999, pp. 122–133.
3. Aref W.G., Ilyas I.F. SP-GiST: an extensible database index for supporting space partitioning trees. J. Intell. Inf. Syst., 17(2/3):215–240, 2001.
4. Hellerstein J.M., Koutsoupias E., Miranker D., Papadimitriou C., and Samoladas V. On a model of indexability and its bounds for range queries. J. ACM, 49(1):35–55, 2002.
5. Hellerstein J.M., Naughton J.F., and Pfeffer A. Generalized search trees for database systems. In Proc. 21st Int. Conf. on Very Large Data Bases, 1995, pp. 562–573.
6. Hellerstein J.M. and Pfeffer A. The RD-Tree: An Index Structure for Sets. University of Wisconsin Technical Report #1252, October 1994.
7. Kornacker M. High-performance generalized search trees. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999.
8. Kornacker M., Mohan C., and Hellerstein J.M. Concurrency and recovery in generalized search trees. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 62–72.
9. Kornacker M., Shah M.A., and Hellerstein J.M. Amdb: a design tool for access methods. Data Eng. Bull., 26(2):3–11, June 2003.
10. Thomas M.C. and Hellerstein J.M. Boolean bounding predicates for spatial access methods. In Proc. Int. Conf. on Database and Expert Systems Applications, 2002, pp. 925–934.
11. Stonebraker M. Inclusion of new types in relational data base systems. In Proc. 2nd Int. Conf. on Data Engineering, 1986.

Generative Models

▶ Language Models

Genetic Algorithms

Colin R. Reeves
Coventry University, Coventry, UK

Synonyms

Evolutionary algorithms; Evolutionary computation

Definition

A genetic algorithm (GA) is one of a number of heuristic techniques that attempt to find high-quality solutions to large and complex optimization problems. The term *evolutionary algorithm* is sometimes used synonymously, but is generally used to denote a rather wider class of heuristics. All such algorithms use the notion of a sequence of cycles that employ mutation of, and subsequent selection from, a population of candidate solutions. While these features are also found in a GA, its most distinctive characteristic is the use of *recombination* (or *crossover*) to generate new candidate solutions. A secondary idea found in many, but not all GAs, is the existence of an encoding function that maps the original optimization problem into a space that is hoped to be more congenial to the application of the GA operators.

Historical Background

The term *genetic algorithm* was first used by John Holland, whose book [8], first published in 1975, was instrumental in creating what is now a flourishing field of research and application that goes much wider than the original GA. The area covers topics such as evolution strategies (ES), evolutionary programming (EP), artificial life (AL), classifier systems (CS), genetic programming (GP), and now the concept of evolvable hardware. These related fields of research are often grouped under the heading of *evolutionary computation* or *evolutionary algorithms* (EAs).

While Holland's influence in the development of the topic has been immense, from an historical perspective it is clear that several other scientists with different backgrounds were also involved in proposing and implementing similar ideas. In the 1960s in Germany, Rechenberg and Schwefel developed the idea of the *Evolutionsstrategie* (in English, *evolution strategy*), while – also in the 1960s – Lawrence Fogel and others in the USA implemented a similar idea that they called *evolutionary programming*. What these proposals had in common was their use of the ideas of mutation and selection that lie at the center of the neo-Darwinian theory of evolution. Relatively little attention was given to the use of recombination – the idea later placed at the heart of GAs by Holland. However, although some promising results were obtained, as an area of research the topic of evolutionary computing did not really catch fire. There may be many reasons for this, but the most obvious one is that the techniques tended to be computationally expensive – unless the applications were trivial – and far beyond the capabilities of the computers of the day. Nevertheless, the work of these early explorers is fascinating to read in the light of our current knowledge. David Fogel (son of one of these pioneers) has tracked down and collected some of this work [6].

The last 30 years has seen an explosion of interest in GAs and their applications, and there is now enormous literature on variations of the basic algorithm and developments in application areas. De Jong [4] – another early pioneer, particularly in relation to the GA's utility as an optimizer – has recently surveyed the most important developments from a unified perspective.

Theoretical analysis of GAs, and of EAs in general, has lagged far behind the development of novel practical applications. Holland's original work gave an account based on his idea of a *schema* that raised hopes that the GA's power was significantly greater than that of other heuristics. Sadly, these hopes have not been realized, and the rise of 'No-Free-Lunch' arguments [13] has put paid to the idea of some universal optimization algorithm that outperforms all others. In the case of GAs, Holland's schema-based theory has proved much less relevant than early accounts suggested. Several other theoretical approaches have been tried, with some success in accounting for large-scale GA behaviour, but something truly comprehensive is still awaited. Reeves and Rowe [10] provide the most recent description of the state of GA theory.

Foundations

Optimization problems arise in many forms, but the most obvious dichotomy is between continuous and discrete (or combinatorial) optimization. While GAs have been used successfully for continuous problems, the evolution strategy approach – although employing similar principles – is on the whole better suited to continuous problems. What is usually in mind for the application of a GA is the following:

Given a discrete search space $\mathcal{V}$, and a function

$$g : \mathcal{V} \mapsto \mathbb{R},$$

find

$$\arg \min_{v \in \mathcal{V}} g.$$

From a mathematical perspective, the decision variables of this problem are encoded as strings or vectors where each component is a symbol from an alphabet $\mathcal{A}$. These strings are then concatenated to provide a *chromosome*. That is, the vector v is represented by a string x of symbols drawn from $\mathcal{A}$, using a mapping

$$c : \mathcal{A}^{\ell} \mapsto \mathcal{V}.$$

where ℓ, the length of the string, depends on the dimensions of both $\mathcal{V}$ and $\mathcal{A}$. The elements of the string correspond to "genes", and the values those genes can take to "alleles". The position of a gene in the string is its "locus". The encoding function c is often designated as the *genotype–phenotype mapping*. The space $\mathcal{A}^{\ell}$ – or, quite often, some subspace $\mathcal{X}$ of it – is known as the genotype space, while the space $\mathcal{V}$, representing the original definition of the problem, is the phenotype space.

Thus the optimization problem becomes one of finding

$$\min_{x \in \mathcal{X}} g(\boldsymbol{x}),$$

where (with a slight abuse of notation)

$$g(\boldsymbol{x}) = g(c(\boldsymbol{x})).$$

In practice the distinction between genotype and phenotype is often blurred, and $g(\boldsymbol{v})$ or $g(\boldsymbol{x})$ is used according to context. Finally, a monotonic transformation of g

is often used to provide what is usually termed a *fitness function*, which can be regarded as

$$f : \mathcal{X} \mapsto \mathbb{R}^{+},$$

so that fitness is always a positive value. It should also be noted that the genotype search space $\mathcal{X} \subseteq \mathcal{A}^{\ell}$, which implies that some strings may represent invalid solutions to the original problem. This can be a source of difficulty for GAs, since GA operators may generate new strings in $\mathcal{A}^{\ell}$ that are not in $\mathcal{X}$. The choice of encoding is usually dictated by the type of problem that is being investigated. Examples and further details on such matters can be found in standard references such as [4] or [10].

With this background in mind, a basic version of a GA can now be presented as in Fig. 1.

Initial populations are usually generated randomly, although there are many variations that incorporate some aspects of problem-specific knowledge. Given a population, the next step is to choose parent chromosomes for reproduction. This is generally carried out by some means of *fitness-proportional* selection. Implementations of this idea vary widely: roulette-wheel, rank-based and tournament selection schemes are all popular and have associated advantages and disadvantages.

Reproduction takes place using crossover and mutation. Having selected parents in some way, they must be recombined – the process called *crossover*. This is simply a matter of replacing some of the alleles in one parent by alleles of the corresponding genes of the other. Suppose there are two strings ***a*** and ***b***, consisting of seven symbols, i.e.,

$$(a_1, a_2, a_3, a_4, a_5, a_6, a_7) \quad \text{and} \quad (b_1, b_2, b_3, b_4, b_5, b_6, b_7),$$

each representing a possible solution to a problem. To implement *one-point* crossover a *crossover point* is chosen at random from the numbers 1,...,6, and a new solution produced by combining the pieces of the original "parents". For instance, if the crossover point was 2, then the "offspring" solutions would be

$$(a_1, a_2, b_3, b_4, b_5, b_6, b_7) \quad \text{and} \quad (b_1, b_2, a_3, a_4, a_5, a_6, a_7).$$

Many other forms of crossover have been invented, with varying degrees of success.

The concept of mutation is even simpler than crossover: a gene (or subset of genes) is chosen randomly and the allele value of the chosen genes is changed. This can easily be represented as a bit-string mask such as

```
0100010
```

generated using a Bernoulli distribution – i.e., at each locus there is a small probability μ of a mutation occurring (indicated in the mask by a `1`). The above example would thus imply that the second and sixth genes are assigned new allele values.

Finally, a new population has to be selected from the joint set of old and newly generated members. Again there are many ways of carrying this out. At the extremes, there are the generational and steady-state approaches. In the first of these, the old population is completely replaced by the new. In the second, a single new candidate is generated, and some member of the old population (often the least fit one) is displaced by it. Again, many other techniques have been tried.

This is an extremely condensed overview of the possibilities available for implementing a GA. A more comprehensive picture of some of the most popular ideas, and a discussion of their effects on GA performance, can be found in [4] or [10].

```
Choose an initial population of chromosomes;
while termination condition not satisfied do
    repeat
        if crossover condition satisfied then
        {select parent chromosomes;
        choose crossover parameters;
        perform crossover;}
        if mutation condition satisfied then
        {select chromosome(s) for mutation;
        choose mutation points;
        perform mutation;}
        evaluate fitness of offspring
    until suffcient offspring created;
    select new population;
endwhile
```

Genetic Algorithms. Figure 1. A genetic algorithm template. This is a fairly general formulation, accommodating many different forms of selection, crossover and mutation. It assumes user-specified conditions under which crossover and mutation are performed, a new population is created, and whereby the whole process is terminated.

Key Applications

The number of NP-hard combinatorial optimization problems continues to grow, and heuristic methods offer the only realistic route to high-quality solutions. The use of GAs is thus motivated by the existence of computational complexity in many aspects of technological development. In the area of database systems, there were early applications to problems of database design [3] and query optimization [11]. Other areas of interest include data clustering methods [7] and data partitioning [2,1]. Genetic algorithms have also been used for rule discovery in data mining applications, seminal work in this area being reported in [5]. Data reduction by means of instance selection using GAs for other techniques such as decision trees, neural nets or nearest neighbour methods has also been reported [9]. Also, several companies claim – without revealing very many details – to use GAs in spam detection software. A recent published example of such work is [12].

Experimental Results

Given the lack of authoritative theoretical models of GAs, assessment of their performance in any individual case is necessarily experimental. Standard methods of evaluation include the *best-so-far curve* – a time series plot of the fitness of the best solution found so far (y-axis) against the number of generations or the number of individuals generated. This is sometimes called the *offline performance* measure, in contrast to the *online performance* measure, which is a time series plot of current (average) population fitness.

Another aspect of experimental evaluation is assessment of convergence – or equivalently, of population diversity. This can be assessed simply by plotting a time series of the coefficient of variation of the current population fitnesses, or more expensively, by means of a locus-wise count of allele percentages in successive populations.

Since GAs are stochastic algorithms, it is obvious that results will differ from one run of the algorithm to the next. In any experimental work it is essential to apply the algorithm several times in order to have some confidence that the results obtained are not freakishly good (or bad).

Cross-references

- ▶ Data Clustering
- ▶ Database Design
- ▶ Data Mining
- ▶ Query Optimization
- ▶ Spam Detection

Recommended Reading

1. Acar A.C. and Motro A. Intensional encapsulations of database subsets via genetic programming. In database and expert systems applications, K.V. Andersen, J. Debenham and R. Wagner (eds.). Lecture notes in computer science 3588. Springer, Berlin, 2005, pp. 365–374.
2. Cheng C.H., Lee W.K., and Wong K.F. A genetic algorithm-based clustering approach for database partitioning. IEEE Trans. Syst. Man Cybernet., 32:215–230, 2002.
3. Corcoran A.L. and Hale J. A genetic algorithm for fragment allocation in a distributed database system. In Proc. 1994 ACM Symp. on Applied Computing, 1994, pp. 247–250.
4. De Jong K.A. Evolutionary Computation: A unified approach. MIT Press, Cambridge, MA, 2006.
5. Flockhart I.W. and Radcliffe N.J. A genetic algorithm-based approach to data mining. In Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, 1996, pp. 299–302.
6. Fogel D.B. Evolutionary Computation: The Fossil Record. IEEE Press, New York, 1998.
7. Hall L.O., Özyurt I.B., and Bezdek J.C. Clustering with a genetically optimized approach. IEEE Trans. Evol. Comput., 3:103–112, 1999.
8. Holland J.H. Adaptation in Natural and Artificial Systems, 2nd edn. MIT Press, Cambridge, MA, 1992.
9. Reeves C.R. and Bush D.R. Using genetic algorithms for training set selection in RBF networks. In Instance selection and construction for data mining. H. Liu and H. Motoda (eds.) Kluwer, Dordecht, 2001, pp. 339–356.
10. Reeves C.R. and Rowe J.E. Genetic Algorithms: Principles and Perspectives. Kluwer, Dordecht, 2002.
11. Wang J.C., Horng J.T., and Liu B.J. A genetic algorithm for set query optimization in distributed database systems. IEEE Conf. Syst. Man Cybernet., 3:1977–1982, 1996.
12. Wang H.B., Yu Y., and Liu Z. SVM classifier incorporating feature selection using GA for spam detection. In Proc. Int. Conf. embedded and ubiquitous computing, 2005, pp. 1147–1154.
13. Wolpert D.H. and Macready W.G. No free lunch theorems for optimization. IEEE Trans. Evol. Comput., 1:67–82, 1997.

Geographical Information Retrieval

Christopher B. Jones[1], Ross S. Purves[2]
[1]Cardiff University, Cardiff, UK
[2]University of Zurich, Zurich, Switzerland

Definition

The provision of facilities to retrieve and relevance rank documents or other resources from an unstructured or partially structured collection on the basis of queries specifying both theme and geographic scope.

Historical Background

Geographical information associates things and events with locations. Since everything one does and everything that happens on Earth happens somewhere, there is a vast amount of information that can be regarded as geographical in some sense. This information is represented within a variety of media, including digital and analogue versions of text documents, images, videos and maps, as well as the structured spatial databases that are employed within most geographical information systems (GIS). In text documents the association with location is often qualitative with place names and natural language terminology (such as in, at, and near) that provide spatial context. This is in contrast to the use in GIS of, for example, digital topographic datasets in which location is described quantitatively with numerical coordinates. Over the past few decades, information technology for accessing geographical information has focused on the latter, quantitative representation of geographic space that characterizes GIS. It is only in recent years that much attention has been paid to the development of computer systems to retrieve geographically-specific information from the relatively unstructured but immense resource of digital documents, many of which can be found on the worldwide web (e.g., [7,9,11]).

Because of the importance of geographical context, it is not surprising that requirements for information on the web often have a geographical focus (it has been estimated that between 13–15% of web queries submitted to traditional search engines contain a place name) [6,12]. It is this need for public access to geographical information on the web that has been a major motivation for the growth of the academic field of geographical information retrieval (GIR).

Much current research in GIR can be regarded as an extension of the field of information retrieval (IR). The products of work in IR have provided the foundation for current search engine technology, with its focus on access to primarily textual documents that are relevant to user queries taking the form of a phrase or set of key words. This latter type of query is in contrast to the more formal approach common in spatial databases, where georeferenced data objects are retrieved from structured tables in response to queries that can stipulate precise spatial constraints. IR methods usually identify documents that contain query terms and rank the retrieved documents using statistical methods that are intended to highlight the most relevant to the user's needs. For some forms of geographical search, particularly when looking for common resources within relatively large geographic extents (e.g., hotels in London), this approach can work well but it is fraught with limitations.

From the user's perspective these limitations can be manifested in a failure to distinguish between different instances of the same place name, a lack of ability to deal with spatial qualifiers such as near or north, a lack of methods to rank and explore results with respect to their spatial relevance and the non-retrieval of resources which are spatially relevant but use a place name different from that specified in the query. GIR is therefore concerned with improving the quality of geographically-specific information retrieval with a focus on access to unstructured documents such as those found on the web. There are several types of functionality that are distinctive to GIR and which constitute significant challenges. These include:

- Detecting geographical references in the form of place names and associated spatial natural language qualifiers within text documents and in user queries.
- Disambiguating place names to determine which particular instance of a name is intended.
- Geometric interpretation of the meaning of vague place names, such as the "Midlands," and of vague spatial language such as "near".
- Indexing documents with respect to their geographic context as well as their non-spatial thematic content.
- Ranking the relevance of documents with respect to geography as well as theme.
- Developing effective user interfaces that help users to formulate effective queries and identify relevant results.
- Developing methods to evaluate the success of GIR.

The following sections elaborate upon these issues.

Foundations

Detecting Geographic References

Place names (or toponyms) can be used to refer to places on Earth, but they also occur frequently within the names of organizations and as part of people's names. It is also the case that place names may be used *metonymically*, for example to refer to administrative entities as in "talks with Washington." The process of geo-parsing is concerned with analyzing text to identify the presence of place names and other spatial language and distinguish the genuine geographical occurrences of place name usage from those where they are being used to refer to some other entity. This process is often treated as an extension of Named Entity Recognition that is a standard part of linguistic analysis in natural language processing (NLP). Various techniques can be employed to identify a place name, including the use of rule-based approaches and machine learning. In order to recognize that a word is a potential place name, it is common practice to employ a list of known place names as is found in a gazetteer. Within gazetteers, each name is usually accompanied by at least one parent region, a classification for the place and a coordinate-based footprint providing a quantitative location.

Disambiguating Place Names

Once it has been established that a place name is being used in a geographic sense, the problem remains of uniquely determining the place to which the name refers. Note that the combination of detecting place names and disambiguating them is referred to as geo-parsing. There are many names that are shared between different places. For example there are over 70 "Springfields" in the US. When humans read a document with a place name in it, they will tend to resolve ambiguity using knowledge gained from contextual clues within the document. Automatic resolution of geographic scope, and hence disambiguation of place names, attempts to mimic the methods humans use, for example by considering together all of the place names in the document. If a place name occurs in association with a set of other place names, several of which refer to neighboring places or are instances of places within the same parent region, then that provides evidence to distinguish which meaning is implied. Equally if the text mentions a parent or child region of an instance of a particular name, then that can help to determine the sense that is intended [14].

Vague Geographic Terminology

Many of the place names that users employ when searching on the web are of an informal or vernacular nature, often without precise boundaries. Examples include the Pennines and the Borders in the UK, the south of France, the Swiss Mittelland and the midwest in the US. Existing geographical search facilities make use of gazetteers that are based largely on the administrative names of places and which do not, in general, include any representations of vernacular names. There have been a few studies of methods of determining what may be a fuzzy extent for such places (e.g., [8]) but there is still much to be done. Recently, web mining methods have been used to determine associations between a vernacular place name and the names of other places within it or in the vicinity [4]. Spatial extent can then be modeled using, for example density surfaces, or Delaunay triangulation based methods [2].

The spatial language (such as near, close, between and north of) that accompanies place name terminology can be as vague as some of the place names. Being able to interpret such terms will help in analyzing the geographic context of documents and in interpreting user queries that employ vague spatial language. There have been previous studies of the meaning of natural language qualitative spatial relations, while [13] show how nearness can be interpreted quantitatively by analyzing the use of phrases such as "walking distance" in combination with knowledge of the coordinates of the places that are referred to.

Spatial and Textual Indexing

When web documents have been categorized according to their geographical context they must be indexed in a way that enables them to be found quickly in response to user queries. Techniques for indexing documents according to the words that they contain are well established. Typically an inverted file of documents is created in which each word is associated with a list of the documents that contain the word. This text index can be combined with a spatial index that records which documents relate to particular regions of space. Building a spatial index of documents can be done if each document has one or more document footprints that represent the regions of geographic space to which the document refers. Each document footprint may correspond to the spatial extent of a geographic reference that occurs in the document. If there are many such references an effort may be made to establish the main geographic foci of the document as represented by a smaller number of footprints [1,14]. These footprints can then be indexed in the same way that any other piece of geometry would be indexed in a conventional GIS. A challenge remains, however, to find efficient ways to combine text and spatial indexes (see [15] for a summary of some approaches to spatio-textual indexing).

Retrieval of relevant documents requires matching the query specification to the characteristics of the indexed documents. As mentioned above, in conventional web search engines this starts by finding those documents that contain the query terms, before ranking the resulting documents. For geographical search, there is a need to match the geographical component of the query with the geographical context of the documents as represented by the document footprints. GIR queries can be characterized as a triplet of <theme> <relationship> <location> composed of a topic of interest in combination with a place name qualified by a spatial preposition such as near, in, or north of. The combination of place name (after disambiguation) and spatial preposition can be used to generate a query footprint representing, for example, the interpretation of an expression such as "near Bristol." This query footprint can then be used to access the relevant part of the spatial index and hence find document footprints that intersect the query footprint. The retrieved documents will then be the members of this latter set of geographically relevant documents that also contain the thematic (text) query terms in the query.

Geographical Relevance Ranking

Having found a set of potentially relevant documents they should then be ranked by some measures of their estimated degree of relevance to the query. Relevance with respect to the thematic part of the query and the retrieved documents can be represented by a score that takes into account factors such as the frequency of occurrence of query terms within retrieved documents as a function of the overall frequency of query terms within the whole collection. The spatial score can be some measure of the geometric match between the query footprint and the document footprints. These two scores can then be combined to find an overall relevance. [16] has described some methods for doing this in which the text and spatial scores are normalized to values between 0 and 1 before calculating their distance from an ideal combined score in the two-dimensional space of text and spatial scores.

User Interfaces

Retrieval of documents using GIR also provides a variety of research challenges in the development of user interfaces. Query formulation generally requires, as indicated above, specification of a triplet of <theme> <spatial relationship> <location>. This can easily be facilitated by a simple structured interface, but this assumes sufficient geographic knowledge to specify relevant place names. Other approaches to query formulation allow users to sketch a region of interest on a map, and enter a related concept within a textbox [11]. This approach presupposes that users are interested in a spatial relationship representing containment, but to date very little progress has been made in developing map-based interfaces that allow users the option of graphically specifying other spatial relationships.

The results of a query to a GIR system can be treated in an identical manner to those of a traditional search engine, and simply displayed as a ranked list. In practice, the nature of geographic search and the pervasiveness of map-based web services means that the overlaying of results on a map has become a natural and expected visualization mechanism. Several challenges exist in displaying the results of GIR searches through a map interface. Many relevant documents may have the same geographic footprint, and techniques are required to allow the aggregation of these relevant documents, while summarizing or filtering duplicate content. Equally, document footprints often have scopes which are not sensibly represented at all scales as a point (e.g., London) and methods are required to allow users to explore documents with extensive geographic scopes in meaningful ways. Finally, there is much room for techniques from the Geovisualization community to be applied in exploring the typically large sets of documents that are returned by GIR.

User Studies and Evaluation

In developing GIR techniques, it is very important to take cognizance of user needs and to develop techniques to evaluate the quality of approaches to GIR. With respect to user needs, work is required to analyze where gaps exist in current approaches and to analyze query logs, firstly to identify the types of geographic search users undertake, and secondly to assess where users fail or have difficulty in search [6,12].

Evaluation is a key part of IR. In general, evaluation strategies within IR either take the form of system-focused or user-centered studies [5]. The former are based around the use of test collections, the measuring of the relevance of documents to specific queries and the calculation of standard IR measures such as precision and recall for a variety of systems and settings. Such methods require substantial resources to implement, particularly since relevance judgments must be performed manually and the numbers of judgments required are large. Within IR, the long-running TREC experiments have provided a mechanism for the pooling of resources to the mutual benefit of the research community. In GIR, GeoCLEF [3] has gone some way towards addressing this need by mounting a campaign to compare performance of a range of approaches to GIR across a set of multilingual queries performed on document collections based on newspaper articles. However, there is much room for further research on relevance judgment and measures for GIR. Current techniques are based on a single dimension of relevance using standard IR measures, which in general do not take account of the G (geography) in GIR [10]. To date, very little user-centered evaluation has been performed in GIR.

Key Applications

Geographical information retrieval is applicable in all areas where there is a need to specify geographical constraints when searching for information in document collections such as those found in digital libraries and on the World Wide Web. This applies in particular to spatially-aware (or geographical) search engines.

▶ *This Chapter derives from the following article which was originally published in the International Journal of Geographical Information Science vol 22, 2008, pp 219–228.*

▶ *The article "Geographical Information Retrieval" and journal International Journal of Geographical Information Science are copyright of Taylor & Francis.*

Cross-references

▶ Gazetteers
▶ Geographic Information System
▶ Georeferencing
▶ Human-Computer Interaction
▶ Information Retrieval
▶ Inverted Files
▶ Spatial Indexing Techniques

Recommended Reading

1. Amitay E., Har'el N., Sivan R., and Soffer A. Web-a-where: geotagging web content. In Proc. of the 27th Int. Conf. on Research and Development in Information Retrieval, 2004, pp. 273–280.
2. Arampatzis A., van Kreveld M., Reinbacher I., Jones C.B., Vaid S., Clough P., Joho H., and Sanderson M. Web-based delineation of imprecise regions. Comput. Environ. Urban Syst., 30:436–459, 2006.
3. Gey F.C., Larson R.R., Sanderson M., Bischoff K., Mandl T., Womser-Hacker C., Santos D., Rocha P., Di Nunzio G.M., and Ferro N. GeoCLEF 2006: the CLEF 2006 Cross-language geographic information retrieval track overview. In Proc. Workshop on the Cross Language Evaluation Forum, 2007, pp. 852–876.
4. Jones C.B., Purves R.S., Clough P.D., and Joho H. Modelling vague places with knowledge from the web. Int. J. Geogr. Inf. Sci., 22:1045–1065, 2008.
5. Jones K.S. and Willett P. (eds.). Readings in Information Retrieval. Morgan Kaufmann, San Francisco, CA, 1997.
6. Jones R., Zhang W.V., Rey B., Jhala P., and Stipp E. Geographic intention and modification in web search. Int. J. Geogr. Inf. Sci., 22(3):229–246, 2008.
7. Larson R. Geographic information retrieval and spatial browsing. In GIS and Libraries: Patrons, Maps and Spatial Information, L. Smith, M. Gluck (eds.). University of Illinois, Urbana-Champaign, 1996, pp. 81–124.
8. Montello D., Goodchild M., Gottsegen J., and Fohl P. Where's Downtown?: Behavioral methods for determining referents of vague spatial queries. Spatial Cogn. Comput., 3:185–204, 2003.
9. McCurley S.K. Geospatial mapping and navigation of the web. In Proc. 10th Int. World Wide Web Conference, 2001, pp. 221–229.
10. Purves R.S. and Clough P. Judging spatial relevance and document location for geographic information retrieval, extended abstract. In Proc. 4th Int. Conf. on Geographic Information Science, 2006, pp. 159–164.
11. Purves R.S., Clough P., Jones C.B., Arampatzis A., Bucher B., Finch D., Fu G., Joho H., Syed A.K., Vaid S., and Yang B. The design and implementation of SPIRIT: a spatially aware search engine for information retrieval on the internet. Int. J. Geogr. Inf. Sci., 21:717–745, 2007.
12. Sanderson M. and Kohler J. Analyzing geographic queries. In Proc. of the ACM SIGIR 2004 Workshop on Geographic Information Retrieval, 2004.
13. Schockaert S., De Cock M., and Kerre E.E. Location approximation for local search services using natural language hints. Int. J. Geogr. Inf. Sci., 22(3):315–336, 2008.
14. Silva M., Martins B., Chaves M., Cardoso N., and Afonso A.P. Adding geographic scopes to web resources. Comput. Environ. Urban Syst., 30:378–399, 2006.
15. Vaid S., Jones C.B., Joho H., and Sanderson M. Spatio-textual indexing for geographical search on the web. In Proc. 9th Int. Symp. Spatial and Temporal Databases. 2005, pp. 218–235.
16. van Kreveld M., Reinbacher I., Arampatzis A., and van Zwol R. Multi-dimensional scattered ranking methods for geographic information retrieval. Geoinformatica, 9:61–84.

Geographic Information Services

▶ Location-Based Services (LBS)

Geographic Information System

Michael F. Goodchild
University of California-Santa Barbara, Santa Barbara, CA, USA

Synonyms

Geospatial information system; Spatial information system

Definition

A geographic information system (GIS) is a computer application designed to perform a wide range of operations on geographic information. Geographic information is defined as information about locations on or near the surface of the Earth, and may be organized in a variety of ways. Thus a GIS includes functions to input, store, visualize, export, and analyze such information. Commercial off-the-shelf GIS software is today capable of virtually any conceivable operation on geographic information, and capable of recognizing hundreds of different formats. GISs are used in a wide range of applications, from the management of the distributed assets of utility companies to emergency response. Their scientific applications are found in any discipline that deals with phenomena distributed over the surface of the Earth, from ecology to criminology. Increasingly GIS technology is encountered by ordinary citizens, in the form of map-making sites based on Google Maps, wayfinding sites such as MapQuest, and hotel-finding sites such as Expedia.

A wide range of introductory textbooks on GIS are available with various levels of sophistication, covering the fundamentals of representation, analysis, and application. Clarke [2] provides a general introduction; Longley et al. [4] give a somewhat more advanced and comprehensive perspective; and Worboys and Duckham [6] provide a computational viewpoint.

Historical Background

GIS has several roots, and today's technology represents a convergence among several independent developments. In the mid 1960s, the Government of Canada developed the Canada Geographic Information System (CGIS), the first to use the term GIS, as a means of generating products from the Canada Land Inventory. CGIS implemented a very small set of functions, including map overlay and area calculation, and in its initial form included no capabilities for map output, yet it solved many fundamental problems. Another set of developments centered around the desire to automate the process of making paper maps, and yet another was stimulated by the need to manage complex sets of data in support of transportation planning and census data collection. By the late 1970s, the commonalities in these various threads had been recognized, and the first commercial software products began to emerge. Foresman [3] provides a comprehensive history of GIS.

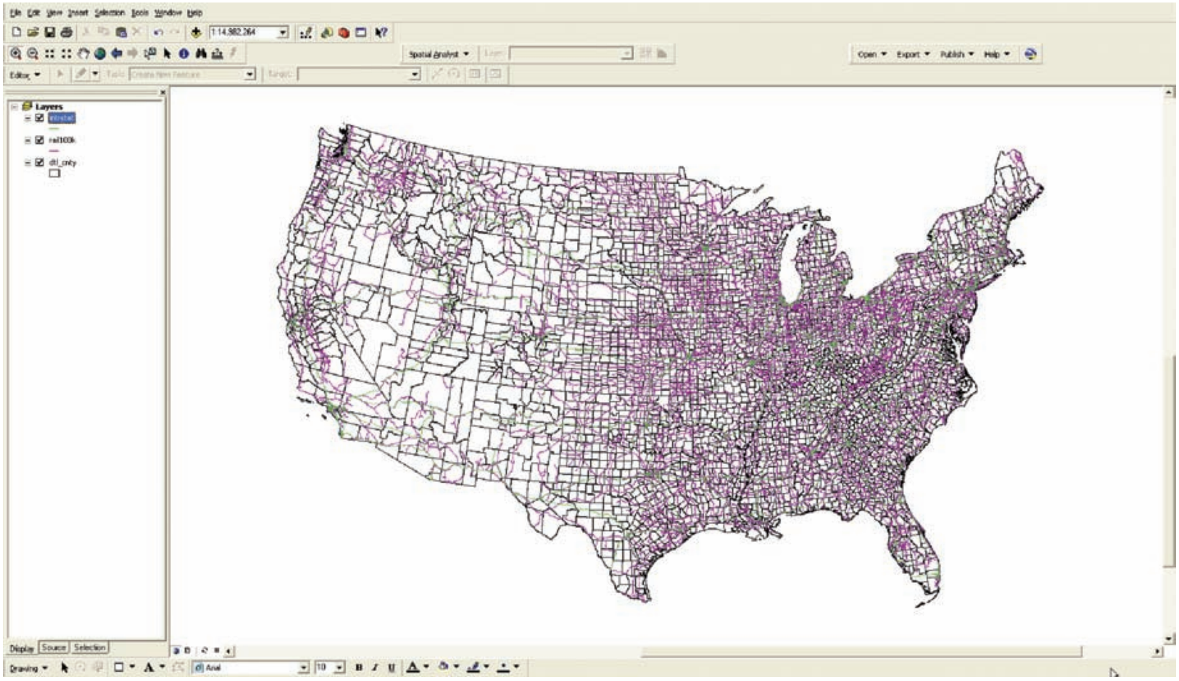
The early efforts to develop GIS were based on idiosyncratic data models and formats. By the late 1970s, the value of relational databases had been recognized, and GIS developers had begun to represent complex geographic features as interrelated collections of elements. A map of counties and their attributes, for example, could be represented as tables of nodes, edges, and faces; and the same approach could be used to represent a road network or a map of land cover types. Nevertheless, early relational database software was not easily adapted to the storage of the variable numbers of vertices needed to represent each edge, and so a hybrid approach had to be adopted in which some information was stored in a relational database and the remainder in a unique, proprietary structure. By the late 1980s, major problems of lack of interoperability began to impact the industry and led to the formation of the Open Geospatial Consortium.

By the 1990s, GIS designers had begun to adopt object-oriented database principles [1], and database technology and computing power had advanced to the point where hybrid architectures were no longer necessary. Greater flexibility in data modeling had opened the possibility of applying GIS tools to phenomena that were never traditionally associated with maps, such as events, transactions, movements and flows, and dynamic phenomena in general. GIS is no longer seen as a container of maps, but as an engine for the representation and analysis of spatio-temporal phenomena. The only limitation is that the information they contain be tied to specified locations on or near the surface of the Earth.

Figure 1 shows a screen shot of a typical application of a contemporary GIS, in this case ESRI's ArcMap.

Foundations

Two fundamentally distinct ways of conceptualizing spatial variation are recognized in GIS. In the discrete-object view, the Earth's surface is analogous to an empty table-top, on which are distributed a countable collection of features. The features may overlap, but between them is emptiness. The features might be represented as points, lines, or areas; and in some cases the third spatial dimension will be important and features may be represented as volumes. In the continuous-field view variation over the Earth's surface is described by a series of functions f of location $\mathbf{x}$, where f may be a measure, as in the case of elevation or temperature, or a class or name, as in the case of maps of soils or counties.



Geographic Information System. Figure 1. Screen shot of ESRI's ArcMap GIS. Three layers are displayed, as listed in the upper left: the county boundaries of the US (black), the railroads (blue), and the interstate highway network (green). Pull-down menus and toolbars provide access to a wide array of manipulation and analysis functions, and many more are available as third-party extensions. The display uses the Lambert Conformal Conic projection.

Location **x** may have two or three spatial dimensions, and may also include time in the case of dynamic phenomena. Phenomena distributed over networks, such as vehicle densities or pavement quality, can be conceptualized as fields distributed over a one-dimensional space that is in turn embedded in two- or three-dimensional space.

These conceptualizations can be implemented in GIS in two ways, as raster or vector structures. In a raster structure the set of possible locations is finite, being defined by a grid that is normally rectangular (though more complex grid geometries are needed when the Earth's curvature is important). In a vector structure, every feature is located using an appropriate number of coordinates. Areas and lines are normally represented as ordered sets of coordinates connected by straight lines, and known as polygons and polylines, respectively. These methods are straightforward in the case of discrete objects, but the representation of continuous fields requires another step of discretization, with six commonly used alternatives in the case of two-dimensional variation: (i) a set of irregularly spaced sample points, as used for example in capturing and representing weather observations; (ii) a set of regularly spaced sample points, as used in representing terrain; (iii) representation of the isolines of the field, as used in contour maps; (iv) a triangular mesh, with linear variation within the triangles and continuity of value across triangle edges; (v) a set of rectangular raster cells, as used in capturing and representing images; and (vi) a set of irregularly shaped, non-overlapping, and space-exhausting areas represented as polygons, as used in mapping land cover types or aggregated census data.

Whether raster or vector, any GIS data set must be referenced to the Earth's surface using some form of coordinate system. In the case of raster data this is usually achieved by registering the corner points of the raster; while in the case of vector data every feature is independently positioned in the coordinate system. Many alternative coordinate systems exist, including latitude and longitude plus numerous systems based on projections of the Earth's curved surface onto a plane. Some have been officially adopted by countries, such as the UK's National Grid, while others have been adopted by international agreement through agencies such as NATO, or by individual states in the case of the US State Plane Coordinate system. Unfortunately this leads to complexity and lack of interoperability, and any GIS must offer a comprehensive set of functions for managing coordinate systems. Moreover, the definitions of latitude and longitude are not entirely standard, being dependent on the mathematical figure adopted to represent the shape of the Earth, otherwise known as the datum, which adds another dimension of complexity. One approach to dealing with interoperability has been through the development of metadata standards, or standardized descriptions of data sets that include specification of such properties as coordinate systems. Metadata are now widely used by portals, libraries, and warehouses of geographic data that offer massive information resources to users over the Internet.

A GIS database is normally conceptualized as a collection of layers, each registered to the Earth's surface. A layer may contain the representation of a field, such as a map of land cover type, or it may contain a collection of discrete objects, such as buildings. This layer-based view of the world is often used as an icon, since it is so fundamental to GIS, and has appeared on the cover of more than one textbook. Raster-based layers will likely describe the geographic variation of only a single property, such as county name or land cover type, and because this property will exist for every cell and will have only a single recorded value per cell there is clearly a natural affinity between raster layers and continuous-field conceptualizations. Vector-based layers on the other hand may represent either collections of discrete objects with any number of associated attributes, or the variation of a single property conceptualized as a continuous field. Unfortunately today's GIS designs do not recognize the field/object distinction, with the result that no automatic procedures exist to avoid violating the constraints that apply in the field case. For example, a GIS will treat a set of digitized isolines representing a field as if they were discrete, independent polyline objects, and will not object if a user attempts to edit them in such a way as to make two isolines cross.

Many GIS applications employ some form of tiling, in which the geographic area covered by the database is divided into smaller units, often in the interests of computational efficiency. Tiles correspond to the map sheets of cartography, and are often defined on a rectangular basis. In some cases they may be evident to the user, who must develop explicit strategies for handling queries or analyses that involve multiple tiles; in other cases the tiling scheme may be transparent. In addition, a GIS will employ one or more systems of indexing, again in the interests of computational efficiency. Van Oosterom [5] provides a comprehensive survey of spatial database indexing options.

GIS implementations vary from the stand-alone desktop system designed to support the work of a single user, to departmental solutions that integrate the work of several people around a common database, to enterprise-wide solutions that organize the entire operation of a company or agency around its GIS. Increasingly, GIS functions are available on-line from servers, either based on data held at the server, or on data supplied by the user. For example, many Web sites now offer the GIS function of geocoding, which takes a list of postal addresses as input and returns their corresponding geographic coordinates.

A wide range of GIS software products are available to match these various implementation strategies, ranging from versions for hand-held devices through desktop systems to server-side GIS. The use of component-based software architectures has allowed vendors to build different products from the same collection of discrete elements. The trend in recent years has been away from the single-user desktop to integrated client-server architectures, and increasing reliance on data downloaded from Internet portals. Open-source GIS is growing steadily in popularity, and a number of low-cost options have recently appeared to challenge the marketplace dominance of the industry leaders, ESRI and Intergraph. Idrisi, originating from Clark University, continues to provide a software option with very strong support for decision making. Niche products also exist, particularly in transportation where the products of Caliper offer some powerful capabilities, and products from developing countries, particularly China, are increasingly competitive.

Key Applications

As noted above, the earliest applications of GIS were in isolated domains: land resource management; automated cartography; and transportation. By the late 1970s a broader vision of GIS had emerged as a general-purpose software application that could be used to solve a vast range of problems, all dealing in one way or another with the surface and near-surface of the Earth. Nevertheless, the first software products that emerged in the early 1980s found their most important applications in resource management, and were heavily adopted by forest-management companies and agencies, where they were used for compiling inventories, planning harvesting, and management of road access.

This viewpoint largely ignores the role of the military in GIS development, however. By the late 1950s the U.S. was heavily engaged in classified programs aimed at acquiring imagery from space, and in developing the systems needed to assemble, interpret, and analyze their products. Civilian satellites were first deployed in the early 1970s, and led quickly to a generation of GIS software, largely raster-based, that found useful applications in agriculture. Today the military and intelligence agencies are among the heaviest users of GIS, and many weapons systems and warfighting strategies depend on ready access to current, digital geographic information.

One of the most successful areas of GIS application has been in facilities management, and more generally in the management of distributed assets. Public and private utilities, planning departments of governments, transportation agencies, and managers of complex facilities such as university campuses will all use GIS routinely to inventory assets, manage and schedule maintenance, and address problems. Investments by individual companies may run to the millions of dollars, and involve hundreds of GIS workstations.

Of rapidly growing importance are GIS applications that impact the daily lives of the general public. These include route guidance, provided perhaps by an in-vehicle navigation system or by a Web service, and map-based services such as Zillow that provide current information on property values. The general public is increasingly familiar with GIS data sources, including remote sensing, and with concepts such as geo-tagging that permit information in sources such as Wikipedia to be georeferenced and thus linked to maps. The term neogeography is being used to describe such novel applications of GIS.

Future Directions

The days when GIS was an exotic computer application familiar only to a small elite, and accessible only after extensive training, are long gone; today virtually everyone with access to the Internet is familiar with at least some of its capabilities. The meaning of the term itself has become confused: some would reserve it exclusively for the single-user desktop application, while others would extend it to cover what is now a vast array of activities, for which the adjective geospatial has emerged recently as an alternative umbrella term. In discussing future directions, then, it is possible to take either the first, somewhat narrow perspective, or the second broader one.

From the narrow perspective, GIS software will continue to evolve, particularly in the direction of greater support for dynamics. Some application areas are currently in their infancy, and will likely grow in importance in the next few years. They include human health, the understanding of disease transmission and diffusion, and the assessment of health service outcomes; and business, whose mantra "location, location, location" feeds directly into GIS applications. Related areas of logistics, real estate, and insurance are also ripe for greater use of GIS.

A rich research agenda has evolved over the past decade, and will likely produce results that will in turn impact future generations of GIS software. The University Consortium for Geographic Information Science has taken the lead in this context, with short- and long-term agendas that address topics ranging from spatial cognition (How can GIS interfaces be made easier to use?) to the social impacts of GIS technology (Who gains, who loses?). The topic of uncertainty remains a thorny issue, with much useful research completed on the sources and management of uncertainty in GIS but little adoption in mainstream software. As a result, the user is left with little choice but to accept the results of GIS analysis at face value, while knowing that the data on which they were based is inevitably imperfect and uncertain.

From the broader perspective, GIS functions will continue to become more available in Web services. In the past year a very interesting series of developments have occurred that are enabling citizens to create their own geographic information, and to integrate and publish it on the Internet. Sites such as OpenStreetMap are enabling communities to create their own maps; Wikimapia is collecting descriptions of features on the Earth's surface from vast numbers of volunteers; and Flickr is assembling a vast collection of geo-referenced photographs. This process of volunteering geographic information is having a profound impact on citizens' knowledge of the planet, and on their engagement with geography and GIS.

Cross-references

- ▶ Data Models
- ▶ Field-Based Spatial Modeling
- ▶ Spatial Indexing Techniques
- ▶ Spatial Network Databases
- ▶ Spatial Data Analysis

Recommended Reading

1. Arctur D. and Zeiler M. Designing Geodatabases: Case Studies in GIS Data Modeling. ESRI Press, Redlands, CA, 2004.
2. Clarke K.C. Getting Started with Geographic Information Systems. Prentice Hall, Upper Saddle River, NJ, 2003.
3. Foresman T.W., ed. The History of Geographic Information Systems: Perspectives from the Pioneers. Prentice Hall, Upper Saddle River, NJ, 1998.
4. Longley P.A., Goodchild M.F., Maguire D.J., and Rhind D.W. Geographic Information Systems and Science. Wiley, Chichester, UK, 2005.
5. van Oosterom P. Spatial access methods. In Geographical Information Systems, vol. 1, P.A. Longley, M.F. Goodchild, D.J. Maguire, and D.W. Rhind (eds.). Wiley, New York, NY, 1999, pp. 385–400.
6. Worboys M.F. and Duckham M. GIS: A Computing Perspective. CRC Press, Boca Raton, FL, 2004.

Geographic Information

▶ Geography Markup Language

Geographic Web Search

▶ Geo-Targeted Web Search

Geographical Analysis

▶ Spatial Data Analysis

Geographical Data Analysis

▶ Spatial Data Analysis

Geographical Databases

▶ Semantic Modeling for Geographic Information Systems

Geographical Metadata

▶ Geospatial Meta Data

Geography Markup Language

Jayant Sharma, John Herring
Oracle Corporation, Nashua, NH, USA

Synonyms

ISO 19136; Geographic information; Geography markup language

Definition

Geography Markup Language (GML) (http://www.opengis.net/gml/) is an XML vocabulary that can be used as the basis for the modeling, transfer, communication, or storage of geographic feature information. It is defined and maintained by members of the Open Geospatial Consortium (OGC), a voluntary consensus standards organization that develops standards for geospatial information handling and location based services. It has been accepted as one of and harmonized with the standards of ISO TC 211: Geographic information (http://www.isotc211.org/). The official copies of all versions of the GML specification and related OGC standards can be found at http://www.opengeospatial.org/standards/. The most recent version of GML, version 3.2.1, is identical with ISO 19136 (http://www.isotc211.org/Outreach/Overview/Factsheet_19136.pdf).

The primary purpose and rationale for GML, as described by the OGC, is to provide an open, vendor-neutral format for the definition of geographical entities used by applications, in various domains, for geospatial information processing and exchange. The enhanced ability of organizations to, efficiently and effectively, re-use, share, integrate, and consequently extract more value from, their extremely valuable geospatial information across operational units and processes is one immediate benefit of using GML.

Historical Background

GML (version 1.0) was first submitted to the OpenGIS Consortium, now Open Geospatial Consortium in 1999, as a "request for comment" (RFC) by authors from CubeWerx, Compusult, Galdos Systems, MapInfo, and Oracle. It was accepted by OGC and published in May 2000. GML 2, the first widely accepted version, was published in February 2001. GML 2.1 is still commonly used today.

GML 3.1 was published by OGC in February 2004, and submitted to ISO TC 211: Geographic Information, for consideration as an ISO standard. This would make it acceptable under the laws of some countries as the basis for national standards. It has since been accepted and ISO 19136 (GML, version 3.2.1), became an International Standard in 2007.

Foundations

GML is based on a feature and property model. Any complete GML file contains a collection of features, and each feature contains a collection of properties, and each property contains its value, which may either be direct (lexically contained within the property as an XML sub element) or indirect (pointed to by the property using an xlink).

Features represent real-world entities, such as roads or parks. Some real-world entities are composed of other entities and hence are represented as collections of features. For example, a city may be represented as a collection of roads, parks, land parcels etc.

Features have properties that have a name and type. For example, a road has a name (i.e., a property called "name") and its value is of type "String." Features may also have named geometry-valued properties. For example, a road might have a geometry property named "centerLineOf" that is of type "Linestring."

GML implements a subset of the ISO 19107 standard geometry model that defines the primitive geometry types and their hierarchy. GML 2 was concerned with simple features that only have a simple two-dimensional geometry property. GML 3 addresses requirements of applications that need to model more complex geographic phenomena that are dynamic (e.g., coverages or sensors), and have non-linear (e.g., circular or parametric arcs) or 3-D (cubes, prisms) geometry, 2-D topology, or temporal properties. To be useful, such feature models must include spatial and temporal reference systems, units of measure, metadata, and default styling (or portrayal) information.

Base and Application Schema

In order to enable an open, interoperable, vendor-neutral means of modeling and exchanging geographic information, GML includes types for features, geometry, topology, coordinate reference systems, grids, measures, observations, and temporal reference system/values.

Feature types form the basic logical construct and all other types are used to express property values for features. Geometry types are used to describe the basic spatial attributes of a feature. Topology types can be used to describe the spatial relations between those geometries. Coordinate and coordinate reference systems describe the manner in which point locations are described for geometry (and related purposes). Grids are commonly used for "image-like" raster structures for distributing measurements across space. Measure and observation types are used for describing measurable attributes and their units. Temporal values and their reference systems are used for expression of time.

Applications define and describe a "feature schema" as defined in ISO 19109: Rules for application schema. These schemata control the types of features that can actually be stored in a GML file. Each schema uses the basic GML structure as defined above and created "feature types" which determine the name of the feature types and which properties a particular feature type must (mandatory) or may (optional) contain, and with what cardinality (maximum and minimum occurrence). Because these application schemata are restricted by ISO 19109 and by the syntax of XML schema, they are single classification objects as is common in most programming languages.

Profiles

GML is quite comprehensive and consequently complex, even its early versions. To compensate for this for particular information communities (informal groups of users with similar data content needs), "profiles" are often defined as subset of the full base schema for use within their applications.

One profile is the "point profile" which simply describes the way geographic point location can be expressed. It is used as the basis for geographic extension of other standards that only require a geographic point as a location reference and hence do not use, or require, the GML feature and geometry model.

The most common profile is the one used for another OGC standard, Simple Feature Access that includes a data architecture volume (http://www.opengeospatial.org/standards/sfa). It was defined first in 1999, and became an ISO standard soon thereafter. The GML simple features profile was developed to support another OGC specification, namely, the OpenGIS Web Feature Service (WFS) Implementation Specification. The primary purpose is to define a useful subset of GML that (i) "lowered the implementation bar," and (ii) supported the Simple Features for SQL (SF-SQL) geometry model and hence the WFS specification. So this profile "prescribes the encoding of GML application schemas in sufficient detail that client applications do not need to deal with the entire scope of XML-Schema and GML but only need to understand a restricted subset of both specifications in order to be able to interpret schema documents generated or referenced by servers offering data encoded in GML." As a consequence, this document is just 49 pages long compared to over 800-page length of the full GML specification.

Other relevant profiles are the common CRS, and common dictionary profiles. These define encodings of commonly used coordinate reference systems and conversions, and simple dictionaries respectively.

Key Applications

The WFS (Web Feature Service) implementation specification defines an interface for retrieving or updating, geographical features encoded in GML, using HTTP.

RSS (real simple syndication, http://www.rssboard.org/rss-specification) is a news feed mechanism for the web and GeoRSS (http://www.rssboard.org/rss-specification) uses the geometry of the simple feature profile to geographically locate the information in the news. See http://www.georss.org/gml.html for some examples of GML usage in GeoRSS.

JPEG2000 (http://www.jpeg.org/jpeg2000/) is a digital image format that allows "XML" data to be stored with the image data. GML is the preferred format for geographic information and the combination of image and geographic information allows JPEG2000 to be used for geographically referencible images such as satellite and aerial imagery.

CityGML (http://www.citygml.org/) is an application schema for GML that integrates building information, at various levels of detail, into 3-D city and landscape models. The basic concept is to create a full 3-D model capable of supporting a wide variety of applications associated to such fields as environment, telecommunications, security, navigation and others requiring highly accurate and detailed building and landscape models. Some examples of GML as used in CityGML are at http://www.citygmlwiki.org/index.php/Examples_and_WFSs. A free viewer for CityGML can be found at http://www.3dgeo.de/citygml.aspx.

The IETF (Internet Engineering Task Force – http://www.ietf.org/) has several "rfc" specifications that use GML for location information. In particular, *IETF RFC 4119 – A Presence-based GEOPRIV Location Object Format and IETF RFC 3863 – Presence Information Data Format (PIDF)*. Each uses a profile of GML for location information.

Schemas for GML

All versions of GML have their schemas posted for public use in some subdirectory of http://schemas.opengis.net/gml/. For example http://schemas.opengis.net/gml/3.2.1/contains the schema for the latest version of GML version 3.2.1 [5], which is the same as ISO 19136.

Examples

Some fairly simple examples of GML can be found in the Wikipedia article located at http://en.wikipedia.org/wiki/Geography_Markup_Language.

Cross-references

- ▶ Abstraction
- ▶ Data Models with Nested Collections and Classes
- ▶ Document Databases
- ▶ Geographical Information Retrieval
- ▶ Geographic Information System
- ▶ Georeferencing
- ▶ Geospatial Metadata
- ▶ Object Data Models
- ▶ Resource Description Framework (RDF) Schema (RDFS)
- ▶ Spatial and Spatio-Temporal Data Models and Languages

- ▶ Spatial Data Types
- ▶ Spatio-Temporal Data Warehouses
- ▶ Unified Modeling Language
- ▶ W3C
- ▶ XML
- ▶ XML Metadata Interchange
- ▶ XML Parsing, SAX/DOM
- ▶ XML Programming
- ▶ XML Schema
- ▶ Xpath/Xquery
- ▶ XSL/XSLT

Recommended Reading

1. Alberto B., Negri M., and Pelagatti G. An ISO TC 211 conformant approach to model spatial integrity constraints in the conceptual design of geographical databases. In Advances in Conceptual Modeling – Theory and Practice, 2006, pp. 100–109.
2. Bacharach S. The GML simple features profile and you, directions magazine.
3. Lake R., Burggraf D., Trininic M., and Rae L. Geography Markup Language: Foundation for the Geoweb, Wiley, New York, 2004.
4. Lu C., Santos R.F. Jr., Sripada L.N., and Kou Y. Advances in GML for geospatial applications. GeoInformatica., 11(1):131–157, 2007.
5. Portele C. (ed.). OpenGIS Geography Markup Language (GML) Encoding Specification 3.2.1, OGC Document #07–036.
6. Vretanos P. A. (ed.). OpenGIS Simple Features Profile (1.0.0), OGC Document #06–049r1.
7. Warnill C. and Bae H. A specification of a moving object query language over GML for location-based services. In Proc. 6th Asia-Pacific Web Conference on Advanced Web Technologies and Applications, 2004, pp. 788–793.
8. Zhong-Ren P. and Zhang C. The roles of geography markup language (GML), scalable vector graphics (SVG), and web feature service (WFS) specifications in the development of internet geographic information systems (GIS). J. Geogr. Syst., 6(2): 95–116, 2004.

Geometric Data Types

- ▶ Spatial Data Types

Geometric Mean Average Precision

- ▶ GMAP

Geometric Stream Mining

CECILIA M. PROCOPIUC
AT&T Labs, Florham Park, NJ, USA

Definition

Let $P = \{p_1, p_2, ...\}$ be a stream of points in the metric space (X, L_q). Usually, $X = \mathbb{R}^d$ or $X = \{1, ..., U\}^d$ (discrete case), and $L_q = L_2$ is the Euclidean distance. The set P is called a *spatial data stream*. Geometric stream mining algorithms compute the (approximate) answer to a geometric question over the subset of P seen so far. For example, the *diameter* problem asks to maintain the pair of points that are farthest away in the current stream. A more comprehensive list of problems is presented later.

Historical Background

Geometric algorithms in the offline setting have been extensively studied over the past decades. Their applications encompass many fields, such as image processing, robotics, data mining, or VLSI design. For an introduction to computational geometry, refer to the book [8]. On the other hand, research on spatial data streams is a recent development. Shortly after the first results on numeric data streams appeared, a slew of papers argued that in many applications numeric streams exist in metric spaces, and that the capabilities of such systems can be expanded by taking into account the underlying metric, and by supporting geometric queries. Queries can be similar to offline databases (e.g., range queries), but they can also be about the shape and extent of the stream (e.g., diameter). Most examples of spatial streams come from sensor networks, traffic monitoring, and satellite data, all of which have location information.

Foundations

Stream Models

Online algorithms come in two flavors. In the first model, points can only be inserted in the stream. In the second, both insertions and deletions are permitted. The latter case is usually referred to as the *turnstile* or *dynamic* model. Allowing deletions introduces distinct challenges. Frahling et al. [5] proved that certain computations over a dynamic stream from $\{1, ..., U\}^d$ require $\Omega(U^d)$ space in any deterministic approach, but only $O(d \log U)$ space if randomization is allowed. This contrasts with the insertions-only case, for which there are deterministic space-efficient methods.

Further classification of stream models depends on whether the metric space X is continuous or discrete. In the following, *data stream* refers to insertions-only streams from a continuous metric space, unless stated otherwise.

Classes of Problems

Some geometric stream problems are closely related, i.e., they can be solved by the same methods. Such a classification is presented below. The details of the techniques are deferred to the next subsection.

Range Counting and Robust Statistics *Range counting* asks for the number of points that lie inside an arbitrary axis-parallel hyper-rectangle. The query hyper-rectangle is not known a priori. The *Tukey depth* of a point p is the minimum proportion of points among all the half-spaces that contain p (the larger the depth, the more "central" the point). The *simplicial depth* of p is the proportion of simplices that contain p, among all simplices that are convex hulls of $d + 1$ data points. These problems can be approximated using a so-called *ε-approximation* of the data.

Extent Measures The *diameter* of a point set is the maximum distance between two points in the set. The convex hull is the smallest convex region that contains the set. The *width* is the minimum distance between two parallel hyper-planes so that the set lies between them. The *minimum enclosing cylinder, respectively sphere* is the cylinder, respectively sphere, of minimum radius containing the points. The *minimum cylindrical, respectively spherical, shell* is the region of minimum width between two concentric cylinders, respectively spheres, that contains the points. These problems can be approximated by computing appropriate *ε-kernels*. More efficient methods exist for the diameter and convex hull, but they do not extend to the other problems.

Geometric Graphs Many graph problems have a natural equivalent when the graph is embedded in a metric space, and the distance between points is given by that metric. Such problems include the *minimum weight spanning tree*, *minimum match*, and *TSP*. In the online version, the solution cannot be stored explicitly, so only the cost is computed. A general technique is to reduce the problems to vector computations over streams, and use existing algorithms from that area. Another approach is to maintain appropriate ε-coresets.

Nearest Neighbor and Skyline The classical nearest neighbor problem asks for the data point closest to a query point. This cannot be computed in sub-linear space, as any data point can be the answer to some query. Thus, several modifications have been proposed for streams. Korn et al. [7] studied *reverse nearest neighbors*, in which the data set is known a priori, and the stream consists of query points. However, some data points may be inactive at times, and the query is only with respect to active points. An indexing structure is pre-computed on the data points (in one dimension, this involves sorting the points and putting a geometric grid between each pair of consecutive points). A different framework is studied by Böhm et al. Queries and points have time spans in which they are active, and the k-nearest neighbors are computed from among the active data points during the query's entire span. This can be reduced to the *skyline* problem, defined as follows. The skyline of a dataset with respect to a fixed quadrant contains those points p_i for which the quadrant translated with the origin in p_i has no other points. For example, for the lower right quadrant, p_i is in the skyline if all points whose x-coordinate is larger than p_i also have larger y-coordinate. For nearest neighbors, the data stream is transformed into new coordinates, with the x-axis being expiration time, and the y-axis being the distance to the query. Then the nearest neighbors are the points on the skyline. The skyline can be maintained exactly, but may have $\Omega(n)$ complexity. The authors propose heuristics for maintaining approximate skylines in that case.

General Techniques

Most geometric problems cannot be solved exactly in sublinear space, so approximate solutions are sought. Three techniques for designing approximation algorithms for spatial data streams are presented below.

1. Merge and Reduce A popular approach in geometric algorithms is to compute a "sketch" of the dataset, so that the optimal solution can be approximated by a computation over the sketch. The sketch is a small subset of the input, whose size usually depends only on the approximation error. Given its space efficiency, this approach is a good candidate for stream algorithms. To maintain a sketch online, the stream is divided into substreams, and a sketch is independently computed for each substream. As more points arrive, the algorithm merges older sketches and replaces them by a smaller sketch.

Figure 1 illustrates this process when the stream is divided based on the *logarithmic method*: Let n denote the number of points seen so far. Let $0 \leq i_1 < i_2 < \ldots i_k \leq \log n$ be the 1-bits in the binary representation of n, i.e., $n = 2^{i_k} + \ldots + 2^{i_2} + 2^{i_1}$. Divide the stream into disjoint substreams of consecutive points $P_{i_k}, \ldots, P_{i_1}$, such that P_{i_j} arrives before $P_{i_{j-1}}$ for all $2 \leq j \leq k$, and $|P_{i_j}| = 2^{i_j}$. When a new point arrives, assign it to P_0. If $|P_0| = 1$, i.e., P_0 was empty before, return. Otherwise, $|P_0| = 2$ violates the cardinality condition. Merge P_0 into P_1, and set $P_0 = \emptyset$. Continue the process until all sets have the correct cardinality (if necessary, create a new set $P_{i_{k+1}}$).

Since P_{i_j} cannot be stored exactly, its sketch S_{i_j} is stored instead. Merging two substreams is simulated by merging their sketches – this is the *merge step*. \To maintain the space bound, a sketch of the merged sketches is computed – the *reduce step*. Finally, a sketch S of the whole stream is computed as a sketch of all S_{i_j}, and the original optimization problem is solved over S. To guarantee correctness, sketches must observe two key properties:

1. If S_1, S_2 are sketches of P_1, P_2, then $S_1 \cup S_2$ is a sketch for $P_1 \cup P_2$ (correctness of merge step).
2. If S is a sketch of P and T is a sketch of S, then T is also a sketch of P, possibly with larger error (correctness of reduce step).

Let σ be the size of one sketch; σ is independent of n, but depends on ε. Then the overall space is $O(\sigma \log n)$. However, errors usually increase during the reduce step. To guarantee that the final sketch has error at most ε, a smaller error ε_i is used for each S_i. This increases the space by logarithmic factors. A recent result replaces the logarithmic method by a geometric strategy, eliminating all $\log n$ factors.

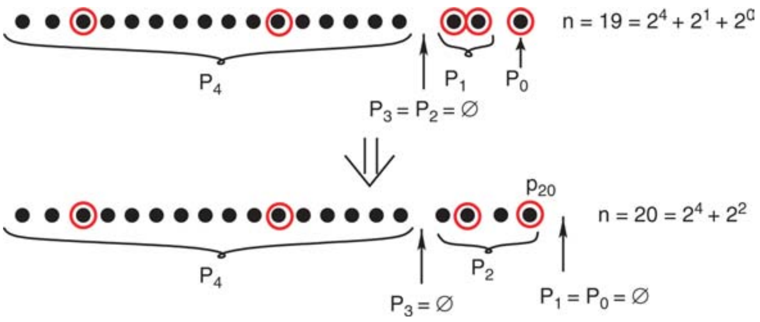
Two types of sketches apply to a wide range of problems. The first, called ε-approximations, was defined by Vapnik and Chervonenkis, in the context of set systems with finite VC dimension. For an introduction to this theory see, e.g., the book. Some classes of set systems to which the theory applies are specified below. The second type of sketches, called ε-coresets, were first introduced by Agarwal et al. and have emerged as a powerful tool in geometric approximation [1].

ε-approximations: A subset $S \subseteq P$ is an ε-approximation of P if for any *range* R from a fixed family of ranges of bounded VC-dimension

$$\frac{|R \cap S|}{|S|} = \frac{|R \cap P|}{|P|} \pm \varepsilon.$$

Examples of families of ranges of bounded VC-dimension include the set of all hyper-rectangles, and the set of half-spaces. Thus, a range counting query can be answered as follows. Let Q be the query hyper-rectangle and S be the current ε-approximation of the stream P. Then the number of points inside Q is approximated by $|Q \cap S| \cdot \frac{|P|}{|S|}$, and the error is $\pm\, \varepsilon |P|$.

Any set system of bounded VC-dimension admits an ε-approximation of size $O(\varepsilon^{-2} \log(\varepsilon^{-1}))$. Matousek designed a deterministic offline algorithm for computing such an ε-approximation. The algorithm was extended to the online setting by Bagchi et al. [5] using the logarithmic method as above.



Geometric Stream Mining. Figure 1. Merge and reduce via logarithmic method: A sketch of size at most two (*circled points*) is stored for each P_i. When p_{20} arrives, the sketch of P_2 is computed from p_{20} and the previous sketches of P_0,P_1.

ε-coresets and ε-kernels: Let $\mu : 2^X \rightarrow \mathbb{R}^+ \cup \{0\}$ be the measure to be optimized. For example, $\mu(P)$ can be the width, or the cost of the minimum weight spanning tree of P. Assume that μ is monotone, i.e., for any $P_1 \subseteq P_2$, $\mu(P_1) \leq \mu(P_2)$. A subset $S \subseteq P$ is an ε-coreset of P if

$$(1-\varepsilon)\mu(P) \leq \mu(S).$$

Many geometric problems have ε-coresets of small size, i.e., $|S| = 1/\varepsilon^{O(1)}$.

The related notion of ε-kernels provides a blueprint for computing coresets for several measures. For any unit vector $v \in \mathbb{R}^d$, let $\mu_v(P)$ denote the *directional width* of P along v. Formally, $\mu_v(P) = max_{p\in P}\langle p, u\rangle - min_{p\in P}\langle p, u\rangle$, where $\langle \cdot,\cdot \rangle$ is the inner product. See Fig. 2a. A subset $S \subseteq P$ is an ε-kernel of P if it simultaneously ε-approximates all directional widths, i.e., for all unit vectors v,

$$(1-\varepsilon)\mu_\upsilon(P) \leq \mu_\upsilon(S).$$

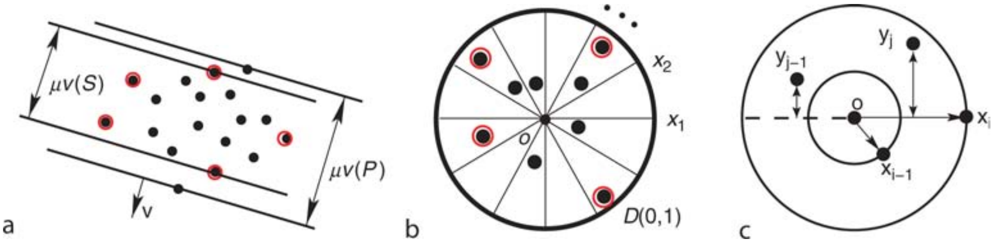
Figure 2b illustrates the offline algorithm by Agarwal et al. [1] for computing an ε-kernel in two dimensions. The algorithm assumes that all points of P lie inside the unit disk centered at the origin $\mathcal{D}(o, 1)$. It also assumes that P is "fat," i.e., there exists a constant $0 < \alpha < 1$ such that $\mu_\upsilon(P) \geq \alpha diam\ (P)$ for any direction v. Both conditions are enforced by applying an affine transform. The algorithm pre-defines a set of directions $\overrightarrow{ox_1},\ldots,\overrightarrow{ox_r}$ and computes the extremal points in each direction. Let $S \subseteq P$ be the set of at most $2r$ extremal points. The authors show that one can choose $r = O(1/\sqrt{\varepsilon})$ directions, so that S is an ε-kernel. For the d-dimensional case, $r = O(\varepsilon^{-(d-1)/2})$.

Maintaining extremal points along a fixed set of directions extends to the online setting. However, computing the affine transform that insures that P is fat requires first scanning the whole set. Thus, the logarithmic method is used for the online version. Different affine transforms are applied to different substreams.

A significant improvement is due to Chan [3], who eliminates the $\log n$ factors from the space bound. He partitions the data stream based on significant changes in the "shape" of the current substream, rather than its number of points. As before, a different affine transform applies to each substream, but the transform can be computed from the first three substream points.

In two dimensions, he uses the following affine transform. For any set P, let $o \in P$ be a fixed point, and let $x, y \in P$ such that $||op|| \leq 2||ox||$ and $d(p, ox) \leq 2d(y, ox)$ for all $p \in P$. Translate P to make o the origin. Rotate and scale P so that $x = (1/2, 0)$. Map each $(a,b) \in P$ to $(a, b/2d(y, ox))$. Let τ be the resulting transform. Then $\tau(P)$ is fat and $\tau(P) \subseteq \mathcal{D}(o, 1)$.

Let o be the first point in the stream; see Fig. 2c. Then o is assigned to all substreams (which are no longer disjoint). Divide the stream into sequences called *epochs*. An epoch E_i starts with a stream point x_i, and contains any subsequent point p so that $||op|| \leq 2||ox_i||$. Once a point q arrives so that $||oq|| > 2||ox_i||$, epoch E_{i+1} starts with $x_{i+1} = q$. Each epoch E_i is further divided into *subepochs* as follows. The jth subepoch $E_{i,j}$ which started with point $y_j \in E_i$ contains all subsequent points $p \in E_i$ for which $d(p, ox_i) \leq 2d(y_j, ox_i)$. When a point q arrives so that $d(q, ox_i) > 2d(y_j, ox_i)$, and q does not start a new epoch, then subepoch $E_{i,j+1}$ starts with $y_{j+1} = q$. A substream is a subepoch $E_{i,j}$, plus o and the point x_i that starts epoch E_i. An affine transform as above is computed at the beginning of each $E_{i,j}$, using points o, x_i and y_j.



Geometric Stream Mining. Figure 2. Computing an ε-kernel: (a) an ε-kernel S (circled points) approximates the directional width for all directions υ; (b) offline algorithm: kernel = extreme points along fixed directions (circled points); (c) online algorithm: x_{i-1}, x_i start epochs $i-1$, resp. i, where $d(o, x_i) > 2d(o, x_{i-1})$; y_{j-1}, y_j start subepochs $j-1$, resp. j, inside epoch i, where $d(y_j, ox_i) > 2d(y_{j-1}, ox_i)$.

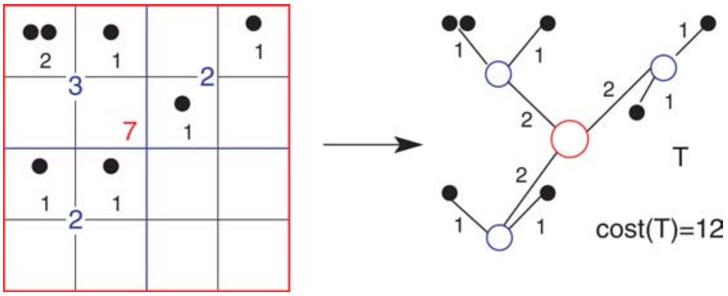
The number of subepochs can be $\Omega(n)$. However, Chan proves that it is sufficient to maintain kernels only for the most recent $\log(1/\varepsilon)$ epochs. Older epochs are so close to o that they can be represented by only two points. A similar argument applies to sub-epochs. The overall space is $O((1/\sqrt{\varepsilon})\log^2(1/\varepsilon))$, which was recently improved to $O(1/\sqrt{\varepsilon})$. In higher dimensions, a slightly different algorithm requires $O([(1/\varepsilon)\log(1/\varepsilon)]^{d-1})$ space.

2. Reduction to Vector Problems This technique has been applied to dynamic spatial streams, as the previous approach cannot handle deletions. The method was introduced by Indyk [6] and was used to approximate several geometric graph problems. It assumes a discrete space $\{1,\ldots,U\}^d$ and imposes a hierarchical decomposition on it. Statistics on the number of points in each cell is maintained under both insertion and deletion, using previous results from numerical data streams.

Figure 3 illustrates the method in two dimensions. The space decomposition consists of $k = \log U$ nested square grids $G_0,\ldots,G_k$ with side lengths $2^0,\ldots,2^k$, randomly shifted along each axis. This decomposition induces a tree structure, where a cell of G_i has at most four children corresponding to the non-empty cells of G_{i-1} contained in it. Data points are stored in leaves. The edge between a cell of G_i and its child from G_{i-1} has weight 2^{i-1}. Conceptually, the graph problem is solved on this tree, where the distance function is the shortest path. Indyk proves that, for randomly shifted grids, the expected cost of the tree solution is within $O(\log U)$ from the cost of the optimal solution. An $O(1)$ approximation of the tree solution can be computed by only maintaining statistics on the number of points in each non-empty cell. For the minimum spanning tree problem, the optimal solution on the induced tree is the tree itself, whose weight is $\sum_{i=0}^{k-1} 2^i n_i$, where n_i is the number of non-empty cells of G_i. Thus, for each i, it is sufficient to estimate the 0-norm of the vector defined over the cells of G_i under increment and decrement operations (i.e., when $p \in P$ is inserted, the counter of the cell containing p in G_i is increased by 1, and when p is deleted, the counter is decreased by 1). Thus, the original problem is reduced to a vector problem over data streams, which has already been studied. Other geometric graph problems can similarly be reduced to estimating vector norms of streams.

3. Random Sampling Several offline algorithms have a single-pass flavor and could work online, except that they need to select random samples from the data. If one can maintain random samples over a stream, then the algorithms extend online. When the stream is insertions-only, a random sample can be maintained via the reservoir sampling technique by Vitter [9]: Select the ith stream point p_i with probability r/i, where r is the size of the sample. If p_i is selected, it replaces a previous sample point, chosen uniformly at random.

For dynamic discrete streams, Cormode et al. [4] proposed a more general technique (a similar approach was independently designed by Frahling et al. [5]). The stream points are hashed to levels, such that the likelihood of being hashed to level l decreases exponentially in l. More exactly, if h is a universal hash function and $x \in \{1,\ldots,U\}$ is the stream point, then x is assigned to the level l for which $2C^{l+1}U \leq h(x) < 2C^lU$ ($0 < C < 1$ is a constant). Hence, there are $O(\log U)$ levels. Each level l has two counters $sum[l]$ and $count[l]$, initialized to zero, as well as a collision detection structure. If x is assigned to level l, then



Geometric Stream Mining. Figure 3. Cell count statistics (e.g., how many are non-zero) lead to approximate solution. Points can be inserted and deleted multiple times.

for each insertion of x, $sum[l] \leftarrow sum[l] + x$ and $count[l] \leftarrow count[l] + 1$; and for each deletion of x, $sum[l] \leftarrow sum[l] - x$ and $count[l] \leftarrow count[l] - 1$. If $count[l] > 0$ and there is only one point assigned to level l (as determined by the collision detection structure), then $sum[l]/count[l]$ is chosen in the random sample (note that $sum[l]/count[l] = x$, where x is the point assigned to l). The authors prove that there is a constant probability of finding such a level l, and that each point present in the stream has equal probability of being assigned to l. (A point is present in the stream if it has been inserted more times than it has been deleted.) To choose a random sample of size r, $O(r)$ independent copies of this structure are maintained.

Since a random sample of size $O(1/\varepsilon^2 \log(1/\varepsilon))$ is an ε-approximation with high probability, this provides a different method for maintaining ε-approximations. Frahling et al. [5] used the technique for the minimum spanning tree problem, improving the $O(\log U)$-approximation to $(1 + \varepsilon)$-approximation.

Key Applications

Most streaming applications whose data are embedded in a metric space require some analysis of the "geometry" of the stream. Three main areas, to which the above algorithms apply, are mentioned below.

Sensor networks Basic analysis of such systems computes information about the shape and extent of some sensor-covered area. For example, a system of sensors that monitor air quality can track the spread of a chemical spill, by maintaining the convex hull of the sensors that signal. In other applications, such as herd monitoring, the sensors are mobile, and the application tracks the distribution and density of the herd. In this case, it is useful to answer range counting and statistics queries.

Fixed wireless telephony This is an application of reverse nearest neighbor searching. Fixed wireless base stations are installed so that residential customers connect to the closest one that can handle the traffic. To optimize the performance, stations may be turned on or off depending on the stream of calls. To insure service quality, the system monitors station loads, as well as the maximum distance between a customer and the nearest station.

Mobile networks Such networks have a dynamic topology, in which connections between nodes appear and disappear based on their relative location. In this case, maintaining information on the network topology requires solving geometric graph problems.

Cross-references

- ▶ Approximate Query Processing
- ▶ Clustering on Streams
- ▶ Clustering Overview and Applications
- ▶ Sensor Networks
- ▶ Spatial Data Mining
- ▶ Stream Mining
- ▶ Stream Models
- ▶ Stream Sampling

Recommended Reading

1. Agarwal P.K., Har-Peled S., and Varadarajan K.R. Approximating extent measures of points. J. ACM, 51(4):606–633, 2004.
2. Bagchi A., Chaudhary A., Eppstein D., and Goodrich M.T. Deterministic sampling and range counting in geometric data streams. In Proc. 20th Annual Symposium on Computational Geometry, 2004, pp. 144–151.
3. Chan T.M. Faster core-set constructions and data-stream algorithms in fixed dimensions. Comput. Geom., 35(1–2):20–35, 2006.
4. Cormode G., Muthukrishnan S., and Rozenbaum I. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 25–36.
5. Frahling G., Indyk P., and Sohler C. Sampling in dynamic data streams and applications. In Proc. 21st Annual Symposium on Computational Geometry, 2005, 142–149.
6. Indyk P. Algorithms for dynamic geometric problems over data streams. In Proc. 41st Annual ACM Symp. on Theory of Computing, 2004, pp. 373–380.
7. Korn F., Muthukrishnan S., and Srivastava D. Reverse nearest neighbor aggregates over data streams. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, 814–825.
8. Preparata F.P. and Shamos M.I. Computational Geometry: An Introduction, 3rd edn. Springer, Berlin Hiedelberg New York, Oct. 1990.
9. Vitter J.S. Random sampling with a reservoir. ACM Trans. Math. Software, 11(1)37–57, 1985.

GEO-RBAC Model

YUE ZHANG, JAMES B. D. JOSHI
University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

LoT-RBAC

Definition

The central idea of the GEO-RBAC model is the role schema, and instances of roles, as defined below [1]:

- Role Schema: *A Role Schema is a tuple <r, ext, loc, m_{loc}>, where r is the name of the role, ext is the feature type of the role extent, loc is the feature type of the logical positions and mloc is the mapping function that maps a real position into a logical position of type loc.*
- Role Instance: *Given a role schema r_s, an instance r_i of r_s is a pair of <r, e> where r = r_s.r and e∈F, such that FT_Type (e) = r_s.ext.*

The notion of permissions, users, sessions, permission-role assignment and user-role assignment in the GEO-RBAC are similar to those in the RBAC model. The role enabling status in GEO-RBAC depends on whether the user's physical location maps the logical location specified in the role schema, as defined below:

- Enabled Roles: *Enabled session roles are defined as the function: EnabledSessionRoles: SES × RPOS→2^{RI} such that EnabledSessionRoles (s, rp) = {<r, e>∈R_I | <r, e>∈SessionRoles (s), lpos = SchemaOf(<r, e>).mloc(rp), Contrains (LocObj(e), LocObj (lpos))* = TRUE}

Finally, the control of the authorization in GEO-RBAC is defined as follows [1]:

- Authorization control function: An access request is a tuple *ar* = <*s, rp, p, o*>, where *s* is the session of the user, *rp* is the real positions of the user, *p* is permission requested, and *o* is the object of the permission. *ar* can be satisfied at position *rp* if

$$(p,o) \in \bigcup_{y \in EnabledSessionRoles(S,rp)} I\_PrmsAssignment^*(y)$$

where *I_PrmsAssignment** (*y*) represents those permissions that are directly assigned to role instance *y* or assigned to the role schema corresponding to *y*.

Key Points

The widespread deployment of location-based services and mobile applications, as well as the increased concerns for the management and sharing of geographical information in strategic applications like environmental protection and homeland security have resulted in a strong demand for fine-grained location based access control models and mechanisms. Efforts have been expended to extend existing RBAC models to develop fine-grained specification and enforcement framework for location based access control requirements.

The GEO-RBAC model, proposed by Bertino et al. [1], supports location constraints to address such location based access control needs. GEO_RBAC is based on the notion of *spatial role* that is a geographically bounded organizational function. The boundary of a role is defined as a geographical feature, such as a road, city, or hospital, and specifies the spatial extent in which the user has to be located in order to use the role. The location constraints and the authorization are integrated through enabled roles. If the user is located inside the spatial boundary of the role that has been selected (activated) during the session, the role is said to be enabled. The user can acquire all the permissions that are assigned to the enabled roles from all the roles available in his/her sessions.

As RBAC, GEO-RBAC encompasses a family of models. Core GEO-RBAC includes the basic concepts of the model, thus the notion of spatial role, role schema, real/logical position, and activated/enabled role. Hierarchical GEO-RBAC extends the conventional hierarchical RBAC model by introducing two distinct hierarchies – one over role schemas and one over role instances. Constrained GEO-RBAC supports the specification of separation of duty (SoD) constraints for spatial roles and role schemas. Since exclusive role constraints are important to support the definition and maintenance of access control policies in mobile contexts, SoD constraints are extended to account for different granularities (schema/instance level), dimension (spatial/non-spatial), and different verification time (static, dynamic at activation time, dynamic at enabling time).

Another model that addresses location based access in conjunction with temporal constraint is the Location and time-based RBAC (LoT-RBAC) model.

Cross-references

► General Based Role Based Access Control
► LoT-RBAC
► Role Based Access Control
► Temporal Access Control

Recommended Reading

1. Bertino E., Catania B., Damiani M.L., and Perlasca P. GEO-RBAC: a spatially aware RBAC. In Proc. 10th ACM Symp. on Access Control Models and Technologies, 2005, pp. 29–37.

Georeferencing

Jordan T. Hastings, Linda L. Hill
Department of Geography, University of California-Santa Barbara, Santa Barbara, CA, USA

Synonyms

Geospatial referencing; Spatial referencing

Definition

Georeferencing is the name given to the process of geospatially referencing data and information objects – datasets, text documents, maps, photographs and imagery, etc. – to their proper locations on Earth. The vast majority of such objects derive from measurements and observations phenomena that are inherently georeferenceable, because humans are largely confined in their activities to the near-surface of the Earth and "Almost everything that happens, happens somewhere" [3].

Georeferencing can be accomplished in two main ways [2]: formally, by assigning geospatial coordinates directly to data and information objects; and informally, by relating such objects to one or more pre-existing ones for which georeferences have already been established. In everyday life, the latter approach predominates, using the mechanism of *place*. A geographic place is a real-world location, perhaps vague, with a recognizable name and type, and optionally other attributes, e.g., "Northern California," Golden Gate Park. A *gazetteer* is a dictionary of such places, which is essential for translating back and forth between informal and formal georeferences.

Numerous problems and subtleties attend the formal, or direct, georeferencing process. The Earth is not a sphere, in fact, but a lumpy body that is cannot be exactly described geometrically. Indeed, multiple definitions exist for the fundamental geospatial coordinates – longitude, latitude, and surface elevation – all of which depend on a definition of the Earth's center. Map projections further confound matters. No projection of the globe onto a flat sheet, to create a map, can simultaneously preserve both area and shape, even locally; thus, all projections introduce distortions, which can be severe. In addition, most projections involve transcendental mathematical functions, which are only approximated by computers.

Other problems apply to informal georeferencing using named places, e.g., "Fisherman's Wharf" or "5 miles south of San Francisco." Except for officially designated and defined administrative regions, geographic places are inherently subjective, based in geospatial cognition and mediated by linguistic, historical, and social conventions. Places are therefore approximate in location, shape and size, and further subject to various typing schemes. What are the distinctions among "metropolis," "city," and "town," for example; or between "lake" and "pond"? A single name may apply to many places – for example, there are many Lake Genevas in addition to the famous water body in Switzerland; some are lakes, in fact, others are cities. Also, a single place can have multiple names and types – for example, New York City is colloquially known as "The Big Apple."

Historical Background

Geospatial referencing originated in marine navigation, where it is essentially a 2-D exercise, using knowledge of celestial mechanics and time. Latitude can be determined from a sighting of the sun at local noon, knowing the day of the year. Determining longitude requires a reliable shipboard time-of-day clock [5] as well. Positional accuracies on the order of kilometers, i.e., within range of sight, are typical.

On land, longitude and latitude are relatively unimportant in people's everyday affairs, supplanted by linear measurements of distance and direction along constrained routes, e.g., riverways and roadways. Even these measurements are subordinate in many cases to the observation of salient places, also known as landmarks, along the routes: topographic features, human settlements, water bodies, etc. Again, relatively low-technology tools suffice: a wheel to measure distance, a compass to indicate direction, and optionally a barometer to estimate elevation. Positional accuracies on the order of ∼100 m are suitable for most purposes.

In space, the position of every satellite must be meticulously tracked. A constellation of ∼30 satellites has been deployed specifically as a global positioning system (GPS). Several of these satellites are visible from most points on Earth at all times, although the particular satellites come and go throughout the day. GPS satellites continuously broadcast their positions on special radio frequencies. A companion GPS receiver on Earth that picks up the signal from three or more of these satellites can back-calculate its own horizontal position by trilateralization (a variant of triangulation); with four or more satellites, vertical position can be determined as well. From multiple "fixes" on position over time, the GPS receiver also can work out its speed and direction of movement, if any. Overall, this system provides positional accuracies of ~5 m horizontally and ~20 m vertically, which is more than sufficient for most georeferencing purposes.

Foundations

Technical Background

The Earth is a globe, but only approximately spherical, with an equatorial radius of 6,378,137.0 m and a polar radius of 6,356,752.3 m, by modern measurement (WGS84). This ellipsoidal description is also approximate, as the Northern hemisphere is squatter than the Southern hemisphere; in addition, both hemispheres are incised by oceans and elevated by continents. The exact 3-D "figure" of the Earth defies analytic mathematical description, in fact. For precise geodetic work, affecting rocket and satellite trajectories among other things, a gridded model of the Earth's shape is used. However, for most mapping applications, an ellipsoidal approximation is satisfactory [1].

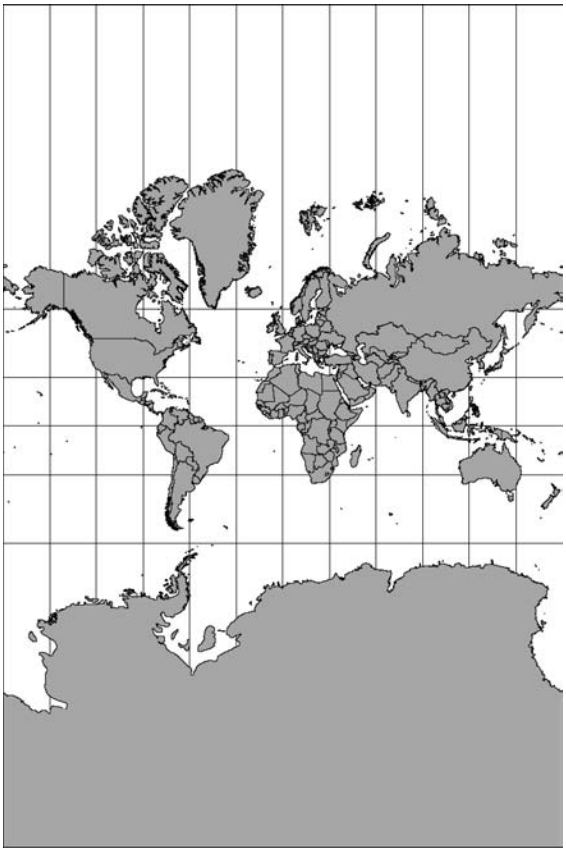
A *datum* is a set of parameters that define an approximate figure for the Earth, typically as an ellipsoid. The horizontal datum defines longitude and latitude with regard to this figure, and the vertical datum defines elevation in relation to an average sea level on it. Until the satellite era, the Clarke 1866 datum was sufficient and stable; since then a progression of datums (Table 1) have been adopted. A *map projection* depicts the globular figure of the Earth on a flat sheet of paper. In the projection process, lines of longitude and latitude (circles on the Earth) become other shapes on paper; inevitably, distortions are introduced. For example, the well known Mercator projection (Figure 1) depicts both longitudes and latitudes as straight lines, resulting in extreme distortion at the poles. No projection preserves both shape and area throughout the map, and many projections preserve neither (but still are useful).

Digital geospatial data representations are numerous. Three well established categories of such representation are: *vectors*, discrete shapes, applying to object-like phenomena; *rasters*, regular grids, best suited to field-like phenomena; and TINS (triangulated irregular networks), for 2½-D surfaces. Vector data further subdivides into points, lines, and polygons. With the exception of points and the simplest polygons – triangles and rectangles – none of these representations fits easily into a single relational database table; rather, table(s) containing open-ended "blob" (binary large object) structures, or a group of tables, are frequently used. A wide variety of proprietary structures exist for geospatial data transcribed to files. The Geographic Markup Language (GML) [4] is an application of XML designed to remedy this complexity by encoding

Georeferencing. Table 1. Selected world datums

Name	Semi-major axis (m)	Semi-minor axis (m)	Flattening ratio
Clarke 1866 (NAD27)	6378206.4	6356583.8	294.979
GCS 80 (NAD83)	6378137.0	6356752.3	298.257
WGS 84[a]	6378137.0	6356752.3	298.257
GCS 87 (Europe)	6378388.0	6356911.9	297.000

[a] Differs imperceptibly from GCS80, for most purposes



Georeferencing. Figure 1. Mercator projection of the world, with 30-degree graticule.

both vectors and rasters (but not TINS), including their associated attributes, in a portable text format.

Georeferencing Techniques

Georeferencing spans a variety of techniques, determined in part by the content and format of the underlying data and information objects. Some objects must be formally georeferenced with longitude and latitude coordinates. Aerial photographs and satellite imagery, for example, are usually located and oriented geospatially by the coordinates of their center and/or corner points. By contrast, text documents are typically georeferenced either by assigned geographic subject headings or by place names (in titles, indexes, and the text itself). Subsequently, using a gazetteer, these informal place references can be translated to coordinates. Hardcopy maps usually can be georeferenced in both ways, because they are drawn in a coordinate framework and also have place names embedded in titles, marginalia, map features, etc. Tabular materials represent another special case, because they can contain both place names and coordinates explicitly. Photographs, works of art, biological specimens, and other such objects are most often georeferenced informally in accompanying notes, although some digital cameras are now equipped with GPS and capable of recording formal coordinates directly with the image.

Following is a synopsis of common georeferencing techniques.

Datasets Scientific and technical data are frequently presented in tables, recorded in spreadsheets or databases. Such tables may include georeferences as coordinates, place names, or both. For statistical data, place codes may appear in lieu of place names for administrative or other formally defined areas, such as census tracts or postal zips. Data in tables may be georeferenced formally, informally using gazetteers, or both.

Documents Textual materials commonly contain geospatial references as place names, which may serve to locate the subject matter more or less precisely, viz. "Washington Mall" versus "apex of Washington Monument". Place names also may appear adjectively ("Washington Redskins' stadium"), prepositionally ("suburbs of Washington"), or they may be the subject itself ("Washington, DC"). A single text document may contain many place names, some relevant to the subject matter, some not, viz. "unlike its neighboring states of Maryland and Virginia, Washington, DC is a federal district". A technique called geoparsing is used to recognize salient place references in text documents and to associate coordinates with them via gazetteers.

Maps Maps are, by definition, depictions of geospatial phenomena in a coordinate framework, i.e., graphical displays of geospatial data. A significant portion of the geography curriculum at all levels concerns locating coordinate pairs on maps and conversely extracting the coordinate locations of mapped points. Linear and areal features are typically represented by sequences of points, which can be intricate. A GIS (below) is, to first approximation, a map in digital form. Map content is inherently georeferenced, but the maps themselves often require external, informal georeferencing.

Photographs It is useful to distinguish between photographs taken *in* the environment – "snapshots" – and photographs taken *of* the environment, typically from an aerial platform such as an airplane. The former may be georeferenced, either formally or informally, as objects at mapped points; the later cover a quasi-rectangular area on a map, indicated by a sequence of points. With aerial photographs, comes the additional issue of othro-rectification: removing the dilation or "bloom" in portions of the photograph except at the single point (if any) focused straight down.

Imagery Satellite imagery is an electronic rather than film product – so-called "digital numbers" (DNs), representing the brightness of a scene in a gridded pattern of picture elements (pixels). Images are taken from sufficient distance above the Earth that dilation issues are small, but not negligible. The satellite's orbit coupled with the Earth's rotation typically results in an image swath that tracks diagonally across the land surface, producing trapezoidal pixels. For ease of use, the trapezoids are usually interpolated back to square pixels during post-processing of the satellite data, in conjunction with ortho-rectification. Georeferencing then proceeds as for aerial photographs.

Key Applications

Georeferencing makes data and information objects more useful for many purposes. In a geographic information system (GIS), for example, vector objects, raster fields and imagery are presented in layers. When such layers are georegistered to a common standard, their content appears superimposed – as on an electronic light table. Users are frequently surprised by even slight mis-registrations of the layers. Hence, the intrinsic accuracy of geospatial measurements relating to the layers also must be considered.

Geographic information retrieval (GIR) is an extension of textual information retrieval (IR) that compares coordinate-based queries to georeferenced data and information objects to find the best matches. In a geospatially-enabled digital library, for example, the query "find information about Washington, DC" can be translated into coordinates for the area, from which a wide variety of materials can be retrieved, including satellite images, maps, and books. The spatial matching operations usually include containment, overlap, and proximity within a specified distance. As with GIS, issues of georeferencing accuracy as well as intrinsic errors must be considered.

A third, hybrid application, is geospatial modeling and analysis, in which GIS layers, are manipulated numerically to yield additional insights and results. When the physical laws governing the behavior of environmental phenomena are known, or hypothesized, these also can be integrated with the GIS layers to produce diagnostic and forecast models of environmental behavior. The purpose of such models, beyond scientific curiosity, is usually to inform political policy and decision-making.

Cross-references

- ▶ Gazetteers
- ▶ Geographic Information Retrieval
- ▶ Geographic Information Systems
- ▶ Spatial Operations and Map Operations

Recommended Reading

1. Clarke K.C. Getting Started with GIS, 4th edn. Prentice-Hall, Upper Saddle River, NJ, 2004.
2. Hill L.L. Georeferencing: The Geographic Associations of Information. MIT Press, Cambridge, MA, 2006.
3. Longley P.A., Goodchild M.F., Maguire D.J., and Rhind D.W. Geographical Information Systems and Science, 2nd edn. Wiley, New York, 2005.
4. Open Geospatial Consortium (OGC). Geography Markup Language (version 3.1.1).
5. Sobel D. Longitude: The True Story of a Lone Genius Who Solved the Greatest Scientific Problem of his Time. Penguin, New York, NY, 1992.

Geoscientific Information System

▶ Three-Dimensional GIS and Geological Applications

Geospatial Information System

▶ Geographic Information System

Geospatial Metadata

Bugra Gedik
IBM T.J. Watson Research Center, Hawthorne, NY, USA

Synonyms

Geographical metadata

Definition

Geospatial metadata are a type of metadata used to describe geospatial data. Data described by geospatial metadata relates to objects that have an explicit or implicit geographic extent or position on the surface of the Globe. Geospatial metadata can be used to create metadata catalogues or directories that describe geographic features of data stored in any environment, ranging from data stored in geographic information systems (GIS) to simple documents, datasets, images, or even services. Geospatial metadata captures the basic characteristics of geospatial data and represents the what, the when, the where, and the who of the geospatial data. A typical geospatial metadata record includes catalog elements such as title, abstract, and publication data; geographic elements such as geographic extent and projection information; and database elements such as attribute label definitions and attribute domain values.

Historical Background

The interest in geospatial metadata grew steadily over the 1980s and 1990s, and as a result several national and international organizations, as well as communities of practice, started to develop their own formats and conventions for collecting and organizing geospatial metadata. For instance, United States National

Aeronautics and Space Administration's (NASA) Directory Interchange Format (DIF) [2] was developed in 1987, and formally approved for adoption in 1988. A similar geospatial metadata development effort was undertaken by the United States Federal Geographic Data Committee (FGDC) throughout 1992 to 1994 [5], which led to FGDC's release of Content Standard for Digital Geospatial Metadata (CSDGM) standard in 1998 [3]. Similarly, in 1996 Australia and New Zealand Land Information Council (ANZLIC) released its metadata guidelines [1]. The emerging need for consolidating various formats and standards developed by different communities of practice was addressed by ISO/TC 211 over the years 1999 to 2002. Consequently, ISO 19115 standard was released in 2003 under the title "Geographic Information – Metadata." Following the release of the ISO 19115 standard, national and international organizations, as well as communities of practice have started the process of fitting their previously adopted metadata standards as profiles or recommended subsets of the ISO 19115 standard, via formal extensions to it. The increasing popularity and adoption of the Extensible Markup Language (XML) for sharing data across different information systems over the Internet led to the development of mechanisms for exchanging geographic metadata on the Web during the 1990s. Consequently, the Geography Markup Language (GML), an XML grammar for expressing geospatial features and metadata, was released in 2004 by the Open Geospatial Consortium (OGC). With the growth of the Semantic Web in the 2000s, several organizations have started to develop ontologies for representing semantic geospatial metadata. Some examples include the Hydrology and Administrative Geography ontologies [4] developed by the Ordnance Survey in the United Kingdom.

Foundations

Digital geospatial data are often used to create a model of the real world geographical objects for use in computer analysis and digital display of geospatial information. Hence, it is an abstraction of the real world geographical objects that are being described. This abstraction often involves various approximations, simplifications, and exclusions. Given that this digital representation is seldom perfect or complete, the assumptions and limitations affecting its accuracy must be fully documented to ensure that such data are not misused. Geospatial metadata allows a data producer to describe a geospatial dataset fully, so that the users can understand the assumptions and limitations associated with the dataset and evaluate its applicability for their intended use. In summary, geospatial metadata provides data producers with appropriate information to characterize their geospatial data, facilitates the organization and management of geospatial data, enables users to apply geospatial data in the most efficient way by knowing its basic characteristics, facilitates data discovery, retrieval and reuse, and enables users to determine whether geospatial data at hand will be of use to them.

Geospatial metadata consists of a schema required for describing geographic information and services. It provides information about the identification, quality, spatial and temporal extent, spatial reference, and distribution of geospatial data. The set of geospatial metadata elements is often large and typically only a subset of the full number of elements is used. For instance, ISO 19115 standard includes a list of core metadata elements – a minimum set of basic elements required to identify a dataset for cataloging purposes. These core metadata elements answer the following questions:

- Does a dataset on a specific topic exist? ("what" aspect)
- For a specific place? ("where" aspect)
- For a specific date or period? ("when" aspect)
- With a point of contact to learn more about the dataset? ("who" aspect)

GML: The Geography Markup Language

The Geography Markup Language (GML) is an XML grammar used to express geospatial features. GML serves as a modeling language for geographic systems as well as an open interchange format for geographic transactions on the Internet. Originally developed by the Open Geospatial Consortium (OGC), it is later adopted as ISO 19136 standard in 2007.

Geospatial Metadata Catalog

A geospatial metadata catalog is a collection of geospatial metadata entries that are managed together. A geospatial metadata catalog is often accompanied by a catalog service, which responds to requests for metadata in the catalog and help browse or search geospatial data described by the collection of geospatial metadata managed in the catalog. In summary, the main goal of a geospatial catalog is to support a wide range of users in discovering relevant geospatial information from heterogeneous and distributed repositories of geospatial data.

Cross-references

- ▶ Geographic Information System
- ▶ Meta Data Management System

Recommended Reading

1. ANZLIC Metadata Working Group. ANZLIC Metadata Guidelines: Core metadata elements for geographic data in Australia and New Zealand, Version 2. http://www.anzlic.org.au/get/2358011755.
2. Major G. and Olsen L. A short history of the DIF. On GCMD website, http://gcmd.nasa.gov/User/difguide/.
3. Metadata Ad Hoc Working Group Federal Geographic Data Committee. FGDC website, http://www.fgdc.gov/metadata/csdgm/.
4. Ordnance – National Mapping Agency of Great Britain. Ordnance Survey Ontologies. Ordnance Website, http://www.ordnancesurvey.co.uk/oswebsite/ontology/.
5. Wolfe R. MIT Libraries website, http://libraries.mit.edu/guides/subjects/metadata/standards/fgdc.html.

Geospatial Referencing

▶ Georeferencing

Geo-Targeted Web Search

Torsten Suel
Yahoo! Research, Sunnyvale, CA, USA

Synonyms

Geographic web search; Local web search

Definition

Geo-Targeted Web Search (Geo Search) describes a set of techniques used in web search engines that aim to return more relevant results to users by exploiting geographic knowledge. Most commonly, a Geo Search Engine (a search engine using Geo Search) tries to infer a geographic area of interest for a user query by considering the query itself, the current location of the user, or the search history of the user. The Geo Search Engine then tries to return documents that are relevant to the query search terms and that also match the geographic area of interest of the query, as determined through analysis of the web page itself, any geographic meta tags (geo tags), link or site structure, or other data. As an example, a Geo Search Engine may return to a mobile user a list of restaurants close to the user's current location given a query "restaurants." Or a user may explicitly ask for information about tourism attractions in Florida, by adding a keyword such as "florida" or pointing to a map. Geo Search techniques can be integrated into standard broad-based or specialized search engines, or deployed as separate search facilities with special interfaces that allow more convenient query input or result display.

Historical Background

Geo Search developed over the last decade as a response to several challenges faced by search engines. First, given the size of the web, there are often too many potentially relevant results, and geography can provide a natural way of filtering such results for those the user really wants. Second, much of commerce is still local in nature, and Geo Web Search can provide a platform for supporting local advertising models on the web. Third, it was found that treating geographic terms in queries just as normal terms does not give the best possible results, and that some amount of geographic domain knowledge should be added to engines to improve results. For example, by simply matching on the term "manhattan" in a query, one might miss out on results referring to New York in general or to a particular neighborhood in Manhattan, or results about Brooklyn that are close enough to be of interest. As a result, there have been significant efforts by major search engine companies and start-ups to develop Geo Search Engines, and there has also been increasing interest in the academic community.

Even before the emergence of the web, researchers in Information Retrieval studied how to exploit geographic information embedded in text documents for better text search and analysis; see Larson [10] for earlier work and Jones and Purves [8] for a recent overview of the slightly more general area of *Geographic Information Retrieval* (GIR). This article focuses on geographic web search technology, which currently dominates the area of GIR in terms of number of users and total economic investments, but there are of course important applications outside the web domain. Initial work on Geo Search on the web appears in [5,6,13], and in recent years a significant amount of research has addressed this new challenge. A lot of the work has studied the problem of reliably identifying geographic references in documents in the presence of noise and ambiguity [1,11–13]; for example, many common place names (e.g., Washington) can also refer to persons, hotels, or other entities, while others can refer to several different places (e.g., Paris, Texas vs. Paris, France). Recent work has also focused on indexing and query processing techniques for Geo Search [2,15,17], and on analyzing and studying user queries in order to better understand the types of geographic search needs that real users have [7,9,14,16].

Foundations

This discussion begins with an outline of the main tasks in Geo Search. As in conventional web search engines, these tasks can be divided into four groups: (i) data acquisition (i.e., web crawling), (ii) preprocessing and data analysis, (iii) index building, and (iv) processing of user queries using these index structures. Geo Search engines that are limited to a particular geographic area need to be able to efficiently crawl sites and documents related to this area; this problem is called *geo crawling*. In the preprocessing phase, the collection has to be analyzed to extract geographic information that can be used for indexing. This can include, e.g., complete addresses, phone numbers, town names, or informal references (such as "big apple" for New York). The output of this phase may be either sets of geographic terms, or sets of points, polygons, or other geometric objects. During index building, additional index structures may have to be built, e.g., spatial index structures for the geometric objects obtained from the previous phase. Finally, during query processing, queries and their contexts are analyzed to determine if the user might be interested in a particular geographic area, and then the index structures is used to find relevant documents.

Geo Parsing and Geo Coding

A significant amount of work has focused on (ii), the analysis of documents for geographic information [1,11–13]. In particular, *geo parsing* is the process of identifying likely geographic references (such as town names, addresses, etc.) in the document text or meta data. Most commonly, this is performed by matching terms in the documents with databases of geographic names (gazetteers) obtained from outside sources, such as the GNIS database made available by the US Geological Survey for the US. If databases of business listings (yellow pages) are available, these can also be used to extract additional geographic references. In addition, techniques for named entity recognition can be applied. When databases of geographic names are available, the problem may appear simple but is in fact not so. Consider the many ways in which city or state names may appear as part of names of persons, businesses, foods, or many other items. Moreover, Washington can be a city or a state, LA can refer to Los Angeles or Louisiana, and there are dozens of towns named Lexington. These problems are referred to as geo/non-geo ambiguity and geo/geo ambiguity, respectively. In addition, it can be desirable to also detect informal place names (e.g., "big apple" for New York).

An alternative proposal is to introduce *geo tags* that allow authors to attach appropriate geographic terms or coordinates to pages [3] or other objects such as images or business listings or reviews, or allow communities of users to supply tags. This avoids many of the challenges of automatic extraction, but potential problems are lack of participation and possible manipulation.

Once the geographic terms have been extracted, there are two main approaches for further processing. One approach performs *geo coding* to map the geographic references into longitude-latitude coordinates, and then uses spatial or geometric algorithms and data structures for further processing. Alternatively, the geographic terms can be kept in textual form and related to a geographic ontology, in the simplest case just a hierarchical grouping of geographic terms representing simple spatial facts (e.g., that San Jose is in Santa Clara County which is in California), that is used for subsequent processing.

The geo coding approach is now discussed in more detail. In this step, the goal is to infer for each document its *geographic focus* [13], which is the area that the document relates to. The estimate of this area will be stored as a spatial data structure called the document's *footprint*. To compute the footprint of a document, any references to countries, cities, counties, or maybe businesses in the document are first translated into coordinates, using coordinate data available as part of many gazetteers. The output is a set of points, rectangles, polygons, or other spatial objects, possibly with appropriate weights or amplitudes. For example, a business may be mapped to a precise coordinate, while a city could be modeled by a circle or rectangle with some diameter, or a polygon approximating city boundaries. Weights might model the importance of a geographic term in the document or the extraction confidence for this term. In general, the geographic footprint of a document might assign an amplitude to every location on a map, with a value of zero for areas that are not relevant to the document, and very high values for areas that are referenced multiple times in the document. Efficient processing will usually require approximation of this data by a small number of polygons or bounding rectangles.

Note that both geo parsing and geo coding could potentially be improved by using site and link structure [12,13], such that pages with ambiguous or missing geographic references are analyzed using information on neighboring pages. It may also be useful to assign footprints on a per-site basis. Finally, note that the geographic focus of a page is related but not identical to the *geographic scope* introduced in [5], which is the geographic area of the readership addressed by a page. For example, a page about travel to Paris in a local newspaper or site in Seattle might have a geographic focus around Paris, but a geographic scope around Seattle.

Indexing and Query Execution

After analyzing the documents for geographic references, it is necessary to build appropriate index structures that allow efficient query processing. Recall that in the analysis phase there were two main approaches, one that keeps extracted geographic references as terms that can be related to some spatial ontology or other knowledge base, and one that translates the extracted geographic terms into a spatial footprint structure based on polygons or rectangles. The former approach then uses textual index structures and techniques such as query expansion on the geographic terms in the user's query [4] in order to find the best results.

Lot of recent work, e.g., [2,15,17], follows the latter approach, which is now discussed in more detail. Assume that a query to a geo search engine consists of two parts, a set of terms as in a standard engine, and a *query footprint* in the same basic format as the document footprints. Such a query footprint could be obtained from the user in a number of ways, e.g., by having the user input a geographic term or point and click in a map, or by using the user's current location or past queries to derive a likely area of interest. Then a possible definition of query processing in geo search engines is as follows:

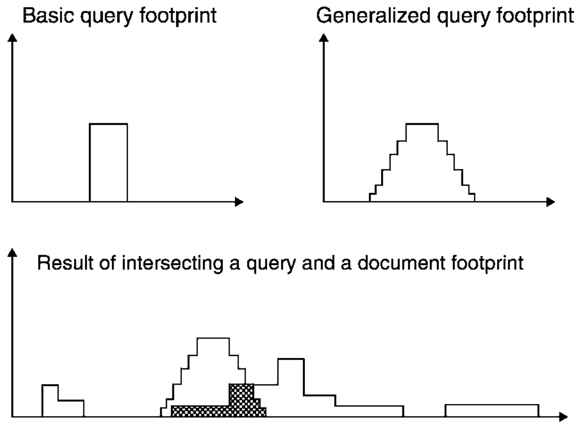
Definition: Given a collection of documents, each consisting of a document footprint and a set of terms, and a query consisting of a query footprint and a set of terms, the top-k query processing problem in geographic search engines is to return k documents such that:

- The footprint of each returned document has a nonempty intersection with the query footprint, i.e., there exists a location where both footprints are non-zero.
- Each returned document contains all terms in the query, or alternatively satisfies some weaker Boolean filter on the query terms (e.g., an OR).
- The returned documents are the k highest scoring documents according to a ranking function of the form $h(s_t,s_g)$ where s_t is a term-based score computed on the query and document terms, s_g is a geographic score obtained from the query and document footprints, and h is a monotone function for aggregating the two scores.

Some of the above assumptions can be relaxed, for example by allowing s_t to also contain scores based on link analysis or page visit frequency. One interesting choice for s_g is a dot product over all locations of the two footprints (or a corresponding integral in the continuous case). As shown in Fig. 1, by properly preprocessing the footprints it is possible to achieve objectives such as scoring documents higher if they are closer to the center of the query footprint or if they overlap with locations that are of particular interest to the user.

Thus, for fast query processing, it is necessary to have index structures that support execution of the spatial and textual filtering conditions, and an efficient way to score the remaining documents. In typical web collections, the textual data to be indexed is larger than the extracted spatial data, and thus efficient text index structures, such as inverted lists, are crucial. The size of the spatial data depends on the format used for the footprints. If the average footprint consists only of a few bounding rectangles, then this data may be only a few percent of the text data and thus may fit in main memory. If a footprint contains a more precise representation involving polygons or amplitudes, then the spatial data may only be moderately smaller than the text data.

In any case, to perform query processing, both textual and spatial index structures are required, and the challenge is to integrate these efficiently. A number of algorithms for this problem were first proposed in [15]. Two recent optimized approaches [2,17]


Basic query footprint
Generalized query footprint
Result of intersecting a query and a document footprint

Geo-Targeted Web Search. Figure 1. [12] An illustration of query and document footprints in a single spatial dimension. The top shows a query footprint with a distance threshold (*left*), and a footprint for a query that gives a lower score for documents that are farther away from the center (*right*). The bottom shows an intersection between the query footprint and a document footprint.

address this issue by combining inverted indexes with spatial structures such as R*-trees and space-filling curves. The approach in [17] assumes that each footprint is a set of rectangles; this results in a fairly small amount of spatial data and very fast query processing as most spatial data will be in memory. Work in [2] assumes that footprints are approximated by a set of bounding rectangles that is used during the filtering phase, but that a more precise representation of the footprints is also stored and must be fetched in order to compute precise values for s_g. The precise footprints are treated as binary objects (BLOBs) whose detailed structure is not important as long as a score can be computed from them; they might for example contain amplitudes or additional context about the geographic terms that were parsed from the documents. Compared to [17], this approach results in more spatial data that needs to be accessed, and thus more disk transfers, but it is more general in terms of the types of ranking functions that can be supported. However, it is not clear to what degree this generality is actually needed, and the approach in [17] may suffice in many practical scenarios.

Geographic Search Queries

In order to provide useful geographic search services, it is necessary to study how users search on the web. Several recent studies have looked at how to automatically identify queries that have a geographic intent and could thus benefit from geographic search technology [7,9,16]. Automatic identification allows a proper query footprint to be assigned to such queries. It may also allow better processing of queries that have no explicit geographic terms. For example, when searching from mobile devices or sometimes also from desktops, users may prefer results close to their current location without explicitly specifying so. Other researchers have studied the basic properties of such queries [14]. It is estimated that about 10% to 15% of queries submitted to standard search engines contain geographic terms. Work in [9] looks at how users modify the geographic terms in query sessions in order to get better results.

Key Applications

All the major search engine companies have developed and deployed geographic search technology as part of their offerings. In particular, these engines provide local search options that allow users to search for businesses and organizations using keywords, geographic terms, and map-based interfaces. In addition, geographic search technology is also used in their standard web search engines to provide better results for queries that are likely to have a geographic intent. Closely related and of particular commercial interest to the search engines are techniques for local advertising on the web.

Other applications on the web are in the automatic compilation of local information for city or neighborhood portals, and in providing better interfaces and search for real estate, car trading, or classified sites where users tend to search in a certain geographic area. Finally, although this discussion has focused on the case of the web, Geographic Information Retrieval has many applications on other textual collections where geographic search, browsing, and analysis are useful.

Future Directions

There is continuing research on many aspects of geographic web search, which is still in its infancy. This includes work on better extraction of geographic terms from documents and queries using techniques from Natural Language Processing and Information Retrieval. Ideally, techniques should be general enough to be applicable across languages, and it would be desirable to also identify implicit geographic intent in queries and informal place names.

Further studies of search logs and user behavior are needed to better understand user intentions and search strategies, particularly for queries that go beyond the types of searches for local businesses that are already handled fairly well by current local search services. Growing interest in technologies for local advertising on the web is also expected.

An intriguing future direction in geographic search involves online virtual worlds, such as Second Life as well as models of the real world such as Google Earth. As such models become more popular and users become more comfortable in navigating them, significant textual content is expected to migrate to or be mapped into these spaces, resulting in yet unexplored research problems related to geographic web search.

Cross-references

▶ Geographic Information System
▶ Information Retrieval
▶ Inverted Index
▶ Named Entity Extraction
▶ Rtree
▶ Search Log Analysis
▶ Space-Filling Curves
▶ Spatial Network Databases
▶ Web Search Engines

Recommended Reading

1. Amitay E., Har'El N., Sivan R., and Soffer A. Web-a-where: geotagging web content. In Proc. 27th ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 273–280.
2. Chen Y., Suel T., and Markowetz A. Efficient query processing in geographic web search engines. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 277–288.
3. Daviel A. Geographic registration of HTML documents, Apr 2001.
4. Delboni T., Borges K., and Laender A. Geographic Web search based on positioning expressions. In Proc. Workshop on Geographic Information Retrieval, 2005, pp. 61–64.
5. Ding J., Gravano L., and Shivakumar N. Computing geographical scopes of web resources. In Proc. 28th Int. Conf. on Very Large Data Bases, 2000, pp. 545–556.
6. Egenhofer M. Toward the semantic geospatial web. In Proc. 10th ACM Int. Conf. on Advances in Geographic Information Systems, 2002, pp. 1–4.
7. Gravano L., Hatzivassiloglou V., and Lichtenstein R. Categorizing web queries according to geographical locality. In Proc. 12th ACM International Conference on Information and Knowledge Management, 2003, pp. 325–333.
8. Jones C. and Purves R. Geographic information retrieval. Int. J. Geo-Graph. Inform. Sci., 22(3):219–228, March 2008.
9. Jones R., Zhang V., Rey B., Jhala P., and Stipp E. Geographic intention and modification in web search. Int. J. Geograph. Inform. Sci., 22(3):229–246, 2008.
10. Larson R. Geographic information retrieval and spatial browsing. In GIS and Libraries: Patrons, Maps and Spatial Information. Linda Smith and Myke Gluck (eds.), 1996, pp. 81–124.
11. Leidner J. Toponym resolution in text: Which Sheffield is it? In Proc. 27th ACM SIGIR Int. Conf. on Research and Development in Information Retrieval, 2004, pp. 602–602.
12. Markowetz A., Chen Y., Suel T., Long X., and Seeger B. Design and implementation of a geographic search engine. In Proc. 8th Int. Workshop on the Web and Databases, 2005.
13. McCurley K. Geospatial mapping and navigation of the web. In Proc. 10th Int. World Wide Web Conference, 2001, pp. 221–229.
14. Sanderson T. and Kohler J. Analyzing geographic queries. In Proc. of the Workshop on Geographic Information Retrieval, 2005.
15. Vaid S., Jones C., Joho H., and Sanderson M. Spatio-textual indexing for geographical search on the web. In Proc. Nineth Int. Symp. on Spatial and Temporal Databases, 2005.
16. Wang L., Wang C., Xie X., Forman J., Lu Y., Ma W., and Li Y. Detecting dominant locations from search queries. In Proc. 28th ACM SIGIR Int. Conf. on Research and Development in Information Retrieval, 2005.
17. Zhou Y., Xie X., Wang C., Gong Y., and Ma W. Hybrid index structures for location-based web search. In Proc. Int. Conf. on Information and Knowledge Management, 2005, pp. 155–162.

GIS

▶ Semantic Modeling for Geographic Information Systems

GIST

▶ Generalized Search Tree (GiST)

GiST

▶ Generalized Search Tree (GiST)

Global Query Optimization

▶ Multi-Query Optimization

Glyphs

▶ Iconic Displays

GMAP

Steven M. Beitzel[1], Eric C. Jensen[2],
Ophir Frieder[3]
[1]Telcordia Technologies, Piscataway, NJ, USA
[2]Twitter, Inc., San Fransisco, CA, USA
[3]Georgetown University, Washington, DC, USA

Synonyms

Geometric mean average precision

Definition

The Geometric Mean Average Precision (GMAP) is the geometric mean of the average precision values for an information retrieval system over a set of *n* query topics. GMAP is expressed as follows (from [1]):

$$\mathrm{GMAP} = \sqrt[n]{\prod_n AP_n}$$

where *AP* represents the Average Precision value for a given topic from the evaluation set of *n* topics. Please refer to the entry on *Average Precision* for the complete definition of *AP*. An alternate calculation method expresses GMAP as an arithmetic mean of logs (also from [1]):

$$\mathrm{GMAP} = \exp \frac{1}{n} \sum_n \log AP_n$$

Key Points

The Geometric Mean Average Precision evaluation metric was first introduced in the Robust track at the 2004 NIST Text Retrieval Conference (TREC). A stated goal of the robust track was to examine and evaluate information retrieval systems with a view towards achieving consistently good performance over all query topics in the evaluation set. The geometric mean was considered for this purpose because it tends to better illustrate large relative improvements in the scores of poorly performing topics between a set of runs while suppressing the effect of minor fluctuations in the scores for topics that achieve good performance. By contrast, the traditional Mean Average Precision metric, which uses the arithmetic mean of average precision scores, gives equal weight to absolute changes in per-topic scores, regardless of the relative size of the change.

As an example, consider a run with five topics, each having the following Average Precision Scores:

	Topic #1	Topic #2	Topic #3	Topic #4	Topic #5
Run 1 AP	0.05	0.10	0.50	0.50	0.75
Run 2 AP	0.10	0.30	0.45	0.45	0.60

In this example, Run 1 and Run 2 have the same Mean Average Precision (0.38), whereas Run 1 has a GMAP of ∼0.25 and Run 2 has a GMAP of ∼0.33. If the evaluator is interested in a measure of consistency and collective performance across all topics, GMAP is a good choice for the evaluation metric.

Cross-references

- ▶ Average Precision
- ▶ Effectiveness Involving Multiple Queries
- ▶ Mean Average Precision

Recommended Reading

1. National Institute of Standards and Technology. TREC-2004 common evaluation measures.

Google Bombing

▶ Web Spam Detection

Grammar Induction

▶ Grammar Inference

Grammar Inference

Matthew Young-Lai
Sybase iAnywhere, Waterloo, ON, Canada

Synonyms

Automata induction; Grammatical inference; Grammatical induction; Grammar induction; Automatic induction; Automatic language induction

Definition

Grammar inference is the task of learning grammars or languages from training data. It is a type of inductive inference, the name given to learning techniques that try to guess general rules from examples.

The basic problem is to find a grammar consistent with a training set of positive examples. Usually, the target language is infinite, while the training set is finite. Some work assumes that both positive and negative examples are available, but this is not true in most real applications. Sometimes probability information is attached to each example. In this case, it is possible to learn a probability distribution for the strings in the language in addition to the grammar. This is sometimes called *stochastic* grammar inference.

A grammar inference algorithm must target a particular grammar representation. More expressive representations are more difficult and expensive to learn. For example, context-free grammars are more expressive than regular languages, but much harder to learn. The target representation for stochastic grammar inference can be viewed as separate structure and probability components. The structure component is a standard grammar. The probability component defines the probabilities associated with individual strings. One way to define this association is to assign a probability to every production of the grammar, or for regular grammars, to every transition of the finite automaton. The probability of a string is then the product of the production or transition probabilities used in its generation.

As with all learning problems, a solution is not necessarily acceptable just because it is consistent with the training set. Infinitely many grammars are consistent with any finite sample. Inference must construct a possible grammar, and also choose from among the possibilities. Usually the grammar must generalize outside the available examples in some useful way. What is useful varies with the application. Some applications require a result that a human can interpret. Others require a grammar that assigns accurate probability predictions to strings.

Historical Background

Grammar inference has been studied by researchers in many fields including information theory, formal languages, automata theory, language acquisition, computational linguistics, machine learning, pattern recognition, computational learning theory, neural networks, and data compression. Several conferences and workshops focus specifically on the subject. One example is the International Colloquium on Grammatical Inference (ICGI) which has been held in 1993, 1994, 1996, and every 2 years through 2006 so far. There have also been associated language learning competitions such as the Abbadingo competition which focused on learning deterministic finite automata and the Omphalos competition that focused on learning context free languages.

The first studies of grammar inference date back to the 1960s. One formalization is called *language identification in the limit* [7]. This assumes two sets of strings: $R+$ is the set of positive examples, and $R-$ is the set of negative examples. A language is said to be *identifiable in the limit* if it can be learned by *some method* for *sufficiently large* $R+$ and $R-$. Put another way, adding new examples to $R+$ and $R-$ must only produce a finite number of changes to the hypothesized model.

There are two decidability results in this formalization. The first is negative and says that no infinite language can be identified in the limit from only positive examples. This is intuitively a consequence of over-generalization since adding more positive examples can never imply that a string mistakenly included in the model should subsequently be removed. In fact, a consistent generalization of any set of positive examples is a language containing all finite strings constructed from the alphabet.

The second decidability result is positive and states that any member of an enumerable class of recursive languages (context-sensitive and below) is identifiable in the limit given both positive and negative data. However, this result says nothing about how large a sample is needed and therefore has limited application. Even if negative examples are available, which is often not the case, the learning problem may be intractable. For example, finding the smallest finite automaton compatible with given positive and negative example sets is NP-hard [3,8]. A polynomial time algorithm can construct a *non-minimal* solution to this problem in the limit [11]. However, the problem remains that there is no bound on the size of the required training data.

Within the standard Chomsky hierarchy from automata theory [10], grammar inference research has focused mainly on regular and context free grammars. Restricted grammars, referred to as *characterizable subclasses*, are also commonly targeted. For example, *k*-reversible languages [4], generalize from 0-reversible languages which are those generated by finite automata that remain deterministic if all of their arcs are reversed. The main justification for using restricted language classes is to reduce the time complexity of the inference process. The defining restrictions are not themselves relevant to real applications.

Much work has been done on stochastic grammar inference because of its applicability to real applications. Early work in information theory used probability information to discover rules. Shannon proposed that language learning can be seen as a problem of estimating probabilities of sequences of n-grams as n approaches infinity. Well known techniques such as the Baum-Welch algorithm [5] for learning hidden Markov models can be considered stochastic grammar inference since hidden Markov models are equivalent to stochastic finite automata. Many learning algorithms for data compression and neural networks perform stochastic grammar inference, although not always using representations that map to standard grammars.

Foundations

A common grammar inference paradigm starts with a model constructed to accept the finite language composed of exactly the strings in the training set. The inference process then transforms this *de-facto* model by applying generalization operations. This is analogous to a natural learning process that starts with a complete memory of specific examples and then combines groups of similar examples into general rules.

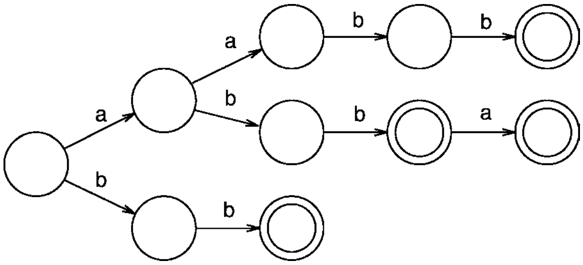
The form of the de-facto model depends on the target representation. The training set {*abb*, *bb*, *aabb*, *abba*}, for example, can be represented by the de-facto grammar in Fig. 1, the de-facto stochastic finite automaton in Fig. 2, or by appropriately chosen models in other representations.

Generalization operations can usually be viewed as merging sub-components of the model. Finite automata, for example, are generalized by merging states, thus creating new loops or paths. Grammars are generalized by merging productions. This expands the language so that it includes new strings with characteristics similar to existing strings. Operations that arbitrarily expand the language (e.g., by adding symbols to the alphabet or adding strings that have nothing in common with any of the training examples) are also possible, but not generally useful.

While the paradigm of moving from specific to more general models is common, the opposite strategy is also possible. This involves starting with a completely general model and then modifying it to describe specific rules or patterns. This is done in the previous example by starting with the finite automaton having a single state and transitions back to that state on input of *a* and *b*. If it is then observed that *b*s always occur in pairs, then a new state can be split off to express this rule.

$$\begin{array}{lcl} S & \rightarrow & abb \\ S & \rightarrow & bb \\ S & \rightarrow & aabb \\ S & \rightarrow & abba \end{array}$$

Grammar Inference. Figure 1. A de-facto grammar.



Grammar Inference. Figure 2. A de-facto stochastic finite automaton.

Generalization from a specific model and specialization from a general model are two paradigms for moving through the infinite space of all languages consistent with a given training set. Also needed is a specific control strategy that defines the order of generalization or specialization operations and when the process terminates. If a metric is available for evaluating intermediate results, then the task can be seen as a search problem, and many well known control strategies such as greedy search are applicable. However, many grammar inference algorithms are not explicitly formulated in this way. Instead, they choose the control strategy, the movement operations, or the representation in some way that is computationally convenient. The use of characterizable subclasses mentioned above is one example of this.

Stochastic grammar inference works with training data that can be viewed as a statistical sample of an unknown distribution. This means that candidate models can be evaluated in a relatively objective manner using either statistical tests or metrics based on information theory. In particular, it is possible to evaluate strings that are generated by the model but are not present in the training data. Probability information thus makes up for the lack of negative examples. Such unseen strings must be assigned low enough probability to fit the assumption that the training set is a random sample of the language generated by the model. Consider the training set $T = \{(a, 20), (aa, 10), (aaa, 5)\}$ consisting of three (*string*, *frequency*) pairs for a total frequency of 35. One possible model is $\{a^n | p(a^n) = 0.5^n\}$ which assigns a total probability of 0.125 to strings of four or more *a*s even though no such strings are present in the sample. A simple statistical test can verify that since T contains only 35 strings it could in fact have realistically been generated by the model. For example, a χ^2 test shows that the chance of getting T or any less likely sample from the model is somewhere between 0.20 and 0.50 – probably not unlikely enough to conclude that the model is a bad one. For the set $T' = \{(a, 200), (aa, 100), (aaa, 50)\}$, on the other hand, the probability is less than one in a thousand. The reason is that a random sample of 350 strings from the model should have included at least a few longer strings such as *aaaa* or *aaaaa*. Alternative evaluation criteria can be based on things other than strict probability of occurrence. Other approaches include Bayesian comparison to prior probability distributions [14] and divergence comparisons based on information theory [12].

Key Applications

The traditional application domain of grammar inference has been syntactic pattern recognition. This assumes that pattern classes can be represented by grammar models, and it is useful to learn these models automatically. Other application areas include speech and natural language processing, optical character recognition, information retrieval, gene analysis, sequence prediction, and cryptography.

In the database context, most recent work on selectivity estimation for query optimization in XML databases can be classified as stochastic grammar inference. Grammar inference also applies to data mining and knowledge discovery. Stochastic grammar inference is a key part of many compression techniques where better models directly correspond to better compression ratios.

For structured document collections, grammars are a natural representation to use as a database schema. This is also true of semi-structured and XML databases. Grammar inference is applicable to such data if it has no explicitly-created schema and there has been some work on grammar inference in this context. Fankhauser and Xu [6] describe a system that learns a grammar for use in automatic markup. Shafer [13] describes a C++ library, the GB (Grammar Builder) Engine which produces a grammar by generalizing examples marked up in an SGML-style. Work by Ahonen et al. [1,2] uses a more classical grammatical inference approach based on a characterizable subclass of regular languages that they call $(k-h)$ contextual languages. Goldman and Widom [9] describe the use of automatically constructed DataGuides for browsing database structure and formulating queries. Young-Lai and Tompa [15] look at applying stochastic grammar inference to the problem.

Cross-references

- ▶ Hidden Markov Models
- ▶ Inductive Inference
- ▶ Information Theory
- ▶ Machine Learning
- ▶ Markov Models
- ▶ Schema
- ▶ XML Selectivity Estimation
- ▶ XML Database

Recommended Reading

1. Ahonen H., Mannila H., and Nikunen E. Generating grammars for SGML tagged texts lacking DTD. In Proc. of the Workshop on Principles of Document Processing, 1994.
2. Ahonen H., Mannila H., and Nikunen E. Forming grammars for structured documents: an application of grammatical inference. In Lecture Notes in Computer Science, 862. R. Carrasco, J. Oncina (eds.), 1994, pp. 153–167.
3. Angluin D. On the complexity of minimum inference of regular sets. Inf. Control, 39:337–350, 1978.
4. Angluin D. Inference of reversible languages. J. ACM, 29:741–785, 1982.
5. Baum L.E., Petrie T., Soules G., and Weiss N. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. Ann. Math. Statist., 41(1):164–171, 1970.
6. Fankhauser P. and Xu Y. MarkItUp! An incremental approach to document structure recognition. Electron. Publ. Orig. Dissem. Des., 6(4):447–456, December 1993.
7. Gold E.M. Language identification in the limit. Inf. Control, 10:447–474, 1967.
8. Gold E.M. Complexity of automaton identification from finite data. Inf. Control, 37:302–320, 1978.
9. Goldman R. and Widom J. DataGuides: enabling query formulation and optimization in semi-structured databases. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 436–445.
10. Hopcroft J.E. and Ullman J.D. Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading, MA, 1979.
11. Oncina J. and García P. Inferring regular languages in polynomial updated time. In Pattern Recognition and Image Analysis. N.P de la Blanca, A. Sanfeliu, E. Vidal (eds.). World Scientific, Singapore, 1992, pp. 49–61.

12. Sánchez J.A. and Benedí J.M. Statistical inductive learning of regular formal languages. In Lecture Notes in Computer Science, 862. R. Carrasco, J. Oncina (eds.), 1994, pp. 130–138.
13. Shafer K. Creating DTDs via the GB-engine and Fred, 1995.
14. Stolcke A. and Omohundro S. Inducing probabilistic grammars by Bayesian model merging. In Lecture Notes in Computer Science, 862. R. Carrasco, J. Oncina (eds.), 1994, pp. 106–118.
15. Young-Lai M. and Tompa F.W. Stochastic grammatical inference of text database structure. Mach. Learn., 40(2):111–137, 2000.

Grammatical Induction

▶ Grammar Inference

Grammatical Inference

▶ Grammar Inference

Graph

Hans Hinterberger
ETH Zurich, Zurich, Switzerland

Synonyms

Graph; Chart; Plot

Definition

Graph theory: A set of nodes (also called **points** or **vertices**) connected by links called **lines** or **edges** or **arcs**. In an **undirected graph**, a line from point ***A*** to point ***B*** is considered to be the same thing as a line from point ***B*** to point ***A***. In a **directed graph**, the two directions are counted as being distinct **arcs** or **directed edges**.

Mathematics: A diagram exhibiting a relationship, often functional, between two or more sets of numbers as a set of points having coordinates determined by the relationship. Also called a plot.

Computer science: A data structure representing relationships or connections in lists, trees, and networks.

Data visualization: Any pictorial device such as a point graph, surface graph or symbol graph used to display numerical relationships. Also called a chart because graphs constitute one of the major categories of charts. Graphs combine two or more straight or circular axes, utilizing one or more quantitative scales. Straight axes are typically drawn perpendicular to each other, sharing a common origin. The most notable exceptions are triangular graphs, nomographs, and parallel coordinates.

Triangular (Trilinear) Graphs place the axes either along the sides or the along the angle bisectors of an equilateral triangle.

Nomographs (Nomograms) arrange the axes in such a way that a straight index line connecting known values on two axes passes through the solution value on another axis. The most common nomographs consist of three parallel axes, some place the axes at an angle to each other, others employ one or more curved axes.

Circular axes are based on polar coordinates, using angle or radius or both to represent values.

Key Points

Because graphs can be powerful tools to reveal the structure of data they are the preferred method to visualize data in science and technology, particularly as a tool for exploratory data analysis.

The use of graphical images to show data has evolved during the past 250 years, reaching unprecedented popularity with computer supported methods. The basic ideas behind good graphical design, however, transcend technology as they follow principles dictated by the human visual system.

W.S. Cleveland [1] has studied methods and basic principles that must be observed if a data analyst wants to realize a graph's potential for visualization. A classic on issues of graphical practice, including a theoretical approach to data graphics has been published by E.R. Tufte [3]. R.L. Harris [2] has assembled a comprehensive overview of different graphs by organizing them categorically according to their type of configuration (e.g., line graph, contour graph), area of application (e.g., technical, business), type of data plotted (e.g., original data, derived data), number of axes, and by purpose (e.g., analysis, monitoring, presentation).

Graphs are not only used to visualize data, they also serve as a tool for data acquisition and storage carried out by mechanical plotting devices or, in the form of nomograms, function as calculating devices, usually designed to perform a specific calculation.

Cross-references

▶ Chart
▶ Data Visualization

▶ Exploratory Data Analysis
▶ Parallel Coordinates

Recommended Reading

1. Cleveland W.S. The Elements of Graphing Data (revised edn.). Hobart Press, Summit, NJ, 1994.
2. Harris R.L. Information Graphics: A Comprehensive Illustrated Reference, Oxford University Press, Oxford, 1999.
3. Tufte E.R. The Visual Display of Quantitative Information. Graphics Press, Cheshire, CT, 1983.

Graph

▶ Chart
▶ Diagram

Graph Data Management in Scientific Applications

AMARNATH GUPTA
University of California-San Diego, La Jolla, CA, USA

Synonyms

Graph theory; Graph data structure; Graph database

Definition

In mathematics and computer science, graphs are mathematical structures used to model pairwise relations between objects from a certain collection. As a data structure, a "graph" is a set of vertices or "nodes" and a set of edges that connect pairs of vertices.

Historical Background

Graph data management has been studied for nearly two decades. A recent survey [1] states "Graph db-models are applied in areas where information about data interconnectivity or topology is more important, or as important, as the data itself. In these applications, the data and relations among the data, are usually at the same level…. It allows for a more natural modeling of data" and "Queries can refer directly to this graph structure. Associated with graphs are specific graph operations in the query language algebra, such as finding shortest paths, determining certain subgraphs, and so forth." One of the earliest applications of graph data models in science was the study of road networks [9]. While the development of medical terminologies started in the 1980s, the need to study the problem of medical terminology as a graph database came about in the early 1990s. Ellis [4] was one of the early attempts to model nomenclature such as SNOMED and the Unified Medical Language System (UMLS) as graphs.

Foundations

Graphs in Science

Many scientific disciplines deal with data that are structured like graphs. In chemistry, molecules can be considered as graphs whose nodes are atoms and edges are atomic bonds. In ecosystem studies, food webs are graphs where the nodes represent organisms and the edges are directed, representing predatory relationship. In infectious disease epidemiology, the contact network is an undirected graph, where the nodes are people affected by the disease, and an edge is created between pairs of people who came in contact with each other, and possibly caused the infection. In each application the properties of the graph are a little different. The nodes and edges may have additional attributes. The edges may have weights on them. The graph may be acyclic or have cycles. The graph may have a regular structure or may be more random in nature. Despite these difference, in all of these cases, the scientific pursuit involves some kind of network (i.e., graph-based) analysis. In the ecosystem example above, scientists analyze the structure of the graph (e.g., strongly connected components, weighted path lengths) as well as the values of node attributes (e.g., estimates of nutrients) [5].

Graph data management becomes important for science when (i) the problem domain has a very large graph and complex structural properties of this graph need to be extracted or computed by searching for structural patterns over the graph, or (ii) the problem domain has a large collection of graphs, each of which is not necessarily large. The fist situation occurs in the case of large protein networks in biology, where queries may correspond to extracting neighborhoods around nodes that satisfy properties given by the conditions of the query. The second situation occurs in the case of a library of chemical structures, from which one may want to find frequently occurring subgraphs from among a set of graphs that satisfy some query conditions. The data management needs and solutions for these two categories of problems are generally different.

Data Management for Large Graphs

Perhaps the most significant development in graph data management for the large graph case has been done in the context of biological networks, which are graphs where the nodes represent different kinds of biological molecules and edges represent observed or computed relationships between molecule pairs. In some cases, the graphs also include ontologies, which are graphs of standard terms and inter-term relationships. The simplest graph database represents a graph where nodes and edges have labels, and is implemented in a relational database system, and has a node table and an edge table over which graph operations are performed. There are three broad classes of operations that are performed in most common science applications.

1. *Relational Queries on the attributes of Nodes and Edges*: In general, the nodes and edges for the graphs are "thick," that is, they have attributes that are spread into one or more relational tables that can be joined with the basic node and edge tables. A typical query against the graph will have two kinds of conditions – those that are on the connectivity structure of the graph, and those that are standard relational queries on the attributes of the nodes and edges.
2. *Substructure Exploration, Extraction and Transformation*: These operations probe into the connectivity structure of the graph. A basic operation is to extract the k-neighborhood of a node, where the node is selected by some other query condition. Another basic operation is to find the shortest path(s) between two nodes. The query becomes much more complex if all paths between two nodes are desired – for a heavily interconnected graph (i.e., a graph with a small diameter) it may return the entire graph. Even if the graph is not heavily interconnected, the number of paths between two nodes is exponential in the number of edges. In practice, many scientific graph management systems do not allow an all-paths operation. A more complex operation seeks to extract a subgraph that satisfies certain conditions. The PQL system [10] allows queries of the form:

   ```
   SELECT A[−2], A[−*]B
   FROM A, B
   WHERE  A[−<  5]B  AND  A  ISA
   'enzyme' AND
   B.name = 'Propane-1,2-diol'
   ```

   The query returns a 2-neighborhood of a graph around node A and all (denoted by −*) the paths between A and B, where A and B are nodes that satisfy the graph pattern specified by WHERE condition. The WHERE condition looks for all B's having a specific name, and all A's that are within five edges away from the B's, and at the same time have an edge labeled ISA with the term "enzyme." In the industry, the Life Sciences Division of IBM has created a graph extension of DB2, which allows similar operations, and further allows construction and merging of new graphs. Eckman and Brown [3] provides a detailed overview of the system as applied to the life science domain. The graph transformation operation is used in applications like the food web [5], where a graph like a binary food web graph is converted into an aggregate graph called the trophic food web, where nodes having similar properties are collapsed into aggregate nodes and edges between individual nodes are also aggregated into composite edges between the corresponding aggregate nodes.
3. *Computation of Structural Properties*: A different class of operations focuses on computing aggregate graph properties like the number of cliques or strong components in the graph, as well as node properties like betweenness centrality (see http://en.wikipedia.org/wiki/Centrality for a definition and other similar measures). Epidemiological studies like [2] first compute the largest connected component and then determine nodes having the highest centrality values. Applications like molecular structure analysis [6] identify operations like maximal clique operations over Cartesian products of graphs. While graph matching and searching techniques have started using structural properties of graphs, relatively little work has been done in using data management techniques to facilitate these computations.

Data Management for Large Collections of Graphs

In the case where a science application uses a large collection of graphs, most of the data management issues center around finding common substructures, structurally similar graphs and ways to index graph collections to improve retrieval efficiency. A specific class of techniques focuses on mining graph collections to find frequently occurring substructures for chemical data.

Some Data Management Techniques

- A class of techniques called *descriptor spaces* use the idea of keyword-based text retrieval and adapt it for graphs. If the graph is conceived as a document, a keyword is a characteristic substructure of the graph. Chains of a certain length, cycles up to a certain size and number, tree-like branching fragments etc. are commonly used as descriptors. Wale and Karypis [12] computes acyclic portions of the graph as descriptors for chemical compounds. Once these descriptors are collected, graphs are represented in a vector space where the dimensions of the space are formed by these extracted graph fragments. Given a query graph, a suitable similarity function is used to find a ranked list of similar graphs. A common similarity function is the radial basis function (see http://en.wikipedia.org/wiki/Radial_basis_function for a definition).
- A class of techniques has also been developed to rank matching graphs that are returned from any graph query. One basis of ranking graphs is by the frequency of their occurrence in a collection. A different way, often valued more in biological sciences, is by assigning a statistical significance to each returned graph, and ranking results on the significance value. In systems biology, such techniques have been developed for path ranking [8] and graph ranking [11]. However, the algorithms are not very efficient especially for large graphs. A more efficient graph ranking technique based on statistical significance has been developed for the frequent subgraph problem in [7].

Key Applications

Road networks, biological pathways, social networks, transportation and communication networks.

Cross-references

▶ Graph Management in the Life Sciences

Recommended Reading

1. Angles R. and Gutierrez C. Survey of graph database models. ACM Comput. Surv., 40:1–39, 2008.
2. De P., Singh A.E., Wong T., Yacoub W., and Jolly A.M. Sexual network analysis of a gonorrhoea outbreak. Sex. Transm. Infect., 80(4):280–285, 2004.
3. Eckman B.A. and Brown P.G. Graph data management for molecular and cell biology. IBM J. Res. Dev., 50(6):545–560, 2006.
4. Ellis G. Managing large databases of complex medical knowledge using conceptual graphs. In Proc. of the National Health Informatics Conference, 1993, pp. 4–13.
5. Gaedke U. A comparison of whole-community and ecosystem approaches (biomass size distributions, food web analysis, network analysis, simulation models) to study the structure, function and regulation of pelagic food webs. J. Plankton Res., 17(6):1273–1305, 1995.
6. Hattori M., Okuno Y., Goto S., and Kanehisa M. Development of a chemical structure comparison method for integrated analysis of chemical and genomic information in the metabolic pathways. J. Am. Chem. Soc., 125:11853–11865, 2003.
7. He H. and Singh A.K. GraphRank: statistical modeling and mining of significant subgraphs in the feature space. In Proc. 22nd Int. Conf. on Data Mining, 2006, pp. 885–890.
8. Kelley B.P, Yuan B., Lewitter F., Sharan R. Stockwell B.R., and Ideker T. PathBLAST: a tool for alignment of protein interaction networks. Nucleic Acids Res., 32(web server issue):W83–88, 2004.
9. Kunii H.S. DBMS with graph data model for knowledge handling. In Proc. Fall Joint Computer Conf. on Exploring Technology: Today and Tomorrow, 1987, pp. 138–142.
10. Leser U. A query language for biological networks. Bioinformatics, 21(Suppl 2):ii33–ii39, 2005.
11. Sharan R., Suthram S., Kelley R.M., Kuhn T., McCuine S., Uetz P., Sittler T., Karp R.M., and Ideker T. Conserved patterns of protein interaction in multiple species. In Proc. Natl. Acad. Sci. USA, 102:1974–1979, 2005.
12. Wale N. and Karypis G. Acyclic Subgraph-based Descriptor Spaces for Chemical Compound Retrieval and Classification. UMN CSE Technical Report #06–008, 2006.

Graph Data Structure

▶ Graph Data Management in Scientific Applications
▶ Information Integration Techniques for Scientific Data

Graph Database

Peter T. Wood
Birkbeck, University of London, London, UK

Synonyms

Network database; Link database; Semi-structured database

Definition

A graph database is a database whose data model conforms to some form of graph (or network or link) structure. The graph data model usually consists of nodes (or vertices) and (directed) edges (or arcs or links), where the nodes represent concepts (or objects) and the edges represent relationships (or connections) between these concepts (objects). Therefore the nodes are typically labeled with the names of concepts or objects, while the edges are labeled with types of relationships. More elaborate labeling might involve sets of attribute-value pairs being associated with nodes and/or edges. In addition, more complex structures, such as nested graphs or hypergraphs, may also be permitted. On the other hand, the graph model may be restricted to allow only certain types of graph structures, for example, only acyclic graphs or those that have a distinguished root node.

Similar to relational databases, users can query the graph database with a high-level query language. However, such query languages are usually designed in such a way as to make it easy to ask about paths through the graph (or routes through the network) of unspecified length.

Historical Background

Connections between graph structures and databases existed from the earliest data models developed in the 1960s and 1970s. For example, the hierarchical data model viewed data as comprising tree structures, while the network data model viewed data as consisting of linked structures. The entity-relationship model also defines what is essentially a graph-based structure. However, none of these would really be considered as a graph database.

The notion of a graph database evolved from the recognition that, in a number of applications, a *single* relationship type connecting instances of a single concept may itself form a graph structure. Examples include the link structure in hypertext documents, road connections in geographical databases, and, more recently, various pathways in biological databases. In all of these applications, useful queries need to be able to ask about paths through the database, where the length of the required paths is not necessarily predetermined or known beforehand by the user. Such capabilities were not available in the data models mentioned above, nor in the relational model as originally defined.

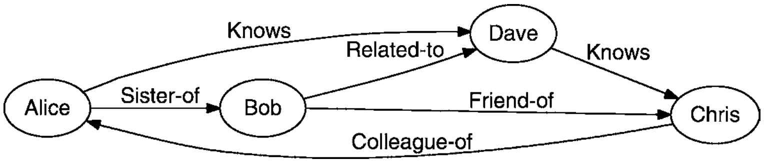
A further requirement of most data models is that data conforms strictly to the structural requirements of the model. Applications in heterogenous data integration and data on the web needed more flexibility, giving rise to the notion of semi-structured data [1] which also conformed to a graph model. Appropriate data models and query languages for graph databases and their applications were investigated mostly in the 1980s and 1990s, with interest having been renewed recently in the context of the semantic web and social networks.

Foundations

A graph database typically comprises a set of graphs. Each graph in the database consists of a set of nodes *N* and a set of edges *E*, with each edge in *E* connecting a pair of nodes in *N*. The edges may be directed or undirected. In the case of the former, an edge is an ordered pair of nodes; in the case of the latter, it is an unordered pair of nodes. Both nodes and edges may have labels associated with them. This is usually done by two functions, one mapping nodes to labels and the other mapping edges to labels. The labels may be simple values, tuples of values, sets of values, and so on.

A very simple example of a directed graph is shown in Fig. 1. Here the node labels denote names of people and the edge labels denote relationships between people, so this graph could represent a tiny fragment of a simple social network. Note that the graph is *cyclic*; for example, Dave knows Chris who is a colleague of Alice who knows Dave.

Given a graph of data like that in Fig. 1, various queries about the data may be relevant. These might be conventional queries such as to find the pairs of people who know each other (Alice knows Dave, and Dave knows Chris), or to find the people who are sisters of friends of Chris (Alice). However, "schema-level"queries may also be needed, for example to find the relationships in which Alice is involved, or to find in what way(s) Alice and Chris are related. Users might also be interested in nodes in the graph that are connected by paths whose lengths are not known a-priori, such as people connected by any sequence of "knows" relationships or any sequence of either "knows" or "friend-of" relationships. In general, users may want to specify a pattern that should be matched by the sequence of edge labels on paths in which they are interested. One way of specifying such a pattern is by using *regular expressions* over the alphabet of edge labels [9]. So for example, the regular expression (know|friend-of)$^+$ specifies sequences of one or more occurrences of edges labeled with either "knows"



Graph Database. Figure 1. A simple example of a graph.

or "friend-of." The use of regular expressions for querying path information has been studied extensively in the area of semi-structured data [1]. XPath, a language for selecting nodes from XML documents, provides a limited form of path pattern specification for querying XML documents, which are essentially trees rather than graphs.

Depending of the application area for which a graph database is being used, there are many other forms of query that may be useful. Examples include shortest path queries (e.g., find the shortest or quickest route from one node to another), subgraph isomorphism queries (the query is viewed as a subgraph whose matches against a graph in the database are to be found), approximate graph matching (where the query may not match a subgraph exactly), finding complete subgraphs (where every node is connected to every other node), queries involving counting the indegree and outdegree of nodes (the number of incoming and outgoing edges, respectively, perhaps to see how "well-connected" a node is), finding the largest common subgraph among a number of graphs, or locating the least common ancestor of two or more nodes (usually in trees).

Unlike in a conventional database, the data in Fig. 1 does not have an explicit *schema* describing it. This lack of explicit *schema* is one of the characteristics of what has been termed semi-structured data [1]. In fact, some schema information is given by the edge labels, so the data are in some sense *self-describing*. However, there is no requirement that the data conform to a set of constraints as usually defined in a schema. Such constraints might assign types to nodes, state what edge labels are permitted, and limit the types of nodes that are permitted to participate in each relationship. Another graph-based model that does not require a schema is RDF (Resource Description Framework), one of the languages proposed for the semantic web. Nevertheless, researchers have defined various forms of schemas for graph databases [3], and RDF does have an associated schema language, although this is used for inference of derived information rather than for validation of RDF graphs.

As in conventional databases, one of the principal concerns in a graph database system is the efficient evaluation of queries. This gives rise to the study of query optimization [5,6] and appropriate index structures for graph data [10,15]. Both of these topics have been studied more extensively in the context of tree-structured XML data.

Some applications require graph structures that are more elaborate than the simple version of nodes and edges described above. For example, hypergraphs, where edges comprise sets of nodes rather than pairs of nodes, have been used to model hypertext [14]. Alternatively, the hypernode model [12] considers nodes that can themselves comprise graphs, thereby providing a useful abstraction mechanism. A different approach to abstraction is provided by so-called blobs that can appear in the hygraphs of the Hy$^+$ system [4]. These blobs comprise sets of nodes and share some similarities with the blobs defined in higraphs [8]. Graphs have also been used as models for object-oriented databases [7].

Key Applications

There are a large number of potential application areas for graph databases. These include hypertext [14] and the Web, discovery of semantic associations in national security (or criminal investigation) applications [13] (also called link analysis), chemical structure modeling, geographic information systems, bibliographic citation analysis, taxonomy and partonomy representation, data provenance graphs, and the semantic web, e.g., RDF [2]. In biology alone [11], there are numerous applications including metabolic pathways, signaling pathways, gene regulatory networks, gene clusterings, and protein interaction networks.

Cross-references

- ▶ Biological Networks
- ▶ Data Integration

- ▶ Graph Management in the Life Sciences
- ▶ Graph
- ▶ Graph Data Management in Scientific Applications
- ▶ Indexing Semi-Structured Data
- ▶ Information Integration
- ▶ Information Integration Techniques for Scientific Data
- ▶ Network Data Model
- ▶ Object Data Models
- ▶ Semi-Structured Data Model
- ▶ Social Networks
- ▶ Resource Description Framework
- ▶ Path Query
- ▶ Query Language

Recommended Reading

1. Abiteboul S., Buneman P., and Suciu D. Data on the Web: From Relations to Semi-structured Data and XML. Morgan Kaufmann, San Francisco, CA, 2000.
2. Angles R. and Gutierrez C. Querying RDF data from a graph database perspective. In Proc. 2nd European Semantic Web Conference, 2005, pp. 346–360.
3. Buneman P., Davidson S., Fernandez M., and Suciu D. Adding structure to unstructured data. In Proc. 6th Int. Conf. on Database Theory, 1997, pp. 336–350.
4. Consens M.P. and Mendelzon A.O. Hy$^+$: a hygraph-based query and visualization system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 511–516.
5. Fernandez M. and Suciu D. Optimizing regular path expressions using graph schemas. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 14–23.
6. Goldman R. and Widom J. DataGuides: enabling query formulation and optimization in semi-structured databases. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 436–445.
7. Gyssens M., Paredaens J., den Bussche J.V., and Gucht D.V. A graph-oriented object database model. IEEE Trans. Knowl. Data Eng., 6(4):572–586, August 1994.
8. Harel D. On visual formalisms. Commun. ACM, 31(5):514–530, May 1988.
9. Mendelzon A.O. and Wood P.T. Finding regular simple paths in graph databases. SIAM J. Comput., 24(6):1235–1258, December 1995.
10. Milo T. and Suciu D. Index structures for path expressions. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 277–295.
11. Olken F. Graph data management for molecular biology. OMICS: A Journal of Integrative Biology, 7(1):75–78, 2003.
12. Poulovassilis A. and Levene M. A nested-graph model for the representation and manipulation of complex objects. ACM Trans. Inf. Syst., 12(1):35–68, 1994.
13. Sheth A., Aleman-Meza1 B., Arpinar I.B., Bertram C., Warke Y., Ramakrishnan C., Halaschek C., Anyanwu K., Avant D., Arpinar F.S., and Kochut K. Semantic association identification and knowledge discovery for national security applications. J. Database Manag., 16(1):33–53, 2005.
14. Tompa F. W. A data model for flexible hypertext database systems. ACM Trans. Database Syst., 7(1):85–100, 1989.
15. Yan X., Yu P.S., and Han J. Graph indexing: a frequent structure-based approach. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 335–346.

Graph Database Mining

▶ Frequent Graph Patterns

Graph Drawing

▶ Visualizing Network Data

Graph Embedding

▶ Dimension Reduction Techniques for Clustering

Graph Layout

▶ Visualizing Network Data

Graph Management in the Life Sciences

Ulf Leser, Silke Trißl
Humboldt University of Berlin, Berlin, Germany

Definition

Graphs play an increasingly important role in many research areas in the life sciences. Especially in systems biology, graphs are used to model the complex temporal and spatial relationships between entities within an organism. For example, graphs are used to model signaling pathways, where nodes are proteins and edges represent the flow of information between proteins. The flow represents physical modifications of the participating proteins, such as the addition or removal of certain chemical groups. Since proteins are often involved in various signaling pathways, one can model the complete signaling management inside a cell as a graph consisting of tens of thousands of nodes and many more edges. However, graphs are also used in less obvious areas. Biological *ontologies* are cycle-free graphs of biological concepts connected by specialization relationships; they are called thesauri in information retrieval. Phylogenetic networks are formed by species and their evolutionary relationships.

Biological graphs are the subject of various types of analyses. *Graph querying* searches for all occurrences of a given (sub-) graph pattern inside a large graph, for instance to find common motifs of gene regulation. *Graph mining* aims at detecting significant patterns inside one or more graphs, for instance finding all graph patterns that are more frequent than expected by chance. *Graph clustering* tries to find dense subgraphs, which potentially can be associated with stable multi-protein complexes.

Formally, a graph G is composed of nodes V and edges E with E $\subseteq$ V×V. Nodes usually represent biological or chemical entities, and edges represent specific types of relationships between those entities. Depending on the application, edges may be directed or undirected, labeled or unlabeled, and weighted or unweighted.

Historical Background

For a long time, graphs have been used in the life sciences only for describing the two- or three-dimensional structure of complex molecules, especially of proteins. In such graphs, nodes represent atoms and edges represent bonds between atoms. However, with the recent massive increase in data production on all levels (including DNA sequences, protein structures, gene expression levels etc.), many more applications of graphs have been suggested. The most prominent types of graphs that emerged are: (i) networks of interacting proteins, (ii) networks of gene regulation, (iii) networks of signal transduction, and (iv) metabolic networks.

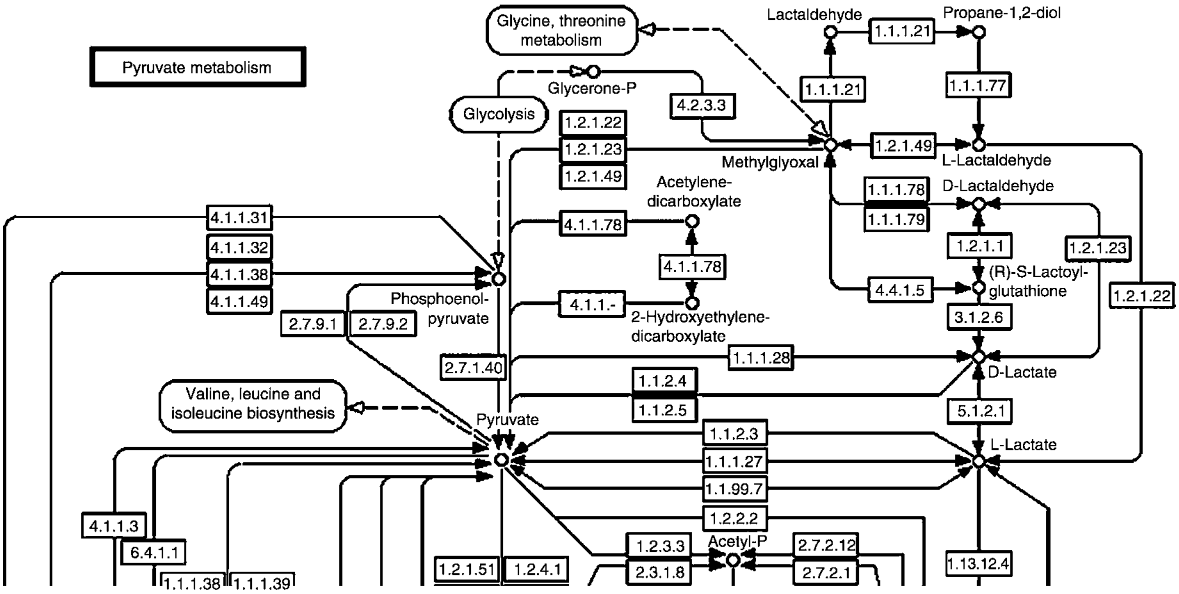
The study of such networks has flourished with the advent of high-throughput techniques to detect the different "types" of edges. The most important ones are the Yeast 2-Hybrid technique for detecting physical interactions between proteins and the co-immunoprecipitation technique for detecting regulation events. The study of the complex interplay of all components inside a cell and across cell boundaries is one of the main goals of modern systems biology. Today, the largest graphs available based on experimental evidence contain in the order of ~15,000 proteins and ~100,000 edges. When predicted information is taken into account, the resulting graphs are much larger, with up to one million nodes and five million edges.

Systems for storing biological networks and their experimental evidence were first developed around 2000, with the notable exception of the KEGG database (first published in 1997) and the EcoCyc system (first published in 1994). With more and more networks being available, various approaches emerged to analyze the information contained in those graphs, for instance to predict further edges based on graphs from evolutionary closely related species, or to predict the function of a protein from its neighbors in a protein-protein interaction network. Only very recently, there have been proposals for developing specific data management solutions that take into account the intrinsic nature of biological graphs.

Foundations

Types and Models of Graphs

The most prominent types of graphs in the life sciences are (i) Networks of physically interacting proteins: Here, nodes represent proteins and edges represent a physical interaction between two proteins. Edges are directed or undirected and may be labeled with the type of the interaction and the evidence for the biological truth of the interaction; (ii) Networks of gene regulation: Nodes represent genes or transcription factors (TF), and edges represent the influence of transcription factors on gene expression. Since transcription factors are also produced by gene expression, TFs also influence the regulation of other TFs. Edges are directed and labeled to reflect a positive or a negative influence; (iii) Signal transduction: Nodes represent proteins or other molecules and edges model the flow of information. Edges are directed and may be labeled; (iv) Metabolic networks (see Fig. 1): Nodes represent arbitrary molecules and edges represent a chemical reaction between the connected molecules catalyzed by enzymes. Edges are labeled and directed



Graph Management in the Life Sciences. Figure 1. Fraction of the Pyruvate Metabolism as represented in the KEGG Database.

or, as many reactions are reversible, bidirectional. In any real system, nodes usually have a wealth of information attached, such name(s), sequences, structures, functions, or experimental values, e.g., values from microarray experiments.

Properties of Biological Networks

A human cell is estimated to contain approximately 20,000 genes that are translated in up to 150,000 different proteins. The same number can also be estimated for other mammals, such as mouse, rat, or chimp. Metabolic networks currently contain up to 20,000 nodes and 40,000 edges, depending on the organism (for bacteria, the networks are much smaller). In contrast, cross-species protein-protein interaction networks currently contain up to 1,000,000 nodes and, depending on the method used to infer the interactions, many more edges.

Biological networks exhibit a specific structure, which is shared by other complex systems, such as the Internet or social networks. Biological networks are scale-free, i.e., the degree distribution follows a power-law. In *scale-free graphs* many nodes have only few edges. These nodes are connected by some nodes that have many edges, called hubs. Scale-free networks can be grown in computer simulations using the so-called preferential attachment, which reflects the evolution of existing biological networks. Preferential attachment means that graphs are grown incrementally. One starts with a small number of nodes and incrementally adds new nodes, which are attached to existing ones, preferably to nodes that are already well connected [1].

Graph Databases

Graph databases in the life sciences can be divided into different categories. First, there are databases about metabolic pathways. The best known sources are KEGG, aMAZE, Reactome, and BioCyc, which contain information added manually by biological experts. For example the reference pathway in KEGG currently (as of 2007) contains 20,000 nodes and 40,000 edges. Secondly, there exist various protein-protein interaction databases, e.g., DIP, MINT, STRING, or PubGene. While the first two databases only accept entries for which strong evidence has been published, STRING also accept entries from less reliable high-throughput experiments. STRING and PubGene also automatically mine the scientific literature for interactions and include those into the database. Therefore the size of the data sets as well as the data quality varies to a great degree.

To exchange data between different databases several standards were established. The “Systems Biology Markup Language” (SBML) and the “BioPAX Data Exchange format” were both designed for the exchange of data in systems biology and especially for biological networks. The "Proteomics Standards Initiative Molecular Interaction XML format" (PSI MI) targets only protein-protein interaction data. For a comparison of the different format standards see [11].

Graph Query Languages

Query languages for graphs are based on the description of properties of subgraphs. Evaluating a graph query on a graph means to find all subgraphs having the specified properties. Numerous different properties may be used, which require the existence of nodes with certain attributes, edges with specific labels, or the existence of paths of various shapes connecting nodes.

The specific difficulty in developing graph query languages is the required support for path queries, i.e., queries over sequences of connected nodes. An exemplary query posed on a biological network might be "Give me all enzymes that affect a compound called glucose in less than five steps and which are also associated with the enzyme 1.6.3.1 in an arbitrary number of steps." Such queries can be solved by either traversing the graph at query time, or by using special graph indexing structures, such as the 2-Hop-Cover. In general, indexing graphs is much harder than indexing trees. Even when graph queries contain no transitive predicates their evaluation is equivalent to solving the subgraph isomorphism problem, which is NP-complete. However, as nodes in biological graphs usually have unique labels, evaluation of queries in real applications is often much less complex.

Examples of graph query languages in biology are the pathway query language [5] and the language described in [2]. Both approaches are implemented on top of a relational database management system.

Graph Mining

The need to analyze graph data has lead to the development of various tools and algorithms that solve specific problems on graphs, of which three are particularly important. (i) Finding cliques or densely connected clusters (quasi-cliques) in a graph of protein-protein interactions is important to identify molecular complexes, which are groups of proteins that together perform a biological function. Jeong and colleagues present different algorithms, for instance based on supermagnetic clustering or a Monto Carlo optimization algorithm [10]; (ii) Subgraphs that are present in the PPI networks of various species (conserved subgraphs) are important evidence towards the evolutionary importance of subnetworks. Algorithms either find perfectly conserved subgraphs or allow insertions and deletions of nodes or edges. One approach based on path alignment is presented in [8]; (iii) Frequent subgraphs in biological networks are believed to hint "building blocks" of the molecular machinery of a cell. A subgraph of a graph is called frequent when it appears more often than expected. An efficient algorithm based on random sampling of isomorphic subgraphs is presented in [14].

Key Applications

Graph-Based Function Prediction

The computer-driven prediction of the biological function of a given protein is a long-standing goal of bioinformatics research, which, until recently, was mostly based on finding similarities between the sequences of different genes or proteins. The possibility to analyze the proteins with which a protein interacts in a cell has added a powerful new source of evidence for function prediction. It has, for instance, been shown that proteins in densely connected regions of a protein-protein-interaction network very often share the same biological function. Furthermore, the fact that two proteins interact with the same other proteins increases the likelihood that these two proteins share a biological function. This interaction can happen within a species, or can be transferred from orthologous (i.e., highly similar) proteins in other species. A survey on those methods has appeared in [8].

Managing Biological Ontologies

Biological ontologies are structured sets of concepts. Usually, concepts have a name, a textual description, and are related to each other through ISA (generalization/specialization) or PART-OF relationships. Prominent examples are the Gene Ontology (for describing the function and location of genes) and the Mammalian Phenotype Ontology (for describing phenotypes of mammalians). Naturally, an ontology may be seen as a graph, where the concepts are nodes and the relationships are edges. Ontologies form trees or directed acyclic graphs. Therefore, many approaches to the management, analysis, and visualization of ontologies use a graph-based framework. For instance, the semantic similarity between two concepts is often computed by finding the least common ancestor of the two concepts in the ontology [6]. In the same spirit, the semantic similarity between two genes based on all their assigned concepts may be computed by comparing the two subgraphs formed by the concepts of the genes and their ancestors in the ontology. Such techniques are often used to score the functional coherence of groups of genes, for instance, whose gene expression react in a similar way under external stimulus.

Management of Phylogenetic Trees

Phylogenetic trees are used to describe the relationship between different species. The trees may be rooted or not and usually are unweighted. Leaf nodes represent different species and inner nodes stand for common, often unknown ancestors. Phylogenetic trees are constructed using one or more common features from the species, usually a gene whose genetic sequence varies slightly in different species. Since there are various algorithms for constructing phylogenetic trees using different features, very often contradicting trees emerge. Recently, tree banks have been proposed as special applications for managing and searching phylogenetic trees. A particular problem is to find for a given query tree those trees in a database of phylogenetic trees that are most similar to the query. This requires a tree distance measure and a search procedure (see for instance, [13]).

It is more and more acknowledged that trees are inappropriate for modeling certain effects in evolution, such as plant hybridization or horizontal gene transfer. This results in efforts to develop algorithms for the inference of phylogenetic networks, i.e., graphs where one species can have two or more ancestors. An interesting method based on a Maximum Parsymony approach to network reconstruction is presented in [3].

Visualization of Biological Networks

Visualization tools for biological networks must address several issues. First of all, they must be able to display the graph using suitable layout algorithms. In addition, these tools must provide visualization of attached information, e.g., name of a node, its cellular location, or results of high-throughput experiments. Several tools were specifically developed for this purpose. A well known tool is Cytoscape, a Java-based open source tool that can display large networks and can be extended using plug-ins. Further tools are Pathway Tools, Pathway Studio, or Ospey, which both also allow various kinds of network analysis. For a review on visualization tools for biological networks see [12].

Future Directions

Systems biology will be a major research area in the life sciences in the coming years. Since it focuses on the interplay of chemical entities in cells, it builds on graphs in a very fundamental manner. Therefore, the importance of graphs in the life sciences is expected to grow further in the near future. This will also bring a steep increase in the size of graphs under study, as more and more experimental data become available. Since, for many applications also the inclusion of predicted edges perfectly makes sense, already now graphs may contain up to several hundreds of thousands of edges. It is therefore foreseeable that studies that address interaction data from many species will have to cope with millions of objects in the near future.

Technically, relational databases nowadays build the backbone of most graph management systems. However, a graph may also be modeled using the *Resource Description Framework* (RDF), an underlying technology of the *Semantic Web*. Therefore, query languages for RDF might be a natural choice for also analyzing biological graphs. However, the current proposals lack the advanced graph search features required in the life sciences. The development of new graph query languages, possibly including certain types of simple graph analysis predicates, will remain an important topic.

URL to Code

A few pathway databases

- "Pathguide: the pathway resource list": http://www.pathguide.org/
- "KEGG: Kyoto Encyclopedia of Genes and Genomes": http://www.genome.ad.jp/kegg/
- "Reactome": http://www.reactome.org/
- "BioCyc Home": http://biocyc.org/
- "DIP:Home": http://dip.doe-mbi.ucla.edu/
- "MINT database": http://mint.bio.uniroma2.it/mint/Welcome.do
- "STRING; functional protein association networks": http://string.embl.de/

A few visualization tools

- "Cytoscape: Analyzing and Visualizing Network Data": http://www.cytoscape.org/
- "Ariadne Genomics: Pathway Studio": http://www.ariadnegenomics.com/products/pathway-studio/
- "OSPREY: Network Visualization System": http://biodata.mshri.on.ca/osprey/servlet/Index

A few pathway management systems

- "PathCase: Metabolic Pathways Database System": http://nashua.cwru.edu/pathways
- "Pathway Tools Information Site": http://brg.ai.sri.com/ptools/
- "PatikaWeb": http://www.cs.bilkent.edu.tr/~patikaweb/

Cross-references

▶ Biomedical Data/content Acquisition, Curation
▶ Biological Networks
▶ Frequent Graph patterns
▶ Grpah Database
▶ Graph Data Management in Scientific Applications
▶ Ontologies and Life Science Data Management
▶ Query Languages for the life Sciences
▶ Tree-based Indexing

Recommended Reading

1. Barabàsi A.-L. and Oltvai Z.N. Network biology: understanding the cell's functional organization. Nat. Rev. Genet., 5:101–113, 2004.
2. Eckman B.A. and Brown P.G. Graph data management for molecular and cell biology. IBM J. Res and Dev., 50:545–560, 2006.
3. Jin G., Nakhleh L., Snir S. and Tuller T. Efficient parsimony-based methods for phylogenetic network reconstruction. Bioinformatics, 23:123–2128, 2006.
4. Karp P.D., Paley S., and Romero P. The pathway tools software. Bioinformatics, 18(Suppl. 1):S225–S232, 2002.
5. Leser U. A query language for biological networks. Bioinformatics, 21(Suppl. 2):ii33–ii39, 2005.
6. Lord P.W., Stevens R.D., Brass A., and Goble C.A. Investigating semantic similarity measures across the Gene Ontology: the relationship between sequence and annotation. Bioinformatics, 19(10):1275–1283, 2003.
7. Schaefer C.F. Pathway databases. Ann NY Acad Sci., 1020:77 –91, 2004.
8. Sharan R. and Ideker T. Modelling cellular machinery through biological network comparison. Nat. Biotechnol. 24(4):427–433, 2006.
9. Sharan R., Suthram S., Kelley R.M., Kuhn T., McCuine S., Uetz P., Sittler T., Karp R.M., and Ideker T. Conserved patterns of protein interaction in multiple species. In Proc. Natl. Acad. Sci. USA., 102(6):1974–1979, 2005.
10. Spirin V. and Mirny L.A. Protein complexes and functional modules in molecular networks. In Proc. Natl. Acad. Sci. USA., 100(21):12123–12128, 2003.
11. Strömbäck L. and Lambrix P. Representations of molecular pathways: an evaluation of SBML, PSI MI and BioPAX. Bioinformatics, 21(24):4401–4407, 2005.
12. Suderman M. and Hallett M. Tools for visually exploring biological networks. Bioinformatics, 23(20):2651–2659, 2007.
13. Wang J.T.L., Shan H., Shasha D., and Piel W.H. Fast structural search in phylogenetic databases. Evol. Bioinform. Online, 1:37–46, 2005.
14. Wernicke S. Efficient detection of network motifs. IEEE/ACM Trans. Comput. Biol. Bioinformatics., 3(4):347–359, 2006.

Graph Mining on Streams

Andrew McGregor
Microsoft Research, Silicon Valley, Mountain View, CA, USA

Synonyms

Graph streams; Semi-streaming model

Definition

Consider a data stream $A = \langle a_1, a_2,\ldots,a_m\rangle$ where each data item $a_k \in [n] \times [n]$. Such a stream naturally defines an undirected, unweighted graph $G = (V, E)$ where

$$V = \{v_1,\ldots,v_n\} and$$
$$E = \{(v_i, v_j) : a_k = (i,j) \text{ for some } k \in [m]\}.$$

Graph mining on streams is concerned with estimating properties of G, or finding patterns within G, given the usual constraints of the data-stream model, i.e., sequential access to A and limited memory. However, there are the following common variants.

Multi-Pass Models: It is common in graph mining to consider algorithms that may take more than one pass over the stream. There has also been work in the *W-Stream* model in which the algorithm is allowed to write to the stream during each pass [9]. These *annotations* can then be utilized by the algorithm during successive passes and it can be shown that this gives sufficient power to the model for PRAM algorithms to be simulated [8]. The *Stream-Sort* model goes one step further and allows *sorting passes* in which the data stream is sorted according to a key encoded by the annotations [1].

Weighted, Dynamic, or Directed Graphs: For many problems it is implicitly assumed that the elements a_k are distinct. When the data items are not distinct, the stream naturally defines a multi-graph, i.e., an edge (v_i, v_j) has multiplicity equal to $|\{k : a_k = (i, j)\}|$. In such a model it would be natural to consider the deletion of edges but this has not been explored thoroughly. Finally, the definition can be generalized to define weighted graphs where a third component of the data item, $a_k \in [n] \times [n] \times \mathbb{R}^+$, would indicate a weight $w_{(u,v)}$ associated with the edge (u,v), or directed graphs.

Adjacency and Incidence Orderings: In some applications it is reasonable to assume that all edges incident on the same node arrive consecutively. This assumption defines the *incidence model*. In the *adjacency model* no such assumption is made.

Historical Background

Graph problems were considered by Henzinger et al. [15] in one of the earliest papers on the data-stream model. Their work considered a variety of problems including pointer jumping problems based on the degrees of various nodes in a directed layered graph. Unfortunately, many of the results showed that a large amount of space is required for these types of problems. Subsequent early work considered counting the number of triangles in a graph [2] and estimating common neighborhoods [3]. Again, a large component of these results were negative. It seemed that more "complicated" computation was not possible in this model using only a small amount of space.

It seems that most graph algorithms need to access the data in a very adaptive fashion. Since the entire graph can not be stored, emulating a traditional algorithm may necessitate an excessive number of passes over the data. This has motivated various specific stream models tailored to processing graphs, including the *Semi-Streaming*, *W-Stream*, and *Sort-Stream* models. The semi-streaming model is characterized by an $O(n$ polylog $n)$ space restriction, i.e., space proportional to the number of nodes rather than the number of edges. For dense graphs this represents considerably less space than that required to store the entire graph. This restriction was identified as an apparent "sweet-spot" for graph streaming in a survey article by Muthukrishnan [17] and was first explored by Feigenbaum et al. [13]. The W-Stream and Stream-Sort models, described earlier, were introduced by Demetrescu et al. [9] and Aggarwal et al. [1] respectively.

Foundations

This section discusses the main upper and lower bounds known for solving graph problems in the data-stream model. The focus is on results for the standard data-stream model (including the Semi-Streaming space restriction) but note that some of the below problems, including testing connectivity, computing shortest paths, and constructing minimum spanning trees, have been considered in the W-Stream and Stream-Sort models [1,9].

Connectivity and Minimum Spanning Trees: Determining whether a graph is connected is a fundamental problem. It can be solved in the semi-streaming model and it can be shown that any one-pass algorithm requires $\Omega(n)$ space [15]. More generally, connectivity is a *balanced property* where a graph property $\mathcal{P}$ is *balanced* if there exists a constant $c > 0$ such that for all sufficiently large n, there exists a graph $G = (V, E)$ with $|V| = n$ and $u \in V$ such that:

$$\min\{|\{v : (V, E \cup \{(u,v)\}) \text{ has } \mathcal{P}\}|,\\ |\{v : (V, E \cup \{(u,v)\}) \text{ has} \neg \mathcal{P}\}|\} \geq cn.$$

It was shown that all balanced properties require $\Omega(n)$ space [12]. This was part of the justification for the semi-streaming space restriction. However, many problems become feasible with $O(n$ polylog $n)$ space such as testing planarity and determining whether a graph is bipartite. Testing higher degrees of vertex-connectivity has also been considered in the semi-streaming model. The below table summarizes the start of the art results for determining if a graph is k-connected, i.e., whether k vertices need to be removed in order to disconnect the graph:

k	Passes	Time (per-edge)	Ref.
1,2,3	1	$O(\alpha(n))$	[12]
4	1	$O(\log n)$	[12]
k	1	$O(k^2 n)$	[18]
k	$k + 1$	$O(k + \alpha(n))$	[18]

where $\alpha(n)$ is the inverse Ackermann function. Lower bounds have also been investigated for k-edge and k-vertex connectivity [15]. Other related results include a semi-streaming, one-pass algorithm for constructing the minimum spanning tree with $O(\log n)$ processing-time per edge.

Distances and Spanners: An undirected graph $G = (V, E)$ naturally defines a distance function $d_G : V \times V \rightarrow \mathbb{R}$ where $d_G(u, v)$ is the length of the shortest path in G between u and v. The diameter of G is the length of the longest shortest path, i.e.,

$$\mathrm{Diam}(G) = \max_{u,v \in V} d_G(u,v),$$

and the girth of G is the length of the shortest cycle in G, i.e.,

$$\mathrm{Girth}(G) = \min_{(u,v)\in E} [w_{(u,v)} + d_{G\backslash(u,v)}(u,v)].$$

To date, most of the algorithms for approximating quantities related to distance are based on constructing sparse *spanners*, where a subgraph $H = (V, E')$ is an (α, β)-spanner of $G = (V, E)$ if, for any vertices $x, y \in V$,

$$d_G(x,y) \le d_H(x,y) \le \alpha \cdot d_G(x,y) + \beta.$$

Note that constructing an $(\alpha, 0)$-spanner H for an unweighted graph also gives an indication of the girth of the original graph. In particular, if $H \neq G$ then $\mathrm{Girth}(G) \le \alpha + 1$ because there exists $(u, v) \in E(G)\backslash E(H)$ and this must satisfy $d_{G\backslash(u,v)}(u,v) \le \alpha$ if H is an $(\alpha, 0)$-spanner.

The first spanner constructions in the data-stream model were presented in [12,13]. The state of the art construction is due to Elkin [10] who presented a randomized, single-pass algorithm that constructs a $(2t - 1,0)$-spanner for an unweighted graph in $O((t \log n)^{1-1/t} n^{1+1/t})$ space (with probability $1 - 1/n^{\Omega(1)}$). The algorithm processes each edge (u, v) in $O(1)$ expected time and $O(\log d_{u,v}/\log \log d_{u,v})$ worst-case time where $d_{u,v}$ is the sum of the degrees of u and v. The algorithm can be generalized to weighted graphs by rounding edge weights to powers of $(1 + \varepsilon)$, constructing spanners for each edge set with the same weight, and taking the union of these spanners. This adds a factor of $O(\varepsilon^{-1} \log \omega)$ (where ω is the ratio between the biggest and smallest weights) in the space and time complexity and adds a factor of $(1 + \varepsilon)$ in the approximation factor of a distance. Elkin and Zhang [11] present various algorithms for constructing (α, β)-spanners.

If only a specific distance, $d_G(u, v)$, needs to be approximated then it may appear that constructing a spanner for the entire graph is excessive. However, this is not the case. In particular, it can be shown that any single-pass algorithm that approximates the (weighted) graph distance between two given nodes up to a factor t with probability at least 3/4 requires $\Omega(n^{1+1/t})$ space [13]. Furthermore, this bound also applies even with the promise $d_G(u,v) = \mathrm{Diam}(G)$. Consequently approximation via spanners is at most a factor 2 from optimal.

Counting Triangles: As was noted earlier, approximating the number of triangles in an undirected graph was one of the earliest problems considered in the data-stream model [2]. The number of triangles is related to the clustering and transitivity coefficients of the graph. The state of the art [4] are one-pass, randomized algorithms that estimate the number of triangles up to a multiplicative factor or $(1 + \varepsilon)$ with probability at least $1 - \delta$ using space:

Model	Space
Adjacency	$O(\varepsilon^{-2}(1 + T_1/T_3 + T_2/T_3) \log\delta^{-1})$
Incidence	$O(\varepsilon^{-2}(1 + T_2/T_3) \log\delta^{-1} \log n)$

where T_i is the number of node triples upon which the induced sub-graph has exactly i edges. While both of these space bounds can be large (e.g., for dense graphs with few triangles) this compares favorably to the

$$O(\epsilon^{-2}(1 + T_0/T_3 + T_1/T_3 + T_2/T_3) \log \delta^{-1})$$

space required by the naive algorithm that randomly samples node-triples and computes the fraction that are triangles. It can also be shown that any p-pass algorithm determining if the number of triangles is non-zero requires $\Omega(p^{-1}n^2)$ space [2]. There has also been work on estimating the count of larger cycles or cliques. However, it can be shown using results in extremal graph theory and a reduction from the set-disjointness communication problem that any p-pass algorithm that determines if $\mathrm{Girth}(G) \ge g$ requires $\Omega(p^{-1}(n/g)^{1+4/(3g-4)})$ space [13].

A related problem is that of estimating *path aggregates*. Define P_k to be the number of pairs of vertices that are joined by a simple path of length k. For a graph with r connected components, it is possible to approximate P_2 in one pass and $O(\varepsilon^{-2} m(m - r)^{-1/4} \log\delta^{-1})$ space even if edges may be deleted [14]. A space lower bound of $\Omega(\sqrt{m})$ is also known.

BFS trees: A common subroutine in many traditional graph algorithms is that of constructing a breadth-first-search (BFS) tree. Unfortunately it can be shown that computing the first l layers of a breadth-first-tree from a prescribed node requires either $\lceil (l - 1)/2 \rceil$ passes or $\Omega(n^{1+1/l})$ of space [13].

Matchings: Given a graph $G = (V, E)$, the *Maximum Cardinality Matching* (MCM) problem is to find the largest set of edges such that no two adjacent edges are selected. More generally, for an edge-weighted graph, the *Maximum Weighted Matching* (MWM) problem is to find the set of edges whose total weight is maximized subject to the condition that no two adjacent edges are selected. The following semi-streaming algorithms are known for these problems:

Weighted	Passes	Approximation Ratio	Ref.
No	1	0.5	[13]
No	$O_\varepsilon(1)$	$1 - \varepsilon$	[16]
Yes	1	0.179	[19]
Yes	$O_\varepsilon(1)$	$0.5 - \varepsilon$	[16]

The first of the above algorithms is based on a simple greedy approach, i.e., the algorithm always maintains a matching and adds the latest edge if it is not adjacent to a currently stored edge. The third and fourth algorithms are variants on this basic idea. The second algorithm is based on finding augmenting paths for a currently stored matching, i.e., odd length paths with every second edge in the currently stored matching. For each augmenting path found it is possible to increase the size of the currently stored matching by one. It can be shown that if there are only a relatively small number of $O(\varepsilon^{-1})$-length augmenting paths then the current matching is already a good approximation. Alternatively, if there are many short augmenting paths then it is possible to find a constant fraction using a randomized approach.

Degree Distributions and Random Walks: Given a stream of edges with possible duplicates, a natural question that arises is to estimate properties of the underlying graph. Work has been done on estimating the frequency moments, heavy hitter, and range sums of the degrees of this underlying graph [6]. For example, if d_v is the degree of v in the underlying graph then it is possible to approximate $M_2 := \sum_v {d_v}^2$ in one pass and $O(\epsilon^{-4}\sqrt{n}\log n)$ space. Note that many of these problem are solvable using standard techniques if there are no duplicates in the stream of edges. A related problem is to estimate the entropy of a random walk on an undirected, unweighted graph. Here the graph stream is an observation of a random walk whose states are nodes of the graph. There exists a single-pass $O(\varepsilon^{-4}\ \log^2 n\ \log^2 \delta^{-1})$ space algorithm that estimates the entropy of the walk [5]. These algorithms use a combination of algorithms for counting distinct items and the AMS sampling technique. The problem of actually constructing random walks has also been considered [7]. This has applications to estimating the page-rank vector, mixing time and conductance of graphs.

Key Applications

Massive graphs arise naturally in many real world scenarios. Two examples are the *call-graph* and the *web-graph*. In the call-graph, nodes represent telephone numbers and edges correspond to calls placed during some time interval. In the web-graph, nodes represent web pages, and the edges correspond to hyper-links between pages. When processing these graphs it is often appropriate to use the data-stream model. For example, the graph may be revealed by a web-crawler or the graph may be stored on external memory devices and being able to process the edges in an arbitrary order improves I/O efficiency. One of the major drawbacks of traditional graph algorithms, when applied to massive graphs such as the web, is their need to have random access to the edge set. Massive graphs also arise in structured data mining, where the relationships among the data items in the data set are represented as graphs, and social networks.

Future Directions

There are numerous specific open problems that arise from the existing work on graph streams. For example, one could attempt to improve the approximation factors of the known approximate matching algorithms or the space required to approximate M_2. Another idea is to explore distance approximation in multiple passes. While computing exact distance may require many passes this does not preclude the possibility of better approximation with a few additional passes.

Other more general ideas include investigating graph problems in the *probabilistic data-stream model* or the *random-order data-stream model*. In the probabilistic data-stream model, a probability p_e would be assigned to each edge e. The goal of an algorithm would be to estimate the probability that the random graph generated has a certain property given that each edge is present independently with probability p_e. In the random-order data-stream model the assumption, as the name suggests, is that the edges arrive in random order. The goal is to design algorithms that estimate some property with high probability where the probability is defined over both the coin flips of the algorithm and the ordering of the stream.

Cross-references

▶ One-Pass Algorithm
▶ Stream Models
▶ Synopsis Structure

Recommended Reading

1. Aggarwal G., Datar M., Rajagopalan S., and Ruhl M. On the streaming model augmented with a sorting primitive. IEEE Symposium on Foundations of Computer Science, 2004, pp. 540–549.
2. Bar-Yossef Z., Kumar R., and Sivakumar D. Reductions in streaming algorithms, with an application to counting triangles in graphs. In ACM-SIAM Symp. on Discrete Algorithms, 2002, pp. 623–632.
3. Buchsbaum A.L., Giancarlo R., and Westbrook J. On finding common neighborhoods in massive graphs. Theor. Comput. Sci., 1–3(299):707–718, 2003.
4. Buriol L.S., Frahling G., Leonardi S., Marchetti-Spaccamela A., and Sohler C. Counting triangles in data streams. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 253–262.
5. Chakrabarti A., Cormode G., and McGregor A. A near-optimal algorithm for computing the entropy of a stream. In ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 328–335.
6. Cormode G. and Muthukrishnan S. Space efficient mining of multigraph streams. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 271–282.
7. Das Sarma A., Gollapudi S., and Panigrahy R. Estimating PageRank on graph streams. In Proc. 27th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2008, pp. 69–78.
8. Demetrescu C., Escoffier B., Moruz G., and Ribichini A. Adapting parallel algorithms to the w-stream model, with applications to graph problems. In Mathematical Foundations of Computer Science, 2007, pp. 194–205.
9. Demetrescu C., Finocchi I., and Ribichini A. Trading off space for passes in graph streaming problems. In ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 714–723.
10. Elkin M. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. In Int. Colloquium on Automata, Languages and Programming, 2007, pp. 716–727.
11. Elkin M. and Zhang J. Efficient algorithms for constructing $(1 + \varepsilon, \beta)$-spanners in the distributed and streaming models. Distrib. Comput., 18(5):375–385, 2006.
12. Feigenbaum J., Kannan S., McGregor A., Suri S., and Zhang J. Graph distances in the data-stream model. SIAM J. Comput., 38(5):1708–1727, 2008.
13. Feigenbaum J., Kannan S., McGregor A., Suri S., and Zhang J. On graph problems in a semi-streaming model. Theor. Comput. Sci., 348(2–3):207–216, 2005.
14. Ganguly S. and Saha B. On estimating path aggregates over streaming graphs. In Int. Symp. on Algorithms and Computation, 2006, pp. 163–172.
15. Henzinger M.R., Raghavan P., and Rajagopalan S. Computing on data streams. External memory algorithms, 1999, pp. 107–118.
16. McGregor A. Finding graph matchings in data streams. In APPROX-RANDOM, 2005, pp. 170–181.
17. Muthukrishnan S. Data Streams: Algorithms and Applications. Foundations and Trends in Theoretical Computer Science, 1(2), 2005.
18. Zelke M. *k*-connectivity in the semi-streaming model. CoRR, cs/0608066, 2006.
19. Zelke M. Weighted matching in the semi-streaming model. In Proc. Symp. on Theoretical Aspects of Computer Science, 2008.

Graph Streams

▶ Graph Mining on Streams

Graph Theory

▶ Graph Data Management in Scientific Applications
▶ Information Integration Techniques for Scientific Data

Graph-based Clustering

▶ Spectral Clustering

Graphic

▶ Diagram

Graphic Design

▶ Visual Interaction

Graphic Representation of Data

▶ Data Visualization

Graphical Displays of Many Variables

▶ Multivariate Visualization Methods

Graphical Interaction

▶ Direct Manipulation

Graphical Representation

▶ Visual Representation

Graphical User Interfaces

▶ Visual Interfaces

Graphics

▶ Image

Graphics for Continuous Data

▶ Visualizing Quantitative Data

Graphics for Discrete Data

▶ Visualizing Categorical Data

Grid and Workflows

Jinjun Chen, Yun Yang
Swinburne University of Technology, Melbourne, VIC, Australia

Synonyms

Workflow on grid

Definition

Built on Internet and World Wide Web, the Grid is a new class of infrastructure which supports coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [6,7]. In a grid architecture, a grid workflow management system is a type of user-level grid middleware. It aims to support large-scale sophisticated scientific and business processes in a variety of complex e-science and e-business applications [7,10,12,15]. Such sophisticated processes are modeled or redesigned as grid workflow specifications at build-time stage by some modeling languages such as Grid Workflow Execution Language (GWEL), Abstract Grid Workflow Language (AGWL), and Martlet [4,5,9]. The specifications normally contain a large number of computation, data and/or transaction intensive activities [1,2,12]. Then, at run-time instantiation stage, grid workflow instances are created [4]. Finally, at run-time execution stage, grid workflow instances are executed by facilitating the super computing and data sharing capability of underlying grid infrastructure to complete the computation, data and/or transaction intensive activities [8,11].

Historical Background

Workflow research can be traced back to mid 1980s. In 1993, the WfMC (Workflow Management Coalition) was founded to promote the research and development of workflow technologies such as workflow reference model [13]. WfMC mainly focuses on business workflows, i.e., workflows in business applications. According to WfMC, workflow is defined as the computerized facilitation or automation of a business process, in whole or part [13].

Grid research can be traced back to mid 1990s. It aims to support the sharing of large-scale distributed and heterogeneous resources in a full and negotiable fashion. Resource providers and consumers can dynamically negotiate the sharing mode and extent. For example, they can share resources in client/server mode at one time, but then change to peer-to-peer fashion at another time [7].

With intensive research poured into grid, around the year 2000 the term grid workflow came into picture. Its basic idea is to execute large-scale complex workflows on grid by facilitating the grid's powerful computing and resource sharing capability. The main motivation for grid workflow is that many grid applications such as climate modeling and disaster recovery simulation often require the creation of a collaborative workflow management system as part of their sophisticated problem solving processes so that scientists and business people who lack the low-level expertise can still utilize the current generation of grid toolkits, such as GT4 (Globus Toolkits), Gridbus, and SwinGrid [7,14,15].

Foundations

Traditionally in the business workflow domain, a business workflow is more about logic between activities rather than computation, data or transaction intensive. As a result, there is a very modest level data transfer between activities. However, a grid workflow is often computation, data or transaction intensive. Hence, the key scientific fundamentals in grid workflow area are how to accommodate huge amount of data transfer, computation and transaction processing, and how to schedule and map to the computation capable resources in the underlying grid infrastructure [4,14]. Research and development are ongoing to tackle the key fundamentals [10,11,14,15].

Key Applications

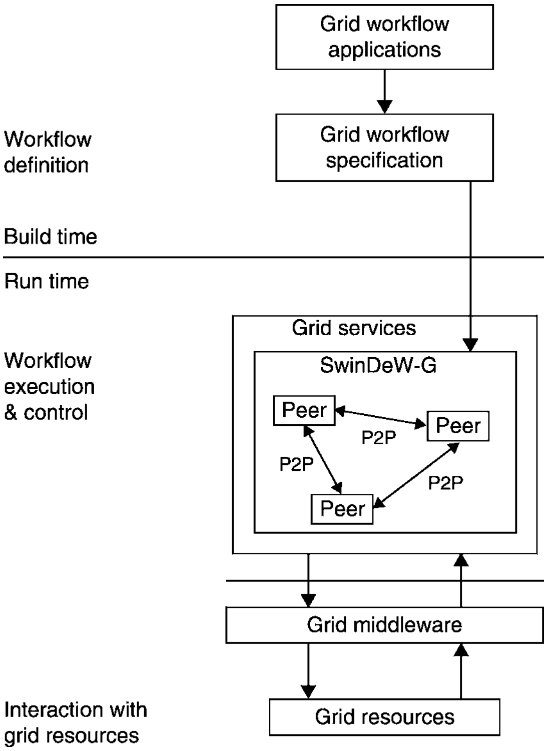
Grid workflow aims to support large-scale sophisticated scientific and business processes in a variety of complex e-science and e-business applications. Typically, such e-science applications are data and computation intensive such as climate modeling, disaster recovery simulation, earth observation data processing, and high energy physics. Such e-business applications are often transaction intensive such as bank, stock modeling, and insurance processing [7,10,12,14,15].

Future Directions

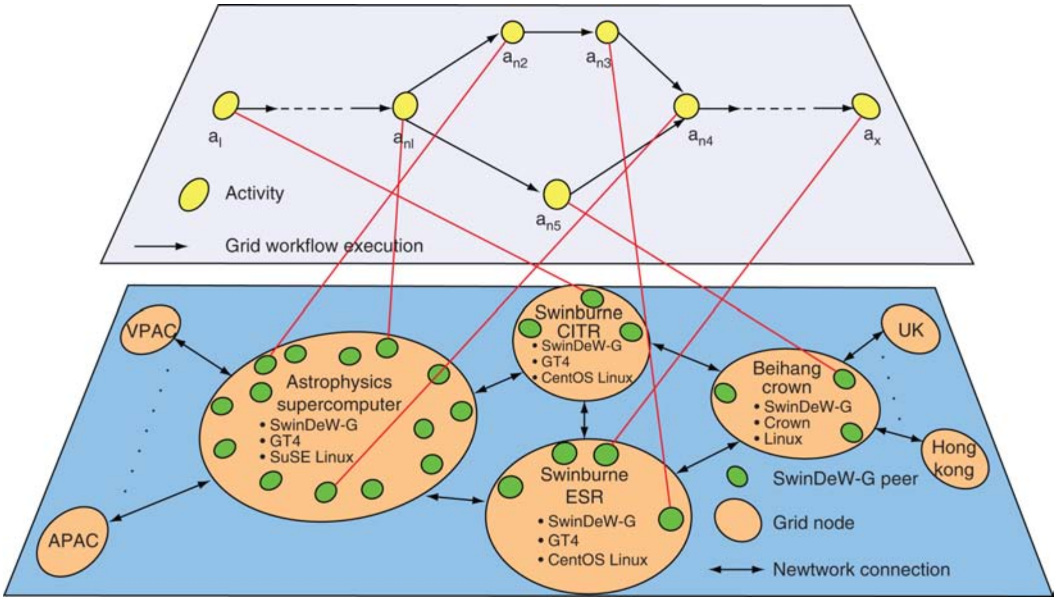
Since in a business workflow, there is a very modest level data transfer between activities, the interaction between activities is normally performed via a centralized workflow engine. However, a grid workflow is normally computation, data and/or transaction intensive. There is often huge amount of data transfer and processing between activities. Therefore, it is inefficient or even congests the centralized workflow engine if the interaction between activities is performed via the centralized engine. As such, the interaction with data transfer should be decentralized. Peer-to-peer processing becomes an ideal alternative. At the same time, the centralized monitoring and management are still necessary. Thus, in overall terms, a promising future direction can be peer-to-peer based interaction with huge amount of data transfer between activities plus centralized monitoring and management with modest information interaction between activities and the centralized engine [4,14]. In this direction, many issues need to be investigated such as global monitoring of distributed peers, workflow scheduling and modeling in support of data, computation and/or transaction intensity.

Experimental Results

The overall architecture of a grid workflow management system can be depicted in Fig. 1. Currently, many grid workflow management systems such as SwinDeW-G, Kepler, Taverna, Triana, Gridbus Workflow, and Pegasus have been developed to experimentally perform the execution of grid workflows [10,11,14,15]. Among them, SwinDeW-G (Swinburne Decentralised Workflow for Grid) as depicted in Fig. 2 is a peer-to-peer based grid workflow management system which naturally matches to the feature of computation, data and/or transaction intensity [3,14]. In SwinDeW-G,


Grid workflow applications
Grid workflow specification
Workflow definition
Build time
Run time
Grid services
SwinDeW-G
Workflow execution & control
Peer
Peer
Peer
P2P
P2P
P2P
Grid middleware
Interaction with grid resources
Grid resources

Grid and Workflows. Figure 1. Overall architecture of a grid workflow management system.



Grid and Workflows. Figure 2. SwinDeW-G system.

a grid workflow is executed by different peers that are distributed in the underlying grid infrastructure. Different peers communicate with each other directly in a peer-to-peer fashion. Kepler supports fault tolerance and p2p or centralised data transfer [10]. Taverna uses centralised approach to support data transfer and retry for fault tolerance [11]. Triana uses p2p for data transfer [15]. Gridbus Workflow also uses a centralised approach for data transfer. Pegasus supports the mapping from abstract workflow specification to specific grid resources for workflow execution [15]. There are more grid workflow management systems which can be referred to [15]. In general, many experimental practices and implementations of grid workflow management are being widely conducted.

Cross-references

- ▶ Activities
- ▶ Adaptive Workflow/Process Management
- ▶ Grid File
- ▶ Grid Storage
- ▶ Scheduler
- ▶ Scientific Workflows
- ▶ Workflow Management and Workflow Management System
- ▶ Workflow Management Coalition
- ▶ Workflow Model
- ▶ Workflow Schema

Recommended Reading

1. Abramson D., Kommineni J., McGregor J.L., and Katzfey J. An atmospheric sciences workflow and its implementation with web services. Future Generation Comput. Syst., 21(1):69–78, 2005.
2. Aloisio G., Cafaro M., Carteni G., Epicoco I., Quarta G., and Raolil A. GridFlow for earth observation data processing. In Proc. 2005 Int. Conf. on Grid Computing and Applications, 2005, pp. 168–176.
3. Chen J. and Yang Y. Adaptive selection of necessary and sufficient checkpoints for dynamic verification of temporal constraints in grid workflow systems. ACM Trans. Auton. Adapt. Syst., 2(2):Article 6, 2007.
4. Cybok D. A Grid Workflow Infrastructure. Concurrency comput. Pract. Experience (Special Issue on Workflow in Grid Systems), 18(10):1243–1254, 2006.
5. Fahringer T., Pllana S., and Villazon A. A-GWL: Abstract grid workflow language. In Proc. 4th Int. Conf. on Computational Science, LNCS 3038, 2004, pp. 42–49.
6. Foster I., Kesselman C., and Tuecke S. The anatomy of the grid: enabling scalable virtual organizations. Int. J. Supercomput. Appl., 15(3):200–222, 2002.
7. Foster I., Kesselman C., Nick J., and Tuecke S. The physiology of the grid: an open grid services architecture for distributed systems integration. In Proc. 5th Global Grid Forum Workshop, 2002.
8. Fox G.C. and Gannon D. Concurrency Comput. Pract. Exp. (Special Issue: Workflow in Grid Systems). 18(10):1009–1019, 2006.
9. Goodman D. Introduction and evaluation of Martlet, a scientific workflow language for abstracted parallelisation. In Proc. 16th Int. World Wide Web Conference, 2007, pp. 983–992.
10. Ludäscher B., Altintas I., Berkley C., Higgins D., Jaeger-Frank E., Jones M., Lee E., Tao J., and Zhao Y. Scientific workflow

management and the Kepler system. Concurrency Comput. Pract. Exp., 18(10):1039–1065, 2006.

11. Oinn T., Greenwood M., Addis M., Alpdemir M.N., Ferris J., Glover K., Goble C., Goderis A., Hull D., Marvin D., Li P., Lord P., Pocock M.R., Senger M., Stevens R., Wipat A., and Wroe C. Taverna: Lessons in creating a workflow environment for the life sciences. Concurrency Comput. Pract. Exp., 18(10):1067–1100, 2006.
12. Simpson D.R., Kelly N., Jithesh P.V., Donachy P., Harmer T.J., Perrott R.H., Johnston J., Kerr P., McCurley M., and McKee S. GeneGrid: a practical workflow implementation for a grid based virtual bioinformatics laboratory. In Proc. of the UK e-cience All Hands Meeting, 2004, pp. 547–554.
13. Workflow Management Coalition: The Workflow Reference Model. TC00–1003, 1995, http://www.wfmc.org.
14. Yang Y., Liu K., Chen J., Lignier J., and Jin H. Peer-to-peer based grid workflow runtime environment of SwinDeW-G, In Proc. 3rd IEEE Int. Conf. on e-Science and Grid Computing, 2007, pp. 51–58.
15. Yu J. and Buyya R. A taxonomy of workflow management systems for grid computing. J. Grid Comput., 3(3):171–200, 2005.

Grid File

▶ Hash-based Indexing

Grid File (and Family)

Apostolos N. Papadopoulos[1],
Yannis Manolopoulos[1], Yannis Theodoridis[2],
Vassilis Tsotras[3]
[1]Aristotle University of Thessaloniki, Thessaloniki,
Greece
[2]University of Piraeus, Piraeus, Greece
[3]University of California at Riverside, Riverside,
CA, USA

Definition

The Grid File is a *multidimensional indexing* scheme capable to efficiently index database records in a symmetrical manner, i.e., by avoiding the distinction between primary and secondary keys. The structure is dynamic and adapts gracefully to its contents under insertions and deletions. A single record retrieval costs two disk accesses at most (upper bound), whereas range queries and partial match queries are also executed efficiently. The Grid File can be thought of as a generalization of dynamic hashing (e.g., *extendible hashing*) in multiple dimensions.

Historical Background

Until the 1980s many file structures for the processing of single attribute queries have been proposed, i.e., queries on the primary key or any secondary key for which a corresponding index has been built. Multi-attribute queries are the ones where the user seeks objects that satisfy constraints (such as equality or range) on several attributes. Such queries can be executed by accessing all the corresponding indices (if they exist) and combine the partial results, or resort to sequential scanning.

To speed up the processing of multiple attribute queries, a better solution is to create an index that leads the search directly to the objects of interest. Such an index can be designed if a data record with k attributes is envisioned as a point in a k-dimensional space. A multi-attribute range query would then be a hyper-rectangle in this k-dimensional space and the answer to it would be all points inside this rectangle. Access methods that can handle multidimensional points are called *Point Access Methods* (PAMs). In 1975, Bentley proposed such a basic PAM, which is called the k-dimensional tree or k-d tree [2]. The Grid File is yet another structure designed to handle similar cases, proposed by Nievergelt, Hinterberger, and Sevcik in 1984 [10]. Since then, several variations have been proposed in the literature in an effort to optimize its space and time performance behavior.

Foundations

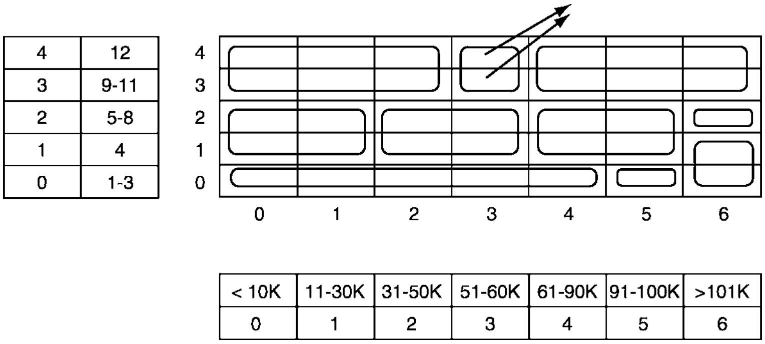
The Grid File can be viewed as an access method comprising of two separate parts: (i) the *directory*, and (ii) the *linear scales*. To illustrate this, assume that an index is to be constructed on the *Employee* file using two attributes, say *salary* and *dept* (extension to more dimensions is straightforward). The Grid File imposes a grid on the two-dimensional attribute space. Each cell in this grid corresponds to one data page. The data points that "fall" inside a given cell are stored in the cell's corresponding page. Each cell must thus store a pointer to its corresponding page. This information is stored in the Grid File's *directory*. However, cells that are empty do not use a page. Rather, two or more cells can share a page (i.e., point to the same page). The grid adapts to the data density by introducing more divisions in areas where there are more points.

The information of how each dimension is divided (and thus how data values are assigned to cells) is kept through *linear scales*. There is one linear scale per dimension (indexed attribute). Each linear scale is a one-dimensional array that divides the values on a particular dimension in such a way that records (points) are uniformly distributed across cells. An example of a Grid File on the "Dept" and "Salary" attributes appears in Fig. 1.

Searching for a record with given attribute values involves two operations: (i) the Grid File's directory is searched to locate the cell that the record is hosted in, (ii) the cell's pointer is followed to access the corresponding data page (say *A*), and (iii) the record is searched only in data page *A*. If the record is found in *A* then the search terminates successfully, otherwise the search for the specific record is unsuccessful (i.e., the record does not exist). The Grid File can also address multi-dimensional *range queries* by selecting the appropriate cells from each dimension's linear scale. For example, such a query may ask for all employee records with the *salary* attribute ranging between 10K and 40K and the *dept* attribute ranging between 2 and 6. Again, the first step examines the directory and determines the cells that are intersected by the query range in both attributes, then the corresponding pointers to data pages are collected and finally the data pages are examined for relevant records. The accessed cells may also contain some records outside the query range. These records are eliminated from further consideration and they are not returned as part of the query answer.

Inserting a new record in this method is straightforward. First, the two linear scales are searched so as to map the record's *salary* and *dept* attribute values in each dimension. This mapping provides a cell in the directory. This cell is then accessed and using its pointer, the appropriate page, say *A*, for the new record is found. If this page has enough space to accommodate the new record, the insertion process is complete. Otherwise, a new page *B* is allocated. If page *A* was pointed to by more than one cell, the pointers of these cells are rearranged such that some will point to page *A* and some to page *B* (and the records of page *A* are redistributed accordingly between *A* and *B*). If page *A* was pointed by a single cell and overflows, a reorganization of the Grid File is needed. This reorganization will expand the directory and the scales by introducing a new column (or row) of cells.

In the sequel, the insertion process is illustrated by an example given in Fig. 2 White dots correspond to existing records, whereas black dots are used to indicate new records being inserted to the Grid File. Assume that each data page can host at most three records. Practically, this number is larger in real applications and depends on the size of the data page and the number of attributes. Assume that initially the Grid File is empty (does not contain any records). The first three records can be easily accommodated in the single data page *A*pointed by the single cell of the directory (corresponding to the whole data space), as illustrated in Fig. 2(a). The next inserted record is *d*. However, the new record cannot be hosted by data page *A* because its capacity is exceeded. Therefore, another data page *B* is allocated and records are distributed to the two data pages as it is shown in Fig. 2 (b). The next two insertions for records *e* and *f* do not cause any reorganization since the new records can be easily accommodated in the corresponding data pages pointed by the cells, as illustrated in Fig. 2(c). Finally, the insertion of record *g* causes an overflow in data page *A*. The corresponding cell is split again using the other attribute and one more data page is allocated and records are distributed accordingly. The final shape of the Grid File is given in Fig. 2(d).



4	12
3	9-11
2	5-8
1	4
0	1-3

< 10K	11-30K	31-50K	51-60K	61-90K	91-100K	>101K
0	1	2	3	4	5	6

Grid File (and Family). Figure 1. A Grid File.

Deletions are also supported, but they are handled differently. Initially, the deleted record is located using the directory and the corresponding data page is determined. If the record is found, it is deleted from the data page. Instead of overflowing, data page deletions may cause the underutilization effect, which means that several data pages may contain too few records. Therefore, appropriate merging operations are required to maintain the storage utilization of the Grid File at an acceptable level. For a detailed description of the methods used for merging as well as for splitting, the reader is directed to [10].

The Grid File has a set of nice properties: (i) it is based on simple mechanisms for insertion, deletion, and search; (ii) it guarantees only two disk accesses for exact match queries (one for the directory and one for the data page); and, (iii) it treats all indexed attributes symmetrically which leads to simple directory management policies. However, it has a set of serious disadvantages such as: (i) it introduces a space overhead for the directory, which can be large for high-dimensional spaces, (ii) it has an extra update overhead, since reorganization affects many cells and not only the cell with the overflowing page; and, (iii) it suffers from performance degradation if the attributes are correlated, since the uniform scheme for performing splits is not adequate to guarantee performance efficiency.

Toward improving the behavior of the Grid File several research efforts have been performed. Some of these efforts are highlighted in the following lines. One of the first variations, the BANG File, has been proposed by Freeston [4]. The BANG File is based on a self-balanced tree-based directory, which better reflects changes of the data distribution. To achieve better storage utilization the Twin Grid File has been proposed by Hulflesz, Six, and Widmayer in [5]. The new scheme is as efficient as the original Grid File during range query processing but shows significant improvements regarding storage utilization. Blanken et al. proposed the Generalized Grid File [3], which offers fast access for single attribute queries. The Multilevel Grid File [12] is another research effort to improve the performance of the original structure for exact-match, partial-match, and range queries. This new scheme uses multiple grid levels and succeeds in better directory management and more efficient query processing than the original structure.

In addition to the variations proposed in the literature, there are efforts to use the Grid File in a parallel environment, toward more efficient data management. In [13] the authors study the problem of partitioning a Grid File to multiple disk devices toward more efficient search. When a data page split is performed, the new data page is carefully allocated to a disk. Since disks can be accessed in parallel, several data pages can be read simultaneously during range query processing, offering significant performance improvements in comparison to a single-disk system. More complex queries on Grid Files, like *spatial joins*, have been also parallelized [6] toward reduced query response times. A different approach has been followed by [7]. The authors have proposed a method to load a Grid File in parallel. The data file is initially partitioned to the available processors using dynamic programing and sampling, and then each processor builds its own part of the Grid File.

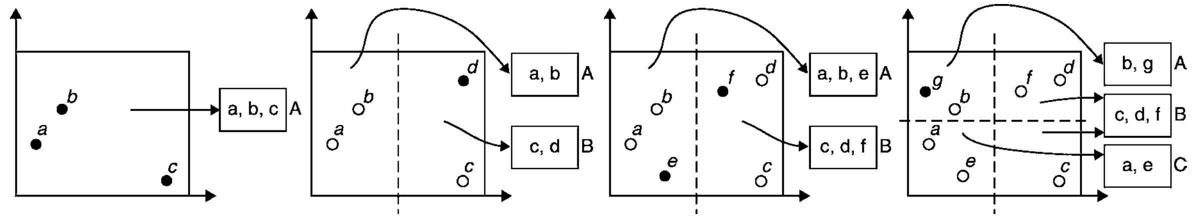
Key Applications

Spatial Databases

In Spatial Databases it is commonly required to join spatial data sets or perform nearest neighbor searches. Several algorithms have been proposed for such operations by adopting the Grid File as the underlying access method [1].

Data Mining

The Grid File can also be used for clustering data sets to identify correlation characteristics of the underlying value space. This stems from its ability to group



Grid File (and Family). Figure 2. Insertions in the Grid File.

patterns into blocks and cluster them with respect to the blocks by a topological neighbor search algorithm [11].

Data Warehouses

The Grid File can be used for efficient data cube storage in warehouses [9].

Future Directions

The Grid File has eventually emerged as a popular theoretical access method. However, although it has been widely honored in theory, in practice it has not been used by the database industry.

Experimental Results

A detailed performance evaluation of the Grid File can be found in [10], where the authors offer a detailed experimental section studying the properties of the structure regarding capacity of data pages, directory size, and evaluation of splitting and merging policies. Moreover, interesting experimental results can be found in [5,3], which compare the original Grid File with the corresponding variation proposed in each work.

Cross-references

- ▶ Extendible Hashing
- ▶ K-D trees
- ▶ Multidimensional Indexing
- ▶ Range Query
- ▶ Spatial Join

Recommended Reading

1. Becker L., Hinrichs K., and Finke U. A new algorithm for computing joins with grid files. In Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 190–197.
2. Bentley J.L. Multidimensional binary search trees used for associative searching. Commun. ACM, 18(9):509–517, 1975.
3. Blanken H.M., Ijbema A., Meek P., and van den Akker B. The generalized grid file: description and performance aspects. In Proc. 14th Int. Conf. on Data Engineering, 1990, pp. 380–388.
4. Freeston M. The BANG file: a new kind of grid file. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 260–269.
5. Hutflesz A., Six H.-W., and Widmayer P. Twin grid files: space optimizing access schemes. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988, pp. 183–190.
6. Kim J.-D. and Hong B.-H. Parallel spatial join algorithms using grid files. In Proc. 6th Int. Conf. on Database Systems for Advanced Applications, 1999, pp. 226–234.
7. Li J., Rotem D., and Srivastava J. Algorithms for loading parallel grid files. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 347–356.
8. Lim Y. and Kim M. A Bitmap Index for Multidimensional Data Cubes. In Proc. 15th Int. Conf. Database and Expert Syst. Appl., 2004, pp. 349–358.
9. Luo C., Hou W.C., Wang C.F., Wang H., and Yu X. Grid File for Efficient Data Cube Storage. Computers and their Applications, conference paper CATA, pp. 424–429, 2006.
10. Nievergelt J., Hinterberger H., and Sevcik K.K. The grid file: an adaptable, symmetric multikey file structure. ACM Trans. Database Syst., 9(1):38–71, 1984.
11. Schikuta E. and Erhart M. The BANG-clustering system: grid-based data analysis, In Adv. in Intelligent Data Analysis, 2nd Int. Symp., 1997, pp. 513–524.
12. Whang K.-Y. and Krishnamurthy R. The multilevel grid file – a dynamic hierarchical multidimensional file structure. In Proc. 2nd Int. Conf. on Database Systems for Advanced Applications, 1991, pp. 449–459.
13. Zhou Y., Shekhar S., and Coyle M. Disk allocation methods for parallelizing grid files. In Proc. 10th Int. Conf. on Data Engineering, 1994, pp 243–252.

Grid Workflow

▶ Scientific Workflows

Group Difference

▶ Emerging Patterns

Grouping

▶ Abstraction

GUID

▶ Resource Identifier

GUIs for Web Data Extraction

CAI-NICOLAS ZIEGLER
Siemens AG, Munich, Germany

Synonyms

Visual web data extraction; Wrapper generator GUIs; Visual web information extraction

Definition

While content management systems (CMS) are geared towards *adding* presentational information to relational and structured data from database systems, thus dynamically generating HTML documents, the goal of GUIs for Web data extraction is diametrically opposed: The commonly semi-automatic Web data extraction tools intend to *remove* all presentational information from Web pages, so that only pure structured content remains. The extraction process itself does not address single documents, but template types, such as the product page of an online retailer or the news page template of an online journal. That is, for each template type, one set of extraction rules is generated. These extraction rules are defined in a graphical manner, by selecting the pieces of information that are relevant and by assigning labels to them. To this end, GUIs are used that largely resemble Web browsers, extended by user actions for highlighting and assigning appropriate labels. For instance, for a book seller's product page, such labels were "author," "publisher," "price," or "number of pages."

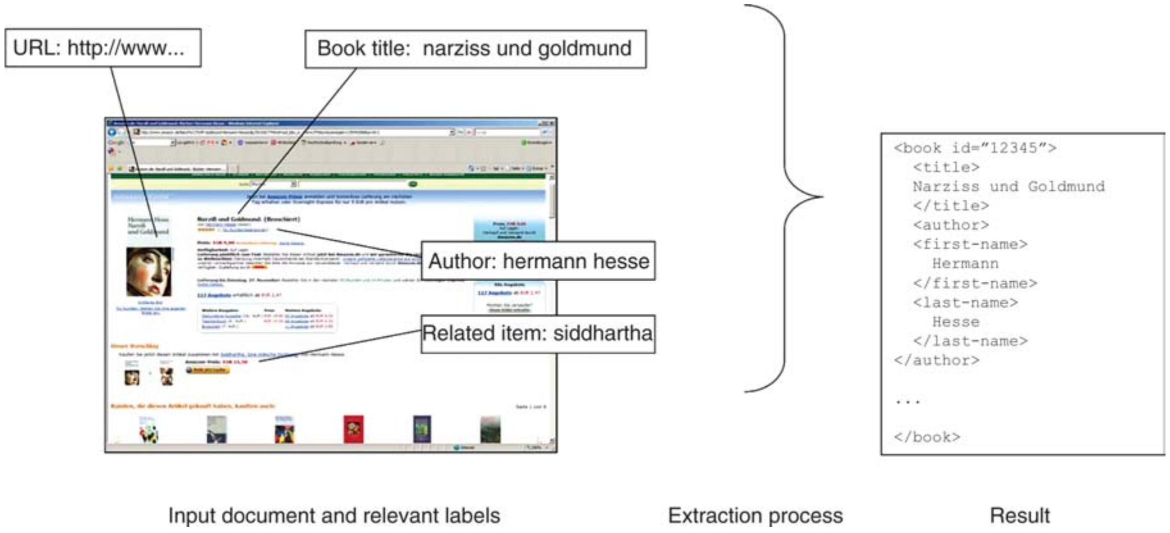
The learned extraction rules for one document template, called "wrapper" [5], are then applied to those documents that have been generated based on the template type at hand. Referring to the book seller's example, one could now extract structured relational information for each book published on the online retailer's website (see Fig. 1).

Historical Background

GUIs for Web data extraction are advanced variants of wrapper induction systems [5]. Being graphics-oriented, these follow-ups add to the state of art by making the extraction process more user-friendly and easier to manage, saving time and making Web data extraction accessible to a broad range of non-experts as well.

Early Web data extraction systems have been implemented by writing small programs in scripting languages like Perl. These programs, termed screen scrapers, contained manually defined extraction rules that focused on structural communalities of information types to be extracted from the documents, e.g., price, title, and publisher. The huge disadvantage was that people had to look at the very HTML code of documents, comparing several instances of the document template used and trying to figure out which sequences of HTML documents and/or textual content remained static and which appeared dynamic.

The tedious nature of screen scraping favored the naissance of wrapper induction systems. These applications did not require the human to manually write extraction rules, but to label a number of documents and have machines inductively learn the extraction rules. Early seminal systems include Kushmerick's wrapper generator [5], Stalker [6,7], and RoadRunner [4]. The latter system takes HTML documents that have been generated using the same template as input



```
<book id="12345">
  <title>
  Narziss und Goldmund
  </title>
  <author>
  <first-name>
    Hermann
  </first-name>
  <last-name>
    Hesse
  </last-name>
</author>

...

</book>
```

GUIs for Web Data Extraction. Figure 1. Web data wrappers extract relevant information from Web documents and store the labeled results into data structures, such as XML documents.

and infers a most specific grammar that subsumes the variety of all of them.

At the same time, early GUI-based extraction systems have begun to evolve. NoDoSe [1] extracts information from plain string sources and provides a user interface for instance labeling. Kushmerick proposed Wien [1], a visual support tool, as an extension to his early wrapper system. Wien receives a set of training pages where the user can label relevant information and learn a wrapper.

The shift from research towards commercial tools became manifest with the advent of Lixto [2], which started off as an academic project and transformed into a commercial product soon after. (See *http://www.lixto.com* for details). The early Lixto system allowed the user to mark relevant, information-bearing passages in a browser-like window, assign labels to these passages, e.g., "price," "manufacturer," etc., test the wrappers on unseen data, and modify the learned extraction rules. Lixto incorporated a system for inductive extraction rule learning based on the ELOG [3] language, which bears traits of both PROLOG and XPATH (see Fig. 2). The extracted and labeled result was written to an XML document.

Complex enterprise-scale commercial solutions have evolved since then, boasting graphical user interfaces that allow for recording HTML form inputs and interaction events in a macro-like fashion, handling dynamic and AJAX-enhanced documents, and creating complex processing and wrapping workflows.

Foundations

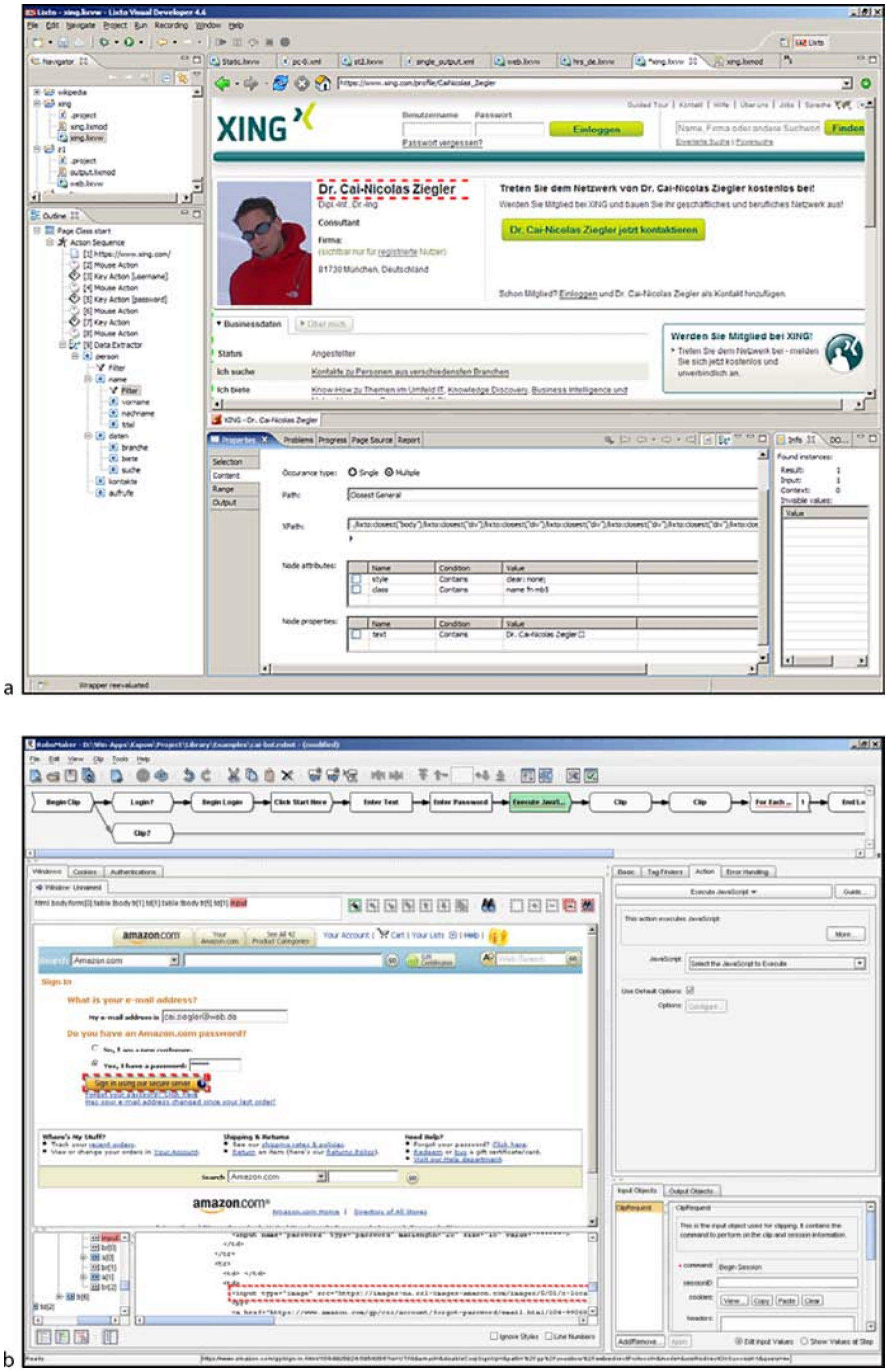
With the advent of graphical user interfaces for Web data extraction, the focus of supervised wrapper generation has greatly changed from one relying on inductive machine learning techniques to one that intends to offer the non-expert user a broad range of easy-to-use actions that allow him to specify what he intends to extract in a comfortable and 100% WYSIWYG fashion. The goal is not so much the use of intelligent techniques for the generalization and conjunction of extraction patterns as much more to cover the full content extraction and publication process, by means of an end-to-end lifecycle approach. This includes the easy interfacing to databases for the parameterization of wrapper input data and output storage, the availability of APIs and Web service interfaces in order to embed the extraction process as yet another component or service of a larger Web application infrastructure, as well as the ability to design wrappers that can be re-used as subordinate components in other wrappers.

At the time of this writing, the major vendors of commercial state-of-the-art GUIs for Web data extraction are Kapow (*http://www.kapow.com*), Lixto (*http://www.lixto.com*), Denodo (*http://www.denodo.com*), and Twinsoft (*http://www.twinsoft.fr*). All products offered exhibit a similar look and feel, with an HTML browser window being the central component that all other dialogues and tool frames are clustered around (see Fig. 3).

Typically, the user can indicate regions that contain relevant information by highlighting the respective region with his mouse. Moreover, most GUIs offer the possibility to navigate, in another window, the DOM tree of the HTML document at hand and select the nodes that match the demanded criteria. Next, the user can assign labels to these selected information regions and thus generate extraction rules based on path expressions. Some tools, such as Lixto Visual Developer, attempt to generalize the matching extraction paths in an intelligent fashion, assuming wildcards for appropriate location steps so as to avoid over-fitting [3]. In order to handle Dynamic HTML and cascading stylesheets (CSS), most graphical

```
 tablesq(S,X)  ← document("www.ebay.com/",S),subsq(S,(.body,[]),(.table,[]),(.table,[]),X),
                 before(S,X,(table,[elementtext,item,substr]),0,0,-,-,-),after(S,X,.hr,0,0,-,-)
  record(S,X)  ← tableseq(-,S),subelem(S,.table,X)
 itemnum(S,X)  ← record(-,S),subelem(S,*.td,X),notbefore(S,X,.td,100)
 itemdes(S,X)  ← record(-,S),subelem(S,(*.td.*.content,[(a,,substr)],X)
   price(S,X)  ← record(-,S),subelem(S,(*.td,[(elementtext,\var[Y].*,regvar)],X),isCurrency(Y)
    bids(S,X)  ← record(-,S),subelem(S,*.td,X),before(S,X,.td,0,30,Y,-)price,(-,Y)
currency(S,X)  ← price(-,S),subtext(S,\var[Y],X), is Currency(Y)
 pricewc(S,X)  ← price(-,S),subtext(S,[0-9]+\.[0-9]+,X)
```

GUIs for Web Data Extraction. Figure 2. Generated ELOG rules for extracting relevant information, such as item number, description, price, and bids, from ebay (*http://www.ebay.com*).



GUIs for Web Data Extraction. Figure 3. Screenshots of two state-of-the-art commercial GUIs for Web data extraction, (a) Lixto Visual Developer and (b) Kapow Mashup Server.

extraction systems have a built-in JavaScript interpretation engine. Debugging capabilities are also state-of-the-art, allowing the user to step through the entire extraction process, from the wrapper's entering of the website to the eventual extraction and storage.

Commercial products for Web data extraction have matured over the years and have become user-friendly and extremely powerful tools. From a scientific perspective, though, there is little new technology to discover.

Key Applications

The number of applications for wrapper generation GUIs is rife and extends to all areas where information extraction on the Web is put to use. Integration of Web data is surely the most important application scenario and heavily advertised by solution vendors. However, instead of referring to "integration," the synonymous term "mashup" is used, which hints at the deliberate placement of such tools into the Web 2.0 universe. For instance, the GUIs could be used to create extraction wrappers for news articles from various sources, which are then presented in one page. Another mashup application scenario is the creation of value-added services that exploit heterogeneous information sources: For instance, the implementation of a travel service that acts as meta search engine for other travel search engines, integrates their results into a single list, and enriches each entry by resorting to another wrapper which obtains information for the destination country at hand from encyclopedic websites such as Wikipedia (*http://www.wikipedia.com*).

Cross-references

► Fully-Automatic Web Data Extraction
► Information Extraction
► Information Filtering
► Wrapper Induction

Recommended Reading

1. Adelberg B. NoDoSE: A tool for semi-automatically extracting structured and semi-structured data from text documents. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 283–294.
2. Baumgartner R., Flesca S., and Gottlob G. Visual web information extraction with lixto. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 119–128.
3. Baumgartner R., Flesca S., and Gottlob G. The ELOG web extraction language. In Proc. Artificial Intelligence on Logic for Programming, 2001, pp. 548–560.
4. Crescenzi V., Mecca G., and Merialdo P. RoadRunner: towards automatic data extraction from large web sites. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 109–118.
5. Kushmerick N., Weld D., and Doorenbos R. Wrapper induction for information extraction. In Proc. 15th International Joint Conference on Artificial Intelligence, 1997, pp. 119–128.
6. Muslea I., Minton S., and Knoblock C. Stalker: learning extraction rules for semistructured, web-based information sources. In Proc. of the AAAI Workshop on AI and Information Integration, 1998.
7. Muslea I., Minton S., and Knoblock C. Hierarchical wrapper induction for semistructured information sources. Auton. Agent. Multi Agent Syst., 4(1–2):93–114, 2001.

H

Handhelds Interfaces

▶ Mobile Interfaces

Handwritten Text

▶ Electronic Ink Indexing

Hanf-Locality

▶ Locality of Queries

Hard Disk

▶ Disk

Hardware-Conscious Database System

▶ Architecture-Conscious Database System

Harmonic Mean of Recall and Precision

▶ F-Measure

Hash File

▶ Hash-based Indexing

Hash Filter

▶ Bloom Filters

Hash Filter Join

▶ Semijoin

Hash Functions

Marina Blanton
University of Notre Dame, Notre Dame, IN, USA

Synonyms

Cryptographic hash functions; One-way hash functions

Definition

A hash function is a well-defined deterministic algorithm that takes as input data of arbitrary length and produces a short fixed-length digital representation of the data, or a digest, as its output. The output of a hash function can serve the role of a digital "fingerprint" of the input data, as an important design property of hash functions is that of collision resilience: two hashes produced on different inputs are very unlikely to result in the same value. Furthermore, given a hash function output, it is normally infeasible to find a (previously unseen) input that matches that output (this property is called preimage resistance).

Key Points

Hash functions have many uses, and in cryptography they are widely used because of their collusion resistance and preimage resistance properties. In particular, hash functions are used to verify integrity of messages and can be used to construct *message authentication codes*. They are also used in many cryptographic protocols and to construct cryptographic primitives (e.g., to construct symmetric encryption schemes and in digital signatures) [4].

The two most commonly used hash functions are MD5 [3] and SHA-1 [1], but a variety of other algorithms is also available. Both MD5 and SHA-1, however, should be used with caution because a recent development of attacks indicate that MD5 is too weak to meet the necessary cryptographic properties and it may become possible to also attack SHA-1 in the near future. SHA-2 [2] is a collection of new hash functions from the SHA family (such as SHA-256, SHA-224, SHA-384, and SHA-512) which were designed by the National Security Agency (NSA) to be a new standard.

Cross-references

▶ Digital Signatures
▶ Merkle Hash Trees
▶ Message Authentication Codes

Recommended Reading

1. Eastlake D. and Jones P. US Secure Hash Algorithm 1 (SHA1). IETF RFC 3174, 2001. http://www.ietf.org/rfc/rfc3174.txt.
2. National Institute of Standards and Technology (NIST). FIPS 180-2: Secure Hash Standard (SHS), Current version of the Secure Hash Standard (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512), 2004.
3. Rivest R. The MD5 message-digest algorithm. IETF RFC 1321, 1992. http://www.ietf.org/rfc/rfc1321.txt.
4. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code (2nd edn.). Wiley, New York, NY, 1996.

Hash Join

Jingren Zhou
Microsoft Research, Redmond, WA, USA

Synonyms

Hash join

Definition

The hash join is a common join algorithm in database systems using hashing. The join predicate needs to be an equality join predicate. The classic algorithm consists of two phases: the "build" phase and the "probe" phase. In the "build" phase, the algorithm builds a hash table on the smaller relation, say *R*, by applying a hash function to the join attribute of each tuple. In the "probe" phase, the algorithm probes the hash table using tuples of the larger relation, say *S*, to find matches.

Key Points

The classic algorithm is simple, but it requires that the smaller join relation fits into memory. If there is no enough memory to hold all the tuples in *R*, an additional "partition" phase is required. There are several variants of the classic hash join algorithm. They differ in terms of utilizing memory and handling overflow.

Grace Hash Join The idea behind grace hash join is to hash partition both relations on the join attribute, using the *same* hash function. As the result, each relation is hashed into *k* partitions, and these partitions are written to disk. The key observation is that *R* tuples in partition *i* can join only with *S* tuples in the same partition *i*. If any given partition of *R* can be hold in memory, the algorithm can read in and build a hash table on the partition of *R*, and then probe the hash table using tuples of the corresponding partition of *S* for matches.

If one or more of the partitions still does not fit into the available memory (for instance, due to data skewness), the algorithm is recursively applied. An additional orthogonal hash function is chosen to hash the large partition into sub-partitions, which are then processed as before.

Hybrid Hash Join The hybrid hash join algorithm is a refinement of the grace hash join algorithm which takes advantage of more available memory. To partition *R* (*S*) into *k* partitions, the grace hash join uses one input buffer for reading in the relation and *k* output buffers, one for each partitions.

```
Algorithm 1: Grace Hash Join: R $\bowtie_{r=s}$ S
// partition R into k partitions
foreach R ∈ R do
  read R and add it to buffer page $h_1$ (R);
  flush the page to disk if full;
end
// partition S into k partitions
foreach S ∈ S do
  read S and add it to buffer page $h_1$ (S);
  flush the page to disk if full;
end
// "build" and "probe" phases
for $i \leftarrow 1$ to $k$ do
  foreach R ∈ partition $R_i$ do
    read R and insert into the hash table using $h_2$ (R);
  end
  foreach S ∈ partition $S_i$ do
    read S and probe the hash table using $h_2$(S);
    for matching R tuples, add {R , S} to result;
  end
  clear the hash table and release the memory;
end
```

Suppose there are enough extra memory to hold an in-memory hash table for one partition, say the first partition, of *R*, the hybrid hash join does not write the partition to disk. Instead, it builds an in-memory hash table for the first partition of *R* during the "partition" phase. Similarly, when partitioning *S*, for the tuples in the first partition of *S*, the algorithm directly probes the in-memory hash table and writes out the results. At the end of the "partition" phase, the algorithm completes the join between the first partitions of *R* and *S* while partitioning the two relations. The algorithm then joins the remaining partitions as the grace hash join algorithm does.

Compared with the grace hash join algorithm, the hybrid hash join algorithm avoids writing the first partitions of *R* and *S* to disk during the "partition" phase and reading them in again during the "build" and the "probe" phases.

Cross-references

- ▶ Hashing
- ▶ Parallel Join Algorithms
- ▶ Evaluation of Relational Operators

Recommended Reading

1. Mishra P. and Eich M.H. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, 1992.

Hash Trees

▶ Merkle Trees

Hash-based Indexing

Mirella M. Moro[1], Donghui Zhang[2],
Vassilis J. Tsotras[3]
[1]Federal University of Minas Gerais, Belo Horizonte, Brazil
[2]Northeastern University, Boston, MA, USA
[3]University of California, Riverside, CA, USA

Synonyms

Hash file; External hashing; Extensible hashing; Linear hashing; Grid file

Definition

Consider a relation *R* with some attribute *A* taking values over domain *D*. A *membership* (or *equality*) *query* retrieves all tuples in *R* with $A = x$ $(x \in D)$. To enable fast processing of such equality selection queries, an access method that can group records by their value on attribute *A* is needed. A hash-based scheme maps the search-key values on a collection of buckets. The bucket to which a value is assigned (mapped) is determined by a function, called the *hashing function*.

Key Points

A major performance goal of a database management system is to minimize the number of I/O's (i.e., blocks or pages transferred) between the disk and main memory when answering a query. To achieve such fast access, additional data structures called access methods (or indices) are designed per database file. There are two fundamental access methods, namely tree-based and hash-based indexing. They differ on the kind of queries that they can efficiently address.

Hash-based indexing does not maintain any ordering among the indexed values; rather it is based on mapping the search-key values on a collection of buckets. Therefore it can only address *equality* (or *membership*) queries. Tree-based indices maintain order and can thus also address range queries.

In database management systems hashing is implemented on disk-resident data (also termed as *external hashing*). Here instead of a slot the scheme uses a *bucket* which is typically the size of a page (or a fixed number of pages) and can thus hold many records. Consider a hashing scheme using *M* buckets. Assume a page can store *B* values. A *hashing function h* is a function that maps *D* to *M* (one example of such function is $h(x) = x \ modM$). To insert a record with search-key x $(x \in D)$, the hashing function $h(x)$ is computed which gives the bucket to store that record. In *static* external hashing, if this bucket has space the record is simply added. If *B* values have already been hashed on this bucket, an *overflow* occurs. An extra page is added (and chained to this bucket) and the new record is stored there.

Given that the number of possible values in domain *D* is typically much larger than *M*, different values can be hashed on the same bucket. Nevertheless, a given hashing function will always hash a given value on the same bucket. Searching for record(s) with a given search-key y (also called *hash-key*) is simple. The hashing function is computed on the *y* and *h(y)* will be the bucket accessed. The record(s) with the requested value will either be found on this bucket or there is no such record in the file. No other buckets need to be searched. Note that searching a hash-based scheme avoids the tree navigation that a tree-index implies. Rather, a hashing function is computed (in main memory) and the only I/O is for bringing in main memory the page(s) of a given bucket.

In static external hashing, the number of buckets is pre-allocated and does not change whether the file is small or large. In contrast, *dynamic* hashing schemes (like *extensible* and *linear* hashing) use a number of buckets that increases/decreases as the size of the indexed file changes by record insertions/deletions.

The cost for updating and querying a hash-based hashing scheme is constant in the expected case (but can be linear in the worst case, for example when most records are mapped to one bucket creating a long overflow chain of pages); its space requirements are linear to the size of the indexed file.

Another hashing related scheme is the Bloom filter, a space-efficient probabilistic data-structure that drastically reducing space by allowing false positives (but not false negatives). For multi-attribute search, the Grid-File has been proposed, which can be considered as an extension of hashing into many dimensions.

Cross-references

- ▶ Access Methods
- ▶ Extendible Hashing
- ▶ Grid File
- ▶ Indexing
- ▶ Linear Hashing

Recommended Reading

1. Manolopoulos Y., Theodoridis Y., and Tsotras V.J. Advanced Database Indexing. Kluwer, Dordrecht, 1999.
2. Elmasri R. and Navathe S.B. Fundamentals of Database Systems, 5th edn. Addisson-Wesley, Reading, MA, 2007.
3. Ramakrishnan R. and Gehrke J. Database Management Systems, 3rd edn. McGraw-Hill, New York, 2003.

Hater's Model

▶ Two-Poisson model

HCC

▶ Human-centered Computing: Application to Multimedia

HCI

▶ Human-Computer Interaction

HCM

▶ Human-centered Computing: Application to Multimedia

Health Informatics

▶ Taxonomy: Biomedical Health Informatics

Healthcare Informatics

▶ Taxonomy: Biomedical Health Informatics

Heat Map

▶ Visualizing Clustering Results

Heavy Hitters

▶ Frequent Items on Streams

Heterogeneous Distributed Database Systems

▶ Distributed Database Systems

Heterogeneously Distributed Data

▶ Vertically Partitioned Data

HHH

▶ Hierarchical Heavy Hitter Mining on Streams

Hidden-Web Search

▶ Deep-Web Search

Hierarchial Clustering

Maria Halkidi
University of Piraeus, Piraeus, Greece

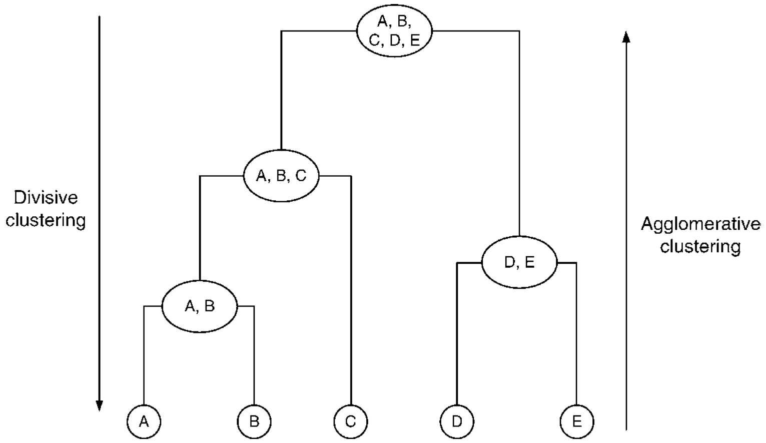
Definition

A *hierarchical clustering* method generates a sequence of partitions of data objects. It proceeds successively by either merging smaller clusters into larger ones, or by splitting larger clusters. The result of the algorithm is a tree of clusters, called dendrogram (see Fig. 1), which shows how the clusters are related. By cutting the dendrogram at a desired level, a clustering of the data items into disjointed groups is obtained.

Historical Background

Hierarchical clustering is one of the main methods used in data mining to partition a data collection. A number of hierarchical clustering algorithms [3] have been developed to deal with various types of data and application requirements. According to the method that the algorithms produce clusters, can be classified into the following categories:

1. *Agglomerative hierarchical clustering.* The algorithms of this category follow a bottom-up strategy and they produce sequence of data clusterings by decreasing number of clusters at each step (resulting in a different tree level). The algorithm starts by considering each data object as a separate cluster. Then, at each step it selects to merge the pair of two closest clusters defining thus a new clustering. To find the similarity of two clusters, one of the following criteria is typically used: *the minimum, the maximum,* or *the average distance* between objects of the two clusters. The merging procedure proceeds iteratively until all objects are in a single cluster or a termination condition is satisfied.
2. *Divisive hierarchical clustering.* These algorithms produce a sequence of clusterings increasing the number of clusters at each step. Contrary to the agglomerative, the divisive algorithms produce at each step a clustering from the previous one by splitting a cluster into two. The algorithm starts with all objects in a single cluster. Then it iteratively subdivides the clusters into smaller ones, until each object forms a separate cluster or a cetrain termination condition is satisfied.



Hierarchial Clustering. Figure 1. Dendogram.

One of the drawbacks of pure hierarchical clustering methods is their inability to perform adjustment when a merge or split decision has been executed. Once a group (cluster) of data is merged or split, the clustering process at the next step will operate on the newly generated clusters. Thus the quality of hierarchical clustering depends on the selection of merge or split point. To tackle this problem the recent studies have focused on the integration of hierarchical agglomerative with multiple phase clustering techniques and relocation methods.

Foundations

This section provides an overview of the main algorithms that are representatives of hierarchical clustering method.

BIRCH [6] uses a hierarchical data structure called CF-tree for partitioning the incoming data objects in an incremental and dynamic way. CF-tree is a height-balanced tree, which stores the clustering features. A clustering feature (CF) is a structure that summarizes statistics for the given sub-clusters. Considering a set $X = \{X_1, \ldots, X_N\}$ of d-dimensional data objects assigned into a sub-cluster C_i, the CF of C_i is defined as $CF = \{N, LS, SS\}$, where $LS = \sum_{i=1}^{N} X_i$ is the linear sum on N objects and $SS = \sum_{i=1}^{N} X_i^2$ is the square sum of data objects.

The BIRCH algorithm is based on two parameters: *branching factor B* and *threshold T*, which refer to the diameter of a cluster (the diameter (or radius) of each cluster must be less than T). It consists of two phases:

1. Scan of the data collection to *build an initial in-memory CF tree*. It can be viewed as a multilevel compression of data that tries to preserve the inherent clustering structure of the data.
2. Application of a clustering algorithm to *cluster the leaf nodes of the CF tree.*

BIRCH adopts an incremental clustering method since the CF tree is built dynamically as new objects are inserted. It can typically find a good clustering with a single scan of the data and improve the quality further with a few additional scans. Thus using a multi-phase clustering BIRCH tries to produce the best clusters with the available resources. However, BIRCH does not always correspond to a natural cluster (as a user may consider it), since each node in CF-tree can hold a limited number of entries due to its size. Also it uses the notion of radius or diameter to control the boundaries of a cluster and thus it tends to favor spherical clusters. Moreover, it is order-sensitive as it may generate different clusters for different orders of the same input data. The computational complexity of BIRCH to cluster a set of n objects is $O(n)$.

CURE [1] is a hierarchical clustering algorithm that combines centroid-based and representative object based approaches. CURE represents each cluster by a certain number of objects that are generated by selecting well-scattered points and then shrinking them toward the cluster center by a specified fraction α. These representative points try to capture the natural schema and the geometry of a cluster. Additionally, moving the dispersed points toward the cluster center by a certain factor (further reffered to as α) the algorithm aims to remove the noise and eliminate the influence of outliers. Significant movements of outliers will eliminate the possibility of merging inappropriate clusters. A high value of α shrinks the representatives closer to the cluster center and thus it favors more compact clusters. On the other hand, a small value of α shrinks more slowly the representatives favoring elongated clusters. Thus CURE efficiently achieves to identify arbitrarily shaped clusters (i.e., non spherical) and it is robust to the presence of outliers.

CURE scales for large databases using a combination of random sampling and partition clustering. The data that are used as input to the algorithm, can be a sample randomly selected from the original data set. The selected data sample is partitioned into a certain number of partitions and then each of the defined partitions is partially clustered. A clustering step follows that aims to generate a hierarchy of partial clusters. The clustering starts considering each data object of input (partial clusters) as a separate cluster and then it iteratively merges the nearest pairs of clusters. The distance between two clusters is defined as the distance between their closest representatives. Thus only the representative points of clusters are used to measure the distance between them. Then the representative points falling in each of the newly defined clusters are moved toward the center of clusters by a shrinking factor α.

The time complexity of CURE is $O(n^2 \, log \, n)$ (where n is the number of data objects to be clustered), while it reduces to $O(n^2)$ in case of low-dimensional data.

ROCK [2], is a robust hierarchical clustering algorithm for Boolean and categorical data. It introduces two new concepts: (i) the *neighbors of a point* and (ii) *links*. The clustering algorithm is based on these concepts to measure the similarity/proximity between a pair of data points. Given a user-defined threshold $\theta \in [0,1]$, a pair of points p_i, p_j are defined to be neighbors if $sim(p_i, p_j) \geq \theta$. Also the term $link(p_i, p_j)$ is defined to be the number of common neighbors between p_i and p_j. The ROCK algorithm exploits the concept of *links* to make decisions about the clusters that will be merged at each step. The similarity between a pair of clusters (C_i, C_j) is measured based on the number of points from C_i and C_j that have neighbors in common. For a pair of clusters C_i and C_j, the algorithm measure the number of cross links between the clusters, that is, $\sum_{pk \in C_i \text{ and } pl \in C_j} links(p_k, p_l)$. Then the *goodness measure* for merging clusters C_i, C_j is defined as follows:

$$goodness(C_i, C_j) = \frac{link(C_i, C_j)}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

Then, at a given step, the algorithm selects to merge the pair of clusters for which the goodness measure is maximum.

CHAMELEON [4] is a clustering algorithm that explores characteristics of dynamic modeling in hierarchical clustering. It is an agglomerative hierarchical algorithm that measures the similarity of two clusters based on a dynamic model. Specifically, it defines the clusters in the data set by using a two-phase algorithm. During the first phase, CHAMELEON uses a graph-clustering algorithm to partition a data set into a large number of relatively small sub-clusters. During the second phase, it uses an agglomerative hierarchical clustering algorithm to find the clusters by repeatedly combining together these sub-clusters. The similarity between clusters is determined by looking at their relative *inter-connectivity* and *relative closeness*. The representation of the data objects is based on the widely used k-nearest neighbor graph approach. The vertices of the graph represents data objects, and there is an edge between two vertices if data object corresponding to one of the nodes is among the k-most similar data objects of the other node. Then the algorithm finds the initial sub-clusters using a graph-partitioning algorithm in order to partition the k-nearest neighbor graph of the considered data set into a large number of partitions. During the next phase CHAMELEON switches to an agglomerative hierarchical clustering that combines together these small sub-clusters. It measures the similarity between each pair of considered clusters based on their *relative inter-connectivity* and *closeness of the sub-clusters*:

1. The *relative interconnectivity* between a pair of clusters C_i and C_j is their absolute interconnectivity normalized with respect to their internal interconnectivities. It is given by

$$RI(C_i, C_j) = \frac{\left| EC_{\{C_i, C_j\}} \right|}{\frac{1}{2}(|EC_{C_i}| + |EC_{C_i}|)}$$

   where $EC_{\{C_i,C_j\}}$ is defined as the sum of weights of the edges that connect vertices in C_i to vertices in C_j and EC_{C_i} is defined as the weighted sum of the edges that partition the graph into two roughly equal parts.
2. The relative closeness between a pair of clusters C_i and C_j is the absolute closeness normalized with respect to the internal closeness of the two clusters:

$$RC(C_i, C_j) = \frac{\overline{S}_{EC_{\{C_i,C_j\}}}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{EC_{C_j}}}$$

   where $\overline{S}_{EC_{\{C_i,C_j\}}}$ is defined as the average weight of the edges that connect vertices in C_i to vertices in C_j, and $\overline{S}_{EC_{C_i}}$ is the average weight of edges that partition the graph into two roughly equal parts.

According to the above definition CHAMELEON makes merging decisions. Those pairs of clusters whose relative inter-connectivity and closeness are above some user-specified threshold are merged. CHAMELEON has been proved to be more powerful at discovering arbitrarily shaped clusters than CURE. However, the processing cost for high dimensional data may require quadratic time to the number of processed objects ($O(n^2)$).

The most recent clustering algorithm that combines characteristics of hierarchical clustering and graph-theory is C^2P [5]. It exploits index structures, and the processing of the *Closest-Pair* queries in spatial databases. The algorithm considers that the data points can be organized in an R-tree data structure to facilitate the searching of the nearest representative point. Then the Closest-Pair Query (CPQ) is used to find the closest pair of points from two datasets indexed with two R-tree data structures.

The C^2P algorithm consists of two main phases. The first phase (Phase I) organizes a set of objects into a number of sub-clusters, which are an effective representation of the final clusters. The *Self Semi-Closest-Pair* Query (Self-Semi-CPQ) is used to find pairs of objects (p, p') that belong to a data set S such as $dist(p, p') = min_{\forall x \in S}\{dist(p, x)\}$. The algorithm uses a graph representation that organizes the proximity information computed by the CPQ. Using the Depth-First Search the algorithm can efficiently find the *c* connected components of the graph, which also comprises the sub-clusters of the data set. All objects that belong to the same connected component can be considered as a sub-cluster. When the number of defined sub-clusters, is equal to the required number of sub-clusters the Phase I terminates. Otherwise, the algorithm finds the center of each sub-cluster to represent it. Then the previously described procedure is iteratively applied to the set of *c* cluster centers until the required number of sub-clusters is defined. The second phase is a specialization of the first phase using a different cluster representation so as to produce the finer final clustering. The second phase (Phase II) has as input the centers of sub-clusters defined in Phase I. At each iteration of Phase II, Self-CPQ finds the closest pair of clusters by finding the closest pair among their representatives. Then these two clusters are merged and the *r* data objects, among all the objects of the merged clusters, that are closest to the cluster center are selected as representatives of the new cluster. Using multi-representatives instead of the center, C^2P can effectively capture the shape and size of the clusters. The procedure terminates when the required number of clusters is reached. The above description shows that Phase II operates in a fashion analogous to a hierarchical agglomerative clustering algorithm.

The C^2P algorithm is shown to scale well to large databases. Its time complexity for a dataset with *n* objects is $O(n \log n)$.

Key Applications

Hierarchical clustering has applications in various fields of real world. In biology, it can be used to define taxonomies, categorize genes with similar functionality and gain insights into structures inherent in populations. Clustering may help to automate the process of analyzing and understanding spatial data. It is used to identify and extract interesting characteristics and patterns that may exist in large spatial databases. Also hierarchical clustering is used to discover significant groups of documents on the Web's huge collection of semi-structured documents. This classification of Web documents assists in information discovery.

Cross-references

- ▶ Balanced Trees
- ▶ Clustering Methods
- ▶ Graph Clustering
- ▶ Indexing
- ▶ k-Nearest Neighbors
- ▶ R-Tree

Recommended Reading

1. Guha S., Rastogi R., and Shim K. CURE: an efficient clustering algorithm for large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 73–84.
2. Guha S., Rastogi R., and Shim K. ROCK: a robust clustering algorithm for categorical attributes. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 512–521.
3. Han J. and Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann, 2001.
4. Karypis G., Han E.-H., and Kumar V. CHAMELEON: a hierarchical clustering algorithm using dynamic modeling. IEEE Comput., 32(8):68–75, 1999.
5. Nanopoulos A., Theodoridis Y., and Manolopoulos Y. *C^2P:* clustering based on closest pairs. In Proc. 27th Int. Conf. on Very Large Daa Bases, 2001, pp. 331–340.
6. Zhang T., Ramakrishnman R., and Livny M. BIRCH: an efficient method for very large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 103–114.

Hierarchical Data Model

Jean-Luc Hainaut
University of Namur, Namur, Belgium

Synonyms

IMS data model

Definition

The hierarchical data model is based on a view of the application domain (i.e., the *world*) as a hierarchical arrangement of concepts where some concepts exist on their own while the others depend on the former. According to this conceptual model, data are organized into records that are recursively composed of other records. Though this paradigm is fairly common in data structures, the term hierarchical model most generally refers to the *IMS model*, a proprietary DBMS developed by IBM from the sixties that is still widely used.

The IMS model organizes data in tree structures of records augmented with additional links that compensate for the weaknesses of this base model. Data processing itself is hierarchical, starting from the root of a record tree then parsing the dependent records in depth-first, left-to-right, traversal order.

Historical Background

In 1966, IBM started the development of ICS (Information Control System), a data management software for, and with the collaboration of, American Rockwell, then in charge of building the Apollo spacecraft which was to send men on the moon. IBM commercialized ICS in 1969 under the name IMS (Information Management System) [1].

The first version offered storage and processing facilities for data records organized in trees and linearized according to the depth-first traversal order. The data could be stored on tapes and on disks and were processed sequentially. Thanks to the addressability of magnetic discs, IMS was later given direct access (through hashing and B-tree techniques) to root records and therefore was able to support transaction processing.

At that time, tree structures were strictly independent and not connected, a fact that led to much redundancy (a record cannot be stored in two different trees without duplication). IBM then introduced logical relationship types which explicitly linked records from different tree structures. A record could then be shared by several trees.

Later on, IMS was added secondary indexes that provided direct access to non-root records. IMS is now a complex and powerful data management and data communication environment mostly used by data-intensive batch and On-Line Transaction Processing (OLTP) applications.

Though System 2000 (SAS Institute), XML document structures (DTD and XML Schema) and standard file structures can legitimately claim to belong to the hierarchical family, the presentation will focus on the IMS model. Data description and manipulation languages, API and implementation techniques will not be addressed in this chapter. They can be found in references [2, 4–6].

Foundations

Due to historical reasons, in particular the incremental development of IMS, the description of its data structures is generally intricate. In this entry, they will be described by way of a simpler approach based on graph theory. Of course, some very specific details will be ignored.

Graphs and Hierarchies

This section relates database schemas with various kinds of graphs. Due to the limited scope of this chapter, only an intuitive view of the equivalence principles will be developed. Is-a hierarchies, in particular, will be ignored.

Preliminary Definitions

A database schema can be seen as a *multigraph*, whose nodes denote entity types and whose labeled edges represent binary relationship types (*rel-types* for short). This applies to any Entity-relationship schema, provided n-ary rel-types have been replaced by relationship entity types through some kind of *reification* transformation (such as T1 in Fig. 1). In this graph, edges are given a pictorial representation that specifies the functional dependencies that hold in the rel-type according to the usual arrow convention. A single arrow denotes a N:1 rel-type, a double arrow a 1:1 rel-type and a plain edge a N:N rel-type. N:1 and 1:1 rel-types are called *functional* as well as the edges that represent them.

A *hierarchical graph* is a binary graph such that (i) the edges are functional and (ii) the edges define a partial order on the nodes. In practical words, there is no chain of successive N:1 edges the starting and arrival nodes of which are the same node (in short, *no circuit*). Considering a directed edge drawn from *B* (the *many* side) to *A* (the *one* side), *A* is called a *parent* of *B* while *B* is a *child* of *A*. *A* is called a *k-node* if it is the child of *k* parents. A graph is a *n-hierarchy* if none of its nodes has more than n parents. A 0-node is a *root* of its hierarchy. A hierarchy comprises at least one root. A node that is not a parent is a leaf node. A *1-hierarchy* is a set of trees, that is, a forest.

Properties of Hierarchies

1. *Any multigraph can be losslessly transformed into a hierarchy.* Three patterns can prevent an arbitrary schema from defining a hierarchy, namely complex rel-types (n-ary, with attributes), N:N rel-types

and rel-types that form a circuit. N-ary and N:N rel-types can be processed by transformations T1 and T2 respectively (Fig. 1). Any functional rel-type of a circuit transformed through T3 or T4 (Fig. 1) destroys this circuit. All these entity-generating transformations have been proved to preserve the semantics of the schema [3]. A large number of hierarchies can be derived from a definite multigraph.

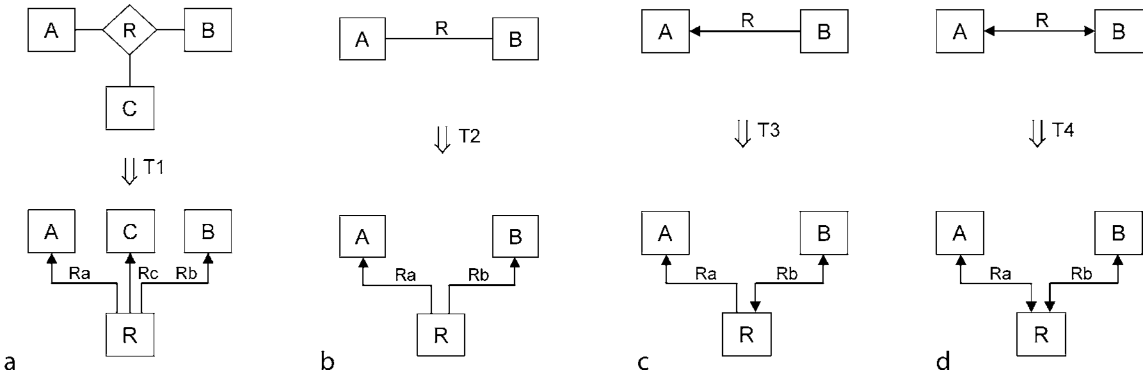
2. *Any hierarchy can be losslessly transformed into a 2-hierarchy.* Transformation T3 shows that *B*, which is the child of *A* in the source schema looses its parent (which becomes its "spouse") in the target schema. A similar transformation exists for 1:1 rel-types (T4). Iteratively applying this transformation produces an equivalent 2-hierarchy from an arbitrary hierarchy (Fig. 2). This process is not unique, so that many 2-hierarchies can be derived from a given hierarchy.
3. *Any 2-hierarchy can be built by superimposing two forests on the same set of nodes.* For each child node of a 2-hierarchy *H* with two parents, one of the parental edges is colored in black and the other one in gray. Each of the remaining edges is arbitrarily colored in black or in gray. The set of nodes together with black (resp. gray) edges form the black (resp. gray) subgraph(s). Each of them is a forest and their union is *H*. There are many ways to color the edges of a given 2-hierarchy (Fig. 3).
4. *Any arbitrary Entity-relationship schema is equivalent to the superimposition of two forests.* Properties 1 and 2 show that any Entity-relationship schema is equivalent to a 2-hierarchy, which in turn can be decomposed into two forests.

IMS Data Structures

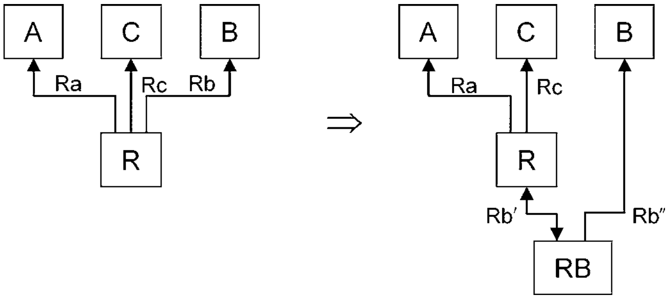
IMS is by far the most popular DBMS based on the hierarchical data model. This section describes its main data structures.

The IMS Global Schema

The global schema of an IMS database can be modeled by a 2-hierarchy formed by two distinct forests. The nodes represent *segment types* (the IMS name for record types) and the edges N:1 *parent-child rel-types*. The black rel-types are called *physical rel-types* and the



Hierarchical Data Model. Figure 1. Transforming arbitrary relationship types into functional relationship types.



Hierarchical Data Model. Figure 2. Reduction of a *n*-hierarchy to a *n-1*-hierarchy by application of transformation T3.

gray ones *logical rel-types* (this naming is purely historical and bears little meaning at this level). According to the *color* of the rel-type that links them, two segment types will be called respectively physical/logical child and physical/logical parent. A rel-type bears no name. If two rel-types link the same pair of segment types, one of them is physical and the other one is logical. IMS schemas use graphical conventions that are close to those of Figs. 1–3 but their edges have no arrow heads (Fig. 4).

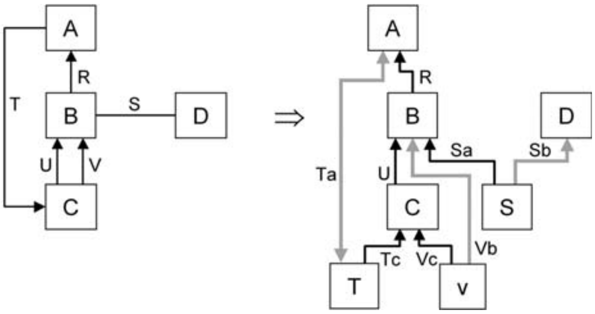
IMS imposes additional constraints of this double tree structure:

1. a logical child must be a physical child; therefore a root segment type cannot be a logical child;
2. a logical child cannot be a logical parent;
3. a logical child cannot have a physical child that is also a logical child.

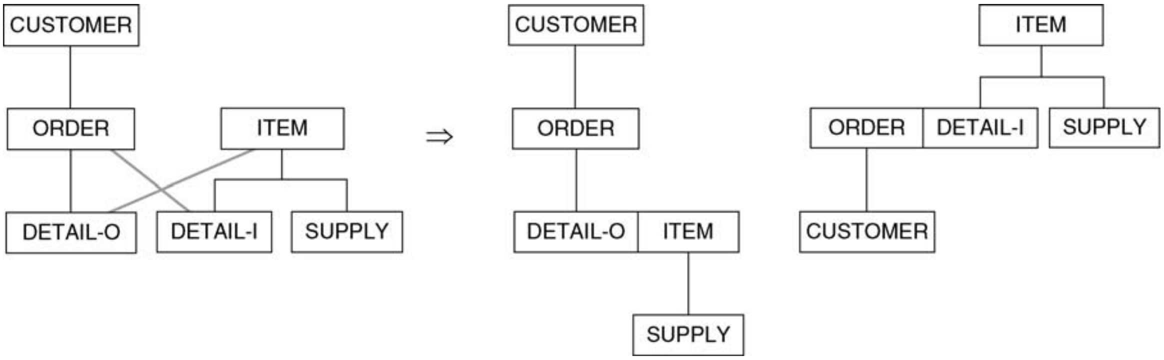
As a consequence, many logical children are leaf physical segment types (Fig. 3 – right).

A segment type has a name and a length. It is made up of fields. A field is an untyped byte string with which a name, a starting position and a length are associated. Some parts of the segment type may be left undefined, while some fields may overlap, which is a way of describing compound fields. In summary, a segment type comprises single-valued mandatory fields that can be atomic or compound.

Now only the physical (black) rel-types are considered. Each 0-node denotes a *physical root segment type*. Each physical root segment type, together with all the segment types to which it is directly or transitively connected through a physical rel-type, form a *physical database*. A root segment followed by all its direct and indirect child segments, in the depth-first order traversal (or preorder), is a *physical record* of this database. A physical database contains a sequence of physical records. One of the fields of the root segment type is the *sequence field* or *key*. No two records can have the same key value and the records are sorted on this key. Direct access is allowed to a record based on its key value. A field of a dependent segment type can also be declared a key. In this case, and unless otherwise stated, the children segments of a definite



Hierarchical Data Model. Figure 3. Any multigraph is equivalent to the superimposition of two forests.



Hierarchical Data Model. Figure 4. Two examples of logical database schemas.

parent have distinct values of their key. The key value of a segment prefixed by the concatenated key of its parent forms its *concatenated key*. The latter identifies this segment in the physical database. The implicit goal of a physical database is twofold:

- historically, a physical record collects all the data that are needed by an application program (process-oriented approach);
- according to the modern way of perceiving a database, a physical record collects the proper data of a major entity of the application domain, independent of the programs that will use it (domain-oriented approach).

An *IMS database* comprises one or several physical databases together with all their logical rel-types. Logical rel-types can be defined between two segment types pertaining to one or two physical databases.

The IMS Logical Database

A *logical database* is a virtual tree data structure derived from an IMS global database schema and tailored to the specific needs of an application program. Its role is similar to that of CODASYL subschemas and of relational views. The main derivation rules are the following:

1. the logical database schema *L* comprises a connected subset of the physical database schema *PH*;
2. the root of *L* is the root of *PH* (secondary indexes allow more flexible structures);
3. the logical parent *P* of a logical child *C* of *L* can be concatenated with *C*, giving fictitious segment type *CP*;
4. all or some of the direct and transitive physical children of *P* can be attached to *CP*;
5. all or some of the direct and transitive physical parents of *P* can be attached to *CP* as fictitious children.

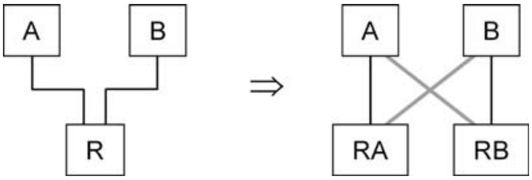
Fig. 4 shows some examples of logical database schemas derived from the same IMS database.

Additional Constructs

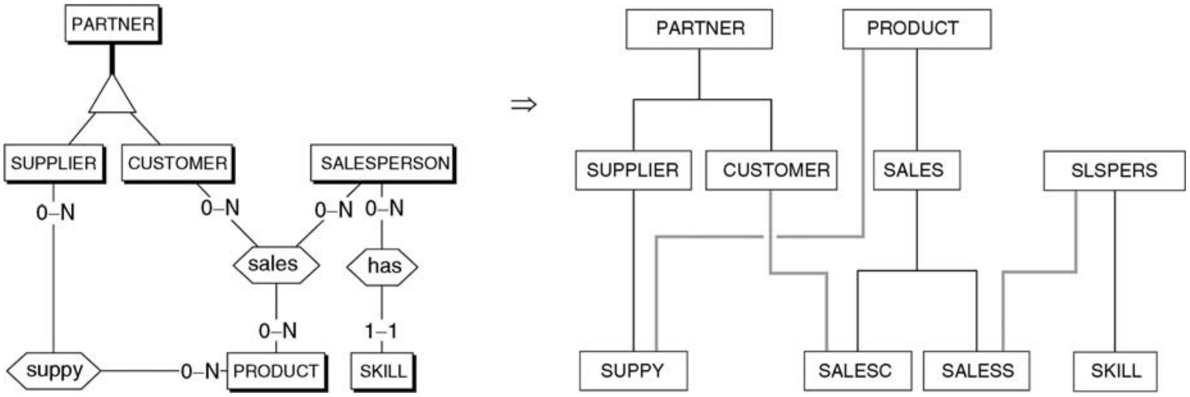
Several other concepts are part of the IMS model and are devoted large sections in usual IMS presentations. They have not been included in the preceding sections inasmuch as they do not contribute to the understanding of the data model but rather appear, according to today's conceptual standards, as minor idiosyncrasies. Two of them are briefly described, namely *segment pairing* and *secondary indexes.*

Segment pairing. Consider the hierarchical schema fragment of Fig. 5 (left), in which *relationship* segment type *R* materializes a N:N link between *A* and *B*. Though this pattern is intuitive from the conceptual and physical points of view, it violates the spirit of tree structures that underlies the hierarchical model of IMS: for bidirectional access, both *A* and *B* must have a normal (physical) child of their own. So, each of them is given a physical child, namely *RA* and *RB*, and it is linked to the other parent through a logical (gray) rel-type. However, since *RA* and *RB* are just clones of each other, they are declared *paired* (Fig. 5 – right). Physically, this pattern can be implemented according to two techniques: with a single child segment type (virtual pairing) or with two redundant child segment types whose contents are synchronized automatically (physical pairing).

Secondary indexes. Physical (or logical) records can be directly accessed through indexing techniques applied to the root sequence field. A secondary index allows additional access based on other fields of the root segment type or access to non root segment types. It is implemented as a special-purpose physical database, the *index database*, made up of one segment type (the *pointer* segment type), that collects all the values of the key fields together with the addresses of the segments that include these values. In fact, this technique is based on two segment types in the indexed database. The indexed values are extracted from the *source segment type* while the access is performed on the *target segment type*. These segment types are generally the same but the former can be the child of the latter. For instance, a secondary index can allow access to *CUSTOMER* segments that have at least one *ORDER* child segment with a given date of order. In addition, a logical database can be built on an index database in such a way



Hierarchical Data Model. Figure 5. Normalization of a *relationship segment type* according to the IMS model.



Hierarchical Data Model. Figure 6. Partial translation of a representative Entity-relationship schema (left) into a hierarchical schema (right).

that its logical root segment type is any, possibly non-root, segment type of the indexed database.

Entity-Relationship to Hierarchical Mapping

As shown above, an Entity-relationship schema can always be translated into an equivalent 2-hierarchy, a structure that is close to the IMS data model. The translation is mostly based on a unique technique, that is, the transformation of a rel-type into a relationship entity type together with two or several functional rel-types. The procedure of deriving an IMS database schema from an Entity-relationship schema can be sketched as follows.

1. Each entity type is represented by a segment type, each attribute by a field and each functional rel-type by a parent-child rel-type (still *uncolored*).
2. A non functional rel-type is transformed into a segment type by techniques T1 and T2.
3. Is-a hierarchies are best transformed through the *one segment type per entity type* technique; subtypes are represented by physical children of their super-type segment type.
4. A circuit is opened by the transformation of one of its links by technique T3 and T4.
5. 1:1 rel-types are implemented as standard 1:N rel-types and controlled by the application programs.
6. The schema is then transformed into a 2-hierarchy by the technique illustrated in Fig. 3. When a segment type has two parents, two techniques can be used. The first one consists in marking one rel-type as *logical.* The second one applies transformations T3 or T4. The latter will be preferred when IMS structural rules concerning logical rel-types are violated or to make the schema more balanced (rel-type *RA* of the schema of Fig. 3 could be processed in the same way as *RB*).

Figure 6 illustrates some of these principles. The IMS global schema comprises three physical databases and three logical relationship types. For readability, segment pairing, according to which each logical child is duplicated as a dependent of its respective logical parent, is not shown.

Key Applications

IMS is a major legacy technology in which many large corporate databases are still implemented. It is mainly used for stable, slowly evolving, batch and OLTP applications, notably in banking companies. The complexity of the hierarchical model and its lack of flexibility in evolving domains make IMS technology less attractive for decisional applications, such as data warehouses.

Cross-references

- ▶ Database Management System
- ▶ Entity-Relationship Model
- ▶ Network Data Model
- ▶ Relational Model

Recommended Reading

1. Blackman K. IMS celebrates thirty years as an IBM product. IBM Syst. J., 37(4):596–603, 1998.
2. Elmasri R. and Navathe S. Fundamentals of Database Systems (3rd edn.). Addison-Wesley, 2000. (The appendix on the hierarchical data model has been removed from later editions but is now available on the authors' site.)

3. Hainaut J-L. The transformational approach to database engineering. In Generative and Transformational Techniques in Software Engineering, R. Lämmel, J. Saraiva, J. Visser (eds.). Springer, New York, NY, 2006, pp. 89–138.
4. Long R., Harrington M., Hain R., and Nicholls G. IMS Primer – IBM Redbooks, 2000.
5. Meltz D., Long R., Harrington M., Hain R., and Nichols G. An Introduction to IMS. IBM Press, Armonk, NY, 2005.
6. Tsichritzis D. and Lochovsky F. Hierarchical data-base management: a survey. ACM Comput. Surv.(Special Issue: Data-Base Management Systems), 8(1):105–124, 1976.

Hierarchical Data Organization

► Indexing and Similarity Search

Hierarchical Data Summarization

Egemen Tanin
University of Melbourne, Melbourne, VIC, Australia

Synonyms

Hierarchical data summarization

Definition

Given a set of records data summaries on different attributes are frequently produced in data management systems. Commonly used examples are the number of records that fall into a set of ranges of an attribute or the minimum values in these ranges. To improve the efficiency in accessing summaries at different resolutions or due to a direct need for investigating a hierarchy that is inherent to the data type, such as dates, hierarchical versions of data summaries can be used. A data structure or algorithm is labelled as hierarchical if that structure or algorithm uses the concept of subcomponents to systematically obtain conceptually larger components. The method of obtaining a larger component is regularly induced by the user's understanding of the domain, such as dates in a year, as well as the fact that hierarchies can also be created automatically by a set of rules embedded into the system. Thus, rules used in a data structure's creation, e.g., B+-trees, are also considered as a means for hierarchical data summarization. In fact, different variants of popular data structures are used in hierarchical data summarization. Various algorithms for data reduction and aggregation have also adopted hierarchical processing techniques.

Historical Background

From a data structures point of view, foundations of hierarchical data summarization (HDS) techniques can be found in indexing literature for databases. Although many of the indexing techniques, e.g., B+-trees, are used for efficiently selecting records stored on a disk, they can also be considered as hierarchical summaries on large amounts of data. For multidimensional and spatial data, indices such as R-trees and quadtrees can be used for HDS.

Today, many versions of popular indexing techniques that directly target retrieval of summary information exist. Some indices are also used in query optimization due to their HDS capabilities, e.g., using a space decomposition one can guess the number of records in a certain region of data before a join operation can take place. More recently, spatial indexing techniques, for example quadtrees, were developed for distributed settings such as sensor networks for HDS.

Historically, histograms are the most basic structures that could be used for data summarization. They are frequently utilized in query optimization decisions. They are also used in data warehousing. Hierarchical versions of histograms were recently built and are of interest for HDS.

From an algorithmic point of view, techniques such as wavelet transformations, sketches, and data clustering with aggregation, when run in a hierarchical fashion, can be considered as HDS techniques. These techniques are extensively deployed in data management as well as in other fields of computer science over many years.

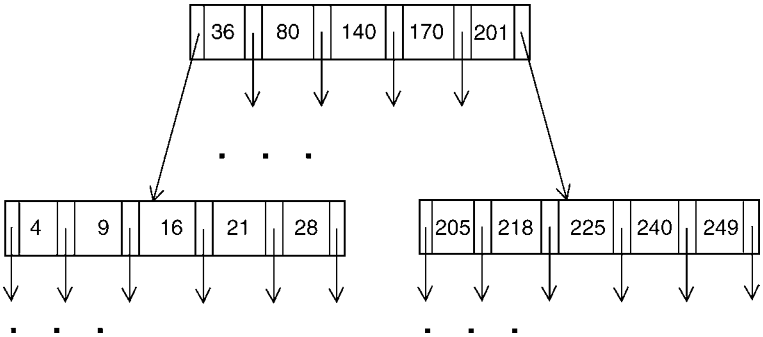
In recent years, for distributed data processing, variants of known algorithms have become popular in HDS. For example, researchers have introduced data aggregation techniques on sensor networks that can be considered as HDS techniques that rely on sketches. In this context, random-tree-based data aggregation algorithms in sensor networks can also be considered as basic HDS techniques. All of these different roots and aspects of hierarchical data summarization are visited in this article.

Foundations

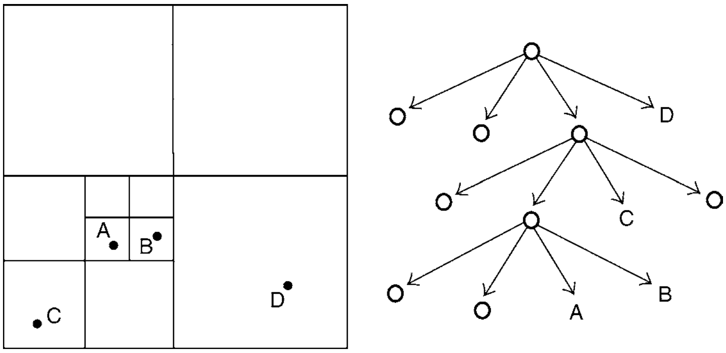
B+-trees are frequently used in databases. A B+-tree is given in Fig. 1 (only some parts of the tree are shown to simplify the presentation). B+-trees are hierarchical structures where internal nodes store keys and the leaf nodes contain the records attached to these keys. Due to their high fanout they are commonly shallow as well as balanced, i.e., in comparison to binary trees. They are used for efficient selection of a range of records from disks. The lowest level contains links between neighboring nodes to allow for sequential access to consecutive data items.

The B+-tree and related data structures can be considered as basic means of keeping hierarchical summary information. Given a fanout for a B+-tree, upper levels of the tree can be easily used for approximate HDS. One can refer to these levels to find the approximate number of items in a range. To make this HDS method more accurate, extra information should be maintained within the tree structure. For example, counts can be kept with each link in this hierarchy [10]. This requires extra space and maintenance costs as each count needs to be stored and updated with insertion and deletion operations. This can cause problems if many levels and nodes need to be maintained per update operation and if updates occur frequently for a given tree. If small errors in counters are tolerated then these overheads can be significantly reduced [3]. Counts form only one form of data summaries. Thus, the idea of counts is extended to other types of data summaries in [4,9].

For a set of queries and objects in space, such as range queries and a set of waterways in a country, spatial data structures can be used to efficiently store the data and answer queries on this data. For example, quadtrees are well-known, space-partitioning based structures. They are used with many different types of spatial data and thus many quadtree variants can be found in the literature. For example, a PR quadtree is given in Fig. 2. In this example, the space is recursively divided into four quadrants until a single data item is left in each quadrant. In Fig. 2, the space partitioning is shown (on the left) with its mapping tree structure (on the right). The positions of all the data items are also stored in the structure (not shown in the figure). Another related space partitioning method is the k-d tree. For k-d trees different dimensions of the



Hierarchical Data Summarization. Figure 1. An example B+-tree.



Hierarchical Data Summarization. Figure 2. An example quadtree with point objects.

underlying space is partitioned in turns at different levels of the tree. (Note that some of these methods are better named as tries, however, due to historical reasons, they are referred to as trees.)

If spatial objects are grouped together using bounding boxes and then a hierarchy of these bounding boxes are created, one can obtain an index called the R-tree. R-trees also have many variants. In comparison to quadtrees, they are commonly more balanced indexing schemes. However, many variants suffer from the fact that multiple bounding boxes, defining the tree nodes, can overlap in space. This nature of R-trees can reduce the pruning capability of this structure as a query may have to investigate multiple branches for the same space. Although disjoint-bounding-box based versions of R-trees exist, these variants could partition the data items into multiple boxes. There are many other spatial indices that are not presented in this article for the brevity of the presentation. The techniques mentioned are used to present HDS methods based on spatial indexing.

Similar to the case in B+-trees, spatial indices can also be viewed as HDS techniques. Moreover, data summaries can be explicitly maintained with these spatial data structures. This information can then be used for query processing, e.g., aggregate queries. Recently, spatial indexing is used in distributed settings for data summarization. For example, [7] introduces fractional cascading in sensor networks. In this approach, each sensor maintains detailed readings that it has obtained as well as data from its nearby neighbors. Information regarding other sensors are not kept as accurately. The space of sensors is partitioned using a distributed quadtree that is overlayed onto the sensor network. The partitioning is done in a similar manner to the PR quadtree example in Fig. 2. With increasing distance to the rest of the sensors in the network (i.e., to faraway quadrants) the amount of data collected from them drops with a function, e.g., a logarithmic function. This paradigm utilized the fact that data and queries in sensor networks are spatially and temporally correlated. Thus, this structure can be used to efficiently serve routing requests using locally summarized data as well as to answer queries. The distributed structure can be seen as a multi-rooted HDS technique as each sensor uses the same summarization scheme independently.

Similar to fractional cascading, [6] introduces the DIMENSIONS system that uses a pyramid-based space decomposition to aggregate and summarize data in a sensor network. Each quadrant finds a "local" leader node for building a distributed pyramid of nodes with their data. Other similar systems are DIM and DIFS systems [8,11] that use spatial indices on sensor networks for processing selection queries as well as resorting to summaries for user interest elimination. In [8] a k-d tree based structure is introduced while [11] introduces a multi-rooted quadtree type for avoiding bottlenecks from having a single root node.

Indices such as quadtrees and R-trees are also utilized in query optimization (e.g., [1]). As they represent summary information about the space they cover, they can easily be used in estimating the runtime costs of a query before it is executed.

In comparison to sophisticated indexing methods, a simple technique for summarizing data is the histogram. Histograms have long been employed in query optimization as they are compact and easy to maintain. With the emergence of data warehousing and On-line Analytical Processing (OLAP) technologies, they have also become crucial components from a new angle in data management. For data sets that explicitly contain hierarchies, e.g., years-months-weeks, histograms can easily be used.

From a processing cost estimation and query optimization point of view, Bruno et al. [5] introduced the concept of nested buckets with histograms. This can be seen as the first form of HDS using histograms. Later, Reiss et al. [14] built on this concept for distributed settings for bandwidth usage reduction. Reiss et al. present hierarchical histograms for aggregate query processing on identification data, i.e., RFIDs.

From an algorithmic point of view many methods that have long been used in approximating and summarizing data can be considered as hierarchical approaches to summarization. For example, wavelet transformations are well-established techniques in signal processing that can be used and considered as HDS methods. The following considers wavelets in the context of spatial data to give a simple example.

A three dimensional (3D) object in space can be approximated using a triangular mesh. One can use different sets of triangles, i.e., small or large, to give a more or less detailed approximation of the surface of the 3D object. Thus, an object can be represented in different resolutions using different meshes. If these meshes are related to each other geometrically, one can easily progressively update the details for this object on demand. Therefore, if M^I denotes a triangular mesh at resolution I, one can then represent an object as a series of meshes, M^0, M^1,...,M^J, where, M^0 is the base mesh and M^J is the final mesh. Figure 3a shows a triangular mesh with one triangle, M^0, (1,2,3), for a 2D object. The triangle is the coarse approximation for the surface of the given circle.

Consider a simple transformation. To obtain a higher resolution approximation of the given surface in the figure, the triangle (1,2,3) is divided into four sub-faces by introducing new vertices (4′,5′,6′), Fig. 3b. The new set of vertices are now displaced to make the mesh better fit to the surface of the circle. The new, finer resolution mesh M^1, is shown in Fig. 3c. This operation can be done recursively and can be represented with a simple transformation function. The coefficients that represent the difference between M^0 and M^1 are $d_4{}^0$, $d_5{}^0$, and $d_6{}^0$. In this simple wavelet transformation, for example, the wavelet coefficient $d_4{}^0$ is obtained by $v_4^1 - \frac{v_1^0+v_2^0}{2} = v_4^1 - v_{4'}^1$. Thus, the wavelet-based decomposition of a mesh M^J produces a base mesh M^0 and the sets, $\{W_0, W_1, \ldots, W_{J-1}\}$, of coefficients. From an HDS point of view, the recursive execution of the above mentioned method can be seen as a hierarchical summarization of a detailed polygonal representation of a complex data set. Various further HDS methods can be derived using the base concept of wavelets. For example, recently, [2] uses wavelets with R-trees to progressively retrieve and refine spatial data from a remote database.

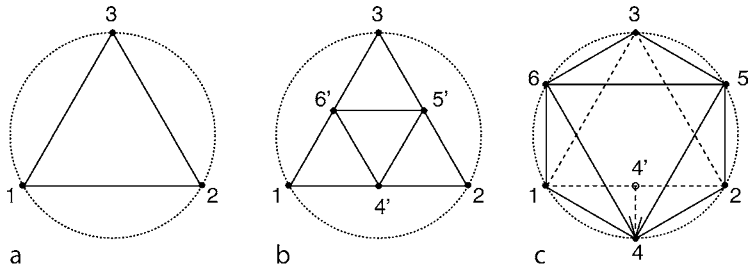
With the emergence of distributed systems such as sensor networks, aggregate query processing itself can now also be considered as a HDS technique. For example, to process an aggregate query in sensor networks, [12] uses a random-tree with in-network aggregation. Each node in this tree can compute an aggregate from its sub-trees, such as a minimum, and then pass this information to the higher-levels of the tree along the data collection path. The base-station, root, can then present a summary of the sensor data to the user.

Random-trees, however, are not robust. A single failure can cause significant problems (especially when a node that is close to the root fails). To address these problems, researchers have been working on multi-path data aggregation methods. In this scheme, multiple reports, due to the wireless coverage advantage in sensor networks, can be sent through different routes for increasing the robustness of the data collection method. However, this can cause deviations in certain aggregation operations, e.g., counts, as the same data is incorporated to the result multiple times. In [13], Nath et al. introduce a sketch theory-based HDS method to address this issue. They map aggregate functions, e.g., counts, to a set of order and duplicate insensitive synopsis generation and fusion functions.

Data clustering forms another area of research that, when applied using data aggregates and hierarchies, can be considered as a source of HDS techniques. For example, [15] introduces the STING system which uses a hierarchy of cells that contain aggregate information about the individual data items. Thus, for many query types, they can resort to these cells, rather than items, to answer queries efficiently. For queries that cannot be answered using summary data, individual data items can still be used as a backup strategy.

Key Applications

Key applications of HDS techniques are aggregate query processing and query optimization. If many queries are interested in retrieving summary data, e.g., aggregate queries, then maintaining a hierarchical summary would be efficient. For example, for data warehousing applications with hierarchical data, the benefits for maintaining a hierarchical summary could be significant. Data summaries have also long been used in query optimization. For example, an optimizer can use HDS techniques for selectivity estimation on attributes.



Hierarchical Data Summarization. Figure 3. A wavelet-based approximation.

Future Directions

There is a significant amount of activity in using data summaries in distributed settings and especially in sensor networks. In addition, with the emerging research directions in location-based services and VANETs, readers may expect to see the use of spatial HDS techniques more frequently. In distributed settings, bandwidth savings on the communication optimization front using HDS could be significant.

Url to Code

Demos for many of the spatial indices that are mentioned in this article can be found at http://www.cs.umd.edu/~hjs/quadtree/index.html.

Cross-references

▶ Aggregate Queries in P2P Systems
▶ B+-Tree
▶ Histogram
▶ Indexing
▶ Quadtrees (and Family)
▶ Query Processing and Optimization in Object Relational Databases
▶ Rtree
▶ Sensor Networks
▶ Sketch
▶ Spatial Network Databases
▶ Wavelets on Streams

Recommended Reading

1. Aboulnaga A. and Aref W.G. Window query processing in linear quadtrees. Distrib. Parallel Dat., 10(10):111–126, 2001.
2. Ali M.E., Zhang R., Tanin E., and Kulik L. A motion-aware approach to continuous retrieval of 3D objects. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 843–852.
3. Antoshenkov G. Query processing in DEC RDB: major issues and future challenges. IEEE Data Eng. Bull., 16(4):42–45, 1993.
4. Aoki P.M. Generalizing "search" in generalized search trees. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 380–389.
5. Bruno N., Chaudhuri S., and Gravano L. STHoles: a multidimensional workload-aware histogram. ACM SIGMOD Rec., 30 (2):211–222, 2001.
6. Ganesan D., Estrin D., and Heidemann J. DIMENSIONS: why do we need a new data handling architecture for sensor networks? In Proc. ACM Workshop on Hot Topics in Networks, 2002.
7. Gao J., Guibas L.J., Hershberger J., and Zhang L. Fractionally cascaded information in a sensor network. In Proc. 3rd Int. Symp. Inf. Proc. in Sensor Networks, 2004, pp. 311–319.
8. Greenstein B., Estrin D., Govindan R., Ratnasamy S., and Shenker S. DIFS: a distributed index for features in sensor networks. In Proc. IEEE Int. Workshop on Sensor Network Protocols and Applications, 2003, pp. 163–173.
9. Hellerstein J.M., Naughton J.F., and Pfeffer A. Generalized search trees for database systems. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 562–573.
10. Knuth D.E. Sorting and Searching, The Art of Computer Programming, vol. 3. Addison Wesley, Redwood City, CA, 1973.
11. Li X., Kim Y.J., Govindan R., and Hong W. Multi-dimensional range queries in sensor networks. In Proc. 1st Int. Conf. on Embedded Networked Sensor Systems, 2003, pp. 5–7.
12. Madden S.R., Franklin M.J., Hellerstein J.M., and Hong W. TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst., 30(1):122–173, 2005.
13. Nath S., Gibbons P.B., Seshan S., and Anderson Z.R. Synopsis diffusion for robust aggregation in sensor networks. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 250–262.
14. Reiss F., Garofalakis M., and Hellerstein J.M. Compact histograms for hierarchical identifiers. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 870–881.
15. Wang W., Yang J., and Muntz R. STING: a statistical information grid approach to spatial data mining. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997, pp. 186–195.

Hierarchical Entity-Relationship Model

▶ Extended Entity-Relationship Model

Hierarchical Faceted Metadata

▶ Faceted Search

Hierarchical Graph Layout

▶ Visualizing Hierarchical Data

Hierarchical Heavy Hitter Mining on Streams

FLIP R. KORN
AT&T Labs–Research, Florham Park, NJ, USA

Synonyms

HHH

Definition

Given a multiset S of N elements from a hierarchical domain D and a count thres hold $\phi \in (0,1)$, Hierarchical Heavy Hitters (HHH) summarize the distribution of S projected along the hierarchy of D as a set of prefixes $P \subseteq D$, and are defined inductively as the nodes in the hierarchy such that their "HHH count" exceeds φN, where the HHH count is the sum of all descendant nodes having no HHH ancestors. The approximate HHH problem over a data stream of elements e is defined with an additional error parameter $\varepsilon \in (0,\phi)$, where a set of prefixes $P \subseteq D$ and estimates of their associated frequencies, with accuracy bounds on the frequency of each $p \in P$, f_{min} and f_{max}, is output with $f_{min}(p) \leq f^*(p) \leq f_{max}(p)$ such that $f^*(p)$ is the true frequency of p in S (i.e., $f^*(p) = \sum_{e \preceq p} f(e)$) and $f_{max}(p) - f_{min}(p) \leq \varepsilon N$. Additionally, there is a coverage guarantee that, for all prefixes $q \notin P$, $\phi N > \sum f(e) : (e \preceq q) \wedge (e \not\preceq P)$, with $\preceq$ denoting prefix containment and $(e \preceq P)$ denoting $(\exists p \in P : e \preceq p)$. Algorithms for maintaining HHHs on data streams aim to minimize space usage as well as update time and possibly query reporting time, while maintaining these accuracy and coverage guarantees.

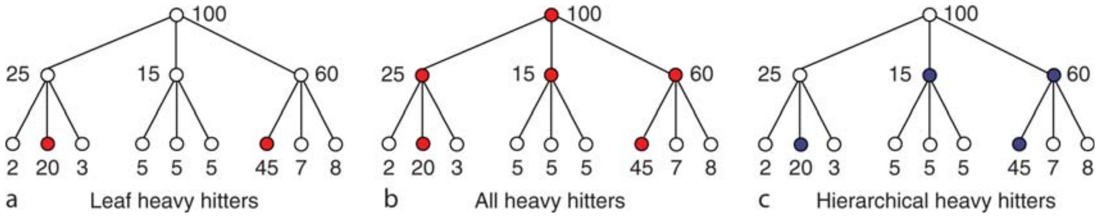
Historical Background

The Hierarchical Heavy Hitters problem was defined in [2] for one-dimensional hierarchical domains, and algorithms for finding online HHHs were given. The multi-dimensional version of the problem was defined and studied in [3,5]. HHHs extend the concept of Heavy Hitters (frequent items), which are all items with frequency at least ϕN, to hierarchical domains; the problem of finding Heavy Hitters on data streams was studied in [6,10,11].

Foundations

Figure 1 shows an example distribution of $N = 100$ items over a simple hierarchy in one dimension, with the counts for each internal node representing the total number of items at leaves of the corresponding subtree. Figure 1a shows that setting $\phi = 0.1$ yields two heavy hitters, that is, items with frequency above 10. However, this does not adequately cover the full distribution, and so one seeks a definition which also considers heavy hitters at points in the hierarchy other than the leaves. A natural approach is to apply the heavy hitters definition at each level of generalization: at the leaves, but also for each internal node. The effect of this definition is shown in Fig. 1b. But this fails to convey the complexity of the distribution: is a node marked as significant merely because it contains a child which is significant, or because the aggregation of its children makes it significant? This leads to the definition of HHHs given a fraction ϕ: find nodes in the hierarchy such that their HHH count exceeds ϕN, where the HHH count is the sum of all descendant nodes which have no HHH ancestors. This is best seen through an example, as shown in Fig. 1c. Observe that the node with total count 25 is not an HHH, since its HHH count is only 5 (less than the threshold of 10): the child node with count 20 is an HHH, and so does not contribute. But the node with total count 60 is an HHH, since its HHH count is 15. Thus the set of HHHs forms a superset of the heavy hitters consisting of only data stream elements, but a subset of the heavy hitters over all prefixes of all elements in the data stream.

One approach to computing HHHs extends the `LossyCounting` algorithm for frequency estimates on streams proposed in [10], which involves maintaining a set of samples from the input stream and associating a small amount of auxiliary information with each sampled item e used for determining the lower- and upper-bounds on the frequency of e. The space usage of `LossyCounting` is $O(\frac{1}{\epsilon}\log(\epsilon N))$, where N is the length of the stream (Better asymptotic space bounds for the heavy hitter problem were obtained in [12], but it was reported in [10] that `LossyCounting` often



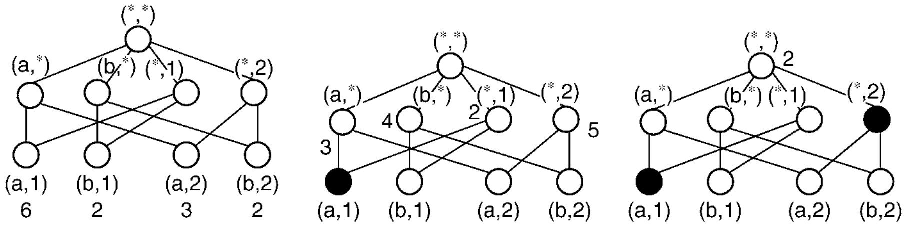
Hierarchical Heavy Hitter Mining on Streams. Figure 1. Illustration of HHH concept ($N = 100$, $\phi = 0.1$).

uses less space in practice). Online algorithms for finding HHHs similarly sample the input stream, but maintain a trie data structure T consisting of a set of tuples which correspond to samples from the input stream and their prefix relationships. There are two classes of these algorithms, which are described as follows.

Using the "Full Ancestry" strategy for finding HHHs, the data structure tracks information about a set of nodes that vary over time, but which always form a subtree of the full hierarchy. It enforces the property that if information about a node in the hierarchy is stored, then all of its ancestors are also stored. When a new node is inserted, information stored by its ancestors is used to give more accurate information about the possible frequency count of the node. This has the twin benefits of yielding more accurate answers and keeping fewer nodes in the data structure (since one can more quickly determine if a node cannot be frequent and so does not need to be stored). Thus, it can be shown that the algorithm requirements are no worse than that required for find heavy hitters at all prefix lengths of maximum length h, which is $O(\frac{h}{\epsilon}\log(\epsilon N))$ based on LossyCounting.

Observe that the previous strategy can be wasteful of space, since it retains all ancestors of the fringe nodes, even when these have low (even zero) count. In the "Partial Ancestry" strategy, such low count nodes can be removed from the data structure, thus potentially using less space. Thus, it is no longer the case that every prefix that is stored has all its ancestors stored as well. Frequency bounds are obtained from the closest existing ancestor of the newly inserted element. While this strategy does not come with space and time guarantees, its performance can be superior to the Full Ancestry strategy in practice [2].

The *Multi-Dimensional Hierarchical Heavy* Hitters problem is to find all items whose count exceeds a given fraction, ϕ, of the total count of all items, after discounting the appropriate descendants that are themselves HHHs. This still needs further refinement, since in this setting nodes can have multiple parents (i.e., the prefix containment forms a lattice rather than a tree), so it is not immediately clear how to compute the count of items at various nodes. In the one-dimensional HHH problem, the semantics of what to do with the count of a single element when it was rolled up was clear: simply add the count of the rolled up element to that of its (unique) parent. In the d-dimensional case, adding the count of the rolled up element to all its d parents runs the risk of overcounting. This can be rectified via careful bookkeeping, as illustrated in the example in Fig. 2. Consider a two-dimensional domain, where the first attribute can take on values a and b, and the second 1 and 2. Figure 2a shows a distribution where $(a, 1)$ has count 6, $(a, 2)$ has count 3, $(b, 1)$ has count 2, and $(b, 2)$ has count 2. Moving up the hierarchy on one or other of the dimensions yields internal nodes: $(a, *)$ covers both $(a, 1)$ and $(a, 2)$ and has count 9; $(*, 2)$ covers both $(a, 2)$ and $(b, 2)$, and has count 5. Setting $\phi = 0.35$ means that a count of 5 or higher suffices, thus there is only one Heavy Hitter over the leaves of the domain, as shown in Fig. 2b. In the multi-dimensional case, since one input item may contribute to multiple ancestors becoming HHHs, each node therefore counts the contributions of all its non-HHH descendants. Figure 2c shows the result on the example: the node $(*, 2)$ becomes an HHH, since it covers a count of 5. $(*, 1)$ is not an HHH, because the count of its non-HHH descendants is only 2. Note that the root node is not a HHH since, after subtracting off the contributions from $(a, 1)$ and $(*, 2)$, its remaining count is only 2. Online algorithms for finding multi-dimensional HHHs in data streams were first proposed and



a Frequency distribution with $N = 13$ b Heavy hitters with $\phi = 0.35$ c HHHs under overlap rule

Hierarchical Heavy Hitter Mining on Streams. Figure 2. Illustration of HHH in two dimensions.

analyzed in [3]; a complete analyis of space and time requirements can be found in [5].

Note that the bounds on the frequency estimates given by online HHH algorithms are for the total frequencies, not the (discounted) HHH counts. By appropriate rescaling of ε, one could find the discounted counts accurately; however, this comes at a high price for the required space, multiplying by a factor proportional to the largest possible number of HHH descendants. It is shown in [9] that such a factor is essentially unavoidable: any approximation guarantee on discounted counts of ϕ-HHHs requires $\Omega(1/\phi^{d+1})$ space. Hence, only guarantees on the total frequencies, not HHH counts, are provided by online algorithms.

Related follow-up work considered the problem of HHH detection without compensating for the count of HHH descendants [13]. The problem therefore simplifies to finding all nodes in the lattice whose count is above the threshold, that is, the heavy hitters over all prefixes. In [1], a sketch-based approach was considered for finding one-dimensional HHHs requiring $O(\frac{h}{\epsilon}\ln(1/\delta))$ space and $O(h\ln(1/\delta))$ time per update.

Key Applications

Hierarchical heavy hitters were implicitly studied in [7,8], to find patterns of traffic (offline) over a multi-dimensional hierarchy of source and destination ports and addresses in what the authors call "compressed traffic clusters." Online HHHs were used for real-time anomaly detection on IP data streams, again based on source and destination ports and addresses in [13]. A DDos detection system based on HHHs was proposed in [12].

Experimental Results

Experiments reported in [5], based on an implementation of HHHs as a User Defined Aggregate Function (UDAF) in the Gigascope data stream system (See [4] for a description of Gigascope UDAFs), show that the proposed online algorithms yielded outputs that were very similar to their offline counterparts and significantly better than that of heavy hitters on all prefixes. The algorithms also require an order of magnitude less space usage while giving competitive performance compared with the heavy hitters approach. The Partial Ancestry strategy is better when space usage is of importance whereas the Full Ancestry strategy is better when update time and output size is more crucial.

Cross-references

▶ Heavy Hitters
▶ Quantiles on Streams

Recommended Reading

1. Cheung-Mon-Chan P. and Clerot F. Finding hierarchical heavy hitters with the count min sketch. In Proc. Int. Workshop on Internet Rent, Simulation, Monitoring, Measurement, 2006.
2. Cormode G., Korn F., Muthukrishnan S., and Srivastava D. Finding hierarchical heavy hitters in data streams. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 464–475.
3. Cormode G., Korn F., Muthukrishnan S., and Srivastava D. Diamond in the rough: finding hierarchical heavy hitters in multi-dimensional data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 155–166.
4. Cormode G., Korn F., Muthukrishnan S., Johnson T., Spatscheck O., and Srivastava D. Holistic UDAFs at streaming speeds. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 35–46.
5. Cormode G., Korn F., Muthukrishnan S., and Srivastava D. Finding hierarchical heavy hitters in streaming data. ACM Trans. Knowl. Discov. Data, 1(4), 2008.
6. Demaine E., López-Ortiz A., and Munro J.I. Frequency estimation of internet packet streams with limited space. In Proc. European Symp. on Algorithms, 2002, pp. 348–360.
7. Estan C., Savage S., and Varghese G. Automatically inferring patterns of resource consumption in network traffic. In Proc. ACM Int. Conf. of the on Data Communication, 2003, pp. 137–148.
8. Estan C. and Magin G. Interactive traffic analysis and visualization with Wisconsin netpy. In Proc. Int. Conf. on Large Installation System Administration, 2005, pp. 177–184.
9. Hershberger J., Shrivastava N., Suri S., and Toth C. Space complexity of hierarchical heavy hitters in multi-dimensional data streams. In Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 2005, pp. 338–347.
10. Manku G.S. and Motwani R. Approximate frequency counts over data streams. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 346–357.
11. Misra J. and Gries D. Finding repeated elements. Sci. Comput. Program., 2:143–152, 1982.
12. Sekar V., Duffield N., Spatscheck O., van der Merwe J., and Zhang H. LADS: large-scale automated DDoS detection system. In Proc. USENIX Annual Technical Conf., General Track, 2006, pp. 171–184.
13. Zhang Y., Singh S., Sen S., Duffield N., and Lund C. Online identification of hieararchical heavy hitters: algorithms, evaluation and applications. In Proc. Internet Measurement Conference. Taormina, 2004, pp. 135–148.

Hierarchical Memory System

▶ Memory Hierarchy

Hierarchical Regular-Decomposition Structures

▶ Quadtrees (and Family)

Hierarchical Spatial Indexes

▶ Quadtrees (and Family)

Hierarchical Storage Management

▶ Storage of Large Scale Multidimensional Data

Hierarchical Visualization

▶ Visualizing Hierarchical Data

Hierarchies

▶ Specialization and Generalization

Hierarchy

Torben Bach Pedersen
Aalborg University, Aalborg, Denmark

Definition

A *hierarchy* is a structure specifying the containment relationships of a number of *values* (or *nodes* or *elements*). A hierarchy has a single *root* (or *top* or *ALL*) value. A value in a hierarchy may be linked to one or more *children* and a non-top value is linked to one or more *parents*. The links between values thus specify the containment structure. Hierarchies are essential for specifying *dimensions* in multidimensional *cubes*.

Key Points

The notion of a dimension is an essential and distinguishing concept for multidimensional cubes. Dimensions are used for two purposes: the *selection* of data and the *grouping* of data at a desired level of detail. As an example, a three-dimensional cube for capturing product sales may have a Product dimension, a Time dimension, and a Store dimension. The Product dimension captures information about the product sold, such as textual description, color, weight, etc., as well as groupings of products (product groups, product families, departments, etc.). The root value, representing "All products" then has store departments such as "Food" and "Electronics" as children, "Food" has product families such as "Dairy," "Meat," etc., as children and so on.

A hierarchy is typically organized into a number of *levels*, each of which represents a level of detail that is of interest to the analyses to be performed. In the example above, the levels could be ALL Products, Departments, Product Families, Product Groups and Products. The instances of the dimension, e.g., "Food" are typically called *dimension values*. Each such value then belongs to a particular level. The hierarchy is used intensively, e.g., when performing *On-Line Analytical Processing (OLAP)*. Here, data is explored by either moving up in the hierarchy to get a better overview (*rollup*) or moving down in the hierarchy to get more details (*drilldown*).

In some cases, it is advantageous for a dimension to have *multiple hierarchies* defined on it. For example, a Time dimension may have hierarchies for both *Fiscal Year* and *Calendar Year* defined on it. Multiple hierarchies share one or more common lowest level(s), e.g., Day and Month, and then group these into multiple levels higher up, e.g., Fiscal Quarter and Calendar Quarter to allow for easy reference to several ways of grouping. Most multidimensional models allow multiple hierarchies. A dimension hierarchy is defined in the metadata of the cube, or the metadata of the multidimensional database, if dimensions can be shared.

Most models require dimension hierarchies to form *balanced trees*. This means that the dimension hierarchy must have uniform height everywhere, e.g., all departments, even small ones, must be subdivided into Product families (and so on, all the way down to Products). If the hierarchy is not balanced like this, it is referred to as a *non-onto* or *unbalanced* hierarchy [2,3]. Additionally, direct links between dimension values can only go between immediate parent-child levels, and not jump two or more levels. For example, all cities are first grouped into states and then into countries, so cities cannot be grouped directly under countries (as is the case in Denmark which has no states). If such non-immediate links occur in the hierarchy, it is called a *non-covering* or *ragged* hierarchy [2,3], Finally, each non-top value has precisely one parent, e.g., a product must belong to exactly one product group. This may not always be desirable, e.g., it would be natural to put skimmed milk into both the "Diet" and "Dairy" product groups. If the hierarchies do not form balanced trees, this affects the so-called *summarizability* of the data, which means that special care must be taken to obtain correct aggregation results [1].

Cross-references

▶ Dimension
▶ Multidimensional Modeling
▶ On-Line Analytical Processing
▶ Summarizability

Recommended Reading

1. Lenz H. and Shoshani A. Summarizability in OLAP and statistical data bases. In Proc. 9th Int. Conf. on Scientific and Statistical Database Management, 1997, pp. 39–48.
2. Pedersen T.B., Jensen C.S., and Dyreson C.E. A foundation for capturing and querying complex multidimensional data. Inf. Syst., 26(5):383–423, 2001.
3. Thomsen E. OLAP Solutions: Building Multidimensional Information Systems. Wiley, New York, NY, 1997.

High Dimensional Indexing

Christian Böhm, Claudia Plant
University of Munich, Munich, Germany

Synonyms

Indexing for similarity search

Definition

The term High Dimensional Indexing [6,9] subsumes all techniques for indexing vector spaces addressing problems which are specific in the context of high dimensional data spaces, and all optimization techniques to improve index structures, and the algorithms for various variants of similarity search (nearest neighbor, reverse nearest neighbor queries, range queries, similarity joins etc.) for high dimensional spaces. The well-known Curse of Dimensionality leads to a worsening of the index selectivity with increasing dimensionality of the data space, an effect which already starts at dimensions of 10–15, also depending on the size of the database and the data distribution (clustering, attribute dependencies). During query processing, large parts of conventional hierarchical indexes (e.g., R-tree) need to be randomly accessed, which is by a factor of up to 20 more expensive than sequential reading operations. Therefore, specialized indexing techniques for high dimensional spaces include e.g., ideas to scale up sequential scans, hybrid approaches combining elements of hierarchical and scan-based indexes, dimensionality reduction and data compression.

Historical Background

One of the first indexing techniques which is nowadays wide spread in commercial databases is the B-tree. One dimensional data items, so-called keys, are stored in a hierarchical balanced search tree. Since the height of a B-tree is bounded above by $O(\log N)$, the index provides retrieval in logarithmic time complexity. Approaches to extend the B-tree to higher dimensions include the R-tree [12], which has been originally designed for two dimensional spatial objects (polygons). The R-tree consists of two types of nodes: directory and data pages. The directory pages contain rectangles (or hyper-rectangles for higher dimensional data) which are the minimum bounding rectangles (MBR) for the underlaying sub-trees, all the way down to the data pages at the leaves. The hyper-rectangles may overlap, and also the directory does not need to cover the whole data space. This implies that the search is not guaranteed to be restricted to one single path from the root to a leave as in the B-tree. However, since the R-tree is a balanced search tree and algorithms for tree construction and restructuring are designed to minimize overlap, search operations on low dimensional data can be performed in almost logarithmic time. Various variants of the R-tree, such as the R*-tree have been proposed.

Foundations

The central problem of similarity search in high dimensional spaces is the deterioration of the index selectivity with increasing dimensionality of the data space, an effect which is essentially inherent to any kind of index structure. Conventional hierarchical index structures achieve impressive performance gains over sequential scan on low to medium dimensional data by excluding large parts of the data contained in sub-trees which do not need to be visited. Special properties of high dimensional spaces, often subsumed by the term Curse of Dimensionality cause that conventional hierarchical indexes break down in performance on high dimensional data. With increasing dimensionality, more points are situated at the boundaries of the data space and the distances between points assimilate. During retrieval, large parts of the index have to be accessed. In the extreme case of very high dimensionality ($d \rightarrow \infty$) it is therefore obvious that no indexing technique is able to outperform the simplest processing technique for similarity queries, the sequential scan. This fact has been discussed in the context of cost models and of various indexing methods [3,7,19] and has also solicited a scientific discussion of the usefulness of similarity queries per se [6]. However, when, and to which extent the dimensionality curse occurs, depends on the size of the database and various other parameters such as the statistical distribution of the data, correlations between the single attributes (i.e., whether the complete data space is covered or the data reside in an arbitrarily complex subspace), and clustering (i.e., if there are clearly distinguishable groups of similar objects in the database). Many indexing methods can be successfully applied in moderately high dimensions, and many dedicated indexing methods are able to index data spaces of a dimensionality which is considerably higher than expected. In order to achieve the goal of efficiently indexing (moderately) high dimensional spaces, the well-known proposals to high-dimensional indexing all apply a combination of tricks which can be categorized into the following classes:

1. Dimensionality reduction
2. Data compression
3. Optimized i/o schedules (page size optimization and fast index scan)
4. Hierarchy flattening
5. Optimizing of the shape of page regions
6. Clustering

In the following, the most relevant approaches to High Dimensional Indexing are described in chronological order.

TV-Tree

High dimensional real data often exhibits a much lower intrinsic dimensionality. In this case, principal component analysis leads to a few highly loaded components almost totally covering the variance in the data. The TV-tree [16] exploits the fact that those top ranked components are highly selective dimensions for similarity search, whereas the remaining dimensions are of minor importance. Therefore, only the most selective components are used for pruning during query processing. Since irrelevant sub-trees should be excluded as early as possible, these components, called *active dimensions* are placed at the topmost levels of the index. A region of the TV-tree is described by a sphere in the active dimensions. The remaining dimensions may be active at lower levels of the index or not selective enough for query processing. The authors report a good speed-up in comparison with the R*-tree if the data can be effectively reduced by PCA, Fourier transform etc. On uniform or other real data not amenable to PCA, the X-tree outperforms the TV-tree.

The main contribution to High Dimensional Indexing is the implicit dimensionality reduction. Depending on the depth of the tree, only the first few dimensions are considered in the directory structure. Further, in these considered dimensions, the pages are approximated by bounding spheres which are more suitable for Euclidean queries.

SS-Tree

The SS-tree [20] also uses spheres instead of bounding rectangles as page regions. For efficiency, the spheres are not minimum bounding spheres. Rather, the centroid of the stored points is used as center for the sphere and the radius is chosen such that all objects are included. The region description consists of the centroid and the radius and thus allows efficient pruning. The SS-tree is suitable for all kinds of data distributions and outperforms the R*-tree by a factor of 2, which is mainly due to the fact that spherical page regions are, as mentioned, more suitable to support Euclidean queries. In addition, the algorithms for tree construction and maintenance are highly efficient and also very effective for low to medium dimensional data. On high dimensional data, the SS-tree encounters similar problems as the R-tree family. Compared to the minimum bounding rectangles (MBR) of R-trees, spherical page regions are even more difficult to split in an overlap-free way. This problem is tackled by the SR-tree [15] which uses a combination of a rectangle and a sphere as page region.

X-Tree

In contrast to the TV-tree and the SS-tree, the X-tree [5] is a high-dimensional index structure which is more closely related to the R-tree, and, in particular, to the R*-tree. It is the first technique which introduces the ideas of flattening the hierarchies of the directory and of enlarging the sizes of directory pages, but in contrast to the techniques described in the next section, this idea was not inspired by a cost analysis but simply from the observation that it can be difficult to split directory nodes (particularly of the higher directory levels) in an overlap-free way. The regions of the child nodes of such directory nodes are axis-parallel rectangles, which have, themselves, been created in a recursive process of splitting. Typically, only the first split in the split history of the child nodes is a good split decision for the parent node. Therefore, each directory node stores this split history of its children and, thus, tries to imitate the splits of its children. If this imitation would result in an unbalanced tree, the split is avoided and, instead, a so-called supernode is created, i.e., a node with an enlarged size and capacity.

Cost Model Based Optimization Techniques

In [3,7], a cost model for vector space index structures with particular emphasis on high dimensional spaces was proposed. The main idea is to estimate the average size of a page region and of a query and to form the Minkowski sum of these two geometric objetcs which has been shown to be a measure of the probability of a page access. To take effects in high-dimensional spaces into account, two concepts were additionally included into the cost model, *the boundary effect* and *the fractal dimension*. The boundary effect means that for high-dimensional spaces typically large parts of a query sphere and of the Minkowski sum are outside the covered data space. The fractal dimension covers the sparsity of the data space. Typically, a high dimensional space is not uniformly and independently in all dimensions covered by data objects. Rather, some of the attributes are dependent on each other, and, therefore, only subspaces of lower dimensionality (not necessarily linear subspaces) are populated. The intrinsic dimensionality of the data can be formalized by the fractal dimension.

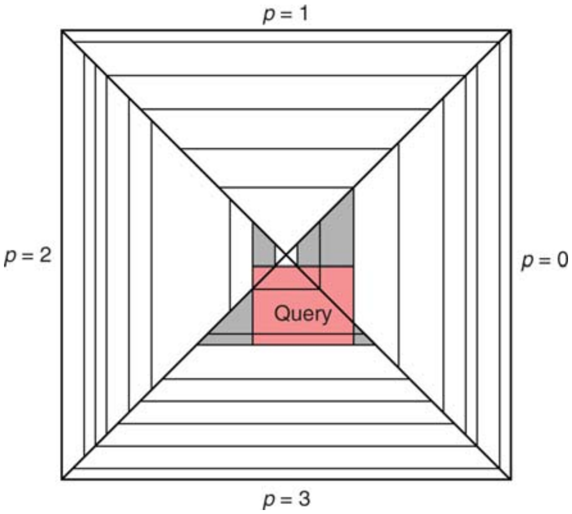
It is important to optimize various parameters of an index using a cost model in order to make it competitive. For instance, in [8], it was proposed to optimize the page size of data pages of the index according to the cost model. To do this automatically is particularly important because data sets with a large intrinsic dimensionality cause scanning of large parts of the index. If small page sizes (such as 4 KB) are used in this case, the random accesses cause a large I/O load, and an unoptimized index is much slower than sequential data processing. In contrast, if the intrinsic dimensionality is small, then large page sizes lead to unnecessarily large reading operations. Carefully optimized page sizes outperform sequential scanning and non-optimized indexes for most of the data sets and tie with the competitors in the worst case. The dynamic page size optimization allows the automatic adaptation of the page size even if the data distribution of the data set changes over time. Moreover, it is possible to use different page sizes for different parts of the index structure, if the index stores groups of data with different characteristics.

Another example of the successful application of a cost model is tree striping [4], where the data objects are vertically decomposed into sub-vectors which are stored in separate search trees. The dimensionality of the sub-vectors (and, inversely, the number of index structures to store them) is an optimization task which is important, because the unnecessary decomposition in too many, small sub-vectors causes a large overhead in the final merging of the results, whereas no decomposition or an insufficient decomposition the query processing inside a single index is too expensive due to the curse of dimensionality. Again, a cost model can decide an optimal dimensionality.

Pyramid Technique

The Pyramid Technique [1] is a *one-dimensional embedding technique*, that means, it transforms the high-dimensional objects into a single-dimensional space which can be efficiently indexed using B-trees, for instance. In contrast to most previous embedding techniques which are based on space-filling curves, the Pyramid Technique does not rely on a *recursive* schema of space partitioning. In contrast, the data space is partitioned into $2 \cdot d$ hyper-pyramids which share the origin of the data space (which can be chosen as the center point of the data set) as top point and have each an individual $(d - 1)$-dimensional basis area (cf. the four pyramids in Fig. 1 $p = 0..3$). The pyramids are systematically numbered which forms the first part of the embedding key (a natural number p). The second part is the distance (with respect to the maximum metric) from the origin (a positive real number r). The embedding key can be formed as an ordered pair $k = (p, r)$, or, equivalently, if the maximum of all r-values (r_{max}) is known, one single embedding key

$k' = r_{max} \cdot p + r$ can be formed. In both cases, a d-dimensional range query can be translated into a set of search intervals on the search keys. The number of intervals is at most $2 \cdot d$ because the query object can at most have one intersection with each of the pyramids. Since nearest neighbor queries can be transformed into range queries (which requires a set of at most two one-dimensional ranking processes per pyramid) it is also possible to evaluate nearest neighbor queries. The Pyramid Technique is in general not limited to a particular metric, but the schema of space partitioning makes it particularly suited for queries using the maximum metric. The Pyramid Technique has inspired a number of other techniques such as the Onion Technique [10] or Concentric Hyperspaces [11] and many others which focus on different metrics including the Euclidean metric.



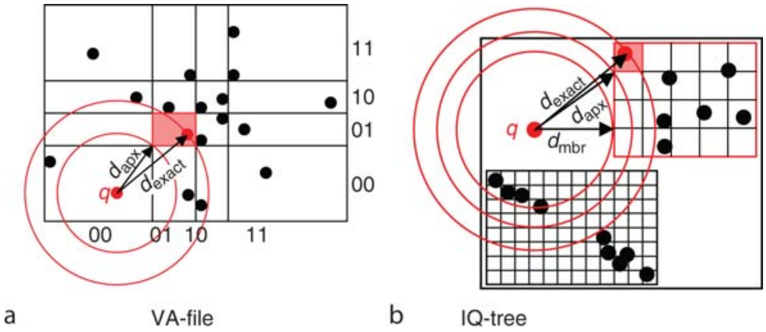
High Dimensional Indexing. Figure 1. Pyramid technique.

VA-File

As an alternative to tree based indexing techniques, the VA-file (Vector Approximation File [6]) has been designed to speed up the linear scan. The basic idea is to store compact approximations of the high dimensional feature vectors in the so-called *approximation file* which is small enough to enable a fast linear scan during query processing. The approximations are derived by dividing each dimension of the data space into equally populated intervals. Thus, the data space is divided into hyper-rectangular cells. Each cell is allocated by a bit string composed of a fixed number of bits per dimension. As approximation for all contained points, this bit string is stored in the approximation file. The principle of vector quantization is visualized in Fig. 2a. In this example, two bits are assigned to each dimension. While the approximation file is scanned, upper and lower bounds on the distances from the query to the approximations d_{apx} are derived and refined. As a result, only very few approximations have to be further checked for answer candidates, which requires random accesses. Extensions to the VA-file e.g., include the parallel VA-file [18] originally designed for parallel nearest neighbor search in large image collections and the kernel-VA-file [13] for complex, kernel supported distance metrics. Due to global quantization, the VA-file is especially suitable for uniformly distributed data. For clustered data, hierarchical techniques show superior performance.

IQ-Tree

With the IQ-tree [2], the idea of quantizing the data using a grid has been integrated into a hierarchical indexing method. In the IQ-tree, every data page has two versions, one which contains the data points in a compressed way, and one which contains the exact



High Dimensional Indexing. Figure 2. Schematic view of vector quantization.

information. In contrast to the VA-file, each page is quantized independently, and this independent quantization gives the structure its name. The quantization grid is not based on quantiles but is a regular grid partition of the rectangular page region. In the IQ-tree, it was shown that quantiles are not necessary because the page regions already serve a sufficient adaptation to the actual data distribution, and storing individual quantiles for each page would result in a prohibitive storage overhead. In contrast to the VA-file, the resolution of the grid is not determined experimentally but dynamically optimized using a cost model. In the IQ-tree, the actual directory is non-hierarchic and contains only one level. Taken together, the IQ-tree consists of three levels, the directory level, the compressed data level and the uncompressed data level. The vector quantization principle is illustrated in Fig. 2b. When considering a query object q and an arbitrary database object, there are two lower bounding approximations of the exact distance d_{exact}, the distance to the minimum bounding rectangle d_{mbr} which can be determined from the directory and the distance to the grid cell (d_{apx}) which can be derived from the compressed data level.

Apart from the idea of independent quantization, the IQ tree contains the idea of the Fast Index Scan (FIS). After scanning of the directory, it can be decided for every page, whether it is certainly needed, certainly excluded, or has some probability to be needed for a given nearest neighbor query. The probability can again be estimated using a cost model. From this probability, a query processing algorithm can derive optimal schedules of pages, i.e., pages which have neighboring disk addresses, can be called in together in a single I/O operation if they have both high probabilities. Depending on data and query characteristics, the pages of the compressed data level can either be accessed by random accesses or in a more sequential style. Another related approach, also designed to allow a flexible adaptation of the height of the directory to the current data distribution is the A-tree [17].

iDistance

iDistance [21] combines a one-dimensional embedding technique with clustering. The embedding is obtained by expressing the similarity ratios in high dimensional space in terms of distances to reference points which can be stored in a single dimensional index. More precisely, the data space is first split into partitions. As a second step, a reference point is selected for each partition. For all data objects the distances to their reference points are stored in a B^+-tree. The performance of the index strongly depends on an appropriate partitioning and on the strategy how to select the reference points. In the original paper, the authors proposed two variants of partitioning: The straightforward approach of equal data space partitioning is especially suitable for uniformly distributed data. For clustered data the partitions can be determined by an arbitrary clustering algorithm, a simple sampling based method is proposed in the paper. Often it is favorable to select the centroid of a partition as reference point, however selecting an edge point may help to reduce overlap.

The most important strategies to cope with the problems of high dimensional spaces can be subsumed by clustering and mapping to one dimensional space. During query processing, the maximum distance between the query point and the points contained in each partition is used for pruning. The single dimensional space of distances can be very efficiently searched supported by the B^+-tree. However, in high dimensional data, the distances to reference points are often not selective and there are no clusters in the full dimensional space such that large parts of the index need to be accessed. But many real world data sets exhibit more selective subspaces. Therefore in [14] a subspace clustering step to identify local correlations in the data is applied before indexing.

Key Applications

High dimensional indexing is important for similarity search systems in various application areas such as multimedia, CAD, systems biology, medical image analysis, time sequence analysis and many others. Complex objects are typically transformed into vectors of a high-dimensional space (feature vectors), and the similarity search thereby translates into a range or nearest neighbor query on the feature vectors. High-dimensional feature vectors are also required for more advanced data analysis tasks such as cluster analysis or classification.

Cross-references

- ▶ Curse of Dimensionality
- ▶ Dimensionality Reduction
- ▶ Indexing Metric Spaces
- ▶ Nearest Neighbor Query

Recommended Reading

1. Berchtold S., Böhm C., and Kriegel H.-P. The pyramid-technique: towards breaking the curse of dimensionality. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 142–153.
2. Berchtold S., Böhm C., Jagadish H.V., Kriegel H.-P., and Sander J. Independent quantization: an index compression technique for high-dimensional data spaces. In Proc. 16th Int. Conf. on Data Engineering, 2000, pp. 577–588.
3. Berchtold S., Böhm C., Keim D.A., and Kriegel H.-P. A cost model for nearest neighbor search in high-dimensional data space. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997, pp. 78–86.
4. Berchtold S., Böhm C., Keim D.A., Kriegel H.-P., and Xu X. Optimal multidimensional query processing using tree striping. In Proc. 2nd Int. Conf. Data Warehousing and Knowledge Discovery, 2000, pp. 244–257.
5. Berchtold S., Keim D.A., and Kriegel H.-P. The x-tree : an index structure for high-dimensional data. In Proc. 22nd Int. Conf. on Very Large Data Bases, 1996, pp. 28–39.
6. Beyer K.S., Goldstein J., Ramakrishnan R., and Shaft U. When is "nearest neighbor" meaningful? In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 217–235.
7. Böhm C. A cost model for query processing in high dimensional data spaces. ACM Trans. Database Syst., 25(2):129–178, 2000.
8. Böhm C. and Kriegel H.-P. Dynamically optimizing high-dimensional index structures. In Advances in Database Technology, Proc. 7th Int Conf on Extending Database Technology, 2000, pp. 36–50.
9. Böhm C., Berchtold S., and Keim D.A. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Comput. Surv., 33(3):322–373, 2001.
10. Chang Y.-C., Bergman L.D., Castelli V., Li C.-S., Lo M.-L., and Smith J.R. The onion technique: indexing for linear optimization queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 391–402.
11. Ferhatosmanoglu H., Agrawal D., and Abbadi A.E. Concentric hyperspaces and disk allocation for fast parallel range searching. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 608–615.
12. Guttman A. R-trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
13. Heisterkamp D.R. and Peng J. Kernel vector approximation files for relevance feedback retrieval in large image databases. Multimed. Tools Appl., 26(2):175–189, 2005.
14. Jin H., Ooi B.C., Shen H.T., Yu C., and Zhou A. An adaptive and efficient dimensionality reduction algorithm for high-dimensional indexing. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 87–98.
15. Katayama N. and Satoh S. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 369–380.
16. Lin K.-I., Jagadish H.V., and Faloutsos C. The tv-tree: an index structure for high-dimensional data. VLDB J., 3(4):517–542, 1994.
17. Sakurai Y., Yoshikawa M., Uemura S., and Kojima H. The A-tree: an index structure for high-dimensional spaces using relative approximation. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 516–526.
18. Weber R., Böhm K., and Schek H.-J. Interactive-time similarity search for large image collections using parallel VA-files. In Proc. 4th European Conf. Research and Advanced Tech. for Digital Libraries. Springer, 2000, pp. 83–92.
19. Weber R., Schek H.-J., and Blott S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 194–205.
20. White D.A. and Jain R. Similarity indexing with the ss-tree. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 516–523.
21. Yu C., Ooi B.C., Tan K.-L., and Jagadish H.V. Indexing the distance: an efficient method to KNN processing. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 421–430.

High-Dimensional Clustering

► Document Clustering

Higher-Order Entity-Relationship Model

► Extended Entity-Relationship Model

Histogram

Qing Zhang
CSIRO ICT Centre, Herston, QLD, Australia

Definition

Given a relation R and an attribute X of R, the domain D of X is the set of all possible values of X, and a finite set $V(\subseteq D)$ denotes the distinct values of X in an instance r of R. Let V be ordered, that is: $V = \{v_i : 1 \leq i \leq n\}$, where $v_i < v_j$ if $i < j$. The instance r of R restricted to X is denoted by T, and can be represented as: $T = \{(v_1, f_1), \cdots (v_n, f_n)\}$. In T, each v_i is distinct and is called a *value* of T; and f_i is the occurrence of v_i in T and is called the *frequency* of v_i, T is called the *data distribution*. A *histogram* on data distribution T is constructed by the following two steps.

1. Partitioning the values of T into $\beta(\geq 1)$ disjoint intervals (called buckets) – $\{B_i : 1 \leq i \leq \beta\}$, such that each value in B_i is smaller than that in B_j if $i < j$.

2. Approximately representing the frequencies and values in each bucket.

Key Points

Histogram, as a summarization of the data distribution, has been successfully used by the query optimizer in RDBMS for the past few decades. Due to its simplicity and straightforwardness, it is also the most popular data reduction technique for approximately processing range aggregates. However, to find a good histogram, which occupies a reasonable small storage space yet can provide accurate approximations, is not an easy work. The key issues in constructing a histogram are: (i) how to partition the original data into buckets, and (ii) how to approximate the original data in each bucket.

Since a histogram only occupies limited storage space in the system catalog, partition methods study how to effectively partition the original data distribution, under a fixed bucket number, to construct a histogram with good performance. Different partition strategies can greatly influence the histogram's performance. Four basic partition methods, based on different partition goals, are: *Equi-width, Equi-sum, Maxdiff, V-optimal* [3]. Empirical results indicate that in most applications, Maxdiff and V-optimal outperform Equi-width and Equi-sum. V-optimal usually leads to more accurate approximation than Maxdiff does. However, the time complexity on constructing a V-optimal histogram seriously impedes its further application.

Techniques on approximately representing the values and frequencies in each histogram bucket usually adopt the *uniform spread* and *average* frequency assumptions. That is distinct values in a same bucket are assumed to evenly span the bucket and every value has a same frequency, the average frequency of the bucket. More accurate approximation techniques on the values and frequencies have also been proposed in recent years, such as *curve-fitting histogram* [2] and *4-level tree index* [1].

There exist other types of histograms in the literature, such as spatio-temporal histogram, multi-dimensional histogram, etc.

Cross-references

▶ Approximate Query Processing
▶ Data Reduction
▶ Hierarchical Data Summarization
▶ Quantiles on Streams

Recommended Reading

1. Buccafurri F., Rosaci D., Doutieri L., and Sacca D. Improving range query estimation on histograms. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp 628–638.
2. Konig A.C. and Weikum G. Combining histograms and parametric curve fitting for feedback-driven query result-size estimation. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999.
3. Poosala V., Ioannidis Y.E., Haas P.J., and Shekita E.J. Improved histograms for selectivity estimation of range predicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1996, pp. 294–305.

Histograms on Streams

MARTIN J. STRAUSS
University of Michigan, Ann Arbor, MI, USA

Synonyms

Piecewise-constant approximations

Definition

A B-bucket histogram of length N is a partition of the set $[0,N)$ of N integers into intervals $[b_0,b_1) \cup [b_1,b_2) \cup ... \cup [b_{B-1},b_B)$, where $b_0 = 0$ and $b_B = N$, together with a collection of B heights h_j, for $0 \le j < B$, one for each bucket. On point query i, the histogram answer is h_j, where j is the index of the interval (or "bucket") containing i; that is, the unique j with $b_j \le i < b_{j+1}$. In vector notation, χ_S is the vector that is 1 on the set S and zero elsewhere and the answer vector of a histogram is $\vec{H} = \sum_{0\le j<B} h_j \chi_{[b_j,b_{j+1})}$.

A histogram, $\vec{H}$, is often used to approximate some other function, $\vec{A}$, on $[0,N)$. In building a B-bucket histogram, it is desirable to choose $B - 1$ boundaries b_j and B heights h_j that tend to minimize some distance, e.g., the sum square error $\left\|\vec{A} - \vec{H}\right\|^2 = \sum_i |\vec{A}[i] - \vec{H}[i]|^2$.

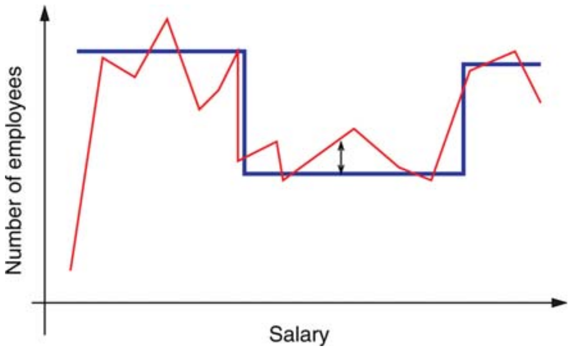
The function $\vec{A}$ may be presented in a stream in one of several ways. In many of these cases, it is infeasible to produce a best histogram; instead, algorithms trade off the representational accuracy of the histogram, the time to process stream items, and the space used.

Historical Background

Fix a universe $[0,N) = \{0,1,2,...,N - 1\}$ of N item types, e.g., salaries. The goal is to summarize a vector $\vec{A} = \langle \vec{A}[0], \vec{A}[1],..., \vec{A}[N-1] \rangle$ over $[0,N)$, in which $\vec{A}[i]$ represents, e.g., the number of employees earning salary *i*. To summarize $\vec{A}$ by a histogram, partition the interval of indices (salaries) into *B* subintervals and represent $\vec{A}$ on interval *j* as a single number, h_j, according to some specification. For example, h_j might be the total number $\vec{A}[b_j,b_{j+1})$ of items in the range (the total number of employees whose salary *i* falls in the range $[b_j,b_{j+1})$). Alternatively, h_j might be the normalized expression $\frac{\vec{A}[b_j,b_{j+1})}{b_{j+1}-b_j}$ (the *average height* of $\vec{A}$), which conveys the same information but is more natural below to analyze, because the average height is the optimal value of h_j from the perspective of sum-square-error; it will be the focus of this article. See Fig. 1.

Classical histograms include equi-width histograms, in which the *j*th boundary b_j, for $0 \le j \le B$, is fixed to be (the integer nearest to) jN/B. In many circumstances, average height of all buckets can be found efficiently and straightforwardly when the bucket boundaries are fixed in advance.

Letting the bucket boundaries vary, however, can lead to more accurate histograms. This generalization can be handled [5] by dynamic programming when time and space polynomial in *N* is allowed; i.e., for non-streaming data. For each $j \le N$ and each $k \le B$, find the best *k*-bucket histogram on $[0,j)$; this clearly suffices to find the best *B*-bucket histogram on $[0,N)$. Observe that the best *k*-bucket histogram on $[0,j)$ consists of a $(k-1)$-bucket histogram on $[0,i)$ for some $i < j$ followed by a single bucket $[i,j)$ with height $\frac{\vec{A}[i,j)}{j-i}$; this naturally leads to an algorithm that uses time $O(N^2B)$, assuming one can find $\vec{A}[i,j)$ in constant time. The latter task may be accomplished by computing, in time and space $O(N)$ during preprocessing, the *prefix array* $\vec{P}$ for $\vec{A}$ defined by $\vec{P}[0] = 0$ and $\vec{P}[j] = \vec{P}[j-1] + \vec{A}[j-1]$ for $j > 0$. Thus $\vec{P}[j] = \vec{A}[0,j) = \vec{A}[0] + \vec{A}[1] + \ldots + \vec{A}[j-1]$ so that $\vec{A}[i,j) = \vec{P}[j] - \vec{P}[i]$ can be computed in constant time.



Histograms on Streams. Figure 1. A histogram of three buckets for a frequency vector of data (jagged polygon). Often the goal is to minimize the sum of square between the data vector and histogram by choosing the bucket boundaries and heights for the histogram.

Many variations of histograms are possible, including equi-depth histograms, also known as quantile summaries, in which bucket boundaries are fixed so that the total number of items in each bucket is equal. For example, a 4-bucket equi-depth histogram has boundaries at the 25th, 50th, and 75th percentiles. Neither quantiles nor other variations will be discussed further in this article.

A fuller history of histogram research is available [4].

Foundations

There are several ways in which a data vector $\vec{A}$ can be presented in a stream; this article discusses two. In the *ordered aggregate* (or *time-series*) presentation, the algorithm receives $\vec{A}[0],\vec{A}[1],\vec{A}[2],\ldots$, in order. In the *turnstile* (or *dynamic*) presentation, $\vec{A}$ is implicit and initially zero. The algorithm receives update commands of the form (i, u), whose semantics is "add u to $\vec{A}[i]$." In general, the update u may be negative or positive. More efficient algorithms are possible if it is known that the updates are positive (or merely that each $\vec{A}[i]$ is never negative); that will not be discussed further here.

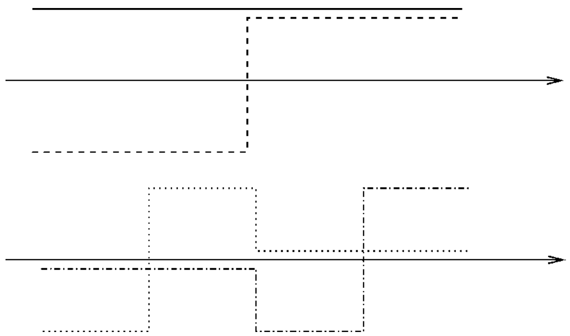
A class of algorithms for building near-optimal histograms relies on the theory of *Haar wavelets*. Assume *N* is a power of 2 (which can be achieved, without loss of generality, by padding $\vec{A}$ implicitly with trailing zeros). A Haar wavelet is either the constant vector $\phi = N^{-1/2}\chi_{[0,N)}$ or a vector of the form

$$\psi_{j,k}(i) = \begin{cases} -\sqrt{2^j/N}, & kN2^{-(j+1)} \le \\ & i < (k+1)N2^{-(j+1)}; \\ +\sqrt{2^j/N}, & (k+1)N2^{-(j+1)} \le \\ & i < (k+2)N2^{-(j+1)}, \end{cases}$$

where $0 \le j < \log_2(N)$ and $0 \le k < 2^j$ are integers. That is, for some scale *j*, divide $[0,N)$ into 2^j subintervals of length $N2^{-j}$. Then $\psi_{j,k}$ takes the value $-\sqrt{2^j/N}$ on the left half of the *k*th interval and takes the value $+\sqrt{2^j/N}$ on the right half. See Fig. 2.

One also writes ψ_ℓ to index all the $\psi_{j,k}$ and ϕ by a single index. A partial Haar wavelet representation is $\Sigma_{\ell\in\Lambda}\langle\vec{A}\psi_\ell\rangle\psi_\ell$, where $\langle.,.\rangle$ denotes the dot product and Λ is a subset of $[0, N)$.

In many streaming situations, it is not possible to find the best *B*-bucket histogram $\vec{H}_{\text{opt}}$ for $\vec{A}$. Instead,



Histograms on Streams. Figure 2. The four coarsest Haar wavelets ϕ,$\psi_{0,0}$,$\psi_{1,0}$, and $\psi_{0,1}$ with normalization modified for visibility.

the goal will be to find a near-best histogram, $\|\vec{H}-\vec{A}\| \le (1+\varepsilon)\|\vec{H}_{\text{opt}}-\vec{A}\|$, where ε is a user-supplied accuracy parameter.

A class of algorithms for building near-optimal histograms efficiently from streaming data proceeds as follows.

1. Find the largest B_1 Haar wavelet terms, where $B_1 \le (B log(N)/\varepsilon)^{O(1)}$. Let $\vec{R}$ be the resulting vector. That is, let Λ_1 be a set of size B_1 consisting of the ℓ's that maximize $|\langle\vec{A},\psi_\ell\rangle|$ and let $\vec{R}$ be $\sum_{\ell\in\Lambda_1} c_\ell\psi_\ell$, where c_ℓ is $\langle\vec{A},\psi_\ell\rangle$ or a close enough approximation. How to find $\vec{R}$ depends on the way that $\vec{A}$ is presented. The representation $\vec{R}$ must be constructible with little space and little per-item time, so that its construction fits within the constraints of a streaming algorithm. Furthermore, a description of $\vec{R}$ has size $|\vec{R}|$ that is small enough so that polynomial time in $|\vec{R}|$ is also within the constraints of a streaming algorithm for the overall problem.
2. Let $\vec{H}_{rob} = \sum_{\ell\in\Lambda_2} c_\ell\psi_\ell$, where Λ_2 is a subset of Λ_1, chosen so that either $\|\vec{R}-\vec{H}_{\text{rob}}\|$ cannot be significantly improved by enlarging Λ_2 within Λ_1 by $\Theta(B\ log(N))$ wavelet terms (if there is such a set Λ_2) or $\Lambda_2=\Lambda_1$ (otherwise). This is done greedily. Let Λ_2 be initially empty. Let Λ^* be the set of $\Theta(B\ log(N))$ indices $\ell \in \Lambda_2 \setminus \Lambda_1$ with maximal $|c_\ell|$; if $\|\Sigma_{\ell\in\Lambda_2\cup\Lambda^*} c_\ell\psi_\ell - \vec{R}\|$ is significantly less than $\|\sum_{\ell\in\Lambda_2} c_\ell\psi_\ell - \vec{R}\|$, then put $\Lambda_2 \leftarrow \Lambda_2 \cup \Lambda^*$ and repeat; otherwise, halt and return the current Λ_2.
3. Let $\vec{H}$ be the best B-bucket histogram representation for $\vec{H}_{\text{rob}}$. This can be done efficiently using dynamic programming, as above, noting that the boundaries of $\vec{H}$ will be among the boundaries of $\vec{H}_{\text{rob}}$. Output $\vec{H}$ as a near-best B-bucket histogram for $\vec{A}$.

In the case of time-series data, the wavelet coefficients can be found efficiently using a structure based on a binary tree in which each leaf corresponds to an input item $\vec{A}[i]$ and each internal node receives inputs denoted x and y from its children, outputs $y-x$ as a wavelet coefficient (up to normalization) and passes $x+y$ as an input to the node's parent. Note that, while reading $\vec{A}[i]$, the algorithm only needs to instantiate nodes on the path from the ith leaf to the root, for a total of $O(log(N))$ space. Each of $O(N)$ nodes performs $O(1)$ work, for a total of $O(N)$ time. The top B_1 wavelet coefficients can be collected from the tree's output stream in time $O(N)$. It follows that the above algorithm, on time-series data, takes total time $c_1N + (B\log(N)/\epsilon)^{c_2}$ and space $(B\log(N)/\epsilon)^{c_3}$, for constants c_1,c_2, and c_3. Using suitable buffering (without significantly increasing the space requirements), the per-item time is $O(1)$.

In the case of dynamic data, updates to $\vec{A}$ positions are transformed into updates to wavelet coefficients of $\vec{A}$ and the large wavelet coefficients are tracked using a sketch-based structure. An update (i,u), with semantics "add u to $\vec{A}[i]$," is regarded for analysis as the vector $u\delta_i$ that takes the value u at position i and zero elsewhere. Using the tree structure of wavelets, it can be shown that, for all i, $\delta_i = \sum_{\ell\in\Lambda_i}\langle\delta_i,\psi_\ell\rangle\psi_\ell$, where Λ_i has size $O(log(N))$ and can be constructed quickly from i. Given update (u,i) to $\vec{A}$, the algorithm quickly forms updates $(u\langle\delta_i,\psi_\ell\rangle,\ell)$ to the vector $\hat{\vec{A}}$ of wavelet coefficients for each $\ell \in \Lambda_i$. This algorithm uses a synopsis data structure, called an "L^2 count sketch," parametrized by N and η, that efficiently supports the following operations on an unordered set S of size N:

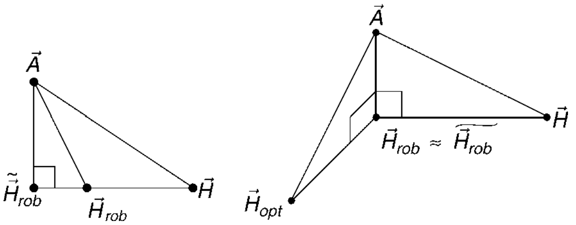
1. UPDATES of the form "add x to a count $\mathbf{C}[i]$ for item $i \in S$."
2. Unparametrized "FIND" queries. The answer is a list containing all j where $C[j]^2 \ge \eta\Sigma_i C[i]^2$.
3. "COUNT" queries about item i, for which the answer is $\tilde{C}[i]$ such that $\left|\tilde{C}[i]-C[i]\right|^2 \le \eta\|C\|^2$.

There are randomized implementations of L^2 count sketches [1,2] that need space $(log(N)/\eta)^{O(1)}$ and time $(log(N)/\eta)^{O(1)}$ for either query. Such an L^2 count sketch with appropriate η at least $(Blog(N)/\varepsilon)^{-O(1)}$ suffices to track the large wavelet terms needed for our algorithm. With appropriate implementations, the algorithm will take space and per-item time $(Blog(N)/\varepsilon)^{O(1)}$, including time to build the histogram completely after each update. (The algorithm depends on a randomized data structure. There is also a cost factor of $O(log\,(1/\delta))$ to achieve success probability $1 - \delta$.)

Correctness of the algorithm depends on which of the two conditions is in force when constructing $\vec{H}_{\mathrm{rob}}$. If $\vec{H}_{\mathrm{rob}} = \vec{R}$, then one can show that the wavelet coefficient magnitudes of $\vec{A}$ decay essentially exponentially. It follows that $\vec{H}_{\mathrm{rob}}$ is a very good approximation to $\vec{A}$, say, $\|\vec{H}_{rob} - A\| \leq \frac{\epsilon}{2}\|\vec{H}_{opt} - A\|$, where $\vec{H}_{\mathrm{opt}}$ is the optimal B-bucket histogram for $\vec{A}$. In that case, since $\vec{H}$ is the optimal B-bucket histogram for $\vec{H}_{\mathrm{rob}}$,

$$\begin{aligned}\|\vec{H} - \vec{A}\| &\leq \|\vec{H} - \vec{H}_{rob}\| + \|\vec{H}_{rob} - \vec{A}\|\\ &\leq \|\vec{H}_{opt} - \vec{H}_{rob}\| + \|\vec{H}_{rob} - \vec{A}\|\\ &\leq \|\vec{H}_{opt} - \vec{A}\| + 2\|\vec{H}_{rob} - \vec{A}\|\\ &\leq (1+\epsilon)\|\vec{H}_{opt} - \vec{A}\|.\end{aligned}$$

Otherwise, if $\vec{H}_{\mathrm{rob}} \neq \vec{R}$, it follows that $\vec{H}_{\mathrm{rob}}$ can not be much improved as an approximation to $\vec{R}$ whence it cannot be much improved as an approximation to $\vec{A}$ by refinements of $B - 1$ more bucket boundaries and bucket heights that are optimal or close enough to optimal so that the effect of approximation of bucket heights can be ignored. In particular, the best linear combination of $\vec{H}_{\mathrm{rob}}$ and any B-bucket histogram is a refinement of $\vec{H}_{\mathrm{rob}}$ by at most $B - 1$ boundaries and so is not much of an improvement over $\vec{H}_{\mathrm{rob}}$. Thus, as in Fig. 3, there are two (near-) right angles at $\vec{H}_{\mathrm{rob}}$ and so $\|\vec{H} - \vec{A}\|$ is (nearly) no worse than $\|\vec{H}_{\mathrm{opt}} - \vec{A}\|$, i.e., $\|\vec{H} - \vec{A}\| \leq (1+\in)\|\vec{H}_{\mathrm{opt}} - \vec{A}\|$, as desired.



Histograms on Streams. Figure 3. Illustration of histograms in correctness discussion. On the left, $\tilde{\vec{H}}_{\mathrm{rob}}$ is the best histogram for $\vec{A}$ in the linear span of $\vec{H}_{\mathrm{rob}}$ and $\vec{H}$; there is a right angle as indicated. On the right, since $\vec{H}_{\mathrm{rob}} \approx \tilde{\vec{H}}_{\mathrm{rob}}$, there are two near right angles at $\vec{H}_{\mathrm{rob}}$. Since $\vec{H}$ is no farther from $\vec{H}_{\mathrm{rob}}$ than $\vec{H}_{\mathrm{opt}}$ is, it follows that $\vec{H}$ is almost as close to $\vec{A}$ as $\vec{H}_{\mathrm{opt}}$ is. (Because the angles at $\vec{H}_{\mathrm{rob}}$ are not quite right angles, it does not follow that $\|\vec{A} - \vec{H}\| \leq \|\vec{A} - \vec{H}_{\mathrm{opt}}\|$, only that $\|\vec{A} - \vec{H}\| \leq (1+\in)\|\vec{A} - \vec{H}_{\mathrm{opt}}\|$.)

Key Applications

Histograms on streaming data can be used for selectivity estimation as part of query execution optimization. They can also be used at the user level, e.g., for visualization of data. Many of the techniques developed for stream processing also support the processing of distributed databases.

Cross-references

▶ Heavy Hitters
▶ Sketch
▶ Wavelets on Streams

Recommended Reading

1. Cormode G. and Muthukrishnan S. An improved data stream summary: the count-min sketch and its applications. In Proc. 6th Latin American Symp. Theoretical Informatics, 2004, pp. 29–38.
2. Gilbert A., Guha S., Indyk P., Kotidis Y., Muthukrishnan S., and Strauss M. Fast, small-space algorithms for approximate histogram maintenance. In Proc. 34th Annual ACM Symp. on Theory of Computing, 2002, pp. 389–398.
3. Guha S., Koudas N., and Shim K. Approximation and streaming algorithms for histogram construction problems. ACM Trans. Database Sys., 31(1):396–438, March 2006.
4. Ioannidis Y. The history of histograms (abridged). In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 19–30.
5. Jagadish H., Koudas N., Muthukrishnan S., Poosala V., Sevcik K., and Suel T. Optimal histograms with quality guarantees. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 275–286.
6. Muthukrishnan S. and Strauss M. Approximate histogram and wavelet summaries of streaming data. In Data-Stream Management – Processing High-Speed Data Streams. Springer, New York (Data-Centric Systems and Applications Series), 2009.

Historical Algebras

▶ Temporal Algebras

Historical Data Model

▶ Temporal Data Models

Historical Data Models

▶ Temporal Logical Models

Historical Database

▶ Temporal Database

Historical Query Languages

▶ Abstract Versus Concrete Temporal Query Languages
▶ Temporal Query Languages

Historical Spatio-Temporal Access Methods

▶ Indexing Historical Spatio-Temporal Data

History

▶ Provenance
▶ Provenance in Scientific Databases

History in Temporal Databases

Christian S. Jensen[1], Richard Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Time sequence; Time series; Temporal value; Temporal evolution

Definition

The *history* of an "object" of the real world or of a database is the temporal representation of that object. Depending on the specific object, one can have *attribute histories, entity histories, relationship histories, schema histories, transaction histories*, etc.

Key Points

"History" is a general concept, intended in the sense of "train of events connected with a person or thing."

In the realm of temporal databases, the concept of history is intended to include multiple time dimensions as well as multiple data models. Thus one can have, e.g., valid-time histories, transaction-time histories, and bitemporal histories. Also multi-dimensional histories can be defined from mono-dimensional ones (e.g., a bitemporal history can be seen as the transaction-time history of a valid-time history).

The term "history," defined formally or informally, has been used in many temporal database papers, also to explain other terms. For instance, salary history, object history, and transaction history are all expressions used in this respect.

Although "history" usually has to do with past events, its use for the future—as introduced by, e.g., prophecies, science fiction, and scientific forecasts—does not seem to present comprehension difficulties (The adjective "historical" seems more problematic for some). Talking about future history requires the same extension of meaning as required by talking about future data.

The synonym "temporal value" appears less general than "history," since it applies when "history" specializes into attribute history (value history), and it suggests a single value rather than a succession of values across time. The concept of a history is a slightly more general than the concept of a time sequence. Therefore the definition of "history" does not prevent defining "time sequence."

Since "history" in itself implies the idea of time, the use of "history" does not require further qualifications as is needed in the case of "sequence" or "series." In particular, "history" lends itself well to be used as modifier, even though "time sequence" is an alternative consolidated term.

Cross-references

▶ Bitemporal Relation
▶ Event
▶ Temporal Database
▶ Temporal Data Models
▶ Temporal Element
▶ Transaction Time
▶ User-Defined Time
▶ Valid Time

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer, 1998, pp. 367–405.

Homogeneous Distributed Database Systems

▶ Distributed Database Systems

Homogeneously Distributed Data

▶ Horizontally Partitioned Data

Homomorphic Encryption

NINGHUI LI
Purdue University, West Lafayette, IN, USA

Definition

Homomorphic encryption is a form of public-key encryption where one can perform a specific algebraic operation on the plaintext by performing a (possibly different) algebraic operation on the ciphertext. For example, knowing only $C_1 = \varepsilon(x)$ and $C_2 = \varepsilon(y)$, but not x, y, one may be able to get $\varepsilon(x + y)$ by calculating $C_1 \times C_2$. A homomorphic encryption scheme allows a third party to operate on encrypted values without knowing the plaintext.

Key Points

Depending on the intended application, the homomorphic property can be seen as a positive or negative attribute of the cryptosystem. Homomorphic encryption schemes are malleable by design. On the other hand, homomorphic encryption algorithms have found applications in a number of cryptographic problems, e.g., for secure voting and private information retrieval. Some encryption schemes that have the homomorphic property are given below.

RSA

If the public key is n, e, then the encryption of a message x is given by $\varepsilon(x) = x^e \bmod n$. The homomorphic property is property is as follows.

$$\begin{aligned}\varepsilon(x_1) \cdot \varepsilon(x_2) &= x_1^e x_2^e \bmod n \\ &= (x_1 x_2)^e \bmod n \\ &= \varepsilon(x_1 \cdot x_2 \bmod n)\end{aligned}$$

El Gamal

If the public key is p, g, $h = g^a$, and a is the secret key, then the encryption of a message x is $\varepsilon(x) = (g^r, x \cdot h^r)$. The homomorphic property is property is as follows.

$$\begin{aligned}\varepsilon(x_1) \cdot \varepsilon(x_2) &= (g^{r_1}, x_1 \cdot h^{r_1})(g^{r_2}, x_2 \cdot h^{r_2}) \\ &= (g^{r_1+r_2}, (x_1 \cdot x_2) h^{r_1+r_2}) \\ &= \varepsilon(x_1 \cdot x_2 \bmod p)\end{aligned}$$

Goldwasser-Micali

If the public key is the modulus n, and x is a quadratic non-residue modulo n, then the encryption of a bit b is $\varepsilon(b) = r^2 x^b \bmod n$. The homomorphic property is property is as follows.

$$\begin{aligned}\varepsilon(b_1) \cdot \varepsilon(b_2) &= r_1^2 x^{b_1} r_2^2 x^{b_2} = (r_1 r_2)^2 x^{b_1+b_2} \\ &= \varepsilon(b_1 \oplus b_2)\end{aligned}$$

where $\bigoplus$ denotes addition modulo 2 (i.e., exclusive-or).

Paillier

If the public key is the modulus n and the base g, then the encryption of a message x is $\varepsilon(x) = g^x r^n \bmod n^2$. The homomorphic property is property is as follows.

$$\begin{aligned}\varepsilon(x_1) \cdot \varepsilon(x_2) &= (g^{x_1} r_1^n)(g^{x_2} r_2^n) = g^{x_1+x_2}(r_1 r_2)^n \\ &= \varepsilon(x_1 + x_2 \bmod n)\end{aligned}$$

Cross-references

▶ Asymmetric Encryption
▶ Data Encryption

Recommended Reading

1. Paillier P. Public-key cryptosystems based on composite degree residuosity classes. In Advances in Cryptology: EUROCRYPT '99. LNCS vol.1592, Springer, 1999, pp. 223–238.

Horizontal Fragmentation

▶ Distributed Database Design

Horizontally Partitioned Data

Murat Kantarcioglu
University of Texas at Dallas, Dallas, TX, USA

Synonyms

Homogeneously distributed data

Definition

Data is said to be horizontally partitioned when several organizations own the same set of attributes for different sets of entities. More formally, horizontal partitioning of data can be defined as follows: given a dataset $DB = (E, I)$ (e.g., hospital discharge data for state of Texas) where E is the set of entities about whom the information is collected (e.g., the set of patients) and I is the set of attributes that is collected about entities (e.g., set of features collected about patients), DB is said to be horizontally partitioned among k sites where each site owns $DB_i = (E_i, I_i)$, $1 \leq i \leq k$ if $E = E_1 \cup E_2 ... \cup E_k$, $E_i \cap E_j = \emptyset$, $1 \leq i \neq j \leq \mathbf{k}$ and $I = I_1 = I_2 ... = I_n$. In relational terms, with horizontal partitioning, the relation to be mined is the union of the relations at the sites.

Historical Background

Cheap data storage and abundant network capacity have revolutionized data collection and data dissemination. At the same time, advances in data mining have made it possible to learn previously unknown and interesting knowledge from the collected data [4]. These developments have caused serious concerns related to privacy implications of data mining. Especially, privacy issues related to sharing data for data mining raised significant concerns [2]. To address these privacy issues for the horizontally partitioned data case, two different solution ideas were initially proposed. In [1], perturbation-based techniques were proposed to build decision trees in a privacy preserving way. In perturbation based techniques data noise is added to the data before it is revealed to the data miner to protect individual values. In [10], the secure multi-party computation based approach was proposed to build decision trees. The definition of privacy followed in this line of research is conceptually simple: anything learned during the data mining process must be derivable given one's own data and the final result. Following these initial works, many different privacy-preserving data mining algorithms have been proposed including association rule mining [10], naive bayes classification [6], k-NN Classification [7], support vector machine classification [11], and k-means and EM clustering [5,9].

Foundations

Most of the work in privacy preserving data mining on horizontally partitioned data deals with the privacy issues that arise in data collection and data sharing for data mining. Due to privacy concerns, individuals and organizations may not be willing to share their data. For example, individuals may not be willing to tell their incomes or companies may not be willing to reveal statistics related to their core businesses. Fortunately, in many cases, the information aggregated over many individuals may not be considered a privacy problem, and it can be assumed that the data mining result itself is not a privacy violation. Therefore the main goal is to learn the data mining results by disclosing as little as possible about the data sources. Two different solution ideas have been suggested for addressing this privacy challenge. One idea is to apply data perturbation techniques to privacy-sensitive data before data mining (e.g., [1]). The other solution idea is to use cryptographic techniques to learn only the final data mining result without revealing anything under some cryptographic assumptions (e.g., [10]).

Key Applications

Since the horizontally partitioned data model assumes that different sites collect the same set of information about different entities, the solution techniques developed could be applied for any application that satisfies this assumption. For example, it could be applied for building fraud detection models using the data that is collected by different credit card companies related to the credit card transactions of different individuals.

The health-care industry provides one of the key application areas for the methods developed for horizontally partitioned data. For example, suppose the Centers for Disease Control (CDC), a public agency, would like to mine health records to try to find ways to reduce the proliferation of antibiotic resistant bacteria. Insurance companies have data on patient diseases and prescriptions. CDC may try to mine association rules of the form $X \Rightarrow Y$ such that the $Pr(X\&Y)$ and $Pr(Y|X)$ are above thresholds. Mining this data for association rules would allow the discovery of rules such as *Augmentin&Summer* ⇒ *Infection&Fall*, i.e., people taking Augmentin in the summer seem to have recurring infections.

The problem is that insurance companies will be concerned about sharing this data. Not only must the privacy of patient records be maintained, but insurers will be unwilling to release rules pertaining only to them. Imagine a rule indicating a high rate of complications with a particular medical procedure. If this rule doesn't hold globally, the insurer would like to know this; they can then try to pinpoint the problem with their policies and improve patient care. If the fact that the insurer's data supports this rule is revealed (say, under a Freedom of Information Act request to the CDC), the insurer could be exposed to significant public relations or liability problems. This potential risk could exceed their own perception of the benefit of participating in the CDC study. In many such cases, privacy preserving distributed data mining techniques could be used to learn the final data mining result without revealing anything other than the result itself.

Future Directions

Although the provably secure distributed data mining protocols that reveal nothing but the resulting data mining model. This work still leaves a privacy question open: Do the resulting data mining models inherently violate privacy? This question is important because the full impact of privacy-preserving data mining will only be realized when there are guarantees that the resulting models do not violate privacy as well. The long list of possible privacy violations due to data mining results given in [8] indicates that care must be taken in revealing data mining results. Recently, in [3], the authors gave a new decision tree learning algorithm which guarantees that the data mining result does not violate the k-anonymity of the individuals represented in the training data. Although current work in this area has made valuable contributions, more research is needed to understand the privacy implications of data mining results.

Cross-references

► Privacy-Preserving Data Mining
► Vertically Partitioned Data

Recommended Reading

1. Agrawal R. and Srikant R. Privacy-preserving data mining. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 439–450.
2. Clifton C. and Marks D. Security and privacy implications of data mining. In Proc. Workshop on Data Mining and Knowledge Discovery, 1996, pp. 15–19.
3. Friedman A., Wolff R., and Schuster A. Providing k-anonymity in data mining. VLDB J., 17(4):789–804, 2008.
4. Han J. and Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco, California, April 2000.
5. Jagannathan G. and Wright R.N. Privacy-preserving distributed *k*-means clustering over arbitrarily partitioned data. In Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2005, pp. 593–599.
6. Kantarcioglu M. and Vaidya J. Privacy preserving naive bayes classifier for horizontally partitioned data. In Proc. Workshop on Privacy Preserving Data Mining, 2003.
7. Kantarcıoğlu M. and Clifton C. Privately computing a distributed *k*-nn classifier. In Principles of Data Mining and Knowledge Discovery, 8th European Conf, 2004, pp. 279–290.
8. Kantarcıoğlu M., Jin J., and Clifton C. When do data mining results violate privacy? In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 599–604.
9. Lin X., Clifton C., and Zhu M. Privacy preserving clustering with distributed EM mixture modeling. Knowl. Inform. Syst., 8(1):68–81, July 2005.
10. Lindell Y. and Pinkas B. Privacy preserving data mining. In Advances in cryptology – CRYPTO 2000. Springer, Berlin, 2000, pp. 36–54.
11. Yu H., Jiang X., and Vaidya J. Privacy-preserving svm using nonlinear kernels on horizontally partitioned data. In Proc. 2006 ACM Symp. on Applied Computing, 2006, pp. 603–610.

Horn Clause Query

► Conjunctive Query

Hot Items

► Frequent Items on Streams

Hotspots

► Spatial Data Mining

HSM

► Storage of Large Scale Multidimensional Data

HTML Fragment

▶ Web Views

Human Centered 38 H Human-Computer Interaction Computing

▶ Human-Computer Interaction

Human Factors

▶ Human-Computer Interaction

Human Interface

▶ Human-Computer Interaction

Human-centered Computing: Application to Multimedia

Nicu Sebe[1,2], Alejandro Jaimes[3]
[1]University of Amsterdam, The Netherlands
[2]University of Trento, Trento, Italy
[3]Telefonica R&D, Madrid, Spain

Synonyms

HCC; Human-centered multimedia; HCM

Definition

Human-centered computing (HCC) is an emerging, interdisciplinary academic field broadly concerned with computing and computational artifacts as they relate to the human condition.

HCC consists of a set of methodologies for designing, implementing, and deploying computing in any field that uses computing in any form. The definition of HCC is purposely broad, and this reflects the view that computing plays an active and crucial role in a huge number of human activities.

One important area of HCC is concerned with the design, implementation, and deployment of computing systems or devices *in everyday environments*. In such cases, it is desirable for HCC approaches to:

- Integrate input from different types of sensors and communicate through a combination of media as output.
- Act according to relevant social and cultural contexts.
- Be useful to diverse individuals.

In general, HCC methodologies guide the design of effective computer systems taking into account personal, social, and cultural aspects. The field of study, therefore, includes topics such as information design, human-information interaction, human-computer interaction, and human-human interaction. Additionally, it includes topics that are not directly related to interaction (algorithms, data collection and analysis, system design, etc.). Of particular interest in HCC is a critical analysis of the relationships between computing technology, society, and culture – so works in fields such as new media art also form part of HCC.

The distinction between computing and multimedia computing is blurry but in spite of this, the main human-centered activities in multimedia can be identified as follows [6]: media production, annotation, organization, archival, retrieval, sharing, analysis, and communication. The above activities can in turn be clustered into three large activity areas: production, analysis, and interaction.

Historical Background

Discussions about HCC have taken place over several years [5,7,16,20]. HCC draws from concepts and research in the humanities and social sciences (cognitive psychology, sociology, anthropology, philosophy, information sciences, and several others), as well as from established and emerging technical and interdisciplinary fields (HCI, computer-supported cooperative work (CSCW), user-centered design (UCD) [11], ubiquitous computing, artificial intelligence, new media art, etc.).

Although it is difficult to quantify the contributions of various fields to HCC, it is generally acknowledged that researchers and practitioners in HCI, UCD, and CSCW have made many important contributions to HCC.

User-centered design can be characterized as a multistage problem-solving process that requires designers not only to analyze and foresee how users are likely to use an interface but also to test the validity of their assumptions with regard to user behavior in the real world [15]. CSCW, on the other hand, combines the understanding of the way people work in groups with the enabling technologies of computer networking and associated hardware, software, services, and techniques [4].

While many of the topics covered by UCD and CSCW are relevant to HCC, the scope of Human-centered computing is much broader, covering more than the traditional areas of usability engineering, HCI, and human factors, which are primarily concerned with user interfaces or user interaction [8]. Compared to HCI, HCC has a twofold perspective [20]:

- HCC is "conceived as a theme that is important for all computer-related research, not as a field that overlaps or is a subdiscipline of computer science."
- HCC acknowledges that "computing connotes both concrete technologies (that facilitate various tasks) and a major social and economic force."

HCC is not to be confused with *human-based computation* [3], a technique also known as interactive evolutionary computation first developed in the early 1990s which puts the human at the center, but in a different way. In traditional computation, a human provides a formalized problem description to a computer and receives a solution to interpret. In human computation, the roles are reversed: the computer asks a person or a large number of people to solve a problem, then collects, interprets, and integrates their solutions. In other words, the computer "asks" the human to do the work it cannot do. This is done as a way to overcome technical difficulties: instead of trying to get computers to solve problems that are too complex, use human beings. In human computation the human helps the computer solve difficult problems. In HCC, the goal is to consider human abilities in creating, developing, and deploying systems or devices that use computing.

Although HCC and human-based computation approach computing from two different perspectives, they both try to maximize the synergy between human abilities and computing resources. Work in human computation can therefore be of significant importance to HCC. On one hand, data collected through human computation systems can be valuable for developing machine-learning models. On the other hand, it can help in better understanding of human behavior and abilities, again of direct use in HCC algorithm development and system design.

Foundations

A human-centered approach to multimedia starts with a deep understanding of human abilities and behavior in particular socio-cultural contexts. When human interaction is part of the computing process, HCM methodologies must consider how human beings understand and interpret multimedia signals at the perceptual, cognitive, and affective levels, and how they interact naturally – embedding the cultural and social contexts as well as personal factors such as emotion, attitude, and attention. The human-centered systems that humans directly interact with should be multimodal, processing inputs and outputs in a naturally rich communication channel; be proactive – understanding cultural and social contexts and responding accordingly; and be easily accessible outside the desktop to a wide range of users [6]. Because of this, HCM must consider work in fields such as neuroscience, psychology, cognitive science, and others. One of the key challenges, therefore, is incorporating what is known in those fields within computational frameworks that integrate different media.

Research on machine learning integrated with domain knowledge, data mining, sensor fusion, and multimodal interaction will play a key role [18]. Further research into quantifying human-related knowledge is necessary, which means developing new theories and mathematical models of multimedia integration at multiple levels.

Although many disciplines contribute to HCM (and implicitly HCC), the field is at an early stage so a research agenda will involve a non-exhaustive list of goals including the following:

- New human-centered methodologies for the design of models and algorithms and the development of diverse human-centered systems
- Focused research on the integration of multiple sensors, media, and human sciences that have people as the central point
- New interdisciplinary academic and industrial programs, initiatives, and meeting opportunities

- Discussions on the impact of computing technology that include the social, economic, and cultural contexts in which such technology is or might be deployed
- Research data that reflects human-centered approaches – for example, rich data collected from real multisensorial and culturally diverse social situations
- Common computing resources – for example, software tools and platforms
- Evaluation metrics for theories, design processes, implementations, and systems from a human-centered perspective
- Methodologies for privacy protection and the consideration of ethical and cultural issues

Clearly, an HCM and HCC research agenda should include a broad understanding and a multidisciplinary approach. This is of particular importance when considering computing in the context of developing regions [2].

Key Applications

The span of HCM application areas is very broad and as computing becomes more ubiquitous, practically every aspect of interaction with objects, and the environment, as well as human-human interaction (e.g., remote collaboration, etc.) will make use of HCM techniques. Several key application area descriptions follow:

Ambient Intelligence and Personal Spaces

Computing is being integrated with everyday objects in a variety of domains. This implies that the model of "user" in which a person sits in front of a "computer" is no longer the only model. Therefore, the actions or events to be recognized by the "interface" are not necessarily explicit commands and must respond to personal context in homes, offices, and other spaces, making ambient intelligence [1] a very important application area of HCM.

Public information kiosk applications [10] present many technical challenges in the design and implementation of natural multimodal interaction: kiosks are often intended to be used by a wide audience, thus there may be few assumptions about the types of users of the system. Thus, they present interesting opportunities for HCM approaches and must integrate context and socio-cultural considerations.

HCM also plays an important role in personal spaces (e.g., cars, office spaces, etc.). On one hand HCM techniques emphasize personalization, and on the other hand they emphasize natural interaction. Computing in vehicles, for example, can be used to monitor the driver [9] and to adjust conditions according to personal preferences. In addition, since the driver must focus on the driving task, traditional interfaces (e.g., GUIs) are not suitable, creating the need for natural, non-disruptive interaction techniques focused on maximizing human abilities while answering human needs in specific situations.

Ubiquitous Computing

One of the major challenges in ubiquitous computing is that while devices such as PDAs and mobile phones have become smaller and more powerful, there has been little progress in developing effective interfaces to access the increased computational and media resources they posses. Computing in ubiquitous devices constitute a very important area of opportunity for research in HCM: they create challenges in interaction for creation and access to content, as well as opportunities for contextual data collection and personalization.

Data Analysis and Data Interaction

HCM approaches can be used to process, analyze, and visualize various types of human-related data. The range of personal information collected in digital format is very wide, and can be used to support many human activities. Social networks, for example, have gained significant popularity and are based largely on connections of data about different individuals. HCM approaches to analyze and interact with data and information are crucial in such types of applications. Web search and recommender systems are two additional areas in which HCM techniques play a significant role because they can strongly impact not only the interaction aspects, but how the information is organized, indexed, and searched. Systems for analyzing, processing, and visualizing customer data for user modeling, profiling, and market segmentation can clearly have positive impacts in fields that use computing (e.g., marketing, advertising) to determine how and where products or services are provided.

Virtual Environments

Virtual and augmented reality have been very active research areas at the crossroads of computer graphics, computer vision, and human-computer interaction. One of the major difficulties of VR systems is the interaction component, and many researchers are currently exploring the use of interaction analysis techniques to enhance the user experience. One reason this is very attractive in VR environments is that it helps disambiguate communication between users and machines (in some cases virtual characters, the virtual environment, or even other users represented by virtual characters [14]). HCM techniques can be applied, however, to many aspects of VR environments: from the visual representation of characters and the environment, to the "physical" behavior of objects and how events and actions are perceived by human "users" of such systems.

Art

Perhaps one of the most exciting application areas of HCM is art, from interactive installation and interactive architecture to performance and audience participation. Multiple modalities (video, audio, pressure sensors) can be used to allow audience participation and influence a performance [19]. Mouth gestures can be interpreted [13] by a wearable camera pointing at the wearer's mouth to generate MIDI sounds (so a musician can play other instruments while generating sounds by moving his mouth). HCM technologies can also be used in museums to augment exhibitions [17] and in many other contexts.

Persons with Disabilities

Persons with disabilities can benefit greatly from HCM techniques [12] because the goal is specifically to enhance or complement person's abilities. Solutions for smart wheel-chair systems, for example, integrate different types of sensors and consider different kinds of interaction such as using only eye blinks and eyebrow movements for navigation.

Other Applications

Other applications include education, medicine, remote collaboration, entertainment, robotics, surveillance, or biometrics. HCM also plays an important role in safety-critical applications (e.g., medicine, military, etc.) and in situations in which a lot of information from multiple sources has to be viewed in short periods of time (e.g., crisis management).

Cross-references

- ▶ Human-Computer Interaction
- ▶ Multimedia Data
- ▶ Multimodal Interfaces
- ▶ Personalized Web Search
- ▶ Social Networks

Recommended Reading

1. Arts E. Ambient intelligence: a multimedia perspective. IEEE Multimedia, 11(1):12–19, 2004.
2. Brewer E. et al. The case for technology in developing regions. Computer, 25–38, June 2005.
3. Gentry C., Ramzan Z., and Stubblebine S. Secure distributed human computation. In Proc. 6th ACM Conf. on Electronic Commerce, 2005, pp. 155–164.
4. Grudin J. Computer-supported cooperative work: Its history and participation. Computer, 27:19–26, 1994.
5. http://www.human-centered-computing.org/
6. Jaimes A. Human-centered multimedia: culture, deployment, and access. IEEE Multimedia, 13(1):12–19, 2006.
7. Jaimes A., Gatica-Perez D., Sebe N., and Huang T.S. Human-centered computing: toward a human revolution. Computer, 30–34, May 2007.
8. Jaimes A., Sebe N., and Gatica-Perez D. Human-centered computing: a multimedia perspective. In Proc. 14th ACM Int. Conf. on Multimedia, 2006, pp. 855–864.
9. Ji Q. and Yang X. Real-time eye, gaze, and face pose tracking for monitoring driver vigilance. Real-Time Imaging, 8:357–377, 2002.
10. Johnston M. and Bangalore S. Multimodal Applications from Mobile to Kiosk. In Proc. W3C Workshop on Multimodal Interaction, 2004.
11. Karat J. and Karat C.M. The evolution of User-centered focus in the Human-computer interaction field. IBM Syst. J., 42 (4):532–541, 2003.
12. Kuno Y., Shimada N., and Shirai Y. Look where you're going: a robotic wheelchair based on the integration of human and environmental observations. IEEE Robotics and Automation, 10(1):26–34, 2003.
13. Lyons M.J., Haehnel M., and Tetsutani N. Designing, playing, and performing, with a vision-based mouth Interface. In Proc. Conf. on New Interfaces for Musical Expression, 2003.
14. Nijholt A. and Heylen D. Multimodal communication in inhabited virtual environments. Int. J. of Speech Technol., 5:343–354, 2002.
15. Norman D.A. and Draper S.W. User-centered system design: New perspectives on Human-computer interaction. Lawrence Erlbaum, Hillsdale, NJ, 1986.
16. NSF Workshop On Human-Centered Systems: Information, Interactivity, and Intelligence (HCS) February 17–19, 1997 (http://www.ifp.uiuc.edu/nsfhcs/)
17. Paradiso J. and Sparacino F. Optical tracking for music and dance performance. In Proc. 4th Conf. on Optical 3-D Measurement Techniques IV, 1997, pp. 11–18.
18. Pentland A. Socially aware computation and communication. Computer, 38(3):33–40, 2005.
19. Wassermann K.C., Eng K., Verschure P.F.M.J., and Manzolli J. Live soundscape composition based on synthetic emotions. IEEE Multimedia Mag., 10(4), 2003.
20. www.cs.berkeley.edu/~jfc/hcc

Human-Centered Multimedia

▶ Human-centered Computing: Application to Multimedia

Human-Computer Interaction

Alan Dix
Lancaster University, Lancaster, UK

Synonyms

HCI; Computer human interaction (CHI); Human interface; Interaction design; Human centered computing; User-centred design; Human factors; Man-machine interaction (obsolete)

Definition

Human–Computer Interaction (HCI) is the study of the way in which computer technology influences human work and activities. The term "computer technology" now-a-days includes most technology from obvious computers with screens and keyboards to mobile phones, household appliances, in-car navigation systems and even embedded sensors and actuators such as automatic lighting. HCI has an associated design discipline, sometimes called *Interaction Design* or *User-Centered Design*, focused on how to design computer technology so that it is as easy and pleasant to use as possible. A key aspect of the design discipline is the notion of "usability," which is often defined in terms of efficiency, effectiveness and satisfaction. However, equally or more important in systems designed for personal use, such as internet shopping, is the idea of user experience – the way people feel about the system as they use it.

Historical Background

The roots of HCI can be traced back to the innovative work of Douglas Englebart at the Augmented Research Center at Stanford in the early 1960s [4] and more pragmatic work of Brain Shackel in the late 1950s [8]. However, it was with the rise of personal computing in the early 1980s that the discipline took shape and got its name(s). Notable was the founding of IFIP Technical Committee 13 on Human-Computer Interaction in 1981 and the first international INTERACT Conference in 1984. Several major national conferences started around the same period ACM CHI in the US, the British HCI conference and Vienna HCI. At this stage the main academic roots were in psychology, computing and ergonomics with much of the early work founded on strong experimental methods.

As in so many areas of computing XEROX PARC played a key role in HCI, developing the WIMP (window-icon-menu-pointer) interface style that was eventually popularized in the Macintosh in 1984. In the second half of the 1980s a number of books were published that shaped the development of the field. In particular the *User-Centered System Design* collection in 1986 [6] set the ground for what would be the dominant view of HCI for many years with a strong focus on cognitive modeling and theoretical accounts of *direct manipulation*. As a contrast to this, Suchman's *Plans and Situated Actions* [12] drew on anthropological traditions and in particular *ethnography* and was formative in the fledgling area of *Computer Supported Collaborative Work* (*CSCW*).

Towards the end of the 1980s and early 1990s technological advances as much as methodological and theoretical factors had a large impact on HCI. The introduction of computer networks both within and between offices was one of the factors that fueled the growth of CSCW as it became possible to develop systems that enabled remote groups of workers to collaborate electronically. Increased graphics and computational capabilities also meant that for the first time it was possible to have high-quality *interactive* visualization of large data sets.

The importance of HCI as a discipline has grown gradually over the years as industry became aware of the importance of the usability of its products in a competitive environment. This has intensified since the late 1990s due to the world-wide web and consumer electronics such as mobile phones. As individual consumer choice took over from corporate IT decisions not only usability but also user experience have become key selling points.

With this growth the field has also spawned subfields such as web usability and mobile HCI, which can be confusing to someone new to the area. HCI has also drawn in fresh theoretical input, for example *Activity Theory* (see [1], chapter 11), as well as developing its own methods, for example ways of studying users in real-world contexts.

Foundations

As noted, HCI is both an academic discipline studying the way technology impacts human activity and also a design discipline aimed at designing that technology for maximum effectiveness. It draws on many results from other disciplines such as psychology and sociology, but also has its own methods and techniques.

Usability

Usability is one of the core issues in HCI. There are various definitions, most notably in ISO 9241–11 it is defined as: "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."

The key terms in this definition are:

Effectiveness–Can the user achieve their goals with accuracy?
Efficiency –Can this be achieved with minimal resources: time, physical, and mental effort?
Satisfaction–Do users feel comfortable or happy in doing this?

For many years satisfaction has been largely ignored, but more recently affective issues such as motivation, trust, enjoyment, and engagement have become increasingly important.

The concept of usability is quite broad. At the lowest level there is the visual layout of information and controls on a screen or on a physical appliance and their immediate behavior. At a higher level one also has to take into account the whole social and organizational context: the people who will use the system being designed, their beliefs and values, the purpose and constraints of the design.

Observation and Empirical Data

Various techniques are used to observe or evaluate technology in use (see also [3] chapter 9 and [9] chapters 12–14). These techniques may be used purely for research, early in the design cycle in order to understand a situation, or later when an artifact has been reduced and one desires to evaluate it and identify any usability problems.

These evaluation or observation methods fall into two main types:

1. *Laboratory experiments* – where users perform a task or interact with an application under some level of controlled conditions
2. *Field studies* – where users are "in the wild"; that is in their workplace, outside, in their homes: wherever they would normally interact with the technology being studied or evaluated

Laboratory studies are often criticized for lack of *ecological validity*, that is that they are too far from the real context of use. To counter this laboratory studies are often performed in semi-realistic situations and a *usability laboratory* may include features such as a one-way mirror to enable unobtrusive observation.

Field studies may include a level of control, for example, giving two groups of people different versions of an interface, or may be more open, simply observing people doing their day-to-day work or activities.

Both types of empirical study can be performed with a closed agenda to evaluate or improve a proposed or existing design, but can also be used in a more open or exploratory way to explore a design idea or attempt to learn unexpected things about an area.

Design and Methodology

It often requires too much time or skill to apply fundamental knowledge to a new design. Instead good practice is often captured in design guidelines, principles or heuristics, for example providing immediate feedback for user actions. There are many collections of detailed guidelines for example the US Department of Health and Human Services *Research-Based Web Design and Usability Guidelines* runs to 18 chapters [13], but also there are shorter lists of more generic rules such as Ben Shneiderman's Eight Golden Rules of Interface Design [10]:

- Strive for consistency
- Enable frequent users to use shortcuts
- Offer informative feedback
- Design dialog to yield closure
- Offer simple error handling
- Permit easy reversal of actions
- Support internal locus of control
- Reduce short-term memory load

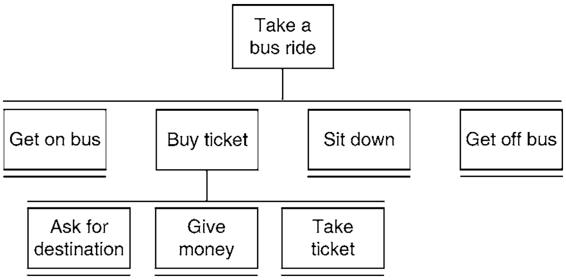
As important as design guidance are the design methods and processes. *User-centered design* refers to a number of practices that put the user first in thinking about the design of an application or product; this will typically involve some direct observation, interviewing of users or otherwise getting to know them. More radical are *participatory design* approaches where users are brought more deeply into the process acting as co-designers of the end product (see [7] Part VII).

Typically user-interface design processes are iterative involving some form of *prototype* or *storyboard* that enables users to see or experience early designs. This reflects the complexity of designing for real people and also the difficulty of even understanding what is wanted. Some level of iterative re-design is always required. However, this iterative design process typically needs a good start point so that more predictive and more iterative methods work together.

Representation and Analysis

Given the focus on users, HCI uses various forms of representation of users and their behavior. These may be informal such as personae or scenarios. A *persona* is a rich description of a real or fictitious user that can be used as a surrogate user in design. A *scenario* is a form of story describing a context in which a device or application is used and the way it gets used. These are both conveyed in text and pictures. More formal representations can also be used, for example, *hierarchical task analysis* (HTA, see [3] chapter 15) focuses on the decomposition of higher-level tasks (such as “take a bus ride” in Fig. 1) into lower evel tasks (such as “buy ticket”).

Various forms of representations are also used to describe the behavior of interactive systems themselves; that is how the system responds to user inputs, the responses it produces, and the internal states insofar as they may affect future user interaction. This is often called a *dialogue specification*. These sometimes take the form of fairly informal diagrams, for example showing a collection of web pages and their interconnections. However, more formal representations may be used, for example, using state-transition networks or Petri nets. The more formal representations can be analyzed to look for potential usability problems or even verified using forms of model checking to verify various forms of usability property. More commonly however, these are used to understand or communicate designs, or to specify them clearly for implementation.



Human-Computer Interaction. Figure 1. Example of an hierarchical task analysis (HTA).

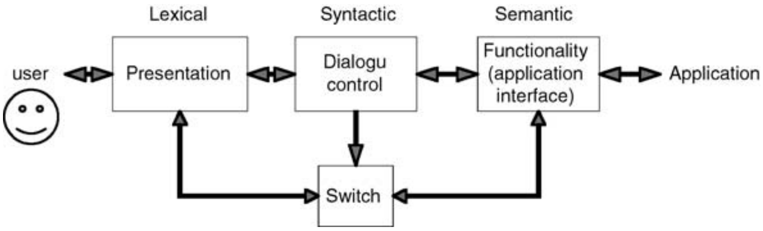
Implementation and User Interface Architecture

For the users of a system all that matters is its external behavior. However, in order to effectively construct such systems the internal structure or *software architecture* has to be in some way consonant with the external design (see also [3] chapter 8). The view that the user has of a system frequently cuts across what would be regarded as different functional areas, sometimes conflicting with what would otherwise be regarded as a suitable decomposition. It is also often desirable that more surface features of a user interface can be changed without a complete redesign of the entire system. Because of this, specific architectures have been developed that separate the parts that are closer to the user (e.g., choice of menu names or font color) from those about the underlying functionality. The most influential of these at a conceptual level and one of the earliest was the *Seeheim model*, which divided the system into presentation, dialogue and functionality (with some form of wrapper) (Fig. 2). At a practical level the *Model-View-Control* (MVC) model has been most widely adopted (for example in the *Java* Swing toolkit). This operates on the level of components or widgets (for example, the tree-view of a file system).

Key Applications

HCI is an issue whenever people use systems so the applications are very wide. Traditionally a substantial focus has been on office work where the desktop PC was the main computing device, although there has always been a strand focused on industrial interfaces and command and control situations (e.g., aircraft cockpits). As noted previously, domestic appliances and devices and systems for leisure and entertainment have become increasingly important.

Two of the biggest application areas over recent years have been in the web [2] and mobile computing [5]. However, these are really platforms and the range



Human-Computer Interaction. Figure 2. Seeheim model of user interface architecture.

applications supported on these have ranged from eCommerce to text messaging.

There has been some work focused specifically on user interfaces for databases to aid in areas such as query formulation or schema visualization. Related to this has been more recent work on usability issues for web search and user interfaces for various forms of ontologies and other semantic web structures. Applications in *information visualization* are also important allowing users to view structures including simple point data, hierarchies, trees, or geographic data [11].

Future Directions

The web will clearly be an important driver for some time to come. Whilst traditional web page usability has been well studied, the implications of *Web2.0* both as a social and a technological phenomenon are only just beginning to be understood. For example, users are increasingly making their own "mashups" and customizing shared web content, so that effectively they are taking on the role that used to be in the hands of developers – it is important that this does not become as confusing as cooking in someone else's kitchen! Developments in HCI aspects of mobile applications will continue to be important, especially as it is expected that mobile access to the Internet will outstrip desktop access in the next few years.

Looking slightly longer term, human aspects of *ubiquitous computing* are still only partially understood, but will be critical as homes, offices and public spaces become filled with sensors, points of interaction and numerous small and large screens. As these increasingly become wirelessly networked they will need to be designed in way that make their effects comprehensible and trustworthy.

In work environments technology is more stable and usability of desktop applications well understood. However, increasing availability of data and communications (e.g., email, instant messaging) is creating demands to manage large volumes of data. This may change the traditional desktop itself and is likely to involve more "intelligent" and often proactive applications, that will in turn need to be understood and controlled by their users.

Mobile and web applications are creating demands to re-target applications to multiple devices, and to share and mash-up data. For cost-effective construction and ease of maintenance it is likely that studies of user interface architecture, marginalized for a number of years, will become more important.

Web-based applications are leading to service-based solutions replacing many traditionally product-based ones. Greater choice and ease of movement between applications will demand changes in design practice away from "one size fits all" designs to more individual designs. This trend reflects similar diversification and shorter more dynamic product runs in traditional manufactured devices, which themselves are more likely to involve computation. This will clearly impact both design research and design practice.

Finally the demographics of computer and technology use is moving out from the relatively young and prosperous to include the increasingly older populations of developed countries, socially-deprived groups, and the growing markets of the developing world. While much of the understanding of human-computer interaction is universal, still many cultural and other assumptions are embedded in design practice and this will be an important issue for some years to come.

Cross-references

▶ Data Visualization

▶ Direct Manipulation

▶ GUIs for Web Data Extraction

▶ Human-Centered Computing: Application to Multimedia

▶ Icon

▶ Interface

- ▶ Mobile Interfaces
- ▶ Natural Interaction
- ▶ Usability
- ▶ Visual Interaction
- ▶ Visual Interfaces
- ▶ Visual Perception
- ▶ WIMP Interfaces

Recommended Reading

1. Carroll J. HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science. Morgan Kaufmann, San Francisco, CA, 2003.
2. Dix A. Human-computer interaction and web design, Chapter 3. In Handbook of Human Factors in Web Design, R.W, Proctor K.L. Vu (eds.). Lawrence Erlbaum, Mahwah, NJ, 2005, pp. 28–47.
3. Dix A., Finlay J., Abowd G., and Beale R. Human-Computer Interaction, 3rd edn. Prentice Hall, Upper Saddle River, NJ, 2004.
4. Engelbart D. Augmenting human intellect: a conceptual framework. Summary Report AFOSR-3223 under Contract AF 49 (638)-1024, SRI Project 3578 for Air Force Office of Scientific Research. Stanford Research Institute, Menlo Park, CA, 1962. Available online at: http://www.bootstrap.org/.
5. Jones M. and Marsden G. Mobile Interaction Design. Wiley, Chichester, UK, 2005.
6. Norman D.A. and Draper S.W. (eds.). User Centered System Design: New Perspectives on Human-Computer Interaction. Lawrence Erlbaum, Mahwah, NJ, 1986.
7. Sears A. and Jacko J. 1(eds). The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications, 2nd edn. CRC Press, West Palm Beach, FL, 2007.
8. Shackel B. Ergonomics for a Computer. Design, 120:36–39, 1959.
9. Sharp H., Rogers Y., and Preece J. Interaction Design: Beyond Human-Computer Interaction, 2nd edn. Wiley, Chichester, UK, 2007.
10. Shneiderman B. and Plaisant C. Designing the User Interface: Strategies for Effective Human-Computer Interaction, 4th edn. Addison-Wesley, Boston, MA, 2005.
11. Spence R. Information Visualization: Design for Interaction, 2nd edn. Prentice Hall, Uppersaddle River, NJ, 2007.
12. Suchman L. Plans and Situated Actions: The Problem of Human-Machine Communication. Cambridge University Press, New York, NY, 1987.
13. US Department of Health and Human Services. Research-Based Web Design and Usability Guidelines. Available online at: http://www.usability.gov/pdfs/guidelines.html (accessed on November 12, 2007).

Hypercube

▶ Cube

Hypermedia

▶ Hypertexts

Hypermedia Metadata

▶ Multimedia Metadata

Hypertexts

Frank Wm. Tompa
University of Waterloo, Waterloo, ON, Canada

Synonyms

Hypermedia

Definition

A hypertext is a collection of interconnected documents or document fragments. The idea of computer-based hypertexts is rooted in Vannevar Bush's vision, as described in his 1945 *Atlantic Monthly* article "As We May Think," of a personal document collection with "a provision whereby any item may be caused at will to select immediately and automatically another." Authors who wish to emphasize the multimedia nature of constituent documents and fragments prefer to use the term hypermedia when describing hypertexts.

Key Points

The term *hypertext* was introduced in the early 1960s by Ted Nelson, who advocated the power of non-linearity in organizing thoughts and discourses. Simultaneously, Doug Englebart demonstrated a system for "augmenting human intellect" that included a facility to expose inline fragments of text in response to users' request for finer detail. These early ideas have evolved to form the basis of HTML (the Hypertext Markup Language) and the World Wide Web.

Hypertexts have been used extensively as features of reference texts, collaborative design documents, and bibliographic systems, and as a structure for creative writing. Links between document fragments are used to represent various rhetorical relationships (support, rebuttal, elaboration, motivation, consequence, etc.).

Systems to support hypertexts include editors, browsers, and visualizers.

Cross-references

- ► Document Databases
- ► Digital Libraries
- ► Web Characteristics and Evolution

Recommended Reading

1. Ashman H. and Simpson R.M. Computing surveys' electronic symposium on hypertext and hypermedia: editorial. ACM Comput. Surv., 31(4):325–334, 1999.
2. Simpson R., Renear A., Mylonas E., and van Dam A. 50 years after "As we may think": the Brown/MIT Vannevar Bush symposium. Interactions, 3(2):47–67, 1996.

Hypothesis Generation and Exploration from Biological Resources

► Text Mining of Biological Resources

I/O cache

► Buffer Pool

I/O Model of Computation

Donghui Zhang[1], Vassilis J. Tsotras[2]
[1]Northeastern University, Boston, MA, USA
[2]University of California-Riverside, Riverside, CA, USA

Synonyms

Disk-based model

Definition

The I/O model of computation measures the efficiency of an algorithm by counting how many disk reads and writes it needs. It is widely applicable to the database environment, since most data is stored on disks and disk access typically dominates CPU time.

Key Points

For many computing-intensive applications, the appropriate model of computation is to measure CPU time. Yet in data-intensive applications, such as databases, it is more relevant to measure the number of disk I/Os [1]. This is termed the "I/O model of computation," or disk-based model. Nowadays, most hard drives use the seek-rotate-transfer protocol [2]. In order to transfer some data from disk to memory (so as the data can be processed by the CPU), or to transfer data back to disk, the hard drive needs first to spend some "seek time" to move the read/write head to the cylinder where the data is located at. Then the "rotational delay" is spent until the sector containing the data rotates to a position under the read/write head. Finally, time is spent to actually transfer the data from/to the CPU. Typically, seek time is longer than rotational delay, which is in turn longer than transfer time. Therefore reading a few bytes of data takes roughly as long as reading thousands of bytes. Due to this reason, data is stored on disks in units called blocks or pages. Every disk I/O corresponds to reading or writing one such page. Moreover, a random disk I/O costs more than a sequential access. This is because the access of multiple sequential pages on the disk does not involve major seek and rotational times (since in sequential access, a page is accessed after its neighbor page). Hence a more accurate I/O model should account for the difference between random and sequential I/O.

There are three ways to minimize the disk accesses in a database environment: (i) by buffering in main memory pages that have already been accessed (and thus future accesses can be served by a buffer access and not a disk access), (ii) by transferring a number of consecutive pages at once, called bucket, anticipating the next requests due to data locality, and, (iii) by using structures (indices) that organize the data into pages so as searching for a particular record takes few page accesses.

To exemplify the importance of the I/O computation in data structures, consider the following scenario. Assume an application (query) requests a record from a database file. If a balanced binary search tree is directly implemented on top of this file, to search for a record would need $O(\log_2 n)$ I/Os, where n is the number of records in the tree. For example, if n is a million records, this means about 20 I/Os. If instead, a disk optimized structure is used (like the B+-tree) the same search is much more efficient (in number of page I/Os). The B+-tree extends the binary tree, by expanding a node size. Every tree node corresponds to one disk page. If for simplicity, every index node as well as leaf node in the B+-tree contains 100 entries, a million records fits into a three-level B+-tree. Then to search for any record in the file, only three I/Os are needed, which is much faster than the 20 I/Os. Disk resident data structures are called paginated or disk-based or external.

Before a new page read is executed, the buffer manager is first examined for whether it already contains the requested page. For instance, if a page is read 100 times, only the first read triggers an actual disk read, while subsequence reads are serviced by returning the in-buffer page (and thus avoiding the cost of a page I/O). Of course the buffer has limited capacity and issues like page replacement policies play an important role.

Cross-references

▶ Access Methods
▶ Buffer Management
▶ Indexing

Recommended Reading

1. Aggarwal A. and Vitter J.S. The input/output complexity of sorting and related problems. Commun. ACM, 31 (9):1116–1127, 1988.
2. Ramakrishnan R. and Gehrke J. Database Management Systems (3rd edn.). McGraw-Hill, New York, NY, 2003.

Icon

Stefano Levialdi
Sapienza University of Rome, Rome, Italy

Synonyms

Picture; Image; Representation

Definition

Originally, the Greek word *eikon* stood for an image, carrying some meaning as in typical portraits of sacred persons within the Orthodox Church. An operational definition of icon was given by Peirce [1] as *anything that stands for something else, to somebody, in some respect or capacity*. Being so general, it covers most customary practices, typically linking linguistic, pictorial (or even auditory) expressions to a *meaning* that needs to be *interpreted* by a human. In general discourse, an icon may imply an idol (as a pop star) or a symbol (the Rotary wheel worn on coat lapels) that represents a group of persons or a life style (the Nike symbol for sports). Within Human-Computer Interaction, the common understanding of an icon is to consider it as a visual metaphor representing a file, a directory, a window, an option or a program. Whenever a number of icons are presented together, this group is referred to as an icon bar, generally at the top of the page in most web browsers.

Key Points

The study of signs, a scientific discipline under the name of Semiotics, started from the work of two contemporary researchers: Ferdinand de Saussure (a linguist) [2] and Charles Sanders Peirce (a philosopher) [1]. De Saussure showed that language may be considered as a collection of *signs*, each one formed by a pair (a twofold nature) named signifier/signified, and only if both were present, the sign could be a valid indication of a meaning, otherwise the sign only represents itself. Peirce made a well known triangular classification (a threefold nature), where the sign stands in the center and the three vertices correspond to (i) the representation, (ii) the object (referent) and (iii) the interpretant (meaning) [3]. The main issue here is that the interpretation is subjective and dynamic, there is no unique meaning corresponding to a class of signs: the *icons*. Following Peirce's sign taxonomy, he indicated three possible sign classes: the *icon* (where a mental process is required to understand it), the index (having a causal relationship to its signified, like smoke for fire) and the *symbol* (having a totally arbitrary relationship, like a red cross for the corresponding, well known, international medical organization). The icon is wrongly considered to be similar to its signified; typically by looking at the icon one should infer information about its signified, yet this is not always the case, since different meanings may be attributed to the icon, depending on the observer. This fact provides a possible approach to ambiguity management in human-computer interfaces, by restricting the user model to a given class of users. The design of effective icons should be performed after an accurate study of the cultural background, age, and motivation of the potential users.

Cross-references

▶ Symbolic representation
▶ Visual metaphor
▶ Visual representation

Recommended Reading

1. Charles Sanders Peirce. Collected Papers of Charles Peirce, vol. 1–8, C. Hartshorne and P. Weiss (eds.). Harvard University Press, Cambridge, MA, 1931–1958.

2. de Saussure F. Introduction to the 2nd Course on General Linguistics (1908–1909), collected by Robert Godel, Raffaele Simone (ed.). Italian Edition, Ubaldini, Rome, 1970.
3. de Souza C. S. The Semiotic Engineering Approach to Human-Computer Interaction. The MIT Press, Cambridge, MA, 2005.

Iconic Displays

Georges Grinstein, Damon Andrew Berry
University of Massachusetts, Lowell, MA, USA

Synonyms

Icons; Glyphs; Iconographics

Definition

Iconic displays are visualizations that generalize traditional displays (especially scatterplots) where each record, instead of being drawn as a point, is represented by a more general primitive called an icon or glyph. The goals are to harness human perception, especially texture, and to display many more parameters. Whereas a pixel is driven by three data values from some color model (typically red, green, and blue) an icon is a geometric object driven by potentially many values, with some icons displaying over 30. Some icons are drawn using lines, some using colored areas, some move and vibrate, and some even have sound output. Some iconic displays drop the Cartesian base of the underlying display and use alternative layout techniques. However, in all of these, the key defining factor is the representation of a record in a visualization by a very general, most often geometric, primitive, with the goal of producing more perceptually-based displays.

Historical Background

The analysis of complex images is still performed largely by human visual means which is most highly effective in situations where the patterns of potential interest are directly visible. The most widely used technique for displaying multivariate data or multiparameter imagery is based on the representation of different parameters as points with two or three position coordinates and color based on a color model such as RGB (Red, Green, Blue) or HLS (Hue, Lightness, Saturation).

Such scatterplots employ simple graphic representations with attributes (retinal variables) characterized by Bertin [2] and Healy et al. [8]. The number of variables or parameters that can be displayed with such a two dimensional scatterplot is easily five using five degrees of freedom (x, y, shape, size, and color) and seven if one thinks of color as three separate variables (e.g., RGB).

But what if one wishes to go beyond five or seven variables? This was the motivation for the development of icons. By increasing the number of variables encoded with more complex representations, an arbitrary number of variables can be encoded. Examples include Chernoff faces [4], asymmetrical faces [5], star glyphs [3], the stick figure icons of Pickett et al. [1,7,13], color icons [11], moving icons [15], and moxels [19].

In addition to the different icons, there are a number of different algorithms for placing them on a two-dimensional plane. Ward presents an overview and taxonomy of such icon or glyph placement strategies [17].

Foundations

Perceptual Foundations

Gibson [6] developed an ecologically-based approach to perception, quite different from the then conventional approach. The human visual system is an ecologically evolved system still guiding the behavior of animals. Humans discriminate textures very effectively and use variations in visual texture as important sources of information in the detection and recognition of objects. Pickett [13] suggested that modern displays (visualizations) should be developed using this ecological spatiotemporal pattern recognition system of the human optical system. Further, experiments developed by Julesz, demonstrated that differences in textons (characteristic features of the surface) or their densities can be preattentively detected by the human visual system [9]. The work developed by Julesz shows that texture is a statistical property of textons. The perception and discrimination of textures seems to be based on the density (first order statistics) of textons.

In 1967, Jacques Bertin [2] in Sémiologie Graphique presented the fundamentals of information encoding via graphic representations as a semiology, a science dealing with sign-systems. His first key point is the strict separation of content (the information to encode) from the container (the properties of the graphic system). Bertin defined a graphical vocabulary consisting of three elements: marks, positional variables, and retinal variables. Marks are artifacts like points, lines, and areas. Positional variables are (usually) two planar dimensions. Retinal variables are encoded entities such as size, color, shape, orientation, texture, and value. These were extended by others to include temporal aspects with animation and work in three dimensions. Additional parameters such as shading, connectivity, labels, visibility, and tabularity were added. Extensions to the non-visual have also appeared such as the use of sound as a perceptual variable [16] as well as touch and smell.

In all of the above the goal is to develop perceptually salient displays.

Iconographics

Iconographics initially developed at the University of Massachusetts Lowell, harnesses texture perception to increase the dimensionality of the represented data visible in an image [13]. Each datum is represented by an icon whose visual features are controlled by the data. Two of the data variables control the position of each icon on the display surface. When icons are used to represent imagery data, the icon position simply corresponds to the pixel position. With sufficient density, the icons form a surface texture display, and structures in the data are revealed as streaks, gradients, or islands of contrasting texture.

Humans discriminate textures very effectively and use variations in visual texture as important sources of information in the detection and recognition of objects. Based on the early work of Pickett, Pickett and Grinstein [13] developed data representation techniques that engaged and harnessed the mechanisms of texture perception. Later, Levkowitz et al. [14] came back to color as a basis for integrating multiparameter images, but with a more sophisticated approach based textural representations.

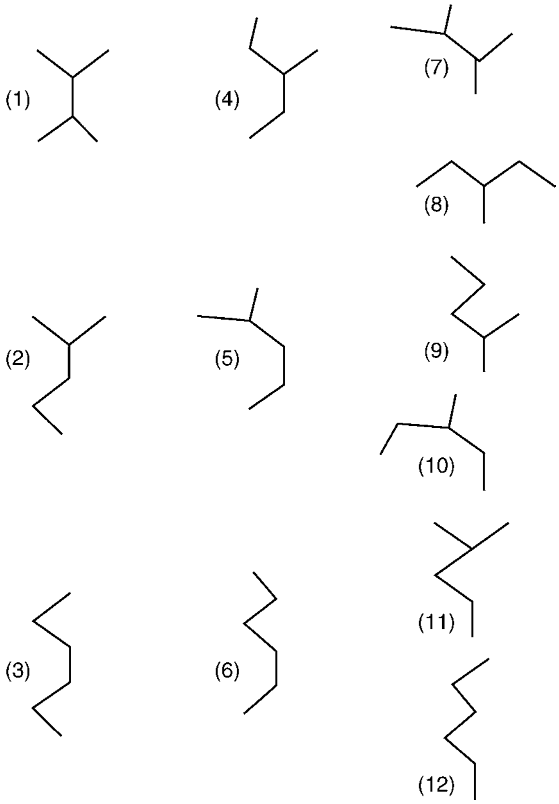
The iconographic technique extends the classic visualization approaches by representing each record of a multidimensional database as an icon whose features (such as color, geometry, and sound) are under the control of the various fields of the record. This technique allows the information content of the objects represented to have high dimensionality and allows for the fusion of multiple data sets (usually images) into a single integrated multi-modal display.

This is done through an icon which is an $m \times n$ box of screen pixels. A single pixel can be thought of as a 1×1 icon. Because an icon is a box of pixels, it can represent many more attributes (data parameters) than a single screen pixel. The increased per-element information content comes at the price of decreased screen resolution, a tradeoff that must be considered in the design of any iconographic display.

Stick Figure Icons

Figure 1 shows a typical stick-figure icon. The line segments are called the "limbs" of the icon. Each icon has a special limb, called the "body," which serves as a reference for the various geometric transformations that an icon can undergo. The first complete family of stick-figure icons had twelve members, each of which has up to five limbs connected in a unique configuration. The five-limbed icon [1] was designed to display multiparameter images on a single display in a way that would engage the vision system's ability to perceive textures.

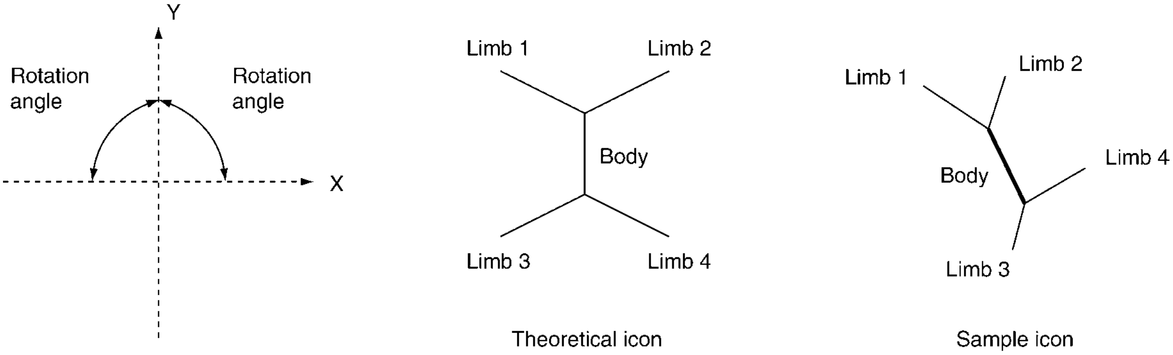
Each limb has at least three parameters that can be bound to data attributes: the angle, intensity, and length. The manner in which limbs are attached to the body or to each other defines the family of stick figure icons. Figure 2 shows one member of the family of stick figure icons. The figure shows the icon in its base



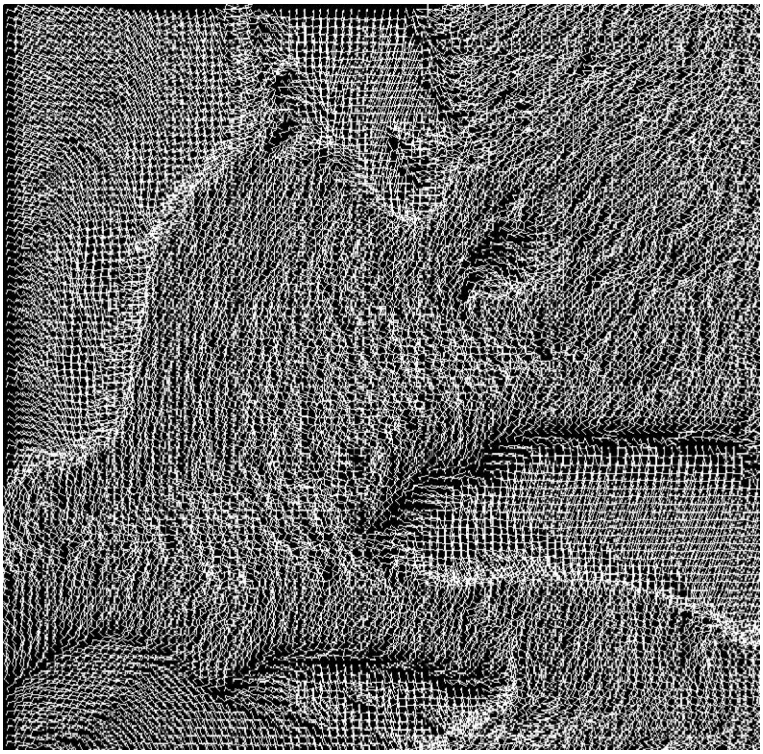
Iconic Displays. Figure 1. Family of stick figure icons.

configuration with no data mapped to any limb parameters and also a sample icon with data mapped to the limbs' parameters. The value of any variable is normalized into the unit interval [0,1]. The variable is mapped to a limb with the limb's reference angle or rotation angle determined from the normalized value (0 = 0°; 0.5 = 90°; 1.0 = 180°).

The general character of displays made with this icon can be seen in Fig. 3, one of Grinstein and Pickett's earliest displays, where they merged four grayscale images. The merged images are satellite images of a portion of the Great Lakes region (readers familiar with this area will immediately recognize Lakes Erie, Ontario, and Huron, as well as Georgian Bay and Lake Simcoe). Two of the images are from the infrared spectrum and two are from the visible spectrum. The goal is to show in one picture both the fine detail from the visible images and the heat signatures of the infrared images (here, the lakes are colder than the surrounding land). The mappings chosen produced an



Iconic Displays. Figure 2. One member of the stick-figure icon family.



Iconic Displays. Figure 3. Satellite image of the Great Lakes – line icon.

integrated display that shows the lakes as flat, concave features in relation to the rougher and predominantly convex land features. These three-dimensional effects are striking and highly effective.

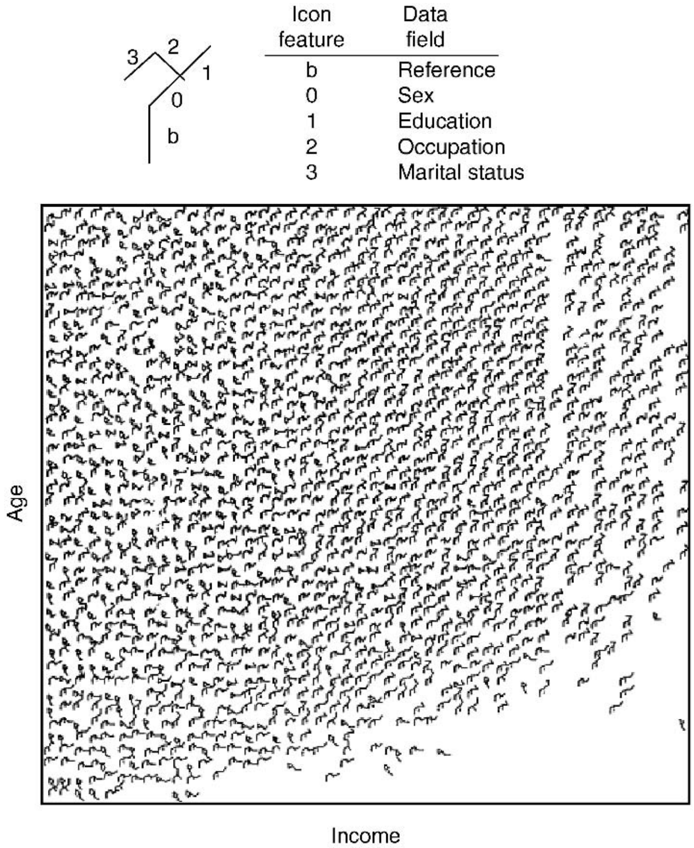
Figure 4 shows a scatterplot of a subset of the PUMS US Census data of engineers and technicians in New England, using the stick figure icon developed by Pickett and Grinstein in 1989. This icon encodes four variables (sex, education, occupation, and marital status). Each variable is mapped to a limb whose angles are determined by the value of the variable. Placing the icon on the scatterplot encodes two more variables (income and age). The result is an iconic display that encodes six variables.

Figure 5 represents some of the possible geometric layouts of icon number 4 from the stick figure family in Fig. 1 for various values of the variables. There are 16 possible configurations of the stick figure icon (Fig. 5), each corresponding to a particular geometric organization of the icon's limbs. Of particular interest are females with a high level of education since the icon forms a triangle, a highly perceptual object. For this data set (engineers and technicians in New England) almost all individuals have a high level of education. Since there are few women engineers and technicians with a low level of education one can state that all icons in Figure 3 having a triangle can be interpreted as female.

It is thus possible to get some insight into this subset. For example one can see the boundary between females and males; one can see that most female's salaries fall within the first third of the salary scale. Once can also quickly see outliers and trends in Fig. 4.

Color Icons

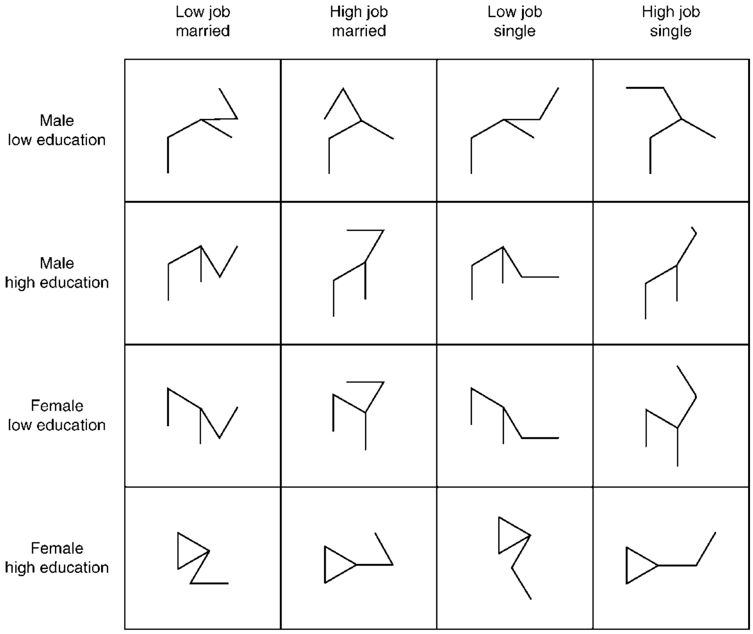
Applying the concepts learned with the stick-figure icon to color, Levkowitz developed a color icon [11]. In the color technique, a pixel in the input data is represented by an arbitrarily-sized *m* x *n* box of screen pixels. In the most frequently used version of the color



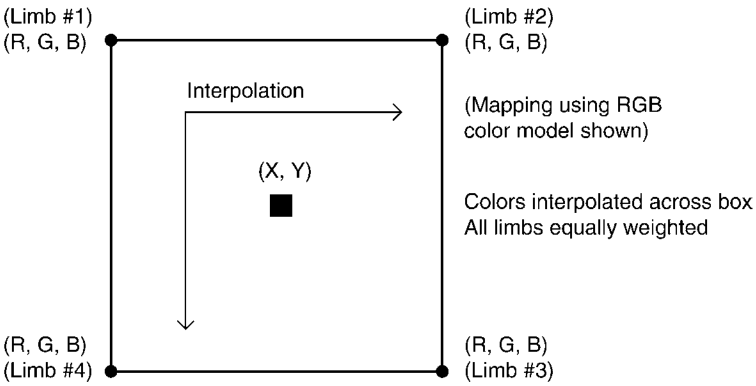
Iconic Displays. Figure 4. PUMS US Census data using a Picket icon.

icon, each of up to 12 data parameters is mapped to a color channel of the four corners of the icon box. Each data value determines the intensity of that color channel. The colors of the remaining pixels in the box are obtained by interpolation. The color icon is independent of color models and was tested with four color models: GLHS (Generalized Lightness, Hue, Saturation), RGB (Red, Green, Blue), Munsell Book of Color, or CIELUV (Commission on Illumination 1976 L*, u*, v*). The design of this icon is shown in Fig. 6. Figure 7 shows the color icon used to display several parameters of the FBI (Federal Bureau of Investigation) homicide database.

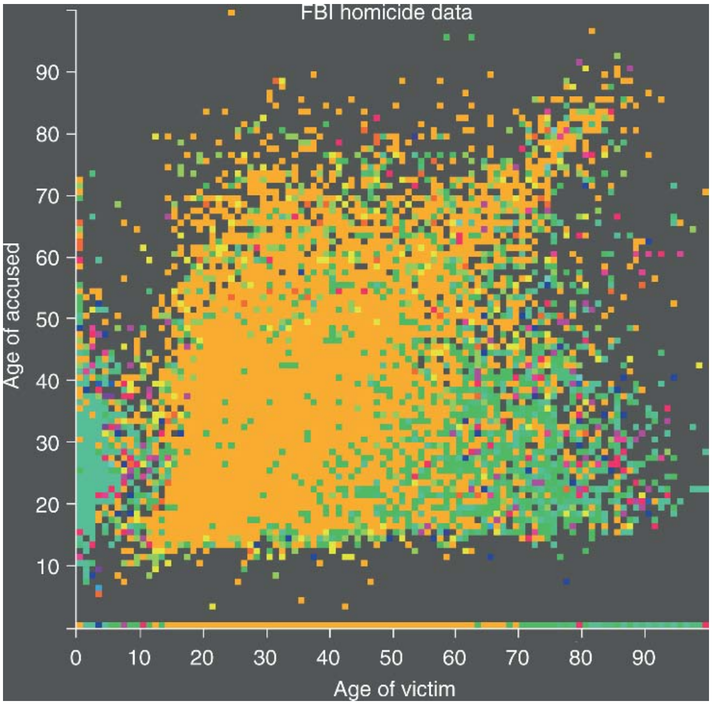
In Fig. 8, standout regions (which are common with other similar color icon displays of the same data set) can easily be identified. The first (labeled A), highlights very young victims with different attributes. The greenish color shows offenders whose homicides were family related. The second (labeled B) is the tail in the display which highlights spouses involved in the homicide. The final obvious pattern (labeled C) is that of zero-aged offenders, clearly produced by missing data.



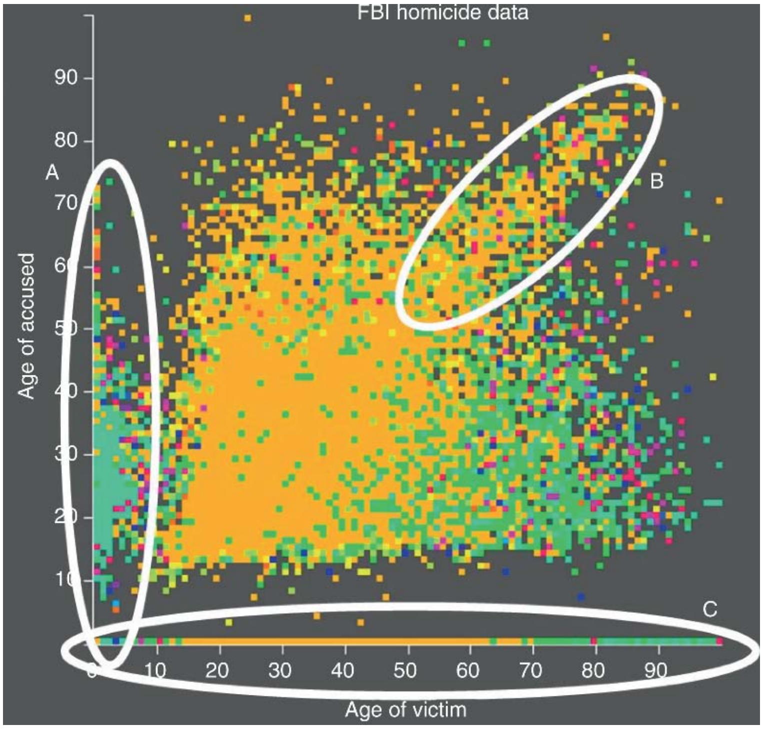
Iconic Displays. Figure 5. The stick figure icon's possible representation for various values of the parameters of a record from the Census data.



Iconic Displays. Figure 6. Design of color icon.



Iconic Displays. Figure 7. FBI homicide data using a color icon.



Iconic Displays. Figure 8. The same figure as in 7, but with three regions of interest.

Kinetic Displays and Moxels

Kinetic displays are extensions of the above icon displays in which motions of the graphical icon's limbs are used to represent attributes of the data, so the elements are in constant motion independent of user interactions. Example motions of data driven graphics include translations and rotations that produce visual vibrations and oscillations. Such kinetic displays are designed to capitalize on the human preattentive ability to perceive and understand motion.

The term "pixel" stands for "picture element;" similarly the term "moxel" stands for "moving element." Moxels are an extension of standard iconographic displays into three dimensional displays that incorporate motion coding of data dimensions.

The first such display was produced by Pinkney [15] with several interactions a "magnet" interaction; a "vibrating" interaction; a "comb" interaction; a "zoom" interaction; and a "pinwheel" interaction. In all of these interactions the mouse is used for specifying a region of interest. Interaction acts over all icons within the selected region. Users can select either a constant value or a variable (data attribute) to drive the interaction. Icons within the selected region are transformed in some way (depending on which interaction) that is proportional to the value driving the interaction. For example in the "magnet" interaction the mouse acts as a magnet repelling or attracting icons. Users can select if the magnet will affect the angle of the icons, the position of the icons, or both. The value associated with the interaction is used to determine the angular and positional response of the icon to the magnet, in the case of magnet interaction, lower values imply greater effect.

Yang et. al. [19] presented an information visualization system that links both static and kinetic visualizations. The images in Fig. 9 show a zoomed out view (left) and a zoomed in view (right) of a moxel visualization of the AVARIS (Airborne Visible/Infrared Imaging Spectrometer) dataset. These static images do not really do justice to the actual moving visualizations, where independent regions are clearly visible not only due to common textures, but due to coordinated movements as well.

Key Applications

Iconographic displays have become part of many different visualization efforts and environments. They have been used for both information visualization as well as scientific visualization in areas such as GIS (Geographic Information System) [20], physics [18], medicine [10], and computer network security [12].

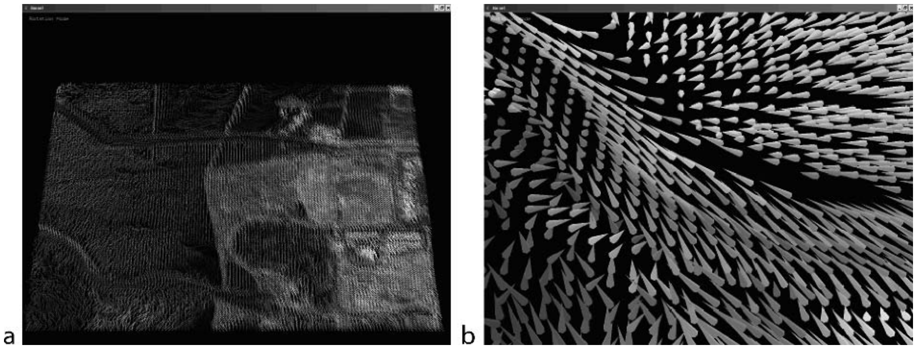
Data Sets

The PUMS US Census data used in Fig. 1 was from the 1980 census. Comparable data fro the last conducted US census may be found at: http://www.census.gov/main/www/pums.html.

The Great Lakes satellite image in Fig. 2 is based on data from the National Oceanic and Atmospheric Administration (NOAA) and is available from the National Geophysical Data Center (NGDC) at: http://www.ngdc.noaa.gov/dmsp/download.html.

The AVIRIS data used for the moxel visualizations in Fig. 9 is available at: http://aviris.jpl.nasa.gov/html/data.html.

The FBI Homicide Data visualized in Figs. 6 and 7 is available at: http://www.fbi.gov/ucr/ucr.htm.



Iconic Displays. Figure 9. Moxel visualizations of the AVIRIS dataset: a zoomed out image (left) and a zoomed in image (right).

Cross-references

- ▶ Applied Perception
- ▶ Glyphs
- ▶ Iconographics
- ▶ Icons
- ▶ Perceptual Displays
- ▶ Visual Perception
- ▶ Visualization

Recommended Reading

1. Bergeron R.D. and Grinstein G.G. A reference model for scientific visualization. In Proc. Eurographics '89', 1989, pp. 393–399.
2. Bertin J. Semiology of Graphics. The University of Wisconsin Press, Madison, WI, 1983.
3. Chambers J.M., Cleveland W.S., Kleiner B., and Tukey P.A. Graphical Methods for Data Analysis. Wadsworth, Belmont, CA, 1983.
4. Chernoff H. The use of faces to represent points in k-dimensional space graphically. J. Am. Stat. Assoc., 68:361–368, 1973.
5. Flury B. and Riedwyl H. Graphical representation of multivariate data by means of asymmetrical faces. J. Am. Stat. Assoc., 76:757–765, 1981.
6. Gibson J.J. The Ecological Approach to Visual Perception. Houghton-Mifflin, Boston, 1979.
7. Grinstein G., Pickett R., and Williams M.G. EXVIS: an exploratory visualization environment. In Proc. Graphics Interface '89, 1989.
8. Healey C.G., Booth K.S., and Enns J.T. Visualizing real-time multivariate data using preattentive processing. ACM Trans. Model. Comput. Simul., 5(3):190–221, 1995.
9. Julesz B. Textons, the elements of texture perception, and their interactions. Nature, 290:91–97, March 1981.
10. Kindlmann G., Weinstein D., Lee A., Toga A., and Thompson P. Visualization of anatomic covariance tensor fields. In Proc. 26th Annual Int. Conf. of the IEEE Engineering in Medicine and Biology Society 2004, pp. 1842–1845.
11. Levkowitz H. Color icons: merging color and texture perception for integrated visualization of multiple parameters. In Proc. IEEE Conf. on Visualization, 1991, pp. 164–170.
12. Perlman J. Visualizing network security events using compound glyphs from a service-oriented perspective. Available at: http://www.csee.umbc.edu/gavl/theses/jpearlman.pdf, 2007.
13. Pickett R.M. and Grinstein G.G. Iconographic displays for visualizing multidimensional data. In Proc. 1988 IEEE Conf. on Systems, Man and Cybernetics, 1988, pp. 514–518.
14. Pickett R.M., Grinstein G., Levkowitz H., and Smith S. Harnessing preattentive perceptual processes in visualization. Perceptual Issues in Visualization, Springer, NY, 1995, pp. 33–45.
15. Pinkney D. Intelligent Iconic Visualization. Ph.D thesis, University of Massachusetts, Lowell, 1997.
16. Smith S., Bergeron R., and Grinstein G. Stereophonic and Surface Sound Generation for Exploratory Data Analysis. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1990.
17. Ward M. A taxonomy of glyph placement strategies for multidimensional data visualization. Inf. Vis., 1:194–210, 2002.
18. Wittenbrink C., Pang A., and Lodha S. Glyphs for visualizing uncertainty in vector fields. Trans. Vis. Comput. Graph., 2 (3):266–279, 1996.
19. Yang F., Goodell H., Pickett R., Bobrow R., Baumann A., Gee A., and Grinstein G.G. Data exploration combining kinetic and static visualization displays. In Proc. 4th Int. Conf. on Coordinated & Multiple Views in Exploratory Visualization, 2006, pp. 21–30.
20. Zhang X. and Pazner M. The icon imagemap technique for multivariate geospatial data visualization: approach and software. Cartogr. Geogr. Inf. Sci., 31(1):29–41, 2004.

Iconographics

▶ Iconic Displays

Icons

▶ Iconic Displays

Identity Disclosure

▶ Disclosure Risk

Identity-based Access Control

▶ Discretionary Access Control

IDF

▶ Inverse Document Frequency

IF

▶ Information Filtering

ILM

▶ Information Lifecycle Management (ILM)

Image

Valerie Gouet-Brunet
CNAM Paris, Paris, France

Synonyms

Multimedia; Digital image; Picture; Photograph; Synthetic image; Graphics

Definition

Image comes from *Imago* in Latin and designates funeral masks. Philosophically, an image represents the static and eternal double of a volatile or ephemeral reality. More commonly, it is a two-dimensional artifact that either records the visual appearance of physical objects, like photographs, or provides a visual representation of concepts or artificial data, like graphics or synthetic images. *Digital* images were born in the early 1920s, as a representation of a two-dimensional image using ones and zeros (binary) obtained by digital cameras, scanners or dedicated materials and softwares. They exist in different forms, as illustrated in Fig. 1. Nowadays, digital images are everywhere. They are involved in a large number of leading applications and cover various domains from medicine to video games, including architecture or robotics.

Historical Background

The first use of digital images dates back to the beginning of the twentieth century with the technological

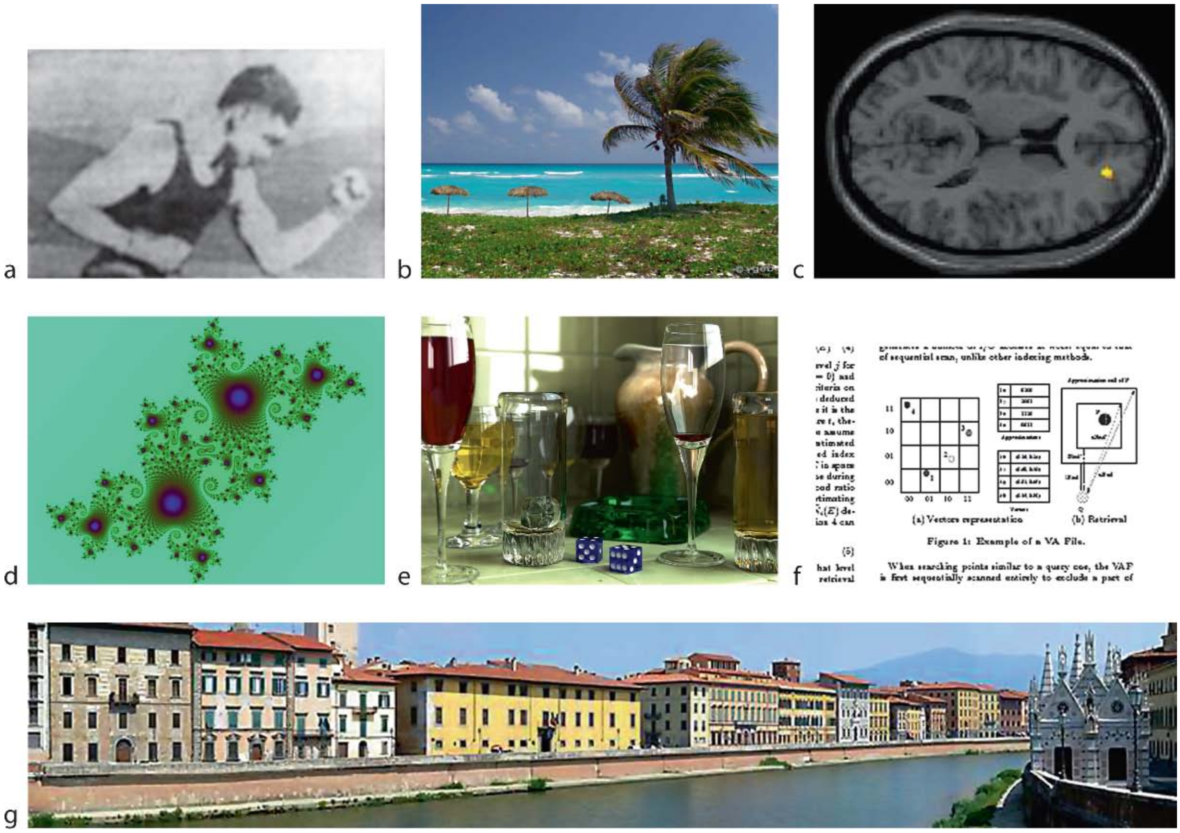


Image. Figure 1. Different representations of digital images: (a) Early digital image reproduced in 1921 by a telegraph printer with special type faces from a coded record (from McFarlane's article, 1972) (b) Photograph taken with a digital camera (c) Axial fMRI slice of a human brain (d) Synthetic image "Julia set," a fractal related to the Mandelbrot set (e) Synthetic image created by G. Tran with POV-Ray 3.6 using Radiosity (f) Zoom on a PDF document where images are vector graphics (g) Panorama stitched from 6 images with image mosaicing technique.

development of facsimile transmission for the newspaper industry. Among several systems developed for reducing picture transportation time, the Bartlane cable transmission system, invented in 1920 in Great Britain, was the first system that translated pictures into a digital code (Baudot code) for efficient picture transmission [7]; in Fig. 1, image (a) represents an image transmitted with the Bartlane system and reproduced by a telegraph printer. Scanners can be considered as the successors of telephotography input devices. The first digital image made on a computer came from a drum scanner associated with photomultiplier tubes built in 1957 by Russell Kirsch at NIST (US National Institute of Standards and Technology). But drum scanning declined and now relates to specialized applications, due to the development of low-cost flatbed and film scanners.

The advent of digital image technology is also closely tied to the development of computers as well as remote sensing applications. The first computers powerful enough to carry out meaningful image processing tasks appeared in the early 1960s, at the same time as development of space programs. With the aim of improving back transmission of images from space exploration on moon surface mapping, work using computer techniques for converting analog to digital signals began at the NASA Jet Propulsion Laboratory in Pasadena in 1964. Computer technology was advanced and pictures transmitted were processed to correct various types of image distortion inherent in the on-board television camera.

In the early 1970s, the invention of computerized axial tomography (CAT) was another important event in the development of digital imaging, in this case for medical diagnosis. From rings of sensors around the object and X-ray source, tomography consists of algorithms that construct an image that represents a slice of the object. Sir Godfrey N. Hounsfield and Prof. Allan M. Cormack shared the 1979 Nobel Prize in Medicine for this invention.

Medical research, government programs for space exploration and espionage with spy satellites clearly helped advance the science of digital imaging. However, the private sector in parallel also made significant contributions in the development of digital cameras. It first became possible to build these cameras with the availability of the CCD image sensor, invented by Willard S. Boyle and George E. Smith at Bell Labs in the early 1970s. In 1981, Sony released the Mavica, first commercial electronic still camera, followed by Canon with the RC-701 camera. Since the mid-1970s, Kodak has invented several solid-state image sensors for professional and home consumer use. In 1986, Kodak scientists invented the world's first CCD mega-pixel sensor, capable of recording 1.4 million pixels. In 1987, they released seven products for recording, storing, manipulating, transmitting, and printing still and video digital images. Since then, a large palette of digital cameras for the consumer-level market has been developed. Current research on digital cameras focuses mainly on sensors and aims at developing sensors with higher resolution, while increasing their sensitivity.

Foundations

Today, the continuing drop in prices of computers and storage, the democratization of digital images (multimedia PCs, digital cameras, cell phones, digital camcorders, etc) and the expansion of networking and communication bandwidth via the Internet or the HDTV (High-definition television) greatly contribute to the production and dissemination of larger and larger volumes of digital images. Digital images are now involved in a large number of leading applications that require the use and development of techniques belonging to several disciplines of computer science. These disciplines are presented below, after a brief introduction to the bases of digital image acquisition and representation.

Image Acquisition

Digital imaging refers to the technical process of digital image acquisition. A 2D digital image may be created directly from a physical scene by a digital camera or similar devices, as illustrated in Fig. 1 with image (b). Alternatively, it may be obtained by a scanner from a document in an analog medium, such as photographs, photographic films or printed papers. Most of these equipments have in common the digital sensors that capture light rays and convert them into electrical signals. At present, the most popular image sensors fall either in the category of charge-coupled devices (CCD) or active-pixel sensors (CMOS for Complementary Metal-Oxide-Semiconductor) for the most recent technology [6].

Unlike human vision, sensors that capture images are not limited to the visual band of the electromagnetic spectrum. Digital images can cover almost the entire spectrum, ranging from gamma to radio waves. Many other specific equipments exist to capture 2D or 3D digital images, mainly for medical, military, astronomy or satellite domains: X-radiographs, MRI scanners (Magnetic Resonance Imaging, see image (c) in Fig. 1), PET scanners (Positron Emission Tomography), SLAR radars (Side-Looking Airborne Radars), radio telescopes, etc.

Finally, a digital image can also be computed from a geometric model or mathematical formula. In this case, it refers to 2D or 3D computer graphics discipline, as illustrated with images (d), (e), and (f) in Fig. 1.

Image Representation

Digital images can be represented either as vector graphics or bitmapped representation (also called raster representation). Vector images are geometrical 2D objects created with drawing software or CAD (computer-aided design) systems. They are represented with geometrical primitives such as points, lines, curves, and shapes or polygons, which are all based upon mathematical equations.

A bitmapped digital image is composed of a set of dots or squares, called pixels (for picture elements), arranged in a matrix of columns and rows. Each pixel has a specific color or shade of gray, and in combination with neighboring pixels it creates the illusion of a continuous tone image. Managing bitmapped images requires the manipulation of several parameters such as color model, dynamic range, and resolution.

Image Processing and Analysis

Because an image can be seen as a two-dimensional signal, image processing is a sub-field of signal processing. A lot of image processing techniques involve treating the image as a two-dimensional signal and applying standard signal processing techniques to it, while other techniques are specific to images. In the digital image community, there is not an unified definition of the tasks associated with image processing. Traditionally for some people, image processing techniques refer to image manipulation or filtering and then take an image as input and provides another image as output. For others, the output can also be a set of characteristics related to the input image, for instance either low-level features obtained by detection or extraction processes (contours, regions, etc), or high-level features obtained by analysis (face recognition, behavior interpretation, etc). According to [4], image processing gathers the three following computerized processing levels:

- Low-level process: This level involves preprocessing operations such as image geometrical transformations (sub-sampling, rotation), image compression, image enhancement (noise reduction, contrast enhancement, image sharpening, etc). Both the inputs and outputs are images.
- Mid-level process: The input is an image processed at previous level and generally the outputs are visual attributes extracted from this image. Mid-level processing on images involves tasks such as automatic extraction of meaningful features (contours, key points, regions, objects). This step is then followed by the visual description of these features to be able to make the distinction between them, and more generally to reduce them to a form suitable for higher-level computer processing.
- High-level process: This ultimate level tries to artificially perform the cognitive function usually associated with human vision. Associated with previous level, it refers to image *analysis* and *interpretation* in literature. From the features extracted at previous level, plus potential external information such as prior knowledge furnished by the application/ domain or training data, the algorithms try to form a decision.

Computer Vision

As the name implies, computer vision is the science and technology that allow computers to see. Computer vision is concerned with the theory for building artificial systems that interpret images, with ultimate goal of reaching performances similar to human vision. This discipline shares many formalisms and techniques with the image processing and analysis one previously described. But in literature, image processing and analysis tends to be interested in the interpretation of 2D images, while computer vision rather focuses on the interpretation of 3D scene from its projection onto one or several images to analyze, e.g., how to reconstruct structure about the 3D scene or other geometrical information (like location of cameras or 3D trajectories of objects) from images. To achieve these tasks, one fundamental aspect of computer vision techniques is geometry and more precisely the manipulation of geometrical models [5], such as the camera model (classically *pinhole model*) and the stereovision model (*epipolar geometry*).

The problems addressed by 3D computer vision are various [3], for example: camera calibration, i.e., the automatic determination of the characteristics of the camera from one image and a 3D pattern; auto-calibration from several images; 3D reconstruction of the scene structure from several images; determination of 3D motion/trajectory of objects in video sequences.

Image Synthesis

While scanners or digital cameras allow building digital images from physical objects, image synthesis focuses on the capability to create novel images representing realistic or artificial scenes. The traditional approaches of *computer graphics* have been to create synthesis images from a given geometric model in 3D by projecting it onto a two-dimensional image. Conversely, *image-based rendering* starts from one or several two-dimensional images in order to directly generate novel two-dimensional images, skipping the manual 3D modeling stage. Applications such as video games, motion picture, audiovisual, architecture, computer simulation, tourism (virtual travels) and e-commerce stand to benefit from these technologies. The most known associated exhibition is the SIGGRAPH annual conference (Special Interest Group on GRAPHics and Interactive Techniques) convened by the ACM SIGGRAPH organization since 1974 [12].

Computer Graphics Computer graphics is concerned with the creation and the manipulation of digital images [10]. The term can refer to 2D computer graphics, where artificial images are diagrams, logos or textures that may be generated by using vector graphics modelers (see example (f) in Fig. 1), mathematical models such as fractals (see example (d) in Fig. 1), or by processing existing images. Most of the time, computer graphics implicitly means 3D computer graphics, that renders images from the three-dimensional representation of geometric data. 3D computer graphics moves on several techniques stemmed from Computer-Aided Design (CAD):

- *3D modeling* is the process of developing a mathematical representation of any three-dimensional object that describes its appearance in terms of shape, colors, and potential textures. 3D models are often created with special softwares called 3D modelers, by an artist, a specialist or engineer or scanned into a computer from real-world objects.
- The object layout and its interactions with other objects must also be described. If the object moves or deforms, *animation* refers to its temporal description. Popular methods include inverse kinematics and motion capture; this last involves the recording of human actors actions, to animate digital character models.
- *Rendering* converts the 3D object into a image by projecting it onto a 2D image with a perspective projection model, while simulating light transport according to illumination models. The basic operations are *transport* (models for light movement) and *scattering* (models for light reflection on the object's surface). Different methods are better suited for either photo-realistic rendering, or real-time rendering. When the goal is photo-realism, the most famous global methods for rendering are *ray tracing* and *radiosity.* These techniques successfully address the problems of occlusion, transparency, shadows and take the interactions between objects into account, as illustrated with the image of Fig. 1e.

Image-Based Rendering Unlike traditional computer graphics approaches previously described in which 3D geometry of the scene is known, the aim of image-based rendering techniques is to render novel views directly from input 2D images, in a realistic way without full 3D model reconstruction [11]. Previous work on image-based rendering reveals a continuum of image-based representations based on the tradeoff between how many input images are needed and how much is known about the scene geometry; such techniques directly refer to geometric and algorithmic models of computer vision. The most popular applications of image-based rendering are *view morphing* and *image mosaicing.* The first one aims at generating intermediate views between two reference images, by interpolating parts of images in correspondence. In the 1990s, they were very popular in the audiovisual industry and motion pictures, while producing aesthetic and spectacular special effects at low cost, such as one face turning into another. The novel views rendered with such techniques do not respect the geometry scene, in the sense that the objects may be distorted in the produced images. Image mosaicing produces panoramic images by registering or stitching multiple regular images that partially share a view of the scene; see example (g) of Fig. 1. The produced image can be projected to a cylindrical or a spherical map for visualization and virtual navigation with the feeling of immersion into the 3D world. The first and most popular application of mosaicing is QuickTimeVR [1].

Image Indexing and Retrieval

The huge volume of digital images now available on the Internet or simply on personal computers makes for images that cannot be easily located. Image indexing techniques aim at developing search engines that manage collections of images and in particular that facilitate their fast and accurate retrieval. But retrieval of images, more generally of multimedia data, is different from retrieval of structured data such as classical databases. It is necessary to exhibit a description of the images, often called *descriptor*, that represents the essence of the image for a given topic and that will be indexed to allow efficient retrieval among the many descriptors associated to the available images.

The most usual descriptors associated to images are textual, making tools of linguistics required to define them. Most of the time, this information is structured in *keywords* that can either be related to a controlled vocabulary (potentially connected to dedicated ontologies or semantic lexicons) or be freely assigned [13]. Keywords can be determined by analyzing the metadata associated with the image, the filename of the image, by parsing text adjacent to the image as Google Image Search does on the Internet, or by using more sophisticated methods such as text extraction from images content. Otherwise, keywords are designed manually by experts of the application domain. A popular new form of free-text keywords on the Internet is *folksonomy* where tags are assigned to images by non-experts or consumers collaboratively. Once the images are described with keywords, evaluating a search query to quickly locate images characterized with given words among a collection of images is done by exploiting an index data structure; inverted files are a popular technique for indexing text data [9].

Text-based indexing has several shortcomings such as ambiguities, subjectivity, language or context-dependence. Indexing the visual content of images is a recent alternative that may help to reduce these drawbacks by giving additional insight into the collections of images. When the number of images makes manual annotation unachievable, it is the only solution. Born in early 1990s, Content-Based Image Indexing (CBIR) is a discipline that exploits techniques of image processing/analysis in addition to databases tools [2]. Indexing an image by its content begins by extracting visual structures that describe the visual content relevantly for the considered application. Such structures are considered as the index of the image, which is digitally represented with one or several multidimensional vectors called *signature* of the image. Here, multidimensional index structures are required to perform retrieval efficiently in terms of processing time and disk access [9].

Key Applications

A classical problem in image processing, image analysis, computer vision and CBIR is the ability to determine whether or not an image contains some specific objects. This task, referred to *object recognition*, can normally be solved robustly and without effort by a human, but is still not satisfactorily solved by computers for the general case of arbitrary objects in arbitrary situations. The existing methods can at best solve it only for specific objects such as print or handwritten characters, human faces or vehicles and in specific situations, typically described in terms of well-defined illumination, background, and pose of the objects relative to the camera. Thus, while object recognition has been an active research area for 40 years [8], it is still challenging for several applications. Nevertheless, some patented approaches for specific object recognition have already proved their efficiency, they are offered as a service by consumer-oriented companies or are well established in several organizations and apply to a large palette of domains. For instance:

- *Surveillance of road traffic:* Automatic recognition of license plates is a mass surveillance method that uses character recognition on images to read plate numbers on vehicles. Many governments, police, and town councils have adopted this technology to automatically detect vehicles in driving offence, to supervise areas such as parking areas, or to control traffic in order to help monitor the movements and flows of vehicles around a road network.

- *Authentication:* Biometrics technologies provide methods for uniquely identifying humans based upon one or more intrinsic physical or behavioral traits. Systems based on visual appearance analysis are based on particular methods of image analysis dedicated to specific images representing faces, retinas, hands, veins or fingerprints. For example, several banks in Japan have adopted palm vein authentication technology on their ATMs (technology developed by Fujitsu in 2003 and called PalmSecure). Among several governments, Israel has also adopted this technology to control the border crossing points between Israel and the Gaza Strip: an ID card, with stored biometrics of fingerprints, facial geometry and hand geometry, is required to go across.
- *E-commerce:* Since few years, several companies provide to their clients the ability to link their products and services directly to specific web pages on the mobile internet by the way of bi-dimensional barcodes called "smart codes." With their mobile phone's camera, consumers can scan the code placed near or on the product, then the code is recognized by image analysis, making it possible the connection to the associated service (product purchasing, timetables consultation, prices comparison, etc).

Cross-references

▶ Annotation-Based Image Retrieval
▶ Feature Extraction for Content-Based Image Retrieval
▶ Icon
▶ Image Content
▶ Image Content Modeling
▶ Image Database
▶ Image Querying
▶ Image Representation
▶ Image Retrieval
▶ Indexing and Similarity Search
▶ Multimedia Data
▶ Multimedia Databases
▶ Object Detection and Recognition
▶ Object Recognition
▶ Video
▶ Video Representation
▶ Visual Content Analysis
▶ Visual Representation

Recommended Reading

1. Chen S.E. QuickTimeVR: an image-based approach to virtual environment navigation. In Proc. Int. Conf. on Computer Graphics and Interactive Techniques, 1995, pp. 29–38.
2. Datta R., Joshi D., Li J., and Wang J. Z. Image retrieval: Ideas, influences, and trends of the new age. ACM Comput. Surv. 40(2), 2008, paper 5.
3. Forsyth D. and Ponce J. Computer Vision - A modern approach. Prentice Hall, 2002.
4. Gonzalez R. and Woods R. Digital Image Processing. Prentice Hall, 3rd edition, 2008.
5. Hartley R.I. and Zisserman A. Multiple view Geometry in Computer Vision. Cambridge University Press, 2nd edition, 2004.
6. Litwiller D. CCD vs. CMOS: Facts and Fiction. Photonics Spectra, January 2001.
7. McFarlane M.D. Digital pictures fifty years ago. Proc. IEEE, 60 (7):768–770, July 1972.
8. Ponce J., Hebert M., Schmid C., and Zisserman A. Toward Category-level Object Recognition, LNCS, vol. 4170. Springer, 2007.
9. Samet H. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann 2006.
10. Shirley P., Ashikhmin M., Gleicher M., Marschner S., Reinhard E., Sung K., Thompson W., and Willemsen P. Fundamentals of Computer Graphics. A.K. Peters, Ltd., 2nd edition, July 2005.
11. Shum H. Y., Chan S. C., and Kang S. B. Image-Based Rendering. Springer, 2006.
12. ACM SIGGRAPH. http://www.siggraph.org/.
13. Yu L. Introduction to the Semantic Web and Semantic Web Services. Chapman & Hall/CRC, 2007.

Image Classification

▶ Automatic Image Annotation

Image Compression

▶ Image Representation

Image Content

▶ Image Representation

Image Content Modeling

HARALD KOSCH, MARIO DÖLLER
University of Passau, Passau, Germany

Synonyms

Image data model; Image meta-data; Conceptual image data model

Definition

Image Content Modeling deals with the issue of representing the content of image data; that is, designing the high- and low-level abstraction model of the raw image objects and their correlations to facilitate various operations. These operations may include media object selection, insertion, editing, indexing, browsing, querying, retrieval, and exchange. The image content model relies, therefore, on the extraction of feature vectors and their respective representations obtained during the annotation process. Several standards for representing the content of an image are known. The most prominent ones are MPEG-7 including low- and high-level abstractions or the EXIF data description vocabulary for the description of very specific technical attributes of an image.

Historical Background

Many models for describing the content of an image have been created in the past. Most of them rely on standards. Many standards are actually in use, mainly due to *different national and organizational interests* of service providers, standardization bodies, and usage groups. Whereas each standard works well in its dedicated application domain, problems arise in frequently occurring cross-domain working environments. The following is a non-exhaustive list of important standards:

- The *Exchangeable Image File (EXIF)* (http://www.exif.org/) format is an international specification that allows imaging companies to encode meta data information into the headers or application segments of a JPEG file. This meta data information includes shutter speed, aperture, date and time of the captured image. Current digital cameras store images using EXIF compressed files. It is a widespread standard, with a simple attribute/value specification. EXIF was last revised in April 2002.
- The *DIG35* Initiative Group of the I3A (International Imaging Industry Association) has also defined a meta data standard for digital images (http://www.i3a.org/i_dig35.html). The *DIG35 Specification* includes technical meta data on media, simple semantic meta data on persons, locations, events and intellectual property and rights management related meta data. The focus of DIG35 is on retrieval, categorizing and browsing of large image archives. The latest version stems from March 2000.
- The *SMPTE* (http://www.smpte-ra.org/mdd/) is a standardization organization that has created the Metadata Dictionary (MDD). Most meta data consists of media-specific attributes, such as timing information as well as some basic semantic descriptions (interpretation, administration and relationships among descriptions). The MDD definition was reviewed in 2004. The SMPTE Material eXchange Format (MXF) allows users to take advantage of non-real-time transfers, and to package together essence and metadata for effective interchange between servers and between businesses.
- The *P/Meta EBU P/Meta Project* (http://www.ebu.ch/metadata/pmeta/v0100/html/start_frame.html) has created the P_META Scheme, a set of definitions that provides a semantic framework for information which is typically exchanged along with audio-visual material. It includes the identification of concepts referenced by other elements. The P_META Scheme has been created for the use in a business-to-business scenario where the participating organizations may retain their internal data structures, workflows, and concepts. Version 1.0 appeared in 2002.
- The MPEG-7 (http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm) is the MPEG standard for description and search of audio and visual content and is the first standard from MPEG which considers multimedia content models. From the point-of-view of the expressiveness of the proposed metadata descriptions it is the richest set, as it enables the user to describe structural, as well as semantic content of an image [3]. It supplies likewise means for the description of creation, usage and management of images. MPEG-7 version 1 was released as international standard in 2002. An improved version was published in 2006.

MPEG-7 visual description tools cover color, texture, shape and face recognition descriptors for images. They mainly use a histogram-based approach of representation; that is, they compute a vector of elements each representing the number of pixels (regions) in a given image, which have similar characteristics. For instance, the color structure descriptor computes a histogram obtained by accounting all colors in a structuring element which slides over the images and takes therefore the spatial layout better into consideration, or the color layout descriptor which uses a discrete cosine transformation to represent more compactly the histogram. MPEG-7 high-level descriptions mainly represent the structure and semantics of an image. The media decomposition tools allow the recursive sub-division of an image into regions of interest. A decomposition tree describes the image source and the spatial structure of the image regions and can be used to create a table of content. Each region can be associated with its shape information and low-level visual descriptions, then information on creation, rights and usage, simple text annotation (what, where, when, why, how), and semantic descriptions. Semantic descriptions represent a narrative world as a set of semantic entities and semantic relations among semantic entities, and segments. Possible semantic entities are objects, agent objects, events, places, and time. MPEG-7 has been employed in many retrieval and database systems. Westermann et al. gives an overview of MPEG-7 based database solutions [11] and provides a persistent DOM (Document Object Model) model that can be used for MPEG-7 systems solutions. A full-fledged database system, MPEG-7 MMDB is described in [1].

The general goal of achieving *image content data interoperability* is affirmed by many researchers. It is expressed by panels and tutorials held at various international conferences, such as ACM Multimedia 2002 and 2004 (See, http://mm02.eurecom.fr/panel.html and http://www.mm2004.org/acm_mm04_call4tutorials.htm) or by input contributions to standardization (e.g., recently proposed to W3C as Semantic Web Image Annotation Interoperability (http://www.w3.org/2001/sw/BestPractices/MM/interop.html)). It has been clearly recognized that missing interoperability is still a major burden for an effective use of image content data in industrial media engineering.

Foundations

Keywords are by far the predominant features used to describe the content of image data. An indexer using keywords or a textual abstract describes the content of the image [6]. Another method, *content-based indexing*, refers to the actual content of the image data. Intensive research has focused on content-based indexing in recent years, with the goal of indexing the image data using certain features derived directly from the data [10]. These features are also referred to as *low-level features*. Examples of low-level visual features for images are color, shape, texture, spatial information, and symbolic strings. The extraction of such features may be done by an automatic analysis of the image data. Indexing *high-level features* (e.g., objects, events, etc.) of image data is an active research area [5]. Different detection mechanisms have been proposed for segmenting the image materials into regions of interest and for attributing semantic meaningful annotation to each region and (their semantic) relationships. In addition to this, and related to the previous issues, ongoing research concentrates on image classification. For example, is this image showing a sport event? And more specifically, is this sport image about basketball? The recognition of an image as basketball facilitates the semantic annotation process, as one knows what kind of objects and persons might appear.

Image Content Modeling refers now to the design of a high- and low-level abstraction model of the raw image objects and their correlations to facilitate various operations, such as media object selection, insertion, editing, indexing, browsing, querying, retrieval, and exchange. Several abstract models have been proposed in the literature [3]. Most of them deal with special low- and high-level features. At least two models introduce a general framework to cover the whole spectrum of available features, the DISIMA model [7] and the indexing pyramid [2]. The later one served as conceptual model for the semantic part of the MPEG-7 standard.

The DISIMA model relies on two main concepts, namely the image and salient objects (logical and physical) and operators to manipulate them [7]. Principally, the image type defines the content of an image as a set of physical salient objects that are regions of the image. The semantic meaning of a physical salient object is represented by a logical salient object. Oria et al. define several properties of physical salient objects, such as color, texture, and shape. The logical salient object gives semantics to a physical salient object; proposed examples are objects, persons and so on. Predicates for comparing objects (e.g., different similarity operators, spatial operator) and a special contain operator which computes if an object is contained in an image, are proposed. As an image can be composed of more than one physical object, logical objects can be found several times. The model can be queried by Visual MOQL, an image query language based on OQL, proposed in a previous paper of [7].

Jörgensen et al. [2] introduced a representation model based on the indexing pyramid for classifying levels of indexing. The pyramid, as shown in Fig. 1, distinguishes between *syntactic* (first four levels) and *semantic* levels (next six levels). The syntactic levels hold attributes that describe the way in which the content is organized, but not their meanings. This corresponds to the definition of low-level features. In images, level 1 (type technique) could be "color image." Global distribution holds attributes that are global to the image (e.g., color histogram), whereas local structure deals with local components (e.g., lines and circles), and global composition relates to the way in which those local components are arranged in the image (e.g., symmetry). The semantic levels deal with the meaning of the elements. This corresponds to the definition of high-level features. With each level, more knowledge is required to represent the objects. They can be described at three levels:

1. Generic: every day objects (e.g., person)
2. Specific: individually named objects (e.g., Roger Moore)
3. Abstract: representing the highest semantic abstraction (e.g., power)

In a similar way, a scene (composition of elements) can be described at these three levels.

Note that the authors of the indexing pyramid also propose a retrieval system that allows a search specific to the levels of the pyramid. For instance, if a user enters "soccer" for image search at the syntactical level, one retrieves images with a description containing the keyword soccer (which does not yet mean that a soccer game is shown in the image). This is the same mechanism as the retrieval engine that Google uses (http://images.google.com). If a user enters soccer at the semantic level, only those images are retained in which the event, "soccer," took place. An online demo is available at http://www.ee.columbia.edu/~ana/mpeg-7/.

Image Content Modeling is dominated by the *use of standards*, as already mentioned in the historical considerations before. Two types of standards have to be considered. First those which intend to specify a standard way of describing the content of image data mainly for information search and exchange. Second those which aim to be used for image storage, manipulation and query. In the first category, MPEG-7 is an important representative [3]. MPEG-7 is introduced in the historical background, and is detailed in the multimedia/image metadata entries. An example MPEG-7 document is given in Fig. 2. In the second category, SQL/MM, recently been integrated into Oracle Multimedia for instance, is representative [8]. SQL/MM will be detailed in the next paragraphs.

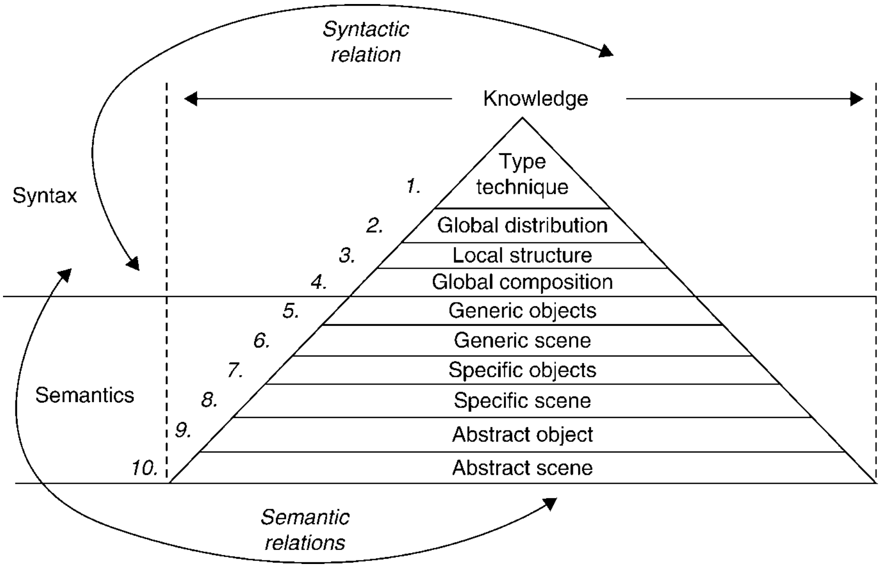


Image Content Modeling. Figure 1. Indexing Pyramid as base for MPEG-7.

SQL/MM as Conceptual Image Data Model for Databases ISO/IEC developed the SQL/MM Multimedia extensions to SQL to deal with image data in a database management system [8]. SQL/MM has been recently introduced to several commercial systems. It introduces new object data types, in the center are the *SI_StillImageType* and related feature vector representations. Principally, one color histogram (*SI_ColorHistogram*), and two non histogram color features are proposed, an average color (*SI_AverageColor*) and an array of dominant colors (*SI_PositionalColor*). The texture feature (*SI_Texture*) contains values that represent the image texture characteristics. The standard defines in addition a composite feature (*SI_FeatureList*) containing up to four different basic features and associated feature weights. A weight value specifies the importance attributed to a particular feature during image matching. The weight value varies from 0.0 to 1.0. A feature weight value of 0.0 indicates that the feature is not considered for image matching.

```
<Mpeg7>
  <Description xsi:type=”ContentEntityType”>
     <MultimediaContent xsi:type=”ImageType”>
        <Image>
          <Semantic>
             <Label>
                <Name> Anne takes a photo of the Leaning Tower </Name>
             </Label>
             <SemanticBase xsi:type=”EventType” id=”TakingPhoto”>
                <Label>
                  <Name> Description of the event of taking a photo </Name>
                </Label>
                <Relation type=”urn:mpeg:mpeg7:cs:SemanticRelationCS:2001:agent”
                     target=”#Arme”/>
                <Relation type=”urn:mpeg:mpeg7:cs:Semant.icRelationCS:2001:source”
                     target=”#Tower”/>
                <SemanticPlace>
                  <Label>
                     <Name>Pisa</Name>
                  </Label>
                </SemanticPlace>
                <SemanticTime>
                   <Label>
                     <Name>September 3, 2007</Name>
                  </Label>
                </SemanticTime>
             </SemanticBase>
             <SemanticBase xsi:type=”objectType” id=”Tower”>
                <Label>
                  <Name> Leaning Tower </Name>
                </Label>
             </SemanticBase>
             <SemanticBase xsi:type=”AgentobjectType” id=”Arme”>
                <Label>
                  <Name> Anne </Name>
                </Label>
                   <Agent xsi:type=”PersonType”>
                   <Name>
                     <GivenName> Anne </GivenName>
                  </Name>
                </Agent>
             </SemanticBase>
          </Semantic>
       </Image>
     </MultimediaContent>
  </Description>
</Mpeg7>
```

Image Content Modeling. Figure 2. MPEG-7 Example Document.

It can be clearly seen that the SQL/MM data model covers only parts of the syntactical levels of the indexing pyramid proposed in [2]. For instance no means for describing the decomposition of an image into region of interests is given.

For instance, a photo book can be described by the following table definition:

```
CREATE TABLE PHOTOBOOK (
PHOTO_ID NUMBER(6), // id for identification
PHOTO SI_StillImage, // the image itself
AVERAGE_COLOR SI_AverageColor, // here
comes the list of
COLOR_HISTOGRAM SI_ColorHistogram, // fea-
tures for querying
FEATURE_LIST SI_FeatureList,
POSITIONAL_COLOR SI_PositionalColor,
TEXTURE SI_Texture,
COMMIT;
```

Key Applications

Key applications for image content modeling are image databases and retrieval systems. SQL/MM is the standard for the use in commercial database management systems. Its simple conceptual model does not allow the use of complex low-level representations, nor high-level descriptions. Instead, a recent research issue is on image meta data databases, which for instances expresses in the search for a good mapping of an image content model into database schemes (see e.g., MPEG-7 databases in [1,11]).

Cross-references

▶ Image Database

▶ Multimedia Metadata

Recommended Reading

1. Döller M. and Kosch H. The MPEG-7 Multimedia database system (MPEG-7 MMDB). J. Syst. Software, 81(9):1559–1580, 2008.
2. Jörgensen C., Jaimes A., Benitez A.B., and Chang S.-F. A conceptual framework and empirical research for classifying visual descriptors. J. Am. Soc. Inf. Sci. Technol., 52(11):938–947, 2001.
3. Kosch H. Distributed Multimedia Database Technologies supported by MPEG-7 and MPEG-21. CRC Press, Boca Raton, FL, 2004.
4. Kosch H., Böszörményi L., Döller M., Kofler A., Schojer P., and Libsie M. The life cycle of multimedia meta-data. IEEE Multimedia, 12(1):80–86, 2005.
5. Liu Y., Zhang D., Lu G., and Ma W.Y. A survey of content-based image retrieval with high-level semantics. Pattern Recognition, 40(1):262–282, 2007.
6. Lu G. Multimedia Database Management Systems. Artech House, London, UK, 1999.
7. Oria V., Özsu M.T., and Iglinski P.J. Foundation of the DISIMA image query languages. Multimedia Tools Appl. J., 23:185–201, 2004.
8. Stolze K. Still image extensions in database systems – A product overview. Datenbank-Spektrum, 2:40–47, 2002.
9. Tseng B.L., Lin C.-Y., and Smith J.R. Using MPEG-7 and MPEG-21 for personalizing video. IEEE Multimedia, 11(1):42–52, 2004.
10. Veltkamp R.C. and Mirela Tanase. Content-Based Image Retrieval Systems: A Survey. Technical Report, Utrecht University, The Netherlands, 2000.
11. Westermann U. and Klas W. An analysis of XML database solutions for the management of MPEG-7 media descriptions. ACM Computing Surv., 35(4):331–373, 2003.

Image Data Model

▶ Image Content Modeling

Image Database

Mario Döller[1], Harald Kosch[1], Paul Maier[2]

[1]University of Passau, Passau, Germany

[2]Technical University of Munich, Munich, Germany

Synonyms

Image retrieval; Image retrieval system; Content-based image retrieval (CBIR)

Definition

Given a collection of images, a full-fledged image database provides means and technologies that support an efficient and rich modeling, storing, indexing, retrieval, and manipulation of images and its meta-data. The modeling of images can range depending on the used meta-data format (e.g., MPEG-7) from simple technical annotations such as file size, creator, etc. to more sophisticated annotations such as low level features (e.g., color) or even high level features (e.g., objects, events, etc.). The storing component is responsible for mapping the used meta-data format to an adequate database schema. Indexing facilities should support efficient retrieval and need to provide means (depending on the used meta-data) for indexing text, multidimensional feature vectors and high level representations. The retrieval and query specification should support some or all of the following concepts: query by example, query by sketch, query by textual information (e.g., creator), query by category browsing, query by concept. Finally, manipulation of images should allow operations such as rotating, shrinking, format conversion, thumbnail creation, feature extraction, etc.

Historical Background

Activities in storing and retrieving images within databases can be traced back to the late 1970s, where first conference (e.g., Data Base Techniques for Pictorial Applications, 1979) contributions introduced the use of relational databases for images [2]. In general, these early works focused on the annotation and retrieval of images by textual information. For this purpose, the images were described by keywords or textual descriptions and common relational database technologies and their text-based retrieval approaches were used for searching within the pool of annotated images. A substantial survey for text-based image retrieval can be found in [11]. The main drawbacks of these early-approaches concern the annotation of images by text. In general, manual annotation of images and its content is very time consuming and for large image sets infeasible. Furthermore, the assignment of descriptions or keywords to images is not based on the use of a controlled and universally valid vocabulary or classification scheme, and is highly affected by cultural, subjective and domain specific influences. First discussions on this topic were raised by Shatford [9] which indirectly resulted in various thesaurus catalogues (e.g., thesaurus for graphical material, Art and Architecture Thesaurus (AAT), etc.).

Based on the identified drawbacks of text-based image retrieval, researches tended to move in the late 1990s to investigate the retrieval of images by content. It is often referred in the literature as content-based image retrieval (CBIR) and concentrated on technologies for the extraction, indexing, comparison and query of images by their low level features. These low level features include representations for color, texture and shape which can be extracted automatically.

In general, a typical CBIR system covers the following main steps. First, the low level features of all searchable images are extracted and stored as high-dimensional feature vectors in a database. Then, in order to improve retrieval efficiency, the feature vectors are indexed by specific access methods which are tuned for that kind of data (see [3] for a survey of existing access methods). Finally, during the query process, an example image (represented by its features) is provided by the client. Based on this, the similarity/distance in relation to the stored features is calculated. The size of the result set depends on the chosen query type (range query or nearest neighbor) and contains the most similar (in terms of the low level features) images according to the given one.

Examples of early successful systems and applications focusing on CBIR were, for instance, the IBM QBIC system (which stands for Query by Image Context), Blobworld or CalPhotos. The CalPhotos image retrieval system had its roots in a 1993–1994 research project called Chabot, at the University of Berkeley. R. C. Veltkamp and M.Tanase compared (updated in 2004) 43 available products [12]. Smeulders et al. surveyed the early years of CBIR systems in [10] with 200 references.

Foundations

As already outlined in the historical background section, the search within image collections has a long tradition in the field of database management systems. In the following, the main components of an image database are introduced in order to support the storage, indexing, and retrieval of images.

Image Database Schema

First approaches transferred meta-data of raw media data into database relations in an ad-hoc way. These simple models only support certain types of queries and operations and were mostly limited to keyword retrieval. The nature of multimedia data, consisting of alphanumeric, graphical, image, video, and audio objects, differs in many ways to simple alphanumeric data that relational database management system are able to handle. One of the first models proposed for visual information that considers semantic queries, was called the Visual Information Management System (VIMSYS) [4] from Virage. The creators recognized within their model that image and video information is preferred to be retrieved by content rather than by keywords or additional textual descriptions. During the past years, research concentrated on the development of data models for images (e.g., [1]) and content-based retrieval. In most cases, these systems concentrate on the retrieval based on low-level features. Recently, data models for image information that support high-level features have been introduced. The data models of these systems are often specialized to one type of genre, e.g., retrieval on the basis of soccer games (e.g., [5]).

Index Structures

Besides the necessity of developing semantically rich image data models and database schemas, an efficient indexing and search of images is essential. During the last decade, image based retrieval concentrated on the use of low-level features and their feature vector representations.

In order to improve search efficiency, different access methods have been established for indexing multidimensional feature vectors, for instance (to mention just a few) LPC-File, X-tree, Pyramid Technique, R-tree and its variants, SR-tree, M-tree, etc. An excellent survey and overview paper of index structures is given by e.g., Gaede et al. [3]. An evaluation of different quantization techniques, such as A-tree versus IQ-tree is given by Garcia-Arellano (University of Toronto) in his master thesis. A complete generalized framework, called GiST, has been established by J. Hellerstein and his group at the University of California, Berkeley. It consists of a set of pre-implemented index structures and generalized access interfaces. Support is given by sophisticated graphical tools providing access statistics. The framework has been used for various image retrieval applications (e.g., Blobworld) and the development of new index structures. In addition, the framework has been integrated within research projects in the Informix database and Oracle database.

One problem in processing feature vectors is their high dimensionality. The quality of results of a similarity search depends on a high degree on the underlying data set (feature vectors). For instance, the more colors a color histogram represents the merrier the similarity search will work. This correlates with a higher dimensionality of the resulting feature vector. This is also the case, when multiple features are combined, such as for images the color, the shape and the texture. Unfortunately, query performance of access methods decreases, if dimensionality of the underlying data set becomes high. This phenomenon is known as *curse of dimensionality* [3]. One possibility of avoiding this problem is the development of specific index structures that are specialized for high-dimensional data (e.g., see [3] for a survey). Another possibility is the reduction of the vector dimension. The main goal of reducing the dimension is to find a low-dimensional representation of the vector that preserves (most of) the information of the original data. The reduction can be realized by two different approaches: *Feature selection* and *Feature extraction*. *Feature selection* stands for choosing a subset of all the features that represent most of the desired information. *Feature extraction* describes the creation of new features by combining the existing ones. Wu et al. [13] introduced a dimensionality reduction for image retrieval called *weighted multi-dimensional scaling*. Another commonly used technology is the *singular value decomposition* (SVD).

Query Optimization

A general architecture of a query optimizer has commonly the following form [1]: A query is forwarded to a *query parser* which checks the syntactical correctness and transforms the query into an internal representation (mostly an algebraic expression). In the following step, the *query optimizer* evaluates the most effective algebraic expressions which represents the given query and chooses the cheapest one. The *code generator* transforms the resulting query plan into calls to the *query processor*, which does the actual query execution. The main task of a query optimizer is to examine all possible alternatives to a given query so that the best alternative can be chosen and the query cost is minimized.

In image databases, queries often contain similarity operations, such as Range or Nearest Neighbor (NN)-operation for low-level features (e.g., color histogram), especially if the system supports content-based retrieval. This can be associated with a similarity-based selection operation in an image database management system. In general, a query optimizer should use three approaches: *selectivity*, *cost model*, or *operator ordering*. Most approaches concentrate on the cost-model and the selectivity, based on the fact that modern database systems, such as Oracle, only provide means for their enhancement.

The cost of a selection operation is in general composed of two different cost factors. The *selectivity* of an operation defines the amount of tuples returned by the selection operation. Whereas, the *cost* of an operation is counted as the amount of data pages which has to be accessed for fulfilling the given task and as the number of necessary CPU cycles. In the image environment, the cost (selectivity, and amount of data pages) of Range- and k-NN-queries in high-dimensional spaces is important. An efficient processing of such kind of queries is guaranteed by a multidimensional index. Therefore, the calculation of the amount of accessed pages (further simply called cost) needs an evaluation of the used index structure. This evaluation can be simple in one case, for instance the selectivity of a k-NN query is determined through k, or difficult, if the selectivity for a Range query is considered which needs an approximation. The cost of a Range query can be easily calculated, based on the known radius which results in an intersection operation of the hypersphere and the minimum bounding rectangles (MBRs) of the index structure. The cost for a k-NN query depends largely on the density around the query location.

In the literature, several cost models exist that concentrate on calculating the amount of page access for Range- and k-NN-queries (e.g., [6]). Selectivity estimation for Range queries is often limited to either the assumption of uniformity and independence of data sets or to the 2- or 3- dimensional data spaces in geographic information systems. The uniformity assumption does not hold for real data sets. In [6], the authors present an efficient cost model for predicting the performance of the k-NN-query independently of the used index tree. The model is accurate for low- and mid-dimensional data with non-uniform distributions. For this purpose, the authors introduce two new concepts: the regional average volume and the density function. The regional average volume is used for calculating the average radius of the hyper-sphere which contains the k-NNs. As indicated before, the average radius is not adequate for highly skewed non-uniform distributions. To overcome this limitation, the authors introduced a density function that estimates the regional radius depending on the location in the data space.

Database Extensions to Support Images

In the last decades, traditional (Object-) Relational Database Management Systems ((O)RDBMS) have been very effective and efficient in managing alphanumerical data. They basically offer services such as indexing, query optimization, buffer management, recovery or concurrency control, which are well investigated and optimized. Unfortunately, common ORDBMS and their techniques have several drawbacks in handling image data. The most important drawback concerns the concept of matching. In traditional databases, the concept of matching relies on a filtering operation which decides for every tuple whether it fits the requirements or not. Basically, the main retrieval paradigm in image data repositories is similarity search. Therefore, extensions to ORDBMS mainly concern the implementation of this search type and for ranking the results, as well as for supporting index structures and query optimization. In the following subparts, various topics for image extensions are addressed.

Query Extension for Multimedia: SQL/MM Traditional query languages, such as SQL, do not provide means for handling multimedia types and their appropriate operations to satisfy the requirements in retrieving multimedia data. Therefore, several query languages (e.g., MOQL, MMDOC-QL, etc.) have been introduced by researchers in order to overcome these limitations. One standard which has been developed explicitly for image data is SQL/MM.

The SQL/MM [8] standard (MM for MultiMedia) is an extension of SQL for supporting any kind of multimedia data. SQL/MM is a multi-part standard and was developed by an ISO subcommittee, namely JTC1/SC32. The various parts of SQL/MM are independent from one another; e.g., Part I (framework) contains definitions of common concepts that are used through out the other parts or *Still Image* (Part V) is all about image data. The SQL/MM Still Image part provides structured user defined types that allow the storage of images within databases and specifies methods for modifying them in various ways and retrieving them efficiently. Images in SQL/MM are represented by the *SI_StillImage* structured data type. The *SI_StillImage* type can store images of several formats depending on what the underlying system supports. Furthermore, it extracts information about each image, such as format, height and width in pixels, color space, etc. In addition, the type provides several methods e.g., for scaling, for rotating or for creating thumbnails of the original image. Moreover, there exist various additional data types for describing several features of images. For instance, the *SI_AverageColor* type represents the average color of a given image. The *SI_ColorHistogram* type provides information about the distribution of colors within the image. The *SI_PositionalColor* type represents the location of specific colors in an image. The SQL/MM Still Image part provides CBIR functionality by combining these types and methods with accurate index structures. For instance, the *SI_PositionalColor* type supports queries like “Give me all images with a color representation of red or orange above dark blue.”

This query would lead to images that in most cases show sunsets at sea.

Multimedia Extensions in Products

Oracle Multimedia The extensibility service provided by Oracle is called *Data Cartridge*. Oracle databases are built as a modular architecture which provides several extensible services. The common way for using Oracle's Extensibility Services is to implement a Data Cartridge that extends the extensibility interface.

Currently, several Data Cartridges have been developed to support multimedia data. Oracle itself provides among others, Cartridges for spatial data and for multimedia data such as image, audio and video. Cartridges of other vendors are for instance, the *Viisage Cartridge* from Virage which provides face recognition on behalf of images. This company provides several face recognition tools for controlling access to restricted areas e.g., Berlin Airport or identifying criminals in a crowd of people.

IBM DB2 The IBM DB2 database provides *Extenders* for enhancing their database management system to meet new requirements, e.g., the storage and retrieval of multimedia data. DB2 Extenders generally specify UDT's (user-defined types) for extending the core database types. At present, there exist a large number of available extenders from different vendors. For instance, IBM itself provides the *AIV* extender for audio, image and video data. Extenders of other vendors in this area are for instance, the *Spatial Data Extender* from Environmental Systems Research Institute (ESRI) for GIS and geo-spatial data or the *SpatialWare Extender* from MapInfo for spatial data. SQLSummit provides an accurate list of Extenders.

IBM Informix The IBM Informix database management system deploys the feature of *DataBlade Modules* to extend their functionality for new requirements. This DBMS provides three different tools for developing DataBlade modules, namely *BladeSmith* for creating DataBlade modules, *BladePack* for packaging them, and *BladeManager* for making them available in the database. Up to now, several DataBlade modules exist such as *Spatial DataBlade module* for managing spatial data inside the database, *Excalibur Image Data-Blade module* for storing and querying images and special vendor products, like DataBlades for MPEG-1.

Image Retrieval Systems from Scratch In contrast to database extensions such as Oracle's Multimedia for managing multimedia data, several image retrieval systems have been developed from scratch (e.g., DISIMA which is an image database management system developed at the University of Alberta). R. C. Veltkamp and M. Tanase compared (updated in 2004) 43 available products [5]. A survey of CBIR systems supporting high level semantics can be found in [7]. In the following, two currently active systems are introduced:

- DISIMA is a pure image database that considers images as a set of salient objects (regions of interest). Salient objects are classified according to a user defined type hierarchy. Other currently active systems are for instance:
- Behold is a large scale image retrieval system and has been developed at the Imperial College in London. It provides means for text-based search and NN-search based on a parameterized similarity metric.

Key Applications

Surveillance Systems

The detection of criminals, wanted persons and known terrorists on airports or border stations bases on face detection algorithms and large scale image (showing persons and/or faces) databases that support the retrieval among millions of data entries.

Medical Diagnosis Supporting Systems

Recent research concentrates on applying image database in the medical domain. Every hospital has a large set of x-rays showing a diversity of diseases. These image collections are used for supporting doctors in completing their diagnosis. For instance, in case of a cancer patient, the doctor can query the image database for similar x-rays based on the patient's.

Web Search

Finding meaningful and high quality images on the web is a goal which appeals to many different people, private and professional alike. The amount of images available on the web is vast, so efficient retrieval mechanisms like CBIR are essential. Research efforts in this direction are for example the before mentioned system Behold (see section, Image Retrieval Systems from scratch).

Cross-references

▶ Image Content Modeling
▶ Nearest Neighbor Query

Recommended Reading

1. Besufekad S.A. Modélisation et traitement de requêtes images complexes. PhD Thesis, L'Institut National des Sciences Appliquées de Lyon, 2003.
2. Chang N.S. and Fu K.S. Query by pictorial example. IEEE Trans. Softw. Eng., 6(6):519–524, 1980.
3. Gaede V. and Günther O. Multidimensional access methods. ACM Comput. Surv., 30(2):170–231, 1998.
4. Gupta A., Weymouth T., and Jain R. Semantic queries with picture: the VIMSYS model. In Proc. 17th Int. Conf. on Very Large Data Bases, 1991, pp. 69–79.
5. Kosch H., Böszörmenyi L., Bachlechner A., Hanin C., Hofbauer C., Lang M., Riedler C., and Tusch R. SMOOTH - a distributed multimedia database system. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 713–714.
6. Lee J.-H., Cha G.-H., and Chung C.-W. A model for k-nearest neighbor query processing cost in multidimensional data spaces. Inf. Process. Lett., 69(2):69–76, 1999.
7. Liu Y., Zhang D., Lu G., and Ma W.Y. A survey of content-based image retrieval with high-level semantics. Pattern Recognit., 40(1):262–282, 2007.
8. Melton J. and Eisenberg A. SQL multimedia application packages (SQL/MM). ACM SIGMOD Rec., 30(4):97–102, 2001.
9. Shatford S. Analyzing the subject of a picture: a theoretical approach. Cataloging and Classification Quarterly, 6(3):39–62, 1986.
10. Smeulders A.W.M., Worring M., Santini S., Gupta A., and Jain R. Content-based image retrieval at the end of the early years. IEEE Trans. Pattern Anal. Mach. Intell., 22(12):1349–1380, 2000.
11. Tamura H. and Yokoya N. Image database systems: a survey. Pattern Recognit., 17(1):29–43, 1984.
12. Veltkamp R.C. and Tanase M. Content-Based Image Retrieval Systems: a Survey. Technical Report, Utrecht University, The Netherlands, 2000.
13. Wu P., Manjunath B.S., and Shin H.D. Dimensionality reduction for image retrieval. In Proc. Int. Conf. Image Processing, 2000, pp. 726–729.

Image Distance

▶ Image Retrieval

Image Indexing

▶ Feature Extraction for Content-Based Image Retrieval
▶ Visual Content Analysis

Image Management for Biological Data

Arnab Bhattacharya[1], Vebjorn Ljosa[2]
[1]Indian Institute of Technology, Kanpur, India
[2]Broad Institute of MIT and Harvard, Cambridge, MA, USA

Synonyms

Image management for life sciences; Databases for biomedical images

Definition

Image management for biological data refers to the organization of biological images and their associated metadata and annotations in a computer in such a way that they can be searched and shared.

Historical Background

The need to manage digital micrographs stored in a computer arose in the early 1990s, when digital cameras started to replace film cameras. Rapid improvement in microscopy technology coupled with steady decline in prices led to an unprecedented increase in the collection of digital biomedical images, and the need for systems to organize, search and share the images became clear. At the same time, the successes of large scale acquisition and analysis of genomic and gene expression data have prompted an increased interest in large-scale image analysis and the use of statistical methods to find patterns in large image sets. The ability of images to capture spatial and temporal relationships not immediately available from other data sources make them vital in understanding the underlying processes in biology that produce them. Database systems for these images promise important benefits, ranging from ease of organization and maintenance to the ability to share, search, explore, summarize, and analyze the data.

Foundations

Content-Based Similarity Search

Similarity search by semantic content may be performed at the level of whole images (content-based image retrieval, CBIR) or regions (region-based image retrieval, RBIR). Datta et al. provide an excellent survey [3]. Some important image retrieval systems are SIMPLIcity, WALRUS, Virage, QBIC, NeTra, Photobook, VisualSEEk, and Keyblock. The two important components of such image comparison systems are the image features and the distance metrics. Examples of general image features are the MPEG-7 standard features, color histograms, texture, wavelets, and shape descriptors. In addition, there are domain-specific feature sets, e.g., for cultured cells or for whole organisms of the *Caenorhabditis elegans* nematode. The most commonly used distance functions are the L_2 (Euclidean) and L_1 (Manhattan) distances. The earth mover's distance (EMD) [6] has also been shown to be useful due to its ability to capture the spatial relationships among the objects and features inside the images. Relevance feedback methods [15] can improve the feature selection and the distance function by capturing the human notion of image similarity.

In biomedical images, particularly in fluorescence micrographs, large portions of the images consist of background that contains little or no information. Therefore, for RBIR purposes, it is imperative to distinguish the foreground image patterns from the irrelevant background. Segmentation and tiling are common techniques for this. Scoring-based mechanisms that model the background content have also been proposed.

Summarization

To explore an image dataset, it is useful to discover latent concepts, found in the literature under different names such as "visual vocabulary," "eigenfaces," "blobs," "visterms," and "visual keywords." Such key patterns can be thought of as analogous to keywords extracted from text, and have been useful for object detection and retrieval, as well as image classification and captioning.

Key Applications

Electronic Notebook

Computer-controlled microscopes and digital cameras have made it possible to acquire images faster than before, but the image collections are in many ways less organized than in the days of photographic films. It is common for images to be spread across personal computers, portable hard drives, and recordable compact disks. Metadata – the data about data, essentially, the description of what was imaged, how, when, and by whom – are kept in handwritten notebooks or in spreadsheets. Besides being cumbersome, this method of organization is also error prone. Images must be converted from a format specific to the vendor of the microscope to a format that is palatable to other programs. Further, contrast enhancement and other image processing is often necessary to visually inspect the images. Through these transformations, the connection between image and metadata is easily lost, and images can be lost accidentally in the process of file format conversion and image processing.

Electronic-notebook–style image databases store the original image files on a server. Derived files for visualization and image processing can be extracted, but the originals are kept pristine. Metadata are also stored on the server, persistently linked to the respective images. Queries to the electronic notebook are simple, and mostly on metadata only. Mainly, users want to find an experiment or an image based on keywords, dates, and other metadata. In particular, they already know what they are looking for, and do not expect to search by image content. They also want to browse experiments and projects by investigator or by lab, explore the images within an experiment, and download a set of images from an experiment.

It is important that the electronic notebook be flexible in regard to the metadata that can be stored. Some fields (such as the configuration of the microscope) are always useful, but others (such as the details of sample preparation and labeling) vary greatly between labs, and even between experiments. The metadata schema must be easily adaptable to the changing needs of the researchers, or the system is unlikely to be adopted.

MetaXpress (http://www.moleculardevices.com/pages/software/metaxpress.html), BioImage [2], Global Image Database (GID) [5], and Cell-Centered Database (CCDB) [9] are examples of electronic-notebook-style image databases. GID provides minimal set of annotations in a fixed format that caters to many image types and are useful for sharing. In CCDB, a backbone schema is provided which can be enhanced by specialized tables. There is additional provision for storing the results of image analyses as pre-defined object types. The open microscopy environment (OME) [13] and Bisque systems (http://www.bioimage.ucsb.edu/) also serve as electronic notebooks, although they provide more advanced functionality, described in later sections. Further, Bisque allows the user to modify the metadata schema.

Sharing and Exploration

Sharing images beyond the investigator's immediate colleagues leads to two new requirements for the image database: integration of metadata and image-content–based search. The flexibility of modifying the schema that is so valuable in an electronic notebook impedes sharing because the system will need to know which fields in one schema corresponds (perfectly or partially) to which fields in another. Schema matching techniques have been developed that unify the schemas based on both structure and contents [1]. A related problem pertains to the contents of the fields. For instance, the same antibody can be known under different names. As a second example, one researcher may describe a cell as a "photoreceptor" whereas another scientist may be more specific and call it a "cone." Whereas the Gene Ontology (GO, http://www.geneontology.org/) is a start, it will need to be combined with other ontologies or controlled vocabularies.

For *in vivo* images, the location in the tissue or organism that was imaged is another important piece of metadata. To understand and explore such information, registration and mosaicking techniques have been developed to align the image to an *atlas* [4].

So far, in addition to the images themselves, only metadata, i.e., information about the images, have been considered. A second kind of image-associated data are *annotations*. Annotations are data *derived from* the images. They can, for instance, indicate specific objects of interest (e.g., crescent-shaped nuclei, neurite branching points) or summarize measurements (e.g., cell counts, neurite length distributions). Image features can also be considered annotations. Annotations can be stored in an electronic notebook, just like metadata. Annotations are, however, more important for sharing and exploration, for they allow the user to find interesting images even if they were acquired for a completely different purpose. For instance, a researcher may want to find all perturbations (drug treatments, gene knockdowns, etc.) that led to a high rate of crescent-shaped nuclei.

Although metadata and annotations have many similarities, the latter are much more challenging for the database designer. Each metadata field has one true value which is provided by the original investigator, typically at the time the image is uploaded to the database (and perhaps updated or corrected later). In contrast, because annotations are interpreted from the image, more than one user or tool may add different values for the same annotation. Two researchers may disagree on what kind of cell a certain object is, and two cell counting tools will produce different results. The database must therefore be able to store uncertain information and track the provenance of the annotations. Handling uncertain information in databases is an active research area [7,11].

Another open question is how to control access to annotations. When sharing sets of images between labs, issues such as who owns the data, who are allowed to access them, who can run analysis tools on them, who can update the database with new results, etc., are critical. Another important factor is ensuring the quality of data, including the validation, accuracy, completeness, and integrity of the images being processed or uploaded, as well as the annotations produced by analysis tools. Similar issues are discussed by Toga et al. [14].

Although search by metadata and annotations is important and useful, it has an obvious limitation: one can only find what others have annotated. For effective exploration, one must be able to search the images by content (see Content-Based Similarity Search and Summarization in the Scientific Fundamentals section).

Analysis

The third principal application of image databases to biomedical images is as a platform for numerical and statistical analysis of the annotations derived from the images.

Combinations of analysis steps can be viewed as networks, where the results of one analysis feeds into another. For example, ridges can be extracted directly from the image, and can, therefore, be considered as low-level annotations. Next, the ridges are filtered and linked into neurites, which are then combined with cell bodies (detected by some other method) into whole neurons. Note that neurites and neurons are objects that have biological meaning. Finally, one may measure the amount of a certain protein within the cell. The Open Microscopy Environment (OME) [13] provides a module system for tracking the provenance of the resulting data. Each module execution is recorded, relating the module and its parameters to the input and output data.

Analysis of annotations frequently involves classification. The reason is that clustering and other unsupervised methods often pick up cell-cycle–dependent variations and other differences that are real and profound, but irrelevant to the question of interests. This can be avoided by classifying perturbed samples from controls [8].

Future Directions

Whereas the size of the images is a problem for image databases, managing the analyzed data is an even bigger problem because the analyses can be numerous and the annotations produced by such analyses can be queried in complex ways. As an example, a single experiment can comprise of 350 GB of images, and over 600 features are extracted from each cell for a total of over 20 GB of annotations [10]. Future experiments are expected to reach 20 times this size. Therefore, efficient index structures and database access methods will be needed.

Queries posed to databases as part of an analysis process generally involve a large number of records. For instance, a classification task may need to look at the records of all the cells perturbed in a certain way. However, due to feature selection, only part of the record may need to be examined – say, 50 of the 600 features. Column-oriented DBMSs [12] have the potential to improve the performance dramatically for such queries. Another very general approach to speeding up queries is to approximate the result based on only part of the data. Techniques that give statistical guarantees about the quality of approximation have been proposed for many query types, but are so far only available in specialized applications and in research prototypes of database systems.

Cross-references

- ▶ Biological Metadata Management
- ▶ Biostatistics and Data Analysis
- ▶ Image Database
- ▶ Machine Learning in Computational Biology
- ▶ Ontologies and Life Science Data Management

Recommended Reading

1. An Y., Borgida A., Miller R.J., and Mylopoulos J. A semantic approach to discovering schema mapping expressions. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 206–215.
2. Carazo J.M., Stelzer E.H., Engel A., Fita I., Henn C., Machtynger J., McNeil P., Shotton D.M., Chagoyen M., de Alarcón P.A., Fritsch R., Heymann J.B., Kalko S., Pittet J.J., Rodriguez-Tomé P., and Boudier T. Organising multi-dimensional biological image information: the BioImage database. Nucleic Acids Research, 27(1):280–283, 1999.
3. Datta R., Li J., and Wang J.Z. Content-based image retrieval: approaches and trends of the New Age. In Proc. 7th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, 2005, pp. 253–262.
4. Fowlkes C.C., Hendriks C.L.L., Keränen S.V.E., Biggin M.D., Knowles D.W., Sudar D., and Malik J. Registering *Drosophila* embryos at cellular resolution to build a quantitative 3D atlas of gene expression patterns and morphology. In Proc. Int. Workshop on Bioimage Data Mining and Informatics, IEEE Computational Systems Bioinformatics Conference, 2005.
5. Gonzalez-Couto E., Hayes B., and Danckaert A. The life sciences global image database (GID). Nucleic Acids Research, 29(1):336–339, 2001.
6. Ljosa V., Bhattacharya A., and Singh A.K. Indexing spatially sensitive distance measures using multi-resolution lower bounds. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 865–883.
7. Ljosa V. and Singh A.K. APLA: indexing arbitrary probability distributions. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 946–955.
8. Loo L.-H., Wu L.F., and Altschuler S.J. Image-based multivariate profiling of drug responses from single cells. Nature Methods, 4(5):445–453, 2007.
9. Martone M.E., Zhang S., Gupta A., Qian X., He H., Price D.L., Wong M., Santini S., and Ellisman M.H. The cell-centered database: a database for multiscale structural and protein localization data from light and electron microscopy. Neuroinformatics, 1(4):379–395, 2003.
10. Moffat J., Grueneberg D.A., Yang X., Kim S.Y., Kloepfer A.M., Hinkle G., Piqani B., Eisenhaure T.M., Luo B., Grenier J.K., Carpenter A.E., Foo S.Y., Stewart S.A., Stockwell B.R., Hacohen N., Hahn W.C., Lander E.S., Sabatini D.M., and Root D.E. A lentiviral RNAi library for human and mouse genes applied to an arrayed viral high-content screen. Cell, 124:1283–1298, 2006.
11. Sarma A.D., Benjelloun O., Halevy A., and Widom J. Working models for uncertain data. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
12. Stonebraker M., Abadi D.J., Batkin A., Chen X., Cherniack M., Ferreira M., Lau E., Lin A., Madden S., O'Neil E., O'Neil P., Rasin A., Tran N., and Zdonik S. C-store: A Column-Oriented DBMS. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 553–564.
13. Swedlow J.R., Goldberg I., Brauner E., and Sorger P.K. Informatics and quantitative analysis in biological imaging, Science, 300(5616):100–102, 2003.
14. Toga A.W. Neuroimage Databases: The Good, the Bad and the Ugly. Nature Reviews Neuroscience, 3(4):302–309, 2002.
15. Zhou X.S. and Huang T.S. Relevance Feedback in Image Retrieval: A Comprehensive Review. Multimedia Systems, 8(6):536–544, 2003.

Image Management for Life Sciences

▶ Image Management for Biological Data

Image Metadata

Frank Nack
University of Amsterdam, Amsterdam,
The Netherlands

Synonyms

Pictorial metadata; Picture metadata; Image representation

Definition

A digital image is a representation of a two- or three-dimensional image, where the representation can be of vector or raster type.

Metadata is data about data of any sort in any media, describing an individual datum, content item, or a collection of data including multiple content items. In that way, metadata facilitates the understanding, characteristics, use and management of data.

Image metadata is structured, encoded data that describes content and representation characteristics of information-baring image entities to facilitate the automatic or semiautomatic identification, discovery, assessment, and management of the described entities, as well as their generation, manipulation, and distribution.

Historical Background

Many of the techniques of digital image processing were developed in the 1960s at, among others, the MIT, Bell Labs, and the University of Maryland. These works tried to automatically generate content representations that could support research on satellite imagery, wire-photo standards conversion, medical imaging, character recognition, and photo enhancement. In the 1970s, digital image processing improved mainly because cheaper computers and dedicated hardware that allowed real-time image processing [2] became available. The next 20 years not only saw a further improvement on automatic feature extraction for domains, such as scientific, industrial, medical, and environmental research, but also a steady infiltration of those technologies in everyday media environments, such as image editing tools (Photoshop (http://en.wikipedia.org/wiki/Adobe_Photoshop), Illustrator (http://en.wikipedia.org/wiki/Adobe_Illustrator_CS2), GIMP (http://en.wikipedia.org/wiki/GTK%2B), or Maya (http://en.wikipedia.org/wiki/Maya_%28software%29)), new media authoring tools (Director/Shockwave (http://www.adobe.com/products/director), or Flash (http://www.adobe.com/products/flash)), and, more importantly, in the development of digital cameras. Images turned into a common information item that people could handle as easily as text.

The real push for the digital image came with the emergence of the world wide web (web) and the improvements of sensor technology, especially in digital photography. These swift developments in image distribution and generation in combination with easy to use web presentation technology (e.g., Dreamweaver (http://en.wikipedia.org/wiki/Dreamweaver), Frontpage (http://en.wikipedia.org/wiki/Microsoft_FrontPage), HTML (http://www.w3.org/MarkUp), and SMIL (http://www.w3.org/AudioVideo)) deeply changed the social way of exchanging information, increasing the available amount of images dramatically.

The answer to the resulting media-based information flood was research that focused again on automatic ways to index the available images in a timely and meaningful way. The goal was now to make use of image processing based on features required for interactive image understanding. Machine-generated metadata, however, turned out to be problematic as it is exclusively organized around the sensory surface structures of media, i.e., the physical features of an image, resulting in the sensory and semantic gap [14].

By the beginning of the twenty-first century this need for semantic-aware metadata schemata forced research to explore new ways of content representation. A large number of initiatives developed metadata standards to allow machines as well as humans to access the semantics of media items, such as Dublin Core (http://www.dublincore.org/), the Art and Architecture Thesaurus (AAT) by the J. Paul Getty Trust (http://www.getty.edu/research/conducting_research/vocabularies/aat/), the semantic web activity of the W3C (http://www.w3c.org/2001/sw/), and ISO's MPEG-4, MPEG-7 and MPEG-21 (http://www.chiariglione.org/mpeg/).

The interesting aspect of particularly the ISO and related standards had been that they tried to merge the high-level conceptual aspects of their content description with low-level structures of feature representation as used in signal processing, to allow the processing of audio-visual information over several semantic levels.

The common foundation for this fusion applies XML-based description languages.

A drawback of these approaches, namely regarding the process of attributing metadata to a media item as a terminated process, resulted in research that introduced new mechanisms to overcome the static structure of annotations, where flexibility is achieved by agreeing upon the collection of semantic-based and machine-processable metadata during established media workflow practices [5,6,9].

Around 2003, the web saw the growth of massive image databases, such as Flickr (http://www.flickr.com), that are mainly build on user generated content. In such environments people do not agree beforehand on an annotation taxonomy or ontology. As a result, research developed folksonomy tagging (also known as collaborative tagging, social classification, social indexing, and social tagging), a method of collaboratively creating and managing tags to annotate and categorize content. In folksonomy tagging metadata is not only generated by experts but mainly by creators and consumers of the content, where a tag is a keyword or term associated with or assigned to a piece of information (a picture, a map, etc.), which enables keyword-based classification and search. The advantage of tagging is its ease of use - creating a vocabulary based on freely chosen keywords instead of a controlled set of terms and structures. This approach, though highly popular, carries serious problems. Typically there is no information about the semantics of a tag, no matter if it is a single tag or a bag of tags. Additionally, different people may use drastically different terms to describe the same concept. This lack of semantic distinction can lead to inappropriate connections.

Foundations

When a person looks at an image, they first perceive the image on an optical level where they try to identify as many objects as possible in the available time of perception. Each object is mentally transformed into an iconic sign, which processes some properties of the object represented. The signification of iconic signs is based on socially determined small semantic systems (codes) and rules for their combination. Some of the code systems are: recognition, tonal, iconic, iconographic, rhetoric, and stylistic [7].

The iconic code is by far the most valuable code, because it defines the articulation potential of visual material. The creation of meaning in visual material is based on a *triple articulation* of figure, sign and *semes* and receives its expression by convention. The three elements are described as such:

1. A *figure* forms conditions for perception, such as relationships between object and background, or contrast in light.
2. A *sign* denotes, using conventionalized graphical methods, units of understanding (nose, sky), abstract models, or idealized diagrams of the object (the sun as a circle with thread-shaped beams).
3. *Semes* are complex iconic phrases, such as "this is a man standing in profile." They are the most simply catalogued, since the iconic code works most often on their level only.

The fact of a conventionalized triple articulation is important since it describes the essential difference from natural language, which has two articulations (phonemes and morphemes). Thus, comparing an object in an image with the corresponding word, the visual object always exceeds the concept of the word, as the image will portray specific qualities about the object for which the word is simply inadequate (see the tag problem mentioned above).

The organization of signs in an image is provided by two types of structures:

1. Syntagmatic, which represents a sequence of signs in which the relation of parts determines their meaning.
2. Paradigmatic, which represents potential substitutions in which a range of potential candidates can take the place of a sign in a syntagmatic structure.

The main application of image metadata is the retrieval of an image, where content features play an essential role. Search based low-level features functions on an iconic level, as most of the low-level feature based description methods. The key methods of how users search for an image are:

- Search by specification: aims for searching the identical replicate of the image the user has in mind or the identical object the user needs. Essential is the provision of optional variants (prosodic features) that form the conditions for perception and thus support identification [4].
- Search by categorization: aims for retrieving an arbitrary image representative of a specific class.

Categories may be derived from labels or emerge from the database [19]. This type of search usually requires a domain specific definition of similarity. Here, ontologies might become applicable.

- Search by association: aimed at browsing through a large set of images from unspecified sources. Association-based search usually implies iterative refinement of the query, which mainly asks for establishing similarities between images. Important in such search applications is the provision of relevance feedback [8] or the provision of additional sources [16].

Supporting these types of search methods are needed and allow the description of content so that a machine or a human user can access the images. Some of them can be applied in an automatic fashion, those are image feature based, and some provide high-semantic descriptions, which are often based on manual annotation. The latter are required, because, as described above, there is additional context that specifies the need of a query, e.g., the esthetics' the picture has to provide.

Content Description Based on Invariant Features

The purpose of image processing is to enhance aspects in the image data relevant to the user query. This can be achieved through the description of invariant features, such as, color, shape and texture. Invariant features tend to be object-specific information as they are insensitive to the accidental conditions of the sensing. It is important to note that it is the user who has to specify the minimal set of invariant features as it is part of his or hers intention [3].

- *Color:* The two general problems addressed in work on color are variances in color and differences in human perception. Color representation techniques, among others, are:
  - RGB color representation describes the image in its literal color properties, namely the Red, Green and Blue properties of each pixel in an image, where two-dimensional images are recorded in frontal view under standard conditions.
  - Opponent color representations, which uses the opponent color axes that isolate the brightness information on the third axis.
  - Lab-space approach, which exploits the Euclidean distance between two color representations to model human perception of color differences.
  - The HSV-representation exploits the hue, which is invariant under the orientation of the object with respect to the illumination and camera direction.
  - Hidden Markov models [11]
  - Clusters in a color histogram, where the RGB space is searched to identify which pixels in the image originate from one uniformly colored object.
- *Shape:* Shape is considered as the collected properties that capture differential geometrical details in the image [15]. Methods to represent shape are scale-based theory, conspicuous shape geometric invariants, or differential geometric invariants.
- *Texture:* Texture covers every aspect of an image that is not described through color and shape. The essential texture analysis techniques are: the Markovian analysis, multiscale autoregressive MRSAR-models, or wavelets.

Content Description Based on Semantic Features

The invariant features serve as a preprocessing step within the usual approach in content-based image retrieval, which divides the image first into parts and then computes the features for those parts. The essential segmentation methods are strong and weak segmentation.

- Strong segmentation divides the data into regions in such a way that a region contains the pixels of the silhouette of an object in the real world and nothing else. The problem is that this approach usually only succeeds sophisticated techniques in very narrow domains, such as trademark validation.
- Weak segmentation aims to support broad domains of general images by grouping image data into inconspicuous regions that are internally homogeneous according to some criterion. Weak segmentation is used in many retrieval systems, either as a purpose of its own or as a preprocessing stage for data-driven model-based object segmentation.

Both approaches end up in a preferred set of features [14], which can be classified as follows:

- Accumulating features aggregate the spatial information of a partitioning irrespective of the image data. The histogram [17] is the common method applied, such as the color histogram. Alternatives are the correlogram, or the autocorrelogram. An example of accumulative features might be those calculated from the top part of a picture, which effectively identify an image as indoor or outdoor space. The danger of accumulative features is the inability to discriminate among different entities and semantic meanings in the image.

- Salient features are a typical example of weak segmentation. The idea here is that grouping of the data is performed resulting in homogeneous regions. From the merged regions, a selection is made on their salience, where saliency is understood as those special points that survive longest while gradually blurring the image in scale space.
- Shape and object features, which focus on segmenting the object in the image. The problem is the automatic segmentation in broad domains. Yet, often it is not necessary to know exactly where an object is located but rather that its presence can be identified. Techniques for that are elastic matching and multi-resolution representation of shapes, multi-scale models of contours, the description of the object boundaries, or the description of global shape invariants.
- Combined entities of features, where calculations for different entities in the image and relationships between them are available. Such a structural feature set may contain feature values and spatial relationships, a hierarchically ordered set of feature values, relationships between point sets or object sets, or a graph of relations between blobs.

Interpretation of Features

In general there are two ways to make use of computed feature sets for the interpretation of an image.

- Deriving an unilateral interpretation from the feature set. This approach encodes an approximate subset of possible interpretations of an image relevant for a particular application. In that way the subset describes the semantics associated to features for the application [13].
- Comparing the feature set with the elements in a given data set based on similarity.

Similarity is an interpretation of the image based on the difference with another image. For each of the earlier described feature types, a different similarity measure is needed:

- *Similarity of objects* can be established through shape comparison based on transforms, moments, deformation matching, scale space matching, and dissimilarity measurement.
- *Similarity of structure* can be established based on a Bayesian framework, topological arrangements of relevant domain parts, spatial relationships between objects, or tree or graph representations.
- Similarity of *salient features* can be achieved by means of the distance between the feature vector composed of, e.g., the color, texture, position, and direction of the two ellipses, derived from the feature values measured of the blobs resulting from weak segmentation of the two images.

Under normal circumstances similarity comparison is not performed on a 1:1 image basis but rather on a 1:n image basis. For the latter cases, as in large image databases, gained experiences during the comparison of images can be exploited for better results. In that way similarity detection is understood as a classification problem to be solved based on statistical pattern recognition. Additional learning methods are the use of transduction (e.g., for partially labeled data), probabilistic constellations of features (e.g., for unlabeled data sets), or latent semantic indexing.

Image Interpretation Based on High-Level Semantic Metadata

As pointed out earlier, the representation of an image based on low-level features might be useful for narrow domains, such as the validation of trademarks. Every information that is based on higher-level codes than the iconic, e.g., the name of the image creator or the image's contribution to the technique of painting or photography, can not be extracted in that manner.

For those codes the annotation is based on concepts. The most basic form of such an annotation is a tag. The most fundamental set of tags for the annotation of images is certainly the Dublin Core set. The most complex description framework so far are those developed by MPEG-7 and the W3C's Multimedia Incubator Group (see also the Media Annotation Working Group: http://www.w3.org/2008/01/media-annotations-wg.html). The metadata falling in this category often require manual annotation [1], where the available structures are organized either in hierarchical or graph structures (the syntagmas) and use thesaurus or ontolological formats (class, subclass and their properties as well as relations between them) to allow for paradigmatic methods of choice to support advanced search (query extension) [10], or interpretation and generation.

Dublin Core [18]

The Dublin Core Metadata Element Set is a vocabulary of fifteen properties for use in resource description. The fifteen properties are: contributor, creator, date, description, format, language, publisher, relation, rights, source, subject, title, type.

Since January 2008, Dublin Core includes formal domains and ranges in the definitions of its properties. For supporting conformance of existing implementations of "simple Dublin Core" in RDF, domains and ranges have not been specified for the fifteen properties of the dc: namespace (http://purl.org/dc/elements/1.1/). Instead, the Dublin Core Metadata Initiative has opted for creating fifteen new properties with "names" identical to those of the Dublin Core Metadata Element Set Version 1.1 in the dcterms: namespace (http://purl.org/dc/terms/). These fifteen new properties have been defined as subproperties of the corresponding properties of DCMES Version 1.1 and assigned domains and ranges as specified in the more comprehensive document "DCMI Metadata Terms."

MPEG-7 [12]

MPEG-7 standardizes descriptions of AV data content in multimedia environments. It provides descriptions of multimedia content on varying complexity levels to let users search, browse, filter, or interpret content using search engines, filter agents, or any other program. MPEG-7 offers a set of AV description tools in the form of descriptors (Ds) and description schemata (DS) a valid MPEG-7 description should adhere to. Descriptors usually bind a feature to a set of values. Description schemata specify the structure and semantics of the relationships between the components of descriptors and between other description schemata. These structures let users create application-specific content descriptions (see Fig. 1 as an example).

The standard has eight parts, each responsible for one aspect of the functionality, of which 3 are relevant for visual data:

- Part 2 - the DDL specifies the language for defining the standard set of description tools (description schemata, descriptors, and data types), new tools, and the main parser requirements. The DDL is based on XML-Schema, developed by the W3C.

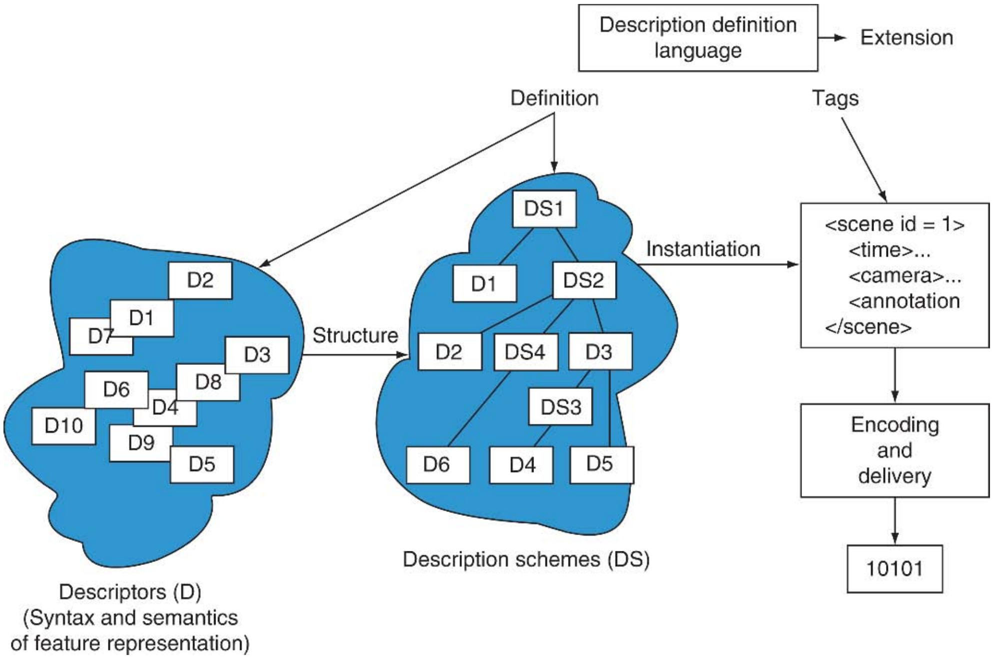


Image Metadata. Figure 1. The main MPEG-7 elements. Content authors can use these structures to create application-specific content descriptions.

- Part 3 - Visual consists of schemata and descriptors covering basic visual features such as color, texture, shape, and face recognition. It provides the descriptor syntax and description schemata in normative DDL specifications and the corresponding binary representations.
- Part 5 - Multimedia Description Schemes (MDS) specifies generic description tools pertaining to multimedia, including audio and visual content. MDS covers the basic elements for building a description, the tools for describing content and relating the description to the data, and the tools for describing content on organization, navigation, and interaction levels. The MDS alone forms more than half of the complete standard and has its own internal structure, shown in Fig. 2.

As the standard is rather large, MPEG has begun to establish media profiles (MPEG-A to MPEG-E) that integrate multiple MPEG technologies. The relevant visual profile is MPEG-C.

The W3c's Incubator Group [20]

The work of this group has explored the advantages of using Semantic Web languages and technologies for the creation, storage, manipulation, interchange and processing of image metadata. In addition, it provided guidelines for Semantic Web-based image annotation, illustrated by use cases. In its publications the relevant RDF and OWL vocabularies are discussed, along with a short overview of publicly available tools.

Key Applications

Image metadata is useful for the creation, manipulation, retrieval and distribution of 2D or 3D digital image sources within domains, such as

- The creative industries (e.g., fine arts, entertainment, journalism, etc.),
- Education (e.g., in computer based training courses, military or industrial training),
- Environmental research (e.g., Interpretation of satellite images),
- Medicine (e.g., Identification of cancer cells in a lung scan etc.).

Future Directions

Research has still to address how the merge between high-level semantic descriptions and low-level feature-based descriptions can be achieved. In this context, solutions need to be found that merge the results of folksonomy tagging and defined description vocabularies into a suitable description

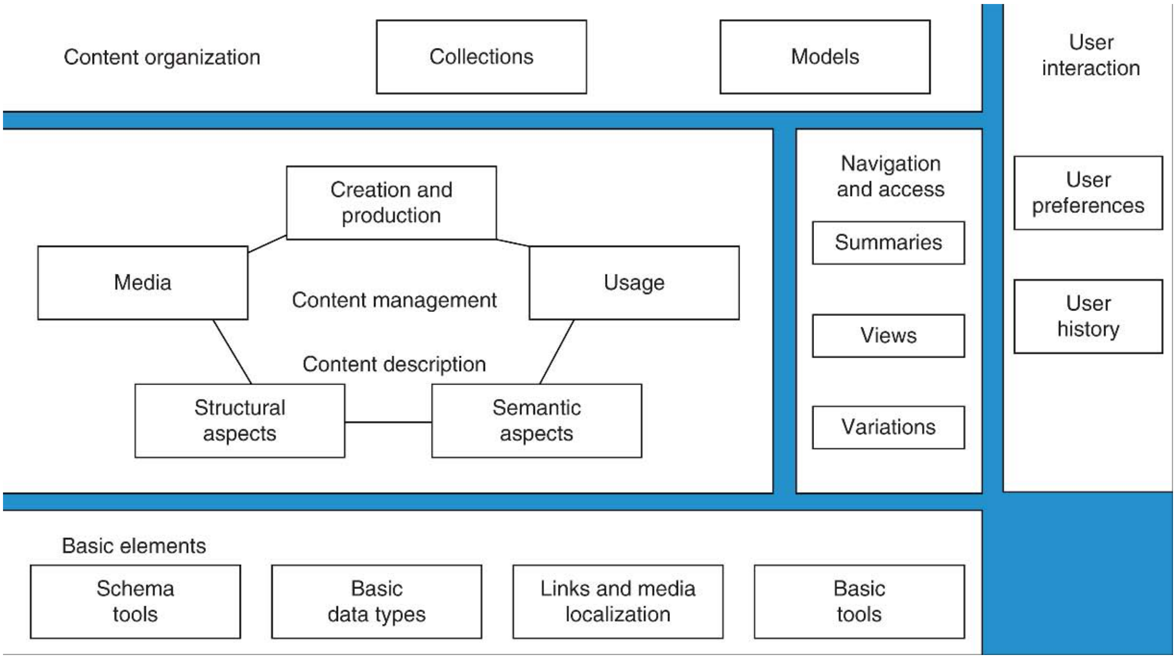


Image Metadata. Figure 2. Overall organization of MPEG-7 multimedia description schemes.

framework. Finally, it is important that easy to use annotation tools will be developed.

Cross-references

▶ Image Content Modeling
▶ Image Representation
▶ Multimedia Metadata
▶ Video Metadata

Recommended Reading

1. Ahern S., Davis M., Eckles D., King S., Naaman M., Nair R., Spasojevic M., and Hui-I Yang J. ZoneTag: designing context-aware mobile media capture to increase participation. In Proc. Pervasive Image Capture and Sharing: New Social Practices and Implications for Technology Workshop, 2006.
2. Blasser A. (ed.) Database Techniques for Pictorial Applications. Lecture Notes in Computer Science, Springer, London, UK, 1979.
3. Burkhardt H. and Siggelkow S. Invariant features for discriminating between equivalence classes. Nonlinear Model-Based Image Video Processing and Analysis. Wiley, NY, 2001, pp. 269–307.
4. Cox I.J., Miller M.L., Minka T.P., and Papathomas T.V. The Bayesian image retrieval system, picHunter: theory, implementation, and pychophysical experiments. IEEE Trans. Image Process., 9(1):20–37, 2000.
5. Davis M. Active capture: integrating human-computer interaction and computer vision/audition to automate media capture. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2003, pp. 185–188.
6. Dorai C. and Venkatesh S. Bridging the semantic gap in content management systems – computational media aesthetics. In Media Computing Computational Media Aesthetics, C. Dorai, S. Venkatesh (eds.). Kluwer, Boston, MA, 2002.
7. Eco U. Articulations of the cinematic code. In Movies and Methods, B. Nichols (ed.). University of California Press, Berkeley, 1976, pp. 590–607.
8. Frederix G., Caenen G., and Pauwels E.J. PARISS: Panoramic, Adaptive and Reconfigurable Interface for Similarity Search. In Proc. Int. Conf. Image Processing, 2000 vol. 3, pp. 222–225.
9. Hardman L., Obrenovic Z., Nack F., Kerherve B., and Piersol K. Canonical processes of semantically annotated media production. Multimedia Systems, 14(6):327–340, 2008.
10. Hollink L. Semantic Annotation for Retrieval of Visual Resaources. Ph.D thesis, Vrije Universiteit, Amsterdam.
11. Lin H.C., Wang L.L., and Yang S.N. Color image retrieval based on hidden Markov models. IEEE Trans. Image Process., 6(2):332–339, 1997.
13. Nack F., Windhouwer M., Hardman L., Pauwels E., and Huijberts M. The role of highlevel and lowlevel features in style-based retrieval and generation of multimedia presentations. New Rev. Hypermedia Multimedia, 7(1):7–37, 2001.
14. Smeulders A.W.M., Worring M., Santini S., Gupta A., and Jain R. Content-based image retrieval: the end of the early years. IEEE Trans. Pattern Anal. Mach. Intell., 22 (12):1349–1380, December 2000.
15. Smith S.M. and Brady J.M. SUSANÐA new approach to low level image processing. Int. J. Comput. Vis., 23(1):45–78, 1997.
16. Swain M.J. Searching for multimedia on the World Wide Web, icms. In Proc. Int. Conf. on Multimedia Computing and Systems, 1999, pp. 32–37.
17. Swain M.J. and Ballard B.H. Color indexing. Int. J. Comput. Vis., 7(1):11–32, 1991.
18. The Dublin core metadata initiative. Available at: http://www.dublincore.org/. Access date: October 12th 2008.
19. Weber M., Welling M., and Perona P. Towards automatic discovery of object categories. In Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition, 2000, pp. 2101–2108.
20. W3C multimedia incubator group. Available at: http//www.w3.org/2005/incubator/mmsem/. Access date: October 12th 2008.

Image Query Processing

▶ Image Querying

Image Querying

Ilaria Bartolini
University of Bologna, Bologna, Italy

Synonyms

Image query processing

Definition

Image querying refers to the problem of finding objects that are *relevant* to a user query within image databases (Image DBs). The classical solutions to deal with such problem include the *semantic*-based approach, where an image is represented through metadata (e.g., keywords), and the *content*-based solution, commonly called *content-based image retrieval* (CBIR), where the image content is represented by means of low-level features (e.g., color and texture). While with the semantic-based approach the image querying problem is transformed into an information retrieval problem, for CBIR more sophisticated query evaluation techniques are required. The usual approach to deal with this is illustrated in Fig. 1: By means of a graphical user interface (GUI), the user provides a query image, by sketching it using graphical tools, by uploading an image she/he has, or by selecting an image supplied by the system. Low-level features are extracted for such image (possibly dividing it into regions, see below); such features are then used by the query processor to retrieve the DB images having similar characteristics.

How the set of relevant DB images is determined depends on which low-level features are used to characterize image content, on the criterion used to *compare* image features, on how DB objects are *ranked* with respect to the query (based on either a quantitative measure of similarity or qualitative preferences), and, finally, on whether the user is interested in the whole query image or only in a part of it. All these aspects strongly influence the *query evaluation* process.

Historical Background

In spite of the many efforts spent so far, the problem of retrieving relevant objects within *image databases* (Image DBs) is still a complex task. Following the semantic-based approach, images are described by means of metadata such as keywords, captions, or descriptions and the retrieval is performed over such words using *annotation-based image retrieval* techniques. In this direction, several solutions have been recently proposed, such as image extensions to public search engines like Google (http://images.google.com/) and Yahoo (http://images.search.yahoo.com/). Such systems consider the contextual information of a crawled image (like the image filename, its title, the surrounding text, etc.) to infer the relevance of the image to the query. In a similar way, some systems, like flickr (http://flickr.com/), assess the relevance of an image to the query by taking into account the characterization of the image provided by the user. However, such a manual image annotation process is expensive and time-consuming. In order to overcome such limitations, there has been a large amount of research done on (semi-)automatic image annotation with the aim to assign meaningful keywords to images by exploiting the information of a pre-annotated set of objects.

With respect to the *content-based image retrieval* (CBIR) solution, the aim is to avoid the use of textual descriptions. This is usually done by using visual similarity to retrieve images, for example, asking for images that are similar to a user-supplied query image, i.e., following the *query by example* (QBE) paradigm fist adopted in the IBM's query by image content (QBIC) system [5]. In particular, each image is characterized using global *low-level features*, such as color and texture, and the result of a query consists in the set of DB

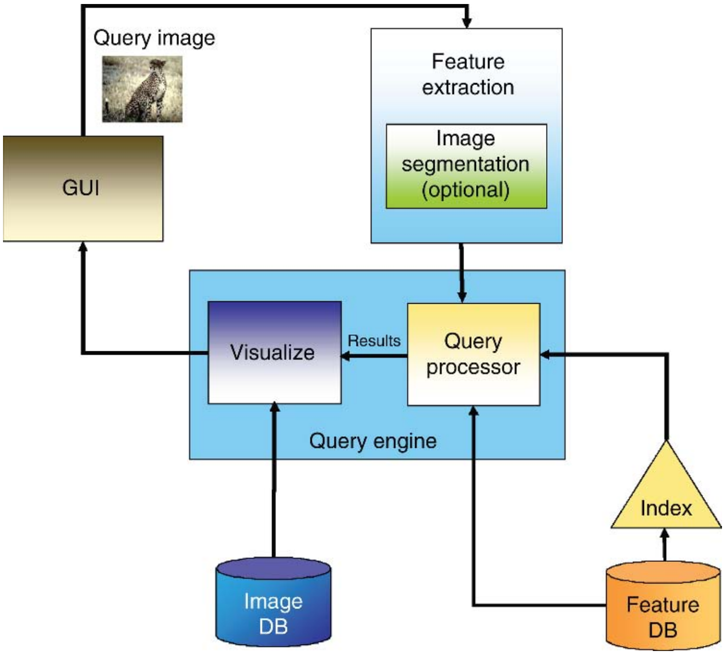


Image Querying. Figure 1. The image querying scenario for CBIR.

objects that better match the visual characteristics of the target image, according to a predefined *similarity criterion*, which is in turn based on low-level features [8]. Although CBIR represents a completely automatic solution for the image querying problem, the accuracy of its results is not always completely satisfactory for the user, especially for high-level concept queries, for which low-level features are hardly exploitable due to their low discriminative power. This is largely due to the so-called semantic gap existing between the concept of similarity as percept by the human brain and the one implemented by the system. The effectiveness of this approach still calls for improvement: The use of *relevance feedback techniques* could be of help, but it is still not enough to reach acceptable levels of accuracy.

More recently, the *region-based image retrieval* (RBIR) approach has been proposed, which has lead to promising results. With respect to the case in which images are represented by means of global descriptors, RBIR is able to characterize the image content in a more precise way by *segmenting* each image into a set of homogeneous regions from which a set of low-level features are extracted. As a consequence, most of the modern image database systems adopts the RBIR paradigm in order to improve the retrieval accuracy [9,1,3,4,6,10]. Almost all such systems treats each region as a separate query and somehow aggregate the so-obtained partial results in order to derive the final answer. This property introduces a number of new interesting query processing problems with respect to the case in which the segmentation is not considered. Among these, which constraints must be satisfied by the aggregation rule in order to provide the query result and which criterion has to be followed to order the DB images with respect to the query. Finally, with RBIR, new query types, such as partial queries, are supported.

Foundations

The general approach followed by RBIR systems is to divide an image I into a set of homogeneous regions, i.e., set of pixels that share similar visual characteristics, and to represent each of them by means of a set of low-level features, such as color and texture. Thus, any image I is seen as a complex object. Regions comparison is obtained by defining a *region similarity function*, s_R, able to produce a scoring value which quantifies their visual similarity. Given a query image I^q, the set of relevant DB images to I^q is computed starting from the similarities between the query regions and the regions of DB images. This requires first to somehow *match* regions of the query to regions of DB images, by using the proper aggregation of region similarities, and then to *rank* DB images so as to produce the query result (see Fig. 2).

Formally, the image querying problem can be concisely formulated as follows:

Problem

Given a query image I^q composed of regions, an image database $\mathcal{IDB}$, where each image $I \in \mathcal{IDB}$ is composed of regions, and a region similarity function, $s_R(R_i, R_j)$, that for each pair of regions, (R_i, R_j), returns their

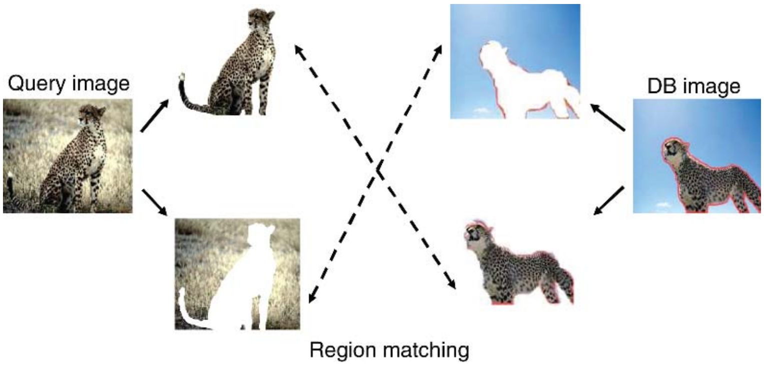


Image Querying. Figure 2. The similarity between the query image and a DB image is assessed by taking into account the similarity between matched regions.

similarity score, determine the set of relevant *images in* $\mathcal{IDB}$ *wrt* I^q.

Instantiating the general problem can be done in different ways, since different coordinates are involved in the definition of what "relevant" actually means. Among these coordinates, the rules according to which a region of the query can be coupled to regions of a DB image (conventionally called *matching type*) and the aggregation modality applied to the region similarity scores in order to assess the query result (i.e., the *ranking model*).

Matching Type

The matching type defines which set of constraints applies when the component regions of the query image $I^q = \{R_1^q,\ldots,R_n^q\}$ have to be matched to the component regions of a DB image $I = \{R_1,\ldots,R_m\}$.

Two relevant cases for matching types are the *one–to–one* ($1 - 1$) and the *many–to–many* ($n - m$) matching types. In the $1 - 1$ case, each region of image I^q is associated to at most one region of I, and vice versa. In particular, each matching has to be *complete*, i.e., if $n > m$ (respectively, $n < m$) then only $n - m$ (resp., $m - n$) regions of I^q (resp., I) have to remain unmatched (refer to Fig. 3 for an example).

With the $n - m$ matching type, each region of I^q can be associated to many regions of I, and vice versa. This, however, could lead to undesired (pathological) results. For example, a single region of the query could be matched to *all* regions of a DB image. This has been termed "the two tigers problem" in [11] since it arises when a single region (a tiger) of the query image is very similar to multiple regions of a DB image (e.g., containing two tigers).

A special case of $n - m$ matching that avoids this problem is the Earth Mover Distance (EMD) matching [7], where variable-sized pieces of regions are allowed to be matched (the size of each region defines the maximum amount for its matching). This contrasts with the $1 - 1$ matching, where elements of fixed size (i.e., regions) are matched individually.

Ranking Model

Two generic models of ranking are possible: *k-Nearest Neighbors* (*k*-NN) and *Best Matches Only* (BMO) [2]. The *k*-NN ranking model (also known as *Top-k selection*) requires to define the image similarity of a DB image I with respect to a query image I^q, $s_I(I^q,I)$, by means of a *numerical scoring function* (*sf*), such as the average, which aggregates the region similarity scores into a global similarity value. In particular, among all valid matchings that satisfy the constraints of the specific matching type, the rationale is to select the one that maximizes the aggregated score. This can be modeled as an optimization problem whose solution depends on the particular choice of the scoring function. For the most commonly used functions efficient algorithms exist: For example, when using the average function with the $1 - 1$ matching type, the problem takes the form of the well-known *assignment problem*, while with the $n - m$ (EMD) matching type (see above), this corresponds to the *transportation problem*. For both such problems the optimal solution can be efficiently found without performing an exhaustive search; however, in the general case, the optimal matching can not be easily found. Figure 3 shows an example of matching for a query image I^q with three regions and a DB image I with four regions under the assumption of $1 - 1$ matching type and the average scoring function. Similarities between regions of I^q and regions of I are arranged in a matrix. Circled cells are those involved in the matching. Note that, since $n < m$, in valid matchings $4 - 3 = 1$ region of I remains unmatched.

Finally, given the query image I^q and two DB images I_1 and I_2, image I_1 will be considered more similar than I_2 to I^q iff $s_I(I^q,I_1) > s_I(I^q,I_2)$ holds. In such a way it is possible to linearly order DB images and return to the user only the k highest scored ones.

The main limitation of the *k*-NN ranking model is that the choice of a particular scoring function clearly influences the final result, i.e., different scoring functions will likely yield different results. This can lead to missing relevant images, because the choice of the scoring function is a difficult task for the user. Moreover, the use of scoring functions limits the expressive power of queries that can be submitted to the system, since all of them will always define a simple linear order on objects. This might prevent their applicability to modern multimedia systems asking for more flexibility in querying [2]. In the BMO model, the result of the query depends on a specific *preference relation* $\succ_p$, where $\succ_p$ is only required to define a strict partial order over images. Image $I_1 \in \mathcal{IDB}$ is in the query result if and only if no other image $I_2 \in \mathcal{IDB}$ is *better* than (or *dominates*) I_1 according to $\succ_p$. Clearly, preference relation $\succ_p$ is based on regions similarity scores (see Fig. 3).

Thus, even if region scores are numerical (by definition), the BMO ranking model does not need to aggregate them using a scoring function. Actually, the result of a BMO query with image I^q is the set of undominated images in $\mathcal{IDB}$, i.e., all and only those images for which no better image (with respect to I^q and to $\succ_p$) can be found in the database.

When considering together the matching type and ranking model coordinates, different scenarios are derived. In the following, algorithms for *k*-NN and BMO queries are provided by considering the simplest way to solve the image querying problem, i.e., when using a sequential scan of the DB. Note that the efficiency of such solutions is clearly quite limited. It is possible to derive efficient algorithms [3,11,2] by exploiting index structures, such as *multi-dimensional or metric indices* (built either on regions of the DB images or on the DB images themselves).

The steps described in Algorithm 1 show the logic of the sequential algorithm for *k*-NN queries, named *k*-NNSeq, to determine the *k* nearest neighbors of the image query I^q: Given the image query I^q, the scoring function *sf*, and the cardinality of the result *k*, the algorithm correctly returns the *k* images that are most similar to I^q according to *sf*. This algorithm covers both cases of $1-1$ and $n-m$ matchings.

Algorithm 2, named BMOSeq, describes the main steps for the sequential evaluation of BMO queries, with the assumption of $1-1$ matching: Given the image query I^q and the preference relation $\succ_p$, the algorithm correctly returns set of undominated images with respect to the query I^q.

It is also important to consider the type of images the user is interested in. The above description deals with *full image* search, i.e., when the user is interested in all regions of the query, but other possibilities exist that introduce minor modifications in the query evaluation process.

For example, *part-of* queries request DB images whose regions are all matched to some query region (the presence of unmatched query regions is not penalized). Two other query types are introduced when the user is given the possibility to select, possibly exploiting a

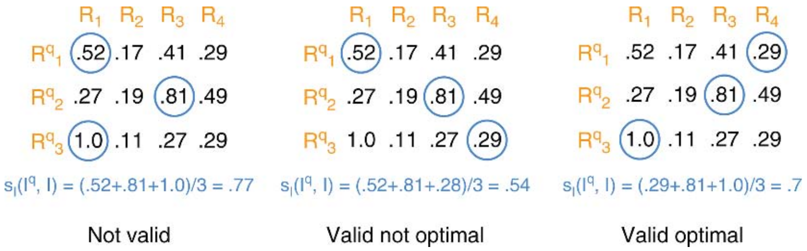


Image Querying. Figure 3. Example of similarity assessment between images I^q and I when adopting the 1–1 matching type and the average scoring function: not valid matching (*left*), valid not optimal matching (*center*), and valid and optimal matching (*right*). If an alternative matching type (e.g., the general $n-m$ matching) is considered, the left matching could become valid (and optimal).

```
Algorithm 1: $k$-NNSeq
Require: $I^q$: query image, $k$: cardinality of result, $\mathcal{IDB}$: image DB, $sf$: scoring function
Ensure: set of relevant $k$ images
  for all images $I \in \mathcal{IDB}$ do
    for all regions $R_j \in I$ do
      for all regions $R_i^q \in I^q$ do
        compute $s_R(R_i^q, R_j)$
    compute matchings between regions $R_i^q \in I^q$ and regions $R_j \in I$
    select the matching that maximizes $s_I(I^q, I)$ by means of $sf$
  return the $k$ images having the highest overall similarity scores $s_I$
```

```
Algorithm 2: BMOSeq
Require: $I_q$: query image, $\mathcal{IDB}$: image DB, $\succ_p$: preference relation
Ensure: set of undominated images
  $\mathcal{U} \leftarrow \emptyset$
  for all images $I_1 \in \mathcal{IDB}$ do
    for all regions $R_j \in I_1$ do
      for all regions $R_i^q \in I^q$ do
        compute $s_R(R_i^q, R_j)$
    $\mathcal{U} \leftarrow \mathcal{U} \cup \{I_1\}$
    for all images $I_2 \in \mathcal{U}$ do
      if $I_1 \succ_p I_2$ then
        $\mathcal{U} \leftarrow \mathcal{U} \setminus I_2$
      else if $I_2 \succ_p I_1$ then
        $\mathcal{U} \leftarrow \mathcal{U} \setminus I_1$
        break (for at line 7)
  return images in $\mathcal{U}$
```

suitable graphical interface, only a subset of query regions: In *partial match* queries the user is looking for DB images containing selected regions of the query (the presence of other regions in the DB image should not be penalized); on the other hand, with a *contains* query DB images are requested to contain selected query regions only (other existing regions reduce the image similarity, differently from the case of part-of queries).

Key Applications

Image querying is an important tool for many modern multimedia applications, such as *digital libraries*, *e-commerce* (where electronic catalogues have to be browsed and/or searched), *edu-tainment* (for example, to search in clipart repositories, or to search and organize personal photo albums in mobile phones or PDAs).

Another interesting application area is the one related to (semi-)automatic *image annotation techniques*, which can be based on assigning to an unlabeled image *I* the keywords associated to the DB images most similar to *I*. Finally, image querying techniques have been also profitably used in *image classification*, for example, to search for similar logo images, for copyright infringement issues, and for the detection of pornography images.

Cross-references

► Annotation-based Image Retrieval
► Feature Extraction for Content-Based Image Retrieval
► Image Database
► Image Retrieval and Relevance Feedback
► Image Segmentation
► Indexing and Similarity Search
► Top-K Selection Queries on Multimedia Datasets
► Video Querying

Recommended Reading

1. Ardizzoni S., Bartolini I., and Patella M. Windsurf: region-based image retrieval using wavelets. In Proc. 1st Int. Workshop on Similarity Search, 1999, pp. 167–173.
2. Bartolini I., Ciaccia P., Oria V., and Özsu T. Flexible integration of multimedia sub-queries with qualitative preferences. Multimedia Tools Applicat., 33(3):275–300, June 2007.
3. Bartolini I., Ciaccia P., and Patella M. A sound algorithm for region-based image retrieval using an index. In Proc. 4th Int. Workshop on Query Processing and Multimedia Issues in Distributed Systems, 2000, pp. 930–934.
4. Carson C., Thomas M., Belongie S., Hellerstein J.M., and Malik J. Blobworld: a system for region-based image indexing and retrieval. In Proc. 3rd Int Conf. on Visual Information Systems, 1999, pp. 509–516.
5. Flickner M., Sawhney H.S., Ashley J., Huang Q., Dom B., Gorkani M., Hafner J., Petkovic D., Steele D., and Yanker P. Query by image and video content: The QBIC system. IEEE Computer, 28(9):23–32, September 1995.
6. Natsev A., Rastogi R., and Shim K. WALRUS: a similarity retrieval algorithm for image databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 396–405.
7. Rubner Y. and Tomasi C. Perceptual Metrics for Image Database Navigation. Kluwer Academic, Boston, MA, December 2000.
8. Smeulders A.W.M., Worring M., Santini S., Gupta A., and Jain R. Content-based image retrieval at the end of the early years. IEEE Trans. Pattern Analy. Machine Intell., 22(12):1349–1380, December 2000.
9. Smith J.R. and Chang S.-F. VisualSEEk: A fully automated content-based image query system. In Proc. 4th ACM Int. Conf. on Multimedia, 1996, pp. 87–98.
10. Wang J.Z., Li J., and Wiederhold G. SIMPLIcity: semantics-sensitive integrated matching for picture libraries. IEEE

Trans. Pattern Anal. Machine Intell., 23(9):947–963, September 2001.

11. Weber R. and Mlivoncic M. Efficient region-based image retrieval. In Proc. Int. Conf. on Information and Knowledge Management, 2003, pp. 69–76.

Image Representation

Valerie Gouet-Brunet
CNAM Paris, Paris, France

Synonyms

multimedia; File format; Image compression; Image metadata; Image content; Image standards

Definition

In computer science, the representation of an image can take many forms. Most of the time, it refers to the way that the conveyed information, such as color, is coded digitally and how the image is stored, i.e., how is structured an image file. Several open or patented standards were proposed to create, manipulate store and exchange digital images. They describe the format of image files, the algorithms of image encoding such as compression as well as the format of additional information often called metadata. Differently, the visual content of the image can also take part in its representation. This more recent concept has provided new approaches of representation and new standards, gathered together into the discipline named *content-based image indexing*.

Historical Background

The first use of digital images began in the early 1920s with the technological development of facsimile transmission, and in particular with the Bartlane cable transmission system that was the first system that translated pictures into a digital code for efficient picture transmission. Later, in the 1960s and 1970s, the advent of digital image technology is closely tied to the development of government programs for space exploration and espionage and also to medical research with the invention of computerized axial tomography. With the availability of the CCD image sensors (charge-coupled device), the private sector also began to make significant contributions in the development of digital cameras. Dissemination of digital images quickly required the proposal of common frameworks for representing images as an improvement of proprietary formats most of the time under license and not easily exchangeable. In the mid-1980s, image format TIFF was created by the company Aldus with the aim of agreeing on a common file format for bitmapped images issued from scanners. In parallel, researchers at Xerox PARC had developed the first laser printer and had recognized the need for a standard means of defining page images. After several fruitless attempts, Adobe Systems proposed the PostScript language in 1982, quickly adapted for driving laser printers. Since then, other file formats dedicated to digital images were also proposed, such as GIF (1987), JPEG (1992), PDF (1993), PNG (1995) and SVG (1998). Today, a great effort is done to propose standard formats able to preserve data integrity for archiving purposes, to migrate easily to future technologies as well as to provide efficient compression for access and dissemination. In 2000, the standard JPEG 2000 was proposed to deal with these objectives for a large range of application domains. Moreover Ambitious programs, MPEG-7 and MPEG-21, started in the late 1990s, are focusing on the harmonization of methods for representing, storing, sharing and accessing multimedia contents (text, audio, image and video) in an unified framework.

Foundations

Basics of Image Representation

Digital images can be classified into two main categories: vector graphics and bitmapped images (also called raster images). Vector images are geometrical 2D objects created with drawing software or CAD (computer-aided design) systems. They are represented with geometrical primitives such as points, lines, curves, and shapes or polygons, which are all based upon mathematical equations. Unlike bitmaps that are resolution-dependent, vector images are scalable, which means that the scale at which they are shown will not affect their appearance. Such images are dedicated to the representation of images with simple content, such as diagrams, icons or logos.

A bitmapped image is composed of a set of dots or squares, called pixels (for picture elements), arranged in a matrix of columns and rows. Each pixel has a specific color or shade of gray, and in combination with neighboring pixels it creates the illusion of a continuous tone image. Unlike human vision, sensors that capture images are not limited to the visual band of the electromagnetic spectrum. Digital images can cover almost the entire spectrum, ranging from gamma to radio waves.

Managing bitmapped images requires the choice and manipulation of several parameters such as:

- *Color model.* A color model is an abstract mathematical model describing the way human color vision can be represented as tuples of numbers. From this model, the combination of dedicated primary colors provide all the colors possible, that are embedded in the corresponding color space. RGB, CMYK, CIELAB and CIELUV are the most known color models and spaces.
- *Dynamic range.* The dynamic range of a digital image, also called color depth, determines the maximum range of gray level or color values carried by each pixel. The number of bits used to represent each pixel determines how many colors can appear in the image. Photographic-quality images are usually associated with 24-bit dynamic range, such as in the JPEG format.
- *Resolution.* Resolution expresses the density of elements, pixels for instance, within a specific area. This term does not have any sense when dealing with digital images as files, but it applies when associating a digital image with a physical support, such as display on a screen, printing on a printer or capture with a scanner. Resolution is classically represented in terms of dpi (dots per inch) unit, which was originally the unit adopted for printing. Appearance of bitmapped images, which are made up of a fixed grid of pixels, clearly depends on the resolution chosen, unlike vector images that are scalable and then have the same appearance whatever the dimensions chosen for visualization.

Image Compression

Image compression is the process of shrinking the size of digital image files. Methods have in common the processes of finding and storing redundant data (e.g., pixels with similar color information) more efficiently or of eliminating information that is difficult for the human eye to see [8]. Compression algorithms are especially characterized by two factors: compression ratio and generational integrity. Compression ratio is the ratio of compressed image size to uncompressed size and generational integrity refers to the ability for a compression scheme to prevent or mitigate loss of data, and therefore image quality, through multiple cycles of compression and decompression. Lossless compression ensures that the image data is retained, as with Run-length encoding, Huffman coding and LZW coding. On the other hand, lossy compression schemes involve intentionally sacrificing the quality of stored images by selectively discarding pieces of data. Most of them have a compression ratio that can be parameterized by the user, to optimize the results for each situation, such as the standards JPEG and JPEG 2000.

Run-Length Encoding Run-length encoding (RLE) is probably the most simple form of lossless data compression: sequences in which the same data value occurs in many consecutive data elements are stored as a single data value and count. Image data is normally run-length encoded in a sequential process that treats the image data as a 1D stream, line by line, column by column or diagonally in a zigzag fashion. Also used in fax machines, common digital image formats for run-length encoded data include TGA, PCX and is possible with BMP, TIFF and JPEG.

Huffman Coding Created in 1951 by David A. Huffman, the Huffman coding is an entropy encoding algorithm used for lossless data compression. The basic idea of this algorithm is to code with few digits the most common input symbols of a document. Each symbol is encoded by using a variable-length code table, where the codes are defined according to the estimated probability of occurrence for each possible symbol. The technique works by creating a binary tree of nodes that contain symbols with their probability for leaf nodes and cumulated probabilities for internal nodes. Traversing this binary tree from the root to the leaves, with the convention '0' when following the left child and '1' when following the right one, allows associating a bit string with each symbol. The result is a prefix-free code: the bit string representing some particular symbol is never a prefix of the bit string representing any other symbol. Today, Huffman coding is often used during the final process of some other compression methods such as JPEG and MP3.

LZW Coding Lempel-Ziv-Welch (LZW) is a lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published in 1984 as an improved version of the LZ77 and LZ78 algorithms published by Lempel and Ziv in 1977 and 1978. The technique was patented by IBM and Unisys Corporation in 1983 and the patents expired in 2003. The compression algorithm build a string translation table that is based on fixed-length codes (usually

12-bit). As the system character-serially examines the document, if the string read is not stored in the table, a new code is created in the table and associated with this string, otherwise the current string is encoded with an existing code. This algorithm became very widely used after it became part of the GIF image format in 1987. In contrast to other compression techniques such as JPEG, it allows preserving very sharp edges, suitable with line art images often stored in GIF format.

JPEG Compression JPEG is the most common image format used for compressing and storing digital cameras and other photographic image capture devices. "JPEG" stands for Joint Photographic Experts Group, the name of the committee that created the standard. The group was organized in 1986, issuing a standard in 1992, which was approved in 1994 as ISO 10918-1 standard. The associated algorithm, described in Fig. 1, is a lossy compression technique suited for photographs and paintings of realistic scenes, but not for line drawings and other textual or graphics, where the compression cause noticeable artifacts on sharp contrasts between adjacent pixels. This algorithm stands on the representation of the image in the frequency domain by using a two-dimensional DCT (Discrete Cosine Transform), that describes the variability of the signal in terms of low-level and high-level frequencies. The human eye notices small differences in brightness over a relatively large area, but does not distinguish the exact strength of a high frequency brightness variation very well. Consequently, the amount of information in the high frequency components of the DCT can be neglected without drastically affecting perceptual image quality: the DCT components are divided by factors of a quantization matrix that increase with the spatial frequency, and then rounded to the nearest integer. This is the main lossy operation in the whole process. Typically, many of the higher frequency components are rounded to zero and the other components become small numbers, which take many fewer bits to store [10].

File Formats

There are a lot of image formats for vector graphics as well as for bitmapped images [3,6]). Most of them are open standards and patent expired for the others. Vector image formats contain a geometric description of the objects which can be rendered smoothly at any desired size. Among the most common formats, there are:

- **SVG** (Scalable Vector Graphics) is an open XML-based standard created in 1998 and developed by the World Wide Web Consortium to address the need for a versatile, scriptable and all-purpose vector format for the web and otherwise.
- **EPS** (Encapsulated PostScript) is a standard file format created by Adobe Systems in the mid-1980s. It follows DSC (Document Structuring Conventions) rules that are a set of standards for PostScript.

File formats abound for bitmapped images, but many digital imaging projects have settled on the formula of TIFF, JPEG, GIF and also PNG files:

- **TIFF**, for Tagged Image File Format, is a file format for storing images such as photographs as well as graphics. It was originally created by the company Aldus, was then under the control of Adobe Systems and is now in the public domain. TIFF supports several lossless and lossy techniques of image

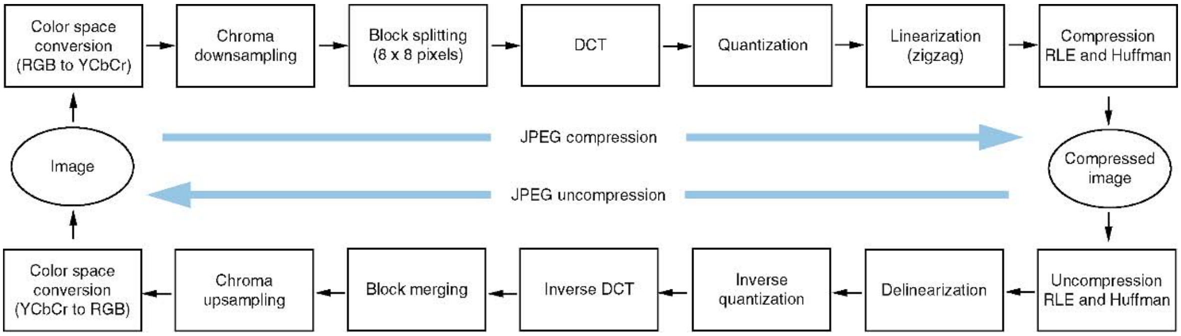


Image Representation. Figure 1. Main steps of the JPEG algorithm for compression and uncompression.

compression, such as LZW, Huffman coding and JPEG. The ability to store image data in a lossless format makes TIFF files a useful method for archiving images and preservation purposes. However, there are so many different implementations of TIFF that many applications can read certain types of TIFF images but not others.

- **JPEG**, for Joint Photographers Experts Group, is a file format that was developed specifically for high-quality compression of photographic images in a 24-bit RGB color model. It is generally employed for online presentation and dissemination; the associated lossy algorithm for compression makes it unappropriated for archiving purposes. The file format associated is JFIF (JPEG File Interchange Format, 1992), a public domain storage format for JPEG compressed images. Unlike TIFF, JFIF does not allow for the storage of associated metadata, a failing that has led to the development of SPIFF (Still Picture Interchange File Format), which is now the international standard.
- **GIF**, for Graphics Interchange Format, is an 8-bit image format for indexed colors, that was introduced by CompuServe in 1987 and has since come into widespread usage for art images such as diagrams or logos with a limited numbers of colors. The original version was called 87a; in 1989, CompuServe developed an enhanced version, called 89a, that supports animations. Due to infringement of Unisys' patent on the LZW compression technique used with GIF, in 1995 CompuServe proposed the **PNG** format (Portable Network Graphics) as a replacement for the GIF format without patent license. PNG offers a better and lossless compression technique called DEFLATE (that combines LZ77 with Huffman coding). Since 2003, it is an international standard.

Today, the status of TIFF as the de facto standard format for archival digital image files is challenged by other formats such as PNG and JPEG 2000, that are able to preserve data integrity as well as to provide efficient compression ratios for access and dissemination.

Metadata Representation

Metadata are commonly defined as "data about data." They constitute the documentation or a structured description associated with a document. Image files automatically include a certain amount of metadata that are stored in an area of the file defined by the file format and called *the header*, but information may also be stored externally.

In the widely used TIFF format, the term "Tagged" indicates that developers can define and apply dedicated tags to enable them to include their own proprietary information (called " private tags") inside a TIFF file without causing problems of compatibility. More recently, Exif format (Exchangeable image file format) was created by the Japan Electronic Industries Development Association. The latest version was published in 2002 and while the specification is not currently maintained by any industry or standards organization, its use by camera manufacturers is nearly universal. Exif fields are generated at the creation of the image and should not be modified after with the aim of including additional information like title or keywords. To do this, other formats are recommended, such as XMP (eXtensible Metadata Platform). This last is an XML-based standard for creating, processing and storing standardized, extensible and proprietary metadata, created by Adobe Systems in 2001 [1]. XMP metadata can be embedded into a significant number of popular file formats: it is used in PDF and other image formats such as JPEG, JPEG 2000, GIF, PNG, TIFF and EPS.

In parallel to generic standards, standards dedicated to specific image applications also exist, such as DICOM (Digital Imaging and Communications in Medicine). This is a standard created in 1992 and widely adopted by hospitals, for handling, storing, printing and transmitting information in medical imaging [7]. It includes a file format definition and a network communications protocol.

Metadata constitute the documentation of all aspects of digital files essential to their persistence, usefulness and access. Images without appropriate metadata may become hard to view, migrate to new technology, or to access among large volumes of images. When annotation is unappropriate or is missing, the representation of the visual content of images by image analysis may be an interesting alternative, as described in the following.

Image Content Representation

Born in early 1990s, Content-Based Image Indexing (CBIR) is a discipline that exploits techniques of image analysis and databases [2]. Indexing an image by its content consists of automatically extracting structures that describe the visual content relevantly for the considered application. These structures can describe the visual content of an image globally or locally by characterizing its distribution of color, shape and texture, or parts or objects of the image. The visual structures exhibited are considered as the index of the image, they are digitally represented with one or several multidimensional vectors called *signature* of the image. According to this description, searching for a particular image in a database of images consists of searching in multidimensional spaces.

Key Applications

JPEG 2000 is a standard that gathers an image file format and an algorithm of image compression, created by the Joint Photographic Experts Group committee in 2000 [9]. The name refers to all of the eleven parts of the standard, some of them are now published as an International Standard, while others are under development.

The coding algorithm of JPEG 2000 is similar to the JPEG one, it mainly differs in the use of wavelets instead of a DCT. Wavelets provide a decomposition of the image into a pyramid of sub-images which store different levels of resolution of the image. They can be of two types, according to the objective: (1) a Daubechies wavelet transform, that requires quantization to reduce the amount of bits representing data, as JPEG does, and then imposes lossy compression; (2) a rounded version of Le Gall wavelet transform, that uses only integer coefficients and then does not require quantization, providing lossless coding.

On average, JPEG 2000 gains up to about 20% lossy compression performance for medium compression rates in comparison to the first JPEG standard. Moreover, the edges remain sharper and more contrasting than with JPEG where blocky artifacts can also appear. The aim of this standard is not only improving compression performance but also adding features, among which transmission error resilience and region of interest (ROI). This last offers the opportunity of storing parts of the same picture using different quality. Some parts of particular interest such as faces, can be stored with higher quality, to the detriment of other ones where low quality/high compression can be tolerated. JPEG 2000 also allows delivering these parts *before* other parts of the image.

The JPEG 2000 standard defines two file formats that support embedded XML metadata: JP2, which supports simple XML, and JPX, which has a more robust XML system based on an embedded metadata initiative of the International Imaging Industry Association (the DIG35 specification). However, as of this writing, commercial implementations for JPEG 2000 are just beginning to appear. The democratization of JPEG 2000 is presently less important than the JPEG standard: It is supported in several web browsers, but is not generally used on the World Wide Web. Nevertheless, this standard will take more place in a near future, because it has been developed to efficiently deal with many applications and markets such multimedia consumers applications, military/surveillance, medical imagery, editing and storage, etc.

Future Directions

The continuing drop in prices of computers and storage, the ocean of image/video/audio data produced and the expansion of networking and communication bandwidth via the Internet or the HDTV (High-definition television) greatly contribute to the production and dissemination of larger volumes of multimedia content. Many techniques of different disciplines of computer science have been studied and developed for managing text, image, video and audio. But today, there is a need in the development of an unified framework for the creation, representation, storage, access, delivery, management and protection of multimedia contents. Standardization goes in this direction by proposing new standards for managing these contents jointly, such as the international standards MPEG-21 and MPEG-7. MPEG-21 is a standard started in 1999 by MPEG (Moving Picture Experts Group) and now normalized as ISO/IEC 21000. Its main objectives are to define an open framework for multimedia applications and more precisely to provide a standardized structure for various media contents and to facilitate their access, delivery, management and protection [4]. MPEG-7 is another ISO/IEC standard started by MPEG in 1998 and formally called *Multimedia Content Description Interface*. It aims at specifying a standard set of descriptors and description schemes dedicated to various types of multimedia information. One of its main objectives is to provide unified and efficient searching, filtering and content identification methods for these media [5].

Cross-references

- ▶ Feature Extraction for Content-Based Image Retrieval
- ▶ Image
- ▶ Image Content Modeling
- ▶ Image Metadata
- ▶ Image Salient Points and Features

- ▶ Image Segmentation
- ▶ Image Segmentation and Features
- ▶ Metadata
- ▶ Multimedia Data
- ▶ Multimedia Data Storage
- ▶ Multimedia Databases
- ▶ Multimedia Metadata
- ▶ Video
- ▶ Video Representation
- ▶ Visual Content Analysis
- ▶ Visual Representation

Recommended Reading

1. Adobe XMP main page: http://www.adobe.com/products/xmp/index.html. 5
2. Datta R. Joshi D. Li J. and Wang J. Z. Image retrieval: Ideas, influences, and trends of the new age. ACM Comput. Surv. 40 (2), 2008.
3. Fileformat.info. http://www.fileformat.info/format/could.htm. 3
4. Ian S., Burnett, Pereira F., Van de Walle R., and Koenen R. The MPEG-21 Book. Wiley, 2006.
5. Manjunath B.S., Salembier P., and Sikora T. Introduction to MPEG-7: Multimedia Content Description Interface. Wiley & Sons, 2002.
6. Murray J. D. and vanRyper W. Encyclopedia of Graphics File Formats. 2nd ed. O'Reilly, 1996.
7. Oleg S., Pianykh. Digital Imaging and Communications in Medicine (DICOM): A Practical Introduction and Survival Guide. Springer, 2008.
8. Salomon D., Motta G., and Bryant D. Data Compression: The Complete Reference. Springer, 4th edition, 2006.
9. Taubman D. S. and Marcellin M. W. JPEG 2000: Image Compression Fundamentals, Standards and Practice. Kluwer International Series in Engineering and Computer Science, Secs 642, 52001.
10. Wallace G. K. The JPEG still picture compressin standard. Commun. ACM, 34(4):30–44, 3 April 1991.

Image Similarity

TAO MEI[1], YONG RUI[2]
[1]Microsoft Research Asia, Beijing, China
[2]Microsoft China R&D Group, Redmond, WA, USA

Synonyms

Image Distance; Visual Similarity; Similarity Measure

Definition

Given a pair of images each described by a feature set, image similarity is defined by comparing the feature set on the basis of a similarity function. In a typical Visual Information Retrieval system, while searching for a query image among the elements of the data set of images, knowledge of the domain will be expressed by formulating a similarity measure between the query and data set based on some visual features. Therefore, measuring meaningful image similarity consists of two intrinsic elements: finding a set of features for adequately describing the image content and finding a suitable metric for assessing the similarity on the basis of feature space. The feature set can be computed globally for the entire image or locally for a small group of pixels such as regions or objects. The similarity measure can be different depending on the types of features. Typically, the feature space is assumed to be Euclidean. The algorithm for image similarity aims to essentially reduce the semantic gap between low-level features and high level semantics as much as possible.

Historical Background

Comparing two images is the fundamental operation for many Visual Information Retrieval systems, in which the user selects a query image and image similarity to the query according to the given criteria are retrieved and presented [11]. Figure 1 shows the role of image similarity in the context of multimedia information retrieval. A set of visual features is first computed for each image or video frame in the database and for the query. Then, given a query image, image similarity is computed for each pair of query and image based on the feature space and some distance metric, and may be further tuned in an interactive way according to user feedback. The images which are visually, semantically, or perceptually similar to the query are finally presented to the user.

A wide variety of methods for image similarity have been devised with research expanding in content-based multimedia retrieval in the last decade. In earlier work, image content is usually described by a set of global features such as color, texture, shape, and so on [8]. The advantage of global features is the high speed for both feature extraction and similarity computation. The features are then transformed and represented by a set of vectors. Image similarity is computed based on different distance metrics on these feature vectors. For example, color histogram serves as an effective representation of the distribution of colors in an image. A color histogram is created by plotting the number of pixels in the image

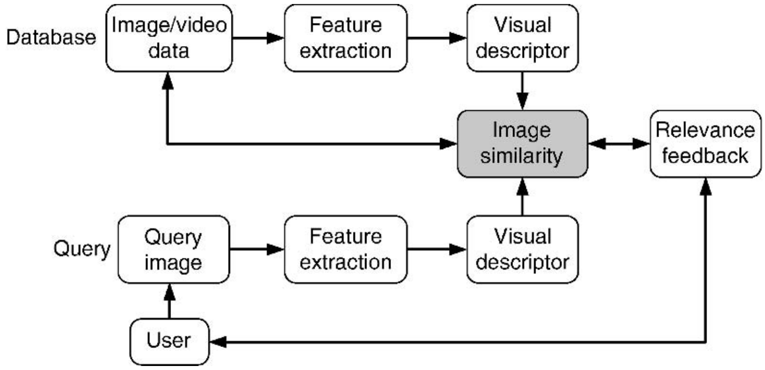


Image Similarity. Figure 1. Image similarity in the context of multimedia information retrieval.

with a particular range of quantized color value in some color space. The Minkowski-form distance or Histogram intersection can be used to compute the similarity between two histograms [5]. However, using an individual type of feature cannot well characterize image content. The multimodal methods combine the similarities from different types of features. The simplest way for combination yielding a scalar is to compute the weighted sum of these similarities based on the assumption that the features are not of the same importance. The weights can be decided manually or by relevance feedback mechanism [9].

As global features are often too rigid to represent an image, there has been a paradigm shift from global feature representation to local descriptors. Local features are computed based on a subset of the image usually in neighborhood of a salient point or a region. It is deemed that local features are closer to the perception of human visual system, as they often correspond to meaningful image components such as salient points, rigid objects, and homogenous regions [2]. The similarity is then computed based on user selected points/objects/regions or weighted sum of all of them. For example, an image is represented by a set of regions where each region is assigned a feature set and a weight indicating the importance of the corresponding region [4]. The image-to-image similarity is measured by the Earth Mover's Distance (EMD) [7] or an Integrated Region Matching method (IRM) [2]. The recent work in computer vision treated image as a set of salient points and extracted Scale Invariant Feature Transform (SIFT) features for each point [6]. Image matching is performed based on the Euclidean distances between pairs of salient points or the histogram comparison built upon a codebook.

Foundations

By the nature of the task of image retrieval, image similarity boils down to two intrinsic problems: (i) how to describe an image using a set of visual features, and (ii) how to assess the similarity between two images based on these features. Figure 2 shows the paradigms of image similarities, including three types of features for describing image content, formulation of the features, distance metrics for computing the similarities, and techniques used for computing the distance. The distance metrics used for image similarity depend on the selection of features and their corresponding formulations.

Image Descriptors

The preliminary step for image similarity is the description of image content. Research has proceeded toward effectively characterizing image content by a variety of visual features (or referred to as *signatures*). These features can be categorized into three types according to the pixels used, i.e., global, regional, and local features. The global features are extracted over the entire image or sub-image based on grid partition, the regional features are computed based on the results of image segmentation which attempts to segment the image into different homogenous regions or objects, while the local features aim at robust descriptors invariant to scale and orientation based on local maxima.

In global features, the most typical ones are color, texture, and shape which are widely used in many retrieval systems [5]. Color histogram is an effective and easy-to-compute representation of the color distribution in an image. It is also robust to the translation and rotation about the view axis, as well as slight occlusion. A color histogram is created by counting the

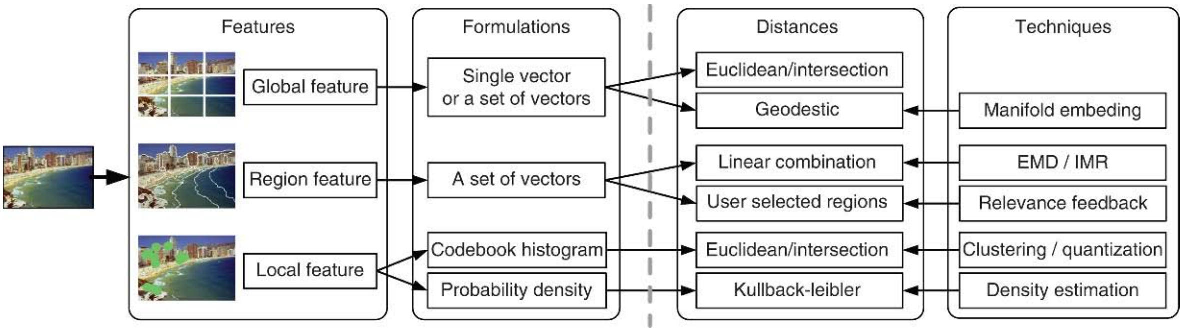


Image Similarity. Figure 2. Paradigms of image similarities, including the corresponding features, formulations, and distance metrics, and techniques. This figure is designed similar to the figure used in [5].

pixels falling into a quantized bin in a specific color space, such as RGB, HSV, and LUV spaces. In addition, color moments are also popular color features. Usually the first three order (i.e., *mean*, *variance*, and *skewness*) color moments are suitable enough to represent the color distribution of image. An alternative way to compute color histogram and moments is to divide an image into non-overlapping blocks, compute histogram or moments for each block, and concatenate all the features into one vector. To deal with large-scale image database, a compact descriptor is proposed in [12] where each image is converted to a K-bit (K is no more than 32) hash code according to its content. Other color features include color coherence, color correlogram, and so on. Texture features are intended to capture the granularity and repetitive visual patterns in an image. The basic texture features include Tamura, multi-resolution simultaneous auto-regressive model (MRSAR), Gabor filter, wavelet transform, and Wold features [11,5]. In contrast to color and texture features, shape features are usually computed after images being segmented into regions. The shape features capture the local geometrical properties. Typical shape features include normalized inertia [1], moment invariants, Fourier descriptors [5], and so on. Using global features, an image can be represented by a single vector corresponding to a uni-modaltiy or a set of vectors and weights corresponding to multi-modal features and their importances.

The regional features are similar with global features, except that they are computed over a local region with homogeneous texture rather than the whole image. The most widely used features for describing a region include color moment [1], color correlogram [8], wavelet transform texture, and normalized inertia [1]. As a result, an image is represented by a set of vectors and weights each corresponding to a region.

Local invariants such as salient points from which descriptors are derived, traditionally used for stereo matching and object recognition, are being used in image similarity. For example, the algorithm proposed by Lowe [6] constructs a scale space pyramid using Difference-of-Gaussian (DoG) filters and finds the local 3D maxima (i.e., salient point) on the pyramid. A robust Scale Invariant Feature Transform (SIFT) descriptor is computed for each point. An image is thus represented by a set of salient points and 128 dimensional SIFT features, or a histogram of codewords built upon a large visual vocabulary.

Similarity Measures

The similarity measures are based on comparisons between the features associated with images. As shown in Fig. 2, since each type of feature can have its own mathematical formulation, the distance metrics and the corresponding techniques for computing the distances are different.

Considering that an image $\mathbf{I}$ is represented by a single vector $\mathbf{f} = (f(1),\ldots,f(M))$ (e.g., histogram or some distribution) in which each dimension is independent of each other and has equal importance, the most widely adopted similarity is computed based on Minkowski-form distance, defined as

$$\mathbf{D}_{L_p}(\mathbf{I}^0,\mathbf{I}^1) = \left\{ \sum_{i=1}^{M} |f^0(i) - f^1(i)|^p \right\}^{1/p} \qquad (1)$$

where $f^0(i)$ and $f^1(i)$ denote i-th feature of image $\mathbf{I}^0$ and $\mathbf{I}^1$, respectively. For $p = 2$, this yields the Euclidean distance. The histogram intersection is a special case of L_1 distance, defined as

$$\mathbf{D}_{HI}(\mathbf{I}^0,\mathbf{I}^1) = \frac{\sum_{i=1}^{M} \min(f^0(i), f^1(i))}{\sum_{i=1}^{M} f^1(i)} \qquad (2)$$

It has been shown that histogram intersection is fairly sensitive to the changes of image resolution, occlusion, and viewing point [5]. A distance robust to noise is Jeffery distance (JD) which is based on the Kullback-Leibler divergence (KL) given by

$$\mathbf{D}_{KL}(\mathbf{I}^0,\mathbf{I}^1)=\sum_{i=1}^{M}f^0(i)\log\frac{f^0(i)}{f^1(i)} \tag{3}$$

Although it is often intuited as a distance metric, the KL divergence is not a true metric since it is not symmetric. The JD distance is defined as

$$\begin{aligned}\mathbf{D}_{JD}(\mathbf{I}^0,\mathbf{I}^1)=\sum_{i=1}^{M}\{&f^0(i)\log\frac{f^0(i)}{m_i}\\&+f^1(i)\log\frac{f^1(i)}{m_i}\}\end{aligned} \tag{4}$$

where $m_i=\frac{f^0(i)+f^1(i)}{2}$. In contrast to KL divergence, JD distance is symmetric and numerically stable when comparing two empirical distributions. Hausdorff distance [2] is another matching method which is symmetrized by computing additionally the distance with image $\mathbf{I}^0$ and $\mathbf{I}^1$ reversed and choosing the larger one of the two distances

$$\begin{aligned}&\mathbf{D}_H(\mathbf{I}^0,\mathbf{I}^1)=\max\left(\max_i\min_j\ \mathbf{d}(f^0(i),f^1(i)),\right.\\&\left.\max_J\min_i\ \mathbf{d}(f^0(i),f^1(i))\right)\end{aligned} \tag{5}$$

where $\mathbf{d}(,)$ can be any form of distance such as L_1 and L_2 distances. The Mahalanobis distance metric deals with the case that each dimension of vector is dependent and has different importance, given by

$$\mathbf{D}_M(\mathbf{I}^0,\mathbf{I}^1)=\sqrt{(\mathbf{f}^0-\mathbf{f}^1)^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{f}^0-\mathbf{f}^1)} \tag{6}$$

where $\mathbf{C}$ is the covariance matrix of the feature vectors. The above distance measures are all derived from a linear feature space which has been noted as the difficulty in measuring perceptual or semantic image distance. Manifold ranking replaces traditional Euclidean distance by the geodesic distance in a non-linear manifold [13]. The similarity is often estimated based on a distance measure $\mathbf{d}(,)$ and a positive radius parameter σ along a manifold

$$\mathbf{D}_{MR}(\mathbf{I}^0,\mathbf{I}^1)=\exp\left\{-\frac{\mathbf{d}(\mathbf{I}^0,\mathbf{I}^1)}{\sigma}\right\} \tag{7}$$

where L_1 distance is usually selected for $\mathbf{d}(,)$.

In a more general situation, an image $\mathbf{I}$ is represented by a set of vectors and weights $\{(\mathbf{f}_i,\omega_i)\}(i=1, 2,\ldots,M)$, where M is the number of modalities (e.g., color, texture, or shape). Each vector (also referred to as distribution) corresponds to a specific region or modality and the weight indicates the significance of associating this vector to the others. Note that the weight ω_i can have the same dimension to $\mathbf{f}_i$ indicating that each dimension of the features in a single vector has the same importance to each other, or just a real value indicating the importance of the entire vector. The simplest way to compute the similarity from different modalities is the weighted sum of the similarity from each single vector. The Earth Mover's Distance (EMD) represents a soft matching scheme for features in the form of set of vectors [7]. The EMD "lifts" the distance from individual features to full distributions. The EMD distance is given by

$$\mathbf{D}_{EMD}(\mathbf{I}^0,\mathbf{I}^1)=\frac{\sum_{i=1}^{M^0}\sum_{j=1}^{M^1}s_{ij}\mathbf{d}(\mathbf{f}_i^0,\mathbf{f}_j^1)}{\sum_{i=1}^{M^0}\sum_{j=1}^{M^1}s_{ij}} \tag{8}$$

where $\mathbf{d}(,)$ is the ground distance between two vectors which can be defined in diverse ways depending on the system, s_{ij} minimizes the value of (8) subject to the following constraints:

$$\begin{aligned}&s_{ij}\geq 0,\quad 1\leq i\leq M^0,\quad 1\leq j\leq M^1\\&\sum_{j=1}^{M^0}s_{ij}\leq\omega_i^0,\quad 1\leq i\leq M^0\\&\sum_{i=1}^{M^1}s_{ij}\leq\omega_j^0,\quad 1\leq j\leq M^1\\&\sum_{i=1}^{M^0}\sum_{j=1}^{M^1}s_{ij}=\min\left(\sum_{i=1}^{M^0}\omega_i^0,\sum_{j=1}^{M^1}\omega_j^1\right)\end{aligned}$$

when ω_i^0 and ω_j^1 are probabilities, EMD is equivalent to the Mallows distance [2]. Another matching-based distance is the Integrated Region Matching (IRM) distance [2]. The IRM distance uses the most similar highest priority (MSHP) principle to match different modalities or regions. The weights s_{ij} are subject to the same constraints as in the Mallows distance, except that $\mathbf{d}(,)$ is not computed by minimization. Another way to the adjustment of weights ω_i in image similarity is relevance feedback which captures the user's precise needs through iterative feedback and query refinement. The goal of relevance feedback is to find the appropriate weights to model the user's information need [9]. The weights are classified into *intra*- and *inter*- weights. The intra-weights represent the different contributions of the components within a single vector (i.e., region or modality), while the inter-weights represent the contributions of different vectors. Intuitively, the intra-weights are decided based on the variance of the same vector components in the relevant feedback examples, while the inter-weights are directly updated according to user's feedback in terms of the similarity based on each vector. For the comparison among these distance metrics, please refer to [2] for more details.

In the context of image being represented by a set of salient points and their corresponding local descriptors, image similarity is computed based on the Euclidean distance between each pair of salient points [6]. An alternative way is to represent each image by a bag of codewords which are obtained by unsupervised learning of local appearance [3]. A large vocabulary of codewords is built by clustering a large amount of local features, and then the distribution of these codewords is obtained by counting all the points or patches within an image. Since the image is described by a set of distributions, histogram intersection and EMD defined in (2) and (8) can be employed to compute the similarity.

Key Applications

Content-based Multimedia Information Retrieval/Multimedia Database

The computation of image similarity is the fundamental operation in content-based multimedia information retrieval systems and multimedia database. Given a query image or video, the images or videos reasonably similar to the query are returned based on the given features and distance metric.

Object Recognition

Object Recognition aims to identify an object in a database of images. Typically an object is represented by a set of overlapping regions each represented by a vector computed from the region's appearance. Recognition of a particular object proceeds by matching the descriptor vectors based on image similarity.

Medical/Satellite/Surveillance Applications

In the applications such as medical, satellite image, and surveillance, image similarity is usually used for managing the database and querying or recognizing a particular object. For example, image similarity can be employed to detect the regions with abnormal characteristics in medical diagnose. Moreover, image similarity can support content-based queries on large database of remote sensing images.

Future Directions

As pointed out in [2], the problem of image similarity is the reliance on visual similarity for judging semantic similarity. As directly applying distance metrics to image similarity cannot well model human similarity perception, automatic learning of image similarity with the help of contextual and human information has been explored. For example, when an image is conceived as a bag of instances which correspond to regions, multiple-instance learning (MIL) can be used for learning semantic similarity [1]. Another interesting problem concerns image similarity in human perception system [10]. The mathematical or computational models are needed to accurately assess perceptual similarity by resembling human's perception.

Cross-references

- ▶ Image Database
- ▶ Image Retrieval
- ▶ Similarity in Video
- ▶ Video Retrieval

Recommended Reading

1. Chen Y. and Wang J.Z. Image categorization by learning and reasoning with regions. J. Machine Learn. Res., 5:913–939, 2004.
2. Datta R., Joshi D., Li J., and Wang J.Z. Image retrieval: ideas, influences, and trends of the new age. ACM Comput. Surv., 40(65), 2008.
3. Fei-Fei L. and Perona P. A bayesian hierarchical model for learning natural scene categories. In Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition, 2005, pp. 524–531.
4. Jing F., Li M., Zhang H.-J., and Zhang B. An efficient and effective region-based image retrieval framework. IEEE Trans. Image Process., 13(5):699–709, May 2004.
5. Long F., Zhang H.-J., and Feng D.D. Fundamentals of Content-based Image Retrieval. Chapter in: Multimedia Information Retrieval and Management – Technological Fundamentals and Applications. D.D. Feng, W.C. Siu, H.-J. Zhang (ed.). Springer, Berlin Heidelberg New York, January 2003.
6. Lowe D.G. Distinctive image features from scale-invariant keypoints. Int. J. Computer Vis., 60(2):91–110, 2004.
7. Rubner Y., Tomasi C., and Guibas L.J. The earth mover's distance as a metric for image retrieval. Int. J. Computer Vis., 40 (2):99–121, 2000.

8. Rui Y., Huang T.S., and Chang S.-F. Image retrieval: current techniques, promising directions and open issues. J. Vis. Commun. Image Rep., 13(10):39–62, 1999.
9. Rui Y., Huang T.S., Ortega M., and Mehrotra S. Relevance feedback: a power tool for interactive content-based image retrieval. IEEE Trans. Circ. Video Tech., 8(5):644–655, September 1998.
10. Santini S. and Jain R. Similarity measures. IEEE Trans. Patt. Analy. Machine Intell., 21(9):871–883, September 1999.
11. Smeulders A.W.M., Worring M., Santini S., Gupta A., and Jain R. Content-based image retrieval at the end of the early years. IEEE Trans. Patt. Anal. Machine Intell., 22(12):1349–1380, 2000.
12. Wang B., Li Z., Li M., and Ma W.-Y. Large-scale duplicate detection for web image search. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2006, pp. 353–356.
13. Zhou D., Bousquet O., Lal T., Weston J., and Scholkopf B. Learning with local and global consistency. In Proc. Advances in Neural Information Processing System, 2003, pp. 321–328.

Image Retrieval

► Image Database

Image Retrieval and Relevance Feedback

Michel Crucianu
National Conservatory of Arts and Crafts, Paris, France

Definition

Relevance feedback is a means of refining a query in an information retrieval system by asking the user to specify how relevant each result of the query is. An image retrieval session relying on relevance feedback is interactive and iterative. The session is divided into several consecutive rounds. At every round, the user provides feedback regarding the current retrieval results, usually by qualifying the returned images as either "relevant" or "irrelevant"; from this feedback, the system attempts to better identify the target of the user and to return improved results. A relevance feedback mechanism must maximize the relevance of the results while minimizing the amount of interaction between the user and the system.

Historical Background

In the early years of content-based image retrieval (CBIR), query by visual example (QBVE) was a prevailing paradigm. To support QBVE, an image retrieval system must first extract, during an off-line phase, a description in terms of low-level features (e.g., distribution of colors, textures, etc.) from every image in the supplied database. Then, when a query image is provided, the system returns the images whose descriptions are the most similar to the description of this query. However, QBVE encountered important difficulties: (i) there is a *semantic gap* between the high-level concept defining what a user is searching for and the existing low-level descriptions, and (ii) a relevant image is seldom available to serve as initial query ("page zero" problem). The introduction of relevance feedback in CBIR was motivated by both difficulties. User feedback provided during consecutive rounds can allow the system to progressively learn a correspondence between the target concept and low-level descriptions, thus bridging the semantic gap. Also, since the user can amend the results at every round, the initial query is less important; many different starting images may allow to identify the target concept.

Relevance feedback is not specific to image retrieval. For other types of content, such as text, music or video, a gap can also be found between automatically extracted descriptions of the content and search criteria that users consider meaningful. Actually, relevance feedback was first introduced for the retrieval of text documents in the seminal work of Salton and van Rijsbergen. These initial proposals inspired the early feedback methods put forward for image retrieval. However, since a user can evaluate the relevance of an image faster than the relevance of a text or of a streaming media item, the appeal of relevance feedback was stronger for image retrieval.

Foundations

Understanding the specific goal of an image retrieval session is important both for defining an appropriate feedback mechanism and for identifying adequate evaluation methods. The earliest and most frequent goal involving relevance feedback consists of finding images that illustrate a target concept the user has in mind [6,10]. The concept can correspond to a perceptive characteristic, but in general has higher-level semantics. The system must solve a ranking problem: the images must be ordered and returned to the user by decreasing relevance. Nevertheless it is accepted that a precise ranking of the relevant images is not required, mainly because the user herself may be unable to define an optimal ranking; the system should simply rank most of the relevant images before the irrelevant ones.

Recently, relevance feedback was also suggested as a way to delineate the class of images illustrating a concept the user has in mind. The aim is to extend textual annotations of some images in the class to the others [7] or to help librarians perform "mass annotation" of the images [3]. To make this procedure efficient, much less effort should be required for defining the target class than for individually annotating all the images in the class. In this case, the system has to solve a classification problem: the boundary between the images belonging to the target class (the relevant images) and the others (irrelevant images) must be reliably identified. The rate of false positives (i.e., images that do not belong to the class but are assigned to it and, consequently, receive a wrong annotation) is considered to be more important than the overall error rate. However, the boundary does not have to be crisp and a degree of confidence can be associated to the resulting annotation.

Image retrieval with relevance feedback can only be effective if some important assumptions are verified. First, discrimination between relevant and irrelevant images must be possible with the available image descriptors; the "numerical gap," due to the use of inappropriate descriptors, should be avoided. Second, the target concept of the user must be consistent during the consecutive feedback rounds of a retrieval session. Third, since the amount of feedback a user can provide is very limited, the target concept that needs to be identified must have a relatively simple representation in the description space of the images.

Relevance Feedback Methods

To support relevance feedback, an image retrieval system should include two components: a *learner* and a *selector*. At every feedback round, the user marks (or labels) images returned by the system as either relevant or irrelevant. The learner makes use of this information to reestimate the target of the user (see Fig. 1). Given the current estimation of the target, the selector chooses other images for which the user is asked to provide feedback during the next round. The recent evolution of the learners and of the selection criteria is briefly presented in the following.

Learners

To estimate the target of the user, the learner can rely on the training data, consisting of the images marked by the user during consecutive feedback rounds, and on prior knowledge when it is available. The task of the learner is particularly difficult in the context of relevance feedback for several reasons (see also [16]). First, since the interaction with the user during a feedback session is limited, the amount of training data is small, sometimes much smaller than the dimension of the image description space. This highlights the importance of prior knowledge. Then, the target class may have a rather complex shape in the description space and even several disconnected modes; since training data is scarce, this can severely limit the generalization expected from the learner. Another difficulty comes from the strong imbalance in the training data: there are usually fewer positive examples (images considered relevant by the user) than negative examples (irrelevant images). The learner should have a low sensitivity to this imbalance or some remedy must be

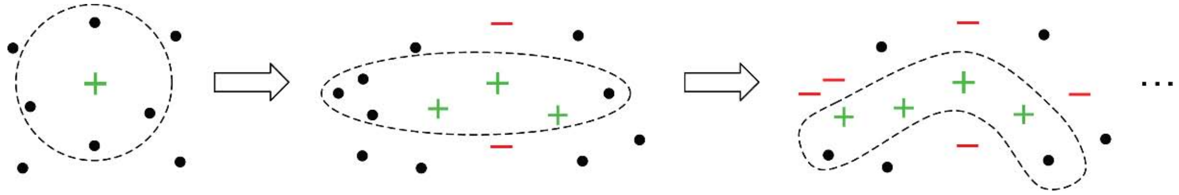


Image Retrieval and Relevance Feedback. Figure 1. Simplified view of some relevance feedback rounds. An initial similarity-based retrieval (leftmost image) returns the four nearest neighbors (points inside the dotted circle) of the query (+), relying on the default metric defined in the image description space. By marking the images returned as either relevant (+) or irrelevant (–), during consecutive rounds, the user allows the system to progressively identify the target class. At every round, the dotted boundary shows the current estimation of the target.

found. Last but not least, to preserve interactivity, both learning from the training data and the evaluation of the remaining images according to the selection criterion should be fast.

Early work on relevance feedback for CBIR directly followed from QBVE, by assuming the existence of an ideal query point and feature weighting that, if found, would allow QBVE to provide the appropriate answer to the user. This motivated the "query point movement" (QPM) approach, for which the task of the learner consists in identifying, at every round, a better query point together with a re-weighting of the individual dimensions of the image description space. Learning sometimes used only the positive examples [14], but usually employed both the positive and the negative examples [12]. These early proposals make the strong assumption that the target class corresponds to a multidimensional Gaussian distribution and some further consider that the covariance matrix of the target class is diagonal. Learning consists then in estimating the parameters of this distribution. This restrictive assumption was removed in [8,11], as a first departure from QPM. In the query expansion scheme put forward in [11], an online clustering of the examples is performed first, then all the other images are evaluated using a nearest-neighbor decision with respect to these clusters. The densities of the positive and of the negative examples are estimated with a Parzen window method in [8]; the images returned to the user are ranked according to the difference between the two densities.

For a large part of the recent work on relevance feedback, the learners employed are kernel methods and especially support vector machines (SVM [13]). With these kernel methods, the data in input space is first mapped to a higher-dimensional feature space using a non-linear transform associated to a reproducing kernel; linear methods in this feature space provide nonlinear solutions in the input space. Learning is usually based on constrained quadratic optimization and the sparsity of the solution is encouraged by various means. Two-class SVM maximize the *margin* between the discrimination boundary and the training data belonging to each class, so the resulting boundary is only defined by those examples that are closest to the boundary (the *support vectors*). One-class SVM model the support of the distribution of data in input space by finding the smallest sphere surrounding the data in feature space (according to one formulation); this sphere is only determined by the outermost examples. Since it is defined in the feature space, a kernel method can be easily applied to different types of data (e.g., vectors, sets, graphs) if an appropriate kernel is found. An SVM has some other properties that make it an interesting choice as a learner for relevance feedback. First, the decision function of an SVM allows both the definition of a boundary and of a ranking. Second, learning with few examples is very fast and, given the sparsity of the solution, computing the value of the decision function for the unmarked images can also be relatively fast. Finally, by relying only on support vectors, two-class SVM are usually less sensitive than density-based learners to the imbalance between positive and negative examples in the training data.

Among kernel methods, recent relevance feedback proposals often make use of two-class SVM to discriminate relevant and irrelevant images (see e.g., [15]). But one-class SVM were also used to model the distribution of the relevant images alone (see [1]). Other kernel methods, such as kernel biased discriminant analysis, Bayes point machines and relevance vector machines, were successfully employed in CBIR with relevance feedback.

Given the small amount of labeled data provided to the learner by user feedback, *semi-supervised* learning received significant attention in this context. Semi-supervised learning attempts to make use of both labeled and unlabeled data in order to find a better model (such as a discrimination boundary or a ranking function). It relies on the assumption that the distribution of unlabeled data does provide valuable information regarding the sought model (e.g., the density of data is lower near the boundary between classes than inside each class). When applied to relevance feedback, this approach has to face two important difficulties. First, the underlying assumption is not necessarily true for a particular image database and target class, and cannot be verified *a priori*. Second, taking unlabeled data into account frequently produces an increase in the computational complexity, which may not be compatible with interactive search.

Selection Criteria

After the learner reestimates the target of the user from the feedback received the selector must provide the user with a new set of images. The ultimate goal of a retrieval session is to present the user with as many relevant images as possible. Accordingly, the earliest and most frequently employed selection criterion returns those unmarked images that are considered to be the most relevant given the current estimation of user's target. The short description for this criterion is that it returns the "most positive" (MP) images. It has the advantage that the user can receive quite early during the retrieval session several relevant images.

However, to reach optimal results with a minimal amount of interaction, the system should elicit from the user at every round as much information as possible regarding the distinction between relevant and irrelevant images. Existing results concerning *active* learning point out that this is achieved when the examples shown to the user are those expected to remove a maximal amount of uncertainty regarding the target. This selection criterion can be briefly described as returning the "most informative" images (MI). It translates into two complementary conditions for the images being selected: each of them should be ambiguous (given the current estimation of the target class) and any two images should be as dissimilar as possible. A selection criterion based on active learning was first proposed for image retrieval with relevance feedback in [2], for a learner using density estimation. The ambiguousness condition was considered in [15] for SVM learners: the most ambiguous images (MA) are those that are nearest to the current boundary between relevant and irrelevant images. The dissimilarity (or low redundancy) condition was added later (see e.g., [3]).

Temporal Structure of the Session

The MI selection criterion was shown to minimize the number of feedback rounds required for defining the target class of the user. But since MI returns ambiguous images, it is inappropriate for presenting the user a maximum of relevant images. A first solution to this problem was to use MI during several rounds in order to define the class and then switch to MP in order to show the most relevant images; it is nevertheless difficult to know when exactly to replace MI by MP. Another solution is to combine the results of both MI and MP in the images shown to the user at every feedback round; this alternative is better adapted to nonprofessional users, who expect more immediate reward.

The initialization of search can also have an impact on the temporal structure of a retrieval session. In some cases, a relevant starting image can be provided by the user (external query) or found with the help of a visual summary of the image database. If an appropriate starting image is not available, then feedback should be used both for finding a truly relevant image (exploration stage) and then for retrieving further relevant images (exploitation stage). The session begins with some random selection of images. During the exploration stage, the system must be able so show the user images that are increasingly relevant. The user may not see any truly relevant image during several rounds, so she is expected to indicate which among the images returned by the system are more relevant than the others. Support for such an exploration stage also helps when the target class has several distinct modes. While the exploratory behavior was addressed by [2] and, to some extent, by a few QPM proposals, later work mainly focused on the exploitation stage. A competitive method for the exploration stage was recently proposed in [4].

Evaluation of Relevance Feedback in Image Retrieval

User satisfaction is the ultimate measure of the success of image retrieval with relevance feedback. Reliable evaluations or comparisons between alternative relevance feedback methods require large groups of users and real world image retrieval problems. Given the difficulty of setting up large scale experiments with real users, most evaluations are actually performed on specific ground truth databases, by emulating the user. A ground truth usually corresponds to the definition of a set of mutually exclusive image classes, covering an entire database. The emulated user knows the ground truth and is assumed to have a stoic and error free behavior: at every feedback round the emulated user correctly marks as either relevant or irrelevant each of the images returned by the selector. To obtain a reliable evaluation, it is very important to employ several ground truth databases having dissimilar characteristics. It was also argued that more diverse and realistic behaviors should be assigned to the emulated user.

When the aim is to rank the relevant images before the irrelevant ones, the quality of retrieval is usually given by the proportion of relevant images in the top N returned by an MP selector; N is the number of images in the target class of the ground truth database. When the aim is to delineate the target class, the quality of discrimination is $1 - \varepsilon$, where ε is either the overall classification error or the rate of false positives. Usually, a "right" number of feedback rounds can not be fixed *a priori.* To evaluate the performance of a relevance feedback method, the appropriate quality measure should be recorded during several consecutive feedback rounds.

Key Applications

Interactive multimedia search engines are the original motivation for content-based image retrieval in general and for the introduction of relevance feedback in particular. Such search engines are of high interest both for the general public and for professional users. Scalability and user-friendliness are fundamental requirements in this context.

Assistance in the annotation of content can be an important application mainly directed to professional users. Relevance feedback can support such users in delineating large classes of images in order to annotate at once all the images belonging to a class (mass annotation). In this case, the ability to reach a low classification error or a low rate of false positives is the major concern.

Future Directions

Since the amount of feedback provided during a retrieval session is very small, the system should make the most of all the information sources potentially available. These sources concern the users (e.g., past sessions of other users, user profiles), the images (e.g., structured or unstructured metadata) and the context of retrieval. The system should be able to evaluate and integrate all these sources with the feedback directly provided by the current user.

An important issue that was not extensively studied is the scalability of relevance feedback to very large databases. Scalability is a challenge both for the learner (especially when semi-supervised methods are employed) and for the selector. To avoid evaluating the decision function for all the unmarked images in the database, an index structure is needed. But existing multidimensional or metric index structures and associated *k*NN retrieval methods can not be directly applied, mainly because the queries that have to be processed are not classical point queries. A good example is the use of the MA (or MI) selection criterion with an SVM learner: in this case the images that are nearest to the discrimination boundary (defined by a hyperplane in feature space) should be returned. This scalability issue is addressed by some recent proposals (see [5,9]).

The interaction between users and current systems is rather limited; typically, a user can only mark as relevant or irrelevant each of the shown images. A more advanced interface should bring in more flexibility, by allowing every user to group together or to separate images, to place them in a visual summary, to mix several image retrieval paradigms, to interact online with other users, etc.

Cross-references

▶ Feature Extraction for Content-Based Image Retrieval

▶ Image Retrieval

▶ Relevance Feedback

Recommended Reading

1. Chen Y., Zhou X.S., and Huang T.S. One-class SVM for learning in image retrieval. In Proc. Int. Conf. Image Processing, 2001, pp. 34–37.
2. Cox I.J., Miller M.L., Omohundro S.M., and Yianilos P.N. An optimized interaction strategy for Bayesian relevance feedback. In Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition, 1998, pp. 553–558.
3. Ferecatu M., Crucianu M., and Boujemaa N. Retrieval of difficult image classes using SVM-based relevance feedback. In Proc. 6th ACM SIGMM International Workshop on Multimedia Information Retrieval, 2004, pp. 23–30.
4. Ferecatu M. and Geman D. Interactive search for image categories by mental matching. In Proc. 11th IEEE Conf. Computer Vision, 2007, pp. 1–8.
5. Heisterkamp D.R. and Peng J. Kernel VA-files for relevance feedback retrieval. In Proc. 1st ACM Int. Workshop on Multimedia Databases, 2003, pp. 48–54.
6. Kurita T. and Kato T. Learning of personal visual impression for image database systems. In Proc. 2nd Int. Conf. Document Analysis and Recognition, 1993, pp. 547–552.
7. Lu Y., Hu C., Zhu X., Zhang H.-J., and Yang Q. A unified framework for semantics and feature based relevance feedback in image retrieval systems. In Proc. 8th ACM Int. Conf. on Multimedia, 2000, pp. 31–37.
8. Meilhac C. and Nastar C. Relevance feedback and category search in image databases. In Proc. Int. Conf. on Multimedia Computing and Systems, 1999, pp. 512–517.
9. Panda N., Goh K.-S., and Chang E.Y. Active learning in very large databases. Multimedia Tools Applicat., 31(3):249–267, 2006.
10. Picard R.W., Minka T.P., and Szummer M. Modeling user subjectivity in image libraries. In Proc. Int. Conf. Image Processing, 1996, pp. 777–780.
11. Porkaew K. and Chakrabarti K. Query refinement for multimedia similarity retrieval in MARS. In Proc. 7th ACM Int. Conf. on Multimedia (Part 1), 1999, pp. 235–238.
12. Rui Y., Huang T.S., Ortega M., and Mehrotra S. Relevance feedback: a power tool in interactive content-based image retrieval. IEEE Trans. Circuits Syst. Video Tech., 8(5):644–655, 1998.
13. Schölkopf B. and Smola A. Learning with Kernels. MIT, Cambridge, MA, 2002.

14. Sclaroff S., Taycher L., and Cascia M.L. Image Rover: a content-based image browser for the world wide web. In Proc. Workshop on Content-Based Access of Image and Video Libraries, 1997, pp. 2–9.
15. Tong S. and Chang E. 1Support vector machine active learning for image retrieval. In Proc. 9th ACM Int. Conf. on Multimedia, 2001, pp. 107–118.
16. Zhou X.S. and Huang T.S. Relevance feedback for image retrieval: a comprehensive review. Multimedia Syst., 8(6):536–544, 2003.

Image Retrieval System

▶ Image Database

Image Segmentation

Frank Y. Shih
New Jersey Institute of Technology, Newark, NJ, USA

Synonyms

Region segmentation; Pixel classification; Edge detection; Thresholding

Definition

The rapid rate of image analysis field has grown enormously in the past few decades. Image analysis intends to construct explicit, meaningful descriptions of physical objects in images. It can be divided into two parts: low-level image analysis and high-level image analysis. Low-level tasks focus on region-based segmentation, whereas high-level tasks are related to object-oriented representation. Image segmentation, a process of pixel classification, aims to extract or segment objects or regions from the background. Intrinsic images can be generated at the low-level processing, revealing physical properties of the imaged scene. This can often be implemented with parallel computation.

Historical Background

Image segmentation is a critical step to the success of object recognition [12], image compression [2], image visualization [7], and image retrieval [3]. Pal and Pal [13] provided a review on various segmentation techniques. It should be noted that there is no single standard approach to segmentation. Many different types of scene parts can serve as the segments on which descriptions are based, and there are many different ways in which one can attempt to extract these parts from the image. Selection of an appropriate segmentation technique depends on the type of images and applications.

The level of segmentation or subdivision relies on the problem domain being dealt with. For example, in the optical character recognition (OCR), the text is separated from the document image, and further partitioned into columns, lines, words, and connected components. In building character subimages, one is often confronted with touching or broken characters that occur in degraded documents (such as fax, scan, photocopy, etc.). It is still challenging to develop techniques for properly segmenting words into their characters.

There are primarily four types of segmentation techniques: thresholding, boundary-based, region-based, and hybrid techniques. Thresholding is based on the assumption that clusters in the histogram correspond to either background or objects of interest that can be extracted by separating these histogram clusters. In addition to thresholding, many image segmentation algorithms are based on two basic properties of the pixel intensities in relation to their local neighborhood: discontinuity and similarity. Methods based on pixel discontinuity are called boundary-based or edge extraction methods, whereas methods based on pixel similarity are called region-based methods. Boundary-based methods assume that the pixel properties, such as intensity, color, and texture, should change abruptly between different regions. Region-based methods assume that neighboring pixels within the same region should have similar values (e.g., intensity, color, texture).

It is well known that such segmentation techniques – based on boundary or region information alone – often fail to produce accurate segmentation results. Hence, there has been a tendency towards hybrid segmentation algorithms which take advantage of the complementary nature of such information. Hybrid methods combine boundary detection and region growing together to achieve better segmentation [5,6,14]. Note that both results should achieve the foreground and background segmentation coherently.

Foundations

A number of image segmentation techniques, including thresholding, component labeling, locating object contours by the snake model, and automatic seeded region growing, are described below.

Thresholding

Thresholding provides an easy and convenient way to perform image segmentation based on the different intensities or colors in the foreground and background regions of an image. Not all images can be segmented successfully into foreground and background using simple thresholding. Its validity relies on the distribution of the intensity histogram. If the intensity distribution of foreground objects is quite distinct from the intensity distribution of background, it will be clear to apply thresholding for image segmentation. In this case, one expects to see distinct peaks in the histogram corresponding to foreground objects, such that threshold values can be picked to isolate those peaks accordingly. If such a peak does not exist, it is unlikely that simple thresholding can achieve a good segmentation.

There are several methods of choosing the threshold value λ. For example, the universal thresholding by Donoho et al. [4] sets

$$\lambda = \frac{\sigma\sqrt{2\log n}}{\sqrt{n}} \tag{1}$$

where σ is the standard deviation of the wavelet coefficients and n is the total size of samples. Other possibility is quantile thresholding where λ is statistically set to replace a percentage of the coefficients with the smallest magnitude to zero. Another adaptive method of automatically choosing the best threshold λ consists of four steps: (i) Choose an initial estimate λ; (ii) Calculate the two mean values μ_1 and μ_2 within the two groups of pixels after thresholding at λ; (iii) Calculate the new threshold value $\lambda = (1/2)(\mu_1 + \mu_2)$; (iv) If the new threshold value has a little change (i.e., smaller than a predefined constant), then the threshold selection is done; otherwise, go back to step 2.

Object (Component) Labeling

It is very possible to have more than one object in a scene. All the objects must be individually extracted for the purpose of establishing the object model base. The object-labeling technique is used, so that the array representation of these objects is a multivalued picture, in which the points of each component all have a unique nonzero label and the points of background are all zeros. This technique only requires two raster scans. To label 4-connected components, only the upper and left neighbors are checked. If the 8-connectedness is used, the upper two diagonal neighbors are also included.

Let the value of object points be "1" and of background points be "0." Assume that the 8-connectedness is adopted. Therefore, if all the above four neighbors (i.e., previously scanned neighbors) of the point P are zeros, then P is assigned a new label. If one of four neighbors is 1, then P gets the same label as that neighbor. If two or more of them are 1's, then P gets any one of their labels, and the equivalence table is established by marking the different labels together for later adjustment. Equivalence processing consists in merging the equivalent pair into the same class; i.e., a unique label is assigned to each class. Finally, a second scan is performed to replace each label by the representative of its class. Each component has now been uniquely labeled. After these processes, each individual model object can be fetched by its label.

Locating Object Contours by the Snake Model

In the original snake formulation of Kass et al. [9], the best snake position was defined as the solution of a variational problem requiring the minimization of the sum of internal and external energies integrated along the length of the snake. The corresponding Euler equations, which give the necessary conditions for this minimizer, comprise a force balance equation. The snake model has provided a number of applications in object segmentation, stereo matching, motion tracking, etc. In image processing, the snake model defines a snake as an energy-minimizing spline guided by external constraint forces and influenced by such forces that pull it toward image features such as lines and edges. It is a kind of the active contour model in the way that it locks on nearby edges, localizing them accurately.

There are two parts in the energy function of the snake model. The first part reflects geometric properties of the contour, and the second part utilizes the external force field to drive the snake. The first part serves to impose a piecewise smoothness constraint, and the second part is responsible for putting the snake near the local minimum of energy. The traditional snakes suffer from a great disadvantage that when an object resides in a complex background, the strong edges may not be the object edges of interest.

Therefore, researchers have proposed statistical and variational methods to enrich the energy function and extend its flexibility. They calculated the difference between the target object and the background using statistical analysis, but these limitations come from the priori knowledge requirements, such as independent probability models and template models. Unfortunately, such priori knowledge is usually unavailable unless the captured images are under very constrained settings. A problem with the snake model is that a user needs to place the initial snake points sufficiently close to the feature of interest.

The Traditional Snake Model A snake is a controlled continuity spline that moves and localizes onto a specified contour under the influence of the objective function. Let a snake be a parametric curve: $v(s) = [x(s), y(s)]$, where parameter $s \in [0, 1]$. It moves around the image spatial domain to minimize the objective energy function as defined by

$$E_{snake}(v) = \sum_{\mathrm{i=1}}^{\mathrm{n}} [\alpha \times E_{cont}(v_i) + \beta \times E_{curv}(v_i) \\ +\gamma \times E_{image}(v_i)], \tag{2}$$

where α, β and γ are weighting coefficients that control the snake's tension, rigidity, and attraction, respectively. The first and second terms are correspondingly the first- and second-order continuity constraints. The third term measures the edge strength (i.e., the image force).

The continuity force E_{cont} encouraging even spacing of points can be calculated as

$$E_{cont}[v_i] = \frac{\left|\overline{d} - |v_i - v_{i-1}|\right|}{\max_j\left\{\left|\overline{d} - |v_i(j) - v_{i-1}|\right|\right\}}, \tag{3}$$

where $\{v_i(j)|j = 1, 2, ..., m\}$ denotes the snake point v_i's m neighbors, and $\overline{d}$ denotes the average length of all the pairs of adjacent points on the snake contour as given by

$$\overline{d} = \frac{\sum_{i=1}^{n} |v_i - v_{i-1}|}{n}, \tag{4}$$

where $v_0 = v_n$. This term tends to keep the distances between each pair of adjacent vertices equal.

The energy of the second-order continuity E_{curv} is represented by

$$E_{curv}[v_i] = \frac{|v_{i-1} - 2v_i + v_{i+1}|}{\max\{|v_{i-1} - 2v_i + v_{i+1}|\}}. \tag{5}$$

The numerator can be rearranged as

$$v_{i-1} - 2v_i + v_{i+1} = (v_{i+1} - v_i) - (v_i - v_{i-1}). \tag{6}$$

If the ith vertex is pushed toward the midpoint of two adjacent vertices, E_{image} is minimized; i.e., the shape of the contour will remain $\subset^2$ continuity.

The third term, image energy E_{image}, is derived from the image so that it takes on its smaller values at the features of interest, such as boundaries. It considers the gradient (denoted as *grad*) magnitude, leading the active contour toward step edges. It is normalized to measure the relative magnitude as

$$E_{image}[v_i] = \frac{\min\{|grad|\} - |grad_{v_i}|}{\max\{|grad|\} - \min\{|grad|\}}, \tag{7}$$

where min and max denote the minimum and maximum gradients in the v_i's local m-neighborhood, respectively. Note that because the numerator in eq. (7) is always negative, E_{image} can be minimized for locating the largest gradient, which is the edge. In general, the traditional snake model can locate object contours in a simple background. If the background becomes complex, it will fail since the complex background will generate noisy edges to compete with the object edges for attracting the snake.

The Improved Snake Model The following two issues are observed when the snake model fails to locate object contours in complex backgrounds. One is the gray-level sensitivity; i.e., the more abrupt change the gray levels have (e.g., noises), the larger impact on the energy function the snake makes. The other is that the snake mistakenly locates the edges belonging to the background details due to their closeness to the snake point. Figure 1 shows a disk object in a complex background. If the snake points are initialized outside the disk, the snake cannot locate the disk contour accurately due to the disturbances from the background grids. The idea is to push the mean intensity of the polygon enclosed by the snake contour to be as close as to the mean intensity of the target object. The smaller intensity difference between the polygon and the object, the closer the snake approaches the object contour. Therefore, a new energy term, called the *Regional Similarity Energy* (RSE), is established for calculating the gray-level differences to be added into the overall energy [16].

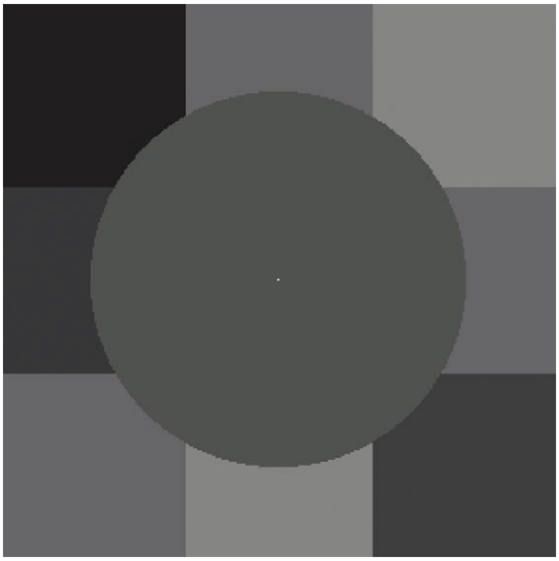


Image Segmentation. Figure 1. A disk object resides in a complex background.

The Gravitation External Force Field and the Greedy Algorithm In this section, the gravitation external force field is introduced and the greedy algorithm [8] is used for the active contour. The concept of gravitation external force field is taken from physics. Two objects attract each other by a force, which is proportional to their mass product and inversely proportional to the distance between their mass centers. Based on this concept, an external energy field, called the *Gravitation Energy Field* (GEF), is developed as given by

$$E_{gravitation} = \int \frac{g(\vec{r})}{\|\vec{r}\|}\vec{r}d\vec{r}, \tag{8}$$

where $\vec{r}$ is a position vector and $g(\vec{r})$ is a first-order derivative. Note that the edge pixels have the local maxima in the first-order derivative. The attractive force enables the snake points to move toward the object. With the gravitation energy field, the active contour can be dragged toward the object even if the snake points are far away. Therefore, the total energy function becomes

$$\begin{aligned} E_{snake} = {} & \alpha E_{cont}[v(s)] + \beta E_{curv}[v(s)] + \gamma E_{image}[v(s)] \\ & + \mu E_{gravitation}[v(s)] + \delta E_{RSE}[v(s)]. \end{aligned} \tag{9}$$

These weighting coefficients can be adjusted according to the user's application. For simplicity, $\alpha = \beta = \gamma = \mu = \delta = 1$.

Experimental Results Both the improved and the traditional snake models are applied on the added salt-and-pepper noise of Fig. 1. The results are shown in Fig. 2. It is observed that the improved model can locate the disk contour, but the traditional model fails. The improved model is suitable for random noise or fixed pattern noise. The banding noise is highly camera-dependent, and is the one which is introduced by the camera when it reads data from the digital sensor. The improved model may not perform well in an image with such noise.

Automatic Seeded Region Growing

Seeded Region Growing (SRG) is one of hybrid methods proposed by Adams and Bischof [1]. It starts with assigned seeds, and grow regions by merging a pixel into its nearest neighboring seed region. Mehnert and Jackway [10] pointed out that SRG has two inherent pixel order dependencies that cause different resulting segments. The first order dependency occurs whenever several pixels have the same difference measure to their neighboring regions. The second order dependency occurs when one pixel has the same difference measure to several regions. They used parallel processing and re-examination to eliminate the order dependencies. Fan et al. [5] presented an automatic color image segmentation algorithm by integrating color-edge extraction and seeded region growing on the *YUV* color space. Edges in *Y*, *U*, and *V* are detected by an isotropic edge detector, and the three components are combined to obtain edges. The centroids between adjacent edge regions are taken as the initial seeds. The disadvantage is that their seeds are over-generated.

Overview of the Improved Seeded Region Growing Algorithm

Figure 3 presents the overview of the improved seeded region growing algorithm [15]. Firstly, the color image is converted from *RGB* to YC_bC_r color space. Secondly, automatic seed selection is applied to obtain initial seeds. Thirdly, the seeded region growing algorithm is used to segment the image into regions, where a region corresponds to a seed. Fourthly, the region-merging algorithm is applied to merge similar regions, and small regions are merged into their nearest neighboring regions.

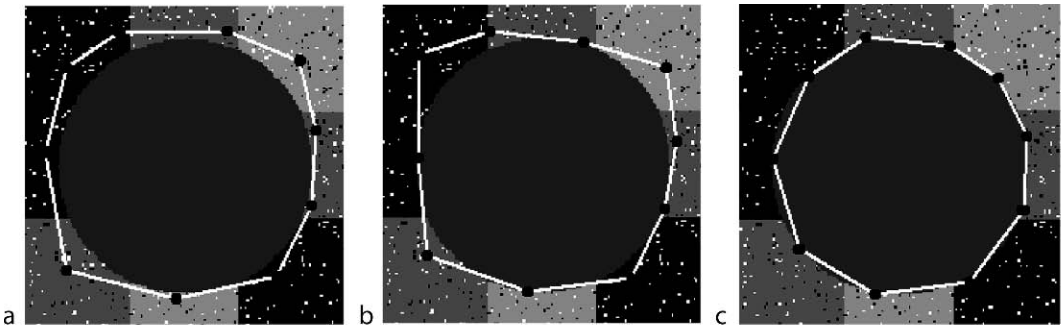


Image Segmentation. Figure 2. (a) The initial snake points, (b) the result by the traditional snake model, (c) the result by the improved snake model.

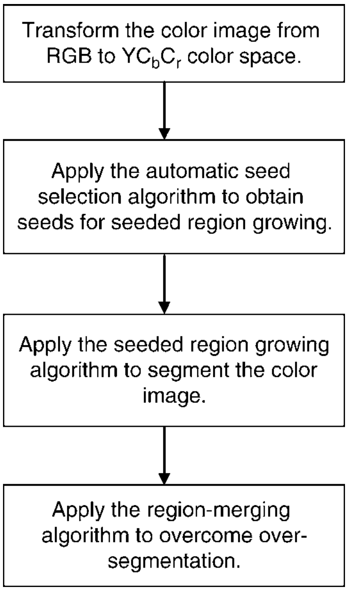

Transform the color image from RGB to YC$_b$C$_r$ color space.
Apply the automatic seed selection algorithm to obtain seeds for seeded region growing.
Apply the seeded region growing algorithm to segment the color image.
Apply the region-merging algorithm to overcome over-segmentation.

Image Segmentation. Figure 3. Outline of the proposed algorithm.

The Method for Automatic Seed Selection

For automatic seed selection, the following three criteria must be satisfied. First, the seed pixel must have high similarity to its neighbors. Second, for an expected region, at least one seed must be generated in order to produce this region. Third, seeds for different regions must be disconnected.

The similarity of a pixel to its neighbors can be computed as follows. Considering a 3 × 3 neighborhood, the standard deviations of Y, C_b and C_r components are calculated using

$$\sigma_x = \sqrt{\frac{1}{9}\sum_{i=1}^{9}(x_i - \bar{x})^2}, \tag{10}$$

where x can be Y, C_b, or C_r, and the mean value $\bar{x} = \frac{1}{9}\sum_{i=1}^{9} x_i$. The total standard deviation is

$$\sigma = \sigma_Y + \sigma_{C_b} + \sigma_{C_r}. \tag{11}$$

The standard deviation is normalized to [0,1] by

$$\sigma_N = \sigma/\sigma_{\max}, \tag{12}$$

where $\sigma_{\max}$ is the maximum of the standard deviation in the image. The similarity of a pixel to its neighbors is defined as

$$H = 1 - \sigma_N. \tag{13}$$

From the similarity, the first condition for the seed pixel candidate is defined as follows:

Condition 1: A seed pixel must have the similarity higher than a threshold value.

Secondly, the relative Euclidean distances (in terms of YC_bC_r) of a pixel to its 8 neighbors is calculated as

$$d_i = \frac{\sqrt{(Y - Y_i)^2 + (C_b - C_{b_i})^2 + (C_r - C_{r_i})^2}}{\sqrt{Y^2 + C_b^2 + C_r^2}} \quad i = 1, 2, \cdots, 8. \tag{14}$$

From experiments, the performance of using relative Euclidean distance is better than using normal Euclidean distance. For each pixel, the maximum distance to its neighbors is calculated as

$$d_{\max} = \max_{i=1}^{8}(d_i). \tag{15}$$

From the maximum distance, the second condition for the seed pixel candidate is defined below.

Condition 2: A seed pixel must have the maximum relative Euclidean distance to its eight neighbors which is less than a threshold value.

A pixel is classified as a seed pixel if it satisfies the above two conditions. In order to choose the threshold automatically in condition 1, Otsu's method [11] is used. The threshold is determined by choosing the value that maximizes the discrimination criterion σ_B^2/σ_w^2, where σ_B^2 is the between-class variance and σ_W^2 is the within-class variance. In condition 2, the value 0.05 is selected as the threshold based on these experiments.

Each connected component of seed pixels is taken as one seed. Therefore, the seeds generated can be one pixel or one region with several pixels. Condition 1 checks whether the seed pixel has high similarity to its neighbors. Condition 2 makes sure that the seed pixel is not on the boundary of two regions. It is possible that for one desired region, several seeds are detected to split it into several regions. The over-segmented regions can be merged later in the region-merging step. Figure 4(a) shows a color image, and (B) shows the detected seeds marked in red color. Note that the connected seed pixels are considered as one seed.

The Segmentation Algorithm

Let $A_1, A_2, \ldots, A_i$ denote initial seeds and S_i denote the region corresponding to A_i. The mean of all seed pixels in S_i in terms of Y, C_b and C_r components is denoted as $(\bar{Y}, \bar{C}_b, \bar{C}_r)$. The segmentation algorithm is described as follows:

1. Perform automatic seed selection.
2. Assign a label to each seed region.
3. Record neighbors of all regions in a sorted list T in a decreasing order of distances.
4. While T is not empty, remove the first point p and check its 4-neighbors. If all labeled neighbors of p have a same label, set p to this label. If the labeled neighbors of p have different labels, calculate the distances between p and all neighboring regions and classify p to the nearest region. Then, update the mean of this region and add 4-neighbors of p, which are neither classified yet nor in T, to T in a decreasing order of distances.
5. Perform region merging.

Note that in step 3, T denotes the set of pixels that are unclassified and are neighbors of at least one of the

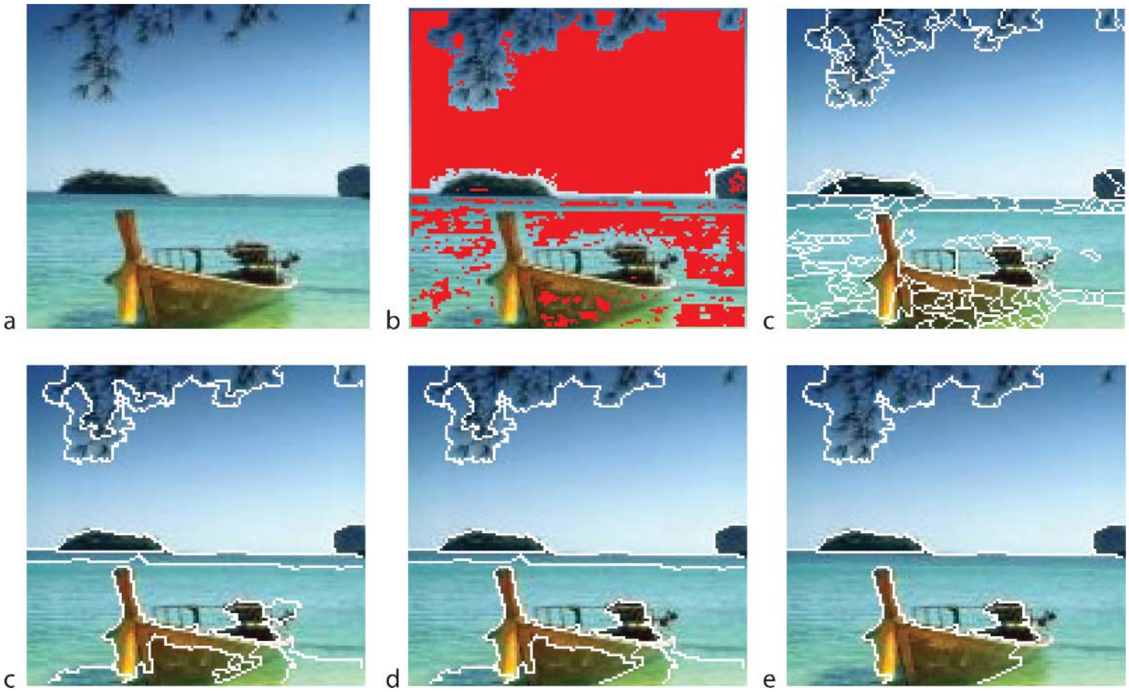


Image Segmentation. Figure 4. (a) Original color image, (b) the detected seeds are shown in red color, (c) seeded region growing result, (d) the result of merging adjacent regions with relative Euclidean distance less than 0.1, (e) the result of merging small regions with size less than 1/150 of the image, and (f) final segmented result.

region. The relative Euclidean distance d_i between the pixel i and its adjacent region is calculate by

$$d_i = \frac{\sqrt{(Y_i - \overline{Y})^2 + (C_{b_i} - \overline{C}_b)^2 + (C_{r_i} - \overline{C}_r)^2}}{\sqrt{Y_i^2 + C_{b_i}^2 + C_{r_i}^2}}, \quad (16)$$

where $(\overline{Y}, \overline{C}_b, \overline{C}_r)$ are the mean values of Y, C_b, and C_r components in that region. In step 4, the pixel p with the minimum distance value is extracted. If several pixels have the same minimum value, the pixel corresponding to the neighboring region having the largest size is chosen. If p has the same distance to several neighboring regions, it is classified to the largest region. Figure 4(c) shows the result of the proposed algorithm, where boundaries of regions are marked in white color.

Key Applications

Object Recognition, Image Compression, Image Visualization, and Image Retrieval.

Cross-references

▶ Computer Vision
▶ Image Segmentation
▶ Pattern Recognition
▶ Scene Analysis

Recommended Reading

1. Adams R. and Bischof L. Seeded region growing. IEEE Trans. Pattern Anal. Mach. Intell., 16(6):641–647, 1994.
2. Belloulata K. and Konrad J. Fractal image compression with region-based functionality. IEEE Trans. Image Process., 11 (4):351–362, 2002.
3. Chen Y. and Wang J.Z. A region-based fuzzy feature matching approach to content-based image retrieval. IEEE Trans. Pattern Anal. Mach. Intell., 24(9):1252–1267, 2002.
4. Donoho D., Johnstone I., Kerkyacharian G., and Picard D. Density estimation by wavelet thresholding. Ann. Statist., 24:508–539, 1996.
5. Fan J., Yau D.K., Elmagarmid A.K., and Aref W.G. Automatic image segmentation by integrating color-edge extraction and seeded region growing. IEEE Trans. Image Process., 10(10): 1454–1466, 2001.
6. Haris K., Efstratiadis S.N., Maglaveras N., and Katsaggelos A.K. Hybrid image segmentation using watersheds and fast region merging. IEEE Trans. Image Process., 7(12):1684–1699, 1998.
7. Hartmann S.L. and Galloway R.L. Depth-buffer targeting for spatially accurate 3-D visualization of medical images. IEEE Trans. Med. Imaging 19(10):1024–1031, 2000.
8. Ji L. and Yan H. Attractable snakes based on the greedy algorithm for contour extraction. Pattern Recognit., 35(4):791–806, 2002.
9. Kass M., Witkin A., and Terzopoulos D. Snakes: active contour models. Int. J. Comput. Vis., 1(4):321–331, 1987.
10. Mehnert A. and Jackway P. An improved seeded region growing algorithm. Pattern Recognit. Lett., 18(10):1065–1071, 1997.
11. Otsu N. A threshold selection method from gray-level histogram. IEEE Trans. Syst., Man, Cybern., 9(1):62–66, 1979.
12. Pachowicz P.W. Semi-autonomous evolution of object models for adaptive object recognition. IEEE Trans. Syst. Man Cybern., 24(8):1191–1207, 1994.
13. Pal N.R. and Pal S.K. A review on image segmentation techniques. Pattern Recognit., 26(9):1277–1294, 1993.
14. Pavlidis T. and Liow Y.T. Integrating region growing and edge detection. IEEE Trans. Pattern Anal. Mach. Intell., 12(3): 225–233, 1990.
15. Shih F.Y. and Cheng S. Automatic seeded region growing for color image segmentation. Image Vis. Comput., 23(10): 877–886, 2005.
16. Shih F.Y. and Zhang K. Efficient contour detection based on improved snake model. Pattern Recognit. Artif. Intell., 18(2): 197–209, 2004.

Image Standards

▶ Image Representation

Image/Video/Music Search

▶ Semantic Modeling and Knowledge Representation for Multimedia Data

Immersive Data Mining

▶ Visual Data Mining

Implementation Abstraction

▶ Visual Data Mining

Implication of Constraints

Wenfei Fan
University of Edinburgh, Edinburgh, UK

Definition

The implication problem is to decide whether or not a given set of constraints logically implies another constraint. With any constraint (dependency) language $\mathcal{L}$ there are two implication problems associated, which do not coincide in general.

In the traditional logic framework, an instance of a schema **R** is a logical structure that is either finite or infinite, referred to as an *unrestricted instance* of **R**. A set Σ of constraints over **R** *implies without restriction* a constraint φ, denoted by $\Sigma \vDash_{\text{unr}}\varphi$, if for each unrestricted instance **I** of **R** that satisfies Σ, **I** also satisfies φ. The *unrestricted implication problem* for $\mathcal{L}$ is to determine, given a set Σ of constraints in $\mathcal{L}$ and another constraint φ in $\mathcal{L}$, whether or not $\Sigma \vDash_{\text{unr}}\varphi$.

In the context of databases, only finite instances are considered and implication analysis lies within finite model theory. A set Σ of constraints over **R** *finitely implies* a constraint φ, denoted by $\Sigma \vDash_{\text{fin}}\varphi$, if for each finite instance **I** of **R** that satisfies Σ, **I** also satisfies φ. The *finite implication problem* for $\mathcal{L}$ is to determine, given any set Σ of constraints in $\mathcal{L}$ and another constraint φ in $\mathcal{L}$, whether or not $\Sigma \vDash_{\text{fin}}\varphi$.

In the context of semi-structured data or XML, implication analysis is also conducted in the absence of schema. For XML, a set Σ of constraints *implies* another constraint φ *in the absence of schema* if all XML documents that satisfy Σ also satisfy φ; similarly for semi-structured data.

Historical Background

Logical implication is one of the key issues in dependency theory. It has been studied for, and has proved valuable in, schema normalization, data integrity maintenance and query optimization, among other things. Recently there has also been renewed interest in this line of work, for XML data management, data exchange and data cleaning.

After functional dependencies were introduced by Codd in 1972, the relevance of implication to database theory was first observed by [8]. Since then the problem of logical implication has been extensively studied for a large variety of dependencies, mostly focusing on the following aspects. (i) The complexity of implication analyses (*e.g.*, [3,6,7,9,10,11,15]). In particular, the separation of finite and unrestricted implication was originally investigated in [7,10], and the chase, a powerful decision procedure for implication, was based on the idea of [3] and articulated in [7,15]. (ii) Axiomatization for characterizing logical implication, first studied by Armstrong [5] for functional dependencies, followed by a flurry of research (see, *e.g.*, [12] for a survey). (iii) View dependencies for propagating dependencies from databases to their views, originally studied in [14]. (iv) The impact of interaction between schema (types) and constraints on implication analysis, first studied by [2] for semi-structured data, and then for XML in the presence and in the absence of schema [13]. See [1,12] for comprehensive surveys.

Foundations

The remainder of this entry discusses various aspects of implication: dependencies, unrestricted and finite implication, implication and satisfiability analyses, finite axiomatization, complexity bounds, and view dependencies.

Constraints (Data Dependencies)

Implication analysis has been studied for a variety of integrity constraints (in this entry, integrity constraints and data dependencies are used interchangeably). Most constraints studied for relational databases can be expressed as first-order logic sentences of the following form, referred to as *embedded dependencies*:

$$\forall x_1 \ldots x_m \ (\phi(x_1, \ldots, x_m) \rightarrow \exists y_1 \ldots y_n \ \psi(z_1, \ldots, z_k)),$$

where (i) $\{y_1, \ldots, y_n\} = \{ z_1, \ldots, z_k\} - \{ x_1, \ldots, x_m\}$; (ii) ϕ is a conjunction of (at least one) relation atoms of the form $R(w_1, \ldots, w_l)$, using all of the variables in $\{x_1, \ldots, x_m\}$, where w_i is a variable for each $i \in [1,l]$; (iii) ψ is a conjunction of either relation atoms or equality atoms $w = w'$, using all of the variables in $\{z_1, \ldots, z_k\}$, where w, w' are variables; and (iv) there exist no equality atoms in ψ using existentially quantified variables.

Embedded dependencies are often classified as follows.

1. Full dependencies are embedded dependencies that have no existential quantifiers, *i.e.*, dependencies of the form $\forall x_1 \ldots x_m \ (\varphi(x_1, \ldots, x_m) \ \rightarrow \ \psi(z_1, \ldots, z_k))$, where $\{z_1, \ldots, z_k\}$ is a subset of $\{x_1, \ldots, x_m\}$. Full

dependencies include functional dependencies (FDs), multivalued dependencies (MVDs) and join dependencies (JDs) (see [1,12] for the definitions of FDs, MVDs, JDs and inclusion dependencies).

2. Tuple generating dependencies (TGDs) are embedded dependencies in which the right-hand side ψ is a relation atom. A TGD says that if a certain pattern of entries appears then another pattern must appear. Inclusion dependencies (INDs), MVDs and JDs are examples of TGDs.
3. Equality generating dependencies (EGDs) are embedded dependencies in which the right-hand side ψ is an equality atom. An EGD says that if a certain pattern of entries appears then a certain equality must hold. The best known equality generating dependencies are FDs.
4. Typed dependencies are embedded dependencies for which there exists an assignment of variables to column positions such that (i) variables in relation atoms occur only in their assigned position, and (ii) each equality atom involves a pair of variables assigned to the same position. Typed dependencies include FDs, MVDs and JDs.

Implication analysis has also been studied for integrity constraints on semi-structured data and XML. This entry focuses on relational dependencies only, and refer the interested reader to [4] for a survey on XML constraints.

Unrestricted Implication Versus Finite Implication

Given a set Σ of constraints and another constraint φ, if $\Sigma \models_{\text{unr}} \varphi$, *i.e.,* Σ implies φ without restriction, then obviously $\Sigma \models_{\text{fin}} \varphi$, *i.e.,* Σ finitely implies φ. However, the converse does not necessarily hold. It is possible that $\Sigma \models_{\text{fin}} \varphi$ whereas $\Sigma \not\models_{\text{unr}} \varphi$. That is, the implication problem and finite implication problem may have to be treated separately as different decision problems. Below are two examples.

1. When Σ is a set of FDs and INDs, and φ is either an FD or an IND [9].

   Consider a binary relation *R(A,B)* with attributes *A,B*. Let $\Sigma = \{A \rightarrow B, R[A] \subseteq R[B]\}$, and φ be $R[B] \subseteq R[A]$. Then from Σ it follows that for any finite instance I of R that satisfies Σ, $|\pi_A(I)| \geq |\pi_B(I)|$ (by the FD in Σ) and $|\pi_A(I)| \leq |\pi_B(I)|$ (by the IND in Σ), where π is the projection operator in the relational algebra, and $|S|$ denotes the number of distinct tuples in a relation S. Therefore, $|\pi_A(I)| = |\pi_B(I)|$. Since I is finite and $\pi_A(I) \subseteq \pi_B(I)$, it follows that $\pi_B(I) \subseteq \pi_A(I)$ and I satisfies φ. Conversely, an infinite instance $\{(i+1,i) | i \geq 0\}$ of R satisfies Σ but does not satisfy φ; thus $\Sigma \not\models_{\text{unr}} \varphi$.

   Similarly, let φ be $R[B] \rightarrow R[A]$ then one can verify that $\Sigma \models_{\text{fin}} \varphi$ but $\Sigma \not\models_{\text{unr}} \varphi$.
2. When Σ is a set of TGDs and φ is a TGD (see [7] for a proof).

   A useful technique for proving the decidability of the implication problem for a constraint language $\mathcal{L}$ is to show that the implication and finite implication problems coincide for $\mathcal{L}$, *i.e.,* for any set Σ in $\mathcal{L}$ and another constraint φ in $\mathcal{L}$, $\Sigma \models_{\text{unr}} \varphi$ if and only if $\Sigma \models_{\text{fin}} \varphi$. Indeed, all relational constraints considered so far are definable in first-order logic. For these constraints implication is *r.e.* (recursively enumerable) and finite implication is co-*r.e.*. As a result, if implication and finite implication coincide, then both are recursive, *i.e.,* decidable. For full dependencies, for example, the implication and finite implication problems coincide, and are both decidable.

Implication Versus Satisfiability

Consider Σ and φ defined over a relational schema **R**. Obviously, $\Sigma \models_{\text{unr}} \varphi$ (resp. $\Sigma \models_{\text{fin}} \varphi$) if and only if the sentence $\wedge\ \Sigma\ \wedge \neg\varphi$ is (resp. finitely) satisfiable, *i.e.,* there exists a (resp. finite) instance of **R** that satisfies the sentence. This tells us that there is close connection between (resp. finite) implication and (resp. finite) satisfiability, problems studied in (resp. finite) model theory. For a constraint language $\mathcal{L}$ that is closed under negation (*i.e.,* if $\varphi \in \mathcal{L}$ then so is $\neg\varphi$), the (resp. finite) implication problem is just a special case of the (resp. finite) satisfiability problem.

Embedded dependencies are not closed under negation, and their (finite) implication analyses are quite different from their (finite) satisfiability counterparts. For any set Σ of embedded dependencies defined over a relational schema **R**, the empty instance $\mathbf{I}\phi$ of **R**, *i.e.,* an empty database with no tuples, satisfies Σ. For the analysis of (finite) satisfiability, embedded dependencies have the *domain independence* property: when considering whether a database **I** satisfies Σ, it suffices to consider the tuples in **I** without worrying about the underlying domains of the attributes in **R**. In contrast, to verify $\Sigma \models_{\text{unr}} \varphi$ (resp. $\Sigma \models_{\text{fin}} \varphi$) one may have to consider all (resp. finite) instances of **R**, possibly ranging over all the values in the underlying domains of the attributes in **R**.

Despite this, when studying (finite) implication analyses for certain subclasses of embedded dependencies, it is still possible to capitalize on results on their (finite) satisfiability problems. For example, when Σ and φ are full dependencies, $\wedge\ \Sigma\ \wedge\neg\varphi$ is an $\exists^*\forall^*$ sentence (with equality or not), in a fragment of first-order logic known as the Bernays-Schőnfinkel-Ramsey class. It is known that for the Bernays-Schőnfinkel-Ramsey class, the satisfiability and finite satisfiability problems coincide and are decidable in NEXPTIME (non-deterministic exponential time; it is in fact NEXPTIME-complete in the absence of functions). This yields an upper bound on the implication and finite implication problems for full dependencies.

It is worth mentioning that when it comes to XML, the interaction between schemas (types) and integrity constraints becomes more intriguing than their relational counterparts, and as a result, the finite satisfiability problem becomes much harder. Indeed, for a class $\mathcal{L}_0$ of unary keys and foreign keys for XML, it is undecidable to decide, given a set Σ of constraints in $\mathcal{L}_0$ and a DTD D_0 (document type definition), whether or not there exists a finite XML document that satisfies both Σ and D_0. There are, however, interesting connections between finite satisfiability and finite implication analyses for XML constraints (see [4,13] for detailed discussions).

Finite Axiomatizability

Another important approach to studying (finite) implication of constraints is based on finite axiomatization. A finite axiom system $\mathcal{A}$ for a class $\mathcal{L}$ of constraints consists of finitely many axiom schemes and inference rules. A *proof* of a constraint φ in $\mathcal{L}$ from a set Σ of constraints in $\mathcal{L}$ *using* $\mathcal{A}$ is a finite sequence $\varphi_1,\ldots,\varphi_n$ such that φ_n is φ, and for each i ∈ [1, n], (i) $\varphi_i \in \Sigma$, or (ii) φ_i is an instance of an axiom scheme in $\mathcal{A}$, or (iii) φ_i follows from preceding constraints in the sequence by one of the inference rules in $\mathcal{A}$. If there exists such a proof, then φ is said to be *provable* from Σ using $\mathcal{A}$, denoted by $\Sigma \vdash_{\mathcal{A}} \varphi$.

For (resp. finite) implication of $\mathcal{L}$, the axiom system $\mathcal{A}$ is *sound* if $\Sigma \vdash_{\mathcal{A}} \varphi$ entails $\Sigma \models_{\text{unr}} \varphi$ (resp. $\Sigma \models_{\text{fin}} \varphi$); it is *complete* if $\Sigma \models_{\text{unr}} \varphi$ (resp. $\Sigma \models_{\text{fin}} \varphi$) entails $\Sigma \vdash_{\mathcal{A}} \varphi$. If $\mathcal{A}$ is both sound and complete then it is called a *finite axiomatization* of (finite) implication of $\mathcal{L}$, which characterizes (finite) implication of $\mathcal{L}$.

The best known example of finite axiomatizations is Armstrong's axioms for functional dependencies (FDs). Recall that an FD is of the form X → Y, where X and Y are sets of attributes. Armstrong's axiom system consists of:

Reflexivity axiom: $X \rightarrow X$.

Augmentation: If $X \rightarrow Y$ then $XZ \rightarrow YZ$, where XZ denotes $X \cup Z$.

Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$.

Relative to a set U of attributes, an instance of the reflexivity axiom $X \rightarrow X$ is $V \rightarrow V$ when V is a subset of U. An FD φ_i follows from preceding constraints in a proof $\varphi_1,\ldots,\varphi_n$ if there exist a reference rule (either augmentation or transitivity) ρ and a substitution σ from the variables in ρ to subsets of U, such that for each FD in the antecedent of ρ, the corresponding FD obtained via σ is φ_j for some $j < i$. For example, one can easily verify that $\varphi = A \rightarrow C$ is provable from $\Sigma = \{A \rightarrow B, B \rightarrow C\}$ using Armstrong's axioms, *i.e.,* $\Sigma \vdash \varphi$.

For finite implication of $\mathcal{L}$, the existence of a finite axiomatization is a stronger property than the existence of a testing algorithm. Indeed, from the existence of a finite axiomatization for finite implication of $\mathcal{L}$ follows the decidability of the finite implication problem for $\mathcal{L}$ (because the finite implication becomes r.e., and it is co-r.e. for first-order logic). On the other hand, there are dependencies (*e.g.,* join dependencies JDs), such that for their finite implication there is no finite axiomatization, but there is a testing algorithm (via the chase [3,15]).

Finite axiomatizations have been developed for implication and finite implication of a variety of dependencies, including FDs, INDs, MVDs, FDs and MVDs taken together, and for full typed dependencies. There is also a sound and complete axiom system for implication (*not* finite implication) of typed TGDs and EGDs. On the other hand, it is known that there exist no finite sound and complete axiom systems for FDs and INDs taken together.

Complexity of Implication Analyses

A number of complexity results have been established on the implication and finite implication problems for a variety of constraint languages. As remarked earlier, for full dependencies, the implication and finite implication problems coincide and are both decidable. For the following classes of full dependencies, the implication and finite implication have the complexity bounds given below, in which n is the length of the input constraints Σ and φ:

(i) in $O(n)$-time for FDs [6];
(ii) in $O(n \log n)$-time for MVDs;
(iii) in $O(n^2)$-time for deciding whether an MVD or an FD is implied by a set of typed full dependencies;
(iv) in $O(n^2 \log^2 n)$-time for deciding whether a JD is implied by a set of FDs.

In contrast to the NEXPTIME upper bound on the Bernays-Schőnfinkel-Ramsey class given earlier, there exists an efficient decision procedure for these fragments of full dependencies. On the other hand, several intractability results have also been established:

(v) It is NP-hard to decide whether a set of MVDs implies a JD, and it is NP-complete to decide whether a JD and an FD imply a JD.
(vi) The implication problem for full dependencies is EXPTIME-complete, typed or untyped.

Beyond full dependencies, fewer positive results are known.

(vii) The implication and finite implication problems for INDs coincide and are PSPACE-complete [9].

A well-known undecidability result is the following, proved independently by Chandra, Vardi [11] and by Mitchell:

(viii) For FDs and INDs put together, the implication and finite implication problems differ, and are undecidable.

When it comes to TGDs, the analysis of implication is also beyond reach in practice:

(ix) For TGDs, the implication and finite implication problems are undecidable, even when only typed TGDs are considered.

One of the most powerful tools for testing implication is the chase [3,15]. To determine whether or not Σ implies φ, the basic idea of the chase is to represent φ as a tableau T, and repeatedly apply constraints in Σ to T. This yields a sequence of tableaux, referred to as a chasing sequence of T by Σ, and as a terminal sequence if it is finite and no constraint in Σ can be further applied to it. If a chasing sequence reaches a tableau satisfying a certain condition (depending on what type of φ is), then one can conclude that φ is implied by Σ.

When Σ is a set of full dependencies and φ is a typed dependency, it is known that chasing terminates and has the Church-Rosser property, *i.e.*, different terminal chasing sequences yield the same unique result. Based on this an EXPTIME algorithm can be developed for testing whether $\Sigma \vDash_{\text{unr}} \varphi$ and $\Sigma \vDash_{\text{fin}} \varphi$, for full dependencies.

When Σ is a set of embedded dependencies, however, the chase may not terminate. Nevertheless, a decision procedure based on the chase can be developed such that it will give a positive answer if $\Sigma \vDash_{\text{unr}} \varphi$, and will not terminate if $\Sigma \nvDash_{\text{unr}} \varphi$. See [1] for detailed discussions.

View Dependencies

An important application of logical implication and the chase is the analysis of constraint propagation from databases to their views. Let $\mathbf{R}$ be a database schema, Σ a set of constraints over $\mathbf{R}$, V a view definition on $\mathbf{R}$, and φ a constraint defined over the view. Then $\mathbf{R}$: Σ *implies* $V: \varphi$, denoted by $\mathbf{R}: \Sigma \vDash V: \varphi$, if $V(\mathbf{I})$ satisfies φ for each instance $\mathbf{I}$ of $\mathbf{R}$ that satisfies Σ. The *constraint propagation* problem (*a.k.a.* the view dependency inference problem) is to determine, given $\mathbf{R}$, Σ, V and φ, whether or not $\mathbf{R}: \Sigma \vDash V: \varphi$. The analysis of constraint propagation is useful in, among other things, data exchange, data integration and data cleaning.

For example, when $\mathbf{R}$ consists of a relation schema R with attributes (A, B, C, D), Σ consists of an FD $A, B \rightarrow C$, V is the query $\pi_{A,\ B,\ C}(\sigma_{A=1}R)$ in the relational algebra, and φ is $B \rightarrow C$, one can see that $\mathbf{R}$: $\Sigma \vDash V: \varphi$.

The constraint propagation problem has been studied for full dependencies and for views defined in the relational algebra, based on an extension of the chase technique. The following complexity bounds are known.

(i) It is undecidable when views V are defined in the relational algebra and the constraints Σ and φ are FDs.
(ii) When $\Sigma \cup \{\varphi\}$ is a set of FDs and MVDs, and views are SPCU queries (selection, projection, Cartesian product and union), the constraint propagation problem is decidable in polynomial time.
(iii) When Σ is a set of FDs and JDs, φ is a JD, and views are SPCU queries, the problem is NP-complete.
(iv) When $\Sigma \cup \{\varphi\}$ is a set of full dependencies and views are SPCU queries, the problem is EXPTIME-complete.

Constraint propagation has also been studied in the context of XML shredding, *i.e.*, for mapping XML data to relations, from XML keys to relational FDs.

Key Applications

Traditional applications of the analysis of constraint implication include schema normalization, data integrity maintenance, storage implementation, and query optimization (see [1]). The prevalent use of the Web has motivated the development of new constraint languages for specifying the semantics of semi-structured data and XML, as well as for capturing the consistency of data. Logical implication has found new applications in XML query optimization, data integration, data exchange and data cleaning.

Future Directions

Several problems remain open for implication analysis of constraints developed for specifying XML semantics and for data cleaning. For example, the exact complexity bounds on the implication problems for certain XML functional dependencies in the presence and in the absence of DTDs are not yet settled. Another topic is the development of efficient algorithms for testing implication of constraints. For a variety of constraint languages the (finite) implication problem is intractable or even undecidable, *e.g.*, for functional and inclusion dependencies taken together. It is important and practical to find effective heuristic algorithms for their implication analyses, ideally with certain performance guarantees. This issue deserves a full treatment.

Cross-references

- ▶ Constraint-Drive Database Repair
- ▶ Data Exchange
- ▶ Database Dependencies
- ▶ Logical Structure
- ▶ Normal Forms and Normalization
- ▶ XML Integrity Constraints

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of databases. Addison-Wesley, Reading, MA, USA, 1995.
2. Abiteboul S. and Vianu V. Regular path queries with constraints. J. Comput. Syst. Sci., 58(3):428–452, 1999.
3. Aho A.V., Beeri C., and Ullman J.D. The theory of joins in relational databases. ACM Trans. Database Syst., 4(3):297–314, 1979.
4. Arenas M., Fan W., and Libkin L. Consistency of XML specifications. In Inconsistency Tolerance. Springer, Berlin, 2005, pp. 15–41.
5. Armstrong W.W. Dependency structures of data base relationships. In Proc. IFIP Congress, 1974, pp. 580–583.
6. Beeri C. and Bernstein P.A. Computational problems related to the design of normal form relational schemas. ACM Trans. Database Syst., 4(1):30–59, 1979.
7. Beeri C. and Vardi M.Y. The implication problem for data dependencies. In Proc. 8th Int. Colloquium on Automata, Languages, and Programming, 1981, pp. 73–85.
8. Bernstein P.A. Synthesizing third normal form relations from functional dependencies. ACM Trans. Database Syst., 1(4):277–298, 1976.
9. Casanova M.A., Fagin R., and Papadimitriou C.H. Inclusion dependencies and their interaction with functional dependencies. In Proc. 1st ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1982, pp. 171–176.
10. Chandra A.K., Lewis H.R., and Makowsky J.A. Embedded implicational dependencies and their inference problem. In Proc. 13th Annual ACM Symp. on Theory of Computing, 1981, pp. 342–354.
11. Chandra A.K. and Vardi M.Y. The implication problem for functional and inclusion dependencies is undecidable. SIAM J. Comput., 14(3):671–677, 1985.
12. Fagin R. and Vardi M.Y. The theory of data dependencies – an overview. In Proc. 11th Int. Colloquium on Automata, Languages, and Programming, 1984, pp. 1–22.
13. Fan W. and Libkin L. On XML integrity constraints in the presence of DTDs. J. ACM, 49(3):368–406, 2002.
14. Klug A.C. Calculating constraints on relational expressions. ACM Trans. Database Syst., 5(3):260–290, 1980.
15. Maier D., Mendelzon A.O., and Sagiv Y. Testing implications of data dependencies. ACM Trans. Database Syst., 4(4):455–469, 1979.

Implications of Genomics for Clinical Informatics

Mollie Ullman-Cullere, Eugene Clark,
Samuel Aronson
Harvard Medical School – Partners Healthcare Center for Genetics and Genomics, Boston, MA, USA

Synonyms

Biomedical informatics; Bioinformatics; Clinical genetics; Clinical genomics; Medical genetics

Definition

Integration of genetic test results, generated in the clinical laboratory, into the electronic medical record, in a fully structured format enabling enhanced security, contextual views, clinical decision support, pharmacovigilance, disease management, outcomes and quality assessment.

Historical Background

Clinical genetics got its start in 1948 with the founding of the American Society of Human Genetics, which formalized a scientific approach to the study of human genetics [11]. Traditionally clinical genetics requires practitioners to function as data integrators. Like traditional healthcare, tests are ordered and results returned as interpretive reports delivered in paper form. Understanding the composite picture of the *phenotype* and *genotype* of the patient requires transcribing key signs, symptoms, test values and their clinical interpretation into yet another document, contributing more paper to the aggregate patient record. This process will be repeated numerous times over the course of the patient's lifetime, as new health problems are assessed.

Correcting this problem is similar to overcoming the historical barriers to the electronic health record (EHR) [8]. Data standards need to be created and adopted, testing laboratory infrastructure built to structure genetic test results, interfaces developed to send and receive these data, and EHR's and associated clinical decision support tools and knowledgebases also must be enhanced to accept genetic data and inform clinicians of implications in the context of patient care. This introduces new challenges for the underlying database technologies used in the clinical IT infrastructure.

Foundations

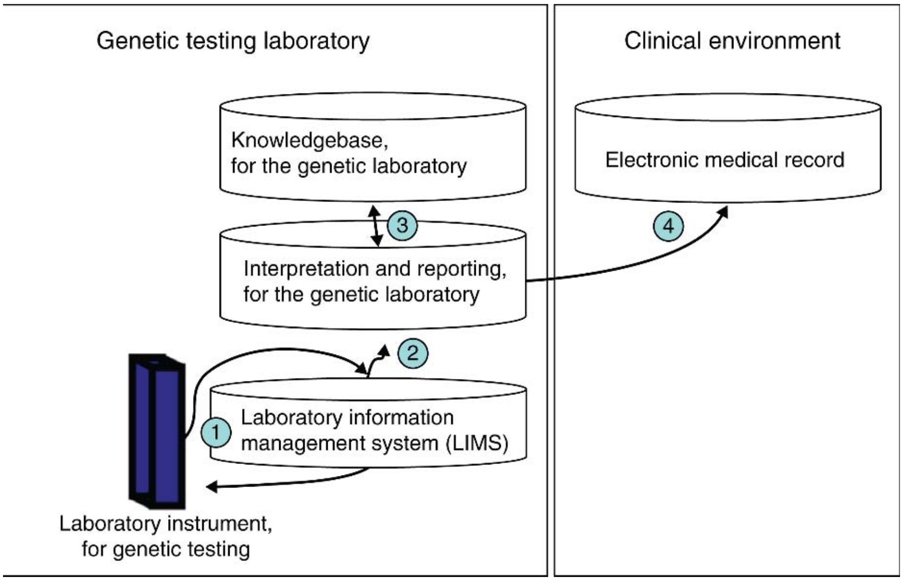
Every person inherits 3 billion base pairs of DNA from each of their parents. Contained in this DNA are regions representing an estimated 22,000 genes. Each of these genes can create RNA that then goes on to create one or more proteins. These proteins participate in complex pathways that perform most of the functions within the body – everything from cell division and food metabolism, to muscle contraction and light perception, just to name a few. When an individual's genomic DNA sequence was compared to the National Center for Biotechnology Information reference nucleotide sequence assembly more than 4.1 million differences were found [6]. Some of these differences, called "DNA variants" are benign in nature, resulting in no measurable phenotypic change. Others result in non-deleterious change, for example, difference in blood type, eye and hair color. While others negatively alter the way protein pathways function within an individual and thereby impact their health. DNA variants, either individually or in combination, are a significant factor, and often an outright cause, of most human disease.

The scope and complexity of this biological genomic infrastructure creates significant knowledge and data management challenges, perhaps the most significant ever encountered. A wide variety of data types such as those associated with DNA, RNA and protein sequences and identified variations have been created. Biological entities such as genes, variants/mutations and proteins will need to be annotated in a manner that provides linkage to public databases for clinical genetic and bioinformatic information, as well as clinical phenotype in the EHR.

Integrating Genetic Data into the Clinical Record

Clinicians and molecular geneticists access genetic information in different ways and for different purposes. This creates new challenges on underlying technologies for supporting new functional requirements. Some key use case scenarios include: establishing the patient context; ensuring data integrity, with patient linkage, throughout the testing process; managing laboratory workflows (often iterative in nature); integrating with instruments; supporting both manual and automated data review; managing quality control and quality assessment; supporting a reporting mechanism; providing decision support for the clinician. In order to meet this challenge a genetic testing laboratory requires: (i) a Laboratory Information Management System (LIMS), to support collection and analysis of raw data, (ii) an interpretation and reporting tool, to assist the geneticist in translating results into clinical implications and report these results in structured form, (iii) a genetic knowledgebase, utilized by the interpretation and reporting tools, to appropriately interpret and structure the results, and (iv) a electronic medical record enhanced to accept and leverage genetic data (see Fig. 1).

Data models, ontologies, and message structure also present a challenge. Properly structuring and annotating genetic data in the EHR is key to effectively displaying, integrating into clinical workflows, and using this data in context of clinical decision support. With this goal in mind, Healthcare Information Technology (HIT) standards are used to create the message between LIMS and EHR's and provide ontological linkage to phenotypic data, while clinical genetic and


Genetic testing laboratory
Clinical environment
Knowledgebase, for the genetic laboratory
Electronic medical record
3
4
Interpretation and reporting, for the genetic laboratory
2
1
Laboratory information management system (LIMS)
Laboratory instrument, for genetic testing

Implications of Genomics for Clinical Informatics. Figure 1. IT Infrastructure required for reporting of structured genetic test results into the electronic medical record.

bioinformatic standards are leveraged for structuring of genetic specific data.

Key Applications

Laboratory Information Management Systems for Genetic Laboratories

Raw data files from instruments used in genetic testing contain highly structured data. It is in the process of summarizing, interpreting and translating these results that the geneticist creates a narrative report, communicating these results to the clinician. Structuring genetic data in the EHR first requires LIMS to manage genetic data during the testing process, even as the data is iteratively collected.

Laboratory Information Management Systems support process flows for individual laboratories. At times it is important to support process flows that span laboratories in which case an umbrella LIMS application, an enterprise LIMS superstructure, is required. An enterprise LIMS superstructure, called the Gateway for Integrated Genomics-Proteomics Applications and Data (GIGPAD) [14], was constructed in a collaboration between Harvard Medical School – Partners HealthCare Center for Genetics and Genomics (HPCGG), Partners HealthCare Information Systems department and Hewlett Packard [5]. GIGPAD is responsible for overall process coordination. It also manages all electronic contact between laboratories and external systems and users. GIGPAD supports multiple laboratories. In some cases, the decision was made to support laboratories by purchasing vendor LIMS and integrating them under the GIGPAD umbrella. In other cases, the decision was made to add support for laboratories by constructing custom LIMS within GIGPAD.

GIGPAD leverages a J2EE architecture backed by an Oracle database. The J2EE platform in general, and EJB in particular, is a good match for this use. GIGPAD needs to support a large number of different types of processes. However, these processes often have common elements. The J2EE platform has enabled the construction of object models that facilitate reuse.

Security is a very important consideration for a LIMS or enterprise LIMS superstructure that handles confidential data. J2EE is a good platform for constructing and enforcing object based security although the platform's generic capabilities need considerable enhancement.

Knowledgebases and Reporting Systems for Genetic Laboratories

The field of clinical genetics maintains knowledgebases called Locus Specific Databases (LSDB), in which DNA variants and associated phenotypes are cataloged [1]. However, LSDB's require manual look-up and synthesis to understand the clinical meaning of a variant and do not utilize HIT standards to code phenotype. The field of bioinformatics utilizes NCBI's dbGaP database to catalog DNA variants and associated phenotype [15]; however, this database contains research data and is not appropriate for clinical care.

In an effort to fill a necessary gap, HPCGG and Partners Healthcare created GeneInsight, a knowledge database that associates genetic variations with clinical annotations using newly extended HIT standards (LOINC, SNOMED, and RxNORM). The variants entered into GeneInsight are validated to ensure that they adhere to the clinical genetics standards for naming variants, using HGVS nomenclature and are valid when compared with locally defined reference sequences. The data stored for each variant includes the DNA change, amino acid change (if applicable), classification (e.g., Pathogenic), source classification (e.g., somatic or germline), and references to other databases including PubMed and dbSNP. In addition, GeneInsight is capable of programmatically deriving the DNA and amino acid change types and flanking sequences.

In addition to variants, GeneInsight also contains records for genes and diseases or conditions. The gene records include references to other databases including NCBI's Gene and PubMed. Reference sequences for the genes are also recorded in GeneInsight and linked to the NCBI's Nucleotide database. Diseases or conditions can represent either a typical disease, such as cystic fibrosis, or a pharmacogenetic condition, such as warfarin metabolism. Medically significant variants are then linked to one or more diseases and/or conditions to establish their clinical context or phenotype.

In order to create the narrative, human readable report (for the clinician) and the machine readable, structured genetic results (for the EHR), a geneticists uses a genetics interpretation and reporting tool which leverages data in a genetic knowledgebase, to create draft reports. Within HPCGG's Laboratory for Molecular Medicine, the Genomic Variant Interpretation Engine (GVIE) is used during the process of reporting genetic test results and co-creates both the narrative report and the structured genetic data for the EHR. GVIE contains definitions of the tests that are run by the HPCGG including the coverage of the test and also defines a series of templates for each disease or condition. The templates are dynamic to allow the automatic insertion of case specific information and can be associated with rules that govern to which cases they are applied. The rules can take into account the specific test that was run, the overall result, the number of variations detected, the classification of those variations, and other information. In addition, the overall result (e.g., Positive) is generated by a separate set of rules that change very infrequently and are driven by the disease's inheritance and the identified variation classification and allele state.

Cerner Corporation has also developed a LIMS and reporting tools for sending structured genetic data into an electronic medical record. Here Cerner uses the Clinical Bioinformatics Ontology (CBO), in lieu of the larger genetic knowledgebase, to report structured genetic variants aligned with NCBI reference sequences [4]. This is a streamlined solution focusing on reporting variants identified during testing.

Integrated Clinical and Genetic Medical Record

A genetically aware clinical medical record effectively integrates highly structured genetic testing results with other laboratory and clinical data, utilizing accepted healthcare informatics standards. These standards include Health Level Seven (HL7) [3] for messaging, Logical Observations Identifiers, Names, Codes (LOINC) [7] as a coding system, and Systematized Nomenclature of Medicine-Clinical Terms (SNOMED CT) [12] and RxNORM [10] as disease and medication terminologies. By using these standards, genetics can be integrated into existing electronic health record databases, leveraging models for storage, retrieval and indexing. Most importantly, these standards aid in the mapping of a patient's genetic information with other clinical data on the patient. As clinical genomics is an emerging field, healthcare informatics standards organizations (including HL7, LOINC, and SNOMED) are in the process of extending their models to meet these needs; therefore, the most up-to-date information can be found on standards organization websites (see Recommended Reading).

Clinical Decision Support

Genomic clinical decision support (CDS) will model existing CDS based on laboratory data. Genetic tests which provide *pharmacogenomic* information will determine how an individual will respond to medication. CDS rules, leveraging this data, would take the form of drug dosage assistance, contraindications, and recommended alternative therapy. Genetic tests which provide *diagnostic* information will feed into problem lists (that can then leverage pre-existing CDS). Genetic tests which identify a patient as being at *increased risk* for disease will feed into other clinical data used to identify high-risk populations and CDS focused on disease management. The precise instantiation models for genomic CDS are based on preexisting models and remain to be thoroughly tested for identification of gaps.

Cross-references

- ▶ Clinical Data Acquisition, Storage and Management
- ▶ Clinical Data and Information Models
- ▶ Clinical Decision Support
- ▶ Electronic Health Record
- ▶ Storage Management

Recommended Reading

1. Fokkema I.F., den Dunnen J.T., and Taschner P.E. LOVD: easy creation of a locus-specific sequence variation database using an “LSDB-in-a-box” approach. Hum. Mutat., 26(2):63–68, 2005.
2. GeneTests – medical genetics information resource including gene reviews, genetic testing laboratory and clinical directories, as well as educational materials. Available online at: http://www.genetests.org.
3. Health Level Seven (HL7) – focusing on messaging, HL7 is an American National Standards Institute (ANSI) accredited. Available online at: http://www.HL7.org/
4. Hoffman M.A. The genome-enabled electronic medical record. J. Biomed. Inform., 40(1):44–46, 2007.
5. HP Supports “Individualized Medicine” Initiative at Partners, Sept. 30, 2003, Available online at: http://www.hp.com/hpinfo/newsroom/press/2003/030930a.html (retrieved on October 5, 2007).
6. Levy S., Sutton G., Ng P.C., Feuk L., Halpern A.L., Walenz B.P., Axelrod N., Huang J., Kirkness E.F., Denisov G., Lin Y., MacDonald J.R., Pang A.W., Shago M., Stockwell T.B., Tsiamouri A., Bafna V., Bansal V., Kravitz S.A., Busam D.A., Beeson K.Y., McIntosh T.C., Remington K.A., Abril J.F., Gill J., Borman J., Rogers Y.H., Frazier M.E., Scherer S.W., Strausberg R.L., and Venter J.C. The diploid genome sequence of an individual human. PLoS Biol., 5(10):2113–2144, 2007.
7. Logical Observation Identifiers Names and Codes (LOINC) – focusing on the pooling of laboratory results and observations, LOINC is ANSI accredited. Available online at: http://www.regenstrief.org/medinformatics/loinc/
8. McDonald C.J. The barriers to electronic medical record systems and how to overcome them. J. Am. Med. Inform. Assoc., 4 (3):213–221, 1997.
9. Online Mendelian Inheritance in Man (OMIM) – a catalog of human genes and genetic disorders. Available online at: http://www.ncbi.nlm.nih.gov/sites/entrez?db=omim.
10. Parrish F., Do N., Bouhaddou O., and Warnekar P. Implementation of RxNorm as a terminology mediation standard for exchanging pharmacy medication between federal agencies. AMIA Annu Symp Proc., 2006:1057, 2006.
11. Rimoin D.L. and Hirschhorn K. History of medical genetics in pediatrics. Pediatr. Res., 56(1):150–9, 2004.
12. Systematized Nomenclature of Medicine-Clinical Terms (SNOMED CT) focusing on clinical terminology, SNOMED is ANSI accredited. Available online at: http://www.ihtsdo.org/
13. The Database of Genotype and Phenotype (dbGaP) – serves as an archive for distribution of clinical study results containing both genotype and phenotype data. Available online at: http://www.ncbi.nlm.nih.gov/sites/entrez?db=gap
14. The Harvard Medical School-Partners HealthCare Center for Genetics and Genomics (HPCGG). Response to the Department of Health and Human Services Request for Information (RFI): Improving Health and Accelerating Personalized Health Care through Health Information Technology and Genomic Information in Population- and Community-based Health Care Delivery Systems. Available online at: http://www.hpcgg.org/News/HPCGG_RFI_Response_1_0.pdf (retrieved on August 14, 2007).
15. Wheeler D.L., Barrett T., Benson D.A., Bryant S.H., Canese K., Chetvernin V., Church D.M., Dicuccio M., Edgar R., Federhen S., Feolo M., Geer L.Y., Helmberg W., Kapustin Y., Khovayko O., Landsman D., Lipman D.J., Madden T.L., Maglott D.R., Miller V., Ostell J., Pruitt K.D., Schuler G.D., Shumway M., Sequeira E., Sherry S.T., Sirotkin K., Souvorov A., Starchenko G., Tatusov R.L., Tatusova T.A., Wagner L., and Yaschenko E. Database resources of the National Center for Biotechnology Information. Nucleic Acids Res., 36: D13–D21, 2008.

Implicit Event

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Definition

In active databases, an implicit event is an event that is implied by an ECA rule definition.

Key Points

ECA rules were developed as an optimization of condition action rules. The performance of rule evaluation was improved by allowing, or even requiring, explicit definition of when rules should be triggered in the form of events. However, it turns out that it is possible to derive events from the condition in a meaningful way in some cases and, thus, there are events that can be implied by the ECA rule definition. For example, if the condition is a logical expression $A = 5 \wedge B = 3$, then a disjunction of the events representing update to the variables can trigger the rule. That is, with implicit events (**IF** $A = 5 \wedge B = 3$ **DO** action) is equivalent to (**ON**$update_A$ **or** $update_B$ **IF** $A = 5 \wedge B = 3$ **DO** action) with explicit events.

Cross-references

- ▶ Atomic Event
- ▶ Composite Event
- ▶ ECA Rules
- ▶ Event
- ▶ Event Detection
- ▶ Event Specification
- ▶ Explicit Event

Imprecise Data

▶ Data Uncertainty Management in Sensor Networks

Imprecise Spatial Queries

▶ Probabilistic Spatial Queries

Imprecise Time

▶ Temporal Indeterminacy

Imputed Data

▶ Synthetic Microdata

IMS Data Model

▶ Hierarchical Data Model

In Silico Experiment

▶ Scientific Workflows

Incoherency Bounds

▶ Replica Freshness

Incomplete Information

Gösta Grahne
Concordia University, Montreal, QC, Canada

Synonyms

Uncertain information; Null values; Indefinite information

Definition

Incomplete information arises in relational databases, when a fact (tuple) has to be inserted in a relation, and values for some required columns are missing. For instance, in an employee database, the phone number of one employee might be missing, as might the address of another employee. There are numerous reasons for such missing information, e.g., the insertion was done through a view, or the incomplete tuple originated from another database that does not record these fields. In information integration and data exchange systems incomplete information is rampant. Note that here the null values only represent unknown, existing values. The other main case, namely the one that the column heading is not applicable to the tuple in question, is not covered here, for a treatment, see e.g., [17].

It is easy to *store* missing or incomplete information, by simply using a symbol, say ⊥, different from the symbols of the domain of the database. As an illustration, consider the following employee database:

Employee	Department	Home Town	Phone no.
Jerry	*Sales*	*New York*	⊥
Elaine	*Accounting*	⊥	*123*
George	*Services*	*New York*	*456*

In the table above, Jerry's phone number is unknown as is Elaine's home address. The third tuple has complete information. So far the picture is quite uncomplicated. The difficulties arise when *queries* are applied to an incomplete database. For instance, suppose the query asks for all employees living in New York. Should Elaine be included in the answer, as it is not ruled out that she lives in New York? Her unknown hometown could very well be New York. It seems that Elaine *might* be in the answer.

For another example, suppose the relation is decomposed into *R*(Employee, Department, Home Town) and *S*(Home Town, Phone no.) Now, joining back *R* and *S*, at least the original relation should be recovered. However, it is not clear how to join the *R*-tuple (*Elaine, Accounting*, ⊥) with the *S*-tuple (⊥, *123*). Nevertheless, whatever Elaine's home address is, the null-value in the *R*-tuple must clearly be the same as the null-value in the *S*-tuple, as they both represent Elaine's unknown home town. Therefore, the tuple (*Elaine, Accounting*, ⊥, *123*) should *surely* be in the result of the join.

Historical Background

The enormously successful relational model emerged out of algebra and logic, starting with Codd's fundamental 1970's papers. By the end of the decade, the problem of null values was noted and resulted in some early efforts not covered here, see e.g., [17,16]. Around this time Ray Reiter also promulgated the idea of "Proof theoretical" vs. "Model Theoretical" interpretation of the relational model (see references in [12]). Roughly, one could say that in proof theory the database is a set of ground facts in First Order Logic, and the query (also a logical formula) answers are something to deduce from the theory by proof-theoretic means. In "Model Theory" on the other hand, the database is a finite structure, and the queries are algebraic operators applied to the database. Indeed, Reiter proposed a first order interpretation of the null value: Since a regular tuple $(a, b) \in r$ is interpreted as a ground fact $R(a, b)$, it is plausible to interpret $(a, x) \in r$, where x is a null value, as the logical sentence $\exists x : R(a, x)$. Reiter then developed query answering algorithms for his (existentially) extended relational theories [12].

At the same time, relational theory started developing, creating its own apparatus, containing lossless joins, tableaux and the chase, in addition to many other useful tools and concepts, such as the "crown jewel" relational algebra. Working along these lines, T. Imielinski and W. Lipski came out in 1981 with a *VLDB* conference abstract of their landmark paper *Incomplete Information in Relational Databases*, published in *J. ACM* in 1984 [8].

First, Imielinski and Lipski call a relation containing null values a *table* (as opposed to a relation). One assumes two disjoint countably infinite sets of *constants*, denoted *a, b, c, Jerry, Elaine*,...,and of *variables*, denoted *x, y, z*,.... The table in the introduction, could be written (concisely) as {(*Jerry, Sales, New York, x*),(*Elaine, Accounting, y, 123*), (*George, Services, New York, 456*)}

Second, [8] adopts the *possible worlds* interpretation of incomplete databases: an *incomplete database is a set of complete databases*, one of which is or corresponds to the real world. In other words, the knowledge about the real world, that the incomplete database has is exactly this set. In the sequel (usually infinite) incomplete databases (sets of complete databases) are denoted by $\mathcal{X}, \mathcal{Y}, \ldots$.

Tables represent incomplete databases through valuations, that instantiate the null values to ordinary, known values. Formally, a *valuation* is a mapping from the variables to the constants, that is identity on the constants. Valuations are extended to tuples and tables in the obvious homeomorphic way. Now a database r represents one of the possible worlds of table T, if there is a valuation v, such that $r = v(T)$. For instance, let $T = \{(a,x), (b, y)\}$. Pretend for a moment that domain of constants is finite and only includes a and b. Then T represents $\mathcal{X}$, where $\mathcal{X} = \{\{(a, b), (b, b)\}, \{(a, a), (b, b)\}, \{(a, b), (b, a)\}, \{(a, a), (b, a)\}\}$. Thus $Rep(T)$, the incomplete database $\mathcal{X}$, represented by a table T, is defined as

$$Rep(T) = \{r : r \supseteq v(T), \text{for some valuation } v\}. \quad (1)$$

Note that the *open world* assumption is made, when requiring only $r \supseteq v(T)$. It means that any fact not recorded in all possible databases is considered unknown. Note further that under an open world assumption, a *complete* database r actually represents all supersets of r. Since nothing is surely false, it is clear that querying under the open world assumption cannot include negation. In a *closed world* interpretation, on the other hand, any fact *not* recorded in the database, or not instantiable from a tuple in a table is considered false. For tables T, the closed world interpretation of the possible databases it represents is defined as $Rep_{\text{cwa}}(T) = \{r : r = v(T)$, for some valuation $v\}$. This means that a tuple that does not belong to any $r \in Rep_{\text{cwa}}(T)$ is considered false.

The main insight when processing a query q on an incomplete database is as follows: since the "real" database r is one within a set $\mathcal{X}$, the "real" answer should ideally be $q(r)$. However, given only $\mathcal{X}$, what can be captured with query q is the set $q(\mathcal{X}) = \{q(r) : r \in \mathcal{X}\}$. In particular, if $\mathcal{X}$ is represented by a table T, it would be desirable to find a table, denote it $\hat{q}(T)$, such that

$$Rep(\hat{q}(T)) = q(Rep(T)). \tag{2}$$

This is where the technical development can begin.

Foundations

First, note that the open world assumption itself is causing a subtle difficulty when trying to satisfy the commutativity requirement (2). Consider the table T containing a single tuple (a, b). After applying a selection $\sigma_{A=a}$ on T one would (correctly) expect that the tuple (a, b), and nothing else, is in the resulting table $\widehat{\sigma_{A=a}}(T)$. However, (2) is not satisfied, since for instance (c, d) occurs in some $r \in Rep(\widehat{\sigma_{A=a}}(T))$, whereas $(c, d) \notin r$, for all $r \in \sigma_{A=a}(Rep(T))$, as such tuples have been dropped by the selection. Unless the closed world assumption is adopted – in which case clearly $Rep_{\text{cwa}}(\widehat{\sigma_{A=a}}(T)) = \sigma_{A=a}(Rep_{\text{cwa}}(T))$ – condition (2) has to be relaxed.

The key concept for achieving this purpose is the notion of *certain answer*. Let q be a query and $\mathcal{X}$ an incomplete database. The certain answer to q on $\mathcal{X}$ consists of those tuples that are in $q(r)$, for *every* possible database $r \in \mathcal{X}$, that is, the certain answer is $\cap_{r \in \mathcal{X}} q(r) = \cap(q(\mathcal{X}))$.

Given a fixed query language, two incomplete databases are not distinguishable if they give the same certain answer to every query in the language. Formally, let $\mathcal{Q}$ be a query language (the set of all queries expressible in the language). Then incomplete databases $\mathcal{X}$ and $\mathcal{Y}$ are said to be $\mathcal{Q}$*-equivalent*, if $\cap(q(\mathcal{X})) = \cap(q(\mathcal{Y}))$, for all $q \in \mathcal{Q}$. This $\mathcal{Q}$-equivalence is denoted $\mathcal{X} \equiv_{\mathcal{Q}} \mathcal{Y}$. There is another equivalence relation on incomplete databases called *coinitiality*. Now $\mathcal{X}$ is coinitial with $\mathcal{Y}$, in symbols $\mathcal{X} \approx \mathcal{Y}$, if $\mathcal{X}$ and $\mathcal{Y}$ have the same minimal (with regard to subset) elements. Let $\mathcal{Q}_P$ be the set of all queries expressible in *positive* relational algebra (no negation, no inequalities in selection conditions), and let $\mathcal{Q}_{RA}$ be the set of *all* queries expressible in the full relational algebra. It turns out that provably $\mathcal{X} \equiv_{\mathcal{Q}_P} \mathcal{Y}$ iff $\mathcal{X} \approx \mathcal{Y}$, and $\mathcal{X} \equiv_{\mathcal{Q}_{RA}} \mathcal{Y}$ iff $\mathcal{X} = \mathcal{Y}$ [8,4].

Therefore, if the open world assumption if followed, the best approximation of (2) is to weaken it to

$$Rep(\hat{q}(T)) \equiv_{\mathcal{Q}} q(Rep(T)) \tag{3}$$

for some query language $\mathcal{Q}$. This led [8] to the notion of *representation system*. Let $\boldsymbol{T}$ be a class of tables, *Rep* the interpretation function, and $\mathcal{Q}$ a query language. Then the triple $(\boldsymbol{T}, Rep, \mathcal{Q})$ is a *representation system* if for all $T \in \boldsymbol{T}$, and all $q \in \mathcal{Q}$, there is a $\hat{q}(T) \in \boldsymbol{T}$, such that (3) is satisfied. Then the certain answer $\cap(q(Rep(T)))$ is clearly equal to $\cap(Rep(\hat{q}(T)))$. This means that the certain answer can be extracted from $\hat{q}(T)$, in some cases efficiently, as will be seen below.

The original paper [8] considers the classes consisting of Codd Tables, Naive Tables, and Conditional Tables. Other classes of tables can be found in [2,4,9,14]. Here, the attention is restricted to the original three table classes.

A *Codd table* is a table where each occurrence of a null value is represented by the same symbol (as for example in the table in the first section of this entry). Let for instance the symbol $\perp$ denote a null value. Evaluating a query q (i.e., computation of $\hat{q}(T)$ in (3)) proceeds as in regular relational algebra, with the additional evaluation rules saying that $\perp \neq \perp$, and $\perp \neq a$, for all constants a, as long as the selection conditions are atomic, i.e., of the form $\sigma_{A=a}$ or $\sigma_{A \neq a}$. This is in fact the solution proposed by Codd, and currently implemented in most commercial database management systems. However, the largest query language that Codd tables can support consists of queries expressible in relational algebra using only projection and selection, where, notably, inequalities are allowed in selections. Using any more operators from the relational algebra makes it impossible to satisfy (3), that is, for some such queries q there is no Codd table $\hat{q}(T)$ satisfying (3). The same holds under the closed world assumption. In addition to this, selections can have arbitrary Boolean combinations of atomic selection conditions, but then testing whether a tuple satisfies such a condition becomes coNP-complete. This is due to the fact that any propositional formula can be encoded as a compound selection formula. Note however, that the coNP-completeness depends on the fact that the query is part of the input (expression complexity). For data complexity, the evaluation can be carried out in time polynomial in the size of the table.

A *Naive table* is a table using *distinguishable* nulls, denoted by variables x,y,.... Query evaluation uses the rules $x \neq a$, $x \neq y$, $x = x$, for all variables x, y and constants a. Thus the tuple (*Elaine, Accounting*,$\perp$) would indeed join with the tuple ($\perp$,*123*), resulting in tuple (*Elaine, Accounting*,$\perp$,*123*). These tuples would of course be represented as (*Elaine, Accounting*,*y*), and (*y*,*123*). It turns out that if $\mathcal{Q}_P$ is chosen as query language, then $(T_N, Rep, \mathcal{Q}_P)$, where T_N is the class of all Naive tables, indeed forms a representation system. It has also been shown that Naive tables can handle recursion, i.o.w. one can add any positive datalog program as a query, and still be able to $\approx$-represent the result [4]. More generally, one can note that query evaluation, i.e., computation of $\hat{q}(T)$, proceeds by treating the null-values as constants, pairwise distinct, and distinct from all the "real" constants. Thus query evaluation in Naive tables has the same computational complexity as in the regular case. A similar result was also independently proved in [15].

Much has been written about certain answers in the database literature, but not always with complete transparency. Recall that the *certain answer* to a query q on $\mathcal{X}$, are those tuples that are in the answer to q for *every* possible database $r \in \mathcal{X}$. In other words, the certain answer to q on $\mathcal{X}$ is $\cap(q(\mathcal{X}))$. Note that $\hat{q}(T)$ is a Naive table that satisfies (3), for all positive datalog programs, or algebraic expressions in $\mathcal{Q}_P$. Conveniently, the certain answer can be obtained by retaining all variable-free tuples in $\hat{q}(T)$, as it is easily seen that $\mathcal{X} \approx \mathcal{Y}$ implies $\cap\mathcal{X} = \cap\mathcal{Y}$. The certain answer however looses some information from $\hat{q}(T)$. For a simple example, similar to the one in the introduction, let $T = \{(a, x, c)\}$ with schema (A, B, C). The certain answer to both $\pi_{A,B}(Rep(T))$ and $\pi_{B,C}(Rep(T))$ is empty. On the other hand, the certain answer to $\pi_{A,C}(\pi_{A,B}(Rep(T)) \bowtie \pi_{B,C}(Rep(T))) = \{(a, c)\}$. Here $\widehat{\pi_{A,B}}(T) = \{(a, x)\}$, $\widehat{\pi_{B,C}}(T) = \{(x, c)\}$ and $\{(a, x)\widehat{\bowtie} \{(x, c)\} = \{(a, c)\}$. Note how variables can be shared over several tables, above the x the left-hand side of the join is the same as the x on the right-hand side of the join. (On the other hand, for instance $\{(a, x)\widehat{\bowtie} \{(y, c)\} = \emptyset$.) Thus, if the query answer is to be used as a *view* for further querying, the answer should consist of all of $\hat{q}(T)$, not just the variable-free tuples. This aspect has been emphasized in [5]. Furthermore, as Lipski has shown in an all but forgotten paper [11], if query evaluation is to be *uniformly recursive* (such as the one for Naive tables), all tuples in $\hat{q}(T)$ need to be stored in the view.

Conditional tables. Naive tables form a representation system for the positive fragment of relational algebra (and for positive recursion). However, allowing set difference or inequalities in selection conditions in the queries might produce results not expressible as Naive tables. Consider for instance $\sigma_{A \neq a}(Rep(T))$, where $T = \{(a, b),(c, d),(x, e)\}$. A moments reflection will reveal the fact that $\sigma_{A \neq a}(Rep(T)) = \{\{(c, d)\},\{(c, d),(b, e)\},\{(c, d),(d, e)\},\ldots\}$. It is easy to see that there is no Naive table that can represent this set. The problem is that the minimal elements in this set either has one tuple (when $x = a$), or two tuples (when $x \neq a$). The minimal elements in $Rep(T)$, for any (non-redundant) Naive table T, each have the same number of tuples. This problem can be overcome by using a stronger class of tables, albeit on the expense of tractability of query evaluation. This stronger class is called Conditional tables, here denoted T_C. A Conditional table is like a Naive Table, with the addition that each tuple has an associated condition, formed by a Boolean combination of atoms of the form $x = y$, $x \neq y$, $x = a$,$x \neq a$, $a = b$, $a \neq b$, for constants a,b and variables x, y. Note that for two distinct constants a and b, condition $a = b$ is tautologically false, and condition $a \neq b$ is tautologically true. In our previous example, $\sigma_{A \neq a}(Rep(T))$ can be represented by the Conditional table $U = \{(c, d);c \neq a,(x, e);x \neq a)\}$. The incomplete database represented by a conditional table T is defined as

$$\begin{aligned} Rep(T) = \{r \supseteq v(T) : v \text{ is a valuation}, v(T) = \\ \{v(t) : t \in T \text{ and } v \text{ makes the condition} \\ \text{in } t \textit{ true}\}\} \end{aligned} \tag{4}$$

For example, if v is the valuation $x \mapsto a$, and v' the valuation $x \mapsto b$, then, for U as above, $v(U) = \{(c, d)\}$ as the condition of the second tuple (x, d); $x \neq a$ becomes false. On the other hand, $v'(U) = \{(c, d),(b, e)\}$, since $v'(x \neq a)$ becomes $b \neq a$, which is true. Set difference is treated in a similar manner, for instance, subtracting $\{(c, b),(e, b)\}$ from $\{(x, b)\}$ results in the conditional table $\{(x, b);x \neq c \wedge x \neq e\}$. All in all, it holds that $(T_C, Rep, \mathcal{Q}_{P\neq})$, is a representation system. Here $\mathcal{Q}_{P\neq}$ denotes the query language obtained from $\mathcal{Q}_P$, by allowing inequalities in selection conditions. If set difference is to be incorporated, the closed world assumption has to be adopted. In this case it holds that $(T_C, Rep_{cwa}, \mathcal{Q}_{RA})$, is a representation system, actually satisfying the stronger condition (2). Positive datalog recursion can also be added both under the open and the closed world assumption.

For the computational complexity of query evaluation, recall that query evaluation in the system $(T_N, Rep, \mathcal{Q}_P)$ is polynomial in the number of tuples in T. Conditional tables are more complex. For all $q \in \mathcal{Q}_{RA}$ and conditional tables T, the conditional table $\hat{q}(T)$ can be computed in polynomial time. However, the conditions in table $\hat{q}(T)$ can have a convoluted structure. Thus testing whether a tuple t is in the certain answer, i.e., if $t \in \cap(Rep(\hat{q}(T)))$ is a coNP-complete problem. This result holds even if T is a Codd table, in which case a query $q \in \mathcal{Q}_{RA}$ is needed to get the lower bound (here the fact that $\hat{q}(T)$ is not a Codd table is ignored). On the other hand, for an arbitrary table $T \in T_C$, the query can be identity. In other words, the set $\cap\ (Rep(T))$ has a coNP-complete membership test. These, and further complexity results can be found in [2].

The theory of representation systems has also been extended to incorporate dependencies, see [8,4]. Let $\mathcal{X}$ be an incomplete database, and Σ a set of equality and weakly acyclic tuple generating dependencies. Denote by $\Sigma(\mathcal{X})$ the set of all minimal relations s, such that $s \supseteq r$, for some $r \in \mathcal{X}$, and s satisfies all dependencies in Σ. It has been shown that for all Naive tables T, there is a Naive table $\Sigma(T)$, such that $Rep(\Sigma(T)) \approx \Sigma(Rep(T))$. If conditional tables are used, the coinitiality can be replaced by equality.

The relationship between tables and constraint databases is explored in [13]. The relationship between tables and probabilistic databases, and between tables and data provenance is elegantly captured in the semiring framework of [7].

Key Applications

Three important applications of incomplete information will be discussed here. The first one is the problem of *view updates*. In this scenario, a view is defined by a query q, but the view is virtual, and only the database is materialized. As queries almost never are one-to-one functions, when a tuple t is inserted through a virtual view $q(r)$, it has to be translated to an insertion of a tuple, say $\hat{t}$, into r, s.t. $q(r \cup \{\hat{t}\}) = q(r) \cup \{t\}$. There is also a further requirement, that essentially guarantees the "minimality" of the change caused be the insertion of $\hat{t}$. Since there in general are several, sometimes infinitely many, insertions $\hat{t}$ that qualify, there actually is a set of possible databases $\{r \cup \{\hat{t}\} : \hat{t}\ qualifies\}$. For example, for $q = R \bowtie S$, where $R(A, B)$ and $S(B, C)$, the deletion of a tuple (a, b, c) from a view $q(r, s)$, can be accomplished by either deleting the tuple (a, b) from r, or deleting the tuple (b, c) from s. This can easily be achieved if the database is stored as a conditional table, by simply replacing the tuples (a, b) in r, and (b, c) in s, with the conditional tuples (a, b); $x = 1$, and (b, c); $x \neq 1$, where x is an arbitrary fresh variable.

The views in the view update scenario are virtual. If views are materialized, they can be used to speed up query processing. The case where all views are materialized, and the database virtual is the classic information integration scenario. The views represent data sources, and the database schema represents the integrated schema that queries are formulated over. Intuitively, one can imagine that the integrated database exists, the view-defining queries are executed, and the views are accordingly populated. After this, the integrated database disappears. It is easy to see that a set of materialized views represent a set of possible databases, each satisfying the requirement that the current content of the views can be derived from it. Note that there is an open world assumption, as the requirement is that the view tuples are a subset (not necessarily proper) of $q(r)$, for all view definitions q and possible databases r. The closed world assumption would require equality, not just subset. For query answers, the certain answer is usually desired. The certain answer means in this context the set of tuples that are in the query result, for every possible database. To answer queries over the integrated schema, one can either rewrite the query in terms of the view-schemas, or reconstruct a representation of all the possible databases, and evaluate the original query on this representation. It perhaps speaks for the robustness of the concept of conditional tables, that they can do the job of representing the set of possible databases, whenever the views are queries in $\mathcal{Q}_{P\neq}$, or in $\mathcal{Q}_{RA}$ if the closed world assumption is adopted [1,6].

The last application is that of *data exchange*. The setting is a peer-to-peer system, where each peer has a relational database and wants to exchange tuples with other peers. The schemas of any two peers, say p_1 and p_2 are usually different from each other, so the data has to be converted when exchanged. This is achieved by defining a mapping from p_1 to p_2. This mapping could for instance be expressed as an equation $q_1(p_1) \subseteq q_2(p_2)$. When we export data from p_1 to p_2, we evaluate q_1 on p_1, and make sure that the resulting tuples are in $q_2(p_2)$. Since the tuples have to be inserted into p_2, it is easy to see that this is similar to the familiar problem of view updates, and thereby also incomplete information. For a very simple example, suppose the schema of p_1 is $R(A, B)$, and the schema of p_2 is $S(A, C)$. Consider then the equation $\pi_A(R) \subseteq \pi_A(S)$. If r contains a tuple (a, b), it is clear, that in order to satisfy the equation, a tuple (a, x) has to be in s. It has for example been shown [10], that if $q_1 \in \mathcal{Q}_P$ and q_2 only uses projection, then given null-free instances of p_1 and p_2, there is a naive table $p_2{}'$, containing p_2, representing all minimal solutions to the equation, that is $q_1(p_1) \subseteq q_2(r)$, for all $r \in Rep(p'{}_2)$, and the equation is not satisfied by any proper subsets of these r's.

Future Directions

It is clear that new applications, new data models, and new query languages will encounter the problem of incomplete information. In order to not "re-invent the wheel," any solution should build on the foundation laid in [8]. A step in this direction has for instance been taken in [3].

Recommended Reading

1. Abiteboul S. and Duschka O.M. Complexity of answering queries using materialized views. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 254–263.
2. Abiteboul S., Kanellakis P.C., and Grahne G. On the representation and querying of sets of possible worlds. Theor. Comput. Sci., 78(1):158–187, 1991.
3. Abiteboul S., Segoufin L., and Vianu V. Representing and querying XML with incomplete information. ACM Trans. Database Syst., 31(1):208–254, 2006.
4. Grahne G. The Problem of Incomplete Information in Relational Databases. Springer, 1991.
5. Grahne G. and Kiricenko V. Towards an algebraic theory of information integration. Inf. Comput., 194(2):79–100, 2004.
6. Grahne G. and Mendelzon A.O. Tableau Techniques for Querying Information Sources through Global Schemas. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 332–347.
7. Green T.J., Karvounarakis G., and Tannen V. Provenance semirings. In Proc. 26th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2007, pp. 31–40.
8. Imielinski T. and Lipski W. Incomplete information in relational databases. J. ACM, 31(4):761–791, 1984.
9. Imielinski T. and Vadaparty K.V. Complexity of query processing in databases with OR-objects. In Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1989, pp. 51–65.
10. Libkin L. Data exchange and incomplete information. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 60–69.
11. Lipski W. Jr. On relational algebra with marked nulls. In Proc. 3rd ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1984, pp. 201–203.
12. Reiter R. A sound and sometimes complete query evaluation algorithm for relational databases with null values. J. ACM. 33(2):349–370, 1986.
13. Revesz P.Z. Introduction to Constraint Databases. Springer, 2002.
14. Sarma A.D., Benjelloun O., Halevy A.Y., and Widom J. Working models for uncertain data. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 7.
15. Vardi M.Y. Querying logical databases. J. Comput. Syst. Sci. 33 (2):142–160, 1986.
16. Vassiliou Y. Null values in data base management: a denotational semantics approach. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 162–169.
17. Zaniolo C. Database relations with null values. In Proc. 1st ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1982, pp. 27–33.

Inconsistent Databases

LEOPOLDO BERTOSSI
Carleton University, Ottawa, ON, Canada

Definition

An inconsistent database is a database instance that does not satisfy those integrity constraints that have been declared together with the schema of the database.

Historical Background

Already in the classical and seminal paper by E. F. Codd [5] on the relational data model it is possible to find the notions of integrity constraint and consistency of a database. The idea of *consistent query answering*, consisting in characterizing and computing semantically correct answers to queries in inconsistent databases, was explicitly introduced in [1].

Foundations

A database can be seen as a model, i.e., as a simplified, abstract description, of an external reality. In the case of relational databases, one starts by choosing certain predicates of a prescribed arity. The *schema* of the database consists of this set of predicates, possibly *attributes*, which can be seen as names for the arguments of the predicates, together with an indication of the domains where the attributes can take their values. Having chosen the schema, the representation of the external reality is given in terms of relations, which are extensions for the predicates in the schema. This set of relations is called an *instance* of the schema.

For example, relational database for representing information about students of a university might be based on the schema consisting of the predicates *Students(StNum, StName)* and *Enrollment(StName,Course)*. The attribute *StNum* is expected to take numerical values; *StName*, character string values; and *Course*, alphanumeric string values. In Fig. 1 there is a possible instance for this schema.

In order to make the database a more accurate model of the university domain (or to be in a more accurate correspondence with it), certain conditions are imposed on the possible instances of the database. Those conditions are intended to capture more meaning from the outside application domain. In consequence, these conditions are called *semantic constraints* or *integrity constraints* (ICs). For example, a condition could be that, in every instance, the student name functionally depends upon the student number, i.e., a student number is assigned to at most one student name. This condition, called a *functional dependency* (FD), is denoted with $StuNumber \rightarrow StuName$, or $Students : StuNumber \rightarrow StuName$, to indicate that this dependency should hold for attributes of relation *Students*. Actually, in this case, since all the attributes in the relation functionally depend on *StuNum*, the FD is called a *key constraint.*

Integrity constraints can be declared together with the schema, indicating that the instances for the schema should all satisfy the integrity constraints. For example, if the functional dependency $Students : StuNumber \rightarrow StuName$ is added to the schema, the instance in Fig. 1 is consistent, because it satisfies the FD. However, the instance in Fig. 2 is *inconsistent.* This is because this instance does not satisfy, or, what is the same, violates the functional dependency (the student number *101* is assigned to two different student names.

Functional dependencies are particular cases of integrity constraints. It is also possible to consider with the schema a *referential integrity constraint* that requires that every student (number) in the relation *Enrollment* appears, associated with a student name, in relation *Students*, the official "table" of students. This is denoted with $Enrollement[StNum] \subseteq Students[StNum]$. If this IC is considered in the schema, the instance in Fig. 1 is inconsistent, because student *105* does not appear in relation *Students.* However, if only the referential constraint were in the schema, the instance in Fig. 2 would be consistent.

In can be seen that the notion of consistency is relative to a set of integrity constraints. When a database is said to be inconsistent, it is meant that the particular instance of the database at hand is inconsistent.

The two particular kinds of integrity constraints presented above and also other forms of ICs can be easily expressed in the language of predicate logic. For example, the FD above can be expressed by the symbolic sentence

$$\forall x \forall y \forall z((Students(x,y) \wedge Students(x,z)) \rightarrow y = z), \qquad (1)$$

Students	StuNum	StuName
	101	john bell
	102	mary stein
	104	claire stevens
	107	pat norton

Enrollment	StuNum	Course
	104	comp150
	101	comp100
	101	comp200
	105	comp120

Inconsistent Databases. Figure 1. A database instance.

Students	StuNum	StuName
	101	john bell
	101	joe logan
	104	claire stevens
	107	pat norton

Enrollment	StuNum	Course
	104	comp150
	101	comp100
	101	comp200

Inconsistent Databases. Figure 2. Another instance.

whereas the referential constraint above can be expressed by

$$\forall x \forall y (Enrollment(x,y) \rightarrow \exists z Students(x,z)). \quad (2)$$

Notice that this language of predicate logic is determined by the database schema, whose predicates are now being used to write down logical formulas. We may also use "built-in" predicates, like the equality predicate. Thus, ICs can be seen as forming a set Σ of sentences written in a language of predicate logic.

A database instance can be seen as an *interpretation structure D* for the language of predicate logic that is used to express ICs. This is because an instance has an underlying domain and (finite) extensions for the predicates in the schema. Having the database instance as an interpretation structure and the set of ICs as a set of symbolic sentences is crucial, and makes it possible to simply apply the notion of satisfaction of a formula by a structure of first-order predicate logic [6]. In this way, the notion of satisfaction of an integrity constraint by a database instance is a precisely defined notion. The database instance D is consistent if and only if it satisfies Σ, which is commonly denoted with $D \models \Sigma$.

Since it is usually assumed that the set of ICs is consistent as a set of logical sentences, in databases the notion of consistency becomes a condition on the database instance. Thus, this use of the term "consistency" differs from its use in logic, where consistency characterizes a set of formulas.

Inconsistency is an undesirable property for a database. In consequence, one attempts to keep it consistent as it is subject to updates. There are a few ways to achieve this goal. One of them consists in declaring the ICs together with the schema, and the database management system (DBMS) will take care of the database maintenance, i.e., of keeping it consistent. This is done by rejecting transactions that may lead to a violation of the ICs. For example, the DBMS should reject the insertion of the tuple (*101*, *sue jones*) into the instance in Fig. 1 if the FD (1) was declared with the schema (as a key constraint). Unfortunately, commercial DBMSs offer limited support for this kind of database maintenance.

An alternative way of keeping consistency is based on the use of triggers (or active rules) that are stored in the database. The reaction to a potential violation is programmed as the action of the trigger: if a violation is about to be produced or is produced, the trigger automatically reacts, and its action may reject the violating transaction or compensate it with additional updates, to make sure that at the end, consistency is reestablished. Consistency can also be enforced through the application programs that interact with the DBMS. However, the correctness of triggers or application programs with respect to (with regard to) ensuring database consistency is not guaranteed by the DBMS.

It is the case that, for whatever reasons, databases may become inconsistent, i.e., they may violate certain ICs that are considered to be relevant to maintain for a certain application domain. This can be due to several reasons, e.g., poorly designed or implemented applications that fail to maintain the consistency of the database, or ICs for which a DBMS does not offer any kind of support, or ICs that are not enforced for better performance of application programs or DBMSs, or ICs that are just assumed to be satisfied based on knowledge about the application domain and the kind of updates on the database. It is also possible to have a legacy database on which semantic constraints have to be imposed; or more generally, a database on which imposing new constrains depending on specific needs, e.g., user constraints, becomes necessary.

In the area of data integration the satisfaction of desirable ICs by a database is much more difficult to achieve. One can have different autonomous databases that are separately consistent with regard to their own, local ICs. However, when their data is integrated into a single database, either material or virtual, certain desirable global ICs may not be satisfied. For example, two university databases may use the same numbers for students. If their data is put together into an integrated database, a student number might be assigned to two different students.

When trying to use an inconsistent database, the application of some *data cleaning* techniques may be attempted, to cleanse the database from data that participates in the violation of the ICs. This is done sometimes. However, data cleaning is a complex and non-deterministic process; and it may also lead to the loss of information that might be useful. Furthermore, in certain cases like virtual data integration, where the data stays at the autonomous data sources, there is no way to change the data without ownership of the sources.

One might try to live with inconsistent databases. Actually, most likely one will be forced to keep using it, because there is still useful information in it. It is also likely that most of the information in it is

Students1	StuNum	StuName
	101	john bell
	104	claire stevens
	107	pat norton

Students2	StuNum	StuName
	101	joe logan
	104	claire stevens
	107	pat norton

Inconsistent Databases. Figure 3. Two repairs.

somehow consistent. Thus, the challenge consists in retrieving from the database only information that is consistent. For example, one could pose queries to the database at hand, but expecting to obtain only answers that are semantically correct, i.e., that are consistent with the ICs. This is the problem of *consistent query answering* (CQA).

The notion of consistency of a database is a holistic notion, that applies to the entire database, and not to portions of it. In consequence, in order to pursue this idea of retrieving consistent query answers, it becomes necessary to characterize the consistent data in an inconsistent database first. The idea that was proposed in [1] is as follows: the consistent data in an inconsistent data is the one that is invariant under all possible way of restoring the consistency by performing minimal changes on the initial database. That is, no matter what minimal consistency restoration process is applied to the database, the consistent data stays in the database. Each of the consistent versions of the original instance obtained by minimal changes is called a *minimal repair*, or simply, a *repair*.

It becomes necessary to be more precise about the meaning of minimal change. In between, a few notions have been proposed and studied (cf. [2–4] for surveys of CQA). Which notion to use may depend on the application. The notion of minimal change can be illustrated using the definition of repair given in [1]. First of all, a database instance D can be seen as a finite set of ground atoms (or database tuples) of the form $P(\bar{c})$, where P is a predicate in the schema, and $\bar{c}$ is a finite sequence of constants in the database domain. For example, *Students(101, john bell)* is an atom in the database. Next, it is possible to compare the original database instance D with any other database instance D' (of the same schema) through their symmetric difference $D\Delta D' = \{A \mid A \in (D \setminus D') \cup (D' \setminus D)\}$.

Now, a repair of an instance D with regard to a set of ICs Σ is defined as an instance D' that is consistent, i.e., $D' \models \Sigma$, and for which there is no other consistent instance D'' that is closer to D than D', i.e., for which it holds $D\Delta D'' \underset{\neq}{\subseteq} D\Delta D'$. For example, the database in Fig. 2 has two repairs with regard to the FD (1). They are shown in Fig. 3 and are obtained each by deleting one of the two conflicting tuples in relation *Students* (relation *Enrollment* does not change).

Having defined the notion of repair, a *consistent answer* from an instance D to a query $Q(\bar{x})$ with regard to a set Σ of ICs is defined as an answer $\bar{c}$ to Q that is obtained from every possible repair of D with regard to Σ. That is, if the query Q is posed to each of the repairs, $\bar{c}$ will be returned as a usual answer to Q from each of them.

For example, if the query $Q_1(x, y) : Students(x, y)$, asking for the tuples in relation *Students*, is posed to the instance in Fig. 2, then *(104, claire stevens)* and *(107, pat norton)* should be the only consistent answers wrt the FD (1). Those are the tuples that are shared by the extensions of *Students* in the two repairs. Now, for the query $Q_2(x) : \exists y Students(x, y)$, i.e., the projection on the first attribute of relation *Students*, the consistent answers are *(101)*, *(104)* and *(107)*.

There might be a large number of repairs for an inconsistent database. In consequence, it is desirable to come up with computational methodologies to retrieve consistent answers that use only the original database, in spite of its inconsistency. Such a methodology that works for particular syntactic classes of queries and ICs, was proposed in [1]. The idea is to take the original query Q that expects consistent answers, and syntactically transform it into a new query Q', such that the *rewritten query* Q', when posed to the original database, obtains as usual answers the consistent answers to query Q. The essential question is, depending on the language in which Q is expressed, what kind of language is necessary for expressing the rewriting Q'. The answer to this question should also depend on the kind of ICs being considered.

The idea behind the rewriting approach presented in [1] can be illustrated by means of an example. The consistent answers to the query $Q_1(x, y) : Students(x, y)$ above with regard to the FD (1) can be obtained by posing the query $Q'(x, y) : Students(x, y) \wedge \neg\exists z (Students(x, z) \wedge z \neq y)$ to the database. The new query collects as normal answers those tuples where the value of the first attribute is not associated to two different values of the second attribute in the relation. It can be seen that the answer set for the new query can be computed in polynomial time in the size of the database.

In this example, a query expressed in first-order predicate logic was rewritten into a new query expressed in the same language. It has been established in the literature that, for complexity-theoretic reasons, a more expressive language to do the rewriting of a first-order query may be necessary. For example, it may be necessary to do the rewritings as queries written in expressive extensions of Datalog [2–4].

If a database is inconsistent wrt referential ICs, like the instance in Fig. 1 and the constraint in (2), it is natural to restore consistency by deleting tuples or inserting tuples containing *null values* for the existentially quantified variables in the ICs. For example, the tuple (*105*, *comp120*) could be deleted from *Enrollment* or the tuple (*105*, *null*) could be inserted in relation *Students*. This requires a modification of the notion of repair and a precise semantics for satisfaction of ICs in the presence of null values [2,4].

Key Applications

Key applications of *consistent query answering* (CQA) are still missing. Applications to virtual data integration look promising, and also applications to data cleaning.

Future Directions

There are many open problems and research directions, among them, and most prominently, the development of key applications of CQA. A more precise characterization of the languages that are needed for doing CQA using query rewriting is also missing. It also becomes necessary to shed more light on the right kind of repair semantics to use depending on the application. CQA in a dynamic setting, when the databases is subject to updates, has not been investigated much. Integrity constraints and consistency issues for the relational model of data have been investigated for many years. However, there are other data models, e.g., spatial databases, for which much research of this kind is still necessary.

Cross-references

- ▶ Active Databases
- ▶ Data Cleaning
- ▶ Logical Data Integration
- ▶ Logics and Databases
- ▶ Null Values
- ▶ Relational Theory

Recommended Reading

1. Arenas M., Bertossi L. and Chomicki J. Consistent query answers in inconsistent databases. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 68–79.
2. Bertossi L. Consistent query answering in databases. ACM SIGMOD Rec., 35(2):68–76, 2006.
3. Bertossi L. and Chomicki J. Query answering in inconsistent databases. In Logics for Emerging Applications of Databases. Springer, Berlin, 2003, pp. 43–83.
4. Chomicki J. Consistent query answering: five easy pieces. In Proc. 11th Int. Conf. on Database Theory, 2007, pp. 1–17.
5. Codd E.F. A relational model of data for large shared data banks. Commun. ACM, 13(6):377–387, 1970.
6. Enderton H. A Mathematical Introduction to Logic, 2nd edn. Academic Press, New York, NY, USA, 2001.

Incremental Computation of Queries

Guozhu Dong[1], Jianwen Su[2]
[1]Wright State University, Dayton, OH, USA
[2]University of California-Santa Barbara, Santa Barbara, CA, USA

Synonyms

Incremental view maintenance

Definition

A view on a database is defined by a query over the database. When the database is updated, the value of the view (namely the answer to the query) will likely change. The computation of the new answer to the query using the old answer is called *incremental query computation* or *incremental view maintenance*. Incremental computation is typically performed by identifying the part in the old answer that need to be removed, and the part in the new answer that need to be added. Incremental computation is desirable when it is much more efficient than a re-computation of the query. Efficiency can be measured by computation time, storage space, or query language desirability/availability, etc. Incremental computation algorithms could use auxiliary relations (in addition to the query answer), which also need to be incrementally computed.

Two query languages can be involved for the incremental query computation problem. One is used for defining the view to be maintained, and the other for describing the incremental computation algorithm. For relational databases, the two languages can be relational algebra, SQL, nested relational algebra, Datalog, SQL embedded in a host programming language, etc.

Historical Background

Reference [14] was an early paper on incremental query computation of relational algebra queries. Reference [8] improved the query rewriting algorithm of [14] to ensure minimality of the changes to a query result in response to updates to base relations. Other articles in the literature (see [9]) considered the incremental computation of queries involving SQL aggregation, duplicates, etc.

Reference [7] was an early paper on incremental query computation of recursive (Datalog, including transitive closure) queries using relational calculus (or equivalently relational algebra) queries. Reference [3] provided relational-calculus queries for maintaining the transitive closure of acyclic graph (binary relation) after an edge insertion or deletion. Reference [6] discussed how to incrementally compute Datalog queries after tuple insertion. References [4,13] provided relation-calculus queries for maintaining the transitive closure of undirected graph after edge insertion or deletion. Reference [1] rewrote the maintenance queries in SQL. Reference [12] provided relational-calculus queries for maintaining the shortest paths in undirected graphs after edge insertion/deletion. Reference [11] studied the power of incremental query computation using SQL as maintenance queries. A number of theoretical results on the power of the arity of the auxiliary relations have been reported. Reference [5] gave a survey of results on the incremental computation of recursive queries up to the year 2000.

Reference [9] contains a collection of papers on incremental query computation/view maintenance.

Foundations

For brevity and concreteness, the discussion below will be restricted to relational databases, although incremental query computation also applies to other kind of databases. Moreover, the discussion is divided into two parts, one part for the incremental computation of relational-algebra queries, and the other part for the recursive queries (including Datalog and transitive closure). Algorithms for incrementally computing other kinds of queries are not covered in this entry.

Incremental query computation is not merely a means to avoid expensive re-computation. Sometimes, incremental computation is a way to do things that could not be done otherwise. For example, the transitive closure query cannot be directly expressed in relational algebra but it can be expressed in relational algebra in the incremental setting.

Some notations are now defined. Let $R_1,...,R_m$ be the base relations of a relational database D. An update to the database consists of $2m$ sets of tuples: for each relation R_i it contains two disjoint sets, R_i^+ and R_i^-, of tuples, where R_i^+ is the set of tuples to be added to R_i and R_i^- is the set of tuples to be deleted from R_i. Let D^{old} denote the old database and R_i^{old} denote the old instance of R_i before the update, and D^{new} denote the new database and R_i^{new} denote the new instance of R_i after the update. It is assumed that the update is minimal in the sense that $R_i^+ \cap R^{\text{old}} = \emptyset$ and that $R_i^- \subseteq R^{\text{old}}$.

The same notation will be used to denote the "update to the query." Let Q be a query. Then Q^- is used to denote the set of tuples to be deleted from the old answer, and Q^+ is used to denote the set of tuples to be added to the old answer, in order to get the new answer; in formula: $Q(D^{\text{new}}) = (Q(D^{\text{old}}) - Q^-) \cup Q^+$.

Incremental Computation of Relational-Algebra Queries

The relational algebra queries are expressions built from relation names using the following operations: selection (σ_p), projection (Π_A), Cartesian product ($\times$), union ($\cup$), intersection ($\cap$), difference ($-$), and join ($\bowtie$). Here p is a predicate or condition, and A is a set of attributes. A relational-algebra query over the database is a relational algebra expression.

The 16 rules [8] in Table 1 can be used to generate the queries for computing Q^+ and Q^-. These rules also ensure that the "update to the query" is minimal in the sense that $Q^+ \cap Q^{\text{old}} = \emptyset$ and $Q^- \subseteq Q^{\text{old}}$.

Observe that one also needs to store the answer to all intermediate queries in order to use the rules; these are auxiliary relations that also need to be maintained.

Incremental Computation of Recursive Queries

The framework used in the incremental computation of recursive queries is quite similar to the one for relational algebra queries as discussed above. However, there are two notable differences. One is that the

Incremental Computation of Queries. Table 1. Rules for incremental computation of relational algebra queries

Query Q	Rewriting Rule for Getting Q^-	Query Q	Rewriting Rule for Getting Q^+
R_i	R_i^-	R_i	R_i^+
$\sigma_p(S)$	$\sigma_p(S^-)$	$\sigma_p(S)$	$\sigma_p(S^+)$
$\Pi_A(S)$	$\Pi_A(S^-) - \Pi_A(S^{new})$	$\Pi_A(S)$	$\Pi_A(S^+) - \Pi_A(S^{old})$
$S \times T$	$(S^- \times T^{old}) \cup (S^{old} \times T^-)$	$S \times T$	$(S^+ \times T^{new}) \cup (S^{new} \times T^+)$
$S \cup T$	$(S^- - T^{new}) \cup (T^- - S^{new})$	$S \cup T$	$(S^+ - T^{old}) \cup (T^+ - S^{old})$
$S \cap T$	$(S^- \cap T^{old}) \cup (S^{old} \cap T^-)$	$S \cap T$	$(S^+ \cap T^{new}) \cup (S^{new} \cap T^+)$
$S - T$	$(S^- - T^{old}) \cup (S^{old} \cap T^+)$	$S - T$	$(S^+ - T^{new}) \cup ((S^{old} - S^-) \cap T^-)$
$S \bowtie T$	$(S^- \bowtie T^{old}) \cup (S^{old} \bowtie T^-)$	$S \bowtie T$	$(S^+ \bowtie T^{new}) \cup (S^{new} \bowtie T^+)$

query definition and the incremental computation may use different query languages, and the other is that the incremental computation may use auxiliary relations that are not explicitly involved in the original query.

Maintenance algorithms can be specified in different (maintenance) languages such as relational calculus, SQL, nested relational algebra, Datalog, or even a host programming language. The choice of a maintenance language can be influenced by the practical constraints imposed by real systems, and it can also be influenced by efficiency issues, as some languages are more efficient or more optimizing than others. For example, it is desirable to maintain the transitive closure query using relational algebra (or equivalently, first-order logic) queries, since the latter are available in all relational database systems and they are more efficient to evaluate. Another desirable language for maintaining recursive queries is SQL.

With the help of auxiliary relations, one can maintain queries that cannot be maintained otherwise. Moreover, the arity of the auxiliary relations bounds the amount of information to be kept and makes a difference on what can be maintained.

Sometimes the incremental computation algorithms can deal with certain special types of updates. Example types of updates include single-tuple insertions, single-tuple deletions, insertions/deletions of sets satisfying certain conditions, or combinations of these.

The following describes the relational algebra (or first-order logic) queries for maintaining the transitive closure of acyclic graphs [2,3]. The incremental queries can handle both tuple insertions and tuple deletions, and they do not use auxiliary relations.

Let G represent the input graph (directed) and TC the transitive closure of G. So a tuple (x,y) is in the relation G if and only if there is a directed edge from the node x to the node y in the input graph, and a tuple (x,y) is in the relation TC if and only if there is a directed path from the node x to the node y in the input graph. An edge insertion is allowed only if this insertion does not lead to cycles in the new graph.

Suppose an edge (a,b) is inserted. Then TC is maintained as follows. Essentially, the new transitive closure is obtained by adding to the old transitive closure the following: (i) all new paths constructed by adding the new edge (a,b) to the back of existing paths ending at a, (ii) all new paths constructed by adding the new edge (a,b) to the front of existing paths starting at b, (iii) all new paths constructed by inserting the new edge (a,b) between an existing path ending at a and an existing path starting at b, and (iv) the new edge itself. New paths added by rules (1), (2), and (3) correspond to paths of type $x \longrightarrow a$, $b \longrightarrow y$, and $x \longrightarrow y$, respectively. These rules cover all new paths because only one occurrence of the new edge is necessary in every new path.

Suppose an existing edge (a,b) is deleted. TC can be maintained as follows. Let $S_{ab} = \{\ (x,y) | TC^{old}(x,a) \wedge TC^{old}(b,y)\}$ be the set of all paths (x,y) in the old TC which go through (a,b). The letter S is for *suspicious* – it is doubtful whether these paths should belong to the new TC. Let $G^{new} = G^{old} - \{\ (a,b)\}$ and $T_{ab} = (TC^{old} - S_{ab}) \cup G^{new}$. Each pair in T_{ab} is definitely in the new TC. (The letter T is for *trusty.*) Surprisingly, the new TC can be completely reconstructed from T_{ab} using several joins and projections given by the following formula:

$$T_{ab} \cup (T_{ab} \circ T_{ab}) \cup (T_{ab} \circ T_{ab} \circ T_{ab})$$

where $R_1 \circ R_2$ is defined as $\{(x,y) | \exists u (R_1(x,u) \wedge R_2(u,y)\}$ and R_1 and R_2 are binary relations.

So the new transitive closure contains (i) all *trusty* paths, (ii) all paths constructed by concatenating two consecutive *trusty* paths, and (iii) all paths constructed by concatenating three consecutive *trusty* paths.

To maintain the transitive closure of undirected graphs in relational calculus after edge deletion, one also needs to maintain some auxiliary queries. The auxiliary relations include a total order on the nodes of the graph, a spanning forest of the graph, and a ternary relation indicating whether a node is on a path between two other nodes in the spanning forest. The details can be found in [4,5], which also discuss other queries.

While the maintenance queries for the transitive closure of undirected graphs and acyclic graphs do not need to use arithmetic operations, the maintenance queries for shortest paths [12] need to use both + and < on numbers.

Key Applications

Incremental query computation is useful for (i) maintaining materialized views, (ii) efficient checking and monitoring of integrity constraints, and (iii) the efficient management of triggers in active databases. Incremental query computation is also related to dynamic descriptive complexity theory [10]. Incremental computation of transitive closure has also been used in formal verification.

Future Directions

It is still open whether the transitive closure of arbitrary directed graphs can be maintained using relational calculus queries after edge deletions.

Cross-references

- ▶ Active Databases
- ▶ Database Trigger
- ▶ FOL modeling of integrity constraints (dependencies)
- ▶ Incremental View Maintenance

Recommended Reading

1. Dong G., Libkin L., Su J., and Wong L. Maintaining transitive closure of graphs in SQL. Int. J. Inf. Technol., 5(1):46–78, 1999.
2. Dong G. and Pang C. Maintaining transitive closure in first-order after node-set and edge-set deletions. Inf. Process. Lett., 62 (3):193–199, 1997.
3. Dong G. and Su J. Incremental and decremental evaluation of transitive closure by first-order queries. Inf. Comput., 120(1):101–106, July 1995.
4. Dong G. and Su. J. Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. J. Comput. Syst. Sci., 57(3):289–308, December 1998.
5. Dong G. and Su J. Incremental maintenance of recursive views using relational calculus/SQL. ACM SIGMOD Rec., 29 (1):44–51, 2000.
6. Dong G., Su J., and Topor R. Nonrecursive incremental evaluation of datalog queries. Annals Math. Artif. Intell., 14:187–223, 1995.
7. Dong G. and Topor. R. Incremental evaluation of datalog queries. In Proc. 4th Int. Conf. on Database Theory, 1992, pp. 282–296.
8. Griffin T., Libkin L., and Trickey H. An improved algorithm for the incremental recomputation of active relational expressions. IEEE Trans. Knowl. Data Eng., 9(3):508–511, 1997.
9. Gupta A. and Mumick I.S. (eds.). Materialized views: techniques, implementations, and applications. MIT, MA, USA, 1999.
10. Immerman N. Descriptive Complexity. Springer, New York, NY, USA, December 1998.
11. Libkin L. and Wong L. On the power of incremental evaluation in SQL-like languages. In Proc. 7th Int. Workshop on Database Programming Languages, 1999, pp. 17–30. See also: SQL can maintain polynomial-hierarchy queries. Technical report, Institute of Systems Science Singapore, 1997.
12. Pang C., Dong G., and Kotagiri R. Incremental maintenance of shortest distance and transitive closure in first-order logic and SQL. ACM Trans. Database Syst., 30(3):698–721, 2005.
13. Patnaik S. and Immerman N. Dyn-FO: a parallel dynamic complexity class. J. Comput. Syst. Sci., 55(2):199–209, Oct 1997.
14. Qian X. and Wiederhold G. Incremental recomputation of active relational expressions IEEE Trans. Knowl. Data Eng., 3(3):337–341, 1991.

Incremental Crawling

Kevin S. McCurley
Google Research, Mountain View, CA, USA

Synonyms

Spidering; Crawler

Definition

Part of the success of the World Wide Web arises from its lack of central control, because it allows every owner of a computer to contribute to a universally shared information space. The size and lack of central control presents a challenge for any global calculations that operate on the web as a distributed database. The scalability issue is typically handled by creating a central repository of web pages that is optimized for large-scale calculations. The process of creating this repository consists of maintaining a data structure of URLs to fetch, from which URLs are selected, the content is fetched, and the repository is updated. This process is called *crawling* or *spidering*.

Unfortunately, maintaining a consistent shadow repository is complicated by the dynamic and uncoordinated nature of the web. URLs are constantly being created or destroyed, and contents of URLs may change without notice. As a result, there will always be URLs for which the content is not present in the repository, as well as URLs whose content is different from the copy in the repository. Many new URLs can only be discovered by recrawling old URLs whose content has now changed to include links to new URLs. In order to minimize the impact of these inconsistencies, URLs should periodically be prioritized and revisited. The process of prioritizing and revisiting URLs is usually referred to as *incremental crawling*. The primary issues in incremental crawling center around defining metrics for performance, both for the quality of the repository and the resources required to build and maintain the repository.

Historical Background

In the early days of the world wide web, it quickly became apparent that documents could be treated as living objects that could be changed at will. Thus the notion of a URL was born as a "resource locator" rather than a document ID. Early attempts to build search engines largely ignored this problem, assuming that if a document was worth retrieving later, then it would remain relatively stable at its URL. Thus early attempts to crawl and index the web simply started with a set of seed URLs, and iteratively crawled pages and extracted new URLs to be crawled. After a period of time, this repository would be used as a "snapshot" of the web to build a keyword index. If a reasonable seed set is used and URLs are appropriately prioritized, the resulting snapshot was useful for constructing a search engine or performing aggregate analysis.

The snapshot approach works well for web pages that are created and remain unchanged, but web usage has evolved quite a bit since the early days. Web authors learned that fresh content would bring repeat readers, and readers have increasingly migrated toward entertainment and news, for which freshness is increasingly important. Thus in some sense the more dynamic parts of the web are those that have the highest readership and therefore the greatest social impact. As a result, incremental crawling strategy has become increasingly important.

Foundations

In order to address the issues surrounding incremental crawling, it is important to first define the goals of the shadow repository. The primary application that has been implemented thus far have been search engines that allows users to find pages that match simply expressed information needs. A prerequisite for success of a search engine is that it reference the best content that is of interest to a majority of users at any given time. For this, the repository should be as complete as possible and as fresh as possible, since pointing users at pages that don't fulfill their information need will result in a bad user experience.

Another application that can use a shadow repository is the discovery of plagarism or copyright-protected content on the web. A third application might seek to track historical evolution of the web, for historical studies. In this case the repository should not only contain the most recent content for a URL, but all versions as they evolve over time. It is easy to see that performance metrics for these applications will vary somewhat. Moreover, these applications will differ in the the degree of cooperation that can be expected from contributors to the web.

Metrics for Incremental Crawling

There is no uniformly accepted notion of how to measure the inconsistency between the repository and the actual web. Moreover, even if there was universal agreement on the proper metric for consistency, it may be difficult to measure, and any such measure would be time-dependent since the web is constantly changing. Three possible metrics that are most obvious are:

Coverage: count the number of pages that exist on the web but are not present in the respository at any given time.
Freshness: count the number of documents that are present in the repository, but whose content was changed after insertion into the repository. Alternatively, the metric may incorporate a measure of the size of the change for individual pages.
Age: the average age of documents in the repository (e.g., time since they were last updated).

For any particular application, these metrics may all be adjusted to incorporate weights of individual pages to reflect their relative importance for the application. Thus for a search engine, documents that are dominated by thousands of other documents for every possible query should probably should be weighted lower than documents that are often found by users of the search engine. The quality of an incremental crawling strategy must also be evaluated on the basis of resources that it consumes in order to maintain a given level of metric.

Incentives and Cooperation

It should be noted that a crawler consumes resources both of the party maintaining the repository as well as the creators of content on the web. If a crawler becomes too aggressive in tracking the changes to URLs, it could quickly overwhelm small sites and degrade the value of the web site to the users of the web. At the same time, it is relatively simple to create a web site that issues dynamic content in response to any request by a crawler, essentially creating an excess of URLs that could theoretically be fetched, archived, and indexed by the crawler. Web sites may also engage in other forms of mischief, such as accepting connections and either responding slowly or with noncomformant content.

Thus both content providers and repository maintainers are threatened by potentially unconstrained resource requirements. As the web has evolved, a number of standards and standard practices have emerged that mitigate these threats, and a form of economic equilibrium has emerged to allow repositories to be maintained.

The situation is complicated by the fact that the goals of content producers and crawlers are sometimes not aligned to each other. Content providers are generally motivated by the desire to shape public opinion through their information, or their desire to drive traffic for commerce. In a competitive market, multiple content providers may find themselves in competition with each other, competing for the attention of humans. As such, they can be expected to engage in a variety of practices that will improve their share of readership.

By contrast, consider the goals of a commercial search engine. Search engines make money from advertising, and in order to maintain the traffic, they need to serve the aggregated needs of users. Even if the search engine had a completely accurate copy of the web, there is still the task of deciding which results to show, and this is where the goals of individual content providers may conflict with those of the search engine (and users). Content providers generally want their content to be indexed, but they also want their content to be shown to users ahead of their competitors. They can be expected to act in their own self interest.

One of the activities that some sites engage in is to create link farms, which are designed to enhance the ranking of pages in search engines. In order for this to be effective, the pages of link farms must be crawled and incorporated into the ranking of the search engine. This is an extreme example of the more general phenomenon that content providers and search engines may disagree on what should be crawled and indexed.

Cache Consistency

Inconsistency of a web shadow repository is similar to any other cache consistency problem, and has been studied extensively in the literature. Traditional approaches to cache consistency include time-to-live (TTL), client polling, and invalidation protocols. Most of the literature on caching has neglected the issue of discovery, which is a critical feature of crawling and one reason why pages need to be revisited (to find new links). Due to the uncoordinated nature of the web, most of the effort has gone into client polling approaches, although elements of the other approaches have also been experimented with. The HTTP protocol [5, section 13] contains a variety of optional features that can help with TTL approaches if they are properly implemented. Examples include the `Cache-Control`, `Expires`, `Last-Modified`, `If-Modified-Since`, `If-Unmodified-Since`, and `Vary` header fields. The cache-consistency protocol of the HTTP protocol is designed to facilitate human interactive browsing, and is not designed for batch processing. Moreover, relatively few web sites have taken the care to implement the existing protocols correctly.

In 2005 Google published an improved mechanism for conveying TTL cache consistency messages, known as sitemaps [7]. This followed an earlier effort called Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH), and was released under a creative commons license. The purpose of the sitemaps initiative was to provide a more efficient polling mechanism, where web sites could provide information in an XML format to specify a list of valid URLs for crawling, along with optional last-modified dates, change frequencies, and priorities. Theoretically this scales up to sites with fifty million URLs, but has only weak consistency guarantees incorporated into it. In addition to conveying information about expiration and priority, the protocol also specifies a way for sites to notify individual search engines of the existence of their sitemap, either through placement in a robots.txt file or through a ping notification message [7]. Sitemaps can also be useful in eliminating intra-site duplicates, which consume resources of search engines and degrade the quality of indexing of sites.

In theory, sitemaps can greatly improve the efficiency of maintaining a shadow repository, but they have their own problems due to previously mentinoed misalignment of incentives between content providers and search engines. Sitemaps are subject to various kinds of abuse, and cannot be completely trusted by the crawler to give an accurate view of a web site.

Resource Management

The resources consumed by incremental crawling can be significant. In one study in 2002 [9], it was estimated that 40% of Internet traffic is due to web crawlers retrieving pages. Similar figures have been observed by others, though the impact is probably higher for low-traffic sites than it is for high-traffic sites. From the point of view of the crawler, it is not obvious how to optimally design a crawl strategy in order to achieve a given level of one of a metrics such as freshness.

One tempting strategy is to adjust the crawl frequency for a page in proportion to the rate of change for the page. Thus if a page changes freqently and substantially, it would receive a higher level of crawling. It is perhaps counterintuitive that this may not be optimal. In the extreme case, a page may in fact change every time it is accessed, which means that no matter how often you access it, you will never have the "up to date copy." Accessing a page too frequently would also violate the standard "politeness" policy of web crawlers.

Another potential policy would be to revisit pages with the same frequency, ignoring the change rate of individual pages. This is evidently wasteful of resources, but Cho and Garcia-Molina [1] showed both experimentally and theoretically that this outperforms the proportional approach. The optimal strategy interpolates between the two, with a visitation schedule that increases monotonically with the change rate, but penalizes pages that change too often.

Theoretical results of this type depend upon an underlying mathematical model of how the web changes, and should be carefully examined when applying it to a restricted subset of the web, and should be re-evaluated in the future should the social forces that shape the web somehow change. Such models should reflect the rate at which new information is produced, the distribution of such production among individual websites, and the rate at which information decays or is revised. Such models are essential for deciding how to balance the resources dedicated to discovery of new content vs. the confirmation that existing content has not changed.

Key Applications

To date, the primary application that has made use of crawling is search engines. The details of their incremental crawling strategies remain unpublished, as they are based on the perceived economic value of the repository. As the web has grown, search engines have become increasingly important for users, allowing them to express an information need in fairly precise terms, and quickly navigate directly to documents on the topic of interest. From the point of view of a search engine, the dynamic nature of the web presents special problems, since a search engine will typically depend upon deep analysis of documents and links between documents. Documents whose content changes on a regular basis present a challenge for indexing, since the freshness of a search engine is a critical measure of utility for users.

Search engines are not the only applications that can make use of incremental crawling. Others include systems for prefetching proxies, notifications and alerts, mirroring and archiving, and business and political intelligence.

Future Directions

Most of the current strategies for incremental crawling are heavily dependent upon polling, and are therefore fairly inefficient. An approach built around notification and invalidation would clearly be more efficient, but there appears to be little economic incentive to implement them. Existing mathematical models for the growth and change rates of the web are fairly simplistic, treating all web pages as equal and failing to recognize the inherent organizational structure of web sites and the different purposes for which pages are created. Finally, as new applications emerge for web shadow repositories, it is natural to expect new requirements to emerge.

Experimental Results

See [1].

Cross-references

▶ Caching
▶ Data Broadcasting, Caching and Replication in Mobile Computing
▶ Focused Web Crawling
▶ Replication
▶ Web Crawler Architecture

Recommended Reading

1. Cho J. and Garcia-Molina H. Effective page refresh policies for web crawlers. ACM Trans. Database Syst., 28(4):390–426, 2003.
2. Coffman E.G., Liu Z., and Weber R.R. Optimal robot scheduling for Web search engines. J. Schedul., 1:15–29, 1998.
3. Dikaiakos M.D., Stassopoulou A., and Papageorgiou L. An investigation of web crawler behavior: characterization and metrics. Comput. Commun., 28:880–897, 2005.
4. Edwards J., McCurley K.S., and Tomlin J. An adaptive model for optimizing performance of an incremental web crawler. In Proc. 10th Int. World Wide Web Conference, 2001, pp. 106–113.
5. Fielding R., Gettys J., Mogul J., Frystyk H., Mastinter L., Leach P., and Berners-Lee T. Hypertext Transfer Protocol – HTTP/1.1, RFC 2616 http://www.w3.org/Protocols/rfc2616/rfc2616.html.
6. Podlipnig S. and Böszörmenyi L. A survey of Web cache replacement strategies, ACM Comput. Surveys, 35(4): 374–398, 2003.
7. Sitemap protocol specification. http://www.sitemaps.org/protocol.php.
8. Wang J. A survey of web caching schemes for the Internet. ACM SIGCOMM Comput. Commun. Rev., 29(5): 36–46, 1999.
9. Yuan X., MacGregor M.H., and Harms J. An efficient scheme to remove crawler traffic from the Internet. In Proc. 11th Int. Conf. on Computer Communications and Networks, 2002, pp. 90–95.

Incremental Maintenance of Views with Aggregates

HIMANSHU GUPTA
Stony Brook University, Stony Brook, NY, USA

Definition

Views are SQL or relational expressions over the given data sources. In a data warehouse, view expression generally involve the aggregate operator. In order to keep *materialized* (precomputed and stored) views up to date, it is necessary to maintain the views in response to the changes at the sources. *Incremental maintenance* of a view involves propagating the changes at the source onto the view so that the view reflects the changes. Incrementally maintaining a view can be significantly cheaper than recomputing the view from scratch.

Historical Background

Incrementally maintaining a view can be significantly cheaper than recomputing the view from scratch, especially if the size of the view is large compared to the size of the changes [1,2,9]. The problem of incremental maintenance of views has been studied extensively, and several algorithms have been proposed over the years [3,4,6–12]. Most works (except for [7,12]) either handled view expressions without aggregate operators [3,4,11], or *aggregate views* (view expressions with the aggregate operator as the last operator) [8,9,10]. Quass in [12] attempts to maintain general view expressions involving aggregate operators, but the expressions obtained are inefficient and very complicated. Recently, Gupta and Mumick [7] developed the change-table technique for incremental maintenance of general view expressions involving aggregate and outerjoin operators. Change table of a particular view is applied to the view using a special refresh operator. The change-table technique can be looked upon as a generalization of the technique of summary tables developed by [9] for maintenance of aggregate views. In contrast to the most other techniques which propagate data in terms of insertions and deletions through a view expression, the change-table technique propagates data (in terms of change-tables) as well as action (in terms of parameters of the refresh operation) through the given view expression.

Foundations

The first step is to explain the framework developed in [3,11] for deriving incremental view maintenance expressions and relate it to the change-table technique. Let a database contain a set of relations $\mathcal{R} = \{R_1, R_2,...,R_n\}$. A *change transaction* t is defined to contain the expression $R_i \leftarrow (R_i \dot{-} \nabla R_i) \uplus \Delta R_i$, for each relation R_i, where ∇R_i are the deletions from R_i, and ΔR_i are the insertions into R_i. Let V be a bag-algebra expression defined on a subset of the relations in $\mathcal{R}$. The *refresh-expression New*(V, t) ([3] uses the notation pre(t, V) instead) is used to compute the new value of V. Griffin and Libkin in [3] define the expression New(V, t) to be:

$$\text{New}(V,t) = (V \dot{-} \nabla(V,t)) \biguplus \Delta(V,t).$$

So, the goal in deriving view maintenance expressions for a view V is to derive two functions $\nabla(V, t)$ and $\Delta(V, t)$ such that for any transaction t, the view V can be maintained by evaluating $(V \dot{-} \nabla(V, t)) \uplus \Delta(V, t)$. In order to derive $\nabla(V, t)$ and $\Delta(V, t)$, [3] gives *change propagation equations* that show how deletions and insertions are propagated up through each of the relational operators. The work of [3] was extended to include aggregate operators by Quass in [12].

The change-table technique can be thought of as introducing a new definition for New(V, t). The expression New(V, t) is defined for general view expressions as

New(V, t) = (V *REFRESH* □(V, t)),

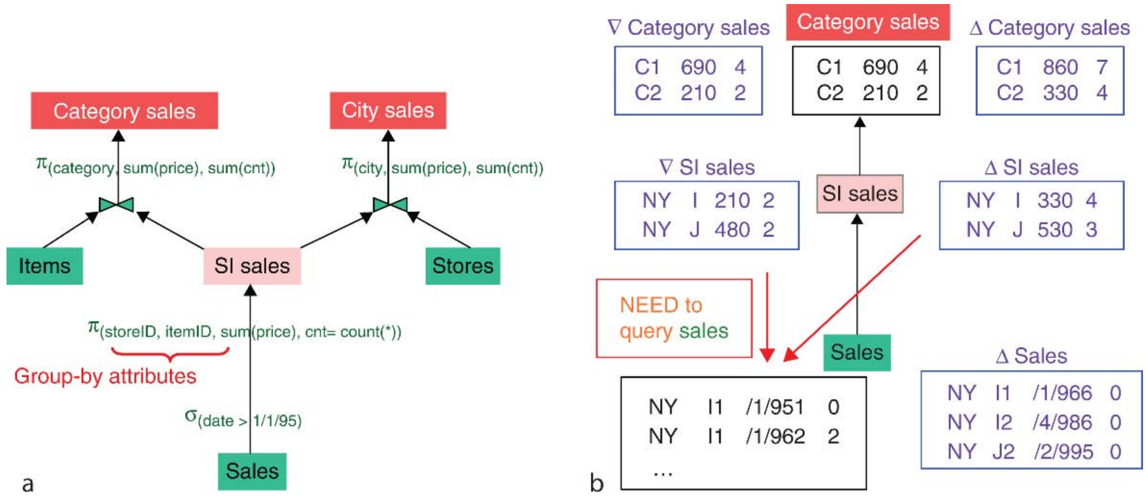
where □(V, t) is the “change-table” for the transaction $\dot{t}$ and *REFRESH* is the “refresh” operator used to apply the changes in the change table to its view. The above new definition of New(V, t) is motivated from the following observation. In the case of general view expressions involving aggregate operators, it is usually more efficient to propagate the change tables beyond an aggregate operator, instead of propagating insertions and deletions. Propagation of a change table is particularly efficient when the change table depends only on the changes to the base relation (self-maintainability [5]), while the insertions and deletions depend on the old value of the view. As shown in the example below, if the aggregate node is not materialized, the computation of insertions and deletions could be very expensive. The above new definition of New(V, t) also means that in order to obtain a complete technique it is necessary to define a general refresh operator, show how to generate a change table, and how to propagate a change table through various operators. The reader is referred to [7] for the above details.

Illustrative Example. Consider the classic example of a warehouse containing information about stores, items, and day-to-day sales. The warehouse stores the information in three base relations viz. `stores`, `items`, and `sales` having the following schemas.

`stores` (storeID, city, state)
`items` (itemID, category)
`sales` (storeID, itemID, date, price)

For each store location, the relation `stores` contains the storeID, the city, and the state in which the store is located. For each item, the relation `items` contains its itemID and its category. An item can belong to multiple categories. The relation `sales` contains detailed information about sales transactions. For each item sold, the relation `sales` contains a tuple storing the storeID of the selling store, itemID of the item sold, date of sale, and the sale price.

Views. Consider the views `SISales`, `CitySales`, and `CategorySales` defined over the base relations as shown in Fig. 1a. The view `SISales` computes for each storeID and itemID the total price of items sold after 1/1/95. The view `SISales` is an intermediate view used to define the views `CitySales` and `CategorySales`. The view `CitySales` stores, for each city, the total number and dollar value of sales of all the stores in the city. The view `CategorySales` stores the total sale for each category of items. All the above described views consider only those sales that occur after 1/1/95. The views `CitySales` and `CategorySales` are stored (materialized) at the data warehouse and this is represented below by the keyword “MATERIALIZED” (The keyword “MATERIALIZED”

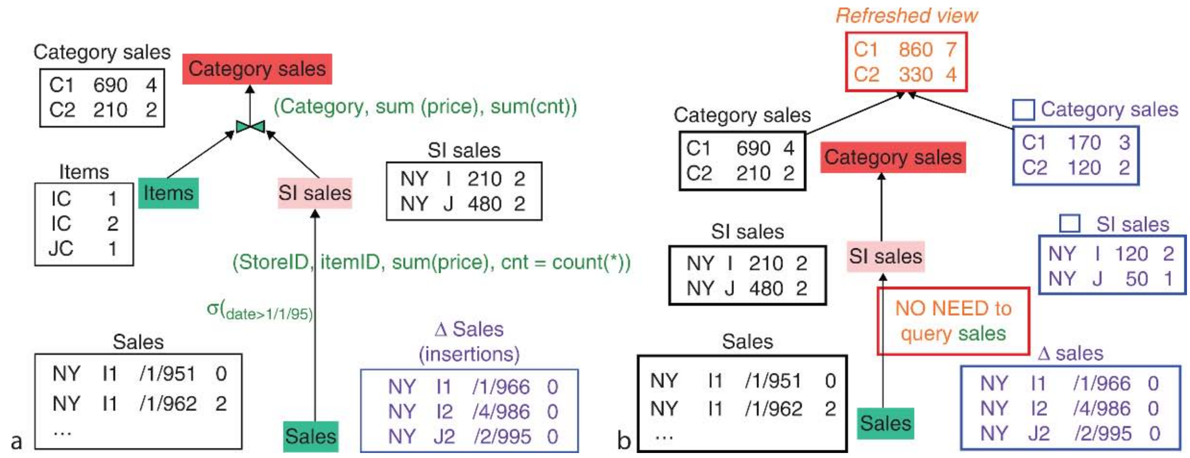


is not supported by SQL, but has been introduced in this article) in the SQL definitions of the views. It is desirable to maintain these materialized views in response to insertions to the base relation `sales` for the instance shown in Fig. 1b.

```
CREATE VIEW SISales AS
SELECT storeID, itemID, sum(price) AS
SumSISales, count(*) AS NumSISales
FROM sales
WHERE date>1/1/95
GROUP BY storeID, itemID;
CREATE MATERIALIZED VIEW CitySales AS
SELECT city, sum(SumSISales) AS SumCi-
Sales, sum(NumSISales) AS NumCiSales
FROM SISales, stores
WHERE SISales.storeID = stores.storeID
GROUP BY city;
CREATE MATERIALIZED VIEW CategorySales
AS
SELECT category, sum(SumSISales) AS
SumCaSales, sum(NumSISales) AS
NumCaSales
FROM SISales, items
WHERE SISales.itemID = items.itemID
GROUP BY category;
```

Insertion-Deletion Technique. Griffin and Libkin in [3] update view expressions by recursively computing insertions and deletions for each of the subexpressions in the view expression in response to changes at the base relations. Quass in [12] extends the techniques in [3] by including aggregate operators. Of the maintenance approaches that propagate insertions and deletions, only [12] provides techniques to maintain general view expressions involving aggregate operators. In the example, the insertions to `sales`, Δ`sales`, result in insertions (Δ`SISales`) and deletions (∇`SISales`) to the view `SISales`, which is an aggregate view over the base relation `sales`. The expressions that compute Δ`SISales` and ∇`SISales`, as derived in [12], are quite complex (see [7]). As `SISales` is not materialized, the maintenance expressions for `SISales` essentially recompute the aggregate values of the affected tuples in `SISales` from the base relation `sales`. Using the propagation equations from [12], one can propagate Δ`SISales` and ∇`SISales` upwards to obtain expressions for ∇ `CitySales`, Δ`CitySales`, ∇`CategorySales`, and Δ`CategorySales`. Figure 2a illustrates the [12] technique for updating `CategorySales` in response to insertions into `sales`. As emphasized in the figure, the computation of Δ`SISales` and ∇`SISales` require querying the base relation `sales`, because the intermediate view `SISales` is *not* materialized.

Change-Table Technique. The change-table technique proposed in [7] is as follows. Instead of computing and propagating insertions and deletions beyond an aggregate node `SISales`, a change table is computed and propagated for `SISales`. (A change table is a general form of summary-delta tables introduced in



Incremental Maintenance of Views with Aggregates. Figure 2. (a) Insertion-deletion approach ([12]), and (b) The change-table approach [8].

[9]). Propagation of change tables yields very efficient and simple maintenance expressions for general view expressions. The change table cannot be simply inserted into or deleted from the materialized view. Rather, the change table must be applied to the materialized view using a special "refresh" operator. Denote the change table of a view V by $\square V$, and a refresh operator by *REFRESH*. In practice, there are certain parameters passed with the *REFRESH* operator, but for simplicity ignore the parameters here.

For our example, start with computing the change table □SISales that summarizes the net changes to SISales. For this first level of aggregates, the expression that computes □SISales is similar to that derived in [9]. The change table □SISales is computed from the insertions and deletions into sales by using the same generalized projection (aggregation) as that used for defining SISales. More precisely,

$$\square\texttt{SISales} = \pi_{\textit{storeID,itemID,SumSISales=sum(price),NumSISales=sum(\_count)}}(\Pi_{\textit{storeID,itemID,price,\_count=1}}(\sigma_p(\triangle\text{sales})) \uplus \Pi_{\textit{storeID, itemID, price= -price,\_count= -1}}(\sigma_p(\nabla\text{sales}))), \text{ where } p \texttt{ is } (\textit{date} > \texttt{1/1/95})$$

Figure 2b presents an instance of the base relation sales and the table Δsales, which is the set of insertions into sales. For the given tables, Figure 2b also shows the computed table □SISales. Next propagate the change table □SISales upwards to derive expressions for the change tables □CitySales and □CategorySales.

$$\square\texttt{CitySales} = \pi_{\textit{City, SumCiSales=sum(SumSISales),NumCiSales=sum(NumSISales)}}(\square\text{SISales} \bowtie \text{stores})$$

$$\square\texttt{CategorySales} = \pi_{\textit{category, SumcaSales=(SumSISales), NumcaSales=sum(NumSISales)}}(\square\text{SISales} \bowtie \text{items})$$

Figure 2b shows the change table □CategorySales for the instance of the base table items in Fig. 1b. The change table □CitySales can be similarly computed. The new propagated change tables are then used to refresh their respective materialized views CitySales and CategorySales using the refresh equations below. The details of the refresh equations are in [7].

$$\texttt{CitySales} = \texttt{CitySales}\ \textit{REFRESH}\ \square\texttt{CitySales}, \text{ and}$$

$$\texttt{CitySales} = \texttt{CategorySales}\ \textit{REFRESH}\ \square\texttt{CategorySales}$$

As SISales is not materialized, it does not need to be refreshed. Also, as emphasized in Fig. 2b, it is not necessary to query the base relation sales for updating CitySales or CategorySales, which results in huge savings. The refresh operation is illustrated by showing how the CategorySales view is refreshed. Figure 2b shows the materialized table CategorySales for the given instance of base tables. For each tuple □v in □CategorySales, one looks for a matching tuple in CategorySales using the join condition CategorySales.category = □CategorySales.category (specified in one of the parameters of *REFRESH*). For example, the tuple <C1,170,3>in □CategorySales matches with the tuple v = <C1,690,4>of CategorySales. The tuple <C1,170,3>in □CategorySales means that three more sales totaling $170 have occurred for C1 category. The total number of sales for C1 is now 7 for a total amount of $860. To reflect the change, the tuple v is updated to <C1,860,7> by adding together the corresponding aggregated attributes (specified in another parameter of *REFRESH*).

Cost Comparison. It is shown in [7] that for typical sizes of base tables and changes to the base tables, the number of tuple accesses (reads and writes) incurred by the change-table technique is an order or magnitude less than that by the insertion-deletion propagation technique.

Cross-references

▶ Data Warehouse
▶ View Maintenance
▶ Views

Recommended Reading

1. Blakeley J.A. and Martin N.L. Join index, materialized view, and hybrid hash join: A performance analysis. In Proc. 6th Int. Conf. on Data Engineering, 1990, pp. 256–263.
2. Colby L., Kawaguchi A., Lieuwen D., Mumick I., and Ross K. Supporting multiple view maintenance policies. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 405–416.
3. Griffin T. and Libkin L. Incremental maintenance of views with duplicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 316–327.
4. Griffin T., Libkin L., and Trickey H. A correction to "incremental recomputation of active relational expressions" by Qian and

Wiederhold. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1994.
5. Gupta A., Jagadish H., and Mumick I.S. Data integration using self-maintainable views. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996, pp. 140–144.
6. Gupta A., Jagadish H.V., and Mumick I.S. Maintenance and self-maintenance of outerjoin views. In Proc. 3rd Workshops on Next Generational Inf. Tech. and Syst., 1997.
7. Gupta H. and Mumick I.S. Incremental maintenance of aggregate and outerjoin expressions. Inform. Syst., 31(6), 2006.
8. Gupta A., Mumick I., and Subrahmanian V. Maintaining views incrementally. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 157–166.
9. Mumick I., Quass D., and Mumick B. Maintenance of data cubes and summary tables in a warehouse. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 100–111.
10. Palpanas T., Sidle R., Cochrane R., and Pirahesh H. Incremental maintenance for non-distributive aggregate functions. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 802–813.
11. Qian X. and Wiederhold G. Incremental recomputation of active relational expressions. IEEE Trans. Knowl. Data Eng., 337–341, 1991.
12. Quass D. Materialized Views in Data Warehouses. PhD thesis, Stanford University, Department of Computer Science, 1997. Chapter 4. Preliminary version appears as Maintenance Expressions for Views with Aggregation in the ACM Workshop on Materialized Views, 1996.

Incremental k-Distance Join

▶ Closest-Pair Query

Incremental Maintenance of Recursive Views

▶ Maintenance of Recursive Views

Incremental View Maintenance

▶ Incremental Computation of Queries

Indefinite Information

▶ Incomplete Information

Index Creation and File Structures

Steven M. Beitzel[1], Eric C. Jensen[2], Ophir Frieder[3]
[1]Telcordia Technologies Piscataway, NJ, USA
[2]Twitter Inc., San Fransisco, CA, USA
[3]Georgetown University, Washington, DC, USA

Synonyms

Indexing; Inverted indexes

Definition

A core element of modern information retrieval systems is the document index. The index is a set of data structures that are constructed from a source document collection with the goal of allowing an information retrieval system to provide timely, efficient response to search queries. The process of index creation typically involves reading and processing the source document collection, parsing the text in each individual document and extracting the necessary features to allow for retrieving and ranking that document in response to a user query. Additionally, indexing systems often use dimension reduction, compression, and other related techniques to drastically reduce the storage footprint of the source collection in its indexed form. Document indexes are frequently stored in a set of file structures that are conducive to rapid retrieval and ranking by an information retrieval system in response to a query.

Historical Background

As document collections have grown in size, the need to search them in an efficient and effective fashion has grown accordingly. At the time of this writing, modern, moderately-sized document collections measure in the hundreds of gigabytes, with search engines indexing tens of billions of pages [2]. Given the sheer scale of the search problem, researchers have invested a considerable amount of effort in recent years on developing efficient file structures for indexing and storing large document collections on disk. With the advent of the web (and the subsequent rapid explosion of searchable digital documents) during a time in which disk space and processing power were substantially more expensive than they are now, indexing approaches typically focused on advanced techniques for data compression to reduce the storage footprint for an index (thoroughly discussed in [3]), and on aggressive dimension reduction, such as the removal of very frequent terms ("stopwords"), or very short documents, with the goal of reducing the total number of objects of interest tracked by the index created over the collection. As processing speeds have increased and persistent storage has gotten cheaper, indexing systems have focused on more complex algorithms intended to capture a richer set of information about the documents in the collection.

In addition to extracting key features (terms, frequencies) from the source documents and constructing data structures for storage and retrieval, indexing systems often tailor their file structures to accommodate the specific needs of the information retrieval system. For example, this may involve special extensions to allow for high-performance updates to the index, distributed indexing techniques for very large collections or high-performance indexing, or the collection of special metadata that are specific to the search tasks at hand (i.e., localized search, news search, expert search, etc). Many recent research efforts have focused on these extensions to core indexing strategies. An excellent survey can be found in [4].

Foundations

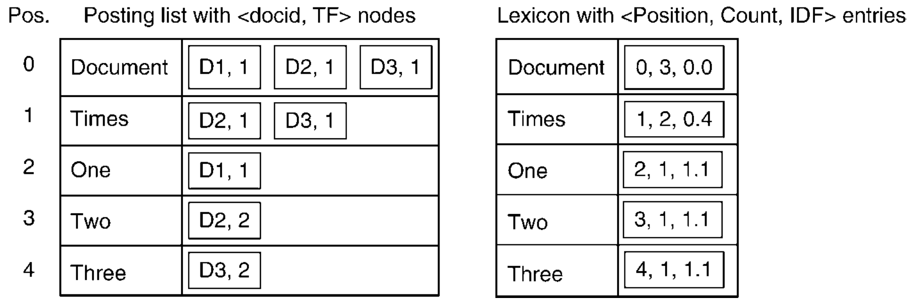
At a high level, indexing a collection of documents involves three main steps. First, the document collection must be parsed. This involves reading in each document in the collection and converting the source text of each document into a single unified form that is readable by other components in the indexing process. This step typically resolves the formatting differences between different types of source documents (i.e., HTML web pages, Microsoft Word™ documents, PDF™ files, E-Mail messages, raw text, etc). For efficiency purposes, it is often possible and desirable to perform the document parsing step in a massively parallel fashion since most documents can be parsed in a completely independent fashion.

The second major step in the indexing process involves extracting key features from the source text of each document that will be used in the retrieval process. In practice, the precise set of features is dependant on the retrieval strategy of the overall system, but distinct terms and their associated frequencies in each document are almost always included.

The final step involves sorting any intermediate representations and compiling them into a unified set of final data structures for storing on the disk. A common type of index structure that is frequently referenced in the literature is the *inverted index* [1], which represents a reasonably static document collection as a posting list, where each element in the list maps a distinct term in the collection to a set of documents containing that term. Furthermore, this list can be sorted in various ways depending on the optimization strategies imposed by the underlying search engine (sorting by descending term frequency is a common approach, but there are many others). Each node in the posting list entry (which represents a document containing the term at the head of the posting list entry) may contain extra information about that term as it pertains to the document in question. A common example of this is the term's position within the document, which is valuable for proximity search, as well as presenting results with *keyword-in-context,* meaning that result summaries will display query terms highlighted with surrounding text to give the user an idea of how their keywords are used in the result document.

In many systems, the posting list is supported by a secondary data structure known as a lexicon, which contains an entry for each distinct term in the collection along with a set of metadata that help the retrieval algorithms score target documents for relevance, including statistics like the inverse document frequency (IDF) of the term and the size of that term's posting list. Additionally, for efficiency purposes, the lexicon may include offset addresses into the inverted file, which is often stored as a random access file on the disk. Other efficiency gains may be observed if the lexicon is sufficiently small to fit in main memory, which eliminates the need for extra disk-bound I/O.

For an example of inverted index construction, consider a small collection with three documents. Document one contains the text "This is document one." Document two reads "This is document two, times two" and document three reads "This is document three, times three." The indexing system must read and parse these three documents, extract their terms, and build the posting list and lexicon. As mentioned above, a common technique used to reduce the size of the index is to remove very frequent terms that add little overall information to a document's content (typically called "stopwords"). For this example, assume that the terms "this" and "is" are considered stopwords and removed accordingly. Thus, at the end of the parsing process, five distinct terms in this example collection remain: "document", "times", "one", "two", and "three". Each distinct



Pos.	Posting list with <docid, TF> nodes			
0	Document	D1, 1	D2, 1	D3, 1
1	Times	D2, 1	D3, 1	
2	One	D1, 1		
3	Two	D2, 2		
4	Three	D3, 2		

Lexicon with <Position, Count, IDF> entries	
Document	0, 3, 0.0
Times	1, 2, 0.4
One	2, 1, 1.1
Two	3, 1, 1.1
Three	4, 1, 1.1

Index Creation and File Structures. Figure 1. Example inverted index with Lexicon.

term is represented as an entry in the posting list with the associated lexicon, as shown in Fig. 1.

As discussed above, each element in the posting list represents a distinct term in the collection, and each term references a list of documents containing that term along with the term frequency (TF) of that term in the document. The Lexicon contains a listing of all distinct terms along with their position in the posting list, the total number of posting list nodes for the term, and the inverse document frequency (IDF) of that term in the collection, which is defined as the log of the ratio of total documents to the number of documents containing the term [1]. It is easy to see how the retrieval process could operate using this data structure, simply looking up each query term in the lexicon, using the position information there to retrieve the posting list entry for that term, and using the term frequency, IDF, and any other available statistics to rank each document for inclusion in the final results presented to the user.

Key Applications

Indexing processes and index file structures are required for virtually all modern Information Retrieval systems, given the size and complexity of modern document collections. In addition to indexing large, relatively static collections of documents, there is also research that addresses the construction and maintenance of dynamic and distributed indexes, which represent a significant area of focus as the amount of searchable data continues to grow and the need to search becomes more and more pervasive and decentralized.

Cross-references

- ▶ Lexical Analysis of Textual Data
- ▶ Stoplists
- ▶ Text Index Compression

Recommended Reading

1. Grossman D. and Frieder O. Information retrieval: algorithms and heuristics. 2nd Edn. Springer, 2004.
2. The size of the World Wide Web: www.worldwidewebsize.com – Retrieved March, 2008.
3. Witten I.H., Moffat A., and Bell T.C. Managing gigabytes: compressing and indexing documents and images. 2nd Edn. Morgan Kaufmann, 1999.
4. Zobel J. and Moffat A. Inverted files for text search engines. ACM Comput. Surv., 38:(2), 2007.

Index Join

Jingren Zhou
Microsoft Research, Redmond, WA, USA

Synonyms

Index join; Index loop join; Index nested loop join

Definition

The index join is a variant of the nested loop join in database systems using index. The join predicate can be either an equality predicate or a range predicate. The algorithm starts with reading the *outer* relation R. For each tuple $\mathcal{R} \in R$, instead of scanning the entire *inner* relation S, an index on S is used to find matching tuples and add them to the result.

Key Points

An index on S is applicable for an index join if one join attribute is the leading indexed key of the index. If the join predicate is an equality predicate, an index lookup is performed for each outer tuple. If the join predicate is a range predicate, for each outer tuple, an index seek is performed to locate the first matching tuple, followed by an index scan for the rest matching tuples. Compared with the nested loop join, the index join saves disk I/Os for reading the entire inner relation *S* multiple times.

Cross-references

- ▶ Evaluation of Relational Operators
- ▶ Parallel Join Algorithms

Recommended Reading

1. Mishra P. and Eich M.H. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, 1992.

Index Loop Join

▶ Index Join

Index Nested Loop Join

▶ Index Join

Index Sequential Access Method (ISAM)

▶ Tree-based Indexing

Index Structures for Biological Sequences

Tamer Kahveci
University of Florida, Gainesville, FL, USA

Definition

Biological sequence databases are mainly composed of DNA, RNA, and protein sequences. DNA and RNA sequences are polymers of nucleotides, whereas proteins are polymers of amino acids. A database of biological sequences contains a set of biological sequences of the same type. The length of each sequence varies from less than a hundred to several hundred million bases. An index structure on a database of biological sequences helps in identifying sequences in that database that are similar to a given query sequence quickly. The definition of similarity depends on two orthogonal parameters; similarity function and the length of the similarity of interest.

The simplest similarity function is the edit distance, which measures the number of substitutions, insertions, and deletions needed to transform one sequence to the other. More complex functions involve variable gap penalties and substitution scores based on how frequent substitutions are observed in nature. The length of the similarity can be either the entire sequence (global alignment) or a subsequence of the database and the query sequence (local alignment). Depending on how the similarity is defined using these parameters, the similarity may or may not be metric. The index structure used for the biological sequence database needs to be suitable to accommodate the underlying similarity measure.

Historical Background

Levene identified the nucleotide bases in 1919. He found that the nucleotides can form a chain as the phosphate in a nucleotide can create bond with other nucleotide. Watson and Crick discovered the structure of the DNA in 1953 using X-ray diffraction. The efforts to create a database of DNA sequences, named GenBank, has started in 1979. By 1983, there were around 2,000 sequences in GenBank. Since then, the size of genome databases is increasing exponentially [2]. Statistics show that the size of GenBank has doubled every 15 months. In August 2005, GenBank contained over 100 billion bases. This rapid growth in the size of the biological sequence databases coupled with the costly distance measures made the use of index structures essential for this data type.

One of the earliest index structures used for biological sequences is the hash table [10]. Hash tables are often used to index *k*-grams (subsequences of length *k*) of the sequences in the database. This index structure is used to find exact matches between the *k*-grams of a given query sequence and that of the database sequences. Popular biological database search tools such as FASTA [10] and BLAST [1] employed hash tables to index *k*-grams. These tools created one hash table entry for each unique *k*-gram. Thus, the number of hash table entries is exponential in *k*. This limits the value of *k* to a small number. Another drawback of the hash table is that the value of *k* has to be predetermined at the time of index construction.

Therefore, the same sized k-grams need to be used for all the queries. Pol and Kahveci incorporated randomization to enable variable k value for hash table construction [11].

Suffix trees and suffix arrays constitute another set of commonly used index structures for biological sequences. Suffix trees were first proposed by Weiner [14] under the name *position tree*. Unlike the hash table, this structure index all the suffixes of all the sequences in the database. McCreight proposed a space efficient technique for the construction of the suffix trees [9]. Later, Ukkonen developed an on-line construction method [12]. A variety of other suffix tree implementations have also been proposed such as implicit suffix tree [12], string B-tree [4] and suffix array [8]. AVID [3] use the suffix tree structure to query biological sequences.

Hash tables and suffix trees can identify exact matches. The tools that use them perform additional processing to identify approximate matches. Reference-based index structures enable approximate searches. The VP-tree (Vantage Point tree) [15] indexes the data in general metric space with the help of references. Omni method [5] proposed to select the references near the convex hull of the database, far away from each other. Venkateswaran et al, later showed that multiple references should be used to index a database sequence, and the references should be a combination of similar and distant sequences [13].

The Sequence Search Tree (SST) [6] maps each sequence to a set of vectors in a high dimensional integer space. It then builds an index on these vectors. The Multi Resolution String (MRS) index structure [7] also maps the subsequences into a high dimensional integer space. Unlike SST, the mapping of the MRS index structure allows computation of a lower bound to the distance. This way MRS avoids false dismissals.

Foundations

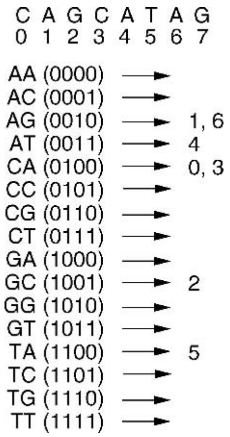
Hash Tables

One of the most widely used index for sequence search is the *hash table* (also called the *lookup table*). Hash table enables quick lookup for a set of pre-specified sequences. Usually, in the bioinformatics literature, the hash table is built on the k-grams, where k is predefined.

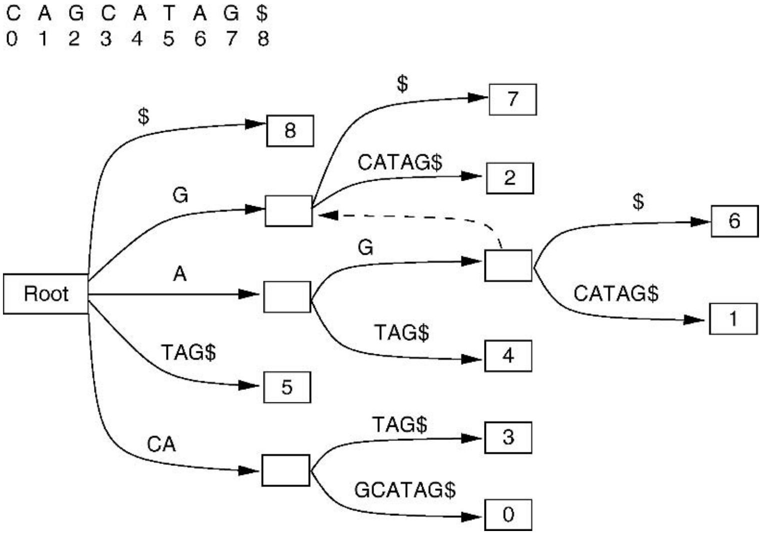
The hash table of a sequence is built using two parameters: alphabet and k. Assume that the alphabet contains σ letters. The hash table encodes the ith letter in the alphabet as the binary representation of $i - 1$ using $\lceil log_2\sigma \rceil$ bits. For example, the letters in the DNA alphabet {A, C, G, T} are encoded as A = 00, C = 01, G = 10, and T = 11. The hash key of a k-gram is the concatenation of the binary representations of the letters that constitute that sequence. For example, the DNA sequence GGCA is represented as 10100100. Figure 1 shows the hash table built on a DNA sequence for $k = 2$. In this example, the hash table contains 4^2 entries since $\sigma = 4$. The total number of pointers to database is the $N - k + 1$, where N is the number of letters of the sequence indexed.

Suffix Trees

Suffix tree is a tree structure where each path from the root to a leaf node denotes a suffix of the sequence that it indexes. Thus, there is a bijection between the leaf nodes and the suffixes. In order to be able to create this bijection unambiguously, suffix tree increases the length of each of the database sequences by one letter by appending a unique "stop" character at the end of the sequence. This character is not contained in the alphabet that constitutes the database sequences. Figure 2 shows the suffix tree built on sequence CAGCATAG. The letter, $, marks the end of the sequence. Every path from the root node to a leaf node through the solid arrows defines a suffix of this sequence. The labels on the edges of each such path show the



Index Structures for Biological Sequences. Figure 1. A DNA sequence and the hash table constructed on it when $k = 2$. The numbers in parenthesis show the binary code of the corresponding sequence.



Index Structures for Biological Sequences. Figure 2. Suffix tree built on the sequence CAGCATAG. The dashed arrow is a suffix link. The letter, $, marks the end of the sequence.

C	A	G	C	A	T	A	G
0	1	2	3	4	5	6	7

6	AG
1	AGCATAG
4	ATAG
0	CAGCATAG
3	CATAG
7	G
2	GCATAG
5	TAG

Index Structures for Biological Sequences. Figure 3. Suffix array built on the sequence CAGCATAG and the corresponding suffixes indexed by the entries of this index.

contents on that suffix. The numbers at the leaf nodes show the starting position of the suffix denoted by that leaf. The dashed arrows in the figure are the suffix links. There is a suffix link from an internal node *u* to another internal node *v* if *u* and *v* are labeled with suffixes $c\alpha$ and α respectively for a given letter *c*. Suffix links enable faster construction of suffix trees.

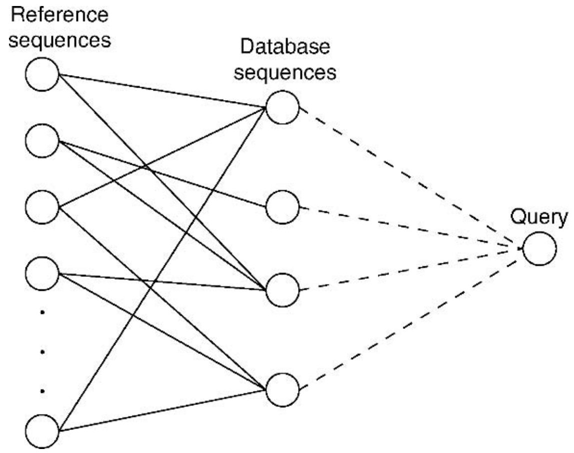
Suffix trees are notorious for their excessive memory usage. Although the space complexity is $O(n)$, the constant in the big-Oh may be large. The size of the suffix tree depends on the alphabet and the distribution of the letters in the database sequence. The literature reports the size of the suffix tree as 10–70 bytes per letter in the database.

Suffix Arrays

Suffix arrays reduces the space consumption of suffix trees to five bytes per letter at the expense of increased search time complexity. The suffix array of a sequence is an array of integers that shows the alphabetical order of all suffixes of that sequence. Figure 3 shows the suffixes of CAGCATAG and the suffix array constructed on it.

Reference-Based Indexing

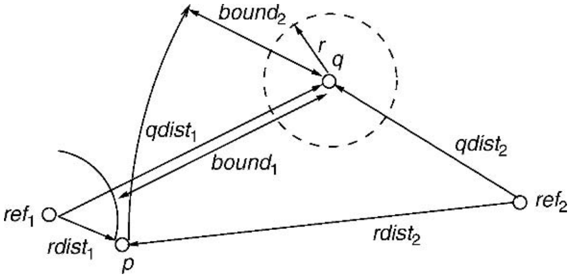
Reference-based index structures work when the distance measure is metric. The distance measure is metric when the distance function is metric and the entire sequences are aligned. An example to metric distance functions is the edit distance. Reference-based index structures select a small number of sequences, referred to as the set of *reference sequences*. Often, these references are selected from the database sequences for simplicity. However, it is possible select sequences that are not in the database as references. The structure pre-computes the distances between the references and the database sequences. For a given a query sequence, the query algorithm first computes the distance from each of the references to the query sequence. It then computes upper and lower bounds to the distance between the query sequence and the database sequences with the help of the pre-computed distances



Index Structures for Biological Sequences. Figure 4. Reference based indexing. The *small circles* represent database, reference and query sequences. Each database sequence is indexed by two references. The *solid lines* denote which references are used for each database sequence. The distances between the sequences connected by *solid line* are precomputed. The *dashed lines* denote the distance between the query and the references. These distances are computed on the fly as the query arrives.

in the index quickly. Thus, without any further comparisons, sequences that are too close to or too far away from a reference may be removed from the candidate set with the help of the triangle inequality. Figure 4 shows a reference based index where each database sequence is indexed using two references.

Figure 5 illustrates reference-based indexing in a hypothetical two-dimensional, space. Here, the database sequences are represented by points. The distance between two points in this space corresponds to the underlying distance between the two sequences (e.g., if the points denote sequences, $rdist_1$ between the points ref_1 and p corresponds to the edit distance between the sequences represented by them). In reference-based indexing, the distances between the sequence p and references ref_1 and ref_2 are pre-computed. Let $rdist_1$ and $rdist_2$ be the two pre-computed distances, respectively. Given a query q with range r, the first step is to compute reference-to-query distances $qdist_1$ and $qdist_2$. A lower bound for the distance between sequences q and p with reference ref_1 is computed as $bound_1 = |qdist_1 - rdist_1|$ using the triangle inequality. Similarly, $bound_2$ gives a lower bound for the distance between q and p with reference ref_2. Since $bound_1 > r$ with ref_1 as the reference, sequence p can be pruned from the candidate set of q.



Index Structures for Biological Sequences. Figure 5. Two dimensional illustration of reference-based indexing. Here ref_1 and ref_2 are references. The query region is given by its center q and radius r. $qdist_1$ and $qdist_2$ are query-to-reference distances. $rdist_1$ and $rdist_2$ are distances from the reference to the data p. $bound_1$ and $bound_2$ are the bounds obtained using references.

Vector Space Indexing

This set of index structures first map the sequences to a vector space (typically in a multi-dimensional integer space) and build index structure on this space. An example to this is the MRS (Multiple Resolution String) index.

Let s be a sequence from the alphabet $\Sigma = \{\alpha_1, \alpha_2,...,\alpha_\sigma\}$. Let n_i be the number of occurrences of the character α_i in s for $1 \leq i \leq \sigma$. The vector space mapping of s is computed as $f(s) = [n_1, n_2,...,n_\sigma]$. The vector, $f(s)$ is called the *frequency vector*, of s. For example, let s = AGCTTTTCATTCTGAC be a DNA sequence. The frequency vector of s is $f(s) = [3,4,2,7]$ ([#As,#Cs,#Gs,#Ts]), since the DNA alphabet contains the letters A, C, G, and T.

The MRS index structure stores a sequence of Minimum Bounding Rectangles (MBRs) at different resolution levels in the index structure. Resolutions are represented using window size, which are powers of two.

In order to obtain the MBRs of a sequence at resolution 2^i, a window of length $w = 2^i$ is placed at the leftmost point of that sequence. Later, this window is slid by one letter until it reaches to the end of that sequence. Each placement of this window produces a subsequence. The frequency vectors of all those windows are computed. First, the minimum box, called

Minimum Bounding Rectangle (MBR), that covers the frequency vector of the first subsequence is computed. This box is later extended to cover more frequency vectors until the box capacity is reached. Box capacity is an integer that denotes the maximum number of frequency vectors that an MBR can contain. Typically, this number is set to 1,000 to achieve good performance result. Once the box capacity is reached, a new MBR is created to cover the next subsequences. This process continues until all subsequences are transformed. Note that only the lower and higher end points of the MBRs along with the starting locations of the first subsequence contained in that MBR are stored for each MBR.

Key Applications

The ability to query large biological sequence databases is needed in nearly all areas of molecular biology, pharmacology, plant sciences and horticulture. Two example applications are as follows.

Repeat identification. DNA sequences contain large amounts of repeating patterns. Identifying these patterns is essential for many purposes. For example, repeat copy numbers can help in determining ancestral relationship, which is often needed to identify victims, criminals, parenthood, whether a race horse is pure breed, etc. Furthermore, identifying these repeats is needed for more accurate sequence assembly and for high quality primer production. Identifying repeats require comparing the sequences in a repeat library and the target sequence (library-based repeat identification) as well as a self comparison of the target sequence (de-novo repeat identification). The size of the target sequence and the repeat library often necessitates the use of an index structure.

Shotgun sequencing. One of the commonly used technologies to identify the letters in large chromosomes is called *shotgun sequencing*. This technology first produces multiple copies of a given long DNA sequence. It then chops these sequences into short fragments from (almost) random locations using restriction enzymes. It then identifies the sequences that have less than 1,000 letters using high throughput sequencing machines. This process produces a bag of short subsequences of the target DNA sequences. The challenge is then to reassemble the original long DNA sequence from these short fragments. This problem requires an all-to-all sequence comparison for repeat and overlap detection. The massive size of the database makes it essential to use an index structure.

Data Sets

GenBank http://www.ncbi.nlm.nih.gov/Genbank/
PDB http://www.rcsb.org/pdb/
SwissProt http://ca.expasy.org/sprot/

Cross-references

▸ Biological Sequence
▸ Query Languages and Evaluation Techniques for Biological Sequence Data

Recommended Reading

1. Altschul S., Gish W., Miller W., Meyers E.W., and Lipman D.J., Basic Local Alignment Search Tool. J. Mole. Biol., 215 (3):403–410, 1990.
2. Benson D., Karsch-Mizrachi I., Lipman D., Ostell J., Rapp B., and Wheeler D. GenBank. Nucleic Acids Res., 28(1):15–18, 2000.
3. Bray N., Dubchak I., and Pachter L. AVID: a global alignment program. Genome Res., 13(1):97–102, 2003.
4. Ferragina P. and Grossi R. The string B-tree: a new data structure for string search in external memory and its applications. J. ACM, 46(2):236–280, 1999.
5. Filho R.F.S., Traina A.J.M., Caetano Traina J., and Faloutsos C. Similarity search without tears: The OMNI family of all-purpose access methods. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 623–630.
6. Giladi E., Walker M., Wang J., and Volkmuth W. SST: an algorithm for finding near-exact sequence matches in time proportional to the logarithm of the database size. Bioinformatics, 18 (6):873–877, 2002.
7. Kahveci T. and Singh A. An efficient index structure for string databases. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 351–360.
8. Manber U. and Myers E. Suffix arrays: a new method for on-line string searches. SIAM J. Comput., 22(5):935–948, 1993.
9. McCreight E. A space-economical suffix tree construction algorithm. J. ACM, 23(2):262–272, 1976.
10. Pearson W. and Lipman D. Improved tools for biological sequence comparison. In Proc. Natl. Acad. Sci., 85:2444–2448, 1988.
11. Pol A. and Kahveci T. Highly scalable and accurate seeds for subsequence alignment. In Proc. IEEE Int. Conf. on Bioinformatics and Bioengineering, 2005.
12. Ukkonen E. On-line Construction of Suffix-trees. Algorithmica, 14:249–260, 1995.
13. Venkateswaran J., Lachwani D., Kahveci T., and Jermaine C. Reference-based indexing for metric spaces with costly distance measures. VLDB J. 17(5):1231–1251, 2008.
14. Weiner P. Linear pattern matching algorithms. In Proc. IEEE Symposium on Switching and Automata Theory, 1973, pp. 1–11.
15. Yianilos P. Data structures and algorithms for nearest neighbor search in general metric spaces. In Proc. 4th Annual ACM - SIAM Symp. on Discrete Algorithms, 1993, pp. 311–321.

Index Tuning

Philippe Bonnet[1], Dennis Shasha[2]
[1]University of Copenhagen, Copenhagen, Denmark
[2]New York University, New York, NY, USA

Definition

Index Tuning is concerned with when and how to construct an index.

Historical Background

When relational models were first introduced, customers complained that vendors had introduced them to slowed down customer applications in order to make the customers buy more hardware. Vendors and researchers responded to the performance challenge by incorporating existing data structures (notably B-trees and hash structures) and improving many new ones (notably bit vectors and multi-dimensional indexes). The decision of which data structures to create was left to the user. This entry concerns that decision.

Foundations

An *index* is a data structure plus a method of arranging the data tuples in the table (or other kind of collection object) being indexed. The syntax for constructing indexes is discussed elsewhere. This entry presents the tuning considerations.

Two data structures are most often used in practice for indexes: B+-trees and Hash structures. Of these, B+-trees are used the most often. The folk wisdom holds that if one doesn't know which kind of indexes to put on a column or set of columns and scanning is too slow, then one should use a B+-tree. When performing accesses based on equality, however, hash structures perform better.

The main tuning consideration for *B+-trees* is to avoid having too many levels. Because an access to disk secondary memory costs a few milliseconds if it requires a seek (as index accesses will) and even an access to flash memory may require switching blocks (a relatively expensive operation), the performance of a B+-tree depends critically on the number of nodes in the average path from root to leaf – the number of levels (the root will tend to be in RAM, but the other levels may or not be, and the farther down the tree the search goes, the less likely it is for the nodes to be in RAM). One technique that database management systems use to minimize the number of levels is to make each interior node have as many children as possible (1,000 or more for many B+-tree implementations). The maximum number of children a node can have is called its *fan-out*. Because a B+-tree node consists of key-pointer pairs, the larger the key is, the lower the fan-out.

For example, a B+-tree with a million records and a fan-out of 1,000 requires three levels (including the level where the records are kept). A B+-tree with a million records and a fan-out of 10 requires seven levels. If one increases the number of records to a billion, the numbers of levels increase to four and ten respectively. This is why accessing data through indexes on large keys is slower than accessing data through small keys on most systems (the exceptions are those few systems that offer good compression).

Hash structures, by contrast, store key–value pairs based on a pseudorandomizing function called a *hash function*. The hash function can be thought of as the root of the structure. Given a key of any size, the hash function returns a location that contains either a page address (usually on disk) or a directory location that holds a set of page addresses. That page either contains the key and associated record or is the first page of a linked list of pages, known as an *overflow chain* leading to the record(s) containing the key. (One can keep overflow chaining to a minimum by allocating enough hash buckets such that the space in the hash buckets is at least twice the space required to hold the data.) In the absence of overflow chains, hash structures can answer equality queries (e.g., find the employee with a certain Social Security number) in one disk access, making them the best data structures for that purpose.

The data structure portion of an index has pointers at its leaves to either data pages or data records. If there is at most one pointer from the data structure to each data page, then the index is said to be *sparse*. If there is one pointer to each record in the table, then the index is said to be *dense*.

If records are small compared to pages, then there will be many records per data page and the data structure supporting a sparse index will usually have one fewer level than the data structure supporting a dense index. This means one less disk (or flash block) access if the table is large. By contrast, if records are almost as large as pages, then a sparse index will rarely have better disk access properties than a dense index.

The main virtue of dense indexes is that they can support certain read queries within the data structure itself in which case they are said to *cover* the query. For example, if there is a dense index on the keywords of a document retrieval system, a query can determine the number of records containing some term, e.g., "derivatives scandals," without accessing the records themselves (Count information is useful for that application, because queriers frequently reformulate a query when they discover that it would retrieve too many documents). A secondary virtue is that a query that makes use of several dense indexes can identify all relevant tuples before accessing the data records. Instead, one can form intersections and unions of pointers to data records or of record identifiers.

A *clustering index* on an attribute (or set of attributes) *X* is an index that puts records close to one another if their *X*-values are *near* one another. What "near" means depends on the data structure. On B-trees, two *X*-values are near if they are close in their sort order. For example, they are close numerically or alphabetically. In hash structures, two *X*-values are near only if they are identical. Index-organized tables are clustering indexes where *X* is the primary key.

Sparse indexes must be clustering, but clustering indexes need not be sparse. In fact, clustering indexes are sparse in some systems (e.g., SQL Server, ORACLE hash structures) and dense in others (e.g., ORACLE B-trees, DB2). Because a clustering index implies a certain table organization and the table can be organized in only one way at a time, if there is a clustering index on the sequence of attributes *Y*, then any other clustering index on attributes *X* must have the property that *X* is a prefix of *Y* or *Y* is a prefix of *X*.

A *nonclustering index* (sometimes called a *secondary* index) is an index on an attribute (or sequence of attributes) *Y* that puts no constraint on the table organization. The table can be clustered according to some other attribute *X* or can be organized as a heap, as discussed below. A nonclustering index must be dense, so there is one leaf pointer per record. There can be many nonclustering indexes per table.

A *heap* is the simplest table organization of all. In the basic implementations, records are ordered according to their time of entry. That is, new insertions are added to the last page of the data structure. In this case, inserting a record requires a single page access.

Nonclustering indexes are very useful if they cover a query but can also be useful if the query retrieves significantly fewer records than there are pages in the file. The word "significant" needs explanation: a table scan can often save time by reading many pages at a time, provided the table is stored on contiguous tracks. Therefore, if a query requires most records in a table, a scan may be 2–10 times faster than an index read.

Decision support applications often entail querying on several, perhaps individually unselective, attributes. For example, "Find people in a certain income range who are male, live in California, buy boating equipment, fish, drive a sports car, and work in the computer industry." Each of these constraints is unselective in itself, but together form a relatively small result. The best all-around data structure for such a situation is the *bitmap*. A bitmap is a collection of vectors of bits. The length of each vector equals the length of the table being indexed and has a 1 in position i if the ith record of the table has some property. For example a bitmap on state in the United States would consist of 50 vectors, one for each state. The vector for California would have a 1 in its ith position if record i pertains to a person from California.

Key Applications

Indexes are necessary in any application that handles large amounts of data. Which indexes to choose can have an enormous impact on performance. An index may reduce the time to execute a query from hours to a few seconds in one application, yet increase batch load time by a factor of 80 in another application. Add them with care.

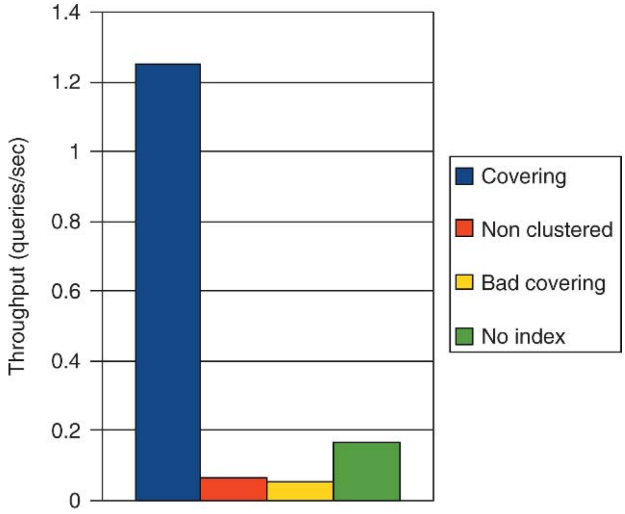
Experimental Results

Covering Experiment

This experiment illustrates two of the issues discussed above: (i) the potential benefit of covering index, and (ii) the trade-off between scans and non-clustered indexes. This experiment considers the table ACCOUNT with 25 attributes, and the following query:

select home_street, last_name from ACCOUNT where first_name='first_name12';

The ACCOUNT table has three different indexes: (i) a covering index on first_name, home_street and last_name, (ii) another covering index on home_street, first_name and last_name, and (iii) a non-clustered index on first_name. The experiment runs the same



Index Tuning. Figure 1. Covering experiment.

query using these three different indexes as well as with no index. The experiment runs those queries on MySQL 6.0 with a cold buffer (i.e., the database cache is empty and IO are required to complete the query). Figure 1 traces the results.

First, the covering index provides the best performance. The reason is that the query can be answered using the index. There is no need to dereference the index leaf pointers to access the underlying table. Second, the order of attributes in the covering index is crucial. In this query, the attribute first_name is used in the WHERE clause. A covering index has the most beneficial effect when the prefix attributes in the index key are those attributes that appear in the where clause. Third, scan sometimes wins over a non clustered index. The reason is that the few random IOs that are necessary to complete the query (the data is generated so that first_name has a 1% selectivity) take more time than the sequential IOs required to scan the whole table.

URL to Code and Data Sets

Index experiments: http://www.databasetuning.org/?sec=index

Cross-references

▶ B+-Tree
▶ Bitmap Index
▶ Hash-based Indexing

Recommended Reading

1. Celko J. Joe Celko's SQL for Smarties: Advanced SQL Programming (3rd edn.). Morgan Kaufmann, San Fransisco, CA, 2005.
2. Kimball R. and Ross M. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (2nd edn.). Wiley, New York, NY, 2002.
3. Shasha D. and Bonnet P. Database Tuning: Principles, Experiments and Troubleshooting Techniques. Morgan Kaufmann, San Fransisco, CA, 2002.
4. Tow D. SQL Tuning. OReilly, Sebastopol, CA, 2003.

Indexed Sequential Access Method

Alex Delis[1], Vassilis J. Tsotras[2]
[1]University of Athens, Athens, Greece
[2]University of California-Riverside, Riverside, CA, USA

Synonyms

Indexed sequential file; ISAM file; ISAM

Definition

An indexed sequential access method is a *static, hierarchical*, disk-based index structure that enables both (single-dimensional) range and membership queries on an ordered data file. The records of the data file are stored in *sequential* order according to some data attribute(s). Since ISAM is static, it does not change its structure if records are added or deleted from the data file. Should new records be inserted into the data file, they are stored in an *overflow* area. Deleted records are removed from the file (leaving empty space).

Historical Background

Although transparent for the user of a DBMS, access methods play a key role in database performance. A major performance goal of a DBMS is to minimize the number of I/Os (i.e., blocks or pages transferred) between the disk and main memory. One way to achieve this goal is to minimize the number of I/Os when answering queries. Note that many queries reference only a small portion of the records in a database table. For example the query: "*find the employees who reside in Santa Monica, CA*" references only a fraction of the records in the *Employee* relation. It would be rather inefficient to have the database system sequentially read all the pages of the *Employee* file and check the residence field of each employee record for the name '*Santa Monica*'. Instead the system should be able to locate the pages with '*Santa Monica*' employee records directly. To allow such fast access, additional disk-resident structures called *indices* (or *access methods*) are designed per database relation. One of the first such methods developed was the *index sequential access method* (ISAM). ISAM was developed at IBM in late 1960s [3] and it is essentially the predecessor to the widely used B+-tree index. A major difference between ISAM and the B+-tree [1] is that instead of overflowing pages, the B+-tree introduces page splitting. The ISAM was later replaced by IBMs *virtual storage access method* (VSAM) [4] which introduced the notion of splitting (data or index pages) when there is not enough space for inserting a new record.

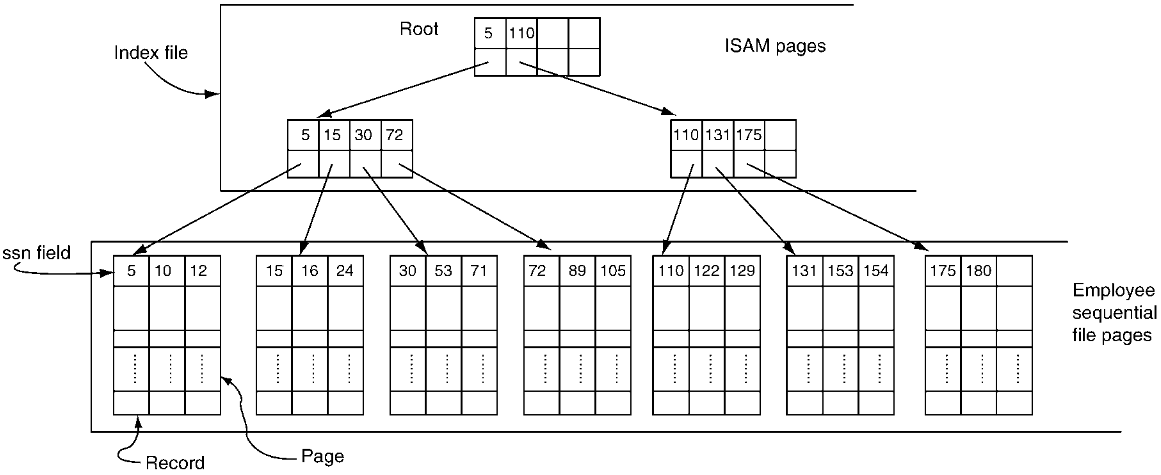
Foundations

The ISAM structure contains three separate storage areas: the data file, the index file and the overflow area. For simplicity, assume that the data file is an *Employee* relation, ordered according to the social security number or *ssn* attribute. Moreover, assume that this relation is stored sequentially on the disk, following the logical order of the *ssn* attribute. If the *Employee* file has n records and one page can hold B *Employee* records, the total number of pages in this file is $O(n/B)$. Note that each file page is full of *Employee* records except possibly the last page. Moreover, given the sequential storage of the file, each page can easily access the next page of the file in *ssn* order (it is simply the next physical page on the disk).

A straightforward way to build an index on the *Employee* file is to create a new (much smaller) file that contains one representative record from each *Employee* file page. The records in this new file are of the form: <*search_value, ptr*> where *ptr* is a pointer to an *Employee* file page (a *page-id* number uniquely identifying the page on the disk) and *search_value* is the smaller *ssn* recorded in that page. Since these records are smaller in size than the *Employee* file records, each page of the new file will contain many of them. If the *Employee* file is large, the new file will spread over a number of pages (this number is clearly bounded by $O(n/B^2)$). However, since the new file is also an ordered file (it has two attributes and is ordered according to the *search_value* attribute) it can be indexed by another (even smaller) level of index pages, and so on. This process continues until the creation of an index layer that consists of a single page. As a result a multi-way, tree-structured index is created whose nodes correspond to pages (see Fig. 1). It is worth pointing out that all index pages from possibly multiple index levels are resident in the index file area of the ISAM.

The ISAM organization is a single-dimensional (as opposed to multi-dimensional) index. It supports searches on the attribute (or collection of attributes) on which the data file is ordered. For example, searching the *indexed sequential access method* for a given *ssn* K (i.e., a membership query) is simple. The search starts from the root page where the record with the largest *search_value* that is less or equal to K is located. The search then continues to the page in the next index level, pointed by this record, until a page of the *Employee* file is reached. If K is found among the *ssn* values of that *Employee* page, the appropriate record is returned as answer to the query. If K is not found the answer is empty. It is easy to see that this search takes $O(logB(n/B))$ page accesses (I/Os) as this is the height (in pages) of the tree. The reader should note that the logarithm is base B, the size of the page, since this is a multi-way tree where each node has $O(B)$ fan-out.

Range queries (as in: *find the Employee records with ssn in the range* [25, 100]) are addressed similarly. A search is first performed for the *ssn* defining the lower part of the range (in the above query example this would be *ssn* = 25). This look-up will lead to an appropriate *Employee* record located in some file page. Records with higher *ssn* values within this page



Indexed Sequential Access Method. Figure 1. An indexed sequential access method.

are accessed until a record with *ssn* larger than the upper limit of the query range is found. If the upper limit of the query range is higher than the highest *ssn* in this page, the next page of the file is accessed and so on (recall that the file is stored sequentially). The search stops when an *Employee* page is found that contains a record with *ssn* larger than the query range.

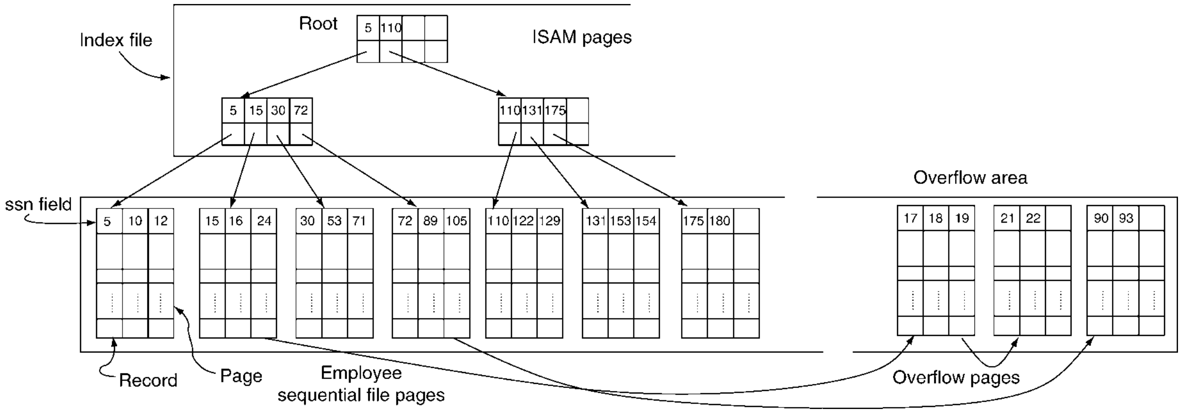
If a denotes the answer size to a range query (number of *Employee* records satisfying the query range predicate), ISAM answers a range query in $O(logB(n/B)+a/B)$ I/Os. Note that the logarithmic part is spent to find the *Employee* page with the first record that satisfies the query predicate (if any) and the $O(a/B)$ part corresponds to accessing the rest of the *Employee* pages that contain answer.

While the use of the index greatly facilitates query time, there is of course a space overhead, since the access method itself uses pages to store its records. However, this overhead is minimal. The number of pages used by the index structure is still bounded by $O(n/B)$. This is because the first level uses at most $O(n/B^2)$ pages, the second at most $O(n/B^3)$ and so on.

An interesting observation is that an indexed sequential access method imitates binary search on a disk-based environment. However, given that at each node of the index a whole page is accessed, there are $O(B)$ choices (instead of just 2 in the binary search) at each node.

The main advantages of the ISAM organization are its simplicity, small space overhead and fast query time. The structure however is *static*. If new records are added in the Employee file they are handled in an *overflow* file. Since there is no empty space in the data file, overflow pages are created to store the new records. Such pages are typically chained to the page where a record should have been stored (see Fig. 2). Various proposals exist on how to handle the overflow file [2,4,5,6]. Nevertheless, the structure of the index does not change as the size of the data file changes. This eventually affects query time. The overflow file can be merged periodically with the main *Employee* file, at which time the index needs to be recreated. Similarly, if records are deleted in the original *Employee* file, pages may be left containing very few records which affects both storage and query time. These problems are solved by the B+-tree, which is a *dynamic* indexing scheme [1].

There are two main differences between ISAM and B+-tree: firstly, when a new page is created in the B+-tree, space is left to accommodate future insertions. In practice, a newly created page starts half empty so that it can store many new records before a structural re-organization is needed. If the page becomes full of records and a new record is directed to it, the page is split into two pages (that are half full). Secondly, a page in the B+-tree is not allowed to become scarce of records (unless it is the tree's root page). As a result, when a page is accessed, it is guaranteed to contain a minimum number of records. If due to deletions a pages record occupancy falls below the threshold (half the page size) the page is merged with another page so that the combination has enough records. Note that leaving pages half empty imposes additional space overhead for the B+-tree than the ISAM; however it



Indexed Sequential Access Method. Figure 2. The indexed sequential access method with overflows.

results into a very effective dynamic height-balanced indexing scheme.

Finally, ISAM can be considered as the tree-based index alternative to the static external hashing. Both schemes are static and overflow areas are used for additional records. Their major difference is that ISAM can perform both range and membership queries, while static external hashing is designed only for membership queries.

Key Applications

ISAM has been used in early database management systems as an index method to provide fast access to range and membership queries. It was later replaced by the VSAM structure [4] which introduced the notion of page splitting. Finally, the B+-tree was proposed as a dynamic indexing structure [1] and is now the standard access method in most relational database systems.

Cross-references

- ▶ B+-Tree
- ▶ Indexing
- ▶ Membership Query
- ▶ Range Query

Recommended Reading

1. Bayer R. and McCreight E. Organization and maintenance of large ordered indexes. Acta Inf., 1(3):173–183, 1972.
2. Behymer J.A., Ogilive R.A., and Merten A.G. Analysis of indexed sequential and direct access file organizations. In Proc. 1974 ACM SIGFIDET (SIGMOD) Workshop on Data Description, Access and Control, 1974, pp. 186–212.
3. IBM Corporation. IBM System/360 Operating System Data Management Services, February 1972. Second Edition, C26-3746-1.
4. Keehn D.G. and Lacy S.O. VSAM Data set design parameters. IBM Syst. J. 13(3):186–212, 1974.
5. Larson P. Analysis of index-sequential files with overflow chaining. ACM Trans. Database Syst., 6(4):671–680, 1981.
6. Mullin J.K. An improved index sequential access method using hashed overflow. Commun. ACM, 15(5):301–307, May 1972.
7. Wong K.F. and Strauss J.G. An analysis of ISAM performance improvement options. Manag. Datamat. J., 4(3):95–107, 1975.

Indexed Sequential File

▶ Index Sequential Access Method

Indexing

▶ Biomedical Scientific Textual Data Types and Processing

▶ Index Creation and File Structures

▶ Physical Database Design for Relational Databases

Indexing and Similarity Search

Michail Vlachos
IBM T.J. Watson Research Center, Hawthorne,
New York, USA

Synonyms

Data organization; Hierarchical data organization; Space segmentation; Space partitioning

Definition

Indexing refers to the process of efficient data organization. It is closely related to similarity search because it allows such costly operations over a large dataset of objects to be efficiently sped up. Indices (or indexes) are hierarchical structures that direct the search to the most promising part of the database, hence eliminating from examination a large portion of objects. One can make the analogy with phone books, where all entries are recorded in sorted alphabetical order; therefore search involves only the lookup at the relevant portion of the book.

Historical Background

Traditional indexing structures include the B-trees. However, B-trees organize the data based on a single attribute/feature. Many of todays multimedia data contain hundreds or thousands of features. As an example, a small B&W image of 50 × 50 pixels contains 2,500 points/features. In order to accommodate objects that contain more attributes, extensions to the indexing schemes have been presented. Such indexes include kd-trees [3], grid files [9], as well as R-trees [7] and its variants [2], and the various incarnations of metric trees (VP-trees [12], M-trees [6]). The ultimate goal of a successful index is to divide object space into areas with approximately equal density using a set of heuristics.

Indexes work on simplified representations of the original data, since they do not perform well for raw data with high dimensionalities. This phenonenon is known as the "curse of dimensionality", which means that for higher dimensionalities, the pruning power of the indices diminishes exponentially, and all sequences are eventually retrieved from the disk. Therefore, when an index is utilized, it is necessary to perform some data compression, also called dimensionality reduction, because this data compaction will boost the index performance.

After the raw data are compacted from the original dimensionality n to some lower dimensionality d, the compressed d-dimensional objects/points are stored in the index structure in order to expedite the search process.

Foundations

An index can facilitate the fast similarity search over the database of objects. Therefore, a user is posing a query object q against a database and is seeking the most similar (or k most similar) objects to the query q, for a given similarity measure that assesses the affinity between a pair of objects.

Index structures attribute their search efficiency to two factors:

1. *Effective space partitioning*. Similar objects are grouped at the same portion of the index, such that for a given query large parts of the database can be eliminated from examination.
2. *Usage of simplified versions of the data objects*. This shrinks significantly the index size, compared to the storage requirements of the original database. Therefore, search operations on the index are much faster than on the original uncompressed data.

Consider a large database of objects, over which fast search needs to be enabled. If objects are high-dimensional, they will first be simplified in order to be stored in an index. Techniques like PCA, Fourier or Wavelet transform, etc., can be utilized at this step. For this example, in order to enhance visualization, the database objects correspond to images. The images can be simplified and represented using two features: the "number of red pixels" and the "number of blue pixels". Using these features, each image will be represented as a point on a two-dimensional space. Given these projected dimensions, two sets of images depicting underwater and wildlife images will be clustered as shown in Fig. 1. The way this new space is partitioned and searched, depends on the specifics of the utilized index.

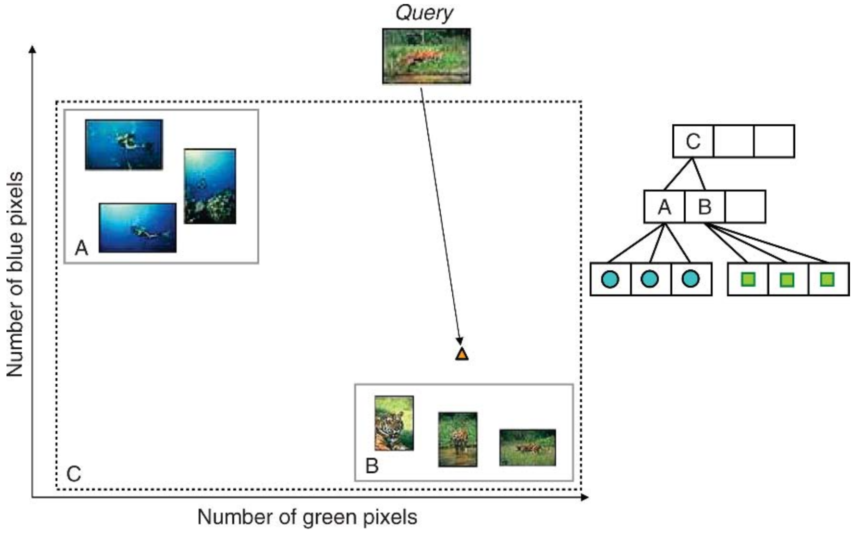
Key Applications

The following sections explain in more detail the inner-workings of widely used instances of index structures, such as the R-trees (space partitioning index) and the VP-trees (metric index).

R-Trees

The R-tree structure represents an extension of the B-tree for multiple dimensions. It has been proposed in 1984 by Antonin Guttman [7] and since then it has been utilized extensively in the database and data-mining fields.

A d-dimensional R-tree is a hierarchical structure of rectangles, with leaves and intermediate nodes. The leaves store d-dimensional hyper-rectangles



Indexing and Similarity Search. Figure 1. *Left*: Database of images and a potential two-dimensional representation. *Right*: The corresponding R-tree is a hierarchical structure of rectangles.

(or *d*-dimensional points) and a pointer to the object they describe. The intermediate nodes store a hyper-rectangle that completely contains the rectangles of their child-nodes, as well as pointers to the child-nodes.

In Fig. 1 a potential R-tree structure is depicted, based on the previously discussed example. Suppose now, that a query *q* is posed against the database of images. The query is mapped into a new point in the projected space (shown as a triangle in Fig. 1). For a *range search* one needs to find the overlapping rectangles within the requested search radius. for a *k-NN search* [10,11] one can define a `MINDIST` operator between a point and a rectangle and start traversing the tree according to the most promising path, while recording the remaining paths in a priority queue. The tree traversal can be either depth-first or breadth-first.

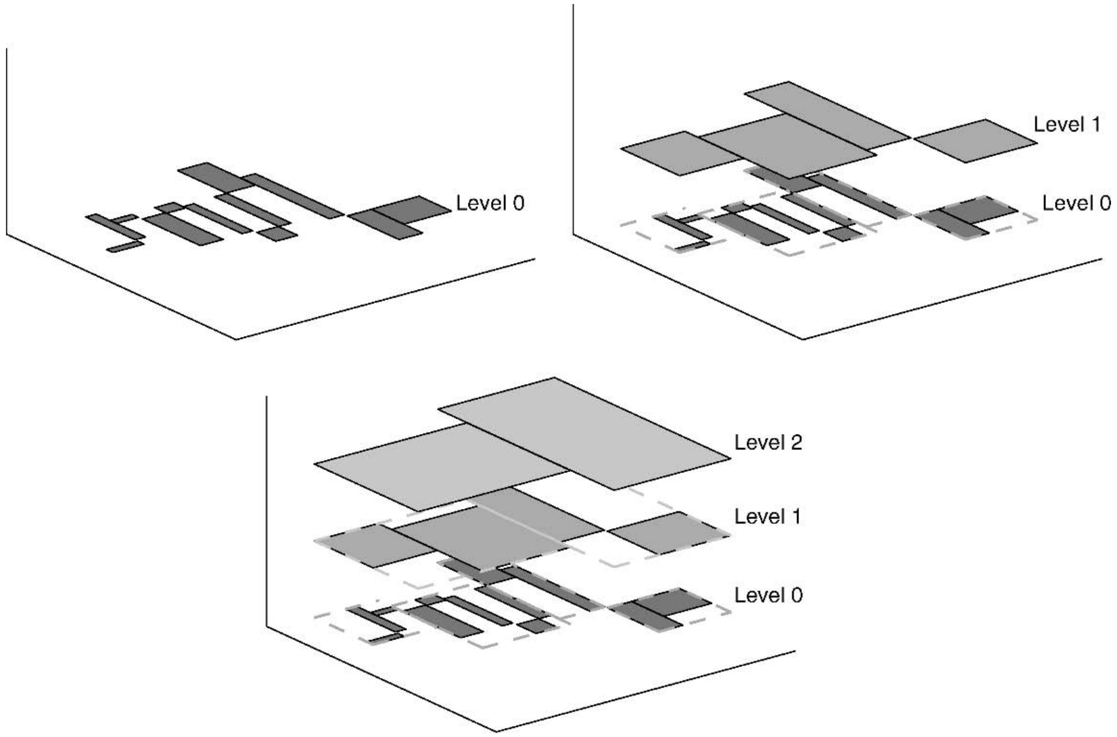
For example, if the user is seeking the 3-NN of the query *q*, then rectangle *C* will be pushed into the priority queue (see Fig. 1). It will be popped out and its children (*A* and *B*) will be pushed in and sorted according to their minimum distance to *q*. Subsequently, rectangle *B* will be examined first, because it is the closest one to the query. Its objects will be retrieved and their true distance to *q* shall be calculated. The search will end here, because the third closest neighbor found so far, has distance smaller than *MINDIST* (*q*, *Rectangle*(*A*)). Therefore, none of the objects of rectangle *A* need to be retrieved from disk.

Figure 2 demonstrates another example of the hierarchical structure of an R-tree.

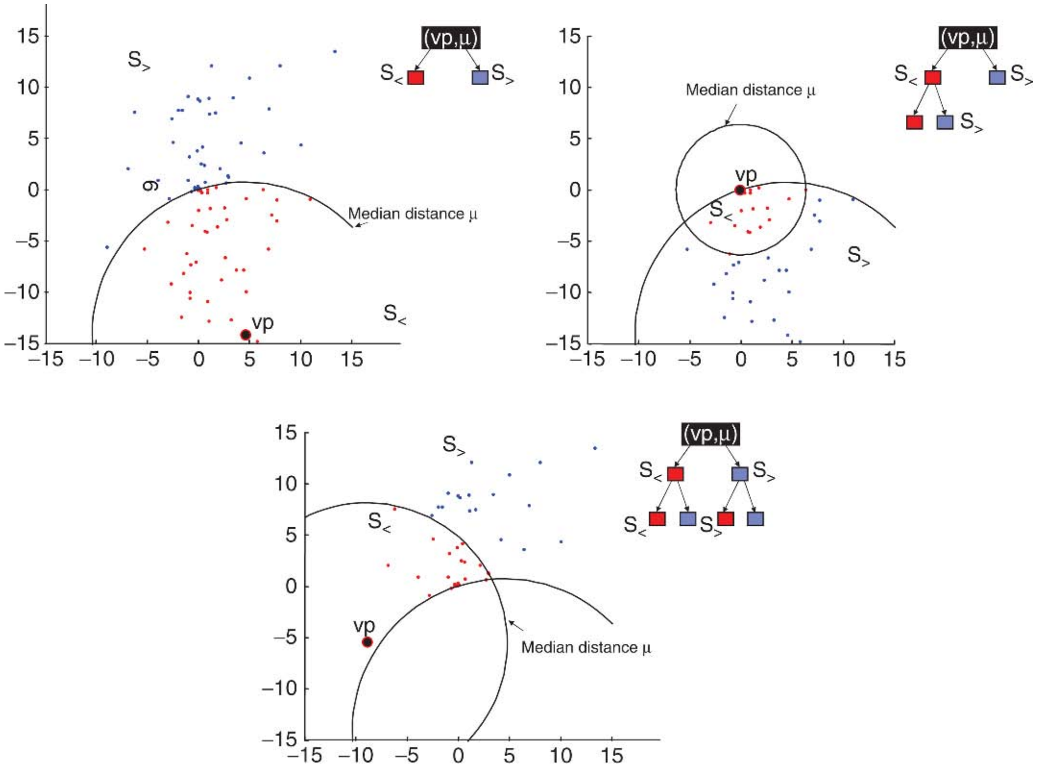
VP-Trees

According to the relevant bibliography, the pruning power of space partitioning indexing structures like R-trees, and its variants, degrades for data dimensionalities larger than 5–8. This means that at high dimensionalities the index will retrieve the majority of objects from the disk.

Metric trees exhibit better performance at larger dimensionalities (e.g., up to 20–25 dimensions). Metric trees utilize the distances between objects in order to create the tree and direct the search process. A popular instance of metric-trees are the VP-trees [4,5,12]. VP-trees partition the space based on distances to selected *vantage points* of the dataset. A tree containing the objects is constructed by recursively partitioning the dataset points into two distinct sets based on the median distance μ to the vantage point/object; the points that are closest to the vantage point ($S_<$) are stored on the left subtree, and those that are further away from the median distance ($S_>$) are directed on the right subtree. The process is repeated recursively and a different vantage point is selected for each of the remaining subsets. Figure 3



Indexing and Similarity Search. Figure 2. Bottom-up hierarchical construction of the two-dimensional R-tree.



Indexing and Similarity Search. Figure 3. Illustration of the creation of a VP-tree.

illustrates this process on two-dimensions for clarity (each point essentially represents one object).

After the tree is constructed and a query is posed, one only has to examine the proper subset based on the position of the query. Only if the query lies close to the median distance, both subsets need to be examined, otherwise one of them is discarded from examination.

Cross-references

▶ Curse of Dimensionality
▶ Data Partitioning
▶ Dimensionality Reduction
▶ Multimedia Data Indexing

Recommended Reading

1. Agrawal R., Faloutsos C., and Swami A. Efficient Similarity Search in Sequence Databases. In Proc. 4th Int. Conf. on Foundations of Data Organization and Algorithms, 1993, pp. 69–84.
2. Beckmann N., Kriegel H.-P., Schneider R., and Seeger B. The r*-tree: An efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 322–331.
3. Bentley J. Multidimensional divide and conquer. Commun. ACM, 23(4):214–219, 1980.
4. Bozkaya T. and Özsoyoglu M. Distance-based indexing for high-dimensional metric spaces. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 357–368.
5. Chee Fu A.W., Chan P.M., Cheung Y.L., and Moon Y. Dynamic VP-tree indexing for N-nearest neighbor search given pair-wise distances. VLDB J., 9(2): 154–173, 2000.
6. Ciaccia P., Patella M., and Zezula P. M-tree: An efficient access method for similarity search in metric spaces. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 426–435.
7. Guttman A. R-trees: A dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
8. Keogh E., Chakrabarti K., Pazzani M., and Mehrotra S. Locally adaptive dimensionality reduction for indexing large time series databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 151–162.
9. Nievergelt J., Hinterberger H., and Sevcik K.C. The grid file: An adaptable, symmetric multikey file structure. ACM Trans. Database Syst., 9(1):38–71, 1984.
10. Roussopoulos N., Kelley S., and Vincent F. Nearest neighbor queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 71–79.
11. Seidl T. and Kriegel H.-P. Optimal multi-step k-nearest neighbor search. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 154–165.
12. Yianilos P. Data structures and algorithms for nearest neighbor search in general metric spaces. In Proc. 4th Annual ACM - SIAM Symp. on Discrete Algorithms, 1993, pp. 311–321

Indexing Compressed Text

Paolo Ferragina, Rossano Venturini
University of Pisa, Pisa, Italy

Synonyms

Compressed full-text indexing; Compressed suffix array; Compressed suffix tree; Compressed and searchable data format

Definition

Given a text $T[1,n]$, the *Compressed Text Indexing* problem requires to building an indexing data structure over T that takes space close to the empirical entropy of the input text and answers queries on the occurrences of an arbitrary pattern $P[1,p]$ in T without any significant slowdown with respect to uncompressed indexes. There are three main queries: `count`(P), that returns the number of pattern occurrences in T, `locate`(P), that returns the starting positions of all pattern occurrences in T, and `extract`(i, j), that retrieves the substring $T[i, j]$.

Historical Background

String processing and searching tasks are at the core of modern web search, information retrieval (IR), data base and data mining applications. Most of text manipulations required by these applications involve, sooner or later, *searching* those (long) texts for (short) patterns or *accessing* portions of those texts for subsequent processing/mining tasks. Despite the increase in processing speed of current CPUs and memories/disks, sequential text searching long ago ceased to be a viable approach, and indexed text searching has become mandatory.

A *(full-)text index* is a data structure built over a text $T[1,n]$, drawn from an alphabet Σ of size σ, which significantly speeds up sequential searches for *arbitrary* pattern strings, at the cost of some additional space. *Suffix trees* and *suffix arrays* are the most well-known full-text indexes [5]. The *suffix tree* of a text T is a trie (or digital tree) built on all the n text suffixes $T[i, n]$, where unary paths are compacted to ensure $O(n)$ overall size. The suffix tree has n leaves, one per text suffix, and each internal node corresponds to a unique substring of T that occurs more than once. The suffix tree can count the *occ* occurrences of any pattern $P[1,p]$ in time $O(p)$ by descending in the suffix tree according to the symbols of P, and it can locate these occurrences in optimal $O(occ)$ time by traversing the subtree of the node reached by counting. The suffix tree, however, uses much more space than the text itself because it requires $\Theta(n \log n)$ bits, whereas the text needs $n\lceil \log \sigma \rceil$ bits (logarithms are in base 2). In practice, a suffix tree requires from 10 to 20 times the text size, if carefully engineered [5].

The *suffix array* is a compact version of the suffix tree, obtained by storing in $SA[1,n]$ the starting positions of the suffixes of T listed in lexicographical order. This data structure still requires $\Theta(n \log n)$ bits in the worst case, but the constant hidden in the big-Oh notation is small in practice, namely it is no more than 4. SA can be obtained by traversing the leaves of the suffix tree, or it can be built directly in optimal linear time via ad-hoc sorting methods [5]. Since any substring of T is the prefix of a text suffix, finding all pattern occurrences boils down to finding all text suffixes that start with P. These suffixes form a lexicographic interval in SA that can be binary searched in $O(p \log n)$ time, as each comparison in the binary search requires examining up to p symbols of the pattern and of a text suffix. The time complexity can be improved to $O(p + \log n)$ by using an auxiliary data structure that doubles the space requirement of the suffix array, or it can be further reduced to $O(p + \log \sigma)$ by a proper sampling of the indexed suffixes (cfr. Suffix Trays). Once the interval $SA[sp,ep]$ containing all text suffixes starting with P has been identified, `count`(P) is answered by returning the value $occ = ep - sp + 1$, and `locate`(P) is answered by retrieving the entries $SA[sp]$, $SA[sp + 1]$,...,$SA[ep]$.

The use of full-text indexes is not limited to (full-) text searching over one single text. It can be easily extended to multiple texts, and can also be used to support (prefix, suffix, or substring) queries over a dictionary $\mathcal{D}$ of strings having variable length. This problem is called *Dictionary Indexing* and occurs frequently in the implementation of IR and data mining applications. It can be solved via a (compressed) index built on a string S_D which is obtained by concatenating all dictionary strings, separated with a special symbol #. A prefix search for P in $\mathcal{D}$ can be implemented by counting/locating the query pattern #P in S_D; a suffix search can be implemented by searching for P# in S_D; substring searches are directly executed on S_D.

Foundations

The large space occupancy of full-text indexes has driven programmers to resort to inverted indexes to solve their searching operations on large textual datasets, and some researchers have actually concluded that the increased query power of full-text indexes has to be paid by additional storage space. Fortunately, a recent body of research showed that *compressed* full-text indexes can be designed by deploying algorithmic techniques and mathematical tools which lie at the crossing point of three distinct fields – data compression, algorithmics and databases (see e.g., [9,13,14]). Most of these indexes can be classified into two families – FM-indexes (FMI) and Compressed Suffix Arrays (CSA) – and achieve efficient query times and space close to the one achievable by the best known compressors, like gzip or bzip2. In theory, these indexes require $O(nH_k(T)) + o(n \log \sigma)$ bits of space, where $H_k(T)$ is the kth order empirical entropy of T (see Table 1). This bound is appealing because it can be sublinear in n, for highly compressible texts, and $nH_k(T)$ is the classic Information-Theoretic lower bound to the storage complexity of T by means of any kth order compressor, like gzip and bzip2 (recall that `extract`$(1,n) = T$).

The FM-Index Family

These compressed indexes were introduced by Ferragina and Manzini in [9], who devised a way to orchestrate in efficient time and space the relation that exists between the suffix array data structure and the *Burrows-Wheeler Transform* (shortly, BWT [4]). The BWT is a reversible transformation that permutes the symbols of the input string T into a new string `bwt`(T) which is easier to compress, and can be computed in three steps (see Fig. 1):

1. Append at the end of T a special symbol \$ smaller than any other symbol of Σ;
2. Form a *conceptual* matrix $\mathcal{M}(T)$ whose rows are the cyclic rotations of string $T\$$ in lexicographic order;
3. Set string `bwt`(T) to the last column L of the sorted matrix $\mathcal{M}(T)$.

Every column of $\mathcal{M}(T)$, hence also the transformed string L, is a permutation of $T\$$. In particular the first column of $\mathcal{M}(T)$, call it F, is obtained by lexicographically sorting the symbols of $T\$$ (or, equivalently, the symbols of L). Note that the sorting of the rows of

Indexing Compressed Text. Table 1. Best known complexities for the time (in big-Oh) and space (in bits) required by the main families of compressed full-text indexes

Index	Count	Locate	Extract	Space	References
FMI	p	$occ{\cdot}\mathtt{polylog}(n)$	$\ell + \mathtt{polylog}(n)$	$nH_k(T) + o(n)$	[8]
CSA	$p/\log_\sigma n + \mathtt{polylog}(n)$	$occ{\cdot}\mathtt{polylog}(n)$	$\ell/\log_\sigma n + \mathtt{polylog}(n)$	$\gamma^{-1}nH_k(T) + o(n)$	[11]
LZ-INDEX	$p^2 \log p + p \log n + occ$	$occ{\cdot}\log n$	$\ell(1 + \epsilon^{-1}/\log_\sigma \ell)$	$(2 + \epsilon)nH_k(T) + o(n \log \sigma)$	[1]

Here $\epsilon > 0$ and $0 < \gamma < \frac{1}{3}$ are constants fixed in advance before the data structures are built; ℓ is the number of text symbols to be retrieved by extract; $H_k(T)$ is the k-th order empirical entropy of text T [14]. The reported complexities are worst-case and hold for $\sigma = O(\mathtt{polylog}(n))$ assuming that $k \le \alpha \log_\sigma n$ with $0 < \alpha < 1$ except for LZ-INDEX in which $k = o(\log_\sigma n)$. For more precise bounds (e.g., coefficients in $\mathtt{polylog}(n)$ terms and the case $\sigma = \Omega(\mathtt{polylog}(n))$ and for a thoughtful comparison of these indexes and their numerous variants, the reader is referred to [14].

		F		L
mississippi$		$	mississipp	i
ississippi$m		i	$mississip	p
ssissippi$mi		i	ppi$missis	s
sissippi$mis		i	ssippi$mis	s
issippi$miss		i	ssissippi$	m
ssippi$missi	⟹	m	ississippi	$
sippi$missis		p	i$mississi	p
ippi$mississ		p	pi$mississ	i
ppi$mississi		s	ippi$missi	s
pi$mississip		s	issippi$mi	s
i$mississipp		s	sippi$miss	i
$mississippi		s	sissippi$m	i

Indexing Compressed Text. Figure 1. Example of Burrows-Wheeler transform for T = mississippi. The matrix on the right has the rows sorted in lexicographic order. The output of the BWT is the column L = ipssm$pissii.

$\mathcal{M}(T)$ is essentially equal to the sorting of the suffixes of T, because of the presence of the special symbol \$. This shows that: (i) symbols preceding the same substring (*context*) in T are grouped together in L, and thus give raise to clusters of nearly identical symbols; (ii) there is an obvious relation between $\mathcal{M}(T)$ and SA. Property (1) is the key for devising modern data compressors, Property (2) is crucial for designing compressed indexes and, additionally, suggests a way to compute the BWT through the construction of the suffix array of T: $L[0] = T[n]$ and, for any $1 \le i \le n$, set $L[i] = T[SA[i] - 1]$.

Burrows and Wheeler [4] devised two properties for the invertibility of the BWT:

1. Since the rows in $\mathcal{M}(T)$ are cyclically rotated, $L[i]$ *precedes* $F[i]$ in the original string T.
2. For any $c \in \Sigma$, the ℓth occurrence of c in F and the ℓth occurrence of c in L correspond to the *same* symbol of the string T.

As a result, the original text T can be obtained backwards from L by resorting to a function LF that maps row indexes to row indexes, and is defined as follows: if the BWT maps $T[j-1]$ to $L[i']$ and $T[j]$ to $L[i]$, then $LF(i) = i'$ (so LF implements a sort of *backward* step over T) [9]. Now, since the first row of $\mathcal{M}(T)$ is \$T, it can be stated that $T[n] = L[0]$ and, in general, $T[n-i] = L[LF^i(0)]$, for $i = 1,\ldots,n-1$.

Starting from these basic properties, Ferragina and Manzini [9] proposed a way to combine the compressibility of the BWT with the indexing power of the suffix array. In particular, they have shown that searching operations on T can be reduced to counting queries of *single* symbols in L, now called rank operations. For any symbol $c \in \Sigma$ and position i in L, the query $\mathtt{rank}_c(L, i)$ returns how many times the symbol c appears in $L[1,i]$. An FM-index then consists of three key tools: a compressed representation of $\mathtt{bwt}(T)$ that supports efficient rank queries, a small array $C[c]$ which tells how many symbols smaller than c appear in T (this takes $O(\sigma \log n)$ bits), and the so called *backward search* algorithm which carefully orchestrates the former two data structures in order to implement efficiently the count query. More precisely, FMI searches the pattern $P[1,p]$ backwards in p steps, which eventually identify the interval of text suffixes that are prefixed by P or, equivalently, the interval of rows of $\mathcal{M}(T)$ that are prefixed by P. This is done by maintaining, inductively for $i = p, p-1,\ldots,1$, the interval $SA[sp_i, ep_i]$ that stores all text suffixes that are prefixed by the pattern suffix $P[i, p]$. At the beginning it is $i = p$, and so $SA[sp_p, ep_p]$ corresponds to all suffixes which are prefixed by the last symbol $P[p]$: hence, it is enough to set $sp_p = C[P[p]] + 1$ and $ep_p = C[P[p] + 1]$. At any other step, the algorithm has inductively computed $SA[sp_{i+1}, ep_{i+1}]$, and thus it can derive the next interval of suffixes prefixed by $P[i, m]$ by setting $sp_i = C[P[i]] + \mathtt{rank}_{P[i]}(L, sp_{i+1} - 1) + 1$ and $ep_i = C[P[i]] + \mathtt{rank}_{P[i]}(L, ep_{i+1})$. These two computations are actually mapping (via *LF*) the first and last occurrences (if any) of symbol $P[i]$ in the substring $L[sp_{i+1}, ep_{i+1}]$ to their corresponding occurrences in *F*. (Indeed, [9] showed that any *LF* computation boils down to a `rank` query on *L*.) As a result, the backward-search algorithm requires to solve $2p$ rank queries on $L = \mathtt{bwt}(T)$ in order to find out the (possibly empty) range $SA[sp, ep]$ of text suffixes prefixed by *P*. `count`(P) can be then solved by returning the value $occ = ep_1 - sp_1 + 1$.

Conversely, `locate` and `extract` need some extra information about the underlying suffix array, this impacts onto the space occupancy of the FMI. Recall that `locate`(P) requires to return, for any $i \in [sp, ep]$, the position $pos(i) = SA[i]$. For space reasons *SA* cannot be stored explicitly so that, for a fixed parameter $\mu = \lceil \log^{1+\in} n \rceil$, FMI samples the rows of $\mathcal{M}(T)$ which correspond to text suffixes that start at positions of the form $1 + j \cdot \mu$. Each such pair ⟨row, position⟩ is stored explicitly in a data structure $\mathcal{S}$ that supports membership queries in constant time (on the *row*-component). Now, given a row index *i*, the value $pos(i)$ can be derived immediately from $\mathcal{S}$, if *i* is a sampled row; otherwise, the algorithm computes $j = LF^t(i)$, for $t = 1,2,\ldots,$ until *j* is a sampled row. In this case, $pos(i) = pos(j) + t$. The sampling strategy ensures that a row in $\mathcal{S}$ is found in at most μ iterations, and thus the *occ* occurrences of the pattern *P* can be located via $O(\mu \cdot occ)$ `rank` queries. The algorithm for `extract`(i,i') requires a similar approach and takes no more than $(i' - i + \mu + 1)$ rank queries.

The net result is that the space and time complexities of FMI depend on the value μ and on the performance guaranteed by the data structure used to compute rank queries on the BWT-string. The extra space required by the data structures added to support `locate` and `extract` is bounded by $O((n \log n)/\mu)$ bits, which is $o(n)$ whenever $\in > 0$. The real challenge thus consists of representing $\mathtt{bwt}(T)$ in a compressed form and answering efficiently rank queries over it. Actually, all implementations of FMI differentiate themselves by the strategy used to solve this problem, as the alphabet size grows. Today, the literature offers many solutions, the most efficient ones are summarized below.

Lemma 1 *Let* $T[1,n]$ *be a string over an alphabet of size* σ, *and let* $L = \mathtt{bwt}(T)$.

1. For $\sigma = O(\mathtt{polylog}(n))$, *there exists a data structure which supports* `rank` *queries on L in* $O(1)$ *time using* $nH_k(T) + o(n)$ *bits of space, for any* $k \leq \alpha \log_\sigma n$ *and* $0 < \alpha < 1$, *and retrieves any symbol of L in the same time bound* [10, *Theorem 5*].
2. *For general* Σ, *there exists a data structure which supports* `rank` *queries on L in* $O(\log \log \sigma)$ *time, using* $nH_k(T) + n\ o(\log \sigma)$ *bits of space, for any* $k \leq \alpha \log_\sigma n$ *and* $0 < \alpha < 1$, *and retrieves any symbol of L in the same time bound* [2, *Theorem 4.2*].

By plugging this Lemma into the FMI data structure, one derives a compressed full-text index that supports efficiently the three full-text queries – namely, `count`, `locate`, `extract`– and occupies space approaching the *k*th order empirical entropy of *T* (see Table 1).

In practice, there are various implementations of FMI, whose engineering choices mainly refer to the way the `rank`-data structure built on $\mathtt{bwt}(T)$ is compressed and scales with the alphabet size of the indexed text. The site Pizza&Chili (see below) reports several implementations for FMI that mainly boil down to the following trick: $\mathtt{bwt}(T)$ is split into blocks (of equal or variable length) and values of $\mathtt{rank}_c$ are precomputed for all block beginnings and all symbols $c \in \Sigma$. A query $\mathtt{rank}_c(L, i)$ is answered by summing up the answer available for the beginning of the block that contains $L[i]$ plus the rest of the occurrences of *c* in that block – they are obtained either by sequentially decompressing the block or by using a proper compressed data structure built on it (e.g., the Wavelet Tree of [12]). The former approach favors compression, the latter favors query speed.

The CSA family. These compressed indexes were introduced by Grossi and Vitter [13], who showed how to compactly represent the suffix array *SA* in O(n log σ) bits and still be able to access any of its entries in efficient time. Their solution is based on a function Ψ, which is the inverse of the function LF introduced for BWT:

$$\Psi(i) = \begin{cases} i' \textit{such that } SA[i'] = SA[i] + 1 & (\textit{if } SA[i] > n) \\ i' \textit{such that } SA[i'] = 1 & (\textit{if } SA[i] = n) \end{cases}$$

In other words, $\Psi(i)$ refers to the position in the suffix array of the text suffix that follows $SA[i]$ in T, namely, the text suffix which is one-symbol shorter. The compact storage of SA proposed by Grossi and Vitter is based on a hierarchical decomposition that deploys Ψ. To represent $SA_0 = SA$ they use three vectors: B_0, Ψ_0 and SA_1. The binary vector $B_0[1,n]$ marks the entries of SA_0 which are even (suffixes). The vector $\Psi_0[1,\lceil n/2 \rceil]$ stores the values Ψ(i) for which $SA[i]$ is odd (hence $B_0[i] = 0$). The vector $SA_1[1,\lceil \mathrm{n}/2 \rfloor]$ is a "halved" version of SA_0, in that it contains the even elements of SA_0 divided by 2. Surprisingly enough, these three vectors suffice to retrieve any entry SA[i]. Of course, it is easy to determine whether $SA[i]$ is even or odd by simply looking at $B_0[i]$. If $SA[i]$ is odd, the following suffix $SA[i] + 1 = SA[\Psi(i)]$ is even, and its suffix-array position can be determined as $\Psi(i) = \Psi_0(\mathtt{rank}_0(B_0,i))$. If $SA[i]$ is even, it is enough to look at its *halved* value stored at $SA_1[\mathtt{rank}_1(B_0,\mathrm{i})]$. The three vectors Ψ_0, B_0 and SA_1 form the first level of the hierarchical decomposition of SA. This idea is applied recursively on SA_1 which is replaced by three other vectors: Ψ_1, B_1 and SA_2. This goes on until SA_h can be represented within O(n) bits, namely when $h = \lceil \log \log n \rceil$. Accessing $SA[i]$ takes h time. By storing the text T, in additional $n\lceil \log \sigma \rceil$ bits, one can search for a pattern P via the classic binary-search, now on the compacted SA. Grossi and Vitter proposed to store vectors B in compressed form via proper `rank`-data structures (see [14] and references therein), and deployed the *piecewise increasing property* for Ψ – namely, if $T[SA[i]] = T[SA[i + 1]]$, then $\Psi(i) < \Psi(i + 1)$ – to store each level of Ψ within $\frac{1}{2} n \log \sigma$ bits, still preserving constant time lookup to any level of Ψ. Other time/space tradeoffs are possible by using different numbers of levels. Essentially, not all the levels are represented and the function Ψ is used to jump from one represented level to the next represented one.

Recently, Csa has been the subject of two main improvements. The first one, due to Sadakane [16], showed that the original text T can be replaced with a binary vector F such that $F[i] = 1$ iff the first symbol of the suffixes $SA[\mathrm{i} - 1]$ and $SA[i]$ differs. Since the suffixes in SA are lexicographically sorted, one can determine the first symbol of any suffix in constant time by just executing *a rank*$_1$ query on F. This fact, combined with the retrieval of Ψ's values in constant time, allows comparing any suffix with the searched pattern $P[1,p]$ in time $O(p)$. Sadakane also provided an improved representation for Ψ achieving $nH_0(T)$ bits. Theoretically, the best variant of Csa is due to Grossi, Gupta and Vitter [12] who devised some further structural properties of Ψ that allow to come close to $nH_k(T)$ bits, still preserving the previous time complexities for all full-text queries (see Table 1). Practically, the best implementation of the Csa is the one proposed by Sadakane that actually does not use the hierarchical decomposition above, but orchestrates a compact representation of the function Ψ together with the backward search and the sampling strategy of the Fmi family. This *hybrid* index is among the fastest compressed indexes to count and locate pattern occurrences over highly-compressible data.

Other Compressed Indexes

Previous families of compressed indexes based their search on the implicit or explicit availability of the suffix array data structure. Recent years have seen the design of several other approaches, the two most notable ones are the *LZ-index*, proposed by Navarro, and the *Compressed Suffix Tree*, devised by Sadakane and then improved by many other authors. The former index bases its design on the parsing of the text T via the LZ78-compression scheme, and then enriches its output by additional data structures that support efficient searches over the parsed phrases. By properly orchestrating LZ78-parsing with compressed dictionary data structures, [1] achieved interesting search and entropy-based space bounds which are not competitive theoretically with the ones obtained by Fmi and Csa indexes (see Table 1) but are, nonetheless, fast in practice. As far as the compressed suffix-tree is concerned, it is worth noticing that the compression of this data structure is obtained by properly orchestrating succinct tree and succinct array encodings [15]. The total space is the one required by the Csa built on T plus no more than $6n + o(n)$ bits; all known suffix-tree operations are supported with a maximum slowdown of $O(\log\ n)$ time with respect to the uncompressed suffix tree.

Key Applications

Compressed full-text indexes might be used at the core of modern web search, IR, data base and data mining applications because, as Knuth observed in the Art of Computer Programming (vol. 3): "*space optimization is closely related to time optimization in a disk memory*". Data compression can not only squeeze the space overhead of an index, but also improve its speed, as remarked earlier. Several authors [3,5,11,17,18] have recently addressed these issues in various settings but, nonetheless, there is much more room for theoretical and practical improvements.

Future Directions

An open challenge concerning compressed indexes is to fasten their `locate` queries in order to achieve the optimal $O(occ)$ time bound. The best known result is due to Ferragina and Manzini [9]: each occurrence is located in constant time, and the index takes $O(nH_k(S) \log^{\in} n) + o(n \log \sigma \log^{\in} n)$ bits, where $\in$ is any positive constant. This bound has the extra log-factor in front of the entropy term! Therefore, it is natural to ask: Is there a full-text index achieving $O(p + occ)$ query time and $O(nH_k(S)) + o(n \log \sigma)$ bits of space occupancy in the worst case? This result would be *provably better* than any known uncompressed full-text index.

Another interesting open problem consists of designing a compressed full-text index which is disk-aware or, better, memory-oblivious in that it scales optimally over all memory levels available in a modern PC. The above data structures are compressed, but their overall size may span many memory levels so that issues pertaining to proper *arrangement of data* and properly *structured algorithmic computations* come into play. The most attractive disk-aware index is the String B-tree [7]; whereas the best cache-oblivious index is the COSB-tree [3,8]. Unfortunately the former is uncompressed, whereas the latter uses a compression heuristic which does not guarantee entropy-bounds in the worst case. It would be therefore valuable, also in practice, to devise a compressed index that combines the I/O-efficiency of the (cache oblivious) String B-tree with the space efficiency of the compressed full-text indexes discussed in this entry. Some preliminary results have been devised in [8], but the ultimate goal has yet to be achieved.

Experimental Results

Site Pizza&Chili [6] provides a full experimental comparison among the major implementations of compressed indexes. The experiments mainly show that these indexes can compress a text within 40–80% of its original size, and support searches for 20,000–50,000 patterns of 20 chars each within a second, locate about 100,000 pattern occurrences per second, and decompress text symbols at a rate of about 1MB/s. The compressed indexes are from one (`count`) to three (`locate`) orders of magnitudes slower than what one can achieve with a plain suffix array, at the benefit of using up to 18 times less space. This slowdown is due to the fact that search operations in compressed indexes access the memory in a non-local way thus eliciting many cache/IO misses, with a consequent degradation of the overall time performance. Nonetheless compressed indexes achieve a (search/extract) throughput which is significant and may match the efficiency specifications of most software tools running on commodity PCs. Recently, Ferragina and Venturini [11] provided a comparison among classic and compressed indexes for the Dictionary Indexing Problem showing that, in this case, compressed indexes may be faster than classic IR approaches.

Data Sets

Calgary Corpus (http://links.uwaterloo.ca/calgary.corpus.html)
Canterbury Corpus (http://corpus.canterbury.ac.nz)
Pizza&Chili Corpus (http://pizzachili.di.unipi.it or http://pizzachili.dcc.uchile.cl) see also[18]

URL to Code

Site Pizza&Chili (http://pizzachili.di.unipi.it or http://pizzachili.dcc.uchile.cl) collects implementations of the major compressed text indexes, and various tools and datasets to test them.

Cross-references

- ► Managing Compressed Structured Text
- ► Suffix Trees
- ► Text Compression
- ► Text Index Compression
- ► Text Indexing & Retrieval
- ► Text Indexing Techniques
- ► Text Representation
- ► XML Compression

Recommended Reading

1. Arroyuelo D., Navarro G., and Sadakane K. Reducing the space requirement of LZ-index. In Proc. 17th Annual Symposium on Combinatorial Pattern Matching, 2006, pp. 319–330.
2. Barbay J., He M., Munro J.I., and Srinivasa Rao S. Succinct indexes for string, binary relations and multi-labeled trees. In

Proc. 18th Annual ACM -SIAM Symp. on Discrete Algorithms, 2007, pp. 680–689.

3. Bender M.A., Farach-Colton M., and Kuszmaul B.C. Cache-oblivious string B-trees. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 233–242.
4. Burrows M. and Wheeler D. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.
5. Ferragina P. String Search in External Memory: Data Structures and Algorithms, In Handbook of Computational Molecular Biology, Chapman & Hall, London, 2005.
6. Ferragina P., González R., Navarro G., and Venturini R. Compressed Text Indexes: From Theory to Practice, J. Exp. Algorithmics, 13:1.12–1.31, 2009.
7. Ferragina P. and Grossi R. The String B-tree: A new data structure for string search in external memory and its applications. J. ACM, 46(2):236–280, 1999.
8. Ferragina P., Grossi R., Gupta A., Shah R., and Vitter J.S. On searching compressed string collections cache-obliviously. In Proc. 27th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2008, pp. 181–190.
9. Ferragina P. and Manzini G. Indexing compressed text. J. ACM, 52(4):552–581, 2005.
10. Ferragina P., Manzini G., Mäkinen V., and Navarro G. Compressed representations of sequences and full-text indexes. ACM Trans. Algorithms, 3(2), 2007.
11. Ferragina P. and Venturini R. Compressed permuterm index. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 535–542.
12. Grossi R., Gupta A., and Vitter J.S. High-order entropy-compressed text indexes. In Proc. 14th Annual ACM-SIAM Symp. on Discrete Algorithms, 2003, pp. 841–850.
13. Grossi R. and Vitter J.S. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. SIAM J. Comput., 35(2):378–407, 2005.
14. Navarro G. and Mäkinen V. Compressed full-text indexes. ACM Comput. Surv., 39(1), 2007.
15. Sadakane K. Compressed suffix trees with full functionality. Theory Comput. Syst., 41(4):589–607, 2007.
16. Sadakane K. New text indexing functionalities of the compressed suffix arrays. J. Algorithms, 48(2):294–413, 2007.
17. Sadakane K. Succinct data structures for flexible text retrieval systems. J. Discrete Algorithms, 5(1):12–22, 2007.
18. Tam S.L., Wong C.K., Lam T.W., Sung W.K., and Yiu S.M. Compressed indexing and local alignment of DNA. Bioinformatics, 24(6):791–797, 2008.

Indexing for Online Function Approximation

► Database Techniques to Improve Scientific Simulations

Indexing for Similarity Search

► High Dimensional Indexing

Indexing Granularity

► Indexing Units

Indexing Historical Spatio-Temporal Data

Mohamed F. Mokbel[1], Walid G. Aref[2]
[1]University of Minnesota, Minneapolis, MN, USA
[2]Purdue University, West Lafayette, IN, USA

Synonyms

Indexing the past; Historical spatio-temporal access methods; Trajectory indexing

Definition

Consider an object O that reports to a database server two consecutive locations $P_0 = (x_0,y_0)$ and $P_1 = (x_1,y_1)$ at times t_0 and t_1, respectively. The database server has no idea about the exact locations of object O between t_0 and t_1. To be able to answer queries regarding the user location at any time, the database server interpolates the two accurate locations through a trajectory that connects P_0 and P_1 through a straight line. While object O keeps sending location samples, the database server keeps accumulating set of consecutive trajectory lines that represent the historical movement of object O. Indexing historical spatio-temporal data includes dealing with such large numbers of trajectories. The main idea is to organize past trajectories in a way that supports historical spatial, temporal, and spatio-temporal queries.

Historical Background

The rapid increase in spatio-temporal applications calls for new auxiliary indexing structures. A typical spatio-temporal application is one that tracks the behavior of moving objects through location-aware devices (e.g., GPS). Through the last two decades, many spatio-temporal access methods were developed.

Spatio-temporal access methods focus on two orthogonal directions: (i) Indexing the past i.e., historical location data, (ii) Indexing the current and predicted future positions. This entry focuses on the former direction. A first approach to support spatio-temporal data is to extend existing spatial data (e.g., R-tree [4]) to support the temporal dimension. However, as the temporal dimension has distinct properties from other spatial dimensions, these approaches are later modified to give special attention to the temporal dimension.

Foundations

The main challenge in indexing historical spatio-temporal data is that the size of the history is continuously increasing over time. Consider moving objects that continuously send their positions. Keeping track of all updates is almost infeasible. Two approaches are used to minimize the history size: (i) *Sampling*. The stream of data is sampled at certain time positions. Linear interpolation may be used between sample points to form trajectory lines. (ii) *Update on change only*. Moving objects send information only when their data is changed (e.g., change in speed or direction). In general, spatio-temporal indexing methods for historical data can be categorized into the following three categories:

1. Three-Dimensional Structures

This category augments existing spatial access methods (e.g., the R-tree [4]) to support the newly introduced temporal dimension. Examples of this category include:

- *RT-tree* [15]: The RT-tree combines the foundation of the R-tree as a spatial access method and the TSB-tree [7] as a temporal access method. In the RT-tree, a new entry is added to the regular R-tree that indicates the start and end times of the current object. An RT-tree entry has the form (id, MBR, t_s, t_e), where id is the object identifier, MBR is the objects minimum bounding rectangle, and t_s and t_e give the time interval in which this object is valid. The RT-tree supports spatial queries as efficient as the regular R-tree. However time slice queries and interval queries may span the whole tree.
- *3D R-tree* [13]: The 3D R-tree treats time as yet another dimension in addition to the spatial dimensions. The main idea is to avoid discrimination between spatial and temporal queries. The 3D R-Tree supports both the temporal and spatial queries, although with performance drawbacks. A main drawback is that timeslice queries are no longer dependent on the live entries at the query time, but on the total number of entries in the history.
- *STR-tree* [9]: The STR-Tree is an extension of the R-Tree, with a different insert/split algorithm. Leaf nodes has the form (id, t_{id}, MBR, o) where t_{id} is the trajectory identifier and o is the orientation of this trajectory in the MBR. The main idea is to keep spatial closeness and partial trajectory preservation by trying to keep line segments belonging to the same trajectory together while keeping spatial closeness as the R-Tree. A parameter p is introduced to balance between spatial properties and trajectory preservation. p indicates the number of levels reserved for trajectory preservation. When inserting a new line segment the goal is to insert it as close as possible to its predecessor in the trajectory within p levels. A smaller p decreases the trajectory preservation, while increasing the spatial closeness.

2. Overlapping Two-Dimensional Structures

This category separates between the spatial and temporal dimensions. The main idea is to use a separate spatial index for each time instance. Then, a temporal index is used to index the spatial indexes. To reduce the storage overhead, consecutive spatial indexes may overlap to avoid storing multiple instances of objects that are not frequently changed over time. Examples of this category include:

- *MR-tree* [15]: The MR-tree employs the idea of overlapping B-trees [2] in the context of the R-tree. The main idea is to avoid the storage overhead of having separate R-trees for each timestamp. The saving in storage is achieved by not storing the common objects among consecutive R-trees. Instead, links from different roots point to the same nodes where all the node entries keep their values over the different timestamps. This idea is perfect in the case of a time slice query. The search is directed to the appropriate root, and then a spatial search is performed using the R-tree. However, the performance of time window queries is not efficient. Also, one major drawback is that many entries can be replicated. Consider the case that only one node entry is changed over two consecutive timestamps, then all other node entries need to be replicated in two consecutive R-trees.

- *HR-tree* [8]: The Historical R-tree (HR-tree) is very similar to the MR-tree. The HR-tree has a concrete algorithm and implementation details of using the overlapping B-tree [2] in the context of the R-tree. The same idea of overlapping trees is applied in the context of quadtrees, where it results in *overlapping quadtrees* [14].
- *HR+-tree* [11]: The HR+-tree is designed mainly to avoid the replication of some entries in the HR-tree. The main reason for having duplicate entries in the HR-tree is that the HR-tree has a condition that any node can contain only entries that belong to the same root, i.e., ones that have the same timestamp. The HR+-tree relaxes this condition by allowing entries from different timestamps to reside in the same node. However, the parent of this node in each R-tree has only access to the entries that belong to the parent's timestamp. In other words, a node may have multiple parents, where each parent has access only to a different part of the node.
- *MV3R-tree* [12]: The MV3R-tree is based mainly on the multi-version B-tree (MVB-tree) [11]. The main idea is to build two trees, an MVR-tree to process timestamp queries, and a 3D R-tree to process long interval queries. Short interval queries are optimized to check which tree is to be used based on a threshold value.

3. Trajectory Indexing

This category is radically different from other categories where the main concern is to support trajectory-oriented queries. On the other side, spatial queries are not well supported. Examples of this category include:

- *TB-tree* [9]: The Trajectory-bundle tree (TB-tree) is an R-tree-like structure that strictly preserves trajectories. A leaf node can only contain segments belonging to the same trajectory. As a drawback, line segments of different trajectories that lie spatially close will be stored in different nodes. The TB-tree grows from left to right. The left-most leaf node is the first inserted node and the right-most leaf node is the last inserted one. The TB-tree is an extension of the STR-tree to handle only trajectories.
- *SETI* [3]: The Scalable and Efficient Trajectory Index (SETI) partitions the spatial dimension into static, non-overlapping partitions. The main observation is that the change of the spatial dimension is limited while the temporal dimension is continuously evolving. Thus, the spatial dimensions are partitioned statically. Within each partition the trajectory segments are indexed using an R-tree. Using a good partitioning function results in having line segments of the same trajectory stored in the same partition. Thus, trajectory preservation is achieved by minimizing the effect of the spatial dimensions in the R-tree. A segment that crosses the boundary of two spatial partitions is clipped and is stored twice in both partitions. This may lead to duplicates in the query result.
- *The SEB-tree* [10]: The Start/End timestamp B-tree (SEB-tree) has an idea similar to SETI, where the space is partitioned into zones that may be overlapped. Each zone is indexed using the SEB-tree that considers only the start and end timestamps of the moving objects. Each moving object is hashed to its zone. A key difference over SETI is that there are no trajectories. Instead only two-dimensional points are indexed. By having the spatial zoning partitioning, two-dimensional points that belong to similar trajectories are kept together.

Key Applications

Moving Object Databases

The wide spread of location-detection devices (e.g., GPS-like devices and cellular phones) along with the recent advances in mobile computing enable the so-called location-based environments. In such environments, a large number of moving objects continuously send their location information to a location-based database server. Storing and indexing past location information enable new types of queries that include: "*What are the vehicles near my shop yesterday between 7:00 and 8:00* A.M. *yesterday*" and "*At what time yesterday, my car was within one mile of a fast food restaurant.*" Indexing historical spatio-temporal data is a major module for efficient query retrieval for moving object databases.

Recommended Reading

1. Becker B., Gschwind S., Ohler T., Seeger B., and Widmayer P. An asymptotically optimal multiversion B-tree. VLDB J. 5(4):264–275, 1996.
2. Burton F.W., Kollias J.G., Matsakis D.G., and Kollias V.G. Implementation of overlapping B-trees for time and space

efficient representation of collections of similar files. The Computer Journal, 33(3):279–280, 1990.
3. Chakka V.P., Everspaugh A., and Patel J.M. Indexing large trajectory data sets with SETI. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
4. Guttman A. R-Trees: A dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
5. Hadjieleftheriou M., Kollios G., Tsotras V.J., and Gunopulos D. Efficient indexing of spatiotemporal objects. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 251–268.
6. Kollios G., Tsotras V.J., Gunopulos D., Delis A., and Hadjieleftheriou M. Indexing animated objects using spatiotemporal access methods. IEEE Trans. Knowledge and Data Eng., 13(5):758–777, 2001.
7. Lomet D.B. and Salzberg B. Access methods for multiversion data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989, pp. 315–324.
8. Nascimento M.A. and Silva J.R.O. Towards historical R-trees. In Proc. 1998 ACM Symp. on Applied Computing, 1998, pp. 235–240.
9. Pfoser D., Jensen C.S., and Theodoridis Y. Novel approaches in query processing for moving object trajectories. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 395–406.
10. Song Z. and Roussopoulos N. SEB-tree: an approach to index continuously moving objects. In Proc. 4th Int. Conf. on Mobile Data Management, 2003, pp. 340–344.
11. Tao Y. and Papadias D. Efficient historical R-trees. In Proc. 13th Int. Conf. on Scientific and Statistical Database Management, 2001, pp. 223–232.
12. Tao Y. and Papadias D. MV3R-Tree: a spatio-temporal access method for timestamp and interval queries. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 431–440.
13. Theodoridis Y., Vazirgiannis M., and Sellis T. Spatio-temporal indexing for large multimedia applications. In Proc. Int. Conf. on Multimedia Computing and Systems, 1996, pp. 441–448.
14. Tzouramanis T., Vassilakopoulos M., and Manolopoulos Y. Overlapping linear quadtrees: a spatio-temporal access method. In Proc. 6th Int. Symp. on Advances in Geographic Inf. Syst., 1998, pp. 1–7.
15. Xu X., Han J., and Lu W. RT-Tree: An improved R-tree indexing structure for temporal spatial databases. In Proc. Int. Symp. on Spatial Data Handling, 1990, pp. 1040–1049.

Indexing Metric Spaces

Pavel Zezula, Michal Batko, Vlastislav Dohnal
Masaryk University, Brno, Czech Republic

Synonyms

Distance indexing

Definition

Metric space indexing is closely related to the recent digitization revolution where almost everything that one can see, hear, read, write or measure is available in digital form. Unlike traditional attribute-like data types such as numbers and strings of sortable domains, instances of these new data types are complex, and the only measure of comparison to apply is a sort of *similarity*. Such a situation implies an application of the *query-by-example* search paradigm where the database is searched for objects that are *near* the example object, also called the *query object*. A useful abstraction of this similarity is to see it as mathematical *metric space* [7]. The problem of organizing and searching large datasets of complex objects can then be considered from the perspective of generic or arbitrary metric spaces, sometimes labeled *distance spaces*. In general, the search problem can be described as follows:

- *Let D be a domain, d a distance measure on D, and (D,d) a metric space. Given a set $X \subseteq D$ of n elements, preprocess or structure the data so that similarity queries are answered efficiently.*

From a practical point of view, *X* can be seen as a file (a dataset or a collection) of objects that takes values from domain *D*, with *d* as the proximity measure, i.e., the distance function defined for an arbitrary pair of objects from *D*. Though several types of similarity queries exist and others are expected to appear in the future, the basic types are known as the *similarity range* query whose results are constrained by a maximum distance expressed as the query radius, and the *nearest neighbor(s)* query bounding the query response size by the maximal number of closest objects $k \leq n$.

The metric space indexing approach significantly extends the scope of traditional search approaches. It supports the execution of similarity queries without obviating the traditional attribute-like searching. The distance-searching approach to indexing is thereby highly *extensible*.

Historical Background

The need for similarity searching in collections of metric objects was recognized quite early and the history documented by numerous citations is nicely summarized in recent surveys such as [2, 6, 8, 10]. The first known proposal was the *Burhard-Keller tree* from 1973, which was later extended into the *Fixed queries tree* and the *Fixed queries array* techniques, all of which specialize in discrete distance functions only. By recursive applications of the basic *ball* and *generalized-hyperplane* partitioning principles, the *Vantage point tree* and the *Generalized-hyperplane tree* were defined. The advantage of exploiting pre-computed distances to speed up retrieval was first recognized in the 1980's and formalized as the *Approximating and Eliminating Search Algorithm*, AESA. However, all of these early attempts were only considering small data collections stored in the main memory.

The importance of processing large datasets stored on disk memory was reflected for the first time in the M-tree [3]. It is a balanced metric tree supporting a disk-oriented storage, trying to optimize the I/O as well as the CPU costs of query processing by means of a synergistic combination of several elementary strategies. It can be considered as a generalization of the R-tree and the B-tree. The M-tree's success has been demonstrated by numerous modifications and extensions concerning mainly insertion algorithms, mechanisms of splitting, and strategies for query execution. The most significant extension is the Slim-Tree [9]. As an orthogonal approach to tree partitioning, the D-index [4] is constructed as a multi-tier hashing structure consisting of search-separable sets of objects on each tier organized in directly-accessible buckets. The structure supports easy insertion and its search costs are bounded because no more than one bucket needs to be accessed at each level for range queries up to a pre-defined value of the search radius. Other important hybrid approaches are known as the *Multi vantage point tree*, the *Geometric near-neighbor access tree*, and the *Spatial approximation tree*.

The similarity search in metric spaces is generally expensive and state-of-the-art access methods designed for a single computer do not provide sufficient performance for highly interactive applications that access large collections of data. The *approximate similarity search* techniques offer greatly improved efficiency vis à vis precise similarity searching at the expense of some imprecision in the results. The use of the approximate similarity search is mainly justified by the following two observations: (i) the similarity between objects is often subjective, thus very difficult to express as a unique rigorous function; (ii) similarity search processes are intrinsically iterative – users typically issue several similarity queries to the search system, possibly reusing the previous query results to express new ones. Practical experiments with approximate similarity search techniques register performance improvements up to two orders of magnitude compared to the precise evaluation of the same queries. Other techniques overcome the *scalability* problem by distributing partitions of data and by executing similarity queries in parallel using structured peer-to-peer networks. Four such peer-to-peer techniques are compared in [1].

Foundations

In the last decade, metric space indexing for similarity search has prompted major research efforts resulting in a number of specific theories, techniques, implementation paradigms and analytic tools aimed at making the distance-based approach viable. There are basically two ways to solve the indexing problem: (i) to transform the metric space so that a working solution from another domain can be applied; or (ii) exploit the general properties of metric functions directly. This entry focuses on native indexing.

Due to the lack of a global ordering or a fixed position as in coordinate spaces, elementary *partitioning* in generic metric spaces is defined with respect to either one or two fixed reference objects taken from *D*. The first principle, known as the *ball partitioning*, uses one given object as a center and a specific radius as a boundary to form a sphere (ball). The set of data is then divided into two disjoint subsets: objects inside the ball and those outside the ball. The other principle partitions the space using two given reference objects where the distance establishes the borderline of the *generalized hyperplane partitioning* – objects that are closer to the first reference object than to the second form one partition, while the rest of the objects belong to the second partition.

In addition to the partitioning principles, strategies for query execution play another important role in search structures because they can significantly influence the efficiency of answering queries. Efficient search algorithms for similarity range and nearest neighbor queries have been defined for metric indexes, based on the *branch and bound* strategy in principle. There are even algorithms for evaluating *incremental nearest neighbor* queries, as well as *similarity joins*. Approximate similarity searches have also been studied thoroughly in order to improve performance. The general idea of approximation algorithms is to relax some constraints of the "precise" similarity search to reduce search costs, as measured in disk accesses and/or the number of distance computations. This inevitably means that *false hits* or *false dismissals* may occur.

Since the computational complexity of some metric distance functions can be quite high, it is very important for metric indexes to limit the number of distance computations as much as possible. The rationale behind such strategies is to exploit already-evaluated distances between some objects while properly applying the metric space postulates – namely the *triangle inequality*, *symmetry*, and *non-negativity* – to determine bounds on distances between other objects. The most elementary case can be explained on three objects a,b, and $c \in D$. Provided the distances $d(a, b)$ and $d(b, c)$ are known, it is clear that the distance between a and c cannot be longer than their sum, i.e., $d(a, c) \leq d(a, b) + d(b, c)$. Alternatively, if the known distances are $d(a, c)$ and $d(b, c)$, the distance $d(a,b)$ must be at least as long as their difference, that is $d(a, b) \geq |d(a, c) - d(b, c)|$. Several *bounding strategies* originally proposed in [5] are summarized in [6]. These techniques represent the foundations of the general pruning rules that are employed, in a specific form, in practically all index structures for metric spaces. They can be considered as the basic formal background of metric indexes.

Key Applications

Treating data collections as metric objects is advantageous in that many data classes and information-seeking strategies conform to the metric view. Accordingly, a single metric indexing technique can be applied to many specific search problems quite different in nature. In this way, the important *extensibility* property of indexing structures is satisfied by default. An indexing scheme that allows various forms of queries, or which can be modified to provide additional functionality, is of more value than an indexing scheme otherwise equivalent in power or even better in certain respects, but which cannot be extended.

Distance functions of metric spaces represent a way of quantifying the closeness of objects in a given domain. The distance functions are often tailored to specific applications or a class of possible applications. In practice, the distance functions are specified by domain experts, however, no distance function restricts the range of query types that can be asked with this metric. The distance functions can be *discrete* or *continuous*. Another classification is possible according to the type of domain of compared objects. For example, the *Minkowski distance* functions form a whole family of metric functions, designated as the L_p metrics, because the individual cases depend on the numeric parameter p. These functions are defined on n-dimensional vectors, where the L_1 metric is known as the *Manhattan distance* (also the *City-Block distance*), the L_2 distance denotes the well-known *Euclidean distance*, and the L_∞ is called the *maximum distance*, the *infinite distance* or the *chessboard distance*. The closeness of sequences of symbols (strings) can be effectively measured by the *edit distance*, also called the *Levenshtein distance*. In order to identify common molecular subsequences, the Smith-Waterman algorithm has been proposed. In case the processed objects are sets, the *Jaccard's coefficient* or the *Hausdorff distance* can be applied. More details about metric functions can be found in [5], but the set of metric distance functions is still growing, as evidenced by the recent *Earth Mover's Distance*, for example.

Experiments show that metric indexing techniques are very competitive even in specific data domains traditionally supported by specialized index structures, the R-tree for *multi-dimensional data* and the B-tree for one-dimensional *attribute-like data* being two such examples. The number and variety of metric distance functions determine the range of applications which can apply the metric indexing approach. It has proved useful for *multimedia data features* since most of the standard MPEG7 image descriptors are metrics. But the application of metric indexing involves a lot of diverse fields such as *spatial databases*, *computer graphics and vision*, *game programming*, *geographic information systems*, *computational geometry*, *computer-aided design*, *robotics*, *computational biology* and many others.

Future Directions

The biggest challenge of a perspective search paradigm is to find self-organized solutions that evolve in time and still scale into the expected data volumes. In general, the self-organizing systems build and maintain an internal knowledgebase in response to the flow of data and queries. Such initiative must be based on solid theoretical backgrounds to avoid possible quick but ad-hoc solutions which will sooner or later fail due to the absence of rigorous definitions and unpredictable behavior. The research needs to go beyond the capabilities of traditional computer science and should try to find an inspiration in other scientific areas. The social sciences offer a promising alternative, especially advances in online social networking.

Experimental Results

Experimental results demonstrate the extensibility of the metric space approach since most of the proposed indexing tools work for any metric distance function. The performance is primarily influenced by the distribution of distances, the smaller the variation of distances, the more expensive the search becomes. At the same time, the scalability of expensive queries is practically linear [8]. In order to deal with the huge volumes of data required by current applications, distributed architectures with P2P navigation [9] seem to offer a solution. Given enough resources, P2P networks maintain almost constant response-time while scaling to data volumes larger by several orders of magnitude.

Data Sets

Several influential research groups are currently collecting large repositories to test the scalability of their indexing tools. One of the most significant and publicly available web pages is the UCI Machine Learning Repository http://mlearn.ics.uci.edu/MLRepository.html with nearly 200 various datasets.

URL To Code

http://www-db.deis.unibo.it/research/Mtree/
: the code of the original M-tree and many references to the related literature

http://lsd.fi.muni.cz/trac/mtree/
: an improved version of the M-tree

http://gbdi.icmc.usp.br/arboretum/
: several implementations of metric indexing structures

http://lsd.fi.muni.cz/trac/messif
: the Metric Similarity Implementation Framework, MESSIF, can be used to implement centralized and distributed similarity search indexing structure prototypes.

Cross-references

- ▶ Approximate Query Processing
- ▶ Closest-Pair Query
- ▶ Data Cleaning
- ▶ Digital Libraries
- ▶ Indexing and Similarity Search
- ▶ Metric Space
- ▶ Multimedia Data Indexing
- ▶ Multimedia Data Querying
- ▶ Nearest Neighbor Query
- ▶ Peer-to-Peer System
- ▶ Spatial Indexing Techniques
- ▶ Text Indexing Techniques

Recommended Reading

1. Batko M., Novak D., Falchi F., and Zezula P. On Scalability of the Similarity Search in the World of Peers. In Proc. 1st Int. Conf. Scalable Information Systems, 2006, pp. 1–12.
2. Chávez E., Navarro G., Baeza-Yates R., and Marroquín J.L. Searching in metric spaces. ACM Comput. Surv., 33 (3):273–321, 2001.
3. Ciaccia P., Patella M., and Zezula P. M-tree: An efficient access method for similarity search in metric spaces. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 426–435.
4. Dohnal V., Gennaro C., Savino P., and Zezula P. D-Index: Distance searching index for metric data sets. Multimedia Tools Appl., 21(1):9–33, 2003.
5. Hjaltason G.R. and Samet H. Incremental similarity search in multimedia databases. Technical Report CS-TR-4199, Computer Science Department, University of Maryland, College Park., November 2000.
6. Hjaltason G.R. and Samet H. Index-driven similarity search in metric spaces. ACM Trans. Database Syst., 28(4):517–580, 2003.
7. Kelly J.L. General Topology. D. Van Nostrand, New York, 1955.
8. Samet H. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, San Francisco, CA, USA, 2005.
9. Traina Jr. Traina A.J.M., Seeger B., and Faloutsos C. Slim-Trees: High Performance Metric Trees Minimizing Overlap between Nodes. In Advances in Database Technology, Proc. 7th Int. Conf. on Extending Database Technology, 2000, pp. 51–65.
10. Zezula P., Amato G., Dohnal V., and Batko M. Similarity Search: The Metric Space Approach, Springer, Berlin Heidelberg, New York, 2006.

Indexing of Data Warehouses

THEODORE JOHNSON
AT&T Labs Research, Florham Park, NJ, USA

Synonyms

Data warehouse indexing

Definition

Indices are data structures especially designed to allow rapid access to data in large databases. Data warehouses are typically used to perform intensive analyses of very large data sets. Several indices, such as projection indices, bitmap indices, bitslice indices, and summary indices, have been developed to address the special needs of data warehousing, and are presented in this entry.

Historical Background

Data warehouses were developed to capture operational information, store it over a long period, and provide support for intensive analysis of historical data. The special needs of data warehouses – very large data sets, high dimensional data, and the extensive use of categorical data – has led to the development of specialized indices intended for use in data warehouses. The use of these indices was pioneered in such systems as Model 204 and Sybase IQ.

Foundations

Indices, such as *B-trees*, *R-trees*, *quad-trees*, *hash tables*, and *inverted indices*, are integral parts of any database system. Query processing in data warehouses place special demands on the indices: very large data sets, high dimensional data, large materialized aggregate views, and categorical data. Several special techniques have been developed in response.

Projection Indices: Data warehouses are often used to store one or more high dimensional *fact tables*, which contain historical records of past events. For example, a SALES fact table might have fields which describe the customer (CustomerID, FirstName, LastName), the product sold (ProductLine, Brand, Size, Color), the method of sale (Store, Referrer, SalesPerson), and values (i.e., *measures*) which quantify the sale (Price, Discount, Quantity).

The large number of dimensions, especially those involving categorical values with a small to moderate range (e.g., Store), or with redundant information (e.g., CustomerID determines FirstName and LastName) can cause an unnormalized fact table to require an excessive amount of storage. *Star schemas* and *snowflake schemas* are common techniques to reduce fact table storage costs. Another useful technique is *vertical partitioning*. Groups of columns, or even individual columns, are stored independently but related via their record ID, or *rid*. Columns with a low cardinality and/or many null values can use special encoding schemes. Sybase IQ made prominent use of this technique.

An advantage of vertical partitioning is that many queries reference only a few of the fields of the fact table. For example, a query might be, "What is the volume of sales which involve a Discount larger than 50%, broken down by Store." If SALES is vertically partitioned, an efficient query plan is to scan the Discount and Store fields. Whenever the Discount field is 50% or larger, retrieve the linked Store field and increment a corresponding counter.

In the above example, the Discount field acts as a *projection index* [9]. Access to the field is fast because of the compact vertically partitioned storage. The set of records which satisfy the "Discount $\geq$ 50%" might be dense (more than 1 in 100), making random access indices inefficient. Furthermore, complex predicates involving several fields can be processed in the same way, providing highly efficient multi-dimensional indices.

Bitmap Indices: Bitmap indices are a logical extension of projection indices [9]. Suppose that the predicate of the query is, "Store = 'Foo' and Brand = 'Bar''. Suppose that both individual predicates is true for about 1 in 100 records – dense enough that linear scanning is effective. However their conjunct is likely to be true for about 1 in 10,000 records – sparse enough that random access is preferable. Therefore an efficient query plan is to:

- Scan the Store vertical partition for values of Foo, and record the corresponding rids
- Scan the Brand vertical partition for values of Bar, and record the corresponding rids
- Compute the intersection of the two rid lists, and fetch the corresponding records

An efficient way to store the rids and then compute their intersection is to use a *bitmap*. Each bit corresponds to a record – the bit at position 0 refers to record 0, the bit at position 1215 refers to record 1215, and so on. A bitmap can be stored as a packed array of bits, e.g., 64 bits per word. Given two bitmaps, Boolean functions can be computed by taking the corresponding Boolean function of the corresponding packed bit array, which are typically very fast CPU register operations.

Store = 'Foo Brand = 'Bar'

$$\boxed{\begin{array}{c}1\\0\\0\\1\\1\\0\\0\\0\\0\\0\\1\\1\\0\\1\end{array}}\ \text{AND}\ \boxed{\begin{array}{c}1\\0\\1\\0\\1\\1\\1\\0\\1\\0\\0\\1\\0\\0\end{array}} = \boxed{\begin{array}{c}1\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\1\\0\\0\end{array}}$$

Given a set of bitmaps, complex Boolean predicates can be evaluated very efficiently. Therefore, a natural extension of projection indices are *bitmap indices*: for each value of a field, create and store a bitmap. When a predicate is to be evaluated, the bitmaps can be fetched directly, bypassing the cost of computing them. Bitmap indices are very effective for fields with low cardinality, or for representing pre-computed predicates. However, bitmap indices require a large amount of storage if the field has more than a few distinct values.

Compressed Bitmap Indices: If a field has a large number of distinct values, the bitmaps for most of the values must be sparse, and therefore compressible. A variety of techniques based on *run-length encoding* were developed in the context of image compression, but are also usable for bitmap index compression. While the RLE-based techniques can achieve near-optimal compression, the compression and decompression can be slow because compression is achieved by compressing run length codes and packing them into a bit array.

A bitmap compression technique better suited for compressing bitmap indices are the *Byte-aligned Bitmap Codes*, or BBC codes [1]. Each code word consists of one or mote bytes, eliminating the need for bit extraction. Furthermore, Boolean operations can be performed on the BBC codes directly, creating BBC code output. However, modern processors operate on multi-byte words, and data elements such as integers must be word-aligned before they can be operated upon. *Word-aligned* codes [11] extend the BBC code idea, and achieve even greater performance by avoiding word alignment costs.

The penalty for using byte-aligned or word-aligned compression codes is that they are less space efficient than the best RLE codes. A bitmap index might therefore be stored using a variety of codecs, depending on the individual bitmap properties. Furthermore, evaluating a complex predicate requires an evaluation plan, and there are a very large number of evaluation plans for a large and complex predicate. Fortunately, there is a linear-time dynamic programming optimization algorithm for choosing an optimal evaluation plan. By using BBC codes, word-aligned codes, and optimized evaluation, a bitmap index can be stored in space similar to that of a conventional index, and yet evaluate complex predicates, including large range queries, very efficiently [1,11].

Hybrid B-tree/Bitmap Indices: If a bitmap index is used for a high cardinality field, the field will have a large number of bitmap indices, and finding the correct bitmaps to use in evaluating a predicate becomes a search problem. Conversely, a B-tree index for a non-key field will in general need to store a set of records for each indexed value. A natural solution to both issues is to record the rid set of a value using a compressed bitmap. Oracle uses this technique, with BBC codes used to compress the bitmap [9].

Bitslice Indices: If the indexed field has a very high cardinality (e.g., ranges over the integers), bitmap indices lose their value. However, one can still make use of bitmap indices by using bitmaps to index value ranges. For example, one can partition the range of an integer or floating point field into, say, 1,000 ranges, then create a bitmap for each range. Range queries are likely to be efficient, but point queries might not be as selective as desired. By a judicious choice of overlapping value ranges, one can obtain more value from each bitmap. For example, a *bitslice index* [8] applied on an integer value creates a bitmap for each bit position of the value. So, for example, the value 13 has set bits in the bit-0, bit-2, and bit-3 bitmaps, reset bits in the bit-1, bit-4, bit-5, etc. bitmaps. Point queries are efficiently supported with a bitslice index; but perhaps surprisingly, one can write range queries into compact predicates on bitslice indices (see below an example of bitslice indices on an integer field).

(193)	1	1	0	0	0	0	0	0
(87)	0	1	0	1	0	0	0	0
(225)	1	1	1	0	0	0	0	1
(7)	0	0	0	0	0	1	1	1

Aggregate-Storing Indices: Data warehouses very often support aggregation queries to summarize the mass of data in the fact tables. A common type of query is to ask for the value of an aggregate within a particular range of field value(s). A hierarchical index structure, such as a B-tree, summarizes the data at every level of its hierarchy. For example, a B-tree node has a field value range, and points (directly or indirectly) to the collection of records which have a value in that range for the indexed field.

One can change the definition of "summarize" to mean storing an aggregate value. In a leaf node entry for a value *v*, instead of pointing to the records which have value *v* for the indexed field, the entry contains an aggregate value, e.g., the sum of a measure field, over all records with value *v* in their indexed field. An entry in an interior node (above the leaf level) contains the sum of the aggregates in the child, as well as a pointer to the child and its indexed value range. Finding the aggregate value of the range can be done by walking the tree, once down to the minimum value of the range, and once to its maximum value.

The principle of the aggregate-summarizing B-tree can be extended to handle more useful situations, including aggregate-summarization of multiple dimensions [4,6], and summarization of intervals (useful for temporal database systems) [5].

Summary Indices: A common technique for managing a very large data warehouse is *horizontal partitioning* – partitioning records by one or more predicates. A data warehouse often receives regular updates of new data into its fact table(s), and stores a window (e.g., 1 year) of the most recent data. Fact tables may also be partitioned by other attributes, to narrow search spaces, help ensure that partitions are dense, etc.

If a fact table is sliced into a large number of vertical partitions, the partition predicates act as another type of index. This idea can be extended to recording properties of the data in a partition. For example, a summary index might record aggregates of the data, such as minimum and maximum timestamps [7]. Alternatively, a summary index can record whether or not particular field values exist in the partition [3]. For example, the index can record a list of all customer IDs that appear in a partition (or conversely in which partitions a customer identifier appears). If the average customer ID appears in only a few partition, this type of index minimized the number of partition indices to be searched.

Join and Star Indices

A *join index* is a collection of pairs $\{(r,s)\}$ such that the record in table R with record ID (RID) r joins with the record in table S with RID s, according to the join predicate which defines the index. A join index accelerates the processing of joins, a common activity in a data warehouse with a star or snowflake schema. A *star index* is a join index between a fact table and each of its dimension tables.

Key Applications

The most commonly implemented of these indices is the bitmap index, which has been implemented in most of the major commercial databases used for data warehousing. Bitmap indices are also commonly used in scientific database applications, such as High Energy Physics. In HEP applications, bitmaps indices are often used to record "interest sets" – collections of events found to satisfy some complex property – to accelerate their retrieval for later processing [10].

Cross-references

- ▶ Bitmap-based Index Structures
- ▶ Bitmap Index
- ▶ Bitslice Signature Files
- ▶ B+-Tree
- ▶ Inverted Index
- ▶ Join Index
- ▶ Measure
- ▶ Quadtrees (and Family)
- ▶ R-Tree (and family)
- ▶ Snowflake Schema
- ▶ Star Schema

Recommended Reading

1. Amer-Yahia S. and Johnson T. Optimizing queries on compressed bitmaps. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 329–338.
2. Apaydin T., Canahuate G., Ferhatosmanoglu H., and Tosun A. Approximate encoding for direct access and query processing over compressed bitmaps. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 846–857.
3. Johnson T. Coarse indices for a tape-based data warehouse. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 231–240.
4. Johnson T. and Shasha D. Some approaches to index design for cube forest. IEEE Data Eng. Bull., 20(1):27–35, 1997.
5. Kline N. and Snodgrass R. Computing temporal aggregates. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 222–231.
6. Kotidis Y. and Roussopoulos N. An alternative storage organization for ROLAP aggregate views based on cubetrees. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 249–258.
7. Moerkotte G. Small materialized aggregates: a light weight index structure for data warehousing. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 476–487.
8. O'Neil P. and Quass D. Improved query performance with variant indices. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 38–49.
9. Rdb7: Performance enhancements for 32 and 64 bit systems. Available online at: http://www.oracle.com/products/servers/rdb/html/fsvlm.html.
10. Wu K., Koegler W.S., Chen J., and Shoshani A. Using bitmap index for interactive exploration of large datasets. In Proc. 15th Int. Conf. on Scientific and Statistical Database Management, 2003, pp. 65–74.
11. Wu K., Otoo E.J., and Shoshani A. Compressing bitmap indexes for faster search operations. In Proc. 14th Int. Conf. on Scientific and Statistical Database Management, 2002, pp. 99–108.

Indexing of the Current and Near-Future Positions of Moving Objects

Simonas Šaltenis, Christian S. Jensen
Aalborg University, Aalborg, Denmark

Definition

A scenario is assumed where a large population of objects capable of reporting positional data to a central server exists. Specifically, an object may report its current position, but may also report its current velocity vector.

A key challenge is to be able to accommodate the very frequent updates inherent to this scenario. Another important challenge is to contend with, and indeed exploit, all the available positional data so that better query results to near-future queries are enabled.

To address these challenges, the position of an object is typically modeled as a linear function from time to space (typically the two- or three-dimensional Euclidean spaces are assumed). Such functions are readily available, and the positions they return get outdated less frequently than do constant functions, thus reducing the update rate. Further, they enable the computation of more accurate results for near-future queries.

The fundamental types of queries to be supported by an index include ones that retrieve the objects with a position within a specified, rectangular spatial region: *timeslice* and *window* queries consider a stationary region for a single time point or a time interval; and so-called *moving queries* attach a rectangle to a moving object and consider a time interval. The times supported range from the current time to some near-future time.

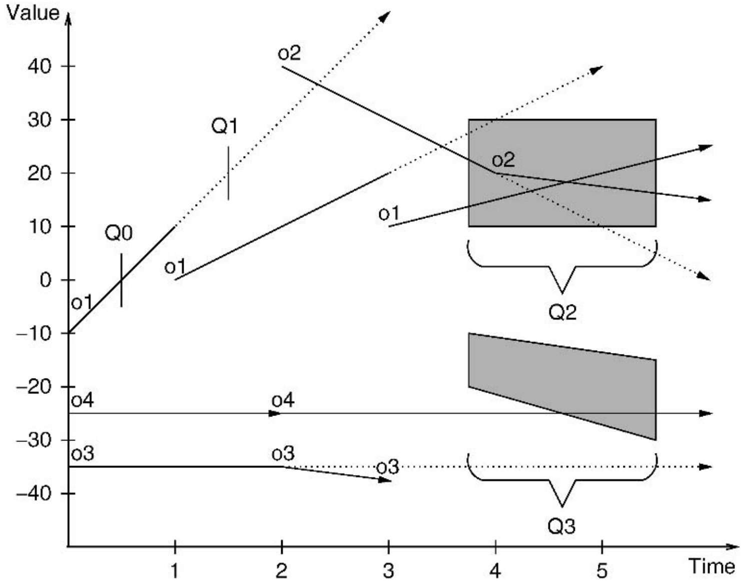
Figure 1 shows a set of trajectories in (x, t)-space of objects and illustrates the three types of queries: $Q0$ and $Q1$ are timeslice queries, $Q2$ is a window query, and $Q3$ is a moving query.

Other types of queries such as nearest neighbor queries or reverse nearest neighbor queries may also be supported. In addition, an index should also support updates, i.e., insertions and deletions.

For its illustration of key concepts, this entry uses primarily R-tree-based techniques; specifically, the TPR-tree [12] is covered in some detail, as it is the ancestor of a sizable family of indices.

Historical Background

The first proposal for the indexing of moving object positions as linear functions is due to Tayeb et al. [14] who propose to use PMR-quadtrees for indexing the future linear trajectories of one-dimensional moving point objects as line segments in (x, t)-space. Kollios et al. [6] also focus mostly on one-dimensional data



Indexing of the Current and Near-Future Positions of Moving Objects. Figure 1. Query examples for one-dimensional data [12].

and employ the so-called *duality* data transformation where a line $x = x(t_{ref}) + v(t - t_{ref})$ is transformed to the point $(x(t_{ref}),v)$, enabling the use of regular spatial indices.

Later research focuses on two-dimensional and, in some cases, three-dimensional data. In general, the proposed indexing methods can be classified according to the space that they index, i.e., what view is taken on the indexed data. Assume the objects move in d-dimensional space (d = 1,2,3). The first approach is to index future trajectories as lines in $(d + 1)$-dimensional space. This is the approach taken by Tayeb et al. [14], but the approach is difficult to extend to higher dimensions, and the index has to be rebuilt periodically.

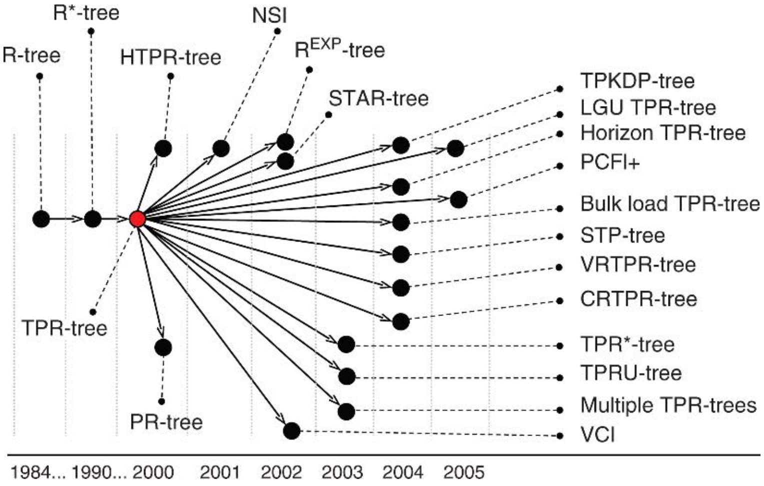
The second approach is to transform the trajectories to points in a higher-dimensional space which are then indexed. Queries are then also transformed to counter the data transformation. The transformation of Kollios et al. [6] maps linear functions in d-dimensional space into static points in $2d$-dimensional space.

In STRIPES [8], PR quadtrees are used to index these $2d$-dimensional points. Yiu et al. [15] instead propose to use space filling curves to further transform $2d$-dimensional points into one-dimensional points that are then indexed by the B^+-tree. Agarwal et al. [1] combine the duality transformation with kinetic data structures [2]. The main idea of kinetic data structures is to schedule future events that update a data structure so that necessary invariants hold.

The third approach, sometimes referred to as indexing in the *primal* space, is to index the data in their native, d-dimensional space, which is possible by parameterizing the index structure using velocity vectors and thus enabling the index to be "viewed" as of any future time. This absence of transformations yields quite intuitive indexing techniques.

The Time-Parameterized R-tree (TPR-tree) [12] exemplifies this approach. Proposed by Šaltenis et al. in 2000, the TPR-tree gave rise to a number of other access methods, as illustrated in Fig. 2. For example, the R^{EXP}-tree [11] extends the TPR-tree to index data with expiration times, so that the positions of the objects that do not update their positions are automatically removed from the index. The TPR*-tree [13] introduces new heuristics with the objective of improving the query performance of the TPR-tree and to optimize it for workloads of queries that differ slightly from those targeted by the TPR-tree. The STAR-tree [10] modifies the TPR-tree by introducing more complex time-parameterized bounding rectangles and making the index self-adjustable. The Velocity Constrained Indexing (VCI) [9] technique uses the regular R-tree with an additional field of v_{max} in each node. The v_{max} is used to expand the bounding rectangles of the R-tree when future queries are processed.

The fourth approach is to index the objects' positions as of some specific time points, so-called label timestamps, and to extend the spatial extents of queries according to the maximum speeds of objects



Indexing of the Current and Near-Future Positions of Moving Objects. Figure 2. TPR-tree origins and follow-up research.

and the difference between the time specified in the query and the label timestamps. The B^x-tree [4,5] uses a combination of space-filling curves and B$^+$-trees to index the (static) positions of objects as of label timestamps.

Foundations

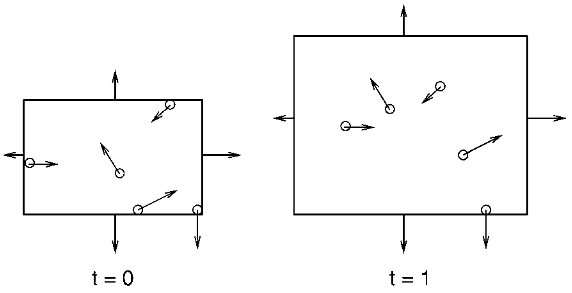
Data and Queries

For an object moving in d-dimensional space, the object's position at some time t is given by $\bar{x}(t) = (x_1(t), x_2(t),\ldots, x_d(t))$, where it is assumed that the times t are not before the current time. This position is modeled as a linear function of time, which is specified by two parameters. The first is a position for the object at some specified time t_{ref}, $\bar{x}(t_{ref})$, which is called the reference position. The second parameter is a velocity vector for the object, $\bar{v} = (v_1, v_2,\ldots, v_d)$. Thus, $\bar{x}(t) = \bar{x}(t_{ref}) + \bar{v}(t - t_{ref})$.

Then, as shown in Fig. 1, a *timeslice* query, $Q = (R, t)$, retrieves points that will be inside the d-dimensional hyper-rectangle R at time t. A *window* query, $Q = (R, t^{\vdash}, t^{\dashv})$, retrieves points that will be inside the hyper-rectangle R sometime during time-interval $[t^{\vdash}, t^{\dashv}]$. A *moving* query, $Q = (R_1, R_2, t^{\vdash}, t^{\dashv})$, retrieves points with trajectories in $(\bar{x}, t)$-space crossing the $(d + 1)$-dimensional trapezoid obtained by connecting R_1 at time $t^{\vdash}$ to R_2 at time $t^{\dashv}$.

The Structure of the TPR-Tree

A large number of indices utilize the structure of the TPR-tree, which indexes moving points in one, two, or three dimensions. It employs the basic structure of the R-tree, which stores data in the leaves of a balanced index tree, and each non-leaf index entry contains a minimum bounding rectangle (MBR) of all the data in the subtree pointed to by the entry. In contrast to the R-tree, the indexed points as well as the bounding rectangles are augmented with velocity vectors. This way, bounding rectangles are time parameterized – they can be computed for different time points. Velocities are associated with the edges of bounding rectangles so that the enclosed moving objects (points or other rectangles) remain inside the bounding rectangles at all times in the future. More specifically, if a number of points p_i are bounded at time t, the spatial and velocity extents of a bounding rectangle along the x axis are computed as follows:



Indexing of the Current and Near-Future Positions of Moving Objects. Figure 3. Example time-parameterized bounding rectangle [3].

$$x^{\vdash}(t) = \min_i\{p_i.x(t)\}; \quad x^{\dashv}(t) = \max_i\{p_i.x(t)\};$$
$$v_x^{\vdash} = \min_i\{p_i.v_x\}; \quad v_x^{\dashv} = \max_i\{p_i.v_x\}.$$

Figure 3 shows an example of the evolution of a bounding rectangle in the TPR-tree computed at $t = 0$. Note that, in contrast to R-trees, bounding rectangles in the TPR-tree are not minimum at all times. In most cases, they are minimum only at the time when they are computed. A process called *tightening*, performed whenever an index node is modified during an insertion or a deletion, recomputes a node's time-parameterized bounding rectangle, rendering it minimum at that time.

Querying the TPR-Tree

The TPR-tree can be interpreted as an R-tree for any specific time, t_q. This suggests that algorithms that are based on the R-tree are easily "portable" to the TPR-tree. For example, answering a timeslice query proceeds as for the R-tree, the only difference being that all bounding rectangles are computed for the time t_q specified in the query before intersection is checked.

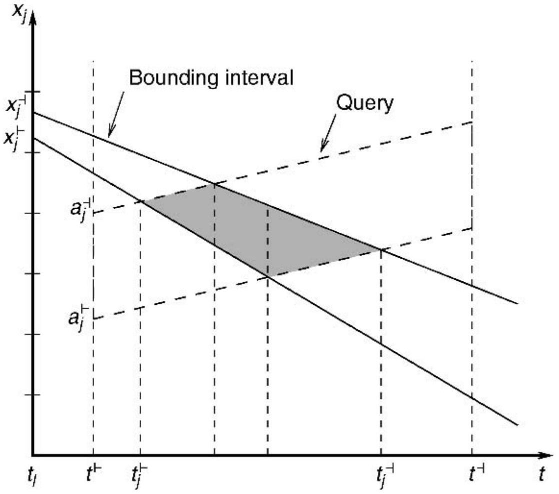
To answer window queries and moving queries, the algorithm has to check if, in $(\bar{x}, t)$-space, the trapezoid of a query intersects with the trapezoid formed by the part of the trajectory of a bounding rectangle that is between the start and end times of the query. This can be checked using a simple algorithm [12]. Figure 4 illustrates the intersection between a one-dimensional time-parameterized bounding rectangle (interval) and a moving query.

Updating the TPR-Tree

An update of the moving object's position is modeled as a deletion of the old position followed by an insertion of the new position. The TPR-tree's update algorithms are based on the update algorithms of the R*-tree, which is an R-tree with improved update algorithms. The update algorithms of the TPR-tree differ from the corresponding algorithms of the R*-tree only in the heuristics that are used. The R*-tree uses heuristics that minimize certain functions, including the area of a bounding rectangle, the intersection of two bounding rectangles, the margin of a bounding rectangle, and the distance between the centers of two bounding rectangles. As the TPR-tree employs time-parameterized bounding rectangles, the above-mentioned functions are time dependent, and their evolution in time should be considered. Specifically, given an objective function $A(t)$, the following integral should be minimized:

$$\int_{t_c}^{t_c+H} A(t)\,dt,$$

where t_c is the time when the heuristics is being applied and H is a so-called *time horizon* parameter, the value of which reflects how far into the future queries are expected to "see" the effects of the application of this heuristics. If $A(t)$ is area (in d-dimensional space), the integral computes the area of the trapezoid that represents part of the trajectory of a bounding rectangle in $(\bar{x}, t)$-space (a d+1-dimensional volume, see also Fig. 4).



Indexing of the Current and Near-Future Positions of Moving Objects. Figure 4. Intersection of a bounding interval and a query [12].

Duality-Transformation Approach

The general approach of indexing moving points in their native space using a time-parameterized index structure such as the TPR-tree is closely related to the duality transformation approach [6, 8, 15].

Considering one-dimensional data, the duality transformation transforms the linear trajectory of a moving point $x = x(t_{ref}) + v(t - t_{ref})$ in (x, t)-space into a point $(x(t_{ref}), v)$, where t_{ref} is a chosen reference time. Queries, then, are also transformed.

Bounding points $(x(t_{ref}), v)$ in the dual space with a minimum bounding rectangle is equivalent to bounding them (as moving points) with a time-parameterized bounding interval computed at t_{ref}. Figure 5 shows the same bounding rectangle and query in $(x(t_{ref}), v)$-space and in (x, t)-space.

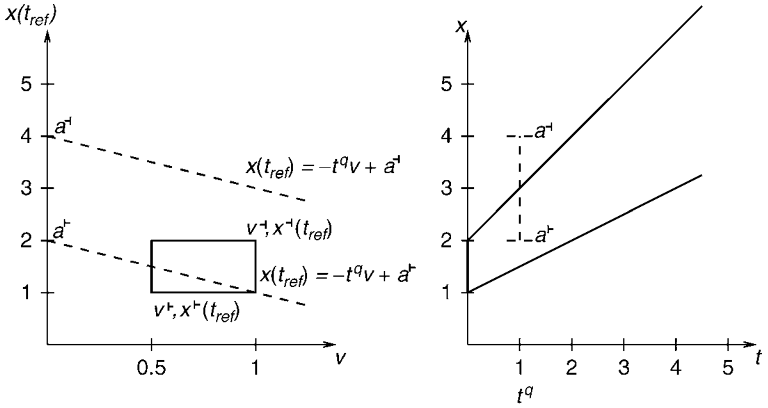
In spite of the equivalence among the bounding rectangles used in the two approaches, the algorithms used in different indexes may vary substantially. A duality-transformation index may not even explicitly use minimum bounding rectangles [15]. Furthermore, while the heuristics of time-parameterized indexes consider the objects' positions at the time the heuristics are applied, the algorithms of duality-transformation approaches always use a pre-chosen constant t_{ref}. For this reason, duality-transformation approaches usually use two (or more) indexes, such that updates are placed into the latest index; and when the earliest index becomes empty, a new index is created [6].

A sufficiently high update rate is crucial in order for both the time-parameterized indexes, such as the TPR-tree, and the indexes using the duality transformation to offer good query performance. The reasons for this are most obvious in the TPR-tree, where query performance degrades due to the uninterrupted expansion of time-parameterized bounding rectangles, which results in more queries intersecting with a given bounding rectangle.

Key Applications

Online, Position-Aware People, Vehicles, and Other Objects

The rapid and continued advances in positioning systems, e.g., GPS, wireless communication technologies, and electronics in general render it increasingly feasible to track and record the changing positions of objects capable of continuous movement. Indexing of such positions is necessary in some Location-Based Services



Indexing of the Current and Near-Future Positions of Moving Objects. Figure 5. Timeslice query (*dashed*) and bounding interval (*solid*) in dual ($x(t_{ref})$, v)-space and (x, t)-space.

(LBS), such as location-based games, tourist-related services, safety-related services, and transport-related services (e.g., fleet tracking).

Process Monitoring

Applications such as process monitoring do not depend on positioning technologies. In these, the position of a "moving point" could for example be a pair of temperature and pressure values at a specific sensor. A timeslice query then would retrieve all sensors with current measurements of temperature and pressure in given ranges.

Future Directions

Tracking continuous real-world phenomena inevitably involves high rates of updates that have to be processed by the index. Very recent research provides a number of interesting ideas for speeding up the processing of index updates. Further research is needed to fully explore trade-offs among update performance, query performance, and query accuracy. Finally, main-memory indexing of such data could be explored to dramatically boost the performance of index updates.

How to handle the always-present uncertainty about the positions of objects has not been sufficiently explored in connection with indexing and warrants further study.

URL to Code

The source code of the TPR-tree can be found at: http://www.cs.aau.dk/∼simas/

The source code of the TPR*-tree can be found at: http://www.rtreeportal.org/

Cross-references

- ▶ Indexing Historical Spatio-Temporal Data
- ▶ R-Tree (and family)
- ▶ Spatial Indexing Techniques
- ▶ Spatio-Temporal Trajectories

Recommended Reading

1. Agarwal P.K., Arge L., and Erickson J. Indexing Moving Points. In Proc. 19th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2000, pp. 175–186.
2. Basch J., Guibas L.J., and Hershberger J. Data Structures for Mobile Data. In Proc. 8th Annual ACM -SIAM Symp. on Discrete Algorithms, 1997, pp. 747–756.
3. Benetis R., Jensen C.S., Karčiauskas G., and Šaltenis S. Nearest and Reverse Nearest Neighbor Queries for Moving Objects. VLDB J., 15(3):229–249, 2006.
4. Jensen C.S., Lin D., and Ooi B.C. Query and Update Efficient B$^+$-Tree Based Indexing of Moving Objects. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 768–779.
5. Jensen C.S., Tiešytė D., and Tradišauskas N. Robust B$^+$-Tree-Based Indexing of Moving Objects. In Proc. 7th Int. Conf. on Mobile Data Management, 2006, p. 12.
6. Kollios G., Gunopulos D., and Tsotras V.J. On Indexing Mobile Objects. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp, Principles of Database Systems, 1999, pp. 261–272.
7. Mokbel M. F., Ghanem T.M., and Aref W.G. Spatio-Temporal Access Methods. IEEE Data Eng. Bull., 26(2):40–49, 2003.
8. Patel J.M., Chen Y., and Chakka V.P. STRIPES: An Efficient Index for Predicted Trajectories. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 637–646.
9. Prabhakar S., Xia Y., Kalashnikov D.V., Aref W.G., and Hambrusch S.E. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. IEEE Trans. Computers, 51(10):1124–1140, 2002.
10. Procopiuc C.M., Agarwal P.K., and Har-Peled S. STAR-Tree: An Efficient Self-Adjusting Index for Moving Objects. In Proc. of ALENEX Workshop, 2002, pp. 178–193.

11. Šaltenis S. and Jensen C.S. Indexing of Moving Objects for Location-Based Services. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 463–472.
12. Šaltenis S., Jensen C.S., Leutenegger S.T., and Lopez M.A. Indexing the Positions of Continuously Moving Objects. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 331–342.
13. Tao Y., Papadias D., and Sun J. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 790–801.
14. Tayeb J., Ulusoy Ö., and Wolfson O. A quadtree based dynamic attribute indexing method. Computer J., 41(3):185–200, 1998.
15. Yiu M.L., Tao Y., and Mamoulis N. The B^{dual}-Tree: indexing moving objects by space filling curves in the dual space. VLDB J., 17(3):379–400, 200.

Indexing the Past

► Indexing Historical Spatio-Temporal Data

Indexing the Web

Edleno Silva de Moura[1], Marco Antonio Cristo[2]
[1]Federal University of Amazonas, Manaus, Brazil
[2]FUCAPI, Manaus, Brazil

Synonyms

Web indexing

Definition

The process of collecting, parsing, and storing data to provide fast and accurate retrieval of content available on the web. The result of this process is a structure called index that maps the collected data (for instance, words, phrases, concepts, or sound fragments) to the web location where it is possible to find content associated with the data (for instance, pages containing these words, phrases, concepts, or music with the sound fragments). Depending on the data collected, several indices may be created. The process can be manual or automatic. Manually generated indices include web directories, back-of-book-style indices, and metadata. Automatically generated indices are normally associated with the infra-structure of search engines.

Historical Background

One of the first efforts to index the web content was developed by a MIT student, Matthew Grey, who created a program to estimate the size of the Web. This program, called Word Wide Web Wanderer, was able to recursively traverse the web's hypertext structure, retrieving all the referenced documents [11,12,15]. It was the first crawler (also known as robot, bot, or spider). The URLs collected by this program formed the first database of web sites, the Wandex. It also caused quite a controversy due to the large amount of bandwidth it used. In response to this in October 1993, Martijn Koster created a program called ALIWEB (Archie-like Indexing of the Web). To avoid using crawlers, Koster asked webmasters to post their own index information for each page they wanted listed. ALIWEB, however, never reached a size large enough to compete with crawler-based indexers.

After its controversy start, crawlers became a standard component in most search engine indexers. By the end of 1993, search services such as JumpStation, World Wide Web Worm, the Repository-based Software Engineering (RBSE), and Excite used crawlers to gather information of Web pages. In general, these services indexed words extracted from URL, title, and text fragments of the pages.

The indices used by search engines were not designed towards human navigation. Thus, unlike book indices or yellow pages, they did not allow users to freely explore web contents by narrowing the field of interest. This observation motivated the University of Texas to create Elnet Galaxy in January 1994, the oldest web directory. In this index, contents were manually reviewed and organized into hierarchical categories. As in ALIWEB, only pages submitted to the service were listed. In April 1994, two students from Stanford University, David Filo and Jerry Yang, expanded a personal list of pages into a searchable directory, Yahoo!. Unlike Galaxy, Yahoo! automated aspects of the gathering and categorization process, blurring the distinction between search engine and directory.

Still in 1994, a student of the University of Washington, Brian Pinkerton, started another crawler-based search engine, the WebCrawler. It was the first one to index the full text of the pages. Soon after, in July, a project from Carnegie-Mellon University, headed by Michael Mauldin, Lycos, provided indices supporting prefix matching, word proximity, and a multimillion page catalog. In 1995, Altavista started indexing text associated with images, music, and videos. In the following years, several other services were stated, such as Inktomi (1996), Google (1997), and MSN Search (1998), improving image and video indexing, addressing spam issues related to indexing, targeting other document and media formats as well as other web elements. For instance, the increasing adoption of link analysis in ranking algorithms motivated the maintenance of citation indices. Also in 1998, the Open Directory was started to create a human-reviewed directory constructed and maintained by a community of volunteers.

Foundations

Introduction

The web is a space for mass publication at an unprecedented scale. The earliest strategies to make all the information in this space discoverable by the users consisted of applying traditional information retrieval techniques to the new content [8,10]. As a consequence, the first search engines adopted approaches based on manually or automatically built indices.

When performing manual indexing, experts describe the web content by selecting terms which indicate what the content is about. These terms are normally selected from controlled vocabularies and are organized as flat or hierarchical lists. It is also common that the authors of the contents select keywords and metadata to describe them. The resulting indices may be oriented towards direct navigation by humans or to be retrieved by other programs.

An example of index oriented towards human navigation is the web directory. In such an index, links to sites are organized into hierarchical categories, according to the sites' contents. In web directories, normally the tasks of collecting and categorizing pages are carried out under supervision of human editors.

An example of index not oriented to humans is a hidden list of metadata. Metadata are data about data. As a mean of assisting a search engine to locate content or an information entity, they can be used to describe that content or entity. While not visible to humans, this information can provide contextual clues to automatic algorithms used by search engines.

Manual indexing presents two problems. First, it does not scale with the size of the web. This is an even more daunting problem considering the dynamic nature of web, where many documents have no persistent state and would require frequent revisions. Second, to be effective as a search tool, experts and users have to agree with respect to the taxonomy employed for classification, which is hard when using large taxonomies [8]. These problems contributed for the dominance of strategies based on automatic indexing.

In automatic indexing, a program (i.e., a web crawler) scans the web, collecting pages to extract data useful to describe them. The contents of the pages (text, media, links, etc.) are parsed, and decomposed into their components (tokens). Each token constitutes an atomic search criterion. For instance, in the case of textual content, the tokens are normally words and the most simple search query is composed of at least one word. They are normally converted to a standard form through some transformations and inserted into one or more data structures. These structures are the base for one or more indices.

The resulting indices are used by search engines to allow several search strategies, such as full-text search, and support different ranking algorithms, central components of systems with automatic indexing. Such algorithms increase the likelihood that the system will automatically order results according to their relevance to the users. Typically, the indexing is performed at a predetermined time interval due to the required time and processing costs.

While manually created indexes are commonly associated with natural language text, automatic indexers are likely to support additional kinds of media, such as images, sounds, and video. It is also common that indexers expand the textual content of a web page *p* with additional evidence such as the anchor text found in pages that link to *p*. The anchor text is the text highlighted by web browsers to indicate the presence of a link. Normally, it provides a useful description of the target page. Thus, pages with many incoming links can accumulate several descriptions emphasizing what the page is about. In fact, the use of different evidence is a common way to assist ranking algorithms to order results.

Data Structures to Support Automatic Indexing A typical data structure used to map tokens to their associated contents is the inverted file or inverted index [2,10]. The inverted file stores a list of occurrences of each token, normally in the form of a hash table or binary tree. In larger indices it normally assumes a form of a distributed hash table. The list associated with each token is called an inverted list.

By using an inverted file, the search engine can find content (for instance, a document) through direct access. In its simplest design, this structure can only determine if a token exists in a document, since the inverted lists store only occurrence information. Further, they can only determine which documents match a query. They do not rank them.

In real indexers, however, the inverted list stores additional information to support searching and ranking. Further, the list can be sorted in such way that some documents are more likely to be retrieved than others, which enables several ranking speedup optimizations.

Additional information typically stored in inverted lists include the frequency and position of a token. In the case of textual content, the frequency information can be used to assist ranking algorithms to judge the relevance of a document to a search query whereas the position information enables search algorithms to support word proximity and phrase search.

To ensure that more promising documents will be retrieved earlier, the inverted list is normally sorted according to a query-independent document score. This score is calculated by combining query-independent factors associated with the document, such as the frequency that users click on it, its link popularity, URL size, and spam score.

Other data structures are used to support automatic web indexing. They are generally adopted to speedup retrieval or provide additional information for ranking algorithms. For instance, by using only the previously described inverted file, a search engine can correctly answer phrase queries. For this, it has to retrieve the inverted lists for all words in the phrase and intersect them. In practice, this is a very slow process and other strategies are employed. A simple alternative strategy is to extend the inverted file to support phrases along with words, as tokens. In such a case, only common phrases are included in the data structure. Another strategy consists in using a structure that facilitates the processing of pairs of words. For instance, the inverted list associated with a word *w* can be divided into sublists, according to the words that follow *w*. For instance, for retrieving the list associated with the phrase “web indexing”, one can retrieve the inverted list of “web” and then the inverted sublist of “indexing.”

Additional data structures used to support automatic web indexing are citation indices, n-gram indices, term document matrices, and tries, to cite a few. A citation index stores hyperlinks between documents to assist citation analysis, a set of bibliometric techniques used by ranking algorithms. N-gram indices store sequences of length of data to support some types of retrieval, such as phrase retrieval. Term document matrices are sparse matrices that store the occurrence of the words in the documents, needed for a technique called latent semantic analysis [4]. Tries are ordered tree data structures used to store an associative array where the keys are usually strings. A particular kind of trie is a suffix tree, used to store the suffixes of data strings in order to carry out fast full text searches.

Challenges and Design Factors in Automatic Indexing

By using automatic methods, search engines are able to index a significant part of the web while sustaining very low response times, when compared to manual indexing. Given the huge size and diversity of the web, many challenges were faced to reach such a performance.

From an engineering point of view, it was necessary to deal with several design factors, such as the amount of computer storage necessary to support the index, how the index should be filtered, compressed, and distributed [14]. Other key design factors are the update policy, reliability, and required electric power.

Regarding distribution, for instance, indices are normally partitioned by documents, that is, each node contains the index for a subset of all documents [8]. To answer a query, such query has to be submitted to each subset of documents and the results merged before being shown to the user.

A strategy to reduce the use of disk, network, and memory resources is to compress the key data structures. These compressed structures enable fewer disk and network accesses leading to faster processing, in spite of the processing costs associated with compression and decompression algorithms.

From a computer science point of view, it was necessary to find properly data structures to support the desired search and ranking operations in the required time and space. Examples of aspects considered included the processing of strings and n-grams, the possibility of approximated searching, document position, and the type of media to be indexed.

It was also necessary to deal with several algorithmic challenges. For instance, the adoption of distributed architectures required parallel algorithms for building the indices and synchronize the crawlers. Space, time, and accuracy requirements motivated researches on data selection with minimum impact on ranking, compression, subject categorization, and clustering. Robustness and completeness requirements motivated research on format analysis in order to design methods able to handle malformed documents, documents whose content is dynamically generated, proprietary formats, and multiple character sets.

Given the importance of ranking results for search engines, indices include much evidence which require the development of new ranking algorithms to take advantage of them. For instance, by storing hyperlinks between pages it is possible to infer the importance of the pages. A page pointed to by many other pages, specially by important ones, is probably an important page. This is the central idea of a very popular ranking strategy called PageRank [3]. This idea can be refined by characterizing the importance of a page according to its role, in particular, as a hub or an authority. The hubness of a page is the quality associated with its capability to point to other interesting pages in the web, whereas its authority is related to how good is considered its own informative content. HITS is a very well known ranking algorithm that infers page importance by taking into consideration these roles [6]. In particular, HITS explores a recursive definition of hub and authority pages, that is, a good hub is the page that points to many good authorities and a good authority is the page that is pointed to by many good hubs.

From a linguistic point of view, it was necessary to deal with the complexity inherent to the processing of natural language [13]. This is particularly difficult due to the necessity to handle multiple languages [8]. In a multilingual indexer, even apparently simple tasks as recognizing the word boundaries represent a challenge, since words are not clearly separated by white spaces in some languages such as Chinese, Japanese, and Arabic. Further, to deal with issues such as ambiguity, compression, and properly match of sentences, many search engines use additional information such as the lexical category of the words and their roots. All of this information requires language-dependent techniques and, by extent, automated language recognition, a subject of ongoing research in natural language processing.

Key Applications

Web indexing is essential for making the wealth of information in the web discoverable by the users.

Future Directions

The web continues growing uncoordinatedly, at large scale and with great diversity of interests [8]. These aspects of the web raise several problems that affect indexing, motivating many studies to address them. For instance, since the number of pages in the web grows beyond the capabilities of search engines collecting them, research has been and will likely continue to be focused on improving the ability of recognizing as early as possible not so useful pages in order to avoid indexing them [1].

On the other hand, most of the information in the web is stored in databases and is only available through systems that generate pages dynamically in response to user interactions. These dynamic pages comprise which is called *the hidden web* (also referred to as *deep web*). As a consequence, much digital content is inaccessible to typical crawlers. Research is under way to make it possible to index the hidden web, for instance, by exploring search query syntax, standard formats of online resources, and application programming interfaces [9].

There is an increasingly interest on indexing rich media formats. Current indexing approaches for such media take advantage of text clues such as image legends, video subtitles or music lyrics. New research has focused on describing them by means of their content [7] in order to make possible, for instance, the retrieval of an image by its description or of music by a sound fragment. The main challenge regarding media indexing is how to represent it such that it can be efficiently stored, retrieved, and compared. For instance, a music retrieval system should be able to map pitches to notes by analyzing waveforms, to store these notes into an index structure, and to match note strings allowing a certain amount of noise [9].

Another research focus is fighting spam [5]. The diversity of interests in the Web along with a growing audience led to the adoption of several strategies to ensure top positions on search engine results lists. Many of these strategies consisted of manipulating the content of the pages to deceive the indexer by using misleading terms or abusing of metatags. In response, movements to standardize metatag content has emerged. Also, research has lead to the design of robust crawling algorithms able to deal with several spam traps and new methods to infer content quality and reliability.

Cross-references

- ▶ Information Retrieval
- ▶ Inverted Files
- ▶ Suffix Tree
- ▶ Text indexing Techniques
- ▶ Trie

Recommended Reading

1. Baeza-Yates R., Castillo C., Marin M., and Rodriguez A. Crawling a country: better strategies than breadth-first for web page ordering. In Proc. 14th Int. World Wide Web Conference, 2005. pp. 864–872.
2. Baeza-Yates R.A. and Ribeiro-Neto B. Modern information retrieval. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
3. Brin S. and Page L. The anatomy of a large-scale hypertextual web search engine. Comput. Netw. ISDN Syst., 30 (1–7):107–117, 1998.
4. Deerwester S., Dumais S.T., Landauer T.K., Furnas G.W., and Harshman R.A. Indexing by latent semantic analysis. J. Soc. Inf. Sci., 41(6):391–407, 1990.
5. Heymann P., Koutrika G., and Garcia-Molina H. Fighting spam on social web sites: A survey of approaches and future challenges. IEEE Internet Comput., 11(6):36–45, 2007.
6. Kleinberg J.M. Authoritative sources in a hyperlinked environment. J. ACM, 46(5):604–632, 1999.
7. Liu Y., Zhang D., Lu G., and Ma W.Y. A survey of content-based image retrieval with high-level semantics. Pattern Recognit., 40 (1):262–282, 2007.
8. Manning C.D., Raghavan P., and Schütze H. Introduction to Information Retrieval, Ch. 18, 19, 20 (optional). Cambridge University Press, Cambridge, 2008.
9. Mostafa J. Seeking better web searches. Sci. Am. Mag., February 2005.
10. Salton G. Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
11. Sonnenreich W. A History of Search Engines, 1997. Available at http://www.wiley.com/legacy/compbooks/sonnenreich/history.html.
12. Underwood L. A Brief History of Search Engines. Available at http://www.webreference.com/authoring/search_history.
13. Voorhees E.M. Natural language processing and information retrieval. In Information Extraction: Towards Scalable, Adaptable Systems, M.T. Pazienza (ed.), 1999, pp. 32–48.
14. Witten I.H., Moffat A., and Bell T.C. Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd edn. Morgan Kaufmann, Los Altos, CA, 1999.
15. Zakon R.H. Hobbes' Internet Timeline. Available at http://zakon.org/robert/internet/timeline/.

Indexing Units

Jaap Kamps
University of Amsterdam, Amsterdam, The Netherlands

Synonyms

Indexing granularity

Definition

Indexing units refers to the granularity of information in the retrieval system's index, which can be in principle any document part of a structured text, and as a consequence determines the possible units of retrieval. There are three basic approaches. The first approach is to index every potentially retrievable unit as a whole – the so-called element-based approach [13]. The second approach is to index disjoint nodes – and relying on aggregation or score propagation methods for scoring higher-level nodes [e.g., 1,12]. The third approach is to index only selected elements, for example by indexing particular element types in separate indexes [10]. Various mixtures of these approaches have also been applied.

All approaches make implicit or explicit assumptions on the (most likely) *unit of retrieval*. Although there may be no designated retrieval unit (such as the document or root node of the structured document), this also does not mean that every document part (such as a sub-tree of the structured document) is an equally desirable retrieval unit. Such assumptions may be relatively generic (such as paragraphs and sections being more informative than very short excerpts in bold or italics) or may depend on the query at hand (such as a structured query requesting elements with a particular tag). In all cases these assumptions depend on the sort of structured documents (which may range from strict XML databases to loosely structured textual documents with mark-up), and on the sort of information need (which may range from a strict database query with well defined semantics to a vague information retrieval topic of request). Structured text retrieval typically deals with loosely structured textual documents and vague information retrieval queries.

Historical Background

Structured text retrieval has a long pre-history in text retrieval. The first test collections consisted of short document surrogates such as bibliographic descriptions (in various forms) or abstracts. Since the 1990s, test collections consisted predominantly of full text newspaper and newswire data. Interestingly, the bibliographic descriptions were highly structured catalogue records, and the newspaper data were typically structured in SGML, yet no particular use was made of the internal document structure. In fact, the use of the internal structure was usually explicitly outlawed, the main motivation for this being the desire to develop retrieval techniques that would work on all (flat) texts.

The use of document structure derived from SGML mark-up was pioneered by Wilkinson [15], studying ad hoc SGML element retrieval, and by Myaeng et al. [11], exploring structured queries that contain references to the SGML document structure. Similar early work on HTML is in [4]. There are also many similarities with the early work on multimedia retrieval, such as the DOLORES system [5] and the FERMI model [3], which are addressing the problem of retrieving information from structured documents. Ad hoc XML element retrieval and best entry point retrieval was studied in the Focus project [9]. In recent years, the main thrust of research in structured text retrieval is the annual initiative for the evaluation of XML retrieval [7].

Foundations

Structured text retrieval typically deals with loosely structured textual documents and vague information retrieval queries, and the discussion is focused exclusively on this case. Indexing structured texts presents a number of challenges since such documents can be decomposed according to their internal structure. XML documents have a hierarchical structure of nested elements (or subtrees). That is, for example, an entire article consisting of front matter, body, and back matter. The body, in turn, consists of sections. The sections, again, consist of subsections. The subsections of paragraphs, and so on. Since there is no fixed unit of retrieval it is an open question what should be put in the index.

A prototypical structured text retrieval task is XML element retrieval. In text retrieval, evaluation is based on a "frozen" set of search requests (or "topics") with a set of known relevant results – XML elements regarded as relevant by the topic author. In XML retrieval, topics consist of a short keyword (or content-only) title; a structured (or content-and-structure) query in NEXI [14]; a single sentence description; and a long narrative statement of the search request. The retrieval system takes as *input* the standard keyword query, or the structured NEXI query. To give a concrete example, the ⟨title⟩ field of the INEX 2004 topic number 104 is

► *Toy Story*

The requested *output* is a ranked list of document components (in this case XML elements in the INEX IEEE collection containing full-text articles). There is no fixed *unit of retrieval*: if a whole article is relevant, return the ⟨article⟩ element, but if only a section is relevant, return the ⟨sec⟩ element. Retrieval systems will, of course, rank the XML elements based on the occurrences of query terms (or possibly phrases, stems, or synonyms based on these terms). However, whether a result is indeed relevant for the query is determined by a human judgment on whether the information in the element satisfies the topic author's information need. For the above mentioned INEX 2004 topic 104, the ⟨narrative⟩ field reads

► *To be relevant, a document/component must discuss some detail of the techniques or computer infrastructure used in the creation of the first entirely computer-animated feature-length movie, "Toy Story."*

A human judge (usually the topic author) will assess the relevance of the retrieved results, where relevant means that the element is both *exhaustive* (it provides useful information on the topic of request) and *specific* (there is a minimal amount of off-topic material) (There have been different measures developed over the years).

Although, in principle, any document part or XML element can be retrieved, some document parts tend to be more likely to be relevant. Table 1 shows the distribution of relevant elements over tag-names (Here relevant is according to the strict quantization function). In this case, most frequently paragraphs (⟨p⟩), sections (⟨sec⟩), and subsections (⟨ss1⟩, ⟨ss2⟩) are judged relevant, but also entire articles (⟨article⟩). The precise tags are a direct result of the particular mark-up structure of the IEEE collection, which is mostly based on the logical structure of the articles. Generalizing over the particular tag-names, there is also a suggestion on the granularity of information that is most likely to be a relevant result. The following two observations present themselves. First, the most frequent elements such as paragraphs and (sub)sections are of relatively short-length.

Indexing Units. Table 1. Distribution of relevant elements over tag names (reproduced from [8])

2002 assessments			2003 assessments		
Tag-name	Frequency	%	Tag-name	Frequency	%
⟨p⟩	383	27.47%	⟨sec⟩	303	20.89%
⟨article⟩	309	22.16%	⟨p⟩	303	20.89%
⟨sec⟩	291	20.87%	⟨article⟩	172	11.86%
⟨ss1⟩	115	8.24%	⟨bdy⟩	167	11.51%
⟨bdy⟩	90	6.45%	⟨ss1⟩	146	10.06%
⟨ip1⟩	61	4.37%	⟨ip1⟩	69	4.75%
⟨ss2⟩	25	1.79%	⟨ss2⟩	36	2.48%
⟨abs⟩	22	1.57%	⟨fig⟩	32	2.20%
⟨fm⟩	13	0.93%	⟨app⟩	20	1.37%
⟨st⟩	11	0.78%	⟨bb⟩	19	1.31%
⟨item⟩	8	0.57%	⟨art⟩	18	1.24%
⟨app⟩	7	0.50%	⟨bm⟩	17	1.17%

Indexing Units. Table 2. Example Document and Indexing Units

Example Document	Indexing subtrees	Indexing disjoint nodes
⟨article⟩ ⟨title⟩XXX⟨/title⟩ ⟨abstract⟩YYY⟨/abstract⟩ ⟨body⟩ ⟨sec⟩ZZZ⟨/sec⟩ ⟨sec⟩VVV⟨/sec⟩ ⟨/body⟩ ⟨/article⟩	1. ⟨article⟩ XXX YYY ZZZ VVV⟨/article⟩	1. ⟨title⟩XXX⟨/title⟩
	2. ⟨title⟩XXX⟨/title⟩	2. ⟨abstract⟩YYY⟨/abstract⟩
	3. ⟨abstract⟩YYY⟨/abstract⟩	3. ⟨sec⟩ZZZ⟨/sec⟩
	4. ⟨body⟩ZZZ VVV⟨body⟩	4. ⟨sec⟩VVV⟨/sec⟩
	5. ⟨sec⟩ZZZ⟨/sec⟩	
	6. ⟨sec⟩VVV⟨/sec⟩	

Second, there is great variety of elements regarded as relevant. Even for a single topic there is a very similar variety of elements, making clear that relevancy is both depending on the topic of request, and on the precise structure of the document at hand.

Recall the question of what to put in the index. The most obvious approach is to index all information in the structured text. But already here different options present themselves, as is illustrated in Table 2. On the left-hand side of Table 2 is a very simple example document, an article with title, abstract and two sections. The first approach, shown in the middle of Table 2, is to index every retrievable unit, in this case every XML element. In a "bag of words" approach all six XML elements of the document are indexed separately, but each with all the content or text contained inside the element. That is, an element is indexed with both the text nodes directly contained in it, and all text nodes of its descendants. The indexing of subtrees of the XML hierarchy is known as the element-based approach [13]. Indexing subtrees is closest to traditional information retrieval since each XML node is a *bag of words* of itself and its descendants, and can be scored as ordinary plain text document. This directly relates structured text retrieval to the more general and well-understood problem of document retrieval. The indexing scheme is only using the structure to decompose the document into all retrievable units, and hence is applicable to any structured text. The main disadvantage is that it leads to a highly redundant index: text occurring at depth *n* of the XML tree is indexed *n* times.

On the right-hand side of Table 2, an alternative approach is shown, in which the text is only indexed once at the node where it occurs. The ⟨article⟩ and ⟨body⟩ elements are missing since they have no textual content. Since there are only four elements with content, the index is much smaller, and there is no redundancy of information. But the main advantage of indexing disjoint nodes is also creating a new problem of scoring higher level nodes. For example, as shown above, the whole article is a reasonably attractive XML element type, but it may not even occur in the index. This creates both a practical and a fundamental problem. The practical problem is that articles may contain thousands of XML elements, and hence may require considerable propagation of scores over the navigational axis. The fundamental problem is that it is not evident how to aggregate scores to ancestor elements, and various approaches exist in the literature. One of the earliest proposals is the augmentation approach of Abolhassini et al. [1], a straightforward propagation of scores to ancestor elements. The following simplified example illustrates the main idea behind augmentation. Assume the following document.

```
⟨body⟩
  ⟨sec⟩cat...⟨/sec⟩
  ⟨sec⟩dog...⟨/sec⟩
⟨/body⟩
```

and a query consisting of the two terms "cat dog." Furthermore assume that: /body/sec[1] scores 0.7 for cat; /body/sec[2] scores 0.4 for dog; and the rest 0 (that is, dog does not occur in the first section, and cat not in the second section). The problem is to determine the score of the body, which is not indexed itself. The *augmentation* approach propagates scores up with a certain weight (the augmentation factor) which is set to 0.3 based on experiments. The motivation for the augmentation factor is to avoid larger elements accumulating scores, and thus (almost) always get higher rankings than elements deep in the hierarchy. So in this case, the /body[1] will score $0.3 * 0.7 = 0.21$ for cat; and $0.3 * 0.4 = 0.12$ for dog. So the element /body[1], although not in the index, will be returned. In fact, it will be the highest ranked result for "andish" query evaluation where only results containing all query terms are returned. A very similar approach is taken by the GPX model and its decay factor [6]. The element specific language models of Ogilvie and Callan [12] provide an elegant alternative approach within the language modeling framework. Here, every XML element forms a particular language model, and the ancestor elements are modeled as mixture language models of their direct children. A final alternative is to just propagate term frequencies and effective reconstructing the element-based index discussed above, e.g., using the region models of Burkowski [2].

A third indexing approach is to index only selected elements in their entirety. This is essentially the approach taken in the FERMI model of Chiaramella [3]. The selection is tailored to the collection at hand, usually based on the human interpretation of the tags in the collection. An example of this approach is to index particular types of elements separately [10]. Mass and Mandelbrod [10] create separate indexes for articles (⟨article⟩, abstracts ⟨abs⟩, sections ⟨sec⟩, subsections ⟨ss1⟩, sub-subsections ⟨ss2⟩, and paragraphs ⟨p⟩ and ⟨ip1⟩). Each index provides statistics tailored to particular components, which may be an advantage if language statistics deviate significantly between element types. This is essentially a distributed approach where queries are issued to all indexes, and the results of each of the indexes are combined after score normalization. The selective indexing approach turned out to be effective for the IEEE collection used at INEX. The requirement to select particular element-types makes it strongly collection-dependent, and it is less straightforward to apply this indexing approach to arbitrary structured text.

Three prototypical indexing approaches for structured text have been discussed above. Some observations present themselves. First, there is a trade-off between exploiting the document structure, and being generically applicable to all structured text. The element-based approach uses the document structure for decomposing documents by focusing on the hierarchical structure only. This ignores (potentially) useful structure like the tag-names, their attributes, the schema or DTD, etc. However, to phrase it positively, since it is completely schema-ignorant the approach can handle data with any type of tag-structure, even mixed-schema XML. The selective indexing approach is on the opposite side of the spectrum. Here, the specific tags and their semantics and importance have to be taken into account when selecting the element types to index. In cases where it is known what the more important elements are, this gives powerful handles to exploit this information. The downside is, of course, that the particular choice of element to index, and thereby the effectiveness of the approach, is completely dependent on the collection at hand. Second, there is a trade-off between indexing and query time complexity. The element based approach seems unattractive since its index is highly redundant: text appearing in a given element, will also appear in all the index entry for all the ancestors of this element. This may not be as undesirable as it may appear at first glance, since it can be viewed as a trade-off between query time and storage space complexity. The redundant index has essentially "precomputed" term frequencies per element, that otherwise need to be computed at query run time, and hence has relatively low query time complexity. The indexing complexity of the disjoint nodes approach requires much less storage space. However, an article may contain thousands of XML elements, and hence will require considerable propagation of scores over the navigational axes of the document at query time. Third, the indexing methods and retrieval models are standard information retrieval approaches or straightforward extensions of them.

Cross-references

▶ Aggregation-based Structured Text Retrieval
▶ Evaluation metrics for structured text retrieval
▶ INitiative for the Evaluation of XML Retrieval
▶ Propagation-based structured text retrieval
▶ XML retrieval

Recommended Reading

1. Abolhassani M., Fuhr N., and Malik S. HyREX at INEX 2003. In N. Fuhr, M. Lalmas, and S. Malik, editors, Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, pp. 27–32.
2. Burkowski F.J. Retrieval activities in a database consisting of heterogeneous collections of structured text. In Proc. 15th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1992, pp. 112–125.
3. Chiaramella Y. Browsing and querying: Two complementary approaches for multimedia information retrieval. In Hypertext - Information Retrieval - Multimedia, 1997, pp. 9–26.
4. Cutler M., Shih Y., and Meng W. Using the structure of HTML documents to improve retrieval. In Proc. 1st USENIX Symp. on Internet Tech. and Syst., 1997.
5. Fuhr N., Gövert N., and Rölleke T. DOLORES: A system for logic-based retrieval of multimedia objects. In Proc. 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1996, pp. 257–265.
6. Geva S. GPX – Gardens Point XML IR at INEX 2004. In Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 211–223.
7. INEX. INitiative for the Evaluation of XML Retrieval, 2007. http://inex.is.informatik.uni-duisburg.de/.
8. Kamps J., de Rijke M., and Sigurbjörnsson B. Length normalization in XML retrieval. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 80–87.
9. Kazai G., Lalmas M., and Reid M. Construction of a test collection for the focussed retrieval of structured documents. In Proc. 25th European Conf. on IR Research, 2003, pp. 88–103.
10. Mass Y., and Mandelbrod M. Retrieving the most relevant XML components. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, pp. 53–58.
11. Myaeng S.H., Jang D.H., Kim M.S., and Zhoo Z.C. A flexible model for retrieval of SGML documents. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 138–145.
12. Ogilvie P., and Callan J. Using language models for flat text queries in XML retrieval. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, pp. 12–18.
13. Sigurbjörnsson B., Kamps J., and de Rijke M. An Element-Based Approch to XML Retrieval. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, pp. 19–26.
14. Trotman A. and Sigurbjörnsson B. Narrowed Extended XPath I (NEXI). In Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 16–40.
15. Wilkinson R. Effective retrieval of structured documents. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1994, pp. 311–317.

Individual Data

▶ Micro Data

Individually Identifiable Data

Chris Clifton
Purdue University, West Lafayette, IN, USA

Synonyms

Personally identifiable data; Personal data

Definition

Individually Identifiable Data is data that identifies the person that the data is about, or that can be used to identify that individual. This generally refers to data that contains either an identification number, or factors relating to physical, mental, economic, cultural, or social identity that could be used to link the data to an individual. Regulatory requirements for privacy generally apply (only) to individually identifiable data.

Key Points

Individually Identifiable Data is a legal term (e.g., personal data in the EU Privacy Directive [1], or Individually identifiable health information in U.S. Healthcare Laws [2]); in general privacy laws only protect individually identifiable data. Unfortunately, it is not clearly defined in technical terms. Anonymous data is presumably not individually identifiable, but what about *k-anonymous* data? (Presumably data is individually identifiable if $K = 1$, but at what level of K is data no longer individually identifiable?) This leads to considerable difficulty in developing data management technology that balances privacy with the utility of disclosing anonymous data.

Regulatory frameworks do give some guidance; for example the U.S. Healthcare Privacy rules allow data to be considered not individually identifiable if names, geographic units of less than 20,000 people, dates of finer granularity than a year (or that can indicate age for those over 89), biometric identifiers (fingerprint, full-face images), or any identifying numbers (telephone, account, vehicle license, IP address, etc.) other than numbers generated specifically for the particular dataset.

Cross-references

▶ Anonymity
▶ K-Anonymity
▶ Privacy Metrics

Recommended Reading

1. Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995. The protection of individuals with regard to the processing of personal data and on the free movement of such data. Off. J. Eur. Communities, 1(281):31–50, 1995.
2. U.S. Department of Health and Human Services Office for Civil Rights. Standard for privacy of individually identifiable health information. Technical report, August 2003.

INEX

▶ INitiative for the Evaluation of XML retrieval (INEX)

Inference Control in Statistical Databases

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona, Catalonia

Synonyms

Statistical disclosure control (SDC); Statistical disclosure limitation (SDL)

Definition

Inference control in databases, also known as Statistical Disclosure Control (SDC), is a discipline that seeks to protect data so they can be published without revealing confidential information that can be linked to specific individuals among those to which the data correspond. SDC is applied to protect *respondent privacy* in areas such as official statistics, health statistics, e-commerce (sharing of consumer data), etc. Since data protection ultimately means data modification, the challenge for SDC is to achieve protection with minimum loss of the accuracy sought by database users.

Historical Background

The literature on inference control started in the 1970s, with the seminal contribution by Dalenius [4] in the statistical community and the works by Schlörer [12] and others in the database community. The 1980s saw moderate activity in this field. An excellent survey of the state of the art at the end of the 1980s is [1]. In the 1990s, there was renewed interest in the statistical community and the discipline was further developed under the names of statistical disclosure control in Europe and statistical disclosure limitation in America. Subsequent evolution has resulted in at least three clearly differentiated subdisciplines:

- *Tabular data protection.* This is the oldest and best established part of SDC, because tabular data have been the traditional output of national statistical offices. The goal here is to publish *static* aggregate information, i.e., tables, in such a way that no confidential information on specific individuals among those to which the table refers can be inferred. See [13] for a conceptual survey and [7] for a software survey.
- *Queryable databases.* The scenario here is a database to which the user can submit statistical queries

(sums, averages, etc.). The aggregate information obtained by a user as a result of successive queries should not allow him to infer information on specific individuals. Since the 1980s, this has been known to be a difficult problem, subject to the tracker attack [11]. SDC strategies here include perturbation, query restriction and camouflage (providing interval answers rather than exact answers).

- *Microdata protection.* This subdiscipline is about protecting static individual data, also called microdata. It is only recently that data collectors (statistical agencies and the like) have been persuaded to publish microdata. Therefore, microdata protection is the youngest subdiscipline and is experiencing continuous evolution in the last years.

Good general works on SDC are [13,9].

Foundations

Statistical disclosure control will be first reviewed for tabular data, then for queryable databases and finally for microdata.

In spite of tables displaying aggregate information, there is risk of disclosure in tabular data release. Several attacks are conceivable:

- *External attack.* For example, let a frequency table "Job" × "Town" be released where there is a single respondent for job J_i and town T_j. Then if a magnitude table is released with the average salary for each job type and each town, the exact salary of the only respondent with job J_i working in town T_j is publicly disclosed.
- *Internal attack.* Even if there are two respondents for job J_i and town T_j, the salary of each of them is disclosed to each other.
- *Dominance attack.* If one (or a few) respondents dominate in the contribution to a cell of a magnitude table, the dominant respondent(s) can upper-bound the contributions of the rest (e.g., if the table displays the total salary for each job type and town and one individual contributes 90% of that salary, he knows that his colleagues in the town are not doing very well).

SDC methods for tables fall into two classes: non-perturbative and perturbative. Non-perturbative methods do not modify the values in the tables; the best known method in this class is *cell suppression* (CS). Perturbative methods output a table with some modified values; well-known methods in this class include *controlled rounding* (CR) and the recent *controlled tabular adjustment* (CTA).

The idea of CS is to suppress those cells (primary suppressions) that are identified as sensitive by the so-called sensitivity rules. The two most common sensitivity rules are:

- *Dominance rule (n, k).* The cell is deemed sensitive if n or less respondents contribute at least k% of the cell value.
- *p%-rule.* The cell is deemed sensitive if some respondent can estimate the contribution by another respondent within p% accuracy.

After primary suppressions, additional suppressions (secondary suppressions) are performed to prevent primary suppressions from being computed or even inferred within a prescribed protection interval using the row and column constraints (marginal row and column totals). Usually, one attempts to minimize either the number of secondary suppressions or their pooled magnitude, which results in complex optimization problems. Most optimization methods used are heuristic, based on mixed integer linear programming or network flows (e.g., [6]), most of them implemented in the τ-Argus free software package [10]. CR rounds values in the table to multiples of a rounding base. This may entail rounding the marginal totals as well. On its side, CTA modifies the values in the table to prevent the inference of values of sensitive cells within a prescribed protection interval. The idea of CTA is to find the *closest* table to the original one that ensures such a protection for all sensitive cells. This requires optimization methods, which are typically based on mixed linear integer programming. Usually CTA causes less information loss than CS.

Regarding SDC of queryable databases, there are three main approaches to protect a confidential vector of numerical data from disclosure through answers to user queries:

- *Data perturbation.* Perturbing the data is a simple and effective approach whenever the users do not require deterministically correct answers to queries that are functions of the confidential vector. Perturbation can be applied to the records on which queries are computed (input perturbation) or to the query result after computing it on the original data (output perturbation). An example of perturbation methods can be found in [5].

- *Query restriction*. This is the right approach if the user does require deterministically correct answers and these answers have to be exact (i.e., a number). Since exact answers to queries provide the user with a very powerful information, it may become necessary to refuse to answer certain queries at some stage to avoid disclosure of a confidential datum. There are several criteria to decide whether a query can be answered; one of them is query set size control, that is, to refuse answers to queries which affect a set of records which is too small. An examples of the query restriction approach can be found in [3].
- *Camouflage*. If deterministically correct non-exact answers (i.e., small interval answers) suffice, confidentiality via camouflage (CVC [8]) is a good option. With this approach, unlimited answers to any conceivable query types are allowed. The idea of CVC is to "camouflage" the confidential vector a by making it part of the relative interior of a compact set Π of vectors. Then each query $q = f(a)$ is answered with an inverval $[q^-, q^+]$ containing $[f^-, f^+]$, where f^- and f^+ are, respectively, the minimum and the maximum of f over Π.

Regarding microdata SDC, given an original microdata set V, its goal is to release a protected microdata set V' in such a way that:

1. Disclosure risk (i.e., the risk that a user or an intruder can use $\mathbf{V}'$ to determine confidential attributes on a specific individual among those in $\mathbf{V}$) is low.
2. User analyses (regressions, means, etc.) on $\mathbf{V}'$ and on $\mathbf{V}$ yield the same or at least similar results.

Microdata protection methods can generate the protected microdata set $\mathbf{V}'$ either by

- *Masking original data*, i.e., generating $\mathbf{V}'$ a modified version of the original microdata set $\mathbf{V}$ or
- *Generating synthetic data* $\mathbf{V}'$ that preserve some statistical properties of the original data $\mathbf{V}$.

Masking methods can in turn be divided in two categories depending on their effect on the original data [13]:

- *Perturbative*. The microdata set is distorted before publication. In this way, unique combinations of scores in the original dataset may disappear and new unique combinations may appear in the perturbed dataset; such confusion is beneficial for preserving statistical confidentiality. The perturbation method used should be such that statistics computed on the perturbed dataset do not differ significantly from the statistics that would be obtained on the original dataset. *Noise addition, microaggregation, data/rank swapping, microdata rounding and PRAM* are examples of perturbative masking methods.
- *Non-perturbative. Non-perturbative masking methods* do not alter data; rather, they produce partial suppressions or reductions of detail in the original dataset. Sampling, global recoding, top and bottom coding and local suppression are examples of non-perturbative masking methods.

Microdata SDC faces an inherent trade-off between loss of information (i.e., analytical utility) and disclosure risk. Given a particular dataset, the microdata SDC method optimizing that trade-off is the best choice. Approaches to modeling such a trade-off include *SDC scores*, *k-anonymity* and R-U maps (plots of disclosure risk against data utility for a certain parameterization of an SDC method).

Key Applications

There are several areas of application of SDC techniques, which include but are not limited to the following:

- *Official statistics*. Most countries have legislation which compels national statistical agencies to guarantee statistical confidentiality when they release data collected from citizens or companies. This justifies the research on SDC undertaken by several countries, among them the European Union (e.g., the CASC project [2]) and the United States.
- *Health information*. This is one of the most sensitive areas regarding privacy. For example, in the U.S., the Privacy Rule of the Health Insurance Portability and Accountability Act (HIPAA) requires the strict regulation of protected health information for use in medical research. In most western countries, the situation is similar.
- *E-commerce*. Electronic commerce results in the automated collection of large amounts of consumer data. This wealth of information is very useful to companies, which are often interested in sharing it with their subsidiaries or partners. Such consumer information transfer should not result in public

profiling of individuals and is subject to strict regulation, especially in the European Union and the United States.

Future Directions

There are many open issues in SDC, some of which can be hopefully solved with further research and some which are likely to stay open due to the inherent nature of SDC. First some of the issues that probably could and should be settled in the near future are listed:

- Identifying a comprehensive listing of data uses (e.g., regression models, association rules, etc.) that would allow the definition of data use-specific information loss measures broadly accepted by the community; those new measures could complement and/or replace the generic measures currently used. Work in this line has been started in Europe in 2006 under the CENEX SDC project sponsored by Eurostat.
- Devising disclosure risk assessment procedures which are as universally applicable as record linkage while being less greedy in computational terms.
- Identifying, for each domain of application, which are the external data sources that intruders can typically access in order to attempt re-identification. This would help data protectors figuring out in more realistic terms which are the disclosure scenarios they should protect data against.
- Creating one or several benchmarks to assess the performance of SDC methods. Benchmark creation is currently hampered by the confidentiality of the original datasets to be protected. Data protectors should agree on a collection of non-confidential original-looking data sets (financial datasets, population datasets, etc.) which can be used by anybody to compare the performance of SDC methods. The benchmark should also incorporate state-of-the-art disclosure risk assessment methods, which requires continuous update and maintenance.

There are other issues whose solution seems less likely in the near future, due to the very nature of SDC methods. If an intruder knows the SDC algorithm used to create a protected data set, he can mount algorithm-specific re-identification attacks which can disclose more confidential information than conventional data mining attacks. Keeping the SDC algorithm secret used would seem a solution, but in many cases the protected dataset itself gives some clues on the SDC algorithm used to produce it. Such is the case for a rounded, microaggregated or partially suppressed microdata set. Thus, it is unclear to what extent the SDC algorithm used can be kept secret.

Cross-references

▶ Data Rank/Swapping
▶ Disclosure Risk
▶ Information Loss Measures
▶ *k*-Anonymity
▶ Microaggregation
▶ Microdata
▶ Microdata Rounding
▶ Noise Addition
▶ Non-Perturbative Masking Methods
▶ PRAM
▶ Record matching
▶ SDC Score
▶ Synthetic Microdata
▶ Tabular Data

Recommended Reading

1. Adam N.R. and Wortmann. J.C. Security-control for statistical databases: a comparative study. ACM Comput. Surv., 21(4):515–556, 1989.
2. CASC. Computational aspects of statistical confidentiality, 2004. European project IST-2000-25069 CASC, Fifth FP, 2001–2004, http://neon.vb.cbs.nl/casc.
3. Chin F.Y. and Ozsoyoglu G. Auditing and inference control in statistical databases. IEEE Trans. Software Eng., SE-8:574–582, 1982.
4. Dalenius T. The invasion of privacy problem and statistics production. An overview. Statistik Tidskrift, 12:213–225, 1974.
5. Duncan G.T. and Mukherjee S. Optimal disclosure limitation strategy in statistical databases: deterring tracker attacks through additive noise. J. Am. Stat. Assoc., 45:720–729, 2000.
6. Fischetti M. and Salazar J.-J. Solving the cell suppression problem on tabular data with linear constraints. Manag. Sci., 47(7):4, 2001.
7. Giessing S. Survey on methods for tabular data protection in argus. In J. Domingo-Ferrer and V. Torra (eds.)., Privacy in Statistical Databases, LNCS. vol. 3050. Springer, 2004, pp. 1–13.
8. Gopal R., Garfinkel R., and Goes P. Confidentiality via camouflage: the CVC approach to disclosure limitation when answering queries to databases. Operat. Res., 50:501–516, 2002.
9. Hundepool A., Domingo-Ferrer J., Franconi L., Giessing S., Lenz R., Longhurst J., Schulte-Nordholt E., Seri G., and DeWolf P.-P. Handbook on Statistical Disclosure Control (version 1.0). Eurostat (CENEX SDC Project Deliverable), 2006.
10. Hundepool A., van de Wetering A., Ramaswamy R., de Wolf P.-P., Giessing S., Fischetti M., Salazar J.-J., Castro J., and Lowthian P τ-*ARGUS v. 3.2 Software and User's Manual,* CENEX SDC Project Deliverable, February 2007. http://neon.vb.cbs.nl/casc/TAU.html

11. Schlörer J. Disclosure from statistical databases: quantitative aspects of trackers. ACM Trans. Database Syst., 5:467–492, 1980.
12. Schlörer J. Identification and retrieval of personal records from a statistical data bank. Methods Inform. Med., 14(1):7–13, 1975.
13. Willenborg L. and DeWaal T. Elements of Statistical Disclosure Control. Springer, Berlin Heidelberg New York, 2001.

Infinity

▶ Forever

Information

▶ Information Quality Assessment

Information Browsing

▶ Information Navigation

Information Displays

▶ Dense Pixel Displays

Information Extraction

Heng Ji
New York University, New York, NY, USA

Definition

Information Extraction (IE) is a task of extracting prespecified types of facts from written texts or speech transcripts, and converting them into structured representations (e.g., databases).

IE terminologies are explained via an example as follows.

- Input Sentence:

Media tycoon Barry Diller on Wednesday quit as chief of Vivendi Universal Entertainment, the entertainment unit of French giant Vivendi Universal whose future appears up for grabs.

- IE output:
  - Entities:
    Person Entity: {Media tycoon, Barry Diller}
    Organization Entity: {Vivendi Universal Entertainment, the entertainment unit}
    Organization Entity: {French giant, Vivendi Universal}
  - "Part-Whole" relation:
    {Vivendi Universal Entertainment, the entertainment unit} is part of {French giant, Vivendi Universal}.
  - "End-Position" event.

The above sentence includes a "Personnel_End-Position" event mention, with the trigger word which most clearly expresses the event occurrence, the position, the person who quit the position, the organization, and the time during which the event happened (Table 1).

Historical Background

The earliest IE system was directed by Naomi Sager of the Linguistic String Project group [11] in the medical domain. However, the specific task of information extraction was formally evaluated through the U.S. Defense Advanced Research Projects Agency (DARPA) sponsored Message Understanding Conferences (MUC) program from 1987 to 1998 [4].

There were four specific evaluations: Named entity, coreference and template element reflected in the evaluation tasks introduced for MUC-6, and template relation introduced in MUC-7.

The MUC tasks have been inherited by the U.S. National Institute of Standards and Technology (NIST) Automatic Content Extraction (ACE) program (The ACE task description can be found at http://www.nist.gov/speech/tests/ace/and the ACE guidelines at

Information Extraction. Table 1. Event extraction example

Trigger	Quit	
Person	Barry Diller	Media tycoon
Organization	Vivendi universal entertainment	The entertainment unit of french giant vivendi universal
Position	Chief	
Time-within	Wednesday	

http://www.ldc.upenn.edu/Projects/ACE/), with more general types of entities/relations/events defined. ACE includes the following tasks.

Entity Detection and Recognition

ACE defines the following terminologies for the entity detection and recognition task:

entity: an object or a set of objects in one of the semantic categories of interest

mention: a reference to an entity (typically, a noun phrase)

name mention: a reference by name to an entity

nominal mention: a reference by a common noun or noun phrase to an entity

Seven types of entities were defined: PER (persons), ORG (organizations), GPE ("geo-political entities" – locations which are also political units, such as countries, counties, and cities), LOC (other locations without governments, such as bodies of water and mountains), FAC (facility), WEA (Weapon) and VEH (Vehicle) mentioned in an input document. This task was proposed in 2000 and evaluated in English, and then expanded to include Chinese and Arabic in 2003, Spanish in 2007.

Relation Detection and Recognition

The relation detection task was proposed in 2002, aiming to find specified types of semantic relations between pairs of entities. ACE 2007 had 7 types of relations, with 19 subtypes. The following table lists some examples (Table 2).

Event Detection and Recognition

ACE defined 8 types of events, with 33 subtypes. Some examples are presented in Table 3.

Entity Translation

Entity Translation is a cross-lingual IE track at ACE 2007 to take in a document in a foreign language (e.g., Chinese or Arabic) and extract the English catalog of the entities.

Foundations

There are two main approaches to develop IE systems, described separately as follows.

Pattern Matching Based IE

Many IE systems during MUC evaluation use high-accuracy rules, dictionaries and patterns for each specific domain. For example, for the end-position event in Table 1, an IE system generates patterns such as

- [Person] quit as [Position] of [Organization]

Manually writing and editing patterns requires some skill and considerable time. So some systems have moved on to learning these patterns automatically based on an annotated corpus pre-processed by syntactic and semantic analyzers. A more comprehensive survey of pattern matching based IE approaches can be found in [8].

The above pattern acquisition is still quite costly because for particular domain a separate annotated corpus is needed. Therefore some systems have used unsupervised learning approach [10,12,13]. The general idea is to obtain a pattern if a pair of arguments (mostly names) (Arg_1, Arg_2) and their context C_{12} appear frequently in other instances of the event.

The idea of using bootstrapping to obtain patterns was first proposed by Riloff [10]. Riloff [10] manually pre-classified the documents into relevant and irrelevant, then collect and score patterns around each noun

Information Extraction. Table 2. Examples of the ACE relation types

Relation Type	Example
Agent-Artifact (User-Owner-Inventor-Manufacturer)	*Rubin Military Design, the makers of the Kursk*
ORG-Affliation (Employment)	Mr. Smith, *the CEO of Microsoft*
Gen-Affiliation (Citizen-Resident-Religion-Ethnicity)	*Salzburg Red Cross officials*
Physical (Near)	*A town some 50 miles south of Salzburg*

Information Extraction. Table 3. Examples of the ACE event types

Event Type	Example
Movement (Transport)	*Homeless people* have been *moved to schools*
Business (Start-ORG)	*Schweitzer founded a hospital* in *1913*
Conflict (Attack)	The *attack* on *Gaza* killed *13 people*
Personnel (Start-Position)	*Cornell Medical Center recruited 12 nursing students*
Justice (Arrest)	*Zawahiri was arrested* in *Iran*

phrase. Yangarber et al. [13] used seed patterns to address the limitation of manual document classification. They started with a few initial seed patterns, and then applied an incremental discovery procedure to identify new set of patterns. Both of [10,13] are based on predicate-argument or subject-verb-object structures. Sudo et al. [12] presented a new Subtree model based on dependency parsing, and proved the Subtree model can obtain higher recall while preserve high precision.

Machine Learning Based IE

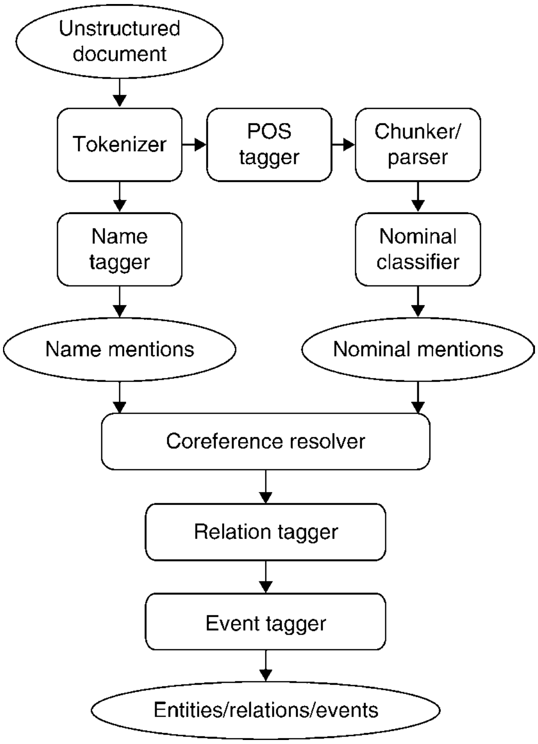
The IE systems relying entirely on pattern matching have attempted some success in MUC domains. However these patterns cannot be easily adapted into new domains. Therefore, IE research has grown by splitting the task into several components and then applying machine learning methods to address each component separately.

Machine learning based IE systems typically include name identification and classification, parsing (or partial parsing), semantic classification of nominal mentions, coreference resolution, relation extraction and event extraction. A typical IE system pipeline is presented in Fig. 1. For instance, state-of-the-art IE systems such as BBN system [2], IBM system [3] and NYU system [5] were developed in this pipeline style. This "pipeline" design provides great opportunity to applying a wide range of learning models and incorporating diverse levels of linguistic features to improve each component. Large progress has been achieved on some of these components. In the following some typical learning methods are described for the important components.

Trainable Name Tagging

The problem of name recognition and classification has been intensively studied since 1995, when it was introduced as part of the MUC-6 Evaluation. A wide variety of unified learning algorithms have been applied to the name tagging task, including Hidden Markov Models (HMMs), Maximum Entropy Models, Decision Trees, Conditional Random Fields and Support Vector Machines.

The most well-known BBN's Nymble name tagger [1] used several methods to improve performance over a simple HMM. Within each of the name class states, a statistical bigram model is employed, with the usual one-word-per-state emission. The various probabilities involve word co-occurrence, word features, and class probabilities. Since these probabilities are estimated based on observations seen in a corpus, several levels of "back-off models" are used to reflect the strength of support for a given statistic, including a back-off from words to word features.



Information Extraction. Figure 1. A minimal machine learning based IE system pipeline.

Trainable Coreference Resolution

Coreference Resolution is the task of determining whether two mentions refer to the same entity. For example in the sentence in Table 1, the name mention "Barry Diller" and the nominal mention "media tycoon" refer to the same person entity.

In a corpus-trained system, coreference resolution is usually converted into a supervised binary classification problem of determining whether a candidate mention is referring to an antecedent or not. Here an "antecedent" can be another single mention, or a cluster of mentions which the system has generated. Each pair is assigned probability value by a supervised learning based classifier. If the sampling is constructed on each mention pair, then a separate clustering algorithm is applied to group coreferring mentions.

Most coreference resolution systems use representations built out of the lexical and syntactic attributes of the mentions for which reference is to be established [9]. A typical feature set includes:

1. Representing agreement of various kinds between mentions (number, gender)
2. Degree of string similarity
3. Synonymy between mention heads
4. Measures of distance between mentions (such as the Hobbs distance)
5. The presence or absence of determiners or quantifiers

Though gains have been made with such methods, there are clearly cases where this sort of local information will not be sufficient to resolve coreference correctly. Coreference is by definition a semantic relationship, therefore a successful coreference system should exploit world knowledge, inference, and other forms of semantic relations in order to resolve hard cases. Since 2005, researchers have returned to the once-popular semantic-knowledge-rich approach, investigating a variety of semantic knowledge sources. For example, [7] incorporated the feedback from semantic relation detection to infer and correct coreference analysis. If, for example, two library mentions which are located in two different cities, then these mentions are less likely to corefer.

Trainable Relation Detection

For ACE-type relations, various machine learning methods have been used such as K-Nearest-Neighbor [9] and Support Vector Machines [14]. The typical features used to classify relations include:

1. The heads of the mentions and their context words
2. Entity and mention type of the heads of the mentions
3. The sequence of the heads of the constituents, chunks between the two mentions
4. The syntactic relation path between the two mentions
5. Dependent words of the mentions

Trainable Event Detection

A typical event extraction pipeline includes three main steps:

1. Trigger Identification
   Identify the trigger word in a given sentence and assign event type using the probability computed from the training corpora.
2. Argument Identification
   For a given trigger and a mention, determine whether the mention is an argument of the trigger or not.
3. Argument Classification
   For an identified argument, classify the argument as a specific event role.

Event detection heavily relies high-quality deep parsing [2,5] have further shown that the predicate-argument structures can provide deeper linguistic analysis and therefore effectively enhance the performance of event detection.

Key Applications

An enormous amount of information is now available through the Web. Much of this information is encoded in natural language, which makes it accessible to some people (those who can read the particular language), but much less amenable to computer processing (beyond simple keyword search). If computers can be enabled to extract and utilize the knowledge embedded in these texts, a powerful knowledge resource for many fields will be unleashed. Some typical applications of IE are presented as follows.

1. IE for Daily News
   IE can be applied to identify the events in the daily news articles. If an informative database can be returned based on the facts extracted by IE from multiple sources of news, it can be a very valuable result and save the time a user has to spend in browsing. For example, for the news articles about Olympic sport games, an IE engine can automatically provide a table of the player's person names, the team names they come from and the game results.
2. IE for Financial Reports
   Every year the U.S. government releases the annual reports from millions of industrial agencies. The financial analysis companies then gather all these reports and analyze the most up-to-date information such as the company start-up and merge events, the competition and cooperation relations among banks or companies. It will be very helpful if an automatic IE system is applied to compress these articles into data bases. Recently such IE systems are widely applied in the financial domain to assist human analysts.
3. IE for Biology Literatures
   In the biology domain, thousands of new papers and data sets are published in natural language on a daily basis. It has become impractical for scientists to manually track all these new results and observations, and manually mine the data sets to construct

a knowledge base. IE can play a significant role by automatically generating an accurate summary of facts (e.g., gene named entities) and predicting new results (e.g., Bio-nano structures of different peptide sequences), and thus assist scientists in decision making.

4. IE for Medical Reports
   Since the early work by Sager et al. [11], IE has obtained successful applications in processing the narrative clinical documents including patient discharge summaries and radiology reports. Some of these systems have shown positive impact on providing information to assist clinical decision, result analysis, error detection, etc.

Future Directions

For each IE component there are different aspects to improve. This section proposes some high-level directions in which IE can be further explored.

1. Cross-document Information Extraction
   One of the initial goals for IE was to create a database of relations and events from the entire input corpus, and allow further logical reasoning on the database. The artificial constraint that extraction should be done independently for each document was introduced in part to simplify the task and its evaluation.

   However, almost all the current event extraction systems focus on processing single documents and, except for coreference resolution, operate a sentence at a time. Therefore, one interesting area worth exploring would be to gather together IE results from a set of related documents, and then apply inference and constraints to propagate correct results and fix the wrong information generated from the within-document IE system [6].
2. IE for Noisy Input
   Recently there has been rapid progress in applying text processing techniques on "noisy" texts such as the output of automatic speech recognition (ASR) and machine translation (MT). The potential ASR transcription and machine translation errors, in particular name recognition errors, make IE more difficult. However, it is possible to optimize the parameters in the ASR or MT systems for IE purpose. Another interesting direction would be using IE results to provide feedback to ASR and MT in a joint inference framework.
3. Cross-lingual IE
   A shrinking fraction of the world's web pages are written in a language different from the user's own, and so the ability to access information from foreign languages is becoming increasingly important. This need can be addressed in part by the research on cross-lingual IE (CLIE).
4. Active Learning for Domain Adaptation
   Since about one decade ago in MUC program, the "portability" problem has become a noticeable bottleneck for IE techniques. Until today this problem has not yet been solved. There is an urgent need to develop effective adaptation algorithms to apply IE systems to a new domain with low cost. Active learning and semi-supervised learning techniques, which have achieved success on name tagging, may be worth expanding to all stages in the IE pipeline.

Experimental Results

The state-of-the-art IE results can refer to the ACE evaluation results on NIST website (http://www.nist.gov/speech/tests/ace/). All IE results are given in terms of the entity/relation/event value scores, as produced by the official ACE scorer. These value scores include weighted penalties for missing extractions, spurious extractions, and for type errors in corresponding extractions (Scoring details can be found in the ACE07 evaluation plan: http://www.nist.gov/speech/tests/ace/ace07/doc/ace07-evalplan.v1.3a.pdf). The top systems obtained mention values in the range of 70–85, entity values in the range of 60–70, relation values in the range of 35–45, event values in the range of 15–30.

Data Sets

- ACE IE: http://projects.ldc.upenn.edu/ace/data/
  IE training data for English/Chinese/Arabic/Spanish
- CONLL 2002: http://www.cnts.ua.ac.be/conll2002/ner.tgz
  Name tagging training data for Dutch and Spanish
- CONLL 2003: http://www.cnts.ua.ac.be/conll2003/ner.tgz
  Name tagging training data for English and German

URL to Code

- UIMA: http://incubator.apache.org/uima/svn.html
  IBM NLP platform

- Jet: http://www.cs.nyu.edu/cs/faculty/grishman/jet/license.html
  NYU IE toolkit
- Gate: http://gate.ac.uk/download/index.html
  University of Sheffield IE toolkit
- Mallet: http://mallet.cs.umass.edu/index.php/Main_Page
  University of Massachusetts NLP toolkit
- MinorThird: http://minorthird.sourceforge.net/
  Carnegie Mellon University NLP toolkit

Cross-references

▶ Column Segmentation
▶ Cross-Language Mining and Retrieval
▶ Languages for Web Data Extraction
▶ Structured and Semi-Structured Document Databases
▶ Text Indexing and Retrieval
▶ Text Summarization
▶ Topic Detection and Tracking
▶ Web Information Extraction
▶ Wrapper Induction

Recommended Reading

1. Bikel D.M., Miller S., Schwartz R., and Weischedel R. Nymble: a high-performance learning name-finder. In Proc. 5th Conf. on Applied Natural Language Processing, 1997, pp. 194–201.
2. Boschee E., Weischedel R. and Zamanian A. Automatic evidence extraction. In Proc Int. Conf. on Intelligence Analysis, McLean, VA, 2005.
3. Florian R., Jing H., Kambhatla N. and Zitouni I. Factorizing complex models: a case study in mention detection. In Proc. 26th Int. Conf. Computational Linguistics, 2006, pp. 473–480.
4. Grishman R. and Sundheim B. Message understanding conference – 6: a brief history. In Proc. 16th Int. Conf. on Computational Linguistics, 1996, pp. 466–471.
5. Grishman R., Westbrook D. and Meyers A. NYU's English ACE 2005 system description. In Proc. ACE 2005 Evaluation/PI Workshop, 2005.
6. Ji H. and Grishman R. Refining Event Extraction Through unsupervised cross-document inference. In Proc. 46th Annual Meeting Assoc. for Computational Linguistics, 2008, pp. 254–262.
7. Ji H., Westbrook D., and Grishman R. Using semantic relations to refine coreference decisions. Proc. Conf. Human Language Tech. and Empirical Methods in Natural Language Proc. 2005, pp. 17–24.
8. Muslea I. Extraction patterns for information extraction tasks: a survey. In Proc. National Conf. on Artificial Intelligence (AAAI-99) Workshop on Machine Learning for Information Extraction, 1999.
9. Ng V. and Cardie C. Improving machine learning approaches to coreference resolution. In Proc. 40th Annual Meeting of the Assoc. for Computational Linguistics, 2002, pp. 104–111.
10. Riloff E. Automatically generating extraction patterns from untagged text. In Proc. 10th National Conf. on AI, 1996, pp. 1044–1049.
11. Sager N. Natural Language Information Processing: A Computer Grammar of English and its Applications. Addison Wesley, Reading, MA, 1981.
12. Sudo K., Sekine S. and Grishman R. An improved extraction pattern representation model for automatic IE pattern acquisition. In Proc. 41st Annual Meeting of the Assoc. for Computational Linguistics, 2003, pp. 224–231.
13. Yangarber R., Grishman R., Tapanainen P. and Huttunen S. Automatic acquisition of domain knowledge for information extraction. In Proc. 20th Int. Conf. Computational Linguistics, 2000, pp. 940–946.
14. Zhou G., Su J., Zhang J. and Zhang M. Exploring various knowledge in relation extraction. In Proc. 43rd Annual Meeting of the Assoc. for Computational Linguistics, 2005, pp. 427–434.

Information Filtering

Christian Fluhr
CEA LIST, Fontenay-aux, Roses, France

Synonyms

SDI, Selective dissemination of information; push transactions; IF

Definition

Information retrieval (IR) and information filtering (IF) are strongly related [3,5]. Information retrieval indexes a large set of documents and when a user asks a query, answers are extracted out of this set. Information filtering processes a stream of documents and for each document arriving in the system a comparison is made with one or more filtering profiles provided by users and in case of a match the document is sent to the user who created the profile. Applications of information filtering are found in competitive intelligence or technology watch. Another way of filtering is to send the document to the user only if the match is negative. This is useful for applications such as child protection or anti-spam.

Another difference is that IR queries are short, for immediate use, with answers expected as if in a conversational mode (in less than few seconds). For Information filtering, queries are permanent, can be elaborated using a long process, are often quite lengthy sometimes including relevant example documents and must be refined using feedback over time.

Historical Background

The first systems of information filtering, then called selective dissemination of information (SDI), appeared in the early 1960s [4]. For a long time they were used principally by librarians to route journal articles, using fixed profiles, to users interested in a particular domain. Because of the time lag then existing between publication, manual indexing, and bibliographic database updating, this kind of service gave delayed access to sometimes stale information.

Newswires about current events was then and remains a good source of fresh information. Companies, eager for information about their competitors, and financiers, interested in having early information about interesting companies, are large consumers of newswire.

The internet has enlarged the use of information filtering because both the number of users and the quantity of information has increased as more information becomes accessible. News filtering remains a major activity, but new activities (such as the filtering of spam and child protection in accessing internet sites) have appeared.

Another new filtering activity brought by the internet is the possibility to advertise new product availability according to user interest, even if this interest is only implicitly observed on by their browsing behavior or search engine queries. This is called *push advertising*.

Foundations

Organization of a filtering system

A filtering system (Fig. 1) processes a stream of documents. Each entering document is compared to a stored expression of user need constructed using key words, possibly limited to relevant structural parts of the input documents. If the document is considered relevant for this profile, the document is sent to the user. The following figure illustrates this process:

One might say that Information Retrieval compares a unique query with several documents whereas Information Filtering compares a unique document to several queries.

In Information Retrieval, a user generates a spontaneous, simple query and desires a rapid response. On the contrary, in Information Filtering, because the subject of the search is permanent, the user has time to elaborate and refine the query, called rather a *profile* here. The search process is continuous. It is not a conversational process though the user still wants to be notified as early as possible about relevant incoming information.

In practice, search engines providers propose a filtering service which is not based on a real filtering system but they use their search engine by periodically submitting the profile query to the database limiting the answers to documents that have arrived since the last run. Of course this is a simulation that does not guarantee the fastest access time to information but this is sufficient for many users.

Profile building and evolution

For Boolean systems, a profile can be a very long and complex equation of keywords that can have hundreds of keywords and logical operators. Generally the elaboration of such queries is performed by professional librarians.

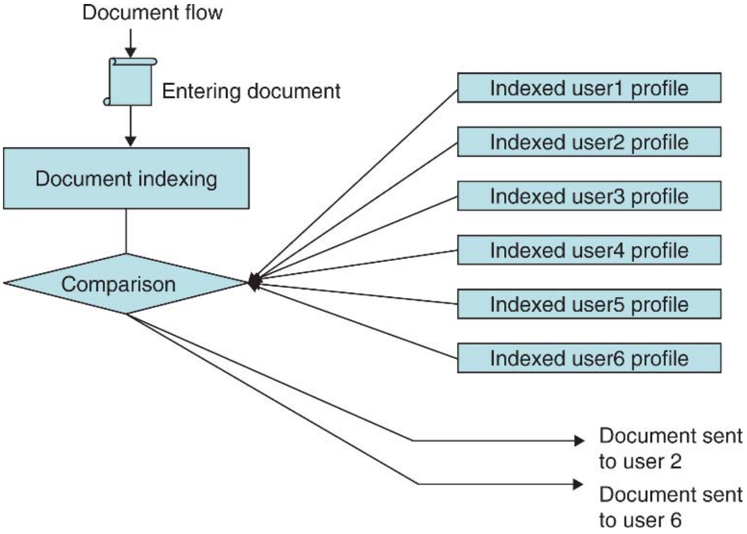
The best way to define a profile is to give a list of keywords along with some document passages relevant for the profile. Such profiles must be processed by systems that can compute a semantic proximity between the profile and the incoming document to decide relevance.

Even if the user's long term information need is stable, the profile expressing this need varies over time. At first, the initial description of user need is often incomplete. Verifying the filtering behavior of an initial set of filtered documents, the user can fine-tune, adding additional keywords, and try to refine ambiguities that he had not foreseen. This process can be simplified for the user by supervised relevance feedback. In this setup, the user need only provide positive or negative reactions to a limited number of proposed documents. According to this positive or negative judgment, the profile can be adjusted by addition or deletion of words, or by modification of their weights in the comparison.

A second reason for profile evolution over time is due to a clearer understanding of the problem on the part of the user, after viewing of retrieved documents. This learning on the part of the user can lead him to rebuild a modified profile.

Comparison procedure

Most filtering systems use the standard information retrieval approach of comparing the vector representing the document with the vector representing the profile. Because the filtering result must be a



Information Filtering. Figure 1. Organization of a filtering system

yes-or-no choice, it is necessary to create a threshold of relevance. A first difficulty is choosing a good threshold. In a filtering system, there is no database as documents arrive in a stream, so document frequency for a term is not known. This can be overcome by computing document frequencies on the fly without producing a database.

The process of attribution of a document to a profile can also be assimilated to a categorization process which assigns documents to one of a set of classes. Key words and document samples car be considered as representative of a category, addition of new relevant documents increase data that can be used to learn the difference between the two classes "good for this profile" and "bad for this profile". Popular categorization methods like Rocchio [6], KNN (K Nearest Neighbors) [9] and SVM (Support Vector Machine) [2] can be used to perform this learning.

As users are interested in a precise domain, the document as an information unit can be too large as a response. Technologies for passage retrieval can be usefully applied to solve this problem. Documents are then cut into parts that are semantically homogeneous. These separate parts are compared to profiles instead of the full document.

Relevance feedback

Because at the beginning for filtering, little information is provided for the profile, it is necessary to improve the profile through relevance feedback. The reaction of the user accepting or rejecting proposed documents gives implicit information about positive or negative influence of the words contained in these documents (Weights of terms in the original query are adjusted and new words are also added to provide a new vector representing the adjusted profile) [1].

Named entities in filtering

Named entities are proper nouns representing names of persons, of organizations, of places, of products, or numerical information like dates, measures, and percentages. They are particularly important for filtering, especially in the case of competitive intelligence.

Thematic filtering versus event filtering

According to the user need, some profiles are more domain oriented and comparison between words vectors are a good solution. Other needs are more event-oriented and necessitate more language processing to identify events that are characterized by a particular action with a particular class of actors. Examples of such event-oriented filtering can be nomination or resigning of company managers, terrorist acts, purchase of a company by another, etc.

Evaluation of filtering systems

Since the beginning of the TREC (Text REtrieval Conference) in 1992, Information Filtering was considered for evaluation. A training corpus was given to participants with queries to train their system. Afterward a new corpus was given to participants and the previously supplied queries were then applied on this new corpus. Participants had to give an ordered list of relevant documents (1000) according to the computed relevance between documents and each query.

This was far from real usage of filtering systems. For this reason, from 1998 to 2002, a new evaluation paradigm was established to evaluate filtering systems [7]. The first difference was that only very few documents are used for training. The participants then had to make a binary decision (document relevant or not) rather than perform a ranking. In addition, documents had to be treated one after the other and for each document proposed as relevant a simulated feedback could be performed. This simulated feedback enabled the tested system to increase their effectiveness. Such process is called adaptive filtering.

Since then, new measures have been established. These measures attempt to take into consideration the different types of users. Some users are interested in having only relevant documents even if they miss many other relevant ones. Other users want to be sure to get all relevant documents but will also accept noise in the form of non relevant documents.

Because of the binary decision about relevance, it is not possible to apply measures developed for answers consisting of ranked list of ordered documents according a level of relevance such as used in the ad hoc track of TREC.

Linear utility: This measure assigns a positive or negative cost to elements in the following contingency table.

	Relevant	Not relevant
Retrieved	R+/A	N+/B
Not retrieved	R–/C	N–/D

R+, R−, N+, N− are the number of document in each category

A, B, C, D give the relative cost for each category.

For example, in TREC 11, the linear utility measure uses the following values $A = 2, B = -1, C = 0, D = 0$

F-beta: This is a modification of the classical F-measure [8]. It combines recall and precision with a parameter beta b which gives a relative weighting between recall and precision.

$0 <= \beta <= 1$

F-measure = $((1 + \beta) * \text{precision} * \text{recall}) / ((\beta * \text{precision}) + \text{recall})$

$\beta = 1$ is neutral, in TREC 2001 $\beta = 0{,}50$ which is an advantage given to precision

Key Applications

Information Filtering is a tool which is used for technology watch and competitive intelligence. It is used both in the security domain and in the civil economy. Companies need to know as early as possible what the activity of their competitors and their potential partners is, in order to decide their future on the technical level and also on the commercial and strategic level.

The other growing need for Information Filtering is the elimination of unsolicited information by filtering spam in the incoming mails. It is also applied to prevent access to some sites (pornographic, pedophile, racist; . . .) for child users of internet.

Cross-references

► Categorization

► Relevance Feedback

Recommended Reading

1. Allan J. Incremental relevance feedback for information filtering. In Proc. 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1996, pp. 18–22.
2. Alsaffar A.H., Deogun J., and Sever H. 1Optimal queries in information filtering, In Proc. 12th Int. Symp. Foundations Intelligent Syst., 2000, pp. 435–443.
3. Belkin N.J. and Croft W.B. Information filtering and information retrieval: two sides of the same coin? Commun. ACM, 35 (12):29–38, December 1992.
4. Brandenberg W., Fallon H.C., Hensley C.B., Savage T.R., and Sowarby A.J. The SDI-2 system. IBM advance system Development Division report, 17–031, Yorktown-Heights, NY, April 1961.
5. Hanani U., Shapira B., and Shoval P. Information filtering: overview of issues, research and systems. User Model. User-adapt. Interact. 11:203–259, 2001.
6. Rocchio J.J. *1*The SMART retrieval system: experiments in automatic document processing, Chapter XIV. Relevance feedback in information retrieval, G. Salton (ed.). Prentice-Hall, NJ, 1971, pp. 313–323.
7. Soboroff I. Robertson S. Building a filtering test collection for TREC 2002, In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 243–250.
8. Van Rijsbergen K. Information retrieval, (2nd ed.). Butterworths, London, 1979.
9. Yang Y., An evaluation of statistical approach to text categorization, report CMU-CS-97-127, Carnegie Mellon University, 1997.

Information Foraging

Peter Pirolli
Palo Alto Research Center, Palo Alto, CA, USA

Synonyms

Information seeking; Browsing

Definition

Information foraging theory [9] provides scientific predictions and explanations of information-seeking behavior in human-computer interaction (HCI). It also provides engineering models for designing and evaluating information systems and their user interfaces. Information foraging theory assumes that people adapt their information-seeking behavior to maximize their rate of gaining useful information to meet their ongoing goals. Specific models of human-information interaction are developed through (i) rational analysis of the structure and constraints of the task environment and information environment that constitute the ultimate forces driving information-seeking behavior and (ii) specification of cognitive models (user knowledge and cognitive processes) that constitute the psychological machinery behind the observed information-seeking behavior of users. Rational analysis draws upon theories and mathematical approaches developed to address rational choice under uncertainty, and specifically draws upon many models developed to address food-foraging strategies in optimal foraging theory (a branch of behavioral ecology in biology). Cognitive models are developed as programs in computational cognitive architectures that simulate human perception, action, thinking, memory, and other aspects of human information processing.

Historical Background

Information Foraging Theory arose during the 1990s, coinciding with an explosion in the amount of information that became available to the average computer user, and with the development of new technologies for accessing and interacting with information. The late 1980s witnessed several strands of HCI research that were devoted to ameliorating problems of exploring and finding electronically stored information. The confluence of increased computing power, storage, networking, and information access and hypermedia research in the late 1980s set the stage for the widespread deployment of hypermedia in the form of the World Wide Web in 1993. The emergence of the Web in the 1990s provided new challenges and opportunities for HCI. The increased wealth of accessible content, and the use of the Web as a place to do business, exacerbated the need to improve the user experience on the Web and other large-scale collections of content.

Models of information foraging were initially developed by analogy to optimal foraging theory. A typical optimal foraging model characterizes an agent's interaction with the environment as an optimal solution to the tradeoff of costs of finding, choosing, and consuming food against the energetic benefit gained from that food. Information foraging theory, however, assumes that costs of interaction are weighed against the benefits gained from gathering information and knowledge. These models attempt to understand the design of an agent's behavior by assuming that it is well engineered (adapted) for the problems posed by the environment. The two optimal foraging models that were initially applied to information seeking and browsing were the diet model (prediction of the food types that an animal selects to eat or not) and the patch model (prediction of when an animal chooses to give up food seeking in a food patch to go to another). The information diet model was applied to the prediction of how users would select some information collections and ignore others in order to maximize the benefits gained in information browsing. The information patch model was used to predict when users would give up on search in an information collection (e.g., a Web site). The first detailed application of the information diet and patch models was used to evaluate a novel document clustering browser called Scatter/Gather [10,11]. Detailed predictions of individual user behavior with Scatter/Gather were derived from computer simulations (cognitive models) of user perceptions, cognition, and action. A novel cognitive modeling technique introduced by information foraging theory predicted how users assessed cues from the user interface to make information seeking and browsing choices. *Information scent* refers to these user interface cues that guide users to the information they seek. Information foraging theory has been used to develop models of information-seeking on the Web, and interaction with information visualizations. It has also been used as a foundation for design guidelines, novel user interfaces, and automated usability evaluation tools.

Foundations

It is assumed that a user's fitness is improved to the extent that the user can gain valuable information to solve problems faced in everyday life. Increasing the rate at which people can find, make sense of, and use valuable information improves the human capacity to behave intelligently. It is assumed that adaptive information systems evolve towards states that maximize gains of valuable information per unit cost.

Users engaged in information foraging will exhibit such adaptive tendencies, and they will prefer technologies that tend to maximize the value (utility) of knowledge gained per unit cost of interaction.

Information Foraging Theory has adopted the *rational analysis* program [1]. Rational analysis addresses the questions: (i) *what* environmental problem is solved, (ii) *why* is a given behavioral strategy a good solution to the problem, and (iii) *how* is that solution realized by cognitive mechanism. The products of this approach include (i) characterizations of the relevant goals and environment, (ii) mathematical rational choice models (e.g., optimization models) of idealized behavioral strategies for achieving those goals in that environment, and (iii) computational cognitive models. Rational analysis in information foraging theory focuses on the task environment that is the aim of performance, the information environment that structures access to valuable knowledge, and the adaptive fit of the human-information interaction system to the demands of these environments. The following recipe has been proposed for rational analysis [1].

1. Precisely specify the goals of the agent.
2. Develop a formal model of the environment to which the agents is adapted.
3. Make minimal assumptions about the computational costs.
4. Derive the optimal behavior of the agent considering (1–3).
5. Test the optimality predictions against data.
6. Iterate.

The recipe focuses on optimal behavior under given goals and environmental constraints with minimal assumptions about the mechanisms that might produce such behavior.

To illustrate, rational analyses of information foraging on the Web have focused on the (i) the choice of the most cost-effective and useful browsing actions to take based on the relation of a user's information need to the perceived cues (information scent) associated with Web links and (ii) the decision of whether to continue at a Web site or leave based on ongoing assessments of the site's potential usefulness and costs. The rational analysis of information scent assumes that the goal of the information forager is to use perceived information scent cues (e.g., a Web link) to predict the utility of desired sources of content (i.e., a Web page that provides a needed answer), and to choose to navigate the links having the maximum expected utility. It is minimally assumed that the user's cognitive system represents information scent cues and information goals in cognitive structures called *chunks*. A Bayesian analysis leads to a specification of *strengths* of association, S_{ji}, among chunks that reflects the log likelihood odds of an information source associated with information cue *j* being relevant to a goal chunk *i*:

$$S_{ji} = \log\left(\frac{\Pr(i|j)}{\Pr(i)}\right), \tag{1}$$

where $\Pr(i|j)$ is the probability (based on past experience) that chunk *i* has occurred when chunk *j* has occurred in the environment, and $\Pr(i)$ is the base rate probability of chunk *i* occurring in the environment. Equation (1) is also known as *Pointwise Mutual Information.* Each chunk *i* in the user's goal is assumed to receive *activation.* A_i, from all associated information scent chunks *j*,

$$A_i = \sum_j S_{ji}, \tag{2}$$

and the total amount of activation received by all goal chunks is,

$$V = \sum_i A_i. \tag{3}$$

It is assumed that the utility of choosing a particular link is just the sum of activation it receives plus random noise drawn from independently and identically distributed (IID) Gumbel distributions. The probability that a user will choose link *L*, having a summed activation V_L, from a set of links *C* on a Web page, given an information goal, *G* is,

$$\Pr(L|G, C) = \frac{e^{\mu V_L}}{\sum_{k \in C} e^{\mu V_k}}. \tag{4}$$

where μ is a scaling parameter. Another choice facing a Web user is whether to continue navigating a particular Web site or leave. The rational analysis of this problem employs a modified patch model from optimal foraging theory. It is assumed that the user employs learning mechanisms to develop an assessment of the potential yield of a Web site, based on the user's current experiential state *x*. This *potential function h*(x) is

$$h(x) = U(x) - C(t). \tag{5}$$

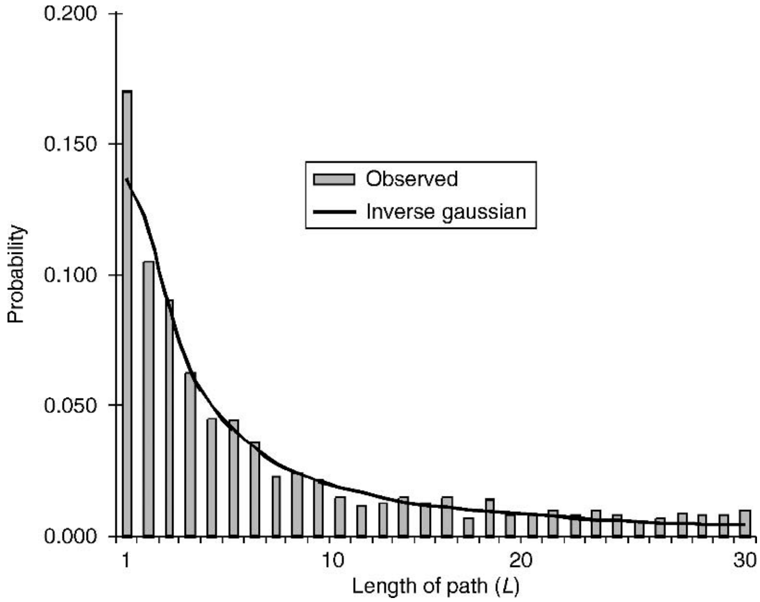
where *U*(x) is the utility of continued foraging in the current Web site and *C*(*t*) is the *opportunity cost* of foraging for the *t* amount of time that is expected to be spent in the information patch. So long as the potential of the Web site is positive (the utility of continuing is greater than the opportunity cost) then the user will continue foraging.

Individual information foraging behavior can be related to the behavior of aggregates of users. For instance, the *Law of Surfing* [6] relates the rational analysis of individual Web surfing to aggregate Web user behavior. Such aggregate distributions are of practical interest because content providers on the Web often want to know how long people will remain at their Web sites. The Law of Surfing characterizes the distribution of the length, *L*, of sequences of page visits by Web users. Figure 1 presents a typical empirical distribution of the length of paths taken by visitors to a Web site. The distribution is skewed with the bulk of path lengths being short, and a long positive tail. The skewness of the distribution, and the long positive tail, imply that the mean of the distribution will typically be larger than the mode.

The Law of Surfing assumes that the expected utility from continuing on to the next state, X_t, is stochastically related to the expected utility of the current state $X_{t\text{-}1}$,

$$U(X_t) = U(X_{t-1}) + \varepsilon_t, \tag{6}$$

where the ε_t are IID Gaussian distributions with mean *m* and variance s^2. This is known as a *Wiener process* (a Brownian motion process) with a random drift parameter *m* and noise s^2. It is assumed that the process continues until a threshold expected utility is reached. That is, from an initial starting page in a Web foraging episode, the expectation is that users continue browsing (surfing) according to the Wiener process specified in (6) until some threshold θ is reached. It is assumed that an individual will continue to surf until the expected cost of continuing is perceived to be larger than the discounted expected value of the information to be found in the future. In the limit, this analysis of Web surfing is the same as the analysis of *first passage times* in Brownian motion. First passage times are distributed as an Inverse Gaussian



Information Foraging. Figure 1. The Law of Surfing: The probability distribution of the length of paths surfed by users prior to leaving a web site are approximated by an inverse Gaussian distribution.

distribution: the probability density function of L, the length of sequences of Web page visits, is distributed is

$$f(L) = \sqrt{\frac{\lambda}{2\pi} L^{-3/2} e^{\frac{\lambda}{2v^2 L}(L-v)^2}}, \quad L > 0, \qquad (7)$$

where the parameter v is the expected value, and λ is related to the expected value and variance as

$$\frac{\lambda = v^3}{Var[L]}.$$

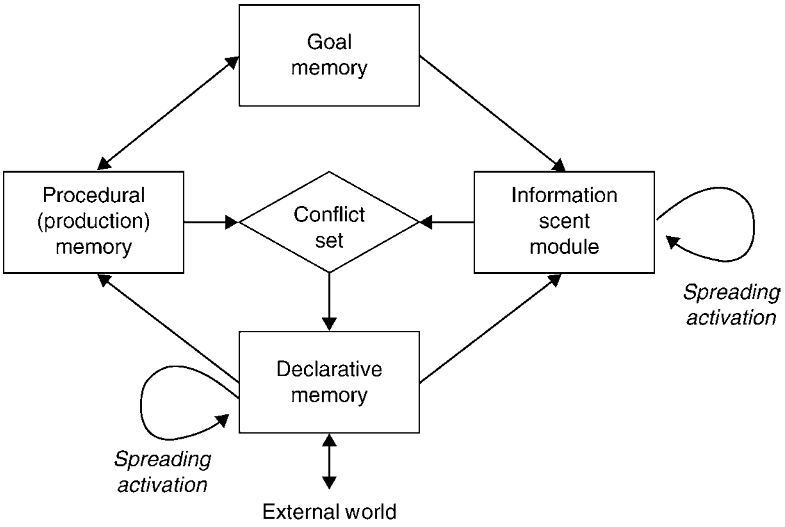
Computational cognitive models developed in information foraging theory simulate the psychological mechanisms that yield adaptive behavior. Typically, these are implemented in a simulation system that embodies a theory of *cognitive architecture* developed in psychology [2]. Theories of cognitive architecture attempt to provide a deeper account of the mechanisms underlying cognition, learning, and performance. The ACT family of *production system* theories has the longest history (since 1976) of these kinds of cognitive architectures.

Figure 2 presents the ACT-Scent cognitive architecture developed in the ACT family. It includes a module that computes *information scent*. The architecture includes a *declarative memory* containing declarative knowledge represented as chunks, which correspond to things that the user's mind is aware it knows and that can be easily described to others, such as the content of Web links, or the functionality of browser buttons, and the current user's goal (e.g., evaluating a link, choosing a link, etc.). The architecture also includes a *procedural memory*. Procedural knowledge is represented as *production rules*. For instance, the following is an English gloss of a production rule, Click-link, that is used to simulate the choice of a Web link,

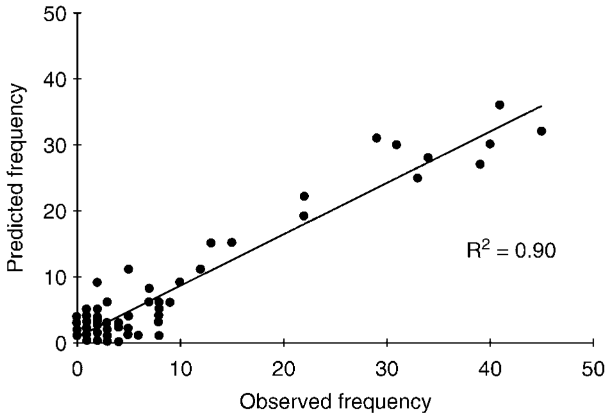
```
Click-link:
IF the goal is to process a link
& there is a information-seeking task
& there is a browser
& there is a link that has been read
& the link has a link description
THEN
Click on the link
```

If selected, the rule will execute the action of clicking on the link. A production rule has a *condition* side and an *action* side. When the all the conditions on the condition side (the "IF" portion) are *matched* to goal information and declarative memory, the production may be *fired* and when it does, the actions (the "THEN" portion) of the production will be *executed* to possibly update goals and memory, or initiate behavior. At any point in time, only a single production can fire. When there is more than one match, the matching productions form a *conflict set*. One production is then selected from the conflict set based on its utility based on information scent. The *goal memory* contains representations of intentions driving behavior. The information scent module computes the utility of actions.

Information perceived from the external world is encoded into chunks in declarative memory. Goals



Information Foraging. Figure 2. The ACT-Scent cognitive architecture used in simulating web foraging.



Information Foraging. Figure 3. The scatter plot for the observed and theoretically predicted frequency that users select links at the Yahoo! web site assessed over eight information-seeking tasks. Predictions were generated by Monte Carlo simulations using an ACT-Scent model.

and subgoals controlling the flow of cognitive behavior are stored in goal memory. The system matches production rules in production memory against goals and activated information in declarative memory and those that match form a conflict set. The matched rule instantiations in the conflict set are evaluated by utility computations in the information scent module. Based on the utility evaluation, a single production rule instantiation is executed, updates are made to goal memory and declarative memory, if necessary, and the cycle begins again. Simulations models generate predicted user behavior. Figure 3 presents the predicted frequencies for links being chosen at a popular Web site generated by Monte Carlo simulations of users working on pre-specified tasks.

Key Applications

Engineering Models of Browser Use

Graph-based algorithms based on information foraging models have been used to predict the flow of user at a Web site and identify Web site navigation problems, and this was implemented in a automated usability analysis tools available through a Web interface [5]. Dynamic programming models based on information foraging theory have been used to identify trade-offs in the design of the Scatter/Gather document cluster browser that varied over task conditions. The CogTool-Explorer [8] tool supports the rapid prototyping of user interfaces and then automatically simulates user interaction with specified designs to make performance time predictions.

Novel Search and Browsing Interfaces

Novel user interface techniques have been developed to automatically render improved information scent cues that yield more efficient user browser [7,14].

Usability Guidelines

Concepts and metaphors from information foraging theory have been influential in the development of Web usability guidelines. The concept of information scent has been used to develop Web page design guidelines aimed at producing efficient matches between user goals and Web link cues [13].

Future Directions

Information foraging theory has been extended to predictions of user interaction with highly interactive information visualizations [12,4], programming environments, and skimming and reading time allocation under varying deadline conditions. Recent work is also extending the research to social information foraging (information seeking and production by collections of users) in systems such as social bookmarking sites [9,4].

Cross-references

- ▶ Browsing
- ▶ Browsing in Digital Libraries
- ▶ Information Navigation
- ▶ Information Retrieval

► Navigation
► Searching Digital Libraries
► Usability

Recommended Reading

1. Anderson J.R. The Adaptive Character of Thought. Lawrence Erlbaum, Hillsdale, NJ, 1990.
2. Anderson J.R., Bothell D., Byrne M.D., Douglass S., Lebiere C., and Qin Y. An integrated theory of mind. Psychol. Rev., 11 (4):1036–1060, 2004.
3. Budiu R., Pirolli P., Fleetwood M., and Heiser J. Navigation in degree of interest trees. In Proc. Working Conf. on Advanced Visual Interfaces, 2006, pp. 457–462.
4. Chi E.H., Pirolli P., and Lam S.K. Aspects of augmented social cognition: social information foraging and social search. In Human Computer Interaction International, D. Schuler (ed.). Springer, 2007, pp. 60–69.
5. Chi E.H., Rosien A., Suppattanasiri G., Williams A., Royer C., Chow C., Robles E., Dalal B., Chen J., and Cousins S. The bloodhound project: automating discovery of web usability issues using the InfoScent simulator. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2003, pp. 505–512.
6. Huberman B.A., Pirolli P., Pitkow J., and Lukose R.J. Strong regularities in World Wide Web surfing. Science, 280 (5360):95–97, 1998.
7. Olston C. and Chi E.H. ScentTrails: integrating browsing and searching on the Web. ACM Trans. Comput. Hum. Interact., 10(3):177–197, 2003.
8. Pirolli P. Exploring browser design trade-offs using a dynamical model of optimal information foraging. In Proc. SIGCHI Conf. on Human Factors in Computing Systems., 1998, pp. 33–40.
9. Pirolli P. Information Foraging: A Theory of Adaptive Interaction with Information. Oxford University Press, New York, NY, 2007.
10. Pirolli P. and Card S.K. Information foraging in information access environments. In Proc. SIGCHI Conf. on Human Factors in Computing Systems., 1995, pp. 51–58.
11. Pirolli P. and Card S.K. Information foraging. Psychol. Rev., 106:643–675, 1999.
12. Pirolli P., Card S.K., and Van Der Wege M.M. The effects of information scent on visual search in the Hyperbolic Tree Browser. ACM Trans. Comput. Hum. Interact., 10(1):20–53, 2003.
13. Spool J.M., Perfetti C., and Brittan D. Designing for the Scent of Information. User Interface Engineering, Middleton, MA, 2004.
14. Woodruff A., Rosenholtz R., Morrison J.B., Faulring A., and Pirolli P. A comparison of the use of text summaries, plain thumbnails, and enhanced thumbnails for web search tasks. J. Am. Soc. Inf. Sci. Technol., 53:172–185, 2002.

Information Graphic

► Chart

Information Hiding

► Steganography

Information Integration

Alon Halevy
Google Inc., Mountain View, CA, USA

Synonyms

Data integration; Enterprise information integration

Definition

Information integration systems offer uniform access to a set of autonomous and heterogeneous data sources. Sources can range from database systems and legacy systems to forms on the Web, web services and flat files. The data in the sources need not be completely structured as in relational databases. The number of sources in an information integration application can range from a handful to thousands.

Historical Background

Database applications are typically heavily designed and tuned for a specific context. But as data management needs in enterprises change and the information economy evolves, the need to combine information from multiple sources arises frequently. Examples of such scenarios include mergers and acquisitions, internal restructuring, the need to interoperate with third parties and to expose data on the web. Since the early 1980s the need to combine multiple heterogeneous data sources has become a research topic for the database research community. In the 1990s, as the web emerged and the many data sources came on line, the need to integrate data and for companies to work with third-parties grew, and these trends fueled the information integration research and industry.

Foundations

Information integration is a challenging problem for three different classes of reasons: systems-level, logical-level and social challenges.

The systems challenges arise fundamentally because enterprises need to enable multiple systems to talk seamlessly to each other. This is hard even when they are all running ODBC/JDBC compliant databases, let alone radically different systems. For example, while SQL is a standard query language for relational databases, there are some differences in the way different vendors implement it. Executing queries efficiently over multiple systems is even more challenging. In addition to the difficulties that arise in distributed databases, information integration systems cannot assume that data is A priori distributed to the different nodes by one entity and in an organized and known fashion. Furthermore, the query processing capabilities of each source can be very different. For example, while one source may be a full SQL engine and therefore may be able to accept very complex queries, another source may be a web-form and therefore only accepts a very small number of query templates.

The second set of challenges has to do with the way data is logically organized in the data sources. At the core, the problem is that when schemas are designed by different people and for different applications, there will be significant differences between them, even when they model the same domain. Differences include (i) naming of tables and attributes, (ii) tabular organization of the data (or hierarchical structure in XML), (iii) domain coverage and level of detail modeled, and (iv) differences in data-level representation of objects.

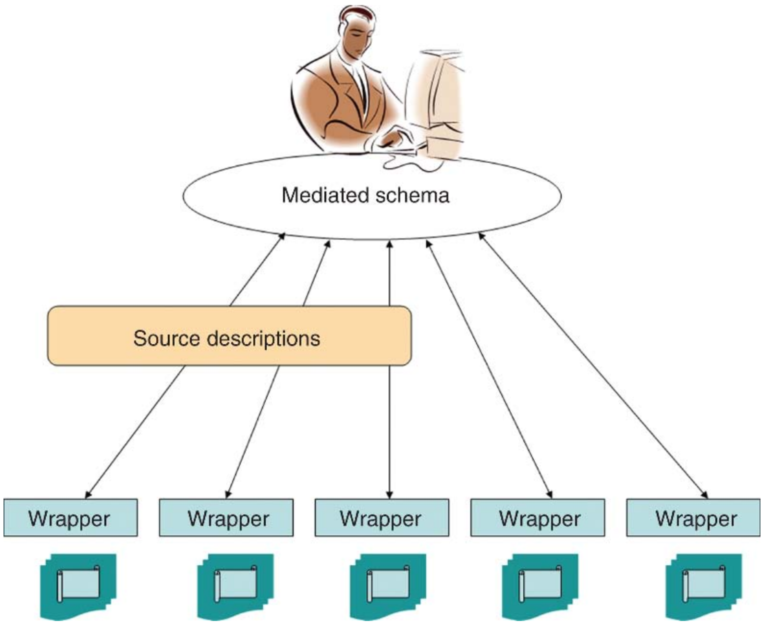
The social challenges in building an information integration application, while not technical, are often very significant in practice. The first challenge may be to *find* a particular set of data. It may be hard to find a particular piece of data within a large enterprise, and in some cases, the data needed is not captured in the proper form. Even when the location of the data is known, owners of the data may not want to cooperate with an integration effort. In some cases this may happen for competitive reasons, and in others it may be because systems are carefully tuned for performance and the owners are afraid that participating in an information integration effort may adversely affect their system. Of course, privacy and security concerns are rampant.

For all these reasons, the goal of information integration is to build tools that make it *easier* to build integration applications, rather than completely automating the process.

Information Integration Architecture

Figure 1 shows a prototypical architecture of an information integration system, often referred to as *virtual information integration*. The *data sources* are shown on the bottom of the figure. As explained earlier, data sources can vary on many dimensions, such as the data model underlying them, their schema, and their ability to process queries.

The *wrappers* are (hopefully small) programs whose role is to send queries to a data source, receive



Information Integration. Figure 1. Logical components of a virtual information integration system.

answers and possibly apply some basic transformations on the answer. For example, a wrapper to a web form source would accept a query and translate into the appropriate HTTP request with a URL that poses the query on the source. When the answer comes back in the form of an HTML file, the wrapper would extract the tuples from the HTML file. There are two main approaches to creating wrappers. The first approach is to explicitly write a set of rules for extracting the structure from the answer returned from the source. The second approach is to train the wrapper on a set of examples, and use Machine Learning techniques to learn the rules.

The top of the figure depicts the *mediated schema*. This is the schema in which users (or applications) pose queries. The mediated schema is built for the information integration application and contains *only* the aspects of the domain that are relevant to the application. Therefore, it does not necessarily contain all the attributes present in the sources, but only a subset of them. The mediated schema is not meant to store any data, but is purely a logical schema.

The key to building an information integration application are the *source descriptions*. These descriptions specify the properties of the sources that the system needs to know in order to use their data. The main component of source descriptions are the *semantic mappings*, that relate the schemata of the data sources to the mediated schema. The semantic mappings specify how attributes in the sources correspond to attributes in the mediated schema (when such correspondences exist), and how the different groupings of attributes into tables are resolved. In addition, the semantic mappings specify how to resolve differences in how data values are specified in different sources. It is important to emphasize that the virtual information integration architecture only requires specifying mappings between the data sources and the mediated schema and not between every pair of data sources. Hence, the number of mappings that need to be specified is the same as the number of sources and not the square of that number.

Schema Mediation Languages

There has been quite a bit of research on developing appropriate languages for specifying schema mappings. The common theme to these languages is that they use query expressions for the mappings, and they differ in how they use these expressions.

The Global-as-View (GAV) language describes the mediated schema as a set of view definitions over the source relations. In contrast, the Local-as-View language (LAV) describes the data sources as views over the mediated schema. The GLAV language combines the expressive power of GAV and LAV. The following example illustrates these differences.

Example: Consider a mediated schema with the following three relations: `Movie(title, director, year, genre), Actors(title, name) Plays (movie, location, startTime)` *and the sources shown in* Figure 2.

The mappings shown in Figure 3 are GAV schema mappings. Note how Movie is defined to be the union of two conjunctive queries.

The $\subseteq$ *symbol in the mapping denotes the* open-world assumption*: that is, the data sources may not contain all the data in the domain.*

The mappings shown in Figure 4 are LAV schema mappings. The first description states that S5 *contains a subset of the projection on a relation in the mediated*

S1:
Movie(MID, title)
Actor(AID, firstName, lastName, nationality, yearOfBirth)
ActorPlays(AID, MID)
MovieDetail(MID, director, genre, year)
S4: ActDir(actor, director)
S5: MovieGenres(title, genre)
S6: MovieDirectors(title, dir)
S7: MovieYears(title, year)

Information Integration. Figure 2.

Movie(title, director, year, genre) $\subseteq$ S1.Movie(MID, title),
S1.MovieDetail(MID, director, genre, year)
Movie(title, director, year, genre) $\subseteq$ S5.MovieGenres(title, genre),
S6.MovieDirectors(title, director),
S7.MovieYears(title, year)

Information Integration. Figure 3.

S5.MovieGenres(title, genre) $\subseteq$ Movie(title, director, year, genre)
S4.ActDir(actor, dir) $\subseteq$ Movie(title, director, year, genre),
Actors(title, actor)

Information Integration. Figure 4.

schema, and the second description states that S4 *is a subset of the join of two relations in the mediated schema. Neither of these sources can be described in GAV.*

As shown, the main advantage of GAV is that query processing is conceptually easier. The main benefit of LAV over GAV is that describing a data source does not require knowing which other data sources exist in the system. Hence, it is easier to add more sources to the system. Furthermore, since the source descriptions in LAV can leverage the expressive power of the view definition language, it was easier to describe precise constraints on the contents of the sources and describe sources that have different relational structures than the mediated schema. Describing such constraints is crucial because it enables the system to select a minimal number of data sources relevant to a particular query.

User queries are posed in terms of the relations in the mediated schema. Hence, the first step the system must do is *reformulate* the query into queries that refer to the schemas of the data sources. To do so, the system uses the source descriptions. The result of the reformulation is a set of queries that refer to the schemata of the data sources and whose combination will yield the answer to the original query. The result of reformulation is referred to as a *logical query plan.*

Hence, an important aspect of schema mapping languages is developing algorithms for (and understanding the complexity of) query reformulation. For GAV, since the mediated schema is defined as a set of views over the sources, query reformulation amounts to query unfolding. In the case of LAV, since sources are described as views, query reformulation amounts to rewriting queries using views. While, in general, the complexity of answering queries using views is exponential, there are several algorithms that work efficiently in practice. Query reformulation in GLAV is a combination of answering queries using views followed by a query unfolding step.

In addition to schema mappings, source descriptions also need to consider (i) access-pattern limitations to sources: for example, a database served behind a web form or web service typically requires a set of inputs to serve tuples, and (ii) completeness (or partial completeness) of data sources: when a data source is known to be complete on a particular slice of the domain, the system can eliminate the need to access other data sources. The differences between complete and incomplete sources are formulated in terms of the open-world (closed-world) assumption.

Generating Schema Mappings

A major bottleneck in setting up an information integration application is the effort requires to create the source descriptions, and more specifically, writing the semantic mappings between the sources and the mediated schema. Writing such mappings (and maintaining them) required database expertise (to express them in a formal language) and business knowledge (to understand the meaning of the schemas being mapped).

To address this problem, several techniques for semi-automatic schema mapping have been developed with the goal of reducing the time it takes a human to create mappings. These techniques rely on several principles. First, the techniques explore methods to map between schemas based on clues that can be obtained from the schemas themselves, such as linguistic similarities between schema elements and overlaps in data values or data types of columns. Second, based on the observation that none of the above techniques is foolproof, the next development involved systems that combined a set of individual techniques to create mappings. Finally, based on the observation that schema mapping tasks are often repetitive, novel schema mapping methods incorporate Machine Learning techniques that enable the system to leverage past work. It should also be noted that the process of generating schema mappings is typically divided into two steps. In the first step, referred to as *schema matching*, the system generate *correspondences* between attributes (or other schema elements) in the two schemas. In the second step, the system builds on the correspondences and creates expressions in the mapping language.

In addition to the schema level, an information integration system also needs to reconcile references at the data level. There are often cases where the same object in the world is referenced in different ways in data sets (e.g., people, addresses, company names, genes). The problem of reference reconciliation is to automatically detect references to the same object and to collapse them. Unlike reconciling schema heterogeneity, the amounts of data are typically much bigger, and therefore these techniques need to put more emphasis on being mostly automatic. These techniques typically rely on some sophisticated forms of string matching, augmented by looking at neighboring values in the same row of a table to gather additional match clues.

Information integration is one of several contexts in which mappings between data sources need to be created, manipulated, composed and inverted. The area of Model Management was developed to provide a formal framework for supporting generic operations on schemas and mappings between them. Knowledge Representation languages, and in particular, Description Logics, have also been shown to offer benefits in modeling data sources in a flexible fashion. In a sense, Description Logics offer a schema and query language that can also express very rich constraints, but still support effective reasoning. More generally, information integration has been an active topic also in the field of Artificial Intelligence, leveraging techniques from Knowledge Representation, Machine Learning and Automated Planning.

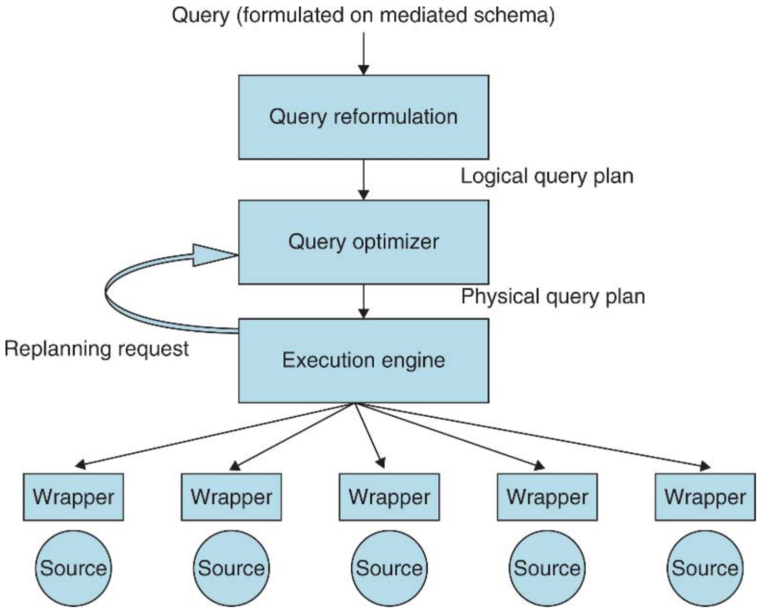
Query Processing

Given a logical query plan, it needs to be optimized and executed by the system (see Fig. 5). Herein lies the second main difference between database systems and information integration systems. Unlike the conventional database setting, an information integration system cannot neatly divide its processing into a query optimization step followed by a query execution step. The contexts in which an information integration system operate are very dynamic and the optimizer has much less information (e.g., statistics) than the traditional setting. As a result, two things happen: (i) the optimizer may not have enough information to decide on a good plan, and (ii) a plan that looks good at optimization time may be arbitrarily bad if the sources do not respond exactly as expected. To address these challenges, researchers developed techniques for *adaptive query processing*. (Adaptive query processing was already investigated in traditional database systems because even there, the estimates of the optimizer may be wrong, leading to bad plans).

The key idea of adaptive query processing is that the engine may decide to change the query plan *during* execution. The variations on the techniques developed have to do with how and when the plan can be changed. For example, one strategy may consider changing the plan only at materialization points, or the system may put conditionals into the plan that examine the intermediate results and decide what to do next. More aggressive techniques consider a different plan for every single tuple, while others continuously monitor the execution and change plans when it appears as if there is a better global plan. One of the challenges that each of these techniques needs to keep in mind is to minimize wasted computation (i.e., intermediate results that are computed and then discarded).

Related Data Management Architectures

Virtual information integration is one of several architectures for sharing and integrating data. Before the development of virtual information integration, data



Information Integration. Figure 5. Components of an information integration system.

warehousing was the common method for integrating data from multiple sources. However, data warehousing suffered from the fact that the data is loaded only periodically into the warehouse and therefore may be stale. Furthermore, data warehousing requires a single physical store thereby limiting the range of contexts in which data can be shared.

The emergence of peer-to-peer file sharing systems inspired the data management research community to consider P2P architectures for data sharing. The advantage of peer-data management systems is that it is no longer necessary to create a single mediated schema in cases where such a schema is hard to build or agree upon.

Finally, data exchange refers to an architecture that includes source and target databases, and the semantic mappings specify how to populate the target from data in the source. Many of the same issues encountered with GLAV mappings also occur here.

The Information Integration Industry

Beginning in the middle 1990s, information integration moved from the lab into the commercial arena. Today, this industry is known as Enterprise Information Integration (EII). The vision underlying this industry is to provide tools for integrating data from multiple sources *without* having to first load all the data into a central warehouse as required by previous solutions. Broadly speaking, the architectures underlying the products were based on the principles investigated in the research arena.

Some of the first applications in which these systems were fielded successfully were customer-relationship management, where the challenge was to provide the customer-facing worker a *global view* of a customer whose data is residing in multiple sources, and digital dashboards that required tracking information from multiple sources in real time. Of the many challenges faced by the industry, perhaps the greatest one was the business question of whether to build a horizontal platform that can be used in any application or to build special tools for a particular vertical. The argument for the vertical approach was that customers care about solving their *entire* problem, rather than paying for yet another piece of the solution and having to worry about how it integrates with other pieces. In addition, there are challenges in integrating with other middleware tools, such as Enterprise Application Integration tools.

In addition to the enterprise market, information integration has also played an important role in internet search. For example, the *vertical search* market focuses on creating specialized search engines that integrate data from multiple deep web sources in specific domains (e.g., travel, jobs). These engines also embed complex source descriptions.

Finally, information integration has also been a significant focus in the life sciences, where diverse data is being produced at increasing rates, and progress depends on researchers' ability to synthesize data from multiple sources. Personal Information Management is also an application where information integration is taking a significant role.

Key Applications

Some of the key applications of information integration are:

1. Enterprise data management, querying across several enterprise data repositories,
2. Accessing multiple data sources on the web (and in particular, the deep web),
3. Large scientific projects where multiple scientists are independently producing data sets,
4. Coordination accross mulitple government agencies.

Future Directions

With all the progress to date, the set-up time for information integration systems is still too long. To set up an information integration application, one still needs to create a mediated schema, and create semantic mappings to obtain any visibility into the data sources. An important research challenge is to provide as many services as possible with as little set up time as possible. The management of *dataspaces* emphasizes the idea of pay-as-you-go data management: offer some services immediately without any setup time, and improve the services as more investment is made into creating semantic relationships.

To support pay-as-you-go data management, the information integration system needs to model and reason about uncertainty. Uncertainty can appear in several forms: in the underlying data, the imprecise nature of semantic mappings and vaguely specified queries (i.e., keyword queries). Hence, incorporating uncertainty into data management and in particular, to information integration systems, is an important challenge going forward.

Cross-references

- ▶ Adaptive Query Processing
- ▶ Model Management
- ▶ Query Rewriting Using Views
- ▶ Query Translation
- ▶ View-Based Data Integration
- ▶ XML Information Integration

Recommended Reading

1. Deshpande A., Ives Z., and Raman V. Adaptive query processing. Foundations and Trends in Databases. Now Publishers, 2007. http://www.nowpublishers.com/dbs.
2. Franklin M., Halevy A., and Maier D. Dataspaces: a new abstraction for data management. ACM SIGMOD Rec. 34(4):27–33, December, 2005.
3. Haas L. Beauty and the beast: The theory and practice of information integration. In Proc. 11th Int. Conf. on Database Theory, 2007, pp. 28–43.
4. Halevy A.Y. Answering queries using views: a survey. VLDB J., 10(4), 2001.
5. Halevy A.Y., Ashish N., Bitton D., Carey M.J., Draper D., Pollock J., Rosenthal A., and Sikka V. Enterprise information integration: successes, challenges and controversies. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 778–787.
6. Lenzerini M. Data integration: a theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
7. Rahm E. and Bernstein P.A. A survey of approaches to automatic schema matching. VLDB J., 10(4):334–350, 2001.

Information Integration Techniques for Scientific Data

Amarnath Gupta
University of California San Diego, La Jolla, CA, USA

Synonyms

Graph theory; Graph data structure; Graph database

Definition

Information integration refers to the field of study of techniques attempting to combine information from disparate sources despite differing conceptual, contextual and lexical representations. One goal of information integration, commonly held by the data management community, is to combine the information in such a way that the user gets a unified view of the data. In other words, the user should see and query the data as though it is present in a common, unified schema.

In the domain of scientific applications, the problems and expectations are different. In this domain the semantics of data play a very strong role in data integration, and semantic compatibility need to be ensured as part of the data integration process. Further, straightforward view-based data integration, which works well for commercial applications, does not always suit the needs of the scientific users. Finally, scientists need to ensure that the result of any query on integrated data is scientifically valid.

Historical Background

Database applications are typically heavily designed and tuned for a specific context. But as data management needs in enterprises change and the information economy evolves, the need to combine information from multiple sources arises frequently. Examples of such scenarios include mergers and acquisitions, internal restructuring, the need to interoperate with third parties and to expose data on the web. Hence, from the early 1980s the need to combine multiple heterogeneous data sources has become a research topic for the database research community. In the 1990s, as the web emerged and the many data sources came on line, the need to integrate data and for companies to work with third-parties grew, and these trends fueled the information integration research and industry.

Foundations

There are several technical challenges in information integration in science.

Semantic Heterogeneity

Semantic heterogeneity [6] refers to a problem that occurs in both scientific and commercial applications and arises when the data to be integrated have been developed by different groups for different purposes. An aspect of semantic heterogeneity is schema heterogeneity (one schema uses two attributes “firstName” and “lastName,” and another uses a single attribute “fullName”); another aspect is data heterogeneity (“IBM” in one database versus “International Business Machines” in another). A more complex kind of semantic heterogeneity involves generalization, where one database has a value like “thoracic surgeon” and another has a value “surgeon.” This difference can be reconciled if the system uses the additional knowledge that a "thoracic surgeon is a surgeon," and have an integration rule that a narrower term can be mapped to the more general term. In scientific databases, the reconciliation between schema and data heterogeneities becomes more complex. The term "Purkinje cell" can be mapped to the more general term "neuron" if the anatomical location of the Purkinje cell is known to be the brain, it should map to "muscle cell" if the location is known to be the heart, otherwise it will map to the even more general term "cell." Similar mapping rules can be defined at the schema level. Thus, one needs a more extensive, yet easily computable, mapping logic, and an integration framework that uses this extra mapping layer. Hakimpour and Geppert [5] and Ludäscher et al. [7] make use of ontologies or other semantic structures to specify these semantic mappings.

Role of Metadata in Managing Semantic Heterogeneity

Metadata refers to additional information about data that cover the scope, the creation, the coverage and so forth that is not directly represented in the data. In science, the norm is to call the experimental output as the "data" and other related information like the experimental protocol, the people conducting the experiment and recording the data etc. as metadata. Metadata can play a partial role in resolving some forms of semantic heterogeneity in both commercial and scientific applications. Suppose there are two tables in two sources where each contains a variable called "oxygen-saturation." Is it possible to create integrated views that would perform unions of the tables, or a join operation with "oxygen-saturation" as the joining column? Since they both have the same data types and notionally represent the same physical quantity, it might seem that combining the relations using this variable is a valid operation. However, if it is known that one database has a daily value of this variable, while the other database has monthly averages, then the answer does not remain so obvious any more. One role of metadata is to capture this kind of information such that the information integration rules can use additional guard conditions based on metadata.

Functional Relationships

Functions play a significant role in scientific applications. The data exposed to a user is often the result of a computation. For example, in many geospatial applications, the value of a variable at a specific location is not directly stored and needs to be provided through interpolation. This is one of the many typical cases where the user can get access to the data only through functions [2]. In an integration scenario, the mapping between two data elements coming from two different data sources may also be established through functions. Suppose there are two sources S1 and S2, and both provide a variable Y when a parameter X is supplied to them. When an integration system accesses S1, it has to use the function f1, and for S2 it has to use the function f2. Suppose, the integration system takes a value x1 and sends it to both sources by executing f1(x1) and f2(x1), and gets back values y1 and y2 respectively. The relationship between y1 and y2 may need to be determined by mapping functions g1 and g2 to be applied to the two sources respectively. Thus if the values returned are equivalent then g1(f1(x1)) will be equivalent to g2(f2(x1)). If the sources need to be integrated through a view that requires a semijoin, transformation functions are needed to convert the output of one source to input of another source. In actual scientific applications that involve wider variety of data types, these functions can be complicated and specialized and need a separate service. While information integration products like DB2 DiscoveryLink from IBM allow sources to export functions, optimized techniques for data integration with functions is still an area of research.

Non-schematic Integration

Not all scientific data integration can be done by schema integration, or even schema integration extended by semantic mapping structures. This is especially true if the task of the "integration" is to compute a cascade of functions whose parameters come from different sources. This form of information integration is covered in detail in the entry *Scientific Workflow Management*. A different class of non-schematic information integration occurs through statistical tools. Critchlow et al. [3] reviews a number of "data integration" tools in the domain of drug discovery. All these two use multiple data sources from domain specific data types like microarrays and protein interaction graphs, and use statistical structures like decision trees and Bayesian networks to achieve data combination. How data management techniques can be effectively used for this class of data integration remains an open issue.

New Issues

Interactive Information Integration The ability to visualize information is common to many scientific disciplines. Scientists often find the fully automated schema mapping and integration techniques developed by computer scientists to be less useful because they would like the data integration process to be more interactive and exploratory. In this mode of integration, the user needs a system's guidance to discover sources that may potentially be integrated, would use exploratory tools to investigate which subset of data from different sources might scientifically qualify for integration, and provide additional semantic input at query time to ensure compatibility of data that need to match each other. Research projects are currently underway to explore how interactivity can be introduced in information integration and querying.

Quality of Integration

A common goal in any science task is to develop a better understanding of a phenomenon by observing, modeling and computation. In the case of information integration, the desired goal is to ensure that when two previously unconnected pieces of information are combined, the resulting product of integration will provide a better or more accurate or novel understanding that the unconnected data pieces could not have provided by themselves. A real impediment in achieving this is that scientific data often has uncertainties and suffers from obsolescence due to the progressive nature of scientific information. Further, data from different sources can be contradictory or incompatible and combining them may need additional techniques that a data integration system needs to support. A formal framework is therefore needed to assess the "goodness" of the integration. Qi et al. [8] has taken an initial step for integrating conflicting data in an archeological application.

Key Applications

A number of scientific information integration systems have developed over the last decade. A few of them are listed below.

BIRN

The Biomedical Informatics Research Network (http://www.nbirn.net) has developed a semantic information integration system for Neuroscience applications. It uses a relational Global-As-View Mediator [4] extended with a semantic network of inter-term relationships to achieve scientific information integration.

Data Foundry

The Data Foundry system (https://computation.llnl.gov/casc/datafoundry) [3] uses a mediator based system for integration of Bioinformatics data. The system contains a modeling language to represent metadata which is used to generate a specific mediator for an information integration task.

GEON

In the domain of geological sciences, the Geosciences Network (http://www.geongrid.org/) [1] uses all information resources including maps and spatial information to be explicitly registered to an OWL (Web Ontology Language – see http://www.w3.org/TR/owl-features/) ontology. The ontology-registered data sources are then queried using an SQL called SOQL.

Cross-references

▶ Ontologies and Life Science Data Management
▶ Scientific Workflows

Recommended Reading

1. Bowers S., Lin K., and Ludaescher B. On integrating scientific resources through semantic registration. In Proc. 16th Int. Conf. on Scientific and Statistical Database Management, 2004, p. 349.
2. Cluet S., Delobel C., Siméon J., and Smaga K. Your mediators need data conversion! In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 177–188.
3. Critchlow T., Fidelis K., Ganesh M., Musick R., and Slezak T. Datafoundry: information management for scientific data. IEEE Trans. Inf. Technol. Biomed., 4(1):52–57, 2000.
4. Gupta A., Ludäscher B., Martone M.E., Rajasekar A., Ross E., Qian X., Santini S., He H., Zaslavsky I. BIRN-M: a semantic mediator for solving real-world neuroscience problems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, p. 678.
5. Hakimpour F. and Geppert A. Resolving semantic heterogeneity in schema integration. In Proc. Int. Conf. on Formal Ontology in Information System, 2001, pp. 297–308.
6. Halevy A. Why your data won't mix. ACM Queue, 3(8):50–58, 2003.
7. Ludäscher B., Gupta A., and Martone M.E. Model-based mediation with domain maps. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 81–90.
8. Qi Y., Candan K.S., Sapino M.L., and Kintigh K.W. Integrating and querying taxonomies with quest in the presence of conflicts. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1153–1155.
9. Searls D.B. Data integration: challenges for drug discovery. Nature Rev. Drug Discov., 4:45–58, 2005.

Information Lifecycle Management

Hiroshi Yoshida
Fujitsu Limited, Yokohama, Japan

Synonyms

ILM

Definition

Information lifecycle management is the activity of storing data in the most appropriate storage infrastructure, according to the business value of that data as it changes over time. The Storage Networking Industry Association (SNIA) defines Information Lifecycle management as follows:

The policies, processes, practices, services and tools used to align the business value of information with the most appropriate and cost-effective infrastructure from the time when information is created through its final disposition. Information is aligned with business requirements through management policies and service levels associated with applications, metadata and data.

Key Points

The business value of information varies during its lifecycle, e.g., from the time when information is created through to its final disposition. Considering the varying business value of information, it is necessary to provide cost-effective storage infrastructure which meets the requirements corresponding to the business value. For example, information related to current transactions is most valuable from the viewpoint of current business operations and is usually stored in online disk arrays with high performance and high availability. Information on transactions during the past year has value as statistics and is stored in less expensive hard disk drives. Finally, information on transactions over several years has value only in meeting regulatory compliance and is stored in offline tape libraries.

Information lifecycle management is the generalization of such practices. In a typical example of ILM process, information is classified based on business values first, referring to its metadata or content. Then appropriate service level requirements are defined for information based on its classification. On the other hand, storage infrastructure resources are also classified based on service level attributes such as performance and availability. Finally the most cost-effective storage infrastructure resources, which meet the classified information requirements, are chosen. When information value changes, the information is migrated between multiple storage infrastructure classes according to predefined policies.

A simple example of information lifecycle management is hierarchical storage management (HSM). In HSM, files are migrated between multiple storage tiers such as high performance disk arrays, low performance hard disks, and tape libraries based on access frequency or latest access date. Instead of such simplistic policies, supporting policies which reflect the change of business value of information is the key to implementing ILM.

Cross-references

- ▶ DAS
- ▶ SAN
- ▶ SRM
- ▶ Storage Consolidation
- ▶ Storage Network Architectures
- ▶ Storage Networking Industry Association
- ▶ Storage Protocols
- ▶ Storage Virtualization

Recommended Reading

1. Storage Network Industry Association. Information Lifecycle Management Initiative, 2007. Available at: http://www.snia.org/forums/dmf/programs/ilmi
2. Storage Network Industry Association. Storage Network Industry Association tutorials, 2007. Available at: http://www.snia.org/education/tutorials/

Information Loss Measures

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona, Catalonia

Synonyms

Data utility measures

Definition

Defining what a generic information loss measure is can be a tricky issue. Roughly speaking, it should capture the amount of information loss for a reasonable range of data uses. It will be said that there is little information loss if the protected dataset is analytically valid and interesting according to the following definitions by Winkler [4]:

A protected microdata set is *analytically valid* if it approximately preserves the following with respect to the original data (some conditions apply only to continuous attributes):

1. Means and covariances on a small set of subdomains (subsets of records and/or attributes)
2. Marginal values for a few tabulations of the data
3. At least one distributional characteristic

A microdata set is *analytically interesting* if six attributes on important subdomains are provided that can be validly analyzed.

More precise conditions of analytical validity and analytical interest cannot be stated without taking specific data uses into account. As imprecise as they may be, the above definitions suggest some possible measures:

- Compare raw records in the original and the protected dataset. The more similar the SDC method to the identity function, the less the impact (but the higher the disclosure risk!). This requires pairing records in the original dataset and records in the protected dataset. For masking methods, each record in the protected dataset is naturally paired to the record in the original dataset it originates from. For synthetic protected datasets, pairing is less obvious.
- Compare some statistics computed on the original and the protected datasets. The above definitions list some statistics which should be preserved as much as possible by an SDC method.

Key Points

A strict evaluation of information loss must be based on the data uses to be supported by the protected data. The greater the differences between the results obtained on original and protected data for those uses, the higher the loss of information. However, very often microdata protection cannot be performed in a data use specific manner, for the following reasons:

- Potential data uses are very diverse and it may be even hard to identify them all at the moment of data release by the data protector.
- Even if all data uses could be identified, releasing several versions of the same original dataset so that the *i*-th version has an information loss optimized for the *i*-th data use may result in unexpected disclosure.

Since datasets must often be protected with no specific data use in mind, generic information loss measures are desirable to guide the data protector in assessing how much harm is being inflicted to the data by a particular SDC technique.

Information loss measures for numerical data. Assume a microdata set with n individuals (records) $I_1, I_2,...,I_n$ and p continuous attributes $Z_1, Z_2,...,Z_p$. Let X be the matrix representing the original microdata set (rows are records and columns are attributes). Let X' be the matrix representing the protected microdata set. The following tools are useful to characterize the information contained in the dataset:

- Covariance matrices V (on X) and V' (on X').
- Correlation matrices R and R'.
- Correlation matrices RF and RF' between the p attributes and the p factors $PC_1,...,PC_p$ obtained through principal components analysis.
- Communality between each of the p attributes and the first principal component PC_1 (or other principal components PCi's). Communality is the percentage of each attribute that is explained by PC_1 (or PC_i). Let C be the vector of communalities for X and C' the corresponding vector for X.
- Factor score coefficient matrices F and F'. Matrix F contains the factors that should multiply each attribute in X to obtain its projection on each principal component. F' is the corresponding matrix for X'.

There does not seem to be a single quantitative measure which completely reflects those structural differences. Therefore, it was proposed in Domingo-Ferrer, Mateo-Sanz and Torra [1] and Domingo-Ferrer and Torra [2] to measure information loss through the discrepancies between matrices X, V, R, RF, C and F obtained on the original data and the corresponding X', V', R', RF', C' and F' obtained on the protected dataset. In particular, discrepancy between correlations is related to the information loss for data uses such as regressions and cross tabulations.

Matrix discrepancy can be measured in at least three ways:

- *Mean square error.* Sum of squared component wise differences between pairs of matrices, divided by the number of cells in either matrix.
- *Mean absolute error.* Sum of absolute component wise differences between pairs of matrices, divided by the number of cells in either matrix.

- *Mean variation.* Sum of absolute percent variation of components in the matrix computed on protected data with respect to components in the matrix computed on original data, divided by the number of cells in either matrix. This approach has the advantage of not being affected by scale changes of attributes.

For alternative [0,1]-bounded information loss measures for numerical data, see Mateo-Sanz et al. [3].

Information loss measures for categorical data. These can be based on direct comparison of categorical values, comparison of contingency tables, on Shannon's entropy. See Domingo-Ferrer and Torra [2] for more details.

Cross-references

▶ Disclosure Risk
▶ Inference Control in Statistical Databases
▶ Microdata
▶ SDC Score

Recommended Reading

1. Domingo-Ferrer J., Mateo-Sanz J.M., and Torra V. Comparing SDC methods for microdata on the basis of information loss and disclosure risk. In Pre-proceedings of ETK-NTTS'2001, vol. 2, 2001, pp. 807–826.
2. Domingo-Ferrer J. and Torra V. Disclosure protection methods and information loss for microdata. In Confidentiality, Disclosure and Data Access: Theory and Practical Applications for Statistical Agencies, P. Doyle J.I. Lane J.J.M. Theeuwes L. Zayatz (eds.). Elsevier, Amsterdam, 2001, pp. 91–110.
3. Mateo-Sanz J.M., Domingo-Ferrer J. and Sebé F. Probabilistic information loss measures in confidentiality protection of continuous microdata. Data Mining Knowl. Discov., 11(2):181–193, 2005.
4. Winkler W.E. Re-identification methods for evaluating the confidentiality of analytically valid microdata. Res. Off. Stat., 1(2):50–69, 1998.

Information Navigation

Thomas Rist
University of Applied Sciences, Augsburg, Germany

Synonyms

Information foraging; Information browsing; Interactive information exploration

Definition

The word "navigate" has its roots in the two Latin words "navis" (ship) and "agere" (to "move"/"direct") and has been used for centuries in the domain of nautics for activities necessary to find one's way, and to control the movement of a vessel while traveling from one location to another. Different scientific discoveries, technical inventions, and cultural backgrounds, have brought about a wide variety of navigation techniques and navigation devices, all of them have in common that they assume a certain structuring of the underlying geographical space. With the advent of electronically stored volumes of information, and electronically networked information sources in particular, the concept of geographical navigation has been generalized and adopted as a metaphor for accessing chunks of digitalized information in a goal-directed way. In this metaphorical view perceivable presentations of meaningful information chunks of an overall information space with form the counterparts of physical locations in a geographical space, and information navigation capitalizes on the fact that structural relations between physical locations, such as neighborhood, proximity, distance, connectedness, reachability, or crossway can be mapped onto meaningful relations between information chunks as well. Depending on nature of an application, a user is confronted with different navigation tasks. A typical task in a database application is to find the shortest access path from an entry point to a certain data record, in an e-Learning system, the navigation task may manifest itself in accessing the learning units in a didactically meaningful sequential order, while in an online bookstore, a user may navigate through the assortment of books following the traces of other customers with a similar interest profile. In any case, goal-directedness and exploration of structure distinguishes navigation activities from arbitrary, disoriented movements, be it in a geographical space, an information space or a combination of both in which an information space is superimposed on a geographical space.

Key Points

With the increasing amount of accessible electronic data, information navigation has become a central issue in many application domains and a challenge for architects and developers of user interfaces as aids to assist users in accomplishing various navigation tasks. Navigation tasks are often described relative to an navigation space that is characterized by the set of potentially accessible information units, and the structural relationships that hold between these units. The structure of an navigation space determines in which order information units can be accessed and thus the effort it takes to navigate from an entry point to a particular unit. The spectrum of commonly used principles includes linear, hierarchical/tree-like, radial, and graph-like structuring of navigation spaces. The structuring of a navigation space should, however, comply with a reasonable, semantically-motivated structuring of the information units of the application domain.

An effective navigation aide supports a user in at least two basic navigation tasks: *orientation* – "where am I," and *path continuation* – "where to go from here to get (closer) to the target?" Techniques to support keeping orientation include: sitemaps and distorted views – such as "*fisheye views*" – both designed to show a user's location in a broader context, or navigation histories, such as "*bread-crump navigation*" as today used by many web-sites to show a user her trail from a chosen entry-point to her current location. Supporting users in the decision making of how to continue a navigation path from a current location/information unit comprises the disclosure of available and relevant options, as well as de-emphasizing or even hiding of options that should not be taken with regard of achieving the underlying navigation objective.

Among the supporting techniques are techniques that aim to provide environmental cues about where to find further interesting information – the "*information scent*" approach is an exemplar of such a technique. Also, the concept of "adaptive menus" falls into this category of techniques. It has been introduced with the aim to facilitate a user's navigation through a menu-based interface by hiding options which are regarded less relevant in a certain context. In practice, however, it is often too hard a task for a computer system to make appropriate assumptions about the user's current tasks and aims and therefore inadequate suggestions for the continuation of a navigation may be given. Also, accessibility, comprehensibility and effectiveness of navigation structures of web-sites have become major criteria in most web-usability evaluations.

Earlier work on information navigation has mainly focused on graphical user interfaces (GUIs) and hypermedia-style information access with view-based navigation aids, assuming a user navigates through an information space by traversing interlinked visual presentations. Research includes the development of visual navigation aids as well as the understanding of the cognitive processes involved in conducting navigation tasks (e.g., [1]).

More recently, navigation issues are being researched in other areas too, including voice-dialogue systems, virtual reality applications, and computer games. In the emerging fields of ubiquitous and mobile computing the mutual interplay of geo-spatial navigation with information navigation provides interesting potential for new services that benefit mobile users. The advent of large-scale internet shops and forums has shed light on yet another aspect of navigation – social navigation, a term that has been coined for a model of navigation support that capitalizes on observed navigation patterns of others [2].

Cross-references

▶ Hypermedia

▶ Visual Data Exploration

Recommended Reading

1. Furnas G.W. Effective view navigation. In Proc. SIGCHI Conf. on Human Factors in Computing Systems., 1997, pp. 367–374.
2. Höök K., Munro A., and Benyon D. Designing Information Spaces: The Social Navigation Approach. Springer, Berlin, 2002.

Information Quality: Managing Information as a Product

DIANE M. STRONG
Worcester Polytechnic Institute, Worcester, MA, USA

Definition

Traditionally, information has been viewed as a by-product of a computer system or an event. From this viewpoint, the focus is on designing and delivering computer systems, rather than designing and delivering information. To increase the quality of information available to information consumers, organizations need to treat information as a product being intentionally produced for those who will use that information. This means actively managing information and its quality. Such an information product (IP) approach focuses attention on information quality, i.e., delivering high-quality information that is fit for use by information consumers, rather than solely on data quality, i.e., maintaining the quality of the data stored in databases or data warehouses. While the terms "data" and "information" are often used interchangeably in the information and data quality literature, the term information quality is more often used in studies that take an IP approach and explicitly acknowledge the needs of information consumers, whereas data quality is more often used when the quality of data is assessed separately from the needs of users.

Historical Background

While there are some early seminal papers, information quality (IQ) only emerged as a recognized field of information technology (IT) research in the mid-1990s. For example, the first International Conference on Information Quality was held in 1996 at MIT [8]. In 1998, Wang et al. [14] published "Manage your Information as a Product", which argued for the concept of treating information as a product that is intentionally managed and produced for consumers who have expectations about its quality. This concept provided researchers with a useful lens for framing their information quality research ideas and led to many studies developing methods, measures, techniques, and recommendations for managing information as a product. Together, these methods, measures, techniques, and recommendations for managing information as a product are referred to as the information product (IP) approach.

Foundations

Managing information as a product, i.e., taking an information product (IP) approach, means actively managing organizational information in order to deliver high-quality information that is fit for use by information consumers [14]. The IP approach involves four principles for managing information as a product, with four corresponding areas of information quality research. These four areas of research, which are described below, are (i) the needs of information consumers, (ii) processes for producing high-quality information products, (iii) the life cycle of information products, and (iv) organizational governance structures and recommendations for actively managing information as a product.

The Needs of Information Consumers

Like other products, the quality of information is judged by those who purchase and use information products, referred to as information consumers. Thus, information is of high-quality to the extent that it is fit for use by information consumers. This means that information quality includes both objective dimensions (those dimensions that can be defined independently of a user or a task), such as accuracy, and subjective or contextual dimensions (those dimensions that are defined relative to the context of a user performing some task that requires that information), such as timeliness. For example, whether information is sufficiently timely depends on the task being performed, e.g., stock prices for a historical analysis or for day trading. Even those dimensions that can be objectively defined may still have a subjective or contextual aspect, e.g., whether the information is sufficiently accurate for the task versus whether it is completely accurate.

Research on the needs of information consumers has used marketing research techniques to understand the dimensions or characteristics of information products that consumers assess when considering an IP for their use. The result is sixteen dimensions that information consumers consider when assessing the quality of information, which are grouped into four categories, see Table 1 [15]. The intrinsic dimensions are those that capture the quality of the information itself. The contextual dimensions are those that capture quality in the context of the task and the users. The accessibility dimensions capture the quality of the process of obtaining and manipulating the information. The representational dimensions capture quality related to how the information is represented

Information Quality: Managing Information as a Product. Table 1. Information quality categories and dimensions

IQ category	IQ dimensions
Intrinsic IQ	Free-of-error, Objectivity, Believability, Reputation
Accessibility IQ	Accessibility, Ease of Manipulation, Security
Contextual IQ	Relevancy, Value-Added, Timeliness, Completeness, Amount of information
Representational IQ	Interpretability, Ease of understanding, Concise representation, Consistent representation

to users. These dimensions have been verified in organizational contexts [11,12] and by numerous studies that have used these dimensions or some subset of them in IQ studies.

The Information Quality Assessment (IQA) questionnaire measures information quality along these sixteen IQ dimensions [6]. Each dimension is represented by several questions for which questionnaire respondents rate the quality of their information on a scale of zero (not at all) to ten (completely). For example, two questions for the free-of-error dimension are “This information is correct” and “This information is accurate.” This questionnaire has demonstrated reliability and validity and has been used in research studies and in organizations seeking to improve their IQ. One interesting result from baseline IQ measures at several companies is that IT professionals responsible for storing and maintaining the information consistently assess information quality higher than do the information consumers who use that information.

While the IQA provides a comprehensive IQ assessment instrument, it is not the only method for assessing IQ [7]. For some dimensions, it is possible to develop objective measures that do not require data and time from information consumers for assessment. The typical approach is to define a metric on a 0–1 scale such as the following metric for timeliness [1]:

$$\text{Timeliness} = \left\{ \text{Max}\left[\left(\frac{1 - \text{currency}}{\text{shelf-life}} \right), 0 \right] \right\}^{s}$$

The parameter *s* adjusts the sensitivity of the timeliness measure to the ratio of the currency of the information relative to the shelf-life or volatility of that information. When such metrics can be defined, IQ can be assessed independently of users.

The Product and Service Performance Model for Information Quality (PSP/IQ model) groups the sixteen IQ dimensions into four IQ quadrants, sound, dependable, useful, and usable, as shown in Table 2 [3]. These quadrants are derived from the total quality management literature. Specifically, on one dimension, the PSP/IQ model captures aspects of IQ that are related to product quality and those related to service quality. On the other dimension, it distinguishes between conforming to specifications and meeting or exceeding information consumer expectations. These quadrants serve as a better foundation for analyzing and improving quality than the original four categories of IQ in Table 1. Several studies have used the PSP/IQ model, in combination with the IQA instrument, to consolidate the 16 dimensional measures into four overall measures, e.g., [6].

The PSP/IQ model and the IQA instrument are useful tools for benchmarking and analyzing IQ. The resulting measures of IQ provide the foundation for gap analyses [6]. One gap is the difference between an

Information Quality: Managing Information as a Product. Table 2. The PSP/IQ model

	Conforms to specifications	Meets or exceeds consumer expectations
Product quality	*Sound Information*	*Useful Information*
	Dimensions of Sound Information:	Dimensions of Useful Information:
	• Free-of-Error	• Appropriate Amount
	• Concise Representation	• Relevancy
	• Completeness	• Understandability
	• Consistent Representation	• Interpretability
		• Objectivity
Service quality	*Dependable Information*	*Usable Information*
	Dimensions of Dependable Information:	Dimensions of Usable Information:
	• Timeliness	• Believability
	• Security	• Accessibility
		• Ease of Manipulation
		• Reputation
		• Value-added

organization and a benchmark organization, which indicates the possibilities for improvement in that organization. Another gap is the difference between the assessments of IT professionals responsible for the information and the users of that information. When IT professionals rate IQ much higher, there are opportunities for better communication about IQ between information consumers and IT professionals.

The needs of information consumers, the first area of research in the IP approach to IQ, address the principle that the needs of information consumers are the primary purpose of improving information quality. Research in this first area addresses the information product itself and its quality as assessed by those who use the product.

Processes for Producing High-Quality Information

Like physical products, information products are produced by production/manufacturing processes. Like manufacturing processes for physical products, information manufacturing processes will only produce high-quality information if they are designed to do so. Thus, research on processes for producing high-quality information products, the second area of IP research, draws on parallels with manufacturing processes for physical products.

This area of research introduces two IQ roles beyond the information consumer described in the last section. Information collectors (also called information suppliers or providers) collect and provide the raw data for the information manufacturing process. The second role consists of those who perform the tasks of manufacturing the IP, which has been referred to as information custodians (also called information manufacturers or IT professionals).

One useful parallel with manufacturing is to apply the Total Quality Management (TQM) recommendations from quality gurus such as Deming, Crosby, and Juran that have resulted in major improvements in the quality of the products produced from manufacturing processes. For information processes, an IQ improvement cycle, similar to Deming's cycle, has been proposed. Called the Total Data Quality Management (TDQM) cycle, it involves the continuous and repeated application of four steps, define, measure, analyze, and improve IQ [13]. The previous section discussed research results that *defined* the dimensions of IQ, developed *measures* for them, and *analyzed* the gaps between an organization's IQ and benchmark IQ measures.

To *improve* the quality of information products, the processes that produce those information products must be defined, measured, analyzed, and improved. Research efforts in this area have focused on modeling information manufacturing processes in a way that permits analysis of process design alternatives and measurement of the IQ of the information products produced. One such research effort is the Information Manufacturing System (IMS), which is a method for developing a network model of information flow [1]. The model consists of five types of components that capture the steps in an information manufacturing system. These five are (i) the data vendor block represents a source of data, (ii) the processing block represents a step that adds value to the information, (iii) the data storage block represents a database, (iv) the quality block represents a step that enhances the quality of the information for information consumers, and (v) the customer block represents the delivery of an information product to an information consumer. At each step in the network of components, various dimensions of IQ are computed given the quality of data from the data vendor block and the quality effects of each of the processing and quality blocks.

Modeling information manufacturing processes in a way that captures the details needed to compute IQ metrics and to analyze process design alternatives is an active area of current research. One extension to the IMS modeling approach adds new modeling components, e.g., an information system boundary block and an organizational boundary block which capture handoff points at which the quality of information may be affected [10]. It also elaborates, and more formally defines, the data needed to specify each block so that IQ metrics can be computed. A variety of other extensions are also under development, e.g., [9], including software tools that assist with the modeling and metric computation tasks.

The Life Cycle of Information Products

The third principle of managing information as a product is to explicitly manage the life cycle of the IP, similar to how the life cycle of a physical product is managed. For physical products, the product is continually improved and extended, so that the product gradually becomes more useful to customers, more reliable, and less costly to produce, which in turn extends the life of the product. Few organizations treat their information as though it had a life cycle.

Instead they view information as a by-product of a computer system or event, and assume the life cycle of the information is the same as the life-cycle of that computer system or event. Information typically is upgraded or its characteristics changed as information consumers make requests, rather than the proactive management of upgrades typical for physical products.

While IQ researchers and practitioners acknowledge the need for managing an IP's life-cycle, there is only a little research addressing this issue. For example, there is some research addressing the optimal or recommended frequency for updating data. The lack of research on managing the life cycle of an IP may be related to the state of research on the governance structures needed for actively managing information as a product.

Governance Structures for Managing Information as a Product

The fourth area of research for managing information as a product focuses on the appropriate management and governance structures needed to support the IP approach. Wang et al.'s paper on managing information as a product [14] recommended appointing an Information Product Manager (IPM), a position similar to the Brand Manager role that organizations employ to manage products across the multiple functional areas of product development, manufacturing, marketing, and service that must all be coordinated to deliver high-quality products and manage their life cycle. Similarly an IPM would manage one or more IPs across information suppliers, information manufacturers, and information consumers. Thus, this is introducing a fourth information role, that of an information product manager. This role differs from that of a CIO, who typically is only responsible for the information manufacturing role and does not have responsibilities for information suppliers or information consumers. The CIO also must typically focus more on the hardware and software architecture and infrastructure than on the architecture and infrastructure of information. Thus, there is a need for a role that provides the cross-functional management structure needed to deliver high-quality information to information consumers.

The IPM role, however, is only one piece of the broader research area of information governance, management structures and best practices needed to adequately provide an IP approach in organizations. Such best practices also need to include the financial side of the IP approach [4], e.g., cost/benefit analysis of IQ improvements, the value and pricing of information, and the auditing of information manufacturing processes. There are still many opportunities for research and the exploration of best practices in the area of IQ governance.

Key Applications

Organizations are at varying stages of attempting to apply an IP approach to managing their information. Examples from several organizations including both for-profit and not-for-profit organizations illustrate the value of the four principles of the IP approach.

Principle 1: Understand the IP Needs of Information Consumers

As an investment bank examined the needs of its information consumers, it realized that it had both external customers who needed information as well as internal information consumers. For its external customers, it was providing adequate information for single requests, but was failing to consider that these customers interacted with multiple divisions, and thus it was failing to provide coordinated and aggregate information. For internal information consumers, it was treating their information needs as by-products of its events with external customers, and largely failed to consider or meet their needs.

A retail chain that sells eyewear had not examined the information needs of its grinders and thus was not treating the information sent from its opticians in retail outlets to the centralized lens grinders as an information product. Its investigation into the information needs of grinders revealed that many of its external customers' complaints about the quality of their eye products were not due to poor quality grinding but to the quality of the information products sent from the opticians to the grinders. When the opticians understood the grinders' needs as an information product, many of the problems with the quality of the eye products were eliminated.

A company that resells data to retailers worked to understand the information needs of its customers and then developed internal information production processes that could tailor its IPs to individual customers. By its attention to its internal and external information customers, it was able to deliver superior information products. A hospital tailored the IQA instrument to its needs and used it to better understand when it was meeting or failing to meet its information consumers' needs.

Principle 2: Manage Information as a Product of a Well-Defined Production Process

To provide information about customers' accounts across their activities with several divisions, the investment bank maintains a centralized customer account database. The production process for this information, however, was flawed, so that the IP's produced did not meet the needs of information consumers. As a result, information consumers in the various divisions created their own customer account databases with information production processes tailored to their needs. While these individual databases met local needs, the global needs for account information across divisions were not being met. The investment bank redesigned its information production processes to produce higher quality information and information tailored to the divisions, so that local needs were better met using information from the global customer account database.

The hospital that used the IQA instrument has also attended to improving its information production processes. It has used a modified version of the IMS modeling technique to model its information processes [2].

Principle 3: Manage Information as a Product with a Life Cycle

A petrochemical company produced a material safety data sheet (MSDS), an IP, for each of its chemicals. While the process for producing the initial MSDS was excellent, it was not managed as an IP with its own life cycle, but rather was updated as a by-product of making changes to the chemical. New hazards, however, were discovered as the chemical was used, and the company was exposed to legal risks when it did not issue a revised MSDS for newly discovered hazards. Because it treated the MSDS as a by-product of chemical development, it did not have an information production process for tracking new hazards and revising the MSDS. To address this problem, it started to treat the MSDS as an IP with its own life cycle, production process, and governance structure. The investment bank was required to maintain risk profiles for its customers. Like the petrochemical company, it did not treat these risk profiles as an IP, and thus did not have a well-defined process for updating them, exposing it to legal risks.

Principle 4: Develop a Governance Structure to Manage Information Production Processes and the Resulting Information Products

Although it is nearly 10 years since the publication of "Manage Your Information as a Product" [14], organizations are still struggling with finding governance structures appropriate for ensuring that information consumers receive information that is fit for their use. Individuals have been assigned roles similar to an IPM, but comprehensive governance structures are not yet common. For example, the petrochemical company appointed someone to develop and manage a process for keeping each MSDS up-to-date with emerging information about hazards. At the company that resells data, the CIO's role was changed to focus more on information processes and products. The hospital mentioned in principles 1 and 2 not only uses the IQA instrument and develops models of its information production processes, but also has an IQ group whose responsibility is to monitor IQ and develop solutions to IQ problems including developing governance structures. At another hospital, there is an information quality analyst role whose responsibilities include uncovering and investigating IQ problems.

Future Directions

The four research areas and principles of the IP approach were presented in approximate order of research maturity. The needs of information consumers in terms of the dimensions of information quality are well-defined; reliable measures of these dimensions have been developed; and methods for analyzing and improving information quality along these dimensions are available. These dimensions continue to be used in research studies and organizational practices. They provide a complete set from which most information quality research studies and practical IQ improvement projects select a subset that is most relevant for the particular IQ effort being undertaken.

Methods for modeling and improving information manufacturing processes are currently an active area of research. These studies borrow heavily from the manufacturing and TQM literature, but also tailor the application of manufacturing research results to address the different characteristics of information as a product. While this is an active area of current research, there is still much to be done to provide methods and techniques that can easily be used in organizations to improve the quality of IP as delivered to information consumers by improving the information production processes producing those IP.

Both the IP life cycle and the governance of IQ are areas with limited current research, but much potential for research. While organizations know how to manage the development and improvement of physical products, the same is not yet true for information products.

More research is also needed that integrates across two or more of these four research areas. For example, more research is needed that integrates the well-defined IQ measures of the IQ dimensions into studies of information production process improvements, e.g., see [5] as an early example of integrating process measures with the IQ outcome measures. Similarly, IP life cycles should be integrated into information production process models. IQ governance studies are needed that includes regular outcome measures to be used to direct efforts at process analysis and improvement. Governance studies are also needed that will develop methods for costing and justifying investments in IQ improvements. The purpose of these research efforts is to develop theories, methods, tools, and recommendations that will help organizations actively manage their information and thus ensure the delivery of high quality information to information consumers.

Cross-references

- ▶ Data Quality
- ▶ Information Quality
- ▶ Information Quality Assessment
- ▶ Information Quality Decision and Making
- ▶ Information Quality Policy and Strategy
- ▶ Information Quality Problem Solving

Recommended Reading

1. Ballou D.P., Wang R.Y., Pazer H., and Tayi G.K. Modeling information manufacturing systems to determine information product quality. Manage. Sci., 44(4):462–484, 1998.
2. Davidson B., Lee Y.W., and Wang R.Y. Developing data production maps: meeting patient discharge data submission requirements. Int. J. Healthc. Technol. Manag., 6(2):223–240, 2004.
3. Kahn B.K., Strong D.M., and Wang R.Y. Information quality benchmarks: product and service performance. Commun. ACM, 45(4ve):184–192, 2002.
4. Lee Y.W., Pipino L., Funk J., and Wang R.Y. Journey to information quality. MIT Press, Cambridge, MA, 2006.
5. Lee Y.W. and Strong D.M. Knowing-why about data processes and data quality. J. Manage. Inf. Syst., 20(3):13–39, 2004.
6. Lee Y.W., Strong D.M., Kahn B.K., and Wang R.Y. AIMQ: a methodology for information quality assessment. Inf. Manage., 40(2):133–146, 2002.
7. Pipino L.L., Lee Y.W., and Wang R.Y. Data quality assessment. Commun. ACM, 45(4ve):38–46, 2002.
8. Proceedings of the International Conference on Information Quality (ICIQ). (1996 and yearly since then), available at: http://mitiq.mit.edu and at http://mitiq.mit.edu/ICIQ.
9. Scannapieco M., Pernici B., and Pierce E. IP-UML: towards a methodology for quality improvement based on the ip-map framework. In Proc. Int. Conf. on Information Systems, 2002, pp. 279–291.
10. Shankaranarayan G., Ziad M., and Wang R.Y. Managing data quality in dynamic decision environment: an information product approach. J. Database Manage., 14(4):14–32, 2003.
11. Strong D.M., Lee Y.W., and Wang R.Y. Data quality in context. Commun. ACM, 40(5):103–110, 1997.
12. Strong D.M., Lee Y.W., and Wang R.Y. Ten potholes in the road to information quality. IEEE Comput., 30(8):38–46, 1997.
13. Wang R.Y. A product perspective on total data quality management. Commun. ACM, 41(2):58–65, 1998.
14. Wang R.Y., Lee Y.W., Pipino L.L., and Strong D.M. Manage your information as a product. Sloan Manage. Rev., 39(4): 95–105, 1998.
15. Wang R.Y. and Strong D.M. Beyond accuracy: what data quality means to data consumers. J. Manage. Inf. Syst., 12(4):5–34.

Information Quality and Decision Making

INDUSHOBHA N. CHENGALUR-SMITH
University at Albany – SUNY, Albany, NY, USA

Synonyms

Data Quality

Definition

Decision-making is an inexact science, and one of the reasons is the utilization of data of imperfect quality in the decision process. If the user is aware of the caliber of the data and incorporates this information in the decision process, the effectiveness of the process is expected to increase.

Historical Background

A decision can be thought of as a set of actions related to and including the choice of one alternative rather than another. A rational model of decision-making is one in which decision-makers consider all aspects of all alternatives before making a decision. However actual decision-making often falls short of this ideal model because knowledge and experience of the consequence is incomplete, there is limited amount of time to explore all alternatives, and humans do not calculate perfectly. Hansson [8] stated that all decisions are made under uncertainty and he presented four components of great uncertainty: uncertainty of options, uncertainty of the consequences of the options, uncertainty of reliability of information and uncertainty of values. The third category, uncertainty of reliability of information, is the focus of this entry.

The decision process is sensitive to task complexity, time pressure and other contextual factors. Task complexity itself is a function of several variables such as the number of alternatives, the number of attributes, and time pressure. If humans have a processing limit of 7±2 items, as believed, the decision process quickly runs into information overload, even without the addition of data quality information. However, increases in decision complexity are often compensated for by a selective use of information or by shifting to more simplistic strategies. Exactly how this plays out, given the additional data quality information that needs to be evaluated, has not yet been conclusively determined. Thus, there could be plausible arguments for and against the use of data quality information in decision making.

Payne et al. [12] proposed that people adopt a decision strategy on the basis of a cost benefit framework, i.e., individuals compromise between making the best possible decision and minimizing their cognitive effort in making that decision. Decision making strategies could be (1) alternative-based or attribute-based, (2) compensatory or non-compensatory, and (3) consistent or selective. Consider, for instance, a satisficing strategy, where alternatives are considered one at a time, in the order in which they are presented. Each attribute is compared to a pre-determined cut-off level and the first alternative whose attributes all meet the cutoffs is selected and the process is stopped. Such a strategy is alternative-based (multiple attributes about an alternative are considered before the next alternative is processed), non-compensatory (no trade-offs are made among alternatives), selective (all the alternatives are not evaluated using the same amount of information). On the other hand, consider a "weighted additive" strategy that assigns relative importances (or weights) to each attribute and evaluates every alternative by multiplying the value of the attribute by its weight and summing up these weighted values for all attributes. Although it is another alternative based strategy, this is a compensatory strategy because tradeoffs are made among alternatives. It is also a consistent strategy because all the alternatives are evaluated using the same amount of information and the alternative with the highest overall evaluation is chosen.

It has been hypothesized that providing information about the quality of the data attributes used by the decision maker could be beneficial. It could also be argued that such data simply creates an additional burden, both on the data provider and the decision maker, without significant payback in improved decision making. The current research in this arena is focused on trying to determine the conditions under which providing data quality information may be useful to decision makers.

Foundations

Data quality is widely considered to be multi dimensional. Wang and Strong [15] identified 15 dimensions of data quality that they factored into 4 distinct groups: intrinsic, contextual, usability and accessibility and determined that accuracy (used interchangeably with reliability) was the most important attribute. It is expected that not only the type of data quality attribute provided but also the format in which the data quality information is recorded and presented would play a role in the decision process.

Data regarding the quality of stored data can be thought of as one type of metadata. In fact, some database systems already do this indirectly through the inclusion of a date field. The record of the time when the data was collected reflects one component of the quality of the data, namely currency. It is also possible to have information regarding data quality (e.g., accuracy) at the level of the individual data item. In certain situations this is warranted and can be achieved via the use of data tags [14]. Although this would be complex and costly, it may be necessary if the processes generating the data items were erratic and highly volatile. At the other extreme one could have data regarding the quality of an entire file or relational table. This approach would be most appropriate where the entire file was generated by a single, rather stable, process. An intermediate approach would be to record data quality information at the level of each data field. This approach is most appropriate where the data being utilized originated from a number of processes which may have substantially different data quality capabilities. Such an approach is neither as storage-intensive as item-level data nor as generic as file-level data. If it is assumed that the quality of data items in a particular domain is the same, although different domains may have different quality, creating such metadata may not be very labor intensive. Although at this time typical databases do not include the kind of data quality information envisioned, since data about data quality are metadata, it would be logical to include such information in the data dictionary.

Even though the inclusion of data quality information has become technologically feasible, the collection of such information is still a challenging undertaking. Quality characteristics of data may be declared through qualitative means such as a "data source" field or through a quantitative system such as a reliability rating. Clearly, providing data quality information on a two-point ordinal scale is more manageable than providing it on a 100-point continuum. But if the latter option leads to more effective decision-making, then the considerable effort required may be worth it. Thus the availability of information is not enough. It needs to be presented in a form that will promote effective decision making. The type of information regarding the data's quality that would be most helpful to users may depend on the sophistication of the users, but is more likely to be a function of the decision process or strategy.

Empirical evidence on the effective use of information during the decision making process has been mixed. Ben Zur and Breznitz [1] found that people under time pressure place more emphasis on negative information about alternatives. The way information is displayed (i.e., on a verbal or numerical scale) has also been found to affect decision behavior [13]. Even within a numerical format, decision making is sensitive to different displays [9]. Thus both content and style of the metadata could have an impact on the decision process. In the public policy arena, Grether et al. [7] found that consumers ignore less relevant information in complex information environments. On the other hand, Gaeth and Shanteau [6] indicated that expert decisions were adversely affected by irrelevant factors, although this could be overcome by training. Several other studies, including [10], found that increasing the amount of attribute information about alternatives increases subjects' confidence in their decisions but also increases the variability of responses. Anecdotal evidence showed that higher information loads lead to simpler processing strategies and hence more consistent decisions.

Having unequivocal measures of the effectiveness of the decision making process is imperative because of the difficulty of identifying what constitutes a good decision. Ideally a decision aid (such as the inclusion of data quality information) should lead to more accurate results. When there are no correct answers, consistency is a desirable outcome of the decision process. MacGregor et al. [11] define two forms of consistency. One is the tendency to produce the same results when applied by the same user under identical circumstances, and the other is the tendency to produce similar results in the hands of different users. MacGregor et al. [11] found that the more structured the decision aid, the greater the improvement in accuracy and consistency of the resulting decision.

Chengalur-Smith et al. [4] expanded on this concept to measure the impact of providing data quality information on decision processes and defined three key measures of the outcome of the decision:

1. Decision Complacency: Complacency refers to the degree to which data quality information is ignored. If the decision-maker does not change the originally preferred alternative after viewing data quality information, he/she has exhibited decision complacency. Complacency can be regarded as a measure of futility, for it implies that providing data quality information has not impacted the final decision. Obviously low levels of decision complacency are the most desirable outcome.
2. Decision Consensus: Consensus examines the degree to which decision-makers can converge on a decision in the presence of data quality information. Often a number of decision-makers will be involved in the same decision process. Consensus explores the impact of data quality information upon the ability of such groups to maintain the prior degree of agreement concerning the preferred course of action. It is a measure of group response to this new incremental information.
3. Decision Consistency: Consistency is designed to capture agreement across all alternatives, when data quality information is provided. Note that the previous two measures, complacency and consensus, focused on only the top-ranked or preferred alternative. In a sense, consistency is an extension of complacency to the entire set of alternatives. Once again a high value for this measure would indicate that major re-orderings in the ranks did not occur.

Note that complacency (not changing the originally preferred alternative in the presence of data quality information) and consensus (ability to converge on an alternative using data quality) deal only with the issue of the top-ranked alternative. Although decision-makers would most often be interested in the top ranked alternative, there are numerous decision contexts where the interest is in the ranking of all alternatives (e.g., the allocation of merit raises across an entire department). In general, complacency and consensus should be considered hierarchically, i.e., consensus should only be evaluated after non-complacency is established.

Key Applications

The widespread use of data warehouses has highlighted the necessity of data hygiene. However, it is infeasible to thoroughly cleanse the data, particularly when the ways in which the data may be used is in flux. A viable alternative may be to tag the fields in the warehouse with data quality information. Some of the issues that need to be considered when designing such a warehouse have been discussed earlier. The results from preliminary experimental studies described below indicate that it would be beneficial to include data quality tags when the data warehouse is used by managers on an ad hoc basis.

Future Directions

Although some experimental research has established that data quality information is used by decision makers, more work is required to determine exactly how such information is used. This would help in identifying the best format in which data quality information should be presented to users. The results of such investigations would also provide guidelines for designers of data warehouses and those that implement warehouse applications.

Experimental Results

Several experiments have been conducted to provide insight into the type of data quality information that would be most effective to support decision processes. A preliminary question that needed to be addressed was the proposition that the type of data quality information that is appropriate may be a function of individual preferences. If individuals processed data quality information in substantially different ways, incorporating data quality information may be counter-productive. A study that used learning styles as a surrogate for individual differences in the mindsets of professionals in an organization found no significant differences that were attributable to learning styles using the three key measures outlined above [4]. This makes for simpler implementation since this implies that the data quality information for a database does not necessarily have to be tailored to individuals.

A series of controlled experiments [2–4] investigating the impact of providing data quality information to decision makers found that the impact depended on factors such as the complexity of the task, the decision strategy and the format in which the information was provided. Preliminary results suggested that when decision makers are confronted with clearly differentiated alternatives, the inclusion of data quality information impacted the selection of a preferred alternative while maintaining group consensus. A follow-up study examined these factors while also taking experience and time pressure into account [5]. The findings suggested that managers with little to no domain-specific experience were more likely to use the data quality information provided. In addition, the results indicated that those who felt time pressure during the decision making process were more likely to use the data quality information provided. Finally, older decision makers were more likely to use data quality information than younger ones. However the level of consensus among the decision makers declined, indicating that improvements gained in individual decision processes through the inclusion of data quality information may be at the expense of group consensus.

Cross-references

- ▶ Data Cleaning
- ▶ Data Quality Assessment
- ▶ Data Quality Dimensions
- ▶ Data Warehouse
- ▶ Data Warehouse Life-cycle and Design
- ▶ Data Warehouse MetaData
- ▶ Data Warehousing Systems: Foundations and Architectures
- ▶ Information Quality Assessment
- ▶ Information Quality Policy and Strategy
- ▶ Information Quality: Managing Information as a Product
- ▶ Meta Data
- ▶ Metadata Repository
- ▶ Quality of Data Warehouses

Recommended Reading

1. Ben Zur H. and Breznitz S.J. The effects of time pressure on risky choice behavior. Acta Psychologica. 47:89–104, 1981.

2. Chengalur-Smith I., Ballou D.P., and Pazer H. The impact of data quality tagging on decision complacency. In Proc. 2nd Conf. on Information Quality, 1997, pp. 209–221.
3. Chengalur-Smith I., Ballou D.P., and Pazer H. The impact of data quality information on decision making: an exploratory analysis. IEEE Trans. Knowledge and Data Eng, 1999, pp. 853–864.
4. Chengalur-Smith I. and Pazer H. Decision complacency, consensus and consistency in the presence of data quality information. In Proc. 3rd Conf. on Information Quality, 1998, pp. 88–101.
5. Fisher C.W., Chengalur-Smith I.N., and Ballou D.P. The impact of experience and time on the use of data quality information in decision making. Inform. Syst. Res., 14(2):170–188, 2003.
6. Gaeth G.J. and Shanteau J. Reducing the influence of irrelevant information on experienced decision makers Organ. Behav. Hum. Perform., 33:263–282, 1984.
7. Grether D.M., Schwartz A., and Wilde L.L. The irrelevance of information overload: An analysis of search and disclosure. Southern California Law Rev., 59:277–303, 1986.
8. Hansson S.O. Decision making under great uncertainty. Philos Social Sci., 26(3):369–386, 1996.
9. Johnson E.J., Payne J.W., and Bettman J.R. Information displays and preference reversals. Organ. Behav. Hum.Decision Process., 42:1–21, 1988.
10. Keller K.L. and Staelin R. Effects of quality and quantity of information on decision effectiveness. J. Consum. Res. 14:200–213, 1987.
11. MacGregor D., Lichtenstein S., and Slovic P. Structuring knowledge retrieval: An analysis of decomposed quantitative judgments. Organ. Behav. Hum. Decision Process., 42:303–323, 1988.
12. Payne J.W., Bettman J.R., and Johnson E.J. The adaptive decision maker. Cambridge University Press, 1993.
13. Stone D.N. and Schkade D.A. Numeric and linguistic information representation in multivariate choice. Organ. Behav. Hum. Decision Process., 49:42–59, 1991.
14. Wang R.Y. and Madnick S.E. A polygon model for heterogeneous database systems: The source tagging perspective. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 519–538.
15. Wang R. and Strong D. Beyond Accuracy: What Data Quality Means to Data Consumers. J. Manage. Inform. Syst., 4:5–34, 1996.

Information Quality Assessment

Leo L. Pipino
University of Massachusetts, Lowell, MA, USA

Synonyms

Information; Data

Definition

This entry uses the terms data and information interchangeably. The classical distinction is that data are raw facts whereas information is data in context or data that have been processed. Nevertheless, other than at an abstract level, it is a distinction that is often not made and one finds that the terms are used interchangeably. It is important to note that one individual's information can be data to another individual. This entry will also use the terms information quality dimension and information quality variable interchangeably.

Further, this chapter defines information of quality as information that is fit for use (or data of quality as data that is fit for use). This means that context and use plays an important role in evaluating information quality. For example, the instantaneous changes in a stock's price may be of importance to the stock trader who may trade stocks on a minute by minute basis. This instantaneous information, however, will be of minor importance to a long term investor whose strategy is "buy and hold" in which long term price trends are of more interest and use.

The process of measuring data quality must confront two major questions: (i) Exactly what should be measured? and (ii) What is the metric that will be used to measure the variable? Often, answering the first question is much more difficult than answering the second.

This entry does not address the area of software quality or the quality of information systems. The former has been examined extensively in the Computer Science literature and the latter has received a great deal of attention in the Management Information Systems literature.

Historical Background

Historically, prescriptions for the measurement of information quality have focused on specific variables. The often cited variables of accuracy, timeliness, completeness, and consistency are examples. Indeed, what dimensions to measure and the definition of what metrics to use are still debated today. This entry will not present a detailed recapitulation of the history and literature of information quality measurement. Rather, it will present one approach – broader in scope than simply measuring specific variables – that can be an effective way to measure data and information quality.

Foundations

As alluded to in the above paragraph, the typical approach to measurement is to measure specific variables or dimensions of information quality. This chapter advocates a broader approach that views the assessment of information quality in an organization to include assessment of the management of the process of insuring information quality. Within this framework, the chapter also widens the scope of what is to be measured.

An assessment of management involvement and management practices is essential. These would include, but are not limited to, examining whether policy and procedures are in place, if learning is proactively fostered in the organization, if proper configuration and control is in place, and whether or not audits are periodically conducted.

Measurement of the information product includes evaluations of fitness for use, such as the existence and maintenance of metadata, whether or not data integrity constraints are in place, and assessments of many of the traditional quality dimensions. These evaluations would be more qualitative than quantitative in nature. An example of an instrument that has been used by a major healthcare facility to help in performing this assessment is the Data Quality Practice & Product Assessment Instrument described in [8].

One must, however, also measure the quality of information along specific dimensions at a more detailed level than described above. Again, the approach suggested here to measure specific information quality dimensions has a wider perspective than is typical. It is based on previous work [8,9]. A variant of this approach, for example, has been used in practice by a major teaching hospital.

The basic approach is to assess (i) subjective perceptions of information quality, that is, the perceived quality of information along some specific quality dimensions and (ii) objective quantitative measures of the same dimensions, when measurable.

The subjective assessments measure the perception of different stakeholders. Specifically three stakeholders are identified:

1. The collector – the individual (or group) that obtains and/or enters the data.
2. The custodian – the individual (or group) that is responsible for storing, retrieving, and providing the data.
3. The consumer – the user of the data.

Each may (and most likely) will have different perceptions of the quality of specific data and information. These subjective measures can provide comparisons across stakeholders and, as a consequence, provide valuable information regarding the disconnects among the different stakeholders with respect to specific data.

For many variables, these perceptions can also be compared to quantitative measures of the same variable. Here, the comparison may show, for example, that stakeholders think data is of good quality along a certain dimension when, in fact, the data is actually poor. Of course, the comparison can also show agreement between the quantitative measure and the subjective perception.

What should be measured? As mentioned earlier, what to measure can be more difficult to decide than how to measure it. The decision will be context dependent and must be determined by the organization and what variables (data quality dimensions) are important to it and its goals, objectives, and environment. The variables accuracy, timeliness, and completeness rank high among those mentioned often. Exactly how these are defined is important and the granularity of the definition and, consequently the measurement, must be carefully considered. For example, accuracy is always mentioned in the same breadth as data quality. Exactly how can one define accuracy. Assume that the unit of measurement is the record. Further assume that records have missing values, that is, they are incomplete. Will the record be considered inaccurate or will this type of problem be defined as incompleteness and accuracy defined more as "free from error," that is that fields in the record with values (those that are not missing) are correct. This chapter will not answer this question. These are questions that the organization and its analysts must answer in the specific context of the organization. Different organizations will answer the question somewhat differently. This simple example points out the issue and helps focus on the importance of the definitions in determining what is to be measured.

The metric itself may be easier to define. For illustration purposes, if the incompleteness definition given above is used, then a simple ratio of complete to total records can be used. Assuming that in most databases the number of complete records far outweigh the records with missing data, the formulation of one minus the above ratio could be used.

Although not an exhaustive nor a completely independent set, a comprehensive set of dimensions from which one can choose has been provided in [13]. The list is not meant to exclude other possible variables. A subset from this list or other variants can be used. The important point is to clearly define what is to be measure.

The subjective perceptions can be measured using questionnaires whose questions are rated numerically using a Likert type scale. One example that has been used in practice is the Information Quality Assessment instrument (IQA) [8] or a subset of the IQA. For example, such statements as shown below are evaluated in the context of a specific organization and data set or database:

1. The information is incomplete.
2. The information is believable.
3. The information is correct.

In the case of the IQA, the respondent (any of the stakeholders) would provide an evaluation on a scale from 0 to 10 with 0 representing complete disagreement with the statement and 10 representing complete agreement. Of course, one could use a different numerical range for the evaluations. It is important to reemphasize that with the IQA the respondents are subjectively assessing the pool of information or database(s) which the organizations has chosen to evaluate. It is not an evaluation of one item or piece of solitary information.

For those variables that can be quantified, metrics can be defined and quantitative measurements can be performed. For example, one might physically measure the number of incomplete records and use a metric similar to the following:

Degree of Completeness = 1 – Number of Incomplete Records/Total Number of Records

For other dimensions, such as the believability of information, it may be decided that only the subjective perceptions will be used. Alternatively, if a reasonable definition of, for example, believability can be specified as a function of other quantitatively measured variable, and this is a big if, then some type of aggregation function can be used. An example might be a weighted average of the numerical values of the quantitative variables where the weights indicate relative importance, range from zero to one, and sum to one. Such aggregation, if done at all, must be performed with care since one must always be attentive to inappropriately combining scales of different types (ratio, interval, ordinal, and nominal [7]). This can lead to inappropriate use. Accepting the resulting precise numerical values as ratio or interval data could lead to poor decisions on the part of the decision maker.

In sum, at the detailed measurement level, the approach recommended here is a comparative approach wherein after what to be measured has been determined:

1. Subjective perceptions are measured.
2. Objective/quantitative measurements are performed. This would then lead to:
3. Comparison of results.
4. The taking of action where necessary to improve the data quality.

Of course, either step (1) or (2) can be used independently, that is, as the sole measure.

Recent work merits mention in a chapter on the measurement of data quality. It has been suggested [5] that the construct of utility be incorporated into the information quality measurement. Here utilities would indicate the relative value or importance of certain records within the set. These would be taken into account in generating quantitative assessments for a specific dimension. Note that, as modeled in [5] utility is not a measure of the relative importance or value of different variables (dimensions) but rather the relative importance of the records within a set measured along one dimension. This is ongoing research but can easily be incorporated into the metrics used in step 2 above.

Key Applications

The key application is the assessment of the quality of information that an organization uses.

Cross-references

▶ Data Deduplication
▶ Data Quality Assessment
▶ Data Quality Dimensions
▶ Information Quality Policy and Strategy

Recommended Reading

1. Ballou D. and Pazer H. Modeling data and process quality in multi-input, multi-output information systems. Manage. Sci., 31(2):150–162, 1985.
2. Ballou D., Wang R., Pazer H., and Tayi G. Modeling information manufacturing systems to determine information product quality. Manage. Sci., 44(4):462–484, 1998.
3. Batini C. and Scannapieco M. Data Quality: Concepts, Methodologies and Techniques. Springer, New York, 2006.
4. Codd E. The Relational Model for Database Management: Version 2. Addison-Wesley, Reading, MA, 1990.
5. Eden A. and Shankaranarayanan G. Understanding impartial versus utility-driven quality assessment in large datasets. In Proc. 12th Conf. on Information Quality, 2007, pp. 265–279.
6. Huang K., Lee. Y., and Wang R. Quality Information and Knowledge. Prentice-Hall, Englewood Cliffs, NJ, 1999.

7. Krantz D., Luce R., Suppes P., and Tversky A. Foundations of Measurement: Additive and Polynomial Representation. Academic Press, New York, 1971.
8. Lee Y.W., Pipino L.L., Funk J.D., and Wang R.Y. Journey to Data Quality. MIT Press, Cambridge, MA, 2006.
9. Pipino L., Lee Y., and Wang R. Data quality assessment. Commun. ACM., 45(4):211–218, 2002.
10. Redman T. Data Quality: The Field Guide. Digital Press, Belford, MA, 2001.
11. Strong D., Lee Y., and Wang R. Data quality in context. Commun. ACM., 40(5):103–110, 1997.
12. Wang R., Lee Y., Pipino L., and Strong D. Manage your information as a product. Sloan Manage. Rev., 39(4):95–105, 1998.
13. Wang R. and Strong D. Beyond accuracy: what data quality means to data consumers. J. Manage. Inf. Syst., 12(4):5–34, 1996.

Information Quality Policy and Strategy

YANG W. LEE
Northeastern University, Boston, MA, USA

Synonyms

Data Quality

Definition

Information quality policy and strategy is an emerging area of study and practice. Information quality policy and strategy involve managing both information product and information practice. A recent piece of literature structured information quality policy around ten key guidelines and provided a questionnaire on four key areas of information product: source data, metadata, system-based data integrity, and data quality, all of which were assessed by all stakeholders. The questionnaire also included the assessment of strategic, managerial, and operational level of information practice [3].

Conventionally, managing information quality means ensuring data integrity of stored data in a database and managing authoritative access and security. Increasingly, organizations manage and use heterogamous databases in dispersed strategic business units. With globalization trends, mergers and acquisitions, and virtual collaboration, managing information entails much more beyond managing data in a stored database within a single organization. Managing information in today's organization, therefore, means managing the comprehensive information process: data collection, storage and utilization processes [3]. High-quality information is used to compete in the market place and is a new strategically competitive asset [5]. To improve the quality of information available to information consumers, organizations need to manage information with an established policy and strategy.

Historical Background

While there are some early seminal papers, information quality (IQ) only emerged as a recognized field of information technology (IT) research in the mid-1990s. For example, the first International Conference on Information Quality was held in 1996 at MIT. In 1998, Wang et al. [6] published "Manage Your Information as a Product," which argued for the concept of treating information as a product which is intentionally managed and produced for consumers who have expectations of its quality. In 2006, Lee et al. [3] published "Journey to Data Quality," which included the first set of generic information quality policies, arguing for the principles of managing information with an institutionalized set of policies that structures decisions on information.

Foundations

This section provides ten policy guidelines that should form the basis of an organizational data quality policy and strategy [3].

Treat Information as a Product, Not By-Product

Treating information as a product, not by-product, is the overarching principle that informs the other guidelines [6]. This principle must be continually communicated throughout the organization. It must be a part of the organization's ethos. More details on the approach are available in Wang et al. [6].

Establish and Maintain Information Quality as a Part of the Business Agenda

An effective policy requires that an organization maintain information quality as part of its business agenda. The organization must understand and document the role of data in its business strategy and operations. This requires recognition of the data needed by the firm's business functions to complete its operational, tactical, and strategic tasks. It should be clearly understood that this data is fundamental to the competitive positions of the firm.

The responsibility for ensuring that data quality is an integral part of the business agenda falls on senior executives. Senior executives should understand their leadership is critical to success and that they must be proactive in fulfilling the leadership role. They must take an active part in ensuring data quality in the organization so as to utilize quality information as a strategically competitive advantage.

Ensure that Information Quality Policy and Procedures are Aligned with Business Strategy, Business Policy, and Business Processes

The primary function of data is to support the business. To do so, a data quality policy must be in place and consistent with an organization's business policy. This implies that a broad view of what constitutes data policy is necessary. An integrated, cross-functional viewpoint is a prerequisite to achieving an aligned data quality data quality policy. Proper alignment will also foster the seamless institutionalization of data quality policy and procedures into the fabric of the organization's business processes.

Improper alignment or too narrow of a focus can cause data quality problems and prove disruptive to the organization. It can lead to conflict which could otherwise be avoided. For example, a view that restricts data quality policy to technical details of data storage and ignores the uses of the data at different levels of decision making would result in misalignment and subsequent problems. In short, data quality policy should reflect and support business policy. It should not be independent or isolated from business activities.

Establish Clearly Defined Information Quality Roles and Responsibilities as Part of Organizational Structure

Clear roles and responsibilities for data quality must be established. There should be specific data quality positions within the organization, not ad hoc assignments to functional areas in times of crisis. Additionally, one of the fundamental goals of data quality function should be to identify data collectors, data custodians, and data consumers as well as make these members of the organization aware of which of these roles that they and others play.

Roles and responsibilities specific to the data quality function range from the senior executive level to the analyst and programmer level. Current examples of titles in practice include chief Data Officer, Senior VP of Information Quality, Data Quality Manager, and Data Quality Analyst.

Ensure Data Architecture is Aligned with Enterprise Architecture

The organization should develop an overall data architecture that is aligned with and supports its enterprise architecture. Enterprise architecture means the blueprint of the organization-wide information and work infrastructure. The data architecture helps promote a view of the information product which is consistent across the enterprise. This promotes accessibility of information across the organization.

The data architecture reflects how the organization defines each data item, how data items are related, and how the organization represents this data in its systems. It also establishes rules and constraints for populating individual data items. All of theses are essential elements of any product – a definition of the item, statements indicating how it is related to other entities in the business, and what constraints exist for populating it in various databases.

To ensure that the data architecture is and remains aligned with the enterprise architecture entails a number of specific and detailed tasks. For example, establishing a data repository is critical to the development and maintenance of viable data architecture. The organization should generate a set of metadata that clearly defines data elements and provides a common understanding of the data. This enables efficient data sharing across the organization.

Be Proactive in Managing Changing Data Needs

The data needs of consumers change over time. The term *consumers* means any individuals and enterprises internal or external to the organization who use the data of their organization. To maintain a high degree of data quality, the organization must be sensitive to ever-evolving environments and changing needs. This entails continual scanning of the external environment and markets along with the changing needs of internal data consumers.

It is also critical to the maintenance of high data quality that the rationale for changes in policy and procedures is clearly and promptly communicated throughout the organization. Steering committees and forums can play a crucial role in the dissemination of this information.

The organization must also be cognizant of local variances as they relate to global data needs. In today's multinational and global firms, long-standing local culture and customs exist. These local differences must be clearly identified and policy adapted to incorporate these local variances accordingly.

Have Practical Data Standards in Place

It is easy to get people to agree that standards should be in place. It is much more difficult to reach agreement on what should be standardized and the prescription of the standard. At times, it is difficult to obtain agreement on the definitions of the most basic data elements. Data standards entail many areas of data practice. Addressing the following questions will help inform the process for instituting practical data standards internally. If external standards are to be used, which ones should be chosen? Does the standard apply locally or globally? In either case, how and when will it be deployed? What processes are appropriate to monitor and adopt changing standards over time? How should they be documented and communicated?

The process an organization uses to determine whether to choose external or internal standards, and under what conditions, should be continually developed. The preferred solution would be to use a standard which has been created and maintained by an organization other than the organization using the standard. If this is possible and acceptable, then existing standards to adopt must be determined. This may be a choice between international, national, or industry standards or between two national standards. For example, the International Standards Organization (ISO) has a standard set of country codes. One's first choice would be to choose the international standard. Other things being equal, this provides the greatest degree of flexibility and lowest costs to the firm.

Plan for and Implement Pragmatic Methods to Identify and Solve Data Quality Problems, and Have in Place a Means to Periodically Review Data Quality Practice and the Data Product

Pragmatic methods based on diagnostic methods and measures should be promoted as standards across the organization. Example methods include system-based data integrity, data quality assessment, and data product and practice assessment methods.

Periodic reviews of the data quality environment and the overall data quality of the organization is a must. One method to achieve this is to assess the data quality practice and the firm's data quality using a standard assessment survey. Multiple assessments over time with a standard survey will provide a series of assessments against which the firm can measure its progress. Further, if an evaluation of an exemplary organization is available, the firm can use this exemplar's results as a benchmark and evaluate its performance against it.

A practice related to the auditing task involves certification of data. Data of both transaction systems and data warehouses must be included in the certification process. A robust process of data certification will ensure that data practice and data product will be of high quality. As a consequence, a data certification policy must be put in place. The policy should stipulate that sources for the information be clearly identified; including definitions, valid values, and any special considerations about the data.

Foster an Environment Conducive to Learning and Innovating with Respect to Data Quality Activities

Initiatives that contribute to a learning culture would include the use of a high-level data quality steering committee, the installation of forums across the organization, the stress on continuous education and training. All should be strongly supported and conveyed by senior management.

The organization should put in place a policy of rewards, and if necessary, sanctions, that are consistent with achieving a high level of data quality and maintaining this high level. What should not be discouraged is the identification and reporting of data quality problems, even if these prove embarrassing to management. Employees should feel free and safe to report data quality problems openly. A formal mechanism for reporting such problems can help in this regard. Establishing a forum for data collectors, data custodians, and data consumers to voice and share data quality problems and discuss data quality issues will be of great value. This will also aid in the dissemination of the experiences of the collectors, custodians, and consumers and foster an appreciation for each other's perspectives and roles.

It is especially important for data collectors to understand why and how consumers use the data. Data collectors also must adhere to the data quality principles as they enter data even though they are not the prime beneficiaries of such activity and do not usually get measured on this portion of the data entry task. Data consumers are critical to providing appropriate feedback on the quality of the data they receive, including whether it is needed data. The function would include apprising the data collectors of changes in data needs. Data custodians must recognize that although they are not directly responsible for the data, they must understand its purpose and use. As with data collectors, data custodians should be aware of how and why data consumers use the data.

Establish a Mechanism to Resolve Disputes and Conflicts Among Different Stakeholders

Data policy and jurisdictional disputes, data definition disagreements, and data use debates will occur. An organization must have in place a mechanism to resolve theses differences and conflicts. These could be a set of hierarchically related mechanisms corresponding to different levels of the firm. Their form is not as important as their existence and that they have a clear charter specifying their responsibilities and authority.

Any of a number of conflict resolution techniques and mechanisms can be used. The specifics must be consistent with an organization's business strategy and culture. Examples are the use of steering committees, data quality boards, data quality work groups, and data quality councils. The scope of responsibilities, the degree of authority, and clearly defined lines of reporting and communications must be specified. Ambiguity in the specifications will render theses resolution mechanism ineffective.

It is most important that a historical record maintains disputes and their resolutions. This institutional memory will serve the organization well as it adapts to changing future environments.

Key Applications

Early adaptors are applying information policy and strategy in their organizations to great avail. Public sector agencies are advanced in their adoption of the policy and strategy since most public agencies approach information quality policy for their planning and implementation of information management that is guided by the mandated Information Quality Act. The public agencies such as the Intelligence Community, Environmental Protection Agency, Department of the Interior, and Department of Housing and Urban Development also align information quality policy with their Enterprise Architecture and Governance policy. In the private sector, most organizations approach information quality policy initially from their data warehouse or customer data management point of view that they need a policy to share the information at the enterprise level. Software vendors align some of the pragmatic policy issues as the underlying methodology which frames their tools; such as data integration, business intelligence, data migration, ETL, and data profiling tools.

Future Directions

The importance of data quality polices and strategies, good data practices and principles, effective data quality training, and communication within the organization cannot be overemphasized. This often overlooked aspect of data quality practice can mean the difference between those that are successful in this "journey to data quality" and those that are not [3].

The ten policy guidelines presented can serve as potential areas for future research. The needs of information consumers are well-defined; reliable measures of data quality assessment have been developed; and methods for analyzing, diagnosing, and improving information quality are readily available. The data products are studied and used in organizational practices. However, the data practice area including policy and strategy is in an early stage and will mature in the future as more firms demand policy and strategy for improving and aligning their information quality agenda with their business agenda.

More research is also needed to integrate the perspective of data product and data practice. To integrate both perspectives effectively, researchers will need to apply multi-disciplinary methods and theories. For example, "context-reflective data quality problem-solving" [2] uses various disciplinary areas and integrates data product and data practice. Another example [4] is seen with "process-embedded data integrity" which integrates data product and data practice. Studies focused on IQ policy and strategy studies are also needed which include organizational outcome measures to be used for directing efforts at policy analysis and strategy forming level. Strategy studies are also needed to develop methods for costing and justifying investments in IQ improvements and assessing competition in the market. The purpose of these research efforts is to develop theories, methods, tools, and policies to help organizations proactively manage their information. Thus, ensuring the delivery of quality information to information consumers and the provision of quality information as a strategic asset; not only to its own organization, but to partner organizations and the extended enterprise of the globalized economy as well.

Cross-references

▶ Information Quality Assessment
▶ Information Quality and Decision Making
▶ Information Quality: Managing Information as a Product

Recommended Reading

1. Ballou D.P., Wang R.Y., Pazer H., and Tayi G.K. Modeling information manufacturing systems to determine information product quality. Manage. Sci., 44(4):462–484, 1998.
2. Lee Y.W. Crafting rules: context-reflective data quality problem-solving. J. Manage. Inf. Syst., 20(3):93–119, 2004.
3. Lee Y.W., Pipino L., Funk J., and Wang R.Y. Journey to Information Quality. MIT Press, Cambridge, MA, 2006.
4. Lee Y.W., Pipino L.L., Strong D.M., and Wang R.Y. Process-embedded data integrity. J. Database Manage, 15(1):87–103, 2004.
5. Matsumura A. and Shouraboura N. Competing with quality information. In Proc. 1st Conf. on Information Quality, 1996, pp. 72–86.
6. Wang R.Y., Lee Y.W., Pipino L.L., and Strong D.M. Manage your information as a product. Sloan Manage. Rev., 39(4):95–105, 1998.
7. Weil P. and Broadbent M. Leveraging the New Infrastructure: How Market Leaders Capitalize on IT. Harvard Business School Press, Cambridge, MA, 1998.

Information Repository

▶ Data Warehouse

Information Retrieval

Giambattista Amati
Ugo Bordoni Foundation, Rome, Italy

Synonyms

Document retrieval; Text retrieval

Definition

Information Retrieval (IR) deals with the construction of automatic systems that allow users to inquire about textual data of any kind through natural language queries. The retrieved information of IR systems may vary from a ranked list of relevant textual items of any kind, such as full documents or their excerpts, or can be distilled into more elaborated forms, such as document summaries or answers to questions. Information Retrieval is an empirical science studying the representation, storage and access to information, and covers a large number of interdisciplinary topics of theoretical computer science including information theory, machine learning, coding theory, probability theory, programming theory, computational semantics, natural language processing, logics and algebra. From a practical perspective research investigation in IR includes: data representation, storage and retrieval, such as indexing, data encoding and text compression, document and term classification and clustering, systems architecture, distributed systems, document-query matching functions (IR models); user-oriented studies and aspects of behavioral science, such as data visualization, browsing, user interfaces, system evaluation, user relevance feedback and automatic query-expansion. With the advent and diffusion of the Web and the dramatic increase of public available information sources IR research also focuses on efficiency in terms of query response time and storage space of indexes in a distributed setting, as well as on personalized search where results can be filtered and adapted to user's area of interest taking into account, for example, geographical knowledge and user's historical data.

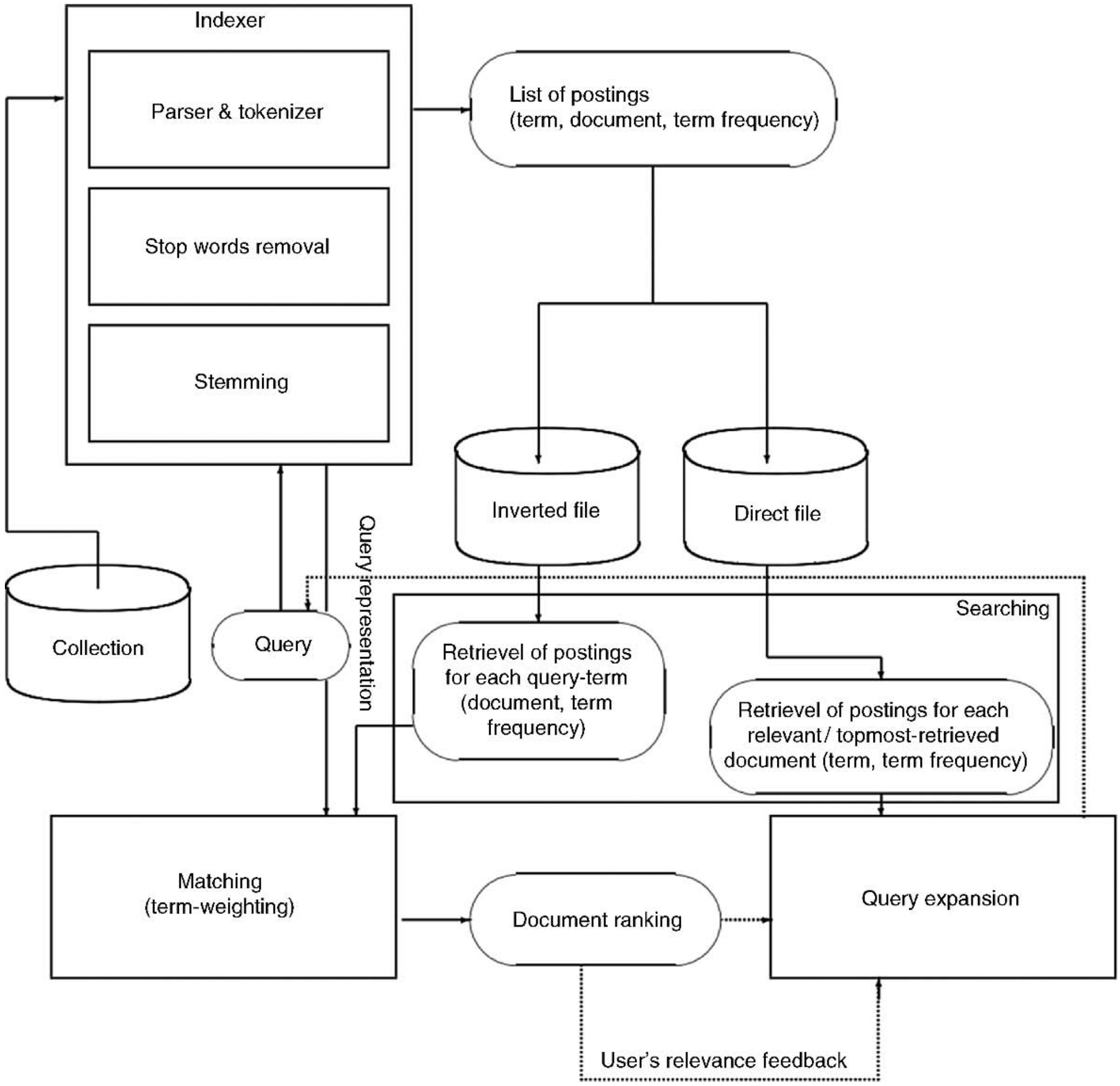
Historical Background

Information Retrieval has its origins in the 1950s to support the librarians activities of indexing and accessing textual collections. Hans Peter Luhn is one of the pioneers of IR [5] with his studies on automatic indexing, which concerns the assignment of significant keywords to documents. In the early stage of IR due to the limitations of computer capabilities document retrieval was restricted to satisfy the boolean exact match between the query-terms and document surrogates (title, subject headings or abstract). Luhn thought that the automation of indexing and abstracting was less prone to errors than human indexers. Before Luhn's original ideas, Shanny, the founder of the mathematical theory of communication, was the first to consider text generation as a "sequence of words" and processed as a discrete information source by a stochastic process (a discrete Markov process) [10]. Following Shanny's idea, Mandelbrot derived theoretically the empirical model by Eustop on word distribution and further studied by Zipf [14]. Mandelbrot showed that text generation is finding the least costly method of coding as obtained in the classical problem in communication theory [6]. The law of Estoup-Zipf-Mandelbrot establishes that the logarithm of rank of words by decreasing frequency is in linear relation with this frequency. An important milestone in the history of Information Retrieval is the introduction of evaluation measures for IR by Cleverdon. in 1953 Cleverdon led a project of Librarian of Cranfield College of Aeronautics to assess retrieval with a "document source – question" method, that consists in collecting questions and their relevance judgments from individuals; the Cranfield test collection is still publicly available for experimentation in IR [3]. In 1960 Maron and Kuhns used the term "probabilistic indexing" in the theory of IR to model the concept of relevance explicated in terms of the theory of probability [7]. Nevertheless, it was the vector space model, a model not based on probability theory, that influenced most of the research for some decades. The SMART project, dedicated to the realization of one of the first IR systems, the SMART system, was initiated at Harvard University in 1961 by Gerard Salton, but actually most of the research was conducted at the Cornell University [9]. The SMART system used the cosine of the index vectors derived from documents and search requests to obtain for each document a coefficient of similarity with each search request. After Maron and Kuhns, a probabilistic model of relevance was introduced by Stephen Robertson and Karen Spärck Jones [8] that has led to the development of the OKAPI system. The probabilistic model is based on Spärck Jones' observation that the significance of a term in a document is due to its rareness in the collection, and it can be measured by the logarithm of the inverse of the relative frequency of the term in the collection (document frequency) [11]. In the early 1970s the "cluster hypothesis" was stated by Jardine and van Rijsbergen [4]. According to the cluster hypothesis both efficiency and effectiveness of search could benefit from clustering similar documents because such clusters are easily retrieved by the same search requests, while Salton thought that clustering would have reduced effectiveness in favor of a better response time.

The first main international conference on IR, ACM SIGIR, was held in 1978 [1]. The first commercial search engines appeared in the 1980s, while the 1990s saw the birth of the first web search engines. Since the beginning of the TREC (Text REtrieval Conference) conference series in 1992, a conference organized by the US government's National Institute of Standards and Technology and dedicated to large-scale evaluation of text retrieval methodologies, very large test collections of full-text documents and standards for retrieval evaluation are available. Evaluation is an important issue in IR, therefore TREC has a significant impact in research because provides an objective evaluation of fresh techniques and approaches, and because promotes new specialized retrieval tasks, as well as, the transfer of emerging new ideas into commercial systems and Web search engines.

Foundations

Information Retrieval systems have four main components: query representation and document indexing, term and document retrieval, query-document matching, relevance feedback processing (see Figure 1). Typically, a user submits a query made up of a few words and phrases, or several sentences. The internal representation of the original query and the text extracted from a document are both processed by the same tokenizer: a *parser* extracts the words taking into account specific properties of the language, then most frequent words and other words that have a functional role in the text (*stop-words*) are removed, and finally the constituents of the lexicon of the system (*tokens*) are created by removing linguistic suffixes or prefixes (*stemmer*), their multiplicity of occurrence in the text is recorded. The text of each document is indexed into two files: the *inverted* and the *direct* file. Both files contain the set of *pointers* of the collection, also knows as *postings*, that is the matrix containing all possible term-document pairs of the collection. With the inverted file one can access the set of pointers, e.g., (term, document, frequency), and thus the set of documents containing a given term together with some extra information about these pointers, that is the frequency of that term in these documents and possibly the positions occupied by that given term in each document. With the direct file, one can instead access the set pointers, e.g., (document, term, frequency), that is the set of terms contained in a given document together with the frequency of each term in that document. Positions of terms within documents are used to restrict search with the use of proximity operators. For example, one can submit the query "information retrieval" wishing to find and rank by relevance all and only all documents containing the word "information" followed by the word "retrieval." While *information AND retrieval* is the query to find and rank all and only all relevant documents containing both the words "information" and "retrieval" irrespective of the positions occupied by these words in the text. Frequencies are instead essential to obtain a ranked list of documents.


Indexer
Parser & tokenizer
Stop words removal
Stemming
List of postings
(term, document, term frequency)
Inverted file
Direct file
Collection
Query
Query representation
Searching
Retrievel of postings for each query-term (document, term frequency)
Retrievel of postings for each relevant/ topmost-retrieved document (term, term frequency)
Matching (term-weighting)
Document ranking
Query expansion
User's relevance feedback

Information Retrieval. Figure 1. A conceptual architecture for an Information Retrieval system.

The direct file is used to perform other post-processing activities, such as *automatic query expansion*, or to *cluster* the retrieved documents into homogeneous or similar classes of documents, or to present results in different output formats (e.g., into an XML form).

Both inverted and direct file are stored in a compressed form, and it is possible to achieve a very good rate of compression with respect to the size of the original textual data. For example the inverted file of the TREC corpus WT10g containing 1,692,096 documents of 10 GB of text and 280,571,311 pointers, can be compressed in about 385.5 MB, each pointer (term-document, term frequency) requiring an average of 11.5 bits of information only. In general, the most frequent words are declared in a special list of words, the *stoplist*, and are not indexed because they require the storage of too many pointers (the size of the collection in the worst case). A comprehensive study of compression algorithms for text retrieval can be found in the book *Managing Gigabytes* [13].

The kernel of an information retrieval system is the query-document matching model, producing the document-scores for the given query, also known as the *retrieval status value* (RSV) of the documents for that query. The matching model is the theoretical component responsible of the effectiveness and the quality of the retrieval.

Similarly to any other empirical science, Information Retrieval makes inferences by analysis of the empirical data consisting of three phases: *model specification*, *parameter estimation* and *model evaluation*. However, unlike other empirical sciences, IR has two types of data: the postings, that is the elements of the term-document matrix, and the user relevance data set, that are in general provided by a set of document-query relevance assessment pairs. The relevance data set can be used to accomplish two different tasks: query re-formulation for improving document retrieval or the evaluation of system performance.

Due to the existence of relevance feedback data, IR has two different kinds of models: the *query-language model* and the *term-document model*. The query models aim at providing the weights of the query terms irrespective of the observed document. The term-document model instead compares and weights the frequency of the term within a document with respect to its frequency in the collection.

For example, the query "What is a prime factor?", which is transformed into the query "prime factor" after stop-words removal and stemming, can be simply represented as the vector (prime = 1, factor = 1). After a first pass retrieval in absence of user's feedback, one may deem the first retrieved documents as relevant and assume that the terms contained in this small portion of retrieved documents as a sample of the term population relative to the topic "prime factor." Then, the query-language model will assign new weights to the original query-terms and add new terms to the query. For example, retrieving just the first three documents the new query might be (prime = 1.3581, factor = 1.2327, integ = 0.2154, number = 0.1778, primal = 0.0941,...). The term-document weights assigned by the term-document model will be resized according to query-term weights, that is as the inner product of these two weight vectors.

A theory on evaluation of IR systems is mainly developed in van Rijsbergen's book [12]. The effectiveness of an IR system is evaluated by two standard mea sures: *recall* and *precision*, that are defined as follows:

$$Recall = \frac{\mu(Rel \cap Ret)}{\mu(Rel)}$$

$$Precision = \frac{\mu(Rel \cap Ret)}{\mu(Ret)}$$

where $Ret = \{d|d \text{ is retrieved}\}$ and $Rel = \{d|d \text{ is relevant}\}$, and $\mu = |\cdot|$ is the counting measure. Then, $|Rel \cap Ret|$ is the number of relevant and retrieved documents, $|Ret|$ is the number of retrieved documents, and $|Rel|$ is the number of relevant documents.

Key Applications

Main applications of Information Retrieval concern the construction of search engines either for specific domains, like biomedicine and genomics, law, web, blogs, or adapted to particular types of document structure (e.g., XML or hypertext documents) search engines, or dedicated to multimedia and digital libraries.

Future Directions

Notwithstanding the increase of computer processing power, most of IR applications deal with very large collections that cannot be in general processed by only one server. Both efficient implementation and distributed versions of IR systems are required. The design of efficient partitions of very large indexes that need to be distributed over a cluster of machines is an important research topic. Also, large community of users may wish to share their information and knowledge, but not local indexes, and therefore merging local search results to obtain one effective global document ranking is a challenging research issue (*data fusion*). Most of the new technology for information retrieval is becoming more and more an asset of the most popular web search engines, and therefore it is mainly the market that influences new academic research directions. Although social network analysis has achieved a quite mature technology based on variations of Markov chain models with stationary probability distributions, a new social phenomenon is now emerging in the web, the *social tagging*; systems also provide collaborative tools to Internet users to store and search structured comments of web pages. Query search on movies, music or books will be in the very next future conditioned, for example, by opinions or by recommendation. More generally, search will be more and more personalized and tailored to one's own profile or to all other profiles similar one's own. The major techniques to discover statistical relationships for hypertext documents and hyperlinks can be found in Chakrabarti's book [2].

Data Sets

Most of data sets and test collections can be found at the TREC (NIST) web site http://trec.nist.gov/data.

html. There are several types of test collections concerning different range of IR applications. To cite few of them, there are the *ad hoc* collections, that are dedicated to the standard document retrieval based on a primitive notion of user's relevance, *web* test collections, that are dedicated to web retrieval, *Blog Track* test collections, that contains a large collection of permalinks from the blogosphere.

URL to Code

There are several open source IR systems on the web. In the following there is a list of the most popular and advanced academic IR research systems.

1. Indri and Lemur search engines developed by the Carnegie Mellon University and the University of Massachusetts, Amherst, United States: http://www.lemurproject.org/
2. Terrier developed by University of Glasgow, United Kingdom: http://ir.dcs.gla.ac.uk/terrier/.
3. Wumpus developed by University of Waterloo, Canada: http://www.wumpus-search.org/
4. Zettair by the RMIT Univerisity of Melbourne in Australia: http://www.seg.rmit.edu.au/zettair/

Other IR systems are the open source Apache Lucene (http://lucene.apache.org/) and the SMART system (ftp://ftp.cs.cornell.edu/pub/smart/).

Cross-references

▶ Biomedical Scientific Textual Data Types and Processing
▶ Clustering
▶ Digital Libraries
▶ Information Retrieval Evaluation Measures
▶ Information Retrieval Models
▶ Information Retrieval Operations
▶ Query Expansion Models
▶ Relevance Feedback
▶ Text Indexing Techniques

Recommended Reading

1. Annual International SIGIR Conference, Proceedings of the ACM Special Interest Group on Information Retrieval Conference. http://www.sigir.org/.
2. Chakrabarti S. Mining the Web: Discovering Knowledge from Hypertext Data. Morgan-Kauffman, 2002.
3. Cleverdon C. The cranfield test on index language devices. Aslib Proc., 19:173–192, 1967.
4. Jardine N. and van Rijsbergen C.J. The use of hierarchic clustering in information retrieval. Inform. Storage Retr., 7 (5):217–240, 1971.
5. Luhn H. A statistical approach to mechanized encoding and searching of literary information. IBM Journal of Research and Development, 1:309–317, 1957.
6. Mandelbrot B. An informational theory of the statistical structure of language. In Communication Theory, the Second London Symposium. W. Jackson (ed.). Butterworth, London, 1953, pp. 486–504.
7. Maron M.E. and Kuhns J.L. On Relevance, Probabilistic Indexing and Information Retrieval. J. ACM 7(3):216–244, 1960.
8. Robertson S.E. and Sparck-Jones K. Relevance weighting of search terms. J. Am. Soc. Inform. Sci., 27:129–146, 1976.
9. Salton G. and Lesk M.E. The SMART automatic document retrieval systems – an illustration. Commun. ACM, 8 (6):391–398, 1965.
10. Shanny C. A mathematical theory of communication. Bell Syst. Tech. J., 27:379–423 and 623–656, 1948.
11. Sparck J. K. A statistical interpretation of term specificity and its application in retrieval. J. Doc., 28(1):11–21, 1972.
12. Van Rijsbergen C. Information Retrieval, 2nd edn. Butterworths, London, 1979.
13. Witten I.H., Moffat A., and Bell T.C. Managing Gigabytes. 2nd edn. San Francisco, California, 1999.
14. Zipf G. Human behavior and the principle of least effort. Addison-Wesley, Reading, Massachusetts, 1949.

Information Retrieval Models

GIAMBATTISTA AMATI
Ugo Bordoni Foundation, Rome, Italy

Synonyms

Document term weighting; Ad hoc retrieval models; Term-document matching function

Definition

An Information Retrieval (IR) model selects or ranks the set of documents with respect to a user query. Text in documents and queries is represented in the same way, so that document selection and ranking can be formalized by a matching function that returns a Retrieval Status Value (RSV) for each document of the collection. Most IR systems represent document contents by a set of descriptors, called terms, belonging to a vocabulary **V**.

Main IR models define the query-document matching function that weights the query terms occurring in a document according to four main approaches:

- The estimation of the probability of user's relevance *rel* for each document **d** and query **q** with respect to a set $R_{\mathbf{q}}$ of training documents

$$\mathrm{Prob}(rel|\mathbf{d}, \mathbf{q}, R_{\mathbf{q}})$$

- The computation of a similarity function between queries and documents in a vector space

$$SIM(\mathbf{d}, \mathbf{q})$$

- The estimation of the probability of generating the document **d** given a query **q**,

$$\mathbf{p}(\mathbf{d}|\mathbf{q})$$

- The information carried by the query terms in the document, that is that is the number of bits necessary to code X_i occurrences of the query terms $\mathbf{t}_i \in \mathbf{q}$ in the document:

$$-\log_2 \mathrm{Prob}(\mathbf{d}|X_1,\ldots,X_{\mathbf{q}})$$

Historical Background

The history of Information Retrieval goes in parallel to the development of IR models. In general, an IR system is mainly identified with retrieval function employed to rank documents, because it is the retrieval *effectiveness* that matters in IR systems. In the early days, research focused on methods for automatic indexing in contraposition to the manual activities made by librarians. Early works concerned the construction of effective methods for keywords selection to represent succinctly the documents, and document-query matching thus was performed by Boolean search of the query terms in the index of such surrogates of the documents. A full exploitation of a probabilistic model was achieved in the 1990s with the birth of the BM25 ranking formula [7]. Before BM25, most of the theoretical investigation was devoted to the vector space model, the first parameter free model for ranking documents [8]. It was only in the late 1990s that other new models appeared on the scene, such as the language models [6] and the information theoretic models that include the Divergence From Randomness models [1]. Boolean model remains attractive because it requires less computational cost both in terms of size of the indices and response time, and quite a few models tried to extend the Boolean model with fuzzy or logical operators, yet the potentialities of such algebraic and logical models have not been fully exploited.

Foundations

IR models can be classified into four types: probabilistic models, algebraic and logical models, information theoretic models and Bayesian models.

Probabilistic models require a training set of data consisting of set of documents which are assessed relevant to a set of queries by users. Algebraic models assume that both queries and terms are represented by vectors, and that the similarity between queries and documents is obtained through a specific normalization of the inner product of these two vectors. In logical models queries and terms are represented by propositions, and the dependency between queries and documents is given by an entailment operator. Information theoretic models are based on the notion of coding cost of terms in the documents, the most informative documents being those generated by the least probable configurations of terms. Bayesian models process two sources of evidence for terms, the term frequencies in the document and in the collection, that are combined to produce an estimate of the probability of the term in the document.

Binary independence model (BIR) and BM25 are probabilistic models. Vector space, Boolean, fuzzy and logical models belong to the algebraic and logical models. The 2-Poisson model [5], and Divergence From Randomness models (DFR) belong to the class of information theoretic models. Language models are instead derived by Bayesian methods.

Probabilistic Indexing

As first IR models, it is interesting to consider the automatic indexing techniques used in the origins of IR. In statistics, observations of empirical data are predicted by stochastic models, that is a probabilistic model which takes into account the presence of some randomness in its variables. In particular, the IR model predicts a probability distribution of possible estimates for term frequencies. An hypothesis that describes the distribution of frequencies is formulated, and the values of inherent parameters of the distribution are estimated by fitting the empirical data to the specified model. Finally, the hypothesis on the distribution form (e.g., Poisson, gaussian, etc.) is accepted or rejected according to an error associated to the fit.

A stochastic model of Information Retrieval models aims to predict a frequency **tf** of a term **t** in a document **d**, that is

$$\mathbf{p}(X_{\mathbf{t}} = \mathbf{tf}, \theta_1, \ldots, \theta_k)$$

where $\mathbf{p}$ is the distribution in the parameters $\theta_1,\ldots,\theta_k$. Both the distribution form and the parameters depend on the observed word.

Once the probability of the frequency **tf** of a term **t** in a document **d** is obtained, then it is also possible to obtain the *probability of relevance of a document* **d** given the term frequency $X_{\mathbf{t}}$:

$$\mathbf{p}(\mathbf{d}|X_{\mathbf{t}} = \mathbf{tf}, \theta_1, \ldots, \theta_k)$$

For example, the Two-Poisson assumes that the distribution of a significant word is a mixture of two Poisson models with mean frequencies $\lambda_{\mathbf{t}} \ll \mu_{\mathbf{t}}$, and the estimate of the probability of the document given the term frequency **tf** is:

$$\mathbf{p}(\mathbf{d}|X_{\mathbf{t}} = \mathbf{tf}, \beta_{\mathbf{t}}, \lambda_{\mathbf{t}}, \mu_{\mathbf{t}}) = \frac{1}{1 + \beta_{\mathbf{t}} \cdot e^{\mu_{\mathbf{t}} - \lambda_{\mathbf{t}}} \left(\frac{\lambda_{\mathbf{t}}}{\mu_{\mathbf{t}}}\right)^{\mathbf{tf}}} \quad (1)$$

with $\lambda_{\mathbf{t}} \ll \mu_{\mathbf{t}}$, and where the values of the parameters $\beta_{\mathbf{t}}$, $\lambda_{\mathbf{t}}$ and $\mu_{\mathbf{t}}$ are learned from data and depend on the observed term **t**.

Information Theoretic Models

The probability estimated by information theoretic models is based on the generation of a document as an unordered partition of terms. Each term $\mathbf{t}_i$ of a vocabulary **V** has a prior probability p_i of occurrence in an arbitrary document of the collection, and a document **d** of length k is conceived as a partition satisfying the constraint $\mathbf{tf}_1 + \ldots + \mathbf{tf}_{\mathbf{V}} = k$ over **V** cells, where $\mathbf{tf}_i$ is the term frequency in the document.

Similarly to the Two-Poisson model the probability of generating a document satisfy the condition

$$\sum_{\mathbf{tf}_i \geq 0} p(X_1 = \mathbf{tf}_1, \ldots, X_{\mathbf{V}} = \mathbf{tf}_{\mathbf{V}}) = 1$$

The connection with Shannon's theory of information is obtained by regarding *a document as a message to transmit.* Shannon explains the notion of information in terms of cost of transmission of a message, which is $-\log_2 p(X_1 = \mathbf{tf}_1, \ldots, X_{\mathbf{V}} = \mathbf{tf}_{\mathbf{V}})$, that is inversely related to the number of choices one has to generate the message with respect to the universe of possible messages: the larger the uncertainty, the larger the information.

If the term independence is assumed, according to which only term frequency counts and positions of terms in documents are irrelevant, a document is treated as an ensemble and is said to be a *bag of words.* In such a case, the probability of generating a document **d** is given by the multinomial distribution:

$$p(X_1 = \mathbf{tf}_1, \ldots, X_{\mathbf{V}} = \mathbf{tf}_{\mathbf{V}}) = \binom{k}{\mathbf{tf}_1 \mathbf{tf}_2 \ldots \mathbf{tf}_{\mathbf{V}}} p_1^{\mathbf{tf}_1} p_2^{\mathbf{tf}_2} \ldots p_{\mathbf{V}}^{\mathbf{tf}_{\mathbf{V}}}$$

It can be shown that

$$\lim_{k \to \infty} \frac{-\log_2 p(X_1 = \mathbf{tf}_1, \ldots, X_{\mathbf{V}} = \mathbf{tf}_{\mathbf{V}})}{k} = D(p_{ML}||p) \quad (2)$$

where $p^i_{ML} = \frac{\mathbf{tf}_i}{k}$ and $D(p_{ML}||p) = \sum_{\mathbf{t}_i \in \mathbf{V}} p^i_{ML} \log_2 \frac{p^i_{ML}}{p_i}$ is the *Kullback-Leibler divergence.*

Exploiting the additivity of the divergence, one can computes the contribution of the query terms to the document:

$$-\log_2 p(X_1 = \mathbf{tf}_1, \ldots, X_{\mathbf{V}} = \mathbf{tf}_{\mathbf{V}}|\mathbf{q}) = \sum_{\mathbf{t}_i \in \mathbf{q}} \mathbf{tf}_i \log_2 \frac{\mathbf{tf}_i}{k \cdot p_i} \quad (3)$$

Similarly to TF-IDF used in the vector space model, also (3) can be used for retrieval.

Different ways to compute the probability $-\log_2 p(X_1 = \mathbf{tf}_1, \ldots, X_{\mathbf{V}} = \mathbf{tf}_{\mathbf{V}}|\mathbf{q})$ and normalize this information with respect the length of the documents is provided by the Divergence From Randomness models.

Vector Space Models

In the vector space model terms and documents are treated as vectors and relevance document scores are obtained by the similarity function

$$SIM(\vec{\mathbf{q}}, \vec{\mathbf{d}}) = \frac{\vec{\mathbf{q}} \cdot \vec{\mathbf{d}}}{||\vec{\mathbf{q}}|| \cdot ||\vec{\mathbf{d}}||}$$

which is the *cosine* function of the angle between the query-document vectors, where the numerator is the inner product of the two vectors and $||\vec{x}|| = \sqrt{\sum_i x_i^2}$ is the norm of a vector. Any term-document weighting model can be employed in the vector space model, differences depend on the ways queries and documents are represented. The vector space model thus is a way to normalize the term-document weight with respect to the length of the document and this normalization is achieved by the cosine function, that is by dividing the inner product by the norms of the vectors.

Assuming each term weight w_i is the number of the bits necessary to encode the $\mathbf{tf}_i$ occurrences of the term in the document, that is $w_i = -\log_2 p(X_i = \mathbf{tf}_i)$, it is easy to show that the inner product can be rewritten as inversely related to the probability of occurrence of the query-terms in the document. According to the *term independence assumption*, the inner product is inversely related to the product of the probabilities of occurrence of each query-term in the document:

$$\sum_i q_i \cdot w_i = -\log_2 \prod_i p(X_i = \mathbf{tf}_i)^{q_i}$$

If $\lambda_i = \frac{\mathbf{n_i}}{\mathbf{N}}$ is the relative document frequency of a term, that is the ratio of the number $\mathbf{n_t}$ of documents in which the term occurs and the number $\mathbf{N}$ of documents of the collection, then the probability $p(X_i = \mathbf{tf}_i)$ can be given by the *geometric distribution*:

$$p(X_i = \mathbf{tf}_i) = \left(\frac{\lambda_i}{\lambda_i + 1}\right)^{\mathbf{tf}_i}$$

that leads to the representation of a document provided by the the first IR vector space model:

$$\vec{\mathbf{d}} = \left(\mathbf{tf}_1 \cdot \log_2\left(1 + \frac{\mathbf{N}}{\mathbf{n}_{\mathbf{t}1}}\right), \ldots, \mathbf{tf}_n \cdot \log_2\left(1 + \frac{\mathbf{N}}{\mathbf{n}_{\mathbf{t}n}}\right)\right)$$

The TF-IDF model weighting of the vector space model is probabilistic, or information theoretic, in its nature, and the *term independence* is an implicit assumption of the model. An estimate of the value of the document relevance is then obtained by dividing the information by the norms of the two vectors $\vec{\mathbf{d}}$ and $\vec{\mathbf{q}}$.

The drawback of this model is the cost of computing document norms. If norms are computed in batch, then a dedicated index must be maintained and updated when documents are added or deleted. On the other hand, if norms are computed online, then one needs to access the direct file to retrieve all the term frequencies $\mathbf{tf}$ of the document.

Bayesian Models

The application of Bayes' theorem to IR concerns the combination of two term frequencies coming from two sources of evidence: the *relative term frequency p_C in the collection*, that is regarded as an estimate of the *prior probability* of occurrence of the term in an arbitrary document of the collection, and *term frequency $p_{ML} = \frac{\mathbf{tf}}{\mathbf{l(d)}}$ in the document*, that is the *maximum likelihood estimate* of the term in the document.

Bayesian models estimate the probability θ_i of a term $\mathbf{t}$ in a document $\mathbf{d}$, considering the θ_i as parameters of the posterior probability

$$\mathrm{Prob}(\theta_1,\ldots,\theta_n|\mathbf{d}) = \frac{L(\mathbf{d}|\theta_1,\ldots,\theta_n) \cdot \pi(\theta_1,\ldots,\theta_n)}{\mathrm{Prob}(\mathbf{d})}$$

where $L(\mathbf{d}|\theta_1,\ldots,\theta_n)$ is the likelihood, $\pi(\theta_1,\ldots,\theta_n)$ is a prior distribution for the parameters θ_i and $\mathrm{Prob}(\mathbf{d}) = \int_0^1 \ldots \int_0^1 L(\mathbf{d}|\theta_1,\ldots,\theta_n) \cdot \pi(\theta_1,\ldots,\theta_n) d\theta_1 \ldots \theta_n$.

If a document is regarded as a set of words, so that permutations of sequences of terms from the vocabulary $\mathbf{V}$ have the same probability distribution (*exchangeable sequences*), then the posterior probability has the form of the product of the multinomial distribution and of a unique prior distribution π:

$$\mathrm{Prob}_{\mathbf{d}}(\theta_1,\ldots,\theta_n|\mathbf{d}) = \frac{\binom{\mathbf{l(d)}}{\mathbf{tf}_1\mathbf{tf}_2\ldots\mathbf{tf}_n} \theta_1^{\mathbf{tf}_1}\theta_2^{\mathbf{tf}_2} \ldots \theta_n^{\mathbf{tf}_n} \cdot \pi(\theta_1,\ldots,\theta_n)}{\mathrm{Prob}(\mathbf{d})}$$

In Bayesian methodology, the prior is chosen in order to have both prior and posterior distribution with the same functional form (*conjugate prior*). The conjugate form to the multinomial distribution is given by the *Dirichlet's distribution*, that is the posterior probability is:

$$\mathrm{Prob}_{\mathbf{d}}(\theta_1,\ldots,\theta_n|\mathbf{d}) \propto \theta_1^{\mathbf{tf}_1+A_1-1}\theta_2^{\mathbf{tf}_2+A_2-1} \ldots \theta_n^{\mathbf{tf}_n+A_n-1} \tag{4}$$

where $\sum_i A_i = A$ are the parameters. The *expectations* $\hat{\theta}_i$ of variables θ_i comes to be:

$$\hat{\theta}_i = \frac{\mathbf{tf}_i + A_i}{\sum_i(\mathbf{tf}_i + A_i)} = \frac{\mathbf{tf}_i + A_i}{\mathbf{l(d)} + A}$$

The Bayesian method constitutes the bulk of the language modeling approach to IR [2,4] (see *language modeling*). However, the values for the parameters A_i need to be learned. The parameters A_i can be further reduced to a single parameter μ by setting the values of A_i to the expected term frequency of $\mathbf{t}_i$ in a document of a fixed length μ, that is $A_i = \mu \cdot p_i$, where p_i is $\frac{\mathbf{TF}}{\mathbf{TFC}}$ is the frequency in the collection (and thus $\mu = \sum_i A_i$). The expectation of (4) thus smoothes the maximum likelihood estimate (MLE) $p_{ML} = \frac{\mathbf{tf}}{\mathbf{l(d)}}$ with the term frequency in the collection, p_i, that is:

$$\hat{\theta}_i = \frac{\mathbf{tf}_i + \mu \cdot p_i}{\mathbf{l(d)} + \mu}$$

The value of the parameter μ depends on several factors, but mainly on the collection and the query length. Another way of smoothing the raw MLE $p_{ML} = \frac{\mathbf{tf}}{\mathbf{l(d)}}$ of the likelihood is the Mercer-Jelinek smoothing technique for IR, that mixes linearly the two frequencies p_{ML} and p_i:

$$\lambda \cdot p_{ML} + (1 - \lambda) \cdot p_i$$

with $0 \leq \lambda \leq 1$.

Probabilistic and Binary Retrieval Models

The independence binary retrieval model (BIR) is based on the assumption that relevance is an event *rel* of the probabilistic space and that relevance can be learned directly from user's feedback. The data containing relevance information are initially gathered by sampling or by assessment on a first pass retrieval, then relevance information is processed to set the values of certain parameters associated with the BIR matching function. The BM25 model also derives from the BIR model.

The BIR model is built upon four probabilities $p_{\mathbf{q}}(\mathbf{t} = a | rel = b)$, where the subscript $\mathbf{q}$ denotes that relevance depends on a specific query, and $a, b \in \{0, 1\}$ are the boolean values *false* and *true* for the variables $\mathbf{t}$ (the term occurs in the documents or not) and for the variable *relevance* (the document is relevant or not) respectively. For the estimation of probability of document relevance BIR uses Bayes' Theorem, which relates the posterior probability of relevance $(p_{\mathbf{q}}(rel = b| \mathbf{t} = a))$ to the prior probability of relevance $(p_{\mathbf{q}}(rel = b))$ and the likelihood of relevance after observing a document $(p_{\mathbf{q}}(\mathbf{t} = a | rel = b))$.

Logical and Algebraic Models

Other approaches to IR modeling include those based on logics, fuzzysets, Bayesian inference networks and the Dempster-Shafer theory of evidence. All these models share the view that (stochastic and/or logical) dependency between sets of words can be captured by a probability defined on top of a (graphical or logical) link between boolean propositional sentences.

A Bayesian inference network is a directed, acyclic dependency graph (DAG) in which nodes represent propositions andedges represent dependence relations between these propositions [10].

Fuzzy models are based on fuzzy-set theory, whichdefines partial membership of elements to a set. Fuzzy models extend logical operators with partial set membership, and processes user queries in a similar way to the conventional Booleanmodel.

A common generalization of both Boolean and fuzzy model is Salton, Fox and Wu's Extended Boolean Information Retrieval Model [9], where the query-document similarity is defined in the L_p-spaces (p-norms).

The logical approach to IR is motivated by modelling retrieval as an inference process. The foundational assumption of logical models is that the semantics of the content of a document is related to a query through a conditional connective, which consists of a type of logical implication set out by Van Rijsbergen in 1986 [11]. One of the IR models based on probabilities on conditionals is obtained by a probability revision method called *imaging* [3].

Key Applications

IR models are applied to a wide range of fields and domains: digital libraries, biomedicine, web search engines, enterprise search, desktop search, search engines for blogs, for legal and other vertical domains, recommendation systems for content providers, information filtering and routing for the selective delivery of news or for detecting spamming and junk mails.

Future Directions

With the dramatic growth of document sources information needs to be retrieved in a distilled manner. The most popular search algorithms for WEB, like PageRank or HITS, clearly indicate that document quality and authoritative content depend on social and collaborative aspects of the community of the users. Users will more and more cite their favorite information sources, express and share their opinions with other in the same network of interests or for entertainment reasons.

Information Retrieval Models will thus naturally evolve along to dynamic search dimensions such as analysis of the social networks, time and context. Contexts include background knowledge about the domain, group of interests, relevant information sources and specific ontologies, as well as profiles with users background, interests, and preferences.

One of the most important research direction is the development of systems that integrates database and IR technology. Such an integration requires new query language, new indexing techniques but more importantly models that combines exact matching, proper of the database systems processing queries for structured information, with partial matching, proper of the IR systems.

Experimental Results

The performance measures of the IR models are based on precision and recall. Significance tests on a set of queries must be conducted to assess the relative performance of models.

Data Sets

Most of data sets and test collections can be found from the TREC (NIST) web site (http://trec.nist.gov/data.html).

Cross-references

- ▶ Digital Libraries
- ▶ Indexing
- ▶ Information Retrieval Evaluation Measures
- ▶ Text Indexing Techniques
- ▶ Text Mining
- ▶ Web Search and Crawling

Recommended Reading

1. Amati G. and Van Rijsbergen C.J. Probabilistic models of information retrieval based on measuring the divergence from randomness. ACM Trans. Inform. Syst., 20(4):357–389, 2002.
2. Berger A. and Lafferty J. Information retrieval as statistical translation. In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 222–229.
3. Crestani F. and Van Rijsbergen C.J. Probability kinematics in information retrieval. In Proc. 18th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 291–299.
4. Croft W.B. and Lafferty J. (eds.). Language Modeling for Information Retrieval. Kluwer Academic, 2003.
5. Harter S.P. A probabilistic approach to automatic keyword indexing. part I: On the distribution of specialty words in a technical literature J. ASIS, 26:197–216, 1975.
6. Ponte J. and Croft B. A language modeling approach in information retrieval. In Proc. 21st ACM SIGIR Conference on Research and Development in Information Retrieval, B. Croft, A. Moffat, and C.J. Van Rijsbergen (eds.). ACM, Melbourne, Australia, 1998, pp. 275–281.
7. Robertson S.E. and Walker S. Some simple approximations to the 2-Poisson model for probabilistic weighted retrieval. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval. Springer, Dublin, Ireland, June 1994, pp. 232–241.
8. Salton G. and McGill M.J. Introduction to Modern Information Retrieval. McGraw–Hill, New York, 1983.
9. Salton G., Fox E.A., and Wu H. Extended boolean information retrieval. Commun. ACM, 26(11):1022–1036, 1983.
10. Turtle H. and Bruce Croft W. Evaluation of an inference network-based retrieval model. ACM Trans. Inform. Syst., 9(3):187–222, 1991.
11. Van Rijsbergen C.J. A new theorethical framework for information retrieval. In Proc. 9th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1986, pp. 194–200.

Information Retrieval Models/Metrics/Operations

▶ Biomedical Scientific Textual Data Types and Processing

Information Retrieval Operations

Edie Rasmussen
University of British Columbia, Vancouver, BC, Canada

Synonyms

Information retrieval processing

Definition

An information retrieval system for collections of unstructured text can be viewed as a set of processing modules, beginning with lexical analysis of documents and leading ultimately to a retrieval process in which user queries are matched against documents. The processes which make up an information retrieval system are referred to as information retrieval operations. In some cases, the end goal is not the retrieval of matching documents from a collection but an auxiliary application, such as document categorization, summarization, or filtering of information from an information stream. While the basic operations in information retrieval systems are similar, there can be considerable variability in the model or details of implementation, making evaluation of information retrieval system effectiveness an important step.

Historical Background

Early work in information retrieval began in the 1950s with the recognition that words in documents could be used as indicators of the content or meaning of the document. Hans Peter Luhn, in early work on automatic abstracting and Keyword-in-Context indexing, recognized that words had differential value in representing the document's meaning [4,5], a concept that is fundamental to modern information retrieval systems. In the 1960s, Salton [7] and his research group developed the SMART system based on the vector space model, combining the basic operations for information retrieval in an information retrieval system. This allowed them to experiment with a variety of information retrieval operations, including stemming, term weighting, similarity functions, and relevance feedback.

Subsequent research has developed and refined information retrieval operations, and there has been increased emphasis on modular implementations which allow specific operations to be isolated. In 1992, Frakes and Baeza-Yates edited Information Retrieval: Data Structures and Algorithms [3], with the goal of making information retrieval more widely known, and making available pseudo-code and code for the basic information retrieval operations. More recently, open-source software such as the Lemur Toolkit has provided easy access to information retrieval systems which allow the isolation and modification of information retrieval operations [1].

Foundations

The basic operations in the building of an information retrieval system are the pre-processing of text, creation of index files, and processing of queries to provide a (usually) ranked list of potentially relevant documents.

Pre-processing of text involves parsing the text to create a list of index terms (lexicon) which will be stored for retrieval. In identifying the index terms, some words may be identified as Stopwords, i.e., common words that are not considered meaningful, and which are not indexed. Words with common suffixes may be identified through an operation called Stemming, and concatenated under a single index entry. While it is possible to select a subset of the words in a text as index terms, usually all words (except words identified as stopwords) in the text are used, so that this is sometimes referred to as a "bag of words" approach.

Early research demonstrated the value of using differential weights for the index terms associated with a document, and testing different models for assigning term weights continues to be an active research area. A long-standing approach was the tf*idf, or term frequency-inverse document frequency weighting scheme, in which every term in a document was assigned a weight proportional to its frequency in that document, and inversely proportional to its frequency in the collection as a whole. More recent weighting schemes include BM25 and weights derived from language models [6].

The resultant document-term matrix is sparse and it is not efficient for storage and processing, so the next set of operations involves building the index to store the terms and associated term weights. The inverted index, in which a postings list of document numbers and associated weights is given for each term, is commonly used. For large data collections, some form of index compression may also be applied [6,10].

Finally, a set of operations on the query results in delivery of the output to the user of the system. A query is presented to the system through a user interface, and it undergoes the same processing as the document collection to identify the index terms which it contains. The order in which query terms are processed may be considered to achieve efficiencies in processing. The query is matched against the documents in the collection using some similarity function, and the documents are ranked in order of similarity. A variety of similarity functions have been proposed and tested; the choice may depend on the underlying information retrieval model. For example, in the vector space model, the similarity function was usually the cosine function which measures the angle between the document and query vector.

Further operations may be carried out on the query in an attempt to improve it. In relevance feedback, information about documents in the ranked list which are either known to be relevant (from user input) or assumed to be relevant, is used to reweight the query terms and potentially offers performance improvement. In query expansion, terms are added to the query, automatically or with the assistance of the searcher, in an attempt to improve retrieval performance.

The retrieval process described above performs what is sometimes referred to as the ad hoc retrieval task, in which the query is matched against a relatively static collection of documents. In some situations it is the query which is (relatively) static, and the documents which are changing. Matching a query against a document stream is referred to as information filtering, and is useful in situations where information which is continually being created must be monitored (for example, from a newswire).

In some instances the end goal is not to produce documents in response to a query, but to perform some other task through specialized processing of the indexed document collection, often in support of information retrieval. An example is document categorization, whereby documents being added to a collection are automatically assigned to what is predicted to be the most appropriate category for them, based on characteristics which have been observed for that category (such as patterns of term occurrence). In document clustering, document-document similarity based on term occurrence and term weights is used to divide a document collection into clusters or groupings of like objects. (Clustering differs from categorization in that the clusters are not a priori known.) Clusters may be created from retrieval output, in order to provide the user with documents which are organized by topic. Similarly, in document summarization, information about the document is used to identify the most important words, sentences or concepts it contains in order to build a summary of it, for instance, to use as a document surrogate to present to the searcher. Another post-retrieval operation is information visualization [11], in which a visual display is generated rather than, or in addition to, a ranked list. In some cases the visual display can be manipulated to provide further interpretation of the retrieval results.

Experimental Results

Almost as soon as computerized information retrieval systems were conceived, evaluation was seen as an important component. The best known of the early tests were the Cranfield experiments [8], so-called because they were conducted by Cyril Cleverdon and a group of researchers at the Cranfield College of Aeronautics, primarily as a test of indexing techniques. The first set of experiments, conducted in 1958–1962, provided controversial results, and in response to criticism of the methodology, a second set of experiments was devised by Cleverdon, with emphasis on rigor and a laboratory model. These second experiments, known as Cranfield II, led to the basic model for information retrieval experimentation in common use today: a document collection, a set of queries and associated relevance judgments, and measurement based on precision and recall. In terms of findings, the Cranfield experiments and a series of experiments on the SMART system [8] showed that the interest of the time in complex indexing systems was misguided, and that in general simpler indexing systems worked as well as more complex techniques. The laboratory model as exemplified by the Cranfield experiments made it possible to isolate individual information retrieval operations and evaluate performance with, for instance, different values or functions.

Through the 1970s and 1980s, the laboratory for information retrieval research expanded, with new test collections, query sets and relevance judgments. The collections grew steadily larger, though still falling far short of those found in operational systems. After 30 years of IR experimentation in the Cranfield model, there was confidence within the IR community that the basic information retrieval operations had been refined to achieve real performance improvements, although the transfer of the technology to the commercial section was extremely limited.

One of the strongest arguments for the lack of commercial success was skepticism about the scalability of performance improvements from the laboratory to large scale systems. Partly in response to this criticism, in 1992 the National Institute of Standards and Technology (NIST) hosted the first Text REtrieval Conference (TREC) [9]. TREC provided the infrastructure for large-scale IR evaluation, and resulted in the improvement and dissemination of many information retrieval operations, notably term weighting, as reports of the success of the BM25 model for term weighting were widely disseminated. TREC also supported the development of standard routines to analyze the results of query runs, minimizing the variance in analysis of data that existed, and making it easier to compare variations in information retrieval operations.

Cross-references

▶ Clustering for Post Hoc Information Retrieval
▶ Index Creation and File Structures
▶ Lexical Analysis of Textual Data
▶ Query Expansion for Information Retrieval
▶ Relevance Feedback
▶ Similarity and Ranking Operations
▶ Stemming
▶ Stopwords
▶ Summarization
▶ Visualization for Information Retrieval

Recommended Reading

1. Eckard E. and Chappelier J.-C. Free software for research in information retrieval and textual clustering, 2007. Available at: infoscience.epfl.ch/record/115460/files/Free_sofware_for_IR.pdf

2. Frakes W.B. and Baeza-Yates R. Information Retrieval: Data Structures and Algorithms. Englewood Cliffs, Prentice Hall, NJ, 1992.
3. Korfhage R.R. Information Storage and Retrieval. Wiley, New York, 1997.
4. Luhn H.P. The automatic creation of literature abstracts. IBM J. Res. Dev., 2:157–165, 1958. Available at: http://www.research.ibm.com/journal/rd/022/luhn.pdf
5. Luhn H.P. Keyword-in-context index for technical literature. Am. Documentation, 11(4):288 (8p), 1960.
6. Manning C.D., Raghavan P., and Schütze H. Introduction to Information Retrieval. Cambridge University Press, Cambridge, UK, 2008.
7. Salton G. The SMART Retrieval System: Experiments in Automatic Document Processing. Prentice-Hall, Englewood Cliffs, NJ, 1971.
8. Sparck Jones K. Retrieval system tests 1958–1978, In: Information Retrieval Experiment, K. Sparck Jones, (ed.). Butterworths, London, 1981. pp. 213–255.
9. Voorhees E.M. and Harman D.K. (eds.). TREC: Experiment and evaluation in Information Retrieval. MIT Press, Cambridge, MA, 2005.
10. Witten I.H., Moffat A., and Bell T.C. Managing Gigabytes: Compressing and Indexing Documents and Images (2nd edn.). Morgan Kaufmann, San Francisco, CA, 1999.
11. Zhang J. Visualization for Information Retrieval. Springer, New York, 2008.

Information Retrieval Processing

▶ Information Retrieval Operations

Information Seeking

▶ Information Foraging

Information Visualization

▶ Data Visualization

Information visualization on hierarchies

▶ Visualizing Hierarchical Data

Information Visualization on Networks

▶ Visualizing Network Data

INitiative for the Evaluation of XML Retrieval

Gabriella Kazai
Microsoft Research Cambridge, Cambridge, UK

Synonyms

INEX; XML information retrieval; Evaluation forum

Definition

The INitiative for the Evaluation of XML retrieval (INEX), launched in 2002, is an established evaluation forum for XML Information Retrieval (IR) with over 90 participating organizations worldwide. The initiative is sponsored by the DELOS Network of Excellence for Digital Libraries and supported by the IEEE Computer Society.

INEX encourages research in XML IR by providing an infrastructure to evaluate the effectiveness of XML IR systems. The infrastructure takes the form of a large XML test collection, appropriate scoring methods, and a forum for participating organizations to compare their results. The construction of the test collection is a collaborative effort, where participating organizations contribute by providing test queries and relevance judgments on the collection of XML documents. The constructed test collection provides participants a means for comparative and quantitative experiments.

Historical Background

The motivation for INEX stems from the increasingly important role of XML IR in many information access systems (e.g., in digital libraries and on the Web) where content is a mixture of text, multimedia, and metadata, formatted according to the adopted W3C standard of the eXtensible Markup Language (XML).

XML offers the opportunity to exploit the internal structure of documents in order to allow for more precise access, providing more focused answers to users' requests. XML IR thus breaks away from the traditional retrieval unit of a document as a single large (text) block and aims to implement more focused retrieval strategies that return document components, i.e., XML elements, instead of whole documents in response to a user query. This focused retrieval approach is seen of particular benefit for information repositories containing long documents, or documents covering a wide variety of topics (e.g., books, user manuals, legal documents), where the user's effort to locate relevant content can be reduced by directing them to the most relevant parts of the documents. Providing effective access to XML based content is therefore a key issue for the success of these systems.

INEX aims to provide the means necessary for the evaluation of XML IR systems. It follows the predominant approach in IR of evaluating retrieval effectiveness using a test collection constructed specifically for that purpose.

A test collection usually consists of a set of documents, user requests (topics), and relevance assessments which specify the set of "right answers" for the requests.

In the field of IR, there have been several large-scale evaluation projects, including the Text REtrieval Conference (TREC) (http://trec.nist.gov/), the Cross-Language Evaluation Forum (CLEF) (http://www.clef-campaign.org/), and the National Institute of Informatics Test Collection for IR Systems (NTCIR) (http://research.nii.ac.jp/ntcir/), which resulted in established test collections and evaluation methodologies. These traditional IR test collections and methodology, however, cannot be directly applied to the evaluation of content-oriented XML retrieval as they do not consider structural aspects and, for example, provide relevance judgments only at the unit of the document. Furthermore, the evaluation is based on assumptions that do not hold in XML IR. For example, the evaluation of IR systems treats documents as independent and well-distinguishable separate units of approximately equal size. XML IR, however, allows for varying sized document components to be retrieved. It is also possible that multiple elements from the same document are retrieved, which cannot be viewed as independent units.

The evaluation of XML retrieval systems thus makes it necessary to build test collections and develop appropriate metrics where the evaluation paradigms are provided according to criteria that take into account the imposed structural aspects. INEX aims to address these goals.

Foundations

Since its launch in 2002, INEX has grown both in terms of number of participants and with respect to its coverage of the investigated retrieval tasks. Throughout the years, INEX faced a range of challenges regarding the evaluation of XML IR approaches. These include the question of suitable relevance criteria, feasible assessment procedures, and appropriate evaluation measures. Different theories and methods for the evaluation of XML IR were developed and tested at INEX, leading to a now stable evaluation setup and a rich history of learned lessons.

In 2002, INEX started with 36 active participating organizations and a small collection of XML documents donated by the IEEE Computer Society, totaling 494 MB in size and containing over eight million XML elements [3, pp. 1–17]. INEX 2002 run a single track, investigating ad hoc retrieval applied to XML documents based on the focused retrieval approach. In IR literature, ad hoc retrieval is described as a simulation of how a library might be used, and it involves the searching of a static set of documents using a new set of topics [15]. While the principle is the same, the difference for INEX is that the library consists of XML documents, the queries may contain both content and structural conditions and, in response to a query, arbitrary XML elements may be retrieved from the library.

Two subtasks were defined within the Ad hoc track based on the query types of Content-Only (CO) and Content-And-Structure (CAS). In the CO task, it was left entirely to the retrieval system to identify the most appropriate relevant XML elements to return to the user, while in the CAS task, systems could make use of the structural clues specified by the user in the query.

The queries were created by the participating groups, contributing 30 CO and 30 CAS topics to the test collection. For each topic, a title, a description and a narrative were specified, where the syntax of the title allowed the definition of target elements and so-called containment conditions (content word and containment element pairs).

The relevance criteria was defined along two dimensions, each with four possible grades for assessors to choose from. Assessors were asked to assign scores for both dimensions to all XML elements of the collection that contained relevant information.

Based on the collected relevance judgments, effectiveness scores were calculated by adopting Raghavan's

Precall measure [3, pp. 1–17]. Table 1 shows summary information on INEX 2002.

In 2003, the CAS subtask was separated into the Strict CAS (SCAS) and the Vague CAS (VCAS) strands so that the effect of interpreting a query's structural clues strictly or vaguely could be studied. The CO subtask remained unchanged, and so did the document collection.

The syntax of the topic title was modified based on the XPath standard (http://www.w3.org/TR/xpath), where a new *about()* function was introduced [4, pp. 192–199].

Due to the fact that the coverage dimension of the relevance criterion was found to be susceptible to misinterpretation, INEX 2003 renamed and redefined the relevance dimensions to Exhaustivity and Specificity.

A new measure, inex_eval_ng [10], was also introduced, allowing to take into account the possible overlap (e.g., paragraph and its container section) and the varying size of the XML elements. Table 2 shows summary information on INEX 2003.

By 2004, INEX had 43 active participating groups and four additional tracks: Relevance feedback, Heterogeneous collection, Natural language processing (NLP) and Interactive tracks. The ad hoc track ran only two of the subtasks defined in 2003: CO and VCAS. The format of a topic's title field was formally defined using the Narrowed Extended XPath I (NEXI) language. The purpose of the Relevance feedback track was to explore issues related to the use of relevance feedback in a structured environment [1]. The Heterogeneous track aimed to address challenges where collections of XML documents from different sources and with different DTDs or Schemas were to be searched. The NLP track investigated whether it was practical to use a natural language query in place of the formal NEXI topic title used in the Ad hoc track [9]. The Interactive track focused on studying the behavior of searchers when presented with components of XML documents that have a high probability of being relevant (as estimated by an XML IR system) [12].

Both the document collection and the relevance dimensions remained unchanged from 2003. The measure of Precall was used as in previous years to report the retrieval effectiveness scores of the participating search systems. Table 3 shows summary information on INEX 2004.

The ad hoc track at INEX 2005 continued studying the role of structure in user queries, and defined four separate strands of the VCAS subtask based on the strict

INitiative for the Evaluation of XML Retrieval. Table 1. Summary information on INEX 2002

INEX 2002	
Organizers:	N. Fuhr, N. Gövert, G. Kazai, M. Lalmas
Participants:	36 active (49 from 21 countries signed up)
Tracks:	Ad hoc retrieval with two subtasks
Ad hoc tasks:	Ad hoc retrieval based on CO (content-only) and CAS (content-and-structure) topics
Document corpus:	12,107 articles from IEEE Computer Society, 1995–2002
	Totaling 494 MB and over eight million XML elements
Topics:	30 CO and 30 CAS topics
Relevance:	Topical relevance (Irrelevant, Marginal, Fairly, Highly relevant),
	Component coverage (No coverage, Tool large, Too small, Exact)
Metrics:	Precall (inex_eval) [3, pp. 1–17]
Proceedings:	[6]

INitiative for the Evaluation of XML Retrieval. Table 2. Summary information on INEX 2003

INEX 2003	
Organizers:	N. Fuhr, M. Lalmas
Participants:	30 active (40 from 18 countries signed up)
Tracks:	Ad hoc retrieval with three subtasks
Ad hoc tasks:	CO (content-only), SCAS (strict content-and-structure) and VCAS (vague content-and-structure)
Document corpus:	Same as in 2002
Topics:	36 CO and 30 CAS topics
Relevance:	Exhaustivity (Not, Marginally, Fairly, Highly exhaustive),
	Specificity (Not, Marginally, Fairly, Highly specific)
Metrics:	Precall (inex_eval) [3, pp. 1–17], and inex_eval_ng [10]
Proceedings:	[4]

INitiative for the Evaluation of XML Retrieval. Table 3. Summary information on INEX 2004

INEX 2004	
Organizers:	Overall: N. Fuhr, M. Lalmas
	Topic format: B. Sigurbjörnsson, A. Trotman
	Relevance assessment tool: B. Piwowarski
	Metrics: G. Kazai, A.P. de Vries
Participants:	43 active (55 from 20 countries signed up)
Tracks:	Ad hoc retrieval with two subtasks
	Relevance feedback track (C. Crouch, M. Lalmas)
	Heterogeneous collection track (T. Rölleke, Z. Szlávik)
	Natural language processing track (S. Geva, T. Sahama)
	Interactive track (A. Tombros, B. Larsen, S. Malik)
Ad hoc tasks:	CO (content-only) and VCAS (vague content-and-structure)
Document corpus:	Same as in 2002 and 2003
Topics:	40 CO and 35 CAS topics
Relevance:	Same as in 2003
Metrics:	Precall (inex_eval) [3, pp. 1–17]
Proceedings:	[6]

or vague interpretations of the structural conditions of a query [13]. The CO subtask has also diversified into six strands based on a combination of three subtasks (Focused, Thorough and FetchBrowse) and the use of the CO or CO+S (CO+Structure) type topics. The latter type expands the CO topic format with an additional CAS title field, where structural hints for the CO title can be expressed. The Focused task asked systems to return a ranked list of the most focused (specific and exhaustive) document parts, without returning overlapping elements. The Thorough task required systems to estimate the relevance of all XML elements in the searched collection and return a ranked list of the top 1,500 elements. The FetchBrowse task asked systems to return to the user the most focused, relevant XML elements clustered by the unit of the document containing the elements. Put another way, the task was to return documents with the most focused relevant elements highlighted within them.

All additional tracks started in 2004 run again in 2005, together with the new Document mining and Multimedia tracks. The Document mining track focused on the tasks of classification and clustering by developing methods that are able to exploit the XML markup for this purpose [2]. The Multimedia track was set up aiming at the evaluation of structured document retrieval approaches which are able to combine the relevance of the different media types into a single (meaningful) ranking that is presented to the user [14].

INEX 2005 obtained additional resources in the form of additional XML articles from the IEEE Computer Society, increasing the total size of the collection to 764 MB. In addition, the Multimedia track made use of an XML version of the Lonely Planet collection.

Other changes to the evaluation framework included new assessment procedures and new metrics. The assessment process was simplified to asking assessors to first highlight relevant passages and then assess the elements overlapping these passages. As a consequence, the Specificity dimension could be automatically measured on a continuous scale [0,1] by calculating the ratio (in characters) of the highlighted text (i.e., relevant information) and the total length of the element.

To report effectiveness scores, INEX 2005 adopted the eXtended Cumulated Gain (XCG) measures [5, pp. 16–29], which were developed specifically for graded (non-binary) relevance values and with the aim to allow XML IR systems to be credited according to the retrieved elements' degree of relevance. Table 4 shows summary information on INEX 2005.

In 2006, INEX finished with 50 active participating organizations and expanded to a total of nine tracks: Ad hoc, Relevance feedback, Heterogeneous collection, Natural language processing, Interactive, Multimedia, Document mining, Use case, and Entity ranking tracks. The Ad hoc track consisted of four subtasks: Focused, Thorough, Relevant in Context (FetchBrowse in 2005), and Best in Context tasks. The new Best in Context task asked systems to return a single best entry point (BEP) to the user per relevant document. Rather than dealing with information access to XML elements, the new Entity ranking track set as its task the retrieval of a list of entities of specific types (e.g., people, products, artifacts). The Use case track attempted to identify examples of how XML IR systems can be exploited by end-users for various purposes [11].

A major change in 2006 was the departure from the use of the IEEE document collection, which has been replaced by a collection of XML articles from the Wikipedia project.

INEX 2006 has also further simplified the assessment procedure by dropping the exhaustivity dimension of the relevance criteria.

INitiative for the Evaluation of XML Retrieval. Table 4.
Summary information on INEX 2005

INEX 2005	
Organizers:	Overall: N. Fuhr, M. Lalmas
	Topic format: B. Sigurbjörnsson, A. Trotman
	Relevance assessment tool: B. Piwowarski
	Metrics: G. Kazai, A.P. de Vries, P. Ogilvie, B. Piwowarski
Participants:	41 active (47 signed up)
Tracks:	Ad hoc retrieval with ten subtasks
	Relevance feedback track (Y. Mass, C. Crouch)
	Heterogeneous collection track (R. Larson)
	Natural language processing track (S. Geva, T. Sahama)
	Interactive track (B. Larsen, A. Tombros, S. Malik)
	Multimedia track (R. van Zwol, G. Kazai, M. Lalmas)
	Document mining track (L. Denoyer, A-M. Vercoustre, P. Gallinari)
Ad hoc tasks:	CO.Focused, CO.Thorough, CO. FetchBrowse,
	COS.Focused, COS.Thorough, and COS.FetchBrowse,
	VVCAS (vague target and vague containment CAS),
	SVCAS (strict target and vague containment CAS),
	VSCAS (vague target and strict containment CAS), and
	SSCAS (strict target and strict containment CAS)
Document corpus:	16,819 articles from IEEE Computer Society, 1995–2004
	Totaling 764 MB, and over 11 million XML elements
	Includes an additional 4,712 new articles
Topics:	40 CO+S and 47 CAS topics
Relevance:	Exhaustivity (Not, Somewhat, Highly exh., Too small),
	Specificity (Continuous scale)
Metrics:	XCG metrics [6, pp. 16–29]
Proceedings:	[6]

A new passage-based recall and precision was adopted to report effectiveness scores for the Relevant in Context task, while XCG was employed for the Focused and Thorough tasks [7, pp. 20–34]. Two further measures, BEP-distance and EPRUM [7, pp. 20–34], provided the performance results for the Best in Context task. Table 5 shows summary information on INEX 2006.

INitiative for the Evaluation of XML Retrieval. Table 5.
Summary information on INEX 2006

INEX 2006	
Organizers:	Overall: N. Fuhr, M. Lalmas
	Wikipedia collection: L. Denoyer, M. Theobald
	Topic format: A. Trotman, B.Larsen
	Task description: J. Kamps, C. Clarke
	Relevance assessment tool: B. Piwowarski
	Metrics: G. Kazai, S. Robertson, P. Ogilvie
Participants:	50 active (68 signed up)
Tracks:	Ad hoc retrieval with four subtasks
	Relevance feedback track (Y. Mass, R. Schenkel)
	Heterogeneous collection track (I. Frommholz, R. Larson)
	Natural language processing track (S. Geva, X. Tannier)
	Interactive track (B. Larsen, A. Tombros, S. Malik)
	Multimedia track (R. van Zwol, T. Westerveld)
	Document mining track (L. Denoyer, A-M. Vercoustre, P. Gallinari)
	Use case track (A. Trotman, N. Pharo)
	Entity ranking track (A.P. de Vries, N. Craswell)
Ad hoc tasks:	Focused, Thorough, Relevant in Context, and
	Best in Context
Document corpus:	659,388 articles of the Wikipedia project, covering a hierarchy of 113,483 categories, totaling over 60 GB (4.6 GB without images) and 30 million XML elements
Topics:	125 CO+S topics
Relevance:	Specificity (Continuous scale)
Metrics:	XCG metrics [6, pp. 16–29], passage-based recall and precision,
	BEP-distance and EPRUM [7, pp. 20–34]
Proceedings:	[7]

For INEX 2007, 100 groups registered to participate. A total of six track were run in 2007: The Ad hoc, Document mining, Multimedia, and Entity ranking tracks were continued, and two new tracks were started: Link the Wiki, and Book search. The Ad hoc track pitted XML element retrieval approaches against passage retrieval methods on three tasks: Focused, Relevant in Context and Best in Context. The Link the Wiki track aims at evaluating the state of the art in automated discovery of document hyperlinks. The Book search track builds on a collection of over 40,000 digitized books, marked up in XML. It aims to investigate book-specific relevance ranking strategies, user interface issues and user behavior, exploiting special features, such as back of book indexes provided by authors, and linking to associated metadata like catalogue information from libraries.

There were no changes in the document collection, which remained the Wikipedia XML corpus, and the relevance assessment criteria and procedures. The metrics from 2006 have been refined to allow for the evaluation of arbitrary passages as retrieval results and a new measure generalized precision and recall has also been introduced to measure retrieval effectiveness for the Relevant in Context task [8, INEX 2007 Evaluation Measures]. Table 6 shows summary information on INEX 2007.

INEX 2008 is to set to start in the Spring of 2008.

INitiative for the Evaluation of XML Retrieval. Table 6. Summary information on INEX 2007

INEX 2007	
Organizers:	Overall: N. Fuhr, A. Trotman, M. Lalmas
	Wikipedia collection: L. Denoyer
	Collection exploration: R. Schenkel, M. Theobald
	Topic format: B.Larsen, A. Trotman
	Task description: J. Kamps, C. Clarke
	Relevance assessment tool: B. Piwowarski
	Metrics: G. Kazai, B. Piwowarski, J. Kamps, J. Pehcevski, S. Robertson, P. Ogilvie
Participants:	100 signed up
Tracks:	Ad hoc retrieval with six subtasks
	Document mining track (L. Denoyer, P. Gallinari)
	Multimedia track (T. Westerveld, T. Tsikrika)
	Entity ranking track (A.P. de Vries, N. Craswell, M. Lalmas, J.A. Thom, A-M. Vercoustre)
	Link the Wiki track (S. Geva, A.Trotman)
	Book search track (G. Kazai, A. Doucet)
Ad hoc tasks:	Focused, Relevant in Context, and Best in Context using either XML element retrieval or passage retrieval approaches
Document corpus:	Same as in 2006
Topics:	130 CO+S topics
Relevance:	Same as in 2006
Metrics:	Passage-based recall and precision, generalized precision and recall, and BEP-distance [8, INEX 2007 Evaluation Measures]
Proceedings:	[8]

Key Applications

Evaluation is a key component of any system development as it allows to quantify improvement in performance. INEX provides an important resource to facilitate the evaluation of XML IR systems.

XML IR is a form of semi-structured text retrieval, which aims to exploit the inherent structure of documents to improve their retrieval, where the structure is given by the XML markup.

Some of the issues and proposed solutions within INEX are applicable to other areas of IR, such as passage, video and Web retrieval, where there is no fixed unit of retrieval and where the evaluation needs to handle overlapping fragments and users' post query browsing behavior.

Data Sets

Until 2004, the document collection consisted of 12,107 articles, marked-up in XML, from 12 magazines and 6 transactions of the IEEE Computer Society's publications, covering the period of 1995–2002, and totaling 494 MB in size, consisting of over eight million XML elements. On average, an article contains 1,532 XML nodes, where the average depth of the node is 6.9.

In 2005, the collection was extended with further publications from the IEEE Computer Society. A total of 4,712 new articles from the period of 2002–2004 were added, giving a total of 16,819 articles, and totaling 764 MB in size and over 11 million XML elements.

The overall structure of a typical article in the IEEE collection consists of a front matter, a body, and a back matter. The front matter contains an article's metadata, such as title, author, publication information, and abstract. The article's body contains the actual content of the article. The body is structured into sections, subsections, and sub-subsections. These logical units start with a section title, followed by a number of paragraphs. In addition, the content has markup for references (citations, tables, figures), item lists, and layout (such as emphasized and bold faced text), etc. The back matter contains a bibliography and further information about the authors.

INEX 2006 and 2007 switched to a different document collection, consisting of 659,388 English articles, marked-up in XML, from the Wikipedia (http://en.wikipedia.org) project, totaling over 60 GB (4.6 GB without images) and 30 million XML elements. The collection's structure is similar to that of the IEEE collection's. On average, a Wikipedia article contains 161.35 XML nodes, where the average depth of an element is 6.72.

In addition to these, the different tracks worked with additional document collections. For example, the Multimedia track in 2005 made use of an XML version of the Lonely Planet collection and the Book Search track in 2007 provided a collection of 42,000 digitized books marked up in XML.

URL to Code

http://inex.is.informatik.uni-duisburg.de/

Cross-references

- ▶ Content-and-Structure Query
- ▶ Content-Only Query
- ▶ Evaluation Metrics for Structured Text Retrieval
- ▶ Narrowed Extended XPath I
- ▶ Presenting Structured Text Retrieval Results
- ▶ Processing Overlaps
- ▶ Relevance
- ▶ Specificity
- ▶ XML

Recommended Reading

1. Crouch C. Relevance feedback at the INEX 2004 workshop. SIGIR Forum 39(1):41–42, June 2005.
2. Denoyer L. and Gallinari P. Report on the XML mining track at INEX 2005 and INEX 2006: categorization and clustering of XML documents. SIGIR Forum 41(1):79–90, June 2007.
3. Fuhr N., Gövert N., Kazai G., and Lalmas M. (eds.). In Proc. 1st Workshop of the INitiative for the Evaluation of XML Retrieval, 2002.
4. Fuhr N., Lalmas M., and Malik S. (eds.). Proc 2nd Workshop of the INitiative for the Evaluation of XML Retrieval, 2003.
5. Fuhr N., Lalmas M., Malik S., and Kazai G. (eds.). Advances in XML Information Retrieval and Evaluation. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005.
6. Fuhr N., Lalmas M., Malik S., and Szlávik Z. (eds.). Advances in XML Information Retrieval. In Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2004.
7. Fuhr N., Lalmas M., and Trotman A. (eds.). Comparative Evaluation of XML Information Retrieval Systems, In Proc. 5th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006.
8. Fuhr N., Lalmas M., Trotman A., and Kamps J. (eds.). Focused access to XML documents. In Proc. 6th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2007.
9. Geva S. and Sahama T. The NLP task at INEX 2004. SIGIR Forum 39(1):50–53. June 2005.
10. Gövert N., Fuhr N., Lalmas M., and Kazai G. Evaluating the effectiveness of content-oriented XML retrieval methods. Inform. Retri., 9(6):699–722, 2006.
11. Pharo N. and Trotman A. The use case track at INEX 2006. SIGIR Forum 41(1):64–66, June 2007.
12. Tombros A., Malik S., and Larsen B. Report on the INEX 2004 interactive track. SIGIR Forum 39(1):43–49, June 2005.
13. Trotman A. and Lalmas M. The interpretation of CAS. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 58–71.
14. van Zwol R., Kazai G., and Lalmas M. The Multimedia Track at INEX 2005: Overview, Advances in XML Information Retrieval and Evaluation. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005.
15. Voorhees E.M. and Harman D.K. TREC: Experiment and Evaluation in Information Retrieval (Digital Libraries and Electronic), MIT, Cambridge, MA.

Initiator

Kaladhar Voruganti
Network Appliance, Sunnyvale, CA, USA

Synonym

SCSI initiator

Definition

In SCSI protocol, the client which requests data is known as the initiator. The initiator typically resides on the host server that is accessing the storage. Initiators can also reside in storage virtualization boxes which can act as both initiators and targets.

Key Points

The SCSI initiator can reside in software or in a hardware adapter card. SCSI protocol provides commands that allow initiators to reserve LUNs and prevent them from being accessed by other initiators. SCSI initiators communicate with SCSI targets using SCSI block level protocol. The block level protocol data is transported via other transport protocols such as parallel SCSI, Fiber Channel, Serial SCSI or Infiniband.

Cross-references

- ▶ Storage Protocols
- ▶ Target

In-Memory DBMS

- ▶ Main Memory DBMS

In-Network Aggregation

- ▶ Query Optimization in Sensor Networks

In-Network Query Processing

Samuel Madden
Massachussetts Institute of Technology, Cambridge, MA, USA

Synonyms

Tiny aggregation (TAG); TinyDB

Definition

In-Network query processing refers to the complete or partial evaluation of database queries at the edges of a network, rather than in a centralized database server. Though this phrase may also apply to the general process of distributed or parallel query evaluation, it is most commonly applied to environments like sensor networks, where the network edges consist of small, wireless devices with power and CPU constraints. Typically these devices produce the data to be processed via local sensors, so processing in the edges of the network can be beneficial because it can reduce bandwidth usage, which in turn can conserve energy.

Historical Background

Sensor networks are collections of small, inexpensive battery-powered, wirelessly networked devices equipped with sensors (microphones, temperature sensors, etc.) that offer the potential to monitor the world with unprecedented fidelity. To capture data from these networks, researchers have proposed several sensor network database systems, including TinyDB [11], Cougar [15], and SwissQM [12] have been proposed. These systems provide a high level SQL-like query language that allows users to specify what data they would like to capture from the network.

The canonical sensor network platform is the Berkeley Mote hardware running the TinyOS operating system [8]. Initial versions of the motes used Atmel 8-bit microprocessors and 40 kbit/s radios; newer generations, developed by companies like Crossbow Technologies (http://www.xbow.com) and Moteiv Technologies (http://www.moteiv.com) use Zigbee (802.15.4) radios running at 250 kbits/s and Atmel or Texas Instruments 8 or 16 bit microprocessors running at 4–8 MHz. Nodes typically are very memory constrained (with 4–10 KB of RAM and 48–128 KB of non-volatile flash-based program memory.) Most varieties can be interfaced to sensors that can capture a variety of readings, including light, temperature, humidity, vibration, acceleration, sounds, or images. The limited processing power and radio bandwidth of these devices constrains sample rates to at most a few kilo-samples per second. The dominant power cost of operating these nodes is often cited as network transmissions [9,10]. Table 1 summarizes the major operations in a sensor network, and illustrates the reason why communication costs are considered dominant: the time spent sending messages is huge compared to the time spent computing or sampling. Therefore, the total energy consumption is dominated by message transmissions (in this example, communication cost is 50 times the combined cost to sample a sensor and evaluate a predicate).

Given these high per-message energy costs, one primary goal of sensor network database systems is to reduce message transmissions; in network processing is key to do this, as explained in the next section.

Foundations

Before describing the various in-network processing techniques that have been proposed, it is important

In-Network Query Processing. Table 1. Major activities in a data collection network, with time, power, and energy for each. Computation is assumed to be one sample, one message transmission, and one predicate evaluation per second (with no reception.) The sensor is assumed to be attached directly to the on-chip ADC. The sensor node is a MicaZ-class [4] mote with Atmel Atmega 128L microprocessor at 4 MHz and a TI/ChipCon CC 2420 250 kbps ZigBee radio running at 0 dBm. Predicate evaluation is assumed to take 400 cycles; sensor sampling is assumed to take 40 cycles

Activity	Time per action	Power	Energy
Send a message	3 ms	60 mW	180 μJ
Sample a sensor	10 μs	24 mW	0.24 μJ
Evaluate a predicate	100 μs	24 mW	2.4 μJ
Idle	899 ms	45 μW	40 μJ

to understand the basics of sensor network query languages and the usage model for the systems.

Network Formation and Communication

Typically, a user deploys one of these systems by placing a collection of static sensor nodes around an area to be monitored. These nodes contain a pre-compiled binary image of the sensor network database, but are not yet running any queries. When powered up, the nodes immediately begin to organize themselves into an ad-hoc network which typically takes the form a *spanning tree* rooted at a *root node* connected to a *basestation*. The basestation is usually a more powerful machine, such as a laptop PC, which issues queries to nodes and collects, processes, and visualizes query results.

Tree formation is accomplished by have the basestation periodically broadcast a beacon message. Nodes that hear this beacon re-broadcast it, indicating that they are one hop from the basestation. Nodes that hear those messages in turn re-broadcast them, indicating that they are two hops from the basestation, and so on. This process of (re)broadcasting beacons occurs continuously, such that (as long as the network is connected) all nodes will eventually hear a beacon message. When a node hears a beacon message, it chooses a node from which it heard the message to be its *parent*, sending messages through that parent when it needs to transmit data to the basestation. (In general, parent selection is quite complicated, as a node may hear beacons from several candidate parents. Early papers by Woo et al. [14] and DeCouto et al. [5] provide details.)

Users interact with the system by issuing queries at the basestation, which in turn broadcasts queries out into the network. Queries are typically disseminated via flooding down the routing tree. As nodes receive the query, they begin processing it. The basic programming model is data-parallel: each node runs the same query over data that it locally produces or receives from its neighbors. As nodes produce query results, they apply in-network processing to reduce query results, and then send those reduced result up the routing tree, towards the basestation, which receives the results and possibly merges them together to form a final query answer.

Query Language and Data Model

Most sensor network databases systems provide a SQL-like query interface. For example, the a TinySQL query (used in TinyDB) that requests the temperature from every node in a sensor network whose value is greater than 25°C once per second would look like:

```
SELECT nodeid, temperature
FROM sensors
WHERE temperature > 25°C
SAMPLE PERIOD 1s
```

This query is essentially standard SQL, with a few small additions. First, the `SAMPLE PERIOD` clause requests that a data reading be produced once every second. This means that each query produces a continuous stream of results rather than a single result set. Second, the `nodeid` attribute is a unique identifier assigned to each sensor node and available in every query. Third, the table `sensors` is *virtual table* of sensor readings. Here, *virtual* means that it conceptually contains one row for every sensor type (light, temperature, etc.) from every sensor node at every possible instant, but all of those rows and columns are not actually materialized. Instead, only the sensor readings needed to answer a particular query are actually generated.

Note also that although this table appears to be a single logical table its rows are actually produced by different, physically disjoint sensors. This is the key to in-network processing: when a sensor receives a query, it begins sampling its sensors at the appropriate rate, applying predicates and aggregating data locally and forwarding on only those readings that will eventually be a part of the final query answer.

It should be clear that selection predicates over a single attribute of the sensors table can always be evaluated locally, inside of the network, before any network transmission is done. The remainded of this section discusses techniques that have been developed for in-network processing for aggregate and join queries.

In-Network Processing of Aggregates

Aggregate queries are particularly common in sensor networks, since users are often more interested in what is happening in general geographic regions rather than at a specific sensor. A common environmental monitoring query might ask for the temperature in a building grouped by room number:

```
SELECT roomNo, AVG(temp)
FROM sensors
GROUP BY roomNo
SAMPLE PERIOD 1 s
```

A naive implementation of sensor network aggregation would be to use a centralized, *server-based* approach where all sensor readings are sent to the base station, which then computes the aggregates. In the TAG system [10], however, the authors proposed computing aggregates in-network, and showed that this approach requires fewer message transmissions, is lower latency, and uses less power than the server-based approach.

Consider first the case of an aggregation query without group. Once the query has been disseminated into the network, the TAG data processing phase begins. In this phase, aggregate values are continually routed up from children to parents. The goal of the TAG algorithm is to produce a single aggregate value (or a single value per group) that combines the readings of all devices in the network once per sample period.

The basic insight of the TAG protocol is that it is possible to *partially aggregate* sensor values together at intermediate points inside the network. TAG accomplishes this by having each parent node collect readings from its children before doing its own transmission. Rather than simply forwarding all of the raw data, each parent node combines its data with data from its children to produce a compact *partial state record* (PSR). For example, suppose the user wants to compute the average temperature in the network. In TAG, the PSR representation for an average is a <*sum*, *count*> pair. Each node transmits exactly one PSR per sample period. Suppose a node receives a set P of n PSRs from its children, such that $P = p_1,\ldots,p_n$. If the node has the local sensor value v; then it can compute its own partial state record as:

$$sum = v + \sum_{i=1}^{n} p_i \cdot sum$$

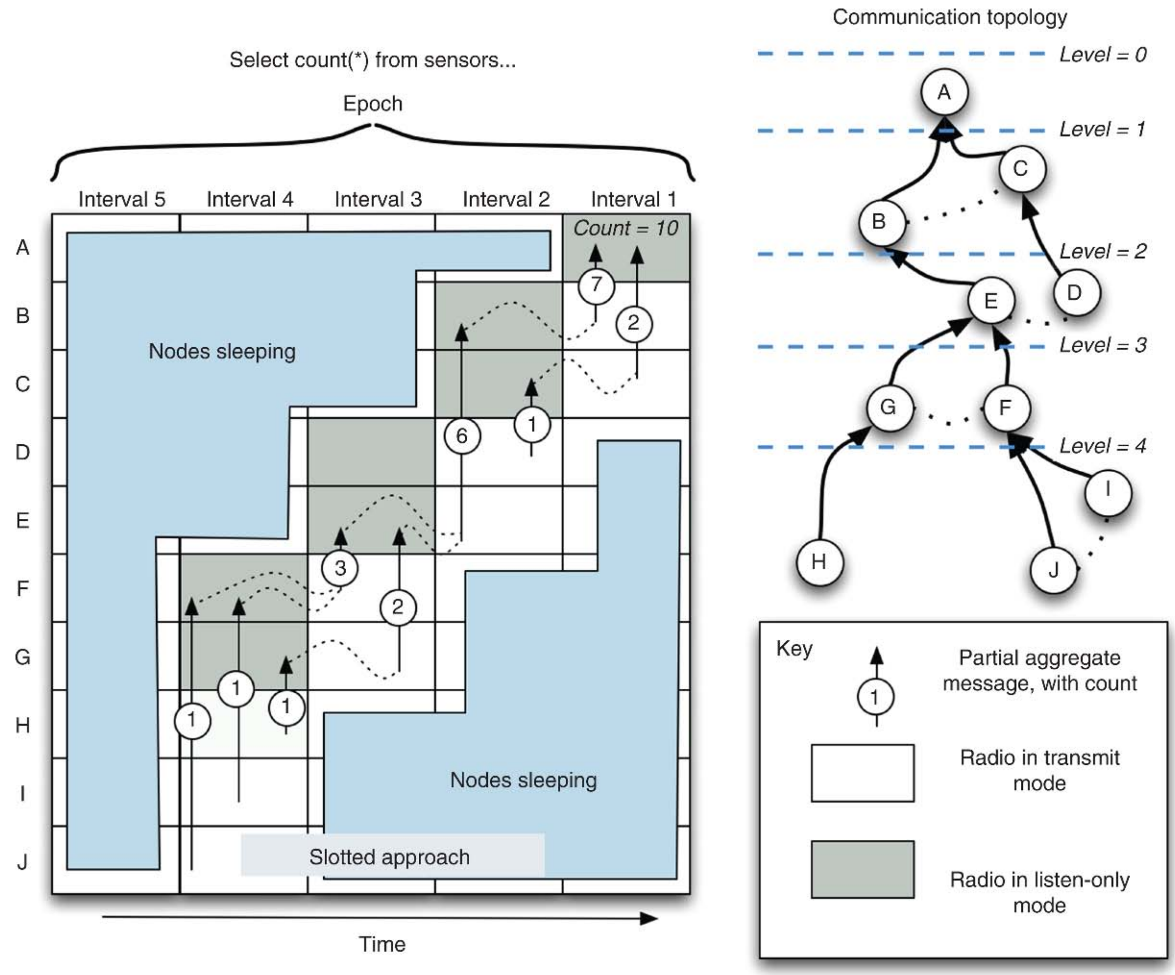
$$count = n + 1$$

By this definition, if a node has no children, it transmits the PSR <v, 1>. Finally, at the root of the network, the final average can be computed from the root's PSR as *PSR.sum/PSR.count*. To support the addition of new aggregation functions to the system, TAG includes a facility that allows programmers to define functions that initialize, merge, and compute the final value of partial state records for different aggregation functions.

Clearly, this technique reduces both the number of messages and total number of bytes that must be sent in networks of even modest size, as a parent with k nodes below it must transmit k readings without the TAG technique but only two values with the TAG approach.

For tag TAG-like protocols to work, nodes must nodes wait to receive readings from children. TAG does this by sub-dividing each sample period up into a series of time intervals corresponding to transmission slots, and assign nodes deeper in the routing tree to an earlier interval. This approach will work as long as sample periods are relatively long (on the order of half a second or so for Mica motes) and as long as it is possible to time synchronize nodes (using a technique like RBS [6]). The TinyDB system demonstrated a proof-of-concept implementation of this technique.

Figure 1 illustrates this in-network aggregation scheme for a simple `COUNT` query that reports the number of nodes in the network. In the figure, time advances from left to right, and different nodes in the communication topology are shown along the Y axis. Nodes transmit during the interval corresponding to their depth in the tree, so `H, I,` and `J` transmit first, during interval 4, because they are at level 4. Transmissions are indicated by arrows from sender to receiver, and the numbers in circles on the arrows represent `COUNT`s contained within each partial state record (which consists simply of a running count.) Readings from these three sensors are combined, by summing the running counts in the PSRs, at nodes `G` and `F`, both of



In-Network Query Processing. Figure 1. Partial state records flowing up the tree during and interval-based aggregation approach.

which transmit new partial state records during interval 3. Readings flow up the tree in this manner until they reach node A, which then computes the final count of 10. Notice that motes are idle for a significant portion of each sample period, during which time they can enter a low power sleeping state.

Supporting grouping in this setting can be done simply by tagging each partial state record with a group number (which can be derived by local evaluation of the grouping predicate at each sensor), and then treating each group as a separate aggregation operation that is independently merged and forwarded through the network.

For queries involving a HAVING clause, the TAG system proposes several optimizations for early in-network rejection of aggregates that do not satisfy the clause. The main observation is that in MIN/MAX queries it may be possible to determine that a particular group will definitely not satisfy the HAVING clause before the group reaches the root of the network.

Classes of Aggregates

One observation is that some aggregates will show more or less benefit from the TAG in-network processing techniques. In particular, if the partial state record is very compact (as in a COUNT or AVERAGE) query, there is a tremendous win to in-network aggregation. However, for aggregate functions that require access to all or most of the sensor readings before that final aggregate can be computed, the benefit is much less. For example, computing the exact median of a collection of readings requires all of the readings to be present. Hence, the TAG implementation of median offers no reduction in data transmission over a naive, centralized approach. (Greenwald and Khanna [7] propose techniques for approximate and efficient computation of order statistics like medians in sensor networks.)

There are other important semantic properties of aggregates. One that is of particular interest relates to the sensitivity of the aggregation function to duplicates. Some functions, like MAX and MIN, are insensitive to duplicates: even if a particular reading is merged together with a partial state record many times, the final value of the aggregate will not be affected. Other aggregates, like SUM, COUNT, and AVERAGE are obviously sensitive to duplicates. The TAG system exploits this property by using a directed acyclic graph that terminates at the network root rather than a simple spanning tree when computing duplicate insensitive aggregates. This substantially improves the reliability of the algorithms (using little additional energy since radios generally operate in a broadcast mode that allows multiple receivers to hear a message at no additional cost). Several researchers [3,13] have proposed methods that accurately approximate the value of duplicate sensitive aggregates using a duplicate insensitive synopsis data structure, allowing the same DAG-based network topologies to be used with a broader range of aggregates.

In-Network Processing of Joins

There have been several join algorithms proposed for specific classes of join queries in sensor networks.

Bonfils and Bonnet [2] view joins as a way to correlate or compare data between several sensors in a network. They propose a setting in which two sensors each produce a data stream, and the user wishes to apply a temporal join operation over these streams that combines readings from approximately the same time together when a predicate is satisfied. Clearly, such an operation can be done at the root of the network. They observe, however, that it can also be done at the root of any subtree in the network that has both producer nodes as a subchild. If the join is data-reducing (e.g., it filters out some readings or produces tuples that are smaller than the combined size of the tuples from both producers) then this should result in an overall reduction in network bandwidth. Rather than trying to compute the best location for such operators centrally (using global network topology information), they propose a distributed algorithm where the join operator slowly moves towards the nodes, tending to move closer to the node that is producing a larger fraction of the join data, since that will result in the greatest overall reduction in network bandwidth.

Abadi and Madden [1] propose a method called REED for in-network execution of joins that involve a static table of data from outside of the sensor network with a stream of sensor data. Such situations arise, for example, where there is a table of thresholds that dictate what data should be sent out of the network at different times of the day. The observation here is that if the table is static and the join is cardinality reducing, then paying the cost of disseminating the table once will save energy in the long run. When there is sufficient memory on each of the nodes to store the complete table, such joins can be evaluated purely locally at each node. The REED system proposes several techniques that can be used to execute queries in-network when the static table exceeds the memory available on any one node. The most effective techniques involve sending just a portion of the table to each node (e.g., the thresholds for 8 A.M. to 8 P.M.) and then sending the raw data out of the network when the needed portion is unavailable (e.g., when it is between 8 P.M. and 8 A.M.)

Key Applications

In-network processing of queries can be used in any setting where declarative queries over sensor networks are needed. In network processing techniques make selection, join, and aggregation queries substantially cheaper to run – often saving an order of magnitude in totally energy cost to process a given query. These savings mean that sensor network deployments can last longer, or provide higher sample rates for the same longevity when compared to solutions that do not use in-network processing.

Cross-references

- ▶ Database Languages for Sensor Networks
- ▶ Distributed Query Processing
- ▶ Partial Pre-aggregation

Recommended Reading

1. Abadi D. and Madden S. Reed: Robust, Efficient Filtering and Event Detection in Sensor Networks. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 769–780.
2. Bonfils B. and Bonnet P. Adaptive and Decentralized Operator Placement for In-network Query Processing. In Proc. 2nd Int. Workshop Inf. Proc. in Sensor Networks, 2003, pp. 47–62.

3. Considine J., Li F., Kollios G., and Byers J. Approximate Aggregation Techniques for Sensor Databases. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 449–460.
4. Crossbow, Inc. Micaz wireless sensor node data sheet. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf.
5. De Couto D.S.J., Aguayo D., Bicket J., and Morris R. A High-Throughput Path Metric for Multi-hop Wireless Routing. In Proc. 9th Annual Int. Conf. on Mobile Computing and Networking, 2003, pp. 134–146.
6. Elson J. and Estrin D. Time Synchronization for Wireless Sensor Networks. In Proc. 15th Int. Parallel & Distributed Processing Symp., 2001, pp. 1965–1970.
7. Greenwald M.B. and Khanna S. Power-Conserving Computation of Order-Statistics Over Sensor Networks. In Proc. 23rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2004, pp. 275–285.
8. Hill J., Szewczyk R., Woo A., Hollar S., Culler D., and Pister K. System Architecture Directions for Networked Sensors. In Proc. 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 2000, pp. 93–104.
9. Madden S. The Design and Evaluation of a Query Processing Architecture for Sensor Networks. PhD thesis, UC Berkeley, 2003.
10. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. In Proc. 5th USENIX Symp. on Operating System Design and Implementation, 2002.
11. Madden S., Hong W., Hellerstein J.M., and Franklin M. TinyDB web page. http://telegraph.cs.berkeley.edu/tinydb.
12. Müller R., Alonso G., and Kossmann D. SwissQM: Next Generation Data Processing in Sensor Networks. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 1–9.
13. Nath S. and Gibbons P.B. Synopsis Diffusion for Robust Aggregation in Sensor Networks. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 250–262.
14. Woo A., Tong T., and Culler D. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In Proc. 1st Int. Conf. on Embedded Networked Sensor Systems, 2003, pp. 14–27.
15. Yao Y. and Gehrke J. Query Processing in Sensor Networks. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.

Instance Identification

► Record Matching

Instance-Completeness

► BP-Completeness

Instant

► Chronon

Instant Relation

► Event in Temporal Databases

Instruction Cache

► Processor Cache

Integrated DB&IR Semi-Structured Text Retrieval

Ralf Schenkel[1], Martin Theobald[2]
[1]Max-Planck Institute for Informatics, Saarbrücken, Germany
[2]Stanford University, Stanford, CA, USA

Synonyms

Using efficient database technology (DB) for effective information retrieval (IR) of semi-structured text

Definition

Integrated DB&IR semi-structured text retrieval combines IR-style scoring and ranking methods for effective search with indexing techniques and processing algorithms from the database world for efficient query evaluation.

Historical Background

Database research has traditionally focused on semi-structured documents that represent structured data with a well-defined schema and only little unstructured, textual content (aka. "data-centric" XML). Typical examples for such documents are invoices, purchase orders, or even complete bibliographies.

Early work in the field concentrated on "classical" data management problems for XML: storing XML data in relational or native XML systems, defining query languages that integrate conditions on the structure and the content of results (like SQL for relational data), efficiently processing these queries on huge collections of documents, and auxiliary structures (like structural summaries and path indexes) to support processing. The focus of this work was on space and runtime efficiency.

Foundations

When semi-structured data formats, especially XML, became popular for storing and exchanging information throughout the 1990s, an abundance of different schemas for such data was developed independently. This created a challenge for existing database query languages like the Structured Query Language (SQL), focusing on exactly matching conditions on the content and structure of results. Now, similar structured and textual information was present in different, heterogeneous formats and schemas, which was often the case when information from different sources was integrated in a single application. In reaction to this, the strict, SQL-style querying paradigm prevalent at that time evolved towards a more relaxed IR-style vague search with partial and imprecise answers. This created three main scientific problems at the intersection of the DB and IR fields: (i) the definition of query languages to specify vague constraints on the structure and/or the content of results, (ii) the definition of relevance scores to rank results by their degree of matching with the query, and (iii) efficient algorithms and auxiliary structures to quickly compute the best results to a vague query, according to the relevance score. Solutions to these problems were developed mainly with data-centric documents in mind, and with a strong focus on query languages and algorithms (thus addressing problems i and iii).

One of the first systems to retrieve semi-structured data using a relaxed query language was the *Lore* [1] system developed at Stanford University. Its object-oriented, OQL-style, query language, coined *Lorel*, provided regular path expressions and tag wildcards to express structural vagueness, as well as keyword conditions over the content of subtrees matching the path condition. However, it was still more of a database query language as it did not yet foresee any ranking for the results.

Querying Semi-Structured Data with IR Support

A large body of proposals have been made for query languages over semi-structured data that support IR-style vague conditions on structure and content. The simplest of them merely aim to extend keyword search as known from text retrieval to semi-structured data by enhancing the keyword conditions with tag names of elements that should be matched (like in the query "`author:widom`" which restricts occurrences of the keyword "`widom`" to elements with tag name "`author`"). Matches to such a query are subtrees of the document that contain matches to all (for conjunctive evaluation) or at least one (for disjunctive evaluation) keyword. An important aspect here is the selection of subtrees of the right granularity, as large subtrees (such as the complete document) would often not be specific enough. The proposed solutions usually consider some variant of lowest common ancestor (LCA) search to identify suitable root nodes of the result trees, sometimes allowing for additional path conditions from elements containing the keywords towards the root element. Hardly any of the early proposed systems consider ranking of results; instead, they focus on efficient methods to retrieve all possible matches. Note that these techniques were primarily developed for data-centric XML (such as bibliographies) and cannot easily be applied for true full-text search over semi-structured documents.

Among the most prominent approaches for keyword-based ranked retrieval of XML data is XRank [7]. It generalizes traditional link analysis algorithms such as PageRank for authority ranking of linked XML collections and conceptually treats each XML element as an interlinked node in a large element graph. Then the *element rank* of an XML element corresponds to the PageRank value computed over a mixture of containment edges, obtained from the XML tree structure, and hyperlink edges, obtained from the inter-document link structure.

Full-fledged XML query languages with rich IR models for ranked retrieval were proposed by [6,13]. *XIRQL* [6], a pioneer in the area of ranked XML retrieval, presents a path algebra based on XQL, an early ancestor of W3C's XQuery, for processing and optimizing structured queries. It combines Boolean query operators with probabilistically derived weights for ranked result output, thus transferring the probabilistic IR paradigm to the XML case. It defines data-type-specific vague predicates for similarity search over differently typed XML elements such as person names or numbers, and it introduces a notion of *index objects* that serve as anchors from which the probabilistic weights are derived (in the classic IR notion of a document). Using index objects follows the idea that only nodes of specific type and granularity in the document hierarchy should be presented as results to the end-user. Defining these index objects, however, may be strongly schema-dependent and assumes substantial knowledge about the general document structure and user intent, which typically requires their manual pre-selection from a – preferably compact – document type definition (DTD).

The *XXL* search engine [13] specifies a full-fledged, SQL-oriented query language for ranked XML IR with a high semantic expressiveness that made it stand way apart from the predominant XQL and XPath language standards at its time. For ranked result output, XXL leverages both a standard IR vector space model and an ontology-oriented similarity search for the dynamic relaxation of structure and term conditions. The principal structure of the query, however, is evaluated in a strictly Boolean manner.

More recently, various groups from both the DB and IR fields have started adding IR-style keyword conditions and full-text search to existing XML query languages. The NEXI (for Narrowed Extended XPath I) query language used in the INEX (INitiative for the Evaluation of XML Retrieval (INEX), see http://inex.is.informatik.uni-duisburg.de) benchmark series aims at a simplified and easy-to-comprehend subset of XPath, the W3C standard language for path matches within a document, with extended IR functionality. Here the `about` operator already anticipates the role of the `ftcontains` operator in the later W3C Full-Text extensions of XPath 2.0 and XQuery 1.0. TeXQuery [2], on the other hand, has been the foundation for the W3C's official Full-Text extension to XPath and XQuery, which extends these languages with the option to express actual full-text queries over XML documents. It provides many retrieval options known from text retrieval, such as phrases, weighting terms, and expanding terms using ontologies, but leaves details of the scoring model used to rank results up to the implementation.

Pioneering work in the area of vague structural matches was done by [3,11] (and refined later by [4]), who proposed relaxing queries with structural constraints to find matches in structurally similar, but not exactly matching documents. The *FlexPath* [4] algorithm integrates structure and keyword queries and regards the query structure as templates for the context of a full-text keyword search. The query structure (as well as the content conditions) can be dynamically relaxed for ranked result output according to predefined tree editing operations when matched against the structure of the XML input documents.

Query Processing for Semi-Structured Text Retrieval

Efficient evaluation and ranking of conditions on content and structure of semi-structured data has been a very fruitful and popular research area in recent years. The majority of the proposed algorithms for efficient query evaluation combine some form of precomputed auxiliary indexes (like inverted files) with top-k aware processing algorithms, most notably Fagin's family of threshold algorithms (TA), which provide threshold-based candidate pruning and early termination.

XRank uses inverted lists containing – for each term – the elements that contain the term, sorted in descending order of element rank, along with a threshold algorithm for pruning the search space. The *FlexPath* query processor uses separate index structures for storing and retrieving the structure- and content-related conditions of a path query. The *Whirlpool* system introduced by Marian et al. [10] provides a flexible architecture for processing top-k queries on XML documents *adaptively*. Whirlpool allows partial matches to the same query to follow different execution plans, and takes advantage of the top-k query model to make dynamic choices during query processing. The key features of Whirlpool are: (i) a partial match that is highly likely to end up in the top-k set is processed in a prioritized manner, and (ii) a partial match unlikely to be in the top-k set follows the cheapest plan that enables its early pruning. Whirlpool provides several adaptivity policies and also supports parallel evaluations.

TopX [12], the actual successor of XXL, focuses on a small, XPath-like subset of the XXL query language which allows for a radically different query processing architecture that outperforms XXL in terms of efficiency by a large margin. As a native top-k engine for XML, TopX also uses sorted index lists, but keeps a candidate queue in-memory and therefore is able to focus on sequential disk access and on minimizing random disk access through sophisticated index structures and judiciously scheduled index access decisions.

A large effort has been made on mapping XML to relational schemas with highly specialized index structures for efficient support of approximate query processing, including support for IR-style retrieval functionality. *PF/Tijah* [8], which is now a part of *MonetDB/XQuery*, is an example for such an XQuery engine.

Support for XML-IR in Commercial Database Systems

Meanwhile, all commercially available databases with XML support, relational or native, provide some support for IR-style content search in combination to structural queries. As the full-text extensions of XPath and XQuery have not yet been finalized, systems typically come with their own extensions of their query language that are incompatible with – and sometimes less powerful than – the W3C proposals. Frequently, existing text search components are extended for XML support and provide the standard text search features (like phrase search, proximity conditions, stemming, etc.) for searching XML elements, usually with some scoring function to rank results.

Key Applications

The techniques presented before can be applied for efficiently retrieving information from large, possibly heterogeneous collections of semi-structured data. This includes more data-centric collections like bibliographies, collections of textual documents (like abstracts or full-text of publications or books), heterogeneous data exported from different sources, and eventually documents on the Web.

Cross-references

- ▶ XML Data Management:XML prototypes/systems
- ▶ XML Indexing
- ▶ Top-k XML Query Processing
- ▶ XQuery Full-Text

Recommended Reading

1. Abiteboul S., Quass D., McHugh J., Widom J., and Wiener J.L. The Lorel Query Language for Semistructured Data. Int. J. Digital Libraries, 1(1):68–88, 1997.
2. Amer-Yahia S., Botev C., and Shanmugasundaram J. TeXQuery: a full-text search extension to XQuery. In Proc. 12th Int. World Wide Web Conference, 2004, pp. 583–594.
3. Amer-Yahia S., Cho S., and Srivastava D. Tree Pattern Relaxation. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 496–513.
4. Amer-Yahia S., Lakshmanan L.V.S., and Pandit S. FleXPath: Flexible Structure and Full-Text Querying for XML. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 83–94.
5. Cohen S., Mamou J., Kanza Y., and Sagiv Y. XSEarch: A Semantic Search Engine for XML. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 45–56.
6. Fuhr N. and Großjohann K. XIRQL: A Query Language for Information Retrieval in XML Documents. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 172–180.
7. Guo L., Shao F., Botev C., and Shanmugasundaram J. XRANK: Ranked Keyword Search over XML Documents. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 16–27.
8. Hiemstra D., Rode H., Van Os R., and Flokstra J. PF/Tijah: text search in an XML database system. In Proc. 2nd International Workshop on Open Source Information Retrieval, 2006.
9. Hristidis V., Papakonstantinou Y., and Balmin A. Keyword Proximity Search on XML Graphs. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 367–378.
10. Marian A., Amer-Yahia S., Koudas N., and Srivastava D. Adaptive Processing of Top-*k* Queries in XML. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 162–173.
11. Schlieder T. and Meuss H. Querying and ranking XML documents. J. American. Soc. for Inf. Sci. & Tech. 53(6):489–503, 2002.
12. Theobald M., Bast H., Majumdar D., Schenkel R., and Weikum G. TopX: efficient and versatile top-*k* query processing for semistructured data. VLDB J., 17(1):81–115, 2008.
13. Theobald A. and Weikum G. Adding Relevance to XML. In Proc. 3rd Int. Workshop on the World Wide Web and Databases, 2000, pp. 105–124.
14. Xu Y. and Papakonstantinou Y. Efficient Keyword Search for Smallest LCAs in XML Databases, In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 537–538.

Integration of Rules and Ontologies

Jan Małuszyński
Linköping University, Linköping, Sweden

Definition

The layered structure of the Semantic Web (see http://www.w3.org/2007/03/layerCake.png) adopted by the World Wide Web Consortium W3C includes, among others, the Ontology layer with the web ontology language OWL and the rule layer with the emerging Rule Interchange Format (RIF) http://www.w3.org/TR/rif-fld/ which allows rules to be translated between rule languages. The integration of rules and ontologies aims at developing techniques for interoperability between rules and ontologies in the Semantic Web. This is necessary for rule-based applications to access existing domain ontologies. In most of the proposals the integration is achieved by defining and implementing a new language which is a common extension of a given rule language and a given ontology language, enhancing the expressive power of each of the components. Alternatively, the integration of rules and ontologies may be achieved by designing from scratch one language sufficiently expressive to define both rules and ontologies as well as their combinations.

Historical Background

In the initial phase of the Semantic Web research a significant effort was devoted to defining a language for ontology modeling. In 2004, it resulted in OWL, a family of three ontology languages OWL Lite, OWL DL, and OWL Full, based on Description Logics(DL). Each of them is a subset of the next one. The first two are syntactic variants of expressive description logics with semantics given by translation to formulae of first-order logic with equality. OWL DL (hence also its subsets) is supported by several reasoners. The original intention was to define OWL as a layer on top of RDF Schema which itself can be seen as ontology language. OWL Full, designed to achieve this goal, has a non-standard semantics and is difficult to implement. On the other hand, OWL DL includes a substantial subset of RDF Schema, which is extensively used in ontology definitions.

The importance of rules for web applications is reflected by the rule layer in the Semantic Web architecture. The rules formalisms considered for this layer offer modeling primitives not expressible in OWL. Their integration with OWL would thus enhance the expressive power of the latter.

In contrast to the ontology layer, no standard has been proposed yet for the rule layer. The rule languages proposed for the Semantic Web originate mainly from *logic programming* (see e.g., [18]). In contrast to OWL DL, they usually adopt the *closed world assumption*. This means that if a fact cannot be derived from a knowledge base it is concluded to be false. In logic programming, the closed world assumption is implemented by the *negation-as-failure* rule which returns $\neg p$ on failure to prove a fact p. This is an example of *non-monotonic reasoning*, not allowed in FOL.

Designing rules languages for the Semantic Web is among the objectives of the RuleML initiative (http://www.ruleml.org/). The W3C RIF Working Group is developing a core rule language as a basis for rule interchange (for more details see http://www.w3.org/2005/rules/wiki/RIF_Working_Group).

Foundations

Shortcomings of OWL

The following examples show why OWL is not sufficient for some applications and motivate its extensions. OWL Ontologies include classes (e.g., *Person*, *Woman*, *Man*) which are unary predicates and properties (e.g., *ParentOf*, *SisterOf*) which are binary predicates, but predicates of arity larger than two are not allowed. In OWL it is not possible to formalize the statement "an aunt of a person is a sister of a parent of that person." Also, the semantics of OWL does not allow to conclude that Mary is not a sister of John if the assertion *SisterOf*(*john*,*mary*) is not a logical consequence of the ontology. This kind of reasoning based on closed world assumption is useful in some applications. Most of the rule languages proposed for the Semantic Web do not share these shortcomings.

Rule Languages for Integration

The rule languages considered in integration proposals are usually extensions of Datalog. Generally rules have a form of "if" statements, where the predecessor, called the *body* of the rule, is a Boolean condition and the successor, called the *head*, specifies a conclusion to be drawn if the condition is satisfied.

In Datalog the condition of a rule is a conjunction of zero or more atomic formulae of the form $p(t_1,...,t_n)$ where p is an n-ary predicate symbol and $t_1,...,t_n$ are constant symbols or variables. Hence they are a restricted kind of FOL *terms* (see First order logic: syntax).

The head of a rule is an atomic formula (atom). For example, the rule

$$auntOf(X, Y) \leftarrow parentOf(Z, Y), sisterOf(X, Z)$$

states that X is an aunt of Y if Z is a parent of Y and X is this parent's sister. The semantics of Datalog associates with every set of rules (rulebase) its least Herbrand model (see e.g., [18]), where each ground (i.e., variable-free) atom is associated with a truth value *true* or *false*. The least Herbrand model is represented as the set of all atoms assigned to true. These are all the ground atoms which follow from the rules interpreted as implications in FOL. For example, the least Herbrand model of the rulebase consisting of the rule above and of the facts *parentOf*(*tom*, *john*), *sister Of*(*mary*, *tom*) includes the formula *auntOf*(*mary*, *john*). On the other hand, *auntOf*(*mary*, *tom*) does not follow in this rulebase. Hence the closed world assumption, used in Datalog, will result in the conclusion $\neg auntOf(mary, tom)$. Datalog rulebases constitute a subclass of logic programs. The latter use FOL terms, not necessarily restricted to constants and variables. Proposals for the integration of rules and ontologies are mostly based on the following extensions of Datalog (which apply also to logic programs):

- *Datalog with negation*, where the body may additionally include negation-as-failure (NAF) literals of the form *not a* where *a* is an atom. Intuitively a NAF literal *not a* is considered true if it does not follow from the program that *a* is true. For example, *happy*(*john*) can be concluded from the rulebase:

  $$happy(X) \leftarrow healthy(X), \text{ } not \text{ } hungry(X)$$
  $$healthy(john) \leftarrow$$

  Two commonly accepted formalizations of this intuition are: the *well-founded semantics* and the *stable model semantics* (see the survey [2]). The well-founded semantics [22] associates with a rulebase a unique (three-valued) Herbrand model, where each ground atom is assigned one of three logical values *true*, *false* or *unknown*. The stable model semantics [9] (called also the *answer set semantics*) associates with each rulebase some (possibly zero) two-valued Herbrand models. For a large class of programs relevant in practice (so called *stratified programs*, see e.g., [2]) both semantics coincide.
- *Extended Datalog*. This extension (see e.g., *extended logic programs* in [2]) makes it possible to state explicitly negative knowledge. This is achieved by allowing negative literals of the form $\neg p$, where $\neg$ is called the *strong negation* connective, in the heads of rules as well as in the bodies. For example, the rule

  $$\neg healthy(X) \leftarrow hasFever(X)$$

  allows to draw an explicit negative conclusion. In addition, NAF literals are also allowed in the bodies.
- *Rulebases with priorities*. Datalog rulebases employing strong negation may be inconsistent, i.e., may allow to draw contradictory conclusions. For example, the rules

  $$fly(X) \leftarrow bird\ (X)$$
  $$bird(Y) \leftarrow penguin(Y)$$
  $$\neg fly(X) \leftarrow penguin(X)$$
  $$penguin(tweety) \leftarrow$$

  allow to conclude *fly*(*tweety*) and $\neg$*fly*(*tweety*). In Defeasible Logic [19] and in Courteous Logic Programs [10] a priority relation on rules can be specified for a rulebase. The contradictions in the derived conclusions are then resolved by means of the defined priorities.
- *Disjunctive Datalog* [5] (see also *disjunctive logic programs* in [2]) admits disjunction of atoms in the rule heads, and conjunction of atoms and NAF literals in the bodies, e.g.,

  $$male(X) \vee female(X) \leftarrow person(X).$$

  A commonly used semantics of Disjunctive Datalog rulebases is an extension of the answer set semantics.

The rule languages are supported by implementations which make it possible to query and/or to construct the models of rulebases.

Approaches to Integration

The integration of a given rule language with a given ontology language is usually achieved by defining a common extension of both, to be called the integrated language. Alternatively, one can adopt an existing knowledge representation language expressive enough to represent rules and ontologies. As OWL is a standard ontology language the ontology languages considered in integration proposals are usually its subsets. The approaches can be classified by the degree of the integration of rules and ontologies achieved in the integrated language.

- *Homogeneous Integration*. The integrated language makes no distinction between the rule predicates and the ontology predicates. It includes the original rule language and the original ontology language as sublanguages. The integration is to be *faithful* in the sense that the sublanguages should have the same semantics as the respective original languages. The homogeneous integration is difficult to achieve since ontology languages are usually based on FOL and rule languages have different kind of semantics. Examples of the homogeneous integration include:
  - *DLP (Description Logic Programs)* [12], which is a language obtained by intersection of a Description Logic with Datalog rules interpreted as FOL implications. DLP has a limited

  expressive power, but a DLP ontology can be compiled into rules and easily integrated into a rulebase of a more expressive rule language. For example *Sweet Rules* http://sweetrules.projects.semwebcentral.org/ combine DLP and Datalog with strong negation and priorities. The technique of compiling ontologies to rules is also used in DR-Prolog [1] based on Defeasible Logic.
  - *F-logic*, extending classical predicate calculus with the concepts of objects, classes, and types. It is expressive enough to represent ontologies, rules and their combinations [13].
  - *SWRL (Semantic Web Rule Language)* http://www.w3.org/Submission/SWRL/, extending OWL DL with rules interpreted as FOL implications. Thus, SWRL is based on FOL and does not offer non-monotonic features, such as negation-as-failure. A so-called DL-safe subset [17] of SWRL is supported by KAON2 system http://kaon2.semanticweb.org which also offers a support for a restricted subset of F-logic.
  - *Hybrid MKNF Knowledge Bases* [16], taking a modal logic as a basis of faithful integration of Description Logic with Disjunctive Datalog under the answer set semantics. A variant of this approach considering nondisjunctive hybrid MKNF knowledge bases under well-founded semantics is presented in [14].
  - *Quantified Equilibrium Logic*, considered in [3] as a unified framework which embraces classical logic as well as disjunctive logic programs, thus providing a foundation for the integration of rules and ontologies.
- *Heterogeneous Integration.* In this approach, the distinction between the rule predicates and the ontology predicates is preserved in the integrated language. The integration of rules and ontologies is achieved by allowing the ontology predicates in the rules of the integrated language. Assume, for example, that an ontology classifies courses as project courses and lecture courses.

$$\mathit{Project} \sqcup \mathit{Lecture} = \mathit{Course}$$

It also includes assertions like $\mathit{Lecture}(cs05)$, $\mathit{Project}(cs21)$ or $\mathit{Course}(cs32)$ (e.g., for courses including lectures and projects). The assertions indicate offered courses. A person is considered a student if he/she is enrolled in an offered lecture or project. This can be expressed by the following rules, using the ontology predicates

$$\begin{aligned} &\mathit{student}(X) \leftarrow \mathit{enrolled}(X,Y),\ \mathit{Lecture}(Y) \\ &\mathit{student}(X) \leftarrow \mathit{enrolled}(X,Y),\ \mathit{Project}(Y) \end{aligned}$$

In addition the rulebase includes enrollment facts, e.g., $\mathit{enrolled}(\mathit{joe}, cs32)$. The extended language allows thus to define ontologies using the constructs of the ontology language and the rulebases with rules referring to the ontologies. An extended rulebase together with an ontology is called a *hybrid knowledge base*. In the heterogeneous approaches implementations are often based on *hybrid reasoning principle*, where a reasoner of the ontology language is interfaced with a reasoner of the rule language to reason in the integrated language.

Two kinds of heterogeneous approaches can be distinguished:

1. *Loose coupling.* In this approach the semantics of hybrid knowledge bases is based on a transformation which eliminates ontology queries from ground extended rules by querying the underlying ontology. A ground set of extended rules is thus reduced to a set of rules without ontology predicates in the following way. If the answer to a ground ontology query in a rule body is positive, the query is removed from the rule, otherwise the rule is removed from the set. The loose coupling approach applied to the example above does not allow to conclude that Joe is a student. This is because neither $\mathit{Lecture}(cs32)$ nor $\mathit{Project}(cs32)$ can be derived from the ontology. Examples of loose coupling include:
   - *dl-programs* [6], combining (disjunctive) Datalog with negation under the answer set semantics with OWL DL. So called DL-queries, querying the ontology, are allowed in rule bodies. They may also refer to a variant of the ontology, where the set of its assertions is modified by the DL-query. This enables bi-directional flow of information between rules and ontologies. A variant of the language based on the well-founded semantics is presented in [7].
   - *TRIPLE* [21], a rule language with the syntax inspired by F-logic. It admits queries to the ontology in rule bodies.
   - *SWI Prolog* http://www.swi-prolog.org/, a logic programming system with a Semantic Web library

which makes it possible to invoke RDF Schema and OWL reasoners from Prolog programs.

2. *Tight integration.* In this approach the semantics of hybrid knowledge bases is defined by combining the model-theoretic semantics of the original rule language with the FOL semantics of the ontology language. For example, tight integration of Datalog (without negation) with a Description Logic can be achieved within FOL by interpreting Datalog rules as FOL implications. In this semantics $student(joe)$ is a logical consequence of the example hybrid knowledge base. As $Course(cs32)$ is an assertion of the ontology, it follows by the axiom $Project \sqcup Lecture = Course$ that in any FOL model of the ontology $Project(cs32)$ or $Lecture(cs32)$ is true. As $enrolled(joe, cs32)$ is true in every model, so the premises of at least one of the implications

$$student(joe) \leftarrow enrolled(joe, cs32), Lecture(cs32)$$
$$student(joe) \leftarrow enrolled(joe, cs32), Project(cs32)$$

   must be true in any model. Hence $student(joe)$ is concluded. Examples of tight integration include:
   - *CARIN* [15], a classical work on integrating Datalog with a family of Description Logics under the FOL semantics.
   - $\mathcal{DL}+log$ [20], integrating Disjunctive Datalog under the answer set semantics with OWL DL. For each FOL model of the ontology the rules of the knowledge base are reduced to rules of Disjunctive Datalog, with stable models defined by the answer set semantics.
   - *Hybrid Rules* [4], integrating logic programs under well-founded semantics with OWL. For each FOL model of the ontology the rules of the knowledge base are reduced to a logic program with the model defined by the well-founded semantics.

The theoretical foundations developed by studying integration of ontologies with variants of Datalog provide a basis for further extensions. This includes dealing with uncertain and inconsistent knowledge, and using integrated Datalog-based languages as condition languages for ECA-Rules.

Key Applications

The integration of rules and ontologies is a relatively new research topic, focused so far on developing tools and prototypes. Key applications include semantic data integration, ontology-based web search and semantic recommendation systems. Industrial applications of this kind are discussed in the video lecture [8]. The Ontobroker system referred therein is based on F-logic. Another field of potential key applications is e-business as discussed in the tutorial video [11], with focus on Sweet Rules.

URL to Code

The following systems integrating rules and ontologies can be downloaded:

- *KAON2* from http://kaon2.semanticweb.org,
- *Sweet Rules* from http://sweetrules.projects.semwebcentral.org/,
- *SWI Prolog* from http://www.swi-prolog.org/,
- *TRIPLE* from http://triple.semanticweb.org/.

A prototype implementation of *dl-programs* (NLP-DL) can be accessed at http://con.fusion.at/nlpdl/.

Cross-references

▶ Datalog
▶ Description Logics
▶ ECA Rules
▶ First-Order Logic: Syntax
▶ First-Order Logic: Semantics
▶ Ontology
▶ OWL: Web Ontology Language
▶ Resource Description Framework (RDF) Schema (RDFS)
▶ Semantic Web
▶ W3C

Recommended Reading

1. Antoniou G. and Bikakis A. DR-Prolog: a system for defeasible reasoning with rules and ontologies on the semantic Web. IEEE Trans. Knowl. Data Eng., 19(2):233–245, 2007.
2. Baral C. and Gelfond M. Logic programming and knowledge representation. J. Logic Program., 19/20:73–148, 1994.
3. de Bruijn J., Pearce D., Polleres A., and Valverde A. Quantified equilibrium logic and hybrid rules. In Proc. 1st Int. Conf. on Web Reasoning and Rule Systems, 2007, pp. 58–72.
4. Drabent W. and Małuszyński J. Well-founded semantics for hybrid rules. In Proc. 1st Int. Conf. on Web Reasoning and Rule Systems, 2007, pp. 1–15.
5. Eiter T., Gottlob G., and Mannila H. Disjunctive datalog. ACM Trans. Database Syst., 22(3):364–418, 1997.
6. Eiter T., Lukasiewicz T., Schindlauer R., and Tompits H. Combining answer set programming with description logics for the

semantic web. In Proc. 9th Int. Conf. Principles of Knowledge Representation and Reasoning, 2004, pp. 141–151.

7. Eiter T., Lukasiewicz T., Schindlauer R., and Tompits H. Well-founded semantics for description logic programs in the semantic web. In Proc. 3rd Int. Workshop on Rules and Rule Markup Languages for the Semantic Web, 2004, pp. 81–97.
8. Erdmann M. Semantic web applications. 2007, first Asian Autumn School on Semantic Web, Tutorial video at: http://rease.semanticweb.org/ubp/.
9. Gelfond M. and Lifschitz V. The stable model semantics for logic programming. In Proc. 5th Int. Conf. Logic Programming, 1988, pp. 1070–1080.
10. Grosof B.N. Prioritized conflict handling for logic programs. In Proc. 14th Int. Conf. Logic Programming, 1997, pp. 197–211.
11. Grosof B. Semantic web rules with ontologies, and their e-Services applications. 2006, iSWC06 tutorial video at: http://videolectures.net/iswc06_grosof_swrot/.
12. Grosof B., Horrocks I., Volz R., and Decker S. Description logic programs: combining logic programs with description logic. In Proc. 12th Int. World Wide Web Conference, 2003, pp. 48–57.
13. Kifer M. Rules and ontologies in F-logic. In Reasoning Web, N. Eisinger and J. Małuszyński (eds.). LCNS, vol. 3564, 2005, pp. 22–34.
14. Knorr M., Alferes J.J., and Hitzler P. A well-founded semantics for hybrid MKNF knowledge bases. In Proc. 20th Int. Workshop on Description Logics, 2007, pp. 347–354.
15. Levy A. and Rousset M.C. CARIN: a representation language combining horn rules and description logics. Artif. Intell., 104 (1–2):165–209, 1998.
16. Motik B. and Rosati R. A faithful integration of description logics with logic programming. In Proc. 20th Int. Joint Conf. on AI, 2007, pp. 477–482.
17. Motik B., Sattler U., and Studer R. Query answering for OWL-DL with rules. J. Web Sem., 3(1):41–60, 2005.
18. Nilsson U. and Małuszyński J. Logic, Programming and Prolog, 2nd edn. Wiley, NY, 1995, now available free of charge at: http://www.ida.liu.se/ ulfni/lpp/.
19. Nute D. Defeasible logic. In Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 3. Oxford University Press, Oxford, 1994, pp. 353–395.
20. Rosati R. *DL*+log: tight integration of description logics and disjunctive datalog. In Proc. 10th Int. Conf. Principles of Knowledge Representation and Reasoning, 2006, pp. 68–78.
21. Sintek M. and Decker S. TRIPLE – a query, inference, and transformation language for the semantic web. In 2002, pp. 364–378
22. van Gelder A., Ross K.A., and Schlipf J.S. Unfounded sets and well-founded semantics for general logic programs. In Proc. 7th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1988, pp. 221–230.

Intellectual Property

► Copyright Issues in Databases
► European Law in Databases

Intelligent Disks

► Active Storage

Intelligent Storage

► Intelligent Storage Systems

Intelligent Storage Systems

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Synonyms

Intelligent Storage

Definition

The term Intelligent Storage System is a general term used to describe a storage system which has the capability of fully or partially realizing functions that used to be or are usually implemented on host computers.

Historical Background

The idea behind Intelligent Storage Systems may have its origin in the early researches on database machines. Similar ideas have continued to be studied in the academic communities to date, and they have recently been partially applied in commercial storage systems.

Foundations

The basic ideas of implementing full or partial application code on controller processors of disk drives may be traced back to the database machines which were actively studied in the 1970s and 1980s. The database machine was an approach of special hardware solutions. The early researchers focused on the development of filter processors, which could do selection operations closely to disk drives so as to obtain strong performance benefits. Several prototypes such as CASSM (University of Florida) [11] and RAP (Ohio University) [7] were implemented in the 1970s. Filter processors were at times coupled with a front-end database server. Such ideas called backend processors [3] were attempted in the industry. When it came to the 1980s, with new algorithms such as hash-based joins, researchers proposed new database machines that pursued intensive parallel processing such as GRACE (The University of Tokyo) [6] and GAMMA (University of Wisconsin) [4]. Parallel machines that were specially designed for database processing were released and then resulted in commercial success. However, the solution of utilizing dedicated hardware lost its unique advantage by the 1990s, since powerful general-purpose machines became easily available. Major database vendors shifted to software-level solutions which used generic hardware instead.

Intelligent Storage Systems were again brought under the spotlight in the late 1990s. Storage network technologies such as Fibre Channel launched in the market. Then enterprise systems began to deploy the storage-centric system architecture, where storage systems could be designed and managed independently from host computers at the infrastructure level. Naturally, sophisticated new functions such as virtualization were being incorporated into storage processors and such solutions were widely accepted. Around the same time, Active Storage [8], Intelligent Disks [5] and Active Disks [1] were published in the academia in 1998. These were trying to exploit the capability of disk processors for data intensive applications such as ad-hoc query processing and image processing. The attempt of active storage looked similar to the database machine, but they carefully discussed software frameworks for running application code on disk processors.

After the twenty-first century began, storage networking has been practiced in many systems and storage resources are being consolidated more. A variety of new functions are being implemented in commercial storage systems. Functions that used to be run on host computers such as volume copy, remote replication and snapshot generation are usually executed in storage processors. Although only limited types of low-level applications are currently implemented in storage systems, the application domain is gradually being widened by the active research and development.

The motivation behind database machines and active storage was mainly in significant performance improvement. By processing data more closely to disk platters, they tried to efficiently exploit the limited bandwidth between main memory and storage systems. This "storage wall" is seen even in recent enterprise systems and thus such solutions are still beneficial. At the same time, in the light of the complexity of recent enterprise systems, Intelligent Storage Systems may have another substantial benefit. That is, storage-level implementation could improve the function-level isolation between components. This would be very helpful for system administrators to design and manage the complicated system.

Discussion on interface standards is a crucial point for realizing Intelligent Storage Systems. OSD (Object Storage Device) [2] has evolved out of the NASD (Network-Attached Secure Disk) project which started in Carnegie Mellon University in 1995. In contrast to traditional storage devices, where the storage space is represented as an array of fixed-size blocks, OSD works as a container of objects, their attributes and their metadata. Specifically, OSD can be seen as a storage device in which lower layers of file systems are implemented. The interface protocol of OSD, designed as an extension of SCSI, has been standardized as ANSI T10 SCSI OSD. Several NAS products and distributed files systems have already supported OSDs as backend storage devices. SNIA, a leading industry association of storage networks, has also promoted standardization. SMI-S (Storage Management Initiative-Specification) [9] is the standard protocol for storage management, which improves the interoperability between different storage devices, switches and management applications of different manufacturers. SNIA has developed and maintained SMI-S and has provided vendors with certification programs. XAM (eXtensible Access Method) [10] is another task operated by SNIA. XAM, a new suite of APIs, would provide an abstraction layer between storage devices which store fixed contents and management applications which access those contents.

Key Applications

Recent commercial storage systems have deployed storage-side implementation of simple functions such as data conversion between main frames and open systems, third-party copy, remote replication and snapshot generation. These were so far implemented only in top-end storage systems, but they are also being implemented in mid-range products and sometimes even in entry-level products.

Cross-references

- ▶ Active Storage
- ▶ Database Machine
- ▶ Network-Attached Secure Device

▶ Storage Network Architecture
▶ Storage Management

Recommended Reading

1. Acharya A., Uysal M., and Saltz J.H. Active disks: programming model, algorithms and evaluation. In Proc. 8th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 1998, pp. 81–91.
2. ANSI. Information Technology - SCSI Object-Based Storage Device Commands (OSD). Standard ANSI/INCITS 400–2004. 2004.
3. Canaday R.H., Harrison R.D., Ivie E.L., Ryder J.L., and Wehr L.A. A back-end computer for data base management. Commun. ACM, 17(10):575–582, 1974.
4. DeWitt D.J., Gerber R.H., Graefe G., Heytens M.L., Kumar K.B., and Muralikrishna M. GAMMA – a high performance dataflow database machine. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 228–237.
5. Keeton K., Patterson D.A., and Hellerstein J.M. A case for intelligent disks (IDISKs). ACM SIGMOD Rec., 27(3):42–52, 1998.
6. Kitsuregawa M., Tanaka Hi., Moto-Oka T. Application of hash to data base machine and its architecture. New Generation Comput, 1(1):63–74, 1983.
7. Ozkarahan E.A., Schuster S.A., and Smith K.C. RAP - An associative processor for database management. In Proc. National Computer Conf., 1975, pp. 379–387.
8. Riedel E., Gibson G.A., and Faloutsos C. Active storage for large-scale data mining and multimedia. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 62–73.
9. SNIA Storage Management Initiative. Storage Management Technical Specification, Overview Version 1.2.0, Revision 6. 2007.
10. SNIA XAM Initiative. XAM Initiative Overview, 2007.
11. Su S.Y.W. and Lipovski G.J. CASSM: a cellular system for very large data bases. In Proc. 1st Int. Conf. on Very Data Bases, 1975, pp. 456–472.

Interaction Design

▶ Human-Computer Interaction

Interactive Capture

▶ Visual Interfaces for Geographic Data

Interactive Information Exploration

▶ Information Navigation

Interactive Layout

▶ Visual Interfaces for Geographic Data

Interactive Visual Exploration of Multidimensional Data

▶ Visual On-line Analytical Processing (OLAP)

Interface

Patrick Eugster
Purdue University, West Lafayette, IN, USA

Definition

An interface describes the functionalities exported by an entity such as a software module. These functionalities typically consist in named operations with signatures describing potential arguments and return values, but interfaces may also include the definitions of data types, constants, exceptions, or even describe semantics.

Interfaces shield the internals of corresponding software modules from the outside, providing several benefits. Interfaces provide abstraction, in the sense that the internals of modules may evolve while other modules can still rely on the same functionalities (encapsulation). Safety and security are promoted by interfaces as these define single access points to respective software modules.

In object-oriented programming, software modules often coincide with classes, which also describe data types implicitly. In the case of such a class, an interface is thus roughly made up of the methods exported by the class. Mostly, interfaces are then defined implicitly by the classes as such sets of exported methods, but languages such as Java provide programmers the possibility of declaring interfaces as first class citizens which are then explicitly implemented by classes.

When objects are physically distributed, they can offer specific remote interfaces facilitating invocations potentially coming from remote hosts. Remote interfaces span typically only a subset of all the methods exported by objects.

A set of interfaces which describe an entire software component or framework is more commonly referred to as an application programming interface (API).

Historical Background

The concept of interface has partly evolved out of the *header files* used with the C programming language. In contrast to several more recent programming languages providing notions of interfaces integrated with the language semantics, those header files served mainly as preprocessing feature.

Interfaces were greatly popularized by programming languages based on *modules*, such as Pascal, Modula-2, or Ada. A first step towards a more rigorous underpinning of modules and measures for a good decomposition of software into such modules was provided by Parnas [8]. Parnas' work primarily aimed at achieving a clean separation of duties in software components and an effective decomposition into modules. As a dual of this problem, the design process for interfaces to modules has also been strongly guided by Parnas' seminal work, coining the term "well-defined interfaces." The main driving force behind the quest for clear-cut interfaces are safety/security concerns aiming at providing well-defined entry points to given modules in the form of interfaces.

The term *interface* itself thus emerged from the software engineering community. Ever since, programming languages have been strongly influenced by this notion, in particular because there is a strong overlapping between the notions of interface and *type* (*data type*). This culminates in object-oriented programming, where data types and modules are largely unified through classes representing the units for both data abstraction and behavior, and thus a type defines the interface to its instances, even if only implicitly (cf. [2]).

More recently, modeling languages such as the Unified Modeling Language (UML) have also introduced interfaces aside classes as purely descriptive means of capturing functionalities of objects.

Foundations

Interfaces describe the functionalities exported by an entity such as a software module. These functionalities typically consist in named operations with signatures describing potential arguments and return values, but interfaces may also include the definitions of data types, constants, exceptions, or even more information.

Interfaces and Classes

Most object-oriented programming languages are class-based, meaning that classes present the main unit of decomposition and as such can be viewed as modules. Interfaces of such classes encompass the functionalities exported by these classes – typically a set of methods which are accessible to other classes. In many integrated development environments (IDEs), there is the possibility to automatically "extract" such an interface view from a class, by only summarizing the names and signatures of its exported features.

Interfaces can be in that sense seen as abstract types, as opposed to classes which may *implicitly* describe interfaces, but primarily describe the implementation of those. Current programming languages such as Java provide interfaces as a first class construct aside classes. Declaring an `interface` leads to defining an abstract type. In this case, an object can provide several interfaces if its class implements several instances; the *conceptual* interface of the instance(s) is then made up of the union of these interfaces.

When collapsed with types, interfaces reap the benefits of the Liskov substitution principle [6], which states rules under which a type `T1` can be considered to be compliant with another type `T2`. Instances of `T1` can then be used whenever entities of type `T2` are expected. Transposed to interfaces this means in short that the interface of the latter type subsumes the interface of the former type: any module that builds on the former interface can be presented with an instance of the latter interface.

Protection

Conceptual interfaces are sometimes decomposed according to potential beholders of references to the corresponding entities. More precisely, different objects, or more generally different modules, can have access to the same entities through individual interfaces. By supporting different visibility rules (e.g., `private`, `public`, `protected`), possibly at the method level, programming languages provide the ability to restrict interfaces (or parts of such interfaces) to families of classes for example based on containment criteria (e.g., packages). In C++ for instance, a class `C1` can be explicitly declared to be a `friend` of another class `C2`, providing instances of `C1` access to certain methods of class `C2` which are otherwise unreachable. Introducing varied levels of protection yields different (accessible) interfaces for different parties.

For languages with weaker inherent support for interfaces and protection levels, design patterns can be used to enforce constraints. The *read-only interface* pattern for instance is used to explicitly define an interface through which entities can be manipulated without modifications to their state.

Remote Interfaces

In remote method invocations, *remote interfaces* are used to explicitly offer methods for invocation from remote hosts. Remote interfaces usually span only a subset of the methods provided by a class. The remaining methods however are limited to invocations within the same address space. In that sense, remote interfaces introduce a specific level of protection.

Remote interfaces commonly enjoy descriptions in a dedicated specification language, when interoperability is desired. Such specification languages allow the description of remote interfaces in an interoperable format, and subsequent generation of language-specific versions with corresponding compilers based on well-defined mappings to target languages. Examples are the *interface definition language* (IDL) used in the Common Object Request Broker Architecture (CORBA), or more recently the *web service description language* (WSDL). An earlier incarnation is the *external data representation* (XDR) introduced with the remote procedure call (RPC) protocol. Compilation of such interfaces commonly goes hand in hand with the generation of proxes or stubs which take care of transforming method invocation arguments and return values to and from an interoperable format such as XDR.

Contents

Besides methods, interfaces can also include constants, exceptions, and sometimes even fields, though exporting fields is sometimes viewed as going against the principle of encapsulation underlying object-oriented programming. In the Eiffel programming language, methods are unified with field (accesses), implicitly yielding access methods for fields. Eiffel is also the most popular language with further *semantic* descriptions in the interfaces, consisting in *contracts* in the form of pre- and postconditions for methods, as well as invariants at the class level. These were introduced as part of the original definition of *abstract data type* [6], and have given rise to a design paradigm called *design by contract* (DbC) [7].

In the case of Eiffel, its IDE also allows the programmer to view the interface of a given class, in this case including contracts. Contracts have a descriptive flavor in the sense that they augment interfaces and thus represent a specification feature, but also have an implementation flavor since they may refer to fields, and can be monitored at runtime as in Eiffel. Spec# follows the Eiffel approach by providing first class support, while the Java Modeling Language (JML) promotes annotations, which can be viewed as optional semantic interface descriptions, to which programmers only must adhere when making use of specific tools (e.g., for compilation).

Besides contracts, other types of information have been considered for augmenting interfaces. *Typestates* [9] capture abstract states associated with instances of a given type and their relationship with exported methods, thus adding state information to interfaces. *Interface automata* [4] focus directly on the *order* in which functionalities exposed by a given interface can be triggered. Such sequences are sometimes also colloquially referred to as *protocols*. *Resource interfaces* [3] are another example of augmented interfaces, describing namely the physical resources necessary for a given module to be able to function properly.

Components

In software engineering, in particular in the *component* view, interfaces may similarly contain far more information than types. A component usually refers to a subsystem of its own consisting in several modules, whose individual interfaces make up the conceptual interface. The interface of a component can for instance also define what kind of interface the component itself relies upon (while this is usually somewhat embedded in object-oriented code).

When several interfaces are bundled either in the context of a component, or a framework, the term *application programming interface* (API) is commonly employed. The different levels of technical protection for functionalities offered at the programming language level are in the case of APIs often complemented by simply limiting the disclosure of APIs to selected corporate players only.

Key Applications

There are various reasons for further studying the domain of interfaces:

- Complexity. With the increasing complexity of computer systems, the problem of "efficiently" decomposing/composing software is given more and more attention. Abstraction and encapsulation are key desired properties. In software engineering terms, one attempts to achieve decoupling and increase modularity (For a precise definition of modularity see [8]). The art of decomposing software is also partly captured by the study of *software architectures*. There is a clear dependency between the decomposition of software into modules and the discipline of designing interfaces between modules/components to respect modularity and achieve low coupling.
- Reuse and maintainability. These are also desired features of any software package. With good design and modularity reflected in rich interfaces, code namely also becomes easier to reuse and maintain.
- Safety. Safety has become a major driving force for programming language and software engineering research. Ensuring that interaction between (sub-)modules takes place in a safe manner helps avoiding runtime errors, and is supported by clear-cut and precise interfaces. In the predominant settings with static typing, the goal is to ensure that once successfully compiled, a module contains no calls violating interfaces.
- Security. Access rights and confinement can be expressed in interfaces as mentioned above. This becomes particularly visible in distributed settings, i.e., for achieving clean interfaces between entities running on distinct physical hosts.
- Interoperability. Not all software is written in the same programming language. With interfaces being described in "neutral" generic languages interoperability can be achieved. Examples are clearly provided by *interface definition languages* (IDLs) for second class distributed object packages such as DCE, CORBA, or DCOM. The idea of interoperability has more recently flown into **service-oriented architectures** (SOA) in general and **Web Services** in particular.

Future Directions

Motivated by the above, there is quite some incentive to further investigate interfaces. Various aspects offer themselves for additional efforts:

- *Evolution*. Low coupling and modularity definitely support evolution of components, but as mentioned above, only in terms of the internals of components. Evolution of components sometimes also necessitates evolution of their interfaces, which can however break any existing code relying on these. Evolution of interfaces is thus an important area to investigate. In programming language terms, this leads to the problem of extending and changing types, which can be partly addressed by introducing versioning, and by assisting the programmer in the adaptation of code after changes to interfaces it relies upon.
- *Information*. Interfaces describe contracts between modules, i.e., functionalities provided by a module and sometimes also requirements for providing these functionalities. So far, interfaces are mostly described through functional aspects of the modules which in programming language terms boils down to syntactic and typing information. Design by contract (DbC) augments these interfaces with semantics, but further information might be of relevance, which is not even directly materialized in the implementation of modules, as illustrated by resource interfaces [3].
- *Discovery*. Interfaces and extended interfaces can also be obtained by *mining* existing modules, e.g., [1]. This is particularly appealing when dealing with legacy code, which might have been described in a programming language with weak inherent support for interface descriptions. Discovery of interfaces has also gained more interest recently in the realm of *aspect-oriented programming* (AOP), whenever aspects advise code *obliviously*, i.e., without knowledge or consent from the main code. In the AOP philosophy aspects attempt to achieve a separation of duty by implementing *crosscutting concerns* in aspects alike modules, separately from the base code. Without clean interfaces for the interaction between base code and aspects however, reasoning about an entire software becomes hard (cf. [5]).

  Discovery can also be understood in the sense of discovering services at runtime. Especially SOAs provide strong support for finding services based on their functionalities, often described as some abstract interfaces. UDDI is an example of a lookup service that supports queries based on a specific notion of interface.

Cross-references

- ▶ Discovery
- ▶ Request Broker
- ▶ Service Oriented Architecture
- ▶ Web Services

Recommended Reading

1. Beyer D., Henzinger T.A., and Singh V. Algorithms for interface synthesis. In Proc. 19th Int. Conf. on Computer Aided Verification, 2007, pp. 4–19.
2. Canning P.S., Cook W.R., Hill W.L., and Olthoff W.G. Interfaces for strongly-typed object-oriented programming. ACM SIGPLAN Not., 24(10):457–467, 1989.
3. Chakrabarti A., de Alfaro L., Henzinger T.A., and Stoelinga M. Resource interfaces. In Proc. 3rd Int. Conf. on Embedded Software, 2003, pp. 117–133.
4. de Alfaro L. and Henzinger T.A. Interface automata. In Proc. 9th ACM SIGSOFT Int. Symp. on Foundations of Software Eng., 2001, pp. 109–120.
5. Griswold W.G., Sullivan K.J., Song W., Shonle M., Tewari N., Cai Y., and Rajan H. Modular software design with crosscutting interfaces. IEEE Softw., 23(1):51–60, 2006.
6. Liskov B.H. and Wing J.M. A behavioral notion of subtyping. ACM Trans. Program. Lang. Syst., 16(6):1811–1841, November 1994.
7. Meyer B. Applying design by contract. IEEE Comput., 25 (10):40–51, October 1992.
8. Parnas D.L. On the criteria to be used in decomposing systems into modules. Commun. ACM, 15(12):1053–1058, 1972.
9. Strom R.E. and Yemini S. Typestate: a programming language concept for enhancing software reliability. IEEE Trans. Softw. Eng., 12(1):157–171, 1986.

Interface Engines in Healthcare

DAN RUSSLER
Oracle Corporation, Redwood Shores, CA, USA

Synonyms

Transformation engines; Mapping engines; Messaging engines; Service buses

Definition

A computer application that supports the transformation of the syntactic and semantic structures in communication content during transmission from a sending system to receiving system(s), ensuring reliable delivery of the communication and minimizing information loss and semantic shift during the communication.

Historical Background

Before the invention of application programming interfaces (API) and CORBA Interface Definition Language (IDL) files, "interface" was the term used to describe the electronic communication of information between two computers [1]. Today, the term "interface" also refers to communications between layers of software and even between software objects within a software layer. A modern example of an interface is a Web Service Definition Language (WSDL) file in XML format (www.w3.org).

Within early interfaces, the syntax of the communication content, i.e., linear arrangement of characters in the communication exported by the sending system, often could NOT be imported directly by the receiving system. Consequently, transformation procedures were employed to rearrange the characters in the communication into a linear structure that COULD be imported by the receiving system. In the same manner, if terms used by the sending systems could not be imported into the receiving system, a substitution of terms that could be imported by the receiving system was also applied to the communication content.

In order to reduce the burden of computing these transformations on the slower computers of the past, the execution of these transformation procedures was transferred to a separate computer that was placed in-between the sending and receiving computers. As time passed, the ability of these "interface systems" to support better and easier transformation authoring and high "transaction" or messaging volumes, these systems became known in the 1990s as "interface engines," a new kind of software application in themselves [4,5].

As the language of the industry evolved, the communications between computers became known as "messages"; the transformation of the syntactic and semantic content became known as "mapping" from sender to receiver; and techniques for ensuring reliable receipt of communications between a sending and receiving computer became known as "reliable delivery protocols" or "reliable messaging protocols."

However, despite the increasing sophistication of the transformation tools, the industry soon discovered that there was a limited ability to ensure that all the information sent by the sending system could be imported by the receiving system [7]. Information loss and shift in meaning of the communication content was observed when evaluating the information content of the sending and receiving systems after the communication occurred. In many ways, the result was similar to the garbling of sentences that occurs in the children's games that test the ability of children sitting in a ring to sequentially whisper a sentence into the ear of the next child. The child who initiates the sentence rarely gets the same sentence whispered back by the last child.

As a consequence, organizations that developed standards in support of many other industries began to support standardized messaging structures for the computer industry, including message standards in the healthcare industry. These standards included both the arrangement of "fields" and special characters in the messages and the sets of terms used to populate these fields. In healthcare, the messages most widely used internationally by the year 2000 were authored by a specialized healthcare standards organization, the Health Level 7 (HL7) standards development organization (www.hl7.org), which focused on ISO Level 7 transaction protocols [1].

These messaging standards reduced the cost of each messaging "interface" between computers by as much as tenfold from the 1980s to the turn of the century. However, the implementation cost of these messaging interfaces (HL7 version 2.x messages) continued to retail at over $20,000 per interface (in addition to the cost of the interface engine), and many hospitals required over one hundred messaging interfaces.

By the early 1990s, planners in HL7 began exploring model-based development methods and new message authoring techniques that would both improve the quality and reduce the cost of communications between computer systems in healthcare. The improvement in quality of communication refers to the preservation of information content and semantic meaning of the communication or what is known as "semantic interoperability." The reduction in cost comes from decreasing the number of choices allowed to developers who "interpret" the standards into actual application code. The result of this planning effort was the publication of the HL7 Reference Information Model (RIM) [6] and RIM-derived messages, electronic documents [2], and web services.

Parallel efforts in other industries as well as the growth of Internet communications between computer systems have caused rapid evolution in both the kinds of communications the healthcare industry wishes to utilize and the techniques for electronically communicating between computer systems.

As the result of many initiatives across industries, there have been great strides in communication methods that no longer utilize the traditional, preconfigured point-to-point HL7 interface engines. The concept of a "bus" was borrowed from the internal computer bus that supports the physical "plug & play" communication of multiple physical components (such as hard drives) within a computer. This concept of a "bus" was abstracted to a "service bus" located within a data center that allows dynamic selection of multiple web services in a service-oriented architecture. As a result, interface communication in a data center no longer needs to rely on a "point-to-point" pre-configured solution. Rather, in a service bus, a "service directory" may be used to dynamically select the method of communication and the endpoint(s) of communication desired. Increasingly, these new kinds of "messaging engines" or "enterprise service buses" are being utilized in healthcare data centers. Finally, "healthcare service bus" is a term used to describe a "virtual service bus" or a system of "federated enterprise service buses" where dynamic web services are supported between individual enterprises and enable healthcare communications across the wider community (www.openhealthtools.org).

Although many people believe that legacy computer systems and traditional, preconfigured point-to-point messaging methods will continue to be used for many years in healthcare, the consequence of new communication techniques used by many other industries is that communication methods in healthcare will also evolve. The role of the traditional data-center-based, HL7 2.x interface engine will gradually be reduced in favor of web-service-based communications that support transformation and routing across healthcare communities.

Foundations

The scientific study of messaging concepts begins with narrow, reductionist models, e.g., the physics of electrons and gravitational bonds communicating between atoms and within molecules. These concepts are studied within incrementally more complex systems in chemistry labs, organic chemistry labs, genetic and hormonal communications, electronic systems and neural communications, human communication, and finally, in computer systems that support human communication.

The study of messaging was enhanced by the development of communication models, many of which include the concept of state machines. Communication may be defined as "transferring awareness of a change of state in the model of the sending system to the receiving system(s)." This communication may be as simple as informing the receiver of the increase in the state of ionic attraction that occurs when an atom has gained or lost an electron or as complex as informing a second organization that the state of a hospital now includes a patient ready for placement in a nursing home.

The concept of a "state machine" was introduced in the mathematical modeling and electronics literature. A "state machine" was a systems model developed in the 1950s and 1960s to describe state transitions in sequential circuits [3]. Generally, finite state machines describe a model wherein a "state" describes the static "snapshot" or configuration of elements within a modeled system. The visual image generated by the use of the term "machine" is that of a machine moving while an observer takes sequential snapshots, each of which illustrates the machine in different configurations or states. A "process" is the change in configuration or transformation of elements in the state machine, i.e., "state transition." "Event-driven state machines" highlight the trigger events used in event driven programming and messaging models; one visualizes an operator pushing buttons on the machine. And communication can then be described as the transferring of awareness between two or more systems, specifically, the awareness of the state transition of the sending system. One visualizes sending a picture of the new state of the machine, or perhaps a picture of the new state and a picture of the former state, as soon as the operator pushes the button on the machine.

A trigger event in healthcare is often a clinical observation result on a patient, a clinical order, or other clinical event. Observation results by clinicians characterize the clinical state of a patient during a specific time period. Changes in the clinical state of the patient are tracked by obtaining sequential observation results. When a transition occurs in the clinical state of a patient, the change may trigger a message alerting a physician. A new clinical order by the physician may trigger a message to the lab requesting a new lab test.

In the same manner, changes in the clinical state of the patient may be recorded in an electronic medical record system. The resulting state transition in the electronic medical record system may trigger a message to another electronic system. As illustrated, state transitions in electronic medical record systems are closely related to the clinical state changes in the patient and the awareness of care providers about the clinical state of the patient.

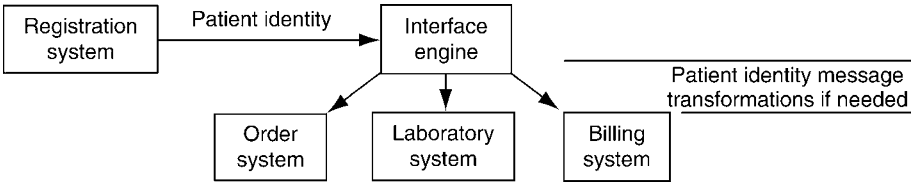
Finally, the study of the science of state changes, the communication of state changes, and the relationship of state changes to electronic communications in healthcare has been applied to more sophisticated techniques for dynamically orchestrating these communications into optimized process flows. Web service orchestrators have evolved as components of data integration engines such as service buses in healthcare. And optimized patient care processes are increasingly implemented with web service orchestrated electronic health record activities (www.infoway-inforoute.ca).

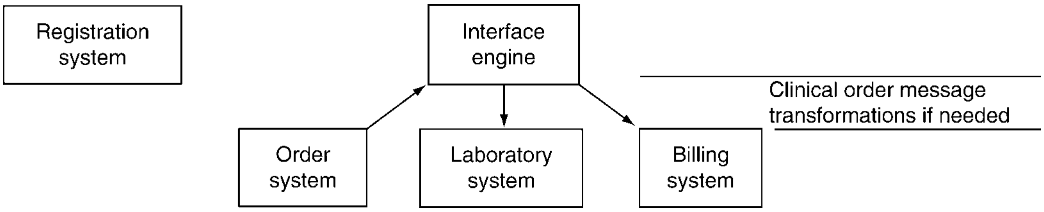
Key Applications

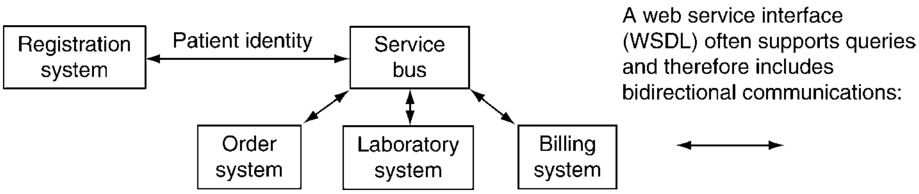
The most common first step in the installation of Hospital Information Systems (HIS) is to establish an identity system for enrolling patients as they enter the hospital, commonly referred to as a "registration system." When the registration system records a new patient identity or updates an older patient's demographic information, data about the patient is communicated via messages to other systems, such as order management systems, laboratory systems, and billing systems. This data includes a patient identifier and at least the first name, last name, date-of-birth, gender, and address. Traditionally, point-to-point interfaces are pre-configured within an interface engine from the registration system to the other systems, allowing broadcast of the identity of the new patient or updated demographic information to the other systems. If needed by any of the receiving systems, transformations may be applied to the syntax and semantics of the message that is outbound from the interface engine to each specific receiving system:

Later, if a clinical order is generated within the Order Management System for a patient, a subset of the fields in the Patient Identity message will be included in the clinical order and sent to the laboratory system and billing system via the interface engine. Again, transformations of the message may occur if needed by the receiving systems:

What is illustrated in the two figures above is that in a traditional interface engine scenario, the Registration System is never queried for additional Patient Identity information. Therefore, Patient Identity information is redundantly stored within the other systems. However, as populations managed by systems grow,






Registration system
Patient identity
Service bus
A web service interface (WSDL) often supports queries and therefore includes bidirectional communications:
Order system
Laboratory system
Billing system

the inefficiency of redundantly storing Patient Identity information grows as well. As a consequence, larger healthcare systems, such as regional or community-sized systems, are evolving towards a "just in time" communication of Patient Identity. A "service bus" replaces the Interface Engine, and the Registration System is dynamically queried as needed for additional Patient Identity information:

Cross-references

- ▶ Clinical Event
- ▶ Clinical Observation
- ▶ Clinical Order

Recommended Reading

1. Collen M. A History of Medical Informatics in the United States, 1950 to 1990. American Medical Informatics Association, Bethesda, MD, 1995.
2. Dolin R. et al. HL7 clinical document architecture, release 2. J. Am. Med. Inform. Assoc., 13(1):30–39, 2006.
3. Gill A. Introduction to the Theory of Finite-state Machines. McGraw-Hill, New York, 1962.
4. Lenz R. et al. A practical approach to process support in health information systems. J. Am. Med. Inform. Assoc., 9(6):571–585, 2002.
5. McDonald C. The barriers to electronic medical record systems and how to overcome them. J. Am. Med. Inform. Assoc., 4(3):213–221, 1997.
6. Russler D. et al. Influences of the unified service action model on the HL7 reference information model. In Proc. Symp. on Computer Applications in Medical Care, 1999, pp. 930–934.
7. White T. et al. Extending the LOINC conceptual schema to support standardized assessment instruments. J. Am. Med. Inform. Assoc., 9(6):586–599, 2002.

Internet Transactions

▶ Web Transactions

Interoperability in Data Warehouses

Riccardo Torlone
University of Rome, Rome, Italy

Synonyms

Data warehouse integration

Definition

The term refers to the ability of combining the content of two or more heterogeneous data warehouses, for the purpose of cross-analysis. This need emerges in a variety of practical situations. For instance, when different designers of a large company develop their data marts independently, or when different organizations involved in the same project need to integrate their data warehouses.

Data Warehouse interoperability is a special case of the general problem of database integration, but it can be tackled in a more systematic way because data warehouses are structured in a rather uniform way, along the widely accepted concepts of dimension and fact. As it happens in the general case, different degrees of interoperability can be pursued by adopting standards and/or by applying reconciliation techniques, likely specific for this context.

The problem is becoming increasingly relevant with the spreading of federated architectures. Nevertheless, it has been the focus of a few systematic works and numerous open problems remain to be solved.

Historical Background

In spite of its relevance, the problem of data warehouse integration has received little attention so far. Conversely, the general problem of databases integration has been studied in the literature extensively and several aspects, both at scheme and instance level, have been deeply investigated, such as the automatic matching of terms and the resolution of structural conflicts (see [8,12] for surveys on these topics).

In the specific context of data warehouses, Kimball [7] has identified the problem for the first time: he has investigated the integration of heterogeneous dimensions in a scenario of data warehouse design and has introduced the informal notions of dimension *conformity*. Intuitively, two dimensions are conformed if their share some information in a consistent way. This is an important requirement in *drill-across queries*, which are basically joins of different facts over common dimensions. The notion of conformity has been formalized and extended by Cabibbo and Torlone in the context of data mart integration [4] under the name of dimension *compatibility*: they have demonstrated that this property gives the ability to perform correct drill-across queries over heterogeneous data marts.

An issue related to the integration of data warehouses, which has been studied in the context of statistical databases, is the *derivability* of summary data. This notion has been defined by Sato [14] as the problem of deciding whether a summary data (which is, in a statistical database, the counterpart of a fact table) can be inferred from another summary data aggregated in a different way. The concept has been extended by Malvestuto [9], by considering the case in which the source is composed by several heterogeneous data sets: he proposes an algebraic approach to this problem and provides some necessary and sufficient conditions of derivability. Unfortunately, statistical databases have some similarity with multidimensional databases, but also some important diversities: this makes the application of these approaches to data warehouses not easy.

Some related work has been done on the problem of integrating a data warehouse with external data stored in XML [6] and in object-oriented [11] format, but just a few works have been devoted to the specific problem of the interoperability between heterogeneous data warehouses. They will be discussed in the following section.

While current commercial tools do not provide a complete support for data warehouse interoperability, they offer facilities that can be very useful in this framework, such as metadata import/export (using XML) and standardized ways to represent data (using a multidimensional model).

Foundations

Since data warehouse interoperability can be considered a special case of the problem of database integration, general data reconciliation techniques can be often used. For instance, methods for the automatic matching of terms or for the resolution of structural conflicts. In addition, it is possible to take advantage on the fact that, in this context, the data sources always have a multidimensional structure. Therefore, the problem can be addressed by focusing on the reconciliation of heterogeneous dimensions and facts. Following this observation, the section discusses: standards that can be adopted to support data warehouse interoperability, conflicts that can arise in this context, and methodologies that can be used to perform the integration.

Standards

An important support for interoperability can be provided by the adoption of standards. Initially, two industry standards have been proposed by multi-vendor organizations for data warehouses: the Open Information Model (OIM) developed by the Meta Data

Coalition (MDC), and the Common Warehouse Metamodel (CWM) developed by the Object Management Group (OMG) [16]. Later, MDC and OMG joined their efforts and proposed a new version of the CWM as the standard metadata model. The Common Warehouse Metamodel is a platform-independent specification for exchanging multidimensional data between different platforms and tools. It is based on the standards UML, XMI, and MOF, and provides a set of generic, external representations of metadata, called *metamodels*, that provide a comprehensive framework for data exchange. These metamodels can be used to describe the various components of the data warehouse architecture: data sources, ETL processes, multidimensional cubes, relational tables, and so on. However, it has been observed that their expressivity is not sufficient to capture all the complex semantics of conceptual multidimensional models, so they hardly can be used for effective integration of different data warehouses [13].

Conflicts

In the integration of different multidimensional data sources, a number of conflicts can arise, both at the schema and at the instance level.

- *Dimension conflicts*:
  - Schema: conflicts can arise on entity names (e.g., different names for the same dimensions and/or different names for similar levels of two dimensions) and on dimension hierarchies (similar dimensions organized over different levels of aggregation and/or inconsistencies on the roll-up relationships between levels).
  - Instance: conflicts can arise on member names (different names for the same members of different dimensions) and on the members of dimensions (similar dimensions populated by different members).
- *Fact conflicts*:
  - Schema: still, conflicts can arise on names (different names for the same measures) and on dimensions that differ in number and/or in the levels of aggregation.
  - Instance: conflicts can arise on measures (inconsistent values for the same measures and/or differences in scales).

As mentioned in the previous section, Cabibbo and Torlone [4] have identified a fundamental property that should be enforced while solving conflicts between heterogeneous data warehouses: dimension and fact *compatibility*. Two different dimensions d_1 and d_2 are compatible when their common information is consistent, that is, when aggregations computed over d_1 and d_2 and aggregations computed over the dimension obtained by merging d_1 and d_2 produce the same results. Having compatible dimensions and facts is important because it gives the ability to look consistently at data across data marts and to combine and correlate such data by means of drill across queries. Building on this notion, they have also identified a number of desirable properties that a *matching* between dimensions (that is, a correspondence between their levels) should satisfy: (i) the *coherence* of the hierarchies on levels, (ii) the *soundness* of the levels in correspondence, according to the members associated with them, and (iii) the *consistency* of the roll-up functions that relate members of different levels within the matched dimensions.

Integration Techniques

Two heterogeneous data warehouses can be combined if they share one or more dimensions and can be actually integrated if their facts can be joined, in a consistent way, over such common dimensions. It follows that a general methodology for achieving interoperability in data warehouses includes the following steps:

1. Identification of the facts that can be integrated and the dimensions of these facts that can be combined to perform the integration
2. Resolution of conflicts between common dimensions
3. Resolution of conflicts between facts to be integrated
4. Reconciliation and integration of dimensions and facts according to the desired level of interoperability

While this process can be supported by general reconciliation techniques based, for instance, on domain ontologies, it is possible to rely on specific techniques that take into account the rather standard structure of dimensions and facts. As usual, the level of interoperability can range from a scenario of loosely coupled integration, in which there is just the need to identify the common information between sources while preserving their autonomy, to a scenario of tightly coupled integration, in which the goal is rather merging the sources. In the former approach, queries are performed over a virtual view defined on the original sources, in the latter, queries are performed against a materialized view built from the sources.

Banek et al. [1] have addressed the problem of matching schema structures specific to data warehouses, the initial step of the above methodology. Their approach consists of two basic tasks. First, similarity matches between multidimensional structures are identified by comparing their names, data types and substructures (e.g., matches cannot violate the partial order in hierarchies). Then, heuristic rules, based on graph similarity, are used to choose the actual mappings, among the possible matches.

A methodology for the resolution of conflicts that guides the designers through the combination of independent data cubes has been proposed by Berger and Schrefl [2]. They also propose a specific language called SQL-MDi (SQL for multi-dimensional integration), supporting the methodology. In their approach, the goal is the generation of a tightly coupled architecture that combine heterogeneous multidimensional data sources into a materialized warehouse.

Cabibbo and Torlone [5] have proposed two practical approaches to the integration of autonomous data warehouses that try to enforce matchings satisfying the properties discussed in the previous section and refer to the scenarios of loosely and tightly coupled integration, respectively. As a preliminary tool, they introduce a powerful technique, the *chase of dimensions*, that can be used in both approaches to test for consistency and combine the content of the dimensions to integrate. This technique operates over a tableau populated by the members of the dimensions to be integrated, and makes use of the roll-up functions defined over such dimensions. Two integration algorithms are then proposed. The first algorithm provides the operations, expressed in an abstract algebra, that applied to the original dimensions, allow the specification of correct drill-across joins between the heterogeneous sources. The second algorithm generates new dimensions and facts, obtained by merging the original data sources, that constitute the reconciled data warehouse.

From a practical point of view, a general federated architecture supporting the interoperability of distributed and autonomous data warehouses has been proposed by Mangisengi et al. [10]. Tseng and Chen [15] have proposed a framework in which, after a resolution of conflicts, autonomous data cubes are first transformed into XML documents, then conflicts are solved by means of XQuery operations, and finally the access to integrated data is achieved through queries posed over an XML global view.

Key Applications

A common practice for building a data warehouse is to implement a series of data marts, each of which provides a dimensional view of a single business process [7]. These data marts should be based on common dimensions but what happens in practice is that, very often, different departments of the same company develop their data marts independently. It turns out that methods and tools for data warehouse reconciliation are very useful in such common situation.

Indeed, the need for combining autonomous data warehouse arises in other common scenarios. For instance, when different companies merge or get involved in a federated project or when there is the need to combine a proprietary data warehouse with data available elsewhere, for instance, in external and likely heterogeneous information sources, or in multidimensional data wrapped from the Web.

Furthermore, methods supporting data warehouse interoperability can be useful when there is the need to migrate a data mart from one implementation platform to another.

Future Directions

The area of data warehouse interoperability is largely unexplored and there is still a compelling need of systematic studies and effective tools. From a conceptual point of view, the problem needs a deeper investigation that takes into account, for instance, cases in which the structure of the data warehouses to be combined is non standard (e.g., for the presence of non-strict hierarchies or many-to-many relationships between facts and dimensions). In particular, the presence of irregular hierarchies makes the problem of dimension compatibility much harder since it requires complex tests at instance level. From a practical point of view, there is still a lack of effective tools specifically supporting the integration of autonomous and heterogeneous data warehouses.

Experimental Results

A preliminary tool supporting the interoperability of data warehouses has been recently proposed [3].

Cross-references

- ▶ Common Warehouse Metamodel
- ▶ Data Mart

- ▶ Data Warehouse
- ▶ Data Warehousing Systems: Foundations and Architectures
- ▶ Data Integration
- ▶ Multidimensional Modeling

Recommended Reading

1. Banek M., Vrdoljak B., Min Tjoa A., and Skocir Z. Automating the schema matching process for heterogeneous data warehouses. In Proc. 9th Int. Conf. Data Warehousing and Knowledge Discovery, 2007, pp. 45–54.
2. Berger S. and Schrefl M. Analysing multi-dimensional data across autonomous data warehouses. In Proc. 8th Int. Conf. Data Warehousing and Knowledge Discovery, 2006, pp. 120–133.
3. Cabibbo L., Panella I., and Torlone R. DaWaII: a tool for the integration of autonomous data marts. In Proc. 22nd Int. Conf. on Data Engineering, Demo session, 2006.
4. Cabibbo L. and Torlone R. On the Integration of Autonomous Data Marts. In Proc. 16th Int. Conf. on Scientific and Statistical Database Management, 2004, pp. 223–234.
5. Cabibbo L. and Torlone R. Integrating heterogeneous multidimensional databases. In Proc. 17th Int. Conf. on Scientific and Statistical Database Management, 2005, pp. 205–214.
6. Jensen M.R., Møller T.M., and Pedersen T.B. Specifying OLAP Cubes on XML Data. J. Intell. Inf. Syst., 17(2–3):255–280, 2001.
7. Kimball R. and Ross M. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, Wiley, 2nd edn., 2002.
8. Lenzerini M. Data integration: a theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
9. Malvestuto F.M. The Classification Problem with Semantically Heterogeneous Data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988, pp. 157–176.
10. Mangisengi O., Huber J., Hawel C., and Eßmayr W. A framework for supporting interoperability of data warehouse islands using XML. In Proc. 3rd Int. Conf. Data Warehousing and Knowledge Discovery, 2001, pp. 328–338.
11. Pedersen T.B., Shoshani A., Gu J., and Jensen C.S. Extending OLAP querying to external object databases. In Proc. Int. Conf. on Information and Knowledge Management, 2000, pp. 405–413.
12. Rahm E. and Bernstein P.A. A survey of approaches to automatic schema matching. VLDB J., 10(4):334–350, 2001.
13. Rizzi S., Abelló A., Lechtenbörger J., and Trujillo J. Research in data warehouse modeling and design: dead or alive? In Proc. ACM 9th Int. Workshop on Data Warehousing and OLAP, 2006, pp. 3–10.
14. Sato H. Handling Summary Information in a Database: Derivability. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1981, pp. 98–107.
15. Tseng F.S.C. and Chen C.W. Integrating heterogeneous data warehouses using XML technologies J. Inf. Sci., 31(3):209–229, 2005.
16. Vetterli T., Vaduva A., and Staudt M. Metadata standards for data warehousing: open information model vs. common warehouse metamodel. ACM SIGMOD Rec., 29(3):68–75, 2000.

Interoperation of NLP-based Systems with Clinical Databases

Yves A. Lussier, Matthew G. Crowson
University of Chicago, Chicago, IL, USA

Synonyms

Semantic web

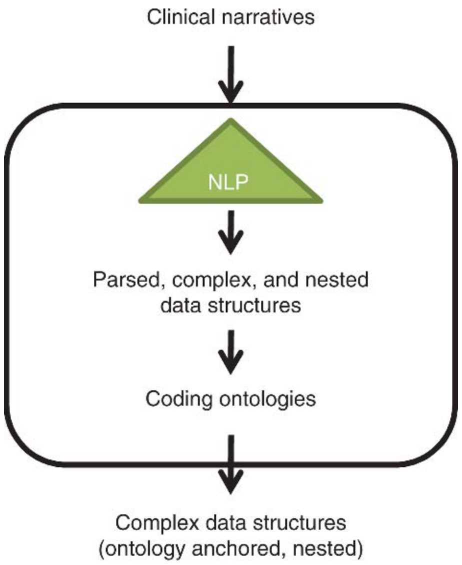
Definition

Natural language processing (NLP) is the automation of processes to interpret and understand meaning in human communications. In the life sciences, NLP assists in wide-scale storage and retrieval of specific "bundles" of clinical data embedded in patient charts which are commonly "free text". Both expert-system and statistical based NLPs have been in use in biomedicine for over three decades and some have shown an expert-like level of accuracy [1,3,6]. With the advent of electronic medical records, the sheer amount of data necessitates automated means for proper analysis to aid in patient care and research purposes.

Key Points

NLP commonly relies on indexing/tokenization, which is a process of breaking down text strings into data bundles. These bundles then need to be understood, which can be accomplished by mapping to clinical ontology. These clinical ontologies provide a means of disambiguating and organizing the mapped concepts to permit more efficient computation. See Fig. 1 below.

Once the tokenization process occurs, the data can be stored in a variety of methods. Fig. 2a demonstrates how a relational database requires an "unbundling process" to fit into its simplified, tabular storage structure. Although this is an efficient process for both storage and retrieval, data loss occurs as simplifying assumptions are made. As a consequence, during retrieval, queries can only be directed at the level of complexity that is stored. In contrast, Fig. 2b, post-tokenization, the data is not forced into a fixed structure, but rather is stored whole, i.e., in XML databases. XML format databases and ontology-anchoring [5] are important components of modern high-performance NLP systems. The retention of data complexity permits more nuanced and complex queries as the tokenized data can be retrieved in its entirety. This more rigorous model is more computationally intensive. However, recent advancements in processing power have made these arguments moot. The main upside of this model over the "lossy relational database model" is that it facilitates better storage and utilization of high-throughput generated biomedical data as researchers cannot always anticipate the clinical question posed *a priori*.


Clinical narratives
NLP
Parsed, complex, and nested data structures
Coding ontologies
Complex data structures (ontology anchored, nested)

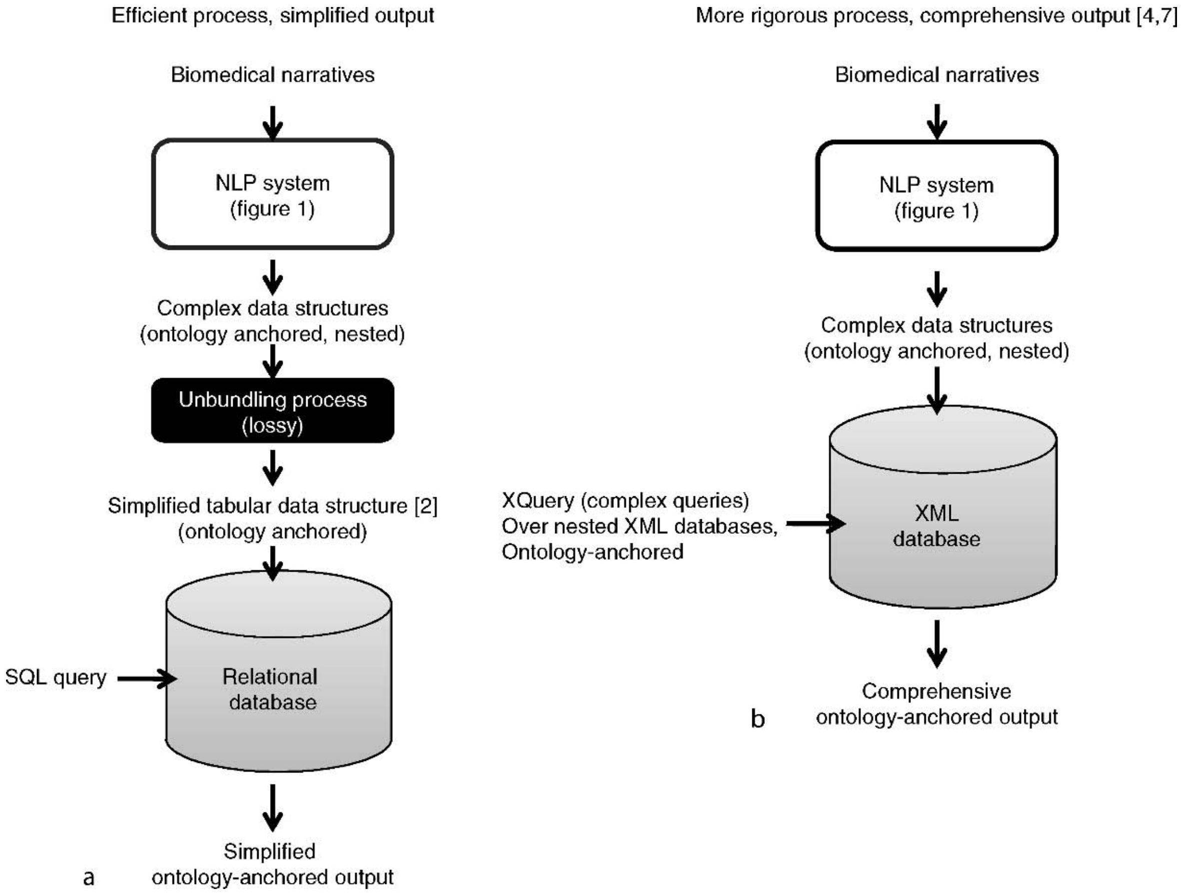
Interoperation of NLP-based Systems with Clinical Databases. Figure 1. NLP system.

Cross-references

- ▶ Clinical Data Acquisition, Storage and Management
- ▶ Clinical Ontologies
- ▶ Electronic Health Record
- ▶ Ontologies and life Science Data Management
- ▶ Storage Management
- ▶ XML Storage

Recommended Reading

1. Chapman WW. Dowling JN. Wagner MM. Classification of emergency department chief complaints into 7 syndromes: a retrospective analysis of 527,228 patients. Ann. Emerg. Med., 46(5):445–455, 2005.


Efficient process, simplified output
Biomedical narratives
NLP system (figure 1)
Complex data structures (ontology anchored, nested)
Unbundling process (lossy)
Simplified tabular data structure [2] (ontology anchored)
SQL query
Relational database
Simplified ontology-anchored output
a
More rigorous process, comprehensive output [4,7]
Biomedical narratives
NLP system (figure 1)
Complex data structures (ontology anchored, nested)
XQuery (complex queries) Over nested XML databases, Ontology-anchored
XML database
Comprehensive ontology-anchored output
b

Interoperation of NLP-based Systems with Clinical Databases. Figure 2. Two models of NLP-directed output.

2. Chen ES., Hripcsak G., and Friedman C. Disseminating natural language processed clinical narratives. AMIA Annu Symp Proc., 2006:126–30.
3. Collier N., Nazarenko A., Baud R., and Ruch P. Recent advances in natural language processing for biomedical applications. Int. J. Med. Inform, 75(6):413–417, 2006.
4. Friedman C., Hripcsak G., Shagina L., and Liu H. Representing information in patient reports using natural language processing and the extensible markup language. J. Am. Med. Inform. Assoc., 6(1):76–87, 1999.
5. Friedman C., Shagina L., Lussier Y., and Hripcsak G. Auomated encoding of clinical documents based on natural language processing. J. Am. Med. Inform. Assoc., 11(5):392–402, 2004.
6. Hripcsak G., Friedman C., Alderson PO., DuMouchel W., Johnson SB., and Clayton PD. Unlocking clinical data from narrative reports: a study of natural language processing. Ann. Intern. Med., 122(9):681–688, 1995.
7. Johnson SB., Campbell DA., Krauthammer M., Tulipano PK., Medonca EA., Friedman C., and Hripcsak G. A native XML database design for clinical document research. AMIA Annu Symp Proc., 2003:883.

Inter-Operator Parallelism

Esther Pacitti
INRIA and LINA, University of Nantes, Nantes, France

Synonyms

Pipelined and independent parallelism

Definition

Inter-operator parallelism enables different operators of the query to be executed in parallel, i.e., by different nodes. Given an operator tree for a query, there are two forms of inter-query parallelism: pipelined and independent parallelism. With pipelined parallelism, an operator that consumes data produced by another operator can proceed in parallel to that operator, as soon as it receives some data. With independent parallelism, two operators with no data dependency can proceed in parallel.

Key Points

Inter-operator parallelism is very attractive when the query is complex, i.e., has many operators on different relations. Thus, the more complex the query, the more opportunities there are for inter-operator parallelism. With pipeline parallelism, operators with a producer-consumer dependency can be executed in parallel. For instance, a select operator can be executed in parallel with the subsequent join operator. The advantage of such execution is that the intermediate select result is not materialized, thus saving memory and disk accesses. Pipeline parallelism puts constraints on the way the data at the consuming operator node is stored, i.e., it should fit in main memory. Independent parallelism is easier since it applies when there is no dependency between the operators that are executed in parallel. For instance, the two select operators on two different relations can be executed in parallel. This form of parallelism is very attractive because there is no interference between the nodes.

Cross-references

► Operator-Level Parallelism
► Parallel Data Placement
► Parallel Query Execution Algorithms

Inter-Query Parallelism

Nikos Hardavellas, Ippokratis Pandis
Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

User-level parallelism

Definition

Inter-query parallelism is a form of parallelism in the evaluation of database queries, in which several different queries execute concurrently on multiple processors to improve the overall throughput of the system.

Key Points

When multiple non-conflicting requests are submitted to a database management system, then the system can execute them in parallel to improve the overall throughput [1]. This form of parallelism is called inter-query parallelism. Inter-query parallelism is a consequence of the concurrency of user requests. It is orthogonal to intra-query parallelism, in which several processors cooperate for the faster execution of a single query [2]. Both forms of parallelism can co-exist in a database management system. Inter-query parallelism is common in on-line transaction processing (OLTP), where multiple concurrent users submit requests to the system. It is a challenge for the database management system to achieve high performance and maintain the ACID properties in the presence of multiple concurrently executing requests or transactions.

Cross-references

- ▶ ACID properties
- ▶ Intra-Operator Parallelism
- ▶ Intra-Query Parallelism
- ▶ Operator-Level Parallelism

Recommended Reading

1. DeWitt D.J. and Gray J. Parallel database systems: the future of high-performance database computing. Commun. ACM, 35(6):85–98, 1992.
2. Graefe G. Encapsulation of Parallelism in the Volcano Query Processing System. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 102–111.

Interval-based Temporal Models

▶ Period-Stamped Temporal Models

Intra-operator Parallelism

Esther Pacitti
INRIA and LINA, University of Nantes, Nantes, France

Synonyms

Single instruction multiple data (SIMD) parallelism

Definition

Intra-operator parallelism enables an operator which accesses some data to be executed by multiple nodes, each working on a different partition of the data. With intra-operator parallelism, the same operator is applied to multiple partitions, thereby dividing the response time by the number of nodes. Intra-operator parallelism exploits the various forms of data placement and dynamic partitioning using specific algorithms for the different relational operators.

Key Points

Intra-operator parallelism is based on the decomposition of a relational operator into a set of independent operator instances, each processing a different relation partition. This decomposition is done using static or dynamic partitioning of the relations. Static partitioning corresponds to the initial data placement and is typically exploited by the select or scan operators. Dynamic partitioning, i.e., repartitioning a relation a different way, is useful for binary operators that are costly. One main repartitioning solution is hashing on some important attribute, e.g., join attribute. Intra-operator parallelism is easier for unary operators such as scan or select and more difficult for binary operators such as join, in which case, more complex parallel execution algorithms are necessary.

Cross-references

- ▶ Parallel Data Placement
- ▶ Parallel Query Execution Algorithms

Intra-Query Parallelism

Nikos Hardavellas, Ippokratis Pandis
Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

Query parallelism

Definition

Intra-query parallelism is a form of parallelism in the evaluation of database queries, in which a single query is decomposed into smaller tasks that execute concurrently on multiple processors.

Key Points

Intra-query parallelism is achieved when several processors cooperate in the execution of a single query to improve the query's response time. Intra-query parallelism is orthogonal to inter-query parallelism, in which multiple independent requests execute concurrently on several processors to improve the overall system throughput.

There exist two forms of intra-query parallelism: operator-level parallelism and intra-operator parallelism. Operator-level parallelism is obtained by executing concurrently several operators of the same query. For example, consider a simple query that consists of a scan operator and an aggregation. The scan operator uses a selection condition to filter tuples. The aggregation calculates some statistics over all qualifying tuples. Operator-level parallelism is obtained by executing concurrently the scan operation as a single task and the aggregation as another, pipelining tuples from the scan to the aggregation as soon as they are produced. To realize this form of parallelism, all participating operators must be pipelineable. Non-pipelineable operators cannot participate because they need their entire input before executing, thereby halting the pipeline.

Intra-operator parallelism is obtained by executing concurrently multiple instances of an operator, with each instance working on a subset of the data. Intra-operator parallelism is based primarily on partitioning the input relation into non-overlapping data segments. In the example above, intra-operator parallelism is achieved by dividing the input relation into non-overlapping data segments and executing the scan and the aggregation of the segments in parallel, followed by a final merge of the results. The interested reader is referred to [1] and [2] as more comprehensive readings.

Cross-references

- ▶ Data Partitioning
- ▶ Inter-Query Parallelism
- ▶ Intra-Operator Parallelism
- ▶ Operator-Level Parallelism
- ▶ Stop-&-Go Operator

Recommended Reading

1. DeWitt D.J. and Gray J. Parallel database systems: the future of high-performance database computing. Commun. ACM, 35(6):85–98, 1992.
2. Graefe G. Encapsulation of parallelism in the volcano query processing system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 102–111.

Intrinsic Time

▶ Valid Time

Intrusion Detection Technology

Tyrone Grandison, Evimaria Terzi
IBM Almaden Research Center, San Jose, CA, USA

Definition

Intrusion Detection (ID) is the process of monitoring events occurring in a system and signalling responsible parties when interesting (suspicious) activity occurs.

Intrusion Detection Systems (IDSs) consist of (i) an agent that collects the information on the stream of monitored events, (ii) an analysis engine that detects signs of intrusion, and (iii) a response module that generates responses based on the outcome from the analysis engine.

Historical Background

The concept of ID has existed for decades in the domains of personal home security, defense and early-warning systems. However, automated IDSs emerged in the public domain in 1980 [2] and sought to identify possible violations of the system's security policy by a user or a set of users.

One of the basic elements of an IDS is the audit log that captures the system activity. The initial IDSs exposed to the academic community stored operating system actions, i.e., addressed the operating system layer. Over time, other IDSs have emerged that store different artifacts, and try to identify intrusive behaviors at different layers of operation. The following layers of operation can be easily identified:

Operating System: The logs in this layer contain information from the kernel and other operating system components and help determine if an attacker is trying to compromise the OS.

Network: At the network layer, communication data is analyzed to determine if an attacker is trying to access one's network.

Application: Application level IDSs examine the operations executed in an application to ascertain if the application is being manipulated to extract behavior that is prohibited. Database-specific IDSs form an important group of application-level IDSs. Examples of such systems include Discovery [14] and RIPPER [10]. Due to the sensitive information stored in database systems, issues related to database-specific IDSs were among the first to be addressed [3,9,15].

The above categorization is historical and mostly depends on the type of log data the IDS uses in order to identify abnormal patterns. Irrespective of the operational layer, the very basic detection techniques used by different IDSs have some common basis, which are described in the next section.

Foundations

A high-level categorization of IDSs is given along with an abstract idea of how they work. For a more complete discussion on IDSs see [1,8,13].

Traditionally, there are two basic approaches to intrusion detection; *anomaly detection* and *misuse detection*. In anomaly detection the goal is to define and characterize legitimate behaviors of the users, and then detect anomalous behaviors by quantifying deviations from the former. However, identifying the distance between anomalous and legitimate behaviors is a rather difficult notion to quantify.

Anomaly detection can be *static* or *dynamic*. A static anomaly detection system is based on the assumption that there is a static portion of the system being monitored. Static portions of the system can be represented as a binary string or a set of binary strings (like files). If the static portion of the system ever deviates from its original form, either an error has occurred or an intruder has altered the static portion of the system.

Dynamic anomaly detectors are harder to build since building them requires a definition of behavior, which is often defined as a sequence (or partially ordered sequence) of distinct events. Differentiating between normal and anomalous activity in dynamic anomaly detection systems is much harder than the problem of distinguishing changes in static elements. Dynamic anomaly detection systems usually create a *base profile* to characterize normal, acceptable behavior. A profile usually consists of a set of observed measures of behavior for a selected set of dimensions. After initializing the base profile the dynamic anomaly detection systems are similar to the static ones; they monitor the behavior by comparing the current behavior with that implied by the base profile. Typically, there is a wide variation of acceptable behaviors and statistical methods are employed to measure deviation from the base profile. The main challenge in dynamic anomaly detection systems is that they must build accurate base profiles and then recognize behaviors that significantly deviate from the profile.

The main advantage of dynamic anomaly detection systems is that they do not require any configuration since they automatically learn the behavior of large number of subjects. Lacking prior knowledge of how an intrusion would manifest itself anomaly detection systems are capable of identifying novel intrusions or variations of known intrusions. However, building base profiles and defining measures of deviations from them is not an easy computational task. For that reason it has been an active area of research, in which several machine learning, time-series analysis and other data-analysis techniques have been employed [3,4–7,11,12].

Misuse detection is concerned with identifying intruders who are attempting to break into a system using some known technique. If a system security administrator was aware of all the known vulnerabilities then a misuse detection system would be able to identify their occurrences and eliminate them. A fairly precisely known kind of intrusion is known as *intrusion scenario*. A misuse detection system compares current system activity to a set of intrusion scenarios in an attempt to identify a scenario in progress.

The differentiating factor between the various misuse detection techniques is the model used for describing bad behaviors that constitute intrusions. *Rules* have been primarily used to model the system-administrator's knowledge about the system. Rule-based systems accumulate large numbers of rules which usually prove difficult to interpret and modify. In order to overcome these problems model-based rule organizations and state-transition representations were proposed. These modeling approaches are more intuitive particularly in misuse detection systems where users need to express and understand scenarios.

The main advantage of a misuse detection systems is that the system knows for a fact how normal behavior should manifest itself. This leads to a simple and efficient processing of the audit data. The obvious disadvantage of such systems is that the specification of the signatures to be detected is a time-consuming task that requires lots of domain knowledge. At the same time, misuse detection systems lack the ability to identify novel intrusion profiles.

Key Applications

A generic classification of the types of attacks that ID systems have traditionally tried to cope with are described below. The classification is mainly inspired by the one provided in [1].

- *External break ins*: When an unauthorized user tries to gain access to a computer system.
- *Masquerander (internal) attacks*: When an authorized user makes an attempt to assume the identity of another user. These attacks are called also internal because they are caused by already authorized users.
- *Penetration attack*: In this attack, a user attempts to directly violate the system's security policy.
- *Leakage*: Moving potentially sensitive data from the system.

- *Denial of Service*: Denying other users the use of system resources, by making these resources unavailable to other users.
- *Malicious use*: Miscellaneous attacks such as file deletion, viruses, resource hogging etc.

Future Directions

One of the major concerns associated with IDSs and their utility is their run-time efficiency. More often than not, IDSs consume too many system resources in order to be effective. Developing resource-aware IDSs systems raises some interesting challenges. One possible way of addressing this concern is via building *Mega Intrusion Detection Systems*. These would be systems that simultaneously monitor all operational layers. That is, the system administrator will not have to run a different ID software for operating system and application specific attacks, but just a single system that will simultaneously be able to detect intrusions in all the desired operational layers. Such systems are expected to be less resource demanding, however their development will certainly create several new design challenges.

This entry has mainly focused on IDSs and described them as mechanisms that guarantee other systems' security. However, IDSs are themselves systems and as such they have their own security risks. Therefore, they also require some protection to prevent an intruder from manipulating the intrusion detection system itself.

Recommended Reading

1. Axelsson S. Research in intrusion detection systems: a survey. In Technical Report 98-17 (revised in 1999). Chalmers University of Technology, 1999.
2. Bace R.G. Intrusion Detection. Macmillan Technical, New York, 2000.
3. Bertino E., Kamra A., Terzi E., and Vakali A. Intrusion detection in rbac-administered databases. In Proc. Asia-Pacific Comp. Syst. Arch. Conf., 2005, pp. 170–182.
4. Bertino E., Leggieri T., and Terzi E. Securing DBMS: characterizing and detecting query floods. In Proc. 7th Int. Conf. on Information Security, 2004, pp. 195–206.
5. Huang Y., Fan W., Lee W., and Yu P. Cross-feature analysis for detecting ad-hoc routing anomalies. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2003, pp. 478.
6. Kruegel C., Mutz D., Robertson W., and Valeur F. Bayesian event classification for intrusion detection. In Proc. Asia-Pacific Comp. Syst. Arch. Conf., 2003.
7. Lane T. and Brodley C.E. Temporal sequence learning and data reduction for anomaly detection. ACM Trans. Inf. Syst. Secur., 2(3):295–331, 1999.
8. Lee W. and Fan W. Mining system audit data: opportunities and challenges. ACM SIGMOD Rec., 30(4):35–44, 2001.
9. Lee V.C.S., Stankovic J.A., and Son S.H. Intrusion detection in real-time database systems via time signatures. In Proc. IEEE Real Time Technology and Applications Symposium, 2000, pp. 124–133.
10. Lee W., Stolfo S.J., and Mok K.W. A data mining framework for building intrusion detection models. In Proc. IEEE Symp. on Security and Privacy, 1999, pp. 120–132.
11. Lee W. and Xiang D. Information-theoretic measures for anomaly detection. In IEEE Symp. on Security and Privacy, 2001, pp. 130–143.
12. Ramadas M., Ostermann S., and Tjaden B.C. Detecting anomalous network traffic with self-organizing maps. In Proc. 6th Int. Symp. Recent Advances in Intrusion Detection, 2003, pp. 36–54.
13. Stolfo S.J., Lee W., Chan P.K., Fan W., and Eskin E. Data mining-based intrusion detectors: an overview of the columbia ids project. ACM SIGMOD Rec., 30(4):5–14, 2001.
14. Tener W.T. Discovery: an expert system in the commercial data security environment. In Proc. 4th IFIP TCII Int. Conf. on Security, 1986, pp. 261–268.
15. Wenhui S. and Tan D. A novel intrusion detection system model for securing web-based database systems. In Proc. 25th Annual Int. Computer Software Applications Conf., 2001, pp. 249–.

Inverse Document Frequency

IADH OUNIS
University of Glasgow, Glasgow, UK

Synonyms

IDF

Definition

The inverse document frequency (*IDF*) is a statistical weight used for measuring the importance of a term in a text document collection. The document frequency *DF* of a term is defined by the number of documents in which a term appears.

Key Points

Karen Sparck-Jones first proposed that terms with low document frequency are more valuable than terms with high document frequency during retrieval [2]. In other words, the underlying idea of *IDF* is that the more frequently the term appears in the collection, the less informative the term is.

In its simplest form, the *IDF* weight of a term is assigned as follows [3]:

$$IDF = \log_2 \frac{N}{DF} \qquad (1)$$

where *N* is the number of documents in the collection, and *DF* is the document frequency of the term, i.e., the number of documents in which the term appears.

There have been different variations of the *IDF* weight in the literature. For example, Robertson and Walker proposed the following formula [1]:

$$IDF = \log \frac{N - DF + 0.5}{DF + 0.5} \qquad (2)$$

where *0.5* is added to avoid having infinite values brought by zero *DF* values.

Cross-references

▶ Probabilistic Ranking Principle

Recommended Reading

1. Robertson S.E. and Walker S. On relevance weights with little relevance information. In Proc. 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp 16–24, 1997.
2. Sparck-Jones K. A statistical interpretation of term specificity and its application in retrieval. J. Doc., 28(1):11–20, 1972.
3. Sparck-Jones K. Index term weighting. Inform. Storage Retr., 9(11):619–633, 1973.

Inverse Element Frequency

▶ Term Statistics for Structured Text Retrieval

Inverted Files

Edleno Silva de Moura[1], Marco Antonio Cristo[2]
[1]Federal University of Amazonas, Manaus, Brazil
[2]FUCAPI, Manaus, Brazil

Synonyms

Inverted index; Full text inverted index; Postings file

Definition

An Inverted file is an index data structure that maps content to its location within a database file, in a document or in a set of documents. It is normally composed of: (i) a vocabulary that contains all the distinct words found in a text and (ii), for each word t of the vocabulary, a list that contains statistics about the occurrences of t in the text. Such list is known as the inverted list of t. The inverted file is the most popular data structure used in document retrieval systems to support full text search.

Historical Background

Efforts for indexing electronic texts are found in literature since the beginning of the computational systems. For example, descriptions of Electronic Information Search Systems that are able to index and search text can be found in the early 1950s [4].

In a seminal work, Gerard Salton wrote a book in 1968, containing the basis for the modern information retrieval systems [6], including a description of a model largely adopted up to now for indexing texts, known as Vector Space Model. The inverted file was adopted as the index structure for implementing the Vector Space Model. It has been widely applied and studied since then.

Foundations

Inverted files allow fast search for statistics related to the distinct words found in a text. They are projected for using words as the search unit, which restricts their use in applications where words are not clearly defined or in applications where the system does not use words as the search unit.

The statistics stored in an inverted file may vary according to the target application. Two alternatives usually found in literature are to record the position of all word occurrences in given text and to record general statistics about the word occurrences across text units. Indexing all the word occurrences is useful in applications where positional information should be taken into account, such as when it is necessary to allow search for phrases or proximity queries. Inverted files that store word statistics are usually deployed in systems that adopt information retrieval models, such as the Vector Space Model. In this particular case the text is divided into units of information, usually documents. For instance, in web search engines, these units are the pages crawled from the web, while the whole set of pages compose the indexed text.

Querying a word in an inverted index consists in first locating the word in the vocabulary and getting the position of its inverted list. This operation can be performed in O(1) by using a hash algorithm. The inverted list of the word is then accessed in order to provide the search results. The vocabulary usually requires a sub-linear space when compared to the size of the inverted list, which makes it usually far smaller than these lists.

Word occurrences are stored in inverted files for applications where positional information should be taken into account, such as when it is desirable to provide support to phrase search or proximity queries. To search for a phrase or proximity pattern (where the words must appear consecutively or close to each other, respectively), each word is searched separately. Then, the resulting lists of occurrences are intersected considering the consecutiveness or closeness of the word positions in the text. The cost to perform such type of queries can be reduced by adopting an auxiliary data structure, known as next word index, which includes information about the next word in the positional inverted list entries. This alternative can significantly reduce the query processing times for phrase queries. Another choice could be to index pairs of consecutive words, but then the vocabulary would be much larger, which would make this option unfeasible.

Building an Inverted File

The texts indexed nowadays by search systems are usually too large for allowing the creation of inverted files completely in main memory. Disk-based algorithms for generating compressed inverted files have been extensively studied in the literature. An example is the *multiway merging* algorithm described in [7].

An implementation of this algorithm can be performed in three phases a, b, and c. In phase a, all documents are sequentially read from disk and parsed into index terms. This allows creating a perfect hashed vocabulary and also allows the application of filtering algorithms to remove undesired words from the index. For instance, search systems usually remove frequent words included in a pre-computed list, known as *stop words*. Heuristics for removing typos can also be adopted in this phase by analyzing the frequency of occurrences of each word in the text. In phase b, all documents are again sequentially read from disk and again parsed into index terms. Triplets composed of an index term number k_i, a document number d_j, and a frequency $f_{i,j}$ are then formed and inserted into a buffer B in main memory. Whenever this buffer fills (i.e., whenever a run is completed), the partial inverted lists are sorted in decreasing order of the frequencies $f_{i,j}$ (or another desired order according to the application), and stored in a temporary file F. In phase c, a disk-based multiway merge is done to combine the partial inverted lists into final lists. The details are shown in Fig. 1. Note that this example considers that the index stores the frequency of each word in each document. However, the algorithm can easily be adapted to create an index to store word occurrence positions or other type of statistics about the text, according to the target application.

The sequential algorithm shown above uses two passes for reading and parsing of the documents in the collection. This allows building a perfect hashed vocabulary which provides for direct access to any inverted

```
  (a1) read all documents sequentially, parse
       into index terms and create perfect hashed
       vocabulary
  (a2) create memory buffer B
a (a3) R = 0 /* initialize number of runs */

  foreach $d_j$ do
b begin
  (b1) read $d_j$, parse into index terms
  (b2) foreach $k_i \in d_j$ do B = B + [$k_i, d_j, f_{i,j}$]
  (b3)  if 'buffer B is full' then
       begin
       (b3.1) do sort triplets on $k_i, f_{i,j}$
       (b3.2) F = write-disk (B)
       (b3.3) B = $\phi$; R = R + 1
       end
  end
c multiway-disk-merge (F,R)
```

Inverted Files. Figure 1. A disk-based algorithm for building inverted files.

list with no need to lookup at a vocabulary entry. Thus, once the perfect hash has been built, it is no longer necessary to keep the vocabulary in memory (all significant memory consumption is now represented by the buffer *B* which stores the inverted lists).

In cases where the text is too large, as it happens in search engines that try to index the whole web, distributed algorithms should be adopted for building inverted files. The current best alternative for building distributed inverted files is also the simplest solution. It partitions the text into small sub-collections, each of them fitting in a single machine. A local index is built for each sub-collection; when queries arrive, they are submitted to every sub-collection and evaluated against every local index. A final step merges the answers produced by each individual machine yielding a single ranking of results to the final users.

Compression

A technique to reduce the space requirements of inverted files is to compress the index. The key idea to reduce the size of inverted files is that the inverted list entries related to each word can be sorted in increasing order, and therefore the gaps between consecutive positions can be stored instead of the absolute values. Then, compression techniques for small integers can be used. As the gaps are smaller for longer lists, longer lists can be compressed better. Previous work has shown that inverted files can be reduced up to 10% of their original size without degrading the performance, and even the performance may improve because of reduced I/O [7].

Another alternative for reducing the space requirements of an inverted file and the query processing regarding the access to the index is to minimize the number of indexed entries by applying static pruning methods. Pruning methods try to avoid processing index entries without cause loss of quality in the final results produced by the search system. They can be classified as *dynamic* and *static*. Dynamic methods maintain the index completely stored on disk and use heuristics to avoid reading unnecessary information at query processing time. In this case, the amount of pruning performed varies according to the user queries, which represents an advantage, since the methods can be better adapted to each specific query. In contrast, static methods try to predict, at index construction time, the entries which will not be useful at query processing time. These entries are then removed from the index. For this reason, static methods can be seen as lossy compression methods. Static methods offer the advantage of both reducing the disk storage costs and time to process each query. A system that uses both static and dynamic methods can also be implemented to take advantage of the two types of pruning options [5].

Updating Operations

In applications where the indexed text changes over the time, with portions being removed, added or changed in the text, it is necessary to reflect such changes in the inverted file. The simplest approach is to rebuild the whole index, which may be acceptable if the index can be updated offline and the indexing time is small. However, if such conditions do not apply, more sophisticated strategies should be adopted. Several index maintenance strategies can be found in literature [8]. They can be divided into three categories, with the index rebuilding being the first obvious choice. The second category is the intermittent merge, where small indexes to register updates are stored in main memory, making the update inexpensive. In this case, the temporary main memory index and the disk index should be merged at query processing time. A real update should be periodically performed to avoid a memory overflow in the temporary index. The third category is the incremental update. It updates the main index term by term using a process similar to the mechanisms used for maintaining variable-length records in conventional database management systems.

Query Processing

The query processing over inverted files can be performed in two distinct forms, being based on a term order or on a document order basis [2]. In the term order basis, each inverted list is processed one at a time. The partial list of results obtained after processing each list is stored in memory, which means this method may require additional memory. The second form of processing queries is the document order processing, where whenever a document information is found in one of the lists, all information about this document is automatically read from the remaining inverted lists of terms present in the query. The document ordering method requires the inverted lists to be stored sorted by document number, or by occurrence when the index store all term occurrences. Previous work indicate the term ordering method results in faster query evaluation. However, for small queries, which are common on many search applications, this difference becomes smaller. A combination of document order and term order may also be implemented, by processing the inverted lists in blocks.

A final comment about query processing is that it can be sped up by using cache strategies. At least three distinct cache layers have been proposed in literature. First, the system can adopt a cache of inverted lists to keep the most frequent portions of the lists in memory. Second, it can also be used a cache of results, which takes the final results provided to the users in a cache. Finally, a projection cache containing frequent intersections of lists can also be adopted. Previous work in literature conclude that these cache techniques can significantly increase the maximum capacity of systems for query processing [3].

Key Applications

Inverted files are by far the most applied indexing structures in text search systems. Such indexes are used, for instance, in the popular large scale web search engines.

Cross-references

- ▶ Compressed Inverted Files
- ▶ Information Retrieval Models
- ▶ Lossless Data Compression
- ▶ Text Retrieval
- ▶ Web Search and Crawling

Recommended Reading

1. Baeza-Yates R. and Ribeiro-Neto B. Modern Information Retrieval. Addison Wesley, Reading, MA, 1999.
2. Kaszkiel M. and Zobel J. Term-ordered query evaluation versus document-ordered query evaluation for large document databases. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 343–344.
3. Long X. and Suel T. Three-level caching for efficient query processing in large Web search engines. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 257–266.
4. Luhn H.P. A statistical approach to mechanized encoding and searching of literary information. IBM J. Res. and Dev., 309–317, October 1957.
5. de Moura E.S., dos Santos C.F., Fernandes D.R., Silva A.S., Calado P., and Nascimento M.A. Improving web search efficiency via a locality based static pruning method. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 235–244.
6. Salton G. Automatic Information Organization and Retrieval. McGraw-Hill, New York, NY, 1968.
7. Witten I., Moffat A., and Bell T. Managing Gigabytes, 2nd edn. Morgan Kaufmann, Los Altos, CA, 1999.
8. Zobel J. and Moffat A. Inverted Files for Text Search Engines. ACM Comput. Surv., 38(2):1–56, July 2006.

Inverted Index

- ▶ Inverted Files
- ▶ Text Index Compression

Inverted Indexes

- ▶ Index Creation and File Structures

IP Storage

KAZUO GODA
The University of Tokyo, Tokyo, Japan

Definition

IP Storage is a storage device which has the capability of communicating over an IP network. The term IP Storage is usually identified with a storage device which provides block-level I/O services rather than file access services. Derived from the original meaning, the term IP Storage is sometimes used to refer to an IP SAN.

Key Points

The benefit of utilizing IP technology is its cost efficiency. Having been used for several decades, IP technology is highly mature. IP family protocols have been standardized and cheap hardware products have wide interoperability. The market has a number of excellent administration tools and educated administration personnel for IP technology. Designing and operating SANs on top of IP technology is much cheaper and easier than with Fiber Channel technology. Thus IP technology is sometimes considered to be replacement of Fiber Channel technology in entry-level SANs.

Internet SCSI (iSCSI), Internet Fiber Channel Protocol (iFCP) and Fiber Channel over Internet Protocol

(FCIP) are three major network protocols used for IP Storage. iSCSI transmits the SCSI protocol over IP networks by encapsulating SCSI data in TCP/IP packets. That is, the idea of iSCSI is to replace classical SCSI bus cables with IP networks. iSCSI is used mainly for transferring data between servers and storage devices and among storage devices. A storage device which communicates over the iSCSI protocol is often called an iSCSI target, whereas a server which accesses such an iSCSI device is called an iSCSI initiator. In contrast, iFCP and FCIP can encapsulate Fiber Channel frames in TCP/IP packets. These protocols are usually implemented in network devices such as Fiber Channel switches and routers, thus enabling bridging two or more Fiber Channel SANs by the use of IP networks. Although iSCSI and iFCP/FCIP are both IP storage techniques, they are used in real systems in different ways. That is, iSCSI is mainly used for constructing a local SAN at low cost, whereas iFCP and FCIP are utilized for integrating remote Fiber Channel SANs (often called SAN islands.)

Cross-references

▶ Storage Network Architectures

Recommended Reading

1. Clark T. IP SANS: A Guide to iSCSI, iFCP, and FCIP Protocols for Storage Area Networks. Addison-Wesley Professional, Reading, MA, 2001.
2. Troppens U., Erkens R., and Müller W. Storage Networks Explained. Wiley, New York, 2004.

ISAM File

▶ Index Sequential Access Method (ISAM)

iSCSI

▶ Storage Protocols

ISO 19136

▶ Geography Markup Language

Isolation

▶ ACID Properties
▶ Two-Phase Locking

Iteration

▶ Loop

Iterator

Evaggelia Pitoura
University of Ioannina, Ioannina, Greece

Synonyms

Iterator; Cursor; Synchronous pipelines

Definition

In general terms, a physical operator is an implementation of a relational operator. For a relational operator, there are many alternative physical operators that implement it, for instance, sort-merge and hash-join provide alternative algorithms for implementing join. The query execution engine provides generic implementations of all physical operators. Typically, each physical operator supports a uniform *iterator interface* that hides any internal implementation details and allows operators to be combined together. The iterator interface includes the functions: (i) *open()* that prepares an operator to produce data, (ii) *next()* that produces an output tuple, and (iii) *close()* that performs the final bookkeeping.

Key Points

During query processing, an input query is transformed to a plan to be executed by the query execution engine. An execution plan can be thought of as a dataflow graph where the nodes correspond to the physical operators and the edges represent the data flow among the physical operators. Generally speaking, a physical operator is an implementation of a relational operator, for instance, sort-merge and hash-join are physical operators providing alternative algorithms for the implementation of join. The query execution engine provides generic implementations of all physical operators. A plan is executed by calling its physical operators in some (possibly interleaved) order. In most modern database systems, each physical operator supports a uniform *iterator interface* that allows an operator in the plan to get results from its input operators one tuple at a time, hiding the internal implementation details of each operator.

The iterator interface for an operator includes the functions open(), next(), and close(). The *open()* function initializes the state of the operator by allocating the appropriate input and output buffers and passing any related arguments. The code for the *next()* function calls the next() function recursively on each input operator until an output tuple is generated. The state of the operator keeps track of how much input has been consumed. Finally, when all output tuples have been produced, through repeated calls of the next() function, the *close()* function deallocates the state information and performs any other final bookkeeping.

By providing a common interface to all physical operators, any two physical operators can be plugged together. Furthermore, the iterator interface supports a pipeline model for the execution of the plan.

The iterator interface is also employed to encapsulate access methods such as the various kinds of indexes that the database system supports. In this case, open() can be used to pass the corresponding selection condition. Finally, parallelism and network communications can be encapsulated within special *exchange iterators*.

Cross-references

▶ Access Path
▶ Evaluation of Relational Operators
▶ Pipelining
▶ Query Plan

Recommended Reading

1. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2): 73–170, 1993.
2. Hellerstein J.M., Stonebraker M., and Hamilton J. Architecture of a database system. Found. Trends Databases, 1(2):141–259, 2007.
3. Ramakrishnan R. and Gehrke J. Database Management Systems. McGraw-Hill, New York, 2003.

J

J2EE

▶ Java Enterprise Edition (JEE)

Java Annotations

▶ Java Metadata Facility

Java Application Server

▶ Application Server

Java Database Connectivity

Changqing Li
Duke University, Durham, NC, USA

Synonyms

JDBC

Definition

Java Database Connectivity (JDBC) [1] is an Application Programming Interface (API) for the Java programming language that enables Java programs to execute Structured Query Language (SQL) statements and defines how an application accesses a database. JDBC provides methods to query and update data in a database. Different from ODBC (Open Database Connectivity), JDBC is oriented towards relational databases only.

Key Points

JDBC (pronounced as separate letters) is similar to ODBC, but it is designed specifically for Java programs, whereas ODBC is language-independent. The Java Standard Edition includes the JDBC API as well as an ODBC implementation of the API enabling connections to any relational database that supports ODBC [1].

JavaSoft, a subsidiary of Sun Microsystems, developed JDBC. Since the release of Java Development Kit (JDK) 1.1, JDBC has been part of the Java Standard Edition. JDBC has been developed under the Java Community Process since version 3.0. JDBC 3.0 (included in Java 2 Standard Edition (J2SE) 1.4) is specified by Java Specification Request (JSR) 54, the JDBC Rowset additions are specified by JSR 114, and JDBC 4.0 (included in Java Standard Edition 6) is specified by JSR 221.

Multiple implementations of JDBC can exist and can be used by the same application. A mechanism is provided by the API to dynamically load the correct Java packages and register them with the JDBC Driver Manager, a connection factory for creating JDBC connections.

Creating and executing statements are supported by JDBC connections. These statements may either be update statements such as SQL CREATE, INSERT, UPDATE and DELETE or be query statements using the SELECT statement. In addition, a statement may invoke a stored procedure.

Because Java itself runs on most platforms, and since nearly all relational database management systems (DBMSs) support SQL, JDBC makes it possible to write a single database application that can run across different DBMSs on different platforms.

Cross-references

▶ Data Integration
▶ Database Adapter and Connector
▶ Interface
▶ .NET Remoting
▶ Open Database Connectivity
▶ Web 2.0/3.0
▶ Web Services

Recommended Reading

1. Hamilton G., Cattell R., and Fisher M. JDBC Database Access with Java: A Tutorial and Annotated Reference. Addison Wesley, Boston, MA, USA, 1997.

Java EE

► Java Enterprise Edition (JEE)

Java Enterprise Edition

Ricardo Jimenez-Peris, Marta Patiño-Martinez
Universidad Politecnica de Madrid, Madrid, Spain

Synonyms

J2EE; Java EE; JEE

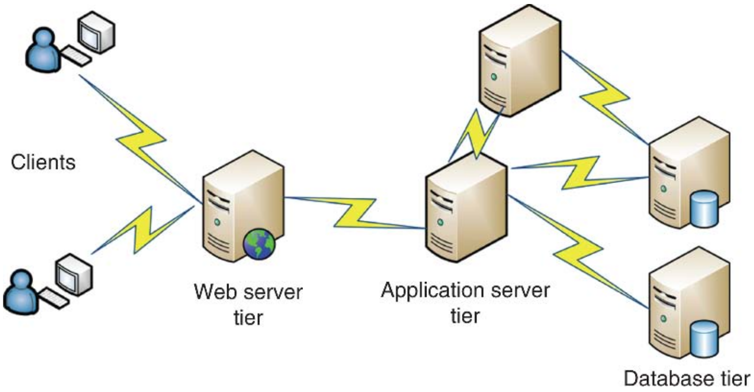
Definition

Java EE (JEE) is a Java Community Process (JCP) specification for Java application servers. JEE consists of a set of related specifications and APIs that shapes an ecosystem for building Java distributed applications over a multi-tier architecture, including web, business logic, and storage tiers. JEE provides components specific for each tier. JEE provides servlets and Java Server Pages (JSPs) for the web tier, Enterprise Java Beans (EJBs) for the business logic tier, and a data driver (JDBC) and an object-relational mapping (via entity beans) for the storage tier. Components can be distributed at different sites and interact with each other. The interaction among distributed JEE components is achieved via RMI (Remote Method Invocation), the JEE specification for remote procedure calls. In its latest editions, JEE has been smoothly integrated with XML processing and web services. It is possible to program web services using EJBs and a large set of APIs enable a standard way to effectively manipulate XML. JEE covers all important non-functional aspects such as transactions, security, etc. Transactions are core to JEE and are considered across all tiers in a consistent fashion through the Java Transaction Service and API (JTS/JTA). A number of specifications address various security issues such as authentication (JAAS, Java Authentication and Authorization Service), encryption, and so on.

Key Points

The JEE project originated back in 1998 when the JPE (the original name) initiative was announced by Sun Microsystems. J2EE 1.2 was released in Dec. 1999. The following versions have been released almost every two years, J2EE 1.3 in Sept. 2001 (1.3 was the first version to be developed as a JCP specification), J2EE 1.4 in Nov. 2003, JEE 1.5 (with version 1.5 J2EE was renamed as JEE) in May 2006, and JEE 1.6 is expected to be released sometime in 2008.

JEE is a multi-tier middleware framework to program all kinds of applications based on the Java language. Multi-tier architectures have become widely used since they allow an adequate separation of concerns. Each tier has a specific container with a component model tailored to the specific mission of the tier. Some of the most common tiers are: web tier, business logic tier, and persistent storage tier.



Java Enterprise Edition. Figure 1. JEE multi-tier architecture.

Figure 1 depicts a typical JEE multi-tier architecture. Clients interact with the system via the web server tier. The web server tier then delegates the business logic to the application server tier. The JEE application server tier can consist of multiple servers, that is, be distributed. Then, the persistent data are kept in the database tier. The data can be kept in multiple distributed databases. The database instances can be shared across application servers. Transactions are used to enforce data consistency across the entire system despite concurrent accesses to the data. Transactions also protect data against server failures providing failure atomicity.

For the web tier, JEE provides a component model for generating dynamic web content. This component model is based on Java Server Pages (JSPs) and servlets. JSPs provide a high level abstraction for dynamic content, whilst servlets provide a lower level view. In fact, JSPs are translated into servlets. Servlet containers are specialized web servers that support servlet execution. Typically servlets deal with the presentation aspects of an application and they are specialized to render dynamic web contents to users. Servlets delegate the business logic to the application server tier, that is, to the JEE application server.

The JEE application server provides a component model for writing general applications. The components in the JEE component model are termed Enterprise Java Beans (EJBs). Two different kinds of EJBs are distinguished: session beans and entity beans. Session beans enable to encapsulate the interaction with users. There are two flavors: stateless and stateful session beans. Stateless session beans provide support for non-conversational interactions, whilst stateful session beans support conversational interactions. Stateless session beans are instantiated per client invocation and their state is discarded after processing the client request that triggered their instantiation. On the other hand, stateful session beans are created upon the first client invocation and then, they are kept during all the client conversation. Their state enables to tracking of the status of the interaction across invocations (e.g., a shopping cart). From JEE 1.5, it is also possible to persist POJOs (Plain Old Java Object, that is, a Java Object) using JPA (Java Persistent API) without the need of a JEE application server.

JEE provides two models of interaction with the database: explicit and implicit. In the explicit model of interaction the EJBs access the database directly via a JDBC driver. The JDBC driver abstracts the database specifics of the underlying implementation providing a uniform vendor-independent interface. However, the object oriented paradigm has a different style than the relational model (the well-known object-relational impedance mismatch) that forces programmers to write a lot of repetitive code prone to errors to create queries and to transform query results into objects. The object-relational mapping (ORM) approach addresses this impedance mismatch by offering the programmer a view of an object oriented database, eliminating the need to write code.

The database tier access can be explicit via JDBC commands or implicit by using the object-relation mapping (ORM) of JEE. Explicit access implies issuing JDBC commands with SQL statements and dealing directly with the results sets returned by JDBC. Implicit access via ORM enables access to the persistent state using the object-oriented paradigm. Tuples are modeled as special kind of JEE components, entity beans, that are accessed as regular objects. Entity beans provide an object-oriented cache of the database. The JEE system takes care of updating the database and implementing the necessary concurrency control to provide transactional serializability.

JEE also provides support for the publish-subscribe paradigm. For that purpose it provides a specific component model and the associated container, Java Message Service (JMS). JMS provides message-driven beans that are components that are activated upon the reception of a particular kind of message. Message-driven beans can be invoked by client applications and EJBs and they can also invoke other EJBs. JMS provides an asynchronous interaction model that complements the synchronous one provided by EJBs and RMI.

In the last few years, a number of initiatives have tried to simplify JEE development. One of the initiatives with highest impact has been the Spring framework [2]. Spring provides a lightweight approach to JEE simplifying the development of JEE applications. Spring is based heavily on programming interfaces (as opposed to programming objects) and the use of aspect oriented programming. Spring has inspired some of the major changes in JEE 1.5 and 1.6.

Cross-references

▶ Replication in Multi-Tier Architectures

Recommended Reading

1. JSR 316: Java^{TM} Platform, Enterprise Edition 6 (Java EE 6) Specification. http://jcp.org/en/jsr/detail?id=316
2. Spring Framework. http://springframework.org/

Java Metadata Facility

David Buttler
Lawrence Livermore National Laboratory, Livermore, CA, USA

Synonyms

Java metadata facility; JSR 175; Java annotations

Definition

The Java Metadata Facility is introduced by Java Specification Request (JSR) 175 [1], and incorporated into the Java language specification [2] in version 1.5 of the language. The specification allows annotations on Java program elements: classes, interfaces, methods, and fields. Annotations give programmers a uniform way to add metadata to program elements that can be used by code checkers, code generators, or other compile-time or runtime components.

Annotations are defined by annotation types. These are defined the same way as interfaces, but with the symbol "@" preceding the "interface" keyword. There are additional restrictions on defining annotation types:

1. They cannot be generic.
2. They cannot extend other annotation types or interfaces.
3. Methods cannot have any parameters.
4. Methods cannot have type parameters.
5. Methods cannot throw exceptions.
6. The return type of methods of an annotation type must be a primitive, a String, a Class, an annotation type, or an array, where the type of the array is restricted to one of the four allowed types.

See [2] for additional restrictions and syntax.

The methods of an annotation type define the elements that may be used to parameterize the annotation in code. Annotation types may have default values for any of its elements. For example, an annotation that specifies a defect report could initialize an element defining the defect outcome to "submitted." Annotations may also have zero elements. This could be used to indicate serializability for a class (as opposed to the current Serializability interface).

Key Points

There are several annotation types that are predefined in the Java 1.5 programming language: "@Override," "@Deprecated," and "@SuppressWarnings" are the most common ones.

"@Override" indicates that a method in a subclass overrides a method from its superclass, as opposed to overloading it. This is an example of an annotation with zero elements. A common, yet difficult to identify, error in writing Java classes occurs when a programmer overloads the `equals` method, rather than overriding it. This leads to errors that are difficult to track down.

"@Deprecated" indicates that a class or method has been deprecated and that programmers should use an alternative. This replaces the javadoc "@deprecated" tag that served the same purpose.

"@SuppressWarnings" indicates that a compiler should not report warnings of a particular type. This particular annotation requires an element, such as "@SupressWarnings('unchecked')," defining the type of warning to ignore for the annotated compilation unit. Warning types are defined by the compiler and are not specified in the Java language specification.

Cross-references

► Metadata

Recommended Reading

1. Coward D. JSR 175: A Metadata Facility for the Java™ Programming Language, 2004. http://jcp.org/en/jsr/detail?id=175
2. Gosling J., Joy B., Steele G., and Bracha G. The Java™ Language Specification. Prentice Hall, NJ, USA, 2005.

JD

► Join Dependency

JDBC

► Java Database Connectivity

Join

Cristina Sirangelo
University of Edinburgh, Edinburgh, Scotland, UK

Definition

The join is a binary operator of the relational algebra that combines tuples of different relations based on a relationship between values of their attributes. The primitive version of the join operator is called *natural join*. Given two relation instances R_1, over set of attributes U_1, and R_2 over set of attributes U_2, the natural join $R_1 \bowtie R_2$ returns a new relation, over set of attributes $U_1 \cup U_2$, consisting of tuples $\{t | t(U_1) \in R_1$ and $t(U_2) \in R_2\}$. Here $t(U)$ denotes the restriction of the tuple t to attributes in the set U.

A derivable version of the join operator is obtained by composing the natural join with the selection operator σ: the *theta-join* $R_1 \bowtie_\theta R_2$ is defined as $\sigma_\theta(R_1 \bowtie R_2)$, where θ is an arbitrary condition allowed in a generalized selection over set of attributes $U_1 \cup U_2$. In the case that θ is a conjunction of equality atoms of the form $A = B$, where A is an attribute in U_1 and B an attribute in U_2, the theta-join is called *equijoin*.

Another derivable join operator is the *semijoin*, denoted by $R_1 \ltimes R_2$; it is defined as $\pi_{U_1}(R_1 \bowtie R_2)$, where π_{U_1} denotes the projection on attributes U_1.

Key Points

In the natural join $R_1 \bowtie R_2$, tuples of R_1 and R_2 having the same values of common attributes are combined. If the sets of attributes of R_1 and R_2 are disjoint, $R_1 \bowtie R_2$ coincides with the cartesian product.

The natural join is often used to combine tuples based on attributes correlated by a foreign key-constraint: consider a relation *Students* over attributes (*student-number, student-name*), containing tuples {(1001, *Black*), (1002, *White*)}, and a relation *Exams* over attributes (*course-number, student-number, grade*), containing tuples {(*EH*1, 1001, *A*), (*EH*1, 1002, *A*), (*GH*5, 1001, *C*)}. Then the natural join *Students* ⋈ *Exams* is a relation over attributes (*student-number, student-name, course-number, grade*) with tuples {(1001, *Black*, *EH*1, *A*), (1001, *Black*, *GH*5, *C*), (1002, *White*, *EH*1, *A*)}.

In the absence of attribute names the only primitive notion of join is the cartesian product. In this case operators of theta-join and equijoin can be derived by composing selection and cartesian product. More precisely, if θ is a boolean combination of atoms of the form $j \alpha k$ with $j \leq arity(R_1)$ and $k \leq arity(R_2)$ and $\alpha \in \{=, \neq, <, >, \leq, \geq\}$, then the theta-join $R_1 \bowtie_\theta R_2$ in the unnamed algebra is defined as $\sigma_{\theta'}(R_1 \times R_2)$, where θ' is obtained from θ by replacing each atom $j \alpha k$ with $j \alpha (arity(R_1) + k)$.

In each of the join operators described above (except the semijoin) there can be tuples of the input relations which do not occur in the output, because they satisfy the join condition with no tuple of the other relation. The *left (right) outer join* adds to the join of R_1 and R_2 all tuples of R_1 (R_2) not occurring in the join, completed with nulls on attributes of R_2 (R_1). The *full outer join* adds both tuples of R_1 and R_2 to the join.

Cross-references

▶ Cartesian Product
▶ Foreign Key
▶ Projection
▶ Relation
▶ Relational Algebra
▶ Selection

Join Dependency

Solmaz Kolahi
University of British Columbia, Vancouver, BC, Canada

Synonyms

JD

Definition

A *join dependency (JD)* over a relation schema $R[U]$ is an expression of the form $\bowtie [X_1,...,X_n]$, where $X_1 \cup \ldots \cup X_n = U$. An instance I of $R[U]$ satisfies $\bowtie [X_1,...,X_n]$ if $I = \pi_{X_1}(I) \bowtie \ldots \bowtie \pi_{X_n}(I)$. In other words, an instance satisfies the join dependency if it is equal to the join of its projections on the sets of attributes $X_1,...,X_n$. A multivalued dependency $X \twoheadrightarrow Y$ is a special case of a join dependency on two sets, and can be expressed as $\bowtie [XY, X(U - XY)]$, where XY represents $X \cup Y$.

Key Points

Join dependencies are particularly important in connection with the decomposition technique for schema design and normalization. The main goal of the decomposition technique is to avoid redundancies due to data dependencies by decomposing a relation into smaller parts. A good decomposition should have the *lossless join* property, meaning that no information should be lost after the decomposition. In other words, the original database instance should be retrievable by joining the smaller relations, and this can be expressed by a JD. The following figure shows an instance of the relation schema $R[A, B, C, D]$ that satisfies the join dependency $\bowtie [BC, AB, AD]$:

$$\begin{array}{|cccc|}\hline A & B & C & D \\ \hline 1 & 2 & 3 & 4 \\ 1 & 5 & 6 & 4 \\ 4 & 2 & 3 & 5 \\ 3 & 2 & 3 & 6 \\ \hline \end{array} = \begin{array}{|cc|}\hline B & C \\ \hline 2 & 3 \\ 5 & 6 \\ \hline \end{array} \bowtie \begin{array}{|cc|}\hline A & B \\ \hline 1 & 2 \\ 1 & 5 \\ 4 & 2 \\ 3 & 2 \\ \hline \end{array} \bowtie \begin{array}{|cc|}\hline A & D \\ \hline 1 & 4 \\ 4 & 5 \\ 3 & 6 \\ \hline \end{array}$$

Join dependencies are usually considered together with functional and multivalued dependencies (FDs and MVDs) in normalization. The implication problem of a JD from a set of JDs, MVDs, and FDs is known to be NP-hard. In addition, the implication problem of JDs cannot be axiomatized. That is, there is no sound and complete set of rules that can be used to check whether a dependency is implied by a set of JDs. However, there is a powerful tool, called *chase*, that could be used to reason about these dependencies in exponential time and space [1].

Cross-references

- ▶ Functional Dependency
- ▶ Join
- ▶ Multivalued Dependency
- ▶ Normal Forms and Normalization
- ▶ Projection

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases. Addison-Wesley, Reading, MA, 1995.

Join Index

Theodore Johnson
AT&T Labs – Research, Florham Park, NJ, USA

Definition

A join index is a collection of pairs $\{(r, s)\}$ such that the record in table R with record ID (RID) r joins with the record in table S with RID s, according to the *join* predicate which defines the index.

Key Points

The purpose of a join index is to accelerate common joins, even equijoins. One of the advantages of join indices is that they can be represented in a very compact way, allowing for highly efficient access. For example, suppose that the DBMS is to evaluate a query, "Select R.a from R,S where R.a = S.b." A conventional join would use a nested loop algorithm, with an indexed scan in the inner loop. With a join index, the join can be computed by scanning the join index, thus minimizing random I/O.

There are a variety of ways of implementing a join index. One can list pairs (clustered on R or clustered on S), or in the case of an equijoin, associate R and S RIDs with attribute values. The example below shows the join index for R.a = S.b, organized as a list of pairs.

	R.a
0	12
1	8
2	14
3	7
4	8
5	12
6	1
7	14

	S.b
0	8
1	2
2	3
3	8
4	14
5	9
6	5
7	12
8	11
9	14

Join Index
(0, 7)
(1, 0)
(1, 3)
(2, 4)
(2, 9)
(2, 9)
(4, 0)
(4, 3)
(5, 7)
(7, 4)
(7, 9)

Cross-references

- ▶ Join
- ▶ Star Index

Recommended Reading

1. Li Z. and Ross K.A. Fast joins using join indices. VLDB J., 8(1):1–24, 1999.
2. Valduriez P. Join indices. ACM Trans. Database Syst., 12(2):218–246, 1987.

Join Indices

- ▶ Star Index

Join Order

Jingren Zhou
Microsoft Research, Redmond, WA, USA

Synonyms

Join order; Join sequence

Definition

A database query typically contains multiple joins. When joins (for example, inner joins) are commutative and/or associative, there can more than one evaluation order for joins. The join order has an enormous impact on the query cost. One of the main responsibilities of the query optimizer is to determine the optimal join order for query evaluation.

Key Points

Choosing a good join order is very important to achieve a good query performance. One important consideration for choosing an join order is to reduce the size of intermediate results as much as possible. For example, it is beneficial to first evaluate a join that returns the least result. Other considerations include join methods, data properties, and access methods, etc.

Depending on the choice of join orders, query plans can be of different shapes.

- *Left-Deep* query plans use a base table as the inner table for each join.
- *Right-Deep* query plans use a base table as the outer table for each join.
- *Bushy* query plans use the intermediate result from other joins as join inputs.

Query optimizers typically use dynamic programming and heuristics to determine join orders.

Cross-references

▶ Parallel Join Algorithms
▶ Query Optimization
▶ Evaluation of Relational Operators

Recommended Reading

1. Mishra P. and Eich M.H. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, 1992.
2. Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., and Price T.G. Access path selection in a Relational Database Management System. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.

Join Processing

▶ Distributed Join

Join Sequence

▶ Join Order

JSR 175

▶ Java Metadata Facility

K

k-Anonymity

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona,
Catalonia

Synonyms

p-Sensitive *k*-Anonymity

Definition

A protected dataset is said to satisfy *k*-anonymity for $k > 1$ if, for each combination of key attribute values (e.g., address, age, gender, etc.), at least *k* records exist in the dataset sharing that combination [2,3].

Key Points

If, for a given *k*, *k*-anonymity is assumed to be sufficient protection, one can concentrate on minimizing information loss with the only constraint that *k*-anonymity should be satisfied. This is a clean way of solving the tension between data protection and data utility. Since *k*-anonymity is usually achieved via generalization (equivalent to global recoding, as said above) and local suppression, minimizing information loss usually translates to reducing the number and/or the magnitude of suppressions.

k-Anonymity bears some resemblance to the underlying principle of microaggregation and is a useful concept because quasi-identifiers are usually categorical or can be categorized, i.e., they take values in a finite (and ideally reduced) range. However, re-identification is not necessarily based on categorical key attributes: sometimes, numerical outcome attributes (which are continuous and often cannot be categorized) give enough clues for re-identification. Microaggregation was suggested as a possible way to achieve *k*-anonymity for numerical, ordinal and nominal attributes [1].

p-Sensitive *k*-anonymity is a stronger property whereby it is required that a dataset is *k*-anonymous and additionally that there are at least *p* distinct values for each confidential attribute within a group of records sharing a combination of key attributes [4].

Cross-references

▶ Global Recoding
▶ Inference Control in Statistical Databases
▶ Local Suppression
▶ Microaggregation
▶ Microdata

Recommended Reading

1. Domingo-Ferrer J. and Torra V. Ordinal, continuous and heterogenerous *k*-anonymity through microaggregation. Data Mining Knowl. Discov., 11(2):195–212, 2005.
2. Samarati P. Protecting respondents' identities in microdata release. IEEE Trans. Knowl. Data Eng., 13(6):1010–1027, 2001.
3. Samarati P. and Sweeney L. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, SRI International, 1998.
4. Truta T.M. and Vinay B. Privacy protection: *p*-sensitive *k*-anonymity property. In Proc. 2nd Int. Workshop on Privacy Data Management, 2006, p. 94.

k-Closest Pair Join

▶ Closest-Pair Query

k-Closest Pair Query

▶ Closest-Pair Query

KDD Pipeline

Hans-Peter Kriegel, Matthias Schubert
Ludwig-Maximilians-University, Munich, Germany

Synonyms

KDD process; Data mining process; Data mining pipeline

Definition

The KDD pipeline describes the complete process of knowledge discovery in databases (KDD), i.e., the process of deriving useful, valid and non-trivial patterns from a large amount of data. The pipeline consists of five consecutive steps:

Selection

The selection step identifies the goal of the current application and selects a data set that is likely to contain relevant patterns.

Preprocessing

The preprocessing step increases the quality of the data set by supplementing missing attributes, removing duplicate instances and resolving data inconsistencies.

Transformation

The transformation step deletes correlated and irrelevant attributes and derives new more meaningful attributes from the current data description.

Data Mining

This step selects a data mining algorithm with respect to the goal which was identified in the selection step and derives patterns or learns functions that are valid for the current data set.

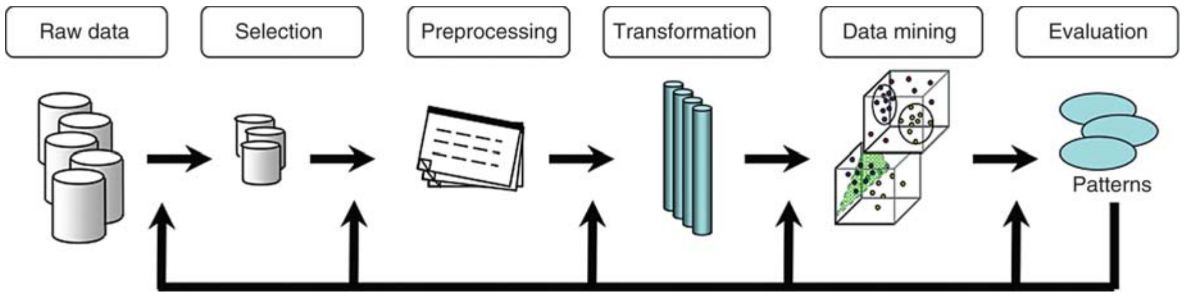
Evaluation and Interpretation

In the last step, the found patterns are checked with respect to their validity. Furthermore, the user examines the usefulness of the found knowledge for the given application.

The quality of the found patterns depends on the methods being employed in each of these steps. Thus, the pipeline is usually repeated after adjusting the parametrization or even exchanging the methods in any of these steps until the quality of the results is sufficient. Figure 1 illustrates the complete KDD pipeline.

Key Points

The KDD pipeline was originally introduced as KDD process [1–3]. It describes a unifying framework containing all necessary and optional steps when deriving patterns using data mining algorithms. An important aspect of this view is that data mining is just one step in the complete KDD process, which emphasizes the importance of a meaningful and consistent data representation. The KDD pipeline is considered to be an interactive and adjustable framework rather than a strict work flow. The necessity for this flexibility arises from the large variety of methods and parameter selections that can be applied in each step. In the majority of cases, it is necessary to adjust parameters or even exchange the applied method in one of the steps if the final patterns do not display satisfactory quality in the evaluation step. Furthermore, the borders of each step cannot be outlined in a strict manner because the quality of results strongly depends on a well selected combination of the methods applied in each step. Additionally, there exist methods fulfilling the tasks of two consecutive steps, e.g., transformation and data mining.



KDD Pipeline. Figure 1. Schema of the KDD pipeline.

Cross-references

- ▶ Data Mining
- ▶ Knowledge Discovery
- ▶ OLAP

Recommended Reading

1. Brachman R. and Anand T. The process of knowledge discovery in databases: a human centered approach. In Proc. 10th National Conf. on AI, 1996, pp. 37–38.
2. Fayyad U., Piatetsky-Shapiro G., and Smyth P. From data mining to knowledge discovery in databases. In Proc. 10th National Conf. on AI, 1996, pp. 1–30.
3. Fayyad U., Piatetsky-Shapiro G., and Smyth P. Knowledge discovery and data mining: towards a unifying framework. In Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, 1996, pp. 82–88.

KDD Process

▶ KDD Pipeline

k-Distance Join

▶ Closest-Pair Query

Key

David W. Embley
Brigham Young University, Provo, UT, USA

Synonyms

Uniqueness constraint

Definition

In the relational model, a *key* for a relational schema is a set of attributes whose value(s) uniquely identify a tuple in a valid instance of the relation. Said another way, key value(s) appear at most once in a relation. Often the set of attributes constituting a key is a set with a single attribute. For example, in the relation the singleton set consisting of just *CustomerID* is a key and so is the set consisting of the pair of attributes (*Name*, *Address*). The values of *CustomerID* (here, 11111 and 22222) each uniquely identify a tuple. In any valid instance for this relation, no *CustomerID*-value may appear twice. Similarly, the pair (*Name*, *Address*) is a key. Pairs of (*Name*, *Address*) values (e.g., <*Pat*, *12 Maple* >) uniquely identify a tuple, and no value pair may appear twice.

Customer	(CustomerID	Name	Address)
	11111	Pat	12 Maple
	22222	Tracy	44 Elm

Designating a set of attributes as a key is a property of a relational schema, not a property of a relation instance. It may just happen that for some valid relation instances a value uniquely identifies a tuple, but a key must always uniquely identify a tuple for *all* valid relations. In this example, *Pat* and *Tracy* uniquely identify tuples, but it would also be valid for another *Tracy*, say, one living at *905 Lincoln Ave.*, to be a customer. Although *Name* uniquely identifies tuples in this *Customer* table instance, *Name* does not uniquely identify tuples in *all* valid *Customer* table instances and thus is not a key for the *Customer* relational schema.

Key Points

The term *key* is often ambiguous and may mean *minimal key*, *superkey*, *primary key*, or *candidate key*. A *minimal key* is a key that does not have superfluous attributes. Clearly, in the sample relational schema above, *CustomerID* together with *Name* uniquely identify a customer because *CustomerID* alone is enough. *Name* is superfluous, and thus the set consisting of the pair of attributes (*CustomerID*, *Name*) is not a minimal key. The set consisting of the pair of attributes (*Name*, *Address*), however, is a minimal key. Both are necessary – in the table above, for example, both <*Tracy*, *905 Lincoln Ave.* > and <*Lynn*, *12 Maple* > could appear so that neither *Name* alone nor *Address* alone would uniquely identify a tuple. Usually only minimal keys are of interest. Keys that may not be minimal are called *superkeys*. A *superkey* is any set of attributes, minimal or not, that uniquely identifies a tuple in any valid relation instance. A relational schema may have several minimal keys. The sample *Customer* relational schema above, for example, has two minimal keys – *CustomerID* and the pair (*Name*, *Address*). When there are several minimal keys, one is chosen as a *primary key*. When there is only one, it is also called a *primary key*. Because there can be a choice among minimal keys, they are often each referred to as *candidate keys*.

Keys and functional dependencies are strongly related. Indeed, keys can be defined in terms of functional dependencies. Let U be a set of attributes, and let F be a set of functional dependencies over U. Let R, a subset of U, be a relational schema. A subset K of R is a *superkey* of R if $K \rightarrow R \in F^+$, where F^+ includes F and any functional dependency implied by F. A subset K of R is a *candidate key* of R (also called a *minimal key* of R) if K is a superkey and there does not exist a proper subset K' of K such that $K' \rightarrow R \in F^+$.

Database normalization is largely about aligning keys and functional dependencies. A relational schema R is in Boyce-Codd Normal Form (BCNF) if for every non-trivial functional dependency $X \rightarrow Y$ (given or implied) that applies to R (satisfies $XY \subseteq R$), X is a superkey of R. More informally, the idea of aligning keys and functional dependencies to yield BCNF is intuitively captured by making every attribute in the relational schema depend functionally on a minimal key, the whole minimal key, and nothing but the whole minimal key.

Cross-references

▶ Functional Dependency

▶ Normal Forms and Normalization

▶ Relational Model

Key Range Locking

▶ B-Tree Locking

Key Value Locking

▶ B-Tree Locking

KL-ONE Style Languages

▶ Description Logics

K-Means and K-Medoids

XUE LI
The University of Queensland, Brisbane, QLD, Australia

Synonyms

K-means partition; PAM (Partitioning Around Medoids); CLARA (Clustering LARge Applications); CLARANS (Clustering large applications based upon randomized search)

Definitions

K-means

Given an integer k and a set of objects $S = \{p_1, p_2, ..., p_n\}$ in Euclidian space, the problem of k-means clustering is to find a set of centre points (means) P = $\{c_1, c_2, ..., c_k\}$, $|P| = k$ in the space, such that S can be partitioned into k corresponding clusters $C_1, C_2, ..., C_k$, by assigning each object in S to the closest centre c_i. The sum of square error criterion (SEC) measure, defined as $\sum_{i=1}^{k} \sum_{p \in C_i} |p - c_i|^2$, is minimized.

K-medoids

Given an integer k and a set of objects $S = \{p_1, p_2, ..., p_n\}$ in Euclidian space, the problem of k-medoids clustering is to find a set of objects as medoids P = $\{o_1, o_2, ..., o_k\}$, $|P| = k$ in the space, such that S can be partitioned into k corresponding clusters $C_1, C_2, ..., C_k$, by assigning each object in S to the closest medoid o_i. The sum of square error criterion (SEC) measure, defined as $\sum_{i=1}^{k} \sum_{p \in C_j} |p - o_j|^2$, is minimized.

Key Points

K-means

The k-means algorithm starts with a random selection of k objects as the centres or the means of k clusters. The challenge is to place these centres in a clever way because each different location of centres may result in different clustering. Hence, a reasonable initial choice is to locate them as far away from each other as possible. Then an iteration process is used to assign remaining objects to their nearest centres. After the membership of all clusters is considered according to the given initial centres, the cluster centres will all be reconsidered. Hence the new cluster centres are calculated a set of k means $P = \{c_1, c_2, \ldots, c_k\}$. For example, for the mean of a set of single-valued objects in cluster i with m points is defined as m_i:

$$m_i = \frac{1}{m} \sum_{j=1}^{m} p_{i,j}$$

The means of high-dimensional objects are calculated in a same by using this formula for each of the dimensions. When cluster centres are updated, the membership computations will be executed again to reallocate the objects to the nearest centers. The main idea of updating the cluster centres and the memberships is based on the computation of Euclidian distances such that the objects in a cluster have the minimum distances and the objects between clusters have the maximum distances. The process will continue until there are no more updates that could reallocate the centres and the memberships of clusters. Even though it is easy to prove that the procedure always terminates, the k-means algorithm does not necessarily find the global optimal solutions, corresponding to the global minimum of the objective function. Also, it is easy to demonstrate that this algorithm is significantly sensitive to the initial randomly selected centres. To mitigate this situation the k-means algorithm can be executed multiple times in order to identify a better global solution. The complexity of the algorithm is $O(nkt)$, for n objects, k clusters and t iterations.

The k-means algorithm is only applicable to the objects with mean defined. The user must specify an initial value k. The k-means approach is only suitable for finding convex shapes with all clusters having similar sizes. Moreover, when there is noise and outliers in the dataset, the means of clusters may appear at locations where the majority objects are far away.

K-medoids

K-medoids algorithm avoids calculating means of clusters in which extremely large values may affect the membership computations substantially. K-medoids can handle outliers well by selecting the most centrally located object in a cluster as a reference point, namely, medoid. The difference between k-means and k-medoids is analogous to the difference between mean and median: where mean indicates the average value of all data items collected, while median indicates the value around that which all data items are evenly distributed around it. The basic idea of k-medoids is that it first arbitrarily finds k objects amongst n objects in the dataset as the initial medoids. Then the remaining objects are partitioned into k clusters by computing the minimum Euclidian distances that can be maintained for the members in each of the clusters. An iterative process then starts to consider objects p_i, $i = 1,\ldots,n$, if a medoid o_j, $j = 1,\ldots,k$, can be replaced by a candidate object o_c, $c = 1,\ldots,n$, $c \neq i$. There are four situations to be considered in this process: (i) *Shift-out membership*: an object p_i may need to be shifted from currently considered cluster of o_j to another cluster; (ii) *Update the current medoid*: a new medoid o_c is found to replace the current medoid o_j; (iii) *No change*: objects in the current cluster result have the same or even smaller SEC for all the possible redistributions considered; (iv) *Shift-in membership*: an outside object p_i is assigned to the current cluster with the new (replaced) medoid o_c.

K-medoids algorithm needs to test if any existing medoids can be replaced by any other objects. By looking at all of these possible replacements, if the overall SEC is improved then the given medoid will be replaced. The computational cost of k-medoid is much higher than the k-means. For each iteration, computational cost is $k(n\text{-}k)^2$ for $k(n\text{-}k)$ pairs of objects to be considered with each $(n\text{-}k)$ evaluations of the SEC. K-meoids algorithm then is considered with a sampling method for the improvement on the computational complexity. In this way, several samples are taken from the dataset, and then the k-medoids algorithm is applied to each of the samples. The convergence of medoids is achieved by choosing the sample that performs the best.

Cross-references

▶ Density Based Clustering
▶ Hierarchical Clustering

Recommended Reading

1. Kaufman L. and Rousseeuw P.J. Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley, NY, 1990.
2. MacQueen J. Some methods for classification and analysis of multivariate observations. In Proc. 5th Berkeley Symposium on Mathematics, Statistics and Probabilities, vol. 1, 1967, pp. 281–297.
3. Ng R.T. and Han J. Efficient and effective clustering methods for spatial data mining. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 144–155.

K-Means Partition

▶ K-Means and K-Medoids

k-Nearest Neighbor Classification

▶ Nearest Neighbor Classification

k-NN Classification

▶ Nearest Neighbor Classification

kNN Query

▶ Nearest Neighbor Query

Knowledge Creation

▶ Ontology Elicitation

Knowledge Discovery from Biological Resources

▶ Text Mining of Biological Resources

Knowledge Discovery from Data

▶ Data Mining

Knowledge Discovery in Streams

▶ Classification on Streams

Knowledge Discovery in Text (KDT)

▶ Text Mining

Knowledge Management

▶ Ontologies and Life Science Data Management

Knowledge Organization Systems

▶ Gazetteers

Knowledge-based Systems

▶ Executable Knowledge

Koch Snowflake

▶ Fractal

L

L1 Cache

▶ Processor Cache

L2 Cache

▶ Processor Cache

L3 Cache

▶ Processor Cache

Language Models

Djoerd Hiemstra
University of Twente, Enschede, The Netherlands

Synonyms

Generative models

Definition

A language model assigns a probability to a piece of unseen text, based on some training data. For example, a language model based on a big English newspaper archive is expected to assign a higher probability to "a bit of text" than to "aw pit tov tags," because the words in the former phrase (or word pairs or word triples if so-called *N-Gram Models* are used) occur more frequently in the data than the words in the latter phrase. For information retrieval, typical usage is to build a language model for each document. At search time, the top ranked document is the one whose language model assigns the highest probability to the query.

Historical Background

The term *language models* originates from probabilistic models of language generation developed for automatic speech recognition systems in the early 1980s [9]. Speech recognition systems use a language model to complement the results of the *acoustic model* which models the relation between words (or parts of words called phonemes) and the acoustic signal. The history of language models, however, goes back to the beginning of the twentieth century when Andrei Markov used language models (Markov models) to model letter sequences in works of Russian literature [3]. Another famous application of language models are Claude Shannon's models of letter sequences and word sequences, which he used to illustrate the implications of coding and information theory [17]. In the 1990s, language models were applied as a general tool for several natural language processing applications, such as part-of-speech tagging, machine translation, and optical character recognition. Language models were applied to information retrieval by a number of research groups in the late 1990s [4,7,14,15]. They became rapidly popular in information retrieval research. By 2001, the ACM SIGIR conference had two separate sessions on language models containing five papers in total [12]. In 2003, a group of leading information retrieval researchers published a research roadmap "challenges in information retrieval and language modeling" [1], indicating that the future of information retrieval and the future of language modeling can not be seen separate from each other.

Foundations

Language models are generative models, i.e., models that define a probability mechanism for generating language. Such generative models might be explained by the following probability mechanism: Imagine picking a term T at random from this page by pointing at the page with closed eyes. This mechanism defines a probability $P(T|D)$, which could be defined as the relative frequency of the occurrence of the event, i.e., by the number of occurrences of a term on the page divided by the total number of terms on the page. Suppose the process is repeated n times, picking one at a time the terms T_1, T_2,...,T_n. Then, assuming independence between the successive events, the probability of the terms given the document D is defined as follows:

$$P(T_1, T_2, \ldots, T_n|D) = \prod_{i=1}^{n} P(T_i|D) \tag{1}$$

A simple language modeling approach would compute (1) for each document in the collection, and rank the documents accordingly. A potential problem might be the following: The equation will assign zero probability to a sequence of terms unless all terms occur in the document. So, a language modeling system that uses (1) will not retrieve a document unless it contains all query terms. This might be reasonable for a web search engine that typically processes small queries to search a vast amount of data, but for many other information retrieval applications, this behavior is a problem. A standard solution is to use *linear interpolation smoothing* of the document model $P(T|D)$ with a collection model $P(T|C)$, which is defined as follows:

$$P(T_1, T_2, \ldots, T_n|D) = \prod_{i=1}^{n} \Big(\lambda P(T_i|D) + (1-\lambda) P(T_i|C)\Big) \tag{2}$$

This way, a term that does not occur in the document will not be assigned zero probability but instead a probability proportional to its number of occurrences in the entire collection C. Here, λ is an unknown probability that should be tuned to optimize retrieval effectiveness. Linear interpolation smoothing was used in several early language modeling approaches [7,14].

Implementation

Although the language modeling equations above suggest the need to compute probabilities for all documents in the collection, this is unnecessary in practice. In fact, most language modeling approaches can be implemented efficiently by the use of standard inverted index search systems. This can be seen by the equation below which can be derived from (2) by two basic transformations: First, dividing it by the probability of the collection model; and second, taking the logarithm.

$$P(T_1, T_2, \ldots, T_n|D) \propto \sum_{i=1}^{n} \log\left(1 + \frac{\lambda P(T_i|D)}{(1-\lambda) P(T_i|C)}\right) \tag{3}$$

Equation (3) no longer produces probabilities, but it ranks the documents in the exact same order as (2), because the collection model does not depend on the document, and the logarithm is a strictly monotonic function. Taking the logarithm prevents the implementation from running out of the precision of its (floating point) representation of probabilities, which can become very small because the probabilities are multiplied for every query term. Similar to for instance vector space models in information retrieval, ranking is defined by a simple sum of term weights, for which terms that do not match a document get a zero weight. Interestingly, the resulting "term weight" can be seen as a variant of *tf.idf* weights, which are often used in vector space models.

Document Priors

The equations above define the probability of a query given a document, but obviously, the system should rank by the probability of the documents given the query. These two probabilities are related by Bayes' rule as follows.

$$P(D|T_1, T_2, \ldots, T_n) = \frac{P(T_1, T_2, \ldots, T_n|D) P(D)}{P(T_1, T_2, \ldots, T_n)} \tag{4}$$

The left-hand side of (4) cannot be used directly because the independence assumption presented above assumes term independence given the document. So, in order to compute the probability of the document D given the query, (2) needs to be multiplied by $P(D)$ and divided by $P(T_1, \ldots, T_n)$. Again, as stated earlier, the probabilities themselves are of no interest, but the ranking of the document by the probabilities is. And since $P(T_1, \ldots, T_n)$ does not depend on the document, ranking the documents by the numerator of the right-hand side of (4) will rank them by the probability given the query. This shows the importance of $P(D)$: The marginal probability, or *prior probability* of the document, i.e., it is the probability that the document is relevant if the query is ignored. For instance, it might be assumed that long documents are more likely to be useful than short documents [5,6]. In web search, such a so-called static ranking (a ranking that is independent of the query) is commonly used. For instance, documents with many links pointing to them are more likely to be relevant, or documents with short URLs are more likely to be relevant. The prior probability of a document is a powerful way to incorporate static ranking in the language modeling approach [10].

Document Generation Models

An implicit assumption of the language models presented is that there is more information available about the documents than about the query. In some applications, however, the situation is reversed. For instance in *topic tracking*, a system has the task of tracking a stream of chronologically ordered stories. For each story in the stream, the system has to decide whether it is on topic. The target topic is usually based on a number of example stories on a certain topic, there is more information available about the topic than about a single story. Unlike query generation models, document generation models need some form of normalization because documents will have different lengths. The probability of generating a document tends to be smaller for long documents than for short documents. Therefore, several normalization techniques might be applied, such as normalization by document length and additional Gaussian normalization [10,18]. *Relevance feedback* (i.e, the user marked some documents as relevant) is another situation in which there is more knowledge available about the query than about each single document. If some relevant documents are known, or if the top ranked documents are assumed to be relevant, then those documents might be used to generate a new, improved query [20]. As an example, consider the following so-called *relevance models* approach [13]

$$P(Q|T_1,\ldots,T_n) \propto \sum_d \left(P(D=d)P(Q|D=d)\prod_{i=1}^{n} P(T_i=t_i|D=d) \right) \tag{5}$$

Here, the formula defines the probability of a new word Q, given the original query $T_1,\ldots,T_n$ by marginalizing over all documents. In practice, only the top ranked documents for the query $T_1,\ldots,T_n$ are used. Interestingly, the relevance model might be used to infer other information from the top ranked documents, for instance the person that is most often mentioned for a certain query, so-called *expert search* [2].

Translation Models

Language models for information retrieval are generative models, and therefore easily combined with other generative models. To add a model of term translation, the following probability mechanism applies: Imagine picking an English term T at random from this page by pointing at the page with closed eyes (which defines a probability $P(T|D)$), and then translate the term T by picking from the term's entry in a English–Dutch dictionary at random a Dutch term S (with probability $P(S|T)$). The model might be used in a cross-language retrieval system to rank English documents given a Dutch query $S_1,\ldots,S_n$ by the following probability [4,6,13,19]:

$$P(S_1, S_2,\ldots,S_n|D) = \prod_{i=1}^{n}\sum_t \Big(P(S_i=s_i|T_i=t) \\ (\lambda P(T_i=t|D) + (1-\lambda)P(T_i=t|C)) \Big) \tag{6}$$

Here, Dutch is the source language and English the target language. The formula uses linear interpolation smoothing of the document model with the target language background model $P(T|C)$ (English in the example) at the right-hand side of the formula. In some formulations, the translation model is smoothed with the source language background model $P(S|C)$ which a estimated on auxiliary data. The two background models are related as follows: $P(S|C) = \sum_t P(S|T=t)P(T=t|C)$. The translation probabilities are often estimated from parallel corpora, i.e., from texts in the target language and its translations in the source language [6,19]. Translation models might also be used in a monolingual setting to account for synonyms and other related words [4].

Aspect Models

In *aspect models*, also called *probabilistic latent semantic indexing* models, documents are modeled as mixtures of aspect language models. In terms of a generative model it can be defined in the following way [8]: (i) select a document D with probability $P(D)$, (ii) pick a latent aspect Z with probability $P(Z|D)$, (iii) generate a term $T$with probability $P(T|Z)$ independent of the document, (iv) repeat Step 2 and Step 3 until the desired number of terms is reached. This leads to (7).

$$P(T_1,T_2,\ldots,T_n|D) \propto \prod_{i=1}^{n}\left(\sum_z (P(T_i|Z=z)P(Z=z|D)) \right) \tag{7}$$

The aspects might correspond with the topics or categories of documents in the collection such as "health", "family", "Hollywood", etc. The aspect Z

is a hidden, unobserved variable, so probabilities concerning Z cannot be estimated from direct observations. Instead, the expectation maximization (EM) algorithm can be applied [9]. The algorithm starts out with a random initialization of the probabilities, and then iteratively re-estimates the probability of arriving at a local maximum of the likelihood function. It has been shown that the EM algorithm is sensitive to the initialization, and an unlucky initialization results in a non-optimal local maximum. As a solution, clustering of documents has been proposed to initialize the models [16]. Another alternative is latent semantic Dirichlet allocation [15] which has less free parameters, and therefore is less sensitive to the initialization.

Key Applications

This entry focuses on the application of language models to information retrieval. The applications presented include newswire and newspaper search [4,5,15], web search [11], cross-language search [6,19], topic detection and tracking [10,18], and expert search [2]. However, language models have been used in virtually every application that needs processing of natural language texts, including automatic speech recognition, part-of-speech tagging, machine translation, and optical character recognition.

Cross-references

▶ N-Gram Models
▶ Probability Smoothing

Recommended Reading

1. Allan J., Aslam J., Belkin N., Buckley C., Callan J., Croft B., Dumais S., Fuhr N., Harman D., Harper D.J., Hiemstra D., Hofmann T., Hovy E., Kraaij W., Lafferty J., Lavrenko V., Lewis D., Liddy L., Manmatha R., McCallum A., Ponte J., Prager J., Radev D., Resnik P., Robertson S., Rosenfeld R, Roukos S., Sanderson M., Schwartz R., Singhal A., Smeaton A., Turtle H., Voorhees E., Weischedel E., Xu J., and Zhai C.X. (eds.). Challenges in information retrieval and language modeling. SIGIR Forum 37(1), 2003.
2. Balog K., Azzopardi L., and Rijke M. Formal models for expert finding in enterprise corpora. In Proc. 29th Annu. Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 43–50.
3. Basharin G.P., Langville A.N., and Naumov V.A. The life and work of A.A. Markov. linear Algebra and its Applications, 386:3–26, 2004.
4. Berger A. and Lafferty J. Information retrieval as statistical translation. In Proc. 22nd ACM Conf. on Research and Development in Information Retrieval, 1999, pp. 222–229.
5. Blei D.M., Ng A.Y., and Jordan M.I. Latent Dirichlet allocation. J. Machine Learn. Res. 3(5):993–1022, 2003.
6. Hiemstra D. and Jong F. Disambiguation strategies for cross-language information retrieval. Lecture Notes in Computer Science, Volume 1696: In Proceedings of the European Conference on Digital Libraries, Springe-Verlag, Berlin Heidelberg New York, 1999, pp. 274–293.
7. Hiemstra D. and Kraaij W. Twenty-One at TREC-7: Ad-hoc and cross-language track. In Proc. 7th Text Retrieval Conference TREC-7. NIST Special Publication 500-242, 1998, pp. 227–238.
8. Hofmann T. Probabilistic latent semantic indexing. In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 50–57.
9. Jelinek F. Statistical Methods for Speech Recognition. MIT Press, Cambridge, MA, 1997.
10. Jin H., Schwartz R., Sista S., and Walls F. Topic tracking for radio, TV broadcast and newswire. In Proc. DARPA Broadcast News Workshop, 1999.
11. Kraaij W., Westerveld T., and Hiemstra D. The importance of prior probabilities for entry page search. In Proc. 25th ACM Conf. on Research and Development in Information Retrieval (SIGIR'02), 2002, pp. 27–34.
12. Kraft D.H., Bruce Croft W., Harper D.J., and Zobel J. (eds.). In Proc. 24th ACM Conf. on Research and Development in Information Retrieval. Association for Computing Machinery, 2001.
13. Lavrenko V. and Croft W.B. Relevance models in information retrieval. In Language Modeling for Information Retrieval, W. Bruce Croft and John Lafferty (eds.). Kluwer, Dordecht, 2003, pp. 11–56.
14. Miller D.R.H., Leek T., and Schwartz R.M. A hidden Markov model information retrieval system. In Proc. 22nd ACM Conf. Research and Development in Information Retrieval, 1999, pp. 214–221.
15. Ponte J.M. and Bruce C. W. A language modeling approach to information retrieval. In Proc. 21st ACM Conf. Research and Development in Information Retrieval, 1998, pp. 275–281.
16. Schwartz R.M., Sista S., and Leek T. Unsupervised topic discovery. In Proc. Language Models for Information Retrieval Workshop, 2001.
17. Shannon C.E. A mathematical theory of communication. Bell Syst. Tech. J., 27:379–423, 623–656, 1948.
18. Spitters M. and Kraaij W. Language models for topic tracking. In Language Modeling for Information Retrieval, W. Bruce Croft and J. Lafferty (eds.). Kluwer, Dordecht, 2003, pp. 95–124.
19. Xu J. and Weischedel R. A probabilistic approach to term translation for cross-lingual retrieval. In Language Modeling for Information Retrieval, W. Bruce Croft and John Lafferty (eds.). Kluwer, Dordecht, 2003, pp. 125–140.
20. Zhai C. and Lafferty J. Model-based feedback in the language modeling approach to information retrieval. In Proc. ACM Int. Conf. on Information and Knowledge Management, 2001, pp. 403–410.

Languages for Web Data Extraction

Nicholas Kushmerick
Decho Corporation, Seattle, WA, USA

Synonyms

Web scraping; Screen scraping; Web site wrappers; Web mining; Information extraction

Definition

Web data extraction is the process of automatically converting Web resources into a specific structured format. For example, if a collection of HTML web pages describes details about various companies (name, headquarters, etc) then web data extraction would involve converting this native HTML format into computer-processable data structures, such as entries in relational database tables. The purpose of web data extraction is to make web data available for subsequent manipulation or integration steps. In the previous example, the goal may be summarizing the results as some form of analytical report.

There are several approaches to Web data extraction. The most common approach is to specify the conversion process using a special-purpose programming *Language for Web Data Extraction*. Web data extraction then becomes a matter of executing a well-defined computer program.

Web data extraction languages are generally not intended to be general-purpose programming languages (although some are formally Turing complete). Rather, these special-purpose languages are intended to simplify various actions that are needed to navigate to and retrieve web documents, extract and clean their data, and populate the data into appropriate data structures for output. For example, most languages for web data extraction provide mechanisms for crawling hypertext links, submitting HTML forms, parsing HTML documents, decomposing a document into components using regular expressions, etc. Some languages also offer features such as natural language processing, web-, text- or data-mining algorithms, or access to structured data (e.g., ODBC sources or third-party applications).

This entry focuses primarily on the features of web data extraction languages. However, note that a key strength of some commercial Web data extraction products, and an active area of academic research, is graphical user interface for visually specifying the conversion process (see for example [2]). Similarly, there have been numerous attempts to automatically generate web data extraction programs using machine learning techniques (see for example [3,5,6,9]).

Note also that web data extraction is related to but distinct from web search technologies. Search engines crawl and index large numbers of web documents to facilitate subsequent ad-hoc keyword-based querying. In contrast, web data extractors retrieve a relatively small number of relevant documents (often in a specific sequence as the web sources perform various transactions), and then isolate specific document fragments in order to answer various predefined structured queries on their contents.

Historical Background

As the Web proliferated in the 1990s, computer science researchers from diverse backgrounds (databases, systems, artificial intelligence, information retrieval, etc) realized that the ability to integrate data across heterogeneous sources would give rise to a wide variety of compelling applications, such as shopping assistants that compare products across multiple retail sites. Many researchers who had been investigating traditional formulations of the data integration problem, turned their attention to integrating Web data. This attention revealed many new challenges. Data extraction quickly became prominent among these new challenges. Researchers simply could not demonstrate their Web data integration algorithms in a compelling manner until they developed systematic ways to automatically access large numbers of Web documents and then extract structured data from these documents' native formats.

The first approach to Web data extraction was simply to code up the necessary URL retrieval and data extraction in conventional programming languages. Indeed, many Web data integration applications continue to use this approach today. However, Web data extraction programs implemented in this manner tend to be relatively large, making it difficult to design, debug, re-use and maintain them.

Based on this experience, researchers quickly observed that web data extraction programs written in conventional programs exhibit many common software patterns. This observation led to efforts to encapsulate these patterns either as reusable libraries for existing languages, or as primitives in specialized web data extraction languages (see for example [1,4,7,8,10,11]). For instance, many web data extraction programs perform crawling behavior such as "parse an HTML document to find all its hyperlinks; then retrieve all newly discovered URLs, and repeat," or "submit a Web form several times, each time binding one of its input parameters to one of several values." To make it easier to build Web crawlers, this behavior could be encoded as a set of functions/classes in a library, or as primitives in a specialized Web data extraction language.

Foundations

From a theoretical perspective, it is meaningless to formally differentiate between Web data extraction languages and "regular" programming languages. Nearly every programming language has features of some aspect of web data extraction, such as retrieving URLs or applying regular expressions. Therefore, nearly any programming language can be used for web data extraction, and so every programming language is a web data extraction language. At the same time, some specialized web data extraction languages are Turing complete, so they can be used to implement arbitrary algorithms that have nothing in particular to do with Web data. Nevertheless, from a practical perspective, it can be much simpler to implement Web data extractors in some languages compared to others. This entry focuses on these practical issues, rather than formal distinctions.

Web data extraction comprises four distinct capabilities:

1. Access: Authenticating to web sites as needed, and fetching particular URLs
2. Extract: Populating data structures from retrieved documents
3. Process: Performing ancillary computations on extracted data
4. Output: Converting data to desired format and delivering to destination

Web data extraction languages vary in the sophistication with which they support these capabilities. All languages support some form of access and extraction, but some provide only limited support for data processing or output.

This entry explains these capabilities by examining one particular Web data extraction language, WebQL. WebQL is a product of QL2 Software Inc (www.QL2.com). While other languages offer a similar range of capabilities, WebQL is a good example for tutorial purposes because it has a familiar SQL-like syntax, and numerous features that address the four capabilities listed above.

WebQL programs consist of one or more *select* statements, which are similar to SQL select statements. When an SQL select statement retrieves values from database tables, a WebQL select statement retrieves data from a Web source. As in SQL, complete WebQL programs comprise a set of select statements that are combined using join, union and other data-flow operations.

Access

The simplest form of Web data access is fetching a single URL. The WebQL program

```
select
   source_content
from
   http://example.com/recipes.html
```

populates a table with one column, and one row, containing the HTML source for the given URL.

As discussed above, it is certainly true that URLs can be easily fetched in every modern programming language. However, web data retrieval languages usually offer a more compact syntax for performing actions that would be much more cumbersome in other languages. For example, the WebQL program

```
select
  source_content
from
  crawl of http://example.com/
  recipes.html
  to depth 3
  following if url matching 'potato'
```

implements a spider in WebQL that follows links that satisfy the stated criteria. This compact syntax takes care of numerous low-level details such as timing out, HTTP redirects, HTML parsing, broken links, etc. As a practical matter, this compact syntax generally leads to data extractors that are simpler to design, code, debug, maintain, and reuse. Just as it is formally possible but practically much more difficult to implement complex algorithms in low-level machine-level languages compared to high-level programming languages, web data extractors implemented in special-purpose programming languages are generally simpler to write and easier to extend and maintain.

Another aspect of data access is submitting HTML forms. For example, the WebQL statement

```
select
  source_content
from
  http://example.com/recipe_search
  submitting values ['potato', 'car-
  rot'] for 'ingredient'
```

parses the specified URL to detect HTML forms, and then submits each such form twice, each time binding the input parameter ingredient to one of the specified values. Again, the web data extraction language hides from the programmer details such as binding hidden form parameters.

Extract

After some specific web content has been accessed, web data extractors typically isolate particular fragments or properties of the document. Web programming languages offer numerous capabilities for data extraction. The simplest approach is to use the low-level string operations or regular expression matching capabilities that most programming languages provide. For example, the following WebQL segment extracts two text fragments from a particular URL using low-level operations:

```
select
  -- extract first paragraph, with low-
  level string functions
  substr(instr
  (source_content,'<p>')+3,instr
  (html,'</p>')
  as firstpara,
  -- extract first bold text, with regu-
  lar expression
  extract_pattern
  (source_content,'<b>(.*)</b>')
  as firstbold
from
  http://example.com/recipes.html
```

While these techniques suffice for many simple forms of extraction, most web data extraction languages provide extraction techniques that operate at a higher-level of abstraction. For example, an extractor may need to extract all images and the dimensions from a HTML document. One could do so using a regular expression such as:

```
<img src="([^"]*)" height="([^"]*)"
width="([^"]*)"
```

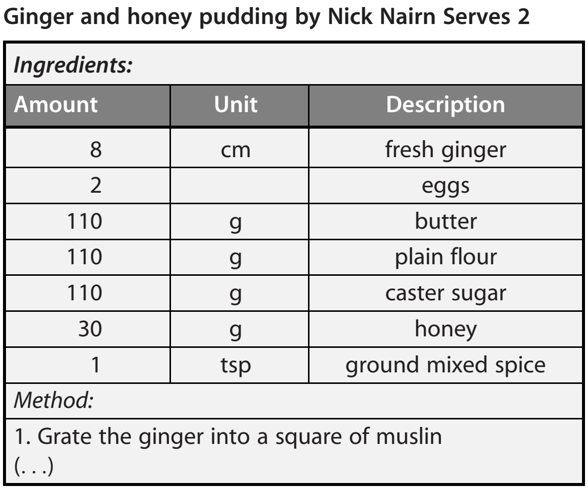
This is a very brittle approach. It assumes that the attributes are double-quoted, that none are missing, that they occur in the order listed, etc. This regular expression could be refined to accommodate these difficulties, but the resulting very long regular expression would need to be copied each time it is needed.

As an alternative, most web data extraction languages offer a suite of high-level extraction capabilities. For example, one can extract images from HTML documents using WebQL as follows:

```
select
  url,
  height,
  width
from
  images
within
  http://example.com/recipes.html
```

The “from images within” portion of this select statement is an example of a high-level extraction capability. WebQL and other web data extraction languages generally offer a range of such capabilities, including robustly parsing HTML forms, rendering HTML documents into plain text, invoking XPath queries on XML data, using linguistic rules to segment text into paragraphs, sentences, phrases, entities, words, etc, and extracting rows and/or columns from HTML tables.

As an example of the last capability, suppose that http://example.com/recipes.html contains tables embedded with text, such as the following:



Ginger and honey pudding by Nick Nairn Serves 2

Ingredients:		
Amount	Unit	Description
8	cm	fresh ginger
2		eggs
110	g	butter
110	g	plain flour
110	g	caster sugar
30	g	honey
1	tsp	ground mixed spice
Method:		
1. Grate the ginger into a square of muslin (. . .)		

The following WebQL select statement identifies the tables in this document and then retrieves the values of three specific columns by that are identified by name:

```
select
  amount,
  unit,
  description
from
  table values (amount, unit,
  description)
within
  http://example.com/recipes.html
```

While the details of these extraction techniques vary widely, and while low-level string or regular expression operations could be used to extract this data in any modern programming language, the general point holds: by providing abstract high-level data extraction operations, web data extraction languages make it much easier to implement extractors that are concise, correct, maintainable, and reusable.

Process

So far, this entry has focused on the advanced access and extraction functionality of web data extraction languages, compared to traditional programming languages. However, real web data extractors usually need to do more than simply access documents and extract their data; they also need to process this data in some way.

Web data extraction languages generally have a rich set of traditional programming language operators and control structures for manipulating and transforming data. For example, WebQL offers most of the data-flow operations in SQL dialects, such as joins, unions, grouping, sorting, and removing duplicates. WebQL also has a wide variety of functions for performing many kinds of operations. To illustrate these capabilities, consider the following query:

```
select as ExistingMeat
  ingredient as existing_ingredient,
  text_to_datetime("best before",
  'dd-mm-yy')
  as best_before_date
from
  table values (ingredient, catego-
  ry, "best before")
within
  http://www.example.com/current_fridge_
  contents.html
where
  category = 'meat'
join to ExistingMeat
select as RequiredIngredient
  description as
  required_ingredient,
  existing_ingredient
from
  table values (description)
within
  http://example.com/recipe_search
  submitting values existing_ingre-
  dient for 'ingredient'
join to ExistingMeat,
RequiredIngredient
  where ExistingMeat.existing_in-
  gredient =
  RequiredIngredient.
  existing_ingredient
select
  required_ingredient,
  best_before_date < now() + 24*60*60
  as urgent
sort by
  best_before_date desc
```

For the sake of this entry, there is no need to explain this program in detail. At a high level, it first extracts a list of ingredients currently in the refrigerator along with their category (meat, vegetables, etc) and expiration date; then requests recipes containing each meat that is in stock, and records which other ingredients are needed to make those recipe. And finally the program sorts the required ingredients by the expiration date of their meat, and also outputs a Boolean value that is true if the meat will soon expire. This example illustrates the ability to join intermediate tables, sort data, perform numerical and logical operations, and transform textual date/time values into a native numerical representation.

Output

After accessing, extracting and processing the required data, web data extractors must deliver the data to some destination in the appropriate format. Web data extraction languages vary in the range of output formats and delivery mechanisms that they support. To illustrate some of these capabilities, consider the following WebQL program:

```
select
  url,
  content
from
  links
within
  http://www.example.com/recipes.
  html
into
  -- write XML to local file
  file:links.xml,
  -- send CSV to an FTP site
  ftp://jsmith@mysecret:ftp.exam-
  ple.com/links.csv,
  -- write data as new rows in a database
  table
  'links'@docdb
```

The "into" clause causes the data to be encoded in specific formats and sent to specific destinations. This example program extracts the links from a web page, and then delivers them in various formats (comma-separated text file, XML) to both a local file and an FTP site, and also inserts the data directly into database table via ODBC.

Key Applications

Web data extraction languages are widely used in numerous commercial applications across a wide variety of industries The most common scenario is that an enterprise wants to extract data from external Web sources for which public Web Services or other programmatic APIs are not available. While numerous (often "Web 2.0"-oriented) data sources offer public APIs or structured data feeds such as RSS, many more sources do not. This is particularly true for high-value data extraction applications such as the gathering of real-time competitive market intelligence. For example, most large retailers, manufacturers and distributors employ some form of Web data extraction to monitor their competitor's prices. In most cases, the only way to harvest such data is by extracting it from public Web sources such as on-line retail catalogs or shopping sites.

Web data extraction is also used as a form of lightweight data integration for interconnecting legacy applications within an enterprise. Many companies are not in a position to modify critical business systems used for inventory management, pricing, customer management, etc., yet they want to integrate these systems. Web data extraction tools are frequently used to facilitate such integration by working with existing Web-based interfaces, rather than forcing the enterprise to modify their legacy systems.

A third important application area is so-called "mashups," applications that combine data from multiple sources into unified service. Web data extraction languages are clearly an appropriate technology with which to develop mashups. To many proponents, a key aspect of the mashup philosophy is the ability of ordinary users to generate their own applications. Therefore, most mashup technologies focus on visual interfaces that allows non-technical users to develop web data extraction programs.

While this entry focuses largely on technical issues, it is important to point out that the use of automated web data extractors may give rise to two sorts of legal issues. First, the terms of use of many Web sites explicitly prohibit automated content extraction. Second, republication of the extracted data may violate the original source's copyright on the data.

Future Directions

The core theoretical and scientific issues regarding web data extraction are well understood. While numerous companies are competing in the web data extraction/integration space, they compete primarily on the basis of the value-added services they deliver on top of the extracted data, rather than on the basis of their extraction technology. Nevertheless, as web technologies change, web data extraction languages will be enhanced to accommodate formats such as Flash/AMF in which more and more web data is embedded.

More significantly, some "web" data extraction languages provide access to a wide range of non-web sources (such as email, FTP sites, execution of external programs, etc) and formats (such as office documents and spreadsheets, archived and compressed data, etc). As these capabilities expand, these languages will not remain focused exclusively on web data, but rather they will evolve into powerful general-purpose tools for manipulating and integrating arbitrary structured and unstructured digital content.

URL to Code

There are several open-source Web data extraction tools, usually in the form of a library for a full-fledged programming language. Examples include:

1. Perl: WWW::Mechanize (search.cpan.org/dist/WWW-Mechanize)
2. PHP: Curl (www.php.net/curl)
3. Java: Web-Harvest (web-harvest.sourceforge.net)

Many companies compete in the Web data extraction market. These companies generally offer standalone data-extraction products, as well as on-demand services based on these technologies that are tailored to specific vertical industries. Examples include:

1. QL2 (www.ql2.com)
2. Fetch (www.fetch.com)
3. Kapow (www.kapowtech.com)
4. Lixto (www.lixto.com)
5. Connotate (www.connotate.com)

Cross-references

▶ Content-and-Structure Query
▶ Data Cleaning
▶ Data Integration
▶ Data Integration in Web Data Extraction Systems
▶ Database Reverse Engineering
▶ Focused Web Crawling
▶ Fully Automatic Web Data Extraction
▶ Hidden-Web Search
▶ Incremental Web Crawling
▶ Indexing Semi-Structured Data
▶ Indexing the "Web"
▶ Information Extraction
▶ Information Integration
▶ Logical Foundations for Web Data Extraction
▶ Metasearch Engines
▶ Query Language
▶ Screen Scraper
▶ Semantic Web
▶ Semi-Structured Data
▶ Semi-Structured Query Language
▶ Semi-Structured Text
▶ Structured Document Retrieval
▶ Text Mining
▶ Text Normalization
▶ Web Crawler Architecture
▶ Web Data Extraction System
▶ Web Harvesting
▶ Web Information Extraction
▶ Wrapper Induction
▶ Wrapper Maintenance
▶ Wrapper Stability
▶ XML Information Integration
▶ Xpath/XQuery

Recommended Reading

1. Arasu A. and Garcia-Molina H. Extracting structured data from Web pages. In Proc. 2003 ACM SIGMOD Int. Conf. on Management of data, 2003, pp. 337–348.
2. Baumgartner R., Flesca S., and Gottlob G. Visual Web information extraction with Lixto. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 119–128.
3. Crescenzi V., Mecca G., and Merialdo P. RoadRunner: towards automatic data extraction from large web sites. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
4. Kistler T. and Marais H. WebL – a programming language for the Web. Comput. Netw. ISDN Syst., 30(1–7):259–270, 1998.
5. Knoblock CA., Lerman K., Minton S., and Muslea I. Accurately and reliably extracting data from the web: A machine learning approach. In Intelligent Exploration of the Web, Piotr S. Szczepaniak, Javier Segovia, Janusz Kacprzyk and Lotfi A. Zadeh (eds.). Physica-Verlag Heidelberg, Heidelberg, pp. 275–287, 2003.
6. Kushmerick N. Wrapper induction: efficiency and expressiveness. Artif. Intell., 118(1–2):15–68, 2000.
7. Laender A.H.F., Ribeiro-Neto B.A., da Silva A.S., and Teixeira J.S. A brief survey of web data extraction tools. ACM SIGMOD Record, 31(2):84–93, 2002.
8. Liu L., Pu C., and Han W. XWRAP: an XML-enabled Wrapper Construction System for Web Information Sources. In Proc. 16th Int. Conf. on Data Engineering, 2000.
9. Muslea I., Minton S., and Knoblock C.A. Hierarchical wrapper induction for semistructured information sources. J. Auton. Agents Multi-Agent Syst., 4(1–2):93–114, 2001.
10. Sahuguet A. and Azavant F. Building intelligent web applications using lightweight wrappers. Data and Knowledge Engineering, 36(3):283–316, 2000.
11. Spertus E. and Andrea Stein L. Squeal: structured queries on the Web. In Proc. 9th Int. World-Wide Web Conference, 2000.

Large Itemsets

▶ Frequent Itemsets and Association Rules

Latch Coupling

▶ B-Tree Locking

Latching

▶ B-Tree Locking

Latent Semantic Indexing

▶ Singular Value Decomposition

Layer Algebra

▶ Spatial Operations and Map Operations

Layered Architecture

▶ Temporal Strata

Layered Transactions

▶ Multilevel Transactions and Object-Model Transactions

Lazy Replication

▶ Optimistic Replication and Resolution

LBS

▶ Location-based Services

Lévy Skew a-Stable Distribution

▶ Stable Distributions

Learning Distance Measures

Carlotta Domeniconi
George Mason University, Fairfax, VA, USA

Synonyms

Flexible metric computation; Adaptive metric techniques

Definition

Many problems in data mining (e.g., classification, clustering, information retrieval) are concerned with the discovery of homogeneous groups of data according to a certain similarity (or distance) measure. The distance measure in use strongly affects the nature of the patterns (clusters, classes, or retrieved images) emerging from the given data. Typically, any chosen fixed distance measure, such as Euclidean or Manhattan distance, does not capture the underlying structure of the data, and fails to find meaningful patterns which correspond to the user's preferences. To address this issue, techniques have been developed that learn from the data how to compute dissimilarities between pairs of objects. Since objects are commonly represented as vectors of measurements in a given feature space, distances between two objects are computed in terms of the dissimilarity between their corresponding feature components. In this setting, learning a (local) distance measure for a query object $\mathbf{q} \in \Re^n$ (to be classified) and an arbitrary object $\mathbf{x} \in \Re^n$ means to learn a weighted p-norm distance metric on the Euclidean space of the input measurement variables (or features):

$$D_p(\mathbf{q},\mathbf{x}) = \{\sum_{i=1}^{n} |[W(\mathbf{q})(\mathbf{q}-\mathbf{x})]_i|^p\}^{1/p},$$

where $p > 0$ and $W(\mathbf{q}) \in \Re^{n\times n}$ is a matrix of weights reflecting the relevance or importance of features at the query $\mathbf{q}$. If $W(\mathbf{q})$ depends on the query point $\mathbf{q}$, the resulting distance measure is *local*; otherwise, if W is invariant to the query, a *global* distance measure is obtained.

Historical Background

The problem of learning distance measures from data has attracted considerable interest recently in the data mining and machine learning communities. Different methodologies have been developed for supervised, unsupervised, and semi-supervised problems. One of the earliest work that discusses the problem of clustering simultaneously both points and features is Hartigan, 1972 [9]. A model based on direct clustering of the data matrix and a distance-based model are introduced, both leading to similar results.

Foundations

The ability to classify patterns is certainly one of the key features of intelligent behavior, whether it is humans' or animals'. This ability emerged with the biogenetic evolution for survival purposes, not only of individuals but also of entire species. An individual receives sensory information that must be processed to perceive, and ultimately act, possibly for orientation in the environment, distinction between edible and poisonous food, or detection of dangerous enemies.

Machine perception and classification as features of artificial systems serve similar but more constrained purposes. Machine classification aims to provide artificial systems with the ability to react to situations and signals that come from the environment to perform specific tasks. As such, pattern classification is a fundamental building block of any cognitive automata.

Recent developments in data mining have posed new challenges to pattern classification. Data mining is a knowledge discovery process whose aim is to discover unknown relationships and/or patterns from a large set of data that make it is possible to predict future outcomes. As such, pattern classification becomes one of the key steps in attempting to uncover the *hidden knowledge* within the data. The primary goal is usually predictive accuracy, with secondary goals being speed, ease of use, and interpretability of the resulting predictive model.

The term *pattern* is a common word and means something exhibiting some form of regularity, able to serve as a model representing a concept of what was observed. As a consequence, a pattern is never an isolated observation, but rather a collection of observations connected in time or space (or both). A pattern exhibits, as a whole, a certain structure indicative of the underlying concept. The pattern classification task can then be seen as the task of inferring concepts from observations. Thus, designing a pattern classifier means defining a mapping from a measurement space into the space of possible meanings, that are viewed as finite and discrete target points.

From this perspective, it makes no difference what kind of observations are considered and to what kind of meanings they may be linked. The same approach can be used to recognize written text, spoken language, objects, or any other multidimensional signals as well. The selection of meaningful observations from a specific domain is a *feature extraction* process. From a theoretical viewpoint, the distinction between feature extraction and classification is arbitrary, but nevertheless useful. In general, the problem of feature extraction is much more domain dependent than the problem of classification.

Statistical Approach

A characteristic of patterns in the context of classification is that every concept (or class) may have multiple representative points in the measurement space. For example, for the task of character recognition from their images, there exists a potentially unlimited plurality of ways to design character images that correspond to the same character. Therefore, the very core of pattern classification is to cope with variability. The difficulty of the task depends on the degree to which the representatives of a class are allowed to vary and how they are distributed in the measurement space. This observation brings together two intrinsic components of the pattern classification task: the *statistical* component and the principle of *learning from examples*.

The problem of classification can be seen as one of partitioning the feature space into regions, one region for each category. Ideally, one would like to arrange this partitioning so that no decisions is ever wrong. This objective may not be achievable for two reasons. The distributions of points of different classes in the measurement space overlap; thus, it is not possible to reliably separate one class from the other. Moreover, even if a rule that does a good job of separating the examples can be found, one has no guarantee that it will perform as well on new points. In other words, that rule may not *generalize* well on data never seen before. It would certainly be safer to consider more points, and check how many of those are correctly classified by the rule. This suggests that one should look for a classification procedure that aims at minimizing the *probability of error*. The problem of classification then becomes a problem in statistical decision theory.

Challenges

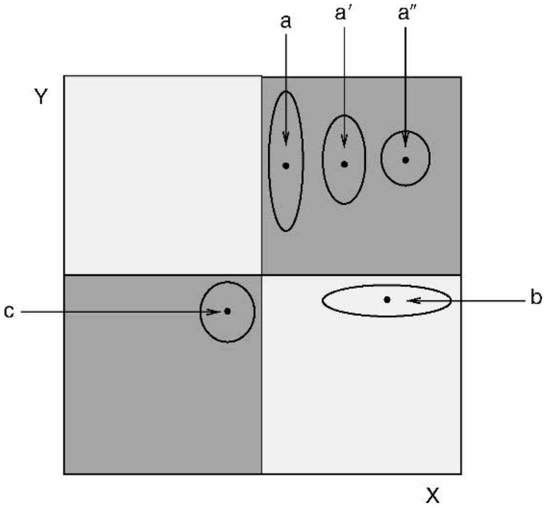
While pattern classification has shown promise in many areas of practical significance, it faces difficult challenges from real world problems, of which the most pronounced is Bellman's *curse of dimensionality* [1]. It states the fact that the sample size required to perform accurate prediction in problems with high dimensionality is beyond feasibility. This is because in high dimensional spaces, data become extremely sparse and are apart from each other. As a result, severe bias that affects any estimation process can be introduced in a high dimensional feature space with finite samples.

Consider, for example, the rule that classifies a new data point with the label of its closest training point in the measurement space (*1-Nearest Neighbor rule*). Suppose each instance is described by 20 attributes, but only three of them are relevant to classifying a given instance. In this case, two points that have identical values for the three relevant attributes may nevertheless be distant from one another in the 20-dimensional input space. As a result, the similarity metric that uses all 20 attributes will be misleading, since the distance between neighbors will be dominated by the large number of irrelevant features. This shows the effect of the curse of dimensionality phenomenon, that is, in high dimensional spaces distances between points within the same class or between different classes may be similar. This fact leads to highly biased estimates. Nearest neighbor approaches are especially sensitive to this problem.

In many practical applications things are often further complicated. In the previous example, the three relevant attributes for the classification task at hand may be dependent on the location of the query point, i.e., the point to be classified, in the feature space. Some features may be relevant within a specific region, while other features may be more relevant in a different region.

These observations have two important implications. Distance computation does not vary with equal strength or in the same proportion in all directions in the feature space emanating from the input query. Moreover, the value of such strength for a specific feature may vary from location to location in the feature space. Figure 1 illustrates a case in point, where class boundaries are parallel to the coordinate axes. For query *a*, dimension *X* is more relevant, because a slight move along the *X* axis may change the class label, while for query *b*, dimension *Y* is more relevant. For query *c*, however, both dimensions are equally relevant. Capturing such information, therefore, is of great importance to any classification procedure in high dimensional settings.

It is important to emphasize that the curse of dimensionality is not confined to classification. It affects any estimation process in a high dimensional feature space with finite examples. Thus, clustering equally suffers from the same problem. The clustering problem concerns the discovery of homogeneous groups of data according to a certain similarity measure. It is not meaningful to look for clusters in high dimensional spaces as the average density of points anywhere in input space is likely to be low. As a consequence, distance functions that equally use all input features may not be effective.



Learning Distance Measures. Figure 1. Feature relevance varies with query locations.

Adaptive Metric Techniques

This section presents an overview of relevant work in the literature on flexible metric computation for classification and clustering problems.

Adaptive Metric Nearest Neighbor Classification

In a classification problem, one is given l observations $\mathbf{x} \in \mathfrak{R}^n$, each coupled with the corresponding class label y, with $y = 1,\ldots,J$. It is assumed that there exists an unknown probability distribution $P(\mathbf{x}, y)$ from which data are drawn. To predict the class label of a given query $\mathbf{q}$, the class posterior probabilities $\{P(j|\mathbf{q})\}_{j=1}^{J}$ need to be estimated.

K nearest neighbor methods are based on the assumption of smoothness of the target functions, which translates to locally constant class posterior probabilities $P(j|\mathbf{q})$, that is: $P(j|(\mathbf{q} + \delta\mathbf{q})) \simeq P(j|\mathbf{q})$, for $\|\delta\mathbf{q}\|$ small enough. Then, $P(j|\mathbf{q}) \simeq \frac{\sum_{\mathbf{x}\in N(\mathbf{q})} P(j|\mathbf{x})}{|N(\mathbf{q})|}$, where $N(\mathbf{q})$ is a neighborhood of $\mathbf{q}$ that contains points $\mathbf{x}$ that are "close" to $\mathbf{q}$, and $|N(\mathbf{q})|$ denotes the number of points in $N(\mathbf{q})$. This motivates the estimates

$$\hat{P}(j|\mathbf{q}) = \frac{\sum_{i=1}^{l} 1(\mathbf{x}_i \in N(\mathbf{q}))1(y_i = j)}{\sum_{i=1}^{l} 1(\mathbf{x}_i \in N(\mathbf{q}))},$$

where $1()$ is an indicator function such that it returns 1 when its argument is true, and 0 otherwise.

The assumption of smoothness, however, becomes invalid for any fixed distance metric when the input observation approaches class boundaries. The objective of locally adaptive metric techniques for nearest neighbor classification is then to produce a modified local neighborhood in which the posterior probabilities are approximately constant.

The techniques proposed in the literature [5,6,10] are based on different principles and assumptions for the purpose of estimating feature relevance locally at query points, and therefore weighting accordingly distances in input space. The idea common to these techniques is that the weight assigned to a feature, locally at a given query point **q**, reflects its estimated relevance to predict the class label of **q**: larger weights correspond to larger capabilities in predicting class posterior probabilities. As a result, neighborhoods get constricted along the most relevant dimensions and elongated along the less important ones. The class conditional probabilities tend to be constant in the resulting neighborhoods, whereby better classification performance can be obtained.

Large Margin Nearest Neighbor Classifiers

The previously discussed techniques have been proposed to try to minimize bias in high dimensions by using locally adaptive mechanisms. The "lazy learning" approach used by these methods, while appealing in many ways, requires a considerable amount of on-line computation, which makes it difficult for such techniques to scale up to large data sets. Recently, a method (called LaMaNNA) has been proposed which, although still founded on a query based weighting mechanism, computes off-line the information relevant to define local weights [3].

The technique uses support vector machines (SVMs) as a guidance for the process of defining a local flexible metric. SVMs have been successfully used as a classification tool in a variety of areas [13], and the maximum margin boundary they provide has been proved to be optimal in a structural risk minimization sense. The decision function constructed by SVMs is used in LaMaNNA to determine the most discriminant direction in a neighborhood around the query. Such direction provides a local feature weighting scheme. This process produces highly stretched neighborhoods along boundary directions when the query is close to the boundary. As a result, the class conditional probabilities tend to be constant in the modified neighborhood, whereby better classification performance can be achieved. The amount of elongation-constriction decays as the query moves farther from the vicinity of the decision boundary. This phenomenon is exemplified in Fig. 1 by queries a, a' and a''.

Adaptive Metrics for Clustering and Semi-Supervised Clustering

Adaptive metric techniques for data without labels (unsupervised) have also been developed. Typically, these methods perform clustering and feature weighting simultaneously in an unsupervised manner [2,4,7,8,12]. Weights are assigned to features either globally or locally.

The problem of feature weighting in K-means [11] clustering has been addressed in [12]. Each data point is represented as a collection of vectors, with "homogeneous" features within each measurement space. The objective is to determine one (global) weight value for each feature space. The optimality criterion pursued is the minimization of the (Fisher) ratio between the average within-cluster distortion and the average between-cluster distortion.

COSA (Clustering On Subsets of Attributes) [7] is an iterative algorithm that assigns a weight vector (with a component for each dimension) to each data point. COSA starts by assigning equal weight values to each dimension and to all points. It then considers the k nearest neighbors of each point, and uses the resulting neighborhoods to compute the dimension weights. Larger weights are credited to those dimensions that have a smaller dispersion within the neighborhood. These weights are then used to compute dimension weights for each pair of points, which in turn are utilized to update the distances for the computation of the k nearest neighbors. The process is iterated until the weight values become stable. At each iteration, the neighborhood of each point becomes increasingly populated with data from the same cluster. The final output is a pairwise distance matrix based on a weighted inverse exponential distance that can be used as input to any distance-based clustering method (e.g., hierarchical clustering).

LAC (Locally Adaptive Clustering) [4] develops an exponential weighting scheme, and assigns a weight vector to each cluster, rather than to each data point. The weights reflect local correlations of data within each discovered cluster, and reshape each cluster as a dense spherical cloud. The directional local reshaping of distances better separates clusters, and allows for the discovery of different patterns in different subspaces of the original input space.

Recently, to aid the process of clustering data according to the user's preferences, a semi-supervised framework has been introduced. In this scenario, the user provides examples of similar and dissimilar points, and a distance metric is learned over the input space that satisfies the constraints provided by the user [14].

Key Applications

Almost all problems of practical interest are high dimensional. Thus, techniques that learn distance measures have significant impact in fields and applications as diverse as bioinformatics, security and intrusion detection, document and image retrieval. An excellent example, driven by recent technology trends, is the analysis of microarray data. Here one has to face the problem of dealing with more dimensions (genes) than data points (samples). Biologists want to find "marker genes" that are differentially expressed in a particular set of conditions. Thus, methods that simultaneously cluster genes and samples are required to find distinctive "checkerboard" patterns in matrices of gene expression data. In cancer data, these checkerboards correspond to genes that are up- or downregulated in patients with particular types of tumors.

Cross-references

- ▶ Classification
- ▶ Cluster and Distance Measure
- ▶ Clustering with Constraints
- ▶ Curse of Dimensionality
- ▶ Data Mining
- ▶ Feature Selection for Clustering
- ▶ Nearest Neighbor Classification

Recommended Reading

1. Bellman R. Adaptive Control Processes. Princeton University Press, 1961.
2. Blansch A., Ganarski P., and Korczak J. Maclaw: a modular approach for clustering with local attribute weighting. Pattern Recognit. Lett., 27(11):1299–1306, 2006.
3. Domeniconi C., Gunopulos D., and Peng J. Large margin nearest neighbor classifiers. IEEE Trans. Neural Netw., 16:899–909, 2005.
4. Domeniconi C., Gunopulos D., Yan S., Ma B., Al-Razgan M., and Papadopoulos D. Locally adaptive metrics for clustering high dimensional data. Data Mining Knowl. Discov. J., 14:63–97, 2007.
5. Domeniconi C., Peng J., and Gunopulos D. Locally adaptive metric nearest neighbor classification. IEEE Trans. Pattern Anal. Mach. Intell., 24:1281–1285, 2002.
6. Friedman J. Flexible metric nearest neighbor classification. In Tech. Report, Dept. of Statistics, Stanford University, 1994.
7. Friedman J. and Meulman J. Clustering Objects On Subsets of Attributes. Technical Report, Stanford University, 2002.
8. Frigui H. and Nasraoui O. Unsupervised learning of prototypes and attribute weights. Pattern Recognit., 37(3):943–952, 2004.
9. Hartigan J.A. Direct clustering of a data matrix. J. Am. Stat. Assoc., 67(337):123–129, 1972.
10. Hastie T. and Tibshirani R. Discriminant adaptive nearest neighbor classification. IEEE Trans. Pattern Anal. Machine Intell., 18:607–615, 1996.
11. Jain A., Mutty M., and Flyn P. Data clustering: a review. ACM Comput. Surv., 31(3), 1999.
12. Modha D. and Spangler S. Feature weighting in K-means clustering. Mach. Learn., 52(3):217–237, 2003.
13. Shawe-Taylor J. and Fiege N. Pietzuch P. Kernel Methods for Pattern Analysis. Cambridge University Press, London, 2004.
14. Xing E., Ng A., Jordan M., and Russell S. Distance metric learning, with application to clustering with side-information. Advances in NIPS, vol. 15, 2003.

Learning in Streams

▶ Classification on Streams

Length Normalization

▶ Document Length Normalization

Level-of-Detail (LOD) Terrain Modeling

▶ Multi-Resolution Terrain Modeling

Levelwise Search

▶ Apriori Property and Breadth-First Search Algorithms

Lexical Affinities

▶ Term Proximity

Lexical Analysis of Textual Data

Chris D. Paice
Lancaster University, Lancaster, UK

Synonyms

Lexical processing; Term processing

Definitions

Lexical analysis refers to the association of meaning with explicitly specified textual strings, referred to here as *lexical terms*. These lexical terms are typically obtained from texts (whether natural or artificial) by a process called *term extraction*. The association of meaning with lexical terms involves a data structure known generically as a *lexicon*. The characteristic operation in using a lexicon is a *lookup*, where the input is a lexical term, and the output is a representation of one or more associated meanings. A lexicon consists of a collection of *entries*, each of which comprises an *entry term* and a meaning structure. Lookup entails finding any entries whose entry term matches the lexical term in question.

Here, the term lexical analysis is used to refer only to operations performed on complete words or word groups. Operations on the characters within words is the concern of *morphology*.

The use of *text corpora* for obtaining data on the properties of words may also be regarded as a branch of lexical analysis. A text corpus provides a large representative sample of a language or sublanguage, and may be used for generating data-sets such as lexicons (see later), for providing statistical information, and for identifying examples of particular language constructs for researchers. Operations on corpora include annotating words or word groups with grammatical or other information. [9,11]

Historical Background

The need for automatic lexical analysis dates back to the 1950s, when the need arose for programing language compilers to recognize variable names and reserved words, and assign an appropriate role to each one. As soon as computers started being used for processing natural language texts, lexical processing was required for extracting, recognizing, organizing, and correlating words and phrases. At first, lexicons for term recognition were compiled by hand, or adapted from existing dictionaries and lists, but later, automatic tools were developed, either for generating lexicons outright, or for reducing the amount of human effort involved in constructing them.

Foundations

A natural language word is a sign whose meaning cannot be inferred (except in special cases such as onomatopoeic words) from its phonological or morphological structure. *Lexical analysis* is a process by which meanings are associated with specific words or other textual strings. These strings, which can be referred to as *lexical terms*, or just *terms*, are typically extracted during the scanning of some document.

Lexical terms may be individual words belonging to a natural or artificial language, but they may also include abbreviations such as "IBM" and "USAAF", 'pseudowords' such as "PL/1" and "B12", and even multiword expressions (MWEs) such as "winter wheat" or "Boeing 747." The key point is that the term must represent a known concept which is, or may be, relevant to some current need. The function of lexical analysis is to return information enabling that need to be satisfied.

A data structure or data collection which associates a set of lexical terms with their meanings is known as a *lexicon*. The meaning associated with a specified lexical term may be referred to as an *output* of the lexicon. The structure of the lexicon and the nature of the output vary according to what kind of information is required for the task at hand. Some applications may require access to two or more different lexicons, covering different (or partly different) sets of terms, and yielding different kinds of output.

Although the detailed arrangements can vary, a lexicon consists essentially of a set of *entries*, each of which maps a term to a meaning. In some lexicons, a given term can yield a number of alternative meanings. In such a case, the application program must choose the most appropriate meaning or (depending on the task at hand) make use of them all.

The archetypal lexicon is a traditional *dictionary*, in which each natural language word is accompanied by a definition of each distinct meaning of the word, together with indicators for pronunciation, syntax, etymology, etc. [11]

In a lexicon, the key operation is the *lookup*, which takes a term and returns an output representing the required meaning or meanings. In cases where the term is not present in the lexicon, a "null" or "false" output is returned.

In applications where the range of terms and meanings are very small, lexical analysis may be performed by encoding (or "hard wiring") the relevant terms as literals in decision structures in a computer program.

Lexical Analysis in Text Processing

1. Lexical extraction Reference to a lexicon is frequently required by an application program designed to process textual data, whether this consists of text in a natural language, or in an artificial language such as a programming language. In extracting terms from a text, the first step is normally to *tokenize* the text – that is, to divide the stream of characters into coherent groups called *tokens*, as appropriate for the task at hand. In a natural text, the tokens may include words, numbers, punctuation symbols, brackets, etc.

As noted earlier, lexical analysis typically involves the lookup of significant items such as words and phrases. Tokenization therefore usually involves, or is immediately followed by, a filtering process which recognizes and discards irrelevant items such as punctuation marks. In fact, in a language like English, extraction of single words can be achieved by regarding all characters except letters, and perhaps digits, as token delimiters. There remain only certain specific issues, such as the handling of hyphens and apostrophes, where the extracted terms need to conform to the practice used in the lexicon. [6,8]

If the lexicon includes terms which are multiword expressions (MWEs), there is a problem in that MWEs are not explicitly flagged in a text, and so cannot be extracted directly. The only exception is the names of places, people and organizations, such as "Department for Transport" or "United Arab Emirates," where the main component words are capitalized.

Lexical terms are almost always represented by grammatical noun phrases (NPs). Hence, one approach to the extraction of MWEs is to parse the text and then extract the NPs as potential terms. Parsing is however a slow process, and only likely to be worthwhile if the output of the parser is to be used for other purposes as well. Fortunately, NPs can also be identified by a more limited syntactic process based on recognizing patterns of grammatical tags, which can be done reasonably quickly. [11]

A more simple-minded approach to term extraction involves generating all sequences of successive tokens up to a specified length, and checking each against the lexicon in decreasing order of length. With a four-token limit, this would seem to involve checking four potential terms per token, but in fact this is an overestimate. Firstly, sequences starting or finishing with a trivial word like "of" or "for", or containing a punctuation symbol, might be discarded at once (though this begs the question of how efficiently these cases can be recognized). Secondly, if the first word of a long sequence fails to match, the shorter sequences starting with that word need not be tested. All in all, with its avoidance of syntactic processing, this can be quite an effective approach.

A different method is to only extract single tokens. but if the current token happens to match the first word of a multiword term, the succeeding tokens of the text are checked against the remainder of that entry (and also against any other entries starting with the same word).

If it is known that the terms of interest will only occur in certain specific contexts, rules can be used for defining those contexts. For instance, a rule may be devised for extracting surnames from personal names, covering examples such as "Mrs Robinson," "John F. Ratley, Jr," and "Revd Ben Wiles". Alternatively, data may need to be extracted from semiformatted texts such as clinical reports. Rules of this kind may consist of, or be based on, regular expressions – see URL below, and reference [4].

As noted later, extracted terms are sometimes compiled into an index before further processing, but for the present it is assumed that each term is immediately passed on for lookup.

Besides terms extracted from a text, lexical processing may need to be applied to terms retrieved from a database or knowledge base, extracted from a user query, or entered into a text box in a user interface. In some cases, the lexical terms might be artificial values, such as employee numbers assigned by a company or internal identifiers assigned by a database system.

2. *Lookup* For a lexicon of any size, the efficiency of the lookup is of major importance. The techniques here are well known, including simple binary searching in arrays, the use of access trees, and hashing techniques. For fixed sets of terms, hashing is potentially the fastest of all methods, but if terms are likely to be added or deleted frequently access trees are probably better. Further details about these methods can be found in any good book on data structures.

Of course, in the case of a lexicon designed for use by human beings, alphabetical ordering is the almost invariable rule.

3. ***Use of the Output*** The output of a lexicon, unless it is simply to be displayed on a user's workstation, will be subjected to further processing by the application program. As noted earlier, the output of each lexicon is designed according to its intended uses.

Each entry in a lexicon consists of an entry term and an output structure. Besides the outputs explicitly defined in the lexicon, there is normally a default output which is returned in cases where the lexical term is not included among the entry terms.

The simplest form of lexicon is one in which the explicit outputs are all void; in other words, the lexicon is simply a *list* of terms. In effect, the lookup process returns the value "true" or "false," depending on whether or not the term was found. Such a lexicon defines a *set* of terms, and the lookup is a test for *set membership.*

The simplest form of nonvoid lexicon is a simple association list, in which each output is a single value. In general, however, the outputs can have an arbitrarily complicated structure, as defined by two factors: multiplicity and complexity.

Multiplicity of output refers to the fact that, in many lexicons, a given entry term may be associated with a number of different outputs (or equivalently, there may be a number of separate entries which have the same entry term). In many natural language tasks, multiplicity of output equates to ambiguity (as in the various distinct meanings of the English word "dock"), which represent a problem to be resolved. In other cases (e.g., when the outputs enumerate the members of a set) the multiplicity is not a problem; for example, in an inverted file (see below) multiple outputs are the norm.

Complexity of output. In many lexicons each output consists of several differentiated data items. The output may for example be structured as a record or tuple, or it may have a deeper structure, as in some examples below.

Types of Lexicon

Word lists and Stoplists As noted above, the simplest form of lexicon is a simple term list representing a set of terms which share some common property. Membership of such a set can result in the acceptance of a term for further processing by the application, or alternatively can cause its rejection. An example of the latter case is a *stoplist*, which is routinely used in information retrieval systems to eliminate trivial words ("stopwords") which can serve no purpose for retrieval. These are mainly syntactic function words ("closed class words") such as the English words "the," "of," "and," "for," "over," "because," etc.

Indexes and Inverted files An *index* is a secondary structure which is provided to facilitate access to the items held in some primary body of information. A back-of-book index provides a familiar everyday example, but computer-based indexes can be generated automatically for any kind of information object which includes lexical data. An important example is the use of *inverted files* for assisting the retrieval of records in databases. Given a database in which each record contains a set of terms summarizing the properties of a particular entity, inspecting all of the records individually will likely be a slow process. Instead, an inverted file is generated in which each distinct term is associated with an entry listing the identifiers of the specific records which contain that term. Searching can then be performed quickly by accessing and comparing only those few entries which relate to the specified search terms [6].

Machine Readable Dictionaries The outputs of traditional dictionaries and glossaries were designed for perusal by human beings. However, with their generally predictable structure, they proved to be a valuable lexical resource for natural language processing applications [2]. Their great advantage was that they provided a ready-made summary of the whole of the language, apart from specialized technical terms. Their disadvantage was that their presentation of information was often inconsistent and incomplete. Experience of such problems soon led to the development of electronic dictionaries with more formally defined structures; perhaps the best-known example is the Longman Dictionary of Contemporary English (LDOCE), which is now available on CD-ROM.

Thesauruses A thesaurus is a structure which records and classifies relationships between distinct lexical terms. Probably its best-known use is for detection of synonyms, to enable distinct but equivalent terms to be merged or conflated. In information retrieval, a thesaurus may be used for expanding a set of query terms, thus improving the recall performance of a query. Some of the most detailed thesauruses provide information about the terminology of a technical domain, such as medicine or agriculture.

In fact, a well-developed thesaurus provides information about a range of distinct interterm relationships, and may support two distinct types of operation: (i) vocabulary control, and (ii) classification of relationships. Vocabulary control consists of mapping each lexical term onto a *preferred term*, denoting some specific concept in the domain, thus effectively conflating different (synonymous) mentions of that concept. Such a thesaurus also includes information about semantic relationships between distinct concepts, including antonymy and various hierarchical relations. The latter almost always include genus/type relations (e.g., plant/tree/oak), and often also whole/part relations (e.g., tree/branch/twig) and others. [11]

The WordNet is an online thesaurus developed at Princeton University in the early 1990s [5,10]. In WordNet, groups of synonymous words are organized into "synsets" (but no preferred terms are designated). Words which are polysemous (i.e., possess more than one meaning) will occur in two or more synsets – for example, (board, plank) and (board, committee). This means that the entry term "board" returns two distinct outputs. WordNet also allows five further relations to be recorded between synsets – antonymy, genus/type, whole/part, manner, and logical entailment.

Although WordNet is an attractive and useful tool, it is really quite "weak" in terms of completeness and consistency. Thus, its use for supplementing search terms in information retrieval systems has led to disappointing results [13]. It may be more useful for providing suggestions or asking questions in an online context.

The EDR dictionary, now administered by the Japanese National Institute of Information and Communications Technology, provides an impressive range of lexical resources [14]. One of its main purposes is to support the development of robust and effective machine translation tools. Primary lexical access is via word dictionaries in Japanese and English, and these include links to a "concept dictionary" containing semantic information and hierarchical links to related concepts. Other components include co-occurrence dictionaries and a Japanese/English bilingual dictionary.

A thesaurus or online dictionary provides a kind of sketchy model of the world (or partial world), and the language used to describe it. The most fully developed systems of this kind are *ontologies* which, besides the features outlined above, may include definitions of processes, interactions, constraints, and temporal relationships [11].

Construction of Lexicons

Any body of text can be converted automatically into an *index* of words (or, with a little more trouble, of phrases) by extracting the words/phrases as discussed earlier and organizing them into an alphabetical list. An index thus produced is of course a kind of lexicon – one in which the outputs are restricted, e.g., to positional information or frequency. Many language processing applications start off by generating an index of extracted terms, which can then be inspected or manipulated at a later stage. Note that an inverted file is a form of index.

The compilation of more complex forms of lexicon can only be automated to a limited extent. Statistical analysis of text corpora can provide useful lexical resources, or at least resources which can be used to facilitate the construction of practically useful lexicons [9]. A key activity is looking for associations (collocations) between words in a text corpus [3]. This involves taking the corpus, dividing it into suitable segments (e.g., paragraphs, sentences, or fixed-size windows) and then measuring the extent to which different terms tend to occur together using, e.g., the mutual information measure. One application of this approach is in the construction of a *statistical thesaurus*, by which strongly associated terms may be used for expanding sets of query terms in information retrieval systems [1].

In a statistical thesaurus, the associations between terms are undifferentiated "tending-to-occur-together" relationships. More detailed analysis of patterns of word association may be used, tentatively, to differentiate between relationships of various types. One noteworthy example of this approach is the work of Greffenstette [7], while some further efforts in this direction have been reviewed by Matsumoto [11] and Srinivasan [6].

Key Applications

As has already been noted, some kind of lexical analysis is performed by almost every program which operates on textual data, including systems for text retrieval, question-answering, summarization, information extraction, data mining, and machine translation.

Data Sets

The URLs given below provide datasets as well as lexical analysis tools.

URL to Code

EDR Electronic Dictionary: http://www2.nict.go.jp/r/r312/EDR/index.html

Longman's Dictionary (LDOCE): http://www.ldoceonline.com/

Regular expressions: http://www.regular-expressions.info/

WordNet: http://wordnet.princeton.edu/

Cross-references

- ▶ Index Creation and File Structures
- ▶ Stemming
- ▶ Stoplists
- ▶ Query Expansion for Information Retrieval

Recommended Reading

1. Chen H., Schatz B., Yim T., and Fye D. Automatic thesaurus generation for an electronic community system. J. Amer. Society for Inform. Science, 46(3):175–193, 1995.
2. Chodorow M., Byrd R., and Heidorn, G. Extracting semantic hierarchies from a large on-line dictionary. In Proc. 23rd Annu. Meeting of the Association for Computation Linguistics, Illinois, USA, 1985, pp. 299–304.
3. Church K. and Hanks P. Word association norms: mutual information and lexicography. Computational Linguistics, 16(1): 22–29, 1990.
4. Clarke C.L.A. and Cormack G.V. On the use of regular expressions for searching text. ACM Transactions on Programming Languages and Systems, 19(3):413–426,1997.
5. Fellbaum C. (ed.). WordNet: An Electronic Lexical Database, MIT Press, Cambridge, MA, 1998.
6. Frakes W.B. and Baeza-Yates R. Information Retrieval: Data Structures & Algorithms, Chapters III, VII, and IX. Prentice-Hall, 1992.
7. Greffenstette G. Explorations in Automatic Thesaurus Discovery, Boston: Kluwer, 1994.
8. Greffenstette G. Tokenization, in H. van Halteren (ed.). Syntactic Wordclass Tagging, Kluwer, The Netherlands, 1999, pp. 117–133.
9. McEnery T. and Wilson A. Corpus Linguistics. Edinburgh University Press, UK, 2001.
10. Miller G.A., Beckwith R., Fellbaum C., Gross, D., and Miller K.J. Introduction to WordNet: an on-line lexical database. Int. J. Lexicography, 3:235–244, 1990.
11. The Oxford Handbook of Computational Linguistics. *1*Chapters III, XXI, XXIV, XXV, and XXXIII. Mitkov R. (ed.). Oxford University Press, UK, 2003.
12. Schneider J.W. and Borland P. Introduction to bibliometrics for construction and maintenance of thesauri: methodical considerations. J. Doc. 60(5):524–549, 2004.
13. Voorhees E.M. Using WordNet to disambiguate word senses for text retrieval. In Proc. 16th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 171–180.
14. Yokoi T. The EDR electronic dictionary. Commun. ACM, 38(11): 42–44, 1995.

Lexical Processing

▶ Lexical Analysis of Textual Data

Lexical Relations

▶ Term Proximity

Library of Congress METS

▶ LOC Mets

License

- ▶ Copyright Issues in Databases
- ▶ European Law in Databases
- ▶ Licensing and Contracting Issues in Databases

Licensing and Contracting Issues in Databases

MICHAEL W. CARROLL
Villanova University School of Law, Villanova, PA, USA

Synonyms

License

Definition

A contract is a legally binding agreement between two or more parties usually formed when each party promises to do some action not otherwise required or to refrain from taking some action not otherwise required. In this context, a license is a grant of rights or permission by the owner of rights under copyright in a database. The key differences between a copyright license and a contractual database license are who is subject to the license and the consequences of breaching the license. A breach of a copyright license usually exposes the breaching party to liability for copyright infringement, which can include remedies such as injunctions, statutory damages, and disgorgement of profits. In contrast, a breach of contract exposes the breaching party to liability for the database owner's actual damages, which must be proven with reasonable certainty. Oftentimes, legal agreements styled as "terms of use" or as licenses in relation to databases are technically only partially copyright licenses and are otherwise merely contracts. This is because such agreements often contain terms governing the use of copyrightable elements of a database – such as its structure – and non-copyrightable data, such as numerical or other factual information. One who copies an entire database without permission would likely face liability for copyright infringement; whereas, one who copies factual data only without permission would be liable only for breach of contract.

Historical Background

Contract law has ancient roots, but it evolved into a general field relatively recently. In "common law" countries, such as the United States, the United Kingdom, Canada, and Australia, the law of contract is defined and applied by the courts. In "civil law" countries, which comprise the large majority of nations, the principles of contract are codified by the legislature and applied by the courts. Under both systems of law, a contract is enforceable if there has been an offer, an acceptance, and an exchange or "consideration," which means that each party has agreed to give the other something of value or has promised to for bear from taking action that would otherwise be lawful.

Foundations

To fill perceived gaps in copyright law, some database owners also have come to require agreement to contracts or license agreements in exchange for access to their databases. Three sets of legal issues arise with respect to these licenses: (i) what are the requirements to make them enforceable; (ii) are they enforceable if they control uses of non-copyrightable data; and (iii) what law applies if there are disputes about interpretation or enforcement of the license.

In the United States, a contract is enforceable if there is an offer, an acceptance, and consideration, which means that the parties have entered into a bargained-for-exchange. If the terms of a license are that the database owner agrees to make the database accessible and usable in exchange for the user's promises to pay for access or to refrain from making certain uses of the data – such as redistributing or copying data, the contract will be enforceable.

Outside of the United States, the requirement of exchange is supplemented by other methods of enforcement such that database licenses will generally be held enforceable under the applicable law of contract or obligations.

The key differences between a copyright license and a contractual database license are who is subject to the license and the consequences of breaching the license. Copyright is a right against the public, and a copyright owner may therefore grant the public a license to use the copyrighted elements of a database subject to a public license, such as a Creative Commons license. Most copyright licenses are drafted to say that if the user breaches, the copyright license terminates and any further use of the copyrighted elements becomes copyright infringement. The remedies for copyright infringement are more severe than for breach of contract.

A contractual database license is enforceable only against one who is a party to the agreement. If the license is from party *A* to party *B*, and party *C* obtains the database or data from party *B*, party *A* cannot enforce the contract against party *C*. To avoid this result, owners of digital databases often include terms in the database to the effect that any person who uses the database becomes a party to the license agreement.

There is some doubt about whether this approach comports with traditional requirements for enforceable contracts. Nonetheless, with respect to "terms of use" on web sites housing electronic databases courts have relaxed the requirement of exchange and have held that users accept such terms through the act of visiting or using the site.

With respect to whether a database owner may control uses of non-copyrightable data or databases by a contractual license when copyright law permits such uses, the question is one of preemption. Does contract law, which is a matter of state law in the United States, conflict with federal copyright law? If there is a conflict, federal law prevails. However, the few courts that have addressed this issue have reasoned that such licenses are not in conflict with copyright because the license applies only to the other party whereas copyright applies to the whole world.

Finally, the issue of which law governs the interpretation and enforcement of a contractual database license is an example of a choice-of-law issue. Often the license will specify what law applies, and courts will generally accept this term as long as the jurisdiction chosen has a nexus with the parties. Otherwise, the question of choice of law is determined through fact-specific balancing of the location of the parties, the place where the contract was formed, and, in the case of a digital library, the location of the server(s).

Key Applications

After the United States Supreme Court held in 1991 that a white pages telephone directory was not copyrightable, a case arose in which a database of more than 3,000 white pages directories had been compiled and released on CD-ROM with a license agreement that restricted those who purchased it to non-commercial uses. The defendant copied the data and resold it through his web site. The question presented was whether the non-commercial use restriction in the license agreement was enforceable even though the data was not protected by copyright. The court of appeals held that the agreement was enforceable. The court held that there was no conflict between the contract's restriction on copying the data and copyright law's position that the data could be freely copied because the contract only restricted those who agreed to its terms. The court reached this result only with respect to a particular section of the Copyright Act and did not address the argument that the constitution also preempted enforcement of the license agreement.

Other courts have largely followed this approach of enforcing similar agreements referred to as "shrinkwraps," when the license terms are inside a shrinkwrapped box of software; "clickwraps," where a user clicks an "I Agree" button to signal agreement to license terms; and, more controversially, to "browsewraps," in which the issue is whether a user of a web site has agreed to its terms of use merely by browsing the site. In this last category, the courts reach different results depending upon whether the user had adequate notice of the terms of use and whether those terms of use were reasonable.

Cross-references

▶ Copyright Issues in Databases
▶ European Law in Databases

Recommended Reading

1. Association of Research Libraries, Principles for Licensing Electronic Resources at http://www.arl.org/sc/licensing/licprinciples.shtml.
2. Bowers V. Baystate Techs., 320:F.3d 1317, 1320, Fed. Cir. 2003.
3. Greater Western Library Alliance Guidelines for Licensing Electronic Information Resources at http://www.gwla.org/reports/licensing.html.
4. Mark A. Lemley. Terms of use. Univ. of Minnesota Law Rev., 91:459–483, 2006.
5. ProCD V. Zeidenberg, 86:F.3d 1447, 7th Cir. 1996.

Lifespan

CHRISTIAN S. JENSEN
Aalborg University, Aalborg, Denmark

Synonyms

Existence time; Temporal domain

Definition

The *lifespan* of a database object is the time during which the corresponding real-world object exists in the modeled reality.

Key Points

Some temporal data models, e.g., conceptual models, provide built-in support for the capture of lifespans, while other models do not. It may be observed that lifespans can be reduced to valid times, in the sense that the lifespan of an object *o* is the valid time of the fact "*o* exists."

In the general case, the lifespan of an object is an arbitrary subset of the time domain and is naturally captured by a temporal element timestamp.

The synonym "existence time" is also used for this concept.

Cross-references

▶ Temporal Conceptual Models
▶ Temporal Database
▶ Temporal Element
▶ Time Domain
▶ Time in Philosophical Logic
▶ Valid Time

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.), Böhlen M., Clifford J., Elmasri R., Gadia S. K., Grandi F., Hayes P., Jajodia S., Käfer W., Kline N., Lorentzos N., Mitsopoulos Y., Montanari A., Nonen D., Peressi E., Pernici B., Roddick J.F., Sarda N.L., Scalas M.R., Segev A., Snodgrass R.T., Soo M.D., Tansel A., Tiberio R., and Wiederhold G., A consensus glossary of temporal database concepts – February 1998 Version, in Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, and S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin Heidelberg New York, 1998, pp. 367–405.

Life-span (in Part)

► Context

Lightweight Ontologies

Fausto Giunchiglia, Ilya Zaihrayeu
University of Trento, Trento, Italy

Synonyms

Controlled vocabularies; Taxonomies; Thesauri; Business catalogues; Faceted classifications; Web directories; Topic hierarchies; User classifications

Definition

Ontologies are *explicit specifications of conceptualizations* [7]. They are often thought of as directed graphs whose nodes represent *concepts* and whose edges represent *relations* between concepts. The notion of a concept is understood as defined in Knowledge Representation, i.e., as a set of objects or individuals [2]. This set is called the *concept extension* or the *concept interpretation*. Concepts are often *lexically defined*, i.e., they have natural language names which are used to describe the concept extensions (e.g., concept `mother` denotes the set of all female parents). Therefore, when ontologies are visualized, their nodes are often shown with corresponding natural language concept names. The backbone structure of the ontology graph is a taxonomy in which the relations are "is-a," whereas the remaining structure of the graph supplies auxiliary information about the modeled domain and may include relations like "part-of," "located-in," "is-parent-of," and many others [8].

In their simplest version, one can think of *lightweight ontologies* as ontologies consisting of backbone taxonomies only. In this entry, the "is-a" relationship is generalized to concept subsumption which still allows it to respect the intrinsic properties of backbone taxonomies: namely, in a lightweight ontology, the extension of the concept of a child node is a subset of the extension of the concept of the parent node. Formally, the notion of lightweight ontology is defined as:

A *(formal) lightweight ontology* is a triple $O = \langle N, E, C\rangle$, where N is a finite set of nodes, E is a set of edges on N, such that $\langle N, E\rangle$ is a rooted tree, and C is a finite set of *concepts* expressed in a *formal language* F, such that for any node $n_i \in N$, there is one and only one concept $c_i \in C$, and, if n_i is the parent node for n_j, then $c_j \sqsubseteq c_i$.

The formal language F, used to encode concepts in C, belongs to the family of Description Logic (DL) languages [2] and it may differ in its expressive power and reasoning capability. However, the less expressive one with still useful reasoning capability has shown to be the *propositional DL language*, i.e., a DL language without roles (see [6,3,5] for examples of practical applications of formal lightweight ontologies based on the propositional DL language).

Taxonomies (e.g., NCBI (See http://www.ncbi.nlm.nih.gov/taxonomy)), thesauri (e.g., GLIN (See http://www.loc.gov/lexico/servlet/lexico.)), business catalogues (e.g., UNSPSC (See http://www.unspsc.org.)), faceted classifications (e.g., Flamenco (See http://flamenco.berkeley.edu.)), web directories (e.g., Yahoo! (See http://www.yahoo.com.)), and user classifications are examples of informal prototypes of formal lightweight ontologies. Hereinafter, they are referred to as *(informal) lightweight ontologies*. Note that lightweight ontologies are easier to understand and construct. In fact, as shown in [15], formal lightweight ontologies can be automatically constructed from user classifications as a by-product of a normal computer use, whereas designing a full-fledged ontology (expressed, for example, in OWL-DL (See http://www.w3.org/TR/owl-features.)) is a difficult and error-prone task even for experienced users [13].

Historical Background

The notion of ontology was borrowed from philosophy and adopted in Computer Science under multiple definitions, where, probably, the first credible and most commonly quoted one is "an explicit specification of a conceptualization" [7]. As ontology research evolved, new definitions and examples of what can be considered to be an ontology started to appear. It is typical to characterize ontologies based on the degree of formality and expressivity of the language used to describe them. This characteristic form a continuum of ontology types (see Fig. 1), starting from terms and web directories, and continuing to rigorously formalized logical theories [14]. However, most specifications agree that an ontology should be defined in a formal language, which, in practice, usually means a logic-based language suitable for automating reasoning.

For a long time, lightweight ontologies were not formally defined but were referred to by examples. For instance, they were referred to as terms, as controlled vocabularies, as thesauri, and as web directories like Yahoo! [14]. As observed, informal lightweight ontologies largely cover the spectrum of informal ontology types shown in Fig. 1. In some other approaches, lightweight ontologies are formal ontologies which use a computationally inexpensive logic language (e.g., see [3,10,11]). In practice, these ontologies often encode a hierarchy of classes which can be (automatically or semi-automatically) derived from web directories like Yahoo! (as in [3,11]) or from more strictly defined but still informal structures such as thesauri and taxonomies (as in [10]).

The first attempt to formally define the notion of lightweight ontology as a type of formal ontology was made in [3]. Here, the definition was restricted to formal web directories, whereas in the present document it has been generalized to provide a formal model for a larger spectrum of informal lightweight ontologies, such as controlled vocabularies, taxonomies, thesauri, business catalogues, faceted classifications, web directories, and user classifications.

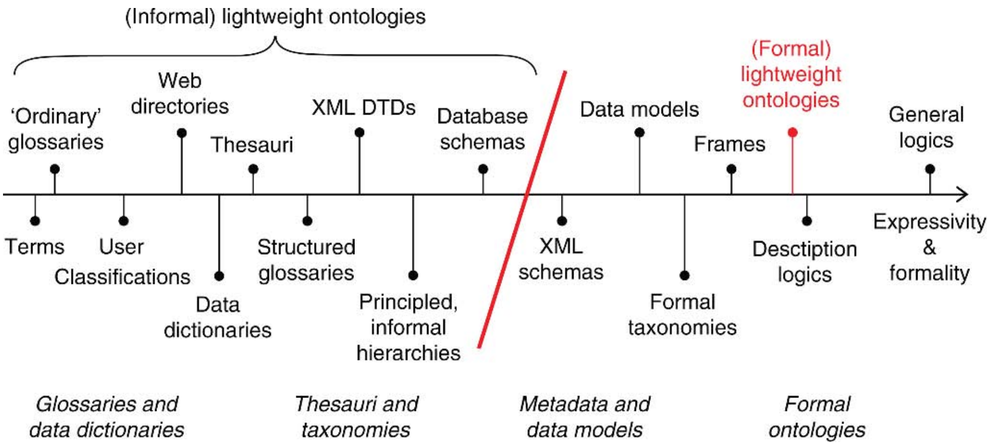
Foundations

This section discusses the main types of lightweight ontologies and their properties and proposes how formal lightweight ontologies can be generated from their informal prototypes.

Types of Lightweight Ontologies

Lightweight ontologies fall into two main types based on their usage: *descriptive* and *classification* lightweight ontologies. Descriptive lightweight ontologies are primarily used for defining the meaning of terms as well as the nature and structure of a domain [9]. Classification lightweight ontologies are primarily used for describing, classifying, and accessing (large) collections of documents or, more generally, data items [9,3]. Due to this difference, formal classification lightweight ontologies have a different domain of interpretation for their concepts. Namely, the extension of a concept in a formal classification lightweight ontology is the *set of documents about* the objects or individuals referred to by the (lexically defined) concept. For example, the extension of concept `mother` is the set of documents about female parents.

Any descriptive lightweight ontology can be used as a classification lightweight ontology, but not vice versa. The two types differ in some principal properties, which are summarized in Table 1 and discussed as follows. Classification lightweight ontologies are usually more complex than descriptive ones, whereas



Lightweight Ontologies. Figure 1. Types of ontologies. Adopted from [14].

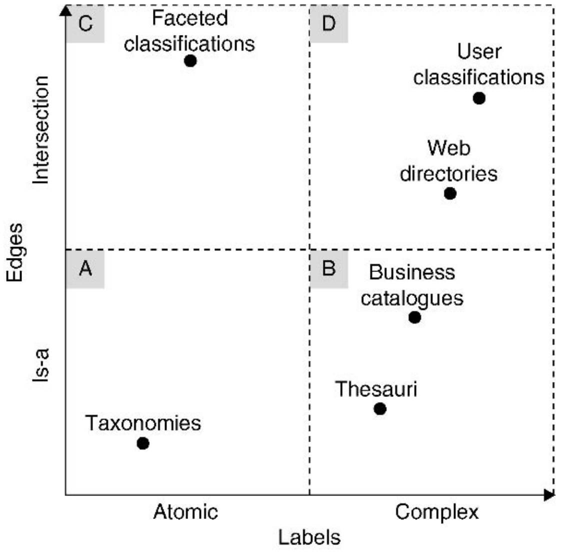
Lightweight Ontologies. Table 1. A classification of lightweight ontologies

	Descriptive lightweight ontologies	Classification lightweight ontologies
	Informal	
Primary use	Defining the meaning of terms as well as the nature and structure of a domain	Describing, classifying, and accessing (large) collections of documents or, more generally, data items
Labels	Single nouns or simple noun phrases denoting atomic concepts as the most typical case	Often, compound noun phrases denoting complex concepts
Edge relations	"is-a" relation	"is-a," "part-of," or, more generally, "intersection" relation
Examples	Taxonomies (e.g., NCBI), thesauri (e.g., GLIN)	Business catalogues (e.g., UNSPSC), faceted classifications (e.g., Flamenco), web directories (e.g., Yahoo!), user classifications
	Formal	
Concept extension	Set of objects or individuals belonging to the class denoted by the concept	Set of documents about the objects or individuals belonging to the class denoted by the concept
Node concepts	Atomic concepts	Atomic and complex concepts

the complexity is defined along two dimensions: *label complexity* (atomic vs. complex labels) and *edge complexity* ("is-a" vs. "intersection" edges). The dimensions of complexity are illustrated in Fig. 2 and discussed in the following.

Category A: atomic labels and "is-a" edges. Ontology labels in this category usually represent single atomic concepts (e.g., "theft," "cellular organisms") and relations between labels are usually "is-a" relations (e.g., "pens" is a child of "writing materials"). Typical examples of this category are (biological) taxonomies such as NCBI. Ontologies in this category are mainly descriptive.

Category B: complex labels and "is-a" edges. Ontology labels in this category can be compound noun phrases which represent complex concepts (e.g., "Open Source and Linux in Education," "Pressure groups representation or participation services") and relations between labels are usually "is-a" relations. Typical examples of this category are thesauri such as GLIN and business catalogues such as UNSPSC. A higher complexity of labels (with regard to category A) in these domains is conditioned by the need for richer descriptions of indexing terms in thesauri and of e-commerce items in business catalogues. Most ontologies in this category are descriptive but some can be classification as well. For instance, the business catalogue UNSPSC can be used as a descriptive ontology or as a classification ontology in which e-commerce items are classified. Note that even if the labels can be complex, they are still mapped to atomic concepts in formal descriptive lightweight ontologies. In classification lightweight ontologies, complex labels represent a dimension of *power of classification* as one label can describe one complex concept that identifies a (very) specific set of documents. Moreover, complex labels can be mapped into complex concepts in formal classification ontologies, which allows for higher modularity in concept definitions. For instance, concept `baby_pictures` can be defined as the intersection



Lightweight Ontologies. Figure 2. Types of labels and edges in lightweight ontologies.

of two concepts, `baby` and `picture`, whereas the interpretation of the former concept is the set of documents about babies (including pictures of babies as a kind of documents) and the interpretation of the latter concept is the set of pictures (including baby pictures). Note that in formal descriptive lightweight ontologies, the extension of concept `baby_pictures` (the set of baby pictures in the world) cannot be expressed as a function of the extension of concept `baby` (the set of babies in the world) and the extension of concept `picture` (the set of pictures in the world).

Category C: atomic labels and "intersection" edges. Ontology labels in this category usually represent single atomic concepts, and relations between labels are usually "intersection" relations, which means that the label of a parent node *specifies* the meaning of the label of its child node. For example, parent node "Italy" specifies the meaning of its child node "Vacation" to the meaning "Vacation in Italy." A typical example of this category is a faceted classification such as Flamenco, in which child nodes represent aspects or facets of their parent nodes along atomic orthogonal dimensions (e.g., time, space, function, material, etc). All ontologies in this category are classification ontologies, for which the "intersection" relation creates an additional dimension of power of classification by allowing it to describe a specific set of documents by means of a set of categories in the ontology. Note that the interpretation domain of formal classification ontologies allows it to treat edges as the *intersection* of parent and child concepts and, therefore, compute concepts of nodes given their position in the ontology tree. For example, the intersection of root concept `italy` with its child concept `vacation` results in a concept whose extension is the set of documents about vacations in Italy, which is the actual meaning of the child node, given its position in the tree.

Category D: complex labels and "intersection" edges. Ontology labels in this category usually represent complex concepts, and relations between labels are usually "intersection" relations. All ontologies in this category are classification ontologies for which the combination of complex labels and "intersection" edges creates the maximum classification power. Labels in this category can represent names of individuals. These labels are mapped to concepts whose extension is the set of documents about the individuals (e.g., the extension of concept `moscow` is the set of documents about the city Moscow). Typical examples of this category are web directories like Yahoo! (in which web pages are classified) and user classifications (in which email messages, favorites, and files are classified). Note that user classifications may have more complex labels and more "intersection" relations than web directories due to the fact that there are basically no rules and restrictions for user classifications that are commonly followed in web directories.

Note that the propositional DL is sufficient for representing formal descriptive lightweight ontologies in categories A and B, as the only thing that needs to be encoded is the subsumption hierarchy of atomic classes. It is also sufficient for representing formal classification lightweight ontologies in categories A and C, since the only relations that need to be represented are subsumption and intersection of atomic classes. The propositional DL is capable of capturing the semantics of labels in classification lightweight ontologies in categories B and D to a significant extent, while approximating the meaning of labels in some cases, as it is discussed in the following section.

From Informal to Formal Lightweight Ontologies

Formal descriptive lightweight ontologies can be generated from informal ones by transforming their organizational structure into a rooted tree (where necessary) and by converting their term labels, expressed in natural language, into concepts in the formal language *F*. The generation of formal classification lightweight ontologies requires an extra step where node concepts are computed as the intersection of the concepts corresponding to the term labels on the path to the root node [3]. Note that the rooted tree structure of formal lightweight ontologies allows it to capture the backbone organization of many informal prototypes. In fact, taxonomies, business catalogues, faceted classifications, web directories, and user classifications use rooted trees to organize their categories. In the simplest case, the hierarchy of thesaurus terms, built based on the Broader Term relation, is a rooted tree; and, a controlled vocabulary can be seen as one level rooted tree, where the root node represents the Top concept and its child nodes are the controlled vocabulary terms. The principal steps of the process of converting term labels into concepts in *F* for classification lightweight ontologies are described below. The conversion of labels of descriptive lightweight ontologies follows the same principles even if it is an easier case due to the relative simplicity of their labels. In the following, it is assumed that *F* is the Propositional DL language.

WordNet [12] senses of adjectives and common nouns in the label become atomic concepts in *F*. The extension of a common noun concept is the set of documents about objects of the class, denoted by the noun; and, the extension of an adjective concept is the set of documents about objects, which possess the qualities, denoted by the adjective. Proper names become atomic concepts as well, whose extension is the set of documents about the individual referenced by the proper name. Notationally, adjective and common noun atomic concepts are represented in the following syntax: *lemma-pos-sn*, where *lemma* is the lemma of the word, *pos* is its part of speech, and *sn* is the sense number in WordNet [12]. Proper name atomic concepts are tagged with $_{NNP}$.

Syntactic relations between words in the label are translated into logical connectives of *F* in order to build complex concepts from atomic concepts. For example, a set of adjectives followed by a noun group is translated into the logical conjunction (⊓) of the concepts corresponding to the adjectives and to the nouns; prepositions like "of" and "in" are translated into the conjunction; coordinating conjunctions "and" and "or" are translated into the logical disjunction (⊔). The final formula for the label is built following these rules and taking into account how words are coordinated in the label.

Consider a relatively complex label: "*Bank and personal details of George Bush*." Its correct translation to *F* produces the following concept:

- ► (bank-noun-1 ⊔ personal-adj-1) ⊓ detail-noun-1 ⊓ george_bush NNP

The extension of the concept above is the intersection of three sets of documents: (1) documents about the President George W. Bush, (2) documents containing isolated facts about something (i.e., details), and (3) the union of documents about bank institutions and documents concerning a particular person or his/her private life. Note that the extension comprises (all and only) documents one would classify under the above mentioned label.

Despite its seeming simplicity, the translation process is subject to various mistakes originating from incorrect natural language processing (NLP). (How the NLP problems, described in this paragraph, can be solved is beyond the scope of this document. Interested readers are referred to [15] for a first account.) For instance, due to a mistake in part-of-speech tagging, the word *personal* might be recognized as a noun, which has only one sense in WordNet defined as "*a short newspaper article about a particular person or group*." Due to a mistake in word sense disambiguation, the sense of the word *bank* might be identified as "*sloping land (especially the slope beside a body of water)*." Due to a mistake in named entity detection, the proper name *George Bush* might not be detected and might then be considered as two distinct nouns, where the noun *bush* means "*a low woody perennial plant usually having several major branches*." Finally, due to a mistake in (syntax) parsing, the input label might be erroneously translated into:

- ► bank-noun-1 ⊔ personal-adj-1 ⊓ detail-noun-1 ⊓ george_bush NP

a concept, whose extension is the union of documents about bank institutions and documents discussing personal details of the President George W. Bush.

Note that the propositional DL can capture the semantics of complex labels to a significant extent only when these labels are built from noun phrases possibly connected through coordinating conjunctions such as "and" and "or" (e.g., "Big city life and civil protection"). It approximates the meaning of some other types of labels, e.g., labels with prepositions. For instance, labels "life in war" and "life after war" will collapse into the same formula, which approximates the meaning of both labels.

Key Applications

Formal (descriptive and classification) lightweight ontologies can be used in various domains, such as document classification (e.g., see [6]), semantic search (e.g., see [3]), and data integration (e.g., see [5]). In the following subsections these three application domains are briefly discussed.

Document Classification

Document classification is the problem of assigning a document to one or more categories based on the document contents. In the context of lightweight ontologies, document classification refers to assigning a document to: (1) controlled vocabulary terms; (2) categories in taxonomies, business catalogues, faceted classifications, web directories, or user classifications. The approach reported in [6] presents fully automatic classification of documents into web directories based on the *get-specific* document classification algorithm.

The underlying idea is that a web directory is converted into a formal lightweight ontology, that a document is assigned a concept, and that the document classification problem is then reduced to reasoning about subsumption on the formal lightweight ontology. Note that this classification approach does not require the creation of a training dataset which would normally be required in machine learning approaches [6].

Semantic Search

In the context of lightweight ontologies, semantic search is the problem of finding categories and/or documents (when applicable) classified in categories of (informal) lightweight ontologies, such that the found objects *semantically* correspond to a provided natural language query. Loosely speaking, semantic correspondence of an object to a query indicates that the meaning associated with the object is more specific or equivalent to the meaning given to the query under common sense interpretation. For instance, document about *Ethiopian villages* semantically corresponds to a query about *African settlements*. The approach reported in [3] formalizes the above informal description and introduces a semantic search algorithm for lightweight classification ontologies populated with documents. The underlying idea is that the user query is converted to a concept in the way presented earlier in this document and that the answer to the query is computed as the set of documents whose concepts are more specific or equivalent to the concept of the query. In order to reduce the computation complexity, the query is first run on the structure of the corresponding formal lightweight ontology in order to identify the scope of relevant nodes and then it is run on the documents populated in some of the nodes from the scope.

Data Integration

Data integration is the process of combining data residing at different sources and providing the user with a unified view of these data. Often, a data source can be represented in the form of a rooted tree, whose nodes are assigned natural language labels, and, in this case, data integration can be facilitated by discovering semantic relations which exist between nodes of the source trees [5]. A semantic relation between two nodes can be more/less general, equivalent, or disjoint. In the domain of lightweight ontologies, semantic relations can be found between elements of controlled vocabularies, taxonomies, thesauri, business catalogues, faceted classifications, web directories, and user classifications. Such relations can then be used for enabling integration or inter-operation of web directories, for merging business catalogues, and so on.

Future Directions

There are two major problems related to formal lightweight ontologies which drive future directions of research in this area. The problems are:

- *Natural language processing.* Since formal lightweight ontologies are supposed to be often generated from their informal prototypes, the quality of these ontologies strongly depends of the correctness and completeness of NLP procedures involved in the conversion process. Note that NLP for informal lightweight ontologies is a potentially new domain in the NLP research due to the particular characteristics of term labels (e.g., they are usually short noun phrases with little context) [15];
- *Lack of background knowledge.* Reasoning on formal lightweight ontologies, which is used, for example, in document classification, in semantic search, and in data integration as discussed above, strongly depends on the set of axioms which must be known a priori [3]. These axioms are extracted from a knowledge base such as WordNet [12]. It was shown that lack of background knowledge is the main source of a relatively low recall in reasoning-based tasks on formal lightweight ontologies [4].

Experimental Results

First evaluation studies of document classification show that re-classification of 1217 HTML pages into a part of the DMoz hierarchy, (See http://www.dmoz.org.) that has 157 nodes distributed in a tree of depth 6, allows it to reach 41% in the micro-averaged F1 measure [6]. Document concepts in the conducted experiments were built by computing the conjunction of the formulas corresponding to the first ten most frequent words appearing in the documents (excluding stop words).

The performance of S-Match, a tool that, among other things, facilitates the integration of lightweight ontologies, reaches up to 65% in recall in some data sets [1].

Data Sets

Some informal lightweight ontologies are available for download in the form of data files. These include the

DMoz web directory, (See http://rdf.dmoz.org.) business catalogues UNSPSC and eCl@ss, (See http://www.eclassdownload.com.) NCBI taxonomy, and many others.

Cross-references

▶ I(a) – Database Fundamentals: Data Models (including semantic data models)
▶ Data Integration

Recommended Reading

1. Avesani P., Giunchiglia F., and Yatskevich M. A Large Scale Taxonomy Mapping Evaluation. In Proc. 4th Int. Semantic Web Conference, 2005, pp. 67–81.
2. Baader F., Calvanese D., McGuinness D., Nardi D., and Patel-Schneider P. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003.
3. Giunchiglia F.M.M. and Zaihrayeu I. Encoding classifications into lightweight ontologies. J. Data Semant., VIII:57–81, 2007.
4. Giunchiglia F., Shvaiko P., and Yatskevich M. Discovering Missing Background Knowledge in Ontology Matching. In Proc. 17th European Conf. on Artificial Intelligence, 2006, pp. 382–386.
5. Giunchiglia F., Yatskevich M., and Shvaiko P. Semantic Matching: Algorithms and Implementation. J. Data Semant., IX, 2007.
6. Giunchiglia F., Zaihrayeu I., and Kharkevich U. Formalizing the Get-Specific Document Classification Algorithm. In 11th European Conf. on Research and Advanced Technology for Digital Libraries, 2007.
7. Gruber T.R. A translation approach to portable ontology specifications. Knowl. Acquis., 5(2):199–220, 1993.
8. Guarino N. Some Ontological Principles for Designing Upper Level Lexical Resources. In Proc. 1st Int. Conf. Lexical Resources and Evaluation. vol. 2830, 1998.
9. Guarino N. Helping People (and Machines) Understanding Each Other: The Role of Formal Ontology. In Proc. CoopIS/DOA/ODBASE, 2004, p. 599.
10. Hepp M. and de Bruijn J. GenTax: A Generic Methodology for Deriving OWL and RDF-S Ontologies from Hierarchical Classifications, Thesauri, and Inconsistent Taxonomies. In Proc. 4th European Semantic Web Conference, 2007, pp. 129–144.
11. Magnini B., Serafini L., and Speranza M. Making Explicit the Hidden Semantics of Hierarchical Classifications. In Proc. 8th Congress of the Italian Association for Artificial Intelligence, 2003, pp. 436–448.
12. Miller G. Wordnet: An electronic Lexical Database. MIT Press, Cambridege, 1998.
13. Rector A.L., Drummond N., Horridge M., Rogers J., Knublauch H., Stevens R., Wang H., and Wroe C. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In Proc. 4th Int. Conf. Knowledge Eng. and Knowledge Management: Ontologies and the Semantic Web, 2004, pp. 63–81.
14. Uschold M. and Gruninger M. Ontologies and semantics for seamless connectivity. ACM SIGMOD Rec., 33(4):58–64, 2004.
15. Zaihrayeu I., Sun L., Giunchiglia F., Pan W., Ju Q., Chi M., and Huang X. From Web Directories to Ontologies: Natural Language Processing Challenges. In Proc. 6th International Semantic Web Conference, 2007.

Lineage

▶ Provenance
▶ Provenance in Scientific Databases

Linear Hashing

DONGHUI ZHANG[1], YANNIS MANOLOPOULOS[2],
YANNIS THEODORIDIS[3], VASSILIS J. TSOTRAS[4]
[1]Northeastern University, Boston, MA, USA
[2]Aristotle University, Thessaloniki, Greece
[3]University of Piraeus, Piraeus, Greece
[4]University of California – Riverside, Riverside, CA, USA

Definition

Linear Hashing is a dynamically updateable disk-based index structure which implements a hashing scheme and which grows or shrinks one bucket at a time. The index is used to support exact match queries, i.e., find the record with a given key. Compared with the B+-tree index which also supports exact match queries (in logarithmic number of I/Os), Linear Hashing has better expected query cost O(1) I/O. Compared with Extendible Hashing, Linear Hashing does not use a bucket directory, and when an overflow occurs, it is not always the overflown bucket that is split. The name Linear Hashing is used because the number of buckets grows or shrinks in a linear fashion. Overflows are handled by creating a chain of pages under the overflown bucket. The hashing function changes dynamically and at any given instant there can be at most two hashing functions used by the scheme.

Historical Background

A hash table is an in-memory data structure that associates keys with values. The primary operation it supports efficiently is a lookup: given a key, find the corresponding value. It works by transforming the key using a hash function into a hash, a number that is used as an index in an array to locate the desired location where the values should be. Multiple keys may be hashed to the same bucket, and all keys in a bucket should be searched upon a query. Hash tables are often used to implement associative arrays, sets and caches. Like arrays, hash tables have O(1) lookup cost on average.

Foundations

The Linear Hashing scheme was introduced by [2].

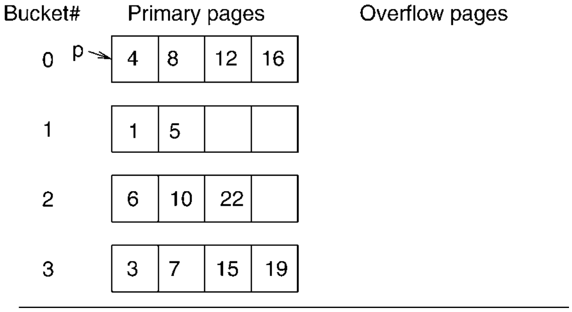
Initial Layout

The Linear Hashing scheme has m initial buckets labeled 0 through $m - 1$, and an initial hashing function $h_0(k) = f(k)\ \%\ \ m$ that is used to map any key k into one of the m buckets (for simplicity assume $h_0(k) = k\ \%\ m$), and a pointer p which points to the bucket to be split next whenever an overflow page is generated (initially $p = 0$). An example is shown in Fig. 1.

Bucket Split

When the first overflow occurs (it can occur in any bucket), bucket 0, which is pointed by p, is split (rehashed) into two buckets: the original bucket 0 and a new bucket m. A new empty page is also added in the overflown bucket to accommodate the overflow. The search values originally mapped into bucket 0 (using function h_0) are now distributed between buckets 0 and m using a new hashing function h_1.

As an example, Fig. 2 shows the layout of the Linear Hashing of Fig. 1 after inserting a new record with key 11. The circled records are the existing records that are moved to the new bucket. In more detail, the record is inserted into bucket $11\%4 = 3$. The bucket overflows and an overflow page is introduced to accommodate the new record. Bucket 0 is split and the records originally in bucket 0 are distributed between bucket 0 and bucket 4, using a new hash function $h_1(k) = k\ \%\ 8$.



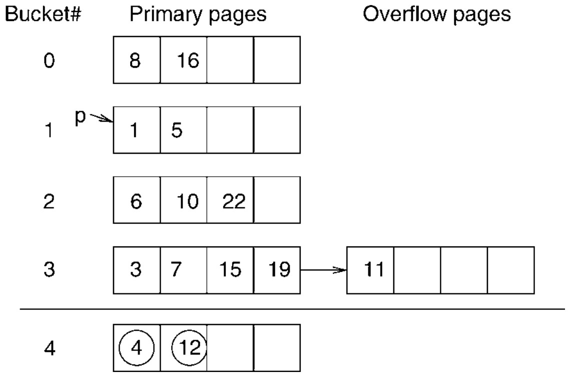
Linear Hashing. Figure 1. An initial Linear Hashing. Here $m = 4$, $p = 0$, $h_0(k) = k\ \%\ 4$.

The next bucket overflow, such as triggered by inserting two records in bucket 2 or four records in bucket 3 in Fig. 2, will cause a new split that will attach a new bucket $m + 1$ and the contents of bucket 1 will be distributed using h_1 between buckets 1 and $m + 1$. A crucial property of h_1 is that search values that were originally mapped by h_0 to some bucket j must be remapped either to bucket j or bucket $j + m$. This is a necessary property for Linear Hashing to work. An example of such hashing function is: $h_1(k) = k\ \%\ 2m$. Further bucket overflows will cause additional bucket splits in a linear bucket-number order (increasing p by one for every split).

Round and Hash Function Advancement

After enough overflows, all original m buckets will be split. This marks the end of splitting-round 0. During round 0, p went subsequently from bucket 0 to bucket $m - 1$. At the end of round 0 the Linear Hashing scheme has a total of $2m$ buckets. Hashing function h_0 is no longer needed as all $2m$ buckets can be addressed by hashing function h_1. Variable p is reset to 0 and a new round, namely splitting-round 1, starts. A new hash function h_2 will start to be used.

In general, the Linear Hashing scheme involves a family of hash functions h_0, h_1, h_2, and so on. Let the



Linear Hashing. Figure 2. The Linear Hashing after inserting 11 into Fig. 1. Here $p = 1$, $h_0(k) = k\ \%\ 4$, $h_1(k) = k\ \%\ 8$.

initial function be $h_0(k) = f(k)\ \%\ m$, then any later hash function $h_i(k) = f(k)\ \%\ 2^i m$. This way, it is guaranteed that if h_i hashes a key to bucket $j \in [0..2^i m - 1]$, h_{i+1} will hash the same key to either bucket j or bucket $j + 2^i m$. At any time, two hash functions h_i and h_{i+1} are used.

Figures 3 and 4 illustrates the cases at the end of splitting-round 0 and at the beginning of splitting-round 1. In general, in splitting round i, the hash functions h_i and h_{i+1} are used. At the beginning of round i, $p = 0$ and there are $2^i m$ buckets. When all of these buckets are split, splitting round $i + 1$ starts. p goes back to 0. The number of buckets becomes $2^{i+1} m$. And hash functions h_{i+1} and h_{i+2} will start to be used.

Component Summary and Search Scheme

In summary, at any time a Linear Hashing scheme has the following components:

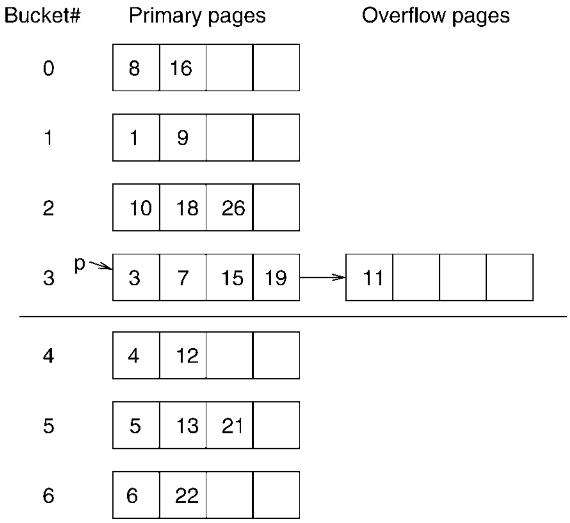
- A value i which indicates the current splitting round.
- A variable $p \in [0..2^i m - 1]$ which indicates the bucket to be split next.
- A total of $2^i m + p$ buckets, each of which consists of a primary page and possibly some overflow pages.
- Two hash functions h_i and h_{i+1}.

A search scheme is needed to map a key k to a bucket, either when searching for an existing record or when inserting a new record. The search scheme works as follows:

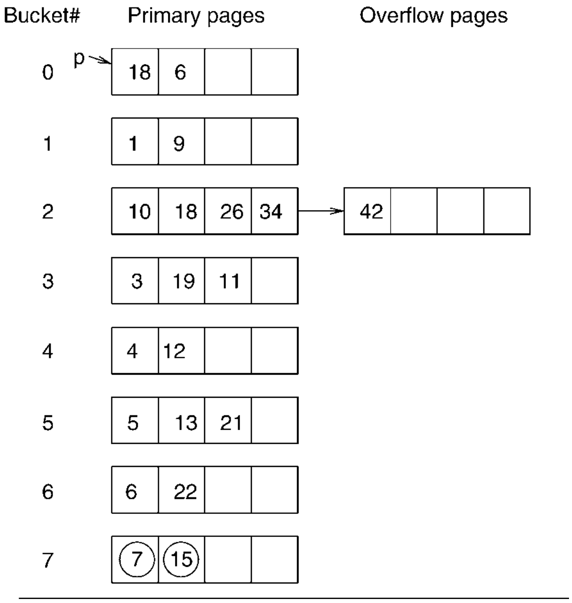
(a) If $h_i(k) \geq p$, choose bucket $h_i(k)$ since the bucket has not been split yet in the current round.

(b) If $h_i(k) < p$, choose bucket $h_{i+1}(k)$, which can be either $h_i(k)$ or its spit image $h_i(k) + 2^i m$.

For example, in Fig. 2, $p = 1$. To search for record 5, since $h_0(5) = 1 \geq p$, one directly goes to bucket to find the record. But to search for record 4, since $h_0(4) = 0 < p$, one needs to use h_1 to decide the actual bucket. In this case, the record should be searched in bucket $h_1(4) = 4$.



Linear Hashing. Figure 3. The Linear Hashing at the end of round 0. Here $p = 3$, $h_0(k) = k\ \%\ m$, $h_1(k) = k\ \%\ 2^1 m$.



Linear Hashing. Figure 4. The Linear Hashing at the beginning of round 1. Here $p = 0$, $h_1(k) = k\ \%\ 2^1 m$, $h_2(k) = k\ \%\ 2^2 m$.

Variations

A split performed whenever a bucket overflow occurs is an uncontrolled split. Let l denote the Linear Hashing scheme's load factor, i.e., $l = S/b$ where S is the total number of records and b is the number of buckets used. The load factor achieved by uncontrolled splits is usually between 50–70%, depending on the page size and the search value distribution [2]. In practice, higher storage utilization is achieved if a split is triggered not by an overflow, but when the load factor l becomes greater than some upper threshold. This is called a controlled split and can typically achieve 95% utilization. Other controlled schemes exist where a split is delayed until both the threshold condition holds and an overflow occurs.

Deletions will cause the hashing scheme to shrink. Buckets that have been split can be recombined if the load factor falls below some lower threshold. Then two buckets are merged together; this operation is the reverse of splitting and occurs in reverse linear order. Practical values for the lower and upper thresholds are 0.7 and 0.9 respectively.

Linear Hashing has been further investigated in an effort to design more efficient variations. In [3] a performance comparison study of four Linear Hashing variations is reported.

Key Applications

Linear Hashing has been implemented into commercial database systems. It is used in applications where exact match query is the most important query such as hash join [4]. It has been adopted in the Icon language [1].

Cross-references

▶ Extendible Hashing
▶ Hashing
▶ Hash-based Indexing

Recommended Reading

1. Griswold W.G. and Townsend G.M. The design and implementation of Dynamic Hashing for sets and tables in icon. Software Pract. Ex., 23(4):351–367, 1993.
2. Litwin W. Linear Hashing: a new tool for file and table addressing. In Proc. of the Sixth International Conference on Very Large Databases, 1980, pp. 212–223.
3. Manolopoulos Y. and Lorentzos N. Performance of Linear Hashing schemes for primary key retrieval. Inf. Syst., 19(5):433–446, 1994.
4. Schneider D.A. and DeWitt D.J. Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In Proc. 16th Int. Conf. on Very Large Databases, 1990, pp. 469–480.

Linear Regression

Jialie Shen
Singapore Management University, Singapore, Singapore

Definition

Linear regression is a classical statistical tool that models relationship between a dependent variable or regress and Y, explanatory variable or regressor $X = \{x_1,..,x_I\}$ and a random term ε by fitting a linear function,

$$Y = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_I x_I + \varepsilon$$

where α_0 is the constant term, the α_is are the respective parameters of independent variable, and I is the number of parameters to be estimated in the linear regression.

Key Points

Linear regression analysis is an important component for several tasks such as clustering, time series analysis, and information retrieval. For instance, it is a very powerful forecasting method for time series data. It helps identify the long-term movement of a certain data set based on given information and explore the dependent variable as function of time.

To solve the linear regression problem, there are various kinds of approaches available to determine suitable regression coefficients [1,2]. They include: (i) least-squares analysis, (ii) assessing the least-squares model, (iii) modifications of least-squares analysis, and (iiii) polynomial fitting. The primary objective is to select a straight line which minimizes the error between the real data and the line estimated to provide a best fit.

Cross-references

▶ Least Squares
▶ Numeric Prediction
▶ Regression

Recommended Reading

1. Draper, N.R., Smith, H. Applied Regression Analysis Wiley Series in Probability and Statistics, 1998.
2. Gross, J. Linear Regression, Springer, Berlin, 2003.

Linearization

▶ Space Filling Curves
▶ Space-Filling Curves for Query Processing

Link Analysis

▶ Web Page Quality Metrics

Link Database

▶ Graph Database

Linked Brushing

▶ Linking and Brushing

Linked Views

▶ Linking and Brushing

Linking and Brushing

Matthew O. Ward
Worcester Polytechnic Institute, Worcester, MA, USA

Synonyms

Linked brushing; Linked views

Definition

Within the context of visual data exploration, *Linking* refers to the process in which user interactions in one display of a multi-display system are applied to some or all other displays. In this same context, *brushing* consists of the interactive selection of a subset of the displayed data by either dragging the mouse over the data of interest or using a bounding shape to isolate this subset. Together, linked brushing is one of the most powerful interactive tools for doing exploratory data analysis using visualization.

Historical Background

Perhaps the earliest reference to linked brushing was by McDonald [10] as a mechanism for cross-referencing between multiple plots. The term *brushing* was introduced in 1978 by Newton [11], who defined it as an interactive method for painting a group of points with a square, circular, or polygonal brush. Since then, researchers have expanded on these concepts, as described in the next section.

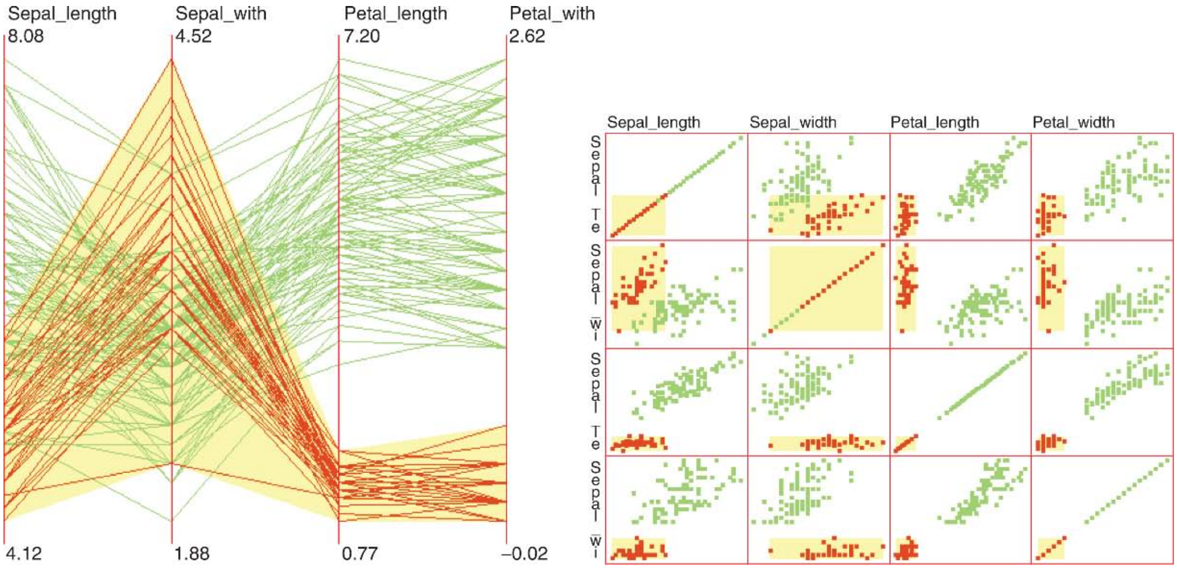
Foundations

Many operations within the process of visual data analysis begin with the process of *Selection*, in which a subset of the data being analyzed is isolated. Once isolated, the subset may be highlighted, deleted, masked, aggregated, or otherwise subjected to some operation. Many methods for performing interactive selection have been developed, including clicking on items in a list, specifying a range query for one or more data variables, typing a set of constraints, and sampling. These methods are sometimes referred to as *indirect*, as their specification is most often done separate from the data visualization. Indeed, most, if not all, can be performed without even viewing the data.

Brushing is a selection method that is specified in a *direct* fashion, i.e., by indicating elements or constraints on the data visualization itself. Many variations on brushing have been proposed over the years, each with its own strengths and weaknesses in terms of ease of use and degree of control a user has on the constitution of the result set. Brushing techniques usually consist of combinations of mouse/cursor motions and button clicks, though more unusual methods, such as using eye/head tracking or gestures in virtual reality environments, have also been proposed.

One key aspect that differentiates many brushing techniques is the *space* in which the interaction is specified. Early brushing implementations, e.g., [1], used screen-space as a constraint. Thus data that mapped to a particular range of pixels in the display were considered selected. Later, brushing was expanded to data-space [9], in which a selection from a multivariate data visualization consisted of a bounding box whose dimensionality matched the number of variables in the data (see Fig. 1). These N-dimensional brushes could be generated either by painting over a subsection of the display or directly manipulating a visual depiction of the boundaries of the N-dimensional hyperbox enveloping the selection. A third space that has been proposed for brushing is structure-space [6], where the structure of a data set, e.g., an ordering, hierarchy, or other organization, is used as a mechanism for data selection. In this case, a visual representation of the structure is provided to the user, and he or she brushes over parts of the structure. Data falling into this part are then considered as selected (see Fig. 2). This type of selection could conceivably be considered both direct and indirect, as the operation is performed on a visualization, but not on the data display itself. Similar to structure-based brushing, other attributes of a data set, such as the uncertainty or quality of the data [13], could be used as a focus for brushing.

Brushing techniques also differ in the manner in which data is isolated. Common methods include bounding boxes, bounding circles, lassos, and arbitrary



Linking and Brushing. Figure 1. Example of linked brushing in data space. Cluster isolated in parallel coordinates, with linked selection in scatterplot matrix. Selected data is dark, brush extents are shown as light bands or rectangles.

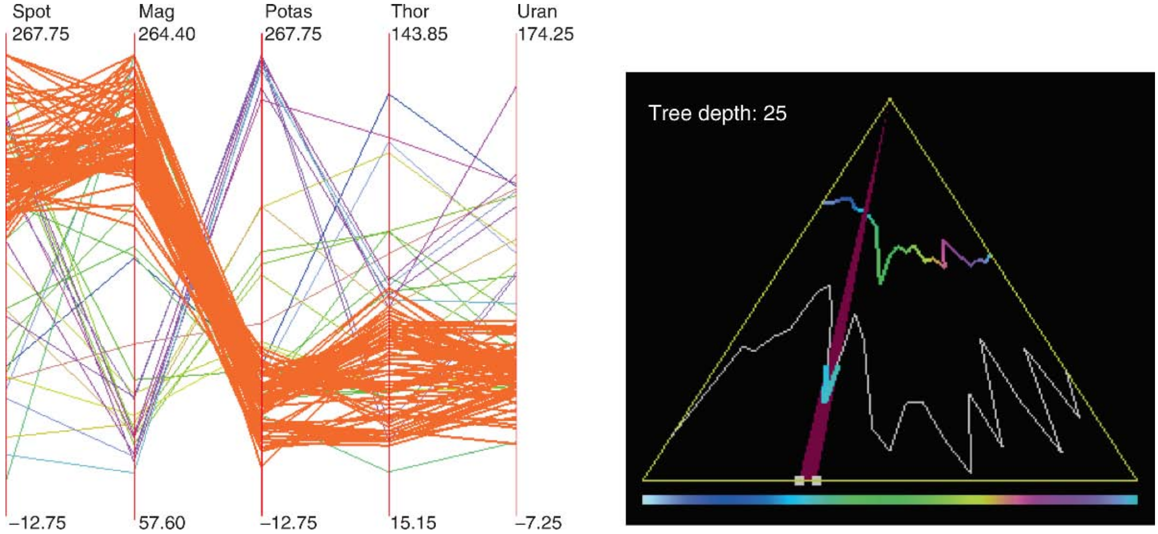
polygons for specifying a boundary around the selection and painting over specific data points. One can even specialize the behavior of a brush based on a specific type of visualization. Hauser et al. [7], for example, use an angled brush on parallel coordinates displays to select only data points whose representation on the display form lines with angles similar to that specified in the brush. In this way, data points that are well correlated with each other between a given pair of dimensions can be readily isolated.

Many approaches have been proposed to increase the richness of the types of selections that are possible. Martin and Ward [9] allow the user to not only control up to four separate brushes, but also allowed selections to be formed by logical combinations of brushes, including unions, intersections, negations, and exclusive or's. Wills [12] expanded on this idea by developing what might be called *incremental* brushes, where by a composite brush could be formed by augmenting an existing selection with additional data and/or constraints. This can result, for example, in selections with nearly arbitrary shapes, including holes.

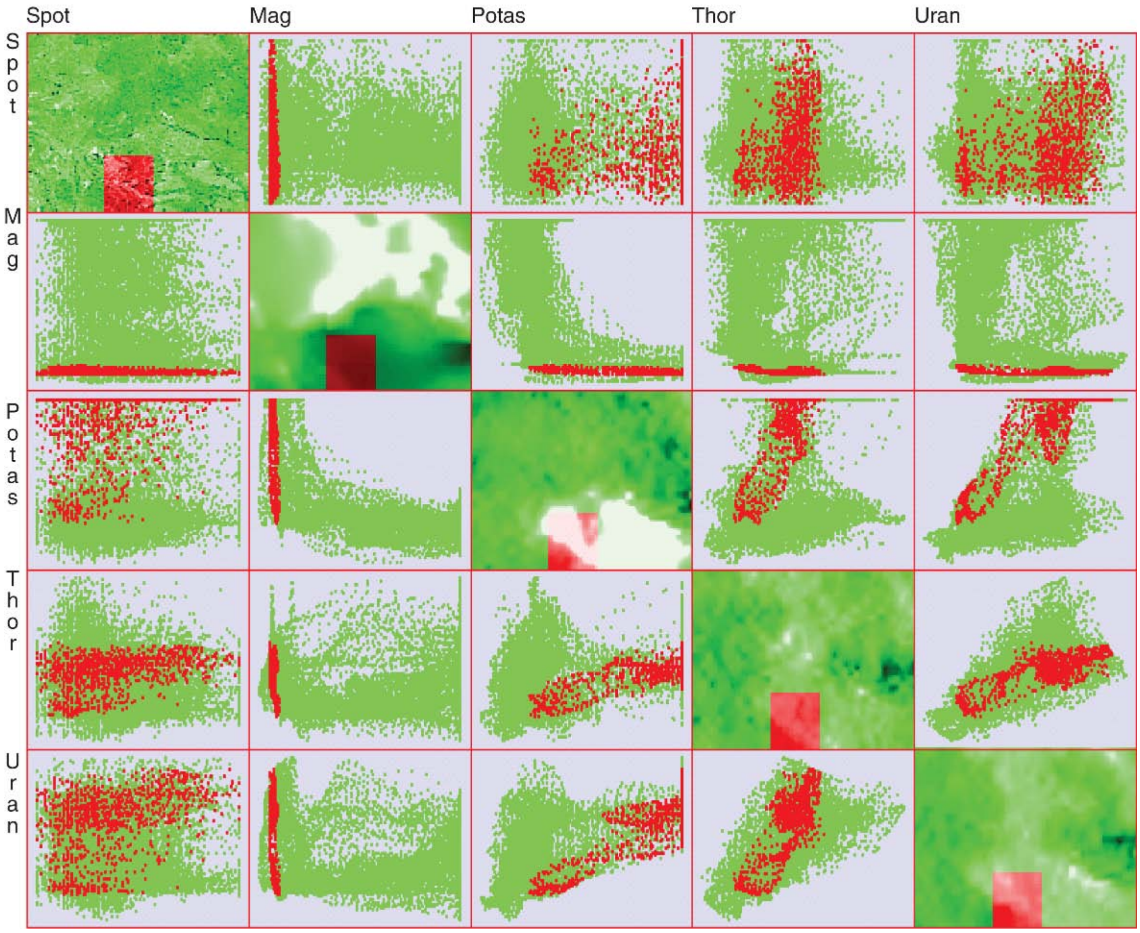
Thus far, the assumption has been that the result of brushing is that every data point in the set is either in the brush or outside the brush. However, some researchers [5,9] have experimented with what might be called *fuzzy* brushes, in which data can have a degree of membership in a brush. Using linear or non-linear functions emanating either from the brush center or from the edge of a plateau, the user can distinguish between points that are entirely covered by a brush versus those near the edge of the brush versus those a significant distance from the brush. This is a crucial functionality in multivariate data brushing, as the concept of distance is distributed across all the dimensions and thus difficult to estimate visually.

A frequent use for brushing is *linking* operations in one view to generate corresponding actions in other views. While other forms of linking between subwindows of an application exist, such as when opening a new data file, linked brushing, especially to highlight corresponding data from several views, is probably the most common form of linking found in modern visualization tools. Its popularity stems in a large part from the fact that each view of one's data can reveal interesting features, and by highlighting such a feature in one view, it is possible to build a more complete mental model of the entity by seeing how it appears in other views. This can also help reveal relationships between this entity and others in the data set. For example, when examining multivariate spatial data, it is often useful to jump between the spatially referenced view and the dependent variable view, which often does not preserve the spatial attributes (see Fig. 3).

Another strength of linked brushing is in specifying complex constraints on one's selection. Each type of view is optimized for both conveying certain types



Linking and Brushing. Figure 2. Example of structure-based brushing. Hierarchy view on right shows shape of cluster tree. Area of interest and its level of detail is isolated in red. Corresponding data highlighted in parallel coordinates view on left, with unselected cluster centers shown at a higher level of the hierarchy.



Linking and Brushing. Figure 3. Example of linking spatial and non-spatial views. Diagonal plots are spatial views of remote sensing data, while non-diagonal plots are parameter views. Characteristics of selected region of spatial views are highlighted in other views.

of information as well as specifying conditions on particular types and with a particular degree of accuracy. Thus, for example, one might specify a temporal constraint using a visualization containing a timeline, a constraint on a name field using a sorted list view, and a geographic constraint using a map. While each is effective as a tool for accurate and intuitive specification of a part of a query, none could be used for the complete query.

In some situations the user may want to *unlink* some visualizations in order to maintain a given view while exploring a different area of the data or different data set. Some systems allow the user to indicate for each window whether it is transmitting information to other views and from which other windows it will receive input. A user may also want to constrain the type of information being communicated, as well as its direction. Some types of interaction may be local to a particular window, e.g., zooming in and out, while others are meant to be shared, such as reordering dimensions. Also, in some situations, such as with hierarchically related windows, it may make more sense for the information to move from parent to child, but not the other way. Thus a fairly rich set of connection and communication options may be needed to maximize flexibility.

Key Applications

Linked brushing had its roots in the field of statistics, but is now commonly employed in any field in which visual data analysis is used, including earth and space science, engineering, homeland security, economics, and bioinformatics.

URL to Code

Many commercial and freeware visualization systems support linked brushing. Two such freeware systems are GGobi (http://www.ggobi.org) and XmdvTool (http://davis.wpi.edu/∼xmdv).

Cross-references

- ▶ Comparative Visualization
- ▶ Data Visualization
- ▶ Multidimensional Visualization
- ▶ Multivariate Visualization Methods
- ▶ Scientific Visualization
- ▶ Text Visualization
- ▶ Visual Data Mining
- ▶ Visual Interfaces
- ▶ Visualizing Hierarchical Data
- ▶ Visualizing Quantitative Data

Recommended Reading

1. Becker R.A. and Cleveland W.S. Brushing scatterplots. Technometrics, 29(2):127–142, 1987.
2. Becker R.A., Cleveland W.S., and Wilks A.R. The use of brushing and rotation for data analysis. In Dynamic Graphics for Statistics, W.S. Cleveland, M.E. McGill (eds.). Wadsworth, Pacific Grove, CA, USA, 1988, pp. 1–50.
3. Chen H. Compound brushing. In Proc. IEEE Symp. Information Visualization, 2003, pp. 181–188.
4. Cook D. and Swayne D.F. Interactive and Dynamic Graphics for Data Analysis with R and GGobi. Springer, New York, 2008.
5. Doleisch H. and Hauser H. Smooth brushing for focus+context visualization of simulation data in 3D. J. WSCG, 10(1):147–155, 2002.
6. Fua Y.-H., Ward M.O., and Rundensteiner E.A. Structure-based brushes: a mechanism for navigating hierarchically organized data and information spaces. IEEE Trans. Vis. Comput. Graph., 6(2):150–159, 2000.
7. Hauser H., Ledermann F., and Doleisch H. Angular brushing of extended parallel coordinates. In Proc. Information Visualization, 2002, pp. 127–130.
8. Henze C. Feature detection in linked derived spaces. In Proc. Conf. Visualization, 1998, pp. 87–94.
9. Martin A.R. and Ward M.O. High dimensional brushing for interactive exploration of multivariate data. In Proc. IEEE Conf. Visualization '95, 1995, pp. 271–278.
10. McDonald J.A. Orion I: interactive graphics for data analysis. Technical report, Stanford University, 1983.
11. Newton C. Graphica: from alpha to omega in data analysis. In Graphical Representation of Multivariate Data, P. Wang (ed.). Academic, New York, 1978, pp. 59–92.
12. Wills G.J. 524,288 ways to say "this is interesting." In Proc. Information Visualization, 1996, pp. 54–60.
13. Xie Z., Ward M.O., Rundensteiner E.A., and Huang S. Integrating data and quality space interactions in exploratory visualizations. In Proc. Int. Conf. on Coordinated and Multiple Views in Exploratory Visualization, 2007, pp. 47–60.

List

- ▶ Table
- ▶ Text Index Compression

List Comprehension

- ▶ Comprehensions

Literature-based Discovery from Biological Resources

► Text Mining of Biological Resources

Load Balancing

► Database Middleware

Load Balancing in Peer-to-Peer Overlay Networks

Anwitaman Datta
Nanyang Technological University, Singapore,
Singapore

Definition

Load balancing in peer-to-peer (P2P) overlay networks is a mechanism to spread various kinds of loads like storage, access and message forwarding among participating peers in order to achieve a fair or optimal utilization of contributed resources such as storage and bandwidth.

Historical Background

Load balancing is a general and critical requirement in distributed and parallel processing systems in order to make efficient and fair use of available resources. In the context of P2P systems, the early works on load-balancing heavily relied on consistent hashing [11], which was proposed in 1997 to originally deal with load-balancing in web caches with minimal movement of data even if new caches are added or if existing ones crash. Consistent hashing was used to achieve storage load-balancing in many early distributed hash table (DHT) P2P networks proposed around 2001.

When a new object is stored, uniform hashing (as used in consistent hashing) helps choosing a peer uniformly from the set of all peers. But the storage load distribution still has high variation as observed in the *balls into bins* phenomenon [14]. The imbalance caused by such variation can be mitigated using an uncoordinated randomized approach called "power of two choices" [13], (More generally, multiple choices.) and was employed for P2P overlays [5] in 2003.

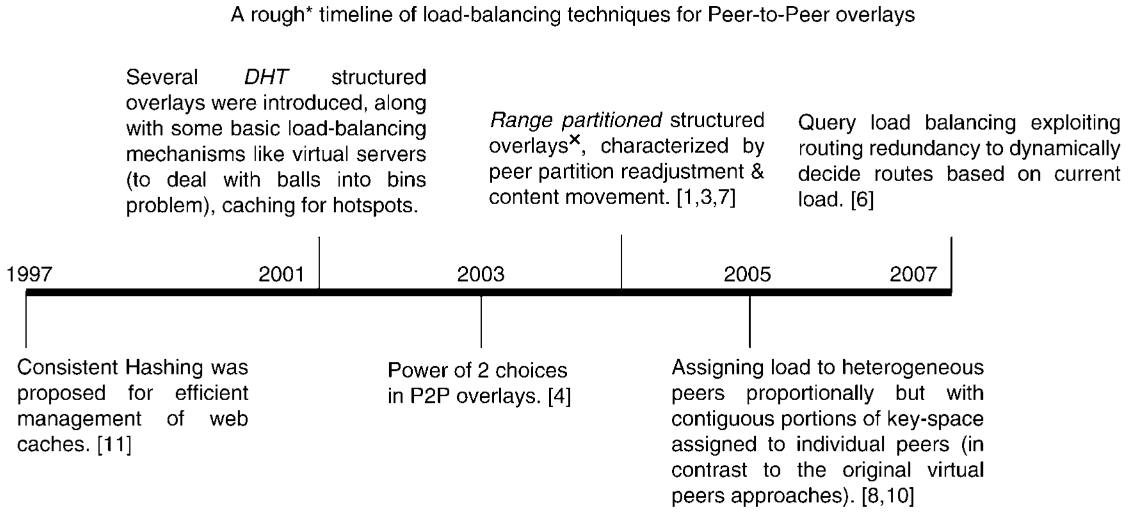
Using uniform hashing leads to loss of existing order relationships such as lexicographic ordering. Objects can be located only based on exact queries. For data-oriented applications involving complex queries like approximate or range queries, uniform hashing is unsuitable. Retaining the ordering relationships however means skewed distribution of the corresponding keys over the key-space of the overlay network. Several new mechanisms and alternate data-structures to dynamically and adaptively partition the key-space based on local load [1,3] and using adaptive reassignment of peers to a different part to the key-space [8] were proposed around 2004–2005 in order to deal with the skewed storage load distribution. Besides workload heterogeneity, the peers themselves have heterogeneous capacities, and assigning proportionally more load to a computer with more resources is another commonly employed approach [4,9] since 2005. Timeline of load-balancing mechanisms developed for P2P overlays is shown in Fig. 1.

While there is an extensive body of work related to storage load-balancing in P2P overlays, it has often been assumed that access load can be dealt with merely by caching. But if each cache is uniformly randomly chosen to answer query for a hot key, it nevertheless suffers from high load imbalance caused by variance, just like the balls into bins problem of uniform hashing. Furthermore, structured overlay routing algorithms do not account for balancing the traffic through each peer, which if unbalanced, may lead to systematic congestion that network level congestion control mechanisms can not deal with. Use of redundant overlay routes to dynamically route requests (e.g., queries) based on load at peers has been proposed in 2007 to deal with traffic and access load balancing [7].

While there have thus been several approaches dealing specifically with diverse kind of load-imbalances, research on a holistic approach (dealing with potentially arbitrary workload skews in terms of storage load, request distribution as well as message forwarding traffic load in the network on one hand, and the capacity of heterogeneous peers particularly in terms of the storage space and bandwidth they have and are willing to contribute) is in an early stage [9].

Foundations

Load in a P2P overlay network is typically either because peers need to store an object, or for answering requests (such as queries) by forwarding them to relevant peers, which may in turn reply back with the



A rough* timeline of load-balancing techniques for Peer-to-Peer overlays

Several *DHT* structured overlays were introduced, along with some basic load-balancing mechanisms like virtual servers (to deal with balls into bins problem), caching for hotspots.

Range partitioned structured overlays×, characterized by peer partition readjustment & content movement. [1,3,7]

Query load balancing exploiting routing redundancy to dynamically decide routes based on current load. [6]

Consistent Hashing was proposed for efficient management of web caches. [11]

Power of 2 choices in P2P overlays. [4]

Assigning load to heterogeneous peers proportionally but with contiguous portions of key-space assigned to individual peers (in contrast to the original virtual peers approaches). [8,10]

NOTES:
* Some papers with similar ideas were formally published in different calendar years, in which case the timeline indicates the publication of the first paper. The timeline shows the relative order of evolution of ideas, but the distances are not proportional. Also, the citations are not exhaustive.
× The original P-Grid [1]data-structure was designed from its very inception to deal with range-partitioned data, and the first paper was published in 2001, which makes it contemporary with the other DHT papers.

Load Balancing in Peer-to-Peer Overlay Networks. Figure 1. Evolution of load-balancing techniques for P2P overlays over time.

appropriate stored objects. A request may originate at any peer, and the routing algorithm of the overlay is used to forward the request to peers which can respond to it. Thus a request in turn creates both request forwarding and answering loads.

Each of these loads – storage and access load (which in turn leads to request forwarding and answering requests) – have different characteristics, both in terms of the resources they need, as well as the temporal characteristics. Storage load is permanent in that a object need to be stored persistently unless it is removed (deleted) based on application logic. Furthermore, even though storage load distribution changes over time, in general it is expected to change gradually. In contrast, access load is typically temporary, however the load-distribution may change abruptly (e.g., flash crowds). These distinct characteristics provide different opportunities and limitations in load-balancing.

Note that various load-balancing issues may also arise in unstructured and super-peer based overlay networks. These are dealt with heuristically. For instance, super-peers take into account the heterogeneity of peers' resources while designating some peers as super-peers. So the discussion here is primarily restricted to structured overlays.

Approaches for Balancing Storage Load

There have been various approaches to balance storage load among peers. Historically, uniform hashing has been used in many structured overlays as the first step to balance storage load. These are generally called distributed hash tables (DHTs). Subsequent networks overcome the limitations of using uniform hashing. An overlay to store range partitioned data has additional challenges in achieving load-balance. (The term "range-partitioned data" was introduced in 2004 [8] to distinguish structured overlays preserving ordering information from those using uniform hashing (DHTs). The same nomenclature is used here. Though structured overlays supporting range partitioned data came to mainstream focus in 2004–2005 with several new developments [1,3,8], P-Grid [1] supports range-partitioned data but was originally proposed in 2001, making it contemporary to the early DHT proposals.). Next is the discussion of load-balancing issues and mechanisms in both these categories of structured overlays.

Distributed Hash Tables (DHTs)

Hashing. Several pioneering structured overlays used uniform hashing as the primary means to balance storage load. The idea was to hash the object descriptor to generate a key uniformly (e.g., using cryptographic hashing like SHA) distributed over the key-space. Assignment of the peers to a partition of the key-space was also done uniformly at random, generally using hashing again. This category of structured overlay networks are commonly called distributed hash tables (DHTs). The rationale of the original DHT approach was that if peers were delegated sub-portions of the key-space uniformly randomly, and the keys were distributed uniformly randomly over the key-space, then the peers will have uniform load.

This intuition fails in practice because of what is known as the "balls into bins" effect. If m balls are put in n bins by choosing the bins independently and uniformly randomly for each of the balls, the variation of the number of balls per bin is high [14], and is of the order of m/n where $m >> n$. Since there are usually many more objects than the number of peers (i.e., high m/n), uniform hashing is inadequate to achieve good load-balance. There has been several approaches to reducing the effect of such statistical noise to improve storage load-balancing in DHTs.

Virtual servers (peers). One potential approach to reduce the variance is to have more peers. This can be achieved by creating multiple mutually independent virtual peers per physical computer [6]. Instantiating multiple virtual peers have several down-sides, including the fact that system states (e.g., routing table entries) need to be maintained for each of these virtual peers. Allocating independent key space partitions to more logical (virtual) peers also lead to inefficient search as more overlay level query forwarding are required.

Power of two (multiple) choices. The variance observed in the "balls into bins" process can be reduced without global coordination or knowledge by using an uncoordinated randomized algorithm. Instead of choosing one bin randomly uniformly, several (say two) distinct bins are chosen randomly uniformly as potential destination for a ball. Then the bin with fewer balls is actually chosen to put the current ball in. This mechanism significantly reduces load-imbalance for little extra overhead of choosing multiple (but constant) bins for each ball, instead of picking one [13]. In the context of DHTs, this can be realized by having multiple hash functions to generate multiple possible keys for the same object. The object is actually stored corresponding to that key which belongs to the key-space partition which is stored by the less loaded peer. In order to query an object, all the potential keys for that object need to be queried, thus significantly increasing the query overhead. An alternative is to store pointers at all the other keys to the actual location of the object. This however leads to higher storage and maintenance cost of these pointers.

Virtual peers for heterogeneous physical peers. Use of multiple but equal number of virtual peers was originally proposed to deal with the load-imbalance caused by the balls into bins effect for homogeneous peers [6]. The same idea of virtual peers has also been extended to deal with peer heterogeneity. Proportionally more virtual peers can be created corresponding to a physical node with more resources [10]. To deal with the drawbacks of virtual peers as mentioned earlier, allocation of contiguous partition of the key-space to virtual peers corresponding to the same physical node has been proposed subsequently [4]. Having contiguous key-space allocated to a physical peer has several advantages. There are fewer routing table links to maintain. Also fewer stored objects need to be moved among physically different peers. Note that using such contiguous partitions of the key-space among virtual peers of the same physical node means that there is actually no need of the "virtual peers" abstraction. It is a special case of partitioning the key-space adapted to the granularity of the load and the capacity of the peers as has been used in several other approaches dealing with range partitioned data [3,8,9].

Beyond DHTs

Using uniform hashing to generate keys lead to loss of ordering relationships such as lexicographic ordering, thus uniform hashing based DHTs are ill suited to deal with any queries other than exact queries. DHTs are not suitable for even simple crucial data-oriented queries like approximate queries or range queries. On the other hand, using an order preserving hashing to generate keys leads to potentially arbitrarily skewed and dynamic distribution of keys over the key-space, calling for different load-balancing mechanisms in structured overlays supporting *range partitioned data.* Several overlays exist to support range-partitioned data. While using slightly different data structures, each of these proposals deal with two critical impacts of skewed distributions of keys over the key-space – namely (i) partitioning the key space among peers in a granularity adaptive to the load distribution, and (ii) establishing a data-structure to keep routing and searching efficient despite non-uniform partitioning of the key-space.

Load-balancing (despite skewed load distribution) leads to allocation of key-space partitions to peers in an uneven manner. This is achieved by re-partitioning key-space of overloaded peers with newly joining peers, or by migrating peers from an under-loaded region. Migration and repartitioning are relatively drastic and need some (partial) global information. The advantage is that it can use any under-loaded peer to alleviate load imbalance for any other peer. In contrast, a localized strategy of readjustment of the key-space partitions with immediate neighbors in the key-space complements the above strategies. By shrinking or expanding the partitions, peers shed (or steal) load by transferring keys to (from) a neighboring peer in the key-space. These strategies are shown in Fig. 2.

In each of these strategies, in absence of global knowledge or coordination, peers need to make autonomous decision about whether it is over-loaded or underloaded with respect to other peers. The load information is obtained by sampling. Often peers exploit their routing topology for sampling a small nevertheless representative subset of the peers. Depending on their local view of the global state, overloaded peers may then request underloaded peers to share load (e.g., as in Mercury [3]), or some underloaded peers may spontaneously decide to relieve overloaded peers (e.g., as in P-Grid [2]). The sampling based load estimate is approximate, moreover the sampling is neither instantaneous nor concurrent over the network, and even if peers had perfect global knowledge, in the absence of coordination, peers make autonomous decisions for load-balancing. All this leads to the risk of oscillatory behavior, which is undesirable. Load-balancing mechanisms do not come free – all of them need transfer of objects, moreover migration also leads to inconsistent routing states at other peers which in turn causes route maintenance overheads. So it is desirable to dampen the load-balancing mechanism somewhat in order to avoid oscillatory effects [2].

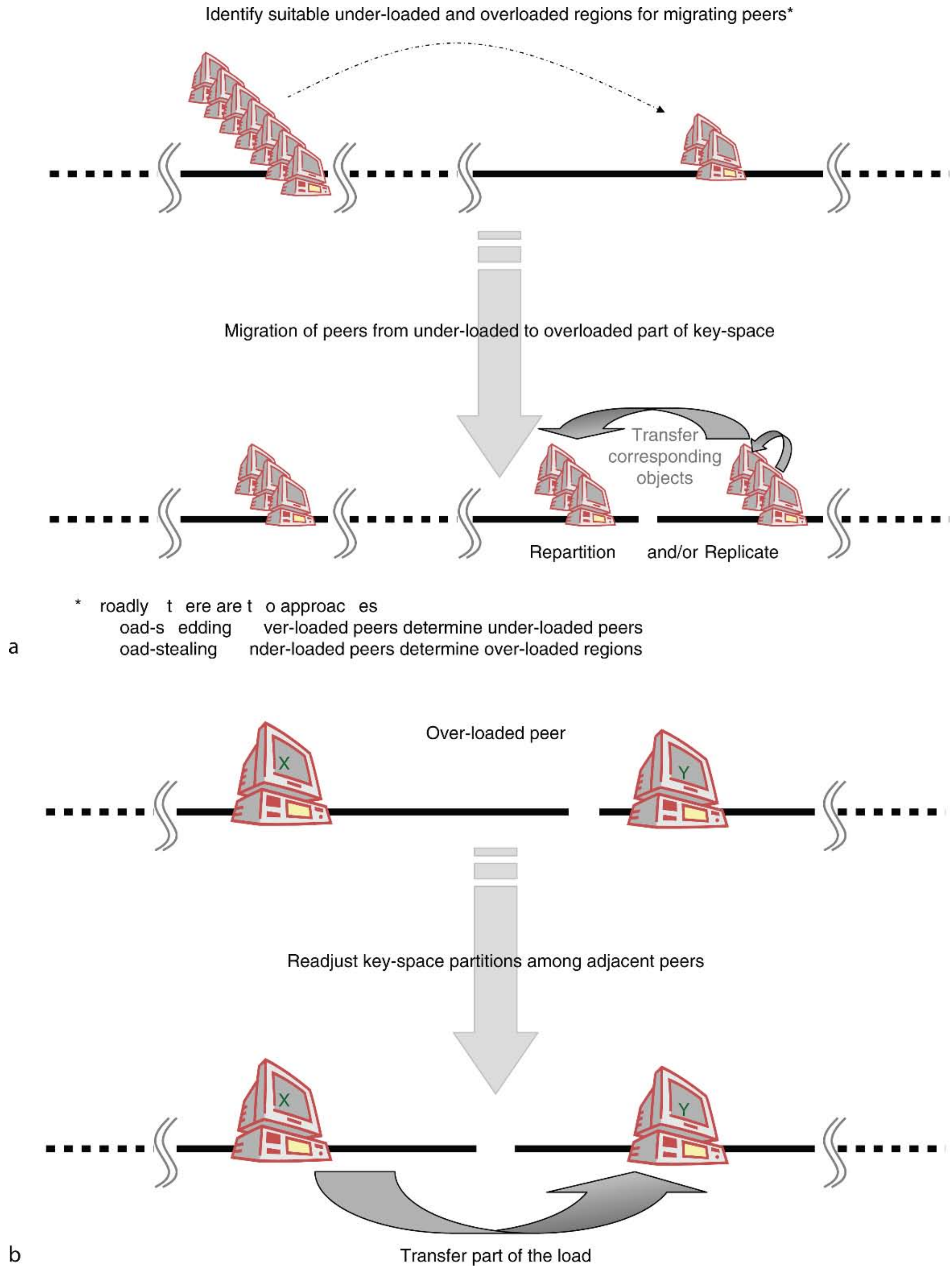
Consequence of load-balancing in overlays supporting range partitioned data. An immediate consequence of load-balancing in structured overlays supporting range partitioned data is that the key-space is partitioned unevenly, and new mechanisms than what is traditionally used in DHTs are required to decide efficient routing table entries and corresponding routing algorithms. Randomized choice of routing tables ensure efficient average routing latency in P-Grid [1] even if the routing tables abstract a (potentially highly) unbalanced trie, while Mercury [3] and Oscar [9] uses randomized routing in a slightly different way, extending the idea of small-world routing [12] over skewed key-space partitions.

Approaches for Balancing Requests Related Load

Relative popularity of different objects vary, and also change over time. As a consequence, even if storage-load is balanced, peers storing popular objects or services are prone to be overloaded. Replication or caching is the intuitive remedy to deal with hot-spots and dissipate load. If, however, one considers that there are n copies of an object (or service), and there are $m >> n$ requests for the same object, and each request is catered by any one of the copies randomly uniformly, again one encounters a "balls into bins" scenario with high variation in load distribution. Likewise, the load of forwarding messages at different peers have high variation if the routing algorithm of the corresponding overlays are used in a load-agnostic manner. This may in turn lead to congestion caused by the overlay layer, even when there is underutilized resources at other peers. Both these request related load imbalances can be mitigated with the use of simple heuristics exploiting the redundancy of routing choices (which is anyway necessary for fault-tolerance) to the least loaded of the peers that satisfy the routing criterion of the routing algorithm [7]. It has also been shown in the work that either caching or load aware routing is inadequate as stand-alone solutions, but complement each other, and hence need to be used together in order to balance request related loads.

Key Applications

Load-balancing is a critical and desirable property in distributed systems, and is necessary to achieve good performance and make fair and judicious use of available resources. Structured overlays provide indexing mechanism to locate objects distributed over the network with a guaranteed recall based on either exact or range queries, and also support other queries like approximate queries. The basic index can in turn be used to support diverse applications including cooperative file systems, P2P information retrieval, peer data management systems (PDMS) and collaborative work-spaces. For good performance (e.g., response time, fewer failures, etc.) at


Identify suitable under-loaded and overloaded regions for migrating peers*
Migration of peers from under-loaded to overloaded part of key-space
Transfer corresponding objects
Repartition
and/or Replicate
* roadly t ere are t o approac es
oad-s edding ver-loaded peers determine under-loaded peers
oad-stealing nder-loaded peers determine over-loaded regions
a
Over-loaded peer
X
Y
Readjust key-space partitions among adjacent peers
X
Y
b
Transfer part of the load

Load Balancing in Peer-to-Peer Overlay Networks. Figure 2. Mechanisms for re-balancing load. Note that a minimal redundancy (replication) is always necessary for fault-tolerance, but details of redundancy for fault-tolerance has been omitted for the sake of simplicity. (a) Underloaded peers migrate (new peers joins) to repartition or replicate key-space. (b) Readjust key-space partitions among adjacent peers.

application level, it is imperative that the underlying overlay has good load-balancing.

Future Directions

Peer-to-peer systems have dynamic and skewed workloads, and participating peers have heterogeneous capacities. Consequently, rather than distributing load equally among peers, which is what most current literature aims at, a more pragmatic approach is to look at how to distribute the whole load without violating the autonomous peers' contribution, desirably with minimal wastage of resources. Complementing this, it is also important to find incentive or punishment mechanisms which ensure that peers do contribute resources and do not participate in the system as parasites or free-riders. A holistic design, looking into mechanisms to ensure that autonomous peers contribute sufficient resources, and then allocating these heterogeneous resources to efficiently and effectively cater to dynamic and skewed workloads is the fundamental open problem in the context of load-balancing in peer-to-peer overlays.

Cross-references

- ▶ Peer Data Management System
- ▶ Peer to Peer Overlay Networks: Structure, Routing and Maintenance
- ▶ Routing and Maintenance
- ▶ Structure

Recommended Reading

1. Aberer K., Datta A., Hauswirth M., and Schmidt R. Indexing data-oriented overlay networks. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005.
2. Aberer K., Datta A., and Hauswirth M. Multifaceted simultaneous load balancing in DHT-based P2P systems: a new game with old balls and bins. In Self-Properties in Complex Information Systems, Springer, Berlin, 2005.
3. Bharambe A., Agrawal M., and Seshan S. Mercury: supporting scalable multi-attribute range queries. In Symp. on Communications Architectures and Protocols, 2004.
4. Brighten Godfrey P. and Stoica I. Heterogeneity and Load Balance in Distributed Hash Tables. In Proc. 24th Annual Joint Conf. of the IEEE Computer and Communications Societies, 2005.
5. Byers J., Considine J., and Mitzenmacher M. Simple load balancing for distributed hash tables. In Proc. 2nd Int. Workshop on Peer-to-Peer Systems, 2003.
6. Dabek F., Kaashoek F., Karger D., Morris R., and Stoica I. Wide-area cooperative storage with CFS. In Proc. ACM Symp. on Operating Systems Principles, 2001.
7. Datta A., Schmidt R., and Aberer K. Query-load balancing in structured overlays. In Proc. IEEE Int. Symp. on Cluster Computing and the Grid, 2007.
8. Ganesan P., Bawa M., and Garcia-Molina H. Online balancing of range-partitioned data with applications to peer-to-peer systems. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
9. Girdzijauskas S., Datta A., and Aberer K. Oscar: Small-world overlay for realistic key distributions. In Proc. Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing, 2006.
10. Godfrey B., Lakshminarayanan K., Surana S., Karp R., and Stoica I. Load Balancing in Dynamic Structured P2P Systems. In Proc. 23rd Annual Joint Conf. of the IEEE Computer and Communications Societies, 2004.
11. Karger D., Lehman E., Leighton T., Levine M., Lewin D., and Panigrahy R. Consistent hashing and random trees: tools for relieving hot spots on the World Wide Web. In Proc. ACM Symposium on Theory of Computing, 1997.
12. Kleinberg J. The small-world phenomenon: an algorithmic perspective. In Proc. ACM Symp. on Theory of Computing, 2000.
13. Mitzenmacher M. The power of two choices in randomized load balancing. IEEE Trans. Parall. Distrib. Syst., 12 (10):1094–1104, 2001.
14. Raab M. and Steger A. Balls into bins – a simple and tight analysis. In Proc. Int. Workshop on Randomization and Approximation Techniques in Computer Science, 1998.
15. Steinmetz R. and Wehrle K. (eds.). Peer-to-Peer Systems and Applications. Springer Lecture Notes in Computer Science, vol. 3485, 2005. Chapters 9 & 10.

Load Shedding

NESIME TATBUL
ETH Zurich, Zurich, Switzerland

Definition

Data stream management systems may be subject to higher input rates than they can immediately process with their available system resources (e.g., CPU, memory). When input rates exceed the resource capacity, the system becomes overloaded and the query answers are delayed. Load shedding is a technique to remove excess load from the system in order to keep query processing up with the input arrival rates. As a result of load shedding, the system delivers approximate query answers with reduced latency.

Historical Background

Load shedding is a term that originally comes from electric power management, where it refers to the process of intentionally cutting off the electric current on certain lines when the demand for electricity exceeds the available supply, in order to save the electric grid from collapsing. The same term has also been used in computer networking to refer to a certain form of congestion control approach, where a network router drops packets when its buffers fill up. More recently, load shedding has been proposed as a way to deal with overload in data stream processing systems [4].

Foundations

The goal of load shedding is to make sure that limited system resources operate below their capacity levels in case of unpredictable bursts in data arrival rates. This is achieved by selectively discarding some of the data items, thereby reducing the load at the expense of producing an approximate query answer. The main challenge in this problem is to minimize the loss in answer accuracy.

Assume a set of continuous queries Q, represented as a query plan of operators, where some of these operators may be shared among multiple queries. A set of inputs I feed these queries with streaming data, exerting a total load of $Load(Q(I))$ on a particular system resource with capacity C. A load shedding scheme must address the following key questions:

1. When to shed load? Conceptually, load needs to be shed whenever $Load(Q(I)) > C$.
2. Where to shed load? Load can be discarded at any point in the query plan. Dropping load at earlier points avoids wasting work; however, because of shared operators in the query plan, an early drop might adversely affect the accuracy of too many query answers.
3. How much load to shed? Just enough of the load at the chosen point(s) in the query plan must be shed so that the total resource demand gets below the available capacity with minimal total loss in accuracy.
4. Which data items must be discarded? The data items to be discarded should be chosen based on the approximation model and the properties of the operators in the query plan.

Furthermore, any load shedding scheme must have low run-time overhead in order not to further stress the limited system resources.

Various approaches have been proposed as solutions to the above listed issues. These approaches differ in their assumptions along several dimensions, including:

1. The limited resource under consideration (e.g., CPU, memory, communication bandwidth),
2. The way to reduce load (e.g., drop data, create summaries),
3. The approximation model/objective (e.g., maximum subset, minimum relative error, maximum throughput),
4. The query operator(s) under consideration (e.g., sliding window aggregates, windowed joins),
5. The data arrival model (e.g., stochastic models, temporal models),
6. The control loop (open vs. closed)
7. The system architecture (centralized vs. distributed).

Within the scope of the Aurora Project, Tatbul et al. have proposed a solution framework for load shedding which focuses on CPU as the main scarce resource, and discarding tuples by inserting special drop operators into the running query plan as the load reduction method [14]. The goal in this work is to minimize utility loss in query answers in terms of two alternative QoS (Quality of Service) dimensions: (i) percent tuple delivery, using a *random drop*, or (ii) output values delivered, using a *semantic drop*. A random drop discards tuples based on a drop probability, whereas a semantic drop does so based on a predicate on the tuple content. The earlier the load is reduced in a query plan, the larger is the saving in processing resources. However, shedding load early in a shared query plan may hurt the accuracy for multiple queries. To address this conflict, it is shown that load reduction should be applied either on the input streams, or on streams that immediately follow a shared operator in the query plan. Furthermore, these potential drop locations are ranked in terms of a metric, called *loss/gain ratio*. The drop location that causes the smallest QoS utility loss for the corresponding CPU processing power gained in return per unit drop of data, is preferred over the other drop locations with larger ratios. This way, the overall loss in QoS utility is minimized. For low run-time overhead, this work has proposed to pre-compute a set of load shedding plans based on system statistics, and insatiate these plans at run time based on the observed input rates. The proposed framework has also been extended to handle load shedding on windowed aggregation queries [15]. The key idea is to use a third type of drop operator, called a *window drop*, which semi-probabilistically discards load in units of windows instead of on a per-tuple basis. This way, window integrity can be preserved throughput a query plan, and query answers are guaranteed to be subsets of the original answers. An alternative to the window drop approach was earlier proposed by Babcock et al. [3]. This work also targets load shedding on aggregation queries under CPU constraints, but uses a different approximation model where the goal is to minimize the maximum relative error across all queries. Drops are applied on a per-tuple basis, leading to query answers with errors in their values. Window statistics and well-known statistical bounds such as the Hoeffding inequality are used to control these errors for a certain set of aggregate functions, including sum, count, and average. A close alternate to dropping tuples under CPU limitations is the selective processing approach proposed by Gedik et al. [8]. This work selectively processes tuples in the stream windows for join operators, in order to maximize the output rate or semantic utility of the query results, in the presence of variations in input rates as well as time correlations between two join inputs.

Load shedding can also be used to deal with memory limitations. Das et al. have focused on this problem for stream joins, where the maximum subset measure is used as the approximation metric [7]. This work assumes a frequency-based data arrival model and proposes two practical heuristics: (i) PROB, which drops tuples from an input stream which had the smallest frequency of occurrence on the opposite stream in the past (assuming that those tuples are the least likely to produce join results also in the future); (ii) LIFE, which drops tuples from an input stream whose product of frequency of occurrence on the opposite stream and remaining window lifetime is the smallest (i.e., the goal is to avoid investing on soon to be expired tuples). Within the scope of the STREAM Project, Srivastava and Widom have proposed an alternative load shedding approach for windowed stream joins in memory-limited environments [12]. This work is based on an age-based data arrival model, where it is assumed that the rate at which a tuple produces join results is solely determined by its age, specified as an age curve. To deal with memory shortage, tuples of a certain age are selectively discarded from the join window to make room for others, which have higher expectation of producing matches. The goal here is again to maximize the size of the join result set. A secondary concern in this work is to be able to produce a random sample from the join in case that the join is followed by an aggregate. In this case, the final output will not be a subset of the exact answer, and the overall goal is then to minimize the relative error, in line with the work of Babcock et al. [3].

Jain et al. have proposed a load shedding approach to reduce the network bandwidth usage [9]. This approach is based on Kalman Filters which can be used to model data streams as processes with states that evolve over time. As new tuples arrive at a source site, it is checked if the current model installed at the remote server site can still answer the query within given precision bounds. If so, there is no need to send the new tuple to the server, i.e., it can be discarded. Otherwise, the server model has to be updated, hence the new tuple is transmitted. Adaptivity is achieved by adjusting model parameters to changing load characteristics.

Stream load can also be reduced based on creating summaries of data instead of discarding data. This idea was pursued by two different lines of work within the scope of the TelegraphCQ Project: (i) Reiss and Hellerstein have proposed a load shedding technique called *data triage*, where excess data is not dropped, but stored in synopsis data structures [11]. At the end of a well-defined query window, the stored synopses are processed through a shadow query plan to compute an approximate result on the summarized portion of the data. Finally, exact and approximate results are merged into one composite result for that query window. This works using an error model based on Minkowski distance. (ii) Chandrasekaran and Franklin have focused on hybrid queries that process live data streams in correlation with historical data archived on disk [5]. In this case, disk becomes the bottleneck resource. To keep processing of disk data up with processing of live data, disk data is organized into multiple resolutions of reduced summaries. Depending on the live data rates, the system picks the right resolution summary to use in query processing. This is a form of load shedding that tries to cut down from disk access cost using data summaries.

The NiagaraCQ Project has taken an integrated approach where load shedding is seen as an extension to continuous query optimization. Kang et al. use a unit-time-based cost model where total cost of join processing is broken into two components, one for each join direction [10]. The optimal index and join algorithm combination for each direction is determined so as to maximize query throughput. Under CPU and memory limitations, the optimizer determines the ideal rate for each input and accordingly places a random drop to control the input rates. Ayad and Naughton use a similar analytical cost model, but extend it to plans with multiple joins [2]. It is shown that if computational resources are enough, then all join plans have the same throughput, however, they may substantially differ in their resource utilization. If all of these plans are infeasible (i.e., lead to CPU overload), then load must be shed via random drops. The focus is on picking the right join plan, the locations on the plan to insert the drops, and the amount of drops. An interesting result shown in this work is that the optimal join plan (i.e., with the lowest utilization) when resources are sufficient is not necessarily the optimal plan (i.e., with the highest throughput) when resources are insufficient.

All of the above described approaches assume that stream processing is performed on a single server. The overload problem can also arise in distributed stream processing systems where queries are distributed onto multiple servers. In a distributed environment, there is load dependency among the nodes that are assigned to run pieces of the same query. As a result, shedding load at an upstream node affects the load levels at its downstream nodes, and the load shedding actions at all nodes along a query plan will collectively determine the quality degradation at the query end-points. Within the scope of the Borealis Project, Tatbul et al. have modeled this problem as a linear optimization problem, and proposed two alternative solutions: (i) a centralized approach, where a coordinator node produces globally optimal plans with the help of an LP solver, and the rest of the nodes adopt their share of these global plans; (ii) a distributed approach, where nodes exchange metadata information (represented in the form of a feasible input table (FIT) which shows input rate combinations that are feasible for a given node) with their neighbors, and each node produces its own plan based on the available metadata [13]. Both of these solutions are based on the idea of pre-computing the load shedding plans in advance and storing them in a quadtree-based plan index. It is shown that the FIT-based plan generation is more efficient than its solver-based counterpart. Furthermore, the distributed solution is expected to be more responsive in dynamic environments due to its ability to incrementally update previously computed load shedding plans, reducing the amount of run-time communication needed among the nodes.

These approaches are all open-loop solutions in that the system load is periodically monitored and the load shedding algorithms are triggered as necessary. There has also been recent work that applies control-theoretic concepts to finer-grained adaptive load shedding on data streams [1,16]. These approaches are based on constructing a feedback loop that continually monitors the high-frequency variations in system parameters and makes the necessary adjustments in the load controllers accordingly. Such closed-loop approaches are shown to be more adaptive for input workloads with higher frequency fluctuations in stream data rates.

Load shedding finds use also in resource-intensive data stream mining applications. As argued by Chi et al. [6], in common data mining tasks such as classification and clustering of multiple data streams, the impact of load shedding on performance is not known a priori as the mining quality often depends on specific feature values observed in the stream in a non-monotonic way. This requires feature value prediction and adaptation. The Loadstar scheme uses a Markov model to predict the distribution of future feature values whose parameters are adaptively updated in time in order to maximize the classification quality under CPU constraints [6]. The high-level idea in this work is to allocate more resources to data streams that carry more uncertainty while shedding the ones whose class labels are more certain for the upcoming time window.

Key Applications

Load shedding can be used in all data-intensive streaming applications for which low latency answers can be more critical than full answer accuracy. These include sensor-based monitoring (e.g., habitat monitoring, bio-medical monitoring, weather monitoring, road traffic monitoring), RFID-based asset tracking, GPS-based location tracking, video-based security monitoring, and network traffic monitoring.

Future Directions

Load shedding in data stream management systems is currently an active area of research. A significant body of research results has been produced in this area since circa 2002. The future directions include development of new load shedding schemes for other sets of assumptions along the dimensions listed above. There is also a need to integrate the complementary and alternative solution schemes under a single framework, which could automatically select the right set of techniques for a broad range of system resources, based on the characteristics of the received workload as well as the application-specific quality of service criteria.

Cross-references

- ▶ Adaptive Stream Processing
- ▶ Approximate Query Processing
- ▶ Data Stream Management Architectures and Prototypes
- ▶ Continuous Query
- ▶ Data Sampling
- ▶ Data Sketch/Synopsis
- ▶ Data Stream
- ▶ Data Quality Dimensions
- ▶ Data Quality Models
- ▶ Data Reduction
- ▶ Stream Mining
- ▶ Stream-Oriented Query Languages and Operators
- ▶ Stream Processing
- ▶ Stream Sampling
- ▶ Wavelets on Streams
- ▶ Window-Based Query Processing
- ▶ Windows

Recommended Reading

1. Amini L., Jain N., Sehgal A., Silber J., and Verscheure O. Adaptive Control of Extreme-scale Stream Processing Systems. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2006.
2. Ayad A. and Naughton J.F. Static Optimization of Conjunctive Queries with Sliding Windows Over Infinite Streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
3. Babcock B., Datar M., and Motwani R. Load Shedding for Aggregation Queries over Data Streams. In Proc. 20th Int. Conf. on Data Engineering, 2004.
4. Carney D., Çetintemel U., Cherniack M., Convey C., Lee S., Seidman G., Stonebraker M., Tatbul N., and Zdonik S. Monitoring Streams - A New Class of Data Management Applications. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
5. Chandrasekaran S. and Franklin M.J. Remembrance of Streams Past: Overload-Sensitive Management of Archived Streams. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
6. Chi Y., Yu P.S., Wang H., and Muntz R.R. Loadstar: A Load Shedding Scheme for Classifying Data Streams. In Proc. SIAM International Conference on Data Mining, 2005.
7. Das A., Gehrke J., and Riedewald M. Approximate Join Processing Over Data Streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
8. Gedik B., Wu K., Yu P.S., and Liu L. CPU Load Shedding for Binary Stream Joins. Knowl. and Inf. Syst., 13(3):271–303, 2006.
9. Jain A., Chang E.Y., and Wang Y. Adaptive Stream Resource Management using Kalman Filters. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
10. Kang J., Naughton J.F., and Viglas S. Evaluating Window Joins over Unbounded Streams. In Proc. 19th Int. Conf. on Data Engineering, 2003.
11. Reiss F. and Hellerstein J.M. Data Triage: An Adaptive Architecture for Load Shedding In TelegraphCQ. In Proc. 19th Int. Conf. on Data Engineering, 2005.
12. Srivastava U. and Widom J. Memory Limited Execution of Windowed Stream Joins. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
13. Tatbul N., Çetintemel U., and Zdonik S. Staying FIT: Efficient Load Shedding Techniques for Distributed Stream Processing. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
14. Tatbul N., Çetintemel U., Zdonik S., Cherniack M., and Stonebraker M. Load Shedding in a Data Stream Manager. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003.
15. Tatbul N. and Zdonik S. Window-aware Load Shedding for Aggregation Queries over Data Streams. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.
16. Tu Y., Liu S., Prabhakar S., and Yao B. Load Shedding in Stream Databases: A Control-Based Approach. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.

LOC METS

Prasenjit Mitra
The Pennsylvania State University, University Park, PA, USA

Synonyms

Metadata encoding and transmission standard; Library of congress METS

Definition

The Library of Congress (LOC) Metadata Encoding and Transmission Standard (METS) is an XML (Extensible Markup Language) based format used for encoding metadata. The metadata is used to markup digital library objects in a repository or for exchange across repositories. METS is a Digital Library Federation initiative that is a successor to the Making of America II project (MOA2).

Key Points

The MOA2 project attempted to provide encoding formats for descriptive, administrative, and structural metadata for text and image-based documents. (http://www.loc.gov/standards/mets/METSOverview.v2.html) The Digital Library Federation (DLF) sponsored the project and the National Endowment for the Humanities funded it. MOA2 involved discussions led by the University of California, Berkeley with participants from New York Public Library and the libraries of Cornell, Penn State, and Stanford universities. The project produced a Document Type Definition

(DTD) that specifies a vocabulary and syntax for encoding digital objects. (http://www.lib.berkeley.edu/digicoll/bestpractices/mets_history.html) The library community realized that the MOA2 DTD was too restrictive, because, MOA2 did not provide some basic functionality required for multimedia objects like video and audio. METS arose from efforts to address these problems in MOA2.

Digital objects and the metadata needed to describe them are different from the metadata required for documents. Digital objects require structural metadata that indicates how the components of the object are glued together and technical metadata that specifies how the digital object was produced. For example, if a digital object contains image and text files, the structural metadata indicates the hierarchical structure of these objects and files. Furthermore, without these metadata, the authenticity of a digital object could be in doubt. METs allows specifications of the structural technical metadata, as well as metadata required for internal management and administration.

A METS document consists of seven sections:

- *METS header*: describes the document itself and publishes information about the creator, editor, etc.
- *Descriptive metadata*: both external, i.e., residing outside the document and internal.
- *Administrative metadata*: describes how the object was created and stored, specifies intellectual property rights, etc. Like the descriptive metadata, administrative metadata can also be both external or internal.
- *File Section*: lists all files that comprise the digital object. Groups of files can be specified using <fileGrp> with individual file elements specified using <file>.
- *Structural Map*: specifies the hierarchical structure of the digital object.
- *Structural Links*: allows creators to link different nodes in the hierarchy using hyperlinks.
- *Behavioral*: associates executable behaviors with content in the METS object.

METS is specified as an XML Schema and can be used in the role of a Submission Information Package (SIP), Archival Information Package (AIP) or Dissemination Information Package (DIP) within the Open Archival Information System (OAIS) Reference Model (http://ssdoo.gsfc.nasa.gov/nost/isoas/ref_model.html) [1–3].

Cross-references

▶ Digital Libraries

Recommended Reading

1. Gartner R. METS: Metadata Encoding and Transmission Standard. JISC Techwatch Report, 2002.
2. Guenther R. and McCallum S. New Metadata Standards for Digital Resources: MODS and METS. Bulletin of the American Society for Information Science and Technology, 2003.
3. Cundiff M.V. An introduction to the Metadata Encoding and Transmission Standard (METS). Library Hi Tech, 2004.

Local Web Search

▶ Geo-Targeted Web Search

Localization Abstraction

▶ Abstraction

Locality

▶ Locality of Queries

Locality of Queries

Pablo Barceló
University of Chile, Santiago, Chile

Synonyms

Locality; Hanf-locality; Gaifman-locality

Definition

Let σ be a relational signature without constant symbols. Given a σ-structure $\mathcal{A}$, its *Gaifman graph*, denoted by $G(\mathcal{A})$, has A (the domain of $\mathcal{A}$) as the set of nodes. There is an edge (a_1, a_2) in $G(\mathcal{A})$ iff there is a relation symbol R in σ such that for some tuple t in the interpretation of this relation in $\mathcal{A}$, both a_1, a_2 occur in t. The *distance* $d(a_1, a_2)$ is the distance in the Gaifman graph, with $d(a, a) = 0$. If $\bar{a}$ and $\bar{b}$ are tuples of elements, then $d(\bar{a}, \bar{b})$ stands for the minimum of $d(a, b)$, where $a \in \bar{a}$ and $b \in \bar{b}$.

Let $\mathcal{A}$ be a σ-structure, and $\bar{a} = (a_1,...,a_m) \in A^m$. The *radius r ball around $\bar{a}$* is the set $B_r^{\mathcal{A}}(\bar{a}) = \{b \in A \mid d(\bar{a}, b) \leq r\}$. The *$r$-neighborhood of $\bar{a}$ in $\mathcal{A}$* is the structure $N_r^{\mathcal{A}}(\bar{a})$ over signature σ expanded with m constant symbols, where the universe is $B_r^{\mathcal{A}}(\bar{a})$, the σ-relations are restrictions of the σ-relations in $\mathcal{A}$ to $B_r^{\mathcal{A}}(\bar{a})$, and the m additional constants are interpreted as $a_1,...,a_m$. Notice that any isomorphism h between $N_r^{\mathcal{A}}(a_1, ..., a_m)$ and $N_r^{\mathcal{B}}(b_1,...,b_m)$ imposes that $h(a_i) = b_i$, $1 \leq i \leq m$.

Let $\mathcal{A}, \mathcal{B}$ be σ-structures, $\bar{a} \in A^m$ and $\bar{b} \in B^m$. Then $(\mathcal{A}, \bar{a}) \rightleftarrows_r (\mathcal{B}, \bar{b})$ if there exists a bijection $f: A \to B$ such that $N_r^{\mathcal{A}}(\bar{a}c) \cong N_r^{\mathcal{B}}(\bar{b}f(c))$, for every $c \in A$.

Hanf-locality. An m-ary query Q on σ-structures is *Hanf-local*, if there exists $r \geq 0$ such that for every σ-structures $\mathcal{A}$ and $\mathcal{B}$, and every $\bar{a} \in A^m$ and $\bar{b} \in B^m$, if $(\mathcal{A}, \bar{a}) \rightleftarrows_r (\mathcal{B}, \bar{b})$ then $(\bar{a} \in Q(\mathcal{A}) \iff \bar{b} \in Q(\mathcal{B}))$.

Gaifman-locality. An m-ary query Q, $m > 0$, on σ-structures, is *Gaifman-local* if there exists $r \geq 0$ such that for every σ-structure $\mathcal{A}$, and every $\bar{a}, \bar{b} \in A^m$, if $N_r^{\mathcal{A}}(\bar{a}) \cong N_r^{\mathcal{A}}(\bar{b})$ then $(\bar{a} \in Q(\mathcal{A}) \iff \bar{b} \in Q(\mathcal{A}))$.

Key Points

Locality is a property of queries that finds its origins in the work by Hanf [1] and Gaifman [2], and that has shown to be very useful in the contexts of finite model theory and relational database theory. In very rough terms, a query is local if its truth value only depends on a small neighborhood of the input around its free variables. Locality is primarily used to prove inexpressibility results over finite structures, but it can also be used for establishing normal forms for logical formulae. It is a particularly helpful tool for finding easy winning strategies for the duplicator in the Ehrenfeucht-Fraïssé game, avoiding complicated combinatorial arguments.

The abstract study of the concepts behind the locality theorems of Hanf and Gaifman was initiated in [3]. It is shown there that the two notions of locality presented above are related: Every m-ary query (with $m > 0$) that is Hanf-local is also Gaifman-local, but the converse does not hold. The most paradigmatic example of a query that is neither Hanf- nor Gaifman-local is the one that computes the transitive closure of a graph.

A typical application of locality to prove the inexpressibility of query Q over logic $\mathcal{L}$ is done in two steps. First, show that Q is not local, and second, prove that every query defined by a formula in $\mathcal{L}$ is local. It follows from [3,4] that first-order logic, as well as many of its extensions with counting and generalized quantifiers, only define queries that are both Hanf- and Gaifman-local. This gives a simple proof of the fact that none of these logics can compute the transitive closure of a graph.

Cross-references

▶ Ehrenfeucht-Fraïssé Games

Recommended Reading

1. Hanf W. Model-theoretic methods in the study of elementary logic. In J.W. Addison et al. (eds.). The Theory of Models. North Holland, Amsterdam, 1965, pp. 132–145.
2. Gaifman H. On local and non-local properties. In Proc. of the Herbrand Symp., Logic Colloquium '81, North Holland, Amsterdam, 1982.
3. Libkin L. On the forms of locality over finite models. In Proc. 12th Annu. IEEE Symp. on Logic in Computer Science, 1997, pp. 204–215.
4. Libkin L. On counting logics and local properties. ACM Trans. Computat. Logic 1(1):33–59, 2000.

Locality of Reference

▶ Memory Locality

Locality Principle

▶ Memory Locality

Locality-Preserving Mapping

▶ Space Filling Curves
▶ Space-Filling Curves for Query Processing

Location Prediction

▶ Spatial Data Mining

Location Services

▶ Location-Based Services

Location-Based Services

Scott A. Bridwell, Harvey J. Miller
University of Utah, Salt Lake City, UT, USA

Synonyms

Location services; Geographic information services; Mobile map services; LBS

Definition

Location-based services (LBS) provide targeted information to individuals based on their geographic location in real or near-real time, typically through wireless communication networks and clients such as portable computers, personal digital assistants, mobile phones, and in-vehicle navigation systems.

Historical Background

LBS have emerged from the convergence of three major technological trends: (i) *geospatial technologies*, including location-aware technologies, geographic information systems (GIS) and spatial databases; (ii) *the Internet*, and; (iii) *information and communication technologies*, in particular, personal computing devices and mobile communication. Some of these technologies date to the late 1960s and early 1970s. For example, the first GIS was developed in the mid-1960s, while the global positioning system, the Integrated Services Digital Network (ISDN), mobile telephony and TCP/IP as a dominant network protocol emerged in the 1970s. The personal computer was introduced in early 1980s, an era that also experienced the deregulation of the telecommunications industry in the USA and Europe. Through the 1990s and early twenty-first century, these technologies matured, diffused and converged sufficiently such that LBS in the contemporary sense is possible [4,8].

Foundations

Location-Aware Technologies

Location-aware technologies (LATs) are devices that can report their geographic location in real or near-real time. Technologies for determining geographic location include the global positioning system, radiolocation methods and interpolation [2]. These can be used in combination.

The *global positioning system* (GPS) exploits time differences of signals arriving from subset of a satellite constellation in Earth orbit. The GPS is traditionally the most common LAT due to its high accuracy and low cost. GPS receivers are becoming small and light enough to embed in many other mobile technologies. However, the GPS requires line-of-sight with orbital satellites; this can be a problem in places with dense foliage or tall buildings, as well as inside built structures.

Radiolocation methods exploit wireless communication systems and determine location using methods such as those based on the time, time difference or angle of the signals' arrivals at base stations from mobile clients. The configuration of the network determines the precision of this method, with greater precision for areas with higher population densities and larger numbers of base stations. Radiolocation methods are less accurate in rural areas and have caused difficulties for wireless carriers attempting to achieve the E911 standards.

Other radiolocation methods include Bluetooth, WiFi, and Radio Frequency Identification (RFID) tags. These methods utilize a fixed network of sensory objects that sense objects in close proximity, interact with other sensors to triangulate the locations of an object based on some measure of signal strength, or utilize training information containing combinations of spatial coordinates and signal strengths. However, these methods are expensive and only cover small coverage areas. Their integration with positioning mechanisms outside of the building may also prove difficult.

Interpolation methods use distances and directions along a route from a known location to determine the current location.

Locational and Spatial Data Management

Since LBS users are likely to be mobile, a fundamental issue concerns the process whereby user locations are communicated to the central database and location server that supports LBS queries. At the time of the query the location service must have an accurate estimate of the user's location. The strategies used to communicate this information vary depending on the capabilities of the mobile device, the required accuracy for a particular query and the costs levied against the application server. This is essentially a question of how often the location of a moving object should be updated in the database.

Simple update strategies update a user's location whenever their location changes. This assumes there is no uncertainty in the tracking process and places high communication burdens on the servers. Temporal update strategies provide periodic updates of a user's location based on a recurring time interval. In the interval between updates, a user's location must be estimated according to an interpolation function. The accuracy of this estimate is based on the duration of the temporal recurrence interval, the distance the user travels between samples and assumed maximum travel speed. Distance or spatial update strategies provide updates when the user has moved a specified distance. This has the advantage of being more sensitive to the behavior of the user than temporal strategies. Dead-reckoning strategies integrate the temporal and distance approaches by comparing an estimated location with a measured location; this usually occurs on the client device. Updates to the database or location server occur when the difference between the estimated and measured location exceeds a given threshold.

Additional spatial data management issues include representations of geographic space and georeferenced content of real-world services such as restaurants and shops. While the former often includes physical features such as mountains, coastlines, and so forth, of critical importance is the transportation network. Transportation networks are important since they serve as the basis for navigation as well as the basis for georeferencing through street addresses. Required are multiple and integrated network representations that can support both navigation and locational referencing. The latter includes both the user's position provided by the LAT as well as linear referencing within the network [3].

Middleware, Open Standards and Interoperability

LBS middleware provides standard mechanisms for connecting the software components necessary for a given service. These components may be entirely present within a single device or distributed across a network of location, data and application servers. The primary objective of middleware is to support the interoperability of applications across devices and mobile service networks with different locational positioning methods and protocols. Middleware also provides a generic interface for querying geographic information stored in different formats and database systems.

The *Java ME Location API* is an example of device-oriented middleware. The API provides an interface defining methods for querying the location of the user with information corresponding to the confidence or uncertainty of the estimated location. This application resides on the device.

The *Open Location Services* (OpenLS) initiative is an example of set of standards to support network-oriented middleware. The OpenLS defines a set of core services and functions that are expected to satisfy most LBS applications. These services are similar database query languages in that they define an explicit structure for asking questions and interpreting the answers. Queries and results are structured using the eXtensible Markup Language (XML). Gateway services provide the mechanisms for querying the location of the user or other users required for the given application (e.g., friend finding); applications utilizing this service may exist on the device or be accessed through the network. Locational utility services provide the means for translating street addresses to geographical coordinates and vice versa. Directory services allow for querying points of interest (POI) such as restaurants. Route services provide a route, between two locations according to the preferences of the user. Presentation services allow for displaying querying results on a graphical map. Each of these services may exist on different providers. Using this framework, an LBS application developer could chain these services together in a similar manner to importing a software library or querying a relational database [7].

Locational Privacy

Detailed movement patterns in space and time are a signature that reveals much about an individual. Locational privacy is an emerging concept that suggests individuals have rights to their signature in space and time and can determine when, how and to what extent location information is communicated to others. Strategies for protecting locational privacy include regulation, privacy policies, anonymity and obfuscation. Regulation and privacy policies are trust-based mechanisms for defining unacceptable uses of location information. However, trust can be broken, making these strategies vulnerable to unintentional and intentional disclosure. Anonymity detaches locational information from an individual's identity. However, GIS can integrate locational information with other data such as remotely sensed imagery, geo-referenced social, economic and cadastral data, point-of-sale data, credit card transactions, traffic monitoring and video surveillance imagery, and other geosensor network data, allowing identity to be inferred. Obfuscation techniques deliberately degrade locational information, using error and uncertainty to protect privacy. Obfuscation techniques include geographic masking for static and mobile data (see [1]).

Location-Based Services. Table 1. Common LBS applications

• News	• Emergency response
• Navigation	• Person finding (friends, children)
• Traffic information	• Pet tracking
• Points of interest	• Electronic toll collection
• Advertising	• Car tracking
• Gaming	• Fleet management
• Weather forecasts	• Asset management

Key Applications

Key LBS applications span a large spectrum from convenient, "concierge" services to critical emergency response. Table 1 lists common LBS applications [9].

Two possible dimensions for classifying LBS applications are person versus device-oriented and push versus pull services [9]. *Person-oriented LBS* encompasses applications that are user-based. The user typically has control of the service: these intend to locate the person and/or use that location to enhance a service. These applications include news, navigation, points of interest, traffic, emergency response, and so on. *Device-oriented LBS* are not controlled by a person. Rather, the intent is to track an object or a set of objects. Applications include asset and fleet management.

Push services are LBS where a person receives information as a result of his or her location without actively requesting it. This information could be sent to the user based on prior consent (e.g., weather or traffic warnings) or without consent (e.g., advertising). *Pull services* are those where the user activity requests the information (e.g., points of interest, navigation, weather forecasts, traffic information).

Cross-references

▶ Mobile Objects Databases

Recommended Reading

1. Duckham M., Kulik L., and Birtley A. A spatio-temporal model of strategies and counter-strategies for locational privacy protection. In Geographic Information Science – Proc. 4th International Conference, 2006, pp. 47–64.
2. Grejner-Brzezinska D. Positioning and tracking approaches and technologies. In Telegeoinformatics: Location-based Computing and Services, H.A. Karimi, A. Hammad (eds.). CRC Press, Boca-Raton, FL, 2004, pp. 69–110.
3. Jensen C. Database aspects of location-based services. In Location-Based Services, J. Schiller, A. Voisard (eds.). Morgan Kaufmann, New York, 2002, pp. 115–145.
4. Jiang B. and Yao X. Location-based services and GIS in perspective. Comput. Environ. Urban. Syst., 30:712–725, 2006.
5. Kolodziej K.W. and Hjelm J. Local Positioning Systems: LBS Applications and Services. Taylor and Francis, London, 2006.
6. Küpper A. Location-Based Services: Fundamentals and Operation. Wiley, Hoboken, NJ, 2005.
7. Lopez X.R. Location-based services. In 1Telegeoinformatics: Location-based Computing and Services, H.A. Karimi, A. Hammad (eds.). CRC Press, Boca Raton, FL, 2004, pp. 171–188.
8. Shiode N., Li C., Batty M., Longley P., and Maguire D. The impact and penetration of location-based services. In Telegeoinformatics: Location-based Computing and Services, H.A., Karimi A. Hammad (eds.). CRC Press, Boca Raton, FL, 2004, pp. 349–366.
9. Spiekermann S. General aspects of location-based services. In Location-Based Services, J. Schiller, A. Voisard (eds.). Morgan Kaufmann, New York, 2004, pp. 9–26.

Lock Coupling

▶ B-Tree Locking

Lock Manager

▶ Concurrency Control Manager

Lock Tuning

▶ Tuning Concurrency Control

Locking Granularity and Lock Types

RALF SCHENKEL
Max-Planck Institute for Informatics, Saarbrücken, Germany

Synonyms

Locking granularity and lock types

Definition

Databases are usually organized hierarchically, with tablespaces containing tables, which in turn contain records. In multigranularity locking, this organization is exploited for a more efficient lock management by allowing transactions to lock objects of different granularities like tables or records. Thus, instead of locking each record of a table separately, a transaction can lock the complete table. To ensure a correct execution when transactions use different granularities for locking, additional lock modes are introduced to avoid non-serializable executions.

Key Points

Transactions that acquire many locks on small items like records or pages incur a non-negligible performance and memory overhead for managing these locks. Such transactions can benefit from acquiring locks on coarser granules like tables or complete tablespaces, avoiding many fine-grained locks. However, concurrency may be lower due to an increased number of conflicts with coarser locks. On the other hand, for transactions that access only a few records, locking these records directly will usually be the best solution.

Concurrent transactions that use different granularities for locking cannot easily coexist if any guarantees on the serializability of the execution should be provided, as locks of different granularities do not conflict with each other and hence cannot prevent any nonserializable executions. In such a *multi-granularity locking scheme*, transactions wanting to acquire a lock on a smaller granule must first acquire locks on all larger granules. While it would be sufficient if the locks on the coarser granules were acquired in the same mode as the lock on the smaller granule, this would lead to many unnecessary blockings due to conflicts on the coarser granules. As an example, consider two concurrent transactions that want to modify different records of the same table and hence need to acquire an exclusive lock on these records. As they modify different records, they do not conflict with each other. However, they would additionally have to acquire an exclusive lock on the table, too, which would make the second transaction wait until the first released the lock again.

Such performance penalties can be circumvented with additional lock types for coarser granules that express the *intended* lock type the transaction wants to acquire on the smaller granule: The lock modes *IS (intentional shared)* and *IX (intentional exclusive)* on the coarse granules correspond to *S (shared)* and *X (exclusive)* on the smaller granule. These lock types coexist with the standard shared and exclusive lock types on the coarse granule (that are used to get shared or exclusive access to *all* items of the smaller granule).

Locking Granularity and Lock Types. Table 1. Compatibility matrix of lock types

	S	X	IS	IX	SIX
S	+	–	+	–	–
X	–	–	–	–	–
IS	+	–	+	+	+
IX	–	–	+	+	–
SIX	–	–	+	–	–

The additional lock type *SIX (shared intentional exclusive)* combines an S and an IX lock. It is used when the transaction plans to read most of the items within the granule and additionally modify some of them. In this situation, an X lock would be too restrictive as it would lock out any other transaction from reading items in that table.

To ensure serializability in a system with multi-granularity locking, a transaction wanting to acquire a lock on any granule (for example, on a record of a table) must first acquire *warning locks* with the corresponding intentional lock type on all coarser granules. Table 1 shows the compatibility matrix for the different lock types.

The best locking granularity for a transaction may change throughout its execution. For example, if the transaction initially plans to access only a few records, it may be best to lock only records; if the access pattern changes later and it turns out that it is necessary to access all or almost all records of the table, it may be better to *escalate* the record-level locks to a single lock on the table. To do this, the transaction must first convert the warning lock (of type IS or IX) on the table to a "real" lock (of type S or X), for which it may have to wait until other transactions have released incompatible locks. It can release any record-level locks subsumed by the new lock on the table without compromising serializability.

Cross-references

- ▶ B-Tree Locking
- ▶ Locking Granularity and Lock Types
- ▶ Serializability
- ▶ SQL Isolation Levels
- ▶ Two-Phase Locking

Recommended Reading

1. Jim Gray, Raymond A. Lorie, Gianfranco R. Putzolu, and Irving L. Traiger. Granularity of locks in a large shared data base. In Proc. 1st Int. Conf. on Very Large Data Bases, 1975, pp. 428–451.
2. Gerhard Weikum, and Gottfried Vossen. Transactional Information Systems. Morgan Kaufman, San Francisco, CA, 2002.

Locking Protocol

▶ Two-Phase Locking

Log Component

▶ Logging/Recovery Subsystem

Log Manager

▶ Logging/Recovery Subsystem

Logging and Recovery

Erhard Rahm
University of Leipzig, Leipzig, Germany

Synonyms

Failure handling; Rollback; Undo; Redo; Checkpoint; Backup; Dump

Definition

Logging and recovery ensure that failures are masked to the users of transaction-based data management systems by providing automatic treatment for different kinds of failures, such as transaction failures, system failures, media failures and disasters. The main goal is to guarantee the atomicity (A) and durability (D) properties of ACID transactions by providing undo recovery for failed transactions and redo recovery for committed transactions. Logging is the task of collecting redundant data needed for recovery.

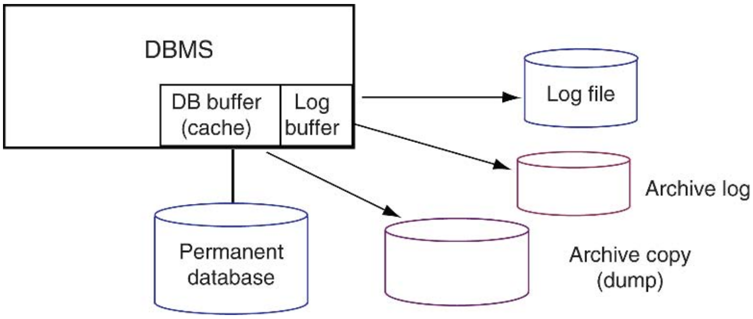
Key Points

The ACID concept requires that no data changes of failed transactions remain in the database. Failed transactions thus have to be rolled back by undoing all their changes (undo recovery). On the other hand, data changes of successfully ended (committed) transactions must not be lost but have to survive possible failures. Failure treatment thus implies a redo recovery for committed transactions. Recovery support is typically provided for transaction failures during normal processing (transaction recovery), for system failures (system recovery), and media failures (media recovery). System and media recovery are also known as two kinds of "crash recovery." In addition, disaster recovery can deal with the complete destruction of a computer center, e.g., due to an earthquake or terror attack. Recovery is typically based on logging, i.e., the collection of protocol data recording which transactions have been executed and which changes have been performed by them.

Figure 1 shows components of a central database management system (DBMS) involved in logging and recovery. The database objects (e.g., tables, records) are persistently stored in the *permanent database*, typically on one or several disks. All database operations including updates are performed in main memory. For this reason, pages of the database are cached in a main memory buffer (database buffer). Log records are persistently stored in a sequential *log file* on dedicated disks. Log records are written for the start, rollback and commit of transactions as well for every database change. For performance reasons log records are first collected in a log buffer in main memory, which is written to the log file when it becomes full or when a transaction commits. A transaction is committed when its commit record is logged on the log file.

Numerous approaches have been proposed and current database management systems provide efficient implementations for logging and recovery. The major tasks to be solved for dealing with the mentioned types of failures are:

- *Transaction recovery* (rollback) is performed when a transaction fails during normal processing, e.g., due to a program error or invalid input data. The log records in the log buffer and in the log file are used to undo the changes of the failed transaction in reverse order.
- *System (crash) recovery* is needed when the whole database (transaction) system fails, e.g., due to a hardware or software error. All transactions which were active and not yet committed at crash time



Logging and Recovery. Figure 1. DBMS components involved in logging and recovery.

have failed so that their changes must be undone. The changes for transactions that have committed before the crash must survive. A redo recovery is needed for all changes of committed transactions that have been lost by the crash because the changed pages resided only in main memory but were not yet written out to the permanent database. Periodically writing out modified pages, e.g., within so-called *checkpoints*, help to reduce the amount of redo work during crash recovery. Furthermore, the number of relevant log records and thus the size of the log file can be reduced by checkpoints.

- *Media (crash) recovery* deals with failures of the storage media holding the permanent database, in particular disk failures. The traditional database approach for media recovery uses archive copies (dumps) of the database as well as archive logs (see Fig. 1). Archive copies represent snapshots of the database and are periodically taken. The archive log contains the log records for all committed changes which are not yet reflected in the archive copy. In the event of a media failure, the current database can be reconstructed by using the latest archive copy and redoing all changes in chronological order from the archive log. A faster recovery from disk failures is supported by disk organizations like RAID (redundant arrays of independent disks) which store data redundantly on several disks. However, they do not eliminate the need for archive-based media recovery since they cannot completely rule out the possibility of data loss, e.g., when multiple disks fail.
- *Disaster recovery* can be achieved by maintaining a backup copy of the database at a geographically remote location. By continuously transferring log data from the primary database to the backup and applying the changes there, the backup can be kept (almost) up-to-date.

Cross-references

- ▶ ACID Properties
- ▶ Application Recovery
- ▶ Backup and Restore
- ▶ Buffer Management
- ▶ Crash Recovery
- ▶ Database Repair
- ▶ Multi-Level Recovery and the ARIES Algorithm
- ▶ RAID

Recommended Reading

1. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1983.
2. Haerder T. and Reuter A. Principles of transaction-oriented database recovery. ACM Comput. Surv., 15(4):287–317, 1983.

Logging/Recovery Subsystem

Andreas Reuter[1,2]
[1]EML Research gGmbH Villa Bosch, Heidelberg, Germany
[2]Technical University Kaiserslautern, Kaiserslantern, Germany

Synonyms

Audit trail; Log component; Log manager; Recovery manager

Definition

The logging/recovery subsystem (LRS) of a DBMS is responsible for implementing the fault tolerance mechanisms needed to support database transactions.

The log component stores the information needed to undo the updates performed by a transaction in case it has to be rolled back, either to an internal save point or to the beginning. It also stores information needed to reapply the updates of committed transactions to the database in case they are (partially) lost due to a system crash or after a storage media failure. In addition, the log component keeps track of all relevant state transitions such as begin-transaction, prepare, commit, abort, checkpoint, etc. The log is the first resource that is activated when restarting the database after a crash [1]. The recovery component orchestrates the activities needed to repair the database, depending on the situation. For example, after a crash, it first locates the last log record written and then reads the log backward, initiating undo for the operations of all transactions for which no "prepare" or "commit" entry has been found. Going backward, it looks for two specific records: The "begin" record of the oldest incomplete transaction – this is where undo stops, and the youngest checkpoint record. From that record, the recovery will read the log in forward direction and initiate a redo for the operations of all transactions that were completed before the crash. The log also plays a crucial role in implementing the two-phase commit protocol, and it can be used to detect security breaches.

Key Points

The log is the starting point of any recovery activity, so it has to be very reliable; therefore, the log devices are stored on devices that will retain their data in case of a power failure – typically disks, with current technology. Since all update operations, some read operations and a number of other activities create records to the log, it has to be very fast in order not to create a bottleneck. For that reason, the log is divided in to two portions: an online log containing all information that would be needed to support crash recovery, and an archive log that contains all information needed to recover from the loss of storage media. Data no longer needed in the online log is continuously moved to the archive by a background process that does not affect the overall system performance.

The recovery manager uses the log component for implementing some variation of the "write ahead log" protocol [2]. It basically states that before any update is applied to the database that may need to be rolled back, the data supporting the undo must be written to the log. Before a transaction is committed, all data needed to repeat the updates must be written to the log. Neither the log manager nor the recovery manager "understand" the structure of the log records. They are created and used by the resource managers that implement the objects (e.g., tuples, B-trees, queues). The log manager writes log records on behalf of those resource managers, and the recovery managers feeds them back to the resource managers when they are needed for undo and/or redo.

Cross-references

- ► Buffer Manager
- ► I/O-Subsystem
- ► Storage Resource Management
- ► Transaction Manager

Recommended Reading

1. Gray J. and Reuter A. Transaction Processing – Concepts and Techniques. Morgan Kaufmann, San Mateo, CA, 1993.
2. Härder T. and Reuter A. Principles of transaction oriented database recovery - a taxonomy. ACM Comput. Surv., 15 (4):287–317, 1983.

Logic of Time

► Time in Philosophical Logic

Logical Database Design: from Conceptual to Logical Schema

Alexander Borgida[1], Marco A. Casanova[2], Alberto H. F. Laender[3]
[1]Rutgers University, Piscataway, NJ, USA
[2]Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
[3]Federal University of Minas Gerais, Belo Horizonte, Brazil

Synonyms

Logical schema design; Data model mapping

Definition

Logical database design is the process of transforming (or mapping) a conceptual schema of the application domain into a schema for the data model underlying a particular DBMS, such as the relational or object-oriented data model. This mapping can be understood as the result of trying to achieve two distinct sets of goals: (i) representation goal: preserving the ability to capture and distinguish all valid states of the conceptual schema; (ii) data management goals: addressing issues related to the ease and cost of querying the logical schema, as well as costs of storage and constraint maintenance. This entry focuses mostly on the mapping of (Extended) Entity-Relationship (EER) diagrams to relational databases.

Historical Background

In the beginning, database schema design was driven by analysis of the prior paper or file systems in place in the enterprise. The use of a conceptual schema, in particular Entity Relationship diagrams, as a preliminary step to logical database design was proposed by Chen in 1975 [2,3], and the seminal paper on the mapping of EER diagrams to relational databases was presented by Teorey et al. in 1986 [7]. Major milestones in the deeper understanding of this mapping include [1,6], which separate the steps of the process, and pay particular attention to issues such as naming and proofs of correctness. Among others, the correct mapping of subclass hierarchies requires a careful definition of table keys [5], and the *maintenance* of optimized relational representations of ER schemas is discussed in [4].

Foundations

The mapping from an Extended Entity Relationship schema to a relational (logical) schema handles the issue of representing the different states of the conceptual schema through a function that provides for every "object set" *O* (entity set or relationship set) in the EER schema a relational table `T_O`. The data management-related issues are handled by merging, partitioning or otherwise reorganizing the tables obtained in the previous step, while making sure that there is no loss of information.

To begin with, some notation and assumptions concerning the conceptual schema expressed in EER notation are required. A unique set of identifying attributes *id*(*E*) is assumed to be available for every strong entity set *E* (i.e., one that is not a weak entity or a subclass). Each object set *O* participating in a relationship set *R* can be marked as being: *total*, indicating that each instance of *O* must participate in at least one relationship instance of *R*; *functional*, indicating that an instance of *O* can participate in at most one relationship instance of *R*; and as playing a particular *role* in the relationship, if *O* (or its sub/super-class) can participate in other relationships or as different arguments of *R*. For every relationship set *R*, *id*(*R*) is assumed to return a subset of participants that uniquely determine each relationship instance. This is needed in examples such as "Exactly one faculty advises each student for each major area," where *id*(`advises`)=`{Student,Major}`, because students can major in several areas. Of course, if the relationship set *R* has one functional participant *O*, then *id*(*R*) returns *O*; when there are multiple functional participants, *id*(*R*) is assumed to return one that is total, if available. Entities can be organized in a ISA/subclass hierarchy, with the ability to declare subclasses as *disjoint*, and/or *covering* the superclass.

In a relational schema, for each table *T*, one must specify its attributes/columns, its primary key, any foreign key referential constraints, and non-null constraints on columns. The notation $T[\underline{K}Y]$ is used to refer to a table named *T*, with columns *KY*and primary key *K*; *key*(*T*) returns *K*.

The Basic E2R Mapping

The initial mapping from entity and relationship sets in the conceptual schema to relational tables (referred to as the E2R mapping) is defined recursively as follows: For a strong entity set *E*, the attributes of *E* form the columns of table `T_E`, and its key is set to *id*(*E*). For example, for entity set `Student`, with attributes `sid`, `name` and `age`, and *id*(`Student`)=`sid`, the E2R mapping will generate a table `T_Student [sid,name,age]`.

To preserve first-normal form `T_E` does not include multi-valued attributes of *E*. For any such attribute *M* of *E*, one adds a separate table `T_EM`, whose attributes are those in *id*(*E*) plus a new attribute $\hat{M}$ that holds the individual values that occur in *M*; the full set of columns forms the key of this table, and a foreign key constraint is added from column *id*(*E*) of `T_EM` to `T_E`. Thus, if the entity set `Student` has a multi-valued attribute `phone`, then the E2R mapping will generate a table `T_StudentPhone[sid,phone]`, with `sid` being a foreign key with respect to `T_Student`.

If *E* is a subclass of some class *F*, then the columns of `T_E` consist of the attributes of *E* together with those in *key*(`T_F`) – the key "inherited" from `T_F`.

Therefore *key*(T_F) is the key of T_E, and a foreign key reference to T_F is needed to ensure that every subclass instance is in the super-class. For example, supposing that GradStudent is given as a subclass of Student, then T_GradStudent will have, among others, a column sid, which will also be its key and a foreign key referencing T_Student. In case subclasses are disjoint or cover the super-class, appropriate SQL assertions need to be added to check these constraints.

For a relationship set *R*, T_R has as attributes all the attributes of *R*, as well as the union of the sets of attributes X_O=*key*(T_O) of all object sets *O* participating in *R*; for each such set of attributes X_O, a foreign key constraint from T_R to T_O is added, in order to avoid dangling references. Moreover, a NOT NULL constraint is added to the corresponding columns of X_O. The keys of the tables generated from the object sets in *id*(*R*) jointly become the key of T_R. For example, suppose that admits is defined as a relationship set with participating entity sets Professor and Student, plus attribute Year. Assuming that *id* (Professor)=pid, then the relational representation of admits would be T_admits[pid,sid,year], with pid and sid being foreign keys that reference T_Professor and T_Student, respectively. Additional constraints found in some EER schemas, such as numeric lower and upper bounds on relationship participation (e.g., a child has between 2 and 2 parents), can only be enforced using general SQL assertions.

For a weak entity set *W*, T_W includes the attributes of *W*, any attributes of the identifying relationship set of *W*, and the identifying attributes *key*(T_E) of the entity set *E* that "owns" *W*. The key of T_W is the union of the local identifier *id*(W) of *W* with the key attributes *key*(T_E) of *E*. A foreign key constraint from T_W to T_E must also be added. For example, suppose that Section is specified as a weak entity set with respect to Course, with local identifier sectionNr and identifying relationship set sectionOf; furthermore, assume that *key*(T_Course)=courseNr. Then, the relational representation of Section would be T_Section[courseNr,sectionNr], with courseNr as a foreign key referencing T_Course.

Refinements

In the creation of tables for relationship sets and for weak entity sets, special attention needs to be paid when duplicate column names arise (because the same object can be involved in a relation in multiple ways). In this case, role names need to be used to disambiguate the columns. For example, if *id*(Professor)=*id*(Student)=ssn (because both are subclasses of Person say), then T_admits should have columns named prof_ssn and student_ssn, or admitter_ssn and admitted_ssn.

The treatment of subclass hierarchies which are not trees and where subclasses may inherit different identifiers from different parents also requires special care, and is discussed in [5].

The previous constructions provide a relational schema that allows every instance of the conceptual schema to be captured precisely. Relational schema restructuring by relation merging and sometimes partitioning is then undertaken for a number of reasons. The cardinal rule of all techniques is the need to be able to recover precisely the original relation instances from the merged instance (viz. lossless join). An additional rule observed in the techniques here is to avoid duplicating information in ways that lead to "update anomalies."

Table Merging

The most familiar technique replaces tables by their outer join, with the goal of making it easier to express and evaluate queries by avoiding the join. For example, one can merge table student[sid,name,age] with minorsIn[sid,minor], to obtain student2 [sid,name,age,minor]. Conceptually, such a change usually merges a functional relationship with the entity it is about, or a sub-class with its super-class. The main disadvantage of this change is the need to store null values as part of the outer join (in the above example, for students who do not have a minor).

The rule being applied in this case can be stated as:

Rule 1: Tables $T[\underline{K}X]$ and $R[\underline{\hat{K}}Y]$, where $\hat{K}$ is a foreign key referencing *T*, can be replaced by table $T_R[\underline{K}XY]$, whose intended use is as the left outer join of *T* and *R*.

Applying the above transformation, one must be mindful of a number of potential problems. First, identical column names occurring in *X* and *Y* cause conflicts, which must be avoided by renaming. Second, this merge may prevent making previously possible distinctions when there is no guarantee that for every tuple in table *T* there is a corresponding tuple in *R* referencing it. For example, in the original schema one can distinguish the case of a student who has a minor that is not known (represented by a tuple with NULL in column minor of table minorsIn) from the case of a student without a minor (represented by the absence of a the student's sid in minorsIn); but in student2 both cases have NULL in the minor column. To capture such distinctions one can add a Boolean attribute isInMinorsIn? to table student2. This technique becomes essential when merging the table of a subclass (e.g., T_GradStudent) into that of the superclass (e.g., T_Student) in case all the additional attributes of the subclass (e.g., advisor) may have null values, and therefore cannot be used to detect membership in the subclass.

Third, previously easy-to-state constraints may now become more convoluted. For example, if graduate students must have both an advisor and a department attribute (hence these columns have NOT NULL constraints in T_GradStudent), then since nulls must be allowed into these columns of T_Student after the merge, one must ensure (using an SQL check constraint) that nulls in the two columns *correlate*. Moreover, foreign key references to table T_GradStudent now become references to T_Student, and must be supplemented by SQL assertions verifying that some attribute associated with graduate students has a non-null value.

Table Partitioning

Tables can also be reorganized by so called "horizontal splitting" as stated by the following rule:

Rule 2: Table $T[\underline{K}X]$ can be replaced by tables $T_1[\underline{K}X]$, $T_2[\underline{K}X]$,...,with the intended use of T_1,T_2,... being as a partition of the tuples in T.

In logical (as opposed to physical) schema design, the partition tables usually have semantic interpretation as subclasses of relationship sets (e.g., T_admitted replaced by T_currentlyAdmitted and T_previouslyAdmitted) or of entitie sets (e.g., T_Student[sid,name,age] replaced by gradStudent$_0$[sid,name,age], undergradStudent$_0$ [sid,name,age], and student$_0$[sid,name, age] – the latter with students that are neither undergraduate nor graduate). Note that after such a split, one can merge gradStudent$_0$[sid,name,age] with gradStudent[sid,office] to get another variant of mapping subclass hierarchies to tables; this one is particularly good for cases when the subclasses cover the superclass.

Table partitioning can be seen as encoding into each table selection criteria, which therefore once again facilitates query statement and evaluation. The down side is that once again built-in constraints, such as foreign keys, now need to be stated as more complex SQL inter-table assertions.

Mapping from Non-ER Conceptual Schemas

UML class diagrams are increasingly popular for the specification of conceptual schemas. The correspondences "class" ⇌ "entity set," "association" ⇌ "relationship set" make it easy to reformulate the E2R mapping as a UML-to-Relational (U2R) mapping. Higher arity relationships (such as assignedTo (Professor,Course,Semester)) need to be reified in UML (i.e., represented as classes of objects Assignment related by functional associations f_1, f_2 and f_3 to Professor, Course and Semester respectively), but these end up producing a similar relational schema as E2R because tables T_f$_i$ are merged into T_Assignment; the artificial key *id*(Assignment) should however be removed.

As in the above example, the main difficulty in U2R is dealing with identifying ("key") attributes for entities, which are not mandated by the UML data model, since it assumes objects have intrinsically unique identity.

Key Applications

Database Design

The mapping of the conceptual schema into a logical schema is the central step of the database design process. A carefully crafted mapping will guarantee that the logical schema correctly represents the application domain that the conceptual schema models.

Future Directions

The publication of the SQL:2003 language standard provides additional mapping opportunities, exploiting some of the object-oriented features of the language.

As a simple example, SQL:2003 introduces the MULTISET data type, which can be used to avoid defining a separate table to accommodate multi-valued attributes of entity or relationship sets.

More importantly, SQL:2003 supports the declaration of an "*identity attribute*" for a table (designated with the special keyword IDENTITY). The value of an identity attribute is unique and automatically generated whenever a new row is inserted into the table. This construct is useful to the E2R and, especially, to the

U2R mapping process since it avoids the creation of an *artificial* key for entity *E* by adding an identity attribute for table T_E.

URL to Code

DBDesigner (http://fabforce.net/dbdesigner4/) is an open-source database design system, available from fabFORCE.net, which integrates EER modeling with the derivation and maintenance of a relational schema for mySQL.

Cross-references

▶ Extended Entity-Relationship Model
▶ Functional Dependency
▶ Information Capacity

Recommended Reading

1. Casanova M.A., Tucherman L., and Laender A.H.F. On the design and maintenance of optimized relational representations of entity-relationship schemas. Data Knowl. Eng., 11(1):1–20, 1993.
2. Chen P.P. The entity-relationship model: toward a unified view of data. In Proc. 1st Int. Conf. on Very Large Data Bases, 1975.
3. Chen P.P. The entity-relationship model – toward a unified view of data. ACM Trans. Database Syst., 1(1):9–36, 1976.
4. da Silva A.S., Laender A.H.F., and Casanova M.A. An approach to maintaining optimized relational representations of entity-relationship schemas. In Proc. 15th Int. Conf. on Conceptual Modeling, 1996, pp. 292–308.
5. da Silva A.S., Laender A.H.F., and Casanova M.A. On the relational representation of complex specialization structures. Inf. Syst., 25(6–7):399–415, 2000.
6. Markowitz V.M. and Shoshani A. Representing extended entity-relationship structures in relational databases: a modular approach. ACM Trans. Database Syst., 17(3):423–464, 1992.
7. Teorey T.J., Yang D., and Fry J.P. A logical design methodology for relational databases using the extended entity-relationship model. ACM Comput. Surv., 18(2):197–222, 1986.

Logical Foundations of Web Data Extraction

CHRISTOPH KOCH
Cornell University, Ithaca, NY, USA

Definition

Several wrapper programming languages for extracting information from Web pages have been based on logic, specifically on fragments of datalog. This entry shows how logical languages can be used for Web information extraction, and surveys expressiveness and complexity aspects of a foundational logical wrapping language, monadic datalog.

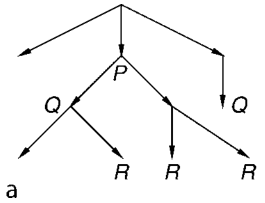
Historical Background

A substantial amount of research has studied the problem of learning wrapper programs from examples (see Wrapper Induction). Unfortunately, it is now known that the expressive power of learnable wrappers is fundamentally limited. This has motivated further work on visual wrapper programming languages, which simplify and speed up wrapper definition. Visual wrapping is now supported by several implemented systems (cf. XWrap [3] and W4F [4]; Lixto [1], a commercial product). The Lixto project was the first to emphasize and study expressiveness of visual wrapper languages. Its approach is based on a datalog-like language, ELog, whose core was shown to be expressively equivalent to monadic second-order logic over finite node-labeled trees, a natural yardstick for such languages. The datalog approach to wrapping has since then be adopted by other researchers; for instance, the recent XLog wrapper language [5] is closely related to ELog.

Foundations

Information Extraction Functions and Wrappers

The core notion on which logic-based wrapping is based is that of an *information extraction function*, which takes a labeled unranked tree (representing a Web document) and returns a subset of its nodes. In other words, an information extraction function is a unary query. A wrapper is a program which implements one or several such functions, and thereby assigns unary predicates to document tree nodes. Based on these predicate assignments and the structure of the input document viewed as a tree, a new tree can be computed as the result of the information extraction process in a natural way, along the paths of the input tree but using the new labels and omitting nodes that have not been assigned a new label. (see Fig. 1 for an example of such a transformation.) That way, the nodes of a document tree can be re-labeled, and some can be dropped as irrelevant, but it is not possible to significantly transform the original tree structure. This coincides with the intuition that a wrapper may change the presentation of relevant information, its packaging or data model (which does not apply in the case of *Web wrapping*), but should not handle substantial data



Logical Foundations of Web Data Extraction. Figure 1. Tree annotated with predicates *P*, *Q*, and *R* defined by information extraction functions (a), and wrapping result (b). (Original node labels of the input tree are not shown.)

transformation tasks. There is wide agreement that wrapping should exclude operations such as joins which are not necessary for information extraction but may be part of data transformation and integration tasks that may follow extraction.

A main goal is thus to find a language for specifying expressive unary queries over trees which nevertheless can be efficiently computed.

Tree Structures

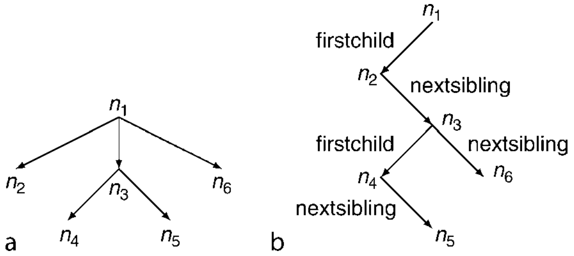
Unranked finite ordered trees with node labels from a finite set of symbols Σ correspond closely to parsed HTML or XML documents. In an unranked tree, each node may have an arbitrary number of children. An unranked ordered tree can be considered as a structure of relational schema

$$\tau_{ur} = \langle \text{dom}, \text{ root}, \text{ leaf}, \text{ } (\text{label}_a)_{a \in \Sigma}, \\ \text{firstchild}, \text{ nextsibling}, \text{ lastsibling}\rangle$$

where “dom” is the set of nodes in the tree, “root”, “leaf”, “lastsibling”, and the “label$_a$” relations are unary, and “firstchild” and “nextsibling” are binary. All relations are defined according to their intuitive meanings. “root” contains exactly one node, the root node. “leaf” consists of the set of all leaves. “firstchild(n_1, n_2)” is true if n_2 is the leftmost child of n_1; “nextsibling(n_1, n_2)” is true if, for some i, n_1 and n_2 are the ith and $(i + 1)$th children of a common parent node, respectively, counting from the left (see also Fig. 2). label$_a(n)$ is true if n is labeled a in the tree. Finally, “lastsibling” contains the set of rightmost children of nodes.

Monadic Datalog

Monadic datalog is obtained from full datalog (see Datalog) by requiring all intensional predicates to be



Logical Foundations of Web Data Extraction. Figure 2. (a) An unranked tree and (b) its representation using the binary relations “firstchild” (↙) and “nextsibling” (↘).

unary. A unary query is a function that assigns a predicate to some elements of dom (or, in other words, selects a subset of dom). For monadic datalog, one obtains a unary query by distinguishing one intensional predicate as the *query predicate*. In the remainder of this entry, a monadic datalog query will always be understood to be a unary query specified as a monadic datalog program with a distinguished query predicate. Of course, monadic datalog allows for the the definition of multiple unary queries within a single program; thus a set of information extraction functions that defines a wrapper can be given through a single monadic datalog program.

Monadic second-order logic (MSO) is obtained from second-order logic by requiring all second-order quantifiers to range over sets (i.e., unary relations). A unary MSO *query* is defined by an MSO formula φ with one free first-order variable. Given a tree t, it evaluates to the set of nodes $\{x \in \text{dom} \mid t \vDash \varphi(x)\}$.

It has been argued in [2] that unary queries in monadic second-order logic (MSO) over trees are an appropriate expressiveness yardstick for information extraction functions. MSO over trees is well-understood theory-wise and quite expressive. It has also been used as an expressiveness yardstick for node-selecting XML query languages.

By restricting the structures considered to trees, monadic datalog acquires a number of nice properties. First, the query evaluation complexity is linear in the size of the data and of the program:

Theorem 1 ([2]). *Over τ_{ur}, monadic datalog has $O(|\mathcal{P}|*|dom|)$ combined complexity (where $|\mathcal{P}|$ is the size of the program and $|dom|$ the size of the tree).*

This is is stark contrast to full datalog, which is EXPTIME-complete w.r.t. combined complexity over arbitrary finite structures and NP-complete over trees. Monadic datalog has some other nice properties which have been studied; in particular, the containment and boundedness properties – which are both considered relevant to query optimization – for monadic datalog are decidable, while they are undecidable for full datalog.

A unary query over trees is MSO-definable exactly if it is definable in monadic datalog.

Theorem 2 ([2]). *A unary query over τ_{ur} unranked ordered finite trees is MSO-definable if and only if it is definable in monadic datalog.*

Theorem 2 asserts that monadic datalog programs can define "universal" properties over trees, such as that a certain fact holds everywhere or nowhere in a tree. This may seem somewhat unintuitive because monadic datalog does not feature negation. Still, it follows from the fact that the relations defining the tree in τ_{ur} allow us to traverse the tree (starting from its "ends" such as the leaves or the root) using a recursive program and to compute universal properties along the way.

Example 1. Consider the problem of selecting all those nodes from an HTML tree which do *not* contain an HTML "table" in their subtrees. A monadic datalog program for this (with query predicate Q) can be defined as follows.

$$\begin{aligned}
NoTableBelow(x) &\leftarrow leaf(x).\\
NoTableBelow(x) &\leftarrow firstchild(x,y), NoTable(y).\\
NoTableRight(x) &\leftarrow lastsibling(x).\\
NoTableRight(x) &\leftarrow nextsibling(x,y), NoTable(y).\\
NoTable(x) &\leftarrow \overline{label}_{table}(x), NoTableBelow(x),\\
& NoTableRight(x).\\
Q(x) &\leftarrow \overline{label}_{table}(x), NoTableBelow(x).
\end{aligned}$$

Here it may either be assumed that there is a predicate $\overline{\text{label}}_{table}$ true for those nodes not labeled "table" (this predicate then needs to be added to τ_{ur}) or $\overline{\text{label}}_{table}$ can be defined in monadic datalog by the rules $\{\overline{\text{label}}_{table}(x) \leftarrow \text{label}_l(x). \mid l \in \Sigma, l \neq \text{"table"}\}$.

Note that both $\overline{\text{label}}_{table}$ and *NoTableBelow* are true for a node if it does not contain a "table" in its subtree, while *NoTable* is true for a node if the same holds (i.e., it does not contain a "table" in its subtree) in the binary-tree model of Fig. 2b.

Each monadic datalog program over trees can be efficiently rewritten into an equivalent program using only very restricted syntax. This motivates a normal form for monadic datalog over trees.

Definition 1. A monadic datalog program $\mathcal{P}$ over τ_{ur} is in *Tree-Marking Normal Form* (TMNF) if each rule of $\mathcal{P}$ is of one of the following four forms:

$$\begin{aligned}
&(1)\ p(x) \leftarrow p_0(x).\\
&(2)\ p(x) \leftarrow p_0(x_0), R(x_0, x).\\
&(3)\ p(x) \leftarrow p_0(x_0),\ R(x, x_0).\\
&(4)\ p(x) \leftarrow p_0(x),\ p_1(x).
\end{aligned}$$

where the unary predicates p_0 and p_1 are either intensional or from τ_{ur} and R is a binary predicate from τ_{ur}.

In the next result, the schema for unranked trees may extend τ_{ur} to include the natural child relation – likely to be the most common form of navigation in trees.

Theorem 3 ([2]). *For each monadic datalog program $\mathcal{P}$ over $\tau_{ur} \cup \{child\}$, there is an equivalent TMNF program over τ_{ur} which can be computed in time $O(|\mathcal{P}|)$.*

Syntax as simple as that of TMNF is important for visual wrapping. A single rule in monadic datalog may still consist of an arbitrary number of joins involving the binary relations of τ_{ur}. TMNF is much simpler and a visual rule definition process is not hard to define (cf. [1]). Nevertheless, TMNF has the full power of MSO and admits efficient evaluation. TMNF is the core of the ELog language of the Lixto system [1].

Key Application

Web Information Extraction; XML Query Languages.

Cross-references

- ▶ Datalog
- ▶ Wrapper
- ▶ Wrapper Generator
- ▶ Wrapper Induction

Recommended Reading

1. Baumgartner R., Flesca S., and Gottlob G. Visual web information extraction with Lixto. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
2. Gottlob G. and Koch C. Monadic datalog and the expressive power of web information extraction languages. J. ACM, 51(1):74–113, 2004.

3. Liu L., Pu C., and Han W. XWRAP: An XML-enabled wrapper construction system for web information sources. In Proc. 16th IEEE Int. Conf. on Data Engineering, 2000, pp. 611–621.
4. Sahuguet A. and Azavant F. Building intelligent web applications using lightweight wrappers. Data Knowl. Eng., 36(3):283–316, 2001.
5. Shen W., Doan A., Naughton J.F., and Ramakrishnan R. Declarative information extraction using datalog with embedded extraction predicates. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 1033–1044.

Logical Models of Information Retrieval

Fabio Crestani
University of Lugano, Lugano, Switzerland

Definition

Logical models of Information Retrieval (IR) are defined as those that follow a *logical definition of relevance*. For Cooper logical relevance is defined as "logical consequence." To make this possible both queries and documents need to be represented by sets of declarative sentences. The query is represented by two formal statements called "component statements" of the form *p* and ¬*p*. A subset of the set of stored sentences is called "premiss set" if and only if the component statement is a logical consequence of that subset. A "minimal premiss set" for a component statement is one that is as small as possible. Logical relevance is therefore defined as a two-place relation between stored sentences and the query represented as component statements:

- ► "A stored sentence is logically relevant to (a representation of) an information need if and only if it is a member of some minimal premiss set of stored sentences for some component statement of that need."

This definition of relevance is essentially just a proof-theoretic notion that has been later generalized to be applicable to information needs involving more than one component statement.

An early common belief was that the logical implication needed to capture relevance was not the classical material implication. The reasons why the use of the classical material implication is not appropriate for IR is in the definition of material implication itself (see the historical background section). The idea that a non-classical form of logical implication was needed for defining relevance was proposed by Van Rijsbergen in the form of the *logical uncertainty principle*, which is defined as follows:

- ► "Given any two sentences *x* and *y*; a measure of the uncertainty of *y* → *x* related to a given data set is determined by the minimal extent to which we have to add information to the data set, to establish the truth of *y* → *x*."

This principle made an explicit connection between non-classical logics and IR uncertainty modelling. However, when proposing the above principle, Van Rijsbergen was not specific about which logic and which uncertainty theory to use. As a consequence, various logics and uncertainty theories have been proposed and investigated. The choice of the appropriate logic and uncertainty mechanisms has been central to logical IR modeling, leading to a number of different approaches (see the Foundations section).

Historical Background

Relevance is one of the most important, if not "the fundamental," concept in the theory of IR. The concept arises from the consideration that if a user of an IR system has an information need, then some information stored in some documents in a document collection may be "relevant" to this need.

A logical definition of relevance was considered for the first time in the context of IR by Cooper in 1971 [4]. For Cooper, *logical relevance* was another name for topic-appropriateness, and he addressed the problem of giving a definition of logical relevance for IR by analogy with the same problem in question-answering systems. The analogy goes only as far as having questions with a yes-no (true-false) type of answer, and while Cooper's work started by analyzing question-answering systems, later he abandoned the analogy. Relevance is defined by Cooper as "logical consequence." To make this possible both queries and documents need to be represented by sets of declarative sentences. In the case of a yes-no query, the query is represented by two formal statements of the form *p* and ¬*p*. The two statements representing the query are called "component statements." A subset of the set of stored sentences is called "premiss set" if and only if the component statement is a logical consequence of that subset. A "minimal premiss set" for a component statement is one that is as small as possible in the sense that if any of its members were deleted, the component statement would no longer be a logical consequence of the set. Logical relevance is defined as a two-place relation between stored sentences and the query represented as component statements (see definitions section).

This definition of relevance is essentially just a proof-theoretic notion that has been generalised to be applicable to information needs involving more than one component statement. Although logical relevance was initially defined only for sentences, it can be easily extended to apply to stored documents: a document is relevant to an information need if and only if it contains at least one sentence which is relevant to that need.

Cooper also attempted to tackle a generalization of such a definition to natural language queries and documents. However, without a formalized language, no precise definition of the logical consequence relation is at hand, and thus one loses a precise definition of relevance. The problems of ambiguity and vagueness of natural language deny the possibility of extending the previous logical notion of relevance, despite the fact that the general idea of implication in natural language is a reasonably clear one. The definition of relevance, so far as natural language in concerned, is only a definition-in-principle – a conceptual definition – but not yet defined on a mathematical level.

Finally, Cooper also tried to tackle the problem of having "degrees of relevance," or as he wrote: "shades of grey instead of black and white." The idea was to extend the system of deductive reasoning used to access logical relevance to a system of plausible reasoning. Cooper argued that plausible or probabilistic inference was not as well defined as deductive inference, even for formalized languages. However, he added that when such tools are formalized enough then this development would become a "sensible and indeed inescapable idea," because it would enable the ranking of documents according to an estimated probability of relevance. What he proposed was to assign a higher probability of relevance to a sentence or a document that has greater probability of belonging to a residual minimal premiss set.

Cooper was the first to associate the topic-appropriateness sense of relevance with logical implication and recognized the importance of evaluating the uncertainty of such implication to rank documents in relation to their estimated measure of relevance. Many other researchers followed this idea proposing the use of different logics to capture relevance. In fact, the use of logic to build IR models enables one to obtain models that are more general than earlier well known IR models. Indeed, some logical models are able to represent within a uniform framework various features of IR systems, such as hypermedia links, multimedia content, users knowledge, cross-lingual, and structured documents. It also provides a common approach to the integration of IR systems with logical database systems. Finally, logic makes it possible to reason about an IR model and its properties. This latter possibility is becoming increasingly important since conventional evaluation methods, although good indicators of the effectiveness of IR systems, often give results which cannot be predicted, or satisfactorily explained.

An early common belief was that the logical implication needed to capture relevance was not the classical material implication. The reasons why the use of the classical material implication $d \supset q$ is not appropriate for IR is in the definition of material implication itself, and there are many ways of explaining why material implication is not suitable for IR. For reasons of brevity, these arguments will not be repeated here. It suffices to say that the material implication is acceptable only for the Boolean model of IR , where it expresses the "modus ponens." In the Boolean model if one observes d (that is if d is true) and if $d \rightarrow q$ is true (that is if all the query terms are all in the documents) then q is also true and thus one retrieves d. Only in this case if $d \rightarrow q$ is equivalent to $d \supset q$. In all other cases this would not hold ant it would necessary to find a way to define a suitable notion of implication that measures the extent in which the implication holds.

More recently, the thought that logic by itself could not fully model IR started to find a number of followers. In fact, in determining the relevance of a document to a query, the success or failure of an implication relating the two is not enough. It is necessary to take into account the *uncertainty* inherent in such an implication. The introduction of uncertainty can also be motivated from the consideration that a collection of documents cannot be regarded as a consistent and a complete set of statements. In fact, documents in the collection could and often do contradict each other in any particular logic, and not all the necessary knowledge is available. To cope with uncertainty a logic for *uncertain inference* was introduced. In fact, if $d \rightarrow q$ is uncertain, then one can measure its degree of uncertainty by $P(d \rightarrow q)$.

In 1986, Van Rijsbergen proposed the use of a non-classical conditional logic for IR [16]. This would enable the evaluation of $P(d \rightarrow q)$ using the *logical uncertainty principle* (see definitions). This principle was the first attempt to make an explicit connection between non-classical logics and IR uncertainty modelling. However, when proposing the above principle, Van Rijsbergen was not specific about which logic and which uncertainty theory to use. As a consequence, various logics and uncertainty theories have been proposed and investigated. This lead to a number of different approaches being proposed over the years.

Foundations

In the following, a brief overview of the most important logical models of IR is provided. It divides them into three classes: local models, local-uncertainty models, and meta-models.

Logical Models

The best known class of logical models of IR is that of the Boolean models, but their inability to capture the uncertainty inherent in the IR process (except by some extension of these models) has always confined them to the margins of modern IR.

Some logical models are able to capture the uncertainty of IR process, mainly in two ways: qualitatively by the logic itself (for example, via default rules, nonmonotonicity, or background conditions), or quantitatively by adding an uncertainty theory to the logic (for example, fuzzy logic). A first example of this class of models is represented by models based on Modal Logic and Conceptual Graphs. Modal Logic adopts the notion of possible worlds that correspond to the interpretations in classical logic, but which are connected to each other via an accessibility relation. The evaluation of the truth of a proposition is with respect to a possible world, and may involve the evaluation of the truth of the proposition in connected worlds. *Modal Logic* was first used to develop a logical model for IR by Nie [12]. Documents are worlds, and queries are formulae. A document represented by a world d is relevant to a query represented by a formula q if q is "true" in d, or if it is true in a world d' accessible from d. The accessibility relation captures the transformation of documents; the fact that the world d is connected to the world d' is interpreted as d being transformed into d'. The accessibility relationship can have different properties. For example, transitivity, meaning that if a world d is related to a world d', which is itself related to a world d'', then the world d is also related to the world d''. Consider the example of a hypertext system. Using Modal Logic, worlds can represent texts (nodes) and the accessibility relation can represent the links between the texts. The model also allows the transformation of both the query and the data set. One query can be transformed into another one using, for example, thesaural information. Query transformation is not a new approach in IR (e.g., query expansion). The novelty is that the transformation process can be formally represented, and hence reasoned upon. Transforming a data set can capture the modeling of a user's state in the retrieval process. The data set can be transformed until it reaches one that reflects the user's state.

Another set of logical models belonging are based on Situation Theory and the related Channel Theory. Situation Theory is a theory of information that provides an analysis of the concept of information and the manner in which cognitive agents handle and respond to the information picked up from their environment. The theory defines the nature of information flow and the mechanisms that give rise to such a flow. A document is a situation s and the query is a type ϕ. The document is relevant to the query if there exists a flow of information from a situation s to a situation s' such that $s' \models \phi$. The nature of the flow depends on the so-called "constraints" which capture semantic relationships. Flows of information do not always materialize because of the unpredictable nature of situations, thus flows are often uncertain. In Situation Theory, an uncertain flow is modeled by a conditional constraint of the form $\phi \rightarrow \psi | B$, which highlights the fact that $\phi \rightarrow \psi$ holds if some background conditions captured within B are met. If the background conditions are satisfied, the corresponding flow arises. However, it is often the case that two situations are systematically related to each other, by way of a flow of information. Therefore, in addition to constraints, there are relationships that link situations. The concept of a channel is introduced to express the relationships, by way of an information flow, between two situations. *Channel theory* defines formally channels, together with the mathematical properties that support the flow of information. The use of Situation and Channel theory to model IR has been investigated in [17], where the connection between IR, logic, probability and information containment was made. It was indicated that the use of channels present many potentials for theoretical IR modelling because they can apply to various IR processes present in advanced IR systems.

A successful class of logical models is based on *Terminological logics*. This family of logics come from the area of artificial intelligence, in particular, knowledge representation. Terminological logics derive from a large group of knowledge representation language (such as, for example, KL-ONE) based on semantic networks and inspired by the notion of frames. They provide object-oriented flavored representations. The use of terminological logic for IR was proposed in [11]. There, documents are represented by individual constants, whereas a class of documents is represented as a concept. Queries are described as concepts. Given a query represented by a concept Q, the retrieval task is to find all those documents d such that $Q[d]$ holds. The evaluation of $Q[d]$ uses the set of assertions describing documents, that is, one is not evaluating whether $d \rightarrow q$, but rather whether individual d is an instance of the class concept Q.

An important class of logical models is based on Belief Revision. *Belief Revision* provides a way to formalize changes done to a knowledge base after the arrival of new information. The use of belief revision in IR was attempted in [10] as a way to compute the similarity of a document to a query for retrieval purpose. The Dalal's revision operator was chosen for implementing the belief revision process, since it provides an order among proposition interpretations, where propositions model index terms. The ordering is used to formulate a similarity measure between a document and a query (expressed as formulae) based on the number of revisions necessary for the document to "reach" the query. It should be noted that both documents and queries can have, each, several interpretations, so normalization becomes necessary.

Finally, there is the class of models based on *Fuzzy logic*. This is a formal framework that is well suited to model vagueness and imprecision. In IR it has been successfully employed at several levels in particular for the definition of a superstructure of the Boolean model, with the appealing consequence that existing Boolean IR systems can be improved without redesigning them completely [2]. Through these extensions the gradual nature of relevance of documents to user queries can be modeled.

Logical-Uncertainty Models

Logical-uncertainty models are based on an uncertainty theory (for instance, probability theory, semantic theory, imaging) that is defined on a logical basis. They enable a more complex definition of relevance than other IR models (than probabilistic relevance model, for instance, which are based mainly upon statistical estimations of the probability of relevance). With logical-uncertainty models, information not present in the query formulation may be included in the evaluation of the relevance of a document. Such information might be domain knowledge, knowledge about the user, user's relevance feedback, or other.

The main example of logical-uncertainty models of IR is represented by models based on Probability Theory. In IR, probabilistic modeling refers to the use of a model that ranks documents in decreasing order of their evaluated probability of relevance to a user's information need. Past and present research has made use of formal theories of probability and of statistics in order to evaluate, or at least estimate, those probabilities of relevance. These attempts are to be distinguished from looser ones like, for example, the vector space model in which documents are ranked according to a measure of similarity with the query. A measure of similarity cannot be directly interpreted as a probability. In addition, similarity based models generally lack the theoretical soundness of probabilistic models. A treatment of models based on Probability Theory is beyond the scope of this entry and is reported in the relevant entries; the models presented in this section are based on the idea that IR is a process of uncertain inference.

A probabilistic formalism for describing inference relations with uncertainty is provided by *Bayesian inference networks*. Turtle and Croft [15] applied such networks to IR. Nodes represent IR entities such as documents, index terms, concepts, queries, and information needs. One can choose the number and kind of nodes one wishes to use according to how complex one wants the representation of the document collection or the information needs to be. Arcs represent probabilistic dependencies between entities and represent conditional probabilities, that is, the probability of an entity being true given the probabilities of its parents being true.

In a Bayesian inference network, the truth value of a node depends only upon the truth values of its parents. To evaluate the strength of an inference chain going from one document to the query the document node d_i is set to "true" and $P(q_k = true \mid d_i = true)$ is evaluated. This gives an estimate of $P(d_i \rightarrow q_k)$. It is possible to implement various traditional IR models on this network by introducing nodes representing Boolean operators or by setting appropriate conditional probability evaluation functions within nodes.

Another important set of examples of logical-uncertainty models is represented by models based on *Probabilistic Datalog*. Datalog is a predicate logic that has been developed in the database field, and makes the link between the relational model and rule-based systems. *Probabilistic Datalog* is the probabilistic extension of Datalog. Probabilistic Datalog is not itself a logical IR framework, but more a platform in which logical probabilistic IR models, as well as other IR models can be expressed. One of the major assets of Probabilistic Datalog is that, since it is a generalization of the Datalog model, it can be used as standard query language for both database and IR. It can then deal with both structured data (as in database) and unstructured data (as in IR) within the same system. It also allows the uniform representation, retrieval and querying of content, fact and structural knowledge [8]. The work has been further extended via the development, implementation and evaluation of the POOL (Probabilistic Object-Oriented Logic) model, which allows the representation of inconsistency (using then a four-valued logic), the modeling of the document and query representation using the object-oriented paradigm.

A very interesting example of logical-uncertainty models and certainly the one that more reflects Van Rijsbergen's Logical Uncertainty Principle is that of models based on Logical Imaging. *Logical imaging* is an approach that defines the probability of conditional $P(d \rightarrow q)$ based on the notion of possible-worlds. In this approach the possible worlds (e.g., retrieval situation, document representation) are spanned by an accessibility relation defined in terms of similarity. The truth value of the implication $p \rightarrow q$ in a world w depends on two cases. If p is true in w, then $p \rightarrow q$ is true (false) in that world if q is also true (false) in that world. However, if p is not true in w, then the implication is evaluated in the worlds that differ minimally from w and in which p is true. The worlds in which p is true are referred to as p-worlds. The set of worlds comes with a probability distribution P, reflecting the probability of each world. The probability of a proposition p is the summation of the probability of those worlds in which p is true. The computation of the probability of $p \rightarrow q$ involves a shift of probability (the imaging process) from non-$p - worlds$ to their closest p-worlds. It can be proved that: $P(p \rightarrow q) = P_p(q)$, where P_p is a new probability distribution, a "posterior probability," derived from P by imaging on p. Therefore, conditioning by imaging causes a revision of the prior probability on the possible worlds w in such a way that the posterior probability is obtained by shifting the original probabilities from non-p-worlds to p-worlds. Each non-p-world moves its probability to its closest p-world (or set of p-worlds in the case of general imaging). Bayesian conditioning, on the other hand, is obtained by cutting off all non-p-worlds and then proportionally magnifying the probabilities of the p-worlds so that the posterior probabilities still add up to one, as required by Probability Theory. The magnification is done in the same way for every p-world, thus keeping constant the ratios between the probabilities assigned to these worlds. It is therefore clear that imaging and Bayesian conditionalization yield, in general, different results. Since the transfer of probabilities is directed towards the closest p-worlds, this technique is just what it is needed to implement Van Rijsbergen's logical uncertainty principle.

Two logical-probabilistic IR models have been developed on the concept of imaging. In the first one [7], worlds model terms, and propositions model documents and queries. A term t "makes a document true" if that term belongs to that document. Imaging with respect to d gives the closest term to t that is contained in d. Of course, this is t itself if t is contained in the document. Imaging consists then of shifting the probabilities from term not contained in d to the terms contained in d (i.e., the terms that make d true). The evaluation of the relevance takes into account the semantics between terms by shifting probabilities to those (semantically) closer terms contained in the document. A second model based on imaging includes user's knowledge in the evaluation of the relevance of a document to a query [13]. In this model both documents and queries are propositions. Possible worlds represent different states of the data set, for example possible states of knowledge that can be held by users. A document d is true in a world w if the document is "consistent" (the term is used here in a broad sense) with the state of knowledge associated with that world. Worlds differ because they represent different states of knowledge and, given a metric on the world space, one can identify the closest world to w for which d is true.

Other logical-probabilistic models are based on *Semantic Information Theory*. The work on Semantic

Information Theory in IR concerns two research directions: the axiomatization of the logical principles for assigning probabilities or similar weighting functions to logical sentences and the relationship between information content of a sentence and its probability [1].

Meta-Models

Meta-models are a completely different class of models from those presented earlier. Meta-models attempts to formally study the properties and the characteristics of IR systems within a uniform logical framework. The advantage is that they make it possible to compare IR systems not only with respect to their effectiveness, but also with respect to formal properties of the underlying models.

The use of logic to formally conduct proofs for IR purposes was thoroughly investigated in [3], where a framework was proposed in which different models of IR could be theoretically expressed, formally studied and compared. The framework was developed within a logic, thus allowing formal proofs to be conducted. The framework defines the aboutness relationship, denoted $\models$, which aims at capturing the notion of information containment primary to IR. Given two objects *a* and *b*, $a \models b$ means that object *a* is about object *b*. Axioms are defined that represent possible properties of IR systems. Examples of simple axioms include reflexivity, symmetry, and transitivity, but more complex ones have also been identified, like for example weakening and monotonicity, borrowed from non-monotonic reasoning. This research has led to the theoretical comparison of IR models. The IR models are mapped down into a logic-based framework and they are compared by looking at the particular aboutness properties they each embody. Through this research, IR has gained a clearer understanding of what aboutness is, and what properties are desirable and not desirable.

Due to space limitations this entry cannot even briefly report here on other logical models of IR. A good survey of a number of these models can be found in [5]. An analysis of the strengths and limitations of logical modelling of IR can be found in [9,14]. Finally, many of the models briefly introduced here are described in details in [6].

Key Applications

The main application area of the models presented in this section is Information Retrieval.

Future Directions

This entry reviewed a number of approaches to logical modelling of IR. So far, no consensus has been reached regarding what the best approach is due to the small number of operational systems based on these approaches. It is hoped that further investigations into various logic-based frameworks accompanied by more evaluation might lead to a unified information-based model theory for expressing the semantics of information retrieval. Such a theory will enable to predict the behavior of IR systems, compare them and prove properties about them. This would be a major strength of logical models of IR.

Cross-references

▶ Fuzzy Logic
▶ Probability Theory
▶ Relevance

Recommended Reading

1. Amati G. and van Rijsbergen C.J. Semantic information retrieval. In Information Retrieval: Uncertainty and Logics, F. Crestani, M. Lalmas, C.J. van Rijsbergen (eds.). Kluwer, Norwell, MA, USA, 1998, pp. 189–220.
2. Bordogna G. and Pasi G. A fuzzy linguistic approach generalizing boolean information retrieval: a model and its evaluation. J. Am. Soc. Inf. Sci., 44(2):70–82, 1993.
3. Bruza P.D. and Huibers T.W.C. Investigating aboutness axioms using information fields. In Proc. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1994, pp. 112–121.
4. Cooper W.S. A definition of relevance for information retrieval. Inf. Storage Retr., 7:19–37, 1971.
5. Crestani F. and Lalmas M. Logic and uncertainty in information retrieval. In Lectures on Information Retrieval, M. Agosti, F. Crestani, G. Pasi (eds.). LNCS, vol. 1980. Springer, Heidelberg, Germany, 2001, pp. 182–210.
6. Crestani F., Lalmas M., and van Rijsbergen C.J. (eds.). Information Retrieval, Uncertainty and Logics: Advanced Models for the Representation and Retrieval of Information. Kluwer, MA, USA, 1998.
7. Crestani F. and van Rijsbergen C.J. A study of probability kinematics in information retrieval. ACM Trans. Inf. Syst., 16(3):225–255, 1998.
8. Fuhr N. Probabilistic datalog – a logic for powerful retrieval methods. In Proc. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 282–290.
9. Lalmas M. Logical models in information retrieval: introduction and overview. Inf. Process. Manage., 34(1):19–33, 1998.
10. Losada D.E. and Barreiro A. Using a belief revision operator for document ranking in extended boolean model. In Proc.

ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 66–73.

11. Meghini C., Sebastiani F., Straccia U., and Thanos C. A model of information retrieval based on a terminological logic. In Proc. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 298–307.
12. Nie J.Y. An information retrieval model based on modal logic. Inf. Process. Manage., 25(5):477–491, 1989.
13. Nie J.Y. Lepage F., and Brisebois M. Information retrieval as counterfactuals. Comput. J., 38(8):643–657, 1995.
14. Sebastiani F. Trends in … a critical review: on the role of logic in information retrieval. Inf. Process. Manage., 34 (1):1–18, 1998.
15. Turtle H.R. and Croft W.B. Evaluation of an inference network-based retrieval model. ACM Trans. Inf. Syst., 9(3):187–222, July 1991.
16. van Rijsbergen C.J. A new theoretical framework for information retrieval. In Proc. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1986, pp. 194–200.
17. van Rijsbergen C.J. and Lalmas M. An information calculus for information retrieval. J. Am. Soc. Inf. Sci., 47(5):385–398, 1996.

Logical Query Processing and Optimization

► Query Processing in Deductive Databases

Logical Schema Design

► Logical Database Design: From Conceptual to Logical Schema

Logical Story Unit Segmentation

► Video Segmentation

Logical Structure

Thijs Westerveld
Teezir Search Solutions, Ede, The Netherlands

Definition

Logical structure refers to the way information in a document is organized; it defines the hierarchy of information and the relation between different parts of the document. Logical structure indicates how a document is built, as opposed to what a document contains.

Key Points

Logical structure is mainly used in the context of XML document to distinguish the logical organization of the content from its physical organization, to distinguish the flow of content from the documents layout and from the presentation of the document. In XML documents, text, images and metadata can be organized in a meaningful, logical manner independently of the document's layout when presented to a user. This contrasts with HTML documents in which logical organization and layout necessarily are the same. The logical structure of a document is of particular interest in structured document retrieval, where it may provide knowledge regarding the organization of information which may lead to better identification of relevant document parts given a user's request.

Cross-references

► Processing Structural Constraints
► Structured Text Models
► XML Retrieval

Logical Time

► Valid Time

Logical Unit Number

Kaladhar Voruganti
Network Appliance, Sunnyvale, CA, USA

Synonyms

LUN; Volume

Definition

LUN is a SCSI protocol term that refers to a logically contiguous piece of storage. This usually corresponds to a volume in a storage array or an individual disk drive on a parallel SCSI bus. The storage volume identifier gets mapped to a LUN by the host operating system and they are usually different.

Key Points

A volume on a storage controller can be assigned different LUN numbers on different hosts. A volume on a storage controller can be partitioned into multiple volumes, and thus, LUNs by storage virtualization software. The storage virtualization software can reside on hosts or network virtualization boxes.

Cross-references

- ▶ LUN Mapping
- ▶ Volume

Logical Unit Number Mapping

Kaladhar Voruganti
Network Appliance, Sunnyvale, CA, USA

Synonyms

LUN mapping

Definition

This is the process by which the host operating system assigns a LUN value to a particular storage volume. LUN Mapping is typically used in cases where the higher level applications require specific LUN numbers for specific storage devices. When there are multiple paths between a SCSI initiator and SCSI target, multi-pathing software is used to properly map target volumes via both the paths. Storage controllers provide access control mechanisms (LUN Masking) that can control how initiators access target storage volumes.

Key Points

At host, LUN mapping is performed by operating system software. If one wants to properly manage multiple paths to a storage volume from a host, then one needs to install a multi-pathing driver on the host. LUN mapping is also performed by virtualization software that is present in network virtualization boxes. These boxes perform mapping between virtual volumes that are exported to hosts and the physical volumes residing on the storage controllers.

Cross-references

- ▶ LUN
- ▶ Multi-Pathing
- ▶ Volume

Logical Volume

- ▶ Volume

Logical Volume Manager

Kenichi Wada
Hitachi Limited, Tokyo, Japan

Synonyms

Virtual disk manager; Volume set manager; LVM

Definition

Logical Volume Manager (LVM) is a kind of storage virtualization. LVM collects one or more disk drives or partitions, creating a storage pool in a host. Furthermore, LVM provides applications with logical volumes consisting of multiple chunks of disk drives or partitions.

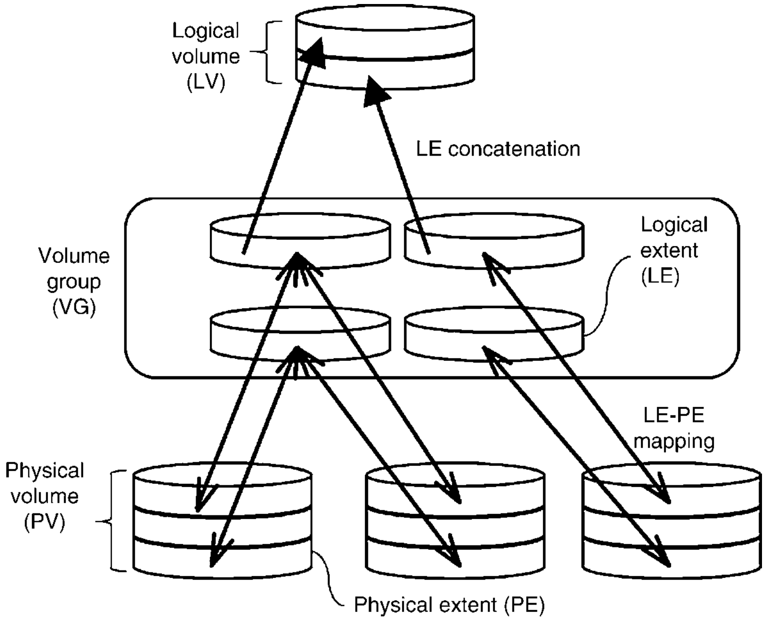
Key Points

Figure 1 shows volume manager implementation. LVM implementations often start with physical volume (PVs), which can be disk drives or partitions. PVs are spilt into chunks called physical extents (PEs). Some LVM implementations have PEs of a uniform size; others have variable-sized PEs that can be split and merged at will.

A logical extent (LE) consists of one or more PEs. For example, users can create an LE which is simply mapped to an PE, or create a striped LE which consists of multiple PEs with a conventional RAID technique. LEs are pooled into a volume group (VG).

A logical volume (LV) consists of one or more LEs in a VG. Users also can create an LV which is simply mapped to an LE, or create a striped LV which consists of multiple LEs with a conventional RAID technique.

In some implementations, LVs can be grown or shrunken by concentrating more LEs from, or returning them to, a VG. Some volume managers allow LVs



Logical Volume Manager. Figure 1. Implementation Architecture.

to be resized in either direction while online. These features help users to create flexible system configurations easily.

Cross-references

- ▶ Disk
- ▶ Redundant Arrays of Independent Disks
- ▶ Storage Management

Logical Window

- ▶ Windows

Log-Linear Regression

Jialie Shen
Singapore Management University, Singapore, Singapore

Definition

In statistic, log-linear regression is a powerful regression technique that models relationship between a dependent variable or regressand Y, explanatory variable or regressor $X = \{x_1,\ldots,x_I\}$ and a random term ε by fitting a log-linear model,

$$\ln Y = \alpha_0 + \alpha_1 \ln x_1 + \alpha_2 \ln x_2 + \ldots. + \alpha_I \ln x_I + \varepsilon$$

where α_0 is the constant term, the α_i s are the respective parameters of independent variables, and I is the number of parameters to be estimated in the log-linear regression.

Key Points

The goal of log-linear regression is to explore effect a set of covariance $X = \{x_1,\ldots,x_I\}$ on the expected rate [1]. It assumes that a linear relationship exists between the log of the regressand Y and the regressor $X = \{x_1,\ldots,x_I\}$. The log-linear regression is appropriate for categorical variables.

When applied to continuous variables, discretization is an essential step for preprocessing. There are two basic steps to using log-linear regression: (i) determining how many factors to be considered and sets of attributes related to each factor and (ii) estimating the numerical values of the parameters.

Cross-references

- ▶ Data Reduction
- ▶ Generalized Linear Models
- ▶ Linear Regression

Recommended Reading

1. McCullagh P., Nelder J. 1Generalized Linear Models, CHAPMAN & HALL/CRC, London, 1989.

Long Running Queries

▶ Continuous Queries in Sensor Networks

Longitudinal Health Record

▶ Electronic Health Records (EHR)

Looking Over/Through

▶ Browsing in Digital Libraries

Loop

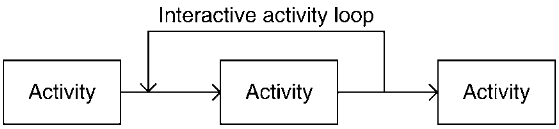
Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

Iteration; Workflow loop; While loop

Definition

A workflow activity cycle involving the repetitive execution of one (or more) workflow activity(s) until a condition is met.



Key Points

A Loop is a control flow statement that allows an activity or activity block to be executed repeatedly, rather than proceeding to the next activity, until a specific condition has been met. Following the execution of the activity (or last activity in the block) the condition of the loop is evaluated.

Cross-references

▶ Activity
▶ Process Life Cycle
▶ Workflow Management and Workflow Management System

Loop Join

▶ Nested Loop Join

Loose Coupling

Serguei Mankovskii
CA Labs, CA, Inc., Thornhill, ON, Canada

Synonyms

Weak coupling; Low coupling

Definition

Word coupling refers to a notion of interdependence between components of software systems. Word loose refers to a mode of coupling where components possess significant degree of autonomy.

Key Points

Term loose coupling consists of two opposite notions of dependency and autonomy. It initially emerged out of study of educational organizations. Organizational scientist Karl Weick noticed that some organizations can be mechanistic (coupled) and organic (loose) at the same time. Loosely coupled organizations are under influence of forces or conditions that hold them together, but at the same time they are acting with the high degree of independence.

In software engineering, loose coupling has similar notion to organizational theory. A loosely coupled system usually consists of a framework or architecture linking together otherwise independent components. The components are usually highly cohesive and have well defined responsibilities. Cohesive components allow for creation of stable interfaces exposing less implementation details through the component interface. Loosely coupled system uses stable interfaces of the components to hold the system together.

As a design, goal loose coupling is desirable when system components need to be developed or used independently of each other and when they need to change over time. This goal can be achieved by deliberate decoupling of system components. De-coupling leads to a more reliable system design and reduces maintenance costs over the life time of system and its components.

Software components developed for loose coupling can be readily re-purposed for use in a broad class of systems leading to higher re-use. They can also be used at the same time by a broad class of systems through standards-based interfaces. In this case they are often called services. Service Oriented Architecture is a special case of software systems built around of loosely coupled services.

Cross-references

▶ Cohesion
▶ Coupling and De-Coupling
▶ Interface
▶ Re-Use
▶ Service Oriented Architecture

Recommended Reading

1. Wieck K.E. Educational organisations as loosely coupled systems. Adm. Sci. Q., 21:1–19, 1976.

Lossless Data Compression

▶ Text Compression

LoT-RBAC

▶ GEO-RBAC Model/ExternalRef>

Low Coupling

▶ Loose Coupling

Lp Distances

▶ Frequency Moments

Lp Norms

▶ Frequency Moments

LSID

▶ Semantic Data Integration for Life Science Entities

LUN

▶ Logical Unit Number (LUN)

LUN Mapping

▶ Logical Unit Number Mapping

LUN Masking

▶ Storage Security

LVM

▶ Logical Volume Manager

MAC

▶ Message Authentication Codes

Machine Learning in Bioinformatics

▶ Machine Learning in Computational Biology

Machine Learning in Computational Biology

Cornelia Caragea, Vasant Honavar
Iowa State University, Ames, IA, USA

Synonyms

Data mining in computational biology; Data mining in bioinformatics; Machine learning in bioinformatics; Machine learning in systems biology; Data mining in systems biology

Definition

Advances in high throughput sequencing and "omics" technologies and the resulting exponential growth in the amount of macromolecular sequence, structure, gene expression measurements, have unleashed a transformation of biology from a data-poor science into an increasingly data-rich science. Despite these advances, biology today, much like physics was before Newton and Leibnitz, has remained a largely descriptive science. Machine learning [6] currently offers some of the most cost-effective tools for building predictive models from biological data, e.g., for annotating new genomic sequences, for predicting macromolecular function, for identifying functionally important sites in proteins, for identifying genetic markers of diseases, and for discovering the networks of genetic interactions that orchestrate important biological processes [3]. Advances in machine learning e.g., improved methods for learning from highly unbalanced datasets, for learning complex structures of class labels (e.g., labels linked by directed acyclic graphs as opposed to one of several mutually exclusive labels) from richly structured data such as macromolecular sequences, three-dimensional molecular structures, and reliable methods for assessing the performance of the resulting models, are critical to the transformation of biology from a descriptive science into a predictive science.

Historical Background

Large scale genome sequencing efforts have resulted in the availability of hundreds of complete genome sequences. More importantly, the GenBank repository of nucleic acid sequences is doubling in size every 18 months [4]. Similarly, structural genomics efforts have led to a corresponding increase in the number of macromolecular (e.g., protein) structures [5]. At present, there are over a thousand databases of interest to biologists [16]. The emergence of high-throughput "omics" techniques, e.g., for measuring the expression of thousands of genes under different perturbations, has made possible system-wide measurements of biological variables [8]. Consequently, discoveries in biological sciences are increasingly enabled by machine learning.

Some representative applications of machine learning in computational and systems biology include: identifying the protein-coding genes (including gene boundaries, intron-exon structure) from genomic DNA sequences; predicting the function(s) of a protein from its primary (amino acid) sequence (and when available, structure and its interacting partners); identifying functionally important sites (e.g., protein-protein, protein-DNA, protein-RNA binding sites, post-translational modification sites) from the protein's amino acid sequence and, when available, from the protein's structure; classifying protein sequences (and structures) into structural classes; Identifying functional modules (subsets of genes that function together) and genetic networks from gene expression data.

These applications collectively span the entire spectrum of machine learning problems including supervised learning, unsupervised learning (or cluster analysis), and system identification. For example, protein function prediction can be formulated as a supervised learning problem: given a dataset of protein sequences with experimentally determined function labels, induce a classifier that correctly labels a novel protein sequence. The problem of identifying functional modules from gene expression data can be formulated as an unsupervised learning problem: given expression measurements of a set of genes under different conditions (e.g., perturbations, time points), and a distance metric for measuring the similarity or distance between expression profiles of a pair of genes, identify clusters of genes that are co-expressed (and hence are likely to be co-regulated). The problem of constructing gene networks from gene expression data can be formulated as a system identification problem: given expression measurements of a set of genes under different conditions (e.g., perturbations, time points), and available background knowledge or assumptions, construct a model (e.g., a boolean network, a bayesian network) that explains the observed gene expression measurements and predicts the effects of experimental perturbations (e.g., gene knockouts).

Foundations

Challenges presented by computational and systems biology applications have driven, and in turn benefited from, advances in machine learning. Some of these developments are described below.

Multi-Label Classification: In the traditional classification problem, an instance $\mathbf{x}_i$, i = 1,...,n, is associated with a single class label y_j from a finite, disjoint set of class labels Y , j = 1,...,k, k = |Y| (*single-label classification problem*). If the set Y has only two elements, then the problem is referred to as the *binary classification problem*. Otherwise, if Y has more than two elements, then it is referred to as *multi-class classification problem*. However, in many biological applications, an instance $\mathbf{x}_i$ is associated with a subset of, not necessarily disjoint, class labels in Y (*multi-label classification problem*). For example, many genes and proteins are multi-functional. Most of the existing algorithms cannot simultaneously label a gene or protein with several, not necessarily mutually exclusive functions. Each instance is then assigned to a subset of nodes in the hierarchy, yielding a *hierarchical multi-label classification problem* or a *structured output classification problem*. The most common approach to dealing with *multi-label classification problem* [7] is to transform the problem into k binary classification problems, one for each different label $y_j \in Y$, j = 1,...,k. The transformation consists of constructing k datasets, D_j, each containing all instances of the original dataset, such that an instance in D_j, j = 1,...,k, is labeled with 1 if it has label y_j in the original dataset, and 0 otherwise. During classification, for a new unlabeled instance $\mathbf{x}_{test}$, each individual classifier C_j,j = 1,...,k, returns a prediction that $\mathbf{x}_{test}$ belongs to the class label y_j or not. However, the transformed datasets that result from this approach are highly unbalanced, typically, with the number of positively labeled instances being significantly smaller than the number of negatively labeled instances, requiring the use of methods that can cope with unbalanced data. Alternative evaluation metrics need to be developed for assessing the performance of multi-label classifiers. This task is complicated by correlations among the class labels.

Learning from Unbalanced Data: Many of the macromolecular sequence classification problems present the problem of learning from highly *unbalanced* data. For example, only a small fraction of amino acids in an RNA-binding protein binds to RNAs. Classifiers that are trained to optimize accuracy generally perform rather poorly on the minority class. Hence, if accurate classification of instances from the minority class is important (or equivalently, the false positives and false negatives have unequal costs or risks associated with them), it is necessary to change the distribution of positive and negative instances *during training* by randomly selecting a subset of the training data for the majority class, or alternatively, assigning different *weights* to positive and negative samples (and learn from the resulting weighted samples). More recently, *ensemble classifiers* [11] have been shown to improve the performance of sequence classifiers on unbalanced datasets. Unbalanced datasets also complicate both the training and the assessment of the predictive performance of classifiers. *Accuracy* is not a useful performance measure in such scenarios. Indeed, no single performance measure provides a complete picture of the classifier's performance. Hence, it is much more useful to examine ROC (Receiver Operating Characteristic) or precision-recall curves [3]. Of particular interest are methods that can directly optimize alternative performance measures that take into account the unbalanced nature of the dataset and user-specified tradeoff between false positive and false negative rates.

Data Representation: Many computational and systems biology applications of machine learning present challenges in data representation. Consider for example, the problem of identifying functionally important sites (e.g., RNA-binding residues) from amino acid sequences. In this case, given an amino acid sequence, the classifier needs to assign a binary label (1 for an RNA-binding residue and 0 for a non RNA-binding residue) to each letter of the sequence. To solve this problem using standard machine learning algorithms that work with a fixed number of input features, it is fairly common to use a *sliding window* approach [12] to generate a collection of fixed length windows, where each window corresponds to the target amino acid and an equal number of its sequence neighbors on each side. The classifier is trained to label the target residue. Similarly, identifying binding sites from a three-dimensional structure of the protein requires transforming the problem into one that can be handled by a traditional machine learning method. Such transformations, while they allow the use of existing machine learning methods on macromolecular sequence and structure labeling problems, complicate the task of assessing the performance of the resulting classifier (see below).

Performance Assessment: Standard approaches to assessing the performance of classifiers rely on k-fold cross-validation wherein a dataset is partitioned into k disjoint subsets (folds). The performance measure of interest is estimated by averaging the measured performance of the classifier on k runs of a cross-validation experiment, each using a different choice of the $k - 1$ subsets for training and the remaining subset for testing the classifier. The fixed length window representation described above complicates this procedure on macromolecular sequence labeling problems. The training and test sets obtained by random partitioning of the dataset of labeled windows can contain windows that originate from the same sequence, thereby violating a critical requirement for cross-validation, namely, that the training and test data be disjoint. The resulting overlap between training and test data can yield overly optimistic estimates of performance of the classifier. A better alternative is to perform sequence-based (as opposed to window-based) cross-validation by partitioning the set of sequences (instead of windows) into disjoint folds. This procedure guarantees that training and test sets are indeed disjoint [9]. Obtaining realistic estimates of performance in sequence classification and sequence labeling problems also requires the use of *non-redundant* datasets [13].

Learning from Sparse Datasets: In gene expression datasets, the number of genes is typically in the hundreds or thousands, whereas the number of measurements (conditions, perturbations) is typically fewer than ten. This presents significant challenges in inferring genetic network models from gene expression data because the number of variables (genes) far exceeds the number of observations or data samples. Approaches to dealing with this challenge require reducing the effective number of variables via variable selection [17] or abstraction i.e., by grouping variables into clusters that behave similarly under the observed conditions. Another approach to dealing with sparsity of data in such settings is to incorporate information from multiple datasets [18].

Key Applications

Protein Function Prediction: Proteins are the principal catalytic agents, structural elements, signal transmitters, transporters and molecular machines in cells. Understanding protein function is critical to understanding diseases and ultimately in designing new drugs. Until recently, the primary source of information about protein function has come from biochemical, structural, or genetic experiments on individual proteins. However, with the rapid increase in number of genome sequences, and the corresponding growth in the number of protein sequences, the numbers of experimentally determined structures and functional annotations has significantly lagged the number of protein sequences. With the availability of datasets of protein sequences with experimentally determined functions, there is increasing use of sequence or structural homology-based transfer of annotation from already annotated sequences to new protein sequences. However, the effectiveness of such homology-based methods drops dramatically when the sequence similarity between the target sequence and the reference sequence falls below 30%. In many instances, the function of a protein is determined by conserved local sequence motifs. However, approaches that assign function to a protein based on the presence of a single motif (the so-called characteristic motif) fail to take advantage of multiple sequence motifs that are correlated with critical structural features (e.g., binding pockets) that play a critical role in protein function. Against this background, machine learning methods offer an attractive approach to training classifiers to assign putative functions to protein sequences. Machine learning methods have been applied, with varying degrees of success, to the problem of protein function prediction. Several studies have demonstrated that machine learning methods, used in conjunction with traditional sequence or structural homology based techniques and sequence motif-based methods outperform the latter in terms of accuracy of function prediction (based on cross-validation experiments). However, the efficacy of alternative approaches in genome-wide prediction of functions of protein-coding sequences from newly sequenced genomes remains to be established. There is also significant room for improving current methods for protein function prediction.

Identification of Potential Functional Annotation Errors in Genes and Proteins: As noted above, to close the sequence-function gap, there is an increasing reliance on automated methods in large-scale genome-wide annotation efforts. Such efforts often rely on transfer of annotations from previously annotated proteins, based on sequence or structural similarity. Consequently, they are susceptible to several sources of error including errors in the original annotations from which new annotations are inferred, errors in the algorithms, bugs in the software used to process the data, and clerical errors on the part of human curators. The effect of such errors can be magnified because they can propagate from one set of annotated sequences to another. Because of the increasing reliance of biologists on reliable functional annotations for formulation of hypotheses, design of experiments, and interpretation of results, incorrect annotations can lead to wasted effort and erroneous conclusions. Hence, there is an urgent need for computational methods for checking consistency of such annotations against independent sources of evidence and detecting potential annotation errors. A recent study has demonstrated the usefulness of machine learning methods to *identify and correct* potential annotation errors [1].

Identification of Functionally Important Sites in Proteins: Protein-protein, protein-DNA, and protein-RNA interactions play a pivotal role in protein function. Reliable identification of such interaction sites from protein sequences has broad applications ranging from rational drug design to the analysis of metabolic and signal transduction networks. Experimental detection of interaction sites must come from determination of the structure of protein-protein, protein-DNA and protein-RNA complexes. However, experimental determination of such complexes lags far behind the number of known protein sequences. Hence, there is a need for development of reliable computational methods for identifying functionally important sites from a protein sequence (and when available, its structure, but not the complex). This problem can be formulated as a sequence (or structure) labeling problem. Several groups have developed and applied, with varying degrees of success, machine learning methods for identification of functionally important sites in proteins (see [21,14,22] for some examples). However, there is significant room for improving such methods.

Discovery and Analysis of Gene and Protein Networks: Understanding how the parts of biological systems (e.g., genes, proteins, metabolites) work together to form dynamic functional units, e.g., how genetic interactions and environmental factors orchestrate development, aging, and response to disease, is one of the major foci of the rapidly emerging field of systems biology [8]. Some of the key challenges include the following: uncovering the biophysical basis and essential macromolecular sequence and structural features of macromolecular interactions; comprehending how temporal and spatial clusters of genes, proteins, and signaling agents correspond to genetic, developmental and regulatory networks [10]; discovering topological and other characteristics of these networks [19]; and explaining the emergence of systems-level properties of networks from the interactions among their parts. Machine learning methods have been developed and applied, with varying degrees of success, in learning predictive models including boolean networks [20] and bayesian networks [15] from gene expression data. However, there is significant room for improving the accuracy and robustness of such algorithms by taking advantage of multiple types of data and by using active learning.

Future Directions

Although many machine learning algorithms have had significant success in computational biology, several challenges remain. These include the development of: efficient algorithms for learning predictive models from distributed data; cumulative learning algorithms that can efficiently update a learned model to accommodate changes in the underlying data used to train the model; effective methods for learning from sparse, noisy, high-dimensional data; and effective approaches to make use of the large amounts of unlabeled or partially labeled data; algorithms for learning predictive models from disparate types of data: macromolecular sequence, structure, expression, interaction, and dynamics; and algorithms that leverage optimal experiment design with active learning in settings where data is expensive to obtain.

Cross-references

▶ Biological Networks
▶ Biostatistics and Data Analysis
▶ Classification
▶ Clustering
▶ Data Mining
▶ Graph Database Mining

Recommended Reading

1. Andorf C., Dobbs D., and Honavar V. Exploring inconsistencies in genome-wide protein function annotations: a machine learning approach. BMC Bioinform., 8:284, 2007.
2. Ashburner M., Ball C.A., Blake J.A., Botstein D., Butler H., Cherry J.M., Davis A.P., Dolinski K., Dwight S.S., Eppig J.T., Harris M.A., Hill D.P., Issel-Tarver L., Kasarskis A., Lewis S., Matese J.C., Richardson J.E., Ringwald M., Rubin G.M., and Sherlock G. Gene ontology: tool for the unification of biology. Nat. Gene., 25:25–29, 2000.
3. Baldi P. and Brunak S. Bioinformatics: the machine learning approach. MIT, Cambridge, MA, 2001.
4. Benson D.A., Karsch-Mizrachi I., Lipman D.J., Ostell J., and Wheeler D.L. Genbank. Nucleic Acids Res., 35D (Database issue):21–D25, 2007.
5. Berman H.M., Westbrook J., Feng Z., Gilliland G., Bhat T.N., Weissig H., Shindyalov I.N., and Bourne P.E. The protein data bank. Nucleic Acids Res., 28:235–242, 2000.
6. Bishop C.M. Pattern Recognition and Machine Learning. Springer, Berlin, 2006.
7. Boutell M.R., Luo J., Shen X., and Brown C.M. Learning multi-label scene classification. Pattern Recogn., 37:1757–1771, 2004.
8. Bruggeman F.J. and Westerhoff H.V. The nature of systems biology. Trends Microbiol., 15:15–50, 2007.
9. Caragea C., Sinapov J., Dobbs D., and Honavar V. Assessing the performance of macromolecular sequence classifiers. In Proc. IEEE 7th Int. Symp. on Bioinformatics and Bioengineering, 2007, pp. 320–326.
10. de Jong H. Modeling and simulation of genetic regulatory systems: a literature review. J. Comput. Biol., 9:67–103, 2002.
11. Diettrich T.G. Ensemble methods in machine learning. Springer, Berlin, In Proc. 1st Int. Workshop on Multiple Classifier Systems, 2000, pp. 1–15.
12. Diettrich T.G. Machine learning for sequential data: a review. In Proc. Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition, 2002, pp. 15–30.
13. El-Manzalawy Y., Dobbs D., and Honavar V. On evaluating MHC-II binding peptide prediction methods, PLoS One, 3(9): e3268, 2008.
14. El-Manzalawy Y., Dobbs D., and Honavar V. Predicting linear B-cell epitopes using string kernels. J. Mole. Recogn., 21243–255, 2008.
15. Friedman N., Linial M., Nachman I., and Pe'er D. Using bayesian networks to analyze expression data. J. Comput. Biol., 7:601–620, 2000.
16. Galperin M.Y. The molecular biology database collection: 2008 update. Nucleic Acids Res., 36:D2–D4, 2008.
17. Guyon I. and Elisseeff A. An introduction to variable and feature selection. J. Mach. Learn. Res., 3:1157–1182, 2003.
18. Hecker L., Alcon T., Honavar V., and Greenlee H. Querying multiple large-scale gene expression datasets from the developing retina using a seed network to prioritize experimental targets. Bioinform. Biol. Insights, 2:91–102, 2008.
19. Jeong H., Tombor B., Albert R., Oltvai Z.N., and Barabasi A.-L. The large-scale organization of metabolic networks. Nature, 407:651–654, 1987.
20. Lahdesmaki H., Shmulevich I., and Yli-Harja O. On learning gene regulatory networks under the boolean network model. Mach. Learn., 52:147–167, 2007.
21. Terribilini M., Lee J.-H., Yan C., Jernigan R.L., Honavar V, and Dobbs D. Predicting RNA-binding sites from amino acid sequence. RNA J., 12:1450–1462, 2006.
22. Yan C., Terribilini M., Wu F., Jernigan R.L., Dobbs D., and Honavar V. Identifying amino acid residues involved in protein-DNA interactions from sequence. BMC Bioinform., 7:262, 2006.

Machine Learning in Systems Biology

▶ Machine Learning in Computational Biology

Machine-Readable Dictionary (MRD)

▶ Electronic Dictionary

Macro

▶ Snippet

Magnetic Disk

▶ Disk

Maid

▶ Massive Array of Idle Disks

Main Memory

Peter Boncz
CWI, Amsterdam, The Netherlands

Synonyms

Primary memory; Random access memory (RAM)

Definition

Primary storage, presently known as main memory, is the largest memory directly accessible to the CPU in the prevalent Von Neumann model and stores both data and instructions (program code). The CPU continuously reads instructions stored there and executes them. Also called Random Access Memory (RAM), to indicate that load/store instructions can access data at any location at the same cost, it is usually implemented using DRAM chips, which are connected to the CPU and other peripherals (disk drive, network) via a bus.

Key Points

The earliest computers used tubes, then transistors and since the 1970s in integrated circuits. RAM chips generally store a bit of data in either the state of a flip-flop, as in SRAM (static RAM), or as a charge in a capacitor (or transistor gate), as in DRAM (dynamic RAM). Some types have circuitry to detect and/or correct random faults called memory errors in the stored data, using parity bits or error correction codes (ECC). RAM of the read-only type, ROM, instead uses a metal mask to permanently enable/disable selected transistors, instead of storing a charge in them.

The main memory available to a program in most operating systems, while primarily relying on RAM, can be increased by disk memory. That is, the memory access instructions supported by a CPU work on so-called virtual memory, where an abstract virtual memory space is divided into pages. At any time, a page either resides in a swap-file on disk or in RAM, where it must be in order for the CPU to access it. When memory is accessed, the Memory Management Unit (MMU) of the CPU transparently translates the virtual address into its current physical address. If the memory page is not in RAM, it generates a page fault, to be a handled by the OS which then has to perform I/O to the swap file. If a high percentage of the memory access generates a page fault, this is called thrashing, and severely lowers performance.

Over the past decades, the density of RAM chips has increased, following a planned evolution of finer chip production process sizes, popularly known as "Moore's Law." This has led to an increase in RAM capacity as well as bandwidth. Access latency has also decreased, however, the physical distance on the motherboard between DRAM chips and CPU results in a minimum access latency of around 50ns (real RAM latencies are often higher). In current multi-GHz CPUs this means that a memory access instruction takes hundreds of cycles to execute. Typically, a high percentage of instructions in a program can be memory access instructions (up to 33%) and the RAM latency can seriously impact performance. This problem is known as the "memory wall."

To counter the performance problems of the memory wall, modern computer architecture now features a memory hierarchy that besides DRAM also includes SRAM cache memories, typically located on the CPU chip. Memory access instructions transfer memory in units of cache-lines, typically 64 bytes at a time (this cache line size is also related to the width of the memory bus). Memory access instruction first checks whether the accessed cache line is in the highest (fastest/smallest) L1 cache. This takes just a few CPU cycles. If a cache miss occurs, the memory access instruction checks the next cache level. Only if no cache contains the cache line, memory access is performed. Therefore, like virtual memory page thrashing, the CPU cache hit ratio achieved by a program now materially affects performance.

While in the past access to the DRAM chips over the bus was typically performed by a chipset, in between CPU and memory, some modern CPU architectures have moved the memory controller logic onto the CPU chip itself, which tends to reduce access latency. Also, to better serve the memory bandwidth requirement multi-CPU systems, modern architectures often have a dedicated memory bus between the CPU and DRAM. In a Symmetric Multi-Processing (SMP) this leads to a so-called Non-Uniform Memory Access architecture (NUMA), where access to the memory directly connected to a CPU is faster than access to the memory connected to another CPU.

While database systems traditionally focus on the disk access pattern (i.e., I/O), modern database systems, as well as main-memory database systems (that do not rely on I/O in the first place) now must carefully plan the in-memory data storage format used as well as the memory access patterns caused by query processing algorithms, in order to optimize the use of the CPU caches and avoid high cache miss ratios. The increased RAM sizes as well as the increased impact of I/O latency also leads to a trend to rely more on main memory as the preferred storage medium in database processing.

Cross-references

- ▶ Cache Memory
- ▶ CPU
- ▶ Disk

Main Memory DBMS

Peter Boncz
CWI, Amsterdam, The Netherlands

Synonyms

In-memory DBMS; MMDBMS

Definition

A main memory database system is a DBMS that primarily relies on main memory for computer data storage. In contrast, conventional database management systems typically employ hard disk based persistent storage.

Key Points

The main advantage of MMDBMS over normal DBMS technology is superior performance, as I/O cost is no more a performance cost factor. With I/O as main optimization focus eliminated, the architecture of main memory database systems typically aims at optimizing CPU cost and CPU cache usage, leading to different data layout strategies (avoiding complex tuple representations) as well as indexing structures (e.g., B-trees with lower-fan-outs with nodes of one or a few CPU cache lines).

While built on top of volatile storage, most MMDB products offer ACID properties, via the following mechanisms: (i) Transaction Logging, which records changes to the database in a journal file and facilitates automatic recovery of an in-memory database, (ii) Non-volatile RAM, usually in the form of static RAM backed up with battery power (battery RAM), or an electrically erasable programmable ROM (EEPROM). With this storage, the MMDB system can recover the data store from its last consistent state upon reboot, (iii) High availability implementations that rely on database replication, with automatic failover to an identical standby database in the event of primary database failure.

Main-memory database systems were originally popular in real-time systems (used in e.g., telecommunications) for their fast and more predictable performance, and this continues to be the case. However, with increasing RAM sizes allowing more problems to be addressed using a MMDBMS, this technology is proliferating into many other areas, such as on-line transaction systems, and recently in decision support. Main memory database systems are also deployed as drop-in systems that intercept read-only queries on cached data from an existing disk-based DBMS, thus reducing its workload and providing fast answers to a large percentage of the workload.

Examples of main-memory database systems are MonetDB, SolidDB, TimesTen and DataBlitz. MySQL offers a main-memory backend based on Heap tables. The MySQL Cluster product is a parallel main memory system that offers ACID properties through high availability.

Cross-references

- ▶ Disk
- ▶ Main Memory
- ▶ Processor Cache

Recommended Reading

1. Bohannon P., Lieuwen D.F., Rastogi R., Silberschatz A., Seshadri S., and Sudarshan S. The architecture of the dalí main-memory storage manager. Multimedia Tools Appl., 4(2):115–151, 1997.
2. Boncz P.A. and Kersten M.L. MIL primitives for querying a fragmented world. VLDB J., 8(2):101–119, 1999.
3. DeWitt D.J., Katz R.H., Olken F., Shapiro L.D., Stonebraker M., and Wood D.A. Implementation techniques for main memory

database systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 1–8.

4. Hvasshovd S-O., Torbjørnsen Ø., Bratsberg S.E., and Holager P. The ClustRa telecom database: high availability, high throughput, and real-time response. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 469–477.

Maintenance of Materialized Views with Outer-Joins

Per-Åke Larson
Microsoft Corporation, Redmond, WA, USA

Definition

An materialized outer-join view is a materialized view whose defining expression contains at least one outer join. View maintenance refers to the process of bringing the view up to date after one or more of the underlying base tables has been updated. View maintenance can always be done by recomputing the result, known as a full refresh, but this is usually prohibitively expensive. Incremental view maintenance, that is, only applying the minimal changes required to bring the view up to date, is normally more efficient.

Historical Background

Full outer join (called generalized join) was proposed by Lacroix and Pirotte in 1976 [4]. During the 1980's, there was considerable discussion in the research literature about the use and power of outer joins. Commercial systems began supporting outer joins in the late 1980's and at the time of writing (2007) all major commercial systems do. Optimization of outer-join queries was an active research area during the 1990's. Outer join was first included in the 1992 SQL standard. The first view matching algorithm for outer-join views was published by Larson and Zhou [5] in 2005.

In 1998 Griffin and Kumar [2] published the first paper covering incremental maintenance of materialized outer-join views. A paper 2006 by Gupta and Mumick [3] described a more efficient procedure but, unfortunately, it does not always produce the correct result. In 2007, Larson and Zhou [6] introduced an method for efficient incremental maintenance of outer-join views. At the time of writing, only Oracle allows (a limited form of) materialized outer-join views.

Foundations

Larson and Zhou [6] showed that incremental maintenance of an outer-join view can be divided into two steps: computing and applying a *primary delta* and a *secondary delta*. The first step is very similar to maintaining an inner-join view while the second step is a "clean-up" step.

This entry describes Larson's and Zhou's maintenance procedure for a view without aggregation when the update consists of insertions into one of its base tables. The reader is referred to the original paper [6] for a more complete description of how to handle deletions, views with aggregation, and how to exploit foreign-key constraints to simplify maintenance. Examples illustrating the procedure use a database consisting of the following three tables. Primary keys are underlined.

```
O(Okey, Odate, Ocustomer),
L(okey, pkey, Qty, Price),
P(Pkey, Pname).
```

The following materialized view consisting of two full outer joins will be used as a running example.

$$MV = \left(L \bowtie^{fo}_{p(l,p)} P\right) \bowtie^{fo}_{p(l,o)} O$$

where the join predicates are defined as $p(l,p) \equiv (l.pkey = p.pkey)$ and $p(l,o) \equiv (l.okey = o.okey)$.

Join-Disjunctive Normal Form

The view maintenance procedure builds on the join-disjunctive normal form for outer-join expressions introduced by Galindo-Legaria [6]. The normal form is described in this section by an example; more details can be found in [6,1].

Let T_1 and T_2 be tables with schemas S_1 and S_2, respectively. The *outer union*, denoted by $T_1 \uplus T_2$, first null-extends (pads with nulls) the tuples of each operand to schema $S_1 \cup S_2$ and then takes the union of the results (without duplicate elimination).

Let t_1 and t_2 be tuples with the same schema. Tuple t_1 is said to *subsume* tuple t_2 if t_1 agrees with t_2 on all columns where they both are non-null and t_1 contains fewer null values than t_2. The operator *removal of subsumed tuples* of T, denoted by $T\downarrow$, returns the tuples of T that are not subsumed by any other tuple in T.

The *minimum union* of tables T_1 and T_2 is defined as $T_1 \oplus T_2 = (T_1 \uplus T_1)\downarrow$. Minimum union is both commutative and associative.

Left outer join can be rewritten as $T_1 \bowtie_p^{lo} T_2 = T_1 \bowtie_p T_2 \oplus T_1$ and right outer join as $T_1 \bowtie_p^{ro} T_2 = T_1 \bowtie_p T_2 \oplus T_2$. Full outer join can be rewritten as $T_1 \bowtie_p^{fo} T_2 = T_1 \bowtie_p T_2 \oplus T_1 \oplus T_2$.

The example view was defined as

$$MV = \left(L \bowtie_{p(l,p)}^{fo} P\right) \bowtie_{p(l,p)}^{fo} O.$$

Conversion to normal form is done bottom up by applying the rewrite rules above. First rewrite the join between L and P in terms of inner joins and minimum union, which yields

$$MV = (\sigma_{p(l,p)}(L \times P) \oplus L \oplus P) \bowtie_{p(l,o)}^{fo} O.$$

Then apply the same rewrite to the second outer join, which produces

$$MV = ((\sigma_{p(l,p)}(L \times P) \oplus L \oplus P) \bowtie_{(l,o)} O) \\ \oplus (\sigma_{p(l,p)}(L \times P) \oplus L \oplus P) \oplus O.$$

Inner join distributes over minimum union in the same way as over regular union. Applying this transformation to the join with O produces

$$MV = \sigma_{p(l,p) \wedge p(l,o)}(O \times L \times P) \\ \oplus \sigma_{p(l,o)}(O \times L) \oplus \sigma_{p(l,o)}(O \times P) \\ \oplus \sigma_{p(l,p)}(L \times P) \oplus L \oplus P \oplus O.$$

The view expression is now in join-disjunctive form but it can be further simplified. The term $\sigma_{p(l,o)}(O \times P)$ can be eliminated because the join predicate will never be satisfied.

$$MV = \sigma_{P(l,p) \wedge P(l,o)}(O \times L \times P) \oplus \sigma_{p(l,o)}(O \times L) \\ \oplus \sigma_{p(l,p)}(L \times P) \oplus L \oplus P \oplus O.$$

The normal form shows what form of tuples are found in MV. For example, it "contains" all tuples in the join of O and L. Most such tuples are represented implicitly by being included in a wider tuple composed of tuples from O, L and P; only the non-subsumed tuples are stored explicitly in the view.

As illustrated by this example, an outer-join expression E over a set of tables $\mathcal{U}$ can be converted to a normal form consisting of the minimum union of terms composed from selections and inner joins (but no outer joins). More formally, the join-disjunctive normal form of E equals

$$E = E_1 \oplus E_2 \oplus \cdots \oplus E_n$$

where each term E_i is of the form $E_i = \sigma_{pi}(T_{i1} \times T_{i2} \times \ldots \times T_{im})$. $\mathcal{T}_i = \{T_{i1}, T_{i2} \ldots T_{im}\}$ is a (unique) subset of the tables in $\mathcal{U}$. Predicate p_i is the conjunction of a subset of the selection and join predicates found in the original form of the query.

The Subsumption Graph

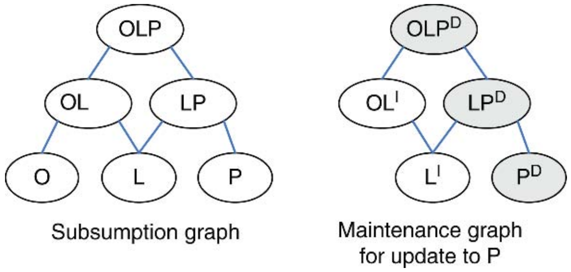
Every term in the normal form of the view has a unique set of source tables drawn from $\mathcal{U}$ and is null-extended on all other tables in the view. The set of source tables of term E_i is denoted by $\mathcal{T}_i$ and the set of tables on which it is is null-extended by $\mathcal{S}_i, \mathcal{S}_i = \mathcal{U} - \mathcal{T}i$.

A tuple produced by a term with source tables $\mathcal{T}_i$ can only be subsumed by tuples produced by terms whose source set is a superset of $\mathcal{T}_i$. The subsumption relationships among terms can be modeled by a DAG called the subsumption graph.

The *subsumption graph* of E contains a node n_i for each term E_i in the normal form and the node is labeled with the source table set $\mathcal{T}_i$ of E_i. There is an edge from a node n_i to a node n_j, if $\mathcal{T}_i$ is a minimal superset of $\mathcal{T}_j$. $\mathcal{T}_i$ is a minimal superset of $\mathcal{T}_j$ if there does not exist a node n_k in the graph such that $\mathcal{T}_j \subset \mathcal{T}_k \subset \mathcal{T}_i$.

The subsumption graph for view MV is shown to the left in Fig. 1. The importance of the subsumption graph lies in the following observation: when checking whether a tuple of a term is subsumed, it is sufficient to check against tuples in the term's immediate parent terms. For example, to determine whether a P tuple p_1 is subsumed, all that is needed is to check whether it joins with an L tuple. If it does, the resulting tuple, which subsumes p_1, is included in the LP term.

The result of an outer-join expression is represented in a minimal form. Only the non-subsumed



Maintenance of Materialized Views with Outer-Joins. Figure 1. Subsumption graph and maintenance graph for view *MV*.

tuples produced by a term E_i in the normal form are explicitly represented. A subsumed tuple is represented implicitly by being included in a subsuming tuple. The *net contribution* of a term, denoted by D_i, is the set of non-subsumed tuples of term E_i in the normal form of expression E. Then E can then be written in the form

$$E = D_1 \uplus D_2 \uplus \cdots \uplus D_n.$$

Consider a view V and suppose one of the its base tables T is modified. This may affect the net contribution of a term D_i in one of three ways:

1. Directly, which occurs if T is among the tables in $\mathcal{T}_i$;
2. Indirectly, which occurs if T is not among the tables in $\mathcal{T}_i$ but it is among the source tables of at least one of its parent nodes;
3. No effect, otherwise.

Based on this classification of how terms are affected, a *view maintenance graph* is created as follows.

1. Eliminate from the subsumption graph all nodes that are unaffected by the update of T.
2. Mark the remaining nodes by D or I depending on whether the node is affected directly or indirectly.

The maintenance graph for view when updating P is shown to the right in Fig. 1. The maintenance graph is used primarily to identify which terms of a view are indirectly affected and thus may require maintenance.

Maintenance Procedure

Suppose table T has been updated. If so, any view V that references T needs to be maintained. The first step is to compute the view's maintenance graph and classify the terms as directly affected, indirectly affected, and unaffected. Without loss of generality, assume that the view has n terms, of which terms $1, 2,\ldots,k$ are directly affected, terms $k + 1$, $k + 2,\ldots,k+ m$ are indirectly affected, and terms $k + m + 1$, $k + m + 2,\ldots,n$ are not affected. The view expression can then be rewritten in the form

$$V = V^D \uplus V^I \uplus V^U \text{where}$$
$$V^D = \uplus_{i=1}^{k} D_i, \quad V^I = \uplus_{i=k+1}^{k+m} D_i,$$
$$V^U = \uplus_{i=k+m+1}^{n} D_i.$$

From this form of the expression, one can see that to update the view, two delta expressions must be evaluated and applied to the view

$$\Delta V^D = \uplus_{i=1}^{k} \Delta D_{i,} \, \Delta V^I = \uplus_{i=k+1}^{k+m} \Delta D_i.$$

ΔV^D is called the *primary delta* and ΔV^I the *secondary delta*. In summary, maintenance of a view V after insertions into one of its underlying base tables can be performed in two steps.

1. Compute the primary delta ΔV^D and insert the resulting tuples into the view.
2. If there are indirectly affected terms, compute the secondary delta ΔV^I and delete the resulting tuples from the view.

Computing the Primary Delta

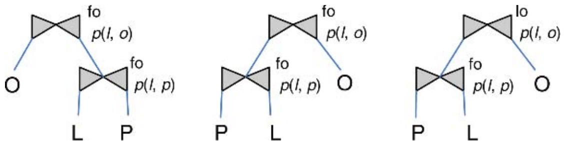
An expression that computes the primary delta, ΔV^D, can be constructed by the following simple algorithm.

1. Traverse the operator tree for V along the path from T to the root. On any join operator encountered, apply commutativity rules to ensure that the input referencing T is on the left.
2. Traverse the path from T to the root of V. Convert any full outer join operator encountered to a left outer join and any right outer join operator to an inner join.
3. Substitute T by ΔT.

Step 1 is a normal rewrite of the view expression and does not change the result. Step 2 modifies the expression so that it computes only V^D. After Step 2, the operators on the path from T to the root consists only of selects, inner joins, and left outer joins and the delta expression is always the left input.

Figure 2 illustrates the transformation process for the example view MV when table P is updated. The resulting expression for computing the primary delta is

$$\Delta MV^D = \left(\Delta P \bowtie^{lo}_{p(l,p)} L\right) \bowtie^{lo}_{p(l,o)} O$$



Maintenance of Materialized Views with Outer-Joins. Figure 2. Constructing primary-delta expression for insertions into table P.

Computing the Secondary Delta

The secondary delta can be computed efficiently from the primary delta and either the view or base tables. Only the case when using the view is described here. Recall that the base tables have already been updated and the primary delta has been applied to the view.

The primary delta ΔV^D contains the union of the deltas for all directly affected terms. However, deltas for individual terms are needed to compute the secondary delta. Each term is defined over a unique set of tables and null extended on all others so tuples from a particular term are easily identified and can be extracted from ΔV^D by simple selection predicates.

Let $null(T)$ denote a predicate that evaluates to true if a tuple is null-extended on table T. $null(T)$ can be implemented in SQL as "$T.c$ is null" where c is any column of T that does not contain nulls, for example, a column of a key. When applying $null$ and $\neg null$ to a set of tables $T = \{T_1, T_2 ..., T_n\}$, the shorthand notations $n(\mathcal{T}) = \bigwedge_{T_i \in \mathcal{T}} null(T_i)$ and $nn(\mathcal{T}) = \bigwedge_{T_i \in \mathcal{T}} \neg null(T_i)$ are used.

For the example view, MV, the primary delta contains deltas of three directly affected terms, see Fig. 1. Non-subsumed tuples from, for example, the LP-term are uniquely identified by the fact that they are composed of a real tuple from L and from P but are null extended on O. Hence, $\mathit{\Delta} D_{LP}$ can be extracted from ΔV^D as follows:

$$\Delta D_{LP} = \pi_{(LP).*} \sigma_{nn(LP) \wedge n(o)} \Delta MV^D$$

where $nn(LP) = \neg null(L) \wedge \neg null(P)$ and $n(O) = null(O)$.

ΔD_{LP} contains only the delta of the net contribution of the term. ΔE_{LP} contains the complete delta of the term, including both subsumed and non-subsumed tuples. Tuples in ΔE_{LP} are composed of real tuples from L, and from P, and may or may not be null extended on O. Hence, ΔE_{LP} can be extracted from ΔV^D as follows:

$$\Delta E_{LP} = \delta \pi_{(LPo).*} \sigma_{nn}(LP) \Delta MV^D.$$

The duplicate elimination (δ) is necessary because an LP tuple may have joined with multiple O tuples.

Continuing with the running example, the secondary delta consists of ΔD_{OL} and ΔD_L. ΔD_{OL} is null extended on P and the OLP-term is its only parent so it can be computed as:

$$\Delta D_{OL} = \sigma_{nn(OL) \wedge n(P)} (MV + \Delta MV^D) \bowtie^{ls}_{eq(OL)} \sigma_{nn(OLP)} \Delta MV^D.$$

This expression makes sense intuitively. The first part selects from the view all orphaned (non-subsumed) tuples of term E_{OL} contained in the view after the primary delta has been applied. The second part extracts from the primary delta all tuples added to the parent term E_{OLP}. The join is a left semijoin and outputs every tuple from the left operand that joins with one or more tuples in the right operand. The complete expression thus amounts to finding all currently orphaned tuples of the term and retaining those that cease to be orphans because of the insert. Those tuples should be deleted from the view.

D_L is null extended on O, and P and has one directly affected parent, the LP-term. ΔD_L can be computed as:

$$\Delta D_L = \sigma_{nn(L) \wedge n(OP)} (MV + \Delta MV^D) \bowtie^{ls}_{eq(L)} \sigma_{nn}(LP) \Delta MV^D.$$

Summary

In summary, after insertion into table P of a set of tuples ΔP, the example view MV can be brought up to date as follows. First compute the primary delta

$$\Delta MV^D = (\Delta P \bowtie^{lo}_{p(l,p)} L) \bowtie^{lo}_{p(l,o)} O$$

and insert the resulting tuples into the view, resulting in $MV + \Delta MV^D$. Then compute the secondary delta

$$\begin{aligned} \Delta MV^I &= \Delta D_{OL} \uplus \Delta D_L \\ &= \sigma_{nn(OL) \wedge n(P)} (MV + \Delta MV^D) \\ &\quad \bowtie^{ls}_{eq(OL)\, \sigma_{nn}(OLP)} \Delta MV^D \uplus \\ &\quad \sigma_{nn(L) \wedge n(OP)} (MV + \Delta MV^D) \\ &\quad \bowtie^{ls}_{eq(L)\, \sigma_{nn}(LP)} \Delta MV^D \end{aligned}$$

and delete the resulting tuples from the view.

Key Applications

Queries containing outer joins are often used in analysis queries over large tables in data warehouses. Materialized outer-join views, especially when aggregated, can be very beneficial in such scenarios.

Cross-references

- ▶ Materialized Views
- ▶ Maintenance of Materialized Views with Outer-Joins
- ▶ Views

Recommended Reading

1. Galindo-Legaria C. Outerjoins as disjunctions. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 348–358.
2. Griffin T. and Kumar B. Algebraic change propagation for semijoin and outerjoin queries. ACM SIGMOD Rec., 27(3):22–27, 1998.
3. Gupta A. and Mumick I.S. Incremental maintenance of aggregate and outerjoin expressions. Inf. Syst., 31(6):435–464, 2006.
4. Lacroix M. and Pirotte A. Generalized joins. ACM SIGMOD Rec., 8(3):14–15, 1976.
5. Larson P. and Zhou J. View matching for outer-join views. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 445–456.
6. Larson P. and Zhou J. Efficient maintenance of materialized outer-join views. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 56–65.

Maintenance of Recursive Views

Suzanne W. Dietrich
Arizona State University, Phoenix, AZ, USA

Synonyms

Incremental maintenance of recursive views; Recursive view maintenance

Definition

A view is a derived or virtual table that is typically defined by a query, providing an abstraction or an alternate perspective of the data that allows for more intuitive query specifications using these views. Each reference to the view name results in the retrieval of the view definition and the recomputation of the view to answer the query in which the view was referenced. When views are materialized, the tuples of the computed view are stored in the database with appropriate index structures so that subsequent access to the view can efficiently retrieve tuples to avoid the cost of recomputing the entire view on subsequent references to the view. However, the materialized view must be updated if any relation that it depends on has changed. Rather than recomputing the entire view on a change, an incremental view maintenance algorithm uses the change to incrementally compute updates to the materialized view in response to that change. A recursive view is a virtual table definition that depends on itself. A canonical example of a recursive view is the transitive closure of a relationship stored in the database that can be modeled as directed edges in a graph. The transitive closure essentially determines the reachability relationship between the nodes in the graph. Typical examples of transitive closure include common hierarchies such as employee-supervisor, bill-of-materials (parts-subparts), ancestor, and course prerequisites. The incremental view maintenance algorithms for the maintenance of recursive views have additional challenges posed by the recursive nature of the view definition.

Historical Background

A view definition relates a view name to a query defined in the query language of the database. Initially, incremental view maintenance algorithms were explored in the context of non-recursive view definitions involving select, project, and join query expressions, known as SPJ expressions in the literature. The power of recursive views was first introduced in the *Datalog* query language, which is a declarative logic programming language established as the database query language for deductive databases in the 1980s. Deductive databases assume the theoretical foundations of relational data but use Datalog as the query language. Since its relational foundations assume first normal form, Datalog looks like a subset of the Prolog programming language without function symbols. However, Datalog does not assume Prolog's top-down left-to-right programming language evaluation strategy. The evaluation of Datalog needed to be founded on the fundamentals of database query optimization. In a database system, a user need only specify a correct declarative query, and it is the responsibility of the database system to efficiently execute that specification. The evaluation of Datalog was further complicated by the fact that Datalog allows for relational views that include union and recursion in the presence of negation. Therefore, the view definitions in Datalog were more expressive than the traditional select-project-join views available in relational databases at that time. Therefore, the incremental view maintenance algorithms for recursive views in the early 1990s are typically formulated in the context of the evaluation of Datalog. The power to define a recursive union in SQL was added in the SQL:1999 standard.

Historically, it is important to note that the incremental maintenance of recursive views is related to the areas of integrity constraint checking and condition monitoring in active databases. These three areas were being explored in the research literature at around the same time. In integrity constraint checking, the database is assumed to be in a consistent state and when a change occurs in the database, it needs to incrementally determine whether the database is still in a consistent state. In active databases, the database is responsible for actively checking whether a condition that it is responsible for monitoring is now satisfied by incrementally evaluating condition specifications affected by changes to the database. Although closely related, there are differences in the underlying assumptions for these problems.

Foundations

Recursive View Definition

A canonical example of a recursive view definition is the reachability of nodes in a directed graph. In Datalog, the reach view consists of two rules. The first non-recursive rule serves as the base or seed case, and indicates that if the stored or base table edge defines a directed edge from the source node to the destination node, then the destination can be reached from the source. The second rule is recursive. If the source node can reach some intermediate node and there is an edge from that intermediate node to a destination node, then the source can reach the destination.

```
reach(Source, Destination):-
   edge(Source, Destination).
reach(Source, Destination):-
   reach(Source, Intermediate),
   edge(Intermediate, Destination).
```

Intuitively, one can think of the recursive rule as an unfolding of the joins required to compute the reachability of paths of length two, then paths of length three, and so on until the data of the underlying graph is exhausted.

In SQL, this recursive view is defined with the following recursive query expression:

```
with recursive reach(source, destina-
tion) as
(select E.source, E.destination
from edge E)
union
(select S.source, D.destination
from reach S, edge D
where S.destination = D.source)
```

SQL limits recursive queries to linear recursions, which means that there is at most one direct invocation of a recursive item. The specification of the reach view above is an example of a linear recursion. There is another linear recursive specification of reach where the direct recursive call appears on the right side of the join versus the left side of the join:

```
reach(Source, Destination):-
   edge(Source, Intermediate),
   reach(Intermediate, Destination).
```

However, there is a logically equivalent specification of reach that is non-linear:

```
reach(Source, Destination):-
   reach(Source, Intermediate),
   reach(Intermediate, Destination).
```

The goal of Datalog evaluation is to allow the user to specify the recursive view declaratively in a logically correct way, and it is the system's responsibility to optimize the evaluation of the query.

SQL also restricts recursions to those defined in deductive databases as stratified *Datalog with negation*. Without negation, a recursive Datalog program has a unique solution that corresponds to the theoretical fixpoint semantics or meaning of the logical specification. In the computation of the reach view, each unfolding of the recursion joins the current instance of the recursive view with the edge relation until no new tuples can be added. The view instance has reached a fixed point and will not change. When negation is introduced, the interaction of recursion and negation must be considered. The concept of stratified negation means that there can be no negation through a recursive computation, i.e., a view cannot be defined in terms of its own negation. Recursive views can contain negation but the negation must be in the context of relations that are either stored or completely computed before the application of the negation. This imposed level of evaluation with respect to negation and recursion are called strata. For stratified Datalog with negation, there also exists a theoretical fixpoint that represents the intuitive meaning of the program.

Consider an example of a view defining a peer as two employees that are not related in the employee-supervisor hierarchy:

```
peer(A, B):- employee(A,...), employee(B,...),
   not (supervisor(A,B)), not(supervisor(B,A)).
supervisor(Emp, Sup):-
   immediateSupervisor(Emp,Sup).
```

```
supervisor(Emp, Sup):-
   supervisor(Emp, S),
   immediateSupervisor(S, Sup).
```

Since peer depends on having supervisor materialized for the negation, peer is in a higher stratum than supervisor. Therefore, the strata provide the levels in which the database system needs to compute views to answer a query.

Evaluation of Recursive Queries

Initial research in the area emphasized the efficient and complete evaluation of recursive queries. The intuitive evaluation of the recursive view that unions the join of the current view instance with the base data at each unfolding is known as a naïve bottom-up algorithm. In a bottom-up approach to evaluating a rule, the known collection of facts is used to satisfy the subgoals on the right-hand side of the rule, generating new facts for the relation on the left-hand side of the rule. To improve the efficiency of the naïve algorithm, a semi-naïve approach can be taken that only uses the new tuples for the recursive view from the last join to use in the join at the next iteration. A disadvantage of this bottom-up approach for evaluating a query is that the entire view is computed even when a query may be asking for a small subset of the data. This eager approach is not an issue in the context of materializing an entire view.

Another recursive query evaluation approach considered a top-down strategy as in Prolog's evaluation. In a top-down approach to evaluation, the evaluation starts with the query and works toward the collection of facts in the database. In the context of the reach recursive view, the reach query is unified with the left-hand side of the non-recursive rule and rewritten as a query involving edge. The edge facts are then matched to provide answers. The second recursive rule is then used to rewrite the reach query with the query consisting of the goals on the right-hand side of the rule. This evaluation process continues, satisfying the goals with facts or rewriting the goals using the rules. The unification of a goal with the left-hand side of a rule naturally filters the evaluation by binding variables in the rule to constants that appear in the query. However, the evaluation of a left-recursive query using Prolog's evaluation strategy enters an infinite loop on cyclic data by attempting to prove the same query over and over again. A logic programmer would not write a logic program that enters an infinite loop, but the deductive database community was interested in the evaluation of truly declarative query specifications.

The resulting evaluation approaches combine the best of top-down filtering with bottom-up materialization. The magic sets technique added top-down filtering by cleverly rewriting the original rules so that a bottom-up evaluation would take advantage of constants appearing in the query [1]. Memoing was added to a top-down evaluation strategy to achieve the duplicate elimination feature that is inherent in a bottom-up evaluation of sets of tuples [3]. This duplicate elimination feature avoids the infinite loops on cyclic data. Top-down memoing is complete for subsets of Datalog on certain types of queries [4]. For stratified Datalog with negation, top-down memoing still requires iteration to guarantee complete evaluation. Further research explored additional optimizations as well as implementations of deductive database systems [12] and led to research in active databases and materialized view maintenance.

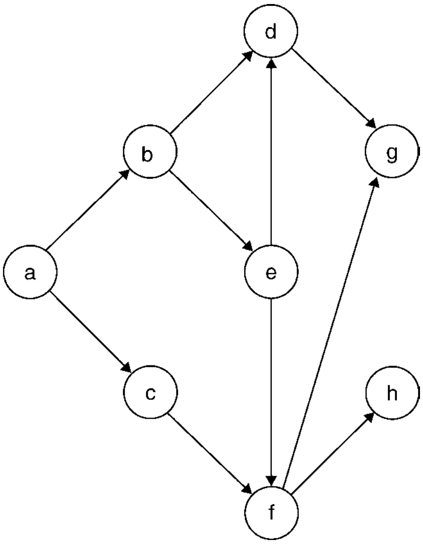
Incremental Evaluation of Recursive Views

A view maintenance algorithm uses the change to incrementally determine updates to the view. Consider a change in the underlying graph for the transitive closure example. If a new edge is inserted, this edge may result in a change to the materialized reach view by adding a connection between two nodes that did not exist before. However, another possibility is that the new edge added another path between two nodes that were already in the materialized view. A similar situation applies on the removal of an edge. The deletion could result in a change in the reachability between nodes or it could result in the removal of a path but the nodes are still connected via another route. In addition, in the general case, a view may depend on many relations including other (recursive) views in the presence of negation. Therefore, the approaches for the incremental maintenance of recursive views typically involve a propagation or derivation phase that determines an approximation or overestimate of the changes, and a filtering or rederivation phase that checks whether the potential change represents a change to the view. There are differences in the underlying details of how these phases are performed.

The two incremental view maintenance algorithms that will be presented by example are the DRed algorithm [7] and the PF Algorithm [8]. Both the DRed and PF algorithms handle recursive stratified Datalog programs with negation. There are other algorithms developed for special cases of Datalog programs and queries, such as the counting technique for nonrecursive programs, but this exposition will explore these more general approaches for incremental view maintenance. Historically, the PF algorithm was developed in the context of top-down memoing whereas DRed assumes a bottom-up semi-naïve evaluation. To assist with the comparison of the approaches, the notation introduced for the DRed algorithm [7] will be used to present both algorithms in the context of the transitive closure motivational example.

Figure 1 provides a graphical representation of an edge relation. Assume that the view for reach is materialized, and the edge (e,f) is deleted from the graph. The potential deletions or overestimates for reach, denoted by δ^-(reach), are computed by creating Δ^- rules for each rule computing reach. Each reach rule has k Δ^- rules where k corresponds to the number of subgoals in the body of the rule. The ith Δ^- rule uses the current estimate of deleted tuples (δ^-) for the ith subgoal. For the nonrecursive rule, there is only one subgoal. Therefore, there is only one Δ^- rule indicating that potential edge deletions generate potential deletions to the reach view.

Δ^-(r1): δ^-(reach(S, D)):- δ^-(edge(S, D)).



Maintenance of Recursive Views. Figure 1. Sample Graph.

Since the recursive rule has two subgoals, there are two Δ^- rules:

Δ^-(r21): δ^-(reach(S, D)):- δ^-(reach(S, I)), edge(I, D).
Δ^-(r22): δ^-(reach(S, D)):- reach(S, I), δ^-(edge(I, D)).

Potential deletions to the reach view as well as the edge relation can generate potential deletions to the view.

These potential deletions need to be filtered by determining whether there exist alternative derivations or paths between the nodes computed in the potential deletion. There is a Δ^r rule defined for each reach rule that determines the rederivation of the potential deletions, which is denoted by δ^+ (reach):

Δ^r(r1): δ^+ (reach(S, D)):- δ^-(reach(S, D)), edge^v(S, D).
Δ^r(r2): δ^+ (reach(S, D)):- δ^-(reach(S, D)), reach^v(S, I), edge^v(I, D).

The superscript v on the subgoals in the rule indicates the use of the current instance of the relation corresponding to the subgoal. If the potential deletion is still reachable in the new database instance, then there exists another route between the source and destination, and it should not be removed from the materialized view. The actual removals to reach, indicated by Δ^-(reach), is the set of potential deletions minus the set of alternative derivations:

Δ^-(reach) = δ^-(reach) – δ^+ (reach)

Table 1 illustrates the evaluation of the DRed algorithm for incrementally maintaining the reach view on the deletion of edge(e,f) from Fig. 1. The DRed algorithm uses a bottom-up evaluation of the given rules, starting with the deletion δ^-(edge(e, f)). In the first step, the Δ^- rules compute the overestimate of the deletions to reach. The result of the Δ^- rules are shown in the right column, which indicates the potential deletions to reach as δ^-(reach). The second step uses the Δ^r rules to filter the potential deletions. The right column illustrates the source destination pairs that are still reachable after the deletion of edge(e,f) as δ^+ (reach). The tuples that must be removed from the materialized view are indicated by Δ^-(reach): {(e,f) (e,h) (b,f) (b,h)}.

The PF (Propagate Filter) algorithm on the same example is shown in Table 2. PF starts by propagating the edge deletion using the nonrecursive rule, which generates a potential deletion of reach(e,f). This approximation is immediately filtered to determine whether there exists another path between e and f.

Maintenance of Recursive Views. Table 1. DRed algorithm on deletion of edge (e,f) on materialized reach view

DRed algorithm		
Step 1	*Compute overestimate of potential deletions*	δ^{-}(reach)
	Δ^{-}(r1): δ^{-}(reach(S, D)):- δ^{-}(edge(S, D)).	(e,f)
	Δ^{-}(r21): δ^{-}(reach(S, D)):- δ^{-}(reach(S, I)), edge(I, D).	(e,g) (e,h)
	Δ^{-}(r22): δ^{-}(reach(S, D)):- reach(S, I), δ^{-}(edge(I, D)).	(a,f) (b,f)
	Repeat until no change: No new tuples for Δ^{-}(r1) and Δ^{-}(r22)	
	Δ^{-}(r21): δ^{-}(reach(S, D)):- δ^{-}(reach(S, I)), edge(I, D).	(a,g) (a,h) (b,g) (b,h)
	Last iteration does not generate any new tuples	
Step 2	*Find alternative derivations to remove potential deletions*	δ^{+} (reach)
	Δ^{r}(r1): δ^{+} (reach(S, D)):- δ^{-}(reach(S, D)), edge(S, D).	
	Δ^{r}(r2): δ^{+} (reach(S, D)):- δ^{-}(reach(S, D)), reach^{v}(S, I), edge^{v}(I, D).	(e,g) (a,f) (a,g) (a,h) (b,g)
Step 3	*Compute actual changes to reach*	Δ^{-}(reach)
	Δ^{-}(reach) = δ^{-}(reach) – δ^{+} (reach)	(e, f) (e,h) (b,f) (b,h)

Maintenance of Recursive Views. Table 2. PF algorithm on deletion of edge (e,f) on materialized reach view

PF algorithm				
Propagate			**Filter**	
	Rule	**δ^{-}(reach)**	**δ^{+} (reach)**	**Δ^{-}(reach)**
δ^{-}(edge):{(e, f)}	Δ^{-}(r1)	(e,f)	{}	(e,f)
Δ^{-}(reach): {(e,f)}	Δ^{-}(r21)	(e,g) (e,h)	(e,g)	(e,h)
Δ^{-}(reach): {(e,h)}	Δ^{-}(r21)	{}		{}
δ^{-}(edge): {(e, f)}	Δ^{-}(r22)	(a,f) (b,f)	(a,f)	(b,f)
Δ^{-}(reach): {(b,f)}	Δ^{-}(r21)	(b,g) (b,h)	(b,g)	(b,h)
Δ^{-}(reach): {(b,h)}	Δ^{-}(r21)	{}		{}

Since there is no alternate route, the tuple (e,f) is identified as an actual change, and is then propagated. The propagation of Δ^{-}(reach): {(e,f)} identifies (e,g) and (e,h) as potential deletions. However, the filtering phase identifies that there is still a path from e to g, so (e, h) is identified as a removal to reach. The propagation of (e,h) does not identify any potential deletions. The propagation of the initial edge deletion δ^{-}(edge): {(e, f)} must be propagated through the recursive rule for reach using Δ^{-}(r22). The potential deletions are immediately filtered, and only actual changes are propagated. The PF algorithm also identifies the tuples {(e,f) (e,h) (b,f) (b,h)} to be removed from the materialized view.

As shown in the above deletion example, the DRed and PF algorithms both compute overestimates or approximations of tuples to be deleted from the recursive materialized view. The PF algorithm eagerly filters the potential deletions before propagating them. The DRed algorithm propagates the potential deletions within a stratum but filters the overestimates before propagating them to the next stratum. There are scenarios in which the DRed algorithm outperforms the PF algorithm and others in which the PF algorithm outperforms the DRed algorithm.

For the case of insertions, the PF algorithm operates in a manner similar to deletions, by approximating the tuples to be added and filtering the potential additions by determining whether the tuple was provable in the old database state. However, the DRed algorithm uses the bottom-up semi-naïve algorithm for Datalog evaluation to provide an inherent mechanism for determining insertions to the materialized view. In semi-naïve evaluation, the original rules are executed once to provide the seed or base answers. Then incremental versions of the rules are executed until a fixpoint is reached. The incremental rules are formed by creating k rules associated with a rule where k corresponds to the number of subgoals in the right-hand side of the rule. The i^{th} incremental rule uses only the new tuples from the last iteration for the i^{th} subgoal. However, when the i^{th} subgoal is a stored relation, then the corresponding incremental rules are removed since they will not contribute to the incremental evaluation. For the motivational example, the incremental rule for reach is

Δreach(S,I), edge(I, D)

where Δreach represents the new reach tuples computed on the previous iteration. Since a set of tuples is being computed, duplicate proofs are automatically filtered and are not considered new tuples. This is the inherent memoing in bottom-up evaluation that handles cycles in the underlying data.

Key Applications

Query Optimization; Condition Monitoring; Integrity Constraint Checking; Data Warehousing; Data Mining; Network Management; Mobile Systems.

Cross-references

▶ Datalog

▶ Incremental Maintenance of Views with Aggregates

▶ View Maintenance

▶ Maintenance of Materialized Views with Outer-Joins

Recommended Reading

1. Bancilhon F., Maier D., Sagiv Y., and Ullman J. Magic sets and other strange ways to implement logic programs. In Proc. 5th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1986, pp. 1–15.
2. Ceri S. and Widom J. Deriving production rules for incremental view maintenance. In Proc. 17th Int. Conf. on Very Large Data Bases, 1991, pp. 577–589.
3. Dietrich S.W. Extension tables: memo relations in logic programming. In 14th Int. Colloquium on Automata, Languages, and Programming, 1987, pp. 264–272.
4. Dietrich S.W. and Fan C. On the completeness of naive memoing in prolog. New Generation Comput., 15:141–162, 1997.
5. Dong G. and Su J. Incremental maintenance of recursive views using relational calculus/SQL. ACM SIGMOD Rec., 29 (1):44–51, 2000.
6. Gupta A. and Mumick I.S. (eds.). Materialized Views: Techniques, Implementations, and Applications, The MIT Press, Cambridge, MA, 1999.
7. Gupta A., Mumick I.S., and Subrahmanian V.S. Maintaining views incrementally. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 157–166.
8. Harrison J.V. and Dietrich S.W. Maintenance of materialized views in a deductive database: an update propagation approach. In Proc. Workshop on Deductive Databases, 1992, pp. 56–65.
9. Küchenhoff V. On the efficient computation of the difference between consecutive database states. In Proc. 2nd Int. Conf. on Deductive and Object-Oriented Databases, 1991, pp. 478–502.
10. Martinenghi D. and Christiansen H. Efficient integrity constraint checking for databases with recursive views. In Proc. 9th East European Conf. Advances in Databases and Information Systems, 2005, pp. 109–124.
11. Ramakrishnan R. (ed.). Applications of Logic Databases. Kluwer, Norwell, MA, 1995.
12. Ramakrishnan R. and Ullman D. A survey of deductive database systems. J. Logic Programming, 23(2):125–149, 1995.
13. Ullman J. Principles of Database and Knowledge Base Systems, Computer Science Press, Rockville, MD, 1989.
14. Urpí T. and Olivé A. A method for change computation in deductive databases. In Proc. 18th Int. Conf. on Very Large Data Bases, 1992, pp. 225–237.

Managing Compressed Structured Text

Gonzalo Navarro
University of Chile, Santiago, Chile

Synonyms

Searching compressed XML; Compressing XML

Definition

Compressing semi-structured text is the problem of creating a reduced-space representation from which the original data can be re-created exactly. Compared to plain text compression, the goal is to take advantage of the structural properties of the data. A more ambitious goal is being able to manipulate this text in compressed form, without decompressing it. This entry focuses on compressing, navigating, and searching semi-structured text, as those are the areas where more advances have been made.

Historical Background

Modeling data using semi-structured text has been a topic of interest at least since the 1980's, with a significant burst of activity in the 1990's [2]. Since then, the widespread adoption of XML (appearing in 1998, see the current version at http://www.w3.org/TR/xml) as the standard to represent semi-structured data has unified the efforts of the community around this particular format. Very early, however, the same features that made XML particularly appealing for both human and machine processing were pointed out as significant sources of redundancy and wasting of storage space and bandwidth. This was especially relevant for wireless transmission and triggered the proposal of the WAP Binary XML Content Format as early as 1999 (see http://www.w3.org/TR/wbxml), where simple techniques to compress XML prior to its transmission were devised.

In parallel, there has been a growing interest in not only compressing the data for storage or transmission, but in manipulating it in compressed form. The reason is the long-standing tradeoff between faster/smaller/more expensive and slower/larger/cheaper memories. A more compact data representation has the potential of fitting in a faster memory, where manipulating it can be orders of magnitudes faster, even if it requires more operations, than a naive representation fitting only in a slower memory.

Foundations

For concreteness, this entry will focus on the de-facto standard XML, where the structure is a tree or a forest marked with beginning and ending tags in the text. In fact this encompasses many other semi-structured text proposals, hence most of the material of the entry applies to semi-structured text in general, with minimal changes. In XML, the tags can have attributes and associated values, and there might be available a grammar giving the permissible context-free syntax of the semi-structured document.

Compression of Semi-Structured Text

An obvious approach to compressing semi-structured text is to regard it as plain text and use any of the well-known text compression methods [5]. Yet, considering the structure might yield improved compression performance compared to ignoring it. Many compressors have been proposed trying to exploit structure in different ways. Rather than describing them individually, the main principles behind them will be presented.

1. The data is a mix of structure and content. The structure can be regarded as a labeled tree, where the labels are the tag names, and the content as free text, which can appear between every consecutive pair of tree nodes, and within tree leaves. Attribute information can be handled as text as well, or as special data attached to tree nodes.
2. The structure and the content can be compressed separately, which has proved to give good results. Later, encoded tags and contents can be stored in the file in their original order, so that the document can be handled as a plain uncompressed document. Alternatively, structure and content can be stored separately with some pointer information to reconstruct the tree, in which case the structure pointers may help to point out relevant content to scan in the querying process.
3. The text content can be compressed using any text compression method. Semi-static compressors permit accessing the content at random without decompressing all from the beginning, whereas adaptive compressors tend to achieve better compression ratios. Splitting the text into blocks that are compressed adaptively permits trading random access time for compression ratio.
4. The structure can be compressed in several ways, which can range from a simple scheme of assigning numbers to the different tag names, to sophisticated grammar-based compression methods. The latter may take advantage of the explicit grammar when it is available.
5. Structure can be used, in addition, to boost compression. If the text contents are grouped according to the structural path towards the root, and each group is compressed separately, compression ratios improve noticeably. This can be as simple as grouping texts that are under the same tag (that is, considering only the deepest tree node containing the text) or as sophisticated as considering the full path towards the root.

A sample of different open-source systems that compress XML based on diverse combinations of these principles is

XMill [13],
Millau [10],
XMLPPM [6],
XGrind [15],
XCQ [12],
XPress [14], and
SCM [1].

See http://pages.cpsc.ucalgary.ca/~gleighto/research/xml-comp.html for a more exhaustive reference.

Navigating and Searching in Compressed Form

The most popular retrieval operations on semi-structured text are related to *navigating* the tree and to *searching* it. Navigating means moving from a node to its children, parent, and siblings. Searching means various *path matching* operations such as finding all the paths where a node labeled A is the parent of another labeled B and that one is the ancestor of another labeled C, which in turn contains text where word W appears. A popular language combining navigation and searching operations is XPath (see http://www.w3.org/TR/xpath20).

Several of the schemes above permit accessing and decompressing any part of the text at random positions. This is because they retain the original order of the components of the document and compress using a semi-static model. Those compression methods are transparent, in the sense that the classical techniques to navigate and search XML data, sequentially or using indexes, can be used almost directly over this compressed representation.

Other techniques, such as *SCM* (Huffman variant) or *XCQ*, allow random access under a slightly more complex scheme, because some work is needed in order to start decompression at a specific point. Finally, techniques based on adaptive compression (such as *XMLPPM* or the PPM variant of *SCM*) usually achieve better compression ratios but need to decompress the whole data before they can operate on it.

Some of these techniques, on the other hand, take some advantage of the separation between structure and content in order to run queries faster than scanning all the data. This is the case of *XCQ*, where the table that points from each different tree path to all the contents compressed under the corresponding model, is useful to avoid traversing those contents if the path does not match a path matching query. Another example is *XPress*, which encodes paths in a way that the codes themselves permit checking containment between two paths. A concept that deviates from the ideas presented is that of using a tree representation that permits sharing repeated subtrees. A good exponent is [5], which permits running a large subset of XPath directly over this compressed representation. The structure can be navigated almost transparently, and path matching operations can be sped up by factoring out the work done on repeated substructures.

Succinct Encodings for Labeled Trees

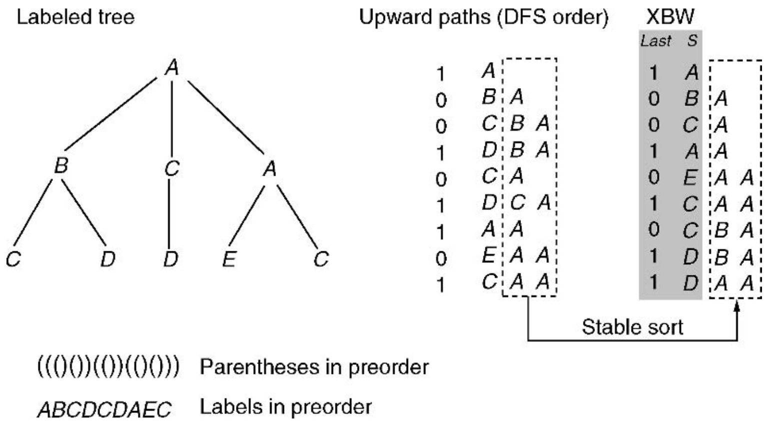
Succinct representations of labeled trees are an algorithmic development that finds applications in navigating semi-structured text in compressed form. In its simplest form, a general labeled tree of n nodes can be represented using a sequence P of $2n$ balanced parentheses and a sequence L of n labels (which correspond to tag names and will be regarded as atomic for simplicity).

This is obtained by traversing the tree in preorder (that is, first the current node and then recursively each of its children). As the tree is traversed, an opening parenthesis is added to P each time one goes down to a child, and a closing parenthesis when going up back to the parent. In L, the labels are added in preorder.

Figure 1 (left) shows an example representation of a labeled tree as a sequence of parentheses and labels in preorder. It is not hard to rebuild the tree from this representation. However, what is really challenging is to navigate the tree directly in this representation (where a node is represented by the position of its opening parenthesis). For this sake, only sublinear extra space on top of the plain representation is needed [9].

An essential operation to achieve efficient navigation in compressed form is the *rank* operation on bitmaps: $rank(P, i)$ is the number of 1s (here representing opening parentheses) in $P[1, i]$. One immediate application of *rank* is to obtain the label of a given node i, as $L[rank(P, i)]$. For example, consider the second child of the root in Fig. 1. It is represented by the opening parenthesis at position 8 in the sequence. Its label is therefore $L[rank(\text{“((()())(())(()()))”}, 8)] = L[5] = \text{“}C\text{”}$. Another application of *rank*, is to compute the depth of a node i. This is the number of opening minus closing parenthesis in $P[1, i]$, that is, $rank(P, i) - (i - rank(P, i)) = 2 \cdot rank(P, i) - i$. For example, the depth of the second child of the root is $2 \cdot rank(\text{“((()())(())(()()))”}, 8) - 8 = 2 \cdot 5 - 8 = 2$.

It is not possible to fully explain, in a short entry, the constant-time solutions to *rank* and other more complex operations needed to navigate the compressed tree. To have a flavor of how those solutions operate, consider the case of supporting *rank* in constant time and $o(n)$ extra bits. Absolute *rank* values are stored at every sth position of P (for some parameter s), and also relative (to the beginning of the last absolute sample) *rank* values at every bth position of P (for some parameter $b < s$). Note that each relative *rank* value



Managing Compressed Structured Text. Figure 1. An example labeled tree and two compression techniques. On the left (*bottom*), as a parentheses plus labels sequence in preorder. On the right, its *xbw* transform. The *dashed boxes* highlight the upward paths. The *grayed box* is the *xbw* transform of the tree.

needs only log s bits to be stored. Then, two table accesses (absolute plus relative *rank*) give the partial *rank* answer up to the b-bit chunk where position i belongs. To complete the query in constant time, a universal table is precomputed, which gives the number of 1-bits in *every possible* chunk of b bits. Some bit masking and a final access to this table suffice to count the remaining 1s within the chunk. By properly choosing s and b one achieves $o(n)$ extra bits overall, and still answers $rank(P, i)$ with three table accesses.

Other basic queries can be computed with similar mechanisms (coarse sampling to store absolute values, finer sampling to store relative values, and universal tables to process short chunks). For example, one can compute $select(P, i)$, the position of the ith opening parenthesis in P, so as to find the tree node corresponding to the ith label in L. Other essential operations for the navigation are $close(i)$, the position of the parenthesis that closes i (that is, the next parenthesis with the same depth of i); and $enclose(i)$, the lowest parenthesis that contains i (that is, the preceding parenthesis with depth smaller than that of i).

With these two operations one can navigate the tree as follows. The next sibling of i is $close(i) + 1$ (unless it is a closing parenthesis, in which case i is the last child of its parent). The first child of i is $i + 1$ unless $P[i + 1]$ is a closing parenthesis, in which case i is a leaf and hence has no children. The parent of i is $enclose(i)$. The size of the subtree rooted at i is $(close(i) - i + 1)/2$. For example, consider the first child of the root in Fig. 1, such that $i = 2$. It finishes at $close(i) = 7$. Its next sibling is $close(i) + 1 = 8$, the node of the previous examples. Its first child is $i + 1 = 3$, the leftmost tree leaf. Its parent is $enclose(i) = 1$, the root. The size of its subtree is $(close(i) - i + 1)/2 = (7 - 2 + 1)/2 = 3$.

In order to enrich the navigation using the labels, sequence $L[1, n]$ is also processed for symbol *rank* and *select* operations, where $rank_c(L, i)$ is the number of occurrences of c in $L[1, i]$ and $select_c(L, j)$ is the position of the jth occurrence of c in L. For example, the following procedure finds all the descendants of node i which are labeled c: (*i*) Find the position $j = rank(P, i)$ of node i in the sequence of labels. (*ii*) Compute $k = rank_c(L, j - 1)$, the number of occurrences of c prior to j. (*iii*) Find the positions $p_r = select_c(L, k + r)$ of c from j onwards, for successive r values until $select(P, p_r) > close(i)$, that is, until the answers are not anymore descendants of i. For example, consider again the first child of the root in Fig. 1, where $i = 2$ and $close(i) = 7$, and find its descendants labeled "D". The first step is to compute $j = rank(P, 2) = 2$, the position of its label in L. Now, $k = rank_{\text{“}D\text{”}}(L, 1) = 0$ tells that there are zero occurrences of "D" before $L[2]$. Now the next occurrences of "D" in L are found as $select_{\text{“}D\text{”}}(L, 1) = 4$, $select_{\text{“}D\text{”}}(L, 2) = 6$, ... The first such occurrence is mapped to the tree node $select(P, 4) = 5$ (the second tree leaf), which is within the subtree of i because $i \leq 5 \leq close(i)$. The second occurrence of "D" is already outside the tree because $select(P, 6) = 9$ exceeds $close(i) = 7$.

Many other powerful navigational operations can be supported, although a more sophisticated parentheses representation and much more technical developments are necessary. A good example can be seen in [4]. Empirical results have been given for the basic preorder parentheses represenation [9].

Integrating Indexing and Compression

In recent work [7,8], by means of introducing a so-called *xbw transform*, indexing and compression are made part of a single integrated process, so that the compressed data represents at the same time the structured text and an index built on it. This is is a very original idea which is likely to have practical impact in the next years.

A brief description of the transform follows. Imagine one takes all the upward paths in the labeled tree. There is one such path per tree node: given a node, its path starts from its parent and finishes in the root. Regard the upward paths as a sequence of labels, and assume for simplicity that labels are atomic symbols that can be sorted. If the tree has n nodes, the resulting n sequences of labels are collected in depth-first order and then stably sorted in lexicographical order. Finally, one forms a sequence with the nodes that originated each of the upward paths, once they are sorted. The *xbw transform* of a labeled tree is the sequence obtained plus a bitmap telling which of those nodes are the last child of their parent.

For example, take the tree of Fig. 1. The upward path from the root is the empty string. The upward path from the three root children are all "*A*", and so on. The list of all the upward paths found in a depth-first traversal is shown in the middle of the figure, and on the right one can see the paths after a stable lexicographical sorting. Now, collecting the nodes that originated those paths in order one gets the labels in the grayed area, S = "*ABCAECCDD*" (the other element is the bitmap marking the last children of their parents, *last* = "100101011"). Sequence S is essentially a permutation of the tree labels. As the sorting is stable, all the nodes originating the same path (e.g., "*A*") stay in depth-first order. In particular, sibling nodes are contiguous.

It turns out that it is possible to compute the *xbw* transform in linear time and space, and moreover to recover the original tree from these two sequences in linear time and space. Furthermore, it is possible to efficiently navigate the tree in *xbw*-transformed form, with operations such as moving to the parent of the current node, ith child, next sibling, ith child labeled X, and so on. Those operations also build on the *rank* and *select* operations described. For example, the third child of the root is represented by the upward path "*AA*". Its position after the *xbw* sorting is $i = 4$, which acts as the identifier for the node in this representation, note $S[4]$ = "*A*" is its label. The process to find its children is as follows: (*i*) Find how many "*A*"s are there before in S, $j = rank_{\text{“}A\text{”}}(S, i-1) = 1$. (*ii*) Find the beginning of the range of the upward paths starting with $S[i]$ = "*A*" in the dashed box, $k = 2$ (this is precomputed in a table storing such value for each different label). (*iii*) Find the number of 1s in *last* before that range, $l = rank(last, k-1) = 1$. (*iv*). Find the area corresponding to the children of i, $select(last, l+j)+1 = 5$ to $select(last, l+j+1) = 6$.

The key operation that makes the *xbw* transform unique compared to other tree representations its its path searching ability: It can identify all the nodes in the tree that descend from a given path sequence in time proportional to the length of the sequence and independent of the collection size (the nodes can then be retrieved one by one). That is, the *xbw*-transformed sequence acts not only as a navigable representation of the tree but also as a powerful index to carry out some path searching operations very efficiently.

Apart from saving all the pointer information, the *xbw*-transformed sequence groups together the node labels that descend from the same paths. Therefore, if root-to-node paths are good predictors of the contents of nodes (this is the property that most sophisticated techniques like *XMLPPM* exploit), the transformed sequence will contain long regions with similar contents. Those are easily compressible by block-wise encoding methods.

In [8] they showed how to apply this conceptual method to real XML data, mixing location path operations with queries on the text content. They present a practical implementation and empirical results showing that it is competitive with the best XML compressors, which do not offer simultaneous indexing capabilities.

Key Applications

Any application managing semi-structured text, particularly if it has to transmit it over slow channels or operate within limited fast memory, even if there is an unlimited supply of slower memory, benefits from these techniques.

Future Directions

Several problems remain open. A fundamental one is the definition of an adequate notion of entropy for semi-structured data, that is, a compressibility limit. While there is reasonable consensus on the entropy of plain text without structure (by taking it as a sequence in general), there is no agreement even on how to measure the entropy of a tree, which is a key part of the entropy of semi-structured text. The fact that this issue is open implies that it is hard to determine how good, in absolute terms, is a compression scheme.

The future of the area is likely to be in manipulating XML in compressed form, and in this aspect the *xbw* transform is a promising direction. Yet, the area is far from offering a competitive and complete path search engine over compressed XML, for example. Similarly, the state of the art in permitting manipulating compressed XML, for example updating a semi-structured text collection, is very preliminary (see [11] for a recent, still theoretical, work on dynamizing the *xbw* transform).

Experimental Results

Experiments can be found in the papers cited. In particular, for the *xbw* transform, see [8].

URL to Code

Several public XML compressors are available, for example

XMill (http://sourceforge.net/projects/xmill), *XMLPPM* (http://sourceforge.net/projects/xmlppm), *SCMPPM* (http://www.infor.uva.es/∼jadiego/download.php), and

XGrind (http://cvs.sourceforge.net/viewcvs.py/xmill/xmill/XGrind).

Cross-references

- ▶ Compression
- ▶ Semi-Structured Data
- ▶ XML
- ▶ XPath/XQuery

Recommended Reading

1. Adiego J., Navarro G., and de la Fuente P. Using Structural Contexts to Compress Semistructured Text Collections. Inf. Proc. & Man., 43:769–790, 2007.
2. Baeza-Yates R. and Navarro G. Integrating contents and structure in text retrieval. ACM SIGMOD Rec., 25(1):67–79, 1996.
3. Barbay J., Golynski A., Munro I., and Rao S. Adaptive searching in succinctly encoded binary relations and tree-structured documents. In Proc. 17th Annual Symp. on Combinatorial Pattern Matching, 2006, pp. 24–35.
4. Bell T., Cleary J., and Witten I. Text Compression. Prentice Hall, Englewood Cliffs, NJ, 1990.
5. Buneman, P., Grohe, M., and Koch, C. Path Queries on Compressed XML. In Proc. 29th Very Large Databases Conference, 2003, pp. 141–152.
6. Cheney J. Compressing XML with multiplexed hierarchical PPM models. In Proc. 11th IEEE Data Compression Conf., 2001, pp. 163–172.
7. Ferragina P., Luccio F., Manzini G., and Muthukrishnan S. Structuring labeled trees for optimal succinctness, and beyond. In Proc. 46th Annu. Symp. on Foundations of Computer Science, 2005, pp. 184–196.
8. Ferragina P., Luccio F., Manzini G., and Muthukrishnan S. Compressing and searching XML data via two zips. In Proc. 15th Int. World Wide Web Conf., 2006.
9. Geary R., Rahman N., Raman R., and Raman V. A simple optimal representation of balanced parentheses. Theoretical Computer Science, 368(3):231–246, 2006.
10. Girardot M. and Sundaresan N. Millau: An encoding format for efficient representation and exchange of XML documents over the WWW. In Proc. 9th Int. World Wide Web Conference, 2000, pp. 747–765.
11. Gupta A., Hon W.K., Shah R., and Vitter J. A framework for dynamizing succinct data structures. In Proc. 34th Int. Colloquium on Automata, Languages, and Programming, 2007, pp. 521–532.
12. Levene M. and Wood P. XML structure compression. In Proc. 2nd Int. Workshop on Web Dynamics, 2002.
13. Liefke H. and Suciu D. XMill: an efficient compressor for XML data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 153–164.
14. Min J.K., Park M.J., and Chung C.W. XPress: a querieable compression for XML data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 122–133.
15. Tolani P. and Haritsa J. XGRIND: A query-friendly XML compressor. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 225–234.

Mandatory Access Control

Bhavani Thuraisingham
The University of Texas at Dallas, Richardson, TX, USA

Synonyms

Multilevel security

Definition

As stated in [1], "*in computer security, 'mandatory access control (MAC)' refers to a kind of access control defined by the National Computer Security Center's Trusted Computer System Evaluation Criteria (TCSEC) as a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such* *sensitivity.*" With operating systems, the subjects are processes and objects are files. The goal is to ensure that when a subject accesses a file, no unauthorized information is leaked.

Key Point

MAC Models: MAC models were developed initially for secure operating systems mainly in the 1970s and early 1980s, and started with the Bell and La Padula security model. This model has two properties: the simple security property and the *-property (pronounced the star property). The simple security property states that a subject has read access to an object if the subject's security level dominated the level of the object. The *-property states that a subject has write access to an object if the subject's security level is dominated by that of the object [2]. Since then, variations of this model as well as a popular model called the noninterference model [3] have been proposed. The noninterference model is essentially about higher-level processes not interfering with lower level processes. Note that with the Bell and La Padula model, a higher level process can covertly send information to a lower level process by manipulating the file locks, even though there can be no write down due to the star property. The noninterference model prevents such covert communication.

MAC for Database Systems: While Database Management Systems (DBMS) must deal with many of the same security concerns as operating systems (identification and authentication, access control, auditing), there are characteristics of DBMSs that introduce additional security challenges. For example, objects in DBMSs tend to be of varying sizes and can be of fine granularity such as relations, attributes and elements. This contrasts with operating systems where the granularity tends to be coarse such as files or segments. Because of the fine granularity in database systems the objects on which MAC is performed may differ. In operating systems MAC is usually performed on the same object such as a file whereas in DBMSs it could be on relations and attributes. The simple security and * property are both applicable for database systems. However many of the database systems have modified, the *-property to read as follows: *A subject has write access to an object if the subject's level is that of the object.* This means a subject can modify relations at its level. Various commercial secure DBMS products have emerged. These products have been evaluated using the Trusted Database Interpretation which interprets the TCSEC for database systems.

MAC for Networks: For applications in defense and intelligence multilevel secure networks are essential. The idea here is for the network protocols such as a TCP/IP (Transmission Control Protocol/Internet Protocol) protocols operate at multiple security levels. The Bell and La Padula model has been extended for networks. Furthermore, the commercial multilevel networks have been evaluated using the Trusted Network Interpretation that interprets the TCSEC for networks.

Cross-references

▶ Multilevel Secure Database Management System

Recommended Reading

1. http://en.wikipedia.org/wiki/Mandatory_access_control
2. Bell D. and LaPadula L. "Secure Computer Systems: Mathematical Foundations and Model," M74–244. The MITRE Corporation, Bedford, MA, 1973.
3. Goguen J. and Meseguer J. Security policies and security models. In Proc. IEEE Symp. on Security and Privacy, 1982, pp. 11–20.

MANET Databases

YAN LUO, OURI WOLFSON
University of Illinois at Chicago, Chicago, IL, USA

Synonyms

Mobile ad hoc network databases

Definition

A mobile ad hoc network (MANET) database is a database that is stored in the peers of a MANET. The network is composed by a finite set of mobile peers that communicate with each other via short range wireless protocols, such as IEEE 802.11, Bluetooth, Zigbee, or Ultra Wide Band (UWB). These protocols provide broadband (typically tens of Mbps) but short-range (typically 10–100 m) wireless communication. On each mobile peer there is a local database that stores and manages a collection of data items, or reports. A report is a set of values sensed or entered by the user at a particular time, or otherwise obtained by a mobile peer. Often a report describes a physical resource such as an available parking slot. All the local databases maintained by the mobile peers form the

MANET database. The peers communicate reports and queries to neighbors directly, and the reports and queries propagate by transitive multi-hop transmissions. Figure 1 below illustrates the definition.

MANET databases enable matchmaking or resource discovery services in many application domains, including social networks, transportation, mobile electronic commerce, emergency response, and homeland security.

Communication is often restricted by bandwidth and power constraints on the mobile peers. Furthermore, reports need to be stored and later forwarded, thus memory constraints on the mobile devices constitute a problem as well. Thus, careful and efficient utilization of scarce peer resources (specifically bandwidth, power, and memory) are an important challenge for MANET databases.

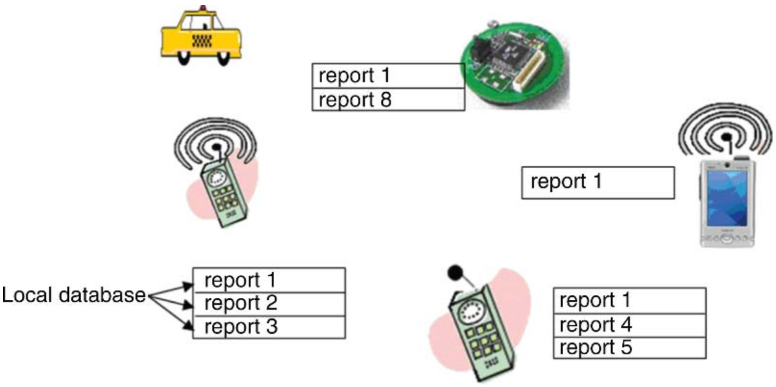
Historical Background

Consider mobile users that search for local resources. Assuming that the information about the existence and location of such a resource resides on a server, a communication infrastructure is necessary to access the server. Such an infrastructure may not be available in military/combat situations, disaster recovery, in a commercial flight, etc. Even if the infrastructure and a server are both available, a user may not be willing to pay the dollar-cost that is usually involved in accessing the server through the cellular infrastructure. Furthermore, cellular bandwidth is limited (e.g., 130 character text messages). In other words, a client-server approach may have accessibility problems.

Currently, Google and local.com provide static local information (e.g., the location of a restaurant, pharmacy, etc.), but not dynamic information such as the location of a taxi cab, a nearby person of interest, or an available parking slot. These dynamic resources are temporary in nature, and thus require timely, real-time update rates. Such rates are unlikely to be provided for the country or the world by a centralized server farm, e.g., Google. Thus, dynamic local resources may require local servers, each dedicated to a limited geographic area. However, for many areas such a local server may not exist due to lack of a profitable business model, and if it exists it may be unavailable (such servers are unlikely to have the reliability of global sites such as Google). Furthermore, the data on the server may be unavailable due to propagation delays (think of sudden-brake information that needs to be propagated to a server and from there to the trailing vehicles), or due to device limitations (e.g., a cab customer's cell-phone may have Bluetooth but not internet access to update the server), or due to the fact that updates from mobile devices may involve a communication cost that nobody is willing to pay, or due to the fact that the local server (e.g., of Starbucks) may accept only updates from certain users or certain applications but not others. In short, a client-(local)-server may have both accessibility and availability problems.

Thus, a MANET database can substitute or augment the client-(local)-server approach. Communication in the MANET is free since it uses the unlicensed spectrum, and larger in bandwidth than the cellular infrastructure, thus can provide media rich information, such as maps, menus, and even video. A mobile user may search the MANET database only, or combine it with a client-server search.

Currently, there are quite a few experimental projects in MANET databases. These can be roughly classified into pedestrians and vehicular projects. Vehicular



MANET Databases. Figure 1. A MANET database.

projects deal with high mobility and high communication topology change-rates, whereas pedestrians projects have a strong concern with power issues. The following are several active experimental MANET database projects for pedestrians and vehicles:

Pedestrians Projects

- *7DS* – Columbia University
  - http://www.cs.unc.edu/~maria/7ds/
  - Focuses on accessing web pages in environments where only some peers have access to the fixed infrastructure.
- *iClouds* – Darmstadt University
  - http://iclouds.tk.informatik.tu-darmstadt.de/
  - Focuses on the provision of incentives to brokers (intermediaries) to participate in MANET databases.
- *MoGATU* – University of Maryland, Baltimore County
  - http://mogatu.umbc.edu/
  - Focuses on the processing of complex data management operations, such as joins, in a collaborative fashion.
- *PeopleNet* – National University of Singapore
  - http://www.ece.nus.edu.sg/research/projects/abstract.asp?Prj=101
  - Proposes the concept of information Bazaars, each of which specializes in a particular type of information; reports and queries are propagated to the appropriate bazaar by the fixed infrastructure.
- *MoB* – University of Wisconsin and Cambridge University
  - http://www.cs.wisc.edu/~suman/projects/agora/
  - Focuses on incentives and the sharing among peers of virtual information resources such as bandwidth.
- *Mobi-Dik* – University of Illinois at Chicago
  - http://www.cs.uic.edu/~wolfson/html/p2p.html
  - Focuses on information representing physical resources, and proposes stateless algorithms for query processing, with particular concerns for power, bandwidth, and memory constraints.

Vehicular Projects

- *CarTALK 2000* – A European project
  - http://www.cartalk2000.net/
  - Develops a co-operative driver assistance system based upon inter-vehicle communication and MANET databases via self-organizing vehicular ad hoc networks.
- *FleetNet* – Internet on the Road Project
  - http://www.ccrle.nec.de/Projects/fleetnet.htm
  - Develops a wireless multi-hop ad hoc network for intervehicle communication to improve the driver’s and passenger’s safety and comfort. A data dissemination method called “contention-based forwarding” (CBF) is proposed in which the next hop in the forwarding process is selected through a distributed contention mechanism based on the current positions of neighbors.
- *VII* – Vehicle Infrastructure Integration, a US DOT project
  - http://www.its.dot.gov/vii/
  - The objective of the project is to deploy advanced vehicle-to-vehicle and vehicle-to-infrastructure communications that could keep vehicles from leaving the road and enhance their safe movement through intersections.
- *Grassroots, Trafficview* – Rutgers University

TrafficInfo – University of Illinois at Chicago

- http://paul.rutgers.edu/~gsamir/dataspace/grassroots.html
- http://discolab.rutgers.edu/traffic/veh_apps.htm
- http://cts.cs.uic.edu/
- These projects develop an environment in which each vehicle contributes a small piece of traffic information (its current speed and location) to the network, using the P2P paradigm, and each vehicle aggregates the pieces into a useful picture of the local traffic.

Foundations

There are two main paradigms for answering queries in MANET databases, one is report pulling and the other one is report pushing.

Report pulling means that a mobile peer issues a query which is flooded in the whole network, and the answer-reports will be pulled from the mobile peers that have them (see e.g., [2]). Report pulling is widely used in resource discovery, such as route discovery in mobile ad hoc networks and file discovery by query flooding in wired P2P networks like Gnutella. Flooding in a wireless network is in fact relatively efficient as compared to wired networks because of the wireless broadcast advantage, but there are also disadvantages which will be explained below.

Another possible approach for data dissemination is report pushing. Report pushing is the dual problem of report pulling; reports are flooded, and consumed by peers whose query is answered by received reports. So far there exist mechanisms to broadcast information in the complete network, or in a specific geographic area (geocast), apart from to any one specific mobile node (unicast/mobile ad hoc routing) or any one arbitrary node (anycast). Report pushing paradigm can be further divided into stateful methods and stateless methods. Most stateful methods are topology-based, i.e., they impose a structure of links in the network, and maintain states of data dissemination. PStree [4], which organizes the peers as a tree, is an example of topology based methods.

Another group of stateful methods is cluster- or hierarchy-based method, such as [14], in which moving peers are grouped into some clusters or hierarchies and the cluster heads are randomly selected. Reports are disseminated through the network in a cluster or hierarchy manner, which means that reports are first disseminated to every cluster head, and each cluster head then broadcasts the reports to the member peers in its group. Although cluster- or hierarchy-based methods can minimize the energy dissipation in moving peers, these methods will fail or cost more energy in highly mobile environments since they have to maintain a hierarchy structure and frequently reselect cluster heads.

Another stateful paradigm consists of location-based methods (see [9]). In location-based methods, each moving peer knows the location of itself and its neighbors through some localization techniques, such as GPS or Atomic Multilateration (see [9]).

The simplest location-based data dissemination is Greedy Forwarding, in which each moving peer transmits a report to a neighbor that is closer to the destination than itself. However, Greedy Forwarding can fail in some cases, such as when a report is stuck in local minima, which means that the report stays in a mobile peer whose neighbors are all further from the destination. Therefore, some recovery strategies are proposed, such as GPSR (Greedy Perimeter Stateless Routing [6]). Other location-based methods, such as GAF (Geographic Adaptive Fidelity [17]) and GEAR (Geographical and Energy Aware Routing [18]), take advantage of knowledge about both location and energy to disseminate information and resources more efficiently.

In stateless methods, the most basic and simplest one is flooding-based method, such as [11]. In flooding-based methods, mobile peers simply propagate received reports to all neighboring mobile peers until the destination or maximum hop is reached. Each report is propagated as soon as it is received. Flooding-based methods have many advantages, such as no state maintenance, no route discovery, and easy deployment. However, they inherently cannot overcome several problems, such as implosion, overlap, and resource blindness. Implosion refers to the waste of resources taking place when a node forwards a message to a neighbor although the latter may have already received it from another source. Overlap occurs when two nodes read the same report, and thus push into the network the same information. Resource blindness denotes the inability of the protocol to adapt the node's behavior to its current availability of resources, mainly power [12]. Therefore, other stateless methods are proposed, such as gossiping-based methods and negotiation-based methods.

Gossiping-based methods, such as [3], improve flooding by transmitting received reports to a subset of randomly selected neighbors; another option is to have some neighbors simply drop the report. For example, the neighbors that are not themselves interested in the report drop it. The advantages of gossiping-based methods include reducing the implosion and lowering the system overhead. However, dissemination, and thus performance, is reduced compared to pure flooding.

Negotiation-based methods solve the implosion and overlap problem by transmitting first the id's of reports; the reports themselves are transmitted only when requested (see [7]). Thus, some extra data transmission is involved, which costs more memory, bandwidth, and energy. In addition, in negotiation-based methods, moving peers have to generate meta-data or a signature for every report so that negotiation can be carried out, which will increase the system overhead and decrease the efficiency.

Another important stateless paradigm for data dissemination in MANET databases is store-and-forward. In contrast to flooding, store-and-forward does not propagate reports as soon as they are received; rather they are stored and rebroadcast later. This obviously introduces storage and bandwidth problems, if too many reports need to be saved and rebroadcast at the same time. To address these, methods such as [5] rank all the reports in a peer's database in terms of their relevance (or expected utility), and then the reports are communicated and saved in the order of their relevance. Or, the reports requested and communicated are the ones with the relevance above a certain threshold. The notion of relevance quantifies the importance or the expected utility of a report to a peer at a particular time and at a particular location. Other store-and-forward methods include PeopleNet [10] and 7DS [13].

In summary, the paradigms for data dissemination in MANET databases are summarized in Fig. 2 below.

Key Applications

MANET databases provide mobile users a search engine for transient and highly dynamic information in a local geospatial environment. MANET databases employ a unified model for both the cellular infrastructure and the mobile ad hoc environments. When the infrastructure is available, it can be augmented by the MANET database approach.

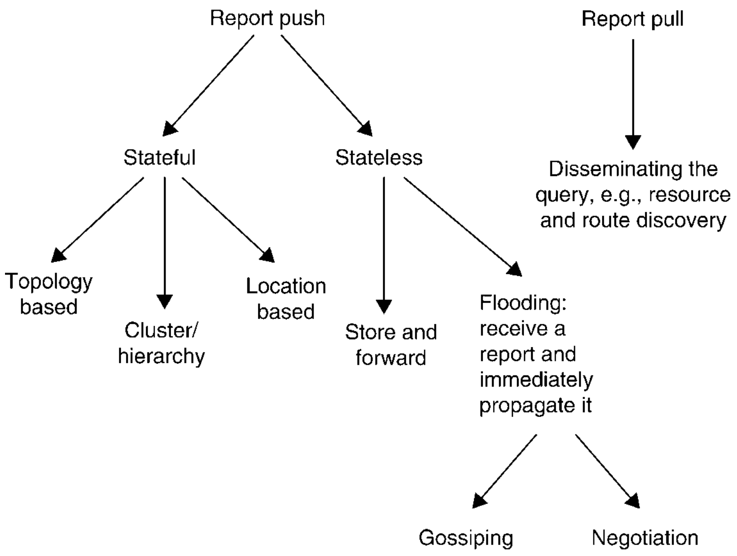
Consider a MANET database platform, i.e., a set of software services for data management in a MANET environment; it is similar to a regular Database Management System, but geared to mobile P2P interactions. Such a platform will enable quick building of matchmaking or resource discovery services in many application domains, including social networks, emergency response and homeland security, the military, airport applications, mobile e-commerce, and transportation.

Social Networks

In a large professional, political, or social gathering, MANET databases are useful to automatically facilitate a face-to-face meeting based on matching profiles. For example, in a professional gathering, MANET databases enable attendees to specify queries (interest profiles) and resource descriptions (expertise) to facilitate face-to-face meetings, when mutual interest is detected. Thus, the individual's profile that is stored in MANET databases will serve as a "wearable web-site." Similarly, MANET databases can facilitate face-to-face meetings for singles matchmaking.

Emergency Response, Homeland Security, and the Military

MANET databases offer the capability to extend decision-making and coordination capability. Consider workers in disaster areas, soldiers and military personnel operating in environments where the wireless fixed infrastructure is significantly degraded or non-existent. As mobile users involved in an emergency response naturally cluster around the location of interest, a self-forming, high-bandwidth network that allows database search without the need of potentially



MANET Databases. Figure 2. Query answering methods in MANET databases.

compromised infrastructure could be of great benefit. For instance, the search could specify a picture of a wanted person.

Airport Applications

A potential opportunity that will benefit both the consumer and the airport operations is the dissemination and querying of real-time information regarding flight changes, delays, queue length, parking information, special security alerts and procedures, and baggage information. This can augment the present audio announcements that often cannot be heard in nearby restaurants, stores, or restrooms, and augment the limited number of displays.

Mobile E-commerce

Consider short-range wireless broadcast and mobile P2P dissemination of a merchant's sale and inventory information. It will enable a customer (whose cell phone is query-capable) who enters a mall to locate a desired product at the best price. When a significant percentage of people have mobile devices that can query retail data, merchants will be motivated to provide inventory/sale/coupons information electronically to nearby potential customers. The information will be disseminated and queried in a P2P fashion (in, say, a mall or airport) by the MANET database.

Transportation Safety and Efficiency

MANET databases can improve safety and mobility by enabling travelers to cooperate intelligently and automatically. A vehicle will be able to automatically and transitively communicate to trailing vehicles its "slow speed" message when it encounters an accident, congestion, or dangerous road surface conditions. This will allow other drivers to make decisions such as finding alternative roads. Also, early warning messages may allow a following vehicle to anticipate sudden braking, or a malfunctioning brake light, and thus prevent pile-ups in some situations. Similarly, other resource information, such as ridesharing opportunities, transfer protection (transfer bus requested to wait for passengers), will be propagated transitively, improving the efficiency of the transportation system.

Future Directions

Further work is necessary on data models for mobile P2P search applications. Work on sensor databases (e.g., Tinydb [8]) addresses data-models and languages for sensors, but considers query processing in an environment of static peers (see e.g., POS [1]). Cartel [5] addresses the translation of these abstractions to an environment in which cars transfer collected data to a central database via fixed access points. Work on MANET protocols deals mainly with routing and multicasting. In this landscape there is a gap, namely general query-processing in MANET's; such processing needs to be cognizant of many issues related to peer-mobility. For example, existing mobile P2P query processing methods deal with simple queries, e.g., selections; each query is satisfied by one or more reports. However, in many application classes one may be interested in more sophisticated queries, e.g., aggregation. For instance, in mobile electronic commerce a user may be interested in the minimum gas price within the next 30 miles on the highway. Processing of such P2P queries may present interesting optimization opportunities.

After information about a mobile resource is found, localization is often critical for finding the physical resource. However, existing (self)-localization techniques are insufficient. For example, GPS is not available indoors and the accuracy of GPS is not reliable. Thus, furthering the state of the art on localization is important for mobile P2P search.

As discussed above, MANET databases do not guarantee answer completeness. In this sense, the integration with an available infrastructure such as the internet or a cellular network may improve performance significantly. This integration has two aspects. First, using the communication infrastructure in order to process queries more efficiently; and second, using data on the fixed network in order to provide better and more answers to a query. The seamless integration of MANET databases and infrastructure databases introduces important research challenges.

Other important research directions include: incentives for broker participation in query processing (see [16]), and transactions/atomicity/recovery issues in databases distributed over mobile peers (virtual currency must be transferred from one peer to another in an atomic fashion, otherwise may be lost).

Of course, work on efficient resource utilization in mobile peers, and coping with sparse networks and dynamic topologies is still very important for mobile P2P search.

Cross-references

▶ Mobile Ad hoc Network Databases
▶ Peer-to-peer Database
▶ Peer-to-peer Network

Recommended Reading

1. Cox L., Castro M., and Rowstron A. POS: Practical Order Statistics for wireless sensor networks. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2006, pp. 52.
2. Das S.M., Pucha H., and Hu Y.C. Ekta: An efficient DHT substrate for distributed applications in Mobile Ad hoc Networks. In Proc. Sixth IEEE Workshop on Mobile Computing Systems and Applications, 2004, pp. 163–173.
3. Datta A., Quarteroni S., and Aberer K. Autonomous gossiping: a self-organizing epidemic algorithm for selective information dissemination in wireless Mobile Ad-Hoc Networks. In Proc. Int. Conf. Semantics of a Networked World, 2004, pp. 126–143.
4. Huang Y. and Molina H.G. Publish/subscribe in a mobile environment. In Proc. 2nd ACM Int. Workshop on Data Eng. for Wireless and Mobile Access, 2001, pp. 27–34
5. Hull B. et al. CarTel: a distributed mobile sensor computing system. In Proc. 4th Int. Conf. on Embedded Networked Sensor Systems, 2006, pp. 125–138.
6. Karp B. and Kung H.T. GPSR: Greedy Perimeter Stateless Routing for wireless sensor networks. In Proc. 6th Annual Int. Conf. on Mobile Computing and Networking, 2000, pp. 243–254.
7. Kulik J., Heinzelman W., and Balakrishnan H. Negotiation-based protocols for disseminating information in wireless sensor networks. Wireless Netw., 8:169–185, 2002.
8. Madden S.R., Franklin M.J., Hellerstein J.M., and Hong W. Tiny DB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst., 30(1):122–173, 2005.
9. Mauve M., Widmer A., and Hartenstein H. A survey on position-based routing in Mobile Ad-Hoc Networks. IEEE Netw., 15(6):30–39, 2001.
10. Motani M., Srinivasan V., and Nuggehalli P. PeopleNet: engineering a wireless virtual social network. In Proc. 11th Annual Int. Conf. on Mobile Computing and Networking, 2005, pp. 243–257.
11. Oliveira R., Bernardo L., and Pinto P. Flooding techniques for resource discovery on high mobility MANETs. In Proc. Int. Workshop on Wireless Ad-hoc Networks, 2005.
12. Papadopoulos A.A., and McCann J.A. Towards the design of an energy-efficient, location-aware routing protocol for mobile, Ad-hoc Sensor Networks. In Proc. of the 15th Int. Workshop on Database and Expert Systems Applications, 2004, pp. 705–709.
13. Papadopouli M. and Schulzrinne H. Design and implementation of a P2P Data dissemination and prefetching tool for mobile users. In Proc. 1st New York Metro Area Networking Workshop, IBM TJ Watson Research Center. Hawthorne, NY, 2001.
14. Visvanathan A., Youn J.H., and Deogun J. Hierarchical Data Dissemination Scheme for Large Scale Sensor Networks. In Proc. IEEE Int. Conf. on Communications, 2005, pp. 3030–3036.
15. Wolfson O., Xu B., Yin H.B., and Cao H. Search-and-discover in Mobile P2P Network Databases. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2006, pp. 65.
16. Xu B., Wolfson O., and Rishe N. Benefit and pricing of spatio-temporal information in Mobile Peer-to-Peer Networks. In Proc. 39th Annual Hawaii Conf. on System Sciences, vol. 9, 2006, pp. 2236.
17. Xu Y., Heidemann J., and Estrin D. Geography-informed energy conservation for Ad hoc Routing. In Proc. 7th Annual Int. Conf. on Mobile Computing and Networking, 2001, pp. 70–84.
18. Yu Y., Govindan R., and Estrin D. Geographical and Energy Aware Routing: a Recursive Data Dissemination Protocol for Wireless Sensor Networks. Technical Report UCLA/CSD-TR-01-0023, UCLA, May 2001.

Manmachine Interaction (Obsolete)

▶ Human-Computer Interaction

Many Sorted Algebra

▶ Data Types in Scientific Data Management

MAP

Steven M. Beitzel[1], Eric C. Jensen[2], Ophir Frieder[3]
[1]Telcordia Technologies, Piscataway, NJ, USA
[2]Twitter, Inc., San Fransisco, CA, USA
[3]Georgetown University, Washington, DC, USA

Synonyms

Mean average precision

Definition

The Mean Average Precision (MAP) is the arithmetic mean of the average precision values for an information retrieval system over a set of *n* query topics. It can be expressed as follows:

$$MAP = \frac{1}{n}\sum_{n} AP_n$$

where *AP* represents the Average Precision value for a given topic from the evaluation set of *n* topics.

Key Points

The Mean Average Precision evaluation metric has long been used as the de facto "gold standard" for information retrieval system evaluation at the NIST Text Retrieval Conference (TREC) [1]. Many TREC tracks over the years have evaluated run submissions using the *trec_eval* program, which calculates Mean Average Precision, along with several other evaluation metrics. Much of the published research in the information retrieval field over the last 25 years relies on observed difference in MAP to draw conclusions about the effectiveness of a studied technique or system relative to a baseline.

Recently, the explosive growth of the World Wide Web and the corresponding difficulty of creating test collections that are representative, robust, and of appropriate scale has created new challenges for the research community. One such challenge is how to best evaluate systems in cases of incomplete relevance information. It has been shown that ranking systems by their MAP scores when relevance information is incomplete does not correlate highly with their rankings with complete judgments. This is a key weakness of MAP as a metric. In response to this problem, new metrics (such as BPref, for example) have been proposed that attempt to compensate for often incomplete relevance information [2].

Cross-references

- ▶ Average Precision
- ▶ BPref
- ▶ Chart
- ▶ Effectiveness Involving Multiple Queries
- ▶ Geometric Mean Average Precision

Recommended Reading

1. National Institute of Standards and Technology. TREC-2004 common evaluation measures. Available online at: http://trec.nist.gov/pubs/trec14/appendices/CE.MEASURES05.pdf (retrieved on August 27, 2007).
2. Sakai T. Alternatives to BPref. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 71–78.

Map Algebra

▶ Spatial Operations and Map Operations

Map Matching

Christian S. Jensen, Nerius Tradišauskas
[1]Aalborg University, Aalborg, Denmark

Synonyms

Position snapping

Definition

Map matching denotes a procedure that assigns geographical objects to locations on a digital map. The most typical geographical objects are point positions obtained from a positioning system, often a GPS receiver. In typical uses, the GPS positions derive from a receiver located in a vehicle or other moving object traveling in a road network, and the digital map models the embedding into geographical space of the roads by means of polylines that approximate the center lines of the roads. The GPS positions generally do not intersect with the polylines, due to inaccuracies. The aim of map matching is then to place the GPS positions at their "right" locations on the polylines in the map.

Map matching is useful for a number of purposes. Map matching is used when a navigation system displays the vehicle's location on a map. In many applications, information such as speed limits are assigned to the representations of roads in a digital map–map matching offers a means of relating such information to moving objects. Map matching may also be used for the representation of a route of a vehicle by means of the (sub-) polylines in the digital map.

Two general types of map matching exist, namely on-line map matching and off-line map matching. With on-line map matching, the map location of an object's current position needs to be determined in real time. Only past, but not future, positions are available. Vehicle navigation systems exemplify this type of map matching. In off-line map matching, a static data set of positions is given, meaning that all future positions are available when map matching a position. Thus better map matching may result when compared to on-line map matching. For example, off-line map matching may be used for billing in pay-per-use scenarios (insurance, road pricing).

Historical Background

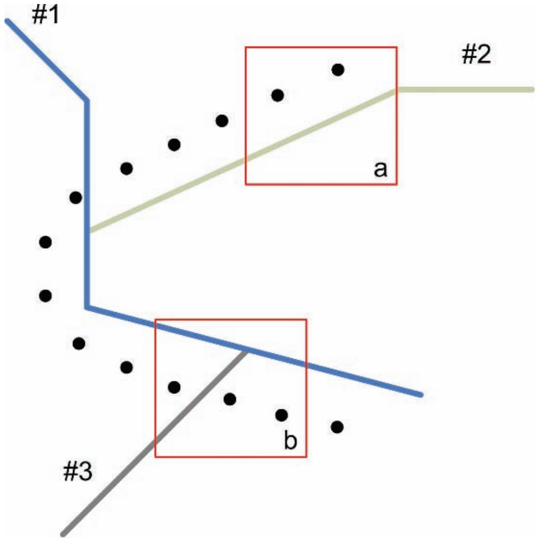
One of the earliest map matching algorithms found in the literature dates back to 1971 and is due to R.L.

French (see the overview in reference [1]). A 1996 paper by Berstein and Kornhauser [2] offers a brief introduction to the map matching problem and its variations.

The scientific literature contains a range of papers that address different aspects of the map matching problem. White et al. [3] study techniques that pay special attention to intersection areas, where map matching can be particularly challenging. Taylor et al. [4] propose a map matching technique that uses differential corrections and height, which leads to improved performance. Quddus et al. [5] provide a summary of different on-line and off-line map matching algorithms and describe advantages and disadvantages of these. Quddus et al. [6] have most recently proposed a map matching algorithm that utilizes techniques from fuzzy logic. This technique shows improved accuracy of polyline identification and the positioning on polylines. A complex off-line map matching algorithm was recently developed by Bratkatsoulas et al. [7] that uses the Fréchet distance to map match GPS position samples recorded only every 30 seconds. (While GPS receivers typically output a position every second, it may be that only some of these are saved for use in subsequent off-line map matching).

Foundations

Basics. The most common use of map matching occurs in transportation where the GPS positions obtained from a GPS receiver in a vehicle are map matched to a digital representation of a road network. An example of GPS positions from a vehicle (dots) and a digital road network are shown in Fig. 1. The vehicle's trip started on road #2 and continued along road #1. In Fig. 2a the start of the trip is enlarged. The dots represent the vehicle's GPS locations, and the triangles represent the corresponding positions map matched onto the road network. The road network locations are typically expressed by using linear referencing, which is a standard means of indicating such locations. With linear referencing, a tuple (#2,5.2,+1) captures the road that the vehicle is driving on (#2), a position on that road measured as a distance from the beginning of the road (5.2 distance units). The third element captures a perpendicular distance from the road location given by the first two elements. In the tuple, the displacement is one distance unit to the left. Because a GPS position is typically mapped to the closest location in the road digital network (i.e., a perpendicular projection is used), the third element is capable of capturing the GPS position that was map matched to the road network location. Linear referencing is supported by, e.g., the Oracle DBMS [6].



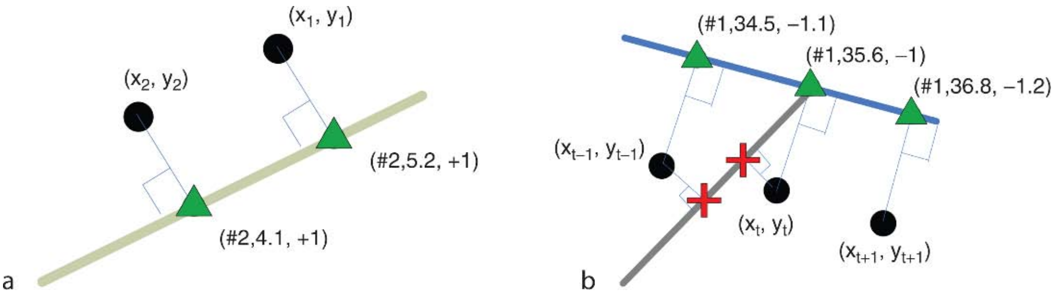
Map Matching. Figure 1. Map and vehicle location example.

In Fig. 2b, a place where map matching is challenging is enlarged. The crosses represent two instance of wrong map matching to the nearest road, and the triangles represent the correct map matching.

Categorization of Map Matching Algorithms. Map matching can be divided into on-line and off-line map matching. On-line map matching occurs in real-time. Here, the map matching algorithm tries to identify the network location of a GPS position every time a new position is received. The algorithm has available the current position as well as information about the map matching of previous positions.

This contrasts off-line map matching, which occurs after a trip is over and all the positions from the start to the end are known. Off-line map matching is more accurate than on-line map matching, as more information (i.e., future positions) is available. An off-line algorithm does not provide a map matching result until the entire trip has been map matched.

On-line map matching is mostly used in vehicle navigation, tracking, and other applications that need the most recent network location of a vehicle. Off-line map matching is mostly used to determine as accurately as possible which route a vehicle was driving. Off-line map matching may also be used in scenarios where



Map Matching. Figure 2. Map matching example.

GPS data are received in batches from content providers and where the purpose is to build speed maps that capture the expected travel speeds for different road segments and time intervals. Network locations of vehicles are essential in road load analysis, road pricing, and similar applications.

Map matching algorithms can also be divided into point-to-point, point-to-polyline, and polyline-to-polyline approaches. In point-to-point map matching, a point out of a point set is identified as a match for the given position. In point-to-polyline matching, a polyline in a polyline set is identified, and a point on the polyline is identified that represents the given point as a polyline-set location. In the typical scenario, the polyline set represents a road network, and the position is a GPS position. Finally, in polyline-to-polyline map matching, polylines are identified from the polylines in a road network that best match a given polyline that is usually constructed from point positions.

Map Matching Principles. This section follows the explanation of the basic principles of map matching by considering in some detail the commonly used point-to-polyline map matching for the on-line case.

Map matching algorithms often consist of two overall steps (some algorithms skip the first of these two steps or include an extra step).

In the start-up step, the polyline in the digital road network on which the vehicle is initially located is found. Specifically, map matching is done for the first GPS position received. The correct outcome of this step is very important, as the map matching algorithms use the connections between roads, or the road network topology, to determine the ensuing polylines of the vehicle's movement path. Therefore, algorithms often perform special operations to determine a reliable match for the first GPS position.

In the steady-state step, the subsequent polylines in the digital road network are identified to form the route along which the vehicle is moving. This subsequent map matching follows a standard pattern:

1. Extract the relevant information from the record received from the GPS receiver.
2. Select candidate polylines from the digital road network.
3. Use algorithm-specific heuristics to determine the most suitable polyline among the candidate polylines.
4. Determine the vehicle position on the selected polyline.

First, information such as latitude, longitude, speed, and heading is extracted from the record obtained from the positioning unit and is converted into an appropriate format (a unified coordinate and metric system consistent with the digital road network).

Second, the candidate polylines are selected. Usually the polylines that are within a certain threshold distance of the GPS position are selected. An alternative approach is to select the *n* polylines nearest to the GPS position. The polylines found make up the candidate polyline set.

Third, specific heuristics are used to select the best polyline among the candidates. A common approach is to assign weights to the candidate polylines according to different criteria. The polyline with the highest sum of weights is then chosen. Some algorithms reduce the set of candidate polylines prior to assigning weights. For example, polylines that are perpendicular to the vehicle's heading may be disregarded, as may polylines that are not connected with the polyline currently being considered.

Fourth, the vehicle position on the selected polyline is found. The usual approach is to select the location on the polyline that is closest to the vehicle's position. This is a point-to-polyline projection. The projection of the position may be an end point of a line segment in the polyline, or it may be in the interior of a line segment. Quddus et al. [5] propose a more sophisticated approach that uses both the distance traveled since the last map matching and the "raw" projection of the GPS position onto the polyline.

Dead Reckoning. In certain regions, the signals emitted by the GPS satellites may be very weak due to obstacles. This in turn degrades the accuracies of the positions produced by GPS receivers in those regions, and in some cases no GPS position may be produced. In such cases, both on-line and off-line map matching algorithms may utilize dead reckoning to estimate the movement of a vehicle in the road network. The use of dead reckoning is particularly attractive when the average speed of the vehicle is known (preferably every second) and when the road on which the vehicle is driving neither splits nor has intersections. This occurs when the road is inside a tunnel with no exits.

Heuristics for the Selection of a Polyline. Different algorithms use different heuristics and weights when attempting to identify the best polyline among the candidate polylines. Each candidate polyline is assigned a weight for each criterion considered, and the polyline with the highest sum of weights is then selected. Common weighting criteria include the following:

- Weight for the proximity of a polyline. A polyline is assigned a weight according to its proximity to the GPS position being map matched. It is natural to assume that the closer a polyline is to the position, the better a candidate the polyline is.
- Weight for the continuity of a polyline. This weight is assigned to each polyline for being a continuation of the previously map matched polyline. This weight represents the reasoning that vehicles tend to drive on the same road most of the time.
- Weight for direction similarity. Polylines whose bearing is similar to the vehicle's heading are assigned higher values.
- Weight for topology. Higher weights are added to polylines that are connected to the polyline currently being map matched to.

Algorithms may also include weights for speed limit changes, shortest distance, road category, one-way streets, etc. Off-line map matching algorithms may use fewer, but more robust weights that are not suitable for the on-line map matching algorithms.

Execution Time Constraints and Accuracy. In on-line map matching, the algorithms must keep up with the GPS device that usually emits one position per second. With current on-board computing units, it is usually not a problem for on-line map matching algorithms map match a GPS position within 1 second. For off-line map matching, there are no real-time constraints.

Key Applications

Map matching is essential for applications that rely on the positioning of a user within a road network.

This occurs in vehicle navigation where the user's position is to be displayed so that it coincides with the road network. When GPS data is used for the construction of speed maps, map matching is used. Such speed maps may be used for travel time prediction, route construction, and capacity planning. In metered services such as insurance and road pricing, map matching is also used.

Further, map matching is used in location-based services where the content to be retrieved is positioned within the road network. Services that offer network-context awareness utilize map matching. These including current network location context awareness [9], and route context awareness [10].

Tracking also plays a key role in intelligent speed adaptation [11] where drivers are alerted when they exceed the current speed limit. The speed limits are attached to the digital road network, so map matching is needed to identify the current speed limit. Finally, map matching is important for a range of other application within intelligent transportation systems.

Future Directions

In the near future, digital road networks will have accurate lane information embedded, and positioning technologies will be accurate enough to enable lane-level positioning. This will necessitate the extension of map matching techniques to function at the lane level.

Cross-references

- ▶ Compression of Mobile Location Data
- ▶ Location Prediction
- ▶ Location-Based Services
- ▶ Mobile Database
- ▶ Mobile Sensor Network Data Management
- ▶ Road Networks

- ► Spatial Network Databases
- ► Spatial Operations and Map Operations
- ► Spatio-Temporal Trajectories
- ► Spatiotemporal Interpolation Algorithms

Recommended Reading

1. Bernstein D. and Kornhauser A. An introduction to map matching for personal navigation assistants. New Jersey TIDE Center, 1996. http://www.njtide.org/reports/mapmatchintro.pdf.
2. Brakatsoulas S., Pfoser D., Salas R., and Wenk C. On map-matching vehicle tracking data. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 853–864.
3. Brilingaitė A. and Jensen C.S. Enabling routes of road network constrained movements as mobile service context. Geoinformatica, 11(1):55–102, 2007.
4. Civilis A., Jensen C.S., and Pakalnis S. Techniques for efficient road-network-based tracking of moving objects. IEEE Trans. Knowl. Data Eng., 17(5):698–712, 2005.
5. French R.L. Historical overview of automobile navigation technology. In Proc. 36th IEEE Vehicular Technology Conf., 1986, pp. 350–358.
6. Oracle Corporation. Oracle Spatial and Oracle Locator. http://www.oracle.com/technology/products/spatial/index.html.
7. Quddus M., Ochieng W., Zhao L., and Noland R. A general map matching algorithm for transport telematics applications. GPS Solut. J., 7(3):157–167, 2003.
8. Quddus M.A., Noland R.B., and Ochieng W.Y. A high accuracy fuzzy logic based map matching algorithm for road transport. J. Intell. Transport. Syst., 10(3), 2006, pp. 103–115.
9. Taylor G., Blewitt G., Steup D., Corbett S., and Car A. Road reduction filtering for GPS-GIS navigation. Transactions in GIS, 5(3):193–207, 2001.
10. Tradisauskas N., Juhl J., Lahrmann H., and Jensen C.S. Map matching for intelligent speed adaptation. In Proc. 6th European Congress on Intelligent Transport Systems and Services, 2007.
11. White C.E., Bernstein D., and Kornhauser A.L. Some map matching algorithms for personal navigation assistants. Transport. Res. C, 8:91–108, 2000.

Mapping

- ► Mediation
- ► Schema Mapping

Mapping Composition

- ► Schema Mapping Composition

Mapping Engines

- ► Digital Rights Management

Markup Language

ETHAN V. MUNSON
University of Wisconsin-Milwaukee, Milwaukee, WI, USA

Definition

A markup language is specification language that annotates content through the insertion of *marks* into the content itself. Markup languages differ from programming languages in that they treat data, rather than commands or declarations, as the primary element in the language.

Key Points

Markup languages were initially developed for text document formatting systems, though they are not limited to text. In fact, the term *markup* was taken directly from the jargon of the publishing business, where editors and typographers would "mark up" draft documents to indicate corrections or printing effects. Markup languages are generally quite declarative and have little, if any, computational semantics. The marks inserted into the content are often called "tags" because that term is used by XML.

Cross-references

- ► Document
- ► Document Representations (Inclusive Native and Relational)

MashUp

ALEX WUN
University of Toronto, Toronto, ON, Canada

Definition

A MashUp is a web application that combines data from multiple sources, creating a new hybrid web application with functionality unavailable in the original individual applications that sourced the data.

Key Points

An emerging trend in web applications is to provide public APIs for accessing data that has traditionally been used only internally by those applications. The main purpose of providing access to traditionally private web application data is to encourage user-driven development. In other words, consumers are expected to take that public data and build custom applications for other consumers – thereby adding value to the original data sources. MashUps are web applications that take advantage of these publicly accessible data sources by correlating the data obtained from different sources and deriving some novel functionality. A simple and common example is correlating a data source that has location information (such as wireless hotspot locations) with cartographic data (from Google or Yahoo maps for example) to produce a graphical map of wireless hotspots.

MashUps are conceptually related to portals, which also collect data from multiple sources for presentation. However, portals perform server-side aggregation whereas MashUps can also perform this aggregation on the client-side (i.e., correlation can occur in the scripts of a web page). Additionally, portals present data collected from disparate sources together but without interaction between data sets. In contrast, MashUps focus heavily on merging disparate data sets into one unified representation. For example, a news portal would simply present a set of interesting articles gathered from various sources on a single page while a news MashUp would correlate textual news with related images and multimedia as well as automatically linking related articles in a single view.

While similar to MashUps, service composition is a more generic concept that focuses on orchestrating web service calls as part of some higher level application logic. The coordination of web service calls in a service composition are often more process-centric rather than data-centric. For example, a flight-booking composite service would query the flight reservation services of different airlines to book a flight based on customer requirements, while a flight-booking MashUp would gather flight data from various airlines and present the data to customers in a single unified view – likely correlated with other useful data such as weather.

There is currently no standardization of technologies or tools used to develop MashUps. In fact, many industry leaders such as Google, Microsoft, and Yahoo are pushing their own MashUp development tools. In particular, many of these tools are targeting non-programmers in hopes of expanding the base of users capable of contributing to and developing MashUps.

Cross-references

- ► AJAX
- ► Service
- ► Web 2.0/3.0

Massive Array of Idle Disks

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Synonyms

MAID

Definition

The term Massive Array of Idle Disks refers to an energy-efficient disk array which has the capability of changing its disk drives into a low-power mode when the disk drives are not busy. Disk drives of the array may be controlled individually or in a group. The term Massive Array of Idle Disks is often abbreviated to MAID. A MAID disk array may have additional functions such as data migration/replication and access prediction to improve the energy saving.

Key Points

The basic idea of MAID is to save energy by exploiting storage access locality. That is, some disk drives which are installed in a disk array are frequently accessed whereas the others are rarely busy. MAID tries to spin down or power off such "low-temperature" disk drives to decrease the total energy consumption. The original papers [1,2] which introduced MAID in 2002 studied different design choices and configurations. MAID disk arrays are mainly used for archival storage (replacing conventional tape libraries) or near-line storage (which falls between online storage and archival storage).

Cross-references

- ► Storage Power Management

Recommended Reading

1. Colarelli D., and Grunwald D. Massive Arrays of Idle Disks for Storage Archives. In Proc. 2002 ACM/IEEE conf. on Supercomputing, 2002, pp. 1–11.
2. Colarelli D., Grunwald D., and Neufeld M. The Case for Massive Arrays of Idle Disks (MAID). In Proc. 1st USENIX Conf. on File and Storage Technologies, Work-in Progress Reports, 2002.
3. Storage Network Industry Association. The Dictionary of Storage Networking Terminology. Also available at http://www.snia.org/.

Matching

▶ Similarity and Ranking Operations

Materialized Query Tables

▶ Physical Database Design for Relational Databases

Materialized View Maintenance

▶ View Maintenance

Materialized View Redefinition

▶ View Adaptation

Materialized Views

▶ Physical Database Design for Relational Databases

Matrix

▶ Table

Matrix Masking

Stephen E. Fienberg, Jiashun Jin
Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

Adding noise; Data perturbation; Recodings; Sampling; Synthetic data

Definition

Matrix Masking refers to a class of statistical disclosure limitation (SDL) methods used to protect confidentiality of statistical data, transforming an $n \times p$ (cases by variables) data matrix Z through pre- and post-multiplication and the possible addition of noise.

Key Points

Duncan and Pearson [3] and many others subsequently categorize the methodology used for SDL in terms of transformations of an $n \times p$ (cases by variables) data matrix Z of the form

$$Z \rightarrow AZB + C, \qquad (1)$$

where A is a matrix that operates on the n cases, B is a matrix that operates on the p variables, and C is a matrix that adds perturbations or noise.

Matrix masking includes a wide variety of standard approaches to SDL: (i) adding noise, i.e., the C in matrix masking transformation of equation [1]; (ii) releasing a subset of observations (delete rows from Z), i.e., sampling; (iii) cell suppression for cross-classifications; (iv) including simulated data (add rows to Z); (v) releasing a subset of variables (delete columns from Z); (vi) switching selected column values for pairs of rows (data swapping). Even when one has applied a mask to a data set, the possibilities of both identity and attribute disclosure remain, although the risks may be substantially diminished.

Cross-references

▶ Individually Identifiable Data
▶ Inference Control in Statistical Databases
▶ Privacy
▶ Randomization Methods to Ensure Data Privacy
▶ Statistical Disclosure Limitation for Data Access

Recommended Reading

1. Doyle P., Lane J.I., Theeuwes J.J.M., and Zayatz L. (eds.). Confidentiality, Disclosure and Data Access: Theory and Practical Application for Statistical Agencies. Elsevier, New York, 2001.

2. Duncan G.T., Jabine T.B., and De Wolf V.A. (eds.). Private Lives and Public Policies. Report of the Committee on National Statistics' Panel on Confidentiality and Data Access. National Academy Press, WA, USA, 1993.
3. Duncan G.T. and Pearson R.B. Enhancing access to microdata while protecting confidentiality: prospects for the future (with discussion). Stat. Sci., 6:219–239, 1991.
4. Federal Committee on Statistical Methodology. Report on statistical disclosure limitation methodology. Statistical Policy Working Paper 22. U.S. Office of Management and Budget, WA, USA, 1994.

Maximal Itemset Mining

▶ Max-Pattern Mining

Max-Pattern Mining

Guimei Liu
National University of Singapore, Singapore,
Singapore

Synonyms

Maximal itemset mining

Definition

Let $I = \{i_1, i_2 ..., i_n\}$ be a set of items and $D = \{t_1, t_2 ..., t_N\}$ be a transaction database, where $t_i (i \in [1,N])$ is a transaction and $t_i \subseteq I$. Every subset of I is called an *itemset*. If an itemset contains k items, then it is called a k-itemset. The support of an itemset X in D is defined as the percentage of transactions in D containing X, that is, $sup(X) = |\{t|t \in D \wedge X \subseteq t\}|/|D|$. If the support of an itemset exceeds a user-specified minimum support threshold, then the itemset is called a *frequent itemset* or a *frequent pattern*. If an itemset is frequent but none of its supersets is frequent, then the itemset is called a *maximal pattern*. The task of maximal pattern mining is given a minimum support threshold, to enumerate all the maximal patterns from a given transaction database.

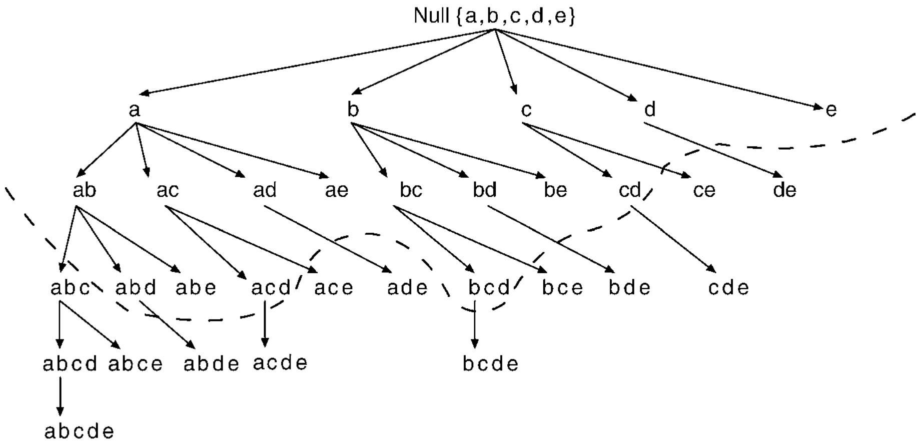
The concept of maximal patterns can be and has already been extended to more complex patterns, such as sequential patterns, frequent subtrees and frequent subgraphs. For each type of pattern, a pattern is maximal if it satisfies the given constraints but none of its super-patterns satisfies the given constraints.

Historical Background

If a k-itemset is frequent, then all of its subsets are frequent and the number of them is $2^k - 1$. Datasets collected from some domains can be very dense and contain very long patterns. Any algorithm which produces the complete set of frequent itemsets suffers from generating numerous short patterns on these datasets, and most of the short patterns may be useless. Some researchers have noticed the long pattern problem and suggested mining only maximal frequent patterns [1,3,8]. The set of maximal patterns provides a concise view of the frequent patterns, and it can be orders of magnitude smaller than the complete set of frequent patterns. The complete set of frequent patterns can be derived from the set of maximal patterns, but the support information is lost.

A different concept called *closed itemset* or closed pattern has also been proposed to reduce result size. A pattern is closed if none of its supersets has the same support as it does. Closed patterns retain the support information of frequent patterns. The complete set of frequent patterns can be derived from the set of frequent closed patterns without information loss. A maximal pattern must be a closed pattern, but not vice versa.

Given a set of items I, the search space of the *frequent itemset* mining problem is the power set of I, and it can be represented as a set-enumeration tree given a specific order of I [9]. Figure 1 shows the search space tree for $I = \{a, b, c, d, e\}$, and the items are sorted lexicographically. Every node in the search space tree represents an itemset. For every itemset X in the tree, only the items after the last item of X can be appended to X to form a longer itemset. These items are called *candidate extensions* of X, denoted as $cand\_ext(X)$. For example, items d and e are candidate extensions of ac, while item b is not a candidate extension of ac because b is before c in lexicographic order. Mining maximal patterns can be viewed as finding a border through the search space tree such that all the nodes below the border are infrequent and all the nodes above the border are frequent. As shown in Fig. 1, the dotted line represents the border. All the nodes above the dotted line are frequent and all the nodes below the dotted line are infrequent. Among all the nodes above the border, only leaf nodes can be maximal, but not all the leaf nodes are maximal; every internal node has at least one frequent child (superset) thus cannot be maximal. The goal of maximal pattern mining is to find the border by counting support for as less as



Max-Pattern Mining. Figure 1. Search space tree for $I = \{a, b, c, d, e\}$.

possible itemsets. Most, if not all, existing maximal pattern mining algorithms try to find some frequent long patterns first, and then use these long patterns to prune non-maximal patterns.

The first attempt at mining maximal patterns is made by Gunopoulos et al. [6], and an algorithm called *dualize and advance* is proposed. The algorithm is based on the observation that given a set of maximal patterns, any other maximal pattern not in the set must contain at least one common item with the complement of every maximal pattern in the set, where the complement of an itemset X is defined as $I - X$. The algorithm works as follows. It first uses a greedy search to generate some maximal patterns, denoted as $\mathcal{H}$, and then finds the minimal patterns that contain at least one common item with the complement of every maximal pattern in $\mathcal{H}$. Here a pattern is minimal if none of its subsets satisfies the condition. These minimal patterns are called minimal transversals of $\mathcal{H}$. If all the minimal transversals of H are infrequent, it means that all the maximal patterns are all in $\mathcal{H}$ already. Otherwise, there exists some minimal transversal X of $\mathcal{H}$ such that X is frequent, the algorithm then finds a maximal superset of X, denoted as Y, and Y must be a maximal pattern. The algorithm puts Y in $\mathcal{H}$, and then generates the minimal transversals of the updated $\mathcal{H}$. This process is repeated until no frequent minimal transversals of $\mathcal{H}$ exists. The upper bound for the time complexity of this algorithm is sub-exponential to the output size.

Pincer-search [7] combines the bottom-up and top-down search strategy, and approaches the border from both directions. The bottom-up search is similar to the A priori algorithm [2], and the top-down search is implemented by maintaining a set called maximum-frequent-candidate-set (MFCS). MFCS is a minimum cardinality set of itemsets such that the union of all the subsets of its elements contains all the frequent itemsets that have been discovered so far but does not contain any itemsets that have been determined to be infrequent. Pincer-search uses MFCS to prune those candidate itemsets that have a frequent superset in MFCS to reduce the database scan times and the support counting cost.

Both the dualize and advance algorithm and the Pincer-search algorithm maintain the set of candidate maximal patterns during the mining process and use them to prune short non-maximal patterns. The main difference between the two algorithms is that Pincer-Search considers large sets that may be frequent first, and then shrinks them to find the real maximal patterns, while the dualize and advance algorithm starts from some seed maximal patterns and uses them to find other maximal patterns. Both algorithms can prune non-maximal patterns effectively, but maintaining and manipulating the set of candidate maximal patterns can be very costly.

Zaki et al. propose two maximal pattern mining algorithms MaxClique and MaxEclat [13]. Both algorithms rely on a preprocessing step to cluster itemsets, and then use a hybrid bottom-up and top-down approach to find maximal patterns from each cluster with a vertical data representation. The two algorithms differ in how the itemsets are clustered. The purpose of the clustering step is to find some potential maximal patterns, and the two algorithms use these potential maximal patterns to restrict the search space. However, the cost of the clustering step can be very high.

Max-Miner [3] is the first successful and practical algorithm for mining maximal patterns. It uses the bottom-up search strategy to traverse the search space as the A priori algorithm [2], but it always attempts to look ahead in order to quickly identify long patterns. By identifying a long pattern first, Max-Miner can prune all its subsets from consideration. Two pruning techniques are proposed in the Max-Miner algorithm: pruning based on superset frequency and the dynamic reordering technique. These two pruning techniques are very effective in removing non-maximal patterns, and have been adopted by all later maximal pattern mining algorithms.

1. *Superset frequency pruning*. This technique is also called the lookahead technique. It is based on the observation that the itemsets in the sub search space tree rooted at X are subsets of $X \cup cand\_ext(X)$. Therefore, if $X \cup cand\_ext(X)$ is frequent, then none of the itemsets in the subtree rooted at X can be maximal and the whole branch can be pruned. There are two ways to check whether $X \cup cand\_ext(X)$ is frequent. One way is to check whether $X \cup cand\_ext(X)$ is a subset of some maximal pattern that has already been discovered. The other way is to look at the support of $X \cup cand\_ext(X)$ in the database, which can be done when counting the support of X's immediate supersets.
2. *Dynamic item reordering*. At every node X, Max-Miner sorts the items in $cand\_ext(X)$ in ascending frequency order. The candidate extensions of an item include all the items that are after it in the ascending frequency order. Let i_1 and i_2 be two items in $cand\_ext(X)$. Item i_1 is before i_2 in the ordering if $sup(X \cup \{i_1\})$ is smaller than $sup(X \cup \{i_2\})$, and item i_2 is a candidate extension of $X \cup \{i_1\}$. The motivation behind dynamic item reordering is to increase the effectiveness of the superset frequency pruning technique. The superset frequency pruning can be applied when $X \cup cand\_ext(X)$ is frequent. It is therefore desirable to make many Xs satisfy this condition. A good heuristic for accomplishing this is to force the most frequent items to be the candidate extensions of all other items because items with high frequency are more likely to be part of long frequent itemsets.

Besides the above two pruning techniques, Max-Miner also uses a technique that can often determine when a new candidate itemset is frequent before accessing the database. The idea is to use information gathered during previous database passes to compute a good lower-bound on the number of transactions that contain the itemset.

The Max-Miner algorithm uses the breadth-first search order to explore the search space, which makes it not very efficient on dense datasets. DepthProject [1], MAFIA [4], GenMax [5] and AFOPT-Max [8] use the depth-first search strategy to traverse the search space. The depth-first search strategy is capable of finding long patterns first, which makes the superset frequency pruning technique more effective. These algorithms differ mainly in their support counting technique. Both DepthProject and MAFIA assume that the dataset fits in the main memory. At any point in the search, DepthProject maintains the projected transaction sets for some of the nodes on the path from the root to the node currently being processed, where a projected transaction of a node contains only the candidate extensions of the node. It is possible that a projected transaction is empty, in this case, the projected transaction is discarded. Since the projected database is substantially smaller than the original database both in terms of the number of transactions and the number of items, the process of finding the support counts is speeded up substantially. DepthProject also uses a bucketing technique to speed up the support counting. MAFIA and GenMax use the vertical mining technique, that is, each itemset is associated with a tid (transaction id) bitmap, and support counting is performed by tid bitmap join. MAFIA uses another pruning technique called parent equivalence pruning (PEP), which is essentially to remove frequent non-closed itemsets. GenMax[5] proposes a progressive focusing technique to improve the efficiency of superset searching. AFOPT-Max uses a prefix-tree structure to store projected transactions, which can make the lookahead pruning technique be performed more efficiently.

Foundations

In practice, mining maximal patterns is much cheaper than mining the complete set of frequent itemsets. However, the worst-case time complexity of maximal pattern mining is the same as mining all frequent patterns. Yang [11] studies the complexity of the maximal pattern mining problem and proves that the problem of counting the number of distinct maximal frequent itemsets in a transaction database, given an arbitrary support threshold, is #P-complete, thereby providing strong theoretical evidence that the problem of mining maximal frequent itemsets is NP-hard.

The concept of maximal patterns has been extended to other similar data mining problems dealing with complex data structures. Yang et al. [12] devise an algorithm that combines statistical sampling and a technique called border collapsing to discover long sequential patterns with sufficiently high confidence in a noisy environment. Xiao et al. [10] propose an algorithm to mine maximal frequent subtrees from a database of unordered labeled trees.

Key Applications

Maximal pattern mining is applicable to dense domains where extracting all frequent patterns is not feasible. It can also be used as a preprocessing step to improve the efficiency of frequent pattern mining and to decide appropriate thresholds for frequent pattern mining and association rule mining.

Experimental Results

Each introduced method has an accompanying experimental evaluation in the corresponding reference. A comprehensive comparison of different algorithms can be found at http://fimi.cs.helsinki.fi/experiments/.

Data Sets

A collection of datasets commonly used for experiments can be found at http://fimi.cs.helsinki.fi/data/.

URL to Code

http://fimi.cs.helsinki.fi/src/.

Cross-references

▶ Closed Itemset Mining and Non-Redundant Association Rule Mining
▶ Frequent Itemsets and Association Rules

Recommended Reading

1. Agarwal R.C., Aggarwal C.C., and Prasad V.V.V. Depth first generation of long patterns. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 108–118.
2. Agrawal R. and Srikant R. Fast algorithms for mining association rules in large databases. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
3. Bayardo R.J. Jr. Efficiently mining long patterns from databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 85–93.
4. Burdick D., Calimlim M., and Gehrke J. Mafia: A maximal frequent itemset algorithm for transactional databases. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 443–452.
5. Gouda K. and Zaki M.J. GenMax: Efficiently mining maximal frequent itemsets. In Proc. 2001 IEEE Int. Conf. on Data Mining, 2001, pp. 163–170.
6. Gunopulos D., Mannila H., and Saluja S. Discovering all most specific sentences by randomized algorithms. In Proc. 6th Int. Conf. on Database Theory, 1997, pp. 215–229.
7. Lin D.I., and Kedem Z.M. Pincer search: A new algorithm for discovering the maximum frequent set. In Advances in Database Technology, Proc. 1st Int. Conf. on Extending Database Technology, 1998, pp. 105–119.
8. Liu G., Lu H., Lou W., Xu Y., and Yu J.X. Efficient mining of frequent patterns using ascending frequency ordered prefix-tree. Data Mining Knowledge Discovery, 9(3):249–274, 2004.
9. Rymon R. Search through systematic set enumeration. In Proc. Int. Conf. on Principles of Knowledge Representation and Reasoning, 1992, pp. 268–275.
10. Xiao Y., Yao J.F., Li Z., and Dunham M.H. Efficient data mining for maximal frequent subtrees. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 379–386.
11. Yang G. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 344–353.
12. Yang J., Wang W., Yu P.S., and Han J. Mining long sequential patterns in a noisy environment. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 406–417.
13. Zaki M.J., Parthasarathy S., Ogihara M., and Li W. New algorithms for fast discovery of association rules. In Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining, 1997, pp. 283–286.

Maybe Answer

▶ Possible Answers

MDIS

▶ Meta Data Interchange Specification

MDR

▶ Metadata Registry, ISO/IEC 11179

MDS

▶ Multidimensional Scaling

Mean Average Precision

▶ MAP

Mean Reciprocal Rank

Nick Craswell
Microsoft Research Cambridge, Cambridge, UK

Synonyms

MRR; Mean Reciprocal Rank of the First Relevant Document; MRR1

Definition

The Reciprocal Rank (RR) information retrieval measure calculates the reciprocal of the rank at which the first relevant document was retrieved. RR is 1 if a relevant document was retrieved at rank 1, if not it is 0.5 if a relevant document was retrieved at rank 2 and so on. When averaged across queries, the measure is called the Mean Reciprocal Rank (MRR).

Key Points

Mean Reciprocal Rank is associated with a user model where the user only wishes to see one relevant document. Assuming that the user will look down the ranking until a relevant document is found, and that document is at rank n, then the precision of the set they view is 1/n, which is also the reciprocal rank measure. For this reason, MRR is equivalent to Mean Average Precision in cases where each query has precisely one relevant document. MRR is not a shallow measure, in that its value changes whenever the required document is moved, although the change is much larger when moving from rank 1 to rank 2 (change is 0.5) compared to moving from rank 100 to 1,000 (change of 0.009).

MRR is an appropriate measure for known item search, where the user is trying to find a document that he either has seen before or knows to exist. This is called *navigational search* in the case of web search. In a case where there are multiple copies of the required document, or otherwise a set of relevant documents that are substitutes, MRR can still be applied based on the first copy.

Cross-references

▶ Mean Average Precision
▶ Precision-Oriented Effectiveness Measures

Mean Reciprocal Rank of the First Relevant Document

▶ MRR (Mean Reciprocal Rank)

Measure

Torben Bach Pedersen
Aalborg University, Aalborg, Denmark

Synonyms

Numerical fact

Definition

A *measure* is a numerical property of a multidimensional *cube*, e.g., sales price, coupled with an aggregation formula, e.g., SUM. It captures numerical information to be used for aggregate computations.

Key Points

As an example, a three-dimensional cube for capturing sales may have a Product *dimension* P, a Time dimension T, and a Store dimension S, capturing the product sold, the time of sale, and the store it was sold in, for each sale, respectively. The cube has two measures: DollarSales and ItemSales, capturing the sales price and the number of items sold, respectively. ItemSales can be viewed as a function: ItemSales: $Dom(P) \times Dom(T) \times Dom(S) \mapsto \mathbb{N}_0$ that given a certain combination of dimension values returns the total number of items sold for that combination. If a dimension value corresponds to a higher level in the dimension *hierarchy*, e.g., a product group or even *all* products, the result is an aggregation of several lower-level measure values.

In a multidimensional database, measures generally represent the properties of the chosen facts that the users want to study, e.g., with the purpose of optimizing them. Measures then take on different values for different combinations of dimension values. The property and formula are chosen such that the value of a measure is meaningful for all combinations of aggregation levels. The formula is defined in the metadata and thus not stored redundantly with the data. Although most multidimensional data models have measures, some do not. In these, dimension values are also used for computations, thus obviating the need for measures, but at the expense of some user-friendliness [2].

It is important to distinguish three classes of measures, namely *additive*, *semi-additive*, and *non-additive* measures, as these behave quite differently in computations. Additive measure values can be combined meaningfully along any dimension. For example, it makes sense to add the total sales over Product, Store, and Time, as this causes no overlap among the real-world phenomena that caused the individual values. Semi-additive measure values cannot be combined along one or more of the dimensions, most often the Time dimension. Semi-additive measures generally occur for so-called "snapshot" facts. For example, it does not make sense to sum inventory levels across time, as the same inventory item, e.g., a specific product item, may be counted several times, but it is meaningful to sum inventory levels across products and stores. Non-additive measure values cannot be combined along any dimension, usually because of the chosen formula. For example, this occurs when averages for lower-level values cannot be combined into averages for higher-level values. The additivity of measures is related to the so-called "type" of the measure (Flow, Stock or Value-Per-Unit).

Cross-references

▶ Cube
▶ Dimension
▶ Hierarchy
▶ Multidimensional Modeling
▶ On-Line Analytical Processing
▶ Summarizability

Recommended Reading

1. Kimball R., Reeves L., Ross M., and Thornthwaite W. The Data Warehouse Lifecycle Toolkit. Wiley Computer Publishing, New York, 1998.
2. Pedersen T.B., Jensen C.S., and Dyreson C.E. A foundation for capturing and querying complex multidimensional data. Inf. Syst., 26(5):383–423, 2001.
3. Thomsen E. OLAP Solutions: Building Multidimensional Information Systems. Wiley, New York, 1997.

Media Recovery

▶ Crash Recovery

Media Semantics

▶ Computational Media Aesthetics

Median

▶ Quantiles on Streams

Mediation

Cesare Pautasso
University of Lugano, Lugano, Switzerland

Synonyms

Transformation; Adaptation; Bridging; Mapping

Definition

Mediation is the process of reconciling differences to reach an agreement between different parties. In databases, the goal of mediation is to compute a common view over multiple, distinct, and heterogeneous sources of data. In software architecture, a component plays the role of mediator if it achieves interoperability by decoupling heterogeneous component having mismatching interfaces. Protocol mediation enables the exchange of information between autonomous endpoints that use incompatible communication protocols.

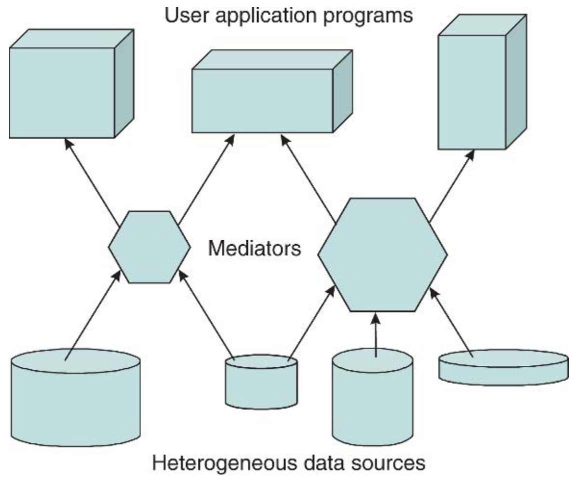
Mediation middleware helps applications deal with heterogeneity. By hiding the multiplicity and the complexity of the underlying systems, it transforms a one-to-many interaction (the application communicating with multiple data sources) into a simpler, one-to-one interaction (the application communicates with the mediator) and shifts the complexity of handling the communication with multiple, heterogeneous parties into a reusable component: the mediator.

Historical Background

In the context of information systems, the concept of mediation has been introduced by Gio Wiederhold in 1992 as the organizing principle for the interoperation of heterogeneous software components and data sources [8]. Mediation is performed by *a layer of intelligent middleware services in information systems, linking data resources and application programs* [9]. Programs need to consume information of multiple data resources and mediators offer them a solution to deal with representation and abstraction problems and exchange objects across multiple, heterogeneous information sources [5]. In the original vision, mediators were seen as independently developed, reusable software modules exploiting expert knowledge about the data to create aggregated information for application programs found at a higher level of abstraction. As shown in Fig. 1, queries from applications are translated by mediators into sub-queries sent out to the different data sources. The various sub-answers are then collected, integrated and returned by the mediator as a single answer to the application.

Foundations

Mediation techniques help to deal with the integration of heterogeneous and incompatible systems. A good understanding of the nature of the problem of dealing with heterogeneity helps to determine when and how mediation should be applied [6]. Conflicts requiring mediation may be found at the level of specific data elements (or messages) to be exchanged or at the level of schema (or interface metadata) definitions. Data values may be stored with multiple representations (syntactic mismatches) or the same representation may be interpreted in different ways (semantic mismatches). Data units (e.g., centimeters versus inches) are also a significant source of conflicts and potential bugs if they are not correctly accounted for when performing data fusion. Identification mismatches are due to the use of locally unique key identifiers that need to be reconciled when tuples from different databases are joined. At the schema level, conflicts involve mismatches in the naming of corresponding elements ("Customer" versus "Client" table); conflicts in the structure ("Street, Number, Zip, City" versus "Address") and granularity ("Purchase Order" versus "Order Line Item") of corresponding data types.



Mediation. Figure 1. Mediation architecture.

It is important to address such mismatches and conflicts at both data and schema levels. To do so, typical mediation middleware tools provide support for operations such as:

1. Selection, filtering, and aggregation of data
2. Data type, encoding and format conversion
3. Join, comparison, and fusion of data originating from multiple data sources
4. Resolution of conflicts between inconsistent sources
5. Multiplexing and demultiplexing over different channels
6. Lookup-based data translation
7. View materialization and caching of intermediate results

Mediator components can be developed using both imperative and declarative programming languages. Automatic schema matching techniques and algorithms that can be applied to support the development of mediators [7].

Mediation can be provided according to two styles:

- Standard-Based Mediation. The mediator transforms each of the incompatible interfaces to comply with a standardized interface that features the least common denominator between all representations. In order for two parties to communicate, this style requires performing two back-to-back transformations (to the standard representation and from the standard representation), thus introduces a larger performance overhead. However, in terms of development cost and maintainability of the mediator, providing support for a new interface only requires introducing one additional transformation in the mediator (assuming that the new interface can be funneled through the standard one).

- Point-to-Point Mediation. The mediator directly maps each pair of incompatible interfaces. This way, the performance overhead is minimized as only one transformation is required as data is exchanged between two parties. Also, there is no need to define a standardized representation, which may be a difficult task in some cases. However, this style should be applied to mediate between a limited (and fixed) set of interfaces only. The complexity of maintaining this kind of mediator does not scale: adding support for a new interface requires to introduce *n* additional mappings in the mediator (one for each existing interface).

Key Applications

Mediation plays a prominent role in applications that require integrating data and functionality originally provided by separate systems (especially in database federation and data integration [10]). For example, a mediator performing currency conversion makes it possible to compare prices of products on sale in different markets. Likewise, mediation is needed to build a country-wide census database out of local databases maintained by different cities, if these have been created independently based on different data models and schemas.

In the context of Service-Oriented Architectures, the role of mediator is associated with the communication bus (or Enterprise Service Bus). Through the bus, services exchange messages without being aware that messages may be transformed while in transit to reconcile differences between service interfaces. Such transformations may be based on context and semantics metadata [4]. Mediation is one of the main features of the Web Services Modeling Framework [1], which applies it to address heterogeneity going beyond traditional data mappings. Also mediators for Business Logic (to compensate between different message ordering constraints found in the public processes of service) and Message Exchange Protocols (for translation between different transport protocols, e.g., to provide reliable and secure message delivery) are introduced in the framework.

In object-oriented design, the *mediator behavioral pattern* has been applied to decouple multiple collaborating objects [3, p. 273]. Instead of having the objects depend on each other and partitioning the interaction logic among them, a mediator object is introduced. It contains and centralizes some potentially complex interaction logic. All objects only refer to the mediator and interact through it. This pattern has also been named *mapper* in the context of enterprise application architecture [2]. Similar to the other applications of mediation, also in this case the interfaces and the overall interactions are simplified thanks to the mediator. However, the price of employing an additional layer of indirection between the elements of a system must be paid.

Cross-references

▶ Enterprise Service Bus
▶ Event Transformation
▶ Schema Mapping
▶ View Adaptation

Recommended Reading

1. Fensel D. and Bussler C. The web service modeling framework WSMF. Electron. Comm. Res. Appl., 1(1):113–137, 2002.
2. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, Reading, MA, November 2002.
3. Gamma E., Helm R., Johnson R., and Vlissides J. Design Patterns: Elements of Reusable Software. Addison-Wesley, Reading, MA, 1995.
4. Mrissa M., Ghedira C., Benslimane D., Maamar Z., Rosenberg F., and Dustdar S. A context-based mediation approach to compose semantic web services. ACM Trans. Internet Technol., 8(1):4, 2007.
5. Papakonstantinou Y., Garcia-Molina H., and Widom J. Object exchange across heterogeneous information sources. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 251–260.
6. Park J. and Ram S. Information systems interoperability: What lies beneath? ACM Trans. Inf. Syst., 22(4):595–632, 2004.
7. Rahm E. and Bernstein P.A. A survey of approaches to automatic schema matching. Int. VLDB J., 10(4):334–350, December 2001.
8. Wiederhold G. Mediators in the architecture of future information systems. Computer, 25(4):38–49, 1992.
9. Wiederhold G. and Genesereth M.R. The conceptual basis for mediation services. IEEE Expert, 12(5):38–47, 1997.
10. Ziegler P. Data Integration Project World-Wide, 2008. http://www.ifi.unizh.ch/~pziegler/IntegrationProjects.html.

Mediation and Adaptation

▶ Database Middleware

Medical Genetics

▶ Implications of Genomics for Clinical Informatics

MEDLINE/ PubMed

▶ Biomedical Scientific Textual Data Types and Processing

Membership Query

Mirella M. Moro
The Federal University of Rio Grande do Sol,
Porte Alegre, Brazil

Synonyms

Equality query; Equality selection

Definition

Consider a relation R whose schema contains some attribute A taking values over a domain D. A *membership query* retrieves all tuples in R with $A = x$ ($x \in D$).

Key Points

A membership query effectively checks membership in a set (relation). As such, it can be implemented using either a hash-based index (built on the attribute(s) involved in the query) or a B+-tree. If a hashing scheme is used, each indexed value is placed on an appropriate hash bucket. Then all records that satisfy $A = x$ are located on the bucket responsible for value x. If A is a numeric attribute (and can thus be indexed using an order-preserving access method like a B+-tree) the membership query is a special case of a range query where the range interval *[low, high]* is reduced to a single value (*low* = *high* = x).

Cross-references

▶ Access Methods
▶ B+-Tree
▶ Hashing

Memory Consistency

▶ Consistency Models For Replicated Data

Memory Hierarchy

Stefan Manegold
CWI, Amsterdam, The Netherlands

Synonyms

Hierarchical memory system

Definition

A Hierarchical Memory System – or Memory Hierarchy for short – is an economical solution to provide computer programs with (virtually) unlimited fast memory, taking advantage of locality and cost-performance of memory technology. Computer storage and memory hardware – from disk drives to DRAM main memory to SRAM CPU caches – shares the limitation that as they became faster, they become more expensive (per capacity), and thus smaller. Consequently, memory hierarchies are organized into several levels, starting from huge and inexpensive but slow disk systems to DRAM main memory and SRAM CPU caches (both off and on chip) to registers in the CPU core. Each level closer to the CPU is faster but smaller than the next level one step down in the hierarchy. Memory hierarchies exploit the principle of locality, i.e., the property that computer programs do not access all their code and data uniformly, but rather focus on referencing only small fractions for given periods of time. Consequently, during each period of time, only the fraction currently referenced – also called hot-set, locality-set, or working-set – needs to be present in the fastest memory level, while the remaining data and code can stay in slower levels. In general, all data in one level is also found in all (slower but larger) memory levels below it.

Historical Background

A three-level memory hierarchy, consisting of (i) the CPU's registers, (ii) DRAM main-memory as primary storage and (iii) secondary storage, has been in use since the introduction of drums and then disk drives as secondary storage. In this memory hierarchy, the decision of which data are loaded into a higher level at what time (and written back in case it is modified) is completely under software control. Application programs determine when to load data from secondary to primary memory, and compilers determine when to load data from main memory into CPU registers.

With the introduction of virtual memory around 1960 [10] application software – and in particular their programmers – are provided with uniformly addressable memory larger than physical memory. The operating system takes care of loading the references portion of data into main memory, automating page transfers, and hence relieving programmers from this task. In the late 1960s, the discovery of the locality principle [7] led to the invention of working sets [6] that exploit locality properties to predict data references, and enabled the design of page replacement algorithms that make virtual memory work efficiently and reliably in multiprogramming environments. Since then, virtual memory has become an inherent feature of operating systems. In contrast to many other application programs, a database management system usually does not rely on the operating system's generic virtual memory management, only. Instead, it implements its own buffer pool, exploiting domain specific knowledge to implement application-specific replacement algorithms.

Until the 1980s, the three-level memory hierarchy has been the state-of-the art, mainly because main memory is considered fast enough to serve the CPU – or better, the CPU were "slow enough." Since then, a continuously growing performance gap develops between CPU and main memory. With the chip integration technology following Moore's Law [12], i.e., doubling the number of transistors per chip area roughly every 1.5 years, the performance has grown exponentially, due to increasing clock-speeds, increasing inherent parallelism, or both. Advanced manufacturing techniques has grown the capacity and – thanks to even wider and faster system buses – their data transfer bandwidth of main memory similarly. Memory access latency, however, has lagged behind, demonstrating at most a slight linear improvement.

To bridge this performance gap, in the 1980's hardware designers started to extend the memory hierarchy by adding small (and expensive) but fast SRAM cache memories between the CPU and main memory. Initially, a single cache level is added, either located on the system board between CPU and main memory, or integrated on the CPU chip. With advancing integration and manufacturing techniques, more levels are added. Nowadays, two to three cache levels integrated on the CPU chip represent the most common configuration.

The main difference between cache memories and the original three levels of the memory hierarchy is that their contents are completely controlled by hardware logic. Relying on the locality principle, carefully turned replacement algorithms (usually variations of LRU), decide when data are loaded into or evicted from which cache level. While ensuring transparent use and easy portability, this approach leaves programs virtually without means to explicitly control the content of CPU caches. In modern systems, software prefetching commands have been introduced to provide programs with limited control to (pre-)load data into caches without actually accessing it.

The scientific computing and algorithm communities – both usually focusing on compute-intensive tasks on memory-based data sets – quickly realized that they have to make the algorithms and data structures *cache-conscious* to exploit the performance potentials of ever faster CPU and CPU caches effectively and efficiently.

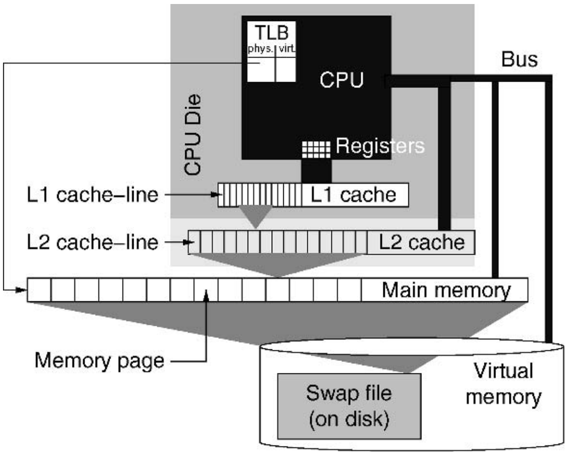
The database world initially ignored the new hardware developments, assuming that optimizing disk access (I/O) is still the key to high performance execution of data-intensive tasks on large disk-based data sets. First proposals of cache-conscious database algorithms occurred in the mid 1990s [14]. It was not until the end of the 20th century that the database community realized that memory access has become a severe bottleneck also for database query processing performance, taking up to 90% of the execution time [4,5]. In the last decade, the development of hardware-aware database technology from system architectures over data structures to query processing algorithms has become a very active and recognized research area [1–3,13].

Foundations

Memory- and Cache-Architectures

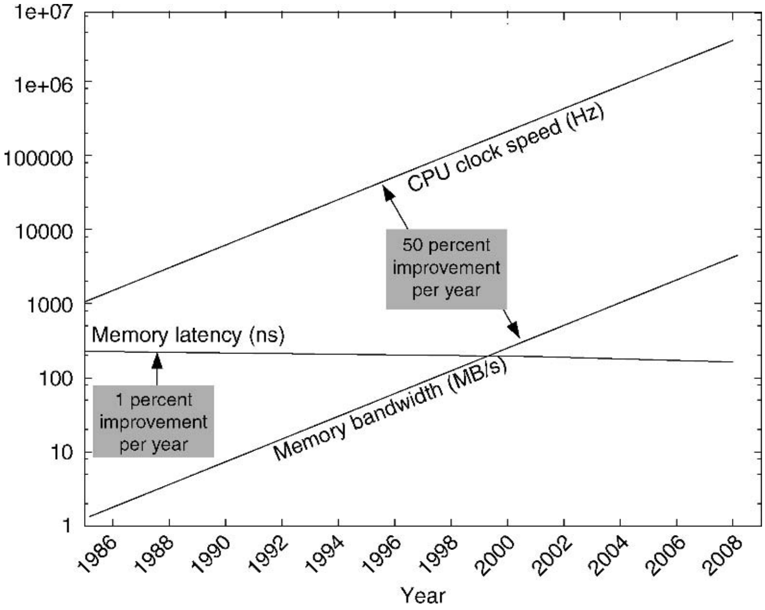
Modern computer architectures have a *hierarchical memory system*, as depicted in Fig. 1. The main memory on the system board consists of *DRAM* (*Dynamic Random Access Memory*) chips. While CPU speeds are increasing rapidly, DRAM access latency has hardly progressed over time. To narrow the exponentially growing performance gap between CPU speed and memory latency (cf., Fig. 2), *cache memories* have been introduced, consisting of fast but expensive *SRAM* (*Static Random Access Memory*) chips. SRAM cells are usually made-up of six transistors per memory bit, and hence, they consume a rather large area on the chips. DRAM cells require a single transistor and a small capacitor to store a single bit. Thus, DRAMs can store much more data than SRAMs of equal (physical) size. But due to some current leakage, the capacitor in DRAMs get discharged over time, and have to be recharged (*refreshed*) periodically to keep their information. These refreshes slow down access.

The fundamental principle of all cache architectures is "*reference locality*," i.e., the assumption that at any time the CPU, thus the program, repeatedly accesses only a limited amount of data (i.e., memory) that fits in the cache. Only the first access is "slow," as the data have to be loaded from main memory. This is called a *compulsory cache miss* (see below). Subsequent accesses (to the same data or memory addresses) are then "fast", as the data are then available in the cache. This is called a *cache hit*. The fraction of memory accesses that can be fulfilled from the cache is called *cache hit rate*; analogously, the fraction of memory accesses that cannot be fulfilled from the cache is called *cache miss rate*.



Memory Hierarchy. Figure 1. Hierarchical memory architecture.

Cache memories are often organized in *multiple cascading levels* between the main memory and the CPU. As they become faster, but smaller, the closer they are to the CPU. Originally, there was one level (typically 64 KB to 512 KB) of cache memory located on the system board. As the chip manufacturing processes improved, a small cache of about 4 KB to 16 KB was integrated on the CPU's die itself, allowing much faster access. The on-board cache is typically not replaced by the on-chip cache, but rather both make up a cache hierarchy, with the one on chip called *first level* (*L1*) cache and the one on board called *second level* (*L2*) cache. Over time, the L2 cache has also been integrated on the CPU's die (e.g., with Intel's Pentium III "Coppermine," or AMD's Athlon "Thunderbird"). On PC systems, the on-board cache has since disappeared, keeping two cache levels. On other platforms, e.g., workstations based on Compaq's (formerly DEC's)



Memory Hierarchy. Figure 2. Trends in CPU and DRAM speed.

Alpha CPU, the on-board cache is kept as *third level* (*L3*) cache, next to the two levels on the die. Most recent multi-core CPUs usually have a private L1 cache of 16 KB to 64 KB per core. The L2 cache is also integrated on the CPU's die. L2 configuration vary between one private L2 per core to one single L2 that is shared among all cores. On Intel's Core 2 Quad, for instance, each of the two L2 caches is shared by two of the four cores. Typical L2 sizes are in the order of 1 MB to 8 MB.

To simplify presentation, the remainder of this entry assumes a typical system with two cache levels (L1 and L2). However, the discussion can easily be generalized to an arbitrary number of cascading cache levels in a straightforward way.

In practice, caches memories do not only cache the data used by an application, but also the program itself, more accurately, the instructions that are currently being executed. With respect to caching, there is one major difference between data and program. Usually, a program must not be modified while it is running, i.e., the caches may be read-only. Data, however, requires caches that also allow modification of the cached data. Therefore, almost all systems nowadays implement two separate L1 caches, a read-only one for instructions and a read-write one for data. The L2 cache, however, is usually a single "unified" read-write cache used for both instructions and data.

Caches are characterized by three major parameters: Capacity (C), Line Size (Z), and Associativity (A):

Capacity (C) A cache's capacity defines its total size in bytes. Typical cache sizes range from 8 KB to 8 MB.

Line Size (Z) Caches are organized in *cache lines*, which represent the smallest unit of transfer between adjacent cache levels. Whenever a cache miss occurs, a complete cache line (i.e., multiple consecutive words) is loaded from the next cache level or from main memory, transferring all bits in the cache line in parallel over a wide bus. This exploits spatial locality, increasing the chances of cache hits for future references to data that are "close to" the reference that caused a cache miss. Typical cache line sizes range from 16 bytes to 256 bytes. Dividing the cache capacity by the cache line size yields the *number of available cache lines* in the cache: $\# = C/Z$.

Associativity (A) To which cache line the memory is loaded depends on the memory address and on the cache's *associativity*. An *A-way set associative* cache allows loading a line in A different positions. If $A > 1$, some *cache replacement* policy chooses one from among the A candidates. *Least Recently Used* (*LRU*) is the most common replacement algorithm. In case $A = 1$, the cache is called *directly-mapped*. This organization causes the least (virtually no) overhead in determining the cache line candidate. However, it also offers the least flexibility and may cause a lot of *conflict misses* (see below). The other extreme case is a *fully associative* cache. Here, each memory address can be loaded to any line in the cache ($A = \#$). This avoids conflict misses, and only *capacity misses* (see below) occur as the cache capacity is exceeded. However, determining the cache line candidate in this strategy causes a relatively high overhead that increases with the cache size. Hence, it is feasible for only smaller caches. Current PCs and workstations typically implement two-way to eight-way set associative caches.

With multiple cache levels, two types are distinguished: inclusive and exclusive caches. With *inclusive caches*, all data stored in L1 is also stored in L2. As data are loaded from memory, they are stored in all cache levels. Whenever a cache line needs to be replaced in L1 (because a mapping conflict occurs or as the capacity is exceeded), its original content can simply be discarded as another copy of that data still remains in the (usually larger) L2. The new content is then loaded from where it is found (either L2 or main memory). The total capacity of an inclusive cache hierarchy is hence determined by the largest level. With *exclusive caches*, all cached data are stored in exactly one cache level. As data are loaded from memory, they get stored only in the L1 cache. When a cache lines needs to be replaced in L1, its original content is first written back to L2. If the new content is then found in L2, it is moved from L2 to L1, otherwise, it is copied from main memory to L1. Compared to inclusive cache hierarchies, exclusive cache hierarchies virtually extend the cache size, as the total capacity becomes the sum of all levels. However, the "swap" of cache lines between adjacent cache levels in case of a cache miss also causes more "traffic" on the bus and hence increases the cache miss latency.

Cache misses can be classified into the following disjoint types [9]:

Compulsory The very first reference to a cache line always causes a cache miss, which is hence classified as a compulsory miss, i.e., an unavoidable miss (even) in an infinite cache. The number of compulsory misses obviously depends only on the data volume and the cache line size.

Capacity A reference that misses in a fully associative cache is classified as a capacity miss because the finite sized cache is unable to hold all the referenced data. Capacity misses can be minimized by increasing the temporal and spatial locality of references in the algorithm. Increasing cache size also reduces the capacity misses because it captures more locality.

Conflict A reference that hits in a fully associative cache but misses in an *A*-way set associative cache is classified as a conflict miss. This is because even though the cache is large enough to hold all the recently accessed data, its associativity constraints force some of the required data out of the cache prematurely. For instance, alternately accessing just two memory addresses that "happen to be" mapped to the same cache line will cause a conflict cache miss with each access. Conflict misses are the hardest to remove because they occur due to address conflicts in the data structure layout and are specific to a cache size and associativity. Data structures would, in general, have to be remapped to minimize conflicting addresses. Increasing the associativity of a cache will decrease the conflict misses.

Coherence Only in case of multi-processor or multi-core systems with private per processor/core high-level caches but shared lower-level caches and/or main memory, the following can occur. If two or more cores access the same data item, it will be loaded in each private cache. If one core than modifies this data item in its private cache, the other copies are invalidated and cannot server futures references. Instead, to ensure cache coherence, a cache miss occurs, and the modified data item has to be loaded from the cache that holds the most up-to-date copy.

Memory Access Costs

In general, memory access costs are characterized by the following three aspects:

Latency Latency is the time span that passes after issuing a data access request until the requested data is available in the CPU. In hierarchical memory systems, the latency increases with the distance from the CPU. Accessing data that are already available in the L1 cache causes *L1 access latency* (λ_{L1}), which is typically rather small (one or two CPU cycles). In case the requested data are not found in L1, an *L1 miss* occurs, additionally delaying the data access by *L2 access latency* (λ_{L2}) for accessing the L2 cache. Analogously, if the data are not yet available in L2, an *L2 miss* occurs, further delaying the access by *memory access latency* (λ_{Mem}) to finally load the data from main memory. Hence, the total latency to access data that are in neither cache is $\lambda_{Mem} + \lambda_{L2} + \lambda_{L1}$. As L1 accesses cannot be avoided, L1 access latency is often assumed to be included in the pure CPU costs, leaving only memory access latency and L2 access latency as explicit memory access costs. As mentioned above, all current hardware actually transfers multiple consecutive words, i.e., a complete cache line, during this time.

When a CPU requests data from a certain memory address, modern DRAM chips supply not only the requested data, but also the data from subsequent addresses. The data are then available without additional address request. This feature is called *Extended Data Output* (EDO). Anticipating sequential memory access, EDO reduces the effective latency. Hence, two types of latency for memory access need to be distinguished. *Sequential access latency* (λ^s) occurs with sequential memory access, exploiting the EDO feature. With random memory access, EDO does not speed up memory access. Thus, *random access latency* (λ^r) is usually higher than sequential access latency.

Bandwidth Bandwidth is a metric for the data volume (in megabytes) that can be transferred between CPU and main memory per second. Bandwidth usually decreases with the distance from the CPU, i.e., between L1 and L2 more data can be transferred per time than between L2 and main memory. The different bandwidths are referred to as *L2 access bandwidth* (β_{L2}) and *memory access bandwidth* (β_{Mem}), respectively. In conventional hardware, the memory bandwidth used to be simply the cache line size divided by the memory latency. Modern multiprocessor systems typically provide excess bandwidth capacity $\beta' \geq \beta$. To exploit this, caches need to be *non-blocking*, i.e., they need to allow more than one outstanding memory load at a time, and the CPU has to be able to issue subsequent load requests while waiting for the first one(s) to be resolved. Further, the access pattern needs to be sequential, in order to exploit the EDO feature as described above.

Indicating its dependency on sequential access, the excess bandwidth is referred to as *sequential access bandwidth* ($\beta^s = \beta'$). The respective *sequential access latency* is defined as $\lambda^s = Z/\beta^s$. For *random access latency* as described above, the respective *random access bandwidth* is defined as $\beta^r = Z/\lambda^r$.

On some architectures, there is a difference between read and write bandwidth, but this difference tends to be small.

Address Translation For data access, logical virtual memory addresses used by application code have to be translated to physical page addresses in the main memory of the computer. In modern CPUs, a *Translation Lookaside Buffer* (*TLB*) is used as a cache for physical page addresses, holding the translation for the most recently used pages (typically 64). If a logical address is found in the TLB, the translation has no additional costs. Otherwise, a *TLB miss* occurs. The more pages an application uses (which also depends on the often configurable size of the memory pages), the higher the probability of TLB misses.

The actual *TLB miss latency* (l_{TLB}) depends on whether a system handles a TLB miss in hardware or in software. With software-handled TLB, TLB miss latency can be up to an order of magnitude larger than with hardware-handled TLB. Hardware-handled TLB fetches the translation from a fixed memory structure that is just filled by the operating system. Software-handled TLB leaves the translation method entirely to the operating system, but requires trapping to a routine in the operating system kernel on each TLB miss. Depending on the implementation and hardware architecture, TLB misses can therefore be more costly than a main-memory access. Moreover, as address translation often requires accessing some memory structure, this can in turn trigger additional memory cache misses.

TLBs can be treated similar to memory caches, using the memory page size as their cache line size, and calculating their (virtual) capacity as *number_of _entries* × *page_size*. TLBs are usually fully associative. Like caches, TLBs can be organized in multiple cascading levels.

For TLBs, there is no difference between sequential and random access latency. Further, bandwidth is irrelevant for TLBs, because a TLB miss does not cause any data transfer.

Unified Hardware Model

Summarizing the above discussion, one can describe a computer's memory hardware as a cascading hierarchy of *N* levels of caches (including TLBs) [11]. An index $i \in \{1,...,N\}$ added to the parameters described above identifies to the respective value of a specific level. The relation between access latency and access bandwidth then becomes $\lambda_{i+1} = Z_i/\beta_{i+1}$. Exploiting the dualism that an access to level $i + 1$ is caused a miss on level i allows some simplification of the notation. Introducing the *miss latency* $l_i = \lambda_{i+1}$ and the respective *miss bandwidth* $b_i = \beta_{i+1}$ yields $l_i = Z_i/b_i$. Each cache level is characterized by the parameters given in Table 1. Costs for L1 cache accesses are assumed to be included in the CPU costs, i.e., λ_1 and β_1 are not used and hence undefined.

Manegold developed a system independent C program called *Calibrator* to measure these parameters on any computer hardware.

Key Applications

In the last decade, the database community has done much research on modifying existing and developing new database technology (system architecture, data

Memory Hierarchy. Table 1. Characteristic parameters per cache level ($i \in \{1,..., N\}$)[2]

Description	Unit	Symbol
Cache name (level)	–	L*i*
Cache capacity	[bytes]	C_i
Cache block size	[bytes]	Z_i
Number of cache lines	–	$\#_i = C_i/Z_i$
Cache associativity	–	A_i
Sequential access		
Access bandwidth	[bytes/ns]	β^S_{i+1}
Access latency	[ns]	$\lambda^s{}_{i+1} = Z_i/\beta^S_{i+1}$
Miss latency	[ns]	$l_i^s = \lambda^S_{i+1}$
Miss bandwidth	[bytes/ns]	$b_i^s = \beta^S_{i+1}$
Random access		
Access latency	[ns]	$\lambda^r{}_{i+1}$
Access bandwidth	[bytes/ns]	$\beta^r{}_{i+1} = Z_i/\lambda^r_{i+1}$
Miss bandwidth	[bytes/ns]	$b_i^r = \beta^r_{i+1}$
Miss latency	[ns]	$l_i^r = \lambda^r_{i+1}$

structures, query processing algorithms) to exploit the characteristics of the extended memory hierarchy efficiently and effectively, improving query evaluation performance up to orders of magnitude.

URL to Code

Manegold's cache-memory and TLB calibration tool *Calibrator* is available at http://homepages.cwi.nl/~manegold/Calibrator/calibrator.shtml

Cross-references

▶ Buffer Management
▶ Buffer Manager
▶ Buffer Pool
▶ Cache-Conscious Query Processing
▶ Locality
▶ Main Memory
▶ Main Memory DBMS
▶ Processor Cache
▶ Secondary Memory

Recommended Reading

1. Ailamaki A., Boncz P.A., and Manegold S. (eds.). Proc. Workshop on Data Management on New Hardware, 2005.
2. Ailamaki A., Boncz P.A., and Manegold S. (eds.). Proc. Workshop on Data Management on New Hardware, 2006.
3. Ailamaki A. and Luo Q. (eds.) Proc. Workshop on Data Management on New Hardware, 2007.
4. Ailamaki A.G., DeWitt D.J., Hill M.D., and Wood D.A. DBMSs on a Modern Processor: Where does time go? In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 266–277.
5. Boncz P.A., Manegold S., and Kersten M.L. Database Architecture Optimized for the New Bottleneck: memory access. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 54–65.
6. Denning P.J. The working set model for program behaviour. Commun. ACM, 11(5):323–333, 1968.
7. Denning P.J. The locality principle. Commun. ACM, 48 (7):19–24, 2005.
8. Hennessy J.L. and Patterson D.A. Computer Architecture – A Quantitative Approach, 3rd edn. Morgan Kaufmann, San Mateo, CA, USA, 2003.
9. Hill M.D. and Smith A.J. Evaluating associativity in CPU caches. IEEE Trans. Comput., 38(12):1612–1630, December 1989.
10. Kilburn T., Edwards D.B.C., Lanigan M.I., and Sumner F.H. One-level storage system. IRE Trans. Electronic Comput., 2(11):223–235, April 1962.
11. Manegold S. Understanding, Modeling, and Improving Main-Memory Database Performance. PhD thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, December 2002.
12. Moore G.E. Cramming more components onto integrated circuits. Electronics, 38(8):114–117, April 1965.
13. Ross K. and Luo Q. (eds.). In Proc. Workshop on Data Management on New Hardware, 2007.
14. Shatdal A., Kant C., and Naughton J. Cache conscious algorithms for relational query processing. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 510–512.

Memory Locality

Stefan Manegold
CWI, Amsterdam, The Netherlands

Synonyms

Principle of locality; Locality principle; Locality of reference

Definition

Locality refers to the phenomenon that computer programs — or computational processes in general — do not access all of their data items uniformly and independently, but rather in a clustered and/or dependent/correlated manner. Some data items are accessed more often than others, repeated accesses to the same data item occur in bursts, and related items are usually accessed together, concurrently or within a short time interval.

There are two types of locality:

1. *Temporal locality* means that accesses to the same data item are grouped in time, i.e., multiple accesses to the same data item occur in rather short time intervals compared to rather long time periods where the same data item is not accessed. Hence, temporal locality is the concept that a data item that is referenced by a program at one point in time will be referenced again sometime in the near future.
2. *Spatial locality* means that data items that are stored physically close to each other tend to be accessed together. Hence spatial locality is the concept that likelihood of referencing a data item by a program is higher if a data item near it has been referenced recently.

Locality belongs to the most fundamental principles of computer science.

Key Point

The discovery of the locality principle dates back to the 1960s. Denning's pioneering work on working-set memory management exploits the locality principle to avoid thrashing of virtual memory systems high levels of multiprogramming [2,3]. Today, this is the key to making virtual memory systems work reliably and efficiently.

In particular, with modern hierarchical memory architectures, exploiting and increasing locality in algorithms and data structures is the key to achieving high performance.

Increased temporal locality ensures that once a data item is referenced, and hence loaded into a fast high-level memory (e.g., cache), all subsequent references occur in short succession while the data item is still available in the cache. Ideally, this data item will not be required, again, once it is evicted from the fast high-level memory.

Data transfer between adjacent levels of hierarchical memory systems does not happen per byte but with larger granularities, e.g., pages of multiple KB or even MB at a time between disk and main memory, cache lines of tens to hundreds of bytes between main memory and CPU cache. *Increased spatial locality* ensures that all data bytes/items that are loaded with each transfer are indeed useful for the program.

Database system architecture exploits and increases locality in numerous ways. Key examples for increased temporal locality are, for instance partitioned join algorithms, where iterating over small partition of the outer relation increases temporal locality of repeated access to the inner relation [5,4]. In fact, smaller partitions of the inner relation also increased spatial locality. Examples of spatial locality range from clustered indices over tuned page layouts such as PAX [1] to the decision between column-stores and row-stores to optimal support of column-major (OLAP) of row-major (OLTP) workloads.

Cross-references

- ▶ Buffer Management
- ▶ Buffer Manager
- ▶ Buffer Pool
- ▶ Cache-Conscious Query Processing
- ▶ Main Memory
- ▶ Main Memory DBMS
- ▶ Memory Hierarchy
- ▶ Processor Cache
- ▶ Secondary Memory

Recommended Reading

1. Ailamaki A.G., DeWitt D.J., Hill M.D., and Skounakis M. Weaving relations for cache performance. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 169–180.
2. Denning P.J. The working set model for program behaviour. Commun. ACM, 11(5):323–333, 1968.
3. Denning P.J. The locality principle. Commun. ACM, 48(7):19–24, 2005.
4. Manegold S., Boncz P.A., and Kersten M.L. Optimizing main-memory join on modern hardware. IEEE Trans. Knowl. Data Eng., 14(4):709–730, July 2002.
5. Shatdal A., Kant C., and Naughton J. Cache conscious algorithms for relational query processing. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 510–512.

Merge Join

▶ Sort-Merge Join

Merge-purge

▶ Deduplication in Data Cleaning
▶ Record Matching

Merkle Hash Trees

▶ Merkle Trees

Merkle Trees

Barbara Carminati
University of Insubria, Varese, Italy

Synonyms

Merkle hash trees; Hash trees; Authentication trees

Definition

Merkle trees are data structures devised to authenticate, with a unique signature, a set of messages, by at the same time making an intended verifier able to verify authenticity of a single message without the disclosure of the other messages. In particular, given a set of messages $M = \{m_1,\ldots,m_n\}$, the Merkle tree created with them is a binary tree whose leaves contain the hash value of each message m in M, whereas internal nodes contain the concatenation of the hash values corresponding to its children.

Key Point

A Merkle tree is a data structure introduced by Merkle in 1979 [1] to improve the Lamport-Diffie one-time signature scheme [2]. In this digital signature scheme, keys can be used to sign, at most, one message. This implies that for each signed message a new public key has to be generated and published. As consequence, Lamport-Diffie one-time digital signature scheme requires publishing a large amount of data. To overcome this drawback, in [1] Merkle proposed a tree-structure, called *authentication tree*, with the aim of authenticating a large number of public keys to be used in one-time signature scheme.

In general, Merkle trees can be exploited to authenticate with a unique signature a set of messages by, at the same, time making an intended verifier able to authenticate a single message without the disclosure of the other messages. Given a set of messages $M = \{m_1,...,m_n\}$, the corresponding Merkle tree is computed by means of the following bottom-up recursive computation: at the beginning, for each different message $m \in M$, a different leaf containing the hash value of m is inserted into the tree; then, for each internal node, the value associated with it is equal to $h(h_l||h_r)$, where $h_l||h_r$ denotes the concatenation of the hash values corresponding to the left and right children nodes, and $h()$ is an hash function. The root node of the resulting binary hash tree is the digest of all the messages, and thus it can be digitally signed by using a standard signature technique. The main benefit of this method is that a user is able to validate the signature by having a subset of messages, providing him/her with a set of additional hash values corresponding to missing messages. Indeed, these additional hash values, together with the provided set of original messages, make a user able to locally re-build the binary tree and, therefore, to validate the signature generated on its root. Consider, for instance, the following set of messages $M = \{m_1, m_2, m_3, m_4\}$. The Merkle tree created with them is a complete binary tree with height of two. More precisely, according to the recursive computation, the root value of the Merkle tree is equal to $h(hr_l||hr_r)$, where hr_l is its left children with value $h(h(m_1)||h(m_2))$, whereas hr_r is its right children with value $h(h(m_3)||h(m_4))$. Assume, now, that a user receives only messages m_1 and m_2. To make him/her able to validate the signature, he/she must be provided also with hash value hr_r. Indeed, by having m_1 and m_2 messages, the user is able to calculate hr_l. Then, using hr_r and hr_l he/she can compute the hash value of the root, and thus verify the signature.

Cross-references

▶ Digital Signatures
▶ Secure Data Outsourcing

Recommended Reading

1. Merkle R. Secrecy, authentication, and public key systems. Electrical Engineering, PhD Thesis, Stanford University, 1979.
2. Lamport L. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, Palo Alto, 1979.

Message Authentication Codes

Marina Blanton
University of Notre Dame, Notre Dame, IN, USA

Synonyms

MAC; Message integrity codes

Definition

A *message authentication code* (MAC) is a short fixed-length value which is used to authenticate a message. A MAC algorithm can be viewed as a hash function that takes as input two functionally distinct values: a secret key and a message. The output of a MAC algorithm is a short string computed in such a way that it is infeasible to produce the same output on the message without the knowledge of the key. Thus, the MAC value protects both the *integrity* and *authenticity* of a message by allowing the entity in possession of the secret key to detect any changes to the message content.

Key Points

While MAC functions can be viewed as keyed cryptographic hash functions, they have specific security requirements for authentication purposes. More precisely, an attacker who does not have access to the secret key and has not seen the MAC value for a specific message before should not be able to compute that value. MAC functions use symmetric techniques (i.e., the same key is used to create and verify a MAC) and thus are different from digital signatures where the signing and verification keys differ. Practical MAC algorithms can be constructed from cryptographic hash functions (for example, HMAC) or from block ciphers (for example, CBC-MAC and others).

Cross-references

- ▶ Authentication
- ▶ Digital Signatures
- ▶ Hash Functions
- ▶ Symmetric Encryption

Recommented Reading

1. Krawczyk H., Bellare M., and Canetti R. HMAC: Keyed-hashing for message authentication, RFC 2104. Internet Engineering Task Force (IETF), 1997.
2. Stallings W. Cryptography and Network Security: Principles and Practices (4th edn.). Pearson-Prentice Hall, Upper Saddle River, NJ, 2006.

Message Integrity Codes

▶ Message Authentication Codes (MAC)

Message Queuing Systems

Sara Bouchenak[1], Noël de Palma[2]
[1]University of Grenoble I — INRIA, Grenoble, France
[2]INPG — INRIA, Grenoble, France

Synonyms

Message-oriented middleware (MOM); Message-oriented systems; Messaging systems; Queuing systems

Definition

A message is an information sent by a sender process to a receiver process. A message queue is a mechanism that allows a sender process and a receiver process to exchange messages. The sender posts a message in the queue, and the receiver retrieves the message from the queue. A message queuing system provides a means to build distributed systems, where distributed processes communicate through messages exchanged via queues.

Key Points

A message queuing system provides several facilities, such as creating messages, creating queues, initializing sender and receiver processes, and providing a means to send and receive messages.

First of all, a message queuing system provides a facility to build a message and fill it with data. Properties may be associated with a message, such as the message size, the message expiration time and the message priority.

A message queuing system also provides facilities to create a queue and, optionally, to associate parameters with a queue, such as the queue length (i.e., the maximum number of messages a queue may hold), the queue topics (i.e., the types of messages the queues may contain), etc.

The senders and receivers of messages may communicate in a synchronous way or in an asynchronous way. With a synchronous communication protocol, a receiver waits for a message from a sender, i.e., it blocks until the message arrives. Whereas with an asynchronous communication protocol, the receiver continues executing and is notified of the reception of a message when this one arrives.

Furthermore, the destination of a message may be specified either explicitly or implicitly. In the explicit mode, the sender specifies the queue to which the message is sent. While in the implicit mode, the sender specifies a topic to which a message is sent, and the message queuing system is responsible of automatically finding the queues that correspond to that topic before sending the message to these queues.

Several message queuing systems are proposed, some are proprietary and others are open source. Oracle proposes Advanced Queuing for Oracle databases [3], Skype has Skytools PgQ for PostgreSQL databases [5], IBM provides WebSphere MQ, Microsoft has MSMQ [1], and Sun Microsystems defines Java Message Service (JMS) as a specification of a Java standard for message queuing systems [6]. Open source message queuing systems include ActiveMQ [7], JBoss Messaging [2], and JORAM [4].

Cross-reference

▶ Adaptive Middleware for Message Queuing Systems

Recommended Reading

1. IBM. WebSphere MQ, 2008. http://www-306.ibm.com/software/integration/wmq/.
2. JBoss. JBoss Messaging, 2008. http://labs.jboss.com/jbossmessaging/.
3. Oracle. Oracle9i Application Developer's Guide – Advanced Queuing, 2008. http://download.oracle.com/docs/cd/B10500_01/appdev.920/a96587/toc.htm.
4. ScalAgent. JORAM: Java Open Reliable Asynchronous Messaging, 2008. http://joram.objectweb.org/.
5. Skype. SkyTools PgQ, 2008. https://developer.skype.com/SkypeGarage/DbProjects/SkyTools.
6. Sun Microsystems. Java Message Service (JMS), 2008. http://java.sun.com/products/jms/.
7. The Apache Software Foundation. Apache ActiveMQ, 2008. http://activemq.apache.org/.

Message-Oriented Middleware (MOM)

▶ Message Queuing Systems
▶ Publish/Subscribe Over Streams

Message-oriented Systems

▶ Message Queuing Systems

Messaging Engines

▶ Interface Engines in Healthcare

Messaging Systems

▶ Message Queuing Systems

Meta Data Base

▶ Meta Data Repository

Metadata Interchange Specification

Wei Tang
Teradata Corporation, El Segundo, CA, USA

Synonyms

MDIS

Definition

Metadata Interchange Specification (MDIS) is a standard proposed by the Metadata Coalition (MDC) for defining metadata.

Key Points

Metadata Coalition (MDC) was an organization of database and data warehouse venders founded in October 1995. Its aim was to define a tactical set of standard specifications for the access and interchange of meta-data between software tools.

In July 1996, Metadata Interchange Specification (MDIS) 1.0 was officially ratified by MDC.

The MDIS Version 1.0 specification represents Coalition member input and recommendations collected and synthesized by the Coalition's technical subcommittee which included representatives from Business Objects, ETI, IBM, Platinum Technology, Price Waterhouse, Prism Solutions, R&O and SAS Institute. The latest version of MDIS is 1.1, which was published in August 1997.

The Metadata Interchange Specification draws a distinction between:

- The Application Metamodel – the tables, etc., used to "hold" the metadata for schemas, etc., for a particular application; for example, the set of tables used to store metadata in Composer may differ significantly from those used by the Bachman Data Analyst.
- The Metadata Metamodel – the set of objects that the MDIS can be used to describe. These represent the information that is common (i.e., represented) by one or more classes of tools, such as data discovery tools, data extraction tools, replication tools, user query tools, database servers, etc. The metadata metamodel should be:
  - Independent of any application metamodel.
  - Character-based so as to be hardware/platform-independent.

- Fully qualified so that the definition of each object is uniquely identified.

There are two basic aspects of the specification:

1. Those that pertain to the semantics and syntax used to represent the metadata to be exchanged. These items are those that are typically found in a specifications document.
2. Those that pertain to some framework in which the specification will be used. This second set of items is two file-based semaphores that are used by the specification's import and export functions to help the user of the specification control consistency.

MDIS consists of a metamodel, which defines the syntax and semantics of the metadata to be exchanged, as well as the specification of a framework for supporting an actual MDIS implementation. The MDIS Metamodel is a hierarchically structured, semantic database model that's defined by a tag language. The metamodel consists of a number of generic, semantic constructs, such as Element, Record, View, Dimension, Level, and Subschema, plus a Relationship entity that can be used in the specification of associations between arbitrary source and target constructs. The MDIS metamodel may be extended through the use of named properties that are understood to be tool-specific and not defined within MDIS. Interchange is accomplished via an ASCII file representation of an instance of this metamodel. Although support for an API is mentioned in the specification, no API definition is provided.

The MDIS Access Framework specifies several fairly general mechanisms that support the interchange of metamodel instances. The Tool and Configuration Profiles define semaphores that ensure consistent, bidirectional metadata exchange between tools. The MDIS Profile defines a number of system parameters (environment variables) that would be necessary in the definition of an MDIS deployment. Finally, Import and Export functions are exposed by the framework as the primary file interchange mechanisms for use by tools.

MDIS 1.1 was planned to be incorporated with Microsoft's Open Information Model (OIM) when Microsoft joined the MDC in December 1998. MDC decided later that MDIS be superseded by OIM. In 2000, the Metadata Coalition merged with the Object Management Group (OMG). OMG has worked on integrating OIM into its Common Warehouse Model (CWM) in order to provide a single standard for modeling meta-data in data warehouses. MDC, MDIS, and OIM are no longer in existence today (as independent entities).

Cross-references

▶ Common Warehouse Metamodel (CWM)
▶ Metadata Coalition (MDC)
▶ Open Information Model (OIM)

Recommented Reading

1. Metadata Interchange Specification (MDIS) Version 1.1. Available at: http://www.eda.org/rassp/documents/atl/MDIS-11.pdf

Meta Data Management System

▶ Meta Data Repository

Meta Data Manager

▶ Meta Data Repository

Meta Data Registry

▶ Meta Data Repository

Meta Data Repository

Christoph Quix
RWTH Aachen University, Aachen, Germany

Synonyms

Meta data base; Meta data manager; Meta data management system; Meta data registry

Definition

A meta data repository (MDR) is a component which manages meta data. In the context of database systems, one example of meta data is information about the database schema, i.e., a description of the data. In addition, MDRs can manage information about the processes which create, use, or update the data, the hardware components that host these processes or the database system, or other (human) resources which make use of the data [6]. As meta data is also data, meta data repositories offer the same functionality for meta data as database management systems (DBMS) for data, e.g., queries, updates, transactions, access control.

Moreover, as meta data is semantically rich data, MDRs often employ object-oriented data models as the basis for the definition of meta data. MDRs should also offer predefined meta models for different types of meta data, so that the user is able to store meta data directly, without defining a meta model in advance.

Another task for a meta data repository is the integration of meta data from various sources into a comprehensive meta data model.

Historical Background

The first components that managed meta data in the context of database systems were dictionary systems which were integrated into the database management systems (DBMS) [12]. These dictionaries were already part of early DBMS products, such as IDMS or IBM IMS. For example, the integrated data dictionary (IDD) of IDMS was a separate database inside the IDMS which was used to maintain meta data of products in the IDMS family [11]. It could be extended also to maintain other types of meta data.

The relational database systems developed in the 1980s also had integrated dictionary systems (also known as system catalogs) to maintain definitions about tables, views, columns, etc. These dictionaries were mainly used by the DBMS itself, but they could be also queried by users and other applications to retrieve information about the contents of a database.

In the 1980s, ANSI started to develop a standard for Information Resource Dictionary Systems (IRDS) which was later adopted by ISO [6]. The standard defined the content, structure, and functionality of an IRDS. The main requirements stated by the IRDS standard are the availability of data modeling facilities, extensibility (i.e., the possibility to add new data types), and the provision of standard DBMS functionality such as query and reporting facilities, integrity and constraint management, and access control.

With the growing need for integrated information systems, stand-alone meta data repository systems became more popular in the 1990s. In contrast to the integrated repositories in DBMS products, a stand-alone MDR was able to manage meta data from different systems. The requirement for meta data integration was especially important for data warehouses, where data that was managed by independent, heterogeneous systems should be integrated into a common data store with a uniformed data model. The availability of meta data of the data sources was a prerequisite for data integration. In this context, some companies tried to build enterprise wide meta data repositories which were supposed to manage all meta data that is available in the enterprise. Such an ambitious goal was hard to achieve, and often, the return-of-investment of such a system was not as high as expected [5]. Therefore, the MDR market was significantly reduced at the end of the 1990s.

Since 2000, two trends for meta data repositories gained importance: community-focused repositories and federated repositories [5]. Communicty-focused repositories are employed in communities (within a company), which share a common interest, such as data warehousing or enterprise application integration. In these communities, the main problem of interoperability of meta data tools could be solved by dedicated bridging technologies, because of the limited scope of the meta data. With the rising importance of service oriented architecture (SOA), MDR needed again to address a broader scope of meta data. Therefore, federated solutions for MDRs are considered to be a solution for the meta data integration problem. In a federated MDR, there are still several MDRs for different communities but federated queries across several MDRs are enabled [5].

Foundations

Requirements

Requirements for MDRs were stated in the IRDS standard [6], in [2], and in [1]:

1. *Dynamic extensibility.* The MDR should provide easy functionalities for the extension of the built-in data models.
2. *Management of objects and relationships.* Objects and relationships between objects should be managed by the MDR.
3. *Notification.* An operation on a specific object in the MDR might trigger other operations on the same or different objects. Therefore, the MDR must be able to notify applications which are interested in certain events. In addition, the invocation

of methods inside the MDR (based on other events) should be also possible.

4. *Version management*. Versioning of meta data is required to track the evolution of a meta data object. It is also important to know which versions of two objects were active at a specific time. It might be also necessary to maintain relationships between older and newer versions of an object.
5. *Configuration management*. A configuration is a set of meta data objects which belong together in respect of content, e.g., they all describe the state of one component. The MDR should be able to manage a configuration as one group. Configurations can be also versioned.
6. *Integrity constraints*. The MDR must provide a language for the definition of integrity constraints on meta data, and enforce the compliance of the meta data with these constraints.
7. *Query and reporting functionality*. To retrieve meta data from the repository, the MDR needs to offer a query language. In addition, user-configurable reports should be also supported.
8. *User access*. If the MDR can be accessed directly by end-users or administrators, the MDR needs to support: a browsing facility for meta data, so that users can navigate through the metadata; an access control, so that users see or update only meta data which they are allowed to; a sophisticated user interface if the users are also allowed to update the meta data, so that the integrity of the MDR is maintained.
9. *Interoperability*. To enable interoperability with other tools and repositories, the MDR should support standards for meta data exchange (such as XMI) and offer an API (application program interface).

Architecture

There are several MDRs already available (see "Systems" section below), each having its own unique architecture. However, by abstracting from these concrete architectures, several components which are common for all MDRs can be identified:

1. *Repository*. The repository component is the internal data store of the MDR and therefore the core of the MDR. As MDRs have to provide similar functionality for meta data as DBMS for data, the repository is often implemented on top of a DBMS.
2. *Meta data manager*. The meta data manager acts as the controller of the repository. As all accesses to the repository should go through the meta data manager, it provides an interface for external applications. Using this interface, applications can store, update, and query meta data.
3. *Models*. A MDR needs to come with already predefined meta models (or information models) which can be directly employed by the users of the MDR to store meta data. If the user has to define its own meta models, the effort to get the MDR running might be too high for the application. Nevertheless, it should be possible to extend the existing models for the specific requirements of the applications that use the MDR.
4. *User interface*. As described above, a MDR can be also accessed by users, which are either end-users using the data or processes described in the MDR, or administrators controlling the system of which the MDR is a part. The user interface can consist of a query facility, a meta data browser, an administrator interface, and an interface to update the meta data.

As mentioned above, a current trend for MDRs is the idea of a federated MDR. This changes the standard architecture described before: the repository component in a federated MDR is not one single data store, the meta data can be distributed across several independent and heterogeneous components. In a federated MDR architecture, the meta data could be stored in files, databases, or managed by specific applications. This increases the complexity of the meta data manager significantly, as meta data queries have to be transformed into queries of the individual systems holding the meta data.

Systems

There are several MDRs available in the market. They can be classified as stand-alone MDRs, repositories integrated into larger software platforms, open source systems and research prototypes.

The market for stand-alone MDRs is changing frequently as companies specialized on MDRs are being acquired by other companies. The current products for separate meta data management solutions are, for example, *ASG Rochade*, *Adaptive Metadata Manager*, and *Advantage Repository*. These products came mainly from the data management area (especially used as MDRs in data warehouse systems), but are now also addressing other areas such as enterprise application integration and service-oriented architectures. Other systems, such as *Logidex* from *LogicLibrary* or *BEA AquaLogic Registry Repository*, have been originally developed as meta data systems for service-oriented architectures.

As mentioned above, large software companies are also addressing the meta data challenges in their software or technology platforms. For example, *IBM* has an integrated MDR in their information integration framework.

There are also a few open source systems which can be used as MDR. Two examples are *Repository in a Box* and *XMDR*. In the research community, *ConceptBase* [9] is a MDR which has been used in several research projects. ConceptBase provides a very flexible data model which can be used for any kind of meta data structure.

Key Applications

There are various application areas for MDRs, basically in all areas in which the management of meta data is necessary. The most important applications for MDRs are situations in which meta data from different sources has to be integrated in one repository. This goes usually beyond the capabilities of builtin MDRs, i.e., repositories which are integrated with other software components.

Data integration in general is an application area in which MDRs play a central role. If data has to be integrated from heterogeneous systems, the description of this data is required to enable the integration. Data warehouse systems [8] are an example for an architecture of integrated data management in which the role of MDRs has been defined explicitely.

In the context of data warehouse systems, also the problem of data quality has been discussed [10,7]. Meta data is often the basis for data quality measurements, e.g., meta data describes the provenance, the age, the semantics of data. Therefore, MDRs are important components for data quality projects.

A MDR can also be used as a resource for structured documentation about IT systems. In addition to the "usual" meta data artefacts such as models and mappings for data integration, also a documentation of the employed systems and their architecture in an organization is useful.

As discussed before, service-oriented architecture (SOA) are becoming an important concept for software development. As a system based on a SOA is a distributed and often heterogeneous system, the management of meta data in a SOA is also important. Therefore, a MDR is often also a component in a SOA.

Another application area for MDRs might the management of meta data on the web, such as RDF or OWL ontologies. However, the web is build on the idea of decentralized data management which is in conflict with the concept of a central, integrated repository for all kind of meta data. Nevertheless, MDRs can be useful to manage the meta data at a specific site, e.g., the ontologies which are offered by that site and their mappings to other ontologies.

Future Directions

The integration of meta data will remain to be a challenge, however the meta data will be integrated, either materialized in a repository, federated with a virtual integration system, or some combination of these. With the rising importance of web applications, service-oriented architectures and similar concepts, it can be expected that more loosely coupled MDRs with a federated integration become more successful. Existing or upcoming meta data standards, such as such as CWM (common warehouse metamodel) and XMI (XML metadata interchange), might simplify the task, but the integration of meta data will remain a problem. Another trend is the integration of MDR into larger software platforms as it is done (or planned) by the major software vendors.

Meta data integration and new architectures for MDRs are also interesting questions for the research community: How can such systems be built, that enable the integrated querying of various meta data sources? Which lessons can be applied for meta data that have already been learned at the data level? Other research questions for MDRs and meta data management are addressed in model management [3,4] which investigates formal methods for working with data models. The challenge for MDRs here is to provide generic structures for the representation of models and mappings.

Cross-references

- ► Data Warehouse Metadata
- ► Meta Data Registry
- ► Meta Model
- ► Meta Object Facility

Recommended Reading

1. Bauer A. and Günzel H. (eds.) Data-Warehouse-Systeme: Architektur, Entwicklung, Anwendung. dpunkt-Verlag, Heidelberg, 2001.
2. Bernstein P.A. Repositories and Object Oriented Databases. ACM SIGMOD Rec., 27(1):88–96, 1998.
3. Bernstein P.A., Halevy A.Y., and Pottinger R. A Vision for Management of Complex Models. ACM SIGMOD Rec., 29(4):55–63, 2000.
4. Bernstein P.A. and Melnik S. Model Management 2.0: Manipulating Richer Mappings. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1-12.
5. Blechar M., IT Metadata Repository Magic Quadrant Update 2002. Gartner, Inc., 2002.
6. ISO/IEC Information technology – Information Resource Dictionary System (IRDS) Framework. International Standard ISO/IEC 10027:1990, DIN Deutsches Institut für Normung, e.V., 1990.
7. Jarke M., Lenzerini M., Vassiliou Y., and Vassiliadis P. (eds.) Fundamentals of Data Warehouses. Springer-Verlag, 2000.
8. Jarke M. and Vassiliou Y. Foundations of Data Warehouse Quality - a Review of the DWQ Project. In Proc. 2nd Int. Conf. Information Quality, 1997, pp. 299–313.
9. Jeusfeld M.A., Jarke M., Nissen H.W., and Staudt M. ConceptBase – Managing Conceptual Models about Information Systems. In Handbook on Architectures of Information Systems, P. Bernus, K. Mertins, and G. Schmidt (eds.). Springer-Verlag, 1998, pp. 265–285.
10. Tayi G.K. and Ballou D.P. Examining Data Quality. Commun. ACM, 41(2):54–57, 1998.
11. Wikipedia – The Free Encyclopedia IDMS (Integrated Database Management System). Article in the encyclopedia, 2008, URL http://en.wikipedia.org/wiki/IDMS.
12. Wikipedia – The Free Encyclopedia. Metadata. Article in the encyclopedia, 2008, URL http://en.wikipedia.org/wiki/Metadata.

Meta Model

► Metamodel

Meta Object Facility

Wei Tang
Teradata Corporation, El Segundo, CA, USA

Synonyms

MOF

Definition

The Meta Object Facility (MOF) is an OMG metamodeling and metadata repository standard. It is an extensible model driven integration framework for defining, manipulating and integrating metadata and data in a platform independent manner. MOF-based standards are in use for integrating tools, applications and data [1].

Key Points

MOF was developed as a response to a request for proposal (RFP), issued by the OMG Analysis and Design Task Force, for Metadata repository facility (http://www.omg.org/cgi-bin/doc?cf/96-05-02). The purpose of the facility was to support the creation, manipulation, and interchange of meta models.

MOF provides a metadata management framework, and a set of metadata services to enable the development and interoperability of model and metadata driven systems. The MOF metadata framework is typically depicted as a four-layer architecture as shown in Table 1:

The MOF specification has three core parts:

1. The specification of the MOF Model
   a. The MOF's built-in meta-metamodel, the "abstract language" for defining MOF metamodels
2. The MOF IDL Mapping
   a. A standard set of templates that map an MOF metamodel onto a corresponding set of CORBA IDL interfaces
3. The MOF's interfaces
   a. The set of IDL interfaces for the CORBA objects that represent an MOF metamodel

The OMG adopted the MOF version 1.0 in November 1997. The most recent revision of MOF, 2.0, was adopted in January 2006 and based on the following OMG specifications:

- MOF 1.4 Specification – MOF 2.0 is a major revision of the MOF 1.4 Specification. MOF 2.0 addresses issues deferred to MOF 2.0 by the MOF 1.4 RTF.

Meta Object Facility. Table 1. OMG's Metadata Architecture

Meta-level	MOF terms	Examples
M3	Meta-metamodel	The "MOF Model"
M2	Metamodel, meta-metadata	UML Metamodel, CWM Metamodel
M1	Model, metadata	UML models, CWM metadata
M0	Object, data	Modeled systems, Warehouse data

- UML 2.0 Infrastructure Convenience Document: ptc/04-10-14 – MOF 2.0 reuses a subset of the UML 2.0 Infrastructure Library packages.
- MOF 2.0 XMI Convenience document: ptc/04-06-11 – Defines the XML mapping requirements for MOF 2.0 and UML 2.0.

The MOF 2 Model is made up of two main packages, Essential MOF (EMOF) and Complete MOF (CMOF).

1. The EMOF Model merges the Basic package from UML2 and merges the Reflection, Identifiers, and Extension capability packages to provide services for discovering, manipulating, identifying, and extending metadata.
2. The CMOF Model is the metamodel used to specify other metamodels such as UML2. It is built from EMOF and the Core:Constructs of UML 2. The Model package does not define any classes of its own. Rather, it merges packages with its extensions that together define basic metamodeling capabilities.

Examples of metadata driven systems that use MOF include modeling and development tools, data warehouse systems, metadata repositories etc. A number of technologies standardized by OMG, including UML, MOF, CWM, SPEM, XMI, and various UML profiles, use MOF and MOF derived technologies (specifically XMI and more recently JMI which are mappings of MOF to XML and Java respectively) for metadata-driven interchange and metadata manipulation. MOF mappings from MOF to W3C XML and XSD are specified in the XMI (ISO/IEC 19503) specification. Mappings from MOF to Java are in the JMI (Java Metadata Interchange) specification defined by the Java Community Process.

Note that MOF 2.0 is closely related to UML 2.0. MOF 2.0 specification integrates and reuses the complementary UML 2.0 Infrastructure submission to provide a more consistent modeling and metadata framework for OMG's Model Driven Architecture. UML 2.0 provides the modeling framework and notation, MOF 2.0 provides the metadata management framework and metadata services.

MOF was also incorporated into an ISO/IEC (the International Organization for Standardization/the International Electrotechnical Commission) standard (19502:2005) in November 2005. The standard defines a metamodel (defined using the MOF), a set of interfaces (defined using ODP IDL – ITU-T Recommendation X.920 (1997) | ISO/IEC 14750:1999), that can be used to define and manipulate a set of interoperable metamodels and their corresponding models (including the Unified Modeling Language metamodel – ISO/IEC 19501:2005, the MOF meta-metamodel, as well as future standard technologies that will be specified using metamodels). It also defines the mapping from MOF to ODP IDL (ITU rec X920|ISO 14750).

In conclusion, the MOF provides the infrastructure for implementing design and reuse repositories, application development tool frameworks, etc. The MOF specifies precise mapping rules that enable the CORBA interfaces for metamodels to be generated automatically, thus encouraging consistency in manipulating metadata in all phases of the distributed application development cycle.

Cross-references

▶ Meta Object Facility
▶ Metadata
▶ Metamodel
▶ Model-Driven Architecture
▶ Unified Modeling Language
▶ XMI

Recommended Reading

1. Common warehouse metamodel (CWM). Available at http://www.omg.org/technology/documents/formal/cwm.htm (accessed on September 22, 2008).
2. ISO/IEC standard 19502:2005 (Information Technology – Meta Object Facility). Available at http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=32621
3. MOF Query/Views/Transformations. Available at http://www.omg.org/spec/QVT/ (current version 1.0)
4. MOF 2.0 versioning and development lifecycle. Available at http://www.omg.org/technology/documents/formal/MOF_version.htm
5. OMG's meta object facility. Available at http://www.omg.org/mof/ (current version 2.0)

Metadata

Manfred A. Jeusfeld
Tilburg University, Tilburg, The Netherlands

Definition

Metadata is data linked to some data item, i.e., metadata is data about data. The metadata of a data item specifies how the data item was created, in which context it can be used, how its was transformed, or how it can be interpreted or processed. The earliest use of metadata are bibliographic records about books, such as the author of the book. In principle, any type of data item can have metadata attached to it. The type of the data item itself can be metadata and determines which other metadata fields may be attached to the data item.

Key Points

The purpose of metadata is to provide contextual information for a data item. It may be used hy humans to determine the usability of a data item. Likewise, computer programs can read the metadata in order to guide the processing of a data item. Metadata can be included in the data item (e.g., the date and location of a photography) or it may be stored apart of the data item. In the latter case, the data item requires being identifiable. In databases, metadata fields can be represented next to data fields, virtually blurring the distinction between metadata and data.

Metadata is mostly used in domains where the structure of the data item is rather complex. Metadata typically has a simple structure such as name/value pairs. Applications domains are word processing, multi-media processing, system design, data warehouses, data quality management, and others. The common characteristic of these domains is the presence of many data items of the same type, which need to be managed according to some criteria. Metadata allows providing the necessary information to check these criteria. In the semantic web, metadata can be represented in RDF and related formalisms such as the Dublin Core.

There is no limitation on the size of metadata attached to data items. It can be that the size of metadata exceeds the size of the data item itself. For example, the complete change history of a document is metadata of the document.

The schema of a database can be interpreted as metadata about the database. It specifies the type of the data items stored in the database. Likewise, a metamodel can be interpreted as metadata about schemas (or models).

Cross-references

▶ Database Schema
▶ Dublin Core
▶ Metamodel
▶ RDF

Recommented Reading

1. Duval E., Hodgins W., Sutton S.A., and Weibel S. Metadata principles and practicalities. D-Lib Magazine, 8(4), April 2002.

Metadata Encoding and Transmission Standard

▶ LOC Mets

Metadata Registry, ISO/IEC 11179

Raymond K. Pon[1], David J. Buttler[2]
[1]University of California, Los Angeles, Los Angeles, CA, USA
[2]Lawrence Livermore National Laboratory, Livermore, CA, USA

Synonyms

Metadata repository; MDR

Definition

ISO/IEC-11179 [10] is an international standard that documents the standardization and registration of metadata to make data understandable and shareable. This standardization and registration allows for easier locating, retrieving, and transmitting data from disparate databases. The standard defines the how metadata are conceptually modeled and how they are shared among parties, but does not define how data is physically represented as bits and bytes. The standard consists of six parts. Part 1 [5] provides a high-level overview of the standard and defines the basic element of a metadata registry – a data element. Part 2 [7] defines the procedures for registering classification schemes and classifying administered items in a metadata registry (MDR). Part 3 [4] specifies the structure of an MDR. Part 4 [6] specifies requirements and recommendations for constructing definitions for data and metadata. Part 5 [8] defines how administered items are named and identified. Part 6 [9] defines how administered items are registered and assigned an identifier.

Historical Background

The first edition of the standard was published by the Technical Committee ISO/IEC JTC1, Information Technology Subcommittee 32, Data Management and Interchange, starting in 1994 and completed in 2000. The second edition was started in 2004 and was completed in 2005. The second edition cancels and replaces the first edition of the standard.

Foundations

Metadata is data that describes other data. A metadata registry is a database of metadata. The database allows for the registration of metadata, which enables the identification, provenance tracking, and quality monitoring of metadata. Identification is accomplished by assigning a unique identifier to each object registered in the registry. Provenance details the source of the metadata and the object described. Monitoring quality ensures that the metadata accomplishes its designed task. An MDR also manages the semantics of data, so that data can be re-used and interchanged. An MDR is organized so that application designers can determine whether a suitable object described in the MDR already exists so that it may be reused instead of developing a new object.

Part 1: Framework

Part 1 introduces the building blocks of the MDR standard: data elements, value domains, data element concepts, conceptual domains, and classification schemes. An MDR is organized as a collection of concepts, which are mental constructs created by a unique combination of characteristics. A concept system is a set of concepts with relations among them. One such concept system that classifies objects is a classification scheme. A classification scheme is organized with some specified structure and is designed for assigning objects to concepts defined within it.

The basic construct in a metadata registry is the data element. A data element consists of a data element concept and a representation. A data element concept (DEC) is a concept that can be represented as a data element described independently of any particular representation. The representation of a data element consists of a value domain, a data-type, units of measure, and a representation class. A data element concept may consist of an object class, which is a set of abstractions in the real world that can be identified with explicit boundaries, and a property, which is a characteristic common to all members of an object class. A value domain is a set of permissible values. Each value domain is a member of the extension of a concept known as the conceptual domain. A conceptual domain is a set of value meanings, which are the associated meanings to values.

An MDR contains metadata describing data constructs. Registering a metadata item makes it a registry item. If the registry item is subject to administration, it is called an administered item. An ISO/IEC 11179 MDR consists of two levels: the conceptual level and the representational level. The conceptual level contains the classes for the data element concept and conceptual domain. The representational level contains classes for data element and value domain.

Part 2: Classification

Part 2 provides a conceptual model for managing concept systems used as classification schemes. Associating an object with a concept from a classification scheme provides additional understanding of the object, comparative information across similar objects, an understanding of an object within the context of a subject matter field, and the ability to identify differences of meaning between similar objects.

Classification schemes are registered in an MDR by recording their attributes, such as those regarding its designation, definition, classification scheme, administration record, reference document, submission, stewardship, registration authority, and registrar.

Part 2 also defines the mechanism for classifying an administered item, which is the assignment of a concept to an object. Objects can also be linked together by relationships linking concepts in the concept system.

Part 3: Registry Metamodel and Basic Attributes

Part 3 describes the basic attributes that are required to describe metadata items and the structure for a metadata registry. The standard uses a metamodel to describe the structure of an MDR. A metamodel is a model that describes other models. The registry metamodel is specified as a conceptual data model, which describes how relevant information is structured in the real world, and is expressed in the Unified Modeling Language [13].

The registry model is divided into six regions:

- **The administration and identification region:** supports the administrative aspects of administered items in the MDR. This region manages the identification and registration of items submitted to the registry, organizations that have submitted and/or are responsible for items in the registry, supporting documentation, and relationships among administered items. An administered item can be a classification scheme, a conceptual domain, context for an administered item, a data element, a data element concept, an object class, a property, a representation class, and a value domain. An administered item is associated with an administration record, which records administrative information about the administered item in the registry.
- **The naming and definition region:** manages the names and definitions of administered items and the contexts for names. Each administered item is named and defined within one or more contexts. A context defines the scope within which the data has meaning, such as a business domain, a subject area, an information system, a data model, or standards document.
- **The classification region:** manages the registration and administration of classification schemes and their constituent classification scheme items. It is also used to classify administered items.
- **The data element concepts region:** maintains information on the concepts upon which the data elements are developed, primarily focusing on semantics.
- **The conceptual and value domain region:** administrates the conceptual domains and value domains.
- **The data element region:** administrates data elements, which provide the formal representations for some information (e.g., a fact, observation, etc.) about an object. Data elements are reusable and shareable representations of data element concepts.

Part 4: Formulation of Data Definitions

Part 4 specifies the requirements and recommendations for constructing data and metadata definitions. A data definition must be stated in the singular. It also must be a descriptive phrase, containing only commonly understood abbreviations, that state what the concept is (as opposed to what the concept is not). A data definition must also be expressed without embedding definitions of other data. The standard also recommends that a data definition should be concise, precise, and unambiguous when stating the essential meaning of the concept. Additionally, a data definition should be self-contained and be expressed without embedding rationale, functional usage, or procedural information, circular reasoning. Terminology and consistent logical structure for related definitions should also be used.

Part 5: Naming and Identification Principles

Part 5 defines the naming and identification of the data element concept, the conceptual domain, data element, and value domain. Each administered item has a unique data identifier within the register of a Registration Authority (RA), which is the organization responsible for an MDR. The international registration data identifier (IRDI) uniquely identifies an administered item globally and consists of a registration authority identifier (RAI), data identifier (DI), and version identifier (VI).

Each administered item has at least one name within a registry of an RA. Each name for an administered item is specified within a context. A naming convention can be used for formulating names. A naming convention may address the scope of the naming convention and the authority that establishes the name. A naming convention may additionally address semantic, syntactic, lexical, and uniqueness rules. Semantic rules govern the existence of the source and content of the terms in a name. Syntactic rules govern the required term order. Lexical rules govern term lists, name length, character set, and language. Uniqueness rules determine whether or not names must be unique.

Part 6: Registration

Part 6 specifies how administered items are registered and assigned an IRDI. Metadata in the MDR is also associated with a registration status, which is a designation of the level of registration or quality of the administered item. There are two types of status categories: lifecycle and documentation. The lifecycle registration status categories address the development and progression of the metadata and the preferences of usage of the administered item. The documentation registration status categories are used when there is no further development in the quality of metadata or use of the administered item.

Each RA establishes its own procedures for the necessary activities of its MDR. Some activities include the submission, progression, harmonization, modification, retirement, and administration of administered items.

Key Applications

The standardization that ISO/IEC 11179 provides enables for the easy sharing of data. For example, many organizations exchange data between computer systems using data integration technologies. In data warehousing schemes, completed transactions must be regularly transferred to separate data warehouses. Exchanges of data can be accomplished more easily if data is defined precisely so that automatic methods can be employed. By having a repository of metadata that describes data, application designers can reuse and share data between computer systems, making the sharing of data easier. ISO/IEC 11179 also simplifies data manipulation by software by enabling the manipulation of data based on characteristics described by the metadata in the registry. This also allows for the development of a data representation model for CASE tools and repositories [3].

There are several organizations that have developed MDRs that comply with ISO/IEC 11179, such as the Australian Institute of Health and Welfare [1], the US Department of Justice [14], the US Environmental Protection Agency [15], the Minnesota Department of Education [11], and the Minnesota Department of Revenue [12]. Currently, there is also an MDR available developed by Data Foundations [2].

Cross-references

▶ Metadata

Recommended Reading

1. Australian Institute of Health and Welfare. Metadata Online Registry (METeOR). http://meteor.aihw.gov.au/, 2007.
2. Data Foundations. Metadata Registry. http://www.datafoundations.com/solutions/data_registries.shtml,2007.
3. ISO/IEC JTC1 SC32. Part 1: Framework for the specification and standardization of data elements. Information Technology – Metadata registries (MDR), 1st edn., 1999.
4. ISO/IEC JTC1 SC32. Part 3: Registry metamodel and basic attributes. Information Technology – Metadata registries (MDR), 2nd edn., 2003.
5. ISO/IEC JTC1 SC32. Part 1: Framework. Information Technology – Metadata registries (MDR), 2nd edn., 2004.
6. ISO/IEC JTC1 SC32. Part 4: Formulation of data definitions. Information Technology – Metadata registries (MDR), 2nd edn., 2004.
7. ISO/IEC JTC1 SC32. Part 2: Classification. Information Technology – Metadata registries (MDR), 2nd edn., 2005.
8. ISO/IEC JTC1 SC32. Part 5: Naming and identification principles. Information Technology – Metadata registries (MDR), 2nd edn., 2005.
9. ISO/IEC JTC1 SC32. Part 6: Registration. Information Technology – Metadata registries (MDR), 2nd edn., 2005.
10. ISO/IEC JTC1 SC32. ISO/IEC 11179, Information Technology – Metadata registries (MDR), 2007.
11. Minnesota Department of Education. Metadata Registry (K-12 Data). 2007. http://education.state.mn.us/mde-dd.
12. Minnesota Department of Revenue. Property Taxation (Real Estate Transactions). 2007. http://proptax.mdor.state.mn.us/mdr.
13. Object Management Group. Unified Modeling Language. 2007. http://www.uml.org/.
14. US Department of Justice. Global Justice XML Data Model (GJXDM). 2007. http://justicexml.gtri.gatech.edu/.
15. US Environmental Protection Agency. Environmental Health Registry. 2007. http://www.epa.gov/edr/.

Metadata Repository

▶ Data Dictionary
▶ Metadata Registry, ISO/IEC 11179

Meta-Knowledge

▶ Multimedia Metadata

Metamodel

Manfred A. Jeusfeld
Tilburg University, Tilburg, The Netherlands

Synonyms

Meta model

Definition

A metamodel is a model that consists of statements about models. Hence, a metamodel is also a model but its universe of discourse is a set of models, namely those models that are of interest to the creator of the metamodel. In the context of information systems, a metamodel contains statements about the constructs used in models about information systems. The statements in a metamodel can define the constructs or can express true and desired properties of the constructs. Like models are abstractions of some reality, metamodels are abstractions of models. The continuation of the abstraction leads to meta metamodels, being models of metamodels containing statements about metamodels. Metamodeling is the activity of designing metamodels (and metametamodels). Metamodeling is applied to design new modeling languages and to extend existing modeling languages.

A second sense of the term metamodel is the specification of the generation of mathematical models, in particular sets of mathematical equations that describe some reality.

Historical Background

One of the earliest metamodels is the definition of the binary data model of Abrial [1]. Abrial distinguished three abstraction levels: the data level of a database, the schema of the database (model), and the category level (metamodel).

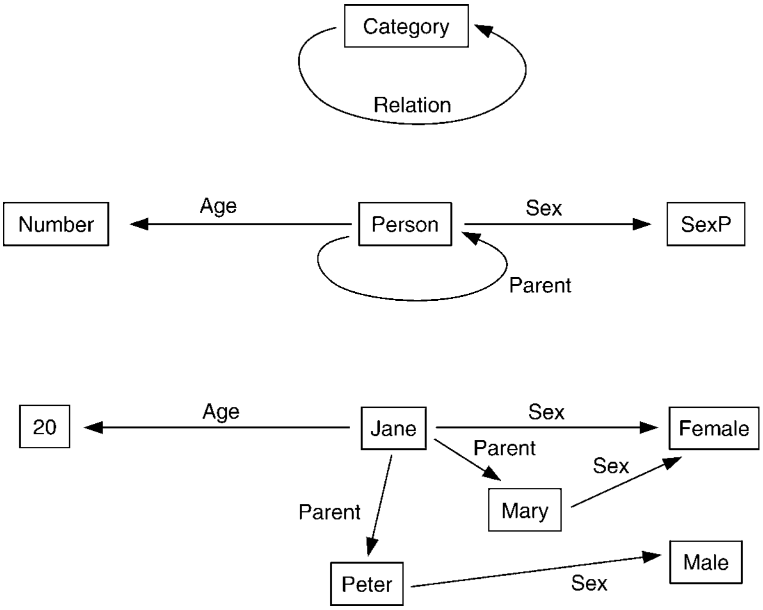
The metamodel defining the binary data model consists of the construct `Category` and the construct `relation`. Abrial interpreted the abstraction between the levels as classification. For example, `Jane` is classified to `Person` and `Person` is classified to `Category`. A similar classification holds for the relations.

In the 1980s, the use of metamodels became so widespread that an ISO standard, the Information Resource Dictionary Standard [2], was defined. It extended Abrial's view by a fourth level, i.e., by metametamodels. In the late 1990s, the Object Management Group (OMG) [5] consolidated and standardized the terminology of metamodels. They distinguished the levels M_0 (information), M_1 (model), M_2 (metamodel), and M_3 (metametamodel). The M_3 level is under control of OMG. It defined four basic constructs (classes, associations, data types, and packages). The M_2 level is used to define modeling languages such as UML, IDL, and so forth.

Besides the standardization efforts, there were several metamodeling languages developed from the 1990s onwards that mostly adopted the four-level approach. Examples is the Telos language and the GOPPR language of MetaEdit+ [9].

Foundations

Meta models in computer science and related domains are mainly used to facilitate conceptual modeling, to define constructs of the conceptual modeling languages, to specify constraints on the use of constructs, and to encode the similarities of different models (and metamodels). As conceptual modeling is about representing concepts, an element of a metamodel is



Metamodel. Figure 1. Abrial's definition of the binary data model.

also a concept say *meta concept*, being interpreted by all entities that are defined or constrained by the meta concept. For example, the meta concept `EntityType` is a construct of the Entity-Relationship Diagramming language. It is interpreted by all possible entity types of all possible entity relationship diagrams. Essentially, `EntityType` is the name of a set that has all possible entity types as elements. A problem with this set-view is that it immediately introduces sets of sets (=concepts in metamodels) and sets of sets of sets (concepts of metametamodels). To overcome this complexity, metamodels were originally only investigated as level-pairs: (metametamodel vs. metamodel), (metamodel vs. model), (model vs. data). First a metametamodel is developed. Then, a metamodel or several metamodels are expressed as instances of the metametamodel, then models are expressed in terms of the metamodels. The lowest level (M_0 in MOF) is typically not expressed in conceptual modeling since it is about data or actual activities of some application domain. The pair-wise approach allows to keep the set-oriented semantics or other forms of semantics specification relying on distinguishing a concept from its instances.

The set-oriented semantics is mirrored by a logical interpretation, in which concepts are represented by unary predicates and relations and attributes are represented by binary predicates. For example, `EntityType (Employee)` is the fact expressing that `Employee` (model level) is classified to `EntityType` (metamodel level). A fact `Employee(Jane)` would then express that `Jane` (data level) is an instance of `Employee`. If one restricts to just two consecutive levels, predicate symbols can be distinguished from constant symbols. In other words, the underlying logic is a first order logic. Scaling the semantics to more than two levels would move the logic to higher order.

One can avoid higher order semantics by introducing a binary predicate `In(x,c)` where `x` is some concept of some modeling level M_i and `c` is a concept of the next higher modeling level M_{i+1}. This framework allows to represent the facts `In(Employee,EntityType)` and `In(Jane,EntityType)` without leaving first order logic.

The specific choice of the underlying semantics for metamodels determines to which degree a metamodel can express the intended meaning of the concepts included in a metamodel. In UML, the semantics of the UML constructs are defined in a metamodel using OCL (object constraint language [6]). Current metamodeling tools dominantly use cardinality constraints as means to constrain the semantics of metamodel concepts. Constraints exceeding cardinalities have to be expressed in OCL or script languages.

The second sense of metamodels, the generation of mathematical equations to describe some reality, is for example used by Bailey and Basili [3] to develop a formal framework for understanding real world phenomena in the domain of software engineering.

Key Applications

Meta models became a popular technique at the end of the 1990s. The current specification of UML is supporting metamodeling in order to extend the capabilities of the language and to adapt it to specific modeling domains. Tools supporting metamodeling are among others MetaEdit+ [9], ConceptBase [4] and Aris [7]. The MetaEdit+ tool claims to accelerate system development by orders of magnitude since the concepts of a metamodel can be linked to parameterized program code.

Future Directions

An open problem of metamodels is their utility. If a metamodel is represented as a UML class diagram, then it does list the allowed constructs but it does not explain how to use it in a meaningful way, i.e., to represent models in terms of the metamodel. Conceptual modeling textbooks motivate constructs by examples and discuss scenarios in which certain constructs are usable. This pragmatic level is neglected by metamodels.

Meta models should be seen as part of the larger model-driven architecture framework. That framework (also defined by OMG) is based on the assumption that system development is essentially a series of model transformations. The design of system development methods is then the combination of metamodeling and the specification of suitable model transformations.

The relationship between metamodels (or metametamodel) with ontologies is not yet well understood. Ontologies rely on two levels of abstraction: the concepts defined in the ontology and the real world objects being the interpretations of the concepts. Apparently, an ontology makes no difference between a model level concept like `Employee` and a metamodel level concept like `EntityType`. See also [8] for a discussion.

Cross-references

► Telos

Recommended Reading

1. Abrial J.R. Data semantics. In Database Management. In Proc. IFIP Working Conf. on Database Management, 1974, pp. 1–60.
2. American National Standard Institute. American National Standard X3.138-1988, Information Resource Dictionary System (IRDS). American National Standard Institute, 1989.
3. Bailey J.W. and Basili V.R. A Meta-model for software development resource expenditures. In Proc. 5th Int. Conf. on Software Eng., 1981, pp. 107–116.
4. Jeusfeld M.A., Jarke M., Nissen H.W., and Staudt M. Managing conceptual models about information systems. In Handbook on Architectures of Information Systems, 2nd edn., P. Bernus, K. Mertins, G. Schmidt (eds.). Springer, Berlin Heidelberg New York, 2006, pp. 273–294.
5. Object Management Group. Meta Object Facility (MOF) Specification, Version 1.4. April 2002. Available at: http://www.omg.org/technology/documents/formal/mof.htm.
6. Object Management Group. Object Constraint Language, OMG Available Specification Version 2.0. May 2006. Available at: http://www.omg.org/cgi-bin/doc?formal/2006-05-01.
7. Scheer A.-W. and Schneider K. ARIS – Architecture of integrated information systems. In Handbook on Architectures of Information Systems, 2nd edn., P. Bernus, K. Mertins, G. Schmidt (eds.).Springer, Berlin Heidelberg New York, 2006, pp. 605–623.
8. Terrasse M.-N., Savonnet M., Leclercq E., Grison T., and Becker G. Do we need metamodels and ontologies for engineering platforms? In Proc. 2006 Int. Workshop on Global Integrated Model Management, 2006, pp. 21–28.
9. Tolvanen J.-P. MetaEdit+: integrated modeling and metamodeling environment for domain-specific languages. In Proc. 21st ACM SIGPLAN Conf. on Object-Oriented Programming Systems, Languages & Applications, 2006, pp. 690–691.

Metaphor

► Visual Metaphor

Metasearch Engines

Weiyi Meng
State University of New York at Binghamton,
Binghamton, NY, USA

Synonyms

Federated search engine

Definition

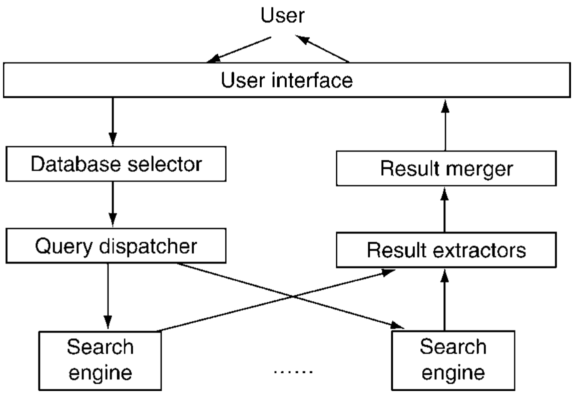
Metasearch is to utilize multiple other search systems (called *component search systems*) to perform simultaneous search. A metasearch engine is a search system that enables metasearch. To perform a basic metasearch, a user query is sent to multiple existing search engines by the metasearch engine; when the search results returned from the search engines are received by the metasearch engine, they are merged into a single ranked list and the merged list is presented to the user. Key issues include how to pass user queries to other search engines, how to identify correct search results from the result pages returned from search engines, and how to merge the results from different search sources. More sophisticated metasearch engines also perform *search engine selection* (also referred to as *database selection*), i.e., identify the search engines that are most appropriate for a query and send the query to only these search engines. To identify appropriate search engines to use for a query requires estimating the usefulness of each search engine with respect to the query based on some usefulness measure.

Historical Background

The earliest Web-based metasearch engine is probably the MetaCrawler system [12] that became operational in June 1995. (The MetaCrawler's website (www.metacrawler.com) says the system was first developed in 1994.) Motivations for metasearch include (i) increased search coverage because a metasearch engine effectively combines the coverage of all component search engines, (ii) improved convenience for users because a metasearch engine allows users to get information from multiple sources with one query submission and the metasearch engine hides the differences in query formats of different search engines from the users, and (iii) better retrieval effectiveness because the result merging component can naturally incorporate the voting mechanism, i.e., results that are highly ranked by multiple search engines are more likely to be relevant than those that are returned by only one of them. Over the last thirteen years, many metasearch engines have been developed and deployed on the Web. Most of them are built on top of a small number of popular general-purpose search engines but there are also metasearch engines that are connected to more specialized search engines (e.g., medical/health search engines) and some are connected to over one thousand search engines.

Even the earliest metasearch engines tackled the issues of search result extraction and result merging. Result merging is one of the most fundamental components in metasearch, and as a result, it has received a lot of attention in the metasearch and distributed information retrieval (DIR) communities and a wide range of solutions has been proposed to achieve effective result merging. Since different search engines may index a different set of web pages and some search engines are better than others for queries in different subject areas, it is important to identify the appropriate search engines for each user query. The importance of search engine selection was realized early in metasearch research and many approaches have been proposed to address this issue. A survey on some earlier result merging and search engine selection techniques can be found in [11].

Most metasearch engines are built on top of other search engines without explicit cooperation from these search engines. As a result, creating these metasearch engines requires a connection program and an extraction program (wrapper) for each component search engine. The former is needed to pass the query from the metasearch engine to the search engine and receive search results returned from the search engine, and the latter is used to extract the search result records from the result pages returned from the search engine. While the programs may not be difficult to produce by an experienced programmer, maintaining their validity can be a serious problem as they can become obsolete when the used search engines change their connection parameters and/or result display format. In addition, for applications that need to connect to hundreds or thousands of search engines, it can be very expensive and time-consuming to produce and maintain these programs. As a result, in recent years, automatic wrapper generation techniques have received much attention. Figure 1 shows a basic architecture of a typical metasearch engine.



Metasearch Engines. Figure 1. Metasearch engine component architecture.

Foundations

Result Merging

Result merging is to combine the search results returned from multiple search engines into a single ranked list. Early search engines often associated a numerical matching score (similarity score) to each retrieved search result and the result merging algorithms at that time were designed to "normalize" the scores returned from different search engines into values within a common range with the goal to make them more comparable. Normalized scores will then be used to re-rank all the search results. When matching scores are not available, the ranks of the search results from component search engines can be aggregated using voting-based techniques (e.g., Borda Count [1]). Score normalization and rank aggregation may also take into consideration the estimated usefulness of each selected search engine with respect to the query, which is obtained during the search engine selection step. For example, the normalized score of a result can be weighted by the usefulness score of the search engine that returned the result. This increases the chance for the results from more useful search engines to be ranked higher.

Another result merging technique is to download all returned documents from their local servers and compute their matching scores using a common similarity function employed by the metasearch engine. The results will then be ranked based on these scores. For example, the Inquirus metasearch engine employs this approach [7]. The advantage of this approach is that it provides a uniform way to compute ranking scores so the resulted ranking makes more sense. Its main drawback is the longer response time due to the delay caused by downloading the documents and analyzing them on the fly. Most modern search engines display the title of each retrieved result together with a short summary called *snippet*. The title and snippet of a result can often provide good clues on whether or not the result is relevant to a query. As a result, result merging algorithms that rely on titles and snippets have been proposed recently (e.g., [9]). When titles and snippets are used to perform the merging, a matching score of each result with the query can be computed based on several factors such as the number of unique query terms that appear in the title/snippet and the proximity of the query terms in the title/snippet.

It is possible that the same result is retrieved from multiple search engines. Such results are more likely to be relevant to the query based on the observation that different ranking algorithms tend to retrieve the same set of relevant results but different sets of irrelevant results [8]. To help rank these results higher in the merged list, the ranking scores of these results from different search engines can be added up to produce the final score for the result. The search results are then ranked in descending order of the final scores.

Search Engine Selection

To enable search engine selection, some information that can represent the contents of the documents of each component search engine needs to be collected first. Such information for a search engine is called the *representative* of the search engine. The representatives of all search engines used by the metasearch engine are collected in advance and are stored with the metasearch engine. During search engine selection for a given query, search engines are ranked based on how well their representatives match with the query. Different search engine selection techniques have been proposed and they often use different types of representatives. A simple representative of a search engine may contain only a few selected key words or a short description. This type of representative is usually produced manually by someone familiar with the contents of the search engine but it can also be automatically generated. As this type of representatives provides only a general description of the contents of search engines, the accuracy of using such representatives for search engine selection is usually low. More elaborate representatives consist of detailed statistical information for each term in each search engine. In [14], the *document frequency* of each term in each search engine is used to compute the *cue validity variance* of each query term, which measures the skew of the distribution of the query term across all component search engines, to help rank search engines for each query. In [3], the *document frequency* and *collection frequency* of each term (the latter is the number of component search engines that contain the term) are used to represent each search engine. In [10], the *adjusted maximum normalized weight* of each term across all documents in a search engine is used to represent a search engine. For a given term *t* and a search engine *S*, the adjusted maximum normalized weight of *t* is computed as follows: compute the normalized weight of *t* in every document (i.e., the term frequency weight of *t* divided by the length of the document) in *S*, find the maximum value among these weights, and multiply this maximum weight by the global *idf* weight of *t* across all component search engines. In [13], the notion of optimal search engine ranking is proposed based on the objective of retrieving the *m* most similar (relevant) documents with respect to a given query Q from across all component search engines: *n* search engines are said to be optimally ranked with order [S_1, S_2,...,S_n] if for any integer *m*, an integer *k* can be found such that the *m* most similar documents are contained in [S_1,...,S_k] and each of these *k* search engines contain at least one of the *m* most similar documents. It is shown in [13] that a necessary and sufficient condition for the component search engines to be optimally ranked is to order the search engines in descending order of the similarity of the most similar document with respect to Q in each search engine. Different techniques have been proposed to estimate the similarity of the most similar document with respect to a given query and a given search engine [10,13]. Since it is impractical to find out all the terms that appear in some pages in a search engine, an approximate vocabulary of terms for a search engine can be used. Such an approximate vocabulary can be obtained from pages retrieved from the search engine using probe queries [2].

There are also techniques that create search engine representatives by learning from the search results of past queries. Essentially such type of representatives is the knowledge indicating the past performance of a search engine with respect to different queries. In the Savvy Search metasearch engine [4], for each component search engine *S*, a weight is maintained for every term that has appeared in previous queries. After each query *Q* is evaluated, the weight of each term in the representative that appears in *Q* is increased or decreased depending on whether or not *S* returns useful results. Over time, if a term for *S* has a large positive (negative) weight, then *S* is considered to have responded well (poorly) to the term in the past. For a new query received by the metasearch engine, the weights of the query terms in the representatives of different search engines are aggregated to rank the search engines. In the ProFusion metasearch engine [5], training queries are used to find out how well each search engine responds to queries in different categories. The knowledge learned about each search engine from training queries is used to select search engines for each user query and the knowledge is continuously updated based on the user's reaction to the search result, i.e., whether or not a particular retrieved result is clicked by the user.

Automatic Search Engine Connection

The search interfaces of most search engines are implemented using the HTML *form* tag with a query textbox. In most cases, the form tag of a search engine contains all information needed to make the connection to the search engine, i.e., sending queries and receiving search results, via a program. Such information includes the name and the location of the program (i.e., the search engine server) that evaluates user queries, the network connection method (i.e., the HTTP request method, usually GET or POST), and the name associated with the query textbox that is used to save the query string. The form tag of each search engine interface is usually pre-processed to extract the information needed for program connection and the extracted information is saved at the metasearch engine. The existence of Javascript in the form tag usually makes extracting the connection information more difficult. After the metasearch engine receives a query and a particular search engine, among possibly other search engines, is selected to evaluate this query, the query is assigned to the name of the query textbox of the search engine and sent to the server of the search engine using the HTTP request method supported by the search engine. After the query is evaluated by the search engine, one or more result pages containing the search results are returned to the metasearch engine for further processing.

Automatic Search Result Extraction

A result page returned by a search engine is a dynamically generated HTML page. In addition to the search result records for a query, a result page usually also contains some unwanted information/links such as advertisements and sponsored links. It is important to correctly extract the search result records on each result page. A typical search result record corresponds to a retrieved document and it usually contains the URL and the title of the page as well as a short summary (snippet) of the document. Since different search engines produce result pages in different format, a separate result extraction program (also called *extraction wrapper*) needs to be generated for each search engine. Automatic wrapper generation for search engines has received a lot of attention in recent years and different techniques have been proposed. Most of them analyze the source HTML files of the result pages as text strings or tag trees (DOM trees) to find the repeating patterns of the search record records. A survey that contains some of the earlier extraction techniques can be found in [6]. Some more recent works also utilize certain visual information on result pages to help identify result patterns (e.g., [15]).

Key Applications

The main application of metasearch is to support search. It can be an effective mechanism to search both surface web and deep web data sources. By providing a common search interface over multiple search engines, metasearch eliminates users' burden to search multiple sources separately. When a metasearch engine employs certain special component search engines, it can support interesting special applications. For example, for a large organization with many branches (e.g., a university system may have many campuses), if each branch has its own search engine, then a metasearch engine connecting to all branch search engines becomes an organization-wide search engine. As another example, if a metasearch engine is created over multiple e-commerce search engines selling the same type of product, then a comparison-shopping system can be created. Of course, for comparison-shopping applications, a different type of result merging is needed, such as listing different search results that correspond to the same product in non-descending order of the prices.

Future Directions

Component search engines employed by a metasearch engine may change their connection parameters and result display format anytime. These changes can make the affected search engines un-usable in the metasearch engine unless the corresponding connection programs and result extraction wrappers are changed accordingly. How to monitor the changes of search engines and make the corresponding changes in the metasearch engine automatically and timely is an area that needs urgent attention from metasearch engine researchers and developers.

Most of today's metasearch engines employ only a small number of general-purpose search engines. Building large-scale metasearch engines using numerous specialized search engines is another area that deserves more attention. The current largest metasearch engine is a news metasearch engine called AllInOneNews (www.allinonenews.com). This metasearch engine currently connects to about 1,800 news search engines. Challenges arising from building very large-scale metasearch engines include automatic generation and maintenance of high quality search engine representatives needed for efficient and effective search engine selection, and highly automated techniques to add search engines into metasearch engines and to adapt to changes of search engines.

Cross-references

▶ Deep-Web Search
▶ Document Length Normalization
▶ Hidden-Web Search
▶ Information Extraction
▶ Information Retrieval
▶ Inverse Document Frequency
▶ Query Routing
▶ Result Integration
▶ Snippet
▶ Term Weighting
▶ Web Data Extraction
▶ Web Data Extraction System
▶ Web Information Retrieval Models
▶ Web Search Engines
▶ Wrapper
▶ Wrapper Generation
▶ Wrapper Induction
▶ Wrapper Maintenance
▶ Wrapper Stability

Recommended Reading

1. Aslam J. and Montague M. Models for metasearch. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 276–284.
2. Callan J., Connell M., and Du A. Automatic discovery of language models for text databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 479–490.
3. Callan J., Lu Z., and Croft W.B. Searching distributed collections with inference networks. In Proc. 18th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 21–28.
4. Dreilinger D. and Howe A. Experiences with selecting search engines using metasearch. ACM Trans. Inf. Syst., 15 (3):195–222, 1997.
5. Fan Y. and Gauch S. Adaptive agents for information gathering from multiple, distributed information sources. In Proc. AAAI Symp. on Intelligent Agents in Cyberspace, 1999, pp. 40–46.
6. Laender A.A., Ribeiro-Neto B., da Silva A., and Teixeira J. A brief survey of web data extraction tools. ACM SIGMOD Rec., 31 (2):84–93, 2002.
7. Lawrence S. and Lee Giles C. Inquirus, the NECi meta search engine. In Proc. 7th Int. World Wide Web Conference, 1998, pp. 95–105.
8. Lee J-H. Combining multiple evidence from different properties of weighting schemes. In Proc. 18th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 180–188.
9. Lu Y., Meng W., Shu L., Yu C., and Liu K. Evaluation of result merging strategies for metasearch engines. In Proc. 6th Int. Conf. on Web Information Systems Eng., 2005, pp. 53–66.
10. Meng W., Wu Z., Yu C., and Li Z. A highly scalable and effective method for metasearch. ACM Trans. Information Syst., 19(3):310–335, 2001.
11. Meng W., Yu C., and Liu K. Building efficient and effective metasearch engines. ACM Comput. Surv., 34(1):48–89, 2002.
12. Selberg E. and Etzioni O. The MetaCrawler architecture for resource aggregation on the web. IEEE Expert, 12(1):11–14, 1997.
13. Yu C., Liu K., Meng W., Wu Z., and Rishe N. A methodology to retrieve text documents from multiple databases. IEEE Trans. Knowledge and Data Eng., 14(6):1347–1361, 2002.
14. Yuwono B. and Lee D. Server ranking for distributed text resource systems on the internet. In Proc. 5th Int. Conf. on Database Systems for Advanced Applications, 1997, pp. 391–400.
15. Zhao H., Meng W., Wu Z., Raghavan V., and Yu C. Fully automatic wrapper generation for search engines. In Proc. 14th Int. World Wide Web Conf., 2005, pp. 66–75.

Metric Space

Pavel Zezula, Michal Batko, Vlastislav Dohnal
Masaryk University, Brno, Czech Republic

Synonyms

Distance space

Definition

In mathematics, a metric space is a pair $M = (D, d)$, where D is a domain of objects (or objects' *keys* or *indexed descriptors*) and d is a total (distance) function. The properties of the function $d : D \times D \mapsto R$, sometimes called the metric space postulates, are typically characterized as:

(p1)	$\forall x, y \in D, d(x, y) \geq 0$	non-negativity,
(p2)	$\forall x, y \in D, d(x, y) = d(y, x)$	symmetry,
(p3)	$\forall x \in D, d(x, x) = 0$	reflexivity,
(p4)	$\forall x, y \in D, x \neq y \Rightarrow d(x, y) > 0$	positiveness,
(p5)	$\forall x, y, z \in D, d(x, z) \leq d(x, y) + d(y, z)$	triangle inequality.

Key Points

Modifying or even abandoning some of the metric function properties leads to interesting concepts that can better suit the reality in many situations. A *pseudo-metric* function does not satisfy the positiveness property (p4), i.e., there can be pairs of different objects that have zero distance. However, these functions can be transformed to the standard metric by regarding any pair of objects with zero distance as a single object. If the symmetry property (p2) does not hold, the function is called a *quasi-metric*. For example, a car-driving distance in a city where one-way streets exist is a quasi-metric. The following equation allows to transform a quasi-metric into a standard metric: $d_{sym}(x, y) = d_{asym}(x, y) + d_{asym}(y, x)$. By tightening the triangle inequality property (p5) to $\forall x, y, z \in D$, $d(x, z) \leq \max\{d(x, y), d(y, z)\}$, an *ultra-metric* also called *super-metric* is obtained. The geometric characterization of the ultra-metric requires every triangle to have at least two sides of equal length, i.e., to be isosceles. A metric space M is bounded if there exists a number r, such that $d(x, y) \leq r$ for any $x, y \in D$. More details about metric functions can be found in [3].

Cross-references

▶ Closest-Pair Query
▶ Indexing Metric Spaces
▶ Information Retrieval
▶ Nearest Neighbor Query
▶ Spatial Indexing Techniques

Recommended Reading

1. Burago D., Burago Y.D., and Ivanov S. A Course in Metric Geometry. American Mathematical Society, Providence, Rhode Island, USA, 2001.
2. Bryant V. Metric Spaces: Iteration and Application. Cambridge University Press, New York, USA, 1985.
3. Zezula P., Amato G., Dohnal V., and Batko M. Similarity Search: The Metric Space Approach, Springer-Verlag, Berlin, 2006.

Microdata

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona, Catalonia

Synonyms

Individual data

Definition

A *microdata* file V with s respondents and t attributes is an $s \times t$ matrix where V_{ij} is the value of attribute j for respondent i. Attributes can be numerical (e.g., age, salary) or categorical (e.g., gender, job).

Key Points

The attributes in a microdata set can be classified in four categories which are not necessarily disjoint [1,2]:

1. *Identifiers*. These are attributes that *unambiguously* identify the respondent. Examples are the passport number, social security number, name-surname, etc.
2. *Quasi-identifiers or key attributes*. These are attributes which identify the respondent with some degree of ambiguity. (Nonetheless, a combination of quasi-identifiers may provide unambiguous identification.) Examples are address, gender, age, telephone number, etc.
3. *Confidential outcome attributes*. These are attributes which contain sensitive information on the respondent. Examples are salary, religion, political affiliation, health condition, etc.
4. *Non-confidential outcome attributes*. Those attributes which do not fall in any of the categories above.

Cross-references

▶ *k*-Anonymity
▶ Data Rank/Swapping
▶ Inference Control in Statistical Databases
▶ Microdata Rounding
▶ Noise Addition
▶ Non-Perturbative Masking Methods
▶ PRAM
▶ SDC Score
▶ Tabular Data

Recommended Reading

1. Dalenius T. The invasion of privacy problem and statistics production: an overview. Statistik Tidskrift, 12:213–225, 1974.
2. Samarati P. Protecting respondents' identities in microdata release. IEEE Trans. on Knowl. and Data Eng., 13(6):1010–1027, 2001.

Microaggregation

JOSEP DOMINGO-FERRER
Universitat Rovira i Virgili, Tarragona, Catalonia

Definition

Microaggregation is a family of masking methods for statistical disclosure control of numerical microdata (although variants for categorical data exist). The rationale behind microaggregation is that confidentiality rules in use allow publication of microdata sets if records correspond to groups of k or more individuals, where no individual dominates (i.e., contributes too much to) the group and k is a threshold value. Strict application of such confidentiality rules leads to replacing individual values with values computed on small aggregates (microaggregates) prior to publication. This is the basic principle of microaggregation.

To obtain microaggregates in a microdata set with n records, these are combined to form g groups of size at least k. For each attribute, the average value over each group is computed and is used to replace each of the original averaged values. Groups are formed using a criterion of maximal similarity. Once the procedure has been completed, the resulting (modified) records can be published.

The optimal k-partition (from the information loss point of view) is defined to be the one that maximizes within-group homogeneity. The higher the within-group homogeneity, the lower the information loss, since microaggregation replaces values in a group by the group centroid. The sum of squares criterion is common to measure homogeneity in clustering. The within-groups sum of squares SSE is defined as

$$SSE = \sum_{i=1}^{g} \sum_{j=1}^{n_i} (x_{ij} - \bar{x}_i)'(x_{ij} - \bar{x}_i)$$

The lower SSE, the higher the within-group homogeneity. Thus, in terms of sums of squares, the optimal k-partition is the one that minimizes SSE.

Key Points

For a microdata set consisting of p attributes, these can be microaggregated together or partitioned into several groups of attributes. Also the way to form groups may vary. Several taxonomies are possible to classify the microaggregation algorithms in the literature: (i) fixed group size vs. variable group size; (ii) exact optimal (only for the univariate case vs. heuristic microaggregation; (iii) continuous vs. categorical microaggregation.

To illustrate, a heuristic algorithm called MDAV (maximum distance to average vector, by Domingo-Ferrer, Mateo-Sanz and Torra) is next given for multivariate fixed group size microaggregation on unprojected continuous data. MDAV has been implemented in the μ-Argus package:

1. Compute the average record $\bar{x}$ of all records in the dataset. Consider the most distant record x_r to the average record $\bar{x}$ (using the squared Euclidean distance).
2. Find the most distant record x_s from the record x_r considered in the previous step.
3. Form two groups around x_r and x_s, respectively. One group contains x_r and the $k - 1$ records closest to x_r. The other group contains x_s and the $k - 1$ records closest to x_s.
4. If there are at least $3k$ records which do not belong to any of the two groups formed in Step 3, go to Step 1 taking as new dataset the previous dataset minus the groups formed in the last instance of Step 3.
5. If there are between $3k - 1$ and $2k$ records which do not belong to any of the two groups formed in Step 3: (i) compute the average record $\bar{x}$ of the remaining records; (ii) find the most distant record x_r from $\bar{x}$; (iii) form a group containing x_r and the $k - 1$ records closest to x_r; (iv) form another group containing the rest of records. Exit the Algorithm.
6. If there are less than $2k$ records which do not belong to the groups formed in Step 3, form a new group with those records and exit the Algorithm.

The above algorithm can be applied independently to each group of attributes resulting from partitioning the set of attributes in the dataset. Microaggregation can be used to achieve k-anonymity.

Cross-references

- ▶ Inference Control in Statistical Databases
- ▶ k-Anonymity

- ▶ Microdata
- ▶ SDC Score

Recommended Reading

1. Domingo-Ferrer J. and Mateo-Sanz J. M. Practical data-oriented microaggregation for statistical disclosure control. IEEE Trans. Knowl. Data Eng., 14(1):189–201, 2002.
2. Domingo-Ferrer J., Sebé F., and Solanas A. A polynomial-time approximation to optimal multivariate microaggregation. Comput. Math. Appl. 55(4):714–732, 2008.
3. Domingo-Ferrer J. and Torra V. Ordinal, continuous and heterogenerous *k*-anonymity through microaggregation. Data Mining Knowl. Dis., 11(2):195–212, 2005.
4. Hundepool A., Van de Wetering A., Ramaswamy R., Franconi L., Capobianchi A., DeWolf P.-P., Domingo-Ferrer J., Torra V., Brand R., and Giessing S. μ-ARGUS Version 4.0 Software and User's Manual. Statistics Netherlands, Voorburg NL, May 2005. http://neon.vb.cbs.nl/casc.

Microbenchmark

Denilson Barbosa[1], Ioana Manolescu[2],
Jeffrey Xu Yu[3]
[1]University of Alberta, Edmonton, AB, Canada
[2]INRIA Saday, Orsay, Cedex, France
[3]The Chinese University of Hong Kong, Hong Kong, China

Definition

A micro-benchmark is an experimental tool that studies a given aspect (e.g., performance, resource consumption) of XML processing tool. The studied aspect is called the target of the micro-benchmark. A micro-benchmark includes a parametric measure and guidelines, explaining which data and/or operation parameters may impact the target, and suggesting value ranges for these parameters.

Key Points

Micro-benchmarks help capture the behavior of an XML processing system on a given operation, as a result of varying one given parameter. In other words, the goal of a micro-benchmark is to study the *precise* effect of a given system feature or aspect *in isolation*.

Micro-benchmarks were first introduced for object-oriented databases [2]. An XML benchmark sharing some micro-benchmark features is the Michigan benchmark [3]. The MemBeR project [1], developed jointly by researchers at INRIA Futurs, the University of Amsterdam, and University of Antwerpen provides a comprehensive repository of micro-benchmarks for XML.

Unlike application benchmarks, micro-benchmarks do not directly help determining which XML processing system is most appropriate for a given task. Rather, they are helpful in assessing particular modules, algorithms and techniques present inside an XML processing tool. Micro-benchmarks are therefore typically very useful to system developers.

Cross-references

- ▶ Application Benchmark
- ▶ XML Benchmarks

Recommended Reading

1. Afanasiev L., Manolescu I., and Michiels P. MemBeR: a micro-benchmark repository for XQuery. In Proc. Database and XML Technologies, 3rd Int. XML Database Symp., 2005, pp. 144–161.
2. Carey M.J., DeWitt D.J., and Naughton J.F. The OO7 Benchmark. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 12–21.
3. Runapongsa K., Patel J.M., Jagadish H.V., Chen Y., and Al-Khalifa S. The Michigan benchmark: towards XML query performance diagnostics. Inf. Syst., 31(2):73–97, 2006.

Microdata Rounding

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona, Catalonia

Synonyms

Rounding

Definition

Microdata rounding is a family of masking methods for statistical disclosure control of numerical microdata; a similar principle can be used to protect tabular data. Rounding replaces original values of attributes with rounded values. For a given attribute X_i, rounded values are chosen among a set of rounding points defining a *rounding set* (often the multiples of a given base value).

Key Points

In a multivariate original dataset, rounding is usually performed one attribute at a time (*univariate* rounding); however, multivariate rounding is also possible [1,2]. The operating principle of rounding makes it suitable for continuous data.

Cross-references

- ▶ Inference Control in Statistical Databases
- ▶ Microdata
- ▶ SDC Score

Recommended Reading

1. Cox L.H. and Kim J.J. Effects of rounding on the quality and confidentiality of statistical data. In J. Domingo-Ferrer and L. Franconi (eds.). Privacy in Statistical Databases-, LNCS, vol. 4302, 2006, pp. 48–56.
2. Willenborg L. and DeWaal T. Elements of Statistical Disclosure Control. Springer-Verlag, New York, 2001.

Middleware Support for Database Replication and Caching

Emmanuel Cecchet
EPFL, Lausanne, Switzerland

Definition

Database replication is a technique that aims at providing higher availability and performance than a single RDBMS. A database replication middleware implements a number of replication algorithms on top of existing RDBMS. Features provided by the replication middleware include load balancing, caching, and fault tolerance.

Historical Background

Database replication is a well-known mechanism for performance scaling and availability of databases across a wide range of requirements. Limitations of 2-phase commit and synchronous replication have been pointed out early on by Gray et al. [7]. Since then, research on middleware-based replication addresses these issues and tries to provide solutions for better performance and availability while maintaining consistency guarantees for applications.

Foundations

Database replication is a wide area of research that encompasses multiple architectures and possible designs. This entry does not address in-core database replication, where the replication algorithms are implemented inside the database engine. Instead, it focuses on middleware-based replication, where the replication logic is implemented in a set of middleware components, outside the database engine.

Shared Disk Versus Shared Nothing

Two main architecture designs can be chosen for database replication. *Shared disk* replication is mostly used by in-core implementations, where replicas share the data storage, such as, a SAN (Storage Area Network). Middleware-based replication usually uses a *shared nothing* architecture, where each replica has its own local storage. This allows disk IOs to be distributed among replicas and prevents the storage from being a Single Point of Failure (SPOF).

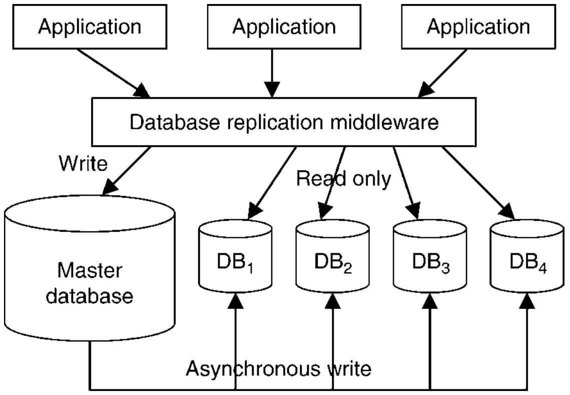
Master/Slave Versus Multi-Master

Database replication is often used as a way to scale up performance. Such efforts are typically targeted at increasing read performance or at increasing write performance; increasing both simultaneously is difficult.

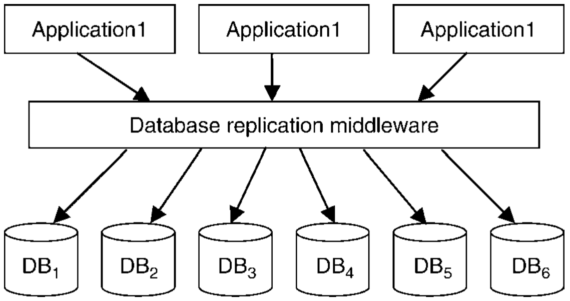
Master-slave replication, depicted in Fig. 1, is popular because it improves read performance. This setup is frequently used in e-commerce applications, with slave databases dedicated to product catalog browsing, while the master performs all catalog updates.

In this scenario, read-only content is accessed on the slave nodes and updates are sent to the master. If the application can tolerate loose consistency, any data can be read at any time from the slaves given a freshness guarantee. As long as the master node can handle all updates, the system can scale linearly simply by adding slave nodes.

Multi-master replication, as shown on Fig. 2, allows each replica that owns a full copy of the database to serve both read and write requests. The replicated system then behaves as a centralized database which theoretically does not require any application modifications. However, replicas need to synchronize to agree on a serializable execution order of transactions so that each replica executes update transactions in the same order. Also, concurrent transactions might



Middleware Support for Database Replication and Caching. Figure 1. Master/slave scale-out scenario.

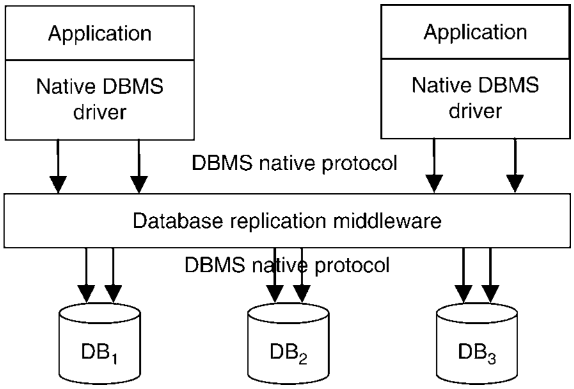


Middleware Support for Database Replication and Caching. Figure 2. Multi-Master database replication.

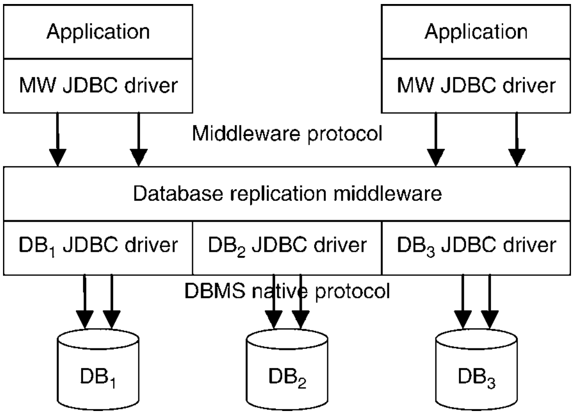
conflict leading to aborts and limiting the system scalability [7]. Even though real applications generally avoid conflicting transactions, significant efforts are spent to optimize for this problem in middleware replication research. However, the volume of update transactions remains the limiting performance factor for such systems.

Middleware Design

The replication middleware has to intercept client requests to process them and run them through the replication algorithm. A first technique consists of keeping existing database drivers and to intercept queries at the database protocol level as show on Fig. 3. This has the significant advantage to not have to re-implement database drivers but the database protocol specification must be available, which is not always the case for commercial databases. Moreover, this design requires multi-protocol implementations and bridges to support heterogeneous clusters.



Middleware Support for Database Replication and Caching. Figure 3. Query interception at the DBMS protocol level.



Middleware Support for Database Replication and Caching. Figure 4. Query interception in JDBC-based replication.

Another technique provides the application with a replacement driver that can communicate with the replication middleware and that is API compatible with the original driver so that application changes are not required. Figure 4 shows an example of a middleware that intercepts queries at the JDBC level. The client application uses the middleware JDBC driver, and the middleware uses the database native JDBC driver to access the replicas. The middleware driver typically adds functionality such as load balancing and failover that is usually not present in standalone database drivers. This is a popular approach introduced by C-JDBC [3] (now Sequoia [9]) and used in other prototypes like Tashkent [5] and Ganymed [8].

Concurrency Control

In replicated database systems, each replica runs Snapshot Isolation (SI) as its local concurrency control and the replicated system provides Generalized Snapshot Isolation (GSI) to the clients.

Snapshot isolation (SI) is a multi-version database concurrency control algorithm for centralized databases. In snapshot isolation, when a transaction begins it receives a logical copy, called snapshot, of the database for the duration of the transaction. This snapshot is the most recent version of the committed state of the database. Once assigned, the snapshot is unaffected by (i.e., isolated from) concurrently running transactions. When an update transaction commits, it produces a new version of the database.

Many database vendors use SI, e.g., PostgreSQL, Oracle and Microsoft SQL Server. SI is weaker than serializability but in practice many applications run serializably under SI including the widely used database benchmarks TPC-C and TPC-W. SI has attractive performance properties. Most notably, read-only transactions do not block or abort-they do not need read-locks, and they do not cause update transactions to block or abort.

Generalized Snapshot Isolation (GSI) extends SI to replicated databases such that the performance properties of SI in a centralized setting are maintained in a replicated setting. In addition, workloads that are serializable under SI are also serializable under GSI.

Informally, a replica using GSI works as follows. When a transaction starts, the replica assigns its latest snapshot to the transaction. All read and write operations of a transaction, e.g., the SELECT, UPDATE, INSERT and DELETE SQL statements, are executed locally on the replica. At commit, the replica extracts the transaction writeset. If the writeset is empty (i.e., it is a read-only transaction), the transaction commits immediately. Otherwise, certification is performed to detect write-write conflicts among update transactions in the system. If no conflict is found, then the transaction commits, otherwise it aborts.

Certification results in total order on the commits of update transactions. Since committing an update transaction creates a new version (snapshot) of the database, the total order defines the sequence of snapshots the database goes through. Therefore, processing update transactions proceeds as follows: When a replica receives update transaction T, it executes T against a snapshot. At commit, the certification service receives the writeset of T and the version of the assigned snapshot. If certification is successful, the replica applies writesets of concurrent update transactions that committed before T in the order determined during certification and then commits T. Certification is a stateful service because it maintains recent committed writesets and their versions.

Statement Replication Versus Transaction Replication

Multi-master replication can be implemented either by multicasting every update statement (statement replication) or by capturing transaction writesets (the set of data *W* updated by a transaction *T*, such that applying *W* onto a replica is equivalent to executing *T* on it) and propagating them after certification (transaction replication). Both approaches have different performance/availability tradeoffs.

Statement-based replication requires that the execution of an update statement produces the same result on each replica. However, many SQL statements may produce different results on every replica if they are not processed before execution. This requires macros related to timing or random numbers to be preprocessed for a deterministic execution cluster-wide. Moreover, stored procedures or user-defined functions must have a deterministic behavior to prevent replicas from diverging in content. The advantage of statement-based replication is that it can replicate any kind of SQL statement including DDL (Data Definition Language) queries that alters the database schema or requests that modify non-persistent objects such as environment variables, sequences or temporary tables.

Transaction replication relies on writeset extraction that is usually implemented using triggers. This requires declaring additional triggers on every database table. This can become complex if the database already uses triggers, materialized views or temporary tables. Writeset extraction does not capture changes such as auto-incremented keys, sequence values or environment variable updates. Queries altering such database structures change the replica they execute on and can result in cluster divergence. Moreover, most of these data structures cannot be rolled back (for instance, an auto-incremented key or sequence number incremented in a transaction is not decremented at rollback time).

With statement replication, all replicas execute write transactions simultaneously in the same serializable order, whereas transaction replication executes update transaction at only one replica and propagates the transaction writeset to other replicas only after certification at commit time. Therefore, transaction replication usually offers better performance over statement replication as long as the writeset extraction and certification mechanisms are efficient. Statement replication offers a better infrastructure for failover during a transaction since each replica has a copy of every transactional context. With transaction replication, the failure of the replica executing the transaction will systematically abort the transaction and force the transaction to retry.

High Availability

High availability is often synonymous with little downtime. Such downtime can be either planned or unplanned, depending on whether it occurs under the control of the administrator or not. Planned downtime is incurred during most software and hardware maintenance operations, while unplanned downtime can strike at any time and is caused by foreseeable and unforeseeable failures (hardware failures, software bugs, human error, etc.).

A system's availability is the ratio of its uptime to total time. In practice, it is computed as the ratio between the expected time of continuous operation between failures to total time, or

$$\begin{aligned} Availability &= \frac{MTTF}{MTTF + MTTR} \Rightarrow Unavailability \\ &= \frac{MTTR}{MTTF + MTTR} \approx \frac{MTTR}{MTTF} \end{aligned}$$

where MTTF is Mean Time To Failure and MTTR is Mean Time To Repair. Since MTTF >> MTTR, one can approximate unavailability (ratio of downtime to total time) as MTTR/MTTF.

The goal of replication together with failover/failback is to reduce MTTR, and thus reduce unavailability. Failover is the ability for users of a database node to be switched over to another database node containing a replica of the data whenever the node they were connected to has failed. Failback happens when the original replica comes back from its failure and users are re-allocated to that replica.

A replicated database built for availability must eliminate any single point of failure (SPOF). This means that the middleware components (load balancer, certifier...) must also be replicated. Group communication libraries are used to synchronize the state of the different components. Total order is usually required by replication protocols to ensure a serializable execution order.

Several database drivers or connection pools offer automatic reconnection when a failure is detected. This technique only offers session failover but not failover of the transactional context. Sequoia [9] (the continuation of the C-JDBC project) provides transparent failover without losing transactional context. Failover code is available in the middleware to handle a database failure and additional code is available in the middleware driver running in the application to handle a middleware failure. A fully transparent failover requires consistently replicated state kept at all components, and is more easily achieved using statement-based rather than transaction-based replication. In the latter case, the transaction is only played at a single replica; if the replica fails, the entire transaction has to be replayed at another replica, which cannot succeed without the cooperation of the application.

Load Balancing

Load balancing aims at dispatching user requests or transactions to the replica that can provide consistent data with the lowest latency. Load balancer design is tightly coupled with the replication strategy implemented. Static strategies such as round-robin or even weighted-round-robin are usually not well adapted to the dynamic nature of transactional workloads. Algorithms taking into account replica resource usage such as LPRF (Least Pending Request First) perform much better. With additional information on transaction working set, it is also possible to optimize load balancing to improve in-memory request execution such as MALB (Memory-Aware Load Balancing) used in Tashkent+ [6]. More information on load balancing can be found in [1].

Caching

To reduce request execution time, the middleware can provide multiple caches. C-JDBC [3] provides three different caches. The parsing cache stores the results of query parsing so that a query that is executed several times is parsed only once. The metadata cache records all ResultSet metadata such as column names and types associated with a query result.

These caches work with query skeletons found in PreparedStatements used by application servers. A query skeleton is a query where all variable fields are replaced with question marks and filled at runtime with a specific API. An example of a query skeleton is "SELECT * FROM t WHERE x=?". In this example, a parsing or metadata cache hit will occur for any value of x.

The query result cache is used to store the ResultSet associated with each query. The query result cache reduces the request response time as well as the load on the database replicas. By default, the cache provides strong consistency. In other words, C-JDBC invalidates cache entries that may contain stale data as a result of an update query. Cache consistency may be relaxed using user-defined rules. The results of queries that can accept stale data can be kept in the cache for a time specified by a staleness limit, even though subsequent update queries may have rendered the cached entry inconsistent.

Different cache invalidation granularities are available ranging from database-wide invalidation to table-based or column-based invalidation. An extra optimization concerns queries that select a unique row based on a primary key. These queries are often issued by application servers using JDO (Java Data Objects) or EJB (Enterprise Java Beans) technologies. These entries are never invalidated on inserts since a newly inserted row will always have a different primary key value and therefore will not affect this kind of cache entries. Moreover, update or delete operations on these entries can be easily performed in the cache.

Key Applications

Middleware-based database replication is currently used in many e-Commerce production environments that require both high availability and performance scalability. Many open problems remain on the integration of databases with replication middleware, failure detection and transparent failover/failback, autonomic management and software upgrades.

Future Directions

Current research trends explore autonomic behavior for replicated databases [4] to automate all management operations such as provisioning, tuning, failure repair and recovery. Heterogeneous clustering is also used in the context of satellites databases [8] to scale legacy databases with open source databases. Partial replication is studied in conjunction with WAN (Wide Area Network) replication for global applications spanning over multiple datacenters distributed on different continents. A summary of the remaining gaps between the theory and practice of middleware-based replication can be found in [2].

URL to Code

The Sequoia source code is available from http://sequoia.continuent.org

Cross-references

- ▶ Autonomous Replication
- ▶ Caching and Replication
- ▶ Consistency Models for Replicated Data
- ▶ 1-Copy-Serializability
- ▶ Data Broadcasting
- ▶ Data Partitioning
- ▶ Data Replication
- ▶ Database Clusters
- ▶ Database Middleware
- ▶ Distributed Database Design
- ▶ Distributed Database Systems
- ▶ Distributed DBMS
- ▶ Distributed Deadlock Management
- ▶ Distributed Query Processing
- ▶ Distributed Recovery
- ▶ Extraction
- ▶ Inter-Query Parallelism
- ▶ Middleware Support for Precise Failure Semantics
- ▶ Partial Replication
- ▶ Replica Control
- ▶ Replica Freshness
- ▶ Replicated Database Concurrency Control
- ▶ Replication
- ▶ Replication Based on Group Communication
- ▶ Replication for High Availability
- ▶ Replication for Scalability
- ▶ Replication in Multi-Tier Architectures Regular
- ▶ Shared-Disk Architecture
- ▶ Shared-Nothing Architecture
- ▶ Snapshot Isolation
- ▶ Strong Consistency Models for Replicated Data
- ▶ Transactional Middleware
- ▶ Transformation and Loading
- ▶ Weak Consistency Models for Replicated Data

Recommended Reading

1. Amza C., Cox A., and Zwaenepoel W. A comparative evaluation of transparent scaling techniques for dynamic content servers. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 230–241.

2. Cecchet E., Candea G., and Ailamaki A. Middleware-based database replication: the gaps between theory and practice. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2008, pp. 739–752.
3. Cecchet E., Marguerite J., and Zwaenepoel W. C-JDBC: flexible database clustering middleware. In Proc. USENIX Annual Technical Conf., 2004.
4. Chen J., Soundararajan G., and Amza C. Autonomic provisioning of backend databases in dynamic content web servers. In Proc. IEEE Int. Conf. Autonomic Computing, 2006, pp. 231–242.
5. Elnikety S., Dropsho S., and Pedone F. Tashkent: uniting durability with transaction ordering for high-performance scalable database replication. In Proc. 1st ACM SIGOPS/EuroSys European Conf. on Comp. Syst., 2006, pp. 117–130.
6. Elnikety S., Dropsho S., and Zwaenepoel W. Tashkent+: memory-aware load balancing and update filtering in replicated databases. In Proc. 2nd ACM SIGOPS/EuroSys European Conf. on Comp. Syst., 2007, pp. 399–412.
7. Gray J.N., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
8. Plattner C., Alonso G., Özsu M.T. Extending DBMSs with satellite databases. VLDB J., 17(4):657–682, 2008.
9. Sequoia project. Available at: http://sequoia.continuent.org

Middleware Support for Precise Failure Semantics

Vivien Quéma
CNRS, INRIA, Saint-Ismier Cedex, France

Definition

Providing support for precise failure semantics requires defining an appropriate correctness criterion for replicated action execution of a replication algorithm. Such a correctness criterion allows formally verifying that a sequence of actions is executed correctly. In the context of replication, a sequence of actions executed is correctly if their side-effect appears to have happened exactly-once.

Historical Background

Reasoning about the behavior of concurrent programs has been an active research area during the past decades. Of particular interest in this area are the works on linearizability, a consistency criterion for concurrent objects [4], and on serializability, a consistency criterion for concurrent transactions [5]. These two criteria facilitate certain kinds of formal reasoning by transforming assertions about complex concurrent behavior into assertions about simpler sequential behavior. Moreover, these consistency criteria are local properties: the correctness of individual objects or services is used to reason about system-level correctness. Recently, Frølund and Guerraoui introduced x-ability [2] (exactly-once ability), a correctness criterion for replicated services. X-ability is independent of particular replication algorithms. Although they facilitate reasoning in similar ways, there are fundamental differences between x-ability on one hand and serializability [5] and linearizability [4] on the other. X-ability has safety as well as liveness aspects to it whereas serializability and linearizability are safety conditions only. X-ability is a theory of distribution and partial failures where serializability and linearizability are theories of concurrency. X-ability does not specify correctness for concurrent invocations of a replicated service. More precisely, x-ability states constraints about the concurrency among replicas in the context of a given request (intra-request concurrency), but ignores the concurrency that originates from different requests (inter-request concurrency). This entry gives a precise description of the x-ability theory.

Foundations

Frølund and Guerraoui proposed x-ability [2] (exactly-once ability), a correctness criterion for replicated services. X-ability is independent of particular replication algorithms. The main idea behind x-ability is to consider a replicated service correct if it provides the illusion of a single, fault-tolerant entity. More precisely, an x-able service must satisfy a contract with its clients as well as a contract with third-party entities. In terms of clients, a service must provide idempotent, non-blocking request processing. Moreover, it must deliver replies that are consistent with its invocation history. The side-effect of a service, on third-party entities, must obey exactly-once semantics. X-ability is a local property: replicated services can be specified and implemented independently, and later composed in the implementation of more complex replicated services.

To model side-effects, x-ability is based on the notion of action execution. Actions are executed correctly (i.e., are *x-able*) if their side-effect *appears* to have happened *exactly-once*. The side-effect of actions can be the modification of a shared state or the invocation of another (replicated or non-replicated) service. X-ability represents the execution of actions as event histories and defines the notion of "appears to have happened exactly-once" in terms of history equivalence: an event history h is x-able if it is equivalent to a history h' obtained under failure-free conditions. Being defined relative to failure-free executions, x-ability encompasses both safety and liveness. It is a safety property because it states that certain partial histories must not occur. It is also a liveness property since it enforces guarantees about what must occur. History equivalence is defined relatively to the execution of two particular kinds of actions, namely *idempotent* and *undoable* actions:

- The side-effect of a history with n incarnations of an idempotent action is equivalent to a history with a single incarnation. For example, writing a particular value to a data object is an idempotent action.
- An undoable action is similar to a transaction: its side-effect can be cancelled up to a certain point (the commit point), after which the side-effect is permanent. Thus, the side-effect of a history with a cancelled action is equivalent to the side-effect of a history with no action at all.

System Model

The replicated service is implemented by a set of replicas. The functionality of the service is captured by a *state machine*. Each replica has its own copy of the state machine. A state machine exports a number of *actions*. An action takes an input value and produces an output value. In addition, an action may modify the internal state of its state machine and it may communicate with external entities. A client can invoke a replica's state machine by sending a request to the replica. A request contains the name of an action and an input value for the action. When it receives a request, a replica invokes its state machine based on the values in the request. If no failures occur, the replica returns the action's output value to the client. The execution of an action may fail or the replica executing the action may fail. If the action fails, it returns an exception (or error) value as the execution result. Formally speaking, action names are modeled as elements of a set `Action` (referred to using the letter a). The set `Value` contains the input and output values associated with actions. Furthermore, two sets, `Request` and `Result`, are defined as follows: `Request`$=$(`Action` $\times$ `Value`) and `Result`$=$`Value`. This signifies that a request is simply a pair that contains an action name and an input value (noted "(a, v)" for a request with action name a and value v).

The actions performed by state machines are represented by *events*. More precisely, the x-ability theory considers two kinds of events: start events to represent the invocation of a state-machine action by a process, and completion events to represent the successful completion of a state-machine action: a process receives a non-exception value back from the state machine. The causal and temporal relationship between action execution and event observation is subject to the following axioms: (i) an action's start event cannot be observed unless the action is invoked, (ii) an action's completion event cannot be observed before its start event, and (iii) if an action returns successfully, then its start and completion events have been observed. Events are modeled as elements of the set `Event`. Events are structured values with the following structure: $e ::= S(a,\ iv) \ \mid \ C(a,\ ov)$. The event $S(a,\ iv)$ captures the start of executing the action a with iv as argument. The event $C(a, ov)$ captures the completion of executing the action a, and ov is the output value produced by the action.

A sequence of events form a *history*. The notion of a sequence captures the total order in which events are observed. Histories are modeled as elements of the set `History`. Histories are structured values as defined by the following syntax: $h \ ::= \ \Lambda \ \mid \ e_1 ... e_n \ \mid h_1 \ \bullet ... \bullet \ h_n$. The symbol Λ denotes the empty history – a history with no events. The history $e_1 ... e_n$ contains the events e_1 through e_n. The history $h_1 \ \bullet ... \bullet \ h_n$ is the concatenation of histories h_1 through h_n. The semantics of concatenating histories is to concatenate the corresponding event sequences. An action a *appears* with input value iv in a history h (noted $(a, iv) \in h$) if h contains a start event produced by the execution of a on iv.

To capture structural properties of histories, the x-ability theory defines the notion of history *patterns*. Formally speaking, patterns are elements of the set `Pattern` (referred to using the letter p). The abstract syntax for patterns is depicted in Fig. 1. A simple pattern sp matches single-action histories. The pattern

$$\begin{aligned} sp \ &::= \ [a,\ iv,\ ov] \mid ?\,[a,\ iv,\ ov] \\ p \ &::= \ sp \mid sp_1 \|_h \, sp_2 \end{aligned}$$

Middleware Support for Precise Failure Semantics. Figure 1. Abstract syntax for history patterns.

$[a, iv, ov]$ matches a history that contains the events from a failure-free execution of an action a. The value iv is the input to a and ov is the output from a. The pattern $?[a, iv, ov]$ matches a history in which a may have failed. A matching history may be the empty history, it may contain a start event only, or it may contain both the start and completion event of a. The pattern $sp_1 \parallel h\ sp_2$ matches a history h' that contains an interleaving of three sub-histories h_1, h_2, and h, where h_1 matches sp_1, h_2 matches sp_2, and h is an arbitrary history. The interleaving is constrained as follows: the first event in h_1 must also be the first event in h' and the last event in h_2 must also be the last event in h'.

X-ability defines *pattern matching* as a relation $\triangleright$ between elements of the set `History` and elements of the set `Pattern`. In other words, $\triangleright$ is a subset of `History` $\times$ `Pattern` (the set of all pairs from `History` and `Pattern`). Pattern matching rules are shown in Fig.2. A history that matches a simple pattern contains at most two events. X-ability defines two operators on such histories: `first` and `second` (see Fig.3). The `first` operator returns the first element in a history, if any, and Λ otherwise. The `second` operator returns the second element in a history of length two, the only element in a history of length one, and the empty history otherwise.

X-Able Histories

To be fault-tolerant, a replicated service must be prepared to invoke the same action multiple times until the action executes successfully. To provide replication transparency, the service must have exactly-once semantics relative to its environment – the service must maintain the illusion that the action was executed once only. In short, an x-able history is a history that maintains the illusion of exactly-once but possibly contains multiple incarnations of the same action. The rest of this section describes how the x-ability theory defines the notion of x-able history.

The x-ability theory defines a *history reduction* relation, $\Rightarrow$, on histories as follows: if $h \Rightarrow h'$, then the execution that produced h has the same side-effect as an execution that produced h'. Essentially, a history is x-able if it can be reduced, under $\Rightarrow$, to a history that could arise from a system that does not fail. Two particular types of actions are considered: `idempotent` and `undoable`. The corresponding sets are called Idempotent and Undoable. The set

$$(1)\qquad \triangleright \subseteq (\mathtt{History} \times \mathtt{Pattern})$$

$$(2)\qquad S(a, iv)C(a, ov) \triangleright [a, iv, ov]$$

$$(3)\qquad \Lambda \triangleright ?[a, iv, ov]$$

$$(4)\qquad S(a, iv) \triangleright ?[a, iv, ov]$$

$$(5)\qquad S(a, iv)C(a, ov) \triangleright ?[a, iv, ov]$$

$$(6)\qquad \frac{h_1 \triangleright sp_1 \quad h_2 \triangleright sp_2}{(h_1 \bullet h \bullet h_2) \triangleright (sp_1 \|_h sp_2)}$$

$$(7)\qquad \frac{h_1 \triangleright sp_1 \quad h_2 \triangleright sp_2}{(\mathrm{first}(h_1) \bullet h_3 \bullet \mathrm{second}(h_1) \bullet h_4 \bullet \mathrm{first}(h_2) \bullet h_5 \bullet \mathrm{second}(h_2)) \triangleright (sp_1 \|_{h_3 \bullet h_4 \bullet h_5}\ sp_2)}$$

$$(8)\qquad \frac{h_1 \triangleright sp_1 \quad h_2 \triangleright sp_2}{(\mathrm{first}(h_1) \bullet h_3 \bullet \mathrm{first}(h_2) \bullet h_4 \bullet \mathrm{second}(h_1) \bullet h_5 \bullet \mathrm{second}(h_2)) \triangleright (sp_1 \|_{h_3 \bullet h_4 \bullet h_5}\ sp_2)}$$

Middleware Support for Precise Failure Semantics. Figure 2. Pattern matching rules.

$$\begin{array}{rrcl@{\qquad}rcl}
(9) & \mathrm{first}(\Lambda) & = & \Lambda & \mathrm{first}(e_1 e_2) & = & e_1 \\
(10) & \mathrm{first}(e) & = & e & \mathrm{second}(\Lambda) & = & \Lambda \\
(11) & \mathrm{second}(e) & = & e & \mathrm{second}(e_1 e_2) & = & e_2
\end{array}$$

Middleware Support for Precise Failure Semantics. Figure 3. The definition of first and second.

`Idempotent` contains the names of idempotent actions. The notation a^i indicates that the action a is idempotent. The set `Undoable` contains names of undoable actions. The notation a^u indicates that an action a is undoable. An undoable action, a^u, has two associated actions: a cancellation action, a^{-1}, and a commit action, a^c. The commit and cancellation actions for an action a^u take the same arguments as a^u, and they return the value nil. Moreover, cancellation and commit actions are idempotent.

Figure 4 defines the⇒operator in terms of idempotent and undoable actions. The first inference rule (13) defines ⇒ as a transitive relation. The second rule (14) captures the semantics of idempotent actions. If a history contains a successfully executed idempotent action a^i, then the events from a previous attempt to execute a^i can be removed. The third rule (15) is concerned with cancellation of undoable actions. Intuitively, if an undoable action is successfully cancelled, then its side-effect can be removed. The fourth rule (16) states that commit actions are idempotent.

The x-ability theory defines a *failure-free history* as a history that could have been produced by a failure-free execution of a single state-machine action. To define the notion of failure-free history, the x-ability theory relies on the definition of a function, called eventsof, which returns the failure-free history associated with an action and its values.

$$\begin{aligned}\texttt{eventsof}(a^u,\ iv,\ ov)\ &= S(a^u, iv)C(a^u, ov)\\ &\quad S(a^c,\ iv)C(a^c, \text{nil})\end{aligned} \tag{17}$$

$$\texttt{eventsof}(a^i)\ =\ S(a^i, iv)C(a^i, ov) \tag{18}$$

Due to non-determinism, there are multiple failure-free histories which are possible for a given action a and a given input value iv. The set of all possible histories, $\texttt{FailureFree}_{(a,iv)}$, is defined as follows:

$$\begin{aligned}\texttt{FailureFree}_{(a,iv)}\ &= \{h \in \texttt{History}\,|\,\exists\, ov \in\\ &\qquad \texttt{Result} : \texttt{h} = \texttt{eventsof}\\ &\qquad (a, iv, ov)\}\end{aligned} \tag{19}$$

An *x-able history* is defined as a history that can be reduced to a failure-free history. Formally speaking, an x-able history is one that satisfies the predicate x-able on histories:

$$\begin{aligned}&\texttt{X-4able}_{(a,iv)}(h)\ =\\ &\begin{cases}\texttt{true} & \text{if } \exists\, h' \in \texttt{FailureFree}_{(a,\,iv)} : h \Rightarrow h'\\ \texttt{false} & \text{otherwise}\end{cases}\end{aligned} \tag{20}$$

This definition of x-able histories applies to single-action histories, that is, a history that arises from a particular request. This reflects the fact that x-ability only specifies correctness relative to distribution and failures, it does not specify correctness for the concurrent processing of multiple requests from different clients.

Client-Service Consistency

The x-ability theory formalizes the relationship between clients and services. More precisely, the reply value given to a client in response to a request must be the value returned from the server-side state machine when the service processes the request. Moreover, the service is not allowed to invent requests. The server-side history is used to define the constraints for requests and replies. This history contains a request value as part of start events and reply values as part of

$$\Rightarrow\ \subseteq (\texttt{History} \times \texttt{History}) \tag{12}$$

$$\frac{h_1 \Rightarrow h_2 \quad h_2 \Rightarrow h_3}{h_1 \Rightarrow h_3} \tag{13}$$

$$\frac{h \triangleright (?[a^i, iv, ov] \,\|_{h'}\, [a^i, iv, ov])}{h_1 \bullet h \bullet h_2 \Rightarrow h_1 \bullet h' \bullet (S(a^i, iv)C(a^i, ov)) \bullet h_2} \tag{14}$$

$$\frac{h \triangleright (?[a^u, iv, ov] \,\|_{h'}\, [a^{-1}, iv, \text{nil}]) \quad (a^u, iv) \notin h_1 \quad (a^c, iv) \notin h'}{h_1 \bullet h \bullet h_2 \Rightarrow h_1 \bullet h' \bullet h_2} \tag{15}$$

$$\frac{h \triangleright (?[a^c, iv, \text{nil}] \,\|_{h'}\, [a^c, iv, \text{nil}]) \quad (a^u, iv) \notin h'}{h_1 \bullet h \bullet h_2 \Rightarrow h_1 \bullet h' \bullet (S(a^c, iv)C(a^c, \text{nil}) \bullet h_2} \tag{16}$$

Middleware Support for Precise Failure Semantics. Figure 4. Definition of history reduction.

completion events. The x-ability theory introduces the notion of *history signature*, which captures the client-side information (request and result) that is legal relative to a given server-side history. Because of non-determinism and server-side retry, a history can have multiple signatures. The set of signatures is defined by the following inference rules:

$$\frac{h \Rightarrow S(a^u, iv)C(a^u, ov)S(a^c, iv)C(a^c, \text{nil})}{(a, iv, ov) \in \texttt{signature}(h)} \tag{21}$$

$$\frac{h \Rightarrow S\left(a^i, iv\right)C\left(a^i, ov\right)}{(a, iv, ov) \in \texttt{signature}(h)} \tag{22}$$

If a client submits a sequence of requests, one after the other, later requests should be processed in the context of earlier requests. To prevent a service from forgetting the effect of previous requests, the x-ability theory assumes the existence of a set `PossibleReply` that contains the possible reply values for a given request. To capture the history-sensitive nature of the set of possible replies, PossibleReply is defined in the context of a request sequence $R_1 ... R_n$. The interpretation of `PossibleReply` in the context of a sequence is the set of possible replies to request R_n after the state machine has executed the requests $R_1 ... R_{n-1}$ one after the other. Thus, the set is written as follows: $\texttt{PossibleReply}_{(R_1 ... R_n)}$.

X-Able Services

The x-ability theory provides a formal specification of replication that is independent of a particular replication protocol. Formally speaking, a replicated service consists of a server-side state machine S and a client-side action *submit*. The state machine captures the functionality of the service. It is executed by a set of server processes $s_1 ... s_n$ that each have a copy of S. The action *submit* can be used by any process p to invoke the service. The action takes a value in the domain `Request` and, when executed, produces a value in the domain `Result`. Correctness is specified relative to a single client C. Thus, the considered system consists of the processes $s_1 ... s_n$ and C only. The client submits one request at a time, and the service is x-able if the following conditions hold:

- R1. The action *submit* is idempotent.
- R2. The client C will eventually be able to execute *submit* successfully.
- R3. If the client submits a request (a, iv), then the server-side history for (a, iv) is either empty or it satisfies $\texttt{x-able}_{(a,iv)}$.
- R4. If the client receives a reply ov in response to a request (a, iv), and if the server-side history for executing this request is h, then $(a, iv, ov) \in \texttt{signature}(h)$.
- R5. If the client successfully submits a sequence of requests, $R_1 ... R_n$, and receives the reply R' in response to R_n, then R' is in $\texttt{PossibleReply}_{(R_1 ... R_n)}$.

The first two requirements (R1 and R2) are concerned with the contract between a service and its clients. Clients use the action *submit* to invoke the service. Because *submit* is idempotent, clients can repeatedly invoke the service without concern for duplicating side-effects. The second requirement (R2) is a liveness property. The action *submit* is not allowed to fail an infinite number of times. The requirement also makes a service non-blocking in the sense that *submit* is guaranteed to eventually return a value. The third requirement (R3) deals with the server-side side-effect of executing a request. The resulting server-side history must be x-able, that is, it must be equivalent (under history reduction) to a failure-free history. The fourth requirement (R4) forces an algorithm to preserve consistency between the client-side view (request and reply) and the server-side view (the side-effect). This requirement, prevents the *submit* action from inventing reply values. It also prevents the service from inventing request values. The fifth requirement (R5) forces the service to correctly maintain S's state, if any. The server-side history must be equivalent to a failure-free execution of the sequence $R_1 ... R_n$. But since R_1 may result in a transformation of S's state, the actions executed for R_2 may depend on this state transformation. So, a replication algorithm must ensure that the state resulting from R_1 is used as a context for executing R_2. The replication algorithm cannot assume that R_1 did not update the state of S, or that the state update is immaterial to the processing of R_2.

Key Applications

A key application of the x-ability theory is the design transactions protocols for three-tier applications. Such applications encompass three layers: human users interact with front-end clients (e.g., browsers), middle-tier application servers (e.g., Web servers) contain the business logic of the application, and perform transactions against back-end databases. Three-tier applications usually rely on replication and transaction-processing techniques. It has been defined in [1,3] the notion of the Exactly-Once Transaction (e-Transaction) abstraction: an abstraction that encompasses both safety and liveness properties in three-tier environments and ensures end-to-end reliability.

Recommended Reading

1. Frølund S. and Guerraoui R. Implementing e-transactions with asynchronous replication. IEEE Trans. Parallel Distrib. Syst., 12 (2):133–146, 2001.
2. Frølund S. and Guerraoui R. X-ability: a theory of replication. Distrib. Comput., 14(4):231–249, 2001.
3. Frølund S. and Guerraoui R. e-Transactions: end-to-end reliability for three-tier architectures. IEEE Trans. Software Eng., 28(4):378–395, 2002.
4. Herlihy M. and Wing J.M. Linearizability: a correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst., 12(3):463–492, 1990.
5. Papadimitriou C.H. The serializability of concurrent database updates. J. ACM, 26(4):631–653, 1979.

Mini

▶ Snippet

Minimal-change Integrity Maintenance

▶ Constraint-Driven Database Repair

Mining of Chemical Data

Xifeng Yan

IBM T. J. Watson Research Center, Hawthorne,
NY, USA

Definition

Given a set of chemical compounds, chemical data mining is to characterize the compounds present in the data set and apply a variety of mining methods to discover relationships between the compounds and their biological and chemical activities.

Historical Background

In 1969, Hansch [6] introduced quantitative structure-activity relationship (QSAR) analysis which attempts to correlate physicochemical or structural properties of compounds with biological and chemical activities. These physicochemical and structural properties are determined empirically or by computational methods. QSAR prefers vectorial mappings of compounds, which are usually coded by existing physicochemical and structural fingerprints. Dehaspe et al. [3] applied inductive logic programming to predict chemical carcinogenicity by mining frequent substructures in chemical datasets, which identifies new structural fingerprints so that QSAR could build comprehensive analytical models.

Foundations

Chemical compounds are unstructured data with no explicit vector representation. For chemical compounds, various similarity measures are defined, which could be classified into three categories: (i) physicochemical property-based, e.g., toxicity and weight; (ii) structure-based; and (iii) feature-based. The structure-based similarity measure directly compares the topology of two chemical compounds, e.g., maximum common subgraph, graph edit distance, and graph kernel. As for the feature-based similarity measure, each graph is represented as a feature vector, $\mathbf{x} = [x_1, x_2,\ldots,x_n]$, where x_i is the value of feature f_i. A feature could be physicochemical or structural. The similarity between two graphs is measured by the similarity between their feature vectors. Bunke and Shearer [1] used maximum common subgraph to measure structure similarity. Given two graphs G and G', if P is the maximum common subgraph of G and G', then the structure similarity between G and G' is defined by

$$\frac{2|E(P)|}{|E(G)| + |E(G')|},$$

where $E(G)$ is the edge set of G.

Kashima et al. [7] introduced marginalized kernels between labeled graphs,

$$K(G, G') = \sum_{h} \sum_{h'} K_z(z, z')p(h|G)p(h'|G'),$$

where $z = [G,h]$ and $K_z(z,z')$ is the joint kernel over z. The hidden variable h is a path generated by random walks and the joint kernel K_z is defined as a kernel between these paths. Other sophisticated graph kernels are also available. For example, Fröhlich et al. [5]

proposed optimal assignment kernels for attributed molecular graphs, which compute an optimal assignment from the atoms of one molecule to those of another one, including local structures and neighborhood information.

The structure-based similarity measure can serve general chemical data mining such as chemical structure classification and clustering. The implicit definition of feature space makes it hard to interpret, and hard to adapt to many powerful data management and analytical tools such as R-tree and support vector machine. An alternative approach is to mine the most interesting features from chemical data directly, such as patterns that are discriminative between compounds with different chemical activities. Figure 1 depicts the pipeline of this approach built on features discovered by a mining process.

The feature-based mining framework includes three steps: (i) mine patterns/features from chemical data, (ii) select discriminative or significant features, and (iii) perform advanced mining. The first step is the process of finding and extracting useful features from raw datasets. One kind of features used in data mining is frequent substructures, the common structures that occur in many compounds. Formally, given a graph dataset $D = \{G_1,G_2,\ldots,G_n\}$ and a minimum frequency threshold θ, frequent substructures are subgraphs that are contained by at least $\theta|D|$ graphs in D. A set of graph pattern mining algorithms are available for mining frequent substructures, including SUBDUE, Warmr, AGM, gSpan, FSG, MoFa/MoSS, FFSM, Gaston, and so on. Generally, for mining graph patterns measured by an objective function F, there are two related mining tasks: (i) enumeration task, find all of subgraphs g such that $F(g)$ is no less than a threshold; and (ii) optimization task, find a subgraph g^* such that

$$g^* = \text{argmax}_g F(g).$$

The enumeration task might encounter the exponential number of patterns as the traditional frequent substructure mining does. To resolve this issue, one may rank patterns according to their objective score and select patterns with the highest value. The feature-based mining framework finds many key applications in chemical data mining including, but not limited to, chemical graph search, classification and clustering.

Chemical graph search aims to find graphs that contain a specific query structure. It is inefficient to scan the whole database and check each graph. Yan et al. [9] applied the feature-based mining framework to support fast search using frequent substructures selected by the following criterion. Let substructures $f_1,f_2,\ldots,f_n$ be selected features. Given a new substructure x, the selectivity power of x can be measured by

$$1 - Pr(x|f_{\varphi_1},\ldots,f_{\varphi_m}), f_{\varphi_i} \subseteq x, 1 \le \varphi_i \le n.$$

which shows the absence probability of x given the presence of $f_{\varphi_1},\ldots,f_{\varphi_m}$ in a graph. When the selectivity is high, substructure x is a good candidate to index.

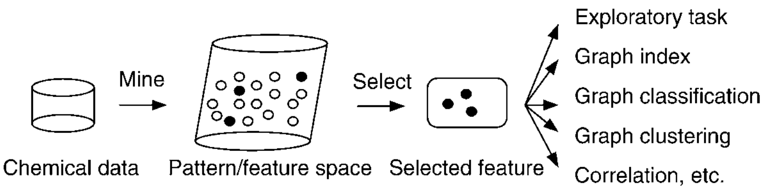
In addition to molecule search, chemical data classification and clustering could also benefit from graph patterns. A typical setting of molecule classification is to induce a mapping $h(\mathbf{g}) : \mathcal{G} \rightarrow \{\pm 1\}$ from the training samples $D = \{\mathbf{g}_i, y_i\}^n{}_{i=1}$, where $\mathbf{g}_i \in \mathcal{G}$ is a labeled graph and $y_i \in \{\pm 1\}$ is the class label. Feature-based classification models were proposed in [8,4]. In these models, graphs are first transformed to vectors using discriminative substructures, which are then processed by standard classification methods.

Key Applications

Chemical Structure Search
Chemical Classification
Chemical Clustering
Quantitative Structure-Activity Relationship Analysis

Experimental Results

PubChem (see dataset URL) provides information on the biological activities of small molecules, containing



Mining of Chemical Data. Figure 1. Feature-based mining framework.

the bioassay records for anti-cancer screen tests with different cancer cell lines. Each dataset belongs to a certain type of cancer screen with the outcome active or inactive. These bioassays are experimented to identify the chemical compounds that display the desired and reproducible behavior against cancers. Chemical compound classification is to computationally predict the activity of untested compounds given the bioassay data. This process can replace or supplement the physical assay techniques. Furthermore, it could also identify substructures that are critical to specific biological or chemical activities.

The following experiment demonstrates the effectiveness of graph kernel method [7,5] (optimal assignment kernel, OA) and pattern-based classification [8,4] (PA), both of which show good accuracy. From the PubChem screen tests, 11 bioassay datasets are displayed. Since the active class is very rare (around 5%) in these datasets, 500 active compounds and 2,000 inactive compounds are randomly sampled from each dataset for performance evaluation. The classification accuracy is evaluated with 5-fold cross validation. For both methods, the same implementation of support vector machine, LIBSVM [2], with parameter C selected from $[2^{-5}, 2^5]$, is used. Table 1 shows AUC by OA and PA. The area under the ROC curve (AUC) is a measure of the model accuracy, in the range of [0,1]. A perfect model will have an area of one. As shown in Table 1, PA achieves comparable results with OA. A detailed examination shows that PA is able to discover substructures that determine the activity of compounds, without domain knowledge.

Mining of Chemical Data. Table 1. Chemical compound classification

Dataset	OA	PA
MCF-7: Breast	0.68 ± 0.12	0.67 ± 0.10
MOLT-4: Leukemia	0.65 ± 0.06	0.66 ± 0.06
NCI-H23: Non-Small Cell Lung	0.79 ± 0.08	0.76 ± 0.09
OVCAR-8: Ovarian	0.67 ± 0.04	0.72 ± 0.06
P388: Leukemia	0.79 ± 0.07	0.82 ± 0.04
PC-3: Prostate	0.66 ± 0.09	0.69 ± 0.09
SF-295: Central Nerv Sys	0.75 ± 0.11	0.72 ± 0.12
SN12C: Renal	0.75 ± 0.08	0.75 ± 0.06
SW-620: Colon	0.70 ± 0.02	0.74 ± 0.06
UACC257: Melanoma	0.65 ± 0.05	0.64 ± 0.05
Yeast: Yeast anticancer	0.64 ± 0.04	0.71 ± 0.05
Average	0.70 ± 0.07	0.72 ± 0.07

Data Sets

PubChem provides bioassay records for anti-cancer screen tests with different cancer cell lines, available at http://pubchem.ncbi.nlm.nih.gov

Cross-references

- ▶ Frequent Graph Patterns
- ▶ Graph Classification
- ▶ Graph-based Clustering
- ▶ Graph Database
- ▶ Graph Database Mining
- ▶ Graph Kernel
- ▶ Graph Search

Recommended Reading

1. Bunke H. and Shearer K. A graph distance metric based on the maximal common subgraph. Pattern Recogn. Lett., 19:255–259, 1998.
2. Chang C.-C. and Lin C.-J. LIBSVM: a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
3. Dehaspe L., Toivonen H., and King R. Finding frequent substructures in chemical compounds. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998, pp. 30–36.
4. Deshpande M., Kuramochi M., Wale N., and Karypis G. Frequent substructure-based approaches for classifying chemical compounds. IEEE Trans. Knowl. Data Eng., 17:1036–1050, 2005.
5. Fröhlich H., Wegner J., Sieker F., and Zell A. Optimal assignment kernels for attributed molecular graphs. In Proc. 22nd Int. Conf. on Machine Learning, 2005, pp. 225–232.
6. Hansch C. A quantitative approach to biochemical structure-activity relationships. Acct. Chem. Res., 2:232–239, 1969.
7. Kashima H., Tsuda K., and Inokuchi A. Marginalized kernels between labeled graphs. In Proc. 20th Int. Conf. on Machine Learning, 2003, pp. 321–328.
8. Kramer S., Raedt L., and Helma C. Molecular feature mining in HIV data. In Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2001, pp. 136–143.
9. Yan X., Yu P.S., and Han J. Graph indexing: A frequent structure-based approach. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 335–346.

Mixed Evidence

▶ Contextualization

Mixed-Media

▶ Multimedia Metadata

MM Indexing

▶ Multimedia Data Indexing

MMDBMS

▶ Main Memory DBMS

Mobile Ad hoc Network Databases

▶ MANET Databases

Mobile Database

Ouri Wolfson
University of Illinois at Chicago, Chicago, IL, USA

Definition

A mobile database is a database that resides on a mobile device such as a PDA, a smart phone, or a laptop. Such devices are often limited in resources such as memory, computing power, and battery power.

Key Points

Due to device limitations, a mobile database is often much smaller than its counterpart residing on servers and mainframes. A mobile database is managed by a Database Management System (DBMS). Again, due to resource constraints, such a system often has limited functionality compared to a full blown database management system. For example, mobile databases are single user systems, and therefore a concurrency control mechanism is not required. Other DBMS components such as query processing and recovery may also be limited.

Queries to the mobile database are usually posed by the user of the mobile device. Updates of the database may originate from the user, or from a central server, or directly from other mobile devices. Updates from the server are communicated wirelessly. Such communication takes place either via a point-to-point connection between the mobile device (the client) and the server, or via broadcasting by the server [Acharya S., Franklin M., and Zdonik S, 1995, Dissemination-based Data Delivery Using Broadcast Disks, Personal Communications]. Direct updates from other mobile devices may use short-range wireless communication protocols such as Bluetooth or Wifi [1].

Cross-references

▶ Mobile Ad hoc Network Databases

Recommended Reading

1. Cao H., Wolfson O., Xu B., and Yin H., “MOBI-DIC: MOBIle DIscovery of loCal Resources in Peer-to-Peer Wireless Network”. Bull Comput Soc Tech Committee Data Eng, 28(3):11–18, 2005 (Special Issue on Database Issues for Location Data Management).

Mobile Interfaces

Giuseppe Santucci
University of Rome, Roma, Italy

Synonyms

Handhelds interfaces; Navigation system interfaces

Definition

Mobile interfaces are interfaces specifically designed for little portable electronic devices (handhelds), like cellular phones, personal digital assistants (PDAs), and pagers. Mobile interfaces provide means to execute complex activities on highly constrained devices, characterized by small screens, low computing power, and limited input/output (I/O) capabilities. While this term is mainly used to refer to interfaces for classical Web applications, like email and Web browsing it should be intended in a broader sense, encompassing all Web and non-Web applications that run on little mobile equipments (e.g., GPS navigation systems).

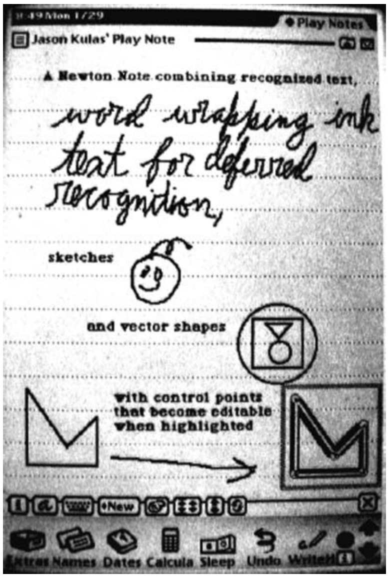
Historical Background

Mobile interfaces came up with the proliferation of cellular phones through the 1980s. At that time, the notion of mobile interface was very primitive and the offered services were very simple: handling lists of contacts, usually limited to <name, phone-number>

pairs, starting/answering a phone call, writing short text, setting phone preferences (volume, light, ring tone, etc.). Even in this restricted scenario, the vendors had to deal with challenging issues, posed by the very limited early cellular phones capabilities, in terms of screen dimension and resolution, limited computing power, and I/O capabilities. No standards were around and different, unrelated solutions were adopted. The most used interaction strategy was to mimic the typical hierarchical computer menus with very poor results: the user was (and still is) forced to access functions and services through a series of boring menus. The main reason of failure is that while PCs can present to the user a complete list of all possible choices mobile phones can show a small number of options at time (usually one), forcing the user to remember the paths to the commands. That results in a very large number of key presses, errors, and mental overwhelming: many of the advantages of the conventional menus are lost [12].

For about ten years the unique available portable devices were mainly cellular phones; from 1993 to 1998 Apple Computer (now Apple Inc.) marketed the "Apple Newton," the first line of PDAs, personal digital assistants, a term introduced on January 7, 1992 by John Sculley at the Consumer Electronics Show in Las Vegas, Nevada, referring to the Apple Newton. The Apple's official name for the PDA was "MessagePad" and it was based on the ARM 610 RISC processor using a dedicated operating system (Newton OS, an allusion to Isaac Newton's apple); however, the word Newton was popularly used to refer to the device and its software. The new device was characterized by a wider touchable monochromatic screen (366 × 240) with retro illumination (only for the top version). The increased computational power and the larger screen allowed for very innovative interaction techniques (for a portable device): icons, simulated touchable QWERTY keyboard, and handwriting recognition (called Calligrapher). Moreover, the user was allowed to turn the screen horizontally ("landscape") as well as vertically ("portrait") preserving the handwriting recognition functionality (see Fig. 1).

Even the data management was quite innovative: programs were able to convert and share data (e.g., the calendar was able to refer to names in the address book or to convert a note into an appointment). Finally all the devices were equipped with a built in infrared and an expansion port for connecting to a modem or Ethernet. Disregarding the color absence and the limited device connectivity, the Palm Newton functionalities were quite similar to the ones of modern PDAs!



A Newton Note combining recognized text,

word wrapping ink text for deferred recognition,

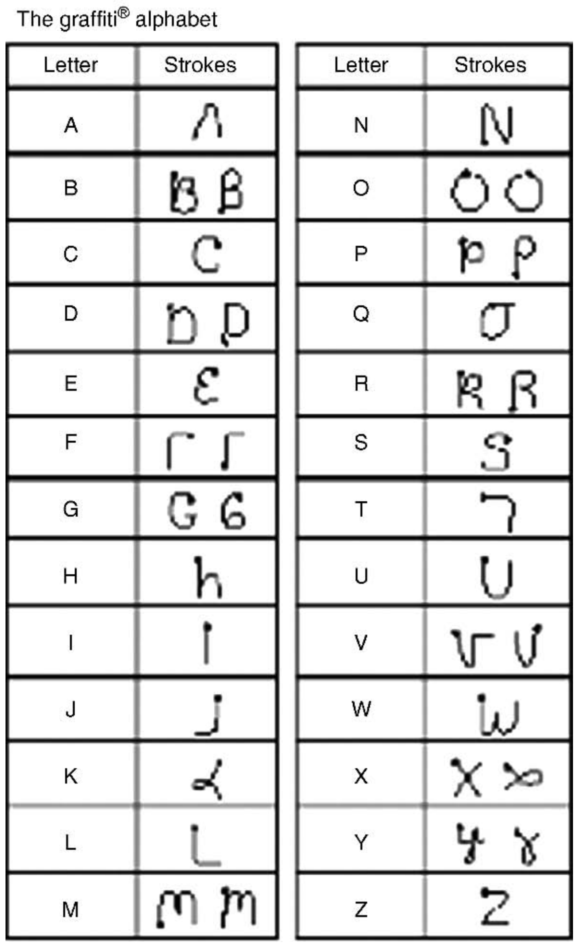
sketches

and vector shapes

with control points that become editable when highlighted

Mobile Interfaces. Figure 1. A screenshot from Apple Newton.

In spite of the innovative interface and capabilities, the product was not a successful one. The main reasons were the poor handwriting recognition accuracy, the high price, and the non comfortable size (it did not fit in a regular pocket). Still, the pioneering Apple ideas were captured by other vendors and many similar devices were around in the next years. Among them, the Palm series with the PalmOS operative system and the Graffiti handwriting recognition system conquered (in 1999) about 80% of the world market, mostly for the effective handwriting system and for the availability of a huge amount of third part software. On February 2000 Palm marketed the PalmIIIc, the first color PDA. Concerning the interface it is worth noting the different Graffiti philosophy: instead of allowing the user to write in his/her own style it forced the user to learn a set of pen strokes for each character (see Fig. 2). This narrowed the possibility for erroneous input, although memorization of the stroke patterns did increase the learning curve for the user. Some studies demonstrated that even if the error rate with Graffiti was higher than using the virtual keyboard (19.3 vs. 4.1%), users prefer Graffiti, because its usage is more natural.


The graffiti® alphabet
Letter
Strokes
Letter
Strokes
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z

Mobile Interfaces. Figure 2. The PalmOS Graffiti.

As soon as the hardware made it possible a deeper integration between PDA and cellular phones started: phones were equipped with larger screen and complex applications (smartphones), while PDA included phone hardware. Moreover, the so called pager devices, mainly intended for on the way email, came up. In 2004, smartphones were outselling PDAs for the first time. Nowadays, the difference between these three categories in term of capabilities and interface is very little. The main consequence of this integration is that all these devices allow for Internet access (through GPRS or WI-FI) and it is very likely that in the next future there will be more people accessing the Internet via mobile devices (phones, PDA, etc) than via conventional PCs. According to this issue many researchers are now dealing with the problem of designing friendly interface for accessing Internet through portable devices.

Foundations

The research effort concerning mobile interfaces followed two main paths: designing interface for a specific application running on a specific device (e.g., the agenda interface of a specific cellular phone) or designing web based applications that can be deployed on multiple devices (e.g., a web browser working on different PDAs and cellular phones).

Mobile Interface for Specific Devices

The problem of dealing with little screen is not new. Much literature from the 1980s and early 1990s, written even before the Web, deals with user interfaces on small screens. At that time the focus was on first generation cash dispensing ATMs, electronic typewriters and photocopiers. All of these systems could display only a limited number of text lines to communicate with the user. Research looked into the impact of reduced screen size on comprehension [3], reading rate [4], and interaction [12]. This research is still valid and has new relevance to mobile Web devices [8].

The availability of graphic display and the diffusion of navigation system raised the issue of interacting with graphics and maps [6]. Moreover, the increasing usage of such systems while driving a car posed several new issues concerning distracting factors and security [1].

Multi Target Applications

In this case, research deals with techniques able to support the design of applications that run on several devices that present different interaction capabilities (e.g., small/large screens, keyboard/keypad input, etc.). In order to address this issue, researchers have adopted two main approaches:

1. Designing applications in order to let them run on different platforms
2. Adapting existing systems in order to render them usable on platform that were not originally included as service provider (e.g., standard web sites)

In the first approach the solution is offered at design time, that is, the designer explicitly knows that the system will run on multiple devices. Some research exists that follows this approach and the key issues consist in giving models, methods, and tools to support the designer in the creation of multi-device applications and in offering techniques and algorithms to generate the final interface at run time for the specific device utilized. In [10], a tool for support is presented together with a well defined development life cycle: the designer starts from abstract tasks definition and upon it designs the interface in abstract. The final interfaces are statically generated at design time by following suggestions offered by the system and further refined in order to accommodate specific details. Puerta et al. in [5] propose a similar framework, defining a model to design user interfaces in abstract as well, but their work envisions also adaptive techniques to produce the final user interface. A central mediator agent is responsible for translating the abstract specification of the user interface, according to a description of device's characteristics, into the final interface. Fundamental questions arising from this kind of research are what kind of models should be employed to specify abstract interaction elements and what strategies/techniques can permit an effective translation of abstract models into real interfaces.

The second approach is the one that attempts to bring existing systems, typically web sites, to small screen devices by means of some sort of adaptation/filtering. Among them, the WAP forum (www.wapforum.org) defined a protocol that enables Web-like services on little, portable devices. The interface uses a card metaphor and the designers claim that it is highly appropriate and usable. However, Nielson [11] raised doubts about its effectiveness. A similar proposal is i-mode, developed by the dominant Japanese carrier NTT DoCoMo, and widely used in Japan. Moreover, the World Wide Web Consortium (W3C) is also developing a framework that will enable Web content to be accessed on a diverse range of devices [13]. This framework will allow a document to exist in multiple variants, each variant specifying the type of support needed by the browser to display its contents. Device capabilities and user preferences will also be captured. The information about documents, devices and users will be used to automatically adapt a Web page to best suit the device and user.

Key Applications

Users of mobile interfaces are growing at high speed. According to IDC's Worldwide Quarterly Mobile Phone Tracker (www.idc.com), worldwide mobile phone shipments rose 19.1% year over year and increased sequentially 8.8% in 2005 to reach 208.3 million units. While unconnected PDA's are a declining segment, the emerging navigator market is still increasing about 10 million cars guided by navigation systems, 43 million dedicated portable navigators, and 16 millions smart phones based navigators forecasted for 2010 (www.strategyanalytics.net).

These figures make clear the importance of mobile interfaces in the coming years.

Future Directions

There are some emerging issues that will likely affect the behavior and structure of mobile interfaces:

- Search versus menu browsing. Several proposals relies on the idea that, in order to start an action, it is better to search it (i.e., to write down the command name) instead of browsing boring menus [9,7].
- Multimodal interaction. When the hardware allows it, mobile interfaces can exploit multimodal input/output (e.g., voice commands, likely together with the aforementioned search strategy).
- Context and user modeling. Some proposal describe interfaces that can adapt themselves according to the user preferences and the context [2], e.g., an interface for browsing tourist information presents a user with list of vegetarian restaurants at walking distance, according to the user location, discovered through the integrated GPS device, knowing that the user is vegetarian and that s/he is currently walking.

These research issues could greatly improve the mobile interfaces in the next years.

Cross-references

▶ Multimodal Interfaces
▶ Visual Interfaces
▶ WIMP Interfaces

Recommended Reading

1. Commission of the European Communities. Commission recommendation of 22 December 2006 on safe and efficient in-vehicle information and communication systems: Update of the European Statement of Principles on Human Machine Interface, 2006.
2. Coutaz J., Crowley J., Dobson S., and Garlan D. Context is key. Commn. ACM, 48(3):49–53, 2005.
3. Dillon A., Richardson J., and McKnight. The effect of display size and text splitting on reading lengthy text from the screen, Behav. Inf. Technol., 9(3):215–227, 1990.
4. Duchnicky R.L. and Kolers P.A. Readability of text scrolled on visual display terminals as a function of window size. Hum. Factors, 25(6):683–692, 1983.
5. Eisenstein J., Vanderdonckt J., and Puerta A. Applying model-based techniques to the development of UIs for mobile

computers. In Proc. Sixth Int. Conf. on Intelligent User Interfaces, 2001, pp. 69–76.

6. Frey P.R., Rouse W.B., and Garris R.D. Big graphics and little screens: designing graphical displays for maintenance tasks. IEEE Trans. Syst. Man Cybernetics, 22(1): 10–20, 1992.
7. Graf S., Spiessl W., Schmidt A., Winter A., and Rigoll G. In-car interaction using search based user interfaces. In Proc. 26th Annual SIGCHI Conf. on Human factors in Computing Systems. 2008, pp. 1685–1688.
8. Jones M., Marsden G., Mohd-Nasir N., Boone K., and Buchanan G. Improving web interaction on small displays. In Proc. W8 Conf., Toronto, 1999 Also reprinted in Int. J. Comput. Telecommn. Netw., 31(11–16):1129–1137, 1999.
9. Marsden G., Gillary P., Jones M. and Thimbleby H. Successful user interface design from efficient computer algorithms. In Proc. ACM CHI 2000 Conf. on Human Factors in Computing Systems, 2000, pp. 181–182.
10. Mori G., Paternò F., and Santoro C. Tool support for designing nomadic applications. In Proc. 2003 Int. Conf. on Intelligent User Interfaces, 2003, pp. 141–148.
11. Nielson J. Graceful degradation of scalable internet services. Available online at:http://www.useit.com/alertbox/991031.html, 1999.
12. Swierenga S.J. Menuing and scrolling as alternative information access techniques for computer systems: interfacing with the user. In Proc. 34th Annual Meeting of Human Factors Society, 1990, pp. 356–359.
13. W3C Mobile Activity Statement. Available online at: http://www.w3.org/Mobile/Activity.

Mobile Map Services

► Location-Based Services (LBS)

Mobile Sensor Network Data Management

Demetrios Zeinalipour-Yazti[1]
Panos K. Chrysanthis[2]
[1]University of Cyprus, Nicosia, Cyprus
[2]University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

MSN data management; Mobile wireless sensor network data management

Definition

Mobile Sensor Network (MSN) Data Management refers to a collection of centralized and distributed algorithms, architectures and systems to handle (store, process and analyze) the immense amount of spatio-temporal data that is cooperatively generated by collections of sensing devices that move in space over time.

Formally, given a set of n homogenous or heterogeneous mobile sensors $\{s_1, s_2,...,s_n\}$ that are capable of acquiring m physical attributes $\{a_1, a_2,...,a_m\}$ from their environment at every discrete time instance t (i.e., datahas a temporal dimension), an implicit or explicit mechanism that enables each s_i $(i \leq n)$ to move in some multi-dimensional Euclidean space (i.e., data has one or more spatial dimensions), MSN Data Management provides the foundation to handle spatio-temporal data in the form $(s_i, t, x, [y, z,]a_1[,...,a_m])$, where x, y, z defines three possible spatial dimensions and the bracket expression "[ ]" denotes the optional arguments in the tuple definition. In a more general perspective, MSN Data Management deals with algorithms, architectures and systems for in-network and out-of-network query processing, access methods, storage, data modeling, data warehousing, data movement and data mining.

Historical Background

The improvements in hardware design along with the wide availability of economically viable embedded sensor systems have enabled scientists to acquire environmental conditions at extremely high resolutions. Early approaches to monitor the physical world were primarily composed of passive sensing devices, such as those utilized in wired weather monitoring infrastructures, that could transmit their readings to more powerful processing units for storage and analysis. The evolution of passive sensing devices has been succeeded by the development of *Stationary Wireless Sensor Networks (Stationary WSNs)*. These are composed of many tiny computers, often no bigger than a coin or a credit card, that feature a low frequency processor, some flash memory for storage, a radio for short-range wireless communication, on-chip sensors and an energy source such as AA batteries or solar panels. Applications of stationary WSNs have emerged in many domains ranging from environmental monitoring [15] to seismic and structural monitoring as well as industry manufacturing.

The transfer of information in such networks is conducted without electrical conductors (i.e., wires) using technologies such as radio frequency (RF), infrared light, acoustic energy and others, as the mobility aspect inherently hinders the deployment of any technology that physically connects nodes with wires. Since communication is the most energy demanding factor in such networks, data management researchers have primarily focused on the development of energy-conscious algorithms and techniques.

In particular, declarative approaches such as TinyDB [9] and Cougar [16] perform a combination of in-network aggregation and filtering in order to reduce the energy consumption while conveying data to the *querying node (sink)*. Additionally, approaches such as TiNA [13] and MINT Views [17] take into account intelligent in-network data reduction techniques to further reduce the consumption of energy. *Data Centric Routing* approaches, such as directed diffusion [8], establish low-latency paths between the sink and the sensors in order to reduce the cost of communication. *Data Centric Storage* [14] schemes organize data with the same attribute (e.g., humidity readings) on the same node in the network in order to offer efficient location and retrieval of sensor data.

The evolution of stationary WSNs in conjunction with the advances made by the distributed robotics and low power embedded systems communities have led to a new class of *Mobile (and Wireless) Sensor Networks (MSNs)* that can be utilized for land [3,5,10], ocean exploration [11], air monitoring [1], automobile applications [7,6], Habitant Monitoring [12] and a wide range of other scenarios. MSNs have a similar architecture to their stationary counterparts, thus are governed by the same energy and processing limitations, but are supplemented with implicit or explicit mechanisms that enable these devices to move in space (e.g., motor or sea/air current) over time. Additionally, MSN devices might derive their coordinates through absolute (e.g., dedicated Geographic Positioning System hardware) or relative means (e.g., *localization techniques*, which enable sensing devices to derive their coordinates using the signal strength, time difference of arrival or angle of arrival). There are several classes of MSNs which can coarsely be structured into the following classes: (i) highly mobile, which contains scenarios in which devices move at high velocities such as cars, human with cell phones, airplanes, and others; (ii) mostly static, which contains scenarios in which devices move at low velocities such as monitoring sensors in a shop floor with moving robots; and (iii) hybrid, which contains both classes such as an airplane that has sensors installed on inside and outside.

Foundations

The unique characteristics of MSNs create novel data management opportunities and challenges that have not been addressed in other contexts including those of mobile databases and stationary WSNs. In order to realize the advantages of such networks, researchers have to re-examine existing data management and data processing approaches in order to consider sensor and user mobility; develop new approaches that consider the impact of mobility and capture its trade-offs. Finally, MSN data management researchers are challenged with structuring these networks as huge distributed databases whose edges consist of numerous "receptors" (e.g., RFID readers or sensor networks) and internal nodes form a pyramid scheme for (in-network) aggregation and (pipelined) data stream processing.

There are numerous advantages of MSNs over their stationary counterparts. In particular, MSNs offer: (i) *dynamic network coverage*, by reaching areas that have not been adequately sampled; (ii) *data routing repair*, by replacing failed routing nodes and by calibrating the operation of the network; (iii) *data muling*, by collecting and disseminating data/readings from stationary nodes out of range; (iv) *staged data stream processing*, by conducting in-network processing of continuous and ad-hoc queries; and (v) *user access points*, by enabling connection to handheld and other mobile devices that are out of range from the communication infrastructure.

These advantages enable a wide range of new applications whose data management requirements go beyond those of stationary WSNs. In particular, MSN system software is required to handle: (i) *the past*, by recording and providing access to history data; (ii) *the present*, by providing access to current readings of sensor data; (iii) *the future*, by generating predictions; (iv) *distributed spatio-temporal data*, by providing new means of distributed data storage, indexing and querying of spatio-temporal data repositories; (v) *data uncertainty*, by providing new means of handling real world signals that are inherently uncertain; (vi) *self-configurability*, by withstanding "harsh" real-life environments; and (vii) *data and service mash-ups*, by enabling other innovative applications that build on top of existing data and services.

In light of the above characteristics, the most predominant data management challenges that have prevailed in the context of MSNs include:

In-Network Storage: The absence of a stationary network structure in MSNs makes continuous data acquisition to some sink point a non-intuitive task (e.g., mobile nodes might be out of communication range from the sink). In particular, the absence of an always accessible sink mandates that acquisition has to be succeeded by in-network storage of the acquired events so that these events can later be retrieved by the user. Mobile devices usually utilize flash memory as opposed to magnetic disks, which are not shock-resistant and thus are not appropriate for a mobile setting. Consequently, a major challenge in MSNs is to extend local storage structures and access methods in order to provide efficient access to the data stored on the local flash media of a sensor device while traditional database research has mainly focused on issues related to magnetic disks.

Flexible and Expressive Query Types: In a traditional database management system, there is a single correct answer to a given query on a given database instance. When querying MSNs the situation is notably different as there are many more degrees of freedom and the underlying querying engine needs to be guided regarding which alternative execution strategy is the right one, typically on the basis of target answer quality and resource availability. In this context, there are additional relevant parameters that include: (i) *Resolution:* physical sensor data can be observed at multiple resolutions along space and time dimensions; (ii) *Confidence:* more often than not, correctness of query results can be specified only in probabilistic terms due to the inherent uncertainty in the sensor hardware and the modeling process; (iii) *Alternative models:* in some cases, several alternative models apply to a single scenario. Each alternative typically represents a different point in the efficiency (resource consumption) and effectiveness (result quality) spectrum, thereby allowing a tradeoff between these two metrics on the basis of application-level expectations. The prime challenge is to define new declarative query languages that make use of these new parameters while allowing a highly flexible and optimizable implementation. Additionally, approximate query processing with controlled result accuracy becomes vital for dynamic mobile environments with varying node velocities, changing data traffic patterns, information redundancy, uncertainty, and inevitable flexible load shedding techniques. Finally, in order to have an efficient and optimized implementation of query types, MSNs will need to consider cross-layer optimization since all layers of the data stack are involved in query execution.

Efficient Query Routing Trees: Query routing and resolution in stationary WSNs is typically founded on some type of query routing tree that provides each sensor with a path over which answers can be transmitted to the sink. In a MSN, such a query routing tree can neither be constructed in an efficient manner nor be maintained efficiently as the network topology is transient. The dynamic nature of the underlying physical network tremendously complicates the interchange of information between nodes during the resolution of a query. In particular, it is known that sensing devices tend to power-down their transceiver (transmitter-receiver) during periods of inactivity in order to conserve energy [2]. While stationary WSNs define transceiver scheduling approaches, such as those defined in TAG [9], Cougar [16] and MicroPulse [1], in order to enable accurate transceiver allocation schemes, such approaches are not suitable for mobile settings in which a sensor is not aware of its designated parent node in the query tree hierarchy. Consequently, nodes are not able to agree on rendezvous time-points on which data interchange can occur.

Purpose-Driven Data Reduction: The amount of data generated from MSNs can be overwhelming. Consequently, a main challenge is to provide data reduction techniques which will be tuned to the semantics of the target application. Furthermore, data reduction must take into account the entire spectrum of uses, ranging from real-time to off-line, supporting both snapshot and continuous queries that take advantage of designated optimization opportunities (e.g., multi-query) especially targeted for mobile environments. Finally, it must also consider the inherently dynamic aspects of these environments and the possibility of in-network data reduction (e.g., in-network aggregation).

Perimeter Construction and Swarm-Like Behavior: In many types of MSNs, new events are more prevalent at the periphery of the network (e.g., water detection and contamination detection) rather than uniformly throughout the network (which is more typically for applications like fire detection). This creates the necessity to construct the perimeter of a MSN in an online and distributed manner. Additionally, many types of MSNs are expected to feature a swarm-like behavior (The term *Swarm (or Flock)* refers to a group of objects that exhibit a polarized, non-colliding and aggregate motion.). For instance, consider a MSN design that consists of several rovers that are deployed as a swarm in order to detect events of interest (e.g., the presence of water) [18]. The swarm might collaboratively collect spatio-temporal events of interest and store them in the swarm until an operator requests them. In order to increase the availability of the detected answers, in the presence of unpredictable failures, individual rovers can perform replication of detected events to neighboring nodes. That creates challenges in data aggregation, data fusion and data storage that have not yet been addressed.

Enforcement of Security, Privacy and Trust: Frequent node migrations and disconnections in MSNs, as well as resource constraints raise severe concerns with respect to security, privacy and trust. Additionally, the cost of traditional secure data dissemination approaches (e.g., using encryption) may be prohibitively high in volatile mobile environments. As such, research on encryption-free data dissemination strategies becomes very relevant here. This includes strategies to deliver separate and under-defined data shares, secure multiparty computation and advanced information recovery techniques.

Context-Awareness and Self-Everything: Providing a useful level of situational awareness in an unobtrusive way is crucial to the success of any application utilizing MSNs as this can be used to improve functionality by including preferences from the users but can also be used to improve performance (e.g., better network routing decisions if the exact topology is known). Note that context is often obvious in stationary WSN deployments (i.e., a specific sensor is always in the same location) but in the context of a MSN additional data management measures need to be taken into account in order to enable this parameter. Additionally, it is crucial for them to be "plug-and-play" and self-everything (i.e., self-configurable and self-adaptive) as application deployment of sensors in the field is famously hard, even without the mobility aspect which is introducing additional challenges. Finally, a crucial parameter is that of being adaptive both in how to deal with the system issues (i.e., how to adapt from failures in network connectivity) and also with user-interface/application issues (i.e., how to adapt the application when the context changes).

Key Applications

MSN Data Management algorithms, architectures and systems will play a significant role in the development of future applications in a wide range of disciplines including the following:

Environmental and Habitant Monitoring: A large class of MSN applications have already emerged in the context of environmental and habitant monitoring systems. Consider an ocean monitoring environment that consists of *n* independent surface drifters floating on the sea surface and equipped with either acoustic or radio communication capabilities. The operator of such a MSN might seek to answer queries of the type: "*Has the MSN identified an area of contamination and where exactly?*". The MSN architecture circumvents the peculiarities of individual sensors, is less prone to failures and is potentially much cheaper. Similar applications have also emerged with MSNs of car robots, such as CotsBots [3], Robomotes [5] or Millibots [10], and MSNs of Unmanned Aerial Vehicles (UAVs), such as SensorFlock [1], in which devices can fly autonomously based on complex interactions with their peers. One final challenging application in this class is that of detecting a phenomenon that itself is mobile, for example a brush fire which is being carried around by high winds.

Intelligent Transportation Systems: Sensing systems have been utilized over the years in order to better manage traffic with the ultimate goal of reducing accidents and minimizing the time and the energy (gasoline) wasted while staying idle in traffic. Since cars are already equipped with a wide range of sensors, the generated information can be shared in a vehicle-to-vehicle network. For example the ABS system can detect when the road is slippery or when the driver is hitting the brakes thus this information can be broadcasted to the surrounding cars but also to the many cars back and forth, as needed, in order to make sure that everybody can safely stop with current weather conditions and car speeds.

Medical Applications: This class includes applications that monitor humans in order to improve living conditions and in order to define early warning systems that identify when human life is at risk. For instance, Nike+ is an example for monitoring the health of a group of runners that have simple sensing devices embedded in their running shoes. Such an application would require embedded storage and retrieval techniques in order to administer the local amounts of data. Applications in support of the elderly and those needing constant supervision (e.g., due to chronic diseases like diabetes, allergies, etc.) are another example in which MSN data management techniques will play an important role. Wellness applications could also be envisioned, where a health "dose" of exercise is administered according to ones needs and capabilities. Another area are systems to protect soldiers on the battlefield. SPARTNET has recently developed wearable physiological sensor systems that collect, organize and interpret data on the health status of soldiers in order to improve situational and medical awareness during field trainings. Such systems could be augmented with functionality of detecting and reporting threats that are either derived from individual signals (e.g., when a soldiers personal health monitor shows erratic life-signals) and from correlated signals that are derived from multiple sensors/soldiers (e.g., by recognizing when a small group of soldiers is deviating away from the expected formation). Finally, disaster and emergency management are another prime area where MSN data management techniques will play a major impact.

Location-Based Services and the Sensor Web: The last group of challenging motivating applications is that of real-time location-based services, for example a service that can report whether there are any available parking spaces or a service that can keep track of buses moving and report how delayed a certain bus is. Many of these services become more powerful with the integration of data from the *Sensor Web* (i.e., live sensor data) with the *Web* (i.e., static content available online) and the *Deep-web* (i.e., data that is stored in a database, but are accessible through a web page or a web service).

Cross-references

- ▶ Mobile and Ubiquitous Data Management
- ▶ Sensor Networks
- ▶ Spatial Network Databases
- ▶ Stream Data Analysis

Recommended Reading

1. Allred J., Hasan A.B., Panichsakul S., Pisano B., Gray P., Huang J-H., Han R., Lawrence D., and Mohseni K. SensorFlock: an airborne wireless sensor network of micro-air vehicles. In Proc. 5th Int. Conf. on Embedded Networked Sensor Systems, 2007, pp. 117–129.
2. Andreou P., Zeinalipour-Yazti D., Chrysanthis P.K., and Samaras G. Workload-aware optimization of query routing trees in wireless sensor networks In Proc. 9th Int. Conf. on Mobile Data Management, 2008, pp. 189–196.
3. Bergbreiter S. and Pister K.S.J. CotsBots: an off-the-shelf platform for distributed robotics. In Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2003, pp. 1632–1637.
4. Chintalapudi K. and Govindan R. Localized Edge Detection in Sensor Fields. Ad-hoc Networks, 1(2–3):273–291, 2003.
5. Dantu K., Rahimi M.H., Shah H., Babel S., Dhariwal A., and Sukhatme G.S. Robomote: enabling mobility in sensor networks. In Proc. 4th Int. Symp. on Information Processing in Sensor Networks, 2005, pp.
6. Eriksson J., Girod L., Hull B., Newton R., Madden S., and Balakrishnan H. The Pothole Patrol: using a mobile sensor network for road surface monitoring. In Proc. 6th Int. Conf. Mobile Systems, Applications and Services, 2008, pp. 29–39.
7. Hull B., Bychkovsky V., Chen K., Goraczko M., Miu A., Shih E., Zhang Y., Balakrishnan H., and Madden S. CarTel: a distributed mobile sensor computing system. In Proc. 4th Int. Conf. on Embedded Networked Sensor Systems, 2006, pp. 125–138.
8. Intanagonwiwat C., Govindan R., and Estrin D. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In Proc. 6th Annual Int. Conf. on Mobile Computing and Networking, 2000, pp. 56–67.
9. Madden S.R., Franklin M.J., Hellerstein J.M., and Hong W. The design of an acquisitional query processor for sensor networks. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
10. Navarro-Serment L.E., Grabowski R., Paredis C.J.J., and Khosla P.K. Millibots: the development of a framework and algorithms for a distributed heterogeneous robot team. IEEE Robot. Autom. Mag., 9(4), December 2002.
11. Nittel S., Trigoni N., Ferentinos K., Neville F., Nural A., and Pettigrew N. A drift-tolerant model for data management in ocean sensor networks. In Proc. 6th ACM Int. Workshop on Data Eng. for Wireless and Mobile Access, 2007, pp. 49–58.
12. Sadler C., Zhang P., Martonosi M., and Lyon S. Hardware design experiences in ZebraNet. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 227–238.
13. Sharaf M., Beaver J., Labrinidis A., and Chrysantrhis P.K. Balancing energy efficiency and quality of aggregate data in sensor networks. VLDB J., 13(4):384–403, 2004.
14. Shenker S., Ratnasamy S., Karp B., Govindan R., and Estrin D. Data-centric storage in sensornets. SIGCOMM Comput. Commun. Rev., 33(1):137–142, 2003.
15. Szewczyk R., Mainwaring A., Polastre J., Anderson J., and Culler D. An analysis of a large scale habitat monitoring application. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 214–226.
16. Yao Y. and Gehrke J.E. The cougar approach to in-network query processing in sensor networks. ACM SIGMOD Rec., 32(3):9–18, 2002.
17. Zeinalipour-Yazti D., Andreou P., Chrysanthis P., and Samaras G. MINT views: materialized in-network top-k views in sensor networks. In Proc. Int. Conf. on Mobile Data Management, 2007, pp. 182–189.
18. Zeinalipour-Yazti D., Andreou P., Chrysanthis P., and Samaras G. SenseSwarm: a perimeter-based data acquisition framework for mobile sensor networks. In Proc. VLDB Workshop on Data Management for Sensor Networks, 2007, pp. 13–18.

Mobile Wireless Sensor Network Data Management

▶ Mobile Sensor Network Data Management

Model Management

Christoph Quix
RWTH Aachen University, Aachen, Germany

Definition

Model management comprises technologies and mechanisms to support the integration, transformation, evolution, and matching of models. It aims at supporting metadata-intensive applications such as database design, data integration, and data warehousing. To achieve this goal, a model management system has to provide definitions for *models* (i.e., schemas represented in some metamodel), *mappings* (i.e., relationships between different models), and *operators* (i.e., operations that manipulate models and mappings). Model management has become more and more important, since the interoperability and/or integration of heterogeneous information systems is a frequent requirement of organizations. Some important operations in model management are *Merge* (integration of two models), *Match* (creating a mapping between two models), and *ModelGen* (transforming a model given in one modeling language into a corresponding model in a different modeling language).

The current understanding of model management has been defined in [4] and focuses mainly on (but is not limited to) the management of *data* models. Most of the problems mentioned before have been already addressed separately and for specific applications. The goal now is to build a model management system (MMS) which unifies the previous approaches by providing a set of *generic* structures representing models and mappings, and the definition of *generic* operations on these structures. Such a system could then be used by an application to solve model management tasks.

Historical Background

Model management, as it is understood today, has been defined by Bernstein et al. [4]. In the 1980s, the term “model management” was used in the context of decision support systems, but this refered mainly to mathematical models. Dolk [7] stated first the requirement for a theory for models similar to the relational database theory. Such a theory should include formal definitions of models and operations on models and could be used as a basis for the implementation of a model management system. This work was based on a draft of the *Information Resource Dictionary System* (IRDS) standard (ISO/IEC 10027:1990) which was accepted in 1990. The IRDS standard clarified the terminology of modeling systems and defined a framework structure for such systems as a four-level hierarchy: at the lowest level reside data instances which are described by a model (or schema) on the next higher level. This model is expressed in some modeling language (or metamodel) which is located at the third level. The highest level contains a metametamodel which can be used to define metamodels.

The new definition of model management in 2000 [4] integrated the research efforts of several previously loosely coupled areas. Therefore, research in model management did not start from scratch, rather it could build already on many results such as schema integration [3], or model transformation [1]. The main contribution of [4] was the definition of operations (such as Match, Merge, Compose) which a model management system should offer. Furthermore, it was required that models and mappings are considered as first class objects and that operations should address them as a whole and not only one model element at a time.

Since the vision of model management has been stated, research diverted into the areas of schema matching [14], model transformation [1], generic metamodels [10], schema integration [3], and the definition and composition of mappings [8]. Each of these areas will be summarized briefly in the next section.

Recently, research on model management has been summarized in [5]. It was also emphasized that more expressive mapping languages are required than proposed in the original vision of model management. Model management systems should also include a component in which the mappings can be executed.

Foundations

Schema Matching

Schema matching is the task of identifying a set of correspondences (also called a morphism or a mapping) between schema elements. Many aspects have to be considered during the process of matching, such as data values, element names, constraint information, structure information, domain knowledge, cardinality relationships, and so on. All this information is useful in understanding the semantics of a schema, but it can be a very time consuming problem to collect this information. Therefore, automatic methods are required for schema matching.

A multitude of methods have been proposed for schema matching [14] using different types of information to identify similar elements. The following categories of schema matchers are frequently used:

- Element-Level Matchers take only the information of one schema element separately into account. These can either use linguistic information (name of the element) or constraint information (data type, key constraints).
- Structure-Level Matchers use graph matching approaches to measure the similarity of the structures implied the schema.
- Instance-Level Matchers use also data instances to match schema elements. If the instance sets of two elements are similar, or have a similar value distribution, this might indicate a similarity of the schema elements.
- Machine-Learning Matchers use either instance data or previously identified matches as training data for a machine-learning system. Based on this training data, the system should then detect similar matches in new schema matching problems.

It has been agreed that no single method can solve the schema matching problem in general. Therefore, matching frameworks have been developed which are able to combine multiple individual matching methods to achieve a better result.

Model Transformation

Model transformation (also called *ModelGen*) is the task of transforming a given model M in some particular modeling language into a corresponding model M' in some other modeling language. A classical example for such a transformation is the transformation of an entity-relationship model into a relational database schema. In general, the transformation cannot guarantee that all semantic information of M is still present in the resulting model M' as the target modeling language might have limited expressivity.

Whereas transformation of models between different modeling languages is a frequent task, it has up to now mainly been adressed in specialized settings which map from one particular metamodel to another fixed metamodel. Recent approaches for generic model transformations are based on generic model representations and use a rule-based system to transform a generic representation of the original model into a different model in the generic metamodel. As a side effect, these approaches generate also instance-level mappings which are able to transform data conforming to the original model into data of the generated model.

Another important application area of model transformation is MDA (Model Driven Architecture). The MDA concept is based on the transformation of abstract, conceptual models into more concrete implementation-oriented models.

Generic Metamodel

A generic representation of models is a prerequisite for building a model management system. Without a generic representation, model operations would have to be implemented for each modeling language that should be supported by the system. Especially for the task of model transformation, a generic representation of models is advantageous as the necessary transformations have just to be implemented for the generic representation.

Such a generic representation is called a *generic metamodel*. A generic metamodel should be able to represent models originally represented in different metamodels (or modeling languages) in a generic way without loosing detailed information about the semantics of the model.

First implementations of model management systems used rather simple graph representations of models, e.g., Rondo [13]. Although the graph-based approach might allow an efficient implementation of operations which do not rely on a detailed representation of the models (such as schema matching), it is more difficult to implement more complex operations (such as model transformation or schema integration).

Schema integration approaches used rather abstract generic metamodels. More detailed generic metamodels have been used for model transformation. In [1], the authors describe a metamodel consisting of "superclasses" of the modeling constructs in the native metamodels. The transition between this internal representation and a native metamodel is described as a set of patterns. This induces the concept of a *supermodel* which is the union of patterns defined for any supported native metamodel.

A detailed generic metamodel called *GeRoMe* (Generic Role based Metamodel) is proposed in [10].

GeRoMe employs the *role based* modeling approach in which an object is regarded as playing roles in collaborations with other objects. This allows to describe the properties of model elements as accurately as possible while using only metaclasses and roles from a relatively small set. Therefore, *GeRoMe* provides a generic, yet detailed representation of data models originally represented in different metamodels.

Schema Integration

In model management, the *Merge* operator addresses the problem of schema integration, i.e., generating a merged model given two input models and a mapping between them. The merged model should contain all the information contained in the input models and the mapping. In this context, a mapping is not just a simple set of correspondences between model elements; it might have itself a complex structure and is therefore often regarded also as a *mapping model*. A mapping model is necessary because the models to be merged also have complex structures, which usually do not correspond to each other; the mapping model then acts as a "bridge" to connect these heterogeneous structures.

These structural heterogeneities are one class of conflicts which have to be solved in schema integration. Other types of conflicts are semantic conflicts (model elements describe overlapping sets of objects), descriptive conflicts (the same elements are described by different sets of properties; this includes also name conflicts), and heterogeneity conflicts (models are described in different modeling languages) [15]. The resolution of these conflicts is the main problem in schema integration.

The problem of schema integration has been addressed for various metamodels, such as variants of the ER metamodel [15] or generic metamodels.

Mappings

Depending on the application area, such as data translation, query translation or model merging, schema mappings come in different flavors. One can distinguish between correspondences, also called morphisms, extensional and intensional mappings. Correspondences usually do not have a formal semantics but only state informally that the respective model elements are similar. Morphisms are often – as the result of a schema matching operation [14] – the starting point for specifying more formal mappings. Intensional mappings are usually used for schema integration (see above) and are based on the *possible* instances of a schema. In contrast to intensional mappings, extensional mappings refer to the actual instances of a schema.

Extensional mappings are defined as local-as-view (LAV), global-as-view (GAV), source-to-target tuple generating dependencies (s-t tgds), [12], second order tuple generating dependencies (SO tgds) [8], or similar formalisms. Each of these classes has certain advantages and disadvantages when it comes to properties such as composability, invertibility or execution of the mappings.

Composition is an important sub-problem when dealing with extensional mappings. In general, the problem of composing mappings has the following definition: given a mapping M_{12} from model S_1 to model S_2, and a mapping M_{23} from model S_2 to model S_3, derive a mapping M_{13} from model S_1 to model S_3 that is equivalent to the successive application of M_{12} and M_{23} [8].

Fagin et al. [8] explored the properties of the composition of schema mappings specified by a finite set of s-t tgds. They proved that the language of s-t tgds is not closed under composition. To ameliorate the problem, they introduced the class of SO tgds which are closed under composition.

Model Management Systems

In the recent years, several prototype systems related to model management have been developed. Many of them focus only on some particular aspects of model management whereas only a few try to address a broader range of model management operators. The systems Rondo [13] and GeRoMeSuite [11] aim at providing a complete set of model management operators and are not restricted to particular modeling languages. Clio [9] focuses especially on mappings between XML and relational databases, the generation, and composition of these mappings. COMA++ [2] provides schema matching functionality.

Key Applications

Model management can be applied in scenarios, in which the management of complex data models is necessary. For example, data warehouses (DWs) are one application area for model management systems [6]. For integrating a new data source into the DW, a *Match* operator could be used to identify the similarities between the source schema and the DW schema. If the DW schema has not yet been created, a *Merge* operator can be used to generate it from several source schemas. The composition of mappings can be used after a source schema *S* has been evolved to a new version *S*′: if a mapping from *S*′ to *S* is known (either given by the schema evolution operation, or computed by a *Match* operator), one can compose this mapping with the original mapping between the source schema *S* and the integrated schema *T* to get a mapping from the new version of the source schema *S*′ to the integrated schema *T*.

Such applications of model management operators are also possible for other integrated information systems. For example, web services or e-business systems often have to take a message in XML format and store it in some relational data store for further processing. To implement this data transformation, a mapping between the XML schema and the schema of the relational database has to be defined. Such tasks are already supported by commercial products (e.g., Microsoft BizTalk, Altova MapForce). These products already use some model management operators (e.g., simple *Match* operators to simplify the task of mapping definition), but could further benefit from more powerful techniques for mapping generation. Clio started as a research prototype for mapping generation between XML and relational schemas, results have now been integrated into the IBM Information Integration platform.

Another application area for model management is the design and development of software applications. Model transformation is an inherent problem in software development, models have to be transformed into new metamodels, and then interoperability between the original model and the transformed model has to be implemented. A classical example is the transformation of an object-oriented data model of an application to a relational database schema, for which data access objects later have to be implemented to enable the synchronization between the data in the database and in the application. Such tasks are supported by frameworks (e.g., ADO.NET Entity Framework or the Entity Beans in Java 2 Enterprise Edition).

Future Directions

Mappings will become more and more important for MMS in the future [5]. The initial vision of model management, which considered mappings as rather simple correspondences attached with some complex expressions which contain the semantics of the mapping, was too limited. Mappings are in practice very complex and therefore, a rich mapping language is required in a MMS. The MMS must also be able to reason about the mappings. Furthermore, the MMS should not only enable the definition of a mapping, but support also its application; for example, using it for the integration of two schemas or for data transformation between two data stores. In addition, complex processes involving mappings (such as ETL process in data warehouses) have also to be considered.

Cross-references

- ▶ Data Integration
- ▶ Meta Data Repository
- ▶ Meta Model
- ▶ Schema Matching
- ▶ Schema Mapping
- ▶ Schema Mapping Composition

Recommended Reading

1. Atzeni P. and Torlone R. Management of Multiple Models in an Extensible Database Design Tool. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996, pp. 79–95.
2. Aumueller D., Do H.H., Massmann S., and Rahm E. Schema and ontology matching with COMA++. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 906–908.
3. Batini C., Lenzerini M., and Navathe S.B. A comparative analysis of methodologies for database schema integration. ACM Comput. Surv., 18(4):323–364, 1986.
4. Bernstein P.A., Halevy A.Y., and Pottinger R. A Vision for Management of Complex Models. ACM SIGMOD Rec., 29(4):55–63, 2000.
5. Bernstein P.A. and Melnik S. Model Management 2.0: Manipulating Richer Mappings. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1–12.
6. Bernstein P.A. and Rahm E. Data Warehousing Scenarios for Model Management. In Proc. 19th Int. Conf. on Conceptual Modeling, 2000, pp. 1–15.
7. D.R. Dolk Model Management and Structured Modeling: The Role of an Information Resource Dictionary System. Commun. ACM, 31(6), 1988.
8. Fagin R., Kolaitis P.G., Popa L., and Tan W.C. Composing schema mappings: Second-order dependencies to the rescue. ACM Trans. Database Syst., 30(4):994–1055, 2005.
9. Hernández M.A., Miller R.J., and Haas L.M. Clio: A Semi-Automatic Tool for Schema Mapping. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 607.
10. Kensche D., Quix C., Chatti M.A., and Jarke M. GeRoMe: A Generic Role Based Metamodel for Model Management. J. Data Semant., VIII:82–117, 2007.
11. Kensche D., Quix C., Li X., and Li Y. GeRoMeSuite: A System for Holistic Generic Model Management. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 1322–1325.

12. Lenzerini M. Data Integration: A Theoretical Perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
13. Melnik S., Rahm E., and Bernstein P.A. Rondo: A Programming Platform for Generic Model Management. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 193–204.
14. Rahm E. and Bernstein P.A. A Survey of Approaches to Automatic Schema Matching. VLDB J., 10(4):334–350, 2001.
15. Spaccapietra S. and Parent C. View Integration: A Step Forward in Solving Structural Conflicts. IEEE Trans. Knowl. Data Eng., 6(2):258–274, 1994.

Model-based Querying in Sensor Networks

Amol Deshpande[1], Carlos Guestrin[2], Sam Madden[3]
[1]University of Maryland, College Park, MD, USA
[2]Carnegie Mellon University, Pittsburgh, PA, USA
[3]Massachussetts Institute of Technology, Cambridge, MA, USA

Synonyms

Approximate querying; Model-driven data acquisition

Definition

The data generated by sensor networks or other distributed measurement infrastructures is typically incomplete, imprecise, and often erroneous, such that it is not an accurate representation of physical reality. To map raw sensor readings onto physical reality, a mathematical description, a *model*, of the underlying system or process is required to complement the sensor data. Models can help provide more robust interpretations of sensor readings: by accounting for spatial or temporal biases in the observed data, by identifying sensors that are providing faulty data, by extrapolating the values of missing sensor data, or by inferring hidden variables that may not be directly observable. Models also offer a principled approach to predict future states of a system. Finally, since models incorporate spatio-temporal correlations in the environment (which tend to be very strong in many monitoring applications), they lead to significantly more energy-efficient query execution – by exploiting such attribute correlations, it is often possible to use a small set of observations to provide approximations of the values of a large number of attributes.

Model-based querying over a sensor network consists of two components: (i) identifying and/or building a model for a given sensor network, and (ii) executing declarative queries against a sensor network that has been augmented with such a model (these steps may happen serially or concurrently). The queries may be on future or hidden states of the system, and are posed in a declarative SQL-like language. Since the cost of acquiring sensor readings from the sensor nodes is the dominant cost in these scenarios, the optimization goal typically is to minimize the total data acquisition cost.

Historical Background

Statistical and probabilistic models have been a mainstay in the scientific and engineering communities for a long time, and are commonly used for a variety of reasons, from simple pre-processing tasks for removing noise (e.g., using Kalman Filters) to complex analysis tasks for prediction purposes (e.g., to predict weather or traffic flow). Standard books on machine learning and statistics should be consulted for more details (e.g., Cowell [3], Russell and Norvig [14]).

The first work to combine models, declarative SQL-like queries and live data acquisition in sensor networks was the BBQ System [7,6]. The authors proposed a general architecture for model-based querying, and posed the optimization problem of selecting the best sensor readings to acquire to satisfy a user query (which can be seen as a generalization of the *value of information problem* [14]). The authors proposed several algorithms for solving this optimization problem; they also evaluated the approach on a several real-world sensor network datasets, and demonstrated that model-based querying can provide high-fidelity representation of the real phenomena and leads to significant performance gains versus traditional data acquisition techniques. Several works since then have considerably expanded upon the basic idea, including development of sophisticated algorithms for data acquisition [12,13], more complex query types [15], and integration into a relational database system [8,11].

The querying aspect of this problem has many similarities to the problem of approximate query processing in database systems, which often uses model-like *synopses*. For example, the AQUA project [1] proposed a number of sampling-based synopses that can provide approximate answers to a variety of queries using a fraction of the total data in a database. As with BBQ, such answers typically include tight bounds on the correctness of answers. AQUA, however, is designed to work in an environment where it is possible to generate an independent random sample of data (something that is quite tricky to do in sensor networks, as losses are correlated and communicating random samples may require the participation of a large part of the network). AQUA also does not exploit correlations, which means that it lacks the *predictive* power of representations based on probabilistic models. Deshpande et al. [4] and Getoor et al. [9] proposed exploiting data correlations through use of graphical modeling techniques for approximate query processing, but, unlike BBQ, neither provide any guarantees on the answers returned. Furthermore, the optimization goal of approximate query processing is typically not to minimize the data acquisition cost, rather it is minimizing the size of the synopsis, while maintaining reasonable accuracy.

Foundations

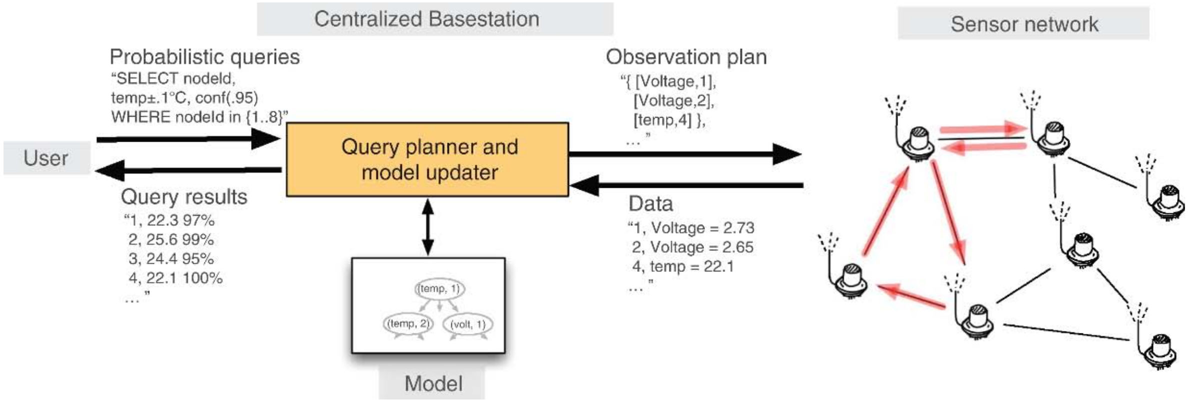
Figure 1 shows the most common architecture of a model-based querying system (adapted from the architecture of the BBQ system). The model itself is located at a centralized, Internet-connected basestation, which also interacts with the user. The user may issue either continuous or ad hoc queries against the sensor network, using a declarative SQL-like language. The key module in this architecture is the *query planner and model updater*, which is in charge of maintaining the model and answering the user queries (possibly by acquiring more data from the underlying sensor network). The following sections elaborate on the various components of such a system.

Model

A model is essentially a simplified representation of the underlying system or the process, and describes how various attributes of the system interact with each other, and how they evolve over time. Hence, the exact form of the model is heavily dependent on the system being modeled, and an astounding range of models have been developed over the years for different environments. For ease of exposition, the rest of this entry focuses on a dynamic model similar to the one used in the BBQ system.

Let $X_1,\ldots,X_n$ denote the (n) attributes of interest in the sensor network. Further, let X_i^t denote the value of X_i at time t (assuming that time is discrete). At any time t, a subset of these attributes may be observed and communicated to the basestation; here, the observations at time t are denoted by $\mathbf{o}^t$ (note that hidden variables can never be observed). The attributes typically correspond to the properties being monitored by the sensor nodes (e.g., *temperature* on sensor number 5, *voltage* on sensor number 8). However, more generally, they may be *hidden variables* that are of interest, but cannot be directly observed. For example, it may be useful to model and query a hidden boolean variable that denotes whether a sensor is faulty [11] – the value of this variable can be inferred using the model and the actual observations from the sensor.

The model encodes the spatial and temporal relationships between these attributes of interest. At any time t, the model provides us with a posterior *probability density function* (pdf), $p(X_1^t,\ldots,X_n^t|\mathbf{o}^{1\ldots(t-1)})$, assigning a probability for each possible assignment to the attributes at time t given the observations



Model-based Querying in Sensor Networks. Figure 1. Architecture of a model-based querying system (adapted from BBQ [7,6]).

made so far. Such a joint distribution can capture all the spatial correlations between the attributes; more compact representations like Bayesian networks can be used instead as well.

To model the temporal correlations, it is common to make a *Markov* assumption; given the values of *all* attributes at time *t*, one assumes that the values of the attributes at time $t + 1$ are independent of those for any time earlier than *t*. This assumption leads to a simple model for a dynamic system where the dynamics are summarized by a conditional density called the *transition model*, $p(X_1^{t+1},\ldots,X_n^{t+1}|X_1^t,\ldots,X_n^t)$. Using a transition model, one can compute $p(X_1^{t+1},\ldots,X_n^{t+1}|\mathbf{o}^{1\ldots t})$ from $p(X_1^t,\ldots,X_n^t|\mathbf{o}^{1\ldots t})$ using standard probabilistic procedures. Different transition models may be used for different time periods (e.g., hour of day, day of week, season, etc.) to model the differences in the way the attributes evolve at different times.

Learning the model. Typically in probabilistic modeling, a *class* of models is chosen (usually with input from a *domain expert*), and learning techniques are then used to pick the best model in the class. Model parameters are typically learned from training data, but can also be directly inferred if the behavior of the underlying physical process is well-understood. In BBQ, the model was learned from historical data, which consisted of readings from all of the monitored attributes over some period of time.

Updating the model. Given the formulation above, model updates are fairly straightforward. When a new set of observations arrives (say $\mathbf{o}^t$), it can be incorporated into the model by conditioning on the observations to compute new distributions (i.e., by computing $p(X_1^t,\ldots,X_n^t|\mathbf{o}^{1\ldots t})$ from $p(X_1^t,\ldots,X_n^t|\mathbf{o}^{1\ldots(t-1)})$. Similarly, as time advances, the transition model is used to compute the new distribution for time $t + 1$ (i.e., $p(X_1^{t+1},\ldots,X_n^{t+1}|\mathbf{o}^{1\ldots t})$ is computed from $p(X_1^t,\ldots,X_n^t|\mathbf{o}^{1\ldots t})$ and $p(X_1^{t+1},\ldots,X_n^{t+1}|X_1^t\ldots X_n^t)$).

Query Planning and Execution

User queries are typically posed in a declarative SQL-like language that may be augmented with constructs that allow users to specify the approximation that the user is willing to tolerate, and the desired confidence in the answer. For example, the user may ask the system to report the temperature readings at all sensors within ± 0.5, with confidence 95%. For many applications (e.g., building temperature control), such approximate answers may be more than sufficient. Tolerance for such approximations along with the correlations encoded by the model can lead to significant energy savings in answering such queries.

Answering queries probabilistically based on a pdf over the query attributes is conceptually straightforward; to illustrate this process, consider two types of queries (here, assume that all queries are posed over attributes $X_1^t,\ldots,X_n^t$, and the corresponding pdf is given by $P(X_1^t,\ldots,X_n^t)$):

Value query. A value query [6] computes an approximation of the values of the attributes to within $\pm\,\varepsilon$ of the true value, with confidence at least $1 - \delta$. Answering such a query involves computing the expected value of each of the attribute, μ_i^t, using standard probability theory. These μ_i^t's will be the reported values. The pdf can then be used again to compute the probability that X_i^t is within ε from the mean, $P(X_i^t \in [\mu_i^t - \varepsilon, \mu_i^t + \varepsilon])$. If all of these probabilities meet or exceed user specified confidence threshold, then the requested readings can be directly reported as the means μ_i^t. If the model's confidence is too low, additional readings must be acquired before answering the query (see below).

Max query. Consider an *entity* version of this query [2] where the user wants to know the identity of the sensor reporting the maximum value. A naive approach to answering this query is to compute, for each sensor, the probability that its value is the maximum. If the maximum of these probabilities is above $1 - \delta$, then an answer can be returned immediately; otherwise, more readings must be acquired. Although conceptually simple, computing the probability that a given sensor is reporting the maximum value is non-trivial, and requires complex integration that may be computationally infeasible [15].

If the model is not able to provide sufficient confidence in the answer, the system must acquire more readings from the sensor network, to bring the model's confidence up to the user specified threshold. Suppose the system observes a set of attributes $\mathcal{O} \subset \{X_1,\ldots,X_n\}$. After incorporating these observations into the model and recomputing the answer, typically the confidence in the (new) answer will be higher (this is not always true). The new confidence will typically depend on the actual observed values. Let $R(\mathcal{O})$ denote the *expected* confidence in the answer after observing

$\mathcal{O}$. Then, the optimization problem of deciding which attributes to observe can be stated as follows:

$$\begin{aligned} &\text{minimize}_{\mathcal{O}\subseteq\{1,\ldots,n\}} && C(\mathcal{O}),\\ &\text{such that} && R(\mathcal{O}) \geq 1-\delta. \end{aligned} \tag{1}$$

where $C(\mathcal{O})$ denotes the *data acquisition cost* of observing the values of attributes in $\mathcal{O}$.

This optimization problem combines three problems that are known to be intractable, making it very hard to solve it in general:

1. Answering queries using a pdf: as mentioned above, this can involve complex numerical integration even for simple queries such as max.
2. hoosing the minimum set of sensor readings to acquire to satisfy the query (this is similar to the classic *value of information problem* [14,12,15,10]).
3. Finding the optimal way to collect a required set of sensor readings from the sensor network that minimizes the total communication cost. Meliou et al. [13] present several approximation algorithms for this NP-Hard problem.

Example

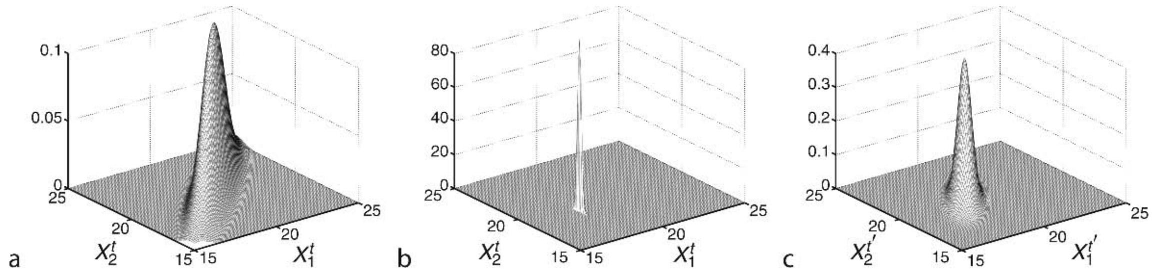
Figure 2 illustrates the query answering process using a simple example, where the model takes the form of time-varying *bivariate Gaussian (normal)* distribution over two attributes, X_1 and X_2. This was the basic model used in the BBQ system. A bivariate Gaussian is the natural extension of the familiar unidimensional normal probability density function (pdf), known as the "bell curve". Just as with its one-dimensional counterpart, a bivariate Gaussian can be expressed as a function of two parameters: a length-2 vector of means, μ, and a 2 × 2 matrix of covariances, Σ. Figure 2a shows a three-dimensional rendering of a Gaussian over the two attributes at time t, X_1^t and $\mathrm{X}_2^{\mathrm{t}}$ the z axis represents the *joint density* that $X_2^t = x$ and $\mathrm{X}_1^{\mathrm{t}} = \mathrm{y}$).

Now, consider a *value query* over this model, posed at time t, which asks for the values of X_1^t and X_2^t, within $\pm\,\epsilon$, with confidence $1-\delta$. The reported values in this case would be the means (μ), and the confidence can be computed easily using Σ (details can be found in [6]). Considering the high initial covariance, it is unlikely that the Gaussian in Figure 2a can achieve the required confidence. Suppose the system decides to observe $X_1{}^t$. Figure 2b shows the result of incorporating this observation into the model. Note that not only does the spread of $X_1{}^t$ reduces to near zero, because of the high correlation between $X_1{}^t$ and $X_2{}^t$, the variance of $X_2{}^t$ also reduces dramatically, allowing the system to answer the query with required confidence.

Then, after some time has passed, the belief about the values of X_1 and X_2 (at time $t' > t$) will be "spread out", again providing a high-variance Gaussian over two attributes, although both the mean and variance may have shifted from their initial values, as shown in Fig. 2c.

Key Applications

Model-based querying systems like BBQ, that exploit statistical modeling techniques and optimize the utilization of a network of resource constrained devices could have significant impact in a number of application domains, ranging from control and automation in buildings [16] to highway traffic monitoring. Integrating a model into the data acquisition process can significantly improve data quality and reduce data



Model-based Querying in Sensor Networks. Figure 2. Example of Gaussians: (a) 3D plot of a 2D Gaussian with high covariance; (b) the resulting Gaussian after a particular value of X_1^t has been observed (because of measurement noise, there might still be some uncertainty about the true value of X_1^t); (c) the uncertainty about X_1 and X_2 increases as time advances to t'.

uncertainty. The ability to query over missing, future or hidden states of the system will prove essential in many applications where sensor failures are common (e.g., highway traffic monitoring) or where direct observation of the variables of interest is not feasible. Finally, model-based querying has the potential to significantly reduce the cost of data acquisition, and thus can improve the life of a resource-constrained measurement infrastructure (e.g., battery-powered wireless sensor networks) manyfold.

Future Directions

Model-based querying is a new and exciting research area with many open challenges that are bound to become more important with the increasingly widespread use of models for managing sensor data. The two most important challenges are dealing with a wide variety of models that may be used in practice, and designing algorithms for query processing and data acquisition; these are discussed briefly below:

Model selection and training. The choice of model affects many aspects of model-based querying, most importantly the accuracy of the answers and the confidence bounds that can be provided with them. The problem of selecting the right model class has been widely studied [3,14] but can be difficult in some applications. Furthermore, developing a new system for each different model is not feasible. Ideally, using a new model should involve little to no effort on the part of user. Given a large variety of models that may be applicable in various different scenarios, this may turn out to be a tremendous challenge.

Algorithms for query answering and data acquisition. Irrespective of the model selected, when and how to acquire data in response to a user query raises many hard research challenges. As discussed above, this problem combines three very hard problems, and designing general-purpose algorithms that can work across the spectrum of different possible model remains an open problem.

See Deshpande et al. [5] for a more elaborate discussion of the challenges in model-based querying.

Cross-references

- ▶ Ad-Hoc Queries in Sensor Networks
- ▶ Approximate Query Processing
- ▶ Continuous Queries in Sensor Networks
- ▶ Data Acquisition and Dissemination in Sensor Networks
- ▶ Data Uncertainty Management in Sensor Networks
- ▶ Models

Recommended Reading

1. Acharya S., Gibbons P.B., Poosala V., and Ramaswamy S. Join synopses for approximate query answering. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 275–286.
2. Cheng R., Kalashnikov D.V., and Prabhakar S. Evaluating probabilistic queries over imprecise data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 551–562.
3. Cowell R., Dawid P., Lauritzen S., and Spiegelhalter D. Probabilistic Networks and Expert Systems. Spinger, New York, 1999.
4. Deshpande A., Garofalakis M., and Rastogi R. Independence is good: dependency-based histogram synopses for high-dimensional data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 199–210.
5. Deshpande A., Guestrin C., and Madden S. Using Probabilistic Models for Data Management in Acquisitional Environments. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 317–328.
6. Deshpande A., Guestrin C., Madden S., Hellerstein J., and Hong W. Model-Driven Approximate Querying in Sensor Networks. VLDB J., 14(4):417–443, 2005.
7. Deshpande A., Guestrin C., Madden S., Hellerstein J.M., and Hong W. Model-driven Data Acquisition in Sensor Networks. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 588–599.
8. Deshpande A. and Madden S. MauveDB: supporting model-based user views in database systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 73–84.
9. Getoor L., Taskar B., and Koller D. Selectivity estimation using probabilistic models. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 461–472.
10. Goel A., Guha S., and Munagala K. Asking the right questions: model-driven optimization using probes. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 203–212.
11. Kanagal B. and Deshpande A. Online Filtering, Smoothing and Probabilistic Modeling of Streaming data. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 1160–1169.
12. Krause A., Guestrin C., Gupta A., and Kleinberg J. Near-optimal sensor placements: maximizing information while minimizing communication cost. In Proc. 5th Int. Symp. Inf. Proc. in Sensor Networks, 2006, pp. 2–10.
13. Meliou A., Chu D., Hellerstein J., Guestrin C., and Hong W. Data gathering tours in sensor networks. In Proc. 5th Int. Symp. Inf. Proc. in Sensor Networks, 2006, pp. 43–50.
14. Russell S. and Norvig P. Artificial Intelligence: A Modern Approach. Prentice Hall, 1994.
15. Silberstein A., Braynard R., Ellis C., Munagala K., and Yang J. A Sampling-Based approach to Optimizing Top-k Queries in Sensor networks. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 68.
16. Singhvi V., Krause A., Guestrin C., Garrett Jr J., and Matthews H. Intelligent light control using sensor networks. In Proc. 3rd Int. Conf. on Embedded Networked Sensor Systems, 2005, pp. 218–229.

Model-driven Data Acquisition

► Model-Based Querying in Sensor Networks

Module

► Snippet

MOF

► Meta Object Facility

Molecular Interaction Graphs

► Biological Networks

Moment

► Chronon
► Time Instant

Monitoring

► Auditing and Forensic Analysis

Monitoring of Real-Time Logic Expressions

► Event Detection

Monotone Constraints

Carson Kai-Sang Leung
University of Manitoba, Winnipeg, MB, Canada

Synonyms

Monotonic constraints

Definition

A constraint C is *monotone* if and only if for all itemsets S and S':

$$\text{if } S \supseteq S' \text{ and } S \text{ violates } C, \text{ then } S' \text{ violates } C.$$

Key Points

Monotone constraints [1–3] possess the following property. If an itemset S violates a monotone constraint C, then any of its subsets also violates C. Equivalently, all supersets of an itemset satisfying a monotone constraint C also satisfy C (i.e., C is upward closed). By exploiting this property, monotone constraints can be used for reducing computation in frequent itemset mining with constraints. As frequent itemset mining with constraints aims to find frequent itemsets that satisfy the constraints, if an itemset S satisfies a monotone constraint C, no further constraint checking needs to be applied to any superset of S because all supersets of S are guaranteed to satisfy C. Examples of monotone constraints include $min(S.Price) \leq \$30$, which expresses that the minimum price of all items in an itemset S is at most \$30. Note that, if the minimum price of all items in S is at most \$30, adding more items to S would not increase its minimum price (i.e., supersets of S would also satisfy such a monotone constraint).

Cross-references

► Frequent Itemset Mining with Constraints

Recommended Reading

1. Brin S., Motwani R., and Silverstein C. Beyond market baskets: generalizing association rules to correlations. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 265–276.
2. Grahne G., Lakshmanan L.V.S., and Wang X. Efficient mining of constrained correlated sets. In Proc. 16th Int. Conf. on Data Engineering, 2000, pp. 512–521.
3. Pei J. and Han J. Can we push more constraints into frequent pattern mining? In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 350–354.

Monotonic Constraints

► Monotone Constraints

Monotonicity Property

► Apriori Property and Breadth-First Search Algorithms

Motion Graphics

► Dynamic Graphics

Moving Object

RALF HARTMUT GÜTING
University of Hagen, Hagen, Germany

Synonyms

Time dependent geometry

Definition

A moving object is essentially a time dependent geometry. Moving objects are the entities represented and queried in moving objects databases.

Key Points

The term emphasizes the fact that geometries may change continuously (whereas earlier work on spatio-temporal databases allowed only discrete changes, e.g., of land parcels). One can distinguish between moving objects for which only the time dependent position is of interest and those for which also shape and extent are relevant and may change over time. The first can be characterized as *moving points*, the second as *moving regions*. For example, moving points could represent people, vehicles (such as cars, trucks, ships or air planes), or animals. Moving regions could be hurricanes, forest fires, spread of epidemic diseases etc. Moving point data may be captured by GPS devices or RFID tags; moving region data may result from processing sequences of satellite images, for example. Moving points and moving regions can be made available as data types in suitable type systems; such a design can be found in [1]. Such an environment may have further "moving" data types (e.g., *moving lines*).

Cross-references

► Moving Objects Databases and Tracking

Recommended Reading

1. Güting R.H., Böhlen M.H., Erwig M., Jensen C.S., Lorentzos N.A., Schneider M., and Vazirgiannis M. A foundation for representing and querying moving objects in databases. ACM Trans. Database Syst., 25:1–42, 2000.

Moving Object Trajectories

► Spatio-Temporal Trajectories

Moving Objects Databases and Tracking

RALF HARTMUT GÜTING
University of Hagen, Hagen, Germany

Synonyms

Spatio-temporal databases; trajectory databases

Definition

Moving objects database systems provide concepts in their data model and data structures in the implementation to represent moving objects, i.e., continuously changing geometries. Two important abstractions are *moving point*, representing an entity for which only the time dependent position is of interest, and *moving region*, representing an entity for which also the time dependent shape and extent is relevant. Examples of moving points are cars, trucks, air planes, ships, mobile phone users, RFID equipped goods, or polar bears; examples of moving regions are forest fires, deforestation of the Amazon rain forest, oil spills in the sea, armies, epidemic diseases, hurricanes, and so forth.

There are two flavors of such databases. The first represents information about a set of currently moving objects. Basically one is interested in efficiently maintaining their location information and asking queries about the current and expected near future positions and relationships between objects. In this case, no information about histories of movement is kept. This is sometimes also called a *tracking database*.

The second represents complete histories of movements. The goal in the design of query languages for moving objects is to be able to ask any kind of questions about such movements, perform analyses, derive information, in a way as simple and elegant as possible. The underlying system must support efficient execution of such analyses. This view is associated with the term *moving objects database*, sometimes also called *trajectory database*.

Historical Background

The field of moving objects databases came into being in the late 1990s mainly by two parallel developments. First, a model was developed [11,16,17] that allows one to keep track in a database of a set of time dependent locations, e.g., to represent vehicles. The authors observed that one should store in a database not the locations directly, which would require high update rates, but rather a motion vector, representing an object's expected position over time. An update to the database is needed only when the deviation between the expected position and the real position exceeds some threshold. At the same time this concept introduces an inherent, but bounded uncertainty about an object's real location. The model was formalized introducing the concept of a *dynamic attribute*. This is an attribute of a normal data type which changes implicitly over time. This implies that results of queries over such attributes also change implicitly over time. A related query language FTL (future temporal logic) was introduced that allows one to specify time dependent relationships between expected positions of moving objects.

Second, the European project CHOROCHRONOS set out to integrate concepts from spatial and temporal databases. In this case, one represents in a database time-dependent geometries of various kinds such as points, lines, or regions. Earlier work on spatio-temporal databases had generally admitted only discrete changes. This restriction was dropped and continuously changing geometries were considered. A model was developed based on the idea of *spatio-temporal data types* to represent histories of continuously changing geometries [2,4–6]. The model offers data types such as *moving point* or *moving region* together with a comprehensive set of operations. For example, there are operations to compute the projection of a moving point into the plane, yielding a *line* value, or to compute the distance between a moving point and a moving region, returning a time dependent real number, or *moving real*, for short. Such data types can be embedded into a DBMS data model as attribute types and can be implemented as an extension package.

A second approach to data modeling for moving object histories was pursued in CHOROCHRONOS. Here the constraint model was applied to the representation of moving objects [10] and a prototype called Dedale was implemented. Constraint databases can represent geometries in *n*-dimensional spaces; since moving objects exist in 3D (2D + time) or 4D (3D + time) spaces, they can be handled by this approach. Several researchers outside CHOROCHRONOS also contributed to the development of constraint-based models for moving objects.

Foundations

Modeling and Querying Current Movement (Tracking)

Consider first moving objects databases for current and near future movement, or tracking databases. Sets of moving entities might be taxi-cabs in a city, trucks of a logistics company, or military vehicles in a military application. Possible queries might be:

- Retrieve the three free cabs closest to Cottle Road 52 (a passenger request position).
- Which trucks are within 10 km of truck T70 (which needs assistance)?
- Retrieve the friendly helicopters that will arrive in the valley within the next 15 min and then stay in the valley for at least 10 min.

Statically, the positions of a fleet of taxi-cabs, for example, could be easily represented in a relation

```
taxi-cabs(id: int, pos: point)
```

Unfortunately this representation needs frequent updates to keep the deviation between real position and position in the database small. This is not feasible for large sets of moving objects.

The MOST (moving objects spatio-temporal) data model [11,16], discussed in this section, stores instead of absolute positions a motion vector which represents a position as a linear function of time. This defines an expected position for a moving object. The distance between the expected position and the real position is called the *deviation*. Furthermore, a *distance threshold* is introduced and a kind of contract between a moving object and the database server managing its position is assumed. The contract requires that the moving object observes the deviation and sends an update to the server when it exceeds the threshold. Hence the threshold establishes a bound on the *uncertainty* about an object's real position.

A fundamental new concept in the MOST model is that of a *dynamic attribute*. Each attribute of an object class is classified to be either static or dynamic. A dynamic attribute is of a standard data type (e.g., *int*, *real*) within the DBMS conceptual model, but changes its value automatically over time. This means that queries involving such attributes also have time dependent results, even if time is not mentioned in the query and no updates to the database occur.

The MOST model assumes that time advances in discrete steps, so-called clock ticks. Hence time can be represented by integer values. For a data type to be eligible for use in a dynamic attribute, it is necessary that the type has a value 0 and an addition operation. This holds for numeric types but can be extended to types like *point*. A dynamic attribute A of type T is then internally represented by three subattributes $A.value$, $A.updatetime$, and $A.function$, where $A.value$ is of type T, $A.updatetime$ is a time value, and $A.function$ is a function f: $int \rightarrow T$ such that at time $t = 0$, $f(t) = 0$. The semantics of this representation is called the value of A at time t and defined as

$$value(A, t) = A.value + A.function(t - A.updatetime) \quad for\ t \geq A.updatetime$$

When attribute A is mentioned in a query, its dynamic value $value(A, t)$ is meant.

With dynamic attributes, for each clock tick one obtains a new state of the database, even without explicit updates. Such a sequence of states is called a database history. With each explicit update, all subsequent states change so that one obtains a new database history. One can now define different types of queries:

- An *instantaneous query* issued at a time t_0 is evaluated once on the database history starting at time $t_{0.}$
- A *continuous query* issued at time t_0 is (conceptually) reevaluated for each clock tick. Hence it is evaluated once on the database history starting at time t_0, then on the history starting at t_1, then on... t_2, and so forth.

Of course, reevaluating a continuous query on each clock tick is not feasible. Instead, the evaluation algorithm for such queries is executed only once and returns a time dependent result, in the form of a set of tuples with associated time stamps. A reevaluation is only necessary when explicit updates occur.

The query language associated with the MOST model is called FTL (future temporal logic). Here are a few example queries formulated in FTL.

1. Which trucks are within 10 km of truck T70?
```
RETRIEVE t
FROM trucks t, trucks s
WHERE s.id = ’T70’ ∧ dist(s, t) <= 10
```
   Here nothing special happens, yet, the result is time dependent.
2. Retrieve the helicopters that will arrive in the valley within the next 15 min and then stay in the valley for at least 10 min.
```
RETRIEVE h
FROM helicopters h
WHERE eventually_within_15
   (inside(h, Valley) ∧
   always_for_10 (inside(h, Valley)))
```
   Here *Valley* is a polygon object.

The general form of a query in FTL is
```
RETRIEVE <target-list> FROM <object
classes> WHERE <FTL-formula>
```
FTL formulas may contain special time dependent constructs, in particular:

- If f and g are formulas, then f **until** g and **nexttime** f are formulas

Informally, the meaning is that for a given database state s, f **until** g holds if there exists a future state s' on the database history such that g holds in state s' and for all states from s up to s', f holds. Similarly, **nexttime** f holds in state s_{i+1} if f holds in state s_i. Based on such temporal operators one can define bounded temporal operators like **eventually_within_c** g or **always_for_c** g as they occur in the second example query.

Modeling and Querying History of Movement

Now consider the problem of representing complete histories of movement in a database. The scope is also extended from point objects to more complex geometrical shapes.

The idea of the approach [4] presented in the following is to introduce spatio-temporal data types that encapsulate time dependent geometries with suitable operations. For moving objects, point and region appear to be most relevant, leading to data types *moving point* and *moving region*, respectively. The *moving point* type (*mpoint* for short) can represent entities such as vehicles, people, or animals moving around whereas the *moving region* type (*mregion*) can represent hurricanes, forest fires, armies, or flocks of animals, for example. Geometrically, values of spatio-temporal data types are embedded into a 3D space (2D + time) if objects move in the 2D plane, or in a 4D space if movement in the 3D space is modeled. Hence, a moving point and a moving region can be visualized as shown in Fig. 1

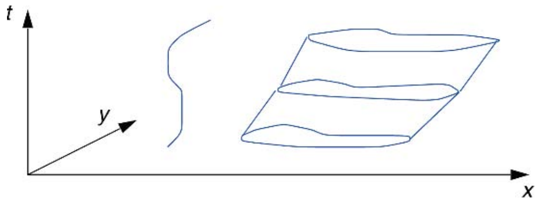
Data types may be embedded in the role of attribute types into a DBMS data model. For example, in a relational setting, there may be relations to represent the movements of air planes or storms:

```
flight (id: string, from: string, to:
  string, route: mpoint)
weather (id: string, kind: string,
  area: mregion)
```

The data types include suitable operations such as:

intersection: *mpoint* × *mregion* → *mpoint*
trajectory: *mpoint* → *line*
deftime: *mpoint* → *periods*
length: *line* → *real*

One discovers quickly that in addition to the main types of interest, *mpoint* and *mregion*, related spatial and temporal as well as other time-dependent types are needed. The operations above have the following meaning: **Intersection** returns the part of a moving point whenever it lies inside a moving region, which is a moving point (*mpoint*) again. **Trajectory** projects a moving point into the plane, yielding a *line* value (a curve in the 2D space). **Deftime** returns the set of time intervals when a moving point is defined, of a data type called *periods*. **Length** returns the length of a *line* value.



Moving Objects Databases and Tracking. Figure 1. A moving point and a moving region.

Given such operations, one may formulate queries:

"Find all flights from Düsseldorf that are longer than 5,000 km."

```
select id
from flights
where from = 'DUS' and length
  (trajectory (route)) > 5000
```

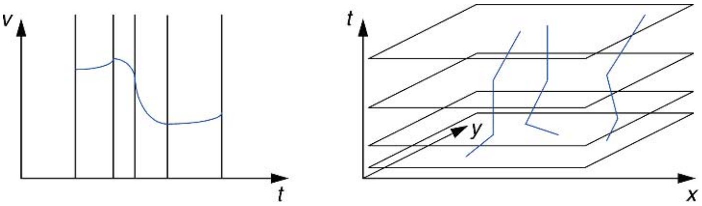
"At what times was flight BA488 within the snow storm with id S16?"

```
select deftime(intersection
  (f.route, w.area))
from flights as f, weather as w
where f.id = 'BA488' and w.id = 'S16'
```

Reference [4] develops the basic idea, discusses the distinction between *abstract models* (using infinite sets, and describing e.g., a moving region as a function from time into region values) and *discrete models* (selecting a suitable finite representation, e.g., describing a moving region as a polyhedron in the 3D space), and clarifies several fundamental questions related to this approach. A system of related data types and operations for moving objects is carefully defined in [6], emphasizing genericity, closure, and consistency. The semantics of these types is defined at the abstract level.

Implementation is based on the discrete model proposed in [5] using algorithms for the operations studied in [2]. The discrete model uses the so-called *sliced representation* as illustrated in Fig. 2. A temporal function value is represented as a time-ordered sequence of *units* where each unit has an associated time interval, and time intervals of different units are disjoint. Each unit is capable of representing a piece of the moving object by a "simple" function. Simple functions are linear functions for moving points or regions, and quadratic polynomials (or square roots of such) for moving reals, for example.

Within a database system, an extension module (data blade, cartridge, extender, etc.) can be provided offering implementations of such types and operations. The sliced representation is basically stored in an array of units. (It is a bit more complicated in case of variable size units as for a moving region, for example.) Because values of moving object types can be large and complex, the DBMS must provide suitable storage techniques for managing large objects. A large



Moving Objects Databases and Tracking. Figure 2. Sliced representation of a *moving*(*real*) and a *moving*(*points*) value.

part of this design has been implemented prototypically in the SECONDO extensible DBMS [1] which is available for download (see URL below).

Related Issues

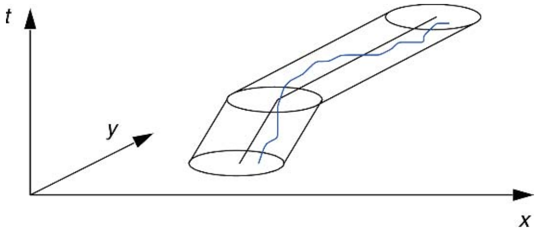
In this short closing section some issues related to moving objects databases are briefly discussed.

Uncertainty Locations of moving objects are most often captured using GPS devices at certain instants of time. This introduces an inherent uncertainty already for the sampled positions (due to some inaccuracy of the GPS device) and in particular for the periods of time between measurements [9]. Bounded uncertainty is also introduced due to a contract between location server and moving object, as discussed above. The MOST model includes concepts to deal with this uncertainty in querying [16,17]. For history of movement, one can consider uncertain trajectories based on an uncertainty threshold, resulting in a shape of a kind of slanted cylinder (Fig. 3). It is only known that the real position is somewhere inside this volume. Based on this model, Trajcevski et al. [15] have defined a set of predicates between a trajectory and a region in space taking uncertainty and aggregation over time into account.

Movement in Networks Whereas the basic case is free movement in the Euclidean plane, it is obvious that vehicles usually move on transport networks. There is a branch of research on network-constrained movement (e.g., [7,13]); there is also work on indexing network based movements. For network based movement, captured GPS positions have to be mapped to the transportation network; this is called map matching.

Spatio-Temporal Indexing A lot of research exists on indexing movement, both for expected near future movement and for history movement.

Query Processing for Continuous/Location Based Queries Continuous queries for moving objects have



Moving Objects Databases and Tracking. Figure 3. Geometry of an uncertain trajectory.

been studied in depth, for example, maintaining the result of nearest neighbor or range queries both for moving query and moving data objects (e.g., [12,14], see continuous monitoring of spatial queries).

Spatiotemporal Aggregation and Selectivity Estimation Another subfield of research in moving objects databases considers the problem of computing precisely or estimating the numbers of moving objects within certain areas in space and time – hence, of computing aggregates. For example, various index structures have been proposed to compute efficiently such aggregates. This is also related to the problem of performing selectivity estimation for spatio-temporal query processing.

Key Applications

Databases for querying current and near future movement like the MOST model described, are the foundation for location-based services. Service providers can keep track of the positions of mobile users and notify them of upcoming service offers even some time ahead. For example, gas stations, hotels, shopping centres, sightseeing spots, or hospitals in case of an emergency might be interesting services for car travelers.

Several applications need to keep track of the current positions of a large collection of moving objects, for example, logistics companies, parcel delivery services, taxi fleet management, public transport systems, air traffic control. Marine mammals or other animals are traced in biological applications. Obviously, the military is also interested in keeping track of fighting units in battlefield management.

Database systems for querying history of movement are needed for more complex analyses of recorded movements. For example, in air traffic control one may go back in time to any particular instant or period to analyze dangerous situations or even accidents. Logistics companies may analyze the paths taken by their delivery vehicles to determine whether optimizations are possible. Public transport systems in a city may be analyzed to understand reachability of any place in the city at different periods of the day. Movements of animals may be analyzed in biological studies. Historical modeling may represent movements of people or tribes and actually animate and query such movements over the centuries.

The data model of such systems offers not only moving point entities but also moving regions. Hence also developments of areas on the surface of the earth may be modeled and analyzed like the deforestation of the Amazon rain forest, the Ozone hole, development of forest fires or oil spills over time, and so forth.

Future Directions

Recent research in databases has often addressed specific query types like continuous range queries or nearest neighbor queries, and then focused on designing efficient algorithms for them. An integration of the many specific query types into complete language designs as presented in this entry is still lacking. Uncertainty may be treated more completely also in the approaches for querying history of movement. A seemless query language for querying past, present, and near future would also be desirable.

A text book covering the topics presented in this article in more detail is [8].

Experimental Results

Running times for queries of the BerlinMOD benchmark (see below), evaluated in the SECONDO system, can be found in [3].

Data Sets

A collection of links to data sets with real spatio-temporal data, partially assembled within the CHOROCHRONOS project mentioned above can be found at

http://dke.cti.gr/people/pfoser/data.html

Recently a benchmark data set is available, the so-called BerlinMOD benchmark [3]. It is based on a simulation of the movements of 2,000 people's vehicles in the city of Berlin, observed over 1 month (at scale factor 1). The benchmark contains a number of test queries. Test data are generated by the SECONDO system. The benchmark can be found and the relevant resources downloaded at

http://dna.fernuni-hagen.de/secondo/BerlinMOD/Berlin MOD.html

See also real and synthetic test data sets; this entry should include links to further test data generators.

URL to Code

SECONDO as a prototypical moving objects database system (for histories of movement, or trajectories) is available for download at

http://dna.fernuni-hagen.de/Secondo.html/

Cross-references

► Constraint Query Languages
► Continuous Monitoring of Spatial Queries
► Indexing Historical Spatio-Temporal Data
► Indexing of the Current and Near-Future Positions of Moving Objects
► Location-based Services
► Map Matching
► Real and Synthetic Test Datasets
► Spatial and Spatio-temporal Data Models and Languages
► Spatio-Temporal Data Mining
► Spatio-temporal Data Types
► Spatio-temporal Data Warehouses
► Spatio-temporal Trajectories

Recommended Reading

1. Almeida V.T., Güting R.H., and Behr T. Querying moving objects in SECONDO. In Proc. 7th Int. Conf. on Mobile Data Management, 2006, pp. 47–51.
2. Cotelo Lema J.A., Forlizzi L., Güting R.H., Nardelli E., and Schneider M. Algorithms for moving object databases. The Comput. J., 46(6):680–712, 2003.
3. Düntgen C., Behr T., and Güting R.H. BerlinMOD: a benchmark for moving object databases. Informatik-Report 340, Fernuniversität Hagen, 2007. Available at: http://dna.fernuni-hagen.de/secondo/BerlinMOD/BerlinMOD.pdf
4. Erwig M., Güting R.H., Schneider M., and Vazirgiannis M. Spatio-temporal data types: an approach to modeling and querying moving objects in databases. GeoInformatica, (3):265–291, 1999.

5. Forlizzi L., Güting R.H., Nardelli E., and Schneider M. A data model and data structures for moving objects databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 319–330.
6. Güting R.H., Böhlen M.H., Erwig M., Jensen C.S., Lorentzos N.A., Schneider M., and Vazirgiannis M. A foundation for representing and querying moving objects in databases. ACM Trans. Database Syst., 25:1–42, 2000.
7. Güting R.H., de Almeida V.T., and Ding Z. Modeling and querying moving objects in networks. VLDB J., 15(2):165–190, 2006.
8. Güting R.H., and Schneider M. Moving Objects Databases. Morgan Kaufmann Publishers, San Fransisco, CA, USA, 2005.
9. Pfoser D., and Jensen C.S. Capturing the uncertainty of moving-object representations. In Proc. 6th Int. Symp. Advances in Spatial Databases, 1999, pp. 111–131.
10. Rigaux P., Scholl M., Segoufin L., and Grumbach S. Building a constraint-based spatial database system: model, languages, and implementation. Inf. Syst., 28(6):563–595, 2003.
11. Sistla A.P., Wolfson O., Chamberlain S., and Dao S. Modeling and querying moving objects. In Proc. 13th Int. Conf. on Data Engineering, 1997, 422–432.
12. Song Z., and Roussopoulos N. K-nearest neighbor search for moving query point. In Proc. 7th Int. Symp. Advances in Spatial and Temporal Databases, 2001, pp. 79–96.
13. Speicys L., Jensen C.S., and Kligys A. Computational data modeling for network-constrained moving objects. In Proc. 11th ACM Int. Symp. on Advances in Geographic Inf. Syst., 2003, pp. 118–125.
14. Tao Y., and Papadias D. Spatial queries in dynamic environments. ACM Trans. Database Syst., 28(2):101–139, 2003.
15. Trajcevski G., Wolfson O., Hinrichs K., and Chamberlain S. Managing uncertainty in moving objects databases. ACM Trans. Database Syst., 29(3):463–507, 2004.
16. Wolfson O., Chamberlain S., Dao S., Jiang L., and Mendez G. Cost and imprecision in modeling the position of moving objects. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 588–596.
17. Wolfson O., Sistla A.P., Chamberlain S., and Yesha Y. Updating and querying databases that track mobile units. Distrib. Parallel Databases, 7:257–387, 1999.

Moving Objects Interpolation

► Spatiotemporal Interpolation Algorithms

Moving Span

► Variable Time Span

MRR

► MRR (Mean Reciprocal Rank)

MRR1

► MRR (Mean Reciprocal Rank)

MSN Data Management

► Mobile Sensor Network Data Management

Multi-Database

► Distributed Architecture

Multidatabases

► Distributed Database Systems

Multidimensional Clustering

► Physical Database Design for Relational Databases

Multidimensional Data Formats

Amarnath Gupta
University of California San Diego, La Jolla, CA, USA

Definition

The term "multidimensional data" is used in two different ways in data management. In the first sense, it refers to data aggregates created by different groupings of relational data for on-line analytical processing. In the second sense, the term refers to data that can be described as arrays over heterogeneous data types together with metadata to describe them.

Example: HDF (Hierarchical Data Format) and NetCDF (network Common Data Form) are well known multidimensional data formats used in scientific applications.

Key Points

The goal of a multidimensional data format is to enable random access to very large, very complex, heterogeneous data, such that the data is self describing, sharable, compact, extendible, and archivable. For example, a composite of 900 files from a seismic simulation has been organized in HDF5 format to create a terabyte-sized dataset. One can mix tables, images, small metadata, streams of data from instruments, and structured grids all in the same HDF file. While multidimensional file formats are very flexible, they present the challenge of storing such large datasets and providing concurrent, random access to any part of the data required by user queries. Design of novel index structures over these formats is an area of active research.

Cross-references

▶ Bitmap-based Index Structures
▶ Query Evaluation Techniques for Multidimensional Data
▶ Storage of Large Scale Multidimensional Data

Recommended Reading

1. Home page of the HDF group. Available at: http://hdf.ncsa.uiuc.edu/
2. Home page of the NetCDF group. Available at: http://www.unidata.ucar.edu/software/netcdf/
3. Wu K., Otoo E.J., and Shoshani A. “An efficient compression scheme for bitmap indices”. Technical Report LBNL-49626, Lawrence Berkeley National Laboratory, Berkeley, CA, 2002.

Multidimensional Database Management System

▶ Storage of Large Scale Multidimensional Data

Multi-dimensional Mapping

▶ Space Filling Curves
▶ Space-Filling Curves for Query Processing

Multidimensional Modeling

Torben Bach Pedersen
Aalborg University, Aalborg, Denmark

Synonyms

Dimensional modeling; Star schema modeling

Definition

Multidimensional modeling is the process of modeling the data in a universe of discourse using the modeling constructs provided by a multidimensional data model. Briefly, multidimensional models categorize data as being either *facts* with associated numerical *measures*, or as being *dimensions* that characterize the facts and are mostly textual. For example, in a retail business, *products* are sold to *customers* at certain *times* in certain *amounts* and at certain *prices*. A typical fact would be a *purchase*. Typical measures would be the amount and price of the purchase. Typical dimensions would be the location of the purchase, the type of product being purchased, and the time of the purchase. Queries then aggregate measure values over ranges of dimension values to produce results such as the total sales per month and product type.

Historical Background

Multidimensional databases do not have their origin in database technology, but stem from multidimensional matrix algebra, which has been used for (manual) data analyses since the late nineteenth century. During the late 1960s, two companies, IRI and Comshare, independently began the development of systems that later turned into multidimensional database systems. The IRI Express tool became very popular in the marketing analysis area in the late 1970s and early 1980s; it later turned into a market-leading OLAP tool and was acquired by Oracle. Concurrently, the Comshare system developed into System W, which was heavily used for financial planning, analysis, and reporting during the 1980s.

A concurrent development started in the early 1980s in the area of so-called *statistical data management* which focused on modeling and managing statistical data [1], initially within social science contexts such as census data. Many important concepts of multidimensional modeling such as *summarizability* (ensuring correct aggregate query results for complex data) have their roots in this area. An overview is found in [14].

In 1991, Arbor was formed with the specific purpose of creating "a multiuser, multidimensional database server," which resulted in the Essbase system. Arbor, now Hyperion, later licensed a basic version of Essbase to IBM for integration into DB2. It was Arbor and Codd who in 1993 coined the term OLAP [2].

Another significant development in the early 1990s was the advent of large *data warehouses* [6] for storing and analyzing massive amounts of enterprise data. Data warehouses are typically based on relational *star schemas* or *snowflake schemas*, an approach to implementing multidimensional databases using relational database technology. The 1996 version of [6] popularized the use of star schema modeling for data warehouses.

From the mid 1990s and beyond, the introduction of the "data cube" operator [4] sparked a considerable research interest in the field of modeling multidimensional databases for use in data warehouses and *on-line analytical processing* (OLAP).

In 1998, Microsoft shipped its MS OLAP Server, the first multidimensional system aimed at the mass market. This has lead to the current situation where multidimensional systems are increasingly becoming commodity products that are shipped at no extra cost together with leading relational database systems.

A more in-depth coverage of the history of multidimensional databases is available in the literature [16]. Surveys of multidimensional data models can also be found in the literature [12,17].

Foundations

First, an overview of the concept of a multidimensional *cube* is given, then dimensions, facts, and measures are covered in turn.

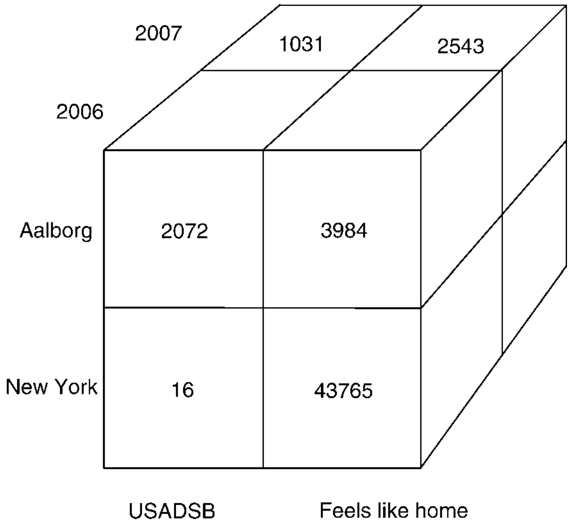
Data Cubes

Data cubes provide true multidimensionality. They generalize spreadsheets to any number of dimensions. In addition, hierarchies in dimensions and formulas are first-class, built-in concepts, meaning that these are supported without duplicating their definitions. A collection of related cubes is commonly referred to as a *multidimensional database* or a *multidimensional data warehouse*.

A dimensional cube for, e.g., CD sales can be obtained by including additional dimensions apart from just the album and the city where the album was sold. The most pertinent example of an additional dimension is a time dimension, but it is also possible to include other dimensions, e.g., an artist dimension that describes the artists associated with albums. In a cube, the combinations of a dimension value from each dimension define the *cells* of the cube. The actual sales counts are stored in the corresponding cells.

In a cube, dimensions are first-class concepts with associated domains, meaning that the addition of new dimension values is easily handled. Although the term "cube" implies three dimensions, a cube can have any number of dimensions. It turns out that most real-world cubes have 4–12 dimensions [6,16]. Although there is no theoretical limit to the number of dimensions, current tools often experience performance problems when the number of dimensions is more than 10–15. To better suggest the high number of dimensions, the term "hypercube" is often used instead of "cube."

Figure 1 illustrates a three-dimensional cube based on the number of CD sales of two particular albums in Aalborg, Denmark, and New York, USA, for 2006 and 2007. The cube then contains sales counts for two cities, two albums, and two years. Depending on the specific application, a highly varying percentage of the cells in a cube are non-empty, meaning that cubes range from *sparse* to *dense*. Cubes tend to become increasingly sparse with increasing dimensionality and with increasingly finer granularities of the dimension values.



Multidimensional Modeling. Figure 1. Sales data cube.

A non-empty cell is called a *fact*. The example has a fact for each combination of time, album, and city where at least one sale was made. A fact has associated with it a number of *measures*. These are numerical values that "live" within the cells. In our case, there is only one measure, the sales count.

Generally, only two or three dimensions may be viewed at the same time, although for low-cardinality dimensions, up to four dimensions can be shown by nesting one dimension within another on the axes. Thus, the dimensionality of a cube is reduced at query time by *projecting* it down to two or three dimensions via *aggregation* of the measure values across the projected-out dimensions. For example, if the user wants to view just sales by City and Time, she aggregates over the entire dimension that characterizes the sales by Album for each combination of City and Time.

An important goal of multidimensional modeling is to "provide as much context as possible for the facts" [6]. The concept of *dimension* is the central means of providing this context. One consequence of this is a different view on *data redundancy* than in relational databases. In multidimensional databases, controlled redundancy is generally considered appropriate, as long as it considerably increases the information value of the data. One reason to allow redundancy is that multidimensional data is often *derived* from other data sources, e.g., data from a transactional relational system, rather than being "born" as multidimensional data, meaning that updates can more easily be handled [6]. However, there is usually no redundancy in the facts, only in the dimensions.

Having introduced the cube, its principal elements, dimensions, facts, and measures, are now described in more detail.

Dimensions

The notion of a dimension is an essential and distinguishing concept for multidimensional databases. Dimensions are used for two purposes: the *selection* of data and the *grouping* of data at a desired level of detail.

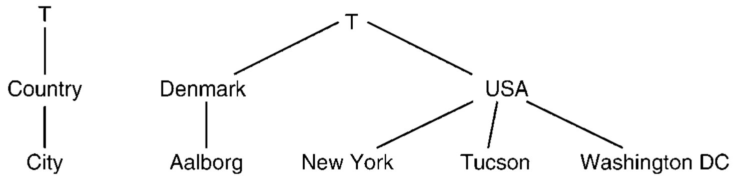
A dimension is organized into a containment-like *hierarchy* composed of a number of *levels*, each of which represents a level of detail that is of interest to the analyses to be performed. The instances of the dimension are typically called *dimension values*. Each such value belongs to a particular level.

In some cases, it is advantageous for a dimension to have *multiple hierarchies* defined on it. For example, a Time dimension may have hierarchies for both *Fiscal Year* and *Calendar Year* defined on it. Multiple hierarchies share one or more common lowest level(s), e.g., Day and Month, and then group these into multiple levels higher up, e.g., Fiscal Quarter and Calendar Quarter to allow for easy reference to several ways of grouping. Most multidimensional models allow multiple hierarchies. A dimension hierarchy is defined in the metadata of the cube, or the metadata of the multidimensional database, if dimensions can be shared.

In Fig. 2, the schema and instances of a sample *Location* dimension capturing the cities where CDs are sold are shown. The Location dimension has three levels, the City level being the lowest. City level values are grouped into *Country* level values, i.e., countries. For example, Aalborg is in Denmark. The ⊤ ("top") level represents *all* of the dimension, i.e., every dimension value is part of the ⊤ ("top") value.

In some multidimensional models, a level may have associated with it a number of *level properties* that are used to hold simple, non-hierarchical information. For example, the duration of an album can be a level property in the Album level of the Music dimension. This information could also be captured using an extra Duration dimension. Using the level property has the effect of not increasing the dimensionality of the cube.

Unlike the linear spaces used in matrix algebra, there is typically no ordering and/or distance metric on the dimension values in multidimensional models. Rather, the only ordering is the containment of



Multidimensional Modeling. Figure 2. Schema and instance for the location dimension.

lower-level values in higher-level values. However, for some dimensions, e.g., the Time dimension, an ordering of the dimension values is available and is used for calculating cumulative information such as "total sales in year to date."

Most models require dimension hierarchies to form *balanced trees*. This means that the dimension hierarchy must have uniform height everywhere, e.g., all departments, even small ones, must be subdivided into project groups. Additionally, direct links between dimension values can only go between immediate parent-child levels, and not jump two or more levels. For example, all cities are first grouped into states and then into countries, cities cannot be grouped directly under countries (as is the case in Denmark which has no states). Finally, each non-top value has precisely one parent, e.g., a product must belong to exactly one product group. Below, the relaxation of these constraints is discussed.

Facts

Facts are the objects that represent the *subject* of the desired analyses, i.e., the interesting "thing," or event or process, that is to be analyzed to better understand its behavior.

In most multidimensional data models, the facts are *implicitly* defined by their combination of dimension values. If a non-empty cell exists for a particular combination, a fact exists; otherwise, no fact exists. (Some other models treat facts as first-class objects with a separate identity [12].) Next, most multidimensional models require that each fact be mapped to precisely one dimension value at the lowest level in each dimension. Other models relax this requirement [12].

A fact has a certain *granularity*, determined by the levels from which its combination of dimension values are drawn. For example, the fact granularity in our example cube is "Year by Album by City." Granularities consisting of higher-level or lower-level dimension levels than a given granularity, e.g., "Year by Album Genre by City" or "Day by Album by City" for our example, are said to be *coarser* or *finer* than the given granularity, respectively.

It is commonplace to distinguish among three kinds of facts: *event* facts, *state* facts, and *cumulative snapshot* facts [6]. Event facts (at least at the finest granularity) typically model *events in the real world*, meaning that a unique instance, e.g., a particular sale of a given (particular physical instance of a) product in a given store at a given time, of the overall real-world process that is captured, e.g., sales for a supermarket chain, is represented by one fact. Examples of event facts include sales, clicks on web pages, and movement of goods in and out of (real) warehouses (flow).

A snapshot fact models the *state* of a given process at a given point in time. Typical examples of snapshot facts include the inventory levels in stores and warehouses, and the number of users using a web site. For snapshot facts, the same physical object, e.g., a specific physical instance of a can of beans on a shelf, with which the captured real-world process, e.g., inventory management, is concerned, may be "measured" at several time points, meaning that data related to that particular physical object will occur in several facts at different time points. This is unlike event facts, where a specific physical object such as a particular instance of a can of beans can only be sold once, and will thus only occur in one fact.

Cumulative snapshot facts are used to handle information about *a process up to a certain point in time*. For example, the total sales in the year to date may be considered as a fact. Then the total sales up to and including the current month this year can be easily compared to the figure for the corresponding month last year.

Often, all three types of facts can be found in a given data warehouse, as they support complementary classes of analyses. Indeed, the same base data, e.g., the movement of goods in a (real) warehouse, may often find its way into three cubes of different types, e.g., warehouse flow, warehouse inventory, and warehouse flow in year-to-date.

Measures

A *measure* has two components: a *numerical property* of a fact, e.g., the sales price or profit, and a *formula* (most often a simple aggregation function such as SUM) that can be used to combine several measure values into one. In a multidimensional database, measures generally represent the properties of the chosen facts that the users want to study, e.g., with the purpose of optimizing them.

Measures then take on different values for different combinations of dimension values. The property and formula are chosen such that the value of a measure is meaningful for all combinations of aggregation levels. The formula is defined in the metadata and thus not replicated as in the spreadsheet example. Most multidimensional data models provide the built-in concept of measures, but a few models do not. In these models, dimension values are used for computations instead [12].

It is important to distinguish among three classes of measures, namely *additive*, *semi-additive*, and *non-additive* measures, as these behave quite differently in computations.

Additive measure values can be summed meaningfully along any dimension. For example, it makes sense to add the total sales over Album, Location, and Time, as this causes no overlap among the real-world phenomena that caused the individual values. Additive measures occur for any kind of fact.

Semi-additive measure values cannot be summed along one or more of the dimensions, most often the Time dimension. Semi-additive measures generally occur when the fact is of type snapshot or cumulative snapshot. For example, it does not make sense to sum inventory levels across time, as the same inventory item, e.g., a specific physical instance of an album, may be counted several times, but it is meaningful to sum inventory levels across albums and stores.

Non-additive measure values cannot be summed along any dimension, usually because of the chosen formula. For example, this occurs when averages for lower-level values cannot be summed into averages for higher-level values. Non-additive measures can occur for any kind of fact.

The Modeling Process

Now, the process to be carried out when doing multidimensional modeling is covered. One difference from "ordinary" data modeling is that the multidimensional modeler should not try to include all the available data and all their relationships in the model, but only those parts which are essential "drivers" of the business. Another difference is that redundancy may be ok (in a few, well-chosen places) if introducing redundancy makes the model more intuitive for the user. For example, time-related information may be stored in both a Calendar time dimension and a Fiscal Year time dimension, or specific customer info may be present both in a person-oriented Customer dimension or a group-oriented Demographics dimension.

Kimball [6,5] advocates a four-step process when doing multidimensional modeling.

1. Choose the business process(es) to model
2. Choose the grain of the business process
3. Choose the dimensions
4. Choose the measures

Step 1 refers to the facts that not all business processes may be equally important for the business. For example, in a supermarket, there are business processes for *sales* and *purchases*, but the sales process is probably the one with the largest potential for increasing profits, and should thus be prioritized. Step 2 says that data should be captured at the right grain, or granularity, compared to the analysis needs. For example, "individual sales items" may be captured, or perhaps (slightly aggregated) "total sales per product per store per day" may be precise enough, enabling performance and storage gains. Step 3 then goes on to refine the schema of each part of the grain into a complete dimension with levels and attributes. For the example above, a Store, a Product, and a Time dimension are specified. Finally, Step 4 chooses the numerical measures to capture for each combination of dimension values, for example dollar sales, unit sales, dollar cost, profit, etc.

When doing multidimensional modeling "in the large" for many types of data (many cubes) and several user groups, the most important task is to ensure that analysis results are comparable across cubes, i.e., that the cubes are somehow "compatible." This is ensured by (as far as possible) picking dimensions and measures from a set of common so-called "conformed" dimensions and measures [6,5] rather than "re-definining" the same concept, e.g., product, each time it occurs in a new context. New cubes can then be put onto the common "DW bus" [5] and used together. This sounds easier than it is, since it often requires quite a struggle with different parts of an organisation to define for example a common Product dimension that can be used by everyone.

Complex Multidimensional Modeling

Multidimensional data modeling is not always as simple as described above. A complexity that is almost always present is that of handling *change* in the dimension values. Kimball [6,5] calls this the problem of *slowly changing dimensions*. For example, customer addresses, product category names, and the way products are categorized may change over time. This must be handled to ensure correct results both for current and historical data. Kimball advises three types of slowly changing dimensions: Type 1 (overwrite previous value with current value), Type 2 (keep versions of dimension rows), and Type 3 (keep previous and current value in different columns). Finally, the concept of *minidimensions* [6] advocates the separation of relatively static information (customer name, etc.) and dynamic information (income, number of kids, etc.) into separate dimensions.

The traditional multidimensional data models and implementation techniques assume that the data being modeled is quite regular. Specifically, it is typically assumed that all facts map (directly) to dimension values at the lowest levels of the dimensions and only to one value in each dimension. Further, it is assumed that the dimension hierarchies are simply balanced trees. In many cases, this is adequate to support the desired applications satisfactorily. However, situations occur where these assumptions fail.

In such situations, the support offered by "standard" multidimensional models and systems is inadequate, and more advanced concepts and techniques are called for. Now, the impact of irregular hierarchies on the performance enhancing technique known as partial, or practical, pre-computation, is reviewed.

Complex multidimensional data are problematic as they are not summarizable. Intuitively, data is *summarizable* if the results of higher-level aggregates can be derived from the results of lower-level aggregates. Without summarizability, users will either get wrong query results, if they base them on lower-level results, or the system cannot use pre-computed lower-level results to compute higher-level results. When it is no longer possible to pre-compute, store, and subsequently reuse lower-level results for the computation of higher-level results, aggregates must instead be calculated directly from base data, which leads to considerable increases in computational costs.

It has been shown that summarizability requires that aggregate functions be distributive and that the ordering of dimension values be *strict*, *onto*, and *covering* [12, 7]. Informally, a dimension hierarchy is *strict* if no dimension value has more than one (direct) parent, *onto* if the hierarchy is balanced, and *covering* if no containment path skips a level. Intuitively, this means that dimension hierarchies must be balanced trees. If this is not the case, some lower-level values will be either double-counted or not counted when reusing intermediate query results.

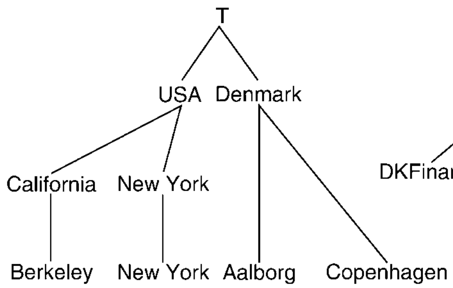
Figure 3 contains two dimension hierarchies: a Location hierarchy including a State level, and the hierarchy for the Organization dimension for some company. The hierarchy to the left is *non-covering*, as Denmark has no states. If aggregates at the State level are pre-computed, there will be no values for Aalborg and Copenhagen, meaning that facts mapped to these cities will not be counted when computing country totals.

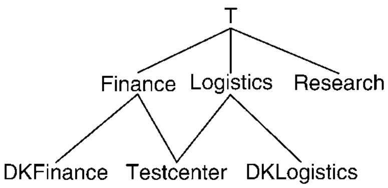
To the right in figure 3, the hierarchy is non-onto because the Research department has no further subdivision. If aggregates are materialized at the lowest level, facts mapping directly to the Research department will not be counted. The hierarchy is also non-strict as the TestCenter is shared between Finance and Logistics. If aggregates are materialized at the middle level, data for TestCenter will be counted twice, for both Finance and Logistics, which is, in fact, what is desired at this level. However, this means that data will be double-counted if these aggregates are then combined into the grand total.

Several design solutions exist that aims to solve the problems associated with irregular hierarchies by altering the dimension schemas or hierarchies [8,11].

Key Applications

Multidimensional data models have three important application areas within data analysis. First, multidimensional models are used in *data warehousing*. Briefly, a data warehouse is a large repository of





Multidimensional Modeling. Figure 3. Irregular dimensions.

integrated data obtained from several sources in an enterprise for the specific purpose of data analysis. Typically, this data is modeled as being multidimensional, as this offers good support for data analyses.

Second, multidimensional models lie at the core of *on-line analytical processing* (OLAP) systems. Such systems provide fast answers to queries that aggregate large amounts of so-called detail data to find overall trends, and they present the results in a multidimensional fashion. Consequently, a multidimensional data organization has proven to be particularly well suited for OLAP. The widely acknowledged "OLAP Report" company [15] provides an "acid test" for OLAP by defining OLAP as "fast analysis of shared multidimensional information" (FASMI). In this definition, "Fast" refers to the expectation of response times that are within a few seconds, "Analysis" refers to the need for easy-to-use support for business logic and statistical analyses, "Shared" suggests a need for security mechanisms and concurrency control for multiple users, "Multidimensional" refers to the expectation that a data model with hierarchical dimensions is used, and "Information" suggests that the system must be able to manage all the required data and derived information.

Third, multidimensional data are increasingly becoming the basis for *data mining*, where the aim is to (semi-) automatically discover unknown knowledge in large databases. Indeed, it turns out that multidimensionally organized data are also particularly well suited for the queries posed by data mining tools.

Future Directions

A pressing need for multidimensional modeling is the aspect of standardization, i.e., agreeing on a common data model, a graphical notation for it, and support by tools. Also, better integration between ordinary "operational modeling" and multidimensional modeling is needed. Another future research line is the modeling of important system aspects such as security, quality, requirements, evolution, and interoperability [13]. This will be extended to also cover the modeling of business intelligence applications such as data mining, patterns, *extraction-transformation-loading* (ETL), *what-if analysis*, and business process modeling [13]. Finally, an important line of research will cover the modeling of more (complex) types of data, including integrating multidimensional data with text data, semi-structured/XML/web data and spatial/spatio-temporal/mobile data [9].

Cross-references

▶ Business Intelligence
▶ Cube
▶ Data Warehouse
▶ Data Warehouse Maintenance, Evolution and Versioning
▶ Data Warehousing Systems: Foundations and Architectures
▶ Dimension
▶ Hierarchy
▶ Measure
▶ On-Line Analytical Processing
▶ Statistical Data Management
▶ Summarizability
▶ What-If Analysis

Recommended Reading

1. Chan P. and Shoshani A. SUBJECT: A directory driven system for organizing and accessing large statistical databases. In Proc. 9th Int. Conf. on Very Data Bases, 1983, pp. 553–563.
2. Codd E.F. Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate. E.F. Codd and Assoc., 1993.
3. Dyreson C.E., Pedersen T.B., and Jensen C.S. Incomplete information in multidimensional databases, M. Rafanelli (ed.). Multidimensional Databases: Problems and Solutions. Idea Group Publishing, 2003.
4. Gray J., Chaudhuri S., Bosworth A., Layman A., Venkatrao M., Reichart D., Pellow F., Pirahesh H. Data cube: A relational aggregation operator generalizing group-by, cross-tab and subtotals. Data Mining Knowl. Dis., 1(1):29–54, 1997.
5. Kimball R. et al. The Data Warehouse Lifecycle Toolkit. Wiley, New York, 1998.
6. Kimball R. and Ross M. The Data Warehouse Toolkit, 2nd ed. Wiley, New York, 2002.
7. Lenz H. and Shoshani A. Summarizability in OLAP and statistical data bases. In Proc. 9th Int. Conf. on Scientific and Statistical Database Management, 1997, pp. 39–48.
8. Niemi T., Nummenmaa J., and Thanisch P. Logical multidimensional database design for ragged and unbalanced aggregation. In Proc. 3rd Int. Workshop on Design and Man of Data Warehouses, CEUR Workshop Proc. 39, 2001, Paper 7.
9. Pedersen T.B. Warehousing the world: a few remaining challenges. In Proc. ACM 10th Int. Workshop on Data Warehousing and OLAP, 2007, pp. 101–102.
10. Pedersen T.B. and Jensen C.S. Multidimensional database technology. IEEE Comput., 34(12):40–46, 2001.
11. Pedersen T.B., Jensen C.S., and Dyreson C.E. Extending practical pre-aggregation in on-line analytical processing. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 663–674.
12. Pedersen T.B., Jensen C.S., and Dyreson C.E. A foundation for capturing and querying complex multidimensional data. Inf. Syst., 26(5):383–423, 2001.
13. Rizzi S., Abello A., Lechtenbrger J., and Trujillo J. Research in data warehouse modeling and design: dead or alive? In Proc.

ACM 9th Int. Workshop on Data Warehousing and OLAP, 2006, pp. 3–10.
14. Shoshani A. OLAP and statistical databases: similarities and differences. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997, pp. 185–196.
15. The OLAP Report web page. http://www.olapreport.com. Current as of November 22, 2007.
16. Thomsen E. OLAP Solutions: Building Multidimensional Information Systems. Wiley, New York, 1997.
17. Vassiliadis P. and Sellis T.K. A survey of logical models for OLAP databases. ACM SIGMOD Rec., 28(4):64–69, 1999.

Multidimensional Scaling

Heng Tao Shen
The University of Queensland, Brisbane,
QLD, Australia

Synonyms

MDS

Definition

Multidimensional scaling (MDS) is a mathematical dimension reduction technique that best preserves the inter-point distances by analyzing gram matrix. Given any two points p_i and p_j in a dataset P, MSD aims to minimize the following objective function:

$$Error = \sum [d(p_i, p_j) - d'(p_i, p_j)]^2$$

Where $d(p_i, p_j)$ and $d'(p_i, p_j)$represent the distance between points p_i and p_j in original space and the lower dimensional subspace respectively.

Key Points

Multidimensional scaling (MDS) is a set of related statistical techniques often used in data visualisation and analysis for exploring similarities or dissimilarities in data. An MDS algorithm starts with a matrix of point-point (dis)similarities, then assigns a location of each point in a low-dimensional space. The points are arranged in this subspace so that the distances between pairs of points have their original distance maximally retained. MDS is a generic term that includes many different specific types. These types can be classified according to whether the data are qualitative (called nonmetric MDS) or quantitative (metric MDS). The number of (dis)similarity matrices and the nature of the MDS model can also classify MDS types. This classification yields classical MDS (one matrix, unweighted model), replicated MDS (several matrices, unweighted model), and weighted MDS (several matrices, weighted model) [1,2].

MDS applications include scientific visualisation and data mining in fields such as cognitive science, information science, psychophysics, psychometrics, marketing and ecology [2].

Cross-references

▶ Dimensionality Reduction
▶ Discrete Fourier Transform
▶ Discrete Wavelet Transform and Wavelet Synopses
▶ Independent Component Analysis
▶ Isometric Feature Mapping
▶ Latent Semantic Indexing
▶ Locality-Preserving Mapping
▶ Locally Linear Embedding (Lle) Laplacian Eigenmaps
▶ Principal Component Analysis
▶ Semantic Subspace Projection

Recommended Reading

1. Cox M.F. and Cox M.A.A. Multidimensional Scaling. Chapman and Hall, 2001.
2. Young F.W. and Hamer R.M. Multidimensional Scaling: History, Theory and Applications. Erlbaum, New York, 1987.

Multidimensional Visualization

▶ Parallel Coordinates

Multi-Granularity Modeling

▶ Multiple Representation Modeling

Multi-Layered Architecture

▶ Multi-Tier Architecture

Multi-Level Recovery and the ARIES Algorithm

Gerhard Weikum
Max-Planck Institute for Informatics, Saarbrücken, Germany

Definition

In contrast to basic database recovery with page-level logging and redo/undo passes, *multi-level recovery* is needed whenever the database system uses fine-grained concurrency control, such as index-key locking or operation-based "semantic" conflict testing, or when log records describe composite operations that are not guaranteed to be atomic by single page writes (as a consequence of concurrency control or for other reasons). Advanced methods perform logging and recovery at multiple levels like pages and data objects (records, index entries, etc.). Page-level recovery is needed to ensure the atomicity and applicability of higher-level operations, and also for efficient redo. Higher-level recovery is needed to perform correct undo for composite operations of loser transactions. In addition, logged actions at all levels must be testable at recovery-time, by embedding extra information in database pages, typically using log sequence numbers (LSNs), and appropriate logging of recovery steps. A highly optimized instantiation of these principles is the *ARIES algorithm* (Algorithms for Recovery and Isolation Exploiting Semantics) [8,7], the de facto standard solution for industrial-strength database systems. Its salient features are: very fast, potentially parallelizable or selective, redo for high availability; use of LSNs and compensation log records (CLRs) for tracking recovery progress; full-fledged support for crash and media recovery; suitability for all kinds of semantic concurrency control methods.

Historical Background

Crash recovery for composite operations on database records and indexes has first been addressed by [3], but that solution required heavy-weight check pointing (based on shadow storage) and was fairly inefficient. Some commercial engines developed various techniques to overcome these problems while supporting fine-grained concurrency control, but there is very little public literature on such system internals [1,2,4]. The ARIES algorithm was the first comprehensive solution [8] and has become the state-of-the-art method for industrial-strength recovery [7]. In parallel to and independently of the ARIES papers, research on multi-level recovery developed general principles and a systematic framework [9,10,6]. The textbook [11] discusses both the general framework and the ARIES algorithm in great detail. A correctness proof for the ARIES algorithm is given in [5].

Foundations

Basic database recovery methods log page modifications during normal operation. During the restart after a system crash (i.e., soft failure of the server with all disks intact), a three-phase recovery procedure is performed with an analysis pass, a redo pass, and an undo pass over the log file. This is appropriate when the database system uses page locking or some other page-granularity concurrency control. Such locking guarantees that all updates of loser transactions (i.e., uncommitted transactions that require undo) that are in conflict with updates of winner transactions (i.e., committed transactions that may require redo) follow the winners' updates in the log file. However, with fine-grained locking or some form of semantic concurrency control (e.g., exploiting commutative operations on hot-spot objects), this invariant does no longer hold and thus necessitates more advanced recovery algorithms.

As an example, consider the following execution history of two transactions t1 and t2 that insert various database records, with time proceeding from left to right and c2 denoting a successful commit of t2:

$$insert1(x) \quad insert1(y) \quad crash \quad insert2(z) \quad c2$$

The insertions may appear to be single operations, but they are not atomic from the system viewpoint. In fact, each of them may require multiple page writes to maintain indexes and other storage structures. For example, the following history of page writes may result from the above execution:

$$w1(p)\ w1(q)\ w1(s)\ w1(r)\ w1(p)\ w1(q)\ crash$$
$$w2(p)\ w2(s)\ c2$$

Here, p would be a data page into which all three records x, y, z are inserted, and q and s may be leaf pages of the same B + -tree index. For simplicity, the history does not show any read accesses like accesses for descending the tree. It may happen that the index update on behalf of record x triggers a leaf split of page q with a newly allocated page s and a corresponding update to the parent page r. The subsequent record operations may then access the new pages or the old page q, depending on where their corresponding keys now reside. This is a very normal situation for a database system, and it is perfectly admissible from a concurrency control viewpoint, because there are no (high-level) conflicts between the three insert operations. The system may have to use additional short-term locks or latches on pages to implement a multi-level concurrency control method, but this is very normal as well.

If the standard page-level recovery were applied to this situation, it would first redo (if necessary) all page writes of winner transaction t2 and then undo all page writes of loser transaction t1, using before-images of pages or byte-range-oriented log records. This would lead to two kinds of problems:

- Redoing the page write w2(s) may be logically flawed and lead to an inconsistent index state if the effects of the preceding leaf split are not properly reflected in page s. But it is indeed possible that the loser updates w1(q) and w1(s) were not written to disk before the crash, and no redo would be performed for them.
- If all index updates were fully recovered and correctly captured in the database by the time the undo recovery for t1 takes place, undoing the write w1(s) of page s would restore the page as of the time before t1 started, thus accidentally – and incorrectly – eliminating also the index update of the winner transaction t2.

If, on the other hand, the logging and recovery procedures were changed so that only record- and index-level operations are captured and redone or undone, one would run into a third problem:

- If the system crashed in the middle of a high-level operation, say in between the w1(q) and the w1(s) steps, the database may be left in an inconsistent state with partial effects of an operation. Such a database would not be recoverable as all logged operations could face a state that they cannot properly interpret.

These problems show the need for multi-level recovery; the solution must meet the following requirements:

- *Operation atomicity:* High-level operations that comprise multiple page writes must be guaranteed to appear atomic (i.e., have an all-or-nothing impact on the database).
- *High-level undo:* Operations of loser transactions must, in full generality, be undone by means of inverse operations at the same level of abstraction. For example, the insertion of an index key must be undone by performing a delete operation on that key, not by restoring the before-images of the underlying pages (and neither by corresponding byte-range modifications).
- *Testable operations:* Before invoking a high-level operation for undoing or redoing the effect of a prior operation, it must be tested whether the effects of the prior operation are indeed present (as they may have been lost by the crash or already undone/redone in a recovery procedure that was interrupted by another crash). This testability is crucial for handling non-idempotent operations.
- *Efficient redo:* As the restart time and thus the unavailability of the system is usually dominated by the redo pass, it is crucial that the redo actions are performed as efficiently as possible. This strongly suggests performing redo in terms of page writes rather than re-executing high-level operations.

Multi-level recovery methods address these requirements in the following way:

- For *proper undo*, both page-level writes and higher-level operations are logged. The page-level log records guarantee that high-level operations can always be made to appear atomic. The high-level log records guarantee that undo can be performed by means of inverse operations. During the undo of a high-level operation, page-level logging is again enabled. This way, the first two requirements are satisfied.
- For *testable operations*, when the recovery procedure undoes a (high-level) operation, both the resulting page writes and a marker for the inverse operation itself are logged. The latter kind of log record is referred to as a compensation log record (CLR). In addition, the standard technique of maintaining log sequence numbers (LSNs) in the headers of modified pages, as a form of virtual time stamping, is used to be able to compare a log record to the state of a page and decide whether the logged action should be undone/redone or disregarded.
- For *efficient redo*, although redoing high-level operations may add to the repertoire of recovery actions, it is much more desirable to perform all redo steps in terms of page writes. This can leverage all kinds of acceleration techniques that have been developed for more conventional, page-level recovery like asynchronous check pointing and dirty-pages bookkeeping, smart scheduling of page reads from the database disk, parallelized per-page redo, and selective redo for pages with very high availability demands.
- Further considerations on the redo pass lead to the *repeating-history principle* [8]: rather than aiming to redo only winner updates or at least as few loser updates as possible, it is much simpler to redo all

logged page writes regardless of their transaction status, thus effectively reconstructing the database as of the time of the crash.

All these principles together result in the following algorithmic template for multi-level recovery:

- Analysis pass for determining loser transactions.
- Redo pass by repeating history in terms of logged page writes.
- Undo pass for loser transactions, with page-level undo for incomplete high-level operations and high-level undo for complete (and possibly just redone) high-level operations. Logging at both levels is enabled during the undo pass, thus creating new log records: page-write log records during the operation's execution, and undo information for the entire operation at the very end, thus also marking the completion of the operation.

For all steps, *idempotence* is ensured by two means: for page writes the standard comparison of page-header LSN versus log-record LSN is performed; for high-level operations, only undo idempotence is a potential issue, and this is guaranteed by the fact that the preceding redo pass always repeats history so that all completely repeated operations need subsequently be undone by definition.

The *ARIES algorithm* is an integrated and highly optimized instantiation of these principles, with various additional features. Its recovery procedure performs three passes over the log: analysis, repeating-history redo, and undo. The analysis pass mostly follows standard recovery methods; the redo pass has been discussed above; the undo pass uses additional techniques based on the use of *compensation log records (CLRs)*. The following undo-relevant log records are produced by ARIES:

- During normal operation, page writes are logged in a way that they can be redone or undone (whichever is needed later), and each high-level operation is logged for undo purposes following all page-write log records that were produced during the operation's execution. The high-level log records have a backward pointer that points to the preceding high-level action of the same transaction, thus allowing the recovery manager to skip the operation's logged page writes.
- During the undo pass, when undoing a page write, a CLR is written with a backward pointer to the log record that precedes the undo page write within the same transaction. When undoing a high-level operation, normal page-write log records are written during the execution of the inverse operation, and a CLR for the entire high-level operation is written at the end. That CLR again has a backward pointer to its preceding high-level action, skipping its own page writes.

With these preparations, the undo procedure itself is rather straightforward. For each loser transaction, it locates the most recent log record and then follows the backward chain of log records. Whenever a CLR is encountered, this tells the recovery manager that the undo of the corresponding action is already completed (either already during normal operation or by the preceding redo pass) and the log record should thus be disregarded. Page-write log records are relevant for incomplete high-level operations; otherwise high-level log records determine the undo logic.

This undo procedure of the ARIES algorithm has a number of great benefits:

- It handles high-level undo in a correct and efficient way, thus allowing *fine-grained and semantic concurrency control.*
- It handles *nested rollbacks* in a correct and efficient way. These are situations where a transaction rollback is interrupted by a crash and later considered for undo or when the undo pass after a server crash is interrupted by a second crash. In all these situations, it is guaranteed that the amount of recovery work stays bounded, regardless of how many "nested" crashes might occur during recovery. This is important for high availability.
- For *media recovery*, restoring the database after disk failures, an analogous but even more severe situation arises. As media recovery always starts with a backup copy of the database and then repeats the history of a potentially very long archive log, rollbacks or undo steps for (soft) system crashes that happened long ago would interfere with log truncation and become performance showstoppers with pre-ARIES recovery methods. The way ARIES generates redo log records for undo actions and CLRs for progress tracking, media recovery is as fast as possible, which is crucial for availability.

For the example scenario given above, ARIES would create the following log records during normal operation, denoted in the form *LSN:action*. Note that log records 5 and 8 will only be used for undo purposes (if necessary). Further note that the example happens to show page-level log records for the second insert operation of t1 but no high-level log record. This may occur because of the crash happening before the high-level log record was flushed to the log disk.

$$1 : w1(p)\ 2 : w1(q)\ 3 : w1(s)\ 4 : w1(r)$$

$$5 : insert1(x)\ 6 : w2(p)\ 7 : w2(s)\ 8 : insert2(z)$$

$$9 : w1(p)\ 10 : w1(q)\ 11 : c2$$

During recovery, the redo pass processes log records 1, 2, 3, 4, 6, 7, 9, and 10. Subsequently the undo pass processes log records 10, 9, and 5 (in this – chronologically reverse – order). As it does so, it will create the following new log records, with CLRs denoted in the form *LSN:action→UndoNextLSN* with UndoNextLSN being the LSN of the log record to which the CLR has a backward pointer.

$$12 : w1(q) \rightarrow 9\ 13 : w1(p) \rightarrow 5\ 14 : w1(s)$$

$$15 : w1(p)\ 16 : delete1(x) \rightarrow 0$$

If the system crashed again immediately after the completion of the delete1(x) undo step, the redo pass would repeat the page writes with LSNs 12, 13, 14, and 15 (in addition to all writes with LSNs 1 through 11 that may need redo again). This means that all effects of t1 have been properly removed. The subsequent undo pass would then encounter the CLR 16, but its backward pointer immediately tells the recovery manager that it can skip all log records of transaction t1 as t1 had already been completely undone before the second crash.

If the system crashed again after the action with LSN 14 (a page write issued on behalf of the high-level undo of insert1(x)), the redo pass would repeat the page writes with LSNs 12, 13, and 14, thus effectively removing all effects of insert1(y) but only some partial effects of insert1(x). The subsequent undo pass would start with LSN 14, undo it and create a new CLR, and then encounter LSN 13, which points to LSN 5 which in turn is the next logged action to undo.

In general, ARIES can be implemented with very low overhead, and it is compatible with other optimizations in the storage engine of a database system: flexible free space management, flexible buffer management, acceleration techniques for the redo pass, and many more. For *high availability*, the redo pass of both crash and media recovery can be parallelized or performed selectively for most important page sets; media recovery efficiently works also with fuzzy backups without ever quiescing the system. For *index management*, extensions of ARIES have been developed that optimize the locking, logging, and recovery of index keys in B + -trees. Finally, there are also extensions of ARIES for the special requirements of *shared-disk clusters* with automated fail-over procedures and very high availability.

Future Directions

The ARIES algorithm is a mature and comprehensive solution that can be readily adopted for most data management systems. A salient property of the multi-level recovery framework is that it can be generalized to arbitrary kinds of composite operations (with deeper and flexible nestings). All the ARIES techniques for efficient repeating-history redo are directly applicable, and the undo procedures need to be extended to handle a conceptual stack of undo log records and corresponding CLRs – with the stack actually being embedded in the linear log. This generalization is of potential interest for modern applications like composite Web services or enterprise-level middleware with integrated recovery.

Cross-references

- ▶ Atomicity
- ▶ Logging
- ▶ Persistence
- ▶ System Recovery
- ▶ Transaction

Recommended Reading

1. Borr A.J. Robustness to crash in a distributed database: a non shared-memory multi-processor approach. In Proc. 10th Int. Conf. on Very Large Data Bases, 1984, pp. 445–453.
2. Crus R.A. Data recovery in IBM database 2. IBM Syst. J., 23(2):178–188, 1984.
3. Gray J., McJones P.R., Blasgen M.W., Lindsay B.G., Lorie R.A., Price T.G., Putzolu G.R., and Traiger I.L. The recovery manager of the system R database manager. ACM Comput. Surv., 13(2):223–243, 1981.
4. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1993.
5. Kuo D. Model and verification of a data manager based on ARIES. ACM Trans. Database Syst., 21(4):427–479, 1996.

6. Lomet D.B. MLR: a recovery method for multi-level systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1992, pp. 185–194.
7. Mohan C. Repeating history beyond ARIES. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 1–17.
8. Mohan C., Haderle D.J., Lindsay B.G., Pirahesh H., and Schwarz P.M. ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. Database Syst., 17(1):94–162, 1992.
9. Moss J.E.B., Griffeth N.D., and Graham M.H. Abstraction in recovery management. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 72–83.
10. Weikum G., Hasse C., Brössler P., and Muth P. Multi-level recovery. In Proc. 9th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1990, pp. 109–123.
11. Weikum G. and Vossen G. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2001.

Multilevel Secure Database Management System

BHAVANI THURAISINGHAM
University of Texas at Dallas, Richardson, TX, USA

Synonyms

Secure database systems; Trusted database systems

Definition

Many of the developments in the 1980s and 1990s in database security were on multi-level secure database management systems (MLS/DBMS). These systems were also called trusted database management systems (TDBMS). In a MLS/DBMS, users are cleared at different clearance levels such as Unclassified, Confidential, Secret and TopSecret. Data is assigned different sensitivity levels such as Unclassified, Confidential, Secret, and TopSecret. It is generally assumed that these security levels form a partially ordered lattice. For example, Unclassified < Confidential < Secret < TopSecret. Partial ordering comes from having different compartments. For example, Secret Compartment A may be incomparable to Secret Compartment B.

Historical Background

MLS/DBMSs have evolved from the developments in multilevel secure operating systems such as MULTICS and SCOMP (see for example [4]) and the developments in database systems. Few developments were reported in the late 1970s on MLS/DBMSs. However, during this time there were many developments in discretionary security, such as access control for System R and INGRES as well as many efforts on statistical database security. Then there was a major initiative by the Air Force and a summer study was convened. This summer study marks a significant milestone in the development of MLS/DBMSs [2].

The early developments in MLS/DBMSs influenced the Air Force Summer Study a great deal. Notable among these efforts are the Hinke-Schaefer approach to operating system providing mandatory security, the Ph.D. Thesis of Deborah Downs at UCLA (University of California at Los Angeles), the IP Sharp Model developed in Canada and the Naval Surveillance Model developed at the MITRE Corporation. The Hinke Schaefer approach [3] essentially developed a way to host MLS/DBMSs on top of the MULTICS MLS operating system. The system was based on the relational system and the idea was to partition the relation based on attributes and store the attributes in different files at different levels. The operating system would then control access to the files. The early efforts showed a lot of promise to designing and developing MLS/DBMSs. As a result, the Air Force started a major initiative, which resulted in the summer study of 1982

Since the summer study, several efforts were reported throughout the 1980s. Many of the efforts were based on the relational data model. At the end of that decade, the National Computer Security Center started a major effort to interpret the Trusted Computer Systems Evaluation Criteria for database systems [7]. This interpretation was called the Trusted Database Interpretation [8]. In the 1990s research focused on non-relational systems including MLS object database systems and deductive database systems. Work was also carried out on multilevel secure distributed database systems. Challenging research problems such as multilevel data models, inference problem and secure transaction processing were being investigated. Several commercial products began to emerge. Since the late 1990s, while the interest in MLS/DBMSs began to decline a little, efforts are still under way to examine multilevel security for emerging data management technologies. A detailed discussion of many of the developments with significant references are given in [5].

Foundations

Many of the developments were based on the relational model. The early systems were based on the Integrity

Lock approach developed at the MITRE Corporation. Two prototypes were designed and developed. One used the MISTRESS relational database system and the other used the INGRES relational database system. Around 1985 TRW designed and developed a MLS/DBMS called ASD and this system was designed to be hosted on ASOS (the Army Secure Operating System). The approaches were based on the Trusted Subject based architecture. Later on TRW developed some extensions to ASD and the system was called ASD Views where access was granted on views (GARV88). Two of the notable systems designed in the late 1980s were the SeaView system by SRI International and LOCK Data Views system by Honeywell. These two efforts were funded by the then Rome Air Development Center and the goal was to focus on the longer term approaches proposed by the Summer Study. Both efforts influenced the commercial developments a great deal. Three other efforts worth mentioning are the SINTRA system developed by the Naval Research Laboratory, the SWORD system developed by the then Defense Research Agency and funded by the Ministry of Defense in the United Kingdom and the SDDBMS effort by Unisys. The SINTRA system was based on the distributed architecture proposed by the Air Force Summer Study. The SWORD system proposed some alternatives to the SeaView and LOCK Data Views data models. While the initial planning for these systems began in the late 1980s, the designs were actually developed in the early 1990s. The SDDBMS effort was funded by the Air Force Rome Laboratory and investigated both the partitioned and replicated approaches to designing an MLS/DBMS.

Around l987, the Rome Air Development Center (now known as Air Force Research Laboratory in Rome) funded an effort to design an MLS/DBMS based on the Entity Relationship (ER) model. The ER model was initially developed in 1976 by Peter Chen and since then it has been used extensively to model applications. The goal of the security effort carried out by Gajnak and his colleagues was to explore security properties for the ER model as well as to explore the use of secure ER models to design DBMSs. The effort produced MLS ER models that have since been used to model secure applications. Furthermore variations of this model have been used to explore the inference problem by Burns, Thuraisingham and Smith. However, there does not appear to have been any efforts undertaken on designing MLS/DBMSs based on the ER model. In summary, the ER approach has contributed extensively toward designing MLS applications.

During the late 1980s, efforts began on designing MLS/DBMSs based on object models. Notable among these efforts is the one by Keefe, Tsai and Thuraisingham who designed the SODA model by Keefe and his colleagues. Later Thuraisingham designed the SORION and SO2 models. These models extended models such as ORION and O2 with security properties. Around 1990 Millen and Lunt produced an object model for secure knowledge base systems. Jajodia and Kogan developed a message-passing model in 1990. Finally MITRE designed a model called UFOS. Designs of MLS/DBMSs were also produced based on the various models. The designs essentially followed the designs proposed for MLS/DBMSs based on the relational model. However with the object model, one had to secure complex objects as well as handle secure method execution. While research progressed on designing MLS/DBMSs based on objects, there were also efforts on using object models for designing secure applications. Notable efforts were those by Sell and Thuraisingham. Today with the development of UML (Unified Modeling Language) there are efforts to design secure applications based on UML.

Around 1989 work began at MITRE on the design and development of multilevel secure distributed database systems (MLS/DDBMS). Prototypes connecting MLS/DBMSs at different sites were also developed. Work was then directed toward designing and developing MLS heterogeneous distributed database systems. These efforts focused on connecting multiple MLS/DBMSs, which are heterogeneous in nature. Research was also carried out on MLS federated databases by Thuraisingham and Rubinovitz.

In the late 1970 and throughout the 1980s there were many efforts on designing and developing logic-based database systems. These systems were called deductive databases. While investigating the inference problem, multilevel secure deductive database systems were designed. These systems were based on a logic called NTML (Non monotonic Typed Multilevel Logic) designed by Thuraisingham at MITRE. NTML essentially provides the reasoning capability across security levels, which are non-monotonic in nature. Essentially, it incorporates constructs to reason about the applications at different security levels. A Prolog language based on NTML, which is called NTML-Prolog, was also designed. Both reasoning with the Closed World

Assumption as well as with the Open World Assumption were investigated. Due to the fact that there was limited success with logic programming and the Japanese Fifth Generation Project, deductive systems are being used only for a few applications. If such applications are to be multilevel secure, then systems such as those based on NTML will be needed. Nevertheless there is use for NTML on handling problems such as the inference problem. Note that presently the integration of NTML-like logic with descriptive logics for secure semantic webs is being explored.

Researchers have identified several hard problems. The most notable hard problem is the Inference problem. Inference problem is the process of posing queries and deducing sensitive information form the legitimate responses received. Many efforts have been discussed in the literature to handle the inference problem First of all, Thuraisingham proved that the general inference problem was unsolvable [6] and this effort was stated by Dr. John Campbell of the National Security Agency as one of the significant developments in database security in [1]. Then Thuraisingham explored the use of security constraints and conceptual structures to handle various types of inferences. Note that the aggregation problem is a special case of the inference problem where collections of data elements are sensitive while the individual data elements are Unclassified. Another hard problem is secure transaction processing. Many efforts have been reported on reducing covert channels when processing transactions in MLS/DBMSs including the work of Jajodia, Bertino and Atluri among others. A third challenging problem is developing a multilevel secure relational data model. Various proposals have been developed including those by Jajodia and Sandhu, the Sea View model by Denning and her colleagues and the LOCK Data Views model by Honeywell. SWORD developed by Wiseman also proposed its own model. The problem is due to the fact that different users have different views of the same element. If multiple values are used to represent the same entity then the integrity of databases is violated. However, if what is called polyinstantiation is not enforced, then there is a potential for signaling channels. This is still an open problem.

Key Applications

The department of defense was the major funding agency for multilevel secure database management systems. The applications are mainly in the defense and intelligence area. However many of the concepts can be used to design systems that have multiple labels, privacy levels or roles. Therefore these systems can also be used to a limited extent for non-defense applications including healthcare and financial applications.

Future Directions

As technologies emerge, one can examine multilevel security issues for these emerging technologies. For example, as object database systems emerged in the 1980s, multilevel security for object databases began to be explored. Today there are many new technologies including data warehousing, e-commerce systems, multimedia systems, real-time systems and the web and digital libraries. Only a limited number of efforts have been reported on investigating multilevel security for the emerging data management systems. This is partly due to the fact the even for relational systems, there are hard problems to solve with respect to multilevel security. As the system becomes more complex, developing high assurance multilevel systems becomes an enormous challenge. For example, how can one develop usable multilevel secure systems say for digital libraries and e-commerce systems? How can one get acceptable performance? How does one verify huge systems such as the World Wide Web? At present, there is still a lot to do with respect to discretionary security for such emerging systems. As progress is made with assurance technologies and if there is a need for multilevel security for such emerging technologies, then research initiatives will commence for these areas.

Cross-references

► Database Security
► Inference Problem
► Mandatory Access Control
► Role Based Access Control

Recommended Reading

1. Campbell J. A year of progress in database security. In Proc. National Computer Security Conf., 1990.
2. Committee on Multilevel Data Management Security, Air Force Studies Board. Multilevel Data Management Security. National Academy Press, Washington, DC, 1983.
3. Hinke T. and Schaefer M. Secure data management system. System Development Corp., Technical Report RADC-TR-75-266, November 1975.
4. IEEE Computer Magazine, Volume 16, #7, 1983.
5. Thuraisingham B. Database and Applications Security: Integrating Data Management and Information Security. CRC Press, Boca Raton, FL, 2005.

6. Thuraisingham B. Recursion theoretic properties of the inference problem. Presented at the IEEE Computer Security Foundations Workshop, Franconia, NH, June 1990 (also available as MITRE technical Paper MTP291, June 1990).
7. Trusted Computer Systems Evaluation Criteria, National Computer Security Center, MD, 1985.
8. Trusted Database Interpretation. National Computer Security Center, MD, 1991.

Multilevel Security

► Mandatory Access Control

Multilevel Transactions and Object-Model Transactions

Gerhard Weikum
Max-Planck Institute for Informatics, Saarbrücken, Germany

Synonyms

Layered transactions; Open nested transactions

Definition

Multilevel transactions are a variant of nested transactions where nodes in a transaction tree correspond to executions of operations at particular levels of abstraction in a layered system architecture. The edges in a tree represent the implementation of an operation by a sequence (or partial ordering) or operations at the next lower level. An example instantiation of this model are transactions with record and index-key accesses as high-level operations which are in turn implemented by reads and writes of database pages as low-level operations. The model allows reasoning about the correctness of concurrent executions at different levels, aiming for serializability at the top level: equivalence to a sequential execution of the transaction roots. This way, semantic properties of operations, like different forms of commutativity, can be exploited for higher concurrency, and correctness proofs for the corresponding protocols can be derived. Likewise, multilevel transactions provide a framework for structuring recovery methods and reasoning about their correctness.

Multilevel transactions have wide applications outside of database engines as well. For example, transactional properties can be provided by middleware application servers, layered on top of a database system. A generalization of this approach is the notion of object-model transactions, also known as open nested transactions (trees where the nodes correspond to arbitrary method invocations). In contrast to multilevel transactions, there is no layering constraint anymore, and arbitrary caller-callee relations among objects of abstract data types can be expressed. This provides a model for reasoning about transactional guarantees in composite web services, and for structuring the design and run-time architecture of web-service-based applications.

Historical Background

Object-model transactions have been around for thirty years, going back to the work of Bjork and Davies on “spheres of control” [2]. The first work that made these concepts explicit and gave formal definitions is by Beeri et al. [1]. Parallel work on the important special case of multilevel transactions has been done by Moss et al. [7] and Weikum et al. [13,11]. The textbook [14] gives a detailed account of both conceptual and practical aspects. A broader perspective of extended transaction models is given by [9]. More recently, the concept of object-model transactions has received considerable attention also for long-running workflows (e.g., [10,15]) and transactional memory (e.g., [8]).

On the system side, multilevel transaction protocols have been employed for concurrency control and recovery in various products and prototypes [3,5,6,12]. Typically, the layered structure of the protocols is only implicit; a suite of additional smart implementation techniques is used for integrated, highly efficient code. An example for concurrency control is transaction-duration locking for index-manager operations combined with operation-duration latching. Examples for recovery are the ARIES family of algorithms by Mohan et al. [6] and the MLR algorithm by Lomet [5].

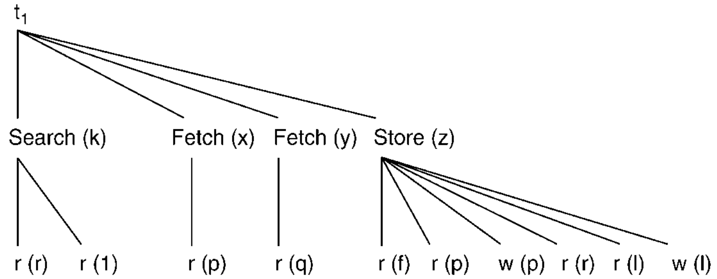
Foundations

Multilevel and object-model transactions are best understood in an object model where operations are invoked on arbitrary objects. This allows exploiting “semantic” properties of the invoked operations for the sake of improved performance. This model also captures situations where an operation on an object invokes other operations on the same or other objects. Often the implementation of an object and its operations requires calling operations of some lower-level types of objects.

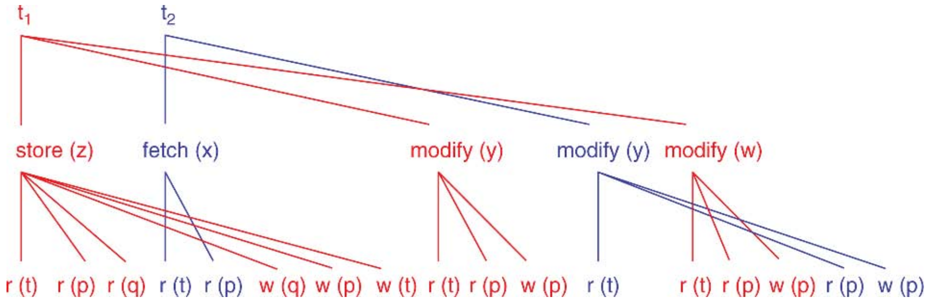
For example, operations at the access layer of a database system, such as index searches, need to invoke page oriented operations at the storage layer underneath. Similar invocation hierarchies may exist among a collection of business objects that are made available as abstract data type (ADT) instances within a data server or an application server, e.g., a “shopping cart” or a “bank account” object type along with operations like deposit, withdraw, get_balance, get_history, compute_interests, etc. The following figure depicts an example of a transaction execution against an object model scenario that refers to the internal layers of a database system.

The figure shows a transaction, labeled t1, which performs, during its execution, (i) an SQL Select command to retrieve all records from a database that satisfy a certain attribute-value condition, and, after inspecting the result set, (ii) an SQL command to insert a record for a new record with this attribute value. Since SQL commands are translated into query execution plans already at compile time, the operations invoked at run time refer to an internal level of index and record accesses. The Select command is executed by first issuing a Search operation with some key k on an index that returns the RIDs (i.e., addresses) of the result records. Next, these records, referred to as x and y in the figure, are fetched by dereferencing their RIDs. The Search operation in turn invokes operations at the underlying storage layer: read and write operations on pages. First the root of a B+ tree is read, labeled as page r in the figure, which points to a leaf page, labeled l, that contains the relevant RID list for key k. The subsequent Fetch operations to access the two result records x and y by their RIDs, require only one page access each to pages p and q, respectively. Finally, the SQL Insert command is executed as a Store operation, storing the new record and also maintaining the index. This involves first reading a metadata page, labeled f that holds free space information in order to find a page p with sufficient empty space. Then that page is read and subsequently written after the new record z has been placed in the page. Finally, the RID of the new record z is added to the RID list of the key k in the index. This requires reading the B+ tree root page r, reading the proper leaf page l, and finally writing page l after the addition of the new RID.

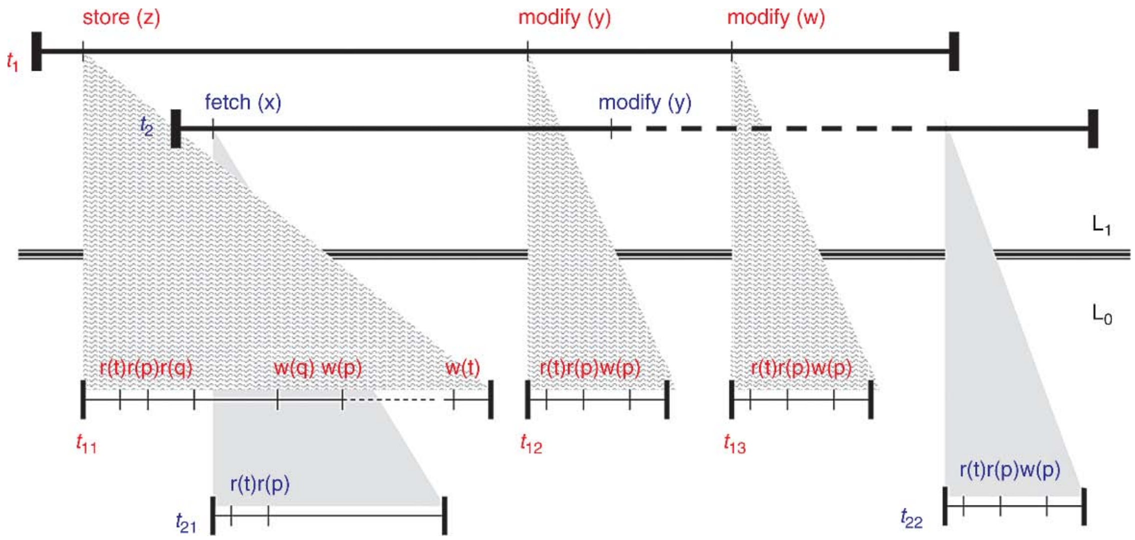
This entire execution is represented in a graphical, compact form, by connecting the calling operation and the called operation with an arc when operations invoke other operations. As a convention, the caller is



Multilevel Transactions and Object-Model Transactions. Figure 1. Example of a multilevel transaction.



Multilevel Transactions and Object-Model Transactions. Figure 2. Example of a multilevel schedule.



Multilevel Transactions and Object-Model Transactions. Figure 3. Concurrent execution of multilevel schedules with a multilevel locking protocol.

always placed closer to the top of the picture than the callee. Furthermore, the order in which operations are invoked is represented by placing them in "chronological" order from left to right, which suffices for illustration purposes.

More formally, a ***multilevel transaction tree*** is defined as a partially ordered labeled tree with node labels being operation invocations and the leaf nodes denoting elementary (i.e., indivisible) read and write operations. Moreover, the tree must be perfectly balanced with all leaves having the same distance from the root; this constraint is dropped for the more general case of ***object-model transactions (open nested transactions)***. Finally, a constraint is imposed on conflicting leaf nodes to be totally ordered so that no concurrent write-write or read-write pair of operations is possible on the same elementary object; for all other cases partial orders are allowed. It is important to note that transaction trees model executions and not programs. Thus, node labels are method names along with concrete input parameter values (and possibly even output parameter values if these are exploited in the reasoning about concurrency); edges denote the dynamic calling structure and not a static hierarchy.

A ***concurrent execution of several transaction trees***, referred to as a multilevel schedule, is essentially an interleaved forest of the individual transaction trees. This is illustrated in the figure below, for two transactions with record-level and page-level operations, in the same spirit as the previous example but with some simplifications. Like before, the ordering of operations is indicated by drawing the leaf nodes in their execution order from left to right (assuming total ordering of leaves for simplicity). As the caller-callee relationship in transaction trees is captured by vertical or diagonal arcs, the crossing of such arcs indicates that two (non-leaf) operations are concurrent. In the figure the two transactions t1 and t2 are concurrent, and also the store and fetch operations execute concurrently and the same holds for the last two modify operations.

To reason about the correctness of such interleavings, it is first necessary to define the ordering of non-leaf operations: node o1 precedes o2 in the execution, o1 < o2, if all leaf-level descendants of o1 precede all leaf-level descendants of o2. The forest of labeled trees and this execution order < define a ***multilevel schedule***, more generally, a schedule of transaction trees.

Following the standard argumentation about serializability for conventional, "flat" transactions, the goal for a correct schedule is to show that the execution is equivalent to a sequential one based on a notion of ***conflicting versus non-conflicting operations***. Usually, ***commutativity*** properties of operations are the basis for defining conflict relations. In the example, this suggests that store and fetch operations on different objects as well as pairs of modify operations on different objects are non-conflicting (even if their implementations write the same page). Thus, their observed execution order could be changed, by swapping adjacent operations, without changing the overall effect of the schedule. But applying this principle to, for example, the store and fetch operations in the above schedule does not work because these two nodes are composite operations (i.e., non-leaf nodes) and executed concurrently among themselves. To disentangle the concurrency between these operations, one needs to reason about the execution ordering of their children, and in an actual system, one would need a ***lower-level concurrency control*** mechanism that treats the two operations as ***subtransactions***. The goal of this disentangling, the counterpart to serial schedules in conventional concurrency control theory, are ***isolated subtrees*** for the two operations. A subtree rooted at node o is isolated if there is a total ordering among all its leaf-level descendants and o either precedes or follows all other operations o' that are not among its descendants (o < o' or o' < o). Once a subtree is isolated, the fact that its root is a composite operation is no longer important, and it is possible, for reasoning about equivalent executions, to ***reduce an isolated subtree*** to its root alone. This argument abstracts from the lower-level executions, as they are now (shown to be equivalent to) sequential.

Putting everything together, the above considerations lead to three rules for transforming a multilevel schedule into equivalent and abstracted executions, ideally leading to a sequential execution of the transaction roots:

- *Commutativity rule:* The order of two ordered leaf operations p and q with, say, the order p < q, can be reversed provided that
  - both are isolated, adjacent in that there is no other operation r with p < r < q, and commutative,
  - the operations belong to different transactions, and
  - the operations p and q do not have ancestors, say p' and q', respectively, which are non-commutative and totally ordered (in the order p' < q').
- *Ordering rule:* Two unordered leaf operations p and q can be (arbitrarily) ordered, i.e., assuming either p < q or q < p, if they are commutative.
- *Tree pruning rule:* An isolated subtree can be pruned and replaced by its root.

A schedule that, by applying the above rules, can be transformed into a sequential execution of the transaction roots is called ***tree-reducible*** or ***multilevel serializable***. Note that this notion of multilevel serializability is much more liberal than the conventional notion of read-write-oriented serializability. The example schedule shown above is not serializable at the leaf level of read and write operations (i.e., if one ignored the level of search, fetch, store, and modify operations and simply connected all leaves directly to the roots), but these seemingly non-serializable effects on the low-level storage structures are irrelevant as long as they are properly handled within the scope of their parent operations and new transactions access the data through the higher-level operations like search, fetch, store, and modify.

The example schedule depicted above is multilevel serializable. It can be reduced as follows. First the two reads of the fetch(x) operation are commuted with their left-hand neighbors so that fetch(x) completely precedes store(z); analogously the r(t) step of modify (y) is commuted with its right-hand neighbors, the children of modify(w), so that modify(w) completely precedes modify(y). This establishes a serial order of the record-level operations, all of them now being isolated subtrees. This enables the application of the pruning rule to remove all page-level operations. Next, the fetch(x) operation of t2 is commuted with t1's store(z), modify(y), and modify(w) operations all the way to the right, producing an order where all of t1's operations precede all of t2's operations. This turns t1 and t2 into isolated subtrees. Finally, pruning the operations of t1 and t2 produces the sequential order of the transaction roots: t1 < t2.

The transformation rules do not directly lead to an efficient concurrency control protocol. Rather their purpose is to prove the correctness of protocols. But for the case of a layered system, the way the example was handled points towards a practically viable protocol. The key is to consider pairs of adjacent levels and apply the transformation rules in a bottom-up manner. So first, the commutativity and ordering rules are used to establish a sequential execution of the parent nodes of the leaf-level nodes, then these isolated parents are reduced. Then, with the lowest level removed, this procedure is iterated through the levels until the roots of the entire transaction trees are isolated. This proof strategy can be directly turned into a protocol by enforcing conventional order-preserving conflict-serializability (OPCSR) for each pair of adjacent levels. Any protocol for OPCSR can be used, and even different protocols for different level pairs are possible. The most widely used protocol, two-phase locking, is often a natural choice, and then forms the following ***multilevel locking protocol***:

- *Lock acquisition rule:* When an operation f(x) is issued, an f-mode lock on x needs to be acquired before the operation can start its execution.
- *Lock release rule:* Once a lock originally acquired by an operation f(x) with parent o (an operation at the next higher level) is released, no other descendant of o is allowed to acquire any locks.
- *Subtransaction rule:* At the termination of an operation o, all locks that have been acquired for descendants of o are released, thus treating o as a committed subtransaction. Note that the o-lock for o itself is still kept – until the parent of o terminates. The releasing of lower-level locks at the end of a subtransaction is the origin of the name "open nested transaction".

A possible execution of the example schedule under this multilevel locking protocol is shown in the figure below (with levels L1 and L0 referring to the record and page layer in a database engine, and t_{ij} denoting the jth subtransaction of transaction t_i).

The example shows that, despite many page-level conflicts, high concurrency is possible by exploiting the finer granularity and richer semantics of record-level operations. These benefits are even more pronounced for index-key operations. For this case, highly optimized special-purpose protocols like ARIES Key-Value Locking have been developed. One important optimization for both record and index operations is that the subtransactions may use light-weight latching instead of full-fledged locks.

Another use case with wide applicability are operations on counters, such as increment and decrement or conditional variants on lower-bounded or upper-bounded counters. Such objects and operations are common in reservation systems, inventory control, financial trading, and so on. The relaxed (but not universal) commutativity properties of the operations can be leveraged for very high concurrency even if operations access the same object. Again, special implementation techniques like escrow locking have been developed for these settings. When counter operations have a composite nature, e.g., by automatically triggering updates on other objects, then the special commutativity techniques need to be embedded in a multilevel transaction framework.

A complication that arises from all these high-concurrency settings is that undo recovery (for transaction abort and to wipe out effects of incomplete transactions after a crash) can no longer be implemented merely by restoring prior page versions. Instead, adequately implemented forms of inverse operations need to be executed. Together with the composite nature of operations, this necessitates a form of multilevel recovery.

Most of the outlined principles and algorithms apply to the general case of object-model transactions as well. However, the absence of a layering does incur some extra difficulties, which are beyond the scope of this entry. The algorithms for fully general object-model transactions are explained in detail in the textbook [14].

Future Directions

Multilevel transactions have originally been developed in the database system context, but their usage and potential benefits are by no means limited to database management. So not surprisingly, object-model transactions and related concepts are being explored in the operating systems and programming languages community. Recent trends include, for example, enhancing the Java language with a notion of atomic blocks that can be defined for methods of arbitrary classes. This could largely simplify the management of concurrent threads with shared objects, and potentially also the handling of failures and other exceptions. The runtime environment could be based on an extended form of software transactional memory [8].

Another important trend is to enhance composite web services with transactional properties. Again, object-model transactions is a particularly intriguing paradigm because of its flexibility in allowing application-specific methods for providing atomicity, isolation, and persistence. Adapting and extending transactional concepts for web services and combining them with other aspects of service-oriented computing is the subject of ongoing research [15].

Cross-references

▶ Atomicity
▶ Concurrency Control
▶ Escrow Transactions
▶ Key Value Locking

- ► Locking
- ► Multi-Level Recovery and the ARIES Algorithm
- ► Nested Transaction Models
- ► System Recovery
- ► Transaction
- ► Transaction Management

Recommended Reading

1. Beeri C., Bernstein P.A., and Goodman N. A model for concurrency in nested transactions systems. J. ACM, 36(2):230–269, 1989.
2. Davies C.T. and Davies C.T. Jr. Data processing spheres of control. IBM Syst. J., 17(2):179–198, 1978.
3. Gray J. and Reuter A. Transaction processing: concepts and techniques. Morgan Kaufmann, Los Altos, CA, 1993.
4. Greenfield P., Fekete A., Jang J., Kuo D., and Nepal S. Isolation support for service-based applications: A position paper. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 314–323.
5. Lomet D.B. MLR: a recovery method for multi-level systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1992, pp. 185–194.
6. Mohan C., Haderle D.J., Lindsay B.G., Pirahesh H., and Schwarz P.M. ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. Database Syst., 17(1):94–162, 1992.
7. Moss J.E.B., Griffeth N.D., and Graham M.H. Abstraction in recovery management. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 72–83.
8. Ni Y., Menon V., Adl-Tabatabai A.-R., Hosking A.L., Hudson R.L., Moss J.E.B., Saha B., and Shpeisman T. Open nesting in software transactional memory. In Proc. 12th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, 2007, pp. 68–78.
9. Ramamritham K. and Chrysanthis P.K. A taxonomy of correctness criteria in database applications. VLDB J., 5(1):85–97, 1996.
10. Schuldt H., Alonso G., Beeri C., and Schek H.-J. Atomicity and isolation for transactional processes. ACM Trans. Database Syst., 27(1):63–116, 2002.
11. Weikum G. Principles and realization strategies of multilevel transaction management. ACM Trans. Database Syst., 16(1):132–180, 1991.
12. Weikum G. and Hasse C. Multi-level transaction management for complex objects: implementation, performance, parallelism. VLDB J., 2(4):407–453, 1993.
13. Weikum G. and Schek H.-J. Architectural Issues of Transaction Management in Multi-Layered Systems. In Proc. 10th Int. Conf. on Very Large Data Bases, 1984, pp. 454–465.
14. Weikum G. and Vossen G. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, Los Altos, CA, 2001.
15. Zimmermann O., Grundler J., Tai S., and Leymann F. Architectural Decisions and Patterns for Transactional Workflows in SOA. In Proc. 5th Int. Conf. Service-Oriented Computing, Germany, 2007, pp. 81–93

Multi-Level Visualization

- ► Visualizing Hierarchical Data

Multilingual Information Retrieval

- ► Cross-Language Mining and Retrieval

Multi-Master System

- ► Optimistic Replication and Resolution

Multimedia

- ► Image
- ► Image Representation
- ► Video

Multimedia Content Enrichment

- ► Automatic Image Annotation

Multimedia Data

RAMESH JAIN
University of California-Irvine, Irvine, CA, USA

Synonyms

Multimodal data

Definition

Multimedia in principle means data of more than one medium. It usually refers to data representing multiple types of medium to capture information and experiences related to objects and events. Commonly used forms of data are numbers, alphanumeric, text, images, audio, and video. In common usage, people refer a data set as multimedia only when time-dependent data such as audio and video are involved.

Historical Background

In early stages of computing, the major applications were scientific computations. In these applications, computers dealt with numbers and were programmed to carry out a sequence of calculations to solve a scientific problem. As people realized power of computing, new applications started emerging. Alphanumeric data was the next type of data to be used in different applications. In early days, these applications were mostly related to businesses. These applications were mostly to store large volumes of data to find desired information from this data set. These applications were the motivation of the development of current database technology.

Text is a special case of alphanumeric data. In text, there is a large string of alphanumeric data that humans associate with written language. Text has been the basis of written human communication and has become one of the most common data form. Most of the information and communication among humans takes place in text.

Next data type to start appearing on computers was images. Images started in many applications where they needed to be analyzed as well as in applications where computers were used to create and display images. Image processing and computer vision emerged as fields dealing with image analysis and understanding while computer graphics emerged as a field dealing with creation and display of images. Images were initially represented in two ways: a list of lines (called vectors) and a 2-dimensional array of intensity values. The second method has now become the most common method of representing images. Images represent a more complex data type because people perceive not the data, which is really a large collection of intensity values, but what the data represents. In computer generated images, the semantics of the pixels is determined and is known at the creation time. In all other images, the semantics must be determined. Computer vision researchers have been developing tools to automatically determine this semantics and have made progress. However, segmenting an image to determine objects in it has been a difficult problem and in general remains an unsolved problem.

Audio data represents variation of a signal over time. Signal processing deals with many types of signals, but due to its closeness to human perception, audio became an important signal type. Unlike regular numerical, alphanumerical, and image data, audio is time-varying or time dependent data. Like images, the numbers have semantics only when they are rendered, in this case using a speaker, to a human. Both images and audio are a collection of numbers that have strong semantics associated with them. This semantics can be associated only by segmenting the data and identifying each segment. Video is next in this sequence of semantic richness. Video is a time-dependent sequence of images synchronized with audio. This means that it brings with it enormous volume of data and richness of semantics.

In late 1980s, people started using the term multimedia to denote combination of text, audio, and video. This gained popularity because the technology had advanced enough to combine these media to articulate thoughts, messages, and stories using appropriate combination of these components and present them easily on computers, save them on CDs, and transmit and receive them using compression/decompression and streaming technologies. By the year 2000, multimedia had become a common data form on computers and Internet.

Foundations

Multimedia data is fundamentally different than the data traditional databases normally manage. Some fundamental differences in multimedia data are discussed here by considering several aspects.

Types and Semantics

The data in early generation databases was either a number or a string of characters. Each data item usually represented value of an attribute. These attribute had clear and explicit semantics in the applications that used the data.

Multimedia data may be considered to be composed of numbers or strings. So an image may be viewed as a two dimensional array of integers. In multimedia applications, however, the semantics is not defined and used at the level of such basic types as in traditional applications. An image is usually considered an image that contains certain objects that are characterized by regions in the image. The relationships among these regions should also be captured. Depending on the context and an application, the semantics associated with an image may change and may need to be represented differently. Similarly an audio file may be viewed as a collection of phonemes rather than just integer values at a time instant representing sound energy. Video is a synchronized combination of audio and images. But if a video is considered just a combination of separate sound energy and images, then the semantics of video is lost. The semantics of video is due to synchronized combination of its components rather than individual elements.

Multimedia types cannot be considered simply by considering its atomic components. One must consider whole data. The data types and the semantics of multimedia data are the result of the "multi" and are not present in single (mono) medium that may be part of the whole data.

Gestalt philosophy is in action in multimedia data: the whole is bigger than the sum of its parts.

Sequence and Order

Many components of multimedia data are measurements using some sensors. These sensors measure some attribute of physical world. These measurements represent the attribute at a point in space at a particular time. The semantics of the data is intimately tied to the space and time underlying the data. The data is usually organized in the time sequence as it is acquired over some predefined spatial ordering of its acquisition using multiple sensors covering the space of interest.

Multimedia data could be archived data or live data. Archived data is the one that was acquired and stored and hence comes from a server. Live data is presented as it is being acquired. Live data is increasingly being used in many applications.

Size

Multimedia data is voluminous. Audio, Images, and video are much larger in size than alphanumeric data and text. Usually the size of traditional data can be measured in bytes to Kilobytes. Images usually, even the regular amateur photographs run into Megabytes and video easily runs into Gigabytes to Terabytes. Due to the size of the multimedia data, it is usually stored and transmitted in compressed form. For analysis and use of the data, it must be usually decompressed.

Meta data plays a significant role in the analysis of multimedia data and is commonly stored as part of the dataset. Metadata can be of two types: about context or about content. Contextual metadata is about the situation of the real world and the parameters of devices used in acquiring the data. Content related metadata is obtained either thru analysis of the data or by human annotation or interpretation of data.

Many different standards have evolved for compression of multimedia data and association and storage of metadata. Usually these standards related to the medium and are developed by international standards body. Some commonly used standards are JPEG for images, MP3 for audio, and MPEG for video.

Accessing Multimedia Data

Each multimedia data is usually large and represents measurements acquired using a sensor over space and time. Even an image is acquired at a location at a particular time and also contains measurements performed in space using an array of pixel. Each pixel represents measurements related to a particular point in three-dimensional space. Each image or audio video is usually represented as a separate file. This file may contain raw measurements in original form or in compressed form and may also contain associated metadata such as in EXIF data for photos acquired using digital cameras.

Multimedia data representing a measurement is usually represented as one file. In databases such data is usually represented as a pointer to the file, as a BLOB, or the name of the file.

In most current applications, multimedia data is accessed based on the metadata. All queries are formed based on metadata and then the correct file is retrieved and presented. The granularity at which multimedia data is accessed is at the level of file. Text search became so useful when it was applied to documents by analyzing and indexing all areas of a document. This content analysis and indexing based on content within a file will be very useful in multimedia data also. Research in content analysis of multimedia documents for content-based retrieval is an active research area currently.

Presentation

Multimedia data must be presented to a user by sending it to appropriate devices. Audio must be sent to speakers and images and video should also be displayed using special display programs. Displaying raw data in a file is not useful to users. In most cases, before displaying the data, it must be decompressed.

Considering large files and copyright issues, many times multimedia data is not transferred to users for storage, users are allowed to see or listen it only once each time a display request is made. Such playback of data is commonly called streaming of data and is commonly used with video. In streaming, the data from server is sent to a client only for displaying it once. This is also used in the context of live data also.

Key Applications

Computing at one time was mostly numeric, then it became alphanumeric. Now it is multimedia. Almost all applications in computing now deal with multimedia data. In a sense, the term multimedia was a good term to use in the last decade, but now it is a redundant term. In early days of computing there were two types of computing: analog and digital. Slowly all computing became digital. Now no body normally uses the term digital computing because all computing is digital. In the same way all computing ranging from scientific to entertainment will use multimedia data and hence the term multimedia data or multimedia computing will shed "multimedia" and simply become data and computing.

Future Directions

Multimedia data has already become ubiquitous. With the increasing popularity of mobile phones with camera, digital cameras, and falling prices of sensors of different kinds multimedia data is becoming as widespread as alphanumeric data. Considering the current trend and human dependence on sensory data, it is likely that soon multimedia data will become more common than the traditional alphanumeric data. In terms of volume, multimedia data already may be far ahead of alphanumeric data.

Most of the current techniques which deal with multimedia data have two major limitations: first they mostly rely on metadata for access and they treat each type, such as images, audio, and video, as a separate type and hence create silos. What is required is dealing with all data, alphanumeric as well as different types of multimedia, as the data related to some physical objects or situations. This unified approach will treat all data in a unified manner and will not distinguish between media. Each media will be considered only as a source helping understand an object or a situation.

Cross-references

► Multimedia Databases

Recommended Reading

1. Jain R. Experiential computing. Commn. ACM, 46(7):48–55, 2003.
2. Kankanhalli M.S., Wang J., and Jain R. Experiential sampling in multimedia systems. IEEE Trans. Multimed., 8(5):937–946, 2006.
3. Rowe L. and Jain R. ACM SIGMM retreat report on future directions in multimedia research. ACM Trans. Multimedia Comp., Comm., and Appl., 1(1):3–13, 2005.
4. Steinmetz R. and Nahrstedt K. Multimedia Fundamentals: Media Coding and Content Processing (IMSC Press Multimedia series). Prentice Hall, 2002.

Multimedia Data Buffering

Jeffrey Xu Yu
Chinese University of Hong Kong, Hong Kong, China

Definition

Multimedia data are large in size and reside on disks. When users retrieve large multimedia data, in-memory buffers are used to reduce the number of disk I/Os, since memory is significantly faster than disk. The problem to be studied is to efficiently make use of buffers in the multimedia system to reduce the number of I/Os in order to get a better performance when multiple users are retrieving multiple multimedia data simultaneously. Existing works on multimedia data buffering focus on either the replacement algorithms to lower the number of cache misses or the buffer sharing algorithms when many simultaneous clients reference the same data item in memory.

Historical Background

Early works on multimedia data buffering focus on replacement algorithms to reduce the number of cache misses. Although in the traditional database systems, a number of different buffer replacement algorithms, such as the least recently used (LRU) and most recently used (MRU) algorithms are used to approximate the performance behavior of the optimal buffer replacement algorithm [1,2,6,8,15]. They do not reduce disk I/O significantly when they are used in a multimedia database system. Many new buffer replacement algorithms are proposed to save as much of the reserved disk bandwidth for continuous media data as possible. In [5], the effects of various buffer replacement algorithms on the number of glitches experienced by clients are studied. In [9], the authors introduce two buffer replacement algorithms, namely, the basic replacement algorithm (BASIC) and the distance-based replacement algorithm (DISTANCE), for multimedia database systems, which have a much better performance in comparison with LRU and MRU schema.

In terms of buffer sharing, a simple buffer replacement strategy may miss some opportunities to share memory buffers [14]. A straightforward use of LRU or LRU-k [6,8] is shown to be inadequate [7]. Bridging [3,4,10–12] as a new technique is studied to facilitate data sharing in memory, but it can degrade the system performance. In [13], the authors observe that an uncontrolled buffer sharing scheme may reduce system performance, and introduce the Controlled Buffer Sharing (CBS), which can trade memory for disk bandwidth in order to minimize cost per stream.

Foundations

Buffer Replacement

Assume that each buffer in the buffer space of a system is of the same size and is tagged as either free or used. All the free buffers are kept in a free buffer pool. In order to meet the rate requirement for clients, the system must pre-fetch the required data block from disks into the buffer space, so that the required piece of data is already in the buffer space before being read. In each service cycle, the system first moves the buffers containing data blocks that were already consumed in the last service cycle to the free buffer pool, then determines which data block need to be pre-fetched from disk to the buffer next. If the block is not in the buffer space, then it allocates buffers, from the set of free buffers for the block and issues disk I/O to retrieve the needed data block from disk into the allocated buffers. The algorithm to decide which of the buffers should be allocated is referred to as the buffer replacement algorithm. Several general replacement algorithms are listed below, which are widely used in database management systems. (i) LRU: when a buffer is to be allocated, the buffer containing the block that is used least recently is selected. (ii) MRU: when a buffer is to be allocated, the buffer containing the block that is used most recently is selected. (iii) Optimal: when a buffer is to be allocated, the buffer containing the block that will not be referenced for the longest period of time is selected. Since arrival, pause, resume and jump time when playing an object are unknown in advance, the optimal algorithm can only be implemented for simulation studies.

For the multimedia database systems, the commonly used LRU and MRU algorithms may not reduce disk I/O significantly. Two buffer replacement algorithms are proposed. They are the basic replacement algorithm (BASIC) and the distance-based replacement algorithm (DISTANCE).

The BASIC Buffer Replacement Algorithm The main idea behind the BASIC buffer replacement algorithm [9] is as follows. It is possible to estimate the duration by assuming each client will remain its consumption rate for a long period, even though it is difficult to decide which block will not be referenced for the longest period of time. It assumes that clients continue to consume data at the specified rate they are accessing the blocks. When there is a new request to allocate a buffer, the BASIC algorithm selects the buffer containing the block that will not be accessed for the longest period of time. If there are several such buffers, the algorithm will select the block with the highest offset-rate ratio (the ratio of offset/rate) to be replaced. The BASIC algorithm may reduce the miss ratio to nearly optimal, but it requires to sort clients and free buffers in the increasing order of their offset, which make the overhead of the BASIC algorithm very high. The DISTANCE algorithm is proposed to handle the overhead.

The DISTANCE Buffer Replacement Algorithm The main idea behind the DISTANCE buffer replacement algorithm is based on distance between clients [9]. Suppose that there are clients, c_1,c_2,..., accessing the same media data, M. Assume that each client, c_i, is accessing the M at a certain position of M, denoted as $p_i(M)$, and the data block on disk starting from $p_i(M)$ is kept in a buffer, B_i. Let all clients that are accessing the same media data M be sorted in order, c_1,c_2,.... Here, c_i is accessing M ahead of c_j if $i < j$, or in other words, $p_i(M) > p_j(M)$, because c_i has already accessed $p_j(M)$ and is now accessing $p_i(M)$. The distance between c_i and its next c_{i+1} is denoted as $dist_i$ which is equal to $p_i(M) - p_{i+1}(M)$. Note that the distance d_i is a value associated with the client c_i. Suppose all clients c_1,c_2,..., are accessing their blocks in the buffers in the current cycle. They all need to move ahead and access the next data blocks. The question becomes which buffer they are accessing in the current cycle needs to be freed if the buffer is full. In brief, the buffers consumed by a client, c_i, will be kept longer if the next client, c_{i+1}, will need them shortly (small distance $dist_i$). The buffers consumed by a client, c_j, will be freed earlier if the next client, c_{j+1}, does not need to access the data block $p_j(M)$, that c_j has just accessed, shortly (large distance $dist_j$). The DISTANCE algorithm frees buffers consumed by clients in the previous cycle in the decreasing order of clients' $dist_i$. In other words, when a new buffer needs to be allocated and there are no free buffers, a buffer consumed by a client, which will not be accessed by its next client shortly, based on the distance between clients, will be selected as a victim to be freed.

The DISTANCE algorithm can be implemented by dynamically maintaining a client list which is ordered in the decreasing order of clients' $dist_i$. The overhead is lower than the BASIC algorithm.

Table 1 shows the comparison of overhead and cache misses of different buffer replacement algorithms, n_B is the number of buffers used.

Buffer Sharing

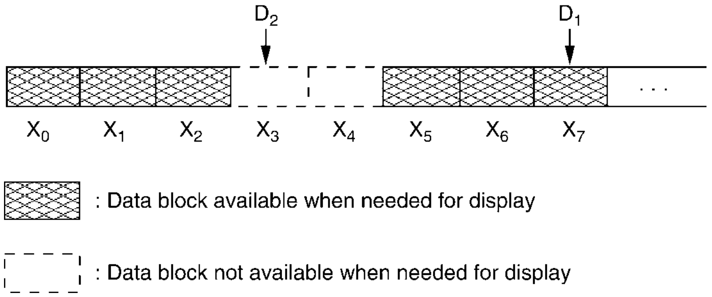
Consider buffer sharing, where cached data can be shared among all the clients. A naive approach is to use LRU or LRU-k, which is shown to be inadequate to efficiently share data. An example is given in Fig. 1. There are two displays, D_1 and D_2, and both reference different blocks of the same clip. With LRU as a global buffer pool replacement policy, the blocks accessed by D_1 may be discarded before D_2 needs to access.

Bridging [3,4,10–12] as a technique is to form a bridge between the data blocks staged by two different clients referencing the same clip, which enables them to share memory and use one disk stream. As shown in Fig. 2, two displays D_1 and D_2 are supported using a single disk stream. The distance between D_1 and D_2 is 5. With the bridging technique, it holds the intermediate data pages between D_1 and D_2 in the buffer pool, and does not swap these pages out from the buffer pool. However, as analyzed in [11,12], a potential problem is that a simple bridging may possibly exhaust the available buffer space, which will have great impacts on the system performance.

A Controlled Buffer Sharing (CBS) technique is proposed in [13], which increases disk bandwidth using memory in order to achieve two objectives, namely, minimization of cost per simultaneous stream, and balancing memory and disk utilization. The latter considers that unlimited memory consumption may in fact degrade the system performance. The framework of CBS is shown in Fig. 3. The framework consists of three components: a configuration planner, a system generator, and a buffer management technique. The configuration planner determines the amount of required buffer and disk bandwidth in support of a prespecified performance objective. The system generator simply acts as a multiplier. The first two components

Multimedia Data Buffering. Table 1. Overhead and cache misses of different buffer replacement algorithms (Table 1 in [9])

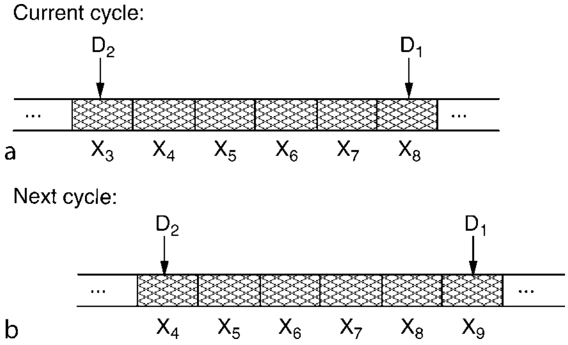
n_B	LRU	MRU	BASIC	DISTANCE	# of refs
300	13:48 s/670,080	13:44 s/668,974	33:30 s/638,974	11:89 s/641,274	670,080
600	13:32 s/670,080	13:18 s/665,748	1:12min/595,416	10:96 s/599,214	670,080
1,200	16:80 s/665,934	16:27 s/657,634	3:41min/549,570	12:28 s/554,640	670,080
2,400	13:22 s/654,240	12:32 s/642,914	5:31min/480,068	8:64 s/481,364	670,080


D_2
D_1
. . .
X_0
X_1
X_2
X_3
X_4
X_5
X_6
X_7
: Data block available when needed for display
: Data block not available when needed for display

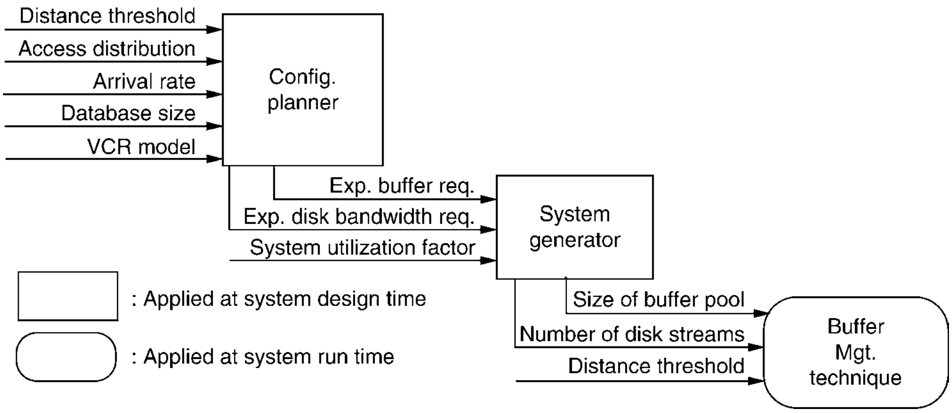
Multimedia Data Buffering. Figure 1. Two displays may compete for buffer frames with LRU (Fig. 1 in [13]).

are applied off-line to determine the system size. The buffer management technique controls the memory consumption at run time.

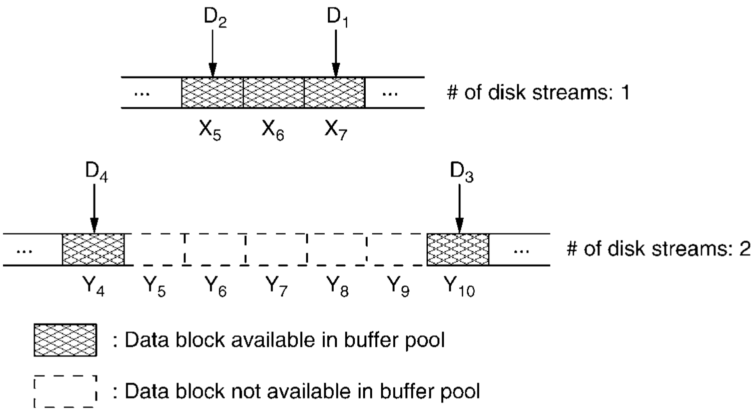
In the CBS framework, a distance threshold, d_t, is used to capture the cost of memory and disk bandwidth and control the number of pinned buffer blocks between two adjacent displays that access the same clip. As shown in Fig. 4, suppose $d_t = 5$, D_1 and D_2 can share one disk stream because their distance is below the specified threshold, while D_3 and D_4 cannot share one disk stream because their distance exceeds the threshold.



Multimedia Data Buffering. Figure 2. Bridging (Fig. 2 in [13]).



Multimedia Data Buffering. Figure 3. The CBS Scheme (Fig. 4 in [13]).



Multimedia Data Buffering. Figure 4. The effectiveness of distance threshold ($d_t = 5$) (Fig 5 in [13]).

Key Applications

Buffering is widely used in retrieving and playing multimedia data, especially for network continuous media applications, where multiple users may need to display multiple medias simultaneously.

Cross-references

- ▶ Buffer Management
- ▶ Buffer Manager
- ▶ Continuous Multimedia Data Retrieval
- ▶ I/O Model of Computation
- ▶ Multimedia Data Buffering
- ▶ Multimedia Data Storage
- ▶ Multimedia Resource Scheduling

Recommended Reading

1. Chew K.M., Reddy J., Romer T.H., and Silberschatz A. Kernel support for recoverable-persistent virtual memory. In Proc. USENIX MACH III Symposium, 1993, pp. 215–234.
2. Chou H.T. and DeWitt D.J. An evaluation of buffer management strategies for relational database systems. In Proc. 11th Int. Conf. on Very Large Data Bases, 1985, pp. 127–141.
3. Dan A., Dias D.M., Mukherjee R., Sitaram D., and Tewari R. Buffering and caching in large-scale video servers. In Digest of Papers - COMPCON, 1995, pp. 217–224.
4. Dan A. and Sitaram D. Buffer management policy for an on-demand video server. IBM Research Report RC 19347.
5. Freedman C.S. and DeWitt D.J. The SPIFFI scalable video-on-demand system. ACM SIGMOD Rec., 24(2):352–363, 1995.
6. Lee D., Choi J., Kim J.H., Noh S.H., Min S.L., Cho Y., and Kim C.S. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (IFU) policies. SIGMETRICS Perform. Eval. Rev., 27(1):134–143, 1999.
7. Martin C. Demand paging for video-on-demand servers. In Proc. Int. Conf. on Multimedia Computing and Systems, 1995, pp. 264–272.
8. O'Neil E.J., O'Neil P.E., and Weikum G. The LRU-K page replacement algorithm for database disk buffering. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 297–306.
9. Özden B., Rastogi R., and Silberschatz A. Multimedia Information Storage and Management, chap. 7: Buffer Replacement Algorithms for Multimedia Storage Systems. Kluwer Academic, 1996.
10. Rotem D. and Zhao J.L. Buffer management for video database systems. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 439–448.
11. Shi W. and Ghandeharizadeh S. Buffer sharing in video-on-demand servers. SIGMETRICS Perform. Eval. Rev., 25(2):13–20, 1997.
12. Shi W. and Ghandeharizadeh S. Trading memory for disk bandwidth in video-on-demand servers. In Proc. 1998 ACM Symp. on Applied Computing, 1998, pp. 505–512.
13. Shi W. and Ghandeharizadeh S. Controlled Buffer Sharing in Continuous Media Servers. Multimedia Tools Appl., 23(2):131–159, 2004.
14. Christodoulakis S., Ailamaki N., Fragonikolakis Y., and Koveos L. Leonidas K. An object oriented architecture for multimedia information systems. Data Eng., 14(3):4–15, 1991.
15. Stonebraker M. Operating system support for database management. Readings in database systems (3rd ed.), Morgan Kaufmann, San Francisco, CA, USA, pp. 83–89, 1998.

Multimedia Data Indexing

Paolo Ciaccia
University of Bologna, Bologna, Italy

Synonyms

MM indexing

Definition

Multimedia (MM) data indexing refers to the problem of preprocessing a database of MM objects so that they can be efficiently searched for on the basis of their content. Due to the nature of MM data, indexing solutions are needed to efficiently support *similarity queries*, where the similarity of two objects is usually defined by some expert of the domain and can vary depending on the specific application. Peculiar features of MM indexing are the intrinsic high-dimensional nature of the data to be organized, and the complexity of similarity criteria that are used to compare objects. Both aspects are therefore to be considered for designing efficient indexing solutions.

Historical Background

Earlier approaches to the problem of MM data indexing date back to the beginning of 1990s, when it became apparent the need of efficiently supporting queries on large collections of non-standard data types, such as images and time series. Representing the content of such data is typically done by automatically extracting some low-level features (e.g., the color distribution of a still image), so that the problem of finding objects similar to a given reference one is transformed into the one of looking for similar features. Although, at that time, many solutions from the pattern recognition field were available for this problem, they were mainly concerned with the *effectiveness* issue (which features to consider and how to compare them), thus almost disregarding *efficiency* aspects.

The issue of making similarity query processing scalable to large databases was first considered in systems like QBIC [6] for the indexing of color images and by more focused approaches such as the one described by Jagadish in [8] for indexing shapes. Not surprisingly, these solution adopted index methods available at that time that had been developed for the case of low-dimensional spatial databases, such as R-trees and Grid files. The peculiarity of MM data then originated a flourishing brand new stream of research, which resulted in many indexes explicitly addressing the problems of high-dimensional features and complex similarity criteria.

Foundations

Figure 1 illustrates the typical scenario to be dealt with for indexing multimedia data. The first step, *feature extraction*, is concerned with the problem of highlighting those relevant features, f_i, of an object o_i on which content-based search wants to be performed. In the figure, this is the shape of the image subject (a cheetah). The second step, *feature approximation*, is optional and aims to obtain a more compact representation, af_i, of f_i that can be inserted into a suitable index structure (third step). It has to be remarked that, while feature extraction is needed to define which are the relevant aspects of objects on which the search has to focus on, feature approximation is mainly motivated by feasibility and efficiency reasons. This is because it might not be possible to directly index non-approximate features and/or indexing approximate features might result in a better performance of the search algorithms.

Consider a collection $O = \{o_1, o_2, ..., o_n\}$ of MM objects with corresponding features $F = \{f_1, f_2, ... f_n\}$ and approximate features $AF = \{af_1, af_2, ... af_n\}$. In order to compare features, a *distance function* d is typically set up, where $d(f_i, f_j)$ measures how dissimilar are the feature values of objects o_i and o_j. Given a reference object q (the *query point*), a range query with radius $\in$, also called an $\in$-similarity query, will return all the objects $o_i \in O$ such that $d(f_i, f(q)) \leq \in$, whereas a k-nearest neighbor query (k-NN) will return the k objects in O whose features are closest to those of q.

A simple yet remarkable result due to Agrawal, Faloutsos, and Swami [1], and now popularly known as the lower-bounding lemma, provides the basis for exactly solving queries by means of an index that organizes approximate features:

The lower-bounding lemma. Let I be an index that organizes the set of approximate features $AF = \{af_1, af_2, ... af_n\}$ and that compares such features using an *approximate distance* d_{appr}. If, for any pair of objects, it is $d_{appr}(af_i, af_j) \leq d(f_i, f_j)$, then the result of a range query obtained from I is guaranteed to contain the exact result, i.e., no false dismissals are present.

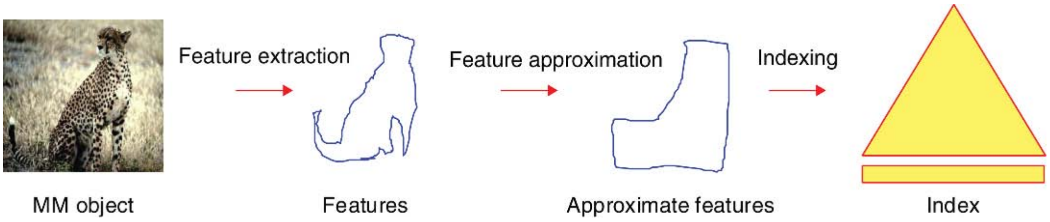
The result easily follows from the observation that, since d_{appr} lower bounds d by hypothesis, $d(f_i, f(q)) \leq \in$ implies $d_{appr}(af_i, af(q)) \leq \in$. The lower-bounding lemma guarantees that querying the index with a search radius equal to $\in$ will return a result set that contains all the objects whose non-approximate features satisfy the query constraint.

Filter & Refine

When indexing is based on an approximate distance, a two-step *filter & refine* process is therefore needed, in which the role of the index is to filter out many irrelevant objects. The so-resulting candidate objects then need to be verified by using the actual distance d. The lower-bounding lemma is also the key for solving k-NN queries using a multi-step query processing approach.

The effectiveness of the filter & refine approach depends on two contrasting requirements:

1. The approximate distance function d_{appr} should be a tight approximation of d, in order to minimize the number of false hits, i.e., those objects that do not satisfy the query constraint yet the index is not able to discard them. These are exactly those objects o_i for which both $d_{appr}(af_i, af(q)) \leq \in$ and $d(f_i, f(q)) > \in$ hold.



Multimedia Data Indexing. Figure 1. The multimedia data indexing scenario.

2. At the same time, d_{appr} should be relatively cheap to compute as compared to d, in order to avoid wasting much time in the filter phase.

The literature on MM indexing abounds of examples showing how to derive effective approximations for complex distance functions. For instance, the QBIC system compares color images using a quadratic form distance function, $d_A^2(f_i, f_j) = (f_i - f_j)A(f_i - f_j)^T = \sum_{k=1}^{D} \sum_{l=1}^{D} a_{k,l}(f_{i,k} - f_{j,k})(f_{i,l} - f_{j,l})$, where $A = (a_{k,l})$ is a color-to-color similarity matrix and features are color histograms. Evaluating d_A has complexity $O(D^2)$, which becomes too costly even for moderately large values of D, the number of bins in the color histograms. In [6] it is demonstrated that using as approximate features the average RGB color of an image, which is a three-dimensional vector, and comparing average colors using the Euclidean distance, i.e., $d_{avg}^2(af_i, af_j) = \sum_{k \in \{R,G,B\}}(af_{i,k} - af_{j,k})^2$, leads to derive that $d_{avg}^2 \leq {d_A}^2/\lambda_1$, where λ_1 is the smallest eigenvalue of matrix A. Then, the lower-bounding lemma guarantees that querying an index built on average colors with a range query of radius $\epsilon/\sqrt{\lambda_1}$ will not lead to any false dismissal.

As another relevant example, consider the problem of comparing feature vectors that represent time-varying signals. A distance function more robust than the Euclidean one to misalignments on the time domain is the dynamic time warping (DTW) distance. However, evaluating DTW has a complexity $O(D^2)$, which is untenable for long time series. In [9] Keogh introduces an effective lower-bounding function for the DTW distance. In essence, the idea is to construct an *envelope*, $Env(q)$, around the query time series q, after that an Euclidean-like distance between $Env(q)$ and a stored sequence f_i can be computed in $O(D)$ time.

The Need for Approximate Features

As anticipated, there are several reasons for which approximate features might have to be considered. First, in many relevant cases the features of a MM object are represented through a high-dimensional vector, $f_i = (f_{i,1}, f_{i,2}, \ldots, f_{i,D})$, with D of the order of the hundreds or even thousands. At such high dimensions it is known that the performance of multidimensional indexes rapidly deteriorates, becoming either comparable to, or even worse than, that of a sequential scan. This phenomenon, known as the dimensionality curse, inhibits any approach based on a direct indexing of feature values. Besides ad hoc solutions, such as those above described, one might consider using some dimensionality reduction technique that projects feature vectors onto a (much) lower D'-dimensional space, $D' \ll D$, and then indexing the so-obtained D'-dimensional feature vectors. The effectiveness of such techniques however is highly variable, being dependent on the actual data distribution.

Another practical reason that could motivate the use of approximate features is the mismatch between the type of the features and the one natively supported by the index. As a simple example, consider an index implementation that only manages entries of an arbitrary, but fixed, size, and that objects to be indexed are regions of pixels described by their boundaries. Clearly, boundary descriptions have different sizes, depending on the shape of the region. In this case a possible solution would be to use a *conservative approximation* of boundaries, like minimum bounding rectangles.

Finally, it might also be the case that, although in principle actual feature values could be stored in the index, the distance function to be used on them cannot be supported by the index organization. A remarkable example is the DTW distance: since DTW is *not* a true metric, in that it does not satisfy the triangle inequality, no multidimensional index can directly process queries with such a distance function.

Metric Indexing

When features are not vectors and/or the distance function is not the Euclidean distance or some other (possibly weighted) L_p norm, coordinate-based spatial indexes cannot be used. There are many cases in which this situation shows up. For instance, in region-based image retrieval (RBIR), each image is first automatically segmented into a set of homogeneous regions, each of them being represented by a vector of low-level features (usually encoding color and texture information). Thus, each f_i is a *set of vectors* and as such cannot be indexed by a spatial index. As a further example, *graphs* representing, say, spatially located objects with their relationships cannot be directly supported by a coordinate-based index. In cases like these, one could consider using a metric index, such as the M-tree [5]. A metric index just requires the distance function d used to compare feature values to be a metric, i.e., a positive and symmetric function that also satisfies the triangle inequality: $d(f_i, f_j) \leq d(f_i, f_k) + d(f_k, f_j)$ $\forall f_i, f_j, f_k$. Although there is nowadays a large number of metric indexes available [14], as demonstrated in [3] all of them are based on the common principle of organizing the indexed features into a set of equivalence classes and then discarding some of these classes by exploiting the triangle inequality. For instance, in the case of the M-tree each class corresponds to the set of feature values stored into a same leaf of the tree. Triangle inequality can also be applied to save some distance computations while searching the index, which turns out to be particularly relevant in the case of computationally demanding distance functions (a common case with MM data). This was first shown for the M-tree, in which distances between each feature value and its parent in the index tree are precomputed and stored in the tree. The idea is quite general and effective, an obvious tradeoff existing between the amount of extra information stored in the tree and the benefit this has on pruning the search space. Along this direction, Skopal and Hoksza propose the M*-tree [12], a variant of the M-tree in which each entry in a node also includes its NN in that node, i.e., the *NN-graph* of the features in each node is maintained.

A common objection to metric indexes is that they are bound to use only a specific distance function, namely the one with which the index is built. Along the direction of increasing flexibility, Ciaccia and Patella [4] introduce the QIC-M-tree, which is an extension of the M-tree able to support queries with any distance function d_Q from the same "family" of the distance d_I used to build the tree. On the condition that there exists a *scaling factor* $S_{d_I \to d_Q}$ such that $d_I(f_i, f_j) \leq S_{d_I \to d_Q} d_Q(f_i, f_j)$ holds (i.e., d_I lower bounds d_Q up to a constant factor), the lower-bounding lemma applies, and the index can answer queries based on d_Q. A similar idea allows the QIC-M-tree to use also a "cheap" approximate distance d_C as a filter before computing the "costly" d_I and d_Q functions.

Ad Hoc Solutions

The availability of general purpose metric indexes does not rule out the possibility of deriving better, more specialized solutions for the problem at hand. For instance, the STRG-Index [10] is a specialized structure for indexing spatio-temporal graphs arising from the modelling of video sequences. Consider a video segment with N frames. Each frame is first segmented into a set of homogeneous color regions, each of which becomes a node in the region adjacency graph (RAG) of that frame, with edges connecting spatially adjacent regions. Node attributes (such as size, color, and location) are then defined, and the same is done for edges (in which case attributes such as the distance and the orientation between the centroids of connected regions can be used). Since a node representing a region can span multiple frames, nodes in consecutive RAGs can be connected to represent temporal aspects. The resulting graph is called spatio-temporal region graph (STRG). The STRG is then decomposed into a set of object graphs (OGs) and background graphs (BGs), and clusters of OGs are obtained for the purpose of indexing. Since the distance function used for comparing OGs (the so-called extended graph edit distance (EGED)) is a metric, any metric index could be used. The ad hoc STRG-Index proposed in [10] is a three-level metric tree, where the root node contains entries for the BGs, the intermediate level stores clusters of OGs, and individual OGs are inserted into the leaf level.

Extensions of available indexes might be also required as a consequence of feature approximation. An example is found in [13], where the problem of providing rotation-invariant retrieval of shapes under the Euclidean (L_2) distance is considered. After converting a shape boundary into a time series $f_i = (f_{i,1}, f_{i,2}, \ldots, f_{i,D})$ (this is quite a common way to represent shapes, see e.g., [2]), a discrete fourier transform (DFT) is applied to obtain a representation of f_i in the frequency domain. Due to Parseval's theorem, the DFT transformation preserves the Euclidean distance [1]. To obtain invariance to rotation, only the magnitude of DFT coefficient is retained. The so-resulting vectors F_i are then compressed by keeping only the k ($k \ll D$) coefficients with the highest magnitude (together with their position in the original vector) plus an error term $\in F_i$ given by the square root of the sum of the squares of dropped coefficients. This information allows a tight lower bound to be derived on the actual rotation-invariant Euclidean distance between f_i and a query shape q. For indexing, a variation of the VP-tree is introduced, which allows compressed features to be stored and searched.

Key Applications

Any application dealing with massive amounts of multimedia data requires effective indexing solutions for efficiently supporting similarity queries. This is further motivated by the complexity of distance functions that are of interest for multimedia data.

Future Directions

All the above indexing techniques and methods assume (at least) that the distance function is a metric. An interesting problem is to devise indexing methods for non-metric distance functions that do not rely on the lower-bounding lemma. The work of Skopal [11] on *semimetrics* appears to be a relevant step on this direction. In the same spirit, Goial, Lifshits and Schütse [7] study how to avoid turning the *similarity* search problem into a *distance*-based one, which in several cases might not yield a metric. Working directly with similarities is however more complex, since there is no analogue of the triangle inequality property for similarity values. Let $rank_y(x)$ be the rank of object x with respect to object y (i.e., x is the NN of y if $rank_y(x) = 1$). Then, [7] introduces the concept of *disorder constant DC*, the smallest value for which the *disorder inequality* $rank_y(x) \leq DC(rank_z(x) + rank_z(y))$ holds $\forall x, y, z$ in the given dataset, and describes algorithms for NN search based on this idea. Making this approach practical for large MM databases remains an open problem.

Cross-references

- ▶ Curse of Dimensionality
- ▶ Dimensionality Reduction
- ▶ High Dimensional Indexing
- ▶ Indexing and Similarity Search
- ▶ Indexing Metric Spaces
- ▶ Multimedia Data Querying
- ▶ Spatial Indexing Techniques

Recommended Reading

1. Agrawal R., Faloutsos C., and Swami A. Efficient similarity search in sequence databases. In Proc. 4th Int. Conf. on Foundations of Data Organizations and Algorithms, 1993, pp. 69–84.
2. Bartolini I., Ciaccia P., and Patella M. WARP: Accurate retrieval of shapes using phase of Fourier descriptors and time warping distance. IEEE Trans. Pattern Anal. Machine Intell., 27(1):142–147, 2005.
3. Chávez E., Navarro G., Baeza-Yates R., and Marroquín J.S. Proximity searching in metric spaces. ACM Comput. Surv., 33 (3):273–321, September 2001.
4. Ciaccia P. and Patella M. Searching in metric spaces with user-defined and approximate distances. ACM Trans. Database Syst., 27(4):398–437, December 2002.
5. Ciaccia P., Patella M., and Zezula P. M-tree: An efficient access method for similarity search in metric spaces. In Proc. 23th Int. Conf. on Very Large Data Bases, 2007, pp. 426–435.
6. Faloutsos C., Barber R., Flickner M., Hafner J., Niblack W., Petkovic D., and Equitz W. Efficient and effective querying by image content. J. Intell. Inf. Sys., 3(3/4):231–262, July 1994.
7. Goyal N., and Lifshits Y., and Schütse H. Disorder inequality: A combinatorial approach to nearest neighbor search. In Proc. 1st ACM Int. Conf. on Web Search and Data Mining, 2008, pp. 25–32.
8. Jagadish H.V. A retrieval technique for similar shapes. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991, pp. 208–217.
9. Keogh E. Exact indexing of dynamic time warping. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 406–417.
10. Lee J., Oh J.H., and Hwang S. STRG-index: Spatio-temporal region graph indexing for large video databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 718–729.
11. Skopal T. On fast non-metric similarity search by metric access methods. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 718–736.
12. Skopal T. and Hoksza D. Improving the performance of M-tree family by nearest-neighbor graphs. In Proc. 11th East European Conf. Advances in Databases and Information Systems, 2007, pp. 172–188.
13. Vlachos M., Vagena Z., Yu P.S., and Athitsos V. Rotation invariant indexing of shapes and line drawings. In Proc. ACM Int. Conf. on Information and Knowledge Management, 2005, pp. 131–138.
14. Zezula P., Amato G., Dohnal V., and Batko M. Similarity Search: The Metric Space Approach. Springer, Berlin Heildelberg, New York, 2005.

Multimedia Data Querying

K. Selcuk Candan[1], Maria Luisa Sapino[2]
[1]Arizona State University, Tempe, AZ, USA
[2]University of Turin, Turin, Italy

Definition

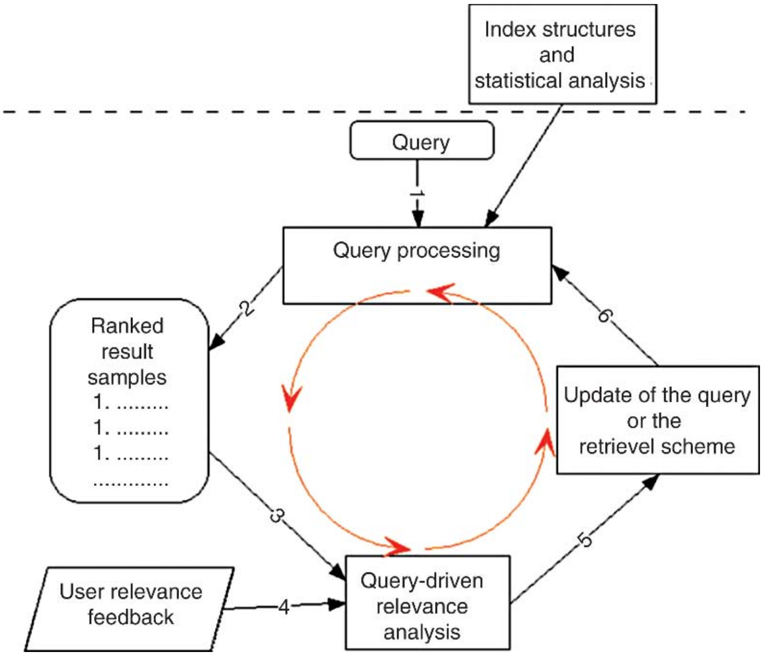
One common characteristic of multimedia systems is the *uncertainty or imprecision of the data.* The models that can capture the imprecise and statistical nature of multimedia data and query processing are fuzzy and probabilistic in nature. Therefore multimedia data query evaluation requires fuzzy and probabilistic data and query models as well as appropriate query processing mechanisms. Probabilistic models rely on the premise that the sources of imprecision in data and query processing are inherently statistical and thus they commit onto probabilistic evaluation. Fuzzy models are more flexible and allow various different semantics, each applicable under different system requirements to be selected for query evaluation.

Historical Background

Due to the possibly redundant ways to sense the environment, the alternative ways to process, filter, and fuse multimedia data, and the subjectivity involved in the interpretation of data and query results, multimedia data quality is inherently imprecise:

- *Feature extraction algorithms that form the basis of content-based multimedia data querying are generally imprecise.* For example, high error rate is encountered in motion capture data due to the multitude of environmental factors involved, including camera and object speed. Especially for video/ audio/ motion streams, data extracted through feature extraction modules are only *statistically* accurate and may be based on the frame rate or the position of the video camera related to the observed object.
- *It is rare that a multimedia querying system relies on exact object matching.* Instead, in many cases, multimedia databases leverage similarities between feature vectors to identify data objects that are similar to the query. In many cases, it is also necessary to account for s*emantic similarities* between associated annotations and *partial matches,* where objects in the result satisfy some of the requirements in the query, but fail to satisfy all query conditions.
- *Imprecision can also be due to the available index structures which are imperfect.* Due to the sheer size of the data, many systems rely on clustering and classification algorithms for pruning during query processing.
- *Query formulation methods are not able to capture user's subjective intention perfectly.* For example, in Query by Example (QBE), which features, feature value ranges, feature combinations, or which similarity notions are to be used for processing is left to the system to figure out through feature significance analysis, user preferences, relevance feedback (Fig. 1), and/or collaborative filtering techniques, which are largely statistical and probabilistic in nature.

In many multimedia querying systems, more than one of these reasons coexist and, consequently, the system must take them into consideration collectively. Figure 2 provides an example query (in an SQL-like syntax used by the SEMCOG system [10]) which brings together imprecise and exact predicates. Processing this query



Multimedia Data Querying. Figure 1. Multimedia query processing usually requires the semantic gap between what is stored in the database and how the user interprets the query and the data to be bridged through a relevance feedback cycle. This process itself is usually statistical in nature and, consequently, introduces probabilistic imprecision in the results.

```
select image P, imageobject object
where contains (P, object) and
          semantically_similar(P.semanticannotation, “travel”) and
          visually_similar(object.imageproperties, “Fujimountain.jpg”) and
          after(P.date, 2007).
```

Multimedia Data Querying. Figure 2. A sample multimedia query with imprecise and exact predicates.

requires assessment of different sources of imprecision and merging them into a single value. Traditional databases are not able to deal with imprecision since they are based on Boolean logic: predicates are treated as propositional functions, which return either *true* or *false*. A naive way to process queries is to transform imprecision into *true* or *false* by mapping values less than a cut-off to *false* and the remainder to *true*. With this naïve approach, partial results can be quickly refuted or validated based on their relationships to the cut-off. User provided cut-offs can also be leveraged for *filtering*, while maintaining the imprecision value of the results for further processing. In general, however, cut-off based early pruning leads to misses of relevant results. This leads to the need for data models and query evaluation mechanisms, which can take into account imprecision in the evaluation of the query criteria. In particular, the data and query models cannot be propositional in nature.

Foundations

Assessments of the degrees of imprecisions in multimedia data can take different forms. For example, if the data is generated through a sensor/operator with a quantifiable quality rate (for instance a function of the available sensor power), then a scalar-valued assessment of imprecision may be applicable. This is similar to the (so called type-1) fuzzy predicates, which (unlike propositional functions which return *true* or *false*) return a membership value to a fuzzy set. *In this simplest case, the quality assessment of a given object, o, is modeled as a value* $0 \leq qa(o) \leq 1$. A more general quality assessment model would take into account the uncertainties in the assessments themselves. These type of predicates, where sets have grades of membership that are themselves fuzzy, are referred to as type-2 fuzzy predicates. For example the assessment of a given object o can be modeled as a normal distribution of qualities, $qa(o) = No(qo, \xi o)$, where qo is the expected quality and ξo is the variance. Although the type-2 model can be more general and use different probability distributions, this specific model (using the normal distribution) is a generally applicable sampling-related imprecision as it relies on the well-known *central limit theorem*, which states that the average of the samples tends to be normally distributed, even when the distribution from which the average is computed is not normally distributed. Note that, in general, such complex statistical assessments of data precision can be hard to obtain. A compromise between the above two models represents the range of possible qualities of an object with a lower- and an upper-bound. In this case, *given an object o, its quality assessment,* $qa(o)$ *is modeled as a pair* $<qo_{low}, qo_{high}>$, *where* $0 \leq qo_{low} \leq qo_{high} \leq 1$.

Fuzzy data and query models for multimedia querying are based on the fuzzy set theory and fuzzy logic introduced by Zadeh in mid 1960s [14]. A fuzzy set, *F*, with domain *D* is defined using a membership function, $F: D \rightarrow [0,1]$. A fuzzy predicate, then, corresponds to a fuzzy set: instead of returning *true(1)* or *false(0)* values as in propositional functions, fuzzy predicates return the corresponding membership values (or scores). Fuzzy clauses combine fuzzy predicates and fuzzy logical operators into complex fuzzy statements. Like the predicates, the fuzzy clauses also have associated scores. The meaning of a fuzzy clause (i.e., the score it has, given the constituent predicate scores) depends on the semantics chosen for the fuzzy logical operators, *not* ($\neg$), *and* ($\wedge$), and *or* ($\vee$).

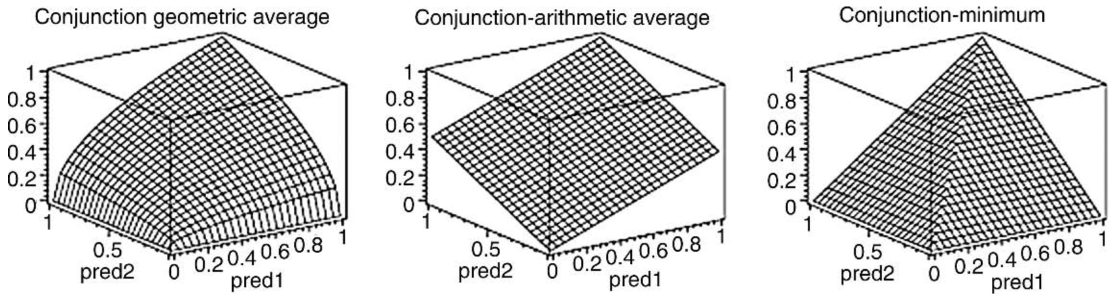
Table 1 shows popular *min* and *product* fuzzy semantics used in multimedia querying. These two semantics (along with some others) have the property that binary conjunction and disjunction operators are triangular-norms (t-norms) and triangular-conorms (t-conorms). Intuitively, t-norm functions reflect the (boundary, commutativity, monotonicity, and associativity) properties of the corresponding Boolean operations. Although the property of capturing Boolean semantics is desirable in many applications of fuzzy logic, for multimedia querying, this is not always the case [3]. For instance, the partial match requirements invalidate the boundary conditions. Monotonicity can be too weak a condition for multimedia query processing. In many cases, according to real-world and artificial nearest-neighbor workloads, the highest-scoring predicates are interesting and the rest is not interesting. This implies that the *min* semantics, which gives the highest importance on the lowest scoring predicate, may not be suitable for real workloads. Other fuzzy semantics used in multimedia systems include arithmetic and geometric average semantics. Figure 3 visualizes the behavior of the fuzzy conjunction operator under different fuzzy semantics. It is well established that the only fuzzy semantics which preserves logical equivalence of statements (involving conjunction and disjunction) and is also monotonic is the *min* semantics. This, and the query processing efficiency it enables due to its simplicity, make it a popular choice despite its shortcomings.

Processing multimedia queries, like the one depicted in Fig. 2, under a fuzzy system requires extending query languages and query processors with fuzzy semantics. Many commercial database management systems include fuzzy extensions that are suitable for multimedia applications. Relational databases can be extended to capture fuzzy data in various different ways. In *tuple-level approaches,* the schema of each fuzzy relation is extended to include one or more attributes, each representing the degrees of imprecision of the tuples in the relation with respect to a different interpretation of the tuples. In these systems, the relational algebra operators (such as select, project, join, union, difference) are also extended to apply the selected fuzzy semantics to the tuple scores. In the *attribute-level approaches*, the degrees of uncertainty are associated individually to the attribute values. Especially when the imprecisions in the various attributes of a multimedia object are due to different reasons, attribute level approaches are more applicable due to their finer granularity. Furthermore, since each attribute can be treated as a fuzzy predicate on the multimedia object, query evaluation within these models can benefit more naturally from fuzzy logic evaluation schemes.

Processing these queries, on the other hand, requires significant extensions to the underlying database engines. For example, the underlying relational concepts, such as functional dependencies and normalization, need to be extended to cope with fuzziness in the stored data. In particular, in multimedia

Multimedia Data Querying. Table 1. Fuzzy *min* and *product* semantics for logic operators: $\mu_i\ (x)$ stands for the score of the predicate P_i on x

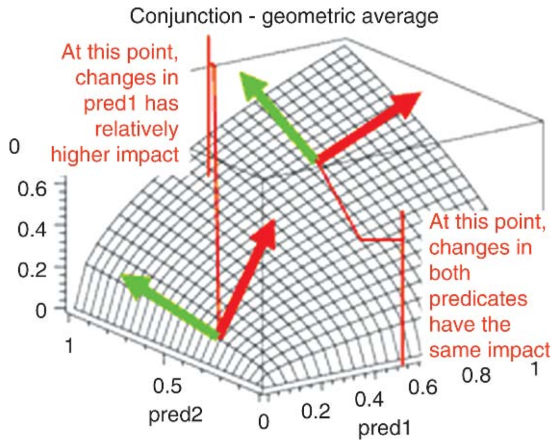
Min semantics	Product semantics
$\mu_{P_i \wedge P_j}(x) = min\{\mu_\imath(x), \mu_j(x)\}$	$\mu_{P_i \wedge P_j}(x) = \frac{\mu_\imath(x) \times \mu_j(x)}{max\{\mu_\imath(x), \mu_j(x), \alpha\}} \qquad \alpha \in [0, 1]$
$\mu_{P_i \wedge P_j}(x) = max\{\mu_\imath(x), \mu_j(x)\}$	$\mu_{P_i \vee P_j}(x) = \frac{\mu_\imath(x)+\mu_j(x)-\mu_\imath(x)\times\mu_j(x)-min\{\mu_\imath(x),\mu_j(x),1-\alpha\}}{max\{1-\mu_i(x),1-\mu_j(x),\alpha\}}$
$\mu_{\neg P_i}(x) = 1 - \mu_i(x)$	$\mu_{\neg P_i}(x) = 1 - \mu_i(x)$



Multimedia Data Querying. Figure 3. Visual representations of various binary fuzzy conjunction semantics: The horizontal axes correspond to the values between 0 and 1 for the two input conjuncts and the vertical axis represents the resulting scores according to the corresponding function.

databases, users are usually interested in a result set which is ranked according to a ranking criterion which is generally user dependent (Fig. 1). Adali et al. [1] introduces a similarity algebra which brings together relational operators and results of multiple similarity implementations in a uniform language. Other algebraic treatments of fuzzy multimedia queries, relying on finer granularity attribute-based models, include the *FNF*^{2} *algebra* [5]. When the requirement for exact matches is removed, the result space becomes significantly large, and thus, the query engine cannot rely on any processing scheme which would need to touch or enumerate all solutions. Consequently, query processing schemes would need to generate results as progressively (in decreasing order of relevance) as possible. Fagin [7] proposes ranked query evaluation algorithms, which assume that individual sources can progressively output sorted results and also enable random access. These algorithms also assume that the query has a monotone combined scoring function. Candan et al. [4] presents *approximate* ranked query processing techniques for cases where not all sub-queries are able to return ordered results. In turn, Fagin et al. [8] recognizes that there may be cases where random accesses are impossible and presents algorithms under monotonicity assumption to enumerate top-*k* objects without accessing all the data. These augment *monotonicity* with an *upper bound* principle, which enables bounding of the maximum possible score of a partial result. Qi et al. [13] establishes an alternative, *sum-max monotonicity* property and shows how to leverage this for developing a self-punctuating, horizon-based ranked join (HR-Join) operator for cases when the more strict monotonicity property does not hold. Top-*k* querying can also be viewed as a *k*-constrained optimization problem, where the goal function includes both a Boolean constraint characterizing the data of interest and a quantifying function which acts as the numeric optimization target [15]. Adali et al. [2] and Li et al. [11] extend the relational algebra to support ranking as a first-class construct. Li et al. [11] also presents a pipelined and incremental execution model of ranking query plans.

A particular challenge in multimedia querying is that (as shown in Fig. 1) the underlying query processing scheme needs to adapt to the specific needs and preferences of individual users. Due to its flexibility, the fuzzy model enables various mechanisms of adaptation. First of all, if user's feedback focuses on a particular attribute in the query, the way the fuzzy score of the corresponding predicate is computed can change based on the feedback. Secondly, the semantics of the fuzzy logic operator can be adapted based on the feedback of the user. A third mechanism through which user's feedback can be taken into account is to enrich the merge function, used for merging the fuzzy scores, with weights that regulate the impacts of the individual predicates. Fagin proposed a generic weighting mechanism that can be used for any fuzzy merge function [7]. The mechanism ensures that (a) the result is a continuous function of the weights (as long as the original merge function is continuous), (b) sub-queries with zero weight can be dropped without affecting the rest of the query, and (c) if all weights are equal, then the result is equal to the original, not-weighted merge function. Candan and Li [3], on the other hand, argued that the relative importance of predicates in a merge function should be measured in terms of the overall impacts changes in the scores that the individual predicates would have on the overall score (Fig. 4). Consequently, the relative importance of predicates can vary based on the scores the individual predicates take and the corresponding partial derivatives. A more direct mechanism to capture the user feedback is to modify the partial derivatives of the scoring functions appropriately. While the generic scheme presented by Fagin [7] would satisfy this for some merge functions, such as the arithmetic average, it would fail to capture this requirement for others, such as the commonly used product semantics.



Multimedia Data Querying. Figure 4. The relative impact of the predicates in a scoring function can vary based on the scores of the individual predicates.

Unlike the fuzzy models, which can capture a large spectrum of application requirements, probabilistic approaches to data and query modeling are applicable only to those cases where the source of imprecision is of statistical nature. These cases include probabilistic noise in data collection, sampling (over time, space, or population members) during data capture or processing, randomized and probabilistic algorithms (such Markov chains and Bayesian networks) used in media processing and pattern detection, and probabilistic consideration of relevance feedback. Dalvi and Suciu [6], for example, associates a value between 0 and 1 to each tuple in a given relation: the value expresses to probability with which a given tuple belongs to the relation. By extending SQL and the underlying relational algebra with probabilistic semantics and a theory of belief, the authors provide a probabilistic semantics for query processing with uncertain matches.

A general simplifying assumption in many probabilistic models is that the individual attributes (and the corresponding predicates) are independent of each other: consequently, the probability of a conjunction can be computed as the product of the probabilities of the conjuncts; i.e., under these conditions, the probabilistic model corresponds to the fuzzy product semantics. However, the independence assumption does not always hold (in fact, it rarely holds). Lakshmanan et al. [9] presents a probabilistic relational data model, an algebra, and aggregate operators that capture various types of interdependencies, including independence, mutual exclusion, as well as positive, negative, and conditional correlation.

While the simplest probabilistic models associate a single value between 0 and 1 to each attribute or tuple, more complete models represent the score in the form of an interval of possible values or more generally in terms of a probability distribution describing the possible values for the attribute or the tuple. Consequently, these models are able to capture more realistic scenarios, where the imprecision in data collection and processing prevents the system to compute the exact *quality* of the individual media objects, but (based on the domain knowledge) can associate probability distributions to them. Note that relaxing the independence assumption or extending the model to capture non-singular probability distributions both necessitate changes in the underlying rank evaluation algorithms.

Other non-relational probabilistic models for multimedia querying includes Markov chains and Bayesian networks. A stochastic process is said to be Markovian if the conditional probability distribution of the future states depends only on the present. A Markov chain is a discrete-time stochastic process which is conditionally independent of the past states. A random walk on a graph, $G(V,E)$, is a Markov chain whose state at any time is described by a vertex of G and the transition probability is distributed equally among all outgoing edges. The transition probability distribution in the corresponding Markov model can be represented as a matrix, where the (i, j)'th element of this matrix, T_{ij}, describes the probability that, given that the current state is i, the process will be in state j in the next time unit; i.e., the n-step transition probabilities can be computed as the n'th power of the transition matrix. Markovian models are used heavily for linkage analysis in supporting queries over web, multimedia, and social network data with graphical representations.

A Bayesian network is another graphical probabilistic model used especially for representing probabilistic relationships between variables (e.g., objects, properties of the objects, or beliefs about the properties of the objects) [12]. In a Bayesian network nodes represent variables and edges between the nodes represent the relationships between the probability distributions of the corresponding variables. Consequently, once they are fully specified, Bayesian networks can be used for answering probabilistic queries given certain observations. However, in many cases, both the structure as well as the parameters of the network have to be learned through iterative and sampling-based heuristics, such as expectation maximization (EM), and Markov Chain Monte Carlo (MCMC) algorithms. Hidden Markov models (HMMs), where some of the states are hidden (i.e., unknown), but variables that depend on these states are observable, are Bayesian networks used commonly in many machine-learning based multimedia pattern recognition applications. This involves training (i.e., given a sequence of observations, learning the parameters of the underlying HMM) and pattern recognition (i.e., given the parameters of an HMM, finding the most likely sequence of states that would produce a given output).

Key Applications

Applications of multimedia querying include personal and public photo/media collections, personal information management systems, digital libraries, on-line and print advertisement, digital entertainment, communications, long-distance collaborative systems, surveillance, security and alert detection, military, environmental monitoring, ambient and ubiquitous systems that provide real-time personalized services to humans, improved accessibility to blind and elderly, rehabilitation of patients through visual and haptic feedback, and interactive performing arts.

Future Directions

While most of the existing work in this area focused on content-based and object-based query processing, future directions in multimedia querying will involve understanding of how media objects affect users and how do they fit into users experiences in the real world. These require better understanding of underlying psychological and cognitive processes in human media processing. Ambient media-rich systems which collect and feed in diverse media from environmentally embedded sensors necessitate novel ways of continuous and distributed media processing and fusion schemes. Intelligent schemes to choose the right objects to process are needed to scale query processing workflows to the immense influx of real-time media data. In a similar manner, collaborative-filtering based query processing schemes that can help overcoming the semantic gap between media and users' experiences will help the multimedia databases scale to Internet-scale media indexing and querying.

Cross-references

- ▶ Fuzzy Models
- ▶ Probabilistic Databases
- ▶ Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model
- ▶ Information Retrieval
- ▶ Multimedia Data
- ▶ Multimedia Databases
- ▶ Multimedia Data Indexing
- ▶ Multimedia Information Retrieval Model
- ▶ Multimedia Retrieval Evaluation
- ▶ Top-K Selection Queries on Multimedia Datasets

Recommended Reading

1. Adali S., Bonatti P.A., Sapino M.L., and Subrahmanian V.S. A multi-similarity algebra. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 402–413.
2. Adali S., Bufi C., and Sapino M.L. Ranked relations: query languages and query processing methods for multimedia. Multimed. Tools Appl., 24(3):197–214, 2004.
3. Candan K.S. and Li W.-S. On similarity measures for multimedia database applications. Knowl. Inf. Syst., 3(1):30–51, 2001.
4. Candan K.S., Li W.-S., and Priya M.L. Similarity-based ranking and query processing in multimedia databases. Data Knowl. Eng., 35(3):259–298, 2000.
5. Chianese A., Picariello A., Sansone L., and Sapino M.L. Managing uncertainties in image databases: a fuzzy approach. Multimed. Tools Appl., (23):237–252, 2004.
6. Dalvi N.N. and Suciu D. Efficient query evaluation on probabilistic databases. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 864–875.
7. Fagin R. Fuzzy queries in multimedia database systems. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 1–10.
8. Fagin R., Lotem A., and Naor M. Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci., 66 (4):614–656, 2003.
9. Lakshmanan L.V., Leone N., Ross R., and Subrahmanian V.S. ProbView: a flexible probabilistic database system. ACM Trans. Database Syst., 22(3):419–469, 1997.
10. Li W.-S. and Candan K.S. SEMCOG: a hybrid object-based image and video database system and its modelling, language, and query processing. Theory & Practice of Object Syst., 5 (3):163–180, 1999.
11. Li C., Chang K.C.-C., Ilyas I.F., and Song S. RankSQL: Query algebra and optimization for relational top-k queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 131–142.
12. Pearl J. Bayesian networks: a model of self-activated memory for evidential reasoning. In Proc. 7th Conf. of the Cognitive Science Society, 1985, pp. 329–334.
13. Qi Y., Candan K.S., and Sapino M.L. Sum-Max monotonic ranked joins for evaluating top-K twig queries on weighted data graphs. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 507–518.
14. Zadeh L.A. Fuzzy sets. Inf. Control, 8(3):338–353, 1965.
15. Zhang Z., Hwang S., Chang K.C., Wang M., Lang C.A., and Chang Y. Boolean + Ranking: querying a database by K-constrained optimization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 359–370.

Multimedia Data Storage

Jeffrey Xu Yu
Chinese University of Hong Kong, Hong Kong, China

Definition

Data storage management, as one of the important functions in database management systems, is to manage data on disk in an efficient way to support data retrieval and data update. Multimedia data storage management is to manage continuous media data (audio/video) on disk. The uniqueness of multimedia data storage management is, in a multiuser environment, how to arrange the data storage to support a continuous retrieval of large continuous media data from disk to be displayed on screen, at a pre-specified rate, without any disruptions, which is also called hiccup-free display.

Historical Background

In a multimedia environment, the continuous media needs to be retrieved and displayed continuously. As magnetic disks are used as the mass storage device for multimedia data, zoning is one approach to increase the storage capacity of magnetic disks. Here, zones of a disk drive are different regions of the disk drive that usually have different transfer rates. A number of studies have investigated techniques to support a hiccup-free display of continuous media (video/audio) using magnetic disk drives with a single zone [1,2,9,10] in the early 1990s. These studies assume a fixed transfer rate for a disk drive. These techniques can be possibly adopted to design a multi-zone disk system, but such a multi-zone disk system is then forced to use the minimum transfer rate of the zones for the entire disk, in order to guarantee a continuous display of continuous media objects. Such an approach is called Min-Z-tfr.

In the late 1990s and early 2000s, many new techniques are proposed to deal with the storage of continuous media in multi-zone disks [4,5]. In [4], VARB and FIXB are proposed to place media objects on the multi-zone disks. VARB and FIXB techniques provide the average transfer rate of zones while ensuring a continuous display, compared with Min-Z-tfr, which is forced to use the minimum transfer rate of zones. VARB and FIXB increase the throughput of the system, while they also (i) increase startup latency, (ii) waste disk space, and (3) increase the amount of memory required to support more simultaneous displays. A configuration planner is proposed to decrease the drawbacks of VARB and FIXB, in order to meet the performance requirements of applications [4]. As VARB and FIXB [4] take account of a single media type only, RP, MTP, and MVP are proposed in [5] to support multiple media types with different bandwidth requirements. In [4,5], the discussions focus on multimedia data placement across disk drives to support continuous display requirement.

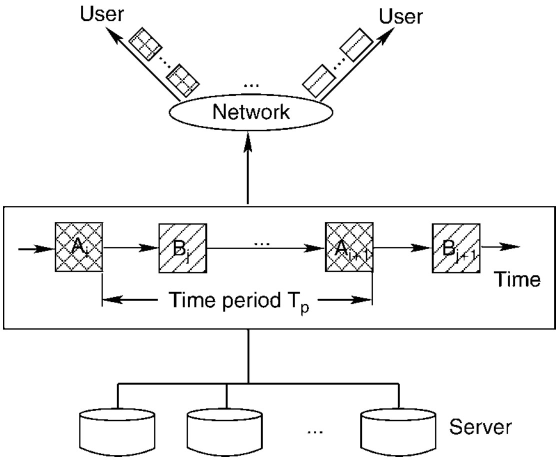
Foundations

Hiccup-Free Display

The size of continuous media, especially videos, can be very large. The transmission of data must be just-in-time. In other words, data must be retrieved from disks and transmitted to the display in a timely manner that prevents hiccups. A cycle-based data retrieval technique [6,12] is designed to provide continuous display for multiple users. As shown in Fig. 1, it is a cycle-based data retrieval technique to support hiccup-free display [5]. Consider a constant-bit-rate (CBR) media. In order to guarantee a continuous display of a continuous media *A*, the system needs to retrieve the block of A_i before its immediate previous block A_{i-1} completes its display. For each block A_i, there are two tasks, namely, block retrieval and display initialization. This process of block retrieval and display initialization repeats in a cyclic manner until all *A* blocks have been displayed. If the time to retrieve a block, termed *block retrieval time*, T_p (as indicated in Fig. 1), is smaller than or equal to the time period to display a block, then the whole display process will be hiccup free [5]. The time interval between the time a request of *A* arrives and the time the display of *A* starts is called *startup latency*.

FIXB and VARB

Modern disk drivers are produced with multiple zones to meet the demands for a higher storage capacity [11]. A zone is a contiguous collection of disk cylinders where the tracks in the cylinders are supposed to have the same storage capacity. The outer zones have a higher transfer rate in comparison with the inner zones. Two approaches, FIXB and VARB, are proposed in [4] to support a continuous display of continuous media using a single disk with multi-zones. Suppose that the disk consists of *m* zones, Z_1, Z_2,...,Z_m, and



Multimedia Data Storage. Figure 1. A hiccup free display (Fig. 3 in [5]).

the transfer rate of zone Z_i is R_i. Assume that each object X is partitioned into f blocks: X_1, X_2,...,X_f.

With FIXB (Fixed Block Size), the blocks of an object X are rendered equi-sized, i.e., $X_i = X_j$, for any i and j. The system assigns the blocks of X to the zones in a round-robin manner. FIXB is designed to support a predetermined number of simultaneous displays (N). The retrieval process of this system is to scan the disk in one direction, for example, starting with the outermost zone moving inward, visiting one zone at a time and multiplexing the bandwidth of that zone among N block reads. A sweep is a scan of the zones. The time to perform one such a sweep is denoted as T_{Scan}. The time of reading N blocks from zone Z_i, denoted $T_{MUX}(Z_i)$, is dependent on the transfer rate of zone Z_i. As the transfer rate of zones varies, the time to read blocks from different zones also varies. To support hiccup-free displays, the system uses buffers to compensate for the low transfer rates of innermost zones.

VARB makes $T_{MUX}(Z_i)$ to be identical for all zones, using variable block sizes. The size of a block, $B(Z_i)$, is a function of the transfer rate of the zone Z_i. This results in an identical transfer time for all the blocks, T_{disk}, i.e., $T_{disk} = \frac{B(Z_i)}{R_i} = \frac{B(Z_j)}{R_j}$, for any i and j. Like FIXB, VARB assigns the blocks of an object to the zones in a round-robin manner. Unlike FIXB, with VARB, the blocks of an object X have different sizes depending on which zones the blocks are assigned to. Also, like FIXB, VARB employs memory to compensate for the low bandwidth of innermost zones.

With FIXB, the blocks of an object is equi-sized, whereas VARB renders the blocks in different sizes, which depends on the transfer rate of its assigned zone. FIXB is easy to be implemented in compared with VARB. But VARB requires a lower amount of memory and incurs a lower latency as compared to FIXB.

RP, MTP, and MVP

Based on zoning, there are different data transfer rates (R_i) to retrieve data from a disk. When a server is required to support multiple media types with different bandwidth requirements, the block reading time varies widely depending on the block size and its assigned zone. Suppose that there are n different media types to be supported. The block size of a media type i object is determined by $B_i = T_p \times D_i$ where D_i is the bandwidth requirement of the media type i, and T_p is a fixed time period which is set to be the same for all the media types. The transfer time (service time) to retrieve a block of a media type i object in zone Z_j is $s_{i,j} = \frac{B_i}{R_j}$. Suppose that there are b blocks, the average service time is computed as

$$\bar{s} = \sum_{i=1}^{b} F_i \sum_{j=1}^{n} P_{i,B_j} \sum_{k=1}^{m} \frac{P_{i,Z_k} B_j}{R_k}$$

where F_i is the access frequency of block i, for $1 \leq i \leq b$, P_{i,B_j} is the probability that the size of block i is B_j, and P_{i,Z_k} is the probability that this block is assigned to zone Z_k. The variance of service time is:

$$\sigma_s^2 = \sum_{i=1}^{b} F_i \sum_{j=1}^{n} \sum_{k=1}^{m} P_{i,B_j} P_{i,Z_k} (s_{j,k} - \bar{s})^2$$

Three approaches are proposed in [5]: RP, MTP, and MVP. RP (Random Placement) assigns blocks to the zones in a random manner. MTP (Maximizing Throughput Placement) sorts blocks based on their size and frequency of access ($F_i \times B_i$). The blocks are assigned to the zones sequentially starting with the fastest zone, i.e., block i with the highest $F_i \times B_i$ value is assigned to the fastest zone. With MVP (Minimizing Variance Placement), a block of size B_i is placed on the zone Z_j (with R_j) which has the closest $\frac{B_i}{R_j}$ value to the average block reading time ($\bar{T}_B$):

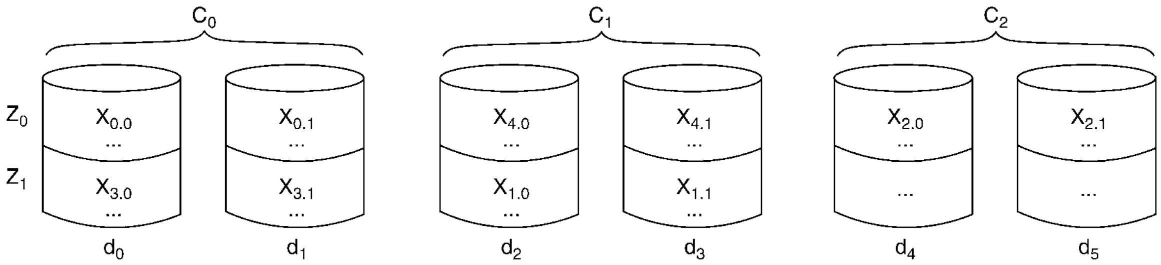
$$\bar{T}_B = \frac{average\ block\ size}{average\ transfer\ rate} = \frac{\frac{1}{n}\sum_{i=1}^{n} B_i}{\frac{1}{m}\sum_{i=1}^{m} R_i}$$

Performance studies in [5] demonstrate that both MTP and MVP are superior to RP. MVP outperforms MTP regarding the average service time and/or variance of service time. One advantage of MVP is that it is not sensitive to the access frequency of objects.

Data Placement across Disk Drivers

The bandwidth of a single disk is insufficient for the multimedia applications that strive to support thousands of simultaneous displays. One approach is to employ a multi-disk architecture. Assuming a system with D homogeneous disks, the data is striped across the disks in order to distribute the load of a display evenly across the disks [2,3,8].

The striping technique is as follows (Fig. 2). First, the disks are partitioned into k disk clusters where each cluster consists of d disks: $k = \lceil \frac{D}{d} \rceil$. An object X is partitioned into f blocks, X_1, X_2,..., X_f, and the blocks



Multimedia Data Storage. Figure 2. Three clusters with two logical zones per cluster (Fig. 12 in [4]).

of X are assigned to the k disk clusters in a round-robin manner, starting with an arbitrarily chosen disk cluster and zone, for example, zone Z_j in disk cluster C_i. In a disk cluster, each block of X, X_i, is declustered [7] into d fragments, $X_{i,j}$, where each fragment is assigned to a different disk in the disk cluster. As shown in Fig. 2, the X_0 block is assigned to the disk cluster C_0, and its two declustered fragments, $X_{0,0}$ and $X_{0,1}$ are assigned to the zone Z_0. Note that the fragments of a block need to be assigned to the same zone on the d disks in the disk cluster where the block is assigned to. In the retrieval of objects, one zone of all disks in a disk cluster is active per time period. To display object X of Fig. 2, it needs to access zone Z_0 in disk cluster C_0, when the disk cluster is idle, followed by accessing zone Z_1 in disk cluster C_1. This process repeats to retrieve/display all blocks of the object X.

Key Applications

Multimedia information systems have emerged as an essential component in many application domains ranging from library information systems to entertainment technology. The data storage management is the basis to support a continuous display of multimedia objects.

Cross-references

▶ Continuous Multimedia Data Retrieval
▶ Multimedia Data Buffering
▶ Multimedia Resource Scheduling
▶ Storage Access Model
▶ Storage Devices
▶ Storage Management
▶ Storage Manager
▶ Storage Resource Management

Recommended Reading

1. Anderson D.P. and Homsy G. A continuous media I/O server and its synchronization Mechanism. Computer, 24(10):51–57, 1991.
2. Berson S., Ghandeharizadeh S., Muntz R., and Ju X. Staggered striping in multimedia information systems. ACM SIGMOD Rec., 23(2):79–90, 1994.
3. Ghandeharizadeh S. and Kim S. Striping in Multi-disk Video Servers. In Proc. SPIE High-Density Data Recording and Retrieval Tech. Conf., 1995.
4. Ghandeharizadeh S., Kim S., Shahabi C., and Zimmermann R. Multimedia Information Storage and Management, chap. 2: Placement of Continuous Media in Multi-Zone Disks. Kluwer Academic, 1996.
5. Ghandeharizadeh S. and Kim S.H. Design of multi-user editing servers for continuous media. Multimedia Tools Appl., 11(1):101–127, 2000.
6. Ghandeharizadeh S., Kim S.H., Shi W., and Zimmermann R. On minimizing startup latency in scalable continuous media servers. In Proc. SPIE Conf. on Multimedia Computing and Networking, 1997.
7. Ghandeharizadeh S., Ramos L., Asad Z., and Qureshi W. Object placement in parallel hypermedia systems. In Proc. 17th Int. Conf. on Very Large Data Bases, 1991, pp. 243–254.
8. Ozden B., Rastogi R., and Silberschatz A. Disk striping in video server environments. In Proc. Int. Conf. on Multimedia Computing and Systems, 1996, pp. 580–589.
9. Rangan P.V. and Vin H.M. Efficient storage Techniques for Digital Continuous Multimedia. IEEE Trans. Knowl. Data Eng., 5(4):564–573, 1993.
10. Reddy A.L.N. and Wyllie J.C. I/O issues in a multimedia system. Computer, 27(3):69–74, 1994.
11. Ruemmler C. and Wilkes J. An introduction to disk drive modeling. Computer, 27(3):17–28, 1994.
12. Tewari R., Mukherjee R., Dias D.M., and Vin H.M. Design and performance tradeoffs in clustered video servers. In Proc. Int. Conf. on Multimedia Computing and Systems, 1996, pp. 144–150.

Multimedia Databases

Ramesh Jain
University of California-Irvine, Irvine, CA, USA

Synonyms

Multimodal databases

Definition

Multimedia Databases are databases that contain and allow key data management operations with multimedia data. Traditional databases contained alphanumeric data and managed it for various applications. Increasingly, applications now contain multimedia data that requires defining additional types and requires development of operations for storage, management, access, and presentation of multimedia data. Multimedia databases must increasingly deal with issues related to managing multimedia data as well as the traditional data. Commonly, databases that manage images, audio, and video in addition to metadata related to these and other alphanumeric information are called multimedia databases. When databases contain only one of the images, audio, or video, they are called image databases, audio databases, and video databases, respectively. Considering the current trend, it is likely that most databases will slowly become multimedia databases.

Historical Background

The first in multimedia databases to appear were image databases that started appearing in late 1980s. Researchers in early image databases were more concerned with using databases for maintaining results of image processing operations to analyze and understand image analysis systems. Remote sensing and medical imaging produced images that needed to be saved and analyzed to extract information for various applications. In most of these applications, an environment to save images and processing results of these images were required.

The idea of making images an integral component of databases first started appearing in early 1990s. Relational data model had become the most common data model to deal with structured data and was used to store images as binary large objects (BLOBs) in these databases. To deal with images as first class data objects in images, a multilayered data model was proposed. This model considered image objects, and domain objects and suggested storage of those along with changes in relationships among objects. Some interesting developments in early systems evolved along two independent directions. In one direction [3], a user was considered an integral part of the query environment and feedback from user resulted in continuous refinement of queries leading to finding images that were required. In the other approach, many low level features were computed and used for finding images using query by example approach. These two approaches adopted distinctly different directions, the first used domain knowledge and the second relied only on image features without any use of domain knowledge. The image features used commonly are different types and characteristics of color histograms and texture measures. These approaches are commonly called content-based retrieval, to differentiate them from metadata based retrieval. Commercially image database systems appeared in traditional database systems in mid 1990s. IBM used its QBIC technology in their DB2 database system and Oracle, Sybase, and Illustra used technology developed by a start-up company Virage. All search engines use image retrieval mostly based on the metadata that includes name of the file and text in the context of the image on a webpage.

Content based video retrieval result started in analyzing video into its constituent parts. At the lowest level is a frame, an individual image. Images are grouped into shots, shots into scenes, and scenes into episodes. All this data is extracted from video and stored in the database. Speech recognition techniques are used to prepare the transcript of the video and are also used in the database. Such systems found early use in TV program production and defense applications. Virage technology was used in these applications. Current search engines usually use metadata for searching video. Some specialized video search companies such as Blinkx (http://www.blinkx.com/) use predominantly text obtained using speech recognition or closed captions in television video. News videos have been one of the major application domains due to their applications as well as to good quality of audio available for these.

In audio databases, the signal is analyzed to detect characteristics that could be used in searching musical pieces that are similar to those. Such techniques, sometimes referred to as query by humming, were thought to be useful in finding music of interest.

Some effort has gone into analyzing CAD databases also for retrieving drawing and objects of interest.

Though some research has started in addressing multimedia data, rather than just one of the above multimedia types, most research is in either images, video, or audio. Text or metadata available in context of multimedia data is being considered in multimedia database research increasingly.

Foundations

The most fundamental difference in multimedia databases compared to the traditional databases is in the rich semantics of the data. Multimedia data, in addition to being lot more voluminous, is very rich in semantics. Many queries that users are interested require understanding of the semantics of the data. The problem becomes more complex because the semantics of multimedia is dependent not only on the data, but also on the specific user, the context in which the query is asked, and other sets of data available in the system.

Early approaches to multimedia databases did not consider the nature of multimedia data and just stored multimedia data either as BLOBs or links to files containing the data. These systems could allow only limited operations on multimedia data – usually limited to display or rendering of the data. No other operations or queries could be performed on this data.

With increasing use of multimedia data and applications that require use of multimedia data in many different ways, the nature of multimedia databases started changing. Currently, multimedia databases are still in their early stages. Many different concepts and approaches are being tried. Some of the important emerging ideas that are being tried in multimedia databases are discussed below. This area is currently a very active one and is likely to receive increasing attention both from academic and industrial research community.

A multimedia database system is considered to have the following four clear modules that need to work together to provide the functionality desired from them.

Data Analysis and Feature Extraction

A MMDB contains multimedia data but just storing the data as a BLOB does not allow any queries related to the content of the data. To solve this problem, data is analyzed to extract features from the data. These features can then be used to derive the required semantics. The features extracted depend on the nature of the type of the data and domain of application for the MMDB. These features could range from low-level features that are very general and do not depend on the application domain such as, color histograms and texture features for images to high level features directly tied to application domain such as shape of the tumor.

A significant amount of research related to MMDB is in specific media related research communities such as audio processing or computer vision. There is strong interest in finding efficient and effective features to interpret multimedia data in general as well as in specific applications.

Domain Knowledge and Interpretation

Multimedia data interpretation requires use of domain knowledge. Moreover, the knowledge required for interpretation of this data is not only the traditional domain knowledge represented using ontologies and similar techniques well developed for interpretation of text; but also media dependent models that require sophisticated classification approaches. In audio and video events must be detected in the data and that requires processing time dependent features.

There is a new emerging perspective that multimedia data should be considered evidence for real world events captured using such data. This requires modeling events and representing knowledge about domain events. This knowledge is then used in interpretation of multimedia data not in silos but together. Some progress is being made in representation of events.

Interaction and User Interface

Interaction environments used in traditional databases and search engines are not satisfactory in many applications of MMDB. Using keywords or names of objects, some limited searches can be performed, but many applications require concepts and ideas that require both continuous interactions and successive refinement of queries in what is called emergent semantics environment. Query by example including query using sketches, humming, and some other non-textual approaches are being developed for some applications.

Presentation of results of queries also requires different techniques. Multimedia data is not very suited to list or record based presentations. Also, in many applications different media sources must be combined to create multimedia presentations with which a user can interact to refine and re-articulate their queries in the emergent semantics environment.

Storage, Matching, and Indexing

In most applications, the size of the multimedia data requires special attention. Commonly the video files can not be stored even using BLOBs. It is common to store file names of multimedia data and compute and store features from the data in the database. In most applications, for each file the number of features that should be stored becomes very large from hundreds to hundreds of thousands for each multimedia data item. These features are used in searching for correct results.

Unlike traditional databases, where records are searched based on exact matches, in MMDB search requires similarity matching. It is very rare to find a result using exact matching. Search in MMDB usually becomes finding data that has maximum similarity based on features. The similarity techniques [] requires comparing the features in queried data with all the data in a MMDB for evaluating similarity.

Indexing in MMDB for similarity computation requires representing features in a way that can allow fast computation for potentially similar objects. Many high dimensional techniques have been developed for organizing this data. The dimensionality of data, sometimes called curse of dimensionality, poses interesting challenges in such organization.

Key Applications

Multimedia data is becoming ubiquitous. Ranging from photos to videos, multimedia data is becoming part of all applications. Most emerging applications now have some kind of multimedia data that must be considered integral part of the databases. Moreover, emerging applications in all applications areas, ranging from homeland security to healthcare contain rich multimedia data. Based on current trend, it is safe to assume that in very near future, much of the data managed in databases will be multimedia. Some particular application domains where multimedia is natural and will continue dominating are entertainment, news, healthcare, and homeland security.

Future Directions

From structured data to semi-structured data and then to unstructured data, databases are being challenged to deal with increasingly semantic-rich environment. MMDB offer the biggest challenge to databases in terms of bridging the semantic gap.

Increasingly, applications are talking about situation modeling using real life sensor data. These applications combine live sensory data with other information to project current situation and also predict near future for users to take appropriate actions. These databases will require sophisticated tools to manage streaming multimedia data. Research efforts in these areas have already started and are likely to accelerate significantly in the near future.

Cross-references

► Multimedia Data

Recommended Reading

1. Bach J., Paul S., and Jain R. An interactive image management system for face information retrieval. IEEE Trans. Knowl. Data. Eng., Special Section on Multimedia Information Systems., 5(4):619–628, 1993.
2. Gupta A., Weymouth T., and Jain R. Semantic Queries with Pictures, The VIMSYS Model. In Proc. 17th Int. Conf. on Very Large Data Bases, 1991, pp. 3–6.
3. Jain R. Out of the Box Data Engineering Events in Heterogeneous Data (Keynote talk). In Proc. 19th Int. Conf. on Data Engineering, 2003.
4. Jain R. Events and experiences in human centered computing. IEEE Comput, 41(2):42–50, 2008.
5. Katayama N. and Shin'ichi Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 369–380.
6. Lew M., Sebe N., Djerba C., and Jain R. Content-based multimedia information retrieval: state of the art and challenges. ACM Trans. Multimedia Comp., Comm., and Appl., 2(1):1–19, 2006.
7. Santini S., Gupta A., and Jain R. Emergent semantics through interaction in Image Databases. IEEE Trans. Knowl. Data. Eng.,13(3):337–351, 2001.
8. Santini S. and Jain R. Similarity Measures. IEEE Trans. Pattern. Anal. and Mach. Intell, 21:9, 1999.
9. Smeulders A., Worring M., Santini S., Gupta A., and Jain R. Image Databases at the end of the early years. IEEE Trans Pattern Anal Mach Intell, 23(1), 2001.

Multimedia Information Discovery

► Multimedia Information Retrieval Model

Multimedia Information Retrieval

► Semantic Modeling and Knowledge Representation for Multimedia Data

Multimedia Information Retrieval Model

Carlo Meghini, Fabrizio Sebastiani, Umberto Straccia
The Italian National Research Council, Pisa, Italy

Synonyms

Content-based retrieval; Semantic-based retrieval; Multimedia information discovery

Definition

Given a collection of multimedia documents, the goal of multimedia information retrieval (MIR) is to find the documents that are relevant to a user information need. A multimedia document is a complex information object, with components of different kinds, such as text, images, video and sound, all in digital form.

Historical Background

The vast body of knowledge nowadays labeled as MIR, is the product of several streams of research, which have arisen independently of each others and proceeded largely in an autonomous way, until the beginning of 2000, when the difficulty of the problem and the lack of effective results made it evident that success could be achieved only through integration of methods. These streams can be grouped into three main areas:

The first area is that of *information retrieval* (IR) proper. The notion of IR attracted significant scientific interest from the late 1950s in the context of textual document retrieval. Early characterizations of IR simply relied on an "objective" notion of topic-relatedness (of a document to a query). Later, the essentially subjective concept of relevance gained ground, and eventually became the cornerstone of IR. Nowadays, IR is synonymous with "determination of relevance" [9].

Around the beginning of the 1980s, the area of multimedia documents came into existence and demanded an IR functionality that no classical method was able to answer, due to the *medium mismatch problem* (in the image database field, this is often called the *medium clash problem*). This problem refers to the fact that when documents and queries are expressed in different media, matching is difficult, as there is an inherent intermediate mapping process that needs to reformulate the concepts expressed in the medium used for queries (e.g., text) in terms of the other medium (e.g., images). In response to this demand, a wide range of methods for achieving IR on multimedia documents has been produced, mostly based on techniques developed in the areas of *signal processing* and *pattern matching,* initially foreign to the IR field. These methods are nowadays known as *similarity-based* methods, due to the fact that they use as queries an object of the same kind of the sought ones (e.g., a piece of text or an image) [6]. Originally, the term *content-based* was used to denote these methods, where the content in question was not the content of the multimedia object under study (e.g., the image) but that of the file that hosts it.

The last area is that of *semantic information processing* (SIP) which has developed across the information system and the artificial intelligence communities starting in the 1960s. The basic goal of SIP was the definition of artificial languages that could represent relevant aspects of a reality of interest (whence the appellation *semantic*), and of suitable operations on the ensuing representations that could support knowledge-intensive activities. Since the inception of the field, SIP methods are rooted in first-order mathematical logic, which offers the philosophically well-understood and computationally well-studied notions of syntax, semantics and inference as bases on which to build. Nowadays, SIP techniques are mostly employed in the context of Knowledge Organization Systems. In MIR, SIP methods have been used to develop sophisticate representations of the contents (in the sense of "semantics") of multimedia documents, in order to support the retrieval of these documents based on a logical model. According to this model, user's information needs are predicates expressed in the same language as that used for documents representations, and a document is retrieved if its representation logically implies the query. A wide range of logical models for IR have been proposed, corresponding to different ways of capturing the uncertainty inherent in IR, of expressing document contents, of achieving efficiency and effectiveness of retrieval [4].

To a lesser extent, the database area has also contributed to MIR, by providing indexing techniques for fast access to large collections of documents. Initially, typical structures such as inverted files and B-trees were employed. When similarity-based retrieval methods started to appear, novel structures, such as R- or M-trees were developed in order to support efficient processing of range and k nearest neighbors queries [15].

Foundations

MIR is a scientific discipline, endowed with many different approaches, each stemming from a different branch of the MIR history. All these approaches can be understood as addressing the same problem through a different aspect of multimedia documents.

Documents can be broadly divided from a user perspective into two main categories: simple and complex.

A document is *simple* if it cannot be further decomposed into other documents. Images and pieces of text are typically simple documents. A simple document is an arrangement of symbols that carry information via meaning, thus concurring in forming what is called the *content* of the document. In the case of text, the symbols are words (or their semantically significant fractions, such as stems, prefixes or suffixes), whereas for images the symbols are colors and textures. Simple documents can thus be characterized as having two parallel dimensions: that of *form* (or *syntax*, or *symbol*) and that of *content* (or *semantics*, or *meaning*). The form of a simple document is dependent on the medium that carries the document. On the contrary, the meaning of a simple document is the set of states of affairs (or "worlds") in which it is true, and is therefore medium-independent. For instance, the meaning of a piece of text is the set of (spatio-temporally determined) states of affairs in which the assertions made are true, and the meaning of an image is the set of such states of affairs in which the scene portrayed in the image indeed occurs.

Complex documents (or simply documents) are structured sets of simple documents. This leads to the identification of *structure* as the third dimension of documents. Document structure is typically a binary relation, whose graph is a tree rooted at the document and having the component simple documents as leaves. More complex structures may exist, for instance those requiring an ordering between the children of the same parent (such as between the chapters of a book), or those having an arity greater than 2 (such as synchronization amongst different streams of an audio-visual document).

Finally, documents, whether simple or complex, exist as independent entities characterized by (meta-) attributes (often called *metadata* in the digital libraries literature), which describe the relevant properties of such entities. The set of such attributes is usually called the *profile* of a document, and constitutes the fourth and last document dimension.

Corresponding to the four dimensions of documents just introduced, there can be four categories of retrieval, each one being a projection of the general problem of MIR onto a specific dimension. In addition, it is possible, and in some cases desirable, to combine different kinds of retrieval within the same operation.

Retrieval based on document structure does not really lead to a genuine discovery, since the user must have already seen (or be otherwise aware of) the sought document(s) in order to be able to state a predicate on their structure. Retrieval based on document profile, from a purely logical point of view, is not different from content-based retrieval and in fact many metadata schema used for document description (notable, the Dublin Core Metadata Set) include attributes of both kinds.

Form-based Multimedia Information Retrieval

The retrieval of information based on form addresses the syntactic properties of documents. In particular, form-based retrieval methods automatically create the document representations to be used in retrieval by extracting low-level features from documents, such as the number of occurrences of a certain word in a text, or the energy level in a certain region of an image. The resulting representations are abstractions which retain that part of the information originally present in the document that is considered sufficient to characterize the document for retrieval purposes. User queries to form-based retrieval engines may be documents themselves (this is especially true in the non-textual case, as this allows overcoming the medium mismatch problem), from which the system builds abstractions analogous to those of documents. Document and query abstractions are then compared by an appropriate function, aiming at assessing their degree of similarity. A document ranking results from these comparisons, in which the documents with the highest scores occur first.

In the case of text, form-based retrieval includes most of the traditional IR methods, ranging from simple string matching (as used in popular Web search engines) to the classical *tf-idf* term weighting method, to the most sophisticated algorithms for similarity measurement. Some of these methods make use of information structures, such as thesauri, for increasing retrieval effectiveness. However, what makes them form-based retrieval methods is their relying on a form-based document representation. Two categories of queries addressing text can be distinguished:

1. *Full-text* queries, each consisting of a *text pattern*, which denotes, in a deterministic way, a set of texts; when used as a query, the text pattern is supposed to retrieve any text layout belonging to its denotation.
2. *Similarity* queries, each consisting of a text, and aimed at retrieving those text layouts which are similar to the given text.

In a full-text query, the text pattern can be specified in many different ways, e.g., by enumeration, via a regular expression, or via *ad hoc* operators specific to text structure such as proximity, positional and inclusion operators [9].

Queries referring to the form dimension of images are called *visual* queries, and can be partitioned as follows:

1. *Concrete visual queries:* These consist of full-fledged images that are submitted to the system as a way to indicate a request to retrieve "similar" images; the addressed aspect of similarity may concern color [2,7], texture [8,14], appearance [12] or combination thereof [13].
2. *Abstract visual queries:* These are artificially constructed image elements (hence, "abstractions" of image layouts) that address specific aspects of image similarity; they can be further categorized into:
   a. *Color queries:* specifications of color patches, used to indicate a request to retrieve those images in which a similar color patch occurs [6,7].
   b. *Shape queries:* specifications of one or more shapes (closed simple curves in the 2D space), used to indicate a request to retrieve those images in which the specified shapes occur as contours of significant objects [6,11].
   c. Combinations of the above [2].

Visual queries are processed by matching a vector of features extracted from the query image, with each of the homologous vectors extracted from the images candidate for retrieval. For concrete visual queries, the features are computed on the whole image, while for abstract visual queries only the features indicated in the query (such as shape or color) are represented in the vectors involved. For each of the above categories of visual queries, a number of different techniques have been proposed for performing image matching, depending on the features used to capture the aspect addressed by the category, or the method used to compute such features, or the function used to assess similarity.

Semantic Content-based Multimedia Information Retrieval

On the contrary, semantic-based retrieval methods rely on symbolic representations of the meaning of documents, that is descriptions formulated in some suitable knowledge representation language, spelling out the truth conditions of the involved document. Various languages have been employed to this end, ranging from net-based to logical. Description Logics [1], or their Semantic Web syntactic forms such as OWL, are contractions of the Predicate Calculus that are most suitable candidates for this role, thanks to their being focused on the representation of concepts and to their computational amenability. Typically, meaning representations are constructed manually, perhaps with the assistance of some automatic tool; as a consequence, their usage on collections of remarkable size (text collections can reach nowadays up to millions of documents) is not viable. The social networking on which Web 2.0 is based may overcome this problem, as groups of up to thousands of users may get involved in the collaborative indexing process (flicker).

While semantic-based methods explicitly apply when a connection in meaning between documents and queries is sought, the status of form-based methods is, in this sense, ambiguous. On one hand, these methods may be viewed as pattern recognition tools that assist an information seeker by providing associative access to a collection of signals. On the other hand, form-based methods may be viewed as an alternative way to approach the same problem addressed by semantic-based methods, that is deciding relevance, in the sense of connection in meaning, between documents and queries. This latter, much more ambitious view, can be justified only by relying on the assumption that there be a systematic correlation between "sameness" in low-level signal features and "sameness" in meaning. Establishing the systematic correlation between the expressions of a language and their meaning is precisely the goal of a *theory of meaning* (see, e.g., [5]), a subject of the philosophy of language that is still controversial, at least as far as the meaning of natural languages is concerned. So, pushed to its extreme consequences, the ambitious view of form-based retrieval leads to viewing a MIR system *as an algorithmic simulation of a theory of meaning*, in force of the fact that the sameness assumption is relied upon in every circumstance, not just in the few, happy cases in which everybody's intuition would bet on its truth. At present, this assumption seems more warranted in the case of text than in the case of non-textual media, as the representations employed by form-based textual retrieval methods (i.e., vectors of weighted words) come much closer to a semantic representation than the feature vectors employed by similarity-based image retrieval methods. Irrespectively of the tenability of the sameness assumption, the identification of the alleged syntactic-semantic correlation is at the moment a remote possibility, so the weaker view of form-based retrieval seems the only reasonable option.

Mixed Multimedia Information Retrieval

Suppose a user of a digital library is interested in retrieving all documents produced after January 2007, containing a critical review on a successful representation of a Mozart's opera, and with a picture showing Kiri in a blueish dress. This need addresses all dimensions of a document: it addresses structure because it states conditions on several parts of the desired documents; it addresses profile because it places a restriction on the production date; it addresses form- (in particular color-) and semantic-based image retrieval on a specific region of the involved image (the region must be blue and represent the singer Kiri) as well as on the whole image (must be a scene of a Mozart's opera); it addresses from-based text retrieval by requiring that the document contains a piece of text of a certain type and content. This is an example of mixed MIR, allowing the combination of different types of MIR in the context of the same query [10].

Emerging standards in multimedia document representation (notably, the ISO standard MPEG21) address all of the dimensions of a document. Consequently, their query languages support more and more mixed MIR.

Key Applications

Nowadays, MIR finds its natural context in *digital libraries*, a novel generation of information systems [3], born in the middle of the 1990s as a result of the First Digital Library Initiative. Digital Libraries are large collections of multimedia documents which are made on-line available on global infrastructures for discovery and access. MIR is a core service of any DL, addressing the discovery of multimedia documents.

Cross-references

▶ Information Retrieval

Recommended Reading

1. Baader F., Calvanese D., McGuiness D., Nardi D., and Patel-Scheneider P. (eds.). The description logic handbook. Cambridge University Press, Cambridge, 2003.
2. Bach J.R., Fuller C., Gupta A., Hampapur A., Horowitz B., Humphrey R., Jain R., and Shu C.-F. The Virage image search engine: an open framework for image management. In Proc. 4th SPIE Conf. on Storage and Retrieval for Still Images and Video Databases, 1996, pp. 76–87.
3. Candela L., Castelli D., Pagano P., Thanos C. Ioannidis Y., Koutrika G., Ross S., Schek H.-J., and Schuldt H. Setting the foundations of digital libraries. The DELOS manifesto. D-Lib Magazine, 13(3/4), March/April 2007.
4. Crestani F., Lalmas M., and van Rijsbergen C.J. (eds.). Logic and uncertainty in information retrieval: advanced models for the representation and retrieval of information, The Kluwer International Series On Information Retrieval, vol. 4. Kluwer Academic, Boston, MA, October 1998.
5. Davidson D. Truth and meaning. In Inquiries into truth and interpretation. Clarendon, Oxford, UK, 1991, pp. 17–36.
6. Del Bimbo A. Visual Information Retrieval. Morgan Kaufmann, Los Altos, CA, 1999.
7. Faloutsos C., Barber R., Flickner M., Hafner J., and Niblack W. Efficient and effective querying by image content. J. Intell. Inform. Syst., 3:231–262, 1994.
8. Liu F. and Picard R.W. Periodicity, directionality, and randomness: Wold features for image modelling and retrieval. IEEE Trans. Pattern Analysis Machine Intell., 18(7):722–733, 1996.
9. Manning C.D., Raghavan P., and Schütze H. An Introduction to Information Retrieval. Cambridge University Press, Cambridge, 2007.
10. Meghini C., Sebastiani F., and Straccia U. A model of multimedia information retrieval. J. ACM, 48(5):909–970, 2001.
11. Petrakis E.G. and Faloutsos C. Similarity searching in medical image databases. IEEE Trans. Data Knowl. Eng., 9(3): 435–447, 1997.
12. Ravela S. and Manmatha R. Image retrieval by appearance. In Proc. 20th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1997, pp. 278–285.
13. Rui Y., Huang T.S., Ortega M., and Mehrotra S. Relevance feedback: a power tool for interactive content-based image retrieval. IEEE Trans. Circuits Syst. Video Tech., 8(5):644–655, September 1998.
14. Smith J.R. and Chang S.-F. Transform features for texture classification and discrimination in large image databases. In Proc. Int. Conf. Image Processing, 1994, pp. 407–411.
15. Zezula P., Amato G., Dohnal V., and Batko M. Similarity Search: The Metric Approach. Springer, Berlin, 2006.

Multimedia Metadata

FRANK NACK
University of Amsterdam, Amsterdam,
The Netherlands

Synonyms

New media metadata; Hypermedia metadata; Meta-knowledge; Mixed-media

Definition

Multimedia is media that utilizes a combination of different content forms. In general, a multimedia asset includes a combination of at least two of the following: text, audio, still images, animation, video, and some sort of interactivity. There are two categories of multimedia content: linear and non-linear. Linear content progresses without any navigation control for the viewer, such as a cinema presentation. Non-linear content offers users interactivity to control progress. Examples are computer games or computer based training applications. Non-linear content is also known as hypermedia content.

Metadata is data about data of any sort in any media, describing an individual datum, content item, or a collection of data including multiple content items. In that way, metadata facilitates the understanding, characteristics, use and management of data.

Multimedia metadata is structured, encoded data that describes content and representation characteristics of information-bearing multimedia entities to facilitate the automatic or semiautomatic identification, discovery, assessment, and management of the described entities, as well as their generation, manipulation, and distribution.

Historical Background

The first appearance of the idea of mixed media was Vannevar Bush's "Memex" system ("memory extender"). In his article "As we may think" [1], published 1945 in the "The Atlantic Monthly," he proposed a device that is electronically linked to a library, able to display books and films from the library and automatically follow cross-references from one work to another. The Memex was the first simple and elegant approach towards multimedia information access. Further important steps toward the use of multimedia in digital systems in the 1960s were the introduction of "hyperlinks and hypermedia" by Ted Nelson [2], which allowed the generation of non-linear presentations, the development of the mouse by Douglas Engelbart [3], which supported the direct manipulation of objects on a computer screen, and the invention of the GUI, developed at Xerox Parc, which introduced media into computing.

In particular, it was the development of the web and digital consumer electronics that allowed computing to leave the realm of research or government organizations and enter society in large. The development of personal computers in the 1980s enabled desktop publishing, which further enhanced in the 1990s into adequate manipulation software for digital media, such as Adobe's Photoshop and Flash, or Appel's Final Cut Express. Yet, only the development of new distribution platforms in the 1990s, such as CD-ROM, DVD, and essentially the World Wide Web (Web) in combination with the rise of media technologies, such as the digital photo and video camera, or midi and MP3 player, stimulated the swift growth of digital media in mixed form. The growing amount of mixed media available to the public, and more recently provided by the public in form of user-generated content, requested for effective access mechanisms, which resulted in a steady development of content description technologies, mainly based on metadata.

Over the years, a number of metadata standards have been developed, which address various aspects of multimedia data, such as

- Low-level features: usually automatically extracted from the content.
- Semantic features: describing high-level concepts in form of key-word, ratings and links.
- Structure and identification: describing the spatial and temporal arrangement of one or more multimedia assets.
- Management: describing the information gathered during the life cycle of the multimedia asset, such as information about reuse, archiving, and rights management.

The major organizations who contributed to the development of standards are: the Dublin Core Metadata Initiative [4], the World Wide Web Consortium (W3C) [5], the Society for Motion Pictures and Television Engineering (SMPTE) [6], the Moving Picture Expert Group (MPEG) [7], the TV-Anytime Consortium [8], and the International Press Telecommunications

Council (IPTC) [9]. The common definition language between all theses languages is in one or the other way the Extensible Markup Language (XML) [10], defined by W3C.

The two major approaches towards open standards for multimedia metadata, with respect to content description and distribution for multimedia assets, were certainly provided by the W3C and MPEG, a working group of ISO/IEC charged with the development of video and audio encoding standards.

The W3C provided, besides the well known markup languages HTML, XHTML, CSS, SVG, in particular the Synchronized Multimedia Integration Language (SMIL). SMIL enables simple authoring of interactive audiovisual presentations, which integrate streaming audio and video with images, text or any other media type. SMIL [11] is an HTML-like language facilitating the authoring of a SMIL application by using a simple text-editor. SMIL 1.0 received recommendation status in 1998, SMIL 2.0 in 2005, and SMIL 3.0 is under development while this article is written.

Work in the Media Annotation Work has started in September 2008. This working group (http://www.w3.org/2008/01/media-annotations-wg.html) is chartered to provide a simple ontology to support cross-community data integration of information related to media objects on the Web, as well as an API to access the information. In addition there is the Media Fragments Working Group (http://www.w3.org/2008/WebVideo/Fragments/), which address temporal and spatial media fragments in the Web using Uniform Resource Identifiers (URI). Both groups should finish their work by June 2010.

MPEG's contribution to multimedia metadata are certainly the three standards MPEG-4 [12], MPEG-7 [13] and MPEG-21 [14].

With MPEG-4 the group entered the realm of media content, arisen due to the growing need for content manipulation and interaction, and expanded MPEG-1 to support video/audio "objects," 3D content, low bitrate encoding and support for Digital Rights Management. With respect to multimedia metadata MPEG4 also provides content authors with a textual syntax for the MPEG-4 Binary Format for Scenes (BIFS) to exchange their content with other authors, tools, or service providers. First, XMT is an XML-based abstraction of the object descriptor framework for BIFS animations. Moreover, it also respects existing practice for authoring content, such as SMIL, HTML, or Extensible 3D (X3D) by allowing the interchange of the format between a SMIL player, a Virtual Reality Modeling Language (VRML) player, and an MPEG player through using the relevant language representations such as XML Schema, MPEG-7 DDL, and VRML grammar. As such, the XMT serves as a unifying framework for representing multimedia content where otherwise fragmented technologies are integrated and the interoperability of the textual format between them is facilitated.

In the mid 1990s, the need for retrieving and manipulating digital media content form exploding digital libraries requested new ways of describing multimedia content on deeper semantic granularity, which resulted in MPEG-7, the multimedia content description standard. MPEG-7 provides a large set of descriptors and description schemata for video (part 3), audio (part 4) and multimedia content, including its presentation (part 5). All schemata are described in the Description Definition Language (DDL), which is modeled on XML-Schema, a schemata language developed by the WC3. Since MPEG-7 tries to establish the richest and most versatile set of audio-visual feature description structures by embracing standards such as SMPTE, or PTC, a 1:1 mapping to the text-oriented XML schema language could not be achieved (see [15] for a detailed description of existing DDL problems). However, the goal to be a highly interoperable standard among well-known industry standards and related standards of other domains – such as the area of digital libraries and ontologies using RDF or Dublin Core, was a useful exercise.

The beginning of the 21st century established an even faster exchange of multimedia data via the web, as higher bandwidth as well as access to high quality data became a commodity. The media businesses, such as the record or film industry, feared, due to peer-to-peer technology, for their markets and requested strict digital rights management enforcement. MPEG reacted with MPEG-21: MPEG describes this standard as a multimedia framework.

Both the W3C as well as MPEG provide open standards, which are able to describe adaptive content, represented in a single file that can be targeted to several platforms, such as mobile, broadband or the web. Both organizations compete with highly successful but rather proprietary industry standards, such as Adobe's Flash standard.

The latest trend on multimedia metadata also reflects the trend towards user-generated content, namely social tagging. A tag is a keyword or term associated with or assigned to a piece of information (a picture, a map, a video clip etc), which enables keyword-based classification and search. The advantage of tagging is its ease of use. This approach, though highly popular (e.g., in YouTube [16] and Flickr [17]), carries serious problems. Typically there is no information about the semantics of a tag, no matter if it is a single tag or a bag of tags. Additionally, different people may use drastically different terms to describe the same concept. This lack of semantic distinction can lead to inappropriate connections.

Foundations

The essential models describing the internal structures of multimedia compositions and the related production processes are: the Dexter Hypertext Reference Model [18], the Amsterdam Hypermedia Model [19], HyTime [20], the reference model for intelligent multimedia presentation systems (IMMPSs) [21], MPEG-4, SMIL, and the model of Canonical processes of media production [22].

The basic five functionalities that every multimedia system needs to address are: media content, layout, timing, linking and adaptivity.

Media Content

Readers who are interested in the fundamentals of the single media elements (i.e., their low-level as well as high-level feature descriptions) are referred to the articles on audio metadata, image metadata and video metadata in this encyclopedia.

Layout

Layout deals with the arrangement and style treatment of media on a screen. This means that layout determines how a particular media item is presented at a point in time and how it is rendered when activated. A multimedia presentation layout describes "the look and feel" of a composite of all of its media components. Thus, layout adds the semantic organization that enables a viewer to quickly and efficiently absorb the multiple content streams in a multimedia presentation.

There are in general three approaches towards multimedia layout, namely

1. *Embedded Layout,* where all layout decisions are resolved at media creation time and then performed by the presentation. Here the control lies ultimately by the designer of the presentation and there is no control, besides the required rendering, at the side of the media player.
2. *Dynamic Layout*, where all layout is dynamically determined by the user's media player, depending on the multimedia document structure or the timeline of the presentation. Here, the actual visual design is mainly in the hand of the media player rather than the presentation's designer.
3. *Compositing Layout*, which decouples media content from media placement. Here, a presentation is understood as a composite of relatively autonomous objects, where each uses embedded or dynamic layout models, which are then positioned into a arrangement by a presentation designer.

The essential classes and their attributes that describe a layout are:

- A root and several region elements, which establish the primary connection between media objects elements and the layout structure.
- Basic layout classes, such as referencing (region name and ID), scaling (z-index fit), positioning (width, height, top, left, bottom, right), background (back ground color, show back ground), and audio (sound level).

Timing

Timing deals with issues on how elements in a multimedia presentation get scheduled. Moreover, once an element is active, it needs to be determined how long it will be scheduled. The aim in a multimedia presentation is to go beyond the timing concepts known from audio and video objects.

Media timing: Media objects in multimedia presentations can either be *discrete* (e.g., text, with no implicit duration) or *continuous* (e.g., a music object, with an explicit duration defined within its encoding).

Presentation timing: A reference list to one or more media objects, describing timing primitives that determine the start and end time relative to one another.

There are basically 4 different ways of defining the active period of an object:

1. *Implicit duration*, as defined when the object was created (e.g., length of a video in sec).
2. *Explicit duration*, which describes the actual duration of the media object in the application (e.g., the actual duration might be shorter or longer than the implicit duration).
3. *Active duration*, which allows repetitions or other temporal manipulations of a media object.

4. *Rendered duration*, which describes the persistence of a media object at the end of its active duration.

There are various ways to describe time in values, such as full clock value (e.g., 7:45:23.76, where the last two items present ms), partial clock values (any sort of short base notation), time count values (numbers with a additional type string, e.g., 10S for "10 s"), and time context values, which are represented in three parts: a date field (YYY:MM:DD), a time field, and a time zone field.

Usually, a type of synchronization is required, as media objects start in relation to the container they belong to. A child element in a parallel container starts relative to the start time of the parent, whereas child elements in a sequential container are started relative to the end of their predecessor.

Linking

Linking defines and activates a non-linear navigation structure within and across documents.

The simplest form of a link is a pointer. The pointer defines an address of a document (e.g., a URI) and, optionally, an offset within the document. The element that identifies that a link exists is called a source anchor. If the anchor points to anything other than the beginning of a document, this anchor is called a destination anchor. The typical elements in HTML for linking are the <a> </a>element, to define the source anchor and link address, and the <area> </area> element, which is roughly the same, but it is applied only to a part of a media object. The basic linking attributes define the uri (href), the source and destination state (e.g., play or pause), the external or target state (e.g., true or false, the display environment) and the impact of link activation on the source and destination presentation (e.g., as non-negative percentage).

Both source and destination anchor need to express temporal moments, as their activation depends on the temporal behavior of the application. The three key temporal moments to be addressable are: the destination is already active, the destination is inactive, and the destination is inactive and the begin time is unresolved.

Finally, the link needs some attribute that describe geometry, as the linking into a region of a media item is possible. The core attributes are shape (values can be rectangle, circle or poly). The size and position of the anchor are defined via the origin of the coordinate space (the 0.0 point) and the resolution of the display device (support of rendering).

Adaptivity

The aim of multimedia presentations is usually to adapt them to the needs of the user, which might address either the runtime environment available to the user, or the personalized presentation wishes by the user.

There are four techniques to customize information in a presentation:

1. *Minimum set:* The multimedia presentation assumes a minimum set of performance, and device and user characteristics and the presentation document is designed based on this lowest denominator set (manageable solution but usually no compelling content)
2. *Multiple presentation set*: Each presentation represents a quality level, which the user selects on runtime (this one-size-fits-all approach is a dead end for the current trend towards portable and quality mix devices).
3. *Over-specified presentation*: All of the potential media items are available and it is the media player at the client side which makes a selection at run time (demands too much during the making phase).
4. *Control presentation*: The presentation contains pointers to all potential alternatives and only those used by the user would be sent from the server to the client (the advantage is that the presentation does not need to send copies of each of the various data streams across the network).

The essential elements a system would need to provide any of the above techniques are:

- Switch: which establishes a collection of alternatives for an interactive multimedia presentation;
- System control attributes, such as sys_language, sys_-captions, sys_bitrate, sys_screensize, syt_cpu, etc.

Key Applications

Multimedia metadata is useful for the creation, manipulation, retrieval and distribution of mixed media sources within domains, such as

- The creative industries (e.g., fine arts, entertainment, commercial art, journalism, games, etc)

- The entertainment industries (e.g., special effects in movies and animations)
- Education (e.g., in computer based training courses and computer simulations, military or industrial training)
- Mathematical and scientific research (e.g., modeling and simulation)
- Medicine (e.g., virtual surgery or simulations of virus spread, etc)

Cross-references

▶ Audio Metadata
▶ Image Metadata
▶ Video Metadata

Recommended Reading

1. Bordegoni M., Faconti G., Maybury M.T., Rist T., Ruggieri S., Trahanias P., and Wilson M. A standard reference model for intelligent multimedia presentation systems (1997). Available at http://kazan.cnuce.cnr.it/papers/abstracts/9708.IJCAI97.Immps.html
2. Bush V. As we may think. Atl. Mon., 176(1):101–108, 1945.
3. Engelbart D. (1968). Available at http://sloan.stanford.edu/mousesite/1968Demo.html
4. Flickr. Available at http://www.flickr.com/
5. Gronbaek K. and Trigg R.H. Design issues for a dexter-based hypermedia system. Commun ACM., 37(2):41–49, 1994.
6. Hardman L., Bulterman D.C.A., and van Rossum G. The amsterdam hypermedia model: adding time and context to the dexter model. Commun. ACM., 37(2):5062, February 1994.
7. Hardman L., Obrenovic Z., Nack F., Kerherve B., and Piersol K. Canonical processes of semantically annotated media production. Multimedia Syst., 14(6):427–433, 2008.
8. Information processing – Hypermedia/Time-based structuring language (HyTime) – 2nd ed., ISO/IEC 10744:1997, WG8 PROJECT: JTC1.18.15.1. Available at http://www1.y12.doe.gov/capabilities/sgml/wg8/document/n1920/
9. MPEG-4: ISO/IEC JTC1/SC29/WG11 N4668 March 2002. Available at http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm
10. MPEG-21: ISO/IEC JTC1/SC29/WG11/N5231 Shanghai, October 2002. Available at http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm
11. MPEG-7: ISO/IEC JTC1/SC29/WG11N6828 Palma de Mallorca, October 2004. Available at http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm
12. Nack F., van Ossenbruggen J., and Hardman L. That obscure object of desire: multimedia metadata on the web (Part II). IEEE MultiMedia, 12(1):54–63, 2005.
13. Nelson T.H. A File Structure for the Complex, the Changing, and the Intermediate. In The New Media Reader, Noha Wardrip-Fruin & Nick Montfort. The MIT Press, Cambridge, MA, 2003, pp. 133–146.
14. SMIL. Available at http://www.w3.org/AudioVideo/
15. The dublin core metadata initiative. Available at http://www.dublincore.org/
16. The extensible markup language (XML). Available at http://www.w3.org/XML/
17. The international press telecommunications council [IPTC). Available at http://www.iptc.org/pages/index.php
18. The moving picture expert group (MPEG). Available at http://www.chiariglione.org/mpeg/
19. The society for motion pictures and television engineering (SMPTE). Available at http://www.smpte.org/home/
20. The TV-anytime consortium. Available at http://www.tv-anytime.org/
21. Youtube. Available at http://www.youtube.com/

Multimedia Presentation Databases

V. S. Subrahmanian, Maria Vanina Martinez, DR. Reforgiato
University of Maryland, College Park, MD, USA

Synonyms

Multimedia presentation databases

Definition

A multimedia presentation consists of a set of media objects (such as images, text objects, video clips, and audio streams) presented in accordance with various temporal constraints specifying when the object should be presented, and spatial constraints specifying where the object should be presented on a screen. Today, multimedia presentations range from the millions of PowerPoint presentations users have created the world over, to more sophisticated presentations authored using tools such as Macromedia Director. Multimedia presentation databases provide the mechanisms needed to store, access, index, and query such collections of multimedia presentations.

Historical Background

Multimedia presentations have been in existence since the 1980s, when PowerPoint emerged as a presentation paradigm and animated computer video games started gaining popularity. Both of these paradigms allowed a multimedia presentation author to specify a set of objects (collections of images, video, audio clips, and text objects) and then specify how these objects should be presented. These objects could be presented in accordance with some temporal constraints that describe when and in conjunction with which other objects a given object should be presented. As an increasing number of authoring frameworks came into being, accompanied by an increasing need to create, collaborate on, and share presentations, the notion of multimedia presentations as a programming paradigm gradually came into existence.

Buchanan and Zellweger [6] were one of the first to recognize the need to treat multimedia presentations in a rigorous framework. They recognized that presentations consisted of a set of media objects, and they proposed presenting these objects in accordance with some very simple precedence constraints.

Later, Candan et al. [8,7] extended the framework of Buchanan and Zellweger by describing a multimedia presentation as a set of media objects together with a very rich, but polynomially computable set of spatial and temporal constraints defining their presentation. They also showed how to help a presentation author identify when their presentation specifications were inconsistent and to minimally modify the presentation constraints so that consistency was restored. Related work by their co-authors showed how to deliver these presentations across a network in the presence of spatial and temporal constraints.

Adali et al. [1] introduced a relational model of data to support interactive multimedia presentations and define a variant of the relational algebra that allows users to dynamically query and create new presentations using parts of existing ones, generalizing select, project, and join operations. Lee et al. [11] present a graph data model for the specification of multimedia presentations, together with two icon-based based graphical query languages for multimedia presentations and the GCalculus (Graph Calculus), a relational calculus-style language that formalizes the use of temporal operators for querying presentation graphs that takes the content of a presentation into account. [5,4] propose methods to present the answer of a query to a multimedia database as a multimedia presentation. [9] focuses on specializations and improvements of the above methods when querying databases consisting solely of PowerPoint information.

[10] treats multimedia presentations like a temporal database, and supports querying and reuse of parts of existing presentations. It considers algebraic operators such as insert, delete, and join, as well as user interface operations such as Fast Forward/Rewind, Skip, and links to other presentation databases, etc. [13] focuses on indexing and retrieving complex Flash movies. A generic framework called FLAME (FLash Access and Management Environment) based on a 3-layer structure is presented to address this problem. This framework mines and understands the contents of the movies to address the representation, indexing and retrieval of the expressive movie elements, including heterogeneous components, dynamic effects, and the way in which the user can interact with the movie.

Foundations

A multimedia presentation consists of a set *O* of media objects and a set of constraints on the presentation of objects in *O*. A media-object *o* is a file such as an image file, a video file, a text file, or an audio file. Each type of file is assumed to have an associated player. For example, a video file may have QuickTime or the Windows Media players as its associated player.

The constraints associated with *o* fall into two categories: temporal and spatial constaints.

On the temporal side, each object *o* in *O* has two associated variables, *st(o)* and *et(o)* denoting, respectively, the start time and end time at which media object *o* is presented using its associated players. The *temporal specification* associated with a presentation is a set of constraints of the form

$$x - y <= c$$

where *x*, *y* are variables of the form $st(o_i)$, $et(o_j)$ and *c* is some constant. For example, if *o* is a video file, and there exists the constraint *et(o') – st(o) = 0*, then this means that the video object *o* should start playing as soon as object *o'* finishes being played out.

Likewise, on the spatial side, each object *o* in *O* has four variables *llx(o), lly(o), urx(y), ury(o)* denoting the *x* coordinate of the lower left corner of object *o*, the *y* coordinate of the lower left corner of object *o*, and likewise for the upper right corner's *x* and *y* coordinates. [8] presents algorithms to check the consistency of spatial and temporal constraints, and to minimally modify the spatial/temporal constraints when they are inconsistent. In addition, each object *o* in *O* has an associated set of properties that can be stored (and queried) using any object oriented database management system.

A *multimedia presentation database* ***M*** consists of a set of multimedia objects (and their associated spatial and temporal constraints). [1] defines methods to query such multimedia presentation databases.

Consider the simplest operation: selection. Suppose the query is "*Select all objects o in* $\boldsymbol{M}$ such that *C[o]* holds and such that $st(o) > 10$." In this case, the goal is to look at each multimedia presentation *m* in $\boldsymbol{M}$, and eliminate all objects *o* from *m* such that $st(o) < 10$. Also, it is necessary to eliminate all remaining objects such that *C[o]* does not hold. The objects that survive this elimination process must be presented in accordance with the original set of temporal and spatial constraints present in *m*. If m^* denotes the modification of *m* in this way, then $\sigma_C(\boldsymbol{M}) = \{m^* \mid m \text{ in } \boldsymbol{M}\}$.

In addition, [1] defines other operations that allow videos to be concatenated together using various chromatic composition operators (such as smoothing, fading, etc.), methods to perform joins across videos, and methods to execute other kinds of relational style operations.

Key Applications

There are numerous possible applications for multimedia presentation databases. A simple application is an engine to query PowerPoint presentations, of which millions exist in the world today. [9] proposes a PowerPoint database query algebra.

Another application is in the area of digital rights management. Suppose a major record company wants to identify all multimedia documents on the web that contain a clip of their copyrighted music. This corresponds to a select query on all multimedia documents on the web. The result would be a set of multimedia documents, some of which might infringe on the copyright holder's rights. The same might apply to online video on sources such as YouTube where it is not uncommon to find copyrighted material. The ability for an entertainment company to find gross violations of their copyright by searching through YouTube archives is critical.

Future Directions

As video games become ever more common, and as virtual worlds such as Second Life become increasingly popular with users, the ability to query games and *avatar* based systems will become increasingly important.

In addition, methods to index multimedia presentation databases are in their very infancy. Taking into account the graph based nature of presentation databases such as those in [1,11,2], it is important to note that methods to index graphs may have a role to play. However, multimedia presentations are more complex than labeled directed graphs because presentation constraints can potentially be satisfied in many different ways.

Cross-references

▶ Spatial Constraints
▶ Temporal Constraints

Recommended Reading

1. Adali S., Sapino M.L., and Subrahmanian V.S. A multimedia presentation algebra. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 121–132.
2. Adali S., Sapino M.L., and Subrahmanian V.S. Interactive multimedia presentation databases, I: algebra and query equivalences. Multimedia Syst., 8(3):212–230, 2000.
3. Bailey B., Konstan J.A., Cooley R., and Dejong M. Nsync – a toolkit for building interactive multimedia presentations. In Proc. 6th ACM Int. Conf. on Multimedia, 1998, pp. 257–266.
4. Baral C., Gonzalez G., and Nandigam A. SQL+D: extended display capabilities for multimedia database queries. In Proc. 6th ACM Int. Conf. on Multimedia, 1998, pp. 109–114.
5. Baral C., Gonzalez G., and Son T.C. Design and implementation of display specification for multimedia answers. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 558–565.
6. Buchanan M.C. and Zellweger P. Automatically generating consistent schedules for multimedia documents. Multimedia Syst., 1(2):55–67, 1993.
7. Candan K., Lemar E., and Subrahmanian V.S. View management in multimedia databases. VLDB J., 9(2):131–153, 2000.
8. Candan K.S., Prabhakaran B., and Subrahmanian V.S. CHIMP: a framework for supporting distributed multimedia document authoring and presentation. In Proc. 4th ACM Int. Conf. on Multimedia, 1996, pp. 329–340.
9. Fayzullin M. and Subrahmanian V.S. An algebra for powerpoint sources. Multimedia Tools Appl. J., 24(3):273–301, 2004.
10. Jiao B. Multimedia presentation database system. In Proc. 8th ACM Int. Conf. on Multimedia, 2000, pp. 515–516.
11. Lee T., Sheng L., Bozkaya T., Balkir N.H., Özsoyoglu Z.M., and Özsoyoglu G. Querying multimedia presentations based on content. In Readings in Multimedia Computing and Networking, K. Jeffay, H. Zhang (eds.). Morgan Kaufmann Publishers, San Francisco, CA, USA, 2001, pp. 413–437.
12. Wirag S. Modeling of adaptable multimedia documents. In Proc. Interactive Distributed Multimedia Systems and Telecommunication Services, International Workshop, LNCS vol. 1309, 1997, pp. 420–429.
13. Yang J., Li Q., Wenyin L., and Zhuang Y. Content-based retrieval of FlashTM movies: research issues, generic framework, and future directions. Multimedia Tools Appl., 34(1):1–23, 2007.

Multimedia Resource Scheduling

Jeffrey Xu Yu
Chinese University of Hong Kong, Hong Kong, China

Definition

Multimedia information systems are different from the traditional information systems, where continuous media (audio/video) requests special storage and delivery requirements due to (i) the large transfer rate, (ii) the storage space required, and (iii) the real-time and continuous nature. Due to the special characteristic of continuous media, different types of scheduling are proposed, namely, the disk scheduling and stream scheduling. On one hand, the disk scheduling is to tackle both the large storage space and the corresponding large transfer rate requirements. On the other hand, the stream scheduling is to schedule requests from multiple clients, in order to minimize the delay in satisfying the requests. It attempts to support as many requests as possible, and at the same time, keep the real-time and continuous nature.

Historical Background

Continuous media adds additional requirements to the traditional information systems. In order to satisfy these new requirements, new scheduling algorithms are proposed.

Disk scheduling is designed to achieve the low service latency and high disk throughput requirements [14]. Continuous media, such as audio and video, adds additional real-time constraints to the disk scheduling problem. There are two categories of disk scheduling, namely, single disk scheduling and multiple disk scheduling. The disk scheduling algorithms, for continuous media, can possibly adapt one of the conventional disk scheduling strategies, which include round-robin, SCAN [14], and EDF [12]. For single disk scheduling for continuous media, there are SCAN-EDF [13], and sorting-set [8,16] scheduling strategies. For multiple disk scheduling for continuous media, multiple disks can be accessed in parallel. The increased parallelism is mainly used to support more streams. There are two categories of multiple disk scheduling strategies, which are stripe data across the disks and replicate data across the disks. In additional, there are three schema for accessing striped data, which are called, striped retrieval, split-striped retrieval [15], and cyclic retrieval [2,3].

Stream scheduling (or session scheduling) is designed to effectively allocate and share server and network resources. It aims at supporting as many clients as possible simultaneously, within the limited server and network resources, by assigning multiple client requests to a shared data stream. Three policies are proposed to effectively select the multimedia to be shared: FCFS, MQL, and FCFS-n [10]. In [6,7], sharing is studied when a client pauses/resumes while viewing a long video. A two-level hierarchical scheduling policy is proposed to deal with time-varying load, such as the load at peek time and off-peek time, regarding the preservation of resources for the future requests [1]. In the two-level hierarchical scheduling, at the higher level, it focuses on channel allocation with consideration of how many requests will come in near future; at the lower level, it focuses on selecting clients to be served with consideration of the clients waiting time.

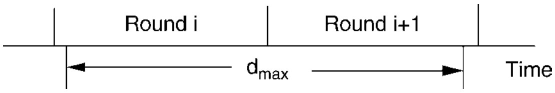
Foundations

There are mainly two types of resource scheduling in a multi-stream environment: (i) disk scheduling, and (ii) stream scheduling.

Disk Scheduling

In a multi-stream environment, multiple users are requesting to retrieve N continuous multimedia data streams in a similar time period. The system will serve each of the N data streams in rounds. In other words, in any i-th round, the system will serve all the requested data streams N by reading enough data for all the requests to be consumed until the next i+1-th round. Because there are differences between the transfer rate and the consumption rate, a scheduling strategy needs to deal with the differences between the transfer rate and the consumption rate.

Let r_c and r_t be the rate of display consumption of a data stream and the rate of data transfer from disk, respectively. And let p_{max} and d_{max} be the maximum time interval in any round and the maximum time interval between two consecutive reads for any data stream, respectively. Figure 1 illustrates the ideas on



Multimedia Resource Scheduling. Figure 1. Round length and delay between reads (Fig. 3 in [9]).

round-length and delay between two consecutive disk reads. In order to have enough data to be consumed by a display, at the consumption rate of r_c, in each round of the time interval at most p_{max} (to prevent starvation until the next round), the amount of data it needs is at least $r_c \cdot p_{max}$. The d_{max} shows the possible delay, called startup delay, to start consuming a data stream, which implies the time interval it needs to wait if it misses in the current round.

Consider the disk scheduling for multiple data streams on a single disk. The round-robin algorithm retrieves data for each data stream in a fixed order in every round. Due to the fixed order, the maximum startup delay (d_{max}) is similar to p_{max}. The SCAN algorithm [13] attempts to read more disk blocks while moving the disk head over the disk, and retrieve the requested blocks when the disk header passes over them [14]. The main advantages of the SCAN algorithm are: (i) it reduces the seek time to move the disk header from one location to another in order to read next disk block, and (ii) it maximizes the throughput of disk accesses. But, because the order of serving each data stream is not fixed, the startup delay for a given data stream can be larger up to $2p_{max}$. The SCAN-EDF algorithm [13] processes the data stream requests with the earliest deadline first, using the idea discussed in EDF [12] (Earliest-Deadline-First), and processes the requests, that be shared, using the SCAN algorithm. Note that the SCAN-EDF algorithm different from the other strategies, is not designed as a round-based algorithm. The sorting-set algorithm [8,16] is designed in a way where round-robin and SCAN are treated as special cases of it. In brief, in the sorting-set algorithm, each round is further divided into several time slots. Each time slot is conceptually considered as a sorting set (or simply set). If a round is divided into m time slots, there are m time slots, and therefore there are m sorting sets, namely, set_1, set_2,..., and set_m. All the sorting sets are served in a fixed order in each run. When there is a request to retrieve a multimedia data stream, the requested data stream is assigned to a sorting set. If the entire round is treated as a single time slot, the sorting-set algorithm behaves like the SCAN algorithm. If the requested data stream is assigned to a unique sorting set, the sorting-set algorithm behaves like the round-robin algorithm.

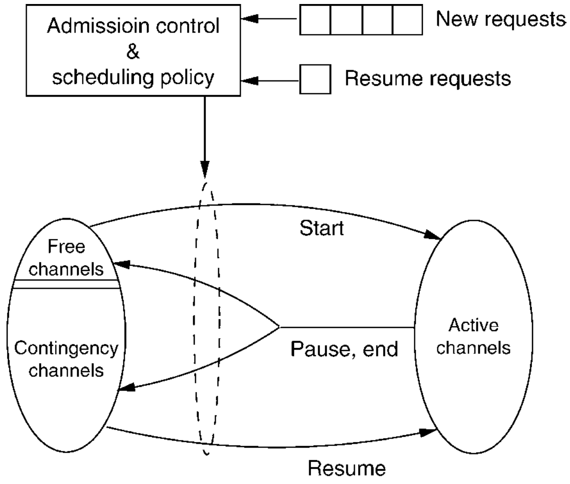
For multiple disks accessing in parallel, there are two main categories of multiple disks scheduling strategies, namely, stripe data across the disks and replicate data across the disks. There are also different schema to retrieve the striped data: striped retrieval, split-striped retrieval, and cyclic retrieval. In the striped retrieval, entire stripes are retrieved in parallel. The split-stripe retrieval [15] retrieves some consecutive units of an entire stripe rather than the entire stripe at one time. The cyclic retrieval [2,3] is designed to retrieve units of stripes for more than one data stream. The main idea behind it is to read a small portion of data for a data stream frequently. As comparison with the other striped approaches, cyclic retrieval does not need a large buffer space. Unlike the stripe based algorithms, data streams can be replicated across disks where each disk is treated individually and independently. If a data stream is requested frequently, it can be replicated in multiple disks.

Stream Scheduling

Continuous data stream retrieval needs to be guaranteed by reserving sufficient resources. The resources are referred to as logical channels (or simply channels). Stream scheduling policies need to increase the server capacity, or in other words, to increase the number of continuous data stream requests to be served with the limited number of channels. The stream scheduling needs to consider several facts regarding the possibility of sharing. There are popular videos which are viewed by many clients most of the time during a certain time period. Several clients may view the same video but start viewing at different times in a short time interval. A client may pause and then resume when viewing a long multimedia (video) at any time. The time interval between such pause and resume is not known, which can be short or long. A client may also change to another video after the pause.

Data stream sharing serves multiple clients by a single I/O stream, which is also referred to as batching of requests [4,6]. A batching factor indicates how many clients can share a single I/O stream. In order to effectively support continuous multimedia requests, some requests need to be delayed, in order to be batched with other requests. There are three batching policies: FCFS, MQL, and FCFS-n [10]. With the FCFS (First Come First Served) policy, it queues all requests into a single queue. When there is a channel available, the FCFS policy selects the video, which is requested by the first client in the queue, to be served, in a first come first served fashion. If there are other requests in the queue that request the same video, they will be served by sharing the I/O stream. The MQL (Maximum Queue Length) policy maintains a queue for a requested video. If there are *n* videos to be requested, there will be *n* queues. The MQL policy chooses the video with the maximum queue length to be served, when a channel becomes available. With the MQL policy, the videos with a small number of requests (short queue) may not be served. Therefore, unlike FCFS which is a fair policy, the MQL policy is seen as unfair to the videos that are not requested by many clients. The FCFS-n policy is similar to the FCFS policy, except that the *n* hottest videos are assigned to dedicated streams. A dedicated stream will be served in a batching window in turn, and the remaining videos, that are not assigned to a dedicated stream, are served following the FCFS policy. With the policies, the batching factor can be increased, but the amount of waiting time for a client to wait may be increased as well.

Sharing a data stream is affected by the fact that a client may pause. A new channel may need to be allocated when the client resumes. Assume that a client may resume shortly after the pause, the system may maintain some channels (contingency channels), which can improve resource utilization. The system needs to guarantee that the delay between the receipt of a resume request from a client and playback is small in order to assure client satisfaction [4,6]. Figure 2 illustrates the scheduling with contingency channels for VCR control operations (pause/resume) [6,7]. The admission control policy determines the acceptance of a new request. The scheduling policy determines which request to be served on the available channel, in order to maximize certain performance objectives [6]. When there is a pause request from a client, the admission control and scheduling policy determines if the client is the only client viewing the multimedia stream. If it is, the channel is freed and returned to either the free channel pool or the contingency pool. A certain number of contingency channels are maintained in the system. When there is a resume request from a client, the scheduling policy checks if there is another request being served within a predetermined time window, in order to maximize the possibility of sharing. If there is another request being served but is beyond the predetermined time window, the scheduling policy tries to use another contingency channel where possible. Otherwise, it will request a free channel with higher priority when such a free channel becomes available.



Multimedia Resource Scheduling. Figure 2. Channel states under contingency policy (Fig. 2 in [5]).

The arrival rate of requests to multimedia system may vary with the time of a day [11], peek time and off-peek time. The scheduling policy also needs to address the issues of time-varying load. Note: the optimal policy at the current may not be the optimal when the load changes shortly. A two-level hierarchical scheduling policy is proposed to deal with such load fluctuations [1]. The high-level scheduler controls channels allocation, whereas the low-level scheduler controls the clients selection to be served. Three high level policies are proposed to control channel allocation rate [1]: on-demand allocation, forced-wait, and pure rate control. The on-demand allocation allocates channels to requests when there are channels available. Under this policy, the waiting time for new requests may be long, which may cause clients to change from one video to another to view at high possibility. The forced-wait policy, as the name implies, forces the first request to a data stream to wait for up to a certain time (minimum wait time). The minimum wait time is a critical parameter, and can be difficult to be selected in a dynamic environment where the load changes dynamically. The pure rate control policy allocates channels uniformly in fixed time intervals (called measurement intervals) during such a time interval only a certain number of channels are used [1].

Key Applications

In a multimedia environment, multimedia objects are retrieved from either a digital library, or a video database, or an audio database, and are delivered to a large number of clients. The applications in such environments range from retrieving small multimedia objects (shopping, medias, education, etc.) to playback of large video objects (movie, entertainment, etc.).

Cross-references

- ▶ Continuous Multimedia Data Retrieval
- ▶ Multimedia Data Buffering
- ▶ Multimedia Data Storage
- ▶ Scheduler
- ▶ Scheduling Strategies Storage Resource Management

Recommended Reading

1. Almeroth K.C., Dan A., Sitaram D., and Tetzlaff W.H. Long Term Channel Allocation Strategies for Video Applications. IBM Research Report (RC 20249), 1995.
2. Berson S., Ghandeharizadeh S., Muntz R., and Ju X. Staggered striping in multimedia information systems. ACM SIGMOD Rec., 23(2):79–90, 1994.
3. Chen M.S., Kandlur D.D., and Yu P.S. Storage and retrieval methods to support fully interactive playout in a disk-array-based video server. Multimedia Syst., 3(3):126–135, 1995.
4. Dan A., Shahabuddin P., Sitaram D., and Towsley D. Channel allocation under batching and VCR control in video-on-demand systems. J. Parallel Distrib. Comput., 30(2):168–179, 1995.
5. Dan A. and Sitaram D. Multimedia Information Storage and Management, chap. 11: Session Scheduling and Resource Sharing in Multimedia Systems. Kluwer Academic, 1996.
6. Dan A., Sitaram D., and Shahabuddin P. Scheduling policies for an on-demand video server with batching. In Proc. 2nd ACM Int. Conf. on Multimedia, 1994, pp. 15–23.
7. Dey-Sircar J.K., Salehi J.D., Kurose J.F., and Towsley D. Providing VCR capabilities in large-scale video servers. In Proc. 2nd ACM Int. Conf. on Multimedia, 1994, pp. 25–32.
8. Gemmell D.J. Multimedia network file servers: multi-channel delay sensitive data retrieval. In Proc. 1st ACM Int. Conf. on Multimedia, 1993, pp. 243–250.
9. Gemmell D.J. Multimedia Information Storage and Management, chap. 1: Disk Scheduling for Continuous Media. Kluwer Academic, 1996.
10. Ghose D. and Kim H.J. Scheduling video streams in video-on-demand systems: a survey. Multimedia Tools Appl., 11(2): 167–195, 2000.
11. Little T.D.C. and Venkatesh D. Prospects for interactive video-on-demand. IEEE Multimedia, 1(3):14–24, 1994.
12. Liu C.L. and Layland J.W. Scheduling algorithms for multi-programming in a hard-real-time environment. J. ACM, 20(1): 46–61, 1973.
13. Reddy A.L.N. and Wyllie J.C. I/O issues in a multimedia system. Computer, 27(3):69–74, 1994.
14. Teorey T.J. and Pinkerton T.B. A comparative analysis of disk scheduling policies. Commun. ACM, 15(3):177–184, 1972.
15. Tobagi F.A., Pang J., Baird R., and Gang M. Streaming RAID: a disk array management system for video files. In Proc. 1st ACM Int. Conf. on Multimedia, 1993, pp. 393–400.
16. Yu P.S., Chen M.S., and Kandlur D.D. Grouped sweeping scheduling for DASD-based multimedia storage management. Multimedia Syst., 1(3):99–109, 1993.

Multimedia Retrieval Evaluation

THIJS WESTERVELD[1,2]
[1]Teezir Search Solutions, Ede, The Netherlands
[2]CWI, Amsterdam, The Netherlands

Definition

Multimedia Retrieval Evaluation is the activity of measuring the effectiveness of one or more multimedia search techniques. A common way of evaluating multimedia retrieval systems is by comparing them to each other in community wide benchmarks. In such benchmarks participants are invited to submit their retrieval results for a given set of topics, the relevance of the submitted items is checked, and effectiveness measures for each of the submissions are reported. Multimedia retrieval evaluation measures the effectiveness of multimedia retrieval systems or techniques by looking at how well the information need as described by a topic is satisfied by the results retrieved by the system or technique. Efficiency of the techniques is typically not taken into account, but may be studied separately.

Historical Background

Until the mid-1990s, no commonly used evaluation methodology existed for multimedia retrieval. An important reason for this is that the field has merely been a showcase for computer vision techniques. Many papers in the field 'proved' the technical merits and usefulness of their approaches to image processing by showing a few well-chosen, and well-performing examples. Since 1996, the problem of systematically evaluating multimedia retrieval techniques has gained more and more interest. In that year, the *Mira* (Multimedia Information Retrieval Applications) working group was formed [2]. The group, consisting of people from the fields of information retrieval, digital libraries, and library science studied user behavior and information needs in multimedia retrieval situations. Based on their findings, they developed performance measures. Around the same time, in the multimedia community, the discussion on proper evaluation started, and Narasimhalu et al. [8] proposed measures for evaluating content-based information retrieval systems. These measures are based on comparing ranked lists of documents returned by a system to the perfect, or ideal, ranking. However, they do not specify how to obtain such a perfect ranking, nor do they propose a common test set. A year later, Smith [10] proposed to evaluate image retrieval using measures from the text retrieval community and in particular from *TREC*, the Text Retrieval Conference [12] for image retrieval evaluation. Again, no dataset was proposed. At the start of the twenty-first century, the evaluation problem gained more attention within the content-based image retrieval community, with the publication of three papers discussing benchmarking in visual retrieval [3,6,7]. These three papers call for a common test collection and evaluation methodology and a broader discussion on the topic. The *Benchathlon* network (http://www.benchathlon.net) was started to discuss the development of a benchmark for image retrieval. Then, in 2001, *TREC* started a video track [9] that evolved into the workshop now known as *TRECVID*.

Today, a variety of initiatives exists for evaluating the retrieval of different types of data in a variety of contexts, a list of these is provided below under data.

Foundations

Information retrieval is interactive. In web search, for example, queries are often changed or refined after an initial set of documents has been retrieved and inspected. In multimedia retrieval, where browsing is common, interactivity is perhaps even more important. Evaluation should take interactivity into account, and measure user satisfaction. The evaluation of a system as a whole in an interactive setting is often called an *operational test*. Such tests measure performance in a realistic situation. Designing such an operational test is difficult and expensive. Many users are needed to free the experiment of individual user effects, the experimental setup should not interfere with the user's natural behavior, and learning effects need to be minimized. Also, because there are many free variables, it is hard to attribute observations to particular causes. In contrast to these tests in fully operational environments, *laboratory tests* are defined as those tests in which possible sources of variability are controlled. Thus, laboratory tests can provide more specific information, even though they are further away from a realistic setting. Also, laboratory tests are cheaper to set up, because the interactive nature is ignored, and the user is removed from the loop. Laboratory tests measure the quality of the document ranking instead of user satisfaction. While some studies exist to evaluate multimedia retrieval systems in an operational setting by investigating user satisfaction, most approaches are studied in laboratory tests.

Current laboratory tests are based on the *Cranfield* paradigm [1]. In this paradigm, a test collection consists of a fixed set of documents, a fixed set of topics, and a fixed set of relevance judgements. Documents are the basic elements to retrieve, topics are descriptions of the information needs, and relevance judgements list the set of relevant documents for each topic.

The focus in laboratory tests is on comparative evaluation. Different approaches are tested, and their relative performance is measured. The process is as follows. Each approach produces a ranked lists of documents for each topic. The quality of the ranked lists is measured based on the positions of the relevant documents in the list. The results are averaged across all topics to obtain an overall quality measure.

For successful evaluation of retrieval techniques, two components are needed in addition to a test collection [4]: a measure that reflects the quality of the search and a statistical methodology for judging whether a measured difference between two techniques can be considered statistically significant. The measures used in multimedia retrieval evaluation are typically based on precision and recall, the fraction of retrieved documents that is relevant and the fraction of relevant documents that is retrieved. To measure recall, the complete set of relevant documents needs to be known. For larger collections this is impractical and a *pooling* method is used instead. With pooling, the assumption is that with a diverse set of techniques contributing runs to the evaluation, the probability that a relevant document is retrieved at a high rank by at least one of the approaches is high. A merged set of top ranked documents is assumed to contain most relevant documents and only this set of documents is manually judged for relevance. Documents that are not judged are assumed not relevant. In reality, some unjudged documents may certainly still be relevant, but it has been shown that this is of no influence to the comparative evaluation of search systems [11,13,14].

A number of aspects influence the reliability of evaluation results. First, a sufficiently large set of topics is needed. ?] suggest a minimum of 75. Second, the measures should be stable. This means it should not be influenced too much by chance effects. Clearly, measures based on few observations are less stable than measures based on many observations. For example, precision at rank 1 (is the first retrieved document relevant) is not a very stable measure. Third, there needs to be a reasonable difference between two approaches before deciding one approach is better than the other. Sparck Jones [5] suggests a 5% difference is noticeable, and a difference greater than 10% is material. Statistical significance tests take all these aspects into account and are useful in deciding whether an observed difference between two approaches is meaningful or simply due to chance.

Key Applications

Multimedia retrieval evaluation helps to better understand what works and what does not in the area of multimedia retrieval. Many practitioners in the field benefit from the area. It gives them the opportunity to test their ideas in a principled manner and allows them to build upon approaches that are known to be successful. More and more, the papers published in renowned journals and conferences demonstrate the usefulness of their techniques by a thorough evaluation on a well-known test collection.

Data Sets

Creating a test collection for multimedia retrieval systems takes a lot of effort. Especially the generation of ground truth data for a sufficient number of topics is something that a small company or research institution cannot manage on its own. Many workshops exist that solve this problem by sharing resources in a collaborative effort to create valuable and re-usable test collections. This approach was first taken in the Text Retrieval Conferences (TREC) [12], but many others followed. Below the main collections and evaluation platforms in multimedia are listed.

The *Corel* document set is a collection of stock photographs, which is divided into subsets each relating to a specific theme (e.g., *tigers*, *sunsets*, or *English pub signs*). The collection is often used in an evaluation setting by using the classification into themes as ground truth. Given a query image, all images from the same theme –and only those– are assumed relevant. Evaluation results based on Corel are highly sensitive to the exact themes used in the evaluation [7]. In addition, Corel has a clear distinction between themes and an unusually high similarity within a theme because the photos in a theme often come from the same photographer or even the same location. This makes the collection more homogeneous than can be expected in a realistic setting.

TRECVID studies video retrieval. The data collections used have been dominated by broadcast news, but other raw and edited professional video footage is studied as well. TRECVID defines a number of tasks and provides test collections for each of them. In 2007, four main tasks existed:

Shot boundary detection: identify shot boundaries in the given video clips with their location and type (hard or soft transition).

High level feature extraction: For each of the predefined high-level features or concepts, detect the shots that contain the feature. Features that have been studied in the past include sky, road, face, vegetation, office and people marching.

Search: Given a textual description of an information need and/or one or more visual examples, find shots that satisfy this need.

Rushes summarization: Given a set of rushes, i.e., raw, unedited footage, provide a visual summary of this data that in a limited number of frames shows the key objects and events that are present in the footage.

As part of the Cross-Language Evaluation Forum (CLEF), *ImageCLEF* studies cross-language image retrieval. ImageCLEF concentrates on two main areas: retrieval of images from photographic collections and retrieval of images from medical collections.

The Initiative for the Evaluation of XML retrieval (INEX) aims to evaluate the effectiveness of XML retrieval systems. Within this initiative, the *INEX multimedia track* evaluates the retrieval of multimedia elements from a structured collection. The data collection used consists of wikipedia documents and the images contained in them. Both the retrieval of multimedia fragments (combinations of text and images) and the retrieval of images in isolation are studied.

ImagEVAL evaluates image processing technology for content-based image retrieval. The assessment focuses on features relating to what collection holders (from defence, industry and cultural sectors) expect in terms of how images may be used. The tasks include recognizing transformed images, combined textual and visual search and object detection. The collections are a heterogeneous mix of professional images including stock photography, museum archives and industrial images.

The Music Information Retrieval Evaluation Exchange (*MIREX*) evaluates subtasks of music information retrieval. The datasets used by MIREX are cd-quality audio originating from (internet) record labels that allow tracks of their artists to be published. The tasks include genre classification, melody extraction, onset detection, tempo extraction and key finding.

Cross-references

- ► Information Retrieval Evaluation Measures
- ► Multimedia Databases
- ► Multimedia Information Retrieval

Recommended Readings

1. Cleverdon C.W. The cranfield tests on index languagr devices. Aslib Proc., 1967, pp. 173–192.
2. Draper S.W., Dunlop M.D., Ruthven I., and van Rijsbergen C.J. (eds.). In Proc. Mira 99: Evaluating Interactive Information Retrieval. Electronic Workshops in Computing, 1999.
3. Gunther N.J. and Beretta G. A benchmark for image retrieval using distributed systems over the internet: BIRDS-I. Technical Report HPL-2000-162, HP Laboratories, 2000.
4. Hull D. Using statistical testing in the evaluation of retrieval experiments. In Proc. 16th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 329–338.
5. Jones K.S. Automatic indexing. J. Doc., 30:393–432, 1974.
6. Leung C.H.C. and Ho-Shing Ip H. Benchmarking for content-based visual information search. In Advances in Visual Information Systems, Fourth Int. Conf., 2000, pp. 442–456.
7. Müller H., Müller W., McG. Squire D., Marchand-Maillet S., and Pun T. Performance evaluation in content-based image retrieval: overview and proposals. Pattern Recogn. Lett. (Special Issue on Image and Video Indexing), 22(5):593–601, 2001.
8. Narasimhalu A.D., Kankanhalli M.S., and Wu J. Benchmarking multimedia databases. Multimedia Tools Appl., 4(3):333–356, 1997. ISSN 1380-7501.
9. Smeaton A.F., Over P., Costello C.J., de Vries A.P., Doermann D., Hauptmann A., Rorvig M.E., Smith J.R., and Wu L. The TREC-2001 video track: information retrieval on digital video information. In Research and Advanced Technology for Digital Libraries, Sixth European Conference, 2002, pp. 266–275.
10. Smith J.R. Image retrieval evaluation. In Proc. IEEE Workshop on Content-based Access of Image and Video Libraries, 1998, pp. 112–113.
11. Voorhees E.M. and Harman D.K. Overview of the eighth text retrieval conference (TREC-8). In Proc. The 8th Text Retrieval Conference, 2000.
12. Voorhees E.M. and Harman D.K. TREC: Experiment and Evaluation in Information Retrieval (Digital Libraries and Electronic Publishing). MIT, Cambridge, MA, 2005. ISBN 0262220733.
13. Westerveld T. Trecvid as a re-usable test-collection for video retrieval. In Proc. Multimedia Information Retrieval Workshop, 2005.
14. Zobel J. How reliable are the results of large-scale information retrieval experiments? In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 307–314.

Multimodal Data

► Multimedia Data

Multimodal Databases

► Multimedia Databases

Multi-modal Information Retrieval

► Cross-Modal Multimedia Information Retrieval

Multimodal Interfaces

Monica Sebillo[1], Giuliana Vitiello[1],
Maria De Marsico[2]
[1]University of Salerno, Salerno, Italy
[2]Sapienza University of Rome, Rome, Italy

Definition

Multimodal interfaces are characterized by the (possibly simultaneous) use of multiple human sensory modalities and can support combined input/output modes.

The term *multimodal* recurs across several domains. Since its first use in the field of interface design, its affinity and derivation from the terms "mode" and "modality" were discussed. According to Merriam-Webster, one of the meanings for *mode* is "a possible, customary, or preferred way of doing something," whereas *modality* can be "one of the main avenues of sensation (as vision)." In *multimodal interfaces*, the former influences the way information is conveyed, the latter refers to the exploited communication channel. Both express peculiar aspects of a multimodal system, which is expected to provide users with flexibility and natural interaction.

Early multimodal interfaces simply combined display, keyboard, and mouse with voice (speech recognition/synthesis). Later, pen-based input, hand gestures, eye gaze, haptic input/output, head/body movements have been progressively used. As a result, modern multimodal interfaces aim at emulating the natural multi-sensorial forms of human-human dialogue, relying on the integration of advanced interaction modes. To support the described variety of modes, the system underlying such an interface must include hardware to acquire and render multimodal expressions and must exploit recognition-based technologies to interpret them, with response times compatible with user's interaction pace.

Historical Background

Bolt's "Put That There" demonstration system [2] can be considered as one of the earliest multimodal systems. The underlying technology allowed a joint use of voice and gesture by processing speech in parallel with manual pointing within a virtual graphical space, where users could be made easily aware of the available facility and its usage. Spoken input was semantically processed, while deictic terms in the speech were resolved by processing the spatial coordinates derived from pointing. Since then, different proposals of multimodal interfaces can be found in literature [5,4]. Many of them aim at integrating natural languages and direct manipulation in specific application domains, such as manufacturing environments and interactive maps.

The *CHI'90 Workshop on Multimedia and Multimodal Interface Design* represented a turning point [1]. The growing interest by the international community toward this research area induced the Program Committee to focus on both the integration of scientific knowledge about this discipline and the direction of future developments. Four main topics were identified for discussion, with respect to which interfaces could be designed and examined: structural principles for composition, media appropriateness, enabling technologies, and paradigms/metaphors/models/representations. Important issues emerged, from which some years later the first remarkable set of guidelines for the design of multimodal user interfaces stemmed [9].

Following the scientific attention to the multimodal paradigm, a variety of systems rapidly came out. Both hardware and software were enhanced to integrate parallel input streams, mostly acquired by speech and pen-based gestures. One of the first such prototypes was the QuickSet system developed in 1994 [3]. It was an agent-based, collaborative, multimodal-multimedia map system, allowing the user to issue commands using a combination of speech and pen input. For illustration purposes, in [8] Oviatt provides a comparison among five different speech and gesture systems, which represented the most exemplifying research-level systems developed until the late 1990s. In her evaluation, the author takes into account some characteristics, such as the type of signal fusion and the sizes of gesture and speech vocabularies, on which integration and interpretation functionalities were based.

More recently, new combinations of speech and other modalities, such as lip movements and gaze, have been exploited thanks to advanced input/output technologies, whose effective integration into flexible yet reliable interfaces requires the underlying system robustness and performance stability. As a result, multimodal interfaces have pervaded new application domains, ranging from virtual reality systems, meant to support expert users in decision making and simulation of scenarios, to training systems and to person identification/verification systems for security purposes. Such systems may be further distinguished on the basis of their input modes, which can be either intentionally exploited by users (active input mode) such as speech and gestures, or captured by the system without an explicit request by users (passive input mode), such as facial expressions.

Foundations

In order to clarify the different aspects involved in a multimodal interface, the complexity of multimodal interaction can be described in terms of the well-known *execution-evaluation cycle* defined by Don Norman in 1988 [6].

In the case of multimodal interactive cycles, Norman's model of interaction may be reformulated as follows:

1. Establishing the goal.
2. Forming the intention.
3. Specifying the multimodal action sequence in terms of human output modalities.
4. Executing the multimodal action.
5. Perceiving the system state in terms of human input modalities.
6. Interpreting the system state.
7. Evaluating the system state with respect to the goals and the intentions.

- *Establishing the goal* is, as usual, the stage when the user determines what needs to be done in the given domain, in terms of a suitable task language.
- *Forming the intention*: at this stage the goal is translated into more precise user's intention, which will help the user determining the right sequence of actions that should be performed to achieve the goal
- *Specifying the multimodal action sequence.* The sequence of actions performed to accomplish the required task should be precisely stated at this stage. Here the complexity of multimodal interaction appears for the first time in the cycle. In fact, each multimodal action can be specified in terms of:
  1. Complementary human output sensory modalities (i.e., multiple utterances at once form the action) and/or
  2. Alternative human output sensory modalities (i.e., alternative, redundant utterances for the same action).

Examples of human output modalities include speaking, gesturing, gazing, touching, moving, facial expressions and some unintentional utterances such as blood pressure, temperature, heartbeat, excretion, etc. A user may, for instance, move an object in the interaction scene by speaking and pointing at (gesturing) the new object location (complementary modalities). Then, instead of gesturing (s)he may want to gaze at the new location on the interface where the object should be moved (alternative modality).

- *Executing each multimodal action.* At this stage, each human modality used to specify an action is translated into corresponding interaction modes. Thus, each action is executed through
  1. Complementary modes or
  2. Alternative modes

Text, speech, Braille, mimicking, eye/motion capture, haptics, bio-electrical sensoring are examples of modes used to translate human output modalities into the system input language.

When the execution of the whole sequence of multimodal actions is complete, the system reaches a new state and communicates it to the user again exploiting (possibly multiple) interaction modes, such as speech synthesis, display, haptic/tactile feedback, smell rendering and so on.

- *Perceiving the system state.* At this stage, the evaluation phase of the cycle begins. Depending on the combination of system output modes, the user may perceive the new state through multiple input sensory modalities, such as visual, auditory, tactile, and (in some revolutionary interfaces) even smelling and tasting.
- *Interpreting the system state.* Here the user is supposed to interpret the output of her/his sequence of actions to evaluate what has happened.
- *Evaluating the system state with respect to the goals and the intentions.* At the final stage, the user compares the new system state with her/his expectations, to evaluate if the initial goal has been effectively reached.

Of course a good mapping should be achieved between the execution and the evaluation phases in order to bridge what Norman calls the *gulf of execution* (i.e., the distance between user's specification of the action and the actions allowed by the system) and the *gulf of evaluation* (i.e., the distance between user's perception of the new state and her/his expectation). In multimodal interfaces the effective reduction of both gulfs crucially depends on the underlying interactive technology that must move as close as possible towards human-human forms of interaction and communication. Thus, on the execution side, human output modalities should be supported by suitable computer input devices, e.g., by mapping gaze, speech, touch, gesturing and smelling onto cameras, microphones, keyboard, haptic sensors and the most recent olfactory sensors, respectively. From the evaluation perspective, computer output and its perception by user should be also tightly linked. This requires the adoption of output devices, like display, audio and haptic/olfactory rendering devices, able to quickly and effectively reach the human input sensory system, so that the user sees, hears, touches and, in general, feels the new system state as it is communicated by the multimodal interface.

As an example, the successful execution of direct manipulation tasks within an immersive virtual reality environment may critically depend on the abolition of any latency time between the moment an action is performed, e.g., by means of datagloves, and the moment the user recognizes the touch. In this way the user perceives gesturing as an act that can be directly realized at the interface. If this cannot be achieved, any usability benefits coming from the direct manipulation paradigm would be lost.

The ultimate advantage of multimodal interfaces is increased usability, in terms of both flexibility and robustness of the interaction when either redundant or complementary information is conveyed by modes. Higher flexibility is gained since multimodal interfaces can accommodate a wide range of users, tasks and environments for which each single mode may not be sufficient. Different types of information may be conveyed using the most appropriate or even less error prone modality, while alternation of different channels may prevent from fatigue in computer use intensive tasks. Redundancy of information through different communication channels is especially desirable when supporting accessibility, since users with different impairments may benefit from information and services otherwise difficult to obtain. As for the increased robustness of the interaction, the weaknesses of one modality may be offset by the strengths of another. More semantically rich input streams can support mutual disambiguation for the execution phase. As in human-human communication, the correct decoding of transmitted messages requires interpreting the mix of audio-video signals.

An important research theme in multimodal interface design is how to integrate and synchronize different modes, taking into account that synchrony of different "tracks" of interaction in different modes, does not imply their simultaneity. At present, each unimodal technique is developed separately, with noticeable advances produced by improvements in both software recognition-based techniques and hardware input/output technologies. However, as pointed out in [9], an effective integration of the involved modal technologies requires a deep understanding of the "natural" integration patterns that characterize people's combined use of different communication modes, as widely studied by psychologists and cognitive experts. The issue of integration may become even more complex when a multimodal interface is designed to support collaborative work, namely the work by multiple users who may interact through the interface using several input/output modes, either synchronously or asynchronously, and either locally or remotely.

Key Applications

Recent advances in technology have been urging IT researchers to investigate innovative multimodal interfaces and interaction paradigms, able to exploit the increased technological power. The common key goal is to reproduce in the best possible way the interaction through different channels, typical of a human-human dialogue. As an example, real "physical" manipulation might be simulated even when the latter is not possible due to logistical problems (e.g., remote operation) or to dangerous settings (e.g., radioactive materials and areas), or when it is convenient to just simulate a real operation (e.g., for training purposes). A more natural and familiar way of managing objects and situations is also expected to improve global user performances and increase applications effectiveness.

Among the most recent efforts, research on haptic equipment deserves a mention. Related advances trigger new potentialities and convey novel features towards many domains, especially industrial, medical, and biotechnological. In the industrial world, the goal of improving competitiveness has led to the experimentation of haptic interfaces in fields like automotive and aerospace engineering, and texture manufacturing. In the medical domain, education and research activities are being increasingly improved by the adoption of haptic environments for virtual surgery simulation. Several new challenges are arising in the field of Biology/Biotechnology, where the adoption of visual interfaces connected to haptic devices is recognized as a powerful and straightforward mean to handle nano-objects, such as cells, and the possibility of force feedback offered by certain haptic systems, is envisioned as a considerable improvement of operator's perception. Last but not least, several multimodal interfaces enhanced with haptic feedback have been conceived to address major societal needs, e.g., by visually impaired people or wheelchair users.

In the following, a brief list of some further application domains is presented, where multimodal interfaces are presently investigated.

Interaction in Mobile Environments

The problem that has to be solved in applications designed for mobile environments is that hands, which are the usual interaction mediator for human-computer communication with traditional input devices, must be devoted to different crucial activities, e.g., controlling a steering wheel. In such situation, alternative modes should rather be exploited to interact with software applications such as a map browser. Moreover, user's visual attention must be focused on catching situations such as obstacles approach, so that relevant software events should be communicated for example through auditory signals, so as to relieve the user from continuously inspecting system state.

Geographic Information Systems

Multimodal interfaces are also being employed as a means to support decision makers in accessing and analyzing geospatial information in specific and critical scenarios, such as crisis management procedures and *what if* analyses. Some systems have been recently proposed, which rely on large screen displays and augmented reality tools for enhanced data visualization, as well as on collaborative advanced interfaces supporting speech and gesture recognition.

In these systems, multimodality becomes the way domain expert users can formulate appropriate requests to the underlying geographic information system and receive rapid responses, provided through different perspectives. Rapid feedback is in fact a crucial issue in situations when risk and vulnerability must be predicted as well as during exceptional events when recovery actions must be taken by users with complementary expertises.

Interaction in Adverse Settings

As discussed above, it is often necessary to substitute the human operator in adverse settings in a way that preserves both his/her health and the effectiveness of the interaction with environment objects. A much simpler case is when some communication channel might be hindered by disturbing conditions and the presence of other modes may provide possible missing information.

Multimodal Biometric Databases

Multimodal biometric databases are an example of tight integration of multiple input modes to achieve reliable person identification and verification. Fingerprints are the most well-known biometric method. More biometrics include hand conformation, iris scanning, features from face, ears or voice or handwriting. Despite noticeable progresses in biometrics research, no single bodily or behavioral trait satisfies acceptability, speed and reliability constraints of authentication in real applications. Single biometric systems are vulnerable to possible attacks, and may suffer from acquisition failures, or from the possible non-universality of the biometric feature, as in the case of deaf-mute subjects for voice recognition. The present trend is therefore towards multimodal systems, as flaws of an individual system can be compensated by the availability of a higher number of alternative biometries. Integration of single responses is a crucial point, especially when different reliability degrees can be assigned to them due to input quality or effectiveness of recognition algorithms.

Interaction in Impairment Conditions

Accessibility is a transversal issue relating to different application domains. Physical impairments call for flexible system interfaces allowing universal access to services and information. What should be affected when designing for accessibility is the structure of both input and output for each application function. Functions need parameters including both data and events triggered by user's actions. In both cases, it is necessary to adapt the format manageable by a user possibly bearing a specific disability to the one acceptable by the functions. Such adaptation could be provided by special pieces of software (wrappers). A different wrapper is needed for each different disability situation. They would be connected to suitable interfaces allowing the user to issue commands and data according to his/her ability, and translating them for function call. Information and data returned by the system undergoes a symmetrical translation. In other words, multimodal input/output should be dynamically provided. The contribution of (disabled) accessibility experts to the overall design of wrappers is essential to obtain significant results. They can suggest the best suited interaction mechanisms and the best input/output modes to use.

Future Directions

The future challenge for multimodal interfaces is the ability to better and better mimic human-like sensory perception. Such interfaces will be able to reliably interpret continuous input from more different visual, auditory, and tactile sources, chosen according to the target users' tasks. More advanced recognition of users' natural communication modalities will be supported, and more sophisticated models of multimodal interaction are expected to replace present bimodal systems. One of the problems to solve is to design and implement effective integration schemas among different modalities, based on available literature on human intersensory perception and on natural human-human multimodal interaction patterns. More research is required on human inclination to multimodal communication with applications, depending on different target tasks, and about integration and synchronization characteristics of multimodal input/output in different contexts and situations.

Cross-references

- ▶ Geographic Information System
- ▶ Mobile and Ubiquitous Data Management
- ▶ Visual Interfaces

Recommended Reading

1. Blattner M.M. and Dannenberg R.B. CHI'90 Workshop on multimedia and multimodal interface design. SIGCHI Bull., 22(2):54–58, 1990.
2. Bolt R.A. Put that there: voice and gesture at the graphics interface. ACM Comput. Graph., 14(3):262–270, 1980.
3. Cohen P.R., Johnston M., McGee D.R., Oviatt S.L., Pittman J., Smith I., Chen L., and Clow J. QuickSet: multimodal interaction for distributed applications. In Proc. 5th ACM Int. Conf. on Multimedia, 1997, pp. 31–40.
4. European Telecommunications Standards Institute. Human Factors (HF); Multimodal interaction, communication and navigation guidelines ETSI EG 202 191 V1.1.1 (2003–08).
5. Jaimes A. and Sebe N. Multimodal human-computer interaction: a survey. Comput. Vis. Image Underst., 108:116–134, 2007.
6. Norman D. The design of everyday things. Doubleday, New York, 1988.
7. Oviatt S. Ten myths of multimodal interaction. Commun. ACM, 42(11):74–81, 1999.
8. Oviatt S. Multimodal interfaces. In The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications, J. Jacko, A. Sears (eds.). Lawrence Erlbaum, NJ, 2003.
9. Reeves L.M., Lai J., Larson J.A., Oviatt S., Balaji T.S., Buisine S., Collings P., Cohen P., Kraal B., Martin J.C., McTear M., Raman T.V., Stanney K.M., Su H., and Wang Q.Y. Guidelines formultimodal user interface design. Commun. ACM, 47(1):57–59, 2004.
10. Yuen P.C., Tang Y.Y., and Wang P.S.P. (eds.). Multimodal Interface for Human-Machine Communication. World Scientific, NJ, 2002.

Multi-Pathing

KALADHAR VORUGANTI
Network Appliance, Sunnyvale, CA, USA

Definition

There can be multiple paths between a SCSI initiator and a SCSI target. Multiple paths between a host and a storage device are useful to provide more fault-tolerance as well as to improve system throughput. Multi-pathing software ensures that the same target volume is not seen as two separate LUNs by the host.

Key Points

The multiple paths can be configured in active-active or active-standby modes. In the active-active mode, both paths are actively transferring data. In the active-standby mode, the standby path does not actively transfer data. Some multi-pathing software allows for dynamic load balancing of traffic between the multiple paths. Typically, the storage controller vendor also provides the multi-pathing driver that runs on the host, and this software is usually limited to only operating with the vendor's storage devices. Software vendors are beginning to provide multi-pathing software that can interoperate with storage controllers from multiple storage vendors.

Cross-references

- ▶ Initiator
- ▶ LUN
- ▶ Target
- ▶ Volume

Multiple Classifier System

▶ Ensemble

Multiple Imputation

▶ Synthetic Microdata

Multiple Linked Plots

▶ Dynamic Graphics

Multiple Query Optimization

▶ Multi-Query Optimization

Multiple Representation Modeling

Christine Parent[1], Stefano Spaccapietra[2],
Christelle Vangenot[2], Esteban Zimányi[3]
[1]University of Lausanne, Lausanne, Switzerland
[2]EPFL, Lausanne, Switzerland
[3]Free University of Brussels, Brussels, Belgium

Synonyms

Multi-scale; Multi-resolution; Multi-granularity modeling

Definition

Geodata management systems (i.e., GIS and DBMS) are said to support *multiple representations* if they have the capability to record and manage multiple representations of the same real-world phenomena. For example, the same building may have two representations, one with administrative data (e.g., owner and address) and a geometry of type point, and the other one with technical information (e.g., material and height) and a geometry of type surface. Multirepresentation is essential to make a data repository suitable for use by various applications that focus on the same real world of interest, while each application has a specific perception matching its goals. Different perceptions translate into different requirements determining what information is kept and how it is structured, characterized, and valued. A typically used case is map agencies that edit a series of national maps at various scales and on various themes.

Factors that concur in generating different representations include the intended use of data and the level of detail matching the applications concerns. The former rules the choice of data structures (which objects, relationships, and attributes are relevant) and of value domains (e.g., whether the temperatures are stored in Celsius or Fahrenheit). The latter rules data resolution from coarse to precise and impacts both the semantic and the spatial representation.

Multiple representation modeling is the activity of designing a data repository that consistently holds multiple representations for various perceptions of a given set of phenomena. It relies on a multirepresentation data model, i.e., a data model with constructs and rules to define and differentiate the various perceptions and for each perception the representations of the phenomena of the real world of interest.

Historical Background

Support for different requirements over the same data set has first been provided by using multiple data files, each one designed for a specific application. Aiming at consistency, databases looked instead for ways to gather data into a single repository, generating the need for multiple representations. First came facilities to define application-specific subschemas, as simple restrictions of the database schema. Later, more flexibility was achieved through the view mechanism. In object-oriented terms, views are virtual representations derived from existing data. They create either an alternative representation for existing objects (object preserving views) or new objects composed from existing objects (object generating views). Each view defines a single new representation. However, views do not provide multiple perceptions, i.e., there is no possibility to identify the collection of views that forms a consistent whole for an application.

Similarly, the concept of is-a link was borrowed from artificial intelligence to provide for various representations of the same object in a classification refinement hierarchy. However, the concept comes with a population inclusion constraint: The subtype population is included in the supertype population. This cannot cope with situations where the populations of two object types only overlap, i.e., the two populations may have specific objects not represented in the other population. Since then, the situation has not changed much. It is only in the last decade that the need for more flexible multirepresentation and explicit support of various perceptions has been stressed by researchers.

Multirepresentation research in spatial databases can be traced back to the 1989 NCGIA program. In the early 1980's, maps began to be stored in geographic databases, and that opened up many new research tracks. The specification and implementation of cartographic generalization was one of them. It relied on the idea that cartographic generalization would allow the automatic derivation of maps at different scales from a single geographic database. Realizing that this full automation was not possible increased the focus on multiple representation databases [7,9,15]. Under the pressure of delivering maps at different scales, the first researches were, in the early 1990's, focusing on multi-scale data structures (i.e., data structures allowing to retrieve geometry of real-world objects for any given scale) and multiscale databases (i.e., databases in which the data for maps at different scales is stored and linked together). Later, the multiscale approach was refined and extended into the multirepresentation approach: scale indeed is not a concept relevant for databases and there is nowadays more to a geographical database than producing maps. The current scope of multirepresentation for GIS database comprises various techniques that can be used, within a single database, to automatically derive a coarser representation from another representation at finer resolution. These techniques aim at computing objects at coarser representations, for multiresolution analyses rather than for display. They include generic cartographic generalization operations (not driven by map display considerations) as well as operations performed on specific object types (e.g., selection of instances based on an ad-hoc predicate, aggregation of instances to create new objects). Some authors use the term model generalization to denote that the use of various techniques leads to the creation of a set of virtual databases for different resolution levels, and the mappings between their schemas (models in GIS terms). The automatic derivation rules (the mappings) allow update propagation from finer to coarser representations.

Currently, only a few simple multi-representation databases exist and are used to their expected potential.

Foundations

Early research on multi-representation in GIS was driven by cartographers' requirements. This explains why the ability to draw maps at different scales has been for long the targeted objective, popularizing the concept of *multi-scale databases*. However, many other spatial applications that need to perform spatial data analysis require storing and managing specific representations where objects may have various geometries (derived one from another or not) and also show varying thematic characteristics (e.g., have different attributes and different relationships). Thus, the research domain evolved from multi-scale databases to *multi-representation databases*. Equally important is the capability to provide each application with a consistent set of data that corresponds to its own perception of the real world of interest, in short its own database. Therefore, support of multirepresentation should be complemented with support of multiperception.

Multiscale Databases

Multiresolution databases are still referred to by the GIS community as multiscale databases, despite the fact that scale does not apply to data storing. Scale is a concept related to the drawing of maps on paper or on screen. It is the ratio between measures on a map and the corresponding measures in the real world. Scale only characterizes an intended use of data. Instead, the level of resolution of a spatial database determines what geometries are stored. It defines a threshold such that only geometries beyond the threshold are captured and stored.

In early work by Timpf, the different map representations of the same real-world entities are interconnected using a directed acyclic graph data structure. The graph allows users to navigate among maps at different scales by zoom-in and zoom-out operations. This work later developed into a more general Map Cube Model [14], supported by a theory on the structure of series of maps of the same region at different scales. Each map is described as a composition of four components: a set of lines representing transportation and hydrology networks, the set of areas, called containers, created by the lines of the trans-hydro network, areas that are a refinement of the container partition, and objects contained in these areas. The elements stored in each of the four components (e.g., streets, land-use areas, buildings) are then organized into aggregation and/or generalization hierarchies. This forms a graph for each component, where each level of the graph corresponds to a given scale.

Stell and Worboys have also proposed a solution to link a series of maps [12]. Their database organization is called a stratified map space. Each map gathers objects of a particular region that share the same semantic and spatial granularity. Maps are grouped by map spaces, i.e., sets of maps at the same granularity, describing various regions. The stratified map space is the set of all maps spaces organized according to a hierarchy based on different granularity levels. Transformation functions allow users to navigate in a stratified map space and propagate updates.

Multi-Representation Databases

Work on spatial multi-representation databases has followed two main tracks: either proposing new (conceptual) data models that include explicit description of multi-representation, or proposing frameworks that organize a set of existing classic (i.e., without multi-representation) databases into a global multirepresentation repository.

Database Models for Multiple Representations Several data models with specific concepts for multiple-representation modeling have been proposed. They range from simple solutions allowing users to associate various geometries to the same objects to more sophisticated solutions. They are discussed here according to the requirements for multiple-representation modeling.

A model for multirepresentation should allow one to characterize the same objects using different sets of attributes, and attributes with different values and different domains. This flexibility is supported by the MADS model [8], where multiple representations of a given phenomenon may be organized according to two strategies. In the first one, the various spatial and semantic descriptions of the same real-world phenomenon are merged into a single database construct. Each element of a description is qualified by a tag (called stamp) whose value identifies the perceptions for which it is relevant. Object and relationship types can thus have various sets of attributes depending on the perception. Attributes may bear various cardinalities or value domains according to the perception stamp; they can also contain a value that is a function of the perception stamp.

The Vuel approach [2] also offers the possibility of associating various semantic and spatial descriptions to the same real-world entities. In addition, various graphical representations, useful for drawing maps at different scales, can be defined. The data model is a snowflake model for spatial data warehousing. The fact table is composed of a specific kind of tuples, called vuels. A vuel fact is a particular representation of a real-world entity. It has three components: a geometry, a graphical description, and a semantic description. The vuel representation may vary according to these three dimensions. Moreover, the semantic dimension is a fact table itself with four dimensions: the class, the attribute, the domain of value, and the value dimensions. This allows the creation of various semantic descriptions by combining the dimensions (different classes, different sets of attributes, attributes with various domains and various values). OMT-G [3], a UML-based model, supports the modeling of multiple representations of data through a specific kind of relationship called conceptual generalization relationship. This relationship allows the definition of various views of the same real-world entities as subclasses of a shared super-class. The superclass describes the thematic attributes that are common to all the representations and it has no spatial representation. Each subclass describes its own view by specifying its own thematic and spatial attributes. The subclasses inherit the common attributes from the superclass. A presentation diagram shows graphical representations that may be associated to a class and the operations to obtain them. MRSL [5], another UML-based model, supports multi-representation through the introduction of two concepts: representation objects (r-objects) and integration objects (i-objects). All r-objects corresponding to the same real-word phenomenon are linked by a monovalued or multivalued link to a single i-object, whose role is to ensure consistency among them. Each r-object specifies a specific set of attributes and values for the same real-world object.

Multiple representation modeling is not limited to associating multiple sets of attributes or values to one object. In particular, when changing the level of detail, objects may disappear, whereas others may be grouped. Thus, in addition, there is the need to put into correspondence one object with several objects or two different sets of objects.

In the second strategy of the MADS approach, the various descriptions of the same phenomena belong to separate object types. They can be linked by inter-representation links that are either traditional associations or multi-associations. Multi-associations are binary relationships that, contrarily to association relationships, do not link two objects but two groups of objects. A multi-association is needed whenever the real-world entities are not represented per se, but through two different decompositions; e.g., a decomposition of a road in segments according to the number of lanes, and one according to crossroads. The other modeling approaches only support correspondence links of kind association, and the supported cardinality of the link varies: in MRSL, a i-object can be linked to r-objects through 1:1 and 1:N links thus providing support for the 1:N and N:M correspondences. In Vuel, corresponding objects can also be linked through 1:1 or 1:N inter-representation links. However, there is no support for N:M inter-representation links. OMT-G does not support inter-representation associations between objects.

Not only do objects need multiple representations, but relationships also do. This is only supported by

MADS. In MADS, all characteristics of a relationship may have various representations: its semantics (e.g., topological, aggregation, or plain), its roles, and its cardinalities. For instance, a relationship type can be a topological adjacency relationship in one description and a near relationship in another one with a more precise resolution.

In the spatial context, as data from one representation may often result from the derivation of the same data represented at another resolution, the representations of the same real-world entity are not independent and one may expect to be able to state constraints between these representations. Consistency constraints in databases are maintained through the definition of integrity constraints. Some constraints, such as cardinalities, are embedded in the concepts of the model – in particular some constraints are inherent to the multiple-representation concepts – while other constraints need to be defined in the application. MRSL is the only model proposing specific multirepresentation constraints: three kinds of rules can be associated to an i-object and its linked r-objects: consistency rules, which can be object or value correspondences, matching rules, and restoration rules. Matching rules specify how to match objects representing the same entity. They can be attribute comparison, spatial match operations, or global identifiers. Restoration rules are used to restore consistency between an i-object and its r-objects when needed.

Finally, considering that a multirepresentation database contains several representations of the same real-world phenomena, it is important to associate metadata to the representations to identify the application(s) they are relevant for, but also in order to know which representations together form a consistent whole for the application. This important requirement is fulfilled by MADS through the concept of perception stamp. In MADS, a perception stamp is a vector of values (e.g., a viewpoint and a resolution) that identifies a particular perception, and all elements of the database (types, properties, instances) are stamped for defining for which perception they are relevant. In Vuel, the designer can define views that are compositions of vuels. Each view defines a particular perception, thus providing a functionality similar to perception stamps.

Architectures for Distributed Representations Instead of proposing new concepts allowing users to integrate multiple representations of the same real-world phenomena into a unique multirepresentation database, other proposals followed a less intrusive approach. Capitalizing on the fact that there already exist many spatial databases, these approaches create a multirepresentation framework out of a set of existing classic (i.e., describing a unique perception and resolution) databases. There are two main kinds of proposals: the first one focuses on the *definition of links between objects* in corresponding databases, the second one aims at building *federated database management systems*.

In the first category, the work of Kilpelainen [6] was one of the first proposals tackling multiple representations from a database point of view. It supports bidirectional links that allow one to propagate updates in both directions and perform reasoning processes in the form of generalization operators.

In federated spatial databases, users access a set of databases through a single integrated schema, which describes virtual multirepresentation objects. During query processing, multirepresentation objects are dynamically constructed by merging all the corresponding monorepresentation objects that exist in the various databases. There have been several proposals for spatial database integration [4]. Particularly interesting are those that build the integrated schema using multirepresentation concepts, e.g., [5], based on MRSL, and [11], based on MADS. Using MRSL, each r-object in the integrated schema holds an UML tag that identifies the corresponding source database. Using MADS, perception stamps can fulfill the same functionality.

Key Applications

Cartography

As they cannot automatically derive maps at different scales from a single detailed database, national map agencies have to create several databases, one per scale. For them, multirepresentation modeling is crucial for two main reasons:

1. To propagate updates [1]: The cost of updating can be lowered by entering updates only once in a database and propagating them, at least semi-automatically, to the other databases.
2. To enforce consistency [10]: Multi-representation databases play an important role in order to

enforce consistency between the same data described at different levels of details. In addition, integrating existing databases to create a multirepresentation database allows one to detect inconsistencies between the databases.

Multi-Scale Analysis

Multirepresentation databases can benefit many applications that need to analyze data at different levels of details or defined for different viewpoints. For example, a fire monitoring application may need very detailed data on current fires (to direct the action of fire brigades as precisely as possible), only need medium-level resolution data for records of past fires, and use low-level resolution data for generic organization of fire management activities.

Other candidate applications are those relying on spatial data warehouses, using spatial OLAP and spatial data cubes to perform multi-dimensional analysis. An example is traffic accident monitoring applications, e.g., for analysis of the number of deadly accidents according to multilevel criteria (by road, region, department, or state). Multirepresentation storage of spatial data is needed in order to drill-up and drill-down the cube [2].

Future Directions

Work in progress explores the use of multirepresentation capabilities in support of modularization of knowledge repositories. In particular, the semantic web community is developing various approaches to turn huge ontologies that are being built in several knowledge domains into smaller sets of more manageable ontological modules. Existing approaches follow both the integrated direction (a single ontology is modularized) and the distributed direction (various existing ontologies are interconnected within a global knowledge sharing system). A forthcoming book on Ontology Modularization [13] is due for publication in 2008.

Cross-references

▶ Database Design
▶ Distributed Spatial Databases
▶ Field-Based Spatial Modeling
▶ Geographic Information System
▶ Multidimensional Modeling
▶ Semantic Modeling for Geographic Information Systems
▶ Spatial and Spatio-Temporal Data Models and Languages
▶ Spatial Data Types
▶ Topological Data Models
▶ Topological Relationships

Recommended Reading

1. Badard T. and Lemarié C. Propagating updates between geographic databases with different scales, chapter 10. In Innovations in GIS 7: GIS and GeoComputation, P. Atkinson, D. Martin (eds.). Taylor and Francis, London, UK, 2000, pp. 135–146.
2. Bédard Y. and Bernier E. Supporting multiple representations with spatial view management and the concept of VUEL. In Proc. Joint Workshop on Multi-Scale Representations of Spatial Data, 2002.
3. Borges K., Davis C.A., and Laender A. OMT-G: an object-oriented data model for geographic applications. GeoInformatica, 5(3):221–260, 2001.
4. Devogele T., Parent C., and Spaccapietra S. On spatial database integration. Int. J. Geogr. Inf. Syst., 12(4):335–352, 1998.
5. Friis-Christensen A., Jensen C.S., Nytun J.P., and Skogan D. A conceptual schema language for the management of multiple representations of geographic entities. Trans. GIS, 9(3):345–380, 2005.
6. Kilpelaïnen T. Maintenance of topographic data by multiple representations. In Proc. Annual Conference and Exposition of GIS/LIS, 1998, pp. 342–351.
7. Mustière S. and Van Smaalen J. Database requirements for generalisation and multiple representations. In Generalisation of Geographical Information: Cartographic Modelling and Applications, W.A. Mackaness, A. Ruas, T. Sarjakoski (eds.). Elsevier, Amsterdam, 2007.
8. Parent C., Spaccapietra S., and Zimányi E. Conceptual Modeling for Traditional and Spatio-temporal Applications. The MADS Approach. Springer, Berlin, 2006.
9. Sarjakoski L.T. Conceptual models of generalisation and multiple representation. In Generalisation of Geographical Information: Cartographic Modelling and Applications, W.A. Mackaness, A. Ruas, T. Sarjakoski (eds.). Elsevier, Amsterdam, 2007, pp. 11–36.
10. Sheeren D., Mustière S., and Zucker J.D. How to integrate heterogeneous spatial databases in a consistent way? In Proc. 8th East-European Conf. Advances in Databases and Information Systems, 2004, pp. 364–378.
11. Sotnykova A., Vangenot C., Cullot N., Bennacer N., and Aufaure M.-A. Semantic mappings in description logics for spatio-temporal database schema integration. Journal on Data Semantics III:143–167, 2005.
12. Stell J.G. and Worboys M.F. Stratified map spaces: a formal basis for multi-resolution spatial databases. In Proc. 8th Int. Symp. on Spatial Data Handling, 1998, pp. 180–189.
13. Stuckenschmidt H., Parent C., and Spaccapietra S. (Eds.). Modular Ontologies. Springer LNCS, 2009.
14. Timpf S. Map cube model: a model for multi-scale data. In Proc. 8th Int. Symp. on Spatial Data Handling, 1998, pp. 190–201.

15. Weibel R. and Dutton G. Generalizing spatial data and dealing with multiple representations. In Geographical Information Systems: Principles, Techniques, Management and Applications, vol. 1, 2nd edn., P. Longley, M.F. Goodchild, D.J. Maguire, D.W. Rhind (eds.). Wiley, 1999, pp. 125–155.

Multiplicity

▶ Statistical Disclosure Limitation For Data Access

Multiprocessor Data Placement

▶ Parallel Data Placement

Multiprocessor Database Management

▶ Parallel Database Management

Multiprocessor Query Processing

▶ Parallel Query Processing

Multi-Query Optimization

Prasan Roy[1], S. Sudarshan[2]
[1]Aster Data Systems, Inc., Redwood City, CA, USA
[2]Indian Institute of Technology, Bombay, India

Synonyms

Multiple query optimization; Global query optimization; Common subexpression elimination; Optimization of DAG-structured query evaluation plans

Definition

Multi-query optimization is the task of generating an optimal combined evaluation plan for a collection of multiple queries. Unlike traditional single-query optimization, multi-query optimization can exploit commonalities between queries, for example by computing common sub-expressions (i.e., subexpressions that are shared by multiple queries) once and reusing them, or by sharing scans of relations from disk.

Historical Background

Early work on multi-query optimization includes work by Sellis [11], Park and Segev [7] and Rosenthal and Chakravarthy [9]. Shim et al. [12] consider heuristics to reduce the cost of multi-query optimization. However, even with heuristics, these approaches are extremely expensive for situations where each query may have a large number of alternative evaluation plans.

Subramanian and Venkataraman [13] consider sharing only among the best plans of the query; this approach can be implemented as an efficient, post-optimization phase in existing systems, but does not guarantee optimality. In fact, Roy et al. [10] show that it can be significantly suboptimal. Rao and Ross [8] address the problem of sharing common computation across multiple invocations of a subquery, which is a special case of multi-query optimization,

Roy et al. [10] address the problem of extending top-down cost-based query optimizers to support multi-query optimization, and present greedy heuristics, as well as implementation optimizations. Their techniques were shown to be practical and to give good results. Dalvi et al. [1] explores the possibility of sharing intermediate results by pipelining, avoiding unnecessary materializations. Diwan et al. [2] consider issues of scheduling and caching in multi-query optimization. Zhou et al. [14] discuss the implementation of multi-query optimization on a commercial query optimizer.

In addition to the motivation of optimizing a collection (batch) of queries, multi-query optimization has also been applied to other settings. For example, Mistry et al. [6] consider the issue of multi-query optimization in the context of view maintenance, while Fan et al. [3] point out the importance of multi-query processing in optimizing XPath queries.

Foundations

Multi-query optimization is more expensive than independent optimization of multiple queries, since a globally optimal plan may involve subplans that are sub-optimal for the individual queries.

Consider a batch consisting of two queries ($A \bowtie B \bowtie C$) and ($B \bowtie C \bowtie D$). A traditional system would evaluate each of these queries independently, using the individual best plans suggested by the query optimizer for each of these queries. Let these best plans be as shown in Fig. 1a. Suppose the base relations A, B, C and D each have a scan cost of 10 units (the actual unit of measure is not relevant to this example). Each of the joins have a cost of 100 units, giving a total evaluation cost of 460 units. On the other hand, in the plan shown in Fig. 1b, the common subexpression ($B \bowtie C$) is first computed and materialized on the disk at a cost of 10. Then, it is scanned twice – the first time to join with A in order to compute ($A \bowtie B \bowtie C$), and the second time to join it with D in order to compute ($B \bowtie C \bowtie D$) – at a cost of 10 per scan. Each of these joins have a cost of 100 units. The total cost of this *consolidated* plan is thus 370 units, which is about 20% less than the cost of the traditional plan of Fig. 1a. Although the benefit here is small, it could be significantly more for batches containing more queries.

The expression ($B \bowtie C$) that is common between the two queries ($A \bowtie B \bowtie C$) and ($B \bowtie C \bowtie D$) in the above example is a *common subexpression* (CSE). A relation used in multiple queries can be thought of as a special case of a common subexpression. Although there is no need to compute and store it, a scan of the relation from disk can be shared by multiple queries.

A plan for a single complex query can have common subexpressions within itself. Traditional optimizers ignore the possibility of exploiting such a common subexpression, but some of the techniques for multi-query optimization, such as [10] can exploit such common subexpressions.

Challenges

The job of a multi-query optimizer can be broken into two parts: (i) recognize possibilities of shared computation by identifying CSEs, and (ii) find a globally optimal evaluation plan exploiting the CSEs identified.

Identifying CSEs Each query can have a large number of alternative evaluation plans. Given a particular evaluation plan for each of a set of queries, it is straightforward to find common subexpressions amongst these plans. However, since the number of possible combinations of such plans is very large, enumerating them is not feasible.

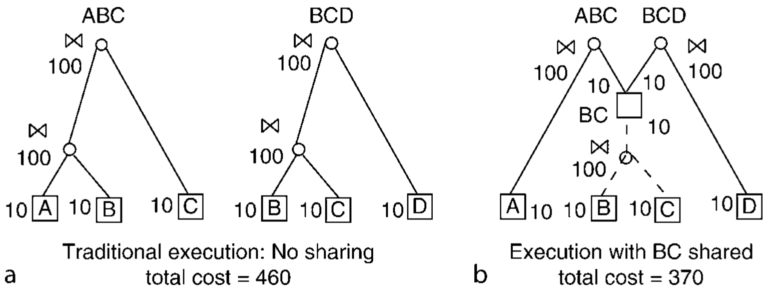
Subexpressions that could be shared between some plans for two or more queries can however be identified without enumerating all possible plan combinations. The number of such potentially common subexpressions is still very large, but smaller than the number of plan combinations.

Finding the Optimal Plan in Presence of CSEs Traditional query optimizers use dynamic programming algorithms to find the best plan for an input query. These dynamic programming algorithms are applicable because, in the absence of sharing of common subexpressions, each subplan of the overall best plan is also the best plan for the subexpression it computes.

In the presence of sharing, such a property does not hold – as shown in the example above, a globally optimal plan can consist of subplans that are not globally optimal – and therefore a straightforward dynamic programming approach does not work. The problem of finding an optimal combined plan in presence of CSEs is therefore a strictly harder problem than traditional query optimization.

Engineering an Efficient Multi-Query Optimizer

As mentioned earlier, a simple minded approach that iterates over all possible plans for each query and analyzes each combination of plans is very expensive, and infeasible for non-trivial queries. And conversely, a heuristic that only considers the individual best plan for each query does not work well, as mentioned earlier.



Traditional execution: No sharing
total cost = 460

Execution with BC shared
total cost = 370

Multi-Query Optimization. Figure 1. Example illustrating benefits of sharing computation.

A more practical approach was presented in [10]. This approach efficiently finds the set of potentially common subexpressions for a set of queries, and then identifies the subset of CSEs to share, and the best resulting consolidated plan, using an iterative greedy heuristic.

Instead of enumerating the search space of possible plan combinations, the idea is to store all the plans across all the queries in a single compact data structure called the Logical Query DAG (LQDAG). The LQDAG is a refinement of the "memo" data structure used in transformational top-down optimizers, such as Volcano [4], to memorize the best plans of the intermediate results. (Such memorization, as done in top-down query optimizers such as Volcano, is equivalent to dynamic programming, as used in System R and other bottom-up query optimizers.)

Figure 2a shows a LQDAG for the query A ⋈ B ⋈ C; this LQDAG represents the three alternative plans for the query: (A ⋈ B) ⋈ C, A ⋈ (B ⋈ C) and B ⋈ (A ⋈ C). Each square node (equivalence node) in the LQDAG represents a distinct intermediate result, and each round node (operation node) below represents a distinct plan to compute the same from the underlying intermediate results. In general, a LQDAG can represent multiple queries in a consolidated manner, with a distinct root node for each distinct query. Figure 2b shows a consolidated LQDAG for the two example queries seen earlier, A ⋈ B ⋈ C, and B ⋈ C ⋈ D.

The CSEs for the given queries correspond to equivalence nodes in the LQDAG that are shared either within the same plan, or between plans for two distinct queries; [10] presents an efficient algorithm that identifies the set of all CSEs in a single bottom-up traversal of the LQDAG.

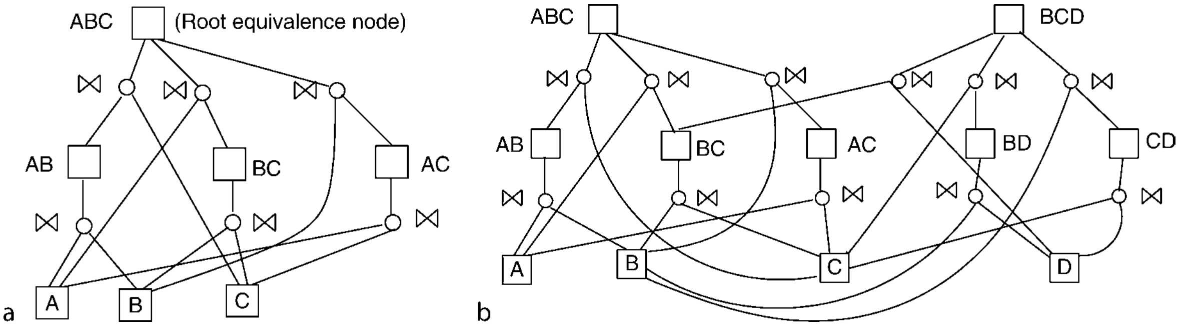
After the CSEs are identified, the next task is to find the best consolidated plan for the queries exploiting these CSEs. When the number of CSEs is large, an exhaustive search is not feasible; a natural approach is then to use a greedy heuristic that iteratively picks the CSE with the greatest benefit (i.e., whose use would result in the greatest decrease in the overall evaluation cost), terminating when no further decrease is possible. This algorithm requires that the benefit of each candidate CSE be recomputed in each iteration – this involves finding the best plan that uses the candidate CSE, in addition to the CSEs selected in earlier iterations.

With multiple such optimization calls in each iteration, a naive implementation of the greedy heuristic would be too expensive to be practical. [10] shows how to make this approach practical by (i) incorporating additional heuristics to significantly reduce the number of benefit computations, and (ii) showing how to efficiently perform a benefit computation by exploiting the LQDAG representation of the plan space. Additional insights on the task of seamlessly incorporating multi-query optimization into the Microsoft SQL-Server query optimizer are presented by Zhou et al. [14].

The above approach assumes that CSEs are materialized and read back from disk when required. Dalvi et al. [1] shows how to schedule queries such that results can be pipelined to multiple uses, even with a limited buffer space, thereby minimizing IO. Diwan et al. [2] addresses the issue of caching results in limited memory, and scheduling queries to minimize cache usage.

Key Applications

The idea of sharing computation among different queries to save on time and resources is ubiquitous.



Multi-Query Optimization. Figure 2. (a) LQDAG for A ⋈ B ⋈ C, and (b) Combined LQDAG for A ⋈ B ⋈ C and B ⋈ C ⋈ D.

As queries become increasingly expensive, the need for multi-query optimization to enable such savings is likely to increase as well. A few representative applications which motivate multi-query optimization are listed below.

- *On-Line Analytic Processing (OLAP) and Reporting*: A typical OLAP and reporting workload consists of queries with a significant amount of overlap. This overlap can occur for several reasons. For instance, queries might overlap in the kind of analysis they perform, or in the subset of data they are interested in; different queries could compute different aggregates over a join of the same set of tables. Alternatively, the queries could be against a virtual view; these queries clearly overlap at least in the computation of the result of the virtual view. Or else, the queries could involve common table expressions (specified using the WITH clause) that could be used in multiple places in the query; parts or whole of such common table expressions could be transiently materialized and reused [14]. Finally, the queries could involve correlated nested subqueries – invariant parts of these nested subqueries could be computed once and shared across invocations of the subquery [8].
- *Materialized View Maintenance*: Materialized views are supported by most major database systems today. Such materialized views must be updated when the underlying relations are updated. The maintenance plans for different views often share common computation. Mistry et al. [6] show how to exploit multi-query optimization to create an optimal view maintenance plan.
- *XML Query Processing*: In systems that store XML data in relational databases, the XPATH queries containing regular path expressions translate into a sequence of queries with significant overlap. Such queries are likely to benefit significantly from multi-query optimization [3].
- *Stream Query Processing*: Monitoring applications such as financial analysis and network intrusion detection often have to process multiple queries over a common stream of data. Such queries are likely to overlap significantly in the expressions they compute, and are likely to gain from multi-query optimization [5].

Cross-references

▶ Cost-based Query Optimization
▶ Query Optimization
▶ Transformational Query Optimization

Recommended Reading

1. Dalvi N.N., Sanghai S.K., Roy P., and Sudarshan S. Pipelining in multi-query optimization. J. Comput. Syst. Sci., 66(4):728–762, 2003.
2. Diwan A.A., Sudarshan S., and Thomas D. Scheduling and Caching in Multi-Query Optimization. In Proc. 13th Int. Conf. Management of Data, 2006.
3. Fan W., Yu J.X., Lu H., Lu J., and Rastogi R. Query translation from XPATH to SQL in the presence of recursive DTDs. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 337–348.
4. Graefe G. and McKenna W.J. The Volcano Optimizer Generator: Extensibility and Efficient Search. In Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 209–218.
5. Krishnamurthy S., Wu C., and Franklin M. On-the-fly sharing for streamed aggregation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 623–634.
6. Mistry H., Roy P., Sudarshan S., and Ramamritham K. Materialized view selection and maintenance using multi-query optimization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 307–318.
7. Park J. and Segev A. Using common subexpressions to optimize multiple queries. In Proc. 4th Int. Conf. on Data Engineering, 1988, pp. 311–319.
8. Rao J. and Ross K.A. Reusing invariants: a new strategy for correlated queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 37–48.
9. Rosenthal A. and Chakravarthy U.S. Anatomy of a modular multiple query optimizer. In Proc. 14th Int. Conf. on Very Large Data Bases, 1988, pp. 230–239.
10. Roy P., Seshadri S., Sudarshan S., and Bhobe S. Efficient and extensible algorithms for multi query optimization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 249–260.
11. Sellis T.K. Multiple query optimization. ACM Trans. Database Syst., 13(1):23–52, 1988.
12. Shim K., Sellis T., and Nau D. Improvements on a heuristic algorithm for multiple-query optimization. Data Knowl. Eng., 12:197–222, 1994.
13. Subramanian S.N. and Venkataraman S. Cost-based optimization of decision support queries using transient views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 319–330.
14. Zhou J., Larson P.Å., Freytag J.C., and Lehner W. Efficient exploitation of similar subexpressions for query processing. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 533–544.

Multi-Resolution

▶ Multiple Representation Modeling

Multi-Resolution Terrain Modeling

Enrico Puppo
University of Genova, Genova, Italy

Synonyms

Level-of-detail (LOD) terrain modeling

Definition

Multi-resolution terrain models provide the capability of using different representations of terrain at different levels of accuracy and complexity, depending on specific application needs. The major motivation behind multi-resolution is improving performance in geometry processing and visualization. Given a terrain database, a multi-resolution model provides the mechanisms to answer queries that combine both spatial and resolution criteria. In the simplest case, one could ask for a representation of terrain on a given area and with a given accuracy in elevation. More sophisticated multi-resolution models support adaptive queries, also known as *selective refinement* queries, where resolution may vary smoothly on the extracted representation, according to some given criterion. For instance, one could ask for an accuracy of at least 10 m on a given range of elevations, and smoothly degrading to say 100 m out of that range; similarly, high resolution could be focused in the proximity of a lineal feature (e.g., a road, a river); in view-dependent visualization, it is useful to have maximal resolution close to the viewpoint, and degrade it smoothly according to distance from it; etc. Multi-resolution engines must be able to answer such queries in real time even on planetary size databases. For instance, in view-dependent visualization it may be necessary to change representation, hence answering a query, at each frame, i.e., 25–30 times per second.

Historical Background

The concept of multi-resolution has been known since the mid 1970s, with seminal work by J. Clark [3]. Since then, many different proposals appeared in the literature, both in the context of terrain modeling and, more generally, in CAD and computer graphics (see [13]).

The design of multi-resolution terrain models is inter-related with *terrain generalization*, i.e., the problem of taking a representation of a terrain and generating another, smaller representation of the same terrain at a lower accuracy. The ideal aim of terrain generalization is to achieve an optimal ratio between accuracy and size of representation. This problem has been shown to be NP-hard by Agarwal and Suri [1]. However, starting with seminal work by Fowler and Little in the late 1970s [8], many algorithms for terrain generalization have been proposed in the literature that achieve good results in practice.

Early multi-resolution models belonged to two general classes: discrete models and tree-like models. In a discrete model, a collection of alternative representations of the same terrain at different resolutions is stored. Tree-like models follow a hierarchical approach: a base model provides a coarse representation of terrain made of a small number of atomic cells; each such cell is refined by decomposition into smaller cells at the next level of resolution; refinement is repeated over several levels and the model in maintained in a tree-like data structure. A notable example is given by *restricted quadtrees*, introduced first by Von Herzen and Barr in 1987 [15] and widely developed later by several authors. This class of models is suited to efficiently manage data with a regular distribution.

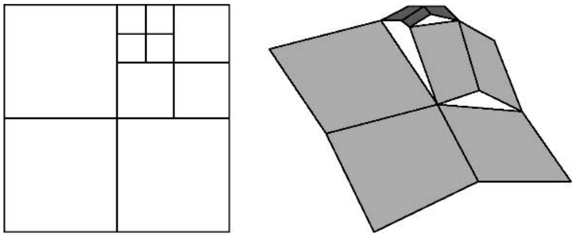
Many models developed (starting in the mid 1990s) are based on Triangulated Irregular Networks (TINs). Such models support also the manipulation of irregularly distributed data and may all be seen as instances of a general framework introduced by Puppo in 1996 [14]. The basic elements of such a framework are local modification operations, which change the resolution of a representation locally, and the hierarchical organization of such modifications on a directed acyclic graph. Such models come in many variants, they all support selective refinement and may achieve the best ratios between level of accuracy and number of triangles used in a representation.

Foundations

The simplest way to perform terrain generalization and implicitly obtain a multi-resolution model from a database of regularly distributed data is based on sub-sampling. Given a grid of elevation data at high resolution, a coarser sub-grid is obtained by regularly sampling data along each axis with a fixed step. In this case, the term *resolution* is referred to the size of cells in the resulting grid, or, in other terms, to the size of the step used to subsample. This method maintains the regular structure of data, but it provides no control on the loss of accuracy and it is not adaptive. Thus, a large number of samples may be used even to represent terrain in flat areas, while vertical error could easily exceed the allowed tolerance on areas that contain sudden variations of altitude.

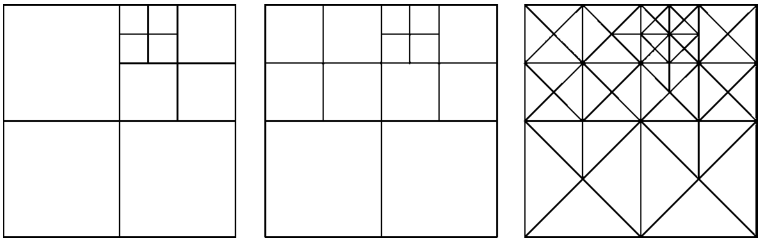
Better results can be obtained by building generalized representations in the form of TINs. Generalization algorithms, run with different thresholds on the same dataset at high resolution, provide a discrete model consisting of a collection of TIN representations at different accuracies. Being adaptive, such representations may contain small triangles in areas where terrain has large variations, and large triangles in relatively flat areas. In this case, the term *resolution* is more related to accuracy than to the size of atomic elements in the representation.

Discrete models support simple queries to extract a representation at fixed resolution. It is sufficient to select the layer corresponding to the desired resolution, and the region of interest within that layer. However, discrete models have several drawbacks: they usually provide only a small number of levels of detail; they cannot relate different representations of the same area at different resolutions; and they cannot combine data at different resolutions within a single representation. The *Delaunay pyramid* proposed by De Floriani in 1989 [5] is a TIN based discrete model in which vertical links between triangles that overlap at successive levels of detail are also maintained. In this sense, it comes midway between discrete and tree-like models. Compact data structures have also been proposed to maintain a Delaunay pyramid with many levels, and some authors have proposed variations of this model that can support selective refinement.

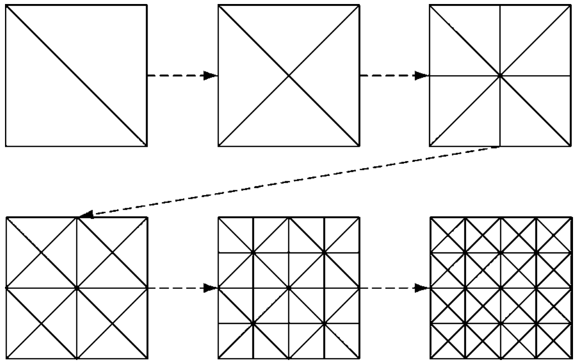


Multi-Resolution Terrain Modeling. Figure 1. A quadtree adaptive subdivision. The corresponding terrain surface has cracks.

In tree-like models, each node represents exactly the same portion of terrain covered by its children. Having a straightforward hierarchical structure, these models can also act as spatial indexes. Most successful models have been developed for regularly distributed data (for instance, the multi-resolution model adopted in *Google Earth* falls in this category). The simplest example consists of a quadtree structure built over a regular grid of data, which directly provides an adaptive version of the discrete model based on subsampling (see Fig. 1). Unfortunately, representations of adjacent portions of terrain from different levels of the quadtree cannot be combined seamlessly, as cracks would appear on the transition between quadrants from different levels (the resulting representation is said to be *non-conforming*). Restricted quadtrees solve this problem by triangulating the quadrants in a quadtree according to some predefined patterns that eliminate cracks, thus obtaining conforming representations (see Fig. 2). In practice, restricted quadtrees may support selective refinement and generate adaptive TINs made of right triangles and having their vertices at a subset of the data in the high resolution grid. Similar results are obtained by a hierarchical decomposition pattern based on triangle bisection (see Fig. 3), proposed in the mid 1990s by Lindstrom et al. [11] and later adopted in many variants by many authors. A square universe S is initially covered by two isosceles right triangles. The bisection rule subdivides a triangle into two similar triangles by splitting it at the midpoint v of its longest edge. A binary tree



Multi-Resolution Terrain Modeling. Figure 2. In a restricted quadtree, cracks can be eliminated by balancing the level of adjacent quadrants and triangulating quadrants with suitable patterns.



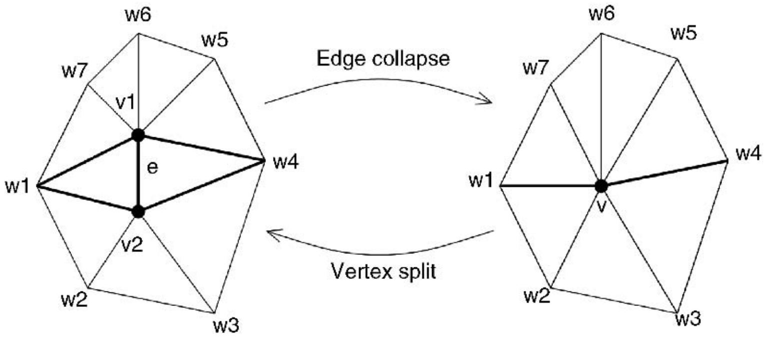
Multi-Resolution Terrain Modeling. Figure 3. Recursive triangle bisection generates a regular hierarchy based on the same triangles that appear in the restricted quadtree, but it exhibits a better flexibility.

of right triangles is thus obtained. In order to extract conforming meshes, such tree must be traversed in a proper way. In practice, adjacent triangles that are split by introducing a given vertex v will have to be split together during selective refinement. This scheme can be easily generalized to a spherical domain starting, e.g., from an octahedron. Efficient algorithms and data structures have been developed for this scheme, which can support selective refinement efficiently even on planetary size databases. A great advantage of restricted quadtrees and triangle bisection schemes comes from the regular distribution of data. Very compact implicit data structures can be designed, which have a small overhead with respect to maintaining just the single resolution data at the highest available detail, and are also suitable for implementation in secondary memory. Most efficient data structures, though, assume that the collection of all data in the database forms a unique regular grid at a given (high) resolution. In most real cases, however, the database is rather a patchwork of partially overlapping grids at different resolutions. In 2003, Gerstner proposed a data structure for the scheme based on triangle bisection, which works also in the latter case [9].

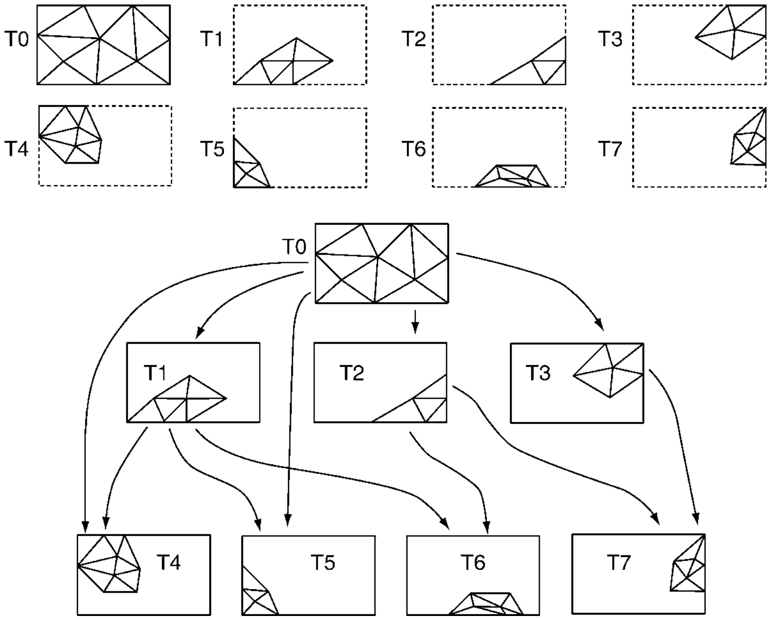
Cignoni et al. in 2003 proposed a model oriented to terrain rendering, the *BDAM*, which combines triangle bisection with adaptive schemes based on TINs [2]. The triangle bisection scheme is used as a spatial index, to obtain a coarse decomposition of the domain. Standard algorithms traverse such an index during selective refinement, and triangular blocks are collected from the proper levels of the tree. A TIN consisting of possibly a large number of triangles is associated to each triangular block in the spatial index. In this way, the representation resulting from selective refinement is in fact given by the collection of TINs corresponding to triangular blocks extracted during traversal of the index. TINs can be maintained on efficient data structures, suitable to be used in combination with Graphics Processing Units (GPUs), which may greatly improve performance. With this mechanism, performances in terrain visualization are excellent even on huge (planetary size) databases.

Other tree-like models have been developed based on TINs, which can work on arbitrary datasets. A triangle in a TIN may be refined by inserting a variable number of points either inside it or on its edges on the basis of an error-driven refinement criterion. Edges that survive across different levels of the hierarchy permit to combine surface patches from different levels of the tree, thus supporting selective refinement. Compared to models for regularly distributed data, these latter models can achieve a better ratio between size and accuracy, because no constraint is imposed on the vertex distribution, but need more complicated and expensive data structures to be stored.

Many other models developed (starting in the mid 1990s) have been oriented to irregularly distributed data and are all based on TINs and local modifications. A *local modification* is an operation that substitutes a small (local) portion of a TIN with another representation, formed by a different number of triangles. Modifications can be described either explicitly, by enumerating triangles that are eliminated and triangles that replace them, or, more often, implicitly through some mesh editing operations. The most famous and widely used editing operation is *edge collapse* (see Fig. 4), which is at the basis of the *Progressive Meshes (PM)* introduced by Hoppe in 1996 [10], and of many other models proposed in the literature. Edge collapse consists of collapsing an edge e of a TIN to a point. As a consequence, the two triangles incident at e will collapse to edges and disappear, thus the number of triangles, edges and vertices in the TIN will decrease by two, three, and one unit, respectively. Iterative edge collapse provides a powerful method for terrain generalization. The resulting sequence of collapses, together with their inverse operations called *vertex splits*, and the base mesh obtained from generalization, constitute a PM. Similar models can be built by using local modifications different from edge collapse, provided that they can be reversed. This simple structure supports the efficient extraction of terrain representations at many different levels of detail, but do not directly support selective refinement. In 1996, Puppo took a more general approach in analyzing models based on local modifications [14]. He proved that the inherent dependency relation between local modifications in a sequence is in fact a partial order, which can be encoded in a directed acyclic graph having such modifications as nodes (see Fig. 5). By traversing such graph in a proper order, selective refinement can be performed efficiently. This general framework consisting of a partial order of local modifications is called a *Multi-Triangulation (MT)*. Among the many schemes that fit in the MT framework, the data structure proposed by El-Sana and Varshney in 1999 is excellent for compactness [7]. Their model is based on edge collapse and just a binary tree of the vertices introduced from collapse operations is maintained, which contains in fact just a subset of the links in the graph of



Multi-Resolution Terrain Modeling. Figure 4. *Edge collapse* is a local modification for iterative terrain generalization: edge *e* together with its endpoints v_1 and v_2 collapse to vertex *v*; triangles adjacent to *e* together with their other edges collapse to edges w_1v and w_4v, respectively. Edge collapse is inverted by a refinement operation called *vertex split.*



Multi-Resolution Terrain Modeling. Figure 5. A sequence of arbitrary local refinement modifications substitute portions of a mesh with other, more refined, groups of triangles. The corresponding Multi-Triangulation is described by a directed acyclic graph: a modification *M* depends on another modification *M′* if and only if *M* eliminates some triangle that was introduced by *M′*.

dependencies of the MT. A clever mechanism based on enumeration of nodes in the tree allows them to retrieve the correct dependencies among nodes and run selective refinement correctly and efficiently.

In the literature, also other kinds of multi-resolution models have been proposed, which follow a functional approach rather than a geometric one (see, e.g., [12]). The basic idea is that a function can be decomposed into a simpler part at low resolution, together with a collection of perturbations called wavelet coefficients which define its details at progressively finer levels of resolution. Wavelets have been widely used for multi-resolution representation and compression of signals and images, while their applications to terrain and surfaces is more recent. The discrete computation of wavelets requires a recursive subdivision of the domain into regular cells like equilateral triangles or squares. Therefore these methods are just suitable for regularly distributed data and resulting hierarchies correspond to either quaternary triangulations or quadtrees.

For a more detailed treatment of multi-resolution terrain modeling see, e.g., [13,4,6] and references therein.

Key Applications

Multi-resolution terrain modeling is essential to manage complexity in those applications that need to either analyze or visualize terrain data at different scales, such as planetary browsers, flight simulators, CAD tools for road design, and all intensive computational tasks related to terrain, such as drainage networks and visibility.

Cross-references

- ▶ Digital Elevation Models
- ▶ Discrete Wavelet Transform and Wavelet Synopses
- ▶ Geographic Information System
- ▶ Quadtrees (and Family)
- ▶ Simplicial Complex
- ▶ Triangulated Irregular Network

Recommended Reading

1. Agarwal P.K. and Suri S. Surface approximation and geometric partitions. In Proc. 5th Annual ACM -SIAM Symp. on Discrete Algorithms, 1994, pp. 24–33.
2. Cignoni P., Ganovelli F., Gobbetti E., Marton F., Ponchio F., and Scopigno R. Planet-sized batched dynamic adaptive meshes (P-BDAM). In Proc. IEEE Visualization, 2003, pp. 147–155.
3. Clark J.H. Hierarchical geometric models for visible surface algorithms. Commun. ACM, 19(10):547–554, 1976.
4. Danovaro E., De Floriani L., Magillo P., Puppo E., and Sobrero D. Level-of-detail for data analysis and exploration: A historical overview and some new perspectives. Comput. Graph., 30(3): 334–344, 2006.
5. De Floriani L. A pyramidal data structure for triangle-based surface description. IEEE Comp. Graph. Appl., 9(2):67–78, 1989.
6. De Floriani L., Magillo P., and Puppo E. Geometric structures and algorithms for geographical information systems. In Handbook of Computational Geometry, J.R. Sack and J. Urrita (eds.), Elsevier Science, Amsterdam, 1999.
7. El-Sana J. and Varshney A. Generalized view-dependent simplification. Comput. Graph. Forum, 18(3):C83–C94, 1999.
8. Fowler R.J. and Little J.J. Automatic extraction of irregular network digital terrain models. In Proc. 6th Annual Conf. Computer Graphics and Interactive Techniques, 1979, pp. 199–207.
9. Gerstner T. Multiresolution compression and visualization of global topographic data. Geoinformatica, 7(1):7–32, 2003.
10. Hoppe H. Progressive meshes. In Proc. 23rd Annual Conf. Computer Graphics and Interactive Techniques, 1996, pp. 99–108.
11. Lindstrom P., Koller D., Ribarsky W., Hodges L.F., Faust N., and Turner G.A. Real-time, continuous level of detail rendering of height fields. In Proc. 23rd Annual Conf. Computer Graphics and Interactive Techniques, 1996, pp. 109–118.
12. Lounsbery M., DeRose T.D., and Warren J. Multiresolution analysis for surfaces of arbitrary topological type. ACM Trans. Graph., 16(1):34–73, 1997.
13. Lübke D., Reddy M., Cohen J.D., Varshney A., Watson B., and Hübner R. Level Of Detail for 3D Graphics. Morgan Kaufmann, Los Altos, CA, 2002.
14. Puppo E. Variable resolution terrain surfaces. In Proc. 8th Canadian Conf. on Computational Geometry, 1996, pp. 202–210.
15. Von Herzen B. and Barr A.H. Accurate triangulations of deformed, intersecting surfaces. In Proc. 14th Annual Conf. Computer Graphics and Interactive Techniques, 1987, pp. 103–110.

Multi-scale

▶ Multiple Representation Modeling

Multiscale Views

▶ Distortion Techniques

Multiscale Interface

▶ Zooming Techniques

Multiset Semantics

▶ Bag Semantics

Multi-Step Query Processing

Peer Kröger, Matthias Renz
Ludwig Maximillian University of Munich, Munich, Germany

Synonyms

Filter/refinement query processing

Definition

A query on a database reports those objects which fulfill a given query predicate. A query processor has to evaluate the query predicate for each object in the database which is a candidate for the result set. Multi-step query processing (filter/refinement query processing) is a technique to speed up queries specifying query predicates that are complex and costly to evaluate. The idea is to save the costs of the evaluation of the complex query predicate by reducing the candidate set for which the query predicate has to be evaluated applying one or more filter steps. The aim of each filter step is to identify as many true hits (objects that truly fulfill the complex query predicate) and as many true drops (objects that truly do not fulfill the query predicate) as possible by applying a less costly query predicate. The remaining candidates that are not pruned as drops or reported as hits in one of the filter steps need to be tested in a refinement step where the exact (costly) query predicate is evaluated. Obviously, the less costly the filter predicates are and the smaller the number of candidates that need to be refined, the higher the performance gain of a multi-step query processing is over a single-step query processing. In addition, if any of the applied filter steps is able to report true hits, first results can be reported to the user significantly sooner by a multi-step query processor compared to a single-step query processor.

Historical Background

In many database applications the management of complex objects is required. For example, the parts of a geographical map such as streets, lakes, forests – or generally *regions* – are stored as polylines or polygons. Queries on these complex objects usually involve complex query predicates that are costly to evaluate. For example, in order to retrieve all regions of a map that intersect with a given query window it is required to test the intersection of the query window and the database polygons which is computationally rather expensive. In such situations, the evaluation of the query predicate (e.g., "intersects the query window") becomes the bottleneck of query processing. Index structures are designed for shrinking down the search space of tentative hits in order to scale well for very large databases. Principally, the aim of index structures is the same as that of the filter-steps in multi-step query processing. However, index structures are only applicable for the first filter step. The reason is that index structures are designed to organize the entire database and cannot be applied to a reduced set of candidates.

To cope with complex data objects and costly query predicates, the paradigm of multi-step query processing (filter/refinement query processing) has been defined originally for spatial queries such as point queries and region queries on databases of spatial objects [6,2]. This paradigm has been applied to similarity search in databases of complex objects performing general similarity queries such as distance range queries [1,3] and k-nearest neighbor (kNN) queries [4] using costly distance functions. The key idea is to apply one or more filter steps each using cheaper query predicates (e.g., cheaper distance functions), the so-called *filter predicates*, in order to identify as many objects as possible as true hits or true drops. For the remaining candidates, for which the query predicate cannot be decided using any of the filter steps, the exact (more costly) query predicate needs to be evaluated in a refinement step. To ensure correct results, the filter predicates are required to be based on *conservative approximations* of the exact objects. This ensures that if any object does not qualify for a filter predicate, it can also not qualify for the exact query predicate. For example, if the regions of a map are conservatively approximated by minimum bounding rectangles (MBRs) of the corresponding polygons, those regions whose corresponding MBRs do not intersect with the query window cannot intersect with the query window. This conservative property of the filter predicates enables discarding true drops. On the other hand, filter predicates that are based on progressive approximations of the exact objects can be used to identify true hits. For example, if the regions of a map are progressively approximated by an incircle of the corresponding polygons, those regions whose corresponding incircle intersect with the query window do also intersect with the query window.

Foundations

Multi-step query processing is usually used in applications where the objects in the database are complex and the queries launched on objects rely on costly predicates that cannot be evaluated efficiently. In such applications, the evaluation of the query predicate becomes the bottleneck in query execution.

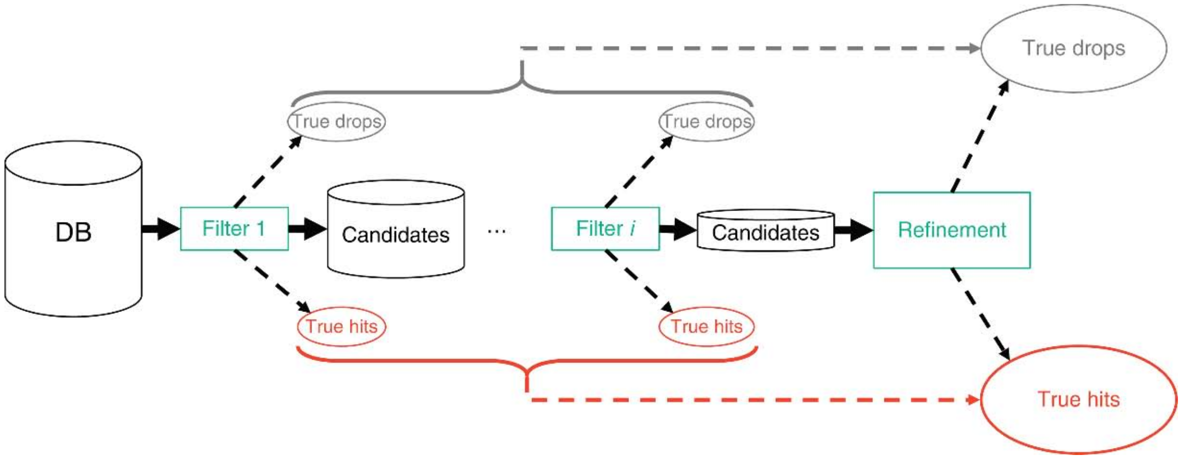
General Schema of Multi-Step Query Processing

Multi-step query processing is based on the following idea: design one or more filter predicates that can be evaluated much faster than the original query predicate and that can be used to shrink down the number of candidates for which it is unknown whether they qualify for the query predicate or not. The query processing starts with all database objects as candidates and applies the designed filters sequentially on the remaining candidates. Each filter ideally identifies true hits that can be added to the result set and true drops that can be pruned. The candidates that cannot be classified as true hits or true drops after all filter steps need to be refined by evaluating the (costly) original query predicate. This general schema is illustrated in Fig. 1. The order in which the single filter steps are applied usually depends on the cost of each filter step and on the *selectivity* of each filter step. The selectivity of a filter step determines the fraction of objects that are identified as true hit or true drop by the corresponding filter and do not need any further processing. In order to produce correct results, obviously, the filter steps must not produce false drops (i.e., drop objects that match the original query predicate according to a filter predicate) and false hits (i.e., report objects that do not match the original query predicate as hits according to a filter predicate).

In order to apply multi-step query processing, it is important to design appropriate filter predicates for the original query predicates of the given application. An appropriate filter predicate can usually be designed by designing a less complex representation that approximates the complex database objects. The evaluation of the query predicate on these less complex object approximations should be less costly than on the original object representation. In the following, special instances of multi-step query processing is discussed in more detail.

Example: Multi-Step Query Processing of Similarity Queries

Usually, similarity between objects is expressed by means of a pair-wise distance function *dist*. A high distance between two objects denotes low similarity of these objects whereas a low distance implies high similarity. For example, if the database objects are points (of any dimensionality), *dist* could be the Euclidean distance, i.e., the vicinity of the corresponding points in the Euclidean space. If the database objects are sequences, *dist* could be the Edit distance. If the database objects are spatial regions (e.g., of a map), *dist* could be the smallest Euclidean distance between the



Multi-Step Query Processing. Figure 1. General schema of multi-step query processing.

corresponding polygons. The two most important and general types of similarity queries are distance range (DR) queries and k-nearest neighbor (kNN) queries.

A distance range query is a general query type in non-standard database systems such as spatial DBS, temporal DBS, and multi-media DBS. Given a query object q, a distance function $dist(.,.)$, and a distance threshold ε, a distance range query returns all database objects o that have a distance less or equal than ε to q, i.e., $dist(q, o) \leq \varepsilon$. They can be efficiently supported using index structures or multi-step query processing.

According to the above definition the query predicate of DR queries is given as follows: all hits o must qualify the predicate $dist(q, o) \leq \varepsilon$, where q is the query object and ε is a distance threshold. The query predicate of kNN queries is quite similar to DR queries: all hits o must qualify the predicate $dist(q, o) \leq d(q, k)$, where q is the query object and $d(q, k)$ is the k-nearest neighbor distance. However, the big difference between DR queries and kNN queries is that the distance threshold ε is given in advance, whereas the value of $d(q, k)$ is usually not known at query time.

A filter predicate for identify true drops (conservative property) can be designed as follows. First, a less complex representations to conservatively approximate the exact objects should to be developed. Usually, this can only be implemented for spatial objects: the conservative approximation must completely contain the exact object, e.g., a minimum bounding box (MBR) is a conservative approximation of a polygon. A second step is essential: A (cheaper) distance function on the approximation must be designed that implements the *lower bounding property.* Let $dist(.,.)$ be the exact distance function on the exact database objects and $LB(.,.)$ the cheaper filter distance. The distance function $LB(.,.)$ lower bounds the exact distance $dist(.,.)$, if the following holds:

$$LB(x, y) \leq dist(x, y)$$

for all database objects x and y. Since the exact predicate of a similarity query usually determines the hits as those objects that have a distance less than a threshold ε to the query object q, all objects o with $\varepsilon < LB(q, o) \leq dist(q, o)$ can be excluded from the result set without further processing. In other words, the filter predicate is similar to the original query predicate, but uses LB instead of $dist$.

For example, if the database contains the regions of a map, and $dist(r_1, r_2)$ is the smallest Euclidean distance between the regions (polygons) r_1 and r_2, an appropriate filter can be designed as follows. The regions are approximated by MBRs and $LB(m_1, m_2)$ is defined as the smallest Euclidean distance between the MBRs m_1 and m_2) of r_1 and r_2, respectively. Obviously, evaluating LB on the MBRs is usually much less complex and costly than evaluating $dist$ on the polygons.

A filter predicate for identifying true hits can be designed analogously. First, a less complex representation to progressively approximate the exact objects should be developed. Again, this can usually be implemented only for spatial objects: the progressive approximation must be completely contained within the exact object, e.g., the maximal circle contained within a polygon (incircle) is a progressive approximation of that polygon. Again, a second step is essential: A (cheaper) distance function on the approximation must be designed that implements the upper bounding property. Again, let $dist(.,.)$ be the exact distance function on the exact database objects and $UB(.,.)$ the cheaper filter distance. The distance function $UB(.,.)$ upper bounds the exact distance $dist(.,.)$, if the following holds:

$$UB(x, y) \geq dist(x, y)$$

for all database objects x and y. All objects o with $\varepsilon > UB(q, o) \geq dist(q, o)$ can be added to the result set without further processing. In other words, the filter predicate is again similar to the original query predicate, but uses UB instead of $dist$.

Sometimes, an approximate representation of the database objects allows the definition of two distance functions, one lower bounding distance and one upper bounding distance. Filter predicates that do not use upper or lower distances cannot be applied to reduce the number of candidates.

Example: Algorithms for Multi-Step Query Processing of Similarity Queries

The algorithm for multi-step distance range queries is rather easy. Since the distance threshold ε is known in advance, in each filter step, true hits and/or true drops are identified as described above, depending on the property of the distance used in the filter predicate.

On the other hand, a multi-step solution for kNN queries is not trivial, because in order to determine the exact value of $d(q, k)$ that can be used to identify objects based on any filter predicates as true hits or true drops, at least k objects need to be refined.

Since the k nearest neighbors are not known in advance, the k objects that need to be refined to determine the exact value of $d(q, k)$ are not known. Obviously, this is a vicious circle.

The multi-step kNN query processing algorithm proposed in [4] tries to approximate $d(q, k)$ by refining any k objects and take the maximum value $d'(q, k)$ of these exact distances. then, a multi-step DR query with query object q and distance threshold $d'(q, k)$ is evaluated. The resulting (refined) objects are ranked in ascending exact distances to q and only the first k objects of this ranking are reported as final result.

In [7], the authors enhance this approach with an algorithm that minimizes the number of refinements. The basic assumption of this algorithm is that only a conservative filter is applied. In the case of only one filter step, the algorithm uses a ranking query in the filter step. Given a query object q and a distance function *dist*(.,.), a ranking query returns a sequence of the database objects in a database D sorted by ascending distances to q. A ranking query is a general query type in non-standard database systems such as spatial DBS, temporal DBS, and multi-media DBS and can be efficiently supported using index structures. In the context of multi-step query processing in the filter step a ranking query returns a ranking of the database objects sorted in ascending filter distances to the query object q. Initially, the first k objects of the ranking are refined and an approximation $d'(q, k)$ of the true value of $d(q, k)$ is determined from these refined distances as above. Then, in each iteration, the next object from the ranking is fetched as long as the filter distance of the next object in the ranking is greater than the current approximation $d'(q, k)$. As long as this is not the case, the currently fetched object is refined and $d'(q, k)$ is updated. This algorithmic schema can easily be extended to applying multiple filter steps. It can be shown that – if only a conservative filter is implemented – this algorithm is optimal with regard to the number of refinements.

Finally, the algorithm in [5] further enhances the preceding algorithms that take only a conservative filter into account, by additionally using a progressive filter. The algorithm is similar to that in [7] but determines $d'(q, k)$ from the progressive filter as long as this is possible rather than from exact distances. As a consequence, the proposed algorithm reduces the number of refinements significantly. It can be shown that – if both a conservative filter and a progressive filter are implemented – this algorithm is optimal with regard to the number of refinements.

Key Applications

More and more applications suffer from the increasing complexity of the objects and of the functions required to evaluate query predicates on such objects, e.g., complex distance functions or spatial intersections. In the meantime, the efficient support of multi-step query processing is essential for many application areas such as molecular biology, medical imaging, CAD systems, and multimedia databases.

In this context, one of the most important application where multi-step query processing is essential for efficient query processing is similarity search in time series databases. Time series may be very large. Typical similarity queries in time series databases are distance range queries and k-nearest neighbor queries. Due to the curse of dimensionality, similarity queries cannot efficiently be supported by indexing the time-series based on the raw data. A common method to overcome this problem is to reduce the dimensionality of the object descriptions and use this lower-dimensional feature space to index the time series. Similarity queries are then performed using the paradigm of multi-step query processing. In the filter step, approximated similarity distances are computed based on the dimensionality reduced representations, while the refinement step applies similarity distance functions based on the raw time series data. Usually, the filter step is conservative, i.e., the filter distances lower bound the exact distances.

Another important application which requires multi-step query processing is the support of proximity queries in spatial networks like road networks where point objects located within the road network that is represented by a graph are queried. Usually, the objects are positions of buildings or individuals like persons or cars that can have a static location or may move within the network. Example queries could be "retrieve all cars within the road network having a smaller distance to the fast-food restaurant Pinky than 5.0 km" or "give me the three filling stations having the smallest distance to my actual position." Since the motion of the objects is restricted by the network, i.e., objects can only move along a path in the network graph, the distance between two objects is not measured using the Euclidean distance. Rather, the length of the shortest path between two objects is used as distance measure. For each distance computation it is necessary to apply the Dijkstra algorithm which is too expensive to answer such proximity queries on large databases in real time. Therefore, distance approximations are needed, which can be computed more efficiently and can be used in the filter step of a multi-step query processing algorithm. The simplest road-network distance approximation that fulfills the lower bound criterion is the Euclidean distance. Another method to achieve suitable distance approximations is the pre-computation of distances based on certain landmarks (reference nodes). The distance approximation based on landmarks has the advantage that, in addition to the lower bounding distance approximation, it is possible to compute a distance approximation which fulfills the upper bounding property.

A further important application of multi-step query processing is the support of spatial queries in spatial databases, i.e., databases containing objects having a spatial extension. One of the most important query types in such databases is the point-in-polygon test. Given a database with two-dimensional polygon objects and a certain query point, retrieve all polygons that include the query point. Several filter steps can be applied for this problem to avoid unnecessary point-in-polygon-tests. For example, the polygons can be conservatively approximated by minimum bounding rectangles (MBRs). Obviously, MBRs that do not contain the query point can be discarded as true drops. On the other hand, progressive approximations of the polygons can be used to identify true hits.

Cross-references

- ▶ Closest-Pair Query
- ▶ High Dimensional Indexing
- ▶ Indexing Metric Spaces
- ▶ Nearest Neighbor Query
- ▶ Spatial Indexing Techniques
- ▶ Spatial Join
- ▶ Spatio-Temporal Data Mining

Recommended Reading

1. Agrawal R., Faloutsos C., and Swami A. Efficient similarity search in sequence databases. In Proc. 4th Int. Conf. on Foundations of Data Organization and Algorithms, 1993, pp. 69–80.
2. Brinkhoff T., Horn H., Kriegel H.-P., and Schneider R. A storage and access architecture for efficient query processing in spatial database systems. In Proc. 3rd Int. Symp. Advances in Spatial Databases, 1993, pp. 357–376.
3. Faloutsos C., Ranganathan M., and Manolopoulos Y. Fast subsequence matching in time series database. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp.419.
4. Korn F., Sidiropoulos N., Faloutsos C., Siegel E., and Protopapas Z. Fast nearest neighbor search in medical image databases. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 215–226.
5. Kriegel H.-P., Kröger P., Kunath P., and Renz M. Generalizing the optimality of multi-step k-nearest neighbor query processing. In Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases, 2007, pp. 75–9.
6. Orenstein J. and Manola F. Probe spatial data modelling and query processing in an image database application. IEEE Trans. Softw. Eng., 14(5), 1988.
7. Seidl T. and Kriegel H.-P. Optimal multi-step k-nearest neighbor search. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 154–16.

Multi-Tier Architecture

HEIKO SCHULDT
University of Basel, Basel, Switzerland

Synonyms

n-tier architecture; Multi-layered architecture

Definition

A *Multi-tier Architecture* is a software architecture in which different software components, organized in tiers (layers), provide dedicated functionality. The most common occurrence of a multi-tier architecture is a three-tier system consisting of a data management tier (mostly encompassing one or several database servers), an application tier (business logic) and a client tier (interface functionality). Novel deployments come with additional tiers. Web information systems, for instance, encompass a dedicated tier (web tier) between client and application layer.

Conceptually, a multi-tier architecture results from a repeated application of the client/server paradigm. A component in one of the middle tiers is client to the next lower tier and at the same time acts as server to the next higher tier.

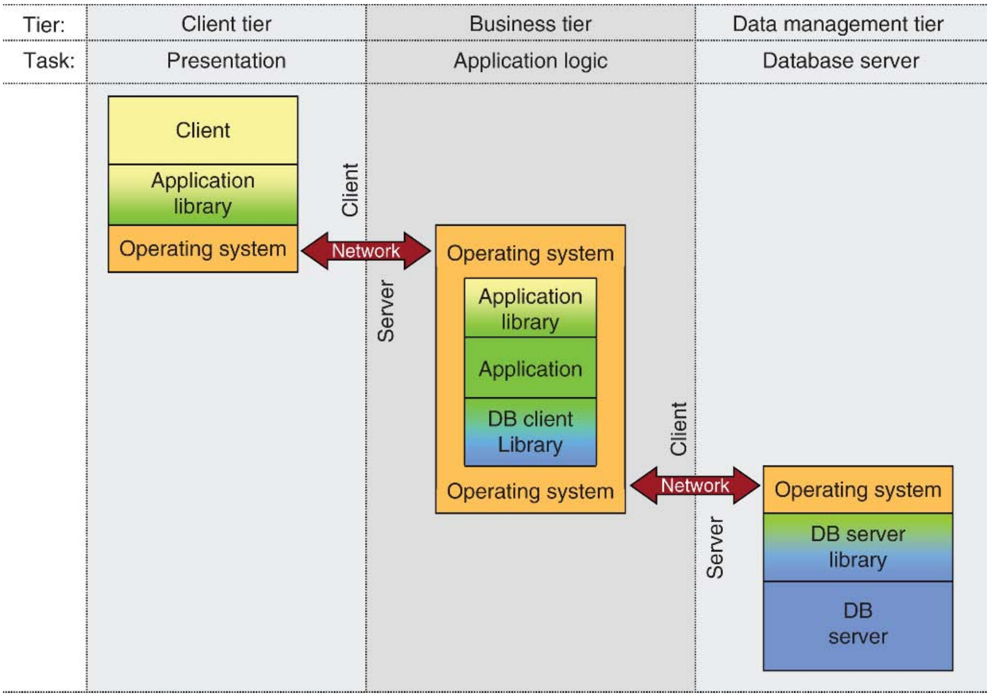
Historical Background

Early generation software systems have been built in a monolithic way. This means that all the different tasks for implementing a particular application and presenting the results to a user are provided by a single dedicated software component. With the advent of client/server architectures in the 1980s, different tasks could be separated and possibly even be distributed across network boundaries. In a client/server architecture (two tier architecture), the client is responsible for presenting the application to the user while the server is in charge of data management. For the provision of business logic, two alternatives have emerged. First, in so-called fat client/thin server architectures, the client also provides business logic, in addition to presentation and user interfaces. This can be realized by using SQL against the underlying database server in the application program run by the client, either by embedding SQL into a higher programming language or by using the database server's call level interface (e.g., JDBC, ODBC). Second, in thin client/fat server architectures, the database server also provides business logic while the client solely focuses on presentation issues. Fat servers can be realized by using persistent stored modules or stored procedures inside the database server. In the case of evolving business logic, fat client architectures, although being the most common variant of client/server systems, impose quite some challenges when new client releases need to be distributed in large deployments. In addition, a fat client architecture usually comes along with a high network load since data is completely processed at the client side. Fat servers, in contrast, impose a single point of failure and a potential performance bottleneck.

Three-tier architectures thus are the next step in the evolution of client/server architectures where both client and database server are freed from providing business logic. This task is taken over by an application layer (business tier) between client and database server. In multi-tier architectures, additional tiers are introduced, such as for instance a web tier between client and application layer.

Foundations

Multi-tier systems follow an architectural paradigm that is based on separation of concerns. The architecture considers a vertical decomposition of functionality into a stack of dedicated software layers. Between each pair of consecutive layers, a client/server style of interaction is applied, i.e., the lower layer acts as server for the next higher layer (see Fig. 1). Typical tiers in a three-tier architecture are *data management*, *business* and *client* tier. Multi-tier architectures consider additional layers, such as a web tier which hosts servlet containers and a web server and which is located between client tier and application tier.



Multi-Tier Architecture. Figure 1. Structure of a three tier architecture.

In addition to vertical decomposition and distribution across tiers, in many cases multi-tier architectures also leverage horizontal distribution within tiers. For the data management tier, this means that several distributed database servers can be used. Most commonly, horizontal distribution is applied at the business tier, i.e., providing several application server instances [7].

The main benefit of multi-tier applications is that each tier can be deployed on different heterogeneous and distributed platforms. Load balancing within tiers, especially for the application tier, is supported by distributing requests across the different application server instances. This can be implemented by a dispatcher which accepts calls from the next higher layer and distributes them accordingly (this is done, for instance, in TP Monitors which allow to distribute requests among application processes at the middle tier in a three-tier architecture).

When multi-tier architectures are used in a business context, they have to support transactional interactions. Due to the inherent distribution of software components across layers and potentially even within layers, distributed transactions are needed. This is usually implemented by a two-phase commit protocol (2PC) [5] (depending on the application server and the middleware used, this can be done, for instance, via CORBA OTS, the Java Transaction Service JTS, etc.). While 2PC provides support for atomicity in distributed transactions, it does not take into account the layered architecture where transactions at one layer are implemented by using services and operations of the next lower layer. Multi-level transactions [11] take this structure into account. SAP ERP [4], for instance, applies multi-level transactions by jointly considering the application server and data management tier. Asynchronous interactions between components in a multi-tier architecture require a message-oriented middleware (MOM). In this case, transactional semantics can be supported by persistent queues and queued transactions [1].

In order to increase the performance of multi-tier systems and to improve response times, caching is used at the application tier. For this, different database technologies such as replication, materialized views, etc. can be applied outside the DBMS [6].

Key Applications

Due to the proliferation of both commercial and open source application servers, multi-tier architectures can be found in a very large variety of different domains. Applications include, but are not limited to, distributed information systems, Web information systems, e-Commerce, etc.

Experimental Results

The Transaction Processing Performance Council (TPC) has defined a benchmark, *TPC-App*, for evaluating the business tier and in particular the performance of application servers in a three- or multi-tier architecture [10]. It includes Web Service interactions, distributed transactions, and asynchronous interactions via message-oriented middleware (reliable messaging and persistent queues).

Cross-references

▶ Application Server
▶ Client/Server Architecture
▶ Database Middleware
▶ Distributed Transaction Management
▶ Java EE
▶ Message Queuing Systems
▶ Middleware Support for Database Replication and Caching
▶ Multilevel Transactions and Object-Model Transactions
▶ Replication in Multi-Tier Architectures
▶ Service Oriented Architecture
▶ Transactional Middleware
▶ Web Services
▶ Web Transactions

Recommended Reading

1. Bernstein P. and Newcomer E. Principles of Transaction Processing. Morgan Kaufmann, Los Altos, CA, 1997.
2. Birman K. Reliable Distributed Systems: Technologies, Web Services, and Applications. Springer, Berlin, 2005.
3. Britton C. IT Architectures and Middleware. Addison Wesley, Reading, MA, USA, 2001.
4. Buck-Emden R. and Galimow J. SAP R/3 System: A Client/Server Technology. Addison-Wesley, Reading, MA, USA, 1996.
5. Lindsay B., Selinger P., Galtieri C., Gray J., Lorie R., Price T., Putzolu F., and Wade B. Notes on Distributed Databases. IBM Research Report RJ2571, San Jose, CA, USA, 1979.
6. Mohan C. Tutorial: Caching Technologies for Web Applications. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
7. Mohan C. Tutorial: Application Servers and Associated Technologies. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
8. Myerson J. The Complete Book of Middleware. Auerbach, Philadelphia, PA, 2002.

9. Orfali R., Harkey D., and Edwards J. Client/Server Survival Guide. Wiley, 3rd edn., 1999.
10. Transaction Processing Performance Council.TPC-App. http://www.tpc.org/tpc_app/default.asp, 2008.
11. Weikum G. and Schek H.J. Concepts and Applications of Multilevel Transactions and Open Nested Transactions. In Database Transaction Models for Advanced Applications, K. Elmagarmid (ed.), Morgan Kaufmann, Los Altos, CA, 1992, pp. 515–553.
12. Weikum G. and Vossen G. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control. Morgan Kaufmann, Los Altos, CA, 2001.

Multivalued Dependency

Solmaz Kolahi
University of British Columbia, Vancouver,
BC, Canada

Synonyms

MVD

Definition

A *multivalued dependency (MVD)* over a relation schema $R[U]$, is an expression of the form $X \twoheadrightarrow Y$, where X, $Y \subseteq U$. An instance I of $R[U]$ satisfies $X \twoheadrightarrow Y$, denoted by $I \models X \twoheadrightarrow Y$, if for every two tuples t_1,t_2 in I such that $t_1[X] = t_2[X]$, there is another tuple t_3 in I such that $t_3[X] = t_1[X] = t_2[X]$, $t_3[Y] = t_1[Y]$, and $t_3[Z] = t_2[Z]$, where $Z = U - XY$ (XY represents $X \cup Y$). In other words, for every value of X, the value of attributes in Y is independent of the value of attributes in Z. A multivalued dependency $X \twoheadrightarrow Y$ is a special case of a *join dependency* expressed as $\bowtie[XY, X(U - XY)]$, which specifies that the decomposition of any instance I satisfying $\bowtie [XY, X(U - XY)]$ into $\pi_{XY}(I)$ and $\pi_{X(U-XY)}(I)$ is lossless, i.e., $I = \pi_{XY}(I) \bowtie \pi_{X(U-XY)}(I)$.

Movies			
Title	Director	Actor	Year
Pulp Fiction	Quentin Tarantino	John Travolta	1994
Pulp Fiction	Quentin Tarantino	Samuel L. Jackson	1994
The Matrix	Andy Wachowski	Keanu Reeves	1999
The Matrix	Andy Wachowski	Laurence Fishburne	1999
The Matrix	Larry Wachowski	Keanu Reeves	1999
The Matrix	Larry Wachowski	Laurence Fishburne	1999

Key Points

Multivalued dependencies, like functional dependencies, can cause redundancy in relational databases. For instance, in the following table, each director of the movie *The Matrix* is recorded once per actor of the movie, and this is because the instance satisfies the MVD *title* $\twoheadrightarrow$ *director.*

Multivalued dependencies have been considered in the normalization techniques that try to improve the schema of a database by disallowing redundancies. The most common normal form that takes MVDs into account is the Fourth Normal Form (4NF). The implication problem for MVDs can be solved in polynomial time. That is, given a set Σ of MVDs, it is possible to check whether an MVD $X \twoheadrightarrow Y$ is logically implied by Σ, denoted by $\Sigma \models X \twoheadrightarrow Y$, in the time that is polynomial in the size of Σ and $X \twoheadrightarrow Y$. Multivalued dependencies are usually considered together with functional dependencies (FDs) in the normalization of relational data. There is a sound and complete set of rules (axioms) that can be used to infer new dependencies from a set of MVDs and FDs defined over a relation $R[U]$:

MVD0 (complementation): If $X \twoheadrightarrow Y$, then $X \twoheadrightarrow (U - X)$.
MVD1 (reflexivity): If $Y \subseteq X$, then $X \twoheadrightarrow Y$.
MVD2 (augmentation): If $X \twoheadrightarrow Y$, then $XZ \twoheadrightarrow YZ$.
MVD3 (transitivity): If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$, then $X \twoheadrightarrow (Z - Y)$.
FMVD1 (conversion): If $X \rightarrow Y$, then $X \twoheadrightarrow Y$.
FMVD2 (interaction): If $X \twoheadrightarrow Y$ and $XY \rightarrow Z$, then $X \rightarrow (Z - Y)$.

It is also known that the set {MVD0,...,MVD3} is an axiomatization for MVDs considered alone.

Cross-references

- ▶ Fourth Normal Form
- ▶ Functional Dependency
- ▶ Join
- ▶ Join Dependency
- ▶ Normal Forms and Normalization
- ▶ Projection

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases. Addison-Wesley, Reading, MA, USA, 1995.

Multivariate Data Visualization

▶ Dynamic Graphics

Multivariate Visualization Methods

Antony Unwin
Augsburg University, Augsburg, Germany

Synonyms

Graphical displays of many variables

Definition

Multivariate datasets contain much information. One- and two-dimensional displays can reveal some of this, but complex pieces of information need more sophisticated displays that visualize several dimensions of the data simultaneously. Usually several displays are needed.

Historical Background

Graphical displays have been used for presenting and analysing data for many years. Playfair [10] produced some fine work over 200 years ago. Minard prepared what Tufte has called "the finest graphic ever drawn" in the middle of the nineteenth century, showing Napoleon's advance on and retreat from Moscow, including information on the size of the army and the temperature at the time. Neugebaur introduced many innovative ideas in the 1920s and 1930s. Most of these graphics are primarily one- or two-dimensional. Techniques for displaying higher dimensional data have mainly been suggested more recently.

Foundations

There are two quite different aims of data display: analysis and presentation. Graphics aid analysts in understanding data and in determining structure. Graphics are good for identifying outliers, for picking out local patterns, and for recognizing global features. Graphics are also valuable for conveying that information to others. Wilkinson's book [13] defines a formal structure. Unwin et al. [12] discuss graphics for large datasets. Theus et al. [11] present interactive graphics for exploring data. The Handbook of Data Visualization [2] provides an overview of the current state of play.

For displaying multivariate data, indeed for displaying data in general, it is important to distinguish between different data types. Variables may be categorical, ordinal, continuous, temporal, spatial or logical (and other specialist types could be added as well). In data analysis the most common types are continuous and categorical. Ordinal may sometimes be treated as categorical (when there are only a few distinct values) and sometimes as continuous (when there are many).

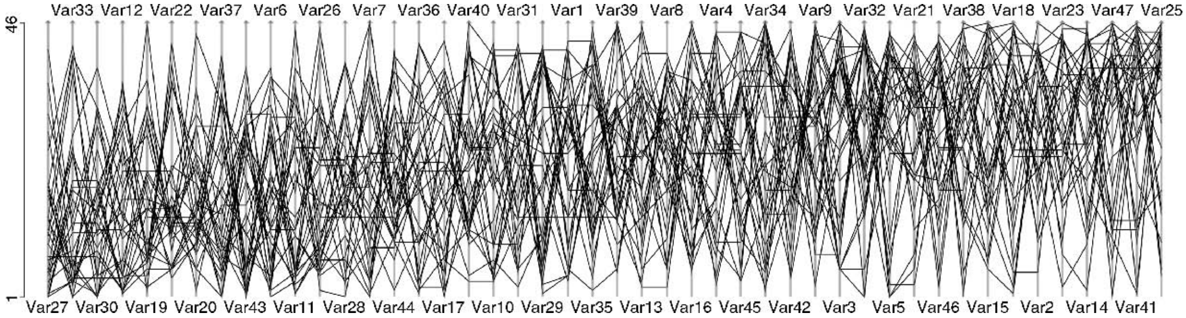
No printed graphic can display more than two dimensions fully at once. Multivariate graphics use projections, conditioning, and linking to capture higher dimensional information. Some displays can deal with very large numbers of cases (area displays such as mosaic plots), and some can potentially handle very many variables (parallel coordinate plots). Most displays are limited in both dimensions. One strategy is to use small multiples, multiple versions of the same graphic restricted to subsets of the data. Trellis plots [1] are the most important example of this approach. Whether many small displays are used or a range of large displays (which might be referred to as large multiples), more than one display will always be necessary to reveal the information in the data.

It is essential to bear in mind that to have enough evidence to confirm complex relationships lots of data are needed. Think of determining the effects of all the influences on car insurance premiums or of estimating the effects of factors in health studies (for instance breast cancer risk). Graphical methods must be able to deal with large datasets to be fully useful [12].

Multivariate Continuous Data

For multivariate continuous data the most popular graphic solution is parallel coordinate plots [8]. Further approaches include scatterplot matrices (sploms), showing all scatterplots of two variables at a time, and trellis plots, which display the data in subsets defined by conditioning variables. Glyphs, individual images for each case whose form depends on the separate variable values, can be an interesting possibility for smaller datasets (at most a few hundred cases). Matrix visualizations [2] are also interesting for smaller datasets and display each value by a color coding, with cases in the rows and variables in the columns. Including options for ordering both rows and columns is essential. Microarray data are often displayed in this way.

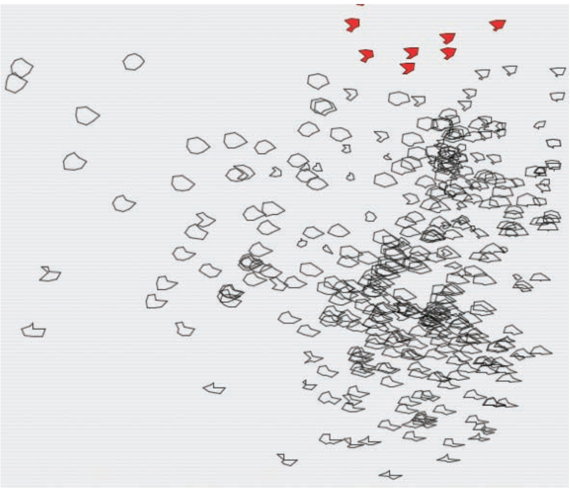
Other alternatives, known under the common heading of dimension reduction plots, display two-dimensional approximations of multivariate data, e.g., multi-dimensional scaling (MDS) and biplots. Dynamic methods include the grand tour and projection



Multivariate Visualization Methods. Figure 1. A parallel coordinate plot of the ratings of 46 wines by 32 judges. The axes are common scaled and have been ordered by mean values.

pursuit [3], both of which work by moving smoothly through two dimensional projections of the data.

Figure 1 shows a parallel coordinate plot of 32 judges' rankings of 46 American and French Cabernet wines from a 1999 tasting. The axes (one for each wine) have been given a common scale and sorted by their mean ratings from left to right, so that the highest ranked wine is on the far left and the lowest on the far right. What is striking is the lack of agreement amongst the judges. While there is a discernible trend, it is obscured by the high variability of the ratings. Most wines were ranked best by at least one judge and worst by at least one other. So the main message of this plot is that while the league table of results and statistical tests (whether the ordering was significantly non-random) imply a consensus ranking of the wines, the data convey otherwise. Parallel coordinate plots, like all high-dimensional plots, require fine-tuning to reveal information. In this case, common scaling and sorting were important tools. As parallel coordinate plots are covered in another entry, they are not discussed in detail here. One interesting new variant is represented by textile plots [9] in which the axes are rescaled to make the individual lines linking cases as horizontal as possible.



Multivariate Visualization Methods. Figure 2. An MDS display based on five variables for cars sold in Germany. Each car is represented by a circular-based glyph.

In MDS [4] the attempt is made to find a low-dimensional (almost always two dimensional) approximation to high dimensional data by positioning points so that the distances between them in the low dimensional display are close to the distances between them in the original dimension. For a high number of dimensions this is unlikely to be effective, but it often produces interesting views. The MDS display depends on the criterion used to match the distances (e.g., emphasising the absolute differences or the relative differences). Since all possible pairs must be considered, it is not efficient for large datasets. MDS displays are not unique for two reasons: an optimal solution in terms of the criterion will not necessarily be found; any solution is rotation invariant. Figure 2 shows an MDS plot of five-dimensional data on 381 cars sold in Germany. Each case is represented by a circular-based glyph using these five dimensions and two additional ones. The selected group at the top of the display seem relatively well separated in this view. They are all midsize luxury cars.

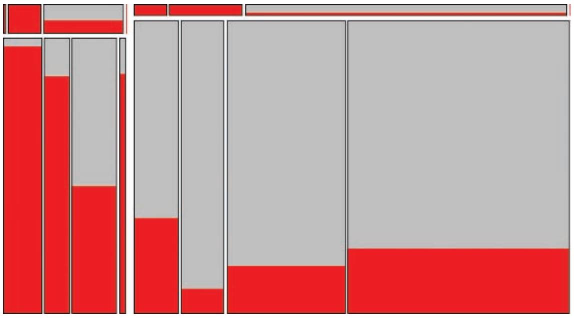
Biplots were developed by Gabriel [5]. He pointed out that both cases and variables could be plotted on the same approximating low dimensional plot. The two axes are usually chosen to be the first two principal components. Lines representing variables which are well approximated appear longer than those which represent variables badly approximated and the angles between the lines reflect the correlations between the variables in the low dimensional hyperplane. More complex biplots are also possible [6]. Like MDS displays, biplots will not work well in general, neither for many cases nor many variables, but often the two-dimensional projections produced can offer insightful views of the data.

Multivariate Categorical Data

Continuous data can always be sensibly binned and compressed, while retaining the option of zooming in to reveal the full level of detail. This does not hold for categorical data, where it may not be possible to combine any of the individual categories with others. While displays of single categorical variables are simple, the number of combinations rises exponentially with the number of variables. One binary variable can be displayed in a barchart of two columns. Twenty binary variables would give rise to 2^{20} combinations, a little over a million, many of which are likely to be empty, even for extremely large datasets.

Classic mosaicplots were suggested by Hartigan [7] for displaying a small number of categorical variables in a multivariate way. Other variations (multiple barcharts, fluctuation diagrams, equal binsize plots, doubledecker plots) [12] are often more useful. All depend very much on a careful choice of the ordering of variables and on an informative choice of size and aspect ratio. The ordering of variables determines which comparisons can be made, while the aspect ratio influences how well the comparison can be made.

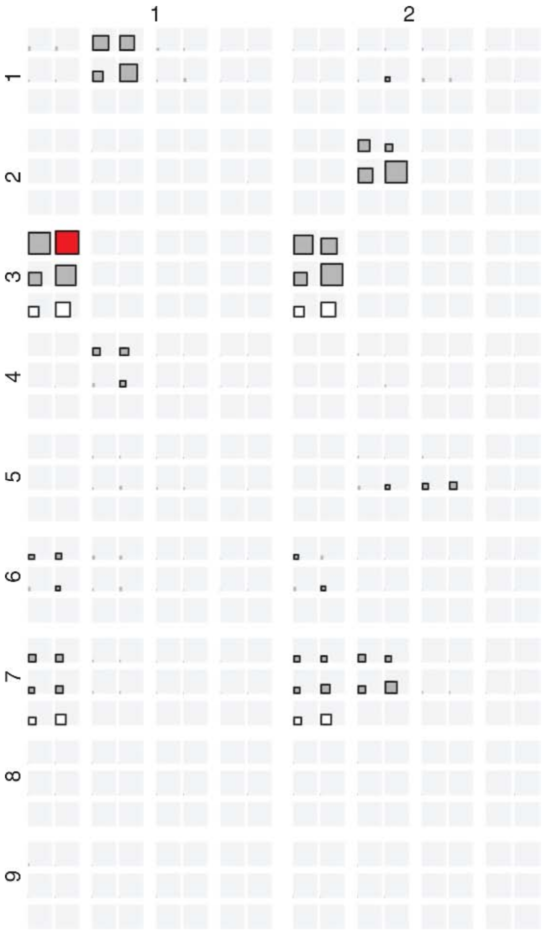
Figure 3 shows a mosaic plot of the Titanic data with the order of variables, gender, age, class. The block of four equally tall columns at the left of the display shows the numbers of adult women in each of the three passenger classes and the crew, with the proportion who survived highlighted. It is obvious that survival rates for adult women declined across the three passenger classes (the number of women in the crew was too small for any conclusion to be drawn). The next block of four columns relates to adult males and shows that the second class adult males had the lowest survival rate, a rather surprising result. The smaller bars at the top of the display refer to the children on board. The survival rates for males and females within classes can be compared approximately in this display, but clearly another display would be better for that, one using the variable ordering class, age, gender. Even in this dataset with only four variables, one plot is not enough.



Multivariate Visualization Methods. Figure 3. A mosaicplot of the numbers who sailed on the Titanic with the survivors selected. Women are to the left and men to the right, adults are below and children above. Within these groups the classes are ordered first, second, third, crew.

The main idea underlying all mosaicplots is that each combination of variable values is displayed by a rectangle whose area is proportional to the number of cases with that combination. The layout of the combinations is key in determining the interpretation, which can be difficult at the best of times, and is eased by providing interactive tools to query and adjust the graphic. Multiple barcharts are for comparing distributions of subsets (and are therefore related to trellis plots). Fluctuation diagrams are best for larger numbers of combinations to identify which are most common. Equal binsize plots and doubledecker plots are for comparing highlighted proportions.

Figure 4 shows a fluctuation diagram of a dataset from the Pakistan Labour Force Survey. Five variables are considered (the numbers of categories, including missings, are in brackets): gender [2], relation to head of household [9], marital status [4], literacy [3], and urban/rural [2], making 432 possible combinations in all. Although there are just under 140,000 cases, many of the combinations are empty or rare (e.g., few women and few single men are heads of households). The biggest single combination (male, son in household, never married, literate, living in a rural area) is highlighted and includes 9,678 cases or 7% of the dataset. Using interactive querying and animating the construction of the plot, one variable at a time, aids interpretation considerably. No display of several categorical variables



Multivariate Visualization Methods. Figure 4. A fluctuation diagram of five variables from the Pakistani Labour Force Survey: gender, relation to head of household, marital status, literacy, urban/rural. The biggest single combination is highlighted.

at once can either be easy to grasp immediately or convey all the potentially available information.

Interactive Graphics and Multivariate Graphics

Although both parallel coordinate plots and mosaicplots can be used for static plots, they are much more effective when used interactively. Their necessarily complex nature (after all, they have to display multivariate structure) demands careful scrutiny to grasp the information in them to the full, and the gain of understanding can be considerably enhanced when these graphics are empowered with interactive tools.

Interactive graphics may also be used to gain insight into multivariate datasets using one- or two-dimensional displays. Multiple linked simple displays of the same dataset can be easier to interpret than complex multivariate plots.

Key Applications

Descriptive statistics and Exploratory Data Analysis.

Future Directions

Many other more or less esoteric multivariate visualizations have been proposed. None should be dismissed out of hand, every visualization is probably ideal for some particular dataset. Nevertheless any succesful graphic should satisfy a number of criteria: it should be based on a readily recognizable and interpretable concept; it should be flexible and capable of being made interactive; it should be able to handle more than just three or four dimensions.

Displaying and interpreting even four-dimensional data is tricky. Dominating features can usually be seen, more subtle effects cannot. In higher dimensions the difficulties become much greater. At the moment it is impossible to visualize large numbers of categorical variables and although several hundred continuous variables can readily be displayed in parallel coordinate plots, the chances of identifying important features are slim. Nevertheless, graphics displays are useful for checking results found analytically and this can be very valuable. Visualizing multivariate data is new and progress is to be expected.

Visualization is currently mainly used for presentation of data, rather than for exploration of data. A single graphic can only display a limited number of aspects of a multivariate dataset and many are needed to convey all information available. The development of multivariate graphics should consider the design of sets of graphics rather than more elaborate versions of single ones. More interactive tools will have be developed. Sorting, rescaling, and querying are just some of the basics required.

Visualization is an important component of data analysis. It provides a complementary approach to analytic modeling and is much more suited to carrying out exploratory data analysis. Results found by models should be checked with graphics and ideas generated with graphics should be investigated analytically. The tighter integration of analytic and graphical methods would be of great advantage.

Cross-references

- ▶ Data Visualization
- ▶ Dynamic Graphics

- ▶ Parallel Coordinates
- ▶ Parallel Coordinates Plot (PCP)
- ▶ Visual Data Mining
- ▶ Visualizing Categorical Data
- ▶ Visualizing Quantitative Data

Recommended Reading

1. Becker R., Cleveland W., and Shyu M.J. The Visual Design and Control of Trellis Display. J. Computational and Graphical Statistics, 5:123–155, 1996.
2. Chen C.H., Haerdle W., and Unwin A. Handbook of Data Visualization. Springer, Berlin, 2007.
3. Cook D. and Swayne D. Interactive and Dynamic Graphics for Data Analysis. Springer, New York, 2007.
4. Cox M. and Cox M. Multidimensional Scaling. Chapman and Hall, London, 2001.
5. Gabriel K. The biplot - graphic display of matrices with application to principal component analysis. Biometrika, 58:453–467, 1971.
6. Gower J. and Hand D. Biplots. Chapman & Hall, London, 1996.
7. Hartigan J.A. and Kleiner B. Mosaics for Contingency Tables. In Proc. 13th Symposium on the Interface, 1981, pp. 268–273.
8. Inselberg A. Parallel Coordinates. Springer, New York, 2008.
9. Kumasaka N. and Shibata R. High Dimesional Data Visualisation: the Textile Plot. Computational Statistics and Data Analysis, 52(7):3616–3644, 2008.
10. Playfair W. Playfair's Commercial and Political Atlas and Statistical Breviary. Cambridge University Press, London, 2005.
11. Theus M. and Urbanek S. Interactive Graphics for Data Analysis. CRC Press, London, 2008.
12. Unwin A.R., Theus M., and Hofmann H. Graphics of Large Datasets. Springer, New York, 2006.
13. Wilkinson L. The Grammar of Graphics. Springer, New York, 2nd edn., 2005.

Multi-Version Concurrency Control

▶ Multi-version Serializability and Concurrency Control

Multi-Version Concurrency Control Algorithms

▶ Multi-version Serializability and Concurrency Control

Multi-Version Database

▶ Supporting Transaction Time Databases

Multi-Version Databases

▶ Multi-version Serializability and Concurrency Control

Multi-version Serializability and Concurrency Control

Wojciech Cellary
Poznan University of Economics, Poznan, Poland

Synonyms

Multi-version databases; Multi-version concurrency control; Multi-version concurrency control algorithms

Definition

Given a multi-version database, where each data item is a sequence of its versions. The number of versions of a data item may be limited or not. If it is unlimited, then each update of a data item over the limit gives rise to its next version. If it is limited, than each update of a data item replaces its oldest version. In case of limited number of versions, a database is called a K-version database. In multi-version databases any read operation of a data item, subsequent to a write operation of this data item, may access any of its currently existing versions. Thus, a multi-version schedule of a transaction set differs form the ordinary, mono-version schedule by a mapping of the data item read operations into the data item version read operations. Multi-version serializability plays the same role for the multi-version databases, as serializability for the ordinary, mono-version ones. Multi-version serializability is used to prove correctness of a concurrent execution of a set of transactions, whose read and write operations interleave, and moreover, read operations may access one of many available versions of a data item.

Historical Background

Multi-version serializability problem was a hot research topic in mid eighties. First works were published by P.A. Bernstein and N. Goodman [2,3] in 1983. Research was continued by G. Lausen [7], next by S. Muro, T. Kameda, and T. Minoura [8]. The next group of researchers involved was composed of C.H. Papadimitriou, P.C. Kanellakis, and T. Hadzilacos [6,9]. There is a lot of work devoted to different variants of multi-version concurrency control algorithms. A comprehensive background may be found in [5].

Foundations

Definition of a multiversion schedule. A *multiversion schedule mvs* of a set of transactions τ is a triple $mvs(\tau) = (T(\tau), h, <_{mvs})$, where (i) $T(\tau)$ is a the set of all database operations involved in the transactions of the set τ extended by the database operations of two hypothetical initial and final transactions and which respectively write the initial state of the database and read the final state of the database; (ii) h is a function which maps each read operation $r_{ij}(x) \in T(\tau)$ into a write operation $w_{kl}(x) \in T(\tau)$; and (iii) $<_{mvs} = \cup_i <_{T_i}$ is a partial order relation over $T(\tau)$ such that: if $T_{ij} <_{T_i} T_{ik}$ then $T_{ij} <_{mvs} T_{ik}$, and if $h(r_{ij}(x)) = w_{kl}(x)$ then $w_{kl}(x) <_{mvs} r_{ij}(x)$. Function h defined above maps a read operation of a data item into the write operation of a version of this data item – more precisely – into the write operation which creates the version of the data item read. Relation $<_{mvs}$ is defined by two conditions. The first one states that $<_{mvs}$ honors all orderings stipulated by transactions of the set τ. The second one states that a transaction cannot read a version of a data item until it has been created. A multi-version schedule is *serial* if no two transactions are executed concurrently, otherwise, it is *concurrent.*

Multiversion schedule equivalence. Two multiversion schedules are equivalent if they are *view* and *state equivalent.* Two multiversion schedules $mvs(\tau) = (T(\tau), h, <_{mvs})$ and $mvs'(\tau) = (T(\tau), h', <_{mvs'})$ of the set τ are *view equivalent* if and only if $h = h'$. If the transactions of two multi-version schedules $mvs(\tau)$ and $mvs'(\tau)$ receive an identical view of the database, i.e., if both multiversion schedules are view equivalent, then all the write operations issued by transactions in both schedules are the same. Two multiversion schedules $mvs(\tau) = (T(\tau), h, <_{mvs})$ and $mvs'(\tau) = (T(\tau), h', <_{mvs'})$ of the set of transactions τ are *final-state equivalent* if and only if for every initial state of the database and any computations performed by the transactions contained in τ the final states of the database reached as the result of schedules $mvs(\tau)$ and $mvs'(\tau)$ are identical.

Standard serial multiversion schedule. A serial multiversion schedule $mvs(\tau) = (T(\tau), h, <_{mvs})$ is *standard* if each read operation $r_{ij}(x) \in T(\tau)$ accesses the version of a data item x created by the last write operation w_{kl} (x) $\in$ $T(\tau)$ preceding r_{ij} (x). Since in a serial schedule, for every two transactions T_i and T_k, either all database operations of T_i precede all database operations of T_k or vice versa, the last write operation preceding a read operation is well defined. Note that a standard serial multi-version schedule in multi-version databases corresponds to a serial mono-version schedule in mono-version databases. From the consistency property of each transaction, i.e., from the assumption that each transaction separately preserves database consistency, it follows that a standard serial multi-version schedule must also preserve database consistency. On the basis of the above observation it is possible to define the multi-version serializability criterion [3].

Multi-version serializability criterion. A multi-version schedule $mvs(\tau)$ is *correct* if it is equivalent to any standard serial multi-version schedule of the set τ. Intuitively, the above criterion can be interpreted as follows. A concurrent schedule of a set of transactions in a multi-version database is correct if it is equivalent to a serial schedule of the transactions in which data replication over versions is transparent.

Key Applications

Multi-version serializability is used to prove correctness of concurrency control algorithms devoted to multiversion databases. As an example, consider a multi-version two-phase locking algorithm, called *WAB* [3,1,4], devoted to K-version databases. The concept of multi-version two-phase locking is broader than the concept of mono-version two-phase locking (cf. section on two-phase locking). An algorithm is a *multi-version two-phase locking algorithm* if it satisfies the following conditions:

1. There are two phases of transaction execution: the *locking phase* and the *unlocking phase.* During the locking phase a transaction must obtain all locks it requests. The moment when all locks are granted, which is equivalent to the end of the locking phase and the beginning of the unlocking phase, is called the *commit point* of a transaction. New versions of the data items prepared in the transaction's private workspace are written to the database during the unlocking phase.
2. The execution order of a set of transactions τ is determined by the order of transaction commit points.
3. The execution of any transaction $T \in (\tau)$ does not require locking data items that T does not access.

The concepts of locking and unlocking phases do not have exactly the same meaning as the similar notions used in the mono-version two-phase locking algorithm. In the *WAB* algorithm, the process of setting the so called "certify lock" is two-phase, but not as in the two-phase locking algorithm, the process of accessing data. In the *WAB* algorithm, each transaction initiated in the database and each version of a data item is *certified* or *uncertified*. When a transaction begins, it is uncertified. Similarly, each new version of a data item prepared in the transaction's workspace is uncertified. A *certify operation* is introduced, denoted by $c(w_{ij}(x))$, where $w_{ij}\ (x)$ is a T_i's write operation, and a new lock mode – the *certify lock* denoted by $CL(x)$. Certify locks are mutually incompatible. The algorithm requires that all certify and read operations of a data item x be $<_{mvs}$ related. Similarly, all certify operations must be $<_{mvs}$ related. The execution order of the certify operations determines a precedence relation $\ll_w$ defined on the set τ. The precedence relation $\ll_w$ specifies the order of transaction executions as follows: $T_i \ll_w T_k$ if and only if there exist such certify operations $c(w_{ij}(x))$ and $c(w_{kl}(x))$ that $c(w_{ij}(x)) \ll_{mvs} c(w_{kl}(x))$.

According to the *WAB* algorithm, any read operation $T_{ij}\ (x)$ concerns the last certified version of a data item x or any uncertified version of this data item. The version selected depends on a particular implementation of the *WAB* algorithm. Any write operation $w_{ij}\ (x)$ prepares a new version of a data item x in the workspace of transaction T_i (the version prepared is uncertified). At the end of transaction execution, the transaction and the new versions of the data items it prepared are being certified. The T_i's certification is a two-phase locking procedure. It consists of certify-locking all data items that the transaction T_i accessed to write. The T_i's certification is completed, when all certify locks are set and the following conditions are satisfied:

1. at the moment of T_i's certification, the versions of all data items read by T_i are certified;
2. for each data item x that T_i wrote, all transactions that read certified versions of x are certified.

To satisfy condition (ii), a *certify token* is allocated to each data item x to forbid reading certified versions of x other than the last one. On the other hand, all uncertified versions of x are allowed to be read. When the transaction T_i's certification is completed (the commit point), the procedure for certifying the versions of data items prepared by T_i is initiated. It was proved in [3] that the *WAB* algorithm is correct in the sense that any schedule produced by it is multi-version serializable. The main drawback of this algorithm is a possibility of a deadlock.

Future Directions

In database systems, multiple versions of data items are necessary to ensure transaction atomicity and to recover form crashes. The original idea of multiversion concurrency control based on multiversion serializability was to use those versions also to increase the degree of transaction concurrency, and as a result to improve database performance. However, such double use of versions decreases database reliability, because of the complexity of multiversion concurrency control. For practice, reliability of databases is of ultimate importance. This is why the concept of multiversion concurrency control was not well accepted in practice, except some implementations of two-version concurrency control concerning two values of each data item: before and after write operations. The concept of multiversion concurrency control may find attention in database systems applied in areas where ACID properties may be relaxed.

Cross-references

- ▶ Atomicity
- ▶ Concurrency Control – Traditional Approaches
- ▶ Replicated Database Concurrency Control
- ▶ Serializability
- ▶ Transaction
- ▶ Transaction Management
- ▶ Transaction Models–the Read/Write Approach

Recommended Reading

1. Bernstein P.A. and Goodman N. A sophisticate's introduction to distributed database concurrency control. In Proc. 8th Int. Conf. on Very Data Bases, 1982, pp. 62–76.
2. Bernstein P.A. and Goodman N. Concurrency Control and Recovery for Replicated Distributed Databases. Tech. Rep. TR-20/83, Harvard University, 1983.
3. Bernstein P.A. and Goodman N. Multiversion concurrency control – theory and algorithms. ACM Trans. Database Syst., 8(4):465–483, 1983.
4. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems.Addison-Wesley, Reading, MA, 1987.
5. Cellary W., Gelenbe E., and Morzy T. Concurrency Control in Distributed Database Systems, Elsevier Science, North-Holland, 1988.

6. Hadzilacos T. and Papadimitriou C.H. Algorithmic aspects of multiversion concurrency control. J. Comput. Syst. Sci., 33(2):297–310, 1986.
7. Lausen G. Formal aspects of optimistic concurrency control in a multiple version database system. Inf. Syst., 8(4):291–301, 1983.
8. Muro S., Kameda T., and Minoura T. Multi-version concurrency control scheme for database system. J. Comput. Syst. Sci., 29:207–224, 1984.
9. Papadimitriou C.H. and Kanellakis P.C. On concurrency control by multiple versions. ACM Trans. Database Syst., 9(1):89–99, 1984.

Music Metadata

► Audio Metadata

Music Retrieval

► Query by Humming

MVD

► Multivalued Dependency

N

Naive Tables

Gösta Grahne
Concordia University, Montreal, QC, Canada

Synonyms

Relations with marked nulls; Extended relations

Definition

The simplest way to incorporate unknown values into the relational model, is to allow *variables*, in addition to *constants*, as entries in the columns of relations. Such constructs are called *tables*, instead of *relations*. A table is an incomplete database, and *represents a set of complete databases*, each obtained by substituting all variables with constants. Different occurrences of the same variable (marked null) are substituted with the same constant. The substitution is thus a function from the variables and constants, to the constants, such that the function is identity on the constants. A table T then represents the set of relations, denoted $rep(T)$, defined as $\{v(T) : v$ is a valuation$\}$. Then the *certain answer* to a query q on a table T, denoted $sure(q, T)$ is the set of tuples that occur in every answer obtained by applying the query to every database in $rep(T)$. In other words, the certain answer to q on T is $sure(q, T) = \cap q(rep(T))$.

Key Points

To illustrate the above concepts, let tables T_1 and T_2 be as below, and let q be the relational expression $\sigma_{A=a \vee A=c}(\pi_{AC}(R_1 \bowtie R_2))$. (The schema of T_i is that of R_i, $i = 1,2$). Then applying q to T_1,T_2, which is denoted $\sigma_{A=a \vee A=c}(\pi_{AC}(T_1 \bowtie T_2))$, yields table $q(T_1, T_2)$ below.

T_1

A	B
a	*X*
Y	*b*
c	*b*

T_2

B	C
X	*d*
b	*Z*

$q(T_1, T_2)$

A	C
a	*d*
c	*Z*

The variables/null-values are written in uppercase, to clearly distinguish them from the (lowercase) constants. Note, however, that $q(T_1, T_2)$ is not (necessarily) yet the certain answer. How was $q(T_1, T_2)$ derived from q and T_1, T_2, and how is the certain answer $sure(q,(T_1, T_2))$ obtained from $q(T_1, T_2)$? The answer to the second question is very simple: just drop all tuples containing variables from $q(T_1, T_2)$. The remaining tuples form the certain answer. In the example, the certain answer consists of tuple (a, c) only. The answer to the first question is not much more complicated: evaluate q on the tables, treating variables as "constants," pairwise distinct, and distinct from all "real" constants. This is also known as the Naive evaluation of q on T [2,3]. In the example above, tuple (a, X) is joined with tuple (X, c) since they have the same value, represented by X, in the join column. This is done even though the "actual" value of X is not known, since in any valuation v the two occurrences of X are mapped to the same value. On the other hand, when performing the selection $\sigma_{A=a \vee A=c}$ the tuple (Y, Z) is not picked, since there is at least one valuation v, for which both $v(Y) \neq a$ and $v(Y) \neq c$. A characterization of the correctness of the Naive evaluation is given below.

Before going to the characterization, note that it is not always ideal to only return the certain answer. Namely, if the answer to q is to be materialized as a view for further querying, essential information is lost if the tuples with variables are dropped. For a simple example, a evaluating π_A on $sure(q, (T_1, T_2))$ gives tuple (a) as a sure answer, whereas evaluating $\pi_A(q(T_1, T_2))$, puts tuples (a) and (c) in the sure answer. As a consequence, query evaluation would not be compositional, unless $q(T_1, T_2)$ is stored as an "intermediate" answer. This "intermediate" answer is called the *exact answer* in [1], where the theory of query rewriting in information integration systems is extended to use the exact answer, instead of the certain one.

The correctness and completeness criteria for tables and query-evaluation is formalized using the notion of the representation system [2]. Here an alternative, equivalent formulation given in [3] is used: Consider a class of tables T, and a query language $\mathcal{Q}$. A triple (T, *rep*, $\mathcal{Q}$) is said to be a *representation system* if for every table $T \in$ T, and for every applicable $q \in \mathcal{Q}$, there exists a function (here also named) q, such that

$$\cap\, rep(q(T)) = \cap q(rep(T)), \text{ and} \tag{1}$$

$$q' \circ q(T) = q'(q(T)), \tag{2}$$

for all applicable $q' \in \mathcal{Q}$.

Condition (1) says that the system can correctly compute the certain answer, and condition (2) states that the computation has to be uniformly recursive, following the structure of q. The important result is that, the class of Naive tables, and the class of all negation-free relational algebra expressions, together with Naive evaluation, form such a representation system. And this result comes without any computational penalty.

Cross-references

- ▶ Certain (and Possible) Answers
- ▶ Incomplete Information
- ▶ Maybe Answer
- ▶ Naive Tables

Recommended Reading

1. Grahne G. and Kiricenko V. Towards an algebraic theory of information integration. Inf. Comput., 194(2): 79–100, 2004.
2. Imielinski T. and Lipski Jr. Incomplete information in relational databases. J. ACM, 31(4):761–791, 1984.
3. Lipski W. Jr. On relational algebra with marked nulls. In Proc. 4th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1985, pp. 201–203.

Name Matching

▶ Record Matching

Namelessness

▶ Anonymity

Narrowed Extended XPath I

Andrew Trotman
University of Otago, Dunedin, New Zealand

Synonyms

NEXI

Definition

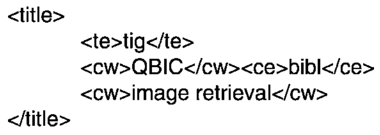
NEXI is an information retrieval (IR) query language for searching structured and semi-structured document collections. The language was first introduced for searching XML documents at the annual INEX [3] evaluation forum in 2004, and it has been used ever since.

Designed as the simplest query language that could possibly work, the language is a tiny subset of XPath [1] with an added *about*() function for identifying elements about some given topic. The language has extensions for question answering, multimedia searching, and searching heterogeneous document collections. NEXI is a language with a strict syntax defined in YACC but it has no semantics; the interpretation of the query is the task of the search engine.

Historical Background

A common information retrieval query language for searching XML documents was needed for specifying information retrieval queries at the first INEX in 2002. There, XML markup was chosen as the method of identifying keywords and the elements in which they should appear. It was also chosen as the method of identifying the preferred XML element to return to the user (the target element). The INEX 2002 query from topic 05 is given in Fig. 1. In this example, QBIC should be in a bibl element, image retrieval may appear anywhere in the document, and the user is interested in a list of tig elements as the result of the query.

Two problems with this format were identified: first it allowed the specification of queries that could be resolved by a simple mechanical process; second the



```
<title>
        <te>tig</te>
        <cw>QBIC</cw><ce>bibl</ce>
        <cw>image retrieval</cw>
</title>
```

Narrowed Extended XPath I. Figure 1. INEX topic 05 in the 2002 XML format.

language was not sufficiently expressive for information retrieval queries.

A modified XPath [1] was used at INEX 2003. In this variant the *contains*() function that required an element to contain the given content was replaced by an *about*() function that required an element to be about the content. Changing XPath in this way allowed fuzzy IR queries to be specified using a highly expressive language. However, an analysis of the XPath queries showed high syntactic and semantic error rates [6].

O'Keefe and Trotman [6] proposed using the simplest query language that could possibly work and a novel syntax. The INEX Queries Working Group [7] rejected the syntax but embraced the philosophy. It identified the minimum requirements of an IR query language for information retrieval queries containing structural constraints. This language, although at the time without syntax or semantics, was to be used at INEX for evaluation purposes.

Trotman and Sigurbjörnsson [9] proposed the Narrowed Extended XPath I (NEXI) language based on the working group report. It was narrowed in so far as only the descendant axis was supported, and extended in so far as the *about*() function was added, all other functions and axis were dropped. A formal grammar and parser were published, and an online syntax checker was hosted by the authors.

The decision to reduce XPath resulted in fewer errors because it reduced the chance of making mistakes. NEXI has a precise mathematical formulation which matches intuitive user profiles [4]. For both naïve users with knowledge of just the tag names, and for more advanced users with additional knowledge of the inter-relationships of those tags, the language is safe and complete. That is, the user cannot make semantic mistakes, and can express every information need they have.

Foundations

Web queries typically contain between 2 and 3 terms per query [8]. Formal query languages for semi-structured data tend to be comprehensive. This mismatch became apparent at INEX 2003 where XPath was chosen as the preferred language for information retrieval experts to specify relatively simple queries, but where they were unable to write syntactically and semantically correct queries. Just as SQL is not an end-user query language, neither, it turned out, was XPath.

Requirements

After two years of experimentation with XML query languages at INEX, the needs of such a language became apparent. The INEX, Queries Working Group [7] specified that the language should:

- Be compatible with existing syntax for specifying content only (keyword) queries.
- Be based on XPath as that language was already well understood, but:
- Remove all unnecessary XPath axis used for describing paths. Limit to just the descendant axis was suggested. The child operator was considered particularly problematic as it was open to misinterpretation.
- Drop exact match of strings, and inequality of numbers. XPath path filtering remained, however all strings were expressed as aboutness.
- Support multiple data types including numeric and string.
- Be open for extensions for new data types (including names, locations, dates, etc.).
- Not include tag instancing (for example author[1], the first author).
- Have vague semantics open to interpretation by the search engine.
- Loosen the meaning of the Boolean operators AND and OR.
- Disallow the multiple target elements. Although not explicit in the requirement, the implication is that the target element must be about the final clause in the query. It is a simple mechanical process to add non-target elements that are not about the query to the result – such as the author, title, source details to sections about something.
- Allow queries in which the target element was not specified and in which the search engine identified the ideal element.

Content Only (CO) Queries

NEXI addresses two kinds of queries on semi-structured and structured data: Content Only (CO) and Content And Structure (CAS) queries.

Content Only (CO) queries are the traditional IR query containing only keywords and phrases. No XML restrictions are seen and no mention is given of a preferred result (target) element. For these the NEXI syntax is derived from popular search engines: search terms can be keywords, numbers, or phrases (delineated with quotes). Term restrictions can be specified using plus and minus.

Information Retrieval queries are by their very nature fuzzy. A user has an information need and from that need they express a query. There are many different queries they might specify from the same need, some of which might be more precise than the others. If a document in the document collection satisfies the user's information need, that document is relevant regardless of the query. That is, no query term might appear in a relevant document, or all the query terms might appear, either way the document is relevant. When specifying an IR query language it is important to avoid specifying semantics that violate this principle of relevance. The semantics of the terms with and without restriction in NEXI is, for example, specified this way:

- *"The '+' signifies the user expects the word will appear in a relevant element. The user will be surprised if a '-' word is found, but this will not prevent the document from being relevant. Words without a sign are specified because the user anticipates such terms will help the search engine to find relevant elements. As restrictions are only hints, it is entirely possible for the most relevant element to contain none of the query terms, or for that matter only the '-' terms."*

Or, in other words, it is the task of the search engine to identify relevant documents even if this involves ignoring the query.

In INEX topic 210 the author states:

- *"I'm developing a new lecture for the Master course 'Content Design' and want to discuss the topic "Multimedia document models and authoring". Therefore I want to do a quick background search to collect relevant articles in a reader. I expect to find information in abstracts or sections of articles. Multimedia content is an essential component of my lecture, thus for fragments to be relevant they should address document models of content authoring approaches for multimedia content. I'm not interested in single media approaches or issues that discuss storing multimedia objects."*

The query they give is

- +multimedia "document models" "content authoring"

in which "`document models`" is a phrase and `+multimedia` is a term-restricted search term (is positively selected for by the user).

Content and Structure (CAS) Queries

The second kind of query addressed by NEXI is the Content and Structure (CAS) query. These queries contain not only keywords but also structural constraints know as *structural hints*. Just as the keywords are hints passed to the search engine in an effort to help with the identification of relevant documents, so too are structural hints. CAS queries contain two kinds of structural hints, where to look (support elements), and what to return to the user (target elements).

Formally, queries many take one of the forms in Table 1:

A and C are paths and B and D are filters. Other forms could easily be added, but since NEXI was originally designed to address the INEX query problem, they are not formally included.

Paths (A and C in Table 1) are specified as a list of descendants separated by the descendant axis //. Formally, a path is an ordered sequence of nodes $//E_1...//E_n$ starting with E_1 and finishing at E_n, and for all e∈n, E_e is a ancestor of E_{e+1}. An attribute node is indicated by the prefix @. Alternative paths are specified $(E_{na}|E_{nb})$. The wildcard * is used as a place holder.

For example, the path:

```
//article//*//(sec|section)//@author
```

describes an `author` attribute beneath either a `sec` or `section` element beneath something beneath an `article` element. The interpretation by the search engine is, of course, loose.

Filters (B and D in Table 1) can be either arithmetic or string. Arithmetic filters are specified as arithmetic comparisons (>, <, =, >=, <=) of numbers to relative-paths, for example:`.//year >= 2000`. String filters take the form `about(relative-path,`

Narrowed Extended XPath I. Table 1. Valid forms of NEXI CAS queries

Form	Target Element	Meaning
`//A[B]`	`A`	Return A tags about B
`//A[B]//C`	`A//C`	Return C descendants of A where A is about B
`//A[B]//C[D]`	`A//C`	Return C descendants of A where A is about B and a C descendant of A are about D

`COquery`). Filters can be combined using the Boolean operators and, and or. Paths and filters are all considered hints and there is no requirement for the search engine to distinguish between the Boolean operators.

The target elements for the forms given in Table 1 are specified in column 2. Target elements, like support elements, are also hints. If, for example, the user specified paragraphs a subsection element might fulfill the user's information need.

An example of a valid NEXI CAS query (again from INEX topic 230) is:

```
//article[about(.//bdy, “artificial intelligence”) and.//yr < = 2000]//bdy[about(., chess) and about(., algorithm)]
```

in which the target element is `//article//bdy`. The user has specified an arithmetic filter.`//yr <= 2000`. Several string filters are used including `about(.//bdy, ‘‘artificial intelligence”)`. A Boolean operator is also used to separate two filters `about(., chess) and about(., algorithm)`.

The NEXI CAS query from INEX topic 210 is an alternative expression of the information need given in the previous section. That query is:

```
//article//(abs|sec)[about(.,+multimedia ‘‘document models’’ ‘‘content authoring’’)]
```

in which the target element is either `//article//abs` or `//article//sec`. The same documents and elements are relevant to both queries, as relevance is with respect to the information need and not the specific query.

Key Applications

Information retrieval from structured and semi-structured document collections.

Future Directions

Although proposed as an XML query language for use in an evaluation forum, there is evidence it may also be an effective end-user language. Van Zwol et al. [13] compared NEXI to a graphical query language called Bricks. They found that a graphical query language reduced the time needed to find information, but that users were more satisfied with NEXI. Inherent in text query languages is the problem that users are required to know the structure (the DTD) of the documents. In a heterogeneous environment this may not be possible, especially if new and different forms of data are constantly being added. Graphical query languages that translate into an intermediary text-based query language are one solution. This solution is seen with graphical user interfaces to relational databases.

Woodley et al. [12] further the model of NEXI as an intermediate language and compare NLPX (a natural language to NEXI translator) to that of Bricks (a graphic to NEXI translator). They show that users prefer a natural language interface, and that the performance of the two is comparable.

Ogilvie [10] examined the use of NEXI for question answering and proposed extensions to the language for this purpose. Dignum and van Zwol [2] proposed extensions for heterogeneous searching. Trotman and Sigurbjörnsson [10] unified these proposals and formally extended the language to include both – however, these extensions are not considered core to the language (language extensions philosophically deviate from the principle of simplest that could possibly work). Multimedia extensions to the language have also been used at INEX [11], again the extensions are not considered core to the language.

Experimental Results

The analysis of XPath queries used at INEX 2003 showed 63% of queries containing either syntactic or semantic errors [3]. An analysis of the errors in NEXI queries used at INEX 2004 showed that only 12% contained errors [12]. NEXI has been in use at INEX ever since.

Data Sets

NEXI queries from INEX 2004 onwards can be downloaded from the INEX web site: http://inex.is.informatik.uni-duisburg.de/

INEX queries for 2003 and 2002 were translated into NEXI (where possible) and can be downloaded from the NEXI web page hosted by the University of Otago: http://metis.otago.ac.nz/abin/nexi.cgi

URL to Code

An online syntax checker, lex and yacc scripts, and a command line syntax checker can be downloaded from the NEXI web page hosted by the University of Otago: http://metis.otago.ac.nz/abin/nexi.cgi

Cross-references

- ▶ Content-and-Structure Query
- ▶ Content-Only Query
- ▶ INitiative for the Evaluation of XML Retrieval
- ▶ Processing Structural Constraints
- ▶ Query by Humming
- ▶ Query Languages for the Life Sciences
- ▶ Semi-Structured Query Languages
- ▶ Temporal Query Languages
- ▶ XML
- ▶ XPath/XQuery
- ▶ XQuery Full-Text
- ▶ XSL/XSLT

Recommended Reading

1. Clark J. and DeRose S. XML path language (XPath) 1.0, W3C recommendation. The World Wide Web Consortium. Available at: http://www.w3.org/TR/xpath 1999.
2. Dignum V. and van Zwol R. Guidelines for topic development in heterogeneous collections. Available at: http://inex.is.informatik.uni-duisburg.de:2004/internal/hettrack/downloads/hettopics.pdf 2004.
3. Fuhr N., Gövert N., Kazai G., and Lalmas M. INEX: initiative for the evaluation of XML retrieval. In Proc. ACM SIGIR 2002 Workshop on XML and Information Retrieval, 2002.
4. Kamps J., Marx M., Rijke Md., and Sigurbjörnsson B. Articulating information needs in XML query languages. Trans. Inf. Syst., 24(4):407–436, 2006.
5. Ogilvie P. Retrieval using structure for question answering. In Proc. 1st Twente Data Management Workshop - XML Databases and Information Retrieval, 2004, pp. 15–23.
6. O'Keefe R.A. and Trotman A. The simplest query language that could possibly work. In Proc. 2nd Workshop of the Initiative for the Evaluation of XML Retrieval, 2003.
7. Sigurbjörnsson B. and Trotman A. Queries: INEX 2003 working group report. In Proc. 2nd Workshop of the Initiative for the Evaluation of XML Retrieval, 2003.
8. Spink A., Wolfram D., Jansen B.J., and Saracevic T. Searching the web: the public and their queries. J. Am. Soc. Inf. Sci. Tech., 53(2):226–234, 2001.
9. Trotman A. and Sigurbjörnsson B. Narrowed extended XPath I (NEXI). In Proc. 3rd Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, pp. 16–40.
10. Trotman A. and Sigurbjörnsson B. NEXI, now and next. In Proc. 3rd Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, pp. 41–53.
11. Westerveld T. and van Zwol R. Multimedia retrieval at INEX 2006. SIGIR Forum, 41(1):58–63, 2007.
12. Woodley A., Geva S., and Edwards S.L. Comparing XML-IR query formation interfaces. Aust. J. Intell. Inf. Process. Syst., 9(2):64–71, 2007.
13. van Zwol R., Baas J., van Oostendorp H., and Wiering F. Bricks: the building blocks to tackle query formulation in structured document retrieval. In Proc. 28th European Conf. on IR Research, 2006, pp. 314–325.

NAS

▶ Network Attached Storage

NAS Servers

▶ Storage Devices

NASD

▶ Network Attached Secure Device

Natural Human-Computer Interaction (NHCI)

▶ Natural Interaction

Natural Interaction

STEFANO BARALDI, ALBERTO DEL BIMBO, LEA LANDUCCI, NICOLA TORPEI
University of Florence, Florence, Italy

Synonyms

Natural human-computer interaction; NHCI

Definition

The aim of Natural Human-Computer Interaction (NHCI) research is to create new interactive frameworks that integrate human language and behaviour into tech applications, focusing on the way people live, work, play and interact with each other. Such frameworks have to be easy to use, intuitive, entertaining and non-intrusive.

The design of *natural interaction* systems is focused on recognizing innate and instinctive *human expressions* in relation to some object, and return to the user a corresponding feedback that has the characteristics of being both *expected* and *inspiring*. All of the technology and the intelligence is built inside the digital artifacts and the user is not asked to use external devices, wear anything, or learn any commands or procedures. An interesting challenge for NHCI is therefore to make systems self-explanatory by working on their “affordance” [11] and introducing simple and intuitive interaction languages.

The human expressions that can be utilized are those considered innate, meaning that they don’t have to be learned. This includes vocal expressions and all the gestures used by humans to explore the nearby space or the immediate surroundings with their bodies, like: touching, pointing, stepping into zones, grabbing and manipulating objects (see Fig. 1). These direct actions express a clear sign of interest and necessitate a sudden reaction from the system.

The application fields range from browsing multimedia contents to exploration of knowledge structures; the scenarios involved can be either task-specific (in office or research contexts) or experiences created for casual interaction between the visitors and the media contents (like in museum exhibits and creative installations).

Natural interaction interfaces are very interesting to access, and explore large data sets like the ones contained in multimedia or geo-referenced databases. Data mining applications, which usually ask the user to enter complex search criteria and tweak the parameters to obtain the wanted data, can be designed with a visual analytic interface following these design guidelines. Queries are expressed visually, selecting interactive objects that embody selection criteria, creating clusters of them. Result sets provided by the database back-end can be iteratively shaped by directly manipulating the visual elements mapping the data, including other sets or reducing them.

Historical Background

At the beginning of the nineties, Human-Computer Interaction [17] was completely integrated into the Computer Science field, its quick growth led to the development of technologies able to go over the standard limiting user-computer communication paradigm, approaching a kind of natural interaction.

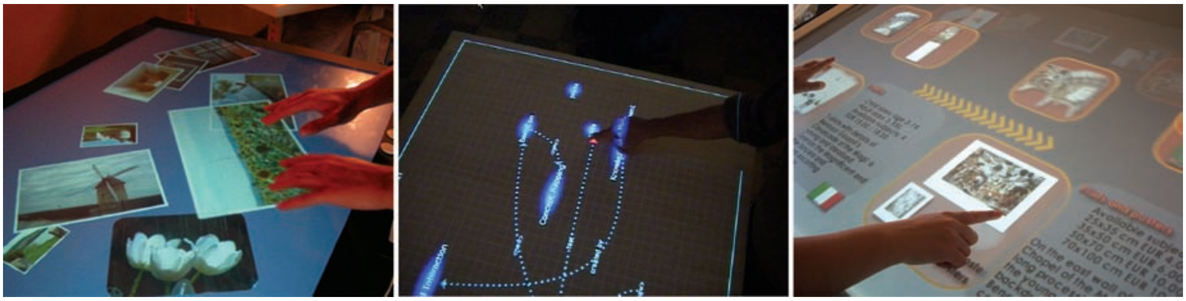
Alex Pentland in his “The Dance of Bits and Atoms” (1996, [13]) said that “There is a deep divide between the world of bits, and the world of atoms. Current machines are blind and deaf; they are unaware of us or our desires unless we explicitly instruct them. Consequently, only experts use most machines, and even they must spend most of their time battling arcane languages and strange, clunky interface devices.”

The key point is that technologies must be designed and developed adapting them to the users, not the contrary: this is the foundation of the “human centered design” [11,12].

The goal, finally, becomes a technology at one user’s service, suitable to the task requested and characterized by the complexity naturally embedded in the task itself [11,12].

Foundations

Interactive artifacts, augmented spaces, ambient technology and ubiquitous computing are all fields of study in HCI, which concentrate on both the technological aspects and the modalities in which users can perform activities. The techniques proposed are often referred to as *multi-modal interaction*, focusing on how machines can understand commands coming from different channels of human communication [10]. Among those: speech recognition, natural language understanding and gesture recognition, have been applied in different mixtures with roles ranging from active (the system observes the user and is pro-active in interacting with him) to passive (the system expects some kind of command, often through a device that is worn by the user). In the last years, studies have been following the concept of natural interaction as a conjunction of



Natural Interaction. Figure 1. Examples of natural interfaces for accessing digital contents and structures.

different technologies and design principles, with a more radical view about the user freedom in using interactive artifacts.

Natural interaction systems can be modelled as the sum of different modules: the *sensing* subsystem, which gathers sensor data about user expressions and behaviour, and the *presentation* module, which realizes the dialogue with the user orchestrating the output of different kind of actuators (graphics display, audio, haptics).

Sensing

The initial expression of interest towards the digital artefact is distracted from its main role: transferring information and stimuli. For this reason, the artefact should hide all the needs for an external controlling device, and be able to *sense* what the user is trying to do.

From a technological point of view, sensing involves the use of different sensors that provide data about some physical dimension in the surroundings of the artefact. There are a great number of electronic sensors that have been used in many industrial fields for robotics, automation and automatic inspection. Ranging from video cameras and image processing algorithms, to capacitive sensors for touch and pressure sensing or accelerometers for gesture recognition and body articulation.

Every sensor can have very different ways of providing this kind of data in terms of resolution, range, tolerances and errors. Some of them are able to provide discrete data with a high certainty, while others (like video cameras) just provide a great amount of data that has to be processed and interpreted by algorithms in order to extract useful information.

Considering the sensing architecture as a whole, a processing logic must be applied on top of it to abstract all the singularities of the sensors and create a homogeneous model of events for the later stages of interaction. Discrete events arriving from the sensors can be considered as belonging to three different categories of data:

- *Presence*: this data usually comes with a high signal-to-noise ratio, meaning that it cannot be directly interpreted as a human expression. Nonetheless, it reports a general activity in a spatial area.
- *Behaviour*: as the certainty level of some event increases (thanks to the combination of different sensors data), data can be studied for a certain period of time and provide information about the behaviour of single users or groups.
- *Activity*: this kind of data are considered "certain" and interpreted as a clear user intention to be used for a direct control of the interface. Instead of providing just the bi-dimensional screen coordinates as, it happens with traditional touch-screens, advanced sensing can interpret speed or pressure as well as distance of the hands as they approach a surface.

Sensors can be displaced in the environment, embedded in the artefact and also worn by the user but for natural interaction systems the last option is not suggested. For indoor use, the most meaningful displacement of sensors is inside the artefact itself. In this way, space can remain flexible and structures can be moved around. Such a thing could not be done if the whole environment was disseminated with.

Intelligence and Presentation

The intelligence substratum is what orchestrates the signals coming from sensors and generates output for presentation.

The devices embedded in the artefacts that manage the output phase are called *actuators* and can range from visual displays, to audio systems, surfaces with tactile feedback, holograms and many others. In scenarios like museums or art installations, even simple forms of stimuli can be used and mixed to convey a sensation or to catch the attention of the user. In an information space like an office or a data-intensive environment, the main channel of parallel communication remains the visual one, and the actuator is the graphical display.

On traditional GUIs (Graphical User Interfaces), what happens "inside the screen" can be subject to inference and prediction. Modern interfaces gather data about their use and are predictive in searching and suggesting data to the user in a light and "polite" way. Natural interfaces can extend this reasoning to a lot more data coming from the sensors, because the interactive artefact itself tries to *sense* beyond the screen. Also, unlike traditional interfaces whose informational layout is totally independent from the physical place in which the device is installed and used, digital artefacts can consider their position in the environment and what is happening around them.

In natural graphical interfaces, the basic assumption is that digital elements are not just a representation of data, but a part of the environment, and they need to follow aesthetics rules too. As a first principle: digital elements should behave like objects in physical systems. Following this *analogy* the graphical display becomes a real *space*, and therefore has to provide the same affordance of a normal surface. If something changes on the visual interface it should happen in a smooth way, without sudden jumps, similar to what happens with hyperlinks on web pages. The models of movement and forces used in transitions have to mimic those of one real world, like: gravity, accelerating forces, momentum and friction. The user has the chance to understand what is going on without feeling disoriented, and thanks to these visual cues he can also expect what is going to happen.

The anticipation is more important because in shared natural interfaces the model of control of traditional interfaces, modelled around a question to confirm every action, cannot be used. A popup message box would block the interface occupying space and this is not feasible in a shared user environment. Instead, every action is directly executed and, in the time in which this happens, the visual features of objects influenced by the action gradually change. Another guideline for the visual interface design is *lightness*. On every graphical display, there is a conflict between the level of contents belonging to the domain of the application (diversified media), and the level of widget elements (labels, buttons, etc) that explain the interface to the user and give hints about the options. In natural interfaces this is even truer. Basically, the space is for contents, so any other element is stealing some attention from the user. For this reason, the widget level is kept to a minimal amount of symbols and there is little or no use of "global" interface elements.

Handling Complexity with TUIs

A challenge in natural interaction systems is tackling the complexity with simplicity. While the category of exploring applications can be realized using only innate gestures, in the case of complex applications (featuring multiple options and actions) simple and spontaneous hand gestures turn out to be not enough, and methods like speech understanding do not adapt well to noisy environments or to scenarios in which multiple people are interacting simultaneously.

The solutions could be:

1. Enriching the interaction language by adding new complex gestures that map to actions. This could distort the naturalness of interaction forcing users to learn unnatural gestures.
2. Introducing an intermediate visual level using interface elements (such as menus, icons etc.). This would reduce the interaction directness causing a conflict between digital contents and interface elements, both sharing the same visualization area.

The result is that such solutions could increase the user cognitive load. Tangible user interfaces (TUIs [8]) can be an alternative solution to those mentioned. They introduce physical, tangible objects that the system interprets as embodiment [6] of the elements of the interaction language. Users, manipulating those objects, inspired by their physical affordance, can have a more direct access to functions mapped to different objects [2].

There is a broad literature about TUIs which dates back to the first experiments of Hiroshi et al at the MIT, where a set of normal objects where "sensed" by a system who could recognize and track some of their features, like the position in space. Many other systems have been developed using this paradigm, and today there is a growing familiarity with taxonomies that try to define the different styles and scenarios in which it can be used [7]. Table 1 illustrates the main different branches.

The physical objects, called *tangibles*, could address some of the issues related to complexity. They can become the *embodiment* of some aspects of the interaction between the user and the domain of multi-media contents handled by the application. In particular three possible roles can be distinguished.

- *The tangible as a simulacrum*. The physical object can be used as a representative of a single digital object or a collection. This means either a uniquely identified static element

Natural Interaction. Table 1. Tangible interaction themes

Theme	Features
Tangible manipulation	Physical manipulation of tactile qualities
Spatial interaction	Movement in space
Embodied facilitation	Configuration of objects affecting group behavior
Expressive representation	Focus on digital and physical expressiveness

with a one-to-one mapping between the physical and the digital world, or a re-assignable element that can be associated with different data (like a container).

- *The tangible as a manipulator.* In this case the tangible represents a *function* that can be applied to a digital object. The closest metaphor is that of a tool in order to change one of its aspects, reveal other connected contents etc. This modality can be also extended to global functions whose target is the entire viewport or collection of digital objects.
- *The tangible as an avatar of the user.* In this case, every user would have a personal object that represents himself, to move across the contexts. An avatar object can be used in conjunction with the others when the application needs an authentication or when the activity proposed requires the user to express a preference.

While the manipulator role is specific to the nature and local interaction with the artefact (e.g., a digital tabletop), the simulacrum and avatar roles exactly provide the abstraction that is needed in order to expand the affordance of the environment, providing a natural way to transport the productions across the artifacts. In this fashion, activities can be initiated on an artefact and continued somewhere else, like in a laboratory, different places provide different contexts and options for the same data.

Key Applications

Multimedia Browsing

Usually they are easy-to use systems where people can interact simultaneously with multimedia contents through their own bare-hand gesture.

This kind of application offers an intuitive approach to various multimedia objects, they don't require any kind of training or instructions.

Knowledge Exploration and Building

Interactive workspace featuring vision-based gesture recognition that allows multiple users to collaborate [3] in order to realize face-to-face contexts, designing a common workspace where users can build knowledge (activities like brainstorming or problem solving sessions), exploiting the useful scenario-specific characteristics.

Interactive Museum and Cultural Exhibits

Museums and exhibitions are often just a collection of objects, standing deaf in front of visitors. In many cases, objects are accompanied by textual descriptions, usually too short or long to be useful for the visitor. In the last decade, progress in multimedia has allowed for new, experimental forms of communication (using computer technologies) in public spaces [1].

Interactive Music Systems

Usually built over a tabletop tangible user interface, Interactive Music Systems allow several simultaneous performers to share complete control over the instrument by moving physical artefacts on the table surface while constructing different audio topologies in a kind of tangible modular synthesizer. The reacTable [9], a clear example, is a novel multi-user electro-acoustic musical instrument with a tabletop tangible user interface.

Cross-references

▶ Human-Computer Interaction
▶ Multimedia Databases
▶ Multimodal Interfaces
▶ Object Recognition
▶ Visual Interfaces
▶ Visual Perception
▶ Visual Representation

Recommended Reading

1. Alisi T.M., Del Bimbo A., and Valli A. Natural interfaces to enhance visitors' experiences. IEEE Multimed., 12(3):80–85, 2005.
2. Baraldi S., Del Bimbo A., Landucci L., Torpei N., Cafini O., Farella, E., Pieracci A., and Benini L. Introducing TANGerINE: a tangible interactive natural environment. In Proc. 5th ACM Int. Conf. on Multimedia, 2007, pp. 831–834.
3. Baraldi S., Del Bimbo A., Landucci L., and Valli A. wikiTable: finger driven interaction for collaborative knowledge-building workspaces. In Proc. 2006 IEEE Int. Conf. on Computer Vision and Pattern Recognition Workshop, 2006, p. 144.
4. Colombo C., Del Bimbo A., and Valli A. Visual capture and understanding of hand pointing actions in a 3-D environment. IEEE Trans. Syst. Man. Cybern. B Cybern., 33(4):677–686, 2003.
5. Dietz P. and Leigh D. DiamondTouch: a multi-user touch technology. In Proc. 14th Annual ACM Symp. on User Interface Software and Technology, 2001, pp. 219–226.
6. Fishkin K.P. A taxonomy for and analysis of tangible interfaces. Pers. Ubiquitous Comput., 8(5):347–358, 2004.
7. Hornecker E. and Buur J. Getting a grip on tangible interaction: a framework on physical space and social interaction. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2006, pp. 437–446.

8. Ishii H. and Ullmer B. Tangible bits: towards seamless interfaces between people, bits and atoms. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1997, pp. 234–241.
9. Kaltenbrunner M., Jordà S., Geiger G., and Alonso M. The reacTable: a collaborative musical instrument. In Proc. Workshop on Tangible Interaction in Collaborative Environments, 2006.
10. Marsic I., Medl A., and Flanagan J. Natural communication with information systems. In Proc. IEEE, 88(8):1354–1366, 2000.
11. Norman D.A. The Design of Everyday Things. MIT PRESS, Cambridge, MA, 1998.
12. Norman D.A. The Invisible Computer. MIT PRESS, Cambridge, MA, 1999.
13. Pentland A., Smart rooms. Sci. Am., 274(4):54–62, 1996.
14. Prante T., Streitz N.A., and Tandler P. Roomware: computers disappear and interaction evolves. IEEE Comput., 37 (12):47–54, 2004.
15. Ulmer B. and Ishii H. Emerging frameworks for tangible user interfaces. IBM Syst. J., 39(3–4):915–931, 2000.
16. Valli A. Notes on Natural Interaction. http://naturalinteraction.org, 2005.
17. Wania C.E., Atwood M.E., and McCain K.W. How do design and evaluation interrelate in HCI research? In Proc. 6th Conf. on Designing Interactive Systems, 2006, pp. 90–98.

Natural Language Generation (NLG)

▶ Text Generation

Navigation System Interfaces

▶ Mobile Interfaces

Near-Duplicate Video Retrieval

▶ Video Querying

Nearest Neighbor Classification

Thomas Seidl
RWTH Aachen University, Aachen, Germany

Synonyms

NN classification; *k*-nearest neighbor classification; k-NN classification

Definition

Nearest neighbor classification is a machine learning method that aims at labeling previously unseen query objects while distinguishing two or more destination classes. As any classifier, in general, it requires some training data with given labels and, thus, is an instance of supervised learning. In the simplest variant, the query object inherits the label from the closest sample object in the training set. Common variants extend the decision set from the single nearest neighbor within the training data to the set of k nearest neighbors for any $k > 1$. The decision rule combines the labels from these k decision objects, either by simple majority voting or by any distance-based or frequency-based weighting scheme, to decide the predicted label for the query object. Mean-based nearest neighbor classifiers group the training data and work on the means of classes rather than on the individual training objects. As nearest neighbor classifiers only require distance computation of objects as a basic operation, their applicability exceeds the domain of vector spaces and attribute-based tuple data, and includes metric data spaces even if there are no attributes or dimensions available. Nearest neighbor classification counts for being highly accurate but inefficient; the latter assessment is to be invalidated by using appropriate indexing structures that support fast k-nearest neighbor retrieval. The abbreviation "NN classification" tends to cause confusion with the totally different concept of neural network classification.

Historical Background

Nearest neighbor classification is one of the earliest classification schemata. Nevertheless, it is one of the most highly accurate ones, whilst obeying a very broad range of applicability. The following aspects distinguish it from other classification methods.

Multiple class labels. Nearest neighbor classification is not restricted to two-class problems where classification decisions have to distinguish two classes only. Moreover, a high number of different classes does not deteriorate the efficiency in contrast to many other classifiers which create individual models for all the classes in the training set.

Applicability to general metric data spaces. Nearest neighbor classification is applicable to data without any structured attribute representation, i.e., to non-vectorial data. As distance computations are the only required basic operation, the applicability includes complex multimedia data, time series and measurement series data, sequences and structural data to name just a few supported domains. Among the competing approaches in the field, only support vector machines equipped with respective kernel functions share this advantage. Other classifiers including decision trees, neural networks and Bayes density models rely on the attribute structure of the data.

Instance-based learning. Nearest neighbor classification derives decisions close to the individual training instances as any other method. No model is created for the training data and, thus, it is called lazy evaluation. Most of the competing classification approaches follow the eager evaluation paradigm which spends significant training effort in the determination of models for the training data. They produce decision tree structures, synaptic weights in neural networks, support vector weights, or density probability functions, respectively. The lazy model of nearest neighbor classification nevertheless allows for fast class decisions when spending (training) effort in the creation of an index on the training data.

Training data that changes dynamically. If the classifier has to follow training data that changes over time, without or even with respect to its statistical characteristics, a nearest neighbor classifier is superior to almost all of the competing classifiers. No model parameters such as decision tree structures or density weights have to be recomputed where the training data has changed. The lazy evaluation strategy ensures that the results always rely on the current training data status. The efficiency depends on how the underlying index structure supports the dynamic data changes.

Intuitive explanation component. Nearest neighbor classifiers provide illustrative explanations of their decisions by revealing the closest neighbors of the query object from the training data. Moreover, the corresponding similarity scores, i.e., the distances of the training items to the query object, provide some insight into the decision process, and the users may derive their confidence in the final decision result. In comparison, support vector machines analogously give illustrative weights to individual training objects, Bayes classifiers have a similar explanation power by providing probability values for each target class, and decision trees are slightly more intuitive by presenting the rules that produced the decision result. On the other hand, neural network classifiers lack illustrative explanations of their decisions.

Foundations

Classification Model

Nearest neighbor classifiers are a common classification model for which several variants exist. Along with the simple nearest neighbor model, *k*-nearest neighbor classification uses a set of k neighbors and the mean-based nearest neighbor model where individual training objects are generalized uses group representatives.

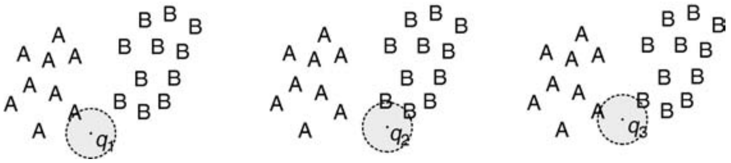
Simple nearest neighbor classification. The simplest variant of nearest neighbor classification may be formalized as follows: For any metric object space O, let $TS \subseteq O$ denote the set of labeled training data and $d : O \times O \rightarrow \Re_0^+$ is the chosen and thus distance function that reflects the dissimilarity of any two objects from O. Then, for any query object $q \in O$, the classifier evaluates the function $C_{\text{preliminary}}(q) = \{label(o)|\ o \in TS, \forall p \in TS - \{o\}: d(o,q) < d(p,q)\}$.

Unfortunately, this formalization is not valid for cases where several objects $o \in TS$ share the same minimal distance $d(o,q)$ to the query object, i.e., when the query hits perpendicular bisectors of these objects. Though in practice, these ties occur very rarely, a formalization that is equivalent to the previous one in other respects but in which ambiguities may be broken simply in a nondeterministic way is preferred (Fig. 1):

$$C_{nn}(q) = \{label(o)|\ o \in TS, \forall p \in TS : d(o,q) \leq d(p,q)\} \quad (1)$$

The decision model of nearest neighbor classification allows for a particularly broad range of applications. Its applicability covers vector data which are characterized by numerical attributes and includes general relational data where the tuples are represented by both numerical and categorical data as long as some metrics are defined on the categories. Moreover, if training and test data are neither taken from a vector space nor carry any categorical attributes but are only compared in terms of a distance function that reflects the dissimilarity of objects, nearest neighbor classifiers are still applicable. Examples of such applications include sequence data or structural data which may be compared by the edit distance, time series and measurement series compared by dynamic time warping, text and document data compared by the cosine distance, or multimedia data compared by complex similarity models.

Mean-based nearest neighbor classification. Nearest neighbor classifiers yield high quality decisions in



Nearest Neighbor Classification. Figure 1. Ambiguity of nearest neighbors for a sample training data set with two classes, A and B. Query object q_1 has a unique nearest neighbor and will be assigned to class A. For query object q_2, two neighbors share the smallest distance but there is no ambiguity with respect to the class decision since both neighbors are in class B. Query q_3 yields a conflict as it has two nearest neighbors from different classes, namely A and B, respectively. In practice, these ties occur very rarely at least for numerical reasons but may be solved nondeterministically.

terms of classification accuracy, and they are quite robust even for small training sets. Nevertheless, they tend to suffer from overfitting since the decisions are made closer at the training data than it is the case for any other method. As a consequence, erroneous training data, noise and outliers may badly affect the decisions. In order to increase the generalization power and to approach the overfitting problem, mean-based nearest neighbor classifiers and *k*-nearest neighbor classifiers have been developed as common variants of the simple model.

For mean-based nearest neighbor classification, the objects in the training data set are grouped into one or more clusters per class. These clusters are represented by their means, and nearest neighbors are then selected among the means rather than among the original individual training objects. The means inherit the class label of their cluster members, and the decision rule (1) from simple nearest neighbor classification is easily adopted to the mean model just by replacing the training data set by the set of means:

$$C_{m-nn}(q) = \{label(m)|\ m \in means(TS), \forall n \in means(TS) : d(m,q) \leq d(n,q)\} \tag{2}$$

Note that the computation of mean values requires the object space O to be a vector space, since the objects of a cluster need to be summed up followed by a scalar multiplication by the reciprocal value of the group's cardinality. Thus, the mean-based model does not apply to non-vectorial metric spaces which, in contrast, are supported by *k*-nearest neighbor classifiers.

k-nearest neighbor classification. Beside mean-based nearest neighbor classifiers, *k*-nearest neighbor classification is another quite common approach to increase the generalization power and to decrease overfitting effects. Instead of looking at a single object to the query among the training data, a set of *k* nearest neighbors, $k > 1$, is taken into account when making the decision. One starts by defining the decision set to be a subset of the training data TS that contains the *k* objects closest to the query object *q*. Again, ties may be broken by a nondeterministic choice among equidistant neighbors:

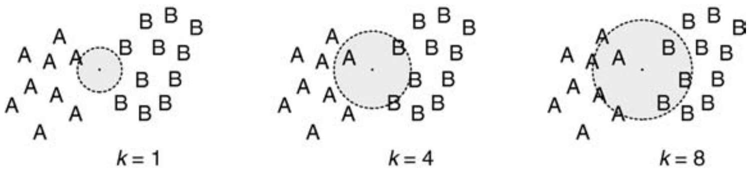
$$NN(q,k) = \{o|\ o \in TS, \forall p \in TS - NN(q,k) : d(o,q) \leq d(p,q)\},\ \ |NN(q,\ k)| = k \tag{3}$$

The decision rule based on majority vote now looks as follows (Fig. 2):

$$C_{majority}(q) = \arg\max_{l \in label(NN(q,\,k))} \{card\{o \in NN(q,k)|\ label(o) = l\}\}. \tag{4}$$

Weighted k-nearest neighbor classification. For *k*-nearest neighbor classifiers, different weighting schemata have been developed which introduce distances or frequencies into the decision rule. Conceptually, majority voting represents a weighting schema with unit weights. A quite common variant is to use the squared distances of the decision objects to the query object as reciprocal weights. This way, the desired effect is obtained that the closer an object is to the query, the higher is its influence to the classification decision. The corresponding decision rule is as follows:

$$C_{dist}(q) = \text{argmax}_{l \in label(NN(q,k))} \left\{ \sum_{\substack{o \in NN(q,k) \\ label(o)=l}} \frac{1}{d(o,q)^2} \right\}. \tag{5}$$



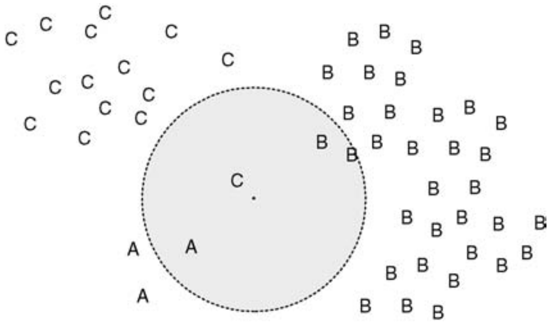
Nearest Neighbor Classification. Figure 2. Dependency of the classification decision on the number of nearest neighbors. In the examples from left to right, the same query object is considered. A choice of k = 1 yields the class label A. For k = 4, class B holds the majority in the nearest neighbor set, and for k = 8, again label A is assigned to the query object.

An alternative weighting schema takes the relative frequency of the classes into account. Naturally, rare classes are represented only by a small number of instances in the training data. If at decision time, a nearest neighbor set contains a few of these rare instances, this occurrence is honored by the frequency weighting schema. This way, rare classes may outvote highly frequent classes even if these dominate the decision set by their number. Formally speaking, decisions are based on the fraction of training instances a class contributes to the decision set rather than on the absolute number of instances among the nearest neighbors (Fig. 3).

$$C_{freq}(q) = \operatorname{argmax}_{l \in label(NN(q,k))} \left\{ \frac{card\{o|\ o \in NN(q,k), label(o) = l\}}{card\{o|\ o \in TD, label(o) = l\}} \right\}. \qquad (6)$$

Lazy Evaluation Model

Nearest neighbor classification follows the model of lazy evaluation. This means that there is no computation or estimation of model parameters at training time, but all calculations to make decisions are deferred to query time. Lazy evaluation is in contrast to the eager evaluation paradigm which requires some effort at training time to build an appropriate model. Examples of eager classifiers include decision trees, Bayes classifiers, neural networks, or support vector machines for which attribute-oriented decision flow diagrams, probability (mixture) density models, synaptic weights, or support vector weights are computed at training time, respectively. Lazy evaluation does not build models for the training data but derives its decisions directly from the training objects and, therefore, is called instance-based learning. Nearest neighbor classification is a prominent example of lazy evaluation, and the simple 1-nn and extended *k*-nn variants purely reflect the instance-based paradigm. The mean-based variant, however, deviates a little from this idea since means of classes are computed at training time, and decisions are based on the means rather than on the individual instances. For lazy evaluation in general, questions about decision efficiency, dynamic changes of training data, and explanations of decisions arise; these issues are discussed in the following subsections. Note that mean-based nearest neighbor classification is not a pure lazy variant since means of classes are computed at training time.



Nearest Neighbor Classification. Figure 3. Dependency of class decision on the weighting schema. In the example, a 4-nearest neighbor set is selected for the query object. Pure majority voting decides for class B as the decision set includes two instances of class B. Distance weighting decides for class C as the close neighbor from class C dominates the farer neighbor from class A and even the two neighbors from class B. Frequency-based weighting yields class A since A contributes the highest fraction of its overall occurrences in the training data to the nearest neighbor set.

Decision efficiency. By following the lazy evaluation model, nearest neighbor classifiers defer all data analysis to query time. So query processing tends to be computationally more expensive than with eager classifiers. Obviously, the most complex task is to determine the decision set, that is the set of (k) nearest neighbors for a query object; the subsequent weighting and voting is negligibly done in $O(k)$. Retrieving the k nearest neighbors in a set of n training objects depends on the data organization. Whereas a sequential scan requires $O(n)$ operations, the use of multidimensional or metric indexing structures may significantly speed-up the retrieval of the decision set depending on the dimensionality and on the statistical characteristics of the data. For high-dimensional data, techniques for reduction of dimensionality and multi-step query processing help to keep k-nearest neighbor retrieval efficient. As strict lazy evaluation does not waste any training time to create a model, it is nevertheless highly recommended to spend some effort for the creation of an appropriate index on the training data. In short, to train an efficient nearest neighbor classifier means to instantiate model creation by index creation.

Dynamically changing training data. As for lazy evaluation, no model is created at training time, a lazy classifier in general allows for dynamic changes of training data particularly well. For a nearest neighbor classifier, efficiency depends on how the underlying index structure maintains the dynamic data set. This represents a common requirement for indexing and is supported well by almost all multidimensional or metric access methods. As training data changes, new examples are inserted into the index and outdated training instances are removed. This way, incremental training for low change rates and also online training with potentially high rates of incoming training data are enabled. Nearest neighbor classification, thus, always produces a result that relies on the current training data status.

Decision explanation. Lazy classifiers do not create models for the training data and, as an inherent limitation, no explicit knowledge is extracted which reflects the structure of the data. Though nearest neighbor classifiers do not provide intuitive models of the data characteristics, they nevertheless explain their individual decisions illustratively by revealing the closest neighbors of the query object within the labeled training data to the user. Additional information is provided by the similarity scores, i.e., the distances of the training items to the query object, thus yielding some insight into the decision process from which the users may derive their confidence in the final decision result.

Key Applications

Key applications of nearest neighbor classification include all areas where classification problems occur. Particularly well suited are nearest neighbor classifiers in the following cases:

1. Many classes need to be distinguished
2. Objects are represented by general metric data without an attribute structure
3. Classifier has to follow training data that change dynamically
4. Users require intuitive explanations of the system's decisions

Real applications are found in all areas of multimedia data exploration and cognitive systems including computer vision, speech recognition, medical imaging, robot motion planning, or sensor-based object recognition in general to name just a few.

Cross-references

- ▶ Applications
- ▶ Indexing Metric Spaces
- ▶ Indexing: R-Trees
- ▶ Multidimensional Indexing
- ▶ Multimedia Information Retrieval
- ▶ Nearest Neighbor Query
- ▶ Spatial Indexing Techniques
- ▶ Spatial
- ▶ Spatiotemporal
- ▶ Text Indexing Techniques

Recommended Reading

1. Ankerst M., Kastenmuller G., Kriegel H.-P., and Seidl T. Nearest neighbor classification in 3D protein databases. In Proc. 7th Int. Conf. on Intelligent Systems for Molecular Biology, 1999, pp. 34–43.
2. Athistos V. Nearest neighbor retrieval and classification. Available at: http://cs-people.bu.edu/athitsos/nearest-neighbors/, 2007.
3. Djouadi A. and Bouktache E. A fast algorithm for the nearest-neighbor classifier. IEEE Trans. Pattern Anal. Mach. Intell., 19(3): 277–282, 1997.
4. Duda R.O., Hart P.E., and Stork D.G. Pattern Classification (2nd ed.). Wiley, New York, 2001.
5. Efros A.A., Berg A.C., Mori G., and Malik J. Recognizing action at a distance. In Proc. 9th IEEE Conf. Computer Vision, 2003, pp. 726–733.
6. Ghosh A.K., Chaudhuri P., and Murthy C.A. On visualization and aggregation of nearest neighbor classifiers. IEEE Trans. Pattern Anal. Mach. Intell. 27(10):1592–1602, 2005.

7. Han J. and Kamber M. Data Mining, Concepts and Techniques (2nd ed.). Elsevier, Amsterdam, 2006.
8. Hastie T., Tibshirami R., and Friedman J. The Elements of Statistical Learning: Data Mining, Inference and Prediction. Springer Series in Statistics. Springer, New York, 2001.
9. Kriegel H.-P., Pryakhin A., and Schubert M. Multi-represented kNN-classification for large class sets. In Proc. 10th Int. Conf. on Database Systems for Advanced Applications, 2005, pp. 511–522.
10. Li Y., Yang J., and Han J. 1Continuous K-nearest neighbor search for moving objects. In Proc. 16th Int. Conf. on Scientific and Statistical Database Management, 2004, pp. 123–126.
11. Shibata T., Kato T., and Wada T. K-D. Decision tree: an accelerated and memory efficient nearest neighbor classifier. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 641–644.
12. Veenman C.J. and Reinders M.J.T. The nearest subclass classifier: a compromise between the nearest mean and nearest neighbor classifier. IEEE Trans. Pattern Anal. Mach. Intell., 27(9): 1417–1429, 2005.

Nearest Neighbor Query

Dimitris Papadias
Hong Kong University of Science and Technology,
Hong Kong, China

Synonyms

NN query; *k*NN query

Definition

Given a dataset *P* and a query point *q*, a *k nearest neighbor* (*k*NN) query returns the *k* closest data points $p_1,p_2,...,p_k$ to *q*. For any point $p \in P-\{p_1,p_2,...,p_k\}$: $dist(p_i, q) \leq dist(p, q)\ \forall\ 1 \leq i \leq k$, where *dist* depends on the underlying distance metric. The usual metric is the L_x norm, or formally, given two points *q*, *p* whose coordinates on the *i*-th dimension ($1 \leq i \leq m$) are q_i and p_i respectively, their L_x distance is:

$$L_x(p,\ q) = (\Sigma_{i=1\sim m}|p_i - q_i|^x)^{1/x},\ \text{for}\ x \neq \infty,$$

$$L_x(p,\ q) = \max\ _{i=1\sim m}(|p_i - q_i|),\ \text{for}\ x = \infty.$$

Most work on spatial and spatio-temporal databases focuses on Euclidean distance (i.e., L_2), but alternative definitions have been used in various domains (e.g., road networks, time series).

Key Points

Nearest neighbor search constitutes an important component for several tasks such as clustering, outlier detection, time series analysis, and image and document retrieval. For instance, finding the most similar series, image, or document to a given input is a NN query in a corresponding data space defined by the features of interest. Unlike spatial and spatio-temporal databases that usually involve 2–3 dimensions, these applications may lead to high-dimensional spaces. In order to avoid the high cost of distance computations in such cases, several methods follow a multi-step framework for processing NN queries [3]: (i) a dimensionality reduction technique is used to decrease the number of dimensions, (ii) the low dimensional data are indexed, (iii) the index is used to efficiently retrieve a candidate NN (in low dimensional space), (iv) the actual distance of the candidate (in the original space) is computed, and (v) steps (iii) and (iv) are repeated until no other candidate can lead to a better solution than the one already discovered.

In addition to conventional NN search, several alternative types of nearest neighbor queries have been proposed in the database literature. Given a multi-dimensional dataset *P* and a point *q*, a *reverse nearest neighbor* query retrieves all the points $p \in P$ that have *q* as their NN (The nearest neighbor relationship is not symmetric; i.e., the fact that *q* is the NN of *p* does not necessarily imply that *p* is the NN of *q*.) [1]. Given a set *P* of data points and a set *Q* of query points, an *aggregate nearest neighbor* query returns the data point *p* with the minimum *aggregate distance* [2]. The *aggregate distance* between a data point *p* and $Q = \{q_1,...,q_n\}$ is defined as $f(dist(p,q_1),...,dist(p, q_n))$. If, for instance, $f = sum$, the corresponding query reports the data point that minimizes the total distance from all query points.

Cross-references

▶ Nearest Neighbor Query in Spatio-temporal Databases
▶ Reverse Nearest Neighbor Query

Recommended Reading

1. Korn F. and Muthukrishnan S. Influence sets based on reverse nearest neighbor queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 201–212.
2. Papadias D., Tao Y., Mouratidis K., and Hui K. Aggregate nearest neighbor queries in spatial databases. ACM Trans. Database Sys., 30(2):529–576, 2005.
3. Seidl T. and Kriegel H-P. Optimal multi-step k-nearest neighbor search. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp 154–165.

Nearest Neighbor Query in Spatio-temporal Databases

Dimitris Papadias
Hong Kong University of Science and Technology,
Hong Kong, China

Synonyms

NN query; NN search

Definition

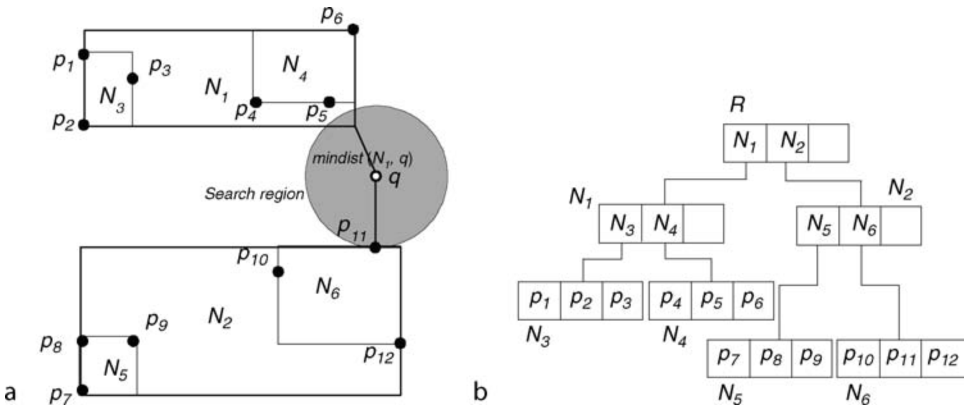
Given a set of points P in a multi-dimensional space, the nearest neighbor (NN) of a query point q is the point in P that is closest to q. Similarly, the k nearest neighbor (kNN) set of q consists of the k points in P with the smallest distances from q. In spatial and spatio-temporal databases, the distance is usually defined according to the Euclidean metric, and the dataset P is disk-resident. Query algorithms aim at minimizing the processing cost. Other optimization criteria in the case of moving objects (or queries) include the network latency, or the number of queries required for keeping the results up-to-date.

Historical Background

Nearest neighbor (NN) search is one of the oldest problems in computer science. Several algorithms and theoretical performance bounds have been devised for exact and approximate processing in main memory [1]. In spatial databases, existing algorithms assume that P is indexed by a spatial access method (usually an *R-tree* [2]) and utilize some pruning bounds to restrict the search space. Figure 1 shows an *R-tree* for point set $P = \{p_1,p_2,\dots,p_{12}\}$ with a capacity of three entries per node (typically, the capacity is in the order of hundreds). Points that are close in space (e.g., p_1, p_2, p_3) are clustered in the same leaf node (N_3). Nodes are then recursively grouped together with the same principle until the top level, which consists of a single root. Given a node N and a query point q, the *mindist* (N,q) corresponds to the closest possible distance between q and any point in the subtree of node N.

The first NN algorithm for *R-trees* [8] searches the tree in a *depth-first* (DF) manner, recursively visiting the node with the minimum *mindist* from q, e.g., in Fig. 1, DF accesses the root, followed by N_1 and N_4, where the first potential nearest neighbor is found (p_5). Note that p_5 is not the actual NN (it is p_{11}), and the search continues. During backtracking to the upper level (node N_1), the algorithm prunes entries whose *mindist* is equal to or larger than the distance (*best_dist*) of the nearest neighbor already retrieved. In the example of Fig. 1, after discovering p_5, (i) *best_dist* is set to $dist(p_5,q)$, (ii) DF backtracks to the root level (without visiting N_3), and (iii) it follows the path N_2, N_6 where the actual NN p_{11} is found. DF can be easily extended for the retrieval of $k > 1$ nearest neighbors: the k points discovered so far with the minimum overall distances are maintained in an ordered list of k pairs $<p, dist(p,q)>$ (sorted on ascending $dist(p,q)$ order) and *best_dist* equals the distance of the k-th NN. Whenever a better neighbor is found, it is inserted in the list, the last element is removed, and the value of *best_dist* is updated.

The DF algorithm is sub-optimal, i.e., it accesses more nodes than necessary. Specifically, an optimal algorithm should visit only nodes intersecting the

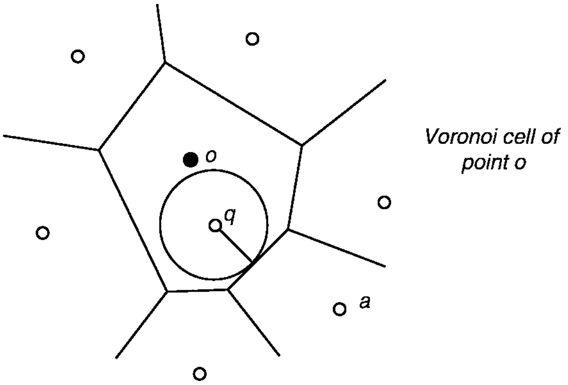


Nearest Neighbor Query in Spatio-temporal Databases. Figure 1. Example of an R-tree and a point NN query.

search region, i.e., a circle centered at the query point q with radius equal to the distance between q and its nearest neighbor [4]. In Fig. 1, for instance, an optimal algorithm should visit only the root, N_1, N_2, and N_6, whereas DF also visits N_4. The *best-first* (BF) algorithm of [5] achieves the optimal I/O performance by maintaining a heap H with the entries visited so far, sorted by their *mindist*. As with DF, BF starts from the root, and inserts all the entries into H (together with their *mindist*), e.g., in Fig. 1, $H = \{<N_1, mindist(N_1,q)>, <N_2, mindist(N_2,q)>\}$. Then, at each step, BF visits the node in H with the smallest *mindist*. Continuing the example, the algorithm retrieves the content of N_1 and inserts all its entries in H, after which $H = \{<N_2, mindist(N_2,q)>, <N_4, mindist(N_4,q)>, <N_3, mindist(N_3,q)>\}$. The next two nodes accessed are N_2 and N_6 (inserted in H after visiting N_2), in which p_{11} is discovered as the current NN. At this time, the algorithm terminates (with p_{11} as the final result) since the next entry (N_4) in H is farther (from q) than p_{11}. Similarly to DF, BF can be easily extended to *k*NN queries ($k > 1$). In addition, BF is *incremental*, i.e., it can output the nearest neighbors in ascending order of their distance to the query without a pre-defined termination condition.

Foundations

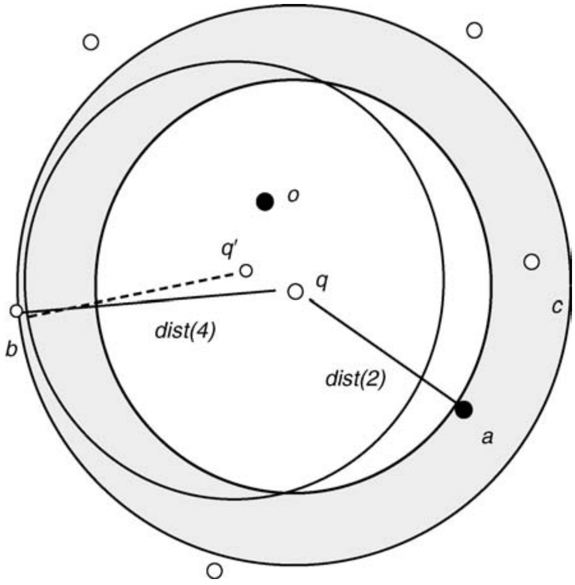
In dynamic environments, the result of a conventional NN query may be immediately invalidated due to the query or data movement. Several techniques augment the result with some additional information regarding its validity. For instance, Zheng and Lee [14] assume an architecture where moving clients (e.g., mobile devices) send their queries to a server. The server precomputes and stores in an *R-tree* the *Voronoi diagram* of the (static) dataset. When a NN query q arrives at the server, the *Voronoi diagram* is used to efficiently compute its nearest neighbor; i.e., the point (o in Fig. 2), whose Voronoi cell covers q. In addition to the result, the server sends back to the client its *validity time T*, which is a conservative approximation assuming that the query speed is below a maximum value. In particular, T is the time that q will cross the closest boundary of the Voronoi cell of object o (in which case point a will become the nearest neighbor). Zhang et al. [13] propose the concept of *location-based* queries that return the *validity region* around the query point, where the result remains the same. For instance, in Fig. 2, a location-based query q will return object o and its Voronoi cell, which is computed on-the-fly using an *R-tree* on the data points.



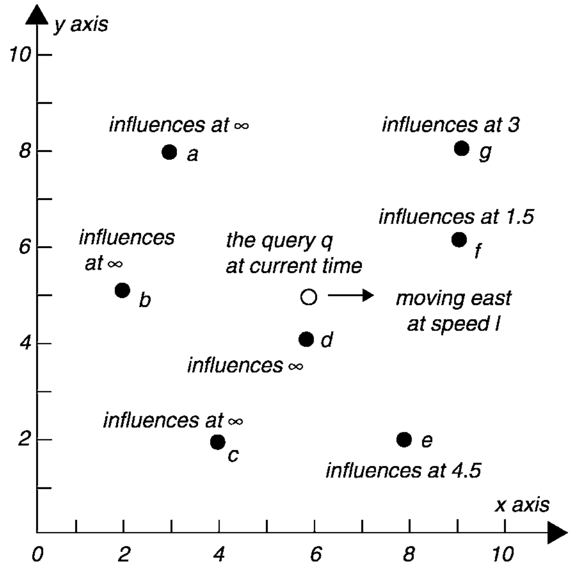
Nearest Neighbor Query in Spatio-temporal Databases. Figure 2. Example of [6].

For the same settings (moving query - static data objects), Song and Roussopoulos [10] reduce the number of queries required to keep the result up-to-date by introducing redundancy. In particular, when a *k*NN query arrives, the server returns to the client a number $m > k$ of neighbors. Let *dist*(k) and *dist*(m) be the distances of the kth and mth nearest neighbor from the query point q. If the client re-issues the query at a new location q', it can be proven that the new k nearest neighbors will be among the m objects of the first query, provided that $2{\cdot}dist(q',q) \leq dist(m)\text{-}dist(k)$. Figure 3 shows an example for a 2NN query at location q, where the server returns four results o, a, b and c (the two nearest neighbors are o and a). When the client moves to a nearby location q', the 2 NN are o and b. If $2{\cdot}dist(q,q') \leq dist(4)\text{-}dist(2)$, the client can determine this by computing the new distances (with respect to q') of the four objects, without having to issue a new query to the server.

Tao and Papadias [11] propose *time-parameterized* (TP) queries, assuming that the clients move with linear and known velocities. In addition to the current result R, the output of a TP query contains its *validity period T* and the *next change C* of the result (that will occur at the end of the validity period). Given the additional information (C and T), the client only needs to issue another TP query after the expiry of the current result. Figure 4 shows a TP NN, where the query point q is moving east with speed 1. Point d is the current nearest neighbor of q. The *influence time* $T_{INF}(p, q)$ of an object p is the time that p starts to get closer to q than the current nearest neighbor. For



Nearest Neighbor Query in Spatio-temporal Databases. Figure 3. Example of [8].



Nearest Neighbor Query in Spatio-temporal Databases. Figure 4. Example of TP NN query.

example, $T_{INF}(g,q) = 3$, because at this time g will come closer to q than d. The expiry time of the current result is the minimum $T_{INF}(o,q)$ of all objects, i.e., in Fig. 4, $T = 1.5$, at which point f will replace d as the NN of q. Based on the above observations, a TP NN query is processed in two steps: (i) a conventional algorithm (e.g., [5,8]) retrieves the NN at the current time, and (ii) a second pass computes the time-parameterized component (i.e., C and T) by applying again NN search and treating $T_{INF}(o,q)$ as the distance metric: the goal is to find the objects (C) with the minimum T_{INF}.

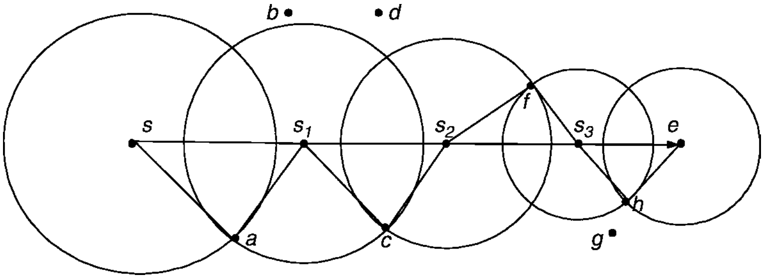
A *continuous* nearest neighbor (CNN) query [3,11] retrieves the nearest neighbor (NN) of every point in a line segment $q = [s, e]$. In particular, the result contains a set of $<R,T>$ tuples, where R (for result) is a point of P, and T is the interval during which R is the NN of q. As an example consider Fig. 5, where $P = \{a,b,c,d,f,g,h\}$. The output of the query is $\{<a, [s,s_1]>, <c, [s_1,s_2]>, <f, [s_2,s_3]>, <h, [s_3,e]>\}$, meaning that point a is the NN for interval $[s,s_1]$; then at s_1, point c becomes the NN etc. The points of the query segment (i.e., s_1, s_2, s_3) where there is a change of neighborhood are called *split points*. CNN algorithms use DF or BF traversal on *R-trees* to visit nodes and data points according to their proximity to the query segment. Visited data points introduce new split points, which are used to prune the search space. For instance, in Fig. 5, if a, c, f and h have already been discovered, every node and data point (b, d, g) outside a circle defined by a split point (center) and its NN (radius) can be eliminated since they cannot affect the result.

A *predictive* NN query retrieves the expected nearest neighbor of a (static or moving) query point (e.g., 30 seconds from now) based on the current motion patterns. Assuming that data points move linearly, they can be indexed by a TPR-tree [9]. The TPR-tree is similar to the *R-tree*, but takes into account both location and velocity in order to group data objects into nodes. Furthermore, each node is assigned a velocity vector so that its extent continuously encloses all the objects inside (i.e., a moving node that may grow with time). A predictive query is answered in the same way as in the *R-tree* (e.g., by adaptations of [5,8]), except that the node extends at the (future) query time are dynamically computed (using the current location and velocity vector of the node). The concepts of TP and continuous queries also apply in this case. Tao et al. [12] propose an architecture and index for processing queries on objects moving with arbitrary motion patterns, unknown in advance.

Key Applications

Geographic Information Systems

Nearest neighbor search is one of the most common query types. Efficient algorithms are important for



Nearest Neighbor Query in Spatio-temporal Databases. Figure 5. Example CNN query.

dealing with the large and ever increasing amount of spatial data in several GIS applications.

Location-based Services

Advanced NN algorithms will provide the means for enhanced location-based services based on the proximity of mobile clients to potential facilities of interest.

Multi-criteria Decision Making

A number of decision support tasks can be modeled as nearest neighbor search. For instance, news WWW-sites usually recommend to users the articles that are most similar to their previous choices. Nearest neighbor algorithms have also been used to process skyline queries.

Future Directions

All the above techniques take as input a single query, and report its nearest neighbor set at the current time, possibly with some validity information (e.g., expiration time, Voronoi cell), or generate future results based on predictive features (e.g., velocity vectors of queries or data objects). On the other hand, *continuous monitoring of spatial queries* [6] (i) involves multiple, long-running, queries (from geographically distributed clients), (ii) is concerned with both computing *and* keeping the results up to date, (iii) usually assumes main-memory processing to provide fast answers in an on-line fashion, and (iv) attempts to minimize factors such as the CPU or communication cost (as opposed to I/O overhead).

Another interesting problem concerns NN search in non-Euclidean spaces. For instance, in road networks the distance between two points can be defined as the length of the shortest path connecting them, or by the minimum time it takes to travel between them. In either case, the problem requires different algorithmic solutions [7].

Experimental Results

In general, for every presented method, there is an accompanying experimental evaluation in the corresponding reference. [5] compares BF and DF traversal for conventional NN search. [9] proposes cost models for TP NN and CNN queries and evaluates their accuracy, as well as the relative performance of the two query types for static (indexed by *R-trees*) and dynamic (indexed by *TPR-trees*) objects.

Data Sets

A large collection of real datasets, commonly used for experiments, can be found at:

http://www.rtreeportal.org/

URL to Code

R-tree portal (see above) contains the code for most common spatial and spatio-temporal indexes, as well as data generators and several useful links for researchers and practitioners in spatio-temporal databases.

Cross-references

▶ Continuous Monitoring of Spatial Queries
▶ Nearest Neighbor Query
▶ Rtree
▶ Voronoi Diagrams

Recommended Reading

1. Arya S., Mount D., Netanyahu N., Silverman R., and Wu A. An optimal algorithm for approximate nearest neighbor searching. J. ACM, 45(6):891–923, 1998.
2. Beckmann N., Kriegel H., Schneider R., and Seeger B. The R*-tree: an efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 322–331.
3. Benetis R., Jensen C., Karciauskas G., and Saltenis S. Nearest neighbor and reverse nearest neighbor queries for moving objects. VLDB J., 15(3):229–249, 2006.

4. Bohm C. A cost model for query processing in high dimensional data spaces. ACM Trans. Database Sys., 25 (2):129–178, 2000.
5. Hjaltason G. and Samet H. Distance browsing in spatial databases. ACM Trans. Database Sys., 24(2):265–318, 1999.
6. Mouratidis K., Hadjieleftheriou M., and Papadias D. Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 634–645.
7. Papadias D., Zhang J., Mamoulis N., and Tao Y. Query processing in spatial network databases. In Proc. of the 29th Conf. on Very Large Databases, 2003, pp. 790–801.
8. Roussopoulos N., Kelly S., and Vincent F. Nearest Neighbor Queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 71–79.
9. Saltenis S., Jensen C., Leutenegger S., and Lopez M. Indexing the positions of continuously moving objects. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 331–342.
10. Song Z., and Roussopoulos N. K-nearest neighbor search for moving query point. In Proc. 7th Int. Symp. Advances in Spatial and Temporal Databases, 2001, pp. 79–96.
11. Tao Y. and Papadias D. Spatial queries in dynamic environments. ACM Trans. Database Sys., 28(2):101–139, 2003.
12. Tao Y., Faloutsos C., Papadias D., and Liu B. Prediction and indexing of moving objects with unknown motion patterns. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 611–622.
13. Zhang J., Zhu M., Papadias D., Tao Y., and Lee D. Location-based spatial queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 443–454.
14. Zheng B., and Lee D. Semantic caching in location-dependent query processing. In Proc. 7th Int. Symp. Advances in Spatial and Temporal Databases, 2001, pp. 97–116.

Negative Dictionary

► Stoplists

Nested Loop Join

Jingren Zhou
Microsoft Research, Redmond, WA, USA

Synonyms

Nested loop join; Loop join

Definition

The nested loop join is a common join algorithm in database systems using two nested loops. The algorithm starts with reading the *outer* relation *R*, and for each tuple $\mathcal{R} \in R$, the *inner* relation *S* is checked and matching tuples are added to the result.

```
Algorithm 1: Nested Loop Join: $R \bowtie_{pred(r,\ s)} S$
  foreach R ∈ R do
      foreach S ∈ S do
          if pred (R.r, S.s) then
              add {R, S} to result
          end
      end
  end
```

Key Points

One advantage of the nested loop join is that it can handle any kind of join predicates, unlike the sort-merge join and the hash join which mainly deal with an equality join predicate. An improvement over the simple nested loop join is the block nested loop join which effectively utilizes buffer pages and reduces disk I/Os.

Block Nested Loop Join

Suppose that the memory can hold *B* buffer pages. If there is enough memory to hold the smaller relation, say *R*, with at least two extra buffer pages left, the optimal approach is to read in the smaller relation *R* and to use one extra page as an input buffer to read in the larger relation *S* and the other extra buffer page as an output buffer.

If there is not enough memory to hold the smaller relation, the best approach is to break the outer relation *R* into *blocks* of $B - 2$ pages each and scan the whole inner relation *S* for each block of *R*. As described before, one extra page is used as an input buffer and the other as an output buffer. In this case, the outer relation *R* is scanned only once while the inner relation *S* is scanned multiple times.

Cross-references

► Parallel Join Algorithms
► Evaluation of Relational Operators

Recommended Reading

1. Mishra P. and Eich M.H. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, 1992.

Nested Transaction Models

George Karabatis
University of Maryland, Baltimore Country (UMBC),
Baltimore, MD, USA

Definition

A *nested transaction model* as proposed by Moss is a generalization of the flat transaction model that allows nesting. A nested transaction forms a tree of transactions with the root being called a *top-level transaction* and all other nodes called *nested transactions (subtransactions).* Transactions having no subtransactions are called *leaf* transactions. Transactions with subtransactions are called *parents (ancestors)* and their subtransactions are called *children (descendants).*

A subtransaction can commit or rollback by itself. However, the effects of the commit cannot take place unless the parent transaction also commits. Therefore, in order for any subtransaction to commit, the top-level transaction must commit. If a subtransaction aborts, all its children subtransactions (forming a subtree) are forced to abort even if they committed locally.

Historical Background

Nested transactions were introduced by Moss in 1981 [8] to overcome some of the limitations of the flat transaction model, as they allow a finer level of control in a transaction. For example, a failed operation in a flat transaction causes the entire transaction to be rolled back. On the contrary, a failed operation in a part of a nested transaction may be acceptable and the entire transaction may be allowed to commit, even if some of its operations failed. As an example, a trip consisting of a flight reservation, hotel accommodation and car rental can be implemented as a nested transaction with three subtransactions. If some of its subtransactions commit (e.g., flight and hotel), and some fail (e.g., car rental) the outcome may still be acceptable and allow the nested transaction to commit with partial results.

Nested transactions are based on the underlying concept of *spheres of control* developed by Bjork and Davies in 1973 [1,3]. Davies describes the spheres of control in more detail in [2]; however, he portrays overall semantics of the spheres which convey a more abstract concept than nested transactions. Moss provides implementation details of nested transactions for a distributed environment. Prior to that, Reed in his dissertation [9] also describes an implementation of nested transactions in a distributed environment and one can see similarities and differences between the two approaches. For example, Moss uses locking whereas Reed uses timestamps. For a detailed comparison of the two approaches see [8].

Foundations

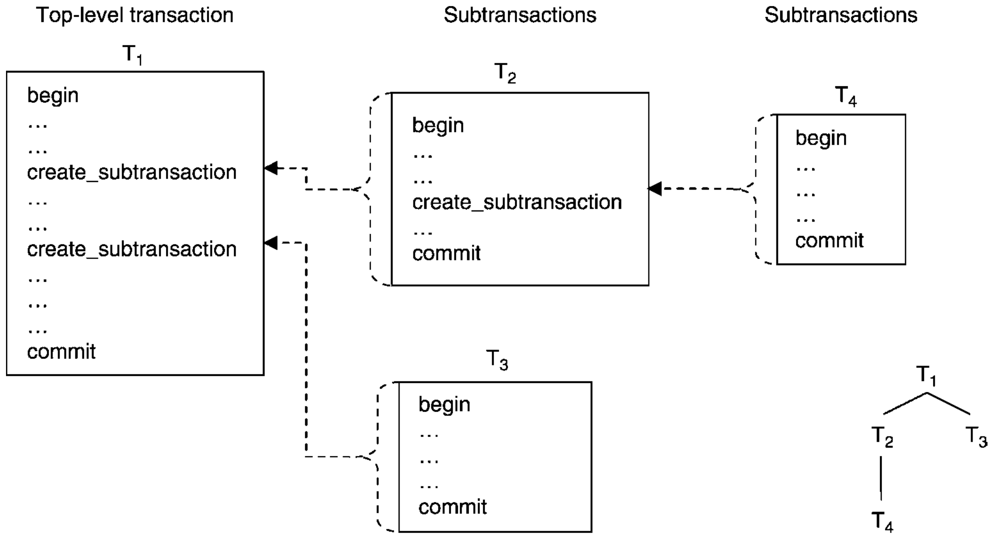
Structure of a Nested Transaction

The structure of a nested transaction is depicted by a transaction tree: its root represents the *top-level* transaction and its children represent *subtransactions* each one corresponding to a transactional unit. A subtransaction may be either a simple (*leaf)* transaction or a nested transaction, recursively expanding the structure to a hierarchy with multiple levels of transactions.

Leaf transactions are very similar to traditional ACID transactions, but they do not preserve the durability property of ACID transactions as will be explained in the Commit/Abort rules. Leaf transactions are the ones that actually manipulate the data in the database and perform the work. Intermediate level subtransactions and the top-level transaction operate on a higher level of abstraction: their purpose is to control when to create a new subtransaction. This hierarchy corresponds to the notion of nested spheres of control [3]. Figure 1 illustrates an example of a nested transaction and its corresponding transaction tree. It consists of a top-level transaction T_1 with two subtransactions T_2 and T_3; subtransaction T_2 is itself a nested transaction invoking subtransaction $T_{4.}$ In this example, subtransactions T_3 and T_4 are leaf transactions performing the actual work. When multiple such nested transactions coexist their trees form a forest.

Synchronization of Nested Transactions

Subtransactions of nested transactions can execute concurrently and for proper synchronization Moss devised the following locking protocol: A subtransaction can either *hold* a lock or *retain* a lock. When a lock is held the transaction has exclusive rights to the object. After a subtransaction commits, its held or retained locks are *inherited* by the parent subtransaction. These inherited locks are not available to other subtransactions outside the subtree (sphere of control) of the holder; however, subtransactions within the subtree (descendants) can acquire a retained lock.



Nested Transaction Models. Figure 1. Structure of an example nested transaction; to the lower right is a depiction of the corresponding nested transaction tree.

The Locking Rules of nested transactions slightly paraphrased from Moss [8] are the following:

1. A transaction may hold a lock in write mode if no other transaction holds the lock (in any mode) and all retainers of the lock are ancestors of the requester.
2. A transaction may hold a lock in read mode if no other transaction holds the lock in write mode and all retainers of write locks are ancestors of the requester.
3. When a transaction commits, its parent inherits all locks the descendants held or retained in the same mode as the child held or retained them.
4. When a transaction aborts, all locks it holds or retains are released. Ancestors of the aborted transaction who were retaining the lock they continue to do so in the same mode as before the abort.

Commit/Abort of Nested Transactions

Moss describes a set of rules (state restoration algorithm in [8]) that dictate the behavior of nested transactions at commit/abort time. A subtransaction can unilaterally commit or abort independently of the others preserving atomicity, consistency and isolation from the ACID properties. The rules reworded from [8] are:

1. All children of a subtransaction must be resolved (committed or aborted) before it commits itself.
2. A subtransaction that unilaterally commits, first reaches its local commit point. However, it cannot finally commit until its parent transaction commits. Consequently, a subtransaction will commit only if it commits locally and all its ancestors commit including the top-level transaction.
3. If one of the ancestors of a subtransaction aborts, then the subtransaction itself will have to abort. In other words, the abort of a subtransaction causes the abort of all its children. Even if the subtransaction commits, its actions must be undone due to the abort of an ancestor. This is the reason that nested subtransactions violate the durability of ACID properties.

A nested transaction at the top level (i.e., the entire nested transaction as a whole) does observe durability. Therefore, nested transactions as a whole obey the ACID properties, despite the fact that individual nested subtransactions may violate the durability property.

Advantages of Nested Transactions

1. Nested transactions allow for a simple composition of subtransactions improving modularity of the overall structure.
2. The concurrent execution of subtransactions that follow the prescribed rules allows for enhanced concurrency while preserving consistency. This enhanced concurrency occurs because transactions can create subtransactions which execute concurrently.
3. The success or failure of a subtransaction is independent of the success of its siblings. When a subtransaction fails, its parent may try to execute it again, or ignore the failure, depending on the

situation. Therefore, failures that are contained within the scope of a subtransaction may be tolerated and do not necessarily cause the failure of the entire nested transaction.

Gray and Reuter present a mechanism to emulate nested transactions using database savepoints [6]. This approach emulates recovery as accurately as prescribed in the nested transaction model; however, it emulates locking not as flexibly as described in nested transactions.

Key Applications

Several systems were influenced by the nested transactions model, and as such they either incorporated or adapted its concept. The implementation of nested transactions by Moss, and especially the locking mechanism was used in the Argus system in which Moss worked with Liskov [7]. Camelot is another system that used nested transactions. It is a transaction processing system developed at Carnegie Mellon University using the Avalon programming language [5]. Camelot was a precursor to the Encina distributed transaction processing monitor which uses the Transactional-C language. Encina was commercialized under the name Transarc Corporation, which was acquired by IBM in 1994.

Nested transactions have also been implemented in Berkeley DB – a database library with bindings for several programming languages, which originated at the University of California – Berkeley, distributed by Sleepycat Software, which was acquired by Oracle Corporation in 2006.

Several researchers have also proposed systems with transaction processing components that are designed for application domains such as manufacturing, CSCW, software engineering, etc. where the ACID properties may be considered too strict; therefore, relaxed or advanced transaction models have been proposed to address the specific requirements that exist in these domains. Several of these transaction models appear in a book by Elmagarmid [4].

A notable example of an extension to the concept of nested transactions is the *multi-level transaction model* and its closely related *open nested transaction model* by Weikum and Schek [10,12]. The multi-level transaction model is represented by a transaction tree like in nested transactions; however, unlike nested transactions the transaction tree is balanced. Multi-level concurrency control takes advantage of the semantics of conflicting operations in the same level of the tree, such as commutativity, in order to enhance concurrency. In open nested transactions, the subtransactions are not restricted to have the same depth and are allowed to commit their results without waiting for their ancestors to commit [11].

Cross-references

- ▶ ACID Properties
- ▶ Concurrency Control – Traditional Approaches
- ▶ Data Conflicts
- ▶ Database Management System
- ▶ Distributed DBMS
- ▶ Distributed Transaction Management
- ▶ Extended Transaction Models
- ▶ Multilevel Transactions and Object-Model Transactions
- ▶ Open Nested Transaction Models
- ▶ Serializability
- ▶ TP Monitor
- ▶ Transaction
- ▶ Transaction Management
- ▶ Transaction Manager

Recommended Reading

1. Bjork L.A. Recovery scenario for a DB/DC system. In Proc. ACM Annual Conference, 1973, pp. 142–146.
2. Davies C.T. Data processing spheres of control. IBM Syst J., 17:179–198, 1978.
3. Davies C.T. Recovery semantics for a DB/DC system, In Proc. ACM Annual Conference, 1973, pp. 136–141.
4. Elmagarmid A.K. Database Transaction models for advanced applications. Morgan Kaufmann Publishers Inc., CA, 1992.
5. Eppinger J.L., Mummert L.B., and Spector A.Z. Camelot and avalon: A Distributed Transaction Facility. Morgan Kaufmann Publishers Inc., CA, 1991.
6. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. 1st edn. Morgan Kaufmann Publishers Inc., CA, 1992.
7. Liskov B. Distributed programming in Argus. Commun ACM., 31:300–312, 1988.
8. Moss E.B. Nested Transactions: An Approach to Reliable Distributed Computing. Technical Report. PhD Thesis. UMI Order Number: TR-260: Massachusetts Institute of Technology, 1981, pp. 178.
9. Reed D.P. Naming and Synchronization in a Distributed Computer System. Technical Report. PhD Thesis, UMI Order Number: TR-205: Massachusetts Institute of Technology, 1978, pp. 181.
10. Weikum G. Principles and Realization Strategies of Multilevel Transaction Management. ACM Trans Database Sys., 16:132–180, 1991.
11. Weikum G. and Schek H.-J. Concepts and Applications of Multilevel Transactions and Open Nested Transactions. In Database Transaction Models for Advanced Applications. Morgan Kaufmann Publishers Inc., 1992, pp. 515–553.

12. Weikum G. and Schek H.-J. Multi-level Transactions and open nested transactions. Q. Bull. IEEE TC on Data Engineering, 14:60–66, 1991.

.NET Remoting

Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA

Synonyms

Subsumed by windows communication framework

Definition

.NET Remoting [1,2] is a Microsoft technology comprising a set of application programming interfaces (APIs) for interprocess communication available within the .NET Framework. .NET Remoting serves as the distribution middleware layer, which enables communication between distributed applications that can choose their own transport protocol, serialization format, an application model and different schemes to manage object lifetimes.

Key Points

Central to the .NET Remoting architecture is the notion of a *remotable object* and a brokering capability that enables communication between *application domains*. An application domain in the .NET framework provides a type-safe and secure unit of execution and isolation. .NET Remoting makes it feasible for multiple application domains to communicate with each other without concern for whether they are hosted within the same operating system process, or across multiple different processes within the same node or across a network.

A remotable object is an object which is declared to be serializable remotable objects can cross application domain boundaries. Remotable Objects are published via an *Activation URL*. Client applications invoking services of a remotable object do so via proxies.

An important feature of .NET Remoting is its ability to allow applications to define their own serialization format, the choice of transport protocol used, and application model used by the communicating entities. By virtue of using the Common Language Runtime (CLR), .NET Remoting can provide interoperability across multiple managed languages. But its portability is restricted to Microsoft platforms only.

Cross-references

- ▶ Client-server architecture
- ▶ CORBA
- ▶ DCE
- ▶ DCOM
- ▶ J2EE Middleware
- ▶ Java RMI
- ▶ Request Broker
- ▶ SOAP

Recommended Reading

1. .NET Framework Remoting Architecture, http://msdn2.microsoft.com/en-us/library/2e7z38xb.aspx.
2. Microsoft Technologies, .NET Remoting: Core Protocol Specification, [MS-NRTP] v20080207, February 2008.

Network Attached Secure Device

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Synonyms

NASD; Object-based Storage Device; OSD

Definition

A Network Attached Secure Device (often abbreviated to NASD) is a networked storage device which offers object-level storage accesses. NASD was first explored in a research project of the same name, which started at Carnegie Mellon University in 1995. NASD is now more often called Object based Storage Device (shortened to OSD).

Key Points

NASD is an approach of decoupling only low-level functions of file systems from NAS devices. A NASD manages only stored objects, each of which is identified with a unique file descriptor, and their additional information such as metadata and attributes, whereas a central policy server manages higher-level information which is used for name space management and authorization. A bunch of NASDs, with the assistance of the central policy server, can together work as a file store. This may look similar to NAS at a glance, but the difference is that most commands and data are directly transferred between clients and NASDs. Conventional storage subsystems have a number of storage devices under a storage controller, which is thus likely to become a performance bottleneck. NASD has benefits of scaling aggregate bandwidth by spreading partial functions over a number of disk processors.

Object-level storage abstraction, introduced by NASD, is recognized to be the third alternative in addition to block-level storage abstraction typically seen in SAN storage environments and file-level storage abstraction in NAS storage environments. Content-Addressable Storage is another example which offers object-level storage accesses and is now widely deployed in enterprise systems.

The idea of NASD has been standardized as ANSI T10 SCSI OSD, which specifies an extension of the SCSI protocol for clients and devices to exchange objects and their related information. Thus, OSD is sometimes seen as a promising infrastructure of intelligent storage devices in which more intelligence will be incorporated.

Cross-references

▶ Active Disks

▶ Intelligent Storage Systems

Recommended Reading

1. ANSI. Information Technology - SCSI Object-Based Storage Device Commands (OSD). Standard ANSI/INCITS 400–2004. 2004.
2. Gibson G.A. and Van Meter R. Network attached storage architecture. Commun. ACM, 43(11):37–45, 2000.
3. Gibson G.A., Nagle D.F., Amiri K., Chang F.W., Feinberg E.M., Gobioff H., Chen Lee, Ozceri B., Riedel E., Rochberg D., and Zelenka J. File server scaling with network-attached secure disks. In Proc. 1997 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Comp. Syst., 1997, pp. 272–284.

Network Attached Storage

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Synonyms

NAS

Definition

Network attached storage is a storage device which is connected to a network and provides file access services. Network attached storage is often abbreviated to NAS. Although the term NAS refers originally to such a storage device, when the term is used in the context of explaining and comparing storage network architectures it sometimes refers to a storage network architecture in which NAS devices are mainly implemented.

Key Points

A NAS device is basically comprised of disk drives which store files and controllers which export access services to the files. A file sever which runs network file system (NFS) and/or common internet file system (CIFS) to export file sharing services is a type of NAS implementation, but recent NAS products are sometimes implemented using dedicated hardware and software to increase reliability and performance. A diskless NAS device which contains only controllers is sometimes referred to as a NAS gateway or a NAS head. A NAS gateway/head has two types of network ports: one is connected to disk storage devices over a SAN and the other is to NAS clients over IP networks. The clients are thus provided with access services towards files stored in the storage devices. That is, a NAS gateway/head can be seen as a service bridge between SAN and NAS systems.

Cross-references

▶ Direct Attached Storage

▶ Storage Area Network

▶ Storage Network Architectures

Recommended Reading

1. Storage Network Industry Association. The Dictionary of Storage Networking Terminology. Available at: http://www.snia.org/.
2. Troppens U., Erkens R., and Müller W. Storage Networks Explained. Wiley, New York, 2004.

Network Data Model

Jean-Luc Hainaut
University of Namur, Namur, Belgium

Synonyms

CODASYL data model; DBTG data model

Definition

A database management system complies with the *network data model* when the data it manages are organized as data records connected through binary relationships. Data processing is based on navigational primitives according to which records are accessed and updated one at a time, as opposed to the set orientation of the relational query languages. Its most popular variant is the CODASYL DBTG data model that was first defined in the 1971 report from the CODASYL group, and that has been implemented into several major DBMSs. They were widely used in the seventies and eighties, but most of them are still active at the present time.

Historical Background

In 1962, C. Bachman of General Electric, New-York, started the development of a data management system according to which data records were interconnected via a network of relationships that could be navigated through [2]. Called Integrated Data Store (IDS), this disk-based system quickly became popular to support the storage, the management and the exploitation of corporate data.

IDS was the main basis of the work of the CODASYL Data Base Task Group (DBTG) that published its first major report in 1971 [3,6], followed by a revision in 1973 [3,9]. This report described a general architecture for DBMSs, where the respective roles of the operating system, the DBMS and application programs were clearly identified. It also provided a precise specification of languages for data structure definition (Data Description Language or Schema DDL), for data extraction and update (Data Manipulation Language or DML) and for defining interfaces for application programs through language-dependent views of data (Sub-schema DDL).

The 1978 report [7,9] clarified the model. In particular physical specifications such as indexing structures and storage were removed from the DDL and collected into the Data Storage Description Language (DSDL), devoted to the physical schema description. In 1985, the X3H2 ANSI Database Standard committee issued standards for network database management systems, called NDL and based on the 1978 CODASYL report. However, due to the increasing dominance of the relational model these proposals have never been implemented nor updated afterwards.

Some of the most important implementations were Bull IDS/II (an upgrade of IDS), NCR DBS, Siemens UDS-1 and UDS-2, Digital DBMS-11, DBMS-10 and DBMS-20 (now distributed by Oracle Corp.), Data General DG/DBMS, Philips Phollas, Prime DBMS, Univac DMS 90 and DMS 1100 and Cullinane IDMS, a machine-independent rewriting of IDS (now distributed by Computer Associates). Other DBMSs have been developed, that follow more or less strictly the CODADYL specifications. Examples include Norsk-Data SYBAS, Burroughs DMS-2, CDC IMF, NCR IDM-9000, Cincom TOTAL and its clone HP IMAGE (which were said to define the *shallow* data model), and MDBS and Raima DbVista that both first appeared on MS-DOS PCs.

In the seventies, IBM IMS was the main competitor of CODASYL systems [11]. From the early eighties, they both had to face the increasing influence of relational DBMSs such as Oracle (from 1979) and IBM SQL/DS (from 1982 [8]). Nowadays, most CODASYL DBMSs provide an SQL interface, sometimes through an ODBC API. Though the use of CODASYL DBMSs is slowly decreasing, many large corporate databases are still managed by network DBMSs. This state of affairs will most probably last for the next decade. Network databases, as well as hierarchical databases, are most often qualified *legacy*, inasmuch as they are often expected to be replaced, sooner or later, by modern database engines.

Foundations

The presentation of the network model is based on the specifications published in the 1971 and 1973 reports, with which most CODASYL DBMSs comply.

The Languages

The data structures and the contents of a database can be created, updated and processed by means of four languages, namely the *Schema DDL* and *Sub-schema DDL*, through which the global schema and the subschemas of the database are declared, the *Data Storage Description Language* or *DSDL* (often named *DMCL*) that allows physical structures to be defined and tuned, and the *DML* through which application programs access and update the contents of the database.

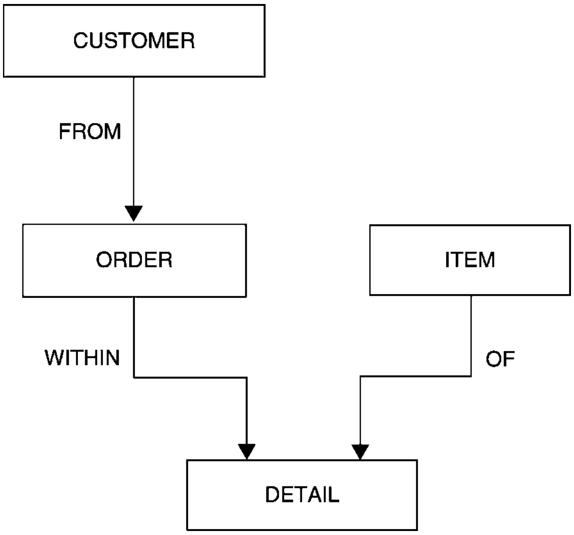
Gross Architecture of a CODASYL DBMS

The CODASYL reports define the interactions between client application programs and the database. The resulting architecture actually laid down the principles of modern DBMSs. The DBMS includes (at least) three components, namely the DDL compiler, the DML compiler and the database control system (DBCS, or simply system). The DDL compiler translates the data description code into internal tables that are stored in the database, so that they can be exploited by the DML compiler at program compile time and by the DBCS at program run time. Either the DML compiler is integrated into the host language compiler (typically COBOL) or it acts as a precompiler. It parses applications programs and replaces DML statements with calls to DBMS procedures. The DBCS receives orders from the application programs and executes them. Each program includes a *user working area* (UWA) in which data to and from the database are stored. The UWA also comprises registers that inform the program on the status of the last operations, and in particular references to the last records accessed/updated in each record type, each area, each set type and globally for the current process. These references, called *currency indicators*, represent static predefined cursors that form the basis for the navigational facilities across the data.

The Data Structures

The pictorial representation of a schema, or *Data Structure Diagram* [1], looks like a graph, where the nodes are the database *record types* and the edges are *set types*, that is, binary 1:N relationship types between record types, conventionally directed from the *one* side to the *many* side. Both record types and set types are named (Fig. 1). In this popular representation, record fields as well as various characteristics of the data structures are ignored.

Records and Record Types A *record* is the data unit exchanged between the database and the application program. A program reads one record at a time from the database and stores one record at a time in the database. Records are classified into *record types* that define their common structure and default behavior. The intended goal of a record type is to represent a real world entity type. A database key, which is a database-wide system-controlled identifier, is associated with each record, acting as an object-id.



Network Data Model. Figure 1. Diagram representation of the schema of a sample database.

Each database includes the SYSTEM record type, with one occurrence only, that can be used to define access paths across user record types through SYSTEM-owned singular set types.

The schema specifies, via the record type *location mode*, how a record is stored and how it is retrieved when storing a dependent record. This feature is both very powerful and, when used at its full power, fairly complex. Two main variants are proposed:

- location mode *calc using field-list*: the record is stored according to a hashing technique (or later through B-tree techniques) applied to the record key, composed of one or several fields of the record type (*field-list*); at run time, the default way to access a record will be through this record key;
- location mode *via set type S*: the record is physically stored as close as possible to the current record of set type *S*; later on, the default way to access a record will be through an occurrence of S identified by its *set selection* mode.

Record Fields A record type is composed of *fields*, the occurrences of which are *values*. Not surprisingly (CODASYL was also responsible for the COBOL specifications), their structure closely follow the record declaration of COBOL. The DDL offers the following field structures:

- *data item*: elementary piece of data of a certain type (arithmetic, string, implementor defined);
- *vector*: array of values of the same type; its size can be fixed or variable;
- *repeating group*: a somewhat misleading name for a possibly repeating aggregate of fields of any kind.

The fields of a record type can be atomic or compound, single-valued or multi-valued, mandatory or optional (through the null value); these three dimensions allow complex, multi-level field structures to be defined.

Sets and Set Types Basically, a CODASYL *set* is a list of records made up of a head record (the *owner* of the set) followed by zero or more other records (the *members* of the set). A *set type* S is a schema construct defined by its name and comprising one owner record type and one or more member record type(s). Considering set type S with owner type A and member type B, any A record is the owner of one and only one occurrence of S and no B record can be a member of more than one occurrence of S. In other words, a set type materializes a 1:N relationship type. The owner and the members of a set type are distinct. This limitation has been dropped in the 1978 specifications, but has been kept in most implementations (exceptions: SYBAS and MDBS). Cyclic structures are allowed provided they include at least two record types. It must be noted that a set type can include more than one member record type.

The member records of S can be ordered (first, last, sorted, application-defined). This characteristic is static and cannot be changed at run-time as in SQL. The insertion of a member record in an occurrence of S can be performed at creation time (automatic insertion mode) or later by the application program (manual insertion mode). Once a record is a member of an occurrence of S, its status is governed by the retention mode; it can be removed at will (optional), it cannot be changed (fixed) or it can be moved from an occurrence to another but cannot be removed (mandatory).

The *set [occurrence] selection* of S defines the default way an occurrence of S is determined in certain DML operations such as storing records with automatic insertion mode.

Areas An area is a named logical repository of records of one or several types. The records of a definite type can be distributed in more than one area. The intended goal is to offer a way to partition the set of the database records according to real world dimensions, such as geographic, organizational or temporal. However, since areas are mapped to physical devices, they are sometimes used to partition the data physically, e.g., across disk drives.

Schema and Sub-schemas The data structures of each database are described by a schema expressed in the DDL. Though DDL is host language independent, its syntax is reminiscent of COBOL. Views are defined by sub-schemas. Basically, a subschema is a host language dependent description of a subset of the data structures of a schema. Some slight variations are allowed, but they are less powerful than relational database capabilities. Figure 2 shows a fragment of the schema declaring the data structures of Fig. 1.

Data Manipulation

The DML allows application programs to ask the DBCS data retrieval and update services. The program accesses the data through a sub-schema that identifies the schema objects the instances of which can be retrieved and updated as well as their properties, such as the data type of each field. Exchange between the host language and the DBCS is performed via the UWA, a

```
schema name is ORDER-MANAGEMENT.

area name is DOMESTIC.
area name is FOREIGN.

record name is CUSTOMER within DOMESTIC, FOREIGN;
location mode is calc using CUST-NO duplicates not allowed.
2 CUST-NO type is character 10.
2 CUST-NAME type is character 45.
2 CUST-ADDRESS.
  4 STREET character 32.
  4 CITY character 20.
  4 PHONE type is character 15 occurs 3 times.

record name is ORDER within DOMESTIC;
location mode is via FROM set;
duplicates not allowed for ORD-NO.
  2 ORD-NO type is decimal 12.
  2 ORD-DATE type is date.

set name is FROM;
owner is CUSTOMER;
order is sorted duplicates not allowed;
member is ORDER mandatory automatic key is ascending ORD-NO;
set selection is through current of FROM.
```

Network Data Model. Figure 2. Fragment of the DDL code defining the schema of Fig. 1.

shared set of variables included in each running program. This set includes the currency indicators, the process status (e.g., the error indicators) and record variables in which the data to and from the database are temporarily stored. Many DML statements use the currency indicators as implicit arguments. Such is the case for set traversal and for record storing. Based on the currency indicators, on the *location mode* of record types and on the *set selection* option of set types, sophisticated positioning policies can be defined, leading to tight application code.

Data Retrieval The primary aim of the *find* statement is to retrieve a definite record on the basis of its position in a specified collection and to make it the current of all the *communities* which it belongs to, that is, its database, its area, its record type and each of its set types. For instance, if an *ORDER* record is successfully retrieved, it becomes the current of the database for the running program (the *current of run unit*), the current of the *DOMESTIC* area, the current of the *ORDER* record type and the current of the *FROM* and *WITHIN* set types. The variants of the *find* statement allow the program to scan the records of an area, of a record type and of the members and the owner of a set. They also provide selective access among the members of a set.

The *get* statement transfers field values from a current record in the UWA, from which they can then be processed by the program.

Data Update

A record *r* is inserted in the database as follows: first, field values of *r* are stored in the UWA, then the current of each set in which *r* will be inserted is retrieved and finally a *store* instruction is issued. The *delete* instruction applies to the current record. For this operation, the DBCS enforces a *cascade* policy: if the record to be deleted is the owner of sets whose members have a mandatory or fixed retention mode, those members are deleted as well. The *modify* statement transfers in the current of a record type the new values that have been stored in the UWA. Insertion and removal of the current of a record type is performed by *insert* and *remove* instructions. Transferring a mandatory member from a set to another cannot be carried out by merely removing then inserting the record. A special case of the *modify* statement makes such a transfer possible. Later specifications as well as some implementations propose a specific statement for this operation.

Figure 3 shows, in an arbitrary procedural pseudocode, a fragment that processes the orders of customer *C400* and another fragment that creates an *ORDER* record for the same customer.

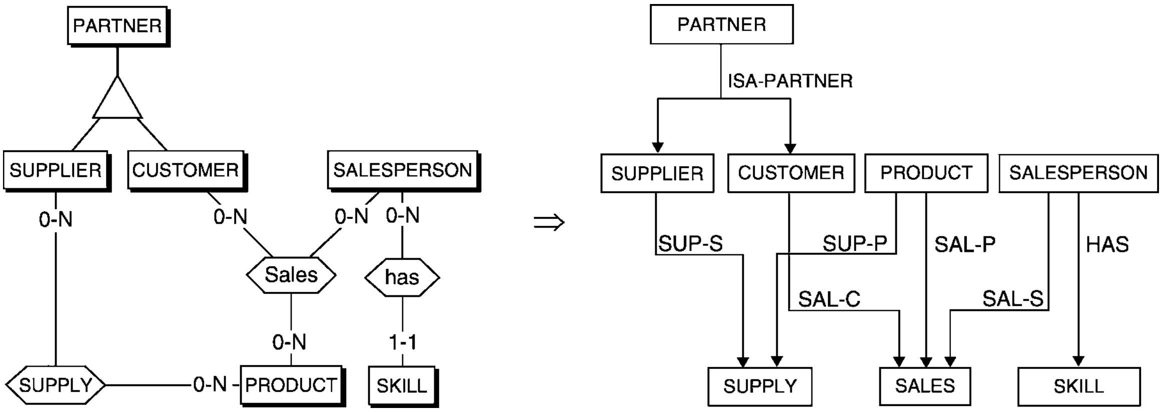
Entity-relationship to Network Mapping

Among the many DBMS data models that have been proposed since the late sixties, the network model is probably the closest to the Entity-Relationship model [5]. As a consequence, network database schemas tend to be more readable than those expressed in any other DBMS data model, at least for simple schemas. Each entity type is represented by a record type, each attribute by a field and each simple relationship type by a set type. Considering modern conceptual formalisms, the network model suffers from several deficiencies,

```
CUSTOMER.CUST-NO := "C400"                      /* store search key value in the UWA
find CUSTOMER record                            /* find CUSTOMER record "C400"
find first ORDER record of FROM set             /* find first ORDER record owned by this CUSTOMER record
while ERROR-COUNT = 0
    get ORDER                                   /* get item values of current ORDER record
    <process item values of current ORDER>
    find next ORDER record of FROM set          /* find first ORDER record
end-while

CUSTOMER.CUST-NO := "C400"
find CUSTOMER record                            /* make CUSTOMER record "C400" the current of FROM
if ERROR-COUNT = 0 then
    ORDER.ORD-NO := 30183                       /* store value of ORD-NO in the UWA
    ORDER.CUST-DATE := "2008/08/19"             /* store value of ORD-DATE in the UWA
    store ORDER                                 /* store ORDER record and insert it in the current FROM set
end-if
```

Network Data Model. Figure 3. Two examples of data manipulation code.



Network Data Model. Figure 4. Partial translation of a representative Entity-relationship schema (left) into a network schema (right).

notably the lack of generalization-specialization (is-a) hierarchies and the fact that relationship types are limited to the 1:N category. Translating an Entity-relationship schema into the network model requires the transformation of these missing constructs into standard structures.

Is-a hierarchies. Three popular transformations can be applied to express this construct in standard data management systems, namely one record type per entity type, one record type per supertype and one record type per subtype. Representing each entity type by a distinct record type and forming a set type *S* with each super-type (as owner of *S*) and all its direct subtypes (as members of *S*) is an appropriate implementation of the first variant.

1:1 relationship type. This category is a special case of 1:N and can be expressed by a mere set type, together with dynamic restriction on the number of members in each set. However, merging both record types when one of them depends on the other one (e.g., as an automatic, mandatory member) is also a common option.

Complex relationship type. In most implementations, n-ary and N:N relationship types as well as those with attributes must be reduced to constructs based on 1:N relationship types only through standard transformations. A complex relationship type *R* is represented by a relationship record type *RT* and by as many set types as *R* has roles. The attributes of *R* are translated into fields of *RT*. Cyclic relationship types, if necessary, will be translated in the same way.

Figure 4 illustrates some of these principles.

Discussion

The network model offers a simple view of data that is close to semantic networks, a quality that accounts for much if its past success. The specifications published in the 1971 and 1973 reports exhibited a confusion between abstraction levels that it shared with most proposals of the seventies and that was clarified in later recommendations, notably the 1978 report and X3H2 NDL. In particular, the DDL includes aspects that pertain to logical, physical and procedural layers.

Though they were not implemented in most commercial DBMSs, the CODASYL recommendations included advanced features that are now usual in database technologies such as database procedures, derived fields, check and some kind of triggers.

Key Applications

CODASYL DBMSs have been widely used to manage large corporate databases submitted to both batch and OLTP (On-line Transaction Processing) applications. Compared with hierarchical and relational DBMS, their simple and intuitive though powerful model and languages made them very popular for the development of large and complex applications. However, their intrinsic lack of flexibility in rapidly evolving contexts and the absence of user-oriented interface made them less attractive for decisional applications, such as data warehouses.

Cross-references

▶ Hierarchical Data Model
▶ Relational Model

► Database Management System
► Entity-Relationship Model

Recommended Reading

1. Bachman C. Data structure diagrams. ACM SIGMIS Database, 1(2):4–9, 1969.
2. Bachman C. The programmer as navigator. Commun. ACM, 16(11):635–658, 1973.
3. DBTG C. CODASYL data base task group, April 1971 report, ACM, New York, 1971.
4. DDLC C. CODASYL data description language committee, CODASYL DDL Journal of Development (June 1973), NBS Handbook 113 (Jan. 1974), 1973.
5. Elmasri R. and Navathe S. Fundamentals of Database Systems (3rd edn.). Addison-Wesley, 2000. (The appendix on the network data model has been removed from later editions but is now available on the authors' site.)
6. Engels R.W. An analysis of the April 1971 DBTG report. In Proc. ACM SIGFIDET Workshop on Data Description, Access, and Control, 1971, pp. 69–91.
7. Jones J.L. Report on the CODASYL data description language committee. Inf. Syst., 3(4):247–320, 1978.
8. Michaels A., Mittman B., and Carlson C.A. Comparison of relational and CODASYL approaches to data-base management. ACM Comput. Surv., 8(1):125–151, 1976.
9. Olle W. The CODASYL Approach to Data Base Management, Wiley, New York, NY, 1978.
10. Taylor R. and Frank R. CODASYL data-base management systems. ACM Comput. Surv., 8(1):67–103, 1976.
11. Tsichritzis D. and Lochovsky F. Data Base Management Systems, Academic Press, New York, NY, 1977.

Network Database

► Graph Database

Network Topology

► Visualizing Network Data

Neural Networks

Pang-Ning Tan
Michigan State University, East Lansing, MI, USA

Synonyms

Connectionist model
Parallel distributed processing

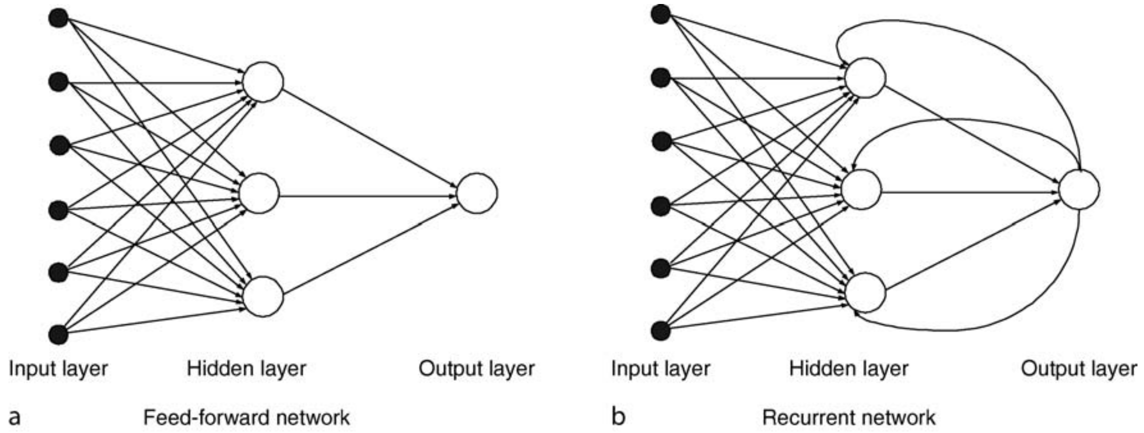
Definition

An artificial neural network (ANN) is an abstract computational model designed to solve a variety of supervised and unsupervised learning tasks. While the discussion in this chapter focuses only on supervised classification, readers who are interested in unsupervised learning using ANN may refer to the literature on vector quantization [6] and self organizing maps [11]. An ANN consists of an assembly of simple processing units called neurons connected by a set of weighted edges (or synapses), as shown in Fig. 1. The neurons are often configured into a feed-forward multi-layered topology, with outputs from one layer being fed into the next layer. The first layer, which is known as the input layer, encodes the attributes of the input data, while the last layer, known as the output layer, encodes the neural network's output. Hidden layers are the intermediary layers of neurons between the input and output layers. A feed-forward ANN without any hidden layer is called a perceptron [16]. Another common ANN architecture is the recurrent network, which allows a neuron to feed its output back into the inputs of other preceding neurons in the network. Such a network topology is useful for modeling temporal and sequential relationships in dynamical systems.

Historical Background

The design of an ANN was inspired by the desire to emulate how a human brain works. Interest in this field began to emerge following the seminal work of McCulloch and Pitts [13], who attempted to understand how complex patterns can be modeled in the brain using a large number of inter-connected neurons. They presented a simplified model of a neuron and showed how a collection of these neurons could be used to represent logical propositions. Nevertheless, they did not provide an algorithm to estimate the weights of the network.

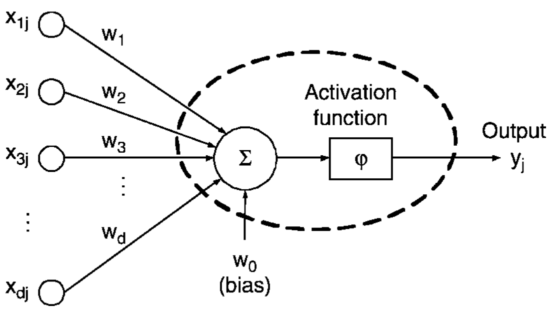
A major step forward in the study of ANN occurred when Hebb [7] formulated a postulate relating the cerebral activities of the brain to the synaptic connections between neurons. Hebb theorized that the process of learning takes place when a pair of neurons is activated repeatedly, thus making their synaptic connection stronger. By strengthening the connection, this enables the network to recognize the appropriate response when the same stimulus is re-applied. This idea forms the basis for what is now known as Hebbian learning.



Neural Networks. Figure 1. Architecture of artificial neural network.

The invention of the perceptron by Rosenblatt [16] marked another major development of the field. Rosenblatt demonstrated that a simple perceptron can be trained to recognize visual patterns by modifying its weights using an iterative error correcting algorithm. A theorem was subsequently proposed to show the guaranteed convergence of the perceptron training rule in finite time – a result that subsequently triggered a wave of interest in the field. During this period, new perceptron training algorithms began to emerge, such as the Widrow-Hoff [18] and stochastic gradient descent [2] algorithms. Such interest, however, began to wane as Minsky and Papert [14] showed the practical limitations of perceptrons, particularly in terms of solving nonlinearly separable problems such as the exclusive-or (XOR) function. Though multi-layer neural networks may overcome these limitations, there has yet to be any feasible learning algorithms that can automatically adjust the weights of the neurons in the hidden layer.

Interest in ANN was finally rekindled in the 1980s, fueled by the successful development of the Boltzmann machine [1] and the re-discovery of the backpropagation algorithm [17], both of which demonstrated the feasibility of training multi-layer neural networks. Furthermore, as computers become cheaper, this permits more researchers to participate in the field and experiment with the capabilities of neural networks, unlike the situation in the 1960s. Research in ANN continued to flourish with the development of more complex networks such as radial basis function networks [15], Hopfield networks [8], Jordan networks [10], and Elman networks [4].



Neural Networks. Figure 2. Structure of an artificial neuron.

Foundations

Neurons are the elementary processing units of an ANN. The structure of a neuron is shown in Fig. 2. Each neuron consists of a set of weighted edges $\{w_1,w_2,,w_d\}$, a threshold w_0 (known as the bias), an adder Σ that computes the weighted sum of its input, and an activation function φ that transforms the weighted sum into its output value. The computation performed by a neuron can be expressed in the following form:

$$y_j = \varphi\left(\sum_{i=1}^{d} w_i x_{ij} + w_0\right) \equiv \varphi(z_j) \qquad (1)$$

The output of a neuron can be discrete or continuous, depending on the choice of activation function. For example, the Heaviside function can be used to produce a binary-valued output:

$$\varphi(z) = \begin{cases} 1, z \geq 0 \\ 0, \text{otherwise} \end{cases}$$

while the linear function $\varphi(z) = z$ and sigmoid function

$$\varphi(z) = \frac{1}{1 + \exp\left(-z\right)}$$

produce a continuous-valued output.

An ANN must be trained to determine the appropriate set of weights for a given classification task. Training is accomplished by processing the training examples one at a time and comparing the predicted class of a training example to its actual class. The error in prediction is then used to modify the weights of the network. This process is repeated until a stopping criterion is satisfied.

Table 1 shows the activation functions and weight update formula employed by several perceptron learning algorithms. Notice that the amount of weight adjustment is proportional to the difference between the true output y_j and the predicted output $\varphi(z_j)$ of a neuron. Computing the error term $[y_j - \varphi(z_j)]$ in the weight update formula is straightforward for perceptrons, but is more challenging for multi-layer networks because the true output of each hidden neuron is unknown. Furthermore, it is unclear the extent to which each of the neuron's weight contributes to the overall network error, an issue that is known as the credit assignment problem. The backpropagation algorithm attempts to overcome this problem by employing a differentiable activation function (such as the sigmoid function) so that a gradient descent strategy can be used to derive the weight update formula. In the backpropagation algorithm, the weight update step is decomposed into two phases. During the forward phase, the outputs of the neurons are computed one layer at a time, starting from neurons in the first hidden layer, followed by those in the next layer until the neurons in the output layer are computed. Next, during the backward phase, the errors are propagated in the reverse direction, starting from neurons in the output layer, followed by those in the preceding layer until the errors of neurons in the first hidden layer are computed.

Neural Networks. Table 1. Perceptron training rules

Perceptron training rule	Activation function	Weight update formula
Rosenblatt's perceptron	Heaviside function	$w_i \leftarrow w_i + \eta[y_j - \varphi(z_j)]x_{ij}$
Widrow-Hoff	Linear function	$w_i \leftarrow w_i + \eta \sum_j [y_j - \varphi(z_j)]x_{ij}$
Stochastic gradient descent	Linear function	$w_i \leftarrow w_i + \eta[y_j - \varphi(z_j)]x_{ij}$

There are several issues that must be considered when building an ANN model. First, the structure of the network must be appropriately chosen to avoid model overfitting. To determine the structure with the right model complexity, model selection approaches such as cross-validation may be used. Regularization methods are also applicable to penalize network structures that are overly complex. Another possibility is to apply the Bayesian approach, which provides a principled framework for handling the model complexity problem. A review of the Bayesian approach for neural networks can be found in [16]. The cascade correlation algorithm [17] is another useful approach to grow the network dynamically from a structure with no hidden layers. Hidden nodes are then added one at a time until the residual error of the network falls within some acceptable level.

Since the error function of a multi-layer neural network is not convex, convergence of the backpropagation algorithm to a local minimum is another potential issue to consider. One way to mitigate the problem is by adding a momentum term to the weight update formula to escape from the local minimum. Alternative strategies include repeating the network simulation with different initial weights and applying a stochastic gradient descent method to estimate the network weights.

Instead of learning a discriminant function to distinguish instances from different classes, an ANN can also be trained to produce posterior probabilities of their class memberships. This is desirable because a probabilistic framework provides a natural way to compensate for imbalanced class distributions, to incorporate the cost of misclassification into decision making, and to fuse the outputs from multiple networks. An ANN can be designed to generate probabilistic outputs by training its weights to minimize the cross-entropy error function [12] instead of the sum-of-square error function.

As ANN encodes a sophisticated mathematical function, the ability to explain how it makes its prediction is crucial for a number of applications, particularly in the medical and legal domains. Algorithms have been developed to extract symbolic representations or rules from a trained network [18]. Such algorithms generally fall into two categories. The first category generates rules based on features constructed from the weights of the network connections. The second category uses the trained network to label a data set and then applies rule extraction algorithms on the labeled data set.

In general, multi-layer neural networks are universal approximators, allowing them to fit any type of function. Furthermore, since it is an abstract computational model, it is also applicable to a variety of supervised, unsupervised, and semi-supervised learning tasks.

Key Applications

ANN has been successfully applied to various applications including pattern recognition (face, handwriting, and speech recognition), finance (bankruptcy prediction, bond rating, and economic forecasting), manufacturing (process control, tool condition monitoring, and robot scheduling), and computer aided diagnosis (arrhythmia identification and tumor detection in biological tissues).

Future Directions

As ANN is an abstract computational model that is applicable to a wide spectrum of learning tasks, much of the future advances in this field are tied to progress in the areas of supervised, unsupervised, and semi-supervised learning.

Experimental Results

The experimental evaluation often depends on the learning tasks. For supervised classification, some of the typical evaluation criteria include model accuracy, specificity, sensitivity, F-measure, and run-time for model building.

Data Sets

A large collection of data sets are available at the UCI data mining and machine learning repository. For a more comprehensive list of data sets and other information about neural networks, go to http://www.neural-forecasting.com/.

Url to Code

A comprehensive set of functions for implementing feed-forward and recurrent neural networks is available in MATLAB toolbox.

Cross-references

▶ Decision Tree Classification

▶ Data Clustering

Recommended Reading

1. Ackley D.H., Hinton G.E., and Sejnowski T.J. A learning algorithm for Boltzmann machines. Cogn. Sci., 9(1) 147–169, 1985.
2. Amari S. A theory of adaptive pattern classifiers. IEEE Trans. Electronic Comput., 16, 299–307, 1967.
3. Craven M. and Shavlik J.W. Learning symbolic rules using artificial neural networks. In Proc. 10th Int. Conf. on Machine Learning, 1993, pp. 73–80.
4. Elman J.L. Finding structure in time. Cogn. Sci., 14, 179–211, 1990.
5. Fahlman S.E. and Lebiere C. The CASCADE-CORRELATION learning architecture. Adv. Neural Inform. Process. Syst., pp. 524–532, 1989.
6. Haykin S. Neural networks – a comprehensive foundation, 2nd edn. Prentice-Hall, Englewood, Cliffs, NJ, 1998.
7. Hebb D. The Organization of Behaviour, Wiley, New York, 1949.
8. Hopfield J.J. Neural networks and physical systems with emergent collective computational abilities. In Proc. National Academy of Sciences, 79, 2554–2558, 1982.
9. Hopfield J.J. Learning algorithms and probability distributions in feed-forward and feed-back networks. In Proc. National Academy of Sciences, 84, 8429–8433, 1987.
10. Jordan M.I. Attractor dynamics and parallelism in a connectionist sequential machine. In Proc. 8th Annual Conf. of the Cognitive Science Society, 1986, pp. 531–546.
11. Kohonen T. Self-Organizing Maps, Springer, Berlin. 2001.
12. Lampinen J. and Vehtari A. Bayesian approach for neural networks – review and case studies, Neural Networks, 14(3): 7–24, 2001.
13. McCulloch W.S. and Pitts W. A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys., 5:115–133, 1943.
14. Minsky M. and Papert S. Perceptrons: An Introduction to Computational Geometry. MIT, Cambridge, MA, 1969.
15. Powell M. Radial Basis Functions for Multivariable Interpolation : A Review. In Algorithms for Approximation. Mason J.C. and Cox M.G. (eds.). pp. 143–167, 1987.
16. Rosenblatt F. Principles of Neurodynamics. Spartan Books, New York, 1959.
17. Rumelhart D.E., Hinton G.E., and Williams R.J. Learning representations by backpropagating errors. Nature, 323, 533–536, 1986.
18. Widrow B. and Hoff M.E. Jr. Adaptive switching circuits. IRE WESCON Convention Record. 1960, pp. 96–104.

New Media Metadata

▶ Multimedia Metadata

NEXI

▶ Narrowed Extended XPath I

NFS

▶ Storage Protocols

NF-SS

▶ Normal Form ORA-SS Schema Diagrams

N-Gram Models

Djoerd Hiemstra
University of Twente, AE Enschede, The Netherlands

Definition

In language modeling, *n*-gram models are probabilistic models of text that use some limited amount of history, or word dependencies, where *n* refers to the number of words that participate in the dependence relation.

Key Points

In automatic speech recognition, *n*-grams are important to model some of the structural usage of natural language, i.e., the model uses word dependencies to assign a higher probability to "how are you today" than to "are how today you," although both phrases contain the exact same words. If used in information retrieval, simple unigram language models (*n*-gram models with $n = 1$), i.e., models that do not use term dependencies, result in good quality retrieval in many studies. The use of bigram models (*n*-gram models with $n = 2$) would allow the system to model direct term dependencies, and treat the occurrence of "New York" differently from separate occurrences of "New" and "York," possibly improving retrieval performance. The use of trigram models would allow the system to find direct occurrences of "New York metro," etc. The following equations contain respectively (1) a unigram model, (2) a bigram model, and (3) a trigram model:

$$P(T_1, T_2, \cdots T_n|D) = P(T_1|D)P(T_2|D) \cdots P(T_n|D) \tag{1}$$

$$\begin{aligned} &P(T_1, T_2, \cdots T_n|D) \\ &\quad = P(T_1|D)P(T_2|T_1, D) \cdots P(T_n|T_{n-1}, D) \end{aligned} \tag{2}$$

$$\begin{aligned} &P(T_1, T_2, \cdots T_n|D) \\ &\quad = P(T_1|D)P(T_2|T_1, D)P(T_3|T_1, T_2, D) \\ &\quad \cdots P(T_n|T_{n-2}, T_{n-1}, D) \end{aligned} \tag{3}$$

The use of *n*-gram models increases the number of parameters to be estimated exponentially with *n*, so special care has to be taken to smooth the bigram or trigram probabilities. Several studies have shown small but significant improvements of using bigrams if smoothing parameters are properly tuned [2,3]. Improvements of the use of *n*-grams and other term dependencies seem to be bigger on large data sets [1].

Cross-references

▶ Language Models
▶ Probability Smoothing

Recommended Reading

1. Metzler D. and Bruce Croft W. A Markov random field model for term dependencies. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 472–479.
2. Miller D.R.H., Leek T., and Schwartz R.M. A hidden Markov model information retrieval system. In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 214–221.
3. Song F. and Bruce Croft W. A general language model for information retrieval. In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 4–9.

NIAM

▶ Object-Role Modeling

NN Classification

▶ Nearest Neighbor Classification

NN Query

▶ Nearest Neighbor Query
▶ Nearest Neighbor Query in Spatio-temporal Databases

NN Search

▶ Nearest Neighbor Query in Spatio-temporal Databases

Node

▶ Activity

Noise Addition

Josep Domingo-Ferrer
Universitat Rouira i Virgili, Tarragona, Catalonia

Synonyms

Additive noise

Definition

Noise addition is a masking method for statistical disclosure control of numerical microdata that consists of adding random noise to original microdata.

Key Points

The noise addition algorithms in the literature are:

1. *Masking by uncorrelated noise addition.* The vector of observations x_j for the jth attribute of the original dataset X_j is replaced by a vector

$$z_j = x_j + \epsilon_j$$

where ϵ_j is a vector of normally distributed errors drawn from a random variable $\varepsilon_j \sim N(0, \sigma^2_{\varepsilon_j})$, such that $Cov(\varepsilon_t, \varepsilon_l) = 0$ for all $t \neq l$. This does not preserve variances nor correlations.
2. *Masking by correlated noise addition.* Correlated noise addition also preserves means and additionally allows preservation of correlation coefficients. The difference with the previous method is that the covariance matrix of the errors is now proportional to the covariance matrix of the original data, i.e., $\varepsilon \sim N(0, \Sigma_\varepsilon)$, where $\Sigma_\varepsilon = \alpha\Sigma$.
3. *Masking by noise addition and linear transformation.* In Kim [1], a method is proposed that ensures by additional transformations that the sample covariance matrix of the masked attributes is an unbiased estimator for the covariance matrix of the original attributes.
4. *Masking by noise addition and nonlinear transformation.* An algorithm combining simple additive noise and nonlinear transformation is proposed in Sullivan [2]. The advantages of this proposal are that it can be applied to discrete attributes and that univariate distributions are preserved. Unfortunately, the application of this method is very time-consuming and requires expert knowledge on the data set and the algorithm.

See Brand [3] for more details on specific algorithms.

Cross-references

▶ Inference Control in Statistical Databases
▶ Microdata
▶ SDC Score

Recommended Reading

1. Kim J. J. A method for limiting disclosure in microdata based on random noise and transformation. In Proc. Section on Survey Research Methods, American Statistical Association. Alexandria VA, 1986, pp. 303–308.
2. Sullivan G.R. The Use of Added Error to Avoid Disclosure in Microdata Releases. PhD Thesis, Iowa State University, 1989.
3. Brand R. Microdata protection through noise addition. In Inference Control in Statistical Databases, J. Domingo-Ferrer (ed.). LNCS, Volume 2316. Springer, 2002, pp. 97–116.

Non-Clustering Index

▶ Secondary Index

Non-Dense Index

▶ Sparse Index

Nonidentifiability

▶ Anonymity

Nonlinear Magnification

▶ Distortion Techniques

Non-Metric Temporal Reasoning

► Qualitative Temporal Reasoning

Nonparametric Data Reduction Techniques

Rui Zhang
University of Melbourne, VIC, Australia

Definition

A nonparametric data reduction technique is a data reduction technique that does not assume any model for the data.

Key Points

Nonparametric data reduction (NDR) techniques is opposite to parametric data reduction (PDR) techniques. A PDR technique must assume a certain model for the data. Parameters of the model are determined before the data reduction is performed. A NDR technique does not assume any model and is applied to the data directly. The data reduction effectiveness of a PDR technique heavily depends on whether the model suits the data well. If well-suited, good accuracy as well as substantial data reduction can be achieved; otherwise, both cannot be achieved at the same time. A NDR technique yields more uniform effectiveness irrespective of the data, but it may not achieve as high data reduction as a well-suited PDR technique.

Popular NDR techniques include histograms, clustering and indexes. Histograms are used to approximate data distributions. An equidepth histogram can adjust itself to the data distribution and always gives good approximation of the data, no matter how the data are distributed. Clustering techniques also find the cluster centers automatically, irrespective of the data distribution. Although parameters are used in some clustering algorithms, such as the value of k in a k-means algorithm, k is a given parameter instead of a parameter determined by the actual data. Conceptually, indexes are similar to clustering and they also adjust themselves to the data distribution. No parameter needs to be determined from the data for indexes. A summary of data reduction techniques including NDR techniques can be found in [1].

Cross-references

► Clustering
► Data Deduction
► Histogram
► Parametric Data Reduction Techniques

Recommended Reading

1. Barbará D., DuMouchel W., Faloutsos C., Haas P.J., Hellerstein J.M., Ioannidis Y.E., Jagadish H.V., Johnson T., Ng R.T., Poosala V., Ross K.A., and Sevcik K.C. The New Jersey data reduction report. Q. Bull. IEEE TC on Data Engineering, 20(4):3–45, 1997.

Non-Perturbative Masking

► Non-Perturbative Masking Methods

Non-Perturbative Masking Methods

Josep Domingo-Ferrer
The Public University of Tarragona, Tarragona, Spain

Synonyms

Non-perturbative masking

Definition

Non-perturbative masking methods are SDC methods for microdata protection which do not alter data; rather, they produce partial suppressions or reductions of detail in the original dataset. Sampling, global recoding, top and bottom coding and local suppression are examples of non-perturbative masking methods.

Key Points

1. Sampling is a non-perturbative masking method for statistical disclosure control of microdata. Instead of publishing the original microdata file, what is published is a sample S of the original set of records. Sampling methods are suitable for categorical microdata, but for continuous microdata they should probably be combined with other masking methods. The reason is that sampling alone leaves a continuous attribute V_i unperturbed for all records in S. Thus, if attribute V_i is present in an external administrative public file, unique matches with the published sample are very likely: indeed, given a continuous attribute V_i and two respondents o_1 and o_2, it is

highly unlikely that V_i will take the same value for both o_1 and o_2 unless $o_1 = o_2$ (this is true even if V_i has been truncated to represent it digitally).

2. Global recoding or generalization is a masking method for statistical disclosure control of microdata. For a categorical attribute V_i, several categories are combined to form new (less specific) categories, thus resulting in a new V'_i with $|D(V'_i)| < |D(V_i)|$ where $|\cdot|$ is the cardinality operator. For a numerical attribute, global recoding means replacing V_i by another attribute V'_i which is a discretized version of V_i. In other words, a potentially infinite range $D(V_i)$ is mapped onto a finite range $D(V'_i)$. This technique is more appropriate for categorical microdata, where it helps disguise records with strange combinations of categorical attributes. Global recoding is used heavily by statistical offices. Global recoding is implemented in the μ-Argus package. In combination with local suppression, it can be used to achieve *k*-anonymity.
3. Top coding and bottom coding are special cases of the global recoding masking method for statistical disclosure control of microdata. Their operating principle is that top values (those above a certain threshold), respectively bottom values (those below a certain threshold), are lumped together to form a new category. Top and bottom coding can be used on attributes that can be ranked, that is, numerical or categorical ordinal.
4. Local suppression or blanking is a masking method for statistical disclosure control of microdata. Certain values of individual attributes are suppressed with the aim of increasing the set of records agreeing on a combination of key values. Local suppression is implemented in the μ-Argus package. Ways to combine local suppression and global recoding are discussed in DeWaal and Willenborg (1995). In fact, the combination of both methods can be used to attain *k-anonymity.* If a numerical attribute V_i is part of a set of key attributes, then each combination of key values is probably unique. Since it does not make sense to systematically suppress the values of V_i, it can be asserted that local suppression is rather oriented to categorical attributes.

Cross-references

▶ Inference Control in Statistical Databases
▶ *k*-Anonymity
▶ Microdata

Recommended Reading

1. DeWaal A.G. and Willenborg L.C.R.J. Global recodings and local suppressions in microdata sets. In Proc. Statistics Canada Symposium'95, 1995, pp. 121–132.
2. Hundepool A., Domingo-Ferrer J., Franconi L., Giessing S., Lenz R., Longhurst J., Schulte-Nordholt E., Seri G., and DeWolf P.-P. Handbook on Statistical Disclosure Control (Version 1.0). Eurostat (CENEX SDC Project Deliverable), 2006. http://neon.vb.cbs.nl/CENEX/
3. Hundepool A., Van de Wetering A., Ramaswamy R., Franconi F., Polettini S., Capobianchi A., DeWolf P.-P., Domingo-Ferrer J., Torra V., Brand R., and Giessing S. μ-ARGUS User's Manual Version 4.1 February 2007. http://neon.vb.cbs.nl/CASC.
4. Willenborg L. and DeWaal T. Elements of Statistical Disclosure Control. Springer-Verlag, New York, 2001.

Non-Pipelineable Operator

▶ Stop-&-Go Operator

Nonsequenced Semantics

Michael H. Böhlen[1], Christian S. Jensen[2], Richard T. Snodgrass[3]

[1]Free University of Bozen-Bolzano, Bozen-Bolzano, Italy
[2]Aalborg University, Aalborg, Denmark
[3]University of Arizona, Tucson, AZ, USA

Synonyms

Nontemporal semantics

Definition

Nonsequenced semantics guarantees that query language statements can reference and manipulate the timestamps that capture the valid and transaction time of data in temporal databases as regular attribute values, with no built-in temporal semantics being enforced by the query language.

Key Points

A temporal database generalizes a non-temporal database and associates one or more timestamps with database entities. Different authors have suggested temporal query languages that provide advanced support for formulating temporal statements. Results of these efforts include a variety of temporal extensions of SQL, temporal algebras, and temporal logics that simplify the management of temporal data.

Languages with built-in temporal support are attractive because they offer convenient support for formulating a wide range of common temporal statements. The classical example of advanced built-in temporal support is *sequenced semantics*, which makes it possible to conveniently interpret a temporal database as a sequence of non-temporal databases. To achieve built-in temporal support, the timestamps are viewed as implicit attributes that are given special semantics.

Built-in temporal support, however, may also limit the expressiveness of the language when compared to the original non-temporal language where timestamps are explicit attributes. Nonsequenced semantics guarantees that statements can manipulate timestamps as regular attribute values with no built-in temporal semantics being enforced. This ensures that the expressiveness of the original language is preserved.

The availability of legacy statements with the standard non-temporal semantics is also important in the context of migration where users can be expected to be well-acquainted with the semantics of their non-temporal language. Nonsequenced semantics ensures that users are able to keep using the paradigm they are familiar with and to incrementally adopt the new features. Moreover, from a theoretical perspective, any variant of temporal logic, a well-understood language that only provides built-in temporal semantics, is strictly less expressive than a first order logic language with explicit references to time [1,4].

Each statement of the original language has the potential to either be evaluated with temporal or non-temporal semantics. For example, a count query can count the tuples at each time instant (this would be temporal, i.e., sequenced, semantics) or count the tuples actually stored in a relation instance (this would be non-temporal, i.e., nonsequenced, semantics).

To distinguish the two semantics, different approaches have been suggested. For instance, TempSQL distinguishes between so-called current and classical users. ATSQL and SQL/Temporal offer so-called statement modifiers that enable the users to choose between the two semantics at the granularity of statements. Below, statement modifiers are used for illustration. Specifically the ATSQL modifier `NSEQ VT` signals standard SQL semantics with full control over the timestamp attributes of a valid-time database.

The illustrations assume a database instance with three relations:

Employee

ID	Name	VTIME
1	Bob	5–8
3	Pam	4–12
4	Sarah	1–5

Salary

ID	Amt	VTIME
1	20	4–10
3	20	6–9
4	20	6–9

Bonus

ID	Amt	VTIME
1	20	1–6
1	20	7–12
3	20	1–12

and the following queries

```
NSEQ VT
   SELECT COUNT(*) FROM Bonus;

NSEQ VT
   SELECT E.ID
   FROM Employee AS E, Salary AS S
   WHERE VTIME(E) PRECEDES VTIME(S) AND
   E.ID = S.ID;
```

Both queries are nonsequenced, i.e., the valid time is treated as a regular attribute without any special processing going on. The first query determines the number of bonuses that have been paid. It returns the number of tuples in the `Bonus` relation, which is equal to three. Note that if the sequenced modifier was used (cf. *sequenced semantics*) then the time-varying count had been computed. With the given example, the count at each point in time would be two. The second query joins `Employee` and `Salary`. The join is not performed at each snapshot (cf. *sequenced semantics*). Instead it requires that the valid time of `Employee` precedes the valid time of `Salary`. The result is a non-temporal table.

Nonsequenced statements offer no built-in temporal support, but instead offer complete control. This is akin to programming in assembly language, where one can do everything, but everything is hard to do. The query language must provide a set of functions and predicates for expressing temporal relationships (e.g., `PRECEDES` [2]) and performing manipulations and computations on timestamps (e.g., `VTIME`). The resulting new query-language constructs are relatively easy to implement because they only require changes at the level of built-in predicates and functions. Instead of using functions and predicates on timestamps, the use of temporal logic with temporal connectives has also been suggested.

Cross-references

▶ Allen's Relations
▶ Sequenced Semantics
▶ SQL-Based Temporal Query Languages
▶ Temporal Database
▶ Valid Time

Recommended Reading

1. Abiteboul S., Herr L., Van den Bussche J. Temporal versus first-order logic to query temporal databases. In Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 1996, pp. 49–57.
2. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 16(11):832–843, 1983.
3. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. ACM Trans. Database Syst., 25(4):48, December 2000.
4. Toman D. and Niwiński D. First-order queries over temporal databases inexpressible in temporal logic. In Advances in Database Technology, Proc. 5th International Conf. on Extending Database Technology, 1996, pp. 307–324.

Nontemporal Semantics

▶ Nonsequenced Semantics

Non-Uniform Distribution

▶ Data Skew

Normal Form ORA-SS Schema Diagrams

Gillian Dobbie[1], Tok Wang Ling[2]
[1]University of Auckland, New Zealand
[2]National University of Singapore, Singapore, Singapore

Synonyms

NF-SS; Normalizing ORA-SS diagrams

Definition

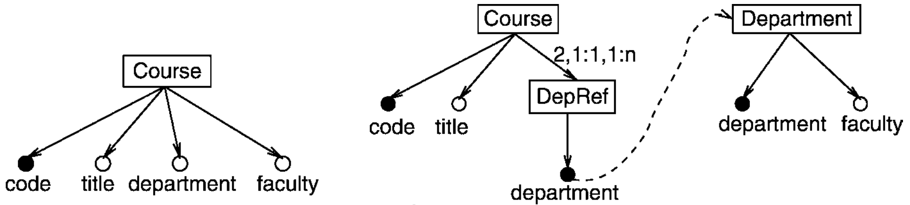
Normal forms have been defined for data models, such as the relational data model, the nested relational data model, the object-oriented data model and more recently, the semi-structured data model, to recognize (and remove) certain kinds of redundant data. A normal form that has been defined for semi-structured data, based on the ORA-SS data model, is described.

Key Points

The definition of a normal form for ORA-SS (Object-Relationship-Attribute data model for Semi-structured data) diagrams is based on the definition of a normal form for the nested relational data model as described by Ling and Yan [1], recognizing the similarity between the nesting relationship in the nested relational data model and hierarchies in the semi-structured data model.

The definition for NF ORA-SS (normal form for the ORA-SS data model) can be broken into three main parts. The first part ensures that every object class is normalized. The second part ensures that every relationship type is normalized. The third part ensures that there is no data that can be derived from other data in the diagram. Examples of ORA-SS schemas that are not in NF ORA-SS are shown below, along with schemas that are in NF ORA-SS that capture the same information.

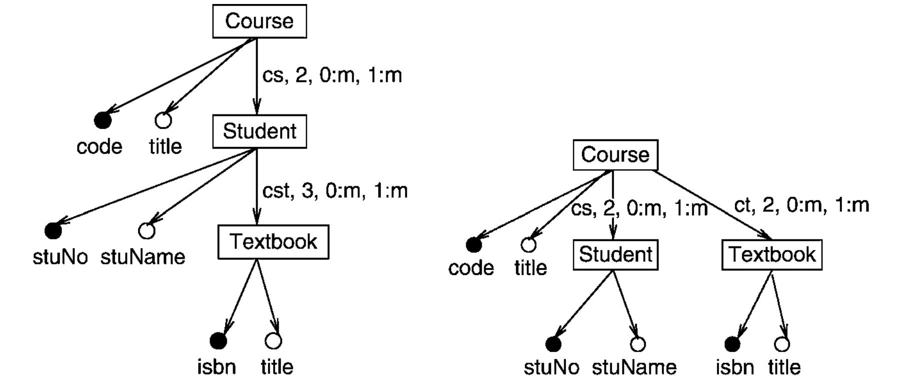
Figure 1a represents an object class *Course*. The attributes of course are *code*, *title*, *department* and *faculty*. This object class is not normalized since *code* determines *title* and *department*, but *department* determines *faculty*. What this means is that a department belongs to only one faculty, and this information needs to be stored once rather than once for every course. Figure 1b has object classes *Course*, *DepRef* and *Department*. Object class *DepRef* has a reference to object class *Department*, so the information about the faculty a department belongs to is stored only once. The object classes in Fig. 1b are normalized.



a Object class *course* is not normalized

b Object classes *course* and *department* are normalized

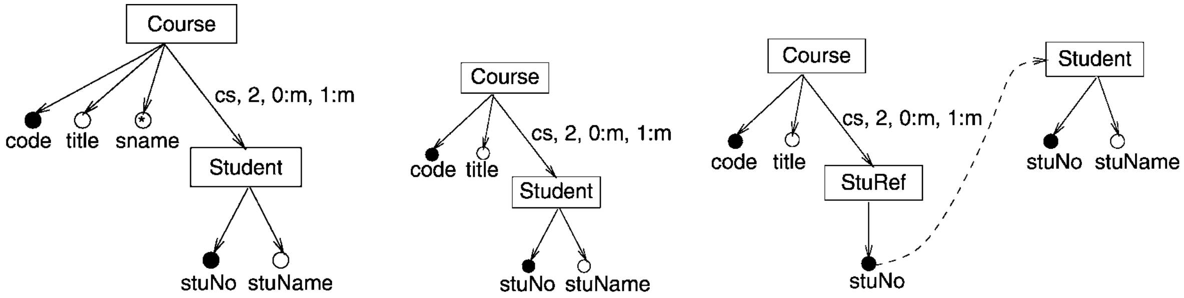
Normal Form ORA-SS Schema Diagrams. Figure 1. Example of object class normal form.



a Relationship type *cst* is not normalized

b Relationship types *ct* and *cs* are normalized

Normal Form ORA-SS Schema Diagrams. Figure 2. Example of relationship type normal form.



a ORA-SS schema where student name is duplicated

b ORA-SS schema where student is duplicated

c An ORA-SS schema in NF ORA-SS

Normal Form ORA-SS Schema Diagrams. Figure 3. Example of ORA-SS normal form (NF ORA-SS).

Figure 2a shows a relationship type *cst*, which is a ternary relationship type among the object classes *Course*, *Student*, and *Textbook*. Intuitively it makes sense because courses have students and textbooks. However, students and textbooks are independent of each other. A course has a certain set of students, irrespective of the textbooks for the course, and a course has a certain set of textbooks irrespective of the students taking the course. In the schema in Fig. 2a, the textbooks for a given course are repeated for each student taking the course. Figure 2b has two relationship types *cs* and *ct*, between *Course* and *Student*, and *Course* and *Textbook*, respectively. Both of these relationship types are normalized. However, the object classes *Student* and *Textbook* in Fig. 2b are still not normalized.

Figure 3a shows two object classes *Course* and *Student*. Object class *Course* has attributes *code*, *title* and *sname*. Attribute *sname* is a multivalued attribute and has the names of the students taking the course. This information is repeated in the attribute *stuName* that belongs to object class *Student*, so the ORA-SS schema in Fig. 3a is not in NF ORA-SS. If the student names are stored only in the attribute *stuName* of object class *Student*, as shown in Fig. 3b that redundancy is removed. Of course the many-to-many relationship type leads to the students name being repeated in every course the student takes. Using a reference to represent this relationship removes this redundancy and the resulting ORA-SS diagram in Fig. 3c is in NF ORA-SS.

Cross-references

▶ Normal Forms and Normalization
▶ Object Relationship Attribute Data Model for Semi-structured Data
▶ Semi-structured Database Design

Recommended Reading

1. Ling T.W. and Yan L.L. NF-NR: a practical normal form for nested relations. J. Syst. Int., 4:309–340.

Normal Forms and Normalization

Marcelo Arenas
Pontifical Catholic University of Chile, Santiago, Chile

Definition

A normal form defines a condition over a set of data dependencies, or semantic constraints, that has been specified as part of a database schema. This condition is used to check whether the design of the database has some desirable properties (for example, the database does not store redundant information), and if this is not the case, then it can also be used to convert the poorly designed database into an equivalent well-designed one.

Historical Background

Information is one of the most – if not the most – valuable assets of a company. Therefore, organizations need tools to allow them to structure, query and analyze their data, and, in particular, they need tools providing simple and fast access to their information.

During the last 30 years, relational databases have become the most popular computer application for storing and analyzing information. The simplicity and elegance of the relational model, where information is stored just as tables, has largely contributed to this success.

To use a relational database, a company first has to think of its data as organized in tables. How easy it is for a user to understand this organization and use the database depends on the design of these relations. If tables are not carefully selected, users can spend too much time executing simple operations, or may not be able to extract the desired information.

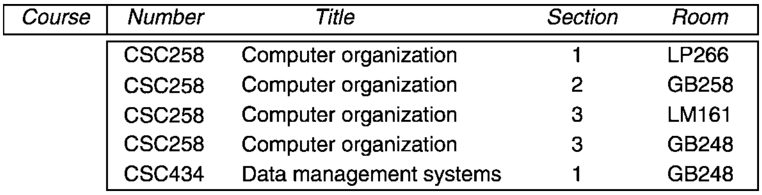
Since the beginnings of the relational model, it was clear for the research community that the process of designing a database is a nontrivial and time-consuming task. Even for simple application domains, there are many possible ways of storing the data of interest. Soon the difficulties in designing a database became clear for practitioners, and the design problem was recognized as one of the fundamental problems for the relational technology.

During the 70s and 80s, a lot of effort was put into developing methodologies to aid in the process of deciding how to store data in a relational database. The most prominent approaches developed at that time – which today are a standard part of the relational technology – were the entity-relationship and the normalization approaches. In the former approach, a diagram is used to specify the objects of an application domain and the relationships between them. The *schema* of the relational database, i.e., the set of tables and column names, is then automatically generated from the diagram. In the normalization approach, an already designed relational database is given as input, together with some semantic information provided by a user in the form of relationships between different parts of the database, called *data dependencies*. This semantic information is then used to check whether the design has some desirable properties, and if this is not the case, it can also be used to convert the poor design into an equivalent well-designed database.

Foundations

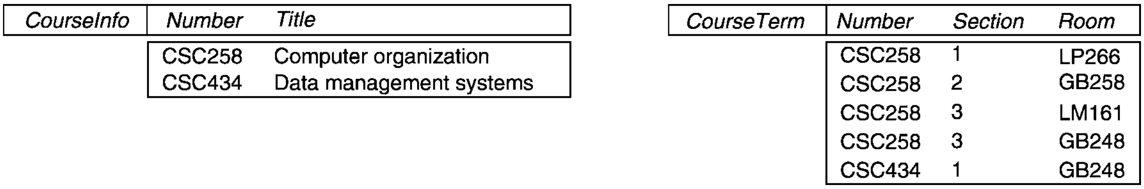
In the relational model, a database is viewed as a collection of relations or tables. For instance, a relational database storing information about courses in a university is shown in Fig. 1. This relation consists of a time-varying part, the data about courses, and a part considered to be time independent, the *schema* of the relation, which is given by the name of the relation (*Course*) and the names of the attributes of this relation.

Usually, the information contained in a database satisfies some *semantic* restrictions. For example, in the



Course	Number	Title	Section	Room
	CSC258	Computer organization	1	LP266
	CSC258	Computer organization	2	GB258
	CSC258	Computer organization	3	LM161
	CSC258	Computer organization	3	GB248
	CSC434	Data management systems	1	GB248

Normal Forms and Normalization. Figure. 1. Example of a relational database.



CourseInfo	Number	Title
	CSC258	Computer organization
	CSC434	Data management systems

CourseTerm	Number	Section	Room
	CSC258	1	LP266
	CSC258	2	GB258
	CSC258	3	LM161
	CSC258	3	GB248
	CSC434	1	GB248

Normal Forms and Normalization. Figure. 2. Example of a normalized relational database.

relation *Course* shown in Fig. 1, it is expected that only one title is associated with each course number. These restrictions are called *data dependencies*, and they are expressed by using suitable languages. For example, the previous constraint corresponds to a functional dependency, which is an expression of the form $X \rightarrow Y$, where X and Y are sets of attributes. A relation satisfies $X \rightarrow Y$ if for every pair of tuples t_1,t_2 in it, if t_1 and t_2 have the same values on X, then they have the same values on Y. Thus, for example, the relation shown in Fig. 1 satisfies functional dependency *Number* → *Title* since each course has only one title.

In [2], Codd showed that a database containing functional dependencies may exhibit some anomalies when the information is updated. For example, consider again the relational database shown in Fig. 1, which includes functional dependency *Number* → *Title*. This database is prone to three different types of anomalies. First, if the name of the course with number CSC258 is changed to Computer Organization I, then four distinct cells need to be updated. If any of them is not updated, then the information in the database becomes inconsistent. This anomaly was called an *update anomaly* by Codd [2], and it arises because the instance is storing redundant information. Second, if the information is updated because a new semester is starting, and the course with number CSC434 is not given in that semester, then the last tuple of the instance is deleted and no information about CSC434 appears in the updated instance. This has the additional effect of deleting the title of the course, which will be the same the next time that CSC434 is offered. This anomaly was called a *deletion anomaly* by Codd [2], and it arises because the relation is storing information that is not directly related; the sections of a course vary from one term to another while its title is likely not to be changed from one semester to the next one. This can also lead to *insertion anomalies* [2]; if a new course (CSC336, Numerical Methods) is created, then it cannot be added to the database until at least one section and one room is assigned to the course.

To avoid updates anomalies, Codd introduced three increasingly restrictive *normal forms* [2,3], which specify some syntactic properties that the set of functional dependencies in a database must satisfy. The most restrictive among them is known today as Boyce-Codd Normal Form (BCNF). Informally, a database schema S including a set Σ of functional dependencies is in BCNF, if there is no set of attributes $X \cup \{A\} \cup \{B\}$ such that $A, B \notin X$, $X \rightarrow A$ holds in S but $X \rightarrow B$ does not hold in S, that is, if there are no attributes in S with different levels of association between them according to Σ (If $X \rightarrow A$ holds in S but $X \rightarrow B$ does not hold in S, then for every value of X, there exists only one value of A in the database but possibly many values of B.). For example, the database shown in Fig. 1 is not in BCNF since *Number* → *Title* holds in this database, while *Number* → *Section* does not hold.

In [2], Codd also introduced the first *normalization algorithm*, that is, a procedure that takes as input a relational schema that includes some data dependencies and does not satisfy some particular normal form, and produces a new schema that conforms to this normal form. For example, if S is the relational schema *Course* (*Number*, *Title*, *Section*, *Room*) and Σ is the set of functional dependencies $\{Number \rightarrow Title\}$, then the standard normalization algorithm for BCNF [1] produces a new schema where *Course* is split into two tables: *CourseInfo*(*Number*, *Title*) and *CourseTerm*(*Number*, *Section*, *Room*). In Fig. 2, it is shown how the information in the initial database in Fig. 1 is stored under the new schema. It should be noticed that the new schema is not prone to any of the anomalies mentioned at the beginning of this section:

- If the name of the course with number CSC258 is changed to Computer Organization I, then only one cell needs to be updated.
- If the information is updated because a new semester is starting, and the course with number CSC434 is not given in that semester, then the last tuple of relation *CourseTerm* is deleted. This does not have the additional effect of deleting the title of the course (this information is kept in the relation *CourseInfo*).
- If a new course (CSC336, Numerical Methods) is created, then it can be added to the database even if no section has been created for this course (tuple (CSC336, Numerical Methods) is included only in the relation *CourseInfo*).

A normalization algorithm takes as input a relational schema and generates a database schema in some particular normal form. It is desirable that these two are as similar as possible, that is, they should contain the same data and the same semantic information. These properties have been called *information losslessness* and *dependency preservation* in the literature, respectively.

Let S_1,S_2 be two database schemas. Intuitively, two instances I_1 of S_1 and I_2 of S_2 contain the same information if it is possible to retrieve the same information from them, that is, for every query Q_1 over I_1 there exists a query Q_2 over I_2 such that $Q_1(I_1) = Q_2(I_2)$, and vice versa. To formalize this notion, a query language has to be chosen. If this query language is relational algebra, then this notion is captured by the notion of calculously dominance introduced by Hull [7]. Schema S_2 *dominates* S_1 *calculously* if there exist relational algebra expressions Q over S_1 and Q' over S_2 satisfying the following property: For every instance I of S_1, there exists an instance I' of S_2 such that $Q(I) = I'$ and $Q'(I') = I$. Thus, every query Q_1 over I can be transformed into an equivalent query $Q_2 = Q_1 \circ Q'$ over I', since $Q_2(I') = Q_1(Q'(I')) = Q_1(I)$, and, analogously, every query Q_2 over I' can be transformed into an equivalent query $Q_1 = Q_2 \circ Q$ over I, since $Q_1(I) = Q_2(Q(I)) = Q_2(I')$.

Normalization algorithms try to achieve the goal of information losslessness; if any of them transforms a database schema S into a database schema S', then S' should dominate S calculously. The standard normalization algorithm for BCNF uses only the projection operator to transform a schema [1] and, thus, calculously dominance is defined in terms of this operator and its inverse, the join operator. More precisely, the normalization algorithm mentioned in this section takes as input a relation R and a set of functional dependencies Σ, and uses the projection operator to transform it into a database schema in BCNF that is composed by some relations $R_1,\ldots,R_n$. Then $R_1,\ldots,R_n$ is a *lossless decomposition* of R if for every instance I of R there is an instance I' of $R_1,\ldots,R_n$ such that:

- For every $i \in \{1,\ldots,n\}$, it holds that $I'_i = \pi_{U_i}(I)$, where I'_i is the R_i-relation of I' and U_i is the set of attribute of R_i, and
- $I = I'_1 \bowtie I'_2 \bowtie \ldots \bowtie I'_n$, where each I'_i is the R_i-relation of I'.

That is, every instance I of S can be transformed into an instance I' of S' by using the projection operator, and I can be reconstructed from I' by using the join operator. For example, the relation *CourseInfo* in Fig. 2 can be obtained by projecting the relation *Course* in Fig. 1 over the set of attributes $\{Number, Title\}$, while the relation *CourseTerm* in Fig. 2 can be obtained by projecting the relation *Course* in Fig. 1 over the set of attributes $\{Number, Section, Room\}$. Moreover, the relation *Course* in Fig. 1 can be obtained by joining relations *CourseInfo* and *CourseTerm* in Fig. 2. Given that this holds for every instance I of the initial schema *Course* (*Number*,*Title*,*Section*,*Room*), the new schema *CourseInfo* (*Number*,*Title*), *CourseTerm* (*Number*, *Section*, *Room*) is said to be a lossless decomposition of the initial one.

Normalization algorithms also try to achieve the goal of dependency preservation; if any of them transforms a database schema S including a set Σ of data dependencies, into a database schema S' including a set Σ' of data dependencies, then Σ should be equivalent to Σ' (no semantic information is lost). In the running example, the standard normalization algorithm for BCNF produces a new schema *CourseInfo*(*Number*, *Title*) and *CourseTerm*(*Number*, *Section*, *Room*), and also includes dependency *Number* → *Title* in the relation *CourseInfo*. Thus, the new schema is a dependency preserving decomposition of the initial one.

The normalization approach was proposed in the early 70s by Codd [2]. Since then, many researchers have studied the normalization problem for relational databases and other data models, and today it is possible to find normal forms for many different types of data dependencies: 3NF [2] and BCNF [3] for functional dependencies, 4NF [4] for multivalued dependencies, PJ/NF [5] and 5NFR [8] for join dependencies, and DK/NF [6] for general constraints. These normal forms, together with normalization algorithms for converting a poorly designed database into a well-designed database, can be found today in every database textbook.

Key Applications

Normal forms and normalization algorithms are essential to schema design, redundancy elimination, update anomaly prevention and efficient storage.

Cross-references

- ▶ Boyce-Codd Normal Form
- ▶ Fourth Normal Form
- ▶ Second Normal Form (2NF)
- ▶ Third Normal Form

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases. Addison-Wesley, Reading, MA, USA, 1995.
2. Codd E.F. Further normalization of the data base relational model. In Data base systems. Prentice-Hall, Englewood Cliffs, NJ, USA, 1972, pp. 33–64.
3. Codd E.F. Recent investigations in relational data base systems. In IFIP Congress. North-Holland, Amsterdam, 1974, pp. 1017–1021.
4. Fagin R. Multivalued dependencies and a new normal form for relational databases. ACM Trans. Database Syst., 2(3):262–278, 1977.
5. Fagin R. Normal forms and relational database operators. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 153–160.
6. Fagin R. A normal form for relational databases that is based on domians and keys. ACM Trans. Database Syst., 6(3):387–415, 1981.
7. Hull R. Relative information capacity of simple relational database schemata. SIAM J. Comput., 15(3):856–886, 1986.
8. Vincent M. A corrected 5NF definition for relational database design. Theor. Comput. Sci., 185(2):379–391, 1997.

Normalized Discounted Cumulated Gain (nDCG)

▶ Discounted Cumulated Gain

Normalizing ORA-SS Diagrams

▶ Normal Form ORA-SS Schema Diagrams

Now in Temporal Databases

Curtis E. Dyreson[1], Christian S. Jensen[2], Richard T. Snodgrass[3]

[1]Utah State University, Logan, Utah, USA

[2]Aalborg University, Aalborg, Denmark

[3]University of Arizona, Tucson, AZ, USA

Synonyms

Current time; Current date; Current timestamp; Until changed

Definition

The word *now* is a noun in the English language that means "at the present time." This notion appears in databases in three guises. The first use of *now* is as a function within queries, views, assertions, etc. For instance, in SQL, `CURRENT_DATE` within queries, etc., returns the current date as an SQL `DATE` value; `CURRENT_TIME` and `CURRENT_TIMESTAMP` are also available. These constructs are nullary functions.

In the context of a transaction that contains more than one occurrence of these functions, the issue of which time value(s) to return when these functions are invoked becomes important. When having these functions return the same (or consistent) value, it becomes a challenge to select this time and to synchronize it with the serialization time of the transaction containing the query.

The second use is as a database variable used extensively in temporal data model proposals, primarily as timestamp values associated with tuples or attribute values in temporal database instances. As an example, within transaction-time databases, the stop time of data that has not been logically deleted and thus is current is termed "until changed." A challenging aspect of supporting this notion of *now* has been to contend with instances that contain this variable when defining the semantics of queries and modification and when supporting queries and updates efficiently, e.g., with the aid of indices.

The third use of *now* is as a database variable with a specified offset (a "now-relative value") that can be stored within an implicit timestamp or as the value of an explicit attribute. Challenges include the specification of precise semantics for database instances that contain these variables and the indexing of such instances.

Historical Background

Time variables such as *now* are of interest and indeed are quite useful in databases, including databases managed by a conventional DBMS, that record time-varying information, the validity of which often depends on the current-time value. Such databases may be found in many application areas, such as banking, inventory management, and medical and personnel records.

The SQL nullary functions CURRENT_DATE, CURRENT_TIME, and CURRENT_TIMESTAMP have been present since SQL's precursor, SEQUEL 2 [1]. The transaction stop time of *until changed* has been present since the initial definition of transaction-time databases (e.g., in the early work of Ben-Zvi [2]).

The notion of *now* and the surprisingly subtle concerns with this ostensibly simple concept permeate the literature, and yield quite interesting solutions.

Foundations

The three notions of "now" in temporal databases are explored in more detail below.

SQL Nullary Functions

The following example illustrates the uses and limitations of *now*, specifically the SQL CURRENT_DATE, CURRENT_TIME, and CURRENT_TIMESTAMP, in conventional databases. Banking applications record when account balances for customers are valid. Examine the relation AccountBalance with attributes AccountNumber, Balance, FromDate, and ToDate. To determine the balance of account 12345 on January 1, 2007, one could use a simple SQL query.

```
SELECT Balance
FROM AccountBalance
WHERE Account = 12345
  AND FromDate <= DATE '2007-01-01'
  AND DATE '2007-01-01' < ToDate
```

To determine the balance of that account *today*, the nullary function CURRENT_DATE is available.

```
SELECT Balance
FROM AccountBalance
WHERE Account = 12345
  AND FromDate <= CURRENT_DATE
  AND CURRENT_DATE < ToDate
```

Interestingly, in the SQL standard, the semantics of the function are implementation-dependent, which opens it to various interpretations: "If an SQL-statement generally contains more than one reference to one or more <datetime value functions>s then all such references are effectively evaluated simultaneously. The time of evaluation of the <datetime value function > during the execution of the SQL-statement is implementation-dependent." (This is from the SQL:1999 standard.) So an implementation is afforded considerable freedom in choosing a definition of "current," including perhaps when the statement was presented to the system, or perhaps when the database was first defined.

Transactions take time to complete. If a transaction needs to insert or modify many tuples, it may take minutes. However, from the point of view of the user, the transaction is atomic (all or nothing) and serializable (placed in a total ordering). Ideally "current" should mean "when the transaction executed," that is, the instantaneous time the transaction logically executed, consistent with the serialization order.

Say a customer opens an account and deposits $200. This transaction results in a tuple being inserted into the AccountBalance table. The transaction started on January 14 at 11:49 P.M. and committed at 12:07 A.M. (that is, starting before midnight and completing afterwards). The tuple was inserted on 11:52 P.M. Another user also created a second account with an initial balance of $500 in a transaction that started on January 14 at 11:51 P.M. and committed on 11:59 P.M., inserting a tuple into the AccountBalance relation at 11:52 P.M. If the system uses the transaction start time, the following two tuples will be in the relation.

According to these two tuples, the sum of balances on January 14 is $700. But note that though both transactions began on January 14, only the second transaction committed by midnight (also note that the second transaction is earlier in the serialization order since it commits first). Hence, the actual aggregate balance on January 14 was never $700: the balance started at $0 (assuming there were initially no other tuples), then changed to $500. Only on January 15 did it increase to $700.

Suppose that instead of using the time at which the entire transaction began, the time of the actual insert statement (e.g., for the first transaction, January 14, 11:52 P.M.) is used; then the same problem occurs.

What is desired is for a time returned by CURRENT_DATE to be consistent with the serialization order and with the commit time, which unfortunately is not known when CURRENT_DATE is executed. Lomet et al. [3] showed how to utilize the lock manager to assign a commit time in such a way that it is consistent with the serialization order as well as with dates assigned as values to prior CURRENT_DATE invocations. Specifically, each use of CURRENT_DATE, etc. defines or narrows a *possible period* during which the commit time of the transaction must reside. For CURRENT_DATE this period is the particular 24 hours of the day returned by this function. Read-write conflicts with other transactions further narrow the possible period. For example, if a particular transaction reads values written by another transaction, the transaction time of the reader must be later than that of the writer. At commit time, it is attractive to assign the earliest instant in the possible period to the transaction. Alternatively, if the possible period ever becomes empty, the transaction must be aborted. Lomet et al. outline important optimizations to render such calculations efficient.

Now in End or Stop Columns

The above discussion concerned how to determine what to store in the begin time of a tuple (e.g., in the FromDate column for the two example tuples in the table displayed above) and how to make this time consistent with that returned by CURRENT_DATE. We now consider what to store in the end time of a tuple (e.g., in the ToDate column). The validity of a tuple then starts when a deposit is made and extends until the current time, assuming no update transactions are committed. Thus, on January 16, the balance is valid from January 14 until January 16; on January 17, the balance is valid from January 14 until January 17; etc.

It is impractical to update the database each day (or millisecond) to correctly reflect the valid time of the balance. A more promising approach is to store a variable, such as *now*, in the ToDate field of a tuple, to indicate that the time when a balance is valid depends on the current time. In the example, it would be recorded on January 14 that the customer's balance of $200 is valid from January 14 through *now*. The CURRENT_DATE construct used in the above query cannot be stored in a column of an SQL table. All major commercial DBMSs have similar constructs, and impose this same restriction. The user is forced instead to store a specific time, which is cumbersome and inaccurate (Clifford et al. explain these difficulties in great detail [4]).

The solution is to allow one or more free, current-time variables to be stored in the database. Chief among these current-time variables is "*now*" (e.g., [5]), but a variety of other symbols have been used, including "–" [2], "∞" [6], "@" [7], and "*until-changed*" [8]. Such stored variables have advantages at both the semantic and implementation levels. They are expressive and space efficient and avoid the need for updates at every moment in time.

As an example, consider a variable database with the tuple ⟨*Jane*, *Assistant*, [*June 1*, *now*]⟩, with *now* being a variable. The query "List the faculty on June 15," evaluated on June 27, results in {⟨*Jane*, *Assistant*⟩}.

Now-Relative Values

A now-relative instant generalizes and adds flexibility to the variable *now* by allowing an offset from this variable to be specified. Now-relative instants can be used to more accurately record the knowledge of Jane's employment. For example, it may be that hiring changes are recorded in the database only 3 days after they take effect. Assuming that Jane was hired on June 1, the definite knowledge of her employment is accurately captured in the tuple ⟨*Jane*, *Assistant*, [*June 1*, *now* − 3 *days*]⟩ . This tuple states that Jane was an Assistant Professor from June 1 and until three days ago, but it contains no information about her employment as of, e.g., yesterday.

A now-relative instant thus includes a displacement, which is a signed duration, also termed a span, from *now*. In the example given above, the displacement is minus 3 days. Now-relative variables can be extended to be indeterminate [4], as can regular instants and the variable *now*.

AccountNumber	Balance	FromDate	ToDate
121345	200	2007-01-14	...
543121	500	2007-01-14	...

The semantics of now variables has been formalized with an *extensionalization* that maps from a *variable database level* containing variables as values to an *extensional database level*. The extensional database exhibits three key differences when compared to the variable database level. First, no variables are allowed – the extensional level is fully ground. Second, timestamps are instants rather than intervals. Third, an extensional tuple has one additional temporal attribute, called a *reference time attribute*, which may be thought of as representing the time at which a meaning was given to the temporal variables in the original tuple. Whereas the variable-database level offers a convenient representation that end-users can understand and that is amenable to implementation, the mathematical simplicity of the extensional level supports a rigorous treatment of temporal databases in terms of first order logic.

When a variable database is queried, an additional problem surfaces: what to do when a variable is encountered during query evaluation. In the course of evaluating a user-level query, e.g., written in some dialect of SQL, it is common to transform it into an internal algebraic form that is suitable for subsequent rule or cost-based query optimization. As the query processor and optimizer are among the most complex components of a database management system, it is attractive if the added functionality of current-time-related timestamps necessitates only minimal changes to these components.

Perhaps the simplest approach to supporting querying is that, when a timestamp that contains a variable is used during query processing (e.g., in a test for overlap with another timestamp), a ground version of that timestamp is created and used instead. With this approach, only a new component that substitutes variable timestamps with ground timestamps has to be added, while existing components remain unchanged.

Put differently, a *bind* operator can be added to the set of operators already present. This operator is then utilized when user-level queries are mapped to the internal representation. The operator accepts any tuple with variables. It substitutes a ground value for each variable and thus returns a ground (but still variable-level) tuple. Intuitively, the *bind* operator sets the perspective of the observer, i.e., it sets the reference time. Existing query languages generally assume that the temporal perspective of an observer querying a database is the time when the observer initiates the query.

Torp et al. has shown how to implement such variables within a database system [9].

With *now* as a database variable, the temporal extent of a tuple becomes a non-constant function of time. As most indices assume that the extents being indexed are constant in-between updates, the indexing of the temporal extents of now-relative data poses new challenges.

For now-relative transaction-time data, one may index all data that is not now-relative (i.e., has a fixed end time) in one index, e.g., an R-tree, and all data that is now-relative (i.e., the end time is *now*) in another index where only the start time is indexed.

For bitemporal data, this approach can be generalized to one where tuples are distributed among four R-trees. The idea is again to overcome the inabilities of indices to cope with continuously evolving regions, by applying transformations to the growing bitemporal extents that render them stationary and thus amenable to indexing. Growing regions come in three shapes, each with its own transformation. These transformations are accompanied by matching query transformations [10].

In another approach, the R-tree is extended to store *now* for both valid and transaction time in the index. The resulting index, termed the GR-tree, thus accommodates bitemporal regions and uses minimum bounding regions that can be either static or growing and either rectangles or stair shapes. This approach has been extended to accommodate bitemporal data that also have spatial extents [11].

Key Applications

The notion of *now* as a nullary function representing the current time is common in database applications. For relational database management, SQL offers `CURRENT_DATE`, `CURRENT_TIME`, and `CURRENT_TIMESTAMP` functions that return an appropriate SQL time value. In other kinds of database management systems, such as native XML database management systems, similar constructs can be found. For example, XQuery has a `fn:current-time()` function that returns a value of XML Schema's `xs:time` type. XQuery also has a `fn:current-date()` function.

Less common in existing database applications are the other notions of now: as a variable stored in a database to represent the ever-changing current time or as a time related to, but displaced from, the current time.

Future Directions

The convenience of using now variables poses challenges to the designers of database query languages.

The user-defined time types available in SQL-92 can be extended to store now-relative variables as values in columns. The TSQL2, language [12] does so, and also supports those variables for valid and transaction time. In TSQL2, the "bind" operation is implicit; NOBIND is provided to store variables in the database. However, such variables have yet to be supported by commercial DBMSs. It may also be expected that at least one of the three uses of *now* will re-emerge as part of a temporal extension of XQuery or a language associated with the Semantic Web.

The impact of stored variables on database storage structures and access methods is a relatively unexplored area. Such stored variables may present optimization opportunities. For example, if the optimizer knows (through attribute statistics) that a large proportion of tuples has a "to" time of *now*, it may then decide that a sort-merge temporal join will be less effective. Finally, new kinds of variables, such as *here* for spatial and spatio-temporal databases, might offer an interesting extension of the framework discussed here.

Cross-references

▶ Bi-Temporal Indexing
▶ Bitemporal interval
▶ Supporting Transaction Time Databases
▶ Temporal Concepts in Philosophy
▶ Temporal Query Languages
▶ Temporal Strata
▶ Temporal XML
▶ Time Period
▶ Transaction Time
▶ TSQL2
▶ Valid Time

Recommended Reading

1. Ben-Zvi J. The Time Relational Model. Ph.D. Dissertation, University of California, Los Angeles, 1982.
2. Bliujūtė R., Jensen C.S., Šaltenis S., and Slivinskas G. Lightweight indexing of bitemporal data. In Proc. 12th Int. Conf. on Scientific and Statistical Database Management, 2000, pp. 125–138.
3. Chamberlin D.D., Astraham M.M., Eswaran K.P., Griffiths P.P., Lorie R.A., Mehl J.W., Reisner P., and Wade B.W., SEQUEL 2: a unified approach to data definition, manipulation, and control. IBM J. Res. Dev. 20(6):560–575, 1976.
4. Finger M. Handling database updates in two-dimensional temporal logic. J. Appl. Non-Classical Logics, 2(2):201–224, 1992.
5. Clifford J., Dyreson C.E., Isakowitz T., Jensen C.S., and Snodgrass R.T. On the semantics of "now." ACM Trans. Database Syst., 22(2):171–214, June 1997.
6. Clifford J. and Tansel A.U. On an algebra for historical relational databases: two views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1985, pp. 247–265.
7. Lomet D., Snodgrass R.T., and Jensen C.S. Exploiting the lock manager for timestamping. In Proc. Int. Conf. on Database Eng. and Applications, 2005, pp. 357–368.
8. Lorentzos N.A. and Johnson R.G. Extending relational algebra to manipulate temporal data. Inf. Syst., 13(3):289–296, 1988.
9. Montague R. Formal Philosophy: Selected Papers of Richard Montague. Yale University Press, New Haven, 1974.
10. Šaltenis S., and Jensen C.S. Indexing of now-relative spatio-bitemporal data. VLDB J., 11(1):1–16, August 2002.
11. Snodgrass R.T. The temporal query language TQuel. ACM Trans. Database Syst., 12(2):247–298, June 1988.
12. Snodgrass R.T. (ed.). The TSQL2 Temporal Query Language. Kluwer, Norwell, MA, USA, 1995.
13. Torp K., Jensen C.S., and Snodgrass R.T. Modification semantics in now-relative databases. Inf. Syst., 29(78):653–683, 2004.
14. Wiederhold G., Jajodia S., and Litwin W. Integrating temporal data in a heterogeneous environment. In Temporal Databases: Theory, Design, and Implementation, Chap. 22, A. Tansel, J. Clifford, S.K. Gadia, S. Jajodia, A. Segev, R.T. Snodgrass (eds.). Benjamin/Cummings, Redwood City, CA, USA, 1993, pp. 563–579.

n-Tier Architecture

▶ Multi-Tier Architecture

Null Values

Leopoldo Bertossi
Carleton University, Ottawa, ON, Canada

Definition

Null values are used to represent *uncertain data values* in a database instance.

Key Points

Since the beginning of the relational data model, *null values* have been investigated, with the intention of capturing and representing data values that are uncertain. Depending on the intuitions and cases of uncertainty, different kinds of null values have been proposed, e.g., they may represent information that is withheld, inapplicable, missing, unknown, etc. Thus, in principle, it could be possible to find in a hypothetical database diverse classes of null values, and also several null values of the same class. However, in commercial relational DBMSs and in the SQL Standard, only a single constant, `NULL`, is used to represent the missing values.

Many semantic problems appear when null values are integrated with the rest of the relational data model, which essentially follows the semantics of predicate logic. Among them, (i) the interpretation of nulls values (for a particular intuition); (ii) the meaning of relational operations when applied to both null values and certain data values; and (iii) the characterization of consistency of databases containing null values.

Different formal semantics for null values have been proposed. A common and well-studied semantic for *incomplete databases* uses null values to represent unknown or missing values. Each null value in the database represents a whole set of possible values from the underlying data domain. The combination of concrete values that null values might take generates a class of alternative instances containing certain values. This *possible worlds semantics* makes true whatever is true in every alternative instance. However, the usage of null values in the SQL Standard and commercial DBMS still lacks a clear and complete formal semantic.

Cross-references

▶ Incomplete Information

Recommended Reading

1. Grahne G. The Problem of Incomplete Information in Relational Databases. LNCS 554, Springer-Verlag, Secaucus, NJ, USA, 1991.
2. Levene M. and Loizou G. A Guided Tour of Relational Databases and Beyond, Chap. 5. Springer, London, UK, 1999.
3. Van der Meyden, R. Logical approaches to incomplete information: a survey. In Logics for Databases and Information Systems, J. Chomicki, G. Saake (eds.). Kluwer, Boston, MA, USA, 1998, pp. 307–356.

Numeric Association Rules

▶ Quantitative Association Rules

Numerical Fact

▶ Measure

Nymity

▶ Pseudonymity

OASIS

Serguei Mankovskii
CA Labs, CA Inc, Thornhill, ON, Canada

Synonyms

Organization for the advancement of structured information standards

Definition

OASIS is a non-for-profit consortium aiming at collaborative development and approval of open international, mainly XML-based, standards.

Key Points

OASIS was founded in 1993 under the name "SGML Open." The initial goal of the organization was to develop guidelines for interoperability among products using Standard Generalized Markup Language (SGML). In 1998 it changed name to OASIS to reflect on changing scope of its technical work.

OASIS consists of an open group of member organizations whose representatives work in committees developing standards, promoting standards adoption, product interoperability and standards conformance. In 2007 OASIS had 5,000 participants representing 600 organizations and individual members in 100 countries. OASIS is governed by a member-elected Board in an annual election process. The board membership is based on the personal merits of Board nominees.

OASIS process allows participants to influence standards that affect their business, contribute to standards advancement and start new standards. The process is designed to promote industry consensus. OASIS strategy values creativity and consensus over conformity and control. It relies on the market to determine the particular approach taken in the development of sometimes overlapping standards.

OASIS maintains collaborative relationships with the International Electrotechnical Commission (IEC), International Organization for Standardization (ISO), International Telecommunication Union (ITU) and United Nations Electronic Commission for Europe (UN/ECE), and National Institute of Standards and Technology (NIST).

Among major accomplishments of the OASIS are such influential of standards as a group of ebXML standards, SAML, XACML, WSRP, WSDM, BPEL, OpenDocument, DITA, DocBook, LegalXML and others.

Cross-references

- ▶ BPEL
- ▶ DITA
- ▶ DocBook
- ▶ ebXML
- ▶ eGovernment
- ▶ Emergency Management
- ▶ LegalXML
- ▶ oBIX
- ▶ Open CSA (SCA, SDO)
- ▶ OpenDocument
- ▶ SAML
- ▶ SOA-RM
- ▶ UDDI
- ▶ WS-Security
- ▶ WSDM
- ▶ WSRP

Recommended Reading

1. OASIS. Available at: http://www.oasis-open.org

Object Constraint Language

Martin Gogolla
University of Bremen, Bremen, Germany

Synonyms

OCL

Definition

The Unified Modeling Language (UML) includes a textual language called Object Constraint Language

(OCL). OCL allows users to navigate class diagrams, to formulate queries, and to restrict class diagrams with integrity constraints. From a practical perspective, the OCL may be viewed as an object-oriented version of the Structured Query Language (SQL) originally developed for the relational data model. From a theoretical perspective, OCL may be viewed as a variant of first-order predicate logic with quantifiers on finite domains only. OCL has a well-defined syntax [1,3] and semantics [2].

Key Points

The central language features in OCL are: navigation, logical connectives, collections and collection operations.

Navigation: The navigation features in OCL allow users to determine connected objects in the class diagram by using the dot operator ".". Starting with an expression `expr` of start class `C`, one can apply a property `propC` of class `C` returning, for example, a collection of objects of class `D` by using the dot operator: `expr.propC`. The expression `expr` could be a variable or a single object, for example. The navigation process can be repeated by writing `expr.propC.propD`, if `propD` is a property of class `D`.

Logical Connectives: OCL offers the usual logical connectives for conjunction (`and`), disjunction (`or`), and negation (`not`) as well as the implication (`implies`) and a binary exclusive (`xor`). An equality check (=) and a conditional (`if then else endif`) is provided on all types.

Collections: In OCL there are three kinds of collections: sets, bags, and sequences. A possible collection element can appear at most once in a set, and the insertion order in the set does not matter. An element can appear multiple times in a bag, and the order in the bag collection does not matter. An element can appear multiple times in a sequence in which the order is significant. Bags and sequences can be converted to sets with `->asSet()`, sets and sequences to bags with `->asBag()`, and sets and bags to sequences with `->asSequence()`. The conversion to sequences assumes an order on the elements. The arrow notation is explained in more detail below.

Collection Operations: There is a large number of operations on collections in OCL. A lot of convenience and expressibility is based upon them. The most important operations on all collection types are the following: `forAll` realizes universal quantification, `exists` is existential quantification, `select` filters elements with a predicate, `collect` applies a term to each collection element, `size` determines the number of collection elements, `isEmpty` tests for emptiness, `includes` checks whether a possible element is included in the collection, and `including` builds a new collection `including` a new element.

In addition to the central language features, OCL also has special operations available only on particular collection, e.g., the operation `at` on sequences for retrieving an element by its position. All collection operations are applied with the arrow notation mentioned above. Roughly speaking, the dot notation is used when a property follows, i.e., an attribute or a role follows, and the arrow notation is employed when a collection operation follows.

Variables in collection operations: Most collection operations allow variables to be declared (possibly including a type specification), but the variable may be dropped if it is not needed.

Retrieving all Current Instances of a Class: Another important possibility is a feature to retrieve the finite set of all current instances of a class by appending `.allInstances` to the class name. In order to guarantee finite results `.allInstances` cannot be applied to data types like `String` or `Integer`.

Return types in collection operations: If the collection operations are applied to an argument of type `Set/Bag/Sequence(T)`, they behave as follows: `forAll` and exists returns a `Boolean`, `select` yields `Set/Bag/Sequence(T)`, `collect` returns `Bag/Bag/Sequence(T')`, size gives back `Integer`, `isEmpty` yields `Boolean`, `includes` returns `Boolean`, and `including` gives back `Set/Bag/Sequence(T)`.

Most notably, the operation `collect(...)` changes the type of a `Set(T)` collection to a `Bag(T')` collection. The reason for this is that term inside the `collect` may evaluate to the same result for two different collection elements. In order to reflect that the result is captured for each collection element, the result appears as often as a respective collection element exists. This convention in OCL resembles the same approach in SQL: SQL queries with the additional keyword `distinct` return a set; plain SQL queries without `distinct` return a bag. In OCL, the convention is similar: OCL expressions using the additional conversion `asSet()` as in `collect(...)->asSet()` return a set; plain `collect(...)` expressions without `asSet()` return a bag.

Cross-references

▶ Unified Modeling Language

Recommended Reading

1. OMG (ed.). OMG Object Constraint Language Specification. OMG, 2007. www.omg.org.
2. Richters M. and Gogolla M. On Formalizing the UML Object Constraint Language OCL. In Proc. 17th Int. Conf. on Conceptual Modeling. 1998, pp. 449–464.
3. Warmer J. and Kleppe A. The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley, Reading, MA, 2003.

Object Data Models

Susan D. Urban[1], Suzanne W. Dietrich[2]
[1]Texas Tech University, Lubbock, TX, USA
[2]Arizona State University, Phoenix, AZ, USA

Synonyms

ODB (object database); OODB (object-oriented database); ORDB (object-relational database)

Definition

An object data model provides support for objects as the basis for modeling in a database application. An object is an instance of a class, which is a complex type specification that defines both the state of its instance fields and the behavior provided by its methods. Object features also include a unique object identifier that can be used to refer to the object, as well as the organization of data into class hierarchies that support inheritance of state and behavior. The term object data model encompasses the data model for both object-oriented databases (OODBs) and object-relational databases (ORDBs). OODBs use an object-oriented programming language as the database language and provide inherent support for the persistence of objects with typical database functionality. ORDBs extend relational databases by providing additional support for objects.

Historical Background

The relational data model was developed in the 1970's, providing a way to organize data into tables with rows and columns [4]. Relationships between tables were defined by the concept of foreign keys, where a column (or multiple columns) in one table contained a reference to a primary key value (unique identifier) in another table. The simplicity of the relational data model was complemented by its formal foundation on set theory, thus providing powerful algebraic and calculus-based techniques for querying relational data.

Initially, relational data modeling concepts were used in business-oriented applications, where tables provided a natural structure for the organization of data. Users eventually began to experiment with the use of relational database concepts in new application domains, such as engineering design and geographic information systems. These new application areas required the use of complex data types that were not supported by the relational model. Furthermore, database designers were discovering that the process of normalizing data into table form was affecting performance for the retrieval of large, complex, and hierarchically structured data, requiring numerous join conditions to retrieve data from multiple tables. Around the same time, object-oriented programming languages (OOPLs) were also beginning to develop, defining the concept of user-defined classes, with instance fields, methods, and encapsulation for information hiding [14].

The OOPL approach of defining object structure together with object behavior eventually provided the basis for the development of Object-Oriented Database Systems (OODBs) in the mid-1980's. The *Object-Oriented Database System Manifesto*, written by leading researchers in the database field, was the first document to fully outline the characteristics of OODB technology [1]. OODBs provided a revolutionary concept for data modeling, with data objects organized as instances of user-defined classes. Classes were organized into class hierarchies, supporting inheritance of attributes and behavior. OODBs differed from relational technology through the use of internal object identifiers, rather than foreign keys, as a means for defining relationships between classes. OODBs also provided a more seamless integration of database and programming language technology, resolving the impedance mismatch problem that existed for relational database systems. The impedance mismatch problem refers to the disparity that exists between set-oriented relational database access and iterative one-record-at-a-time host language access. In the OODB paradigm, the OOPL provides a uniform, object-oriented view of data, with a single language for accessing the database and implementing the database application.

The relational database research community responded to the development of OODBs with the *Third Generation Database System Manifesto*, defining the manner in which relational technology can be extended to support object-oriented capabilities [13]. Rowe and Stonebraker developed Postgres as the first object-relational database system (ORDB), illustrating an evolutionary approach to integrating object-oriented and relational concepts [10]. ORDB concepts parallel those found in OODBs, with the notions of user-defined data types, object tables formed from user-defined types, hierarchies of user-defined types and object tables, rows of object tables with internal object identifiers, and relationships between object tables that use object identifiers as references.

Today, several OODB products exist in the market, and most relational database products provide some form of ORDB support. The following section elaborates on the common features of object data models and then differentiates between OODB and ORDB modeling concepts.

Foundations

Characteristics of Object Data Models

An object is one of the most fundamental concepts of the object data model, where an object represents an entity of interest in a specific application. An object has state, describing the specific structural properties of the object. An object also has behavior, defining the methods that are used to manipulate the object. Each method has a signature that includes the method name as well as the method parameters and types. The state and behavior of an object is expressed through an object type definition, which provides an interface for the object. Objects of the same interface are collected into a class, where each object is viewed as an instance of the class. A class definition supports the concept of encapsulation, separating the specification of a class from the implementation of its methods. The implementation of a method can therefore change without affecting the class interface and the way in which the interface is used in application code.

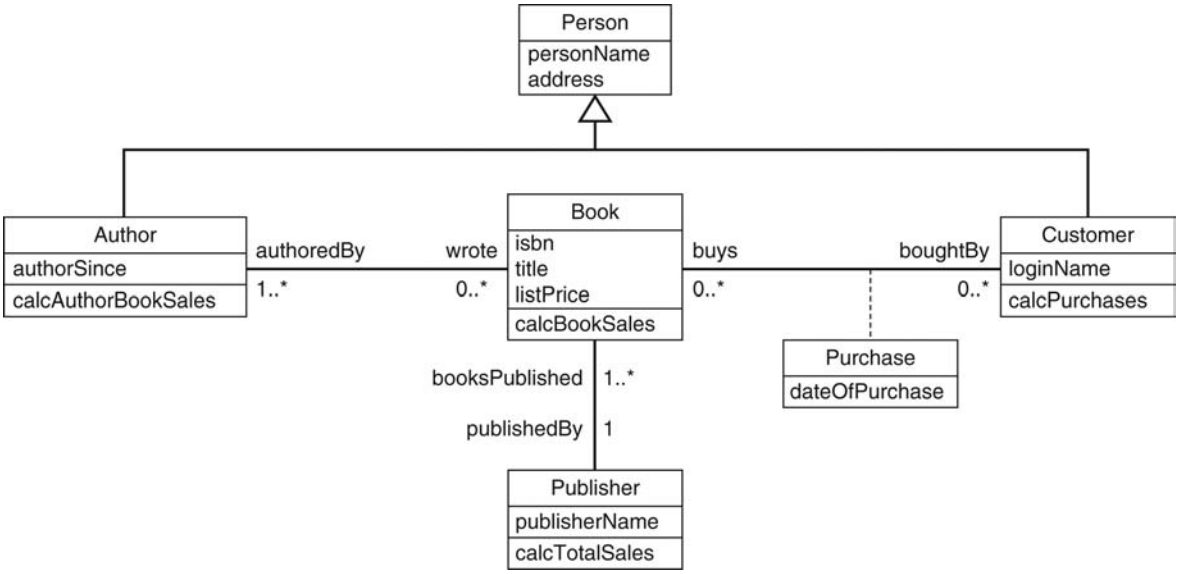
When an object of a class is instantiated, the object is assigned a unique, internal object identifier, or oid [6]. An oid is immutable, meaning that the value of the identifier cannot be changed. The state of an object, on the other hand, is mutable, meaning that the values of object properties can change. In an object data model, object identity is used as the basis for defining relationships between classes, instead of using object state, as in the relational model. As a result, the values of object properties can freely change without affecting the relationships that exist between objects. Object-based relationships between classes are referred to as object references.

Classes in an object model can be organized into class hierarchies, defining superclass and subclass relationships between classes. A class hierarchy allows for the inheritance of the state and behavior of a class, allowing subclasses to inherit the properties and methods of its superclasses while extending the subclass with additional properties or behavior that is specific to the subclass. Inheritance hierarchies provide a powerful mechanism to represent generalization/specialization relationships between classes, which simplify the specification of an object schema, as well as queries over the schema.

As an example of the above concepts, consider the Publisher application described in Fig. 1 using a Unified Modeling Language (UML) class diagram [11]. A Book is a class that is based on an object type that defines the state of a book (isbn, title, and listPrice), as well as the behavior of a book (the method calcBookSales for calculating the total sales of a book based on customer purchases). Publisher, Person, Author, and Customer are additional classes, also having state and behavior. Since authors and customers are specific types of people, the Author and Customer classes are defined to be subclasses of Person. Since personName and address are common to authors and customers, these attributes are defined at the Person level and inherited by instances of the Author and Customer classes. Furthermore, Author introduces additional state and behavior that is specific to authors, defining the date (authorSince) when an author first wrote a book as well as a method (calcAuthorBookSales) for calculating the total sales of the author's books. The Customer class similarly introduces state and behavior that is specific to a customer.

Relationships are also defined between the classes of the application:

- A book is authored by one or more authors; an author writes many books
- A book is published by one publisher; a publisher publishes many books
- A book is bought by many customers; a customer buys many books, also recording the date of each purchase



Object Data Models. Figure 1. The publisher object data model.

For each relationship, specific instances of each class are related based on the object identity of each instance. For example, a book will establish a relationship to the publisher of the book using the oid of the publisher. In the relational model, the publisher name would be used as a foreign key to establish the relationship. If the publisher name changes, then the change in name must be propagated to the book that references the publisher. In the object data model, such changes in state do not affect relationships between objects since the relationship is based on an immutable, internal object identity.

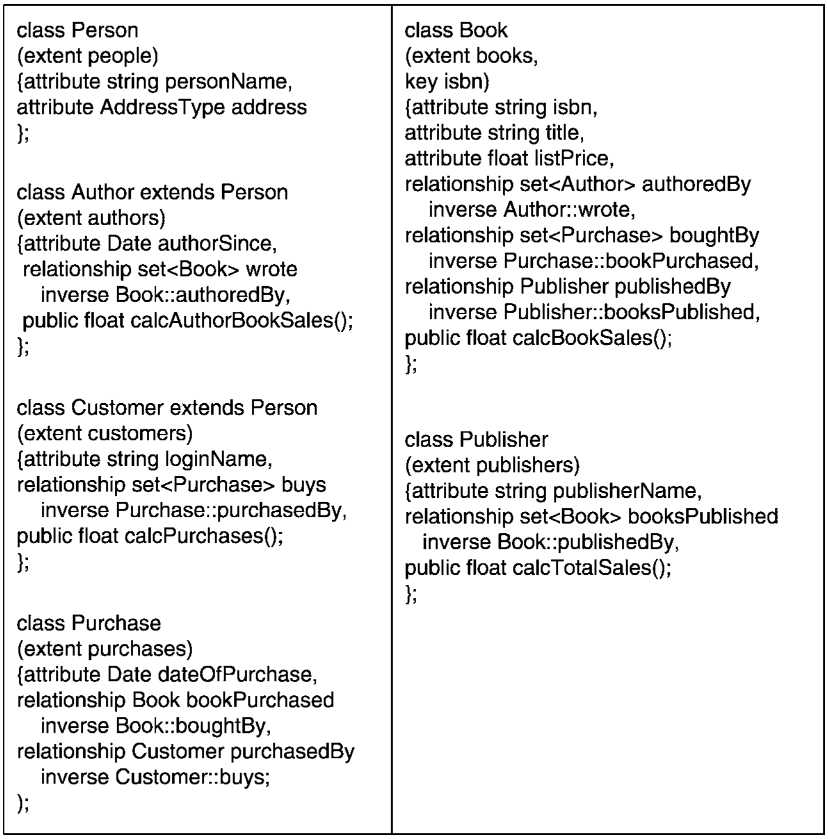
A generic object model, such as the one shown in Fig. 1, can be mapped to either an object-oriented data model or an object-relational data model. The following subsections use the Publisher application in Fig. 1 to illustrate and explain OODB and ORDB approaches to object data modeling.

Object-Oriented Data Model

An object-oriented database (OODB) is a term typically used to refer to a database that uses objects as a building block and an object-oriented programming language as the database language. The database system supports the persistence of objects along with the features of concurrency and recovery control with efficient access and an ad hoc query language.

The Object Data Standard [2] developed as a standard to describe an object model, including a definition language for an object schema, and an ad-hoc query language. The object model supports the specification of classes having attributes and relationships between objects and the behavior of the class with methods. The Object Definition Language (ODL) provides a standard language for the specification of an object schema including properties and method signatures. A property is either an attribute, representing an instance field that describes the object, or a relationship, representing associations between objects. In ODL, relationships represent bidirectional associations with the database system being responsible for maintaining the integrity of the inverse association. An attribute can be used to define a unidirectional association. If needed, the association can be derived in the other direction using a method specification. The decision is based on trade-offs of storing and maintaining the association versus deriving the inverse direction on demand.

Fig. 2 provides an ODL specification of the Publisher application. Each class has a named extent, which represents the set of objects of that type. The Author and Customer classes inherit from the Person class, extending each subclass with specialized attributes. The Book class has the isbn attribute that forms a key, being a unique value across all books. The association between Author and Book is represented as an inverse relationship, and the cardinality of the association is many-to-many since an author can write many books and a book can be written by multiple authors. The set collection type models multiple



```
class Person
(extent people)
{attribute string personName,
attribute AddressType address
};

class Author extends Person
(extent authors)
{attribute Date authorSince,
 relationship set<Book> wrote
    inverse Book::authoredBy,
 public float calcAuthorBookSales();
};

class Customer extends Person
(extent customers)
{attribute string loginName,
relationship set<Purchase> buys
    inverse Purchase::purchasedBy,
public float calcPurchases();
};

class Purchase
(extent purchases)
{attribute Date dateOfPurchase,
relationship Book bookPurchased
    inverse Book::boughtBy,
relationship Customer purchasedBy
    inverse Customer::buys;
);
```

```
class Book
(extent books,
key isbn)
{attribute string isbn,
attribute string title,
attribute float listPrice,
relationship set<Author> authoredBy
   inverse Author::wrote,
relationship set<Purchase> boughtBy
   inverse Purchase::bookPurchased,
relationship Publisher publishedBy
   inverse Publisher::booksPublished,
public float calcBookSales();
};


class Publisher
(extent publishers)
{attribute string publisherName,
relationship set<Book> booksPublished
   inverse Book::publishedBy,
public float calcTotalSales();
};
```

Object Data Models. Figure 2. ODL schema of the publisher application.

books and authors. Since the Purchase association class from Fig. 1 has an attribute describing the association, Purchase is modeled in ODL using reification, which is the process of transforming an abstract concept, such as an association, into a class. As a result, the Purchase class in Fig. 2 represents the many-to-many association between Book and Customer. Each instance of the Purchase class represents the purchase of a book by a customer. The Purchase class has the dateOfPurchase instance field, as well as two relationships indicating which Book (bookPurchased) and which Customer (purchasedBy) is involved in the purchase. The inverse relationships in Book (boughtBy) and Customer (buys) are related to instances of the Purchase class.

This ODL specification forms the basis of the definition of the object schema within the particular OOPL used with the OODB, such as C++, Java, and Smalltalk. The specification of the schema and the method implementation using a given OOPL is known as a language binding. In some OODB products, the ODL specification of the properties of the class are used to automatically generate the definition of the schema for the OOPL being used.

The standard also includes a declarative query language known as the Object Query Language (OQL). The OQL is based on the familiar select-from-where syntax of SQL. The select clause defines the structure of the result of the query. The from clause specifies variables that range over collections within the schema, such as a class extent or a multivalued property. The where clause provides restrictions on the properties of the objects that are to be included in the result. Object references are traversed through the use of dot notation for single-valued properties and through the from clause for multivalued properties.

Consider a simple query that finds the name of a publisher of a book given its isbn:

```
select b.publishedBy.publisherName
from books b
where b.isbn = "0-13-042898-1";
```

This OQL query looks quite similar to SQL. In the from clause, the alias b ranges over the books extent.

The where clause locates the book of interest. The select clause provides a path expression that navigates through the publishedBy single-valued property to return the name of the publisher.

Consider another query that finds the title and sales for books published by Springer-Verlag:

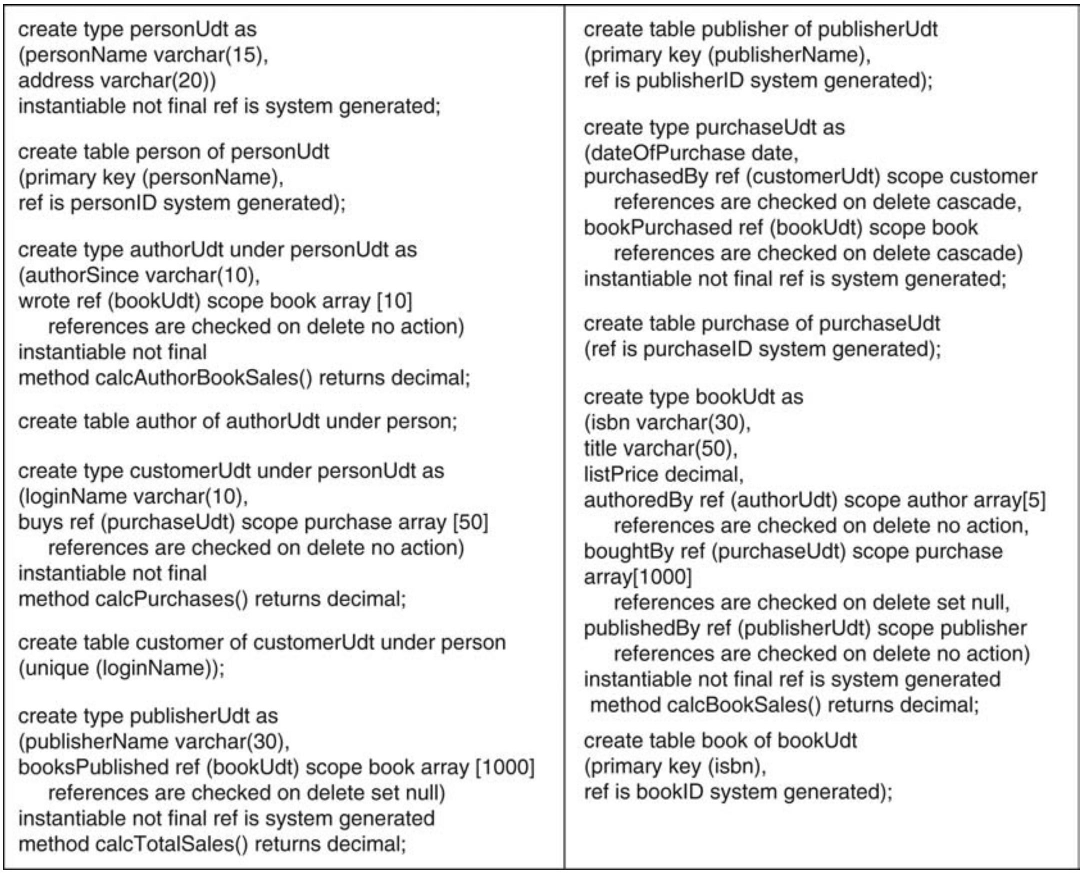
```
select title: b.title, sales: b.calcBookSales()
from p in publishers, b in p.booksPublished
where p.publisherName = ‘‘Springer-Verlag’’
```

This query illustrates the alternative syntax for the alias in the from clause, using the syntax "variable in collection". The alias p ranges over the publishers extent, whereas the alias b ranges over the multivalued relationship booksPublished of each publisher that satisfies the where condition. The select clause returns the name of each field and its value, where sales returns the results of a method call.

Object-Relational Data Model

An object-relational database (ORDB) refers to a relational database that has evolved by extending its data model to support user-defined types along with additional object features. An ORDB supports the traditional relational table in addition to introducing the concept of a typed table, which is similar to a class in an OODB. A typed table is created based on a user-defined type (UDT), which provides a way to define complex types with support for encapsulation. UDTs and their corresponding typed tables can be formed into class hierarchies with inheritance of state and behavior. The rows (or instances) of a typed table have object identifiers that are referred to as object references. Object references can be used to define relationships between tables that are based on object identity.

Figure 3 presents an ORDB schema of the Publisher application that is defined using the object-relational extensions to the SQL standard. The type personUdt is



```
create type personUdt as
(personName varchar(15),
address varchar(20))
instantiable not final ref is system generated;

create table person of personUdt
(primary key (personName),
ref is personID system generated);

create type authorUdt under personUdt as
(authorSince varchar(10),
wrote ref (bookUdt) scope book array [10]
   references are checked on delete no action)
instantiable not final
method calcAuthorBookSales() returns decimal;

create table author of authorUdt under person;

create type customerUdt under personUdt as
(loginName varchar(10),
buys ref (purchaseUdt) scope purchase array [50]
   references are checked on delete no action)
instantiable not final
method calcPurchases() returns decimal;

create table customer of customerUdt under person
(unique (loginName));

create type publisherUdt as
(publisherName varchar(30),
booksPublished ref (bookUdt) scope book array [1000]
   references are checked on delete set null)
instantiable not final ref is system generated
method calcTotalSales() returns decimal;

create table publisher of publisherUdt
(primary key (publisherName),
ref is publisherID system generated);

create type purchaseUdt as
(dateOfPurchase date,
purchasedBy ref (customerUdt) scope customer
   references are checked on delete cascade,
bookPurchased ref (bookUdt) scope book
   references are checked on delete cascade)
instantiable not final ref is system generated;

create table purchase of purchaseUdt
(ref is purchaseID system generated);

create type bookUdt as
(isbn varchar(30),
title varchar(50),
listPrice decimal,
authoredBy ref (authorUdt) scope author array[5]
   references are checked on delete no action,
boughtBy ref (purchaseUdt) scope purchase
array[1000]
   references are checked on delete set null,
publishedBy ref (publisherUdt) scope publisher
   references are checked on delete no action)
instantiable not final ref is system generated
 method calcBookSales() returns decimal;

create table book of bookUdt
(primary key (isbn),
ref is bookID system generated);
```

Object Data Models. Figure 3. ORDB schema of publisher application.

an example of specifying a UDT. The UDT defines the structure of the type by identifying attributes together with their type definitions. The phrase "instantiable not final ref is system generated" defines three properties of the type:

1. "Instantiable" indicates that the type supports a constructor function for the creation of instances of the type. The phrase "not instantiable" can be used in the case where the type has a subtype and instances can only be created at the subtype level.
2. "Not final" indicates that the type can be specialized into a subtype. The phrase "final" can be used to indicate that a type cannot be further specialized.
3. "Ref is system generated" indicates that the database system is responsible for automatically generating an internal object identifier. The SQL standard supports other options for the generation of object identifiers, which include user-specified object-identifiers as well as identifiers that are derived from other attributes.

Definition of the personUdt type is followed by the specification of the person typed table, which is based on the personUdt type. The person typed table automatically acquires columns for each of the attributes defined in personUdt. In addition, the person typed table has a column for an object identifier that is associated with every row in the table. The phrase "ref is personID" defines that the name of the object identifier column is personID. The definition of a typed table can add constraints to the columns that are defined in the type associated with the table. For example, personName is defined to be a primary key in the person typed table.

The authorUdt type is defined as a subtype of personUdt, as indicated by the "under personUdt" clause. In addition to defining the structure of the type, authorUdt also defines behavior with the definition of the calcAuthorBookSales method. Since authorUdt is a subtype of personUdt, authorUdt will inherit the object identifier (personID) defined in personUdt. For consistency, the author table is also defined to be a subtable of the person object table. The typed table hierarchy therefore parallels the UDT hierarchy. In a similar manner, customerUdt is defined to be a subtype of personUdt and the customer typed table, based on customerUdt, is defined to be a subtable of the person table. UDTs and typed tables are also defined for the Book and Publisher classes from Fig. 1, as well as the (reified implementation of the) Purchase association class.

Figure 3 also illustrates the use of object references to represent identity-based relationships between UDTs. Recall from the object data model in Fig. 1 that a book is published by one publisher; a publisher publishes many books. In an ORDB, this relationship is established through the use of reference types. In the bookUdt, the publishedBy attribute has the type ref (publisherUdt), indicating that the value of publishedBy is a reference to the object identifier (publisherID) of a publisher. In the inverse direction, the type of booksPublished in the publisherUdt is an array of ref(bookUdt), indicating that booksPublished is an array of object references to books. Each attribute definition includes a scope clause and a "references are checked" clause. Since a UDT can be used to define multiple tables, the scope clause defines the table of the object reference. The references clause specifies the same options for referential integrity of object references as originally defined for traditional relational tables.

To establish the fact that a book is published by a specific publisher, the object identifier of publisher is retrieved to create the relationship:

```
update book
set publishedBy = (select publisherID
                   from publisher
                   where publisherName =
                   ''Prentice Hall'')
where isbn = ''0-13-042898-1'';
```

A similar update statement can be used to establish the relationship in the inverse direction by adding the book oid to the array of object references of the publisher.

References can be traversed to query information about relationships. For example, to return the name of the publisher of a specific book, the following query can be used:

```
select publishedBy.publisherName
from book
where isbn = ''0-13-042898-1'';
```

The dot notation in the select clause performs an implicit join between the book table and the publisher table, returning the name of the publisher. The deref() function can also be used to retrieve the entire structured type associated with a reference value. For example, the following query will return the full instance of the publisherUdt type, rather than just the publisherName:

```
select deref(publishedBy)
from book
where isbn = ‘‘0-13-042898-1’’;
```

In this case, the result of the query is a value of type publisherUdt, containing the publisher name and the array of references to books published by the publisher.

Key Applications

Computer-Aided Design, Geographic Information Systems, Computer-Aided Software Engineering, Embedded Systems, Real-time Control Systems.

Cross-references

- ▶ Conceptual Schema Design
- ▶ Database Design
- ▶ Extended Entity-Relationship Model
- ▶ OQL
- ▶ Relational Model
- ▶ Semantic Data Model
- ▶ Unified Modeling Language

Recommended Reading

1. Atkinson M., Bancilhon F., DeWitt D., Dittrich K., Maier D., and Zdonik S. The Object-Oriented Database System Manifesto. In Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases, North Holland, 1990.
2. Cattell R.G.G., Barry D.K., Berler M., Eastman J., Jordan D., Russell C., Schadow O., Stanienda T., and Velez F. (eds.). The Object Data Standard: ODMG 3.0 Morgan Kaufmann, San Mateo, CA, 2000.
3. Chaudhri A. and Zicari R. (eds.). Succeeding with Object Databases: A Practical Look at Today’s Implementations with Java and XML. J. Wiley, New York, 2000.
4. Codd E.F. A relational model of data for large shared data banks. Comm. ACM, 13(6), 1970.
5. Dietrich S.W. and Urban S.D. An Advanced Course in Database Systems: Beyond Relational Databases. Prentice Hall, Upper Saddle River, NJ, 2005.
6. Koshafian S. and Copeland G. Object identity. ACM SIGPLAN Not., 20(11), 1986.
7. Loomis M.E.S. and Chaudhri A. (eds.). Object Databases in Practice: Prentice Hall, Upper Saddle River, NJ, 1997.
8. Melton J. Advanced SQL:1999: Understanding Object-Relational and Other Advanced Features. Morgan Kaufmann, San Mateo, CA, 2002.
9. Object Database Management Systems: The Resource Portal for Education and Research, http://odbms.org/
10. Rowe L. and Stonebraker M. The Postgres Data Model. In Proc. 13th Int. Conf. on Very Large Data Bases. 1987.
11. Rumbaugh J., Jacobson I., and Booch G. The Unified Modeling Language Reference Manual. Addison-Wesley, Reading, MA, 1991.
12. Stonebraker M. Object-Relational DBMSs: The Next Great Wave. Morgan Kaufmann, San Mateo, CA, 1995.
13. Stonebraker M., Rowe L., Lindsay B., Gray J., Carey M., Brodie M., Bernstein P., and Beech D. Third generation database system manifesto. ACM SIGMOD Rec., 19(3), 1990.
14. Stroustrup B. The C++ Programming Language, 3rd edn. Reading, MA. Addison-Wesley, Reading, MA, 1997.
15. Zdonik S.B. and Maier D. Readings in Object-Oriented Database Systems. Morgan Kaufmann, San Mateo, CA, 1990.

Object Detection and Recognition

▶ Automatic Image Annotation

Object Flow Diagrams

▶ Activity Diagrams

Object Identification

▶ Object Recognition

Object Identification

▶ Semantic Data Integration for Life Science Entities

Object Identifier

▶ Object Identity

Object Identity

Susan D. Urban[1], Suzanne W. Dietrich[2]

[1]Texas Tech University, Lubbock, TX, USA

[2]Arizona State University, Phoenix, AZ, USA

Synonyms

Object identifier; Oid; Object reference

Definition

Object identity is a property of data that is created in the context of an object data model, where an object is assigned a unique internal object identifier, or oid. The object identifier is used to define associations between objects and to support retrieval and comparison of object-oriented data based on the internal identifier rather than the attribute values of an object.

Key Points

In an object data model, an object is created as an instance of a class. An object has an object identifier as well as a state. An object identifier is immutable, meaning that the value of the object identifier cannot change over the lifetime of the object. The state, on the other hand, is mutable, representing the attributes that describe the object and the relationships that define associations among objects. Relationships in an object data model are defined using object references based on internal object identifiers rather than attribute values as in a relational data model. As a result, the attribute values of an object can freely change without affecting identity-based relationships.

Variables that contain object references can be compared using either object identity or object equality. Two object references are identical if they contain the same object identifiers. In contrast, two object references, that possibly contain different object identifiers, are equal if the values of attributes and relationships in each object state are identical. Shallow equality is the process of comparing the immediate values of attributes and relationships. Deep equality involves the traversal of object references in the comparison process. Query languages for objects must incorporate operators to distinguish between object identity and object equality in the specification of object queries.

Cross-references

- ▶ Conceptual Schema Design
- ▶ Extended Entity-Relationship Model
- ▶ Object Data Models
- ▶ Object-Role Modeling
- ▶ Semantic Data Model
- ▶ Unified Modeling Language

Recommended Reading

1. Beeri C. and Thalheim B. Identification as a Primitive of Database Models. In Proc. 7th Int. Workshop on Foundations of Models and Languages for Data and Objects, 1999.
2. Koshafian S. and Copeland G. Object identity. ACM SIGPLAN Not., 20(11), 1986.

Object Labeling

▶ Object Recognition

Object Monitor

▶ Transactional Middleware

Object Query Language

▶ OQL

Object Recognition

Ming-Hsuan Yang
University of California at Merced, Merced, CA, USA

Synonyms

Object identification; Object labeling

Definition

Object recognition is concerned with determining the identity of an object being observed in the image from a set of known labels. Oftentimes, it is assumed that the object being observed has been detected or there is a single object in the image.

Historical Background

As the holy grail of computer vision research is to tell a story from a single image or a sequence of images, object recognition has been studied for more than four decades [9,22]. Significant efforts have been spent to develop representation schemes and algorithms aiming at recognizing generic objects in images taken under different imaging conditions (e.g., viewpoint, illumination, and occlusion). Within a limited scope of distinct objects, such as handwritten digits, fingerprints, faces, and road signs, substantial success has been achieved. Object recognition is also related to content-based image retrieval and multimedia indexing as a number of generic objects can be recognized. In addition, significant progress towards object categorization from images has been made in the recent years [17]. Note that object recognition has also been studied extensively in psychology, computational neuroscience and cognitive science [4,9].

Foundations

Object recognition is one of the most fascinating abilities that humans easily possess since childhood. With a simple glance of an object, humans are able to tell its identity or category despite of the appearance variation due to change in pose, illumination, texture, deformation, and under occlusion. Furthermore, humans can easily generalize from observing a set of objects to recognizing objects that have never been seen before. For example, kids are able to generalize the concept of "chair" or "cup" after seeing just a few examples. Nevertheless, it is a daunting task to develop vision systems that match the cognitive capabilities of human beings, or systems that are able to tell the specific identity of an object being observed. The main reasons can be attributed to the following factors: relative pose of an object to a camera, lighting variation, and difficulty in generalizing across objects from a set of exemplar images. Central to object recognition systems are how the regularities of images, taken under different lighting and pose conditions, are extracted and recognized. In other words, all the algorithms adopt certain representations or models to capture these characteristics, thereby facilitating procedures to tell their identities. In addition, the representations can be either 2D or 3D geometric models. The recognition process, either generative or discriminative, is then carried out by matching the test image against the stored object representations or models.

Geometry-Based Approaches

Early attempts at object recognition were focused on using geometric models of objects to account for their appearance variation due to viewpoint and illumination change. The main idea is that the geometric description of a 3D object allows the projected shape to be accurately predicated in a 2D image under projective projection, thereby facilitating recognition process using edge or boundary information (which is invariant to certain illumination change). Much attention was made to extract geometric primitives (e.g., lines, circles, etc.) that are invariant to viewpoint change [13]. Nevertheless, it has been shown that such primitives can only be reliably extracted under limited conditions (controlled variation in lighting and viewpoint with certain occlusion). Mundy provides an excellent review on geometry-based object recognition research [12].

Appearance-Based Algorithms

In contrast to early efforts on geometry-based object recognition, most recent efforts have been centered on appearance-based techniques as advanced feature descriptors and pattern recognition algorithms are developed [8]. Most notably, the eigenface method has attracted much attention as it is one of the first face recognition systems that are computationally efficient and relatively accurate [21]. The underlying idea of this approach is to compute eigenvectors from a set of vectors where each one represents one face image as a raster scan vector of gray-scale pixel values. Each eigenvector, dubbed an eigenface, captures certain variance among all the vectors, and a small set of eigenvectors captures almost all the appearance variation of face images in the training set. Given a test image represented as a vector of gray-scale pixel values, its identity is determined by finding the nearest neighbor of this vector after being projected onto a subspace spanned by a set of eigenvectors. In other words, each face image can be represented by a linear combination of eigenfaces with minimum error (often in the L2 sense), and this linear combination constitutes a compact reorientation. The eigenface approach has been adopted in recognizing generic objects across different viewpoints [14] and modeling illumination variation [2].

As the goal of object recognition is to tell one object from the others, discriminative classifiers have been used to exploit the class specific information. Classifiers such as k-nearest neighbor, neural networks with radial basis function (RBF), dynamic link architecture, Fisher linear discriminant, support vector machines (SVM), sparse network of Winnows (SNoW), and boosting algorithms have been applied to recognize 3D objects from 2D images [16,6,1,18,19]. While appearance-based methods have shown promising results in object recognition under viewpoint and illumination change, they are less effective in handling occlusion. In addition, a large set of exemplars needs to be segmented from images for generative or discriminative methods to learn the appearance characteristics. These problems are partially addressed with parts-based representation schemes.

Feature-Based Algorithms

The central idea of feature-based object recognition algorithms lies in finding interest points, often occurred at intensity discontinuity, that are invariant to change due to scale, illumination and affine transformation (a brief review on interest point operators can be found in [8]). The scale-invariant feature transform (SIFT) descriptor is arguably one of the most widely used feature representation schemes for vision applications [8]. The SIFT approach uses extrema in scale space for automatic scale selection with a pyramid of difference of Gaussian filters, and keypoints with low contrast or poorly localized on an edge are removed. Next, a consistent orientation is assigned to each keypoint and its magnitude is computed based on the local image gradient histogram, thereby achieving invariance to image rotation. At each keypoint descriptor, the contribution of local image gradients are sampled and weighted by a Gaussian, and then represented by orientation histograms. For example, the 16 × 16 sample image region and 4 × 4 array of histograms with 8 orientation bins are often used, thereby providing a 128-dimensional feature vector for each keypoint. Objects can be indexed and recognized using the histograms of keypoints in images. Numerous applications have been developed using the SIFT descriptors, including object retrieval [15,20], and object category discovery [5].

Although the SIFT approach is able to extract features that are insensitive to certain scale and illumination changes vision applications with large base line changes entail the need of affine invariant point and region operators [11]. A performance evaluation among various local descriptors can be found in [10], and a study on affine region detectors is presented in [11]. Finally, SIFT-based methods are expected to perform better for objects with rich texture information as sufficient number of keypoints can be extracted. On the other hand, they also require sophisticated indexing and matching algorithms for effective object recognition [8,17].

Key Applications

Biometric recognition, and optical character/digit/document recognition are arguably the most widely used applications. In particular, face recognition has been studied extensively for decades and with large scale ongoing efforts [23]. On the other hand, biometric recognition systems based on iris or fingerprint as well as as handwritten digit have become reliable technologies [3,7]. Other object recognition applications include surveillance, industrial inspection, content-based image retrieval (CBIR), robotics, medical imaging, human computer interaction, and intelligent vehicle systems, to name a few.

Future Directions

With more reliable representation schemes and recognition algorithms being developed, tremendous progress has been made in the last decade towards recognizing objects under variation in viewpoint, illumination and under partial occlusion. Nevertheless, most working object recognition systems are still sensitive to large variation in illumination and heavy occlusion. In addition, most existing methods are developed to deal with rigid objects with limited intraclass variation. Future research will continue searching for robust representation schemes and recognition algorithms for recognizing generic objects.

Data Sets

Numerous face image sets are available on the web

- FERET face data set: http://www.itl.nist.gov/iad/humanid/feret/
- UMIST data set: http://images.ee.umist.ac.uk/danny/database.html
- Yale data set: http://cvc.yale.edu/projects/yalefacesB/yalefacesB.html
- AR data set: http://cobweb.ecn.purdue.edu/%7Ealeix/aleix_face_DB.html
- CMU PIE data set: http://www.ri.cmu.edu/projects/project_418.html

There are several large data sets for object recognition experiments,

- COIL data set: http://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php
- CalTech data sets: http://www.vision.caltech.edu/html-files/archive.html
- PASCAL visual object classes: http://www.pascal-network.org/challenges/VOC/

URL to Code

There are a few excellent short courses on object recognition in recent conferences available on the web.

- "Recognition and matching based on local invariant features" by Schmid and Lowe in IEEE Conference on Computer Vision and Pattern Recognition 2003: http://lear.inrialpes.fr/people/schmid/cvpr-tutorial03/
- "Learning and recognizing object categories" by Fei-Fei, Fergus and Torralba in IEEE International Conference on Computer Vision 2005:http://people.csail.mit.edu/torralba/shortCourseRLOC/
- "Recognizing and Learning Object Categories: Year 2007" by Fei-Fei, Fergus and Torralba in IEEE Conference on Computer Vision and Pattern Recognition 2005: http://people.csail.mit.edu/torralba/shortCourseRLOC/

Sample code for face recognition and SIFT descriptors:

- Face recognition: http://www.face-rec.org/
- Lowe's sample SIFT code: http://www.cs.ubc.ca/~lowe/keypoints/
- MATLAB implementation of SIFT descriptors by Vedaldi: http://vision.ucla.edu/~vedaldi/code/sift/sift.html
- libsift by Nowozin: http://user.cs.tu-berlin.de/~nowozin/libsift/

Grand challenge in object recognition:

- NIST face recognition grand challenge: http://www.frvt.org/FRGC/
- NIST multiple biometric grand challenge: http://face.nist.gov/mbgc/
- PASCAL visual object classes challenge 2007: http://www.pascal-network.org/challenges/VOC/voc2007/index.html

Cross-references

▶ Object Detection and Recognition

Recommended Reading

1. Belhumeur P., Hespanha J., and Kriegman D. Eigenfaces vs. fisherfaces: recognition using class specific linear projection. IEEE Trans. Pattern Analy. Machine Intell., 19(7):711–720, 1997.
2. Belhumeur P. and Kriegman D. What is the Set of Images of an Object under All Possible Illumination Conditions. Int. J. Comput. Vision, 28(3):1–16, 1998.
3. Daugman J. Probing the uniqueness and randomness of iris-codes: Results from 200 billion iris pair comparisons. In Proc. IEEE, 94(11):1927–1935, 2006.
4. Edelman S. Representation and recognition in vision. MIT, Cambridge, MA, 1999.
5. Fergus R., Perona P., and Zisserman A. Object class recognition by unsupervised scale-invariant learning. In Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition. 2003, pp. 264–271.
6. Lades M., Vorbrüggen J.C., Buhmann J., Lange J., von der Malsburg C., Würtz R.P., and Konen W. Distortion Invariant Object Recognition in the Dynamic Link Architecture. IEEE Trans. Comput., 42:300–311, 1993.
7. Lecun Y., Bottou L., Bengio Y., and Haffner P. Gradient-based learning applied to document recognition. In Proc. IEEE, 86(11):2278–2324, 1998.
8. Lowe D. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision, 60(2):91–110, 2004.
9. Marr D. Vision. W.H. Freeman and Company, San Francisco, CA, USA, 1982.
10. Mikolajczyk K. and Schmid C. A performance evaluation of local descriptors. IEEE Trans. Pattern Analy. Machine Intell., 27(10):1615–1630, 2005.
11. Mikolajczyk K., Tuytelaars T., Schmid C., Zisserman A., Matas J., Schaffalitzky F., Kadir T., and Van Gool L. A comparison of affine region detectors. Int. J. Comput. Vision, 65(1/2):43–72, 2006.
12. Mundy J. Object recognition in the geometric era: a retrospective. In Toward category-level object recognition. J. Ponce, M. Hebert, C. Schmid, and A. Zisserman (eds.). Springer, Berlin, 2006, pp. 3–29.
13. Mundy J. and Zisserman A. Geometric invariance in computer vision. MIT, Cambridge, MA, 1992.
14. Murase H. and Nayar S.K. Visual learning and recognition of 3-D objects from appearance. Int. J. Comput. Vision, 14:5–24, 1995.
15. Nister D. and Stewenius H. Scalable recognition with a vocabulary tree. In Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition. 2006, pp. 2161–2168.
16. Poggio T. and Edelman S. A Network that Learns to Recognize 3D Objects. Nature, 343:263–266, 1990.
17. Ponce J., Hebert M., Schmid C., and Zisserman A. Toward category-level object recognition. Springer, Berlin, 2006.
18. Pontil M. and Verri A. Support Vector Machines for 3D Object Recognition. IEEE Trans. Pattern Analy. Machine Intell., 20(6):637–646, 1998.
19. Roth D., Yang M.-H., and Ahuja N. Learning to Recognize Objects. Neural Comput., 14(5):1071–1104, 2002.
20. Sivic J. and Zisserman A. Video Google: a text retrieval approach to object matching in videos. In Proc. 9th IEEE Conf. Computer Vision. 2003, pp. 1470–1477.
21. Turk M. and Pentland A. Eigenfaces for recognition. J. Cognitive Neurosci., 3(1):71–86, 1991.
22. Ullman S. High-level vision: Object recognition and visual recognition. MIT, Cambridge, MA, 1996.
23. Zhao W., Chellappa R., Rosenfeld A., and Phillips J.P. Face recognition: A literature survey. ACM Comput. Surv., 35(4):399–458, 2003.

Object Reference

▶ Object Identity

Object Relationship Attribute Data Model for Semi-structured Data

Gillian Dobbie[1], Tok Wang Ling[2]
[1]University of Auckland, Auckland, New Zealand
[2]National University of Singapore, Singapore

Synonyms

ORA-SS data model; ORA-SS schema diagram

Definition

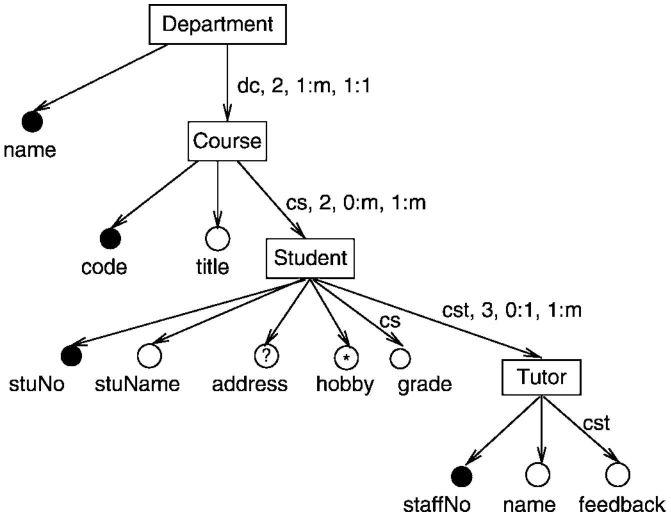
When a database schema is designed, a data model is initially used to model the real world constraints that are taken into account in the design of the schema. For semi-structured database design, it is necessary to capture the following constraints: object classes, n-ary relationship types, attributes of object classes, attributes of relationship types, cardinality, participation and uniqueness constraints, ordering, irregular and heterogeneous structures, for both data- and document-centric data.

Key Points

The ORA-SS (Object-Relationship-Attribute Data Model for Semi-structured Data) data model was designed [1] specifically to capture the constraints that are necessary for designing semi-structured databases, for normalization of schemas, and for defining views.

Figure 1 models the scenario where there is a department, with a name and many courses. A course has a unique code, a title, and many students, and a student has a unique student number, name, address, and many hobbies. For each course that a student takes, they have a grade. There is a tutor for each student in each course they take. A tutor has a unique staff number and a name, and each student can give feedback for the tutor they have in each course that they take.

A closer look is now taken at the notation used. Each of the rectangles represents an object class, the circles represent attributes and the labeled directed edges between object classes represent relationship types. A filled circle is an identifier, which is similar to a key in relational databases and an identifier of an object class. The "?" in the circle represents zero or one occurences of that attribute, while a "*" represents zero or more occurences. The default is one. An attribute in an ORA-SS diagram could be represented as an attribute or an element in an XML document. The label on the edge has *name, n, a:b, c:d*, where *name* is the name of the relationship type, *n* is the degree, *a:b* is the participation constraint on the parent and *c:d* is the participation constraint on the child. The participation constraint *a:b* indicates that the parent object participates in a minimum of *a* and a maximum of *b* relationships. Whereas, the participation constraint *c:d* indicates that the child object participates in a minimum of *c* and a maximum of *d* relationships. A label on the edge between an object class and an attribute, *name*, indicates that the attribute belongs to relationship type *name*.



Object Relationship Attribute Data Model for Semi-structured Data. Figure 1. An ORA-SS schema diagram.

Consider the example in Fig. 1. There are object classes *Department*, *Course*, *Student* and *Tutor*. Object class *Department* has an identifier *name*, and is the parent object class in the relationship type between *Department* and *Course*. The relationship type is a binary relationship with name *dc*, and the participation constraint *1:m* indicates that a department has a minimum of *one* course and a maximum of *m* courses, where *m* means many, i.e., any number of courses. The participation constraint *1:1* indicates that each course must belong to *one* department and can belong to a maximum of *one* department. Each course has an identifier *code*, a required attribute *title*, and is the parent object class in the relationship type, *cs*, between *course* and *student*. Object class *Student* has an identifier *stuNo*, a required attribute *stuName*, an optional attribute *address*, and zero or more *hobby*. There is a binary relationship type, *cs*, between *Course* and *Student*, where a *Course* can have zero or more *Students*, and a *Student* takes one or more *Courses*. The attribute *grade* belongs to the relationship type, *cs*, that is it represents the grade a student scored in a particular course. There is a ternary relationship type among object classes *Course*, *Student* and *Tutor*. Each course-student pair can have zero to one tutor, and each tutor belongs to one or more course-student pairs. Each tutor has an identifier *staffNo*, a required attribute *name*, and there is an attribute *feedback* for each tutor from a particular student in a particular course.

Cross-references

- ▶ Entity Relationship Model
- ▶ Hierarchical Data Model
- ▶ Object Data Models
- ▶ Semi-structured Data Model
- ▶ Semi-structured Database Design
- ▶ XML Integrity Constraints
- ▶ XML Schema

Recommended Reading

1. Ling T.W., Lee M.L., and Dobbie G. Semi-structured Database Design. Springer, Berlin Heidelberg New York, 2005.

Object Request Broker

- ▶ CORBA
- ▶ Request Broker

Object-based Storage Device

- ▶ Network Attached Secure Device

Object-Role Modeling

Terry Halpin
Neumont University, South Jordan, UT, USA

Synonyms

Fact-oriented modeling; NIAM

Definition

Object-Role Modeling (ORM), also known as *fact-oriented modeling*, is a conceptual approach to modeling and querying the information semantics of business domains in terms of the underlying facts of interest, where all facts and rules may be verbalized in language readily understood by non-technical users of those business domains. Unlike Entity-Relationship (ER) modeling and Unified Modeling Language (UML) class diagrams, ORM treats all facts as relationships (unary, binary, ternary etc.). How facts are grouped into structures (e.g., attribute-based entity types, classes, relation schemes, XML schemas) is considered a design level, implementation issue that is irrelevant to the capturing of essential business semantics.

Avoiding attributes in the base model enhances semantic stability, populatability, and natural verbalization, facilitating communication with all stakeholders. For information modeling, fact-oriented graphical notations are typically far more expressive than those provided by other notations. Fact-oriented textual languages are based on formal subsets of native languages, so are easier to understand by business people than technical languages like UML's Object Constraint Language (OCL). Fact-oriented modeling includes procedures for mapping to attribute-based structures, so may also be used to front-end other approaches.

The fact-oriented modeling approach comprises a family of closely related "dialects", known variously as Object-Role Modeling (ORM), Natural Language Information Analysis Method (NIAM), and Fully-Communication Oriented Information Modeling (FCO-IM). While not adopting the ORM graphical notation, the Object-oriented Systems Model (OSM) [4] and the Semantics of Business Vocabulary and business Rules (SBVR) [13] initiative within the Object Management Group (OMG) are close relatives, with their attribute-free philosophy.

Historical Background

In 1973, Falkenberg generalized work by Abrial and Senko on binary relationships to *n*-ary relationships, and excluded attributes at the conceptual level to avoid "fuzzy" distinctions and to simplify schema evolution. Later, Falkenberg proposed the fundamental ORM framework, which he called the "object-role model" [5]. This framework allowed *n*-ary and nested relationships, but depicted roles with arrowed lines. Nijssen adapted this framework by introducing a circle-box notation for objects and roles, and adding a linguistic orientation and design procedure to provide a modeling method called ENALIM (Evolving NAtural Language Information Model) [12]. Nijssen's team of researchers at Control Data in Belgium developed the method further, including van Assche who classified object types into lexical object types (LOTs) and non-lexical object types (NOLOTs). Today, LOTs are commonly called "entity types" and NOLOTs are called "value types". Meersman added subtyping to the approach, and made major contributions to the RIDL query language [11] with Falkenberg and Nijssen. The method was renamed "aN Information Analysis Method" (NIAM). Later, the acronym "NIAM" was given different expansions, and is now known as "Natural language Information Analysis Method".

In the 1980s, Nijssen and Falkenberg worked on the design procedure and moved to the University of Queensland, where the method was further enhanced by Halpin, who provided the first full formalization, including schema equivalence proofs, and made several refinements and extensions. In 1989, Halpin and Nijssen co-authored a book on the approach, followed a year later by Wintraecken's book [16]. Today several books, including major works by Halpin [10], and Bakema et al. [1] expound on the approach.

Many researchers contributed to the fact-oriented approach over the years, and there is no space here to list them all. Today various versions exist, but all adhere to the fundamental object-role framework. Habrias developed an object-oriented version called MOON (Normalized Object-Oriented Method). The Predicator Set Model (PSM), developed mainly by ter Hofstede et al. [7], includes complex object constructors. De Troyer and Meersman developed a version with constructors called Natural Object-Relationship Model (NORM). Halpin developed an extended version simply called ORM, and with Bloesch and others developed an associated query language called ConQuer [2]. Bakema et al. [1] recast all entity types as nested relationships, to produce Fully Communication Oriented NIAM, which they later modified to Fully Communication Oriented Information Modeling (FCO-IM).

More recently, Meersman and others adapted ORM for ontology modeling, using a framework called DOGMA (Developing Ontology-Grounded Methodology and Applications) (http://www.starlab.vub.ac.be/website/). Nijssen and others extended NIAM to a version called NIAM2007. Halpin and others developed a second generation ORM (ORM 2), whose graphical notation is used in this article.

Foundations

ORM includes graphical and textual *languages* for modeling and querying information at the conceptual level, as well as *procedures* for designing conceptual models, transforming between different conceptual representations, forward engineering ORM schemas to implementation schemas (e.g., relational database schemas, object-oriented schemas, XML schemas, and external schemas) and reverse engineering implementation schemas to ORM schemas.

Attributes are not used as a base construct. Instead, all fact structures are expressed as *fact types* (relationship types). These may be unary (e.g., Person smokes), binary (e.g., Person was born on Date), ternary (e.g., Person visited Country in Year), and so on. This attribute-free nature has several advantages: *semantic stability* (minimize the impact of change caused by the need to record something about an attribute); *natural verbalization* (all facts and rules may be easily verbalized in sentences understandable to the domain expert); *populatability* (sample fact populations may be conveniently provided in fact tables); *null avoidance* (no nulls occur in populations of base fact types, which must be elementary or existential). Although attribute-free diagrams typically consume more space, this apparent disadvantage is easily overcome by using an ORM tool to automatically create attribute-based structures (e.g., ER, UML class, or relational schemas) as views of an ORM schema.

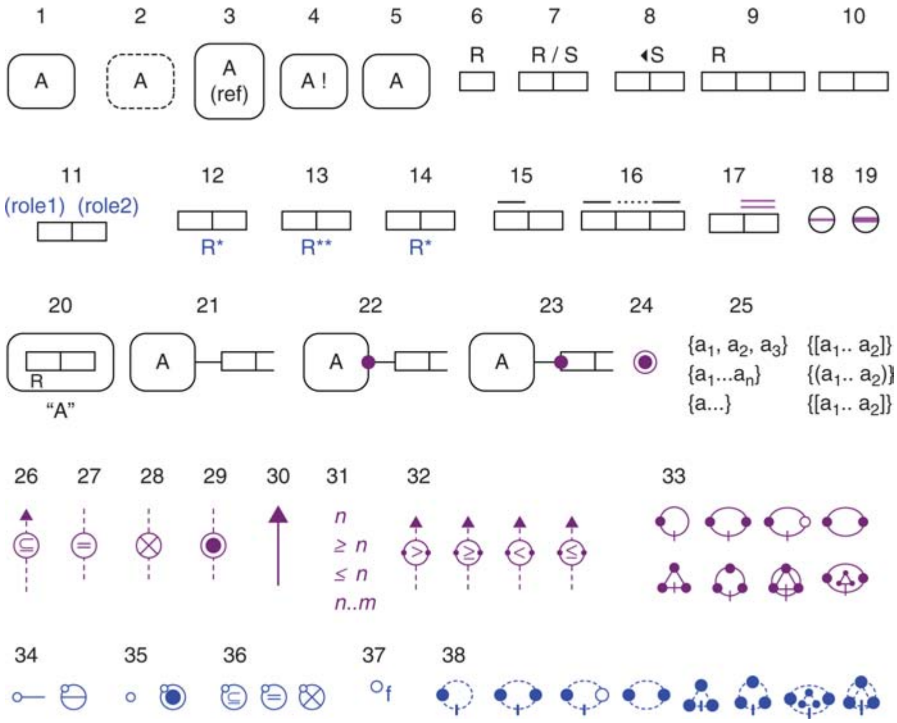
ORM's graphical language is far more expressive for data modeling purposes than that of UML or industrial versions of ER, as illustrated later. The *rich graphical notation* makes it easier to detect and express constraints, and to visually transform schemas into equivalent alternatives.

ORM includes *effective modeling procedures* for constructing and validating models. In step 1a of the Conceptual Schema Design Procedure (CSDP), the domain expert informally verbalizes facts of interest. In step 1b, the modeler formally rephrases the facts in natural yet unambiguous language, using standard reference patterns to ensure that entities are well identified. Verbalized fact instances are abstracted to fact types, which are then populated with sample instances. The constraints on the fact types are verbalized formally, a process that may be automated [8], and these verbalizations are checked with the domain expert, using positive populations to illustrate satisfaction of the constraints as well as counterexamples to illustrate what it means to violate a constraint. This approach to model *validation by verbalization and population* has proved extremely effective in industrial practice, with correct models typically obtained from the outset rather than going through unreliable iterative procedures.

Figure 1 lists the main graphical symbols in the ORM 2 notation [8], numbered for easy reference. An *entity type* (e.g., Person) is depicted as a named, soft rectangle (symbol 1), or alternatively an ellipse or hard rectangle. *Value type* (e.g., Person Name) shapes have dashed lines (symbol 2). Each entity type has a *reference scheme*, indicating how each instance may be mapped via predicates to a combination of one or more values. Injective (1:1 into) reference schemes mapping entities (e.g., countries) to single values (e.g., country codes) may be abbreviated as in symbol 3 by displaying the *reference mode* in parentheses, e.g., Country (.code). The reference mode indicates how values relate to the entities. Values are constants with a known denotation, so require no reference scheme.

Relationships used for *preferred reference* are called *existential facts* (e.g., there exists a country that has the country code ‘US’). The other relationships are *elementary facts* (e.g., The country with country code ‘US’ has a population of 301,000,000). The exclamation mark in symbol 4 declares that an object type is *independent* (instances may exist without participating in any elementary facts). Object types displayed in multiple places are shadowed (symbol 5).

A fact type results from applying a logical *predicate* to a sequence of one or more object types. Each predicate comprises a named sequence of one or more *roles* (parts played in the relationship). A predicate is



Object-Role Modeling. Figure 1. Main ORM graphic symbols.

sentence with object holes, one for each role, with each role depicted as a box and played by exactly one object type. Symbol 6 shows a unary predicate (e.g., ... smokes), symbols 7 and 8 depict binary predicates (e.g., ... loves ...), and symbol 9 shows a ternary predicate. Predicates of higher *arity* (number of roles) are allowed. Each predicate has at least one *predicate reading*. ORM uses *mixfix* predicates, so objects may be placed at any position in the predicate (e.g., the fact type Person introduced Person to Person involves the predicate "... introduced ... to ..."). Mixfix predicates allow natural verbalization of *n*-ary relationships, as well as binary relationships where the verb is not in the infix position (e.g., in Japanese, verbs come at the end). By default, forward readings traverse the predicate from left to right (if displayed horizontally) or top to bottom (if displayed vertically). Other reading directions may be indicated by an arrow-tip (symbol 8). For binary predicates, forward and inverse readings may be separated by a slash (symbol 7). Duplicate predicate shapes are shadowed (symbol 10).

Roles may be given *role names*, displayed in square brackets (symbol 11). An asterisk indicates that the fact type is *derived* from one or more other fact types (symbol 12). If the fact type is derived and stored, a double asterisk is used (symbol 13). Fact types that are semi-derived are marked "+" (symbol 14). *Internal uniqueness constraints*, depicted as bars over one or more roles in a predicate, declare that instances for that role (combination) in the fact type population must be unique (e.g., symbols 15, 16). For example, a uniqueness constraint on the first role of Person was born in Country verbalizes as: **Each** person was born in **at most one** Country. If the constrained roles are not contiguous, a dotted line separates the constrained roles (symbol 16). A predicate may have many uniqueness constraints, at most one of which may be declared *preferred* by a double-bar (symbol 17). An *external uniqueness constraint* shown as a circled uniqueness bar (symbol 18) may be applied to two or more roles from different predicates by connecting to them with dotted lines. This indicates that instances of the role combination in the join of those predicates are unique. For example, if a state is identified by combining its state code and country, an external uniqueness constraint is added to the roles played by Statecode and Country in: State has Statecode; State is in Country. Preferred external uniqueness constraints are depicted by a circled double-bar (symbol 19).

To talk about a relationship, one may *objectify* it (i.e., make an object out of it) so that it can play roles. Graphically, the objectified predicate (a.k.a. *nested* predicate) is enclosed in a soft rectangle, with its name in quotes (symbol 20). Roles are connected to their players by a line segment (symbol 21). A *mandatory role constraint* declares that every instance in the population of the role's object type must play that role. This is shown as a large dot placed at the object type end (symbol 22) or the role end (symbol 23). An *inclusive-or* (*disjunctive mandatory)* constraint applied to two or more roles indicates that all instances of the object type population must play at least one of those roles. This is shown by connecting the roles by dotted lines to a circled dot (symbol 24).

To restrict the population of an object type or role, the relevant values may be listed in braces (symbol 25). An ordered range may be declared separating end values by "..". For continuous ranges, a square/ round bracket indicates an end value is included/excluded. For example, "(0..10)" denotes the positive real numbers up to 10. These constraints are called *value constraints*.

Symbols 26–28 denote *set comparison constraints*, which apply only between compatible role sequences. A dotted arrow with a circled subset symbol depicts a *subset constraint*, restricting the population of the first sequence to be a subset of the second (symbol 26). A dotted line with a circled "=" symbol depicts an *equality constraint*, indicating the populations must be equal (symbol 27). A circled "X" (symbol 28) depicts an *exclusion constraint*, indicating the populations are mutually exclusive. Exclusion and equality constraints may be applied between two or more sequences. Combining an inclusive-or and exclusion constraint yields an *exclusive-or constraint* (symbol 29).

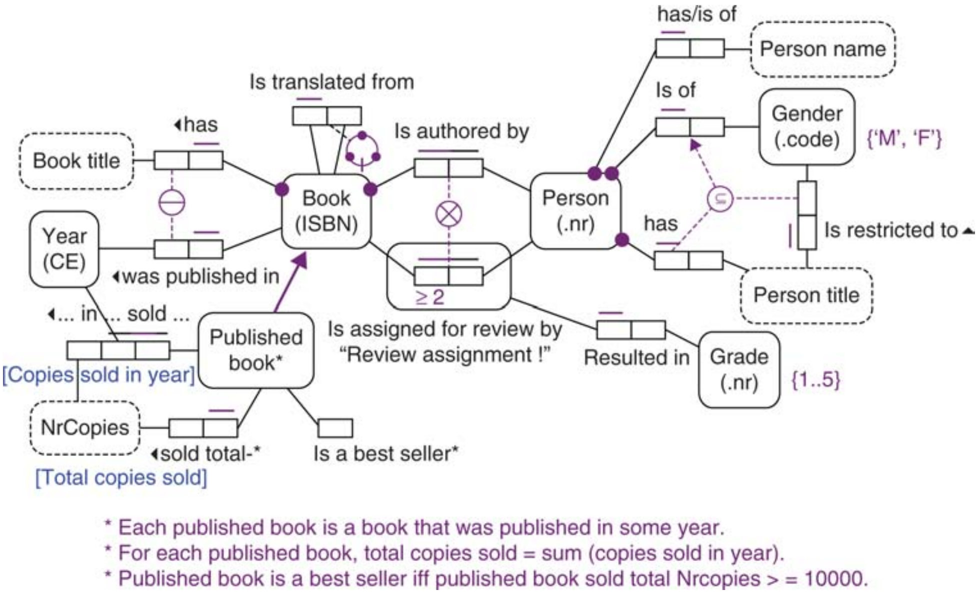
A solid arrow (symbol 30) from one object type to another indicates that the first is a (proper) *subtype* of the other (e.g., Woman is a subtype of Person). Mandatory (circled dot) and exclusion (circled "X") constraints may be displayed between subtypes, but are implied by other constraints if the subtypes have formal definitions. Symbol 31 shows four kinds of *frequency constraint*. Applied to a role sequence, these indicate that instances that play those roles must do so *exactly n* times, *at least n* times, *at most n* times, or *at least n and at most m* times. Symbol 32 shows four varieties of *value-comparison constraint*. The arrow shows the direction in which to apply the circled operator between two instances of the same type (e.g., **For each** Employee, hiredate > birthdate).

Symbol 33 shows the main kinds of *ring constraint* that may apply to a pair of compatible roles. Read left to right and top row first, these indicate that the binary relation formed by the role population must respectively be irreflexive, asymmetric, antisymmetric, reflexive, intransitive, acyclic, intransitive and acyclic, or intransitive and asymmetric.

The previous constraints are *alethic* (necessary, so can't be violated) and are colored violet. ORM 2 also supports *deontic* rules (obligatory, but can be violated). These are colored blue, and either add an "o" for obligatory, or soften lines to dashed lines. Displayed here are the deontic symbols for uniqueness (symbol 34), mandatory (symbol 35), set-comparison (symbol 36), frequency (symbol 37) and ring (symbol 38) constraints.

Figure 2 shows a sample ORM schema for a book publishing domain. A detailed discussion using the CSDP to develop this schema may be found elsewhere [9]. Each book is identified by an International Standard Book Number (ISBN), each person is identified by a person number, each grade is identified by a grade number in the range 1 through 5, each gender is identified by a code ('M' for male and 'F' for Female), and each year is identified by its common era (CE) number. Published Book is a derived subtype determined by the subtype definition shown at the bottom of the figure. Review Assignment objectifies the relationship Book is assigned for review by Person, and is independent since an instance of it may exist without playing any other role (one can known about a review assignment before knowing what grade will result from that assignment).

The internal uniqueness constraints (depicted as bars) and mandatory role constraints (solid dots) verbalize as follows: **Each** Book is translated from **at most one** Book; **Each** Book has **exactly one** Book Title; **Each** Book was published in **at most one** Year; **For each** Published Book **and** Year, **that** Published Book in **that** Year sold **at most one** NrCopies; **Each** Published Book sold **at most one** total NrCopies; **It is possible that the same** Book is authored by **more than one** Person **and that more than one** Book is authored by **the same** Person; **Each** Book is authored by **some** Person; **It is possible that the same** Book is assigned for review by **more than one** Person **and that more than one** Book is assigned for review by **the same** Person; **Each** Review Assignment resulted in **at most one** Grade; **Each** Person has **exactly one** Person Name; **Each** Person has **at most one** Gender; **Each** Person has **at most one** Person Title; **Each** Person Title is restricted to **at most one** Gender.



Object-Role Modeling. Figure 2. An ORM schema for a book publishing domain.

The external uniqueness constraint (circled bar) indicates that the combination of BookTitle and Year applies to at most one Book. The acyclic ring constraint (circle with three dots and a bar) on the book translation predicate indicates that no book can be a translation of itself or any of its ancestor translation sources. The exclusion constraint (circled cross) indicates that no book can be assigned for review by one of its authors. The frequency constraint (≥ 2) indicates that each book that is assigned for review is assigned for review by at least two persons. The subset constraint (circled subset symbol) means that if a person has a title that is restricted to some gender, then the person must be of that gender. The first argument of this subset constraint is a person-gender role pair projected from a join path that performs a conceptual join on PersonTitle. The last two lines at the bottom of the schema declare two derivation rules, one specified in attribute-style using role names and the other in relational style using predicate readings.

Key Applications

ORM has been used productively in industry for over 30 years, in all kinds of business domains. *Commercial tools* supporting the fact-oriented approach include Microsoft's Visio for Enterprise Architects, and the FCO-IM tool CaseTalk (www.casetalk.com). CogNIAM, a tool supporting NIAM2007 is under development at PNA Active Media (http://cogniam.com/). *Free ORM tools* include VisioModeler and Infagon (www.mattic.com). Dogma Modeler (www.starlab.vub.ac.be) and T-Lex [15] are academic ORM-based tools for specifying ontologies. NORMA (http://sourceforge.net/projects/orm), an open-source plug-in to Microsoft® Visual Studio, is under development to provide deep support for ORM 2 [3].

Future Directions

Research in many countries is actively extending ORM in many areas (e.g., dynamic rules, ontology extensions, language extensions, process modeling). A detailed overview of this research may be found in [9]. General information about ORM, and links to other relevant sites, may be found at www.ORMFoundation.org and www.orm.net.

Cross-references

- ▶ Conceptual Schema Design
- ▶ Data Model
- ▶ Entity Relationship Model
- ▶ UML

Recommended Reading

1. Bakema G., Zwart J., and van der Lek H. Fully Communication Oriented Information Modelling. Ten Hagen Stam, The Netherlands, 2000.
2. Bloesch A. and Halpin T. Conceptual queries using ConQuer-II. In Proc. 16th Int. Conf. on Conceptual Modeling, 1997, pp. 113–126.
3. Curland M. and Halpin T. Model Driven Development with NORMA. In Proc. 40th Annual Hawaii Int. Conf. on System Sciences, 2007.
4. Embley D., Kurtz B., and Woodfield S. Object-Oriented Systems Analysis: A Model-Driven Approach. Prentice Hall, Englewood Cliffs, 1992.
5. Falkenberg E. Concepts for modeling information. In Proc. IFIP Working Conference on Modelling in Data Base Management Systems, 1976, pp. 95–109.
6. Halpin T. Comparing metamodels for ER, ORM and UML data models. In Advanced Topics in Database Research, vol. 3, K. Siau (ed.). Idea Publishing Group, Hershey, 2004, pp. 23–44.
7. Halpin T. Fact-oriented modeling: past, present and future. In Conceptual Modelling in Information Systems Engineering, J. Krogstie A. Opdahl S. Brinkkemper (eds.). Springer, Berlin Heidelberg New York, 2007, pp. 19–38.
8. Halpin T. and Curland M. Automated verbalization for ORM 2. In On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, LNCS vol. 4278, 2006, pp. 1181–1190.
9. Halpin T., Evans K., Hallock P., and MacLean W. Database Modeling with Microsoft® Visio for Enterprise Architects. Morgan Kaufmann, San Francisco, CA, 2003.
10. Halpin T. and Morgan T. Information Modeling and Relational Databases, 2nd Edition. Morgan Kaufmann, San Francisco, CA, 2008.
11. Meersman R. (1982) The RIDL conceptual language, Research report. International Centre for Information Analysis Services, Control Data Belgium, Brussels, 1982.
12. Nijssen G.M. Current issues in conceptual schema concepts. In Proc. IFIP Working Conference on Modelling in Data Base Management Systems, 1977, pp. 31–66.
13. OMG 2007, Semantics of Business Vocabulary and Business Rules (SBVR). URL: http://www.omg.org/cgi-bin/doc?dtc/2006-08-05.
14. ter Hofstede A.H.M., Proper H.A., and Weide th.P. van der. Formal definition of a conceptual language for the description and manipulation of information models. Inf. Syst., 18(7):489–523, 1993.
15. Trog D., Vereecken J., Christiaens S., De Leenheer P., and Meersman R. T-Lex: a role-based ontology engineering tool. In On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, LNCS vol. 4278, 2006, pp. 1191–1200.
16. Wintraecken J. (1990) The NIAM Information Analysis Method: Theory and Practice, Kluwer, Deventer, The Netherlands, 1990.

OCL

▶ Object Constraint Language

ODB (Object Database)

▶ Object Data Models

ODBC

▶ Open Database Connectivity

Office Automation

▶ Enterprise Content Management

Oid

▶ Object Identity

OKAPI Retrieval Function

▶ BM25

OLAP

▶ On-Line Analytical Processing

On-Disk Security

▶ Storage Security

One-Copy-Serializability

Bettina Kemme
McGill University, Montreal, QC, Canada

Synonyms

Transactional consistency in a replicated database

Definition

While transactions typically specify their read and write operations on logical data items, a replicated database has to execute them over the physical data copies. When transactions run concurrently in the system, their executions may interfere. The replicated database system has to isolate these transactions. The strongest, and most well-known correctness criterion for replicated databases is one-copy-serializability. A concurrent execution of transactions in a replicated database is one-copy-serializable if it is equivalent to a serial execution of these transactions over a single logical copy of the database.

Key Points

A transaction is a sequence of read and write operations on the data items of the database. A read operation of transaction T_i on data item x is denoted as $r_i(x)$, a write operation on x as $w_i(x)$. A transaction T_i either ends with a commit c_i (all operations succeed) or with an abort a_i (whereby all effects on the data are undone before the termination).

A replicated database consists of a set of database servers A, B, ... and each logical data item x of the database has a set of physical copies x^A, x^B, ... where the index refers to the database server on which the copy resides. *Replica Control* translates each operation $o_i(x), o_i \in \{r, w\}$ of a transaction T_i on data item x into operations $o_i(x^A), o_i(x^B)$ on physical data copies. Given a set of transactions $\mathcal{T}$, a replicated history RH describes the execution of these transactions in the replicated database. For simplicity the following discussion only considers histories where all transactions commit. A database server A executes the subset of operations of the transactions in $\mathcal{T}$ performed on copies residing on A. The local history RH^A describes the order in which these operations occur. For simplicity a local history is assumed to be a total order. RH is the union of all local histories with some additional ordering. In particular, if a transaction T_i executes $o_i(x)$ before $o_i(y)$, and RH^A contains $o_i(x^A)$ and RH^B contains $o_i(y^B)$, then $o_i(x^A) <_{RH} o_i(y^B)$.

As an example, given $T_1 = w_1(y)w_1(x)$ and $T_2 = r_2(y)$ $w_2(x)$, and database servers A and B, both having a copy of both x and y, the local histories could be:

$$RH^A : w_1(y^A)r_2(y^A)w_1(x^A)w_2(x^A)c_1c_2$$

$$RH^B : w_1(y^B)w_1(x^B)w_2(x^B)c_2c_1$$

The replicated history *RH* is the union of these two local histories plus the ordering of $r_2(y^A) <_{RH} w_2(x^B)$.

Using this notation, the following defines one-copy-serializability for the case that replica control uses ROWA (read-one-write-all-approach), i.e., where each read operation is performed on one copy while write operations are performed on all copies of the data item. Failures are ignored. In this restricted case *conflict-equivalence* can be exploited. Two operations o_i and o_j conflict, if they are from two different transactions, access the same data copy, and at least one is a write operation.

Definition *A replicated history RH over a set of transactions* $\mathcal{T}$ *in a replicated system with servers A, B,...is one-copy-serializable if it is conflict-equivalent to a serial history H over* $\mathcal{T}$ *in a single-copy system with one logical server L. This means that if* $o_i(x^A)$, $o_j(x^A) \in RH$ *and the operations conflict, then* $o_i(x^L) <_H o_j(x^L) \in H$ *if and only if* $o_i(x^A) <_{RH^A} o_j(x^A) \in RH$.

Using conflict-equivalence, one can easily determine whether *RH* is one-copy-serializable. For each local history RH^A the serialization graph $SG(RH^A)$ has each committed transaction as node, and contains an edge from T_i to T_j if $o_i(x^A) <_{RH^A} o_j(x^A)$ and the two operations conflict. The serialization graph *SG*(*RH*) is then the union of the local serialization graphs.

Theorem *A replicated history RH over a set of transactions* $\mathcal{T}$ *and database servers A, B,...following the ROWA strategy is one-copy-serializable if and only if its serialization graph SG(RH) is acyclic.*

The example history above is one-copy-serializable because its serialization graph contains only an edge from T_1 to T_2, i.e., in all local histories, and for any conflict between T_1 and T_2, T_1's operations are ordered before T_2's operation.

As soon as node failures are considered or both read and write operations only access a subset of copies, conflict-equivalence is not appropriate anymore because it might miss catching conflicts at the logical level. For that purpose, one can define one-copy-serializability based on view-equivalence which observes which data versions a read operation accesses and in which order write operations occur.

Cross-references

- ▶ Replica Control
- ▶ Replicated Database Concurrency Control
- ▶ Strong Consistency Models for Replicated Data
- ▶ Traditional Concurrency Control for Replicated Databases

Recommended Reading

1. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency control and recovery in database systems. Addison Wesley, USA, 1987.

One-Pass Algorithm

Nicole Schweikardt
Johann Wolfgang Goethe-University, Frankfurt am Main, Frankfurt, Germany

Synonyms

One-pass algorithm; Streaming algorithm; Data stream algorithm

Definition

A one-pass algorithm receives as input a list of data items x_1, x_2, x_3,.... It can read these data items only once, from left to right, i.e., in increasing order of the indices $i = 1, 2, 3$,.... Critical parameters of a one-pass algorithm are (1) the size of the memory used by the algorithm, and (2) the processing time per data item x_i. Typically, a one-pass algorithm is designed for answering one particular query against the input data. To this end, the algorithm stores and maintains a suitable data structure which, for each i, is updated when reading data item x_i.

The two parameters *processing time per data item* and *memory size* are usually measured as functions depending on the size *N* of the input (different measures of the *input size* are considered in the literature, among them, e.g., the number of data items occurring in the input, as well as the total number of bits needed for storing the entire input). The ultimate goal when designing a one-pass algorithm is to keep the processing time per data item and the memory size *sublinear*, preferably polylogarithmic, in *N*. In particular, one typically aims at algorithms whose memory size is far smaller than the size of the input.

Key Points

The design and study of one-pass algorithms has a long tradition in many areas of computer science. For example, they are used in the area of *data stream processing*, where streams of huge amounts of data have to be monitored *on-the-fly* without first storing the entire data. A deterministic finite automaton on words can be viewed as a (very simple) example of a one-pass algorithm whose memory size and processing time per data item is constant, i.e., does not depend on the input size. For most computational problems, however, the amount of memory necessary for solving the problem grows with increasing input size. *Lower bounds* on the memory size needed for solving a problem by a one-pass algorithm are usually obtained by applying methods from *communication complexity* (see, e.g., [1,2] for typical examples).

For many concrete problems it is even known that the memory needed for solving the problem by a deterministic one-pass algorithm is at least linear in the size N of the input. For some of these problems, however, *randomized* one-pass algorithms can still compute good *approximate* answers while using memory of size sublinear in N (cf. [1,2,3]). Typically, such algorithms are based on *sampling*, i.e., only a "representative" portion of the data is taken into account, and *random projections*, i.e., only a rough "sketch" of the data is stored in memory (see [3] for a comprehensive survey of according algorithmic techniques).

In the context of database systems these techniques are relevant, for example, for maintaining information needed for cost-based query optimization, e.g., estimates for the number of distinct values of an attribute, or the self-join size of a database relation. Efficient one-pass algorithms for incrementally updating these estimates can be found in [1].

In some application areas, rather than just a single pass, a small number P of sequential passes over the data may be available; the resulting algorithms are called *multi-pass algorithms* (see e.g., [2] for an analysis of the trade-off between the memory size and the number of passes necessary for solving particular problems).

Cross-references

- ▶ Approximate Query Processing
- ▶ Clustering on Streams
- ▶ Data Stream
- ▶ Data Sketch/Synopsis
- ▶ Event and Pattern Detection over Streams
- ▶ Stream Processing
- ▶ XML Stream Processing

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. J. Comput. Syst. Sci., 58:137–147, 1999.
2. Henzinger M., Raghavan P., and Rajagopalan S. Computing on data streams. In External Memory Algorithms. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 50. American Mathematical Society, Boston, MA, USA, 1999, pp. 107–118.
3. Muthukrishnan S. Data streams: algorithms and applications. Found. Trends Theor. Comput. Sci., 1(2):117–236, 2005.

One-Way Hash Functions

▶ Hash Functions

Online Advertising

▶ Web Advertising

On-Line Analytical Processing

Alberto Abelló, Oscar Romero
Polytechnic University of Catalonia, Barcelona, Spain

Synonyms

OLAP

Definition

On-line analytical processing (OLAP) describes an approach to decision support, which aims to extract knowledge from a data warehouse, or more specifically, from data marts. Its main idea is providing navigation through data to non-expert users, so that they are able to interactively generate ad hoc queries without the intervention of IT professionals. This name was introduced in contrast to on-line transactional processing (OLTP), so that it reflected the different requirements and characteristics between these classes of uses. The concept falls in the area of business intelligence.

Historical Background

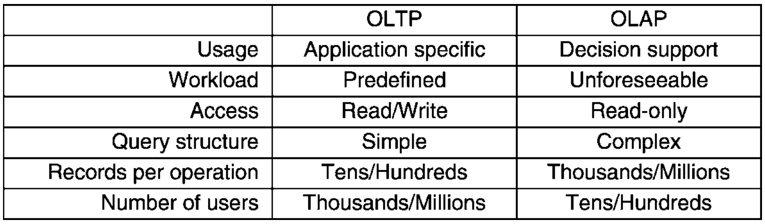
From the beginning of computerized data management, the possibility of using computers in data analysis has been evident for companies. However, early analysis tools needed the involvement of the IT department to help decision makers to query data. They were not interactive at all and demanded specific knowledge in computer science. By the mid-1980s, executive information systems appeared introducing new graphical, keyboard-free interfaces (like touch screens). However, executives were still tied to IT professionals for the definition of ad hoc queries, and prices of software and hardware requirements where prohibitive for small companies. Eventually, cheaper and easy-to-use spreadsheets became very popular among decision makers, but soon it was clear that they were not appropriate for using and sharing huge amounts of data. Thus, it was in 1993 that Codd et al. [2], coined the term OLAP. In that report, the authors defined 12 rules for a tool to be considered OLAP. These rules caused heated controversy, and they did not succeed as Codd's earlier proposal for relational database management systems (RDBMS). Nevertheless, the name OLAP became very popular and is broadly used.

Although the name OLAP comes from 1993 and the idea behind them goes back to the 1980s, there is not a formal definition for this concept, yet. As proposed by Nigel Pendse [6], OLAP tools should pass the FASMI (fast analysis of shared multidimensional information) test. Thus, they should be fast enough to allow interactive queries; they should help analysis task by providing flexibility in the usage of statistical tools and what–if studies; they should provide security (both in the sense of confidentiality and integrity) mechanisms to allow sharing data; they should provide a multidimensional view so that the data cube metaphor can be used by users; and, finally, they should also be able to manage large volumes of data (gigabytes can be considered a lower bound for volumes of data in decision support) and metadata. However, there are not measures and thresholds for all these characteristics in order to be able to establish whether one of them is fulfilled or not, and therefore it is always arguable that a given tool fulfills them. Nevertheless, it is generally agreed that in order to be considered an OLAP tool, it must offer a multidimensional view of data.

Since their first days, OLAP tools have been losing weight and lowering prices, while at the same time, offering more functionality, better user interfaces and easier administration. Thus, time has come for small companies to use OLAP. They can afford it and they are willing to use it in their decision processes. Part of OLAP industry was associated into the OLAP Council (created in January 1995), whose aim was the promotion and standardization of OLAP terminology and technology. However, some major vendors never became members of this council, so eventually it disappeared (last news date from 1999). Nowadays, there is no standardization institution specifically devoted to OLAP. Therefore, it seems difficult to have a standard data model and query language in the near future, despite the fact that it is clearly desirable.

Foundations

OLAP environments have completely different requirements, compared to OLTP. Figure 1 summarizes the main differences. Firstly, their usage is different. While OLTP systems are conceived to solve a concrete problem and are used in the daily work of companies, OLAP systems are used in decision support. Thus, in the first case, since the addressed problem can be completely specified, the workload of the system is clearly predefined. Conversely, a decision support system aims to solve new problems every day. Therefore, ad hoc queries are executed. OLTP systems read as well as write data, while OLAP systems are considered read-only, because decision makers do not directly modify data. Nevertheless, the queries in a decision support system are much more complex, since they usually include big volumes of information processed by joining several tables, grouping data and calculating functions. Queries in OLTP systems do not usually involve volumes of data of the same magnitude, neither as many tables, nor groupings or calculations. The



	OLTP	OLAP
Usage	Application specific	Decision support
Workload	Predefined	Unforeseeable
Access	Read/Write	Read-only
Query structure	Simple	Complex
Records per operation	Tens/Hundreds	Thousands/Millions
Number of users	Thousands/Millions	Tens/Hundreds

On-Line Analytical Processing. Figure 1. Comparing OLTP Versus OLAP.

number of records in OLTP operations can be estimated as tens or hundreds at most, while OLAP queries usually involve thousands or even millions of records. Finally, the number of users is also different in both kinds of systems. OLTP systems can have thousands or millions of users (like in the case of cash machines), while OLAP systems have tens or maybe hundreds of users.

The main characteristic of OLAP is multidimensionality. The data cube metaphor is used to make user interaction easier and closer to decision makers' way of thinking, who would probably find SQL or any other text-based query language hard to understand and error prone. Thus, it is much easier for them to think in terms of the multidimensional model, where a Fact is a subject of analysis and its Dimensions are the different points of view that analysts could use to study the Fact. In this way, the instances of a Fact are shown in an *n*-dimensional space usually called Cube or Hypercube.

In order to show *n*-dimensional Cubes in two-dimensional interfaces, Cross-tabs or Statistical Tables such as the one in Fig. 2 (its data is entirely fictitious) are used. While in relational tables it is found that fixed columns and different instances are shown in each row, in Cross-tabs both columns and rows are fixed and interchangeable. In this example, you see three dimensions (i.e., Product, Place, and Year) that show the different points of view to analyze the OLAP tools market.

Multidimensionality is based on this fact-dimension dichotomy. A Dimension is considered to contain a hierarchy of aggregation levels representing different granularities (or levels of detail) to study data, and an aggregation level to contain descriptive attributes. On the other hand, a Fact contains quantitative attributes that are called measures. Dimensions of analysis arrange the multidimensional space where the Fact of study is depicted. Each instance of data is identified (i.e., placed in the multidimensional space) by a point in each of its analysis dimensions. Two different instances of data cannot be spotted in the same point of the multidimensional space. Therefore, given a point in each of the analysis dimensions they only determine one, and just one, instance of factual data. Moreover, data summarization that is performed must be correct, i.e., aggregated categories must be a partition (complementary and disjoint) and the kind of measure, aggregation function, and the dimension along which data is aggregated must be compatible. For example, stock, sum and time are not compatible, since stock measures cannot be added along temporal dimensions.

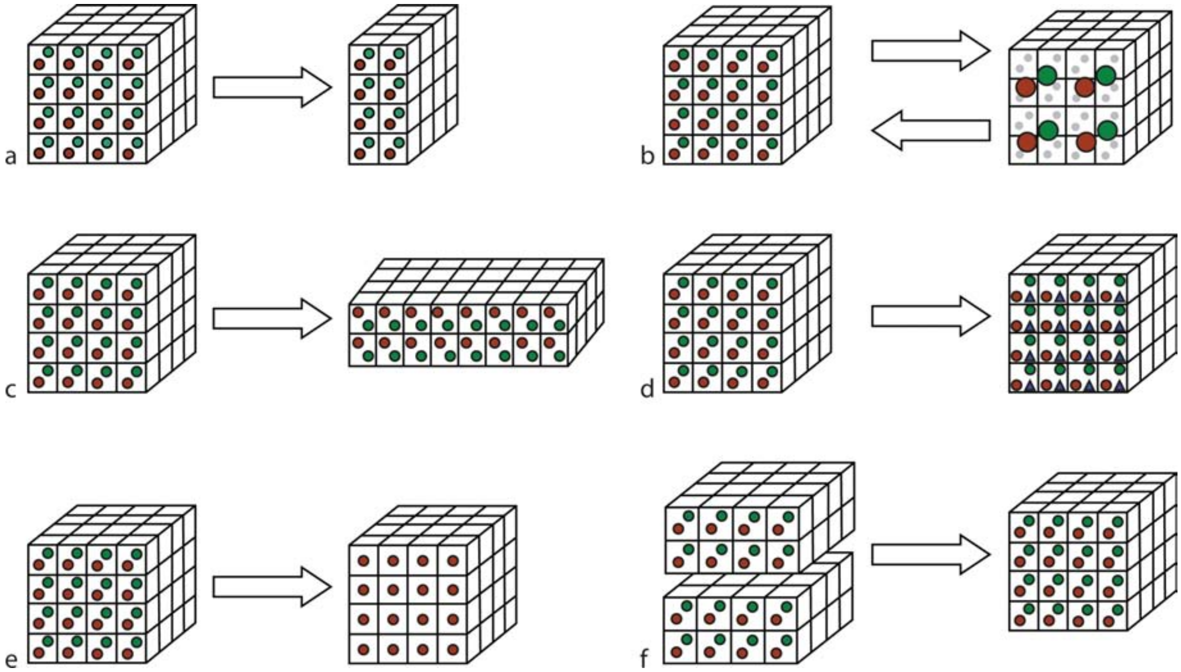
Market (billion US$)	ROLAP tools		MOLAP tools	
	Europe	USA	Europe	USA
2005	1	2	0.75	0.25
2006	1.5	2.5	1	0.5

On-Line Analytical Processing. Figure 2. Example of cross-tab or statistical table representation of a 2 × 2 × 2 data cube.

Operations

Unfortunately, there is no consensus on the set of multidimensional operations and how to name them. However, [10] provides a comparison of algebraic proposals in the academic literature, as well as a set of operations subsuming all of them. A sequence of these operations is known as an OLAP session. An OLAP session allows transformation of a starting query into a new query. Figure 3 draws the transitions generated by each one of these operations (circles and triangles represent different attributes for Fact instances):

1. *Selection or dice*. By means of a logic predicate over the dimension attributes, this operation allows users to choose the subset of points of interest out of the whole *n*-dimensional space (Fig. 3a).
2. *Roll-up*. Also called "Drill-up", it groups cells in a Cube based on an aggregation hierarchy. This operation modifies the granularity of data by means of a many-to-one relationship which relates instances of two aggregation levels in the same Dimension, corresponding to a part-whole relationship (Fig. 3b from left to right). For example, it is possible to roll-up monthly sales into yearly sales moving from "Month" to "Year" aggregation level along the temporal dimension.
3. *Drill-down*. This is the counterpart of Roll-up. Thus, it removes the effect of that operation by going down through an aggregation hierarchy, and showing more detailed data (Fig. 3b from right to left).
4. *ChangeBase*. This operation reallocates exactly the same instances of a Cube into a new *n*-dimensional space with exactly the same number of points (Fig. 3c). Actually, it allows two different kinds of changes in the space: rearranging the multidimensional space by reordering the Dimensions, interchanging rows and columns in the Cross-tab (this



On-Line Analytical Processing. Figure 3. Schema of operations on cubes.

is also known as Pivoting), or adding/removing dimensions to/from the space.

5. *Drill-across*. This operation changes the subject of analysis of the Cube, by showing measures regarding a new Fact. The *n*-dimensional space remains exactly the same, only the data placed in it change so that new measures can be analyzed (Fig. 3d). For example, if the Cube contains data about sales, this operation can be used to analyze data regarding production using the same Dimensions.
6. *Projection*. It selects a subset of measures from those available in the Cube (Fig. 3e).
7. *Set operations*. These operations allow users to operate two Cubes defined over the same *n*-dimensional space. Usually, Union (Fig. 3f), Difference and Intersection are considered.

This set of algebraic operations is minimal in the sense that none of the operations can be expressed in terms of others, nor can any operation be dropped without affecting functionality (some tools consider that the set of measures of a Fact conform to an artificial analysis dimension, as well; if so, Projection should be removed from the set of operations in order to be considered minimal, since it would be done by Selection over this artificial Dimension). Thus, other operations can be derived by sequences of these. It is the case of Slice (which reduces the dimensionality of the original Cube by fixing a point in a Dimension) by means of Selection and ChangeBase operations. It is also common that OLAP implementations use the term Slice&Dice to refer to the selection of fact instances, and some also introduce Drill-through to refer to directly accessing the data sources in order to lower the aggregation level below that in the OLAP repository or data mart.

Declarative Languages

There are some research proposals of declarative query languages for OLAP. Cabibbo and Torlone [1] propose a graphical query language, while Gyssens and - Lakshmanan [3] propose a calculus. From the industry point of view, MDX (standing for multidimensional expressions) [5] is the de facto standard. It was introduced in 1997, and in spite of the specification being owned by Microsoft, it has been widely adopted. Its syntax resembles that of SQL:

```
[WITH <MeasureDefinition>+]
SELECT <DimensionSpecification>+
FROM <CubeName>
[WHERE <SlicerClause>]
```

However, its semantics are completely different. Roughly speaking, an MDX query gets the instances of a given Cube stated in the FROM clause and places them in the space defined by the SELECT clause. Moreover, complex calculations can be defined in the WITH clause, and the dimensions not used in the SELECT clause can be sliced in the WHERE clause (if not explicitly sliced, it is assumed that dimensions that do not appear in the SELECT are sliced at the higher aggregation level: All).

```
WITH MEMBER [Measures].[pending] AS
‘[Measures].[Units Ordered]-[Mea-
sures].[Units Shipped]’ SELECT
 {[Time].[2006].children} ON COLUMNS,
 {[Warehouse].[Warehouse Name].mem-
bers} ON ROWS
FROM Inventory
WHERE ([Measures].[pending],[Trade-
mark].[Acme]);
```

In the previous MDX query, an ad hoc measure “pending” is first defined as the difference between units ordered and shipped. Then, the children of the instance representing year 2006 (i.e., the 12 months of that year) are placed on columns, and the different members of the aggregation level “Warehouse Name” on rows. Now, this matrix is filled with the data in “Inventory” cube, showing the previously defined measure “pending” and slicing “Acme” trademark.

Key Applications

Managers are usually not trained to query databases by means of SQL. Moreover, if the query is relatively complex (several joins and subqueries, grouping, and functions) and the database schema is not small (with maybe hundreds of tables), using interactive SQL could be a nightmare even for SQL experts. Thus, OLAP is used to ease the tasks of these managers in extracting knowledge from the data warehouse by means of Drag&Drop, instead of typing SQL queries by hand.

OLAP market is estimated around US$ 6 billion in 2006, which is mainly devoted to decision making. However, this paradigm can also be used in any other field with non-expert users, where schemas and queries are relatively complex. For example, its usage is under investigation in bioinformatics [8], and the semantic web [9].

Future Directions

OLAP is used to extract knowledge from the data warehouse. Data mining tools can also be used for this purpose. Until now, both research communities have been evolving separately. The former must be interactive, while the latter presents computational complexity problems. However, it seems promising to integrate both kinds of tools so that one can benefit from the other [4]. Some tools like Microsoft Analysis Services already integrate them in some way. Nevertheless, there is still much work to do in this field.

On the other hand, security is usually a flaw in data warehousing projects. Reference [7] contains a survey of OLAP security problems. In the past, OLAP tools used to have just a few users and all of them had high responsibilities in the organization, so this was not really a concern in the sense of confidentiality. Nowadays, with the increase in potential users of OLAP systems inside as well as outside the organization, security has emerged as a priority in these projects. Moreover, personal data (like those of customers) are usually analyzed in almost all companies. Thus, inference control mechanisms need to be studied in data mining as well as OLAP tools.

Other research directions in OLAP can be the improvement of user interaction and flexibility in the calculation of statistics, and the integration of what–if analysis (see What–if Analysis definitional entry).

Url to Code

Some OLAP vendors:

1. Microsoft Analysis Services: http://www.microsoft.com/sql/technologies/analysis/default.mspx
2. Hyperion Solutions: http://www.hyperion.com
3. Cognos PowerPlay: http://www.cognos.com/products/business_intelligence/analysis/index.html
4. Business Objects: http://www.businessobjects.com/products/queryanalysis/olapaccess/businessobjects.asp
5. MicroStrategy: http://www.microstrategy.com/Solutions/5Styles/olap_analysis.asp

Some open source OLAP tools:

1. Mondrian: http://mondrian.pentaho.org
2. Palo: http://www.palo.net

Cross-references

- ▶ Business Intelligence
- ▶ Cube Implementations
- ▶ Database Management System
- ▶ Data Mart
- ▶ Data Mining
- ▶ Data Warehouse
- ▶ Dimension
- ▶ Hierarchy
- ▶ Hierarchical Data Summarization
- ▶ Measure
- ▶ Multidimensional Modeling
- ▶ Star Schema
- ▶ Summarizability
- ▶ Visual On-Line Analytical Processing (OLAP)

Recommended Reading

1. Cabibbo L. and Torlone R. From a procedural to a visual query language for OLAP. In Proc. 10th Int. Conf. on Scientific and Statistical Database Management. 1998, pp. 74–83.
2. Codd E.F., Codd S.B., and Salley C.T. Providing OLAP to user-analysts: An IT mandate. Technical Report, E. F. Codd & Associates, 1993.
3. Gyssens M. and Lakshmanan L.V.S. A foundation for multi-dimensional databases. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997, pp. 106–115.
4. Han J. OLAP Mining: Integration of OLAP with Data Mining. In Proc. IFIP TC2/WG2.6 Seventh Conf. Database Semantics, 1997, pp. 3–20.
5. Microsoft. Multidimensional Expressions (MDX) Reference. Available at http://msdn2.microsoft.com/en-us/library/ms145506.aspx, 2007. SQL Server books online.
6. Pendse N. The OLAP Report – What is OLAP? Available at http://www.olapreport.com/fasmi.html, 2007. Business Application Research Center.
7. Priebe T. and Pernul G. Towards OLAP Security Design – Survey and Research Issues. In Proc. ACM Int. Workshop on Data Warehousing and OLAP, 2000, pp. 33–40.
8. Rahm E., Kirsten T., and Lange J. The GeWare data warehouse platform for the analysis of molecular-biological and clinical data. J. Integr. Bioinformat., 1(4):47, 2007.
9. Romero O. and Abelló A. Automating Multidimensional design from ontologies. In Proc. ACM Int. Workshop on Data Warehousing and OLAP, 2007, pp. 1–8.
10. Romero O. and Abelló A. On the need of a reference algebra for OLAP. In Proc. Int. Conf. on Data Warehousing and Knowledge Discovery, 2007, pp. 99–110.

Online Handwriting

▶ Electronic Ink Indexing

Online Recovery

▶ Crash Recovery

Online Recovery in Parallel Database Systems

RICARDO JIMENEZ-PERIS
Universidad Politecnica de Madrid, Madrid, Spain

Synonyms

High availability; Continuous availability; 24x7 operation

Definition

Replication (also known as clustering) is a technique to provide high availability in parallel and distributed databases. High availability aims to provide continuous service operation. High availability has two faces. On one hand, it provides fault-tolerance by introducing redundancy in the form of replication, that is, having multiple copies or replicas of the data at different sites. On the other hand, since sites holding the replicas may crash and/or fail, in order to keep a given degree of availability, failed or new replicas should be reintroduced into the system. Introducing new replicas requires transferring to them the current state in a consistent fashion (known as *recovery*). A simple solution to this problem is *offline recovery*, that is, in order to obtain a quiescent state, request processing is suspended, then the state is transferred from a working replica (termed *recoverer replica*) to the new replica (*recovering replica*) and finally, request processing is resumed. Unfortunately, offline recovery results in a loss of availability, which defeats the original goal of replication, that is to provide high availability. The alternative is *online recovery*, in which transaction processing is not stopped while the recovery is performed. The main challenge for online recovery is to attain consistency, since the state to be transferred to the recovering replica(s) is a moving target. While the recovery takes place, new transactions are processed and the state evolves during the recovery itself. Online recovery needs to be coordinated with the replica control protocol to enforce consistency.

Historical Background

Recovery is used in centralized databases to bring the database to a consistent state after a crash [1]. The consistency is attained by ensuring that the updates of committed transactions are reflected in the database and the updates of uncommitted (aborted) transactions are not reflected in it. In clustered databases, centralized recovery is used to bring a failed replica to a local consistent state, but then, since other working replicas may have processed transactions, centralized recovery is not sufficient and it has to be followed by a replica recovery [11]. During replica recovery, the failed replica (or a fresh new one) recovers the current state from the working replicas. What is meant by *current state* the state reflecting all the updates from transactions that will not be processed by the recovering replica, and not reflecting any of the updates from transactions that will be processed by the recovering replica after recovery. A failed replica only needs to recover the missed updates, while a new replica needs to recover the full database.

The seminal paper on online recovery for clustered databases is [8]. In this paper, a suite of protocols for online recovery is proposed. One of the protocols lies in a locking-based online recovery. The full database is locked atomically using recovery locks, a special kind of read lock. This guarantees a quiescent state of the database. Then, the recovery locks are released as data is transferred to the recovering replica. The atomic setting of the recovery locks acts as a synchronization point. All update requests processed before the recovery lock setting should be reflected in the transferred state. Requests submitted after the recovery locks are set should be processed by the recovering replica after recovery finishes. This protocol lies inbetween offline and online recovery. In the beginning, all the data are locked, and therefore the database is unavailable. This situation improves as recovery progresses, since recovery locks are released and the corresponding data become available. Another online recovery protocol proposed in [8] is a multi-round recovery. In this protocol, the state to be transferred (missed updates) to the recovering replica is sent in rounds. The first round would contain all the updates missed until the start of recovery. In the second round, the updates performed during the first round are sent, and so on. When the number of updates performed during the last round is small enough, a last round is run. During the last round, the recovering replica has to store all client requests to process them after finishing recovery. This recovery protocol if fully online and also significantly reduces the number of transactions to be stored during recovery, since it is only during the last round that the recovering replica has to store incoming transactions (typically only the resulting updates from them).

Another piece of work on online recovery was presented in [5]. This paper describes a log-based online recovery protocol in which a failed replica receives the prefix of the log corresponding to the update transactions it has missed. Since the log grows as the recovery progresses, the protocol has a special handshake protocol to finish recovery. The recoverer traverses the log from the first transaction the failed replica missed until it reaches the end of the log. At this point, the end recovery handshake protocol is started to determine which will be the last transaction to be sent as part of the recovery. The recovering replica starts storing requests that follow this transaction to process them after recovery finishes.

Online recovery has also been used for replicated data warehouses across the Internet [9]. In this work, each replica is located at an autonomous organization and exhibits an interface to execute queries. The online recovery protocol exploits the underlying architecture and performs recovery by issuing queries from the recovering replica to the working replicas. It also takes advantage of a facility for historical queries that enables executing a read only query providing a timestamp T. This historical query will return the same results as it happened at time T. The recovery protocol has 3 phases. In the first phase, the recovering site determines the latest time T for which it has all the committed updates. In the second phase, the recovering site runs historical queries at working replicas (that act as recoverer replicas) with a timestamp between the recovery point and a time closer to the present to catch up missed updates. Historical queries do not set read locks, and therefore do not block updates at the recoverer sites. In the final phase, a regular query (non-historical) is run to get the latest updates. In this case, read locks are set to guarantee the consistency.

Online recovery has also been studied in other contexts, such as diverse data replication and Byzantine data replication. In diverse data replication [4], each replica runs a database from a different vendor. This approach enables tolerating software failures since it has been observed that bugs in one product typically do not appear in a database from a different vendor [4]. In diverse data replication, recovery is slightly more complex, since it requires using a common abstract representation of the data. This might need some tuple translation during recovery in order to align fields with slightly different types.

Byzantine data replication [17] tolerates intrusions and provides continuous correct service despite them. An intrusion happens at a site when it is attacked. This site may behave arbitrarily to disrupt service provision. Intrusions are modeled as Byzantine (also known as arbitrary) failures. Byzantine replication typically resorts to diverse data replication to avoid common vulnerabilities. Typically tolerating f Byzantine failures requires $3 \cdot f+1$ replicas. Byzantine data replication has a more involved recovery, since it should also mask Byzantine recoverers. This means that a sufficient number of recoverers is needed in order to mask Byzantine failures during recovery [2]. An additional issue in Byzantine fault tolerance is that once one replica has been successfully attacked, another one can be attacked, and so on. In order to reduce the window of vulnerability, proactive recovery has been proposed [2]. This approach recovers replicas proactively without waiting until they are crashed or attacked. During recovery, the recovering replica boots from read-only media and recovers the state from working replicas using a Byzantine recovery protocol. During recovery, the state is transferred from multiple working replicas to be able to tolerate Byzantine (attacked) recoverers. Replicas are recovered in a round-robin fashion forced by a reboot provoked by a hardware watchdog. In this way, even if a replica has been attacked silently (without the system and administrator noticing it), it will become operative again, thus, reducing the window of vulnerability of the system.

Foundations

High availability consists of two inseparable aspects. On one hand, it requires the ability to tolerate failures. This is typically achieved by introducing redundancy in the form of replication. However, in order to keep a given degree of availability, the ability to recover failed (or new) replicas is also necessary. Recovering failed replicas requires obtaining a quiescent state from a working replica (or *recoverer replica*) and transferring it to the new replica (or *recovering replica*). This quiescent state can be easily obtained by stopping transaction processing. This results in *offline recovery.* Although offline recovery guarantees the consistency of recovery, its major drawback is that it results in a loss of availability during the recovery process that defeats the original goal of replication, which is to provide high availability. The alternative is *online recovery*, that is, to transfer the state to a recovering replica without stopping transaction processing.

Replication can be used to provide scalability in addition to providing availability. In this case, availability results insufficient as a metrics to express the goodness of a recovery protocol [5]. Performability becomes a more appropriate metrics in this context. *Performability* is defined as the cumulative performance of a highly available system over a period of time in the presence of failures and recoveries. To understand why it is important, an extreme situation will be illustrated. A replicated system has a throughput of 1000 transactions per second (tps). During online recovery the system remains available, but its throughput decreases to 1 tps. Although this system is available, it is clearly worse than a system that would deliver 990 tps during the online recovery (assuming that online recovery takes the same amount of time in both systems). However, when comparing the performability of both systems during online recovery, the former system would offer a very poor performability, while the second would offer a very high one close to the one in which the system is not recovering any replica.

Online recovery protocols typically consist of five phases:

1. *Local Recovery* brings the local state to a consistent state by means of centralized recovery.
2. *Find Last Committed Update Transaction* determines the last update transaction reflected in the local state.
3. *Global Recovery Start* initiates online recovery taking care of obtaining a quiescent state from one or more working replicas to be transferred to the recovering replica.
4. *Global Recovery* transfers a quiescent state from a working replica to the recovering replica.
5. *Global Recovery End* is the handshake protocol to determine the end of recovery.

The first and second phases depend on whether the recovering replica is a failed replica or a fresh new replica. For a failed replica, the first phase is typically performed automatically by the underlying database system upon recovery after a crash. For a failed replica, the second phase implies traversing the log or any other recovery information repository to find out which was the last committed update transaction. In most protocols, this information does not need to be precise. It can be enough to obtain the identifier of a committed transaction (e.g., a timestamp, log sequence number or transaction identifier, TID) close to the failure instant such that all previously committed transactions are also reflected in the local state. If some updates of latter transactions are reflected in the state, they will be rewritten during recovery. For a fresh new replica, the first phase is empty and the second phase first involves obtaining a checkpoint of the database and then performing the same processing as a failed replica. The checkpoint of the database also needs to be obtained in an online fashion using techniques such as point-in-time recovery [16] in order to keep the system available.

In the third phase, there is some communication between the recovering replica and the working replicas. This interaction has several purposes. First, the working replicas become aware of the new replica wanted to join the system. Second, the recovering replica informs the working replicas about the last known update. Third, a recoverer is elected to transfer the state to the recovering replica.

The fourth phase transfers the state from the recoverer to the recovering replica. The recovery process is synchronized with replica control to guarantee that a quiescent state is transferred to the recovering replica.

The fifth phase aims at finishing the recovery, which requires splitting the sequence of transactions into two disjoint sets (the ones whose state is reflected in the state transfer to the recovering replica, and the ones that should be processed by the recovering replica after finishing the recovery).

Online recovery depends on the specific features of the replica control protocol being used, such as eager vs. lazy, primary-backup vs. update-everywhere, kernel-based vs. middleware-based replication, etc. In eager replication, the coordination between replica control and online recovery is very tight to guarantee consistency [6,7,13,14,10]. In lazy replication, the coordination can be looser. For instance, in a freshness-based approach [12,3], as far as the freshness requirement is satisfied the coordination can be more relaxed. In a primary-backup approach, the recovery of backups is simpler since they do not execute updates on their own, they just apply the updates coming from the primary [15]. In update-everywhere approaches [6,7,13,14,10] the recovery needs to be interleaved carefully with replica control to guarantee consistency [8,5]. In kernel-based approaches, it is possible to use recovery protocols that use mechanisms within the kernel (such as locking) [8]. However, in middleware-based approaches, the recovery can only use those mechanisms available at the database interface [5].

In this entry, two approaches will be examined in detail: an online recovery for kernel-based replication based on locking [8], and an online recovery for middleware-based replication based on logging [5].

First, a locking-based online recovery approach is studied. This approach is based on the most basic protocol from [8]. This basic version of the protocol transfers the full database. The recovery is coordinated with replica control to guarantee consistency. This coordination is materialized through locking. In what follows, the first generic phases of online recovery for this particular protocol are described. The first phase, local recovery, is orthogonal to online recovery it can just be ignored. The second phase consists in determining the last update known by the recovering replica. Since in this approach the full database is sent, this phase does not exist.

The third phase is recovery start. The recovering replica notifies to the working replicas that is willing to join the system and recover. In this protocol, in order to guarantee the quiescence of the transferred state, the recovery is started by initiating a transaction that sets atomically special read locks or recovery locks over all the tuples in the database. The quiescent state to be transferred corresponds to the state just after the atomic setting of the recovery locks. The recovery then takes place gradually. As soon as a recovery lock is granted, the tuple is read and sent to the recovering replica. Then, the lock is released. It should be noted that recovery locks are read locks and therefore do not delay read operations, only update ones. During the recovery, the recovering replica should store the incoming transactions (only those involving updates) to process them after recovery, since the associated updates are not incorporated in the state being transferred to it.

The end-of-recovery handshake is simple in this protocol. It is initiated after the sending of the last tuple, and the recovery message piggybacks is marked to indicate this fact. After receiving this message, the recovering replica starts to process all the stored transactions during the recovery. Depending on the replica control in place the recovering replica will execute the update transactions, or will receive the resulting updates from the other replicas (which is usually the case). In the latter case, incoming transactions do not need to be stored-just the update propagation messages from the other replicas.

There are many optimizations that can be performed over this basic protocol as described in [8]. Batching recovery messages limit the amount of data transferred during recovery by recording the updates performed during recovery, shortening the amount of updates to be stored during online recovery by using multiple phases, etc.

The second protocol that will be described is a log-based protocol [5]. This protocol is combined with a pessimistic replica control implemented as a middleware layer. A pessimistic replica control freely executes non-conflictive transactions in the replicated system, while conflictive ones are executed in the same relative order at all replicas. The replica control protocol over which online recovery is built is Nodo, described in [13]. Nodo is based on the notion of conflict classes. A transaction might access one or more conflict classes. Each conflict class has a master replica. Each combination of conflict classes also has a master replica, one of the master replicas of the individual conflict classes. Each conflict class has an associated transaction queue. Update transactions are sent to all replicas in the same order. Read-only transactions are only sent to one of the replicas. Transactions are queued in the conflict class queues relevant to them (the ones that they read or write). When a transaction is at the front of all the queues it has been enqueued, the master of the conflict class combination associated to the transaction executes it locally. If the transaction is read-only, then the results are returned directly to the client and removed from all the queues. If the transaction contains updates, the master extracts them from the database and sends them to all other replicas. Nodo keeps a log for each individual conflict class that is exploited by the online recovery protocol.

The online recovery protocol for Nodo guarantees consistent recovery by careful coordination with the replica control protocol of Nodo. Individual conflict classes can be recovered independently, that is, each individual conflict class could use a different recoverer replica. The recovery of an individual conflict class is detailed in what follows. The first phase, local recovery, occurs when the replica recovers. The second phase, determining the last update reflected in the replica state, is done by looking at the local log. The recovering replica traverses the log to determine the identifier of the last transaction processed for the conflict class being recovered. This identifier is used in the fourth phase to initiate the recovery of the conflict class. One of the working replicas is elected as recoverer replica for this conflict class. In the fourth phase, the recoverer replica traverses the log of the conflict class being recovered. The recoverer replica continues processing updates involving this conflict class. This means that the log grows as is being traversed. The recovering replica just applies all the updates corresponding to the conflict class under recovery, but will discard them, since it is receiving them via the recovery. The recoverer replica will eventually reach the end of the log. At this point the fifth phase is performed to finish recovery. In the end-of-recovery handshake three different roles are involved, namely, the recoverer and recovering replicas, and the master replica of the conflict class under recovery. There is a race condition, since the master produces updates from locally executed transactions, while the recoverer needs to know the last update to be forwarded to the recovering replica and the recovering replica needs to know from which transaction it has to start to process update transactions. When the recoverer reaches the end of the log, it sends a request-end-of-recovery message to all replicas (only the master needs this message in the failure-free case). The master will then piggyback with the next update it forwards an end-of-recovery marker. This marker will indicate to the recoverer that this is the last update to be sent to the recovering replica. The marker will tell the recovering replica which is the last update part of the recovery, and therefore it will know from which point it has to apply updates received from the master replica.

The online recovery for Nodo, as stated before, allows recovering conflict classes independently. In fact, once a conflict class is recovered, the recovering replica could become master of that class. Additionally, several conflict classes can be recovered in parallel using different recoverers for each of them. The online recovery for Nodo has been designed to be adaptive. The goal is to obtain the highest performability.

If the replicated system has a low load, the resources employed to recover the replica are increased (i.e., increasing the number of recoverers). If there is a peak load, during recovery, then the resources devoted to recovery can be decreased (i.e., decreasing the number of recoverers). In this way, performability is maximized. Nodo also enables dealing with overlapping recoveries in parallel. If a batch of sites is recovered in a time interval, they will start recovery at slightly different times. The recovery protocol takes care of recovering simultaneously conflict classes for all recovering replicas it knows. In this way, if one replica starts recovery after another one has recovered two conflict classes, they will perform simultaneously the recovery of the remaining N-2 conflict classes, being N the number of conflict classes, and alone the recovery of the first two conflict classes. This enables a more efficient dissemination of the recovery messages (e.g., exploiting multicast) and efficiently recovering batches of replicas.

Key applications

Online recovery is a crucial technique to provide true high availability, that is, 24×7 operation. Its main potential users are all those organizations that provide services that should be continuously available. Among these potential users one can find enterprise data centers, software as a service (SaaS) platforms, services governed by service level agreements (SLAs), services for critical infrastructures (health-care, energy, police, etc.).

Cross-references

▶ Data Replication
▶ Replica control

Recommended Reading

1. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison Wesley, 1987.
2. Castro M. and Liskov B. Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst., 20(4):398–461, 2002.
3. Gançarski S. , Naacke H., Pacitti E., and Valduriez P. The leganet system: Freshness-aware transaction routing in a database cluster. Inform. Syst., 32(2):320–343, 2007.
4. Gashi I., Popov P., and Strigini L. Fault Tolerance via Diversity for Off-The-Shelf Products: a Study with SQL Database Servers. IEEE Trans. Depend. Secur. Comput., 4(4):280–294, 2007.
5. Jiménez-Peris R., M. Patiño-Martínez, and Alonso G. Non-Intrusive, Parallel Recovery of Replicated Data. In Proc. 21st Symp. on Reliable Distributed Syst., 2002, pp. 150–159.
6. Kemme B. and Alonso G. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 134–143.
7. Kemme B. and Alonso G. A New Approach to Developing and Implementing Eager Database Replication Protocols. ACM Trans. Database Syst., 25(3):333–379, 2000.
8. Kemme B., Bartoli A., and Babaoglu O. Online Reconfiguration in Replicated Databases Based on Group Communication. In Proc. Int. Conf. on Dependable Systems and Networks, 2001, pp. 117–130.
9. Lau E. and Madden S. An Integrated Approach to Recovery and High Availability in an Updatable, Distributed Data Warehouse. In Proc. 32nd Int. Conf. on Very Large Data Bases. 2006, pp. 703–714.
10. Manassiev K. and Amza C. Scaling and Continuous Availability in Database Server Clusters through Multiversion Replication. In Proc. Int. Conf. on Dependable Systems and Networks, 2007, pp. 666–676.
11. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems. Prentice-Hall, 2nd ed., 1999.
12. Pacitti E. and Simon E. Update Propagation Strategies to Improve Freshness in Lazy Master Replicated Databases. VLDB J., 8 (3):305–318, 2000.
13. Patiño-Martínez M., Jiménez-Peris R., Kemme B., and Alonso G. Middle-R: Consistent Database Replication at the Middleware Level. ACM Trans. Comput. Syst., 23(4):375–423, 2005.
14. Pedone F., Guerraoui R., and Schiper A. The Database State Machine Approach. Distributed and Parallel Databases, 14(1): 71–98, 2003.
15. Plattner C. and Alonso G. Ganymed: Scalable Replication for Transactional Web Applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004, pp. 155–174.
16. PostgreSQL PostgreSQL Point in Time Recovery. http://www.postgresql.org/docs/8.0/interactive/backup-online.html.
17. Vandiver B., Balakrishnan H., Liskov B., and Madden S. Tolerating Byzantine Faults in Database Systems using Commit Barrier Scheduling. In Proc. 21st ACM Symp. on Operating System Principles, 2007, pp. 59–72.

Ontological Engineering

▶ Ontology
▶ Ontology Engineering

Ontologies

▶ Gazetteers

Ontologies and Life Science Data Management

Robert Stevens[1], Phillip Lord[2]
[1]University of Manchester, Manchester, UK
[2]Newcastle University, Newcastle-Upon-Tyne, UK

Synonyms

Knowledge management

Definition

Biology is a knowledge-rich discipline. Much of bioinformatics can, therefore, be characterized as *knowledge management*: organizing, storing and representing that knowledge to enable search, reuse and computation.

Most of the knowledge of biology is categorical; statements such as "fish gotta swim, birds gotta fly" cannot be easily represented as mathematical or statistical relationships. These statements can, however, be formalized using ontologies: a form of model which represents the key concepts of a domain.

Ontologies are now widely used in bioinformatics for a variety of tasks, enabling integration and management of multiple data or knowledge sources, and providing a structure for new knowledge as it is created.

Historical Background

Biological knowledge is highly complex. It is characterized not by the large size of the data sets that it uses, but by the large number of data types; from relatively simple data such as raw nucleotide sequence, through to anatomies, systems of interacting entities, to descriptions of phenotype.

In addition to its natural complexity, biology has traditionally operated as a "small science" – with a large number of individual, autonomous laboratories working independently. This has resulted in highly heterogeneous data; in addition to the natural complexity of the data, knowledge is often represented in many different ways [5]. There are, for example, at least twenty different file formats for representing DNA sequence.

Ontologies can be used to enable the knowledge management that overcomes these two forms of complexity. First, they can be used to represent complex, categorical knowledge of the sort common in biology. Second, by describing the heterogeneity of the representation of knowledge, they can provide a common, shared understanding that can be used to overcome this heterogeneity.

In post-genomic biology, the first of these has been the most common usage. Here, ontologies are used to generate a controlled vocabulary; for this, the Gene Ontology (GO) [14] provides the paragon for biological sciences. It represents three key aspects of genetics; the *molecular function* of a gene (product), the *biological process* in which the product is involved, and the *cellular component* in which it is located. GO has been used to annotate many genomes and has been used for annotations in UniProt and InterPro. Following on from the success of GO, the Open Biomedical Ontologies (OBO) now provides controlled vocabularies for describing many aspects of biological knowledge (http://obo.sf.net).

The second major use has been to enable access to or querying over multiple independent data sources. EcoCyc [8], for example, uses an ontology to provide a schema to integrate genome, proteome data and a number of pathway resources, while RiboWeb [1] was a similar style of ontology driven application for storing, managing and analyzing data from experiments on ribosomal structure. The TAMBIS system [7] used an ontology to mediate queries to a number of different data sources.

Most of these examples are *post-hoc* additions to existing systems; GO, for example, presents knowledge which is already present in other, less formal, representations. More recently, however, there has been a shift to the use of ontologies as a primary representation. The MGED Ontology (MO) [17] provides a vocabulary for reporting microarray experiments, while the Systems Biology Ontology (SBO) (http://www.ebi.ac.uk/sbo/) supports the representation of systems biology models.

Over the past decade, the use of ontologies has now become well-entrenched as a tool for organizing and structuring biomedical knowledge and, therefore, has become a key part of life science knowledge management [3].

Foundations

It was recognized early in bioinformatics that there is a massive problem with heterogeneous representations of data [5]. Such heterogeneities, particularly at the semantic level, exist both in the meanings of the structures that hold values (the schema) and in the meanings of the values themselves. To query or analyze meaningfully across data from different sources, therefore, there is a necessary reconciliation step to enable the data to be understood. A database, for example, might have either separate tables for substrate or product, or just one for reactant; as an orthogonal issue, a chemical might be "acetic acid" in one or "ethanoic acid" in another. Both types of mis-match need to be overcome.

This heterogeneity can occur both in descriptions of biology and also bioinformatics: it is possible to disagree on which genes are involved in a process, what those genes are called and what structure is used to hold information about those genes.

Ontologies provide a computable mechanism for working around these problems; they describe the entities and what is known about these entities within the data of biology, and provide a set of labels to describe these entities and their properties. As a result, an ontology can be used to describe the entities within a biological database. Terms from the Gene Ontology, for example, are used to describe the major functional attributes of gene products in many databases; this, in turn, allows comparative studies of genes and their actions across species.

Ontologies need to be represented in a language; these are often called knowledge representation languages. They have a set of language elements for describing categories (also called classes, types, concepts or universals) of instances (also called objects, entities, individuals or particulars). The languages vary in their expressivity – that is, how much is it possible to say about what these elements mean. For example, if we state that *ArB*, where *A* and *B* are classes and *r* is a relationship, does this mean that every *A* has a *B* related to it by *r*; that for any *A* with a *r* relation, this *r* is to a *B*; to how many *B*'s can an *A* have a relationship; that *r* has an inverse relationship, and so on. Some languages allow only trees of categories to be made, others more complex graphs. Finally, languages differ in their computational amenability [11].

Ontologies have found a variety of uses within bioinformatics:

- **Reference Ontology:** An ontology can be used simply as a reference, encompassing what is understood about a domain with high-fidelity. Such an ontology is not skewed by any application bias, except that of correctness.

- **Controlled Vocabulary:** The labels on the categories in an ontology provide a vocabulary with which discussion of those categories can be accomplished. By committing to use that vocabulary – controlling the words used in communication – a controlled vocabulary is established. This is the principal means of overcoming a large portion of heterogeneity.

- **Computational Component:** An ontology can form a component in a software application. The strict semantics of its representation language can be used to make inferences from data described in terms of that ontology. This can simply be retrieving all the children of a given category (all instances of a child are also instances of the parent) to recognizing membership of a category based on facts known about an object.

  There are many technical and social difficulties associated with using an ontology for data management. The choice of representation language can be key; an ontology for use as a computational component probably needs a more computationally amenable and expressive language than an ontology intended to provide a controlled vocabulary. It is often hard to engage with domain experts to ensure that the ontology reflects the domain, while maintaining ontological precision. There are a number of methodologies for ontology building [6], but the discipline is still nascent. Finally, adapting and updating the ontology when it is already in use can require rigorous, yet flexible policies.

Key Applications

Perhaps the best known ontology in biology is the Gene Ontology (GO) [14]. It started in 1998 with an aim of enabling queries across multiple databases for the key aspects or properties of the genes or proteins; it achieves this not by schema reconciliation but enabling the augmentation of existing knowledge; in short, it is one of the best examples of a reference ontology.

Another domain well served by reference models is that of anatomy. The Foundational Model of Anatomy, for example, aims to provide a "symbolic modeling of the structure of the human body in a computable form that is also understandable by humans" [12]. The aim is that this ontology provides a common representation into which others can be mapped.

One of the early uses for ontologies was enabling schema and value reconciliation. The TAMBIS [7] system was an early example. It uses an ontological representation of entities and their properties in biology expressed in a description logic [2]. These descriptions, that could be composed to more complex concept descriptions, were then transformed to queries against bioinformatics analysis services capable of retrieving instances of the concepts within the ontology. The ontology, then, could tell that the user that a `Protein` might have one or more `Homologs`, while the system would understand that a BLAST search might reveal these `Homologs`. A concept therefore also defined a query plan. More recently, the BioPAX ontology [10] provides a schema for representing biological pathway data; the different contributing databases could then release knowledge in the format. Both of these operate on the level of schema, but reconciliation to a common model also occurs at the level of the values held within a schema. The Gene Ontology, for example, in providing a controlled vocabulary for the functional attributes of gene products has allowed many genome resources to use common values within their schema.

Both BioPAX and TAMBIS enabled querying by assigning objects to categories in the ontology. Ontologies represent the properties by which objects can be recognized to be a member of a category. If these properties are recognizable computationally, then the ontology can be used to classify these members automatically. This approach has been used to classify the phosphatase proteins from three parasite genomes [4]. A final approach to ontological querying is to use the ontology as a basis for statistical analysis of individuals annotated with these ontologies. GO has been widely used for this purpose [9,15].

Finally, ontologies have begun to be used for the representation of metadata about primary experimental data. The MGED society has led the way with the MGED Ontology (MO) which has been used for describing microarray data[17]. This ontology describes a number of aspects about an experiment including: the biological material used; experimental design and microarray equipment. Similar work is now underway for describing proteomics experiments [13]. These are coming together in the Ontology for Biomedical Investigations OBI that is providing a general framework for describing the protocols and analyses for many different kinds of experiment [16].

Future Directions

In the past decade ontologies have come to form a major aspect of information management in the life sciences. The field of ontology development in the life sciences now faces several challenges in the short and medium term.

- The wide scope of biology is a challenge; to describe many parts of it, also needs descriptions of closely related areas such as chemistry, geology and geography.
- Ontologies are starting to get very large. It is not clear whether current methodologies are scalable, both in terms of building, maintaining or using them. This has many implications for the formal expressive structures of the knowledge representation language, the ability to support modularity of these languages, and the social processes used to build ontologies.
- Dealing with change both as a result of the ontology development process and, perhaps more importantly, as a result of changes in knowledge itself. There are many different techniques for dealing with the former situation; there are many fewer for dealing with the latter. If datasets gathered over a long period of time are to be understood in the future, it may become as important to understand what was thought in the past as it is to manage the current sate of knowledge.
- Currently many ontologies deal with a single level of granularity or the view point of a single discipline building sophisticated, computationally amenable ontologies necessitates crossing boundaries of granularity and discipline. It remains, however, unclear how to integrate these sorts of ontology.
- Ontologies currently fulfill the luxury end of the metadata market; they can be very expensive to build, maintain and deploy. Lower the cost is critical. Probably the best way to achieve this is to make them easier for domain scientists to build which leads to a second challenge; maintaining usability of ontologies and representation languages, while increasing their scale and computability.

It seems clear that ontologies will be in heavy use in the future within the life sciences. How well these challenges are answered will determine the uses to which they are put.

Cross-references

▶ Ontology
▶ Query Languages for Ontological Data

Recommended Reading

1. Altman R., Bada M., Chai X. Whirl Carillo M., Chen R., and Abernethy N. RiboWeb: an ontology-based system for collaborative molecular biology. IEEE Intell. Syst., 14(5):68–76, 1999.
2. Baader F., Calvanese D., McGuinness D., Nardi D., and Patel-Schneider P. (eds.) The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003.
3. Bodenreider O. and Stevens R. Bio-ontologies: current trends and future directions. Brief Bioinform., 7(3):256–274, 2006.
4. Brenchley R., Tariq H., McElhinney H., Szoor B., Stevens R., Matthews K., and Tabernero L. The TriTryp Phosphatome analysis of the protein phosphatase catalytic domains. BMC Genome, 8:434, 2007.
5. Davidson S., Overton C., and Buneman P. Challenges in integrating biological data sources. J. Comput. Biol., 2(4):557–572, 1995.
6. Fernàndez-Lòpez M. and Gòmez-Pèrez A. Overview and analysis of methodologies for building ontologies. Knowl. Eng. Rev., 17 (2):129–156, 2002.
7. Goble C.A., Stevens R., Ng G., Bechhofer S., Paton N.W., Baker P., Peim M., and Brass A. Transparent access to multiple bioinformatics information sources. IBM Syst. J., Special issue on deep computing for the life sciences, 40(2):532–552, 2001.
8. Karp P., Riley M., Saier M., Paulsen I., Paley S., and Pellegrini-Toole A. The EcoCyc and metacyc databases. Nucleic Acids Res., 28:56–59, 2000.
9. Lord P.W., Stevens R., Brass A., and Goble C.A. Investigating semantic similarity measures across the Gene Ontology: the relationship between sequence and annotation. Bioinformatics, 19–(10):1275–1283, 2003.
10. Luciano J. PAX of mind for pathway researchers. Drug Discov. Today, 10:937–942, 2005.
11. Ringland G. and Duce D. Approaches to Knowledge Representation: An Introduction Knowledge-Based and Expert Systems Series. John Wiley, Chichester, 1988.
12. Rosse C. and Mejino J.L.V. A reference ontology for bioinformatics: the foundational model of anatomy. J. Biomed. Inform., 36:478–500, 2003.
13. Taylor C., Paton N., Lilley K., Binz P., Julian Jr R., Jones A., Zhu W., Apweiler R., Aebersold R., Deutsch E., Dunn M., Heck A., Leitner A., Macht M., Mann M., Martens L., Neubert T., Patterson S., Ping P., Seymour S., Souda P., Tsugita A., Vandekerckhove J., Vondriska T., Whitelegge J., Wilkins M., Xenarios I., Yates J. (3rd) and Hermjakob H. The minimum information about a proteomics experiment (MIAPE). Nat. Biotech., 25:887–893, 2007.
14. The Gene Ontology Consortium Gene Ontology: Tool for the Unification of Biology. Nat. Gene., 25:25–29, 2000.
15. Wang H., Azuaje F., Bodenreider O., and Dopazo J. Gene expression correlation and gene ontology-based similarity: an assessment of quantitative relationships. In Proc. IEEE Symp. on Computational Intelligence in Bioinformatics and Computational Biology. 2004, pp. 25–31.
16. Whetzel P., Brinkman R., Causton H., Fan L., Field D., Fostel J., Fragaso G., Gray T., Heiskanen M., Hernandez-Boussard T., Morrison N., Parkinson H., Rocca-Serra P., Sansone S.A., Schober D., Smith B., Stevens R., Stoeckert C., Taylor C., White J., and Wood A. the FuGo working group development of FuGo: An ontology for functional genomics investigations. OMICS J. Integrat. Biol., 10:199–204, 2006.
17. Whetzel P.L., Parkinson H., Causton H.C., Fan L., Fostel J., Fragoso G., Game L., Heiskanen M., Morrison N., Rocca-Serra P., Sansone S.A., Taylor C., White J., and Stoeckert C.J. The mged ontology: a resource for semantics-based description of microarray experiments. Bioinformatics, 22(7):866–873, 2006.

Ontology

Tom Gruber
RealTravel, Emerald Hills, CA, USA

Synonyms

Computational ontology; Semantic data model; Ontological engineering

Definition

In the context of computer and information sciences, an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application. In the context of database systems, ontology can be viewed as a level of abstraction of data models, analogous to hierarchical and relational models, but intended for modeling knowledge about individuals, their attributes, and their relationships to other individuals. Ontologies are typically specified in languages that allow abstraction away from data structures and implementation strategies; in practice, the languages of ontologies are closer in expressive power to first-order logic than languages used to model databases. For this reason, ontologies are said to be at the "semantic" level, whereas database schema are models of data at the "logical" or "physical" level. Due to their independence from lower level data models, ontologies are used for integrating heterogeneous databases, enabling interoperability among disparate systems, and specifying interfaces to independent, knowledge-based services. In the technology stack of the Semantic Web standards

[1], ontologies are called out as an explicit layer. There are now standard languages and a variety of commercial and open source tools for creating and working with ontologies.

Historical Background

The term "ontology" comes from the field of philosophy that is concerned with the study of being or existence. In philosophy, one can talk about an ontology as a theory of the nature of existence (e.g., Aristotle's ontology offers primitive categories, such as substance and quality, which were presumed to account for All That Is). In computer and information science, ontology is a technical term denoting an artifact that is *designed* for a purpose, which is to enable the modeling of knowledge about *some* domain, real or imagined.

The term had been adopted by early Artificial Intelligence (AI) researchers, who recognized the applicability of the work from mathematical logic [6] and argued that AI researchers could create new ontologies as computational models that enable certain kinds of automated reasoning [5]. In the 1980s the AI community came to use the term ontology to refer to both a theory of a modeled world (e.g., a Naïve Physics [5]) and a component of knowledge systems. Some researchers, drawing inspiration from philosophical ontologies, viewed computational ontology as a kind of applied philosophy [10].

In the early 1990s, an effort to create interoperability standards identified a technology stack that called out the ontology layer as a standard component of knowledge systems [8]. A widely cited web page and paper [3] associated with that effort is credited with a deliberate definition of ontology as a technical term in computer science. The paper defines ontology as an "explicit specification of a conceptualization," which is, in turn, "the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold among them." While the terms specification and conceptualization have caused much debate, the essential points of this definition of ontology are:

- An ontology defines (specifies) the concepts, relationships, and other distinctions that are relevant for modeling a domain.
- The specification takes the form of the definitions of representational vocabulary (classes, relations, and so forth), which provide meanings for the vocabulary and formal constraints on its coherent use.

One objection to this definition is that it is overly broad, allowing for a range of specifications from simple glossaries to logical theories couched in predicate calculus [9]. But this holds true for data models of any complexity; for example, a relational database of a single table and column is still an instance of the relational data model. Taking a more pragmatic view, one can say that ontology is a tool and product of engineering and thereby defined by its use. From this perspective, what matters is the use of ontologies to provide the representational machinery with which to instantiate domain models in knowledge bases, make queries to knowledge-based services, and represent the results of calling such services. For example, an API to a search service might offer no more than a textual glossary of terms with which to formulate queries, and this would act as an ontology. On the other hand, today's W3C Semantic Web standard suggests a specific formalism for encoding ontologies (OWL), in several variants that vary in expressive power [7]. This reflects the intent that an ontology is a specification of an abstract data model (the domain conceptualization) that is independent of its particular form.

Foundations

Ontology is discussed here in the applied context of software and database engineering, yet it has a theoretical grounding as well. An ontology specifies a vocabulary with which to make assertions, which may be inputs or outputs of knowledge agents (such as a software program). As an *interface specification*, the ontology provides a language for communicating with the agent. An agent supporting this interface is not required to use the terms of the ontology as an *internal encoding* of its knowledge. Nonetheless, the definitions and formal constraints of the ontology do put restrictions on what can be *meaningfully* stated in this language. In essence, committing to an ontology (e.g., supporting an interface using the ontology's vocabulary) requires that statements that are asserted on inputs and outputs be *logically consistent* with the definitions and constraints of the ontology [3]. This is analogous to the requirement that rows of a database table (or insert statements in SQL) must be consistent with integrity constraints, which are stated declaratively and independently of internal data formats.

Similarly, while an ontology must be formulated in *some* representation language, it is intended to be a semantic level specification – that is, it is independent of data modeling strategy or implementation. For instance, a conventional database model may represent the identity of individuals using a primary key that assigns a unique identifier to each individual. However, the primary key identifier is an artifact of the modeling process and does not denote something in the domain. Ontologies are typically formulated in languages which are closer in expressive power to logical formalisms such as the predicate calculus. This allows the ontology designer to be able to state semantic constraints without forcing a particular encoding strategy. For example, in typical ontology formalisms one would be able to say that an individual was a member of class or has some attribute value without referring to any implementation patterns such as the use of primary key identifiers. Similarly, in an ontology one might represent constraints that hold across relations in a simple declaration (A is a subclass of B), which might be encoded as a join on foreign keys in the relational model.

Ontology engineering is concerned with making representational choices that capture the relevant distinctions of a domain at the highest level of abstraction while still being as clear as possible about the meanings of terms. As in other forms of data modeling, there is knowledge and skill required. The heritage of computational ontology in philosophical ontology is a rich body of theory about how to make ontological distinctions in a systematic and coherent manner. For example, many of the insights of "formal ontology" motivated by understanding "the real world" can be applied when building computational ontologies for worlds of data [4]. When ontologies are encoded in standard formalisms, it is also possible to reuse large, previously designed ontologies motivated by systematic accounts of human knowledge or language [11]. In this context, ontologies embody the results of academic research, and offer an operational method to put theory to practice in database systems.

Key Applications

Ontologies are part of the W3C standards stack for the Semantic Web, in which they are used to specify standard conceptual vocabularies in which to exchange data among systems, provide services for answering queries, publish reusable knowledge bases, and offer services to facilitate interoperability across multiple, heterogeneous systems and databases. The key role of ontologies with respect to database systems is to specify a data modeling representation at a level of abstraction above specific database designs (logical or physical), so that data can be exported, translated, queried, and unified across independently developed systems and services. Successful applications to date include database interoperability, cross database search, and the integration of web services.

Cross-references

► Data Model
► Data Modeling
► Knowledge Base
► Knowledge Engineering

Recommended Reading

1. Berners-Lee T., Hendler J., and Lassila O. The semantic web. Scientific American, May 2001.
2. Gruber T.R. A translation approach to portable ontology specifications. Knowl. Acquisition, 5(2):199–220, 1993.
3. Gruber T.R. Toward principles for the design of ontologies used for knowledge sharing. Int. J. Hum. Comput. Stud., 43 (5–6):907–928, 1995.
4. Guarino N. Formal ontology, conceptual analysis and knowledge representation. Int. J. Hum. Comput. Stud., 43(5–6):625–640, 1995.
5. Hayes P.J. The second naive physics manifesto. In Formal Theories of the Common-Sense World, Moore (eds.). Hobbs, Ablex, Norwood, MA, 1985.
6. McCarthy J. Circumscription – a form of non-monotonic reasoning. Artif. Intell., 5(13):27–39, 1980.
7. McGuinness D.L. and van Harmelen F. OWL web ontology language. W3C Recommendation, February 10, 2004. Available online at: http://www.w3.org/TR/owl-features/.
8. Neches R., Fikes R.E., Finin T., Gruber T.R., Patil R., Senator T., and Swartout W.R. Enabling technology for knowledge sharing. AI Mag., 12(3):16–36, 1991.
9. Smith B. and Welty C. Ontology – towards a new synthesis. In Proc. Int. Conf. on Formal Ontology in Information Systems, 2001.
10. Sowa J.F. Conceptual Structures: Information Processing in Mind and Machine, Addison Wesley, Reading, MA, 1984.
11. Standard Upper Ontology Working Group (SUO). IEEE P1600.1. Available online at: http://suo.ieee.org/.

Ontology Acquisition

► Ontology Elicitation

Ontology Argumentation

► Ontology Elicitation

Ontology Elicitation

Pieter De Leenheer
Vrije Universiteit Brussel, Collibra nv, Brussels,
Belgium

Synonyms

Ontology acquisition; Ontology learning; Ontology argumentation; Ontology negotiation; Knowledge creation

Definition

Ontology elicitation embraces the family of methods and techniques to explicate, negotiate, and ultimately agree on a partial account of the structure and semantics of a particular domain, as well as on the symbols used to represent and apply this semantics unambiguously.

Ontology elicitation only results in a *partial* account because the formal definition of an ontology cannot completely specify the intended structure and semantics of each concept in the domain, but at best can approximate it. Therefore, the key for scalability is to reach the appropriate amount of consensus on relevant ontological definitions through an effective meaning negotiation in an efficient manner.

Historical Background

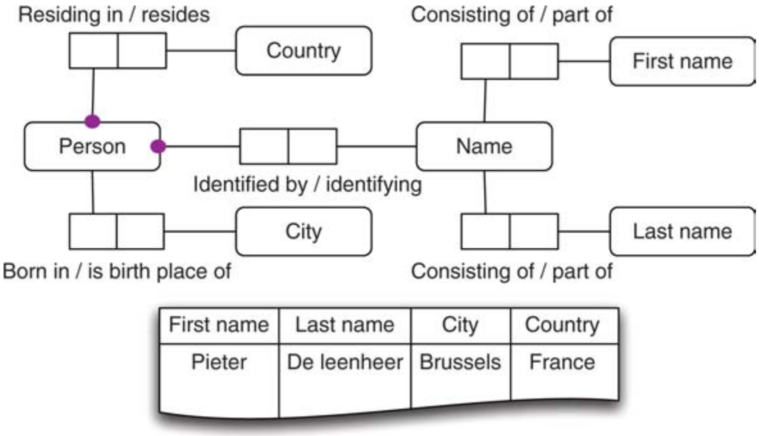
Ontology elicitation is based on techniques of *knowledge acquisition*, a subfield of AI that is concerned with eliciting and representing knowledge of human experts so that it can later be used in some application. Two typical knowledge acquisition methods can be distinguished:

1. *Top-down* (deductive) *knowledge elicitation* techniques are used to acquire knowledge directly from human domain experts. Examples include interviewing, case study, and mind mapping techniques.
2. *Bottom-up* (inductive) *machine learning* techniques use different methods to infer knowledge (e.g., concepts and rules) patterns from sets of data. A well-known example is *formal concept analysis* [10].

More formal methods for top-down knowledge acquisition use *knowledge modeling* as a way of structuring projects, acquiring and validating and storing knowledge for future use. Knowledge models include: symbolic character-based languages (e.g., logic, OWL), diagrammatic representations (networks, ladders, taxonomies, concept maps), tabular representations (e.g., matrices), structured text (e.g., hypertext) [16], and conceptual modeling.

Conceptual Modeling

Certain methods and techniques from the database field for *conceptual modeling* (e.g., ER, UML, dataflow diagrams) have been proven useful for ontology elicitation. For example, in [13], ORM/NIAM has been adopted. Figure 1 shows an example of a minimal ORM diagram (on top) explicating the semantics that is implicit in the relational table schema and population (on the bottom). For example, this ORM diagram already reveals what the table cannot, the semantics of



First name	Last name	City	Country
Pieter	De leenheer	Brussels	France

Ontology Elicitation. Figure 1. Illustration of a minimal ORM diagram explicating the implicit semantics for a relational table.

the relation of attribute "person" to attributes "city" and "country," and that "first name" and "last name" are both part of a "name." Furthermore, "city" and "country" appear not to be related at all.

A key characteristic of NIAM/ORM is that the analysis of information is based on natural language. This brings the advantage that the analysis can be done by the domain experts using their own vocabulary, and hence avoiding invalid interpretations. Furthermore, this attribute-free approach seen in the NIAM/ORM approach promotes semantic stability.

Data Schema Versus Ontology

Data models, such as data or XML schemas, typically specify the structure and integrity of data sets. Hence, building data schemas for an enterprise usually depends on the specific needs and tasks that have to be performed within this enterprise. *Data engineering* languages such as SQL aim to maintain the integrity of data sets and only use a typical set of language constructs to that aim, e.g., foreign keys. The schema vocabulary is basically to be understood intuitively (via the terms used) by the human database designer(s). The semantics of data schemas often constitute an informal agreement between the developers and an intended group of users of the data schema, and finds its way only in application programs that use the data schema instead of manifesting itself as an agreement that is shared amongst the community [20]. When new functional requirements pop up, the schema is updated on the fly. One designated individual usually controls this schema update process.

In (collaborative) ontology elicitation, however, absolute meaning is essential for all practical purposes, hence all elements in an ontology must ultimately be the result of agreements among human agents such as designers, domain experts, and users. In practice, correct and unambiguous reference to concepts or entities in the schema vocabulary is a real problem; often harder than agreeing about their properties, and obviously not solved by assigning system-owned identifiers.

Foundations

In collaborative ontology elicitation, multiple stakeholders have overlapping or contradicting perspectives about the intended structure, semantics, and vocabulary of the domain concepts [5]. This is principally caused by three facts: (i) no matter how expressive ontologies might be, they are all in fact lexical representations of concepts, relationships, and semantic constraints; (ii) linguistically, there is no bijective mapping between a concept and its lexical representation; and (iii) concepts can have different properties and values in different contexts of use. Hence, humans play an important role in the interpretation and negotiation of meaning during the elicitation and application of ontologies [7]. These principles can be illustrated by considering Stamper's *semiotic ladder* [21] that consists of six views or levels on signs from the perspective of physics, empirics, syntactics, semantics, pragmatics and the social world, that together form a complex conceptual structure. In this article, we only consider syntactical or *lexical* level, *semantic* level, and *pragmatic* level (Fig. 2). Ontology elicitation can be considered as a process that gradually takes ontological elements through these levels.

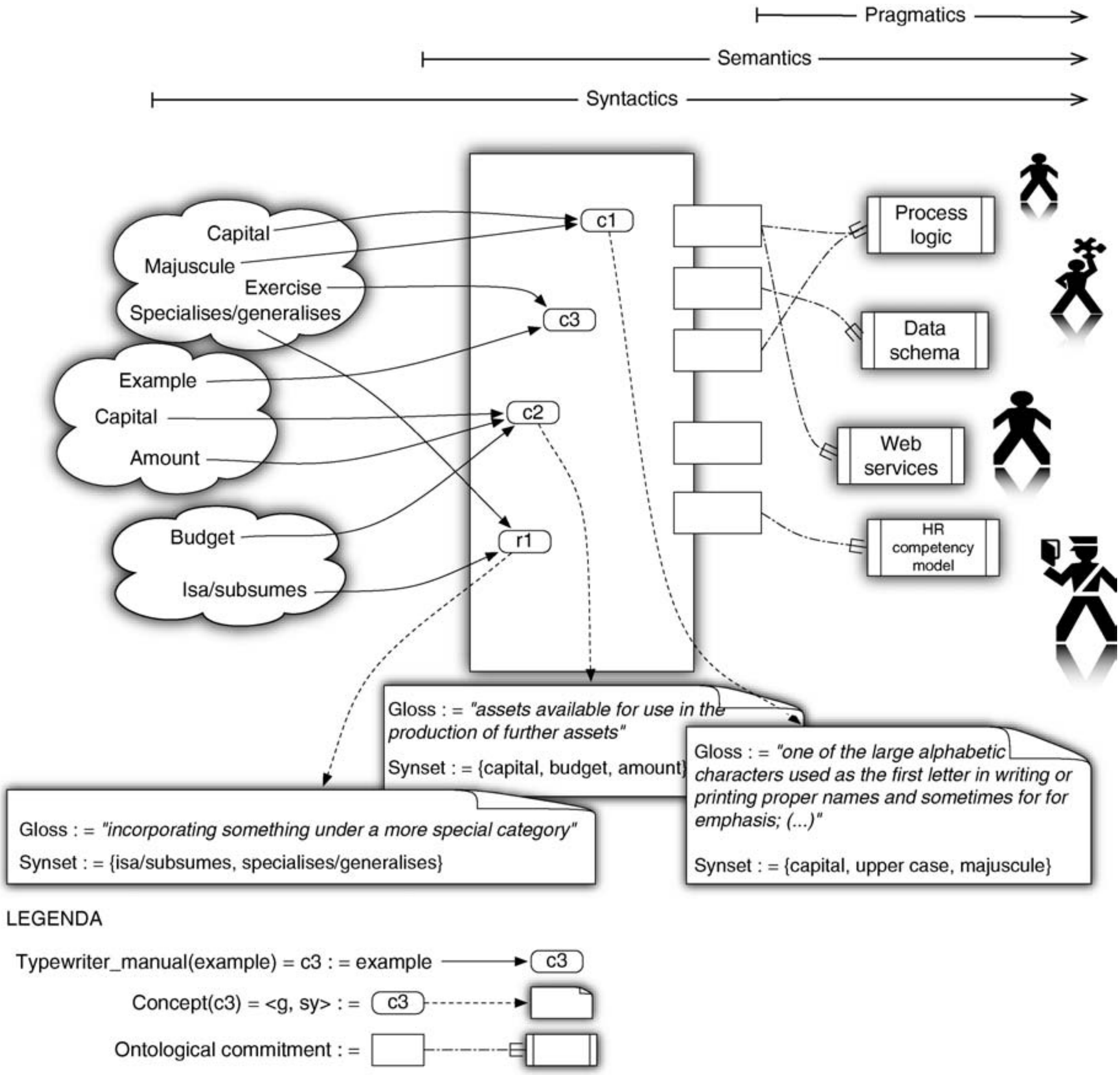
Lexical Versus Semantic Level

At the start of the elicitation of an ontology, its basic terminology for labeling concepts and relationships are extracted from various resources such as a text corpus [3], existing schemas [16], from so-called *serious games* [19] or rashly formulated by human domain experts through, e.g., tagging [22]. Many ontology engineering approaches focus merely on the conceptual modeling task, hence the distinction between lexical level (term for a concept) and semantic level (the concept itself) is often weak or ignored. In order to represent concepts and relationships lexically, they usually are given a uniquely identifying term (or label). However, the meaning of a concept behind a lexical term is influenced by the *elicitation context*, which is the context of the resource the term was extracted from. When eliciting and unifying information from multiple sources, this can easily give rise to misunderstandings and ambiguities, therefore the meaning of all terms used for ontology representation purposes should be articulated appropriately.

This is illustrated in Fig. 2: the full arrows denote the *meaning articulation* mappings between terms in organizational vocabularies (cloud on the left) and unique *concept identifiers* (e.g., c_1, r_1, etc.). The mapping of each unique concept identifiers to a particular explication of a meaning, i.e., a *concept definition* is defined by the dashed arrows.

Lexical Variability and Reusability

Even within one conversation, it turned out that in a less than a quarter of the cases, two individuals use



Gloss : = *"assets available for use in the production of further assets"*

Synset : = {capital, budget, amount}

Gloss : = *"one of the large alphabetic characters used as the first letter in writing or printing proper names and sometimes for for emphasis; (...)"*

Synset : = {capital, upper case, majuscule}

Gloss : = *"incorporating something under a more special category"*

Synset : = {isa/subsumes, specialises/generalises}

Typewriter_manual(example) = c3 : = example

Concept(c3) = <g, sy> : =

Ontological commitment : =

Ontology Elicitation. Figure 2. Three levels of ontology elicitation: lexical level, semantic level, and pragmatic level.

the same symbolic reference for a concept, and hence the freedom to use synonyms should be accommodated [8]. To engender creativity, domain experts should initially be allowed to use their own vocabularies, instead of being harshly restricted by an unfamiliar controlled taxonomy dictated by a central authorship. Gradually, this variability will converge towards one or more vocabularies that are commonly accepted.

For example, thousands of shared vocabularies or so-called folksonomies emerge, are sold and advertised, prosper or wither in a self-organizing manner on Web 2.0, through reuse and adaptation of natural language labels for tagging their resources. Natural language labels for concepts and relationships bring along their inherent ambiguity and variability in interpretation. Folksonomies provide on the one hand an unbounded reusability potential for specific reference in a given application context, which is important for scalable ontology elicitation. On the other hand, however, an analysis of multiple contexts is generally needed to disambiguate successfully [1,5].

Semantic Versus Pragmatic Level

The meaning articulation mappings and the concept definition service respectively provide unambiguous reference and semantic explication of terms, independent of the preferred vocabulary. However, the meaning of these concepts should be further formalized for appropriately serving application purposes, by combining and linking them with other concepts, and axiomatizing them with semantic constraints and rules. In the section on applications, there is an overview of typical applications, including process logic and (legacy) information system interoperability, web service orchestration, and competency model gap analysis in human resources (HR).

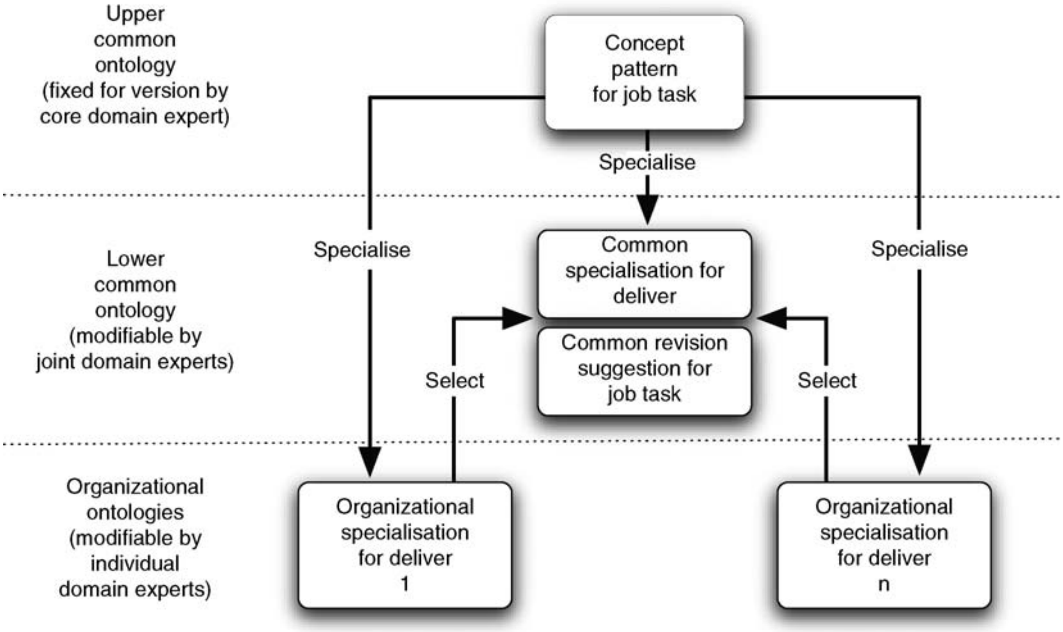
The relevant properties and values for the concepts to be agreed on, depend on the application requirements. For the sake of scalability, as in any realistic system or knowledge engineering scenario, in ontology elicitation, (parts of) existing semantic resources are reused and adopted as much as possible for new application purposes. This asks for a methodological trade-off between the reuse of relevant consensus from existing application contexts as much as possible, while allowing specific variations for new application requirements at stake to be collaboratively negotiated, based on (parts of) existing consensus.

Figure 3 illustrates this with a model for collaborative ontology elicitation, as introduced by [7], and inspired by the Delphi method (http://en.wikipedia.org/wiki/Delphi_method.). For bootstrapping the elicitation of a concept, knowledge workers are given a *pattern* that defines the current insights and interests of the community for that type of concepts. The example here concerns the elicitation of a concept "Deliver," which is a type of "Job Task." Each of the stakeholding organizations elicit the relevant properties and values of "Deliver" by specializing the pattern abstracted from "Job Task." If no pattern exists, it is bootstrapped by the core domain experts overlooking the domain.

Divergence and Conflict

Divergence is the point in collaborative ontology elicitation where domain experts disagree or have a conflict about the meaning of some concept in such a way that consequently their ontologies evolve in widely varying directions. Although they share common goals for doing business, divergent knowledge positions appear as a natural consequence when people collaborate in order to come to a unique common understanding.

Rather than considering it to be a problem, conflicts should be seen as an opportunity to *negotiate* about the subtle differences in interpretation of the domain, which will ultimately converge to a shared understanding disposed of any subjectivity. However, meaning conflicts and ambiguities should only be resolved when relevant. It is possible that people have


Upper common ontology (fixed for version by core domain expert)
Concept pattern for job task
Specialise
Lower common ontology (modifiable by joint domain experts)
Specialise
Common specialisation for deliver
Specialise
Common revision suggestion for job task
Select
Select
Organizational ontologies (modifiable by individual domain experts)
Organizational specialisation for deliver 1
Organizational specialisation for deliver n

Ontology Elicitation. Figure 3. A model for collaborative ontology elicitation.

alternative conceptualizations in mind for business or knowledge they do not wish to share. Therefore, in building the shared ontology, the individual ontologies of the various partners only need to be aligned insofar necessary, in order to avoid wasting valuable modeling time and effort.

Basically they only need to agree on a common specialization of the current properties present in the pattern. However, it could be the case that a considerable part of the stakeholders identifies new relevant properties, or see other properties to be obsolescent. This provides a feedback suggestion to revise the patterns for a next version of the ontology.

Relevant techniques for collaborative ontology negotiation and argumentation include [14,17].

Convergence and Patterns

Once, a common specialization is agreed on, it is lifted up in the upper common levels of the ontology. Gradually, this would result in the emergence of increasingly stable generally deployable ontology patterns that are key for enabling future business interoperability needs in a scalable manner [2,6,9,7].

Key Applications

Ontologies have become an integral part of many academic and industrial applications in various domains, including Semantic Web services, regulatory compliance, and human resources.

Semantic Web Services

Service-oriented (SOA) is an architecture that relies on *service-orientation* as its fundamental design principle. In a SOA environment, independent services can be accessed without knowledge of their underlying platform implementation. Within this paradigm, the creation of automation logic is specified in the form of services. Service orientation is another design paradigm that provides a means for achieving a separation of concerns, which obviously increases the potential for software reusability.

The Semantic Web aims to make data accessible and understandable to intelligent machine processing. Semantic Web services additionally aim to do the same for services available on the Semantic Web, targeting automation of service discovery, composition and invocation. For describing Semantic Web services, it is required to elicit the so-called "domain ontologies" or that formalize the knowledge necessary for capturing the meaning of services and exchanged data. In other words, given a particular business goal, the domain ontologies enable the weaving of the relevant concerns that are separated in relevant services.

A key challenge here is to overcome the *ontology-perspicuity bottleneck* [11] that constrains the use of ontologies, by finding a compromise between top-down imposed formal semantics expressed in expert language and bottom-up emerging real-world semantics expressed in layman user language.

For more on infrastructure, theory, business aspects, and experiences on ontology elicitation and management for Semantic Web applications, see [12].

Regulatory Compliance

Businesses and government must be able to show compliance of their outputs, and often also of their systems and processes, to specific regulations. Demonstrable evidence of this compliance is increasingly an auditable consideration and required in many instances to meet acceptable criteria for good corporate governance. Moreover the number and the complexity of applicable regulations in Europe and elsewhere is increasing. This includes mandatory compliance audits and assessments against numerous regulations and best practice guidelines over many disciplines and against many specific criteria. The implementation of information communications technology also means that previous manual business processes are now being performed electronically and the degree of compliance to applicable regulations depends on how the systems have been designed, implemented and maintained. Keeping up with the rate of new regulations for a major corporation and small business alike is a never ending task. What is the answer to all of this regulatory complexity? First one should simplify regulations where possible and then apply automatic tools to assist.

The automated data demands of networked economies and an increasingly holistic view on regulatory issues are driving and yet partially frustrating attempts to simplify regulations and statutes. In an ideal world companies and other organizations would have the tools and online services to check and measure their regulatory compliance; and governmental organizations would be able to electronically monitor the results. This requires a more systemic shared approach to regulatory assurance assessment and compliance certification.

Lessig [15] has a simple yet profound thesis "Code is law." The application of this concept taken in conjunction with the emergence of regulatory ontologies opens up a new way of assessing whether burgeoning systems are compliant with regulations they seek and claim to embody. First specific regulations (e.g., data privacy, digital rights management) are converted into and expressed as "Regulatory Ontologies." These ontologies are then used as the base platform for a "Trusted Regulatory Compliance Certification Service."

Over time the resulting ontology describing and managing the areas analyzed can literally replace the regulations and compliance criteria. So much so, it is envisaged that an eventual outcome could be that the formal writing (codification) of future laws will start with the derived ontologies and use intelligent agents to help propose specific legal text which ensures that the policy objectives are correctly coded in law. In addition automatic generation of networked computer applications that are perfectly compliant with the wide variety of directives and laws in any country is one of the ultimate goals of this type of ontology based work.

For an overview of ontology-grounded trusted regulatory compliance, see [18].

Human Resources

Competencies describe the skills and knowledge individuals should have in order to be fit for particular jobs. Especially in the domain of vocational education, having a central shared and commonly used competency model is becoming crucial in order to achieve the necessary level of interoperability and exchange of information, and in order to integrate and align the existing information systems of competency stakeholders like schools or public employment agencies. Only few organizations however, have successfully implemented a company-wide "competency initiative," let alone a strategy for inter-organizational exchange of competency related information.

Several projects (See, e.g., the EU-funded CoDrive project.) aim at contributing to a competency-driven vocational education by using state-of-the-art ontology methodology and infrastructure in order to collaboratively develop a conceptual, shared and formal KR of competence domains.

For a business case study on vocational competency ontology elicitation, see [4].

Future Directions

The ever-changing interoperability requirements between the stakeholding communication partners (See, e.g., diverse (legacy) systems in the open extended enterprise.) requires ontologies to continuously evolve. Usually the domain is too large and complex to be explicated in one single effort, and the knowledge workers understanding of the domain is in continuously changing, requiring timely renegotiation of existing consensus. Therefore, one should not merely focus on the practice of eliciting ontologies in a project-like context, but consider it as a real-time collaborative and continuous process that is integrated with and in the operational processes of the community itself. The shared background of communication partners is continuously negotiated as are the characteristics or values of the concepts that are agreed upon.

There are many additional complexities that should be considered. As investigated in FP6 integrated projects (See, e.g., http://ecolead.vtt.fi/.) on collaborative networked organizations, the different professional, social, and cultural backgrounds among communities and organizations can lead to misconceptions, resulting in costly ambiguities and misunderstandings if not aligned properly. This is especially the case in inter-organizational settings, where there may be many pre-existing organizational sub-ontologies, inflexible data schemas interfacing to legacy data, and ill-defined, rapidly evolving collaborative requirements. Furthermore, participating stakeholders usually have strong individual interests, inherent business rules, and work practices. These may be tacit, or externalized in workflows that are strongly interdependent, hence further complicate the conceptual alignment. Finally this also involves ontology elicitation cost estimation. Simperl and Sure (chapter 7, [12]) propose a parametric cost estimation model for ontologies by identifying relevant cost drivers having a direct impact on the effort invested in ontology elicitation.

For an overview of future directions towards community-driven ontology elicitation and management, see [6].

Cross-references

▶ Emergent Semantics
▶ Ontology
▶ Ontology Engineering

Recommended Reading

1. Bachimont B., Troncy R., and Isaac A. Semantic commitment for designing ontologies: a proposal. In Proc. 13th Int. Conf. on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, 2002, pp. 114–121.
2. Blomqvist E. OntoCase – a pattern-based ontology construction approach. In Proc. OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS, 2007, pp. 971–988.

3. Buitelaar P., Cimiano P., and Magnini B. Ontology learning from text: methods, evaluation and applications, vol. 123 of Frontiers in Artificial Intelligence and Applications, IOS, Amsterdam, 2005.
4. Christiaens S., De Leenheer P., and de Moor A. Robert Meersman R. Ontologising Competencies in an Interorganisational Setting. In Ontology Management. vol. 7 of Semantic Web and Beyond Computing for Human Experience, Springer, Berlin, 2008, pp. 265–288.
5. De Leenheer P., de Moor A., and Meersman R. Context dependency management in ontology engineering: a formal approach. J. Data Semantics, 8:26–56, 2006.
6. De Leenheer P. and Meersman R. Towards community-based evolution of knowledge-intensive systems. In Proc. OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS, 2007, pp. 989–1006.
7. de Moor A., De Leenheer P., and Meersman R. DOGMA-MESS: a meaning evolution support system for interorganizational ontology engineering. In Proc. 14th Int. Conf. on Conceptual Structures, 2006, pp. 189–203.
8. Furnas G., Landauer T., and Dumais S. The vocabulary problem in human-system communication. Commun. ACM, 30(11):964–971, 1987.
9. Gangemi A. Ontology design patterns for semantic web content. In Proc. 4th Int. Semantic Web Conf., 2005, pp. 262–276.
10. Ganter B., Stumme G., and Wille R. (eds.), Formal concept analysis, foundations and applications, LNCS, vol. 3626, Springer, Berlin, 2005.
11. Hepp M. Possible ontologies: how reality constrains the development of relevant ontologies. IEEE Internet Comput., 11(1):90–96, 2007.
12. Hepp M., De Leenheer P., de Moor A., and Sure Y. (eds.) Ontology management, semantic web, semantic web services, and business applications, vol. 7 of Semantic Web and Beyond Computing for Human Experience. Springer, Berlin, 2008.
13. Jarrar M., Demey J., and Meersman R. On reusing conceptual data modeling for ontology engineering. J. Data Semantics, 1(1):185–207, 2003.
14. Kotis K. and Vouros G. Human-centered ontology engineering: the Hcome methodology. Knowl. Inf. Syst., 10:109–131, 2005.
15. Lessig L. Ontology Management, Semantic Web, Semantic Web Services, and Business Applications. Basic Books, 1999.
16. Milton N. Knowledge Acquisition in Practice: A Step-by-Step Guide. Springer, London, 2007.
17. Pinto H., Staab S., and Tempich C. DILIGENT: towards a fine-grained methodology for DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies. In Proc. 16th European Conf. on Artificial Intelligence, 2004.
18. Ryan H., Spyns P., De Leenheer P., and Leary R. Ontology-based platform for trusted regulatory compliance services. In OTM Workshops, LNCS, vol. 2889, Springer, Berlin, 2003, pp. 675–689.
19. Siorpaes K. and Hepp M. Games with a purpose for the semantic web. IEEE Intell. Syst., 23(3):50–60, 2008.
20. Spyns P., Meersman R., and Jarrar M. Data modelling versus ontology engineering. ACM SIGMOD Rec., 31(4): 12–17, 2002.
21. Stamper R. Information in Business and Administrative Systems. Wiley, NY, 1973.
22. Van Damme C., Hepp M., and Siorpaes K. Folksontology: an integrated approach for turning folksonomies into ontologies. In Proc. ESWC Workshop Bridging the Gap between Semantic Web and Web 2.0, 2007.

Ontology Engineering

Avigdor Gal
Technion – Israel Institute of Technology,
Technion City, Haifa, Israel

Synonyms

Ontological engineering

Definition

Ontology Engineering is "the set of activities that concern the ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies" [2]. It provides "a basis of building models of all things in which computer science is interested" [4]. Ontology engineering aims at providing standard components for building knowledge models. Ontologies play a similar role to design rationale in mechanical design. It allows the reuse of knowledge in a knowledge base by providing conceptualization, reflecting assumptions and requirements made in the problem solving using the knowledge base. Ontology engineering provides the means to build and use ontologies for building models.

Key Points

Eight levels (from shallow to deep) of using ontologies can be defined [4]. At level 1, ontologies are used as a common vocabulary for communication. At level 2, it is used as a conceptual schema of a relational data base. At the third level, ontologies are used as backbone information for using a knowledge base. The remaining five levels are the levels where ontology engineering comes into play. Ontologies at the fourth level are used to answer competence questions and then they are used for standardization (of terminology or of tasks) at level 5. At level 6, ontologies are used for structural and semantic transformation of schemas. Reusing knowledge is done at the seventh level and knowledge reorganization is considered the eighth and highest level of using ontologies.

Ontology engineering makes use of ontologies (in the sense of levels 4–8) to generate standard tools for knowledge representation. This does not imply that knowledge is standardized. Using ontology engineering, one can design knowledge for specific applications, similar to production based on engineering tools in other engineering fields.

The use of ontology engineering is now illustrated in two key applications, namely functional design and schema matching. For the former, [3] describes a seventh level of using ontologies in a real world application of plant and production systems. There, an ontology that describes two types of functional models, two types of organization of generic knowledge, and two ontologies of functionality were put into use in sharing functional design knowledge on production systems. The users (engineers) of the system have indicated that this framework enabled them to make implicit knowledge possessed by each designer explicit, and to share it among team members.

Ontologies are used in schema matching in many ways. One of these, corresponding to the sixth level of use, was presented in the OntoBuilder toolcase [1]. Special ontological constructs were identified for the matching of Web form data. Ontologies were built using these constructs and dedicated matching algorithms were constructed to determine the amount of certainty to assign with the matching of attribute pairs. An example of an ontological construct, unique to OntoBuilder, is *precedence*. This construct determines the order in which attributes are presented to the user on a Web form and generate a partial order on attributes. Attribute similarity is then measured based on their relative positioning in their own ontologies.

Cross-references

▶ Ontology
▶ Semantic Matching

Recommended Reading

1. Gal A., Modica G., Jamil H., and Eyal A. Automatic ontology matching using application semantics. AI Mag., 26(1):21–32, 2005.
2. G'omez-P'erez A., Fern'andez-L'opez M., and Corcho O. Ontological Engineering. Springer, Berlin, 2003.
3. Kitamura Y., Kashiwase M., Fuse M., and Mizoguchi R. Deployment of an ontological framework of functional design knowledge. Adv. Eng. Inform., 18(2):115–127, 2004.
4. Mizoguchi R. and Ikeda M. Towards Ontology Engineering. Technical Report AI-TR-96-1, I.S.I.R., Osaka University, 1996.
5. Paslaru Bontas E. and Tempich C. Ontology Engineering: A Reality Check. In Proc. 5th Int. Conf. on Ontologies, DataBases, and Applications of Semantics, 2006, pp. 836–854.
6. Sure Y., Tempich C., and Vrandecic D. Ontology engineering methodologies. In John Davies, Rudi Studir and Paul Warren (Eds). Semantic Web Technologies: Trends and Research in Ontology-Based Systems. Wiley, UK, 2006.

Ontology Learning

▶ Ontology Elicitation

Ontology Negotiation

▶ Ontology Elicitation

Ontology Query Languages

▶ Semantic Web Query Languages

Ontology Visual Querying

Sean Bechhofer, Norman W. Paton
University of Manchester, Manchester, UK

Definition

An *ontology definition language* provides constructs that can be used to describe concepts and the relationships in which they participate. Because such languages define the properties concepts can exhibit, they can be used to restrict the questions that can meaningfully be asked about the concepts. Given a specification of the questions that can legitimately be asked, a user interface can direct query construction tasks towards meaningful requests, which in turn are expected to yield non-empty answers. Thus *ontology visual querying* is the use of an ontology to direct interactive query construction. A related topic is *faceted browsing*, in which the incremental description of concepts of interest is closely integrated with retrieval, thereby providing information about the results of a request as it is being constructed.

Historical Background

The history of visual query languages is almost as long as that of textual query languages, with Query-by-Example [14] developed in parallel with SQL. Query-by-Example contained two features that recur in almost all visual query languages: (i) a representation of the model over which the query is to be expressed; and (ii) a notation for incrementally constructing queries from the collections, relationships and value ranges of interest, with reference to the representation in (i). The evolution of visual query languages [4] has tracked the evolution of data models and interactive paradigms, and proposals have been made that support querying over many different data models (relational, object-oriented, temporal, etc) using a variety of interaction objects (forms, graphs, icons, etc). An orthogonal aspect is the closeness of the relationship between query construction and answer presentation; for example, in *dynamic queries* the answer to a request is constructed automatically and incrementally as a query is refined [12]. This provides immediate feedback to users on the size and nature of the result, but may require specialised storage structures to support incremental result computation.

Ontology visual querying has been developed so that the knowledge expressed in an ontology can be used to direct query construction; query answers may then be constructed from an instance store that is closely integrated with the ontology definition language, or by evaluating requests over external data sources. The latter is quite common, as ontologies are widely used to provide conceptual models for web (e.g., [1]) or data (e.g., [3]) resources.

Foundations

As in other visual query languages, ontology visual querying requires a visual representation of the concepts over which a request is to be constructed. Visual query formulation allows users to explore the domain of interest by *recognition* rather than *recall*: that is, it should not be necessary to remember (or even be fully aware of) the ontology in order to express a query over it.

Ontologies are represented visually other than for querying; for example, ontology design tools typically support both form and graph-based views of concepts and their relationships (e.g., [10]). As concept definition and query formulation may have significant common ground, representations that are useful for ontology browsing and concept definition may also be relevant for querying. For example, in the TAMBIS ontology-based data integration system [2], a query is a concept definition in a Description Logic (DL) [8], so writing a query is essentially the same as defining a new concept.

Although expressive ontology languages may present challenges for navigation and thus query construction, they also present certain opportunities for query interface designers, as various forms of reasoning may be useful for guiding query construction. For example, an interface can prevent the submission of queries that are unsatisfiable (i.e., that are known from the definitions in the ontology to return no results) either by making it impossible for the user to construct such queries or by detecting when such requests have been created (e.g., [7,5]). Such feedback can be seen as *intensional*, with the constraints or knowledge in the ontology determining the behaviour of the interface. Feedback may also be *extensional*, for example, with the number of results to be returned being shown to the user. This is common in facted browsing systems, and such direct result construction is generally supported in combination with closely integrated stores.

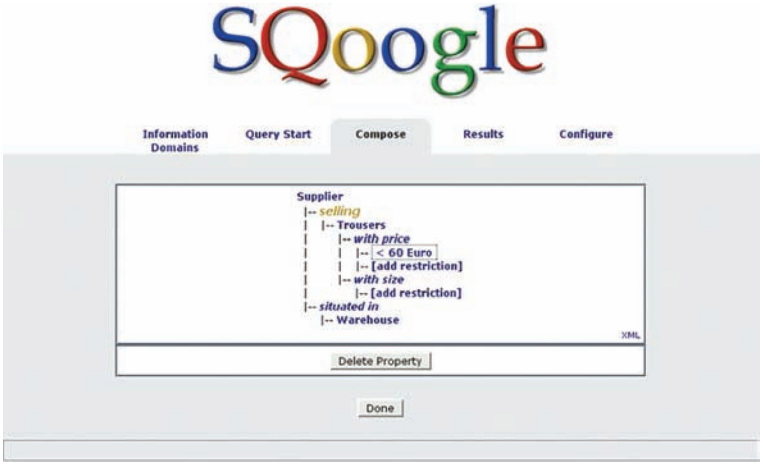
In addition to the feedback described above, systems may also provide alternative renderings of the query being constructed – for example a natural language description of the query. This can be of use in helping naïve or inexperienced users in forming appropriate queries. Note that this involves the rendering of the query in natural language, rather than translating a query posed using natural language.

A common approach is to specify queries through an iterative refinement process, in which the content of the ontology and current context of the query impact on the options presented. The principle of intensional navigation uses the vocabulary to guide the user during query formulation, employing constraints in the ontology to either flag to the user that the query is in some way violating the constraints, or preventing the user from forming queries that would be unsatisfiable, and thus return no results. Operations available for query manipulation in SEWASIE [5] include the addition of a new role/property with an associated filler, or the replacement of a filler value. In the latter case, a classification or super/sub class taxonomy is used to support the manipulation, with value fillers being specialised or generalised. Although ontology languages differ in the constructors and expressivity offered, some notion of hierarchical classification is nearly always present, and so can be exploited in visual query interfaces. An additional operation offered by the TAMBIS system is to *refocus* the query, which takes a sub node of the query and reorganises the query to promote that node to the root. This operation introduces an additional requirement on the ontology language, namely that properties or relations have *inverses*.

Various of the notions discussed above are illustrated in Figs. 1 and 2 for SQoogle, which was developed in the SEWASIE project. Figure 1 shows the composition phase, with the graphical depiction of the query. In Fig. 2, the user is being offered generalisations or specialisations of a particular node in the query.

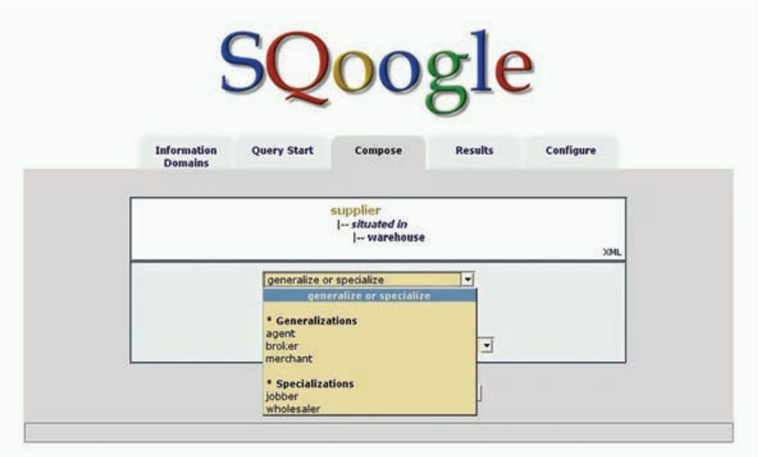
Overall, ontology visual query systems can be characterized by a number of features:

- *Identification of starting points.* The construction of a query has to start from some place in the ontology; systems may offer predetermined entry points, user defined bookmarks, or a search mechanism across the concepts in the ontology.
- *Query language.* More expressive query languages support more precise question answering, but may contain constructs that require explanation for



```
Supplier
 |-- selling
 |    |-- Trousers
 |         |-- with price
 |         |    |-- < 60 Euro
 |         |    |-- [add restriction]
 |         |-- with size
 |              |-- [add restriction]
 |-- situated in
      |-- Warehouse
```

Ontology Visual Querying. Figure 1. Visual query expression in SQoogle. The query is to select suppliers that sell trousers that cost less than 60 euros, where the supplier is situated in a warehouse.



```
supplier
 |-- situated in
      |-- warehouse
```

Ontology Visual Querying. Figure 2. Visual query refinement in SQoogle. The query can be revised by replacing the concept supplier with a more general (e.g., agent, broker) or specialized (e.g., wholesaler) concept.

users or that are challenging to represent using certain visual paradigms. Ontology visual query languages rarely support features such as aggregation or grouping.

- *Query modification operations.* Query construction involves manipulation of query expressions, for example, to include additional relationships or to specialise a concept named in the query.
- *The ontology definition language.* Richer ontology definition languages are generally more complex to display and navigate, but may provide more options for generalizing and specializing query components, and can express constraints that are useful for directing query construction.
- *Relationship between query language and ontology language.* Proposals implement different relationships between the query language and the ontology definition language. For example, in SEWASIE, the query language and ontology language are explicitly separated. The ontology language supports reasoning services that are used to guide the intensional navigation process described above. In contrast, in TAMBIS, queries *are* concept descriptions, thus the interface is tied more closely to the ontology language.
- *Feedback mechanisms.* Feedback can inform the user about the results of the query or the state of the query with respect to the underlying ontology. Feedback may be intensional (in terms of the ontology), or extensional (in terms of the result set).
- *Query presentation.* Queries may be presented solely using the visual query, or may also offer, for example, natural language renderings of the query.
- *Domain specificity.* Visual interfaces provide the opportunity to represent models or results using general-purpose or domain-specific representations. Most ontology visual query languages are general purpose, but faceted browsing interfaces are often designed to support specific applications.

Table 1 describes a collection of representative visual query systems using a selection of the above criteria: the TAMBIS and SEWASI visual query languages, and the Flamenco and /facet faceted browsing systems. In all these proposals, the ontology directs query construction, and thus the design of the ontology has a significant influence on the utility of the interface.

Key Applications

Ontology visual query systems have most commonly been deployed in areas of science and culture where there are rich data resources to be explored. For example, TAMBIS provided access to multiple biological information sources. Faceted browsing and querying has been widely used to browse image collections and in the cultural heritage domain, for example to support access to Finnish Museums [9] and galleries in the Netherlands [11]. The faceted approach is also common in on-line shopping sites such as eBay.

Ontology Visual Querying. Table 1. Representative examples of ontology visual query systems

Proposal	TAMBIS	SEWASIE	Flamenco	facet
Reference	[7]	[5]	[13]	[8]
Query language	Concept definition	Conjunctive queries	Path expressions	Path expressions
Query modification operations	Property add/remove; filler specialization or generalization; refocus	Property add/remove; filler specialization or generalization	Property add/remove; filler specialization or generalization	Property add/remove; filler specialization or generalization
Ontology definition language	Description logic	Description logic	Hierarchical categories	RDFS
Feedback mechanism	Intensional	Intensional	Extensional	Extensional
Query presentation	Visual	Visual plus NL rendering	Path expression	Path expression
Domain specificity	Generic	Generic	Specific (image repositories)	Generic

Future Directions

Large amounts of a data are beginning to emerge using representation languages like OWL. Current work in ontology languages should see standardisation of the SPARQL query language finalised in the near future. Query interfaces that sit on top these standardized languages will then be required in order to support access to this data – interfaces that support naïve or non-expert users will clearly be required.

Cross-references

▶ OWL: Web Ontology Language
▶ Visual Query Language

Recommended Reading

1. Antoniou G. and van Harmelen F. A Semantic Web Primer. MIT Press, Cambridge, MA, 2004.
2. Baader F., Calvanese D., McGuinness D., Nardi D., and Patel-Schneider P. (eds.). The Description Logic Handbook. Cambridge University Press, Cambridge, 2003.
3. Calvanese D., De Giacomo G., Lenzerini M., Nardi D. and Rosati, R. Data integration in data warehousing. Int. J. Cooperative Inf. Syst. 10(3):237–271, 2001.
4. Catarci T., Costabile M.F., Levialdi S., and Batín C. Visual query systems for databases: a survey. J. Vis. Lang. Comput. 8(2): 215–260, 1997.
5. Catarci T., Dongilli P., Di Mascio T., Franconi E., Santucci G., and Tessaris S. An ontology based visual tool for query formulation support. In Proc. 16th European Conf. on AI, 2004, pp. 308–312.
6. Colucci S., Noia T.D., Sciascio E.D., Donini F.M., Ragone A., and Rizzi R. A semantic-based fully visual application for matchmaking and query refinement in B2C e-marketplaces. In Proc. 8th ACM Int. Conf. on Electronic Commer. 2006, pp. 174–184.
7. Goble C.A., Stevens R., Ng G., Bechhofer S., Paton N.W., Baker P.G., Peim M., and Brass A. Transparent access to multiple bioinformatics information sources. IBM Syst. J., 40(2):532–551, 2001.
8. Hildebrand M., van Ossenbruggen J., and Hardman L. /facet: a browser for heterogeneous semantic web repositories. In Proc. 5th Int. Semantic Web Conf., 2006, pp. 272–285.
9. Hyvyonen E., Myakelya E., Salminen M., Valo A., Viljanen K., Saarela S., Junnila M., and Kettula S. Museum Finland – Finnish museums on the semantic web. J. Web Semantics 3(2):224–241, 2005.
10. Knublauch H., Fergerson R.W., Noy N.F., and Musen M.A. The protégé OWL plugin: an open development environment for semantic web applications. In Proc. 3rd Int. Semantic Web Conf., 2004, pp. 229–243.
11. Schreiber G., et al. MultimediaN E-culture Demonstrator. In Proc. 5th Int. Semantic Web Conf. 2006, pp. 951–958.
12. Shneiderman B. Dynamic queries for visual information seeking. IEEE Software 11(6):70–77, 1994.
13. Yee K.-P., Swearingen K., Li K., and Hearst M.A. Faceted metadata for image search and browsing. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2003, pp. 401–408.
14. Zloof M.M. Query-by-example: the invocation and definition of tables and forms. In Proc. 1st Int. Conf. on Very Large Data Bases. 1975, pp. 1–24.

On-Wire Security

▶ Storage Security

OODB (Object-Oriented Database)

▶ Object Data Models

Open Database Connectivity

Changqing Li
Duke University, Durham, NC, USA

Synonyms

ODBC

Definition

Open Database Connectivity (ODBC) [1] is an Application Programming Interface (API) specification to use database management systems (DBMS). The ODBC API is a library of functions for the ODBC-enabled applications to connect any ODBC-driver-available database, execute Structured Query Language (SQL) statements, and retrieve results. ODBC is independent of programming languages, database systems and operating systems.

Key Points

ODBC (pronounced as separate letters), is a standard database access method developed by the SQL Access Group in 1992. Its objective is to make any application to access any data regardless of the database management systems. To achieve this objective, ODBC inserts a database driver as a middle layer between an application and the DBMS, the purpose of which is to translate the application queries to commands understood by the DBMS. In practice, both the application and the DBMS must be ODBC-compliant; that is, the application must be capable of issuing ODBC commands and the DBMS must be capable of responding to them.

A procedural API is offered by the ODBC specification for using SQL queries to access data. One or more applications will be contained in an implementation of ODBC, a core ODBC library, and one or more "database drivers." Independent of the applications and DBMS, the core library acts as an "interpreter" between the applications and the database drivers, whereas the database drivers contain the DBMS-specific details. Thus applications can be written to use standard types and features without concerning the specifics of each DBMS that the applications may encounter. Similarly, database driver implementors only need to know how to attach to the core library. This makes ODBC modular.

ODBC operates with a variety of operating systems and drivers existing for relational database as well as non-relational data such as spreadsheets, text and XML files.

Cross-references

- ▶ Data Integration
- ▶ Database Adapter and Connector
- ▶ Interface
- ▶ Java Database Connectivity
- ▶ .NET Remoting
- ▶ Web 2.0/3.0
- ▶ Web Services

Recommended Reading

1. Geiger K. Inside ODBC. Microsoft Press, 1995.

Open Nested Transaction Models

Alejandro Buchmann
Darmstadt University of Technology, Darmstadt, Germany

Synonyms

Extended transaction models; advanced transaction models

Definition

Open nested transactions are hierarchically structured transactions with relaxed ACID properties. Individual subtransactions may commit independently before the complete top level transaction commits. Therefore, conventional rollback is not possible and the effects of a commited subtransaction have to be compensated if the top level transaction aborts. Depending on the particular open nested transaction model, subtransactions may be vital or non-vital and may have alternative or contingency subtransactions. Open nested transaction models are characterized through relaxed visibility rules, abort and commit dependencies.

Historical Background

Open nested transaction models evolved in the 1980s in response to two major sets of requirements: the needs of federated multidatabase systems integrating autonomous legacy database systems, and the demands of long running, cooperative processes for higher levels of concurrency while maintaining some of the main benefits of transactional processes. Extended transaction models influenced the tightly coupled transaction model of distributed object systems and the activity model of the OMG, as well as the transaction and coordination models of Web Services.

Foundations

Transaction models are characterized by the structure of its transactions, the commit and abort dependencies and the visibility rules among transactions.

Nested transactions are hierarchically structured transactions consisting of a top level transaction and subtransactions that may themselves be tree-structured. Typical of the execution model of nested transactions is the fact that higher level transactions do not execute while their subtransactions are active. The order of execution of subtransactions can be either sequential or parallel. The closed nested transaction model proposed by Moss does not specify an execution order and preserves the atomicity, consistency, isolation and durability properties of traditional flat transactions. The commit dependencies of closed nested transactions specify that all subtransactions must commit to their immediate ancestor and the top level transaction can only commit after all the subtransactions terminated. The abort dependencies specify that the whole transaction tree must be aborted if the top level transaction aborts while the abort of a subtransaction can be handled by the immediate ancestor transaction. The visibility rules among transactions and subtransactions specify that subtransactions may see changes of other subtransactions only once they are commited to the common ancestor. Changes of a nested transaction become visible to the outside world only after the top level transaction commits. Since changes of committed subtransactions are made visible only after the top level transaction commits, they can be undone through conventional roll-back.

Open nested transaction models are also based on hierarchically structured transactions. However, depending on the particular transaction model, the subtransactions may be of different types. Different types of subtransactions require different commit and/or abort dependencies. The visibility rules are also relaxed with respect to those of closed nested transaction models.

Open nested transactions may consist of the following component transactions:

- One top level transaction that has mostly a coordination function
- Vital subtransactions that must all commit for the top level transaction to be allowed to commit
- Non-vital subtransactions of which one or more may abort without causing the top level transaction to fail
- Contingency transactions are alternative subtransactions that often are executed by different autonomous systems or service providers
- Compensating transactions that must be defined to undo the changes of subtransactions that may have committed but must be undone
- Triggers that are executed as subtransactions and may execute either immediately, deferred before the commit of the triggering transaction or as detached transactions.

Open nested transaction models were defined for long running transactions and for transactions executing on federated autonomous (legacy) systems. Therefore, the degree of control of the coordinating top level transaction over the execution of the subtransactions is reduced compared to closed nested transactions running on a single database management system. Distributed short lived transactions and closed nested transactions are typically implemented through a two-phase commit protocol. In a two-phase commit protocol the first phase serves to reach agreement among the participants whether to commit or to abort, and once consensus to commit has been reached, the commit is carried out in the second phase. This protocol requires holding all the resources required by all the subtransactions through the negotiation phase until the global commit. This approach is not feasible for long running transactions because of performance reasons and in federated multidatabase systems because of the autonomy of the participating database systems.

The coordinating top level transaction in an open nested transaction model cannot secure the resources of the participating systems that execute subtransactions as autonomous individual short transactions until all participating systems have executed their subtransactions and are ready to commit. Therefore, subtransactions may commit immediately upon completion and their results are thus visible to the outside world before the top level transaction commits. Compensating transactions execute the semantically inverse operation of the committed subtransaction but do not guarantee that the exact initial state can be restored since other transactions may have executed in the interim.

The commit dependencies of open nested transaction models depend on whether they distinguish between vital and non-vital subtransactions or not. A commit dependency exists between the top level transaction and all vital subtransactions, i.e., the coordinating top level transaction may only commit if all the vital subtransactions committed. This is the default case if non-vital subtransactions are not provided. The abort of a non-vital transaction does not affect the possibility to commit the top level transaction.

The abort dependencies of open nested transaction models are the same as for closed nested transactions: the abort of the parent transaction causes the abort and undo (roll-back or compensation) of the subtransaction. The abort of a non-vital subtransaction is handled by the parent transaction, the abort of a vital transaction causes the abort of the parent and all siblings.

Contingency transactions represent alternative actions. For example, if the top-level transaction represents the booking of a trip consisting of a roundtrip flight, a hotel and a rental car booking, two different flights on different airlines may be defined as a subtransaction and a contingency transaction. Contingency transactions may only commit if the primary subtransaction aborts. Open nested transaction models that do not provide contingency transactions must implement this functionality as application-specific code in the corresponding parent transaction or the top-level transaction.

Triggers have become commonplace mechanisms in commercial relational databases and execute as subtransactions according to the semantics of closed nested transactions. The order of execution may be immediate or deferred, meaning that the trigger executes either at the point of occurrence of the triggering event or at the end of the triggering transaction, respectively. Some extended transaction models allow the triggering of subtransactions as detached or autonomous transactions, i.e., following the semantics of open nested transactions. This, however, implies all the problems resulting from the violation of the isolation and atomicity properties and requires the definition of compensating transactions for those triggers.

The concepts developed as part of extended transaction models resulted in the definition of the CORBA Activity Service Framework. The Activity Service is a general purpose event signalling mechanism that can be used to program activities to coordinate themselves according to the transaction model under consideration. The Activity Service has also been incorporated in the J2EE framework but is not widely used.

Key Applications

The main application areas for open nested transaction models are multidatabase systems in which autonomous systems are loosely coupled. Web Services with their loose coupling, distribution and often long running interactions prompted renewed interest in extended transaction models and open nested transactions. Applications based on Web Services span the whole range of e-business applications. Mobile commerce is another key application domain for open nested transactions. Proposed mobile transaction models are instances of open nested transaction models, e.g., the model proposed by Chrysanthis and extended in Kangaroo Transactions.

Future Directions

Several proposals for long running activities based on Web Services have been advanced by different consortia. The OASIS Business Transaction Protocol was proposed in 2001 by a consortium consisting of HP, BEA, and Oracle. Their model provides for the execution of business logic between the two phases of a 2 phase commit protocol. Arjuna, Oracle, Sun Microsystems, IONA Technologies and Fujitsu in 2003 founded the OASIS Web Services Composite Application Framework that defines three transaction protocols, each aimed at a specific use case. The WS-TX builds on and extends the web Services Coordination specification and provides two kinds of transaction models. These are not meant to cover all possible use cases and can be extended with the semantics of other extended transaction models as required by emerging applications.

Atomic Transactions are meant for short lived transactional interactions within trusted domains and provide full isolation (no dirty reads and repeatable reads), atomicity (well-formedness and two-phase commit protocol), and durability. For loosely coupled, long running interactions Web Services Transactions can use the Business Activity protocol, a more flexible transaction and coordination protocol that relaxes the ACID properties and draws heavily on previous research on open nested transaction models.

Business Activities are designed for long-lived transactions. They are based on the original Sagas open nested transaction model. Services are treated as Sagas and if there is the appropriate compensation behaviour defined, they can execute the undo behaviour if so instructed by the Business Activity. The responsibility of writing correct compensation services to ensure consistency rests with the developer of a service.

Business Activities consist of (possibly nested) Saga-like service invocations. Such scopes can handle errors, i.e., the abort of a task, through application logic and continue processing without globally aborting. Upon completion a child subtransaction can either leave the scope of the Business Activity or it can signal the parent that its work can be compensated later. In any event, the visibility rules are such that the results of child tasks can be seen by the outside world. Business Activities must record application state and keep a record of all sent and received messages, all request messages must be acknowledged, and requests and responses are decoupled. Two different protocols exist for Business Activities: Business Agreement With Coordinator Complete and Business Agreement With Participant Complete. The main difference between these two protocols is that in the former a participating task may not leave the scope of the Business Activity unilaterally and must wait for it to terminate. In case of abort, the subtask must compensate. In the latter a task may leave the scope of the Business Activity unilaterally.

Cross-references

► Extended Transaction Models and the ACTA Framework

► Compensating Transactions

► ConTract

► CORBA

► Distributed Database Systems

- ▶ Distributed Transaction Management
- ▶ Extended Transaction Models
- ▶ Flex transactions
- ▶ Loose Coupling
- ▶ Multilevel Transactions and Object-Model Transactions
- ▶ Nested Transaction Models
- ▶ Orchestration
- ▶ Sagas
- ▶ Transaction
- ▶ Web Transactions
- ▶ Workflow Transactions

Recommended Reading

1. Buchmann A., Özsu M.T., Hornick M., Georgakopoulos D., and Manola F. A transaction model for active distributed object systems. In Database Transaction Models for Advanced Applications, A.K. Elmagarmid (ed.). Morgan Kaufmann Publishers, Los Altos, CA, 1992.
2. Cabrera L.F., Copeland G., Feingold M. et al. Web services atomic transaction (WS-AtomicTransaction), Version 1.0, Aug. 2005. Available at: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-tx/WS-AtomicTransaction.pdf.
3. Cabrera L.F., Copeland, G., Feingold M. et al., Web services business activity framework (WS-BusinessActivity), Version 1.0, Aug. 2005. Available at: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-tx/WS-BusinessActivity.pdf.
4. Cabrera L.F., Copeland G., Feingold M. et al. Web services coordination (WS-Coordination), Version 1.0, Aug. 2005. Available at: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-tx/WS-Coordination.pdf.
5. Chrysanthis P.K. Transaction processing in a mobile environment. In Proc. IEEE Workshop on Advances in Parallel and Distributed Systems. 1993, pp. 77–82.
6. Chrysantis P. and Ramamritham K. ACTA: The saga continues. In Database Transaction Models for Advanced Applications, A.K. Elmagarmid, (ed.). Morgan Kaufmann Publishers, Los Altos, CA, 1992.
7. Dunham M.H., Helal A., and Balakrishnan S. A mobile transaction model that captures both data and movement behavior. MONET, 2(2):149–162, 1997.
8. Elmagarmid A.K. (ed.), Database Transaction Models for Advanced Applications. Morgan Kaufmann Publishers, Los Altos, CA, 1992.
9. Garcia-Molina H. and Salem K. SAGAS. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 249–259.
10. Houston I., Little M., Robinson I., Shrivastava S.K., and Wheater S.M. The CORBA activity service framework for supporting extended transactions. In Proc. IFIP/ACM Int. Conf. on Dist. Syst. Platforms, 2001, pp. 197–215.
11. Little M. A history of extended transactions. Available at: http://www.infoq.com/articles/History-of-Extended-Transactions.
12. Moss E. Nested Transactions. MIT Press, Cambridge, MA, 1985.
13. Weikum G. and Schek H.J. Concepts and applications of multilevel transactions and open nested transactions. In Database Transaction Models for Advanced Applications, A.K. Elmagarmid (ed.). Morgan Kaufmann Publishers, Los Altos, CA, 1992.

Open Nested Transactions

- ▶ Multilevel Transactions and Object-Model Transactions

Operating Characteristic

- ▶ Receiver Operating Characteristic (ROC)

Operator-Level Parallelism

Nikos Hardavellas, Ippokratis Pandis
Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

Inter-operator parallelism

Definition

Operator-level parallelism (or inter-operator parallelism) is a form of intra-query parallelism obtained by executing concurrently several operators of the same query. By contrast, intra-operator parallelism is obtained by executing the same operator on multiple processors, with each instance working on a different subset of data.

Historical Background

Parallelism has been a key focus of database research since the 1970s. For example, as early as 1978 Teradata was building highly-parallel database systems and quietly pioneered many of the ideas on parallel query execution [5]. However, the intra-query parallelism employed by these early systems was mostly intra-operator or independent parallelism (see Classes of Parallelism below). Gamma [4] was one of the first database systems that allowed operator-level parallelism through pipelining.

Foundations

Parallel processing uses multiple processors cooperatively to improve the performance of application programs. With relations growing larger and queries becoming more complex, parallel processing is an increasingly attractive option for improving the performance of database management systems. The widespread adoption of the relational database model has enabled the parallel execution of relational queries, as these queries are composed of uniform operators applied to uniform streams of data. Each operator produces a new relation, so the operators can be composed into highly parallel dataflow graphs. At the same time, multiprocessor systems and high-speed interconnection networks have become mainstream, providing an excellent basis for parallel execution.

Classes of Parallelism

Parallelism in the evaluation of database queries is classified into two main categories: inter-query parallelism (see Inter-Query Parallelism), in which different queries execute on different processors to improve the overall throughput of the system, and intra-query parallelism, in which several processors cooperate for the faster execution of a single query. Intra-query parallelism is further classified into intra-operator and inter-operator parallelism. Intra-operator parallelism (see Intra-Operator Parallelism) is obtained by executing the same operator on multiple processors, with each instance working on a different subset of data. Operator-level parallelism (or inter-operator parallelism), is obtained by executing concurrently several operators of the same query. This latter form of parallelism is the subject of this chapter.

Operator-level parallelism is in two forms: independent parallelism and pipelined parallelism. Independent parallelism (or bushy parallelism) is achieved when there is no dependency between the operators executed in parallel. For example, consider a simple query plan with two select operators and a join, that it is not nested-loops. The select operators are independent of each other and can execute concurrently, thereby exhibiting independent parallelism. Algebraically, independent parallelism can be expressed by a relation of the form f(g(X), h(Y)), where X and Y are relations and f, g, and h are relational operators. In this example, g and h exhibit independent parallelism.

Because the operators participating in bushy parallelism are independent, they do not directly affect the execution of one another. Interference is only indirect, e.g., due to the concurrent use of shared resources like disks, caches, or main memory bandwidth. Thus, independent parallelism is simpler to employ as it is easier to schedule the execution of the participating independent operators, and it has the potential to deliver high performance improvements.

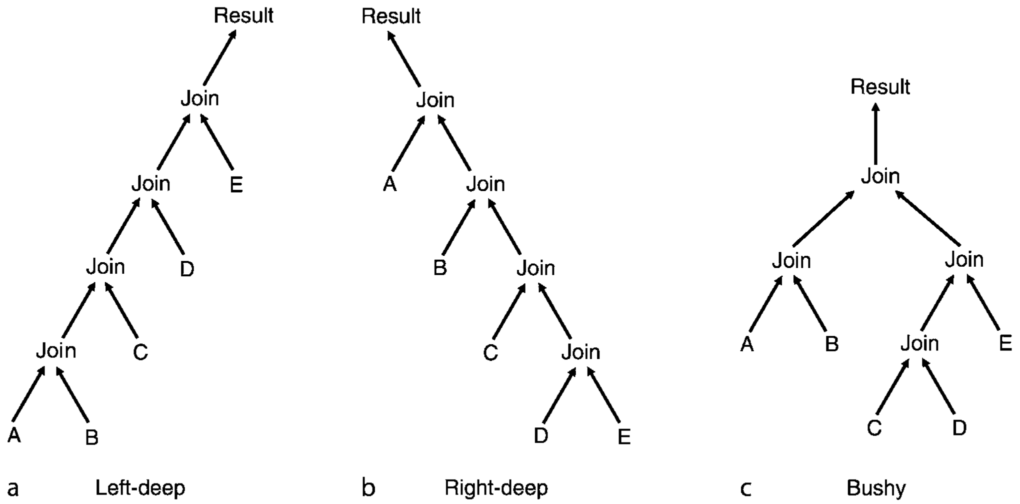
Alternatively, operator-level parallelism can take the form of pipelined parallelism, also called dataflow parallelism. Pipelined parallelism can be achieved when the concurrent operators form producer/consumer pairs in which the consumer can start executing without requiring its entire input to be available. For example, consider the aforementioned simple query that consists of two select operators and a join. The select operator can execute in parallel with the join operator. However, they are not independent, because the intermediate results produced by the select are consumed by the subsequent join. Thus, the tuples output by the select can be pipelined to the join operator to be consumed immediately. This example illustrates a significant advantage of pipelined parallelism: intermediate results are used immediately and are not materialized, saving memory and disk accesses. Algebraically, pipelined parallelism can be expressed by a relation of the form f(g(X)), where X is a relation and f and g are relational operators. The operators that cannot produce tuples unless they have processed their entire input are called Stop-&-Go operators.

Effect of Query Plan Selection on Operator-Level Parallelism

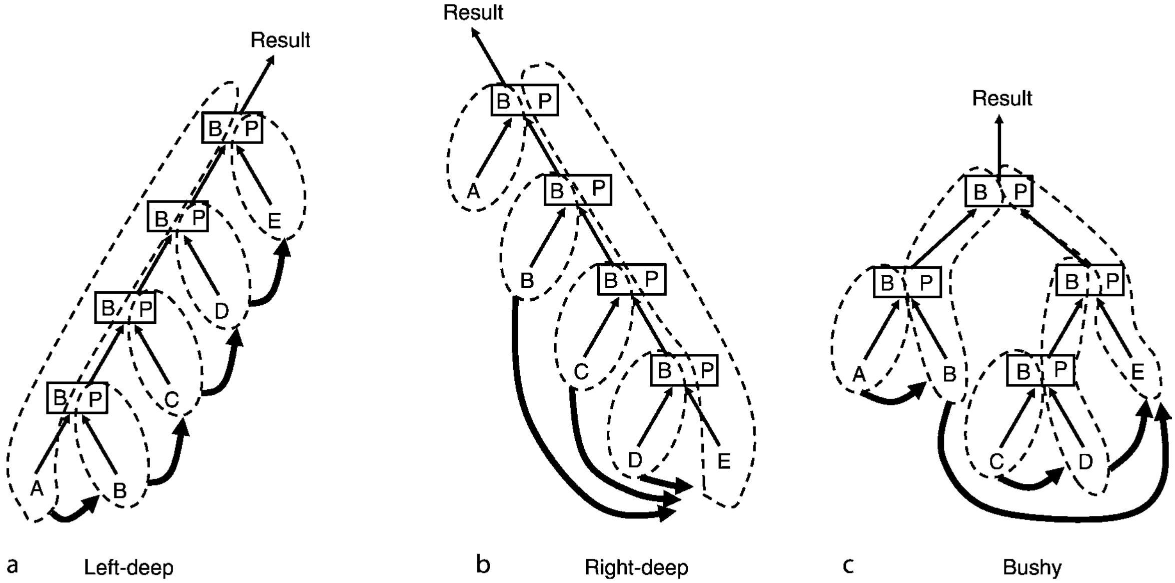
The query plan determines the execution sequence of a query's operators. The selection of a query plan greatly affects the degree of attainable operator-level parallelism. To illustrate this point, and without loss of generality, let's assume a multi-way hash-join query with four joins: A×B×C×D×E where A, B, C, D, E are relations and × is the join operator. The query plan is typically depicted graphically as a tree with vertices representing relations. Because every operator in the relational model defines a new relation, the operators in the internal vertices denote the relation they represent. If an operator Y takes relation X as one of its inputs, then a directed edge connects X to Y in the tree representation.

Three forms of query execution trees are explored in the literature: left-deep trees, right-deep trees, and bushy trees. Figure 1 shows the query execution trees for the example multi-way join query used above. Left-deep trees and right-deep trees represent the two extreme strategies of query execution, while bushy trees claim a middle ground.

To compare the trade-offs between the alternative query plans, Fig. 2 shows the execution dependencies



Operator-Level Parallelism. Figure 1. (a) Left-deep, (b) right-deep, and (c) bushy query plans.



Operator-Level Parallelism. Figure 2. Operator dependencies for (a) left-deep, (b) right-deep, and (c) bushy plans.

between the operators of each execution strategy in Fig. 1. The execution dependencies are shown using operator dependency graphs [10]. The dotted lines encircle the operators amenable to pipelined parallelism. The bold directed arcs between subgraphs show which sets of operators must be executed before other sets of operators are executed, thereby determining the maximum level of parallelism and resource requirements (e.g., memory) for the query. As discussed in [10] hash joins have two distinct phases, the build and the probe phase. Since the first phase must completely precede the second, the hash joins in Fig. 2 can be viewed as if consisting of two operators, the build and the probe operator.

The operator dependency graph of the left-deep query plan shows that only a scan, the build phase of a join, and the probe phase of a join can execute in parallel. Thus, although the left-deep query plan has low memory requirements (it needs enough memory to fit the hash tables of two joins) it offers only limited pipelined parallelism and no independent parallelism. In contrast, the operator dependency graph for the right-deep query plan shows that significant operator-level parallelism is available: all scans but one have a producer/consumer relationship with the build phase of the subsequent hash join, thereby exhibiting pipelined parallelism, while the scan/build pairs are independent of one another so they exhibit independent parallelism. However, the high degree of parallelism comes at the expense of high shared resource pressure. The hash tables for all joins should fit in main memory simultaneously, or risk spilling to disk.

Finally, the operator dependency graph for the bushy query plan has characteristics that are between the left-deep and the right-deep query plans. The bushy plan enables independent parallelism, albeit at a lower degree than the right-deep plan, as about half the scans can proceed in parallel. However, the bushy plan imposes lower pressure on shared resources than the right-deep plan, because fewer operators execute in parallel. Bushy query plans allow the formation of deeper pipelines, some of which extend all the way from a leaf to the root of the tree. Thus, bushy trees enable pipelined parallelism as well, but it may be harder to balance the load within their deeper pipelines due to execution skew.

Because bushy plans achieve a balance between pipelined parallelism, independent parallelism, and resource utilization, researchers further investigated their applicability in improving query execution. For example, segmented right-deep trees [3] (bushy trees of right-deep subtrees) have been shown to outperform their left-deep and right-deep counterparts.

Other Factors Limiting Operator-Level Parallelism

The selection of the query plan determines the degree of available operator-level parallelism. This section discusses factors that limit the effectiveness of operator-level parallelism, given a query plan. Among other things, the discussion in this section touches on issues of load balancing and processor allocation.

The operators participating in independent parallelism interfere only indirectly through the concurrent use of shared resources. The factors limiting the benefit of independent parallelism are the constraints imposed by the hardware resources. All relations that execute in parallel produce intermediate results which increase the data footprint of the application, resulting in higher cache miss rates and higher memory pressure. The larger data footprint, in turn, may oversubscribe memory bandwidth or induce more spills to disk if the relations do not fit in main memory.

Resource contention affects pipelined parallelism as well, but to a lesser degree because the intermediate data in pipelined parallelism are short lived as they are consumed immediately after their production. The benefits of pipelined parallelism are generally limited by three factors [5]: (i) relational pipelines are rarely very long – a chain of length ten is unusual. (ii) some relational operators are blocking operators, i.e., they do not emit their first output until they have consumed all their inputs. Sort and the partitioning phase of hash join are examples of blocking relational operators. Such operators cannot be pipelined, and (iii) there are dependencies between the operators participating in a pipeline. Often, the execution cost of one operator is much greater than the others, a phenomenon referred to as execution skew. In this case, the performance of the pipelined execution is dominated by the slowest operator, which significantly limits parallelism.

The execution skew also gives rise to startup/teardown execution delays: processors assigned to operators at the end of a pipeline are idle at the beginning of the computation, whereas processors assigned to operators at the beginning of a pipeline are idle towards the end of the computation. It is important to note here that data skew may induce execution skew in some cases. For example, in a sort-merge join with data skew, some sort partitions may be much larger than others, creating execution skew.

A potential solution to execution skew is to predict the execution load for each operator and schedule them accordingly across the parallel processors. However, the predictions may fail as the costs are estimated in the query optimization phase using typically inaccurate cost models and statistics.

The assignment of processors to operators and their scheduling is an important and hard problem that affects all forms of operator-level parallelism. It is an optimization problem that attempts to utilize all the available processors efficiently to minimize the execution time of a query. Sometimes operators may need to be scheduled as a team (e.g., producer/consumer pairs), while other times gang scheduling should be avoided (e.g., scheduling together the first and the last operator of a deep pipeline would leave the last operator mostly idle). Scheduling is easier when there are no dependencies between the operators executing in parallel, in which case load balancing is of primary concern.

The processor allocation is based on the selection of the query plan and estimates on the execution cost of each operator. If the execution cost of some operators is much higher than others, the system may be subject to fragmentation: after a sequence of processor allocations and releases there may be a few processors left idle and rebalancing the workload dynamically is not always possible or beneficial. These cases may benefit from the concurrent employment of multiple forms of parallelism (see next section). However, the application of multiple forms of parallelism adds an extra dimension to the processor allocation problem, making it harder to solve.

Relation to Inter-Query and Intra-Operator Parallelism

Operator-level parallelism is orthogonal to inter-query and intra-operator parallelism and can work synergistically with them to improve performance even further. For example, if there is imbalance in the execution times of a query's operators and there are free processors, intra-operator parallelism can be applied to split a long-running operator into multiple ones, each executing on a smaller subset of data. This will allow for faster execution of the expensive operators and may balance the execution times of operators participating in a pipeline, avoiding execution skew.

For a more comprehensive treatment of operator-level parallelism, the interested reader is referred to [5,10,11].

Key Applications

Several parallel database systems have been developed that utilize operator-level parallelism to improve performance. Systems built in academic institutions include GAMMA [4], BUBBA [2], Volcano [6], MonetDB/X100 [1], and StagedDB [7]. Commercial systems that support operator-level parallelism include Oracle [9] and IBM DB2 [8].

Cross-references

▶ Data Skew
▶ Execution Skew
▶ Intra-Operator Parallelism
▶ Inter-Query Parallelism
▶ Parallel Hash Join, Parallel Merge Join, Parallel Nested Loops Join
▶ Parallel Query Processing
▶ Pipelining
▶ Query Plan
▶ Stop-&-Go Operator

Recommended Reading

1. Boncz P., Zukowski M., and Nes N. MonetDB/X100: hyper-pipelining query execution. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 225–237.
2. Boral H. Prototyping bubba: a highly parallel database system. IEEE Trans. Knowl. Data Eng., 2(1), 1990.
3. Chen M.-S., Lo M., Yu P.S., and Young H.C. Using segmented right-deep trees for the execution of pipelined hash joins. In Proc. 18th Int. Conf. on Very Large Data Bases, 1992, pp. 15–26.
5. DeWitt D.J. and Gray J. Parallel database systems: the future of high-performance database computing. Commun. ACM, 35(6):85–98, 1992.
4. DeWitt D.J., Gerber R.H., Graefe G., Heytens M.L., Kumar K.B., and Muralikrishna M. GAMMA – A high performance dataflow database machine. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 228–237.
6. Graefe G. Volcano – an extensible and parallel query evaluation system. IEEE Trans. Knowl. Data Eng., 6(1):120–135, 1994.
7. Harizopoulos S. and Ailamaki A. Staged D.B.: designing database servers for modern hardware. IEEE Data Eng. Bull., 28 (2):11–16, 2005.
8. IBM Corp. DB2 Version 9 Performance Guide. Part No. SC10–4222–00, 2006.
9. Oracle Corp. Oracle Database Data Warehousing Guide. 10g Release 1 (10.1). Part No. B10736–01, 2003.
10. Schneider D.A. and DeWitt D.J. Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 469–480.
11. Yu P.S., Chen M.-S., Wolf J.L., and Turek J.J. Parallel query processing. In Advanced Database Systems, N. Adam, B. Bhargava, (eds.). LNCS, vol. 759, Springer, Berlin, 1993, pp. 239–258.

Operator Scheduling

▶ Scheduling Strategies for Data Stream Processing

Operator Tree

▶ Query Plan

Opinion Mining

Bing Liu
University of Illinois at Chicago, Chicago, IL, USA

Synonyms

Sentiment analysis

Definition

Given a set of evaluative text documents D that contain opinions (or sentiments) about an object, opinion mining aims to extract attributes and components of the object that have been commented on in each document $d \in D$ and to determine whether the comments are positive, negative or neutral.

Historical Background

Textual information in the world can be broadly classified into two main categories, *facts* and *opinions*. Facts are objective statements about entities and events in the world. Opinions are subjective statements that reflect people's sentiments or perceptions about the entities and events. Much of the existing research on text information processing has been (almost exclusively) focused on mining and retrieval of factual information, e.g., information retrieval, Web search, and many other text mining and natural language processing tasks. Little work has been done on the processing of opinions until only recently. Yet, opinions are so important that whenever one needs to make a decision one wants to hear others' opinions. This is not only true for individuals but also true for organizations.

One of the main reasons for the lack of study on opinions is that there was little opinionated text before the World Wide Web. Before the Web, when an individual needs to make a decision, he/she typically asks for opinions from friends and families. When an organization needs to find opinions of the general public about its products and services, it conducts surveys and focused groups. With the Web, especially with the explosive growth of the user generated content on the Web, the world has changed. One can post reviews of products at merchant sites and express views on almost anything in Internet forums, discussion groups, and blogs, which are collectively called the *user generated content*. Now if one wants to buy a product, it is no longer necessary to ask one's friends and families because there are plenty of product reviews on the Web that give the opinions of the existing users of the product. For a company, it may no longer need to conduct surveys, to organize focused groups or to employ external consultants in order to find consumer opinions or sentiments about its products and those of its competitors.

Finding opinion sources and monitoring them on the Web, however, can still be a formidable task because a large number of diverse sources exist on the Web and each source also contains a huge volume of information. In many cases, opinions are hidden in long forum posts and blogs. It is very difficult for a human reader to find relevant sources, extract pertinent sentences, read them, summarize them and organize them into usable forms. An automated opinion mining and summarization system is thus needed. *Opinion mining*, also known as *sentiment analysis*, grows out of this need.

Research on opinion mining started with identifying *opinion* (or *sentiment*) *bearing words*, e.g., great, amazing, wonderful, bad, and poor. Many researchers have worked on mining such words and identifying their *semantic orientations* (i.e., positive or negative). In [5], the authors identified several linguistic rules that can be exploited to identify opinion words and their orientations from a large corpus. This method has been applied, extended and improved in [3,8,12]. In [6,9], a bootstrapping approach is proposed, which uses a small set of given seed opinion words to find their synonyms and antonyms in WordNet (http://wordnet.princeton.edu/). The next major development is sentiment classification of product reviews at the document level [2,11,13]. The objective of this task is to classify each review document as expressing a positive or a negative sentiment about an object (e.g., a movie, a camera, or a car). Several researchers also studied sentence-level sentiment classification [9,14,15], i.e., classifying each sentence as expressing a positive or a negative opinion. The model of feature-based opinion mining and summarization is proposed in [6,10]. This model gives a more complete formulation of the opinion mining problem. It identifies the key pieces of information that should be mined and describes how a structured opinion summary can be produced from unstructured texts. The problem of mining opinions from comparative sentences is introduced in [4,7].

Foundations

Model of Opinion Mining

In general, opinions can be expressed on anything, e.g., a product, a service, a topic, an individual, an organization, or an event. The general term *object* is used to denote the entity that has been commented on. An object has a set of *components* (or *parts*) and a set of *attributes*. Each component may also have its subcomponents and its set of attributes, and so on. Thus, the object can be hierarchically decomposed based on the *part-of* relationship.

Definition (object): An *object O* is an entity which can be a product, topic, person, event, or organization. It is associated with a pair, (T, A), where T is a hierarchy or taxonomy of *components* (or *parts*) and *subcomponents* of O, and A is a set of *attributes* of O. Each component has its own set of sub-components and attributes.

In this hierarchy or tree, the root is the object itself. Each non-root node is a component or sub-component of the object. Each link is a part-of relationship. Each node is associated with a set of attributes. An opinion can be expressed on any node and any attribute of the node.

However, for an ordinary user, it is probably too complex to use a hierarchical representation. To simplify it, the tree is flattened. The word "*features*" is used to represent both components and attributes. Using features for objects (especially products) is quite common in practice. Note that in this definition the object itself is also a feature, which is the root of the tree.

Let an evaluative document be d, which can be a product review, a forum post or a blog that evaluates a particular object O. In the most general case, d consists of a sequence of sentences $d = \langle s_1, s_2,\ldots,s_m\rangle$.

Definition (opinion passage on a feature): The *opinion passage* on a feature f of the object O evaluated in d is a group of consecutive sentences in d that expresses a positive or negative opinion on f.

This means that it is possible that a sequence of sentences (at least one) together expresses an opinion on an object or a feature of the object. It is also possible that a single sentence expresses opinions on more than one feature, e.g., "The picture quality of this camera is good, but the battery life is short."

Definition (opinion holder): The *holder* of a particular opinion is a person or an organization that holds the opinion.

In the case of product reviews, forum postings and blogs, opinion holders are usually the authors of the posts. Opinion holders are important in news articles because they often explicitly state the person or organization that holds a particular opinion [9]. For example, the opinion holder in the sentence "John expressed his disagreement on the treaty" is "John."

Definition (semantic orientation of an opinion): The *semantic orientation* of an opinion on a feature f states whether the opinion is positive, negative or neutral.

Putting things together, a *model* for an object and a set of opinions on the features of the object can be defined, which is called the *feature-based opinion mining model*.

Model of Feature-Based Opinion Mining: An object O is represented with a finite set of features, $F = \{f_1, f_2,\ldots,f_n\}$, which includes the object itself. Each feature $f_i \in F$ can be expressed with a finite set of words or phrases W_i, which are *synonyms*. That is, there is a set of corresponding synonym sets $W = \{W_1, W_2,\ldots, W_n\}$ for the n features. In an evaluative document d which evaluates object O, an opinion holder j comments on a subset of the features $S_j \subseteq F$. For each feature $f_k \in S_j$ that opinion holder j comments on, he/she chooses a word or phrase from W_k to describe the feature, and then expresses a positive, negative or neutral opinion on f_k. The opinion mining task is to discover all these hidden pieces of information from a given evaluative document d.

Mining output: Given an evaluative document d, the mining result is a set of quadruples. Each quadruple is denoted by (H, O, f, SO), where H is the opinion holder, O is the object, f is a feature of the object and SO is the semantic orientation of the opinion expressed on feature f in a sentence of d. Neutral opinions are ignored in the output as they are not usually useful.

Given a collection of evaluative documents D containing opinions on an object, three main technical problems can be identified (clearly there are more):

Problem 1: Extracting object features that have been commented on in each document $d \in D$.

Problem 2: Determining whether the opinions on the features are positive, negative or neutral.

Problem 3: Grouping synonyms of features (as different opinion holders may use different words or phrase to express the same feature).

Opinion Summary: There are many ways to use the mining results. One simple way is to produce a *feature-based summary* of opinions on the object [6]. An example is used to illustrate what that means.

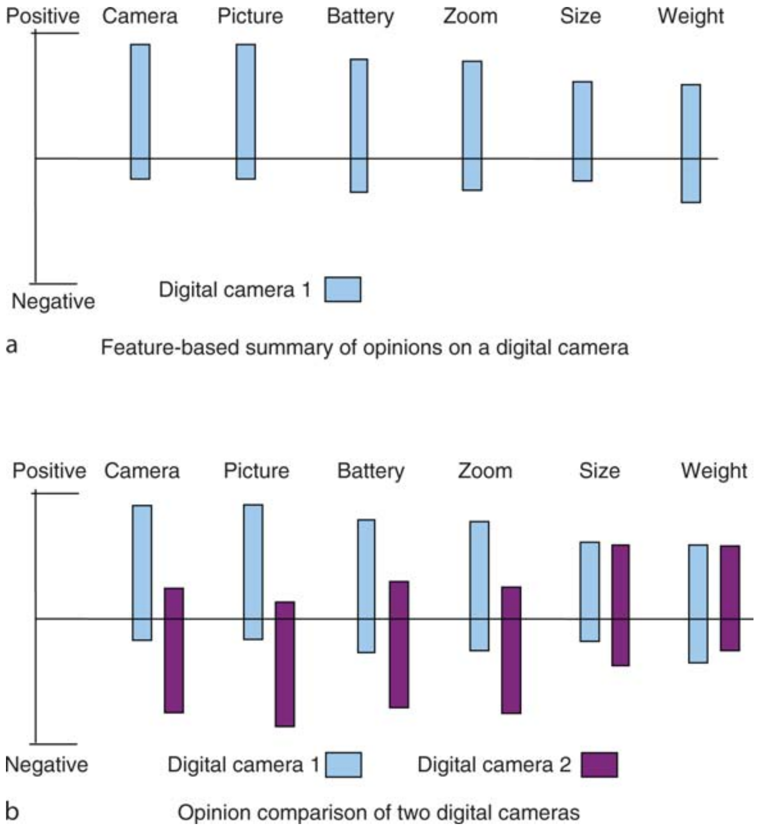
Figure 1 summarizes the opinions in a set of reviews of a particular digital camera, *digital_camera*_1. The opinion holders are omitted. In the figure, "CAMERA" represents the camera itself (the root node of the object hierarchy). One hundred and twenty-five reviews expressed positive opinions on the camera and seven reviews expressed negative opinions on the camera. "picture quality" and "size" are two product features. One hundred and twenty-three reviews expressed positive opinions on the picture quality, and only 6 reviews expressed negative opinions. The ⟨individual review sentences⟩ points to the specific sentences and/or the whole reviews that give the positive or negative comments about the feature. With such a summary, the user can easily see how existing customers feel about the digital camera. If he/she is very interested in a particular feature, he/she can drill down by following the ⟨individual review sentences⟩ link to see why existing customers like it and/or dislike it.

*Digital_camera*_1:

Camera:
    Positive: 125   <individual review sentences>
    Negative: 7   <individual review sentences>
Feature: **picture quality**
    Positive: 123   <individual review sentences>
    Negative: 6   <individual review sentences>
Feature: **size**
    Positive: 82   <individual review sentences>
    Negative: 10   <individual review sentences>
...

Opinion Mining. Figure 1. An example of a feature-based summary of opinions.

The summary in Fig. 1 can be easily visualized using a bar chart [10]. Figure 2(a) shows such a chart. In the figure, each bar above the *X*-axis gives the number of positive opinions on a feature (listed at the top), and the bar below the *X*-axis gives the number of negative opinions on the same feature. Obviously, other visualizations are also possible. For example, one may only show the percentage of positive (or negative) opinions on each feature. Comparing opinion summaries of a



Opinion Mining. Figure 2. Visualization of feature-based opinion summary and comparison.

few competing objects is even more interesting [10]. Figure 2(b) shows a visual comparison of consumer opinions on two competing digital cameras. One can clearly see how consumers view different features of each camera.

Sentiment Classification

Sentiment classification has been widely studied in the natural language processing (NLP) community [e.g., 2,11,13]. It is defined as follows: Given a set of evaluative documents D, it determines whether each document $d \in D$ expresses a positive or negative opinion (or sentiment) on an object. For example, given a set of movie reviews, the system classifies them into positive reviews and negative reviews.

This is clearly a classification learning problem. It is similar but also different from the classic topic-based text classification, which classifies documents into predefined topic classes, e.g., politics, sciences, and sports. In topic-based classification, topic related words are important. However, in sentiment classification, topic-related words are unimportant. Instead, opinion words that indicate positive or negative opinions are important, e.g., great, excellent, amazing, horrible, bad, worst, etc. There are many existing techniques. Most of them apply some forms of machine learning techniques for classification (e.g., [11]). Custom-designed algorithms specifically for sentiment classification also exist, which exploit opinion words and phrases together with some scoring functions [2,13].

This classification is said to be at the document level as it treats each document as the basic information unit. Sentiment classification thus makes the following assumption: Each evaluative document (e.g., a review) focuses on a single object O and contains opinions of a single opinion holder. Since in the above opinion mining model an object O itself is also a feature (the root node of the object hierarchy), sentiment classification basically determines the semantic orientation of the opinion expressed on O in each evaluative document that satisfies the above assumption.

Apart from the document-level sentiment classification, researchers have also studied classification at the *sentence-level*, i.e., classifying each sentence as a subjective or objective sentence and/or as expressing a positive or negative opinion [9,14,15]. Like the document-level classification, the sentence-level sentiment classification does not consider object features that have been commented on in a sentence. Compound sentences are also an issue. Such a sentence often express more than one opinion, e.g., "The picture quality of this camera is amazing and so is the battery life, but the viewfinder is too small."

Feature-Based Opinion Mining

Classifying evaluative texts at the document level or the sentence level does not tell what the opinion holder likes and dislikes. A positive document on an object does not mean that the opinion holder has positive opinions on all aspects or features of the object. Likewise, a negative document does not mean that the opinion holder dislikes everything about the object. In an evaluative document (e.g., a product review), the opinion holder typically writes both positive and negative aspects of the object, although the general sentiment on the object may be positive or negative. To obtain such detailed aspects, going to the feature level is needed. Based on the model presented earlier, three key mining tasks are:

1. Identifying *object features*: For instance, in the sentence "The picture quality of this camera is amazing," the object feature is "picture quality." In [10], a supervised pattern mining method is proposed. In [6,12], an unsupervised method is used. The technique basically finds frequent nouns and noun phrases as features, which are usually genuine features. Clearly, many information extraction techniques are also applicable, e.g., conditional random fields (CRF), hidden Markov models (HMM), and many others.
2. Determining opinion orientations: This task determines whether the opinions on the features are positive, negative or neutral. In the above sentence, the opinion on the feature "picture quality" is positive. Again, many approaches are possible. A lexicon-based approach has been shown to perform quite well in [3,6]. The lexicon-based approach basically uses opinion words and phrases in a sentence to determine the orientation of an opinion on a feature. A relaxation labeling based approach is given in [12]. Clearly, various types of supervised learning are possible approaches as well.
3. Grouping synonyms: As the same object features can be expressed with different words or phrases, this task groups those synonyms together. Not much research has been done on this topic. See [1] for an attempt on this problem.

Mining Comparative and Superlative Sentences

Directly expressing positive or negative opinions on an object or its features is only one form of evaluation. Comparing the object with some other similar objects is another. Comparisons are related to but are also different from direct opinions. For example, a typical opinion sentence is "The picture quality of camera x is great." A typical comparison sentence is "The picture quality of camera x is better than that of camera y." In general, a comparative sentence expresses a relation based on similarities or differences of more than one object. In English, comparisons are usually conveyed using the *comparative* or the *superlative* forms of adjectives or adverbs. The structure of a comparative normally consists of the stem of an adjective or adverb, plus the suffix -*er*, or the modifier "more" or "less" before the adjective or adverb. The structure of a superlative normally consists of the stem of an adjective or adverb, plus the suffix -*est*, or the modifier "most" or "least" before the adjective or adverb. Mining of comparative sentences basically consists of identifying what features and objects are compared and which objected are preferred by their authors (opinion holders). Details can be found in [4,7].

Key Applications

Opinions are so important that whenever one needs to make a decision, one wants to hear others' opinions. This is true for both individuals and organizations. The technology of opinion mining thus has a tremendous scope for practical applications.

Individual consumers: If an individual wants to purchase a product, it is useful to see a summary of opinions of existing users so that he/she can make an informed decision. This is better than reading a large number of reviews to form a mental picture of the strengths and weaknesses of the product. He/she can also compare the summaries of opinions of competing products, which is even more useful.

Organizations and businesses: Opinion mining is equally, if not even more, important to businesses and organizations. For example, it is critical for a product manufacturer to know how consumers perceive its products and those of its competitors. This information is not only useful for marketing and product benchmarking but also useful for product design and product developments.

Cross-references

▶ Text Mining

Recommended Reading

1. Carenini G., Ng R., and Zwart E. Extracting Knowledge from Evaluative Text. In Proc. 3rd Int. Conf. on Knowledge Capture, 2005.
2. Dave D., Lawrence A., and Pennock D. Mining the peanut gallery: opinion extraction and semantic classification of product reviews. In Proc. 12th Int. World Wide Web Conference, 2003.
3. Ding X., Liu B., and Yu P. A holistic lexicon-based approach to opinion mining. In Proc. 1st ACM Int. Conf. on Web Search and Data Mining. 2008.
4. Ganapathibhotla G. and Liu B. Identifying preferred entities in comparative sentences. In Proc. 22nd Int. Conf. on Computational Linguistics. 2008.
5. Hatzivassiloglou V. and McKeown K. Predicting the semantic orientation of adjectives. In Proc. 8th Conf. European Chapter of Assoc. Comp. Linguistics. 1997.
6. Hu M. and Liu B. Mining and summarizing customer reviews. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004.
7. Jindal N. and Liu B. Mining comparative sentences and relations. In Proc. of National Conf. on Artificial Intelligence, 2006.
8. Kanayama H. and Nasukawa T. Fully automatic lexicon expansion for domain-oriented sentiment analysis. In Proc. 2006 Conf. on Empirical Methods in Natural Language Processing, 2006.
9. Kim S. and Hovy E. Determining the sentiment of opinions. In Proc. 20th Int. Conf. on Computational Linguistics, 2004.
10. Liu B., Hu M., and Cheng J. Opinion observer: analyzing and comparing opinions on the web. In Proc. 14th Int. World Wide Web Conference, 2005.
11. Pang B., Lee L., and Vaithyanathan S. Thumbs up? Sentiment classification using machine learning techniques. In Proc. 2002 Conf. on Empirical Methods in Natural Language Processing, 2002.
12. Popescu A.-M. and Etzioni O. Extracting product features and opinions from reviews. In Proc. 2005 Conf. on Empirical Methods in Natural Language Processing, 2005.
13. Turney P. Thumbs up or thumbs down? Semantic Orientation Applied to Unsupervised Classification of Reviews. In Proc. 40th Annual Mfg. of Assoc. Comp. Linguistics, 2002.
14. Wiebe J. and Riloff E. Creating subjective and objective sentence classifiers from unannotated texts. In Proc. Int. Conf. on Intelligent Text Processing and Computational Linguistics, 2005.
15. Wilson T., Wiebe J., and Hwa R. Just how mad are you? Finding strong and weak opinion clauses. In Proc. National Conf. on Artificial Intelligence, 2004.

Optical Storage

▶ Storage Devices

Optimistic Replication

▶ Optimistic Replication and Resolution

Optimistic Replication and Resolution

Marc Shapiro
INRIA Paris-Rocquencourt and LIP6, Paris, France

Synonyms

Optimistic Replication; Reconciliation-Based Data Replication; Lazy Replication; Multi-Master System

The term "Optimistic Replication" is prevalent in the distributed systems and distributed algorithms literature. The database literature prefers "Lazy Replication"

Definition

Data replication places physical copies of a shared logical item onto different sites. *Optimistic replication* (OR) [11] allows a program at some site to read or update the local replica at any time. An update is *tentative* because it may conflict with a remote update. Such conflicts are resolved after the fact, in the background. Replicas may *diverge* occasionally but are expected to converge eventually (see entry on EVENTUAL CONSISTENCY).

OR avoids the need for distributed concurrency control prior to using an item. It allows a site to execute even when remote sites have crashed, when network connectivity is poor or expensive, or while disconnected from the network. *Disconnected operation,* the capability to compute while disconnected from a data source, e.g., in mobile computing, requires OR. In *computer-supported co-operative work*, OR enables a user to temporarily insulate himself from other users.

The defining characteristic of OR is that any synchronization between sites occurs in the background, after local termination, i.e., off the critical path of the application.

Historical Background

The first historical instance of OR is Johnson's and Thomas's replicated database (1976). (The vocabulary used in this history is defined in Section "Foundations.")

Usenet News (1979) was an important and inspirational development. News supports a large-scale ever-growing database of (read-only) items, posted by users all over the world. A Usenet site connects infrequently (e.g., daily) with its peers. New items are flooded to other sites, and are received in arbitrary order. Users occasionally observe ordering anomalies, but this is not considered a problem. However, system administrators must deal manually with conflicts over administrative operations.

In 1984, Wuu and Bernstein's replicated mutable key-value-pair database use an operation log, transmitted by an *anti-entropy* protocol: site A sends to site B only the tail of A's log that B has not yet seen [15]. Concurrent operations either commute or have a natural semantic order; non-concurrent operations execute in happens-before order.

The Lotus Notes system (1988) supports co-operative work between mobile enterprise users. It replicates a database of discrete items in a peer-to-peer manner. Notes is state-based, and uses a Last-Writer Wins policy. A deleted item is replaced by a *tombstone.*

Several file systems, designed in the early 1990s to support disconnected work, e.g., Coda [6], are state based and uses version vectors for conflict detection. Conflicts over some specific object types (e.g., directories or mailboxes) cause automatic resolver programs to run. The others must be resolved manually.

Golding (1992) [3] studies a replicated database of mutable key-value pairs. This system purges an operation from the log when it can prove that it was delivered to all sites. Consistency is ensured by defining a total order of operations.

Bayou (1994–1997) is an innovative general-purpose database for mobile users [9]. Bayou is operation-based and uses an anti-entropy protocol. Each site executes transactions in arbitrary order; transactions remain tentative. The eventual serialization order is the order of execution at a designated *primary* site. Other sites roll back their tentative state, and re-execute committed transactions in commit order.

In 1996, Gray et al. argued that OR databases for disconnected work cannot scale [4], because conflict reconciliation is expensive, conflict probability rises as the third power of the number of nodes, and the wait probability further increases quadratically with disconnection time.

Breitbart et al. [1] describe a partially-replicated database that uses a form of OR. Each item has a designated primary site and may be replicated at any number of secondary sites. A read may occur at a secondary site but a write must occur on the primary. It follows that write transactions update a single site. If transactions are serialisable at each site, and update propagation is restricted to avoid ordering anomalies, then transactions are serialisable despite lazy propagation.

The Computer-Supported Cooperative Work (CSCW) community invented (1989) a form of OR called Operational Transformation (OT). Conflicting operations are *transformed*, by modifying their arguments, in order to execute in arbitrary order [12].

Foundations

Figure 1 depicts a logical item *x*, concretely replicated at three different sites. In OR, any site may *submit* or *initiate* a transaction reading or writing the local replica. If the transaction succeeds locally, the system *propagates* it to other sites, and *replays* the transaction on the remote sites, in a *lazy* manner, in the background. Local execution is *tentative* and may be rolled back later, because of a *conflict* with a concurrent remote transaction. (The happens-before and concurrency relations are defined formally by Lamport [7]. Transaction A happens-before B, if B was initiated on some site after A executed at that site. Two transactions are concurrent if neither happens-before the other.)

OR is opposed to pessimistic (or eager) replication, where a local transaction terminates only when it commits globally. Pessimistic replication establishes a total order for committed transactions, at the latest when each transaction terminates. In contrast, OR generally relaxes the ordering requirements and/or converges to a common order *a posteriori*. The effects of a tentative transaction can be observed, thus OR protocols may violate the *isolation* property and allow cascading aborts and retries to occur.

Transmitting and Replaying Updates

In OR, updates are propagated lazily, in the background, after the transaction has terminated locally. Transmission usually uses peer-to-peer epidemic or anti-entropy techniques (see entry on Peer-to-Peer Content Distribution).

A site that receives a remote update *replays* it, i.e., incorporates it into the local replica. There are two main approaches. In the *state-based* approach, the initiator site transmits the after-values of the transaction, and other sites assign the after-value to their local replica. In the *operation-based* approach, the initiator sends the program of the transaction itself, and other sites re-execute the transaction.

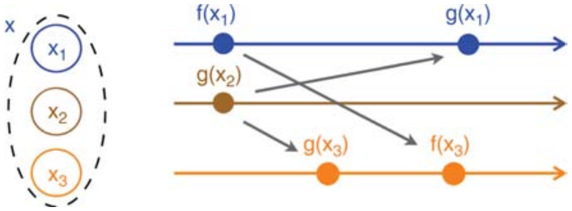
State-based replay is guaranteed to be deterministic. State-based replay can be more efficient, since the replay code is just a write. On the downside, if the granularity is large, then state-based transmission is expensive and replay is subject to false conflicts. Furthermore, logical operations are more likely to commute than writes, thus operation-based replay typically causes fewer aborts.

Conflicts

Each transaction taken individually is assumed correct (the C of the ACID properties), i.e., it maintains semantic invariants. For example, ensuring that a bank account remains positive, or that a person is not scheduled in two different meetings at the same time.

As is clear from Fig. 1, concurrent transactions may be delivered to different sites in different orders. (Dependent transactions are assumed to execute in dependency order; see Section "Scheduling Transactions Content and Ordering.") However, consistency requires that local schedules be equivalent. In this respect, one may classify pairs of concurrent transactions as commuting, non-commuting, and antagonistic. Transactions *conflict* if they are mutually non-commuting or mutually antagonistic.

The relative execution order of *commuting* transactions is immaterial; they require no remote synchronization. Formally, two transactions T_1 and T_2 commute if execution order $T_1;T_2$ returns the same results to the



Optimistic Replication and Resolution. Figure 1. Three sites with replicas of logical item *x*. Site 1 initiates transaction *f*, Site 2 initiates *g*. The system propagates and replays on remote sites. Site 3 executes in the order *g;f*, whereas Site 1 replays *f* before *g*. Eventually, Site 2 will also execute *f*.

user and leaves the database in the same state as the order $T_2;T_1$. For instance, depositing €10 in a bank account commutes with a depositing €20 into the same account, and also commutes with withdrawing €100 from an independent account.

If running concurrent transactions together would violate an invariant, they are said *antagonistic*. Safety requires aborting one or the other (or both). For instance, if T_1 schedules me in a meeting from 10:00 to 12:00, and T_2 schedules a meeting from 11:00 to 13:00, they are antagonistic since no combination of both T_1 and T_2 can be correct.

If two transactions are *non-commuting* and neither is aborted, then their relative execution order must be the same at all sites. Consider for instance T_1 = "transfer balance to savings" and T_2 = "deposit €100." Both orders $T_1;T_2$ and $T_2;T_1$ make sense, but the result is clearly different. There must be a system-wide consensus on the order chosen.

Conflict Resolution and Reconciliation

Conflict resolution rewrites or aborts transactions to remove conflicts. Conflict resolution can be either manual or automatic. Manual conflict resolution simply allows conflicting transactions to proceed, thereby creating conflicting versions; it is up to the user to create a new, merged version.

Reconciliation detects and repairs conflicts, and combines non-conflicting updates. Thus transactions are *tentative*, i.e., a tentatively-successful transaction may have to roll back for reconciliation purposes. OR resolves conflicts *a posteriori* (whereas pessimistic approaches avoid them *a priori*).

In many systems, data invariants are either unknown or not communicated to the system. In this case, the system designer conservatively assumes that, if concurrent transactions access the same item, and one (or both) writes the item, then they are antagonistic. Then, one of them must abort, or both.

A few systems, such as Bayou [14] or IceCube [10] support an application-specific check of invariants.

Last Writer Wins

When transactions consist only of writes, a common approach is to ensure a global precedence order.

For instance, many replicated file systems follow the "Last Writer Wins" (LWW) approach. Files have timestamps that increase with successive versions. When the file system encounters two concurrent versions of the same file, it overwrites the one with the smallest timestamp with the "younger" one (highest timestamp). The write with the smallest timestamp is lost; this approach violates the Durability property of ACID.

Semantic Resolvers

A resolver is an application-specific conflict resolution program that automatically merges two conflicting versions of an item into a new one. For example, the Amazon online book store resolves problems with a user's "shopping cart" by taking the union of any concurrent instances. This maximizes availability despite network outages, crashes, and the user opening multiple sessions.

A resolver should ensure that the conflicting transactions are made to commute. In a state-based approach, a resolver generally parses the item's state into small, independent sub-items. Then it applies a LWW policy to updated and tombstoned sub-items, and a union policy to newly-created sub-items.

The most elaborate example exists in Bayou. A Bayou transaction has three components: the dependency check, the write, and the merge procedure. The former is a database query that checks for conflicts when replaying. The write (a SQL update) executes only if the consistency check succeeds. If it fails, the merge procedure (an arbitrary but deterministic program) provides a chance to fix the conflict. However, it is very difficult to write merge procedures in the general case.

Operational Transformation

In Operational Transformation (OT), conflicting operations are *transformed* [12]. Consider two users editing the shared text "abc". User 1 initiates *insert ("X",2)* resulting in "aXbc" and User 2 initiates *delete (3)*, resulting in "ab." When User 2 replays the insert, the result is "aXb" as expected. However for User 1 to observe the same result, the delete must be transformed to *delete(2)*.

In essence, the operations were specified in a non-commuting way, but transformation makes them commute. OT assumes that transformation is always possible. The OT literature focuses on a simple, linear, shared edit buffer data type, for which numerous transformation algorithms have been proposed.

OT requires two correctness conditions, often called TP1 and TP2. TP1 requires that, for any two concurrent operations A and B, running "A followed by {B transformed in the context of A}" yield the same result as "B followed by {A transformed in the context of B}." TP1 is relatively easy to satisfy, and is sufficient if replay is somehow serialized.

TP2 requires that transformation functions themselves commute. TP2 is necessary if replay is in arbitrary order, e.g., in a peer-to-peer system. The vast majority of published non-serialized OT algorithms have been shown to violate TP2 [8].

Scheduling Transactions Content and Ordering

In order to capture any causal dependencies, transactions execute in happens-before order. As explained in Section "Conflicts," antagonistic transactions cause aborts, and non-commuting transactions must be mutually ordered. This so-called *serialization* requires a consensus.

Whereas pessimistic approaches serialize *a priori*, most OR systems execute transactions tentatively in arbitrary order and serialize a posteriori. Some executions are rolled back; cascading aborts may occur.

A prime example is the Bayou system [14]. Each site executes transactions in the order received. Eventually, the transactions reach a distinguished *primary* site. If a transaction fails its dependency check at the primary, then it aborts everywhere. Transactions that succeed commit, and are serialized in the execution order of the primary.

The IceCube system showed that it is possible to improve the user experience by scheduling operations intelligently [10]. IceCube is a middleware that relieves the application programmer from many of the complexities of reconciliation. Multiple applications may co-exist on top of IceCube. Applications expose semantic annotations, indicating which operation pairs commute or not, are antagonistic, dependent, or have an inherent semantic order. The user may create atomic groups of operations from different applications. The IceCube scheduler performs an optimization procedure over a batch of operations, minimizing the number of aborted operations. The user commits any of the alternative schedules proposed by the system.

Freshness of Replicas

Applications may benefit from *freshness* or quality-of-service guarantees, e.g., that no replica diverges by more than a known amount from the ideal, strongly-consistent state. Such guarantees come at the expense of decreased availability.

The Bayou system proposes qualitative "session guarantees" on the relative ordering of operations [13]. For instance, Read-Your-Writes (RYW) guarantees that a read observes the effect of a write by the same user, even if initiated at a different site. RYW ensures, that immediately after changing his password, a user can log in with the new password. Other similar guarantees are Monotonic-Reads, Writes-Follow-Reads, and Monotonic-Writes.

Systems such as TACT control replica divergence quantitatively [5]. TACT provides a time-based guarantee, allowing an item to remain stale for only a bounded amount of time. TACT implements this by pushing an update operation to remote replicas before the time limit elapses. TACT also provides "order bounding," i.e., limiting the number of uncommitted operations: when a site reaches a user-defined bound on the number of uncommitted operations, it stops accepting new ones.

Finally, TACT can bound the difference between numeric values.

For this, each replica is allocated a quota. Each site estimates the progress of other sites, using vector clock techniques. The site stops initiating operations once its cumulative modifications, or the estimated remote updates to the item, reach the quota. At that point the site pushes its updates and pulls remote operations. For example a bank account might be replicated at ten sites.

To guarantee that the balance observed is within €50 of the truth, each site's quota is €50/10 = €5. Whenever the difference estimated by a site reaches €5, it synchronises with the others.

Optimistic Replication Versus Optimistic Concurrency Control

The word "optimistic" has different, but related, meanings when used in the context of replication and of concurrency control.

Optimistic replication (OR) means that updates propagate lazily. There is no *a priori* total order of transactions. There is no point in time where different sites are guaranteed to have the same (or equivalent) state. Cascading aborts are possible.

Optimistic concurrency control (OCC) means that conflicting transactions are allowed to proceed concurrently. However, in most OCC implementations, a transaction validates before terminating. A transaction is serialized with respect to concurrent transactions, at the latest when it terminates, and cascading aborts do not occur.

Key Applications

Usenet News pioneered the OR concept, allowing to share write-only information over a slow, but cheap network using dial-up modems over telephone lines.

Mobile users want to be able to work as usual, even when disconnected from the network. Thus, mobile computing is a key driver for OR applications. Systems designed for disconnected work that use OR include the Coda file system [6], the Bayou shared database [14], or the Lotus Notes collaborative suite.

Another important application area is Computer-Supported Collaborative Work. In this domain, users must be able to update shared artefacts in complex ways without interfering with one another. OR allows a user to insulate himself temporarily from other users. A key example is the Concurrent Versioning System (CVS), which enables collaborative authoring of computer programs [2]. Bayou and Lotus Notes, just cited, are also designed for collaborative work.

OR is used for high performance and high availability in large-scale web sites. A recent example is Amazon's "shopping cart," which is designed to be highly available, even if the same user connects to several instances of the Amazon store discussed earlier.

Cross-references

- ▶ Eventual Consistency
- ▶ Peer-to-Peer Content Distribution
- ▶ Peer-to-Peer System
- ▶ Strong Consistency Models for Replicated Data
- ▶ Traditional Concurrency Control for Replicated Databases
- ▶ WAN Data Replication

Recommended Reading

1. Breitbart Y., Komondoor R., Rastogi R., and Seshadril S. Update propagation protocols for replicated databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 97–108.
2. Cederqvist P. et al. Version Management with CVS. Network Theory, Bristol, 2006.
3. Golding R.A. Weak-Consistency Group Communication and Membership. Ph.D. thesis, University of California, Santa Cruz, CA, USA, 1992, tech. Report no. UCSC-CRL-92-52. Available at ftp://ftp.cse.ucsc.edu/pub/tr/ucsc-crl-92-52.ps.Z
4. Gray J., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
5. Haifeng Yu and Amin Vahdat. Combining Generality and Practicality in a Conit-Based Continuous Consistency Model for Wide-Area Replication. In Proc. 21st Int. Conf. on Distributed Computing Systems, USA.
6. Kistler J.J. and Satyanarayanan M. Disconnected operation in the Coda file system. ACM Trans. Comp. Syst., 10(5):3–25, 1992.
7. Lamport L. Time, clocks, and the ordering of events in a distributed system. Commun. ACM, 21(7): 558–565, 1978.
8. Oster G., Urso P., Molli P., and Imine A. Proving correctness of transformation functions in collaborative editing systems. Rapport de recherche RR-5795, LORIA – INRIA Lorraine, 2005, Available at http://hal.inria.fr/inria-00071213/.
9. Petersen K., Spreitzer M.J., Terry D.B., Theimer M.M., and Demers A.J. Flexible update propagation for weakly consistent replication. In Proc. 16th ACM Symp. on Operating System Principles, 1997, pp. 288–301.
10. Preguiça N., Shapiro M., and Matheson C. Semantics-based reconciliation for collaborative and mobile environments. In Proc. Int. Conf. on Cooperative Inf. Syst., 2003, pp. 38–55.
11. Saito Y. and Shapiro M. Optimistic replication. ACM Comput. Surv., 37(1):42–81, 2005.
12. Sun C. and Ellis C. Operational transformation in real-time group editors: issues, algorithms, and achievements. In Proc. Int. Conf. on Computer-Supported Cooperative Work, 1998, p. 59.
13. Terry D.B., Demers A.J., Petersen K., Spreitzer M.J., Theimer M.M., and Welch B.B. Session guarantees for weakly consistent replicated data. In Proc. Int. Conf. on Parallel and Distributed Information Systems, 1994, pp. 140–149.
14. Terry D.B., Theimer M.M., Petersen K., Demers A.J., Spreitzer M.J., and Hauser C.H. Managing update conficts in Bayou, a weakly connected replicated storage system. In Proc. 15th ACM Symp. on Operating System Principles, 1995, pp. 172–182.
15. Wuu G.T.J. and Bernstein A.J. Efficient solutions to the replicated log and dictionary problems. In Proc. ACM SIGACT-SIGOPS 3rd Symp. on the Principles of Dist. Comp., 1984, pp. 233–242.

Optimization and Tuning in Data Warehouses

Ladjel Bellatreche
LISI/ENSMA–Poitiers University, Futuroscope Cedex, France

Definition

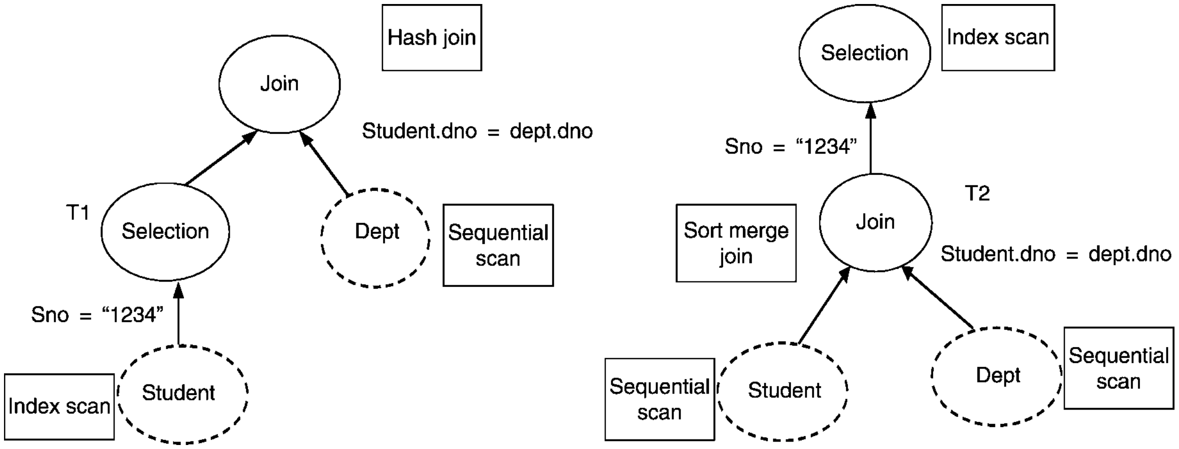
Optimization and tuning in data warehouses are the processes of selecting adequate optimization techniques in order to make queries and updates run faster and to maintain their performance by maximizing the use of data warehouse system resources. A data warehouse is usually accessed by complex queries for key business operations. They must be completed in seconds not days. To continuously improve query performance, two main phases are required: physical design and tuning. In the first phase, data warehouse administrator selects optimization techniques such as materialized views, advanced index schemes, denormalization, vertical partitioning, horizontal partitioning and parallel processing. Generally, this selection is based on most frequently asked queries and typical updates. Physical design generates a configuration Δ containing a number of optimization techniques. This configuration should evolve, since data warehouse dynamically changes during its lifetime. These changes necessitate a tuning phase so as to keep the performance of the warehouse from degrading. Changes may be related to the content of tables, sizes of optimization structures selected during physical design (materialized views, indexes and partitions), frequencies of queries/updates, addition/deletion of queries/updates, etc. The role of the tuning phase is to monitor and to diagnose the use of configuration Δ and different resources assigned to Δ (like buffer, storage space, etc.). For instance, if an optimization technique, like an index, is not used by the whole workload, it will be dropped by a tuning tool and might be replaced with another technique.

Historical Background

There has been extensive work in query optimization since the early 1970s in traditional databases. Several algorithms and systems have been proposed, such as *System-R project*, and its ideas have been largely incorporated in many commercial optimizers. Note that each SQL query corresponding to a select-project-join query in the relational algebra may be represented by many query trees. The leaves of each query tree represent base relations and non-leaf nodes are algebraic operators like selections, projections, unions, joins. An intermediate node indicates the application of the corresponding operator on the relations generated by its children, the result of which is then sent further up. Thus, the edges of a tree represent data rows from bottom to top, i.e., from the leaves which correspond to data in the database, to the root, which is the final operator producing the query answer. For a complicated query, the number of all possible query trees may be very high due to many algebraic laws that hold for relational algebra: commutative and associative laws of joins, laws involving selection and projection push down along the tree, etc.

To choose an optimal query tree, a database optimizer may employ one of two optimization techniques: *rule-based optimization approach* and *cost-based optimization approach*. The rule-based optimizer is the oldest one. It is simple since it is based on a set of rules concerning join algorithms, the use of an index or not, the choice of the external relation in a nested loop, and so on. The optimizer chooses an execution plan based on the available access paths and their ranks. For instance, Oracle's ranking of the access paths is heuristic. If there is more than one way to execute a query, then the rule-based optimizer always uses the operation with the lower rank. Usually, operations of lower rank execute faster than those associated with constructs of higher rank. In cost-based optimization, the optimizer estimates the cost of each possible execution plan (Query execution plan is a set of steps used to access data in relational databases. Figure 1 gives an example of an execution plan, where dashed circles and solid circles represent base tables and algebraic operations, respectively. Other execution plans might be generated.) by applying heuristic formulas using a set of statistics concerning database (sizes of tables, indexes, tuple length, selectivity factors of join and selection predicates, sizes of intermediate results, etc.) and hardware (size of buffer, page size, etc.). For each execution plan, the query optimizer performs the following tasks: (i) it selects an order and grouping for associative-and-commutative operations like joins, unions and intersection, (ii) it chooses implementation algorithms for different algebraic operations: for example, selections may be implemented using either a sequential scan or an index scan; join operation may be implemented in different ways: *nested loop*, *sort-merge join* and *hash join* (see Fig. 1), (iii) it manages additional operators like group-by, sorting, etc., and (iv) it manages the intermediate results ($T1$ and $T2$ are an example of intermediate results of two execution plans of the same query in Fig. 1), etc. Cost-based optimizer chooses the plan that has the lowest cost using dynamic programming approaches (e.g., in System R).

Cost-based optimization is more effective than rule-based optimization, since all decisions taken on a query execution plan are validated by a cost model. An important point to be mentioned is that the quality of cost-based optimization depends strongly on the



Optimization and Tuning in Data Warehouses. Figure 1. Two different query execution plans.

recency of the statistics. Determining which statistics to create is a difficult task [6].

Due to the difficulty query optimizers have in selecting an optimal execution plan, some commercial database systems offer the data warehouse administrator to use *hints* in order to force query optimizer to choose an execution plan.

The above cited optimization techniques are enough to optimize traditional database applications (called, OLTP: On-Line Transaction Processing). It will be interesting to see whether they are also sufficient for decision support applications built around a large data warehouse.

A data warehouse is usually modeled with a relational schema (star schema, snow flake schema). A star schema consists of a single fact table that is related to multiple dimension tables via foreign key joins. Dimension tables are relatively small compared to the fact table and rarely updated. They are typically *denormalized* so as to minimize the number of join operations required to evaluate a query. Due to the interactive nature of decision support applications, having a fast query response time is a critical performance goal. The above optimization techniques are not suitable for data warehouse applications due to their different requirements and workload. Data warehouse applications operate in mostly-read environments, which are dominated by large and complex queries. The typical queries on the star schema are called *star join queries*. They are characterized by: (i) a *multi-table* join among a *large fact table* and dimension tables and (ii) each of the dimension tables involved in the join operation has *multiple* selection predicates (a selection predicate has the following form: $D_i.A_j\ \theta\ value$, where A_j is an attribute of dimension table D_i and θ is one of six comparison operators $\{=,<,>,\leq,\geq\}$, and value is the predicate constant) on its descriptive attributes and (iii) no join operation between dimension tables. Unfortunately, conventional query optimization techniques are not efficient for star-join queries for two main reasons. First, traditional indexing techniques (B-tree) are not efficient to process such a selection predicate. This is due to the fact that dimension attributes often have low cardinality (e.g., gender) and each selection predicate typically has a low selectivity. Second, since the fact table is very large, computing the multi-table joins using traditional join algorithms (nested loop, sort-merge, hash joins) is inefficient because it requires scanning the fact table.

Without efficient optimization techniques, such queries may take hours or days, which is unacceptable in most cases. As a consequence, the physical design becomes sophisticated to cope with complex decision support queries [6]. To speed up these queries, in addition to the existing ones (developed for OLTP applications), a large spectrum of optimization techniques were proposed in the literature and mostly supported by commercial database systems. These techniques include materialized views, partitioning, advanced indexing schemes, denormalization, parallel processing. In the next section, all these techniques will be described in details. For each one, its principle, advantages, disadvantages and selection problem will be presented.

Foundations

The various optimization techniques selected during the physical design process may be classified into two main categories: redundant techniques and non-redundant techniques.

Redundant Techniques

This category includes four main techniques: materialized views, advanced indexing schemes, denormalization, and vertical partitioning.

1. *Materialized views.* A virtual view (A view is a derived relation defined in terms of base relations.) can be materialized by storing its tuples in the databases. Materialized views are used to precompute and to store aggregated data. They can also be used to precompute joins with or without aggregations. So, materialized views are suitable for queries with expensive joins or aggregations. Once materialized views are selected, all queries will be rewritten using materialized views (this process is known as *query rewriting*). A rewriting of a query Q using views is a query expression Q' referencing to these views. The query rewriting is done *transparently* by the query optimizer. To generate the best rewriting for a given query, a cost-based selection method is used. Two major problems related to materialized views are: (i) the view selection problem and (ii) the view maintenance problem.

Views selection problem. The database administrator cannot materialize all possible views, as he/she is constrained by some resources like, disk space, computation time, maintenance overhead and cost required for query rewriting process. Hence, he/she needs to select an appropriate set of views to materialize under some resource constraint. Formally, view selection problem (VSP) is defined as follows: given a set of most frequently used queries $Q = \{Q_1, Q_2, \ldots, Q_n\}$, where each query Q_i has an access frequency f_i ($1 \leq i \leq n$) and a resource constraint M, the view selection problem consists of selecting a set of materialized views that minimizes one or more objectives, possibly subject to one or more constraints. Many variants of this problem have been studied: (i) minimizing the query processing cost subject to storage size constraint [5], (ii) minimizing query cost and maintenance cost subject to storage space constraint [9], (iii) minimizing query cost under a maintenance constraint [8], etc. This problem is known to be an NP-hard problem [8]. Several algorithms were proposed to deal with this problem [8,9]. The following table summarizes all possible formalizations of view selection problem. Two symbols are used in this table (√ and ?), where each one has its own interpretation: √: formalizations and selection algorithms already exist and ?: Inapplicable.

Objectives	*Constraints*		
	Without constraints	Maintenance cost	Storage cost
Query cost	√	√	√
Maintenance cost	√	?	√
Query cost & Maintenance Cost	√	?	√

View maintenance problem. Note that materialized views store data from base tables. In order to keep the views in the data warehouse up to date, it is necessary to maintain the materialized views in response to the changes at the base tables. This process of updating views is called view maintenance which has generated a great deal of interest. Views can be either recomputed from scratch, or incrementally maintained by propagating the base data changes onto the views. As recomputing the views can be prohibitively expensive, the incremental maintenance of views is of significant value [8].

2. *Indexing* has been at the foundation of performance tuning for databases for many years. A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns. An index can be either clustered or non-clustered. It can be defined on one table (or view) or many tables using a join index [10]. The traditional indexing strategies used in database systems do not work well in data warehousing environments since most OLTP queries are point queries. B-trees, which are used in most common relational database systems, are geared towards such point queries. In the data warehouse context, indexing refers to two different things: (i) indexing techniques and (ii) index selection problem.

Indexing techniques. A number of indexing strategies have been suggested for data warehouses: Value-List Index, Projection Index, Bitmap Index, Bit-Sliced Index, Data Index, Join Index, and Star Join Index. Bitmap index is probably the most important result obtained in the data warehouse physical optimization field. The bitmap index is more suitable for low cardinality attributes since its size strictly depends on the *number of distinct values of the column on which it is built*. Besides disk space saving (due to their binary representation and potential compression), such index speeds up queries having Boolean operations (such as AND, OR and NOT) and COUNT operations. Bitmap join index is proposed to speed up join operations. In its simplest form, it can be defined as a bitmap index on a table *R* based on a single column of another table *S*, where *S* commonly joins with *R* in a specific way.

Index selection problem. The task of index selection is to automatically select an appropriate set of indices for a data warehouse (having a fact table and dimension tables) and a workload under resource constraints (storage, maintenance, etc.). It is challenging for the following reasons [3]: the size of a relational data warehouse schema may be large (many tables with several columns), and indices can be defined on a set of columns. Therefore, the search space of indices that are relevant to a workload can be very large [2]. To deal with this problem, most selection approaches use two main phases: (i) *generation of candidate attributes* and (ii) *selection of a final configuration*. The first phase prunes the search space of index selection problem, by eliminating *non-relevant* attributes. In the second phase, the final indices are selected using greedy algorithms [4], linear programming algorithms [3], etc. The *quality of the final set of indices depends essentially on the pruning phase*. To prune the search space of index candidates, many approaches were proposed [1–3], that can be classified into two categories: *heuristic enumeration-driven approaches* and *data mining driven approaches*.

In heuristic enumeration-driven approaches, heuristics are used. For instance, in [4], a greedy algorithm is proposed that uses optimizer cost of SQL Server to accept or reject a given configuration of indices. The weakness of this work is that it *imposes the number of generated candidates*. IBM DB2 Advisor is another example belonging to this category [15], where the query parser is used to pick up selection attributes used in workload queries. The generated candidates are obtained by a few *simple combinations* of selection attributes [15].

In data mining-driven approaches, the pruning process is done using data mining techniques, like in [2]. In this approaches the number of index candidates is not a priori known as in the first category. The basic idea is to generate frequent closed itemsets representing groups of attributes that could participate in selecting the final configuration of bitmap join indexes. A data mining based approach has been developed for selecting bitmap join indexes [2].

3. *Vertical partitioning* can be viewed as a redundant structure even if it results in little storage overhead. The vertical partitioning of a table *T* splits it into two or more tables, called, sub-tables or vertical fragments, each of which contains a subset of the columns in *T*. Note that the key columns are duplicated in each vertical fragment, to allow "reconstruction" of an original row in *T*. Since many queries access only a small subset of the columns in a table, vertical partitioning can reduce the amount of data that needs to be scanned to answer the query. Unlike horizontal partitioning, indexes or materialized views, in most of today's commercial database systems there is no native database definition language support for defining vertical partitions of a table [14].

To vertically partition a table with m non-primary keys, the number of possible fragments is equal to $B(m)$, which is the mth Bell number [12]. For large values of m, $B(m) \cong m^m$. For example, for m = 10; $B(10) \cong 115{,}975$. These values indicate that it is futile to attempt to obtain optimal solutions to the vertical partitioning problem. Many algorithms were proposed and classified into two categories: grouping and splitting [12]. Grouping starts by assigning each attribute to one fragment, and at each step, joins some of the fragments until some criteria is satisfied. Splitting starts with a table and decides on beneficial partitionings based on the query frequencies.

In the data warehousing environment, [7] proposed an approach for materializing views in vertical fragments, each including a subset of measures possibly taken from different cubes, aggregated on the same grouping set. This approach may unify two or more views into a single fragment.

4. *Denormalization* is the process of attempting to optimize the performance of a database by adding redundant data to save join operations. Denormalization is usually promoted in a data warehouse environment.

Non Redundant Techniques

In this category, two main techniques exist: horizontal partitioning and parallel processing.

1. *Horizontal partitioning* represents an important aspect of physical database design. It allows tables, indexes and materialized views to be partitioned into *disjoint* sets of rows that are physically stored and accessed separately [14] or in parallel. Horizontal partitioning may have a significant impact on performance of queries and manageability of very large data warehouses. Not only do data partitions reduce the time it takes to perform database maintenance and management tasks, by eliminating non-relevant partition(s), they also have a positive effect on the performance of applications. Another characteristic of horizontal partitioning is its ability to be combined with other optimization structures like indexes, materialized views. Splitting a table, a materialized view or an index into smaller pieces makes all operations on individual pieces much faster. Contrary to materialized views and indexes, data partitioning does not replicate data, thereby reducing space requirements and minimizing update overhead [13].

A native database definition language support is available for horizontal partitioning, where several fragmentation modes are available [11]: range, list and hash. In the range partitioning, an access path (table, view, and index) is decomposed according to a range of values of a given set of columns. The hash mode decomposes the data according to a hash function (provided by the system) applied to the values of the partitioning columns. The list partitioning splits a table according to the listed values of a column. These methods can be combined to generate composite partitioning (List-List, Range-Range, Hash-Hash, Range-List, ...). Recently, a new mode of horizontal partitioning became available in Oracle11g [11], called *virtual column-based partitioning*. It is defined by one of the above mentioned techniques and the partitioning key is based on a virtual column. Virtual columns are not stored on disk and only exist as metadata.

Two versions of horizontal partitioning are available [12]: *primary* and *derived* horizontal partitioning. Primary horizontal partitioning of a table is performed using predicates defined on that relation. It can be performed using the different fragmentation modes above cited. Derived horizontal partitioning is the partitioning of a table that results from predicates defined in other table(s). The derived partitioning of a table R according to a fragmentation schema of table S is feasible if and only if there is a join link between R and S.

In the context of relational data warehouses, derived horizontal partitioning is well adapted. In other words, to partition a data warehouse, the best way is to *partition some/all dimension tables using their predicates, and then partition the fact table* based on the fragmentation schemas of dimension tables. This fragmentation takes into consideration requirements of star join queries (these queries impose restrictions on the dimension values that are used for selecting specific facts; these facts are further grouped and aggregated according to the user demands). To illustrate this fragmentation, suppose that a relational warehouse is modeled by a star schema with d dimension tables and a fact table F. Among these dimension tables, g tables are fragmented ($g \leq d$). Each dimension table D_i ($1 \leq i \leq g$) is partitioned into m_i fragments: $\{D_{i1}, D_{i2}, ..., D_{im_i}\}$, where each fragment D_{ij} is defined as: $D_{ij} = \sigma_{cl_j^i}(D_i)$, where cl_j^i and σ ($1 \leq i \leq g, 1 \leq j \leq m_i$) represent a conjunction of simple predicates and the selection operator, respectively. Thus, the fragmentation schema of the fact table F is defined as follows: $F_i = F \ltimes D_{1j} \ltimes D_{2k} \ltimes ... \ltimes D_{gl}$, ($1 \leq i \leq m_i$), where $\ltimes$ represents the semijoin operation.

Derived horizontal partitioning has two main advantages in relational data warehouses, in addition to classical benefits of data partitioning: (i) precomputing joins between fact table and dimension tables participating in the fragmentation process of the fact table [1] and (ii) optimizing selections defined on dimension tables. Similar advantages hold for bitmap join indexes.

2. *Parallel Processing* Data partitioning is always coupled with parallel processing. To design a parallel data warehouse, two main issues must be addressed: data partitioning and data placement (allocation). Data placement is a key factor for high performance parallel data warehouses. Determining an effective data placement is a complex administration problem depending on many parameters including system architecture, database and workload characteristics, hardware configuration, etc. The easier way to design a parallel data warehouse is to first partition dimension tables using the primary horizontal partitioning and then derived partition the fact table. This partitioning alternative generates a set of sub-star schemas. In order to ensure a high performance of complex queries, these sub-star schemas shall be allocated to various machines in efficient manner.

Tuning

Before talking about tuning phase, a summarization of physical design is required to understand the need for tuning.

The first point concerns the different formalizations of problems of selecting optimization techniques in physical design phase. They are mostly based on a set of most frequently asked queries, *a priori known.* However, in dynamic environments, like data warehousing with various ad-hoc queries, it is difficult to identify potential useful optimization structures in advance. The second point is about the similarities between optimization techniques: materialized views and indexes, bitmap join indexes and derived horizontal partitioning. These similarities are not always taken into account during physical design phase. This situation may incur the following limitations: (i) non-consideration of the mutual interdependencies between optimization structures (sometimes it is better to select more materialized views than indexes and vice-versa or replacing bitmap join indexes by a non redundant technique like derived horizontal partitioning to reduce maintenance overhead) gives sub-optimal solutions, (ii) absence of metrics for efficient distribution of storage space between redundant optimization techniques and (iii) redistribution of space among optimization structures after update operations.

Based on the above points, tuning tools are recommended since they supervise the good use of different optimization techniques selected during physical design phase. Tuning tools might be triggered when user requirements evolve (new queries/updates, not considering some existing queries), query frequencies change, sizes of tables, materialized views, indexes and partitions increase, etc. To keep data warehouse applications running at high performance, several aspects of physical design should be tuned: buffer pool, allocation of working memory, materialized views, indexes, storage space, horizontal partitioning, vertical partitioning, data placement, etc.

During the data warehouse life cycle, some structures may be added/dropped (e.g., materialized views and indexes), merged (indexes and horizontal partitions) or splitted (e.g., horizontal partitions). For instance, some commercial database systems provide monitoring tools observing the good utilization of indexes. If an index is not used by a workload, it will be dropped. Its storage space might be used for creating another optimization technique. Merging operations deal mainly with indexes and horizontal partitions. They are crucial for data warehouse applications [5,14]. This is because optimization structures are often either too large (for redundant structures) to fit in the available storage, or cause updates to slow down significantly. They are supported by most commercial database systems.

Many commercial database systems offer tools for physical design tuning: "What-If" analysis tool of SQL Server used to facilitate manual tuning. SQL Server proposes a tuning using a relation-based approach. The optimizer can replace a large useful index with smaller, less useful ones. For example, operations that required a single traversal through a complex index may be implemented as the intersection of two traversals through simple indexes. Other transformations include index merging (implementing an index scan of relation B as a full scan through relation A), prefixing (building an index on a; b instead of a; b; c), and the removal of structures. DB2 design advisor tool provides integrated recommendations for indexes, materialized views, shared-nothing partitioning and multidimensional clustering. ORACLE 10G takes as input a workload and a set of optimization candidates for that workload (these candidates are generated by Oracle Automatic Tuning Optimizer) and provides a recommendation for the overall workload.

In academic research work, a tuning tool called AutoPart which combines horizontal and vertical table partitioning to reduce I/O costs for each query by eliminating unnecessary accesses to non-relevant data is proposed [13]. AutoPart recommends the combination of partitioning with a small set of key indexes. A similar work proposed to combine derived horizontal partitioning with bitmap join indexes [1].

Key Applications

The proposed techniques within this paper can be applied in any database applications having the same characteristics (huge tables, complex queries with many join operations and restriction) and requirements of data warehouse (response time). Scientific and statistical database applications are a good example. Historically, the main techniques explored in this paper were proposed and supported in decision support applications. Materialized views and advanced indexing schemes could be easily applied in traditional OLTP applications, when update operations are not important. Horizontal partitioning can also be applied, but moderately. The choice of partitioning attributes is a crucial performance issue. For example, if a database is partitioned based on changing value attributes like Age, the database will be faced with the problem of *instance migration*.

Future Directions

As mentioned in the previous section, physical design and tuning are very crucial decisions for the performance of data warehouse applications. In this section, some of the interesting open issues for physical design and tuning are highlighted:

1. Multi-objective algorithms for indexes: most index selection algorithms have one objective function which represents the query processing cost subject to one constraint representing the storage cost. It will be interesting to propose multi-objective algorithms for selecting indexes. Such formalizations will reduce the query processing and maintenance cost (which is not negligible for indexes). Since there is a strong similarity between materialized views and indexes, an easier way to deal with this problem is to adapt multi-objective algorithms for materialized views to indexes.
2. Incorporating query rewriting using materialized views in selection process: after selection of materialized views, all queries will be rewritten using them. Choosing an optimal rewriting is a difficult problem. It will be interesting to combine the problem of selecting materialized views and the problem of rewriting queries. A simple combination may involve the formalization of materialized view selection subject to time requiring for query rewrite process.
3. Pruning search space of materialized view selection problem: selecting materialized views is an NP-hard problem. To prune search space of this problem, vertical and horizontal partitioning (primary and derived) might be used, because a materialized view may involve selection, projection, join operations.
4. Partition allocation over table spaces: assigning different fragments generated by horizontal partitioning process over various table spaces may be a crucial issue for performance of queries. This problem does not get enough attention from data warehouse research community. This problem is quite similar to data placement studied in distributed and parallel databases areas. It will be interesting to adapt the existing algorithms. Most of these algorithms are static. Tuning of data placement will be recommended since partition usages change, partition might be merged/splitted, etc.
5. Supporting derived horizontal partitioning: Today's commercial database systems support derived horizontal partitioning, where a table is decomposed based on the fragmentation schema of only one table, using referential partitioning [11]. In real data warehouse applications, a fact table may be derived partitioned based on fragmentation schemas of several dimension tables in order to satisfy star join queries requirements. From an industry perspective, this situation is quite unsatisfactory and requires further thought.

Cross-references

► Bitmap-Based Index Structures for Multidimensional Data
► Data Partitioning
► Index Join Physical Schema Design
► Query Rewriting Using Views
► Semijoin
► View Maintenance in Data Warehouses
► Virtual Partitioning

Recommended Reading

1. Bellatreche L., Boukhalfa K., and Mohania M.K. Pruning search space of physical database design. In Proc. 18th Int. Conf. Database and Expert Syst. Appl. 2007, pp. 479–488.
2. Bellatreche L., Missaoui R., Necir H., and Drias H. Selection and pruning algorithms for bitmap index selection problem using data mining. In Proc. Int. Conf. on Data Warehousing and Knowledge Discovery, 2007, pp. 221–230.
3. Chaudhuri S. Index selection for databases: a hardness study and a principled heuristic solution. IEEE Trans. Knowl. Data Eng., 16 (11):1313–1323, 2004.
4. Chaudhuri S. and Narasayya V. An efficient cost-driven index selection tool for microsoft sql server. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997, pp. 146–155.
5. Chaudhuri S. and Narasayya V. Index merging. In Proc. 15th Int. Conf. on Data Engineering. 1999, pp. 296–303.
6. Chaudhuri S. and Narasayya V. Self-tuning database systems: a decade of progress. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
7. Golfarelli M., Maniezzo V., and Rizzi S. Materialization of fragmented views in multidimensional databases. Data & Knowl. Eng., 49(3):325–351, June 2004.
8. Gupta H. Selection and maintenance of views in a data warehouse. Ph.D. Thesis, Stanford University, September 1999.

9. Lawrence M. Multiobjective genetic algorithms for materialized view selection in OLAP data warehouses. In Proc. The Genetic and Evolutionary Computation Conf., 2006, pp. 699–706.
10. O'Neil P. and Quass D. Improved query performance with variant indexes. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 38–49.
11. Oracle Data Sheet. Oracle partitioning. White Paper: http://www.oracle.com/technology/products/bi/db/11g/, 2007
12. Özsu M.T. and Valduriez P. Principles of distributed database systems. Second edition. Prentice Hall, Englewood Cliffs, NJ, 1999.
13. Papadomanolakis S. and Ailamaki A. Autopart: automating schema design for large scientific databases using data partitioning. In Proc. 16th Int. Conf. on Scientific and Statistical Database Management, 2004, pp. 383–392.
14. Sanjay A., Narasayya V.R., and Yang B. Integrating vertical and horizontal partitioning into automated physical database design. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 359–370.
15. Valentin G., Zuliani M., Zilio D.C., Lohman G.M., and Skelley A. Db2 advisor: an optimizer smart enough to recommend its own indexes. In Proc. 16th Int. Conf. on Data Engineering, 2000, pp. 101–110.

Optimization of DAG-Structured Query Evaluation Plans

► Multi-Query Optimization

Optimization of Parallel Query Plans

► Parallel Query Optimization

OQL

Peter M.D. Gray
University of Aberdeen, Aberdeen, UK

Synonyms

Object query language

Definition

OQL was developed to play the role of SQL for *Object-Oriented Databases*, especially those adhering to the *ODMG Standard* [4] where the language is defined. Unlike SQL, OQL is a functional language, and its operators can be composed to an arbitrary level of nesting within a query provided the query remains type-correct. Fegaras and Maier [8] have shown how OQL expressions have a direct translation into monoid *Comprehensions*.

Optimisation techniques for OQL that exploit its inherent functional nature are discussed in [5,6,8]. OQL has been influential in the development of the SQL3 standard and also the functional core of the XQuery language for XML. Thus optimisation techniques developed for OQL are also applicable to these languages.

Key Points

The fundamental modelling concept of *object identifiers* for entity instances was accepted into the database mainstream in the late 1980s, and the move to using SQL-like syntax for querying such data models followed soon after. Early influential systems were OSQL [2] and *AMOSQL* (q.v.). This resulted in query language proposals for object-oriented databases such as the very influential O2 query language [1] and its successor OQL, which was included in the ODMG Standard [4]. For example, the DAPLEX query

```
FOR EACH S IN STUDENT
  SUCH THAT name(S)="Fred Jones"
      PRINT name(S), age(S);
```

is expressed as follows in OQL, basically by syntactic reordering of the query clauses and using path expressions rather than function application:

```
SELECT S.name, S.age FROM STUDENT S
WHERE S.name="Fred Jones"
```

When restricted to sets, monoid comprehensions are equivalent to set monad comprehensions [3], which capture precisely the nested relational algebra [8]. Most OQL expressions have a direct translation into the monoid calculus. For example, the OQL query

```
SELECT DISTINCT HOTEL.price
FROM HOTEL IN(
  SELECT h
  FROM c IN CITIES, h IN c.hotels
  WHERE c.name="Arlington")
WHERE EXIST r IN HOTEL.rooms:r.bed_num = 3
AND HOTEL.name IN (
  SELECT t.name
  FROM s IN STATES, t IN s.attractions
  WHERE s.name = "Texas" );
```

finds the prices of hotels in Arlington that have rooms with three beds and are also are named after a tourist attraction in Texas. This query is translated into the following monoid comprehension [7]:

```
fold(Union,Empty,
  [ price(h) | c <- Cities; h <- hotels
  (c); name(c) = ''Arlington'';
        fold(Or,False,
        [bednum(r)=3 | r <- rooms(h) ]),
        fold(Or,False,
  [ name(h)=name(t) | s <- States; t <-
  attractions(s);name(s)=''Texas''])])
```

Here, as in Functional Programming

```
  fold(Or,False,[x1,x2, ... xn]) = x1
  Or x2 Or ... xn Or False
```

computes the logical *Or* of a list of boolean values, so it is true only if *some* of them are true. Likewise *fold(Union,Empty,L)* copies the list L into a set without duplicates. Mathematically *fold* implements *monoid* operations with a given *merge* operation and a *zero*.

Cross-references

▶ AMOSQL
▶ Comprehensions
▶ Functional Query Language

Recommended Reading

1. Bancilhon F., Delobel C., and Kanellakis P.C. Building an Object-Oriented Database System, The Story of O2. Morgan Kaufmann, Los Altos, CA, 1992.
2. Beech D. A foundation of evolution from relational to object databases. In Advances in Database Technology, In Proc. 1st Int. Conf. on Extending Database Technology. 1988, pp. 251–270.
3. Buneman P., Libkin L., Suciu D., Tannen V., and Wong L. Comprehension syntax. ACM SIGMOD Rec., 23(1):87–96, 1994.
4. Cattell R.G.G. (ed.). The Object Data Standard: ODMG 3.0. Morgan Kaufmann, Los Altos, CA, 2000.
5. Cluet S. and Delobel C. A general framework for the optimization of object-oriented queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1992, pp. 383–392.
6. Fegaras L. Query unnesting in object-oriented databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1998, pp. 49–60.
7. Fegaras L. Query Processing and Optimization in λ-DB. In The Functional Approach to Data Management, Chapter 13. P.M.D., Gray L., Kerschberg P.J.H., and King A. (eds.). Springer, Berlin, 2004.
8. Fegaras L. and Maier D. Towards an effective calculus for Object Query Languages. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1995, pp. 47–58.

ORA-SS Data Model

▶ Object Relationship Attribute Data Model for Semi-structured Data

ORA-SS Schema Diagram

▶ Object Relationship Attribute Data Model for Semi-structured Data

Orchestration

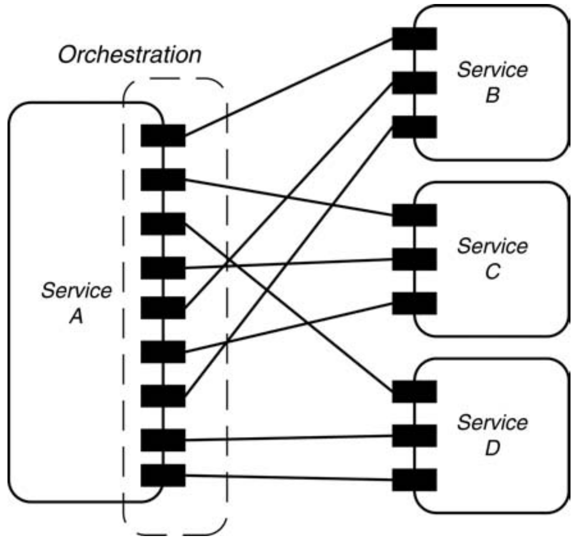
W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven,
The Netherlands

Definition

In a Service Oriented Architecture (SOA) services are interacting by exchanging messages, i.e., by combining services more complex services are created. Orchestration is concerned with the composition of such services seen from the viewpoint of single service.

Key Points

The terms "orchestration" and "choreography" describe two aspects of integrating services to create business processes [2,3]. The two terms overlap somewhat and the distinction is subject to discussion. Orchestration and choreography can be seen as different "perspectives." Choreography is concerned with the exchange of messages between those services and is often be characterized by analogy "Dancers dance following a global scenario without a single point of control." Orchestration is concerned with the interactions of a single service with its environment. Here an analogy can also be used. In orchestration, there is



Orchestration. Figure 1. Orchestration.

someone, “the conductor”, who tells everybody in the orchestra what to do and makes sure they all play in sync.

Figure 1 illustrates the notion of orchestration. Service A is interacting with other services to create a more complex service. The dashed area shows the focal point of orchestration, i.e., the control-flow related to message exchanges of a single party. Languages such a BPEL are proposed to model and enact such orchestrations [1]. Note that languages like BPEL are very close to traditional workflow languages, i.e., the same types of control-flow patterns need to be supported.

Orchestration often assumes that services have a “buy side” and a “sell side,” i.e., services can be used by other services (“sell side”) and at the same time use services (“buy side”). Orchestration is mainly concerned with the “buy side.” Unlike choreography, there is a single party coordinating the process.

Cross-references

▶ BPEL
▶ Business Process Management
▶ Orchestration
▶ Web Services
▶ Workflow Management

Recommended Reading

1. Alves A., Arkin A., Askary S., Barreto C., Bloch B., Curbera F., Ford M., Goland Y., Guzar A., Kartha N., Liu C.K., Khalaf R., Koenig D., Marin M., Mehta V., Thatte S., Rijn D., Yendluri P., and Yiu A. Web Services Business Process Execution Language Version 2.0 (OASIS Standard). WS-BPEL TC OASIS, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007.
2. Dumas M., van der Aalst W.M.P., and ter Hofstede A.H.M. Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley, New York, 2005.
3. Weske M. Business Process Management: Concepts, Languages, Architectures. Springer, Berlin, 2007.

ORDB (Object-Relational Database)

▶ Object Data Models

Order Item

▶ Clinical Order

Order Statistics

▶ Quantiles on Streams

Ordering

▶ Similarity and Ranking Operations

Orientation Relationships

▶ Cardinal Direction Relationships

Oriented Clustering

▶ Subspace Clustering Techniques

Origin

▶ Provenance
▶ Provenance in Scientific Databases

OR-Join

Nathaniel Palmer
Workflow Management Coalition, Hingham,
MA, USA

Synonyms

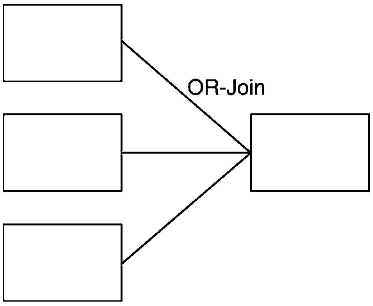
Synchronous join

Definition

The point of convergence within a workflow following alternative, mutually exclusive paths.

Key Points

An OR-Join (Fig. 1) represents a point within a workflow where two or more alternative workflow branches re-converge following an OR-Split into a single common activity as the next step within the workflow. In contrast with an AND-Join, no parallel activity execution has occurred at the join point, therefore no synchronization is required. With an OR-Join a thread of control may arrive at the specific activity via any of several alternative preceding activities.



OR-Join. Figure 1. OR-Join.

Cross-references

- ► Join
- ► OR-Split
- ► Process Life Cycle
- ► Workflow Management and Workflow Management System

OR-Split

Nathaniel Palmer
Workflow Management Coalition, Hingham,
MA, USA

Synonyms

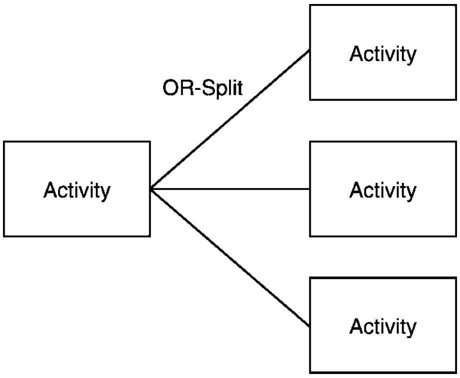
Conditional branching; Conditional routing; Switch; Branch

Definition

A point within the workflow where a single thread of control makes a decision upon which branch to take when encountered with multiple alternative workflow branches.

Key Points

An OR-Split (Fig. 1) establishes alternative and mutually exclusive workflow branches. For example, in a mortgage application, different paths may represent different branches based on conditional logic, such as credit risk or the amount to be barrowed. As paths are mutually exclusive, no parallel execution of activities occur and thus no synchronization is required, so the workflow branching converges with an OR-Join rather than an AND-Join. An OR-Split is conditional and the (single) specific transition to next activity is selected according to the outcome of the Transition Condition(s).



OR-Split. Figure 1. OR-Split.

Cross-references

- ► AND-Join
- ► OR-Join

▶ Process Life Cycle
▶ Split
▶ Workflow Management and Workflow Management System

OSD

▶ Network Attached Secure Device

OSQL

Tore Risch
Uppsala University, Uppsala, Sweden

Definition

OSQL [1,2] is an functional query language and data model similar to Daplex, first implemented in the Iris DBMS [4]. The data model of OSQL is object oriented with three kinds of system entities: objects, types, and functions. A database consists of a set of objects, the objects are classified into types, and functions define the semantics of types. The data model is similar to an ER model with the difference that both entity relationships and attributes are represented as functions and that (multiple) inheritance among entity types is supported. OSQL provide object identifiers (OIDs) as first class objects, and, unlike Daplex, queries can return OIDs in results. Queries are expressed using a SELECT syntax similar to SQL. Derived functions are also defined using select statements similar to functions in SQL-2003.

Key Points

With the OSQL data model a database consists of a set of objects. The objects are classified into subsets by types and each type has an extent consisting of the objects belonging to that type. Type inheritance is supported with the type named OBJECT as most general type. The extent of a type is a subset of the extents of its supertype(s). For example if entity type STUDENT is a subtype of type PERSON then the extent of type STUDENT is also a subset of the extent PERSON. Types are defined dynamically using a CREATE TYPE statement, e.g.,:

```
CREATE TYPE STUDENT SUBTYPE OF PERSON;
```

Functions define relationships among entities and properties of entities. Functions can be stored in the databases, derived in terms of other functions, or be defined as foreign functions implemented in some conventional programming language. Stored functions correspond to tables in relational databases, and derived functions are parameterized views similar to function definitions in SQL-2003.

Queries and derived functions are defined declaratively using a SELECT statement, e.g.,:

```
SELECT NAME(P)
FOR EACH PERSON P
WHERE AGE(P)>20 AND SEX(P) =
``Female'';

CREATE FUNCTION GRANDPARENTS(PERSON
P)-> PERSON
AS SELECT PARENT(PARENT(P));
```

Queries are expressed as constraints over extents. Functions composition allows easy traversal of relationships between entity types. As in Daplex, if a function returns a set of objects (e.g., PARENT) functions applied on it iterate over the elements of the set. This is a form of extended path expressions through function composition.

OSQL was implemented in the Iris DBMS [4] and HP's OpenODB product. The Amos II DBMS [3] uses a modified OSQL language, AmosQL.

Cross-references

▶ AmosQL
▶ Daplex
▶ Functional Data Model

Recommended Reading

1. Beech D. A foundation of evolution from relational to object databases. In Advances in Database Technology, In Proc. 1st Int. Conf. on Extending Database Technology. 1988, pp. 251–270.
2. Fishman D.H., Beech D., Cate H.P., Chow E.C., Connors T., Davis J.W., Derrett N., Hoch C.G., Kent W., Lyngbaek P., Mahbod B., Neimat M.A., Ryan T.A., and Shan Iris M.C. An Object-Oriented Database Management System, ACM Trans. Off. Inf. Syst., 5(1):48–69, 1987.

3. Risch T., Josifovski V., and Katchaounov T. Functional data integration in a distributed mediator system. In Functional Approach to Data Management – Modeling, Analyzing and Integrating Heterogeneous Data, P. Gray, L. Kerschberg, P. King, A. Poulovassilis (eds.). Springer, Berlin, 2003.
4. Wilkinson K., Lyngbaek P., and Hasan W. The iris architecture and implementation, IEEE Trans. Knowl. Data Eng., 2(1):63–75, 1990.

Overlay Network

Wojciech Galuba, Sarunas Girdzijauskas
Ecole Polytechnique Fédérale de Lausanne (EPFL),
Lausanne, Switzerland

Definition

An *overlay network* is a communication network constructed on top of an another communication network. The nodes of the overlay network are interconnected with logical connections, which form an *overlay topology*. Two overlay nodes may be connected with a logical connection despite being several hops apart in the underlying network. Overlay networks may define their own *overlay address space* which is used for efficient message routing in the overlay topology.

Key Points

When a distributed application is deployed in a computer network, the individual nodes on which the application is running need to be able to discover and communicate with one another. A solution to this problem is the overlay network. The overlay network interconnects all the application nodes and provides the basic communication primitives such as flooding, random walks or point-to-point overlay message routing and multicast.

Overlay networks are typically deployed on top of the Internet and by far the most common usage is in peer-to-peer systems. For example, Gnutella, an early peer-to-peer file sharing system connects all the peers in an overlay network, each peer shares its files, and files are searched for using query flooding in the overlay network.

Overlay network topologies can be divided into two broad classes: unstructured and structured. Unstructured overlay networks do not construct a globally consistent topology, instead peers choose their neighbor sets independently and in a largely ad-hoc way. In unstructured overlay networks nodes reach the other nodes by message flooding or random walks. Structured overlays define an address space and each of the overlay nodes has a unique address. The addresses are used to construct an overlay topology that enables efficient and scalable messages passing between the overlay nodes. In most of the modern structured overlays the expected number of routing hops scales as $O(log\ N)$ with the network size. Distributed Hash Tables are a specific case of structured overlay networks. Apart from structured and unstructured there also exist hybrid overlays.

Overlay networks are designed to be robust to *churn*, i.e., arrivals and departures of the overlay network nodes to and from the network. As overlay network nodes loose their overlay topology connections, new connections have to be added in their place. In structured overlay networks the additional challenge is to maintain the overlay topology such that the overlay routing remains efficient, i.e., the routing paths are kept short.

Cross-references

► Distributed Hash Table
► Peer to Peer Overlay Networks: Structure, Routing and Maintenance
► Peer-to-Peer System

OWL: Web Ontology Language

Sean Bechhofer
University of Manchester, Manchester, UK

Synonyms

Web ontology language

Definition

The Web Ontology Language OWL is a language for defining ontologies on the Web. An OWL Ontology describes a domain in terms of classes, properties and individuals and may include rich descriptions of the characteristics of those objects. OWL ontologies can be used to describe the properties of Web resources. Where earlier representation languages have been used to develop tools and ontologies for specific user-communities in areas such as sciences, health and e-commerce, they were not necessarily designed to be compatible with the

World Wide Web, or more specifically the Semantic Web, as is the case with OWL.

Features of OWL are a collection of expressive operators for concept description including boolean operators (intersection, union and complement), plus explicit quantifiers for properties and relationships; the ability to specify characteristics of properties, such as transitivity or domains and ranges; a well defined semantics facilitating the use of inference and automated reasoning; the use of URIs for naming concepts and ontologies; a mechanism for importing external ontologies; and compatability with the architecture of the World Wide Web, in particular other representation languages such as RDF and RDF Schema.

OWL consists of a suite of World Wide Web Consortium (W3C) Recommendations – six documents published in February 2004 describe Use Cases and Requirements, an Overview of the language, a Guide, Reference, OWL Semantics and a collection of Test Cases [3].

Key Points

Ontology languages allow the representation of ontologies. An ontology "defines a set of representational primitives with which to model a domain of knowledge or discourse" (see Ontology). The definition of an ontology can encompass a wide range of artefacts, from simple word lists, through taxonomies, thesauri and rich logic-based models and there are a corresponding range of languages for their representation.

Standardization of representation languages is a cornerstone of the Semantic Web effort. A standard representation facilitates interoperation – in particular, well-defined, unambiguous *semantics* ensure that applications can agree on the meaning of expressions. OWL is intended to provide that standard representation.

OWL builds on RDF and RDF Schema and adds more vocabulary for describing properties and classes. The design of the language was influenced by a number of factors. Description Logics, Frame-based modeling paradigms, and Web languages RDF and RDF Schema were key inputs, as was earlier work on languages such as OIL and DAML+OIL.

Knowledge Representation in a Web setting introduces particular requirements such as the distribution across many systems; scalability to Web size; compatibility with Web standards for accessibility and internationalization; and openness and extensibility. OWL uses URIs for naming and extends the description framework for the Web provided by RDF to address some of the issues above.

OWL defines three sublanguages: OWL Lite, OWL DL and OWL Full. OWL Full is essentially RDF extended with additional vocabulary, with no restrictions on the way in which that vocabulary is used. OWL DL places restrictions on the way in which the vocabulary can be used in order to define a language for which a number of key reasoning tasks (for example concept satisfiability or subsumption) are decidable. OWL Lite further restricts the expressivity allowed – for example, explicit union or complement are disallowed in OWL Lite. OWL DL and OWL Lite have a model theoretic semantics that corresponds to a Description Logic (DL) [1] and thus facilitate the use of DL reasoners to provide reasoning support for the language [2].

The design of representation languages often involves trade-offs, and there are limitations on what can be expressed using OWL, in particular in OWL-DL. These limitations have been selected primarily to ensure that these language subsets are well-behaved computationally, with decidable procedures for concept satisfiability. For example, OWL does not provide support for general purpose rules, which are seen as an important paradigm in knowledge representation, for example in expert systems or deductive databases. Extensions to OWL are being proposed to cover, among others, rules, query, additional expressivity, metamodeling and fuzzy reasoning.

Cross-references

▶ Description Logics
▶ Ontology
▶ RDF
▶ RDF Schema
▶ Semantic Web

Recommended Reading

1. Baader F., Calvanese D., McGuinness D.L., Nardi D., and Patel-Schneider P.F. (eds.). The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge, UK, 2003.
2. Horrocks I., Patel-Schneider P.F., and van Harmelen F. From *SHIQ* and RDF to OWL: the making of a web ontology language. J. Web Semant., 1(1):7–26, 2003.
3. World Wide Web Consortium. Web Ontology Language (OWL). W3C Recommendation. Available at: http://www.w3.org/2004/OWL/.

P/FDM

Peter M.D. Gray
University of Aberdeen, Aberdeen, UK

Definition

P/FDM [5–7] integrated a functional data model with the logic programming language Prolog for general-purpose computation. The data model can be seen as an Entity-Relationship diagram with sub-types, much like a UML Class Diagram. The idea was for the user to be able to define a computation over objects in the diagram, instead of just using it as a schema design aid. Later versions of P/FDM included a graphic interface [2,4] to build queries in DAPLEX syntax by clicking on the diagram and filling in values from menus.

P/FDM was subsequently extended with constraints [3] and with alternative back-ends to remote databases [6], in the spirit of the original MULTIBASE system.

P/FDM is a vehicle to test a system designed on the principle of Data Independence, whereby Functions represent computations that are expressed in a way that is completely independent of data storage (arrays, lists of objects, indexed files etc.). Functions can be sent across the internet and applied to data in a different form, which was very useful for federated data.

Key Points

In P/FDM, the DAPLEX language was deliberately altered from its original specification so that its semantics could be defined by equivalent *Comprehensions* [1]. In particular, simple assignment operations, if present, could only take place within the innermost loop. Based on this, a very successful early optimizer was written in Prolog by Paton, and used in several bioinformatics applications [6,8].

In P/FDM, data independence is ensured by using a small sparse set of built-in predicates *getentity (entity-type, key, instance-variable)* and *getfunctionval (function-name, argument-variable, value-variable).* The first of these predicates requires that abstract entities be identifiable by unique, possibly compound, scalar keys [7]. The keys help with object identity and are very important when accessing or loading bulk data. This is a feature first used in ADAPLEX and EFDM, though P/FDM extends it to allow composed (single-valued) functions, which aids in forming hierarchical keys.

P/FDM queries are written in DAPLEX and translated into Prolog for evaluation. Backtracking in Prolog is used to give the effect of lazy evaluation in a functional programming language, accessing tuples or objects on demand. For example, assuming the following P/FDM declarations:

```
declare student ->> entity
declare name(student) -> string
key_of student is name

declare course ->> entity
declare cname(course) -> string
key_of course is cname
declare attends(student) ->> course
```

to print all the courses attended by Fred Jones, the DAPLEX query is:

```
for each F in student such that name
(F)="Fred Jones"
   for each C in attends(F)
   print(cname(C));
```

which generates the following Prolog:

```
getentity(student,'Fred Jones',F),
getfunctionval(attends,F,C),
getfunctionval(cname,C,N),write(N),
fail; true.
```

Cross-references

- ▶ Comprehensions
- ▶ Federated Database

▶ Functional Query Language
▶ Query Languages and Evaluation Techniques for Biological Sequence Data

Recommended Reading

1. Embury S.M. User Manual for P/FDM V.9.1. Technical report, Dept. of Computing Science, University of Aberdeen, 1995.
2. Gil I., Gray P.M.D., and Kemp G.J.L. A Visual Interface and Navigator for the P/FDM Object Database. In Proc. User Interfaces to Data Intensive Systems, 1999, pp. 54–63.
3. Gray P.M.D., Embury S.M., Hui K.Y., and Kemp G.J.L. The evolving role of constraints in the functional data model. J. Intell. Inform. Syst., 12:113–137, 1999.
4. Gray P.M.D. and Kemp G.J.L. Capturing quantified constraints in FOL, through interaction with a relationship graph. In Proc. 15th Int. Conf. Knowledge Eng. and Knowledge Management: Ontologies and the Semantic Web, 2006, pp. 19–26.
5. Gray P.M.D., Moffat D.S., and Paton N.W. A Prolog interface to a Functional Data Model database. In Advances in Database Technology, Proc. 1st Int. Conf. on Extending Database Technology, 1988, pp. 34–48.
6. Kemp G.J.L., Dupont J., and Gray P.M.D. Using the functional data model to integrate distributed biological data sources. In Proc. 8th Int. Conf. on Scientific and Statistical Database Management, 1996, pp. 176–185.
7. Paton N.W. and Gray P.M.D. Identification of database objects by key. In Proc. 2nd Int. Workshop on Object-Oriented Database Systems. LNCS 334. Springer, 1988, pp. 280–285.
8. Paton N.W. and Gray P.M.D. Optimising and executing daplex queries using prolog. Comput. J., 33(6):547–555, 1990.

P@n

▶ Precision at n

P2P Database

▶ Structured Data in Peer-to-Peer Systems

Page Cache

▶ Buffer Pool

Page Locking

▶ Concurrency Control – Traditional Approaches

Page Model

▶ Transaction Models – The Read/Write Approach

Page Representations

▶ Document Representations

Paging in Web Search Engines

▶ Web Search Result Caching and Prefetching

PAM (Partitioning Around Medoids)

▶ K-Means and K-Medoids

Parallel and Distributed Data Warehouses

Todd Eavis
Concordia University, Montreal, QC, Canada

Synonyms

Scalable decision support systems High performance data warehousing

Definition

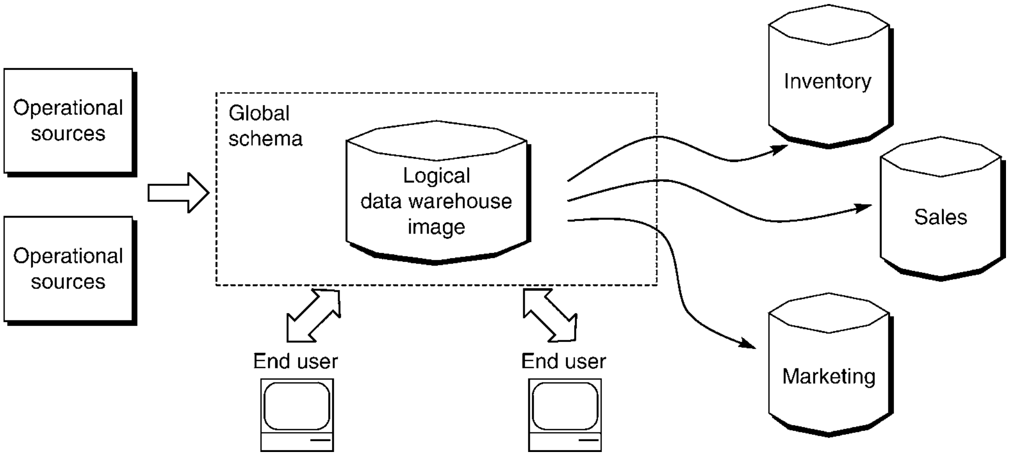
To support the burgeoning data volumes now encountered in decision support environments, *parallel and distributed data warehouses* are being deployed with greater frequency. Having evolved from haphazard and often poorly understood repositories of operational information, the data warehouse itself has become one of the cornerstones of corporate IT architectures. However, as the underlying operational databases grow in size and complexity, so too do the associated data warehouses. In fact, it is not unusual for many corporate or scientific repositories to exceed a terabyte in size, with the largest now reaching 100 TB or more. While processing power has grown significantly during the past decade, the sheer scale of the workload places enormous strain on single CPU data warehousing servers. As a result, some form of data and/or query distribution is often employed in production environments. It is important to note, however, that while contemporary data warehouses are almost always based upon relational DBMS platforms, the unique characteristics and requirements of data warehouse environments often suggest design and optimization choices that are not often employed with general purpose parallelized database systems.

Historical Background

The terms "parallel DW" and "distributed DW" are very often used interchangeably. In practice, however, the distinction between the two has historically been quite significant. Distributed DWs, much like distributed DBs, grew out of a need to place processing logic and data in close proximity to the users who might be utilizing them. In general, multi-location organizations consist of a small number of distinct sites, each typically associated with a subset of the information contained in the global data pool. In the data warehousing context, this has traditionally lead to the development of some form of *federated* architecture. In contrast to monolithic, centralized DWs, federated models are usually constructed as a cooperative coalition of departmental or process specific *data marts*. A simple example is illustrated in Fig. 1. For the most part, design and implementation issues in such environments are similar to those of distributed operational DBMSs [11]. For example, it is important to provide a single transparent conceptual model for the distinct sites and to distribute data so as to reduce the effects of network latency and bandwidth limitations. One unique feature of the distributed data warehouse environment is perhaps the emphasis on integration and consolidation of distributed operational data, a process known in DW terminology as Extract, Transform, and Load (ETL).

With respect to parallelism, research has again been influenced by parallel DBMS projects such as Gamma [3] that were initiated in the mid-to-late 1980s. By the 1990s, it had become clear that commodity-based "shared nothing" databases provided significant advantages over the earlier SMP (Symmetric Multi-Processor) architectures in terms of cost and scalability [4]. Subsequent research therefore focused on partitioning and replication models for the tables of the parallelized DBMS [13]. In general, researchers identified the importance of full *p*-way horizontal striping for large database tables.

Data warehouse researchers have continued to explore the issues related to table partitioning. In addition to complete *p*-way partitioning schemes, full or partial replication (i.e., duplication) of fragments has been investigated [12]. This technique has been further extended by virtualizing partial fragments over



Parallel and Distributed Data Warehouses. Figure 1. A simple federated model illustrating the mapping of a logical global schema on to a series of physically independent data marts.

physically replicated tables [7]. Typically, recent research in the area of table partitioning has exploited the use of "DBMS clusters." Here, rather than constructing a complete, parallel DBMS platform, the parallel system is essentially constructed as a series of commodity DBMS systems, "glued together" with a thin partition-aware wrapper layer.

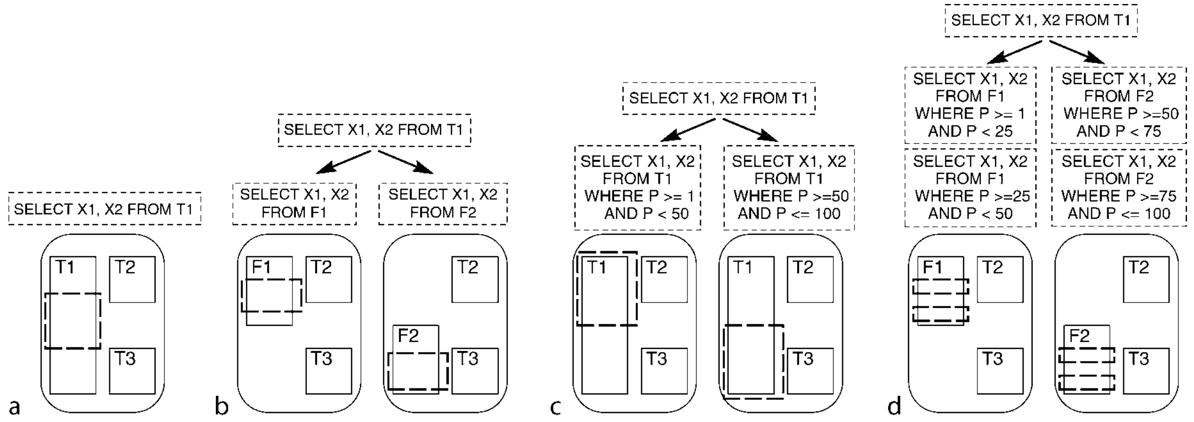
In addition to the continuation of the traditional partitioning work, a second important theme has been the parallelization of state-of-the-art sequential *data cube* generation methods. The data cube has become the primary abstraction for the multi-dimensional analysis that is central to modern data warehousing systems. Cube parallelization efforts have, in fact, taken two forms, one based upon data that is physically represented as array-based storage [8] and the other based upon relational storage [9,2]. In both cases, the complexity of the algorithm and implementation issues has lead to the development of relatively complete DBMS prototypes.

Foundations

Central to data warehousing systems is a denormalized logical *multi-dimensional model* known as the *Star Schema* (the normalized version is referred to as a Snowflake). A *Star Schema* consists of a single, very large *fact* table housing the measurement records associated with a given organizational process. During query processing, this fact table is joined to one or more *dimension* tables, each consisting of a relatively small number of records that define specific business entities (e.g., customer, product, store). A complete data warehouse typically consists of multiple such *Star Schema* designs.

While the *Star Schema* forms the basis of the relational data warehouse, it can be extremely expensive to query the fact table directly, given that it often consists of tens of millions of records or more. Typically, the basic *Star Schema* is augmented with compact, pre-computed aggregates (called *group-bys* or *cuboids*) that can be queried much more efficiently at run-time. This collection of aggregates is known as the data *cube*. Specifically, for a *d*-dimensional space, $\{A_1, A_2, ..., A_d\}$, the *cube* defines the aggregation of the 2^d unique *dimension* combinations across one or more relevant *measure* attributes. In practice, the generation and manipulation of the data *cube* is often performed by a dedicated *OLAP* (online analytical processing) server that runs on top of the underlying relational data warehouse. While the *OLAP* server may utilize either array-based (MOLAP) or table-based (ROLAP) storage, both provide the same, intuitive multi-dimensional representation for the end user.

Given the enormous size of these new data warehouses, some form of parallelism is often employed in production environments. One option is a "shared everything" architecture. Here, designers would likely employ a CC-NUMA system (Cache Coherent Non Uniform Memory Access) that supports a single global memory pool and some form of single virtual disk


SELECT X1, X2 FROM T1
T1
T2
T3
a
SELECT X1, X2 FROM T1
SELECT X1, X2 FROM F1
SELECT X1, X2 FROM F2
F1
T2
T3
T2
F2
T3
b
SELECT X1, X2 FROM T1
SELECT X1, X2 FROM T1 WHERE P >= 1 AND P < 50
SELECT X1, X2 FROM T1 WHERE P >=50 AND P <= 100
T1
T2
T3
T1
T2
T3
c
SELECT X1, X2 FROM T1
SELECT X1, X2 FROM F1 WHERE P >= 1 AND P < 25
SELECT X1, X2 FROM F2 WHERE P >=50 AND P < 75
SELECT X1, X2 FROM F1 WHERE P >=25 AND P < 50
SELECT X1, X2 FROM F2 WHERE P >=75 AND P <= 100
F1
T2
T3
T2
F2
T3
d

Parallel and Distributed Data Warehouses. Figure 2. A three dimensional OLAP space showing all 2^d cuboids and the parent-child relationships between them. Each cell would contain a Total Sales aggregate value.

(e.g., a disk array). Such systems have the advantage that they are relatively easy to administer as the hardware transparently performs much of the "magic." That being said, shared everything designs also tend to be quite expensive and have limited scalability in terms of both the CPU count and the number of available disk heads. In terabyte-scale data warehouse environments, either or both of these constraints might represent a serious performance limitation.

For this reason, many vendors and researchers have turned towards distributed, "shared nothing" platforms for high performance database applications. In fact, the characteristics of DW processing environments make such models particulary attractive. The key distinctions between operational and DW processing include:

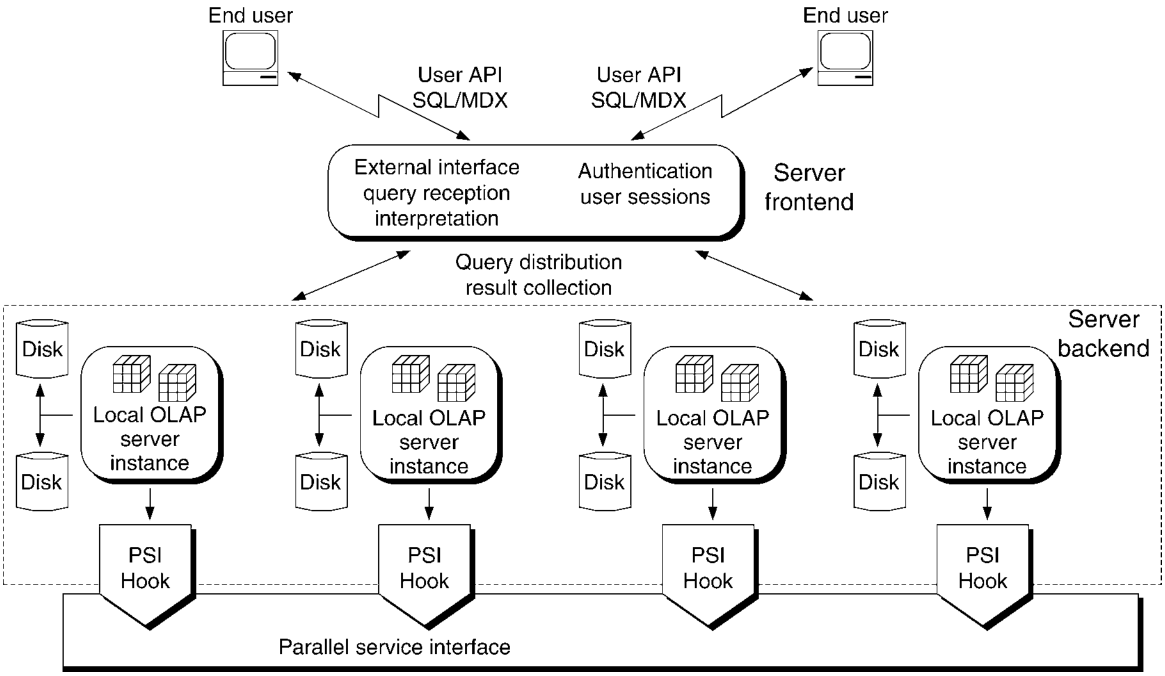
1. Operational systems tend to have high volume query streams. In contrast, DW systems typically process a much smaller number of user queries.
2. Operational queries have *high selectivity*, meaning that they touch relatively few records. Conversely, DW queries are comprehensive in scope, leading to very low selectivity and high I/O and computational load.

Why does this bode well for the use of shared nothing architectures? In large operational environments, high performance can often be achieved through the exploitation of what is often termed a "high throughout engine." Here, improved performance is obtained via *inter-query* parallelism; that is, a stream of small queries is executed concurrently across the parallel/distributed system. However, the low volume/low selectivity combination in DW environments generally argues against the use of inter-query parallelism since such a division of work would likely lead to extensive disk thrashing (i.e., resource conflict). Note, as well, that the use of indexes, a staple in operational systems, is often of relatively little value in DW environments since full table scans are generally more cost effective for low selectivity queries.

As a consequence, the fully distributed, "shared nothing" DBMS model has been particulary attractive for high performance DW practitioners. By partitioning the core fact tables across a large number of independently controlled CPU/disk combinations, shared nothing designs are ideally suited to the exploitation of *intra-query parallelism*. In this case, an individual query q is decomposed and simultaneously executed across the p nodes of the system, with each *partial query* running against approximately $1/p$ records of the partitioned fact table (the small dimension tables are typically replicated on each node). Figure 3b provides a simple example of this technique, contrasting it with the non-partitioned model typically utilized on a single node server (Fig. 3a). Though merging of results may be necessary, it is important to note that while the input partitions may be massive, the output of user-directed analytical DW queries is typically quitesmall. As a result, it should be seen that shared nothing data warehousing excels precisely because it offers tremendous I/O performance while simultaneously requiring only modest inter-node communication [10].

It is against this backdrop that many recent data warehousing partitioning projects have been set. Essentially, there are three forms of DW partitioning. In the first case, the fact tables are physically partitioned in the manner just described. While the performance with this approach can be impressive [15], it is also true that imbalances caused by inherent data skew make effective *a priori* data striping quite challenging. In response, the *virtual* partition model was proposed [1]. Here, as illustrated in Fig. 3c, the fact tables are replicated in full across the nodes of the DBMS cluster. Queries are then decomposed into sub-queries ($1/p$ records per node) that are run against virtual partitions mapped on top of the fully replicated tables. The advantage is that sub-queries may be (i) run against any cluster node and (ii) dynamically migrated to underutilized nodes. The downside is (i) the storage requirement associated with p copies of the primary fact table and (ii) the fact that the commodity DBMS will invoke a full table scan if the partitions are too large. The third partitioning method attempts to combine the best features of the previous methods. In *adaptive virtual partitioning* (AVP) [7], small virtual partitions are layered on top of larger physical partitions. The AVP algorithm dynamically determines the maximum sub-query size that does not invoke a table scan by iteratively probing the commodity DBMS with successively larger queries. The result is a set of re-locatable sub-queries that can generally be answered efficiently without access to the code base of the commodity DBMS systems. A simple illustration is provided in Fig. 3d.

While partitioning is the cornerstone of contemporary high performance data warehouses, it should be clear that such methods represent a sort of "brute



Parallel and Distributed Data Warehouses. Figure 3. Fundamental DW partitioning schemes. (a) Query executed on a single node server that houses a large fact table (T1) and two small dimension tables (T2, T3). (b) Physical partitioning, creating fact table *fragments* (F1, F2). (c) Virtual partitioning on the *clustering attribute* **P** of the replicated fact table. (d) Adaptive virtual partitioning, using fragments and multiple sub-queries.

force" approach to query resolution. Though the response time may indeed represent close to linear speedup as a function of CPU/disk count, the implicit assumption is that full processing of the atomic, transactional data is necessary. This is certainly true for arbitrary or ad-hoc user queries since nothing is known about the possible query format. However, most data warehouses support user interaction through some form of *OLAP* interface. At its core, *OLAP* represents an analytical environment for the intuitive manipulation of the data *cube*. As noted above, there are $O(2^d)$ such views or cuboids. By materializing some (or occasionally all) of these pre-aggregated views, DW/*OLAP* systems can dramatically improve run-time performance on common queries without resorting to massive fact table scans. The trade-off, of course, is that aggregates must be accurately maintained (i.e., updated). In fact, this requirement exerts considerable pressure on the underlying server as data volumes explode in size while, at the same time, update *windows* shrink. It is important to understand, however, that the run-time performance benefit resulting from cube materialization is likely to be significantly larger than the (at best) linear speed-up achieved by a conventional parallel query engine. As such, parallel resources may be more valuable when utilized for the construction and ongoing maintenance of the *OLAP* data structures (e.g., summaries), rather than for brute force query execution.

Because of the extensive computational requirements of *cube* generation in large data warehouses, a number of parallel *cube* construction and querying architectures have in fact been proposed. Note the use of the word "architectures" to indicate comprehensive models that include computation, memory management, and physical storage. In the parallel environment, both array-based (MOLAP) and relational (ROLAP) systems have been explored. With respect to MOLAP parallelism, a distributed memory framework targeting IBM's SP2 was described in [8]. Here, cube construction begins by first equi-partitioning on a single attribute A_1. Using sequential MOLAP techniques, all partial cuboids containing A_1 (exactly $\frac{2^d}{2}$ of the total) are constructed independently on each node. The data set is then re-partitioned on A_2 and the process is repeated. In total, d such rounds are required. The MOLAP mechanism is attractive in the parallel environment as it is well suited to the shared nothing model. Moreover, the parallelized MOLAP cuboids have the potential to provide extremely fast run-time performance (as is generally the case with MOLAP). Having said that, like all MOLAP systems, the sparsity of array-based storage does necessitate complex sparse array compression. In addition, data skew has the potential to generate significant load imbalance during each of the d re-partitioning phases.

Alternatively, parallelism can be exploited in purely relational environments. The cgmCube project, for example, also utilizes state-of-the-art sequential cube construction methods but materializes results in the form of conventional relational tables [2]. Both shared nothing and shared disk platforms are targeted. In this case, the *cube lattice* is physically materialized by first creating a weighted minimum cost spanning tree representation that identifies the most cost effective means by which to materialize all $O(2^d)$ cuboids. Parallelism is supported by decomposing the spanning tree into *p*sub-trees (using a k-min-max graph partitioning algorithm), each of which is computed in its entirety on a single node. The advantage here is that global costing decisions can be employed so as to significantly improve load balancing and to eliminate communication costs (note that the final cuboids can be re-partitioned in any way once the construction algorithm has terminated). Unlike array-based MOLAP systems, ROLAP does not provide implicit indexing, a problem of some significance given that traditional "single dimension" indexes such as the *b-tree* do not work particularly well in multi-dimensional spaces. cgmCube addresses this shortcoming by adding a forest of fully parallelized, multi-dimensional R-tree indexes. Ultimately, cgmCube's algorithmic components are integrated into a fully parallelized server prototype called Sidera that essentially functions as a federation of single node *OLAP* servers [5]. In addition to cube generation and indexing, Sidera adds functionality for caching, selectivity estimation, and the management of *dimensionhierarchies*, as well as fully parallelized sorting and aggregation support. A logical picture of the Sidera server is illustrated in Fig. 3. It should be noted that due to the enormous complexity of parallelized OLAP systems, no direct performance comparison of the MOLAP and ROLAP alternatives has ever been performed.

Finally, one should be aware that commercial parallel DW implementations are also available. Of particular interest in the current context are the dedicated *appliance* style architectures. Here, an integrated, parallel shared nothing hardware/software bundle is tailored specifically to data warehousing workloads and query patterns. Recently, the traditional "high end" DW system provider, Teradata, has been joined by new vendors such as Netezza, DATAllegro, and Greenplum that build upon commodity hardware and open source DBMS software. While each provides proprietary mechanisms for elements such as indexing and partitioning, the DW appliance vendors tend to rely primarily on the aforementioned "brute force" style of parallelism. At present, it does not appear that there is a direct movement of theoretical results from the research community to the industrial sector.

Key Applications

A quick review of the recent database literature indicates that the majority of the current work in high performance databases is actually associated with data warehousing. This should perhaps not be surprising since data warehouses represent some of the largest databases in existence today. In fact, with the emergence of the Internet as a vehicle for both data collection and distribution, one can only expect this trend to continue. So as the *average* size of production DWs pushes past the terabyte limit, parallelism is likely to become an increasingly common theme.

Future Directions

In the short term, one would expect to see continued interest in the exploitation of commodity DBMS systems within loosely federated parallel DBMS clusters. This approach makes a great deal of sense given the considerable maturity of such platforms. Beyond this, OLAP query performance might be further improved by investigating the parallelization of more recent sequential cube methods. The tree-based Dwarf Cube is an obvious target in this regard [14]. We can also expect to see a greater focus on the emerging trend of real time or *near* real time data warehousing, particulary for parallel systems that target large, dynamic data environments.

In the longer term, one possible area for further investigation is the convergence of parallel/distributed data warehouses and the new data centric Grid technologies [6]. To this point in time, the Grid framework has primarily been associated with the concurrent processing of "flat files" rather than highly structured databases. Nevertheless, the Grid model offers interesting opportunities for the distributed, secure, and equitable processing of publicly accessible data repositories.

Experimental Results

Virtually all of the research mentioned in this discussion has relevant experimental evaluations within the associated references. In general, the parallel implementations described above – both partition oriented and OLAP-based – are capable of achieving near linear speedup on contemporary shared nothing parallel systems of 8–32 nodes.

Cross-references

- ▶ Cube
- ▶ Cube Implementations
- ▶ Cube Lattice
- ▶ Data Warehousing Systems: Foundations and Architectures
- ▶ Dimension
- ▶ Measure
- ▶ Multidimensional Modeling
- ▶ OLAP
- ▶ On-Line Analytical Processing
- ▶ Optimization and Tuning in Data Warehouses
- ▶ Query Processing in Data Warehouses
- ▶ Star Schema

Recommended Reading

1. Akal F., Böhm K., and Schek H.-J. OLAP query evaluation in a database cluster: a performance study on intra-query parallelism. In Proc. 6th East European Conf. Advances in Database and Information Systems, 2002, pp. 218–231.
2. Dehne F., Eavis T., and Rau-Chaplin A. The cgmCUBE project: optimizing parallel data cube generation for ROLAP. J. Distr. Parallel Databases, 19(1):29–62, 2006.
3. DeWitt D., Ghandeharizadeh S., Schneider D., Bricker A., Hsaio H., and Rasmussen R. The gamma database machine project. Trans. Knowl. Data Eng., 2(1):44–62, 1990.
4. DeWitt D. and Gray J. Parallel database systems: the future of high performance database systems. Commun. ACM, 35(6):85–98, 1992.
5. Eavis T., Dimitrov G., Dimitrov I., Cueva D., Lopez A., and Taleb A. Sidera: a cluster-based server for online analytical processing. In Proc. Int. Conf. on Grid Computing, High-Performance, and Distributed Applications, 2007.
6. Fiser B., Onan U., Elsayed I., Brezany P., and Tjoa A.M. On-line analytical processing on large databases managed by computational grids. In Proc. 15th Int. Conf. Database and Expert Syst. Appl., 2004, pp. 556–560.
7. Furtado C., Lima A., Pacitti E., Valduriez P., and Mattoso M. Physical and virtual partitioning in OLAP database clusters. In Proc. Int. Symp. on Computer Architecture and High Performance Computing, 2005, pp. 143–150.
8. Goil S. and Choudhary A. High performance multidimensional analysis of large datasets. In Proc. 1st ACM Int. Workshop on Data Warehousing and OLAP, 1998, pp. 34–39.
9. Jin R., Vaidyanathan K., Yang G., and Agrawal G. Communication and memory optimal parallel data cube construction. IEEE Trans. Parallel Distr Syst., 16(12):1105–1119, 2005.
10. Morse S. and Isaac D. Parallel Systems in the Data Warehouse. Prentice-Hall, Englewood Cliffs, 1998.
11. Özsu M.T. and Valduriez P. Principles of distributed database systems 2nd edn. Prentice-Hall, Englewood Cliffs, NJ, 1999.
12. Röhm U., Böhm K., and Schek H.-J. Routing and physical design in a database cluster. In Advances in Database Technology, Proc. 7th Int. Conf. on Extending Database Technology, 2000, pp. 254–268.
13. Scheuermann P., Weikum G., and Zabback P. Data partitioning and load balancing in parallel disk systems. VLDB J., 7(1):48–66, 1998.
14. Sismanis Y., Deligiannakis A., Roussopoulos N., and Kotidis Y. Dwarf: shrinking the PetaCube. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 464–475.
15. Stohr T., Märtens H., and Rahm E. Multi-dimensional database allocation for parallel data warehouses. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 273–284.

Parallel Axes

▶ Parallel Coordinates

Parallel Coordinates

Alfred Inselberg[1,2]
[1]Tel Aviv University, Tel Aviv, Israel
[2]University of California-San Diego, La Jolla, CA, USA

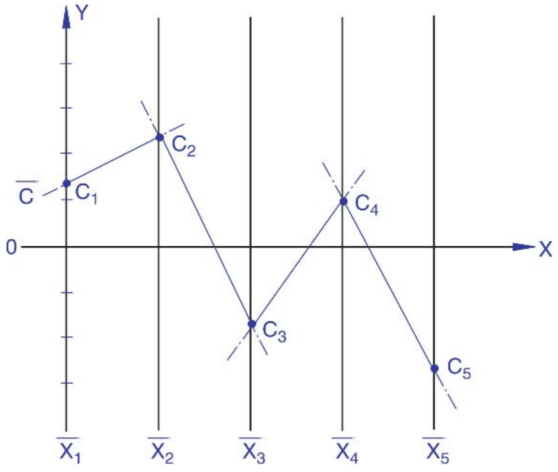
Synonyms

Parallel axes; Parallel Coordinates System (PCS); ||-coords; Multidimensional visualization; Parallel Coordinates Plot (PCP)

Definition

In the plane with *xy*-Cartesian coordinates N copies of the real line labeled $\bar{X}_1$, $\bar{X}_2$,...,$\bar{X}_N$ are placed equidistant and perpendicular to the *x*-axis. They are the axes of the multidimensional system of *Parallel Coordinates* all having the same positive orientation as the *y*-axis. An N-tuple, N-dimensional point, $C = (c_1, c_2,...,c_N)$ is represented by the polygonal line $\bar{C}$ whose N vertices are at the c_i values on each $\bar{X}_i$-axis as shown in Fig. 1. In this way, a 1–1 correspondence between points in N-dimensional space and polygonal lines with vertices on the parallel axes is established. In principle, a large number of axes can be placed and be seen parallel to each other. The representation of points is deceptively simple and much development with additional ideas is needed to enable the visualization of *multivariate relations* or equivalently multidimensional objects.

A dataset with M items has 2^M subsets anyone of which may be the one wanted. With a good data display the fantastic human pattern-recognition can not only cut great swaths searching through this combinatorial explosion but also extract insights from the visual patterns. These are the core reasons for data visualization. With Parallel Coordinates (abbr. ∥-coords) the search for multivariate relations in high dimensional datasets is transformed into a 2-D pattern recognition problem. A geometric classification algorithm based on ∥-coords is presented and applied to a complex dataset. It has low computational complexity providing the classification rule explicitly and *visually*. The minimal set of variables required to state the rule is found and ordered *optimally* by their predictive value. A visual economic model of a real country is constructed and analyzed to illustrate how multivariate relations can be modeled by means of hypersurfaces. Recent results like viewing *convexity in any dimension* and non-orientability (as in the Möbius strip) provide a prelude of what is on the way.



Parallel Coordinates. Figure 1. The polygonal line $\bar{C}$ represents the N-tuple $C = (c_1, c_2, c_3, c_4, c_5)$.

Historical Background

Legend has it that while constructing a proof, Archimedes was absorbed in a diagram when he was killed by a Roman soldier. “Do not disturb my circles” he pleaded as he was being struck by the sword. Visualization flourished in Geometry and Archimedes’ is the first recorded death in defense of visualization. Visual *interaction* with diagrams is interwoven with testing of conjectures and construction of proofs. The tremendous human *pattern recognition* enables interaction for the extraction of insight from images. This essence of visualization is abstracted and adapted into the general problem-solving process to the extent that, a *mental image* of a problem is formed and at times one says *see* when it is meant *understand*.

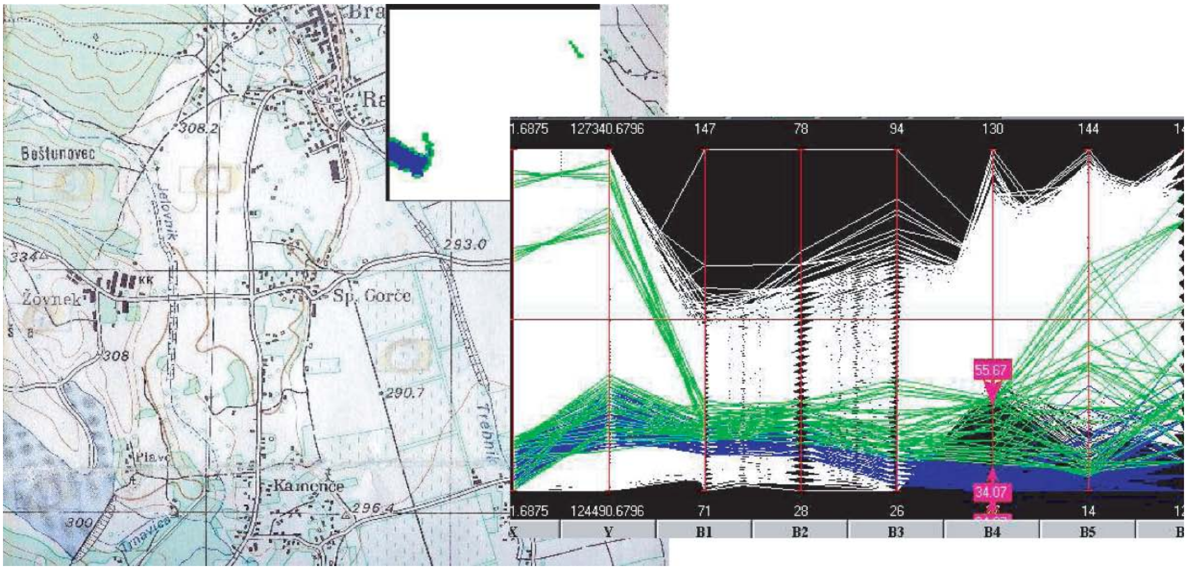
Is there a way to make accurate pictures of multidimensional problems analogous to Descartes coordinate system? What is “sacred” about *orthogonal* axes which use up the plane very fast? After all, in Geometry parallelism rather than orthogonality is the fundamental concept and they are not equivalent for orthogonality requires a prior concept of “angle.” By 1959, while studying Mathematics at the University of Illinois, these thoughts lead to the *multidimensional* coordinate system based on *Parallel Coordinates*. With the encouragement of Professors Cairns and Bourgin (both topologists) basic properties like the *point* ↔ *line* duality were derived. It was not until 1977 when, while teaching a Linear Algebra course, the “challenge” was raised to *show* spaces of dimension higher than 3; parallel coordinates were recalled and their systematic development began. There was early interest and acceptance [6]. The comprehensive report [7] and later [9] layed the foundations of what became the ∥-coords methodology. Collaboration with Dimsdale, (A long-time associate of John von Neuman.) Hurwitz, Boz and Addison ushered to five USA patents (Collision Avoidance Algorithms for Air-Traffic Control, Data Mining and Computer Vision) and applications to Optimization, Process Control [4] and elsewhere. Hurwitz, Adams and later Chomut experimented with multi-query interactive software written in APL for exploratory data analysis (EDA) with ||-coords [5]. There followed [10] on the discovery of structure in data, Gennings et al. [4] on response surfaces based on ||-coords for statistical applications, Wegman using the aforementioned duality promoted the EDA application, Fiorini on Robotics and Hinterberger [14] on comparative multivariate visualization (and earlier on data density analysis). The results of Eickemeyer [2], Hung and Inselberg [5] and Chatterjee were seminal. More recently the work of Ward et al. [15] (Hierachical ||-coords and more), Jones [13] (Optimization), Yang (Association Rules – Databases), Hauser (Categorical Variables and more) and Choi and Heejo Lee (on Intrusion Detection), increased the versatility and sophistication of ||-coords. This list is by no means exhaustive, Shneiderman, Grinstein, Keim, Mihalisin, and others have made significant contributions to data and information visualization.

Foundations

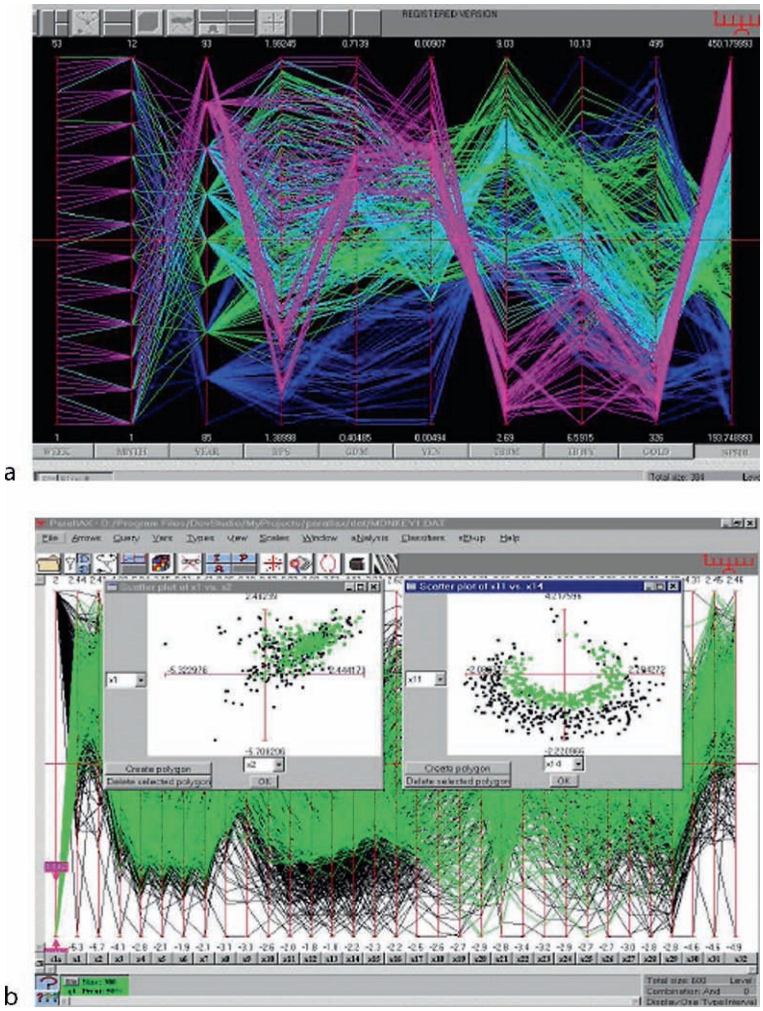
"A picture is worth a thousand words" – But how does one say this with a picture?

The value of data visualization is not seeing "zillions" of objects but rather recognizing *relations* among them. Wonderful successes like Minard's "Napoleon's March to Moscow," Snow's "dot map" and others (see [3]) are *ad hoc* (i.e., one-of-a-kind) and exceptional. Succinct multivariate relations are rarely apparent from *static* displays. With *interactivity* the visual clues in a good data display (*Parallax* MDG's proprietary Data Mining software is used by permission.) can masterfully guide knowledge discovery. Follow up on anything that catches the eyes, gaps, regularities, holes, twists, peaks and valleys, density contrasts like the ones which reveal the water regions in Fig. 2. For the financial data in Fig. 3 (left) *multidimensional contouring* is applied to the axis with *SP500* index values (right) uncovering multivariate relations like high *SP500* low *Gold* and *Interests* correlate with high *Yen* and more. These are examples of two queries and they are others. With Boolean operators compound queries are formed to perform complex tasks. Classification is an important operation in data mining. Powerful geometrical classifiers based on ||-coords [12] have been constructed. An example is shown Fig. 3 (right). The mixing of the two categories is seen on the left plot of the first two parameters. The classifier found the 9 (out of 32) parameters needed to state the rule with 4% error and ordered them according to their predictive value. The two best predictors are plotted on the right showing the separation achieved [12].

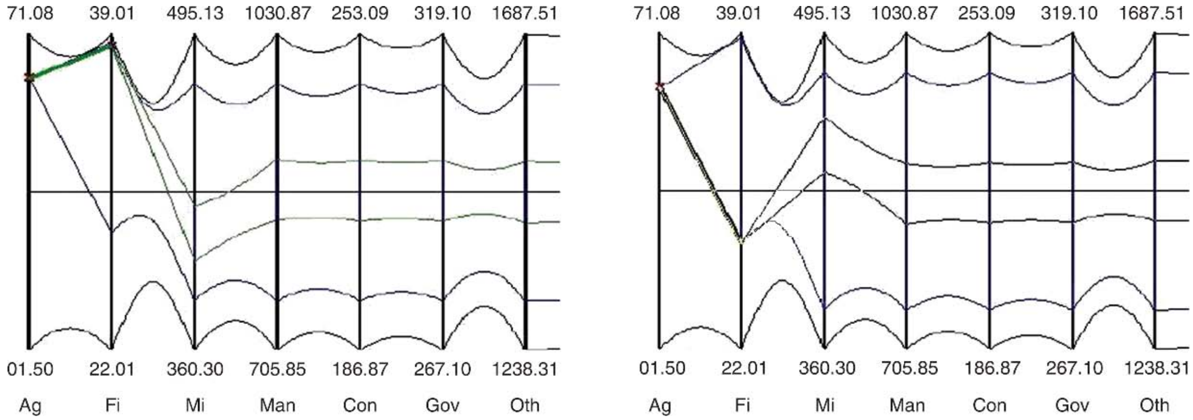
Multivariate relations can be modeled in terms of hypersurfaces – just as a relation between two variables can be represented by a planar region. From a dataset



Parallel Coordinates. Figure 2. Ground emissions measured by satellite on a region of Slovenia (*left*) are displayed on the right. The water and lake's edge are discovered with two queries.

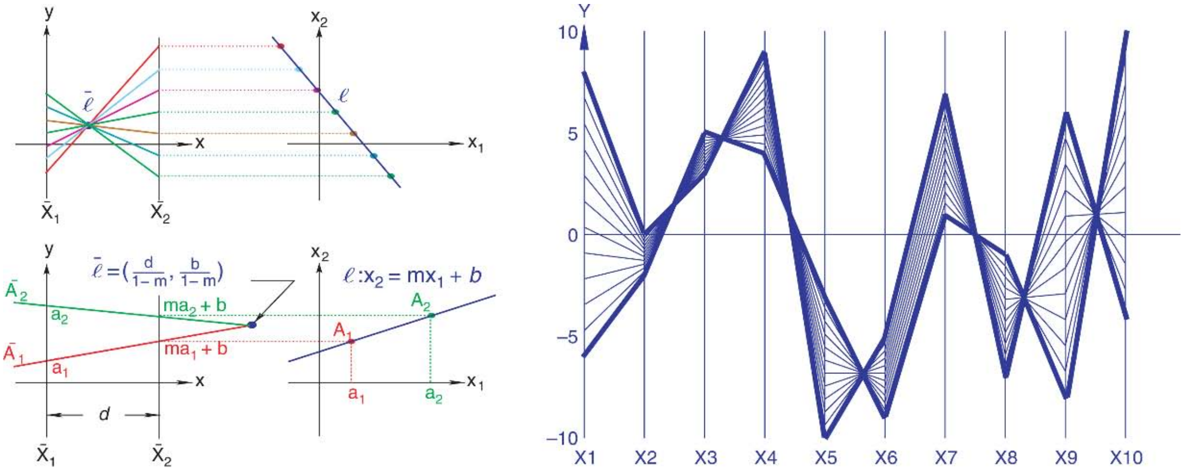


Parallel Coordinates. Figure 3. (a) The *multidimensional-contouring* on financial data reveals multiple interelations. (b) Classification – dataset with 32 parameters and two categories.

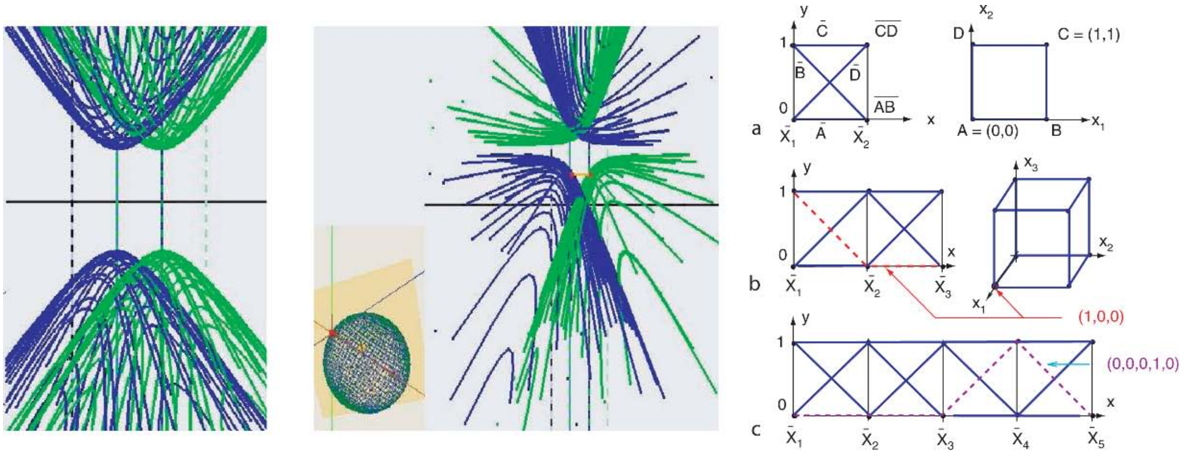


Parallel Coordinates. Figure 4. Hypersurface modeling a country's economy. An interior point represented by a polygonal line depicts a feasible economic policy.

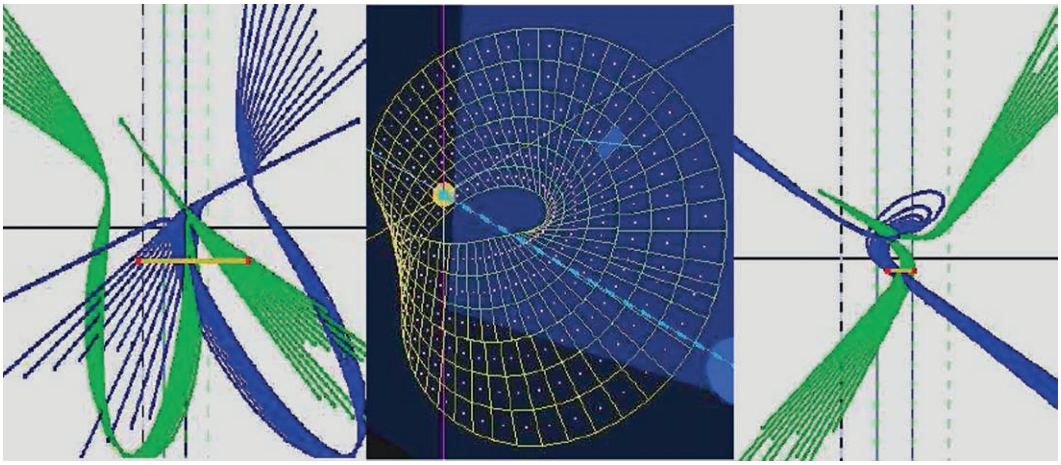
consisting of the outputs of various economic sectors of a real country a visual model of its economy is constructed and shown in Fig. 4 represented by the upper and lower curves. An interior (i.e., a combination of sector outputs) point satisfies all the constraints simultaneously therefore represents a feasible economic policy for that country. Such points can constructed by sequentially choosing variable values within their allowable range. Once a value of the first variable is chosen (in this case the *Agr*icultural output) within its range, the dimensionality of the region is reduced by one. In fact, the upper and lower curves between the second and third axes show the reduced available range of the second variable *Fish*ing and similarly for the remaining variables. Hence the impact of a decision can be seen downstream. In this way it was found that a high value from the available range of *Fish*ing corresponds to very low values of the *Min*ing sector – seen in the right of Fig. 4 and vice versa. This inverse correlation was investigated and found that the two sectors compete for the *same* group of migrating workers. When fishing is doing well, most of them leave the mountains where the mines are located to work on the fishing boats and the reverse. This is an example of ||-coords used for Decision Support and Trade-Off Analysis.



Parallel Coordinates. Figure 5. (*Left*) *point* ↔ *line* duality in 2-D. (*Right*) A N-dimensional line ℓ is represented by $N - 1$ points – the intersections of polygonal representing points of ℓ.



Parallel Coordinates. Figure 6. Convex surfaces represented by $N - 1$ hyperbola-like regions. (left)Sphere in 3-D centered at origin. (*center*) Tranlated sphere *translation* ↔ *rotation* duality (*right*) cube and hypercube in 5-D repeating pattern.



Parallel Coordinates. Figure 7. Möbius strip and its representation for two orientations.

Key Applications

There are numerous others like GIS, Process Control, Trading & Financial Analysis the visualization and analysis of multivariate/multidimensional problems in general.

Future Directions

Though some refer to ‖-coords as a "plot" it is actually *a coordinate system* where the goal, since its inception, is to concentrate N-dimensional relational information into *patterns* the earliest being that N-dimensional lines are represented by $N - 1$ points with *two indices* Fig. 5. Hyperplanes are represented by $N - 1$ points with N indices, and $N - 1$ regions with N indices represent a surface – i.e., the tangent planes enveloping it. Figures 6 and 7 show recent breakthroughs where *convexity in any dimension* and *non-orientability* can be seen. Work is progressing on transforming multivariate relations in datasets into planar patterns completely eliminating display clutter [11].

Cross-references

- ▶ Abstraction
- ▶ Association Rule Mining on Streams
- ▶ Business Intelligence
- ▶ Classification
- ▶ Clustering
- ▶ Cluster Visualization
- ▶ Comparative Visualization
- ▶ Curse of Dimensionality
- ▶ Data Mining
- ▶ Data Visualization
- ▶ Data Warehouse
- ▶ Dimension
- ▶ Dimension Reduction Techniques for Clustering
- ▶ Dimensionality Reduction
- ▶ Dynamic Graphics
- ▶ Event Detection
- ▶ Exploratory Data Analysis
- ▶ Feature Selection for Clustering
- ▶ Geographic Information System
- ▶ High Dimensional Indexing
- ▶ Individually Identifiable Data
- ▶ Information Extraction
- ▶ Interface
- ▶ Machine Learning in Computational Biology
- ▶ Mining of Chemical Data
- ▶ Model Management
- ▶ Multidimensional Data Formats
- ▶ Multidimensional Modeling
- ▶ Multidimensional Scaling
- ▶ Multivariate Visualization Methods
- ▶ OLAP
- ▶ Parallel Visualization
- ▶ Principal Component Analysis
- ▶ Process Optimization
- ▶ Query Evaluation Techniques for Multidimensional Data
- ▶ Range Query
- ▶ Scientific Visualization
- ▶ Spatial Data Analysis
- ▶ Spatial Data Mining
- ▶ Spatio-Temporal Data Mining
- ▶ Temporal Data Mining
- ▶ Temporal Dependencies

- ▶ Visual Analytics
- ▶ Visual Association Rules
- ▶ Visual Classification
- ▶ Visual Clustering
- ▶ Visual Interfaces
- ▶ Visualization Techniques for Scientific Databases
- ▶ Visualizing Quantitative Data
- ▶ What-if Analysis

Recommended Reading

1. Chomut T. Exploratory data analysis in parallel coordinates. M. Sc. Thesis, Department of Computer Science, UCLA, 1987.
2. Eickemeyer J. Visualizing p-Flats in N-Space Using Parallel Coordinates. Ph.D. Thesis, Department of Computer Science, UCLA, 1992.
3. Friendly M. et al. Milestones in Thematic Cartography. www.math.yorku.ca/scs/SCS/Gallery/milestones/, 2005.
4. Gennings C., Dawson K.S., Carter W.H., and Myers R.H. Interpreting plots of a multidimensional dose-response surface in parallel coordinates. Biometrics, 46:719–35, 1990.
5. Hung C.K. and Inselberg A. Parallel Coordinate Representation of Smooth Hypersurfaces. USC Tech. Report # CS-92-531, Los Angeles, 1992.
6. Inselberg A. N-dimensional coordinates. In Proc. of IEEE Conf. Picture Data Description, 1980.
7. Inselberg A. N-dimensional graphics, LASC Tech. Rep. G320-2711, IBM, 1981.
8. Inselberg A. Intelligent instrumentation and process control. In Proc. Second IEEE Conf. on AI Application, 1985, pp. 302–307.
9. Inselberg A. The plane with parallel coordinates. Visual Computer, 1:69–97, 1985.
10. Inselberg A. Discovering multi-dimensional structure with parallel coordinates (*invited paper*). In Proc. ASA.– Stat. Graphics *1*–16, 1989.
11. Inselberg A. Parallel Coordinates : *VISUAL* Multidimensional Geometry and its Applications. Springer, 2009.
12. Inselberg A. and Avidan T. The *Automated* Multidimensional Detective. In Proc. IEEE Information Visualization, 1999, pp. 112–119.
13. Jones C. Visualization and optimization. Kluwer Academic, Boston, 1996.
14. Schmid C. and Hinterberger H. Comparative Multivariate Visualization Across Conceptually Different Graphic Displays, In Proc. 6th Int. Conf. on Scientific and Statistical Database Management, 1994.
15. Ward M.O. XmdvTool: integrating multiple methods for visualizing multivariate data. In Proc. IEEE Conf. on Visualization, 1994, pp. 326–333.

Parallel Coordinates Plot (PCP)

- ▶ Parallel Coordinates

Parallel Coordinates System (PCS)

- ▶ Parallel Coordinates

Parallel Data Placement

Patrick Valduriez
INRIA, LINA, Nantes, Cedex, France

Synonyms

Multiprocessor data placement

Definition

Parallel data placement refers to the physical placement of the data in a multiprocessor computer in order to favor parallel data access and yield high-performance. Most of the work on data placement has been done in the context of the shared-nothing architecture. Data placement in a parallel database system exhibits similarities with data fragmentation in distributed databases since fragmentation yields parallelism. However, there is no need to maximize local processing (at each node) since users are not associated with particular nodes and load balancing is much more difficult to achieve in the presence of a large number of nodes.

The main solution for parallel data placement is a variation of horizontal fragmentation, called *partitioning*, which divides database relations into partitions, each stored at a different disk node. There are three basic partitioning strategies: round-robin, hashing, and interval. Furthermore, to improve availability, replication of partitions is needed.

Historical Background

Parallel data placement was proposed in the early 1980s for the first software-oriented parallel database systems. Instead of proposing expensive changes to the disk technology as in database machines with filtering devices, the main idea was to distribute the data onto multiple (smaller) disks and parallelize the I/O bandwidth which could be better consumed by multiple processors.

Data partitioning was first proposed for shared-nothing architectures which are a natural evolution of distributed database architectures. The same basic idea has been also used (with different partitioning strategies) for the Redundant Arrays of Inexpensive Disks (RAID) to build powerful disks out of many smaller ones.

Most of the work in parallel data placement has been done in the context of the relational model with relation partitioning as the main strategy. Extensions have been proposed for object-oriented or semi-structured (XML) data by partitioning collections of objects or elements.

Foundations

Most of the work on data placement has been done in the context of the shared-nothing architecture which is the most general architecture. However, data placement is also important in shared-memory and shared-disk architectures since there can be multiple disks. Thus, the data placement techniques designed for shared-nothing can also be used, sometimes in a simplified form, to other architectures.

Data placement in a parallel database system exhibits similarities with data fragmentation in distributed databases. An obvious similarity is that fragmentation can be used to increase parallelism. In what follows, the terms partitioning and partition are used instead of horizontal fragmentation and horizontal fragment, respectively, to contrast with the alternative approach which clusters all data of each relation at one node. Vertical fragmentation can also be used to increase parallelism and load balancing much as in distributed databases. Another similarity is that queries should be executed as much as possible where the data reside. However, there are two important differences with the distributed database approach. First, there is no need to maximize local processing (at each node) since users are not associated with particular nodes. Second, load balancing is much more difficult to achieve in the presence of a large number of nodes. The main problem is to avoid resource contention, which may result in the entire system thrashing (e.g., one node ends up doing all the work while the others remain idle). Since queries are executed where the data reside, data placement is a critical performance issue.

Data placement must be done to maximize system performance, which can be measured by combining the total amount of work done by the system and the response time of individual queries. Maximizing response time (through intra-query parallelism) may result in increased total work due to communication overhead. For the same reason, inter-query parallelism results in increased total work. On the other hand, clustering all the data necessary to a query minimizes communication and thus the total work done by the system in executing that query. In terms of data placement, the following trade-off exists: maximizing response time or inter-query parallelism leads to partitioning, whereas minimizing the total amount of work leads to clustering. This problem is addressed in distributed databases in a rather static manner. The database administrator is in charge of periodically examining fragment access frequencies, and when necessary, moving and reorganizing fragments.

A solution to data placement is full partitioning, whereby each relation is horizontally fragmented across all the nodes in the system. There are three basic strategies for data partitioning: round-robin, hash, and range partitioning.

1. *Round-robin partitioning* is the simplest strategy, it ensures uniform data distribution. With n partitions, the ith tuple in insertion order is assigned to partition (i mod n). This strategy enables the sequential access to a relation to be done in parallel. However, the direct access to individual tuples, based on a predicate, requires accessing the entire relation.
2. *Hash partitioning* applies a hash function to some attribute which yields the partition number. This strategy allows exact-match queries on the selection attribute to be processed by exactly one node and all other queries to be processed by all the nodes in parallel.
3. *Range partitioning* distributes tuples based on the value intervals (ranges) of some attribute. In addition to supporting exact-match queries as with hashing, it is well-suited for range queries. For instance, a query with a predicate "A between a_1 and a_2 may be processed by the only node(s) containing tuples whose A value is in the range [a_1 and a_2]." However, range partitioning can result in high variation in partition size.

Full partitioning generally yields better performance than clustering relations on a single (possibly very large) disk. Although full partitioning has obvious performance advantages, highly parallel execution may cause a serious performance overhead for complex queries involving joins. Furthermore, full partitioning is not appropriate for small relations that span a few disk blocks. These drawbacks suggest that a compromise between clustering and full partitioning needs to be found.

A solution is to do data placement by variable partitioning, where the degree of partitioning, in other words, the number of nodes over which a relation is partitioned, is a function of the size and access frequency of the relation. This strategy is much more involved than either clustering or full partitioning because changes in data distribution may result in reorganization. For example, a relation initially placed across eight nodes may have its cardinality doubled by subsequent insertions, in which case it should be placed across 16 nodes.

In a highly parallel system with variable partitioning, periodic reorganizations for load balancing are essential and should be frequent unless the workload is fairly static and experiences only a few updates. Such reorganizations should remain transparent to compiled queries that run on the database server. In particular, queries should not be recompiled because of reorganization. Therefore, the compiled queries should remain independent of data location, which may change rapidly. Such independence can be achieved if the run-time system supports associative access to distributed data. This is different from a distributed DBMS, where associative access is achieved at compile time by the query processor using the data directory.

A serious problem in data placement is dealing with skewed data distributions which may lead to non-uniform partitioning and hurt load balancing. Range partitioning is more sensitive to skew than either round-robin or hash partitioning. A solution is to treat non-uniform partitions appropriately, e.g., by further fragmenting large partitions.

Another important function is data replication for high availability. The simple solution is to maintain two copies of the same data, a primary and a backup copy, on two separate nodes. This is the mirrored disks architecture promoted by many computer manufacturers. However, in case of a node failure, the load of the node having the copy may double, thereby hurting load balancing. To avoid this problem, an interesting solution is Teradata's interleaved partitioning which partitions the backup copy on a number of nodes. In failure mode, the load of the primary copy gets balanced among the backup copy nodes. But if two nodes fail, then the relation cannot be accessed thereby hurting availability. Reconstructing the primary copy from its separate backup copies may be costly. In normal mode, maintaining copy consistency may also be costly. A solution to this problem is Gamma's chained partitioning which stores the primary and backup copy on two adjacent nodes. The main idea is that the probability that two adjacent nodes fail is much less than the probability that any two nodes fail. In failure mode, the load of the failed node and the backup nodes are balanced among all remaining nodes by using both primary and backup copy nodes. In addition, maintaining copy consistency is cheaper.

Key Applications

Parallel data placement has been primarily used by parallel database systems on multiprocessor computers. Data partitioning is also used as a basic technique for dealing with very large volumes of data in different environments such as database clusters, data grids, search engines, document management systems, and P2P systems.

Cross-references

► Distributed Databases

Recommended Reading

1. Copeland G.P., Alexander W., Boughter E.E., and Keller T.W. Data placement in bubba. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988, pp. 99–108.
2. DeWitt D.J. and Gray J. Parallel database systems: the future of high performance database systems. Commun. ACM, 35(6):85–98, 1992.
3. Mehta M. and DeWitt D.J. Data placement in shared-nothing parallel database systems. VLDB J., 6(1):53–72, 1997.
4. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 3rd edn. Prentice Hall, 2009.
5. Valduriez P. and Pacitti E. Parallel database systems. In Handbook of Database Technology, J. Hammer, M. Scheider (eds.). CRC Press, Boca Raton, FL, 2007.

Parallel Database

► Distributed Architecture

Parallel Database Management

Patrick Valduriez
INRIA, LINA, Nantes, Cedex 3, France

Synonyms

Multiprocessor database management

Definition

Parallel database management refers to the management of data in a multiprocessor computer and is done by a parallel database system, i.e., a full-fledge DBMS implemented on a multiprocessor computer. The basic principle employed by parallel DBMS is to partition the data across multiprocessor nodes, in order to increase performance through parallelism and availability through replication. This enables supporting very large databases with very high query or transaction loads.

Parallel database systems can exploit distributed database techniques. In particular, database partitioning is somewhat similar to database fragmentation. Essentially, the solutions for transaction management, i.e., distributed concurrency control, reliability, atomicity, and replication, can be reused. However, the critical issues for parallel database systems are data placement, parallel query processing (including optimization and execution) and load balancing. These issues are much more difficult than in distributed DBMS because the number of nodes may be much higher. Furthermore, the interconnection network of a multiprocessor system provides better opportunities for improving performance than a general-purpose network.

Historical Background

The performance objective of parallel database systems was also that of the database machines in the 1980s. The problem faced by conventional database management on a monoprocessor computer has long been known as "I/O bottleneck," induced by high disk access time. Database machine designers tackled this problem through special-purpose hardware, e.g., by introducing data filtering devices within the disk heads. However, this approach did not succeed because of a poor cost/performance ratio compared to the software solution which can easily benefit from progress in standard hardware technology (processor, RAM, disk). However, the idea of pushing database functions closer to the disk has received renewed interest with the introduction of general-purpose microprocessors in disk controllers, thus leading to intelligent disks.

An important result of database machine research, however, is in the general solution to the I/O bottleneck which can be summarized as increasing the I/O bandwidth through parallelism. If a database is partitioned, e.g., by horizontal fragmentation of the relational tables, across a number *d* of disks, then disk throughput can be increased by accessing the *d* disks in parallel. The same principle has been applied to Redundant Arrays of Inexpensive Disks (RAID) which build a powerful disk with many small disks.

The main memory database solution which maintains the entire database in main memory to avoid disk accesses, is complementary rather than alternative to parallel database systems. In particular, the "memory access bottleneck" induced by high memory access time relative to processor speed can also be tackled using parallel database techniques. In other words, a parallel database system can use main memory database nodes.

Foundations

Parallel database system designers strive to develop software-oriented solutions in order to exploit the various kinds of multiprocessor systems which are now available. This can be achieved by extending distributed database technology, for example, by partitioning the database across multiple disks so that much inter- and intra-query parallelism can be obtained. This can lead to significant improvements in both response time and throughput (number of transactions per time unit).

A parallel database system can be loosely defined as a DBMS implemented on a multiprocessor computer. This definition includes many alternatives ranging from the porting of an existing DBMS, which may require only rewriting the operating system interface routines, to a sophisticated combination of parallel processing and database system functions into a new hardware/software architecture. Thus, there is a trade-off between portability (to several platforms) and efficiency. The sophisticated approach is better able to fully exploit the opportunities offered by a multiprocessor at the expense of portability. Interestingly, this gives different advantages to computer manufacturers and software vendors.

The main objectives of a parallel database system are the following:

High-performance. This can be obtained through several complementary solutions: data-based parallelism, query optimization, and load balancing. Parallelism can increase throughput, using inter-query parallelism, and decrease transaction response times, using intra-query parallelism. Load balancing is the ability of the system to divide a given workload equally among all processors.

Depending on the parallel system architecture, it can be achieved by static physical database design or dynamically at run-time.

High-availability. Because a parallel database system consists of many redundant components, it can well increase data availability and fault-tolerance. Replicating data at several nodes is useful to support failover, a fault-tolerance technique which enables automatic redirection of requests from a failed node to another node which stores a copy of the data, thereby providing nonstop operation.

Extensibility. In a parallel system, accommodating increasing database sizes or increasing performance demands (e.g., throughput) should be easier. Extensibility is the ability of smooth expansion of the system by adding processing and storage power to the system. Ideally, the increase in the configuration of the parallel database system (i.e., more nodes) should speed up existing workloads or scale up to larger databases.

The functions supported by a parallel database system are the same as in a typical DBMS. The differences, though, have to do with implementation of these functions which must now deal with parallelism, data partitioning and replication, and distributed transactions. Depending on the architecture, a processor node can support all (or a subset) of these functions. In the early database machines for instance, some nodes were specialized with hardware to implement some functions, e.g., data filtering.

The way the main hardware elements, i.e., processors, main memory, and disks, are connected (through some interconnection network) has a major impact on the design of a parallel database system. The term processor is used in the general sense of central processing unit (CPU). However, the processor itself can be made of several core processors integrated in a single chip, i.e., a multi-core processor, and perform instruction-level multithread parallelism. But the parallelism being discussed here is much higher-level (query- or operator-level) so processors (e.g., multi-core processors) are simply considered as black-box components accessing other resources (main memory, disk, network). Depending on how main memory or disk is shared, three basic architectures are obtained: shared-memory, shared-disk and shared-nothing.

In the shared-memory architecture, any processor has access to the entire main memory, and thus to all the disks, through the interconnection network. The network is typically very fast (e.g., a high-speed bus or a cross-bar switch) and redundant to avoid any unavailability. All the processors are under the control of a single operating system. Shared-memory makes it easy to build a parallel database system because the operating system deals with load balancing. It is also very efficient since processors can communicate via the main memory. However, the complexity and cost of the interconnection network hurt extensibility which is limited to a few tens of processors. Most parallel database system vendors provide support for shared-memory.

In the shared-disk architecture, any processor has access to any disk unit through the interconnection network but exclusive (non-shared) access to its main memory. Each processor-memory node is under the control of its own copy of the operating system. An important function that is needed is cache coherency which allows different nodes to cache a consistent disk page. This function is hard to support and requires some form of distributed lock management. Shared-disk has better extensibility than shared-memory since the main memory is distributed. However, scalability depends on the efficiency of the cache coherency mechanism. Migrating from a centralized system to shared-disk is relatively straightforward since the data on disk need not be reorganized. Shared-disk is the main architecture used by Oracle for its parallel database system.

The shared-nothing architecture is fully-distributed: each node is made of processor, main memory and disk and communicates with other nodes through the interconnection network.

Similar to shared-disk, each processor-memory-disk node is under the control of its own copy of the operating system. Then, each node can be viewed as a local site (with its own database and software) in a distributed database system. Therefore, most solutions designed for distributed databases such as database fragmentation (called partitioning in parallel databases), distributed transaction management and distributed query processing may be reused. Using a fast interconnection network, it is possible to accommodate large numbers of nodes. The major advantages of shared-nothing over shared-memory or shared-disk are those of distributed systems: relatively low cost, extensibility and availability. However, it is much more complex to manage as physical data portioning is necessary. Shared-nothing has been adopted by the major DBMS vendors (except Oracle) for their high-end parallel database systems.

Various possible combinations of these three basic architectures are possible to obtain different trade-offs between cost, performance, extensibility, availability, etc. Hierarchical architectures typically combine the three architectures using a shared-nothing design where each node can be shared-memory or shared-disk and communicates with other nodes through the interconnection network.

Key Applications

Parallel database systems are used to support very large databases (e.g., tens or hundreds of terabytes). Examples of applications which deal with very large databases are e-commerce, data warehousing, and data mining. Very large databases are typically accessed through high numbers of concurrent transactions (e.g., performing on-line orders on an electronic store) or complex queries (e.g., decision-support queries). The first kind of access is representative of On Line Transaction Processing (OLTP) applications while the second is representative of On Line Analytical Processing (OLAP) applications. Although both OLTP and OLAP can be supported by the same parallel database system (on the same multiprocessor), they are typically separated and supported by different systems to avoid any interference and ease database operation.

Cross-references

▶ Distributed Databases

Recommended Reading

1. DeWitt D.J. and Gray J. Parallel database systems: the future of high performance database systems. Commun. ACM, 35 (6):85–98, 1992.
2. Özsu T. and Valduriez P. Distributed and parallel database systems – Technology and current state-of-the-art. ACM Comput. Surv., 28(1):125–128, 1996.
3. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 3rd edition. Prentice Hall, 2009.
4. Valduriez P. Parallel database systems: the case for shared-something. In Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 460–465.
5. Valduriez P. and Pacitti E. Parallel Database Systems. In Handbook of Database Technology, J. Hammer, M. Scheider (eds.). CRC Press, Boca Raton, FL, 2007.

Parallel Distributed Processing

▶ Neural Networks

Parallel Hash Join, Parallel Merge Join, Parallel Nested Loops Join

Goetz Graefe
Hewlett-Packard Laboratories, Palo Alto, CA, USA

Synonyms

Parallel join algorithms

Definition

These join algorithms are parallel versions of the traditional serial join algorithms. They are designed to exploit multiple processors on a network, within a machine, or even within a single chip.

Key Points

Parallel join algorithms are based on the traditional serial join algorithms, namely (index) nested loops join, merge join, and (hybrid) hash join. The goal of parallel execution is to reduce the input sizes in each processing element, thus reducing the time for query completion even at the expense of increasing overall query execution effort due to data movement. Ideally, parallel join algorithms exhibit linear speed-up and linear scale-up.

Parallel join algorithms are orthogonal to pipelining among join operations in a complex query execution plan. Even a query with a single join can benefit from a parallel join algorithm. The essence of pipelining (and also of "bushy parallelism" in appropriate query execution plans) is to execute different operations with different predicates concurrently, whereas the essence of parallel join algorithms is that a single logical operation exploits multiple processors or processing cores. The conceptual foundations of parallel join algorithms are that database query processing manipulates sets of records, that sets can be divided into disjoint subsets, and that query results based on one record can be computed independently of results based on other records.

Parallel join algorithms apply not only to inner joins but also to semijoins, outer joins, and set operations such as intersection, union, and difference. Thus, they apply to query execution techniques such as index intersection for conjunctive predicates and the star join techniques that are generalizations traditional binary joins.

For a parallel join algorithm, input records are partitioned or replicated such that each result record is produced exactly once. The usual choices are to partition both inputs or to partition one input and to replicate the other. For parallel joins without any equality predicate, the processing elements can be organized in a rectangular grid, one input partitioned over rows of the grid and replicated within each row, and the other join input partitioned over columns of the grid and replicated within each column. These choices mirror schemes in which database records may be stored partitioned or replicated.

Partitioning schemes include range partitioning, hash partitioning, and hybrid schemes such as range partitioning of hash values or hashing (identifiers of) key ranges. For intermediate query results, hash partitioning is simple yet reasonably robust against skewed key distributions.

All of the join algorithms can reduce data transfer efforts using semijoin reduction or its heuristic approximation, bit vector filtering or Bloom filters. Hash join can benefit most obviously due to its separate build and probe phases, with the bit vector filter populated during the build phase and exploited during the probe phase. Merge join can exploit bit vector filtering if the filter can be populated prior to a stop-and-go operation such as sort, and can exploit it with the most gain if the other input also must be processed by an expensive operation such as a sort.

Symmetric partitioning means that both inputs are partitioned. Ideally, the inputs are stored partitioned in such a way that join processing does not incur any cost or delay for partitioning. This is often called "local indexing" or "table partitioning" when it refers to indexes of the same table and "aligned partitioning" when it refers to separate tables. Aligned partitioning often coincides with frequent join operations, foreign key constraints, and complex objects on the conceptual level. The opposites are "global indexing," "index partitioning," and "non-aligned partitioning."

If one of the join inputs is very large, and if the degree of parallelism is only moderate, it might be less expensive to replicate the small input than to partition the large input. An appropriate cost calculation needs to consider record counts, record sizes, communication costs per message or per byte, the available memory allocation for each thread, etc. A first approximation compares the quotient of the two input sizes with the degree of parallelism.

Cost calculation during compile-time query optimization can focus either on the average cost in each of the parallel processing elements or on the maximal cost. The former metric focuses on overall system throughput in multi-user systems, whereas the latter metric focuses on the query elapsed time as perceived by a single user or application.

Cross-references

► Parallel Query Execution Algorithms

Recommended Reading

1. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73–170, 1993.
2. Mishra P. and Eich M.H. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, 1992.

Parallel Join Algorithms

► Parallel Hash Join, Parallel Merge Join, Parallel Nested Loops Join

Parallel Query Execution Algorithms

GOETZ GRAEFE
Hewlett-Packard Laboratories, Palo Alto, CA, USA

Synonyms

Partitioned query execution; Partitioning

Definition

Parallel query execution algorithms employ multiple threads that together execute a single query execution plan, with the goal of increased processing bandwidth and decreased response time. Multiple forms of parallelism may be employed in isolation or in combination. Most parallel query execution algorithms are serial algorithms executed in multiple threads, usually with no or only minor adaptations to function correctly or efficiently in the multi-threaded environment.

Historical Background

Set-oriented query languages and the relational algebra have given rise to query optimization and parallel database processing almost from their beginning in the 1970s. They represented a new complexity or liability in database software design as non-procedural languages and physical data independence prevent users from specifying query processing steps and thus require an automatic system component to provide plans for data access and for query execution. At the same time, they created a new opportunity as to prior database systems and their interfaces did permit query rewrite techniques or parallel execution.

The last 30 years have seen many successes and many failures in parallel query execution algorithms. Parallel versions of naïve nested loops join were a failure, because even hardware acceleration cannot compete with well-chosen indexes, whether those are persistent indexes supporting index nested loops join or temporary indexes in form hash tables in hash join. Parallel set operations, including non-indexed selection, join, grouping, and sorting, have been great successes, because they can provide both good speed-up and scale-up. Parallelism is sometimes associated with specific set operations, e.g., hash join instead of all joins, but almost all join algorithms benefit quite similarly if designed and implemented with similar care.

Foundations

In principle, parallel query execution algorithms are traditional sequential algorithms with multiple threads executing different operations or executing the same operation on different data. The relational algebra of sets permits pipelining between operations, partitioning of stored data and of intermediate results, and bushy execution in complex query execution plans. Individual parallel algorithms exploit partitioning but also overcome the issues introduced by partitioning and parallelism.

The value and success of parallel query execution is often expressed as speed-up and scale-up. Speed-up compares execution times of serial and parallel execution for a constant data volume. Linear speed-up is considered ideal; it is achieved if the ratio of execution times is equal to the degree of parallelism. Scale-up compares execution times for a constant ratio of data volume and degree of parallelism. Linear scale-up is considered ideal; it is achieved if the execution times remain constant. There are, however, some effects that permit super-linear speed-up. For example, adding nodes to a parallel database machine increases not only processing bandwidth but also available memory, and might thus turn an external sort into an in-memory sort at a high degree of parallelism.

Pipelining

The pipeline between a producer operation and a consumer operation may pass individual records, value packets, pages, or even runs. Value packets within a sorted stream are defined by equal values in the sort columns. As an example for iterators passing run information, consider a sort operation split into run generation and merging; the pipeline between those two components may pass entire runs at-a-time. In addition to data, the pipeline contents may include control records, e.g., artificially created keys design to advance the merge logic further up in the sequence of operations.

More importantly for multi-threaded query execution, the pipeline between threads may be demand-driven or data-driven. Within each thread, iterators commonly implement demand-driven dataflow, which lend themselves well to joins and other binary operations but not to sharing intermediate results or common sub-expressions, which occur much less frequently than joins in query execution plans. Between threads, data-driven dataflow is commonly regarded superior.

On the other hand, if flow control is implemented, the slower one among producer and consumer dictates the overall pipeline flow, and it might not truly be useful to attempt distinguishing demand- and data-driven dataflow. The important aspect is that some "slack" is created that permits the producer to get ahead of the consumer to some reasonable amount without requiring an excessive amount of buffer space. In general, the quotient of buffer space and bandwidth is equal to the interval that can be smoothed out, i.e., the amount of time the producer may get ahead of the consumer or the consumer may fall behind the producer.

Partitioning

Since pipelining with flow control often leads to consumers throttling producers or the other way around, and since pipelining alone permits only a fairly limited degree of parallelism, it is usually combined with partitioning. Partitioning means that the input into a relational algebra operation is divided into disjoint subsets, which are processed by separate threads or processors.

Many partitioning methods are in common use. The basic partitioning methods are random, round-robin, range, and hash. Both random and round-robin partitioning permit perfect load balancing but do not exploit key values and their importance in relational algebra operations such as joins and duplicate elimination. When random partitioning or round-robin partitioning is applied to stored data, neither queries nor updates can be directed to a single location and instead must be processed in all partitions. Range partitioning assigns key ranges (of the partitioning column) to storage or processing sites. For perfect load balancing, accurate quantile information is required. For intermediate results, quantiles must be estimated during query optimization or during an earlier query execution phase. Hash partitioning applies a hash function to determine storage or processing sites. A suitable hash function can achieve very good load balancing, which is why hash partitioning is sometimes associated with linear speed-up and scale-up. On the other hand, even hash partitioning fails if the data distribution suffers from duplicate skew, i.e., the data contain many duplicate key values.

There are many ways to combine these basic partitioning methods. For example, it may be desirable that small range queries are processed at a one or two sites with little overhead and that large range queries are processed by many or all sites with maximum throughput. To achieve both these effects, many small key ranges can be hashed to storage sites. In other cases, one column may be used to hash data items to sites and another column to hash data items to storage locations within each site. This is just one of many multi-dimensional partitioning methods. Note that partitioning among sites, partitioning within sites, and local organization are orthogonal. The possibilities are endless, and almost each vendor claims to have found a new and superior "secret sauce." The most sophisticated systems even offer partitioning on computed columns or expressions.

Partitioning of stored data usually determines where initial query operations execute, including selection and projection as well as preliminary stages of sorting, duplicate elimination, and "group by" operations. Partitioning of stored data may also interact with high availability consideration, e.g., each data item is hashed to two sites using two hash functions, carefully designed to ensure that no data item is hashed to the same site by both hash functions.

For binary operations, "co-location" of the two inputs permits local execution. These binary operations include inner and outer joins, semijoins including "exists" nested queries, and set operations such as intersection, union, and difference. Set operations after scans of non-clustered indexes are common in query execution plans for complex queries, for ad-hoc queries, and for star joins in relational data warehouses using a star schema. If all indexes for a table or view are partitioned in the same way, both set operations and fetch operations can remain local. On the other hand, there are many situations in which it is advantageous to partition each index on its search columns.

If co-location and local execution are not possible, join input data need to be re-partitioned. In most cases, both inputs are partitioned on the columns participating in the join operation's equality predicate. If one input is much smaller than the other, it may be more efficient to broadcast the small input and not move the large input. In some case, in particular non-equality joins, a combination can be applied. In this technique, the processing sites can be thought of as a rectangular matrix, with one input partitioned among rows and broadcast within rows, and the other input partitioned among columns and broadcast within columns. The essence of all partitioning strategies for binary operations such as joins is to ensure that each pair of input rows that may contribute to the join result "meets" at precisely one processing site.

Bushy Execution

In addition to pipelining and partitioning, a third form of parallel query execution is enabled by bushy query execution plans, as opposed to linear query execution plans. In a linear query execution plan, one input of each binary operation always is a stored table. In a bushy query execution plan, both inputs may be intermediate query results. For example, two sort operations may provide the inputs for a merge join. During the sort operations' input phases, the two sort operations can run in parallel. In more complex query execution plans, numerous operations and plan branches may be active concurrently and independently.

Even the simple example with two sort operations and a merge join suffices to demonstrate the substantial difficulties, however. One problem is finding an appropriate degree of parallelism for each operation or branch. A harder problem is assigning appropriate resources such as memory, disk space and bandwidth, etc. The hardest problem is to ensure that concurrent branches produce their results at the right time. In the example, resources may be wasted if one sort operation finishes before the other, because the former will likely hold on to resources such as memory while waiting for the merge join to need its input and thus for the other sort operation to finish.

Specifics of Parallel Algorithms

In order to keep parallel query execution engines modular, traditional serial query execution algorithms are combined with mechanisms for parallel execution, specifically pipelining and partitioning. The interface between traditional serial query execution operations usually takes the form of iterator. Iterator methods support forward-only scans of intermediate results, rewinding, binding parameters as needed for nested iteration and general predicate evaluation, etc. Iterator instances may recursively invoke their input iterators to implement those methods, and by doing so can execute a very complex query execution plan within a single thread.

Mechanisms for parallel query execution can be encapsulated in a special iterator. This design has been used in a number of research prototypes and commercial database systems. Commonly called the "exchange" operation but also known as "river" or "asynchronous table queue," this iterator encapsulates data transfer and flow control, often provides initialization and teardown of threads and communication paths, and batches records in order to reduce inter-process communication. If parallel execution of nested iteration is desired, both data transfer and flow control must work both from producer to consumer for intermediate query results and from consumer to producer for bindings of correlation columns from outer to inner inputs of nested iteration operations.

As the exchange operation or its equivalent encapsulate the basic mechanisms for parallel query execution, the following only indicates how specific parallel algorithms differ from their serial equivalents.

For parallel sorting, after the input data is partitioned as appropriate, the individual threads of a parallel sort operation usually can work entirely independently. If, however, the query optimizer relies on the sort operation not only for the correct sequence of records but also to avoid the Halloween problem by means of phase separation, the individual threads must coordinate their transition from consuming unsorted input records to producing sorted output records. If one of them produces output too early, it might affect the input set of another thread still consuming input and thus create the Halloween problem.

A similar coordination point exists in parallel hash join. A serial hash join might choose between Grace join, which devotes its first phase entirely to partitioning, and hybrid hash join, which performs some join logic even during its first phase. Multiple threads in a parallel hash join might choose differently, e.g., due to different data volumes. Scheduling other operations within the query execution plan might require, however, that all hash join threads follow the same choice. As for parallel sorting with built-in Halloween protection, such a parallel hash join requires a very brief communication and decision phase.

For parallel hash aggregation and duplicate elimination, a common technique is to run the operation twice, once before and once after data partitioning, in cases in which the input data are not yet partitioned on the grouping columns. The goal is to reduce the number of data records that need to be shipped between threads, processes, and processors. Assuming uniform random distribution of input rows prior to re-partitioning, this goal can be achieved if the average group size, i.e., the ratio of row counts in the input and in the output, exceeds the degree of parallelism. A common name for this technique is "local-global aggregation." The initial, local aggregation may be opportunistic, i.e., it performs as much aggregation as can readily be achieved with the available memory, or it may be complete, i.e., it will spill overflow partitions to disk if required. If network transfer is faster than local disk I/O, overflow partitions may be send to their final destination rather than to a local disk.

Sort-based aggregation and duplicate elimination can benefit from the same ideas. The opportunistic variant performs merely run generation and early duplicate elimination prior to data transfer. The full variant performs complete local sort operations and all aggregation and duplicate elimination prior to the data transfer.

Order-preserving and merging re-partitioning operations may experience a deadlock, assuming flow control or bounded buffers in the data exchange operation. The essence of this deadlock is that producers need to send data in the order produced, whereas consumers need to consume data in the appropriate sort order. For example, if producer 1 has data only for consumer 1, and producer 2 has data only for consumer 2, then consumer 1 waits for data from producer 2 in order to advance its merge logic, yet producer 2 waits for consumer 2 to release flow control, etc. There are several possible solutions for this problem. A typical deadlock avoidance strategy is to let producers send dummy records to all consumers, such that the consumers can advance their merge logic. A typical deadlock resolution strategy is to spill data to disk, effectively going beyond bounded buffers and relaxing flow control.

Parallel nested loops join, index nested loops join, and general nested iteration add further complexity to the data exchange operation. One issue is the number of threads: if the nested operation should run in 4 threads (due to four on-disk partitions, for example) yet the outer operation runs in three threads, will in inner operation actually run in 12 threads? If not, communication and synchronization need careful design, including avoidance or resolution of deadlocks. Moreover, invoking the inner operation for batches of outer correlation values rather than for individual outer rows may reduce communication costs as well as create optimization opportunities in the inner query execution plan.

Another technique to reduce communication costs in all matching operations is bit vector filtering. Sometimes thought to apply only to hash join, it also applies to merge join and nested iteration. If the two inputs for the matching operation are scanned or computed in two separate phases, the first such phase can populate a bit vector filter that the second phase can exploit to eliminate some items without further predicate evaluation and in particular without communication in parallel query execution. The bit vector filter might permit hash collisions or hash collisions might be prevented by some means, typically involving dynamic growth of the bit vector filter. In addition to the actual bit vector filter, it can prove useful to retain information about the minimal and maximal values.

In addition to expensive join and grouping operations, parallel execution and bit vector filtering can benefit set operations such as intersecting lists of row identifiers when exploiting multiple indexes on the same table for a query with a conjunctive predicate. Other index operations include union, difference, and join. Index join sometimes refers to joining indexes of two tables prior to fetching complete rows from either table and sometimes refers to joining indexes from a single table in order to obtain all columns required in a query and to avoid fetching rows one-by-one. Both kinds of index joins can benefit from parallel execution and bit vector filtering.

Some operations, however, cannot exploit parallel execution. The most notable example is a “top” operation, e.g., a query to find the three most productive sales people in the organization. While the “top” operation permits local-global computation with potentially tremendous reduction in serial computation and in communication, the final computation cannot benefit from multiple concurrent threads. An exception to this exception are “top” operations that apply to groups of records, e.g., a query looking for the three most productive sales people in each region.

Some algorithm implementations also exploit local parallelism, independent of parallel execution of the overall query execution plan. Asynchronous I/O (sequential read-ahead, single-page prefetch, and write-behind) are forms of local concurrent execution. Expensive operations, e.g., sort operations, may benefit from multiple threads, e.g., fitting records into a sort operation’s workspace including initialization of a pointer array, sorting and rearranging the pointer array to represent the desired sort order, and forming on-disk runs from pointer arrays already sorted.

Parallel Execution beyond Queries

In addition to query execution, parallelism is also needed for large update operations and, perhaps most importantly, in utilities that scan or modify entire indexes, tables, or databases.

Parallel update operations are required in shared-nothing databases, simply because updates like filters are processed locally at each data site. Otherwise, if all update operations are relatively small, parallel update operations are not required for performance. However, update operations can be large, e.g., in bulk insertion and bulk deletion, also known as load, import, or roll-in and as roll-out.

A parallel update algorithm can follow the partitioning of stored data, it can assign disjoint key ranges within B-tree indexes to separate update threads, or it can assign threads to individual indexes. The latter approach is a special form of index-by-index updates (as opposed to row-by-row updates) as it permits the changes for each index to be sorted like the index for fast modification with read-ahead, etc.

Not only the actual database modifications but also other activities associated with updates can be parallel, including verification of integrity constraints, update propagation to materialized and indexed views, etc. Of course, verification of integrity constraints can be parallel also during definition of new constraints. Definition of new materialized and indexed views creates an opportunity for parallelism both while computing the query defining the view and while loading the query result into the new data structures. Refreshing a view by re-computation is very similar.

In practice, however, the most important parallel operation is probably index creation. High performance index creation is particularly important if online index creation is not supported, i.e., an entire table is read-locked while the index utility scans, sorts, and inserts data. Fortunately, parallel versions of all three of these steps are readily available using standard mechanisms for parallel query execution.

Key Applications

Parallel algorithms for database query execution are used for two reasons. First, if there are more processors than active users, the only means by which these processors can do useful work on behalf of the users are parallel algorithms. Pure pipelining often leads to poor load balancing; thus, partitioning stored data and intermediate results, in particular using hash functions, often permits higher and more reliable speed-up. In the future, many-core processors may well increase the importance of parallel algorithms in database query execution.

Second, the hardware might require parallel algorithm because the database and its data are partitioned across multiple nodes each with its own memory, operating system, etc. If communication is more expensive than immediate data reduction (using selection, projection, and local aggregation), parallel algorithm are required. The future of improvements in processing bandwidth and in communication bandwidth will probably continue to be unbalanced; the current trend towards networked storage over direct-attached storage may reverse, in particular for database systems that support a single-system image over many direct-attached storage devices including management of that storage.

Future Directions

While processor speed kept increasing, the value of parallel query execution has been doubted at times. With hardware development now focusing on many-core processors rather than ever higher clock speeds, there should be no doubt that parallel query execution is required for relational data warehousing and business intelligence. It may be worth noting, however, that parallel utilities such as index creation are more urgently needed by customers with growing data volumes than parallel query execution. Many parallel utilities are implemented using mechanisms shared with parallel query execution, however, such that parallel query technology is needed in any case.

With the emergence of XML and semi-structured data in database type systems, there probably will be growing interest in parallel algorithms for relational algebra extended for unstructured data and graph manipulation. In addition, more data cleaning, data mining, and scientific applications are integrated with databases, and they, too, require parallel execution very similar to parallel query execution. Perhaps the future of parallel query execution is limited by advances in extensible query optimization, such that both together enable deep integration of those parallel algorithm into the query processing framework.

Cross-references

- ▶ Partitioning
- ▶ Query Optimization for Parallel Execution
- ▶ Relational Algebra
- ▶ Storage Resource Management
- ▶ Workload Management

Recommended Reading

1. Bratbergsengen K. Hashing methods and relational algebra operations. In Proc. 10th Int. Conf. on Very Large Data Bases, 1984, pp. 323–333.
2. DeWitt D.J., Gerber R.H., Graefe G., Heytens M.L., Kumar K.B., and Muralikrishna M. GAMMA – a high performance dataflow database machine. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 228–237.
3. Fushimi S., Kitsuregawa M., Tanaka, H. An overview of the system software of a parallel relational database machine GRACE. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 209–219.
4. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73–170, 1993.

Parallel Query Optimization

Hans Zeller, Goetz Graefe
Hewlett-Packard Laboratories, Palo Alto, CA, USA

Synonyms

Optimization of parallel query plans

Definition

Parallel query optimization is the process of finding a plan for database queries that employs parallel hardware effectively. The details of this process depend on the types of parallelism supported by the underlying hardware, but the most common method is partitioning of the data across multiple processors.

Historical Background

Most parallel database systems today can trace part of their heritage back to the Gamma project at the University of Wisconsin, Madison in the 1980s – with the exception of the Teradata system, which predates Gamma by several years. Also influential were the GRACE database machine, developed at the University of Tokyo, and work at the Norwegian Institute of Technology, University of Trondheim. These projects did not publish a description of their parallel query optimization algorithms, however. Later projects, like XPRS (University of California, Berkeley) and IBM DB2 Parallel Edition describe that process in more detail.

A simple way of generating parallel query plans is to take the optimal plan produced by a conventional query optimizer and to parallelize it where possible. This may not result in the optimal parallel plan. For example, the chosen join order may require repartitioning of the data, while a slightly different join order might have avoided this costly step. Therefore, most of today's systems integrate parallelism into the optimization phase itself. Different parallel and serial plans are compared on the basis of their estimated cost, and the best plan is selected.

Most optimizer designers of parallel systems chose a very natural extension of non-parallel optimizers. This approach is described in the next section.

Foundations

Most of the parallel database systems use an operator-based model for parallel queries: Traditional relational operators such as join and groupby are employed in a parallel framework, allowing individual CPUs of a parallel system to work on a partition of the data. The task of the query optimizer is to find the right partitioning and degree of parallelism for each step.

With such an operator-based approach, optimizing parallel queries is similar to conventional query optimization, except that a new dimension is added – partitioning.

Extending a non-parallel optimizer to handle parallelism consists of four basic steps: First, the space of semantically correct parallel plans is defined. Second, the optimizer's search algorithm needs to be extended to enumerate a good part of the search space that was defined in the first step. Third, the cost model of the optimizer needs to be able to cost parallel plans such that the best one can be selected. Finally, all these extensions will likely increase the complexity of the optimization algorithm significantly. Heuristics are required to avoid an explosion of the cost of the optimization itself.

The following explores each of these four steps briefly.

Extending Query Plans to Execute in Parallel on Partitioned Data

The optimizer is responsible for generating semantically valid parallel plans. It therefore needs to have a good understanding of and model for parallel operator semantics. This is typically done by assigning a "partitioning" property to each relational operator and by using these properties to constrain the search space to semantically correct query plans. The partitioning property specifies a partitioning scheme (usually hash-based), the partitioning key, and the number of partitions.

In the following, the basic idea behind partitioning properties of a few key relational operators will be discussed.

When a file scan operator is executed in parallel, each parallel instance simply reads a partition of the data. On a shared-nothing system, the partitioning property describes how the data is physically partitioned. On a shared-everything architecture, another way is to let parallel scan operators compete for each block as it is read from disk, achieving a natural load balancing between the parallel operators, with a partitioning property that specifies only the number of partitions, not a partitioning key.

For joins, there are basically two different parallelization strategies. The first is to let each parallel instance join one partition of the first table R with a matching (co-located) partition of the second table S. Both R and S must have a matching partitioning property on the equi-join columns for this case. This is so that for a given row from a partition of R, all its matching rows from S are found in the corresponding partition of S.

The second parallel join algorithm is to join each partition of R with every partition of S or vice-versa. Such joins combine a partitioned table R with a replicated table S or vice versa. This type of parallel join is applicable in nearly all cases, except for full outer joins.

Terms like "co-located nested loop join," "broadcast hash join" or "repartitioned merge join" are used to distinguish these different cases [13].

Semijoins and outer joins can be parallelized as well, with the exception that only the operand requiring special handling can be broadcast. Parallelizing the UNION operator puts no special constraints on the partitioning properties, except what is needed for duplicate elimination.

To increase its choices, the execution engine typically is capable to produce partitioning artificially. This is in most systems implemented by a separate operator, called Exchange [11], Table Queue (DB2), or River [2]. Generally, the exchange operator is the only one that sends data through messages, all other operators process local data and pass their results on in local memory or in a local disk file.

Extending the Search Algorithm to Include Parallel Plans

Almost all query optimizers operate under a principle borrowed from dynamic programming that combines optimal sub-solutions to a larger optimal solution. Sub-solutions are optimal with respect to relevant properties, as described first in [Selinger]. Probably the key aspect of query optimization for parallel queries is that this system of properties can be very elegantly extended to include partitioning properties. Initially only consisting of ordering, with a sort operator to create artificially sorted results, partitioning properties and the exchange operator are now included to create artificially partitioned results.

Extending the properties of query execution plans with partitioning solves the key part of parallel query optimization, namely the problem of enumerating the parallel plans that are possible for a query. The driver for this enumeration are the different ways the other operators can be executed in parallel.

The exchange operator acts as the "enforcer" [11] or "glue" [14] for partitioning.

This extension applies to classical Selinger-style optimizers (e.g., DB2) as well as to rule-based optimizers derived from the Exodus, Volcano and Cascades approaches (e.g., Microsoft SQL Server).

Costing Parallel Query Plans

Executing queries in parallel usually increases the amount of resources consumed, due to communication and synchronization overhead. Parallel execution is therefore not well-suited to minimize the resource cost of queries. Most parallel query optimizers try to minimize the time a query would take if executed in single-user mode on the system. When costing a parallel operator, they therefore compute the cost for a single partition only (assuming that the data is equally distributed over the partitions without skew) and add the cost for needed synchronization. With the goal of keeping the overall query response time low, it is usually not beneficial to reduce the degree of parallelism, therefore the optimizers consider only plans that use all the processing nodes of the system and compare them with a serial plan. The explosion of choices comes with different methods of partitioning and/or replicating data to allow for different types of join execution plans.

Heuristics to Reduce the Number of Parallel Plans Considered

One heuristic, the use of "interesting orders" [Selinger], can be used for partitioning as well. Optimizers with top-down exploration achieve this filtering effect naturally. Without some heuristics, the number of possible partitioning schemes could easily explode. For example, consider a co-located parallel merge join for the query "select * from R join S on R.a = S.a and R.b = S.b and R.c = S.c." Any ordering on a subset of columns (a,b,c), combined with a partitioning on some or all of the columns in this subset would constitute a valid set of properties for the join operator – assuming it is present for both tables R and S. Most implementations of optimizers will heuristically try only a small number of possibilities in this case.

Key Applications

Parallel computers allow database applications to scale, and optimization of parallel queries allows users to write applications without expending effort on parallelizing database queries. Parallel databases are the main application today where automatic parallelization of generic requests is performed successfully.

Future Directions

Challenges for future systems include adaptive systems that learn from poorly parallelized queries and that avoid data skew automatically, as well as automatic advisors and optimizers of physical designs for parallel databases.

Cross-references

► Parallel Query Execution Algorithms
► Query Optimization

Recommended Reading

1. Ballinger C and Fryer R. Born to be parallel. why parallel origins give teradata an enduring performance edge. IEEE Data Eng. Bull., 20(2):3–12, 1997.
2. Barclay T., Barnes R., Gray J., Sundaresan P. Loading databases using dataflow parallelism. ACM SIGMOD Rec. 23(4):72–83, 1994.
3. Baru C.K., Fecteau G., Goyal A., Hsiao H.-I., Jhingran A., Padmanabhan S., and Wilson W.G. An overview of DB2 parallel edition. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 460–462.
4. Boral H., Alexander W., Clay L., Copeland G.P., Danforth S., Franklin M.J., Hart B.E., Smith M.G., and Valduriez P. Prototyping Bubba, a highly parallel database system. IEEE Trans. Knowl. Data Eng., 2(1):4–24, 1990.
5. Bratbergsengen K. Algebra operations on a parallel computer – performance evaluation. In Proc. 5th Int. Workshop on Data Machines, 1987, pp. 415–428.
6. Chen A., Kao Y.-F., Pong M., Shak D., Sharma S., Vaishnav J., Zeller H. Query processing in nonstop SQL. IEEE Data Eng. Bull., 16(4):29–41, 1993.
7. DeWitt D.J., Gerber R.H., Graefe G., Heytens M.L., Kumar K.B., and Muralikrishna M. GAMMA – a high performance dataflow database machine. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 228–237.
8. DeWitt D.J. and Gray J. Parallel database systems: the future of high performance database systems. Commun. ACM, 35(6):85–98, 1992.
9. DeWitt D.J., Smith M., and Boral H. A single-user performance evaluation of the Teradata database machine. In Proc. of the 2nd Int. Workshop on High Performance Transaction Systems, 1987.
10. Fushimi S., Kitsuregawa M., and Tanaka H. An overview of the system software of a parallel relational database machine GRACE. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 209–219.
11. Graefe G. Encapsulation of parallelism in the volcano query processing system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 102–111.
12. Graefe G. and Davison D.L. Encapsulation of parallelism and architecture-independence in extensible database query execution. IEEE Trans. Software Eng., 19(8):749–764, 1993.
13. Jhingran A., Malkemus T., and Padmanabhan S. Query optimization in DB2 parallel edition. IEEE Data Eng. Bull., 20(2):27–34, 1997.
14. Lee M.K, Freytag J.C., Lohman G.M. Implementing an interpreter for functional rules in a query optimizer. In Proc. 14th Int. Conf. on Very Large Data Bases, 1988, pp. 218–229.
15. Mohan C. Pirahesh H., Tang W.G., and Wang Y. Parallelism in relational database management systems. IBM Sys. J., 33(2):349–371, 1994.
16. Neches P.M. The anatomy of a database computer system. In Digest of Papers - COMPCON, 1985, pp. 252–254.
17. Selinger, P., Astrahan, M., Chamberlin, D., Lorie, R. and Price, T. Access Path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.
18. Stonebraker M., Katz R.H., Patterson D.A., Ousterhout J.K. The design of XPRS. In Proc. 14th Int. Conf. on Very Large Data Bases, 1988, pp. 318–330.
19. von Bueltzingsloewen G. Optimizing SQL queries for parallel execution. In Proc. on Workshop on Database Query Optimization, 1989.

Parallel Query Processing

Esther Pacitti[1,2]
[1]University of Nantes, Nantes, Cedex, France
[2]INRIA, LINA, Nantes, Cedex, France

Synonyms

Multiprocessor query processing

Definition

Parallel query processing designates the transformation of high-level queries into execution plans that can be efficiently executed in parallel, on a multiprocessor computer. This is achieved by exploiting the way data is placed in parallel and the various execution techniques offered by the parallel database system. As in query processing, the transformation from the query into the execution plan to be executed must be both correct and yield efficient execution. Correctness is obtained by using well-defined mappings in some algebra, e.g., relational algebra, which provides a good abstraction of the execution system. Efficient execution is crucial for high-performance, e.g., good query response time or high query throughput. It is obtained by exploiting efficient parallel execution techniques and query optimization which selects the most efficient parallel execution plan among all equivalent plans.

The main forms of parallelism which can be inferred by high-level queries are inter-query parallelism (several queries executed in parallel) and intra-query parallelism (each query executed in parallel), and within a query, inter- and intra-operator parallelism. These various forms of parallelism are obtained based on how data is placed in the parallel system, i.e., physically on disk or memory. Hence, the term data-based parallelism is communally used. These forms of parallelism can be combined and are exploited by parallel execution techniques.

Historical Background

Parallel query processing had been initiated in the context of database machines in the late 1970's. At that time, parallel processing techniques were already used successfully in scientific computing to improve the response time of numerical applications. However, these techniques are very complex as they typically involve parallelizing compilation which must infer parallelism from programs written in a procedural language, e.g., Fortran or C. Since these programs may be sequential, inferring parallelism to obtain good performance is hard.

With a high-level language like SQL, parallelism is relatively easier to infer. SQL enables programmers to specify what data to access (with predicates) without any detail on how. Thus, this provides the parallel database system much leverage to decide how to access data.

In the context of database machines, parallel query processing was very simple and concentrated on select-project-join queries using brute-force algorithms and special-purpose hardware. As parallel database systems evolved to exploit general-purpose multiprocessors and software-oriented solutions, parallel query processing has become much more complex. Also, the emergence of new applications such as OLAP and data mining translated into new features in SQL which have made parallel query processing more complex.

Compared to distributed query processing (which can also exploit parallelism by executing a query using multiple sites), the main difference is that the parallel system can have many more nodes and a very fast interconnection network. Thus, the execution techniques and execution plans are fairly different.

Foundations

Parallel query processing is the major solution to high-performance management of very large databases. The basic principle is to partition the database across multiple disks or memory nodes so that much inter- and intra-query parallelism can be gained. This can lead to significant improvements in both response time (the time the query takes to be executed and return results) and throughput (number of queries or transactions per time unit). However, decreasing the response time of a complex query through large-scale parallelism may well increase its total time (by additional communication) and hurt throughput as a side-effect. Therefore, it is crucial to optimize and parallelize queries in order to minimize the overhead of parallelism, e.g., by constraining the degree of parallelism for the query.

The performance of a parallel database system should ideally demonstrate two advantages: linear speedup and linear scaleup. *Linear speedup* refers to a linear increase in performance for a constant database size and linear increase in the number of nodes (i.e., processing and storage power). Thus, the addition of computing power should yield proportional increase in performance. *Linear scaleup* refers to a sustained performance for a linear increase in both database size and workload and number of nodes. Furthermore, extending the system should require minimal reorganization of the existing database. Thus, an increase of computing power proportional to the increase in database size and load should yield the same performance. These advantages are ideal since, in practice, increasing the configuration tends to increase the overhead of parallelism (e.g., more communications between nodes, more interference when accessing shared resources, etc.).

Partitioned data placement is the basis for the parallel execution of database queries. Given a parallel system with n nodes (each node may have one or more processors accessing a main memory and disks), each database table T is typically partitioned onto a number of nodes (less or equal to n) so that each node stores a subset of the tuples of T. This corresponds to horizontal fragmentation in distributed databases and similar techniques can be used. However, because there can be many nodes, more scalable techniques are also often used such as round robin or hashing on the placement attribute(s). Furthermore, to improve availability and performance, some partitions (typically those which are accessed more than others) can be replicated.

Given a partitioned database, parallel query execution can exploit two forms of parallelism: inter- and intra-query. *Inter-query parallelism* enables the parallel execution of multiple queries, each at a different node, in order to increase query throughput. This form of parallelism, reminiscent of that used in transaction processing systems, is quite simple since queries need not be parallelized. Thus, incoming queries can simply be dispatched (by a load balancer of the parallel database system) to the nodes which store the data corresponding to the queries. However, this can only work if all the data accessed by a query are stored at the node, or accessible at the node. For instance, in the case of a shared-nothing parallel database system, if a query Q involves data that is partitioned at two different nodes, then either node cannot entirely execute the query. But in the case of a shared disk parallel database system, since all the disks can be accessed from each node, the same query Q could be executed by either node.

A query can be decomposed in a tree of relational operators, where each operator takes data as input (either base data or temporary data) and produces temporary data, with the root operator producing the final result. This decomposition allows for *intra-query parallelism*) which has two forms: inter-operator and intra-operator parallelism. *Inter-operator* parallelism is obtained by executing in parallel several operators of the query on several nodes while with intra-operator parallelism, the same operator is executed by many nodes, each one working on a different partition of the data. For inter-query parallelism, much attention has been devoted to pipelined (or consumer-producer) parallelism which enables consumer operators to start execution as soon they get input data from producer operators. Pipelined executions do not require temporary relations to be materialized, i.e., a tree node corresponding to an operator executed in pipeline is not stored. These two forms of parallelism complement each other well since inter-operator parallelism gets beneficial as the query gets complex (with many operators) while intra-operator parallelism gets beneficial for heavy operators (which access large parts of the data).

To exploit inter- and intra-query parallelism, using database partitioning, parallel algorithms for the execution of relational operators are necessary. Such algorithms must yield a good trade-off between parallelism and communication cost since increasing parallelism involves more communication among nodes. Parallel algorithms for unary operators (e.g., select) are relatively simple as they typically exploit data placement or secondary indices to restrict access to base data. Parallel algorithms for binary operators such as join are much more involved since join can incur much communication. The main algorithms exploit the placement of one of the two relation to reduce communication or artificially create a good data placement using hashing. When there is sufficient main memory to hold one of the two relations (which is a very practical case), the execution of hash-based join can be pipelined across multiple operator nodes, thus increasing performance. Since the intermediate results of the operators can be skewed, parallel algorithms must also deal with load unbalancing, e.g., by redistributing a heavy work at one node to multiple nodes using hashing.

Parallel query optimization is the process of selecting the best parallel execution plan for a query. Compared to distributed query optimization, it focuses on taking advantage of both intra-operator parallelism and inter-operator parallelism. As any query optimizer, a parallel query optimizer can be seen as three components: a search space, a cost model, and a search strategy. Parallel execution plans are abstracted by means of operator trees, which define the order in which the operators are executed. Operator trees are enriched with annotations, which indicate additional execution aspects, such as the algorithm of each operator. An important aspect to be reflected by annotations is that operators are executed in pipeline. The cost model provides the cost functions for parallel operator algorithms.

Key Applications

Parallel query processing has been developed in the context of parallel database systems, with a focus on OLAP applications, where good response time is crucial. Most of the work has been done in the context of the relational model. In order to support new OLAP applications which may access all kinds of data, including unstructured and semi-structured (XML) data, major extensions to parallel query processing are necessary. In particular, techniques from parallel database and information retrieval need to be combined.

Cross-references

- ▶ Distributed Databases
- ▶ Parallel Data Placement
- ▶ Parallel Query Execution Algorithms
- ▶ Query Optimization

Recommended Reading

1. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73–170, 1993.
2. Kossmann D. The state of the art in distributed query processing. ACM Comput. Surv., 32(4):422–469, 2000.
3. Lanzelotte R., Valduriez P., Zait M., and Ziane M. Industrial-strenght parallel query optimization: issues and lessons. Inf. Sys., (19)4:311–330, 1994.
4. Özsu M.T. and Valduriez P. Principles of distributed database systems. Prentice-Hall, 2nd edn, 1999.
5. Valduriez P. and Pacitti E. Parallel database systems. In Handbook of Database Technology, J. Hammer, M. Scheider (eds.). CRC, USA, 2007.

Parallel SCSI

▶ Storage Protocols

Parameterized Complexity of Queries

Christoph Koch
Cornell University, Ithaca, NY, USA

Definition

Parameterized complexity theory is the study of the interaction between the fixing of parameters of input problems and their computational complexity. A central parameterized complexity concept is that of a fixed-parameter tractable (FPT) problem, which captures a strong notion of well-behavedness of a problem under the assumption that parameter values do not grow with input sizes. There is also a solid theory of fixed-parameter intractability, which gives strong evidence that for certain parameterizations of problems, no FPT algorithms can be found.

Historical Background

Fixed-parameter complexity theory is strongly associated with R. Downey and M. Fellows, who did much seminal work in the area (cf. [3,5]). The first fixed-parameter complexity result in the context of database query evaluation was the linear-time query processing algorithm for acyclic conjunctive queries by Yannakakis in 1981 [12], which preceded the development of parameterized complexity theory (cf. also [13]).

Foundations

Fixed-Parameter Tractability

A *parameterized (decision) problem* is a set of pairs (x,k), where x is called the input and k the parameter (an integer). As a convention, $n = ||x||$ will be used to denote the size of the input. A parameterized problem is called strongly uniformly *fixed-parameter tractable* (subsequently, just fixed-parameter tractable, or FPT), if there is a computable integer function f, a constant c, and an algorithm that, given a parameterized problem instance (x, k), decides x in time $O(f(k) \cdot n^c)$.

It is important to note the difference between an algorithm that runs in time $O(n^k)$ and an FPT algorithm. If $k = 10$ and $c = 1$, an algorithm that runs in time $O(f(10) \cdot n)$ is linear (with possibly a large constant), while an $O(n^{10})$ algorithm will not even scale to very moderately sized problem instances.

Consider the *data complexity* of queries [11], i.e., the complexity of evaluating a query on a database if only the database is considered part of the input, while the query is fixed and part of the problem specification. It is well known that quite naive query evaluation techniques can evaluate an arbitrary relational algebra query in time $O(n^k)$, where k is the size of the query (e.g., the number of algebra operators involved). By fixing k, the query evaluation time becomes polynomial, but the $O(n^k)$ time query evaluation technique does not yield a fixed-parameter tractability result. Indeed, such an FPT result is considered unlikely to exist.

For another example, consider the Set Cover problem, a well known combinatorial problem that appears in database problems such as subspace clustering or finding pipelined query plans in stream processors.

A set C is called a *cover* of a set of sets $\mathbf{S}$ if for each $S \in \mathbf{S}$, $S \cap C \neq \emptyset$. The Set Cover problem is defined as follows.

Set Cover
Input: An integer k, a finite set V, and a set $\mathbf{S} \subseteq 2^V$ of subsets of V.
Question: Is there a set C with $|C| \leq$ k such that C is a cover for $\mathbf{S}$?

The Set Cover problem is NP-complete. A naive brute-force algorithm would check for each set $C \subseteq V$ with $|C| \leq k$ whether C is a cover. This algorithm runs in time $O(|V|^k \cdot n)$, which is polynomial in n if k is fixed. However, this is not an FPT algorithm with parameter k because $|V|$ may get arbitrarily large with the input.

Consider the following refined algorithm for Set Cover (cf. [5,8]).

```
$\mathbf{C}_0 := \{\emptyset\}$;
**for** $i = 1$ **to** $k$ **do** $\mathbf{C}_i = \{C \cup \{a\} | C \in \mathbf{C}_{i-1}, a \in S(C)\}$;
**if** at least one element of $\mathbf{C}_k$ is a cover for $\mathbf{S}$ **then**
**output** true;
**else output** false
```

Here, $S(C)$ is a function that returns, in a deterministic way, an element S of $\mathbf{S}$ such that, if this condition can be satisfied by any element of $\mathbf{S}$, $S \cap C \neq \emptyset$. One way of defining $S(C)$ is as

$$S(C) = \begin{cases} \min \mathbf{S}(C) \ldots \mathbf{S}(C) \neq \emptyset \\ \min \mathbf{S} \ldots \text{otherwise.} \end{cases}$$

where $\mathbf{S}(C) = \{S \in \mathbf{S} \mid S \cap C \neq \emptyset\}$ and min returns the smallest element of a subset of **S** with respect to some arbitrary fixed order among the elements of **S** (e.g., the order in which the elements of **S** are stored in memory).

Consider for example $\mathbf{S} = \{\{a, b\},\{a, c\},\{b, d\},\{c, e\}, \{c, f\}\}$ with the elements of **S** ordered as just enumerated. Then, $S(\emptyset) = \{a, b\}$, $\mathbf{C}_1 = \{\{a\},\{b\}\}$, $S(\{a\}) = \{b, d\}$, $\mathrm{S}(\{b\}) = \{a, c\}$, and $\mathbf{C}_2 = \{\{a, b\},\{a, d\},\{b, c\}\}$. Since $\{b, c\}$ is a cover, the algorithm returns true.

It is easy to check that this algorithm runs in time $O(s^k \cdot n)$ where s is the maximum cardinality among the elements of **S**, i.e., $s = \max\{|S| : S \in \mathbf{S}\}$. Thus, Set Cover with parameter $k + s$ (which is not the standard parameterization for Set Cover, which would be k) is FPT. If s is small, this may be a useful algorithm.

Fixed-Parameter Intractability

For a number of parameterized versions of NP-hard problems it can be shown that they are either provably not FPT or not FPT unless P=NP. An example for the former scenario would be EXPTIME Turing machine acceptance (which is EXPTIME-complete) with a dummy parameter that is unrelated to the input. An example of the latter scenario is the satisfiability of CNF formulae whose clauses have no more than k literals, with parameter k. For $k = 3$, this is the 3SAT problem, which is NP-complete. But there are also parameterized problems for which apparently no FPT algorithm exists and for which more subtle hardness arguments based on special complexity classes for parameterized problems have to be developed.

A *fixed-parameter many-one reduction* is a reduction that maps each instance (x, k) of a parameterized problem to an instance (x', k') of another parameterized problem using an FPT algorithm (i.e., in time $O(f(k) \cdot x^c)$), with the additional condition that the size of k' must only depend on k but not on x, such that (x, k) is a yes-instance of the first problem if and only if (x', k') is a yes-instance of the second problem. Closure under such reductions yields robust complexity classes. The class of parameterized problems that are fixed-parameter many-one reducible to problem Π shall be denoted by $[\Pi]^{\text{fpt}}$. A problem in $[\Pi]^{\text{fpt}}$ is called *complete* for $[\Pi]^{\text{fpt}}$ if Π is fixed-parameter many-one reducible to it.

Let $\Gamma_{0,d} = \Delta_{0,d}$ be the class of all propositional formulae constructible from propositional variables using negation and the *binary* operations conjunction $\wedge$ and disjunction $\vee$ such that the maximum expression depth, not taking into account negations, is d. In other words, for any such formula, the maximum number of disjunctions or conjunctions occurring on paths from the root of the expression tree to a leaf is d. Now, $\Gamma_{t,d}$ consists of the formulae of the form $\wedge\ \Phi$, where Φ is a finite subset of $\Delta_{t-1,d}$, and $\Delta_{t,d}$ consists of the formulae of the form $\vee\ \Psi$, where Ψ is a finite subset of $\Gamma_{\mathrm{t-1,d}}$.

A usual way of defining the W hierarchy is in terms of a weighted version of the propositional satisfiability problem. The weight of a truth assignment for the variables of a propositional formula is the number of variables set to true in that truth assignment. The weighted satisfiability problem WSAT($\Gamma_{t,d}$) for the class of formulae $\Gamma_{t,d}$ is as follows: Given a formula $\phi \in \Gamma_{t,d}$ and parameter k, does ϕ have a satisfying assignment of weight k?

The W-hierarchy, for each $t \geq 1$, is defined as

$$W[t] := \underset{d \geq 0}{U} \left[\mathrm{WSAT}(\mathit{\Gamma}_{t,d})\right]^{\text{fpt}}$$

with $W[t] \subseteq W[t+1]$ and is believed to be strict ($W[t] \subsetneq W[t+1]$). For a few examples, natural parameterized versions of Clique and Independent Set are $W[1]$-complete, while parameterized versions of Hitting Set, Dominating Set, and Kernel are $W[2]$-complete (see [5]). These results depend on the choice of parameter, and the standard parameterization is the size of the structure (set) whose existence is to be guessed and verified. The question whether FPT $\neq$ W[1] is the parameterized complexity analogue of the question whether P $\neq$ NP. It remains unproven, but is strongly suspected.

Returning to the evaluation complexity of relational algebra queries with the size of queries as the parameter,

Parameterized Query Evaluation
Input: A Boolean relational calculus query Q and a relational database.
Parameter: The size k of a reasonable representation of Q in bits.
Question: Does Q return true on the input database?

which is known to be $W[1]$-complete [9] already for conjunctive queries (i.e., select-project-join queries) and AW[*]-complete for full relational calculus [4]. AW[*] is a complexity class that subsumes the $W[t]$ classes, for all t, but may contain additional problems.

Further Positive Results on Query Evaluation Complexity

While relational query languages such as relational algebra, calculus, and datalog (a generalization of conjunctive queries) are W[1]-hard and thus unlikely to be fixed-parameter tractable, there are important fragments of these languages that are FPT. Moreover, there are other logics and query languages, specifically on tree- and graph-structured data models, which are FPT with the size of the query as the parameter.

The first FPT result in the context of database query processing was on the evaluation of *acyclic* conjunctive queries by Yannakakis. Consider a Boolean conjunctive query in datalog notation, $q \leftarrow R_1(\vec{y}_1),\ldots,R_k(\vec{y}_k)$. The acyclicity of queries refers to a notion of acyclicity of associated hypergraphs, whose nodes are query variables and whose hyperedges are the sets of variables $\vec{y}_i$ occurring together in an atomic formula of the query. Acyclicity can be defined in a number of ways, such as by a low-complexity algorithm or using guarded logic. An exact definition is beyond the scope of this article, but in the case that all input relations are binary and no atom is of the form $R(y, y)$, the hypergraph is an undirected graph and hypergraph acyclicity coincides with the standard graph-theoretic notion of acyclicity. By exploiting the tree structure of this graph, which describes the necessary joins, and projecting away columns that will not be involved in further joins as early as possible, it is always possible to find a query plan that can be evaluated in time $O(f(k) \cdot n)$ where k is the size of the query – i.e., the problem is fixed-parameter linear.

A Boolean acyclic conjunctive query can be written using just selections and semijoin operations, with a $\pi_\emptyset$ operation on top. If Q_1 and Q_2 are select-semijoin-queries, then $Q_1(x) \ltimes Q_2(x) = \pi_{\mathsf{sch}(Q_1)}(Q_1(x) \bowtie Q_2(x))$ is a subset of $Q_1(x)$ and can be computed in linear time in $|Q_1(x)| + |Q_2(x)|$, where x is the input database. By induction it follows that overall query evaluation is FPT.

Consider the acyclic conjunctive query $\pi_\emptyset\ (R \bowtie S \bowtie T \bowtie U \bowtie V)$ for database schema $R(A,B)$,$S(B,C)$,$T(C,D)$,$U(C,E)$,$V(A,F)$. The hypergraph (here, graph) is acyclic:

The query plan $\pi_\emptyset\ ((R \ltimes V) \ltimes ((S \ltimes \mathbf{T}) \ltimes U))$ admits efficient evaluation.

For nonboolean conjunctive queries, the running time is polynomial of degree c where c is the arity of the query result (which must be considered a true constant rather that a parameter if this is to be considered an FPT result), and clearly, this is optimal because the output size of a query that simply computes the product of the input relations is $\Omega(n^c)$.

The notion of hypergraph acyclicity has been generalized to queries of bounded hypertree-width, which is a measure of how tree-like the query hypergraph is: hypertree-width 1 coincides with hypergraph acyclicity. It was shown in [7] that conjunctive query evaluation with the hypertree-width of the query as the parameter is FPT and that this generalizes in a natural way to relational calculus queries. The fixed-parameter tractability of queries of bounded hypertree-width > 1 (in fact, 2) has been used to explain the polynomial-time complexity of XPath queries [1].

A classical result by Courcelle [2] shows that very powerful queries – in monadic second-order logic, a language that strictly subsumes relational calculus – over databases of bounded tree-width (the exact definition is technical, but in the case that all relations are binary, the condition is bounded tree-width of the (undirected) graph obtained by unioning the relations together) are fixed-parameter linear. Note that in this result the structure of the data is restricted but the structure of queries is not. The special case where the database is a tree with node labels from a finite alphabet is due to Doner, Thatcher and Wright [10]. The algorithm is based on compiling the query into a tree automaton which, once obtained, can be evaluated on the data tree in linear time. Thus, the running time is $O(f(k) + n)$, where k is the size of the query. This is a famous case where f is much worse than singly exponential: f is nonelementary – a tower of twoes $2^{2^{2^{\cdot^{2}}}}$ whose height grows with k. This is apparently necessarily so: Frick and Grohe [6] show that unless $P = NP$, any FPT algorithm for the problem must have a nonelementary f.

Key Applications

Fixed-parameter complexity results can assist designers of data management algorithms in two ways. There is now a large set of positive results for a variety of parameterized versions of NP-hard problems, which may surface in contexts such as query optimization and data mining. For a particular data management problem, it may be known that a particular parameter is always bounded in the problem instances that arise. If the problem is FPT for that parameter, there is an efficient algorithm for solving the problem. Furthermore, FPT results exist for fragments of the relational query languages such as relational algebra as well as for query languages for tree- and graph-structured data.

Conversely, if it is known that a parameterized problem is W[1]-hard, then it is quite hopeless to try to develop an efficient algorithm for solving the parameterized problem; in that case one may look for different acceptable parameterizations or for efficient approximation techniques.

Cross-references

▶ Complexity
▶ Complexity Results
▶ Tree Automata

Recommended Reading

1. Benedikt M. and Koch C. XPath Leashed. ACM Comput. Surv., 4(1), 2008.
2. Courcelle B. Graph rewriting: an algebraic and logic approach. In Handbook of Theoretical Computer Science, J. van Leeuwen (ed.). vol. 2, chap. 5, Elsevier B.V., Amsterdam, The Netherlands, 1990, pp. 193–242.
3. Downey R.G. and Fellows M.R. Parameterized Complexity. Springer, Berlin, 1999.
4. Downey R.G., Fellows M.R., and Taylor U. The parameterized complexity of relational database queries and an improved characterization of W[1]. In Proc. DMTCS ’96. Combinatorics, Complexity, and Logic, 1996, pp. 194–213.
5. Flum J. and Grohe M. Parameterized Complexity Theory. Springer, Berlin, 2006.
6. Frick M. and Grohe M. The complexity of first-order and monadic second-order logic revisited. In Proc. 17th Annual IEEE Symp. on Logic in Computer Science, 2002, pp. 215–224.
7. Gottlob G., Leone N., and Scarcello F. Hypertree decompositions and tractable queries. J. Comput. Syst. Sci., 64(3):579–627, 2002.
8. Grohe M. Parameterized complexity for the database theorist. ACM SIGMOD Rec., 31(4), 2002.
9. Papadimitriou C.H. and Yannakakis M. On the complexity of database queries. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997.
10. Thatcher J. and Wright J. Generalized finite automata theory with an application to a decision problem of second-order logic. Math. Syst. Theory, 2(1):57–81, 1968.
11. Vardi M.Y. The complexity of relational query languages. In Proc. 14th Annual ACM Symp. on Theory of Computing, 1982, pp. 137–146.
12. Yannakakis M. Algorithms for acyclic database schemes. In Proc. of the Seventh Int. Conf. on Very Large Data Bases, 1981, pp. 82–94.
13. Yannakakis M. Perspectives on database theory. In Proc. of the 36th IEEE Symp. on Foundations of Computer Science, 1995, pp. 224–246.

Parametric Data Reduction Techniques

RUI ZHANG
University of Melbourne, Melbourne, VIC, Australia

Definition

A parametric data reduction technique is a data reduction technique that assumes a certain model for the data. The model contains some parameters and the technique fits the data into the model to determine the parameters. Then data reduction can be performed.

Key Points

Parametric data reduction (PDR) techniques is opposite to nonparametric data reduction (NDR) techniques. A model with parameters is used in a PDR technique and therefore some computation is required to determine these parameters, which may be costly. However, if a PDR technique is well-chosen, it may result in much more data reduction than NDR techniques. A representative example is linear regression [3]. Linear regression assumes that the data fall on a straight line, expressed by the following formula

$$Y = a + bX \tag{1}$$

Given a set of points (Assuming two dimensions.) $\{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \ldots\}$, parameters a and b in Equation (1) are determined from the points using the least squares criteria. The result is

$$b = \frac{\sum(X - \overline{X})(Y - \overline{Y})}{\sum(X - \overline{X})^2}$$

$$a = \overline{Y} - b\overline{X}$$

where $\overline{X}$ and $\overline{Y}$ are the average values of $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$, respectively. If the data are actually distributed on almost a line, linear regression is a very efficient data reduction technique. Besides the values of a and b, only one dimension of every point is needed to represent the data set. The data volume is reduced by half. However, if the data are not distributed on a line, linear regression will result in large errors.

Other popular PDR techniques include Singular Value Decomposition (SVD) [2] and Discrete Wavelet Transform (DWT). SVD assumes that a matrix is decomposed to the form of $\mathbf{A} = \mathbf{USV}^t$ and $\mathbf{U}, \mathbf{S}, \mathbf{V}$ need to be determined; DWT assumes that a signal is projected to a set of orthogonal basis vectors and the wavelet coefficients need to be determined. A summary of data reduction techniques including PDR techniques can be found in [1].

Cross-references

▶ Data Deduction
▶ Discrete Wavelet Transform and Wavelet Synopses
▶ Linear Regression
▶ Nonparametric Data Reduction Techniques
▶ Singular Value Decomposition

Recommended Reading

1. Barbará D., DuMouchel W., Faloutsos C., Haas P.J., Hellerstein J.M., Ioannidis Y.E., Jagadish H.V., Johnson T., Ng R.T., Poosala V., Ross K.A., and Sevcik K.C. The New Jersey data reduction report. IEEE Data Eng. Bull., 20(4):3–45, 1997.
2. Jolliffe I.T. Principal Component Analysis. Springer, Berlin, 1986.
3. Wonnacott R.J. and Wonnacott T.H. Introductory Statistics. Wiley, New York, 1985.

Partial Replication

Bettina Kemme
McGill University, Montreal, QC, Canada

Definition

A replicated database consists of a set of nodes $\mathcal{N}$ (database servers) and each logical data item x has a physical copy on a subset $\mathcal{N}_x$ of the nodes in $\mathcal{N}$. The replication degree $r_x = |\mathcal{N}_x|$ of a data item is the number of copies it has. Using *full replication*, each logical data item has a copy on each of the nodes, i.e., for each data item x of the database, $\mathcal{N}_x = \mathcal{N}$. Whenever there is at least one data item that does not have copies at all nodes, one refers to a partial replication architecture.

Key Points

Partial Replication can be used for different purposes:

Cluster Replication

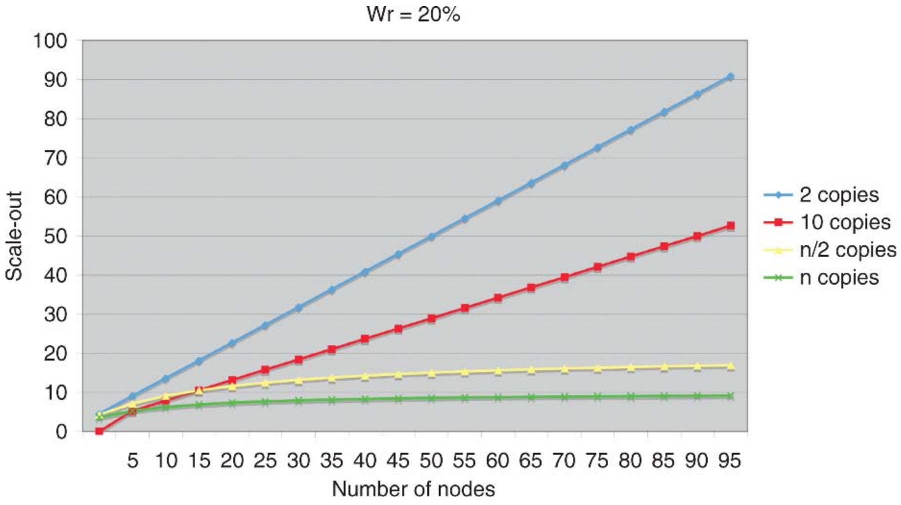
In order to achieve scalability, data can be replicated across nodes residing in a single cluster. Read access to data items can then be distributed across the existing copies. Write operations, however, have to be performed on all copies. The potential for scalability can be best understood by a simple analytical model. It assumes each transaction accesses one data item and has execution cost normalized to one unit. A fraction of *wr* are update transactions. Communication costs and concurrency control issues are ignored. A read-only transaction is executed at any node with a copy of the data item while an update transaction is executed at all nodes. The system consists of n nodes each having the capacity to process C execution units per time unit. There are m data items each having the same replication factor r. The copies are equally distributed across the nodes ($\frac{m*r}{n}$ copies per node). All data items are accessed with the same frequency and requests are equally distributed across all nodes.

If the replicated system can execute l transactions per unit then $(1 - wr) * l$ are read-only transactions executed at one node and $wr * l$ are update transactions executed at r nodes. Thus, the total capacity of the system is used as follows: $n * C = (1 - wr) * l + r * wr * l$. The scale-out is the number of transactions l the replicated system can handle per time divided by the number of transactions a non-replicated system can handle (which is C). Therefore, the scale-out of an n-node system is $\frac{n}{1+(r-1)*wr}$.

Figure 1 shows the scale-out for systems up to 100 nodes with an update load of 10%, and each data item has 2, 10, $n/2$ or n copies. While a constant replication factor provides linear scalability (and the smaller the better), a replication factor that increases with the number of nodes leads to a scalability cealing. Once the saturation point is reached, adding more nodes will not increase the throughput because applying update transactions consumes most of the available resources. More detailed performance models are given in [2,1].

WAN Replication

Replicating data items at different geographical locations is mainly used to provide fast local access to clients of the different regions. Having a data item replicated at a specific location decreases



Partial Replication. Figure 1. Scale-out of partial replication.

communication costs for read operations but increases communication and processing costs for update transactions, and has additional storage costs. Thus, placement algorithms have to decide where to put replicas in order to find a trade-off between the different factors [3].

Challenges

When a transaction executes at a specific node but the node does not have a copy of a requested data item, remote access is necessary. This poses several challenges. First, for update transactions concurrency control becomes more complicated. Second, a location mechanism must be implemented that finds nodes with appropriate data copies. Third, for complex SQL queries, query execution becomes distributed potentially requiring data shipping and advanced operators. Therefore, in practice, data that is often accessed together (within a transaction or query) is co-located. This, however, reduced the flexibility in terms of optimizing load-balancing and update processing.

Cross-references

- ▶ Replica Control
- ▶ Replication for Scalability
- ▶ WAN Data Replication

Recommended Reading

1. Nicola M. and Jarke M. Performance modeling of distributed and replicated databases. IEEE Trans. Data Knowl. Eng., 12(4):645–672, 2000.
2. Serrano D., Patiño-Martínez M., Jiménez-Peris R., and Kemme B. Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation. In Proc. IEEE Pacific Rim Dependable Computing Conference, 2007.
3. Wolfson O., Jajodia S., and Huang Y. An adaptive data replication algorithm. ACM Trans. Database Syst. 22(2): 255–314, 1997.

Partitioned Query Execution

▶ Parallel Query Execution Algorithms

Partitioning

▶ Parallel Query Execution Algorithms

Passage Retrieval

▶ Structured Document Retrieval

Path Functional Dependencies

▶ Functional Dependencies for Semi-structured Data

Path Index

▶ Structure Indexing

Path Query

Yuqing Wu
Indiana University, Bloomington, IN, USA

Synonyms

Document path query

Definition

Given a semi-structured data set D, a *path query* identifies nodes of interest by specifying the *path* lead to the nodes and the predicates associated with nodes along the *path*. The *path* is identified by specifying the labels of the nodes to be navigated and structural relationship (parent-child or ancestor-descendant) among the nodes. A predicate can be a path query itself, relative to the node that it is associated with.

Historical Background

Using path information in query processing has been studied in the object-oriented database systems, in which most queries require the traversing from one object to another following object identifiers, in the mid 1990's. The notion of path query, in which the path and predicates along the path are specified as the core of the query, became popular with the growth of the information on the web and the introduction of semi-structured data, especially XML.

Most of the popular query languages for querying XML data, such as XPath [6] and XQuery [4], employ path query as the approach to identify nodes of interest. However, some techniques, such as schema-free XQuery, relax the requirements of specifying the exact path, but rely more heavily on the database management systems to detect the least common ancestors of the keywords specified in the query.

Algebraic research has been fruitful in studying, comparing and characterizing fragments of path queries [2,9], as well as methods and techniques for optimizing and evaluating path queries.

Foundations

Semi-structured data, for example, XML, consists of data entries and containment relationships among the data entries. The data, usually referred to as *a document*, is frequently represented by an ordered node-labeled tree or graph, depending on whether only the containment relationships are treated as first-class relationship, or the id-reference relationships among the data entries are also treated as first-class relationships. The tree representation is more popular:

A *document* D is a 3-tuple (V, Ed, λ), with V the finite set of nodes, $Ed \subseteq V \times V$ a set of parent-child edges, and λ: $V \rightarrow \mathcal{L}$ a node-labeling function into a countably infinite set of labels $\mathcal{L}$.

In addition, even though not always substantial, the ordering among siblings is usually an important feature for nodes in a *document*. The pre-order among the data entries is called the *document order*. Among others, the pre-order is a dominant approach used to identify data entries in a document, while the document is stored in a database system, relational or native.

The aim of the path query is to express the precise requirement of retrieving a set of data entries that satisfy certain value and structural requirements.

The algebraic form of a path query consists of the primitives of $\emptyset$, ε, $\widehat{l}(l \in \mathcal{L})$ for token matching, ↑ and ↓ for upward and downward navigation, and operations $\diamond$ for composition of two algebraic expression $E_1 \diamond E_2$, Π_1 and Π_2 for the first and second projection of an algebraic expression, and set operations $\cap$, $\cup$, and $-$. Given a document $D = (N, Ed, \lambda)$ and a path query E, the path semantics of $E(D)$ is a binary relation:

$$\begin{aligned}
\emptyset(D) &= \emptyset \\
\epsilon(D) &= (n, n) \mid n \in N\} \\
\widehat{l}(D) &= \{(n, n) \mid n \in N \ and \ \lambda(n) = l\} \\
\downarrow(D) &= Ed \\
\uparrow(D) &= Ed^{-1} \\
\Pi_1(E)(D) &= \pi_1(E(D)) \\
\Pi_2(E)(D) &= \pi_2(E(D)) \\
E_1 \diamond E_2(D) &= \pi_{1,4}\sigma_{2=3}(E_1(D) \times E_2(D)) \\
E_1 \cap E_2(D) &= E_1(D) \cap E_2(D) \\
E_1 \cup E_2(D) &= E_1(D) \cup E_2(D) \\
E_1 - E_2(D) &= E_1(D) - E_2(D)
\end{aligned}$$

These primitives and operations identify the path of navigation in a document to reach the resultant data entries.

The $\uparrow^*$ and $\downarrow^*$ are frequently used to identify the ancestor-descendant relationship among data entries along a path:

$$\downarrow^*(D) = \bigcup_{i=0..height(D)} \underbrace{\downarrow \ldots \downarrow}_{i}(D)$$

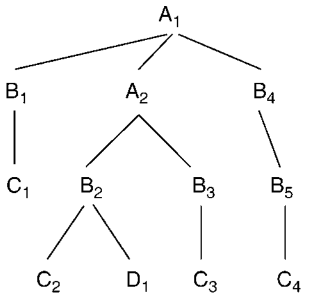
$$\uparrow^*(D) = \bigcup_{i=0..height(D)} \underbrace{\uparrow \ldots \uparrow}_{i}(D)$$

The result of a path query against document D under the *path semantics* is a set of node pairs whose neighborhood data entries and structures satisfy the query expression.

A localized semantics, also called the *node semantics* of a path query Q is to apply the path expression to a document D and a specific node n_0 in the document, such that $Q(D, n_0) = \{n|(n_0, n) \in Q(D)\}$. Usually, the results are presented in a list that honors the *document order*.

For example, assuming document D is represented as a tree structure as shown in Fig. 1, some sample path queries are evaluated as follows:

Path Query	Sample Result
$Q_1 = \downarrow \diamond B$	$Q_1(D) = \{(A_1, B_1), (A_1, B_4), (A_2, B_2), (A_2, B_3), (B_4, B_5)\}$ $Q_1(D, A_1) = \{B_1, B_4\}$
$Q_2 = \Pi_1(\downarrow \diamond \downarrow \diamond C)$	$Q_2(D) = \{(A_1, A_1), (A_2, A_2), (B_4, B_4)\}$ $Q_2(D, A_2) = \{A_2\}$ $Q_2(D, B_4) = \{B_4\}$
$Q_3 = \epsilon \diamond \Pi_1(\downarrow \diamond \downarrow \diamond C) \diamond \downarrow \diamond B \diamond \downarrow$	$Q_3(D) = \{(A_1, C_1), (A_1, B_5), (A_2, C_2), (A_2, D_1), (A_2, C_3), (B_4, C_4)\}$ $Q_3(D, A_1) = \{C_1, B_5\}$ $Q_3(D, B_4) = \{C_4\}$ $Q_3(D, B_2) = \emptyset$
$Q_4 = \uparrow \diamond \Pi_1(\downarrow^* \diamond D) \diamond \Pi_1(\downarrow \diamond \Pi_1(\downarrow \diamond B))$	$Q_4(D) = \{(B_1, A_1), (A_2, A_1), (B_4, A_1)\}$ $Q_4(D, B_1) = \{A_1\}$
$Q_5 = \epsilon \diamond \Pi_1((\downarrow \diamond B) \cup (\uparrow \diamond A)) \diamond \downarrow \diamond \downarrow$	$Q_5(D) = \{(A_2, C_2), (A_2, D_1), (A_2, C_3)\}$ $Q_5(D, A_2) = \{C_2, D_1, C_3\}$



Path Query. Figire 1. A sample semi-structured document presented in a tree structure (subscripts are used to distinguish data entries with the same label).

Path Query and Pattern Tree Matching

Path queries are frequently represented as tree structure, called *pattern trees*, in which nodes (optionally labeled) represents the query requirement on the data entries along the path, and edges (labeled with parent-child or ancestor-descendant relationship) represent the structural requirements among the data entries.

The evaluation of a path query is also called the process of *pattern matching*, which is to find the *witness trees* that consist of data entries that satisfy both the node and structural constraint expressed by the pattern tree.

Path Query Languages

Path query is the core of various query languages for semi-structured data.

XPath [6] is a W3C standard for addressing part of an XML document and for matching and testing whether a node satisfies a pattern. The primary syntactic construct in XPath is the expression, which is evaluated to yield a node set, a boolean value, a number, or a string. The core of the XPath query language is the path query, which is called *location paths* in XPath. Location path consists of relative location paths and absolute location path. A relative location path consists of a sequence of one or more location steps separated by "/". A step is composed from left to right and each step selects a set of nodes relative to a context node, which will in turn serve as the context node for the following step. An absolute location path consists of a leading "/", followed optionally by relative location paths.

XPath, in turn, is the core of other XML query languages and transformation languages, such as XSLT [5] and XQuery [4].

Path Query Evaluation

Even though the way in which the document is stored has great impact on how a path query can be evaluated, some common challenges exist in the evaluation of path queries, comparing to their relational peers, and numerous new operators, optimization techniques and index structures have been proposed to facilitate efficient path query evaluation.

One major characteristic of path query is that the query requirements are expressed not only on data entries, such as the tag (label), attributes, and text values of the entries, but also on the structural relationship among the data entries, which are highlighted by the primitives $\uparrow$ (parent axis), $\downarrow$ (child axis), $\uparrow^*$ (ancestor axis), and $\downarrow^*$ (descendant axis).

Navigation is a natural approach for evaluating a path query. The nagivational approach scans the whole document to verify the query requirement on data entries and the structural relationship among data entries. This approach works more naturally while the document is stored as file, in object-oriented database, or in a native data store. In addition, the nagivational approach is potentially very expensive in cost, especially when the ancestor/descendent axis is involved.

The invention of structural join operator [1,13] and multiple algorithms for efficient computing of structural join advanced the evaluation technique of path queries dramatically. A structural join operation takes two sets of data entries as input and returns pairs of data entries that satisfy the desired structural relationship. The basis of the structural join operation is the pre-order and post-order numbering encoding of data entries in the document. The parent-child relationship is a special case of the ancestor-descendant relationship, in which the nodes that satisfies the desired structural relationship have to be exactly one level apart from each other.

$$(a,b) \in \downarrow^*(D) \Leftrightarrow a_{pre} < b_{pre} \wedge a_{post} > b_{post}$$
$$(a,b) \in \downarrow (D) \Leftrightarrow a_{pre} < b_{pre} \wedge a_{post}$$
$$b_{post} \wedge a_{level} + 1 = b_{level}$$

The structure join operation and various algorithms that support efficient evaluation of the operation enables the evaluation of path queries by decomposition. This approach partitions a path query into twigs that consists of either a node or a pair of nodes with a desired parent-child or ancestor-descendant relationship. The matching of the nodes can be easily evaluated via indices that are similar to value indices in RDB. The structural relationships between node pairs are evaluated by the structural join. Holistic approach has been proposed and widely adopted in answering path queries. This approach uses a chain of linked stacks to compute and represent intermediate results of a path in a compactly fashion.

Path Query Optimization

As any database management system, optimization is a critical step in evaluating path queries.

Syntax based optimization rewrites a path query into a path query in normal format, in pursuit of minimum expression length, minimum number of predicates, and no redundant value and structural requirements. The rewrite may also aim at decomposing a path query into sub-queries that belong to some fragments of path queries that are simpler to answer.

The cost-based optimization relies on the data statistics of the document to be queried, and the cost-model of the physical operators to be employed, to enumerate various physical evaluation plans and choose one with minimum or acceptable estimate performance. This process involves the study of access method selection, query decomposition, cost estimation, etc.

Indices for Path Query Evaluation

Indexing, being one of the most important ingredients in efficient query evaluation, has seen its importance in the context of XML. Over 20 different types of indices have been proposed and have led to significant improvements in the performance of XML query evaluation.

Indices similar to the ones used in RDBs, namely value indices on element tags, attribute names and text values, are first used, together with the structural join algorithms [1,3,10,13], in XML query evaluation. This approach turns out to be simple and efficient, but is not capable of capturing the structural containment relationships native to the XML data.

To directly capture the structural information of XML data, a family of structural indices has been introduced. DataGuide [7] was the first to be proposed, followed by the 1-index [11], which is based on the notion of bi-simulation among nodes in an XML document. These indices can be used to evaluate some path expressions accurately without accessing the original data graph. Milo and Suciu [11] also introduced the 2-index and T-index, based on similarity of pairs (vectors) of nodes. Unfortunately, these and other early structural indices tend to be too large for practical use because they typically maintain too fine-grained structural information about the document. To remedy this, Kaushik et al. introduced the $A(k)$-index which uses a notion of bi-similarity on nodes relativized to paths of length k [8]. This captures localized structural information of a document, and can support path expressions of length up to k. Focusing just on local similarity, the $A(k)$-index can be substantially smaller than the 1-index and others. Several works have investigated maintenance and tuning of the $A(k)$ indices. The $D(k)$-index and $M(k)$-index extend the $A(k)$-index to adapt to query workload. The integrated use of structural and value indices has been explored, and there have also been investigations on covering indices and index selection.

Other directions of XML indexing techniques proposed by researchers include indexing frequent subpatterns, indexing XML tree and queries as sequences, forward and backward index, HOPI index, XR-tree, and encoding-based indices.

Key Applications

Path query is the core concept behind the query languages for semi-structured data. It is also the foundation of path algebra that are used to represent and reason about queries expressed against semi-structured data, especially those focusing on retrieving fragments of the document that satisfy certain value and structural constraints.

Future Directions

Even though the basic idea of the path query has been around for decades, its popularity exploded since XML prevails.

Path query has been adopted as the core of expressing queries again semi-structural data, especially XML. Even though XPath query language has been very stable, new language and language features keep emerging. As to the path query itself, there are on-going study of the sub-languages and the characteristics of such languages, their relationship to each other, the decidability of the queries in these languages, and the complexity of answer the queries.

On the practical side, systems have been developed to answer path queries. Various query evaluation, query optimization and indexing techniques have been proposed to facilitate efficient evaluation of path queries. However, the level of maturity of these techniques are not at the same level as those of relational queries.

In the relational world, the use cases are well understood and various benchmarks have been developed to measure the performance of relational DBMSs. Those of the queries on semi-structured documents are less understood, despite the existence of a few XML benchmarks, such as XMark [12]. In depth research on the usage of path queries and the design and development of benchmarks is yet another promising direction.

Data Sets

Example semi-structured data and queries can be found in benchmarks such as XMark (http://www.xml-benchmark.org), XMach (http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html), and MBench (http://www.eecs.umich.edu/db/mbench).

Cross-references

- ▶ Bi-Similarity
- ▶ Semi-Structured Data Model
- ▶ XML Data Management
- ▶ XPath/XQuery
- ▶ XSL/XSLT

Recommended Reading

1. Al-Khalifa S., Jagadish H.V., Patel J.M., Koudas N., Srivastava D., and Wu Y. Structural joins: a primitive for efficient XML query pattern matching. In Proc. 18th Int. Conf. on Data Engineering, 2002.
2. Benedikt M., Fan W., and Kuper G.M. Structural properties of XPath fragments. Theor. Comput. Sci., 226(1):3–31, 2005.

3. Bruno N., Koudas N., and Srivastava D. Holistic twig joins: optimal XML pattern matching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 310–321.
4. Chamberlin D., Clark J., Florescu D., Robie J., Simeon J., and Stefanescu M. XQuery 1.0: an XML Query Language, May 2003.
5. Clark J. XSL Transformations (XSLT) version 1.0. http://www.w3.org/TR/XSLT
6. Clark J. and DeRose D. XML Path Language (XPath) version 1.0. http://www.w3.org/TR/XPATH
7. Goldman R. and Widom J. Data Guides: enabling query formulation and optimization in semistructured databases. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 436–445.
8. Kaushik R., Shenoy P., Bohannon P., and Gudes E. Exploiting local similarity for efficient indexing of paths in graph structured data. In Proc. 18th Int. Conf. on Data Engineering, 2002.
9. Koch C. Processing queries on tree-structured data efficiently. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems. 2006, pp. 213–224.
10. McHugh J. and Widom J. Query optimization for XML. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 315–326.
11. Milo T. and Suciu D. Index structures for path expressions. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 277–295.
12. Schmidt A., Waas F., Kersten M.L., Carey M.J., Manolescu I., and Busse R. XMark: a benchmark for XML data management. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 974–985.
13. Zhang C., Naughton J.F., DeWitt D.J., Luo Q., and Lohman G.M. On supporting containment queries in relational database management systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001.

Pattern Based Clustering

▶ Subspace Clustering Techniques

Pattern Discovery

▶ Data Mining

Pattern-Growth Methods

Hong Cheng[1], Jiawei Han[2]
[1]Chinese University of Hong Kong, Hong Kong, China
[2]University of Illinois-Urbana-Champaign, Urbana, IL, USA

Definition

Pattern-growth is one of several influential frequent pattern mining methodologies, where a pattern (e.g., an itemset, a subsequence, a subtree, or a substructure) is *frequent* if its occurrence frequency in a database is no less than a specified *minimum_support* threshold. The (frequent) pattern-growth method mines the data set in a divide-and-conquer way: It first derives the set of size-1 frequent patterns, and for each pattern *p*, it derives *p*'s projected (or conditional) database by data set partitioning and mines the projected database recursively. Since the data set is decomposed progressively into a set of much smaller, pattern-related projected data sets, the pattern-growth method effectively reduces the search space and leads to high efficiency and scalability.

Historical Background

Frequent itemset mining was first introduced as an essential subtask of association rule mining by Agrawal et al. [1]. A candidate set generation-and-test approach, represented by the Apriori algorithm, was proposed by Agrawal and Srikant [2]. The approach effectively reduces the search space by exploring the *downward closure* property of frequent patterns, i.e., *any subpattern of a frequent pattern is frequent*. Various kinds of refinements of and extensions to this approach were proposed afterwards. However, since the Apriori-like candidate set generation-and-test approach repeatedly scans the whole database and checks the candidates by pattern matching, it is still rather costly.

The pattern-growth approach, represented by the *FP-growth* algorithm, was first proposed by Han, Pei, and Yin [8] for mining frequent itemsets. Since then, the method has been developed in several directions: (i) further enhancement of mining efficiency using refined data structures, such as FP-growth* [6], which uses an array-based implementation of prefix-tree-structure of FP-growth; (ii) mining closed and max patterns [6,13], where a pattern *p* is *closed* if there exists no super-pattern with the same support, and *p* is a *max pattern* if there exists no super-pattern that is frequent; (iii) mining sequential patterns [11] and frequent substructures [14]; and (iv) mining high-dimensional data set [7] and colossal patterns [15], and pattern-based classification [4] and clustering [12]. A comprehensive overview of such extensions is presented in [7].

Foundations

Frequent Itemset Mining

To illustrate the pattern-growth method, the FP-growth method is briefly introduced here that exploits pattern-growth in frequent itemset mining. FP-growth works in a divide-and-conquer way. The first scan of the database derives a list of frequent items in which items are ordered by frequency-descending order (Notice that this particular ordering is not essential, and different ordering schemes can be explored). According to this ordering, the database is compressed into a frequent-pattern tree, or *FP-tree*, which retains the itemset association information.

The FP-tree is mined by starting from each frequent length-1 pattern (as an initial suffix pattern), constructing its *conditional pattern base* (a "subdatabase", which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern), then constructing its conditional FP-tree, and performing mining recursively on such a tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree. Figure 1 shows an example of a global FP-tree as well as a set of conditional trees and the recursive mining process on top of them. Therefore, the FP-growth algorithm transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix.

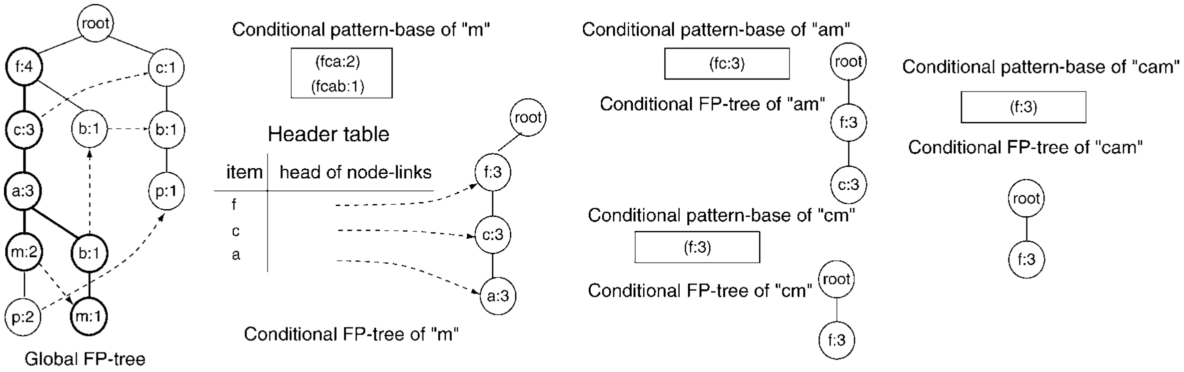
Sequential Pattern Mining

The pattern-growth philosophy has been extended to sequential pattern mining, where a sequential pattern is a set of (gapped) subsequences that occur frequently in a set of sequences. PrefixSpan, developed by Pei et al. [11], is a typical sequential pattern mining algorithm based on the pattern-growth approach. It works in a divide-and-conquer way, by first scanning the sequence database to derive the set of length-1 sequential patterns. Then each sequential pattern is treated as a prefix and the complete set of sequential patterns can be partitioned into different subsets according to different prefixes. To mine the subsets of sequential patterns, corresponding projected databases are constructed and mined recursively.

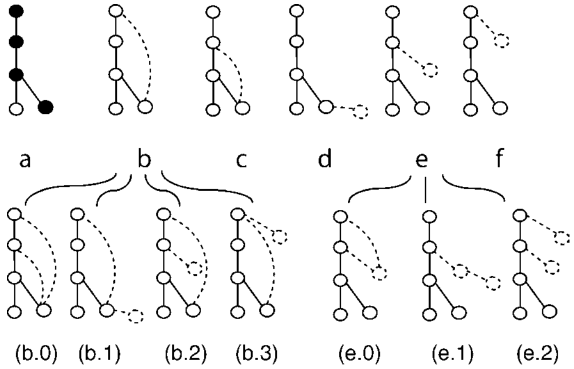
Frequent Subgraph Mining

Inspired by the pattern-growth-based frequent itemset and sequential pattern mining algorithms, there are quite a few pattern-growth-based graph pattern mining algorithms developed. Here gSpan [14] is used an example to explain the ideas.

The pattern-growth graph mining algorithm extends a frequent graph by adding a new edge, in every possible position. A potential problem with the edge extension is that the same graph can be discovered many times. gSpan solves this problem by introducing a new lexicographic order among graphs, and maps each graph to a unique minimum DFS code as its canonical label. Based on this lexicographic order, gSpan adopts the pattern-growth philosophy with a *right-most extension* technique, where the only extensions take place on the *right-most path*. A right-most path is the straight path from the starting vertex v_0 to the last vertex v_n, according to a depth-first search on the graph. In Fig. 2, the graph shown in 2a has several potential children with one edge growth, which are shown in 2b–f (assume the darkened vertices constitute the rightmost path). Among them, 2b–d grow from the rightmost vertex while 2e and 2f grow from other vertices on the rightmost path. 2(b.0)–(b.3) are children of 2b, and 2(e.0)–(e.2) are children of 2e. Backward edges can only grow from the rightmost vertex while forward edges can grow from vertices on the rightmost path. The enumeration order of these children is enhanced by the DFS lexicographic order, i.e., it should be in the order of 2b–f.


root
f:4
c:1
c:3
b:1
b:1
a:3
p:1
m:2
b:1
p:2
m:1
Global FP-tree
Conditional pattern-base of "m"
(fca:2)
(fcab:1)
Header table
item
head of node-links
f
c
a
root
f:3
c:3
a:3
Conditional FP-tree of "m"
Conditional pattern-base of "am"
(fc:3)
root
Conditional FP-tree of "am"
f:3
c:3
Conditional pattern-base of "cm"
(f:3)
Conditional FP-tree of "cm"
root
f:3
Conditional pattern-base of "cam"
(f:3)
Conditional FP-tree of "cam"
root
f:3

Pattern-Growth Methods. Figure 1. FP-growth mining on conditional FP-trees.



Pattern-Growth Methods. Figure 2. Graph growth with DFS lexicographic order.

With the DFS lexicographic order definition, the frequent subgraph mining is performed in a *DFS Code Tree*. In a DFS Code Tree, each node represents a DFS code. The relation between parent and child node complies with the parent–child relation; the relation among siblings is consistent with the DFS lexicographic order. Figure 3 shows a DFS Code Tree, the nth level nodes contain DFS codes of $(n-1)$-edge graphs. Through depth-first search of the code tree, all the minimum DFS codes of frequent subgraphs can be discovered. That is, all the frequent subgraphs can be discovered in this way. One should note that if in Fig. 3, the dark nodes contain the same graph but different DFS codes, then one of them must not be the minimum code. Therefore, the search space of that sub-branch can be pruned since it does not correspond to a minimum DFS code.

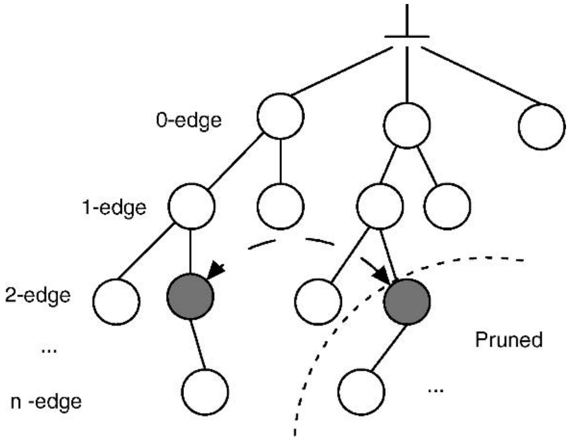
Key Applications

Pattern-growth methods have been used in many tasks that need the mining of frequent patterns, including the discovery of association and correlation relationships among large sets of transactions, event sequences, or complex structures, the discovery of discriminative frequent features for classification and clustering, as well as many other data mining and pattern recognition applications, such as biological data mining.

Future Directions

Mining Approximate or Noise-Tolerant Patterns

Pattern-growth method has been widely applied for efficient mining of precise and complete set of frequent patterns. However, in some real applications, the capacity to accommodate approximation in the mining process has become critical due to inherent noise and



Pattern-Growth Methods. Figure 3. A search space: DFS code tree.

imprecision in complex data sets, for example, gene mutations in genomic DNA sequences, and protein–protein interaction networks. Approximate or noise-tolerant frequent patterns could be the natural choice to handle noise or variations in many applications. Some recent studies proposed new algorithms for mining approximate itemsets [9] and subgraphs [3]. Among them, [9] adopts a candidate generation-and-test approach with the noisy mining model while [3] takes a pattern-growth approach. It is interesting to explore whether pattern-growth method could be naturally applied into a noisy mining model to efficiently discover the approximate or noise-tolerant patterns.

Pattern-Based Classification and Clustering

Frequent patterns have been demonstrated useful in other data mining tasks such as classification [4] and clustering [12], where frequent patterns are used as discriminative classification features and clustering subspaces, respectively. Instead of using the complete set of patterns, usually only a small number of frequent patterns are used in classification and clustering tasks, e.g., the subset of highly discriminative patterns in classification and the subset of frequent patterns with dense regions in clustering. It would be desirable to directly mine the subset of patterns of interests without generating the complete set, for efficiency consideration. To achieve this goal, pruning strategies need to be designed and integrated into the pattern-growth mining methodology to effectively prune the search space which does not yield high-quality patterns. This is a non-trivial task since many quality measures, such as information gain, density, or entropy, are not *anti-monotonic* which is the essential pruning strategy in frequent pattern mining.

Experimental Results

In general, for every proposed method, there is an accompanying experimental evaluation in the corresponding reference. In addition, for frequent itemset mining methods, [5] (The FIMI workshop) provided a detailed and comprehensive experimental evaluation on a large set of benchmark data.

Data Sets

Synthetic Data

A synthetic tree generator can be found at http://www.cs.rpi.edu/∼zaki/software

Real Data

A large collection of real transaction datasets can be found at http://fimi.cs.helsinki.fi/data/. Commonly used real graph data includes AIDS anti-viral screening datasets at http://dtp.nci.nih.gov, and NCI anti-cancer screening datasets at http://pubchem.ncbi.nlm.nih.gov.

URL to Code

The binary codes for FP-growth, PrefixSpan and gSpan are provided by the IlliMine project at http://illimine.cs.uiuc.edu/.

The source codes of FP-growth*, FPClose, and FPMax* [6] are provided by Grahne and Zhu at http://fimi.cs.helsinki.fi/src/fimi06.tgz

Cross-references

- ▶ Apriori property and Breadth-First Search Algorithms
- ▶ Frequent Graph Patterns
- ▶ Frequent Itemsets and Association Rules
- ▶ Sequential Patterns

Recommended Reading

1. Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In Proc. ACM-SIGMOD Int. Conf. on Management of Data, 1993, pp. 207–216.
2. Agrawal R. and Srikant R. Fast algorithms for mining association rules. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
3. Chen C., Yan X., Zhu F., and Han J. gApprox: Mining frequent approximate patterns from a massive network. In Proc. 2007 IEEE Int. Conf. on Data Mining, 2007, pp. 445–450.
4. Cheng H., Yan X., Han J., and Yu P.S. Direct discriminative pattern mining for effective classification. In Proc. 24th Int. Conf. on Data Engineering, 2008.
5. Goethals B. and Zaki M. An introduction to workshop on frequent itemset mining implementations. In Proc. ICDM Int. Workshop on Frequent Itemset Mining Implementations, 2003, pp. 1–13.
6. Grahne G. and Zhu J. Efficiently using prefix-trees in mining frequent itemsets. In Proc. ICDM Int. Workshop on Frequent Itemset Mining Implementations, 2003.
7. Han J., Cheng H., Xin D., and Yan X. Frequent pattern mining: Current status and future directions. Data Mining and Knowledge Discovery, 15:55–86, 2007.
8. Han J., Pei J., and Yin Y. Mining frequent patterns without candidate generation. In Proc. ACM-SIGMOD Int. Conf. on Management of Data, 2000, pp. 1–12.
9. Liu J., Paulsen S., Sun X., Wang W., Nobel A., and Prins J. Mining approximate frequent itemsets in the presence of noise: Algorithm and analysis. In Proc. SIAM Int. Conf. on Data Mining, 2006, pp. 405–416.
10. Pan F., Cong G., Tung A.K.H., Yang J., and Zaki M. CARPENTER: Finding closed patterns in long biological datasets. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 637–642.
11. Pei J., Han J., Mortazavi-Asl B., Wang J., Pinto H., Chen Q., Dayal U., and Hsu M.-C. Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Trans. Knowl. Data Eng., 16:1424–1440, 2004.
12. Pei J., Zhang X., Cho M., Wang H., and Yu P.S. Maple: A fast algorithm for maximal pattern-based clustering. In Proc. IEEE Int. Conf. on Data Mining, 2001, pp. 259–266.
13. Wang J., Han J., and Pei J. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 236–245.
14. Yan X. and Han J. gSpan: Graph-based substructure pattern mining. In Proc. 2002 IEEE Int. Conf. on Data Mining, 2002, pp. 721–724.
15. Zhu F., Yan X., Han J., Yu P.S., and Cheng H. Mining colossal frequent patterns by core pattern fusion. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 706–715.

PCA

- ▶ Principal Component Analysis

PDMS

- ▶ Peer Data Management System

Pedigree

- ▶ Provenance
- ▶ Provenance in Scientific Databases

Peer Data Management

▶ Structured Data in Peer-to-Peer Systems

Peer Data Management System

Philippe Cudré-Mauroux
Massachussetts Institute of Technology, Cambridge, MA, USA

Synonyms

PDMS; Decentralized data integration system

Definition

A Peer Data Management System (PDMS) is a triple $\mathcal{S} = \langle \mathcal{P}, \mathcal{S}, \mathcal{M} \rangle$ where $\mathcal{P}$ is a set of autonomous peers, $\mathcal{S}$ a set of heterogeneous schemas used by the peers to represent their data, and $\mathcal{M}$ a set of schema mappings, each enabling the reformulation of queries between a given pair of schemas.

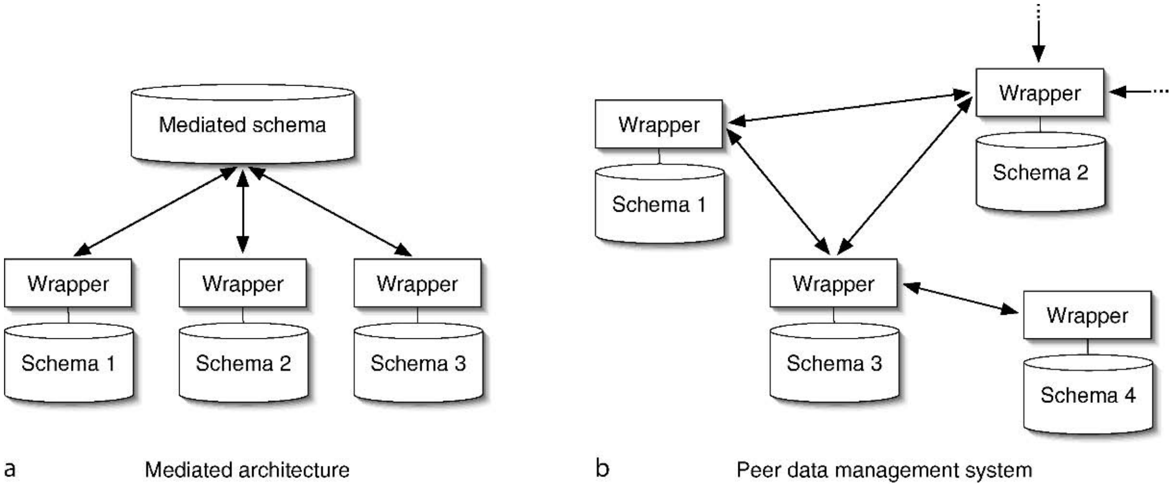
Key Points

A Peer Data Management System (PDMS) is a distributed data integration system providing transparent access to heterogeneous databases without resorting to a centralized logical schema. Instead of imposing a uniform query interface over a mediated schema, PDMSs let the peers define their own schemas and allow for the reformulation of queries through mappings relating pairs of schemas (see Fig. 1). PDMSs typically exploit the schema mappings transitively in order to retrieve results from the entire network.

Compared to centralized data integration systems, PDMSs suggest a scalable, decentralized and easily extensible integration architecture where any peer can contribute data, schemas, and mappings. Peers with new schemas simply need to provide a mapping between their schema and any other schema already used in the system to be part of the network.

The languages used to define the mappings in PDMSs may vary, but are typically derived from GLAV formulae with extensions to support both inclusion and equality mappings. The Piazza system [3] proposes new algorithms to retrieve certain answers in this context. Hyperion [1] is a system focusing on relating data not only at the schema level, but also at the instance level through mapping tables. GridVine [2] provides distributed probabilistic analyses in order to automatically detect mapping inconsistencies in PDMS settings.

PDMSs generally use Peer-to-Peer overlay networks to support their distributed operations. Some use unstructured overlay networks [1,3] to organize the peers into a random graph and use flooding or random walks to contact distant peers. Other PDMSs maintain a decentralized yet structured Peer-to-Peer network [2] to allow any peer to contact any other peer by taking advantage of a distributed index.



Peer Data Management System. Figure 1. Contrary to the mediated architecture (a), Peer Data Management Systems (b) do not impose any form of centralization but consider instead networks of heterogeneous data sources related to each other through pairwise schema mappings.

The lack of global coordination has raised several questions as to the global properties of such systems in the large. In particular, new complex systems perspectives – such as the emergent semantics approach – have been proposed to characterize the global semantics of PDMSs.

Cross-references

- ▶ Data Integration
- ▶ Emergent Semantics
- ▶ Peer-to-Peer Data Integration
- ▶ Peer to Peer Overlay Networks: Structure, Routing and Maintenance

Recommended Reading

1. Arenas M., Kantere V., Kementsietsidis A., Kiringa I., Miller R.J., Mylopoulos J. The hyperion project: from data integration to data coordination. ACM SIGMOD Rec., 32(3):53–58, 2003.
2. Cudré-Mauroux P. Emergent Semantics. EPFL Press, 2008.
3. Halevy A., Ives Z., Madhavan J., Mork P., Suciu D., Tatarinov I. The piazza peer data management system. IEEE Trans. Knowl. Data Eng., 16(7):787–798, 2004.

Peer Database Management

▶ Structured Data in Peer-to-Peer Systems

Peer to Peer Network

▶ Storage Grid

Peer to Peer Overlay Networks: Structure, Routing and Maintenance

Wojciech Galuba, Sarunas Girdzijauskas
EPFL, Lausanne, Switzerland

Definition

A peer-to-peer overlay network is a computer network built on top of an existing network, usually the Internet. Peer-to-peer overlay networks enable participating peers to find the other peers not by the IP addresses but by the specific logical identifiers known to all peers. Usually, peer-to-peer overlays have the advantage over the traditional client-server systems because of their scalability and lack of single-point-of-failure. Peer-to-peer overlays are commonly used for file sharing and realtime data streaming.

Historical Background

The rise of the Internet brought the first instances of peer-to-peer overlays like the Domain Name System (DNS), the Simple Mail Transfer Protocol (SMTP), USENET and more recently IPv6, which were needed to facilitate the operation of the Internet itself. These peer-to-peer overlays were intrinsically decentralized and represented symmetric nature of the Internet, where every node in the overlay had equal status and assumed cooperative behavior of the participating peers. The beginning of the file-sharing era and the rise and fall of the first file-sharing peer-to-peer system Napster [9] (2000–2001) paved the way for the second generation of peer-to-peer overlays like Gnutella [5] (2000) and Freenet [4] (2001). The simple protocols and unstructured nature made these networks robust and lacking Napster's drawbacks like single-point-of-failure. Since 2001, these peer-to-peer overlays became extensively popular and accounted for the majority of the Internet traffic. Soon after it was evident that the unstructured nature of Gnutella-like systems is embarrassingly wasteful in bandwidth, more efficient structured overlays appeared, like the Distributed Hash Tables (DHTs), which used the existing resources more effectively (e.g., Chord [13]). Currently, unstructured peer-to-peer overlays are sparsely used, as the most popular peer-to-peer applications for file-sharing and data-streaming (e.g., Skype [12], Kademlia [8], KaZaA [6]) are implemented using structured or hybrid overlay concepts.

Foundations

Taxonomy

There are many features of peer-to-peer overlays, by which they can be characterized and classified [2,11]. However, strict classification is not easy since many features have mutual dependencies on each other, making it difficult to identify the distinct overlay characteristics (e.g., overlay topologies versus routing in overlays). Although every peer-to-peer overlay can differ by many parameters, but each of them will have to have certain network structure with distinctive routing and maintenance algorithms allowing the peer-to-peer application to achieve its purpose. Thus, most commonly, peer-to-peer overlays can be classified by:

1. Purpose of use;
2. Overlay structure;
3. Employed routing mechanisms;
4. Maintenance strategies.

Purpose of Use

Peer-to-peer overlays are used for an efficient and scalable sharing of individual peers' resources among the participating peers. Depending on the type of the resources which are shared, the peer-to-peer overlays can be identified as oriented for:

1. Data-sharing (data storage and retrieval);
2. Bandwidth-sharing (streaming);
3. CPU-sharing (distributed computing).

Data-sharing peer-to-peer overlays can be further categorized by their purpose to perform one or more specific tasks like file-sharing (by-far the most common use of the peer-to-peer overlays), information retrieval (peer-to-peer web search), publish/subscribe services and semantic web applications. The examples of such networks are BitTorrent [3] (file-sharing), YaCy-Peer [14] (web search), etc.

Bandwidth-sharing peer-to-peer overlays to some extent are similar to the data-sharing ones, however, are mainly aimed at the efficient streaming of real-time data over the network. Overlay's ability to find several disjoint paths from source to destination can significantly boost the performance of the data streaming applications. Bandwidth-sharing peer-to-peer overlays are mostly found in peer-to-peer telephony, peer-to-peer video/TV, sensor networks and peer-to-peer publish/subscribe services. Currently Skype [12] is arguably the most prominent peer-to-peer streaming overlay application.

For the computationally intensive tasks, when the CPU resources of a single peer cannot fulfill its needs, a *CPU-sharing* peer-to-peer overlays can provide plenty of CPU resources from the participating idle overlay peers. Currently, only a major scientific experiments employ such strategy for the tasks like simulation of protein folding or analysis of an astronomic radio signals. Although *not* being a pure peer-to-peer overlay, Berkeley Open Infrastructure for Network Computing (BOINC) is very popular among such networks, supporting such distributed computing projects as SETI@home, folding@home, AFRICA@home, etc.

Overlay Structure

Peer-to-peer overlays significantly differ by the topology of the networks which they form. There exist a wide scope of possible overlay instances, ranging from centralized to purely decentralized ones, however, most commonly, three classes of network topology are identified:

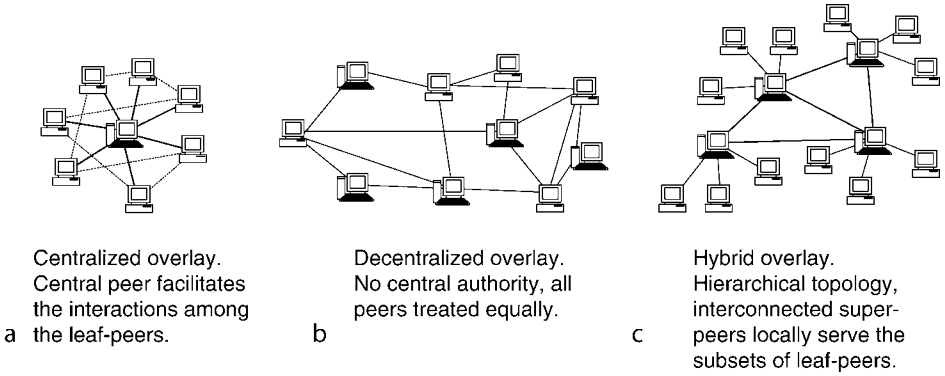
1. Centralized overlays;
2. Decentralized overlays;
3. Hybrid overlays.

Depending on the routing techniques and whether the overlay network was created by some specific rules (deterministically) or in ad hoc fashion (nondeterministically), overlay networks can be also classified into *structured* and *unstructured* peer-to-peer overlays.

Centralized Overlays Peer-to-peer overlays based on *centralized* topologies are pretty efficient since the interaction between peers is facilitated by a central server which stores the global index, deals with the updates in the system, distributes tasks among the peers or quickly responds to the queries and give complete answers to them (Fig. 1(a)). However, not all the purposes of use fit the centralized network overlay model. Centralized overlays usually fail to scale with the increase of the number of participating peers. The centralized component rapidly becomes the performance bottleneck. The existence of a single-point-of-failure (e.g., Napster [9]) also prevents from using centralized overlays for many potential data-sharing applications.

Decentralized Overlays Because of the aforementioned drawbacks, *decentralized* structured and unstructured overlays emerged, which use purely decentralized network model, and do not differ peers as servers or clients, but treat all of them equally – as they were both servers and clients at the same time (Fig. 1(b)). Thus, such peer-to-peer overlays successfully deal with the scalability and can exist without any governing authority.

The simplest decentralized overlays usually are *unstructured*. Unstructured overlay networks typically have arbitrary topology and use flooding based routing among the peers. The distribution of the resources among the peers is completely unrelated to the



a Centralized overlay. Central peer facilitates the interactions among the leaf-peers.

b Decentralized overlay. No central authority, all peers treated equally.

c Hybrid overlay. Hierarchical topology, interconnected super-peers locally serve the subsets of leaf-peers.

Peer to Peer Overlay Networks: Structure, Routing and Maintenance. Figure 1. Examples of peer-to-peer overlays.

network topology. Because of their simplicity, the unstructured overlays are pretty robust to network and peer failures, although are rather inefficient in bandwidth consumption and have poor querying performance.

Structured overlay networks, however, use more efficient routing techniques and the topology of the structured overlays is not arbitrary but typically exhibit Small-World properties, specifically high clusterization and low network diameter. The link establishment among the peers is usually strictly defined by the specific protocols. The topologies can result in various structures like rings, toruses, hypercubes and de-Bruin networks or more loose randomized networks, which do have properties of Small-World networks. A particular instance of structured peer-to-peer overlays is a Distributed Hash Table (DHT) enabling an efficient lookup service, by using a predefined hashing algorithms to assign an ownership for a particular resource (e.g., Chord [13], P-Grid [1], Symphony [7], etc.). In contrast to the traditional Hash Table, the DHTs share the global hashing information among all the participating peers equally and the DHT protocols ensure that any part of a global hash table is easily reachable (usually in logarithmic steps) and there is enough replication to sustain the consistency in the system.

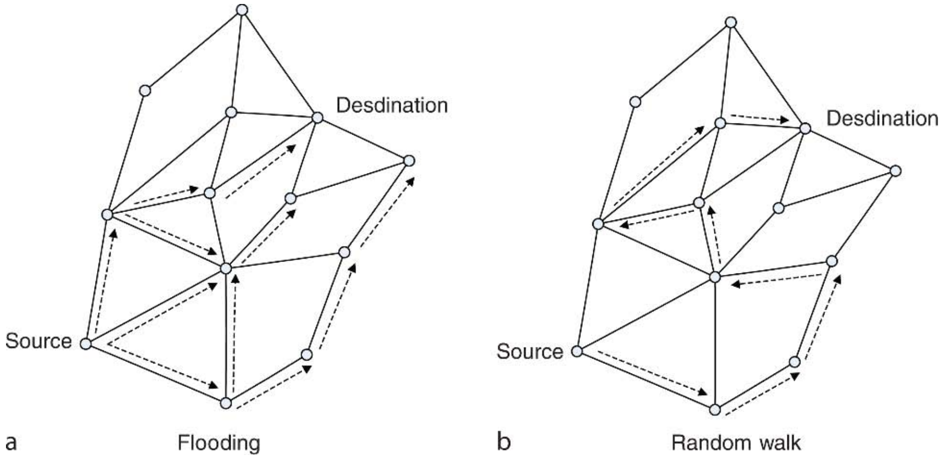
Hybrid Overlays There also exist many *hybrid* peer-to-peer overlays (super-peer systems) which trade-off between different degree of topology centralization and structure flexibility. Hybrid overlays usually use hierarchical network topology consisting of regular peers and super-peers, which act as local servers for the subsets of regular peers (Fig. 1(c)). For example, a hybrid overlay might consist of the super-peers forming a structured network which serves as a backbone for the whole overlay, enabling an efficient communication among the super-peers themselves. Hybrid overlays have advantage over simple centralized networks since the super-peers can be dynamically replaced by regular peers, hence do not constitute single points of failure, but have the benefits of centralized overlays.

Routing

Peer-to-peer overlay networks enable the peers to communicate with one another even if the communicating peers do not know their addresses in the underlying network. For example, in an overlay deployed on the Internet, a peer can communicate with another peer without knowing its IP address. The way it is achieved in the overlays is by *routing* overlay messages. Each overlay message originates at a source and is forwarded by the peers in the overlay until the message reaches one or more destinations. A number of routing schemes have been proposed.

Routing in Unstructured Overlays Unstructured overlay networks use mainly two mechanisms to deliver routed messages: flooding and random walks (Fig. 2). When some peer v receives a flood message from one of its overlay neighbors w, then v forwards the flood message to all of its neighbors except w. When v receives the same flood message again, it is ignored. Eventually the flood reaches all of the destinations.

For example, in Gnutella [10], a file-sharing peer-to-peer system, a peer s that wants to download a file floods the network with queries. If some peer d that has the file desired by s is reached by the query flood, then d sends a response back to s. Flooding consumes a significant amount of network bandwidth. To reduce



Peer to Peer Overlay Networks: Structure, Routing and Maintenance. Figure 2. *Routing in unstructured overlay networks.* The *circles* and *solid lines* represent the overlay topology. The *dashed arrows* illustrate the flow of messages. Peers routing in an unstructured network do not know the exact location of the destinations so they have to either look in all possible directions via flooding or randomly walk to find the destination peer.

it, the flooded messages typically contain a Time-To-Live (TTL) counter included in every message that is decremented whenever the message is forwarded. This limits how far the flood can spread from the source but at the same time lowers the chance of reaching the peer that holds the searched file.

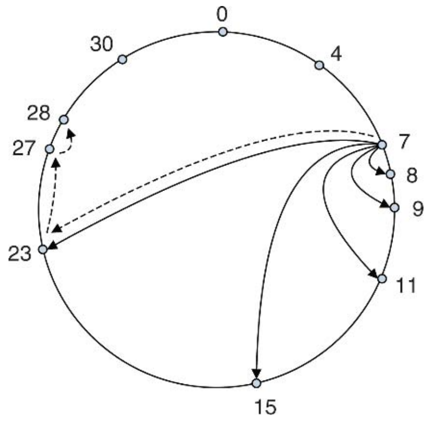
The high bandwidth usage of flooding has led to the design of an alternative routing scheme for unstructured overlay networks: *random walks*. Instead of forwarding a message to all of the neighbors, it is only forwarded to a randomly chosen one. Depending on the network topology random walks provide different guarantees of locating the destination peer(s), however all of the random walk approaches share one disadvantage: a significant and in most cases intolerable delivery latency.

Routing in Structured Overlays As more peers join the overlay network and there are more messages that need to be routed, flooding and random walks quickly reach their scalability limits. This problem has prompted the research on structured overlays.

In structured overlays each peer has a unique and unchanging identifier picked when a peer joins the overlay. The peer identifiers enable efficient routing in the structured overlays. Each routed message has a destination identifier selected from the peer identifier set. Instead of blindly forwarding the message to all neighbors as in the unstructured overlays, a peer in a structured overlay uses the destination identifier to forward the message only to one neighbor. The next hop neighbor is selected to minimize the number of hops to the destination, i.e., the routing is greedy. This selection is made using the *peer identifier distance*.

Most of the modern structured overlays define the notion of distance between any two peer identifiers. For example, in Chord [13] identifiers are selected from the set of integers $[0,2^m - 1]$ and are ordered in a modulo 2^m circle. The distance $d(x, y)$ between two identifiers x and y is defined as the difference between x and y on that identifier circle, i.e., $d(x, y) = (y - x)$ mod 2^m In another overlay, Kademlia [8], the identifiers are 160-bit integers and the distance between two identifiers x and y is defined as their exclusive bitwise OR (XOR) interpreted as an integer, i.e., $d(x, y) = x \oplus y$.

Although the modern structured overlays differ in the details of how they make use of the peer identifiers for routing efficiency, they are all based on the same general *greedy routing* principle. When some peer v receives a message with a given destination identifier it forwards the message to that next hop whose identifier is the closest to the destination identifier. In other words, in every hop the message gets as close as possible to the destination. Routing terminates when TTL is exhausted or one of the peers decides it is the destination for the message. The latter decision is application dependent. For example, in a Distributed Hash Table each peer knows for which hash table keys it is responsible. The key space is mapped onto the peer identifier space in DHTs and the destination identifier of each



Peer to Peer Overlay Networks: Structure, Routing and Maintenance. Figure 3. *Routing in chord.* The *big circle* represents the peer identifier space with IDs in the interval [0,2^5]. The *small circles* are the peers and the number beside them is their ID. Peer 7 is connected (*solid arrows*) to peers with exponentially increasing distance from 7: $7 + 1 = 8, 7 + 2 = 9, 7 + 4 = 11, 7 + 8 = 15, 7 + 16 = 23$. Assume that peer 7 wants to route a message to peer 28. *Dashed arrows* represent the routing path. The peer 23 is the neighbor of 7 that is closest on the ring to the destination 28 and that neighbor is chosen as the first hop for the message. One hop is not enough to reach 28, but the peer 23 brings the message closer to the destination and peer 27 finally delivers it. The greedy routing rule of always selecting such next hop that brings the message as close as possible to its destination is the main building block of all structured overlay networks.

DHT lookup message specifies the hash table key K the lookup is querying for. The lookup message is greedily routed hop by hop from the origin until the message reaches a peer responsible for K. Routing then terminates and the responsible peer sends a lookup response to the origin. The lookup response contains the hash table value stored under the key K.

Maintenance

Peer-to-peer systems are commonly deployed in environments characterized by high dynamicity, peers can depart or join the system at any time. These continuous joins and departures are commonly referred to as *churn*. Instead of gracefully departing from the network peers can also abruptly fail or the network connection with some of its neighbors may be closed. In all of these cases the changes in the routing tables may adversely affect the performance of the system. The overlay topology needs to be *maintained* to guarantee message delivery and routing efficiency.

There are two main approaches to overlay maintenance: proactive and reactive. In *proactive maintenance* peers periodically update their routing tables such that they satisfy the overlay topology invariants. For example, Chord periodically runs a "stabilization" protocol to ensure that every peer is linked to other peers at exponentially increasing distance. This ensures routing efficiency. To ensure message delivery each Chord peer maintains connections to its immediate predecessor and successor on the Chord ring.

In contrast to proactive maintenance, *reactive maintenance* is triggered immediately after the detection of a peer failure or peer departure. The missing entry in the routing table is replaced with a new one by sending a connect request to an appropriate peer.

Failures and departures of peers are detected in two ways: by probing or through usage. In probe-based failure detection each peer continuously runs a ping-response protocol with each of its neighbors. When ping timeouts occur repeatedly the neighbor is considered to be down and is removed from the routing table. In usage-based failure detection when a message is sent to a neighbor but not acknowledged within a timeout, the neighbor is considered to have failed.

The more neighbors a peer must maintain the higher the bandwidth overhead incurred by the maintenance protocol. In modern structured overlays maintenance bandwidth typically scales as $O(\log(N))$ in terms of the network size.

Key Applications

The key applications of peer-to-peer overlay networks include:

1. File-sharing systems, e.g., BitTorrent [3]
2. VoIP (Voice over IP) and VoD (Video on Demand) systems, e.g., Skype [12]
3. Information retrieval, e.g., YaCy-Peer [14]

Cross-references

- ▶ Load Balancing in Peer-to-Peer Overlay Networks
- ▶ Peer-to-Peer Content Distribution
- ▶ Peer-to-Peer Storage
- ▶ Peer-to-Peer System
- ▶ Peer-to-Peer Web Search

Recommended Reading

1. Aberer K. P-Grid: A self-organizing access structure for P2P information systems. In Proc. Int. Conf. on Cooperative Inf. Syst., 2001.
2. Androutsellis-Theotokis S. and Spinellis D. A survey of peer-to-peer content distribution technologies. ACM Comput. Surv., 36 (4):335–371, December 2004.
3. Bittorrent. http://www.bittorrent.com/.
4. Clarke I., Sandberg O., Wiley B., and Hong T.W. Freenet: A distributed anonymous information storage and retrieval system. In Designing Privacy Enhancing Technologies: Proc. Int. Workshop on Design Issues in Anonymity and Unobservability, 2001.
5. Gnutella Homepage. http://www.gnutella.wego.com/.
6. Kazaa Homepage. http://www.kazaa.com/.
7. Manku G.S., Bawa M., and Raghavan P. Symphony: Distributed hashing in a small world. In Proc. 4th USENIX Symp. on Internet Tech. and Syst., 2003.
8. Maymounkov P. and Mazières D. Kademlia: A peer-to-peer information system based on the XOR metric. In Proc. 1st Int. Workshop Peer-to-Peer Systems, 2002, pp. 53–65.
9. Napster. http://www.napster.com/.
10. Ripeanu M., Foster I., and Iamnitchi A. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. IEEE Internet Comput. J., 6(1), August 2002.
11. Risson J. and Moors T. Survey of research towards robust peer-to-peer networks: Search methods. Comput. Netw., 50(17):3485–3521, 2006.
12. Skype Homepage. http://www.skype.com/.
13. Stoica I., Morris R., Karger D.R., Kaashoek M.F., and Balakrishnan H. Chord: A scalable peer-to-peer lookup service for internet applications. In Proc. ACM Int. Conf. of the on Data Communication, 2001, pp. 149–160.
14. YaCyPeer. http://www.yacyweb.de/.

Peer-To-Peer Content Distribution

Pascal Felber[1], Ernst Biersack[2]
[1]University of Neuchatel, Neuchatel, Switzerland
[2]Eurecom, Sophia Antipolis, France

Synonyms

Cooperative content distribution; Peer-to-peer file sharing

Definition

Peer-to-peer content distribution is an approach for cost-effective distribution of bandwidth-intensive content to large numbers of clients based on *swarming*. Content is split into blocks that are sent to different clients. Thereafter, the clients can directly exchange blocks with one another, in a peer-to-peer manner, without the help of the original server.

Historical Background

Peer-to-peer systems are application-layer networks that directly interconnect users. They have enjoyed a phenomenal success in the last ten years together with the democratization of broadband access. The first disruptive peer-to-peer applications were designed for file sharing (e.g., Napster, Gnutella, KaZaA): they provided users with the means to (i) *search* for content, and (ii) *distribute* content. In the research community, most of the initial focus was on structured peer-to-peer overlays, also known as distributed hash tables (e.g., CAN, Chord). These systems have proved very efficient for *looking up* specific content.

Once users have started exchanging larger files, notably full-length movies, the classical client-server techniques used to transfer files were no longer sufficient and the focus has shifted on developing more scalable mechanisms for content distribution. The first and most emblematic example of peer-to-peer content distribution application is BitTorrent [5]. Its development started in 2001 and it reached widespread popularity in 2003.

BitTorrent is a peer-to-peer application that capitalizes the *bandwidth* of peer nodes to quickly replicate a single large file to a set of clients. The challenge is thus to maximize the speed of replication. The clients involved in a torrent cooperate to replicate the file among each other using *swarming* techniques: the file is broken into fixed size blocks and the clients in a peer set exchange blocks in parallel with one another. BitTorrent only deals with distribution and does not provide any mechanisms for locating content. Several alternatives to BitTorrent have been proposed, but none of them has reached the same level of popularity nor demonstrated sufficient benefits to supersede it.

More recently, some peer-to-peer content distribution systems have been developed specifically for streaming media, such as live television (e.g., PPLive). For such applications, blocks have to be received in order and in time, and clients are only interested in the part of the stream that is being broadcast while they are online.

Foundations

The principle underlying peer-to-peer content distribution is to capitalize the upstream bandwidth of the clients. A user that downloads some content will, at the same time, send part of the content to other peers.

Peer-to-peer content distribution networks are inherently *self-scalable*, in that the bandwidth capacity of the system increases as more peers arrive: each new peer requests service from, but also provides service to, the other peers. The peer-to-peer system can thus spontaneously adapt to the demand by taking advantage of the resources provided by every peer.

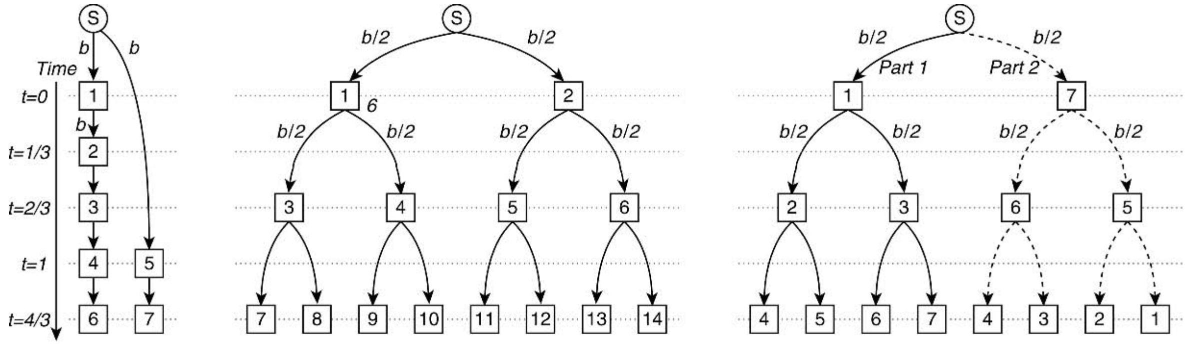
The behavior of each peer in a peer-to-peer content distribution is determined by two factors: (i) the *peer selection strategy* that determines the set of peers a given peer will exchange blocks with and (ii) the *block selection strategy* that determines which blocks will be exchanged. The choice of the peer and the block selection strategy has an important impact on the architecture of the content distribution system. One can distinguish between *structured* and *unstructured* architectures.

Structured architectures organize the peers in a directed acyclic graph such as linear chain or a tree (see Fig. 1). The peer selection is done once for the whole duration of the content distribution. The block selection becomes trivial as each peer simply forwards all the blocks it receives to its children. While structured architectures are easy to understand and model, they are best suitable for content distribution over networks where the bandwidth is homogeneous and nodes stay connected during the whole duration of the content distribution. If these conditions are not met, which is typically the case for the Internet, complex mechanisms are required to make structured approaches work.

Self-organization is the ability of a peer-to-peer network to dynamically determine how to best manage the block exchange as peers join, leave, or fail. Unstructured architectures are self-organizing: they use an underlying mesh structure and build directed graphs through which data is forwarded along several possible paths from the source to each peer. Meshes adapt well to bandwidth fluctuations/heterogeneity and nodes leaving and joining during the transfer at the price of more complex peer and block selection strategies. Since there exist multiple paths for receiving data, each peer must coordinate with its neighbors to avoid receiving the same block multiple times. One such system that uses an unstructured mesh-based approach is BitTorrent, a very popular peer-to-peer system for file distribution.

File Distribution

The BitTorrent protocol makes sensible choices for peer and block selection that are based on a few simple principles. *Peer selection:* first, every peer maintains a limited number of active connections to the other peers that offer the best upload and download rates, thus optimizing bandwidth utilization; second, a peer preferentially sends data to another peer that reciprocally sent data to it, which enforces fairness; third, every peer periodically sends some data to newcomers, so that peers can have an active role in the torrent independent of their arrival time. *Block selection:* a peer requests the block from its neighbors for which the least number of copies exist (rarest first), as rare blocks have a high trading value and can potentially increase the lifetime of the torrent. Maintaining a good



Peer-To-Peer Content Distribution. Figure 1. Evolution of three simple distribution models with time: linear chains (*left*), trees (*center*), and parallel trees (*right*). The nodes in each graph are labeled with the peer number and the edges with the bandwidth allocated for the block transmission. In this example, $C = 3$ and $k = 2$.

diversity of the blocks available in the systems assures that each peer can fully use its upload capacity since it has blocks that its neighbors are interested in.

Despite the simplicity of these design principles and the lack of theoretical foundations to back them when they were introduced, studies [8] have shown they perform exceptionally well in practice. The uplink utilization at the peers is remarkably high, sharing incentives are very effective and resilient to freeriders, and one can observe that peers spontaneously cluster according to their capacity [10].

The choice of the peer and block selection strategy is very important for the good performance of mesh-based systems, because peers can fully use their upload capacity only if the *diversity* of blocks is high, i.e., if they own blocks their neighbors are interested in. Another way to assure that a peer has useful information to transmit to its neighbors is to use *network coding*. In this case, every block that is transmitted is a linear combination of all or a subset of the blocks available at the peer. Since with high probability every such block is unique and contains useful information for the receiving peer, the block selection strategy becomes trivial. Avalanche [6] is a peer-to-peer system for file distribution that uses network coding.

Video Streaming

File distribution systems such as BitTorrent are often used to download files that contain audio or video content. Since the blocks can be downloaded in any order, the consumption of the content cannot start before the download is complete.

In recent years, some peer-to-peer content distribution systems have been developed to support live streaming, where the playback of the content occurs simultaneously with its production. Supporting video streaming is more challenging than file distribution: the users expect that once viewing has started it will be continuous, which means that data must be received in sequence and the rate of data reception must be equal the rate at which data are injected by the source. The architectural variants available for live streaming are the same as for file distribution, namely structured and unstructured approaches.

SplitStream [4] uses a structured architecture and constructs parallel trees, with every peer belonging to all trees. Content is split in multiple layers, each sent along a different tree. As was discussed for the case of file distribution, structured architectures lack flexibility. For this reason, the systems that have been used most widely for video streaming are mesh-based. One can notably mention PPlive [7] that has been used to stream video to tens of thousands of peers.

A robust video streaming system must be able to cope with the heterogeneity of the peers, which can have widely varying download bandwidth. To assure real-time data reception for all clients despite heterogeneity, one can use layered coding: the video is encoded into a base layer and several enhancement layers. While the video cannot be viewed if the base layer is not completely received, the enhancement layer only improve the viewing experience of the video and can be omitted if the download bandwidth is not sufficient.

Many peers are connected via ADSL links that provide much higher download rates than upload rates. A peer-to-peer streaming system is only stable if the aggregate upload rate of the participating peers is at least as high as the aggregate playout rate [9]. In many cases this can only be achieved if there are some peers ("super-peers") that have an upload capacity much higher than the video playout rate [7].

Besides participating in a live streaming event, users may want to watch a video at any point of time, which is referred to as video on demand. While there exist designs for peer-to-peer systems that support video on demand [1], none of these systems has been deployed on a large scale.

Performance Analysis of Distribution Architectures

To evaluate the potential of peer-to-peer file distribution, one can consider very simple distribution models [3] where a server S distributes a file to N peers. The server splits a file into C blocks and serves the file sequentially and infinitely at rate b. The time needed to download the complete file from the server to a *single* peer at rate b is referred to as *one round*.

Consider first a linear chain architecture where the peers are organized in a chain with the server uploading the blocks to the first peer, which in turn uploads the blocks to the second peer and so on (Fig. 1, left). Peers disconnect once they have uploaded the whole file once. The number of peers served in a given number of rounds grows linearly with the number of blocks C, because one can faster engage all peers in the distribution process, and quadratically with the number of rounds t, because the source forks additional chains. Interestingly, when $N/C \ll 1$, the time necessary to serve N peers converges asymptotically to 1.

Consider now a tree architecture where the peers are organized in a tree with an outdegree k (Fig. 1, center). The server serves k peers in parallel, each at rate b/k, and all the peers that are not leaves in the tree in turn upload the blocks to k other peers at rate b/k. This means that it takes k rounds for a peer to download the file and interior nodes upload an amount of data equivalent to k times the size of the file, while leaf nodes do not upload the file at all. The performance of the tree architecture depends on the node degree, and it turns out that for $N/C \leq 1$ the best value is $k = 1$, i.e., a linear chain. For $N/C > 1$ a binary tree is more efficient. Trees with higher degrees are penalized by the higher number of non-contributing leaf nodes.

Finally, consider a forest of k parallel trees, each of which contains all the peers (Fig. 1, right). The server partitions the file into k parts and constructs k spanning trees to distribute each part along a separate tree to all peers. The trees can be built such that each peer is an interior node in at most one tree and a leaf in the remaining $k - 1$ trees. The parallel tree architecture is as efficient as the linear chain when $N/C \ll 1$ and significantly outperforms the other two architectures in other scenarios. The optimal performance is obtained for trees with an outdegree $k = e \approx 3$.

Table 1 summarizes the scaling behavior of the different approaches. One can see that for both the tree and parallel trees architectures, the number of clients served increases exponentially in time and in the number of blocks C.

These performance results are remarkable: with the same amount of effort at the server, distributing the content to N peers using a peer-to-peer architecture can be done in a little more than one round, i.e., it takes just slightly longer than to distribute the file from the server to a single peer. While structured architectures are neither robust nor practical, these performance models still give valuable insights on the asymptotically performance of peer-to-peer content distribution. See [11,2] for studies that take into account other factors such as bandwidth heterogeneity.

Developing performance models for unstructured architectures is much more difficult than for structured ones. Yet, it turns out that the following unstructured architecture, called *Interleave*, is analytically tractable [12]. The performance model for Interleave assumes that all blocks are numbered in increasing order, all peers have the same bandwidth, and the transmission of a block takes one slot. Peer selection is random, while block selection is as follows: in every odd time slot the source and the other peers push the highest numbered block they own to a random peer; in every even time slot each peer requests from a random peer the lowest numbered block that the peer does not yet have. Note that Interleave does not require any peer to exchange information with other peers about the block it already has. As a consequence the protocol will experience some inefficiency as a peer p may ask another peer q for a block that q does not have, or may send to q a block that q already has. It can be shown analytically that with high probability the entire file that consists of C blocks can be distributed to all N peers in a time of $3.2 + \frac{\log N}{C}$. Using simulation the result obtained was $2 + \frac{\log N}{C}$, which is similar to the time required by a structured binary tree (see Table 1). Such good performance is quite astonishing as is indicates that there exist robust mesh-based file distribution schemes that can achieve performance as good as structured architectures.

Key Applications

Peer-to-peer content distribution is a technique to transfer large contents simultaneously to many users.

Peer-To-Peer Content Distribution. Table 1. Performance comparison of three structured distribution models

Architecture	Clients served	Service time	Copies served	Download rate	Upload rate
Linear chain	$C \cdot t^2$	$\frac{1}{2} + \sqrt{\frac{1}{4} + \frac{2 \cdot N}{C}}$	1	b	b (except leaves)
Tree	$k^{(t-k)\frac{C}{k}+1}$	$k + (\lfloor \log_k N \rfloor - 1) \cdot \frac{k}{C}$	k	$1/b$	b (except leaves)
Parallel trees	$k^{(t-1)\frac{C}{k}}$	$1 + \lfloor \log_k N \rfloor \cdot \frac{k}{C}$	1	b	b

Key applications include file sharing systems, live TV, video on demand, or distribution of critical updates.

It also represents a cost-effective alternative to scaling up server architectures. It has notably been used as a way to protect servers from unexpected surges in request traffic, called flash crowds, by replicating popular content on multiple peers.

Future Directions

Peer-to-peer content distribution networks are overlays that typically ignore the underlay, i.e., the underlying IP connectivity. Therefore, such networks generate much unnecessary traffic between Internet service providers.

Another important problem that plagues content distribution networks is the fact that the validity of content cannot be verified until after it is downloaded. The media industry contributes to content pollution in existing networks to deter exchange of copyrighted files.

One can expect video on demand to be an important application domain for peer-to-peer content distribution technology. A major challenge is that users must be able to start watching a movie at any time, as well as suspend it, rewind, skip chapters, etc. (VCR functionality). In these scenarios, there is no such synchronization between the peers as with live streaming.

Cross-references

- ▶ Distributed Hash Table
- ▶ Peer-to-Peer System

Recommended Reading

1. Annapureddy S., Guha S., Gkantsidis C., Gunawardena D., and Rodriguez P. Is high quality VoD feasible using P2P swarming. In Proc. 16th Int. World Wide Web Conference, 2007.
2. Biersack E.W., Carra D., Cigno R.L., Rodriguez P., and Felber P. Overlay architectures for file distribution: Fundamental performance analysis for homogeneous and heterogeneous cases. Computer Networks, 51(3):901–917, 2007.
3. Biersack E., Rodriguez P., and Felber P. Performance analysis of peer-to-peer networks for file distribution. In Proc. 5th International Workshop on Quality of Future Internet Services, 2004, pp. 1–10.
4. Castro M., Druschel P., Kermarrec A.M., Nandi A., Rowstron A., and Singh A. SplitStream: High-bandwidth multicast in a cooperative environment. In Proc. 19th ACM Symp. on Operating System Principles, 2003.
5. Cohen B. Incentives to Build Robustness in BitTorrent. Tech. rep., http://www.bittorrent.org/, 2003.
6. Gkantsidis C., Miller J., and Rodriguez P. Anatomy of a P2P content distribution system with network coding. In Proc. 5th Int. Workshop Peer-to-Peer Systems, 2006.
7. Hei X., Liang C., Liang J., Liu Y., and Ross K. A measurement study of a large-scale P2P IPTV system. IEEE Trans. on Multimedia, 9(8):1672–1687, 2007.
8. Izal M., Urvoy-Keller G., Biersack E., Felber P., Al Hamra A., and Garces-Erice L. Dissecting BitTorrent: Five months in a torrent's lifetime. In Proc. 5th Passive and Active Measurement Workshop, 2004.
9. Kumar R., Liu Y., and Ross K.W. Stochastic fluid theory for P2P streaming systems. In Proc. 26th Annual Joint Conf. of the IEEE Computer and Communications Societies, 2007.
10. Legout A., Liogkas N., Kohler E., and Zhang L. Clustering and sharing incentives in bittorrent systems. In Proc. 2007 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Comp. Syst., 2007.
11. Mundinger J., Weber R., and Weiss G. Optimal scheduling of peer-to-peer file Dissemination. Journal of Scheduling, 2007.
12. Sanghavi S., Hajek B. and Massoulie L. Gossiping with multiple messages. IEEE Transactions on Information Theory, 53(12), 2007.

Peer-to-Peer Data Integration

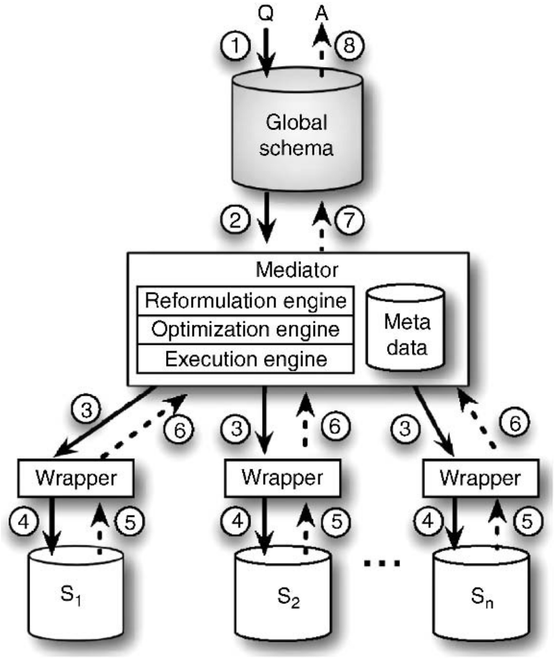
Anastasios Kementsietsidis
IBM T.J. Watson Research Center, Hawthorne,
NY, USA

Definition

Peer-to-Peer data integration lies in the intersection of two popular research topics, namely, Peer-to-Peer systems and Data Integration, and is one of the key topics in the area of *Peer-to-Peer Data Management.* A Peer-to-Peer data integration setting involves a set $\mathcal{P}$ of autonomous, heterogeneous, independently evolving (peer) sources whose pairwise schema or data-level mappings, collectively denoted by $\mathcal{M}$, induce a peer-to-peer network. In this setting, each (peer) source in the network can be queried and act as an *access point* to the data residing in the other network sources. Research in this area focuses on studying the specification and expressiveness of the peer mappings; the corresponding query languages used; algorithms for rewriting queries between peer source schemas; and, to some extent, topics that concern the propagation of updates between peer sources. Key characteristics of the peer-to-peer data integration setting that differentiate it from *traditional* data integration settings and typical Peer-to-Peer systems include (i) the fact that each peer can be full-fledged database; (ii) the lack of centralized control and global schemas; (iii) the need for more diverse set of mapping specifications; and (iv) the need for integration of data across diverse domains.

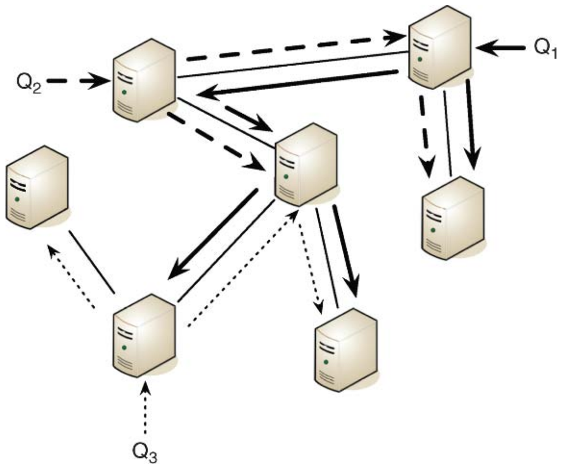
Historical Background

Data integration [11] has been characterized as one of the longest-standing research problems faced by the data management community. A typical data integration setting involves a set $\mathcal{S}$ of sources, a global schema $\mathcal{G}$, and a set $\mathcal{M}$ of mappings between the sources in $\mathcal{S}$ and the global schema $\mathcal{G}$. Figure 1 illustrates such a setting along with the logical steps during query evaluation. Central to the evaluation of queries are the mappings between the global and local schemas (indicated as *Metadata* in the figure). The mappings are used during the rewriting of a user query over the global schema $\mathcal{G}$ to a set of queries over the sources. Two basic approaches have been used to specify the mappings. In a nutshell, in the *Global-as-View* (GAV) approach, the global schema $\mathcal{G}$ is expressed as a view of the set of local schemas $\mathcal{S}$. Main advantage of the GAV approach is the simplicity of the rewriting algorithm. However, a drawback of the approach is that the addition of source in $\mathcal{S}$ results in a revision of the mapping and, in the worst case, a complete re-design of the global schema. In the second approach, called *Local-as-View* (LAV), each source $S_i \in \mathcal{S}$ is expressed as a view over the global schema $\mathcal{G}$. Query rewriting is substantially more involved here, but the approach handles source additions more gracefully.



Peer-to-Peer Data Integration. Figure 1. Typical data integration setting.

Peer-to-peer computing involves an open-ended network of computational peers, where each peer exchanges data and services with a set of other peers, called its *acquaintances*. The peer-to-peer paradigm, initially popularized by file-sharing systems such as Napster [12] and Gnutella [5], offers an alternative to traditional architectures found in distributed systems and the web. Distributed systems are rich in services, but require considerable overhead to launch and have a relatively static, controlled architecture. In contrast, the web offers a dynamic, anyone-to-anyone architecture with minimum startup costs but limited services. Combining the advantages of both architectures, peer-to-peer offers an evolving architecture where peers come and go, choose with whom they interact, and enjoy some traditional distributed services with less startup cost. Figure 2 shows an example of such a system, often referred to in the literature as an *unstructured* peer-to-peer system, to differentiate it from the complementary class of *structured* peer-to-peer systems whose architecture is based on *Distributed Hash Tables* (DHT's). As shown in the figure, mappings in unstructured peer-to-peer system (represented by solid lines) exist between any pair of peers while queries (represented by various line arrows) can be initiated



Peer-to-Peer Data Integration. Figure 2. A Peer-to-Peer system.

at any peer in the system. Furthermore, different queries might involve a different set of peers.

Probably the first works to consider the interaction between database and peer-to-peer systems were the ones by Gribble et al. [6] and Bernstein et al. [3]. The former work focuses on the problem of *data placement*, that is, to find an optimal placement of a set of objects on a peer-to-peer system, under a prespecified query workload, so as to minimize the cost of evaluating the queries. In more detail, the authors consider graph whose nodes correspond to the peers in a peer-to-peer network. Each node is associated with a storage capacity and a query workload. Furthermore, each pair of nodes connected by an edge has an associated data trasfer cost, over this edge. Queries over this network are object lookups and each query has an associated frequency and cost (with the cost being zero, if the query can be served locally in a peer, and a function of the object size and edge transfer cost, if served remotely). In this setting, the main result of this work is that the problem of data placement with optimal cost is NP-complete. The work by Gribble et al. [6] initiated the Piazza System [17] (described in the next section) although the follow-up work in Piazza addresses a setting much closer to data integration. As presented here, the setting in Gribble et al. [6] is more closely related to the one found on the work on DHT's like CAN [15] or Chord [16].

Bernstein et al. [3] is the first work to actually discuss data integration in a peer-to-peer context. The focus in this work is the introduction of the Local Relational Mode (LRM), a model designed specifically for the peer-to-peer setting. Key notions in the model are that of *coordination formulas* and *domain relations*. Much like a mapping in a typical data integration setting, a coordination formula establishes a relationship between the data items stored in acquainted peers. This relationship can be used to express a constraint, like for example, that a particular data item must be stored in either one of three different acquainted peer databases, or it can be used during query answering by expressing, for example, a GAV (LAV) type mapping where the data items returned by query Q_1 in a peer P_1 are contained in those returned by query Q_2 in peer P_2. Since different peers might use different vocabularies to express the same real-world notion, it is the responsibility of domain relations to express the mapping between these vocabularies. For example, a domain relation might be used to map prices expressed in US dollars, in a peer, to those in euro in another peer. The work by Bernstein et al. [3] initiated the Hyperion project [2] the details of which are described in the next section.

Foundations

Important aspects in every peer-to-peer data integration system include (i) the peer joining process and especially the type of mappings used by the system to resolve the heterogeneity of the acquainted peer; (ii) the supported query language and the processing of queries; and (iii) how the system deals with the lack of centralization and the inherent dynamic and unreliable nature of the peer-to-peer setting. Not every system addresses all these aspects and different systems put more emphasis on a different system aspect. The following paragraphs provide an overview of some of the main systems in this area.

In Piazza [17], each peer in the system *exports* a schema which can be one of two kinds: (i) a virtual schema, used for querying and mapping the schemas of its acquainted peers, or (ii) a schema which internally is mapped to actual stored data in the peer. In either case, a peer joining the system establishes GAV or LAV mappings between its export schema and the export schemas of peers that are already part of the system. The creation of such mappings is a complicated process which cannot be fully automated. *Schema matching* techniques are used, both in Piazza and elsewhere, to facilitate their creation. Both the created mappings and the supported query language are expressed in a fragment of XQuery. User queries in this fragment are expressed over the schema of a single peer. The query is answered locally by the peer, if the peer actually stores data, and it is also reformulated to a set of queries over the acquaintances of the current peer. Briefly, the reformulation algorithm combines view unfolding, if a GAV mapping exists between acquainted peers, and an answering queries using views algorithm [7], in the case of LAV mappings. The reformulation algorithm terminates once all the queries that result in from the reformulation refer only to relations that correspond to stored data in peers (and no virtual schemas).

Influenced by the work in the LRM model, in Hyperion [2] two types of mappings are used to support the sharing of information between relational database peers, namely, *mapping expressions* and *mapping tables*. Similar to the GAV/LAV mappings in Piazza (and elsewhere), mapping expressions are schema-level mappings used during query answering for the reformulation of queries between peers. A distinguishing feature of Hyperion, mapping tables [9] are *data-level* mappings that associate the values of acquainted peers (inspired by the domain relations in the LRM model). A mapping table T is a relation over the set of attributes $X \cup Y$, where X and Y are non-empty sets of attributes belonging to two acquainted peers. Each tuple $t \in T$ (whose values might include constants and/or variables) associates the set of values in $t[X]$ to those in $t[Y]$.

While schema matching can still be employed for the creation of mapping expressions, the creation of mapping tables requires the development of specialized techniques. Indeed, the work in [9] formalizes mapping tables as data-level constraints over the sharing of data between peers and illustrates how new mapping tables can be inferred automatically (from existing tables) while establishing an acquaintance between two peers. Due to the dynamic nature of peer-to-peer system, peers are expected to continuously evolve and change their schema (less frequently) and their data (very frequently). This change influences both the existing mapping expressions and the existing mapping tables. As a result, work in the Hyperion project is concerned with how to maintain the existing mappings with emphasis in the frequently updated mapping tables.

In terms of query answering, for the case of mapping expressions, Hyperion relies on techniques similar to the ones developed for Piazza. For the case of mapping tables, Hyperion uses a specialized algorithm to rewrite select-project-join (SPJ) queries between peers, relying only on the use of mapping tables for the rewriting [10]. No algorithm is provided in Hyperion to rewrite queries by combining mapping expressions and tables. However, mapping expressions are used in conjunction with mapping tables in Hyperion for *data coordination* [8]. Indeed, another distinguishing characteristic of Hyperion is that it supports the creation and distributed execution of Event-Condition-Action (ECA) rules over multiple acquainted peer.

The PeerDB [14] system is built on top of a generic peer-to-peer platform, called BestPeer [13]. The BestPeer platform supports two types of peers, namely, a large number of normal (data) peers and a smaller number of *location independent global names lookup (LIGLO)* peers. A LIGLO peer acts as a name server with which every peer in the system registers, when joining, in order to acquire a unique identifier.

Similar to Piazza and Hyperion, in PeerDB, data sharing between heterogeneous peers is achieved in two steps. In the first step, mappings are established between the schemas of the peers. In the second step, the mappings are used to rewrite a query over the schema of one peer to a query over the schema of another. PeerDB has a number of distinguishing characteristics over the previous approaches. First, it differs from the previous systems in that it uses an information retrieval-based technique as a basis for its schema mappings. In more detail, each peer relation and attribute name is associated with a set of descriptive keywords. A mapping between two relations (attributes) that reside on different peers is established if their corresponding descriptive keywords overlap *significantly*. Once a mapping is established, it is used to rewrite a query that refers to one relation into a query that refers to its mapped counterpart. The second distinguishing characteristic of PeerDB is that mappings are established dynamically, at query time. In more detail, PeerDB employs an agent-based technology both during the discovery of schema mappings and during the rewriting of queries. After a user initiates a query over a local peer schema, software agents are responsible for crawling the peer-to-peer network, looking for peers whose schemas can be mapped to the schema where the user query is posed. The agents carry all the necessary functionality to perform the rewriting once such schemas are discovered.

Similar to PeerDB, the GridVine [4] system is built on top its own generic peer-to-peer platform, called P-Grid [1]. However, while the architecture of BestPeer follows the peer/super-peer paradigm, the architecture of P-Grid is that of a structured overlay network. Capitalizing on the efficiency of P-Grid in terms of indexing, routing and load balancing, GridVine builds a semantic mediation layer on top of P-Grid. In this semantic layer, schemas and data are represented as RDF triples, while queries over these triples are expressed as *triple patterns*. A triple pattern query issued at any peer is routed to the peer that can answer the query by using the services of the overlay network (by hashing the constants appearing in the query). In terms of heterogeneity, GridVine employs OWL statements to relate semantically similar schema elements that belong to pairs of schemas. During query evaluation, these mappings are used for the rewriting of queries which are then executed using the overlay network, as described earlier.

Key Applications

Scientific databases, Health informatics databases, Business-to-Business (B2B).

Cross-references

- ▶ Data Integration
- ▶ Distributed Database Systems
- ▶ Parallel Database Systems
- ▶ Peer Data Exchange
- ▶ Peer-to-Peer Data Management
- ▶ Schema Matching

Recommended Reading

1. Aberer K., Cudré-Mauroux P., Datta A., Despotovic Z., Hauswirth M., Punceva M., and Schmidt R. P-Grid: a self-organizing structured P2P system. ACM SIGMOD Rec., 32(3):29–33, 2003.
2. Arenas M., Kantere V., Kementsietsidis A., Kiringa I., Miller R.J., and Mylopoulos J. The hyperion project: from data integration to data coordination. ACM SIGMOD Rec., 32(3):53–58, 2003.
3. Bernstein P., Giunchiglia F., Kementsietsidis A., Mylopoulos J., Serafini L., and Zaihrayeu I. Data management for peer-to-peer computing: a vision. In Proc. 5th Int. Workshop on the World Wide Web and Databases, 2002.
4. Cudre-Mauroux P., Agarwal S., and Aberer K. GridVine: an infrastructure for peer information management. IEEE Internet Comput., 11(5):36–44, 2007.
5. Gnutella Protocol Specification. World Wide Web URL: http://gnet-specs.gnufu.net/.
6. Gribble S., Halevy A., Ives Z., Rodrig M., and Suciu D. What can databases do for peer-to-peer? In Proc. 4th Int. Workshop on the World Wide Web and Databases, 2001.
7. Halevy A.Y. Answering queries using views: a survey. VLDB J., 10(4):270–294, 2001.
8. Kantere V., Kiringa I., Mylopoulos J., Kementsietsidis A., and Arenas M. Coordinating peer databases using ECA rules. In Proc. Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing, 2003, pp. 108–122.
9. Kementsietsidis A., Arenas M., and Miller R.J. Data mapping in peer-to-peer systems: semantics and algorithmic issues. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 325–336.
10. Kementsietsidis A. and Arenas M. Data sharing through query translation in autonomous sources. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 468–479.
11. Lenzerini M. Data integration: a theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
12. Napster. World Wide Web URL: http://www.napster.com/.
13. Ng W.S., Ooi B.C., and Tan K.-L. BestPeer: a self-configurable peer-to-peer system. In Proc. 18th Int. Conf. on Data Engineering, 2002, p. 272.
14. Ng W.S., Ooi B.C., Tan K.-L., and Zhou A. PeerDB: a P2P-based system for distributed data sharing. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 633–644.
15. Ratnasamy S., Francis P., Handley M., Karp R., and Shenker S. A scalable content addressable network. In Proc. ACM Int. Conf. on Data Communication, 2001, pp. 161–172.
16. Stoica I., Morris R., Karger D., Kaashoek F., and Balakrishnan H. Chord: a scalable peer-to-peer lookup service for internet applications. In Proc. ACM Int. Conf. on Data Communication, 2001, pp. 149–160.
17. Tatarinov I., Ives Z., Madhavan J., Halevy A., Suciu D., Dalvi N., Dong X.L., Kadiyska Y., Miklau G., and Mork P. The piazza peer data management project. ACM SIGMOD Rec., 32(3):47–52, 2003.

Peer-to-peer Database

▶ Distributed Architecture

Peer-to-peer File Sharing

▶ Peer-To-Peer Content Distribution

Peer-to-peer Network

▶ Peer-to-Peer System

Peer-to-peer Overlay

▶ Peer-to-Peer System

Peer-to-Peer Publish-Subscribe Systems

Peter Triantafillou, Ioannis Aekaterinidis
University of Patras, Rio Patras, Greece

Definition

Publish/Subscribe (a.k.a. pub/sub) software systems constitute a facility for asynchronous filtering of information. Users, consumers of information, present the system with continuous queries, coined *subscriptions*. Sources of data generation (producers) present the system with data-carrying *publication events*. The pub/sub system infrastructure is responsible for (asynchronously) matching the publication events to all relevant subscriptions. Hence, in essence, this infrastructure filters all available information for every user and presents to each user only the information units (s)he has defined as relevant. As such, a pub/sub infrastructure can play a vital role in large-scale data systems, with huge volumes of data, shielding users from the burden of always actively searching for and retrieving relevant information units.

Peer-to-Peer (P2P) systems are software systems, which in fact constitute *overlay* networks, which are built over physical networks, such as the Internet. Their key discriminative feature is the complete decentralized algorithmic and system design, which, in turn, leads to guarantees with respect to system *scalability* in terms of the network size and stored data. In addition, P2P systems are characterized by *self-organization*, being able to withstand high dynamics with respect to network nodes joining and leaving the system while continuing to offer efficient operation.

Pub/sub P2P systems are an attempt to combine these two important technologies. The central aim is to offer facilities for asynchronous matching of continuous user queries to publication events, at very large scales. This entails dealing with large numbers of producers and consumers of information, which are geographically distributed and, also, supporting large publication-event and subscription arrival rates. Coupling pub/sub technology with the P2P paradigm achieves this aim, while introducing the additional benefits of scalability and self-organization into the pub/sub realm.

Historical Background

Publish/subscribe systems have evolved significantly over time, differing on a number of fundamental characteristics. They are classified into two major categories, according to the way subscribers express their interests; the *topic-based* [6,10,16] and the *content-based* [4,5,7,8] systems. Historically, the first systems were topic-based, in which users subscribe to specific topics. All incoming publication events associated with a particular topic are sent to all user subscribers of the topic. This paradigm mimics the way news groups operate. *Content-based* pub/sub systems emerged subsequently, offering users the much-needed ability to express their interests on specific publication events, carrying specific content. Principally, users issue subscriptions which specify predicates over the *values* of a number of well defined attributes. The matching of publication events to subscriptions is performed based on the content (i.e., the values of attributes) being carried by the publication events.

At the turn of the century, related research efforts were maturing and a number of content-based pub/sub systems were already available. Influential examples, with respect to content-based pub/sub systems research, include SIENA [5], Gryphon [4], Le Subscribe [8] and JEDI, [7]. Already, some of this work targeted the challenging issues arising in a distributed system, where publishers and subscribers are geographically distributed [4,5]. Around the same time and in parallel, research related to P2P networks and systems was also maturing. Worthy of special mention are *structured P2P overlay networks*, like Chord [14], Pastry [13], CAN [12], and P-Grid [1]. Several of these networks influenced work on pub/sub systems, especially endeavors aiming to leverage existing P2P networks for providing pub/sub functionality with scalability, efficiency, and self-organization. The Scribe system [6] was a first attempt at providing topic-based pub/sub functionality over the Pastry P2P network. This effort was followed by endeavors to build content-based publish/subscribe systems over P2P networks, [2,3,9,11,15,17].

Foundations

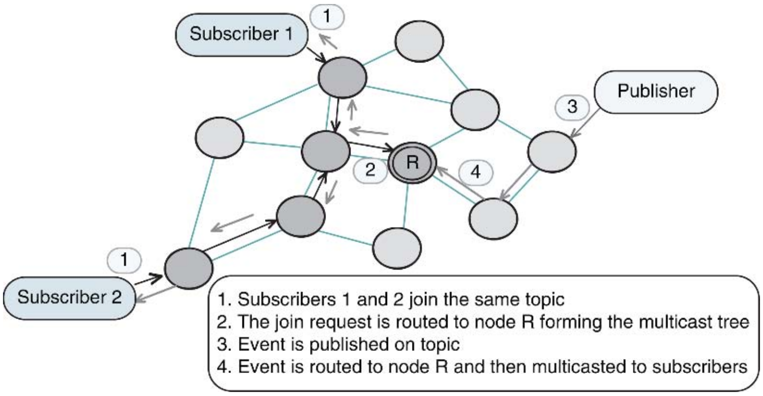
Pub/sub data management represents a significant point of departure compared to traditional data management. In the latter, data items are stored and the system is responsible for appropriate indexing and processing queries and updates issued by users against these data items. In the pub/sub model, it is the users' continuous queries (subscriptions) that are stored and indexed appropriately and the system is responsible for processing data-carrying publication events, matching them to all relevant stored subscriptions.

The fundamental functionality offered by a publish/subscribe system rests on the pillars of subscription processing and publication-event processing and matching. In a P2P environment, all this functionality is based on distributed algorithms that appropriately leverage the P2P network capabilities in order to ensure scalability and efficiency of operation, free of concern for network topology dynamics. The core functionality exported by a P2P network is the so-called *lookup()* function, which receives as input a *key* and returns the network address of the peer node where the data item associated with the key is located. Structured P2P networks, such as those built using *Distributed Hash Tables (DHTs)* (such as [12–14]) can ensure that *lookup()* executes in $O(logN)$ messages, in a network of N nodes, a feature that in essence guarantees scalability and efficiency.

In topic-based publish/subscribe systems, events and subscriptions are associated with specific topics (a.k.a. groups/subjects). A straightforward approach, on which Scribe [6] for example is based for supporting topic-based pub/sub functionality in a P2P environment is to further associate each topic with a *multicast tree*. Topic (and thus multicast tree) creation, involves identifying the node responsible for the tree root, which is determined simply by issuing a *lookup (hash(topic_name))*, using the DHT's hash function. The DHT hash function introduces a randomizing effect when selecting root nodes for different multicast trees. Subscription processing then involves storing the subscription locally at the origin peer node and having that node join the multicast tree. This is basically accomplished by also issuing a *lookup(hash(topic_name))* and creating a path in the multicast tree consisting of all the DHT nodes visited during the execution of the *lookup()* call. Finally, publication-event processing is performed by locating the tree root node (again, using the *lookup ()* function) and sending the publication to it, which it subsequently distributes to the multicast tree. Figure 1 illustrates this process.

The content-based pub/sub model has dominated the area, since it allows for greater user-query expressiveness, resulting in more efficient information filtering. However, this model incurs a much higher complexity. The publication event and subscription schema adopted in this model, defines a set of A attributes (a_i, $1 \leq i \leq A$). Each attribute a_i consists of a name, a type (usually string or numerical), and a value $v(a_i)$. A publication event is defined to be a set of $k < attribute, value >$ pairs ($k \leq A$), while a subscription is defined through an appropriate set of predicates on attributes' values over a subset of the A attributes of the schema. The complexity emerges from the need to support subscriptions with complex predicates. For numerical-typed attributes, the allowable predicates may involve equality, inequality, $\leq, \geq$, and ranges of values. For string-typed attributes, subscribers may define prefix, suffix, sub-string and equality predicates. Fundamentally, solutions in a P2P environment can be classified according to whether they require knowledge of the internal DHT routing state, which is maintained by each DHT node, and its association with additional state that is needed for subscription and publication processing [3,11,15]. A key characteristic in several of these approaches [11,15] is that each node n_1 associates with each other node in its routing table, say n_2, additional state that consists of all subscriptions that n_1 has received from n_2. When a publication event arrives at n_1, it is forwarded to every node n_2 in its routing table only if the publication event matches one of the subscriptions sent to n_1 by n_2.

Approaches that do not belong in this category avoid the maintenance costs associated with the extra,



1. Subscribers 1 and 2 join the same topic
2. The join request is routed to node R forming the multicast tree
3. Event is published on topic
4. Event is routed to node R and then multicasted to subscribers

Peer-to-Peer Publish-Subscribe Systems. Figure 1. Multicast tree construction for event dissemination in topic-based publish/subscribe.

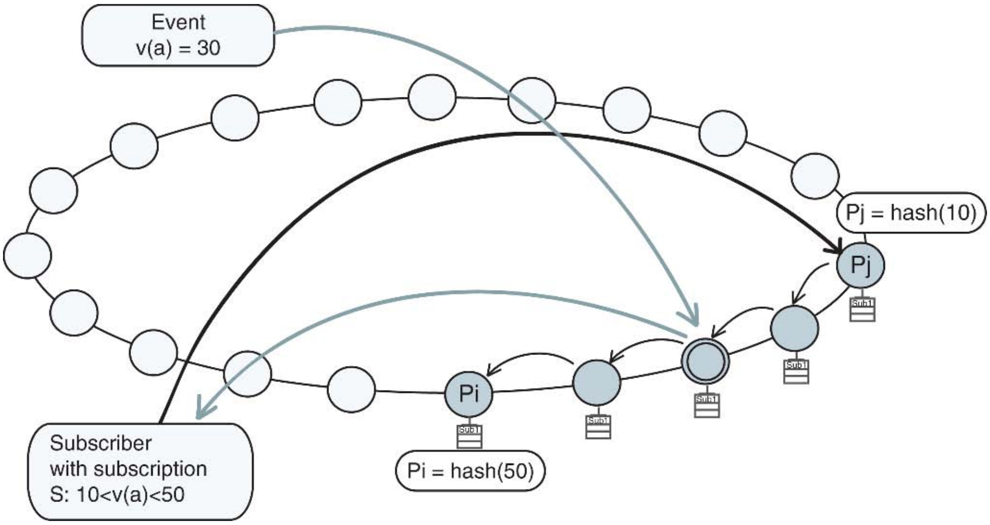
per-node state and enjoy wider applicability as they can be easily integrated with a number of DHTs. Hereafter, the focus is on these approaches, where the content (i.e., the values defined by the predicates) will determine at which peer nodes the subscriptions will be stored. Similarly, the values carried by a publication event will determine which route must be followed within the network so to reach every possible peer node storing a relevant subscription.

Equality predicates can be handled straightforwardly – the basic idea being the following. A subscription s associating with attribute a the value $v(a)$, can be stored simply at the peer node with identifier p, where $p = hash(v(a))$. A publication event e carrying $a = v(a)$, is processed by visiting the peer node $p = hash(v(a))$ and retrieving all locally-stored subscriptions, such as s.

However, for more complex predicates, this is inadequate. At an abstract, high-level view, proposed solutions, albeit very different, share the following characteristics. The node ID namespace is typically associated in solution-specific manners with the value domain of an attribute. A subscription with a complex predicate on an attribute is stored at several nodes, whose IDs depend on the values defined by the subscription's predicate. In other words, a subscription is mapped to a subspace of the node ID namespace. Publications, associating an attribute with a value correspond to a point in the namespace. The node associated with this point, by construction, will be storing all subscriptions whose predicates on this attribute include this event's value. In this way, the *subscription to publication event rendezvous* occurs and in this rendezvous node the event can be matched to all relevant subscriptions.

For concreteness, the basics of the approach adopted in [17] – one of the first to leverage an existing P2P network on top of which to build scalable content-based pub/sub systems – for processing subscriptions with range predicates, is outlined. The approach is simple, requires no additional state maintenance, and no knowledge of the internals of the underlying DHT state. This approach utilizes the Chord DHT [14] in which nodes are arranged in a circular list according to their IDs. However, in [17] specific order-preserving hashing is employed to store subscriptions in the Chord network. That is, when processing subscriptions with values $v(a)_i$ and $v(a)_j$, they are stored at nodes p_i and p_j whose IDs are given by $hash(v(a)_i)$ and $hash(v(a)_j)$, respectively. If $v(a)_j < v(a)_i$, then $p_j < p_i$. An incoming subscription s, identifying a range of values $[v(a)_j, v(a)_i]$ is stored on all nodes in the arc of the ring starting at node $hash(v(a)_j)$ and ending at node $hash(v(a)_i)$. Given this, a publication carrying value $v(a)_k$, with $v(a)_j < v(a)_k < v(a)_i$ will be directed at node $hash(v(a)_k)$ and this node by construction falls on the arc of the ring where the subscription s has been stored. Thus, the matching can be locally performed at this node. Figure 2 illustrates this process.

Finally, the aforementioned discussion has focused on single-attribute (one-dimensional) subscriptions and publications. In general, pub/sub systems involve subscriptions and publications specifying multi-dimensional



Peer-to-Peer Publish-Subscribe Systems. Figure 2. Processing range queries in a content-based pub/sub system.

content. Processing multi-dimensional publication events and subscriptions is a source of additional complexity. In this setting, a publication event *e* will match a subscription *s* if and only if all attributes named by *s* are also named by *e* and the predicates defined by *s* on its attributes are satisfied by the values for these attributes carried by the publication event. A straightforward approach dealing with multi-dimensional events and subscriptions is the following. First, perform the processing as discussed above for each named attribute in the publication event and subscription. Second, for each attribute collect a candidate result set, consisting of the subscriptions being matched only for that attribute of the publication. Lastly, merge and filter the per-attribute candidate result sets into a single result set, by removing those subscriptions in the candidate result sets which have at least one attribute predicate not satisfied by the publication event. Figure 3 illustrates this process.

As these candidate result sets are distributed and can contain very large numbers of subscriptions, the tasks of merging them and filtering subscriptions from them can introduce large overheads. Therefore, multi-dimensional event processing is open to a number of crucial performance optimizations which reveal key trade-offs with respect to network bandwidth overheads, number of required messages, and event-matching latencies [2].

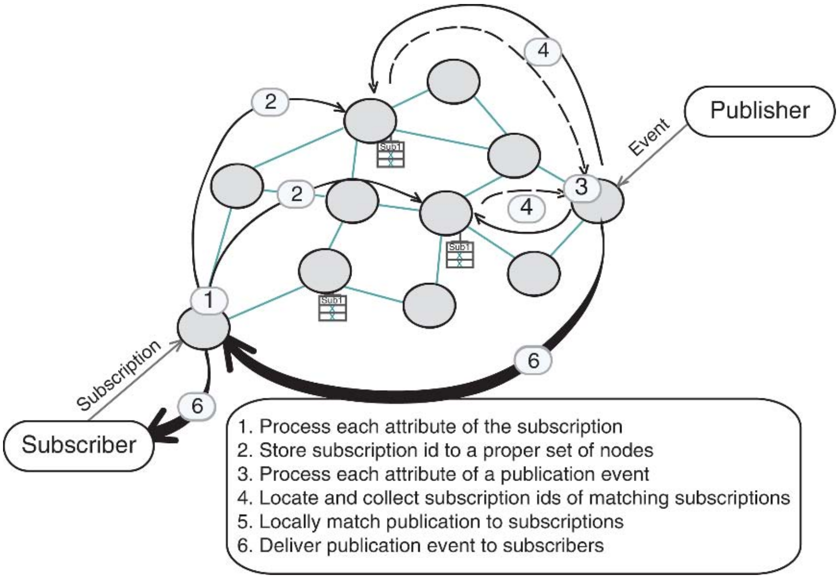
Key Applications

In recent years one notices a proliferation of data-intensive and compute-intensive networked applications aiming for efficiency, scalability, and self-organization. A key characteristic in many such applications is the massive amounts of data of various types and characteristics being generated continuously, from many different sources, at different times, and possibly at very high rates. Users can thus be inundated by the sheer volume of this data and are confronted with severe difficulties in accessing only the typically very small fraction of this data that is of interest to them. Hence, what is very much needed for such applications is an infrastructure that can offer decentralized, scalable, self-organizing asynchronous filtering of this massive information base.

Content-based publish subscribe systems operating in the distributed environment of a P2P overlay network appear as the appropriate infrastructure, and are used to design and implement such applications. A few representative example applications, which also indicate open research and development challenges, are listed below:

(i) *Information Feeds*

A classic example for publish/subscribe application infrastructures are data applications based on information feeds. In these applications publishers can be, for example, news agencies that publish



1. Process each attribute of the subscription
2. Store subscription id to a proper set of nodes
3. Process each attribute of a publication event
4. Locate and collect subscription ids of matching subscriptions
5. Locally match publication to subscriptions
6. Deliver publication event to subscribers

Peer-to-Peer Publish-Subscribe Systems. Figure 3. Multi-dimensional event and subscription processing in content-based P2P publish/subscribe systems.

news articles. Human subscribers declare their interests with proper content-based subscriptions involving numerical-attribute predicates such as date ranges, and string-attribute predicates such as words defining article title prefixes. In general, any RSS feed-like information dissemination system like stock exchange reports falls in this category. The distributed aspects occur when considering large, multi-national news agencies whose computers form a large, geographically-dispersed network. One can even further imagine alliances of such multi-nationals, increasing dramatically the application's scale.

(ii) *Grid Computing*

Consider a farm of computing clusters, each one including a number of computing nodes with heterogeneous characteristics involving, for example, various types of operating systems, hardware specifications, etc. and forming a peer-to-peer computing grid. Each node publishes its characteristics while possible users that wish to execute their programs are interested in specific node attributes (subscriptions) defining for example acceptable CPU speed ranges, minimum memory requirements, desirable operating system versions, etc.

(iii) *Pub/sub and the Web*

The web is of course a massive distributed data repository. Web information systems can be significantly enriched by adopting pub/sub technology. Overlay networks can be created consisting of nodes representing web pages. A pub/sub system over this overlay network can be used, for instance, to inform existing web users of interesting new pages being created, of updates to pages already marked as relevant, etc.

(iv) *Bio-Informatics Applications*

In the area of bioinformatics research, one might imagine a network of research data bases, belonging to specific governmental or private, non-for-profit research institutions, storing results or matching protein sequences and specific sub-sequences. The publishers in this case are the institutions' researchers publishing specific findings and subscribers are researchers inquiring for specific sequences' matching, as defined in their subscriptions.

Experimental Results

Typically, all proposed research solutions are accompanied by independent experimental results. So far the community has not produced widely acceptable general benchmarks, standard data sets, and workloads.

Data Sets

As mentioned, there are no generally-agreed upon real data sets to be used for testing research and development results. Some appropriate data sets are available, however, for some champion pub/sub applications, such as those based on RSS feeds. One example is data made available by the New York Stock Exchange (NYSE) (http://www.nysedata.com/nysedata/default. aspx).

Cross-references

▶ Channel-Based Publish/Subscribe
▶ Content-Based Publish/Subscribe
▶ Peer-to-Peer Content Distribution
▶ Peer to Peer Overlay Networks: Structure, Routing and Maintenance
▶ Peer-to-Peer System
▶ Publish/subscribe
▶ Publish/Subscribe over Streams
▶ Routing and Maintenance
▶ State-Based Publish/Subscribe
▶ Topic-Based Publish/Subscribe
▶ Type-Based Publish/Subscribe

Recommended Reading

1. Aberer K. P-Grid: a self-organizing access structure for P2P information systems. In Proc. Int. Conf. on Cooperative Inf. Syst., 2001.
2. Aekaterinidis I. and Triantafillou P. Internet scale string attribute publish/subscribe data networks. In Proc. Int. Conf. on Information and Knowledge Management, 2005.
3. Aekaterinidis I. and Triantafillou P. PastryStrings: a comprehensive content-based publish/subscribe DHT Network. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2006.
4. Banavar G., Chandra T., Mukherjee B., Nagarajarao J., Strom J., and Sturman D. An efficient multicast protocol for content-based publish-subscribe systems. In Proc. 19th Int. Conf. on Distributed Computing Systems, 1999.
5. Carzaniga A., Rosenblum D.S., and Wolf A.L. Design and evaluation of a wide-area event notification service. ACM Trans. Comput. Syst., 2001.
6. Castro M., Druschel P., Kermarrec A., and Rowstron A. Scribe: A large-scale and decentralized application-level multicast infrastructure. J. Select. Areas Commun., 2002.
7. Cugola G., Nitto E.D., and Fuggetta A. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. In Proc. 23rd Int. Conf. on Software Eng., 2001.
8. Fabret F., Jacobsen A., Llirbat F., Pereira J., Ross K., and Shasha D. Filtering algorithms and implementation for very fast publish/subscribe. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001.

9. Gupta A., Sahin O.D., Agrawal D., and Abbadi A.E. Meghdoot: content-based publish subscribe over p2p networks. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004.
10. Lehman T., Laughry S., and Wyckoff P. Tspaces: The next wave. In Proc. 32nd Annual Hawaii Int. Conf. on System Sciences, 1999.
11. Pietzuch P.R. and Bacon J. Hermes: a distributed event-based middleware architecture. In Proc. 1st Int. Workshop Distributed Event-Based Systems, 2002.
12. Ratnasamy S., Francis P., Handley M., Karp R., and Shenker S. A scalable content addressable network. In Proc. ACM Int. Conf. on Data Communication, 2001.
13. Rowstron A. and Druschel P. Pastry: Scalable and distributed object location and routing for large-scale peer-to-peer systems. In Proc. IFIP/ACM Int. Conf. on Dist. Syst. Platforms, 2001.
14. Stoica I., Morris R., Karger D., Kaashoek F., and Balakrishnan H. Chord: A scalable peer-to-peer lookup service for internet applications. In Proc. ACM Int. Conf. on Data Communication, 2001.
15. Terpstra W.W., Behnel S., Fiege L., Zeidler A., and Buchmann A.P. A peer-to-peer approach to content-based publish/subscribe. In Proc. 2nd Int. Workshop Distributed Event-Based Systems, 2003.
16. TIBCO TIB/Rendezvous. Tech. rep., White paper, Palo Alto, CA, http://www.tibco.com, 1999.
17. Triantafillou P. and Aekaterinidis I. Publish-subscribe over structured P2P networks. In Proc. 3rd Int. Workshop Distributed Event-Based Systems, 2004.

Peer-to-Peer Storage

Anwitaman Datta
Nanyang Technological University, Singapore,
Singapore

Synonyms

Distributed storage systems; Cooperative storage systems; Wide-area storage systems

Definition

Peer-to-peer (P2P) storage is a paradigm to leverage the combined storage capacity of a network of storage devices (peers) contributed typically by autonomous end-users as a common pool of storage space to store and share content, and is designed to provide persistence and availability of the stored content despite unreliability of the individual autonomous peers in a decentralized environment.

Historical Background

For diverse reasons including fault-tolerance, load-balance or response time, or geographic distribution of end users, distributed data stores have been around for a long while. This includes distributed databases, distributed file systems and *Usenet* servers among others. *Usenet* servers communicated among each other in a peer-to-peer manner, and replicated content.

While some redundancy is necessary for fault tolerance, replicating all content at all peers is a very special case of a peer-to-peer storage system, and is very inefficient. In general, an object is replicated at fewer locations. This leads to the problem of locating the object in the network. Plaxton et al's work on accessing nearby copies in a distributed environment [9] using key based routing is one of the seminal works, which subsequently influenced the design of many peer-to-peer storage systems. Advances in *structured overlay* networks address the problem of object location in a decentralized manner, that is "If an object is in the network, how to find it?" This is important in realizing an efficient decentralized peer-to-peer storage system.

Systems like *OceanStore* [7] aimed at archival storage, *Freenet* [3] for anonymous file sharing and distributed hash table based *DHash* storage layer of *CFS* [5] for a cooperative file system all store data at peers based on key associations determined by the structured overlay. These systems thus have the functionalities of object location and storage coupled together. However, note that locating and storing objects in a peer-to-peer system are in principle independent of each other. One can imagine the routing as a distributed index structure, which can store a pointer to the actual storage location, instead of the object itself. In the file sharing network *Napster*, the storage was peer-to-peer, however the indexing was in fact fully centralized, and very well illustrates the orthogonality of object location and storage issues. So in the rest of this entry the focus will be only on storage.

A critical requirement in a peer-to-peer storage system is to ensure that once a user (application) stores an object, this object should be available and persist in the network, notwithstanding the unreliability of individual peers and membership dynamics (churn) in the peer-to-peer network. This throws open a host of interesting design issues. For resilience, redundancy is essential. Redundancy can be achieved either by replication or using coding techniques, using schemes similar to the *RAID* technique [8]. While coding is in principle storage space efficient for achieving a certain level of resilience, it leads to various kinds of overheads, including computational overhead, and in the context of peer-to-peer systems, communication overhead and even storage overhead to keep track of encoded object fragments, thus making coding mechanisms worthwhile only for relatively larger or rarely accessed objects and applications like archival storage.

Redundancy is lost unless replenished because of departure of peers from the system. Trade-off considerations of redundancy maintenance effort and resilience led to the design of different maintenance strategies [2,6,10].

Note that file sharing systems – early ones like *Napster* and *Gnutella* as well as more recent ones like *Kazaa* and *BitTorrent* – also store and provide access to stored objects. However they are not reliable storage systems. In file sharing networks, users store locally only files they are interested in, and allow others to download the same. However there is no explicit intention to provide a highly available and persistent storage, and hence there are no mechanisms for redundancy management. The design of these systems is often focused on improving the efficiency of search and data transfer, while the content available in the network is considered ephemeral. Nevertheless, popular content may get so widely replicated that it is coincidentally (but not by design) always available.

Foundations

Distributed storage systems have traditionally been managed in a centralized manner, for instance in distributed databases and distributed file systems. The advent of decentralized file sharing networks demonstrated the potential as well as feasibility of decentralized peer-to-peer storage systems composed of autonomous peers. Peer-to-peer storage systems use the combined capacity of the peers to provide storage functionality to end users.

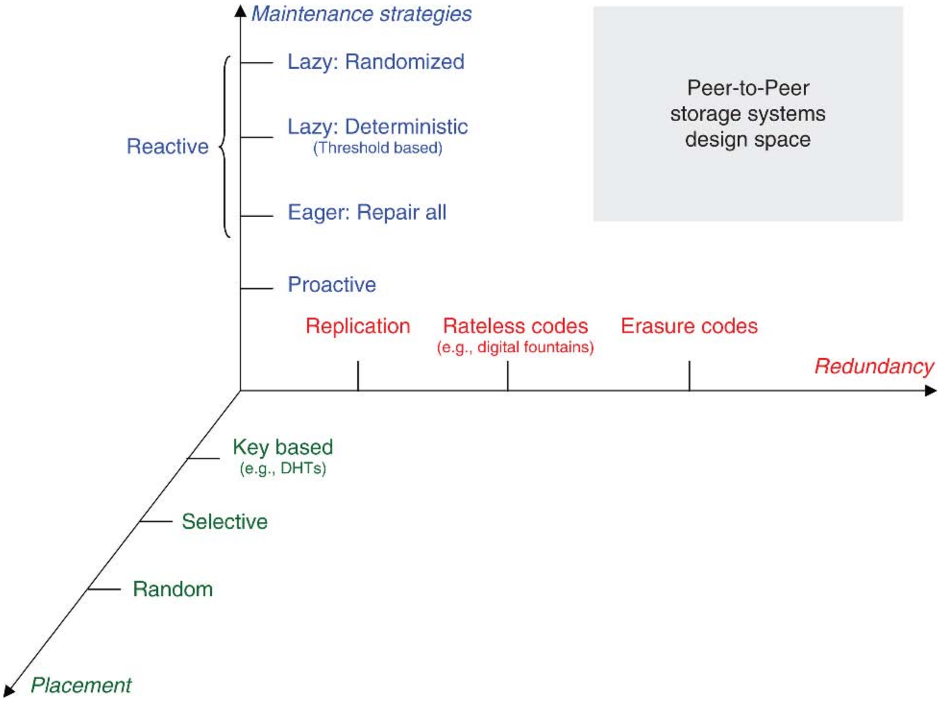
There are several reasons to have such a distributed storage. Multiple users may share and access some stored objects – data, files, etc. Individual users may not have the capacity to store all the objects they wish to access. Furthermore, by storing the same objects at other peers (and reciprocating by storing other users' objects), all peers benefit from an automatic back up service. Against these advantages, the main drawbacks include the unreliability of individual peers, who may leave and re-join autonomously, or even leave permanently, as well as transient communication failures, and delay in accessing objects stored only at a remote peer. Thus the main functionality of peer-to-peer storage systems is to make the distribution of stored objects transparent to the end users, even while users benefit from this distribution.

File sharing networks can be viewed as a special case of a peer-to-peer storage system. However they are not designed to guarantee availability or persistence of stored content, which are essential for a storage system. Resilience and performance in terms of access cost and latency pose some of the crucial challenges in realizing peer-to-peer storage systems.

Resilience of storage systems is measured in terms of two metrics - availability and durability (persistence). Availability of an object within a period of time is essentially the time averaged probability that it is accessible at any random time within that period of time. Durability of an object is the probability that it persists in the system indefinitely (or long enough, depending on the application requirements) under an assumption of a worst case failure scenario.

Consequently, crucial to peer-to-peer storage systems research are some of the following questions: What kind of redundancy is most efficient from various perspectives including storage overhead as well as access and maintenance costs and latency, and implementation complexity? What minimal redundancy is necessary to meet a desired level of resilience? Which maintenance strategy to apply to maintain the necessary redundancy? Which peers to store the redundant blocks in, based possibly on issues like reliability of individual peers, locality and load? The following delves into these. Figure 1 summarizes some of the important design factors.

In a decentralized setting, it is not only essential to ensure that stored objects stay available and persist over time despite changes in the network membership – churn – caused by peers leaving and (re-)joining the network, but also it is important to locate the stored objects efficiently. Some of the early storage systems like *OceanStore* [7] use one of the precursors [9] of contemporary structured overlays to locate objects. The basic idea is to assign to each peer an unique identifier, and also to each object an unique identifier from a same key-space, for instance using a hashing function. Objects are stored at peers which have identifiers closest to the object's identifier. When looking for an object, or the peer(s) at which an object is to be stored, the network of peers is typically searched in a greedy manner, by approaching peers with closer identifiers to the object's key, that is, using *key based routing*. Similar ideas are also used in *Freenet* [3], which aims at



Peer-to-Peer Storage. Figure 1. Some of the important design decisions that need to be taken into consideration when designing and deploying a peer-to-peer storage system.

providing anonymity to its users. Other subsequent storage systems have also followed this approach [5].

This traditional dual use of structured overlays for both routing (indexing) and storage tends to blur the difference between the two, and many people consider the structured overlay (e.g., distributed hash tables) to be the storage system also. In order to identify a full spectrum of design space of peer-to-peer storage systems it is crucial to understand that storage and indexing are actually two different but necessary ingredients for them. Combining these two sometimes simplifies the system architecture and implementation. However, the structured overlay may be used to store only pointers to the actual objects, which are placed using any independent criterion. One may also employ any other kind of directory (for example, *Napster* has a centralized directory) or indexing mechanism instead.

Resilience from Redundancy

Redundancy is essential to achieve resilience. Storage systems realize redundancy by using either replication or error correcting codes (e.g., erasure codes) as discussed next.

Replication Replication, i.e., mirroring, stores the same object on multiple nodes. It is normally efficient when the size of the object is small, or the object is frequently accessed. Note that a large object may still be split in multiple fragments, and each of these fragments will then be replicated.

Erasure codes Erasure codes are a class of error-correcting codes, which can transform a M-fragment object into N ($N > M$) encoded fragments, such that the original object can be reconstructed from any M *out of the N* encoded fragments. This typically leads to a storage overhead slightly more than N/M. The rate of erasure codes r is defined as the fraction of fragments required for decoding, i.e., $r = \frac{M}{N}$. Based on the coding rate, erasure codes can be categorized into two types, fixed-rate codes in which N is limited and r has a fixed value, and rateless codes (for example *Digital Fountains* [4]) in which N is potentially unlimited and r approaches zero (therefore called rateless).

Although replication is sometimes regarded as a special case of erasure codes, a subtle difference exists between them, which has its implications on the maintenance of redundancy. For replication, every reintegrated replica can enhance the object availability. In contrast, for a fixed-rate erasure code, if the reintegrated fragment is identical to one of the existing fragments, it does not improve the availability of the whole object. Even with *M* fragments, one may still not be able to reconstruct the original object, unless the fragments are distinct. The above problem does not occur for rateless codes, in which unique fragments are generated, and thus every reintegrated fragment is useful. In this aspect, replication is more similar to rateless codes, instead of fixed-rate codes.

When accessing an object, if it had been stored in coded fragments, enough fragments need to be accessed first, and then the object needs to be decoded, causing computational and possibly communication overheads. The benefit of coding is that for the same targeted resilience, using coding techniques drastically reduces the storage overhead, alternatively said, for same storage overhead, coding provides a much higher resilience. However, depending on the implementation details, using coding techniques may also lead to additional storage overheads to keep track of the encoded fragments. For frequently accessed objects or small objects, the overheads associated with coding may thus outweigh the benefits.

Because of these considerations, a rule of thumb is small or frequently accessed objects are typically replicated, while large or rarely accessed objects erasure coded. Another option is to use a hybrid approach [11]. Erasure coded fragments are used to achieve high persistence at a low storage overhead. Replication is used for performance.

Maintaining Redundancy

Individual peers may be occasionally unavailable because users go offline, machines fail, or the network gets disconnected. Typically, if there is sufficient redundancy, such *temporary churn* has marginal effect on the availability of a stored object. Furthermore temporary churn does not directly affect long term persistence, since the peers join back, bringing back in the network whatever is stored in them. However, over a period of time, some participating peers may leave the system permanently, in turn leading to permanent loss of redundancy. *Permanent churn* thus makes the system more vulnerable to temporary churn, and unless mitigated, leads to permanent loss of stored objects, thus adversely affecting persistence/durability. So while redundancy provides resilience against temporary churn, maintenance strategies are necessary to restore the lost redundancy and make the system resilient against permanent churn.

Henceforth both coded fragments or replicas will be referred as fragments. The simplest maintenance mechanism is to probe periodically all the peers which are supposed to store redundant fragments of an object, and whenever a probe fails, reactively replenish the redundancy be reintegrating a new suitable redundant fragment. Example storage systems employing this strategy include CFS [5]. This strategy is referred to as an *eager maintenance* strategy. Such an eager reactive (The simultaneous use of the adjectives eager and reactive is admittedly oxymoronic, and is a vestige of the historical development and nomenclature of various maintenance mechanisms.) maintenance mechanism means the system always operates in a state where redundancy level remains constant apart from temporary reduction between repair periods. It has been empirically observed [2] that eager (reactive) maintenance strategy is bandwidth expensive.

A periodic probing and reintegration of all unavailable fragments ignores the fact that many of the fragments are only temporarily unavailable, and will be restored back in the system automatically once the corresponding peers join back. Consequently, a lot of overhead in regenerating and transferring new redundant fragments is avoidable if only the system can wait long enough for some of the absent peers to rejoin. This observation is key to the design of the first lazy repair (reactive) strategy, used in the *TotalRecall* system [2]. All nodes are probed periodically, and repairs are initiated only when less than a certain threshold $T_a > M$ of nodes (and corresponding data) are available. Thus to say, when an object has no more than T_a fragments available in the system, then a repair process for the object is initiated so that at the end of the repair process all *N* fragments are again available. As long as an object has a redundancy more than the parameter T_a, there is no maintenance. This lazy maintenance mechanism saves overheads caused by transient failures. However if the redundancy reduces below this threshold, then it is considered imprudent to further delay repair operations. So once the threshold is breached for an object, *TotalRecall* regenerates all the corresponding unavailable fragments. Thus, this reactive lazy mechanism has a deterministic trigger for initiating repairs.

While saving on bandwidth in comparison to the above mentioned eager reactive maintenance strategy, the threshold based lazy repair strategy suffers from several undesirable effects. First of all, by waiting for the redundancy to fall below a threshold, the system is allowed to degenerate and become more susceptible as compared to other systems which have the same maximum redundancy N. Secondly, while most of the time this approach does not use the bandwidth even if it were available, once the threshold is breached, this approach tries to replenish all the missing fragments at once, thus causing bandwidth spikes. Finally, between the maintenance spikes, as redundancy falls, the available fragments are accessed more frequently, causing access load imbalance, particularly overloading the available peers. Two different works try to address these shortcomings.

A randomized variant of the lazy repair strategy [6] is to probe only a fraction of the stored fragments randomly (uniformly), until a minimal $T_b \geq M$ number of live fragments are detected. Thus a random number $T_b + X$ of probes (determined according to a probability distribution which in turn depends on the actual number of live fragments) will be required to locate T_b live fragments. Then X fragments which were detected to be unavailable are replaced by the system. The beauty of this randomized approach is that the expected value of X adapts with the number of live fragments. If there are fewer live fragments, then X will typically be large, and vice versa. Normally X can be typically much smaller than the total number of unavailable fragments at that instant. As a consequence, the repair process is continuous – thus available bandwidth is used more judiciously, avoiding spikes. The randomized repair process is naturally more aggressive when more redundant fragments are missing, while if very few fragments are missing then there are very few repairs.

While the randomized lazy repair strategy [6] tries to strike a balance between the periodic repair and the threshold based deterministic lazy repair strategies, and as a consequence the bandwidth usage is continuous and smoother, another independent approach [10] makes it an explicit goal to not exceed a bandwidth budget per unit time. Subject to the bandwidth budget per unit time it proactively creates new replicas. In contrast to the previously mentioned approaches all of which react to lost redundancy, the proactive approach does not aim to reduce overall bandwidth usage, but instead tries to ensure that by not exceeding the per unit time maintenance bandwidth budget, a better bandwidth provisioning can be achieved, and thus the maintenance operation does not interfere with applications. Another consequence of this approach is that typically enough redundancy is available to ensure equitable load distribution. However, a maximum redundancy threshold needs to be defined in using such a proactive replication approach to make a judicious use of the storage capacity.

Placement Strategies

A final system design issue is determining the placement strategy to be used to store the objects. One of the most widely used approach is to determine the placement of objects based on keys. In this scheme, all the peers as well as objects are assigned unique keys (e.g., by hashing), and then objects are stored at peers with closest or similar keys. The peers themselves communicate among each other by forming a structured overlay network, and messages are routed based on key similarity (key based routing). Such an object placement strategy can be seen in systems like *CFS* [5], *Oceanstore* [7], *Freenet* [3] and *Tempo* [10] to name a few. This approach essentially combines the storage of the object with the search mechanism.

Alternatively, object placement may be decoupled from the search mechanism, thus giving the systems designer, or even the applications using the storage system much more flexibility in choosing the storage location. This choice may be random (for example, in *TotalRecall* [2]), or based on reliability prediction derived from the history of peers' availability, or based on proximity (locality) from the end users accessing the object, load at peers, or other considerations like storing the object within a specific domain. Furthermore, this choice may be made by the peer-to-peer storage system designer or its administrator, or even independently by the applications and end users using the storage system.

Against this flexibility, the main disadvantage of decoupling storage from search is that one then needs some kind of directory service (potentially realized with a structured overlay) to perform the search functionality. The search provides pointer(s) to the stored object fragments. This creates additional storage and maintenance (of the pointers) overheads. Thus, in contrast to the key based storage approach, there is an additional level of indirection, and the storage system needs to explicitly keep track of any changes, including network level changes like change of peers' network address, unlike the structured overlay (key based storage) approach, where the structured overlay maintenance mechanism takes care of such changes and simplify storage systems design.

Analysis Techniques

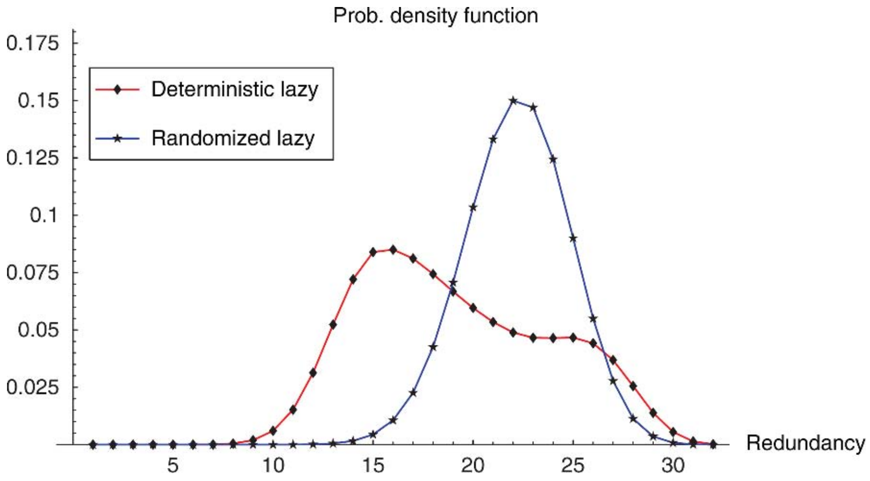
There are several approaches to analyze and study the behavior of peer-to-peer storage systems to better understand and refine its algorithms and design, make better parameter choices and validate its implementation.

Static resilience: Given a certain amount of redundancy, if a certain fraction of the peers are not accessible, either because they left the network, or the machines temporarily crashed, or because of communication problems, one can determine the probability that any specific object will become inaccessible or permanently lost. For example, for pure replication, if there are ρ replicas, and each peer storing a replica is available only p_{on} fraction of the time (randomly and independently from the other peers replicating the object), then the probability of the object becoming unavailable is $(1 - p_{on})^{\rho}$. Such a static resilience analysis gives a system designer a reference for determining an adequate redundancy to tolerate a certain degree of membership dynamics.

Time-evolution: Static resilience does not take into account the combined effect of membership dynamics, which leads to loss of redundancy over time, and repairing strategy, which regenerates redundancy, thus improving the health of the storage system. Given a particular level of network dynamics, the choice of maintenance strategy affects the resilience of the storage system, as well as the overheads incurred. A pragmatic system design thus needs to take into account the combined effect of individual peers' unreliability as well as the specifics of the deployed redundancy maintenance strategy.

Of particular interest is the actual probability distribution of object redundancy under the combined effect of churn and maintenance. This is in contrast to the maximum or average redundancy of objects in the system, based on which static resilience is typically estimated. Objects with smaller actual redundancy at a time instant are more vulnerable to become temporarily unavailable or even permanently lost. Also access to these corresponding objects causes higher load at the peers storing the object fragments or replicas. Finally, more effort and bandwidth is required to restore redundancy for the objects with fewer fragments, thus the maintenance operation witnesses spikes in bandwidth usage. The probability distribution provides a more fine-grained state of the storage system's health, by showing what fraction of all the stored objects are expected to have what level of actual redundancy. Such information can be obtained by studying the time evolution of storage systems [6,12].

For example, Fig. 2 shows the probability density function of the actual redundancy of objects when the



Peer-to-Peer Storage. Figure 2. A snapshot of the probability distribution of the available number of object fragments in the storage system when using (i) Deterministic lazy (reactive) maintenance, (ii) Randomized lazy (reactive) maintenance. An any *8 out-of 32* fragments (rate 0.25) erasure code was used to store the objects. Churn was simulated synthetically, such that online peers could go offline with a probability of 0.2 and offline peers could come back online with a probability of 0.1, such that on an average one third of the peers were online, i.e., $p_{on} = 1/3$. The thresholds T_a and T_b for the two variants of the lazy maintenance strategies were chosen such that the aggregate bandwidth usage for maintenance was the same in both experiments. This figure has been obtained from [6].

deterministic or the randomized lazy maintenance algorithms are used for an otherwise identical specific scenario, that is, same redundancy for objects, same churn level, and algorithm parameters chosen such that total bandwidth usage in both cases are comparable. There is a greater area under the curve corresponding to low redundancy if the deterministic lazy maintenance mechanism is used, in comparison to the case when the randomized variation is used.

Key Applications

Peer-to-peer storage systems have been used to realize traditional applications like file systems [2,5], backup [1] and archival storage [7]. A peer-to-peer backup system has several advantages in comparison to traditional offline backup systems based on secondary storage. It is easier to verify that the backup has indeed worked correctly (in comparison to physical medium like magnetic tapes), and backup as well as maintenance of the backed up data and its restoration whenever necessary can all be completely automated. Furthermore, peer-to-peer storage is not vulnerable to geographically localized catastrophes. This gives large corporations with geographically dispersed offices strong economic incentives to use their employees' desktop computers as a private corporate peer-to-peer storage infrastructure, instead of managing a separate secondary storage based backup. Similarly, individuals can back-up their data by relying on the resource pooled in a public peer-to-peer storage network.

Peer-to-peer storage systems also find use in web proxies for caching, content distribution and file sharing [3] applications. Peer data management systems too rely on an underlying reliable storage service.

Cross-references

▶ Peer Data Management System
▶ Peer-to-Peer Content Distribution
▶ Peer to Peer Overlay Networks: Structure, Routing and Maintenance
▶ Updates and Transactions in Peer-to-Peer Systems

Recommended Reading

1. http://www.cleversafe.org/dispersed-storage
2. Bhagwan R., Tati K., Cheng Y., Savage S., and Voelker G.M. TotalRecall: systems support for automated availability management. In Proc. 1st USENIX Symp. on Networked Systems Design & Implementation, 2004.
3. Clarke I., Miller S.G., Sandberg O., and Wiley B. Protecting free expression online using Freenet. IEEE Internet Computing, 6(1):40–49, 2002.
4. Codes R. and Shokrollahi A. IEEE Trans. Inform. Theory, 2006.
5. Dabek F., Kaashoek F., Karger D., Morris R., and Stoica I. Wide-area cooperative storage with CFS. In Proc. 18th ACM Symp. on Operating System Principles, 2001.
6. Datta A. and Aberer K. Internet-scale storage systems under churn – a study of the steady state using Markov models. In Proc. Sixth IEEE Int. Conf. on Peer-to-Peer Computing, 2006.
7. Kubiatowicz J., Bindel D., Chen Y., Czerwinski S., Eaton P., Geels D., Gummadi R., Rhea S., Weatherspoon H., Weimer W., Wells C, and Zhao B. OceanStore: an architecture for global-scale persistent storage. In Proc. 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 2000.
8. Patterson D., Gibson G.A., and Katz R. A case for redundant arrays of inexpensive disks (RAID). In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988.
9. Plaxton C.G., Rajaraman R., and Richa A.W. Accessing nearby copies of replicated objects in a distributed environment. In Proc. ACM Symp. on Parallel Algorithms and Architectures, 1997.
10. Sit E., Haeberlen A., Dabek F., Chun B.G., Weatherspoon H., Morris R., Frans Kaashoek M., and Kubiatowicz J. Proactive replication for data durability. In Proc. 5th Int. Workshop Peer-to-Peer Systems, 2006.
11. Williams C., Huibonhoa P., Holliday J., Hospodor A., and Schwarz T. Redundancy management for P2P storage. In Seventh IEEE Int. Symp. on Cluster Computing and the Grid, (CCGrid), 2007.
12. Wu D., Tian Y., Ng K.-W., and Datta A. Stochastic analysis of the interplay between object maintenance and churn. Elsevier Journal of Computer Communications, Special Issue on Foundations of Peer-to-Peer Computing, Elsevier, 2007.

Peer-to-Peer System

Wojciech Galuba, Sarunas Girdzijauskas
EPFL, Lausaune, Switzerland

Synonyms

Peer-to-peer network; peer-to-peer overlay

Definition

A peer-to-peer system is a computer network which enables peers to share the network resources, computational power and data storage, without relying on a central authority. Most commonly, peer-to-peer systems form *overlay networks* deployed in the Internet and are used for file sharing, realtime data streaming and computationally intensive tasks.

Key Points

In contrast to client-server systems, peer-to-peer systems consist of interconnected peers of similar capabilities and responsibilities, where the peers can act as both servers and clients. Most commonly, the architecture of the peer-to-peer systems is flat and all peers are assumed to be functionally equal. However, a number of peer-to-peer systems employ hierarchical architecture where some peers (superpeers) act as local servers for the subsets of regular peers. It is also widely accepted in peer-to-peer systems, where some services can be provided by a centralized component, in particular system bootstrapping, authentication etc.

Peer-to-peer systems are designed to be self-organizing and to scale well with the number of participating peers. Each peer contributes a different amount of resources and peer-to-peer systems employ various methods to distribute the load evenly across these resources. Good load balancing is crucial to the scalability of peer-to-peer systems. To account for frequent peer departures and arrivals and to maintain high availability peer-to-peer systems replicate the services and data across several peers for redundancy.

Peer-to-peer systems have a wide range of applications in such areas as the Internet infrastructure (e.g., DNS), file sharing (e.g., Kazaa [3]), communication and data streaming (e.g., Skype [4]), distributed computing (e.g., BOINC [1]), and can even make use of human presence at the participating peers (e.g., Galaxy Zoo project [2]). In addition, the decentralized nature of peer-to-peer systems allows the peer-to-peer applications to be highly resistant to censorship.

High robustness, scalability and lack of the single-point-of-failure make peer-to-peer systems a viable alternative to large client-server systems.

Cross-references

▶ Distributed Hash Table

Recommended Reading

1. Berkeley Open Infrastructure for Network Computing Homepage. http://boinc.berkeley.edu/.
2. Galaxy Zoo Homepage. http://galaxyzoo.org/.
3. Kazaa Homepage. http://www.kazaa.com/.
4. Skype Homepage. http://www.skype.com/.

Peer-to-Peer Web Search

Gerhard Weikum
Max-Planck Institute for Informatics, Saarbrücken, Germany

Definition

The peer-to-peer (P2P) computing paradigm is an intriguing alternative to centralized search engines for querying and ranking Web content. In a P2P network with many thousands or millions of computers, every peer can have locally compiled content such as recently visited or thematically gathered Web pages, and can employ its own, potentially personalized search engine on the locally indexed data. In addition, queries can be forwarded to judiciously chosen other peers for collaborative evaluation. Such P2P architectures enable keyword search and result ranking on the network-wide global content. Thus, all peers together form a P2P search engine. Conversely, such P2P architectures could be utilized for scalable, distributed implementations of Web indexing with appropriate partitioning across the nodes of a large server farm.

Historical Background

Peer-to-peer (P2P) systems aim to provide scalable and self-organizing ways of loosely coupling thousands or millions of computers in order to jointly achieve some global functionality [14]. In the last decade, very successful systems of this kind have been built. Most notably, these include *file-sharing networks* such as Gnutella or BitTorrent, and IP telephony like Skype and other collaborative messaging services. They organize peers in so-called *overlay networks* on top of the standard Internet infrastructure, and use various forms of *epidemic dissemination* or distributed data structures like *distributed hash tables (DHTs)* such as Chord or Pastry. The basic functionality that underlies many of these systems is the distributed and dynamic maintenance of a dictionary with efficient support for exact-match key lookups.

Web search requires much richer functionality than exact-match lookups: keyword queries combine multiple dimensions of a very-high-dimensional data space in an ad hoc manner so that standard multi-dimensional indexes are not applicable, and they require ranking of query results based on statistics about local and global keyword frequencies. On the other hand, it seems very natural to build Web search in a P2P manner as the producers and owners of Web pages are widely distributed and autonomous. In fact, the standard approach of crawling the Web for centralized search engines can be seen as rather artificial, but has advantages regarding system management and commercial services such as query-related advertisements.

Approaches to P2P Web search are reminiscent of earlier work on *distributed information retrieval (IR)*, most notably, various kinds of *metasearch engines* where queries are routed to judiciously chosen search providers [15]. However, the P2P setting is much more challenging regarding the enormous scale of the underlying data sources, the dynamics of the system, and the autonomy of the individual peers.

Foundations

Peer-to-peer (P2P) Web search has been studied from two perspectives: *global computing (GC)* and *social computing (SC)*. The GC approaches consider peers that dedicate all their storage and computing resources to the P2P network. The network as a whole emulates a traditional search engine architecture by partitioning a global Web index and assigning partitions and query load to peers. The SC approaches, on the other hand, consider architectures where each peer corresponds to one user, and emphasizes the autonomy of peers, with every peer having full control over its local content and extent of sharing it with other peers. Both GC and SC can be embedded in a server-farm environment with a high-speed physical network or in a geographically distributed environment over the wide-area Internet.

P2P Global Computing for Scalable Search-Engine Functionality

For *indexing the Web*, commercial search engines use large data centers consisting of thousands of low-end computers, carefully designed parallelism and redundancy, and customized software with very low overhead [2]. The key technique to sustain a peak throughput of thousands of queries per second with sub-second response times is *data partitioning*. Index entries – postings that consist of a keyword id, a page id, and a score – are hashed on page id, and each resulting hash partition is assigned to one computer in a very large *cluster* (with hundreds or thousands of nodes). A query with one or more keywords is simply sent to all nodes for parallel evaluation. The load is perfectly balanced across the nodes of the cluster, so that the approach scales up extremely well. For failure resilience, high availability, and higher throughput, an entire cluster is often replicated sufficiently.

The above architecture is distributed, but it is not a P2P approach, for it assumes a fixed, reasonably time-invariant system configuration with a high-speed homogeneous network between nodes. If one tried to carry this design over to a setting with a dynamically varying number of peers that communicate over a high-latency wide-area network, the fine-grained parallelism would probably result in a poor cost/performance ratio. Thus, a geographically distributed architecture for Web indexing needs more sophisticated techniques and is still viewed as a research challenge, but promising work is on its way [1].

The current approaches differ in their ways of partitioning the data across peers, the overlay networks that are employed, their load balancing mechanisms, and their methods for caching and replication, which in turn influences query processing strategies. As for *data partitioning*, three common ways are: hash-partitioning on page ids (documents), hash-partitioning on keywords (terms), or thematic clustering based on page contents or query logs. *Load balancing* needs to counter skewed distributions of both index-list lengths and query popularities for different keywords; many techniques from the distributed computing literature are applicable. *Caching* can consider different granularities like index lists for individual keywords or entire query results, and needs to employ appropriate strategies for cache refreshing and replacement. Finally, efficient algorithms for *distributed top-k query processing* are called for.

P2P Social Computing with Autonomous Peers

In the SC line of work, an architecture is pursued where every peer has a powerful local search engine, with its own crawler, indexer, and query processor. Such a peer can compile its own content from thematically focused crawls and other sources, and make this content available in a P2P overlay network. Search requests issued by a peer can first be executed locally, on the peer's locally indexed content. When the recall of the local search result is unsatisfactory, the query can be forwarded to a small set of other peers that are expected to provide thematically relevant, high-quality and previously unseen results. Deciding on this set of target peers is the *query routing* problem in a P2P search network, also known as *collection selection*. Subsequently, the actual search on the chosen target peers requires efficient algorithms for *top-k query processing*. Search results are then returned by different peers and need to be meaningfully merged, which entails specific problems for *result ranking*. Both query routing and result ranking can build on various forms of *distributed statistics*, computed in a decentralized way and aggregated and disseminated in a scalable P2P manner (using compact synopses such as Bloom filters or hash sketches and leveraging the DHT infrastructure).

Query Routing

A practically viable query routing strategy needs to consider the similarity of peers in terms of their thematic profiles, the overlap of their contents, the potential relevance of a target peer's collection for the given query, and also the costs of network communication and peer-specific processing loads.

Traditionally, the most important measure for assessing the benefit of a candidate target peer for a given query is the estimated relevance of the peer's overall content for the query. This standard IR measure can be estimated frequency statistics over the query's keywords (terms in IR jargon). In conventional, document-oriented IR, these would be term frequencies (tf) within a document and the so-called inverse document frequencies (idf), the reciprocal of the global number of documents that contain a given term. In P2P IR, the estimation is based on the overall content of a peer as a whole. Instead of tf, the *document frequency (df)* of a peer is considered, which is the total number of documents that contain the term and are in the peer's collection; and instead of idf, the *inverse collection frequency (icf)* is considered, which is the reciprocal of the total number of peers that contain (at least one document with) the term. These basic measures are combined into a relevance or query-specific quality score for each candidate peer. There are various models for the combined scores; among the most cited and best performing models are *CORI* [4], based on probabilistic IR, and statistical language models adapted to the setting of P2P collection selection [9]. The Decision-Theoretic Framework (DTF) [11] provides a unified model for incorporating quality measures of this kind as well as various kinds of cost measures.

Selecting peers solely by query relevance, like CORI routing, potentially wastes resources when executing a query on multiple peers with highly overlapping contents. To counter this problem, methods for estimating the overlap of two peers' contents have been developed. These estimates are then factored into an *overlap-aware query routing* [10] method by using a weighted combination of peer quality and overlap (or novelty) as ranking and decision criterion.

Query routing decisions are typically made at query run-time, when the query is issued at some peer. But the above methods involve directory lookups, statistical computations, and multi-hop messages; so it is desirable to precompute preferred routing targets and amortize this information over many queries. A technique for doing this is to encode a similarity-based precomputed binary relation among peers into a *Semantic Overlay Network (SON)* [5]. The routing strategy would then select target peers only or preferably from the SON neighbors of the query initiator.

Search Result Ranking

When a query returns results that have been obtained from different peers, the scores that the peers assign to them are usually not comparable. The reason is that different peers may use different statistics, for example, for estimating the idf value of a term which is crucial for weighting the importance of different query terms, or they may even use completely different IR models. This situation leads to the problem of *result merging*. It is addressed by re-normalizing scores from different peers to make results meaningfully comparable [15]. A variety of such methods exist in the literature, some using only the peer-specific scores and some aggregated measures about peers (e.g., the total number of documents per peer), some using sampling-based techniques, and some using approaches that first reconstruct the necessary global statistics (e.g., global document frequencies for each term) for optimal re-normalization of scores.

Web search ranking usually also considers the query-independent authority of pages as derived from link analysis, and a P2P network is a natural habitat for such "social ratings". *Link analysis* algorithms such as PageRank are centralized algorithms with very high memory demand. Executing them in a distributed manner would allow scaling up to even larger link graphs, by utilizing the aggregated memory of a P2P system. Various decentralized methods have been developed to this end, including a general solution to the spectral analysis of graphs and matrices [8], which underlies the PageRank computation.

Most of these methods assume that the underlying Web graph can be nicely partitioned among peers. In contrast, a P2P system with autonomous peers faces a situation where the Web pages and links that are known to the individual peers are not necessarily disjoint. The JXP algorithm [12] computes global authority measures such as PageRank in a decentralized and scalable P2P manner, when the Web graph is spread across autonomous peers and peers' local graph fragments overlap arbitrarily, and peers are (a priori) unaware of other peers' fragments. The scores computed by JXP provably converge to the same values that would be obtained by a centralized PageRank computation on the full Web graph. These kinds of algorithms seem to be highly relevant also for analyzing authority and reputation measures in large-scale social networks.

Key Applications

The technology for P2P Web search has many potential applications, ranging from keyword search on the Web or in blogs, possibly in a personalized manner and exploiting social-network affinities among users, all the way to more "semantic" search capabilities in large enterprises, scholarly communities, federations of digital libraries, and distributed Internet archives. The latter may include searching XML, RDF, or multimedia data as well as providing temporal querying and other forms of enhanced search capabilities. With richer functionality, centralized systems face scalability bottlenecks, whereas decentralized approaches can leverage the fact that the underlying information is naturally distributed at a large scale.

Future Directions

Today, there is still no P2P Web search system that would scale anywhere near the sizes of the major commercial search engines. But as the Web continues to grow, search functionality is becoming richer (e.g., by personalization), and workloads are becoming much more demanding, the P2P paradigm is likely to gain more momentum for Web applications. Currently, there is a variety of research prototypes (e.g., [3,6,7,13]), including some that offer open-source software.

Cross-references

► Distributed Hash Table
► Link Analysis
► Peer-to-Peer Data Management
► Social Networks
► Top-k XML Query Processing

Recommended Reading

1. Baeza-Yates R.A., Castillo C., Junqueira F., Plachouras V., and Silvestri F. Challenges on distributed web retrieval. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 6–20.
2. Barroso L.A., Dean J., and Hölzle U. Web search for a planet: the Google cluster architecture. IEEE Micro, 23(2):22–28, 2003.
3. Bender M., Michel S., and Parreira J.X., and Crecelius T. P2P web search: make it light, make it fly. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 164–168.
4. Callan J.P., Lu Z., and Croft W.B. Searching distributed collections with inference networks. In Proc. 18th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 21–28.
5. Crespo A. and Garcia-Molina H. Semantic overlay networks for P2P systems. In Proc. 3rd Int. Workshop Agents and Peer-to-Peer Computing, 2004, pp. 1–13.
6. Cuenca-Acuna F.M., Peery C., Martin R.P., and Nguyen T.D. PlanetP: using gossiping to build content addressable peer-to-peer information sharing communities. In Proc. 12th IEEE Int. Symp. High Performance Dist. Comp., 2003, pp. 236–249.
7. Kalnis P., Ng W.S., Ooi B.C., and Tan K.-L. Answering similarity queries in peer-to-peer networks. Inf. Syst., 31 (1):57–72, 2006.
8. Kempe D. and McSherry F. A decentralized algorithm for spectral analysis. In Proc. 36th Annual ACM Symp. on Theory of Computing, 2004, pp. 561–568.
9. Lu J. and Callan J.P. Content-based retrieval in hybrid peer-to-peer networks. In Proc. Int. Conf. on Information and Knowledge Management, 2003, pp. 199–206.
10. Michel S., Bender M., Triantafillou P., and Weikum G. IQN routing: integrating quality and novelty in P2P querying and ranking. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 149–166.
11. Nottelmann H. and Fuhr N. Comparing different architectures for query routing in peer-to-peer networks. In Proc. 28th European Conf. on IR Research, 2006, pp. 253–264.
12. Parreira J.X., Donato D., Michel S., and Weikum G. Efficient and decentralized pageRank approximation in a peer-to-peer web search network. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 415–426.
13. Podnar I., Rajman M., Luu T., Klemm F., and Aberer K. Scalable peer-to-peer web retrieval with highly discriminative keys. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 1096–1105.
14. Steinmetz R. and Wehrle K. Peer-to-peer systems and applications. Springer, 2005.
15. Weiyi M., Yu C.T., and Liu K.-L. Building efficient and effective metasearch engines. ACM Comput. Surv., 34(1):48–89, 2002.

Performance Analysis of Transaction Processing Systems

ALEXANDER THOMASIAN
Thomasian and Associates, Pleasantville, NY, USA

Synonyms

Queueing analysis; Probabilistic analysis; Cache performance; Storage systems; Concurrency control

Definition

The performance of *transaction (txn) processing (TP)* systems and more generally *database management systems (DBMSs)* is measured on operational systems, prototypes, and benchmarks. Probabilistic and queueing analyses have been used to gain insight into TP system performance, but also to develop capacity planning tools. The following is considered: (i) queueing analysis of processors and disks, (ii) *queueing network models (QNMs)* of computer systems, (iii) techniques to estimate the database buffers miss rate, (iv) factors affecting RAID performance, (v) *concurrency control (CC)* methods for high data contention TP systems and their analyses.

Historical Background

Early performance studies of TP were concerned with processor or *central processing unit (CPU)* scheduling. *Queueing network models – QNMs* were developed in the 1970s to estimate delays at active computer system resources, i.e., CPU and disks. The effect of passive resources, such as the memory size constraint, was later incorporated into capacity planning tools for TP systems [5]. The *Transaction Processing Council's (TPC's)* debit–credit benchmark in 1985 compares the performance of TP systems using their throughput at which a certain txn response time is reached (http://www.tpc.org). Buffer/cache management policies in processors, databases, and disk controllers have been studied since the 1970s. Coherence issues of CPU caches in multiprocessors and database buffers in shared disk systems gained importance in 1980s. Magnetic disks invented in 1950s have a significantly improved capacity and transfer rate, but not random access time. This led to the proposal and analysis of numerous disk arm scheduling policies. The 1988 *Redundant Array of Independent Disks (RAID)* classification provided renewed impetus to improve the performance and reliability of storage systems [1]. Shared-everything, -disk, and -nothing computer organizations have limitations for high-performance TP and DBMS applications, i.e., the first two pose cache coherence problems and shared-nothing systems are susceptible to processing time skew. (For example, there is a twofold increase with exponentially distributed processing times at four nodes, since $\sum_{i=1}^{4} 1/i = 2.08$.) CC has limited effect on TP performance in modern DBMSs, but this was not so in early relational DBMSs with coarse granularity locking, therefore many CC methods were proposed and evaluated [10].

Foundations

CPU Scheduling. Processing of txns at the CPU has been modeled by an M/G/1 queueing model, which consists of a *First-Come, First-Served (FCFS)* queue and a single server [4]. M stands for Poisson arrivals with rate λ, G stands for a general service time distribution with $\overline{x^i}$ as the *i*th moment. The server utilization is $\rho = \lambda\overline{x} < 1$, the mean waiting time at the queue is $W_{M/G/1} = \lambda\overline{x^2}/(2(1-\rho))$, and the mean response time $R = W_{M/G/1} + \overline{x}$ [4]. M/M/1 is a special case of M/G/1 with an exponential service time distribution with $\overline{x^2} = 2(\overline{x})^2$, so that the mean of the exponentially distributed response time is $R_1 = \overline{x}/(1-\rho)$. M/M/m has m servers, hence $\rho = \lambda\overline{x}/m$. For $m = 2$ there is $R_2 = \overline{x}/(1-\rho^2)$ (for $m > 2$ see [4]). For a single-server which is twice as fast as the previous servers: $R_3 = (\overline{x}/2)/(1-\rho)$. The same ρ is maintained in the three systems with R_i, $1 \leq i \leq 3$, by setting the arrival rate to 2λ, but utilizing two M/M/1 queues with uniform routing in the first case, so that the total processing capacity is 2μ in all three cases. There is $R_1 > R_2 > R_3$, where the first inequality reflects the resource sharing advantage of a shared queue, while the second inequality reflects the advantage associated with a single fast server when service times are not highly variable [4]. For K fork-join requests initiated at K M/M/1 queues the expected value of the maximum of K response times is: $R_k^{max} = H_K R$, where $H_K = \sum_{k=1}^{K} 1/k$ is the Harmonic sum. The components of a fork-join request are correlated, so that $R_2^{F/J} < R_k^{max}$, e.g., $R_2^{F/J} = (1.5 - \rho/8)R < R_2^{max}$. An approximate expression for $R_k^{F/J}$ for $K > 2$ has been used in the analysis of RAID5 disk arrays. With two txn classes the response time for the more urgent (class 1) txns can be improved by processing them at a higher priority than less urgent (class 2) txns. The arrival rates are λ_1 and λ_2 and the *i*th moments of service time are $\overline{x_1^i}$ and $\overline{x_2^i}$, respectively. With preemptive priorities and a negligible preemption overhead, class 1 txns are processed without being affected by class 2 txns. With non-preemptive priorities $W_1 = W_0 / (1 - \rho_1)$ and $W_2 = W_0 / ((1 - \rho_1)(1 - \rho))$, where $\rho_j = \lambda_j\overline{x}_j$, $j = 1{,}2$ and $W_0 = \frac{1}{2}\sum_{j=1}^{2} \lambda_j\overline{x_j^2}$ is the remaining processing time of txns at the CPU [4]. Note that W_1 is only affected by ρ_1, but not $\rho = \rho_1 + \rho_2$. Kleinrock's *conservation law* states that the improvement in waiting time of one txn class is at the cost of the other, so that the weighted sum of waiting times remains constant: $\sum_{j=1}^{2} \rho_j W_j = W_0/(1-\rho)$. Preemptive (resp. non-preemptive) priorities are applicable to CPU (resp. disk) scheduling.

Queueing Network Models – QNMs

In *open QNMs* there are txn arrivals and departures, while in a *closed QNM* a completed txn is immediately replaced by a new txn, so that the number of txns remains fixed at N. Txns are processed at $K > 1$ nodes, where each node is a single or multiserver queueing system. The QNM of a computer system may be organized as the *Central Server Model (CSM)*, with the CPU the central server and the disks as peripheral servers. Txns arrive at the CPU (node 1), access one of the $K-1$ disks with probability p_k, $2 \le k \le K$, return to the CPU for additional processing, and leave the system after CPU processing with probability p_1, so that $\sum_{k=1}^{K} p_k = 1$. The transition probability matrix yields the mean number of visits to the nodes: i.e., $v_1 = 1/p_1$ and $v_k = p_k/p_1, 2 \le k \le K$. QNMs satisfying the BCMP theorem allow four types of nodes, most notably exponential service times at single- or multi-server queues with FCFS scheduling and nodes with an infinite number of servers [4,5]. The steady-state probability $Pr[n_1,n_2,...,n_K]$ in such QNMs can be expressed in product-form and their performance metrics computed efficiently. Approximation techniques or simulation can be applied otherwise.

The mean residence times at the nodes of an open product-form QNM can be obtained separately. With an external arrival rate Λ, the arrival rate to node k is $\lambda_k = v_k \Lambda$, its utilization $\rho_k = \lambda_k \overline{x}_k / m_k$, where $\overline{x}_k$ is its mean service time and m_k is the number of its servers. The service demand or the total txn processing time at a node k is given as $D_k = v_k \overline{x}_k$, $1 \le k \le K$, so that alternatively $\rho_k = \Lambda D_k$. The mean txn residence time at node k is $r_k = \overline{x}_k/(1-\rho_k^{m_k})$, $1 \le k \le K$ with $m_k \le 2$. The mean txn response time is $R = \sum_{j=1}^{K} v_k r_k$ and the mean number of txns at the system following Little's result is the product of the arrival rate and the mean response time: $\overline{N} = \Lambda R$ [4,5]. $\overline{N}$ can be used to estimate the degree of txn concurrency or *Multiprogramming Level (MPL)* and the memory size requirements for TP.

Closed QNMs can be analyzed via the convolution or the *mean value analysis – MVA* algorithm [4,5]. The MVA *arrival theorem* states that in a closed QNM with n txns the mean queue-length at node k at an arrival instant, is that of a closed system with $n-1$ txns, i.e., $A_k(n) = Q_k(n-1)$ [5]. The mean queue-length includes the request at the server, whose mean residual service time is the same as its service time due to the memoryless property of the exponential distribution: $\overline{x'}_k = \overline{x^2}_k/(2\overline{x}_k) = 2(\overline{x}_k)^2/(2\overline{x}_k) = \overline{x}_k$ [4]. The mean residence time at node k for v_k visits is $R_k(n) = v_k r_k(n) = v_k \overline{x}_k [1 + Q_k(n-1)] = D_k[1 + Q_k(n-1)]$.

MVA Algorithm for Closed QNM with Single Servers.

```
Input parameters: N (MPL), K (number of nodes), $D_k$, $1 \le k \le K$ (mean service demands).
for $k = 1$ to $K$ do $Q_k(0) = 0$ (initialize queuelengths at $n = 0$)
for $n = 1$ to $N$ do {(vary the number of txns)
for $k = 1$ to $K$ do {(vary the node index)
        if delay server $R_k(n) = D_k$ (no queueing at "infinite servers")
        if single server $R_k(n) = D_k[1 + Q_k(n-1)]$}
$R(N) = \sum_{k=1}^{K} R_k(n)$ (txn residence time)
$X(n) = n/R(n)$ (txn throughput using Little's result)
for $k = 1$ to $K$ do $Q_k(n) = X(n)R_k(n)$ (mean queue-length at the nodes, Little's result) }
```

The convolution algorithm applied to single-server queues computes $G(0 : N)$, after the initialization $G(0) = 1$ and $G(n) = 0$, $n \neq 0$, as: $G(n) = G(n) + D_k G(n-1)$, $1 \le n \le N$, $1 \le k \le K$, so that $X(N) = G(N-1)/G(N)$. Compared to MVA the order of the iterations on n and k are reversed. Given that $\sum_{k=1}^{K} D_k$ =constant, the throughput of a closed QNM is maximized when the service demands at all the nodes are equal. Due to symmetry $Q_k(N) = (N-1)/K$, so that $R(N) = KD(1 + (N-1)/K) = (N + K - 1)D$. Next, consider Poisson arrivals to a QNM with a maximum degree of concurrency M due to limited memory size. For $n < M$ arriving txns are activated immediately and otherwise enqueued at a FCFS memory queue. A hierarchical solution method is applicable in this case [5]. The lower level model yields the throughput characteristic of the closed QNM: $X(n)$, $1 \le n \le M$. Additional txns waiting in the memory queue do not contribute to throughput, so that $X(n) = X(M)$, $n > M$. A birth-death model is used at the higher level, with the state n denoting the number of txns at the system. The arrival rate at all states is Λ and the completion rate at state n is $X(n)$, $n \ge 1$. The probability that there are n txns in the system is: $P_n = \Lambda P_{n-1}/X(n)$, $n \ge 1$. The normalization condition $\sum_{n \ge 0} P_n = 1$ yields P_0 and hence $P_n, \forall n > 0$. The mean number of txns at the computer system, mean number of txns at the memory queue, and the mean number of activated txns is: $\overline{N} = \sum_{n>0} nP_n$, $\overline{N}_B = \sum_{n>M}(n-M)P_n$, and $\overline{N}_A = \sum_{n>0} min(n,M)P_n$. The mean txn response time, mean waiting time at the memory queue, and the mean residence time at the computer system are, respectively: $R = \overline{N}/\Lambda$, $W_M = \overline{N}_B/\Lambda$, and $R_A = \overline{N}_A/\Lambda$. $\overline{N} = \overline{N}_B + \overline{N}_A$, so $R = W_M + R_A$. The throughput characteristic $X(n)$, $n \geq 1$ vs. n (without an MPL constraint) is a convex function, which reaches an asymptotic value $min\{m_k/D_k, \forall k\}$. The node with the smallest throughput is the bottleneck resource, since it determines the maximum system throughput Λ_{max}. The throughput may drop due to thrashing in an overloaded virtual memory system [5] or due to excessive lock contention with the standard locking method.

Buffer Miss Rate (BMR)

BMR is applicable to the various levels of the memory hierarchy. Misses at the database buffer and disk controller cache result in disk accesses, which significantly degrade the performance of TP systems. BMR is affected by the buffer size, the replacement policy, and the workload. Analyses of replacement policies, such as FIFO, LRU, and CLOCK, using Markov chain modeling are based on the *Independent Reference Model (IRM)*. (http://www.informatik.uni-trier.de/~ley/db/dbimpl/buffer.html.) Data streams can also be characterized by stack depth and the hierarchical reuse or fractal models. Trace-driven simulation is a straightforward technique for evaluating the performance of buffer management policies, which can be used for calibrating empirical formulas. A recent top-down analysis derives an equation for the number of page faults (n) vs. memory size (M) [9]. M_0 and M^* are the min and max M that are sufficient and necessary. n^* is the number of cold memory misses, i.e., misses starting with an empty buffer. The parameters vary with program behavior and replacement policy.

$$n = \frac{1}{2}[H + \sqrt{H^2 - 4}](n^* + n_0) - n_0,$$
$$H = 1 + \frac{M^* - M_0}{M - M_0}, M \leq M^*.$$

Performance Analysis of Storage Systems

RAID performance is improved by striping and caching. Striping eliminates disk access skew by partitioning large datasets into fixed size *stripe units (SUs)*, which are allocated in round-robin manner across all disks [1]. The RAID controller cache complementing the database buffer in main memory is partially *nonvolatile storage (NVS)* and is used for *fast writes*, which are as reliable as writing to disk. Disk writes can be deferred and processed at a lower priority than reads, since read response times affect txn response time. Dirty data blocks in NVS are possibly overwritten several times, before they are destaged or written to disk. This eliminates unnecessary disk accesses. Applying disk scheduling to batched destaging of multiple blocks leads to higher disk access efficiency. Both effects can be captured by trace analysis [15].

RAID1 or disk mirroring, which predates the RAID classification, replicates data on two disks to achieve fault-tolerance. A data block can be read from either disk, but both copies should be eventually updated. Disk bandwidth can be saved by writing modified blocks to NVS first. The doubling of data access bandwidth is beneficial in view of increasing disk capacities, but an even further improvement in bandwidth can be attained by judicious routing of requests. When one of mirrored disks fails the read load on the surviving disk is doubled, but a smaller load increase can be attained with *interleaved declustering*, which replicates the contents of one disk on the other $n - 1$ disks of the cluster, so that the read load increase is $n/(n - 1)$ [13].

Erasure coding in RAID5 and RAID6 uses one and two check disks to protect against single and double disk failures, respectively [1]. RAID5 uses parity coding, so that the check SU is the *exclusive-OR (XOR)* of the remaining SUs in the same stripe or row. RAID6 uses Reed-Solomon or specialized parity codes to protect two disks out of N. The updating of a data block requires the corresponding check block(s) to be computed and updated. If the old copies of data and check blocks are not cached, four (resp. six) disk accesses are required for RAID5 (resp. RAID6), this is therefore referred to as the *small write penalty* [1]. Check SUs are placed in repeating left-to-right diagonals to balance the load for updating the check blocks. When one of the N disks fails, RAID5 continues its operation in degraded mode, by reconstructing missing blocks of the broken disk on demand as follows. The controller issues a fork-join request to the $N - 1$ corresponding blocks on surviving disks and these blocks are XORed to reconstruct the missing block. The read load on surviving disks is doubled, but the load increase for write requests is smaller. The *clustered RAID* paradigm reduces the disk load in degraded mode by using a parity group size $G < N$, so that the increase in read load is $\alpha = (G-1)/(N-1) < 1$ [1]. The RAID5 rebuild process reconstructs the contents of a failed disk track-by-track on a spare disk, by reading successive tracks from surviving disks, XORing all corresponding tracks to reconstruct a missing track, and writing it on the spare disk [15]. Distributed sparing distributes the spare areas among $N + 1$ disks. so that the bandwidth of the spare disk is not wasted [15]. The *vacationing server method (VSM)* starts reading tracks from disks when they become idle, so the rebuild process does not affect the disk load. No new tracks are read after there is an external arrival, but the reading of the current track is completed. The increase in mean disk waiting time over M/G/1 is the mean residual time to read a track: $W_{VSM} = W_{M/G/1} + \overline{x'}_{track}$ [4]. RAID6 deals with the possibility of unsuccessful rebuild, when unreadable sectors or *latent sector failures – LSFs* are encountered during the rebuild process or to cope with rare two disk failures.

Mean disk response time with FCFS disk scheduling can be obtained using the M/G/1 queuing model [4]. Random disk accesses incur a seek, latency or rotational delay (half a rotation time for small blocks), and transfer time (negligible for small blocks). One has $x_{disk} = x_{seek} + x_{latency} + x_{xfer}$. The moments of disk access time can be obtained under various assumptions, e.g., all disk cylinders are accessed uniformly. A significant reduction in disk response time is possible via the *Shortest Access Time First (SATF)* policy [14]. Given n random disk requests, the disk access time drops as $n^{1/5}$, e.g., $n = 32$ halves the access time with respect to FCFS.

Performance Analysis of Concurrency Control Methods

Performance degradation due to CC is in the form of txn blocking and restarts. The *standard locking* method is based on the strict *two-phase locking (2PL)* method [3], i.e., locks are requested on demand and held to completion, at which point the txn commits, releasing all of its locks. Locks are also released when a txn is aborted to resolve deadlocks or other reasons [3]. Restart-oriented locking methods and *optimistic CC (OCC)* methods are also discussed.

Standard Locking

Consider an abstract model of standard locking [12]. There are M txns in a closed system. A txn of size k has $k + 1$ steps with mean duration s, so that the mean txn residence time with no lock contention is: $r_k = (k + 1)s$. A lock is requested at the end of the first k steps and all locks are released at the end of the k + 1st step, i.e., strict 2PL, so that lock requests are uniformly distributed over the lifetime of a txn. The mean number of locks held by a txn is the ratio of the time–space of the number of locks held by a txn and r_k, which is an approximation to R_k at lower levels of lock contention, which yields: $\overline{L}_k \approx k/2$. Assuming that all lock requests are exclusive and are uniformly distributed over the D objects in the database, the probability of lock conflict is: $P_c(k) \approx (M-1)\overline{L}_k/D$. When the fraction of shared lock requests is f_S, the analysis can proceed as if all the lock requests are exclusive with an effective database size $D_{eff} = D/(1 - f_S^2)$ [8]. The probability that a txn encounters a lock conflict for $P_c(k) \ll 1$ can be used to obtain the probability of a two-way deadlock [3]:

$$P_w(k) = 1 - (1 - P_c(k))^k \approx kP_c(k) \approx \frac{(M-1)k^2}{2D}.$$

$$\begin{aligned} P_{D(2)}(k) &= \frac{1}{M-1} Pr[T_1 \rightarrow T_2] Pr[T_2 \rightarrow T_1] \\ &= \frac{(P_w(k))^2}{M-1} \approx \frac{(M-1)k^4}{4D^2}. \end{aligned}$$

A two-way deadlock can only occur if txn T_A requests a lock held by txn T_B, which is blocked because of its previous lock conflict with T_A. Multiway deadlocks are much less common and are ignored [3]. We obtain the mean blocking delay of a txn with respect to an active txn holding the requested lock ($W_k^{(1)}$) by noting that the probability of lock conflict increases with the number of locks held by the txn (j) and that there are ($k - j$) remaining steps each with mean duration ($s + u$) [10,11]:

$$\begin{aligned} W_k^{(1)} &\approx \sum_{j=1}^{k} \frac{2j}{k(k+1)} [(k-j)(s+u)] \\ &\quad + s' = \frac{k-1}{3}[r+u] + s'. \end{aligned}$$

$u = P_c(k)W_k$ is the mean blocking time per step, where W_k is the mean waiting time per lock conflict, and s' is the remaining processing time of a step. Since deadlocks are rare, the mean txn response time only takes into account txn blocking time due to lock conflicts: $R_k \approx (k + 1)s + kP_c(k)W_k$. The fraction of txn blocking time per lock conflict with an active txn is $A = W_k^{(1)}/R_k \approx 1/3$. A more accurate expression for two-way deadlocks as verified by simulation is: $P'_{D(2)}(k) \approx AP_{D(2)}(k) = P_{D(2)}(k)/3$ [16]. Next consider K txn classes in a closed system, so that a completed txn is replaced by a class k txn with frequency $f_k, \sum_{k=1}^{K} f_k = 1$. The mean response time over all txn classes is: $R = (K_1 + 1)s + K_1 P_c W$, where $K_i = \sum_{k \geq 0} k^i f_i$ is the ith moment of txn size. The mean number of txns in class k is $\overline{M}_k = MR_k f_k / R$ (the summation $\sum_{k=1}^{K}$ yields M on both sides). Given that $\overline{M}_k = M(k+1) f_k / (K_1 + 1) \approx M k f_k / K_1$, the mean number of locks held per txn is then:

$$\overline{L} = \frac{1}{M} \sum_{k=1}^{K} \overline{L}_k \overline{M}_k \approx \frac{1}{M} \sum_{k=1}^{K} \frac{k}{2} \overline{M}_k$$
$$\approx \frac{1}{2K_1} \sum_{k=1}^{K} k^2 f_k = \frac{K_2}{2K_1}.$$

Similarly to fixed size txns $P_c \approx (M-1)\overline{L}/D$ and the expression for $P_{D(2)}$ is given in [16,12]. Both P_c and $P_{D(2)}$ are affected by the variability of txn sizes, e.g., for a truncated geometric distribution there is a twofold and tenfold increase with respect to fixed txn sizes, respectively [16]. W_1 for variable txn sizes can be obtained as the expected value of $W_1(k)$ over all txn sizes. The normalized value for W_1 is [10,11]:

$$A = \frac{W_1}{R} \approx \frac{K_3 - K_1}{3K_1(K_2 + K_1)}.$$

Denote the fraction of blocked txns with β and ignore the difference in the number of locks held by active and blocked txns. The probability that a txn has a lock conflict with an active txn at level $i = 0$ (resp. has a lock conflict with a txn blocked at level $i = 1$) is $1 - \beta$ (resp. β) and the mean blocking time is W_1 (resp. $W_2 = 1.5 W_1$). The expression for W_2 relies on the fact that at the random instant when the lock conflict occurs the active txn in the waits-for-graph is halfway to its completion. Generally, the probability that a txn is blocked at level i is: $P_b(i) \approx \beta^{i-1}, i > 1$ and $P_b(1) = 1 - \beta - \beta^2 - \beta^3 \ldots$. The mean waiting time at level $i > 1$ is $W_i \approx (i - 0.5) W_1$, which is motivated by the expression for W_2. The mean blocking time is a weighted sum of delays incurred by txns blocked at different levels:

$$W = \sum_{i \geq 1} P_b(i) W_i = W_1 [1 - \sum_{i \geq 1} \beta^i + \sum_{i > 1} (i - 0.5) \beta^{i-1}].$$

Let $\overline{M}_A$ and $\overline{M}_B$ denote the mean number of active and blocked txns, so that $\overline{M}_A + \overline{M}_B = M$. The fraction of blocked txns is $\beta = \overline{M}_B / M$. Dividing $\overline{M}_B$ and M by the system throughput yields $\beta = K_1 P_c W/R$. Multiplying both sides of the above equation by $K_1 P_c / R$ yields β on the left hand side and $\alpha = K_1 P_c W_1 / R = AK_1 P_c$ on the right hand side. Note that α is the product of the mean number of lock conflicts per txn and the normalized waiting time with respect to active txns. It follows: $\beta = \alpha(1 + 0.5\beta + 1.5\beta^2 + 2.5\beta^3 + \ldots)$. A closed-form expression for β can be obtained by noting that $\beta < 1$ and assuming that the series is infinite:

$$f(\beta) = \beta^3 - (1.5\alpha + 2)\beta^2 + (1.5\alpha + 1)\beta - \alpha = 0.$$

Plotting $f(\beta)$ vs. β it is observed that the cubic equation has three roots $0 \leq \beta_1 < \beta_2 \leq 1$ and $\beta_3 \geq 1$ for $\alpha \leq \alpha^* = 0.226$, which makes its discriminant $f(\alpha) = \alpha^3 + \frac{4}{5}\alpha^2 + \frac{28}{15}\alpha - \frac{64}{135} < 0$ (http://en.wikipedia.org/wiki/Cubic_equation.), and a single root $\beta_3 > 1$ for $\alpha > \alpha^*$. *The single parameter α determines the level of lock contention for the standard locking and its critical value α^* determines the onset of thrashing*. The smallest root β_1 for $\alpha \leq \alpha^*$ determines $\overline{M}_A = M(1 - \beta_1)$, which increases with M, but drops after a maximum value is achieved. This is because of a snowball effect that blocked txns cause further txn blocking. Given the throughput characteristic $X(n)$, $n \geq 1$ with no lock contention, the throughput of a TP system when $\overline{M}_A$ is non-integer is: $X(\overline{M}_A) = (\lceil \overline{M}_A \rceil - \overline{M}_A) \quad X(\lfloor \overline{M}_A \rfloor) + (\overline{M}_A - \lfloor \overline{M}_A \rfloor) X(\lceil \overline{M}_A \rceil)$. The analysis for txns with variable step durations uses iteration to estimate ρ, which is the ratio of the number locks held by blocked and active txns. It is shown in [12] that the critical value of ρ matches the *conflict ratio* parameter in [17], which is the ratio of the number of locks held by all and active txns.

Restart-Oriented Locking Methods

Performance degradation in standard locking is mainly caused by txn blocking. Txn aborts and restarts to resolve deadlocks have little effect on performance, because deadlocks are rare. Restart-oriented locking methods follow the strict 2PL paradigm, but txns encountering lock conflicts are restarted to increase the degree of txn concurrency, so that a higher txn throughput can be attained. This is beneficial for high lock contention TP systems with excess processing capacity, but in hardware resource bound systems this may result in performance degradation with respect to standard locking. Since active txns are not guaranteed to complete successfully and commit, txn throughput cannot be expressed as $X(\overline{M}_A)$, as in the case of standard locking. The *Running Priority* (RP) method aborts txn T_B in the waits-for-graph $T_A \rightarrow T_B \rightarrow T_C$, when T_A has a lock conflict with T_B [2]. Similarly T_B which is already blocking T_A is aborted when it is blocked requesting a lock held by T_C. In effect RP increases the degree of txn concurrency even if the restart of an aborted txn is delayed. The no-waiting [8] and cautious waiting methods abort and restart a txn encountering a lock conflict with an active and blocked txn, respectively. There is no benefit, since the restarted txn will encounter the same lock conflict as before. The *Wait Depth Limited* (WDL) method measures txn progress by its length (L), which in [2] is approximated by the number of locks that it holds in a lock contention bound system, while attained CPU time would be a concern otherwise. WDL differs from RP in that it attempts to minimize wasted processing by not restarting txns holding many locks or txns nearing completion. This requires a priori knowledge of txn's locking requirements. For example, if T_A which is blocking another txn has a lock conflict with T_B, then if $L(T_A) < L(T_B)$ then abort T_A, else abort T_B [2]. The superior performance of WDL over RP and especially standard locking has been shown via random-number-driven (Monte-Carlo) simulation in [2], but also trace-driven simulation in [17]. RP and WDL attempt to attain *essential blocking*, i.e., allowing a txn to be blocked only by active txns and not requesting a lock until it is required. Immediately restarting an aborted txn may result in *cyclic restarts* or *livelocks*, which should be prevented to reduce wasted processing and to ensure timely txn completion. *Restart waiting* delays the restart of an aborted txn until *all* conflicting txns are completed [2]. Conflict avoidance delays with random duration are a less reliable method to prevent cyclic restarts.

Restart-oriented methods are analyzed using a Markov chain model in [11], while flow-diagrams are utilized in the analysis of the no-waiting method in [8]. Active (resp. blocked) states correspond to S_{2j}, $0 \leq j \leq k$ (resp. S_{2j+1}, $0 \leq j \leq k-1$). For example, in the case of the no-waiting policy there is a transition forward when there is no lock conflict: $S_{2j} \rightarrow S_{2j+2}$, $0 \leq j \leq k-1$, otherwise the txn is aborted: $S_{2j} \rightarrow S_0$. The state equilibrium equations for the Markov chain yield the steady-state probabilities π_i, $0 \leq i \leq 2k$. The mean number of visits v_i to S_i can be similarly obtained by noting that $v_{2k} = 1$, which is the state at which the txn commits. Given that h_i denotes the mean holding time at S_i, then $\pi_i = v_i h_i / \sum_{j=0}^{2k} v_j h_j$, $0 \leq i \leq 2k$. The mean number of txn restarts is given by $v_0 - 1$. A shortcoming of the analytic approach is that requested locks are implicitly resampled.

Optimistic Concurrency Control

A txn starting its execution first copies all objects required for its processing into its private workspace [6]. To ensure serializability after completing its processing, the txn undergoes validation to ascertain that none of the copied objects was modified by another txn after it was read. If validation is successful the txn commits and otherwise it is immediately restarted. Given that the k objects updated by txns are uniformly distributed over a database of size D, the probability of data conflict between two txns of size k is: $\psi = 1 - (1 - k/D)^k \approx k^2/D$. The completion rate of successfully validated txns is $p\mu s(M)$, where p is the probability of successful validation, μ is the completion rate of txns, and $s(M)$ takes into account the hardware resource contention due to multiprogramming. A txn with processing time x will require $xM/s(M)$ time units to complete. It observes the commits of other txns as a Poisson process with rate $\lambda = (1 - 1/M)p\mu s(M)$. Txns encountering data conflict with rate $\gamma = \lambda\psi$, fail their validation with probability $q = 1 - e^{-\gamma xM/s(M)}$. The number of txn executions in the system follows a geometric distribution $P_j = q(1-q)^{j-1}$, $j \geq 1$ with a mean $\overline{J} = 1/q = e^{\gamma xM/s(M)}$. The system efficiency p can be expressed as the ratio of the mean execution time of a txn without and with data contention, i.e., with the possibility of restarts due to failed validation:

$$p = \frac{E[xM/s(M)]}{E[(xM/s(M))e^{\gamma xM/s(M)}]} = \frac{\int_0^\infty xe^{-\mu x}dx}{\int_0^\infty xe^{(\gamma M/s(M)-\mu)x}dx}$$
$$= [1-(M-1)\psi p)]^2.$$

The equation yields one acceptable root $p = [1+2(M-1)\psi - \sqrt{1+4(M-1)\psi}]/[2(M-1)^2\psi^2] < 1$, provided $\gamma M/s(M) < \mu$ and the integral in the denominator converges. The txn execution time is not resampled in this analysis, because resampling processing times results in a mix of completed txns which is shorter than original. Shorter txns have a higher probability of successful validation and resampling results in overestimating performance. This analysis in [6] is that of the OCC *silent* [7] or *die* method [2], which is inefficient in that a conflicted txn is allowed to execute to the end, at which point it is restarted. The analysis in [6] also considers *static* data access, i.e., all objects are accessed at the beginning of execution, while *dynamic* or *on-demand* data access is more realistic. The OCC *broadcast* or *kill* method aborts a txn as soon as a conflict is detected. The analysis of the static/silent method in [6] is extended to the other three cases in [7]. The distribution of txn processing time affects performance and in a system with multiple txn sizes the wasted processing is mainly due to lengthy txns [7]. This is due the *quadratic effect* that the probability that a txn encounters a data conflict increases with the number of objects it accesses (k) and its processing time, which is also proportional to k [2]. While the optimistic die method seems to be less efficient than the optimistic kill method, this is not the case when due to buffer misses txns make disk accesses. It is advantageous then to allow a conflicted txn to run to its completion to prime the database buffer with all the objects required for its re-execution in a second processing phase. In *two-phase processing* txns in the second execution phase have a much shorter processing time, since disk accesses are not required, provided *access invariance* prevails, i.e., a restarted txn accesses the same set of objects as it did in its first execution phase [2]. The second execution phase may use the optimistic kill method or even lock preclaiming which ensures that the txn will not be restarted. In general we have multiphase processing methods with different CC methods with increasing strengths, e.g., locking vs. OCC. WDL and RP restart-oriented locking methods outperform two-phase methods [2]. The analysis in [7] can be extended to take into account the variability in txn processing times and the use of different CC methods across txn phases.

Conclusion

There is a need for methodologies combining analytic models, simulation, and trace analysis to explore the effectiveness of new computer organizations for TP, since it is expensive to build realistic prototypes for experimentation. An abstract model of a standard locking system is analyzed here to provide an understanding of the thrashing phenomenon in standard locking. It is shown that a single parameter α determines the level of lock contention with a critical value, which determines the onset of thrashing. Restart-oriented locking methods selectively abort txns to increase the degree of txn concurrency and reduce the level of lock contention, by disallowing txns to wait for the completion of already blocked txns. Two-phase processing methods reduce the data contention level in TP systems with access invariance, by lowering the holding time of database objects. Both methods require excess processing capacity to cope with the additional processing when txns are restarted.

Cross-references

- ▶ Optimistic Concurrency Control
- ▶ RAID
- ▶ Strict 2PL
- ▶ Two-Phase Locking

Recommended Reading

1. Chen P.M., Lee E.K., Gibson G.A., Katz R.H., and Patterson D.A. RAID: High-performance, reliable secondary storage. ACM Comput. Surv., 26(2):145–185, 1994.
2. Franaszek P., Robinson J.T., and Thomasian A. Concurrency control for high contention environments. ACM Trans. Database Syst., 17(2):304–345, 1992.
3. Gray J.N. and Reuter A. Transaction Processing: Concepts and Facilities. Morgan Kauffmann, Los Altos, CA, 1992.
4. Kleinrock L. Queueing Systems: Vol. 1/2: Theory/Computer Applications. Wiley, New York, 1975/1976.
5. Lazowska E.D., Zahorjan J., Graham G.S., and Sevcik K.C. Quantitative System Performance. Prentice-Hall, Englewood Cliffs, NJ, 1984.
6. Morris R.J.T. and Wong W.S. Performance analysis of locking and optimistic concurrency control algorithms. Perf. Eval., 5(2):105–118, 1985.
7. Ryu I.K. and Thomasian A. Performance evaluation of centralized databases with optimistic concurrency control. Perf. Eval., 7(3):195–211, 1987.
8. Tay Y.C. Locking Performance in Centralized Databases. Academic Press, New York, 1987.
9. Tay Y.C. and Zou M. A page fault equation for modeling the effect of memory size. Perf. Eval., 63:99–130, 2006.
10. Thomasian A. Concurrency control: Methods, performance, and analysis. ACM Comput. Surv., 30(1):70–119, 1998.
11. Thomasian A. Performance analysis of locking policies with limited wait-depth. Perf. Eval., 33(1):1–21, 1998.
12. Thomasian A. Two-phase locking and its thrashing behavior. ACM Trans. Database Syst., 18(4):579–625, 1993.
13. Thomasian A. and Blaum M. Mirrored disk reliability and performance. IEEE Trans. Comput., 55(12):1640–1644, 2006.
14. Thomasian A., Fu G., and Han C. Performance evaluation of two-disk failure tolerant arrays. IEEE Trans. Comput., 56(6):799–814, 2007.

15. Thomasian A. and Menon J. RAID5 performance with distributed sparing. IEEE Trans. Parallel Distr. Syst., 8(6):640–657, 1997.
16. Thomasian A. and Ryu I.K. Performance analysis of two-phase locking. IEEE Trans. Software Eng., 17(5):386–402, 1991.
17. Weikum G., Hasse C., Moenkeberg A., and Zabback P. The COMFORT automatic tuning project. Inf. Syst., 19(5):381–432, 1994.

Performance Benchmark

► Application Benchmark

Performance Measures

► Search Engine Metrics

Performance Metrics

► Evaluation Metrics for Structured Text Retrieval

Performance Monitoring Tools

Philippe Bonnet[1], Dennis Shasha[2]
[1]University of Copenhagen, Copenhagen, Denmark
[2]New York University, New York, NY, USA

Definition

Performance monitoring tools denote the utilities and programs that give access to database server internals.

Historical Background

Relational database systems have had to prove their performance from the outset [1]. Tools to measure their performance have therefore been present in early versions of most major systems.

Foundations

Performance Monitoring tools are useful in finding out how queries are being serviced, and how the underlying resources are being used. In this entry, the characteristics of the most relevant types of tools are described.

Event Monitor

Event monitors capture the aggregate resources associated to a given event (e.g., query execution, deadlock, session) and report the collected data when the event completes. Event monitors should have low overhead as the collected data is usually accumulated in performance counters that are updated as a side effect of the operations monitored (e.g., CPU usage, IO issued, locks collected during query execution) and only accessed once the event has completed.

Event monitors are useful to identify the queries that consume most resources and should require the attention of the database tuner.

Query Plan Explainer

Query plan explainers display the execution plan chosen by the query optimizer for a given query. Explainers represent a query plan, either in textual form or graphically, as a tree whose leaves are base tables and internal nodes are access methods and relational operators. The nodes are possibly annotated with relevant properties (e.g., cardinality, estimated cost).

Query plan explainers are useful to check that the optimizer relies on updated statistics to generate a query execution plan, to check that indexes are used as intended, or to find out whether costly operators are inserted against the programmer's better judgment.

Profiler

There are two types of profilers:

1. Time-based profilers address the question: *how is the time spent*? A time-based profiler logs the time spent in the different components of the database system and presents the time spent processing as well as the time spent waiting for resources.
2. Counter-based profilers address the question: *how are resources used*? A counter-based profiler uses counters to monitor the resources used during execution and presents either database-wide or system-wide aggregate counters.

Profilers might incur high running cost because of the overhead of logging a potentially large number of actions. Profilers are most useful to understand the behavior of critical queries.

Key Applications

Database management system developers need performance monitoring tools to make sure that their system is performing well. Database administrators need performance monitoring tools to make sure that their database instance is performing as well as possible.

Cross-references

▶ Query Optimization

Recommended Reading

1. McJones P. The 1995 SQL reunion: people, projects, and politics. Technical Report: SRC–TN–1997–018.
2. Millsap C. and Holt J. Optimizing Oracle Performance. O'Reilly, Sebastopol, CA, 2003.
3. Shasha D. and Bonnet P. Database Tuning: Principles, Experiments and Troubleshooting Techniques. Morgan Kaufmann, San Francisco, CA, 2002.

Period-Stamped Temporal Models

NIKOS A. LORENTZOS
Agricultural University of Athens, Athens, Greece

Synonyms

Interval-based temporal models

Definition

A period-stamped temporal model is a *temporal data model* for the management of data in which time has the form of a *time period*.

Historical Background

There are applications that require the recording and management not only of data but also of the time during which this data is valid (*valid time*). A typical example is the data maintained by pension and life insurance organizations in order to determine the benefits for which a person qualifies. Similarly, such organizations have to record their financial obligations at various times in the future.

There are also sensitive applications, in which it is important to record not only the data but also the time at which this data was either recorded in the database or deleted or updated (*transaction time*). This time is recorded automatically by the Database Management System (DBMS) and it cannot be modified by the user.

Finally, there are applications in which both data, valid time, and transaction time need be recorded.

A data model for the management of data and also of either valid time or transaction time is a *temporal data model*. If time has the form of a *time period* the model is a *period-stamped data model*.

In the case of the relational model, data and time are recorded in a relation. A relation to record data and valid time is a *valid time* relation. A relation to record data and transaction time is a *transaction time* relation. Finally, a relation to record data, valid time and transaction time is a *bitemporal relation*.

In spite of the interest for the management of temporal data, there are many practical problems, which cannot be supported directly by the use of a conventional DBMS. As a consequence, much programming is required in order to support such applications. Hence, the satisfaction of actual user requirements necessitated the definition of a period-stamped temporal model. The bulk of research, on the definition of such a model, appeared in the 1980's, at a time when computers became powerful enough to process large volumes of data.

Foundations

Problems related to the management of temporal data are depicted next. The illustration is restricted to the relational model, in particular to the management of valid time relations. Note, however, that relevant problems can be identified in the management of transaction time relations whereas the management of bitermporal relations is much more complicated.

Examples Illustrating Inadequacy of Conventional Models to Support Time Period

Figure 1 shows SALARY, a valid time relation that records salary histories. The notation "d_number," for values recorded in attributes Begin and End,

Salary

Name	Amount	Begin	End
Alex	100	d100	d299
Alex	150	d300	d499
Alex	150	d800	d999
John	100	d200	d899

Period-Stamped Temporal Models. Figure 1. Example of a valid time relation.

represents a point in time, which is of a date-time data type. For example, d100 could represent "January 1, 2007." The interpretation of the first tuple is that Alex earned 100 on each of the dates in the *time period* [d100, d299], i.e., on each of the dates d100, d101,..., d299. Then the amount earned was 150 for each of the dates in [d300, d499].

Contrary to the simple representation of a valid time relation, a conventional DBMS lacks the functionality required for the management of such relations. This is illustrated below by a number of examples.

Data insertion: If the tuple (Alex, 150, d400, d799) is inserted into SALARY, it will be recorded in addition to the other tuples. In this case, however, SALARY will contain duplicate data for Alex's payment on the dates d400, d401,...,d499, since this data is already recorded in the second tuple of the relation. Moreover, the query, to return *the time at which Alex's salary became 150*, if not formulated carefully, will return three dates, d300, d400 and d800 whereas the correct answer should be d300. To avoid such problems, it would be appropriate for the newly inserted tuple to be combined with the second and third tuple for Alex into a single one, (Alex, 150, d300, d999), i.e., tuples with identical data for attributes Name and Amount (*value equivalence*), which have either *overlapping* or *adjacent* time periods, to coalesce into a single time period (*temporal coalescing*).

Key: Declaring the primary key of SALARY, to be a multi-attribute key of Name and Amount, will not disallow the recording of the tuple (Alex, 200, d400, d799). As a result, SALARY will contain conflicting data, since its second tuple will be showing that Alex's payment is 150 for each of the dates d400, d401,...,d499 whereas, from the newly inserted tuple, it will also be shown that, for these dates, this payment is 200. Hence, in the case of a conventional DBMS, special integrity constraints have to be defined for every relation like SALARY.

Data deletion: Assuming that John's payment for the dates d400, d401,...,d799 was recorded by mistake and has to be deleted, the last tuple of SALARY has to be replaced by two tuples, (John, 100, d200, d399) and (John, 100, d800, d899).

Data update: Assuming that John's payment for dates d400, d401,...,d799 has to be corrected to be 150, the last tuple of SALARY has to be replaced by three tuples, (John, 100, d200, d399), (John, 150, d400, d799) and (John, 100, d800, d899).

Data projection: The query "*for every employee, list the time during which he is paid*," requires a projection on attributes Name, Begin, End. This will generate four tuples. In practice, however, a *temporal coalescing* of the time periods recorded in SALARY is also required in order to obtain the correct result, consisting of only three tuples, (Alex, d100, d499), (Alex, d800, d999) and (John, d200, d899).

Since a conventional DBMS lacks the functionality illustrated above, special code has to be written to handle correctly all the cases described. This limitation gave rise to research on the definition of a temporal data model. Many of the research efforts led to proposals for the definition of various period-stamped data models.

Common Characteristics of Period-Stamped Data Models

The majority of researchers adopted the principle that a temporal data model should be a minimal extension of the conventional relational data model. The most common characteristics of these approaches are outlined below.

A period-stamped, valid time relation always has two types of attributes: The first type consists of one or more ordinary attributes of a conventional relation. These attributes are termed *explicit* by some authors. Valid time itself is not considered to be data, it is considered as being *orthogonal* to data. Hence, the second type consists of special attributes to record valid time. These attributes are often termed *implicit* and can be system-assigned by default. Some approaches consider one pair of such attributes, to record the Begin and End of valid time periods. The valid time recorded in them is usually assumed to represent a time period of the form [Begin, End], i.e., an interval closed on both sides. It is said that the data recorded in the explicit attributes is *stamped* by the time recorded in the implicit attributes.

According to the concepts discussed above, the explicit attributes of the relation in Fig. 1 are Name and Amount and the implicit attributes are Begin and End. The first tuple of the relation is (Alex, 100) and it is *stamped* by the time period [d100, d299].

Some approaches define a *time period* data type; therefore they consider only one implicit attribute. In such approaches, the equivalent scheme for the relation in Fig. 1 is SALARY (Name, Amount, Period).

Since valid time is not considered to be data, it cannot be part of the primary key of a relation. Hence, Name is designated as the primary key of SALARY, i.e., it matches the primary key of an ordinary relation.

In all the approaches, valid time is considered to be discrete. Hence, the time period [d100, d299] is interpreted as consisting of the dates d100, d101,...,d299. Various *temporal granularities* can be declared by the user, depending on the application.

Temporal Algebras, *Temporal Query Languages* or Temporal Calculi are proposed in various approaches. The functionality of the operations of the conventional relational model is revised accordingly, so as to overcome the problems illustrated earlier. Appropriate predicates are also defined, applicable to time periods, which can be used in selection operations. In some approaches additional operations are defined to capture temporal functionality that cannot be achieved by simply extending existing languages. In general, an implicit *temporal coalescing* takes place in all the temporal operations introduced.

Most of the proposed models in the literature support the valid time property over past, present and future time periods.

Desired Behavior of Period-Stamped Data Models

Given the above limitations, new problems have to be faced by the proposed period-stamped models. The most interesting of them are illustrated next by making use of relation SALARY in Fig. 1.

Data projection: The query "*list all the employees ever paid*" requires a projection of SALARY on Name. Such an operation normally yields a relation R with tuples (Alex), (John). Given however that R lacks implicit attributes, it gives rise to the question whether it is an appropriate response. To overcome this problem, in some approaches projection is defined to yield, after a *temporal coalescing*, the rows (Alex, d100, d499), (Alex, d800, d999) and (John, d200, d899). Notice however that, in this case, the result matches exactly that of another query, "*for every employee, list the time during which he is paid.*"

Stamp projection: Since time is not data, some special operation, to project only on time, may be necessary. For example, the answer to the query "*give the time during which at least one employee was paid*" should consist of one row, (d100, d999). Given however that this result lacks explicit attributes, it is necessary to determine whether it is appropriate to allow time stamp projection without explicit attributes. Similar questions arise in the case of projections of only one of the two implicit attributes.

Association of data in distinct relations: Such an association requires the use of the Cartesian product operation. For an illustration, consider also the relation in Fig. 2, which is used to record the history of employee assignments to departments. Then a Cartesian product of SALARY with ASSIGNMENT seems to yield a relation with two pairs of Begin and End time stamps, one pair from each relation. Clearly this is not a correct period-stamped relation. The same is also true for a join operation. As a consequence, some researchers avoid defining Cartesian product and restrict to the definition of a *temporal join* operation. This operation yields relation R, in Fig. 3, with a single pair of implicit attributes.

Literature Overview

The characteristics of individual approaches are outlined below. Unless otherwise specified, a point-stamped valid time relation, in the approach under discussion, is that of Fig. 1.

Jones and Mason [1] define LEGOL, a language for the management of period-stamped, valid time relations. It is an early, yet incomplete piece of work.

Ben-Zvi defines a model for the management of period-stamped, bitermporal relations ([14], Chap. 8). One more attribute is used, to record the time at which a tuple is deleted. Sets of time periods are also

Assignment

Name	Department	Begin	End
Alex	Shoe	d100	d199
Alex	Food	d300	d800
John	Food	d200	d899

Period-Stamped Temporal Models. Figure 2. Another valid time relation.

R

Name	Amount	Department	Begin	End
Alex	100	Shoe	d100	d199
Alex	150	Food	d300	d499
Alex	150	Food	d800	d800
John	100	Food	d200	d899

Period-Stamped Temporal Models. Figure 3. Result of a temporal join operation.

considered, consisting of mutually disjoint periods. Some SQL extension is provided, too. It is also an early and incomplete piece of work.

Snodgrass defines a QUEL extension, for the management of valid time, transaction time and bitemporal relations [9]. Data may alternatively be stamped by time points (*point-stamped temporal models*). Time may not be specified for the primary key. Formal semantics are provided. A tuple calculus and a relational algebra are defined in [14], Chap. 6.

Navathe and Ahmed propose an SQL extension for period-stamped valid time relations ([14], Chap. 4, [5,6]). In this approach, the primary key of the relation in Fig. 1 is <Name, Begin>. The definition of the primary key of a relation, in conjunction with the definition of a *Time Normal Form*, enables temporal coalescing. One variation of the select operation takes as argument a time period value. A *moving window* operation enables a temporary *split* of the time stamps of all the tuples into time periods of a fixed size, and the subsequent application of aggregate functions on the data that are associated with these fixed size periods.

Lorentzos and Johnson define a relational algebra for the management of either period-stamped or point-stamped valid time data [3,2]. It is also shown that there are practical considerations for having relations with more than one valid time. A period data type is later defined in [14], Chap. 3. A generic period data type is defined in [14], Chap. 3, and in [4], enabling the use of periods of numbers and strings. Such periods can be used for the management of non-valid time relations which, however, require a functionality identical with that of period-stamped, valid time relations. The operations of the conventional relational model are not revised but two new algebraic operations are defined. Time is treated as data. As such, it may participate in the primary key [4]. In [14], Chap. 3, and in [4], it is shown that there are applications in which a *temporal coalescing* should be disallowed. An SQL extension is also defined in [4].

Sarda defines a relational algebra in [7] and an SQL extension for valid time relations in [14], Chap. 5, and in [8]. A time period data type is also defined in [7]. Data may alternatively be stamped by time points. Time may not be specified for the primary key. The operations of the conventional relational model are not revised but, as reported in [14], if time is projected out, the result relation is not a correct valid time relation. Similarly, the result returned by the Cartesian product operation is considered not to be a correct period-stamped relation. Two additional relational algebra operations are also defined, functionally equivalent with those defined by Lorentzos.

TSQL2 is a consensus temporal SQL2 extension for the management of either period or point-stamped valid time relations, transaction time relations and bitermporal relations. It is complemented by the definition of a relational algebra.

All the previous approaches incorporate time at the tuple level (*tuple time-stamping*). Contrary to these, Tansel incorporates time at the attribute level (*attribute time-stamping*) [10]. Hence, to record the history of salaries and of assignment to departments, it suffices to consider a relation like that shown in Fig. 4, in place of the two distinct relations in Figs. 1 and 2. The relation consists of two tuples, one for Alex and another for John. Time periods are open for the End time. One exception is the case where the end point of a time period equals *now*, in which case the period is closed (see for example Fig. 4, the assignment of Alex to the Food department). Therefore, data valid in the future is not supported.

The approach considers four types of attributes, *atomic* (Name, in Fig. 4), *set valued* (e.g., to record data such as {Alex, Tom}), *triplet valued* (e.g., to record data such as <[d100, d300), 100>) and *set triplet valued* (Salary and Department, in Fig. 4). The operations of the relevant conventional model are revised accordingly. Four additional operations are defined, which enable transformations between relations with different types of attributes. A QUEL extension is defined in [11]. The approach in [10] is extended in [14], Chap. 7, and in [12], to a nested model, incorporating the well-known nest and unnest operations. A Calculus and Algebra are also defined. The approach in [12] is extended in [13], in order to support nested bitemporal relations, in which valid and transaction time are incorporated at the attribute level.

Employee

Name	Salary	Department
Alex	{<[d100, d300), 100> <[d300, d500), 150> <[d800, d1000), 150>}	{<[d100, d200), Shoe> <[d300, now], Food>}
John	{<[d200, d900), 100>}	{<[d200, d900), Food>}

Period-Stamped Temporal Models. Figure 4. A relation in Tansel's approach.

Key Applications

There are many applications that necessitate the management of period-stamped and, more generally, temporal data. Some examples are the following.

Period-stamped, valid time relations can be used by pension and life insurance organizations in order to determine the benefits a person qualifies for. Such relations are also needed for these organizations to record their financial obligations at various times in the future.

Similarly, period-stamped, transaction time relations can be used in sensitive applications, to enable tracing the content of the database and evaluate some decision taken in the past, with respect to the content of the database at that time.

Future Directions

Given the practical interest of period-stamped data, and given the many differences between different approaches, it will be useful to agree on and develop a standard model by an international organization, such as ISO.

Further research is necessary for the management of period-stamped geographical data and for the definition of a nested period-stamped data model.

Cross-references

▶ Point-stamped Temporal Models
▶ Supporting Transaction Time Databases
▶ Temporal Algebras
▶ Temporal Database
▶ Temporal data models
▶ Temporal Logical Models
▶ Temporal Object-Oriented Databases
▶ Temporal Query Languages

Recommended Reading

1. Jones S. and Mason P.S. Handling the time dimension in a database. In Proc. Int. Conf. Data Bases, 1980, pp. 65–83.
2. Lorentzos N.A. and Johnson R.G. Extending relational algebra to manipulate temporal data. Inf. Sys., 13(3):289–296, 1988.
3. Lorentzos N.A. and Johnson R.G. TRA a model for a temporal relational algebra. In Temporal Aspects in Information Systems, C. Rolland, F. Bodart, M. Leonard (eds.). North-Holland, pp. 203–215, 1988.
4. Lorentzos N.A. and Mitsopoulos Y.G. SQL extension for interval data. IEEE Trans. Knowl. Data Eng., 9(3):480–499, 1997.
5. Navathe S.B. and Ahmed R. A temporal relational model and a query language.Inf. Sci. Int. J., 47(2):147–175, 1989.
6. Navathe S.B. and Ahmed R. TSQL: a language interface for history databases. In Temporal Aspects in Information Systems, C. Rolland, F. Bodart, M. Leonard (eds.). North-Holland, pp. 109–122, 1988.
7. Sarda N.L. Algebra and query language for a historical data model. Computer J., 33(1):11–18, 1990.
8. Sarda N.L. Extensions to SQL for historical databases. IEEE Trans. Knowl. Data Eng., 2(2):220–230, 1990.
9. Snodgrass S. The temporal query language TQUEL. ACM Trans. Database Sys., 12(2):247–298, 1987.
10. Tansel A.U. Adding time dimension to relational model and extending relational algebra. Inf. Sys., 11(4):343–355, 1986.
11. Tansel A.U. A historical query language. Inf. Sci., 53(1–2):101–133, 1991.
12. Tansel A.U. Temporal relational data model. IEEE Trans. Knowl. Data Eng., 9(3):464–479, 1997.
13. Tansel A.U. Canan Eren Atay: Nested bitemporal relational algebra. In Proc. 21st Int. Symposium on Computer and Information Sciences, 2006, pp. 622–633.
14. Tansel A., Clifford J., Gadia J., Segev A., and Snodgrass R. (eds.). Temporal databases: theory, design and implementation. Benjamin/Cummings, 1993.

Persistence

▶ ACID Properties

Persistent Applications

▶ Application Recovery

Persistent Archives

▶ Digital Archives and Preservation

Personal Data

▶ Individually Identifiable Data

Personalized Interfaces

▶ Adaptive Interfaces

Personalized Search

▶ Personalized Web Search

Personalized Web

▶ Data Integration in Web Data Extraction System

Personalized Web Search

Ji-Rong Wen[1], Zhicheng Dou[2], Ruihua Song[1]
[1]Microsoft Research Asia, Beijing, China
[2]Nankai University, Tianjin, China

Synonyms

Personalized search

Definition

For a given query, a personalized Web search can provide different search results for different users or organize search results differently for each user, based upon their interests, preferences, and information needs. Personalized web search differs from generic web search, which returns identical research results to all users for identical queries, regardless of varied user interests and information needs.

Historical Background

Web search engines have made enormous contributions to the web and society. They make finding information on the web quick and easy. However, they are far from optimal. A major deficiency of generic search engines is that they follow the "one size fits all" model and are not adaptable to individual users. This is typically shown in cases such as these:

1. Different users have different backgrounds and interests. They may have completely different information needs and goals when providing exactly the same query. For example, a biologist may issue "mouse" to get information about rodents, while programmers may use the same query to find information about computer peripherals. When such a query is issued, generic search engines will return a list of documents on different topics. It takes time for a user to choose which information he/she really wants, and this makes the user feel less satisfied. Queries like "mouse" are usually called ambiguous queries. Statistics has shown that the vast majority of queries are short and ambiguous. Generic web search usually fails to provide optimal results for ambiguous queries.
2. Users are not static. User information needs may change over time. Indeed, users will have different needs at different times based on current circumstances. For example, a user may use "mouse" to find information about rodents when the user is viewing television news about a plague, but would want to find information about computer mouse products when purchasing a new computer. Generic search engines are unable to distinguish between such cases.

Personalized web search is considered a promising solution to address these problems, since it can provide different search results based upon the preferences and information needs of users. It exploits user information and search context in learning to which sense a query refers. Consider the query "mouse" mentioned above: Personalized web search can disambiguate the query by gathering the following user information:

1. The user is a computer programmer, not a biologist.
2. The user has just input a query "keyboard," but not "biology" or "genome." Before entering this query, the user had just viewed a web page with many words related to computer mouse, such as "computing," "input device," and "keyboard."

Foundations

User Profiling

To provide personalized search results to users, personalized web search maintains a user profile for each individual. A user profile stores approximations of user tastes, interests and preferences. It is generated and updated by exploiting user-related information. Such information may include:

1. Demographic and geographical information, including age, gender, education, language, country, address, interest areas, and other information;
2. Search history, including previous queries and clicked documents. User browsing behavior when

viewing a page, such as dwelling time, mouse click, mouse movement, scrolling, printing, and bookmarking, is another important element of user interest.

3. Other user documents, such as bookmarks, favorite web sites, visited pages, and emails. Teevan et al. [15] and Chirita et al. [1] demonstrate that external user data stored in a user client is useful to personalize individual search results.

User information can be specified by the user (explicitly collecting) or can be automatically learnt from a user's historical activities (implicitly collecting). As the vast majority of users are reluctant to provide any explicit feedback on search results and their interests, many works on personalized web search focus on how to automatically learn user preferences without involving any direct user efforts [6,8,9,10,13]. Collected user information is processed and organized as a user profile in a certain structure, depending on the need of personalization algorithm. This can be completed by creating vectors of URLs/domains, keywords, topic categories, tensors, or the like.

A user profile can usually aggregate a user's history information and represent the user's *long-term* interests (information needs). Some work has investigated whether such a long-term user profile is ineffective in some cases. Consider the second case that was described in the historical background section: a user will have different needs at different times based on circumstances. In such situations, personalization based on a user's long-term interests may not provide a satisfying performance, because similar results could be returned. Some work [10] has considered the use of a user's active context to represent *short-term* information needs. Search context is incorporated into the user profile, or is constructed as a separate short-term user model/profile and is used in helping infer a user's information needs.

Personalized Search Based on Content Analysis

Personalized web search can be achieved by checking content similarity between web pages and user profiles.

Some work has represented user interests with topical categories. User's topical interests are either explicitly specified by users themselves, or can be automatically learned by classifying implicit user data. Search results are filtered or re-ranked by checking the similarity of topics between search results and user profiles. In some work [2,8], a user profile is structured as a concept/topic hierarchy. User-issued queries and user-selected snippets/documents are categorized into concept hierarchies that are accumulated to generate a user profile. When the user issues a query, each returned snippet/document is also classified. The documents are re-ranked based upon how well the document categories match user interest profiles. Chirita et al. [2] use the ODP (Open Directory Project, http://www.dmoz.org/) hierarchy to implement personalized search. User favorite topics nodes are manually specified in the ODP hierarchy. Each document is categorized into one or several topic nodes in the same ODP hierarchy. The distances between the user topic nodes and the document topic nodes are then used to re-rank search results.

Some other work uses lists of keywords (bags of words) to represent user interests. In [13], a user profile is built as a vector of distinct terms and is constructed by aggregating past user click history. The cosine similarity between the user profile vector and the feature vector of returned web pages are used to re-rank results. Shen et al. [10] first use language modeling to mine immediate search contextual and implicit feedback information. The approach selects appropriate terms from related preceding queries and corresponding search results to expand the current query. In a query session, the viewed document summaries are used to immediately re-rank documents that have not yet been seen by the user. Teevan et al. [15] and Chirita et al. [1] exploit rich models of user interests, built from both search-related information, and other information about the user. This includes documents and emails the user has read and created. In [6], keywords are associated with categories and thus user profiles are represented by a hierarchical category tree based on keywords categories.

Personalized Web Search Based on Hyperlink Analysis

Most generic web search approaches rank importance of documents based on the linkage structure of the web. An intuitive approach of personalized web search is to adapt these algorithms to compute personalized importance of documents. A large group of these works focuses on personalized PageRank. PageRank, proposed by Page and Brin [7], is a popular link analysis algorithm used in web search. The fundamental motivation underlying PageRank is the recursive notion that important pages are those linked-to by many important pages. This recursive notion can be formalized by the "random surfer" model [7] on the directed web graph G. A directed edge $<p, q>$ exists in G if page p has a hyperlink to page q. Let $O(p)$ be the outdegree of web page p in G. $O(p)$ is equivalent to number of web pages that linked by page p. Let $\mathbf{A}$ be the matrix corresponding to the web graph G, where $A_{ij} = 1/O(j)$ if page j links to page i, and $A_{ij} = 0$ otherwise. In the random surfer model, when a surfer visits page p, he/she keeps clicking outlinks at random with probability (1-c), and jumps to a random web page with probability c. c is called teleportation constraint or damping factor. The PageRank of a page p is defined as the probability that the surfer visited page p. Iterative computation of PageRank is done as the following equation:

$$\boldsymbol{v}^{k+1} = (1 - c)\ \boldsymbol{A}\boldsymbol{v}^{k} + c\boldsymbol{u} \tag{1}$$

Here, $\boldsymbol{u}$ is defined as a preference vector, where $|\boldsymbol{u}| = 1$ and $u(i)$ denotes the amount of preference for page i when the surfer jumps to a random web page i. The global PageRank vector is computed when there is no particular preference on any pages, i.e., $\boldsymbol{u} = [1/n,...,1/n]^T$. By setting variant preference to web pages, a PageRank vector with personalized views of web page importance is generated. It recursively favors pages with high preference, and pages linked by high-preference page. This PageRank vector is called a *personalized PageRank vector* (*PPV*). To accomplish personalized web search, a personalized PageRank is computed for each user based upon the user's preference. For example, web pages in the user's bookmarks are set higher preferences in $\boldsymbol{u}$. Rankings of the user's search results can be biased according to the user's Personalized PageRank vector instead of the global PageRank.

Unfortunately, computing a PageRank vector usually requires multiple scans of the web graph [7], which makes it impossible to carry out online in response to a user query. Furthermore, when a large number of users employ a search engine, it is impossible to compute and store so many personalized PageRank vectors offline. Many later works [4,5] make efforts to reduce the computation and storage cost of personalized PageRank vectors. Jeh and Widom [5] support the concept that a user's preference set is a sub-set of a set of hub pages H, selected as those of greater interest for personalization. For each hub page p in H, setting the preference to 1 for page p and 0 for other pages, the corresponding personalized PageRank vector is called a basis hub vector. The authors decompose each basis hub vector in two parts: hub skeleton vector and partial vector. Hub skeleton vector represents common interrelationships between hub vectors, and is computed offline. Each partial vector for a hub page p represents the part of p's hub vector unique to itself. Partial vector can be computed at construction-time efficiently. Finally, a personalized PageRank vector can be expressed as a linear combination of a set of basis hub vectors, and is computed at query time efficiently. Experiments show that the approach is feasible when size of hub set $> 10^4$.

Haveliwala [4] use personalized PageRank to enable "topic-sensitive" web search. The approach precomputes k personalized PageRank vectors using k topics, e.g., the 16 top level topics of the Open Directory. For each topic i, a preference vector $\boldsymbol{u}_\mathbf{i}$ is generated. $(u_i)_j$ represents the confidence that web page j is classified into topic i. A PPV is computed base upon preference vector $\boldsymbol{u}_\mathbf{i}$. The k personalized PageRank vectors are combined at query time, using the context of the query to compute the appropriate topic weights. The experiments concluded that the use of personalized PageRank scores can improve web search, but the number of personalized PageRank vectors used was limited due to the computational requirements. In fact, this approach modulates the rankings based on the topic of the query and query context, rather than for truly "personalizing" the rankings to a specific individual. Qiu and Cho [9] develop a method to automatically estimate a user's topic preferences based on Topic-Sensitive PageRank scores of the user's past clicked pages. The topic preferences are then used to bias future search results.

Community-based Personalized Web Search

In most of the above personalized search strategies, each user has a distinct profile and the profile is used to personalize search results for the user. There are also some approaches that personalize search results for the preferences of a community of like-minded users. These approaches are called community-based personalized web search or collaborative web search. In a community-based personalized web search, when a user issues a query, search histories of users who have similar interests to the user are used to filter or re-rank search results. For example, documents that have been selected for the target query or similar queries by the community are re-ranked higher in the results list. Sugiyama et al. [13] use a modified collaborative filtering algorithm to constructed user profiles to accomplish personalized search. Sun et al. [14] proposed a novel method named CubeSVD to apply personalized web search by analyzing correlations among users, queries, and web pages in clickthrough data. Smyth et al. [12] show that collaborative web search can be efficient in many search scenarios when natural communities of searchers can be identified.

Server-Side and Client-Side Implement

Personalized web search can be implemented on either server side (in the search engine) or client side (in the user's computer or a personalization agent).

For server-side personalization, user profiles are built, updated, and stored on the search engine side. User information is directly incorporated into the ranking process, or is used to help process initial search results. The advantage of this architecture is that the search engine can use all of its resources, for example link structure of the whole web, in its personalization algorithm. Also, the personalization algorithm can be easily adapted without any client efforts. This architecture is adopted by some general search engines such as Google Personalized Search. The disadvantage of this architecture is that it brings high storage and computation costs when millions of users are using the search engine, and it also raises privacy concerns when information about users is stored on the server.

For client-side personalization, user information is collected and stored on the client side (in the user's computer or a personalization agent), usually by installing a client software or plug-in on a user's computer. In client side, not only the user's search behavior but also his contextual activities (e.g., web pages viewed before) and personal information (e.g., emails, documents, and bookmarks) could be incorporated into the user profile. This allows the construction of a much richer user model for personalization. Privacy concerns are also reduced since the user profile is strictly stored and used on the client side. Another benefit is that the overhead in computation and storage for personalization can be distributed among the clients. A main drawback of personalization on the client side is that the personalization algorithm cannot use some knowledge that is only available on the server side (e.g., PageRank score of a result document). Furthermore, due to the limits of network bandwidth, the client can usually only process limited top results.

Challenges of Personalized Search

Despite the attractiveness of personalized search, there is no large-scale use of personalized search services currently. Personalized web search faces several challenges that retard its real-world large-scale applications:

1. Privacy is an issue. Personalized web search, especially server-side implement, requires collecting and aggregating a lot of user information including query and clickthrough history. A user profile can reveal a large amount of private user information, such as hobbies, vocation, income level, and political inclination, which is clearly a serious concern for users [11]. This could make many people nervous and feel afraid to use personalized search engines. A personalized web search will be not well-received until it handles the privacy problem well.
2. It is really hard to infer user information needs accurately. Users are not static. They may randomly search for something which they are not interested in. They even search for other people sometimes. User search histories inevitably contain noise that is irrelevant or even harmful to current search. This may make personalization strategies unstable.
3. Queries should not be handled in the same manner with regard to personalization. Personalized search may have little effect on some queries. Some work [1,2,3] investigates whether current web search ranking might be sufficient for clear/unambiguous queries and thus personalization is unnecessary. Dou et al. [3] reveal that personalized search has little effect on queries with high user selection consistency. A specific personalized search also has different effectiveness for different queries. It even hurts search accuracy under some situations. For example, topical interest-based personalization, which leads to better performance for the query "mouse," is ineffective for the query "free mp3 download." Actually, relevant documents for query "free mp3 download" are mostly classified into the same topic categories and topical interest-based personalization has no way to filter out desired documents. Dou et al. [3] also reveal that topical interest-based personalized search methods are difficult to deploy in a real world search engine. They improve search performance for some queries, but they may hurt search performance for additional queries.

Key Applications

Personalized web search is considered a promising solution to improve the performance of generic web search. Currently, Google and other web search engines are trying to do personalized search.

Experimental Results

Experimental results have shown that personalized web search can indeed improve performance of web search. Detailed experimental results can be found in the corresponding reference for each presented method. Dou et al. [3] propose a personalized web search evaluation framework based upon large-scale query logs.

Cross-references

▶ Information Retrieval
▶ Privacy
▶ Relevance Feedback
▶ WEB Information Retrieval Models
▶ Web Search Relevance Feedback

Recommended Reading

1. Chirita P.A., Firan C., and Nejdl W. Summarizing local context to personalize global web search. In Proc. Int. Conf. on Information and Knowledge Management, 2006.
2. Chirita P.A., Nejdl W., Paiu R., and Kohlschütter C. Using ODP metadata to personalize search. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 178–185.
3. Dou Z., Song R., and Wen J. A large-scale evaluation and analysis of personalized search strategies. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007.
4. Haveliwala T.H. Topic-sensitive pagerank. In Proc. 11th Int. World Wide Web Conference, 2002.
5. Jeh G. and Widom J. Scaling personalized web search. In Proc. 12th Int. World Wide Web Conference, 2003, pp. 271–279.
6. Liu F., Yu C., and Meng W. Personalized web search by mapping user queries to categories. In Proc. Int. Conf. on Information and Knowledge Management, 2002, pp. 558–565.
7. Page L., Brin S., Motwani R., and Winograd T. The pagerank citation ranking: bringing order to the web. Technical report, Computer Science Department, Stanford University, 1998.
8. Pretschner A. and Gauch S. Ontology based personalized search. In Proc. 11th IEEE Int. Conf. on Tools with Artificial Intelligence, 1999, pp. 391–398.
9. Qiu F. and Cho J. Automatic identification of user interest for personalized search. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 727–736.
10. Shen X., Tan B., and Zhai C. Implicit user modeling for personalized search. In Proc. Int. Conf. on Information and Knowledge Management, 2005, pp. 824–831.
11. Shen X., Tan B., and Zhai C. Privacy protection in personalized search. SIGIR Forum, 41(1):4–17, 2007.
12. Smyth B., Coyle M., Boydell O., Briggs P., Balfe E., Freyne J., and Bradley K. A live-user evaluation of collaborative web search. In Proc. 19th Int. Joint Conf. on AI, 2005.
13. Sugiyama K., Hatano K., and Yoshikawa M. Adaptive web search based on user profile constructed without any effort from users. In Proc. 12th Int. World Wide Web Conference, 2004, pp. 675–684.
14. Sun J.-T., Zeng H.-J., Liu H., Lu Y., and Chen Z. CubeSVD: a novel approach to personalized web search. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 382–390.
15. Teevan J., Dumais S.T., and Horvitz E. Personalizing search via automated analysis of interests and activities. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 449–456.

Personally Identifiable Data

▶ Individually Identifiable Data

Perturbation Techniques

▶ Randomization Methods to Ensure Data Privacy

Perusal

▶ Browsing

Pessimistic Scheduler

▶ Two-Phase Locking

Petri Nets

W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven,
The Netherlands

Synonyms

Place transition nets; Condition event nets; Colored nets

Definition

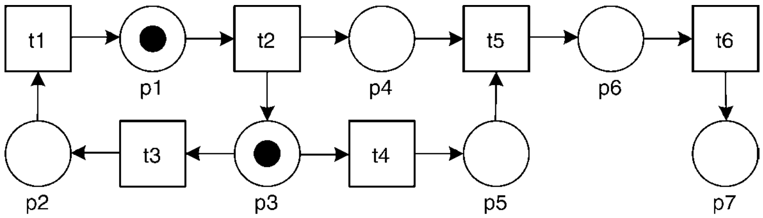
The Petri net formalism provides a graphical but also formal language which is appropriate for modeling systems and processes with concurrency and resource sharing. It was introduced in the beginning of the 1960's by Carl Adam Petri and was the first formalism to adequately describe concurrency. The classical Petri net is a directed bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Places are represented by circles and transitions by rectangles. The network structure of the Petri net is static. However, places may contain tokens and the distribution of tokens of places may change as described in the *firing rule*. Petri nets have formal semantics and allow for all kinds of analysis. Moreover, due to the strong theoretical foundation much is known about the properties of different subclasses of Petri nets. Petri nets have been extended in many different application domains, e.g., workflow management systems, flexible manufacturing, embedded systems, communication protocols, web services, asynchronous circuits, etc. Moreover, many modeling languages have been influenced by Petri nets (e.g., UML) and have been mapped onto Petri nets for analysis purposes (e.g., BPEL, BPMN, etc.). The classical Petri net has been extended to allow for the modeling of complex systems, e.g., so called "colors" have been added to model data, timestamps have been added to model time, and hierarchy concepts have been proposed to structure large models. The resulting models are called a *high-level Petri nets.*

Historical Background

Petri started his scientific career with his dissertation "Communication with Automata" [7], which he submitted to the Science Faculty of Darmstadt Technical University in July, 1961. He defended his thesis there in June, 1962 [8]. Petri's dissertation has been quoted frequently, but the Petri net formalism as it is known today emerged later. However, the fundamental idea that asynchronous systems are more powerful than synchronous systems was already present. Petri nets as they are used today first appeared in Petri's talk "Fundamentals on the description of discrete processes" at the Third Colloquium on Automata Theory in Hannover in 1965. Using a simple graphical representation consisting of two types of nodes (places and transitions), Petri was able to describe processes exhibiting concurrency and possibly infinitely many states. Figure 1 shows a simple Petri net consisting of six transitions and seven places. The black dots in places *p1* and *p3* denote tokens and are used to represent the initial state. Despite the addition of concurrency, various properties are decidable for Petri nets which are not decidable for Turing machines (e.g., reachability, liveness, boundedness, etc.).

Initially the focus was on understanding concurrency through Petri nets and little emphasis was put on practical applications. This changed over time and Petri nets were used more and more in all kinds of applications. A nice example is the work on office information systems in the late seventies and early eighties. People like Skip Ellis, Anatol Holt, and Michael Zisman worked on information systems which were driven by explicit process models. It is interesting to see that the three pioneers in this area independently used Petri-net variants to model office procedures. The ideas of these people resulted in all kinds of workflow management systems and today most of the workflow management systems use a notation close to Petri nets [1]. Petri nets have been used in many other application domains ranging from embedded systems and asynchronous circuits to flexible manufacturing, communication protocols, and web services. Petri nets also influenced the development of many languages ranging from process calculi to more application oriented languages such as UML, EPCs, etc.

The classical Petri net allows for the modeling of states, events, conditions, synchronization, parallelism, choice, and iteration. However, Petri nets describing real processes tend to be complex and extremely large.



Petri Nets. Figure 1. A Petri nets consisting of six transitions and seven places.

Moreover, the classical Petri net does not allow for the modeling of data and time. To solve these problems, many extensions have been proposed. Three well-known extensions of the basic Petri net model are: (1) the extension with *color* to model data, (2) the extension with *time*, and (3) the extension with *hierarchy* to structure large models. A Petri net extended with color, time, and hierarchy is called a *high-level Petri net* [2]. Using high-level Petri nets it is much easier to describe complex systems and processes. However, analysis of high-level Petri net other than simulation is often intractable.

Foundations

The classical Petri net is a directed bipartite graph consisting of *places* and *transitions* connected via directed *arcs*. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles as shown in Fig. 1.

A *Petri net* is formally defined by a triple (P, T, F):

1. P is a finite set of *places*,
2. T is a finite set of *transitions* $(P \cap T = \emptyset)$,
3. $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs* (flow relation)

A place p is called an *input place* of a transition t iff there exists a directed arc from p to t. Place p is called an *output place* of transition t iff there exists a directed arc from t to p. The notation $\bullet t$ is used to denote the set of input places for a transition t. The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g., $p\bullet$ is the set of transitions sharing p as an input place. In Fig. 1 transition *t2* has one input place (*p1*) and two output places (*p3* and *p4*).

At any time a place contains zero or more *tokens*, drawn as black dots. The *state*, often referred to as marking, is the distribution of tokens over places, i.e., $M \in P \to \mathrm{IN}$. A state is represented as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the state with one token in place p_1, two tokens in p_2, one token in p_3 and no tokens in p_4. The notation $p_1 + 2p_2 + p_3$ can also be used represent this state. To compare states a partial ordering is defined. For any two states M_1 and M_2, $M_1 \leq M_2$ iff for all $p \in P$: $M_1(p) \leq M_2(p)$.

The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

1. A transition t is said to be *enabled* iff each input place p of t contains at least one token.
2. An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token for each output place p of t.

The initial marking shown in Fig. 1 is *p1* + *p3*. In this marking, *t2*, *t3*, and *t4* are enabled. Firing *t2*, results in the state 2*p3* + *p4*. Firing *t3*, results in the state *p1* + *p2*. Firing *t4*, results in the state *p1* + *p5*. Note that firing *t3* disables *t4* and vice versa, i.e., there is a non-deterministic choice between *t3* and *t4*. However, *t2* can fire independently from *t3* and *t4* because it is in parallel.

Given a Petri net (P, T, F) and a state M_1, the following notations are used:

1. $M_1 \xrightarrow{t} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2
2. $M_1 \rightarrow M_2$: there is a transition t such that $M_1 \xrightarrow{t} M_2$
3. $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 ... t_{n-1}$ leads from state M_1 to state M_n via a (possibly empty) set of intermediate states $M_2,...M_{n-1}$, i.e., $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$

A state M_n is called *reachable* from M_1 (notation $M_1 \xrightarrow{*} M_n$) iff there is a firing sequence σ such that $M_1 \xrightarrow{\sigma} M_n$. Note that the empty firing sequence is also allowed, i.e., $M_1 \xrightarrow{*} M_1$.

(PN, M) is used to denote a Petri net PN with an initial state M. A state M' is a *reachable state* of (PN, M) iff $M_1 \xrightarrow{*} M'$.

The marked Petri net shown in Fig. 1 has an unbounded number of states. Note that the sequence *t2 t3 t1* can be repeated over and over again. In each cycle an additional token is put in place *p4*.

In the remainder, some standard properties for Petri nets are defined [3,4]. First, properties related to the dynamics of a Petri net are defined. Then some structural properties are given.

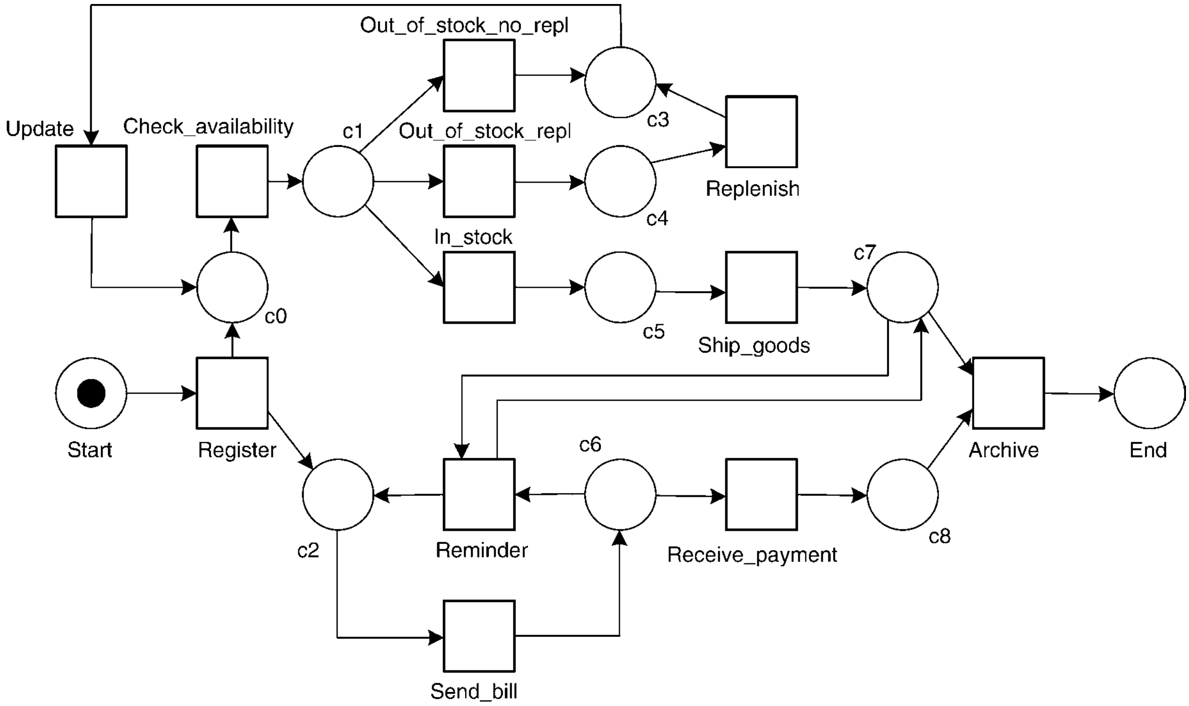
A Petri net (PN, M) is *live* iff, for every reachable state M' and every transition t there is a state M' reachable from M' which enables t. Note that the marked Petri net shown in Fig. 1 is not live. After firing *t4* twice, it becomes impossible to execute any of the first three transitions. A Petri net is *structurally live* if there exists an initial state such that the net is live. Figure 1 is also not structurally live.

A Petri net (PN, M) is *bounded* iff for each place p there is a natural number n such that for every reachable state the number of tokens in p is less

P

than *n*. The net is safe iff for each place the maximum number of tokens does not exceed 1. As indicated before, the marked Petri net shown in Fig. 1 is not bounded because the number of tokens in *p4* can exceed any number. A Petri net is *structurally bounded* if the net is bounded for any initially state.

Figure 2 shows a more meaningful example of a Petri net. In fact it models the workflow related to some ordering process and the transitions correspond to tasks. The token in *start* corresponds to an order. Task *register* is represented by a transition bearing the same name. From a routing point of view it acts as a so-called AND-split (two outgoing arcs) and is enabled in the state shown. If a person executes this task, the token is removed from place *start* and two tokens are produced: one for *c0* and one for *c2*. Then, in parallel, two tasks are enabled: *check_availability* and *send_bill*. Depending on the eagerness of the workers executing these two tasks either *check_availability* or *send_bill* is executed first. Suppose *check_availability* is executed first. Based on the outcome of this task a choice is made. This is reflected by the fact that three arcs are leaving *c1*. If the ordered goods are available, they can be shipped, i.e., firing *in_stock* enables task *ship_goods*. If they are not available, either a replenishment order is issued or not. Firing *out_of_stock_repl* enables task *replenish*. Firing *out_of_stock_no_repl* skips task *replenish*. Note that *check_availability*, place *c1* and the three transitions *in_stock*, *out_of_stock_repl*, and *out_of_ stock_no_repl* together form a so-called OR-split: As a result of this construct one token is produced for either *c3*, *c4*, or *c5*. Suppose that not all ordered goods are available, but the appropriate replenishment orders were already issued. A token is produced for *c3* and task *update* becomes enabled. Suppose that at this point in time task *send_bill* is executed, resulting in the state with a token in *c3* and *c6*. The token in *c6* is input for two tasks. However, only one of these tasks can be executed and in this state only *receive_payment* is enabled. Task *receive_payment* can be executed the moment the payment is received. Task *reminder* is an AND-join/AND-split and is blocked until the bill is sent and the goods have been shipped. However, it is only possible to send a reminder if the goods have actually been shipped. Assume that in the state with a token in *c3* and *c6* task *update* is executed. This task does not require human involvement and is triggered by a message of the warehouse indicating that relevant goods have arrived. Again *check_availability* is enabled. Suppose that this task is executed and the result is



Petri Nets. Figure 2. A workflow expressed in terms of a Petri net.

positive, i.e., the path via *in_stock* is taken. In the resulting state *ship_goods* can be executed. Now there is a token in *c6* and *c7* thus enabling task *reminder*. Executing task *reminder* enables the task *send_bill* for the second time. A new copy of the bill is sent with the appropriate text. It is possible to send several reminders by alternating *reminder* and *send_bill*. However, assume that after the first loop the customer pays resulting in a state with a token in *c7* and *c8*. In this state, the AND-join *archive* is enabled and executing this task results in the final state with a token in *end*.

The marked Petri net shown in Fig. 2 is bounded but not live. In fact the model is safe because there are never two tokens in the same place.

Reachability, liveness, and boundedness can be decided for any classical Petri net. For example, it is possible to construct the so-called coverability graph. However, this may be quite inefficient. Fortunately, more efficient techniques are available for different subclasses of Petri nets. Some of these subclasses are listed below.

A Petri net is a *free-choice Petri net* [5]. iff, for every two transitions t_1 and t_2, $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ implies $\bullet t_1 = \bullet t_2$. Figure 1 is free-choice but Fig. 2 is not. Figure 2 is not free-choice because of the two arcs between *c7* and *reminder*. Many properties of free-choice nets can be exploited in their analysis. For example, the combination of liveness and boundedness can be decided in polynomial time.

A Petri net is *state machine* iff each transition has exactly one input and one output place. A Petri net is *marked graph* iff each place has exactly one input and one output transition. Both state machines and marked graphs are examples of free-choice nets and can be analyzed efficiently.

A Petri net $PN = (P, T, F)$ is a WF-net (Workflow net, [1,6]) if and only if:

1. There is one source place $i \in P$ such that $\bullet i = \emptyset$.
2. There is one sink place $o \in P$ such that $o\bullet = \emptyset$.
3. Every node $x \in P \cup T$ is on a path from i to o.

The class of WF-net has been extensively studied because of its applications in workflow management processes and other case-driven processes. Clearly, Fig. 2 is a WF-net while Fig. 1 is not.

For a WF-net with one token on the source place i it is interesting to know whether the net will always terminate properly with a token in the sink place o. This corresponds to the so-called *soundness property*. Soundness can be expressed in terms of liveness and boundedness. If the sink place o is connected to the source place i using some transition $t*$, the so-called short-circuited net is obtained. The original WF-net is sound if and only if the corresponding short-circuited net is live and bounded. This result can be exploited in the analysis of complex workflows.

Key Applications

Workflow Management

Petri nets are used for the modeling, analysis, and enactment of workflows. Many workflow languages have a graphical representation close to Petri nets. Moreover, today's workflow engines use mechanisms comparable to using tokens and the firing rule.

Discrete/Flexible Manufacturing

Traditionally, there have been may applications of Petri nets in manufacturing. The models are used to analyze the manufacturing process in detail, e.g., to find deadlocks or to analyze the performance.

Communication Protocols

Different protocols have been modeled using Petri nets. Various errors have been discovered by verifying established protocols using Petri-net-based analysis techniques.

Embedded Systems

The interactions between hardware and software in embedded systems (e.g., copiers, cars, mobile phones, etc.) may result in all kinds of problems. Therefore, Petri nets are used to model and analyze such systems.

Cross-references

- ▶ Business Process Management
- ▶ Process Mining
- ▶ Web Services
- ▶ Workflow Model Analysis
- ▶ Workflow Patterns

Recommended Reading

1. Brauer W. and Reisig W. Carl Adam Petri and Petri Nets. Informatik-Spektrum, 29(5):369–374, 1996.
2. Desel J. and Esparza J. Free choice Petri nets, volume 40 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK, 1995.

3. Jensen K., Kristensen L.M., and Wells L. Coloured Petri nets and CPN tools for modelling and validation of concurrent systems. Int. J. Softw. Tools Technol. Trans., 9(3–4):213–254, 2007.
4. Murata T. Petri nets: properties, analysis and applications. Proc. IEEE, 77(4):541–580, April 1989.
5. Petri C.A. Kommunikation mit Automaten. PhD Thesis, Fakultät für Mathematik und Physik, Technische Hochschule Darmstadt, Darmstadt, Germany, 1962.
6. Reisig W. and Rozenberg G. editors. Lectures on Petri nets I: basic models, Springer-Verlag, Berlin Heidelberg New York, 1998.
7. van der Aalst W.M.P. The application of Petri nets to workflow management. J. Circuit. Syst. Comput., 8(1):21–66, 1998.
8. van der Aalst W.M.P. and Hee K.M. van Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge, MA, 2004.

Photograph

► Image

Physical Clock

Curtis Dyreson
Utah State University, Logan UT, USA

Synonyms

Clock

Definition

A *physical clock* is a physical process coupled with a method of measuring that process to record the passage of time. For instance, the rotation of the Earth measured in *solar days* is a physical clock. Most physical clocks are based on cyclic processes (such as a celestial rotation). One or more physical clocks are used to establish a *time-line clock* for a temporal database.

Key Points

Every concrete time is a measurement of some physical clock. For instance a wind-up watch provides the time "now" by measuring the rate at which a coiled, wound spring unwinds. A physical clock is limited by the durability and regularity of the underlying physical process that it measures, so to provide a clock for every instant in a time-line for a temporal database several physical clocks might be needed. For instance a clock based on the rotation of the Earth will (likely be) limited as current models predict the Earth will eventually be gravitationally drawn into the Sun. The modifier "physical" distinguishes this kind of clock from other kinds of clocks in an application, in particular a "time-line" clock. Atomic clocks, which are a kind of physical clock, provide the measurements for International Atomic Time (TAI).

Cross-references

► Chronon
► Time-Line Clock
► Time Instant

Recommended Reading

1. Dyreson C.E. and Snodgrass R.T. Timestamp Semantics and Representation. Inf. Syst., 18(3), 1993, pp. 143–166.
2. Dyreson C.E. and Snodgrass R.T. The baseline clock. The TSQL2 temporal query language, Kluwer, pp. 73–92, 1987.
3. Fraser J.T. Time: the familiar stranger. University of Massachusetts Press, 1987.

Physical Database Design for Relational Databases

Sam S Lightstone
IBM Canada Ltd., Markham, ON, Canada

Synonyms

Database design; Database materialization; Database implementation; Table design; Table normalization; Indexing; Clustering; Multidimensional clustering; Range partitioning; Materialized views; Materialized query tables

Definition

Physical database design represents the materialization of a database into an actual system. While logical design can be performed independently of the eventual database platform, many physical database attributes depend on the specifics and semantics of the target DBMS. Physical design is performed in two stages:

1. Conversion of the logical design into table definitions (often performed by an application developer): includes pre-deployment design, table definitions, normalization, primary and foreign key relationships, and basic indexing.

2. Post deployment physical database design (often performed by a database administrator): includes improving performance, reducing I/O, and streamlining administration tasks.

Generally speaking, physical database design covers those aspects of database design that impact the actual structure of the database on disk. Stage 1 is variably referred to in the industry as an aspect of *logical database design* or *physical database design*. It is known as *logical database design* in the sense that it can be designed independent of the data server or the particular DBMS used. It is also often performed by the same people who perform the early requirements building and entity relationship modeling. Conversely, it is also called *physical database design* in the sense that it affects the physical structure of the database and its implementation. For the sake of this entry, the latter assumption is used, and it is therefore included as part of *physical database design*.

Although it is possible to perform logical design independently of the physical platform and knowledge of database software that will eventually be used, many physical database design tasks depend on the specifics and semantics of the target DBMS (software and hardware). The major tasks of physical database design include: table normalization, table denomalization, indexing, clustering, multidimensional clustering (MDC), range partitioning, hash partitioning, shared nothing partitioning (hash), materialized views (MVs), memory allocation, database storage topology, and database storage object allocation.

Historical Background

Physical database design is as old as database research. Specifically, if this question is limited to relational databases, the first systems were prototyped in the early 1970's. The most elementary problems of physical database design are those of table normalization and index selection. The relational model for databases was first proposed in 1970 by E.F Codd at IBM. The first relational databases using SQL and B+-tree was IBM's System R, in 1976 and the INGRES database at the University of California, Berkeley. The B+-tree, the most commonly-used indexing storage structure for user designed indexes, was first described in the paper *Organization and Maintenance of Large Ordered Indices* by Rudolf Bayer and Edward M. McCreight. While System R was a research prototype, commercialization followed with products by IBM (DB2 – DB2 and Informix are trademarks or registered trademark of International Business Machines Corporation in the United States, other countries, or both. Oracle is a trademarks or registered trademark of Oracle Corporation in the United States, other countries, or both. Other company, product, or service names may be trademarks or service marks of others.) and Oracle around 1980. INGRES also became commercial and was followed by POSTGRES which was incorporated into Informix. Prior to these relational systems, other data models for databases had been used, such as IMS which uses a hierarchical storage model and dominated industrial practice between 1967 and 1980. These pre-relational systems also had unique physical database design challenges.

As relational database systems advanced, new techniques were introduced to help improve operational efficiency, and reduce I/O by partitioning, distributing, and/or better indexing data. All of these innovations, while improving the capabilities and potential for databases, expanded the scope of physical database design increasing the number of design choices and therefore the complexity of optimizing database structures. Though the 1980's and 1990's were dominated by the introduction of new physical database design capabilities, it can also be said that the years since have been dominated by efforts to simplify the process through automation, best practices, or deploying pre-configured "database appliances".

Foundations

The vast majority of physical database design tasks have the primary goal of reducing I/O consumption at runtime for query processing. However, to a lesser degree there are physical design aspects that help improve administrative efficiency by reducing the granularity of backup and restore operations (as in the case of range partitioning), or by improving the efficiency of mass insertion or deletion of data (so called "roll-in" and "roll-out" of data) as in the case of range partitioning. Some features can help reduce CPU or network consumption as in the case of memory tuning for heaps associated with these resources (such as the compiled SQL statement cache found in many commercial database server products).

Infrastructure Mechanics

Indexing schemes typically exploit tree-based structures or hash methods to provide fast access to data, with B+-trees being the dominant strategy. Range partitioning and clustering methods are based on grouping strategies to physically cluster like or similar data nearby on disk in order to reduce I/O. Hash partitioning methods hash data horizontally across storage objects so that processing can be performed on records associated with the different hash partitioning by different CPUs, providing parallel processing efficiencies. MVs are based on the strategy of pre-computation, so that results are available either in full or in part when needed without scanning and calculation. Considerable literature has been published on strategies to match the information within the MV at query compilation time to the queries requiring them, which is non trivial except in cases where the incoming query is an exact match to the MV.

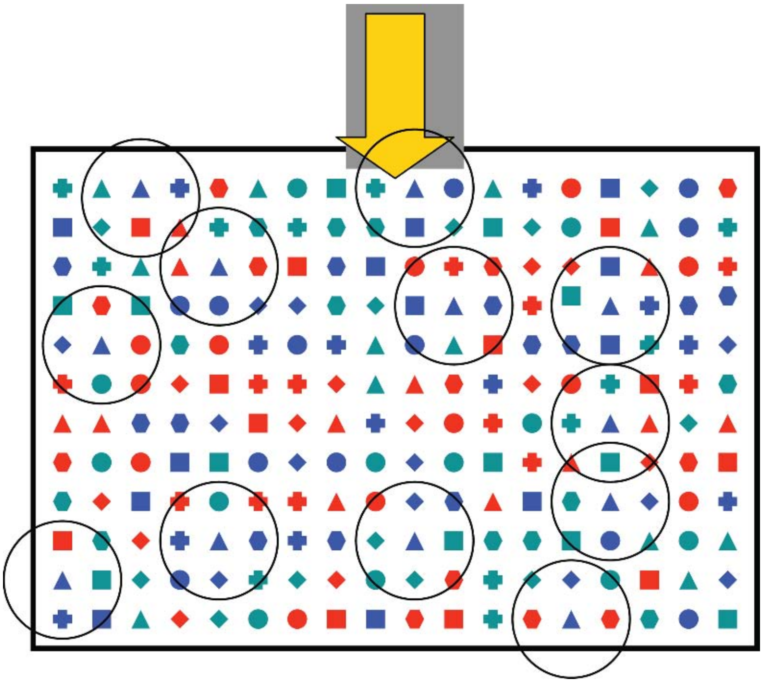
The full breadth of physical database design is beyond the scope of this entry, so this discussion touches only on major aspects.

Design Choices

Physical database design, as performed by end users, has not yet been codified into a scientifically based strategy, and remains as much an art as a science. Typical strategies include cost-benefit analysis of applying various design features. Trial-and-error is still a significant part of the design process. For query processing, indexing is the first design feature considered, and the design process can generally explore the columns used in query predicates within the workload. Range partitioning is generally designed around date fields (by month or by quarter) for roll-in and roll-out processing, and backup granularity. Hash partitioning typically focuses on a unique key or near-unique key in order to efficiently distribute records across CPUs without skew. Data clustering strategies typically focus on columns used in range predicates, or in equality clauses that have high cardinality results sets in order to place similar data that are likely to be jointly required for query resolution physically nearby one another on disk.

Combining Physical Design Choices

Figure 1 shows a conceptual illustration of data stored in one large physical object without any specific treatment of the data, so that the data is unsorted, unclustered, unpartitioned, and not indexed. Data of different characteristics is illustrated by the different colored shapes that appear in the set. One can consider a query that was interested in data represented by the

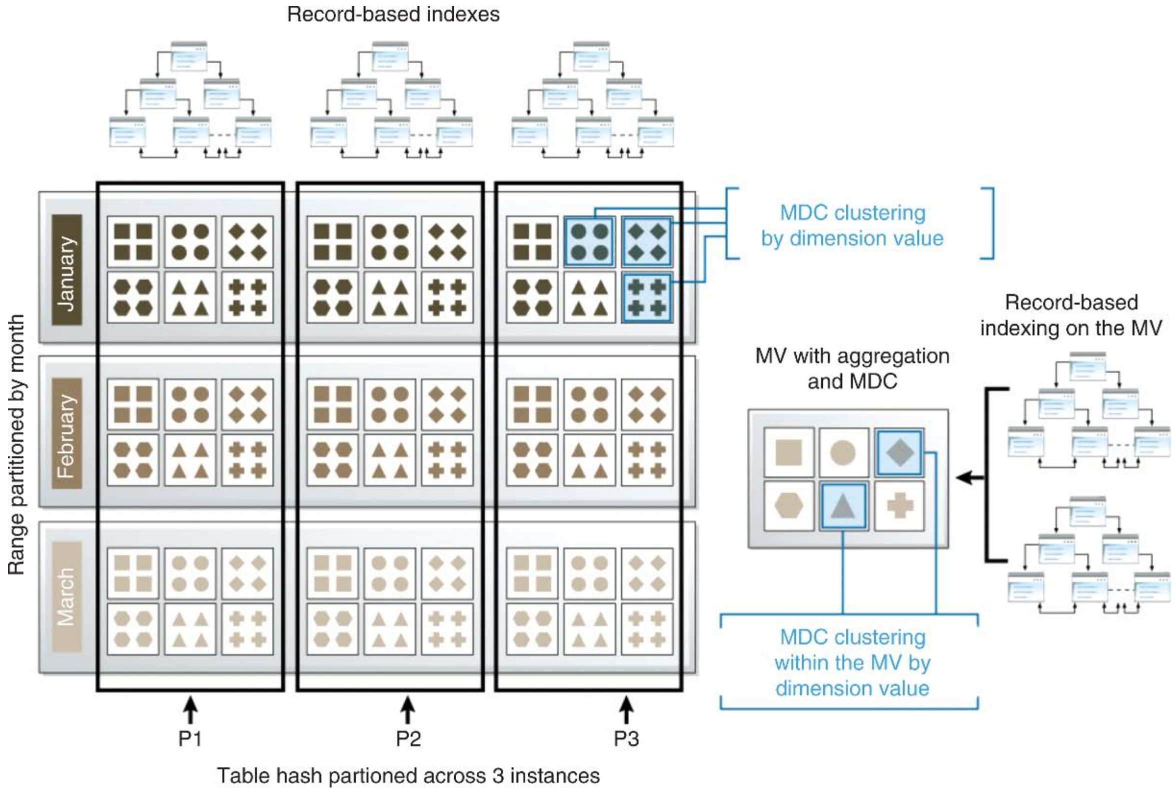


Physical Database Design for Relational Databases. Figure 1. Data of various attributes, stored in a single monolithic table. (image courtesy of K. Beck at IBM) blue triangles in the image, which appear scattered throughout the object. A full data scan of the set would be required to find and fetch these data.

Figure 2 shows a database with the same data as Fig. 1, which has been carefully designed to exploit a number of physical database design features in combination, such as indexing, range partitioning, MVs, MDC, and hash partitioning. In this new configuration a query for the data represented by the blue triangles is not only clustered in order to perform minimal IO, but the clearly indexed and can be distributed across several CPUs due to hash partitioning. Aggregation on the data can be precomputed and stored in the MV as well.

Other Physical Database Design Techniques

1. Other indexing mechanisms: hash index, bitmap index
   - These indexing strategies use hash functions, or bitmaps to index data that is typically slow changing. Bitmap indexes are favors for data with low cardinality (i.e., few distinct values).
2. Physical design for XML data
   - Semi-structured databases, like XML databases, require alternate indexing strategies.
3. Indexing for Information Retrieval
   - Text indexing is often based on an “inverted tree” index method.
4. Storage and object layout
   - Data placement on disk is affects the efficiency of storage and retrieval. Particularly important when some data has very different frequency or urgency of access than others.
5. Page and block size for I/O efficiency
   - Size of the storage pages, and the blocking levels (number of pages) used during I/O dramatically affect I/O efficiency, where random access favors smaller blocking and while sequential access favors large blocking.
6. Memory configuration and management
   - Complex design choice in the distribution of memory for caching, sorting, hashing, locking, tat typically has large impact on the efficiency of a database, especially if any key memory


Record-based indexes
MDC clustering by dimension value
Record-based indexing on the MV
MV with aggregation and MDC
January
February
March
Range partitioned by month
MDC clustering within the MV by dimension value
P1
P2
P3
Table hash partioned across 3 instances

Physical Database Design for Relational Databases. Figure 2. Data stored using a number of physical design extensions concurrently.

consumer is constrained. Since memory is a fixed resource, the distribution of memory is a zero sum game (i.e., adding memory for one purpose requires reducing it for another)

7. Distributed database design
   - Distributed databases require complex design choices for data placement and choice of access (particularly where the distributed database includes redundancy, providing choices for which node to access).
8. Shared nothing partitioning
   - Shared nothing partitioning, common in data warehousing, hash partitions data across CPUs (and/or server) to achieve parallel processing over large volumes of data. The choice of hashing key is non trivial. However, in a shared nothing architecture, queries over each hash bucket later need to be shipped and merged since each node (or partition) is assumed to be distinct.
9. Hash partitioned tables
   - Hash partitioning within a server, can divide process of a query across CPUs, while still allowing the efficiencies of shared memory processing within a server.

Key Applications

Lifecycle Differences

It is important to distinguish between physical database design tasks performed during application development, versus those performed following deployment by an administrator.

During application development, the application designer performs logical database design and derives from it a physical design of the database. The physical design assigns typing to the elements of the entities in the database. This phase also designs the database tables, including normalization. Indices are defined, most commonly on primary and foreign keys. Views (non-materialized) are created to abstract the physical implementation away from the application as much as possible and to improve security of the system by controlling access to the data by the logical definition of the views, rather than by the physical structures of the storage.

Following deployment of the database, physical design continues, usually conducted by a database administrator. The design tasks here focus on performance tuning, or performance problem alleviation. Common tasks include the selection of secondary indexes, clustering, MDC, and range partitioning.

A common problem in database application development is that the application designer is forced to develop the database application using small samples of data. DBAs face design challenges when trying to optimize data access for applications that operate over large data sets, which the application designers themselves did not have access to when performing the initial application and database design.

Another key difference between the database design tasks performed at application development time and those following deployment relate to the physical resources of the database server. In most cases the properties of the database server – number and speed of CPUs, CPU-cores, disks, memory, bus bandwidth, etc – are unknown during application development. For applications that are deployed to possibly many sites or many customers there can be many different eventual target servers. Thus physical design qualities of the database that depend on the server's characteristics are inherently the role of the DBA to design. These include data placement and storage topology, memory allocation, and to a lesser degree clustering choices.

The design tasks performed by the DBA occur frequently with the deployment of new applications, application upgrades, or due to the evolution of the data set.

Application Domains

Different application domains require different physical design techniques. The section below highlights the two largest categories of application domains, namely Online Transaction processing (OLTP) and Decision Support Systems (DSS).

OLTP/ERP. Such applications deal with small numbers of records per transaction, and are commonly subject to extremely high concurrency rates. Key physical design characteristics include: (i) Small storage page sizes; (ii) Highly normalized data; (iii) Use of clustering indices or single-dimension MDC to enforce clustering of data; (iv) Memory allocation is heavily used for data caching (bufferpools), and to a lesser degree for caching compiled SQL statements. Memory work area for sorting, hash joins, etc, has considerably reduced demand in this space; (v) Due to the small selectivity of the transactions, the use of range partitioning, MDC with high dimensionality, and view materialization are not heavily used.

Decision Support, Data Warehousing and OLAP. These applications are characterized by large numbers of records accessed for aggregation, cubing etc. Table scans are common, and the workloads often include a high percentage of ad-hoc unpredictable queries, which can not be explicitly optimized for. Key physical design considerations here include: (i) Larger page size and storage blocking to increase the efficiency of scans and prefetching; (ii) Data may be stored in 3NF or Star Schema; (iii) Use of clustering is extremely common, especially long date and geography dimensions; (iv) Range partitioning is heavily used for roll-in and roll-out of data within the warehouse. This is almost always performed along a date dimension (such as by month or by quarter); (v) Memory allocation is complex in these environments. Data caching remains dominant, however, the common occurrence of sorting and hashing in these workloads places increased demand on work memory for these operations. The low concurrency characteristics of these workloads (compared to OLTP systems) place a reduced demand on memory for locking and caching of compiled statements.

Future Directions

The powerful capabilities of modern RDBMSs add complexity and have given rise to an entire professional domain of experts, known as Database Administrators (DBAs), to manage these systems. The problem is not unique to RDBMSs, but is ubiquitous in Information Technology today. Some have called this explosive growth of complexity "creeping featurism". In fact many companies now spend more money recruiting administrators to manage their technology than the money they have spent on the technology itself. One of the key roles of such DBAs is in the development and refinement of physical database design, to achieve "tuning" of the database.

The only real viable long term strategic solution is to develop systems that manage themselves. Such systems are called "self-managing" or "autonomic" systems. Several research efforts have focused on such self-designing database systems, working on problems such as:

1. What-if analysis reusing a database's native query optimizer to explore hypothetical design choices over a real or synthetic workload.
2. Data sampling and statistical analysis to explore the impact of design choices on data distribution and density, or conversely to assess the suitability of target data for design attributes.

Cross-references

▶ Advanced Storage Systems
▶ Database Design
▶ Database Management System
▶ Database Middleware
▶ Database Security
▶ Database Tuning and Performance
▶ Distributed Database Systems
▶ Hash Partitioning
▶ Indexing
▶ Multidimensional Clustering
▶ Parallel Database
▶ Range Partitioning
▶ Self-Management Technology in Databases
▶ Storage Systems
▶ Workflow Management

Recommended Reading

1. Astrahan M.M., Blasgen M.W., Chamberlin D.D., Jim Gray W., King F. I.I., Lindsay B.G., Lorie R.A., Mehl J.W., Price T.G., Putzolu G.R., Schkolnick M., Selinger P.G., Slutz, D.R., Strong H.R., Tiberio, P., Traiger, I.L., Bradford W., Yost W.R.A. System R: A relational data base management system. IEEE Comput, 12(5):42–48, 1979.
2. Bayer R. and McCreight E.M. Organization and Maintenance of Large Ordered Indexes. SIGFIDET Workshop, 107–141, 1970.
3. Finkelstein S., Schikolnick M. and Tiberio P. Physical Database Design for Relational Databases. ACM Trans Database Syst, 13(1):91–128, 1988.
4. Jun R., Chun Zhang, Megiddo N., and Lohman G.M. Automating physical database design in a parallel database. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 558–569.
5. Lightstone S., and Bishwaranjan B. Automated design of Multi-dimensional clustering tables for relational databases. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp.1170–1181.
6. Lightstone S., Teory T., and Nadeau T. Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More. Morgan Kaufmann, 2007.
7. Markos Z., Cochrane R., Lapis G., Hamid P., and Monica U. Answering complex SQL queries using automatic summary tables. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 105–116.
8. Sanjay A., Surajit C., and Narasayya V.R. Automated Selection of Materialized Views and Indexes in SQL Databases, In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 496–505.
9. Sriram P., Bishwaranjan B., Malkemus T., Cranston L., and Huras M. Multi-dimensional clustering: A new data layout scheme in DB2. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 637–641.

10. Surajit C., and Narasayya V.R. AutoAdmin "What-if" index analysis utility. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp.367–378.
11. Surajit C., and Narasayya V.R. Microsoft Index Tuning Wizard for SQL Server 7.0, In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp.553–554.
12. Teorey T., Lightstone S., and Nadeau T. Database Modeling & Design: Logical Design, 4th edn. Morgan Kaufmann, 2005.
13. Zilio D.C., Jun R., Lightstone S., Lohman G.M., and Storm A.J. Christian Garcia-Arellano, and Scott Fadden. DB2 Design Advisor: Integrated Automatic Physical Database Design. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 1087–1097.
14. Zilio, D.C., Zuzarte C., Lightstone S., Wenbin Ma, Lohman G. M., Cochrane R., Hamid P., Latha S.C., Gryz J., Alton E., Liang D., Valentin G.: Recommending materialized views and indexes with IBM DB2 design advisor. In Proc. 1st Int. Conf. on Autonomic Computing, 2004, pp. 180–188.

Physical Layer Tuning

Philippe Bonnet, Dennis Shasha
University of Copenhagen, Copenhagen, Denmark
New York University, New York, NY, USA

Definition

Tuning the physical layer entails choosing and configuring the underlying hardware and operating system to improve the performance of a database system running a given set of applications.

Historical Background

Ultimately, a database management system runs on processor and memory chips as well as secondary devices such as flash memory and disks. Trying to use those physical resources in the best way possible has been a problem since the first databases were available. Many researchers have noticed that disk capacity has been less of an issue than disk speed (or aggregate bandwidth from an entire disk array). Main memory size has always been a critical factor, because accesses to main memory are so much faster than accesses to disk. Processor speed has been important in applications that mix sophisticated computation with data access (for example in financial analysis), such computations are often handled outside the database server. New technologies have improved the capacity and speed of processors, memory, and disks, but memory is still vastly faster than disk (and somewhat faster than flash memory), so certain tuning principles will remain true.

Foundations

Hardware Tuning

Hardware tuning essentially consists of picking the hardware components on which the database management server is run. The goal is to design a balanced system [2].

Each processing unit consists of one or more processors, one or more disks, and some memory. Assuming a n-GIPS (billion instructions per second) processor, disks will be the bottleneck for on-line transaction-processing applications until the processor is attached to around 30n disks (counting 500,000 instructions per random IO issued by the database system and 70 random IOs per second). Each transaction spends far more time waiting for head movement on disk than in the processor.

Decision-support queries, by contrast, often entail massive scans of a table. In theory, an n-GIPS processor is saturated when connected to n/4 disks (counting 500,000 instruction per sequential IO and 8,000 IO per second, considering 50 MB/second per disk and 64 KB per IO). In practice, the system bus might become the bottleneck. Thus, decision-support sites may need fewer disks per processor than transaction-processing sites for the purposes of matching aggregate disk bandwidth to processor speed. The numbers change when disks are implemented using flash memory, but the principle remains the same.

Operating System Tuning

Tuning the operating system primarily consists of providing an efficient input/output (IO) subsystem to the database server. A database server can issue buffered or unbuffered IO using the file system, or it can issue IO on raw disks thus bypassing the file system. Raw disks offer control and performance to the database system at the cost of a very low level of abstraction. The file system, on the other hand, provides a high level of abstraction, at the cost of decreased performance and double buffering. Further, IO is either synchronous or asynchronous. A synchronous IO forces the thread that issued the issuing thread to block until the IO is complete. Asynchronous IO allows the database server to issue concurrent IO requests on multiple files. Modern operating system such as Linux or Windows, provide database servers with direct asynchronous IO, that provide file abstraction and efficient IO scheduling at a low overhead. Note that out-of-the-box, a database system will most likely be configured to use standard buffered IO and thus needs to be tuned to use direct asynchronous IO.

As noted above, if one could eliminate the overhead caused by seeks and rotational delay, the aggregate bandwidth could increase by a factor of 10–100. Making this possible requires laying out the data to be read sequentially along disk tracks. Recognizing the advantage of sequential reads on properly laid-out data, most database systems encourage administrators to lay out tables in relatively large *extents* (consecutive portions of disk). Having a few large extents is a good idea for tables that are scanned frequently or (like database recovery logs or history files) are written sequentially. Large extents, then, are a necessary condition for good performance, but not sufficient, particularly for history files. Consider, for example, the scenario in which a database log is laid out on a disk in a few large extents, but another hot table is also on that disk. The accesses to the hot table may entail a seek from the last page of the log; the next access to the log will entail another seek. So, much of the gain of large extents will be lost. For this reason, each log or history file should be the only hot file on its disk, unless the disk makes use of a large RAM cache to buffer the updates to each history file.

When accesses to a file are entirely random (as is the case in on-line transaction processing), seeks cannot be avoided. But placement can still minimize their cost, since seek time is roughly proportional to a constant plus the square root of the seek distance.

The operating system also allows to control the thread model used by the database server and to assign priorities to the different database server threads (and processes). The goal is to favor low latency requests (log writes), ensure fairness among database clients while avoiding starvation. This has several pitfalls however as discussed in [3] below.

Key Applications

A database system depends on the underlying hardware and operating system. Tuning the physical layer is thus a necessary step when deploying any database application.

Cross-references

▶ Database Management System
▶ Disk
▶ Main Memory
▶ Processor Cache

Recommended Reading

1. Gray J. and Kukol P. Sequential disk IO tests for GBps land speed record. MS Research Technical Report MSR-TR-2004–62.
2. Gray J. and Shanoy P.J. Rules of thumb in data engineering. In Proc. 18th Int. Conf. on Data Engineering, 2000.
3. Hall C. and Bonnet Ph. Getting priorities straight: improving Linux support for database I/O. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005.
4. Shasha D. and Bonnet P. Database tuning: principles, experiments and troubleshooting techniques. Morgan Kaufmann, 2002.

Physical Time

▶ Transaction Time

Physical Volume

▶ Volume

Physical Window = Tuple-based Windows

▶ Windows

Physician Order Entry

▶ Computerized Physician Order Entry

Pictorial Metadata

▶ Image Metadata

Picture

▶ Icon
▶ Image

Picture Metadata

► Image Metadata

Piecewise-Constant Approximations

► Histograms on Streams

Pipeline

Ryan Johnson
Carnegie Mellon University, Pittsburg, PA, USA

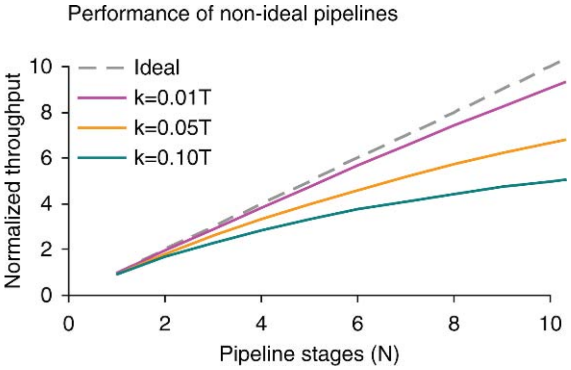
Definition

Pipelining is a performance-enhancing technique which breaks a job into multiple overlapping pieces of work assigned to *pipeline stages*. Each pipeline stage operates in parallel, receiving inputs from a previous stage and passing intermediate results to the next. A pipeline with N stages can accommodate up to N in-progress jobs at a time. A well-designed pipeline requires only a fraction of the resources needed to achieve the same throughput with non-pipelined execution. A *balanced* pipeline, which distributes processing time evenly between all stages, can produce N results during the time it would have taken to process a single job all at once; unbalanced pipelines do not perform nearly as well. Pipelining also provides a secondary benefit of allowing each pipeline stage to specialize for its particular task, potentially making the combined stages both faster and less expensive than the non-pipelined server. Factory assembly lines leverage pipelining to great effect, allowing N specialized workers to achieve far higher throughput than N general purpose workers, while requiring less training per worker and fewer resources overall. The impressive throughput and efficiency pipelining provides make it a useful tool in both hardware and software design. Database systems often improve performance of query processing by executing query plans in pipelined fashion. Pipelining is a form of parallel dataflow execution.

Key Points

Suppose a non-pipelined server requires T time units and R resources to process a job all at once. The server's *latency* (time to process a job start-to-finish) and *occupancy* (time until it can accept more work) are both T. In contrast, a pipelined server with N stages ideally provides latency and occupancy of T' $\leq$ T and T'/N time units, respectively, while consuming R + ε resources. Because peak throughput is the reciprocal of the server's occupancy, pipelined execution potentially provides an N-fold increase in throughput over non-pipelined execution. When plenty of work is available, an initially empty pipeline *fills* or *ramps up* over a period of T' time units to reach peak operating capacity with N in-progress jobs. The pipeline must *stall* any time the first pipeline stage becomes ready and no new jobs are available. Stalls propagate through the pipeline as *bubbles* and reduce throughput by leaving stages idle.

Ideal pipelines assume that (i) the job can be broken into N pieces requiring no more than T/N time units each (possibly less due to gains from specialization), and that (ii) passing work between stages imposes no extra overhead. In practice, pipelines usually have non-negligible stage overhead and tend to be unbalanced, with a *bottleneck stage* that runs slower than the others. Real pipelines therefore have occupancy and latency of T/N + k and T + kN, where k is the total extra delay imposed by imbalance and overhead. Pipeline bubbles, unbalanced stages, and



Pipeline. Figure 1. Impact of unbalanced pipelines and pipelining overhead for as pipeline depth varies. Normalized throughput is equal to 1/(T/N + k)/T. Note that just 1% overhead (and/or imbalance) costs 10% ideal throughput in a 10-stage pipeline.

overhead all become more severe for *deep* pipelines containing many stages, and limit the useful depth of any pipeline. Figure 1 illustrates how even small values of k reduce peak throughput from the ideal as a pipeline gets deeper.

Cross-references

▶ Dataflow Execution
▶ Pipelined Query Execution

Pipelined and Independent Parallelism

▶ Inter-operator Parallelism

Pipelining

Evaggelia Pitoura
University of Ioannina, Ioannina, Greece

Definition

The query execution engine operates on an execution plan produced during query processing which typically is a physical operator graph whose edges specify the dataflow among the operators. There are two basic alternatives for evaluating the execution plan: materialization and pipelining. With materialized evaluation, the results of each operator are created (materialized) and stored in disk and then used as input for the evaluation of the next operator. With pipelining, the results of an operator are passed along as input to the next operator, before the first operator completes its execution. In general, pipelining improves the efficiency of query evaluation by reducing the number of temporary files that are produced.

Key Points

The output of query optimization is an execution plan or operator tree. The execution plan can be thought of as a dataflow graph where the nodes correspond to the physical operators and the edges represent the data flow among these physical operators. The query execution engine is responsible for the execution of this plan. The engine provides generic implementations of all physical operators that take as input one or more data streams and produce one output data stream.

In most database systems, each operator supports an *iterator* interface that allows a parent operator to get the results from its children one tuple at a time. The iterator interface supports *pipelining*, since tuples produced as output by a child operator can be fed as input to their parent in the operator tree, even before the child operator has finished its execution. The alternative would be *materialized evaluation*, in which each operator would fully complete its operation and write its results to disk before the next operator starts execution. By reducing the number of intermediate results written to disk, pipelining improves the efficiency of query evaluation. However, not all physical operators can use pipelining efficiently. For instance, merge-join cannot exploit pipelining in general, since it requires both its input relations to be sorted and it is not possible to sort a relation until all its tuples are available.

Pipelines can be executed in one of two ways: demand-driven or producer-driven. In a *demand-driven* or *demand-pull* pipeline, an operator computes the next tuple or tuples to be returned after receiving a corresponding request from its parent operator. In a *producer-driven* or *push* pipeline, operators do not wait for requests to produce tuples, instead, they generate output tuples eagerly and put them in their output buffers until the buffers become full. Demand-driven pipelining is more common, since it is easier to implement. It also allows for the whole plan to be executed by a single process or thread.

Cross-references

▶ Execution of Relational Operators
▶ Iterator
▶ Parallel Query Execution Algorithms
▶ Query Processing

Recommended Reading

1. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2): 73–170, 1993.
2. Silberschatz A., Korth H.F., and Sudarshan S. Database System Concepts, 5th edn. McGraw Hill, New York, 2005.

PiT Copy

▶ Point-in-Time Copy

Pixed Oriented Visualiyation Techniques

▶ Dense Pixel Displays

Pixel Classification

▶ Image Segmentation

Place Names

▶ Gazetteers

Place Transition Nets

▶ Petri Nets

Player

▶ Actors/Agents/Roles

Plot

▶ Graph

Plots for Qualitative Information

▶ Visualizing Categorical Data

Point-based Temporal Models

▶ Point-Stamped Temporal Models

Point-based Temporal Data

▶ Atelic Data

Point-in-Time Copy

Kenichi Wada
Hitachi Limited, Tokyo, Japan

Synonyms

Snapshot; PiT copy

Definition

A point-in-time copy is a copy of original data as it appeared at a point in time. In a conventional backup operation, users often create a PiT Copy, while an application is in quiescing, to make the PiT Copy a consistent copy of original data.

Key Points

There are two popular implementation techniques for creating PiT Copies inside a storage system: split mirror and copy on write.

Split mirror is a technique for replicating the original data at a point in time. In some implementations, a storage system replicates the original data, and when users create the PiT Copy, a storage system splits the replication. Copy on Write (CoW) is a technique for capturing data changes to storage and creating a PiT Copy after specifying the point in time. In some implementations, when users create the PiT Copy, a storage system creates its image with both the original data and the modified data.

Techniques for creating PiT Copies have a tradeoff between occupied storage capacity and performance. A PiT Copy created by split mirror occupies as much storage as the original data, but a PiT Copy by CoW usually occupies less storage. On the other hand, CoW usually has more time for accessing the copy source and creating PiT Copies than split mirror.

Cross-references

- ▶ Backup and Restore
- ▶ Logging and Recovery

Recommended Reading

1. Alain Azagury et al. Point-in-time copy: yesterday, today, and tomorrow. In Proc. of IEEE Symp. Conf. on Mass Storage Systems and Technologies, 2002.

Point-Stamped Temporal Models

David Toman
University of Waterloo, Waterloo, ON, Canada

Synonyms

Point-based temporal models

Definition

Point-stamped temporal data models associate database objects with time instants, *indivisible* elements drawn from an underlying time domain. Such an association usually indicates that the information represented by the database object in question is *valid* (i.e., believed to be true by the database) at that particular time instant. The time instant is then called the *timestamp* of the object.

Historical Background

Associating time-dependent data with time instants has been used in sciences, in particular in physics, at least since Isaac Newton's development of classical mechanics: time is commonly modeled as a two-way infinite and continuous linear order of indivisible time instants (often with distance defined as well). Similarly, in many areas of computer science, ranging from protocol specifications to program verification to model checking, discrete point-based timestamps play an essential role. In database systems (and in AI), however, the requirement of *finite* and compact representation has often lead to the use of more complex timestamps, such as intervals. This is particularly common when information about *durations* is stored in a database (in AI, such timestamps are called *fluents*). However, Chomicki [2] has shown that many of these approaches are simply compact representations of large and potentially infinite sets of time instants associated with a particular fact and that, at the conceptual level, the underlying information is often better understood as being timestamped by time instants.

Foundations

Temporal data models are typically defined as extensions of standard non-temporal data models that provide means for representation and manipulation of time-dependent data. Temporal extensions of the relational model accomplish this by relativizing *truth*, i.e., the sets of facts the database believes to be true in the modeled reality, with respect to elements of the time domain. These elements are then called the timestamps of the facts and, intuitively, specify *at which times* the associated facts are true. In the case of point-stamped temporal models, these elements are indivisible time instants drawn from an appropriate time domain.

The main ideas are outlined in a setting in which the time domain is an unbounded countably infinite linear order of *time instants*. Moreover, while the main focus is on *temporal extensions of the relational model*, most of the issues arise in other data models as well, and admit similar solutions.

Timestamps and Database Objects

The choice of timestamps is orthogonal to other choices that define the flavor and properties of the resulting temporal data model, in particular:

1. To the decision of *what database objects the timestamps are attached to*, and
2. To the decision of *how many timestamps* (per such object) are used.

First to explore is the choice of objects to be timestamped. Intuitively, timestamps should only be attached to objects that actually *capture* information in the underlying data model, indicating that the particular (piece of) information is valid for that timestamp. In the relational model these are *tuples* and their membership in *relations*. Thus, the two principal choices are as follows:

The Snapshot Model. Temporal databases in the *snapshot model* are formally defined as mappings from the time domain T to (the class of) standard relational databases with a particular fixed schema ρ. In the case of a linearly ordered time domain, such a temporal database can be viewed as a time-indexed sequence of standard relational databases, commonly called *snapshots*. Note that such a sequence is not necessarily discrete or finite: that depends on the choice of the underlying time domain. Intuitively, to capture the fact that, in a snapshot temporal database D, a relationship r holds among uninterpreted constants $a_1,...,a_k$ at time t it must be the case that $r(a_1,...,a_k)$ holds in $D(t)$, where $D(t)$ is a standard relational instance, the snapshot of D at time t; this statement is denoted by writing $r^{D(t)}(a_1,...,a_k)$.

These structures, in the area of *modal logics*, are called *Kripke structures* in which worlds are described by relational databases and where the time domain serves as the accessibility relation.

The Timestamp Model. Temporal databases in the *timestamp model* are defined in terms of temporal relations, relations whose schemas are extended with an additional attribute ranging over the time domain. This attribute is commonly referred to as the *timestamp attribute* or simply *timestamp*.

More formally, a relational symbol R is a *timestamped extension* of a symbol $r \in \rho$ if it contains all attributes of r and a single additional attribute t of the temporal sort (without loss of generality, assuming that it is always the first attribute). A *timestamp temporal database* D is then a first-order structure $D = \{R_1^D,...,R_k^D\}$ consisting of the interpretations (instances) for all the temporal extensions R_i of r_i in ρ. The instances R_i^D are called *temporal relations*. Similarly to the snapshot case, there is no restriction on the number of timestamps (i.e., the cardinality of the set of timestamps) in such instances; issues connected with the actual finite representation of these relations are addressed below. However, the relation

$$\{a_1,...,a_k : (t, a_1, ..., a_k) \in R_i^D\},$$

a snapshot of R at t, must be finite for every timestamp $t \in T$.

It is easy to see that snapshot and timestamp temporal models are simply different views of the same (isomorphic) sets of facts and thus represent the same class of temporal databases. Formally, a snapshot temporal database D corresponds to a timestamp temporal database D' (and vice versa) as follows:

$$\forall t.\forall x_1,...,x_k.r^{D(t)}(x_1,...,x_k) \;\Leftrightarrow\; R^{D'}(t, x_1,...,x_k),$$

where r and R are a relation in the schema of D and its temporal extension in the schema of D', respectively. This correspondence makes the two models interchangeable. Hence, for temporal queries formulated in query languages such as Temporal Relational Calculus (TRC) or First-order Temporal Logic (FOTL), these two models of point-stamped temporal databases can be used interchangeably.

The Parametric Model. Several temporal data models propose to use time-dependent *unary functions* as attribute values of tuples in temporal relations. These models are called *parametric models* or *attribute timestamped models*. As unary functions can be represented by binary relations (e.g., with the first attribute being the timestamp and the second attribute the value of the function for that timestamp), these models are examples of nested, non-first normal form models [4,5]. Moreover, if the implicit grouping of tuples in such relations is accidental or in the presence of appropriate keys, a first-normal form representation can be obtained by unfolding, similarly to the case of the nested relational model. The transformation is then defined by:

$$R := \{(t, f_1(t),..., f_k(t)) \mid (f_1,...,f_k) \in R^P, t \in T\},$$

where R is a timestamp relation corresponding to the parametric relation R^P. Note also that if a varying number of tuples is to be modeled in this model, *partial functions* must be used in R^P; then, however, tuples can become only partially defined for a given time instant, and therefore additional conditions must be enforced. This makes the model quite cumbersome to use, in particular when compared to the snapshot and timestamp models. Many of these difficulties originate from associating timestamps with *uninterpreted constants* that, in the relational model, *do not* carry any information on their own.

Multiple Atomic Timestamps

The second issue that arises is the question of *how many* timestamps are attached to database objects. So far single-dimensional temporal databases were considered, i.e., where each database object was associated with a single timestamp. The intuition behind such models is that *validity* of data is determined by a single time instant. In the case of the timestamp model, this translates to the fact that temporal relations were allowed only a single (often *distinguished*) temporal attribute. However, there are two natural reasons to relax this requirement and to allow multiple timestamps to be associated with database objects (i.e., multiple timestamp attributes to appear in schemas of relations, either explicitly or implicitly). There are two cases to consider:

Models with a fixed number of timestamp attributes. In many cases data modeling requires attaching several timestamps to database objects. Intuitively, a particular piece of information can be associated with, e.g., a timestamp for which the information is *valid* in the modeled world and another timestamp that states when the information is *recorded* in the database. Hence, every object has two timestamps. The resulting data model is called the bitemporal model [6] and the timestamps are called valid time and transaction time, respectively. Similarly, one can envision temporal models with three (or any fixed number) of *distinguished temporal attributes* – attributes with a predetermined interpretation – that are *common to all temporal relation schemas*. Temporal data models with an apriori *fixed* number of timestamp attributes can still be equivalently represented using the snapshot and the timestamp approaches. In the snapshot case, database instances are indexed by fixed tuples of time instants. Hence the bitemporal model can be seen as a two-dimensional plane of relational instances. The timestamp model simply adds an appropriate number of (distinguished) attributes to the timestamp extensions of relational schemes.

Models with a varying number of timestamp attributes. While most of the data modeling techniques require only a fixed number of timestamp attributes in schemas of temporal relations, it is often convenient to allow arbitrary number of timestamp attributes to be associated with a database object. This leads to allowing temporal data models without limits on the number of timestamp attributes in schemas of temporal relations. In such models, time dependencies are captured by explicit (user-defined) attributes ranging over the time domain. For a varying number of timestamps, only the *timestamp* view of the temporal data model makes sense.

Fixed-dimensional temporal data models are appealing as they commonly provide an additional built-in *interpretation* for timestamps, e.g., the valid time timestamp always states that the information is *true* in the world at that particular time. Temporal data models with varying, user-defined timestamps do not posses this additional interpretation, and the exact meaning of the timestamps depends on how the world is modeled by the associated attributes (similarly to the standard relational case). However, there is another need for models with varying and unbounded number of timestamp attributes: for temporal query languages based on temporal relational calculus, there cannot be an equivalent temporal relational algebra defined over any of the fixed-dimensional temporal data models (see the entry on Temporal Logic in Database Query Languages or [7,8] for details).

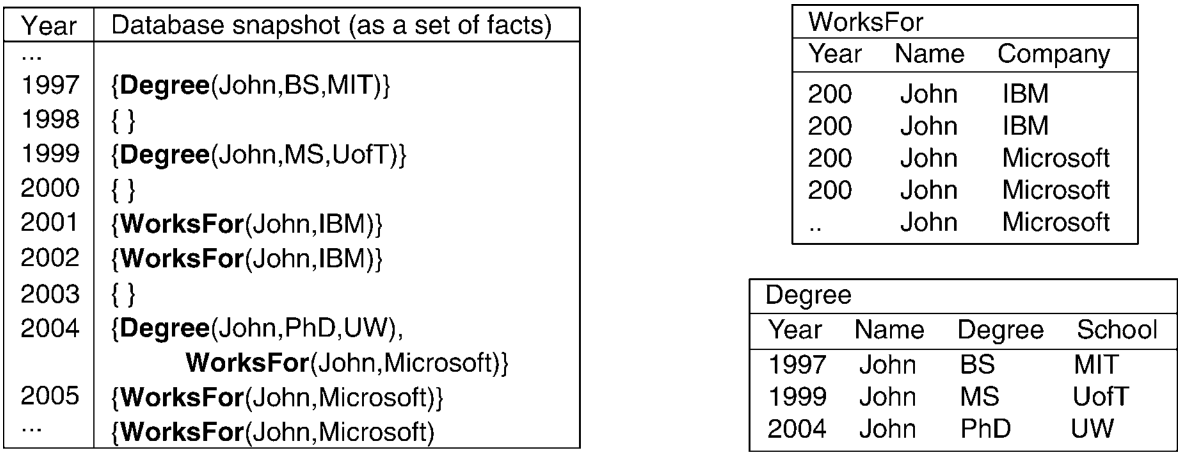
Sets of Timestamps: Compact Representation

The point-stamped temporal data models and the associated temporal query languages provide an excellent vehicle for defining a precise semantics of queries and for studying their properties. However, in *practical applications* an additional hurdle has to be overcome: a naive storage of point-stamped temporal relations, either in the timestamp or in the snapshot model, is often not possible (as the instances of the relations can be infinite, e.g., sets of time instants that represent bounded durations are infinite when a *dense time domain* is used) or impractical (the number of instants is very large). For these and other reasons, many temporal data models associate database objects with *sets* of time instants rather than with single individual time instants.

Temporal models that use complex timestamps, such as intervals, no longer appear to be point-stamped – or in the first normal form (1NF). However, Chomicki [2] has shown that in many cases these complex objects are merely compact representations of a possibly large or even infinite number of time instants associated with a particular fact. The most common approach along these lines is to attach timestamps in the form of *intervals* to facts that persist over time. For example, the temporal relation **WorksFor** in Fig. 1 can be compactly (and finitely) represented by the relation:

$$\mathbf{WorksFor}' = \{([2001, 2002], \mathrm{John}, \mathrm{IBM}),\\ ([2004, \infty], \mathrm{John}, \mathrm{Microsoft})\}.$$

Note that such an *interval encoding* is not unique and multiple snapshot equivalent representations can exist. For single-dimensional temporal relations, uniqueness



Year	Database snapshot (as a set of facts)
...	
1997	{**Degree**(John,BS,MIT)}
1998	{ }
1999	{**Degree**(John,MS,UofT)}
2000	{ }
2001	{**WorksFor**(John,IBM)}
2002	{**WorksFor**(John,IBM)}
2003	{ }
2004	{**Degree**(John,PhD,UW), **WorksFor**(John,Microsoft)}
2005	{**WorksFor**(John,Microsoft)}
...	{**WorksFor**(John,Microsoft)

WorksFor		
Year	Name	Company
200	John	IBM
200	John	IBM
200	John	Microsoft
200	John	Microsoft
..	John	Microsoft

Degree			
Year	Name	Degree	School
1997	John	BS	MIT
1999	John	MS	UofT
2004	John	PhD	UW

Point-Stamped Temporal Models. Figure 1. Instances of matching Snapshot and Timestamp Temporal Database.

of the representation can be achieved using temporal coalescing. However, it is not clear that meaningful, canonical ways of defining coalescing for multidimensional temporal relations, including bitemporal relations, exist [3].

On the other hand, the use of temporal coalescing and other set-oriented operations on interval timestamps, such as intersection of timestamps in temporal joins, indicates that the interval timestamps are indeed solely a representation tool for point-stamped relations: these operations cannot be used in a model that associates truth with the intervals themselves as there is no clear way to do so for the intervals that result from such operations, e.g., it is unclear – from a conceptual point of view – what facts should be associated with an intersection of two intervals *without* implicitly assuming that facts associated with the original two intervals hold *for all time points contained in those two intervals*, or at least for certain sub-intervals.

The choice of *intervals* to compactly represent adjacent timestamps originates from the structure of the temporal domain: intervals are the only (convex) one-dimensional sets that can be defined by (first-order) formulas in the language of linear order. For more structured time domains, however, the repertoire of encodings for sets of timestamps can be richer, e.g., using formulas describing *periodic sets*, etc., as compact timestamps.

Query Languages and Integrity Constraints

Point-stamped data models support point-based temporal query languages that are commonly based on extensions of the relational calculus (first-order logic). The two principal extensions are as follows:

1. First-order Temporal Logic: a language with an implicit access to timestamps using temporal connectives and
2. Temporal Relational Calculus: a two-sorted logic with variables and quantifiers explicitly ranging over the time domain.

These languages are the counterparts of the snapshot and timestamp models. However, unlike the data models that are equivalent in their expressiveness, the second language is strictly more expressive than the first [1,8]. Temporal integrity constraints can also be conveniently expressed in these languages [3].

Key Applications

Point-based temporal data models serve as the underlying data model for most abstract temporal query languages.

Cross-references

- ▶ Bitemporal Relation
- ▶ Constraint Databases
- ▶ Data Domain
- ▶ Duplicate Semantics
- ▶ First-Order Temporal Logic
- ▶ Key
- ▶ Nested Transaction Models
- ▶ Non First Normal Form (N1NF)
- ▶ Point-Stamped Temporal Models
- ▶ Relational Model
- ▶ Snapshot Equivalence

- ▶ Temporal Coalescing
- ▶ Temporal Data Models
- ▶ Temporal Element
- ▶ Temporal Granularity
- ▶ Temporal Integrity Constraints
- ▶ Temporal Joins
- ▶ Temporal Logic in Database Query Languages
- ▶ Temporal Query Languages
- ▶ Temporal Relational Calculus
- ▶ Time Domain
- ▶ Time Instant
- ▶ Transaction Time
- ▶ Valid Time

Recommended Reading

1. Abiteboul S., Herr L., and Van den Bussche J. Temporal versus first-order logic to query temporal databases. In Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1996, pp. 49–57.
2. Chomicki J. Temporal query languages: a survey. In Proc. 1st Int. Conf. Temporal Logic, 1994, pp. 506–534.
3. Chomicki J. and Toman D. Temporal Databases. In Handbook of Temporal Reasoning in Artificial Intelligence. M. Fischer, D. Gabbay, and L. Villa (eds.). Elsevier, London, UK, 2005, pp. 429–467.
4. Clifford J., Croker A., and Tuzhilin A. On the completeness of query languages for grouped and ungrouped historical data models. In Temporal Databases: Theory, Design, and Implementation, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R.T. Snodgrass (eds.). Benjamin/Cummings, Redwood City, CA, USA, 1993, pp. 496–533.
5. Clifford J., Croker A., and Tuzhilin A. On completeness of historical relational query languages. ACM Trans. Database Syst., 19(1):64–116, 1994.
6. Jensen C.S., Soo M.D., and Snodgrass R.T. Unification of temporal data models. In Proc. 9th Int. Conf. on Data Engineering, 1993.
7. Toman D. On incompleteness of multi-dimensional first-order temporal logics. In Int. Symp. on Temporal Representation and Reasoning and Int. Conf. on Temporal Logic., 2003, pp. 99–106.
8. Toman D. and Niwinski D. First-order queries over temporal databases inexpressible in temporal logic. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996.

Point-versus Period-based Semantics

- ▶ Telic Distinction in Temporal Databases

Polyhedron

- ▶ Simplicial Complex

Polytransactions

George Karabatis
University of Maryland, Baltimore County (UMBC),
Baltimore, MD, USA

Definition

A polytransaction $T^{+}$is a transitive closure of a transaction T submitted to an Interdependent Data Management System (a type of a multidatabase system which enforces dependencies among related data objects). The transitive closure is computed with respect to an Interdatabase Dependency Schema (IDS) consisting of a collection of data dependency descriptors (D^3s), which specify the dependencies between data objects.

Key Points

Data objects in multiple (possibly distributed and/or heterogeneous) systems which form dependencies among them are called *interdependent data*. These dependencies are identified through *data dependency descriptors* (D^3s). A D^3 specifies the relationships between source and target objects including how much inconsistency can be tolerated between them before it is necessary to restore it, and the actions to take to maintain consistency when it reaches intolerable levels. All these D^3s together form an *Interdatabase Dependency Schema* (IDS) [3].

When a source object in a D^3 is updated, the consistency between itself and the target object in the same D^3 may be violated beyond allowable levels. Then, a system-generated transaction updates the target object to maintain consistency. The updated target object may participate as source in another D^3, therefore a series of related transactions may execute to maintain consistency. These related transactions form a transaction tree called a *polytransaction* [3,1,2].

A polytransaction tree contains nodes (corresponding to its component transactions), and edges (identifying the "execution mode" between the parent and children transactions) and is determined as follows. For each D^3_ε IDS containing a source object updated by T, and in need to restore consistency with its target object, a new node is added corresponding to a new transaction T' (child of T). The operations of T' which restore consistency are the actions specified in the D^3.

Execution modes: A child transaction is *coupled* if the parent transaction must wait until the child transaction completes before proceeding further. It is *decoupled* otherwise. Also, a coupled transaction is *vital* (the parent transaction must fail if the child fails), or *nonvital* (the parent transaction survives the failure of a child).

Cross-references

- ▶ Database Dependencies
- ▶ Database Management System
- ▶ Distributed Transaction Management
- ▶ Extended Transaction Models
- ▶ Inconsistent Databases
- ▶ Transaction
- ▶ Transaction Management
- ▶ Transaction Manager

Recommended Reading

1. Karabatis G., Rusinkiewicz M., and Sheth A. Correctness and enforcement of multidatabase interdependencies, Advanced Database Systems, LNCS, vol. 759, Springer-Verlag, NY, 1993, pp. 337–358.
2. Karabatis G., Rusinkiewicz M., and Sheth A. Interdependent database systems. In Management of Heterogeneous and Autonomous Database Systems. A. Elmagarmid, M. Rusinkieuicz, and A. Sheth (eds.), Morgan-Kaufmann, San Francisco, CA, 1999, pp. 217–252.
3. Rusinkiewicz M., Sheth A., and Karabatis G. Specifying interdatabase dependencies in a multidatabase environment, IEEE Computer, 24(12): December 1991, pp. 46–53.

Port Binding

▶ Storage Security

Position Snapping

▶ Map Matching

Positive Infinity

▶ Forever

Positive Predictive Value

▶ Precision and Recall

Positive Relational Algebra

Cristina Sirangelo
University of Edinburgh, Edinburgh, UK

Synonyms

SPJRU-Algebra; SPCU-Algebra

Definition

Positive relational algebra is the fragment of relational algebra which excludes the difference operator. Relational algebra queries are expressions defining mappings from database instances of an input database schema to an output relation. In particular a positive relational algebra query over a database schema τ is one of the following expressions, each one with an associated set of attributes:

- A constant relation over a set of attributes U is a positive relational algebra query with associated set of attributes U;
- If $R(U)$ is a relation schema in τ, the relation symbol R is a positive relational algebra query with associated set of attributes U;
- If Q and Q' are positive relational algebra queries with sets of attributes U and U' respectively, the following are positive relational algebra queries:
  - The *selection* $\sigma_{A=B}(Q)$ or $\sigma_{A=c}(Q)$, with set of attributes U, where A and B are attributes in U, and c is a constant value;
  - The *projection* $\pi_X(Q)$, with set of attributes X, where $X \subseteq U$;
  - The *natural join* $Q \bowtie Q'$, with set of attributes $U \cup U'$;
  - The *renaming* $\rho_{U \to W}(Q)$, with set of attributes W;
  - The *union* $Q \cup Q'$, in the case that $U = U'$, with associated set of attributes U.

The semantics of a positive relational algebra query on a database instance I of schema τ is a relation instance defined as follows: the semantics of a constant relation is the relation itself; the semantics of a relation symbol R of τ is the value of R in I; the semantics of selection, projection, natural join, renaming and union expressions above, is defined according to the semantics of the corresponding operator on inputs given by the semantics of Q and Q'.

Key Points

The basic operators of the positive relational algebra are a non-redundant set: by removing any of these operators the set of expressible query mappings would be reduced. Moreover they allow the simulation of other operators. Among these, the intersection $R_1 \cap R_2$ of two relations over the same set of attributes can be simulated as $R_1 \bowtie R_2$; the generalized selection whose condition is a positive boolean combination of equality atoms can be simulated using composition and union of primitive selection operators; consequently also the theta-join, with the same restriction on the join condition, and the equijoin can be simulated.

An example of positive relational algebra query over a database schema consisting of relation schemas *Students*(*student-number, student-name*) and *Exams* (*course-number, student-number, grade*), is the expression $\pi_{student-name}(Students \bowtie \sigma_{grade\ =\ A}(Exams))$. It returns the names of the students that have passed at least one exam with grade A.

The positive relational algebra is equivalent (in that it expresses the same query mappings) to other query languages such as *unions of conjunctive queries* in safe relational calculus, and *nonrecursive datalog* with one target predicate.

In the absence of attribute names, the basic operators of positive relational algebra are $\{\sigma, \pi, \times, \cup\}$. The positive relational algebra without names turns out to be equivalent to the positive relational algebra with names and thus equivalent to the other above mentioned query paradigms.

Cross-references

- ▶ Difference
- ▶ Join
- ▶ Nonrecursive Datalog
- ▶ Projection
- ▶ Relation
- ▶ Relational Algebra
- ▶ Relational Calculus
- ▶ Renaming
- ▶ Selection
- ▶ Union

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases. Addison-Wesley, Reading, MA, 1995.

Possible Answers

Gösta Grahne
Concordia University, Montreal, QC, Canada

Synonyms

Maybe answer; Credulous reasoning; Consistent facts

Definition

In an incomplete database, which is a set of complete databases (possible worlds), the possible answer to a query is the set of tuples that are in the answer to the query, when posed on *some* possible world. The dual of the possible answer is the *certain* answer, which consists of all tuples true in the the answer to the query when posed simulatenously on *all* possible worlds.

Key Points

For more information on the certain and possible answers, as well as on various (partial) orders on incomplete databases, see [1,2].

Cross-references

- ▶ Certain (and Possible) Answers
- ▶ Incomplete Information
- ▶ Naive Tables

Recommended Reading

1. Grahne G. The Problem of Incomplete Information in Relational Databases. Springer, Berlin, 1991.
2. Libkin L. Aspects of Partial Information in Databases. PhD Thesis, University of Pennsylvania, 1994.

Postings File

▶ Inverted Files

Post-Randomization Method

▶ PRAM

PRAM

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona, Catalonia

Synonyms

Post-randomization method

Definition

The Post-RAndomization Method (PRAM) is a probabilistic, perturbative masking method for disclosure protection of categorical microdata. In the masked file, the scores on some categorical attributes for certain records in the original file are changed to a different score according to a prescribed probability mechanism, namely a Markov matrix. The Markov approach makes PRAM very general, because it encompasses noise addition, data suppression and data recoding.

Key Points

The PRAM matrix contains a row for each possible value of each attribute to be protected. This rules out using the method for continuous data. PRAM was invented by Gouweleeuw et al. [1]. The information loss and disclosure risk in data masked with PRAM largely depend on the choice of the Markov matrix and are still (open) research topics [2].

Cross-references

▶ Inference Control in Statistical Databases
▶ Microdata
▶ SDC Score

Recommended Reading

1. Gouweleeuw J.M., Kooiman P., Willenborg L.C.R.J., and DeWolf P.-P. Post randomisation for statistical disclosure control: Theory and implementation, 1997. Research Paper No. 9731 (Voorburg: Statistics Netherlands).
2. de Wolf P.-P. Risk, utility and PRAM. In J. Domingo-Ferrer and L. Franconi (eds.). Privacy in Statistical Databases LNCS, vol. 4302, 2006, pp. 189–204.

Precision

Ethan Zhang[1,2], Yi Zhang[2]
[1]University of California-Santa Cruz, Santa Cruz, CA, USA
[2]Yahoo! Inc., Santa Clara, CA, USA

Definition

Precision measures the accuracy of an information retrieval (IR) system. More precisely, precision is the fraction of retrieved documents that are relevant. Consider a test document collection and an information need Q. Let R be the set of documents in the collection that are relevant to Q. Assume an IR system processes the information need Q and retrieves a document set A. Let $|R_a|$ denote the number of documents that are in both R and A. Further let $|R|$ and $|A|$ be the numbers of documents in R and A, respectively. The *precision* of the IR system for Q is defined as $P = |R_a| / |A|$.

Key Points

Precision and recall are the most frequently used and basic retrieval performance measures. Many other standard performance metrics are based on the two concepts.

Cross-references

▶ Average Precision
▶ F-Measure
▶ 11-Point Precision-Recall Curve
▶ Precision-Oriented Effectiveness
▶ Recall
▶ Standard Effectiveness Measures

Precision and Recall

Ben Carterette
University of Massachusetts Amherst, Amherst, MA, USA

Synonyms

Positive predictive value; Sensitivity; False negative rate

Definition

Recall measures the ability of a search engine or retrieval system to locate relevant material in its index. *Precision* measures its ability to not rank nonrelevant material. With everything above rank cut-off n considered "retrieved" and everything below considered "not retrieved," precision and recall can be stated mathematically as:

$$\text{precision} = \frac{|\text{retrieved \& relevant at rank } n|}{|\text{retrieved at rank } n|}$$

$$\text{recall} = \frac{|\text{retrieved \& relevant at rank } n|}{|\text{relevant}|}$$

Key Points

Precision and recall are the traditional metrics for retrieval system performance evaluation, and nearly all other performance measures can be seen as either precision-based, recall-based, or a combination of the two.

There is a trade-off between precision and recall: as more is done to bring more relevant results into a ranking, increasing recall, inevitably nonrelevavnt results will be captured as well, decreasing precision. This can be seen in precision-recall curves, which plot precision against recall while varying rank cut-off n.

		Condition	
		True	False
Prediction	True	True Positive	False Positive
	False	False Negative	True Negative

Precision and recall are not limited to information retrieval; they can be defined more formally on any 2 × 2 contingency table. For the table precision is True Positives divided by True Positives + False Positives and recall is True Positives divided by True Positives + False Negatives. In medical literature, precision is known as *positive predictive value* and recall as *sensitivity*; in the statistics literature, recall is equivalent to the *power* (or 1 − Type II error rate) of a hypothesis test (precision has no analogue).

Cross-references

► Average Precision

► Precision at n

► Recall

Recommended Reading

1. van Rijsbergen C.J. Information Retrieval. Butterworths, London, UK, 1979.

Precision at n

Nick Craswell
Microsoft Research Cambridge, Cambridge, UK

Synonyms

P@n

Definition

In an information retrieval system that retrieves a ranked list, the top-n documents are the first n in the ranking. Precision at n is the proportion of the top-n documents that are relevant.

Key Points

If r relevant documents have been retrieved at rank n, then:

$$\text{Precision at n} = \frac{r}{n}$$

The value of n can be chosen based on an assumption about how many documents the user will view. In Web search a results page typically contains ten results, so n = 10 is a natural choice. However, not all users will use the scrollbar and look at the full top ten list. In a typical setup the user may only see the first five results before scrolling, suggesting Precision at 5 as a measure of the initial set seen by users. It is the document at rank 1 that gets most user attention, because this is the document that users view first, suggesting the use of Precision at 1 (which is equivalent to Success at 1).

It is possible to calculate precision at a later cutoff, although precision gives equal weight to every result in the list. For example, when calculating precision at 1,000 the 1,000th document is as important as the 1st, whereas users of a ranked retrieval system are likely to consider the 1st document most important. Other information retrieval measures place a greater emphasis on early ranks, such as Mean Average Precision and Mean Reciprocal Rank.

In set-based retrieval, where there is no ranking, the precision of a retrieved set of size n can still be calculated, as r/n.

In order to calculate precision at n it is only necessary to obtain relevance judgments for the top-n documents, unlike recall, which can only be measured if the complete set of relevant documents has been identified.

Cross-references

- ▶ Average Precision
- ▶ Precision-Oriented Effectiveness Measures
- ▶ Recall
- ▶ Success at n

Precision-Oriented Effectiveness Measures

NICK CRASWELL
Microsoft Research Cambridge, Cambridge, UK

Definition

Precision-oriented evaluation in information retrieval considers the relevance of the top *n* search results, for small *n* and using a set of relevance judgments that need not be complete. Such "shallow" evaluation is consistent with a user who only cares about the top-ranked documents. Relaxing the requirement of identifying all relevant documents for every query means that certain measures, such as recall at *n*, cannot be applied. However, it also allows evaluation on a very large corpus, where employing human relevance assessors to find the complete relevant set for each query would be too expensive. Both aspects of precision-oriented evaluation, the shallow viewing of results and the large corpus, are associated with Web search, where search results are typically a top-10 and the corpus may contain tens of billions of documents.

Historical Background

The Cranfield II experiments in 1963 were a landmark effort in information retrieval evaluation [3]. A test collection comprising 1,440 documents, 225 queries and relevance judgments was created. Because the document set was small, it was possible to judge the relevance of every document for each query. Using complete relevance judgments it is possible to evaluate precision, the proportion of retrieved documents that are relevant, and recall, the proportion of relevant documents that are retrieved. The test collection is reusable, in that it is possible to run a new retrieval experiment using existing queries and relevance judgments, without encountering any unjudged document in the search results.

Judging all documents for every query does not scale well with corpus size, and becomes untenable even with tens of thousands of documents. The best-known solution to this problem is the pooling method, as introduced in the Text Retrieval Conference (TREC) in 1991 [5]. Under pooling, the top-n documents are merged from a large number of systems. This pool of documents does not necessarily contain all relevant documents. However, if a sufficient variety of top-n lists are merged, for a large enough n, the pool will contain the majority of the relevant documents. After judging the pool, the set of relevance judgments can be thought of as "sufficiently complete" to be reusable. In other words, a new retrieval experiment may retrieve unjudged documents, but these can be assumed to be irrelevant since almost all the relevant documents have already been identified.

Most TREC experiments have resulted in test collections where the pools are sufficiently complete for reuse. However, for the largest TREC corpora, it is unlikely that the full relevant set has been identified, or could be identified due to practical limitations in the judging resources [4]. In such cases it is possible to judge with shallow pools, for example judging the top 20 documents and using a precision-oriented measure. Since the full relevant set has not been discovered, it is not straightforward to measure recall, or use any measure that relies on knowing the size of the relevant set. The test collection may not be reusable.

Foundations

To understand precision-oriented evaluation, it is useful to revisit the fundamental measures of information retrieval, precision and recall. If there are *R* known relevant documents, and *r* of them have been retrieved at rank n, then:

$$\text{Precision at n} = \frac{\mathrm{r}}{\mathrm{n}}$$

$$\text{Recall at n} = \frac{\mathrm{r}}{\mathrm{R}}$$

For a given cutoff n, these are just different ways of normalizing r. If r increases, because a new retrieval system retrieves a better set of *n* documents, both precision and recall improve.

One aspect of precision-oriented evaluation is where not all relevant documents have been identified, so R is unknown. In such a case it is not straightforward to estimate recall, which depends on knowledge of R. However, if the top n documents have been judged, it is possible to accurately measure precision.

Another aspect of precision-oriented evaluation is shallow evaluation. At an early cutoff (small n) few relevant documents will have been retrieved (small r), but precision may still be quite high. By contrast, recall might be low, particularly if R is much higher than r and n. In a standard precision-recall curve, which shows precision and recall at multiple cutoffs, precision-oriented evaluation considers the left-hand end of the curve. Precision-oriented evaluation models a user who does not need to see all relevant documents and is impatient, unwilling to look deep into the ranking.

Some experiments have both shallow evaluation and incomplete judgments, for example the TREC Very Large Collection Track [4] judged with a pool depth of 20, to measure Precision at 20. Metrics such as Precision at n, Success at n, Mean Reciprocal Rank and Average Precision at n have been applied in such settings. More details about these metrics can be found from the corresponding entries in this encyclopedia.

In other experiments, judgments may be incomplete or evaluation shallow, but not both. In those cases, it is not clear that the evaluation should be called precision-oriented. It is possible to evaluate with a recall-oriented measure using incomplete judgments, via measures such as bpref and inferred AP [1,6]. It is also possible to perform shallow evaluation with complete judgments, particularly in cases with very few relevant documents such as known item search. In a setting where R tends to be between 1 and 10, a metric such as R-Precision could be thought of as precision-oriented.

Key Applications

Web search is probably the most well-known application where precision-oriented evaluation is used. Users do not often look deep into the results list, so evaluation that concentrates on shallow ranks is appropriate. In addition, the document collection is so large, that it is difficult to be sure that all good answers for a query have been identified. The exception is where a user has a very focused need, for example navigational search, where only a single document is required. Even then the web is volatile, with documents being created, modified and deleted without warning, so it is necessary to perform careful maintenance of a query set. For queries with multiple correct answers, it is usual to calculate precision-oriented measures, and very difficult to estimate recall.

Data Sets

Data sets for precision-oriented evaluation, and Information Retrieval experimentation in general, are available from the National Institute of Standards and Technology, in the Text Retrieval Conference (TREC) series [5]. TREC experiments typically involve hundreds of thousands of documents or more, and 50 or more query topics.

Cross-references

- ▶ Average Precision at n
- ▶ Bpref
- ▶ Mean Reciprocal Rank
- ▶ Precision
- ▶ Precision at n
- ▶ R-Precision
- ▶ Standard Effectiveness Measures
- ▶ Success at n

Recommended Reading

1. Buckley C. and Voorhees E.M. Retrieval evaluation with incomplete information. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 25–32.
2. Clarke C.L.A., Scholer F., and Soboroff I. The TREC 2005 terabyte track. In Proc. The 5th Text Retrieval Conference, 2005.
3. Cleverdon C. The Cranfield Tests on Index Language Devices. Readings in Information Retrieval. Morgan Kaufmann, San Fransisco, CA, 1997, pp. 47–59.
4. Hawking D. and Craswell N. The very large collection and web tracks. In TREC Experiment and Evaluation in Information Retrieval, E. Voorhees, D. Harman (eds.). MIT Press, Cambridge, MA, 2005, pp. 199–231.
5. Voorhees E.M. and Harman D.K. TREC: Experiment and Evaluation in Information Retrieval. MIT Press, Cambridge, MA, 2005.
6. Yilmaz E. and Aslam J.A. Estimating average precision with incomplete and imperfect judgments. In Proc. Int. Conf. on Information and Knowlege Management, 2006, pp. 102–111.

Predicate Calculus

- ▶ First-Order Logic: Semantics
- ▶ First-Order Logic: Syntax

Predicate Logic

- ▶ First-Order Logic: Semantics
- ▶ First-Order Logic: Syntax

Prediction of Event Occurrence

► Event Prediction

Prediction Regarding Future Events

► Event Prediction

Prefix Tree

► Trie

Presenting Structured Text Retrieval Results

Jaap Kamps
University of Amsterdam, Amsterdam,
The Netherlands

Synonyms

Structured Text Retrieval Tasks

Definition

Presenting structured text retrieval results refers to the fact that, in structured text retrieval, results are not independent and a judgment on their relevance needs to take their presentation into account. For example, HTML/XML/SGML documents contain a range of nested sub-trees that are fully contained in their ancestor elements. As a result, structured text retrieval should make explicit the assumptions on how the retrieval results are to be presented. Four of the main assumptions to be addressed are the following. First, the *unit of retrieval* assumption: is there a designated retrieval unit (such as the document or root node of the structured document) or can every sub-tree be retrieved in principle? Second, the *overlap* assumption: may retrieval results contain text or content already part of other retrieval results (such as a full article and one of its individual paragraphs)? Third, the *context* assumption: can results from the same structured document be interleaved with results from other structured documents? Fourth, the *display* assumption: is a retrieval result (say a document sub-tree corresponding to a paragraph) presented as an autonomous unit of text, or as an entry-point within a structured document?

Historical Background

Although similar considerations play an important role in the design of user interfaces (see, for example, [6]), this entry will focus on the underlying principles of the different structured text retrieval tasks. Structured text retrieval dates back, at least, to the early days of passage retrieval [14]. Early passage retrieval approaches have been using either the document structure (sentences, paragraphs, sections, etc.), or arbitrary text windows of fixed length. The early experimental results primarily confirmed the effectiveness of passage-level evidence for boosting document retrieval. The use of document structure derived from SGML mark-up was pioneered by [20], studying adhoc SGML element retrieval. Probabilistic indexing approaches for databases have been studied even earlier [4], allowing to rank results based on vague queries. Adhoc XML element retrieval and best entry point retrieval was studied in the Focus project [3,10].

The main thrust in recent years is the initiative for the evaluation of XML retrieval [7]. The retrieval task descriptions heavily evolved during the different years. Initially, in 2002, INEX studied adhoc XML element retrieval for keyword (Content-Only) and structured (Content-And-Structure) queries with the goal to "retrieve the most specific relevant document components" [5, p. 2]. This generic adhoc XML element retrieval task was continued at INEX 2003 [9, p. 200] and at INEX 2004 [12, p. 237], asking for "components that are most specific and most exhaustive with respect to the topic of request." Ongoing discussion, and vivid disagreement, on the interpretation of generic adhoc XML element retrieval task prompted the introduction of three different retrieval strategies at INEX 2005 [11, pp. 385–386]: *Thorough* aims to find all highly exhaustive and specific elements (roughly corresponding to the earlier INEX task); *Focussed* aims to find the most specific and exhaustive element in path (no overlapping results); and *Fetch and browse* aims to first identify relevant articles, and then to find the most specific and exhaustive elements within the fetched articles (results grouped by article). These different adhoc XML element retrieval tasks have been continued and further explicated at INEX 2006 [1], with the Fetch and browse task refined to: *Relevant in Context* aims to retrieve a set of non-overlapping relevant elements per article; and *Best in Context* aims to retrieve, per article, a single best entry point to read its relevant content. At INEX 2007 three tasks are continued: Focused, Relevant in Context, and Best in Context, but liberalized to arbitrary passages [2].

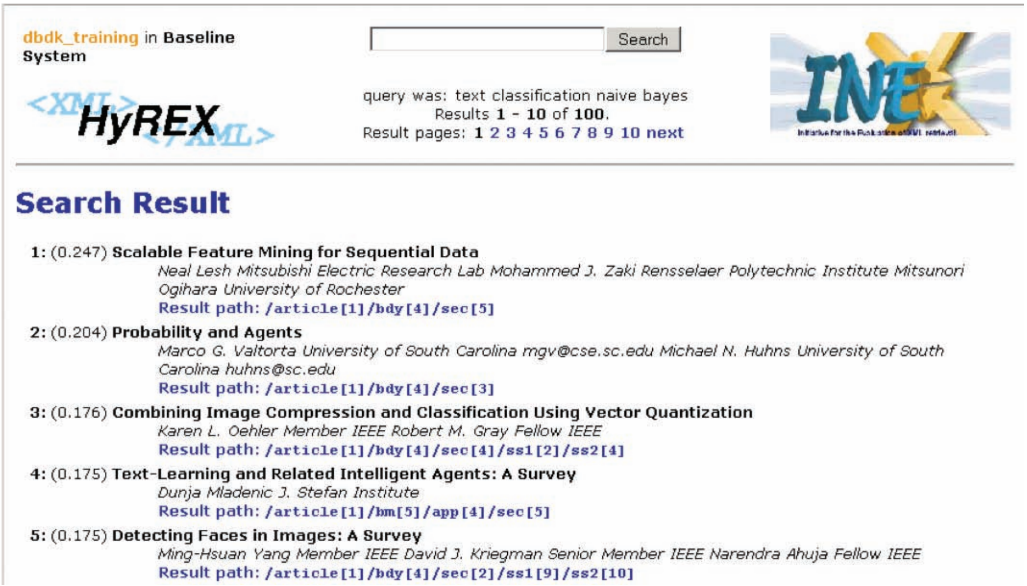
Foundations

The way in which retrieval results are presented to users, is always a crucial factor determining the success or failure of an operational retrieval system. However, within the Cranfield/TREC tradition of evaluating document retrieval systems it is unproblematic to abstract away from presentation issues and analyze retrieval effectiveness by regarding retrieved documents as atomic and independent results. In structured text retrieval, the situation is different, and there is a need to make explicit some of the assumptions underlying the retrieval task since these have an impact on what is regarded as a "relevant" retrieval result.

First, the *unit of retrieval* assumption: is there a designated retrieval unit (such as the document or root node of the structured document) or can every sub-tree be retrieved in principle? Rather than treating documents as atomic, structured documents have internal document structure that allows any logical unit of them to be retrieved. For example, in case of a textual document where the layout structure is marked up, it is possible to retrieve sections, paragraphs, or still the whole article if its completely devoted to the topic of request. Figure 1 contains a screen-shot of a XML element retrieval system that retrieves a ranked list of XML elements.

Second, the *overlap* assumption: may retrieval results contain text or content already part of other retrieval results (such as a full article and one of its individual paragraphs)? Interactive experiments at INEX 2004 [17] clearly revealed that test persons disliked a ranked list of element results that overlap in whole or part in their content. Hence, if the retrieval tasks should reflect a scenario in which the ranked elements are directly displayed to an end-user, retrieval results should be disjoint.

Third, the *context* assumption: can results from the same structured document be interleaved with results from other structured documents? A further finding of the interactive experiments at INEX 2004 [17] is that test persons prefer results from the same document be grouped together. Figure 2 contains a screen-shot of a XML element retrieval system that retrieves XML elements displayed in document order in their article's context.



dbdk_training in Baseline System

query was: text classification naive bayes
Results 1 - 10 of 100.
Result pages: 1 2 3 4 5 6 7 8 9 10 next

Search Result

1: (0.247) Scalable Feature Mining for Sequential Data
Neal Lesh Mitsubishi Electric Research Lab Mohammed J. Zaki Rensselaer Polytechnic Institute Mitsunori Ogihara University of Rochester
Result path: /article[1]/bdy[4]/sec[5]

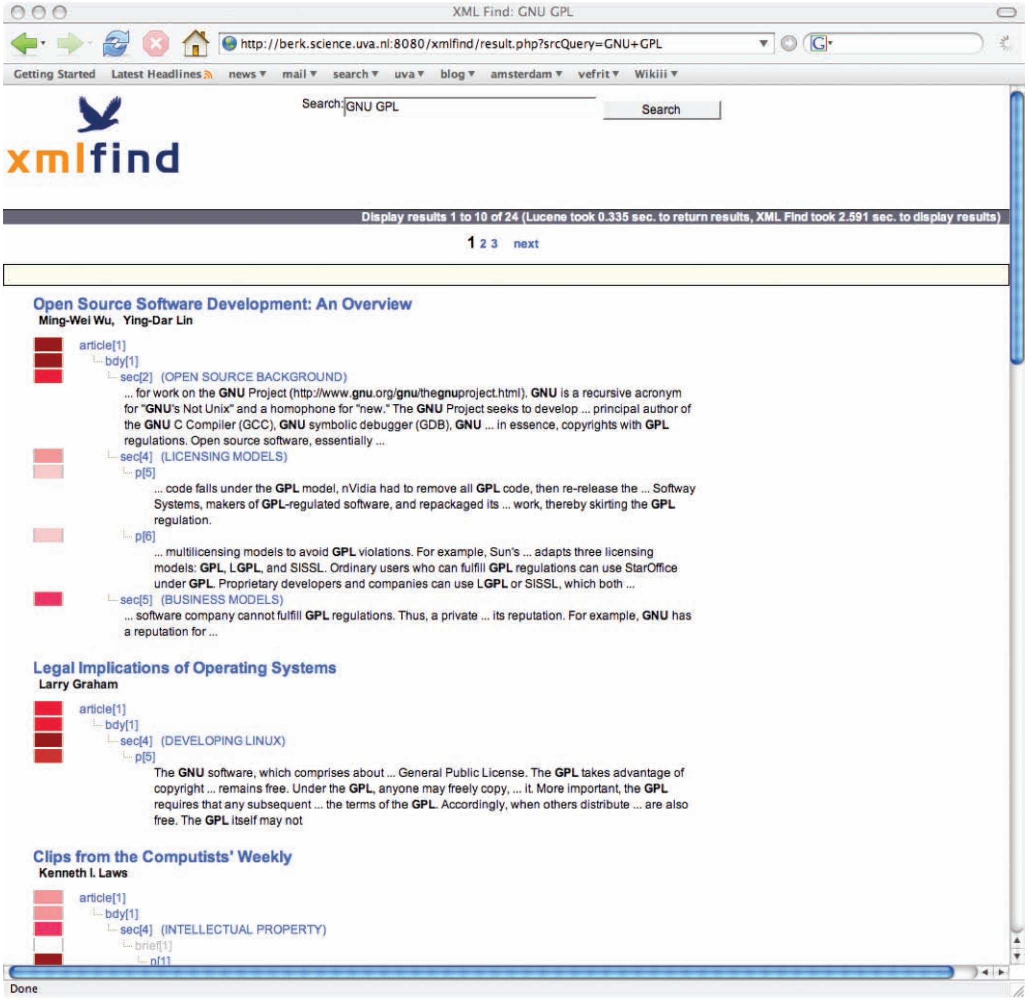
2: (0.204) Probability and Agents
Marco G. Valtorta University of South Carolina mgv@cse.sc.edu Michael N. Huhns University of South Carolina huhns@sc.edu
Result path: /article[1]/bdy[4]/sec[3]

3: (0.176) Combining Image Compression and Classification Using Vector Quantization
Karen L. Oehler Member IEEE Robert M. Gray Fellow IEEE
Result path: /article[1]/bdy[4]/sec[4]/ss1[2]/ss2[4]

4: (0.175) Text-Learning and Related Intelligent Agents: A Survey
Dunja Mladenic J. Stefan Institute
Result path: /article[1]/bm[5]/app[4]/sec[5]

5: (0.175) Detecting Faces in Images: A Survey
Ming-Hsuan Yang Member IEEE David J. Kriegman Senior Member IEEE Narendra Ahuja Fellow IEEE
Result path: /article[1]/bdy[4]/sec[2]/ss1[9]/ss2[10]

Presenting Structured Text Retrieval Results. Figure 1. Displaying structured text retrieval results as a ranked list of elements (reproduced from [13]).



http://berk.science.uva.nl:8080/xmlfind/result.php?srcQuery=GNU+GPL

Search: GNU GPL

Display results 1 to 10 of 24 (Lucene took 0.335 sec. to return results, XML Find took 2.591 sec. to display results)

1 2 3 next

Open Source Software Development: An Overview
Ming-Wei Wu, Ying-Dar Lin

article[1]
bdy[1]
sec[2] (OPEN SOURCE BACKGROUND)
... for work on the **GNU** Project (http://www.**gnu**.org/**gnu**/the**gnu**project.html). **GNU** is a recursive acronym for "**GNU**'s Not Unix" and a homophone for "new." The **GNU** Project seeks to develop ... principal author of the **GNU** C Compiler (GCC), **GNU** symbolic debugger (GDB), **GNU** ... in essence, copyrights with **GPL** regulations. Open source software, essentially ...
sec[4] (LICENSING MODELS)
p[5]
... code falls under the **GPL** model, nVidia had to remove all **GPL** code, then re-release the ... Softway Systems, makers of **GPL**-regulated software, and repackaged its ... work, thereby skirting the **GPL** regulation.
p[6]
... multilicensing models to avoid **GPL** violations. For example, Sun's ... adapts three licensing models: **GPL**, L**GPL**, and SISSL. Ordinary users who can fulfill **GPL** regulations can use StarOffice under **GPL**. Proprietary developers and companies can use L**GPL** or SISSL, which both ...
sec[5] (BUSINESS MODELS)
... software company cannot fulfill **GPL** regulations. Thus, a private ... its reputation. For example, **GNU** has a reputation for ...

Legal Implications of Operating Systems
Larry Graham

article[1]
bdy[1]
sec[4] (DEVELOPING LINUX)
p[5]
The **GNU** software, which comprises about ... General Public License. The **GPL** takes advantage of copyright ... remains free. Under the **GPL**, anyone may freely copy, ... it. More important, the **GPL** requires that any subsequent ... the terms of the **GPL**. Accordingly, when others distribute ... are also free. The **GPL** itself may not

Clips from the Computists' Weekly
Kenneth I. Laws

article[1]
bdy[1]
sec[4] (INTELLECTUAL PROPERTY)
brief[1]
p[1]

Presenting Structured Text Retrieval Results. Figure 2. Displaying structured text retrieval results within article context (reproduced from [15]).

Fourth, the *display* assumption: is a retrieval result (say a document sub-tree corresponding to a paragraph) presented as an autonomous unit of text, or as an entry-point within a structured document? A decision on the relevance of a particular document component crucially depends on whether it will be presented as an isolated excerpt, or within its original context. In the first case, the component should be fully self-contained: it should not only contain the relevant information (say, for example, a description of an algorithm) but also establish that this information is, indeed, satisfying the topic of request (for example, that the algorithm is the fastest way to lexicographically sort a list, if that were the topic of request). This is related to linguistics, where there is a common distinction between the context (or topic/theme: that what is being talked about), and the information (or comment/rheme/focus: that what is being said). If results are to be presented in their document context, the link to the topic of request can be taken for granted and only the sought information can be regarded as relevant. If results are to be presented out of context, both the information and its relation to the topic of request are needed to establish the relevance of a document component.

Table 1 shows how the different structured text retrieval tasks are based on different underlying assumptions. For traditional document or article retrieval, there is a fixed unit of retrieval and assumptions on overlap, context, or display do not apply. For the generic adhoc element retrieval task (INEX 2002–2004) or Thorough (INEX 2005–2006), any document

Presenting Structured Text Retrieval Results. Table 1. Structured text retrieval tasks

	Unit of retrieval	Overlap	Context	Display
Article retrieval	Whole article	–	–	–
Thorough	Arbitrary element	Allowed	Scattered	Elements
Focussed/Focused	Arbitrary element	Non-overlapping	Scattered	Elements/Passages
Fetch and browse	Arbitrary element	Allowed	List per article	Elements
Relevant in context	Arbitrary element	Non-overlapping	Set per article	Elements/Passages
Best in context	Arbitrary element	Non-overlapping	One result per article	Starting point

component can be retrieved, and there are no restrictions on overlap, context, or display. Basically, the task is system-biased, reflecting the ability of the retrieval engine to estimate the relevance of individual document components, for example for further processing methods. For Focussed/Focused (INEX 2005–2007), a ranked list of non-overlapping document components is asked for, with no restrictions on context or display. This task reflects a scenario where a ranked-list of document components is directly presented to the searcher. For Fetch and browse (INEX 2005), retrieval results from the same structured document need to be returned consecutive, with no restriction on overlap or display. This results in a tasks resembling on the one hand traditional document retrieval, whilst on the other hand providing deep-linking to relevant document components. The same holds for Relevant in context (INEX 2006–2007), where there is an unranked set of now non-overlapping elements per article, reflecting results to be presented in document order. Finally, Best in context (INEX 2006–2007) explicitly asks for a single best entry point into the article (so non-overlapping and non-scattered articles by definition). This scenario captures a "relative" notion of relevance, where users desire access to the best information, rather than all relevant information.

These different retrieval tasks lead to different evaluations of what systems and techniques are effective for structured text retrieval. Although these tasks are not unrelated, for example, the generic Thorough task (capturing the ability of a system to estimate the relevance of an element) can be use as input for further processing for the other tasks, each of these different retrieval tasks is capturing a different aspect of structured text. The retrieval tasks bring in elements from the task context in which they are to be applied, either in a end-user setting or system setting. As a result, the richer descriptions of the task's context and underlying assumptions are resonating more closely with actual real-world applications [19]. Bringing task-specific elements into information retrieval benchmark testing has been identified as one of the main research directions for further enhancing information access in general [18].

Key Applications

Structured text retrieval has the potential to improve information access by giving more direct access to the relevant information inside documents. As [14, p. 49] put it:

- Large collections of full-text documents are now commonly used in automated information retrieval. When the stored document texts are long, the retrieval of complete documents may not be in the users' best interest. In such circumstances, efficient and effective retrieval results may be obtained by using passage retrieval strategies designed to retrieve text excerpts of varying size in response to statements of user interest.

Structured document retrieval is becoming increasingly important in all areas of information retrieval, the application to full-text book searching is obvious and such commercial systems already exist [19].

Future Directions

Improving information access by formulating retrieval tasks that capture interesting aspects of real-world structured text searching is an ongoing open problem. There has been a series of workshops addressing open problems, including real-world applications, the unit of retrieval, tasks and measures, and the problem of overlap [18].

The traditional picture of IR takes as input a document collection and a query, and gives as output a ranked list of documents. In the retrieval task, there no distinction between the hit list (communicating the ranked list) and the actual result documents. Where structured document retrieval is going beyond a linear ranked list of results, at least conceptually, interesting new research questions present themselves. By presenting related results from the same article, like in Fig. 2, the hit-list becomes a query-biased summary of the discourse structure of the retrieved article. [16] conduct experiments on the level of detail desired by searchers. Evaluation of such a system seems to require taking both retrieval effectiveness and document summarization aspects into account.

Data Sets

Notable data-sets are:

1. The *Shakespeare test collection* used in the Focus project 2000–2001 [3].
2. The *IEEE Computer Society collection* used at INEX 2002–2004 [7].
3. The expanded *IEEE Computer Society collection* used at INEX 2005 [7].
4. The *Wikipedia XML Corpus* used at INEX 2006–2007 [7].

Cross-references

▶ Evaluation Metrics for Structured Text Retrieval
▶ INitiative for the Evaluation of XML Retrieval
▶ XML Retrieval

Recommended Reading

1. Clarke C., Kamps J., and Lalmas M. INEX 2006 retrieval task and result submission specification. In INEX 2006 Workshop Pre-Proceedings, 2006, pp. 381–388.
2. Clarke C.L.A., Kamps J., and Lalmas M. INEX 2007 retrieval task and result submission format. In Pre-Proceedings of INEX, 2007, pp. 445–453.
3. Focus. Focussed Retrieval of Structured Documents – A Large Experimental Study, 2001.
4. Fuhr N. A probabilistic framework for vague queries and imprecise information in databases. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 696–707.
5. Gövert N. and Kazai G. Overview of the INitiative for the Evaluation of XML retrieval (INEX) 2002. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2003, pp. 1–17.
6. Hearst M.A. User interfaces and visualization, Chapter X. In Modern Information Retrieval. ACM, New York, NY, USA, 1999, pp. 257–324.
7. INEX. INitiative for the Evaluation of XML Retrieval, 2007. http://inex.is.informatik.uni-duisburg.de/.
8. Jones K.S. What's the value of TREC – is there a gap to jump or a chasm to bridge? SIGIR Forum, 40(1):10–20, 2006.
9. Kazai G., Lalmas M., Gövert N., and Malik S. INEX'03 retrieval task and result submission specification. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2003, pp. 200–203.
10. Kazai G., Lalmas M., and Reid J. Construction of a test collection for the focussed retrieval of structured documents. In Proc. 25th European Conf. on IR Research, 2003, pp. 88–103.
11. Lalmas M. and Kazai G. INEX 2005 retrieval task and result submission specification. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 385–390.
12. Lalmas M. and Malik S. INEX 2004 retrieval task and result submission specification. In Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, pp. 237–240.
13. Malik S., Klas C.-P., Fuhr N., Larsen B., and Tombros A. Designing a user interface for interactive retrieval of structured documents – lessons learned from the INEX interactive track. In Proc. 10th European Conf. Research and Advanced Technology for Digital Libraries, 2006, pp. 291–302.
14. Salton G., Allan J., and Buckley C. Approaches to passage retrieval in full text information systems. In Proc. 16th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 49–58.
15. Sigurbjörnsson B. Focused Information Access using XML Element Retrieval. SIKS dissertation series 2006–2028, University of Amsterdam, 2006.
16. Szlávik Z., Tombros A., and Lalmas M. Feature- and query-based table of contents generation for XML documents. In Proc. 29th European Conf. on IR Research, 2007, pp. 456–467.
17. Tombros A., Larsen B., and Malik S. The interactive track at INEX 2004. In Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrival, 2004, pp. 410–423.
18. Trotman A., Geva S., and Kamps J. (eds.). In Proc. of the SIGIR 2007 Workshop on Focused Retrieval, 2007.
19. Trotman A., Pharo N., and Lehtonen M. XML-IR users and use cases. In Proc. 5th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 400–412.
20. Wilkinson R. Effective retrieval of structured documents. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1994, pp. 311–317.

Preservation

▶ Archiving Experimental Data

Preserving Database Consistency

▶ Correctness Criteria Beyond Serializability

Preview

► Result Display

Primary Index

Yannis Manolopoulos[1], Yannis Theodoridis[2], Vassilis J. Tsotras[3]
[1]Aristotle University of Thessaloniki, Thessaloniki, Greece
[2]University of Piraeus, Piraeus, Greece
[3]University of California-Reverside, Riverside, CA, USA

Synonyms

Clustering index; Sparse index

Definition

A tree-based index is called a *primary index* if its order is the same as the order of the file which it indexes Consider a relation *R* with some numeric attribute *A* taking values over an (ordered) domain *D*. Assume that relation *R* has been physically stored as an *ordered* file, following the order of the values of attribute *A*. Furthermore, assume that a tree-based index (e.g., B +-tree) has been created on attribute *A*. Then this index is primary.

Key Points

Tree-based indices are built on numeric attributes and maintain an order among the indexed search-key values. They are further categorized by whether their search-key ordering is the same with the file's physical order (if any). Note that a file may or may not be ordered. Ordered is a file whose records are stored in pages according to the order of the values of an attribute. Obviously, a file can have at most a single such order since it is physically stored once. For example, if the *Employee* relation is ordered according to the *name* attribute, the values in the other attributes will not be in order. A file stored without any order is called an unordered file or heap. If the search-key of a tree-based index is the same as the ordering attribute of a (ordered) file then the index is called *primary*. An index built on any non-ordering attribute of a file is called *secondary*.

Note that a primary index also controls the physical placement of the data records in a file. Since such an index also clusters the values of the file, the term *clustering* index has alternatively been used. In contrast, the order in the leaf pages of a secondary index is not the same as the order of the records in the file. Hence, a secondary index (also called *non-clustering*) needs an extra level of indirection, namely, a pointer to the actual position of a record with a given value in the relation file. In other words, a secondary index only clusters references to records (in the form of <value, pointer> fields), but *not* the records themselves. This extra indirection from a leaf page of a secondary index to the actual position of a record in a file has important subsequences on optimization. Consider for example a secondary index (B + -tree) on the *ssn* attribute of the *Employee* relation (which assume is ordered by the *name* attribute). A query that asks for the salaries of employees with *ssn* in the range *[x,y]* can facilitate the B + -tree on *ssn* to retrieve references to all records in the query range. Assume there are 1,000 such *ssn* values in the *Employee* file. Since the actual *Employee* records must be retrieved (so as to report their salaries), each such reference needs to be materialized by possibly a separate page I/O (since the actual records can be in different pages of the *Employee* file). If instead the file was ordered on the *ssn* attribute, the B + -tree on *ssn* would have clustered (as primary index) the *Employee* records on *ssn*, and thus the 1,000 records that need to be retrieved would be located within few pages.

In practice, the search-key of a primary index is usually the file's primary key, however this is not necessary. That is, primary indexes need not be on the primary keys of relations. (In the above example, it was first assumed that the *Employee* file is ordered according to the *name* attribute and not according to the primary key *ssn* attribute). A relation can have several indices, on different search-keys; among them, at most one is primary (clustering) index and the rest are secondary ones.

Cross-references

► Access Path
► Index Creation and File Structures
► Index Sequential Access Method (ISAM)
► Secondary Index

Recommended Reading

1. Elmasri R. and Navathe S.B. Fundamentals of Database Systems, 5th edn. Addisson-Wesley, Boston, MA, 2007.

2. Manolopoulos Y., Theodoridis Y., and Tsotras V.J. Advanced Database Indexing. Kluwer, Dordrecht, 1999.
3. Ramakrishnan R. and Gehrke J. Database Management Systems, 3rd edn. McGraw-Hill, New York, 2003.

Primary Memory

► Main Memory

Primitive Event

► Atomic Event

Principal Component Analysis

Heng Tao Shen
The University of Queensland, Brisbane, QLD, Australia

Synonyms

PCA

Definition

Principal components analysis (PCA) is a linear technique used to reduce a high-dimensional dataset to a lower dimensional representations for analysis and indexing. For a dataset P in D-dimensional space with its principal component set Φ, given a point p∈P, its projection on the lower d-dimensional subspace can be defined as: p. Φ_d, where Φ_d represents the matrix containing 1st to d^{th} largest principal components in Φ and $d < D$.

Key Points

PCA finds a low-dimensional embedding of the data points that best preserves their variance as measured in the high-dimensional input space [1]. It identifies the directions that best preserve the associated variances of the data points while minimize "least-squares" (Euclidean) error measured by analyzing data covariance matrix. The first principal component is the eigenvector corresponding to the largest eigenvalue of the dataset's co-variance matrix, the second component corresponds to the eigenvector with the second largest eigenvalue and so on. The chosen principal components form the lower dimensional subspace of interest. Typically, the top *d* principal components are selected since they carry the most information of original data. The lower dimensional representation for a point is then generated by multiplying the data point with the selected principal components. PCA makes a stringent assumption of orthogonality to make it amenable to linearity.

PCA has been widely used in many applications for exploratory data analysis and compression, making predictive models, indexing, information retrieval, etc.

Cross-references

► Dimensionality Reduction
► Discrete Fourier Transform
► Discrete Wavelet Transform and Wavelet Synopses
► Independent Component Analysis
► Isometric Feature Mapping
► Latent Semantic Indexing
► Locality-Preserving Mapping
► Locally Linear Embedding Laplacian Eigenmaps
► Multidimensional Scaling
► Semantic Subspace Projection
► Singular Value Decomposition

Recommended Reading

1. Jolliffe I.T. Principlal Componet Analysis. 2nd edn. Springer, New-York, 2002.

Principle of Locality

► Memory Locality

Privacy

Patrick C. K. Hung[1], Vivying S. Y. Cheng[2]
[1]University of Ontario Institute of Technology, Oshawa, ON, Canada
[2]Hong Kong University of Science and Technology, Hong Kong, China

Definition

Privacy helps to establish personal autonomy and create individualism. Privacy is a state or condition of limited access to a person (e.g., client). In particular, information privacy relates to an individual's right to determine how, when, and to what extent information about the self will be released to another person or to an organization.

Key Points

With the rising occurrence of information privacy violations, people have begun to take interest in how, when, and where their personal information is being used. People usually interchange the concepts of security with privacy. In fact, they are essentially two different concepts used to protect data. In general, security involves the use of cryptographic tools to protect information in terms of confidentiality, availability, and access control and integrity enforcement. Privacy mainly focuses on the ability of keeping data away from public access, and the way to protect it according to the individual rights.

There are various definitions of privacy in the literature. Some researchers describe confidentiality as privacy, while some regard privacy as an aspect different from using cryptographic tools. Anderson [1] defined privacy as secrecy for benefit of the individual, and confidentiality as secrecy for the benefit of the organization. Alternatively, privacy can also be described by the ability to have control over the collection, storage, access, communication, manipulation and disposition of data. Privacy is also referred as the right for individuals to determine for themselves when, how, and to what extent information about them is communicated to others. As Westin [2] notes,

> ▶ *No definition [of privacy]. . . is possible, because [those] issues are fundamentally matters of values, interests and power.*

It can be said that privacy is a much broader concept than security; privacy protection is based on security protection. Security may enable privacy protection from authorized access, but security alone cannot provide privacy.

To enhance the privacy protection of personal information, legislative schemas and practice guidelines have been proposed by different organizations such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States of America (USA) and the European Union (EU) Data Protection Act. These legislations and requirements can be abstracted as access control policies or rules to serve as a privacy foundation to control access to personal information.

Cross-references

▶ Data Privacy and Patient Consent
▶ Privacy-enhancing technologies
▶ Privacy Metrics
▶ Privacy policies and preferences
▶ Privacy-preserving data mining

Recommended Reading

1. Anderson R.J. A security policy model for clinical information systems. In Proc. 1996 IEEE Symp. on Security and Privacy, 1996.
2. Westin A. Privacy and freedom. New York, Atheneum, 1967.

Privacy Measures

▶ Privacy Metrics

Privacy Metrics

Chris Clifton
Purdue University, West Lafayette, IN, USA

Synonyms

Privacy measures

Definition

Measures to determine the susceptibility of data or a dataset to revealing private information. Measures include ability to link private data to an individual, the level of detail or correctness of sensitive information, background information needed to determine private information, etc.

Historical Background

Legal definitions of privacy are generally based on the concept of *Individually Identifiable Data*. Unfortunately, this concept does not have a clear meaning in the context of many database privacy technologies. The official statistics (census) community has long been concerned with measures for privacy, particularly in the contexts of microdata sets (datasets that represent real data, but obscured in ways to protect privacy) and tabular datasets. Measures have largely been based on the probability that a specific value belongs to a given individual, given the disclosed data. As technologies have been developed to anonymize and analyze private data, measures have been developed to quantify the potential for disclosure of private information by those techniques. In 2001, Samarati and Sweeney published papers on k-anonymity as a measure of individual identifiability, leading to considerable research in anonymity measures. At the same time, the rise in privacy-preserving data mining techniques lead to measures based on the ability to estimate values from disclosed data. As of this writing, this field is still open, with many competing measures, often tied to specific anonymization, analysis, or privacy protection techniques.

Foundations

There have been two distinct approaches to measuring privacy, or more specifically probability of disclosure. The first is measuring anonymity. k-anonymity states that any released data item must be linked to at least k individuals with equal probability. It builds on a concept of *Quasi-Identifier QI*: a quasi-identifier is information that an adversary can use to link a released data record to an individual. Formally,

k-Anonymity [8] A given table T^* is said to satisfy k-anonymity if and only if each sequence of values in $T^*[QI_{T^*}]$ appears at least k times in T^*.

While simple to compute, k-anonymity does not by itself guarantee that sensitive data is protected. As first pointed out in [7], if all k values with the same quasi-identifiers also have the same value for a sensitive attribute, k-anonymity can be met while still revealing the sensitive value for an attribute to an adversary. k-anonymity can be extended to deal with this issue:

ℓ-Diversity Principle [4] A $q^\star$-block is ℓ-diverse if contains at least ℓ "well-represented" values for the sensitive attribute S. A table is ℓ-diverse if every $q^\star$-block is ℓ-diverse.

This still does not resolve the issue, as real values may have skewed distributions; a particular rare value may be "well-represented" by occurring in only one $q^\star$-block, identifying the individuals represented in that block as having an unusually high probability of possessing that rare value. A further refinement is:

The t-closeness Principle [3] An equivalence class is said to have t-closeness if the distance between the distribution of a sensitive attribute in this class and the distribution of the attribute in the whole table is no more than a threshold t. A table is said to have t-closeness if all equivalence classes have t-closeness.

A challenge with any of these metrics is choosing appropriate values for k. Achieving a suitably low risk of disclosure may require a high value of k to protect particularly easy to identify individuals, resulting little specificity (and thus low value) in the anonymized dataset. This leads to a second set of metrics for anonymity, measuring not privacy but the fidelity of the disclosed data. These are typically based on the levels of generalization required to achieve a given level of anonymization, but as yet it isn't clear how well they relate to actual data utility (see [6]).

An alternative is to measure specifically the risk of identifying individuals, as with

δ-Presence[6] Given an external public table P, and a private table T, *δ-presence* holds for a generalization $T^\star$ of T, with $\delta = (\delta_{min}, \delta_{max})$ if

$$\delta_{min} \leq \mathcal{P}(t \in T \mid T^*) \leq \delta_{max} \qquad \forall\ t \in P.$$

The above approaches measure the ability of an adversary to link a data item to a specific individual. An alternative is to measure the ability to estimate sensitive data for an individual, once such linkage is performed. In [2], the assumption was that individuals provided sensitive data directly to the data analyst; the link between data and individual was thus known. Instead, the sensitive data is distorted to prevent the analyst from knowing values for an individual. This leads to a metric based on bounding the knowledge gained by an adversary from seeing the (distorted) sensitive value. The metric in [2] was based on two things: an interval and confidence level. If the analyst can determine that a value lies within the range $[x_1, x_2]$ with c% confidence, then the privacy at that confidence level is $x_2 - x_1$.

This measure raises two issues. First (as with ℓ-diversity), it relies on two numbers. This increases the difficulty of using it in practice. Second, as pointed out in [1], knowledge of the distribution of the original data (acquired by the analyst from the distorted dataset) may allow tightening of the bounds. They propose a measure based on the *differential entropy* of a random variable. The differential entropy $h(A)$ is a measure of the uncertainty inherent in A. Their metric for privacy is $2^{h(A)}$. Specifically, when adding noise from a random variable A, the privacy is:

$$\Pi(A) = 2^{-\int_{\Omega_A} f_A(a) log_2 f_A(a) da}$$

where Ω_A is the domain of A. This metric has several nice features. It is intuitively satisfying for simple cases. For noise generated from A, a uniformly distributed random variable between 0 and a, $\Pi(A) = a$. Thus this privacy metric is exactly the width of the unknown region. Furthermore, if a sequence of random variables A_i converges to B, then $\Pi(A_i)$ converges to $\Pi(B)$. For most random variables, e.g., a gaussian, the notion of width of the unknown region does not make sense. However, by calculating Π for such random variables, the above properties can be used to make the case that the privacy is equivalent to having no knowledge of the value except that it is within a region of width Π. This gives an intuitively satisfying way of comparing the privacy of different methods of adding random noise.

The authors extend this definition to *conditional privacy*, capturing the possibility that the inherent privacy from obscuring data may be reduced by what can be learned from a collection. The conditional privacy $\Pi(A|B)$ follows from the definition of conditional entropy:

$$\Pi(A|B) = 2^{-\int_{\Omega_{A,B}} f_{A,B}(a,b) log_2 f_{A|B=b}(a) dadb}$$

They show how this can be applied to measure the actual privacy after reconstructing distributions of the original data to improve the accuracy of decision trees build on the obscured data [1,2].

Another use of this metric is to evaluate the inherent loss of privacy caused by data mining results. The use of conditional privacy enables estimating how much privacy is lost by knowing the data mining results, even with a "perfect" privacy-preserving technique such as secure multiparty computation. The literature has not yet addressed this issue; the assumption has generally been that the data mining results do not of themselves violate privacy.

Numerous other metrics have been proposed; the above give a representative view of the approaches and challenges faced in measuring privacy.

Key Applications

Primary applications are in aggregate analysis of privacy-sensitive data, such as individual health-care, personal preference, or financial information. Direct access to individual data is typically a confidentiality and access control issue; access to data is either allowed or it is not. Privacy measures come into play when individual data is analyzed as part of a broader study, where the end result is not solely for the benefit of the individuals whose data was used. (e.g., Studies of demographics or medical research.) Privacy laws typically control the use of data when not specifically to complete a transaction on behalf of the individual; measures are thus needed to show that privacy is maintained.

Future Directions

One approach is measures that correlate with legal and regulatory standards (e.g., the HIPAA safe harbor rules). Perhaps more promising as a research direction is risk-based measures such as δ-presence [5]. Location data (as from mobile devices) also poses new issues; European Community rules specifically discuss protection of location, but it is not clear to what extent existing measures can be applied. Another need is measures that adjust for differences in individual privacy needs, e.g., protecting outliers.

Cross-references

▶ Individually Identifiable Data
▶ Privacy
▶ Privacy-Preserving Data Mining
▶ Randomization Methods to Ensure Data Privacy
▶ Statistical Disclosure Limitation for Data Access

Recommended Reading

1. Agrawal D. and Aggarwal C.C. On the design and quantification of privacy preserving data mining algorithms. In Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2001, pp. 247–255.
2. Agrawal R. and Srikant R. Privacy-preserving data mining. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 439–450.
3. Li N. and Li T. T-closeness: privacy beyond k-anonymity and l-diversity. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
4. Machanavajjhala A., Gehrke J., Kifer D., and Venkitasubramaniam M. *l*-diversity: privacy beyond *k*-anonymity. ACM Trans. Knowl. Discov. Data, 1(1), No.3, March 2007.
5. Nergiz M., Atzori M., and Clifton C. Hiding the presence of individuals from shared databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 665–676.
6. Nergiz M.E. and Clifton C. Thoughts on k-anonymization. Data Knowl. Eng., 63(3):622–645, December 2007.
7. Øhrn A. and Ohno-Machado L. Using boolean reasoning to anonymize databases. Artif. Intell. Med., 15(3):235–254, 1999.
8. Sweeney L. Achieving *k*-anonymity privacy protection using generalization and suppression. Int. J. Uncertainty Fuzziness Knowl.-based Syst., 10(5):557–570, 2002.

Privacy Policies and Preferences

Patrick C. K. Hung, Yi Zheng, Stephanie Chow
University of Ontario Institute of Technology, Oshawa, ON, Canada

Definition

Privacy policies describe an enterprise's data practices, what information is collected from individuals (subjects), what the information (objects) will be used for, whether the enterprise provides access to the information, who the recipients are of any result generated from the information, how long the information will be retained, and who will be informed in the event of a dispute. A subject releases his or her data to the custody of an enterprise while consenting to the set of purposes for which the data may be used. The subject can express his or her preferences in a set of preference-rules to make decisions regarding the acceptability of privacy policies.

Historical Background

People have been concerned with privacy policies and preferences for more than 200 years. For example, the Hippocratic Oath was written as a guideline of medical ethics for doctors in respect to a patient's health condition, and states as follows: "*Whatsoever things I see or hear concerning the life of men, in my attendance on the sick or even apart there from, which ought not to be noised abroad, I will keep silence thereon, counting such things to be sacred secrets*." Privacy policies and preferences are often expressed in natural language to specify or regulate how a system or an organization is to preserve privacy. In response to privacy policy and preference violations, many countries have enacted legislation to protect privacy for individuals. There are also different legislations for different industry sectors. For example, in the financial sector the Gramm-Leach-Bliley Act (GLB Act) of the U.S. requires banks to implement security programs to protect customer information; in the healthcare sector the Health Insurance Portability and Accountability Act (HIPAA) of the U.S. sets an American national standard to protect and enhance the rights of consumers, clients, and patients to control how their health information is used and disclosed. Similarly, the Personal Information Protection and Electronic Documents Act (PIPEDA) of Canada sets out ground rules for how private sector organizations may collect, use, or disclose personal information in the course of commercial activities. Failing to comply with these legislations, in their respective countries, may lead to civil and/or criminal penalties and/or imprisonment. In addition to the penalties, organizations may even suffer the loss of reputation and goodwill when the non-compliance of legislation is publicized. Looking at the above factors, it is evident that the privacy legislations have forced a widespread impact on the privacy policies and preferences.

Foundations

One can imagine that information privacy is usually concerned with the confidentiality of information. Threats to information privacy can come from insiders and from the outsiders in each enterprise. Privacy control is usually not concerned with individual subjects. Privacy policies are often expressed in natural language to specify or regulate how a system or an enterprise is to preserve privacy. Here are the principles of information privacy protection:

- Principle 1: Data-level security protection principle
  - Principle 1.1: Personal information shall be protected by security safeguards in a secure way such that data confidentiality, integrity, and availability can be achieved.
  - Principle 1.2: In needs to be accurate, complete, and up-to-date. The correction of inaccurate information should be allowed for maintaining data quality.
- Principle 2: Communication-level security protection principle
  - Principle 2.1: Information shall be transported in a secure way that data confidentiality and authentication of users/systems/services are achieved.
  - Principle 2.2: Data integrity must be maintained during the transmission in the communication channel.
- Principle 3: Consent requirement principle
  - Principle 3.1: Consent must be given by the owner of the information for the collection, disclosure, use, and retention of his/her information.
  - Principle 3.2: Owners of information should be allowed to review, control, and setup restrictions to their information on collection, usage, disclosure, and retention.

Principle 3 includes four limitations to specify how personal information can be released upon request:

- *Limitation on Collection*: The collection of personal information shall be limited to specific legitimate purposes of collection only.
- *Limitation on Disclosure*: The owner of information should be able to make special restrictions on the disclosure of his/her own information.
- *Limitation on Use*: The use of personal information shall be identified as legitimate use by the services provider and/or the owner of information.
- *Limitation on Retention*: Personal information shall be retained for only as long as the purpose for which it is used.

A subject releases his or her data to the custody of an enterprise while consenting to the set of purposes for which the data may be used. The subject can express his or her preferences in a set of preference-rules to make decisions regarding the acceptability of privacy policies. Preference assumes a real or imagined choice between alternatives and the possibility of rank ordering of these alternatives in a privacy policy. Privacy preferences are formally expressed by a set of rules and should preferably be captured through an interface. Subjects who are aware of how their personally identifiable information is being used, collected or disclosed can better understand how to protect it. Personally Identifiable Information (PII) includes information that identifies and individual and information that an organization may be able to identify. This includes, but is not limited to a subject's name, address, telephone number, social insurance number and credit card number(s). Among the most sensitive PII are medical and financial records.

Key Applications

In many instances, subjects do not have the knowledge or understanding of how his or her PII is being used, collected or disclosed, nor what privacy preferences are available. The adoption of information technology and the Internet have further added to this complexity. Even the most common daily transactions such as Internet browsing, grocery shopping and online banking increase the exposure and vulnerability of threats to information privacy. In response to these threats, and concerns surrounding data integrity, security, online privacy and confidentiality, professional services firms have begun to expand their service lines to include third-party enforcement programs. Seals, or other easily distinguishable symbols, are issued to enterprises whose privacy policies and procedures have been concluded to be in compliance with its governing board(s). The purposes of third-party enforcement programs are three-fold: (i) to build consumer trust; (ii) to educate subjects of their privacy preferences; and (iii) to develop a complaint resolution mechanism. Here are the three major procedures of enforcing privacy policies in an organization:

- Building consumer trust
  - There exists a natural conflict of interest between organization and subject.
  - Subject is absent from the development of privacy policy.
  - Independent review of an organization's compliance with governing board(s) adds reliability and credibility.
- Educate subjects of their privacy preferences
  - Promote awareness of privacy issues and how to get in contact with privacy coordinator.
  - If for any reason a subject's PII is required for another purpose from when it was collected, it is the enterprise's responsibility to obtain additional consent.
  - At a minimum, the enterprise is to inform the subject of the circumstance and provide an opportunity for the subject to opt out of such a use.
- Complaint resolution mechanism
  - Organizations rewarded a seal of validation are required to provide subjects a method to resolve any problems or discuss any complaints.
  - Complaint resolution process should be easily accessible and comprehendible.

In order to stand apart from industry rivals, companies strive to obtain a competitive advantage. As an organization, it is advantageous to be knowledgeable of the external risks associated affecting the protection of a subject's privacy. Participants of third-party enforcement programs are continuously monitored for adaptability to legislative frameworks and threats to privacy compliance.

Privacy technologies have been researched for a period of time. For example, the Platform for Privacy Preferences Project (P3P) working group at World Wide Web Consortium (W3C) develops the P3P specification for enabling Web sites to express their privacy practices in a standard and machine-readable XML format. P3P user agents allow users to automatically be informed of site practices and to automate decision-making based on the Web sites' privacy practices. In addition, P3P also provides a language called P3P Preference Exchange Language 1.0 (APPEL1.0) that is used to express the user's preferences for making automated or semi-automated decisions regarding the acceptability of machine-readable privacy policies from P3P enabled Web sites. It provides a base schema for the data collected and a vocabulary to express purposes, the recipients, and the retention policy. Although it captures common elements of privacy policies, sites may have to provide further explanations in human-readable policies.

Furthermore, WS-Privacy has been mentioned in industry for a period of time for defining subject privacy preferences and organizational privacy practice statements for Web services. At this minute, the WS-Privacy specification has not been released to public yet. Then, the Enterprise Privacy Authorization Language (EPAL) technical specification is used to formalize privacy authorizations for actual enforcement within an intra- or inter- enterprise for business-to-business privacy control. On the other hand, the XACML is a general- purpose access control policy language used to describe policy and access control decision request/response [11]. Though XACML has drafted a privacy policy profile document [12], the current XACML framework can not handle the privacy enforcement.

One of the most significant privacy technologies is the IBM Tivoli Privacy Manager for e-business. This privacy middleware technology converts privacy policy and data-handling rules from applications and IT systems to P3P format. The future development will include the integration of privacy technologies into some specific Web-based applications like in healthcare sector (e.g., Microsoft Healthvault and Google Health Portal), especially in light of recent changes in health privacy legislative environment.

Cross-references

- ▶ Data Privacy and Patient Consent
- ▶ Privacy
- ▶ Privacy-Enhancing Technologies
- ▶ Privacy Metrics
- ▶ Privacy-Preserving Data Mining

Recommended Reading

1. Cheng V.S.Y. and Hung P.C.K. Health Insurance Portability and Accountability Act (HIPAA) compliant access control model for web services. IJHISI, 1(1):22–39, 2005.
2. Fischer-Hubner S. IT-security and privacy. Lecture notes in computer science, Springer, Berlin Heidelberg New York, 2001.
3. Online Privacy Alliance. Effective Enforcement of Self Regulation. Online: http://www.privacyalliance.org/resources/enforcement.shtml
4. Powers C.S., Ashley P., and Schunter M. Privacy promises, access control, and privacy management – enforcing privacy throughout an enterprise by extending access control. In Proc. Third Int. Symp. on Electronic Commerce, pp. 13–21. IEEE Computer Society, 2002.

Privacy Protection

▶ Statistical Disclosure Limitation For Data Access

Privacy-Enhancing Technologies

SIMONE FISCHER-HÜBNER
Karlstad University, Karlstad, Sweden

Synonyms

PETs

Definition

Privacy-Enhancing Technologies (PETs) can be defined as technologies that are enforcing privacy principles in order to protect and enhance the privacy of users of information technology (IT) and/or of individuals about whom personal data are processed (the so-called data subjects). Privacy principles that PETs are enforcing can be derived from internationally acknowledged privacy guidelines or legislation, such as the OECD Privacy Guidelines [10] and the EU Data Protection Directive 95/46/EC [7]. One fundamental privacy principal that serves as the foundation for the Privacy-Enhancing Technologies that are aiming at providing *anonymity*, *pseudonymity*, or *unobservability* for users and/or other data subjects, is the privacy principles of data minimization. It requires that the collection of personally identifiable data should be minimized (and if possible avoided), because obviously privacy is best protected if no personal data at all (or at least as little data as possible) is collected or processed. Further important privacy principles addressed by other PETs are the informed consent by data subjects (as a prerequisite for making data processing legitimate), transparency (i.e., openness) of data processing, and appropriate technical security means for protecting the confidentiality, integrity and availability personal data.

Historical Background

IT security technologies for protecting the confidentiality, integrity and availability of (personal) data, such as *access control*, have been developed and researched since the beginning of computing – *data encryption* as a technique for protecting the confidentiality of data has been used since ancient times. *Inference controls* for protecting the data of individuals stored in statistical databases (e.g., medical databases) have been elaborated since the 1970's.

Most of the fundamental anonymity technology concepts were introduced by David Chaum in the 1980's (see: [3,4,2]). The term "Privacy-Enhancing Technologies" was first introduced in 1995 in a report on data minimization technologies, which was jointly published by the Dutch Registratiekamer and the Information and Privacy Commissioner in Ontario with the title "Privacy-Enhancing Technologies: The path to Anonymity" [13]. Since then, further research and development has been done in data minimization technologies (e.g., [12,14]), and areas such as privacy policies [9,10] and privacy enhancing identity management [11].

Privacy as an expression of the rights of self-determination and human dignity is considered a core value in democratic societies and is recognized either explicitly or implicitly as a fundamental human right by most constitutions of democratic societies. However, in the network society, individuals are increasingly at risk that all their communications, transactions and movements can be monitored and profiled. Profiles collected at various sites can be easily combined, aggregated and retained without time limitations and without the individuals' knowledge or consent. Research and development of PETs have been motivated by the vision to provide technical means allowing individuals to retain control over their personal spheres, i.e., to protect their privacy in the electronic information age.

Foundations

Privacy-Enhancing Technologies can basically be divided into:

1. The class of PETs for minimizing or avoiding identifiable data for users and/or other data subjects.
2. The class of PETs for safeguarding lawful and privacy-friendly personal data processing.
3. PETs that are a combination of the two aforementioned PET classes.

PETs for Minimizing or Avoiding Personally Identifiable Data

To the class PETs that are minimizing or eliminating personally identifiable data of data subjects (that are not acting as users) belong *inference controls* in statistical databases. In general-purpose database systems (e.g., medical databases), some users (e.g., physicians) need to access personal data attributes, while other user (e.g., researchers) only need to access a personal database by statistical queries. Thus, access control mechanisms should allow certain users to access data in anonymous form by performing only statistical queries. However, by correlating different statistics (i.e., by launching so-called tracker attacks), a user may succeed in deducing confidential information about some individual. *Inference controls* in statistical databases shall ensure that the statistics released by the database do not lead to the disclosure of any confidential personal data.

PETs that are minimizing/avoiding identifiable data of users, and that are thereby providing *anonymity*, *pseudonymity*, and/or *unobservability*, can be divided dependent on whether data is minimized on communication level or on application level. Examples for anonymous communication schemes for achieving sender anonymity are DC-nets [4], Mix-nets [2] or Crowds [14], where the latter two ones will be presented in more detail below.

With DC-nets, the fact that someone is sending a message is hidden by a one-time pad encryption, which means that DC-nets can offer *perfect sender anonymity* (in the information-theoretic sense). By the use of message broadcast and implicit addresses (i.e., by the use of an attribute which allows only the addressee to recognize that the message is addressed to him, e.g., the message is public-key encrypted and the addressee is the only one who can successfully decrypt it with his secret key), it also provides *receiver and relationship anonymity.*

Another example for a protocol providing receiver anonymity is private information retrieval [2], which assures that a powerful attacker is not able to find out what information another user has requested (i.e., who has received certain information). The goal is to request exactly one datum that is stored in a remote memory cell of a server without revealing which datum is requested.

Mix Nets The technique of a Mix network, which was originally introduced by David Chaum [4], realizes unlinkability of a sender and recipient (*relationship anonymity*), as well as, *sender anonymity* against the recipient, and optionally *recipient anonymity* (via so-called anonymous return addresses, which is however not elaborated further here).

A mix is a special network station, which collects and stores incoming messages, discards repeats, changes their appearance by encryption, and outputs them in a different order, and by this hides the relation between incoming and outgoing messages. If a sender of a message uses one mix for forwarding a message to a recipient on his behalf, the relation between sender and recipient is hidden from everybody but the mix and the sender of the message. This means also that the recipient only learns that the message was sent to him by the mix, but he does not know the identity of the real sender. However, if only one mix is used, this mix can in detail learn and potentially profile who is communicating with whom. To improve security, a message is sent over a mix net, which consists of a chain of independent mixes. The sender must perform cryptographic operations inverse to those of the mix, because the recipient must be able to read the message. A global attacker, who can monitor all communication lines, should in principle only be able to trace a message through the mix network, if he has the cooperation of all mix nodes on the path or if he can break the cryptographic operations. Thus, in order to ensure unlinkability of sender and recipient, at least one mix in the chain has to be trustworthy.

Assume that Alice wants to send anonymously a message msg to recipient Bob (which could be encrypted with Bob's public key for providing also message secrecy). Alice chooses a mix sequence Mix_1, Mix_2,..., Mix_m (In Fig. 1, Alice chooses m = 3 Mixes). Let for simplicity Mix_{m+1} denote the recipient (Bob). Each Mix_i with address A_i has initially chosen a key pair (c_i, d_i), where c_i is a public key of Mix_i and d_i is its private key. Let z_i be a random string.

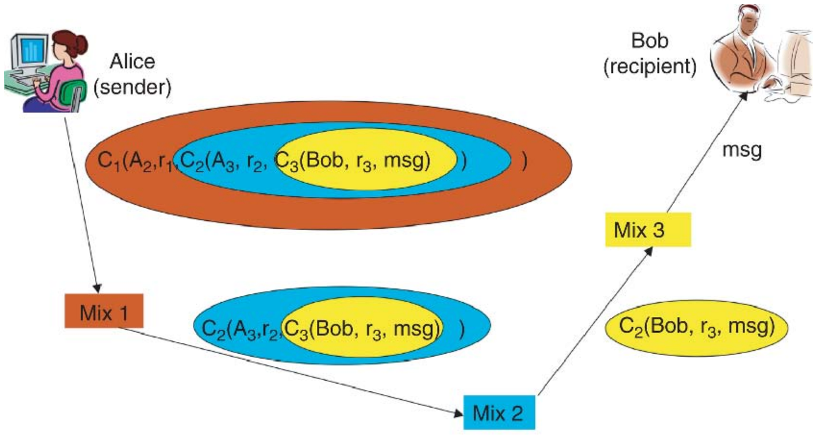
Alice recursively creates the following encrypted messages, where M_i is the message that Mix_i will receive:

$$M_{m+1} = \mathrm{msg}$$

$$M_i = c_i(z_i, A_{i+1}, M_{i+1}) \quad \text{for } i = 1,\ldots, m,$$

i.e., she adds one layer of public key encryption to the message in reverse order to the sequence of mixes in the chain. She then sends the result M_1 to Mix_1.

Each Mix_i decrypts the incoming message $M_i = c_i(z_i, A_{i+1}, M_{i+1})$ with its private key d_i, discards the random string z_i, and finds the address A_{i+1} of the next mix Mix_{i+1} in the chain and the message M_{i+1}, which it forwards to Mix_{i+1}. The random string z_i is needed, because otherwise an attacker could again, by


Alice
(sender)
Bob
(recipient)
$C_1(A_2,r_1,C_2(A_3, r_2, C_3(Bob, r_3, msg)))$
msg
Mix 3
Mix 1
$C_2(A_3,r_2,C_3(Bob, r_3, msg))$
$C_2(Bob, r_3, msg)$
Mix 2

C_i: public key of Mix_i, r_i: random number, A_i: address of Mix_i

Privacy-Enhancing Technologies. Figure 1. Mix-network with a chain of three mixes.

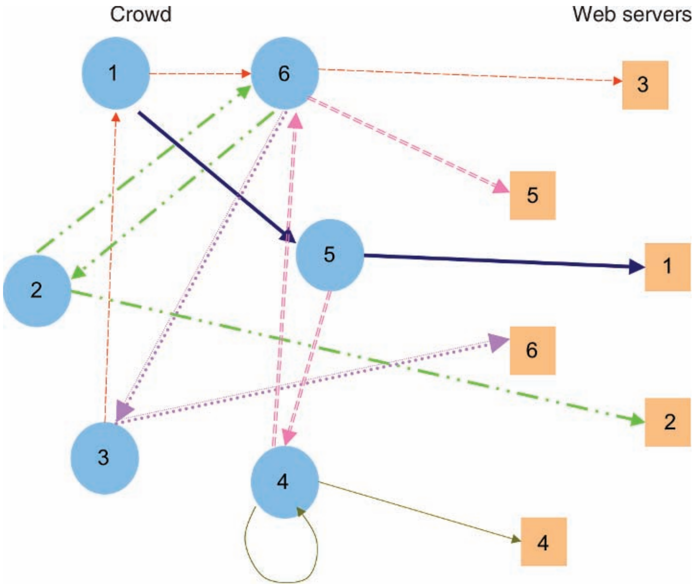
encrypting an outgoing message from a mix with the public key of the mix and comparing the result with the former inputs to the mix, succeed to relate an input to its output (i.e., he would be able to trace the message flow through the mix). Besides *anonymity* protection, mix nets can also provide *unobservability*, if dummy messages (i.e., meaningless messages) are sent out by the participants in order to hide information about if and when meaningful messages were sent.

Attacks on mix nets and possible countermeasures have been discussed in [8].

Mix-nets have first been developed for anonymous email applications. Later the Mix-net concept has been used for establishing anonymous bi-directional real-time communication. One example for distributed overlay networks designed to anonymize real-time, bidirectional TCP-based communications is Onion Routing, in which the sender's proxy constructs an onion which encapsulates a route to the responder (e.g., Web server). An Onion is an object with layers of public key encryption, which is used to produce anonymous bi-directional virtual circuit between communication partners (in Mix nets layered public key encryption are also used to build up a path) and to distribute symmetric keys to the nodes on the path. The sender's proxy constructs an onion which encapsulates a route to the responder. Faster symmetric encryption with the symmetric keys that were distributed to the nodes on the path with the help of the Onion is used for data communication via the virtual path once it has been established.

Crowds Reiter and Rubin developed Crowds – a system based on the idea that users can make anonymous Web transactions when they blend into a "crowd" [14]. A crowd is a geographically diverse group that performs Web transactions on behalf of its members. Each crowd member runs a process on his local machine called "jondo." Once started, the jondo engages in a protocol to join the crowd, during which it is informed of the other current crowd members and in which the other crowd members are informed of the new jondo's membership. Besides, the user configures his browser to employ the local jondo as a proxy for all network services. Thus all http requests are directly sent to the jondo rather than to the end Web-server and the jondo forwards the request to a randomly chosen crowd member. Whenever a crowd member receives a request from another jondo in the crowd, it makes a random choice to forward the request to another crowd member with a probability $p_f > ½$ or to submit the request to the end Web server to which the request was destined. Figure 2 shows a possible set of virtual paths that are created according to the Crowds protocol.

The server's reply is sent backward along the path, with each jondo sending it to its predecessor on the



Privacy-Enhancing Technologies. Figure 2. Paths in a crowd, where for each path the initiator (sender) and web server are labeled the same [14].

path. All communication between jondos is encrypted by a cryptographic key shared between the jondos involved.

Crowds provides sender anonymity against the end Web-server, since the end server obtains no information regarding who initiated any given request (each Crowds member is equally "suspicious"). Second, since a jondo on the path cannot distinguish whether its predecessor on the path initiated the request or is forwarding it, no jondo on the path can learn the initiator of a request. Since all communication between jondos is encrypted, Crowds also offers receiver anonymity against a local eavesdropper (e.g., local gateway administrator) that can observe the communication involving the user's machine unless the originator of the request ends up submitting the request itself. (However, the probability that an originator submits its own request decreases as the crowd size increases).

Crowds enables very efficient implementations that typically outperforms mixes that uses layered encryption techniques. However, in contrast to mix-nets, crowds cannot protect against global attackers.

PETs for anonymous or pseudonymous applications/transactions can be implemented on top of anonymous communication protocols. Examples are anonymous E-cash protocols based on *blind signatures* [3], as well as anonymous credential systems (such as Idemix [1]), which can be used to implement for instance anonymous or pseudonymous access control, e-health or e-government, and other anonymous or pseudonymous applications.

PETs for Safeguarding Lawful and Privacy-Friendly Personal Data Processing

It is not always possible to avoid the processing of personal data. Public authorities, health care providers, employers are example of organizations that still need to process personal data about their citizens, patients, employers for various legitimate reasons. If personal data are collected and processed, legal privacy requirements need to be fulfilled. The class of PETs described in this section comprises technologies that enforce legal privacy requirements in order to safeguard the lawful processing of personal data. The driving principles behind these types of PETs can also be found in data protection legislation, such as the EU Data Protection Directive 95/46/EC, which requires that controllers implement security measures which are appropriate to the risks presented for personal data in storage or transmission, with a view to protecting personal data against accidental loss, alteration, unauthorized access and against all other unlawful forms of processing.

Examples of technologies belonging to this class of PETs include classical security technologies, such as *data encryption* or *access control*, which protects the confidentiality and integrity of personal data. Other examples are technologies for stating or enforcing privacy policies: The Platform for Privacy Preferences Protocol (P3P) [6] increases transparency for the end users by informing them about the privacy policies of web sites and can hence be used to enforce the legal principles and requirement (e.g., according to Art.10 EU Data Protection Directive 95/46/EC) to inform data subjects about the purpose of data processing, identity of the controller, retention period, etc. Privacy policy models (such as the privacy model presented in [8]) can technically enforce privacy requirements such as purpose binding, and privacy policy languages, such as the Enterprise Privacy Authorization Language EPAL [9], can be used to encode and to enforce more complex enterprise privacy policies within and across organizations.

PETs that are a Combination of the Two Aforementioned PET Classes

The third class of PETs comprises technologies that are combining PETs of class 1 and class 2. An example is provided by privacy-enhancing identity management technologies as the ones that have been developed within the EU FP6 project PRIME (Privacy and Identity Management for Europe) [11]. Identity Management (IDM) subsumes all functionalities that support the use of multiple identities by the identity owner (user-side IDM) and by those parties with whom the owner interacts (services-side IDM). The PRIME project addresses privacy-enhancing IDM to support strong privacy by particularly avoiding or reducing identification and by technically enforcing informational self-determination.

PRIME is based on the principle that design must start from maximum privacy. Therefore, with the help of anonymous communication technologies and anonymous credential protocols, a priori all interactions are anonymous, and individuals can chose pseudonyms to link different interactions to each other, bind attributes and capabilities to pseudonyms and can establish end-to-end secure channels between pseudonyms. Whether or not interactions are linked to each other or to a certain pseudonym is under the individual's control. For this, PRIME tools allow individuals to act under different pseudonyms with respect to communication partners, roles or activities. Policy management tools are helping them to define and negotiate privacy policies with services sides regulating who has the right to do what with one's personal data under which circumstances. Those policies are enforced at the receiving end, and there is enough evidence, so that users can actually trust in this enforcement. Transparency tools allow users to be informed about who has received what personal data related to them, and to trace personal data being passed on, and include online functions for exercising their rights to object to data processing or to rectify, block, delete data (see also [11] for more information).

Key Applications

As illustrated above, PETs can be applied in all kinds of application areas of IT, in which personal data is or can be collected or processed.

Cross-references

- ▶ Access Control
- ▶ Anonymity
- ▶ Blind Signatures
- ▶ Data Encryption
- ▶ Data Security
- ▶ Inference Control in Statistical Databases
- ▶ Privacy Policies and Preferences
- ▶ Pseudonymity
- ▶ Unobservability

Recommended Reading

1. Camenisch J. and van Herreweghen E. Design and implementation of the idemix anonymous credential system. In Proc. Ninth ACM Conf. on Computer and Communications Security, 2002, pp. 21–30.
2. Chaum D.L. Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM, 24(2):84–88, 1981.
3. Chaum D.L. Security without identification: card computers to make big brother obsolete. Informatik-Spektrum, 10:262–277, 1987.
4. Chaum D.L. The dining cryptographers problem: unconditional sender and recipient untraceability. J. Cryptol., 1(1):65–75, 1988.
5. Cooper D.A. and Birman K.P. Preserving privacy in a network of mobile computers. In Proc. IEEE Symp. on Security and Privacy, 1995, pp. 26–83.
6. Cranor L. Web Privacy with P3P. O'Reilly, Sebastopol, CA, 2002.
7. European Union. Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. Official Journal L, 281:1995.
8. Fischer-Hübner S. IT-Security and Privacy: Design and Use of Privacy Enhancing Security Mechanisms. LNCS, vol. 1958, Springer, Berlin, 2001.
9. Karjoth G., Schunter M., and Waidner M. Platform for enterprise privacy practices: privacy-enabled management of customer data. In Proc. Second Workshop on Privacy Enhancing Technologies, 2002, pp. 69–84.
10. Organization for Economic Co-operation and Development. Guidelines on the protection of privacy and transborder flows of personal data. OECD Guidelines, 1980.
11. PRIME EU FP6 Project. Privacy and Identity Management for Europe. Available at: http://www.prime-project.eu/.
12. Reed M.G., Syverson P.F., and Goldschlag D.M. Anonymous connections and onion routing. IEEE J. Select. Areas Commn., 16(4):482–494, 1998.
13. Registratiekamer, Information and Privacy Commissioner/ Ontario. Privacy-enhancing technologies: the path to anonymity. Achtergrondstudies en Verkenningen 5B, vols. I and II, Rijswijk, 1995.
14. Reiter M. and Rubin A. Anonymous web transactions with crowds. Commun. ACM, 42(2):32–48, 1999.

PETs

▶ Privacy-Enhancing Technologies

Privacy-Preserving Data Mining

CHRIS CLIFTON
Purdue University, West Lafayette, IN, USA

Definition

Data Mining techniques that use specialized approaches to protect against the disclosure of private information may involve anonymizing private data, distorting sensitive values, encrypting data, or other means to ensure that sensitive data is protected.

Historical Background

The field of privacy-preserving data mining began in 2000 with two papers of that name[1,4]. Both papers addressed construction of decision trees, approximating the ID3 algorithm while limiting disclosure of data. While the problems appeared similar on the surface, the fundamental difference in privacy constraints shows the complexity of this field. In [1], the assumption was that individuals were providing their own data to a common server, and added noise to sensitive values to protect privacy. The key to the technique was to discover the original distribution of the data, enabling successful construction of the decision tree. In [4], the data was presumed to be divided between two (or a small number) of parties, and cryptographic *Secure Multiparty Computation* techniques are used to construct the decision tree without any site disclosing the actual data values it holds.

A third approach to privacy-preserving data mining, *Data Transformation*, was introduced in [5]. The idea behind this approach is to transform the data space (e.g., using techniques such as Random Projection) so that data items can no longer be tied to individuals, but sufficient information is preserved to enable data mining.

Research in the field has largely been directed toward:

- Supporting more data mining tasks and algorithms (e.g., clustering, classification, outlier detection)
- Different privacy constraints (e.g., *vertically partitioned* and *horizontally partitioned* data)

Techniques provide varying levels of privacy guarantees, data mining result accuracy, and run-time performance.

Foundations

There are several factors that must be identified in developing or choosing a privacy-preserving data mining approach. The first is the source of and privacy constraints on the data. In some cases, data is held by a relatively small number of trusted parties, such as credit reporting bureaus, insurance companies, or government agencies. The challenge is that while these parties are allowed to use the data for their own purposes, privacy constraints prevent them from disclosing the data to others.

Secure Multiparty Computation Methods address this problem, allowing a data mining model to be constructed from the distributed data using cryptographic approaches that prevent disclosure of the individual data items between the parties. The key components of such a method are the algorithm, which uses cryptographic computations to duplicate a distributed data mining algorithm without disclosing data from any site, and a proof that disclosure is controlled. Typically such proofs use a simulation argument, showing that given the final result, a party can simulate the messages received in running the protocol (thus showing that no valuable information was disclosed other than that inherent in the result.)

When data is more widely distributed, such as individuals providing their data to a server for use in data mining, *Data Transformation Methods* and *Data Perturbation* come into play. Data transformation methods modify the data so that the original values are lost, but important information (e.g., distances) is preserved. A typical example of transformation is dimensionality reduction, for example random projection [3]. Typically the providers of data must agree on a transformation, then transform their data and send it to a central data collector / data miner. Details of the transformation are kept secret from the data miner. In many cases, the transformation is individualized or constructed collaboratively in a way that protects against collusion. Such techniques require specialized versions of data mining algorithms to mine the transformed data. The techniques also involve a proof; in this case, showing the difficulty/impossibility of reversing the transformation, or the amount of background knowledge required to do so.

Data Perturbation is similar in that individuals modify their data before providing it to a central data warehouse. The differences is that data perturbation techniques add noise, while preserving the structure of the original data. The key to these techniques is specialized data mining algorithms that can utilize knowledge of the distribution of the noise, along with the noisy data, to obtain better results than mining on the noisy data alone. These approaches typically reconstruct the data *distribution*, then use this to guide the data mining, rather than mining the data directly. As the goal of most data mining is to obtain models that generalize well, models consistent with the distribution of the data are likely to have good accuracy. In addition to the reconstruction algorithm, data perturbation methods need a measure of the privacy provided; as data is not completely hidden, it is important to be able to measure the tradeoff between privacy and model accuracy.

Once the data privacy constraints are understood and a general approach is chosen, the next step is to develop an algorithm for the data mining model to be built. Privacy-preserving data mining algorithms typically replicate the results of traditional data mining algorithms, and have been developed for many of the standard machine learning approaches.

Key Applications

Privacy-preserving data mining has application wherever data mining is applied to data about individuals. Examples include recommender systems, medical studies, intelligence, and social network analysis. In addition, this technology can be useful for sharing sensitive corporate information, for example in supply chain management [2].

As an example, Figure 1 shows a scenario evaluating an Algorithm developed to identify terrorist organization financial patterns. To evaluate this, it is necessary to compare real financial records to see how many are identified by the algorithm. An outlier detection algorithm could be used to determine if the space identified by the algorithm reflects a cluster of "normal" financial patterns or just a few outliers, but accessing real financial records is problematic. Using the secure multiparty computation technique in [6], it is possible to identify the number of items in the vicinity of "Tom Terrorist" without revealing either the identity or specific values of any of the items. This gives us the result sought without exposing private data.

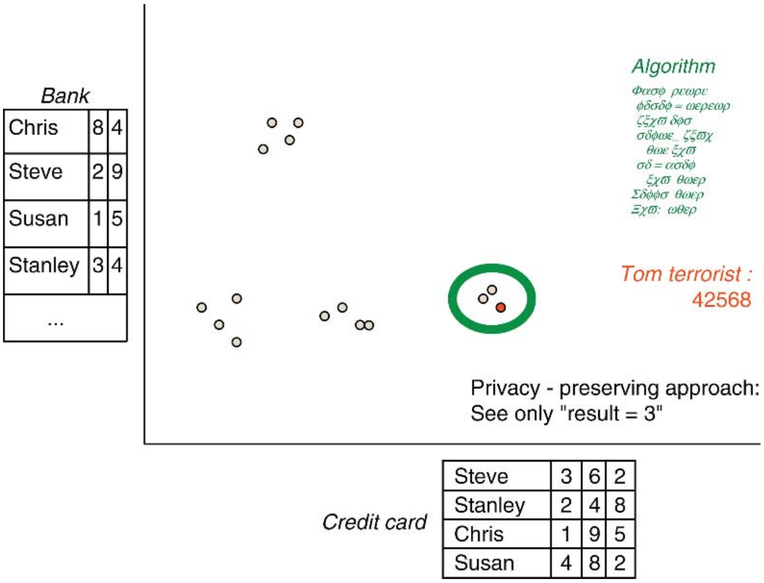
Future Directions

The main factors limiting adoption of this technology are:

1. The difficulty of doing exploratory data analysis. Privacy-preserving data mining techniques require that the data mining task to be performed be known without seeing the data.
2. Privacy standards and definitions. Current privacy regulations do not make it clear if this technology is either necessary or sufficient. Further work is needed to establish technically meaningful definitions of privacy.
3. Cost/performance. Many of these techniques are expensive in either computational power, their effect on result quality, and perhaps most important the cost of implementing a solution for a particular task.

Ongoing work is addressing these issues. Applications in corporate collaboration using corporate-sensitive data is likely to lead commercialization, as cost/benefit tradeoffs are easier to evaluate than with personal private data.

The field is closely related to *inference control in statistical databases* and *statistical disclosure limitation*; these fields have extensively studied the privacy risks inherent in releasing data under various types of perturbation and transformation. The key difference with privacy-preserving data mining is that the specific use of the data is assumed to be known. This allows methods that drastically distort the data, keeping only specific information intact rather than trying to preserve general statistical properties. The success of statistical disclosure limitation provides a good historical



Privacy-Preserving Data Mining. Figure 1. Using a privacy-preserving outlier detection algorithm to determine if an algorithm isolates terrorist behavior.

framework for the adoption of privacy-preserving data mining.

Cross-references

- ▶ Inference Control in Statistical Databases
- ▶ Privacy Metrics
- ▶ Randomized Methods to ensure Data Privacy
- ▶ Secure multiparty Computation Methods
- ▶ Statistical Disclosure Limitation for Data Access

Recommended Reading

1. Agrawal R. and Srikant R. Privacy-preserving data mining. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 439–450
2. Atallah M.J., Elmongui H.G., Deshpande V., and Schwarz L.B., Secure supply-chain protocols. In Proc. IEEE Int. Conf. on E-Commerce, 2003, pp. 293–302.
3. Kaski S. Dimensionality reduction by random mapping. In Proc. Int. Joint Conference on Neural Networks, 1999, pp. 413–418.
4. Lindell Y. and Pinkas B. Privacy preserving data mining. In Advances in Cryptology – CRYPTO 2000. Springer, 2000, pp. 36–54.
5. Oliveira S.R.M. and Zaïane O.R. Privacy preserving clustering by data transformation. In Proc. 18th Brazilian Symp. on Databases, 2003.
6. Vaidya J. and Clifton C. Privacy-preserving outlier detection. In Proc. 2004 IEEE Int. Conf. on Data Mining, 2004, pp. 233–240.
7. Vaidya J., Clifton C., and Zhu M. Privacy Preserving Data Mining. ser. Springer, Berlin, 2006.

Privacy-Preserving Spatial Queries

▶ Spatial Anonymity

Probabilistic Analysis

▶ Performance Analysis of Transaction Processing Systems

Probabilistic Data

▶ Data Uncertainty Management in Sensor Networks

Probabilistic Databases

DAN SUCIU
University of Washington, Seattle, WA, USA

Synonyms

Uncertain databases

Definition

A probabilistic database is a database in which every tuple t belongs to the database with some probability $\mathbf{P}(t)$; when $\mathbf{P}(t) = 1$ then the tuple is certain to belong to the database; when $0 < \mathbf{P}(t) < 1$ then it belongs to the database only with some probability; when $\mathbf{P}(t) = 0$ then the tuple is certain not to belong to the database, and it is not necessary to bother representing it. A traditional (deterministic) database corresponds to the case when $\mathbf{P}(t) = 1$ for all tuples t. Tuples with $\mathbf{P}(t) > 0$ are called *possible tuples*. In addition to indicating the probabilities for all tuples, a probabilistic database must also indicate somehow how the tuples are correlated. In the simplest cases the tuples are declared to be either independent (when $\mathbf{P}(t_1 t_2) = \mathbf{P}(t_1 t_2)$), or exclusive (or disjoint, when $\mathbf{P}(t_1 t_2) = 0$).

Historical Background

Extensions of databases to handle probabilistic data have been considered since the 1980s. In 1992, Barbara, Garcia-Molina, and Porter [6] developed a probabilistic data model that is an extension of the relational data model, in which relations have deterministic keys, and non-key attributes that are stohastic. They also introduced probabilistic analogs to the relational algebraic operators, later called *extensional operators*. Queries were restricted to return all deterministic keys of relations mentioned in the query body. Efforts to remove this restriction on queries lead to several extensions. In 1997, Lakshmanan, Leone, Ross, and Subrahmanian [19] modified the query semantics and proposed that probabilities be combined by user-defined *strategies*. In 1997, Fuhr and Roellke [17] proposed an alternative way to compute the correct semantics for all queries, using *intensional semantics*, by which every tuple in the query answer is annotated with a propositional formula called *event expression*: a general-purpose probabilistic inference algorithm can be used to compute the tuple's probability from the event expression, but this is known to be expensive in general. Dalvi and Suciu in 2004 [12] showed that some queries can be computed efficiently using extensional operators if these are ordered carefully, to ensure that they are applied correctly: a plan consisting of correctly applied operators is called a *safe plan*. Not all queries admit a safe plan, but those that do not admit one can be shown to be #P-hard, thus they cannot be evaluated efficiently unless $P = NP$. Recent systems for managing probabilistic or incomplete databases include Trio [8,24], MystiQ [20], and MayBMS [4].

Foundations

For an illustration, consider the probabilistic database in Fig. 1. It has three tables, of which `Product`^{p} and `Customer`^{p} are probabilistic and `Order`^{p} is deterministic. `Product`^{p} contains three products; their names and their prices are known, but their color and shape are unknown. For example `Gizmo` may be have `color` = `red` and `shape` = `oval` with probability $p_1 = 0.25$, or may have `color` = `blue` and `shape` = `square` with probability $p_2 = 0.75$ respectively. `Camera` has three possible combinations of `color` and `shape`, and `IPod` has two. In other words, for every product in the database, the values for `color` and `shape`, are not known, but instead there is a probability space over their values. Each color-shape combination should exclude the others, so $p_1 + p_2 \leq 1$, $p_3 + p_4 + p_5 \leq 1$ and $p_6 + p_7 \leq 1$, holds for our table. When the sum is strictly less than one then that product may not occur in the table at all: for example `Camera` may be missing from the table with probability 1-p_3-p_4-p_5.

Representation and Smantics

A popular representation of a probabilistic database consists of a regular database with two additional pieces of information: for each probabilistic table a distinguished attribute **P** is designated as the probability attribute, and a set of attributes is designated as *possible worlds key*. The key is interpreted as follows. If two tuples have the same values of the key attributes, then they are exclusive, i.e., only one of them may actually occur in the database. Otherwise, if the values of their key attributes differ then the tuples are independent. Tuples from different tables are always assumed to be independent. Figure 2a illustrates such a representation. Here the attribute **P** is designated as the probability attribute, and the two attributes `prod` and `price` are the possible worlds key. There are seven possible tuples, but at most three of them may actually occur in the database, because the first two are exclusive tuples, the next three are exclusive, and the last two are exclusive.

Tuples with the same possible worlds key are sometimes called *xor-tuples* or *x-tuples*. The entire set of such tuples is called a *maybe tuple*. The three `Camera`

`Product`^{P}

Prod	Price	Color	Shape	P
Gizmo	20	red	oval	$p_1 = 0.25$
		blue	square	$p_2 = 0.75$
Camera	80	green	oval	$p_3 = 0.3$
		red	round	$p_4 = 0.3$
		blue	oval	$p_5 = 0.2$
IPod	300	white	square	$p_6 = 0.8$
		black	square	$p_7 = 0.2$

`Order`

Prod	Price	Cust
Gizmo	20	Sue
Gizmo	80	Fred
IPod	300	Fred

`Customer`^{P}

Cust	City	P
Sue	NewYork	$q_1 = 0.5$
	Boston	$q_2 = 0.2$
	Seattle	$q_3 = 0.3$
Fred	Boston	$q_4 = 0.4$
	Seattle	$q_5 = 0.3$

Probabilistic Databases. Figure 1. A probabilistic database.

Product ^{p}

Prod	Price	Color	Shape	P
Gizmo	20	red	oval	p_1
Gizmo	20	blue	square	p_2
Camera	80	green	oval	p_3
Camera	80	red	round	p_4
Camera	80	blue	oval	p_5
IPod	300	white	square	p_6
IPod	300	black	square	p_7

a

Product$_1$

Prod	Price	Color	Shape
Gizmo	20	red	oval
Camera	80	blue	oval
IPod	300	white	square

P(Product$_1$) = $p_1 p_5 p_6$

Product$_2$

Prod	Price	Color	Shape
Gizmo	20	blue	square
Camera	80	blue	oval
IPod	300	white	square

P(Product$_2$) = $p_2 p_5 p_6$

Product$_3$

Prod	Price	Color	Shape
Gizmo	20	red	oval
IPod	300	white	square

b **P**(Product$_3$) = $p_1(1{-}p_3{-}p_4{-}p_5)p_6$

Probabilistic Databases. Figure 2. A representation of a probabilistic table (a): it shows the probability of each tuple, and the key of the possible worlds: if two tuples share the same key values then they are exclusive, otherwise they are independent. Three possible worlds (b).

tuples in Fig. 2a are x-tuples, and together they form one maybe-tuple.

The semantics of a probabilistic database is a probability space over *possible worlds*. Product^p has 16 possible worlds, since there are two choices for Gizmo, four for Camera (including removing Camera altogether) and two for IPod. Figure 2b illustrates three of these sixteen possible worlds and their probabilities.

Query Evaluation

The semantics of a query q on a probabilistic database is defined as follows. Consider all possible worlds of the probabilistic database: each such world is a standard, deterministic database. Evaluate q on each possible world, using the standard semantics of a query on a deterministic database. Any tuple that is an answer to the query in some possible world is called a *possible answer tuple*. The *probability of a possible answer tuple* is defined as the sum of the probabilities of all worlds where the tuple is an answer. The semantics of the query is the set of possible answer tuples together with their probabilities.

A central problem in probabilistic databases is query evaluation. It is infeasible to enumerate all possible worlds, because their number is exponential is the size of the database. There are two approaches to query evaluation: extending relational algebra operators to manipulate probabilities explicitly, or using a general-purpose probabilistic inference algorithm. The first approach is rather similar to standard query processing, and is quite efficient: the declarative SQL query is translated into a relational algebra plan, called an *extensional plan*, then this plan is executed. We illustrate this below. However not all queries admit correct extensional plans. Those that do not admit such plans require a more expensive, general-purpose probabilistic inference, which have been studied in the context of knowledge representation systems and are not be discussed here.

The three main extensional operators on the three queries are illustrated in Fig. 3, then show how to combine them for more complex queries. The left column in the figure shows the queries in SQL syntax, the right column shows the same queries in datalog notation. In datalog, underline the variables that occur in the key positions. The first query, Q_1, asks for all the oval products in the database, and it returns two possible answer tuples:

prod	price	P
Gizmo	20	p_1
Camera	80	$p_3 + p_5$

For example the probability of the possible answer tuple Gizmo is the sum of the probabilities of the

```
Q1  : SELECT DISTINCT prod, price
      FROM Product^P                          Q1(x,y) : -     Product^P(x,y,'oval',z)
      WHERE shape = 'oval'

Q2  : SELECT DISTINCT city
      FROM Customer^P                         Q2(y)   : -     Customer^P(x,y)

Q3  : SELECT DISTINCT*
      FROM Product^P, Order, Customer^P       Q3(*)   : -     Product^P(x,y,z),
      WHERE Product^P.prod = Order.prod                       Order(x,y,u)
      and Product^P.price = Order.price                       Customer^P(u,v)
      and Order.cust = Customer^P.cust
```

Probabilistic Databases. Figure 3. Three simple queries, expressed in SQL and in datalog.

8 possible worlds for `Product` (out of 16) where `Gizmo` appears as `oval`, and this turns out (after simplifications) to be p_1. These probabilities can be computed without enumerating all possible worlds, directly from the table in Fig. 2a by the following process: (i) Select all rows with `shape='oval'`, (ii) project on `prod`, `price`, and **P** (the probability), (iii) eliminate duplicates, by replacing their probabilities with the sum, because they are disjoint events. The latter two steps are called a disjoint project:

Disjoint Project, $\pi_{\bar{A}}{}^{pD}$ If k tuples with probabilities $p_1,\ldots,p_k$ have the same value, $\bar{a}$, for their $\bar{A}$ attributes, then the disjoint project will associated the tuple $\bar{a}$ with the probability $p_1 + \ldots + p_k$. The disjoint project is correctly applied if any two tuples that share the same values of the $\bar{A}$ attributes are disjoint events.

Q_1 can therefore be computed by the following plan:

$$Q_1 = \pi^{pD}_{\texttt{prod,price}}(\sigma_{\texttt{shape='oval'}}(\texttt{Product}^p))$$

$\pi^{pD}_{\texttt{prod,price}}$ is correctly applied, because any two tuples in $\texttt{Product}^{\mathrm{P}}$ that have the same `prod` and `price` are disjoint events.

Query Q_2 in Fig. 3 asks for all cities in the `Customer` table, and its answer is:

city	P
New York	q_1
Boston	$1-(1-q_2)(1-q_4)$
Seattle	$1-(1-q_3)(1-q_5)$

This answer can also be obtained by a projection with a duplicate elimination, but now the probabilities p_1, p_2, p_3,... of duplicate values are replaced with $1 - (1 - p_1)(1 - p_2)(1 - p_3)\ldots$, since in this case all duplicate occurrences of the same city are independent. This is called an independent project:

Independent Project, $\pi^{pI}_{\bar{A}}$ If k tuples with probabilities $p_1,\ldots,p_k$ have the same value, $\bar{a}$, for their $\bar{A}$ attributes, then the independent project will associated the tuple $\bar{a}$ with the probability $1 - (1 - \mathrm{p}_1)(1 - p_2)\ldots(1 - p_\mathrm{k})$. The independent project is correctly applied if any two tuples that share the same values of the $\bar{A}$ attributes are independent events.

Thus, the disjoint project and the independent project compute the same set of tuples, but with different probabilities: the former assumes disjoint probabilistic events, where $\mathbf{P}(t \vee t') = \mathbf{P}(t) + \mathbf{P}(t')$, while the second assumes independent probabilistic events, where $\mathbf{P}(t \vee t') = 1 - (1 - \mathbf{P}(t))(1 - \mathbf{P}(t'))$. Continuing our example, the following plan computes Q_2: $Q_2 = \pi^{pI}_{\texttt{city}}(\texttt{Customer}^{\mathrm{P}})$. Here $\pi^{pI}_{\texttt{city}}$ is correctly applied because any two tuples in $\texttt{Customer}^{\mathrm{P}}$ that have the same `city` are independent events.

Query Q_3 in Fig. 3 illustrates the use of a join, and its answer is:

prod	price	color	shape	cust	city	P
Gizmo	20	red	oval	Sue	New York	p_1q_1
Gizmo	20	red	oval	Sue	Boston	p_1q_2
Gizmo	20	red	oval	Sue	Seattle	p_1q_3
Gizmo	20	blue	square	Sue	New York	p_2q_1
...	...					

It can be computed by modifying the join operator to multiply the probabilities of the input tables:

Join, $\bowtie^p$ Whenever it joins two tuples with probabilities p_1 and p_2, it sets the probability of the resulting tuple to be p_1p_2.

A plan for Q_3 is: $Q_3 = \texttt{Product} \bowtie^{p} \texttt{Order} \bowtie^{p} \texttt{Customer}$.

Extensional operators can be combined to form an extensional plan. A plan is called *safe* if each operator is applied correctly. It is possible to illustrate with the following probabilistic database schema:

$\texttt{Product}^p(\underline{\texttt{prod, price}}, \texttt{color, shape})$
$\texttt{Order}^p(\underline{\texttt{prod, price, cust}})$

Both tables are probabilistic. The following query finds all colors of products ordered by the customer named Sue:

$$Q_4(c) : -\texttt{Product}^p(\underline{x, y}, c, s), \texttt{Order}^p(\underline{x, y, Sue})$$

Q_4 admits the following safe plan:

$$Q_4 = \Pi^{pI}_{\texttt{color}}(\Pi^{pD}_{\texttt{prod,price,color}}(\texttt{Product}^p) \bowtie^p \sigma_{\texttt{cust='Sue'}}(\texttt{Order}^p))$$

This is safe because $\Pi^{pD}_{\texttt{prod,price,color}}$ combines only disjoint tuples, while Π^{pI}_{color} combines only independent tuples. Similarly, the join operators combines independent tuples, since the left operand consists of tuples combined from the $\texttt{Product}^p$ table while the right operand consists of tuples from the $\texttt{order}^p$ table.

In contrast, the following plan is not safe:

$$\Pi^{pI}_{\text{color}}(\texttt{Product}^p \bowtie^p \sigma_{\texttt{cust='Sue'}}(\texttt{Order}^p))$$

because the outermost projection Π^{I}_{color} will combine tuples that are not independent: there may be two or more distinct tuples in the join $\texttt{Product}^p \bowtie^p \texttt{Order}^p$ that have the same $\texttt{Product}^p$ tuple, hence they are not independent.

Some queries do not admit any safe plans at all. Figure 4 illustrates three queries that are known not to admit any safe plan. All three are boolean queries, i.e., they return either "true" or nothing, but they still have a probabilistic semantics, and it is necessary to compute the probability of the answer "true." No combination of extensional operators result in a safe plan for any of these queries. For example $\pi^{pI}_{\emptyset}(\texttt{R} \bowtie \texttt{S} \bowtie \texttt{T})$ is an incorrect plan for H_1, because two distinct rows in $\texttt{R} \bowtie \texttt{S} \bowtie \texttt{T}$ may share the same tuple in R, hence they are not necessarily independent events. In fact it has been shown that the complexity of each of these three queries is #P-hard, hence there is no polynomial time algorithm for computing their probabilities unless $P = NP$.

Key Applications

Probabilistic databases are design to allow uncertain data to be managed directly by a relational database system. Several types of applications have been considered.

In *Information Extraction* the goal is to extract structured data from a collection of unstructured text documents. Examples include address segmentation, citation extraction, extractions for comparison shopping, hotels, restaurant guides, etc. The schema is given in advance by the user, and the extractor is tailored to that specific schema. All approaches to extraction are imprecise, and most often can associate a probability score to each item extracted. A probabilistic database allows these probability scores to be stored natively.

In *Fuzzy Object Matching* the problem is to reconcile objects from two collections that have used different naming conventions. This is a central problem in data cleaning and data integration, and has also been called record linkage, or de-duplication. The basic approach is to compute some similarity score between pairs of objects, usually by starting from string similarity scores on their attributes then combining these

```
Schema: R(A),S(A,B), T(B)
H1: SELECT DISTINCT 'true' AS A          H1  : - R(x),S(x,y),T(y)
    FROM R, S, T
    WHERE R.A = S.A and S.B = T.B

Schema: R(A,B), S(B)
H2: SELECT DISTINCT 'true' AS A          H2  : - R(x,y),S(y)
    FROM R, S
    WHERE R.B = S.B

Schema: R(A,B), S(A, B)
H3: SELECT DISTINCT 'true' AS A          H3  : - R(x,y),S(x,y)
    FROM R, S
    WHERE R.A = S.A and R.B = S.B
```

Probabilistic Databases. Figure 4. Three queries that do not admit safe plans, and that are known to be #P-complete.

scores into a global score for the pair of objects. Thus, the result of fuzzy object matching is inherently probabilistic. In traditional data cleaning it needs to be determinized somehow, e.g., by comparing the similarity score with a threshold and classifying each pair of object into a *match*, a *non-match*, and a *maybe-match*. A probabilistic database allows users to keep the similarity scores natively.

Other applications include: management of sensor data and of RFID data, probabilistic data integration, probabilistic schema matches, and flexible query interfaces.

Cross-references

▶ Data Uncertainty Management in Sensor Networks
▶ Incomplete information
▶ Inconsistent databases
▶ Uncertainty Management in Scientific Database Systems

Recommended Reading

1. Adar E. and Re C. Managing uncertainty in social networks. IEEE Data Eng. Bull., 30(2):15–22, 2007.
2. Andritsos P., Fuxman A., and Miller R.J. Clean answers over dirty databases. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
3. Antova L., Koch C., and Olteanu D. 10ˆ(10ˆ6) worlds and beyond: Efficient representation and processing of incomplete information. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
4. Antova L., Koch C., and Olteanu D. MayBMS: Managing incomplete information with probabilistic world-set decompositions (demonstration). In Proc. 23rd Int. Conf. on Data Engineering, 2007.
5. Antova L., Koch C., and Olteanu D. World-set decompositions: expressiveness and efficient algorithms. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 194–208.
6. Barbara D., Garcia-Molina H., and Porter D. The management of probabilistic data. IEEE Trans. Knowl. Data Eng., 4(5):487–502, 1992.
7. Benjelloun O., Das Sarma A., Halevy A., and Widom J. ULDBs: databases with uncertainty and lineage. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 953–964.
8. Benjelloun O., Das Sarma A., Hayworth C., and Widom J. An introduction to ULDBs and the Trio system. IEEE Data Eng. Bull., 29(1):5–16, 2006.
9. Cheng R., Kalashnikov D., and Prabhakar S. Evaluating probabilistic queries over imprecise data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 551–562.
10. Cheng R. and Prabhakar S. Managing uncertainty in sensor databases. ACM SIGMOD Rec., 32(4):41–46, December 2003.
11. Dalvi N., Re C., and Suciu D. Query evaluation on probabilistic databases. IEEE Data Eng. Bull., 29(1):25–31, 2006.
12. Dalvi N. and Suciu D. Efficient query evaluation on probabilistic databases. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
13. Dalvi N. and Suciu D. Management of probabilistic data: foundations and challenges. In Proc. 26th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2007, pp. 1–12. (invited talk).
14. Das Sarma A., Benjelloun O., Halevy A., and Widom J. Working models for uncertain data. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
15. Deshpande A., Guestrin C., Madden S., Hellerstein J.M., and Hong W. Model-driven data acquisition in sensor networks. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 588–599.
16. Deshpande A., Guestrin C., Madden S., Hellerstein J.M., and Hong W. Using probabilistic models for data management in acquisitional environments. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 317–328.
17. Fuhr N. and Roelleke T. A probabilistic relational algebra for the integration of information retrieval and database systems. ACM Trans. Inf. Syst., 15(1):32–66, 1997.
18. Gupta R. and Sarawagi S. Creating probabilistic databases from information extraction models. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 965–976.
19. Lakshmanan L., Leone N., Ross R., and Subrahmanian V.S. Probview: a flexible probabilistic database system. ACM Trans. Database Syst., 22(3), 1997.
20. MystiQ: a probabilistic database system, http://mystiq.cs.washington.edu/.
21. Re C. and Suciu D. Materialized views in probabilistic databases for information exchange and query optimization. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
22. Re C., Letchner J., Balazinska M., and Suciu D. Event queries on correlated probabilistic streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2008.
23. Re C. and Suciu D. Approximate lineage for probabilistic databases. In Proc. 34th Int. Conf. on Very Large Data Bases, 2008.
24. Widom J. Trio: A system for integrated management of data, accuracy, and lineage. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 262–276.

Probabilistic Model

▶ BM25
▶ Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model

Probabilistic Model of Indexing

▶ Two-Poisson model

Probabilistic Querying

▶ Data Uncertainty Management in Sensor Networks

Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model

Thomas Roelleke[1], Jun Wang[1], Stephen Robertson[2]
[1]Queen Mary University of London, London, UK
[2]Microsoft Research Cambridge, Cambridge, UK

Synonyms

BIR model; Probabilistic model; RSJ model

Definition

Information retrieval (IR) systems aim to retrieve relevant documents while not retrieving non-relevant ones. This can be viewed as the foundation and justification of the binary independence retrieval (BIR) model, which proposes to base the ranking of documents on the division of the probability of relevance and non-relevance.

For a set r of relevant documents, and a set $\bar{r}$ of non-relevant documents, the BIR model defines the following term weight and retrieval status value (RSV) for a document-query pair "d, q":

$$\mathrm{birw}(t, r, \bar{r}) := \frac{P(t|r) \cdot P(\bar{t}|\bar{r})}{P(t|\bar{r}) \cdot P(\bar{t}|r)} \qquad (1)$$

$$\mathrm{RSV}_{\mathrm{BIR}}(d, q, r, \bar{r}) := \sum_{t \in d \cap q} \log \mathrm{birw}(t, r, \bar{r}) \qquad (2)$$

Here, $P(t|r)$ is the probability that term t occurs in the relevant documents, and $P(t|\bar{r})$ is the respective probability for term t in non-relevant documents.

There are two ways to estimate the term probabilities: (I_1) $P(t|r) := \frac{r_t}{R}$ and $P(t|\bar{r}) := \frac{n_t}{N}$, or (I_2) $P(t|r) := \frac{r_t}{R}$ and $P(t|\bar{r}) := \frac{n_t - r_t}{N - R}$. R denotes the number of relevant documents, and N denotes the number of all documents; r_t denotes the number of relevant documents containing term t, and n_t denotes the respective number of all documents. I_1 assumes term independence in the relevant documents and in *all* documents; I_2 assumes term independence in the relevant documents and in the *non-relevant* documents.

Given the two independence assumptions, and given the option to consider term presence only, there are four forms/variations of the BIR term weight. Further, the BIR model proposes a technique for dealing with missing relevance, where a zero probability problem needs to be avoided. The definitions given next show the case referred to as F4 where I_2 is assumed and term absence is considered. The constant 0.5 caters for missing relevance data.

$$\mathrm{birw}_{+0.5}(t, r, \bar{r}) := \frac{(r_t + 0.5)/(R - r_t + 0.5)}{(n_t - r_t + 0.5)/(N - R - (n_t - r_t) + 0.5)} \qquad (3)$$

$$\mathrm{RSV}_{\mathrm{BIR,\,F4}}(d, q, r, \bar{r}) := \sum_{t \in d \cap q} \log \mathrm{birw}_{+0.5}(t, r, \bar{r}) \qquad (4)$$

The derivation of the term weight and the retrieval status value is to be found in the scientific fundamentals of this entry.

Historical Background

The paper [10] on "Relevance Weighting of Search Terms" proposed what became known as the probabilistic retrieval model, binary independence retrieval model, or RSJ (initials of the authors) model.

Closely related is the "The Probability Ranking Principle in IR" [8] to justify that a ranking based on the probability of relevance is an optimal ranking, since the ranking minimises the costs for browsing the ranking.

Reference [2] on "Using Probabilistic Models of Document Retrieval without Relevance Information" investigated the generalisation for missing relevance, and [4] contributed "An Evaluation of Feedback in Document Retrieval using Co-occurrence Data."

Foundations

Derivation of the BIR Model

Derivations of the BIR model can be found in [6], [3] (pp. 21–31), and [5] (pp. 167–175).

The BIR model is derived from the odds of the probability of relevance:

$$O(r|d, q) := \frac{P(r|d, q)}{P(\bar{r}|d, q)} = \frac{P(r|d, q)}{\mathrm{I} - P(r|d, q)} \qquad (5)$$

Here, $P(r|d, q)$ is the probability that the event relevant (r) occurs for a given document-query pair. $P(\bar{r}|d, q)$ is the respective probability for non-relevant.

The derivation of the BIR model can be represented in six steps:

1. Bayes theorem and reduction of $O(r|d, q)$.
2. Representation of d as a vector $\vec{x}$ of binary term features (the binary term features are "term does occur" and "term does not occur").
3. Independence assumption for features.
4. Split product over features into two products: a product for term presence, and a product for term absence.
5. Non-query term assumption: if non-query terms occur with the same frequency in relevant and non-relevant documents, then they do not affect the ranking.
6. Rewrite the expression to obtain a compact form.

Bayes Theorem To estimate the unknown probability $P(r|d, q)$, the theorem of Bayes is applied:

$$P(r|d,q) := \frac{P(r|d,q)}{P(d,q)} = \frac{P(d|r,q)\cdot P(r|q)\cdot P(q)}{P\,(d,q)} \tag{6}$$

For $O(r|d, q)$, the probabilities $P(q)$ and $P(d, q)$ drop out. With respect to the ranking of documents, the factor $\frac{P(r|q)}{P(\bar{r}|q)}$ has no effect, since it is document-independent. Thus, the ranking of documents is based on the following ranking equivalence:

$$O(r|d,q) \overset{rank}{=} \frac{P(d|r,q)}{P(d|\bar{r},q)} \tag{7}$$

Binary Feature Vector $\vec{x}$ The next step concerns the representation and decomposition of the document d. For this, the BIR model assumes that d is a vector of binary features; this vector is denoted $\vec{x} = (x_1,\ldots,x_n)$. Each x_i corresponds to a term t_i, and for all x_i, $x_i = 1$ if term is present, and $x_i = 0$ if term is absent.

Independence Assumption Next, the BIR model assumes that the binary features are independent. This leads to the following equation for $P(d|r, q)$:

$$P(d|r,q) = P(\vec{x}|r,q) = \prod_{x_i} P(x_i|r,q) \tag{8}$$

The equation for $P(d|\bar{r}, q)$ is accordingly.

Product Split The product over all features is split into two products: one product for term presence ($x_i = 1$), and one for absence ($x_i = 0$).

$$P(d|r,q) = \left[\prod_{x_i=1} P(x_i|r,q)\right]\cdot\left[\prod_{x_i=0} P(x_i|r,q)\right] \tag{9}$$

Since $x_i = 1$ corresponds to "term occurs," and $x_i = 0$ corresponds to "term does not occur," the next equation replaces x_i by the term event t, where t means "term occurs."

$$P(d|r,q) = \left[\prod_{t\in d} P(t|r,q)\right]\cdot\left[\prod_{t\notin d} P(\bar{t}|r,q)\right] \tag{10}$$

In classical literature [10,6], the symbols p_i and q_i are employed instead of $P(t|r, q)$ and $P(\bar{t}|r, q)$; however, since the probabilities are more explicit, and avoid a confusion of query (q) and absence of term (q_i), and facilitate to relate the BIR model to other retrieval modes, the probabilistic notion is chosen here, while keeping in mind that $P(t|r, q)$ refers to the binary feature of term t.

Non-Query Term Assumption For all non-query terms, it is assumed that their frequency in relevant documents is equal to their frequency in non-relevant documents, i.e., $\forall t \notin q : P(t|r, q) = P(t|\bar{r}, q)$. This leads to a reduction of the product, i.e., the non-query terms are dropped.

$$\frac{P(d|r,q)}{P(d|\bar{r},q)} = \left[\prod_{t\in d\cap q} \frac{P(t|r,q)}{P(t|\bar{r},q)}\right]\cdot\left[\prod_{t\in q\setminus d} \frac{P(\bar{t}|r,q)}{P(\bar{t}|\bar{r},q)}\right] \tag{11}$$

This assumption corresponds to $\frac{P(t|r,q)}{P(t|\bar{r},q)} = 1$, and the more general assumption is $\frac{P(t|r,q)}{P(t|\bar{r},q)} = c$, where c is a constant.

Rewriting to Achieve Compact Form Finally, a rewriting leads to a compact form. The rewriting is based on the following equation:

$$1.0 = \prod_{t\in d\cap q}\left[\frac{P(\bar{t}|\bar{r},q)}{P(\bar{t}|r,q)}\cdot\frac{P(\bar{t}|r,q)}{P(\bar{t}|\bar{r},q)}\right] \tag{12}$$

Equation 11 is multiplied with this 1.0. This fills up the right product over query-only terms to a product over all query terms; then, this product is document-independent, and it can be dropped for ranking purpose, as the next equations illustrate.

$$O(r|d,q) = \left[\prod_{t \in d \cap q} \frac{P(t|r,q)}{P(t|\bar{r},q)}\right] \cdot \left[\prod_{t \in q \setminus d} \frac{P(\bar{t}|r,q)}{P(\bar{t}|\bar{r},q)}\right] \cdot$$
$$\left[\prod_{t \in d \cap q} \frac{P(\bar{t}|\bar{r},q)}{P(\bar{t}|r,q)} \cdot \frac{P(\bar{t}|r,q)}{P(\bar{t}|\bar{r},q)}\right]$$
$$\overset{rank}{=} \prod_{t \in d \cap q} \frac{P(t|r,q) \cdot P(\bar{t}|\bar{r},q)}{P(t|\bar{r},q) \cdot P(\bar{t}|r,q)} \tag{13}$$

The fractional expression is referred to as the BIR term weight, where either the above form or the logarithm of it may be viewed as the term weight. The following definition follows the first approach, and the logarithm is applied to the term weight in the definition of $\mathrm{RSV}_{\mathrm{BIR}}$.

$$\mathrm{birw}(t,r,\bar{r}) := \frac{P(t|r,q) \cdot P(\bar{t}|\bar{r},q)}{P(t|\bar{r},q) \cdot P(\bar{t}|r,q)} \tag{14}$$

$$\mathrm{RSV}_{\mathrm{BIR}}(d,q,r,\bar{r}) := \sum_{t \in d \cap q} \log \mathrm{birw}(t,r,\bar{r}) \tag{15}$$

The BIR weight is greater than 1.0 for *good* terms, i.e., good terms occur more frequently in relevant documents than in non-relevant documents. For $P(t|r,q) = P(t|\bar{r},q)$, the BIR weight is 1.0; thus, such terms have no effect on the ranking. For *poor terms*, the BIR weight is less than 1.0; thus, poor terms have a negative effect on the $\mathrm{RSV}_{\mathrm{BIR}}$.

Alternative Derivation An alternative, shorter derivation is shown next. The abbreviations $p_i := P(x_i = 1|r)$ and $q_i := P(x_i = 1|\bar{r})$ are applied, and the document probabilities are expressed as follows:

$$\frac{P(d|r)}{P(d|\bar{r})} = \prod_i \frac{p_i^{x_i} \cdot (1-p_i)^{1-x_i}}{q_i^{x_i} \cdot (1-q_i)^{1-x_i}} \tag{16}$$

The exponent $x_i = 1$ selects the probability p_i that the term occurs, and $x_i = 0$ selects the probability $(1 - p_i)$ that the term does not occur. Resolving the exponent $1 - x_i$ leads to the next equation and ranking equivalence.

$$\frac{P(d|r)}{P(d|\bar{r})} = \left[\prod_i \left(\frac{p_i}{1-p_i} \cdot \frac{1-q_i}{q_i}\right)^{x_i}\right] \cdot \prod_i \frac{1-p_i}{1-q_i} \tag{17}$$

$$\overset{rank}{=} \prod_{t \in d \cap q} \mathrm{birw}(t,r,\bar{r}) \tag{18}$$

The right product ($\prod_i \frac{1-p_i}{1-q_i}$ in 17) is constant, and therefore does not affect the ranking. Since $x_i = 0$ for non-document terms, the first product reduces to $x_i = 1$ in d. With $p_i = q_i$ for non-query terms, the product reduces to $x_i = 1$ in d and q, i.e., the product is over $t \in d \cap q$.

Estimation of Term Probabilities

For describing the estimation of term probabilities, the following notation is employed:

Traditional notation	Event-based notation	Comment
r_t	$n_D(t, r)$	Number of relevant documents with term t
R	$N_D(r)$	Number of relevant documents
n_t	$n_D(t, c)$	Number of documents with term t in the collection c
N	$N_D(c)$	Number of documents in the collection c
p_t	$P_D(t\|r)$	Term probability in relevant documents
q_t	$P_D(t\|\bar{r})$	term probability in non-relevant documents

The notation comprises the traditional symbols, and it shows an alternative, namely an event-based notation. The event-based notation is explicit regarding the event-space (D for documents). Therefore, it is applicable in a dual way to document-based and location-based event spaces. Also, events such as a document, a collection, and any set of documents (relevant, non-relevant, retrieved, all) can be referred to systematically.

The estimates of the term probabilities are given next:

Traditional notation	Event-based notation	Comment
$p_t = \frac{r_t}{R}$	$P_D(t\|r) = \frac{n_D(t,r)}{N_D(r)}$	Probability of term in relevant
$q_t = \begin{cases} \frac{n_t}{N} & I_1 \\ \frac{n_t - r_t}{N-R} & I_2 \end{cases}$	$P_D(t\|\bar{r}) = \begin{cases} \frac{n_D(t,c)}{N_D(c)} & I_1 \\ \frac{n_D(t,c)-n_D(t,r)}{N_D(c)-N_D(r)} & I_2 \end{cases}$	Probability of term in non-relevant assumptions I_1 and I_2

The I_2 assumption and the consideration of term absence lead to the following equation for the BIR term weight:

$$\mathrm{birw}(t, r, \bar{r}) = \frac{r_t/(R - r_t)}{(n_t - r_t)/(N - R - (n_t - r_t))} \tag{19}$$

The next section groups the variations of the term weight.

Variations of the BIR Term Weight

[10] introduced four variations (forms) of the term weights: the four variations (referred to as F1, F2, F3, and F4) follow from the two estimates for $P(t|\bar{r})$, and whether or not term absence is considered.

Assumption	Term absence	Variation	Term Weight
I_1	no	F1	$\frac{r_t/R}{n_t/N}$
I_2	no	F2	$\frac{r_t/R}{(n_t-r_t)/(N-R)}$
I_1	yes	F3	$\frac{r_t/(R-r_t)}{n_t/(N-n_t)}$
I_2	yes	F4	$\frac{r_t/(R-r_t)}{(n_t-r_t)/(N-R-(n_t-r_t))}$

For further reading, the BIR term weights are summarised in [3], page 27. The variations form a comprehensive mathematical coverage; F4 is the prime choice.

Solving the Zero Probability Problem

The BIR term weight is not defined for missing relevance; $R = r_t = 0$ leads to a division by zero. Therefore, [10] proposes to add constants to the frequency counts:

$$\mathrm{birw}_{+0.5}(t, r, \bar{r}) := \frac{(r_t + 0.5)/(R - r_t + 0.5)}{(n_t - r_t + 0.5)/(N - R - (n_t - r_t) + 0.5)} \tag{20}$$

The subscript +0.5 in $\mathrm{birw}_{+0.5}(t, r, \bar{r})$ marks this term weight to be different from the bare term weight $\mathrm{birw}(t, r, \bar{r})$ in 19; $\mathrm{birw}_{+0.5}(t, r, \bar{r})$ is also referred to as $\mathrm{rsj}(t, r, \bar{r})$. What is the explanation underlying this zero-probability "fix"?

To reach an explanation, insert $r_t + 0.5$ for each r_t, and insert $R + 1$ for each R. This corresponds to assuming that there is a *virtual* document that is relevant, and each term occurs in *half* of the relevant documents, i.e., $p_t = 0.5$. The virtual document is retrieved, i.e., $n_t + 1$. Finally, this requires to assume $N + 2$ documents, which corresponds to assuming that there are two virtual documents, one of which is relevant. The next equation illustrates the explanation via virtual documents:

$$\mathrm{birw}_{+0.5}(t, r, \bar{r}) := \frac{(r_t + 0.5)/(R + 1 - (r_t + 0.5))}{((n_t + 1) - (r_t + 0.5))/((N + 2) - (R + 1) - ((n_t + 1) - (r_t + 0.5)))} \tag{21}$$

This expanded form reduces to 20.

From a more general point of view, the estimate $P(t|r) := \frac{r_t + k}{R + K}$ is applied to cover the case for missing relevance, where $\frac{k}{K} = 0.5$, and $k = 0.5$, and $K = 1$, for $\mathrm{birw}_{+0.5}$. This formulation reminds of the Laplace law of succession. Since the notion of "half of a relevant document" has no intuition, the number of virtual documents could be scaled to four rather than two [13]. This leads to: $N + 4$, $n_t + 2$, $R + 2$, $r_t + 1$. Then, the BIR weight is as follows:

$$\mathrm{birw}_{+1}(t, r, \bar{r}) := \frac{(r_t + 1)/(R - r_t + 1)}{(n_t - r_t + 1)/(N - R - (n_t - r_t) + 1)} \tag{22}$$

Relationship between the BIR Model, IDF, and BM25

The BIR model can be viewed as a theoretical argument to support IDF-based retrieval.

[2] proposes for missing relevance to assume that for all term, $P(t|r)$ is the same. This leads to a co-ordination level component in the retrieval function:

$$\sum_{t \in d \cap q} \log \frac{P(t|r)}{1 - P(t|r)} + \sum_{t \in d \cap q} - \log \frac{P(t|\bar{r})}{1 - P(t|\bar{r})}$$

The left component expresses the co-ordination level match. For large N and $N >> n_t$, the right component is similar to the IDF component. i.e., $\sum_{t \in d \cap q} - \log \frac{n_t}{N - n_t} \approx \sum_{t \in d \cap q} idf(t, c)$. [11] investigates how these assumptions are affected in the case of little relevance information.

[9] reviews the relationship of the BIR model and IDF, and theoretical arguments for IDF. The relationship between BIR and IDF is based on the definition $idf(t, c) := -\log P_D(t|c)$ and the estimation of the BIR probability $P(t|\bar{r})$. By assuming

$P(t|\bar{r}) = P_D(t|c)$, the relationship is established, i.e., *idf* $(t, c) = -\log P(t|\bar{r})$.

[13] builds on this relationship and shows a fully *idf*-based formulation of the BIR term weight:

$$\log \text{birw}(t, r, \bar{r}) = \log \frac{P(t|r) \,.\, P(\bar{t}|\bar{r})}{P(t|\bar{r}) \,.\, P(\bar{t}|r)} \qquad (23)$$

$$= idf(t, \bar{r}) - idf(t, r) + idf(\bar{t}, r) - idf(\bar{t}, \bar{r}) \qquad (24)$$

$$\approx idf(t, c) - idf(t, r) + idf(\bar{t}, r) - idf(\bar{t}, c) \qquad (25)$$

This linear combination of *idf* values underlines the issue that the collection *c* is much larger than the set *r* of relevant documents ($|c| >> |r|$), and therefore the maximum-likelihood estimates may be not comparable, and may need to be normalised [13].

Ignoring the negated term events (dropping the term absence, see section 1, variations of BIR term weight) leads to the following simplified term weight:

$$\log \frac{P(t|r)}{P(t|\bar{r})} = idf(t, \bar{r}) - idf(t, r) \approx \\ idf(t, c) - idf(t, r) \qquad (26)$$

The formulation shows that the term weight is $idf(t, c)$, if $idf(t, r) = 0$. The latter is the case for terms that occur in *all* relevant documents, i.e., $P(t|r) = 1$. Therefore, from this simplified form of the BIR model, *idf*-based retrieval assumes $P(t|r) = 1$. On the other hand, the full formulation (25) assumes $idf(t, r) = idf(\bar{t}, r)$, i.e., $P(t|r) = 0.5$, for missing relevance.

Regarding BM25, the BIR term weight is in BM25 what IDF is in TF-IDF, i.e., in BM25, a TF-component is multiplied with the BIR F4 term weight, whereas in basic TF-IDF, a TF-component is multiplied with the IDF term weight.

Key Applications

The BIR model is applied to incorporate relevance feedback data into document ranking; this model has become a key foundation of retrieval models. In 2004, the BIR model served as a foundation for a probabilistic ranking of tuples for database queries [1].

Cross-references

▶ BM25
▶ Language Models
▶ TF*IDF

Recommended Reading

1. Chaudhuri S., Das G., Hristidis V., and Weikum G. Probabilistic ranking of database query results. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 888–899.
2. Croft W.B. and Harper D.J. Using probabilistic models of document retrieval without relevance information. J. Doc., 35:285–295, 1979.
3. Grossman D.A. and Frieder O. Information Retrieval. Algorithms and Heuristics, 2nd edn., volume 15 of The Information Retrieval Series. Springer, Berlin, 2004.
4. Harper D.J. and van Rijsbergen C.J. An evaluation of feedback in document retrieval using cooccurrence data. J. Doc., 34:189–216, 1978.
5. Richard K. Belew. Finding out about. Cambridge University Press, 2000.
6. van Rijsbergen C.J. Inform. Retr.. Butterworths, London, 2nd edn., 1979. http://www.dcs.glasgow.ac.uk/Keith/Preface.html.
7. Robertson S. On event spaces and probabilistic models in information retrieval. Inform. Retr. J., 8(2):319–329, 2005.
8. Robertson S.E. The probability ranking principle in IR. J. Doc., 33:294–304, 1977.
9. Robertson S.E. Understanding inverse document frequency: On theoretical arguments for idf. J. Doc., 60:503–520, 2004.
10. Robertson S.E. and Sparck Jones K. Relevance weighting of search terms. J. Am. Soc. Inform. Sci., 27:129–146, 1976.
11. Robertson S.E. and Walker S. On relevance weights with little relevance information. In Proc. 20th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1997, pp. 16–24.
12. Roelleke T. and Wang J. A parallel derivation of probabilistic information retrieval models. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 107–114.
13. de Vries A. and Roelleke T. Relevance information: a loss of entropy but a gain for IDF? In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 282–289.

Probabilistic Spatial Queries

REYNOLD CHENG, JINCHUAN CHEN
The University of Hong Kong, Hong Kong, China

Synonyms

Imprecise spatial queries

Definition

An uncertain item is defined as a range-limited probability density function (pdf) in a multi-dimensional space, which can model the uncertainty of location, sensor and biological data. Given a set of uncertain items, a probabilistic spatial query returns results augmented with probabilistic guarantees for the validity of answers. The impreciseness of query answers is an inherent property of these applications due to data uncertainty, unlike the techniques for approximate processing that trade accuracy for performance. New query definitions, processing and indexing techniques are required to handle these queries.

Historical Background

Data uncertainty is an inherent property in a number of important and emerging applications. Consider, for example, a habitat monitoring system used in scientific applications, where data such as temperature, humidity, and wind speed are acquired from a sensor network. Due to physical imperfection of the sensor hardware, the data obtained are often inaccurate [9]. Moreover, a sensor cannot report its value at every point in time, and so the system can only obtain data samples at discrete time instants. As another example, in the Global-Positioning System (GPS), the location collected from the GPS-enabled devices (e.g., PDAs) also has measurement and sampling error [16,14]. The location data transmitted to the system may further encounter some network delay. In biometric databases, the attribute values of the feature vectors stored are not exact [1]. Hence, the data collected in these applications are often imprecise, inaccurate, and stale.

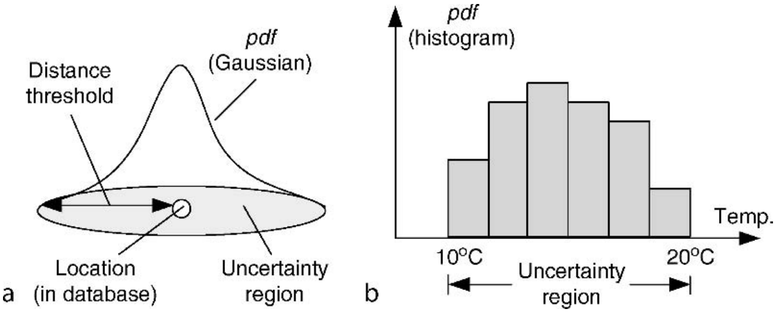
Services or queries that base their decisions on these data can produce erroneous results. There is thus a need to manage these data errors more carefully. In particular, the idea of *probabilistic spatial queries* (PSQ in short), which is a variant of spatial queries that handle data uncertainty, has been recently proposed. The main idea of a PSQ is to consider the models of the data uncertainty (instead of just the data value reported), and augment probabilistic guarantees to the query results. For example, a traditional query asking who is the nearest neighbor of a given point q can tell the user that John is the answer, while a PSQ informs the user that John has a probability of 0.8 of being the closest to q. The probabilities reflect the degree of correctness of query results, thereby facilitating the system to produce a more confident decision.

In this entry, the recent research efforts on PSQ will be summarized. Specifically, the details of how a PSQ can be classified will be discussed. Then, the issues of evaluating and indexing different types of PSQ in a large database will be addressed.

Foundations

Spatial Uncertainty Models To understand a PSQ, it is important to first discuss a commonly-used model of data uncertainty. This model assumes that the actual data value is located within a closed region, called the *uncertainty region*. In this region, a non-zero probability density function (*pdf*) of the value is defined, where the integration of pdf inside the region is equal to one. The cumulative density function (*cdf*) of the item is also provided. In an LBS, a normalized Gaussian pdf is used to model the measurement error of a location stored in a database [16,14] (Fig. 1a). The uncertainty region is a circular area, with a radius called the "distance threshold"; the newest location is reported to the system when it deviates from the old one by more than this threshold (Fig. 1a). Gaussian distributions are also used to model values of a feature vector in biometric databases [1]. Figure 1b shows the histogram of temperature values in a geographical area observed in a week. The pdf, represented as a histogram, is an arbitrary distribution between 10 and 20°C.

A logical formulation of queries for this kind of uncertainty model has been recently studied in [12,15].



Probabilistic Spatial Queries. Figure 1. Location and sensor uncertainty.

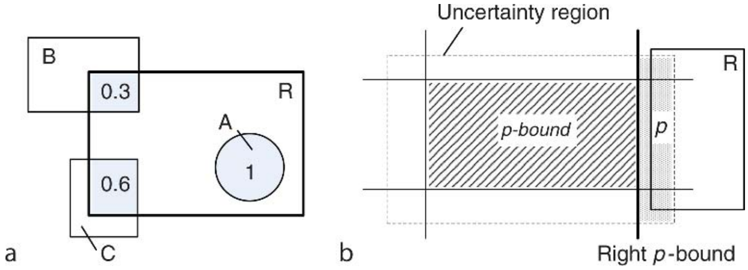
Other variants have also been proposed. In [11], piecewise linear functions are used to approximate the cdf of an uncertain item. Sometimes, point samples are derived from an item's pdf [10,13]. In the *existential uncertainty model*, every object is represented by the value in the space, as well as the probability that this object exists [8]. With these modified models, it is possible to develop fast processing techniques for PSQs.

Query Classification Given the spatial uncertainty model, the semantics of PSQs can be defined. Cheng et al. proposed a classification scheme for different types of PSQ [4]. In that scheme, a PSQ is classified according to the forms of answers. An *entity-based query* is one that returns a set of objects (e.g., list of objects that satisfy a range query or join conditions), whereas a *value-based query* returns a single numeric value (e.g., value of a particular sensor). Another criterion is based on whether an *aggregate* operator is used to produce results. An aggregate query is one where there is interplay between objects that determines the results (e.g., a nearest-neighbor query). Based on these two criteria, four different types of probabilistic queries are defined. Each query type has its own methods for computing answer probabilities. In [4], the notion of *quality* has also been defined for each query type, which provides a metric for measuring the ambiguity of an answer to the PSQ.

In the rest of this section, two important PSQs, namely the probabilistic range queries and the probabilistic nearest-neighbor queries, will be studied.

Probabilistic Range Queries A well-studied PSQ is the *probabilistic range query* (PRQ). Figure 2a illustrates this query, which shows the shape of the uncertainty regions, as well as the user-specified range, *R*. The task of the range query is to return the each object that can be inside *R*, as well as its probability. The PRQ can be used in location-based services, where queries like: "return the suspect vehicles in a crime scene" can be asked. It is also used in sensor network monitoring, where sensor IDs whose physical values (e.g., temperature, humidity) are returned to the user. Figure 2a shows the probability values of the items (*A*, *B*, and *C*) that are located inside *R*. A PRQ is an *entity-based* query, since it returns a list of objects. It is also a *non-aggregate* query, because the probability of each object is independent of the existence of other objects [4].

To compute an item's probability for satisfying the PRQ, one can first find out the overlapping area of each item's region within *R* (shaded in Fig. 2a), and perform an integration of the item's pdf inside the overlapping area. Unfortunately, this solution may not be very efficient, since expensive numerical integration may need to be performed if the the item's pdf is arbitrary [7]. Even if an R-tree is used to prune items that do not overlap *R*, the probability of each item that are non-zero still needs to computed. A more efficient solution was developed in [7], where the authors proposed a user-defined constraint, called the *probability threshold P*, with $P \in (0,1]$. An item is only returned if its probability of satisfying the PRQ is not less than *P*. In Fig. 2a, if $P = 0.6$, then only *A* and *C* will be returned to the user. Under this new requirement, it is possible to incorporate the uncertainty information of items into a spatial index (such as R-tree). The main idea is to precompute the *p-bounds* of an item. A *p*-bound of an uncertain item is essentially a function of *p*, where $p \in [0,0.5]$. In a 2D space, it is composed of four line segments, as illustrated by the hatched region in Fig. 2b. The requirement of right *p*-bound (illustrated by the thick solid line) is that the probability of the location of the item on the right of the line has to be exactly equal to *p* (the shaded area). Similarly, the probability of the item on the left of the left *p*-bound



Probabilistic Spatial Queries. Figure 2. Probabilistic range queries over uncertain items, showing (a) the probability of each item, and (b) the *p*-bound of an uncertain item.

is exactly equal to p. The remaining line segments (top and bottom p-bounds) are defined analogously. Once these p-bounds are known, it is possible to know immediately whether an item satisfies the PRQ. Figure 2b shows that a range query R overlaps the item's uncertainty region, but does not cut the right p-bound, where p is less than P. Since the integration of the item's pdf inside the overlapping area of R and the uncertainty region cannot be larger than P, the item is pruned without doing the actual probability computation.

By precomputing a finite number of p-bounds, it is possible to store them in a modified version of the R-tree. Called *Probability Threshold Index* (PTI), this index can facilitate the pruning of uncertain items in the index level [7]. Compared with the R-tree which uses the MBR of the object for pruning, the use of p-bounds in the PTI provides more pruning power. In Fig. 2b, for example, although the range R overlaps with the MBR of the object (dashed rectangle), it does not cut the p-bounds, and so it can be pruned by the PTI but not by the R-tree. [7] also examined special cases of pdf (uniform and Gaussian distributions) and proposed an indexing scheme where p-bounds can be computed on-the-fly without being stored in the index. Since storing p-bounds for high-dimensional uncertain items can be expensive, Tao et al. [17,18] proposed a variant of PTI called *U-tree*, which only stores approximate information of p-bounds in the index. With these improvements, it is possible to index uncertain items in the high-dimensional space. The p-bound techniques were also used in [6] to facilitate the processing of join queries over uncertain spatial data.

Another PRQ evaluation technique was recently proposed by Ljosa et al. [7], who used piecewise linear functions to approximate the cdf of an uncertain item in order to avoid expensive integration. They also described an index that stored these piecewise linear functions, so that a PRQ can be evaluated more efficiently. More recently, the problem of evaluating *imprecise location dependent range queries* is studied [18,2]. This is a variant of PRQ, where the range query is defined with reference to the (imprecise) position of the query issuer. For instance, if the query issuer looks for his friends within 2 miles of his current position, and his position is uncertain, then the actual query range (a circle with a 2-mile radius) cannot be known precisely. The authors of [2] proposed several approaches to efficiently evaluate these queries, by (i) using the Minkowski Sum (a computational geometry technique), (ii) switching the role of query issuer and data being queried, and (iii) using p-bounds.

Nearest-Neighbor Queries Another important PSQ for uncertain items is the probabilistic nearest-neighbor queries (PNNQ in short). This query returns the non-zero probability of each object for being the nearest neighbor of a given point q [4]. A PNNQ can be used in a sensor network, where sensors collect the temperature values in a natural habitat. For data analysis and clustering purposes, a PNNQ can find out the district(s) whose temperature values is (are) the closest to a given centroid. Another example is to find the IDs of sensor(s) that yield the minimum or maximum wind-speed from a given set of sensors [9,4]. A minimum (maximum) query is essentially a special case of PNNQ, since it can be characterized as a PNNQ by setting q to a value of $-\infty$ (∞).

Evaluating a PNNQ is not trivial. In particular, since the exact value of a data item is not known, one needs to consider the item's possible values in its uncertainty region. Moreover, since the PNNQ is an entity-based aggregate query [4], an item's probability depends not just on its own value, but also on the relative values of other objects. If the uncertainty regions of the objects overlap, then their pdfs must be considered in order to derive their corresponding probabilities. This is unlike the evaluation of PRQ, where each item's probability can be computed independent of others. To evaluate PNNQ, one method is to derive the pdf and cdf of each item's distance from q. The probability of an item for satisfying the PNNQ is then computed by integrating over a function of distance pdfs and cdfs [9,4,5]. In [5], an R-tree-based solution for PNNQ was presented. The main idea is to prune items with zero probabilities, using the fact that these items' uncertainty regions must not overlap with that of an item whose maximum distance from q is the minimum in the database. The *probabilistic verifiers*, recently proposed in [3], are algorithms for efficiently computing the lower and upper bounds of each object's probability for satisfying a PNNQ. These algorithms, when used together with the probability threshold defined by the user, avoid the exact probability values to be calculated. In this way, a PNNQ can be evaluated more efficiently.

There are two other solutions for PNNQ that base on a different representation of uncertain items. Kriegel et al. [10] used the Monte-Carlo method, where the pdf of each object was sampled as a set of points. The probability was evaluated by considering the portion of points that could be the nearest neighbor. In [11], Ljosa et al. used piecewise linear representation of the cdf for an uncertain item to propose efficient evaluation and indexing techniques.

Another important entity-based aggregate query over uncertain items, namely the probabilistic skyline queries, has been studied in [13]. A skyline query returns a set of items that are not dominated by other items in all dimensions. In that paper, the issues of defining and computing the probability that an uncertain item was in the skyline were addressed. Two bounding-pruning-refining based algorithms were developed: the bottom-up algorithm used selected instances of uncertain items to prune other instances of uncertain items, while the top-down algorithm recursively partitions the instances of uncertain items into subsets. The authors showed that both techniques enable probabilistic skyline queries to be efficiently computed.

Key Applications

A PSQ can be used in applications that require the processing of uncertain spatial data. These applications include location-based services, road traffic monitoring, wireless sensor network applications, and biometric feature matching, where the data collected from the physical environments (e.g., location, temperature, humidity, images) cannot be obtained with a full accuracy. Recent works that propose to inject uncertainty to a user's location for location privacy protection also requires the use of a PSQ [2].

Future Directions

A lot of work remains to be done in the area of uncertain spatial data processing. An important future work will be the definition and evaluation of important spatial queries, such as reverse nearest-neighbor queries. It will also be interesting to study the development of data mining algorithms for spatial uncertainty. Another direction is to study spatio-temporal queries over historical spatial data (e.g., trajectories of moving objects). Other works include revisiting query cost estimation, query plan evaluation, and user interface design that allows users to visualize uncertain data. A long term goal is to consolidate these research ideas and develop a comprehensive spatio-temporal database system with uncertainty management facilities.

URL to Code

The following URL contains source codes of the ORION system, which is a database system that provides querying facilities for uncertain spatial data: http://orion.cs.purdue.edu

Cross-references

- ▶ Nearest Neighbor Query
- ▶ R-Tree (and Family)
- ▶ Spatial Anonymity
- ▶ Spatial Indexing Techniques

Recommended Reading

1. Böhm C., Pryakhin A., and Schubert M. The Gauss-Tree: Efficient object identification in databases of probabilistic feature vectors. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
2. Chen J. and Cheng R. Efficient evaluation of imprecise location-dependent queries. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
3. Cheng R., Chen J., Mokbel M., and Chow C. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In Proc. 24th Int. Conf. on Data Engineering, 2008.
4. Cheng R., Kalashnikov D., and Prabhakar S. Evaluating probabilistic queries over imprecise data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 551–562.
5. Cheng R., Kalashnikov D.V., and Prabhakar S. Querying imprecise data in moving object environments. IEEE Trans. Knowl. and Data Eng., 16(9), 2004.
6. Cheng R., Singh S., Prabhakar S., Shah R., Vitter J., and Xia Y. Efficient join processing over uncertain data. In Proc. Int. Conf. on Information and Knowledge Management, 2006.
7. Cheng R., Xia Y., Prabhakar S., Shah R., and Vitter J. S. Efficient indexing methods for probabilistic threshold queries over uncertain data. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 876–887.
8. Dai X., Yiu M. L., Mamoulis N., Tao Y., and Vaitis M. Probabilistic spatial queries on existentially uncertain data. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005, pp. 400–417.
9. Deshpande A., Guestrin C., Madden S., Hellerstein J., and Hong W. Model-driven data acquisition in sensor networks. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
10. Kriegel H., Kunath P., and Renz M. Probabilistic nearest-neighbor query on uncertain objects. In Proc. 12th Int. Conf. on Database Systems for Advanced Applications, 2007, pp. 337–348.
11. Ljosa V. and Singh A. APLA: Indexing arbitrary probability distributions. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 946–955.
12. Parker A., Subrahmanian V., and Grant J. A logical formulation of probabilistic spatial databases. IEEE Trans. Knowl. and Data Eng., 19(11), 2007.

13. Pei J., Jiang B., Lin X., and Yuan Y. Probabilistic skylines on uncertain data. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
14. Pfoser D. and Jensen C. Capturing the uncertainty of moving-objects representations. In Proc. 11th Int. Conf. on Scientific and Statistical Database Management, 1999.
15. Singh S., Mayfield C., Shah R., Prabhakar S., Hambrusch S., Neville J., and Cheng R. Database support for probabilistic attributes and tuples. In Proc. 24th Int. Conf. on Data Engineering, 2008.
16. Sistla P.A., Wolfson O., Chamberlain S., and Dao S. Querying the uncertain position of moving objects. In Temporal Databases: Research and Practice. Springer Verlag, 1998.
17. Tao Y., Cheng R., Xiao X., Ngai W. K., Kao B., and Prabhakar S. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 922–933.
18. Tao Y., Xiao X., and Cheng R. Range Search on Multidimensional Uncertain Data. ACM Trans. Database Syst. 32(3), 2007.

Probabilistic Temporal Databases

V. S. Subrahmanian
University of Maryland, College Park, MD, USA

Synonyms

Temporally uncertain databases; Temporally indeterminate databases

Definition

There are many applications where the fact that a given event occurred is known, but where there is uncertainty about exactly when that event occurred. Such events are called temporally indeterminate events. Probabilistic temporal databases attempt to store information about events that are both temporally determinate and temporally indeterminate. For the latter, they specify a set of time points (often an interval) – it is known that the event occurred at some time point in this set. The probability that the event occurred at a specific time point is given by a probability distribution or by one of a set of probability distributions.

Historical Background

There is no shortage of events that are known to have certainly occurred, but where the exact dates are not known with certainty. For instance, the exact date of the extinction of dinosaurs is unknown – nor does the historical record show the precise date when Cyrus the Great of Persia was born. The latter, estimated by scholars to be between 590–576 BC, is an excellent example of a temporally indeterminate event.

Techniques to study such temporally indeterminate events in databases started with information about *partial null values* [9,13]. Meanwhile, in a largely separate community, researchers focused on the problem of incorporating probabilistic information within a relational database. One of the earliest efforts was due to Cavallo and Pittarelli [4] who proposed an extension of the relational data model to include probabilities – they proposed a partial algebra consisting of projection and join operators. Much early work in this field, such as that of [2,7] made important advances under some restrictions Lakshmanan et al.'s ProbView system [11] proposed two representations of probabilistic data: a *probabilistic tuple* has the form $((t_1,V_1),\ldots,(t_n,V_n))$ where each t_i is a set of possible values and V_i is a probability distribution over the set. They then showed that each probabilistic tuple could be "flattened" into an "annotated" representation. Annotated tuples look like ordinary tuples except that a probability is attached to the tuple as a whole (and a special "path" field was introduced to handle tuples with identical data). ProbView introduced the important concept of conjunction and disjunction strategies that allowed users to specify – in their query – their knowledge about the dependencies between events when posing the query. Thus, the barrier of the independence assumption in previous works was overcome. They proposed a query algebra for annotated tuples, developed query rewrite rules, and developed view maintenance algorithms.

Much work has subsequently been done on temporal probabilistic data. Despite a long history of reasoning about time and uncertainty in AI [10], a major advance in temporal probabilistic databases occurred when Dyreson and Snodgrass [8] proposed the notion of an indeterminate instant. This is just like one of the (t_i,V_i) pairs in the annotated representation of [11] with the exception that the t_i component represents a set of time points. They then extended the SQL query language in several ways. First, they added constructs to indicate that a temporal attribute in indeterminate. Second, they added the concept of "correlation credibility" which allows a query to modify indeterminate temporal data (e.g., by choosing a max temporal value or an expected temporal value). Third, they introduced the concept of "ordering plausibility" which specifies an ordering about the plausibility levels of different conditions in the WHERE clause of an SQL query. The next major breakthrough in temporal probabilistic databases came when Dekhtyar et al. [6] proposed a temporal probabilistic relational database model that added “tp-cases” to ordinary relational tuples. A tp-case contained two constraints one of which was used to denote valid time points as solutions of the constraints, and the other was used in conjunction with a distribution function to infer a probability distribution on the solutions of the first constraint. They developed an extension of the relational algebra that directly manipulated tp-tuples and used the conjunction and disjunction strategies of ProbView [11] to avoid making independence assumptions. TP-databases were one of the first temporal probabilistic DBMSs to report real world applications for the US Navy [12]. It was later used to build similar applications for the US Army as well. Later, Biazzo et al. [3] extended this work to the case of object bases containing temporal indeterminacy. The Trio system [1] extends temporal indeterminacy models to include lineage information as well. More recently, SPOT databases [14] allow reasoning in the presence of space, time and uncertainty.

Foundations

Consider a relation $R(A_1,\ldots,A_n)$ which describes events whose temporal validity is not precisely known. In order to identify when the tuples in such a relation are valid, *constraints* can be to describe the period of validity of the tuple. The table below (ignoring the shaded column for now) is a temporally indeterminate database about vehicle locations.

The first row in this table says that the event “Vehicle V1 was at location *a*” occurred at some time during the closed interval [11,20]. This is a form of temporal indeterminacy with no probabilities. Suppose there is a probability distribution over the last column. A common temptation is to add an additional column specifying the “name” of the distribution (e.g., “u” might be the uniform distribution, g_r may be a geometric distribution with a parameter r, and so forth). Such a table might now look like this:

VehicleID	Vehicle Type	Location	Time	PDF
V1	T72	*a*	11 < = t < = 20	U
V1	T72	*b*	18 < = t < = 25	$\gamma_{0.5}$
V1	T72	*c*	24 < = t < = 27	$\gamma_{0.2}$

Now consider the table above with the shaded column included. The first row in the table now says that there is a 10% probability that vehicle V1 will be at location *a* at time 11 (and the same for times 12, 13, and so on till time 20) and that the probability is uniformly distributed (probability distribution function or pdf “u”). In contrast, the second row says something different because the pdf γ treats time intervals differently. It says, implicitly, that the probability that vehicle V1 will be at location *b* at time 18 is 0.15, at time 19 is 0.25, at time 20 is 0.125, and so on.

How should queries over this representation of the database be answered? To see this, consider a temporal query which says “Select all tuples where $20 < Time < 23$.” The only tuple that has any chance of satisfying this constraint is the second one. However, there is no *guarantee* that the second tuple actually is valid during this interval. Fortunately, there is a 9.375% chance that this is the case – so the system could return the probabilistically valid response saying that the second tuple is valid with probability 9.375%. Unfortunately, there is no way of returning this answer to the user *unless the implicit assumption in all relational databases that the output schema for selection queries should match the input schema is sacrificed.* It is clearly inappropriate to just return the table:

VehicleID	Vehicle Type	Location	Time	PDF
V1	T72	*b*	20 <t<23	$\gamma_{0.5}$

The *Time* field here is computed merely by solving the constraint in the second tuple of the input relation in conjunction with the constraint in the query. Unfortunately, the distribution in the *PDF* field is incorrect, and would need to be recomputed. This is further complicated by the fact that the shape of the original geometric distribution does not look the same when restricted to a portion (of interest) of the original distribution.

There have been two attempts to solve the problem of dealing with what happens when a distribution is manipulated. Dyreson and Snodgrass [8] develop a “rod and point” method to store distributions and infer new distributions when selection operations of the kind above are performed. They approximate a probability mass by splitting it into chunks called “rods.” However, each chunk can have a different length. An approximate representation of the original distribution is obtained through this rod mechanism.

Dekhtyar et al. [6] solve this problem by using some extra space. They require that the "base relation" contains *two constraints*, both of which are identical initially. They would represent the original relation as follows:

VehicleID	Vehicle Type	Location	Time	Time 2	PDF
V1	T72	*a*	11 < = t < = 20	11 < = t < = 20	U
V1	T72	*b*	18 < = t < = 25	18 < = t < = 25	$G_{0.5}$
V1	T72	*c*	24 < = t < = 27	24 < = t < = 27	$G_{0.2}$

In base relations, the *Time* and *Time2* fields have exactly the same constraints in them. When queries are executed, the *Time* field ends up denoting valid time and changes based on the query – however, the *Time2* field rarely changes. When the selection query mentioned above is execution, they would return the answer:

VehicleID	Vehicle Type	Location	Time	Time 2	PDF
V1	T72	*B*	20 < t < 23	18 < = t < = 25	$G_{0.5}$

This is very subtle. The *Time* attribute here specifies the valid time (in this case, time points 21 and 22). The *Time2* attribute is a system-maintained attribute that need not be shown to the user which says apply the PDF mentioned in the *PDF* field to the solutions of the *Time2* constraint – but only show the probability values for the solutions of the *Time* constraint). Thus, *Time2* is used to derive the probabilities for each valid time point. In the above case, the valid time points are 21 and 22, and their probabilities are derived – using the distribution in the *PDF* column applied to the constraint in the *Time2* column – to get probabilities of 0.0625 and 0.03125, respectively. Dekhtyar et al. [6] goes on and specifies how to add additional "low" and "high" probability fields to such relations and provides normalization methods.

Cartesian products between two relations are more complex. When concatenating tuple *st1* and *t2* from two different relations, it is important to consider the probability that both tuples will be valid at a given time point. This requires knowing the relationship between the events being denoted by these two tuples: are they independent? Are they correlated somehow? Is there no information about the relationship? Thus, a *conjunction strategy* must be specified in the query when a Cartesian product (and hence a join operation) is being performed. Dekhtyar et al. [6] show how Cartesian products and joins can be computed under any assumption specified in a user query.

Another recent effort is the one on Trio [1] which attempts to deal with time, uncertainty, and lineage. They address the fact that in many applications involving uncertainty (such as crime-fighting applications), any "final" answer needs to be explainable. As a consequence, they introduce "lineage" parameters when answering a query – informally speaking, the lineage parameter associated with a tuple in an answer (similar to the "path" parameter in [11]) associates a "justification" for each tuple. This justification references the set of base tuples that caused the derived tuple to be placed in an answer.

Key Applications

Temporal probabilistic databases have already been used in defense applications. For instance, [12] describes work in which temporal probabilistic databases are used to store the results of predictions about where enemy submarines will be in the future, when they will be there, and with what probability. In fact, this raises a large set of possible applications based on reasoning about moving objects. For defense applications, cell phone applications, logistics applications, and many other application domains, there is interest in knowing where a moving object will be in the future, when it is expected to be there, and with what probability. Cell phone companies can use such data to understand and better handle load on cell towers.

Moreover, as the world is becoming increasing "geo-location aware" through the use of devices like RFID tags and GPS locators, it is clear that reasoning about where vehicles will be in the future and with what probability will be important in a wide range of applications such as traffic light settings to ease road congestion, recommending detours on highway signs, and more effectively directing 911 traffic in congested situations. These would not be possible without a good estimate of when and where and with what probabilities vehicles will be in the future.

Logistics applications are another important class of applications where companies need to plan activities in the presence of uncertainty about when various supply items will arrive. Corporations today use complex prediction models to learn about suppliers' performance.

Financial applications are another major source of temporal uncertainty. Banks need to have a good idea of their incoming funds. For instance, a credit card provider deals with constant uncertainty about when people will pay their credit card bills, how much of the bills they will pay, and how much they will carry forward as debt. Such applications embody a mix of data uncertainty and temporal uncertainty.

Future Directions

Three major areas of expansion include:

1. *Temporal probabilistic aggregates*. To date, there are almost no techniques to manage aggregates efficiently in temporal probabilistic databases. Most methods would compute aggregates by first answering a non-aggregate query and then deriving aggregates from there: however, techniques to scalably answer aggregate queries are required.
2. *Probabilistic spatio-temporal reasoning*. Moving objects clearly have a spatial component – hence, reasoning about them involves a neat fusion of temporal reasoning, spatial reasoning, and probabilistic reasoning. Some work on indexing in such domains has recently been proposed. However, much future work is needed, especially in understanding the correlations that exist between the presence of a vehicle at time *t* and its presence at another location at time *t* + *1*.
3. *Query optimization*. Recent work [5] has made a good start on query optimization in probabilistic databases – however, query optimization in databases involving time and uncertainty has a ways to go. Such methods are critical for scaling applications.

Cross-references

▶ Qualitative Temporal Reasoning
▶ Temporal Constraints

Recommended Reading

1. Agrawal P., Benjelloun O., Sarma A.D., Hayworth C., Nabar S.U., Sugihara T., and Widom J. Trio: a system for data, uncertainty, and Lineage. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 1151–1154.
2. Barbará D., Garcia-Molina H., and Porter D. The management of probabilistic data. IEEE Trans. Knowl. Data Eng., 4(5): 487–502, 1992.
3. Biazzo V., Giugno R., Lukasiewicz T., and Subrahmanian V.S. Temporal probabilistic object bases. IEEE Trans. Knowl. Data Eng., 15(4):921–939, 2003.
4. Cavallo R. and Pittarelli M. The theory of probabilistic databases. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 71–81.
5. Dalvi N. and Suciu D. Answering queries from statistics and probabilistic views. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 805–816.
6. Dekhtyar A., Ross R., and Subrahmanian V.S. Probabilistic temporal databases, I: algebra. ACM Trans. Database Syst., 26(1): 41–95, 2001.
7. Dey D. and Sarkar S. A probabilistic relational model and algebra. ACM Trans. Database Syst., 21(3):339–369, 1996.
8. Dyreson C.E. and Snodgrass R.T. Supporting valid-time indeterminacy. ACM Trans. Database Syst., 23(1):1–57, 1998.
9. Grant J. Partial values in a tabular database model. Inf. Process. Lett., 9(2):97–99, 1979.
10. Kraus S. and Subrahmanian V.S. Multiagent reasoning with probability, time and beliefs. Int. J. Intell. Syst., 10(5):459–499, 1994.
11. Lakshmanan L.V.S., Leone N., Ross R.B., and Subrahmanian V.S. ProbView: a flexible probabilistic database system. ACM Trans. Database Syst., 22(3):419–469, 1997.
12. Mittu R. and Ross R. Building upon the coalitions agent experiment (COAX) – integration of multimedia information in GCCS-M using IMPACT. In Proc. Ninth Int. Workshop on Multimedia Information Systems, 2003, pp. 35–44.
13. Ola A. Relational databases with exclusive disjunctions. In Proc. 8th Int. Conf. on Data Engineering, 1992, pp. 328–336.
14. Parker A., Subrahmanian V.S., and Grant J. A logical formulation of probabilistic spatial databases. IEEE Trans. Knowl. Data Eng., 19(11):1541–1556.

Probability Ranking Principle

Ben He
University of Glasgow, Glasgow, UK

Synonyms

PRP

Definition

The probability ranking principle asserts that relevance has a probabilistic interpretation. According to this principle documents are ranked by a probability $p(Rel|d, q)$, where *Rel* denotes the event of a document *d* being relevant to a query *q*. Robertson called this principle the *probability ranking principle* [1].

Key Points

By assuming independence between query terms, Robertson and Sparck-Jones proposed for the probability $p(Rel|d,q)$ the following model (the RSJ model [2]):

$$\log(p(Rel|d,q)) \propto \sum_{t \in q} \log \frac{p(t|Rel) \cdot p(\overline{t}|\overline{Rel})}{p(t|\overline{Rel}) \cdot p(\overline{t}|Rel)} \qquad (1)$$

where $\overline{Rel}$ indicates the event of non-relevance; t and $\overline{t}$ indicate the events that the term t occurs in document d or does not, respectively. For each query term t, the probability $p(Rel|d,t)$ is given by the sum of two log-odds, $\log \frac{p(t|Rel)}{p(t|\overline{Rel})}$ and $\log \frac{p(\overline{t}|\overline{Rel})}{p(\overline{t}|Rel)}$.

If N is the number of documents in the whole collection, R is the number of relevant documents, r is the number of relevant documents containing t, N_t is the document frequency, i.e., the number of documents containing t, [3] instantiated the RSJ model as follows:

$$w^{(1)} = \log \frac{(r+0.5)(N-N_t-R+r+0.5)}{(R-r+0.5)(N_t-r+0.5)} \qquad (2)$$

where $w^{(1)}$ is the raw weight of a term t in a document d. The number 0.5 is used to avoid assigning negative weights. The formula is called the "point-5" formula.

If relevance information is not available, i.e., $R = r = 0$, the point-5 formula can be written as:

$$w^{(1)} = \log \frac{N - N_t + 0.5}{N_t + 0.5} \qquad (3)$$

As one of the most well-established IR systems, Okapi uses a weighting model that is based on the RSJ model introduced above, and takes also term frequency (tf) and query term frequency (qtf) into consideration.

Cross-references

► Information Retrieval
► Information Retrieval Models
► Term weighting

Recommended Reading

1. Robertson S.E. The probability ranking principle in IR. J. Doc., 33:294–304, 1977.
2. Robertson S.E. and Sparck-Jones K. Relevance weighting of search terms. J. Am. Soc. Inf. Sci., 27:129–146, 1977.
3. Robertson S.E. and Walker S. On relevance weights with little relevance information. In Proc. 20th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1997, pp. 16–24.

Probability Smoothing

Djoerd Hiemstra
University of Twente, AE Enschede, The Netherlands

Definition

Probability smoothing is a language modeling technique that assigns some non-zero probability to events that were unseen in the training data. This has the effect that the probability mass is divided over more events, hence the probability distribution becomes more *smooth*.

Key Points

Smoothing overcomes the so-called *sparse data problem*, that is, many events that are plausible in reality are not found in the data used to estimate probabilities. When using maximum likelihood estimates, unseen events are assigned zero probability. In case of information retrieval, most events are unseen in the data, even if simple unigram language models are used documents are relatively short (say on average several hundreds of words), whereas the vocabulary is typically big (maybe millions of words), so the vast majority of words does not occur in the document. A small document about "information retrieval" might not mention the word "search," but that does not mean it is not relevant to the query "text search." The sparse data problem is the reason that it is hard for information retrieval systems to obtain high recall values without degrading values for precision, and smoothing is a means to increase recall (possibly degrading precision in the process). Many approaches to smoothing are proposed in the field of automatic speech recognition [1]. A smoothing method may be as simple so-called Laplace smoothing, which adds an extra count to every possible word. The following equations show respectively (1) the unsmoothed, or maximum likelihood estimate, (2) Laplace smoothing, (3) Linear interpolation smoothing, and (4) Dirichlet smoothing [3]:

$$P_{ML}(T=t|D=d) = tf(t,d) / \sum_{t'} tf(t',d) \qquad (1)$$

$$P_{LP}(T=t|D=d) = (tf(t,d)+1) / \sum_{t'} (tf(t',d)+1) \qquad (2)$$

$$P_{LI}(T=t|D=d) = \lambda P_{ML}(T=t|D=d) + (1-\lambda)P_{ML}(T=t|C) \quad (3)$$

$$P_{Di}(T=t|D=d) = (tf(t,d) + \mu P_{ML}(T=t|C)) / ((\sum_{t'} tf(t',d)) + \mu) \quad (4)$$

Here, $tf(t,d)$ is the frequency of occurrence of the term t in the document d, and $P_{ML}(T|C)$ is the probability of a term occurring in the entire collection C. Both linear interpolation smoothing and Dirichlet smoothing assign a probability proportional to the term occurrence in the collection to unseen terms. Here, λ $(0<\lambda<1)$ and μ $(\mu>0)$ are unknown parameters that should be tuned to optimize retrieval effectiveness. Linear interpolation smoothing has the same effect on all documents, whereas Dirichlet smoothing has a relatively big effect on small documents, but a relatively small effect on bigger documents. Many smoothed estimators used for language models in information retrieval (including Laplace and Dirichlet smoothing) are approximations to the *Bayesian predictive distribution* [2].

Cross-references

- ▶ Language Models
- ▶ N-Gram Models

Recommended Reading

1. Chen S.F. and Goodman J. An empirical study of smoothing techniques for language modeling. Technical report TR-10-98, Center for Research in Computing Technology, Harvard University, August 1998.
2. Zaragoza H., Hiemstra D., Tipping M., and Robertson S. Bayesian extension to the language model for ad hoc information retrieval. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 4–9.
3. Zhai C. and Lafferty J. A study of smoothing methods for language models applied to information retrieval. ACM Trans. Inf. Syst., 22(2):179–214, 2004.

Procedure Order

▶ Clinical Order

Procedure Request

▶ Clinical Order

Process Composition

▶ Composition

Process Definition

▶ Workflow Model

Process Evolution

▶ Workflow Evolution

Process Life Cycle

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

Workflow lifecycle; Thread lifecycle; Process state model

Definition

The stages of life from the start to the end of a process instance within the context of workflow management.

Key Points

The Process Life Cycle represents the stages of a process instance as it evolves from instantiation to termination. This life cycle is most closely related to the life cycle of a thread, and is distinct from the life cycle approach to Business Process Management initiatives, involving an iterative or recursive evolution through the five stages of design, modeling, execution, monitoring, and optimization.

The latter notion of Business Process Management Life cycle is associated with the discipline of continuous process improvement, whereby processes are never deemed "complete" and thus no longer subject to change, but rather are continuously improved through multiple instances of execution, examination, and modification. In contrast, the Process Life Cycle as defined herein refers to the "life span" of a process instance and has definitive start and end points. Thus the Process Life Cycle of the individual process instance is more aptly described as linear as opposed to cyclical, although at various steps in the process it may cycle between running and suspended states.

The steps of the Process Life Cycle are "Instantiate" representing the creation of a new instance (making it live but not necessarily running); "activate" representing the activation of the process instance (now live and running); "passivate" which refers to temporarily suspending the instance (live but not running); "terminate" which represents the end of life of the process instances, through either abortion, cancellation, or completion after running through the full process as defined.

Cross-references

- ▶ Business Process Model
- ▶ Process Definition
- ▶ Workflow Model

Process Management

- ▶ Business Process Management

Process Mining

W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven,
The Netherlands

Synonyms

Workflow mining

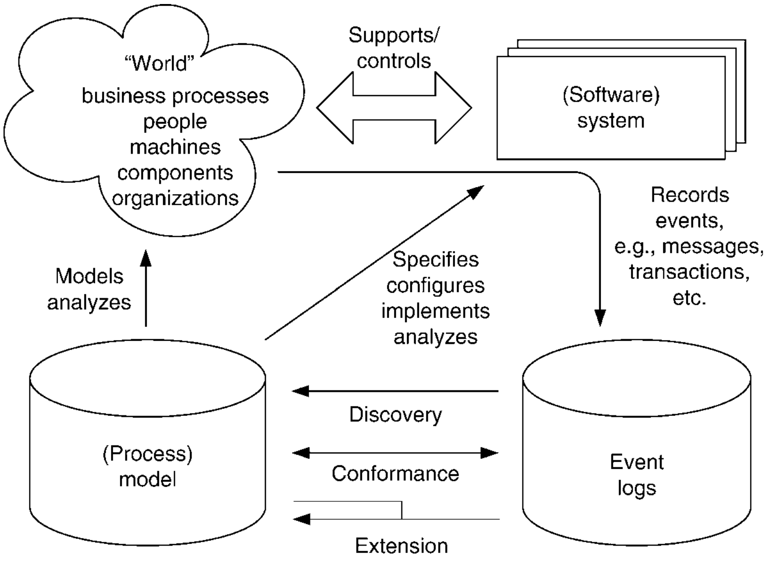
Definition

Process mining techniques allow for the analysis of business processes based on event logs. For example, the audit trails of a workflow management system, the transaction logs of an enterprise resource planning system, and the electronic patient records in a hospital can be used to discover models describing processes, organizations, and products. Moreover, such event logs can also be used to compare event logs with some a priori model to see whether the observed reality conforms to some prescriptive or descriptive model.

The basic idea of *process mining* is to discover, monitor, and improve *real* processes (i.e., not assumed processes) by extracting knowledge from event logs. Today many of the activities occurring in processes are either supported or monitored by information systems. Consider for example ERP, WFM, CRM, SCM, and PDM systems to support a wide variety of business processes while recording well-structured and detailed event logs. However, process mining is not limited to information systems and can also be used to monitor other operational processes or systems. For example, process mining has been applied to complex X-ray machines, high-end copiers, web services, careflows in hospitals, etc. All of these applications have in common that *there is a notion of a process* and that *the occurrences of activities are recorded in so-called event logs*. Assuming that the supporting systems log events, a wide range of *process mining techniques* comes into reach. The basic idea of process mining is to learn from observed executions of a process and can be used to (i) *discover* new models (e.g., constructing a Petri net that is able to reproduce the observed behavior), (ii) check the *conformance* of a model by checking whether the modeled behavior matches the observed behavior, and (iii) *extend* an existing model by projecting information extracted from the logs onto some initial model (e.g., show bottlenecks in a process model by analyzing the event log). All three types of analysis have in common that they assume the existence of some *event log*.

Key Points

The goal of process mining is to discover, monitor, and improve real processes by extracting knowledge from event logs. Clearly, process mining is relevant in a setting where much flexibility is allowed or needed, because the more ways in which people and organizations can deviate, the more variability and the more interesting it is to observe and analyze processes as they are executed. Three basic types of process mining can be identified (Fig. 1):



Process Mining. Figure 1. Three types of process mining: (i) discovery, (ii) conformance, and (ii) extension.

1. *Discovery*. There is no a priori model, i.e., based on an event log some model is constructed. For example, using the α-algorithm [2] a process model can be discovered based on low-level events.
2. *Conformance*. There is an a priori model. This model is used to check if reality conforms to the model. For example, there may be a process model indicating that purchase orders of more than one million Euro require two checks. Another example is the so-called four-eyes principle where two activities need to be executed by different people. Conformance checking may be used to detect deviations, to locate and explain these deviations, and to measure the severity of these deviations.
3. *Extension*. There is an a priori model. This model is extended with a new aspect or perspective, i.e., the goal is not to check conformance but to enrich the model. An example is the extension of a process model with performance data, i.e., some a priori process model dynamically annotated with performance data (e.g., bottlenecks are shown by coloring parts of the process model). Figure 1 shows that a log and a model are used to create a new model.

Traditionally, process mining has been focusing on *discovery*, i.e., deriving information about the original process model, the organizational context, and execution properties from enactment logs. An example of a technique addressing the control flow perspective is the α-algorithm, which constructs a Petri net model describing the behavior observed in the event log. However, process mining is not limited to process models (i.e., control flow) and recent process mining techniques are more and more focusing on other perspectives, e.g., the organizational perspective or the data perspective. For example, there are approaches to extract social networks from event logs and analyze them using social network analysis [1]. This allows organizations to monitor how people and groups are working together.

Conformance checking compares an a priori model with the observed behavior as recorded in the log. In [4] it is shown how a process model (e.g., a Petri net) can be evaluated in the context of a log using metrics such as "fitness" (Is the observed behavior possible according to the model?) and "appropriateness" (Is the model "typical" for the observed behavior?). However, it is also possible to check conformance based on organizational models, predefined business rules, temporal formula's, Quality of Service (QoS) definitions, etc.

There are different ways to *extend* a given process model with additional perspectives based on event logs, e.g., decision mining. Decision mining, also referred to as decision point analysis, aims at the detection of data dependencies that affect the routing of a case. Starting from a process model, one can analyze how data attributes influence the choices made in the process based on past process executions. Classical data mining techniques such as decision trees can be leveraged for this purpose. Similarly, the process model can be extended with timing information (e.g., bottleneck analysis).

Process mining is strongly related to classical data mining approaches. However, the focus is not on data but on process-related information (e.g., the ordering of activities). Process mining is also related to monitoring and business intelligence [3].

Cross-references

- ▶ Association Rule Mining on Streams
- ▶ Data Mining
- ▶ Workflow Management

Recommended Reading

1. van der Aalst W.M.P. Reijers H.A., and Song M. Discovering social networks from event logs. Comput. Support. Coop. Work, 14(6):549–593, 2005.
2. van der Aalst W.M.P. Weijters A.J.M.M., and Maruster L. Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng., 16(9):1128–1142, 2004.
3. Grigori D., Casati F., Castellanos M., Dayal U., Sayal M., and Shan M.C. Business process intelligence. Comput. Ind., 53(3):321–343, 2004.
4. Rozinat A. and van der Aalst W. M. P. Conformance checking of processes based on monitoring real behavior. Inf. Syst., 33(1):64–95, 2007.

Process Optimization

Danilo Ardagna
Politechnico di Milano University, Milan, Italy

Synonyms

Business process optimization; QoS-based web services composition

Definition

With the development of the service oriented architecture (SOA), complex applications can be composed as business processes invoking a variety of available Web services (WSs) with different characteristics. Advanced SOA systems [1,12,15] allow the development of applications by specifying component WSs in a process only through their required functional characteristics, and to select WSs during process execution from the ones included in a registry of available services. Service descriptions are stored and retrieved from enhanced UDDI registries, which also provide information about quality of service (QoS) on the provider side. Usually, a set of functionally equivalent services can be selected, i.e., services which implement the same functionality but differ for the quality parameters.

Process optimization identifies the best set of services available at run time, taking into consideration end-user preferences and constraints on the QoS properties.

Historical Background

Process optimization has its roots in workflow scheduling problems. Scheduling of workflows [13] is the problem of finding a correct execution sequence of workflow tasks such that some temporal constraints or resource constraints (i.e., agents which can support tasks executions) are met. In workflow management systems, agents could be human beings or software applications, in Process optimization the available resources are component WSs. Process optimization is related also to workflow/process planning [3] where the problem of synthesizining a complex behavior from an explicit goal and a set of candidates which contribute to a partial reaching of this goal is investigated. In Process optimization, vice versa, the *process schema*, i.e., the sequence of activities, is given and the optimum mapping of activities to component WSs candidate for their execution is identified.

Process optimization has been applied in context-aware business processes and e-science research fields. The literature has provided *three generations* of solutions. First generation solutions implemented *local* approaches [3,14,15] which select WSs one at the time by associating the running abstract activity to the best candidate service which supports its execution. Local approaches can guarantee only local QoS constraints, i.e., candidate WSs are selected according to a desired characteristic, e.g., the price of *a single WS* is lower than a given threshold.

Second generation solutions proposed *global* approaches [5,8,10,12,15]. The set of services which satisfy the process constraints and user preferences for the whole application are identified before executing the process. In this way, QoS constraints can predicate at a global level, i.e., constraints posing restrictions over the *whole composed service execution* can be introduced. In order to guarantee the fulfilment of global QoS constraints, second generation optimization techniques consider the worst case execution scenario for the composed service. For cyclic processes, loops are unfolded, i.e., unrolled according to their maximum number of iterations [5,15]. This approach could be very conservative and constitutes the main limitation of second generation techniques. Furthermore, global approaches introduce an increased complexity with respect to local solutions. The main issue for the fulfillment of global constraints is WSs performance variability. Indeed, the QoS of a WS may evolve relatively frequently, either because of internal changes or because of workload fluctuations [15]. If a business process has a long duration, the set of services identified by the optimization may change their QoS properties during the process execution or some services can become unavailable or others may emerge. In order to guarantee global constraints, WS selection and execution are interleaved: optimization is performed when the business process is instantiated and its execution is started, and is iterated during the process execution performing *re-optimization* at run time.

To reduce optimization/re-optimization complexity, a number of solution have been proposed which guarantee global constraints only for the critical path [15] (i.e., the path which corresponds to the highest execution time), or reduce loops to a single task [5], satisfying global constraints only statistically, by applying the reduction formula proposed in [7].

Another drawback of second generation solutions is that, if the end-user introduces severe QoS constraints for the composed service execution, i.e., limited resources which set the problem close to un-feasibility conditions (e.g., limited budget or stringent execution time limit), no solutions could be identified and the composed service execution fails [5].

Third generation techniques [3] overcome the limits of the previous approaches and focus on the execution of processes under severe QoS constraints. Severe constraints are very relevant whenever processes have to be performed with stringently limited resources. Third generation solutions are based on loops peeling, which significantly improves the solutions based on loops unfolding. Furthermore, negotiation is exploited if a feasible solution cannot be identified, to bargain QoS parameters with service providers offering services, reducing process invocation failures.

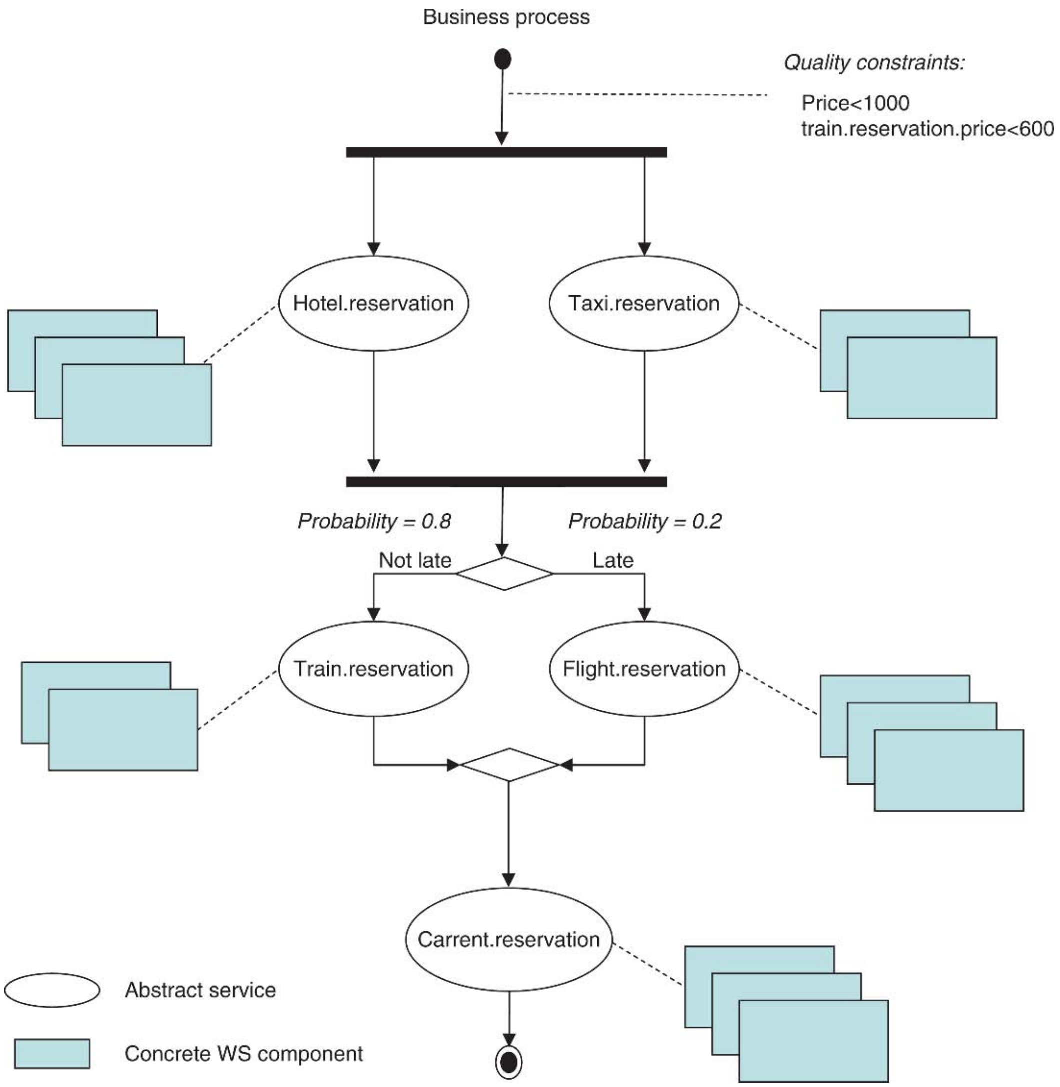
Foundations

Process optimization allows the specification of complex applications as business processes composed by *abstract services* which act as *place holders* of WS components invoked at run time. The best set of services, selected by solving an optimization problem, is invoked at run time by implementing a *dynamic/late binding* mechanism.

Process optimization is usually formalized as a multi-objective optimization problem since several quality criteria can be associated with WS execution. Past approaches [5,12,15] focussed on *execution time* (the expected delay, between the time instant when a request is sent and the time when the result is obtained), *availability* (the probability that the service is accessible), *price* (the fee that a service requester has to pay to the Service Provider for the service invocation), and *reputation* (a measure of the service trustworthiness). Furthermore, the optimization is performed statistically, i.e., by considering the probability of execution of the execution paths of the business process (i.e., any possible sequence of invocations of abstract services). For this reason, some annotations are added to the BPEL specification in order to identify: (i) the maximum [15] or the probability distribution [3] of the number of iterations of loops; (ii) the expected frequency of execution of conditional branches; (iii) global and local constraints on quality dimensions.

Figure 1 shows an example of composed process which implements a virtual travel agency, and the corresponding annotations which specify constraints. The BPEL specification includes invocation to abstract WSs which can be supported at run time by *concrete* WS components.

The probability distribution of the number of iterations of loops and the frequency of execution of conditional branches can be evaluated from past executions by inspecting system logs or can be specified by the composite service designer. If an upper bound for loops execution cannot be determined, then the optimization cannot guarantee that global constraints are satisfied [15]. Prior to perform process optimization, loops are unfolded [15] or peeled [3] (see Fig. 2) and are modeled as directed acyclic graphs (DAGs). Loops peeling is a form of loops unrolling where loop iterations are represented as a sequence of branches and each branch condition evaluates if the loop has to continue with the next *i*th iteration or it has to exit with probability p_i.

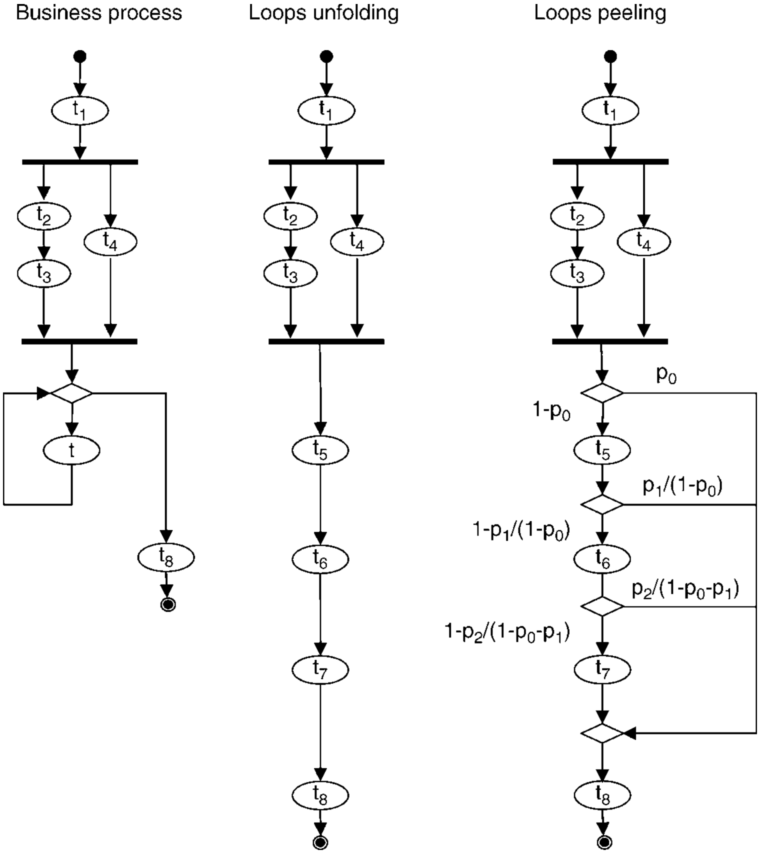


Process Optimization. Figure 1. Virtual travel agency process specification.

The objective function to be optimized is, usually [3,14,15], the aggregated value of QoS for the end user which can be obtained by applying the simple additive weighting (SAW) technique. SAW is one of the most widely used techniques to obtain a *score* from a list of dimensions. Since the quality dimensions have different units of measure, the SAW method first normalizes the raw values for each quality dimension. Each quality dimension is also associated with a weight which expresses the user preferences among multiple quality parameters. The overall value of QoS is calculated as a weighted sum of the normalized values of quality dimension. The SAW method originates a linear objective function, other proposal introduces more general utility functions (i.e., functions which map each possible configuration of the business process to a scalar value) which can be non-linear [5].

First generation solutions considered only *local constraints*. In that case, the process optimization is very simple and the optimum solution can be identified by a greedy algorithm which selects the best candidate service suitable for the execution. An example of first generation technique can be found in [11], where Web agents can migrate to invoke services locally in order to minimize also the network bandwidth.

Second generation solutions support *global constraints* and introuce NP-hard optimization problems. In [4] the complexity of some variants of the global process optimization problem is analyzed, while an overview of heuristic techniques, which hence identify



Process Optimization. Figure 2. Loops unfolding and peeling.

only sub-optimal solutions, can be found in [10]. In [14] global process optimization has been modeled as a multiple choice multiple dimension knapsack problem (MMKP) and as a graph constrained optimum path problem. A MMKP is one kind of knapsack problem where the resources are multidimensional, that is, there are multiple resource constrains for the knapsack (e.g., weight and volume) and items are classied in groups. Each item of the group has a particular value and it requires resources. The objective of the MMKP is to pick exactly one item from each group for maximum total value of the collected items, subject to the resource constraints of the knapsack. Process optimization can be reduced to a MMKP since each abstract service corresponds to a group, each concrete WS is an item in a group and each QoS constraint corresponds dimension of the knapsack. Authors in [14] has implemented ad hoc efficient techniques to identify sub-optimal solutions of the MMKP.

Global approaches have been proposed for the first time in [15], where the Process optimization problem has been formalized as a mixed integer linear programming problem, solved by integer linear programming solvers. The authors separately optimize each execution path and obtain the abstract services to concrete services mapping by composing separate solutions according to the frequency of execution. This approach has some limitations (e.g., availability and response time constraints are guaranteed only for the critical path, and global constraints cannot always be fulfilled) which have been solved by following research proposals [3,14].

Some recent proposals face the Process optimization problem by implementing genetic algorithms [5,8]. In Canfora et al. [5] the reduction formulas presented in [7] are adopted, the re-optimization is considered but abstract services specified in loops are always assigned to the same concrete WS component. Furthermore, by applying reduction formulas the solution guarantees global constraints only statistically. At run time, if low probability paths are taken (see [5]), then the solution could become infeasible and re-optimization must be triggered. In [8], the multi-objective evolutionary approach NSGA-II (non-dominated sorting genetic algorithm) is implemented, which identifies a set of Pareto optimal solutions without introducing a ranking among different quality dimensions. Every identified solution is characterized by the fact that no other plans exist such that a quality dimension is improved without worsening the other ones. Genetic algorithms are more flexible than mixed integer linear approaches, since they allow considering also non-linear composition rules for composed WSs, but are less computationally efficient. In current implementations, some execution time is wasted by generating also non-feasible solutions and, sometimes, no solution can be identified even when the problem is feasible in case the global constraints are stringent.

The work presented in [3] proposes a third generation solution which poses the basis for the execution of processes under severe QoS constraints. The solution is based on loops peeling which significantly improves the solutions based on loops unfolding (up to 40%). Furtheremore, negotiation techniques are exploited to identify a feasible solution of the problem, if one does not exist, reducing process invocation failures. The joint optimization and negotiation approach has been proved to be effective for large processes (including up to 10,000 abstract services), when QoS constraints are severe and reduces also the re-optimization overhead.

Key Applications

Context-Aware Business Process

Dynamic WS selection for composed WSs focused in particular on context aware business processes. Context awareness may be needed both when considering WS personalization, where a generic process is personalized choosing services according to user preferences, and in mobile composed services, to provide ubiquitous services where selection and execution depend on the available services and their QoS [1].

E-Science and Grid Computing

In e-science complex processes, defined as workflows enacted in grid environments, are being developed reaching the dimension of thousands of tasks in "in silico" experiments [9]. Each task is performed selecting and invoking a service. In this case, the Process optimization problem is more challenging, since the concrete resources have to be modeled with a more fine grain and are represented by the physical machines which can support tasks execution instead of abstract WSs. Current solutions propose to create a hierarchy of processes or distributing the workflow over a number of engines, partitioning in this way the optimization problem but leading to sub-optimal solutions.

Future Directions

All of the above approaches consider the optimization of a single process instance and assume a constant QoS profile. Cardellini et al. [6] tackled the problem of optimization of multiple process instances in order to reduce optimization overhead. The work presented in [2] considered variable (periodic) quality of service profiles of component WSs and explicitly addressed long term process execution. The execution of multiple instances, variable QoS profiles, and long term process execution make the optimization problem more cumbersome. Only heuristic approaches have been proposed so far, more efficient solutions, both in term of optimization time and quality of the final solution, are needed.

Experimental Results

For every presented approach, there is an experimental evaluation in the corresponding reference. Zeng et al. [15] discuss local and global approaches. Tao Yu et al. [14] present a comparison among linear integer programming approaches and heuristic solutions. The work presented in [3] analyzes loops peeling and second and third generation solutions.

Cross-references

▶ Composed Services and WS-BPEL
▶ Grid Workflow
▶ Workflow Management and Workflow Managemenet System

Recommended Reading

1. Ardagna D., Comuzzi M., Mussi E., Pernici B., and Plebani P. PAWS: A Framework for Executing Adaptive Web-Service Processes. IEEE Software, 24(6):39–46, 2007.
2. Ardagna D., Giunta G., Ingraffia N., Mirandola R., and Pernici B. QoS-driven Web services selection in autonomic grid environments. In Proc. OTM Confederated Int. Conf. CODPIS, DOA, GADA, and ODBASE, 2006, pp. 1273–1289.
3. Ardagna D. and Pernici B. Adaptive Service Composition in Flexible Processes. IEEE Trans. Software Eng., 2007.
4. Bonatti P.A. and Festa P. On optimal service selection. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 530–538.
5. Canfora G., Penta M., Esposito R., and Villani M.L. QoS-Aware Replanning of Composite Web Services. In Proc. IEEE Int. Conf. on Web Services, 2005.

6. Cardellini V., Casalicchio E., Grassi V., and Mirandola R. A framework for optimal service selection in broker-based architectures with multiple QoS classes. In Proc. IEEE Services Comput. Workshops, 2006, pp. 105–112.
7. Cardoso J. Quality of Service and Semantic Composition of Workflows, Ph. D. Thesis, Univ. of Georgia, 2002.
8. Claro D.B., Albers P., and Hao J.K. Selecting Web Services for Optimal Composition. In Proc. IEEE Int. Conf. on Web Services., 2005.
9. Fox G.C. and Gannon D. Workflow in Grid Systems. Concurrency and Computation: Practice and Experience, 18(10):1009–1019, 2006.
10. Jaeger M.C., Muhl G., and Golze S. QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms. In Proc. Int. Conf. on Cooperative Inf. Syst., 2005.
11. Maamar Z., Sheng Q.Z., and Benatallah B. Interleaving Web Services Composition and Execution Using Software Agents and Delegation. In Proc. Web Services and Agent-Based Eng., 2003.
12. Patil A.A., Oundhakar S.A., Sheth A.P., and Verma K. METEOR-S web service annotation framework. In Proc. 12th Int. World Wide Web Conference, 2004, pp. 553–562.
13. Senkul P. and Toroslu I.H. An architecture for workflow scheduling under resource allocation constraints. Inf. Syst., 30(5):399–422, 2005.
14. Yu T., Zhang Y., and Lin K.J. Efficient algorithms for Web services selection with end-to-end QoS constraints. ACM Trans. Web, 1(1):1–26, 2007.
15. Zeng L., Benatallah B., Dumas M., Kalagnamam J., and Chang H. QoS-Aware Middleware for Web Services Composition. IEEE Trans. Software Eng., 30(5), 2004.

Process Semantics

► Workflow Constructs

Process State Model

► Process Life Cycle

Process Structure of a DBMS

Pat Helland
Microsoft Corporation, Redmond, WA, USA

Synonyms

Cluster databases; Scale-out databases; Scale-up databases; Shared-disk databases; Shared-nothing databases; Shared-everything databases

Definition

Database Management Systems are typically implemented on top of operating systems which allow execution within processes. Different systems have chosen different process structures as they map their computation onto the operating system. This section surveys some of these choices.

Historical Background

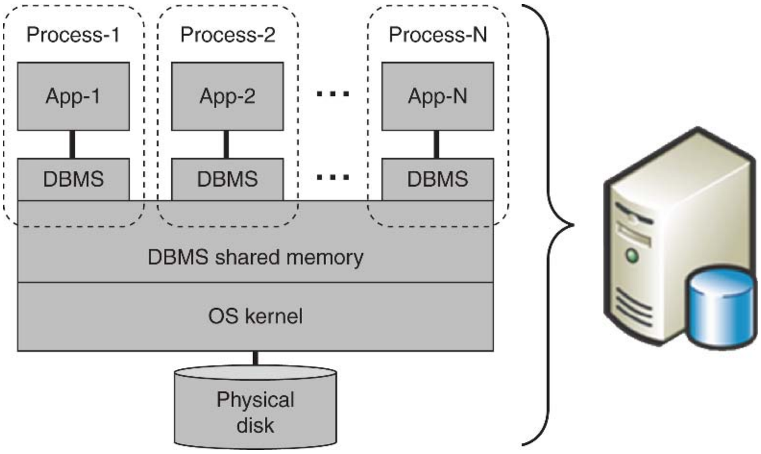
The first database management systems were simple libraries that ran inside the process of the application. While the use of these libraries offered leverage to the applications by providing essential functionality, they did not offer protection for the data in the presence of application errors.

To provide protection, DBMSs were initially moved into higher security rings accessible by hardware protected transitions to memory and code which was more secure than the application but less secure than the operating system kernel. Running the DBMS in shared (but secured) memory allowed access by multiple applications (in separate processes). The shared memory within the DBMS allowed for efficient cross application management of data (See Fig. 1).

Two trends caused gradual retreat from the implementation of the DBMS within a trusted security ring. First, there is the emergence of DBMSs that were designed to be ported across operating systems. Second, the emergence of distributed computing drove the need to run different portions of the computing stack (both application and database) across different machines. Both of these trends grew throughout the 1980s and led to a constellation of process and processor architectures for both applications and database management systems.

The process structure of database management systems has evolved and today can be seen in many forms. As mentioned above, initial implementations of DBMS systems in the 1960s and 1970s were embedded in the same process (but soon with protection for the DBMS within a security ring).

In the 1980s, a number of distributed databases emerged, exemplified by Tandem's NonStop SQL. In these, the application interface remained an in-memory call to portions of the database system but behind the scenes there were cross-process and cross-processor calls to other portions of the DBMS. This was implemented transparently to the application except, of course, with some performance implications which could be both positive and negative.



Process Structure of a DBMS. Figure 1. Early implementations of DBMS systems used shared memory as a technique to allow the DBMS to run in process with the application. This minimized app to DBMS communication costs.

In the 1990s, the client-server computing first arrived with the separation of the client and server-side database in what became known as *two-tier client-server* applications. This necessitated the creation of the *database connection*, exemplified by ODBC (Open Database Connectivity). With a database connection, both DML (Data Manipulation Language) and the resultant data sets were returned across process boundaries, allowing the application process (and processor) to be different than the database process and processor. Subsequently, the application itself began to break across processes and processors resulting in *three-tier client-server* or even later *N-tier client-server systems*. Within each of these architectures, the application program still perceived the notion of a single database even though the work was potentially spread across multiple systems.

Shortly after the year 2000, the industry began to recognize the importance of the relationships of applications and databases running independently and speak about what is today called *SOA (Service Oriented Architecture)*. This is delineated from N-tier client-server by the absence of a common DBMS; SOA services each have the own DBMS whereas a client-server system shares a common DBMS, even if the common DBMS is distributed across processes and/or processors.

Foundations

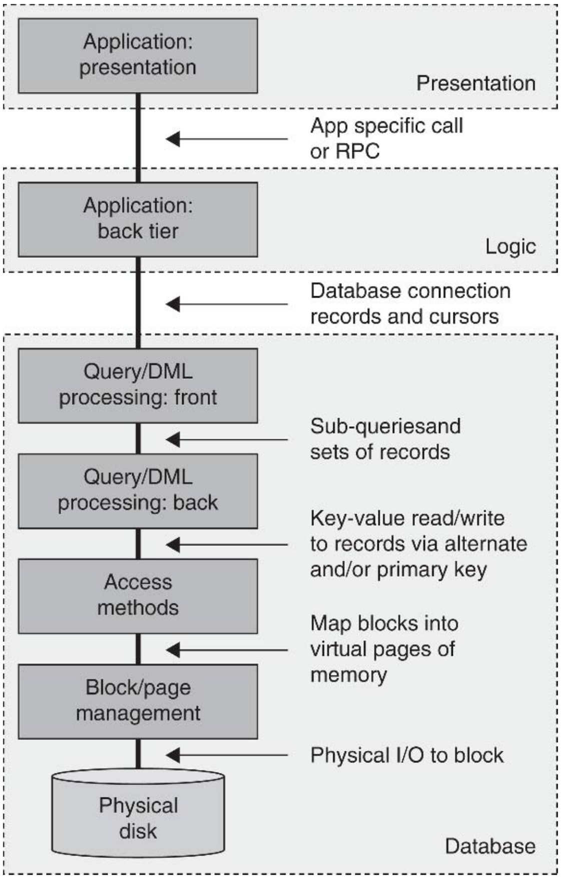
To understand the process and processor architecture of DBMS and applications, the reader first looks at a sketch of their high-level architecture independent of the processes and processors implementing the components of the architecture, followed by some common patterns for breaking this work up.

Layers of the DBMS and Application

Figure 2, depicts a breakdown of both the 3-tier application and the underlying DBMS. The dotted lines show the classic 3-tier architecture. The *presentation tier* is connected to the *logic tier* with an application specific call or RPC. The *logic tier* is connected with the *database tier* using a database connection.

Inside the database, more layers of abstraction are utilized to examine different process and processor architectures. Closest to the application is the front-end to the *Query and DML Processor*: it accepts a database connection from the application and is responsible for the processing DML and queries and returning the results across the connection. The front-end of this function will interact with a back-end which is intimate with the various access methods which hold the database records and/or alternate keys. The back-end of the Query and DML Processor will interact with the *Access Methods* using keys and records. The Access Methods, in turn, use blocks (which are mapped into in-memory pages) by the *Block and Page Manager*, which issues physical I/Os to the disks themselves.

In the simplest DBMS architectures, the *Shared-Everything* design, all DBMS components are resident on the same computer system and they interact through memory. In the *Shared Disk* architecture, the interactions between the *Block and Page Management*



Process Structure of a DBMS. Figure 2. The architecture of a 3-Tier application and its supporting database management system. The notations beside the arrows denote the formats of the requests and data flowing between the layers.

and the *Physical Disk* are spread apart. Finally, the *Shared Nothing* architecture separates the front and back ends of the Query and DML Processor.

Client-Server Computing

When client-server computing first arrived on the scene, its hallmark was the separation of the application from the DBMS itself leaving the database on its own system. Initially, this was done with two tiers, the client and DBMS-server. The client interacted with the server using a database connection such as ODBC (See Fig. 3).

The transition between two-tier client-server systems and three-tier (or even N-tier) client-server systems lies in the architecture of the application itself. It is simply the splitting up of the application tiers that differentiate these (See Fig. 4).

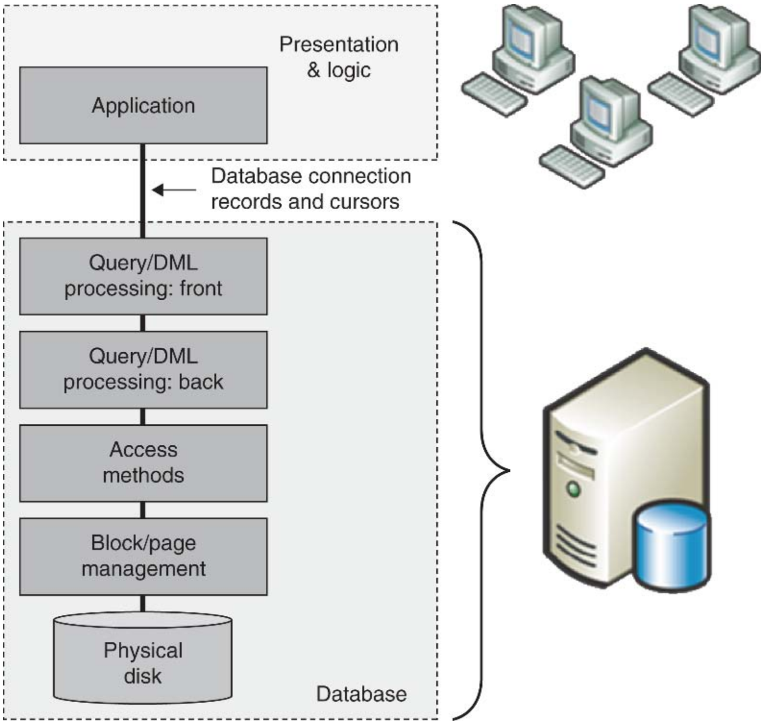
As soon as the 3-tier architecture was introduced, new challenges arose in the management of transactions as they propagated through the system. In 2-tier client-server architectures, the transactional scope was bounded by a time interval on the database connection. Now, in a 3-tier client-server scheme, the work initiated by the Presentation layer may need to be atomic as it propagates through different servers implementing the Logic layer. For the moment, consider this architecture with a single process DBMS at the back-end (See Fig. 5).

New notions in the management of transaction identifiers needed to be created for this to work. The identity of the atomic transaction needed to be propagated with the RPC or other app-specific call from the client to the middle-tier server. Also, the same transaction-id could now arrive at the database from *different* database connections. A window of time using the database-connection could no longer be a surrogate for the transaction. These challenges came as the application that related to the database underwent changes in its process architecture.

Multi-Database Computing

As distributed systems progressed, there were occasions in which a 2-tier client wished to do transactional work across multiple database servers. This was only possible with the arrival of BOTH distributed transactions (implemented with two-phase commit) AND the ability for the client application to create a transaction independent of the database connection and manage the association of the transaction to the connections to the separate databases. This involved both extensions to the client libraries and the creation of DBMS server to DBMS server distributed transaction management.

Most systems did not implement transactional support for multiple databases in 2-tier applications until after this was accomplished for 3-tier applications. The need to have a client call different middle-tier application servers with different databases was the pressure that led to the implementation of two phase commit for distributed transactions across the multiple databases. Only later were the facilities for sharing transactions across database connections directly connected to a single client implemented. See Fig. 6 for a depiction of a 3-tier client-server application where some of the middle-tier servers use different databases all associated with a single common transactional scope.


Presentation & logic
Application
Database connection records and cursors
Query/DML processing: front
Query/DML processing: back
Access methods
Block/page management
Physical disk
Database

Process Structure of a DBMS. Figure 3. Two-tier client-server architecture. The application is separated out from the database. The interaction is maintained with a database connection.

Service-Oriented Architectures (SOA)

A recently popular application architecture is the service-oriented architecture. It is distinguished from the client-server application architecture in that there are no transactions shared across the service boundaries and, indeed, no direct visibility to the partner's database. All access to the database is indirect and mitigated by the application. Because there are no shared transactions across the service boundaries and there are no database connections or semantics across these boundaries, the database management system does not see the SOA architecture.

SOA implementations often use an internal client-server architecture to build out a single service. Still, from the standpoint of the process architecture of a DBMS, the use of an application combined with its database in a larger SOA system is not of any impact on the DBMS process structure.

Shared-Memory (Single Database) DBMS Architectures

In a shared-memory (or sometimes called shared-everything) system, the DBMS runs in a single process or, at least, in a fashion where the processes are able to share their memory. Figure 1 is an example of a shared-memory system, one where multiple processes have a special mechanism to access special (protected) memory. The architecture depicted in Fig. 1 is not a client-server system.

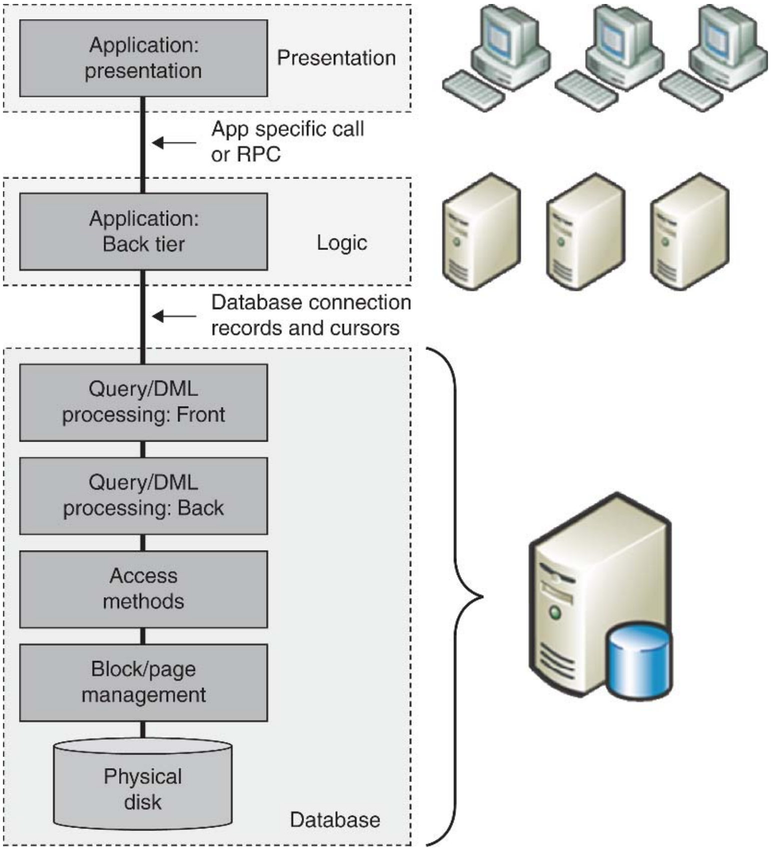
Most shared-nothing systems are a little less exotic and look like Figs. 3–5. There is a single database and it runs on a large process (perhaps with multiple threads within a shared-memory multiprocessor).

Figure 6 is considered to be a shared-everything system *precisely because each database is a separate database*, and there cannot be queries issued across them. There is no way to view the data from these multiple databases except through the assistance of application logic. There cannot exist a single database connection to that pool of databases – they are separate databases.

The presence of distributed transactions spanning multiple databases does not make them a single database.

Shared-Disk (Single Database) DBMS Architectures

Shared-Disk systems comprise a cluster in which multiple independent machines have access to a pool of disks. In addition to some mechanism for sending messages across the machines, each of the computers



Process Structure of a DBMS. Figure 4. 3-tier client-server architecture. The presentation, logic, and database are distributed across different systems.

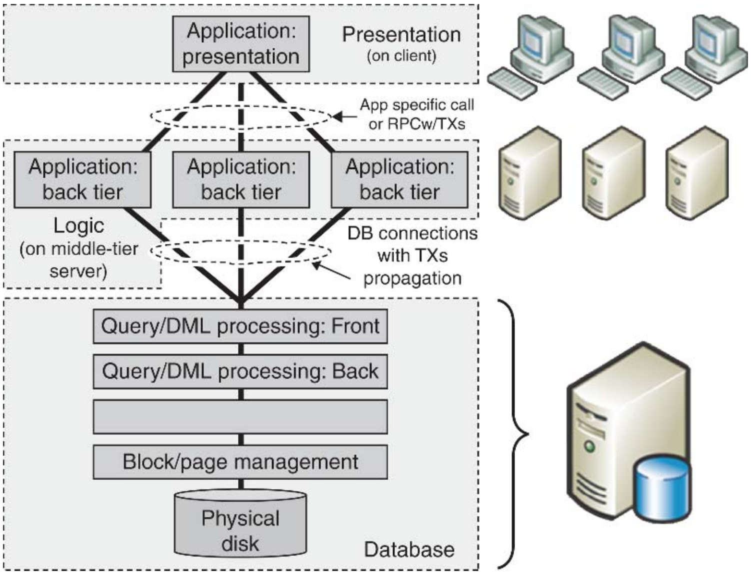
in the cluster can read and write pages of disk across a collection of many disk spindles. Typically, this is implemented with some form of disk controller managing access by the computers to the disks. Figure 7 shows a sketch of the hardware architecture of a shared-disk cluster.

The distinction between the database process architecture versus the application process architecture is sometimes confusing. The first shared-disk systems used block mode terminals and there was no notion of a smart client at that time. As an example, consider Fig. 8 which is a slight modification of Fig. 7. Figure 8 depicts a system in which the business logic (the middle tier of a three tier system) runs on the same processors (or potentially the same processes in the same spirit as shown in Fig. 1). In this example, the business logic of the application as well as the entire processing of the DBMS can run in the same process (or at least the same processor) while still scaling across a multiprocessor cluster with tremendous efficiency if the load characteristics are appropriate for the architecture.

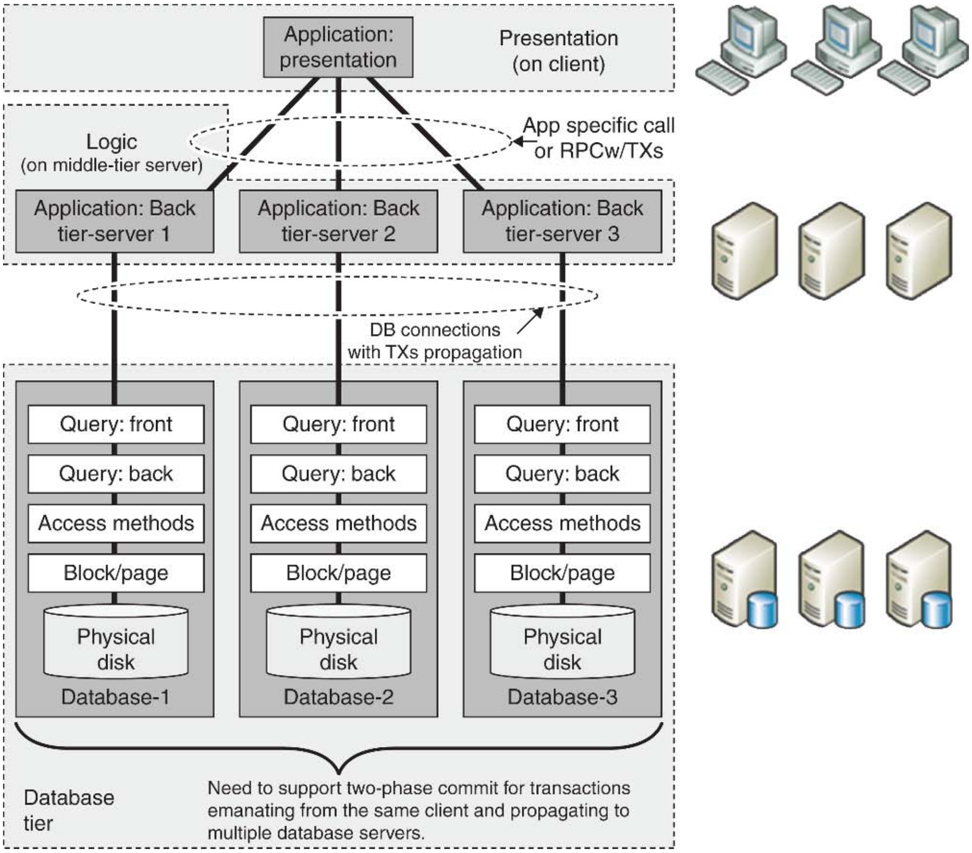
Shared-Disk DBMS systems have the advantage that, once all of the blocks are brought into memory, the entire query can be processed within a single process. This style of distributed implementation delivers high performance when the workload splits into easily partitioned sets of blocks but works less efficiently when the workload has conflicts over the blocks needed by the different processes.

Shared-Nothing (Single Database) DBMS Architectures

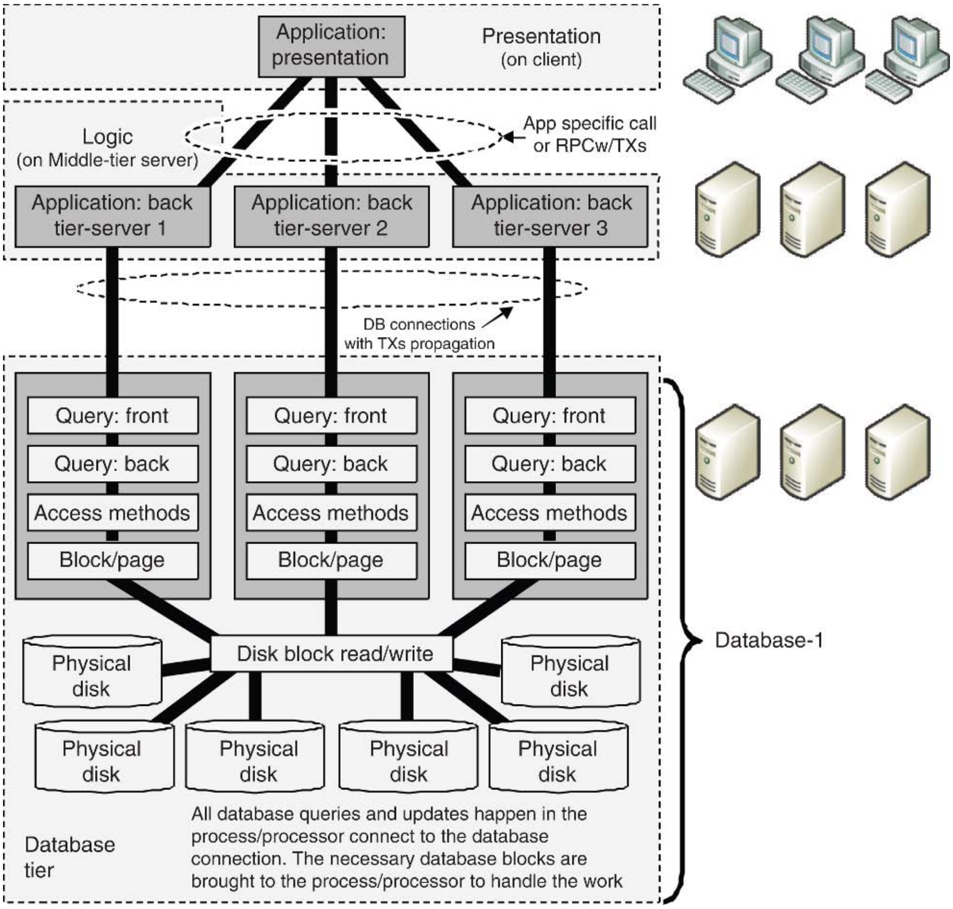
Share-Nothing DBMS architectures are implemented by splitting the query processing engine into a front-end and a back-end. Messaging is used to pass portions of the query or update from the front-end to the back-end. Resulting sets of records are passed from the back-end to the front-end where the completion of the query is performed. The optimizations used in these architectures have been the source of significant research and engineering and can result in fascinating performance gains. Figure 9 shows a Shared-Nothing DBMS in which the application's Business Logic is


Application: presentation
Presentation (on client)
App specific call or RPCw/TXs
Application: back tier
Application: back tier
Application: back tier
Logic (on middle-tier server)
DB connections with TXs propagation
Query/DML processing: Front
Query/DML processing: Back
Block/page management
Physical disk
Database

Process Structure of a DBMS. Figure 5. Building a 3-tier client-server application necessitates both the management of the same transaction coming into the DBMS on different database connections AND the propagation of transactions from the client to the middle-tier server.


Application: presentation
Presentation (on client)
Logic (on middle-tier server)
App specific call or RPCw/TXs
Application: Back tier-server 1
Application: Back tier-server 2
Application: Back tier-server 3
DB connections with TXs propagation
Query: front
Query: back
Access methods
Block/page
Physical disk
Database-1
Query: front
Query: back
Access methods
Block/page
Physical disk
Database-2
Query: front
Query: back
Access methods
Block/page
Physical disk
Database-3
Database tier
Need to support two-phase commit for transactions emanating from the same client and propagating to multiple database servers.

Process Structure of a DBMS. Figure 6. A 3-tier client-server application with DIFFERENT databases not only has to manage the propagation of transactions from client to app-server AND app-server to DBMS, it must in addition manage the atomic two-phase commit of the transactions across database servers.

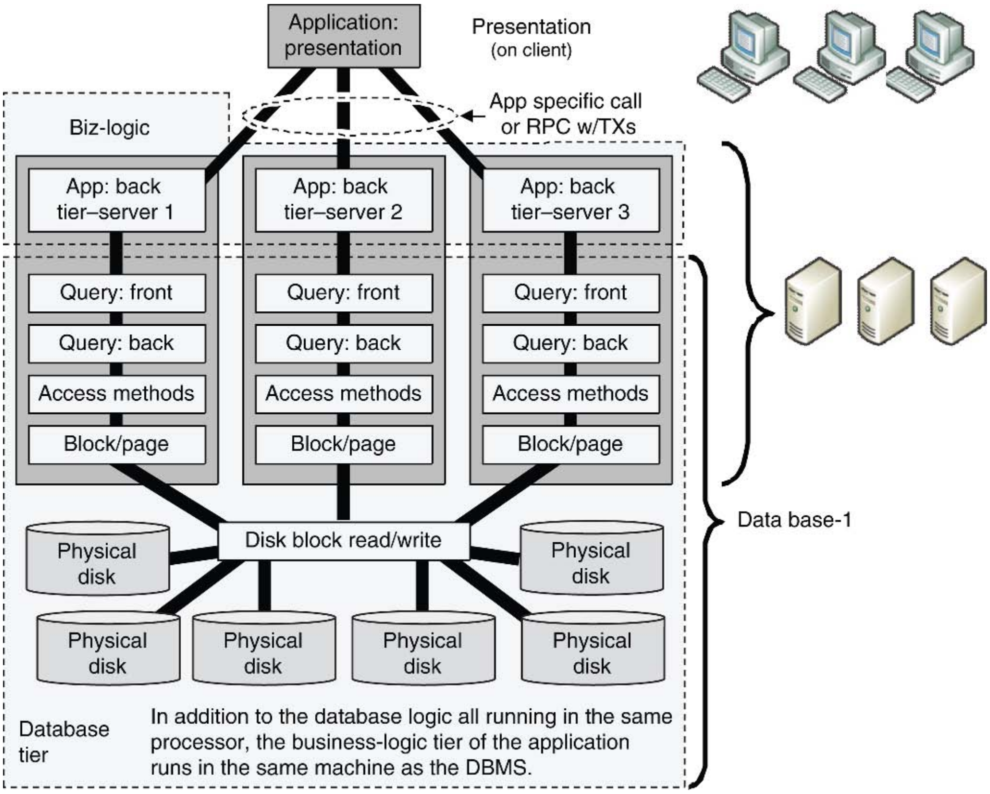

Application: presentation
Presentation (on client)
Logic (on Middle-tier server)
App specific call or RPCw/TXs
Application: back tier-server 1
Application: back tier-server 2
Application: back tier-server 3
DB connections with TXs propagation
Query: front
Query: back
Access methods
Block/page
Disk block read/write
Physical disk
Database-1
Database tier
All database queries and updates happen in the process/processor connect to the database connection. The necessary database blocks are brought to the process/processor to handle the work

Process Structure of a DBMS. Figure 7. A Shared-Disk Database Management System. For the first time in our figures, the single database SPANS multiple processes and, indeed, multiple processors. The single database (with single database semantics presented to the application) runs on many different computers but sharing the access to the physical disks. Special locking infrastructure on the contents of the blocks of disk must be maintained.

running on the same scale out cluster as the DBMS. Just like the Shared-Disk DBMS, it is important to realize that the application's process/processor architecture may take different forms. Running the Business Logic close to the Front-End of the DBMS is one configuration of Shared-Nothing.

Implications of Shared-Nothing, Shared-Disk, and Shared-Everything Architectures

For a number of years, there have been debates in the industry about the strengths and weaknesses of different process architectures for a DBMS system. Before even engaging in these, it is important to remember the delineation of the DBMS architecture from that of a Service Oriented Architecture and, also, from the application architecture of an N-tier system. The term DBMS is used to refer to a single collection of records across which relational operations may occur. Service Oriented Architectures (SOAs) offer an aggregation of computation connected by business logic without the presence of spanning relational operations or transactions. N-tier application environments frequently offer atomic transactions across different databases but do not offer relational operations across the contents of the databases. So, the taxonomy of DBMS process structures refers only to the portion of the system providing a single relational database semantic.

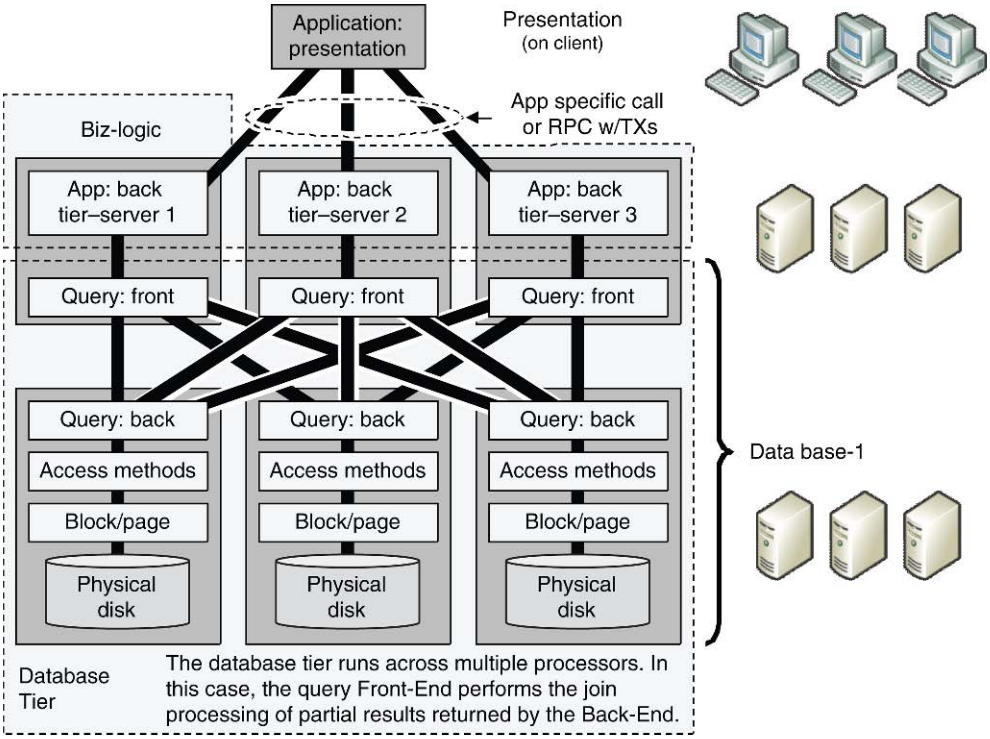


Process Structure of a DBMS. Figure 8. Shared-Disk DBMS within a Two-Tier application architecture.

The *Shared-Everything* DBMS architecture is by far the highest performing architecture in that it offers the most throughput for a single computer (it is not necessarily the most scalable one; scalability is discussed below). Shared-Everything DBMSs do not need to perform any messaging or other cross process communication because they don't cross processes in their implementation of the DBMS. All of the work is done within a single machine. Included in this architecture are various single memory multi-processor implementations. There are few concerns in a Shared-Everything DBMS about the types of queries and usage patterns because everything coexists in a single shared process. Again, Shared-Everything works wonderfully until the DBMS grows too large to fit into one system.

A *Shared-Disk* DBMS architecture allows for additional sharing by letting multiple DBMS processes and processors to access the same physical disks. When the usage pattern of the application has a low probability of conflict in its updates of a shared page, Shared-Disk DBMS systems are very efficient. The pages needed for a query are brought into the processor requesting them and the work of the application's transaction is handled inside one processor of the cluster in what is (hopefully) a very efficient fashion. When the data in question has low update rates, this typically works well. When a single data item is rapidly updated (called a "hot-spot"), this can lead to performance conflicts which cause the block of the database to be pulled back-and-forth across processors. Another challenge occasionally presented by Shared-Disk DBMS systems lies in block (or page) mode locking. Distinct records in the same block may observe lock conflicts causing performance challenges that would not be present in other architectures. Still, Shared-Disk DBMS systems have been wildly successful for many applications and allow many databases to scale beyond what a single shared-memory system could offer.

Shared-Nothing DBMS architectures typically carry a heavier cost to set up a complex transaction but can, in some cases, offer greater throughput over a particular amount of data. It is typical for a Shared-Nothing system to offer record locking. For systems with very high throughput "hot-spots," the data for the



Process Structure of a DBMS. Figure 9. A Shared-Nothing DBMS running with an application configuration which places the business logic tier on the same cluster as the Shared-Nothing DBMS. Also, in this configuration, the Query Front-End of the DBMS runs on the same processors as the Business-Logic.

"hot-spot" does not move around the cluster and it is possible for the system to get more transactions over the same piece of data in a fixed period of time. This is sometimes referred to as "moving the operation to the data" (Shared-Nothing) rather than "moving the data to the operation" (Shared-Disk).

When a database can fit into a single shared-memory multiprocessor, Shared-Everything offers distinct advantages (at least until the application's demands exceed the single machine and you have a problem). For databases that exceed this size and yet want full database semantics, there is a lively debate within the community about which architecture is better and different applications offer different performance characteristics.

Key Applications

Database process structures are an essential part of scaling past a single system. As discussed, the distinction between an application multi-processor architecture and a DBMS multi-processor architecture has many nuances.

Cross-references

- ▶ Application Server
- ▶ Client-Server Architecture
- ▶ Clustering
- ▶ Database Client
- ▶ Distributed Concurrency Control
- ▶ Distributed Database Systems
- ▶ Distributed DBMS
- ▶ Distributed Query Processing
- ▶ Distributed Transaction Management
- ▶ Multi-Tier Architecture
- ▶ ODBC
- ▶ Service Oriented Architecture
- ▶ Shared-Disk Architecture
- ▶ Shared-Memory Architecture
- ▶ Shared-Nothing Architecture
- ▶ Two-Phase Commit
- ▶ Multi-Tier Architecture

Recommended Reading

1. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Mateo, CA, 1992.

2. Michael S. (UC Berkeley). The case for shared nothing architecture. Database Eng., 9(1):4–9, 1986.
3. Oracle RAC (Real Application Clusters). http://www.oracle.com/database/rac_home.html
4. Susanne E., Jim G., Terrye K., and Praful S. A Benchmark of NonStop SQL Release 2 Demonstrating Near Linear Speedup and Scaleup on Large Databases. In Proc. 2000 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Comp. Syst., 1990, pp. 24–35.
5. The Tandem Database Group. NonStop SQL: a distributed high performance, high availability implementation of SQL. In Proc. of 2n High Performance Transaction Processing Workshop, 1989.

Processing Overlaps

Georgina Ramírez
Yahoo! Research Barcelona, Barcelona, Spain

Synonyms

Removing overlap; Controlling overlap

Definition

In semi-structured text retrieval, processing overlap techniques are used to reduce the amount of overlapping (thus redundant) information returned to the user. The existence of redundant information in result lists is caused by the nested structure of semi-structured documents, where the same text fragment may appear in several of the marked up elements (see Fig. 1). In consequence, when retrieval systems perform a focused search on this type of document and use the marked up elements as retrieval objects, very often result lists contain overlapping elements. In retrieval applications where it is assumed that the user does not want to see the same information twice, it may be necessary to reduce or completely remove this overlap and return a ranked list of no overlapping elements. Thus, depending on the underlying user model and retrieval application, different processing overlap techniques are used in order to decide, given a set of relevant but overlapping elements, what are the most appropriate elements to return to the user.

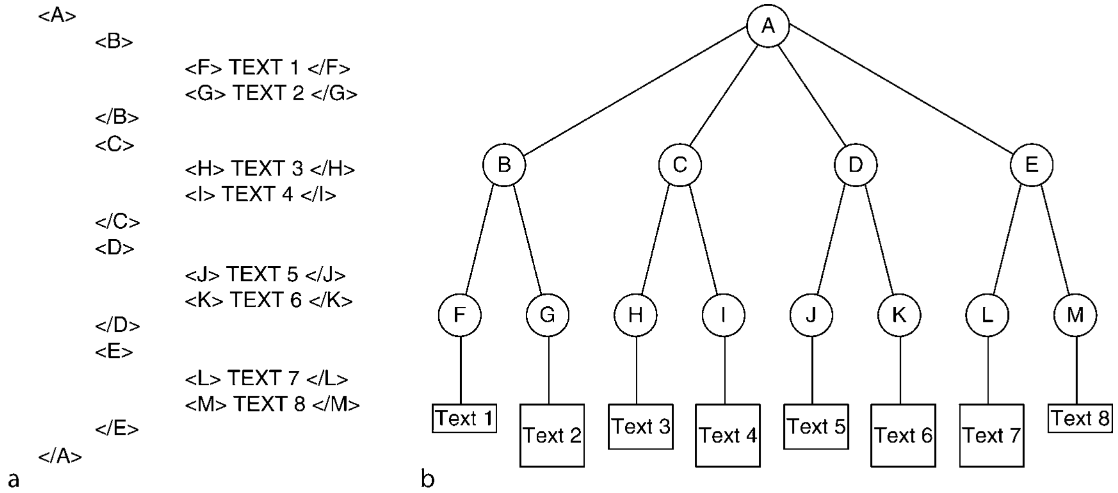
Historical Background

Although the problem of overlap in semi-structured text retrieval is as old as the semi-structured documents themselves, not much work has been published on processing techniques for reducing or removing overlap. Some related work can be found in the area of passage retrieval, where approaches that use a varying window size for passage selection might produce result lists with overlapping passages. However, most of this work is performed on unstructured documents and the approaches taken for processing overlap tend to be simpler.

In the domain of semi-structured documents, there are several areas where different overlap issues are studied. For example, there is quite some work in the area of evaluation of XML systems that addresses the so called overlap problem (e.g., [3]). A different overlap problem is created by the possibility that standards like SGML provide of having multiple annotations (markups) on the same document (a.k.a. multiple hierarchies). In this case the overlap is produced by the structure of the different annotations. Since the multiple hierarchies complicate the use of standard retrieval techniques on this type of documents, work on this area is still focusing on addressing other indexing and retrieval issues. It is only recently that in the domain of XML documents, several approaches have been presented that address the problem of processing overlap from result lists containing overlapping elements. The next section summarizes some of them.

Foundations

One of the advantages of semi-structured documents is that retrieval systems can perform focused search by simply using the marked up divisions of the documents (elements) and retrieving those instead of the whole documents. However, since elements overlap with each other (see Fig. 1), when using traditional ranking techniques to independently rank these elements, result lists often contain many overlapping elements. This is due to the nested structure of semi-structured documents, where the same text fragment is usually contained in several of the marked up elements. Thus, when a specific element is estimated relevant to the query, all the elements containing this element (a.k.a. ancestors) will also be estimated to some degree relevant to the query. Furthermore, this element is probably estimated relevant because it contains several relevant elements (a.k.a. descendants). For example, if a section of a document is estimated highly relevant, it probably contains several highly relevant paragraphs and it is contained in a relevant article. If all of these elements are returned to the user, the amount of redundant information contained in the result list will be considerable. In retrieval scenarios where users do not like to see the same information twice, retrieval systems



```
<A>
    <B>
            <F> TEXT 1 </F>
            <G> TEXT 2 </G>
    </B>
    <C>
            <H> TEXT 3 </H>
            <I> TEXT 4 </I>
    </C>
    <D>
            <J> TEXT 5 </J>
            <K> TEXT 6 </K>
    </D>
    <E>
            <L> TEXT 7 </L>
            <M> TEXT 8 </M>
    </E>
</A>
```

Processing Overlaps. Figure 1. Example of a semi-structured document and its tree structure representation. Note that each fragment of the document is contained in three different elements (nodes in the tree).

need to decide which of these relevant but overlapping elements is the most appropriate piece of information to return to the user. The final decision on which elements the system should return depends on the search application and the underlying user model but a common goal is to reduce redundancy in the result lists. Although this can be done at indexing time (e.g., by selecting a subset of non-overlapping elements as potential retrievable objects), commonly this is done by removing overlapping elements from the result set, after retrieval systems have produced an initial ranking of all elements. Processing overlap techniques have recently been widely discussed in the domain of XML retrieval, where different approaches have been presented. The rest of this section presents and discusses some of them.

Using Element Types

A simple way to reduce overlap is to select a subset of element types and consider only these for retrieval. For example, if sections of documents are considered to be the most appropriate pieces of information, retrieval systems might want to use only these as retrievable objects and ignore all the other element types. This can be done at indexing time or by post-filtering the result lists. Depending on the number and element types selected, overlap is reduced at different degrees. Note that it might not always be possible to completely remove overlap; even if a single element type is selected as a unique retrievable object, it is still possible that elements of the same type overlap each other. The main drawback of this approach is that, since it is desirable to select element types that are likely to be relevant and useful to the user, it requires knowledge of the structure of the documents and its common usage.

Using Paths

A common approach for processing overlaps keeps the highest ranked element on each path and removes its ancestors and descendants from the result list, i.e., all the elements in the result list that contain or are contained within it (e.g., [6], [2]). It is important to notice that depending on the order in which the different paths are processed different outputs might be produced.

In [6] the authors present a two steps algorithm to remove the overlap. The first step is used to select the highest scored element from each relevant path. Since their algorithm selects the elements from the different paths simultaneously, the output may still contain overlapping elements. That is why a second step is needed, to completely remove overlap. This is done by selecting again (this time from the output of the first step) the highest scored element from each path:

Algorithm 1

1. Select highest scored element from each relevant path.
2. Select highest scored element from each relevant path in output of step 1.

Another common way to remove overlap using paths [5] is to recursively process the result list by selecting the highest ranked element and removing any element from lower ranks that belongs to the same path (it contains the selected element or it is contained within it):

Algorithm 2

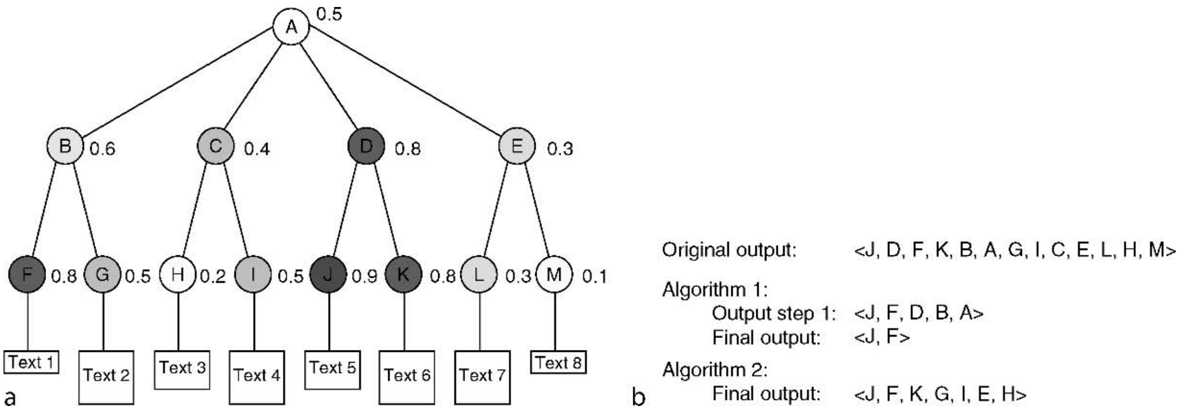
1. Return highest ranked element from result list.
2. Remove from result list all the elements belonging to the same path.
3. Repeat step 1 and 2 until result list is empty.

The underlying assumption of this type of approaches is that the most appropriate piece of information in each path has been assigned a higher score than the rest and therefore, removing overlap is simply a presentation issue. These approaches rely completely on the underlying retrieval models to produce the best ranking. This could indeed be the case if the retrieval model would consider, when ranking, not only the estimated relevance of the element itself but also its *appropriateness* compared to other elements in the same path. However, since many retrieval models rank elements independently, the highest scored element may not be the most appropriate one, i.e., the one the user prefers to see.

To illustrate the different outputs of the previous algorithms, have another look at the example document from Fig. 1. Imagine now that, given a query, the retrieval model estimates the relevance of each element in the document. Figure 2 shows the retrieval scores obtained by each of the elements and the outputs produced when removing overlap with the algorithms described above. Both algorithms produce a result list of non-overlapping elements. However, there are substantial differences. The main drawback of the first algorithm is that it might miss some relevant information. For example, one could argue that element K should also be contained in the output list. To be able to do that, the algorithm should consider structural relationships between elements and re-add element K to the result list when it decides to remove element D in the second step.

Although the second algorithm produces a more complete list, someone could argue that the output produced is not the most desirable. For example, imagine that elements F and G are, respectively, the title and the abstract of the document. In this case, even if the title has been ranked high (it may contain most or all of the query terms), users might prefer to see a result item containing both, title and abstract (i.e., element B) instead of seeing both elements independently. In general, it can be argued that retrieval systems should only return those elements that have enough content information to be useful and can stand alone as independent objects. A similar example can be seen for elements D, J, and K. Even if J is ranked higher, element D contains mostly relevant information and therefore, it might be more desirable (from a user perspective) to read element D, than J, and K independently. Furthermore, for elements E, L, and M it could be argued that it is better to return L than E because the relevance estimated for element E is due to the content of L and no extra benefit is obtained when returning E.

In these, cases a better output might be produced if the algorithms would consider structural relationships between elements when deciding which elements to return to the user.



Original output: <J, D, F, K, B, A, G, I, C, E, L, H, M>
Algorithm 1:
Output step 1: <J, F, D, B, A>
Final output: <J, F>
Algorithm 2:
Final output: <J, F, K, G, I, E, H>

Processing Overlaps. Figure 2. Example of a retrieval run and the resulting outputs when removing overlapping elements with algorithm 1 and 2. The numbers on the tree indicate the initial retrieval scores obtained by each of the elements.

Using Structural Relationships for Re-Ranking

The following approaches present more advanced techniques that exploit the structural relationships within a document to decide which elements should be removed or pushed down the ranked list and which ones should be returned to the user (e.g., [5], [1], [4]). These techniques often modify the initial ranking predicted by the retrieval model.

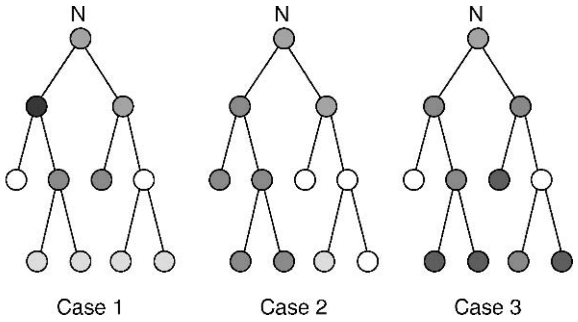
In [1] elements are re-ranked by adjusting the element scores of the lower ranked elements according to their containment relationship with other higher ranked elements. The assumption underlying this approach is that the score of the elements that are contained or contain other elements that have been already shown to the user (i.e., are ranked higher) should be adjusted in order to reflect that the information they contain might be redundant. They do that by reducing the importance of terms occurring in already reported elements. The basic algorithm is similar to algorithm 2 but instead of removing the ancestors and descendants of the reported elements, their scores get adjusted:

Algorithm 3

1. Report the highest ranked element.
2. Adjust the scores of the unreported elements.
3. Repeat steps 1 and 2 until m elements are reported.

The author also presents an extended version of the algorithm where different weighting values are used for ancestors and descendants and where the number of times an element is contained within others is considered. For example, a paragraph contained in an already seen section and in an already seen article is further punished because the user has already seen this information twice. Note that this algorithm is not designed to remove the overlap but to push down the result list those elements that contain redundant information.

In [4] a completely different approach is taken. The authors present a two step re-ranking algorithm for removing overlap. The first step identifies clusters of highly ranked results and picks the most relevant element from each cluster. The second round is used to remove any remaining overlap between the selected elements. The selection criteria for the first step is based on three different cases (illustrated in Fig. 3): (i) if an element N has a descendant that is substantially more relevant, the element N is removed from the result list, (ii) if case 1 does not hold and the element N has a child that contains most of the relevant information (the relevant elements are concentrated under this child), the element N is also removed from the result list, and (iii) if none of the previous cases hold and the results are evenly distributed under the element N, then the element N is kept and all its descendants removed from the result list. In the rest of the cases they do not do anything and leave the final overlap removal for the second phase. In the second step, they remove overlap by comparing the score of each element with the ones of its descendants. If the score of the element is bigger, all the descendants are removed and the element is kept. Otherwise, the element is removed.



Processing Overlaps. Figure 3. Illustration of the three cases considered in the approach presented in [4]. The grayer the node, the higher the relevancy estimated for that node.

In [5] the authors present an approach that makes use of an *utility* function that captures the amount of *useful* information contained in each element. They argue that to model the *usefulness* of a node three important aspects need to be considered: (i) the relevance score estimated by the retrieval model, (ii) the size of the element, and (iii) the amount of irrelevant information the element contains. They present an algorithm that selects elements according to the estimated *usefulness* of each element. If an element has an estimated *usefulness* value higher than the sum of the *usefulness* values of its children, then the element is selected and the children are removed. Otherwise, the children elements whose *usefulness* value exceeds some threshold are selected and the element is removed.

Key Applications

Processing overlap techniques are needed in any retrieval application where users need to be directed to specific parts of documents and there are not predefined retrieval units.

Experimental Results

For every presented approach, there is an accompanying experimental evaluation in the corresponding reference. Commonly, the referred article provides an overview of the performance of the approach and of its variations. However, it is difficult to use the reported evaluations to compare approaches between articles. The main reason is that all approaches make use of different retrieval models for their initial runs, thus it is not clear whether the performance obtained after removing overlap is due to the underlying retrieval model or to the approach used to remove the overlap. Besides, not all the presented approaches experiment on the same dataset or use the same evaluation measures to report their numbers.

In [5] the authors present an experimental comparison between several of the approaches described above. They show that the best performing approach is the one that returns only paragraphs. As a general trend for the first type of approach (the ones that select a specific element type), the longer the element type selected, the worse the performance. The authors also show that their approach of estimating the *usefulness* of an element can help to improve retrieval performance (in terms of precision at low recall levels) when compared to the approach of using paths (algorithm 2).

Data Sets

Since 2005 the *INitiative for the Evaluation of XML Retrieval* (INEX) provides a data-set that can be used to test processing overlap strategies (see http://inex.is.informatik.uni-duisburg.de/).

Cross-references

▶ INEX
▶ Structured Document Retrieval
▶ XML Retrieval

Recommended Reading

1. Clarke C.L.A. Controlling overlap in content-oriented XML retrieval. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 314–321.
2. Geva S. GPX – gardens point XML IR at INEX 2005. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrievals, 2006, pp. 240–253.
3. Kazai G., Lalmas M., and de Vries A.P. The overlap problem in content-oriented XML retrieval evaluation. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 72–79.
4. Mass Y. and Mandelbrod M. Using the INEX environment as a test bed for various user models for XML retrieval. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrievals, 2006, pp. 187–195.
5. Mihajlovi V., Ramírez G., Westerveld T., Hiemstra D., Blok H.E., and de Vries A.P. TIJAH scratches INEX 2005: vague element selection, image search, overlap and relevance feedback. 2006, pp. 72–87.
6. Sauvagnat K., Hlaoua L., and Boughanem M. XFIRM at INEX 2005: ad-hoc and relevance feedback tracks. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrievals, 2006, pp. 88–103.

Processing Structural Constraints

Andrew Trotman
University of Otago, Dunedin, New Zealand

Definition

When searching unstructured plain-text the user is limited in the expressive power of their query – they can only ask for documents that are about something. When structure is present in the document, and with a query language that supports its use, the user is able to write far more precise queries. For example, searching for "smith" in a document is not necessarily equivalent to searching for "smith" as an author of a document. This increase in expressive power should lead to an increase in precision with no loss in recall. By specifying that "smith" should be the author, all those instances where "smith" was the profession will be dropped (increasing precision), while all those in which "smith" is the author will still be found (maintaining recall).

Historical Background

With the proliferation of structured and semi-structured markup languages such as SGML and XML came the possibility of unifying database and information retrieval technologies. The Evaluation of XML Retrieval (INEX) was founded in 2002 to examine the use of semi-structured data for both technologies. It was expected that the use of structure would not only unify the two technologies but would also improve the performance of both.

Foundations

User Querying Behavior

When using an information retrieval search engine the user typically has some information need. This information need is expressed by the user as a keyword query. There are many different queries that could be drawn from the same information need. Some might contain only keywords, others phrases, and others a combination of the two. It is the task of the search engine to satisfy the information need given the query. Because the task is not to satisfy the query, the terms in the query can be considered nothing more than hints by the user on how to identify relevant documents. It is likely that some relevant documents will not contain the user's keywords, while others that do might not be relevant.

If the user does not immediately find an answer they will often change their query, perhaps by adding different keywords, or by removing keywords. If the query syntax permits they might add emphasis markers (plus and minus) to some terms.

With a few exceptions semi-structured search engines remain experimental, so user behavior cannot be studied in a natural environment. Instead the user behavior is expected to mirror that of other search engines, or search engines that include some structural restriction.

The model used at INEX is that a user will give a query containing only search terms, then, if they are dissatisfied with the results, they might add structural constraints to their query. The keyword searches are known as Content Only (CO) and when structure is added as Content Only + Structure (CO + S) or Content And Structure (CAS) queries. Just as the keywords are hints, so too are the structural constraints. For this reason they are commonly referred to as structural hints.

The addition of structure to an otherwise content only search leads to a direct comparison of the performance of a search engine before and after the structural constraint has been added. The two queries are instantiations of the same information need, so the same documents or document components are relevant to each query making a direct comparison meaningful.

The analysis of runs submitted to INEX 2005 (against the IEEE document collection) showed no statistical difference in performance between the top CO and top CO + S runs – having structural hints in the query did not improve performance [12]. Even at low levels of recall (1 and 10%) no significant improvement was seen. About half the systems showed a performance gain, the other half no gain.

There are several reasons why improvements are not seen: first it could be a consequence of the structure present in the IEEE collection; second (and more likely) it could be that users are not proficient at providing structural hints.

The result was backed up by a user study [13] in which users were presented with three ways of querying the document collection: keywords, natural language (including structure), and Bricks [15] (a graphical user interface). Sixteen users each performed six simulated work tasks, two with each interface. The same conclusion is drawn, that is no significant improvement was seen when structure was used in querying.

Structural Constraints

There are two reasons a user might add structural constraints to a query. The first is to constrain the size of the result. When searching a collection of textbooks it is, perhaps, of little practical use to identify a book that satisfies the user need. A better result might be a chapter from the book, or a section from the chapter, or even a single paragraph. One way to identify the best granularity of result is to allow the user to specify this as part of the query. These elements are known as target elements.

The user may also wish to narrow the search to just those parts of a document he or she knows to be appropriate. In this case the user might search for "smith" as an author in order to disambiguate the use from that as any of: an author, a profession, a street, or a food manufacturer. Restricting a query to a given element does not affect the granularity of the result; instead it lends support on where to look so such elements are known as support elements. Both target elements and support elements can appear in the same query.

It is not at all obvious from a query whether or not the user expects the constraint to be interpreted precisely (strictly) or imprecisely (vaguely). In the case of "smith" as an author, it is likely that "smith" as a profession is inappropriate, but "smith" as an editor might be appropriate. If the target element is a paragraph, then a document abstract (about the size of a paragraph) is likely to be appropriate, but a book not so.

The four possible interpretations of a query were examined at INEX 2005 [11]. Runs that perform well with one interpretation of the target elements do so regardless of the interpretation of the support elements. The interpretation of the target element does, however, matter. The consequence is that the search engine needs to know, as part of the query, whether a strict or vague interpretation of the target element is expected by the user.

Processing Structural Constraints

Given a search engine, the strict interpretation of target elements can be satisfied by a simple post-process eliminating all results that do not match. As just discussed above, strictly processing support elements has been shown to be unnecessary.

Several techniques for vaguely satisfying target element constraints have been examined including ignoring them, pre-generating a set of tag-equivalences, boosting the score of elements that match the target element, and propagating scores up the document tree.

Ignoring Structural Constraints

Structural constraints might be removed from the query altogether and a Content Only search engine used to identify the correct granularity of result.

Tag Equivalence

A straightforward method for vaguely processing structural constraints is tag equivalence. A set of informational groups are chosen *a priori* and all tags in the DTD (DTD is the Document Type Definition, specifying the format of the XML documents forming the collection.) are mapped to these groups. If, for example, <p> is used for paragraphs and <ip> is used for initial paragraphs, these would be grouped into a single paragraph group.

Mass and Mandelbrod [5] *a priori* choose appropriate retrieval units (target elements) for the document collection and build a separate index for each. The decision about which units these are is made by a human before indexing. A separate index is built for each unit and the search is run in parallel on each index. Within each index the traditional vector space model in used for ranking. The lexicon of their inverted index contains term and context (path) information making strict evaluation possible. Vague evaluation of paths is done by matching lexicon term contexts against a tag-equivalence list.

Mihajlović et al. [6] build their tag equivalence lists using two methods, both based on prior knowledge of relevance. For INEX 2005 they build the first list by taking the results from INEX 2004 and selecting the most frequent highly and fairly relevant elements and adding the most frequently seen elements from the queries. In the second method they take the relevant elements from previous queries targeting the same element and normalize a weight by the frequency of the element in the previous result set (the training data, in this case INEX 2004). Using this second method they automatically construct many different tag equivalence sets using the different levels of relevance seen in the training data.

In a heterogeneous environment in which many different tags from many different DTDs are semantically but not syntactically identical, techniques from research into schema-matching [1] might be used to automatically identify tag equivalence lists.

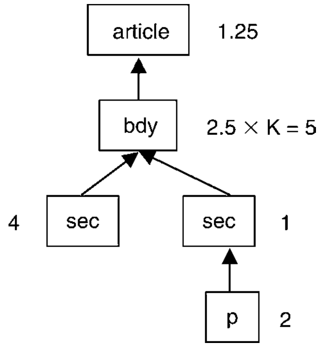
Structure Boosting

Van Zwol [14] generates a set of results ignoring structural constraints then boosts the score of those that do match the constraints by linearly scaling by some tag specific constant. The consequence is to boost the score of elements that match the structural constraints while not removing those that do not. A score penalty is also used for deep and frequent tags in the expectation of lowering the score of highly frequent (and short) tags. A similar technique is used by Theobald et al. [9] who use it with score propagation.

Score Propagation

Scores for elements at the leaves of the document tree (that is, the text) are computed from their content. Scores for nodes internal to the document tree are computed from the leaves by propagating scores up the tree until finally a score for the root is computed. Typically as the score propagates further up the tree, its contribution to the score of an ancestor node is reduced (see the entry on *Propagation* based structured text Retrieval for details).

Figure 1 illustrates score propagation. A search term is found to occur two times in the p element, and four times in the (left) sec element. With a decay factor of 0.5, the score of the bdy element is computed as 4 *0.5 + 2 * 0.5 * 0.5 = 2.5. The score for the article element is computed from that score likewise. If the target element is bdy and the score, for example, is boosted by $K = 5$, then the element with the highest



Processing Structural Constraints. Figure 1. Score Propagation, each time a score is propagated the score is weakened (in the example: halved), but at target nodes it is boosted (in the example: K = 2).

score is that element. The score for K and the propagation value are chosen here for illustrative purposes only and should be computed appropriately for a given document collection.

Hurbert [3] uses score propagation with structure reduction – if a node in the tree does not match a constraint in the query then the score there is reduced by some factor. In this way all nodes in the tree obtain scores but those matching the constraints are over-selected for. Sauvagnat et al. [8] use score propagation in a similar way but in combination with tag equivalence.

Key Applications

Retrieval of document components from structured document collections.

Future Directions

The best performing search engines that interpret structural constraints have not yet significantly outperformed those that ignore them. Several reasons have been forwarded.

There is evidence to suggest specifying a structural constraint is difficult for a user. Studies into the use of structure in INEX queries suggest that even expert users, when asked to give structured queries, give simple queries [12]. This is inline with studies that show virtually no use of advanced search facilities on the web.

Structure aware search engines are not as mature as web search engines and as yet the best way to use structural constraints (when present in a query) is unknown. The annual INEX workshop provides a forum for testing and presenting new methods.

Improvements were not seen when the IEEE document collection was used, but this result may not generalize to all collections. Alternative collections including newspapers, radio broadcast, and television, have been suggested [7,10]. In 2006, INEX switched to using the Wikipedia as the primary test collection for *ad hoc* retrieval, but a comparative study on that collection has not yet been conducted.

Relevance feedback including structural constraints has been examined. Users might provide feedback on both the desired content and the preferred target element, or just one of these. Evidence suggests that including structure in relevance feedback does improve precision.

Experimental Results

Evidence that use of structure increases precision is tentative. In the XML search engine of Kamps et al. [4], no significant difference is seen overall, however significant differences are seen at early recall points (the first few tens of documents). The search engine of Geva [2] recently performed better without structure than with.

At INEX 2005, a comparative analysis of performance with and without structural constraints on the same set of information needs was performed [1]. The best structure run was compared to the best non-structure run and no significant difference was found. Not even a significant difference at early recall points was found. On a system by system basis about half the search engines show a performance increase.

Cross-references

- ▶ Content-and-Structure Query
- ▶ Content-Only Query
- ▶ INitiative for the Evaluation of XML Retrieval
- ▶ Mean Average Precision
- ▶ Narrowed Extended XPath 1
- ▶ Propagation-based Structured Text Retrieval
- ▶ XML Retrieval

Recommended Reading

1. Doan A. and Halevy A.Y. Semantic integration research in the database community: a brief survey. AI Magazine, 26(1): 83–94, 2005.
2. Geva S. GPX – gardens point XML IR at INEX 2006. In Proc. 5th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2007, pp. 137–150.

3. Hubert G. XML retrieval based on direct contribution of query components. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 172–186.
4. Kamps J., Marx M., Rijke M.D., and Sigurbjörnsson B. Articulating information needs in XML query languages. Trans. Inf. Sys., 24(4):407–436, 2006.
5. Mass Y. and Mandelbrod M. Using the INEX environment as a test bed for various user models for XML retrieval. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 187–195.
6. Mihajlovic V., Ramírez G., Westerveld T., Hiemstra D., Blok H.E., and de Vries A.P. Vtijah scratches INEX 2005: Vague element selection, image search, overlap, and relevance feedback. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 72–87.
7. O'Keefe R.A. If INEX is the answer, what is the question? In Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 54–59.
8. Sauvagnat K., Hlaoua L., and Boughanem M. Xfirm at INEX 2005: ad-hoc and relevance feedback tracks. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 88–103.
9. Theobald M., Schenkel R., and Weikum G. Topx and xxl at INEX 2005. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 282–295.
10. Trotman A. Wanted: element retrieval users. In Proc. INEX 2005 Workshop on Element Retrieval Methodology, 2005, pp. 63–69.
11. Trotman A. and Lalmas M. Strict and vague interpretation of XML-retrieval queries. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 709–710.
12. Trotman A. and Lalmas M. Why structural hints in queries do not help XML retrieval. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 711–712.
13. Woodley A., Geva S., and Edwards S.L. Comparing XML-IR query formation interfaces. Australian J. Intelligent Inf. Proc. Syst., 9(2):64–71, 2007.
14. van Zwol R. B^3-sdr and effective use of structural hints. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 146–160.
15. van Zwol R., Baas J., van Oostendorp H., and Wiering F. Bricks: the building blocks to tackle query formulation in structured document retrieval. In Proc. 28th European Conf. on IR Research, 2006, pp. 314–325.

Processor Cache

Peter Boncz
CWI, Amsterdam, The Netherlands

Synonyms

Data cache; Instruction cache; CPU cache; L1 cache; L2 cache; L3 cache; Translation Lookaside Buffer (TLB)

Definition

To hide the high latencies of DRAM access, modern computer architecture now features a memory hierarchy that besides DRAM also includes SRAM cache memories, typically located on the CPU chip. Memory access first check these caches, which takes only a few cycles. Only if the needed data is not found, an expensive memory access is needed.

Key Points

CPU caches are SRAM memories located on the CPU chip, intended to hide the high latency of accessing off-chip DRAM memory. Caches are organized in cache lines (typically 64 bytes). In a fully-associative cache, each memory line can be stored in any location of the cache. To make checking the cache fast, however, CPU caches tend to have limited associativity, such that storage of a particular cache line is possible in only 2 or 4 locations. Thus only 2 or 4 locations need to be checked during lookup (these are called 2-resp. 4-way associative caches). The cache hit ratio is determined by the spatial and temporal locality of the memory accesses generated by the running program(s).

Cache misses can either be compulsory misses (getting the cache lines of all used memory once), capacity misses (caused by the cache being too small to keep all multiply used lines in cache), or conflict misses (due to the limited associativity of the cache).

Most modern CPUs have at least three independent caches: an instruction cache to speed up executable instruction fetch, a data cache to speed up data fetch and store, and a Translation Lookaside Buffer (TLB) used to speed up virtual-to-physical address translation for both executable instructions and data. The TLB is not organized in cache lines, it simply holds pairs of (virtual, logical) page mappings, typically a fairly limited amount (e.g., 64). In practice, this mean that algorithms that repeatedly touch memory in more than 64 pages (whose size is often 4 KB) shortly after each other, run into TLB thrashing. This problem can sometimes be mitigated by setting a large virtual memory page size, or by using special large OS pages (sometimes supported in the CPU with a separate, smaller, TLB for large pages).

Another issue is the tradeoff between latency and hit rate. Larger caches have better hit rates but longer latency. To address this tradeoff, many computers use multiple levels of cache, with small fast caches backed up by larger slower caches. Multi-level caches generally operate by checking the smallest Level 1 (L1) cache first; if it hits, the processor proceeds at high speed. If the smaller cache misses, the next larger cache (L2) is checked, and so on, before external memory is checked. As the latency difference between main memory and the fastest cache has become larger, some processors have begun to utilize as many as three levels of on-chip cache.

For multi-CPU and multi-core systems, the fact that some of the higher levels of cache are not shared, yet provide coherent access to shared memory, causes additional cache-coherency inter-core communication to invalid stale copies of cache lines on other cores when one core modifies it. In multi-core CPUs, an important issue is which cache level is shared among all cores – this cache level is on the one hand a potential hot-spot for cache conflicts, on the other hand provides an opportunity for very fast inter-core data exchange.

In case of sequential data processing, the memory controller or memory chipset in modern computers often detect this access pattern and start requesting the subsequent cache lines in advance. This is called hardware prefetching. Prefetching effectively allows to hide compulsory cache misses. Without prefetching, the effective memory bandwidth would equate cache line size divided my memory latency (e.g., 64/50 ns = 1.2 GB/s). Thanks to hardware prefetching, modern computer architectures reach four times that on sequential access. Modern CPUs also offer explicit prefetching instructions, which a software writer can exploit to perform (non-sequential) memory accesses in advance, hiding their latency. In database systems, such software prefetching has successfully been used in making hash-table lookup faster (e.g., in hash-join and hash-aggregation).

In database systems, a series of cache-conscious data storage layouts (e.g., DSM and PAX) have been proposed to improve cache line usage. Also, a number of cache-conscious query processing algorithms, such as cache-partitioned hash join and hash-join using memory prefetching, have been studied. In the area of data structures and theoretical computer science, there has recently been interest in cache-oblivious algorithms, that regardless the exact parameters of the memory hierarchy (number of levels, cache size, cache line sizes and latencies) perform well.

Cross-references

- ▶ Architecture-Conscious Database System
- ▶ Cache-Conscious Algorithms
- ▶ Disk
- ▶ DSM
- ▶ Main Memory
- ▶ Main Memory DBMS
- ▶ PAX Storage Layouts

Production-based Approach to Media Analysis

▶ Computational Media Aesthetics

Projected Clustering

▶ Subspace Clustering Techniques

Projection

Cristina Sirangelo
University of Edinburgh, Edinburgh, UK

Definition

Given a relation instance R over set of attributes U, and given a subset X of U, the projection of R on X – denoted by $\pi_X(R)$ – is defined as a relation over set of attributes X whose tuples are the restriction of tuples of R to attributes X. That is $t \in \pi_X(R)$ if and only if $t = t'(X)$ for some tuple t' of R (here $t'(X)$ denotes the restriction of t' to attributes X).

Key Points

The projection is one of the basic operators of the relational algebra. It operates by "restricting" the input relation to some of its columns.

The arity of the output relation is bounded by the arity of the input relation. Moreover the number of tuples in $\pi_X(R)$ is bounded by the number of tuples in R. In particular, the size of $\pi_X(R)$ can be strictly smaller than the size of R since different tuples of R may have the same values on attributes X.

As an example, consider a relation *Goods* over attributes (*code*, *price*, *quantity*), containing tuples

{(001, 5.00, 10), (002, 5.00, 10), (003, 25.00, 3)}. Then $\pi_{price,quantity}(Goods)$ is a relation over attributes (*price, quantity*) with tuples {(5.00, 10), (25.00, 3)}.

In the case that attribute names are not present in the relation schema, the projection is specified by an expression of the form $\pi_{i_1,...,i_n}(R)$. Here $i_1,...,i_n$ is a sequence of positive integers – where each i_j is bounded by the arity of R – identifying attributes of R. The output is a relation of arity n with tuples $(t(i_1),..., t(i_n))$, for each tuple t in R. In this case the arity n of the output relation can be possibly larger than the arity of R, since integers $i_1,...,i_n$ need not be distinct.

Cross-references

- ▶ Relation
- ▶ Relational Algebra

Projection Index

▶ Bitmap-based Index Structures for Multidimensional Data

Propagation-based Structured Text Retrieval

Karen Pinel-Sauvagnat
IRIT-SIG, Toulouse Cedex, France

Synonyms

Relevance propagation; Score propagation

Definition

Evaluate the relevance score of text components. Approaches using propagation view the logical structure of a structured document as a tree whose nodes are components of the document and whose edges represent the relationships between the connected nodes. A document component can be either a leaf or an inner node. Leaf nodes are document components that correspond to the last elements of hierarchical relationship chains and that contain raw data (textual information). With propagation, relevance scores are first calculated for leaf components. They are then propagated upwards in the document tree structure to calculate relevance scores for the inner components.

Historical Background

With appropriate query languages, users may want to exploit the structure of structured text documents to perform fine-grained and flexible retrieval. Instead of treating documents as atomic units, structured text retrieval systems aim thus at retrieving document components that answer a given information need in the most *specific* way. Classical information retrieval (IR) models (e.g., the vector space model, the probabilistic model, language models) have been extended in order to take into account the structural dimension of documents. IR research has shown that document term weighting is a crucial concept for effective retrieval, thus classical formalisms were adapted with additional weighting parameters: component type, number of descendant components, frequency of the component type, to name a few. However, most of these adaptations do not use (or make little use of) the tree representation of structured documents to identify the most specific components.

The idea behind relevance propagation method is to follow the way *relevance* changes in a document tree to estimate the relevance of the document components. Indeed *relevance* in structured text documents has been expressed in terms of *specificity* and exhaustivity: the specificity (its coverage of the topic and nothing else) of components in the tree typically decreases as one moves up the tree, and when a component has multiple relevant descendants, its exhaustivity (coverage of the topic) usually increases compared to that of each of the descendant elements.

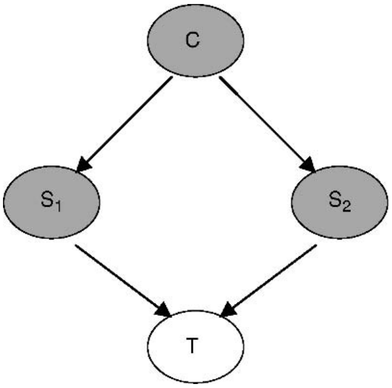
In propagation methods, relevance scores are thus first computed at leaf level, and then propagated up the document tree: to allow the identification of the most specific components, the relevance score of leaf components should be somehow decreased while being propagated upwards in the document tree, and to allow the identification of the most exhaustive components, relevance scores may be aggregated (a parent score should be evaluated using its children scores). A naive solution would be to just sum the relevance scores of each inner component relevant children. However, this would ultimately result in root components being returned at top ranks although they may not be the most specific components for the given information need.

Few relevance propagation approaches were proposed before 2002 (when *INEX*, the evaluation Initiative for XML Retrieval, was set-up). One can however cite the method presented in [5] using inference nets. The retrieval process is applied to SGML documents but can be extended to any type of structured documents. The basic retrieval strategy is to calculate the degree to which a component at any level of the document hierarchy satisfies the query by considering what components are contained in that component. This strategy makes it possible to systematically calculate the expected relevance of a component at any level, taking into account its relationship with other components in the hierarchy. Consider the following net made of two section components S_1 and S_2 that are the parent node of a term node T and the children nodes of the component C (see Fig. 1).

The retrieval process is performed in a bottom-up fashion, because of the way documents are represented in the inference net: no components other than leaf components contain actual text, and the retrieval process must start from leaf components whose text contains a query term.

The degree of belief of T given the network topology is computed with a simplified formula as follows (the simplification comes from considering only positive events):

$$\begin{aligned} B(T|C) &= P(T|S_1) \times P(S_1) + P(T|S_2) \times P(S_2) \\ &= P(T|S_1) \times P(S_1|C) \times P(C) + P(T|S_2) \\ &\quad \times P(S_1|C) \times P(C). \end{aligned} \tag{1}$$



Propagation-based Structured Text Retrieval. Figure 1. A simple network.

Children components S_i and their parent C are represented as $S_i = <s_{i1}, s_{i2},...,s_{in}>$ and $C = <c_1, c_2,...,c_n>$ respectively, where n is the number of index terms and s_{ij} and c_i are calculated using standard term frequency (TF) and inverse document frequency (IDF) statistics. The probability $P(S_i|C)$ of observing S_i given C (or the degree of belief that C supports S_i) is calculated as a similarity between the two vectors:

$$P(S_i|C) = \lambda_i \times (S_i \cdot C), \tag{2}$$

where λ_i is a weight associated to the component type, representing its overall importance relative to other types of components sharing the same parent. This computation incorporates both the content and the type of components.

The probability $P(T|S_i)$ of observing a term T given a component S_i (or the degree of belief that S_i supports T) is estimated with:

$$P(T|S_i) \cong IDF_T \times TF_{i,S_i}. \tag{3}$$

$P(C)$ is the probability of observing C assuming that C is the root node (i.e., document) in the inference net. In the implementation, it is set to 1. This approach was however not evaluated, since no suitable test collection was available.

Other approaches presented in the rest of the entry were developed and evaluated in the context of the *INEX* evaluation campaign, which is concerned with the evaluation of XML retrieval. In this case, document components correspond to XML elements.

Foundations

As all IR approaches, propagation-based approaches can be caracterized in terms of indexing and scoring methods (here both for leaf and inner elements).

Indexing

Approaches using relevance propagation consider *indexing units* as disjoint units: the text of each element is the union of one or more of its disjoint parts. Thus, as textual information is only present in leaf elements, inverted index only concern leaf elements. Propagation accounts then for the fact that the text in a given leaf element is also contained in its ancestors. The document structure is generally stored in a separate index and used to build the document trees.

Relevance Scores Evaluation for Content-Only Queries

In the rest of the entry, the following notations are used. Let q be a query composed of k terms $t_{1,}...t_k$. In the document tree, *le* is a leaf element and e an inner element having n children. *RSV*(q, *le*) is the relevance score of the leaf element *le* with respect to query q and *RSV*(q, e) is the final score of element e with respect to query q.

Scoring Leaf Elements To evaluate leaf element scores (*RSV*(q, *le*)), approaches found in the literature have used the following parameters:

1. Frequency of term t_i in query q or leaf element *le*
2. Frequency of term t_i in the whole collection
3. Number of leaf elements containing t_i
4. Length of leaf element
5. Average length of leaf elements
6. Inverse element frequency *ief*, which is similar to *idf* (inverse document frequency) but that takes into account the collection of leaf elements instead of the collection of documents.

In [1], the weighting scheme used to compute the relevance score of leaf elements also uses the cross-structural importance of t_i relative to *le* and q, which allows to increase or decrease the importance of a term depending on its location in the query (some terms may be more important than others in a content-and-structure query). In [3], the frequency of terms in the leaf elements and in the whole collection are used with a parameter scaling up the score of elements having multiple query terms. The formula penalizes elements with frequently occurring query terms (frequent in the collection), and rewards elements with more unique query terms within a result element. One can also cite the leaf elements weighting scheme presented in [9], where term frequency and inverse element frequency *ief* are applied together with *idf*. This allows to take into account the importance of terms in both the collection of leaf elements and the collection of documents.

Propagating Relevance Scores Once the relevance score of the leaf elements have been calculated, they are used to evaluate the relevance score of the inner elements. The main issue here is how to combine these scores. As already said, a naive solution that simply sums the relevance sores of leaf elements will result in root elements being ranked at the top of the result lists, although they are likely to not constitute the most specific elements to the query.

In [1] the relevance of inner elements is for instance evaluated using the maximum of their leaf element scores. Other approaches use a weighted sum of leaf or children element scores, and make some assumptions related to the document tree structure. They are described below.

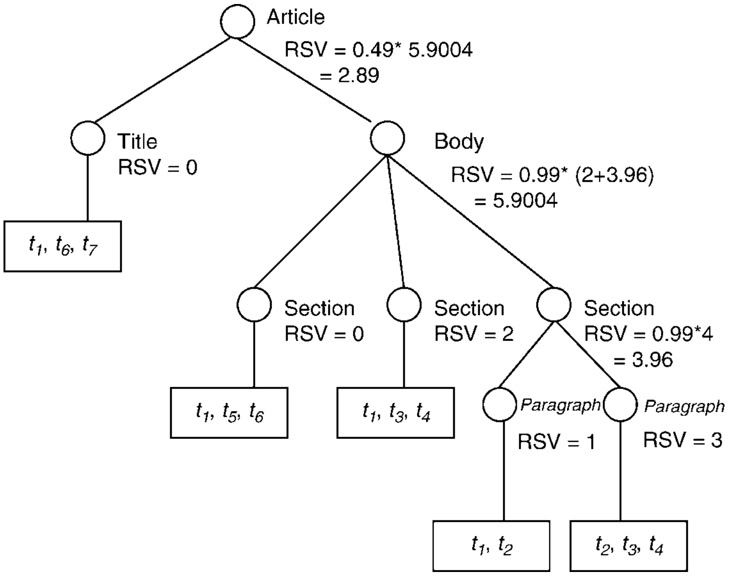
For example, in the GPX approach [3], a heuristically derived formula is proposed to evaluate the scores of inner elements that accounts for specificity and exhaustivity:

$$\mathrm{RSV}(q,e) = D(n)\sum_{l=1}^{n}\mathrm{RSV}(q,l), \tag{4}$$

where n is the number of children elements, $D(n)$ = 0.49 if n = 1, 0.99 otherwise, and *RSV*(q, l) is the relevance score of the lth child element.

The value of the decay factor D depends on the *number of relevant children* that the inner element has. If the element has one relevant child then the decay constant is 0.49. An element with only one relevant child will be ranked lower than its child. If the element has multiple relevant children the decay factor is 0.99. An element with many relevant children will be ranked higher than its descendants. Thus, a section with a single relevant paragraph would be judged less relevant than the paragraph itself, but a section with several relevant paragraphs will be ranked higher than any of the paragraphs.

"t_2 t_3 t_4". The method is illustrated in Fig. 2, with the content-only query "t_2 t_3 t_4". To simplify, the weight of a term in a leaf element is equal to 1 if it is a query term and 0 otherwise. For example the /article[1]/body[1]/section[3]/paragraph[2] element that contains three query terms has thus a score of 3. Its parent (/article[1]/body[1]/section[3]) contains two relevant children with scores 1 and 3. Its score is then calculated with a decay factor equal to 0.99 as follows: 0.99 × (1 + 3). To evaluate the score of the root element (which only contains one relevant child), the decay factor used is 0.49, and the root element has consequently a lower score than its child /article[1]/body[1]. As a result of the propagation, the /article[1]/body[1] element has the highest score and will be ranked first by the GPX system.



Propagation-based Structured Text Retrieval. Figure 2. Relevance propagation according to [3].

The *distance between an element and its descendant leaf elements* has also been used as a weight in the weighted sum that calculates the relevance score of inner elements. Terms that occur close to the root of a given subtree are more significant to the root element that ones at deeper levels of the subtree. It seems therefore that the greater the distance of a element from its ancestor, the less it should contribute to the relevance of its ancestor. This can be modeled with the $d(x, y)$ parameter, which is the distance between elements x and y in the document tree, i.e., the number of arcs joining x and y.

In [4], the relevance score of an inner element takes into account this distance and also the distance separating the root element and leaf elements. The latter is used as a normalization factor. The relevance score of an inner element e is calculated as follows:

$$\mathrm{RSV}(q,e) = \sum_{le_j \in L_e} \left(1 - 2\lambda\lambda \frac{d(e, le_j)}{d(e, le_j) + d(root, le_j)}^{2}\right) \mathrm{RSV}(q, le_j). \tag{5}$$

λ is a constant coefficient ≥ 0, le_j are leaf elements being descendant of e, and L_e is the set of leaf elements being descendant of e. This process tends to consider an element having a relevant descendant element less relevant than the descendant element itself, which is comparable to the approach followed in [3], as above described.

Finally, in [9], the distance between elements and the number of relevant descendant leaf elements are used together with an additional parameter, β, in the weighted sum. The β factor captures the assumption that small elements may be used by authors to highlight important information (*title* elements, *bold* elements, ...). *Small elements can therefore give important indications on the relevance of their ancestors* and their importance should be increased during propagation. The relevance value of an element e is thus computed according to the following formula:

$$\mathrm{RSV}(q,e) = |L_e^r| . \sum_{le_j \in L_e} \alpha^{d(e, le_j) - 1} \times \beta(le_j) \times \mathrm{RSV}(q, le_j), \tag{6}$$

where α is a constant coefficient $\in]0,1]$ used to tune the importance of the distance $d(e, le_j)$ parameter and $|L_e{}^r|$ is the number of leaf elements being descendant of e and having a non-zero relevance value. $\beta(le_j)$ is experimentally fixed and allows to increase the role of elements smaller than the average leaf element size in the propagation function.

Sauvagnat et al. (2005) [9] also propose a backward propagation after the first propagation, in order to account for the whole document relevance (and thus for the element context) in the calculation of the relevance score of inner elements.

Content-and-Structure Queries Processing

In most of the approaches using relevance propagation, results elements are simply filtered to satisfy the structural constraints of *content-and-structure queries* [3,4]. The approach described in [8] however uses relevance propagation to process structural constraints. Queries may have several constraints, and each constraint is composed of both a content and a structure condition, one of them indicating which type of elements should be returned (target elements). For each constraint, propagation starts from leaf nodes answering the content condition and goes until an element that matchs the structure constraint is found. The score of result elements of each constraint is then propagated again to elements belonging to the set of targeted structures.

Key Applications

The access to structured text documents such as SGML or XML documents is the main application of the relevance propagation approach described in this entry. Propagation can also be used to access HTML documents in the context of web retrieval, for example to determine among a set of related web pages, which one corresponds to the best entry in the set. Moreover, propagation can also be applied to distributed IR, as presented in [2].

Future Directions

Propagation methods presented in this entry are relatively independent of the DTDs of collections, since most of them do not use the type of elements to estimate relevance. However, propagation cannot be done in the same way on small document collections and large document collections. Methods should be adapted to process large document collections and efficiency issues that follow.

Relevance propagation may evolve in the future with the use of another source of evidence to calculate inner element relevance score, for example, link information. Indeed, web approaches using relevance propagation have also used links to find relevant web pages (a relevant page may be linked to other relevant pages) [7]. The same approaches could be used for structured text retrieval.

Experimental Results

INEX evaluation campaign. For example, the approach in [3] was ranked in the top 5 for the focused retrieval task (which aims at targeting the appropriate level of granularity of relevant content that should be returned to the user for a given topic) and comparable results were achieved by the approach described in [8] using content-and-structure queries.

Cross-references

- ▶ Aggregation-Based Structured Text Retrieval
- ▶ INEX
- ▶ On Enlve
- ▶ Relevance
- ▶ Specificity
- ▶ Term Statistics for Structured Text Retrieval
- ▶ XML Retrieval

Recommended Reading

1. Anh V.N. and Moffat A. Compression and an IR approach to XML Retrieval. In Proc. 1st Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2002.
2. Baumgarten C. A probabilistic model for distributed information retrieval. In Proc. 20th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1997, pp. 258–266.
3. Geva S. GPX – Gardens Point XML IR at INEX 2005. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 240–253.
4. Hubert G. XML retrieval based on direct contribution of query components. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 172–186.
5. Myaeng S.-H., Jang D.-H., Kim M.-S., and Zhoo Z.-C. A flexible model for retrieval of SGML documents. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 138–145.
6. Ogilvie P. and Callan J. Parameter estimation for a simple hierarchical generative model from XML retrieval. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 211–224.
7. Qin T., Liu T.-Y., Zhang X.-D., Chen Z., and Ma W.-Y. A study of relevance propagation for web search. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 408–415.
8. Sauvagnat K., Boughanem M., and Chrisment C. Answering content-and-structure-based queries on XML documents using relevance propagation. Inf. Syst., 31:621–635, 2006.
9. Sauvagnat K., Hlaoua L., and Boughanem M. XFIRM at INEX 2005: adhoc and relevance feedback tracks. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 88–103.

Protein Sequence

▶ Biological Sequence

Protein-Protein Interaction Networks

▶ Biological Networks

Provenance

Wang-Chiew Tan
University of California-Santa Cruz, Santa Cruz,
CA, USA

Synonyms

Lineage; Origin; Source; History; Pedigree

Definition

Let t be a data element in the result of a query Q applied to a dataset D. A data element may be a tuple in the relational model, or a subtree in the semi-structured model. The *provenance* of t is the set of all proofs for t according to Q and D. A proof for t according to Q and D is a set D' of data elements in D so that t is in the result of applying Q on D'. In some cases, a proof also details the process by which t is derived from Q and D'.

Most work on provenance in databases focused on finding data elements of D that witness the existence of t in the result, as well as which data elements of D is t copied from. In scientific workflows, the provenance of a result is typically a detailed description of the entire workflow used to arrive at the result.

Key Points

The need to understand and manage the provenance of data arises in almost every application. For example, it is natural to ask for the provenance of data seen on the Web. In scientific experiments, the provenance of results generated by the experiments constitutes to the proof of correctness directly or indirectly, and is typically regarded to be as important as the result itself.

There are generally two types of provenance: data provenance and workflow provenance [1,3].

Data provenance is an account of the derivation of a piece of data in a dataset that is typically the result of a database operation on an input database. There are two approaches for computing data provenance [2]: annotation-based versus non annotation-based approaches. In annotation-based approaches, provenance is captured by propagating annotations (i.e., extra information) from input to output along database transformations. Hence, the provenance of an output data can typically be determined by analyzing the associated annotations. In contrast, non annotation-based approaches do not carry along extra information. Instead, the provenance of an output data is in the determined by analyzing the database transformation, as well as the input and output databases.

Workflow provenance refers to the record of the history of the derivation of some dataset in a scientific workflow. The amount of information recorded for workflow provenance varies, depending on application needs. For example, workflow provenance of a scientific result may include details about the type and model of external devices used, as well as the versions of softwares used for deriving the result.

Cross-references

▶ Data Provenance
▶ Provenance
▶ Provenance in Experimental Data Management
▶ Provenance in Scientific Databases
▶ Storage Security

Recommended Reading

1. Bose R. and Frew J. Lineage retrieval for scientific data processing: a survey. ACM Comput. Surv., 37(1):1–28, 2005.
2. Buneman P. and Tan W.-C. Provenance in databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1171–1173.
3. Simmhan Y., Plale B., and Gannon D. A survey of data provenance in E-Science. ACM SIGMOD Rec., 34:31–36, 2005.

Provenance Metadata

▶ Data Provenance

Provenance in Scientific Databases

Sarah Cohen-Boulakia[1], Wang-Chiew Tan[2]
[1]University of Pennsylvania, Philadelphia, PA, USA
[2]University of California-Santa Cruz, Santa Cruz,
CA, USA

Synonyms

Lineage; Origin; Source; History; Pedigree

Definition

Scientific databases contain data which may have been produced as answer to a query posed over other resources, or generated by in-silico experiments (or scientific workflow) involving various softwares, or manually curated by domain experts based on analysis of several other resources. The provenance of a piece of data in scientific databases typically includes information of where this piece of data originates from, as well as details of the scientific process (e.g., parameters used in the experiments, software versions etc.) by which it arrived in the scientific database.

Historical Background

Provenance of scientific databases has been studied in two granularities: *workflow provenance* and *data provenance*.

Workflow provenance (or *coarse-grained provenance*) refers to the record of the history (or workflow) of the derivation of some dataset in a scientific workflow [5,13,14]. The amount of information recorded for workflow provenance varies, depending on application needs. For example, in some cases, a workflow provenance may record the type and model of external devices used, such as sensors, cameras, or other data collecting equipments, as well as the associated versions of software used for processing data; in other cases, these information may be deemed unnecessary.

More recently, workflow systems developed within the scientific community to conduct and manage experiments have started recording information about the processes used to derive intermediate and final data objects from raw data. Survey papers dedicated to workflow provenance approaches have been proposed [5,14] and "provenance challenges" [13] have been held to encourage system designers to learn about the capabilities and expressiveness of each others' systems and work towards interoperable solutions.

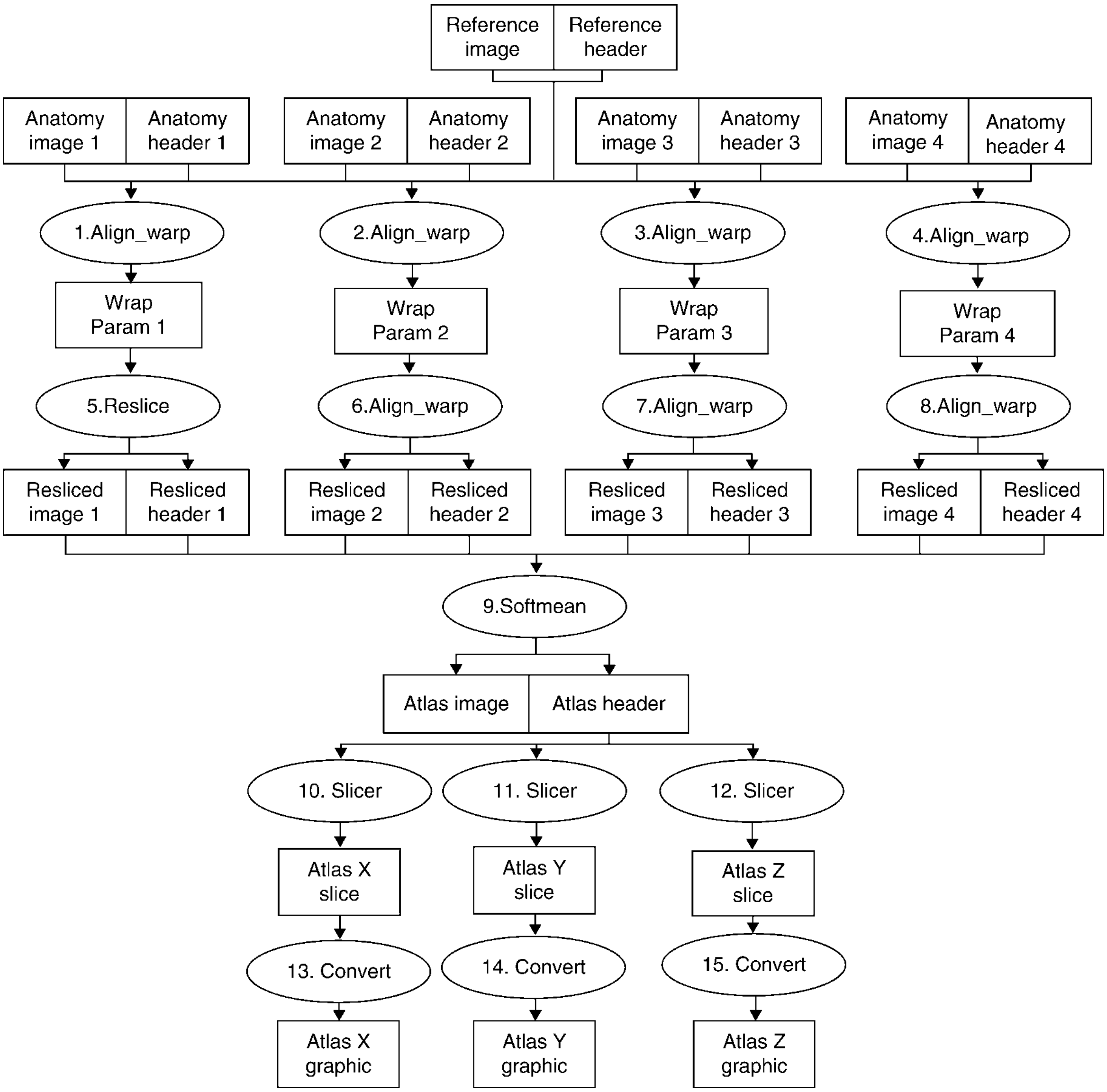
Data provenance (or *fine-grained provenance*) is an account of the derivation of a piece of data in a dataset that is typically the result of executing a database query against a source database [9]. This type of provenance determines the parts of source data that were used to generate a piece of data in the resulting dataset. Typically, data provenance is obtained by carefully reasoning about the algebraic form of the database query and the underlying data model of the source and resulting databases. In contrast, transformations occurring in scientific workflows are often external processes (e.g., perl scripts), and the log files of scientific workflows provide only object identifiers to pieces of data involved in the transformation at best. Often, such external processes and data involved do not possess good properties for detailed analysis. Hence, a fine-grained analysis of an external process is not always possible and the provenance recorded for such transformation is usually more coarse-grained.

Foundations

Workflow Provenance: The notion of workflow provenance is illustrated using the workflow of the first provenance challenge, which aimed to establish an understanding of the capabilities of available provenance-related systems. This simple workflow is inspired from a real experiment in the area of functional Magnetic Resonance Imaging (fMRI) and formed the basis of the challenge. It is represented in Fig. 1 as a graph where oval nodes represent steps and rectangle nodes represent kind of data exchanged between them. Typical provenance queries ask for the history of some data item, e.g., "What caused Atlas X Graphic (one final output) to be as it is?". Depending on whether the complete history of the data is asked or only the previous step and its inputs, queries are qualified as *deep* (or *recursive*) or *immediate*. For example, the immediate provenance of *Atlas X Graphic* is given by the step *13.convert* and its input, *Atlas X slice*, while its deep provenance is given by all the data and steps used to compute the step *9.softmean* and *Atlas Image, Atlas Header, 10.slicer, Atlas X slice*, and *13.convert*. Other provenance queries may include *annotation queries* finding data annotated with some specific metadata, such as "Find all invocations of procedure align_warp using a twelfth order nonlinear 1365 parameter model?" (metadata on the procedure used should be recorded) and "Find the outputs of align_warp where the inputs are annotated with center = UChicago" (metadata on the data used should be recorded).

Various workflow provenance approaches have been designed and applied to very diverse scientific domains (e.g., biology, ecology, astronomy, meteorology). As workflow provenance is by nature associated with actual scientific experiments, most of these approaches have been implemented into prototypes or systems that have typically participated in such challenges [13].

Approaches differ in various aspects. In particular, provenance can be tracked at various degrees of granularity: from OS or instrument level to input/output data and metadata associated to them. The graphs formed by the workflow provenance information are



Provenance in Scientific Databases. Figure 1. Workflow of the first provenance challenge.

also represented using various data models, from relational to tree-based data models (e.g., XML, such as in the Kepler and ES3 projects) or graph-based data models (e.g., Web semantic approaches based on RDF, such as in the myGrid or Wings/Pegasus projects). As a consequence, a plethora of languages have been used to query workflow provenance information including SQL, XQuery, and SPARQL while several dedicated graph query languages have been designed.

More precisely, Foster et al. [13] tackle with the problem of reproducing experiments by providing a *virtual data system* approach offering the ability to not only track the data consumed and produced during computations but also rederive deleted intermediate results of an experiment (virtual data).

Callahan et al. [13] focus on the problem of *workflow evolution*: *VisTrails* provides techniques to compute workflow differences and similarities.

In an orthogonal way, Miles et al. define an *open architecture for provenance* [13] designed to be domain and technology independent. In this architecture, *process documentations* describe what actually occurred at execution time and are created and stored by provenance-aware applications.

Finally, Biton et al. provide abstraction mechanisms to help users face the overwhelming amount of provenance information in workflow environments, offering the possibility of focusing on the most relevant provenance information. The technique pursued in *ZOOM*UserViews* is that of “user views” [3]: Since bioinformatics tasks may themselves be complex sub-workflows, a user view determines what level of sub-workflow the user can see, and thus what data and steps are visible in provenance queries. Algorithms to compute relevant user views have been proposed in [4].

Data Provenance: Here, biological databases are used as example scientific databases for illustrating data provenance. The relations and SQL query are shown below.

```
SELECT swissprot, pir, s.desc
FROM SWISS-PROT s, PIR p, Mapping
      Table m
WHERE s.id = m.swissprot AND p.id = m.
       pir
```

The result of executing the SQL query consists of two tuples (a231, p445, AB) and (w872, p267, CD). Work on data provenance [7,11] has provided explanations for why a tuple, such as (a231, p445, AB), is in the query result. The reason is because the first tuples in SWISS-PROT, PIR and Mapping-Table respectively, joined according to the query to produce the output tuple. This type of provenance is termed *why-provenance* in [7]. In contrast, the *where-provenance* of “AB” in the output tuple (i.e., where “AB” is copied from) is the desc attribute of the first tuple in SWISS-PROT. In particular, no other source tuples are involved in the explanation of where-provenance. Subsequent work by Green et al. [12] is also able to explain *how* a tuple is derived in the result of a query.

Another line of work captures provenance by propagating annotations of source data to the output, based on provenance, along query transformations. Wang and Madnick [15] first articulated the idea of using propagated annotations to analyze source attribution. Subsequently, Buneman et al. [8] studied the *annotation placement problem* using a variation of the propagation scheme of [15]. The propagation scheme of [8,15] is essentially based on where data is copied from. This scheme forms the *default* propagation scheme of DBNotes [2]. A serious drawback of the default scheme is that two equivalent SQL queries (select-project-join-union queries) may not propagate annotations in the same way. In DBNotes, the authors proposed an alternative *default-all* scheme to overcome this limitation. In the default-all propagation scheme, equivalent queries always propagate annotations in the same way. Additionally, DBNotes also support the *custom* propagation scheme, where one is allowed to customize a propagation scheme as desired. So far, all systems [2,8,15] only allow annotations on attribute values. Subsequent research efforts have proposed techniques to relax this restriction.

SWISS-PROT

id	desc
a231	AB
w872	CD
u812	DD

PIR

id	desc
p445	AB
p267	CD
p547	ED

Mapping-Table

mid	swissprot	pir
1	a231	p445
2	w872	p267

Most database and workflow techniques provide little help for managing provenance in curated databases. This is because curated databases are, by definition, not constructed from the result of executing a query but rather, created manually by scientists through the analysis of information from several sources. In [6], the authors proposed a copy-and-paste model that captures a scientist’s manual curation efforts as provenance-aware transactions. These transactions are logged and various hierarchical compression techniques were proposed and validated experimentally.

Key Applications

There are many application domains that can benefit from a provenance system in science. Several uses of provenance information are considered here, as described in the literature [5,13,14].

Informational: Interpreting and understanding the result of a scientific experiment necessitates to know the provenance or context in which the data has been produced.

Data Quality: By providing the source data and transformations, provenance may help to estimate data quality and data reliability.

Error Detection: Provenance can be used to detect errors in data generation.

Re-use: Reproducing a local experiment or an experiment described in a research paper may be possible if precise provenance information is provided.

Attribution: The copyright and ownership of data can be determined using provenance information. It enables its citation, and may determine liability in case of erroneous data.

Probabilistic Databases: Provenance has been used to compute the confidences in the result of a probabilistic query correctly, as well as to correctly reason about the set of possible instances in the result of the query [1].

Data Sharing and Data Integration: Provenance has also been used to describe trust policies in the data sharing system Orchestra and to prioritize updates, as well as to understand and debug a data exchange or data integration specification [10].

URL to Code

Chimera Project: http://www.ci.uchicago.edu/wiki/bin/view/VDS/VDSWeb/WebMain

DBNotes Project: http://www.soe.ucsc.edu/∼wctan/Projects/dbnotes

ES3 Project: http://eil.bren.ucsb.edu

GriPhyN Virtual Data System Project: http://www.ci.uchicago.edu/wiki/bin/view/VDS/VDSWeb/WebMain

Kepler Project: http://kepler-project.org

myGrid Project: http://www.mygrid.org.uk

Orchestra Project: http://www.cis.upenn.edu/∼zives/orchestra

Pasoa Project: http://users.ecs.soton.ac.uk/lavm/projects/pasoa.html

Provenance Challenges: http://twiki.ipaw.info/bin/view/Challenge

SPIDER Project: http://www.soe.ucsc.edu/wctan/Projects/debugger/index.html

VisTrails Project: http://vistrails.sci.utah.edu/index.php/Main_Page

Wings/Pegasus Project: http://www.isi.edu/ikcap/wings

ZOOM*UserViews Project: http://zoomuserviews.db.cis.upenn.edu

Cross-references

- ▶ Data Provenance
- ▶ Provenance
- ▶ Provenance in Experimental Data Management
- ▶ Scientific Databases
- ▶ Scientific workflows

Recommended Reading

1. Benjelloun O., Sarma A.D., Halevy A.Y., and Widom J. ULDBs: databases with uncertainty and lineage. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 953–964.
2. Bhagwat D., Chiticariu L., Tan W.C., and Vijayvargiya G. An annotation management system for relational databases. VLDB J., 14(4):373–396, 2005.
3. Biton O., Cohen-Boulakia S., and Davidson S. Zoom* User-Views: querying relevant provenance in workflow systems. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 1366–1369.
4. Biton O., Cohen-Boulakia S., Davidson S., and Hara C.S. Querying and managing provenance through user views in scientific workflows. In Proc. 24th Int. Conf. on Data Engineering, 2008.
5. Bose R. and Frew J. Lineage retrieval for scientific data processing: a survey. ACM Comput. Surv., 37(1):1–28, 2005.
6. Buneman P., Chapman A., and Cheney J. Provenance management in curated databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 539–550.
7. Buneman P., Khanna S., and Tan W.C. Why and where: a characterization of data provenance. In Proc. 8th Int. Conf. on Database Theory, 2001, pp. 316–330.
8. Buneman P., Khanna S., and Tan W.C. On propagation of deletions and annotations through views. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 150–158.
9. Buneman P. and Tan W.C. Provenance in databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1171–1173.
10. Chiticariu L. and Tan W.C. Debugging schema mappings with routes. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 79–90.
11. Cui Y., Widom J., and Wiener J.L. Tracing the lineage of view data in a warehousing environment. ACM Trans. Database Syst., 25(2):179–227, 2000.
12. Green T.J., Karvounarakis G., and Tannen V. Provenance semirings. In Proc. 26th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2007, pp. 31–40.
13. Moreau L., Ludäscher B., Altintas I., Barga R.S., Bowers S., Callahan S., Chin G., Clifford B., Cohen S., Cohen-Boulakia S.,

Davidson S., Deelman E., Digiampietri L., Foster I., Freire J., Frew J., Futrelle J., Gibson T., Gil Y., Goble C., Golbeck J., Groth P., Holland D.A., Jiang S., Kim J., Koop D., Krenek A., McPhillips T., Mehta G., Miles S., Metzger D., Munroe S., Myers J., Plale B., Podhorszki N., Ratnakar V., Santos E., Scheidegger C., Schuchardt K., Seltzer M., Simmhan Y.L., Silva C., Slaughter P., Stephan E., Stevens R., Turi D., Vo H., Wilde M., Zhao J., and Zhao Y. The first provenance challenge. Concurrency Comput. Pract. Exp., 20(5):409–418, 2007, Special issue on the First Provenance Challenge.

14. Simmhan Y., Plale B., and Gannon D. A survey of data provenance in e-science. ACM SIGMOD Rec., 34:31–36, 2005.
15. Wang Y.R. and Madnick S.E. A polygen model for heterogeneous database systems: the source tagging perspective. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 519–538.

Proximity

▶ Similarity and Ranking Operations

PRP

▶ Probability Ranking Principle

p-Sensitive k-Anonymity

▶ k-Anonymity

Pseudonymity

Simone Fischer-Hübner
Karlstad University, Karlstad, Sweden

Synonyms

Nymity

Definition

The term pseudonymous originates from the Greek word "*pseudonymos*" meaning "having a false name."

A pseudonym is an identifier of a subject other than one of the subject's real names, and pseudonymity is the use of pseudonyms as identifiers. Sender pseudonymity is defined as the sender being pseudonymous, recipient pseudonymity as the recipient being pseudonymous [2].

Key Points

Pseudonymity resembles anonymity as both concepts aim at protecting the real identity of a subject. The use of pseudonyms, however, allows one to maintain a reference to the subject's real identity, e.g., for accountability purposes [1]. A trusted third party, adhering to agreed rules, could for instance reveal the real identities of misbehaving pseudonymous users. Pseudonymity also allows a user to link certain actions under one pseudonym. For instance, a user could re-use the same pseudonym for purchasing items at a certain online shop for building up a reputation or for collecting bonus points under this so-called relationship pseudonym.

The degree of anonymity protection provided by pseudonyms depends on the amount of personal data of the pseudonym holder that can be linked to the pseudonym, and on how often the pseudonym is used in various contexts/for various transactions. The best protection can be achieved if for each transaction a new so-called transaction pseudonym is used that is unlinkable to any other transaction pseudonyms and at least initially unlinkable to any other personal data items of its holder (see also [2]).

Cross-references

▶ Anonymity
▶ Privacy
▶ Privacy-enhancing Technologies

Recommended Reading

1. Common Criteria Project, Common Criteria for Information Technology Security Evaluation, Version 3.1, Part 2: Security Functional Requirements, September 2006, www.commoncriteriaportal.org
2. Pfitzmann A. and Hansen M. Anonymity, unlinkability, unobservability, pseudonymity, and identity management – a consolidated proposal for terminology. Version 0.29, http://dud.inf.tu-dresden.de/Anon_Terminology.shtml, July, 2007.

Public-Key Encryption

▶ Asymmetric Encryption

Publish/Subscribe

Hans-Arno Jacobsen
University of Toronto, Toronto, ON, Canada

Definition

Publish/Subscribe is an interaction pattern that characterizes the exchange of messages between publishing and subscribing clients. Subscribers express interest in receiving messages and publishers simply publish messages without specifying the recipients for a message. The publish/subscribe message exchange is decoupled and anonymous. That is, publishers neither know subscribers' identities nor whether any subscribers with matching interests exist at all. This supports a many-to-many style of communication, where data sources publish and data sinks subscribe. Different classes of publish/subscribe approaches have crystallized. Their main differences lie in the way subscribers express interest in messages, in the structure and format of messages, in the architecture of the system, and in the degrees of decoupling supported. Publish/Subscribe is widely used as middleware abstraction, applied to enterprise application integration, system and network monitoring, and selective information dissemination.

Historical Background

The definition in this entry aims to be general and characterizes publish/subscribe as an *interaction pattern* that governs the interaction between *many* publishing data sources and *many* subscribing data sinks. However, in practice, publish/subscribe is often interpreted to mean many slightly different concepts, such as an asynchronous communication style, a messaging paradigm, a message routing approach, an event filtering (matching) approach, or a design pattern. Moreover, research on publish/subscribe has been conducted in different communities, such as distributed systems, networking, programming languages, software engineering, and databases. The exact origins of publish/subscribe are therefore difficult to pinpoint exactly. Today, publish/subscribe-style abstractions can be found in many messaging standards, messaging products, databases, and even in special purpose hardware solutions.

The interpretation of publish/subscribe as asynchronous communication style emphasizes the data dissemination aspect and the associated qualities of service. The primary concern is the distribution of data from *many* sources to *many* sinks. Sometimes only a single source is considered and publish/subscribe is viewed as a *one-to-many* data dissemination paradigm, analogous to multicast. The channel-based publish/subscribe model best represents this interpretation. The indirect ancestry of publish/subscribe in this context are early reliable broadcast protocols [3] and primitives for process group management in operating systems [4], which inspired work on group communication as abstraction for reliable many-to-many communication, surveyed in [5]. It is this context that resulted in the subject-based publish/subscribe approach (a.k.a. topic-based model), first articulated by Oki et al. [16] and popularized by TIBCO with its TIBCO/RV product.

The interpretation of publish/subscribe as mesaging paradigm emphasizes the asynchronously decoupled nature of publishing data sources and subscribing data sinks. This view of publish/subscribe results from the inclusion of publish/subscribe functionality into standard messaging system specifications and products, such as MQ Series from IBM or the Java Message Service specification [9]. While these abstractions focus more on asynchronous one-to-one communication realized through message queues, they also include publication, subscription and filtering capabilities, often realized as subscriber-side filtering.

The interpretation of publish/subscribe as event filtering and matching approach emphasizes the selective filtering capabilities of the approach. Subscriptions represent filter expressions and publications represent observations about events in the environment that need to be selectively brought to the attention of subscribing entities. The ancestry of this work goes back to the formulation of the *many-to-many pattern matching problem* in the artificial intelligence domain by Forgy [7] who proposed the Rete algorithm for solving this problem. Rete is the basis of many rule-based expert systems. It efficiently evaluates observed facts provided as input against rules compiled into a memory resident graph (network), or an equivalent program [7]. This work was succeeded by approaches for efficient trigger management in database systems [2,6] and research on active databases. In software engineering, similar concepts were applied for integrating software tools resulting in approaches such as YEAST [12]. Around the same time work on specifying events for event correlation and root cause analysis appeared in network management [14].

The interpretation of publish/subscribe as design pattern is based on the *Observer design pattern*, first articulated in the Gang of Four book [8], where the Observer pattern is also synonymously referred to as Publish/Subscribe. This reference is somewhat unfortunate, as the prescribed realization of the Observer pattern in the literature violates a key property of publish/subscribe. The Observer pattern is suggested for use in expressing one-to-many dependency between objects in a system. When the state of one object (the subject) changes state, all dependent objects are notified. This is achieve by having the subject know about its dependents by maintaining a list of them and requiring dependents to register, if they are interested in the subject's state changes. This violates the anonymous communication property of publish/subscribe, which requires that publishers and subscribers do not know each other.

There are a few other approach that have appeared independently in different contexts, such as tuple spaces in the programming languages context to model concurrency in the 1980's blackboard architectures in the context of artificial intelligence to model the interaction of agents, continuous query processing in the data management context, and stream processing, also in the data management context. All these approaches resemble publish/subscribe, but also differ in fundamental ways.

Foundations

A publish/subscribe system comprises *publishers*, *subscribers*, and *publish/subscribe message broker(s)*, also referred to as *message router(s)*.

Publishers and subscribers are roles held by applications built with the publish/subscribe abstraction. That is a client of the system could be publisher as well as subscriber at the same time. Subscribers express interest by registering subscriptions with the publish/subscribe system and publishers report on events by publishing messages to the publish/subscribe system. The system evaluates publications against registered subscriptions and determines which subscriptions match for a given publication. The complexity of matching varies among different publish/subscribe models. In the content-based publish/subscribe model, matching involves the evaluation of publication message content against expressive subscription filters. In the topic-based publish/subscribe model matching involves the evaluation of message topics against path-like subscription language expressions. In the channel-based publish/subscribe model, no explicit matching takes place; subscribers select among a set of channels and listen for messages broadcast on the channel.

Publications are transient and once match are not further stored or processed. Exceptions to this treatment are state-based publish/subscribe systems and the Subject spaces model [10,11] where publications might be maintained as partial matching state and as persistent state per se, respectively.

Matching and notification are performed by the publish/subscribe message brokers. In centralized installations, there is a single broker to which subscribers and publishers connect. In distributed installations, there are multiple brokers to which subscribers and publishers connect.

Publish/Subscribe offers the following decoupling characteristics. Decoupling in space allows clients to be physically distributed. Decoupling in time allows clients to be independently available. Decoupling in location means that clients do not know each others identity. This last characteristics is also referred to as anonymous communication.

A publish/subscribe system is defined by the subscription language model, the publication data model, and the matching semantic. All these elements closely depend on each other and define the subscription, the notification, the advertisement, the publication and specify the matching of the former.

The subscription language model defines the language for expressing subscriptions. It determines the expressiveness of the publish/subscribe model. For example, in the content-based model a subscription is a Boolean function over Boolean predicates. Predicates test conditions, such as equality, binary relations, or string operators over attribute values in publications. Subscriptions are also referred to as *filters*, as they specify which publication to filter out from a flow of publications processed. Some systems distinguish among the *subscriber* and the *consumer*. The subscriber is the entity that specifies subscriptions and the *consumer* is the entity that receives notifications when certain subscriptions match. Subscriber and consumer entities must not be the same.

The publication data model defines the structure, the format, and the content type of publication messages. Publications are the messages emitted by publishers. They represent the *event* of interest about the state of the system or world in the context of the modeled application. An *event* is an asynchronous state transition of interest to subscribers. The publication is the message that conveys the occurrence of the event to any interested subscribers. In practice, the term publication and event are often used synonymously without distinguishing between the actual state transition and the message published about the event. The publication concept can be further refined by introducing *notifications*. A notification is the message sent from the publish/subscribe system to subscribers with matching subscriptions, whereas a publication is the message published by publishers. A notification must not be identical to a publication that triggered it. Systems may define a notification semantic that specified which values of a publication to forward to subscribers. Also, more refined notification semantics that apply transformations to publications before notifying subscribers are imaginable. However, many authors do not distinguish between the publication and notification concept defined above and use both terms synonymously. Some publish/subscribe approaches rely on the concept of an *advertisement*. An *advertisement* is similar to a type in programming languages or schemas in databases and specifies the kind of information a publisher will publish. It is used by publish/subscribe systems to optimize matching and routing of publications. In symmetry to the difference between subscriber and publisher, a similar difference could be made among publisher and producer, where one entity publishes, while the other entity merely advertises. However, this difference does not seem to have been explored in practice so far. Not all publish/subscribe approaches use advertisements.

The matching semantic defines the conditions under which a publication matches a subscription. For example, in content-based publish/subscribe, subscriptions are often conjuncts of Boolean predicates. That is a publication matches a subscription, if each predicate in the subscription evaluates to true given the values specified in the publication. This is a *crisp* matching semantic; it requires that the publication matches the subscription exactly by either evaluating to true or false. In contrast, an approximate matching semantic weakens the matching condition and tolerates that certain predicates do not match or only match to a certain degree. A model based on fuzzy set theory and possibility distribution is model realized in the Approximate Toronto Publish/Subscribe (A-ToPSS) project [13]. Similarly, probabilistic matching semantics that are defined by evaluating the probability that a publication matches a subscription are conceivable. Also, similarity-based semantics that measure the similarity between a publication and subscription based on some similarity measure or metric are conceivable.

Publish/Subscribe. Table 1. Comparison of models

Model	Filtering	Publication	Subscription
Channel-based	No filtering	Messages	Listening to channels
Topic-based	Topics &topic hierarchy	Messages tagged with topics	Expressions with wildcards over topics
Type-based	Type checking	Objects	–
Content-based	Message content	Messages	Content-based filters

Various publish/subscribe models have crystallized over time. These models are the channel-based, topic-based, type-based, content-based, state-based, and subject spaces. Table 1 compares these models with respect to publications, subscriptions, and filtering capabilities they support. More details about each model and additional subject spaces and state-based smodels is provided in a separate definition for each term.

Rule-based systems are intended to process *facts* against *rules* through logical inference, forward chaining or backward chaining algorithms, or by evaluating rules on events. Facts represent the state of the world and rules represent knowledge. Rule-based systems generally require the maintenance of state in the rule engine, as multiple facts processed over time may contribute to the evaluation of rules.

Key Applications

Applications of publish/subscribe include Information dissemination, information filtering, alerting and notification.

Standards that implement the publish/subscribe are the CORBA Event Service [18], the CORBA Notification Service [19], AMQP [1], JMS [9], WS Topics, WS Notifications, WS Brokered Notifications, WS Eventing, OMG's Data Dissemination Service Specification [17], OGF's INFO-D [15].

Data Sets

The publish/subscribe research community has yet to produce benchmarks and collect data sets. Initial efforts are starting to emerge [20].

Cross-references

- ▶ Channel-Based Publish/Subscribe
- ▶ Content-Based Publish/Subscribe
- ▶ Event
- ▶ Event Routing
- ▶ State-Based Publish/Subscribe
- ▶ Streams
- ▶ Subject Spaces
- ▶ Topic-Based Publish/Subscribe
- ▶ Triggers
- ▶ Type-Based Publish/Subscribe

Recommended Reading

1. AMQP Consortium. Advanced Message Queuing Protocol Specification, version 0–10 edition, 2008.
2. Chakravarthy S. and Mishra D. Snoop: an expressive event specification language for active databases. Data Knowl. Eng., 14(1):1–26,1994.
3. Chang J.M. and Maxemchuk N.F. Reliable broadcast protocols. ACM Trans. Comput. Syst., 2(3):251–273, 1984.
4. Cheriton D.R. and Zwaenepoel W. Distributed process groups in the v kernel. ACM Trans. Comput. Syst., 3(2):77–107, 1985.
5. Chockler G.V., Keidar I., and Vitenberg R. Group communication specifications: a comprehensive study. ACM Comput. Surv., 33(4):427–469, 2001.
6. Cohen D. Compiling complex database transition triggers. ACM SIGMOD Rec., 18(2):225–234, 1989.
7. Forgy C.L. Rete: A fast algorithm for the many pattern/many object pattern match problem. Artifi. Intell., 19 (1):17–37, 1982.
8. Gamma E., Helm R., Johnson R., and Vlissides J. Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., 1995.
9. Hapner M., Burridge R., and Sharma R. Java Message Service. Sun Microsystems, version 1.0.2 edition, November 9th 1999.
10. Ka Yau Leung H. Subject space: a state-persistent model for publish/subscribe systems. In Proc. Conf. of the Centre for Advanced Studies on Collaborative Research, 2002, p. 7.
11. Ka Yau Leung H. and Jacobsen H.A. Efficient matching for state-persistent publish/subscribe systems. In Proc. Conf. of the Centre for Advanced Studies on Collaborative Research, 2003, pp. 182–196.
12. Krishnamurthy B. and Rosenblum D.S. Yeast: A general purpose event-action system. IEEE Trans. Softw. Eng., 21(10):845–857, 1995.
13. Liu H. and Jacobsen H.A. Modeling uncertainties in publish/subscribe system. In Proc. 20th Int. Conf. on Data Engineering, 2004.
14. Mansouri-samani M. and Sloman M. Gem: A generalized event monitoring language for distributed systems. Distrib. Syst. Eng. J., 4:96–108,1997.
15. OGF. Information Dissemination in the Grid Environment Base Specifications, 2007.
16. Oki B., Pfluegl M., Siegel A., and Skeen D. The information bus: an architecture for extensible distributed systems. In Proc. 14th ACM Symp. on Operating System Principles, 1993, pp. 58–68.
17. OMG. Data Distribution Service for Real-time Systems, version 1.2, formal/07-01-01 edition, January 2007.
18. OMG. Event Service Specification, version 1.2, formal/04-10-02 edition, October 2004.
19. OMG. Notification Service Specification, version 1.1, formal/04-10-11 edition, October 2004.
20. The PADRES Team. Publish/subscribe data sets. http://research.msrg.utoronto.ca/Padres/DataSets, 2008.

Publish/Subscribe over Streams

Yanlei Diao[1], Michael J. Franklin[2]
[1]University of Massachusetts Amherst, MA, USA
[2]University of California-Berkeley, Berkeley, CA, USA

Definition

Publish/subscribe (pub/sub) is a many-to-many communication model that directs the flow of messages from senders to receivers based on receivers' data interests. In this model, publishers (i.e., senders) generate messages without knowing their receivers; subscribers (who are potential receivers) express their data interests, and are subsequently notified of the messages from a variety of publishers that match their interests.

Historical Background

Distributed information systems usually adopt a three-layer architecture: a presentation layer at the top, a resource management layer at the bottom, and a *middleware layer* in between that integrates disparate information systems. Traditional middleware infrastructures are tightly coupled. Publish/Subscribe [13] was proposed to overcome many problems of tight coupling:

- With respect to communication, tightly coupled systems use static point-to-point connections (e.g., remote procedure call) between senders and receivers. In particular, a sender needs to know all its receivers before sending a piece of data. Such communication does not scale to large, dynamic systems where senders and receivers join and leave frequently. Pub/sub offers loose coupling of senders and receivers by allowing them to exchange data without knowing the operational status or even the existence of each other.

- With respect to content, tight coupling can occur in remote database access. To access a database, an application needs to have precise knowledge of the database *schema* (i.e., its structure and internal data types) and is at risk of breaking when the remote database schema changes. Extensible Markup Language (XML)-based pub/sub has emerged as a solution for loose coupling at the content level. Since XML is flexible, extensible, and self-describing, it is suitable for encoding data in a generic format that senders and receivers agree upon, hence allowing them to exchange data without knowing the data representation in individual systems.

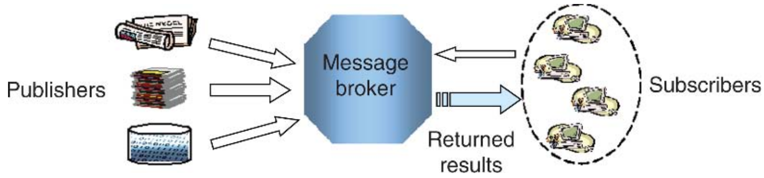
In many pub/sub systems, message brokers serve as central exchange points for data sent between systems. Figure 1 illustrates a basic context in which a broker operates. Publishers provide information by creating streams of messages (Besides "messages," the words "events," "tuples," and "documents" are often used with similar meanings in various contexts in the database literature.) that each contain a header describing application-specific information and a payload capturing the content of the message. Subscribers register their data interests with a message broker in a subscription language that the broker supports. Inside the broker, arriving subscriptions are stored as *continuous queries* that will be applied to all incoming messages. These queries remain effective until they are explicitly deleted. Incoming messages are processed on-the-fly against all stored queries. For each message, the broker determines the set of queries matched by the message. A query result is created for each matched query and delivered to its subscriber in a timely fashion.

Figure 2 shows a design space for publish/subscribe over data streams. In this diagram, pub/sub systems are first classified by the data model and the query language that these systems support. Roughly speaking, there are three main categories.

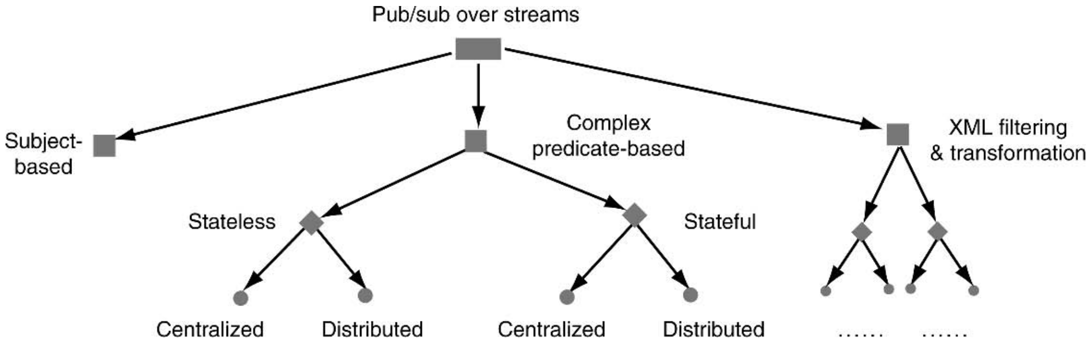
- *Subject-based*: Publishers label each message with a subject from a pre-defined set (e.g., "stock quote") or hierarchy (e.g., "sports/golf"). Users subscribe to the messages in a particular subject. These queries can also contain a filter on the data fields of the message header to refine the set of relevant messages within a particular subject.
- *Complex predicate-based*: Some pub/sub systems model the message content (payload) as a set of attribute-value pairs, and allow user queries to contain predicates connected using "and" and "or" operators to specify constraints over values of the attributes. For example, a predicate-based query applied to the stock quotes can be "Symbol = 'ABC' and (Change > 1 or Volume > 50,000)."
- *XML filtering and transformation*: Recent pub/sub systems have started to exploit the richness of XML-encoded messages, in particular, the hierarchical, flexible XML structure. User queries can be written using an existing XML query language such as XQuery. The rich XML structure and use of an XML query language enable potentially more accurate filtering of messages and further re-structuring of messages for customized result delivery.

Pub/sub systems can be further classified based on the style of query processing. In some systems, queries are applied only to individual messages, e.g., filtering messages, which does not involve any interaction across message boundaries. Such processing is referred to as *stateless*. Stateless processing is in contrast to stream query processing that maintains *state* over a long stream of messages, hence referred to as *stateful* processing. This distinction is illustrated for complex predicate-based systems in Fig. 2.

Finally, pub/sub systems can be distinguished based on the distribution of the architecture, as also shown in Fig. 2. In a coarse-grained fashion, this design space considers centralized and distributed processing. Distributed processing spreads the processing load for



Publish/Subscribe over Streams. Figure 1. Overview of publish/subscribe.



Publish/Subscribe over Streams. Figure 2. Design space for publish/subscribe over streams.

larger-scale pub/sub services; accordingly, it requires a more sophisticated routing functionality.

Foundations

As with stream processing, subscriptions, stored as continuous query inside a broker, need to be evaluated as data continuously arrives from other sources; that is, queries are evaluated every time when a new data item is received. Besides stream processing, pub/sub raises several additional challenges:

- *Scalability.* A key distinguishing requirement of pub/sub is scalability, in particular, in query population that pub/sub systems need to support. Such query populations can range from hundreds to millions in applications such as personalized content delivery. Given such populations, a salient issue is to efficiently search the huge set of queries to find those that can be matched by a message and to construct complete query results for them.
- *Robustness.* A second requirement of message brokers is the ability to perform in highly-dynamic environments where subscribers join and leave and their data interests change over time. Since message brokers see a constantly changing collection of queries, they must react quickly to query changes without adversely affecting the processing of incoming messages.
- *Distribution.* Due to the scale of message volume and query population, large-scale pub/sub may require the use of a network of message brokers to distribute the query population and message processing load. In this case, an additional issue is how to efficiently route a message from its publishing site to the set of brokers hosting relevant queries for complete query processing.

Scope of this entry. The rest of the entry focuses on complex predicate-based pub/sub systems. Pub/sub systems exploring XML filtering and transformation are described in detail in the entry "XML Publish/Subscribe."

Centralized, Stateless Publish/Subscribe

Le Subscribe [9] and *Xlyeme* [12] are predicate-based message filtering systems that use centralized processing. In these systems, a predicate is a comparison between an attribute and a constant using relational operators such as "=", ">", and "<". The main issue they address is how to efficiently match an incoming event, in the form of attribute value pairs, with the predicates of a large number of queries. The key idea is to *index* predicates as well as to *cluster* queries. In particular, *Le Subscribe* uses multi-attribute hash indexes to evaluate several predicates in a query with a single operation. In addition, it groups queries based on the number of contained predicates and the common conjunction of equality predicates, so many queries can be (partly) evaluated using a single operation. It further offers cost-based algorithms to find optimal clustering and to dynamically adjust it.

Centralized, Stateful Publish/Subscribe

NiagaraCQ [6] considers continuous queries with more complex predicates that can compare attributes of an input message to constants or to attributes of another message. To efficiently handle multiple queries, it groups query plans of continuous queries based on common *expression signatures*: an expression signature presents the same syntax structure, but possibly different constant values, in different queries. Consider queries that are interested in stock quotes of different symbols. Traditional query processing involves repeated retrieval of the symbol attribute from input and evaluation of different predicates on this attribute for different queries. The group plan employs a constant table to store the constant values form different queries, and retrieves this attribute from the input once and then performs an equality join of the retrieved value and the constant table to find all matching queries. For robust processing, NiagaraCQ constructs group plans incrementally: Given a new query, it constructs the query plan, and merges the query plan bottom up with a group plan with the same signature, extending the group plan with the mismatched branch(es) if necessary. This process is incremental as the addition of the new query plan does not affect existing queries.

Recently, there has been a significant amount of activity on handling continuous and time-varying tuple streams, resulting in the development of multiple general-purpose stream management systems [1,5,11]. These systems support complex continuous queries that join multiple streams and/or compute aggregate values over a period of time called a *window* (hence, performing stateful processing). While this surge of research explores a broad set of issues such as adaptivity and approximation, shared processing of window-based queries [5,11,10] is of particular relevance to pub/sub.

Several special-purpose pub/sub systems have been recently proposed to handle temporal correlations among events in a stream. SASE [15] supports sequencing operators that integrate parameterized predicates (i.e., predicates that compare different events), negation, and windowing. It explores a new query processing abstraction that uses an automaton-based implementation for fast sequence operations and relational-style post-processing for other tasks such as negation and windowing. It also devises a set of optimizations in this automaton-based framework for efficiency and scalability. Cayuga [7] offers an algebra for expressing event sequences that may address a finite yet unbounded number of events with a similar property, and employs a more sophisticated automaton model to support this algebra. Its implementation focuses on multi-query optimization including merging states of automata for different queries and further indexing query predicates.

Distributed, Stateless Publish/Subscribe

In distributed pub/sub systems, messages are published and subscriptions are registered to different brokers. A key issue is to efficiently route each message from its publishing site to the subset of brokers hosting relevant queries for complete query processing. For complexity reasons, most distributed pub/sub systems restrict themselves to stateless services.

ONYX [8] presents an overview of a pub/sub network exploring *content-based routing*. In this paradigm, brokers are organized as an application-level overlay network with a particular topology. When a new message enters the broker network, the root broker as well as each intermediate broker routes the message to its neighboring brokers based on the correspondence between the content of the message and the subscriptions residing at and reachable from those brokers. Content-based routing is used as a key mechanism to avoid the flooding of messages to all brokers in the network, hence reducing bandwidth usage and broker processing load and rendering better scalability.

ONYX consists of two layers of functionality. The lower layer deals with the overlay network; in particular, for each broker, it constructs a broadcast tree that is rooted at that broker and reaches all other brokers in the network. On top of these broadcast trees, the higher layer performs content-based processing by dealing with subscriptions and messages. Two issues determine the effectiveness of content-based routing. The first is how to partition subscriptions and assign them to host brokers. Results of ONYX show that content-based routing is most effective if the clustering of subscriptions results in mutual exclusiveness in data interest among host brokers. The second issue is how to aggregate subscriptions from their host brokers and place such aggregations as routing specifications in the intermediate brokers for later directing the message flow. Different degrees of generalization are possible depending on the precision-efficiency tradeoff suitable for each pub/sub system.

For content-based routing, Gryphon [2] and Siena [3] both aggregate subscriptions into compact, precise in-network data structures and use efficient algorithms to search these data structures to determine the routing of the messages to other brokers in the network.

XPORT [14] focuses on the construction, maintenance, and optimization of an overlay dissemination tree of the available message brokers in the system. Its tree-oriented optimization framework consists of a generic aggregation model that allows system cost to be expressed through various combinations of aggregation functions and local metrics, and distributed iterative optimization protocols for cost-based optimization of the overlay structure.

Distributed, Stateful Publish/Subscribe

For stateful publish/subscribe, Chandramouli et al. [4] adopt the following model: Users define subscriptions as SQL views over a conceptual (possibly distributed) database and messages are published as updates to the database; if a database update affects a subscription, the pub/sub system sends a notification to the subscriber containing the change to the content of the subscription view. The main idea of this work is to explore appropriate interfacing between the database and the pub/sub network so that existing stateless pub/sub networks can be leveraged for efficient dissemination. To do so, the key is to transform published messages into a semantic description of affected subscriptions (performed at the database side) and subscriptions into a predicate over the semantic description (evaluated in the stateless pub/sub network). Consider a selection-join subscription $\sigma_p(\sigma_{p\text{-}R}\ R \bowtie \sigma_{p\text{-}S}\ S)$. If a new message applies an update ΔR to table R, its effect on the subscription, $\sigma_p(\sigma_{p\text{-}R}\ \Delta R \bowtie \sigma_{p\text{-}S}\ S)$, requires access to table S that is not in the original update message (hence, stateful processing). The proposed solution reformulates each message ΔR into a series of messages containing the tuples in $\Delta R \bowtie S$ at the database side; to utilize a stateless pub/sub network, it transforms the select-join subscription into a simple condition that evaluates $\sigma_{p\ \wedge\ p\text{-}R\ \wedge\ p\text{-}S}$ over reformulated messages.

Key Applications

Personalized content delivery. This class of applications provide personalized filtering and delivery of news feeds, web feeds (RSS), stock tickers, sport tickers, etc. to large numbers of online users.

Online auction and online procurement. Through these applications, users can create their own feeds for their favorite searches, for example, on eBay.

Enterprise information management. Pub/sub has been traditionally used to support application integration, which integrates disparate, independently-developed applications into new services via the loose coupling of senders and receivers based on receivers' data interest.

System and network monitoring. Pub/sub has been recently used in system and network monitoring, where large-scale complex systems generate reports categorizing various aspects of system performance and resource utilization, and system administrators, end users, and visualization applications subscribe to receive updates on particular aspects of those reports.

Data Sets

Many data sources are available online, including RSS feeds (indicated by the orange button labeled with "RSS" or "XML" in many web pages) and financial feeds (e.g., Yahoo! Finance).

Cross-references

▶ Continuous Query
▶ Stream Processing
▶ XML
▶ XML Document
▶ XML Publish/Subscribe
▶ XPath/XQuery

Recommended Reading

1. Abadi D., Carney D., Cetintemel U., Cherniack M., Convey C., Lee S., Stonebraker M., Tatbul N., and Zdonik S. Aurora: a new model and architecture for data stream management. VLDB J., 12(2):120–139, 2003.
2. Aguilera M.K., Strom R.E., Sturman D.C., Astley M., and Chandra T.D. Matching events in a content-based subscription system. In Proc. ACM SIGACT-SIGOPS 18th Symp. on the Principles of Dist. Comp., 1999.
3. Carzaniga A. and Wolf A.L. Forwarding in a content-based network. In Proc. ACM Int. Conf. of the on Data Communication, 2003, pp. 163–174.
4. Chandramouli B., Xie J., and Yang J. On the database/network interface in large-scale publish/subscribe systems. In Proc. ACM SIGMOD Int. Conf on Management of Data, 2006, pp. 587–598.
5. Chandrasekaran S., Cooper O., Deshpande A., Franklin M.J., Hellerstein J.M., Hong W., Krishnamurthy S., Madden S., Raman V., Reiss F., and Shah M.A. TelegraphCQ: continuous dataflow processing for an uncertain world. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
6. Chen J., Dewitt D.J., Tian F., and Wang Y. NiagaraCQ: a scalable continuous query system for Internet databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 379–390.
7. Demers A.J., Gehrke J., Hong M., Riedewald M., and White W.M. Towards expressive publish/subscribe systems. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 627–644.
8. Diao Y., Rizvi S., and Franklin M.J. Towards an Internet-scale XML dissemination service. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 612–623.
9. Fabret F., Jacobsen H.A., Llirbat F., Pereira J., Ross K.A., and Shasha D. Filtering algorithms and implementation for very fast

publish/subscribe systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 115–126.

10. Krishnamurthy S. Shared query processing in data streaming systems. Ph.D. Dissertation, University of California, Berkeley.
11. Motwani R., Widom J., Arasu A., Babcock B., Babu S., Datar M., Manku G., Olston C., Rosenstein J., and Varma R. Query processing, resource management, and approximation in a data stream management system. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
12. Nguyen B., Abiteboul S., Cobena G., and Preda M. Monitoring XML data on the Web. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 437–448.
13. Oki B., Pfleugl M., Siegel A., and Skeen D. The information bus: an architecture for extensible distributed system. In Proc. 14th ACM Symp. on Operating System Principles, 1993, pp. 58–68.
14. Papaemmanouil O., Ahmad Y., Çetintemel U., Jannotti J., and Yildirim Y. Extensible optimization in overlay dissemination trees. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 611–622.
15. Wu E., Diao Y., and Rizvi S. High-performance complex event processing over streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 407–418.

Punctuations

David Maier[1], Peter A. Tucker[2]
[1]Portland State University, Portland, OR, USA
[2]Whitworth University, Spokane, WA, USA

Definition

A *punctuation* is an item embedded in a data stream that specifies a subset of the domain of that stream. A data item that belongs to the subset specified by a punctuation is said to *match* that punctuation. A data stream is said to be *grammatical* if, for each punctuation, no data items will follow that match that punctuation. Consider a stream of bids for online auctions. When an auction closes, a punctuation *p* is embedded in the stream that matches all bids for that auction. In a grammatical stream, *p* indicates no more bids will arrive from that stream for that auction.

In the most common format, a punctuation is a tuple of *patterns*, where each pattern corresponds to an attribute of the data items in a stream. Typically, four patterns are used: a *wildcard* (denoted by '$\ast$') is matched by all values, a *literal* is matched by only the given value, a *list* (denoted by { }) is matched by any value in the given list, and a *range* (denoted by [ ]) is matched by any value that falls in the given range. If each value in a data item matches its corresponding pattern in a punctuation, then the data item matches the punctuation. For example, if auction bids have schema <*auction_id*, *bidder_id*,*price*,*timestamp*>, given the punctuation $p = P{<}\{\ 105{,}106\},\ast,[0.00{,}100.00],\ast{>}$, all bids for auctions 105 and 106 with prices between 0 and 100 match *p*. Thus, the data item <105,95,90.00,10495> matches *p*, but the data items <104,95,90.00,10495> and <105,95,105.00,10495> do not.

Key Points

Punctuations have been shown to unblock query operators and reduce the amount of state required by query operators [2] when processing non-terminating data streams. In addition, punctuations have proven useful for dealing with disorder in streams [1] and in specifying window semantics [1,3].

A *punctuation-aware operator* is a stream query operator that can take advantage of grammatical streams. Such an operator implements three different behaviors in the presence of punctuations [2]. A *pass behavior* allows operators to output data items due to punctuations. For example, when a punctuation arrives, an aggregate can output results for a particular group if all possible data items for that group match that punctuation. A *keep behavior* specifies which data items must remain in state when a punctuation arrives. For example, when a punctuation arrives, some data items that are held in state for symmetric hash join may be released. Finally, a *propagation behavior* defines which punctuations can be output when punctuation arrives. For example, union can output any punctuation that has arrived from both inputs.

Most punctuation-aware operators are counterparts of operators on finite relations. However, there can be more than one way to define pass, keep and propagation behavior when extending a relational operator to be punctuation-aware. An important property here is for a stream operator *s* to be *faithful* to its analogous relational operator *r*, which means that on any finite prefix *x* of a stream, the output of *s* on *x* is consistent with the output of *r* on any finite extension of *x*.

Cross-references

▶ Continuous Query
▶ Data Stream
▶ Stream-Oriented Query Languages and Operators
▶ Stream Models
▶ Stream Processing
▶ Window-Based Query Processing

Recommended Reading

1. Li J., Maier D., Tufte K., Papadimos V., and Tucker P.A. Semantics and evaluation techniques for window aggregates in data streams. In ACM SIGMOD International Conference on Management of Data, 2005, pp. 311–322.
2. Tucker P.A., Maier D., Sheard T., and Fegaras L. Exploiting punctuation semantics in continuous data streams. IEEE Trans. Knowl. Data Eng., 15(3):555–568, 2003.
3. Tucker P.A., Maier D., Sheard T., and Stephens P. Using punctuation schemes to characterize strategies for querying over data streams. IEEE Trans. Knowl. Data Eng., 19(9):1227–1240, 2007.

Push Transactions

▶ Information Filtering

Push/Pull Delivery

▶ Data Broadcasting, Caching and Replication

Q

QE, Query Enhancement

► Query Expansion for Information Retrieval

QoS-Based Web Services Composition

► Process Optimization

Quadtree Variations

► Quadtrees (and Family)

Quadtrees (and Family)

MICHAEL VASSILAKOPOULOS[1],
THEODOROS TZOURAMANIS[2]
[1]University of Central Greece, Lamia, Greece
[2]University of the Aegean, Salmos, Greece

Synonyms

Hierarchical spatial indexes; Hierarchical regular-decomposition structures; Quadtree variations

Definition

In general, the term Quadtree refers to a class of representations of geometric entities (such as points, line segments, polygons, regions) in a space of two (or more) dimensions, that recursively decompose the space containing these entities into blocks until the data in each block satisfy some condition (with respect, for example, to the block size, the number of block entities, the characteristics of the block entities, etc.).

In a more restricted sense, the term Quadtree (Octree) refers to a tree data-structure in which each internal node has four (eight) children and is used for the representation of geometric entities in a two (three) dimensional space. The root of the tree represents the whole space/region. Each child of a node represents a subregion of the subregion of its parent. The subregions of the siblings constitute a partition of the parent's regions.

Several variations of quadtrees are possible, according to the dimensionality of the space represented, the criterion guiding the subdivision of space, the type of data represented, the type of memory (internal or external) used for storing the structure, the shape, position and size of the subregions, etc. However, the term Quadtree usually refers to tree structures that divide space in a hierarchical and regular (decomposing to equal parts on each level) fashion. Since final subregions/blocks (that are no further subdivided, i.e., the blocks of the tree leaves) do not overlap, the underlying space is partitioned to a set of regions, in favor of the efficient processing of spatial queries.

Historical Background

As computer science evolved, the need for representing and manipulating spatial data (data expressing geometric properties of entities, conceptually expressed by points, line segments, regions, geometric shapes, etc.) arose in several applications areas, like Computer Graphics, Multimedia, Geographical Information Systems, or VLSI Design. The recursive decomposition of space was naturally identified as a means for organizing spatial data. The term "quadtree" was used by Finkel and Bentley [3] to express an extension of the Binary Search Tree in two dimensions that was able to index points (Point Quadtree). Since then, several Quadtree variations for a multitude of spatial data types that were used for almost all sorts of spatial-data manipulations have appeared in the literature.

The term Quadtree has taken a generic meaning and is used to describe a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space. The Quadtree, a variable resolution structure, is often confused with the Pyramid [11], a multi-resolution representation consisting of a hierarchy of arrays. Quadtrees and family are space-driven methods (follow the embedding space hierarchy): the division to subregions obeys a predefined way. On the contrary, R-trees and family are data-driven methods (follow the data space hierarchy): the division to subregions depends on the data.

The Region Quadtree is the most famous such structure. As its name implies, is used for representing regions and was initially termed Q-tree by Klinger and Dyer [6]. This is a main memory tree. A secondary memory implementation, the Linear Quadtree, was proposed by Gargantini [4]. The MX and PR Quadtrees are adaptations of the Region Quadtree for representing point data [11]. The MX-CIF Quadtree is a structure able to represent a collection of rectangles [11]. The PM family of Quadtrees are used for representing polygonal maps and collections of line segments. In general, extensions to three or more dimensions of each quadtree-like structure are possible. For example, the extension of the Region Quadtree for three dimensional volume data is called Region Octree. More details about the history of these and other quadtree-like structures can be found at [11,10]. There have also been proposed Quadtree variations for storing a pictorial database, like the DI-Quadtree, for binary images and the Generic Quadtree, for grayscale and color images. These and other quadtree-based methods that have been proposed for representation and querying in image database applications are reviewed in [8]. Quadtree variations have also been proposed for evolving regional data, like Overlapping Quadtrees [15], Overlapping Linear Quadtrees, the Multiversion Linear Quadtree, the Multiversion Access Structure for Evolving Raster Images and the Time-Split Linear Quadtree [13]. The XBR tree [13] is a quadtree-like structure for indexing points, or line segments especially designed for external memory. The Skip Quadtree [2] is based on Region Quadtrees and Skip Lists. It indexes points and can be used for efficiently answering point location, approximate range, and approximate nearest neighbor queries. Quadtrees have also been used in commercial Database Management Systems [7]. Demos of several Quadtree structures can be found at [1].

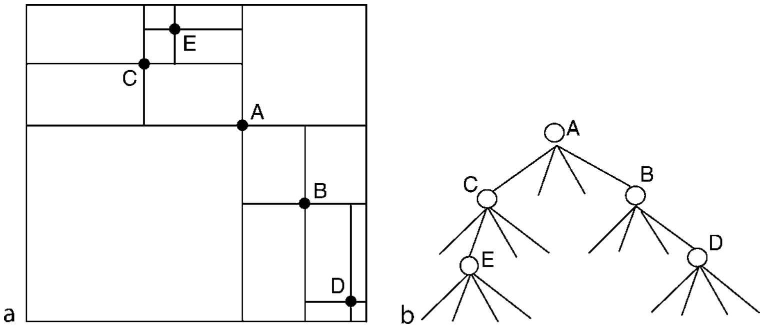
Foundations

This section briefly presents some key Quadtree structures among the numerous structures that have appeared in the literature.

Point Quadtree

The Point Quadtree [11] is an indexing mechanism for points. Each tree node corresponds to a point that subdivides the region of the node in four parts defined by two lines (parallel to the coordinate axes) on which this point lies. Thus, the shape, and position of subregions depend on the coordinates of the point (data-driven subdivision). However, each region is always subdivided in four parts. A region is considered closed in relation to its lower and left border. If multiple points with the same coordinates are allowed, each node contains a list of the points it stores. In Fig. 1, a collection of points and the resulting partitioning of space (a) and the corresponding Point Quadtree (b) are depicted. By convention, in this and other Quadtree variations, the children of a node correspond in order to the North-West, North-East, South-West and South-East subregions of the node. Note that the partitioning of space and the tree shape depend on the order of insertion of the points. Balanced versions of this tree have been proposed in the literature [11].

A Point Quadtree is not only suitable for point indexing (e.g., discovering if there is and which is the



Quadtrees (and Family). Figure 1. A collection of points (a) and the corresponding point quadtree (b).

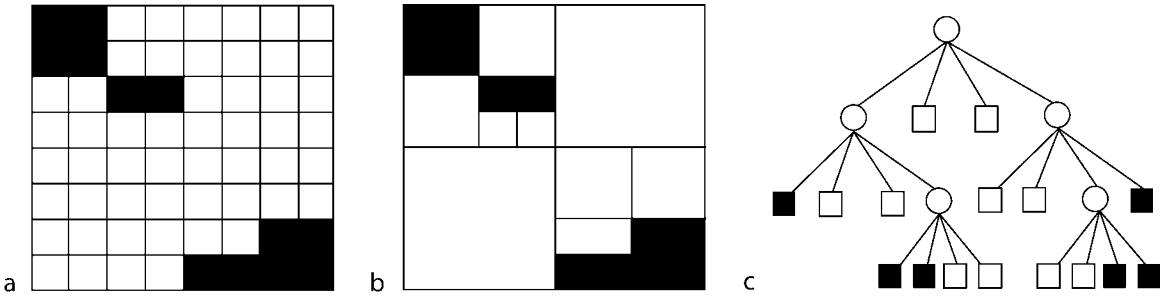
gas station that lies at a given pair of coordinates), but for answering range queries, as well (e.g., finding all the cities that reside within a specified distance from a given pair of coordinates). The efficiency of the Point Quadtree during this operation comes from pruning the search space only to nodes that may contain part of the answer.

Region Quadtree

The Region Quadtree [11] (or Q-tree [6]) is a popular hierarchical data structure for the representation of binary images, or regional data. Such an image can be represented as a $2^n \times 2^n$ binary array, for some natural number n, where an entry equal to 0 stands for a white pixel and an entry equal to 1 stands for a black pixel. The Region Quadtree for this array is made up either of a single white (black) node if every pixel of the image is white (black), or of a gray root, which points to four subquadtrees, one for every quadrant of the original image. Each region is always subdivided in four equal parts (space-driven subdivision). An example of an 8 × 8 binary image, its Region Quadtree and the unicolor blocks to which it is partitioned by the Quadtree external nodes are shown in Figs. 2a, 2b and 2c, respectively.

The Region Quadtree, depending on the distribution of data, may result in considerable space savings. However, it can be used for several operations on regional data. One of the them is the determination of the color of a specific pixel, or the determination of the block where this pixel resides (a specialized point-location query). Set theoretic operations are also well adapted to Region Quadtrees (recoloring, or dithering of a single image, or overlaying, union, intersection of sets of images). Connected component labeling (that is grouping pixels according to their color) is performed by utilizing Region Quadtrees. Other operations that are well adapted to Region Quadtrees include window clipping and certain transformations (like scaling by a power of two, or rotating by 90°). For more operations and details, see [9].

The pointer-based implementation of Region Quadtrees is a main memory implementation with one node type that consists of pointer fields and one color field. This node type is used for internal and external nodes according to the value of the color field. Although such an implementation is memory consuming it simplifies several operations. A secondary memory implementation of a Region Quadtree, called a Linear Quadtree [4], consists of a list of values where there is one value for each black node of the pointer-based Quadtree. The value of a node is an address describing the position and size of the corresponding block in the image. These addresses can be stored in an efficient structure for secondary memory (such as a B-tree or one of its variations). Evidently, this representation is very space efficient, although it is not suited to many useful algorithms that are designed for pointer-based Quadtrees. The most popular linear implementations are the FL (Fixed Length) and the FD (Fixed length – Depth) linear implementations. In the former implementation, the address of a black Quadtree node is a code-word that consists of n base 5 digits. Codes 0, 1, 2 and 3 denote directions NW, NE, SW and SE, respectively, while code 4 denotes a do-not-care direction. If the black node resides on level i, where $n >= i >= 0$, then the first n-i digits express the directions that constitute the path from the root to this node and the last i digits are all equal to 4. In the latter implementation, the address of a black Quadtree node has two parts: the first part is a code-word that consists of n base 4 digits. Codes 0, 1, 2 and 3 denote directions NW, NE, SW and SE, respectively. This code-word is formed in a similar way to the code-word of the FL-linear implementation with the



Quadtrees (and Family). Figure 2. A binary image (a), its partition to blocks (b) and its region quadtree (c).

difference that the last i digits are all equal to 0. The second part of the address has $[\log_2(n + 1)]$ bits and denotes the depth of the black node, or in other words, the number of digits of the first part that express the path to this node. Another interesting secondary memory implementation of the Region Quadtree is the Paged-pointer Quadtree [12] that partitions the tree nodes into pages and manages these pages using B-tree techniques.

PR Quadtree

The PR Quadtree [11] (P comes from point and R from region) is an indexing technique for points (with similar functionality to Point Quadtrees) that is based on the Region Quadtree. Points are associated with quadrants that are formed according to the Region Quadtree rules. A leaf node may be white (without any points residing in its region), or black (with one point residing in its region). In Fig. 3, the collection of points of Fig. 1 and the resulting partitioning of space (a) and the corresponding PR Quadtree (b) are depicted.

The final shape of the PR Quadtree is independent to the order of insertion of the points. A problem with this structure is that the maximum depth of recursive decomposition depends on the minimum distance between two points (if there are two points very close to each other, the decomposition can be very deep). This effect is reduced by allowing leaf nodes to hold up to C points. When this capacity is exceeded, the node is split in four. By storing leaf nodes on secondary memory and setting C according to the disk-page size, a structure that partially resides on disk is created.

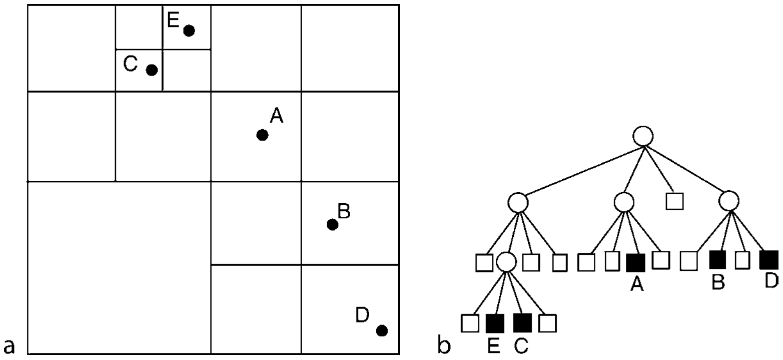
PMR Quadtree

The PMR Quadtree [11] is capable of indexing line segments and answering window queries (e.g., find the line segments that intersect a given window/area in the plane). The internal part of the tree consists of an ordinary Region Quadtree. The leaf nodes of this Quadtree are bucket nodes that hold the actual line segments. Each line segment is stored in every bucket whose quadrant (region) it crosses. A line segment can cross the region of a bucket either fully or partially. Each bucket has a maximum capacity. When this capacity is exceeded due to an insertion of a line segment, the bucket is split in four equal quadrants. However, it is possible that one (or more) of these quadrants holds a number of line segments that still exceeds the bucket capacity. Since this is not occurring very often in practical applications (e.g., when line segments represent a road network) a bucket is split only once in four and overflow buckets are created when needed. The PMR quadtree can be implemented with bucket nodes residing on disk.

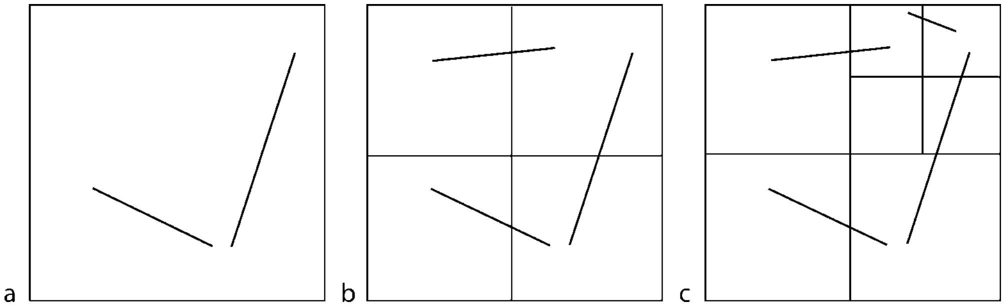
In Fig. 4, an example of the splitting of regions during the creation of a PMR Quadtree by the successive insertion of line segments is depicted. The bucket capacity is two (just for demonstration purposes). Figures 4a, 4b and 4c show the subdivision of space and the buckets created as line segments are inserted. Overflow buckets do not result from the insertions of Fig. 4. Note that the shape of the PMR Quadtree depends on the order of insertion of the line segments.

XBR Tree

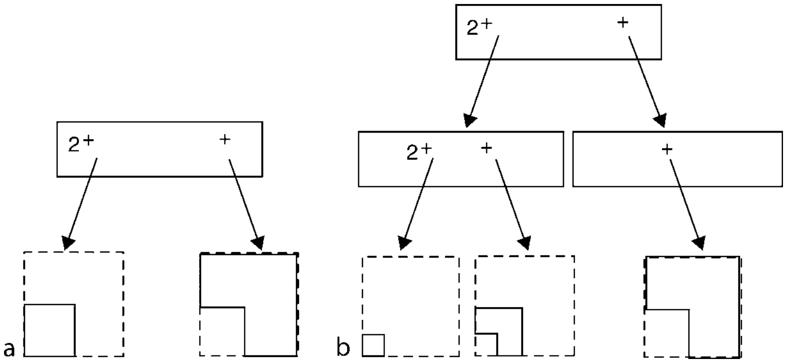
The XBR tree (XBR stands for eXternal Balanced Regular) is an indexing method suitable for indexing points (like a bucket PR Quadtree), or line segments (like a PMR Quadtree). It totally resides in secondary memory. Its hierarchical decomposition of space is the same as the one in Region Quadtrees. There are two types of nodes in an XBR-tree. The first are the internal nodes



Quadtrees (and Family). Figure 3. A collection of points (a) and the corresponding PR quadtree (b).



Quadtrees (and Family). Figure 4. Splitting of PMR-quadtree regions by the successive insertion of line segments.



Quadtrees (and Family). Figure 5. XBR trees with one level (a) and two levels (b) of internal nodes.

that constitute the index. The second are the leaves containing the data items. Both the leaves and the internal nodes correspond to disk pages.

In an internal node, a number of pairs of the form <address, pointer> are contained. The number of these pairs is non-predefined because the addresses being used are of variable size. An address expresses a child node region and is paired with the pointer to this child node. Both the size of an address and the total space occupied by all pairs within a node must not exceed the node size. The addresses in these pairs are used to represent certain subquadrants that result from the repetitive subdivision of the initial space. This is done by assigning the numbers 0, 1, 2 and 3 to NW, NE, SW and SE quadrants respectively. For example, the address 1 is used to represent the NE quadrant of the initial space, while the address 10 to represent the NW subquadrant of the NE quadrant of the initial space.

In the XBR-tree, the region of a child is the subquadrant specified by the address in its pair, minus the subquadrants corresponding to all the previous pairs of the internal node to which it belongs. Figure 5 presents XBR-trees of one (a) and two (b) levels of internal nodes. The $^{+}$, is used to denote the end of each variable size address. The address 2^{+} in the root denotes the SW quadrant of the initial space. On the other hand, the address $^{+}$ in the root specifies the initial space minus the SW quadrant.

Each leaf node in the XBR tree may contain a number of data items, which is limited by a predefined capacity C. When an insertion causes the number of data items of a leaf to exceed C, the leaf is split following a hierarchical decomposition analogous to the quadtree decomposition. In case line segments are stored, (like PMR Quadtrees) a leaf is split only once in four and overflow buckets are created when needed.

Due to the incremental (level-by-level) formation of absolute addresses and the variable length coding of them, XBR trees are very compact structures. Thus, I/O during query processing is reduced, favoring processing efficiency.

Quadtree and Time-Evolving Regional Data

In Tzouramanis et al. [13] and previous papers by the same authors, four temporal extensions of the Linear Region Quadtree are presented: the Time-Split Linear

Quadtree, the Multiversion Linear Quadtree, the Multiversion Access Structure for Evolving Raster Images and Overlapping Linear Quadtrees. These methods comprise a collection of specialized quadtree-based access methods that can efficiently store and manipulate consecutive raster images. Through these methods, efficient support for spatio-temporal queries referring to the past is provided. An extensive experimental space and time performance comparison of all the above-mentioned access methods, presented in Tzouramanis et al. [13], has shown that the Overlapping Linear Quadtrees is the best performing method. For more details, see [13] and its references.

Key Applications

Quadtree-based access methods speed up access and queries in database systems that support spatial data. Some common uses of Quadtrees include representation and indexing of images, spatial indexing for several spatial types (points, regions, line segments, polygonal maps), several set operations, point location, range, and nearest neighbor queries and temporal queries on series of evolving images. Quadtrees have been employed in numerous application areas that require efficient retrieval of complex objects, such as Computer Graphics and Animation, Computer Vision, Robotics, Geographical Information Systems (GIS), Image Processing, Image and Multimedia Databases, Content-Based Image Retrieval, Medical Imaging, Urban Planning, Computer-Aided Design (CAD), or even in recent novel database applications such as P2P Networks. Furthermore, together with the R-tree family, Quadtrees serve as an important bridge for extending spatial databases to applications of several scientific areas, such as Agriculture, Oceanography, Atmospheric Physics, Geology, Astronomy, Molecular Biology, etc. Commercial database vendors like IBM and Oracle [7] have recently implemented the Quadtree and the Linear Quadtree to cater for the large and diverse above application markets.

Future Directions

Since Quadtrees were mainly introduced as main memory structures, the development of further external memory versions of several Quadtree variations and the study of their performance for several queries remain a target. Recent papers, like [5], show that the comparative performance study for several query types between space-driven and data-driven indexing techniques can lead to interesting conclusions. Algorithms for queries based on the joining of data (e.g., image data) traditionally stored in Quadtree structures and other types of spatial data stored in data-driven structures (e.g., point data stored in R-tree family structures) are worth developing and studying, especially when the evolution of the data is considered (spatio-temporal data).

Cross-references

▶ Indexing
▶ Indexing Historical Spatio-temporal Data
▶ Main Memory
▶ Multidimensional Indexing
▶ Query Processing and Optimization in Object Relational Databases
▶ R-Tree (and Family)
▶ Raster Data Management
▶ Spatial Indexing Techniques
▶ Tree-based Indexing

Recommended Reading

1. Brabec F. and Samet H. Spatial Index Demos. http://donar.umiacs.umd.edu/quadtree/index.html
2. Eppstein D., Goodrich M.T., and Sun J.Z. The skip quadtree: a simple dynamic data structure for multidimensional data. In Proc. 21st Annual Symp. on Computational Geometry, 2005, pp. 296–305.
3. Finkel R. and Bentley J.L. Quad trees: a data structure for retrieval on composite keys. Acta Informatica, 4(1):1–9, 1974.
4. Gargantini I. An effective way to represent quadtrees. Commun. ACM, 25(12):905–910, 1982.
5. Kim Y.J. and Patel J.M. Rethinking choices for multi-dimensional point indexing: making the case for the often ignored quadtree. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 281–291.
6. Klinger A. and Dyer C. Experiments on picture representation using regular decomposition. Comput. Graph. Image Process., 5:68–105, 1976.
7. Kothuri R., Ravada S., and Abugov D. Quadtree and r-tree. indexes in oracle spatial: a comparison using gis data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 546–557.
8. Manouvrier M., Rukoz M., and Jomier G. Quadtree-Based Image Representation and Retrieval. In Spatial Databases: Technologies, Techniques and Trends. Idea Group Publishing, 2005, pp. 81–106.
9. Samet H. Applications of Spatial Data Structures. Addison Wesley, Reading, MA, USA, 1990.
10. Samet H. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, 2006.
11. Samet H. The Design and Analysis of Spatial Data Structures. Addison Wesley, 1990.
12. Shaffer C.A. and Brown P.R. A Paging Scheme for Pointer-Based Quadtrees. In Proc. 3rd Int. Symp. Advances in Spatial Databases, 1993, pp. 89–104.

13. Tzouramanis T., Vassilakopoulos M., and Manolopoulos Y. Benchmarking access methods for time-evolving regional data. Data Knowl. Eng., 49(3):243–286, 2004.
14. Vassilakopoulos M. and Manolopoulos Y. External balanced regular (x-BR) trees: New structures for very large spatial databases. In Advances in Informatics, D.I. Fotiadis, S.D. Nikolopoulos. World Scientific, 2000, pp. 324–333.
15. Vassilakopoulos M., Manolopoulos Y., and Economou K. Overlapping quadtrees for the representation of similar images. Image Vis. Comput., 11(5):257–262, 1993.

Qualitative Relations between Time Intervals

► Allen's Relations

Qualitative Temporal Constraints between Time Intervals

► Allen's Relations

Qualitative Temporal Reasoning

Paolo Terenziani
University of Turin, Turin, Italy

Synonyms

Non-metric temporal reasoning; Reasoning with qualitative temporal constraints

Definition

Qualitative temporal constraints are non-metric *temporal constraints* stating the *relative* temporal position of facts that happen in time (e.g., fact F_1 is *before* or *during* fact F_2). Different types of qualitative constraints can be defined, depending on whether facts are instantaneous, durative, and/or repeated. *Qualitative temporal reasoning* is the process of reasoning with such temporal constraints. Given a set of qualitative temporal constraints, qualitative temporal reasoning can be used for different purposes, including checking their *consistency*, determining the *strictest constraints* between pairs of facts (e.g., for query answering purposes), pointing out a *consistent scenario* (i.e., a possible instantiation of all the facts on the timeline in such a way that all temporal constraints are satisfied).

Historical Background

In several domains and\or application areas, *temporal indeterminacy* has to be coped with. In such domains, the *absolute time* when facts hold (i.e., the exact temporal location of facts) is generally unknown. On the other hand, in many of such domains, *qualitative* temporal constraints about the *relative* temporal location of facts are available, and reasoning about such constraints is an important task. As a consequence, there is a long tradition for qualitative temporal reasoning within *philosophical logic* (consider, e.g., Walker [15], who first formulated a kind of logical calculus about durative periods, and Prior's seminal branching time logics [11].

In particular, qualitative temporal reasoning is important in several artificial intelligence areas, including planning, scheduling, and natural language understanding. Therefore, starting from the beginning of the 1980s, several *specialized* approaches (as opposed to *logical* approaches, which are usually general-purpose) to the *representation* of *qualitative* temporal constraints and to temporal *reasoning* about them have been developed, especially within the artificial intelligence area.

The first milestone in the specialized approaches to qualitative temporal reasoning dates back to Allen's Interval Algebra [1], coping with qualitative temporal constraints between intervals (called *periods* by the database community, and henceforth in this entry), to cope with durative facts. Further on, several other algebrae of qualitative temporal constraints have been developed, to cope with instantaneous [14] or repeated/periodic [9,12] facts, and several temporal reasoning systems have been implemented and used in practical applications (see e.g., some comparisons in Delgrande et al. [4]).

Significant effort in the area has been devoted to the analysis of the *trade-off* between the *expressiveness* of the representation formalisms and the *complexity* of *correct* and *complete* temporal reasoning algorithms operating on them (see e.g., the survey by Van Beek [13]). Since consistency checking in Allen's algebra is NP-complete, several approaches, starting from Nebel and Burkert [10], have focused on the identification of *tractable fragments* of it. The *integration* of qualitative and metric constraints has also been analyzed (see e.g., Jonsson and Backstrom [7]). Recent developments also include *incremental* [6] and *fuzzy* [2] qualitative temporal reasoning. Moreover, starting in the 1990s, some approaches also started to investigate the adoption of qualitative temporal constraints and temporal reasoning within the temporal database context [8,3].

Foundations

Qualitative temporal constraints concern the relative temporal location of facts on the timeline. A significant and milestone example is Allen's Interval Algebra (IA). Allen pointed out the 13 primitive qualitative relations between time periods: before, after, meets, met-by, overlaps, overlapped-by, starts, started-by, during, contains, ends, ended-by, equal. These relations are *exhaustive* and *mutually exclusive*, and can be combined in order to represent disjunctive relations. For example, the constraints in (Ex.1) and (Ex.2) state that F_1 is before or during F_2, which, in turn, is before F_3.

(Ex.1) F_1 (BEFORE,DURING) F_2
(Ex.2) F_2 (BEFORE) F_3

In Allen's approach, qualitative temporal reasoning is based on two algebraic operations over relations on time periods: intersection and composition. Given two possibly disjunctive relations R_1 and R_2 between two facts F_1 and F_2, temporal intersection (henceforth $\cap$) determines the most constraining relation R between F_1 and F_2. For example, the temporal intersection between (Ex.2) and (Ex.3) is (Ex.4). On the other hand, given a relation R_1 between F_1 and F_2 and a relation R_2 between F_2 and F_3, composition (@) gives the resulting relation between F_1 and F_3. For example, (Ex.5) is the composition of (Ex.1) and (Ex.2) above.

(Ex.3) F_2 (BEFORE,MEETS,OVERLAPS) F_3
(Ex.4) F_2 (BEFORE) F_3
(Ex.5) F_1 (BEFORE) F_3

In Allen's approach, temporal reasoning is performed by a *path consistency* algorithm that basically computes the *transitive closure* of the constraints by repeatedly applying intersection and composition. Abstracting from many optimizations, such an algorithm can be schematized as follows:

```
Repeat
For all triples of facts<$F_i$,$F_k$,$F_j$>
    Let $R_{ij}$ denote the (possibly ambiguous) relation
between $F_i$ and $F_j$
    $R_{ij} \leftarrow R_{ij} \cap (R_{ik} @ R_{kj})$
    Until quiescence
```

Allen's algorithm operates in a time *cubic* in the number of periods. However, such an algorithm is *not complete* for the Interval Algebra (in fact, checking the consistency of a set of temporal constraints in the Interval Algebra is NP-hard [14]).

While in many approaches researchers chose to adopt Allen's algorithm, in other approaches they tried to design less expressive but *tractable* formalisms. For example, the Point Algebra is defined in the same way as the Interval Algebra, but the temporal elements are time points. Thus, there are only three primitive relations between time points (i.e., <,=, and>), and four disjunctive relations (i.e., (<,=), (>,=), (<,>), and (<,=,>). In the Point Algebra, sound and complete constraint propagation algorithms operate in polynomial time (namely, in $O(n^4)$, where n is the number of points [13]). Obviously, the price to be paid for tractability is expressive power: not all (disjunctive) relations between periods can be mapped onto relations between their endpoints. For instance *F_1 (BEFORE,AFTER) F_2* cannot be mapped into a set of (possibly disjunctive) pairwise relations between time points; indeed an explicit disjunction between two different pairs of time points is needed (i.e., *(end(F_1)<start(F_2)) or (end(F_2)<start(F_1))*. The Continuous Point Algebra restricts the Point Algebra excluding inequality (i.e., (<,>)). Allen's path consistency algorithm is both sound and complete for such an algebra, and operates in $O(n^3)$ time, where n is the number of time points (for more details, see e.g., the survey by Van Beek [13]).

A different simplification of Allen's Algebra has been provided by Freksa [5]. Freska has identified coarser qualitative temporal relations than Allen's ones, based on the notion of *semi-intervals* (i.e., beginnings and ending points of durative events). As an example of relation on semi-intervals, Freksa has introduced the "older" relation (*F_1* is *older* than *F_2* if *F_1*'s starting point is before *F_2*'s starting point, with no constraint on the ending points). Notice that Freska's *older* relation corresponds to a disjunction of five Allen's relations (i.e., *before*, *meets*, *overlaps*, *finished-by*, *contains*). Freska has also shown that relations between semi-intervals result in a possible more compact notation and more efficient reasoning mechanisms, in particular if the initial knowledge is, at least in part, coarse knowledge.

Another mainstream of research about qualitative temporal reasoning focused on the identification of *tractable fragments* of Allen's algebra. The milestone work by Nebel and Burkert [10] first pointed out the "*ORD-Horn subclass*," showing that reasoning in such a class is a polynomial time problem and that it constitutes a maximal tractable subclass of Allen's algebra.

Starting in the early 1990s, some *integrated* temporal reasoning approaches were devised in order to deal with both qualitative and quantitative (i.e., metric) temporal constraints. For instance, Jonsson and Backstrom [7] proposed a framework, based on *linear programming*, that deals with both qualitative and metric constraints, and that also allows one to express constraints on the relative duration of events (see e.g., (Ex.6)).

(Ex.6) John's drive to work is at least 30 minutes more than Fred's.

Many other important issues must be taken into account when considering qualitative temporal reasoning. For example, starting from Ladkin's seminal work [9], qualitative constraints between *repeated* facts have been considered. In the same mainstream, Terenziani has proposed an extension of Allen's algebra to consider qualitative relations between periodic facts [12]. Terenziani's approach deals with constraints such as (Ex.7):

(Ex.7) Between January 1, 1999 and December 31, 1999 on the first Monday of each month, Andrea went to the post office *before* going to work.

In Terenziani's approach, temporal reasoning over such constraints is performed by a path consistency algorithm which extends Allen's one. Such an algorithm is sound but not complete and operates in cubic time with respect to the number of periodic facts.

As concerns more strictly the area of (temporal) databases, starting in the mid 1990s, some researchers started to investigate the treatment of qualitative temporal constraints within temporal (relational) databases (see e.g., [8,3]). In such approaches, the *valid time* of facts (tuples) is represented by symbols denoting time periods, and *qualitative* and *quantitative temporal constraints* are used in order to express constraints on their relative location in time, on their duration, and so on.

Koubarakis [8] first extended the constraint database model to include indefinite (or uncertain) temporal information (including qualitative temporal constraints). Koubarakis proposed an explicit representation of temporal constraints on data; moreover, the local temporal constraints on tuples are stored into a dedicated attribute. He also defined the algebraic operators, and theoretically analyzed their complexity. On the other hand, the work by Brusoni et al., [3] mainly focused on defining an integrated approach in which "standard" artificial intelligence temporal reasoning capabilities (such as the ones sketched above in this entry) are suitably extended and paired with an (extended) relational temporal model. First, the data model is extended in such a way that each temporal tuple can be associated with a set of identifiers, each one referring to a time period. A separate relation is used in order to store the qualitative (and quantitative) temporal constraints of such periods. The algebraic operations of intersection, union and difference are defined over such sets of periods, and *indeterminacy* (e.g., about the existence of the intersection between two periods) is coped with through the adoption of *conditional intervals*. Algebraic relational operators are defined on such a data model, and their complexity analyzed. Finally, an integrated and modular architecture combining a temporal reasoner with an extended temporal database is described, as well as a practical application to the management of temporal constraints in clinical protocols and guidelines.

Key Applications

Qualitative temporal constraints are pervasive in many application domains, in which the absolute and exact time when facts occur is generally unknown, while there are constraints on their relative order (or temporal location). Such domains include the "classical" domains of planning and scheduling, but also more recent ones such as managing multimedia presentations or clinical guidelines.

As a consequence, temporal reasoning is already a well-consolidated area of research, especially within the artificial intelligence community, in which a large deal of works aimed at building *application-independent* and *domain-independent* managers of temporal constraints. Such managers are intended to be *specialized knowledge servers* that represent and reason with temporal constraints, and that *co-operate* with other software modules in order to solve problems in different applications. For instance, in planning problems, a temporal manager could co-operate with a planner, in order to check incrementally the temporal consistency of the plan being built. In general, the adoption of a specialized temporal manager is advantageous from the computational point of view (e.g., with respect to general logical approaches based on theorem proving), and it allows programmers to focus on their domain-specific and application-specific problems and to design modular architectures for their systems.

On the other hand, the impact and potentiality of extensively exploiting qualitative temporal reasoning in temporal databases have only been minimally explored by the database community, possibly due to the computational complexity that it necessarily involves. However, (temporal) databases will be increasingly applied to new applications domains, in which the structure and the inter-dependencies of facts (including the temporal dependencies) play a major role, while the assumption that the absolute temporal location of facts is known does no longer hold. Significant application areas include database applications to store workflows, protocols, guidelines (see, e.g., the example in [3]), and so on. To cope with such applications, "hybrid" approaches in which qualitative (and/or quantitative) temporal reasoning mechanisms are paired with classical temporal database frameworks (e.g., along the lines suggested in [3]) are likely to play a significant role in a near future. The role of qualitative temporal constraints (and temporal reasoning) may be even more relevant at the *conceptual* level. Several temporal extensions to conceptual formalisms (such as the Entity-Relationship) have been proposed in recent years, and there is an increasing awareness that, in many domains, qualitative (and/or quantitative) temporal constraints between conceptual objects are an intrinsic part of the conceptual model. As a consequence, qualitative temporal reasoning techniques such as the ones discussed above, may in the near future, play a relevant role at the conceptual modeling level.

Future Directions

One of several possible future research directions of qualitative temporal reasoning, which may be particularly interesting for the database community, is its application to "Active Conceptual Modeling." In his keynote talk at ER'2007, Prof. P. Chen, the creator of the Entity-Relationship model, has stressed the importance of extending traditional conceptual modeling to "Active Conceptual Modeling." Roughly speaking, the term "active" denotes the need for coping with evolving models having learning and prediction capabilities. Such an extension is needed in order to adequately cope with a new range of phenomena, including disaster prevention and management. Chen has stressed that "Active Conceptual Modeling" requires, besides the others, an explicit treatment of time. The extension and integration of qualitative temporal reasoning techniques into the "Active Conceptual Modeling" context is likely to give a major contribution to the achievement of predictive and learning capabilities, and to become a potentially fruitful line of research.

Cross-references

- ► Absolute Time
- ► Allen's Relations
- ► Relative Time
- ► Temporal Constraints
- ► Temporal Indeterminacy
- ► Temporal Integrity Constraints
- ► Temporal Periodicity
- ► Time in Philosophical Logic
- ► Time Period
- ► Valid Time

Recommended Reading

1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832–843, 1983.
2. Badaloni S. and Giacomin M. The algebra IA^{fuz}: a framework for qualitative fuzzy temporal reasoning. Artif. Intell., 170 (10):872–908, 2006.
3. Brusoni V., Console L., Pernici B., and Terenziani P. Qualitative and quantitative temporal constraints and relational databases: theory, architecture, and applications. IEEE Trans. Knowl. Data Eng., 11(6):948–968, 1999.
4. Delgrande J., Gupta A., and Van Allen T. A comparison of point-based approaches to qualitative temporal reasoning. Artif. Intell., 131(1–2):135–170, 2001.
5. Freksa C. Temporal reasoning based on semi-intervals. Artif. Intell., 54(1–2):199–227, 1992.
6. Gerevini A. Incremental qualitative temporal reasoning: algorithms for the point algebra and the ORD-Horn class. Artif. Intell., 166(1–2):37–80, 2005.

7. Jonsson P. and Backstrom C. A unifying approach to temporal constraint reasoning. Artif. Intell., 102:143–155, 1998.
8. Koubarakis M., Database models for infinite and indefinite temporal information. Inf. Syst., 19(2):141–173, 1994.
9. Ladkin P. Time representation: a taxonomy of interval relations. In Proc. 5th National Conf. on AI, 1986. pp. 360–366.
10. Nebel B. and Burkert H.J. Reasoning about temporal relations: a maximal tractable subclass of Allen's interval algebra. J. ACM, 42(1):43–66, 1995.
11. Prior A.N. Past, Present and Future. Oxford University Press, Oxford, 1967.
12. Terenziani P. Integrating calendar-dates and qualitative temporal constraints in the treatment of periodic events. IEEE Trans. Knowl. Data Eng., 9(5):763–783, 1997.
13. Van Beek P. Reasoning about qualitative temporal information. Artif. Intell., 58(1–3):297–326, 1992.
14. Vilain M. and Kautz H. Constraint propagation algorithms for temporal reasoning. In Proc. 5th National Conf. on AI, 1986, pp. 377–382.
15. Walker A.G. Durèes et instants. La Revue Scientifique, (3266), p. 131, 1947.

Quality and Trust of Information Content and Credentialing

Chintan Patel, Chunhua Weng
Columbia University, New York, NY, USA

Definition

The quality and reliability of biomedical information is critical for practitioners (clinicians and biomedical scientists) to make important decisions about patient conditions and to draw key scientific conclusions towards developing new drugs, therapies and procedures. Evaluating the quality and trustworthiness of biomedical information [1] requires answering questions such as, *where did the data come from*, *under what conditions was the data generated, how accurate and complete is the data,* and so on.

Key Points

The quality and reliability of biomedical information is dependent on the task or the context of the application. There is a basic set of domain-independent features that can be used to characterize the quality and trustworthiness of the information:

Accuracy: The correctness of the information. Inherent noisiness in the underlying data generating clinical processes or biological experiments often leads to various errors in the resulting data. It is critical to quantify the frequency and source of such errors using the measure of accuracy to enable the clinicians and researchers for making informed decisions using the data.

Completeness: In biomedical settings, it is often necessary to have all the pertinent (complete) information at hand while making important clinical decisions or performing complex biological experiments. There are various constraints due to limited technology and the nature of clinical practice that leads to incomplete biomedical data:

1. In healthcare settings, only partial data gets recorded in electronic form and vast amount of data is still only available on paper [4]. For example, various clinical notes such as admit, progress and discharges notes containing valuable clinical information are generally written on paper charts and not entered electronically.
2. In several instances, despite the availability of electronic data, the information is not usable or accessible due to differences in underlying data standards, information models, terminology, etc. Consider for example, two biological databases using different ontologies for protein annotation, which will not be able to support data reuse or sharing across.

Transparency: An important aspect for determining the trustworthiness of information is the understanding or knowledge of the underlying processes or devices generating the data. Various tools or resources that provide transparency to data sources are more likely to be trusted by clinicians or biologists [2].

Credentialing: The trustworthiness of information in turn depends on the level of trust in data originators. Hence, it is important to attribute the information to its source in order to allow consumers of information to make appropriate decisions. Consider for example that a biological annotation reviewed by human curators will be trusted more than an annotation automatically extracted from literature using text-mining [3].

As the biomedical domain becomes more and more information driven, the methods and techniques to characterize the quality and trustworthiness of information will gain more prominence.

Cross-references

► Clinical Data Quality and Validation

Recommended Reading

1. Black N. High-quality clinical databases: breaking down barriers. Lancet, 353(9160):1205–1211, 1999.
2. Buza T.J., McCarthy F.M., Wang N., Bridges S.M., and Burgess S.C. Gene Ontology annotation quality analysis in model eukaryotes. Nucl. Acids Res., 36(2):e12, 2008.
3. D'Ascenzo M.D., Collmer A., and Martin G.B. PeerGAD: a peer-review-based and community-centric web application for viewing and annotating prokaryotic genome sequences. Nucl. Acids Res., 32(10):3124–3159, 2004.
4. Thiru K., Hassey A., and Sullivan F. Systematic review of scope and quality of electronic patient record data in primary care. BMJ, 26(7398):1070, 2003.

Quality Assessment

▶ Clustering Validity

Quality of Data Warehouses

Rafael Romero[1], Jose-Norberto Mazón[1], Juan Trujillo[1], Manuel Serrano[2], Mario Piattini[2]
[1]University of Alicante, Alicante, Spain
[2]University of Castilla – La Mancha, Spain

Definition

Quality is an abstract and subjective aspect for which there is no universal definition. It is usually said that there is a quality definition for each person. Perhaps the most abstract definition for this topic is that the data warehouse quality means the data is suitable for the intended application by all users. In this way, it is very complex to measure or assess the quality of a data warehouse system. Normally, the *data warehouse* quality is determined by: (i) the quality of the data presentation, and (ii) the quality of the *data warehouse* itself. The latter is determined by the quality of the Database Management System (DBMS), the data quality and the quality of the underlying data models used to design it. A good design may (or may not) lead to a good data warehouse, but a bad design will surely render a bad data warehouse of low quality. In order to measure the quality of a data warehouse, a key issue is defining and validating a set of measures to help to assess the quality of a data warehouse in an objective way, thus guaranteeing the success of designing a good data warehouse.

Historical Background

Few works have been presented in the area of objective indicators or measures for data warehouses. Instead, most of the current proposals for data warehouses still delegate the quality of the models to the experience of the designer.

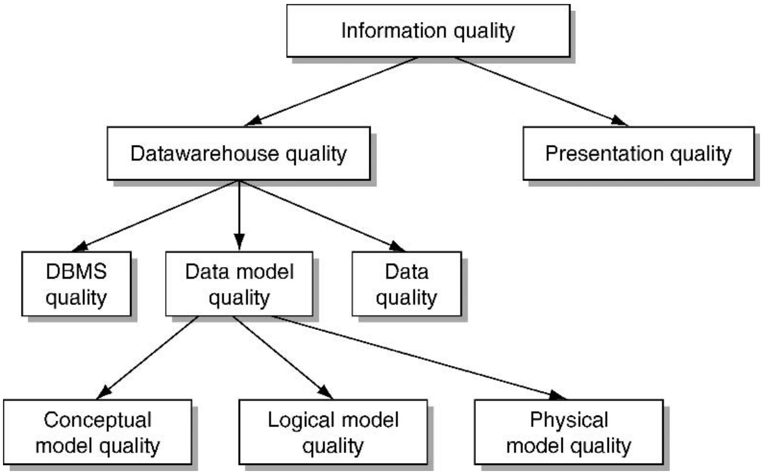
Only the model proposed by Jarke et al. [5], which is described in more depth in Vassiladis' Ph.D. thesis [14], explicitly considers the quality of conceptual models for data warehouses. Nevertheless, these approaches only consider quality as intuitive notions. In this way, it is difficult to guarantee the quality of data warehouse conceptual models, a problem which has initially been addressed by Jeusfeld et al. [6] in the context of the DWQ (Data Warehouse Quality) project. This line of research addresses the definition of measures that allows designers to replace the intuitive notions of quality of conceptual models of the data warehouse with formal and quantitative measures. Sample research in this direction includes normal forms for data warehouse design as originally proposed in [8] and generalized in [7]. These normal forms represent a first step towards objective quality metrics for conceptual schemata.

Following the idea of assessing the quality of data warehouses in an objective way, several measures for evaluating the quality of data warehouse logical models have been proposed in recent years and validated both formally and empirically [11,12].

Lately, Si-Saïd and Prat [13] have proposed some measures for multidimensional schemas analyzability and simplicity. Nevertheless, none of the measures proposed so far has been empirically validated, and therefore, their practical utility has not been proven.

Foundations

The information quality of a data warehouse is determined by: (i) the quality of the system itself, and (ii) the quality of the data presentation (see Fig. 1). In fact, it is important that the data of the data warehouse not only correctly reflects the real world, but also that the data are correctly interpreted. Regarding data warehouse quality, three aspects must be considered: the quality of the DBMS (Database Management System) that supports it, the quality of the data models



Quality of Data Warehouses. Figure 1. Quality of the information and the data warehouse.

used in their design (conceptual, logical and physical), and the quality of the data contained in the data warehouse. The presentation quality of data warehouses is more related to the data presentation according to front-end tools such as OLAP (On-Line Analytical Processing), data reporting or data mining. For this reason, the following issues pertaining to data warehouse quality are described next: (i) DBMS quality, (ii) data model quality, and (iii) data quality.

Quality of DBMS

The quality of the DBMS in which the data warehouse is implemented is important, since the database engine is the core of the data warehouse system and has a deep impact on the performance of the whole system. A good DBMS could improve the performance of the system and the quality of the data by implementing constraints and integrity rules. In order to assess the quality of a DBMS, several international standards can be used, such as ISO/IEC 9126 [3] and ISO/IEC 9075 [4] or even information from database benchmarks [9].

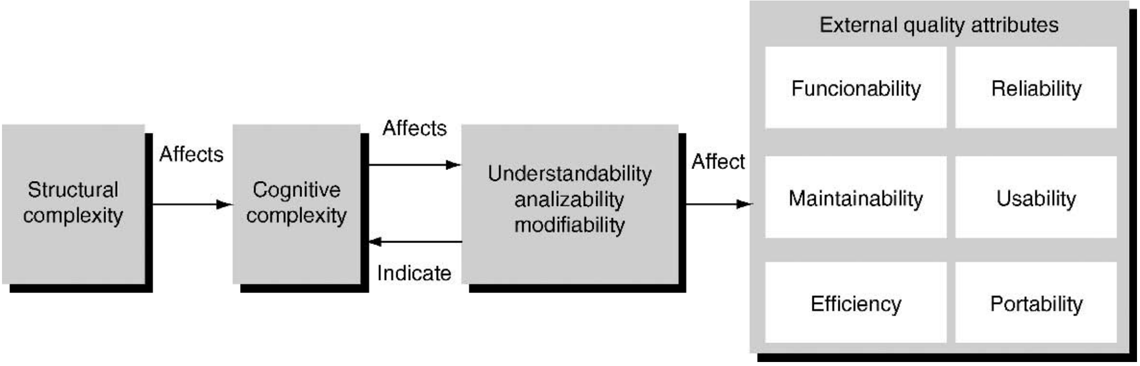
Quality of Data Warehouse Data Models

The quality of the data models used in the design of data warehouses relies on the quality of the conceptual, logical, and physical models used for its design. A first step towards obtaining high quality data models is the definition of development methodologies. However, a methodology, though necessary, may not be sufficient to guarantee the quality of a data warehouse. Indeed, a good methodology may (or may not) lead to a good product, but a bad methodology will surely render a bad product of low quality. Furthermore, many other factors could influence the quality of the products, such as human decisions. Therefore, it is necessary to complete specific methodologies with measures and techniques for product quality assessment.

Structural properties (such as structural complexity) of a software artifact have an impact on its cognitive complexity as shown in Fig. 2. Cognitive complexity means the mental burden on those who have to deal with the artifact (e.g., developers, testers, maintainers). High cognitive complexity of an artifact reduces its understandability and leads to undesirable external quality attributes as defined in the standard ISO/IEC 9126 [3]. The model presented in Fig. 2 is an adaptation of the general model for software artifacts proposed in Briand et al. [2].

Indeed, as data warehouse models are software artifacts, it is reasonable to consider that they follow the same pattern. It is thus important to investigate the potential relationships that can exist between the structural properties of these schemas and their quality factors.

In order to get a valid set of data warehouse measures, the definition of measures should be based on clear measurement goals and the measures should be defined following the organization's needs related to external quality attributes. In defining measures, it is also advisable to take into account the expert's



Quality of Data Warehouses. Figure 2. Relationship between Structural Properties, Cognitive Complexity, Understandability and External Quality Attributes – based on the work described in Briand et al. [12].

knowledge. Figure 3 presents a method (based on the method followed in [11,12]) for obtaining valid and useful measures. In this figure, continuous lines show measure flow and dotted lines show information flow.

This method has five main phases starting at the identification of goals and hypotheses and leading to the measure application, accreditation, and retirement:

1. *Identification*: Goals of the measures are defined and hypotheses are formulated. All of the subsequent phases will be based upon these goals and hypotheses.

2. *Creation*: This is the main phase, in which measures are defined and validated. This phase is divided into three sub phases:

a) *Measures definition*. Measure definition is made by taking into account the specific characteristics of the system to be measured, the experience of the designers of these systems and the work hypotheses. A goal-oriented approach as GQM (Goal-Question-Metric [1]) can also be very useful in this step.

b) *Theoretical validation*. The formal (or theoretical) validation helps identify when and how to apply the metrics. There are two main tendencies in measuring formal validation: the frameworks based on axiomatic approaches [7] and the ones based on measurement theory [10]. The goal of the former is merely definitional, as in this type of formal framework, a set of formal properties is defined for given software attributes and it is possible to use this set of properties for classifying the proposed measures. On the other hand, in the frameworks based on measurement theory, the information obtained is the scale to which a measure pertains and, based on this information, statistics and transformations, which can be applied to the measure, can be known.

c) *Empirical validation*. The goal of this step is to prove the practical utility of the proposed measure. Empirical validation is crucial for the success of any software measurement project as it helps to confirm and understand the implications of the measurement of products. Although there are various ways perform to this step, the empirical validation can be divided into experiments, case studies, and surveys.

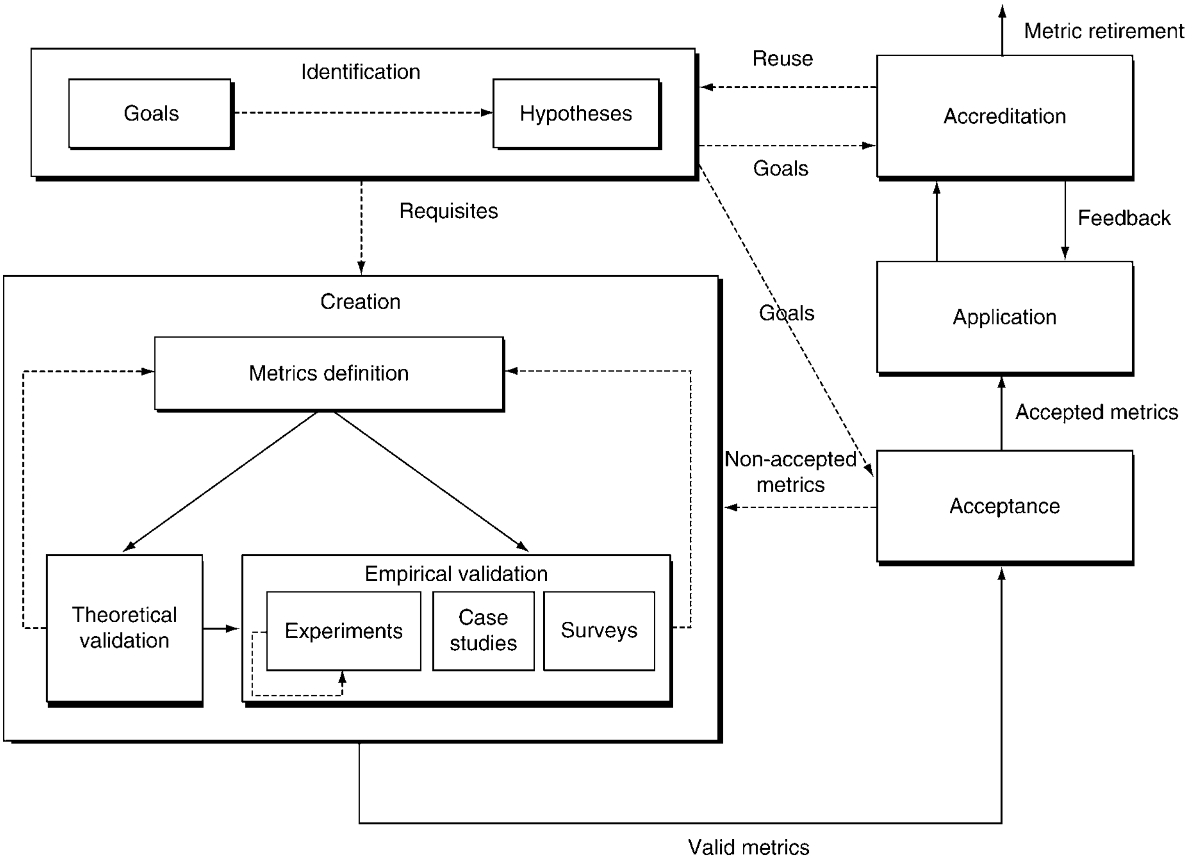
This process is evolutionary and iterative and as a result of the feedback, the measure could be redefined or discarded, depending on their formal and empirical validation. As a result of this phase a valid metric is obtained.

3. *Acceptance*: The goal of this phase is the systematic experimentation of the measure. This is applied in a context suitable to reproduce the characteristics of the application environment, with real business cases and real users, to verify its performance against the initial goals and stated requirements.

4. *Application*: The accepted measure is used in real cases.

5. *Accreditation*: This is the final phase of the process. It is a dynamic phase that proceeds simultaneously with the application phase. The goal of this phase is the maintenance of the measure, so it can be adapted to the changing environment of the application. As a result of this phase, the measure can be retired or reused for a new measure definition process.

The most important data models for measuring quality are conceptual and logical models as the quality of physical models has to do with performance issues and the physical distribution of data. Following the above method, a set of measures can be defined and validated for data warehouse conceptual models



Quality of Data Warehouses. Figure 3. Metrics creation process.

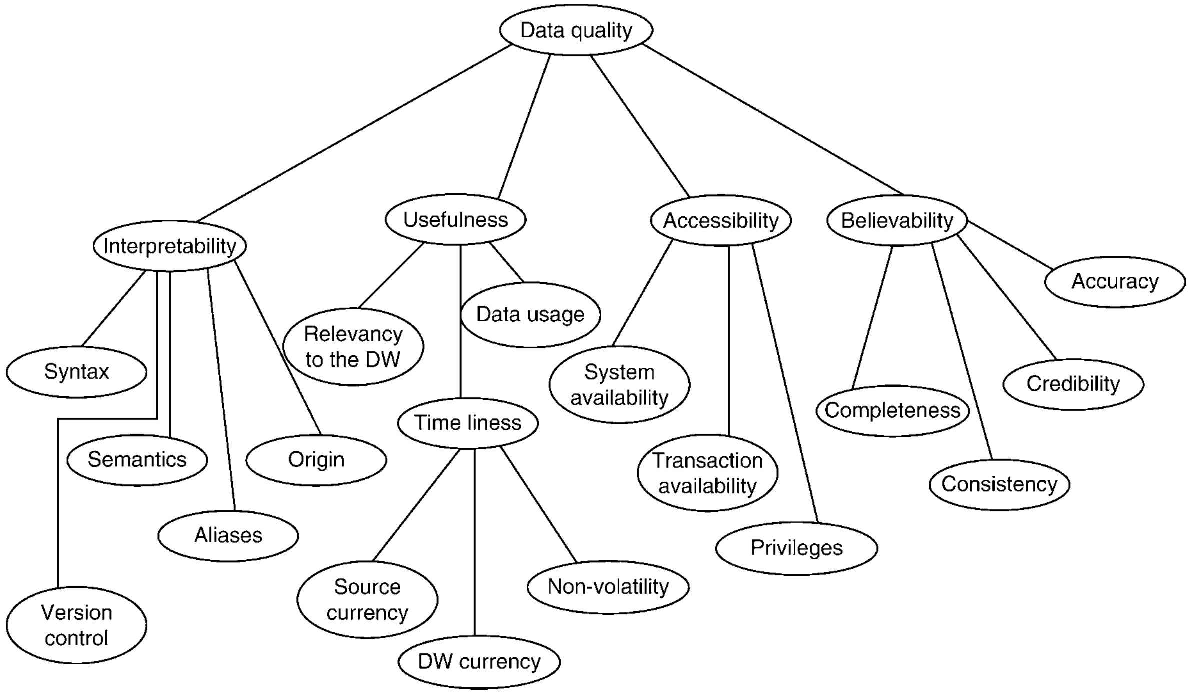
quality [12]. Some examples of the measures defined in [12] are: number of dimensions of the multidimensional conceptual model, number of hierarchy levels of the multidimensional conceptual model, ratio of hierarchy levels (i.e., number of hierarchy levels per dimension of the multidimensional conceptual model), and the maximum depth of the hierarchy relationships of the multidimensional conceptual model.

Taking into account these characteristics, a set of measures for data warehouses logical models have also been defined and validated in [11]. Specifically, these measures are defined for a relational implementation (tables, columns, foreign keys, etc.). Some examples of the measures defined in [11] are: number of fact tables of the schema, number of shared dimension tables (i.e., number of dimension tables shared for more than one star (fact) of the schema), number of foreign keys in all the fact tables of the schema, and ratio of schema attributes (i.e., number of attributes in dimension tables per attributes in fact tables).

Data Quality

Finally, data quality is a multidimensional concept that is made up of several aspects, all of which depend on the needs of the data consumers or system designers. The most accurate definition for data quality is fitness for use, i.e., how suitable the data is for a concrete task. This concept can be defined in terms of particular criteria, related either to the data life-cycle or to the way data are meant to be used. Usually, the way of determining the data quality of an application is to define a framework that specifies the data quality dimensions in a concrete domain. The reader is referred to the DWQ project [6]. Figure 4 shows the quality factors as stated in the DWQ project.

As stated in the DWQ project, data warehouse data quality is related to several characteristics or quality dimensions, such as interpretability, usefulness, accessibility and believability. These quality dimensions should be interpreted as the ability to understand, use, access and trust the data that is stored in



Quality of Data Warehouses. Figure 4. Data warehouse quality factors.

the data warehouse. Each dimension tackles a different aspect of data quality and is broken up into several sub-categories, such as timeliness, availability, clarity, and accuracy.

When dealing with interpretability, the data must be easy to understand, and special attention should be placed on the format and structure of the data. In this sense, the syntax of the data should be inspected, since aliases and abbreviations may be eluded and the origin and the different versions of the data must be controlled.

Data relevancy and timeliness are the main problems to be tackled in data usefulness. Data relevancy can be analyzed in ETL processes, while timeliness is more related to the currency of the data. It is universally known that data warehouses are typically not updated everyday, but irrelevant data cannot be allowed in the system. A good schedule of data warehouse refresh process can avoid this problem.

Accessibility is related to system availability and access privileges. Data warehouse systems should be in stable and well-dimensioned systems and privileges should be studied in great detail. It is also necessary to test and maintain the system periodically.

Believability is a very important aspect of data warehouse data quality, as it deals with completeness, consistency, credibility and accuracy of the data. Probably this quality dimension is the most studied by data quality experts, but it is usually the most difficult to assess. This quality dimension is not only related to source data quality and ETL processes' quality, it is also related to the data warehouse update processes that can pollute the data warehouse data. In this way, data cleaning algorithms could be very useful.

In order to detect data quality problems, it is advisable to have indicators that can help identify the different quality threats for each of the data warehouse data quality dimensions. There are several quality indicator proposals, but they are not oriented to the data warehouse field and are only usually oriented to a specific quality dimension such as accuracy, completeness and timeliness. Further research has to be done in this field in order to obtain a complete set of data warehouse data quality indicators.

Key Applications

The main application of Data Warehouse Quality is oriented to the work of data warehouse designers. Introducing quality aspects from early phases of data warehouse design is a good way to improve the quality and success of a data warehouse. In this way, quality models and quality driven design methods are good approaches to design successful data warehouse systems. These design methods should be based on the metrics and techniques that have been discussed in the previous sections.

Future Directions

Some open research lines within data warehouse quality are: (i) to provide a complete set of quality measures that allows to measure and assess data warehouse quality models in different dimensions such as the complexity, understandability, usability and so on (Fig. 2), (ii) the traceability of the measures throughout all the design steps (conceptual, logical and physical), and (iii) the measure's thresholds, which should be the last step in defining measures, i.e., being able to provide a number for each measure under which one can completely assure the quality of a model. The latter is definitely the most difficult issue that has not yet been covered.

Experimental Results

Generally speaking, before doing an experiment with data quality measures, the settings must first be defined (including the main goal of the experiment, the subjects that will participate in the experiment, the main hypothesis under which the experiment will be run, the independent and dependent variables to be used in the model, the experimental design, the experiment running, material used and the subjects that performed the experiment). After that, the collected data validation must be discussed. Finally, the results are analyzed and interpreted to find out if they follow the formulated hypothesis. Only after a complete set of experiments, and following the method presented in Fig. 3, a measure can be accepted or rejected.

The reader is referred to the work of the Software Engineering Lab [15] for expanding the concepts about experimental process. A complete set of experimental works can be found in [11,12]. In those works several data warehouse quality measures are empirically validated as data warehouse quality indicators.

Cross-references

▶ Data Warehouse
▶ Data Warehouse Life-Cycle and Design
▶ Data Warehouse Security

Recommended Reading

1. Basili V. and Weiss D.A. Methodology for collecting valid software engineering data. IEEE Trans. Software Eng., 10:728–738, 1984.
2. Briand L., Morasca S., and Basili V. Property-based software engineering measurement. IEEE Trans. Software Eng., 22 (1):68–86, 1996.
3. ISO/IEC, 9126: Software Engineering – Product quality. Geneva, Switzerland, 2003.
4. ISO/IEC, 9075: Information Technology – Database languages. Geneva, Switzerland, 2006.
5. Jarke M., Lenzerini M., Vassiliou Y., and Vassiliadis P. Fundamentals of Data Warehouses. Springer, 2002.
6. Jeusfeld M.A., Quix C., and Jarke M. Design and analysis of quality information for data warehouses. In Proc. 17th Int. Conf. on Conceptual Modeling, 1998, pp. 349–362.
7. Lechtenbörger J. and Vossen G. Multidimensional normal forms for data warehouse design. Inf. Syst., 28:415–434, 2003.
8. Lehner W., Albretch J., and Wedekind H. Normal forms for multidimensional databases. In Proc. 10th Int. Conf. on Scientific and Statistical Database Management, 1998, pp. 63–72.
9. Othayoth R. and Poess M. The Making of TPC-DS. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 1049–1058.
10. Poels G. and Dedene G. DISTANCE: a framework for software measure construction, Research Report DTEW9937, Department of Applied Economics, Katholieke Universiteit Leuven, Belgium, 1999, p. 46.
11. Serrano M., Calero C., and Piattini M. Validating metrics for data warehouses. IEE Proc. Software, 149(5):161–166, 2002.
12. Serrano M., Trujillo J., Calero C., and Piattini M. Metrics for data warehouse conceptual models understandability. Inf. Software Technol., 49(8):851–870, 2007.
13. Si-Saïd S. and Prat N. Multidimensional schemas quality: assessing and balancing analyzability and simplicity. In Proceedings of the ER 2003 Workshops, 2003, pp. 140–151.
14. Vassiliadis P. Data Warehouse Modeling and Quality Issues, Ph.D. Thesis, National Technical University of Athens, Athens, Greece, 2000.
15. Wohlin C., Runeson P., Höst M., Ohlson M., Regnell B., and Wesslén A. Experimentation in Software Engineering: An Introduction. Dordrecht, Kluwer Academic, 2000.

Quantiles on Streams

Chiranjeeb Buragohain[1], Subhash Suri[2]
[1]Amazon.com, Seattle, WA, USA
[2]University of California-Santa Barbara, Santa Barbara, CA, USA

Synonyms

Median; Histogram; Selection; Order statistics

Definition

Quantiles are order statistics of data: the ϕ-quantile $(0 \leq \phi \leq 1)$ of a set S is an element x such that $\phi|S|$ elements of S are less than or equal to x and the remaining $(1 - \phi)|S|$ are greater than x. This entry describes data stream (single-pass) algorithms for computing an approximation of such quantiles.

Historical Background

Since the earliest days of data processing, there has been a need to summarize data. Large volumes of raw, unstructured data easily overwhelm the human ability to comprehend or digest. Tools that help identify the major underlying trends or patterns in data have enormous value. Quantiles characterize distributions of real world data sets in ways that are less sensitive to outliers than simpler alternatives such as the mean and the variance. Consequently, quantiles are of interest to both database implementers and users: for instance, they are a fundamental tool for query optimization, splitting of data in parallel database systems, and statistical data analysis.

Quantiles are closely related to the familiar concepts of frequency distributions and histograms. The cumulative frequency distribution $F()$ is commonly used to summarize the distribution of a (totally ordered) set S. Specifically, for any value x,

$$F(x) = \text{Number of values less than } x. \qquad (1)$$

The quantile $Q(\phi)$, or the ϕ-th quantile is simply the inverse of $F(x)$. Specifically, if the set S has n elements, then the element x has the property that

$$Q(F(x)/n) = x. \qquad (2)$$

Thus, the 1⁄2-quantile is just the familiar median of a set, while 0- and 1-quantiles are the minimum and the maximum elements of the set. Histograms are another popular method for summarizing data into a smaller number of "buckets": the buckets only retain the information how many elements fall between two consecutive bucket boundaries, but not their precise values. It is easy to see that a sequence of quantiles resembles a *histogram* of the underlying set, and provides a natural and complete summary of the entire *distribution* of the data values.

In computer science, sorting and selection (another name for quantile computation) are two of the most basic problems, with long and intellectually rich history of research. Indeed, the computational complexity of selection, namely, determining the element of a given rank, is one of the earliest celebrated problems, and the elegant, linear-time algorithm of Blum et al. [2] is a classical result, taught regularly in the undergraduate algorithms and data structures course. For more recent theoretical results on the complexity of selection in the classical comparison-complexity model, please refer to the survey by Paterson [16].

Foundations

The problem with quantile computation, while well-solved in the classical model of computation, assumes a new and challenging character within the constraints of single-pass computation (the streaming model). Indeed, when the algorithm's memory is limited and significantly smaller than the size of the data set S, it is not possible to compute the quantile precisely, and the best possible solution is an approximation. This was formalized in a 1980 paper by Munro and Paterson [15], who proved that any algorithm that determines the median of a set by making at most p sequential scans of the input requires at least $\Omega(n^{1/p})$ working memory. Thus, computing the true median will require memory linear in the size of the set.

Against this backdrop, the main focus of recent research has been on achieving provable-quality approximation of the quantiles. In particular, an ε-approximate quantile summary of a sequence of n elements is a data structure that can answer quantile queries about the sequence to within a precision of εn. In other words, when queried for a ϕ-quantile, for $0 \leq \phi \leq 1$, the structure returns an element x that is guaranteed to be in the $[\phi - \varepsilon, \phi + \varepsilon]$ quantile range. The key evaluation metric for the performance of these approximation schemes is the size (memory footprint) of their summary data structures, although other factors such as simplicity of implementation are also desirable.

The discussion in this entry will focus on algorithms that operate in the *data stream* model: the algorithm is endowed with a finite memory, which is significantly smaller in size than the size of the input. The input is presented to the algorithm in an arbitrary (perhaps adversarial) order, and any data not explicitly stored by the algorithm is irretrievably lost. Thus, the algorithm is restricted to a single scan of the data in the input order, and after this scan it must output an approximation of the quantiles of the input values.

Before discussing the state of the art for this problem, it may help to consider a real-world scenario for the use of quantiles in data streams, both to illustrate a motivating application and to appreciate the scale of the problem. A web site, such as a search engine, consists of several web server hosts. The users' queries (requests) are collectively handled by these servers (using some scheduling protocol); and the overall performance of the web site is characterized by the latency (delay) encountered by the users. The distribution of the latency values is typically very skewed, and a common practice is to track some particular quantiles, for instance, the 95th percentile latency. In this context, one can ask several questions.

- What is the 95th percentile latency of a single web server?
- What is the 95th percentile latency of the entire web site (over all the servers)?
- What is the 95th percentile latency of the website during the last 1 h?

The Yahoo website, for instance, handles more than 3.4 billion hits per day, which translates to 40,000 requests per second. The Wikipedia website handles 30,000 requests per second at peak, with 350 web servers.

While all three questions relate to computing of quantiles, they have different technical nuances, and often require different algorithmic approaches. In particular, the first question is the most basic version, asking for a determination of quantiles for a stream of data; the second extends the setting to *distributed input*, and thus demands an algorithm in the distributed computing model. The third problem is an instance of the *sliding window* model, where the computation must occur over a subset (time window) of the stream, and this subset is continuously changing. This entry is primarily focused on the stream setting (problem 1), also known as the cash register model, but will also discuss, when appropriate, extensions to these other models.

Randomized Algorithms

One can estimate the quantiles of a stream by the quantiles of a random sample of the input. The key idea is to maintain a random sample of appropriate size and when asked for a quantile of the input set, simply report the corresponding quantile of the random sample. If the size of the input stream, N is known, then the following simple algorithm can compute a random sample of size k in one-pass: choose each element independently with probability k/N to include in the sample. If the size of the full data stream is not known in advance, or if the ability to answer queries during reading the stream is required as well, then the reservoir sampling algorithm of Vitter [19] can be used instead. To maintain a sample of size k, the reservoir sampling algorithms begins by including the first k stream elements in the sample; from then on, the ith element from the stream is chosen with probability i/n. If the ith element is chosen, one of the elements from the current sample is evicted uniformly at random to keep the size of the sample constant at k. While straightforward to implement, random sampling has the disadvantage of needing a rather large sample to achieve expected approximation accuracy. Specifically, in order to estimate the quantiles with precision εn, with probability at least $1 - \delta$, a sample of size $\Theta(\frac{1}{\epsilon^2}\ \log\ \frac{1}{\delta})$ is required, where $0 < \delta < 1$.

In [3], Cormode and Muthukrishnan proposed a more space-efficient data structure, called Count-Min sketch, which is inspired by Bloom filters. Count-Min sketch allows ε-approximation of quantiles using $O(\frac{1}{\epsilon}\log^2 n \log(\frac{\log n}{\phi\delta}))$ memory. Although the space needed by Count-Min is worse than the two deterministic schemes discussed below, it has the advantage of allowing general updates to the streams: past elements can be deleted as well as their values updated.

Deterministic Algorithms

The first deterministic streaming algorithm for quantiles was proposed by Manku et al. [13,14], building on the prior work by Munro and Paterson [2]. This algorithm has space complexity $O(\frac{1}{\epsilon}\log^2 \epsilon n)$, meaning that using memory that grows poly-logarithmically in the stream size and inversely with the accuracy parameter ε, the quantiles can be estimated with precision εn. This result has since been improved by two groups: in [10], Greenwald and Khanna propose a $O(\frac{1}{\epsilon}\, log\ \epsilon n)$ memory scheme, and in [18], Shrivastava et al. propose a $O(\frac{1}{\epsilon}\log U)$ memory scheme, where U is the size of domain from which the input is drawn.

The Greenwald-Khanna (GK) algorithm is based on the idea that if a sorted subset $\{v_1, v_2, \ldots\}$ of the input stream S (of current size n) can be maintained such that the ranks of v_i and v_{i+1} are within 2ε of each other, then an arbitrary quantile query can be answered with precision εn. Their main contribution is to show how to maintain such a subset of values using a data structure of size $O(\frac{1}{\epsilon}\log \epsilon n)$. The Q-Digest scheme of Shrivastava et al. [18] approaches the quantile problem as a histogram problem over a universe of size U; thus $\log U$ is the number of bits needed to represent each element. Q-Digest maintains a set of buckets dynamically, merging those that are light (containing few items of the stream) and splitting those that are heavy, with an aim to keep the relative sizes of all the buckets nearly equal. Specifically, using a $O(\frac{1}{\epsilon}\log U)$ size data structure, Q-Digest ensures that the input stream is divided into $O(1/\epsilon)$ buckets, with each bucket containing $O(\epsilon n)$ items. Thus, the rank of any item can be determined with precision εn by locating its bucket.

In theoretical terms, the GK scheme has better performance when the input is drawn from a large universe, but the stream itself has only modest size. The Q-Digest, on the other hand, is superior when the stream size is huge but elements are drawn from a smaller universe. The GK algorithm is very clever, but requires a sophisticated analysis. The Q-Digest is simpler to understand, analyze, and implement, and it lends itself to easy extensions to distributed settings.

Practical Considerations

A detailed study of the empirical performance of quantile algorithms was carried out by Cormode et al. [5] on IP stream datasets. They concluded that with careful implementation, a commodity hardware machine (dual Pentium 2.8 GHz CPU and 4 GB RAM) can keep up with a 2 Gbit/s stream (310,000 packets per second). Performance numbers can depend also on the input distribution. For example, the deterministic algorithms presented above can have different memory usage and accuracy depending on the order in which the input values are presented, but sketching techniques such as the Count-Min sketch are not affected by the order of the input. The input value distribution can also impact perceived accuracy of the approximate quantiles. For example, for skewed distributions, the *numeric value* of the exact ϕ-th quantile can be arbitrarily far from the numeric values of the $(\phi \pm \epsilon)$-th quantile.

Extensions

Given the fundamental nature of quantiles and their widespread applications in data processing, it is no surprise that there are multiple extensions of the basic setting that have been considered so far. There are many interesting and practically-motivated applications, such as the latency of the web site mentioned earlier, where quantiles must be computed over distributed data, or over a sliding window portion of the stream etc. In the following, the current state of algorithms for these variants are briefly discussed.

Quantiles in Distributed Streams In many settings, data of interest are naturally distributed across multiple sources, such as servers in a web application and measurement devices in a sensor network. In these applications, it is necessary to compute the quantile summary of the entire data, but *without* creating a centralized repository of data, which could be undesirable because of the additional latency, communication overhead, or energy constraints of untethered sensors. The efficiency of an algorithm in this distributed setting is measured by the amount of information each node in the system must transmit during the computation.

One natural approach for distributed approximation of quantiles is for each node (server, sensor, etc.) to compute a local summary of its data, and arrange the nodes in a virtual hierarchy that guides them to merge these summaries into a final structure computed at the root of the hierarchy. The tree-based Q-digest [18] algorithm extends rather easily to the distributed setting, as the histogram boundaries of the Q-Digest are aligned to binary partition of the original value space U. The space complexity of the distributed version remains the same as the stream version, namely, $O(\frac{1}{\epsilon}\log U)$. The GK algorithm is little more complicated to extend to distributed streams, but Greenwald and Khanna themselves developed such an extension in [9]. However, the space complexity of their distributed data structure grows to $O(\frac{1}{\epsilon}\log^3 n)$ [9]. The Bloom filter based Count-Min sketch [3] also extends easily to the distributed setting also without any increase in the space complexity.

Quantiles in Sliding Windows In many applications, the user is primarily interested in the most recent portion of the data stream. This poses the *sliding window* extension of the quantiles problem, in which the desired quantile summary for the most recent N data elements – the window slides with the arrival of each new element, as the oldest element of the window is discarded and the new arrival added. In [12], Lin et al. presented such a sliding window scheme for quantile summaries, however, the space requirement for their algorithm is $O(\frac{1}{\epsilon^2} + \frac{1}{\epsilon}\log \epsilon^2 N)$. This was soon improved by Arasu and Manku [1] to $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\log N)$.

Biased Estimate of Quantiles The absolute measure of approximation precision is quite reasonable as long as the error εn is quite small compared to the rank of the quantile sought, namely, ϕn. This holds as long as the quantiles of interest are in the middle of the distribution. But if ϕ is either close to 0 or 1, one may prefer a *relative* error, so that the estimated quantile is in the range $[(1-\varepsilon)\phi, (1+\varepsilon)\phi]$. This variant was solved by Gupta and Zane [13] using random sampling techniques with a $O(\frac{1}{\epsilon^3}\log n \log\frac{1}{\delta})$ size data structure. The space bound has since been improved by Cormode et al. [4] to $O(\frac{1}{\epsilon}\log U \log(\epsilon n))$ using a deterministic algorithm.

Duplicate Insensitive Quantiles In some distributed settings, a single event can be observed multiple times. For example in the Internet, a single packet is observed at multiple routers. In wireless sensor networks, due to the broadcast nature of the medium, and to add fault-tolerance, data can be routed along multiple paths. Summaries such as quantiles or the number of distinct items are clearly not robust against duplication of data; on the other hand, simpler statistics such as minimum and maximum are not affected by duplication. Flajolet and Martin's distinct counting algorithm [8] is the seminal work in this direction. Cormode et al. have introduced algorithms based on sampling to compute various duplicate insensitive aggregates [6]. Their Count-Min sketch can be also easily adapted to compute duplicate insensitive quantiles.

Key Applications

Internet-scale network monitoring and database query optimization are two important applications that originally motivated the need for quantiles summaries over data streams. Gigascope [7] is a streaming database system that employs statistical summaries such as quantiles for monitoring network applications and systems. Quantile estimates are also widely used in query optimizers to estimate the size of intermediate results, and use those estimates to choose the best execution plan [5]. Distributed quantiles have been used to succinctly summarize the distribution of values occurring over a sensor network [8]. In a similar context, distributed quantiles are also used to summarize performance of websites and distributed applications [17].

Future Directions

The field of computing approximate quantiles over streams have led to a fertile research program and is expected to bring up new challenges in both theory and implementation. Although there is an obvious lower bound of $\Omega(\frac{1}{\epsilon})$ memory required to compute ε-approximate quantiles, there is no known non-trivial lower bound on memory. Since the current best algorithms requires $O(\frac{1}{\epsilon}\log(\epsilon n))$ or $O(\frac{1}{\epsilon}\log(U))$ memory, it will be useful to either lower the memory usage or prove a better lower bound.

Another direction in which improvements are highly desirable is running time. The current deterministic algorithms require amortized running time of $O(\log\frac{1}{\epsilon} + \log\log(\epsilon n))$ or $O(\log\frac{1}{\epsilon} + \log\log(U))$ per item. In high data-rate streams, even such low processing times are not fast enough: what is desired is a $O(1)$ insert time, or even a sublinear time quantile algorithm. As of now, there is no memory efficient sub-linear time quantile algorithm known except for random sampling.

Cross-references

- ▶ Adaptive Stream Processing
- ▶ Approximate Query Processing
- ▶ Continuous Query
- ▶ Data Aggregation in Sensor Networks
- ▶ Data Stream
- ▶ Distributed Data Streams
- ▶ Distributed Query Processing
- ▶ Geometric Stream Mining
- ▶ Hierarchical Heavy Hitter Mining on Streams
- ▶ In-Network Query Processing
- ▶ Stream Mining
- ▶ Stream Processing
- ▶ Streaming Applications

Recommended Reading

1. Arasu A. and Manku G.S. Approximate counts and quantiles over sliding windows. In Proc. 23rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2004, pp. 286–296.
2. Blum M., Floyd R., Pratt V., Rivest R., and Tarjan R.E. Time bounds for selection. J. Comput. Syst. Sci., 7:448–461, 1973.
3. Cormode G. and Muthukrishnan S. An improved data stream summary: the count-min sketch and its applications. J. Algorithms, 55(1):58–75, 2005.
4. Cormode G., Korn F., Muthukrishnan S., and Srivastava D. Space- and time-efficient deterministic algorithms for biased quantiles over data streams. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 263–272.
5. Cormode G., Korn F., Muthukrishnan S., Johnson T., Spatscheck O., and Srivastava D. Holistic UDAFs at streaming speeds. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 35–46.

6. Cormode G., Muthukrishnan S., and Zhuang W. What's different: distributed, continuous monitoring of duplicate-resilient aggregates on data streams. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 57.
7. Cranor C., Johnson T., Spataschek O., and Shkapenyuk V. Gigascope: a stream database for network applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 647–651.
8. Flajolet P. and Martin G.N. Probabilistic counting algorithms for data base applications. J. Comput. Syst. Sci., 31(2):182–209, 1985.
9. Greenwald J.M. and Khanna S. Power-conserving computation of order-statistics over sensor networks. In Proc. 23rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2004, pp. 275–285.
10. Greenwald J.M. and Khanna S. Space-efficient online computation of quantile summaries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 58–66.
11. Gupta A. and Zane F. Counting inversions in streams. In Proc. 14th Annual ACM-SIAM Symp. on Discrete Algorithms, 2003, pp. 253–254.
12. Lin X., Lu H., Xu J., and Yu J.X. Continuously maintaining quantile summaries of the most recent N elements over a data stream. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 362–374.
13. Manku G.S., Rajagopalan S., and Lindsay B.G. Random sampling techniques for space efficient online computation of order statistics of large datasets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 251–262.
14. Manku G.S., Rajagopalan S., and Lindsay B.G. Approximate medians and other quantiles in one pass and with limited memory. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 426–435.
15. Munro J.I. and Paterson M.S. Selection and sorting with limited storage. Theor. Comput. Sci., 12:315–323, 1980.
16. Paterson M.S., Progrees in Selection. In Proc. Scandinavian Workshop on Algortithm Theory, 1996, pp. 368–379.
17. Pike R., Dorward S., Griesemer R., and Quinlan S. Interpreting the data: parallel analysis with sawzall. Sci. Program. J., 13 (4):227–298, 2005.
18. Shrivastava N., Buragohain C., Agrawal D., and Suri S. Medians and beyond: new aggregation techniques for sensor networks. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 239–249.
19. Vitter J.S. Random sampling with a reservoir. ACM Trans. Math. Softw., 11(1):37–57, 1985.

Quantitative Association Rules

Xingquan Zhu
Florida Atlantic University, Boca Ration, FL, USA

Synonyms

Numeric association rules

Definition

Quantitative association rules refer to a special type of association rules in the form of $X \rightarrow Y$, with X and Y consisting of a set of numerical and/or categorical attributes. Different from general association rules where both the left-hand and the right-hand sides of the rule should be categorical (nominal or discrete) attributes, at least one attribute of the quantitative association rule (left or right) must involve a numerical attribute. Examples of this type of association rule can be categorized into the following two classes, depending on whether the rules are measured by the frequency of the supporting data records or by some distributional features of some numerical attributes.

1. *Frequent Rules:* Out of all applicants whose age is between 30 and 39 and marriage status is "yes," (→) 95% of them have two cars, and 10% applicants in the database satisfy this rule.
2. *Distributional Rules:* If the patient is non-smoker and wine-drinker then (→) his/her average life expectancy is 85 (whereas the average life expectancy in the overall population is 80).

In the above first example, "applicant age" is a numerical attribute, and "marriage status" and "# of cars" are categorical attributes. The support and confidence measures indicate that out of all the data records matching the left-hand of the rule, 95% of them satisfy the whole rule (*Confidence*) and there are 10% of the data records which hold this rule (Support). The key to discovering this type of rules is to discretize numerical attributes into discrete regions, so the frequency of the data records satisfying the rule can be measured and the general Apriori-based association rule mining approaches [1] can apply.

Distributional rules denote a set of quantitative association rules with instances covered by the rules rendering themselves statistically different from the overall population (with respect to some statistical measures, such as mean and standard deviation values). Because the calculation of the statistical values over numerical attributes is straightforward, attribute discretization is unnecessary, which consequently avoids possible information loss from the data discretization. The key to discovering this type of rules is to find a small group of instances (with respect to the left-hand side of the rules) statistically and significantly different from the overall population, if the right-hand of the rules is considered.

Historical Background

The problem of discovering quantitative association rule was first introduced by Srikant and Agrawal [4], and the main focus then was to discover frequent rules. The solution proposed in the paper [4] adopts a data discretization for mining association rules. Their approach first partitions the values of the numerical attributes into fine intervals and then combines adjacent intervals if necessary. A partial completeness measure was introduced to quantify the information loss due to the partitioning, so the algorithm can properly determine the number of partitions. Traditionally, numerical attribute discretization can be achieved through the following two approaches, with the latter preferred in practice:

- *Equal-width discretization* divides the range of a numerical attribute into N intervals of equal width. For example, ages from 20 to 60 can be divided into four intervals of 10-year width. This method can be easily implemented, but subject to a clear drawback that there may be too few instances in some intervals and too many in other intervals, and both cases hinder mining high quality association rules.
- *Equal-depth discretization* divides the range of numerical attribute into N intervals such that each interval equally contains $1/N$ of the total instances. This method avoids the possible imbalance inherent in the equal-width discretization method, but it may separate similar attribute values into different intervals and group dissimilar attributes into the same interval.

Noticing the adoption of the information discretization can lead to unavoidable information loss or misleading association rules, Aumann and Lindell [2] suggested distributional quantitative association rules, where the general structure of the rules take the form of:

$$\text{population} - \text{subset} \rightarrow \text{interesting} - \text{behavior}$$

More specifically, the left-hand of the rule confines a population subset, whereas the right-hand of the rule provides the statistical measures of the confined subset. The objective is to discover subsets significantly different from the whole population (with respect to the underlying measures, e.g., mean or standard deviation values). Following this principle, Webb [5] extended the framework to include other measures such as the minimum and the count measures.

Foundations

Frequent Rules

To demonstrate the procedures of the frequent quantitative association rule discovery, take the toy database in Fig. 1(a) as an example [4], where attribute "Age" is numerical, and "Married" and "NumCars" are categorical attributes. Assuming the user specified *Support* and *Confidence* values are 40% and 50% respectively, it means that a prospective rule (the left-hand and the right-hand together) should cover at least two records (5 × 40%). For all data records satisfying the left-hand of the rule, 50% of them should also contains the right-hand sides of the rule ($X \rightarrow Y$). The major steps of discovering frequent quantitative association rules can then be summarized as follows [4]:

1. Determining the partition numbers and the region of partitioning for each numerical attribute. E.g., Fig. 1(a) lists four partitions for "Age" (denoted by [20,24], [25,29], [30,34], and [35,39]), with each region mapping to one integer value {1,2,3,4}.
2. Applying each mapping table to all records of the corresponding numerical attribute, with numerical values replaced by the matching integer values. The example of the mapped database is shown in Fig. 1(c).
3. Generating frequent itemsets based on the mapped database and the user specified *Support* value (the existing Apriori like association rule mining methods can be applied directly).
4. Using discovered frequent itemsets to generate quantitative association rules, with each frequent itemset decomposed into two (left- and right-hand) components. For example, if itemset "ABCD" is found frequent, a possible quantitative association rule can be made by decomposing "ABCD" as "AB" → "CD." As long as the validate check asserts that confidence value of this rule ("AB" → "CD") is greater than the user specified value (*Confidence*), the rule is taken as a valid rule.
5. Collect all quantitative association rules generated from the above process and prune redundant rules. *E.g.*, if "AB" → "CD" and "AB" → "CDE" are both valid rules, "AB" → "CD" can be pruned as it can be generalized (inferred) from "AB" → "CDE".

a

People

RecordID	Age	Married	Numcars
100	23	No	1
200	25	Yes	1
300	29	No	0
400	34	Yes	2
500	38	Yes	2

b

Mapping age

Interval	Integer
20..24	1
25..29	2
30..34	3
35..39	4

c

After mapping attributes

RecordID	Age	Married	Numcars
100	1	2	0
200	2	1	1
300	2	2	1
400	3	1	2
500	4	1	2

d

Frequent itemsets: sample

Itemsets	Support
{ ⟨Age: 20..29⟩ }	3
{ ⟨Age: 30..39⟩ }	2
{ ⟨Married: Yes⟩ }	3
{ ⟨Married: No⟩ }	2
{ ⟨Numcars: 0..1⟩ }	3
{ ⟨Age: 30..39⟩, ⟨Married: Yes⟩ }	2

e

Rules: sample

Rule	Support	Confidence
⟨Age: 30..39⟩ and ⟨Married: Yes⟩ ⇒ ⟨Numcars: 2⟩	40%	100%
⟨Age: 20..29⟩ ⇒ ⟨Numcars: 0.1⟩	60%	66.6%

Quantitative Association Rules. Figure 1. Example of problem decomposition for quantitative association rule mining (revised from [2]).

Distributional Rules

Different from the frequent quantitative association rules, where the main challenge is to determine the "optimum" number of partitions and the region of partitioning, the distributional rules represent a set of quantitative association rules, where the statistical features of the samples covered by the rule are different from the whole population. Similar to general association rules, the distributional rules also contain the left- and the right-hand. The left-hand side of the rule is a description of a subset of the database, while the right-hand side provides a description of outstanding behaviors of this subset. A rule is only interesting if the mean for the subset (specified by the left-hand side) is significantly different from the rest and is therefore unexpected. Consider the following distributional rule, the left-hand side consists of two categorical attributes (Non-smoker: {Yes or No}, and Wine-drinker: {Yes or No}), and the right-hand side is a numerical attribute (life expectancy). The rule is considered informative and meaningful as it indicates that individuals characterized by the left-hand side of the rule (Non-smoker and wine-drinker) have a longer average life expectancy (85) than the whole population (80).

$$\begin{aligned}&\text{Non} - \text{smoker and wine} - \text{drinker}\\&\quad \rightarrow \text{life expectancy} = 85\ (\text{overall} = 80)\end{aligned}$$

Following this definition, one can easily extend the framework to allow one or multiple numerical attributes to appear on the left- or right-hand sides of the rule (or both), or employ other statistical measures rather than the mean values to assess the rules.

In order to discover distributional rules, Aumann and Lindell [3] proposed two methods to discover the following two types of rules:

- $X \rightarrow Mean_J\ (T_X)$, where X and J denote a single numerical attribute, T_X denotes transactions confined by attribute X and $\text{Mean}_J(T_X)$ represents the mean value of attribute J (for all samples in T_X).
- $X \rightarrow M_J\ (T_X)$, where X denotes one or multiple categorical attributes, J consists of one or multiple quantitative attributes, and M means one particular statistical measure (there is no restriction on the number of attributes in X and J).

The solution to the first type of distributional rules is straightforward, since the rules only involve two numerical attributes (one on each side), an algorithm can afford to go through each pair of numerical attributes to discover meaningful rules. More specifically, for any two numerical attributes *i* and *j*, one can first sort all records in the database based on the attribute *i*, then any above or below average continuous region of values in *j* can form a prospective quantitative rule. For example, given the toy database in Fig. 2(a) which records the age and the size of the striped bass, the sorted database with respect to attribute *i* (age) is given in Fig. 2(a). The average of attribute *j* for the top three records (001, 003, and 002) is 0.77, which is significantly lower than the mean of the whole population (1.83). Therefore, the below average region (001, 003, and 002) forms a meaningful quantitative association rule denoted by:

$$\text{Age} \leq 2 \rightarrow \text{Mean weight } 0.77 \text{ lbs (Overall } 1.83)$$

Because any above or below average continuous region of values in *j* can form a prospective quantitative rule, one can continuously span the region with respect to the attribute *j*, and discover the maximum region satisfying the user specified requirements (e.g., α times less/larger than the average).

In order to discover the second type of distributional quantitative association rules, one can employ a two-stage approach, which applies general association rule to the whole database by considering categorical attributes only, followed by a refining process to check each rule by considering the numerical attribute values [5].

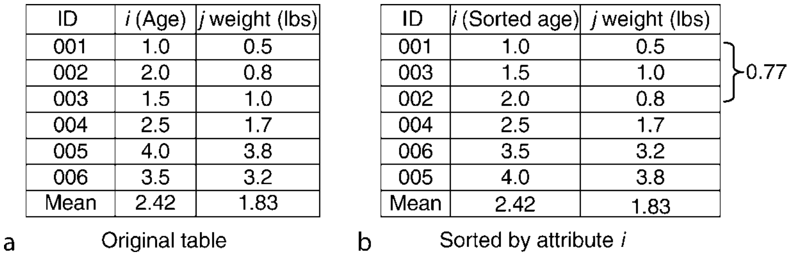
1. *Discovering frequent itemsets:* Finding all frequent itemsets by considering categorical attributes of the database only (this can be easily achieved through existing Apriori-like algorithms [1].
2. *Calculating statistical distribution values:* For each numerical attribute, calculate the value of the distribution measures (mean/variance) over samples confined by each frequent itemset. For example, if "Non-smoker and Wine-drinker" are found frequent (i.e.,, a frequent itemset), all samples matching this itemset form a sample set *P*, from which the statistical distribution value of a numerical attribute can be calculated.
3. *Refining quantitative association rules:* For every frequent itemset (denoted by *X*) and one numerical attribute *e*, the algorithm continuously check if $X \rightarrow Mean_e\ (T_X)$ and $X \rightarrow Variance_e\ (T_X)$ are meaningful rule (comparing to the whole population). In addition, for any two rules $X \rightarrow Mean_e\ (T_X)$ and $Y \rightarrow Mean_e\ (T_Y)$, the algorithm will check whether the former is a sub-rule of the latter, or vice versa, such that the algorithm can output compact rules with minimum redundancy.

Key Applications

Business intelligence, market basket analysis, fraud detection (fraud medical insurance claims).

Future Directions

All of the above techniques intend to discover quantitative association rules in the forms of the conjunction of individual attributes. One interesting problem is to find quantitative association rules with (linearly or nonlinearly) combined attributes. For example, finding rules like "Age $\leq \alpha \rightarrow$ Length/Weight $\leq \beta$." Here the right-hand of the rule is a non-linear combination of the numerical attributes (Length and Height), and α and β are some discovered values. In [3], *Ruckert* et al. proposed a quantitative association rule mining approach which is able to discover similar rules with linearly weighted attributes like "Age $\leq \alpha \rightarrow \alpha_1$·Length + α_2·Weight $\leq \beta$." Future research may emphasize the generalized quantitative association



ID	*i* (Age)	*j* weight (lbs)
001	1.0	0.5
002	2.0	0.8
003	1.5	1.0
004	2.5	1.7
005	4.0	3.8
006	3.5	3.2
Mean	2.42	1.83

a Original table

ID	*i* (Sorted age)	*j* weight (lbs)
001	1.0	0.5
003	1.5	1.0
002	2.0	0.8
004	2.5	1.7
006	3.5	3.2
005	4.0	3.8
Mean	2.42	1.83

b Sorted by attribute *i*

Quantitative Association Rules. Figure 2. Example of distributional association rule mining (age and weight of the striped bass).

rule discovery, where rules consist of non-linearly combined attributes.

Another interesting problem concerning quantitative association rules is to discover relational patterns of the quantitative association rules across multiple databases. *E.g.*, Finding patterns that are frequent with a support level of α in database A, but significantly infrequent with a support level of β in databases B or/and C. In [6], Zhu and Wu proposed a hybrid frequent pattern tree based solution to address this problem with a focus on the general association rules. Extending the problem of relational frequent pattern discovery to quantitative association rules is another interesting topic for future research.

Cross-references

▶ Association Rules

Recommended Reading

1. Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 207–216.
2. Aumann Y. and Lindell Y. A statistical theory for quantitative association rules. In Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 261–270.
3. Ruckert U., Richter L., and Kramer S. Quantitative association rules based on half-spaces: an optimization approach. In Proc. 2004 IEEE Int. Conf. on Data Mining, 2004, pp. 507–510.
4. Srikant R. and Agrawal R. Mining quantitative association rules in large relational tables. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 1–12.
5. Webb G.I. Discovering associations with numeric variables. In Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2001, pp. 383–388.
6. Zhu X. and Wu X. Discovering relational patterns across multiple databases. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 726–735.

QUEL

Tore Risch
Uppsala University, Uppsala, Sweden

Definition

QUEL was the query language used in the original Ingres system from Berkeley University.

Key Points

QUEL was one of the first relational database query languages. It can be seen as a syntactically sugared tuple relational calculus language. The Postgres extension of Ingres originally used an extention of QUEL called PostQUEL, but was later replaced with SQL.

Cross-references

▶ Ingres
▶ Postgres
▶ Tuple Relational Calculus

Query Answering in Analytical Domains

▶ Query Processing in Data Warehouses

Query Assistance

▶ Web Search Query Rewriting

Query by Example

▶ Video Querying

Query by Humming

Yingyi Bu[1], Raymond Chi-Wing Wong[2],
Ada Wai-Chee Fu[1]
[1]Chinese University of Hong Kong, Hong Kong, China
[2]Hong Kong University of Science and Technology, Hong Kong, China

Synonyms

Music retrieval; Time series database querying

Definition

With the appearance of large scale audio and video databases in various application areas, novel information retrieval methods adapted to the specific characteristics of these data types are required. A natural way of searching in a musical audio database is by humming the tune of a song as a query, which is so-called "query by humming". In this entry, state-of-the-art techniques for effective and efficient querying by humming are described.

Historical Background

In 1995, Asif Ghias et al. [1] proposed the basic architecture for a system supporting query by humming. Three main components are introduced in the system: a pitch-tracking module, a melody database, and a query engine. Queries are hummed into a microphone, digitized, and fed into a pitch-tracking module. Then, a symbol sequence representation upon the relative pitch transitions of the hummed melody is sent to the query engine, which produces a ranked list of matching melodies.

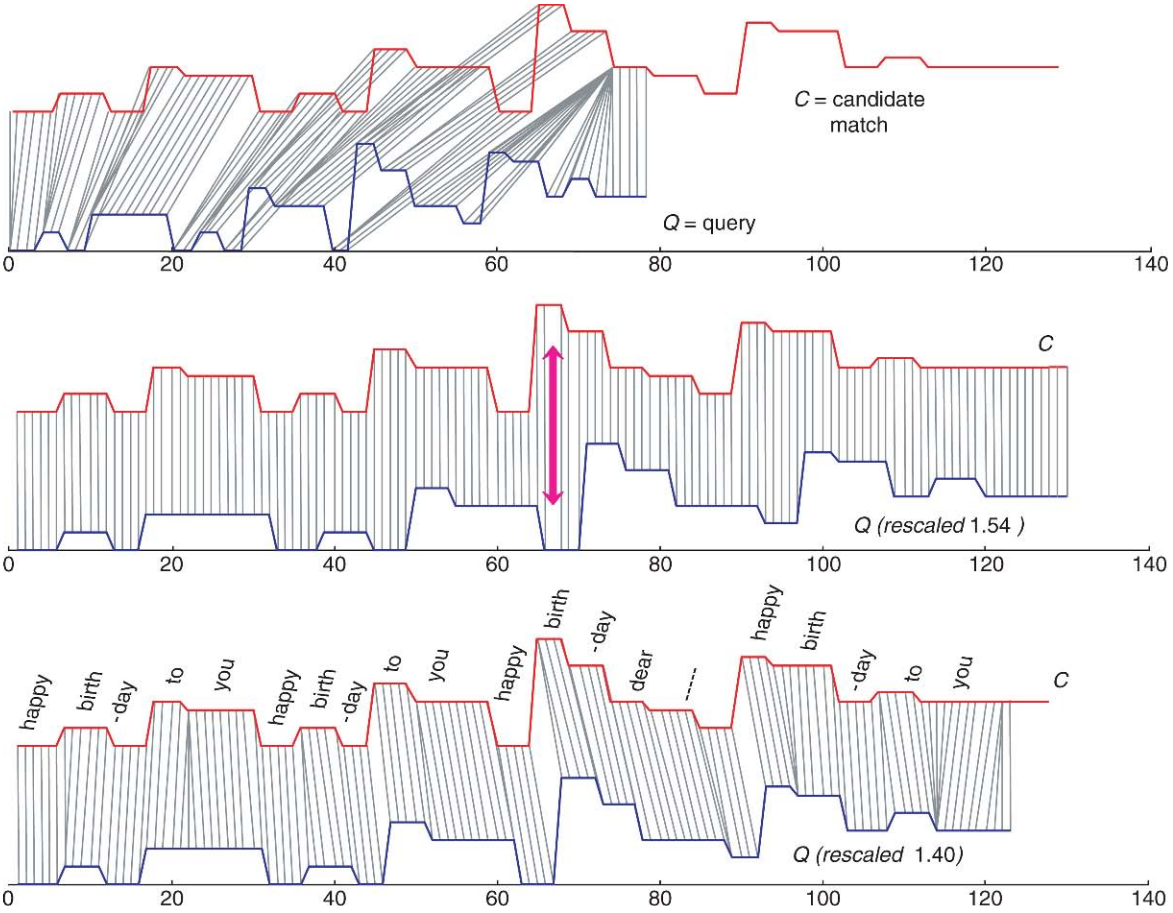
Foundations

In recent researches, musical data are modeled as time series which are real valued sequences rather than symbol sequences. In speech comparisons, small fluctuation of the tempo of the speaker could be allowed in order to identify similar contents. There have been some works that match a melody more effectively by considering warping and scaling in humming queries. This generally gives better query results because it is free from the error-prone note segmentation. However, those works rely on distance measures such as universal scaling (US), dynamic time warping (DTW) [2,4] and scaling and time warping (SWM) [5], the efficiency might be rather poor. Fortunately, tight lower bounds for DTW and SWM could greatly improve the efficiency by pruning large portions of non-candidate data at an early stage.

Comparisons of Distance Measures on Examples

Figure 1 demonstrates the effects of different distance measures with a typical piece of music, *Happy Birthday to You*, from top to bottom:

1. Since the query sequence is performed at a much faster tempo, direct application of DTW fails to produce an intuitive alignment;



Query by Humming. Figure 1. Motivating example.

2. Rescaling the shorter performance by a scaling factor of 1.54 seems to improve the alignment, but the higher pitched note produced on the third "*birth...*" of the candidate is forced to align with the lower note of the third "*happy...*" in the query;
3. Only the application of *both* uniform scaling and DTW produces the appropriate alignment.

Dynamic Time Warping (DTW)

Intuitively, *dynamic time warping* is a distance measure that allows time series to be *locally* stretched or shrunk before the base distance measure is applied. Given two sequences $C = C_1, C_2,...,C_n$ and $Q = Q_1, Q_2,...,Q_m$, the time warping distance DTW is defined recursively as follows:

$$\begin{aligned}
\text{DTW}(\phi,\phi) &= 0\\
\text{DTW}(C,\phi) = \text{DTW}(\phi,Q) &= \infty\\
\text{DTW}(C,Q) &= D_{base}(\text{First}(C),\text{First}(Q))+\\
&\min\begin{cases}\text{DTW}(C,\ \text{Rest}(Q))\\ \text{DTW}\ (\text{Rest}(C),\ Q)\\ \text{DTW}\ (\text{Rest}(C),\text{Rest}(Q))\end{cases}
\end{aligned}$$

where Φ is the empty sequence, $\text{First}(C) = C_1$, $\text{Rest}(C) = C_2, C_3,...,C_n$, and D_{base} denotes the distance between two entries. Several L_p measures were used as the D_{base} distance in previous literature, such as Manhattan Distance (L_1), squared Euclidean Distance (L_2) and maximum difference (L_∞). Typically *Squared Euclidean Distance* is used as the D_{base} measure. That is,

$$D_{base}(C_i, Q_j) = (C_i - Q_j)^2.$$

Thus in the following parts, without loss of generality, it is assumed that D_{base} is the squared Euclidean Distance and D is also used to denote it. However, the time complexity of DTW distance calculation is *O(mn)*, and intensive computations are employed for the corresponding dynamic programming. Thus, lower bounds on the distance are adopted to effectively prune the search space and support efficient search.

Constraints and Lower Bounds on Dynamic Time Warping

Keogh et al. [2] viewed a global constraint as a constraint on the warping path entry $w_k = (i, j)_k$ and gave a general form of global constraints in terms of inequalities on the indices to the elements in the warping matrix,

$$j - r \le i \le j + r$$

where r is a constant for the Sakoe–Chiba Band and r is a function of i for the Itakura Parallelogram. Incorporating the global constraint into the definition of dynamic time warping distance, DTW can be modified as follows.

Given two sequences $C = C_1,C_2,...,C_n$ and $Q = Q_1, Q_2,...,Q_m$, and the time warping constraint r, the constrained time warping distance cDTW is defined recursively as follows:

$$\begin{aligned}
Dist_r\,(C_i, Q_j) &= \begin{cases}D_{base}\,(C_i, Q_j) & if|\ i\ -\ j\ |\ \le r\\ \infty & otherwise\end{cases}\\
{}_{C}\text{DTW}(\phi,\phi,r) &= 0\\
{}_{C}\text{DTW}(C,\phi,r) &= {}_{C}\text{DTW}(\phi,\ Q,\ r) = \infty\\
{}_{C}\text{DTW}(C,Q,r) &= Dist_r(\text{First}(C),\ \text{First}(Q))+\\
&\min\begin{cases}{}_{C}\text{DTW}((C,\ \text{Rest}(Q),r)\\ {}_{C}\text{DTW}(\ (\text{Rest}(C),\ Q,\ r)\\ {}_{C}\text{DTW}((\text{Rest}(C),\ \text{Rest}(Q),r)\end{cases}
\end{aligned}$$

where Φ is the empty sequence, $\text{First}(C) = C_1$, $\text{Rest}(C) = C_2, C_3,...,C_n$. The *upper bounding sequence UW* and the *lower bounding sequence LW* of a sequence C are defined using the time warping constraint r as follows.

Let $UW = UW_1, UW_2,...,UW_n$ and $LW = LW_1, LW_2,...,LW_n$,

$$\begin{aligned}
UW_i &= \max(C_{i-r},...,C_{i+r}) \text{ and}\\
LW_i &= \max(C_{i-r},...,C_{i+r})
\end{aligned}$$

Considering the boundary cases, the above can be rewritten as

$$\begin{aligned}
UW_i &= \max(C_{\max(1,i-r)},...,C_{\min(i+r,n)}) \text{ and}\\
LW_i &= \min(C_{\min(1,i-r)},...,C_{\min(i+r,n)})
\end{aligned}$$

$E(C) = < UW, LW >$ is called the envelope sequences of Keogh et al. [2] propose the lower bound distance LB_Keogh based on envelope sequences. The time warping distance between two sequences Q and C is lower bounded by the squared Euclidean distance between Q and the envelope sequences of C. Equation (1) below formally defines the lower bounding distance.

$$\begin{aligned}
\text{LB\_Keogh}(Q,\ C) &= D\,(Q,\ E(C))\\
&= \sum_{i=1}^{m}\begin{cases}(Q_i - UW_i)^2\ if & Q_i > UW_i\\ (Qi - LW_i)^2\ if & Q_i < LW_i\\ 0 & \text{otherwise}\end{cases}
\end{aligned} \qquad (1)$$

Zhu et al. [3] further improve on LB_Keogh. If a transformation T is a linear transform and lower-bounding, and $Env_r(C_i)$ is the envelope of C_i by global constraint r then

$$D(T(Q),\ T(Env_r(C_i))) \leq \mathrm{cDTW}(\mathrm{Q},\ \mathrm{C_i}\ ,\ \mathrm{r}) \quad (2)$$

Therefore transforms such as PAA, DWT, SVD and DFT on the envelope sequence of a candidate sequence could still lower bound DTW distance, since those transforms are both linear and lower bounding.

Uniform Scaling (US)

Given two sequences $Q = Q_1,\ldots,Q_m$ and $C = C_1,\ldots,C_n$ and a scaling factor bound $l, l \geq 1$. Let $C(q)$ be the prefix of C of length q, where $\lceil m/l \rceil \leq q \leq lm$ and $C(m,q)$ be a rescaled version of $C(q)$ of length m,

$$C(m,\ q)_i\ =\ C(q)_{\ \lceil i.q/m \rceil} \text{ where } 1\ \leq\ \mathrm{i}\ \leq\ \mathrm{m}$$

$$\mathrm{US}(C, Q, l) = \min_{\mathrm{q}=\lceil m/l \rceil}^{min(lm,n)} D(C(m,q), Q)$$

where D(X, Y) denotes the Euclidean distance between two sequences X and Y.

Lower Bounding Uniform Scaling

The two sequences $UC = UC_1,\ldots,UC_m$ and $LC = LC_1,\ldots, LC_m$, such that

$$UC_i = \max(C_{\lceil i/l \rceil},\ldots,C_{\lceil il \rceil})$$
$$LC_i = \min(C_{\lceil i/l \rceil}),\ldots,C_{\lceil il \rceil})$$

bound the points of the time series C that can be matched with Q. The lower bounding function, which lower bounds the distance between Q and C for any scaling ρ, $1 \leq \rho \leq l$, can now be defined as:

$$LB_s(Q, C) = \sum_{i=1}^{m} \begin{cases} (Q_i - UC_i)^2 & \text{if } Q_i > UC_i \\ (Q_i - LC_i)^2 & \text{if } Q_i < UC_i \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Scaling and Time Warping (SWM)

Having reviewed time warping, uniform scaling, and lower bounding, this part introduces *scaling and time warping* (SWM) distance. Given two sequences $Q = Q_1,\ldots,Q_m$ and $C = C_1,\ldots,C_n$, a bound on the scaling factor $l, l \geq 1$ and the Sakoe–Chiba Band time warping constraint r which applies to sequence length m. Let $C(q)$ be the prefix of C of length q, where $\lceil m/l \rceil \leq q \leq \min(lm, n)$ and $C(m, q)$ be a rescaled version of $C(q)$ of length m,

$$\mathrm{C}(m, q)i = C(q)_{\lceil i.q/m \rceil} \text{ where } 1 \leq i \leq m$$

$$\mathrm{SWM(C, Q, l, r)} = \min_{\mathrm{q=\lceil m/l \rceil}}^{\mathrm{min(lm,n)}} \mathrm{cDTW(C(m, q), Q, r)}$$

If time warping is applied on top of scaling, i.e., the sequence is first scaled, and then measure the time warping distance of the scaled sequence with the query. Typically, time warping with Sakoe–Chiba Band constrains the warping path by a fraction of the data length, which is translated into a constant r. Hence, if the fraction is 10%, then $r = 0.1|C|$. If the length of C is changed according to the scaling fraction ρ, that is, C is changed to ρC, then the Sakoe–Chiba Band time warping constraint is $r = 0.1|\rho C|$. Hence, $r = r'\rho$, where r' is the Sakoe–Chiba Band time warping constraint on the unscaled sequence, and ρ is the scaling factor.

Lower Bounding SWM

The lower envelope L_i and upper envelope U_i on C can be deduced as follows: recall that the upper and lower bounds for uniform scaling between $1/l$ and l is given by the following:

$$UC_i = \max(C_{\lceil i/l \rceil},\ldots,C_{\lceil il \rceil})$$
$$LC_i = \min(C_{\lceil i/l \rceil},\ldots,C_{\lceil il \rceil})$$

and the upper and lower bounds for a Sakoe –Chiba Band time warping constraint factor of r for a point C_i is given by:

$$UW_i = \max(C_{\max(1,i-r)},\ldots,C_{\min(i+r,n)})$$
$$LW_i = \min(C_{\max(1,i-r)},\ldots,C_{\min(i+r,n)})$$

Therefore, when time warping is applied on top of scaling the upper and lower bounds will be:

$$\begin{aligned} U_i &= \max(\mathrm{UW}_{\lceil i/l \rceil},\ldots,UW_{\lceil il \rceil}) \\ &= \max(C_{\max(1,\lceil i/l \rceil - r')},\ldots,C_{\min(\lceil i/l \rceil + r',n)},\ldots, \\ &\qquad C_{\max(1,\lceil il \rceil - r')},\ldots,C_{\min(\lceil il \rceil + r',n)}) \\ &= \max(C_{\max(1,\lceil i/l \rceil - r')},\ldots,C_{\min(\lceil il \rceil + r',n)}) \end{aligned} \quad (4)$$

$$\begin{aligned} L_i &= \min(\mathrm{LW}_{\lceil i/l \rceil},\ldots,LM_{\lceil il \rceil}) \\ &= \min(C_{\max(1,\lceil i/l \rceil - r')},\ldots,C_{\min(\lceil i/l \rceil + r',n)},\ldots, \\ &\qquad C_{\max(1,\lceil il \rceil - r')},\ldots,C_{\min(\lceil il \rceil + r',n)}) \\ &= \min(C_{\max(1,\lceil i/l \rceil - r')},\ldots,C_{\min(\lceil il \rceil + r',n)}) \end{aligned} \quad (5)$$

In [4], the lower bound function which lower bounds the distance between Q and C for any scaling in the range of {1/*l*,*l*} and time warping with the Sakoe–Chiba Band constraint factor of r' on C is given by:

$$LB(Q,C) = \sum_{i-1}^{m} \begin{cases} (Q_i - U_i)^2 & \text{if } Q_i > U_i \\ (Q_i - L_i)^2 & \text{if } Q_i < L_i \\ 0 & \text{otherwise} \end{cases} \qquad (6)$$

Efficient Pruning Algorithm by Lower Bounds

Algorithm 1 gives the pseudocode for the search algorithm, which utilizes the computational efficient lower bounds on computational intensive distance measures to prune candidate sequences at an early stage. “real_distance” could be DTW, US, or SWM distance, while “lower_bound_distance” denotes the corresponding lower bound.

```
Algorithm 1: Lower_Bounding_Sequential_Scan(Q)
best_so_far = infinity;
for each sequence with index i in database do
    LB_dist = lower_bound_distance($C_i$, $Q$);
    if LB_dist < best_so_far then
        true_dist = real_distance($C_i$, $Q$);
        if true dist < best_so_far then
            best_so_far = true_dist;
            index_of_best_match = $i$;
        end
    end
end
```

Key Applications

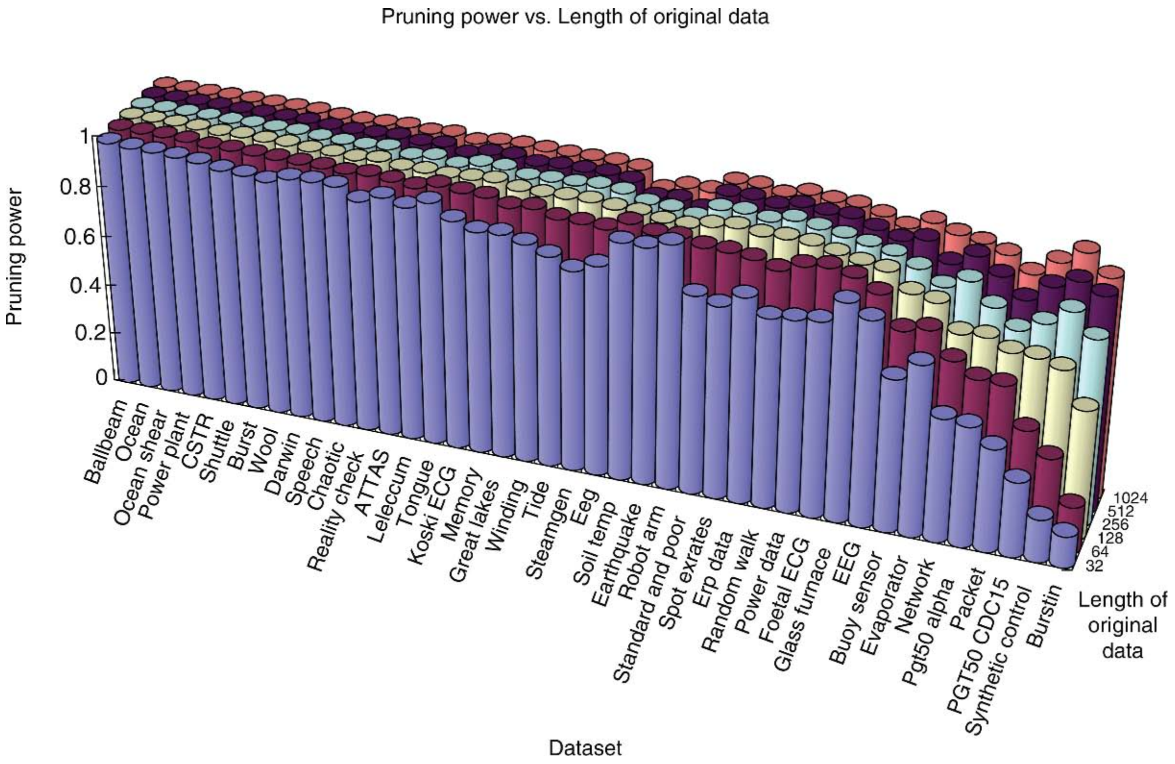
Query by humming is essential for audio information retrieval in terms of both effectiveness and efficiency.

Future Directions

In [3], a unified framework is proposed to explain the existing lower-bound functions for dynamic time warping distance. A new lower-bound function that is shown by experiments to be superior is also proposed. For future studies, this function can be investigated for the effectiveness in query by humming.

Experimental Results

This section describes the experiments carried out to verify the effectiveness of the proposed lower bounding



Query by Humming. Figure 2. Pruning power vs. length of original data.

distance for the most effective distance measure: SWM. The *Pruning Power P* is defined in [2] as follows:

$$p = \frac{\text{Number of objects that do not require full SWM}}{\text{Number of objects in database}}$$

The pruning power is an objective measure because it is free of implementation bias and choice of the underlying spatial index. This measure has become a common metric for evaluating the efficiency of lower bounding distances, therefore, it was adopted in evaluating the proposed lower bounding distance.

Figure 2 shows how the pruning power of the lower bounding measure varies as the length of data changes for different datasets. More than 78% (32 out of 41) of the datasets achieved a pruning power above 90%.

Data Sets

http://www.cs.ucr.edu/∼eamonn/VLDB2005/

Cross-references

- ▶ Multimedia Information Retrieval
- ▶ Spatial Network Databases

Recommended Reading

1. Ghias A., Logan J., Chamberlin D., and Smith B.C. Query by humming: musical information retrieval in an audio database. In Proc. 3rd ACM Int. Conf. on Multimedia, 1995, pp. 231–236.
2. Keogh E.J. Exact indexing of dynamic time warping. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 406–417.
3. Zhou M. and Hon Wong M. Boundary-based lower-bound functions for dynamic time warping and their indexing. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 1307–1311.
4. Zhu Y. and Shasha D. Warping indexes with envelope transforms for query by humming. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 181–192.
5. Wai-Chee Fu A., Keogh E.J., Yung Hang Lau L., and Chotirat (Ann) Ratanamahatana. Scaling and time warping in time series querying. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 649–660.

Query Compilation

- ▶ Query Optimization
- ▶ Query Optimization (in Relational Databases)

Query Compilation and Execution

- ▶ Query Processing (In Relational Databases)

Query Containment

Rada Chirkova
North Carolina State University, Raleigh, NC, USA

Definition

One query is contained in another if, independent of the values of the "stored data" (that is, database), the set of answers to the first query on the database is a subset of the set of answers to the second query on the same database. A formal definition of containment is as follows: denote with $Q(D)$ the result of computing query Q over database D. A query Q_1 is said to be contained in a query Q_2, denoted by $Q_1 \sqsubseteq Q_2$, if for all databases D, the set of tuples $Q_1(D)$ is a subset of the set of tuples $Q_2(D)$ – that is, $Q_1(D) \subseteq Q_2(D)$. This definition of containment, as well as the related definition of query equivalence, can be used to specify query containment and equivalence on databases conforming to both relational and nonrelational data models, including XML and object-oriented databases.

Historical Background

Testing for query containment on finite databases is, in general, co-recursively enumerable: The procedure is going through all possible databases and simultaneously checking for noncontainment via bottom-up evaluation. See [6] for the details and for a discussion of the relationship between regular and unrestricted (that is, on not necessarily finite databases) containment and equivalence of relational expressions.

Chandra and Merlin [3] have shown that the problems of containment, equivalence, and minimization of conjunctive queries are NP complete. (Conjunctive queries are a subset of Datalog that is equivalent in expressive power to SQL select-project-join queries with only equality comparisons permitted.) It is also shown [3] that there is a simple test for containment, and thus for equivalence. While the question of whether one conjunctive query is contained in another is NP complete, all the complexity is caused by "repeated predicates," that is, predicates appearing three or more times in the body. In the very common case that no predicate appears more than twice in any query, containment can be tested in linear time [11]. Moreover, conjunctive queries tend to be short, so in practice containment testing is not likely to be too inefficient.

Klug [8] has shown that containment for conjunctive queries with arithmetic – that is, inequality or disequality – comparisons (CQAC) is in $\Pi_2{}^P$, while being in NP for some proper subclasses. In addition to reporting some new results on the complexity of CQAC query containment, Afrati and colleagues [2] provide a survey, comprehensive as of 2006, on the complexity of query containment, for query classes including conjunctive queries, CQAC and its subclasses, as well as recursive and nonrecursive Datalog. More original work and further references on query containment in relational and nonrelational databases can be found in [1,5,7,9,10,12–15].

Foundations

This overview of key ideas concerning conjunctive queries, Datalog programs, and their containment is based on [9] as well as on [15], which includes the details and references to the original sources of the results discussed here. The first item on the agenda is a review of containment of conjunctive queries. A conjunctive query is a single Datalog rule with subgoals that are assumed to have predicates referring to stored relations. (The standard Datalog notation is reviewed in, e.g., Sect. 6.1 of [4].) A conjunctive query is applied to the stored relations in a given database by considering all possible substitutions of values for the variables in the body. If a substitution makes all the subgoals true, then the same substitution, applied to the head of the rule, is an element of the set of answers to the head's predicate on the given database.

For example, rule

$$p(X,Z) \ :- \ a(X,Y), \ a(Y,Z)$$

talks about predicate a, for stored relation that contains information about arcs in a directed graph: $a(X, Y)$ means that there is an arc from node X to node Y in the graph. The rule also talks about predicate p whose relation is constructed by the rule. The rule says "$p(X, Z)$ is true if there is an arc from node X to some node Y and also an arc from Y to Z." That is, conjunctive query p represents paths of length 2, in the sense that $p(X, Z)$ will be inferred exactly when there is a path of length 2 from X to Z in the graph represented by stored relation a.

It was proved in [3] that $Q_1 \sqsubseteq Q_2$ if and only if there is a homomorphism $h : Q_2{}^D \rightarrow Q_1{}^D$, where Q^D is the canonical database associated with conjunctive query Q. The canonical database Q^D for query Q is defined as the result of "freezing" the body of Q, which turns each subgoal of Q into a fact in the database. That is, the "freezing" procedure replaces each variable in the body of Q by a distinct constant, and the resulting subgoals are the only tuples in the canonical database Q^D.

Consider now an example of checking conjunctive-query containment using canonical databases. Rule

$$r(W,W) \ :- \ a(W,U), \ a(U,W)$$

defines conjunctive query r, whose answer represents circular paths of length 2 in a graph specified by arcs stored in relation a. As each circular path of length 2 is also an (arbitrary) path of length 2, the containment $r \sqsubseteq p$ is expected to hold. Indeed, the canonical database r^D, for the query defining predicate r, is a set of tuples $\{a(w, u), a(u, w)\}$, while the canonical database p^D, for the query defining predicate p, is a set of tuples $\{a(x, y), a(y, z)\}$. There exists a homomorphism from p^D to r^D that maps x into w, y into u, and z into w (or, alternatively, maps $a(x, y)$ into $a(w, u)$ and $a(y, z)$ into $a(u, w)$). Thus, by [3], the set of tuples in the answer to r on every database D is a subset of the set of tuples in the answer to p on D.

As no homomorphism exists from r^D to p^D, the conclusion is that p is not contained in r, which is to be expected from an intuitive interpretation of the two queries. However, if conjunctive query t defined by the rule

$$t(L,N) \ :- \ a(L,M), \ a(M,N), \ a(M,S)$$

is also considered, it is possible to ascertain both $p \sqsubseteq t$ and $t \sqsubseteq p$, by constructing a homomorphism from p^D to t^D (for the query defining predicate t) and another homomorphism from t^D to p^D. Thus, conjunctive queries p and t are *equivalent.* Indeed, each of p and t represents arbitrary paths of length 2 in a graph. However, p is more efficient to evaluate than t, because computing the set of answers to p requires only one join on the stored relation a, while evaluating t requires two joins. As described in [3], one can *minimize* conjunctive queries for more efficient evaluation; in fact, query p in this example can be obtained as a result of minimizing query t.

Another test for containment of conjunctive query Q_1 in conjunctive query Q_2 consists in computing the set of answers to Q_2 on the canonical database $Q_1{}^D$ for Q_1. The test succeeds if the frozen head of Q_1 is an element of $Q_2({Q_1}^D)$; otherwise the database ${Q_1}^D$ is a counterexample to the containment.

An important extension of the theory of containment of conjunctive queries is the inclusion of arithmetic comparisons as subgoals, with so-called built-in, or interpreted, predicates (e.g., subgoal $X \leq Y$ with built-in predicate $\leq$). When testing two conjunctive queries with arithmetic comparisons (CQACs) for containment $Q_1 \sqsubseteq Q_2$ using canonical databases, one must consider the set of values in the database as belonging to a totally ordered set, e.g., the integers or reals.

The containment test using canonical databases is conducted as follows. Each basic canonical database is constructed from only those subgoals of Q_1 that have uninterpreted predicates. Each basic canonical database is the canonical database (Q^D for query Q) defined above, together with a partition of the variables of the query into a list of blocks, such that each block is associated with a distinct integer value for all its variables, in increasing order of the integer values on the list of the blocks. The containment test succeeds if the frozen head of Q_1 is an element of $Q_2(Q_1^{D_i})$ on *all* basic canonical databases $Q_1^{D_i}$ for Q_1; otherwise any $Q_1^{D_i}$ on which the test fails is a counterexample to containment. Another more general containment test for conjunctive queries with interpreted (not necessarily arithmetic-comparison) predicates uses homomorphisms on the uninterpreted predicates and logical implication on the built-in predicates; see [15] for the details.

The problem of testing CQACs for containment is complete for ${\Pi_2}^{\mathrm{P}}$, at least in the case of a dense domain such as real numbers. (In fact, the problem is ${\Pi_2}^{\mathrm{P}}$ complete even for conjunctive queries with disequalities $\neq$ as the only comparison predicate.) A containment test for conjunctive queries with negation is outlined in [15]. The test in this case is also complete for ${\Pi_2}^{\mathrm{P}}$, as it also involves exploring an exponential number of canonical databases. The query-containment problem is also ${\mathit{\Pi}_2}^P$ complete for unions of conjunctive queries [1].

Containment questions involving Datalog programs are often harder than for conjunctive queries. It is known that containment of Datalog programs is undecidable, while containment of a Datalog program in a conjunctive query is doubly exponential. However, the important case for purposes of information integration is the containment of a conjunctive query in a Datalog program, and this question turns out to be no more complex than containment of conjunctive queries. To test whether a conjunctive query Q is contained in a Datalog program P, one would "freeze" the body of Q to make a canonical database D. Then a check is done to see if $P(D)$ contains the frozen head of Q. The only significant difference between containment in a conjunctive query and containment in a Datalog program is that in the latter case one must keep applying the rules until either the head of Q is derived or no more facts can be inferred in evaluating $P(D)$.

Key Applications

Query containment was recognized fairly early as a fundamental problem in database query evaluation and optimization. The reason is, for conjunctive queries – a broad class of frequently used queries, whose expressive power is equivalent to that of select-project-join queries in relational algebra – query containment can be used as a tool in query optimization, since the problem of conjunctive-query equivalence is equivalent to the problem of conjunctive-query containment. Specifically, to find a more efficient *and* answer-preserving formulation of a given conjunctive query, it is enough to "try all ways" of arriving at a "shorter" query formulation, by removing a query subgoal, in a process called query minimization [3]. In this process, a subgoal-removal step succeeds only if a containment test ensures equivalence (via containment) of the "original" and "shorter" query formulations.

Note that the problems of query containment and query equivalence are equivalent for conjunctive queries under the common setting of *set semantics for query evaluation,* where both stored relations and query answers are interpreted as sets of tuples. Interestingly, the relationship between the problems of containment and equivalence is very different under *bag semantics for query evaluation,* where both stored relations and query answers are allowed to have duplicates. See Jayram and colleagues [5] for a discussion and references on containment and equivalence under bag semantics, for conjunctive queries as well as for more expressive classes of queries, including CQACs and queries with grouping and aggregation. Jayram and colleagues [5] also present original undecidability results for containment of conjunctive queries with inequalities under bag and bag-set semantics for query evaluation.

In recent years, there has been renewed interest in the study of query containment, because of its close relationship to the problem of answering queries using views [4]. Intuitively, the problem of answering queries using views is as follows: Given a query on a database schema and a set of views (i.e., named queries) over the same schema, can the query be answered (efficiently) using only the views? Alternatively, what is the maximum set of tuples in the answer to the query that can be obtained from the views? As another alternative, in case it is possible to access both the views and the database relations, what is the cheapest query-execution plan for answering the query?

The problem of answering queries using views has emerged as a central problem in integrating information from heterogeneous sources, an area that has been the focus of concentrated research efforts for a number of years [4,15]. An information-integration system can be described logically by views that specify what queries the various information sources can answer. These views might be conjunctive queries or Datalog programs, for example. The "database" of predicates over which these views are defined is not a concrete database but rather a collection of "global" predicates whose actual values are determined by the sources, via the views. Information-integration systems provide a uniform query interface to a multitude of autonomous data sources, which may reside within an enterprise or on the World-Wide Web. Data-integration systems free the user from having to locate sources relevant to a query, interact with each one in isolation, and manually combine data from multiple sources.

Given a user query Q, typically a conjunctive query, on the global predicates, an information-integration system determines whether it is possible to answer Q by using the various views in some combination. In addressing the problem of answering queries using views in the information-integration setting, query containment appears to be more fundamental than query equivalence. In fact, answering a query using only the answers to the views is considered "good enough" even in cases where equivalence does not hold (or cannot be demonstrated), provided that the view-based query rewriting can be shown to be contained in the query and is a maximal (i.e., returning the maximal set of answers) rewriting of the query using the available views and a given rewriting language. (For the details and references on maximally contained rewritings see, e.g., [2].)

Besides its applications in information integration, the problem of answering queries using views is of special significance in other data-management applications. (Please see [4] for the details and references.) For instance, in query optimization finding a rewriting of a query using a set of materialized views (i.e., the answers to the queries defining the views) can yield a more efficient query-execution plan, because part of the computation necessary to answer the query may have already been done while computing the materialized views. Such savings are especially significant in decision-support applications when the views and queries contain grouping and aggregation.

In the context of database design, view definitions provide a mechanism for supporting the independence of the logical and physical views of data. This independence enables the developers to modify the storage schema of the data (i.e., the physical view) without changing its logical schema, and to model more complex types of indices. Provided the storage schema is described as a set of views over the logical schema, the problem of computing a query-execution plan involves figuring out how to use the view answers (i.e., the physical storage) to answer the query posed on the logical schema.

In the area of data-warehouse design the desideratum is to choose a set of views (and indexes on the views) to materialize in the warehouse. Similarly, in web-site design, the performance of a web site can be significantly improved by choosing a set of views to materialize. In both problems, the first step in determining the utility of a choice of views is to ensure that the views are sufficient for answering the queries expected to be posed over the data warehouse or the web site. The problem, again, translates into the view-rewriting problem.

The problem of query containment is also of special significance in artificial intelligence, where conjunctive queries, or similar formalisms such as description logic, are used in a number of applications. The design theory for such logics is reducible to containment and equivalence of conjunctive queries. Original results and a detailed discussion concerning an intimate connection between conjunctive-query containment in database theory and constraint satisfaction in artificial intelligence can be found in [9].

Cross-references

- ▶ Conjunctive Query
- ▶ Query Optimization
- ▶ Query Optimization (in Relational Databases)
- ▶ Query Rewriting Using Views
- ▶ SQL

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases. Addison-Wesley, 1995.
2. Afrati F.N., Li C., and Mitra P. Rewriting queries using views in the presence of arithmetic comparisons. Theor. Comput. Sci., 368(1–2):88–123, 2006.
3. Chandra A.K. and Merlin P.M. Optimal implementation of conjunctive queries in relational data bases. In Proc. 9th Annual ACM Symp. on Theory of Computing, 1977, pp. 77–90.
4. Halevy A.Y. Answering queries using views: A survey. VLDB J., 10(4):270–294, 2001.
5. Jayram T.S., Kolaitis P.G., and Vee E. The containment problem for REAL conjunctive queries with inequalities. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 80–89.
6. Kanellakis P.C. Elements of Relational Database Theory. In Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B). Elsevier and the MIT Press, 1990, pp. 1073–1156.
7. Kimelfeld B. and Sagiv Y. Revisiting Redundancy and Minimization in an XPath Fragment. In Advances in Database Technology, Proc. 11th Int. Conf. on Extending Database Technology, 2008, pp. 61–72.
8. Klug A.C. On conjunctive queries containing inequalities. J. ACM, 35(1):146–160, 1988.
9. Kolaitis P.G. and Vardi M.Y. Conjunctive-Query Containment and Constraint Satisfaction. J. Comput. Syst. Sci., 61 (2):302–332, 2000.
10. Miklau G. and Suciu D. Containment and equivalence for a fragment of XPath. J. ACM, 51(1):2–45, 2004.
11. Saraiya Y. Subtree elimination algorithms in deductive databases. Ph.D. thesis, Stanford University, 1991.
12. Ullman J.D. CS345 lecture notes. http://infolab.stanford.edu/~ullman/cs345-notes.html.
13. Ullman J.D. Principles of Database and Knowledge-Base Systems, Volume II. Computer Science press, 1989.
14. Ullman J.D. The database approach to knowledge representation. In Proc. 13th National Conf. on Artificial Intelligence and 8th Innovative Applications of AI Conf., Volume 2, 1996, pp. 1346–1348.
15. Ullman J.D. Information integration using logical views. Theor. Comput. Sci., 239(2):189–210, 2000.

Query Engine

▶ Query Processor

Query Evaluation

▶ Evaluation of Relational Operators
▶ Query Processing
▶ Query Processing and Optimization in Object Relational Databases

Query Evaluation Plan

▶ Query Plan

Query Evaluation Techniques for Multidimensional Data

Amarnath Gupta
University of California-San Diego, La Jolla, CA, USA

Synonyms

Spatial data

Definition

There are two senses of the term "multidimensional data." The first relates to the data warehousing and online analytical processing (OLAP). The second sense of the term, used mostly in the context of scientific data, refers to variants of array-representable data where the dimensionality refers to the dimensions of the array. Query processing for this class of data uses array algebras [3] and array-specific storage [1] indexing techniques [4].

Key Points

In many scientific applications, data can be represented as multidimensional arrays. For example, the current flow in oceans is a time-varying vector field and can be roughly viewed as 4-dimensional data. In many applications the array may not be uniform, but it can be nested, and even irregular. Query evaluation on this kind of data is greatly dependent on applications. Some applications need to perform operations like value-based clustering on the data, while other applications need fast computation of aggregates such as temporal trends in regions that are selected by user queries. It has been established that storing the multidimensional data as relational tables and developing special routines to handle such relational data is feasible but not optimal, especially when the data is large. ESRI, the GIS provider, has assembled a number of operations collectively called the MapAlgebra for the case where the arrays are two dimensional and the applications are spatial. Recently [2] developed a generic algebra called the gridfield algebra for manipulating arbitrary gridded datasets (i.e., arrays), and present algebraic optimization techniques in these applications. This system is implemented in a system called CORIE (http://www.ccalmr.ogi.edu/CORIE/) for an environmental observation and forecasting system.

Cross-references

▶ Multidimensional Modeling
▶ On-Line Analytical Processing
▶ Raster Data Management and Multi-Dimensional Arrays

Recommended Reading

1. Furtado P. and Baumann P. Storage of multidimensional arrays based on arbitrary tiling. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 480–489.
2. Howe B. and Maier D. Algebraic manipulation of scientific datasets. VLDB J., 14(4):397–416, 2005.
3. Marathe A.P. and Salem K. Query processing techniques for arrays. VLDB J. 11(1):68–91, 2002.
4. Sinha R.R. and Winslett M. Multi-resolution bitmap indexes for scientific data. ACM Trans. Database Syst., 32(3):16, 2007.

Query Execution Engine

▶ Query Processor

Query Execution in Star/ Snowflake Schemas

▶ Query Processing in Data Warehouses

Query Execution Plan

▶ Query Plan

Query Expansion

▶ Web Search Query Rewriting

Query Expansion for Information Retrieval

Olga Vechtomova
University of Waterloo, Waterloo, ON, Canada

Synonyms

QE, Query enhancement; Term expansion

Definition

Query expansion (QE) is a process in Information Retrieval which consists of selecting and adding terms to the user's query with the goal of minimizing query-document mismatch and thereby improving retrieval performance.

Historical Background

The work on query expansion following relevance feedback dates back to 1965, when Rocchio [9] formalized relevance feedback in the vector-space model. Early work on using collection-based term co-occurrence statistics to select query expansion terms was done by Spärck Jones [10] and van Rijsbergen [12].

Foundations

The central task of information retrieval (IR) is to find documents that satisfy the user's information need. This is usually taken to mean finding documents or some parts of them, such as passages, which contain information that would help resolving the user's information need. Therefore, at least in a more traditional sense, IR does not involve providing the user directly with the information needed. The user usually expresses his/her information need in free-text using natural language words and phrases (terms). Sometimes, prior to retrieving the documents, the user's free-text query is translated into controlled vocabulary which is a subset of the words that comprise a natural language. A document is similarly indexed either by the natural language terms that constitute its content, or by a set of controlled index terms that map its contents to the concepts in a given domain. Both directly matching free-text query terms to free-text index terms and translating natural language words to controlled vocabularies are inherently imprecise. In the first case, the main problem is that the user and the author of a document may express the same idea by means of different terms. In the second case, the shades of meanings that natural language terms carry may be lost in the translation process. In addition to these problems, the user's query may be incomplete or inaccurate, i.e., the user may not specify his/her information need exactly or express it accurately.

The goal of query expansion is to enrich the user's query by finding additional search terms, either automatically, or semiautomatically that represent the user's information need more accurately and completely, thus avoiding, at least to an extent, the aforementioned problems, and increasing the chances of matching the user's query to the representations of relevant ideas in documents. Query expansion techniques may be categorized by the following criteria:

- Source of query expansion terms;
- Techniques used for weighting query expansion terms;
- Role and involvement of the user in the query expansion process.

Query expansion can be performed automatically or interactively. In automatic query expansion (AQE), the system selects and adds terms to the user's query, whereas in interactive query expansion (IQE), the system selects candidate terms for query expansion, shows them to the user, and asks the user to select (or deselect) terms that they want to include into (or exclude from) the query.

There are three main sources of QE terms: (i) hand-built knowledge resources such as dictionaries, thesauri, and ontologies; (ii) the documents used in the retrieval process; (iii) external text collections and resources (e.g., the WWW, Wikipedia).

Hand-built knowledge resources have three main limitations: they are usually domain-specific, have to be kept up-to-date, and typically do not contain proper nouns. Experiments with QE using knowledge resources did not show consistent performance improvements. For example, in [13] QE with words manually selected from WordNet, a large domain-independent lexical resource with lexical-semantic relations between words, did not improve well-formulated queries, but significantly improved performance of poorly-constructed queries.

The most common source of QE terms is the text collection used in the retrieval process or its subset (e.g., retrieved documents). These QE techniques showed overall better performance than techniques using hand-built knowledge resources. They can be subdivided into the following categories:

- QE following relevance feedback. QE terms are extracted from the documents retrieved in response to the user's query and judged relevant by the user.
- QE following blind (pseudo-relevance) feedback. QE terms are extracted from the top-ranked documents retrieved in response to the user's query.
- QE using automatically built association thesauri and collection-wide word co-occurrences.

Query Expansion Following Relevance and Pseudo-Relevance Feedback

Relevance feedback (RF) is a process by which the system, having retrieved some documents in response to the user's query, asks the user to assess their relevance to his/her information need. Documents are typically shown to the user in some surrogate form, for example, as document titles, abstracts, snippets of text, query-biased, or general summaries, keywords and key-phrases. The user may also have an option to see the whole document before making the relevance judgement. After the user has selected some documents as relevant, query expansion terms are extracted from them, weighted and the top-weighted terms are either added to the query automatically, or shown to the user for further selection.

Query expansion following relevance feedback has consistently yielded substantial gains in performance in experimental settings. Many term selection methods have been proposed for query expansion following RF. The general idea behind all such methods is to select terms that will be useful in retrieving previously unseen relevant documents. Below is a brief description of a query expansion method [11] which showed consistently high performance results on Text REtrieval Conference (TREC) test collections. The first step is to retrieve documents in response to the user's initial query, which is done by calculating document matching score (Eq. 1) using the Robertson/Spärck-Jones probabilistic model.

$$MS = \sum_{i \in Q} \frac{(k_1 + 1) \times tf_i}{k_1 \times NF + tf_i} \times w_i \tag{1}$$

Where i is a term in the query Q, tf_i is the frequency of i in the document, k_1 is term frequency normalization factor, NF is document length normalization factor and is calculated as $NF = (1\text{-}b) + b \times DL/AVDL$, where DL is document length, $AVDL$ is average document length, b is a tuning constant, w_i is term collection weight, calculated as $w_i = log(N/n_i)$, where N is the number of documents in the collection, n_i is the number of documents containing i.

After the user looks through the top-retrieved documents, and judges some of them as relevant, the system extracts all terms from these documents, ranks them according to the Offer Weight (OW) in Eq. 2, and either adds a fixed number of terms to the query (automatic query expansion), or asks the user to perform the term selection.

$$OW = r \times RW \tag{2}$$

Where r is the number of relevant documents containing the candidate query expansion term and RW is Relevance Weight calculated as shown in Eq. 3:

$$RW = \log\left(\frac{(r+0.5)(N-n-R+r+0.5)}{(R-r+0.5)(n-r+0.5)}\right) \tag{3}$$

Where R is the number of documents judged relevant; r is the same as above; N is the number of documents in the collection; n is the number of documents containing the term.

The subsequent document retrieval with the expanded query is performed using Eq. 1 for document ranking with RW used instead of w.

A related approach to RF is pseudo-relevance or blind feedback (BF), which uses a number of top-ranked documents in the initially retrieved set for query expansion without asking the user to assess their relevance. The query expansion method described above can be used in BF with R being the number of top documents *assumed* to be relevant. Many other QE methods following blind feedback have been proposed. Two of the methods that showed good performance on large test collections are briefly introduced below.

Local Context Analysis (LCA) [14] technique consists of extracting terms (nouns and noun phrases) from n top ranked passages retrieved in response to the user's query. The extracted terms are ranked by their similarity to the entire user's query, and top m terms are added to the query. Carpineto et al. [3] proposed a term ranking method for QE based on Kullback-Leibler divergence (KLD) measure. The method ranks candidate QE terms based on the difference between their distribution in pseudo-relevant documents and in the entire collection.

In general, blind feedback has been demonstrated to be less robust in performance than relevance feedback. The QE performance following BF depends greatly on the performance of the user's initial query: if many of the top-ranked documents retrieved in response to the initial query are relevant, then it is likely that QE terms from these documents will be useful in retrieving other relevant documents. However, if the initial query is poorly formulated or ambiguous, and many top-ranked documents are nonrelevant, then QE terms extracted from such documents may deteriorate performance. Billerbeck and Zobel [2] report that blind feedback in their experiments improves performance of less than a third of queries. They also conclude that the best values for such BF parameters as the number of documents and terms used for QE vary widely across topics.

Query Expansion Using Association Thesauri and Term Co-Occurrence Measures

Unlike QE following relevance or pseudo-relevance feedback, where terms are selected from documents at search time, QE techniques in this category rely on lexical resources automatically constructed prior to the search process. Typically, statistical measures of term similarity are applied to identify terms in a large document collection that co-occur in the same contexts, and which, therefore, are likely to be conceptually related.

For example, Qiu and Frei [7] developed a query expansion method where query expansion terms are selected from an automatically constructed co-occurrence based term-term similarity thesaurus on the basis of the degree of their similarity to all terms in the query. Jing and Croft [4] developed a technique for automatic construction of a co-occurrence thesaurus. Each indexing unit, defined through a set of phrase rules, is recorded in the thesaurus with its most strongly associated terms. An evaluation of different similarity measures (Dice, Jaccard, Cosine, Average Conditional Probability, and Normalized Mutual Information) for selecting query expansion terms from a document collection is reported in [5]. The Dice, Jaccard, and Cosine led to better QE results than Average Conditional Probability and Normalized Mutual Information.

Interactive Query Expansion

In interactive query expansion the task of selecting and adding terms to the user's query is split between the user and the system in such a way that the system selects the candidate query expansion terms, but it is the user who makes the final decision which terms to include into the query. Similarly to automatic query expansion, the most common source of terms used in IQE is a set of documents resulting from either relevance, or pseudo-relevance feedback. Terms are extracted by the system, ranked, and the top-ranked terms are shown to the user for selection. The process of IQE is iterative, and may be triggered either by the system, or the user.

Several studies comparing the effectiveness of AQE and IQE have been conducted. Intuitively, since the user is the one who decides which document is relevant to his/her information need, the user should be able to make better decisions than the system with respect to which terms to add to the query. However, experiments do not offer conclusive results that IQE is more effective than AQE. For instance, Beaulieu [1] showed that AQE is more effective than IQE in operational settings. On the other hand, Koenemann and Belkin [6] reported higher subjective user satisfaction with an IQE system, as well as better performance. This suggests that the effectiveness of IQE is highly variable, depending on the specific interactive system, the users, and the task. Ruthven [8] did a simulation study of a potential benefit of IQE. He concludes that while IQE has potential to achieve higher performance than AQE, this potential may not be easy to realize, because it is difficult for the users to make decisions about which terms are better in differentiating between relevant and nonrelevant documents.

Key Applications

Query expansion is used in some web search engines and enterprise search systems. Although QE following relevance feedback has been experimentally shown as one of the most successful IR techniques, its use in Web search has been limited.

Cross-references

- ▶ BM25
- ▶ Information Retrieval
- ▶ Information Retrieval Models
- ▶ Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model.
- ▶ Query Expansion Models
- ▶ Relevance Feedback for Text Retreival

Recommended Reading

1. Beaulieu, M. Experiments with interfaces to support query expansion. J. Doc. 53(1):8–19, 1997.
2. Billerbeck B. and Zobel J. Questioning query expansion: an examination of behaviour and parameters. In Proc. 15th Australasian Database Conf., 2004, pp. 69–76.
3. Carpineto C., de Mori R., Romano G., and Bigi B. An information-theoretic approach to automatic query expansion. ACM Trans. Information Syst., 19(1):1–27, 2001.
4. Jing Y. and Croft B. An association thesaurus for information retrieval. In Proc. 4th Int. Conf. Computer-Assist IR. "Recherche d'Information Assistée par Ordinateur", pp. 146–160.
5. Kim M-C. and Choi K-S. A comparison of collocation-based similarity measures in query expansion. Inf. Proc. & Man., 35, 19–30, 1999.
6. Koenemann J. and Belkin N.J. A case for interaction: a study of interactive information retrieval behavior and effectiveness. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1996, pp. 205–212.
7. Qiu Y. and Frei H.P. Concept based query expansion. In Proc. 16th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 160–169.
8. Ruthven I. Re-examining the potential effectiveness of interactive query expansion. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 213–220.
9. Salton G. The SMART retrieval system (Chapter 14). Prentice-Hall, Englewood Cliffs NJ. (Reprinted from Rocchio J.J. (1965). Relevance feedback in information retrieval. In Scientific Report ISR-9, Harvard University), 1971.
10. Spärck Jones K. Automatic keyword classification for information retrieval. Butterworths, London, 1971.
11. Spärck Jones K., Walker S., and Robertson S.E. A probabilistic model of information retrieval: development and comparative experiments. Inf. Proc. & Man., 36(6):779–808 (Part 1); 809–840 (Part 2), 2000.
12. van Rijsbergen C.J. A theoretical basis for the use of co-occurrence data in information retrieval. J. Doc., 33, (2):106–119, 1977.
13. Voorhees E. Query expansion using lexical-semantic relations. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1994, pp. 61–69.
14. Xu J. and Croft B. Query expansion using local and global document analysis. In Proc. 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1996, pp. 4–11.

Query Expansion Models

Ben He
University of Glasgow, Glasgow, UK

Synonyms

Term expansion models

Definition

In information retrieval, the query expansion models are the techniques, algorithms or methodologies that reformulate the original query by adding new terms into the query, in order to achieve a better retrieval effectiveness.

Historical Background

The idea of expanding a query to achieve better retrieval performance emerged around the early 1970's. A classical query expansion algorithm is Rocchio's relevance feedback technique proposed in 1971 [10] for the Smart retrieval system. Since then, many different query expansion techniques and algorithms have been proposed.

Foundations

Query expansion models can be classed into three categories: manual, automatic, and interactive. Manual query expansion relies on searcher's knowledge and experience in selecting appropriate terms to add to the query. Automatic query expansion weights candidate terms for expansion by processing the documents returned from the first-pass retrieval, and expands the original query accordingly. Interactive query expansion automates the term weighting process, but it is the user who decides which are the expanded query terms.

Manual query expansion inspires and motivates the user to refine the initial query through heuristics, for instance, by providing a list of candidate terms mined from query log, and allowing user the to choose appropriate terms to add.

Automatic query expansion, different from manual query expansion, expands search queries without any form of human interaction. According to [15], the reason for not involving human interaction could be that the searcher does not want to make an effort perhaps he/she simply does not understand and does not bother adding more terms to the query. Efthimiadis [4] categorized the automatic query expansion methods into three subgroups on which the expansion process is based: search results, collection dependent data structures (e.g., term co-occurrence, term frequency distribution etc.), and collection independent data structures (e.g., lexicon relation between terms, synonyms etc.).

The first category (automatic query expansion based on search results) uses the returned documents from the first-pass retrieval with or without relevance information, as feedback for query expansion. In [10], Rocchio proposed a classical query expansion algorithm based on the Vector Space model. His algorithm has the following steps:

1. The first-pass retrieval consists of ranking the documents for the given query.
2. A term weight w(t, d) is assigned to each term occurring in one of the top-ranked document set D_{psd}. Such a document set is usually called a *pseudo relevance set*. A term weight is first assigned to each term document pair, that is the same weight assigned in the first-pass retrieval. The weighting model used is tf·idf (see TF IDF model).
3. Add the most weighted terms in the pseudo relevance set to the query, and modify the query term weights by taking into account both the original-query term weight (qtw) used in the first-pass retrieval, and the weight assigned by the term weighting model. The query used in the first-pass-retrieval is called the *original query*. Moreover, the query with the modified query term weights is called the *reweighed query*. The reweighed query-with the added query terms is called the *expanded query*. Using Rocchio's method, the new query term weight $qtw_m$is given by Rocchio's query expansion formula as follows.

$$qtw_m = \alpha \cdot qtw + \beta \cdot \sum_{d \in D_{psd}} \frac{w(t,d)}{|D_{psd}|} \qquad (1)$$

If an expanded query term is not in the original query, *qtw* is zero. α and β are free parameters. The new query term weight is given by an interpolation of the original query term weight and the average term weight in the pseudo relevance set with $\alpha + \beta = 1$.

Another popular and successful automatic query expansion algorithm was proposed by Robertson [7,8] in the development of the Okapi system. Okapi's query expansion algorithm is similar to Rocchio's, while using a different term weighting function. It takes the top R documents returned from the first-pass retrieval as the pseudo relevance set. Unique terms in this set are ranked in descending order of the Robertson Selection Value (RSV) weights [7]. A number of top-ranked terms, including a fixed number of non-original query terms, are then added to the query. A major difference between Rocchio's and Robertson's methods is that the former explicitly uses relevant documents for feedback, while the latter assumes a pseudo relevance set. Okapi's query expansion method has been shown to be very effective for ad-hoc information retrieval. For example, in TREC-3 ad-hoc retrieval task, a 15.43% improvement over the Okapi BM25 baseline brought by Okapi's query expansion was reported [9].

Recently, Amati proposed a query expansion algorithm in his Divergence from Randomness (DFR) framework [1,2], which similarly follows the steps in Rocchio's algorithm. However, in Amati's method, term weights are assigned by a DFR term weighting model. Two DFR term weighting models have been proposed, namely Bo1 based on Bose-Einstein statistics, and KL based on Kullback-Leibler divergence. For example, using the Bo1 model, the weight of a term *t* in the pseudo relevance document set D(Rel) is given as:

$$w(t) = tf_x \cdot \log_2 \frac{1+\lambda}{\lambda} + \log_2(1+\lambda) \qquad (2)$$

where λ is the mean of the assumed Poisson distribution of the term in the pseudo relevant document set D(Rel). It is given by $\frac{tf_{rel}}{N_{rel}}$. tf_{rel} is the frequency of the term in the pseudo relevant documents, and N_{rel} is the number of pseudo relevant documents. tf_x is the frequency of the query term in the pseudo relevance document set.

Once the first-pass retrieval is finished, using a document weighting model, a weight is assigned to each term in the top-ranked documents returned from the first-pass retrieval. This corresponds to the first and second steps of the relevance feedback process as introduced above.

In the next step, the original query terms are reweighed. The modified query term weight qtw_m is given by the following parameter-free formula [1]:

$$qtw_m = qtw + \frac{w(t)}{M} \qquad (3)$$

where *qtw* is the original query term weight. *w*(*t*) is the weight of the query term that is given by Bo1. *M* is the upper bound of the weight of a term in the top-ranked documents. For Bo1, it is computed by:

$$\begin{aligned} \mathrm{M} &= \lim_{tf_c \to tf_{c,max}} w(t) \\ &= tf_{c,max} \log_2 \frac{1+P_{max}}{P_{max}} + \log_2(1+P_{max}) \end{aligned} \qquad (4)$$

where tf_c is the frequency of the query term in the whole collection, and $tf_{c,max}$ is the frequency of the query term with the highest *w*(*t*) weight in the top-ranked documents. P_{max} is given by $tf_{c,max}/N$. *N* is the number of documents in the collection. An obvious advantage of Amati's approach is that the query term reweighting formula is parameter free. The parameter α in Rocchio's formula (see Equation (1)) is omitted.

In addition to automatic query expansion based on search results, various query expansion methods have been proposed based on collection dependent data structures (e.g., [6,11,12]), or collection independent data structures (e.g., [3,5,13]).

In interactive query expansion, on one hand, the search system offers the user a list of terms for expansion. On the other hand, it relies on the user to choose the appropriate expanded query terms. Similarly to automatic query expansion, interactive query expansion methods can also be categorized into three subgroups by on which the expansion process is based [4]: search results, collection dependent data structures, and collection independent data structures.

Key Applications

Query expansion models are employed in many information search systems such as library search systems and Web search engines for boosting their retrieval effectiveness.

Experimental Results

Query expansion models are usually helpful in improving retrieval effectiveness for general search tasks such as ad-hoc retrieval [14]. For example, a 15.43% improvement over the BM25 baseline was reported in the TREC-3 adhoc task using Okapi's query expansion method [9].

Table 1 demonstrates the effectiveness of query expansion for ad-hoc retrieval on the TREC (http://trec.nist.gov/) test data. The weighting model used for retrieval is DFRee proposed by G. Amati and implemented in the Terrier Platform (http://ir.dcs.gla.ac.uk/terrier/). Terrier's default query expansion setting is applied, which expands the original query with the 10 most informative terms from the top-3 returned documents. Table 1 shows markable improvement in the retrieval performance, measured by mean average precision, over the baseline. In all cases, the improvement is statistically significant according to the Wilcoxon matched-pairs signed-ranks test.

However, the effectiveness of query expansion becomes unreliable when there are only very few relevant documents, or when the information needed is very specific, in which cases expanding the query does

Query Expansion Models. Table 1. The mean average precision (MAP) obtained with and without query expansion

Task	MAP, No QE	MAP QE	Difference (%)	p-value
TREC-1 ad-hoc	0.2148	0.2478	15.36	4.665e-06
TREC-2 ad-hoc	0.1821	0.2370	30.15	5.306e-08
TREC-3 ad-hoc	0.2557	0.3070	20.06	2.818e-07
TERC-8 small-web	0.2829	0.3164	11.84	9.261e-05
TREC2004 robust	0.2485	0.2920	17.50	2.41e-20
TREC-9 Web	0.2034	0.2180	7.18	0.01561
TREC-10 Web	0.2027	0.2526	24.62	1.715e-06
TREC-2004 Terabyte	0.2646	0.3027	14.40	4.537e-05
TREC-2005 Terabyte	0.3293	0.3828	16.25	2.818e-07
TREC-2006 Terabyte	0.2859	0.3348	17.10	1.582e-05
TREC-2004 Genomics	0.2921	0.3398	16.33	0.0005686
TREC-2005 Genomics	0.2172	0.2500	15.10	0.001759

not bring up more relevant documents. There are also other factors that may affect the effectiveness of query expansion, such as the quality of the top-ranked documents, how much noise the document collection contains etc. For example, query expansion does not improve retrieval performance on the Blog06 test collection (http://ir.dcs.gla.ac.uk/test_collections/blog 06info.html), possibly due to the large amount of spam.

Cross-references

▶ Binary Independence Model
▶ Probabilistic Models
▶ Relevance feedback
▶ Relevance Feedback for Content-Based Information Retrieval
▶ Rocchio's Formula
▶ Web search Relevance Feedback

Recommended Reading

1. Amati G. Probabilistic models for information retrieval based on divergence from randomness. Ph.D thesis, Department of Computing Science, University of Glasgow, Glasgow, UK, 2003.
2. Carpineto C., de Mori R., Romano G., and Bigi B. An information-theoretic approach to automatic query expansion. ACM Trans. Inf. Syst., 19(1):1–27, 2001.
3. Croft B. and Das R. Experiments with query acquisition and use in document retrieval systems. In Proc. 12th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1989, pp. 349–368.
4. Efthimiadis N.E. Query expansion. Annu. Rev. Inf. Syst. Technol., 31:121–183, 1996.
5. Jarvelin K., Kristensen J., Niemi T., Sormunen E., and Keskustalo H. A deductive data model for query expansion. Technical report, Department of Information Studies, 1995.
6. Minker J., Wilson G., and Zimmerman B. An evaluation of query expansion by the addition of clustered terms for a document retrieval system. Inf. Storage Retrieval, 8:329–348, 1972.
7. Robertson S.E. On term selection for query expansion. J. Doc., 46:359–364, 1990.
8. Robertson S.E., Walker S., Beaulieu M.M., Gatford M., and Payne A. Okapi at TREC-4. In Proc. The 4th Text Retrieval Conference, 1995.
9. Robertson S.E., Walker S., Jones S., Hancock-Beaulieu M.M., and Gatford M. Okapi at TREC-3. In Proc. The 3rd Text Retrieval Conference, 1994.
10. Rocchio J. Relevance Feedback in Information Retrieval. Prentice-Hall, USA, 1971, pp. 313–323.
11. Sparck J.K. Automatic Keyword Classification for Information Retrieval. Butterworths, London, 1971.
12. Sparck J.K. Collection properties influencing automatic term classification. Inf. Storage Retrieval, 9:499–513, 1973.
13. Voorhees E. Query expansion using lexical-semantic relations. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1994, pp. 61–69.
14. Voorhees E. TREC: Experiment and Evaluation in Information Retrieval. MIT, Cambridge, MA, USA, 2005.
15. Walker S. The Okapi Online Catalogue Research Projects. Library Association, London, 1989.

Query Language

TORE RISCH
Uppsala University, Uppsala, Sweden

Synonyms

Data manipulation language

Definition

A query language is a specialized programming language for searching and changing the contents of a database. Even though the term originally refers to a sublanguage for only searching (querying) the contents of a database, modern query languages such as SQL are general languages for interacting with the DBMS, including statements for defining and changing the database schema, populating the contents of the database, searching the contents of the database, updating the contents of the database, defining integrity constraints over the database, defining stored procedures, defining authorization rules, defining triggers, etc. The data definition statements of a query language provide primitives for defining and changing the database schema, while data manipulation statements allow populating, querying, as well as updating the database. Queries are usually expressed declaratively without side effects using logical conditions. However, modern query languages also provide general programming language capabilities through the definition of stored procedures. Most query languages are textual, meaning that the queries are expressed as text string processed by the DBMS. There are also graphical query languages, such as Query-By-Example (QBE), where the queries are expressed graphically and then translated into textual queries interpreted by the DBMS.

Key Points

Since data is represented in terms of the data model used by the DBMS, the syntax and semantics of a query language depends on the data model of the DBMS. For example, the relational data model is based on representing data as tables which allows expressing queries over the tables using expressions expressed in variant of predicate logic called relational calculus. Such queries are non-procedural since the user need not specify the details how a query is executed, but rather only how to select and match data from the tables. It is up to the query optimizer to translate the non-procedural logical queries into an efficient program for searching the database. Queries can also be expressed as relational algebra expressions. A query language is said to be relationally complete if its power is equivalent to the power of the relational algebra or the relational calculus. The most widespread query language is SQL the standard language used for interacting with relational DBMSs. Queries in SQL are mainly expressed as syntactically sugared relational calculus expressions. However SQL also has query constructs based on the relation algebra and other paradigms. The logical query language Datalog, as well as SQL's predecessor QUEL, is purely based on predicate calculus. Other data models use other query languages. For example, OQL is used for searching object-oriented databases and Daplex for functional databases.

Cross-references

▶ Data Model
▶ Datalog
▶ Daplex
▶ OQL
▶ QBE
▶ Relational Calculus
▶ SQL
▶ Stored Procedure

Query Languages and Evaluation Techniques for Biological Sequence Data

Sandeep Tata[1], Jignesh M. Patel[2]
[1]IBM Almaden Research Center, San Jose, CA, USA
[2]University of Wisconsin-Madison, Madison, WI, USA

Synonyms

Querying DNA sequences; Querying protein sequences

Definition

A common type of data that is used in life science applications is biological sequence data. Data such as DNA sequence and protein sequence data are growing at a very fast rate. For example, the data at GenBank [GB07] has been growing exponentially, doubling roughly every 18 months. These sequence datasets are often queried in complex ways and the methods required to query these sequences go far beyond the simple string matching methods that have been used in more traditional string applications. In order to enable users to easily pose sophisticated queries on these biological sequences, different languages have been designed to support a rich library of functions. In addition, some database systems have been extended to support a rich set of operators on the sequence data type. Compared to the stand-alone approach, the database method brings the power of algebraic query optimization and the use of indexes making it possible to find efficient execution plans for sophisticated queries. Furthermore, biological sequence processing can be integrated with traditional database processing, such as selecting a subset of data for analysis, or combining data from multiple sources, to produce a powerful sequence analysis and mining system.

Historical Background

Several research efforts have tackled the problem of enabling complex and efficient querying on biological data. Early efforts such as [4] investigated the idea of a Genomics Algebra, which would abstract several biological processes and enable users to construct complex expressions, thereby providing a sophisticated platform for processing biological sequences. As part of another effort, the Periscope project, an algebra for biological sequences called PiQA (Protein Query Algebra) was proposed in [12]. PiQA can be used to construct complex expressions that allow sophisticated manipulation of both genetic and protein sequence data. The Periscope/SQ [13] system and the PQL query language are based on this algebra.

One of the primary operations in biological sequence processing is local alignment. A dynamic programming algorithm was proposed in [9] for this problem. The popular BLAST algorithm [1] is a heuristic version proposed in 1990 to support local alignment search on massive datasets.

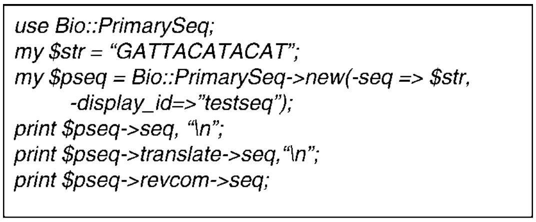
Foundations

The amount of biological sequence data available to scientists for analysis has grown rapidly. While small sequences can be analyzed and queried fairly quickly on computers, large datasets require careful design of algorithms and data structures. Database systems have dealt with the problem of managing and processing large datasets for several years, and the area of query processing for biological sequences aims to bring the benefits of this research to the domain of biological sequences.

One of the most common computations in the context of sequence data is that of finding "similar" sequences. This is often the first step biologists perform to begin investigation of a particular biological sequence they have discovered. For instance, finding a known protein with a sequence similar to the one under investigation may yield some clues to the function of the protein. Sequence similarity is defined in many ways and the type of similarity used may depend on the sequence and the application for which the similarity is being computed. Some distance measures are based on the evolutionary distance between the sequences, while in some cases they simply count the number of mismatches between an alignment of two sequences of the same length. Examples of evolutionary distance matrices include the PAM [3] and BLOSUM [5] matrices for comparing protein sequences. Simple models for mismatches include simply finding sequences that have up to a specified number of mismatches, or models in which mismatches are only tolerated in specific positions (such at the middle portion of the sequence).

The problem of finding a local alignment for two sequences requires finding a region of maximum similarity in a longer sequence that will match with a shorter query sequence. This version of sequence similarity is extremely popular, especially for large DNA sequence repositories. While the Smith-Waterman algorithm [9] is an elegant dynamic programming technique to compute an optimal alignment with $O(n^2)$ cost, for large sequence databases this cost can be prohibitive. In practice, methods that approximate Smith-Waterman but are much faster are used. Popular methods in this category include BLAST [1] and FASTA [8]. While these methods allow the fast evaluation of simple sequence queries, for sophisticated queries, they do not address sophisticated queris, such as a complex string pattern query. Additional methods are required that include a language to express such quires and algorithms to efficiently evaluate the operations in these queries.

Query languages provide a convenient way to express common computational tasks. The class of query languages can be divided into two categories: (i) procedural and (ii) declarative. Several procedural query languages are often general purpose programming languages or scripting languages with a library of functions that support operations on biological sequences. BioPerl and BioPython are two such examples. Figure 1 shows an example of a BioPerl code snippet that uses library function to translate the DNA sequence into the corresponding protein sequence, and another function to compute the reverse complement. These libraries also provide functions to read and translate between several popular sequence file formats. Although such languages make it somewhat easier to develop biological sequence processing applications, they suffer



```
use Bio::PrimarySeq;
my $str = "GATTACATACAT";
my $pseq = Bio::PrimarySeq->new(-seq => $str,
        -display_id=>"testseq");
print $pseq->seq, "\n";
print $pseq->translate->seq,"\n";
print $pseq->revcom->seq;
```

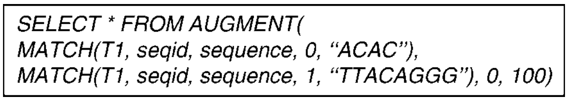
Query Languages and Evaluation Techniques for Biological Sequence Data. Figure 1. Sample BioPerl code.

from two major drawbacks: (i) inability to rapidly express sophisticated queries (ii) lack of an indexing and optimization infrastructure to choose efficient query plans. Applications written in procedural languages often need to solve several database problems in order to scale to larger data sets. As a result, researchers have recently focused on declarative query processing for biological data.

In addition to an algebra, a declarative query infrastructure needs efficient physical operators corresponding to the operators in the algebra in order to produce query evaluation plans. The research prototype system Periscope [14] uses the PiQA algebra and the PiQL query language. Figure 2 shows an example PQL query that finds all instances of the sequence "ACAC" followed by "TTACAGGG" within 100 symbols in the given sequence database. The Periscope system is built as an extension to an object-relational system, which allows the user to mix sequence queries with traditional relation data manipulation queries such as selections and joins. Relational database systems are now commonly used to manage large biological datasets (for example the GMOD project). Arguably, the declarative query processing with the relational frameworks provides a better framework (compared to the procedural framework) for complex biological data analysis.

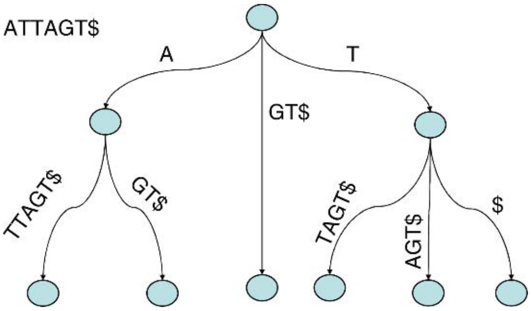
Index Structures

To speed up the evaluation of complex biological sequence operations, a number of index-based methods have been designed. Exploiting these indexes is crucial in having efficient query execution plans, especially to cope with the increasing data volumes and query complexity. The suffix tree is one of the useful data structures in the world of string processing [15]. A Suffix tree is a tree that is built using an input string such that every path from the root of the tree to a leaf node corresponds to a suffix in the string. The edges are labeled with substrings from the string. A suffix tree can be used to evaluate exact substring queries in time proportional to the length of the query. Figure 3 shows an example suffix tree on the string ATTAGT$. In order to check if the string TAG is a substring of the database string (on which the suffix tree is built), one simply follows the labeled edges from the root of the tree while trying to consume the symbols (T,A,G) from the query string.



```
SELECT * FROM AUGMENT(
MATCH(T1, seqid, sequence, 0, "ACAC"),
MATCH(T1, seqid, sequence, 1, "TTACAGGG"), 0, 100)
```

Query Languages and Evaluation Techniques for Biological Sequence Data. Figure 2. Sample PiQL query.



Query Languages and Evaluation Techniques for Biological Sequence Data. Figure 3. Sample suffix tree built on the string "ATTAGT."

Suffix trees can be constructed in time proportional to the length of the input string, i.e., in time O(n). Several algorithms have been designed to accomodate very large input datasets and construct disk based suffix trees. Suffix trees can also be used to efficiently answer approximate matching queries and even compute the local alignment of a query string with the database string.

Another interesting approach to indexing sequence data in the context of similarity search is the use of metric space indexing [7]. While extremely efficient for distance measures that are metrics, such indexing techniques tend to be less flexible than suffix trees.

Given the popularity of BLAST for sequence query processing, a number of approaches have added support for invoking BLAST from a database engine [2,6,11], and regular expression engines in SQL engines have also been adapted for more sophisticated biological pattern matching [10]. Their support for scanning sequences with complex patterns (including arbitrary position-specific weight matrices containing probabilistic models for matching a nucleic or protein sequence) is limited. However, such extensions have recently been made for the declarative framework using suffix trees [12].

Key Applications

Any application that requires complex sequence matching, which include combinity sequence homology matching, and TFBS prediction.

Cross-references

- ▶ Biological Sequences
- ▶ Data Types in Scientific Data Management
- ▶ Index Structures for Biological Sequences
- ▶ Query Languages for the Life Sciences
- ▶ Scientific Databases
- ▶ Suffix Tree

Recommended Reading

1. Altschul S.F., Gish W., Miller W., Myers E.W., and Lipman D.J Basic local alignment search tool. J. Mol. Biol., 215:403–10, 1990.
2. Barbara A. and Eckman A.K. Querying BLAST within a data federation. Q. Bull. IEEE TC on Data Engineering, 27(3):12–19, 2004.
3. Dayhoff M.O., Schwartz R.M., and Orcutt B.C. A model of evolutionary change in proteins. Atlas of Protein Sequence and Structure, 5:345–352, 1978.
4. Hammer J. and Schneider M. Genomics algebra: a new, integrating data model, language, and tool for processing and querying genomic information. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003, 176–187.
5. Henikoff S. and Henikoff J. Amino acid substitution matrices from protein blocks. In Proc. Natl. Acad. Sci., 89(22): 10915–10919, 1992.
6. Hsiao R-L., Stott Parker D., and Yang H-C. Support for BioIndexing in BLASTgres. In Data Integration in the Life Sciences (DILS). LNCS, Vol. 3615. Springer, Berlin, 2005, pp. 284–287.
7. Mao R., Xu Weijia., Singh Neha., and Miranker D.P. An assessment of a metric space database index to support sequence homology. In Proc. IEEE 3rd Int. Symp. on Bioinformatics and Bioengineering, 2003, pp. 375–382.
8. Pearson W.R. and Lipman D.J. Improved tools for biological sequence comparison. In Proc Natl Acad Sci., 85(8):2444–2448, 1988.
9. Smith T.F. and Waterman M.S. Identification of Common Molecular Subsequences. J. Mol. Biol., 147:195–197, 1981.
10. Stephens S., Chen J.Y., Davidson M.G., Thomas S., and Trute B.M. Oracle database 10 g: a platform for BLAST search and regular expression pattern matching in life sciences. Nucleic Acids Res., 33(Database-Issue):675–679, 2005.
11. Stephens S., Chen J.Y., and Thomas Shiby. ODM BLAST: sequence homology search in the RDBMS. Q. Bull. IEEE TC on Data Engineering, 27(3):20–23, 2004.
12. Tata S., Lang W., and Patel J.M. Periscope/SQ: interactive exploration of biological sequence databases. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 1406–1409.
13. Tata S. and Patel J.M. PiQA: an algebra for querying protein data sets. In Proc. 15th Int. Conf. on Scientific and Statistical Database Management, 2003, pp. 141–150.
14. Tata S., Patel J.M., Friedman J.S., and Swaroop A. Declarative querying for biological sequences. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 87.
15. Weiner P. Linear pattern matching algorithm. In Proceedings of the 14th Annual IEEE Symp. on Switching and Automata Theory, 1973, pp. 1–11.

Query Languages for the Life Sciences

Zoé Lacroix
Arizona State University, Tempe, AZ, USA

Synonyms

Biological query Languages; Scientific query Languages; Biological data retrieval, integration, and transformation.

Definition

A *scientific query language* is a query language that expresses the data retrieval, analysis, and transformation tasks involved in the dataflow pertaining to a scientific protocol (or equivalently workflow, dataflow, pipeline). Scientific query languages typically extend traditional database query languages and offer a variety of operators expressing scientific tasks such as ranking, clustering, and comparing in addition to operators specific to a category of scientific objects (e.g., biological sequences).

Historical Background

A scientific query may involve data retrieval tasks from multiple heterogeneous resources and perform a variety of analysis, transformation, and publication tasks. Existing approaches used by scientists include hard coded scripts, data warehouses, link-based federations, database mediation systems, and workflow systems. Hard coded scripts written in Perl and Python are widely used in the biological community. Unlike an approach based on a query language, scripts are very limited in terms of scalability and flexibility. Extending or altering a protocol over time requires writing new scripts. Further, scripts are limited in the degree to which they capture biological expertise so they can be re-used for future related queries. Finally, unless explicitly written to do so, scripts do not assist the user in filtering retrieved data, resolving inconsistencies and sorting and ranking the results, or optimizing the overall execution. Data warehousing consists in collecting data from many possible data sources, curating the data, and creating a new database. Data warehouses typically are relational databases designed to provide users an integrated platform to answer a pre-defined set of scientific tasks.

A federation, e.g., NCBI Entrez and the Sequence Retrieval System (SRS) [4], links semi-autonomous distributed databases with powerful full text indexing and keyword based search techniques for cross database retrieval. The approach expresses navigational queries over linked entries retrieved from multiple databases. In addition, useful tools such as BLAST are often made available. However, the approach is limited in that a federation relies on a materialized index that is difficult to keep up-to-date. Moreover, it focuses on data retrieval and does not support complex queries. Most existing data integration approaches rely on an internal query language that captures data management operations and a user query interface that aims at expressing queries meaningful to the scientists. Existing mediation approaches rely on traditional database query languages such as SQL adapted to handle biological data. K2 follows the Object Data Management Group (ODMG 1999) and its internal query language, called K2 Mediator Definition Language (K2MDL), is a combination of the Object Query Language (OQL) and the Object Definition Language (ODL) both specified by the ODMG. Kleisli (also known as Discovery Hub) provides the collection programming language (CPL) [2], a nested version of SQL, and Java and Perl access programming interfaces (API). Users may express their queries in SQL or through a graphical interface that limits access to query capabilities. The system that supports the Object Protocol Model (OPM) [3] provides a query language similar to OQL (ODMG 1999) and also exploits SQL when integrated data sources are retrofitted from a relational data model. Additional query capabilities may be integrated through CORBA classes. Tools such as BLAST are wrapped through an Application Specific Data Type (ASDT). Users may express their queries in a simplified version of OQL and generate query forms that do not offer access to all query capabilities. Discovery Link (also known as Information Integrator) offers a rich subset of SQL3 (SQL 1993) including user-defined functions, stored procedures, recursion, row types, object views, etc. P/FDM provides support to access specific capabilities of sources. P/FDM uses Daplex that offers a richer syntax than SQL [10] and Prolog. In particular it allows uses of function calls within queries. In addition to these query languages, P/FDM offers users a visual interface that enables users to build their queries by browsing and clicking through the database schema. The P/FDM Web interface uses HTML forms and access the mediator through CGI programs. Both interfaces restrict the query capabilities of the mediator. TAMBIS, which uses CPL internally, is primarily concerned with overcoming semantic heterogeneity through the use of ontologies. It provides users an ontology-driven browsing interface. Thus it too restricts the extent to which sources can be exploited for scientific discovery. To summarize, these systems have made many inroads into the task of data integration from diverse data sources. However they all rely on significant programming effort to adjust to specific scientific tasks, are difficult to maintain and provide user's query language that require programming ability such as SQL, OQL, Daplex, etc. or user's friendly interfaces that significantly limit the query capabilities. A comparative study of these systems and the many causes of failure of traditional database approaches to support the process of scientific discovery are detailed in [6].

Workflows are used in business applications to assess, analyze, model, define, and implement business processes. A workflow automates the business procedures where documents, information or tasks are passed between participants according to a defined set of rules to support an overall goal. In the context of scientific applications, a workflow approach may address the collaboration among scientists, as well as the integration of scientific data and tools. The procedural support of a workflow resembles the query-driven design of scientific problems and facilitates the expression of scientific pipelines (as opposed to a database query). However, because workflows are designed to orchestrate various applications (e.g., Web services) into a combined dataflow they do not provide a query language and do not express the specific queries against biological datasets. Peer-to-Peer (e.g., Chinook) and grid approaches (e.g., myGRID) may offer the support for resource and data sharing without providing a query language.

Foundations

Systems that focus on specific datasets such as sequences, protein structure, phylogenetic trees, metabolic pathways, rely on query languages designed to handle the characteristics of the biological datasets. Life sciences data are exceptionnaly diverse. Biological data include string data (e.g., sequences over various alphabets), 3D geometric data (e.g., protein structure), tree data (e.g., phylogenetic tree), and graph data (e.g., pathways). When their description and annotations are mostly textual and deeply nested in one or several documents, the intrinsic structure of the datasets are often poorly represented by traditional data models, thus poorly accessed and transformed by the corresponding query languages [7]. Each of these specific biological datasets has motivated the design of suitable query languages. For example PiQA, an algebra designed to query sequences, offers a match operator that expresses sequence alignment.

Query languages that express search queries (using regular expressions, wildcards, and Boolean operators) are often developed for life scientific applications. This is because scientific data are mostly textual (annotation on scientific instances) and scientists seem to mimic the manual query process they follow when exploring data sources on the Web. Querying systems that are compatible with scripting languages such as Perl are also favored because most scientific protocols are implemented with scripts. Additional features expected by scientists include the ability to compare and rank results. The ability to express cross-database queries and to integrate scientific analysis tools such as BLAST has a significant value to the scientist.

Scientific queries may exploit metadata as well as data. Querying scientific data through their meaning expressed in a terminology or an ontology is scientist-friendly because it hides the complex structure of data as they are organized and stored in the various repositories. A domain ontology may also provide a global schema for multiple integrated scientific resources. It facilitates scientific tasks such as *profiling* which aims at combining all information known about a scientific object. TAMBIS is an example of a system that provides a ontology-driven query interface.

The challenge for the design of a biological query language is to express the data management tasks involved in the scientific dataflow while offering meaningful and scientist-friendly query functionality. A scientific query language typically handles complex datatypes including text, string, tree, graph, list, variant, etc. for data and metadata. Scientific queries may invoke an increadibly diverse range of functions including extracting information in a textual document, identifying a feature common to two biological sequences, exploring a biological pathway, profiling a gene, etc.

Key Applications

The Acedb Query Language (AQL) is the language developed for the AceDB genome database. It is inspired by the Object Query Language (OQL), and Lorel the query language developed for semi-structured and XML data (also based on OQL). AQL expressions use the Select, From, Where syntax. Queries in AQL express traditional database retrieval and transformation against a data warehouse mostly populated with annotated scientific data (textual). It provides AcePerl and object-oriented Perl module access local or remote AceDB databases transparently. Many biological resources use the AceDB system that is still updated and maintained by the Sanger institute. The former version of AQL consisting of search statements of the form ”find Gene TP*” is currently used to query WormBase.

SRS has its own query language that express retrieval queries using string comparison, wildcards, regular expresions, numeric comparison, Boolean operations, and the *link* operator specific to SRS that allows the navigation through resources exploiting the interal cross-references (hyperlinks, indices, and composite structures) made available by the providers. SRS uses Icarus as its internal interpreted object-oriented programming language. SRS queries return sets of entries or lists of entry identifiers that can be sorted with respect to various criteria. Because SRS integrates scientific analysis tools (e.g., BLAST) as tool-specific databanks, SRS queries may exploit the expressive power of the scientific analysis tools integrated in the federation. However, the SRS query language mostly expresses retrieval queries over cross databases. For a detailed description of the system and its query language see Chapter 5 in [6]. SRS is currently updated and mainted by Biowisdom Ltd.

The Life Sciences community has developed unique approaches to handle complex retrieval and integration tasks. They express the query “retrieve everything known about [a specific scientific instance].” Unlike a query language, the resources where the data are to be retrieved and the criteria for retrieval cannot be selected by the user. These approaches include sequence profiling tools that express complex data retrieval and integration queries over multiple heterogeneous resources to generate summaries of all information related to a particular biological sequence. For example, the Karlsruhe Bioinformatik Harvester (KIT) crawls and crosslinks 25 biological resources. Data integration is limited as queries against KIT are species-specific boolean expressions of keywords and the result is a list of summaries (each summary corresponds to a biological sequences), where each summary is the concatenation of relevant retrieved pages. Other approaches focus on the textual material contained in the biological data sources and use biological language processing to achieve information retrieval, extraction, classification, and integration [5]. These approaches retrieve documents from various resources and generate a document summary as input. Textpresso uses the Gene Ontology (GO) to markup documents related to a particular organism retrieved from PubMed. It provides a query language that expresses Boolean expressions on keyword, category, and attribute. Because the approach is document-driven, the query language expresses advanced search queries. Patent databases are other documented sources of biological information. In addition to the traditional features of querying documented data, Patent Lens offers the useful feature of querying the biological sequences contained in the patent documents with BLAST [9].

The use of a domain ontology as an interface (or view) for biological data is usually well received by the scientists. Such approches typically support exploration, navigation, and search queries. TAMBIS uses an ontology to provide an homogeneous layer over the integrated resources (data sources and applications) that acts as a global schema, to hide the heterogeneities of the integrated resources, and to be used as a query interface. WormBase is a warehouse derived from ACeDB and BioMart databases and uses biomaRt, a bioconductor package that provides a platform for data mining. BioMart databases are annotated with the Gene Ontology (GO) and the eVOC ontologies. These annotations provide support for querying data. EnsMart organizes data with respect to *foci*, that are central scientific objects (e.g., gene). All data are linked to the instances of those central objects. Queries consist of the selection of the species and focus of interest and the specification of filters and outputs. AmiGO provides a query interface to GO, genes, and gene products.

The Pathway Tools developed and maintained at SRI International to support pathway databases produces a query form for basic or advanced search against BioCyc databases. Each form consists of the selection of the BioCyc database (queries are limited to the scope of a single resource), the field(s) to be searched or the ontology term(s), and the format of the output. Most of the databases developed with BioCyc allow the use of BLAST. The BioVelo query language is designed to retrieve data from BioCyc databases. It is based on the object-oriented BioCyc data model composed of object classes that are identified to ontology classes and express set comprehension statements used in functoinal programming languages such as Python.

Future Directions

The design of query languages for the Life Sciences is a challenging problem. The desired expressive power and features of the languages are not well understood. The expectations of the users (life scientists) and the developers (computer scientists) are often dissimilar. Life scientists expect support for all their scientific questions which often go beyond the scope of traditional database query languages handling data retrieval and transformation. Scientific questions are often complex protocols that invoke not only data retrieval and transformation tasks but ranking, comparison, clustering and classification as well as sophisticated analysis tasks. The implementation of such a "query" would require accessing multiple resources and coordinating the dataflow as currently achieved by scientific workflow systems such as Taverna. However, workflow systems lack the elegance of a query language and its benefits such as query planning and optimization. Because many of the scientific tasks invoked in a scientific protocol could be expressed by operators of a query language, the design of a generic query language to handle data retrieval, transformation, comparison, clustering, and integration operations would be valuable to scientific data management.

Data Sets

AceDB databases http://www.acedb.org/Databases/
Public SRS servers http://downloads.biowisdomsrs.com/publicsrs.html

Biopathways Graph Data Manager (BGDM) http://hpcrd.lbl.gov/staff/olken/graphdm/graphdm.htm
BioCyc database collection http://www.biocyc.org/
Textpresso for C. elegans http://www.textpresso.org/
Patent Lens http://www.patentlens.net/daisy/patentlens/patentlens.html
WormBase http://www.wormbase.org/ and http://www.wormbase.org/db/searches/wb_query
BioMart http://www.biomart.org/
EnsMart http://www.ensembl.org/EnsMart

URL to Code

Chinook http://www.bcgsc.bc.ca/chinook/
myGRID http://www.mygrid.org.uk/
AceDB Query Language http://www.acedb.org/Cornell/aboutacedbquery.html
AQL - Acedb Query Language http://www.acedb.org/Software/whelp/AQL/
Lorel - Lore Query Language http://infolab.stanford.edu/lore/
BioVelo Query Language http://www.biocyc.org/bioveloLanguage.html
BioCyc Pathway Tools http://bioinformatics.ai.sri.com/ptools/ptools-overview.html
Karlsruhe Bioinformatik Harvester (KIT) http://harvester.fzk.de/harvester/
biomaRt and Bioconductor http://www.bioconductor.org/
AmiGO http://amigo.geneontology.org/cgi-bin/amigo/go.cgi

Cross-reference

- ▶ Database mediation
- ▶ Data integration
- ▶ Graph management for the life sciences
- ▶ Management of gene expression data
- ▶ Metadata management and resource discovery
- ▶ Scientific workflows.

Recommended Reading

1. Bartlett J. C. and Toms. E.G. Developing a Protocol for Bioinformatics Analysis: An Integrated Information Behaviors and Task Analysis Approach. J. American. Soc. for Inf. Sci. & Tech., 56(5):469–482, 2005.
2. Buneman P., Naqvi S.A., Tannen V., and Wong L. Principles of Programming with Complex Objects and Collection Types. Theoretical Computer Science, 149(1):3–48, 1995.
3. Chen and I-Min A. Markowitz. Victor M. An Overview of the Object-Protocol Model (OPM) and OPM Data Management Tools. Inf. Syst., 20(5):393–418, 1995.
4. Etzold T., Harris H., and Beaulah S. SRS: An Integration Platform for Databanks and Analysis Tools in Bioinformatics, chapter 5, pp. 109–146. In Lacroix and Critchlow [6], 2003.
5. Hunter L. and Bretonne K. Cohen. Biomedical Language Processing: Perspective What's Beyond PubMed? Molecular Cell, 21 (5):589–594, 2006.
6. Lacroix Z. and Critchlow T. (eds.) Bioinformatics: Managing Scientific Data. Morgan Kaufmann, 2003.
7. Lacroix Z., Ludaescher B., and Stevens R. Integrating Biological Databases, chapter 42, pp. 1525–1572. Vol. 3 of Lengauer [8], 2007.
8. Lengauer T. (ed). Bioinformatics - From Genomes to Therapies. Wiley-VCH Publishers, 2007
9. Seeber I. Patent searches as a complement to literature searches in the life sciences-a 'how-to' tutorial. Nature Protocols, 2(10): 2418–28, 2007.
10. Shipman D.W. The Functional Data Model and the Data Language DAPLEX. ACM Trans. Database Syst., 6(1):140–173, 1981.
11. Stevens R., Goble C., Baker P., and Brass A. A Classification of Tasks in Bioinformatics. Bioinformatics, 17(2):180–188, 2001.

Query Load Balancing in Parallel Database Systems

Luc Bouganim
INRIA, Rocquencourt, Lechesnay Cedex, France

Synonyms

Resource scheduling

Definition

The goal of parallel query execution is minimizing query response time using inter- and intra-operator parallelism. Inter-operator parallelism assigns different operators of a query execution plan to distinct (sets of) processors while intra-operator parallelism uses several processors for the execution of a single operator thanks to data partitioning. Conceptually, parallelizing a query amounts to divide the query work in small pieces or tasks assigned to different processors. The response time of a set of parallel tasks being that of the longest one, the main difficulty is to produce and execute these tasks such that the query load is evenly balanced within the processors. This is made more complex by the existence of dependencies between tasks (e.g., pipeline parallelism) and synchronizations points. Query load balancing relates to static and/or dynamic techniques and algorithms to balance the query load within the processors so that the response time is minimized.

Historical Background

Parallel database processing appeared very early in the context of database machines in the 1970s. Parallel algorithms (e.g., hash joins) were later proposed in the early 1980s, where tuples are uniformly distributed at every stage of the query execution. However several works (e.g., [8]) gave considerable evidence that data skew, i.e., non-uniform distribution of tuples, exists and its negative impact on query execution was shown in e.g., [7]. This motivated numerous studies [10] on intra- and inter-operator load balancing in the 1990s.

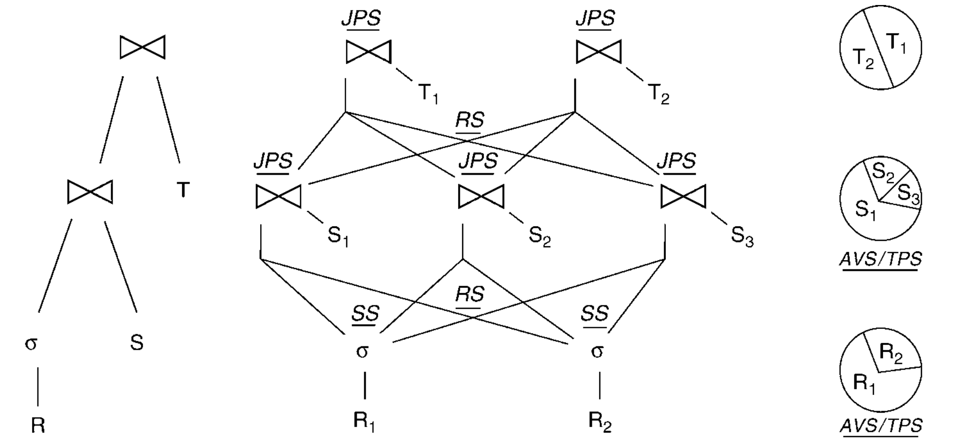
Foundations

Load balancing problems can appear with intra-operator parallelism (variation in partition size, namely *data skew*) and inter-operator parallelism (variation in the complexity of operators, synchronization problems). Intra- and inter-operator load balancing problems are first detailed on a simplified query execution plan example considering a static allocation of processors to the query operators. The main load balancing techniques proposed to address these problems are described next:

Load balancing problems

Figure 1(a) shows a simplified query execution plan for the following query: "Select T.b from R, S, T where R. Rid = S.Rid and S.Sid = T.Sid and R.a = value" (the Scan and Project operators were omitted to simplify the drawing). The following assumptions are made: (i) the degree of parallelism (i.e., number of processors allocated) for the selection on R (called σR), the join with S (called ⋈S) and the join with T (called ⋈T) has been statically fixed, using a cost model, to respectively 2, 3, and 2; and (ii) these operators are processed in pipeline, thus leading to a total degree of parallelism of 7.

Intra-operator load balancing issues are first illustrated using the classification proposed in [13]. As shown in Fig. 1(c), R and S are poorly partitioned because of *Attribute Value Skew (AVS)* inherent in the dataset and/or *Tuple Placement Skew (TPS)*. The processing time of the two instances σR1 and σR2 are thus not equal. The case of ⋈S is likely to be worse (see Fig. 1b). First, the number of tuples received can be different from one instance to another because of poor redistribution of the partitions of R (*Redistribution Skew, RS*) or variable selectivity according to the partition of R processed (*Selectivity Skew, SS*). Finally, the uneven size of S partitions (*AVS/TPS*) yields different processing times for tuples sent by the σR operator and the result size is different from one partition to the other due to join selectivity (*Join Product Skew, JPS*). The skew effects are therefore propagated toward the query tree and even with a perfect partitioning of T, the processing time of ⋈T1 and ⋈T2 can be highly different (uneven size of their left input resulting from ⋈S). Intra-operator load balancing is thus difficult to achieve statically, given the combined effects of different types of data skew.



a Query plan

b Parallel query plan

c Data partitioning

Query Load Balancing in Parallel Database Systems. Figure 1. Intra- and inter-operator load balancing problems on a simple example.

In order to obtain good load balancing at the inter-operator level, it is necessary to choose how many and which processors to assign to the execution of each operator. This should be done while taking into account pipeline parallelism, which requires inter-operator communication and introduces precedence constraints between operators (i.e., an operator must be terminated before the next one begins). In [15], three main problems are described: (i) the degree of parallelism and the allocation of processors to operators, when decided in the parallel optimization phase, are based on a possibly inaccurate cost model. Indeed, it is difficult, if not impossible, to take into account highly dynamic parameters like interference between processors, memory contentions, and obviously, the impacts of data skew; (ii) the choice of the degree of parallelism is subject to errors because both processors and operators are discrete entities. For instance, considering Fig. 1(b), the number of processors for σR, ⋈S and ⋈T may have been computed by the cost model as respectively 1.5, 3.8 and 2.4 and have been rounded to 2, 3 and 2 processors. But the good distribution, taking into account data skew on S partitions should have been 1, 4 and 2; (iii) the processors associated with the latest operators in a pipeline chain may remain idle a significant time. This is called the pipeline delay problem. For instance, while tuples do not match the selection on R or the join with S, processors assigned to ⋈T remain idles.

In a shared-nothing architecture, the inter-operator load balancing problem is even more complex, since the degree of parallelism and the set of processors assigned for some operators is constrained by the physical placement of the manipulated data. For instance, if R is partitioned on two nodes, σR must be executed on these nodes.

This simple example thus shows that static allocation of processors to operators is usually far from optimal, thus advocating for more dynamic strategies. In the following section, existing proposals at the intra- and inter-operator level are detailed.

Intra-Operator Load Balancing

Good intra-operator load balancing depends on the degree of parallelism and on the allocation of processors for the operator. For some algorithms, e.g., the parallel hash join algorithm, these parameters are not constrained by the placement of the data. Thus, the home of the operator (the set of processor where it is executed) must be carefully decided. The skew problem makes it hard for a parallel query optimizer to make this decision statically (at compile-time) as it would require a very accurate and detailed cost model. Therefore, the main solutions rely on adaptive techniques or specialized algorithms which can be incorporated in the query optimizer/processor. These techniques are described below in the context of parallel joins, which has received much attention. For simplicity, each operator is given a home either statically or just before execution, as decided by the query optimizer/processor.

Adaptive techniques: The main idea is to statically decide on an initial allocation of the processors to the operator (using a cost model) and, at execution time, adapt this decision to skew using load reallocation. A simple approach to load reallocation is detecting the oversized partitions and partition them again onto several processors (among those already allocated to the operator) to increase parallelism [6]. This approach is generalized in [2] to allow for more dynamic adjustment of the degree of parallelism. It uses specific *control operators* in the execution plan to detect whether the static estimates for intermediate result sizes differ from the run-time values. During execution, if the difference between estimate and real value is high enough, the control operator performs data redistribution in order to prevent join product skew and redistribution skew. Adaptive techniques are useful to improve intra-operator load balancing in all kinds of parallel architectures. However, most of the work has been done in the context of shared-nothing where the effects of load unbalance are more severe on performance.

Specialized algorithms: Parallel join algorithms can be specialized to deal with skew. The approach proposed in [3] is to use multiple join algorithms, each specialized for a different degree of skew, and to determine the best at execution time. It relies on two main techniques: range partitioning and sampling. Range partitioning is used instead of hash partitioning (in the parallel hash join algorithm) to minimize redistribution skew of the building relation. Thus, processors can get partitions of equal number of tuples, corresponding to different ranges of join attribute values. To determine the values that delineate the range values, sampling of the building relation is used to produce a histogram of the join attribute values, i.e., the numbers of tuples for each attribute value.

Sampling is also useful in determining which algorithm to use and which relation to use for building or probing. The parallel hash join algorithm can then be adapted to deal with skew as follows: (i) Sample the building relation to determine the partitioning ranges. (ii) Redistribute the building relation to the processors using the ranges. Each processor builds a hash table containing the incoming tuples. (iii) Redistribute the probing relation using the same ranges to the processors. For each tuple received, each processor probes the hash table to perform the join. This algorithm can be further improved to deal with high skew using additional techniques and different processor allocation strategies [3]. A similar approach is to modify the join algorithms by inserting a scheduling step which is in charge of redistributing the load at runtime [14].

Inter-Operator Load Balancing

The inter-operator load balancing problem was extensively addressed during the 1990s. Since then many processor allocation algorithms have been proposed for different target parallel architectures and considering CPU, I/Os or other resources, such as available memory.

The main approach in shared-nothing is to determine dynamically (just before the execution) the degree of parallelism and the localization of the processors for each operator. For instance, the Rate Match algorithm [9] uses a cost model in order to define the degree of parallelism of operators having a producer-consumer dependency such that the producing rate matches the consuming rate. It is the basis for choosing the set of processors which will be used for query execution (based on available memory, CPU, and disk utilization). Many other algorithms are possible for the choice of the number and localization of processors, for instance, by a dynamic monitoring and adjustment of the use of several resources (e.g., CPU, memory and disks) [11].

Shared-disk and shared-memory architectures provide more flexibility since all processors have equal access to the disks. Hence there is no need for physical relation partitioning and any processor can be allocated to any operator [12].

Considering the shared-disk architecture, Hsiao et al. [5] propose to assign processors recursively from the root up to the leaves of a so-called *allocation tree*. This tree is derived from the query tree, each pipeline chain (i.e., set of operators having pipeline dependencies) being represented as a node. The edges of the allocation tree represent precedence constraints. All available processors are assigned to the root node of the allocation tree (the last pipeline chain to be executed). Then, a cost model is used to divide the CPU power between each child of the root in order to ensure that all the data necessary for the execution of the root pipeline chain will be produced synchronously.

The approach proposed in [4] for shared-memory allows the parallel execution of independent pipeline chains called tasks. The main idea is combining IO-bound and CPU-bound tasks to increase system resource utilization. Before execution, a task is classified as IO-bound or CPU-bound using cost model information. CPU-bound and IO-bound tasks can then be run in parallel at their optimal IO-CPU balance, by dynamically adjusting the degree of intra-operator parallelism of the tasks.

Intra-Query Load Balancing

Intra-query load balancing combines intra- and inter-operator parallelism. To some extent, given a parallel architecture, the load balancing techniques presented above can be extended or combined. For instance, the control operators used a-priori for intra-operator load balancing can modify the degree of parallelism of an operator, thus impacting inter-operator load balancing [2].

A general load balancing solution in the context of hierarchical parallel architectures (a shared-nothing system whose nodes are shared-memory multiprocessors) is the execution model called Dynamic Processing (DP) [1]. In such systems, the load balancing problem is exacerbated because it must be addressed both locally (among the processors of each shared-memory node) and globally (among all nodes). The basic idea of DP is decomposing the query into self-contained units of sequential processing, each of which can be carried out by any processor. Intuitively, a processor can migrate horizontally (intra-operator parallelism) and vertically (inter-operator parallelism) along the query operators. This minimizes the communication overhead of inter-node load balancing by maximizing intra and inter-operator load balancing within shared-memory nodes.

Key Applications

Load balancing techniques are essential in applications dealing with very large databases and complex queries, e.g., data warehousing, data mining, business intelligence and more generally all OLAP (On Line Analytical Processing) applications.

Data Sets

DBGen, a synthetic data generator can be used to generate biased data distribution, for studying intra-operator load-balancing issues. It allows generating data with non-uniform distribution (Zipfian, Poisson, Gaussian, etc). See http://research.microsoft.com/∼Gray/DBGen/

Cross-references

▶ Parallel Architectures
▶ Parallel Database Management
▶ Parallel Data Placement
▶ Parallel Query Processing
▶ Storage Resource Management

Recommended Reading

1. Bouganim L., Florescu D., and Valduriez P. Dynamic load balancing in hierarchical parallel database systems. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 436–447.
2. Brunie L. and Kosch H. Control strategies for complex relational query processing in shared nothing systems. ACM SIGMOD Rec., 25(3):34–39, 1996.
3. DeWitt D.J., Naughton J.F., Schneider D.A., and Seshadri S. Practical skew handling in parallel joins. In Proc.18th Int. Conf. on Very Large Data Bases, 1992, pp. 27–40.
4. Hong W. Exploiting inter-operation parallelism in XPRS. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1992, pp. 19–28.
5. Hsiao H., Chen M.S., and Yu P.S. On parallel execution of multiple pipelined hash joins. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 185–196.
6. Kitsuregawa M. and Ogawa Y. Bucket spreading parallel hash: a new, robust, parallel hash join method for data skew in the super database computer. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 210–221.
7. Lakshmi M.S. and Yu P.S. Effect of skew on join performance in parallel architectures. In Int. Symp. Databases in Parallel and Distributed Systems, 1988, pp. 107–120.
8. Lynch C. Selectivity estimation and query optimization in large databases with highly skewed distributions of column values. In Proc. 14th Int. Conf. on Very Large Data Bases, 1988, pp. 240–251.
9. Metha M. and DeWitt D. Managing intra-operator parallelism in parallel database systems. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 382–394.
10. Özsu T. and Valduriez P. Principles of Distributed Database Systems (2nd edn.). Prentice Hall, 1999 (3rd edn., forthcoming).
11. Rahm E. and Marek R. Dynamic multi-resource load balancing in parallel database systems. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995.
12. Shekita E.J. and Young H.C. Multi-join optimization for symmetric multiprocessor. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 479–492.
13. Walton C.B., Dale A.G., and Jenevin R.M. A taxonomy and performance model of data skew effects in parallel joins. In Proc. 17th Int. Conf. on Very Large Data Bases, 1991, pp. 537–548.
14. Wolf J.L., Dias D.M., Yu P.S., Turek J. New Algorithms for parallelizing relational database joins in the presence of data skew. IEEE Trans. Knowl. Data Eng., 6(6):990–997, 1994.
15. Wilshut N., Flokstra J., and Apers P.G. Parallel evaluation of multi-join queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 115–126.

Query Mapping

▶ Query Translation

Query Optimization

Evaggelia Pitoura
University of Ioannina, Ioannina, Greece

Synonyms

Query compilation

Definition

A query optimizer translates a query into a sequence of physical operators that can be directly carried out by the query execution engine. The output of the optimizer is called a *query execution plan*. The execution plan may be thought of as a dataflow datagram that pipes data through a graph of query operators. The goal of query optimization is to derive an efficient execution plan in terms of relevant performance measures, such as memory usage and query response time. To achieve this, the optimizer needs to provide: (i) a space of execution plans (search space), (ii) cost estimation techniques to assign a relevant cost to each plan in the search space, and (iii) an enumeration algorithm to search through the space of plans.

Key Points

The query optimizer takes as input a parsed query and produces as output an efficient execution plan for the query. The task of the optimizer is nontrivial, since given a query (i) there are many logically equivalent algebraic expressions (for instance, resulting from the commutativity property among the logical operators), and (ii) for each expression, there are many physical operators supported by the query execution engine for implementing each logical operator (for example, there are several join algorithms, e.g., nested-loop and sort-merge join, for implementing join). The task of finding an equivalent algebraic expression is often called *query rewriting*.

The optimizer needs to enumerate all possible execution plans, estimate their cost and select the one with the smallest cost. The cost assigned to each plan is based on a *cost model* that provides an estimation of the resources needed for its execution, where the resources include CPU time, I/O cost, memory, and communication bandwidth. The cost model relies on statistics maintained on relations and indexes, and uses cost formulas for estimating the selectivity of the various operators and their expected recourse usage. Often, dynamic programming techniques are used to enumerate different plans. These techniques, exploit the fact that to obtain an optimal plan for an expression, it suffices to consider only the optimal plans for its sub-expressions [1,2,3].

Cross-references

- ▶ Query Optimization (in Relational Databases)
- ▶ Query Optimization in Sensor Networks
- ▶ Query Plan
- ▶ Query Processing
- ▶ Query Rewriting

Recommended Reading

1. Chaudhuri S. An overview of query optimization in relational systems. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 1998, pp. 34–43.
2. Ioannidis Y. Query optimization. In Handbook of Computer Science, A.B. Tucker (ed.). CRC Press, 1996.
3. Jarke M. and Koch J. Query optimization in database systems. ACM Comput. Surv., 16(2):111–152, 1984.

Query Optimization (in Relational Databases)

Thomas Neumann
Max-Planck Institute for Informatics, Saarbrücken, Germany

Synonyms

Query compilation

Definition

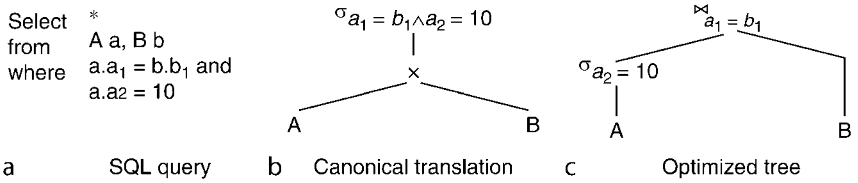
Database queries are given in declarative languages, typically SQL. The goal of query optimization is to choose the best execution strategy for a given query under the given resource constraints. While the query specifies the user intent (i.e., the desired output), it does not specify how the output should be produced. This allows for optimization decisions, and for many queries there is a wide range of possible execution strategies, which can differ greatly in their resulting performance. This renders query optimization an important step during query processing.

Historical Background

One of the first papers to discuss query optimization in relational database systems was the seminal System R paper [2]. It introduced a dynamic programming algorithm for optimizing the join order, and coined the concept of interesting orders for exploiting available orderings. Later approaches increased the set of optimized operators, and included a rule based description of optimization techniques (whereas the optimizations in System R were hard wired). One prominent example is the Starburst [3] optimizer. It introduced a different internal representation (query graph model), that could express complex queries suitably for optimization, and proposed using grammar-like rules to combine low level physical operators (LOLEPOPs) in execution plans. Besides being rule-based, the optimization itself used a bottom-up approach similar to System R. Another family of optimizers was introduced by Volcano [4] (which itself evolved from Exodus) and Cascades [5]. Instead of bottom-up constructive optimization they used transformative top-down optimization with memorization. Besides these more fundamental approaches, a rich literature of optimization techniques exist, ranging from support for specific operators like outer joins or expensive predicates to fundamental data reduction like magic sets.

Foundations

The key idea behind query optimization is the observation that the same query can be formulated in different ways. When a user gives a query in SQL, the query is first parsed and analyzed, and then brought into an internal representation, for example in relational algebra. This translation first creates a canonical representation, as shown in Fig. 1. As a first translation



Query Optimization (in Relational Databases). Figure 1. Translating a SQL query.

step, the *select-from-where* queries in a) can be answered by combining all relations in the *from* clause with a cross product and then checking the *where* condition on each resulting tuple (shown in b)). But cross products are expensive operations, therefore it is preferable to move a part of the *where* condition into the cross product to form a join. The remaining part of the condition can be checked before the join, which further reduces the effort for the join (shown in c)). Both operator's trees produce the same tuples when executed, and thus are different representations of the same query. But executing the tree in (i) is most likely cheaper than executing the tree in (ii), which means that (iii) is preferable. The query optimizer therefore starts by translating the query into a canonical representation, which is easy to construct but inefficient to execute, and therefore finds better representations of the same query.

The base for finding better alternatives is the concept of *algebraic equivalences*. Two algebra expressions are equivalent if they produce the same result when executed. As this is difficult to decide in general, query optimizers instead rely on known equivalences. For example the join operator is commutative and associative:

$$A \bowtie B \equiv B \bowtie A$$
$$A \bowtie (B \bowtie C) \equiv (A \bowtie B) \bowtie C$$

Many equivalences are known from the literature [7]. The equivalences form the search space that is explored by the query optimizer: When two (sub-)expressions are equivalent, the optimizer is free to choose any of the two. Optimizers that are directly based upon this principle are called *transformative* optimizers, as they transform algebra expressions into other algebra expressions by applying algebraic equivalences. Transformative optimizers are relatively easy to build and can potentially make use of arbitrary equivalences, but an efficient exploration of the search space is very difficult. Most transformative optimizers are therefore only heuristical. The family of *constructive* optimizers does not apply the equivalences directly, but builds expressions bottom-up from smaller expressions such that the resulting expression is still equivalent to the original expression. This allows for a much more efficient exploration of the search space, but is difficult to organize for more complex equivalences. Therefore most constructive optimizers are at least partially transformative, applying transformative rewrite heuristics for complex equivalences before (and after) the constructive optimization step.

The goal of query optimization is improving query processing, which means that the query optimizer needs to take into account the runtime effect of different alternatives. This is done by estimating the *costs* of executing an alternative. A primitive way to estimate the costs is to estimate the number of tuples processed. The intuition here is that a larger number of processed tuples implies more effort spent on executing the query. This estimation requires statistical information, in particular the cardinalities of the relations and the selectivities of the operators involved, but given these it can be computed directly from the algebra expression. Unfortunately this is much too inaccurate for practical purposes. A proper cost function should model the expected execution time (as this is the most common optimization goal), which implies taking into account access patterns on disk, costs for evaluating expensive predicates, etc. The cost function is therefore usually a linear combination of expected I/O costs and expected CPU costs. But this information cannot be derived from the algebra expression, as it is not detailed enough.

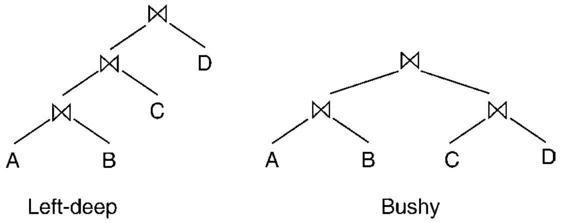
Optimizers therefore distinguish between *logical algebra* and *physical algebra*. The logical algebra consists of all operator concepts known to the optimizer, while the physical algebra consists of the operator implementations supported by the execution engine. For example the logical algebra contains one inner join operator ⋈, while the physical algebra contains one join operator for each supported implementation, like nested loop join $\bowtie^{NL}$ or sort-merge join $\bowtie^{SM}$. While the initial query is represented in logical algebra (or an equivalent calculus), the final result must be in physical algebra, e.g., the optimizer must decide which operator implementations should be used. As these physical algebra expressions (including some annotations) could be executed by the runtime system, they are often called *query execution plans*. The logical algebra is more abstract, which can be useful for optimization, but ultimately the optimizer must construct physical algebra expressions. In particular, cost-based optimization requires physical algebra, as then only costs can be estimated.

Optimizing Simple Queries

The Select-Project-Join queries (SPJ) are relatively simple yet commonly used. They correspond to SQL queries of the form SELECT ... FROM ... WHERE ... without any nested queries. They can be answered by using only selections (σ), joins/cross-products ($\bowtie/\times$), and projections (Π). Nevertheless, it has been shown that finding the optimal execution plan is NP hard in general, even for these simple queries. Several simplification are commonly used to reduce the optimization time. As a first step, the optimization concentrates on the join operators. The projections can be added as needed, i.e., whenever an operator materializes its input (and thus breaks the processing pipeline), all attributes that are no longer needed are projected away. Selections are added greedily, i.e., a selection is applied as early as possible. The rationale is that most selection predicates are cheap to evaluate and the selections reduce the work required by the following operators. Thus the optimizer only has to order the join/cross-product operators, and the other operators are added greedily. Unfortunately the problem remains NP hard even with this simplification.

The problem of finding the optimal join order can be seen as finding the optimal binary tree whose leaves correspond to the relations in the *from* clause. The inner nodes are joins or cross products, depending on available predicates suitable for a join, and are determined implicitly by the relations involved in their subtrees. Therefore, only the structure of the binary tree and the labeling of the leaf nodes has to be specified by the query optimizer. However the number of binary trees with n leaf nodes is $\mathcal{C}(n-1)$, where $\mathcal{C}(n)$ are the Catalan Numbers, which grow in the order of $\Theta(4^n/n^{\frac{3}{2}})$. As this grows very fast, some approaches reduce the search space by considering only a limited set of binary trees. A popular restriction is the limitation to *left-deep* join trees (or more general to *linear* join trees). A linear join tree is a join tree where only one of the two subtrees of a join operator may contain other join operators. If only the left subtree may contain other join operators, the trees are called left-deep. Figure 2 shows an example. General join trees without restrictions are called *bushy* join trees. Left-deep join trees are attractive (e.g., System R used them), as they are potentially easier to execute and the number of left-deep join trees is much smaller than the number of bushy join trees. As there are $n!$ ways to label the leaf nodes of the join tree for n relations, there are $n!\mathcal{C}(n-1)$ bushy trees, but "only" $n!$ left-deep trees. Unfortunately the optimal join tree can be a bushy tree, and these cases are not uncommon, which means that generating only left-deep trees can hurt query execution performance. Most modern query optimizers therefore construct bushy join trees.

The huge factor of $n!$ is caused by the fact that all combinations of relations are considered valid. The search space can be significantly reduced by avoiding the creation of cross products. When combining two relations, the optimizer can either use a cross-product, or use a join if there is a suitable join predicate in the *where* condition. Joins are much more efficient than cross-products, and although it is sometimes beneficial to use cross-products between separate relations, these cases are rare. When allowing cross-products, any relations can be combined, otherwise combinations are only possible if a suitable join predicate exists. The join possibilities implied by the query are captured in the query graph, as shown in Fig. 3. The relations from the *from* clause form the nodes, while the potential join conditions form the edges. Now the optimizer only has to consider sub-problems, i.e., sets of relations, that are connected in the query graph (this assumes that the



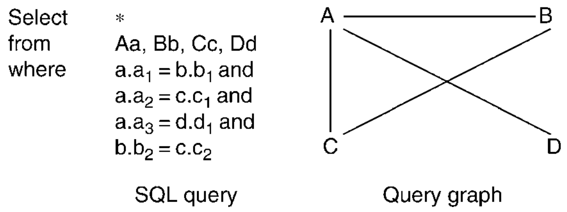
Query Optimization (in Relational Databases). Figure 2. Left-deep Versus bushy join trees.

query graph itself is connected, which can be guaranteed by adding additional edges). For example the relations A,B,C can be joined and thus could be part of a join tree, while no join tree will consist only of B,C,D, as this would require a cross-product. The structure of the query graph greatly affects the size of the search space. If the query graph forms a chain, the join ordering problem is no longer NP hard and can be solved in $O(n^3)$. If the query graph forms a clique, the problem is still NP hard (and just as difficult as when including cross products). Most queries are between those two extremes, and more like a chain than like a clique, and can thus be optimized much more efficiently by avoiding cross products.

Putting these observations together, SPJ queries can be optimized by the following strategy:

1. The only optimization decision is the join order, selections and projections are added greedily
2. The relations in a join tree must be connected in the query graph
3. When constructing left-deep trees, the right hand side of a join must contain only one relation.

The seminal System R paper on query optimization [9] introduced a dynamic programming (DP) strategy to optimize the join order. A slightly generalized version that generates bushy trees is shown in Fig. 4. It computes the optimal join order of the relations $R_1,\ldots,R_n$ (ignoring selections and projections for a moment). The basic strategy is to construct solutions for larger (sub-)problems (i.e., problems involving more relations) from optimal solutions of smaller problems. For example, consider the top-most join in a join tree with four relations. It will either combine two join trees with two relations each or a single relation with a join tree with three relations. The problem of joining four relations can thus be expressed as combining smaller problems with one to three relations each. Accordingly, the DP table is first organized by the size (i.e., number of relations) of a problem. For a given size, the table stores the optimal execution plan for a given set of relations. In lines 1–2 the algorithm initializes the DP table by adding table scans as the optimal solution for problems involving a single relation. The loop starting in line 3 now creates solutions of size s by combining smaller solutions. Lines 4–5 find all pairs of problems already solved (S_1, S_2) that have a combined size of s. If S_1 and S_2 overlap (i.e., they have relations in common) the pair is ignored, as no valid join tree can be constructed (line 6). Similarly (S_1, S_2) is ignored if S_1 and S_2 are not connected in the query graph to avoid the creation of cross-products (line 7). Otherwise it is possible to construct a new execution plan p that joins the known solutions for S_1 and S_2 (line 8). If no solution for $S_1 \cup S_2$ is known, (or the estimated costs of the new plan are less than for the currently known solution), p is added as a solution of size s for $S_1 \cup S_2$ in the DP table (lines 9–10). At the end of the algorithm the DP table entry for size n contains the optimal solution (line 11).



```
Select  *
from    Aa, Bb, Cc, Dd
where   a.a1 = b.b1 and
        a.a2 = c.c1 and
        a.a3 = d.d1 and
        b.b2 = c.c2
```

Query Optimization (in Relational Databases). Figure 3. A query and its query graph.

```
DPsize($R = \{R_1,\ldots,R_n\}$)
for each $R_i \in R$
    dpTable[1][$\{R_i\}$] = $R_i$
for each $1 < s \le n$ ascending // size of plan
    for each $1 \le s_1 < s$ // size of left subplan
        for each $S_1 \in$dpTable[$s_1$], $S_2 \in$dpTable[$s - s_1$]
            if $S_1 \cap S_2 \neq \emptyset$ continue
            if $\neg(S_1$ connected to $S_2)$ continue
            $p$ = dpTable[$s_1$][$S_1$]$\bowtie$dpTable[$s - s_1$][$S_2$]
            if dpTable[$s$][$S_1 \cup S_2$]=$\emptyset$ V cost($p$)<cost(dpTable[$s$][$S_1 \cup S_2$])
                dpTable[$s$][$S_1 \cup S_2$] = p
return dpTable[$n$][$\{R_1,\ldots,R_n\}$]
```

Query Optimization (in Relational Databases). Figure 4. Dynamic programming strategy for SPJ queries.

The algorithm in Fig. 4 is simplified, as it ignores selections and projections. They can be added greedily in lines 2 and 8 and do not affect the algorithm otherwise. A more complex part that is missing is the selection of the physical join operator. Line 8 simply states ⋈, but in a real system, there are multiple join operators available, typically at least nested-loop join, hash join, and sort-merge join. Lines 8–10 should therefore loop over the different join implementations and try all of them. What complicates this choice is that the different implementations behave differently, in particular the sort-merge join. When the input has to be sorted, the sort-merge join is relatively expensive, but when it is already sorted it is very cheap. And the output of a sort-merge join is itself sorted, which can render a following sort-merge join cheap if the order can be reused. Tuple orders that could be used by other operators are called *interesting orderings*, and the set of interesting orderings for the current query is computed before starting the *plan generation* (i.e., the DP algorithm, which constructs execution plans). During plan generation, plans that are more expensive than others have to be preserved, but provide an interesting order the others do not. This can be generalized by the concept of *physical properties*. A physical property is a characteristic of a plan that affects its runtime behavior (i.e., the costs of subsequent operators), but not its logical equivalence. The physical properties define a partial order between plans, describing which plans satisfy "more" properties. For example a plan with sorted output dominates another plan with unsorted output (concerning the physical property "ordered"), while two plans with differently sorted output are not directly comparable. During plan generation, a plan only dominates another plan if both the physical properties are dominating and the estimated costs are lower. As a consequence, the DP table entries no longer consist of single optimal plans, but of sets of plans, in which no plan dominates the other. Note that physical properties are an example for re-establishing the principle of optimality required for dynamic programming, which is also required in other contexts.

More Complex Queries

While Selection-Projection-Join queries are a important class of queries, queries can be much more complex. When only optimizing (inner) joins, the joins are freely reorderable, which means that any join order is valid as long as syntax constraints are satisfied (i.e., the relations required for the join predicates are available). This is no longer the case for other operators like outer joins and aggregations. A simple approach to handle these is to split the query into blocks that are freely reorderable (e.g., above and below an outer join) and to optimize the blocks individually. But this is too restrictive, and outer joins still allow for some reorderings, which have been described as algebraic equivalences. The challenge is to incorporate these equivalences into a cost-based (and potentially constructive) query optimizer. For outer joins, it has been shown how the possible reorderings can be expressed as dependencies on input relations [8]. The algorithm analyzes the original query and computes the set of relations that have to be part of a join tree before a specific outer join is applicable. Using this information the outer joins can be integrated easily, the plan generator just has to check the additional requirements before inserting an outer join. The main difficulty is computing this dependency information, but once it is available, the optimization is relatively simple. Other operators like aggregations are more difficult to integrate. Aggregations can be moved down a join if the join itself does not affect the aggregation results (e.g., a 1:1 join involving the grouping attribute where the join behaves like a selection). Pushing the aggregation down can reduce the effort for the join itself and might thus be beneficial. But in cases where the aggregation can simply be moved are relatively limited. Here, a more general movement of aggregations is possible by allowing for compensation actions. These are computations added to the plan that compensate the fact that the aggregation is performed at a different position. Adding these kinds of optimizations into a cost-based, constructive query optimizer is very challenging, which is why they are usually implemented as heuristical rewrite operations.

Another important aspect of optimizing complex queries is unnesting nested queries and the related problem of view resolution. The SQL query language allows for nesting queries inside other queries, either explicitly by including a nested *select* block, or implicitly by accessing a view. The nested query could be optimized independently from the outer query, and then during the optimization of the outer query treated like a base relation with specific costs. But this will often lead to inferior plans, for example if selection predicates from the outer query could be pushed down into a view. Instead, the optimizer tries to merge the nested query with the outer query into one flat query, which is then optimized in one step. Perhaps even more important than the unified optimization is a decoupling between the nested query and the outer query. The evaluation of the nested query can depend on the attributes of the outer query, which implies a very expensive nested-loop evaluation. In many cases it is possible to unnest these queries such that the nested (and apparently dependent) part can be evaluated independently, and then joined appropriately with the outer part of the query [9]. These optimizations greatly improve the evaluation of nested queries.

Key Applications

Query optimization techniques can improve the query execution time by orders of magnitude. All modern relational database systems therefore implement at least some optimization techniques.

Cross-references

- ▶ Parallel data placement
- ▶ Parallel database management
- ▶ Parallel query execution algorithms
- ▶ Parallel query processing

Recommended Reading

1. Chaudhuri S. An overview of query optimization in relational systems. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 34–43.
2. Garcia-Molina H., Ullman J.D., and Widom J. Database system implementation. Prentice-Hall, 2000.
3. Graefe G. The cascades framework for query optimization. Q. Bull. IEEE TC on Data Engineering, 18(3):19–29, 1995.
4. Graefe G. and McKenna W.J. The volcano optimizer generator: Extensibility and efficient search. In Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 209–218.
5. Haas L.M., Freytag J.C., Lohman G.M., and Pirahesh H. Extensible query processing in starburst. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989, pp. 377–388.
6. Moerkotte G. Building query compilers, available at http://db.informatik.uni-mannheim.de/moerkotte.html.en, 2006.
7. Muralikrishna M. Improved unnesting algorithms for join aggregate SQL Queries. In Proc. 18th Int. Conf. on Very Large Data Bases, 1992, pp. 91–102.
8. Rao J., Lindsay B.G., Lohman G.M., Pirahesh H., and Simmen D.E. Using EELS, a practical approach to outerjoin and antijoin reordering. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 585–594.
9. Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., and Price T.G. Access path selection in a relational database management System. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.

Query Optimization for Multidimensional Systems

▶ Query Processing in Data Warehouses

Query Optimization in Distributed Database Systems

▶ Distributed Query Optimization

Query Optimization in Sensor Networks

Kian-Lee Tan
National University of Singapore, Singapore, Singapore

Synonyms

In-network query processing

Definition

Query optimization is the process of producing a query evaluation plan (QEP) for a query that minimizes or maximizes certain objective functions. A query in a sensor network has additional clauses that specify the life time of a query, the frequency in which the sensor data should be monitored, and even the rate in which query answers should be returned. As such, the query plan must reflect these. In addition, a typical query plan comprises two main components: a communication component that sets up the communication structure for data delivery, and a computation component that performs the operation in the sensor network and/or the root node. Because sensor nodes are low-powered, besides minimizing computation cost, optimization criterion include minimizing energy consumption (e.g., by minimizing transmission cost) or maximizing the lifespan of the entire sensor network. As such, the cost model must consider these various factors.

Historical Background

Query optimization has always been an important area of research in database query processing. This is because a poorly chosen query execution plan can result in significant waste of resources, and more importantly, user dissatisfaction. In sensor network, the problem is more critical because of the resource constrains of the sensor nodes which are limited in computation power, bandwidth, memory, and energy.

Most of the existing works focused on in-network aggregation [2,3,5,6,10]. While work in [3,10] aim at precise answers, work in [2,6] assume errors (approximations) can be tolerated. In particular, the work in [6] exploits the tradeoff between data quality and energy consumption to extend the lifetime of the sensor network. Both the Cougar project [10] and the TinyDB project [3], in their prototype design of a sensor network management system, offer broader insights into query processing and optimization issues.

Foundations

To support applications in sensor networks, the database community has viewed the sensor network as a database [3,10]. This provides a good logical abstraction for sensor data management. Users can issue declarative queries without having to worry about how the data are generated, processed, and transferred within the network, and how sensor nodes are (re)programmed. As such, query optimization techniques can also be applied to optimize the network operations.

Queries

Queries are typically expressed using an extended SQL that include additional clauses that specify the duration and sampling rates. As an example, the following query counts the number occupied nests in each loud region of a certain island [3,10]:

```
SELECT region, CNT(occupied), AVG
(sound)
FROM sensors
GROUP BY region
HAVING AVG(sound) > 200
SAMPLE PERIOD 10 FOR 3,600
```

Here, the clause "SAMPLE PERIOD ... FOR" indicates that the sensors will sample the environment every 10 seconds for a duration of 3,600 seconds.

Query Plan

To evaluate a query, a query plan has to be generated for the sensors. A query plan specifies the role of each sensor (the computation to be performed, the rate at which it should sample the data), and the communication structure between sensors. For the sample query above, three alternative plans can be considered: (i) Each sensor samples its data every 10 seconds, and then transmits the data back to the base station; at the base station, it will perform the grouping and aggregate computation as in a centralized system. (ii) A sensor within a region is selected as a cluster head (CH); all sensors within the region send their sampled data to the corresponding CH; the CH performs the aggregate and sends it back to the base station if it satisfies the HAVING clause (i.e., >200). (iii) This is similar to Plan 2, except that sensors within a region can be further hierarchically partitioned so that each partition has a leader that performs partial aggregation of the sampled data from its child nodes.

For Plan 1, each sensor node takes on at most two roles: (i) sample the data and transmit the sampled value back to the base station, and (ii) relay the data it receives from its child nodes if it is an internal node along the path to the base station. For Plan 2, the CH has the additional computation task of performing the aggregate and determining whether it should be routed back to the base station. For Plan 3, leaders have the responsibility to perform partial aggregates.

All the plans may also specify the sensor nodes in which one should be transmitting data to and/or receiving data from, i.e., the communication structure.

Clearly, there are many other possible plans that can be generated, and it is the optimizer's task to pick the one that best suits the objectives.

Metadata

The optimizer makes its decision based on certain metadata. For example, the metadata for each sensor include the static information about its location (region), the sensor types, the amount of memory, the energy consumption per operation type, the energy consumption per sampling per sensor type, and so on. It may also be necessary to maintain dynamic statistics on a sensor's estimated (remaining) battery life, and the selectivity of query predicates. These metadata are periodically collected (via the routing tree) by the optimizer, and used in several ways. For example, for sensor nodes with short battery life, the sampling rate may be adjusted so that they can operate in a doze mode longer in order to extend the lifespan of the entire network. As another example, nodes with long battery life may be selected as the cluster heads or leaders, nodes with moderate amount of energy can relay messages, and nodes with low battery power will simply transmit their sampled data. As such, it is possible to balance the energy across all nodes. As another example, knowing the energy consumption per sampling per sensor type may allow the optimizer to reorder predicates on different sensor types.

Cost Model

For each query plan, the optimizer estimates the cost of the plan. The cost model depends on the objective function. In a sensor network environment, there are two key metrics: energy consumption and transmission cost (as communication consumes most energy).

As an example, consider an arbitrary sensor node along the path of the routing tree. Let N be the number of sensor types and k be the number of predicates. Let E_i is the cost to sample sensor type i, E_{trans} and E_{recv} be the energy to transmit and receive a message respectively, E^i_{pred} be the energy consumed to evaluate predicate i, and E_{agg} be the energy to compute a (partial) aggregate, and C be the number of child nodes routing through this node. The energy at a node s to collect a sample, and transmit its partial aggregate, including the costs to forward data, can be estimated as follows:

$$e_s = \sum_{i=0}^{N} E_i + \sum_{i=0}^{N} E^i_{pred} + E_{recv} \times C + E_{agg} + E_{trans}$$

The energy consumed by the node is the cost to read all the sensors at the node (first component), plus the cost to evaluate the predicates (second component), plus the cost to receive partial aggregates from its child nodes (third component), plus the cost to compute a partial aggregate (fourth component), plus the cost to transmit the partial aggregate (fifth component).

Suppose sensor s has a remaining battery capacity of B_s Joules. Then, s has enough power to last B_s/e_s sample collections. To extend the lifespan of the system, a plan that maximizes this value has to be determined. Let there be P plans, then a plan that maximizes the minimim number of sample collections is the one that should be selected (to maximize the lifespan of the system), i.e.,

$$max^P_{i=1}(min^N_{j=1} B_j/e_j)$$

Depending on different objectives and model, similar cost models can be derived.

Centralized Optimization

Most of the existing work on optimizing sensor network queries are based on a centralized optimizer, i.e., the optimizer at the base station determines the query plan [3,10]. There are two dimensions in which query processing can be optimized. The first dimension deals with the grouping of the sensor nodes to support in-network computation (e.g., aggregates). Essentially, the optimizer enumerates the possible alternative plans based on different ways in which nodes are grouped. Some heuristics are:

1. On one extreme, all nodes belong to a single group;
2. On another extreme, each node forms a single group;
3. Between the two extremes, different group-based heuristics can be adopted: proximity-based schemes cluster nodes that are close-by together, semantic-based schemes cluster nodes that are semantically related, e.g., same set of sensor types, same resources, same metadata, etc.

The second dimension covers heuristics that are used to reorder the operations of a query. Some of these are:

1. When a sensor node is required to sample multiple attributes (temperature, humidity, light) and the query involves a number of predicates, different orderings of the samping and predicate evaluations may result in different energy consumption. This is the case because sampling consumes more power than evaluating a predicate. As such, to conserve energy, it makes sense to order selective predicates first before the non-selective predicates; in addition, data should be sampled only when necessary. Intuitively, once a sampled value is discarded (because it does not satisfy a predicate), there is no need to expend energy to sample other attributes. Thus, the optimizer enumerates different sequences of sampling and predicate evaluations to find the one that is most energy efficient. As an example, consider a query that finds sensor nodes whose accerlerometer and magnetometer readings exceed thresholds a1 and m1 respectively. As the magnetometer consumes 50 times more energy than the accerlerometer to sample a reading, if the selectivity of the predicate on the accerlerometer reading is low, then it makes sense to sample the accerlerometer first. Moreover, the magnetometer should only be sampled when the accerlerometer reading is larger than a1.
2. For certain event monitoring queries, they can be rewritten into a window join queries that can be more efficiently processed in the network.

Distributed Optimization

As centralized optimization schemes require certain metadata that must be periodically obtained from the sensor nodes, it may be costly (in terms of power) to collect this information. An alternative approach is to perform distributed optimization [4]. Here, the basic idea is to identify clusters of nodes with its associated cluster head (CH) so that these CHs optimize the queries for processing within the cluster. Thus, the base stations disseminate the query to the CHs, each of which optimizes the query based on the local metadata and controls the processing within the cluster.

With distributed optimization, the metadata is collected at the CH, rather than the base station. This means that the transmission overhead for metadata is reduced. Moreover, "local" metadata are more accurate as some globally collected metadata may be too coarse (e.g., distribution of data). However, the CHs incur the overhead of optimizing the query.

Here, the challenge is to determine the clusters (either spatially or semantically). In spatial-based clustering scheme, the number of groups are based on the radius of a group, which is the number of hops between a non-CH node and its CH. In the semantic-based clustering scheme, nodes with similar netadata are formed into groups.

Key Applications

Sensor databases can be applied in many applications, e.g., environmental monitoring, military surveillance. To process queries in these applications, it is necessary to optimize the queries in order to ensure that resources are well utilized, and the system lifespan can be sufficiently large to minimize any need to replace the batteries regularly.

Future Directions

Most of the existing work in the literature focused on optimizing aggregate queries one at a time. Recently, join queries have been studied. In [1], static join queries are considered, and in [9], continuous join queries are examined. In both cases, join processing is pushed into the sensor network. In the former, sensor nodes are grouped so that a static table can be partitioned across them; new sampled records are broadcast within a group, and only answers need to be sent back to the base stations. In the latter, the focus is on minimizing the number of subqueries that need to be injected into the sensor network. As sensor nodes become more powerful, there is much opportunity for optimizing complex queries that have not been previously studied. Another promising direction that has recently received attention is the multi-query optimization problem [7,8]. Here, multiple queries submitted by users are optimized collectively to exploit commonality among them so that any redundant data accesses from the sensors can be eliminated. Current focus is on non-join queries. Extending these schemes to handle more complex queries is an interesting direction to explore. Yet another direction is to consider a large scale sensor network deployment that involves multiple base stations. Here, the challenge is to optimize the network lifespan as well as the load across the base stations.

Cross-references

- ▶ Continuous Queries in Sensor Networks
- ▶ Data Aggregation in Sensor Networks
- ▶ In-Network Query Processing
- ▶ Sensor Networks

Recommended Reading

1. Abadi D.J., Madden S., and Lindner W. REED: robust, efficient filtering and event detection in sensor networks. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 769–780.
2. Deligiannakis A., Kotidis Y., and Roussopoulos N. Hierarchical in-network data aggregation with quality guarantees. In Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 658–675.
3. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. TINYDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst., 30(1):122–173, 2005.
4. Rosemark R. and Lee W.-C. Decentralizing query processing in sensor networks. In Proc. 1st Annual Int. Conf. on Mobile and Ubiquitous Syst., 2005, pp. 270–280.
5. Silberstein A. and Yang J. Many-to-many aggregation for sensor networks. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 986–995.
6. Tang X. and Xu J. Extending network lifetime for precision constrained data aggregation in wireless sensor networks. In Proc. 25th Annual Joint Conf. of the IEEE Computer and Communications Societies, 2006.
7. Trigoni N., Yao Y., Demers A.J., Gehrke J., and Rajaraman R. Multi-query optimization for sensor networks. In Proc. 1st IEEE Int. Conf. on Dist. Comput. in Sensor Syst., 2005, pp. 307–321.
8. Xiang S., Lim H.B., Tan K.L., and Zhou Y. Two-tier multiple query optimization for sensor networks. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2007, p. 39.
9. Yang X., Lim H.B., Ozsu T., and Tan K.L. In-network execution of monitoring queries in sensor networks. In Proc. ACM SIGMOD Int. Conf. on Management of Data., 2007, pp. 521–532.
10. Yao Y. and Gehrke J. Query processing in sensor networks. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.

Query Parallelism

► Intra-query parallelism

Query Plan

Evaggelia Pitoura
University of Ioannina, Ioannina, Greece

Synonyms

Query plan; Query execution plan; Query evaluation plan; Query tree; Operator tree

Definition

A query or execution plan for a query specifies precisely how the query is to be executed. Most often, the plan for a query is represented as a tree whose internal nodes are operators and its leaves correspond to the input relations of the query. The edges of the tree indicate the data flow among the operators. The query plan is executed by the query execution engine.

Key Points

During query processing, an input query is transformed into an internal representation, most often, into a relational algebra expression. To specify how to evaluate a query precisely, each relational operator is then mapped to one or more physical operators, which provide several alternative implementations of relational operators.

A query plan for each query specifies the physical operators to be used for its execution and the order of their invocation. Most commonly, a query plan is represented as a tree. The leaf nodes of the query tree correspond to the input (or base) database relations of the query and its internal nodes to physical operators. The edges of the tree specify the data flow among the operators. Each operator receives input from its child nodes in the tree and, in turn, its output is used as input to its parent node.

The query optimizer enumerates alternative query plans for each query and selects the most efficient among them using a cost estimation model. The selected query plan is then passed for execution to the query execution engine that results in generating answers to the query.

Query plans can be divided into prototypical shapes and query execution engines can be divided into groups based on which shapes of plans they can evaluate. Such shapes include left-deep, right-deep and bushy plans. Deep plans are plans in which each join involves at least one base relation, whereas bushy plans are more general in that a join could involve one or two base relations or the results of one or two other join operations. Finally, for queries with common subexpressions, the query evaluation plan may be an acyclic directed graph (DAG) instead of a tree.

Cross-references

► Evaluation of Relational Operators
► Query Optimization
► Query Processing

Recommended Reading

1. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73–170, 1993.
2. Ramakrishnan R. and Gehrke J. Database Management Systems. McGraw-Hill, New York, 2003.

Query Planning and Execution

► Query Processing and Optimization in Object Relational Databases

Query Point Movement Techniques for Content-Based Image Retrieval

Kien A. Hua, Danzhou Liu
University of Central Florida, Orlando, FL, USA

Definition

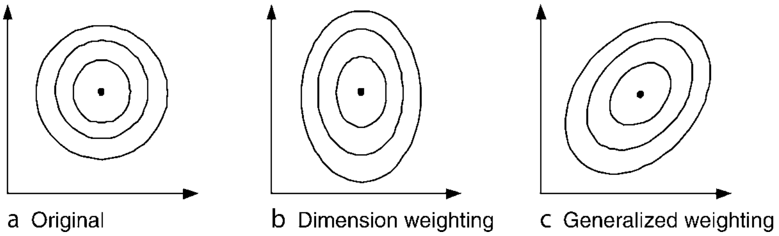
Target search in content-based image retrieval (CBIR) systems refers to finding a specific (target) image such as a particular registered logo or a specific historical photograph. To search for such an image, query point movement techniques iteratively move the query point closer to the target image for each round of the user's relevance feedback until the target image is found. The goals of query point movement techniques include avoiding local maximum traps, achieving fast convergence, reducing computation overhead, and guaranteeing to find the target.

Historical Background

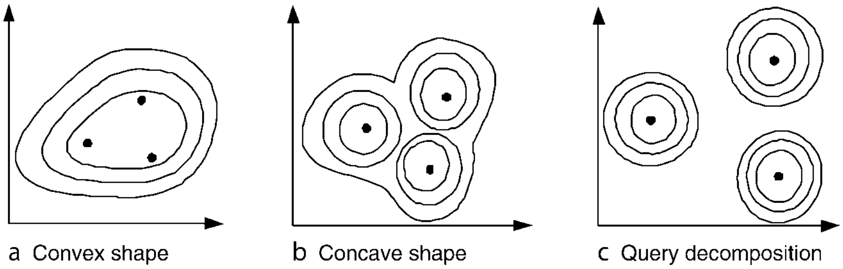
Images in a database are characterized by their visual features, and represented as points in a multidimensional feature space. A *query point* is one of these image points, selected to find similar images represented by image points nearest to the query point in the feature space. This cluster of nearby or relevant image points has a shape (see Figs. 1 and 2) referred to as the *query shape*.

For each iteration, a query point movement technique attempts to move the query point closer to the target image by refining the query based on user's relevance feedback. Existing query point movement techniques can be divided into two categories: single-point and multiple-point movement techniques. A technique is classified as a single-point movement technique if the refined query Q_r at each iteration consists of only one query point; otherwise, it is a multiple-point movement technique. In the latter category, the query result is the set of images nearest to the set of query points. Typical query shapes of single-point movement and multiple-point movement techniques are illustrated in Figs. 1 and 2, respectively, where the contours represent equi-similarity surfaces. Single-point movement techniques, such as MARS [9] and MindReader [5], construct a single query point close to relevant images and away from irrelevant ones. MARS uses a weighted distance (producing shapes as shown in Fig. 1b), where each dimension weight is inversely proportional to the standard deviation of the relevant images' feature values in that dimension. The rationale is that a small variation among the values is more likely to express restrictions on the feature, and thereby should carry a higher weight. On the other hand, a large variation indicates that this dimension is not significant in the query, and should thus assume a lower weight. MindReader achieves better results by using a generalized weighted distance, see Fig. 1c for its shape.

In multiple-point movement techniques such as Query Expansion [1], Qcluster [6], and Query Decomposition [4], multiple query points are used to define the ideal space that is most likely to contain relevant results. Query Expansion groups query points into clusters and choose their centroids as the representatives of the query Q_r (see Fig. 2a). The distance of a point to Q_r is defined as a weighted sum of individual distances to those representatives. The weights are proportional to the number of relevant objects in the clusters. Thus, Query Expansion treats local clusters differently, compared to the equal treatment in single-point movement techniques. In some queries, clusters



Query Point Movement Techniques for Content-Based Image Retrieval. Figure 1. Single-point movement query shapes.



Query Point Movement Techniques for Content-Based Image Retrieval. Figure 2. Multiple-point movement query shapes.

are too far apart for a unified, all-encompassing contour to be effective; separate contours can yield more selective retrieval. This observation motivated Qcluster to employ an adaptive classification and cluster-merging method to determine optimal contour shapes for complex queries. Qcluster supports disjunctive queries, where similarity to any of the query points is considered relevant (see Fig. 2b). To bridge the semantic gap more effectively, A Query Decomposition technique was presented in [4]. Based on user's relevance feedback, this scheme automatically decomposes a given query into localized subqueries, which more accurately capture images with similar semantics but with very different appearance (see Fig. 2c).

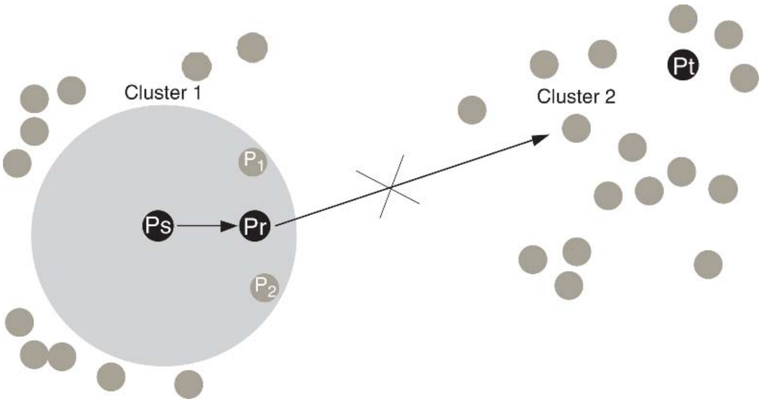
Standard query point movement techniques, explained above, allow re-retrieval of previously determined relevant images when they fall in the search range again. This leads to two major disadvantages:

1. *Local maximum traps*. Since query points in relevance feedback systems have to move through many regions before reaching a target, it is possible that they get trapped in one of these regions. Figure 3 illustrates a possible scenario where p_s and p_t denote the starting query point and the target point, respectively. As a result of a 3-NN search at p_s, the system returns points p_1 and p_2, in addition to query point p_s. Since both p_1 and p_2 are relevant, the refined query point p_r is their centroid and the anchor of the next 3-NN search. In this situation, the system will retrieve exactly the same set; from which, points p_1 and p_2 are again selected. In other words, the system can never get out of the subspace because the retrieval set is saturated with the k selected images. Although the system can escape with a larger k, it is difficult to guess a proper threshold. Consequently, the user might not even know a local maximum trap is occurring, and there is no guarantee to find the target image.
2. *Slow convergence*. The centroid of the relevant points is typically selected as the anchor of refined queries. This, coupling with possible retrieval of already visited images, prevents aggressive movement of the search process (see Fig. 4, where $k = 3$). Slow convergence incurs longer search time, and significant computation and disk access overhead.

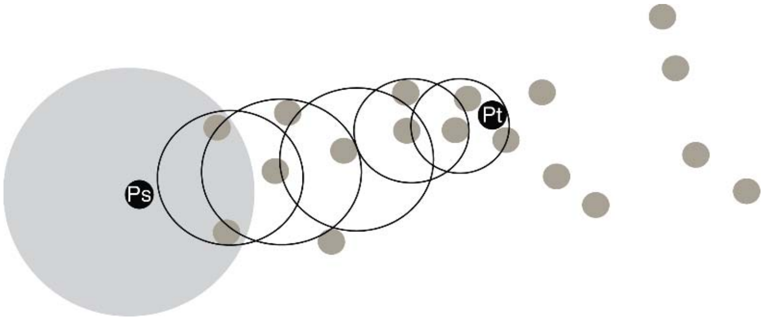
Foundations

To address the limitations of standard query point movement techniques, four target search methods have been proposed [8]: Naïve Random Scan (NRS), Local Neighboring Movement (LNM), Neighboring Divide and Conquer (NDC), and Global Divide and Conquer (GDC) methods. All these methods are designed around a common strategy: they do not retrieve previously selected images (i.e., shrink the search space). Furthermore, NDC and GDC exploit Voronoi diagrams to aggressively prune the search space and move towards the target image faster.

More formally, a query in target search is defined as $Q = \langle n_Q, P_Q, W_Q, D_Q, \mathbb{S}, k \rangle$, where n_Q denotes the number of query points in Q, P_Q the set of n_Q query points in the search space $\mathbb{S}$, W_Q the set of weights associated with P_Q, D_Q the distance function, and k the number of points to be retrieved in each iteration. Using these notations, the four target search techniques can be described as follows:



Query Point Movement Techniques for Content-Based Image Retrieval. Figure 3. Local maximum trap.



Query Point Movement Techniques for Content-Based Image Retrieval. Figure 4. Slow convergence.

Naïve Random Scan Method (NRS)

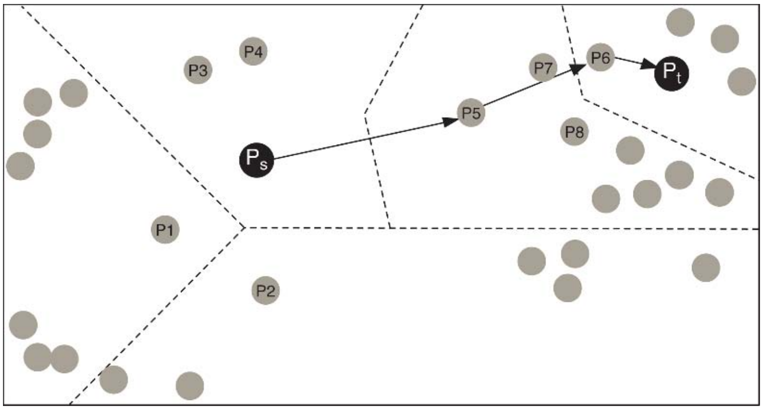
This method randomly retrieves k different images at a time until the user finds the target image or the remaining set is exhausted. Specifically, at each iteration, a set of k not previously selected images are randomly retrieved from the candidate set $\mathbb{S}'$ for relevance feedback, and $\mathbb{S}'$ is then reduced by k for the next iteration. Clearly, this strategy does not suffer local maximum traps and is able to locate the target image after some finite number of iterations. In the best case, NRS takes one iteration, while the worst case requires $\left\lceil \frac{|\mathbb{S}|}{k} \right\rceil$. On average, NRS can find the target in $\left\lceil \sum_{i=1}^{\left\lceil \frac{|\mathbb{S}|}{k} \right\rceil} i / \left\lceil \frac{|\mathbb{S}|}{k} \right\rceil \right\rceil = \left\lceil (\left\lceil \frac{|\mathbb{S}|}{k} \right\rceil + 1)/2 \right\rceil$ iterations. In other words, NRS takes $\mathcal{O}(|\mathbb{S}|)$ to reach the target point. Therefore, NRS is only suitable for a small database set.

Local Neighboring Movement Method (LNM)

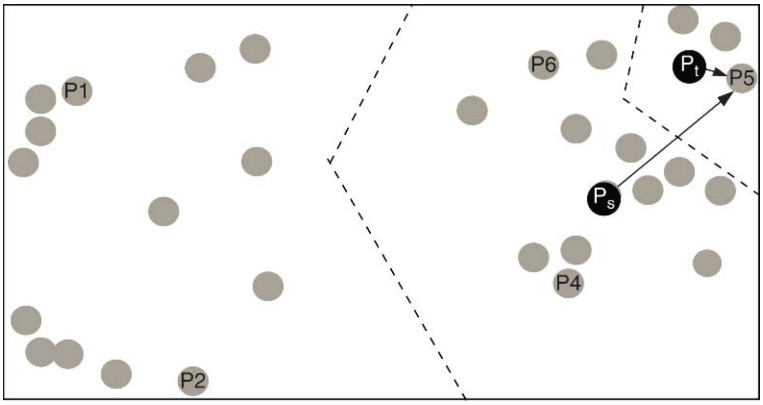
This method applies the non-re-retrieval strategy to MindReader [5]. Specifically, Q_r is constructed so that it moves towards neighboring relevant points and away from irrelevant ones, and k-NN query is now evaluated against $\mathbb{S}'$ instead of $\mathbb{S}$. When LNM detects a local maximum trap, it requests that the user selects the most relevant image. This way, LNM can overcome local maximum traps, although it could take many iterations to do so. Again, one iteration is required in the best case. If data is uniformly distributed in the n-dimensional hypercube, the worst and average cases are $\left\lceil \sqrt{n}\sqrt[n]{|\mathbb{S}|}/\lceil \log_{2^n} k \rceil \right\rceil$ and $\left\lceil (\frac{\sqrt{n}\sqrt[n]{|\mathbb{S}|}}{\lceil \log_{2^n} k \rceil} + 1)/2 \right\rceil$, respectively. If the data are arbitrarily distributed, then the worst case could be as high as that of NRS, i.e., $\left\lceil \frac{|\mathbb{S}|}{k} \right\rceil$ iterations (e.g., when all points are on a line). In summary, in the worst case LNM could take anywhere from $\mathcal{O}(\sqrt[n]{|\mathbb{S}|})$ to $\mathcal{O}(|\mathbb{S}|)$.

Neighboring Divide and Conquer Method (NDC)

Although LNM can overcome local maximum traps, it does so inefficiently, taking many iterations and in the process returning numerous false hits. To speed up convergence, Voronoi diagrams are used in NDC to reduce the search space after each round of relevance feedback. That is, the k-NN search in each iteration is performed only within the Voronoi cell containing the query point. It can be proved that the target image must reside in this Voronoi cell [8]. Since this strategy aggressively prunes the search space and moves rapidly towards the target image, it can overcome local maximum traps and achieve fast convergence. Figure 5 illustrates how NDC approaches the target after pruning the search space three times. In the first iteration, points p_1, p_2, and p_s are randomly chosen by the system, assuming $k = 3$; and they are used to construct a Voronoi diagram partitioning the search space into three regions. The user identifies p_s as the most relevant point (i.e., most similar to the target image). Since the target image must reside in the Voronoi cell containing p_s, the computation of the k-NN query anchored at p_s can be confined to this cell while the other two Voronoi regions can be safely ignored. This step retrieves three new nearest neighbors p_3, p_4, and p_5. Their Voronoi diagram further partitions the current search space into three regions. The user again correctly selects p_5 as the most relevant point, and therefore the query point for the third iteration. This refined query results in another set of relevant points p_6, p_7, and p_8; and another Vonronoi diagram is constructed. This time, the user selects p_6 as the most relevant image. Using it



Query Point Movement Techniques for Content-Based Image Retrieval. Figure 5. Example of NDC.

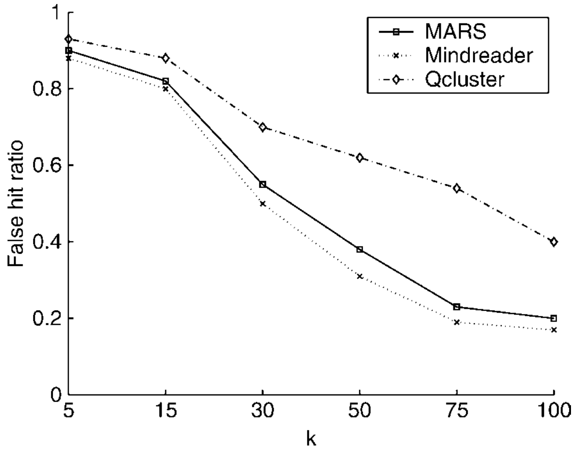


Query Point Movement Techniques for Content-Based Image Retrieval. Figure 6. Example of GDC.

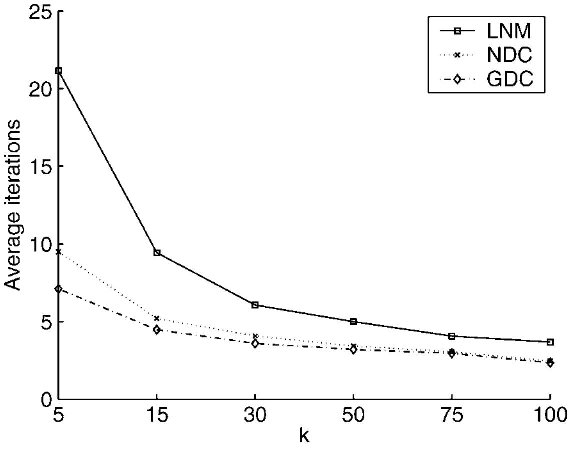
as the query point for the fourth iteration, the system returns three relevant points and the user identifies p_t as the target image. If the data points are uniformly distributed, NDC reaches the target point in no more than $\mathcal{O}(\log_k|\mathbb{S}|)$ iterations. When $\mathbb{S}$ is arbitrarily distributed, the worst case could take up to $\lceil\frac{\mathbb{S}}{k}\rceil$ iterations (e.g., all points are on a line), the same as that of NRS. In other words, NDC could still require $\mathcal{O}(|\mathbb{S}|)$ iterations to reach the target point in the worst case.

Global Divide and Conquer Method (GDC)

To reduce the number of iterations in NDC under the worst case scenario, GDC constructs the Voronoi diagram based on points randomly selected from the current search space, instead of using points from the query result. An example is given in Fig. 6. In the first iteration, a Voronoi diagram is constructed based on three randomly sampled points p_1, p_2, and p_s, assuming $k = 3$. The user selects p_3 as the most relevant point, and this results in p_4, p_5, and p_6 as the query result as computed in NDC. The user now selects p_5 as most relevant in the second iteration. The system randomly selects three points in the Voronoi cell associated with p_5, and uses them to further partition this cell into three smaller Voronoi cells. The computation of the k-NN query anchored at p_5 over the smaller Voronoi cell containing p_5 returns three new points, and the user identifies p_t as the target image in the third round. As proved in [8], the worst case for GDC is bounded by $\mathcal{O}(\log_k|\mathbb{S}|)$. This implies that for arbitrarily distributed datasets, GDC converges faster than NDC in general, although NDC might be as fast as GDC for certain queries, e.g., the starting query point is close to the target point. In the previous example (Fig. 5), NDC could also take three iterations, instead of four, to reach the target point if the initial k points were the same as in Fig. 6, as opposed to Fig. 5.



Query Point Movement Techniques for Content-Based Image Retrieval. Figure 7. False hit ratio.



Query Point Movement Techniques for Content-Based Image Retrieval. Figure 8. Average iterations.

Key Applications

Multimedia Search Engine, Crime Prevention, Graphic Design.

Future Directions

1. Incorporate information from the log file on user relevance feedback to determine the query results in each feedback iteration, instead of performing the traditional *k*-NN computation. This strategy can minimize the effect of the semantic gap between the low-level visual features and the high-level concepts in the images.
2. Multiple query points, as in standard query point movement techniques such as Query Expansion [1] and Qcluster [6], can be used in each iteration to better convey user's relevance feedback.
3. With the growing interest in Internet-scale image search applications, it is desirable to extend the target search techniques because it will enable concurrent users to share computation [7].

Experimental Results

Figure 7 shows that standard techniques MARS [9], MindReader [5], and Qcluster [6] have poor false hit ratio when *k* is small. This is due to the effect of local maximum traps. Even for fairly large *k*, their false hit ratios remain very high. As a result, users of these techniques have to examine a large number of returned images, but might still not find their intended targets. Figure 8 shows that NDC and GDC perform more efficiently when *k* is small, with GDC being slightly better than NDC. Specifically, when $k = 5$, the average numbers of iterations for LNM, NDC, and GDC are roughly 21, 10, and 7, respectively. Experimental studies based on a prototype [8] showed that only seven iterations on average were needed to locate a given target image. Additional performance results can be found in [8].

Data Sets

The data set consists of more than 68,040 images from the COREL library. Thirty-seven visual image features divided into three main groups were used: colors (9 features), texture (10 features), and edge structure (18 features).

Cross-references

- ▶ Content-based Image Retrieval (CBIR)
- ▶ Relevance Feedback
- ▶ Target Search

Recommended Reading

1. Chakrabarti K., Ortega-Binderberger M., Mehrotra S., and Porkaew K. Evaluating refined queries in top-k retrieval systems. IEEE Trans. knowledge and Data Eng., 16(2):256–270, 2004.
2. Cox I.J., Miller M.L., Minka T.P., Papathomas T.P., and Yianilos P.N. The Bayesian image retrieval system, PicHunter: theory, implementation, and psychophysical experiments. IEEE Trans. Image Processing, 9(1):20–37, 2000.
3. Flickner M., Sawhney H.S., Ashley J., Huang Q., Dom B., Gorkani M., Hafner J., Lee D., Petkovic D., Steele D., and Yanker P. Query by image and video content: The QBIC system. IEEE Computer, 28(9):23–32, 1995.
4. Hua K.A., Yu N., and Liu D. Query decomposition: A multiple neighborhood approach to relevance feedback processing in contentbased image retrieval. In Proc. 22nd Int. Conf. on Data Engineering, 2006.

5. Ishikawa Y., Subramanya R., and Faloutsos C. MindReader: Querying databases through multiple examples. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 218–227.
6. Kim D.-H. and Chung C.-W. Qcluster: relevance feedback using adaptive clustering for content-based image retrieval. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 599–610.
7. Liu D. and Hua K.A. Support concurrent queries in multiuser CBIR systems. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 1379–1381.
8. Liu D., Hua K.A., Vu K., and Yu N. Fast query point movement techniques with relevance feedback for content-based image retrieval. In Advances in Database Technology, In Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 700–717.
9. Rui Y., Huang T., Ortega M., and Mehrotra S., Relevance feedback: A power tool for interactive content-based image retrieval.IEEE Trans. Circuits Syst. Video Technol., 8(5):644–655, 1998.

Query Processing

EVAGGELIA PITOURA
University of Ioannina, Ioannina, Greece

Synonyms

Query evaluation

Definition

A query processor receives a query, validates it, optimizes it into a procedural dataflow execution plan and executes it to obtain the results of the query.

Key Points

Query processing consists of several phases. In the first phase, the *query parser* checks whether the query is correctly specified, resolves any names and references, verifies consistency, and performs authorization tests. If the query passes validation, it is converted into an internal representation that can be easily processed by the subsequent phases. Then, the *query rewrite module*, or *rewriter*, simplifies the query and transforms it into an equivalent form by carrying out a number of optimizations that do not depend on the physical state of the system, such as view expansion and logical rewriting of predicates. These initial phases rely only on data and metadata in the system catalog.

Next, the *query optimizer* transforms the internal query representation into an efficient query plan. It decides which indices to use, which methods to use for executing the query operators and in which order. A *query plan* may be thought of as a dataflow diagram that pipes data through a graph of query operators. The optimizer enumerates alternative plans and chooses the best plan using a cost estimation model that relies on various selectivity estimation techniques and available statistics regarding the physical state of the system. The *code generation component* transforms the plan produced by the optimizer into an executable plan. Finally, the *query execution engine* operates on the fully-specified query plan. It provides generic implementations for every operator. Most execution engines are based on the *iterator model*. In such a model, operators are implemented as iterators, with all iterators having the same interface [1,2,3].

Cross-references

► Evaluation of Relational Operators
► Iterator
► Query Optimization
► Query Optimization in Sensor Networks
► Query Plan
► Selectivity Estimation

Recommended Reading

1. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73–170, 1993.
2. Haas L.M., Freytag J.C., Lohman G.M., and Pirahesh H. Extensible query processing in starburst. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989, pp. 377–388.
3. Hellerstein J.M., Stonebraker M., and Hamilton J. Architecture of a database system. Foundations and Trends in Databases, vol. 1(2), 2007.

Query Processing (in Relational Databases)

VOLKER MARKL
IBM Almaden Research Center, San Jose, CA, USA

Synonyms

Query compilation and execution

Definition

Query processing denotes the compilation and execution of a query specification usually expressed in a declarative database query language such as the structured query language (SQL). Query processing consists of a compile-time phase and a runtime phase. At compile-time, the query compiler translates the query specification into an executable program. This translation process (often called *query compilation*) is comprised of lexical, syntactical, and semantical analysis of the query specification as well as a query optimization and code generation phase. The code generated usually consists of physical operators for a database machine. These operators implement data access, joins, selections, projections, grouping, and aggregation. At runtime, the database engine interprets and executes the program implementing the query specification to produce the query result.

Historical Background

In the 1960s and 1970s, navigational database management systems emerged. The hierarchical database management system, IMS from IBM is a prominent example that is still widely used. The Database Task Group developed the CODASYL approach to data management during the standardization of COBOL. The CODASYL approach focuses on following pointers that link a network of data records. CODASYL lacks a declarative query language. Query processing for CODASYL databases is mainly about efficient access to data, where queries are implemented as procedural programs by software developers. There was no query optimization in the CODASYL model.

In 1970, Edgar Codd invented a relational algebra over tables. This relational data model was much more flexible than navigational data models. Relational algebra allowed the specification of arbitrary queries over relations. A combination of five primitive operations (selection, projection, union, filter, Cartesian product) over tables define a query in the relational algebra. Initially, the relational model was considered to be impractical, as it required data to be normalized into relations, and queries were very expensive operations. Queries often have to use the join operation (a combination of a Cartesian product with a selection) in order to reconstruct normalized relations. However, over the years, many inventions (database indexes, join methods like nested-loop join, merge join or hash-join, and techniques for query optimization) have together made relational database management systems feasible.

The prime example of a relational query language is SQL, which is based on a logical calculus called *tuple calculus*. An SQL query specifies *what* data to access and process in order to compute a query result, but not *how* to access and process the data. An SQL query over a relational database can be implemented in many different ways. For complex queries on databases with many indexes and tables, there may be millions of different ways to implement an SQL query. Each implementation may use different orders for combining the normalized tables, or use different join methods like hash join or nested-loop join, or use different types of indexes like B-Trees or bitmap indexes, or using indexes on different columns to efficiently process a selection over a table. Choosing the right method for processing a query over a large database can produce a query result in seconds (or faster), whereas choosing of the wrong method can result in queries running for hours, or even days.

Query optimization aims at selecting the most efficient access path (often called *query execution plan*, or *plan*) for any given query. The task of a query optimizer is to find the most efficient overall implementation of the query. Query optimization has been an active area of research since the 1970s, with advances still being made today. Some standard optimizations are based on simple heuristics. A typical objective may be to avoid large intermediate results during query processing by applying selections as early as possible. However, determining the most efficient access method for each table as well as the best join methods and join orders to combine tables cannot be carried out by simple heuristics alone. In order to find the best plan, the query processor actually needs to know some characteristics of the data that the query will process. For example, a nested-loop join method works best if one of the two tables to be joined is relatively small. A hash join has higher overhead for small tables, but will produce the query result much faster if both tables are large. Similarly, it is good to use an index if a selection results in very few rows of a table. If many rows qualify, a table scan will be much faster, as it can use sequential I/O, avoiding the consecutive repositioning of the read/write head of a hard disk as required when processing an index.

In 1979, Pat Selinger proposed a cost-based query optimizer for System R which determines the optimal query plan based on a mathematical model of the execution cost of each operator in a query execution plan. The System R optimizer, which has become the basis for many commercial database systems (e.g., IBM's DB2 and Oracle), enumerates all possible physical query execution plans for a query in a stratified bottom-up fashion. This is done by first determining the cheapest table accesses, then processing all two-table joins, three-table joins, and so on, and uses dynamic programming to prune the search space. In 1994, volcano introduced an alternative approach of a top-down optimizer based on goal-directed search and branch-and-bound pruning, which has found its way into commercial DBMS like Microsoft's SQL Server.

During the 2000s, advanced techniques for query processing have been proposed and implemented into commercial systems. One of these developments relies on feedback loops to improve query execution. Feedback obtained by monitoring some parameters of the mathematical cost model during query execution is used to either alter the query execution plan while the query is running, or to adjust the mathematical model to increase its accuracy for subsequent queries. In addition, integration of semi-structured data (XML) and the Xquery language are active areas of research, with new requirements imposed on a query processor due to the navigational operators like XPath needed for processing hierarchical data.

Foundations

The figure below gives an architectural overview of a query processor. The query compiler of a relational database translates a declarative SQL query into a procedural program. Initially, a parser carries out tokenization and creates a parse tree of the query based on the grammar of the query language. Semantical analysis then tests semantical correctness of the query. Those tests usually validate if the table names or column names in the query exist and have the right types. These parsing steps are similar to the steps a programming language parser would carry out, except that the query compiler has to generate a program for a data flow engine, not for a microprocessor. The query is therefore usually represented as a data flow graph (also called *query graph*) in a query compiler. A query graph is a graph whose edges represent the data flow and whose nodes represent operations on the data. Typical data flow operations are table access and join, group, and filter. Many query optimizers, like the optimizer of IBM's DB2 or Oracle's rule based optimizer, utilize a rule-based query rewrite phase before carrying out cost-based optimization. Query rewrite translates the query graph into a semantically equivalent query graph, which is preferable to execute. Rewrite rules include, among others, translations from subqueries into joins, or rules to generate transitive predicates to involve indexes in query processing, which would otherwise not be applicable.

The System R approach to cost-based query optimization enumerates all possible physical query execution plans for a the query graph model, associates a cost with them, and selects the cheapest plan to be executed. The cost of a query plan is expressed as a linear combination of intermediate result sizes (cardinalities) weighed by factors for CPU cost, I/O cost. Cost models for distributed systems also factor in a the communication cost for transferring data between the processing components. The optimizer uses dynamic programming and prunes the search space as early as possible by only retaining the plan with the lowest cost whenever possible. The search space can be pruned whenever some (intermediate) plans are comparable. Plans are comparable, when they are semantically equivalent, i.e., when their execution produces the same intermediate result. However, in practice, there are cases where semantically equivalent plans are not yet comparable. For example, a semantically equivalent plan may not be comparable to an intermediate plan that has an interesting property, like a particular sort order for a an intermediate result, which could be exploited at a later stage during query processing to lower the overall cost.

Plans are enumerated bottom-up in a stratified way. This stratification starts with enumerating table access plans for each table then enumerates all (intermediate) plans for combining two of these table access plans, then all (intermediate) plans for three tables, and so on, until the all plans have been enumerated that produce the overall query result. As soon as several semantically equivalent (intermediate) plans have been computed, the optimizer retains only the plan with the lowest cost for further consideration. Dynamic programming has a complexity, which is exponential in the number of tables joined in a query. If the exponential memory requirement of dynamic programming is too high due to too many joins in a query, the optimizer uses a greedy algorithm as a fallback. While dynamic programming is guaranteed to determine the optimal plan with respect to the cost model, greedy algorithms usually do not return the optimal plan. Moreover, greedy algorithms tend to have a bias towards bushy plans (i.e., balanced join trees as opposed to left-deep join trees), which usually has negative impact on pipelining and transactional queries with small intermediate results. Many commercial database systems like DB2 and Oracle use a System R style bottom-up optimizer.

Volcano's alternative approach refines the query graph model by replacing logical operators like join with physical implementations like hash-join or merge-join and uses branch and bound to limit recursion. For instance, the top-down approach is used by the query optimizer of Microsoft's SQL Server.

Both bottom-up and top-down optimizers achieve the goal of determining the optimal plan with respect to a cost model. The quality of the plan does not depend on which of these search methods is used, but rather on the repertoire of rules available for generating or expanding plans. Top-down optimizers have an advantage, though, in that they always maintain a feasible execution plan, so it is possible to stop optimization at any time and execute the currently cheapest plan. Bottom-up optimizers have the risk of running out of memory without having produced a feasible plan, thus the necessary fallback to greedy or other heuristics.

Both bottom-up and top-down optimizers need a good model for the cost of producing the intermediate results that occur during query processing. This execution cost is largely dependent upon the number of rows – often called *cardinality* – that will be processed by each operator in the plan. Typically, an estimate for the cardinality of some intermediate result relies on statistics of database characteristics. Many database systems use simple statistics to approximate the size of an intermediate result, like the number of rows for each table and as well as the number of distinct values for each column. For a simple selections with an equality predicate "$C = value$" on column C of a table with n rows and c distinct values in column C, many cost models use the simple formula $1/c * n$ to estimate the cardinality of the selection predicate. This simple formula assumes a uniform distribution of all values in column C. The cardinality estimate may be vastly incorrect if some values in column C occur more frequently than others.

Estimating the number of rows (after one or more predicates have been applied) has been the subject of much research since the 1980s. The percentage of the number of rows in a table or intermediate result satisfying a predicate P is often called *selectivity* of P. The selectivity of P effectively represents the probability that any row in the database will satisfy P. Database statistics are expensive to compute and cannot always easily be maintained incrementally, when the database changes. In general, there is a trade-off between accuracy of statistics and their storage and maintenance cost. The goal of database statistics is to be good enough to enable the optimizer to produce a robust and efficient plan. The cost model of a database systems therefore uses simplifying assumptions to compute selectivities and cardinalities from the available database statistics. Examples of these assumptions include:

1. *Currency of information*: The statistics are assumed to reflect the current state of the database, i.e., that the database characteristics are relatively stable. This may not be true, if a table is changing rapidly. In this case, statistics need to be collected frequently, using the database statistics collection tool. Outdated statistics are a major source for performance problems in database queries. Many modern database management systems have an infrastructure that tries to automatically detect when statistics are outdated by monitoring update, insert, and delete operations on a table. Those systems can be configured to (re-)collect statistics automatically when a certain amount of changes have occurred.

2. *Uniformity*: As describe above, without detailed statistics on a column, the data values within a column are assumed to be uniformly distributed. If that is not the case, the database administrator can create histograms for particular columns to deal with skew in values. This will improve the accuracy of selectivity estimation for selection predicates on a single table. Only recently have researchers begun to explore ways to improve the estimation of selectivities for join predicates which combine multiple tables.

3. *Independence of predicates*: Without any knowledge about interactions of predicates, selectivities for each predicate are calculated individually and multiplied together, even though the underlying columns may be related, e.g., by a functional dependency. This independence assumption usually results in severe underestimation of the selectivity of predicates on correlated columns. For a table containing cars with two columns, *make* and *model*, the independence assumption will result in severe estimation errors for a selection predicate like "make='VW' and model = 'Jetta'." If 10% of the cars in the table are VW, and 1% of the cars are Jetta, the independence assumption would result in a selectivity:

$$^{s}{}_{\text{VW Jetta}} = {}^{s}{}_{\text{VW}} \times {}^{s}{}_{\text{Jetta}} = 10\% \times 1\% = 0.1\% = 0.001$$

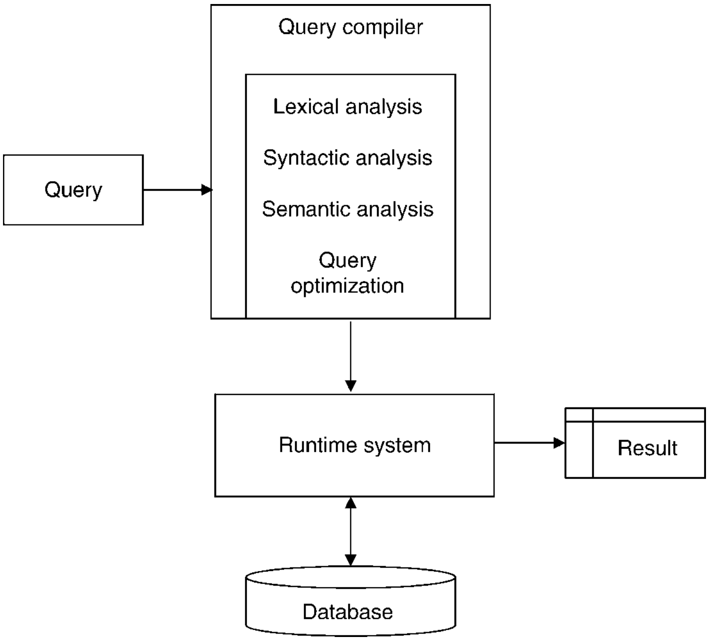
Since only VW makes Jettas, the real number is 1%, with an order of magnitude error in the estimation. This means that the intermediate result size for that part of a query will be ten times larger than what the optimizer assumes. This can result in not allocating enough memory for processing the query, or choice of a suboptimal plan. Overall, this estimation error progressively worsens, as more predicates are present in a query.

Collecting multivariate statistics across multiple columns can overcome the independence assumption when data is correlated. A simple correlation parameter is the number of distinct values over a set of attributes. This statistic is employed for instance by IBM's DB2 to correct the independence assumption for correlated local predicates, or to better estimate the selectivity of a join predicate with correlation between the two tables. While the number of distinct values over a set of columns addresses the correlation between these columns, it assumes all combinations of values in these columns to be uniformly distributed. Correlations inflate errors more than the uniformity assumption. Thus distinct values have proved worthwhile in practice as correlation statistics. In case of non-uniform correlations, multidimensional histograms have been proposed. These histograms store and maintain multivariate distribution statistics. However, multidimensional histograms do not work well for equality predicates or for maintaining correlations over a large set of columns. For that reason, they have not been widely adopted in commercial databases.

4. Principle of inclusion: The selectivity for a join predicate X.a = Y.b is typically defined to be 1/max{|a|, |b|}, where |b| denotes the number of distinct values of column b. This implicitly assumes the "principle of inclusion," i.e., that each value of the smaller domain has a match in the larger domain (which is frequently true for joins between foreign keys and primary keys). Again, correlation statistics can help overcome this assumption. Products like IBM's DB2 or Microsoft's SQL Server offer statistics on views, which can address incorrect assumptions about correlations of columns between tables or within tables.

Applications commonly used today have hundreds of columns in each table and thousands of tables, making it very hard for a database administrator to know on which columns to collect and maintain multivariate statistics or statistics on views. Tooling, either based on query feedback or proactive sampling, is addressing that problem by determining the most important correlated columns that joint statistics need to be collected on, employing statistics methods like χ^2-testing for correlation detection. Methods such as entropy maximization have been proposed to generalize the independence assumption and allow for dealing with arbitrary correlation between columns, either within or across tables.

Once the best overall query plan according to the optimizer's model has been determined, it is handed to the runtime system for execution. In some architectures,



Query Processing (in Relational Databases). Figure 1. An overview of query processing.

e.g., Informix, the query plan is directly interpreted by the runtime system. Other architectures follow the System R design and employ a code generator to produce code for a database machine, which is then executed by the runtime system. This additional code generation step allows for optimizations typically found in programming languages, to reduce path length and copying of data between CPU and memory (Fig. 1).

Key Applications

All major database systems (DB2, SQL Server, Oracle, Sybase, Informix, MySQL, PostGres) implement a query processor that largely follows the previously described architecture and concepts. DB2 and Oracle follow the System R style optimizer as described above, Microsoft SQL server uses a top-down optimizer.

Recommended Reading

1. Codd E.F. A relational model of data for large shared data banks. Commun. ACM., 13(6):377–387, 1970.
2. Freytag J.C., Maier D., and Vossen G. (eds.) Query processing for advanced database systems. Morgan Kaufmann, 1994.
3. Graefe G. Volcano – an extensible and parallel query evaluation system. IEEE Trans. Knowl. Data Eng., 6(1):120–135, 1994.
4. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73–170, 1993.
5. Lorie R.A. and Fischer N.J. An access specification language for a relational data base system. IBM J. Res. Dev., 23(3):286–298, 1979.
6. Markl V., Haas P.J., Kutsch M., Megiddo N., Srivastava U., and Tran T.M. Consistent selectivity estimation via maximum entropy. VLDB J., 16(1):55–76, 2007.
7. Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., and Price T.G. Access path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data., 1979, pp. 23–34.
8. Yu C.T. and Meng W. Principles of Database Query Processing for Advanced Applications. Morgan Kaufmann, 1997.

Query Processing and Optimization in Object Relational Databases

Johann-Christoph Freytag
Humboldt University of Berlin, Germany

Synonyms

Query evaluation; Query planning and execution

Definition

In an (object) relational database management system (DBMS) query processing comprises all steps of processing a user submitted query including its execution to compute the requested result. Usually, a user query – for example a SQL query – declaratively describes *what* should be computed. Then, it is the responsibility of the DBMS to determine *how* to compute the result by generating a (procedural) query execution plan (QEP) that is semantically equivalent to the original query. Query processing also includes the execution of this generated QEP.

While generating a QEP for a user submitted query the DBMS explores a large number of potential execution alternatives. To choose the best ones among those alternatives requires one or more (query) optimization phases.

Historical Background

In the late 1960s and early 1970s it became clear that existing DBMSs were too complex and cumbersome to use. The programmer was forced to know the physical layout of the data on disc for efficient access. The answer to those problems was the Relational Model developed by E.F. Codd in the early 1970s.

With the design and implementation of relational DBMSs in the mid-1970s it became clear very quickly that query processing and optimization are the key for implementing DMBSs with acceptable performance for the end user. In 1979, the first relational DBMS, IBM's System-R laid the foundation for today's query processing and optimization approaches: In System-R, query processing consists of three major phases:

1. Syntactic and semantic checking
2. (physical) query optimization
3. Query execution

The INGRES DBMS (Stonebreaker, UC Berkeley) took a similar approach to implement query processing.

Over the last 25 years this phase-based model has been extended by

1. adding more query processing phases such as logical query optimization or query rewrite
2. considering more alternatives during query optimization to generate better QEPs

Furthermore, the complexity of query processing has increased due to more complex query languages (such as SQL-2 or SQL-3), more complex query execution environments such as parallel processors, distributed data collections, and more complex data types, richer data structuring capabilities including object orientation.

As a response to the development of object oriented database management systems (OODBMS) the Relational Model was extended with object oriented concepts and features to embrace the major concepts of those languages. The extension of the language forced the database vendors to extend query optimization and query extension to handle those new concepts correctly efficiently such as objects, classes, path expressions, inheritance, methods, and polymorphism [10]. Most of those language features are reflected in the SQL-3 Standard.

Foundations

The Four Phases of Query Processing

Today's DBMSs usually implement query processing (including query optimization) in three different phases before executing the query to generate the requested result [2,8].

During the first phase, the query is checked for syntactic and semantic correctness. Then, during the second phase the query is rewritten into a semantically equivalent one using additional (logical/conceptual) information, such as schema information (candidate keys/Primary key, uniqueness of values, foreign keys) or integrity constraints. Rewriting might also strive for a normalized form of the query to simplify the processing during the following phases [7].

Example 1:

The SQL query

```
SELECT *
FROM CUSTOMER
WHERE EXIST (
 SELECT *
 FROM ORDER
 WHERE CUSTOMER.ID = ORDER.CUSTOMER_ID
 AND ORDER.VALUE > 10.000)
```

could be rewritten into the query

```
SELECT *
FROM CUSTOMER, ORDER
WHERE CUSTOMER.ID = ORDER.CUSTOMER_ID
AND ORDER.VALUE > 10.000
```

if the inner query block returns zero or one tuple.

The third phase – commonly known as the (physical) query optimization – translates a user query into a query execution plan (QEP). While the initial query declaratively expresses which properties the result should have, the QEP determines the execution steps to evaluate the query generating the requested result.

In general, there are many different ways (QEPs) to evaluate a user submitted query based on the physical properties of the tables accessed, such as available indexes, available materialized views, sorting order, or clustering. Thus, the query optimizers must consider and evaluate many different alternatives to execute a query using different access paths and different operators.

During the QEP generation the optimizer determines

- The best (optimal) way to access each individual relation referenced in the query using indexes (including how and in which order to evaluate local predicates)
- The best (optimal) order to join two or more tables
- The best algorithm to perform the join between two tables

Example 2:

The rewritten query of Example 1

```
SELECT *
FROM CUSTOMER, ORDER
WHERE CUSTOMER.ID = ORDER.CUSTOMER_ID
AND ORDER.VALUE > 10.000
```

might be translated into the following QEP. For the sake of clarity the QEP uses a LISP-like notation to represent the relational algebra-like expression [4].

```
(OUTPUT
  (PROJECT
  (ID CUSTOMER_NAME ADDRESS)
  (NESTED_LOOP_JOIN
    (CUSTOMER.ID = ORDER.CUSTOMER_ID)
    (SCAN CUSTOMER)
    (FILTER
      (ORDER.VALUE > 10.000)
      (SCAN ORDER)))))□
```

Finally, during the last phase the DBMS executes the generated QEP to compute the requested result, i.e., the set of tuples that match the submitted query.

The Fundamentals of a (Physical) Query Optimizer

To generate the best (optimal) QEP any query optimizer embodies the following three aspects [4]:

1. A search strategy
2. (A set of) Cost functions
3. A QEP generation strategy

The search strategy determines which alternative (partial) QEPs should be generated while looking for the best (optimal) QEP among possibilities. Strategies such as exhaustive search strategies (together with dynamic programming) or heuristics (Greedy heuristic) are approaches used in current DBMSs. However it seems that exhaustive strategies – with some restrictions – are still the preferred approach to generate the best QEP despite a large search space whose size is determined by various parameters, such as the number of tables in the query, the available indexes, or the available join methods (algorithms).

While generating different alternatives for query execution, the optimizer must determine which of these alternatives is better. Therefore, cost functions assign cost values to (partially generated) QEPs by determining the resource consumption for that plan. The resources used might be CPU time, space (amount of memory), communication time (distributed query processing), and – most importantly – the number of disk IOs since disk IOs are the dominating cost in almost all QEPs. Cost functions therefore estimate ("foresee") the amount of resources that a plan will consume when being executed.

To feed the cost functions with input, most DBMSs maintain so-called *histograms* which record value distributions for various (sets of) attributes in different tables [5]. Those value distributions allow the optimizer to make informed decisions regarding the different operators of a QEP with respect to their efficiency and effectiveness.

Once the optimizer settled for one alternative during partial QEP generation it must determine how to advance the QEP generation in its search for the best (complete) QEP. This progress might be implemented by extending the current partial QEP with new operators possibly on additional tables not yet considered, or by generating alternatives for the existing QEP by transforming it into a syntactically different QEP. The latter might include considering alternative access paths (using different indexes) or replacing an existing join algorithm with a different one.

The query optimization phase terminates once the optimizers found the best QEP based on the given (searched) alternatives and based on the given cost functions. However, since the search might take time, the optimizer might also terminate after having reached a pre-determined tie limit or after having evaluated a certain number of alternatives.

Formalisms and Approaches for Query Processing and Query Optimization

The (research) literature reports on many different approaches to query processing and query optimization. However, there does not seem to be one formalism to describe those different approaches. Some of them use a relational algebra like notation (with extensions) to show operational approaches. Others use logic based notation such domain/tuple relational calculus or the tableaux notation. Some presentations use proprietary notations based on innovative data structures to present sophisticated algorithms.

Of course, the approach to query processing and query optimization is heavily influenced by the data model and the expressiveness of the query language. The continuous extension of SQL to SQL-2 or SQL-3 has lead to an extended portfolio of query processing and query optimization techniques. The same is true object oriented and object relational DBMSs where introduced. Similarly, the extension of SQL to express Data Warehouse queries leads to new techniques (algorithms and "tricks") for query processing and query optimization.

Implementing Query Optimizers

The first optimizers were implemented for the database prototypes IBM System-R [8] and Berkley's INGRES. The former system laid the foundation conceptually and architecture wise for many optimizers to come including those that are used in today's DBMS products.

In many cases, DBMS products only provide access to the architecture, the optimization strategies, or the cost functions used. Most of the dominant products allow the user to view the QEP as generated by the optimizer together with cost values and cardinality estimates.

However, for several DBMS products, there exist well known prototypes that provide hints how some of the optimizers in existing products might work. The newly researched ideas and novelties in those prototypes are often described in research papers published at well-know conferences like the ACM Sigmod conference, the VLB conference, the IEEE ICDE conference, or the European EDBT conference.

Key Applications

The query optimizer is an integral part of any DBMS (product). The degree and extent of optimization is determined by the vendors or implementers. The repertoire of the optimization step is continuously extended, depending on the requirements coming from OLTP queries, data warehousing/OLAP queries (star schema, aggregate queries), data mining queries, or queries coming from systems such as geographical information systems booking systems, and others.

Future Directions

Over the last several decades, query processing and optimization has adapted to the changes in computing hardware. In the 1990s, parallel hardware became readily available at a reasonable cost, DBMSs and query processing had to be extended and adapted to parallel QEP execution.

In addition, many database experts doubt that the general two-phase query processing approach (optimize the query, then execute) is the right one for future DBMs that should run on the next generation of hardware. Especially when the underlying system continuously changes query optimization and query execution must be closely intertwined to react to those changes in (almost) real time. Furthermore, when executing queries on large tables, it might be desirable to change execution strategies "on the fly." Currently, once the query processing phase is done (determining the best plan) the QEP is fixed. Recent research in the area of query processing has developed approaches to change QEPs either "on the fly" when necessary.

Now, multi-core CPUs seem to be the new development direction for hardware. At the same time, new storage technology comes to life (flash disks) with different operating characteristics that will change the architecture and therefore the processing models and processing capabilities of existing DBMSs. In addition, large CPU farms, large storage farms, and increasing main memory sizes already have a dramatic impact on existing approaches and techniques in query processing and query optimization. Understanding the Web as one large database could completely change today's DBMS architecture standards and assumptions on how to build future DBMSs.

Experimental Results

Unfortunately, there are no benchmarks that consider the database query optimizer as a separate component for benchmarking. However, there are efforts to ensure the stability of the optimizers despite estimation errors during query optimization and despite changes in the expected resources available during query execution.

Cross-references

► Cost Function
► Database Products that include optimizer technology
► Histogram
► IBM DB2
► Indexing
► Informix (owned by IBM)
► Ingres
► Logical Schema Design
► Materialized Views
► Microsoft SQLServer
► Oracle
► Parallel Database
► Physical schema
► Query Language
► Relational Algebra
► Relational Integrity Constraints
► Relational Model
► SAP MaxDB
► Search Strategy
► Sybase
► System-R

Recommended Reading

1. Deshpande A., Ives Z., and Raman V. Adaptive query processing. Foundations and Trends in Databases, 1(8):1–140, 2007.
2. Freytag J.C. The basic principles of query optimization in relational database management systems. In Proc. IFIP 11th World Computer Congress, 1989, pp. 801–807.
3. Freytag J.C., Maier D, and Vossen G (eds.) Query Processing for Advanced Database Systems. Morgan Kaufmann, 1994.
4. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73–170, 1993.
5. Ioannidis Y.E. The history of histograms (abridged). In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 19–30.
6. Jarke M. and Koch J. Query optimization in database systems. ACM Comput. Surv., 16(2):111–152, 1984.
7. Pirahesh H., Hellerstein J.M., and Hasan W. Extensible/rule based query rewrite optimization in starburst. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1992, pp. 39–48.
8. Selinger P.G., Astrahan M.M., and Chamberlin D.D., Lorie R.A., and Price T.G. Access path selection in a relational database

management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.

9. Yu C.T. and Meng W. Principles of Database Query Processing for Advanced Applications. Morgan Kaufmann, 1998.

Query Processing in Data Warehouses

WOLFGANG LEHNER
Dresden University of Technology, Dresden, Germany

Synonyms

Data warehouse query processing; Query answering in analytical domains; Query optimization for multidimensional systems; Query execution in star/snowflake schemas

Definition

Data warehouses usually store a tremendous amount of data, which is advantageous and yet challenging at the same time, since the particular querying/updating/modeling characteristics make query processing rather difficult due to the high number of degrees of freedom.

Typical data warehouse queries are usually generated by *on-line analytical processing* (OLAP) or *data mining* software components. They show an extremely complex structure and usually address a large number of rows of the underlying database. For example, consider the following query: 'Compute the monthly variation in the behavior of seasonal sales for all European countries but restrict the calculations to stores with > 1 million turnover in the same period of the last year and incorporate only 10% of all products with more than 8% market share. 'In a first step, this query has to identify certain stores based on the previous year's sales statistics and it needs to define the top-selling products on a monthly basis. In a second step, the system is able to compute the different behavior based on individual countries.

The structural complexity as well as the huge data volume addressed by data warehouse queries makes it a true challenge to optimize and execute these queries. Therefore, query processing in data warehouse systems encompasses the definition, the logical and physical optimization, as well as the efficient execution of analytical queries. The specific techniques range from classic rewrite rules to rules considering the special structure of a query. For example, data warehouse queries usually target *star schema*s. Such star queries with references to a fact table and foreign-key joins to multiple dimension tables can be beneficially exploited during query optimization and query processing. Moreover, the embedding of specialized operators (like *CUBE* or ROLLUP) or approximate query processing methods using samples or general-purpose synopses has to be addressed in the context of query processing in data warehouse environments. Finally, query processing in data warehouse systems has to address the support of non-*SQL* query languages (e.g., specialized analytical query languages like MDX [13]), alternative storage structures (e.g., column-wise organization), or alternative processing models based purely on main-memory structures.

Historical Background

Query processing has a long history with research and commercial products. The generation of efficient plans for the smooth execution of database queries thus represents one of the most challenging research directions in database history. With the advent of descriptive query languages, this has turned into a crucial problem, since systems must now figure out the optimal way to execute queries.

Traditional *transactional*-style (on-line transaction processing = OLTP) interaction patterns between an application and the database system consist of short read and write queries with correlated sub-queries, which typically compete with a large number of concurrently running transactions. DWH-style query patterns follow a significantly different style. Since the update/load of data warehouse databases is under the control of data warehouse monitors, user queries are typically read queries that touch a large number of rows with star joins and group-by operations as their functional core. Around 1995, the concept of data warehousing emerged and showed that OLTP-style query processing and optimization techniques were insufficient in dealing with this specific class of requirements. This deficit triggered a tremendous amount of research activities to push the envelope in multiple directions. After more than 10 years of research and development, a fairly large number of extensions/modifications have turned it into a commercially available system. However, due to the increasing need for analytical tasks performed within (and not on top of) database systems, the improvement of query processing techniques for analytical applications still represents an active research area with significant potential for innovation.

Foundations

Query processing in data warehouse systems does not address individual methods or techniques. In contrast, query processing in the analytical domain is characterized by a broad range of individual techniques ([3,7]; see 3.13). The challenge in this context is to orchestrate the different and individually deployed methods. The following introduction to important query processing aspects, including a brief discussion of the requirements and specifics of query processing in data warehouse systems and possible solutions, provides a comprehensive overview of the topic.

Requirements and Specifics of the Analytical Context

While the specific character of data warehouse systems poses challenges for query processing from multiple perspectives, it also has constraints, which can be exploited and therefore reduce the number of space limitations. Although the processing and optimization of *SQL* queries attracts the most attention, query processing in data warehouse environments addresses other query languages like MDX [13], DMX [5], or SQL/XML [12] with analytical functionality specified within the *XPath/XQuery* fragments. The complexity of handling domain-specific query languages and of mapping these expressions either to highly specialized storage structures or to SQL is further increased by the need for support of domain-specific operators. In this context, the integration of data mining algorithms for cluster searches or for the computation of association rules into database systems can be seen as a prominent example.

To counterbalance these requirements, query processing in data warehouse systems shows a number of limitations or specializations that facilitate adequate solutions. For example, a high ratio of analytical queries is issued to compute standardized reports (e.g., legal reporting in the financial sector, cockpit solutions, etc.). These canned queries show huge potential to pre-compile the execution plans, to apply multiple-query optimization or to pre-compute partial results using materialized views. Furthermore, the explicitly controlled update transactions in data warehouse systems typically show append-only characteristics, which enable the system to apply specifically tailored optimization strategies. For example, adding incoming rows to a single partition and periodically attaching it to the analytical database in an atomic way affects the concurrently running analytical queries only to a minimal extent. Finally, although analytical queries usually apply aggregation techniques over a large number of rows, the analytical queries may exhibit quite selective predicates. For example, consider the following query: ‘Return the weighted sales distribution of all Californian stores selling Mac and Linux machines with more than 4GB RAM in 2007 and divide it by month.’ It shows a highly selective predicate but may still incorporate a huge number of rows within the computation of the measure. Query processing may tackle this phenomenon by introducing special index structures to support selective predicates of multiple columns from different tables or by applying specialized processing techniques such as online aggregation or approximate query answering in general.

Potential and Solutions for Efficient Data Warehouse Query Processing

The specific characteristics as well as the specific circumstances create a wide range of options to perform query processing in data warehouse scenarios in an efficient way. The following list provides the most prominent classes of methods with an outline of their specific role in this context:

Part I: Query Planning and Execution

Optimizing the Optimization Process

- Since analytical queries usually show a complex structure, the optimization of a query may require a considerable fraction of the overall query execution cost. Therefore, modern techniques propose a variety of solutions ranging from the a-priori limitation of the search space (usually by applying certain heuristics – e.g., the optimization pattern of star queries) to the pruning of alternative plans as early as possible [15].

Optimization Goal

- In contrast to OLTP-style query optimization, the general goal of OLAP-style query optimization is to generate only a good (i.e., not an optimal) but robust plan. Due to the complex structure of a query, potential errors in the cost estimation usually have their origin at the leaf nodes of an operator tree and – after having been propagated along an operator path – they may result in extremely vague estimates. A robust plan is required to expose well-performing behavior even if the data show different characteristics than those used in the planning phase. The notion of robust plans and the design of more adequate statistics (e.g., sample or wavelet synopses) are currently the subject of intense discussions in the research community.

Specific Rewrite Rules

- As already mentioned, data warehouse environments imply a specific characteristic of queries – usually following the notion of star queries, which consist of a join of the fact table and multiple dimension tables followed by a complex selection predicate and a grouping condition with aggregation. Query processing has to detect such situations and apply specific optimization patterns. For example, group-by operations can be pushed down in certain cases to reduce the number of rows for join operations [18]. Another example addresses star joins: the star join technique may decide to compute the Cartesian product of the selected parts of all dimension tables in a first step and then join the (large) fact table in a last step.

Multi-Query Optimization

- A final class of techniques – which have not been considered in OLTP-style query processing due to restrictions of *ACID properties* – address the issue of optimizing a set of queries simultaneously, thus resulting in shared use of different resources. Key applications for these techniques can be found on the application level (in the computation of multiple similarly structured statistical reports) or within a system (in the propagation of changes of a base table to multiple dependent materialized views to share update efforts).

Part II: Considering Logical Access Paths

Partition Management

- Partitions reflect a concept that is useful for the administration of a database system as well as for the optimized execution of partition-aware queries. Within a data warehouse environment, incoming data items are typically stored within staging tables. After transformation and cleaning steps, tables are converted into partitions and attached to the global data warehouse database. Whenever data items in a partition are obsolete, a partition can be detached and moved (as a table) to the archive. From a query optimization perspective, partition pruning represents a powerful mechanism to potentially reduce the number of rows accessed within a query. Whenever a selection predicate refers to a partitioning criterion, the system may restrict the query execution to only those partitions that are actually referenced by the query. The main partitioning criterion usually addresses the time dimension, e.g., by month, but may also be used to partition other dimensions, e.g., by product category or geographical entities.

Materialized Views

- A second component of the logical access path consideration is the transparent use and the implicit maintenance of materialized views. Similar to classic index structures, a query is matched against a view description and internally rewritten to exploit an existing materialized view by producing the same result as the original query. Especially in the context of aggregation queries, the concept of materialized views represents an extremely powerful mechanism to speed up analytical queries. For example, a materialized view may hold summary data grouped on the family level within a product dimension, with an additional grouping based on city and month. Every query with a compatible aggregation function and a grouping condition that is “coarser” than the grouping combination of the materialized view, e.g., sum per quarter, product category, and state, may benefit from the pre-aggregated data stored within a materialized view. In addition (and similar to physical index structures), materialized views are transparently maintained in the case of changing base tables to provide a consistent view of the data.

Part III: Considering Physical Access Paths

Data Organization

- While classic relational database engines favor the concept of row-based storage, a variety of specialized systems exist that follow the concept of column-based storage layout. Query processing in column-based systems benefits from reading only those columns that are actually required to answer an incoming query. This feature is extremely beneficial in the presence of wide dimension tables and queries referring to just a few attributes.

Compression and Main-Memory Techniques (see *main memory DBMS*)

- Although (persistent) storage costs have been decreasing significantly, a compressed version of data may speed up the query processing because of the reduced number of I/O operations. While block- and table-based compression schemes have been well understood, the static characteristics of data warehouse data have provided the motivation to push compression techniques into commercial systems. For column-based systems with the primary goal to be main-memory-centric, compression is a must; a large variety of techniques (e.g., Huffman coding schemes) is applied in different systems.

Additional Index Structures

- The history of database research has shown that different application areas can be supported with specialized index structures. Particularly aimed at the support of queries within the analytical domain, *bitmap index* structures have experienced a rejuvenation and are now part of the prominent commercially available systems [2,16]. Another approach to take advantage of the static characteristic of a data warehouse database can be seen in the concept of *star index*es (also called join indexes or foreign-table indexes), which follow the basic idea to index the result of a join operation. Join indexes are particularly well-suited to exploit the relationship of dimension tables with the corresponding fact table.

Multidimensional Clustering Schemes

- *Clustering* in general tries to preserve the topological relationship of entries in a database. Since data warehouse datasets are typically multidimensional in nature (e.g., sales by shop, date, and product), multidimensional clustering schemes represent a valuable solution to store logically related facts within the same block with the overall goal to reduce the number of I/O operations.

Part IV: Alternative Query Answering Models

Online Aggregation

- Analytical queries typically show a response time in the range of multiple seconds, minutes, and sometimes hours. Online aggregation [9] is a promising technique to return intermediate results of the queries while the query is still running; the intermediate results are refined step by step until the final result is computed or until the application is satisfied with the precision of the answer.

Approximate Query Answering/Approximate Query Processing

- Many application areas that perform statistical analyses prefer to yield an approximate but quick answer instead of delaying an exact answer. The concept of approximate query answering addresses this idea and proposes techniques for the online or offline design of database synopses, which are then used to answer incoming queries. The design of synopses ranges from simple uniform samples over specifically tailored (stratified) samples to wavelet synopses. The main challenge of approximate query answering consists of the creation and exploitation of synopses on the fly (i.e., within the context of the execution of a single query) and of the provision of error bounds to derive a quality measure to be returned to the user.

This list of methods and techniques outlines the most influential factors for query processing in data warehouse environments. Behind every perspective touched in this description, there are huge numbers of specific research issues – solved and unsolved ones.

Key Applications

Over the last 10 years, database systems have become the foundation of every larger analytical infrastructure (usually embedded within a data warehouse system). Therefore, all analytically-flavored applications heavily exploit specialized database support. This obviously large spectrum of applications ranges from mass reporting for the computation of thousands of parameterized reports to the support of interactive data cube exploration (OLAP). As data mining methods are becoming more and more standardized techniques, the computation of association rules, classification trees, and (hierarchical) clusters represent key applications.

Future Directions

In the future, query processing in analytical domains will be driven by two factors. On the one hand, data volumes will continue to grow significantly, mainly because of advances in data integration efforts (to ease the pain of manual integration of additional data sources) and the growing presence of sensors to track individual items. More data will go hand in hand with the presence of increased main-memory capacity and the integration of solid-state disks (SSD) into the memory hierarchy. Physical database design and the corresponding exploitation of specialized structures will be major challenges.

On the other hand, the topic of query processing will be faced with more sophisticated statistical methods, which have to be natively supported by the database engine to yield reasonable query performance. While for many analytical application infrastructures, the "bring the data to the statistical package" method still holds, this approach will be inverted at some point. Statistical packages will work more closely with database systems, implying that computational components are brought closer to the data (as an integral part of a database engine) to be integrated within the overall optimization process. Also, the scope of analytical applications will widen and incorporate comprehensive *data visualization* techniques.

Both future developments on the application level will have significant consequences for the design of query processing techniques ranging from low-level data organization up to the support of a specialized query interface.

Cross-references

▶ Approximate Query Processing
▶ Bitmap Index
▶ Cube
▶ Data Sampling
▶ Main-Memory DBMS
▶ Multidimensional Modeling
▶ Parallel Query Processing
▶ Query Optimization
▶ Query Processing
▶ Query Rewriting Using Views
▶ Snowflake Schema
▶ Star Schema
▶ SQL

Recommended Reading

1. Celko J. Joe Celko's Data Warehouse and Analytic Queries in SQL. Morgan Kaufmann, 2006.
2. Chan C.-Y. Bitmap index design and evaluation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 355–366.
3. Chaudhuri S.and Dayal U. An Overview of Data Warehousing and OLAP Technology. ACM SIGMOD Rec., 26(1): 65–74, 1997.
4. Clement T.Y. and Meng W. Principles of Database Query Processing for Advanced Applications. Morgan Kaufmann, 1997.
5. Data Mining Extensions (DMX) Reference. Available at: http://msdn2.microsoft.com/en-us/library/ms132058.aspx
6. Graefe G. Query Evaluation Techniques for Large Databases. In ACM Comput. Surv., 25(2), 1993, S. 73–170.
7. Gray J. et al. The Lowell Database Research Self Assessment, June 2003. Available at: http://research.microsoft.com/~gray/lowell/
8. Gupta A. and Mumick I. Materialized Views: Techniques, Implementations and Applications. MIT Press, Cambridge, MA, 1999.
9. Hellerstein J.M., Haas P.J., and Wang H.J. Online Aggregation. In Proc. ACM SIGMOD Int. Conf. on Management of Data,1997, pp. 171–182.
10. Inmon W.H. Building the Data Warehouse. 2nd edn, Wiley, NY, USA.
11. Niemiec R. Oracle Database 10g Performance Tuning Tips & Techniques, 2007.
12. N.N. ISO/IEC 9075–14:2003: Information technology – Database languages – SQL – Part 14: XML-Related Specifications (SQL/XML). Available at: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber = 35341.
13. N.N. Multidimensional Expressions (MDX) Reference. Available at: http://msdn2.microsoft.com/en-us/library/ms145506.aspx.
14. Roussopoulos N. The logical access path schema of a database. In IEEE Trans. Softw. Eng., 8, (6)S.563–573,1982.
15. Tao Y., Zhu Q., Zuzarte C., and Lau W. Optimizing large star-schema queries with snowflakes via heuristic-based query rewriting. In Proc. Conf. of the IBM Centre for Advanced Studies on Collaborative Research, 2003, pp. 279–293.
16. Valduriez P. Join indices. ACM Trans. Database Syst., 12(2):218–246, 1987.
17. Weininger A. Efficient execution of joins in a star schema. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 542–545.
18. Weipeng P.Y. and Larson, P. Eager Aggregation and Lazy Aggregation. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 345–357.

Query Processing in Deductive Databases

LETIZIA TANCA
Politecnico di Milano University, Milan, Italy

Synonyms

Datalog query processing and optimization; Recursive query evaluation; Logical query processing and optimization

Definition

Most of the research work on deductive databases has concerned the *Datalog language*, a query language based on the logic programming paradigm which was designed and intensively studied for about a decade. Its origins date back to the beginning of logic programming, but it became prominent as a separate area around 1978, when Hervé Gallaire and Jack Minker organized a workshop on logic and databases. In this entry, the definition of the typical computation styles of Datalog will be given, the most important optimization types will be summarized, and some developments will be outlined.

Historical Background

The research on deductive databases was concentrated mostly between the mid-1980's and the mid-1990's. In those years, substantial efforts were made to merge Artificial Intelligence technologies with those of the Database area, with the aim of building *large and persistent Knowledge Bases*. An important contribution towards this goal came from database theory, which concentrated on the formalization of Datalog – specifically designed for the logic-based interaction with large knowledge bases – and on the definition of computation and optimization methods for Datalog rules [2,5,7,14]. In parallel, various experimental projects showed the feasibility of Datalog as a data-oriented logic programming environment [4,13].

The reaction of the database community to Datalog has often been marked by skepticism. In particular, the immediate practical use of research on sophisticated rule-based interaction has often been questioned. However, the research experience on Datalog, properly filtered, has taught important lessons to Database researchers, setting the basis for the theoretical systematization of several related issues [1].

Foundations

Datalog is in many respects a simplification of the more general *Logic Programming* paradigm [11], where a program is a finite set of *facts* and *rules*. Facts are assertions about the reality of interest, like *John is a child of Harry*, while rules are sentences which allow deducing of facts from other facts. For instance, a rule might say *If X is a child of Y, and Y is a child of Z, then X is a grandchild of Z*. Facts and rules may contain variables; facts that only contain constants are called *ground facts*.

In the formalism of Datalog, both facts and rules are represented as *Horn clauses*:

$$L_0 : -L_1, \ldots, L_n$$

where each L_i is a *literal* of the form $p(t_1 \ldots t_n)$, such that p is a *predicate symbol* and the t_i are *terms*. A term is either a constant or a variable, while functional symbols are not allowed, at least in the basic syntax of Datalog. The left hand side (LHS) of a Datalog clause is also called its *head*, while the right hand side (RHS) is its *body*. Clauses with an empty body are the facts: indeed, the body contains the clause premises, thus if there are no premises this means that the head is an assertion. Clauses with a non-empty body are the rules. A set of ground facts can easily be thought of as a relational database, since each fact *parent (John, Harry)* (between John and Harry there is a child-parent relationship) can be written as a tuple ⟨*John, Harry*⟩ stored into a relation PARENT. Both facts and rules are a form of *knowledge*; indeed, the knowledge stored in the database (alternatively represented as the set of ground facts) is enriched by the knowledge which can be deduced from the rules.

In the context of general Logic Programming, it is usually assumed that all application-relevant knowledge (facts and rules) are contained within a single logic program. On the other hand, Datalog has been developed for applications which use large, relational databases; therefore two sets of clauses will be considered: the set of ground facts, called *Extensional DataBase (EDB)*, and the set of rules, i.e., the Datalog program, called the *Intensional DataBase (IDB)*. The *Herbrand Base* is the set of all ground facts that can be expressed in the language of Datalog, by using all the constants present in the database and all the predicates of *EDB* ∪ *IDB*.

A *Datalog program* is a set of ground facts (possibly stored into a relational database) and rules, satisfying the safety condition that *all the variables contained in the rule head must also be present in its body*.

Note that a Datalog program can be considered as the specification of a *query against the EDB*, producing as answer the (set of) relation(s) of the IDB. It often happens that a user is interested in a subset of the (large) relation(s) that can be defined from a Datalog program: a *goal* is a single literal, preceded by a question mark and a dash, used to express constraints on the relations specified by the Datalog program. For example, ? − *parent*(*John*,*X*). specifies all the *X* such that *John* is a child of *X*.

Consider, as an example, the EDB constituted by a unary relation PERSON and a binary relation PARENT (containing all the pairs ⟨*child*,*parent*⟩) and the following program:

$$r_1 : \ sgc(X, X) : \ -person(X).$$

$$r_2 : \ sgc(X, Y) : \ -parent(X, X1), sgc(X1, Y1), \\ parent(Y, Y1).$$

Rule r_1 simply states that a person is a cousin at the same generation of him/herself, while rule r_2 says that two persons are same generation cousins if they have parents who, in turn, are same generation cousins. Note that r_2 is *recursive*, and that rule r_1 constitutes the recursion base. A typical goal against the set $\{r_1, \ r_2\}$ is ? − *sgc*(*John*, *Y*)., asking for all the same generation cousins of John.

Evaluation of Datalog Programs

Consider a Datalog rule $R = L_0 : -L_1,...,L_n$, and a set of ground facts $F = \{F_1,...F_n\}$. If a substitution θ exists, such that $\forall 1 \leq i \leq n, L_i\theta = F_i$, then, from the rule *R* and from the facts *F*, the fact $L_0\theta$ can be inferred in one step. This fact might be new, or already known.

The just described inference rule – which is actually a meta-rule – is called *EPP (Elementary Production Principle)*. Consider now a whole Datalog program *S*. New knowledge can be obtained from the Datalog program by applying it to the set of all ground facts of the database, to obtain new ground facts. Informally, it can be said that a ground fact *F* is *inferred* from *S* ($S \vdash F$) iff, either *a)* $F \in S$ or *b)* *F* can be obtained by applying the EPP *a finite number of times*. More precisely:

- $S \vdash F$ if $F \in S$.
- $S \vdash F$ if a rule *R* and ground facts $F_1,...,F_n$ exist such that $\forall 1 \leq i \leq n$, $S \vdash F_i$, and F_i can be inferred in one step by the application of *EPP* to *R* and $F_1,...,F_n$.

The sequence of applications of *EPP* which is used to infer a ground fact *F* from *S* is called a *proof* of *F* from *S*. The *proof-theoretical framework* thus established allows to infer new ground facts from an original set of Datalog clauses; on the other hand, there is a model-theoretic approach, which provides a definition of *logical consequence* (⊨). It is possible to prove that:

Theorem 1. (Soundness and completeness of Datalog) Let *S* be a set of Datalog clauses, and let *F* be a ground fact. Then $S \vdash F$ if and only if $S \vDash F$.

A proof of this theorem can be found, for example, in [5].

In order to check whether EPP applies to a rule $R : L_0 : -L_1,...,L_n$ and to an ordered list of ground facts $F_1,...,F_n$, an appropriate substitution θ for the variables of *R* must be found, such that $\forall 1 \leq i \leq n$, $L_i\theta = F_i$.

Given a finite set *S* of Datalog clauses, i.e., a Datalog program, according to the soundness and completeness theorem, the set of all facts which are *derivable* from *S* is the set *cons(S)* of the *logical consequences* of *S*, which can be computed by the following algorithm:

```
FUNCTION INFER(S)
INPUT: a finite set S of Datalog clauses
OUTPUT: cons(S)
begin
W:= S
while EPP produces a new ground fact F ∉ W
do W := W ∪ F
return (facts(W))
end
```

The *INFER* algorithm always terminates and produces as output a *finite* set of ground facts, *cons(S)*, since the number of constants and predicates symbols, as well as the number of arguments of these predicates, is finite. The order in which *INFER* generates new facts corresponds to the *bottom-up* order of a proof tree, thus the principle underlying *INFER* is called *bottom-up evaluation*, or, as in Artificial Intelligence, *forward chaining* (forward in the sense of the logical implication contained in the Datalog rules).

The set *cons(S)* can also be characterized as *the least fixpoint* of the transformation T_S, a mapping from 2^{HB} to itself defined as follows:

$$\forall W \in HB, T_S(W) = W \cup FACTS(S) \\ \cup INFER1(RULES(S) \cup W)$$

where *INFER1(S)* denotes the set of *all* ground facts that can be inferred in one step from S via *EPP*.

Accordingly, *cons(S)* can be computed by fixpoint iteration, i.e., by computing, in order, $T_S(\emptyset)$,$T_S(T_S(\emptyset)$, $T_S(T_S(T_S(\emptyset)))$,...,until a term which is equal to its predecessor is reached. This final term is *cons(S)*.

The *top-down* evaluation of a Datalog program is based on a radically different approach, where proof trees are built from the top to the bottom, by applying *EPP* "backwards," which is much more appropriate when a goal is specified together with *S*. The general principle of *backward chaining* corresponds, in Prolog, to the *SLD resolution (SL resolution with Definite clauses)* inference rule, introduced by Robert Kowalski [9]. Its name is derived from SL resolution, which is both sound and refutation complete for the unrestricted clausal form of logic.

The logic programming formalism is now related to a database query language, in order to show how easy the integration between the two realms is. Each clause of a Datalog program can be translated into an inclusion relationship of Relational Algebra; then, the set of relationships which refer to the same predicate are interpreted as Relational Algebra *equations*, whose constants are the EDB relations and whose variables are the IDB predicates, defining virtual relations. Determining a solution of the thus composed system corresponds to determining the values of the variable relations, i.e., to finding the ground facts in the IDB predicates. Consider a Datalog clause:

$$C : p(\alpha_1 \ldots \alpha_n) : -q_1\left(\beta_{k_1} \ldots \beta_{k_h}\right), \ldots q_m\left(\beta_{k_j} \ldots \beta_{k_m}\right)$$

the translation associates to *C* an inclusion relationship $Expr(Q_1,\ldots,Q_m) \subseteq P$ among the relations $Q_1,\ldots,Q_m,P$ that correspond to the predicates $q_1,\ldots,q_m,p$, adopting the convention that relation attributes are named according to the number of the corresponding argument of the related predicate. For example, the Datalog rules of the *same generation cousins* program are translated into the inclusion relationships:

$$\pi_{1,1} PERSON \subseteq SGC$$

$$\pi_{1,5}\left((PARENT \bowtie_{2=1} SGC) \bowtie_{4=2} PARENT\right) \subseteq SGC$$

The rationale behind this translation is that literals with common variables correspond to equi-joins over those variables, while the variables exported to the rule head correspond to projections. The new (virtual) relation SGC (actually, a *view*) is defined as the extension of the predicate *sgc*. For each IDB predicate, all the related inclusion relationships are collected, generating an algebraic equation that obtains the predicate by performing the union:

$$\begin{aligned} SGC = {} & \pi_{1,1} PERSON \\ & \cup \pi_{1,5}((PARENT \bowtie_{2=1} SGC) \bowtie_{4=2} PARENT) \end{aligned}$$

Logical goals are also translated into algebraic queries, over the EDB or over the just defined views: ? $-$ *sgc* (*John*,*Y*). corresponds to $\sigma_{1=\text{``}John\text{''}} SGC$.

Note that this translation is based on the use of all classical algebraic operators, except *the difference*. In fact, it can be shown that Datalog without recursion is equivalent to Relational Algebra deprived of the difference operator.

Optimization of Datalog Programs

The evaluation of Datalog programs according to various forms of fixpoint computation, similar to the INFER algorithm, is called *naive*, as opposed to better, more performant techniques whose mutual relationship is not always obvious. Optimization methods can be observed with regard to different orthogonal dimensions: the *formalism* (logical vs. algebraic), the *search strategy* (bottom-up vs. top-down), the *technique* (rewriting programs into more efficient ones vs. directly applying an efficient evaluation method), and the *type of information exploited by the optimization process* (semantic vs. syntactic).

Since Datalog programs can equivalently be written as sets of algebraic equations, a Datalog program can actually be evaluated in the same way as any algebraic query, provided that a way to process recursion is available. Actually, algebraic evaluation methods that mimick the naive method have been introduced, and the classical results of algebraic query optimization, like common subexpression analysis and equivalence transformations, have been profitably transformed to be applied to recursive queries [5].

As far as the search strategy is concerned, observe that bottom-up methods actually consider rules as *productions*, generating all possible consequences of *EDB* ∪ *IDB* until no new facts can be deduced; thus, these methods are applied in a set-oriented fashion, which is a desirable feature in the database context, where large amounts of data are stored in mass memory and must be retrieved in the buffer in a "set-at-a-time" way. On the other hand, also observe that bottom-up methods do not take in immediate advantage the selectivity due to the existence of bound arguments in the goal predicate.

By contrast, top-down methods [5,9] use rules as *subproblem generators*, since each goal is considered as a problem to be solved. The initial goal ? $-$ $p(\alpha_1 \ldots \alpha_n)$. is matched against some rule $C : p(\alpha_1 \ldots \alpha_n) : -q_1\left(\beta_{k_1} \ldots \beta_{k_h}\right), \ldots q_m\left(\beta_{k_j} \ldots \beta_{k_m}\right)$, and generates *subgoals* ? $-$ $q_i(\beta_1 \ldots \beta_i)$ that represent new subproblems to be solved. In this case, if the goal contains some bound (i.e., constant) argument, then only facts that are related to the goal constants are involved in the computation. Suppose, for instance, that the goal ? – $sgc(John, Y)$. be given. Then, when applying top-down rule r_2, the subgoal $parent(X, X1)$ is only further analyzed with regard to the parents of John, that is, *only the subrelation* $\sigma_{1\,=\,\text{“John”}}PARENT$ is involved in the computation of the first literal. However, although the algorithms based on this evaluation method already produce some optimization, they may be inappropriate for the database context, because many of them work "one-tuple-at-a-time."

Another analysis dimension for optimization methods is whether they directly interpret the program or first rewrite it into an equivalent, more efficient form and then evaluate it in a naive way. To this category belong, for instance, the *Magic Sets* and *Counting* methods [2], where the authors "simulate" the binding propagation achieved by top-down evaluation by applying a simple program transformation to a certain class of programs, in a similar way as algebraic database optimization techniques (e.g., push of the selections) are applied. A similar approach, directly introduced by means of the algebraic formalism, is taken in the *Reduction of Variables* and *reduction of Constants* methods [5].

Finally, also *semantic information* can be used to optimize programs: for instance, [6] base the optimization on the additional semantic knowledge provided by *database constraints*. For example, a constraint might state that *all sailing vessels in Ischia are sheltered in the main harbour*, thus the query asking for *the harbour in Ischia where the sailing boat "Roxanne" is located* can be answered without even accessing the DB.

Negation in Datalog

In pure Datalog, the negation sign ¬ is not allowed; however, negative facts can be inferred from Datalog programs by adopting the *Closed World Assumption (CWA)*, which, in this context, reads as follows:

If a fact is not derivable from a set of Datalog clauses, then the negation of that fact is true.

For example if, after computing the SGC relation, the tuple $\langle LUCY, JOHN\rangle$ is not found, the fact $\neg sgc\ (lucy, john)$ – that is, Lucy is not a child of John – is inferred.

The CWA applied to Datalog clauses allows the deduction of negative facts, but not their use within the Datalog rules in order to deduce some new facts. For instance, in Relational Algebra, all the pairs of persons who are not same generation cousins are specified as: $NONSGC = (PERSON \times PERSON) - SGC$. Accordingly, one would like to write:

$$r_3:\ nonsgc(X, Y):-person(X), person(Y),\ \neg sgc(X, Y).$$

The language *Datalog*$^{\neg}$ has the same syntax as Datalog, but here negated literals are allowed in the rule bodies. For safety reasons, all the variables occurring within a negated literal must also occur in a positive literal of the body. Without delving into semantic details, note that unfortunately, because of recursion, the computation of Datalog$^{\neg}$ programs is not as straightforward as that of a pure Datalog program; indeed, by simply applying the *EPP* and the INFER algorithm to a recursive Datalog$^{\neg}$ program, one may incur into a contradiction, since some negative facts that are inferred at step n of the computation might be derived as positive at some step $n + k$. Intuitively, consider the program composed by rules r_1, r_2, r_3, and think of the set of same generation cousins derived at the first computation step. Suppose that $sgc(lucy, john)$ has not been derived; then, at the second step, $\neg sgc(lucy, john)$ may be derived by applying r_3; yet, it might be that $sgc(lucy, john)$ is derived from r_1, r_2 at some later step $k(k > 2)$.

The most common policy used to avoid such problems is only allowing Datalog$^{\neg}$ programs which are *stratified* [7], according to the following intuition: when evaluationg a rule with one or more negative literals in the body, *evaluate first the predicates corresponding to these negative literals*. Then, the CWA is "locally" applied to these predicates. Consider the example above: since the *sgc* predicate appears in negative form in rule r_3, it is first evaluated by applying only rules r_1,r_2 (the "first stratum" of the program); then, once the whole extension of *sgc* has been computed, all the pairs of persons who are not in the SGC relation can be derived (actually by difference).

However, one can intuitively guess that not necessarily all the Datalog programs can be stratified in this way, that is, it may be possible that there is another rule r_4 which in turn contains the predicate *nonsgc* in negative form, and so on and so forth. More formally, define the *Dependency Graph* $DG(P) = \langle N(P), E(P)\rangle$ of a Datalog$^{\neg}$ program P as follows: $N(P)$ is the set of predicates p occurring in the rule heads of P, while an edge $\langle p,q \rangle$ belongs to $E(P)$ if the predicate symbol q occurs positively or negatively in some rule whose head predicate is p. Moreover, $\langle p,q \rangle$ is labeled by ¬ if there is at least one rule in P with head predicate p, whose body contains a negative occurrence of q.

A Datalog$^{\neg}$ program P is *stratified* iff DG(P) does not contain any cycle involving an edge labeled by¬. If P is stratified, it is quite easy to construct a stratification of P [1,5,14], that is, a sequence of subprograms $P_0 = EDB$, $P_1,\ldots,P_n$ of P such that $P_0 \cup P_1 \cup \ldots \cup P_n = P$ and, by evaluating them separately and in order from P_0 to P_n, and by applying the CWA to P_k when computing P_{k+1}, the result does not contain any contradiction. Note that stratifications are not unique, that is, if the condition on the $DG(P)$ is satisfied, P can be stratified in several different ways. It is easy to see that the sample program $P = \{r_1,r_2,r_3\}$ is stratified, and that a stratification is $P_0 = EDB$, $P_1 = \{r_1,r_2\}$, $P_2 = \{r_3\}$.

Much further work has been done on the semantics of negation and of non-monotonic programs [1]. Semantics based on various kinds of partial models, like the *stable models* [12] and the *well-founded models* [10] are often studied. An example of more recent work on the subject is [3], where the new concept of soft stratification, based on a new bottom-up query evaluation method based on the Magic Set approach, is proposed.

Key Applications

The research on deductive databases was concentrated in the decade between the mid-eighties and the mid-nineties. This work, and all logic-based approaches to database problems, constitute the foundational experience for speculation that have lead to a number of results in different fields. The field of active databases is one interesting example of the application of the theoretical foundations of Datalog. An active database system is a DBMS endowed with active rules, i.e., stored procedures activated by the system when specific events occur. The processing of active rules is characterized by two important properties: termination and confluence. In [8], a set of active rules is translated into logical clauses, taking into account the system's execution semantics, and simple results about termination and determinism available in the literature for deductive rules are transferred to the active evaluation process. Another, more recent application is the integration of Databases with the Semantic Web, an interesting example of which is the *SWRL (Semantic Web Rule Language)* [15], a proposal combining sublanguages of the OWL Web Ontology Language with the rule-based paradigm. SWRL is (roughly) the union of Horn logic and OWL, where rules are of the form of an implication between an antecedent (body) and consequent (head), with a semantics very similar to that of Datalog.

Cross-references

- ▶ Active Databases
- ▶ Logics and Databases
- ▶ Query Language
- ▶ Semantic Web and Ontology
- ▶ Views

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases. Addison-Wesley, 1995.
2. Bancilhon F., Maier D., Sagiv Y., and Ullman J.D. Magic sets and other strange ways to implement logic programs. In Proc. 5th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1986, pp. 1–15.
3. Behrend A., Soft stratification for magic set based query evaluation in deductive databases. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2003, pp. 102–110.
4. Bocca J.B. EDUCE: a marriage of convenience: Prolog and a Relational DBMS. In Proc. Symp. in Logic Programming, 1986, pp. 36–45.
5. Ceri S., Gottlob G., and Tanca L. Logic Programming and Databases. Springer, Berlin, 1990.
6. Chakravarthy U.S., Minker J., and Grant J. Semantic query optimization: additional constraints and control strategies. In Proc. Expert Database Conference, 1986, pp. 345–379.
7. Chandra A.K. and Harel D. Horn clauses queries and generalizations. J. Log. Program., 2(1):1–15, 1985.
8. Comai S. and Tanca L. Termination and confluence by rule prioritization. IEEE Trans. Knowl. Data Eng., 15(2):257–270, 2003.
9. Kowalski R.A. and Kuehner D. Linear resolution with selection function. Artif. Intell., 2:227–260, 1971.
10. Laenens E. and Vermeir D. Assumption-free semantics for ordered logic programs: on the relationship between well-founded and stable partial models. J. Log. Comput., 2(2):133–172, 1992.
11. Lloyd J.W. Foundations of Logic Programming, 2nd edn. Springer, Berlin, 1987, ISBN 3-540-18199-7.
12. Sacca' D. and Zaniolo C. Stable models and non-determinism in logic programs with negation. In Proc. 9th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1990, pp. 205–217.
13. Tsur S. and Zaniolo C. LDL: a logic-based data language. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 33–41.
14. Ullman J.D. Principles of Database and Knowledge-Base Systems, Computer Science Press, Rockville, MD, USA, 1988.
15. W3C Member Submission. SWRL: a semantic Web rule language combining OWL and RuleML. 21 May 2004, Available at: http://www.w3.org/Submission/SWRL/.

Query Processor

Anastasia Ailamaki, Ippokratis Pandis
EPFL, Lausanne, Switzerland
Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

Query execution engine; Relational query processor; Query engine

Definition

The query processor in a database management system receives as input a query request in the form of SQL text, parses it, generates an execution plan, and completes the processing by executing the plan and returning the results to the client.

Key Points

In a relational database system the query processor is the module responsible for executing database queries. The query processor receives as input queries in the form of SQL text, parses and optimizes them, and completes their execution by employing specific data access methods and database operator implementations. The query processor communicates with the storage engine, which reads and writes data from the disk, manages records, controls concurrency, and maintains log files.

Typically, a query processor consists of four sub-components; each of them corresponds to a different stage in the lifecycle of a query. The sub-components are the query parser, the query rewriter, the query optimizer and the query executor [3].

The parser initially reads the SQL text, renames the table references to the *schema.table* template, and validates the structure. As a second step, it uses the database catalog to check the existence of the referenced tables, as well as ensuring that the user who submitted the specific query has the appropriate privileges for the particular operation on the particular data. If everything succeeds, the output from the parser is a data structure understood internally by both the rewriter and the optimizer. This data structure is handed over to the query rewriter.

The query rewriter modifies the query without changing its semantics. The rewriter replaces references to views as references to base tables, simplifies arithmetic expressions, and applies logical transformations to predicates. After the query is parsed and rewritten (and before it is passed on to the optimizer), the system checks a cache of execution plans of recently optimized queries for an execution plan for the specific query in order to avoid the (usually expensive) optimization phase.

The optimizer generates an efficient execution plan for answering a specific query. The decision on which specific access method or database operator implementation will be used relies heavily on the statistics kept by the system and the selectivity estimation. The output of the optimizer is the query execution plan. In the common case, the query plan is an interpretable dataflow directed acyclic graph, where each node is a specific implementation of a database operation. There are some systems where the optimizer generates directly executable machine code, such as in Daytona [2]. The optimizer can choose from many techniques that can speed up the execution of a query. For example, it may decide to generate a plan where multiple threads or processes work in parallel to answer a specific query. Such an execution strategy works well especially if the machine where the system is running contains multiple processors. Query optimization is the responsibility of a fairly sophisticated software module [1,3,4].

Although the query plan describes in detail the various operations needed for the execution of the query, it is the query executor that contains the algorithms for accessing base tables and indexes, as well as various database operator execution algorithms. The objective of the query executor is to execute the plan as fast as possible and return the answer to the client. The query optimizer and query executor are tightly coupled together. The query executor determines which algorithms implement the plans generated by the optimizer. For instance, if a query executor supports only Hash and Sort-Merge joins, then the optimizer is restricted to producing plans that use only those two join implementations.

Typically, query executors employ the *iterator* model, a simple and intuitive way to filter data. Each operator is implemented as a subclass of the iterator class, using as interface functions such as `init()`, `get_next()`, and `close()`. An iterator can be used as input to any other iterator, thereby enabling universal handling of iterators or iterator combinations by the system, regardless of the particular function they implement. However, recently researchers argue that the overhead of processing data in a tuple-at-a-time, iterator-based fashion leads to inefficient execution of queries when running on modern hardware and deep memory hierarchies [5].

There are many interpretations of what constitutes a query processor. The query executor is the subcomponent that does the "real" job of answering a query. It pulls the data out of the database and employs the various data manipulation algorithms towards them. Thus, a frequent misinterpretation is to consider the query processor and query executor as being synonyms.

Cross-references

- ▶ Hash Join
- ▶ Iterator
- ▶ Parallel Query Processing
- ▶ Query Optimization
- ▶ Query Plan
- ▶ Query Rewriting
- ▶ Sort-Merge Join

Recommended Reading

1. Graefe G. The cascades framework for query optimization. Q. Bull. IEEE TC on Data Engineering, 18(3):19–29, 1995.
2. Greer R. Daytona and the fourth-generation language Cymbal. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 525–526.
3. Hellerstein J.M., Stonebraker M., and Hamilton J. Architecture of a database system. Foundations and Trends in Databases, 1(2):141–259, 2007.
4. Selinger P.G., Astrahan M., Chamberlin D., Lorie R., and Price T. Access path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.
5. Zukowski M., Boncz P., Nes N., and Heman S. MonetDB/X100 – a DBMS in the CPU cache. Q. Bull. IEEE TC on Data Engineering, 28(2):17–22, 2005.

Query Reformulation

▶ Web Search Query Rewriting

Query Rewriting

Evaggelia Pitoura
University of Ioannina, Ioannina, Greece

Synonyms

Query transformations

Definition

Query rewriting is one of the initial phases of query processing. After the original query is parsed and translated into an internal representation, query rewrite transforms it to an equivalent one by carrying out a number of optimizations that are independent of the physical state of the system. Typical transformations include un-nesting of subqueries, views expansions, elimination of redundant joins and predicates and various other simplifications.

Key Points

Query rewriting is one of the phases of query processing. It refers to the application of a number of transformations to the original query in order to produce an equivalent optimized one. Such transformations do not depend on the physical state of the system (such as the size of the relations, the system workload, etc). They are usually based on well-defined rules that specify how to transform a query expression into a logically equivalent one.

The goal of query rewriting is threefold: (i) the construction of a standardized starting point for query optimization (standardization), (ii) the elimination of redundancy (simplification), and (iii) the construction of expressions that are improved with respect to evaluation performance (amelioration).

To satisfy this goal, common responsibilities of the query rewriter include:

- View expansion
- Logical rewriting of predicates. For example, improving the match between expressions and the capabilities of index-based access methods
- Various semantic optimizations such as elimination of redundant joins and predicates
- Sub-query flattening

Typically, query rewriting is performed after parsing the original query. It can be thought of as either being the first part of query optimization or as an independent component preceding the query optimizer and the generation of the alternative execution plans. Rewriting is particularly important for complex queries, including queries with many sub-queries or many joins.

Cross-references

- ▶ Query Optimization
- ▶ Query Processing

Recommended Reading

1. Hellerstein J.M., Stonebraker M., and Hamilton J. Architecture of a database system. Foundations and Trends Databases, 1(2):141–259, 2007.
2. Jarke M. and Koch J. Query optimization in database systems. ACM Comput. Surv., 16(2):111–152, 1984.
3. Pirahesh H., Hellerstein J.M., and Hasan W. Extensible/rule based query rewrite optimization in starburst. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1992, pp. 39–48.

Query Rewriting Using Views

▶ Answering Queries Using Views

Query Suggestion

▶ Web Search Query Rewriting

Query Transformations

▶ Query Rewriting

Query Translation

Zhen Zhang
University of Illinois at Urbana-Champaign, Urbana,
IL, USA

Synonyms

Query translation; Query mapping

Definition

Given a source query Q_s over a source schema and a target query template over a target schema, query translation generates a query that is *semantically closest* to the source query and *syntactically valid* to the target schema. The semantically closest is measured by a closeness metrics, typically defined by precision and/or recall of a translated query Versus a source query over a database content. Syntax validness indicates the answerability of a translated query over the target schema. Therefore, the goal of query translation is to find a query that is answerable over the target schema and meanwhile retrieves the closest set of results as the source query would retrieve over a database content.

Historical Background

Query translation is an essential problem in any data integration system and has been studied extensively in the database area. Since a data integration system needs to integrate many different sources, query translation is thus needed to mediate heterogeneous query capabilities presented by those sources. A source typically only accepts and processes queries of certain formats. Such restrictions on acceptable queries form the query capability of the source. For instance, a Web database may only accept queries through their Web query interfaces, a relational database may accept SQL queries, and a legacy system may only accept selection queries over certain attributes through their wrappers.

Foundations

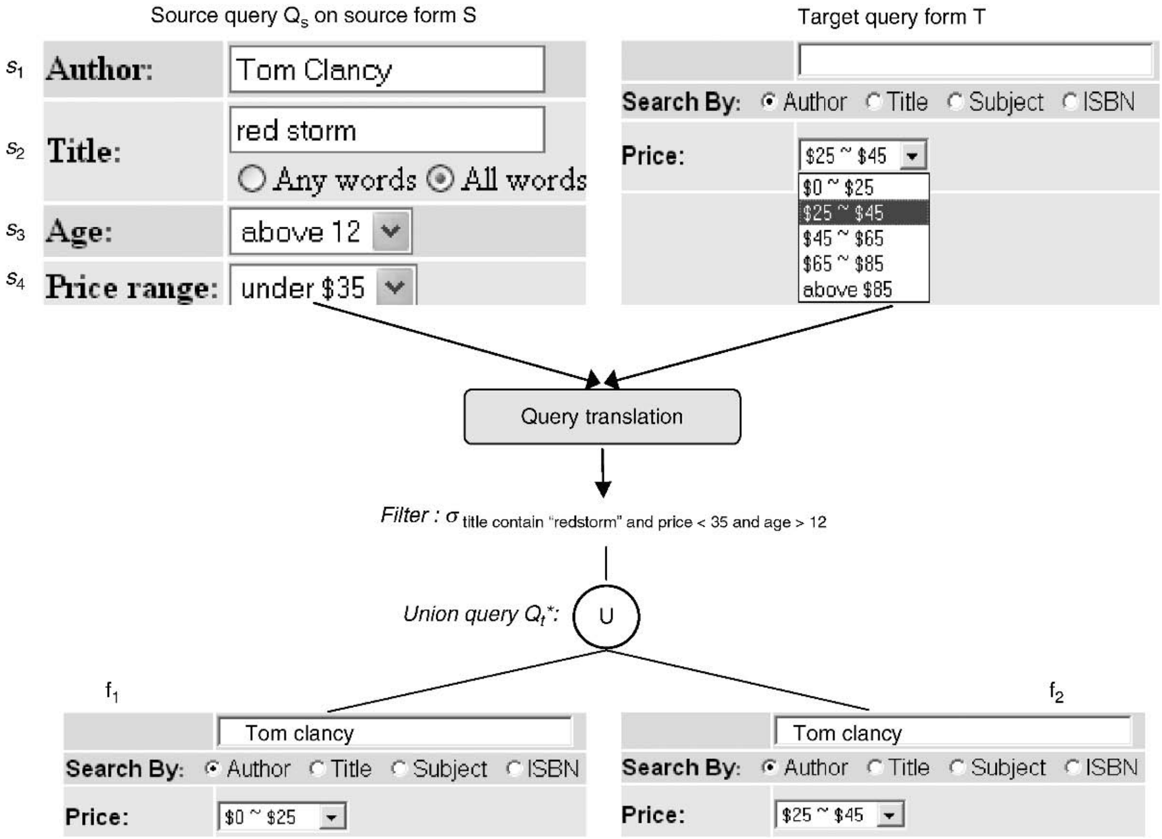
To represent query capabilities, different description languages have been proposed. Vassalos [13] proposes p-Datalog, a datalog-like language for describing query capabilities. Halevy [8] uses capability records to describe, for accepted queries, their binding requirements of input parameters and the attribute name of output parameters. Rajaraman [12] proposes query templates in the format of parameterized queries to specify binding patterns of acceptable queries. Similarly, Zhang [15] uses predicate templates and form templates to represent query capabilities of Web databases through Web query interfaces. As an example, Fig. 1 shows two query forms with different query capabilities. The source query form S accepts conjunctive queries over four predicates: such as s_1 : [`author`; *contain*; Tom Clancy], s_2 : [`title`; *contain*; red storm], s_3 : [`age`; $>$; 12], and s_4 : [`price`; $\leq$; 35], i.e., $Q_s = s_1 \wedge s_2 \wedge s_3 \wedge s_4$. A target query form T supports predicate templates on `author`, `title`, `subject`, `ISBN` one at a time with an optional template on predicate price.

More specifically, the heterogeneity of query capabilities can be categorized into three levels:

Attribute Heterogeneity

Two sources may query a same concept using different attribute names. For instance, the source schema S in Fig. 1 supports querying the concept of reader's age, while the target schema T does not. Also, S denotes book price using price range, while T using price.

Q


Source query Q_s on source form S
s_1 Author: Tom Clancy
s_2 Title: red storm
○ Any words ● All words
s_3 Age: above 12
s_4 Price range: under $35
Target query form T
Search By: ● Author ○ Title ○ Subject ○ ISBN
Price: $25 ~ $45
$0 ~ $25
$25 ~ $45
$45 ~ $65
$65 ~ $85
above $85
Query translation
Filter : $\sigma_{\text{title contain "redstorm" and price < 35 and age > 12}}$
Union query Q_t^*:
U
f_1
Tom clancy
Search By: ● Author ○ Title ○ Subject ○ ISBN
Price: $0 ~ $25
f_2
Tom clancy
Search By: ● Author ○ Title ○ Subject ○ ISBN
Price: $25 ~ $45

Query Translation. Figure 1. Form assistant: A translation example.

Predicate Heterogeneity

Two sources may use different predicates for the same concept. For instance, the price predicate in T has a different set of value ranges from those of S. As a result, a translated target predicate can only be as "close" to the source predicate as possible. Therefore, a closeness metrics needs to be introduced to set up a goal of translation. For instance, a *minimal subsumption* translation requires that a translated target query subsume the source query with fewest extra answers.

Query Structure Heterogeneity

Two sources may support different sets of valid combinations of predicates. In the above example, the target schema T only supports queries on one of the four attributes `author`, `title`, `subject` and `ISBN` at a time with an optional attribute `price`. Therefore, T cannot query author and `title` together, while S can.

The goal of query translation is to generate an appropriate query expressed upon the target schema T. Such a query, as Fig. 1 shows, in general, consists of two parts: a *union query* Q_t^* which is a union of queries upon the target schema to retrieve relevant answers from a target database, and a *filter* σ which is a selection condition to filter out false positives retrieved by Q_t^*. To minimize the cost of post processing, i.e., filtering, translation aims at finding a union query Q_t^* that is as "close" to the source query Q_s as possible so that it retrieves fewest extra answers. Q_t^* in Fig. 1 is such a query.

To realize the translation, query translation needs to reconcile the heterogeneities at the three levels – attribute, predicate and query. Techniques have been studied extensively for addressing the heterogeneity at each level.

Schema Matching for Attribute Heterogeneity

Schema matching (e.g., see the survey of [11]) focuses on mediating the heterogeneity at the attribute level. Recent schema matching approaches follow two different forms – pairwise matching and holistic matching. The pairwise matching approaches (e.g., [3,7]) take two schemas as input, and find best attribute matchings between the two. The holistic matching approaches, pursued by, e.g., [5,6,14], take a collection of schemas as input and generate a set of matchings over all these schemas. Different approaches suit different application settings. The holistic matching approaches are suitable for applications that would dynamically query a large scale of sources simultaneously, while the pairwise matching approaches are suitable for applications that query a small set of pre-configured sources.

Predicate Mapping for Predicate Heterogeneity

Predicate mapping focuses on addressing the heterogeneity at the predicate level. Existing solutions can be categorized into two categories: static predicate mapping mechanism and dynamic predicate mapping mechanism. The static predicate mapping mechanism works with a set of pre-configured data sources. It predefines mapping knowledge between a source schema and a target schema. In such scenarios, it is common, e.g., as [1] studies, to use pairwise rules to specify the mapping. Figure 2 gives some example rules that encode the mapping knowledge required for translation in the example of Fig. 1.

In contrast, the dynamic predicate mapping mechanism works with dynamically discovered sources in a domain. It does not encode the mapping knowledge for specific sources, but instead defines common domain-based translation knowledge that handles most sources in a domain. Such a system may use rules to encode domain knowledge or alternatively may use a search-driven mechanism to dynamically search for the best mapping. Such a search-driven mechanism "materializes" the semantics of a query as results over a database. For instance, to realize rule r_3 in Fig. 2 in the search mechanism, the dynamic mechanism projects both the source and target predicates onto an axis of real numbers, and thus compares their semantics based on their coverage. Finding the closest mapping thus naturally becomes a search problem – to search for the ranges expressible in the target form that minimally cover the source predicate.

Query Rewriting for Query Structure Heterogeneity

Capability-based query rewriting focuses on mediating the heterogeneity at query structure level. Most query rewriting works [4,8,9,10,12] are studied for data integration systems following a *mediator-wrapper* architecture, where a global mediator integrates local data sources through their wrappers. Query rewriting specifically studies the problem of how to mediate a "global" query (from the mediator) into "local" subqueries (for individual sources) based on their query capabilities. There are two basic approaches for addressing query rewriting in data integration system – *global as view* (GAV) and *local as view* (LAV). In global as view, each relation in the mediated (global) schema is defined as a view over schemas of local data source. Query rewriting in GAV is straightforward – simply replacing relation names in a query (over global schema) with their view definitions will yield a valid rewriting of query (over local schemas). In contrast, in local as view, each relation of a local schema is defined as a view over the mediated global schema. Query rewriting in LAV is thus to find a query plan or query expression which uses only views (i.e., local schemas) to answer queries over global schema. In particular, this problem is often abstracted as *answering queries using views*. For a thorough survey of related techniques for answering query using views, please refer to [4].

Key Applications

Query translation is a key component in any data integration system. The broad range of applications for data integration gives rise to the diverse applications of query translation. Two examples are:

Vertical Integration Systems

A vertical integration system integrates information from multiple pre-configured sources (usually in the same domain of data), and thus requires translating queries from a unified query interface to individual data sources. As data sources are usually pre-configured, such a system usually replies on static query translation mechanism such as [1] to handle translation with predefined source knowledge.

r_1 [author; *contain*; \$t] → *emit:* [author; *contain*; \$t]
r_2 [title; *contain*; \$t] → *emit:* [title; *contain*; \$t]
r_3 [price; *under*; \$t] → **if** \$t ≤ 25, *emit:* [price; *between*; 0,25]
elif \$t ≤ 45, *emit:* [price; *between*; 0,25] ∨ [price; *between*; 25,45]
......

Query Translation. Figure 2. Example mapping rules of source *S* and target *T*.

Meta Querying Systems

A meta querying system, e.g., [2], integrates dynamically selected sources relevant to user's queries, and on-the-fly translates user's queries to these sources. As sources are dynamically discovered without predefined source knowledge, such a system needs a dynamic query translation mechanism such as [15] which handles translation without relying on source-specific knowledge.

Cross-references

▶ Information Integration
▶ Query Rewriting
▶ Query Rewriting Using Views
▶ Schema Matching
▶ View-Based Data Integration

Recommended Reading

1. Chen-Chuan C.K. and Garcia-Molina H. Approximate query mapping: Accounting for translation closeness. VLDB J., 10(2–3):155–181, September 2001.
2. Chen-Chuan C.K., He B., and Zhang Z. Toward large scale integration: Building a metaquerier over databases on the web. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 44–55.
3. Doan A., Domingos P., and Halevy A.Y. Reconciling schemas of disparate data sources: A machine-learning approach. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 509–520.
4. Halevy A.Y. Answering queries using views: A survey. VLDB J., 10(4):270–294, 2001.
5. He B. and Cheng-Chuan C.K. Statistical schema matching across web query interfaces. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 217–228.
6. He B., Cheng-Chuan C.K., and Han J. Discovering complex matchings across web query interfaces: A correlation mining approach. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 148–157.
7. Kang J. and Naughton J.F. On schema matching with opaque column names and data values. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 205–216.
8. Levy A.Y., Rajaraman A., and Ordille J.J. Querying heterogeneous information sources using source descriptions. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 251–262.
9. Papakonstantinou Y., Gupta A., Garcia-Molina H., and Ullman J.D. A query translation scheme for rapid implementation of wrappers. In Proc. 4th Int. Conf. on Deductive and Object-Oriented Databases, 1995, pp. 161–186.
10. Papakonstantinou Y., Gupta A., and Haas L. Capabilities-based query rewriting in mediator systems. In Proc. Int. Conf. Parallel and Distributed Information Systems, 1996, pp. 170–181.
11. Rahm R. and Bernstein P.A. A survey of approaches to automatic schema matching. VLDB J., 10(4):334–350, 2001.
12. Rajaraman A., Sagiv Y., and Ullman J.D. Answering queries using templates with binding patterns. In Proc. 14th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1995, pp. 105–112.
13. Vassalos V. and Papakonstantinou Y. Expressive capabilities description languages and query rewriting algorithms. J. Logic Program., 43(1):75–122, 2000.
14. Wu W., Yu C.T., Doan A., and Meng W. An interactive clustering-based approach to integrating source query interfaces on the deep web. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 95–106.
15. Zhang Z., He B., and Chen-Chuan Chang K. Light-weight domain-based form assistant: querying web databases on the fly. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 97–108.

Query Tree

▶ Query Plan

Query Tuning

▶ Application-Level Tuning

Querying DNA Sequences

▶ Query Languages and Evaluation Techniques for Biological Sequence Data

Querying Protein Sequences

▶ Query Languages and Evaluation Techniques for Biological Sequence Data

Querying Semi-Structured Data

▶ Structured Document Retrieval

Queuing Analysis

▶ Performance Analysis of Transaction Processing Systems

Queuing Mechanism

▶ Scheduler

Queuing Systems

▶ Message Queuing Systems

Quorum Systems

Marta Patiño-Martinez
Universidad Politecnico de Madrid, Madrid, Spain

Definition

Replication is a technique that provide scalability and high availability by introducing redundancy. This entry focuses on full replication for simplicity, that is, each node (site) has a copy of the entire database. Therefore, the terms nodes and replicas are used interchangeably. Replication increases performance (scale-out) as access to the data can be distributed across the replicas. Furthermore, the data remains available as long as some replicas are accessible. The most common approach is to execute all write operations (updates) at all replicas in order to keep them consistent while read operations are executed at a single replica. For read-intensive workloads this achieves the desired scalability. However, this approach, known as read-one/write-all (ROWA), has poor availability for write operations (writes cannot operate once a single replica fails) and does not provide scalability under write-intensive workloads. Quorum systems address both of these issues. They reduce the number of copies involved in write operations at the cost of increasing the number of copies involved in read operations. Reducing the number of copies to be accessed implies increased availability, tolerance of network partitions, reduced communication costs and the possibility of balancing the load among replicas.

In order to preserve data consistency of write operations, and a read and a write operation on the same data item must overlap at least at one replica. For that, each operation has to execute on a subset of replicas called a quorum. A *quorum system* over a set of nodes $\mathcal{N} = \{N_1,...N_n\}$ is defined as a collection $\mathcal{S}$ of subsets $S_i \subseteq \mathcal{N}$ with pair-wise non-null intersection. This means, for each $S_i, S_j \in \mathcal{S}$, $S_i \cap S_j \neq \emptyset$. Each subset $S_i \subseteq \mathcal{N}$ is called a *write quorum.* A write operation on a data item of the database must be executed in a write quorum. The requester asks all sites in the quorum for permission. If all of them grant permission, the write operation is executed at all nodes in the quorum. Data items are tagged with versions. If a write operation succeeds in a write quorum S_i, all nodes in S_i set the version of the affected data item to the same value, namely a value higher than any current version among the nodes in the quorum. The non-empty intersection property guarantees that only one quorum can make a decision at a time. That is, if two concurrent write operations on the same data item ask for permission to two write quorums, at least one node is member of both write quorums and gives permission to only one of the write operations. Read operations are executed on a *read quorum*, a set of sites $R_j \in \mathcal{N}$ such that $\forall S_i \in \mathcal{S}, S_i \cap R_j \neq \emptyset$. The requester of the read operation executes the read at all the sites of the read quorum, which return the value read and the version. The requester selects the value corresponding to the highest version. Since a read quorum intersects with all write quorums, it is guaranteed that at least one of the versions read is the latest one.

Historical Background

Quorum systems for data replication were proposed concurrently by [13] and [6] in order to provide availability despite individual node failures and network partitions. *Majority* quorum (also known as *quorum consensus*) [13] exploits the concept of majority to guarantee the intersection property. A quorum can be any majority of nodes. *Weighted voting* [6] generalizes majority by assigning votes to each site and defining a quorum to be a majority over the total number of votes in the system. An overview of early quorum systems is given in [4].

Maekawa [9] initiated a research line for increasing the scalability of quorum systems by reducing their size (in the order of $O(\sqrt{n})$). After this seminal work a large number of quorum systems exploiting different schemes have been proposed, mainly in the nineties. Extensive surveys on quorum systems can be found in [11,7].

Many of the proposed quorum systems exploit geometrical properties to satisfy the intersection property. *Grid quorums* arrange sites as a grid and then define read and write quorums as rows and columns to enforce their intersection. Grids quorums can be rectangular [5] or can have other shapes, such as triangles [11]. Grid quorums were generalized into hierarchical grid quorums by [8]. Another popular way to arrange sites are trees. Tree quorums were introduced by [1].

Quorums have been extended to tolerate Byzantine (arbitrary) failures requiring the intersection of two quorums to be bigger than one. Such quorums have been termed Byzantine quorums [10].

The properties of quorums have been studied extensively, especially availability and scalability. Optimal availability for sites with homogeneous failure probability were studied in [2,3,12]. Scalability (also known as load) has been first studied for symmetric update processing (all the sites in the write quorum fully execute the update transaction or operation) in [11], and then under asymmetric update processing (only one site executes the operations, the others only apply the changes) in [7].

Foundations

There are two basic ways to define quorum systems, plain (or exclusive) quorums and read/write quorums. Read/write quorums allow reduction of cost of read operations by exploiting the knowledge of whether an operation is a read or a write. They are more adequate for data replication where this knowledge is typically exploited. Plain quorums, or just quorums, are typically defined for mutual exclusion purposes, but can also be used for data replication by using exclusive quorums for both reads and writes.

A set system $\mathcal{S}$ is a collection of subsets $S_i \subseteq \mathcal{N}$ of a finite universe $\mathcal{N}$. A quorum system defined over a set of sites $\mathcal{N}$ is a set system $\mathcal{S}$ that fulfils the following property: $\forall S_i, S_j \in \mathcal{S},\ S_i \cap S_j \neq \emptyset$. Given a quorum system $\mathcal{S}$, each $S_i \in \mathcal{S}$ is a quorum. A read-write quorum system, over the set of sites $\mathcal{N}$, is a pair (R,W) where W is a quorum system (write quorums), and R a set system (read quorums) with the following property: $\forall\ W_i \in W,\ \forall\ R_j \in R,\ W_i \cap R_j \neq \emptyset$.

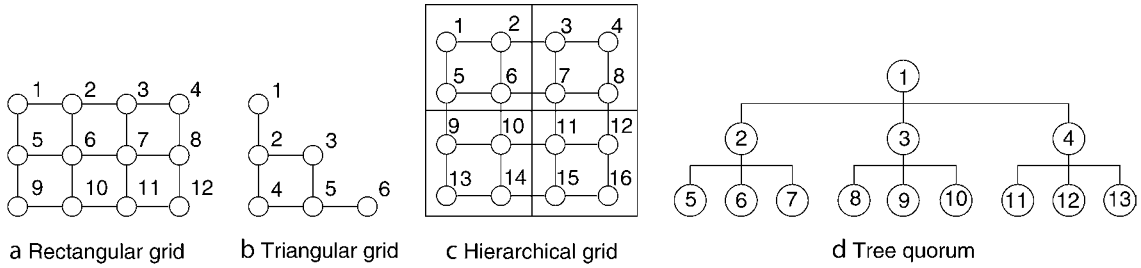
Quorum Types and Their Sizes

For scalability purposes it is beneficial to keep both read and write quorums as small as possible. Furthermore, in order to distribute the load fairly, each node should ideally participate in the same number of quorums. In the following, n denotes the number of nodes in $\mathcal{N}$.

Majority In the majority quorum system [13] read and write quorums must fulfill the following constraints (wq and rq stand for the write and read quorum sizes): $2 \cdot wq > n$ and $rq + wq > n$. The minimum quorum sizes satisfying these constraints are: $2 \cdot wq = n+1$ and $rq + wq = n+1$ and therefore, $wq = \lfloor \frac{n}{2} \rfloor + 1$ and $rq = \lceil \frac{n}{2} \rceil = \lfloor \frac{n+1}{2} \rfloor$. The ROWA approach can be seen as an extreme case of majority, in which $rq=1$ (a single replica is read) and $wq = n$ (all replicas are written). An example of a majority quorum system for three sites (1,2,3) with $rq = wq = 2$ is: {{1, 2}, {2, 3}, {1, 3}}. The majority quorum system is fair since each node has the same probability to be part of a quorum. In weighted majority [6], each site has a non-negative weight (votes). A write quorum consists of nodes such that the sum of their votes is more than half of the total number of votes. A read quorum consists of nodes that have at least half of the total number of votes. Assigning votes allows to adjust to heterogeneous environments where nodes have different processing power and availabilities.

Grids Another family of quorums is grid quorums. The simplest form of grid quorum is the rectangular grid [5]. A rectangular grid quorum organizes n sites in a grid of r rows and c columns (i.e., $n = r \cdot c$). Figure 1a depicts a 3 × 4 rectangular grid. A read quorum consists of accessing an element of each column of the grid ($rq = c$). A write quorum consists of a full column and one element from each of the remaining columns ($wq = r + c - 1$). In the quorum system of Fig. 1a {1,10,7,12} would be a read quorum and {2,6,10,5,3,8} a write quorum. The rectangular grid with the optimal (smallest) quorum size is the square. In this case, $rq = \sqrt{n}$ and $wq = 2 \cdot \sqrt{n} - 1$.

A variation of rectangular grids are *hierarchical grids* [8]. For instance, 16 sites can be configured into a two-level-grid with 2 × 2 grids at each level (Fig. 1c).



Quorum Systems. Figure 1. Different quorum systems.

A hierarchical grid organizes sites into a multi-level hierarchy, such that they reside on the leaves of this hierarchy, while other levels are represented by logical nodes. Each node at level i of the hierarchy (beside leaves) is defined by a rectangular $m \times n$ grid of nodes at level $i+1$. Two constructions are used to build quorums: row covers and full rows. A row cover is formed recursively by selecting a set of $(i+1)$-level nodes where each node pertains to a different row of the grid. A full row is formed recursively by selecting at level i a set of $(i+1)$-level nodes all pertaining to a single row of the grid. A read quorum consists of a row cover. A write quorum consists of the union of a full row and a row cover. A read quorum would be $\{1,6,7,8\}$ and a write quorum would be $\{1,5,6,7,8,10,14\}$. For square grids, this results in a read quorum size of $\sqrt{n}$ and a write quorum size of $2 \cdot \sqrt{n} - 1$, i.e., identical to its non-hierarchical counterpart.

Another way to arrange a grid quorum is a *triangular grid* [11]. Sites in a triangular grid are arranged in d rows such that row i $(1 \le i \le d)$ has i elements (Fig. 1b). A write quorum is defined as the union of one complete row and one element from every row below the full row. Therefore, the quorum size is always d. Read quorums are either a write quorum or an element from each row. In Fig. 1b a sample quorum would be $\{2,3,5\}$.

Triangle quorums are not fair since nodes that are higher in the triangle are more likely to be part of a quorum. In contrast, both rectangular and hierarchical grid quorums are fair.

Trees Another important family of quorum systems is tree quorums. Tree quorums were introduced in [1]. Similar to grid quorums, tree quorums are arranged as a logical structure over the nodes in order to reduce quorum sizes. The nodes are organized into a tree of height h and degree d, i.e., each inner node in the tree has d children. Figure 1d shows a tree with $d=h=3$. A *tree quorum* $q = \langle l, b \rangle$ over a tree with height h and degree b is a tree of height l and degree b constructed as follows. Read and write quorums have the same structure. The quorum contains the root of the tree and b children of the root. Then, recursively for each selected child, b of its children have to be selected, and so on, until a depth l is reached. In the case all nodes are accessible the quorum forms a tree of height l and degree b. If some node is inaccessible at depth h' from the root, the node is replaced by b tree quorums of height $l-h'$ starting from the children of the inaccessible node. In order to guarantee intersecting quorums, quorums must overlap both in height and degree. A read quorum $rq = \langle l_r, b_r \rangle$ and write quorum $wq = \langle l_w, b_w \rangle$ overlap if $l_r + l_w > h$ and $b_r + b_w > d$. Two write quorums overlap if $2 \cdot l_w > h$ and $2 \cdot b_w > d$. Tree quorums are generally not fair since nodes that are closer to the root take part in more quorums.

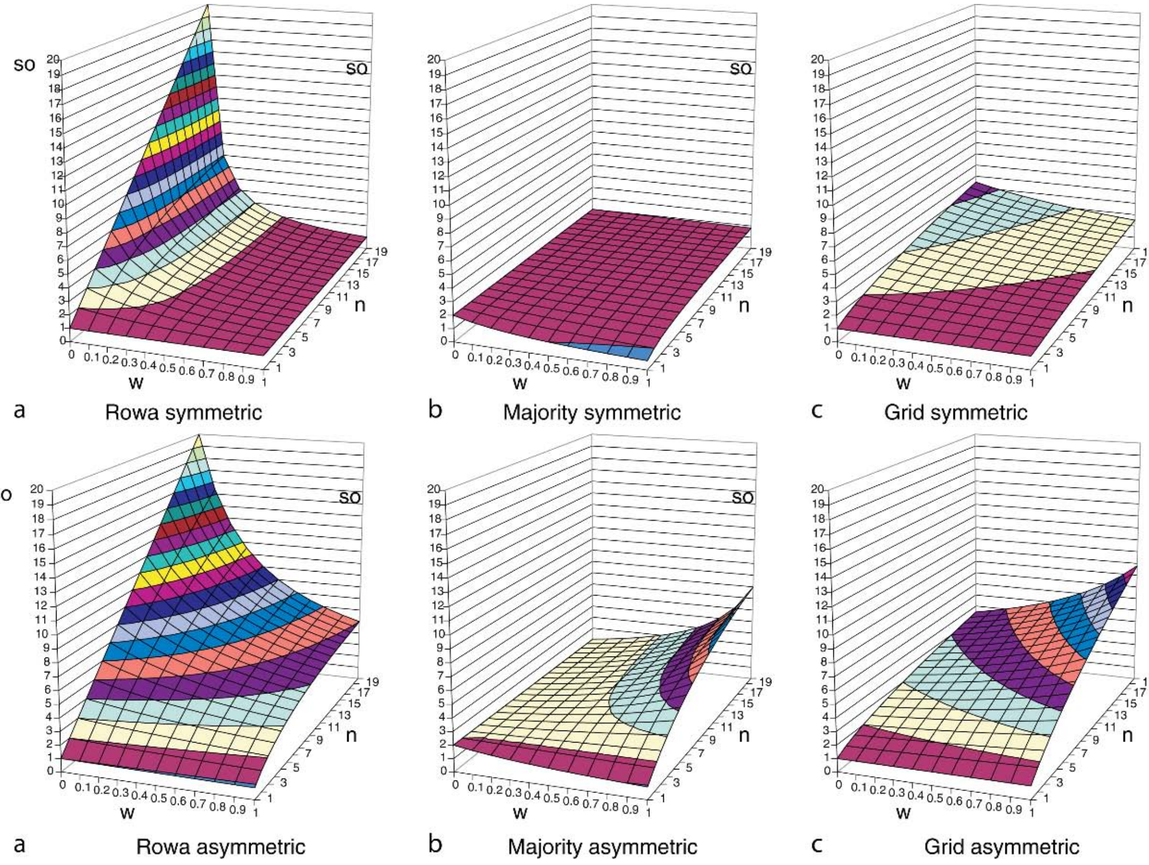
Depending on the values of l and b, different tree quorum systems can be defined. The most well-known is the *ReadRoot* in which a write quorum $wq = \langle h, d/2 + 1 \rangle$, i.e., at each level in the tree a majority of nodes needs to be accessed while a read quorum is $rq = \langle 1, (d+1)/2 \rangle$. That is, all reads go to the root. If the root fails, reads go to the next level while writes cannot be performed anymore. More availability is provided by *MajorityTree* where a majority approach is used both for the degree and height parameters. That is, a read quorum is $rq = \langle (h+1)/2, (d+1)/2 \rangle$ and a write quorum is $wq = \langle h/2 + 1, d/2 + 1 \rangle$. This increases the availability of write operations (since write quorums can be built without the root) but also the access costs of read operations. For the tree depicted in Fig. 1d, {1, 2, 3} is a read and a write quorum, or, if the root is down, {2, 5, 6, 3, 8, 9}.

Availability

Availability of quorums has also been studied extensively. Barbara and Garcia-Molina [3] demonstrated that majority is the most available quorum system for homogeneous failure probabilities (all sites have the same failure probability), if the failure probability p is higher than 0.5. Later, it was shown in [12] that for $p<0.5$ monarchy (all access goes to a single node) was the most available quorum system. When the failure probabilities are heterogeneous, the most available quorum system is weighted majority. The computation of the optimal weights has been provided for the different cases: all sites with $p>0.5$ [14] and the general case in which $0<p<1$ [2]. An extensive comparative study of the availability of different quorum systems is presented in [12].

Experimental Results

A comparison of the performance of quorum systems has been performed by [7,11]. In [7], two different forms of update processing were considered. Using symmetric update processing all the nodes in the write quorum fully execute update transactions (i.e., update operations). In contrast, asymmetric processing lets one node execute the operation while the others only apply the changes. As applying changes typically requires less resources than fully executing the operation, asymmetric processing imposes less load per write operation on the system than symmetric processing. Figure 2 compares the scalability of ROWA, majority and rectangular grids using the analytical model from [7]. The x-axis shows the the fraction of writes ($w = 1.0$ means 100% write operations, $w = 0.0$ means 100% reads). The y-axis shows the total number of nodes (replicas). The z-axis shows the scalability (how many times the throughput of a single-node system is multiplied by the replicated system). The first row of figures shows the performance for symmetric update processing. The scalability is generally very poor, especially for majority and grid. The second row shows results for asymmetric update processing when applying writes has 15% of the costs of fully executing the operation. Scalability



a Rowa symmetric

b Majority symmetric

c Grid symmetric

a Rowa asymmetric

b Majority asymmetric

c Grid asymmetric

Quorum Systems. Figure 2. Scalability of majority.

improves substantially. Majority and grid quorums can improve over ROWA for write-intensive workloads, but at the cost of performing worse in read-intensive environments.

Key Applications

One of the main applications of quorum systems is data replication. However, they are also used for other decentralized control protocols such as distributed mutual exclusion, distributed consensus, Byzantine replication, and group membership.

Future Directions

One of the main open issues with data replication based on quorum systems is how to manage efficiently collections of objects such as tables. Quorum systems might work reasonably well for accesses to individual objects. However, the access of collections of objects has not yet been adequately addressed. When accessing collections of objects the system is forced to collect all the instances of the collection from a read quorum to obtain the latest version of every object and only then, it becomes possible to select the subset of objects from the collection (typically by means of a predicate as in a SELECT statement). This compilation of the full collection from all the sites in the quorum at a single site ruins the performance and scalability of the quorum approach.

Recommended Reading

1. Agrawal D. and Abbadi A.E. The Generalized Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. ACM Trans. Database Syst., 17(4):689–717, 1992.
2. Amir Y. and Wool A. Optimal Availability Quorums Systems: Theory and Practice. Inf. Proc. Letters, 65(5):223–228, 1998.
3. Barbara D. and Garcia-Molina H. The reliability of vote mechanisms. IEEE. Trans. Comput., 36:1197–1208, 1987.
4. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison Wesley, 1987.
5. Cheung S.Y., Ahamad M., and Ammar M.H. The grid protocol: a high performance scheme for maintaining replicated data. In Proc. 6th Int. Conf. on Data Engineering, 1990, pp. 438–445.
6. Gifford D.K. Weighted Voting for Replicated Data. In Proc. 7th ACM Symp. on Operating System Principles, 1979, pp. 150–162.
7. Jiménez-Peris R., Patiño-Martínez M., Alonso G., and Kemme B. Are Quorums an Alternative for Data Replication. ACM Trans. Database Syst., 28(3):257–294, 2003.
8. Kumar A. Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data. IEEE Trans. Comput., 40 (9):996–1004, 1991.
9. Maekawa M. A Algorithm for Mutual Exclusion in Decentralized Systems. ACM Trans. Computer Syst., 3(2):145–159, 1985.
10. Malkhi D., Reiter M.K., and Wool A. The Load and Availability of Byzantine Quorum Systems. SIAM J. Comput., 29(6): 1889–1906, 2000.
11. Naor M. and Wool A. The Load, Capacity, and Availability of Quorum Systems. SIAM J. Comput., 27(2):423–447, 1998.
12. Peleg D. and Wool A. The Availability of Quorum Systems. Information and Computation, 123(2):210–223, 1995.
13. Thomas R.H. A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases. ACM Trans. Database Syst., 4(9):180–209, 1979.
14. Tong Z. and Kain R.Y. Vote Assignments in Weighted Voting Mechanisms. In Proc. 7th Symp. on Reliable Distributed Syst., 1988, pp. 138–143.

R

RAID

▶ Redundant Array of Independent Disks

Random Access Memory (RAM)

▶ Main Memory

Randomization Methods to Ensure Data Privacy

ASHWIN MACHANAVAJJHALA, JOHANNES GEHRKE
Cornell University, Ithaca, NY, USA

Synonyms

Perturbation techniques

Definition

Many organizations, e.g., government statistical offices and search engine companies, collect potentially sensitive information regarding individuals either to publish this data for research, or in return for useful services. While some data collection organizations, like the census, are legally required not to breach the privacy of the individuals, other data collection organizations may not be trusted to uphold privacy. Hence, if U denotes the original data containing sensitive information about a set of individuals, then an untrusted data collector or researcher should only have access to an *anonymized* version of the data, U^*, that does not disclose the sensitive information about the individuals. A randomized anonymization algorithm R is said to be a *privacy preserving randomization method* if for every table T, and for every output $T^* = R(T)$, the privacy of all the sensitive information of each individual in the original data is provably guaranteed.

Historical Background

This is a brief survey of state of the art randomization methods. The reader is referred to the classical survey by Adam and Wortman [1] for a more comprehensive description of older randomization techniques.

Existing literature can be classified based on whether the individuals sharing the data trust the data collector or not. The untrusted data collector scenario is discussed first. Randomization methods have been historically used to elicit accurate answers to surveys of sensitive `yes/no` questions. Respondents may be reluctant to answer such questions truthfully when the data collector (surveyor) is untrusted. Warner's classical paper on *randomized response* [17] proposed a simple technique, where each individual i independently randomized the answer as follows: i answers truthfully with probability p_i, and lies with probability $(1 - p_i)$. Randomized response intuitively ensures privacy since no individual reports the true value. However, Warner did not formalize this intuition.

Subsequent works [7,2] generalized the above randomized response technique to other domains. Evfimievski et al. [7] studied the problem where individuals share itemsets (e.g., a set of movies rented) with an untrusted server (e.g., an online movie rental company) in return for services (e.g., movie recommendations), and they proposed a formal definition of privacy breaches. They invented a provably private randomization technique where users submit independently randomized itemsets to the server. They also proposed data reconstruction algorithms to help the server mine association rules from these randomized itemsets, and experimentally illustrated the accuracy of the reconstruction techniques. The above methods are called *local randomization techniques* since every individual perturbs his/her data locally before sharing it with the data collector.

In the trusted data collector scenarios, while the individuals trust the data collector, they may not trust any third party with whom their data is shared. Hence, randomization methods have been proposed to help privately share the collected data. These techniques can be broadly categorized into *input randomization* and *output randomization* techniques. Input randomization techniques publish a perturbed version of the table; queries are answered using the perturbed data. Output randomization techniques, on the other hand, execute queries on the real data, and return perturbed answers.

One thread of work on input randomization techniques was initiated by Agrawal and Srikant [3]. They proposed an input randomization technique wherein 0-mean random noise is added to the numeric attributes of each individual in the table. The algorithm was experimentally shown to be utility preserving. Nevertheless, Kargupta et al. [10] and Huang et al. [9] showed that adding noise independently to each record in the table does not guarantee privacy.

Yet another thread of work involves publishing synthetic data that has the same properties as the original data, but preserves privacy. Synthetic data generation is a very popular technique in the statistics community, and real applications (like OnTheMap [13]) publish sensitive information using this technique. First proposed by Rubin [16], these techniques build a statistical model using a noise infused version of the data, and then generate synthetic data by randomly sampling from this model. While much research has focused on deriving variance and confidence estimators from synthetic data, only recently has the privacy of these techniques been formally analyzed [12,15].

Among output randomization techniques, the SULQ framework proposed by Blum et al. [5] stands out since it has provable guarantees of privacy. Here, numeric query answers are perturbed by adding Laplace noise. Unlike in input perturbation techniques, privacy is guaranteed if and only if the number of queries that are answered is sub-linear in the number of entities in the table. Nevertheless, Blum et al. show that a large number of useful data mining tasks can be performed using this framework. However, exploratory research could be hindered in this framework, since the researchers need to formulate their queries before seeing the data.

Foundations

Local Randomization Techniques

Let U be the original data and let D_U be its (potentially multi-dimensional) domain. Each record $u \in U$ corresponds to the sensitive information of a distinct individual. Each u is independently randomized using a perturbation matrix **A**. The entry $\mathbf{A}[u, v]$ describes the transition probability $\Pr[u \rightarrow v]$ of perturbing a record $u \in D_U$ to a value v in the perturbed domain D_V. This random process maps to a Markov process, and the perturbation matrix **A** should therefore satisfy the following properties:

$$\mathrm{A} \geq 0, \qquad \sum_{v \in D_V} \mathbf{A}[u, v] = 1 \ \forall_u \in D_U \tag{1}$$

Privacy

Since each record $u \in U$ is perturbed independent of the rest of the records, it is sufficient to reason about the privacy of each record separately. A *privacy breach* [7] is said to occur if for some predicate ϕ of an individual's private information, the prior belief in the truth of ϕ is very different from the posterior belief in its truth after seeing the randomized record $R(u)$. More precisely, an *upward* (ρ_1, ρ_2) *privacy breach* with respect to a predicate ϕ occurs if

$$\begin{aligned}&\exists u \in U, \exists v \in Dv, \text{ s.t.,}\\ &\Pr[\phi(u)] \leq \rho_1 \text{ and } Pr[\phi(u)|R(u) = v] \geq \rho_2\end{aligned} \tag{2}$$

Similarly, a *downward* (ρ_1, ρ_2) *privacy breach* with respect to a predicate ϕ occurs if

$$\begin{aligned}&\exists u \in U, \exists v \in Dv, \text{ s.t.,}\\ &\Pr[\phi(u)] \geq \rho_1 \text{ and } \Pr[\phi(u)|R(u) = v] \leq \rho_2\end{aligned} \tag{3}$$

A randomization method R is defined to be γ*-amplifying* if

$$\forall v \in D_V, \ \forall u_1, u_2 \in D_U, \ \frac{Pr[u_1 \rightarrow v]}{Pr[u_2 \rightarrow v]} \leq \gamma \tag{4}$$

Evfimievski et al. [8] showed that a local randomization method that is γ-amplifying permits a (ρ_1, ρ_2) privacy breach if and only if

$$\frac{\rho_2}{\rho_1} \cdot \frac{1 - \rho_1}{1 - \rho_2} \geq \gamma \tag{5}$$

Algorithms

Randomized Response: Warner's randomized response technique [17] can be instantiated in this model as follows. Each entry $u \in U$ is a *yes/no* answer given by a distinct individual to a sensitive question Q (e.g., *"Have you ever used illegal drugs?"*). Hence, D_U ={0,1}. In order to preserve privacy, each individual flips a coin with bias p, and answers honestly if the coin lands heads and lies otherwise. The perturbation matrix is the 2 × 2 matrix with $\mathbf{A}[0,0] = \mathbf{A}[1,1] = p$ and $\mathbf{A}[0,1] = \mathbf{A}[1,0] = (1 - p)$.

Given n such perturbed answers, the aggregate answer can be estimated as follows. Let π be the fraction of the population for which the true response to Q is *yes* . Then the expected proportion of *yes* responses is

$$\Pr[\text{yes}] = \pi \cdot p + (1 - \pi) \cdot (1 - p) \tag{6}$$

$$\text{Hence, } \pi = \frac{\Pr[\text{yes}] - (1 - p)}{2p - 1} \tag{7}$$

If m out of the n individuals answered *yes* , then the following $\hat{\pi}$ is an unbiased estimator for π.

$$\hat{\pi} = \frac{\frac{m}{n} - (1 - p)}{2p - 1} \tag{8}$$

Warner also proposed a second randomization technique wherein, instead of lying with probability $(1 - p)$, the respondent answers the question Q honestly with probability p and answers a different innocuous question Q_I with probability $(1 - p)$. For instance, with probability p, the respondent truly answers if she had used illegal drugs, and with probability $(1 - p)$, the respondent flips a coin with bias α and answers *yes* if the respondent got a head. In this case, the probability that the answer to Q_I is *yes* is α. Hence, if m out of the n individuals answered *yes* , then the following $\bar{\pi}$ is an estimator for π.

$$\Pr[\text{yes}] = \pi \cdot p + \alpha \cdot (1 - p) \tag{9}$$

$$\pi = \frac{\Pr[\text{yes}] - (1 - p) \cdot \alpha}{p} \tag{10}$$

$$\bar{\pi} = \frac{\frac{m}{n} - (1 - p) \cdot \alpha}{p} \tag{11}$$

Typically, the innocuous question method is better than the former method, since the estimator $\bar{\pi}$ has a smaller variance than $\hat{\pi}$ when the probability of answering the correct question p is not too small.

Itemset Randomization

Itemset randomization is a useful tool that allows users to privately share their (e.g., shopping) histories with a centralized server in return for recommendation services. Let $\mathcal{I}$ represents the set of all items (e.g., products bought by at least one of the users). Then each entry $u \in U$ corresponds to a set of items $\mathcal{I}_u \subseteq \mathcal{I}$. Suppose, for simplicity, all the itemsets $\mathcal{I}_u$ are assumed to have the same number of items, say m. The server wants to learn the *frequent itemsets*, i.e., itemsets $A \subseteq \mathcal{I}$ whose support $\left(sup(A) := \frac{|\{u \in U | A \subseteq \mathcal{I}_u\}|}{|U|}\right)$ is $\geq s_{min}$. The following *Select-a-Size* algorithm, with parameters ρ and $\{p[j]\}_j{}^m$=0, is used to randomize itemsets.

- Select an integer $j \in [1,m]$, with probability $p[j]$.
- Select a simple random sample of size j of $\mathcal{I}_u$, called $\mathcal{I}'_u$.
- For every $a \in \mathcal{I} - \mathcal{I}_u$, add a to $\mathcal{I}'_u$ with probability ρ.

Evfimievski et al. [8] proved sufficient conditions on the parameters, ρ and $\{p[j]\}_j{}^m$=0, in order for this algorithms to be γ-amplifying while simultaneously maximizing the utility of the randomization method (e.g., maximizing the number of original items retained in the randomized itemset, $|I_u \cap I_u'|$). Algorithms for recovering the original data from the randomized itemsets and unbiased estimators for the mean and the covariance of these estimates are provided in [8].

Output Perturbation Techniques

Again let U be the original data. Output perturbation techniques execute queries on the real data, and then return perturbed answers. More precisely, if Q is a query on the data U, and R is the perturbation algorithm, $R(Q(U))$ is returned as the answer. In any such technique, there needs to be a limit on the number and the type of queries that can be posed to the database; for instance, answering the same query Q a large number of times discloses the exact answer to Q.

Privacy

Recall that each record in U corresponds to a distinct individual, and that the value of every record in U is independent of the other records. If each query accesses only one record in the table, then output perturbation essentially reduces to local randomization and the privacy breach analysis can be used.

When a query Q accesses multiple records ($U_Q \subseteq U$), however, one cannot reason about the privacy of a single record $u \in U$ (and hence, the privacy of an individual i) in isolation from the rest of the records $U_Q - \{u\}$. Moreover, depending on the amount of prior information known about U_Q, the extent of u's disclosure varies. For instance, if u is, say, the salary of Betty, Q is the query returning the total salary of women in the table, and the adversary knows that Betty is the only woman in the department ($|U_Q| = 1$), then disclosing $Q(U)$ discloses the value of u completely.

Differential privacy [6] can be used to quantify privacy in this case. Answering a query $Q(.)$ using $R(Q(.))$ is ε- *differential private* if for every pair of original tables U_1 and U_2 that differ in the value of a single record u, and for every possible answer A,

$$\left|\log\left(\frac{\Pr[R(Q(U_1)) = A]}{\Pr[R(Q(U_2)) = A]}\right)\right| \leq \epsilon \tag{12}$$

Intuitively, the above definition preserves privacy as follows. Consider a worst-case adversary who knows the exact values of all the records in $U - \{u\}$ and who is attempting to discover the value of record u_i. Now, if α_S denote the adversary's prior belief that $u \in S$ ($S \subseteq D_U$), then after seeing the answer $R(Q(U))$, the adversary's posterior belief β_S conditional on his knowledge of the rest of the records in the table is bounded by,

$$\alpha_S / e^{\epsilon} \leq \beta_S \leq e^{\epsilon} \times \alpha_S \tag{13}$$

Algorithms

The SULQ framework, introduced by Blum et al. [6], answers aggregate queries by adding random noise. Let Q be a function $D_U{}^n \rightarrow \mathfrak{R}$ The *sensitivity* of query Q, is the smallest number $S(Q)$, such that

$$\begin{aligned} &\forall U_1, U_2 \text{that differ in one record,} \\ &|Q(U_1) - Q(U_2)| \leq S(Q) \end{aligned} \tag{14}$$

Let Lap(λ) denote the *Laplace* distribution which has a density function $h(y) \propto \exp(-|y|/\lambda)$. Suppose a query $Q(U)$ posed to a database U is answered using $Q(U) + Y$, where $Y \sim \text{Lap}(S(Q)/\varepsilon)$. This perturbation scheme satisfies ε-differential privacy. For every U_1,U_2 that differ in only one record u,

$$\begin{aligned} \frac{\Pr[Q(U_1) + Y = x]}{\Pr[Q(U_2) + Y = x]} &= \frac{h(x - Q(U_1))}{h(x - Q(U_2))} \\ &= \frac{\exp(-|x - Q(U_1)| \times \epsilon / S(Q))}{\exp(-|x - Q(U_2)| \times \epsilon / S(Q))} \end{aligned} \tag{15}$$

$$\leq \exp(\epsilon \times |Q(U_1) - Q(U_2)| / S(Q)) = \exp(\epsilon) \tag{16}$$

This technique is useful when the amount of noise added is small; i.e., when the Q has low sensitivity. Examples of queries with low sensitivity are histograms, linear aggregation queries.

Input Perturbation Techniques

Most research is exploratory; the output perturbation techniques can be inconvenient, since researchers must specify queries before seeing the data. Input perturbation techniques on the other hand publish a perturbed version of the data that the researchers can then directly query. Though the two techniques seem different, technically they are the same; input perturbation uses a single perturbed query over the data, namely the sanitization algorithm. Dwork et al. [6] and Kifer et al. [11] show that in many cases publishing perturbed answers to multiple queries gives more utility than publishing a single perturbed dataset like in input perturbation.

Privacy

A simple technique to perturb the data is to independently add 0-mean noise to each attribute of each record. Let V be a noise matrix, then the perturbed data is $U_p = U + V$. The random noise added to each cell ($v \in V$) is usually either a uniform random variable in $[-\alpha,\alpha]$ or distributed as a Gaussian with 0 mean and a known variance. The privacy of such a scheme is unclear; in fact, Kargupta et al. [10] and Huang et al. [9] showed that the very accurate estimates of the original data can be recovered from such additively perturbed data due to dependencies inherent in U. For instance, suppose an adversary knows that all the records in U have the same value, say z. Then, additive randomization does not guarantee any privacy; the mean of the perturbed data accurately estimates z if there are enough records in U.

Additive randomization can be broken using Principal Components Analysis (PCA). Suppose the data has m dimensions and is perturbed by adding noise independently to each dimension. Usually, different attributes in the data are correlated; hence, it can projected onto a smaller number, $p < m$, of dimensions. The first principle component (PC) of the data is the direction, e_1, along which the data has the highest variance. The i^{th} PC, e_i, is a vector orthogonal to the first $(i - 1)$ PC's with the largest variance. These vectors are the eigenvectors of the covariance matrix of the data. In correlated data, only the variances along p directions are large. However, for the random data, the variances are the same along all directions. The variances of the perturbed data are roughly the sum of the variances of the original data and the random noise. Hence, by dropping $(m - p)$ directions along which the perturbed data has the least variance, while much information is not lost about the original data, a $(1 - p/m)$ fraction of the noise added is removed; this might lead to privacy breaches.

Algorithms

In order to guarantee privacy, the noise added should be correlated to the data. This is ensured by *synthetic data generation*. Here, a statistical model is generated from a noise infused version of the existing data, and synthetic data points are sampled from this model. Noise is introduced into the synthetic data on two counts: the noise infused prior to building the model, and the noise due to random sampling.

Different algorithms for generating synthetic data can be created by varying the synthetic model that is built using the data. One simple technique is based on Dirichlet resampling [12]. Let H denote the histogram of U, i.e., $H=\{f(v)| \, v \in D_U, f(v) = \text{multiplicity of } v \text{ in U}\}$, and let R denote the noise histogram. Then the statistical model is $D(H + R)$, where D denotes the Dirichlet distribution. Synthetic data is generated as follows. Draw a vector of probabilities, X, from $D(H + R)$, and generate m points according to the probabilities in X. The above process is mathematically equivalent to the following resampling technique. Consider an urn with balls marked with values $v \in D_U$ such that the number of balls marked with v equals the sum of the frequency of v in U and the frequency of v in the noise histogram. Synthetic data is generated in m sampling steps as follows. In each sampling step, a ball, say marked v, is drawn at random and two balls marked v are added back to the urn. In this step, the synthetic data point is v.

Machanavajjhala et al. [12] characterized the privacy guaranteed by this algorithm in terms of noise distribution. Specifically, they showed that in order to guarantee ε-differential privacy, the frequency of every $v \in D_U$ in the noise histogram should be at least $m/(e^{\varepsilon} - 1)$. For large m and small ε the noise required for privacy overwhelms all of the signal in the data and renders the synthetic data completely useless. Such large requirements of noise is due to the following worst case requirement of differential privacy. Consider a scenario where an adversary knows that U contains exactly one record u_i that takes either the value v_1 or v_2. Now suppose that in the output sample, every record takes the value v_1. If m is large, then the adversary's belief that $r_u = v_1$ is close to 1. In order to guard against such adversaries, differential privacy requires a large amount of noise. However, the probability that such synthetic data is output is negligibly small. This can be remedied using a weaker (ε, δ)-probabilistic differential privacy definition, where an algorithm is private if it satisfies ε-differential privacy for all outputs that are generated with a cumulative probability of at least (1 − δ). Under this weaker definition, the Dirichlet resampling technique is private with much smaller noise requirements.

Barak et al. [4] propose a solution to publish marginals of a contingency table (i.e., a histogram) using the SULQ framework. Publishing a set of noise infused marginals is not satisfactory; such marginals may not be consistent, i.e., there may not exist a contingency table that satisfies all these marginal contingency tables. Barak et al. solve this problem by adding noise to a small number of Fourier coefficients; any set of Fourier coefficients correspond to a (fractional and possibly negative) contingency table. They show that only a "small" number of Fourier coefficients are required to generate the required marginals, and hence only a small amount of noise (proportional to the size of the marginal domain) is required. The authors employ a linear program solution (in time polynomial in the size of multidimensional domain) to generate the final non-negative integral set of noise infused marginals.

Rastogi et al. [14] propose the $\alpha\beta$ algorithm for publishing itemsets. It is similar to the select-a-size randomization operator. Given an itemset I that is a subset of the domain of all items D, the $\alpha\beta$ algorithm creates a randomized itemset V by retaining items in I with probability $\alpha + \beta$ and adding items in $D - I$ with probability β. This algorithm satisfies a variant of ε-differential privacy. Moreover, the authors show that for queries $Q : 2^D \rightarrow \mathbb{R}$, $Q(I)$ can be estimated as follows:

$$\hat{Q}(I) = (Q(V) - \beta Q(D))/\alpha \qquad (17)$$

where $Q(V)$ and $Q(D)$ are the answers to the query Q on the randomized itemset V and the full domain D, respectively. $\hat{Q}(I)$ is shown to provably approximate Q (I) with high probability.

Key Applications

Privacy preserving techniques have been used in Census applications and various web-applications.

The Dirichlet resampling based synthetic data generation technique is used in the web-based OnTheMap Census application that plots worker commute patterns on the U.S. map to study workforce indicators [13]. Warner's randomized response has been used in eliciting responses to sensitive survey questions.

Future Directions

One common property of all provably private randomization methods is that the probability of perturbing a value $u \in D_U$ to every value $v \in D_V$ (the perturbed domain, which is usually the same as D_U) should be positive. As shown by Machanavajjhala et al. [12], this causes a problem when the size of the domain is very large. For instance, in the On The Map [13] application, the domain D_U is the set of census blocks on the U.S. map and there are about 8 million such blocks. However, given a destination, there is only a few hundred workers commuting to it. Hence, even if a small amount of noise is added to each block on the map, spurious commute patterns will arise in the synthetic data. All of the techniques discussed in this article should be revisited in the context of real scenarios with sparse data.

Cross-references

▶ Association Rule Mining on Streams
▶ Differential Privacy
▶ Principal Component Analysis
▶ Statistical Disclosure Limitation for Data Access
▶ Synthetic Data

Recommended Reading

1. Adam N.R. and Wortmann J.C. Security-control methods for statistical databases: a comparative study. ACM Comput. Surv., 21(4):515–556, 1989.
2. Agrawal R. and Srikant R. Privacy preserving data mining. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 439–450.
3. Agrawal S. and Haritsa J.R. A framework for high-accuracy privacy-preserving mining. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 193–204.
4. Barak B., Chaudhuri K., Dwork C., Kale S., McSherry F., and Talwar K. Privacy, accuracy and consistency too: a holistic solution to contingency table release. In Proc. 26th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2007.
5. Blum A., Dwork C., McSherry F., and Nissim K. Practical privacy: the SuLQ framework. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 128–138.
6. Dwork C., McSherry F., Nissim K., and Smith A. Calibrating noise to sensitivity in private data analysis. In Proc. 3rd Theory of Cryptography Conf., 2006, pp. 265–284.
7. Evfimievski A., Gehrke J., and Srikant R. Limiting privacy breaches in privacy preserving data mining. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2003, pp. 211–222.
8. Evfimievsky A., Srikant R., Gehrke J., and Agrawal R. Privacy preserving data mining of association rules. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 217–228.
9. Huang Z., Du W., and Chen B. Deriving private information from randomized data. In Proc. 23th ACM SIGMOD Conf. on Management of Data, 2004.
10. Kargupta H., Datta S., Wang Q., and Sivakumar K. On the privacy preserving properties of random data perturbation techniques. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 99–106.
11. Kifer D. and Gehrke J. Injecting utility into anonymized datasets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006.
12. Machanavajjhala A., Kifer D., Abowd J., Gehrke J., and Vihuber L. Privacy: from theory to practice on the map. In Proc. 24th Int. Conf. on Data Engineering, 2008.
13. On The Map (Version 2) http://lehdmap2.dsd.census.gov/.
14. Rastogi V., Suciu D., and Hong S. The Boundary Between Privacy and Utility in Data Publishing. Tech. rep., University of Washington, 2007.
15. Reiter J. Estimating risks of identification disclosure for microdata. J. Am. Stat. Assoc., 100:1103–1113, 2005.
16. Rubin D.B. Discussion statistical disclosure limitation. J. Off. Stat., 9(2):461–468, 1993.
17. Warner S.L. Randomized response: a survey technique for eliminating evasive answer bias. J. Am. Stat. Assoc., 60(309): 63–69, 1965.

Range Partitioning

▶ Physical Database Design for Relational Databases

Range Query

Mirella M. Moro
The Federal University of Rio Grande do Sul, Porto Alegre, Brazil

Synonyms

Range search; Range selection

Definition

Consider a relation *R* with some numeric attribute *A* taking values over an (ordered) domain *D*. A *range query* retrieves all tuples in R whose attribute A has values in the interval $[low, high]$. That is, $low \leq R.A \leq high$. The range interval may be closed as above, open (e.g., $low < R.A < high$), or half-open in either side (e.g., $low < R.A \leq high$). A range query can be one-sided (e.g., low $\leq R.A$ retrieves all tuples with R.A value greater or equal to *low*). When $low = high$ the range query becomes an *equality* (or *membership*) query.

Key Points

Range queries involve *numeric* (or numerical) attributes. These are attributes whose domain is totally ordered and thus a query interval (e.g., *[low, high]*) can be formed. In contrast, attributes whose domain is not naturally ordered are called *categorical* (or *nominal*). Range queries correspond to selections and are thus amenable to indexing. The standard access method for a range query on some attribute A is a B+-tree built on the values of attribute A. Since the B+-tree maintains the order of the indexed values in its leaf pages, a range query is implemented as a search for the leaf page with the lower value of the range interval, followed by the accessing of sibling pages until a page that contains the higher value of the range interval is reached. The above discussion considers the one-dimensional range search. Multidimensional range queries are also important. A typical example is the *spatial range query* that retrieves all objects which fall within (or intersect, overlap, etc.) a region (a rectangle specified by ranges in each dimension). Such multidimensional range queries are typically indexed by R-trees.

Cross-references

- ▶ Access Methods
- ▶ B+-Tree
- ▶ Indexing
- ▶ Rtree
- ▶ Spatial Range Query

Range Search

▶ Range Query

Range Selection

▶ Range Query

Rank Swapping

▶ Data/Rank Swapping

Ranked Multimedia Retrieval

▶ Top-k Selection Queries on Multimedia Datasets

Ranked XML Processing

Amélie Marian[1], Ralf Schenkel[2],
Martin Theobald[3]
[1]Rutgers University, Piscataway, NJ, USA
[2]Max Planck Institute for Computer Science, Saarbrücken, Germany
[3]Stanford University, Stanford, CA, USA

Synonyms

Aggregation and threshold algorithms for XML; Approximate XML querying; Top-k XML query processing

Definition

When querying collections of XML documents with heterogeneous or complex schemas, existing query languages like XPath or XQuery with their exact-match semantics are often not the perfect choice. Such exact querying languages will typically miss many relevant results that do not conform to the strict formulation of the query.

Top-k query processing for XML data, which focuses on finding the k top-ranked XML elements to an XPath (or XQuery) query with full-text search predicates, is a particularly appropriate query model for querying semi-structured data when the actual content or structure of the underlying data is not fully known. Challenges in processing top-k queries over XML data include scoring individual answers based on how closely they match the query, supporting IR-style vague search over both content and structure, and ranking the k best answers in an efficient manner.

Historical Background

Non-schematic XML data that comes from many different sources and inevitably exhibits heterogeneous structure and annotations in the form of hierarchical tags and deeply nested XML elements often cannot be adequately searched using pure database-style query languages like XPath or XQuery. Typically, queries either return too many or too few results using only Boolean search predicates. Rather, the ranked-retrieval paradigm needs to be called for, with relaxable search conditions, various forms of similarity predicates on tags and contents, and quantitative relevance scoring. The information retrieval (IR) community has historically focused on scoring documents based on how closely they match a user's keyword query. Intense research on applying IR techniques to XML data has started in the early 2000's and has meanwhile gained considerable attention. Recent IR extensions to XML query languages such as XPath 1.0 Full-Text or the NEXI query language used in the INEX benchmark series [5] reflect this emerging interest in IR-style ranked retrieval over semi-structured data. So far, various work on scoring answers to XML queries have focused on adapting IR-style scoring techniques from unstructured text to the semi-structured world, see, e.g., [2,5,11]. On the IR side, some foray into adding structure to standard IR search has taken place before the advent of XML [13]. The popularity of XML provides an opportunity to combine efforts led separately by both the DB and IR communities and provide robust techniques to query semi-structured data.

Threshold Algorithms

The method of choice for efficient processing of top-*k* similarity queries is the family of threshold algorithms (TA), most notably presented by Fagin et al. [8] and originally developed for multimedia databases and structured records stored in relational database systems (RDBMS). These algorithms rely on making dynamic choices for scheduling index lookups during query execution in order to prune low-scoring candidate items as early as possible. They typically scan precomputed index lists for text terms or attribute values of structured records in descending order of local (i.e., per-term) scores and aggregate these scores for the same data item into a global score, using a monotonic score aggregation function such as (weighted) summation. Based on clever bookkeeping of score intervals and thresholds for the top-*k* matches, these index scans can often terminate early, namely as soon as the final top-*k* results can be safely determined, and thus the algorithm often only has to scan short prefixes of the inverted lists. In contrast to the heuristics adopted by many Web search engines, these threshold algorithms compute exact results and are provably optimal in terms of asymptotic costs.

XML and IR

Efficient evaluation and ranking of XML path conditions is a very fruitful research area. Solutions include various forms of structural joins, multi-predicate merge joins, the staircase join based on index structures with pre- and postorder encodings of elements within document trees [10], and holistic twig joins [6]. The latter, also known as path stack algorithm, is probably the most efficient method for twig queries using a combination of sequential scans over index lists stored on disk and linked stacks in memory. However, these approaches are not dealing with uncertain structure and do not support top-*k*-style threshold-based early termination.

IR on XML data has become popular in recent years. Some approaches extend traditional keyword-style querying to XML data [7,11], introduced full-fledged XML query languages with rich IR models for ranked retrieval [9,19], or developed extensions of the vector space model for keyword search on XML documents. FleXPath [4] was among the first approaches to combine this theme with full-text conditions over search predicates. Meanwhile, various groups have started adding IR-style keyword conditions to existing XML query languages. TeXQuery is the foundation for the W3C's official full-text extensions to XPath 2.0 and XQuery 1.0. TIX and TAX are query algebras for XML that integrate IR-style query processing into a pipelined query evaluation engine. TAX furthermore comes with an efficient algorithm for computing structural joins. Here, the results of a query are scored subtrees of the data, and TAX already provides a threshold operator that drops candidate results with low scores from the result set. TOSS is an extension of TAX that integrates ontological similarities into the TAX algebra.

XIRQL [9], a pioneer in the field of ranked XML retrieval, presents a path algebra based on XQL, an early ancestor of W3C's XQuery, for processing and optimizing structured queries. It combines Boolean query operators with probabilistically derived weights for ranked result output, thus carrying the probabilistic IR paradigm over to the XML case. Finally, XXL [19], specifies a full-fledged, SQL-oriented query language for ranked XML-IR with a high semantic expressiveness that made it stand apart from the Boolean

XQL and XPath language standards being predominant at that time. For ranked result output, XXL leverages both a standard IR vector space model and an ontology-oriented similarity search for the dynamic relaxation of structure and term conditions. TopX [17], the actual successor of XXL, on the other hand, focuses on a smaller, XPath-like, subset of the XXL query language which allows for a radically different query processing architecture that outperforms XXL in terms of efficiency by a large margin.

Foundations

Applying the TA paradigm for inverted index lists to XML ranked retrieval is not straightforward. In a data-centric XML setting, a ranking of query results to a query is typically induced by defining some form of structural similarity, whereas in a more text-centric view (with a rich mixture of XML tags and text contents), ranking is derived from IR-style text relevance measures, or a combination of structural similarity and text relevance. More precisely, the XML-specific difficulties arise from the following challenges:

Query Processing and Index Structures: Relevant intermediate results to a mixture of structural and content-related search conditions must be tested as to whether they satisfy the path conditions of the query, and this may incur repetitive and expensive *random access* to large, disk-resident index structures. Furthermore, instead of enforcing conjunctive query processing, it is desirable to relax path conditions and rather rank documents by a combination of content scores and an additional degree to which the structural query conditions are satisfied. *Incremental path evaluations* are required when the index structures are accessed mostly using efficient *sequential disk access* in order to limit or entirely avoid the more expensive random accesses. Yet all incremental updates to candidate score bounds during the query processing need to stay monotonic, in order to guarantee a correct algorithmic basis for top-*k* query evaluation with early candidate pruning.

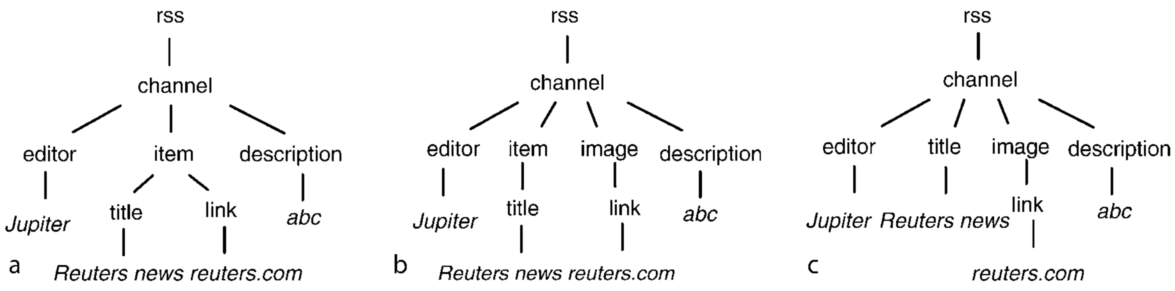
IR Scoring Models and Vague Search: Existing IR scoring models for text documents cannot be directly carried over to the XML case, because they would not consider the specificity of content terms in combination with hierarchical elements or attribute tags. For example, the term "transactions" in a bibliographic data set should be viewed as specific (and lead to a high score) when occurring within elements of type section or caption but be considered less informative within tags like journalname. Furthermore, it should be possible to relax search terms and, in particular, tag names, using tree editing operations, or ontology- and thesaurus-based similarities. For example, a query for a book element about "XML" should also consider a monograph element on "semi-structured data" as a relevant result candidate.

Result Granularity: Scores and index lists refer to individual XML elements and their content terms, but from an IR point-of-view it is desirable to aggregate scores at the document level and return the most relevant XML subtrees, up to the entire XML document, as results. Thus, the query evaluation has to weigh *different result granularities* against each other dynamically in the top-*k* query processing, and the relevance scoring model should consider XML-specific ranking aspects such as *exhaustiveness* and *specificity* when choosing the most suitable result granularity to address the user's information need.

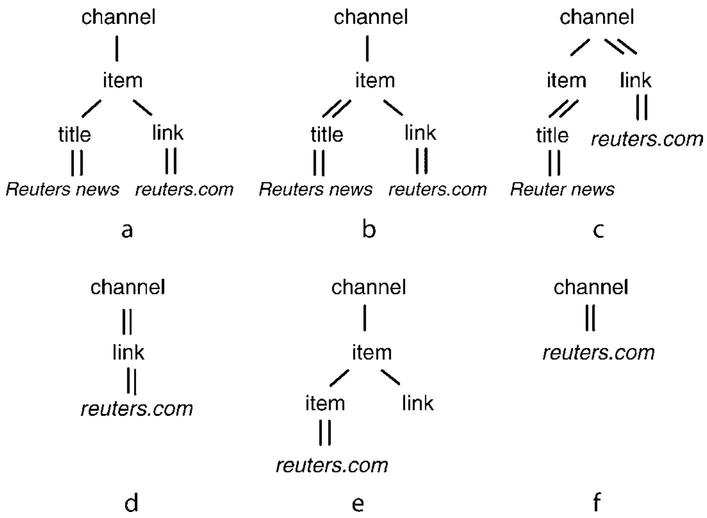
Scoring Structure

Structural similarity is considered in the sense that documents can qualify even if they do not satisfy all path conditions, i.e., if there were too few results otherwise. For dynamically relaxing tag names and structural relationships of tags in path queries, various tree editing operations can be employed, such that only the most similar matches in the collection are returned.

XML data can typically be represented as forests of node-labeled trees. Figure 1 shows a database instance containing fragments of heterogeneous news documents. Figure 2 gives examples of several queries drawn as trees: the root nodes represent the returned answers, single and double edges represent the descendant and child axes, respectively, and node labels stand for names of elements or keywords to be matched. Hence, different queries match the different news documents in Fig. 1. For example, query (a) in Fig. 2 matches document (a) exactly, but would neither match document (b) (since link is not a child of item) nor document (c) (since item is entirely missing). Query (b) matches document (a), also since the only difference between this query and query (a) is the descendant axis between item and title. Query (c) matches both documents (a) and (b) since link is not required to be a child of item while query (d) matches all documents in Fig. 1. Intuitively, it makes sense to return all three news documents as candidate matches, suitably ranked



Ranked XML Processing. Figure 1. Heterogeneous XML database example.



Ranked XML Processing. Figure 2. Query tree patterns and relaxations.

based on their similarity to query (a) in Fig. 2. Queries (b), (c), and (d) in Fig. 2 correspond to *structural relaxations* of the initial query (a) as defined in [4]. In the same manner, none of the three documents in Fig. 1 matches query (e) because none of their title elements contains *reuters.com*. Query (f), on the other hand, is matched by all documents because the scope of *reuters.com* is broader than in query (e). It is thus desirable to return these documents suitably ranked according to their similarity to query (e).

In order to achieve the above goals, [4] defines query relaxations, including *edge generalization* (replacing a child axis with a descendant axis), *leaf deletion* (making a leaf node optional), and *subtree promotion* (moving a subtree from its parent node to its grandparent). These relaxations capture approximate answers but still guarantee that exact matches to the original query continue to be matches to the relaxed query. For example, query (b) can be obtained from query (a) by applying edge relaxation to the axis between `item` and title and still guarantees that documents where title is a child of item are matched. Query (c) is obtained from query (a) by composing edge generalization between item and title and subtree promotion (applied to the subtree rooted at link). Finally, query (d) is obtained from query (c) by applying leaf deletion to the nodes *ReutersNews*, title and item. Query (d) is a relaxation of query (c) which is a relaxation of query (b) which is in turn a relaxation of query (a). Similarly, query (f) in Fig. 2 can be obtained from query (e) by a combination of subtree promotion and leaf deletion. Other works have considered additional query relaxation such as *node renaming*, *node generalization*, *node insertion*, and *node deletion* [1,9,15,16]. The use of schema knowledge can reduce the number of possible relaxed queries by ignoring relaxations that are guaranteed not to lead to additional matches [16].

Amer-Yahia et al. [3] presented strategies to assign scores to query relaxations. These strategies are based on the traditional *tf* · *idf* measure derived from IR-style ranking of keyword queries against an unstructured document collection. The *twig scoring* method introduced in [3] computes the score of an answer taking occurrences of *all* structural and content-related (i.e., keyword) predicates in the query. For example, a match to query (c) would be assigned an *inverse document frequency* score, *idf*, based on the fraction of the number of channel nodes that have a child item with a descendant title containing the keyword *ReutersNews* and a descendant link that contains the keyword *reuters.com*. Such a match would then be assigned a *term frequency* score, *tf*, based on the number of query matches for the specific channel answer.

Scoring Text

A variety of IR-style scoring functions has been proposed and adopted for XML retrieval, ranging from the classic vector space model with its *tf* · *idf* family of scoring approaches, typically using *Cosine* measure for score aggregations, over to the theoretically more sound *probabilistic scoring models*, with *Robertson & Sparck-Jones* and *Okapi BM25* being the most widely used ranking approaches in current IR benchmark settings such as TREC or INEX, up to even more elaborated *statistical language models*. An important lesson from text IR is that the influence of the term and document frequency values, *tf* and *df* – in the following referred to as their element-specific counterparts *ftf* and *ef*, should be sub-linearly dampened to avoid a bias for short elements with a high term frequency of a few rare terms. To address these considerations, the TopX engine, presented by Theobald et al. [17,18], adopts the empirically very successful Okapi BM25 probabilistic scoring model to a generic XML setting by computing individual relevance models for each element type occurring in the collection.

For a typical NEXI query pattern of the form $q = // A[\text{about}(.//, t_1, \ldots, t_m)]$, the following *relevance score* is computed for an element *e* with tag name A:

$$score(e,q) = \sum_{i=1}^{m} \frac{(k_1+1)ftf(t_{i,}e)}{k + ftf(t_i,e)}$$

$$.\log\left(\frac{N_A - ef_A(t_i) + 0.5}{ef_A(t_i) + 0.5}\right)$$

$$\text{with } K = k_1\left((1-b) + b\frac{length(e)}{avg\_length_A}\right)$$

Here, $ftf(t_i, e)$ models the *relevance* of a term t_i for an element's *full content*, i.e., the frequency of t_i in all the descending text nodes of element *e*; while $ef_A(t_i)$ models the *specificity* of t_i for a particular element with tag name A by capturing how many times t_i occurs under a tag A across the whole collection having N_A elements with this tag name.

That is, this extended BM25 model computes a separate relevance model for each term t_i with respect to its enclosing tag name A, thus maintaining detailed element frequency statistics $ef_A(t)$ of each individual tag-term pair that occurs in the collection. It provides a smoothed (i.e., dampened) influence of the *ftf* and *ef* components, as well as a compactness-based normalization that takes the average length of each element type into account. Note that the above function also includes the tunable parameters k_1 and b just like the original BM25 model that now even allows for fine-tuning the influence of the *ftf* components and the length normalization for each element type individually – if desired. For an about operator with multiple keyword conditions as used in the NEXI query language of the INEX benchmark series (or similarly for ftcontains in the XPath 2.0 Full-Text specification), that is attached to an element *e* with tag name A, the aggregated score of *e* is simply computed as the sum of the element's scores over the individual tag-term conditions. For path queries with more than one structural tag condition or with multiple full-text operators, the content scores of each element can be combined with the structural scores described above.

Various extensions for more specific full-text predicates such as keyword proximity and phrase matching, as well as incorporating ontological concept similarities for query expansion, have been proposed in the literature – some of which are leaving some interesting research questions for a top-*k*-style query processor, since most of the more sophisticated proximity- or graph-based compactness measures inherently lead to non-monotonic score aggregation functions.

XML Top-k Query Evaluation Techniques

Combining Structure Indexes and Inverted Lists: Kaushik et al. [12] proposed one of the first, exact-match, top-*k* algorithms for XML by employing various path index operations as basic steps for non-relaxed evaluations of branching path queries. Their strategy combines two forms of auxiliary indexes, a *DataGuide*-like path index for the structure whose extent identifiers are linked to an inverted index for processing relevance-ranked keyword conditions. The index processing steps are then invoked within a TA-style top-*k* algorithm, involving eager random access to these inverted index structures.

XRank: Among the most prominent IR-related approaches for ranked retrieval of XML data is XRank [11]. It generalizes traditional link analysis algorithms such as *PageRank* for authority ranking in linked Web collections and conceptually treats each XML element as an interlinked node in a large element graph. Then the *element rank* of an XML element corresponds to the authority weight computed over a mixture of containment edges, obtained from the XML tree structure, and hyperlink edges, obtained from the inter-document XLink structure, similar to the HTML case. XRank may indeed return deeply nested elements but merely supports conjunctive keyword search; it does not yet support structured and/or path query languages such as XPath. For efficient retrieval of multi-keyword queries, it also uses inverted lists sorted in descending order of element ranks and sketches the usage of standard threshold algorithms for pruning the search space.

FlexPath: FlexPath [4] integrates structure and keyword queries and regards the query structure as templates for the context of a full-text keyword search. The query structure (as well as the content conditions) can be dynamically relaxed for ranked result output according to predefined tree editing operations when matched against the structure of the XML input documents. The FlexPath query processor already comprises the usage of top-*k*-style query evaluations for a slightly modified, XPath-like, query language that later evolved as part of the official W3C Full-Text extensions to XPath 2.0 and XQuery 1.0. Like [12], it uses separate index structures for storing and retrieving the structural and content-related conditions of an XPath 2.0 Full-Text query; it may thus require a substantial amount of random access to disk-resident index structures for resolving the final structure of a result candidate.

Whirlpool: The Whirlpool system introduced by Marian et al. [14] provides a flexible architecture for processing top-*k* queries on XML documents *adaptively*. Whirlpool allows partial matches to the same query to follow different execution plans, and takes advantage of the top-*k* query model to make dynamic choices during query processing. The key features of Whirlpool are: (i) a partial match that is highly likely to end up in the top-*k* set is processed in a prioritized manner, and (ii) a partial match unlikely to be in the top-*k* set follows the cheapest plan that enables its early pruning. Whirlpool provides several adaptivity policies and supports parallel evaluation; details on experimental results can be found in [14].

TopX: The biggest challenge in further accelerating full-text query evaluations over large, semi-structured data collections lies in finding appropriate encodings of the XML data, for indexes that can be read *sequentially* in big chunks directly from disk when the collection (or index) no longer fits into the main memory of current machines. Thus, the TopX engine [17,18] operates over a *combined inverted index* for content- and structure-related query conditions by precomputing and materializing joins over tag-term pairs, the most common query patterns in full-text search. This simple precomputation step makes the query processing more scalable, with an encoding of the index structure that is easily serializable and can directly be stored sequentially on disk just like any inverted index, for example using conventional B^+-tree indexes or inverted files.

At query processing time, TopX scans the inverted lists for each tag-term pair in the query in an interleaved manner, thus fetching large element blocks into memory using only sorted access to these lists and then iteratively joining these blocks with element blocks previously seen at different query dimensions for the same document. Using pre-/postorder tree encodings [10] for the structure, TopX only needs a few final random accesses for the potential top-*k* items to resolve their complete structural similarity to a path query. An extended hybrid indexing approach using a combination of DataGuide-like path indexes and pre-/postorder-based range indexes can even fully eliminate the need for these random accesses – however at the cost of more disk space.

TopX further introduces pluggable extensions for *probabilistic candidate pruning*, as well as a *probabilistic cost-model* for adaptively scheduling the sorted and random accesses, that help to significantly accelerate query evaluations in the presence of additional, precomputed index list statistics such as index list selectivities, score distribution histograms or parameterized score estimators, and even index list (i.e., keyword) correlations. For *dynamic query expansions* of tag and term conditions, TopX can incrementally merge the inverted lists for similar conditions obtained from an exchangeable background thesaurus such as *WordNet* or *OpenCyc*. Thus, TopX provides a whole toolkit of specialized top-*k* operators for efficient full-text search, including *incremental merge operators* for dynamic query expansion and *nested top-k operators* for high-dimensional phrase expansions.

Key Applications

Scalable, Web-Style Search over Semi-structured Collections: Efficient IR over large Web collections will remain one of the most challenging applications for XML-top-*k* query processing with full-text search, with an ever-increasing demand for scalability, interactive runtimes, and vague search involving dynamic query relaxation and/or expansion over heterogeneous collections or unknown schemata.

INEX Benchmark Series: INEX provides a comprehensive forum for IR research on semi-structured data, that goes beyond using the formerly prevalent synthetic data collections such as XMark or XBench for evaluating retrieval quality in true IR-style settings, with a variety of subtasks, XML-IR-specific evaluation metrics, and peer assessments of retrieval results [5].

Future Directions

Graph Top-k: Current XML-top-*k* algorithms are restricted to XML data trees. Future work could focus on further generalizing the scoring approach, in order to handle cycles arising from inter- or intra-document XLinks, with the need to still derive tight and accurate bounds for early candidate pruning. This may involve efficient index structures for arbitrary graphs and potentially non-monotonic score aggregation functions to incorporate graph compactness measures such as *Steiner trees.*

More XQuery: Similarly, current work has only focused on implementing various subsets of the XPath query language. Providing top-*k*-style bounds and pruning thresholds for more complex XQuery constructs such as *loops* and *if-cases* would be an intriguing issue for future work.

Experimental Results

Extensive experiments can be gleaned from the various approaches presented in the literature, see, e.g., [3,4,12,14,17].

Data Sets

Links to the INEX IEEE and Wikipedia collections can be obtained from the INEX homepage: http://inex.is.informatik.uni-duisburg.de

URL to Code

http://topx.sourceforge.net

Cross-references

► Text Indexing and Retrieval
► XML Information Integration
► XQuery Full-Text

Recommended Reading

1. Amer-Yahia S., Cho S., and Srivastava D. Tree pattern relaxation. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002. pp. 496–513.
2. Amer-Yahia S., Curtmola E., and Deutsch A. Flexible and efficient XML search with complex full-text predicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 575–586.
3. Amer-Yahia S., Koudas N., Marian A., Srivastava D., and Toman D. Structure and content scoring for XML. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005.
4. Amer-Yahia S., Lakshmanan L.V.S., and Pandit S. FleXPath: flexible structure and full-text querying for XML. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 83–94.
5. Amer-Yahia S. and Lalmas M. XML search: languages, INEX and scoring. ACM SIGMOD Rec., 35(4):16–23, 2006.
6. Bruno N., Koudas N., and Srivastava D. Holistic twig joins: optimal XML pattern matching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 310–321.
7. Cohen S., Mamou J., Kanza Y., and Sagiv Y. XSEarch: a semantic search engine for XML. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 45–56.
8. Fagin R., Lotem A., and Naor M. Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci., 66(4):614–656, 2003.
9. Fuhr N. and Großjohann K. XIRQL: a query language for information retrieval in XML documents. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001. pp. 172–180
10. Grust T., van Keulen M., and Teubner J. Staircase join: teach a relational DBMS to watch its (axis) steps. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 524–525.
11. Guo L., Shao F., Botev C., and Shanmugasundaram J. XRank: ranked keyword search over XML documents. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
12. Kaushik R., Krishnamurthy R., Naughton J.F., and Ramakrishnan R. On the integration of structure indexes and inverted lists. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
13. Kilpeläinen P. and Mannila H. Retrieval from hierarchical texts by partial patterns. In Proc. 16th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 214–222.
14. Marian A., Amer-Yahia S., Koudas N., and Srivastava D. Adaptive processing of top-k queries in XML. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 162–173.
15. Schenkel R., Theobald A., and Weikum G. Semantic similarity search on semistructured data with the XXL search engine. Inf. Retr., 8(4):521–545, 2005.

16. Schlieder T. Schema-driven evaluation of approximate tree-pattern queries. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 514–532.
17. Theobald M., Schenkel R., and Weikum G. An efficient and versatile query engine for TopX search. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005.
18. Theobald M., Schenkel R., and Weikum G. The TopX DB&IR engine. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1141–1143.
19. Theobald A. and Weikum G. Adding relevance to XML. In Proc. 3rd Int. Workshop on the World Wide Web and Databases, 2000, pp. 105–124.

Ranking

► Web Search Relevance Ranking

Raster Data Management

► Storage of Large Scale Multidimensional Data

Raster Data Management and Multi-Dimensional Arrays

Peter Baumann
Jacobs University, Bremen, Germany

Synonyms

Array databases; Raster databases

Definition

In this enry, the management of raster data is described based on the concept of an array database system. An array database system is a database system that supports the array (also called raster) data structure, as understood in programming languages: a homogeneous collection of data items where each item has a coordinate associated, and coordinates sit at grid points in a rectangular, axis-parallel subset of the Euclidean space $\mathbf{Z}^d$ for some $d > 0$.

Such arrays convey a very regular structure with a well-defined neighborhood relation between its cells, which requires suitable storage structures. Frequently, operations on cells are applied simultaneously to large contiguous portions of arrays, such as overlaying two equally-sized images; efficient query processing needs to provide support for such patterns. Finally, arrays tend to be very large, with single objects frequently ranging into Terabyte and soon Petabyte sizes; for example, today's earth and space observation archives grow by Terabytes a day.

Query language support for arrays consists of array primitives which can be nested as in standard SQL, and preferably integrated into SQL. Once sufficiently declarative query languages are available, query optimization and parallelization can be applied which has been proven to accelerate array evaluation up to orders of magnitude. Storage management must give efficient access to large arrays and sub-arrays thereof. Due to the large sizes, compression often is applied, but some high-volume applications still require storage hierarchies.

Today, an important application domain is Web services on geo raster data, such as 1-D environmental sensor time series, 2-D satellite images, 3-D x/y/t image time series (Fig. 1), 3-D x/y/z exploration data, and 4-D x/y/z/t climate simulation data. Other spatio-temporal applications can be found, e.g., in the Life Sciences. *On-line analytical processing* (OLAP) and Statistical Databases consider arrays as well, often combining the time dimension with abstract axes such as sales and products.

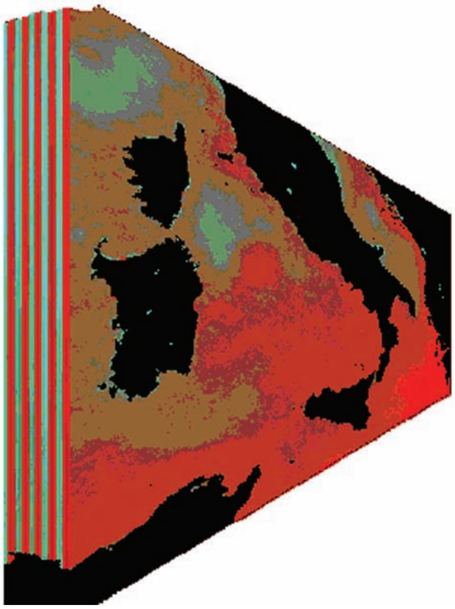
Historical Background

The array paradigm is not directly supported by the relational data model. Therefore, two different approaches have traditionally been pursued to achieve array support: BLOBs and OLAP data *cubes*.

Imagery in Array Databases

If stored in databases, image pixel matrices go into BLOBs (Binary Large OBjects), linearized and encoded in some data exchange format, and result in a byte string whose semantics is unknown to the database system. The database system, therefore, cannot provide any array operation aside from reading and writing the complete BLOB. As a consequence, for many potential application domains, users believe that databases cannot support their raster data, and they still tend to develop their own file-based services in an ad-hoc manner.

The requirement for further array processing capabilities distinguishes array databases from multimedia databases and imaging systems. Array databases are different from multimedia databases which analyze images using some built-in algorithms to obtain



Raster Data Management and Multi-Dimensional Arrays. Figure 1. Retrieval result from a 3-D x/y/t satellite image time series data cube on sea surface temperature; original data cube contains about 10,000 satellite images (image: rasdaman screenshot, data: NASA/DLR).

feature vectors. Subsequently, search is performed on the feature vectors, not the images. Array databases, conversely, do not attempt to interpret the raster data, but always operate on the cell values themselves.

Likewise, array databases are different from image processing and understanding systems. Imaging systems offer elaborate functionality, however are not constrained to excessively large images in relation to main memory size. Array databases, conversely, constrain themselves in functionality to remain safe in evaluation, but aim to have no object size limit.

Statistics Data in Array Databases

Statistical data are usually modeled as OLAP data cubes. Today, the main paradigm is Relational OLAP (ROLAP) where each data item, together with its coordinates, is stored in a so-called fact table in a relational DBMS. This explains why ROLAP is suitable only for very sparse data, where listing only valid points together with their coordinates represents an efficient compression scheme.

Relational query operators, extended with specific data cube operators, provide powerful analysis capabilities on such data cubes.

A Unified View

Although treated differently, both images (here understood to have any spatio-temporal dimensions, not just 2-D x/y axes) and data cubes share characteristics to a large extent. Both consist of a data structure mapping n-D coordinates with rectangular boundaries into a value space. Operations also convey similarities, as the following examples show: Subsetting (cutting out sub-cubes) is known in both worlds, and an OLAP roll-up from days to weeks is equivalent to scaling an image by a factor of seven using linear interpolation.

Actually the main difference motivating different treatment lies in a data property. Statistical data tend to be very sparse, typically only up to 5% of the data cube holds values of interest. Image data, on the other hand, tend to be 100% dense.

Some authors [7] argue that OLAP data cubes are not strictly arrays because, in the case of categorical axes (in OLAP called measures), the neighborhood relation between points does not induce a strict ordering. Others [2] subsume data cubes under the array paradigm, for example because categorical measures ultimately are mapped to integer coordinates during physical modeling.

History and Current State

A first important step beyond BLOBs was accomplished with PICDMS [4] where a 2-D array query language, still procedural and without suitable storage support, has been introduced. A declarative n-D array query language with an algebraic foundation and a suitable architecture, which is implemented and in operational use, has been presented in [2,1]. Marathe and Salem present a 2-D database array language [8]. In [4] nested relational calculus is extended with multidimensional arrays, obtaining a model called NCRA and a query language derived from it, AQL. Both seem to be more theoretically motivated and aim in particular at complexity studies for array addressing. Mennis et al. [9] introduce 3-D map algebra, a framework suitable for handling 2-D and 3-D geo raster data.

While OLAP is a well established business today, arrays are considered by industry only recently. ESRI's ArcSDE and Oracle's GeoRaster cartridge offer tiled storage of 2-D maps with mainly spatial subsetting support; the rasdaman system offers n-D full query support.

Foundations

The collection paradigm in databases encompasses sets, bags, lists, and arrays. The array concept forms a separate, distinct information category. While it might be emulated, e.g., by nested lists, this will not lead to usable query concepts (e.g., for slicing a data cube along all its axes), nor can this be a basis for efficient implementations.

Consequently, all database aspects need to be reconsidered for array support, including conceptual modeling (what are appropriate operations?), storage management (how to manage objects spanning many disk blocks, if not several media?), and query evaluation (what are efficient and optimizable processing strategies?).

Conceptual Modeling

Formally, an array a is a function a: $X \rightarrow F$ where X is a finite axis-parallel hypercube in Euclidean space $\mathbf{Z}^d$ for some $d > 0$ and F is some algebra. Let X be the array's domain, F its cell type, and the individual locations $x \in X$ carrying some value $f \in F$ the array's cells.

Following good practice in databases, an array query language should be declarative and safe in evaluation. Declarative in this context means that there is no explicit iteration sequence over an array (or part of it) during evaluation – conceptually, all cells should be inspected simultaneously. This opens up avenues for efficient storage and evaluation patterns, and query optimization in general (see below). Avoiding explicit iterations also contributes to be safe in evaluation, i.e., every query is evaluated in a finite number of (finite-time) steps.

Actually it turns out that a wide range of practically relevant operations can be expressed using only a few primitives [2]. The MARRAY operator creates an array over some given domain and sets its cells by evaluating an also given expression at each cell location. An application of the MARRAY operator has the general form

marray *index-range-specification*
values *cell-value-expression*

where *index-range-specification* defines the resulting array's domain and binds an iteration variable to it. In *cell-value-expression* the value of the cells are determined for each cell coordinate; this expression may or may not use the current value of the iteration variable, e.g., to read out cells from some given array. The first example, domain subsetting, makes use of this addressing by copying the values of cells within the cutout region from the original array into the new array.

Example: "A cutout of array *a* specified by the corner points (100,100) and (200,300)."

marray *p* in [100:200,100:300]
values *a*[*p*]

In the query language this particular application pattern of MARRAY can be abbreviated as

a[100:200,100:300]

Aside from copying cells as above, the resulting array can also have new values assigned which may or may not depend on other arrays.

Example: "Array *a* (which is known to be of size 1024 × 768), with intensity reduced by a factor of 2." Auxiliary function dom(*a*) returns the domain of *a*, over which the query has to iterate:

marray *p* in dom(*a*)
values *a*[*p*]/2

Such operations are abbreviated in the query language by applying the operation on hand directly to the array, in the example obtaining:

a/2

All unary and binary operations on cells can thus be lifted to become array operations.

Example: "Array *a*'s values where they are above threshold *t*, and 0 otherwise."

$(a > t) * a$

In this example, Boolean and arithmetic operations are combined. Similar to many programming languages, Boolean results from the comparison are interpreted as 0 and 1 for the subsequent multiplication.

The CONDENSE operator aggregates cell values into one scalar result, similar to SQL aggregates. Its application has the general form

condense *condense-op*
over *index-range-specification*
using *cell-value-expression*

where, like with MARRAY before, *index-range-specification* indicates the array index domain to be iterated over and binds a variable to it. Similarly, *cell-value-expression* represents an expression to be evaluated for each index range coordinate. The additional element, *condense-op*, specifies the aggregating operation used to combine the cell value expressions into one single value.

Example: "The sum of all values in *a*."

condense+
over p in sdom(a)
using $a[p]$

This is abbreviated as *add_cells*(a); further predefined condensers include *count_cells*(), minimum, maximum, and quantifiers.

Histogram computation demonstrates nesting of these operations.

Example: "A histogram of 256 buckets over 8-bit greyscale array *a*."

marray n in [0:255]
values count_cells($a = n$)

The inner expression, $a = n$, is an application of the formerly introduced "lifted" operation. For each coordinate p of a, the comparison $a[p] = n$ is performed, yielding a Boolean array.

The SORT operator allows slicing of an array along one of its axes and reordering of slices according to some sorting criterion.

By embedding this into a set-oriented model with array-valued attributes one can write queries over sets of arrays. The next example assumes a table *BrainActivationMaps* where one of the attributes, named *data*, is an array. The data array might contain processed scans of human brains indicating activity like electrical flow, temperature, blood pressure, etc. Note that for the query it does not matter whether the *data* array is, say, 2-D or 3-D. This way queries can be kept rather generic.

Example: "Tuple identifier and histogram of all those *BrainActivationMaps* tuples where the average data intensity exceeds threshold 127."

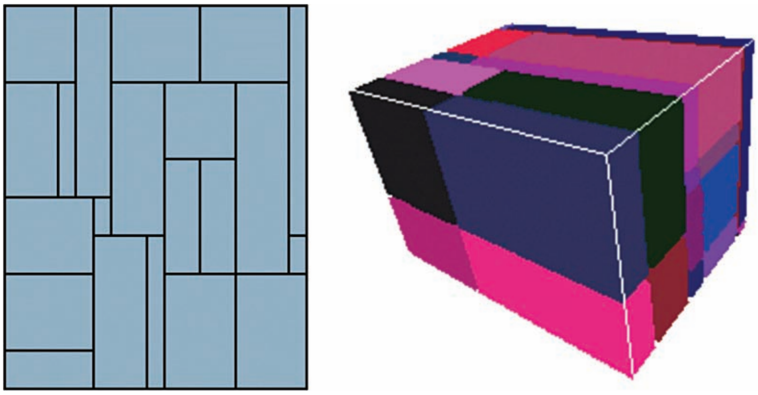
select *id*,
marray n in [0:255]
values *count_cells*(*bam.data* = n)
from *BrainActivationMaps* as *bam*
where *avg_cells*(*bam.data*) > 127

Such languages allow formulating statistical and imaging operations which can be expressed analytically without using loops. In [6] it has been proven that the expressive power of such array languages in principle is equivalent to relational query languages with ranking.

Physical Modeling

In almost all practically relevant scenarios, array objects are by orders of magnitude larger than disk blocks. The foremost task for an array storage manager is to preserve spatial proximity on disk. For n-D data this obviously is impossible to achieve on a linear storage space, hence approximations need to be employed. One such technique is tiling, adopted from imaging. Tiling (also called chunking) partitions an n-D cube into a set of non-overlapping n-D subcubes (Fig. 2), each of which is stored in a BLOB in the traditional manner. The partitioning pattern can be chosen freely, which opens up a wide space for storage optimization. In the most simple case, equally sized tiles are employed where the tile size is chosen suitably for the disk and bus specs; an advanced alternative is to determine an optimal tile distribution with individual tile proportions and sizes for a given query set.

A spatial index, such as the R-Tree, helps to quickly determine the tiles affected by a query, which usually is a *range query*. As opposed to spatial (i.e., vector) databases the situation is simple: the target objects, which



Raster Data Management and Multi-Dimensional Arrays. Figure 2. Sample tiling of 2-D and 3-D arrays.

have a regular box structure, partition a space of known extent. Hence, almost any spatial index will perform decently.

Often compression of tiles is advantageous. Still, in face of very large array databases tertiary storage may be required, such as tape robots [10,11].

Query Evaluation

A tile-based storage structure suggests a tile-by-tile processing strategy. Indeed a large class of practically relevant queries can be evaluated by inspecting a tile stream, rather than loading the whole source object into main memory. Additionally, for some query types such as condensers, tile streaming can be terminated prematurely.

It turns out that array query evaluation times typically are CPU bound, except for the very rare cases where only domain subsetting and no processing is required. The reason lies in the large number of array cells to be processed per query. Preliminary results show that query parallelization by assigning tiles to different CPUs yields promising performance gains [5].

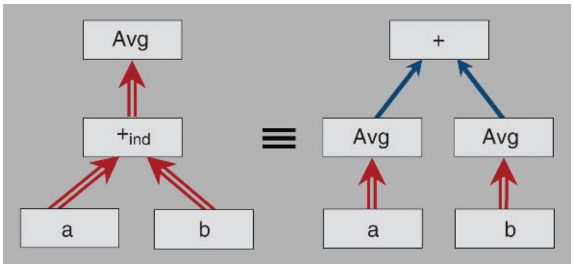
Query Optimization

Array queries lend themselves well towards *query optimization*. Currently only heuristic optimization is reasonably understood, but shows good results. Consider the query

select *avg_cells*$(a + b)$
from a, b

Figure 3 shows an equivalence rule which can speed up evaluation: substituting the left-hand side array expression with the right-hand side yields

select *avg_cells*(a) + *avg_cells*(b)
from a, b



Raster Data Management and Multi-Dimensional Arrays. Figure 3. Algebraic equivalence rule "*avg_cells* (*m1* + *m2*) ≡ *avg_cells*(*m1*) + *avg_cells*(*m2*)."

This query involves two array traversals and two (negligible) scalar operations. Bottom line, three array traversals have been reduced to two this way.

Key Applications

In many cases where some phenomenon is sampled or simulated, the result is a rasterized data set. Given the high volumes and multi-faceted retrieval arising, there is a remarkably wide application domain for array databases. Briefly, it can be summarized as sensor, image, and statistics data in the widest sense. Exemplarily geo and life sciences will be inspected next.

Earth Sciences

By far, the most important and visible application domain for large-scale array information systems is geospatial raster data, with high public visibility through novel, user-friendly interaction techniques like the Virtual Globes of NASA WorldWind (worldwind.arc.nasa.gov) and GoogleEarth (earth.google.com).

The Open GeoSpatial Consortium (OGC, www.opengeospatial.org), in collaboration with ISO, OASIS, W3C and others, provides standards for open, interoperable geo Web service interfaces. OGC develops a family of modular geo service standards, of which the Web Coverage Service (WCS) suite is particularly relevant for raster data (there also called coverage data). It offers basic subsetting services, scaling, and reprojection. This static request type is augmented with a coverage processing language in the Web Coverage Processing Service (WCPS) Implementation Specification [3].

A typical WCPS request computes the Normalized Difference Vegetation Index (NDVI), which computes a measure of the degree of vegetation for each pixel. For a multi-spectral satellite image *s* with near-infrared channel *nir* and red channel *red*, the NDVI is defined as:

NDVI(s) = ($s.nir$ - $s.red$)/($s.nir$ + $s.red$)

The result is a real value between -1 and +1; the closer it is to +1, the higher is the likelihood for vegetation. For some server object *LandsatScene* (note that WCPS lists single objects and does not operate on sets like database languages) an 8-bit greyscale NDVI image is specified as follows:

for *s* in (*LandsatScene*)
return
encode((char) (255 * (*s.nir* - *s.red*)/(*s.nir* + *s.red*) + 1), "TIFF")

Life Science

Human Brain Imaging In human brain imaging, the research goal is to understand the relations between brain structure and its function. In experiments, human subjects have to perform some mental task while activity parameters such as brain temperature, electrical activity, and oxygen consumption are measured by PET or fMRI CAT scans. The resulting 3-D x/y/z data cubes have a resolution of 1 mm and an overall volume in the Megabyte size. In a computationally expensive warping operation, the brain images get normalized against some chosen standard brain so that organs always sit at known voxel coordinates (Fig. 4 left). In the end, a brain image set as large as possible is desired to achieve high statistical significance.

Traditionally, feature search is the only property through standard SQL on structured and semi-structured data; specialized tools can perform search on single images or small sets thereof. With array databases, thousands of experiments each with large image sets can be searched by brain feature. The brain organs can be registered as voxel masks (Fig. 4 right). The query types arising mainly perform standard statistics per brain.

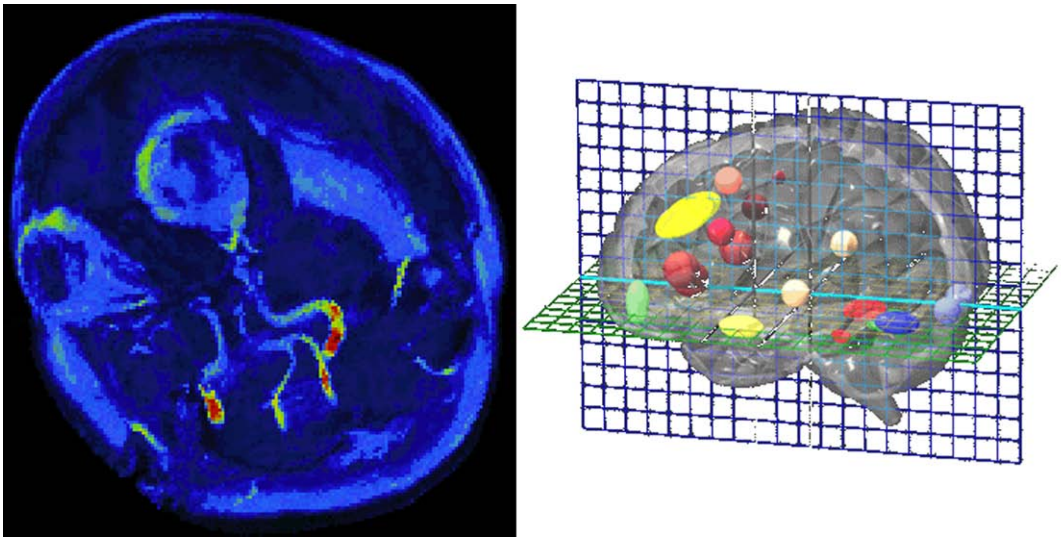
Example: "A parasagittal view of all scans containing critical Hippocampus activations, TIFF-coded." Positional parameter $1 denotes the slicing position, $2 the intensity threshold value, $3 the confidence, as chosen by a user through some browser interface.

select tiff(*ht*[$1, *:*, *:*])
from *HeadTomograms* as *ht*, *Hippocampus* as *mask*
where count_cells(*ht* > $2 and *mask*)/count_cells(*mask*) > $3

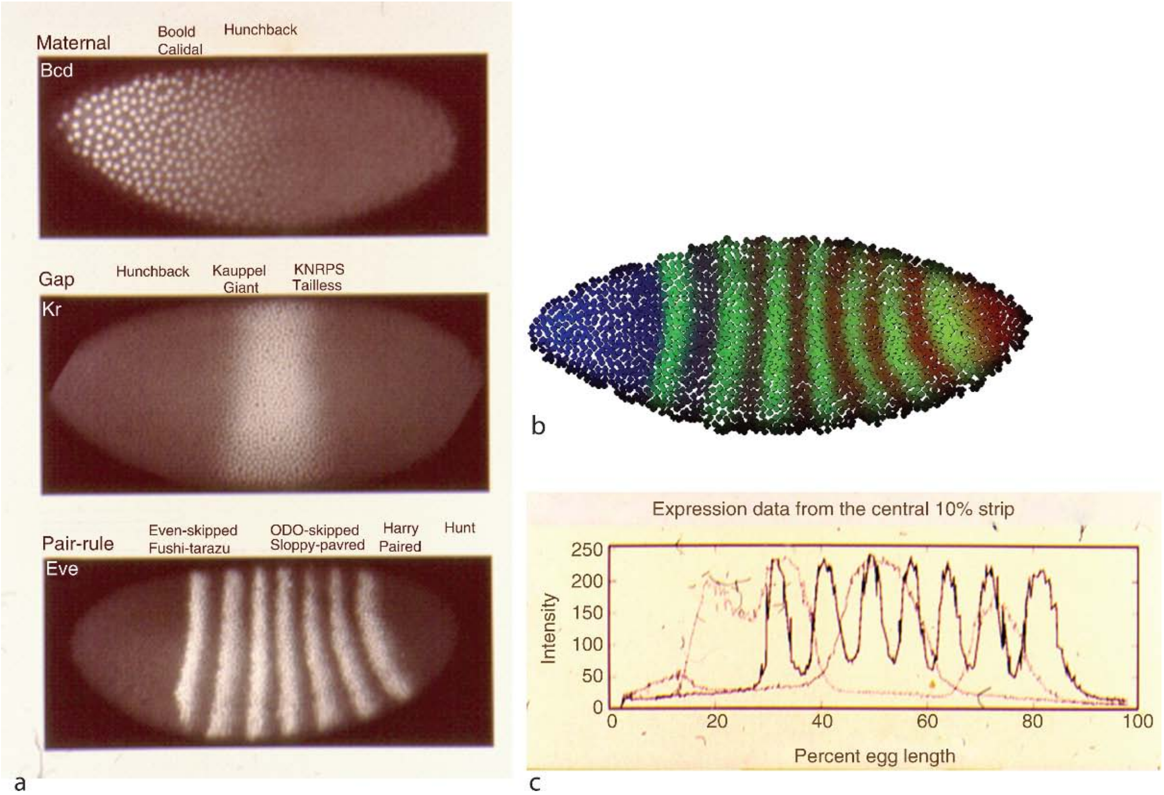
Gene Expression Analysis Gene expression is the activity of reading out genes for reproduction in a living body. The research goal in gene expression analysis is to understand how and when genes express to become manifest in the phenotype. One step towards this is to understand the spatio-temporal expression patterns in a well-known research object, the fruit fly Drosophila melanogaster.

The staining process delivers activity patterns (Fig. 5 left). It is common to combine three gene images into one by randomly assigning them to the red, green, and blue channel, resp. (Fig. 5 top right). Traditionally, diagrams like Fig. 5 bottom right are constructed.

Array queries allow searching the resulting 4-D x/y/z/t activity cube and generate, among others, the views researchers are used to. A gene expression database might contain 4-D objects where the first dimension lists the fruitfly genes in some chosen order and the other three spatial dimensions allow addressing into the fruitfly body. Then, a query can slice these 4-D cubes and recombine these slices into the aforementioned RGB overlay.



Raster Data Management and Multi-Dimensional Arrays. Figure 4. Human brain activation map (left) and brain organ masks (right).



Raster Data Management and Multi-Dimensional Arrays. Figure 5. Gene activity maps for a fruitfly embryo (left), overlay into RGB (top right), slice aggregating activity over the 10% central strip of a snapshot (bottom right).

Example: “Genes $1, $2, and $3 at time $4, as RGB images.”

```
select jpeg({1c,0c,0c}*e[$1, *:*, *:*, $4]
+{0c,1c,0c}*e[$2, *:*, *:*, $4]
+{0c,0c,1c}*e[$3, *:*, *:*, $4])
from EmbryoImages as e
where oid(e) = 193537
```

Future Directions

Array databases are in their infancy, with many inviting research avenues. On a conceptual level, a unified algebraic array theory seems promising which is to unify (dense) image handling with (sparse) statistical and OLAP handling. Advanced query optimization deserves further investigation, with topics like cost-based optimization and complex array addressing schemes (such as filter kernels). Physical tuning parameters are not yet fully understood in their effects and interplay. Finally, further experience in as many applications as possible is desirable for a better understanding of the relevant query patterns. The ultimate goal is to establish arrays as first-class database citizens.

Cross-references

- ▶ Biomedical Image Data Types and Processing
- ▶ Digital Elevation Models
- ▶ Discrete Wavelet Transform and Wavelet Synopses
- ▶ Geographic Information System
- ▶ Image Database
- ▶ Index Creation and File Structures
- ▶ On-Line Analytical Processing
- ▶ Query Languages and Evaluation Techniques for Biological SEQUENCE Data
- ▶ Query Load Balancing in Parallel Database Systems
- ▶ Query Optimization
- ▶ Query Processing and Optimization in Object Relational Databases
- ▶ Query Processing in Data Warehouses

- ▶ Query Rewriting
- ▶ Query Translation
- ▶ Range Query
- ▶ R-Tree (and Family)
- ▶ Scientific Databases
- ▶ Semantic Modeling for Geographic Information Systems
- ▶ Spatial and Spatio-Temporal Data Models and Languages
- ▶ Spatial Data Analysis
- ▶ Spatial Data Mining
- ▶ Spatial Data Types
- ▶ Spatial Indexing Techniques
- ▶ Spatio-Temporal Data Warehouses

Recommended Reading

1. Baumann P. On the management of multidimensional discrete data. VLDB J., 4(3):401–444, 1994. Special Issue on Spatial Database Systems.
2. Baumann P. A database array algebra for spatio-temporal data and beyond. In Proc. Fourth Int. Workshop on Next Generation Information Technologies and Systems, 1999, pp. 76–93.
3. Baumann P. and Chulkov G. Web coverage processing service implementation specification. Jacobs University Technical Report #9, July 2007.
4. Chock M., Cardenas A., and Klinger A. Database structure and manipulation capabilities of a picture database management system (PICDMS). IEEE Trans. Pattern Analy. Machine Intell., 6(4):484–492, 1984.
5. Hahn K., Reiner B, Höfling G., and Baumann P. Parallel query support for multidimensional data: inter-object parallelism. In Proc. 13th Int. Conf. on Database and Expert Syst. Appl. 2002, pp. 820–830.
6. Libkin L., Machlin R., and Wong L. A query language for multidimensional arrays: design, implementation and optimization techniques. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 228–239.
7. Machlin R. Index-based multidimensional array queries: safety and equivalence. In Proc. 27th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2008, pp. 175–184.
8. Marathe A. and Salem K. A language for manipulating arrays. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 46–55.
9. Mennis J., Viger R., and Tomlin C.D. Cubic map algebra functions for spatio-temporal analysis. Cartogr. Geogr. Inf. Sci., 32(1):17–32, 2005.
10. Reiner B. and Hahn K. Hierarchical storage support and management for large-scale multidimensional array database management systems. In Proc. 13th Int. Conf. Database and Expert Syst. Appl., 2002, pp. 689–700.
11. Sarawagi S. and Stonebraker M. Efficient organization of large multidimensional arrays. In Proc. 10th Int. Conf. on Data Engineering, 1994, pp. 328–336.

Raster Databases

▶ Raster Data Management and Multi-Dimensional Arrays

RBAC

▶ Role Based Access Control

RBAC Standard

▶ ANSI/INCITS RBAC Standard

RDF

▶ Resource Description Framework

Reactive Rules

▶ ECA Rules

Read/Write Model

▶ Transaction Models – The Read/Write Approach

Real and Synthetic Test Datasets

Thomas Brinkhoff
Institute for Applied Photogrammetry and
Geoinformatics (IAPG), Oldenburg, Germany

Synonyms

Spatio-temporal data generator; Spatio-temporal benchmarking

Definition

In the area of mobile and ubiquitous data management, real and synthetic test datasets are used for experimental investigations of performance and robustness.

Typical applications are the examination of access methods for spatio-temporal databases and the simulation of mobility for location-based services. Besides synthetic and real datasets, combinations of these two types of data are often used that integrate a predefined infrastructure.

Historical Background

Beginning in the mid 1990s, the development of algorithms and data structures for spatio-temporal data took place of the research in the field of pure (geo-) spatial applications. Previous spatial test datasets were not sufficient for investigating the performance and robustness of those algorithms and data structures. Consequently, first data generators for spatio-temporal test datasets were published in the end of the 1990s.

Foundations

Comprehensible performance evaluations are an important requirement in the field of mobile and ubiquitous data management. This demand covers the preparation and use of well-defined test datasets and benchmarks enabling the systematic and comprehensible evaluation and comparison of algorithms and data structures in this area.

In experimental investigations, synthetic data following some statistical distributions as well as data from real-world applications are used as test datasets. The use of *synthetic datasets* allows testing the behavior of an algorithm or of a data structure under exactly specified conditions or in extreme situations. In addition, for testing the scalability, synthetic data sets are often suitable. However, it is difficult to assess the performance of real applications by employing synthetic data. The use of *real datasets* tries to solve this problem. In this case, the selection of the data is crucial. For non-experts it is often difficult to decide whether a dataset reflects a "realistic" situation or not. Furthermore, real datasets are typically connected with a special type of application. For example, a dataset recording cars driving within a city may have completely different properties than the traffic in rural areas or the movement of vehicles in battlefields.

In the area of mobile and ubiquitous data management, *infrastructure-based dataset generators* have become popular. These generators compute moving objects that are restricted by some infrastructure like a network or prohibited areas. The infrastructure may be a real-world dataset or completely artificial. The number, distribution, speed and other properties of the moving objects are influenced by the infrastructure as well as by parameters specified by the user of the generator. Dataset produced by infrastructure-based dataset generators are a compromise between real and synthetic datasets in respect of realism on the one hand and of controllability on the other hand.

Synthetic Datasets

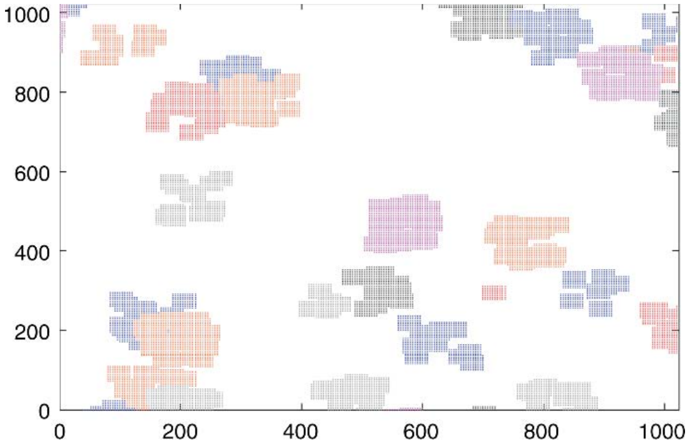
A prominent example for a program providing synthetic spatio-temporal test datasets is the *GSTD (Generate SpatioTemporal Data)* algorithm by Theodorides, Silva and Nascimento [8]. The basic idea of this algorithm is to start with a distribution of point or rectangular objects, e.g., a uniform, a Gaussian or a skewed data distribution. These objects are modified by computing positional and shape changes using parameterized random functions. If a maximum time stamp is exceeded, an object will become invalid. If an object leaves the predefined spatial data space, different approaches can be applied: the position remains unchanged, the position is adjusted to fit into the data space, or the object re-enters the data space at the opposite edge of the data space. In order to create more realistic movements, a later version of the GSTD algorithm considers rectangles for simulating an infrastructure: each moving object has to be outside of these rectangles.

The *Oporto generator* by Saglio and Moreira [7] is designed for a specific scenario: fishing at sea. The generator supports different object types, which allow modeling different behavior. In order to generate smooth movements, objects of one type may be attracted (e.g., fish by plankton) or repulsed by objects of other types (e.g., ships by stormy areas).

G-TERD (Generator for Time-evolving Regional Data) by Tzouramanis, Vassilakopoulos and Manolopoulos [9] generates two-dimensional raster data (see Fig. 1). The user can control the behavior of the generator by defining parameters and statistical models. The parameters influence the color, the maximum speed, size and rotation of the moving regions. Other moving or static objects may have impact on the speed and on the direction of a moving object.

Real Datasets

There exist many spatio-temporal datasets in the WWW [6]. Very often GPS positions of cars or other vehicles like buses, trains or bikes are recorded. An example is the INFATI dataset from the Aalborg University [3]. It represents a collection of spatio-temporal



Real and Synthetic Test Datasets. Figure 1. Raster regions generated by G-TERD.

data that was collected for a project about intelligent speed adaptation. The dataset contains the GPS tracks of about two dozen cars equipped with GPS receivers and logging equipment.

Other popular objects are animals equipped with active satellite tags, e.g., whales, sharks and turtles. The recordings of weather phenomena like hurricanes and the orbits of satellites can also be used as spatio-temporal test datasets.

The section "Data Sets" of this entry gives a small overview on real datasets that are provided for download.

Datasets from Infrastructure-Based Generators

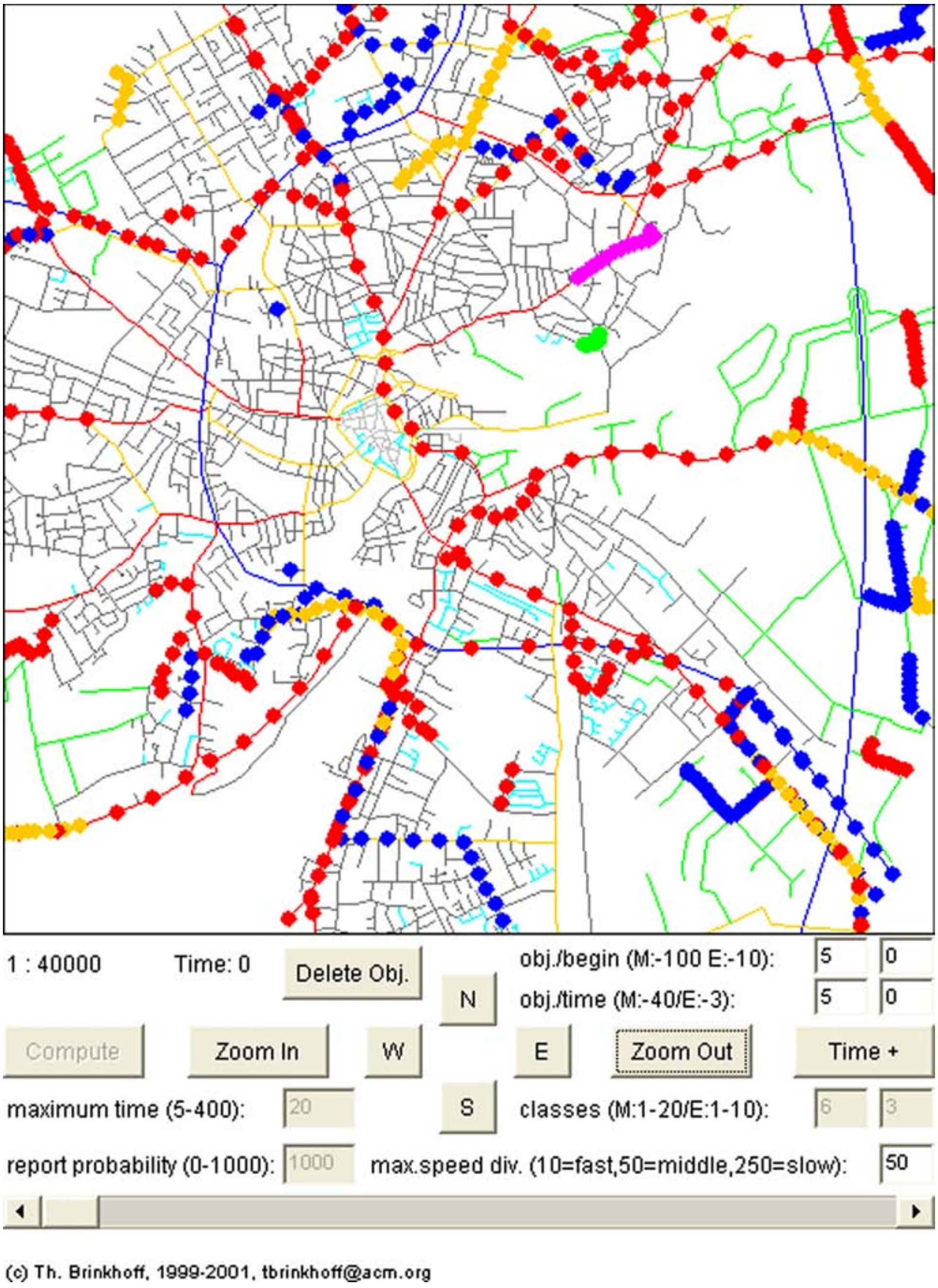
The *Network-based Generator* by Brinkhoff [1] is based on the observation that the motion of objects is often restricted by a network. Examples are streets, railways, air corridors or waterways. Therefore, the generator computes moving objects according to a network that is provided by a file or by a spatial database (see Fig. 2). For each edge of the network, a speed limit and a maximum capacity can be defined. If the number of objects traversing an edge at the same time exceeds the specified capacity, the speed limit on this edge may decrease. In addition, each moving object has a (maximum) speed. The computations of the number of new objects per time stamp, of the start location, of the length of a new route and of the location of the destination are done by time-dependent Java functions that can be overloaded by the user of the generator. This concept allows modeling daily commuting and rush hours. The route of a moving object is computed at the time of its creation. However, the fastest path may change over the time by the motion of other objects and of other influences. Therefore, the re-computation of a route is triggered by events depending on the travel time and on the deviation between the current speed and the expected speed on an edge.

The *City Simulator* by Kaufman, Myllymaki, and Jackson [4] is a scalable, three-dimensional model city that enables the creation of dynamic spatial data simulating the motion of up to one million moving objects. The data space of the city is divided into different types of places that influence the motion of the moving objects: roads, intersections, lawns and buildings are such places that define together a city plan (see Fig. 3). Each building consists of an individual number of floors for modeling the third dimension.

SUMO (Simulation of Urban Mobility) [5] developed by Institute of Transport Research at the German Aerospace Center and the Center for Applied Informatics (ZAIK) is open-source software for traffic simulation. Its objective is to support the traffic research community with a common platform for testing and comparing models of vehicle behavior, traffic light optimization, routing etc. Therefore, the car movement model of SUMO is much more evaluated than the models of the other generators.

Key Applications

The primary field of application of the presented test datasets is the evaluation of spatio-temporal databases, but also in other fields the datasets are used. According to Citeseer and the ACM Portal, applications cover (among others) the evaluation of spatio-temporal data structures, the analysis of spatio-temporal queries



1 : 40000 Time: 0

obj./begin (M:-100 E:-10): 5 0

obj./time (M:-40/E:-3): 5 0

maximum time (5-400): 20

classes (M:1-20/E:1-10): 6 3

report probability (0-1000): 1000

max.speed div. (10=fast,50=middle,250=slow): 50

(c) Th. Brinkhoff, 1999-2001, tbrinkhoff@acm.org

Real and Synthetic Test Datasets. Figure 2. Demo of the network-based generator.

and of mobility patterns, the simulation of wireless environments by mobile agents, the design of (web) server architectures for moving objects, and the test of car-to-car communication.

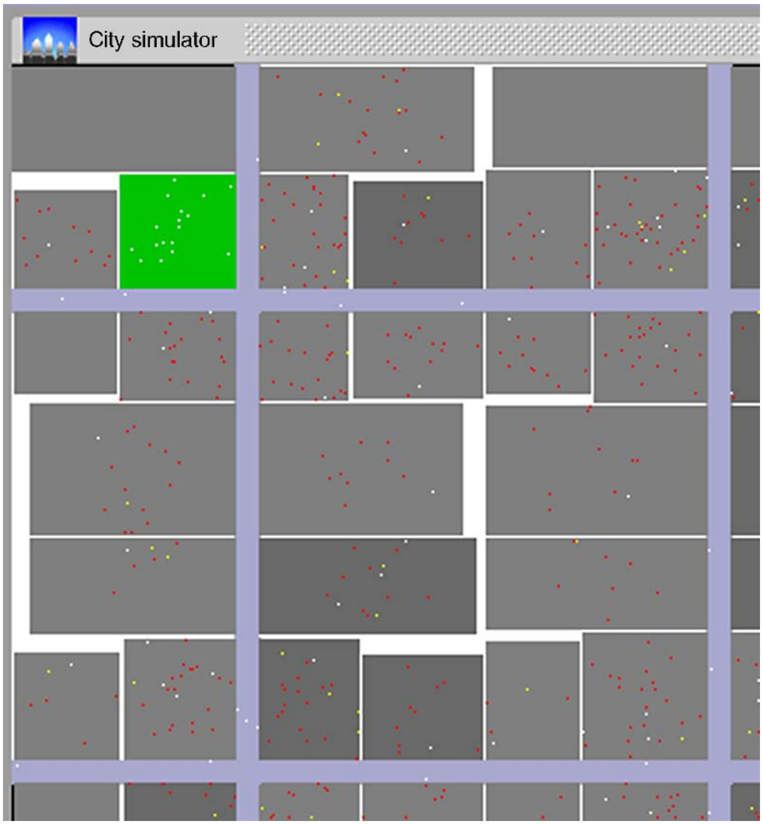
Future Directions

It can be expected that spatio-temporal test datasets will be more often used for evaluations of sensor networks, peer-to-peer communication and positioning techniques.

Data Sets

INFATI Dataset: http://arxiv.org/abs/cs.DB/0410001

Pfoser, D. Where can I get spatio-temporal data? http://dke.cti.gr/people/pfoser/data.html



Real and Synthetic Test Datasets. Figure 3. Visualization of a city plan by the city simulator.

Sea Turtle Migration-Tracking:
http://www.cccturtle.org/satellitetracking.php
Unisys Weather – Hurricane/Tropical Data: http://weather.unisys.com/hurricane/index.html
WhaleNet: http://whale.wheelock.edu/whalenet-stuff/stop_cover.html

URL to code

GSTD: see http://www.cs.ualberta.ca/~mn/ for further notices
G-TERD: http://delab.csd.auth.gr/stdbs/g-terd.html
Network-based generator: http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/
Oporto: http://www.inf.enst.fr/~saglio/etudes/oporto/
SUMO: http://sumo.sourceforge.net/

Cross-references

- ▶ Geographic Information System
- ▶ Indexing Historical Spatio-temporal Data
- ▶ Indexing of the Current and Near-Future Positions of Moving Objects
- ▶ Location-based services (LBS)
- ▶ Moving Objects Databases and Tracking
- ▶ Road Networks
- ▶ Spatial and Spatio-temporal Data Models and Languages
- ▶ Spatial Network Databases
- ▶ Spatio-Temporal Trajectories

Recommended Reading

1. Brinkhoff T. A framework for generating network-based moving objects. GeoInformatica, 6(2):153–180, 2002.
2. Jensen C.S (ed.). Special issue on infrastructure for research in spatio-temporal query processing. Bull. Tech. Comm. Data Eng., 26(2):51–55, 2003.
3. Jensen C.S., Lahrmann H., Pakalnis S., and Runge J. The INFATI Data, 2004. Available at: http://oldwww.cs.aau.dk/research/DP/tdb/TimeCenter/TimeCenterPublications/TR-79.pdf
4. Kaufman J., Myllymaki J., and Jackson J. City Simulator. IBM alphaWorks emerging technologies, 2001. Available at: https://secure.alphaworks.ibm.com/aw.nsf/techs/citysimulator
5. Krajzewicz D., Hertkorn G., Rössel C., and Wagner P. SUMO (Simulation of Urban MObility): an open-source traffic simulation. In Proc. of the Fourth Middle East Symp. on Simulation and Modelling, 2002, pp. 183–187.

6. Nascimento M.A., Pfoser D., and Theodoridis Y. Synthetic and real spatiotemporal datasets. Q. Bull. IEEE TC on Data Engineering, 26(2):26–32, 2003.
7. Saglio J.-M. and Oporto M.J. A realistic scenario generator for moving objects. GeoInformatica, 5(1):71–93, 2001.
8. Theodoridis Y., Silva J.R.O., and Nascimento M.A. On the generation of spatiotemporal datasets. In Proc. Int. Symp. on Large Spatial Databases, 1999, pp. 147–164.
9. Tzouramanis T., Vassilakopoulos M., and Manolopoulos Y. On the generation of time-evolving regional data. GeoInformatica, 6(3): 207–231, 2002.

Real-Time Transaction Processing

Jörgen Hansson[1], Ming Xiong[2]
[1]Carnegie Mellon University, Pittsburgh, PA, USA
[2]Bell Labs, Murray Hill, NJ, USA

Synonyms

Time-constrained transaction management

Definition

Real-time transaction processing focuses on (i) enforcing time constraints of transactions, i.e., meet time constraints on invocation and completion, and (ii) ensuring temporal consistency of data, i.e., data should be valid/ fresh at the time of usage.

The successful integration of time-cognizant behavior and transaction processing into a database system is generally referred to as a real-time database system (RTDB).

Historical Background

The area of real-time transaction processing has emerged from the need for real-time systems, which often are safety-critical, to handle large amounts of data in a systematic fashion, and the increasing expectation of non-critical applications that have used conventional databases but are now needed to deal with "soft" real-time data applications, e.g., multimedia. Real-time systems have traditionally managed data in an ad hoc manner, i.e., system developers have stored and manipulated data in regular data structures resident in the application code. This approach does not scale well as applications increase in complexity and in their needs for managing large amounts of data. Conventional databases, however, are considered inadequate for handling real-time requirements. Conventional databases impose a throughput-centric design, e.g., maximizing average throughput, and is thus not concerned about the specific outcome of a transaction (in fact, transactions are considered equally important). Furthermore, conventional databases are general-purpose transaction processing system design to be satisfy the needs of a multitude of non-real-time applications. In contrast, a predominant part of the real-time systems are very resource limited, normally these systems have a couple of orders of magnitude less resources, e.g., primary memory, and are not using hard disks. Thus, system functionality need to be tailored to application-specific needs to accommodate a minimum footprint and overcome different architectural assumptions and data requirements.

Note that the term real-time has many connotations in industry and in the general literature it is often used interchangeably to denote that something is fast. This differs from the notion used here, where real-time is synonymous to the notion of predictability. The notion of real-time indicates that the correctness of a result depends not only on the logical result of its computation but also on the time at which the result is derived. Thus, the system correctness depends both on the functional and temporal behavior of the system execution. The temporal behavior is tightly connected to the time constraints associated with transactions and data, e.g., deadlines. This requires that a transaction management system needs to enforce that the timeliness of transactions and data, in turn requiring that scheduling and concurrency control algorithms are time-cognizant.

Examples of real-time applications are control systems, multimedia, to air-traffic control, and as evident, these applications differ in their real-time requirements, complexity, and the type of data they manage. In order to maintain external/internal correctness, real-time systems have to respond to input stimuli and by an actuator, i.e., produce an output result/action within a finite and sufficiently small time bound. These systems must feature predictability, i.e., the ability to show that the system meets the specified requirements under various conditions the system is expected work under [11].

Several research platforms have developed, e.g., ARTS-RTDB & BeeHive (both from University of Virginia, USA), COMET (Linköping University, Sweden), DeeDS (University of Skövde, Sweden), REACH (Technical University of Darmstadt, Germany), Rodain

(University of Helsinki, Finland), and STRIP (Stanford University, USA).

Foundations

Real-transaction processing systems operate under and are benchmarked against different performance goals, correctness criteria, and assumptions applications than throughput-centric systems. The performance metrics adopted for real-time transaction processing system reflect that real-time transactions have time constraints on the execution, and the validity of the data. These requirements impose constraints on the system, which is reflected in the performance metrics used for measuring the timeliness of the system, in addition to conventional metrics for measuring consistency. Typical performance metrics, out of which several originate from the area of real-time systems, include deadline miss ratio of transactions, tardiness/lateness of transactions, cost/effects due transactions missing their deadlines, data temporal consistency (external and logical). The requirements of timeliness can exceed that of consistency and isolation, i.e., it might be more important to deliver a result of adequate precision/quality in time than delivering an exact result late. This implies that correctness of a result can be traded for timeliness by relaxing consistency (referring to ACID properties).

Transaction Model

Real-time transactions are typically characterized along the dimensions of the temporal scope, criticality, and transaction type.

The temporal scope of a transaction is typically described by an arrival time a_i, a release time r_i, a (worst case) execution time e_i, and a deadline d_i, where the following conditions hold (the times are expressed as absolute variables): $a_i \leq r_i < d_i$, $r_i + e_i \leq d_i$. A system is, thus, considered schedulable if it can be shown that all transactions can meet their deadlines.

Real-time transactions can be categorized given the criticality and the stringency of meeting their deadlines and the notions of hard, firm, and soft deadlines are generally used. A hard deadline is critical and, thus, the transaction must always complete within the deadline. Missing a hard deadline will have severe or even cataclysmal consequences. In contrast, firm and soft deadlines are used to for transactions when it system can tolerate occasional time constraint violations. The distinction between them is in the utility of completing a task after its deadline; a soft deadline indicates there is often value of completing the task albeit late. A firm deadline indicates that there is no value for late completion and tardy transactions, i.e., tasks running beyond their deadline, can thus be aborted to minimize resource usage.

Transactions may have precedence constraints that put constraints on the relative order of execution among transactions.

The temporal consistency of a data object has two parts: *absolute consistency*, which reflects the state of an external environment and how that state is reflected in the database, and *relative consistency*, which concerns the consistency among data elements used to derive other data. Thus, absolute consistency (a.k.a. external consistency) is necessary to ensure that a system's view of the external environment (e.g., the controlled system) is consistent with the actual state of the environment. Relative consistency (a.k.a. logical consistency) ensures that only valid data is used to derive new data. To express temporal consistency, a real-time data object o_j is annotated with an absolute validity interval avi_j denoting the time length and a timestamp ts_j when the data object was last updated/sampled. The data object is considered valid in the time interval $[\mathrm{ts}_j, \mathrm{ts}_j + \mathrm{avi}_j]$.

To define the notion of a relative validity interval *rvi*, a *relative consistency set R* is introduced for each derived data object, which contains the set of data objects used for its derivation. Furthermore, each such set is has a relative validity interval denoted R_{rvi}. A data object o_j is temporally consistent if the following two conditions hold: (i) $(t + ts_j) \leq avi_j$ (absolute consistency) and (ii) (assume R is the set for o_j) for all $\mathrm{o}_k \in R | \mathrm{ts}_j - \mathrm{ts}_k | \leq R_{rvi}$.

Concurrency Control

A conventional non-real-time two-phase-locking scheme (2PL) is prone to priority inversion, unknown blocking times, and deadlocks, making it infeasible in a real-time context. Priority inversion occurs when a high-priority transaction is blocked due to a resource locked by a low-prioritized transaction. Thus, the high-priority transaction experiences blocking delays that can jeopardize this timeliness. Several 2PL variants have been developed to overcome these deficiencies, and the most well-known scheme is two-phase-locking with high priority resolution (2PL-HP). Consider the case a transaction T_R is requesting a lock, which is held by another transaction T_H. In 2PL-HP, T_R will abort T_H if the priority of T_R exceeds that of T_H; otherwise T_R will wait until T_H completes. There are additional schemes that deploy more elaborate conditions to avoid that a process is aborted unnecessarily, e.g., considering the remaining execution time of T_R and the needed execution time of T_R.

In real-time optimistic concurrency control (OCC), transactions execute in three stages: read, validation, and write. In the read stage, transactions read and update data items freely, storing their updates into private workspaces. Note that the updates of a transaction stored in the private workspaces are installed as global copies in the write stage (i.e., after the transaction is validated). In the validation stage, a validating transaction may conflict with ongoing transactions. Depending on the transaction characteristics, there are several real-time conflict resolution mechanisms for the validation [5], i.e., a validating transaction may commit, abort, or be "put on the shelf" to wait for the conflicting transactions.

In priority-wait conflict resolution mechanism [5], which is demonstrated to have superior performance for OCC, a transaction that reaches validation and finds higher priority transactions in its conflict set is "put on the shelf", that is, it is made to wait and not allowed to commit immediately. This gives the higher priority transactions a chance to make their deadlines first. After all conflicting higher priority transactions leave the conflict set, either due to committing or due to aborting, the on-the-shelf waiter is allowed to commit. Note that a waiting transaction might be restarted due to the commit of one of the conflicting higher priority transactions.

Distributed Real-Time Transaction Processing The distributed transaction execution model for a real-time two-phase commit protocol is presented next. A commonly adopted sub-transaction model has been presented [4] in which there is one process, called the master, which is executed at the site where the transaction is submitted, and a set of other processes, called cohorts, which execute on behalf of the transaction at the various sites that are accessed by the transaction. Cohorts are created by the master sending a STARTWORK message to the local transaction manager at that site. This message includes the work to be done at that site and is passed on to the cohort. Each cohort sends a WORKDONE message to the master after it has completed its assigned data processing work. The master initiates the commit protocol (only) after it has received this message from all its cohorts. Within the above framework, a transaction may execute in either sequential or parallel fashion. The distinction is that cohorts in a sequential transaction execute one after another, whereas cohorts in a parallel transaction execute concurrently.

Real-Time Two-Phase Commit The master implements the classical two-phase commit protocol [3] to maintain transaction atomicity. In this protocol, the master, after receiving the WORKDONE message from all its cohorts, initiates the first phase of the commit protocol by sending PREPARE (to commit) messages in parallel to all its cohorts. Each cohort that is ready to commit first force-writes a prepare log record to its local stable storage and then sends a YES vote to the master. At this stage, the cohort has entered a prepared state wherein it cannot unilaterally commit or abort the transaction, but has to wait for the final decision from the master. On the other hand, each cohort that decides to abort force writes an abort log record and sends a NO vote to the master. Since a NO vote acts like a veto, the cohort is permitted to unilaterally abort the transaction without waiting for the decision from the master.

After the master receives votes from all its cohorts, the second phase of the protocol is initiated. If all the votes are YES, the master moves to a committing state by force-writing a commit log record and sending COMMIT messages to all its cohorts. Each cohort, upon receiving the COMMIT message, moves to the committing state, force-writes a COMMIT log record, and sends an ACK message to the master. On the other hand, if the master receives one or more NO votes, it moves to the aborting state by force-writing an abort log record and sends ABORT messages to those cohorts that are in the prepared state. These cohorts, after receiving the ABORT message, move to the aborting state, force-write an abort log record and send an ACK message to the master. Finally, the master, after receiving ACKs from all the prepared cohorts, writes an end log record and then "forgets" the transaction (by removing from virtual memory all information associated with the transaction).

A real-time two-phase commit protocol such as PROMPT (Permits Reading Of Modified Prepared data for Timeliness) [6], works differently from traditional two-phase commit protocol in that transactions requesting data items held by other transactions in the prepared state are allowed to access this data. That is, prepared cohorts lend their uncommitted data to concurrently executing transactions (without releasing the update locks) in the optimistic belief that this data will be committed. If the lender is aborted later, the borrower is also aborted since it has utilized dirty data. On the other hand, if the borrowing cohort completes its local data processing before the lending cohort has received its global decision, the borrower is "put on the shelf", that is, it is made to wait until either the lender receives its global decision or its own deadline expires, whichever happens earlier. In this case, the borrower can only commit if the lender commits.

In contrast to centralized databases where transactions that validate successfully *always* commit, a distributed transaction that is successfully locally validated might be aborted later because it fails during global validation. This can lead to "wasteful" aborts of transactions–a transaction that is locally validated may abort other transactions in this process. If this transaction is itself later aborted during global validation, it means that all the aborts it caused during local validation were unnecessary.

Key Applications

The number of real-time applications that handle real-time data is large. Thus, a range of applications that manage data with temporal constraints is given in the following.

- *Air-traffic control systems.* This system is used to monitor air-traffic around an airport, which requires continuous monitoring of aircraft positions and weather data, as well as (non-real-time) data of different aircraft types.
- *Control system.* A specific case is an engine control system, which approximately uses readings of twenty sensors, and these sensor values are fundamental in the computation of 400 other variables, where the validity of the derived data is a function of the sensed value. Derived data is used control the mixture of fuel and air in the ignition, as well as diagnostics.
- Wireless networking systems for retrieval of subscriber information such as service subscription and device location stored in Home Location Register (HLR) as well as billing information in the case of prepaid phone calls. For example, the network communication protocols have various timers for call delivery that may cause connection failure if information is not retrieved before a transaction deadline.

Future Directions

The increasing number of applications that handle large amounts of real-time data calls for a strong support from the underlying transaction processing system to satisfy timeliness and consistency requirements. The development of time-cognizant concurrency control and scheduling algorithms has provided a foundation for real-time transaction processing systems. There are several challenging areas where progress would be conducive to enforce the predictability, and below a few examples are provided

- The underlying assumption which most real-time scheduling approaches build upon is the knowledge of the worst-case execution times of the transactions, either a priori to the system starts its execution or they are made available upon their arrival to the system. The dynamic nature of transaction workloads can cause the system to experience transient overloads. Techniques that are shown to effectively manage the uncertainty of execution times or can measure tight and not overly conservative worst case execution times would increase schedulability and resource utilization, as well as techniques for resolving and minimizing the effects of transient overloads.
- *Extended transaction models supporting relaxed ACID properties.* Enforcement of ACID in real-time applications has shown to be costly, jeopardize timeliness, and does not utilize the potential parallelism that can be exploited by relaxation of the ACID properties. This has resulted in the development of, e.g., epsilon-serializability and the notion of data similarity. There is a need for additional application-centric consistency that would capture the tolerance of the applications and the presence of multiple models coexisting in parallel.
- *Transaction scheduling for guaranteeing data temporal consistency.* There is need to maintain coherency between the state of the environment and data used in applications such as control systems.

For example, in order to react to abnormal situations in time, it is necessary to monitor the environment continuously. Such requirements pose a great challenge for maintaining the freshness of data while scheduling transactions to meet their deadlines.

- The notion of data precision and data confidence are becoming increasingly important in real-time applications, i.e., data is annotated with additional attributes to represent the confidence in the data value, (i.e., the level of trust in how the data was derived) and the precision/accuracy of the data. It is unclear how these two notions can be effectively used as decision parameters for concurrency control and schedulability.

Cross-references

▶ ACID Properties
▶ Temporal Consistency

Recommended Reading

1. Abbott R. and Garcia-Molina H. Scheduling real-time transactions: a performance evaluation. In Proc. 14th Int. Conf. on Very Large Data Bases, 1988.
2. Bestavros A. and Fay-Wolfe V. Real-Time Database and Information Systems – Research Advances. Kluwer Academic Publishers, MA, USA, 1997.
3. Carey M. and Livny M. Conflict detection tradeoffs for replicated data. ACM Trans. Database Syst., 16:703–746, 1991.
4. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, CA, USA, 1992.
5. Haritsa J., Carey M., and Livny M. Data access scheduling in firm real-time database systems. J. Real-Time Syst., 4: 203–241, 1992.
6. Haritsa J., Ramamritham K., and Gupta R. The PROMPT real-time commit protocol. IEEE Trans. Parall. Distr. Syst., 11(2):160–181, 2000.
7. Ramamritham K. Real-time databases. Int. J. Distr. Parall. Databases, 1, (2) 1993.
8. Ramamritham K., Son S.H., and Cingiser D.L. Real-Time databases and data services. Real-Time Syst. J., 28:179–215, 2004.
9. Son S.H. Advances in Real-Time Systems. Prentice-Hall, NJ, USA, 1995.
10. Soparkar N., Korth H.F., and Silberschatz A. Time-Constrained Transaction Management – Real-Time Constraints in Database Transaction Systems. Kluwer Academic Publishers, MA, USA, 1996.
11. Stankovic J. and Ramamritham K. What is Predictability for Real-Time Systems? Real-Time Syst., 2(4):247–254, 1990.
12. Xiong M., Han S., and Lam K.-Y. A Deferrable Scheduling Algorithm for Real-Time Transactions Maintaining Data Freshness. In Proc. 26th IEEE Real-Time Systems Symp., 2005, pp. 27–37.

Real-World Time

▶ Valid Time

Reasoning with Qualitative Temporal Constraints

▶ Qualitative Temporal Reasoning

Recall

Ethan Zhang[1,2], Yi Zhang[1]
[1]University of California, Santa Cruz, Santa Cruz, CA, USA
[2]Yahoo! Inc., Santa Clara, CA, USA

Definition

Recall measures the coverage of the relevant documents of an information retrieval (IR) system. It is the fraction of all relevant documents that are retrieved. Consider a test document collection and an information need Q. Let R be the set of documents in the collection that are relevant to Q. Assume an IR system processes the information need Q and retrieves a document set A. Let $|R|$ and $|A|$ be the numbers of documents in R and A, respectively. Let $|R \cap A|$ denote the number of documents that are in both R and A. The *recall* of the IR system for Q is defined as $R = |R \cap A| / |R|$.

Key Points

Precision and recall are the most frequently used and basic retrieval performance measures. Many other standard performance metrics are based on the two concepts.

Cross-references

▶ Eleven Point Precision-recall Curve
▶ F-Measure
▶ Precision
▶ Standard Effectiveness Measures

Receiver Operating Characteristic

Pang-Ning Tan
Michigan State University, East Lansing, MI, USA

Synonyms

Operating characteristic; Relative operating characteristic; ROC

Definition

Receiver operating characteristic (ROC) analysis is a graphical approach for analyzing the performance of a classifier. It uses a pair of statistics – true positive rate and false positive rate – to characterize a classifier's performance. The statistics are plotted on a two-dimensional graph, with false positive rate on the x-axis and true positive rate on the y-axis. The resulting plot can be used to compare the relative performance of different classifiers and to determine whether a classifier performs better than random guessing.

Historical Background

ROC analysis was originally developed in signal detection theory to deal with the problem of discriminating known signals from a random noise background [11]. It was first applied to the radar detection problem to quantify how effective targets such as enemy aircrafts can be identified according to their radar signatures. In the 1960s, ROC analysis was applied to experimental psychology and psychophysics [6]. The approach has subsequently found its application in a variety of areas including radiology, epidemiology, finance, weather forecasting, and social sciences. In machine learning, the benefits of using ROC analysis for evaluating and comparing the performance of classifiers was first demonstrated by Spackman [14]. The approach was brought to the attention of the data mining community by Provost and Fawcett [12], who proposed the idea of using a convex hull of ROC curves to compare multiple classifiers in imprecise and changing environments.

Foundations

Predictive accuracy has traditionally been used as the primary evaluation measure for classifiers. However, its limitation is well-documented, particularly for data sets with skewed class distributions [12]. ROC analysis provides an alternative way for measuring performance by examining the trade-off between the successful detection of positive examples and the misclassification of negative examples. The approach was originally developed for binary classification problems, where each example is assigned to either a positive or a negative class. When applying the classifier to a given example, four possible outcomes may arise: (i) true positive (TP), when a positive example is classified correctly, (ii) true negative (TN), when a negative example is classified correctly, (iii) false positive (FP), when a negative example is misclassified as positive, and (iv) false negative (FN), when a positive example is misclassified as negative. These outcomes can be tabulated in a 2 × 2 table known as the confusion matrix, as shown in Table 1.

An ROC graph is constructed by examining the true positive rate and false positive rate of a classifier. The true positive rate (TPR), also known as hit rate or sensitivity, corresponds to the proportion of positive examples that are correctly labeled by the classifier, whereas the false positive rate (FPR), also known as false alarm rate, corresponds to the proportion of negative examples that are incorrectly labeled. Mathematically, these statistics are computed from a given confusion matrix as follows:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{1}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{2}$$

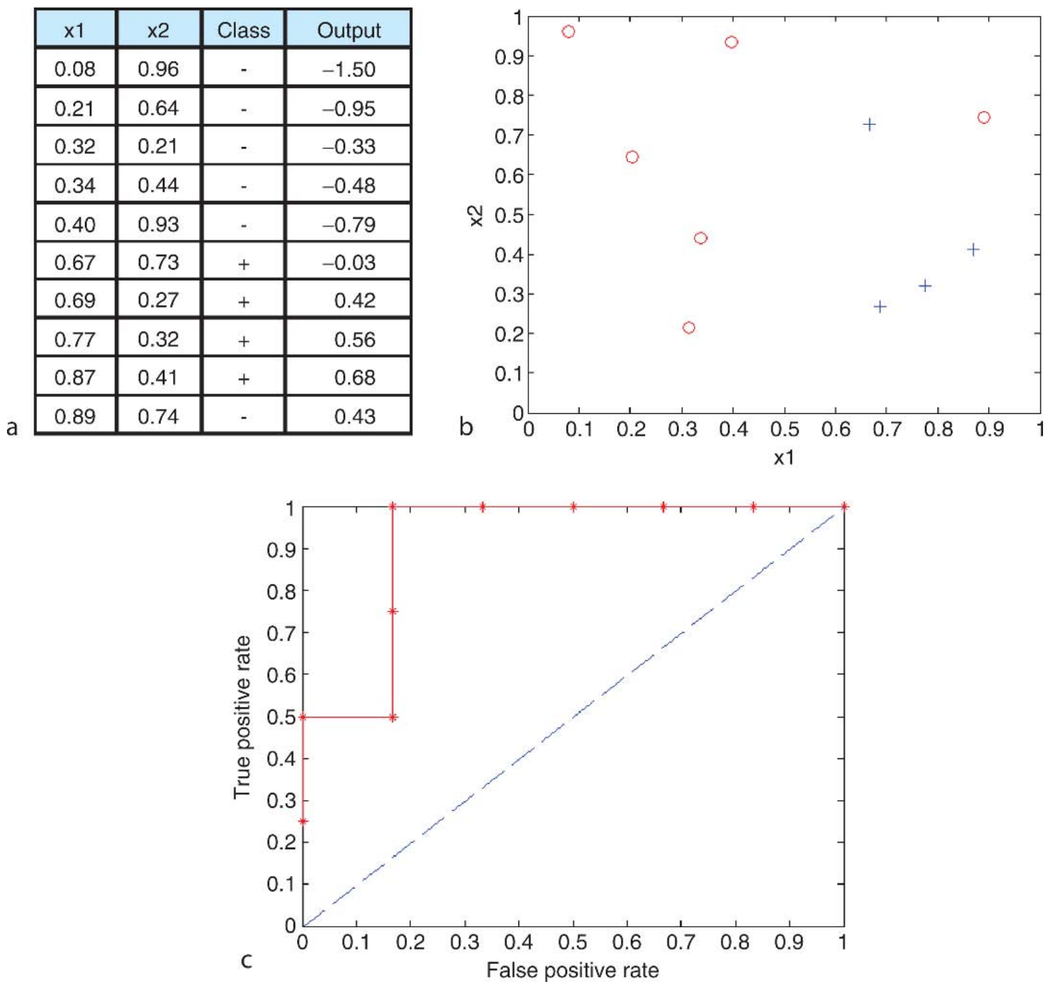
The number of points in an ROC graph depends on the type of output produced by the classifier. A classifier that produces a discrete-valued output is mapped to a single point in the ROC graph because there is only one confusion matrix. Other classifiers such as naïve Bayes and neural networks can produce numeric-valued outputs to indicate the degree to which an example belongs to the positive class. A threshold must be specified to determine the class membership – if the classifier's output exceeds the threshold, the example is assigned to the positive class. Each threshold setting leads to a different point in the ROC graph. By varying the threshold, a piecewise linear curve, known as the ROC curve, is formed. For example, the solid line in Fig. 1c shows the ROC curve obtained by varying the threshold on classification outputs produced by a neural network classifier.

There are several critical points in the diagram that are of practical significance. The critical point (FPR = 0, TPR = 0) corresponds to an extremely conservative classifier, i.e., one that assigns every example to the negative class. In contrast, the critical point (FPR = 1, TPR = 1) corresponds to a classifier that liberally declares every example to be positive. Since the ideal classifier corresponds to the critical point (FPR = 0, TPR = 1), points closer to the upper-left corner of the ROC graph are generally better classifiers. A random classifier, on the other hand, produces points that reside along the diagonal line connecting the bottom-left to the upper-right corner of the ROC graph, as shown by the dashed line in Fig. 1c. For example, a classifier that randomly assigns one-fourth of the examples to the positive class has TPR = 0.25 and FPR = 0.25.

Receiver Operating Characteristic. Table 1. Confusion matrix for a 2-class problem

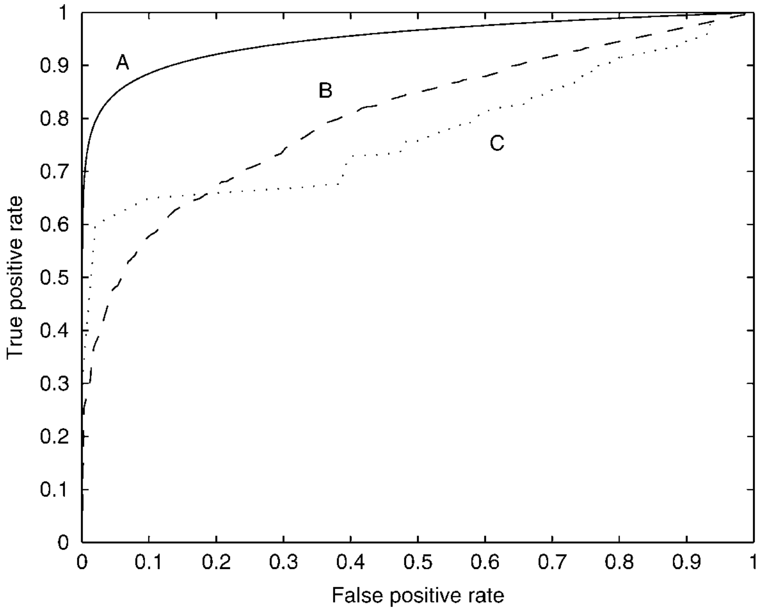
		Predicted Class	
		+	−
Actual Class	+	TP	FN
	−	FP	TN

An ROC curve X dominates another ROC curve Y if X lies above and to the left of Y. The more dominant the ROC curve, the better the classifier is. For example, Fig. 2 shows that classifier A is better than classifiers B and C because it has a more dominant ROC curve. In practice, however, one seldom finds an ROC curve that completely dominates other ROC curves. Instead, one would find different ranges of FPR values in which one classifier is better than another. For example,



x1	x2	Class	Output
0.08	0.96	-	–1.50
0.21	0.64	-	–0.95
0.32	0.21	-	–0.33
0.34	0.44	-	–0.48
0.40	0.93	-	–0.79
0.67	0.73	+	–0.03
0.69	0.27	+	0.42
0.77	0.32	+	0.56
0.87	0.41	+	0.68
0.89	0.74	-	0.43

Receiver Operating Characteristic. Figure 1. ROC curve for a two-dimensional data set classified using a single layer neural network.



Receiver Operating Characteristic. Figure 2. Performance comparison of classifiers using ROC curve.

in Fig. 2, classifier C is better than B when FPR below 0.2 or above 0.95. Therefore, Provost and Fawcett [12] introduced the ROC convex hull (ROCCH) method to combine the ROC curves from a set of classifiers to obtain the most dominant ROC curve. The convex hull can be used to identify ranges of FPR values in which a classifier is potentially optimal.

Although an ROC curve provides a visual display of a classifier's performance, it is often useful to summarize the curve into a single metric to estimate the overall classifier's performance. By analyzing the statistical properties of the metric [10], this provides a quantitative way to determine whether the observed difference in performance of two classifiers is statistically significant. Examples of ROC-derived metrics include area under ROC curve (AUC), slope intercept index, and the ROC break-even point. AUC is also equivalent to the Wilcoxon-Mann-Whitney statistic [7], thus allowing us to compute its statistical properties such as standard error and confidence interval.

In addition to ROC curves, there are alternative ways to visualize the performance of a classifier such as precision-recall curves [2] and cost curves [3]. A precision-recall curve plots the tradeoff between precision, which is the fraction of examples classified as positive that are actually positive, against recall, which is equivalent to true positive rate. Davis and Goadrich [2] has shown that a curve that dominates in the ROC space also dominates in the precision-recall space. Nevertheless, an ROC curve may not effectively capture important differences between classifiers when applied to data sets with skewed class distributions. For example, the confusion matrices shown in Table 2 have very similar TPR and FPR values even though their precision values are very different. The ROC representation also does not commit to any particular cost function or class distribution. As a result, it does not convey information such as the misclassification costs and class probabilities for which a classifier performs better than another. To overcome this limitation, Drummond and Holte [3] proposed the idea of using cost curves to explicitly represent the cost information.

Receiver Operating Characteristic. Table 2. Confusion matrices to illustrate the difference between ROC and precision-recall curves

(a)		Predicted Class	
		+	–
Actual Class	+	35	5
	–	25	435
(b)		Predicted Class	
		+	–
Actual Class	+	35	5
	–	5	455

A cost curve plots the expected cost of a classifier against a probability cost function which is defined as follows:

$$\text{Probability Cost Function} = \frac{P(+)C(-|+)}{P(+)C(-|+) + P(-)C(+|-)} \quad (3)$$

where $P(+)$ and $P(-)$ are the prior probabilities of each class, $C(-|+)$ is the cost of misclassifying a positive example as negative, while $C(+|-)$ is the cost of misclassifying a negative example as positive.

Key Applications

ROC analysis has been successfully applied to many application domains including psychology (in the studies of perception to resolve the issue of sensory threshold) [15], radiology (to distinguish between subjective judgment and objective detectability in imaging systems) [9], and epidemiology [13]. In addition to classification problems, ROC analysis is also applicable to other ranking problems such as recommender systems, information retrieval, and anomaly detection.

Future Directions

Most of the previous work on ROC analysis are limited to binary class problems. For multi-class problems, the analysis is more complicated as the confusion matrix is no longer a simple 2 × 2 table. An obvious way to extend the approach is to generate a different ROC graph for each class. However, with this approach, the ROC analysis is no longer insensitive to the class distribution [4]. Furthermore, combining the AUC statistics from multiple ROC graphs remains an open problem.

Another research direction that has attracted considerable interests in recent years is designing classification algorithms that directly optimize the area under ROC curve, or equivalently, the Wilcoxon-Mann-Whitney statistic. For instance, Cortes and Mohri [1] showed that, under certain conditions, the objective function optimized by the RankBoost algorithm is identical to AUC. New algorithms have also been developed to incorporate AUC into decision tree induction [5] and support vector machines [8].

URL to Code

The WEKA data mining software provides codes for plotting ROC curves and cost curves (http://www.cs.waikato.ac.nz/~ml/weka/.) The software for plotting ROC convex hull is available at http://home.comcast.net/~tom.fawcett/public_html/ROCCH/index.html.

Cross-references

▶ Area Under ROC Curve
▶ Classification

Recommended Reading

1. Cortes C. and Mohri M. Auc optimization vs. error rate minimization. In Advances in Neural Inf. Proc. Syst. 16, Proc. Neural Inf. Proc. Syst., December 2003.
2. Davis J. and Goadrich M. The relationship between precision-recall and roc curves. In Proc. 23rd Int. Conf. on Machine Learning, 2006.
3. Drummond C. and Holte R.C. Explicitly representing expected cost: an alternative to roc representation. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 198–207.
4. Fawcett T. An introduction to roc analysis. Pattern Recogn. Lett., 27(8):861–874, 2006.
5. Ferri C., Flach P.A., and Hernandez-Orallo J. Learning decision trees using the area under the roc curve. In Proc. 19th Int. Conf. on Machine Learning, 2002.
6. Green D.M. and Swets J.A. (eds.). Signal Detection Theory and Psychophysics. Wiley, New York, 1966.
7. Hanley J.A. and McNeil B.J. The meaning and use of the area under a receiver operating characteristic (roc) curve. Radiology, 143(1):29–36, 1982.
8. Joachims T. A support vector method for multivariate performance measures. In Proc. 22nd Int. Conf. on Machine Learning, 2005.
9. Lusted L.B. Signal detectability and medical decision making. Science, 171, 1971.
10. McNeil B.J. and Hanley J.A. Statistical approaches to the analysis of the receiver operating characteristic (roc) curves. Med. Decis. Making, 4(2):137–150, 1984.
11. Peterson W.W., Birdsall T.G., and Fox W.C. The theory of signal detectability. IRE Trans., PGIT-4, 1954.
12. Provost F.J. and Fawcett T. Analysis and visualization of classifier performance: comparison under imprecise class and cost distributions. In Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining, 1997, pp. 43–48.
13. Sackett D.L. Clinical diagnosis and the clinical laboratory. Clin. Invest. Med., 1, 1978.
14. Spackman K.A. Signal detection theory: Valuable tools for evaluating inductive learning. In Proc. 6th Int. Workshop on Machine Learning, 1989, pp. 160–163.
15. Swets J.A. The relative operating characteristics in psychology. Science, 182, 1973.

Recodings

▶ Matrix Masking

Reconciliation-based Data Replication

▶ Optimistic Replication and Resolution

Record Extraction

▶ Column Segmentation

Record Linkage

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona, Catalonia

Synonyms

Record matching; Re-identification

Definition

Record linkage is a computational procedure for linking each record *a* in file *A* (e.g., a file masked for disclosure protection) to a record *b* in file *B* (original file). The pair (*a*, *b*) is a match if *b* turns out to be the original record corresponding to *a*.

Key Points

Record linkage techniques were created for data fusion and to increase data quality. However, they have also found an application in measuring the risk of identity disclosure in statistical disclosure control. In the SDC context, it is assumed that an intruder has an external dataset sharing some (key or outcome) attributes with the released protected dataset and containing additionally some identifier attributes (e.g., passport number, full name, etc.). The intruder is assumed to attempt to link the protected dataset with the external dataset using the shared attributes. The number of matches gives an estimation of the number of protected records whose respondent can be re-identified by the intruder. Accordingly, disclosure risk is defined as the proportion of matches among the total number of records in *A*.

There are two main types of record linkage used to measure identity disclosure in SDC: distance-based record linkage and probabilistic record linkage.

Distance-based record linkage consists of linking each record *a* in file *A* to its nearest record *b* in file *B*. Therefore, this method requires a definition of a distance function for expressing *nearness* between records. This record-level distance can be constructed from distance functions defined at the level of attributes. Construction of record-level distances requires standardizing attributes to avoid scaling problems and assigning each attribute a weight on the record-level distance. A straightforward choice is to use the Euclidean distance, but other distances can be used.

The main advantages of using distances for record linkage are simplicity for the implementer and intuitiveness for the user. Another strong point is that subjective information (about individuals or attributes) can be included in the re-identification process by properly modifying distances.

The main difficulty of distance-based record linkage consists of coming up with appropriate distances for the attributes under consideration. For one thing, the weight of each attribute must be decided and this decision is often not obvious. Choosing a suitable distance is also especially difficult in the cases of categorical attributes and of masking methods such as local recoding where the masked file contains new labels with respect to the original dataset.

Like distance-based record linkage, probabilistic record linkage aims at linking pairs of records (*a*, *b*) in datasets *A* and *B*, respectively. For each pair, an index is computed. Then, two thresholds *LT* and *NLT* in the index range are used to label the pair as linked, clerical or non-linked pair: if the index is above *LT*, the pair is linked; if it is below *NLT*, the pair is non-linked; a clerical pair is one that cannot be automatically classified as linked or non-linked and requires human inspection. When independence between attributes is assumed, the index can be computed from the following conditional probabilities for each attribute: the probability $P(1|M)$ of coincidence between the values of the attribute in two records *a* and *b* given that these records are a real match, and the probability $P(0|U)$ of non-coincidence between the values of the attribute given that *a* and *b* are a real unmatch.

To use probabilistic record linkage in an effective way, one needs to set the thresholds *LT* and *NLT* and estimate the conditional probabilities $P(1|M)$ and $P(0|U)$ used in the computation of the indices. In plain words, thresholds are computed from: (i) the probability $P(LP|U)$ of linking a pair that is an unmatched pair (a *false positive* or *false linkage*) and (ii) the probability $P(NP|M)$ of not linking a pair that is a match (a *false negative* or *false unlinkage*). Conditional probabilities $P(1|M)$ and $P(0|U)$ are usually estimated using the EM algorithm.

Cross-references

- ► Disclosure Risk
- ► Inference Control in Statistical Databases
- ► Microdata
- ► Record Matching

Recommended Reading

1. Fellegi I.P. and Sunter A.B. A theory for record linkage. J. Am. Stat. Assoc., 64(328):1183–1210, 1969.
2. Torra V. and Domingo-Ferrer J. Record linkage methods for multidatabase data mining. In Information Fusion in Data Mining, V. Torra (ed.). Germany, Springer, Berlin. 2003, pp. 101–132.

Record Matching

Arvind Arasu[1], Josep Domingo-Ferrer[2]
[1]Microsoft Research, Redmond, WA, USA
[2]The Public University of Tarragona, Tarragona, Spain

Synonyms

Record linkage; Merge-purge; Entity resolution; Data deduplication; Duplicate detection; Instance identification; Name matching

Definition

Record matching is the problem of identifying whether two records in a database refer to the same real-world entity. For example, in Fig. 1, the customer record *A*1 in Table *A* and record *B*1 in Table *B* probably refer to the same customer, and should therefore be matched. (The example in Fig. 1 was adapted from an example in [21].) As Fig. 1 suggests, the same entity can be encoded in different ways in a database; this phenomenon is fairly common and occurs due to a variety of natural reasons such as different formatting conventions, abbreviations, and typographic errors. Record matching is often studied in the following setting: Given two relations *A* and *B*, identify all pairs of matching records, one from each relation. For the two tables in Fig. 1, a reasonable output might be the pairs (*A*1,*B*1) and (*A*2,*B*2). In some settings of the record matching problem, there is a constraint that each record of table *A* be matched with at most one record of table *B*. This asymmetric setting is typically used when records in table *B* are "clean" and those of table *A*, "dirty." The record matching problem is closely related to the *deduplication* problem. The focus of record matching is to identify pairs of matching records while that of deduplication is to partition records so that records in the same partition refer to the same entity. In practice, the output produced by record matching is inaccurate; it is not an equivalence relation and does not correspond to a natural partitioning. The above distinction between record matching and deduplication is often ignored and the two terms are used synonymously.

Historical Background

Record matching has a rich history dating back to the work by Newcombe and others [16] in 1959. Fellegi and Sunter [10] formalize the intuition of Newcombe and others. Specifically, they cast the record matching problem as a classification problem that can be stated as follows: Given a vector of *similarity scores* between attribute values for a pair of records, classify the pair as a *match* or a *nonmatch*. For a given attribute, a similarity score indicates how similar the two records are on that attribute. A simple similarity measure assigns a score 1 if the records agree on the attribute and 0 otherwise. More sophisticated similarity measures are discussed subsequently. Fellegi and Sunter [10] use a naive Bayes classifier, but subsequent work has considered other kinds of classifiers such as decision trees [7] and SVMs [3]. The classifiers are typically trained using learning examples comprising of a set of record pairs, each labeled as a match or a non-match [3,7,10]. One of the problems with this approach is that learning examples required to train an accurate classifier are hard to generate, since they should not be either obvious matches or obvious non-matches. Sarawagi and Bhamidipaty [17] address this problem and propose an approach based on active learning to interactively identify useful learning examples. Jaro [13] and Winkler [20] propose an alternate approach where the classifier is learned in an unsupervised setting without learning examples using a variant of EM (expectation maximization) algorithm.

A large class of work on record matching has focused on more sophisticated measures of similarity between attribute values of records. As mentioned above, these form the basis for classifying record pairs as a match or a nonmatch. A variety of classic similarity functions such as edit distance and variants [5] and cosine similarity with

a

Id	Name	Address	Age
A1	J H Smith	16 Main St	17
A2	Javeir Marteenez	49 Apple cross Road	33
A3	Gillian Jones	645 Reading Ave	24

b

Id	Name	Address	Age
B1	John H Smith	16 Main Street	17
B2	Javier Martinez	49 E Apple cross Road	33
B3	Jilliam Brown	123 Norcross Blvd	43

Record Matching. Figure 1. Record matching example.

tf-idf (term frequency-inverse document frequency) weights [8] have been used for record matching. Jaro [12] proposes a more domain specific similarity measure designed for people names. Sarawagi and Bhamidipaty [17] propose using a weighted linear combination of simple similarity functions such as those mentioned above, and present techniques for learning the weights using training examples. Bilenko and Mooney [3] present a generalization of edit distance whose parameters can be learnt from a large text corpus. Arasu and others [1] present a framework for *programmable similarity*, where a similarity function can be programmed to be sensitive to synonymous words or phrases such as Robert and Bob or US and United States.

Another body of work on record matching has focused on efficiency issues. For large inputs, it is impractical to exhaustively consider all pairs of records and check if they are a match or not. Hernandez and Stolfo [11] present the *sorted neighborhood approach*, which linearly orders all the records based on a carefully selected key and considers only pairs of records that are close to each other in the linear ordering. McCallum and others [15] present an approach based on *canopies*, which are overlapping clusters of the input records. Only pairs of records within a cluster are checked for a match. Chaudhuri and others [6] identify *set-similarity join* as a useful primitive for large scale record matching.

Foundations

String Similarity

Most record matching approaches are based on the observation that two matching records have similar values for their attributes. For example, the records A2 and B2 have similar names and similar addresses and the same age, and are therefore likely to be matches. The similarity between two values is typically determined using a *similarity function* that takes two values and produces as output a number that quantifies the similarity of the values. String similarity functions that quantify the similarity between two strings are particularly relevant for record matching since many attributes in record matching are textual in nature. One of the earliest and well-known string similarity measure is *edit distance* or *Levenshtein distance*. The edit distance between two strings $s1$ and $s2$ is defined as the smallest number of edit operations required to produce $s2$ from $s1$, where an edit operation is an insertion, deletion, or substitution at the character level. For example, the edit distance between Martinez and Marteenez is 2 since one can derive the second string from the first using one substitution and one insertion. Edit distance is often a poor fit for record matching since two strings representing different entities can have small edit distance (e.g., 148th Ave NE and 147th Ave NE) and two strings representing the same entity, a large edit distance (e.g., 148th Ave NE and 148th Ave Northeast).

An alternate approach that works well for some domains is to treat strings as a bag of words (or *tokens*) and use a token-based similarity function such as *cosine* or *jaccard* (defined below). For a string s, let $Tokens(s)$ denote the set of tokens in s. For a token t, let $w(t)$ denote the *weight* of token t. The weight of a token represents its "importance" for the purposes of computing similarity. The jaccard similarity of two strings s_1 and s_2 is defined as:

$$\frac{|\ Tokens(s_1) \cap Tokens(s_2)\ |}{|\ Tokens(s_1) \cup Tokens(s_2)\ |}$$

For a set S, $|S|$ denotes the weighted cardinality of S, i.e., the sum of weights of the tokens in B. The cosine similarity between two strings s_1 and s_2 is defined as:

$$\frac{||\ Tokens(s_1) \cap Tokens(s_2)\ ||_2}{||\ Tokens(s_1)\ ||_2 \cdot ||\ Tokens(s_2)\ ||_2}$$

For a set S, $||S||_2$ is defined as $\sqrt{\sum_{t \in S} w(t)^2}$. Token-based similarity functions differ from edit distance in two aspects: First, they ignore the ordering of tokens; for example, `148th Ave NE` represents the same bag of tokens as `NE 148th Ave`. This feature is desirable in some domains. For example, the list of authors appears before the title string in some citations and after the title in other citations. An order-sensitive similarity function would produce a low similarity score for two citation strings referring to the same publication but with different author-title orderings. Second, token-based similarity functions are not sensitive to intra-token edits. Two tokens are considered different even if they are textually very similar. As mentioned earlier, this feature is useful in domains such as addresses where, for example, two street names can differ by a single character. Token-based similarity functions enable weighting of tokens to capture their relative importance. A commonly used weighting scheme is the idf-based one, where the weight $w(t)$ of a token t is defined to be $log(N/N_w)$, where N is the number of records in a reference table and N_w denotes the number of records in the reference table containing the token t. With idf-based weighting, the similarity of the pair (`Applecross Rd`, `Applecross Road`) would be higher than the similarity of the pair (`Applecross Rd`, `Maltby Rd`) although both pairs have one token that is common to the two strings in the pair. This happens since the idf-weights of the rarer words `Applecross` and `Maltby` are higher than the weights for the common words `Road` and `Rd`. A related class of similarity functions is obtained by tokenizing strings to their character-level n-grams instead of words. For example, the set of 2-grams of the string `Applecross` is: `{Ap, pp, pl, le, ec, cr, ro, os, ss}`. These similarity functions share some characteristics with edit distance since they can capture intra-word edits, but they are not as order sensitive as edit distance.

A variety of other domain specific similarity functions have been used for record matching. These are covered in depth in the survey by Elmagarmid and others [9] and the tutorial by Koudas and others [14].

Record Matching

Let R and S denote the input tables of record matching. The record matching problem can be viewed as a binary classification problem, where a given pair of records (r, s), $r \in R$ and $s \in S$, has to be classified either as a *match* or a *nonmatch*. A slight variant, not discussed here, is to consider a third category *possible match* consisting of hard-to-classify pairs that require manual inspection. One common approach for record matching is to train a standard binary classifier using learning examples consisting of a set of record pairs prelabeled as a match or nonmatch. The original Fellegi and Sunter approach uses a naive Bayes classifier and is discussed below. Other kinds of classifiers such as SVMs [3] and decision trees [7] have also been used.

Fix a record pair $(r, s) \in R \times S$. Define a *similarity vector* $\overline{x} = [x_1,..., x_n]$, where each x_i denotes the similarity between r and s on some attribute A. The similarity can be computed using one of the functions discussed earlier (for string attributes). It can also be a simple binary value based on equality, i.e., $x_i = 1$ if r and s agree on the attribute and 0, otherwise. Let M denote the event that the pair (r,s) is a match and U the event that it is a nonmatch. Using Bayes rule, one gets

$$Pr(M|\overline{x}) = \frac{Pr(\overline{x}|M)Pr(M)}{Pr(\overline{x})}$$

$$Pr(U|\overline{x}) = \frac{Pr(\overline{x}|U)Pr(U)}{Pr(\overline{x})}$$

Therefore, $Pr(M|\overline{x}) \geq Pr(U|\overline{x})$ whenever,

$$\frac{Pr(\overline{x}|M)}{Pr(\overline{x}|U)} \geq \frac{Pr(U)}{Pr(M)}$$

The expression $\ell(\overline{x}) = Pr(\overline{x}|M)/Pr(\overline{x}|U)$ is called the *likelihood function*. The construction of the naive Bayes classifier assumes that x_i and x_j, $i \neq j$, are conditionally independent given M or U, implying:

$$Pr(\overline{x}|M) = \prod_{i=1}^{n} p(x_i|M)$$

$$Pr(\overline{x}|U) = \prod_{i=1}^{n} p(x_i|U)$$

The probabilities $p(x_i|M)$ and $p(x_i|U)$ can be estimated using the learning examples, which can be used to estimate $Pr(\overline{x}|M)$ and $Pr(\overline{x}|U)$ using the expressions above. It is harder to estimate $Pr(U)$ and $Pr(M)$. A simple approach is to empirically pick a threshold

T and classify the pair (r, s) as a match if $\ell(\overline{x}) \geq T$ and a nonmatch otherwise.

A simple classifier that has performance advantages over more sophisticated machine-learning (ML) classifiers is the *threshold-based* classifier. Here, the record pair (r, s) is classified as a match if the similarity vector $\overline{x}$ satisfies:

$$(x_1 \geq T_1) \wedge (x_2 \geq T_2) \wedge \cdots \wedge (x_n \geq T_n)$$

where $T_1,...,T_n$ are thresholds that can be learned using labeled examples or set manually based on domain expertise. Record matching based on a threshold-based classifier can be performed efficiently using the string similarity join primitive discussed below. Chaudhuri and others [4] show that the record matching accuracy of union of several threshold-based classifiers is comparable to that of state-of-art ML classifiers.

A related approach, sometimes called *distance-based* [19] record matching, is to define a distance measure between two records by suitably combining the similarity (distance) scores of the attributes of the two records. A pair (r, s) is considered a match if the distance between r and s is smaller than a given threshold value. This approach is also easily extended to the asymmetric version of record matching where each record in R is constrained to match at most one record in S: For each record r in R, the record in S with the smallest distance to r is selected as a match.

More details of the other approaches used for record matching can be found in the survey by Elmagarmid et al. [9].

Performance

For large input tables R and S, it is impractical to consider every pair of records $(r, s) \in R \times S$ and classify them as a match or a nonmatch. Therefore, an important challenge in record matching is to identify the matching pairs without exhaustively considering every pair in $R \times S$. An important primitive for efficient record matching is *string similarity join*: Given two tables R and S, identify all pairs of records $(r, s) \in R \times S$ such that the string similarity of $r.A$ and $s.B$ is above some specified threshold, where the string similarity is measured using one of the similarity functions mentioned earlier. A string similarity join can be used to heuristically identify promising matching candidates, which can then be subjected to more complex classification. The string similarity join primitive can also be used to perform record matching based on the threshold-based classifier discussed above [4].

Since many of the string similarity measures such as jaccard and cosine are set-based, a more foundational primitive is *set-similarity join* [6]. A set-similarity join conceptually takes two collections of sets U and V as input and outputs all pairs of sets $(u, v) \in U \times V$ having high set-similarity. String-similarity joins can be evaluated using set-similarity joins as a primitive even for many non-set based similarity functions such as edit distance. A simple algorithm for set-similarity join involves building an inverted index over the collection V. The inverted index efficiently retrieves for any given element e all sets in V containing e. For each set u in U, the inverted index is used to retrieve all sets v in V that share one or more elements with u; the pair (u, v) is then produced as output if its satisfies the set-similarity join condition. More efficient algorithms for set-similarity joins are presented in [2,18].

Key Applications

The primary application of record matching is data cleaning. The presence of duplicate records in a database adversely affects the quality of the database and its utility for analysis and mining. Another related application of record matching is data integration. The same real-world entity is often represented differently in other databases being integrated, and record matching is necessary to evolve a common representation for the entity. Another application of record matching is to measure the risk of identity disclosure in statistical disclosure control (SDC) [19]. In the SDC context, a dataset A (for public release) is produced from a source dataset B by masking values for identity protection. The goal of this step is to make it hard for an adversary to link a record in A to the record in B from which it was generated. The number of records in A that can be matched to records in B using the record matching techniques discussed above gives an estimate for disclosure risk of the released dataset A. Distance-based record matching is often used for this purpose.

Cross-references

- ▶ Column segmentation
- ▶ Constraint-Driven Database Repair
- ▶ Data Cleaning
- ▶ Data Deduplication
- ▶ Deduplication in Data Cleaning
- ▶ Record Linkage

Recommended Reading

1. Arasu A., Chaudhuri S., and Kaushik R. Transformation-based framework for record matching. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 40–49.
2. Arasu A., Ganti V., and Kaushik R. Efficient exact set-similarity joins. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 918–929.
3. Bilenko M. and Mooney R.J. Adaptive duplicate detection using learnable string similarity measures. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 39–48.
4. Chaudhuri S., Chen B.C., Ganti V., and Kaushik R. Example-driven design of efficient record matching queries. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 327–338.
5. Chaudhuri S., Ganjam K., Ganti V., and Motwani R. Robust and efficient fuzzy match for online data cleaning. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 313–324.
6. Chaudhuri S., Ganti V., and Kaushik R. A primitive operator for similarity joins in data cleaning. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
7. Cochinwala M., Kurien V., Lalk G., and Shasha D. Efficient data reconciliation. Inf. Sci., 137(1–4):1–15, 2001.
8. Cohen W.W. Data integration using similarity joins and a word-based information representation language. ACM Trans. Inform. Syst., 18(3):288–321, 2000.
9. Elmagarmid A.K., Ipeirotis P.G., and Verykios V.S. Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng., 19 (1):1–16, 2007.
10. Felligi I.P. and Sunter A.B. A theory for record linkage. J. Am. Stat. Soc., 64(328):1183–1210, 1969.
11. Hernandez M. and Stolfo S. The merge/purge problem for large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 127–138.
12. Jaro M.A. Unimatch: A Record Linkage System: User's Manual. Tech. rep., US Bureau of the Census, Washington DC, 1976.
13. Jaro M.A. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. J. Am. Stat. Assoc., 84(406):414–420, 1989.
14. Koudas N., Sarawagi S., and Srivastava D. Record linkage: similarity measures and algorithms. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 802–803.
15. McCallum A., Nigam K., and Ungar L.H. Efficient clustering of high-dimensional data sets with application to reference matching. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 169–178.
16. Newcombe H.B., Kennedy J.M., Axford S.J., and James A.P. Automatic linkage of vital records. Science, 130:954–959, 1959.
17. Sarawagi S. and Bhamidipaty A. Interactive deduplication using active learning. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 269–278.
18. Sarawagi S. and Kirpal A. Efficient set joins on similarity predicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 743–754.
19. Torra V. and Domingo-Ferrer J. Record Linkage methods for multidatabase data mining. In Information Fusion in Data Mining, V. Torra (ed.), Springer, 2003, pp. 101–132.
20. Winkler W. Improved Decision Rules in the Felligi-Sunter Model of Record Linkage. Tech. rep., Statistical Research Division, US Bureau of the Census, Washington DC, 1993.
21. Winkler W. The state of record linkage and current research problems. Tech. rep., Statistical Research Division, US Bureau of the Census, Washington DC, 1999.

Records Management

► Enterprise Content Management

Recovery Guarantees

► Application Recovery

Recovery in Distributed Commit Protocols

► Distributed Recovery

Recovery in Distributed Database Systems

► Distributed Recovery

Recovery in Replicated Database Systems

► Distributed Recovery

Recovery Manager

► Logging/Recovery Subsystem

Recursive Query Evaluation

► Query Processing in Deductive Databases

Recursive View Maintenance

► Maintenance of Recursive Views

Redo

► Crash Recovery
► Logging and Recovery

Redundant Arrays of Independent Disks

Kenichi Wada
Hitachi Limited, Tokyo, Japan

Synonyms

Array; Disk array; Storage array; RAID

Definition

A set of disks from one or more commonly accessible disk subsystems is combined with a body of control software in which part of the physical storage capacity is used to store redundant information about user data stored on the remainder of the storage capacity. The redundant information enables regeneration of user data in a storage emergency in which a disk in the array or an access path fails.

Key Points

The term *RAID* was adopted from the 1988 SIGMOD paper "A Case for Redundant Arrays of Inexpensive Disks (RAID)." In the paper, RAID refers to a group of storage schemes that divide and replicate data among multiple disks to provide better performance, cost and power consumption rate with reasonable availability comparing with conventional Single Large Expensive Disk (SLED) used for Mainframe. Currently, the term of "independent" is usually used rather than "inexpensive" because SLED becomes obsolete.

A number of standard schemes have evolved which are referred to as *levels*. Originally, five RAID levels were conceived, but many more variations have evolved. Currently, there are several sublevels as well as many non-standard levels.

The two key concepts in RAID are striping, and error correction using parity. The striping of RAID means dividing user data into fixed length blocks (striping units). The error correction using parity conceals one disk failure in an array of disks from the host computer and user data lost by one disk failure is regenerated using the parity.

Original five RAID levels are as follows:

1. Level 1: Mirrored disk. Two or more identical copies of data are maintained on separate disks.
2. Level 2: Using Hamming Code and striping with small striping unit (each read or write spread across all disks in an array). During read or write, on the fly ECC is adopted using Hamming Code in order to conceal some number of disk failures in an array.
3. Level 3: Using parity and striping with small striping unit (each read or write spread across all disks in an array). If one disk fails in an array, data on the failed disk are regenerated using corresponding parity and data on other disks in the array.
4. Level 4: Using parity and striping with large striping unit (small read or write can fit in one disk). One disk in an array is allocated to store the parity and user data is striped across remaining disks.
5. Level 5: Using parity and striping with large striping unit (small read or write can fit in one disk). Parity and user data is striped across all disks in an array.

In addition to these 5 levels, Level 0 is often used to represent arrays using striping without data redundancy in which fixed-length sequences of virtual disk data addresses are mapped to sequences of member disk addresses in a regular rotating pattern.

Cross-references

► Disk

Recommended Reading

1. Patterson D., Gibson G., and Katz R. A case for redundant arrays of inexpensive disks (RAID). In Proc. ACM SIGMOD Conf. on Management of Data, 1988.

Reference

► Citation

Reference Collections

▶ Archiving Experimental Data

Reference Knowledge

Chintan Patel, Chunhua Weng
Columbia University, New York, NY, USA

Definition

Reference knowledge is the knowledge about a particular part of the world in a way that is independent from specific objectives, through a theory of the domain [2]. Different knowledge bases reuse or extend subsets of the reference knowledge for application specific tasks.

Key Points

Developing knowledge bases is a laborious process involving domain experts, knowledge engineers and computer scientists. To minimize this effort, often the core theory and concepts of a given domain are represented in a reference knowledgebase that are reused or extended by other application specific knowledge bases. Consider for example, in the biomedical domain, the Foundational Model of Anatomy [3] is considered a reference knowledge base for representing anatomy terms that can be used for representing the mouse anatomy.

One of the important advantages of using reference knowledge bases is that they provide interoperability [1] across the applications using the knowledge. The applications that heavily reuse or extend the reference knowledge achieve higher levels of interoperability.

The reference knowledge bases are generally developed without any application requirements or objectives. The reference knowledge imposes a rigid set of constraints on representing the application specific or local knowledge base. Such constraints sometimes conflict with the application or turn out to be difficult to implement or maintain, hence impeding the use of reference knowledge. Secondly, the reference knowledge bases tend to be very large [1] and creating subsets appropriate for small applications is challenging.

Recommended Reading

1. Brinkley J., Suciu D., Detwiler L., Gennari J., and Rosse C. A framework for using reference ontologies as a foundation for the semantic web. In Proc. AMIA Annual Symposium, 2006.
2. Burgun A. Desiderata for domain reference ontologies in biomedicine. J Biomed Inform, 39(3):307–313, 2003.
3. Rosse C., and Mejino J. A reference ontology for biomedical informatics: the foundational model of anatomy. J Biomed Inform. 36(6):478–500, 2003.

Reference Reconciliation

▶ Deduplication in Data Cleaning

Refinement

▶ Specialization and Generalization

Region Algebra

Matthew Young-Lai
Sybase iAnywhere, Waterloo, ON, Canada

Definition

A region algebra is a collection of operators, each of which returns a set of regions as a result and takes as arguments one or more sets of regions. A *region* of a string is a pair of natural number positions (s,e) that correspond to the substring starting at s and ending at e. A position is a count of bytes, characters, or words from the beginning of the string.

Choosing a set of operators defines a particular region algebra. Operators are chosen for efficiency as well as utility. For example, if regions correspond to structure elements such as chapters and sections in a document, then many operators for querying structure conditions are useful and can be implemented efficiently. One example of such an operator is $\mathit{containedIn}(X,Y)$ which takes two sets of regions X and Y and returns the subset of regions in X that are contained in some region of Y, i.e., $\{(s_x, e_x) \in X \mid \exists (s_y, e_y) \in Y\,(s_x \geq s_y) \wedge (e_x \leq e_y)\}$. A similar operator is $\mathit{contains}(X,Y)$ which returns the subset of regions in X that contain some region of Y, i.e., $\{(s_x, e_x) \in X \mid \exists (s_y, e_y) \in Y\,(s_x \leq s_y) \wedge (e_x \geq e_y)\}$.

Another defining characteristic of a particular region algebra is the restrictions that apply to the sets of regions.

If sets are unrestricted, then a string of length n has $\binom{n}{2}$ regions. This means that the worst case cost of evaluating a single operator cannot be linear in the length of the string. Thus, region algebras are often defined with restrictions on the nesting or overlap of regions in a set.

Historical Background

The first use of a region algebra for text search was in the PAT system [10]. The operators in PAT assume the use of a PAT array (also known as a suffix array) as the underlying implementation. This data structure can provide sets of matches for various lexical patterns (e.g., find all positions corresponding to a given substring of the text). It can also filter matches based on frequency or length conditions (e.g., return only the most frequent words matching a substring). The query language includes operators for dynamically combining matches into regions – a very flexible capability. It also provides set operators such as *union* and *difference* and structure operators such as *including*. Thus, it is a good example of the idea of using a region algebra to mix structure and content operations in text search. The system makes a distinction between sets of regions and sets of points which means that it does not completely fit the definition of a region algebra. Its operators are typed in that their arguments and return values must be either sets of regions or sets of points. This means that the language is not fully compositional. Also, region sets are not allowed to include nesting or overlapping regions which causes semantic problems. For example, when an operator such as *union* results in overlapping regions, only the start points are returned.

Burkowski describes a region algebra for combined content and structure search in text [2,1]. It treats both single words and structure elements spanning many words uniformly as regions. Thus it avoids the problems that result from distinguishing points and regions. Like PAT, region sets do not include nesting or overlapping regions. Later extensions to this work allow regions to overlap which is just as efficient [3]. An advantage of overlapping regions is that it makes the ability to dynamically define regions outside of a fixed hierarchical structure even more flexible and useful. The earlier papers by Burkowski describe simple structure operations such as *containing*, simple content operators for selecting words, and ranking operators based on inverse document frequency from traditional information retrieval. Later work explores ranking for information retrieval in more depth [6].

Jaakkola continues the pattern of imposing fewer restrictions on region sets [7,8]. Sets are allowed to nest and overlap arbitrarily, abandoning the guarantee of linear size relative to the length of the string. However, the maximum depth of nesting tends to be constant in most data, and independent of the length of the data. This is true, at least, when talking about structure in text documents where region algebras are usually applied. Therefore, allowing arbitrary sets gives even more flexibility while not abandoning efficiency in practice.

Compared to relational algebra, there has been relatively little work exploring expressiveness or optimization issues with region algebras. Many systems make an informal effort to balance expressiveness with efficiency. For many useful algebras, the operators are restricted enough that the efficiency of evaluating arbitrary compositions is obvious without need of formal proof. Some work does explore more general issues. For example, Consens and Milo examine equivalence testing and whether operators like direct inclusion can be efficiently supported [4,5]. Young-Lai and Tompa look at characterizing operators that allow the possibility of efficient evaluation [12].

Foundations

The operators of a region algebra can combine and select regions. An important question, however, is where the regions originate. The most flexible approach is to scan the string for matches at every query. These matches can then be treated as regions and further manipulated using the algebra. In principle, any type of pattern language can be used for scanning. For example, it is possible to search for simple substrings, for regular expressions, or for words (possibly with linguistic processing such as stemming or lemmatization). It is also possible to parse the string with a grammar and return structures as regions. This is useful for data such as programming language source code where a well-defined grammar exists. For a scanning operation such as parsing with a grammar, the functionality of the parser starts to overlap with what can be accomplished with the region algebra. In fact, it is possible to use appropriately defined region algebra operators to simulate the process of parsing a string of tokens with a grammar.

If a string is too long to efficiently scan at every query, then regions can be pre-computed and stored persistently. This means making choices before-hand about what queries to support, and involves a tradeoff between flexibility and space. For example, consider storing a list of regions for every unique word. It is possible to apply a stoplist of common words to save space, but then it is impossible to search for phrases involving these stop words. There is also a tradeoff between efficiency and flexibility. For example, consider stemming the input by converting all forms of a word such as "stemming," "stemmed," and "stemmer" to the root word "stem." This does not reduce the number of stored regions (ignoring for the moment the fact that it can reduce the total space requirement in some compressed representations). However, it reduces the flexibility since it is no longer possible to search for just one of the forms. On the other hand, it increases the efficiency of searching for all forms together since there is no need to combine separate lists. Overall, choosing what regions to pre-compute means predicting the types of queries that need to be supported.

Various representations are possible for persistently stored regions. The simplest is a collection of unordered lists. Given that region lists generated by scanning are ordered by position in the string, however, it makes sense to store them in that order. This can be done in a flat file which allows sequential access or binary searching. Alternatively, it can be in an index structure such as a B-tree which allows more efficient searches and updates. Ordered representations allow the possibility of particularly efficient compression using various techniques.

For a non-nesting set of regions, there is a single, unique sort order. For a set of regions with nesting there are two possible sort orders. That is, regions may ordered primarily by either *s* or by *e*. The choice has some effect on evaluation strategies for expressions that compose multiple operators. If regions are stored persistently, then a single sort order must be chosen. Of course, a physical operator can be provided to convert between the two orderings, although this is not possible in linear time and constant memory. Note that considering such an operator part of the algebra itself implies modifying the definition to use lists of regions with physical order properties rather than sets of unordered regions. Alternatively, one can make a distinction between the logical algebra that works with un-ordered sets, and the physical operators that are used for optimization and evaluation.

One way to avoid choosing between the two possible sort orders in persistent storage is to only index non-nested regions. This is less of a limitation than it might appear since it is easy to generate nested regions dynamically given two lists of endpoints. For example, it is possible to index the start and end tags for nested sections in a text in separate lists. The tags themselves have a unique order since they are not nested. A physical operator can scan and merge the two lists of tag regions, pushing the start tags onto a stack and popping the stack at every end tag. This results in a region list ordered primarily by *e*. A very similar operator can be used to dynamically generate regions that contain two words of interest. In this case, there is no natural pairing of word occurrences and the result may be as big as the cross product of the two lists. A useful solution is to define the operator so that it discards all but the shortest regions in the cross product, or equivalently, those that do not contain another nested region.

Given operands that are stored in an ordered list representation, there are many useful operators such as *contains* and *containedIn* that can be evaluated by scanning both lists and performing a type of merge join. This produces an ordered result that can then be used as an input to another operator. If an algebra consists exclusively of operators that can be evaluated with such merge strategies, then it is possible to evaluate arbitrarily composed expressions with very little query optimization effort and relatively good efficiency. A simple strategy is to evaluate the operator with the two smallest available inputs at each step until all operators in the tree are finished. Intermediate results can be buffered.

There are many ways to improve on this basic strategy. Some of them are implied by viewing the problem as an instance of relational query optimization and have not been explicitly described in the context of region algebras. For example, it is possible to consider additional physical operators such as index nested loop join. This can be used in the above strategy when joining a small operand with a larger one. Rather than scan the entire large operand, the join operator performs a binary search or index lookup within the large operand for each region in the small operand.

It is also possible to consider optimizing an entire query rather than making a local, greedy decision at each step about which operator to evaluate next. This implies the need to determine equivalences between expressions, to consider different physical evaluation plans based on these equivalences, and to choose between the plans. The choice must involve estimating the costs of physical plans which is a difficult problem in general. However, it may admit heuristic or approximate solutions that do well in practice as is the case with relational query optimization.

There are some inherent limitations of the region model. One has to do with the limited information contained in a region. Given just numeric endpoints, it is easy to tell whether two given regions overlap or nest. However, nothing is known about their relationship with other regions that may exist. For example, given two regions *a* and *b* such that *a* contains *b*, there is no way to tell if *a* directly contains *b* meaning that there is no region *c* in the system such that *a* contains *c* and *c* contains *b*. Ignoring the possibility of constraints that may allow this to be inferred, the only way to know if this is the case is to search every other region list. This is likely to be inefficient.

By definition, operators in a region algebra are also excluded from referring to the content of a region. Thus, it is not possible to define an operator that takes the region endpoints and goes back to the original string to check some condition. Note that this is related to the issue of which regions have been indexed in a string. If there is a set of regions for "stem," an operator cannot check the string to see which of those regions were really the word "stemmed." However, even if there are separate region lists for "stem" and "stemmed," it is not possible to look up the value of words that follow "stem" in the text.

Of course, both of these limitations can be overcome by extending from regions to arbitrary tuples of information. To solve the direct containment problem, it is possible to store a third number *depth* with each region that indicates its nesting level. Then *b* is directly contained in *a* if it is contained in *a* and its depth is higher by one. Similarly, it is possible to add a text column to a region to store the word that follows the region and carry it with subsequent intermediate results. Essentially then, it is possible to consider extending to a full relational model. However, the region columns in such an extended model might admit the possibility of useful physical operators not generally provided in relational systems. For example, many region algebra operators can be executed with list merges but cannot be formulated as joins with equality conditions.

Another general limitation of region algebras has to do with the use of regions as a structure model. Consider a collection of region sets where each set has an associated type or label. This can represent arbitrarily nested structures in a strictly non-overlapping hierarchy. It can also represent multiple independent hierarchies over the same string. It can represent overlapping structures which goes beyond what is possible with a strict hierarchy. One thing that it cannot do is represent an arbitrary graph structure. For example, if regions are mapped to nodes in a graph, and there is an edge between any two nodes that overlap, then not all graphs are possible. For example, this can represent a chain of nodes with edges between them, but cannot add an edge from the first node in the chain to the last node without adding other edges at the same time. This limitation applies even if using regions only to encode a structure without requiring that the endpoints correspond to physical locations in a contiguous string.

Key Applications

A region algebra can be used as a query language for searching structured or semi-structured text, for searching structured or semi-structured data encoded in a text format such as SGML or XML, or for information retrieval. Alternatively, it can be used as an underlying model for query optimization or execution for some other query language.

As a query language, a region algebra can also serve as a component of a larger text processing task. One example is structure recognition where features such as composability and a loose structure model give advantages over alternatives such as grammars [11]. Other examples include constraint definition, outlier finding, and editing by example [9].

Cross-references

- ▶ Information Retrieval
- ▶ Relational Calculus
- ▶ Semi-Structured Query Languages
- ▶ XML
- ▶ XML Indexing

Recommended Reading

1. Burkowski F.J. Retrieval activities in a database consisting of heterogeneous collections of structured text. In Proc. 15th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1992, pp. 112–125.
2. Burkowski F.J. An algebra for hierarchical organized text-dominated databases. Inform. Process. Manag., 28:313–324, 1994.

3. Clarke C.L.A., Cormack G.V., and Burkowski F.J. An algebra for structured text search and a framework for its implementation. Comput. J., 38(1):43–56, 1995.
4. Consens M.P. and Milo T. Algebras for querying text regions. In Proc. 14th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1995, pp. 11–22.
5. Consens M.P. and Milo T. Algebras for querying text regions: expressive power and optimization. J. Comput. Syst. Sci., 57:272–288, 1998.
6. Cormack G.V., Clarke C.L.A., Palmer C.R., and Good R.C. The multitext retrieval system (demonstration abstract). In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, p. 334.
7. Jaakkola J. and Kilpelinen P. Using sgrep for querying structured text files. In Proc. SGML Finland 1996, J. Saarela (ed.), 1996, pp. 56–67.
8. Jaakkola J. and Kilpeläinen P. Nested text-region algebra. Technical Report C-1999-2, Department of Computer Science, University of Helsinki, January 1999.
9. Miller R.C. Lightweight Structure in Text. Ph.D thesis, School of Computer Science, Carnegie Mellon University, 2002.
10. Salminen A. and Tompa F. PAT expressions: an algebra for text search. Acta Linguistica Hungarica, 41(1–4):277–306, 1992.
11. Young-Lai M. Text Structure Recognition Using a Region Algebra. Ph.D thesis, Department of Computer Science, University of Waterloo, 2000.
12. Young-Lai M. and Tompa F.W. One-pass evaluation of region algebra expressions. Inf. Syst., 28(3):159–168, 2003.

Region Segmentation

► Image Segmentation

Registration Time

► Transaction Time

Regulatory Compliance in Data Management

Radu Sion
Stony Brook University, Stony Brook, NY, USA

Definition

Regulatory compliance in data management refers to information access, processing and storage mechanisms designed according to regulations governing their respective data types and semantics. For example, in the United States, health-related data falls under the incidence of the Health Insurance Portability and Accountability Act (HIPAA) and any associated data management systems need to provide compliance to HIPAA requirements, including data retention and secure deletion assurances. Such compliance has potential for far reaching impact in the design of data processing semantics, from relational ACID properties to complex query processing and index optimization.

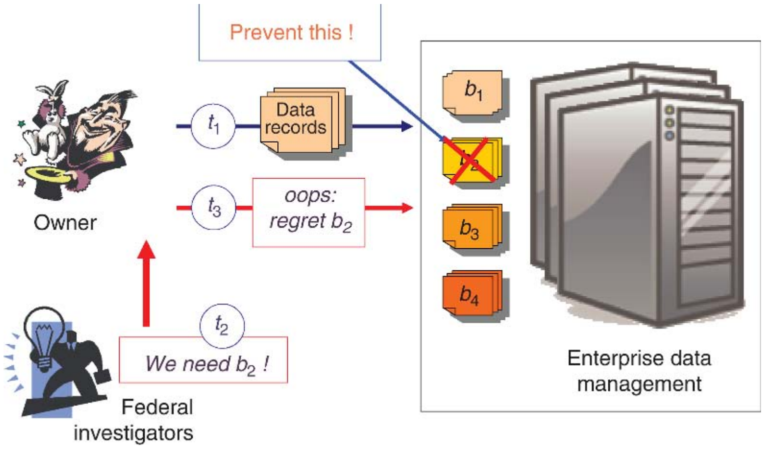
Historical Background

Modern digital societies and markets increasingly mandate consistent procedures for the access, processing and storage of information. In the United States alone, over 10,000 such regulations can be found in financial, life sciences, health-care and government sectors, including the Gramm–Leach–Bliley Act (1999), the Health Insurance Portability and Accountability Act (1996), and the Sarbanes–Oxley Act (2002). A recurrent theme in these regulations is the need for regulatory – compliant data management as an underpinning to ensure data confidentiality, access integrity and authentication; provide audit trails, guaranteed deletion, and data migration; and deliver Write Once Read Many (WORM) assurances, essential for enforcing long-term data retention and life-cycle policies.

Foundations

Overview

Over 10,000 regulations govern the management of documents in the United States alone [8], in financial, life sciences, health-care industries, and the government. These regulations impose a wide range of policies, ranging from information life-cycle management (e.g., mandatory data retention and deletion) to audit trails and storage confidentiality. Examples include the Gramm–Leach–Bliley Act [6], Health Insurance Portability and Accountability Act [13], Federal Information Security Management Act [14], Sarbanes–Oxley Act [15], Securities and Exchange Commission rule 17a-4 [12], Department of Defense Records Management Program under directive 5015.2 [9], Food and Drug Administration 21 CFR Part 11 [11], and the Family Educational Rights and Privacy Act [10]. In Europe, the Directive 95/46/EC of the European Union on the protection of personal data, provides privacy and security guarantees for personal information [3]. In addition, many



Regulatory Compliance in Data Management. Figure 1. WORM prevents history "re-writing".

countries have their own data protection laws. For example, in the United Kingdom, the Data Protection Act of 1998 [1] regulates, among other information, personal health-care records. It requires mandatory disposal of electronic records after retention period, accuracy of information, logging any changes, and strict confidentiality.

While each regulation has its own unique characteristics, certain assurance features can be found throughout:

Guaranteed data retention. To address this requirement, the goal of compliant data management is to support Write Once Read Many (WORM) semantics: once written, data cannot be undetectably altered or deleted before the end of their regulation-mandated life span, even with physical access to its hosting server.

Secure deletion. Once data has reached the end of its lifespan, it can (and in some cases must) be deleted. Deleted records should not be recoverable even with unrestricted access to the underlying medium; moreover, after data is deleted, no hints of its existence should remain on the server, even in the indexes. The term *secure deletion* is used to describe this combination of features.

Compliant data migration. Retention periods are measured in years. For example, national intelligence information, educational records, and certain health records have retention periods of over 20 years. To address this requirement, compliant data management needs *data migration* mechanisms that allow information to be transferred from obsolete to new storage media while preserving its associated security guarantees.

Litigation holds. Even if a data record has reached the end of its lifespan, it should remain fully accessible if it is the subject of current litigation.

In addition to these features, a common thread running through many of these regulations is the perception of powerful insiders as the primary adversary. These adversaries have superuser powers coupled with full access to the storage system hardware. This corresponds to the perception that much recent corporate malfeasance has been at the behest of CEOs and CFOs, who also have the power to order the destruction or alteration of incriminating records. Since the visible alteration or destruction of records is tantamount to an admission of guilt in the context of litigation, a successful adversary must perform their misdeeds *undetectably.*

Regulatory Compliant Commercial Systems

As the ability to control data retention is an essential requirement to be found in a majority of regulations, systems aimed at providing WORM assurances have been brought to the market by virtually all major storage vendors, including companies such as IBM [5], HP, EMC [2], Hitachi Data Systems Zantaz, StorageTek, Sun Microsystem Network Appliance and Quantum Inc. [7]. A set of representative instances are discussed in the following.

Tape-based WORM

Due to the favorable cost-per-MB ratio of tape-based storage in the past, it was a natural choice for massive data storage in commercial enterprise deployments (where regulatory compliance is of concern). Thus storage vendors started offering tape-based WORM mechanisms first. The Quantum DLTSage predictive, preventative and diagnostic tools for tape storage environments [7] are a representative instance. The WORM assurances of the tape systems are provided under the assumption that only Quantum tape-readers are deployed. "DLTSage WORM provides features to assure compliance, placing an electronic key on each cartridge to ensure WORM integrity. This unique identifier cannot be altered, providing a tamper-proof archive cartridge that meets stringent compliance requirements to ensure integrity protection and full accessibility with reliable duplication." [7]. Such systems however, are subject to a set of impractical assumptions. Given the nature of magnetic tape, an attacker can easily dismantle the plastic tape enclosure and access the underlying data on a different customized reader, thus compromising its integrity. Relying on the physical integrity of a "plastic yellow label" to safe-guard essential enterprise information is likely unacceptable in any medium to high-stake commercial scenarios.

Optical-Disk WORM

Optical (compact) disk (CD) media has been around experimentally since 1969 and commercially available since 1983. Given the prohibitive costs of high-powered lasers in small form factors, in the early days, most CD devices were only capable of reading disk information. As the technology matured, write-once (and later read-write) media appeared. Optical WORM-disk solutions rely on the irreversible physical primitive write effects to ensure the inability to alter existing content. However, with ever increasing amounts of information being produced and requiring constant low-latency accessibility in commercial scenarios, it is challenging to deploy a scalable optical-only WORM solution. Moreover, optical WORM disks are plagued with other practical issues such as the inability to fine-tune WORM and secure deletion granularity (problems partially shared also by tape-based solutions). Moreover, as the WORM disks do not provide any strong security features (due to bulk-production requirements) any optical WORM solutions are vulnerable to simple data replication attacks as discussed above. Optical WORM also perform relatively poorly in the price-performance measurements because current technology is somewhat under-sized for the volumes of data associated with compliance. Sony's Professional Disk for Data optical disk system, for example, holds only 23 GB per disk side. Nevertheless, because it is faster than tape and cheaper than hard disks, optical WORM storage technology is often deployed as a secondary, high-latency storage medium to be used in the framework of a hard disk-based solution. Care needs to be taken in establishing points of trust and data integrity when information leaves the secured hard disk store for the optical media.

Hard Disk-based WORM

Magnetic disk recording currently offers better overall cost and performance than optical or tape recording. Moreover, while immutability is often specified as a requirement for records, what is required in practice is that they be "term-immutable", i.e., immutable for a specified retention period. Thus almost all recently-introduced WORM storage devices are built atop conventional rewritable magnetic disks, with write-once semantics enforced through software ("soft-WORM").

EMC Centera. The EMC Centera Compliance Edition [2] is a content addressed storage (CAS) solution that also offers regulatory compliance capabilities. Each data record "has two components: the content and its associated content descriptor file (CDF) that is directly linked to the stored object (business record, e-mail, etc.). A digital fingerprint derived from the content itself is the content's locator (content address). The CDF contains metadata record attributes (e.g., creation date, time, format) and the object's content address. The CDF is used for access to and management of the record. Within this CDF, the application will assign a retention period for each individual business record. Centera will permit deletion of a pointer to a record upon expiration of the retention period. Once the last pointer to a record has been so deleted, the object will be eliminated" [2], and, in the Plus version, also "shredded" (from the media). Given its software-only nature, these mechanisms are vulnerable to simple software or physical direct disk-access attacks like described above. Data integrity can be easily compromised.

Hitachi message archive for compliance. Similarly, Hitachi Data Systems provide the Data Retention Utility, a software-based "virtual" WORM mechanism for mainstream Hitachi storage systems. The system allows customers to "lock down archived data, making it non-erasable and non-rewritable for prescribed periods, facilitating compliance with governmental or industry regulations".

IBM LockVault compliance software. IBM offers multiple soft-WORM solutions. The LockVault compliance software is a layer that operates on top of IBM System Storage N series to provide "disk-based regulatory compliance solutions for unstructured data".

IBM system storage archive manager. The IBM Tivoli Storage Manager is part of the IBM TotalStorage Software [5] and provides certain software data retention protection. It "makes the deletion of data before its scheduled expiration extremely difficult. *Short of physical destruction to storage media or server, or deliberate corruption of data or deletion of the Archive Manager database*, Archive Manager will not allow data [...] to be deleted before its scheduled expiration date." However, it is not desirable to base the functioning of the regulatory compliance mechanism on the correct behavior of the main system. After all, the compliance mechanism's main role is to guarantee exactly such fault-less behavior. The main adversary of concern in regulatory settings is *exactly* one with incentives for data corruption and physical compromise attacks.

Network appliance snaplock compliance/enterprise software. The NetApp SnapLock software suite is designed to work on top of NetApp NearStore and FAS storage systems. It provides soft-WORM assurances, "preventing critical files from being altered or deleted until a specified retention date". As opposed to other vendors, NetApp SnapLock supports open industry standard protocols such as NSF and CIFS.

Sun StorageTek compliance archiving software. Sun also offers soft-WORM assurances through its StorageTek Compliance Archiving Software . The software runs on top of the Sun StorageTek 5320 NAS Appliance to "provide compliance-enabling features for authenticity, integrity, ready access, and security".

Security Properties

The design of compliance data management is extremely challenging due to the conflict between security, cost-effectiveness, and efficiency. For example, the requirement to find requested information quickly means in practice data must be indexed. But trustworthy indexing of compliance data is a challenging problem, as it is easy to tamper with traditional indexes stored on WORM. Further, trustworthy indexes will make it very hard to delete all traces of documents that are past their retention periods, as required by certain regulations. Yet another complicating factor is the decades-long retention periods required by many regulations; it is unrealistic to expect data to reside on the same device for so long. One of the main challenges of such an endeavor lies in securing the system against attack by insiders with superuser powers, and in balancing the conflicting requirements for trustworthiness, high performance, and low cost.

Current regulatory compliant data management systems constitute an important step in the right direction. These systems however, are unfortunately fundamentally vulnerable to faulty behavior or illicit adversaries with incentives to alter stored data, as they rely on enforcement primitives – such as software and/or simple hardware device-hosted on/off switches – ill-suited to their target (insider) adversarial setting.

Achieving a secure, cost-effective, and efficient design when the insider is the adversary is extremely challenging. To defend against insiders, processing components that are both tamper-resistant and *active*, such as general – purpose trustworthy hardware are needed. By offering the ability to run logic within a secured enclosure, such devices allow fundamentally new paradigms of trust. Trust chains spanning untrusted and possibly hostile environments can now be built by deploying secure tamper-resistant hardware at the storage components' site. The trusted hardware can run certified logic; close proximity to data coupled with tamper-resistance guarantees allow an optimal balancing and partial decoupling of the efficiency/security trade-off. Assurances can now be both efficient and secure.

However, trusted hardware devices are not a panacea. Their practical limitations pose a set of significant challenges in achieving sound regulatory-compliance assurances. Specifically, heat dissipation concerns under tamper-resistant requirements limit the maximum allowable spatial gate-density. As a result, general-purpose secure coprocessors (SCPUs) are often significantly constrained in both computation ability and memory capacity, being up to one order of magnitude slower than host CPUs. Such constraints mandate careful consideration in achieving efficient protocols. Direct implementations of the full processing logic *inside* the SCPU are bound to fail in practice due to lack of performance. The server's main CPUs will remain starkly under-utilized and the entire cost-proposition of having fast untrusted main CPUs and expensive slower secured CPUs will be defeated. Efficient protocols need to access the secure hardware sparsely, asynchronously from the main data flow.

Recently, Hsu et al. [4] analyzed some of the principal requirements for trustworthy "content immutable storage" (CIS) of electronic records and identified a set of principles, including: (i) increasing the cost and conspicuity of any attack against the system, (ii) focusing on end-to-end trust, rather than single components, (iii) using a small trusted computing base, isolate trust-critical modules and make them simple, verifiable and correct, (iv) using a simple, well-defined interface between trusted and untrusted components, and (v) trust, but verify every component and individual operation.

Key Applications

Recent compliance regulations are intended to foster and restore humans trust in digital information records and, more broadly, in our businesses, hospitals, and educational enterprises. As increasing amounts of information are created and live digitally, compliance data management will be a vital tool in restoring this trust and ferreting out corruption and data abuse at all levels of society.

Future Directions

Future research will need to explore novel regulatory compliant data management solutions that address the appropriate insider adversary. Numerous technical challenges are associated with such an endeavor, including designs for SCPU overhead-minimizing techniques for expected transaction loads, amortized commitment schemes to enforce write-once-read-many (WORM) semantics at the throughput rate of the servers ordinary processors, while benefiting from existing work on compliance indexing to create efficient indexes secured by novel cryptographic constructs such as fast short-lived signatures. Compliant data migration mechanisms will likely require backwards-compatible, inter-device trust chains and fast re-encryption techniques.

Cross-references

► Trusted Hardware

Recommended Reading

1. British Parliament. Data Protection Act of 1998. http://www.staffs.ac.uk/legal/privacy/dp10rules/index.php, 1998.
2. EMC. Centera Compliance Edition Plus. http://www.emc.com/centera/ and http://www.mosaictech.com/pdf_docs/emc/centera.-pdf, 2007.
3. European Parliament. Legislative Documents. Online at http://ec.europa.eu/justice_home/fsj/privacy/law/index_en.htm, 2006.
4. Hsu W., Huang L., and Ong S. Content Immutable Storage: Truly Trustworthy and Cost-Effective Storage for Electronic Records. Research Report RJ 10332. Technical Report, 2004.
5. IBM Corp. IBM TotalStorage Enterprise. http://www03.ibm.com/servers/storage/, 2007.
6. National Association of Insurance Commissioners. Graham-Leach-Bliley Act, 1999. www.naic.org/GLBA.
7. Quantum Inc. DLTSage Write Once Read Many Solution. http://www.quantum.com/Products/TapeDrives/DLT/SDLT600/DLTIce/Index.aspx and http://www.quantum.com/pdf/DS00232.pdf, 2007.
8. The Enterprise Storage Group. Compliance: The effect on information management and the storage industry. Online at http://www.enterprisestoragegroup.com/, 2003.
9. The U.S. Department of Defense. Directive 5015.2: DOD Records Management Program. Online at http://www.dtic.mil/whs/directives/corres/pdf/50152std_061902/p50152s.pdf, 2002.
10. The U.S. Department of Education. 20 U.S.C. 1232g; 34 CFR Part 99:The Family Educational Rights and Privacy Act (FERPA). http://www.ed.gov/policy/gen/guid/fpco/ferpa, 1974.
11. The U.S. Department of Health and Human Services Food and Drug Administration. 21 CFR Part 11: Electronic Records and Signature Regulations. http://www.fda.gov/ora/compliance_ref/part11/FRs/background/pt11finr.pdf, 1997.
12. The U.S. Securities and Exchange Commission. Rule 17a-3&4, 17 CFR Part 240: Electronic Storage of Broker-Dealer Records. Online at http://edocket.access.gpo.gov/cfr_2002/aprqtr/17cfr240.17a-4.htm, 2003.
13. U.S. Department of Health & Human Services. The Health Insurance Portability and Accountability Act (HIPAA), 1996. www.cms.gov/hipaa.
14. U.S. Public Law 107-347. The E-Government Act, 2002.
15. U.S. Public Law No. 107-204, 116 Stat. 745. newblock The Public Company Accounting Reform and Investor Protection Act, 2002.

Re-identification

► Record Linkage

Re-Identification Risk

► Disclosure Risk

Relational Algebra

VAL TANNEN
University of Pennsylvania, Philadelphia, PA, USA

Definition

The operators of the relational algebra were already described in Codd's pioneering paper [2]. In [3] he introduced the term *relational algebra* and showed its equivalence with the tuple relational calculus.

This entry details the definition of the relational algebra in the *unnamed perspective* [1], with selection, projection, cartesian product, union and difference operators. It also describes some operators of the *named perspective* [1] such as join.

The flagship property of the relational algebra is that it is equivalent to the (undecidable!) set of domain independent relational calculus queries thus providing a standard for *relational completeness.*

Key Points

Fix a countably infinite set $\mathbb{D}$ of constants over which Σ-*instances* are defined for a relational schema Σ.

The relational algebra is a *many-sorted* algebra, where the sorts are the natural numbers. The idea is that the elements of sort n are finite n-ary relations. The *carrier* of sort n of the algebra is the set of finite n-ary relations on $\mathbb{D}$ If f is a many-sorted k-ary operation symbol that takes arguments of sorts $n_1,\ldots,n_k$ (in this order) and returns a result of sort n then its type is written as follows: $f : n_1 \times\ldots \times n_k \rightarrow n_0$, and this is simplified to n for nullary ($k = 0$) operations. Bold letters $\mathbf{x}$, $\mathbf{y}$ used for tuples and x_i for the i'th component of $\mathbf{x}$. The operations of the algebra, with their types and their interpretation over the relational carriers are the following:

constant-singletons $\{c\} : 1(c \in \mathbb{D})$.

selection1 $\sigma^n_{ij} : n \rightarrow n$ $(1 \leq i < j \leq n)$ interpreted as $\sigma^n_{ij}(R) = \{\mathbf{x} \in R \mid x_i = x_j\}$

selection2 $\sigma^n_{ic} : n \rightarrow n$ $(1 \leq i \leq n, c \in \mathbb{D})$ interpreted as $\sigma^n_{ic}(R) = \{ \mathbf{x} \in R | x_i = c \}$

projection $\pi i^{\mathrm{n}}_1\ldots ik : n \rightarrow k$ $(1 \leq i_1,\ldots,i_k \leq n$, not necessarily distinct) interpreted as $\pi^n_{i_1\ldots i_k}(R) = \{x_{i_1},\ldots, x_{i_k} \mid \mathbf{x} \in R\}$

cartesian(cross-) product $\times^{mn} : m \times n \rightarrow m + n$ interpreted as $\times^{mn}(R, S) = \{x_1,\ldots,x_m, y_1,\ldots,y_n \, | \mathbf{x} \in R \wedge \mathbf{y} \in S\}$

union $\cup^n : n \times n \rightarrow n$ interpreted as $\cup^n(R, S) = \{\mathbf{x} | \mathbf{x} \in R \vee \mathbf{x} \in S\}$

difference $-^n : n \times n \rightarrow n$ interpreted as $-^n(R, S) = \{\mathbf{x} | \mathbf{x} \in R \wedge \mathbf{x} \notin S\}$

Relational algebra *expressions* are built, respecting the sorting, from these operation symbols, using the relational schema symbols *as variables.*

Note that an obvious operation, intersection, is missing. Of course, intersection can be defined from union and difference, by De Morgan's laws. Interestingly, intersection is also definable just from cartesian product, selection, and projection. Other useful operations are definable also.

Given a relational schema Σ, a relational algebra *query* is an algebraic expression constructed from the symbols in Σ and the relational algebra operation symbols. Given a database instance $\mathcal{I}$ as input, such a query e returns a relation $e(\mathcal{I})$ as output. For example if R, S are binary, the expression $\pi_{2414}(\sigma_{13}(R \times S)) - (R \times R)$ defines a query that returns a 4-ary relation (omit the operation's superscripts because they can usually be reconstructed and use infix notation for the binary operations). Clearly, each of the operations of the relational algebra maps finite instances to finite relations, and more importantly, it is easily seen that *relational algebra queries are domain independent.* Moreover, there exists an (easily) computable translation that takes any relational algebra query into an equivalent domain independent FO (first-order) query.

The converse of this last fact is the main "raison d'être" for the relational algebra. However, because the set of domain independent FO queries is not decidable, it is not possible to define an effective translation just for these queries. Instead, define a "translation" for *all* FO queries, such that domain independent FO queries are indeed translated to equivalent relational algebra queries. Recalling the notation $q(\mathcal{I}/D)$ gives

Theorem

There exists a total computable translation that takes any FO query q into a relational algebra query e such that for any instance $\mathcal{I}$, gives $e(\mathcal{I}) = q(\mathcal{I}/adom(\mathcal{I}) \cup adom(q))$ (and for domain independent queries the right-hand side further equals $q(\mathcal{I})$.

The key to the proof is the observation that active domains can be computed in the relational algebra.

This result justifies Codd calling a query language *relationally complete* whenever it has the expressive power of the relational algebra. However, it is necessary to note that the relational algebra also inherits the negative results about first-order logic: it is undecidable whether there exists some instance on which a given query returns a non-empty answer (satisfiability) and it is undecidable whether two queries are equivalent.

The presentation above assumes that the relational symbols have just arity. This is the so-called *unnamed perspective* [1] and it is convenient for theoretical investigations. The practical descriptions of the relational model, e.g., [4], use the *named perspective*[1] in which a set of *attributes* is fixed and each relation is organized vertically by a finite set of them. For such as relation, a tuple is function from its attributes to constants. The relational algebra operators in the named perspective are similar to the ones above except that attributes are used instead of the integers identifying components of tuples. Complications arise with cartesian product when the two relations have attributes in common. Thus, in the named perspective an additional operator is needed for the *renaming* of a relation's attributes. On the positive side, using attributes leads to an nice generic definition of *natural join*, and operation that generalizes both cartesian product (when attribute sets are disjoint) and intersection (when attribute sets are identical) and which has many elegant properties.

Cross-references

- ▶ Cartesian Product
- ▶ Computationally Complete Relational Query Languages
- ▶ Difference
- ▶ Division
- ▶ First-Order Logic: Semantics
- ▶ First-Order Logic: Syntax
- ▶ Join
- ▶ Natural Join
- ▶ Projection
- ▶ Relational Algebra
- ▶ Selection
- ▶ Union

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases: The Logical Level. Addison Wesley, Reading, MA, 1994.
2. Codd E.F. A Relational Model of Data for Large Shared Data Banks. Commun. ACM, 13(6):377–387, 1970.
3. Codd E.F. Relational completeness of database sublanguages, In Courant Computer Science Symposium 6: Data Base Systems, R. Rustin (ed.). Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 65–98.
4. Ramakrishnan R. and Gehrke J. Database Management Systems, 3rd edn. McGraw-Hill, New York, 2003.

Relational Algebra for XML

▶ XML Tuple Algebra

Relational Calculus

VAL TANNEN
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

Domain relational calculus; Tuple relational calculus; First-order query

Definition

The relational database model was proposed by Codd in [2] where he assumed that its "data sublanguage" would be based on the predicate calculus (FOL) and where he introduced various algebraic operations on relations. Only in [3] did he introduced the terms *relational algebra* and *relational calculus*.

Later, it became customary to talk about the *domain relational calculus* (detailed below), which is closely related to the syntax of first-order logic and has quantified variables ranging over individual constants, and about the *tuple relational calculus* which is in fact the one given by Codd in [3] and whose variables range over tuples of constants. The two calculi are equivalent, via easy back and forth translations. However, both calculi allow the formulation of *domain dependent* queries which are inappropriate for database languages. While domain independence is undecidable, it is possible to define decidable sublanguages of *safe* queries which are themselves domain independent and such that any domain independent query is equivalent to a safe one.

Key Points

A *relational (database) schema* is a finite first-order vocabulary consisting only of relation symbols. In fact, relational formalisms also permit constants so it is necessary to fix a countably infinite set $\mathbb{D}$ of constants and work with formulae over the first-order vocabulary $\Sigma \cup \mathbb{D}$ where Σ is a relational schema. Also, the set $\mathbb{D}$ is taken as the sole *universe of discourse* for the interpretation of formulae. A *relational (database) instance* for a given schema Σ (a Σ-*instance*) is a first-order structure whose domain, or universe, is $\mathbb{D}$ and in which the relation symbols are interpreted by *finite* relations, while the constants are interpreted as themselves.

For a given schema Σ , a *domain relational calculus query* (a.k.a. first-order query) has the form $\{\langle e_1,\dots,e_n\rangle \mid \varphi\}$ where $e_1,\dots,e_n$ are (not necessarily distinct) variables or constants and φ is a first-order formula over the vocabulary $\Sigma \cup \mathbb{D}$ with equality such that the all free variables of φ occur among $e_1,\dots,e_n$. The *inputs* of the query are the Σ-instances. For each input $\mathcal{I}$, the *output* of the query $q \equiv \{\langle e_1,\dots,e_n\rangle \mid \varphi\}$ is the n-ary relation

$$q(\mathcal{I}) = \{\langle \bar{\mu}(e_1),\dots,\bar{\mu}(e_n)\rangle \mid \\ \textit{assignment } \mu \textit{ such that } \mathcal{I}, \mu \models \varphi\}$$

Let $\mathcal{I}$ be an instance. The *active domain* of $\mathcal{I}$, notation $adom(\mathcal{I})$, is the set of all elements of $\mathbb{D}$ that actually appear in the relations that interpret Σ in $\mathcal{I}$. While $\mathbb{D}$ is infinite, $adom(\mathcal{I})$ is always finite. Moreover, given a query $q \equiv \{\langle e_1,\dots,e_n\rangle \mid \varphi\}$ $adom(q)$ is denoted by the (finite) set of constants that occur in φ or among $e_1,\dots,e_n$. One expects that the instance $\mathcal{I}$ together with the query q completely determines the output $q(\mathcal{I})$. In particular, only the elements in $adom(\mathcal{I}) \cup adom(q)$ can appear in the output. However, this is not the case for all first-order queries. For example, the outputs of $\{x \mid \neg R(x)\}$ or $\{\langle x,y\rangle \mid R(x) \vee S(y)\}$ are in fact infinite! More subtly, the following query is also problematic: $\{x \mid \forall y R(x,y)\}$. Here the output contains only elements from $adom(\mathcal{I})$ but whether a tuple is in the output or not depends on the set of elements that y ranges over. These queries are "dependent on the domain." More precisely, for any instance $\mathcal{I}$ and any D such that $adom(\mathcal{I}) \cup adom(q) \subseteq D \subseteq \mathbb{D}$, denote by $q(\mathcal{I}/D)$ the output of the query q on the input structure obtained by restricting the domain to D. A query q is *domain independent* if for any $\mathcal{I}$ and any D_1,D_2 where $adom(\mathcal{I}) \cup adom(q) \subseteq D_i \subseteq \mathbb{D}, i = 1, 2$ one has $q(\mathcal{I}/D_1) = q(\mathcal{I}/D_2)$. It is generally agreed that in a reasonable *query language*, all the queries should be domain independent. Therefore, general first-order queries do not make a good query language. Worse, it is undecidable whether a first-order query is domain independent [1]. So how does one get a reasonable query language? It is possible (in several ways) to define decidable *safety* restrictions on general first-order formulae such that the safe queries are domain independent and moreover for any domain independent query there exists an equivalent safe query [4,1]. The safety restrictions tend to be complicated and have little practical value. A better idea is the *relational algebra*.

Clearly, not all functions that map instances to relations are meanings of first-order queries. However, the meanings of first-order queries are all *generic*, i.e., invariant under bijective renamings of the constants in $\mathbb{D} \setminus adom(q)$. Conversely, a function f that maps instances to relations and is generic is said to be *first-order definable* (a.k.a. definable in the relational calculus) if there exists a first-order query q whose semantics is f (contrast this with the definability within a given structure described in FIRST-ORDER LOGIC: Semantics). A well-known example of function that is not first-order definable is taking the *transitive closure* of a binary relation.

Codd's *tuple relational calculus* [4,3] differs from the domain relational calculus in that its variables range over tuples and its terms are either constants or of the form $t.k$ where t is a variable and k is a positive integer selecting one of the components of the tuple (for example if t is assigned to (a,b,c) then the meaning of $t.2$ is b).

Cross-references

▶ Computationally Complete Relational Query Languages
▶ First-Order Logic: Semantics
▶ First-Order Logic: Syntax
▶ Relational Algebra

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases: The Logical Level. Addison Wesley, Reading, MA, 1994.
2. Codd E.F. A relational model of data for large shared data banks. Commun. ACM, 13(6):377–387, 1970.
3. Codd E.F. Relational Completeness of Database Sublanguages. In Courant Computer Science Symposium 6: Data Base Systems, R. Rustin (ed.). Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 65–98.
4. Ullman J.D. Principles of Database and Knowledge-Base Systems Volume, I. Computer Science Press, Rockville, MD, 1988.

Relational Database

▶ Relational Model

Relational Integrity Constraints

▶ FOL Modeling of Integrity Constraints (Dependencies)

Relational Model

David W. Eembley
Brigham Young University, Provo, Utah, USA

Synonyms

Relational database

Definition

The *Relational Model* describes data as named relations of labeled values. For example, customer ID's can relate with customer names and addresses in the relational model as *Customer*: {<(*CustomerID*, *11111*), (*Name*, *Pat*), (*Address*, *12 Maple*)>, <(*CustomerID*, *22222*), (*Name*, *Tracy*), (*Address*, *44 Elm*)>}. In this example, there is a name for the relation – *Customer*; label-value pairs – e.g., (*CustomerID*, *11111*), which provide the labeled values; and tuples – e.g., <(*CustomerID*, *11111*), (*Name*, *Pat*), (*Address*, *12 Maple*)>, which are the tuples of the named relation.

Usually, the relations of the relational model are viewed as tables. Figure 1 shows an example of several relations viewed as tables. Together, they constitute a relational database. The first table in Fig. 1 is the table view of the relation described in the previous paragraph.

Besides their structures, tables in the relational model also have constraints. Typical constraints include key constraints (e.g., *CustomerID* values must be unique), type constraints (e.g., *DateOrdered* values must be of type *Date*), and referential integrity constraints (e.g., the *CustomerID* values in table *Order* must refer to existing *CustomerID* values in table *Customer*).

Historical Background

Codd's seminal paper [2] introduced the relational model as a model based on *n*-ary relations. The seminal paper also introduces normal forms for relations and a query language for relations. In perspective, Codd essentially showed how to apply fundamental concepts in set theory to form the relational data model, the foundation of relational databases.

Several textbooks describe the relational data model. Widely read descriptions are in the textbook by Elmasri and Navathe [3] and the textbook by Silberschatz, Korth, and Sudarshan [5], both of which appear in their fifth edition. A few textbooks focus solely on the relational model and treat it from a theoretical perspective [1,4].

Foundations

The relational model can be viewed in several ways: (i) intuitively, (ii) conceptually, (iii) as an implementation description, and (iv) formally. Each view satisfies different user needs.

Intuitive View

Figure 1 shows three tables that together model a simple item-order application. An intuitive description of the relational model has the following characteristics:

- A group of tables stores the data of an application, and together the tables constitute a relational database (e.g., the tables in Fig. 1).

Customer	(CustomerID	Name	Address)
	11111	Pat	12 Maple
	22222	Tracy	44 Elm

Item	(ItemNr	Description)
	X11-222	Ball
	Y33-444	Game
	Z11-555	Book

Order	(OrderNr	CustomerID	ItemNr	DateOrdered	NrOrdered)
	123-1	11111	Y33-444	12 May	1
	114-7	22222	X11-222	10 May	5
	114-7	22222	Y33-444	10 May	5

Relational Model. Figure 1. Sample named relations of labeled data viewed as tables in the relational model.

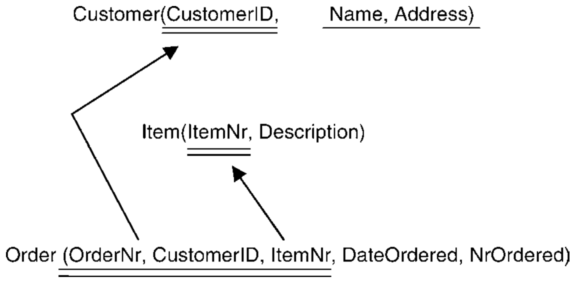
- Each table has a name (e.g., *Customer* for the first table in Fig. 1).
- Each table has labeled column headers (e.g., *CustomerID*, *Name*, and *Address* for the first table in Fig. 1).
- Each table has data for each column header (e.g., *Pat* and *Tracy* for the *Name* column of the first table in Fig. 1).
- The rows of each table constitute a record for the table (e.g., <*11111*, *Pat*, *12 Maple*> for the first row in the first table in Fig. 1).
- Each record asserts a fact that relates the table name, column headers, and record values (e.g., the first row of the first table in Fig. 1 asserts that a *Customer* exists whose *CustomerID* is *11111*, whose *Name* is *Pat*, and whose *Address* is *12 Maple*).
- Each value belongs to a specified domain, a defined set of possible values (e.g., *CustomerID* values are five-digit numbers).
- A key uniquely identifies each record (e.g., in Fig. 1 *CustomerID* uniquely identifies customers and the triple *OrderNr*-*CustomerID*-*ItemNr* uniquely identifies a line item in an order).
- Values link records in different tables (e.g., in Fig. 1 the value *11111* links the customer whose *CustomerID* is *11111* with the orders for that customer by recording the *CustomerID* in the same record as the *OrderNr*).
- For these cross-table linkages to make sense, referenced values must exist (e.g., in Fig. 1 an order record references customers by *CustomerID*'s; thus, any *CustomerID* value recorded in the *Order* relation must exist in the *Customer* relation).

Conceptual View

In a conceptual view of the relational model, the focus is on the conceptual schema of each table. The collection of table schemas for all the tables constitutes the relational database schema.

Figure 2 shows a schematic view of the relational database schema for the database in Fig. 1. A *relational schema* is a named template for a set of *n*-tuples with the following features.

- The schema has a name – *Customer*, for example, in Fig. 2.
- The schema has *n* attribute names, one for each component of the *n*-tuples being represented. The *n* attribute names must all be different so that they form a set. The attribute names are usually just called attributes. In Fig. 2 the *Customer* schema has three attributes: *CustomerID*, *Name*, and *Address*.


Customer(CustomerID, Name, Address)
Item(ItemNr, Description)
Order (OrderNr, CustomerID, ItemNr, DateOrdered, NrOrdered)

Relational Model. Figure 2. Schematic of table headers.

- The schema has key constraints. An attribute (or a set of attributes) whose value (or values) uniquely determine at most one tuple is a key. One of these keys is the primary key; often there is just one key, which therefore is the primary key. In the schematic view, underlines designate keys; double underlines designate primary keys. In the *Customer* schema in Fig. 2, *CustomerID* is the primary key, and the pair *Name*-*Address* is another key.
- The schema has referential integrity constraints. This prevents referencing objects that do not exist. In the example in Fig. 2, the customers referred to in the *Order* table must exist in the *Customer* table, and the items referred to must exist in the *Item* table. Almost always, the referring attribute refers to a key, usually the primary key. Thus, the referring attribute is called a foreign key – a key in another table. When an attribute is a foreign key, the values for the attribute in the *n*-tuples for the table must be a subset of the values for the attribute in the *m*-tuples of the referenced table. Schematically, as Fig. 2 shows, an arrow is drawn from a foreign key to a key. In the example, the customer IDs in the *Order* tuples must be a subset of the customer IDs in the *Customer* tuples, and the item numbers in the order tuples must be a subset of the item numbers in the *Item* tuples.
- Optionally, the schema may also have domain specifications for the attributes. For example, customer IDs may always be five-digit numbers. If so, "*CustomerID*: *five-digitnumber*" could be written as a note in the schematic diagram in Fig. 2.
- Optionally, the schema may also have additional information. Additional constraints or notes may

be written – for example, a note to say that the *DateOrdered* should be initialized as today's date.

Implementation View

The language used to express implementation views is called a Data-Definition Language (DDL). Figure 3 shows an example for the sample customer-order application in Fig. 1. The DDL in Fig. 3 is SQL, a standard commercial database language for specifying, updating, and querying a database.

SQL allows database developers to declare relations for the database and constraints over and among these relations. The SQL syntax provides a way to declare these basic relational-model features as follows.

- A developer declares a schema for a table with a *create table* declaration. The developer must provide a name for the table, and then must declare each of the table's attributes. *Customer* is the name for the first table declared in Fig. 3.
- A developer declares an attribute by giving its name and then listing the constraints that apply to the attribute. The attributes in the *Customer* table declared in Fig. 3 are *CustomerID*, *Name*, and *Address*.
- A developer declares domain constraints for an attribute by giving a type declaration. In Fig. 3 the type declaration for *CustomerID* is *numeric* (5), declaring that customer IDs are 5-digit numbers. SQL provides various types such as numbers, strings, dates, time, and money. Type declarations are unfortunately not uniform across all database systems.
- A developer declares key constraints for an attribute either by stating that it is the *primary key* or that it is *unique* – a key but not the primary key. In Fig. 3 *CustomerID* in the first table schema and *ItemNr* in the second table schema are primary keys.
- A developer declares foreign-key constraints for an attribute with a *references* clause. The *references* clause designates the table in which the referenced attribute is found. When the attribute in the referenced table has the same name as the attribute in the referencing table, this simple declaration is sufficient. If the name is different, then the *references* clause must also include the name of the attribute being referenced. In Fig. 3, *CustomerID* in the *Order* table references *CustomerID* in the *Customer* table, and *ItemNr* references *ItemNr* in the *Item* table. These foreign-key constraints ensure that the *CustomerID* and *ItemNr* values in the *Order* table refer to existing values in the *Customer* and *Order* tables.
- When constraints involve multiple attributes, SQL provides syntax that allows a developer to declare these constraints in a separate entry in a table declaration. Thus, as Fig. 3 shows, a developer can declare that the attribute combination consisting of *Name* and *Address* constitutes a key for the *Customer* table, and that the attribute combination consisting of *OrderNr*, *CustomerID*, and *ItemNr* constitutes the primary key for the *Order* table. The SQL syntax also provides for multiple-attribute foreign keys. Thus, although neither necessary nor even desirable in the example in Fig. 3, the attributes *Name* and *Address* could be added to the *Order* table and a foreign-key constraint could then be declared as *foreign key* (*Name*, *Address*) *referencesCustomer* (*Name*, *Address*).
- Typical additional constraints declarable with SQL include null constraints and check constraints. Null constraints let developers decide whether null values can or cannot appear as values for attributes. Values for primary-key attributes may never be null; other attributes require a *not null* designation (otherwise they can have null values). Check constraints let developers add conditions that must hold. For example, a developer can declare that *NrOrdered* can be neither negative nor zero by adding the constraint *check*($NrOrdered > 0$) to the *NrOrdered* attribute in the *Order* table in Fig. 3.

```
create table Customer (
    CustomerID numeric(5) primary key,
    Name varchar(20),
    Address varchar(25),
    unique (Name, Address)
    );

create table Item (
    ItemNr char(7) primary key,
    Description varchar(50)
    );

create table Order (
    OrderNr varchar(10),
    CustomerID numeric(5) references Customer,
    ItemNr char(7) references Item,
    DateOrdered date,
    NrOrdered smallint,
    primary key (OrderNr, CustomerID, ItemNr ),
    );
```

Relational Model. Figure 3. SQL implementation schemas.

Formal View

The formal view of the relational model captures the essence of a relational schema in terms of mathematical concepts. The definition is based on the concepts of sets, relations, and functions.

A *relational schema* R is a non-empty set of attribute names $R = \{A_1, A_2,...,A_n\}$. Usually an "attribute name" as just called an "attribute." Further, as a notational convenience, when an attribute name is a single letter, the set notation is reduced by dropping the braces, commas, and spaces. Thus, a *relational schema* R is a non-empty set of attributes $R = A_1A_2...A_n$. Each attribute A has a domain, denoted $dom(A)$, which is a set of values.

A relation is always defined with respect to a relational schema. The notation $r(R)$ denotes that relation r is defined with respect to a relational schema R and is read "r is a relation on schema R" or just "r on R" when the context is clear. A *relation* is a set of n-tuples, $\{t_1,..., t_k\}$, where n is $|R|$, the cardinality of R. Let schema R be $A_1A_2...A_n$. Then, an *n-tuplet* for a relation $r(R)$ is a function, from R to the union of domains $D = dom(A_1) \cup dom(A_2) \cup ... \cup dom(A_n)$, with the restriction that $t(A_i) \in dom(A_i)$, $1 <= i <= n$.

A relation is usually written as a table – the table in Fig. 4, for example. In Fig. 4 the relation r has relational schema $R = AB$. To declare the attribute domains, assume $dom(A)$ is $\{a, b, c\}$ and $dom(B)$ is $\{0,...,9\}$. The set of n-tuples for r is the set of discrete functions $\{\{(A, a), (B, 1)\}, \{(A, b), (B, 1)\}, \{(A, b), (B, 2)\}\}$.

Another way to view the relational model formally is to view it as an interpretation in first-order logic. In this view of the relational model each relation r, whose schema has n attributes, is an n-place predicate. An *interpretation* for a first-order language consists of (1) a non-empty domain D of values, which under the unique name assumption each represent themselves, and (2) for each n-place predicate, an assignment of *True* or *False* for each possible substitution of n values from D.

As an example, consider the relation in Fig. 4. To see this relation as an interpretation, let $r(x, y)$ be a two-place predicate. The domain D is {a, b, c, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. *True* is assigned to the substitutions $(x = a, y = 1)$, $(x = b, y = 1)$, and $(x = b, y = 2)$ and *False* is assigned to all other substitutions.

r =	A	B
	a	1
	b	1
	b	2

Relational Model. Figure 4. Sample table for illustrating the formal definition.

Most often in this first-order-logic view, the "closed world assumption" is used to conveniently assign *True* and *False* to each substitution. The "closed world assumption" states that whatever is not *True* is *False*, and thus only the *True* facts need to be recorded. This, then, reduces to the equivalent of just giving a table such as the one in Fig. 4 – the rows in the table represents the substitutions for which the predicate is *True* and the only substitutions for which the predicate is *True*.

In a first-order-logic view of the relational model, it is also possible to express constraints. For the table in Fig. 4, for example, the constraint that A should be a key can be expressed by the closed formula $\forall x_1 \forall y_1 \forall x_2 \forall y_2 (r(x_1, y_1) \wedge r(x_2, y_2) \wedge x_1 = x_2 \Rightarrow y_1 = y_2)$. This does not hold for the table in Fig. 4, however, since the second and third tuples have $x_1 = x_2$, but $y_1 \neq y_2$. A check constraint stating that the values in the B column must all be less than 5 can be expressed as $\forall x \, \forall y (r(x, y) \Rightarrow y < 5)$. This constraint does hold in the table in Fig. 4.

For an interpretation for a first-order language, when all the closed formulas hold, the interpretation is said to be a *model*. Database instances in which all constraints hold are therefore models – models of the world they represent.

Key Applications

Relations, stored in relational databases, are widely used in industry. Indeed, their combined usage constitutes a mega-billion-dollar industry. Relational databases range from relatively small databases used as backends to web applications such as "items for sale" to relatively large databases used to store corporate data.

Cross-references

- ▶ Database Normalization
- ▶ Key
- ▶ Relational Algebra
- ▶ Relational Calculus
- ▶ SQL

Recommended Reading

1. Atzeni P. and De Antonellis V. Relational Database Theory, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1993.

2. Codd E.F. A relational model for large shared data banks, Commun. ACM, 13(6)377–487, 1970.
3. Elmasri R. and Navathe S.B. Fundamentals of Database Systems, Fifth Edition. Addison-Wesley, Boston, MA, 2007.
4. Maier D. The Theory of Relational Databases, Computer Science Press, Inc., Rockville, MA, 1983.
5. Silberschatz A., Korth H.F., and Sudarshan S. Database System Concepts, Fifth Edition. McGraw-Hill, New York, 2006.

Relational Query Processor

▶ Query Processor

Relation-Completeness

▶ BP-Completeness

Relations with Marked Nulls

▶ Naive Tables

Relationship of Reliance

▶ Trust in Blogosphere

Relationships in Structured Text Retrieval

Mounia Lalmas
Queen Mary, University of London, London, UK

Definition

In structured text retrieval, the relationship between text components may be used in ranking components relative to a given query.

Key Points

In a structured text document, there exists a relationship between the document components. In the context of XML retrieval, the relationships between elements are provided by the logical structure of the XML mark-up. An element, unless it is the root element (the document itself), has a parent element, which itself may have a parent element. Similarly, non-leaf elements have children elements, and so on. Considering relationships between elements appears to be beneficial for XML retrieval. For instance, in a collection of scientific articles, it is reasonable to assume that the "abstract" of an article is a better indicator of what the article is about than a "future work" section in the same article. The challenge in XML retrieval is what types of relationship should be considered, and how this information can be used to score elements according to how relevant they are for a given query. In the contextualization approach [1], considering the "root element – element" relationship to rank an element in addition to the element own content has shown to improve retrieval effectiveness [2,3].

Cross-references

▶ Contextualization
▶ Logical Structure
▶ Structure Weight
▶ XML Retrieval

Recommended Reading

1. Arvola P., Junkkari M., and Kekäläinen J. Generalized contextualization method for XML information retrieval. In Proc. ACM Conf. on Information and Knowledge Management, 2005, pp. 20–27.
2. Mass Y. and Mandelbrod M. Component ranking and automatic query refinement for XML retrieval. In Advances in XML Information Retrieval and Evaluation, LNCS, vol. 3493, Springer, 2005, pp. 73–84.
3. Sigurbjörnsson B., Kamps J., and de Rijke M. An element-based approach to XML retrieval. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2003, pp. 19–26.

Relative Operating Characteristic

▶ Receiver Operating Characteristic (ROC)

Relative Time

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

Viewing a temporal database as a collection of time-referenced (or timestamped) facts, a time reference in such a database is called *relative* if its value is dependent of the context. For example, this context can be the current time, *now*, or it can be another instant.

Key Points

The relationship between times can be qualitative (before, after, etc.) as well as quantitative (3 days before, 397 years after, etc.). If quantitative, the relationship is specified using a time span.

Examples: "Mary's salary was raised yesterday," "it happened sometime last week," "it happened within 3 days of Easter, 2005" "the Jurassic is sometime after the Triassic," and "the French revolution occurred 397 years after the discovery of America".

The simplest example of a relative timestamp is a period that starts at a time in the past and extends to *now*, such as (2005, *now*). As the clock ticks, this period gets longer.

Cross-references

- ▶ Absolute Time
- ▶ Now in Temporal Databases
- ▶ Qualitative Temporal Reasoning
- ▶ Temporal Database
- ▶ Time Span

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS, vol. 1399. Springer, Berlin, 1998, pp. 367–405.

Relevance

Jovan Pehcevski[1], Birger Larsen[2]
[1]INRIA Paris-Rocquencourt, Le Chesnay Cedex, France
[2]Royal School of Library and Information Science, Copenhagen, Denmark

Synonyms

Exhaustivity; Specificity; Topical-hierarchical relevance

Definition

Relevance is the extent to which some information is pertinent, connected, or applicable to the matter at hand. It represents a key concept in the fields of documentation, information science, and information retrieval.

In information retrieval, the notion of relevance is used in three main contexts. Firstly, an algorithmic relevance score is assigned to a search result (usually a whole document) representing an estimated likelihood of relevance of the search result to a topic of request. This relevance score often determines the order in which search results are presented to the user. Secondly, when the performance of information retrieval systems is tested experimentally, the retrieved documents are assessed for their actual relevance to the topic of request by human assessors (topic experts). A binary relevance scale is typically used to assess the relevance of the search result, where the relevance is restricted to be either zero (when the result is not relevant to the user request) or one (when the result is relevant). Thirdly, in experiments involving users (or in operational settings) a broader notion of relevance is often used, with the aim of expressing the degree to which the retrieved documents are perceived as useful in solving the user's search or work task.

In structured text (XML) retrieval, the search result is typically an XML element, and the relevance score assigned by an XML retrieval system again represents an estimated likelihood of relevance of the search result to the topic of request. However, when the results are subsequently assessed for relevance, the binary relevance scale is not sufficient, primarily due to the hierarchical relationships that exist among the elements in an XML document. Accordingly, in XML retrieval one or more relevance dimensions (each with a multi-graded relevance scale) have been used to assess the relevance of the search result.

Key Points

In traditional information retrieval experiments where whole documents are retrieved, a fairly simple notion of relevance may suffice for most purposes [1]. The challenge in XML retrieval is that the relevance assessments must capture not only whether the retrieved elements are relevant, but also how they relate to one another.

The different relevance definitions for XML retrieval have mainly been investigated by the INitiative for the Evaluation of XML Retrieval (INEX). In 2002, the INEX relevance definition comprised two relevance dimensions named *topical relevance* and *component coverage*. This relevance definition has not been used by INEX since then, partly because of the vague terminology used for the names of the two relevance dimensions, and partly because it has been subsequently shown that the INEX 2002 assessors did not fully comprehend component coverage. In 2003 and 2004, the two INEX relevance dimensions were named Exhaustivity and *Specificity*, which respectively reflect the extent that an element *covers* and is *focussed on* aspects of an information need represented by the topic of request. The two INEX relevance dimensions used four grades to assess the relevance of an element (either its exhaustiveness or its specificity): "none," "marginally," "fairly," and "highly." The grades from each dimension were then combined into a single 10-point relevance scale.

From 2005 onwards, a highlighting assessment procedure is used at INEX to gather relevance assessments for the XML retrieval topics. As a result, the INEX relevance definition was simplified such that only three Exhaustivity values were assigned to a relevant element, while the *Specificity* of the relevant element was measured on a continuous relevance scale and computed automatically as the ratio of highlighted to fully contained text. From 2006 onwards, relevance in INEX is defined only according to the notion of *Specificity*, where the continuous relevance scale is used to assess the relevance of retrieved elements.

The experience of both assessors and users is important when defining relevance in XML retrieval. An interactive track was established at INEX in 2004 to investigate the behavior of users when elements of XML documents (rather than whole documents) are presented as answers. The interactive track was run again at INEX in 2005 and 2006, comprising various tasks and different XML document collections [2].

A *topical-hierarchical* relevance definition was used by the INEX Interactive tracks in 2005 and 2006, which comprises two relevance dimensions and a five-point nominal relevance scale [2,3]. An analysis of the feedback gathered from users participating in the INEX 2005 Interactive track showed that users did not find the five-point scale to be very hard to understand [3]. Furthermore, a mapping between the five-point relevance scale (used by users) and the continuous *Specificity* scale (used by the expert assessors) can easily be established [3], which allows for a better understanding of the definition of relevance in XML retrieval.

Cross-references

► Evaluation Metrics for structured text retrieval
► Similarity and Ranking Operations
► Specificity

Recommended Reading

1. Borlund P. The concept of relevance in IR. J. Am. Soc. Inf. Sci. Technol., 54(10):913–925, 2003.
2. Malik S., Tombros A., and Larsen B. The interactive track at INEX 2006. In Proc. 5th Int. Workshop of the Initiative for the Evaluation of XML Retrievals, 2007, pp. 387–399.
3. Pehcevski J. Relevance in XML retrieval: the user perspective. In Proc. SIGIR 2006 Workshop on XML Element Retrieval Methodology, 2006, pp. 35–42.

Relevance Evaluation of IR Systems

► Effectiveness Involving Multiple Queries

Relevance Feedback

Ben He
University of Glasgow, Glasgow, UK

Synonyms

RF

Definition

The relevance feedback technique expands the initial query by taking into account relevance assessments on documents by the user. Relevance feedback techniques aim to improve the retrieval accuracy and to better satisfy the user information need.

Key Points

The relevance feedback algorithms may use explicit (e.g., user labels), implicit (e.g., click-through data) or blind (e.g., pseudo relevance documents) relevance information for the feedback purpose.

A classical relevance feedback algorithm based on the Vector Space model was proposed by Rocchio in 1971 [1].

Cross-references

► Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model
► Probabilistic Model
► Query Expansion Models
► Relevance Feedback for Content-Based Information Retrieval

▶ Rocchio's Formula
▶ Web Search Relevance Feedback

Recommended Reading

1. Rocchio J. 1Relevance Feedback in Information Retrieval. Prentice-Hall, Englewood Cliffs, NJ, USA, 1971, pp. 313–323.

Relevance Feedback for Content-Based Information Retrieval

XIN-JING WANG, LEI ZHANG
Microsoft Research Asia, Beijing, China

Definition

Relevance feedback (RF) [5,2] is an on-line approach which tries to learn the users' intentions on the fly. It leverages users to guide the computers to search for relevant documents. An RF mechanism has two components: a learner and a selector. At every feedback round, the user marks (part of) the images returned by the search engine as *relevant* or *irrelevant*. The learner exploits this information to re-estimate the target of the user. This information is used both quantitatively (retrieving more documents like the relevant documents) and qualitatively (retrieving documents similar to the relevant ones before other documents). With the current estimation of the target, the selector chooses other images that are displayed by the interface of the search engine; then the user is asked to provide feedback on these images during the next round. The process of RF is usually presented as a cycle of activity: an IR system presents a user with a set of retrieved documents; the user indicates those that are relevant; and the system uses this information to produce a modified version of the query. The modified query is then used to retrieve a new set of documents for presentation to the user. This process is known as an iteration of RF which ends until the user is satisfied.

Historical Background

RF is a powerful tool which is traditionally used in text-based information retrieval systems. It was introduced to CBIR during mid-1990s [4], with the intention of bringing the user into the retrieval loop and reducing the "semantic gap" between what queries represent (low-level features) and what the user thinks.

Early approaches assumed the existence of an ideal query point that, if found, would provide the appropriate answer to the user. These approaches belong to the family of query point movement (QPM) methods, for which the task of the learner consists in finding, at every round, a better query point together with a re-weighting of the individual dimensions of the description space.

Recent work on RF often relies on support vector machines (SVM). With SVMs, generally the data is first mapped to a higher-dimensional feature space using a non-linear transform kernel, then is classified in this mapped feature space using maximum margin strategy [6].

Foundations

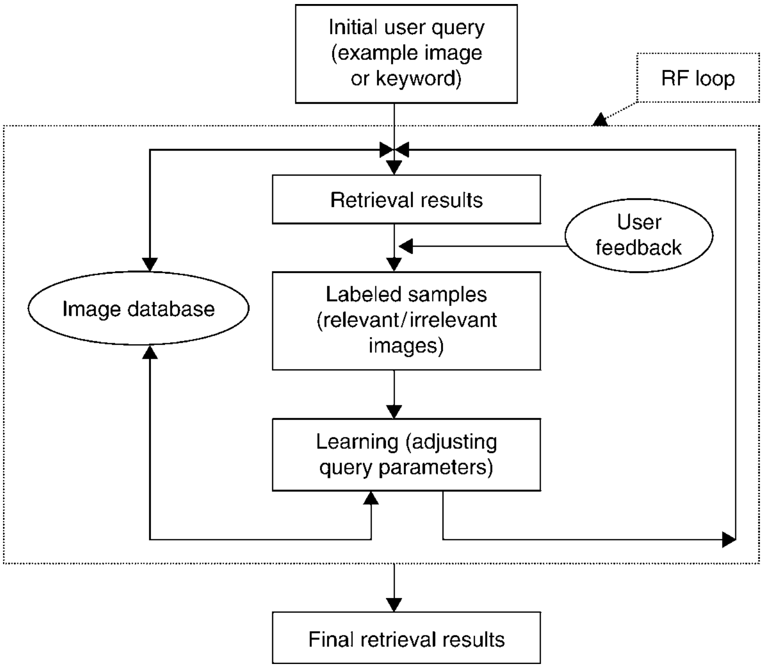
General Assumptions

Before developing relevance feedback mechanisms for content-based image retrieval, it is necessary to understand some general assumptions underlying. According to [2], the assumptions are as the following:

1. The discrimination between relevant and irrelevant images must be possible with the available image descriptors.
2. There are some relatively simple relations between the topology of the description space and the characteristics shared by the images that the user is searching for.
3. Relevant images are a small part of the entire image database.
4. While some of the early work on RF assumed that the user could (and would be willing to) provide a rather rich feedback, including relevance notes for many images, currently the assumption is that this feedback information is scarce. The user will only mark a few relevant images as positive and some very different images as negative.

Figure 1 shows the flowchart of a typical Content-based image retrieval process with relevance feedback [3]. Based on the above general assumptions, a typical scenario for relevance feedback in content-based image retrieval is as below [3,9]:

1. The system provides initial retrieval results given query examples;
2. User judges the above results as to what degree, they are relevant (positive examples) or irrelevant (negative examples) to the query.


Initial user query (example image or keyword)
RF loop
Retrieval results
User feedback
Image database
Labeled samples (relevant/irrelevant images)
Learning (adjusting query parameters)
Final retrieval results

Relevance Feedback for Content-Based Information Retrieval. Figure 1. CBIR with RF.

3. Machine learning algorithm is applied to learn a new ranking model based on the user's feedback. Then go back to (2).

Steps (2)–(3) are repeated till the user is satisfied with the results.

Step (3) is comparably the most important step and different approaches can be used to learn the new query. A few generally adopted approached are introduced in the following.

Re-Weighting Approaches

A typical approach in step (3) is to automatically adjust the weights of low-level features to accommodate the users' need, rather than asking the user to specify the weights as adopted in earlier content-based image retrieval systems. This re-weighting step dynamically updates the weights embedded in the query (not only the weights to different types of low-level features such as color, texture, shape, but also the weights to different components in the same feature vector) to model the high-level concepts and perception subjectivity [4].

Query Point Movement Approaches

Another method is called query-point-movement (QPM) [3]. It improves the estimation of the query point by moving it towards the positive examples and away from the negative examples. A widely adopted query point removing technique is called the Rocchio's formula [1] (see (1) below):

$$Q' = \alpha Q + \beta\left(\frac{1}{N_{R'}} \Sigma_{i \in D'_R} D_i\right) - \gamma\left(\frac{1}{N_{N'}} \Sigma_{i \in D'_N} D_i\right) \quad (1)$$

In (1), Q and Q' are the original query and updated query, respectively, and D'_R and D'_N are sets of the positive and negative images returned by the user, and $N_{R'}$ and $N_{N'}$ are the set sizes. α, β and γ are weights.

Machine Learning Approaches

Machine learning techniques are also widely used. As mentioned previously, support vector machine (SVM) is used to capture the query concept by firstly applying the kernel trick which projects images onto a hyperspace and then separates the relevant images from irrelevant ones using maximum margin strategy. The advantages of adopting SVM are that (i) it has high generalization ability, and (ii) it works for small training sets.

Another step-forward approach is proposed by Tong and Chang [8] called SVM active learning, which was reported to be able to effectively use negative and non-labeled samples, and learn the query concept faster and with better accuracy.

Since manually labeling images is tedious and expensive, training image set is usually very small. This is called the small sample problem. To handle this problem, some researchers proposed boosting methods, e.g., Discriminant-EM (D-EM) [7], which boosts the classifier learnt from the limited labeled training data.

Decision-tree learning methods such as C4.5, ID3 were also used in RF loop to classify the database images into relevant and irrelevant.

Key Applications

Relevance feedback approach can help not only content-based image retrieval, but also applications like image annotation, segmentation, etc.

Future Directions

There are still many research work on adopting relevance feedback to content-based image retrieval [2]:

1. It is better to exploit prior information, such as domain-specific similarity, clustering, context of session, etc., in the RF mechanisms.
2. The impact of the data and of the policy of the user on both the learner and the selector must be addressed.
3. How to scale up RF to handle very large image databases is an important issue which was not extensively studied.

Cross-references

▶ Content-Based Image Retrieval
▶ Relevance Feedback

Recommended Reading

1. Chen Z. and Zhu B. Some formal analysis of Rocchio's similarity-based relevance feedback algorithm. Information Retrieval 5:61–86, 2002.
2. Crucianu M., Ferecatu M., and Boujemaa N. Relevance feedback for image retrieval: a short survey. In state of the art in audiovisual content-based retrieval, Information Universal Access and Interaction, Including Datamodels and Languages. Report of the DELOS2 European Network of Excellence (FP6), (2004).
3. Liu Y., Zhang D., Lu G., and Ma W.-Y. A survey of content-based image retrieval with high-level semantics. Pattern Recognition 40(1):262–282, 2007.
4. Rui Y., Huang T.S., Ortega M., and Mehrotra S. Relevance feedback: a power tool for interactive content-based image retrieval. IEEE Transactions on Circuits and Systems for Video Technology 8(5):644–655, 1998.
5. Ruthven I. and Lalmas M. A survey on the use of relevance feedback for information access systems. The Knowledge Engineering Review(2003). Cambridge University Press, London.
6. Schölkopf B. and Smola A. Learning with Kernels. MIT Press, Cambridge, MA, 2002.
7. Tian Q., Yu Y., and Huang T.S. Incorporate discriminant analysis with EM algorithm in image retrieval. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2000, pp. 299–302.
8. Tong S. and Chang E. Support vector machine active learning for image retrieval. In Proc. 9th ACM Int. Conf. on Multimedia, 2001, pp. 107–118.
9. Zhu X.S. and Huang T.S. Relevance feedback in image retrieval: a comprehensive review. Multimedia System 8(6):536–544, 2003.

Relevance Feedback for Text Retrieval

Olga Vechtomova
University of Waterloo, Waterloo, ON, Canada

Synonyms

RF

Definition

Relevance feedback (RF) is a process by which the system, having retrieved some documents in response to the user's query, asks the user to assess their relevance to his/her information need. The user's relevance judgements are then used to either adjust the weights of the query terms, or add new terms to the query (query expansion).

Key Points

Searchers may have difficulties in finding the words and phrases (terms) to express their information needs accurately and completely. They may also use different words in the queries than the words used by the authors of documents. On the other hand, searchers tend to know relevant information when they see it. In other words, it may be easier for them to tell which documents are relevant, instead of formulating a detailed query.

A typical relevance feedback process consists of the following steps: the user formulates and submits an initial query to an information retrieval system, which retrieves a ranked list of documents. Documents are typically presented in the ranked list in some surrogate form, for example, as document titles, abstracts, snippets of text, query-biased, or general summaries. The user then reads the contents of the documents that seem promising. After reading each document, the user is expected to indicate to the system whether it is relevant or not to his/her information need. After the user has judged some documents as relevant, either the system, or the user initiate the process of query expansion or query term reweighting. The relevance feedback process can be iterative.

When relevance feedback is used for reweighting, the weights of the user-entered query terms are adjusted based on their occurrence in the judged documents. For example, in the probabilistic model [2], terms are reweighted based on their presence in the documents judged relevant to give a more accurate estimation of the probability of the document's relevance to the query.

Relevance feedback has consistently yielded substantial gains in performance in experimental settings when used for query expansion. In this case, documents judged relevant by the searcher are used as the source of new terms to be added to the query. Many term selection methods have been proposed for query expansion following relevance feedback. The general idea behind all such methods is to select terms that discriminate between relevant and nonrelevant documents, and are useful in retrieving previously unseen relevant documents, e.g., [1,2].

A related process is known as pseudo-relevance or blind feedback (BF). The difference from RF is that instead of asking the user to indicate the relevance of a document, the system assumes that *n* top ranked documents are relevant. An advantage of BF over RF is that it requires less cognitive effort on the part of the user. The performance of BF, however, depends on the performance of the user's query. If the documents in the top ranks are nonrelevant, then query expansion or query term reweighting will deteriorate the performance.

Another technique is implicit relevance feedback (IRF), where the system infers the user's interest in the documents based on his/her behavior with respect to the IR system. A detailed study analyzing the utility of IRF was conducted by [4].

The adoption of relevance feedback in its traditional form by the web search engines has been limited. A study of the use of relevance feedback in the Excite search engine is reported in [3].

Cross-references

- ▶ Information Retrieval
- ▶ Information Retrieval Models
- ▶ Query Expansion for Information Retrieval

Recommended Reading

1. Carpineto C., de Mori R., Romano G., and Bigi B. An information-theoretic approach to automatic query expansion. ACM Trans. Inf. Syst., 19(1):1–27, 2001.
2. Spärck Jones K., Walker S., and Robertson S.E. A probabilistic model of information retrieval: development and comparative experiments. Inf. Process. Manage., 36(6):779–808 (Part 1); 809–840 (Part 2), 2000.
3. Spink A., Jansen B.J., and Ozmultu H.C. Use of query reformulation and relevance feedback by Excite users. Internet Res.: Electron. Networking Appl. and Policy, 10(4):317–328, 2000.
4. White R.W., Ruthven I., and Jose J.M. A study of factors affecting the utility of implicit relevance feedback. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 35–42.

Relevance Propagation

▶ Propagation-Based Structured Text Retrieval

Remote Method Invocation

▶ RMI

Removing Overlap

▶ Processing Overlaps

Rendezvous

▶ Workflow Join

Replica and Concurrency Control

▶ Replicated Database Concurrency Control

Replica Consistency

▶ Consistency Models For Replicated Data

Replica Control

Ricardo Jimenez-Peris, Marta Patiño-Martinez
Universidad Polytecnica de Madrid, Madrid, Spain

Synonyms

Database replication; Data replication protocols; Cluster database replication

Definition

In database replication, each data item has several physical copies, also called replicas, that are distributed over different nodes (sites). In case of full replication, each data item has a copy on each site. In this case, the term replica can also refer to a node hosting a copy of the entire database. *Replica control* is in charge of translating the read and write operations that clients submit on the logical data items into operations on the physical data copies. The goal is to keep a consistent state among all the replicas and to provide a consistent view of the data to the client. Replica control extends concurrency control in order to coordinate the execution of concurrent transactions at different replicas.

Historical Background

Replica control in databases has been studied since the 1980's. Early approaches mostly explored distributed locking and had their main focus on providing high availability and enforcing strong consistency in the advent of site failures and network partitions [3]. In particular, the correctness criterion 1-copy-serializability requires the execution in a replicated system to be equivalent to a serial execution over a non-replicated database.

In the 1990's a seminal paper from Gray et al. [7] criticized existing database replica control protocols providing analytical evidence of the lack of scalability due to the high deadlock probabilities when the number of replicas increases. This paper raised a new wave of research around data replication searching for scalable approaches. Some of these approaches explore weaker levels of consistency (e.g., [2,6,15]). Others aim at maintaining strong levels of consistency while still offering good scalability (e.g., [1,8,11]), sometimes relying on powerful abstractions such as group communication. Snapshot isolation has been explored as a new isolation level [5,9,14] to allow for more concurrency in the replicated system. A considerable effort has been undertaken to optimize those parts of replication control that have a large impact on the performance. Both kernel based replication solutions (e.g., [8,11] as middleware-based replication tools (e.g., [1,9,14] have been developed. Different solutions have been developed for LAN and WAN environments.

All these efforts have led to a wide range of replica control protocols. Which one to choose for a specific scenario depends on the environment and the application requirements. A trade-off between correctness, performance, generality, and potential of scalability is nearly always unavoidable.

Foundations

Replica control algorithms and their implementations can be categorized by a wide range of parameters.

Architecture

One aspect of replica control is *where* the protocol is implemented. It can be implemented within the database what is known as *kernel-based* or *white box* approach (e.g., Postgres-R [8] and the database state machine [11]). A client connects to any database replica which then coordinates with the other replicas. Typically, replica control is tightly coupled with the concurrency control mechanism of the database system.

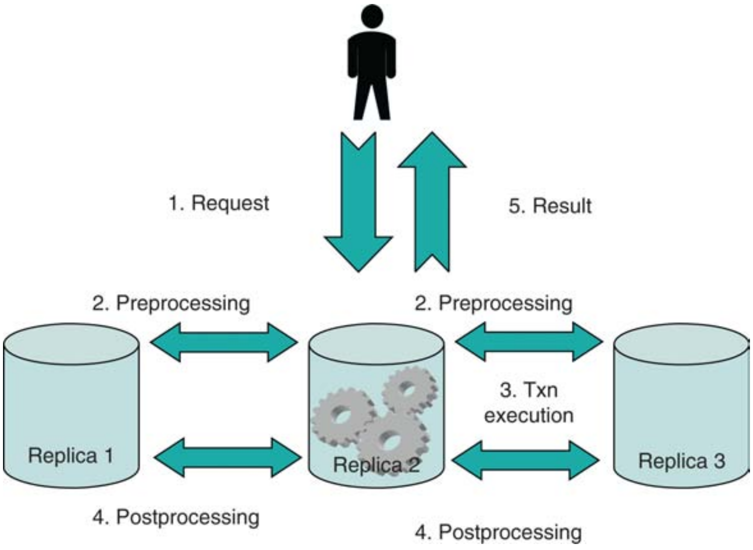
Alternatively, replica control can be implemented outside the database as a *middleware* layer (e.g., [1,9,10,14,15]). Clients connect to the middleware that appears as a database system. The middleware then controls the execution and directs the read and write operations to the individual database replicas. This can be instrumented in two ways: (i) A *black-box approach* uses standard database systems to store the database replicas [1,14,15]; (ii) A *gray-box* approach expects the database system to export some minimal functionality that can be used by the middleware for a more efficient implementation of replica control, e.g., providing the tuples that a transaction has updated so far in form of a writeset [10]. A middleware-based approach typically has its own concurrency control mechanism which might partially depend on the concurrency control of the underlying database systems. There might be a single middleware component (centralized approach), or the middleware might be replicated itself. For example, the middleware could have a backup replica for fault-tolerance. Other approaches have one middleware instance per database replica, and both together build a replication unit.

Replica Control Phases

A replica control protocol has to control the execution of transactions during regular operation (no failure), while failures occur, and during recovery (failed sites with stale data replicas rejoin the system or new sites are added to the system). As shown in Fig. 1, the execution of a transaction during regular operation may have a number of phases (adjusted from [12]):

1. *Client connection*. The client interacts with the database through a client proxy (such as a JDBC or an ODBC driver, or the database client API). The client connects to the replicated database by invoking the client proxy connection method. In the middleware approach, the client connects to the middleware (or one of the middleware instances if they are replicated). In the kernel-based approach, it connects to one of the database replicas. Connecting to a replicated database may require some mechanism to enable *replica discovery* in a transparent way (such as a well-known registry or IP multicast). With this, the client can connect to the database independently of which replicas are currently available. If transparency is required, i.e., the client application should not be aware of replication, then replica discovery can be hidden in the client proxy.
2. *Request submission*. The client submits a request via the client proxy that forwards it to the middleware or database replica to which it is connected.
3. *Pre-processing coordination*. Some protocols require sending requests to all replicas. As a result, the replica to which the client is connected might forward the request to one, some or all replicas. This phase may also be used for other purposes such as distributed concurrency control or load-balancing.
4. *Request processing*. The request is processed by one or more replicas.
5. *Post-processing coordination*. Once a request is processed some protocols perform concurrency control tasks, propagate changes, aggregate results from a quorum, or guarantee atomicity by executing a two-phase commit protocol or another termination protocol.
6. *Result return*. The result of the request is sent back to the client proxy that returns it to the client.

A request can be either an entire transaction (consisting of one or multiple read and write operations), or an operation within a transaction. In the latter case, phases 2–6 may be repeated for each operation and for the final commit of a transaction (as in distributed locking [3]). Moreover, the order of the last two phases could be reversed, i.e., first a result is sent to the client and after that, some coordination takes place.



Replica Control. Figure 1. Replica control phases.

Replica control protocols are implemented in the preprocessing phase, in the post-processing phase or in both of them. five phases are independent of the architecture of the replicated database (replica control is implemented as a middleware layer, either centralized or distributed, or is a kernel based implementation).

Mapping Approaches

One of the main tasks of replica control is to map read and write operations on logical data items to operations on the physical data copies such that replicas eventually converge to the same value and reads see consistent data. With ROWA (read-one-write-all) protocols, read-only operations are processed at a single replica, while write operations are executed at all replicas. The extension to ROWAA (read-one-write-all-available) protocols aims at handling site failures. However, in case of network partitions ROWAA might result in two partitions executing transactions. In contrast, quorum-based replica control can handle both site and network failures. Both read and write operations need to access a quorum of replicas. Any two write quorums, and each read and write quorum have to overlap in at least one replica in order to guarantee data consistency and consistent read operations.

Correctness Criteria

One of the main aspects of replica control is the correctness criterion to be supported. Several, relatively strong correctness criteria extend the notion of isolation from a non-replicated database to a replicated setting. *1-copy correctness* states that the replicated data should appear as one logical non-replicated database. Depending on the isolation level used in the non-replicated execution different correctness criteria can be defined. In a non-replicated database, *serializability* guarantees that the concurrent execution of transactions is equivalent to a serial execution. *1-copy-serializability* [3] (1CS) extends this notion and guarantees that the concurrent execution of transactions over the replicated database is equivalent to a serial execution over a non-replicated database. Non-replicated database systems usually offer more relaxed forms of isolation, such as the ANSI isolation levels or snapshot isolation. 1-copy-snapshot-isolation has been defined as a correctness criterion for a replicated database (e.g., [9], and several replica control protocols consider snapshot isolation (e.g., [5,9,14]).

Weak consistency models do not require 1-copy correctness. Depending on the consistency level, clients might read replicas that are stale, that is, have not yet applied the latest updates, or copies might even diverge if different replicas are allowed to concurrently apply conflicting updates [16]. The degree of staleness or divergence might be bound.

Concurrency Control

In cases where *1-copy* correctness is required, distributed concurrency control is needed to enforce the correctness criteria by restricting the execution of concurrent conflicting transactions. That is, many replica control protocols extend concurrency control to a replicated system.

Concurrency control protocols can be either optimistic or pessimistic. A *pessimistic* approach restricts concurrency to enforce consistency across replicas. The easiest way consists in executing all update transactions sequentially in the same order at all sites, what would provide 1CS trivially. Other protocols increase concurrency by exploiting knowledge about the data that will be accessed. With this information, transactions accessing disjoint data sets can be executed in parallel while those that potentially access common data are executed sequentially. The granularity of data can be at different levels, such as tables, tuples or conflict classes (i.e., data partitions). Pessimistic replica control protocols are implemented in the pre-processing phase. *Optimistic* approaches, in contrast, execute potentially conflicting transactions concurrently. Only when the transaction has completed execution, a validation phase (also known as certification) takes place. It checks whether the transaction being validated conflicts with concurrent transactions. If there is a conflict, some transaction must be aborted. A standard mechanism to guarantee serializability is to abort the validating transaction if the set of data items it read during execution overlaps with the set of data items written by a concurrent transaction that already validated. With snapshot isolation, a transaction fails validation if it wrote some data item that a concurrent, already validated transaction also wrote.

Processing Update Transactions

Another important feature of replica control is how update transactions are processed. With *symmetric update processing* each update transaction is fully executed at all replicas. In contrast, *asymmetric update* *processing* executes update transactions at one site (or subset of sites) and then, the resulting changes (known as *writeset*) are propagated to the rest of the replicas. Some protocols [1] lie in between, being symmetric at the statement level, but asymmetric at the transaction level. That is, if an update transaction contains both write and read statements, the read statements are executed at one site while write statements are executed at all sites. Asymmetric update processing has typically much less overhead than symmetric update processing and thus, allows for better scalability in update-intensive environments [8].

Timepoint of Synchronization

As described earlier, a client typically connects to one replica and submits its transactions to this replica. An important question is *when* this replica coordinates with other replicas to guarantee data consistency. In *eager* (aka synchronous) protocols, coordination takes place before the transaction commits locally. Typically, this means that the replica sends the changes (asymmetric processing) or the operation request (symmetric processing) and the concurrency control component decides on a serialization order for this transaction before it commits. In contrast, with *lazy replication* (aka asynchronous), updates are asynchronously propagated after the transaction commits. Lazy replication usually applies asymmetric processing. The propagation can be done immediately after the commit, periodically, or be triggered by some weak consistency criteria such as freshness. With lazy replication, transaction execution has usually no preprocessing coordination phase, and phases five and six are switched. This means, the results are first returned to the client, and then, the post-processing phase is run. Eager replication usually results in longer client response times since communication is involved but can more easily provide strong consistency.

Who Executes Transactions

Primary copy replication requires that all update transactions are executed at a given site (the primary) (e.g., [5,14,15]). The primary propagates the changes to the other replicas (secondary replicas). Secondaries are only allowed to execute read-only transactions themselves. In order to be able to forward read-only transactions to secondaries and update transactions to the primary, the system must be aware at the start time of a transaction whether it is read-only or not (e.g., through a tagging mechanism). If update transactions can be executed at any site, the replica control protocol follows an *update everywhere* approach (also referred to as *update anywhere*). An alternative to primary copy replication and update everywhere is based on partitioning the database items such that each partition has a primary, but different partitions may have different primaries (e.g., [4,10]). This avoids that the primary becomes a bottleneck under write-intensive workloads. However, it is only suitable for applications where the database can be partitioned such that each transaction only accesses data of one partition.

Degree of Replication

A further dimension is the *degree of replication*. In *full replication* every data item is replicated at each site. At the other extreme, in a *distributed database* each data item is stored at only one site and there is no replication. In partial replication each data item is replicated at a subset of nodes.

Coordination Steps

Another aspect considers the number of coordination steps, and thus, the number of message rounds, per transactions. Some protocols use a constant number of message rounds, while others require a linear number of message rounds, depending on the number of (write) operations within the transaction (e.g., in distributed locking). In the former case, the pre-processing and/or the post-processing phases are executed once for each transaction. In the latter case, these phases are executed per (write) operation.

Furthermore, some replica control protocols require a coordination protocol among the replicas in order to decide the outcome of a transaction (*voting termination*), similar to a distributed commit protocol. In others, each replica decides by itself deterministically about the outcome of a transaction (*non-voting termination*).

In early eager approaches, atomicity was achieved by running a commit protocol, such as two-phase-commit, at the end of transaction. This was not only time-consuming by itself but also required that all sites had completely executed the transaction before the transaction was committed at any site. In more recent approaches, atomicity is often achieved by other means such as reliable multicast that provides the required failure atomicity. Even in eager approaches where the participating sites agree on a serialization order before the transaction commits at any site, this allows the user to receive the commit outcome before the transaction is actually executed at all replicas.

Restrictions

Another point to consider when analyzing replica control protocols are the possible constraints they set on the kind of transactions that are supported. Some protocols only allow single statement transactions (known as auto-commit mode in JDBC). Other protocols allow several statements within a transaction, but they have to be known at the beginning of the transaction. This is typically implemented using stored procedures (or prepared statements in JDBC) [1,10]. The more general protocols do not have any restriction on the number of statements a transaction contains [8,9,14].

Other Aspects

Replica control also needs to work correctly in the case of failures and when new or repaired replicas are added to the system. Replica control can also be affected by the implementation of self-* properties or autonomic behavior.

Another crucial feature of replica control is the environment for which it is designed. Many approaches target local area networks (LAN) where the bandwidth is high and partitions are rare. Some approaches have explored how to attain acceptable response times in wide area networks (WANs). Other kinds of networks have also been explored such as mobile networks and peer-to-peer networks. There are specific entries for all them.

Key Applications

Replica control is always needed if replicated data is updated. In particular, in database replication, where updates occur in the context of transactions, replica control is also needed to guarantee transactional correctness.

Future Directions

An important future direction is to combine replica control with modern multi-tier and service-oriented architectures. Providing high availability and scalability for a multi-tier architecture is a non-trivial task. It implies replicating all the tiers to avoid single points of failure and performance bottlenecks. In this setting, guaranteeing consistency is very challenging, especially in the face of failures. Some initial work has already been started in this direction [13] and future developments are expected along this line.

Cross-references

► Autonomous Replication
► Online Recovery in Parallel Database Systems
► Optimistic Replication and Resolution
► Replication Based on Group Communication
► Replication for High Availability
► Replication for Scalability
► Strong Consistency Models for Replicated Data
► Traditional Concurrency Control for Replicated Databases
► WAN Data Replication

Recommended Reading

1. Amza C., Cox A.L., and Zwaenepoel W . Distributed versioning: consistent replication for scaling back-end databases of dynamic content web sites. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2003, pp. 282–304.
2. Bernstein P.A., Fekete A., Guo H., Ramakrishnan R., and Tamma P. Relaxed-currency serializability for middle-tier caching and replication. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 599–610.
3. Bernstein P.A, Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison Wesley, 1987.
4. Breitbart Y., Komondoor R., Rastogi R., Seshadri S., and Silberschatz A. Update propagation protocols for replicated databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 97–108.
5. Daudjee K. and Salem K. Lazy Database Replication with Snapshot Isolation. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 715–726.
6. Gançarski S., Naacke H., Pacitti E., and Valduriez P. The leganet system: Freshness-aware transaction routing in a database cluster. Inf. Syst., 32(2):320–343, 2007.
7. Gray J., Helland P., O'Neil P., and Shasha D. The Dangers of Replication and a Solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
8. Kemme B. and Alonso G. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 134–143.
9. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Middleware based data replication providing snapshot isolation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 419–430.
10. Patiño-Martínez M., Jiménez-Peris R., Kemme B., Alonso G. MIDDLE-R: Consistent database replication at the middleware level. ACM Trans. Computer Syst., 23(4): 375–423, 2005.

11. Pedone F., Guerraoui R., and Schiper A. The Database State Machine Approach. Distrib. Parall. Databases, 14(1), 2003.
12. Pedone F., Wiesmann M., Schiper A., Kemme B., and Alonso G. Understanding replication in databases and distributed systems. In Proc. 20th Int. Conf. on Distributed Computing Systems, 2000, pp. 464–474.
13. Perez-Sorrosal F., Patiño-Martínez M., Jiménez-Peris R., and Kemme R. Consistent and scalable Cache replication for multi-tier J2EE applications. In Proc. ACM/IFIP/USENIX 8th Int. Middleware Conf., 2007, pp. 328–347.
14. Plattner C. and Alonso G. Ganymed: scalable replication for transactional web applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004, pp. 155–174.
15. Röhm U., Böhm K., Schek H.-J., and Schuldt H. FAS – A Freshness-Sensitive Coordination Middleware for a Cluster of OLAP components. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 754–765.
16. Saito Y. and Shapiro M. Optimistic replication. ACM Computer Surveys 37(1):42–81, 2005.

Replica Freshness

Alan Fekete
University of Sydney, Sydney, NSW, Australia

Synonyms

Divergence control; Freshness control; Incoherency bounds

Definition

In a distributed system, information is often *replicated* with copies of the same data stored on several sites. Ideally, all copies would be kept identical, but doing this imposes a performance penalty. Many system designs allow replicas to lag behind the latest value. For some applications, it is acceptable to use out-of-date copies, provided they are not too far from the true, current value. *Freshness* refers to a measure of the difference between a replica and the current value.

Historical Background

The tradeoff between consistency and performance or availability is an old theme in distributed computing. In the database community, many researchers worked on ideas connected with explicitly allowing some discrepancy between replicas during the late 1980s and early 1990s. Early papers identified many of the diverse freshness measures discussed here, from groups at Princeton, Bellcore and Stanford [1,10,11]. A mixed model that integrated freshness limits of several kinds was Pu's epsilon-serializability [6,7]. TACT is a more recent mixed model, introduced by Yu and Vahdat [12]. Much of the research since the 1990s focused on optimization decisions to improve the performance of a system with slightly stale replicas. Many different system assumptions and metrics to optimize have been considered. Among influential papers are [4,5,9]. Research continues on defining different models which bound the staleness of replicas. Unlike earlier work, the focus has recently been on setting bounds which relate to the clients' view of divergence, rather than to the underlying state of the replicas. Röhm et al. designed a system with transaction-level client-defined staleness limits for read-only transactions, within the framework of 1-copy serializability [8]. Guo et al. suggested SQL extensions to express query-specific limits on the perceived staleness and inter-object drift [3], and later developed a theory to express these constraints even when stale reads are allowed within update transactions [2].

Foundations

Consider a distributed system, where information is stored at multiple sites, connected by a communications network. If several sites all store values that are intended to represent the same information in the real world, one can call these copies or *replicas*; it is usual to write x^A to represent the replica at site A of the logical data item x. At any instant, the true or current value of the logical item is the value assigned to x in the most recent non-aborted update of x. The ideal, of course, has this value stored in every replica, always (or at least, at any instant when a read operation can occur). This is possible using traditional eager replication (see entry on Traditional Concurrency Control for Replicated Databases). However, many systems prefer to propagate updates lazily, and to allow replicas or cached copies that are not completely up-to-date, as described in the entry on Optimistic Replication and Resolution. While there are many applications that can tolerate data that is not current, there are usually limits on the application's tolerance for inaccuracy. Thus many system designs allow for a bound to be placed on how far a replica can diverge from the true value; such a bound expresses a constraint on the *freshness* of the replica. This entry describes some of the main ideas that have been proposed to quantify the freshness of replicas.

The focus here is on the properties that define what is allowed, rather than on implementation details that control how these bounds are enforced. For example, some system designs have a single master replica, which always has the true value, while in others the most up-to-date information is sometimes found in one copy and sometimes in another. However, the definitions can be stated without concern for this issue. Another axis of variation is whether the bounds have to apply always, or only when a replica is used in a query. Some papers give a numeric value for freshness or precision (which should be high), others measure staleness or imprecision. This paper uses measures where a low value is better, so zero means that the replica is completely up-to-date.

Value-Based Divergence

Suppose the logical data item comes from a numeric domain such as real numbers. In that case, one can just use the metric on the values themselves. For example, if the true current value is 10.5 and a replica has the value 9.2, then the divergence of that replica is measured as 1.3. In some proposals, the interest is in the value-based divergence between two replicas, even if neither has the current true value. If each replica is within δ of the true value, the difference between any pair of replicas is bounded above by 2δ.

Delay-Based Staleness

For some applications, a useful measure of tolerance for imprecise data is to quantify how recently the data was correct. For example, when a person moves house, the post-office will often redirect mail that had the former address, for a short period. Thus staleness can be measured by how long the replica has to wait before learning of an update that has occurred. For example, suppose the true value was 9.2 until time 100, and which time the value was updated to 10.5; if at time 103 a replica still contains 9.2, one says it is stale by 3 time units. Where the values are representing a real-world quantity, and the quantity can't change by more than v units each time interval, then a bound of δ on the delay implies a bound of $v\delta$ on the value divergence.

Many system designs can't determine, at a replica, how stale it is. Instead they keep a timestamp with each value, indicating when that update instruction was first applied. If they bound the difference between the timestamp and the current time, they also bound the staleness. Suppose one wants to keep delay below 5 time units; one can say that this holds at time 103, if the replica contains a value whose timestamp is 98 or greater. However, the argument does not work in reverse: even at time 103, a replica with timestamp 93 might be stale by less than $103 - 93 = 10$, depending on when the next update occurred after time 93. One says that a value has a valid period, which is a half-closed interval from the time at which the value first appeared in an update, until just before the next update occurred which changed the value.

Measures of Missed Updates

In many system designs, only a subset of updates are propagated to the replica (in order to save bandwidth). This motivates a definition where one measures how many updates occurred on the logical item, without being recorded (yet) at the replica. For example, suppose the logical item is updated from 9.2 to 10.5 at time 100, and then to 10.1 at time 102, and then to 9.6 at time 104. A replica with value 9.2 at time 103 has missed 2 updates that have occurred by that time. This definition can be related to a delay-based measure if the updates come periodically, as is common for sensor readings.

Inter-Object Consistency Drift

When a query reads several logical objects, one might bound the drift between the versions seen in the two reads. For example, if a query examines copies of the temperature and humidity from a sensor, the user might care that the two measurements were taken at almost the same time, because the humidity affects the accuracy of the thermometer. Suppose a replica x^A contains a value whose valid period is the interval $[T,U)$ and y^B contains a value valid in the interval $[V, W)$. Measure the drift between the replicas, as the smallest separation between the valid intervals; this is zero if $[T,U) \cap [V,W) \neq ø$, and otherwise it is $min(|V - U|, |T - W|)$. A drift bound of zero is called a "snapshot" condition, as all the data examined must come from a common state of the database.

Transaction Semantics

Often, each update and each query is a separate operation. However, several updates or several queries can be placed in a single transaction. In 1-copy serializability (q.v.), all the operations in a transaction have to appear to take place together, but a transaction with reads may be serialized in the past, and so one might desire a common maximum delay for all the reads done in that transaction. A different approach is taken in epsilon-serializability. Here one ascribes a numeric measure to each of the various situations of divergence between the value read and the correct value, and adds up the divergence measure seen in each read, there is then a bound on the total divergence accumulated during the reads in a given query transaction.

Mixed Measures

Each of the definitions of freshness seems to work for some applications but not for others. Thus the TACT model has separate bounds of each measure which must be applied to all the logical data items within a user-specified grouping of items. For example, a particular logical item might require that whenever it is read, it is separated from the true value by no more than 1.5 in numeric value, it is stale by no more than 10 in delay, and it must be missing no more than 3 updates done at the master.

Key Applications

The commercial database platforms generally offer best-effort update propagation, rather than giving applications guaranteed bounds on freshness. Thus the properties described here are usually found in research prototypes or special-purpose data management, for example in cache management for web content or sensor data.

Future Directions

The ideas of replica freshness reappear in new domains, where applications might be able to tolerate some imprecision and where there is a high cost to keeping data up-to-date. For example, somewhat stale data might be used in a sensor network, or delivered in dynamic web page content. Many of these ideas are being generalized, in a broad research agenda that deals with uncertain or imprecise data.

Cross-references

- ▶ Data Acquisition
- ▶ Data Replication
- ▶ Real-Time Transaction Processing
- ▶ Sensor Networks
- ▶ Uncertainty Management in Scientific Database Systems

Recommended Reading

1. Alonso R., Barbará D., and Garcia-Molina H. Data caching issues in an information retrieval system. ACM Trans. Database Syst., 15(3):359–384, 1990.
2. Bernstein P.A., Fekete A., Guo H., Ramakrishnan R., and Tamma P. Relaxed-currency serializability for middle-tier caching and replication. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 599–610.
3. Guo H., Larson P.-Å., Ramakrishnan R., and Goldstein J. Relaxed currency and consistency: how to say "good enough" in SQL. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 815–826.
4. Olston C., Loo B.T., and Widom J. Adaptive precision setting for cached approximate values. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 355–366.
5. Pacitti E., Coulon C., Valduriez P., and Özsu M.T. Preventive replication in a database cluster. Distrib. Parall. Databases, 18(3):223–251, 2005.
6. Pu C. and Leff A. Replica control in distributed systems: as asynchronous approach. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991, pp. 377–386.
7. Ramamritham K. and Pu C. A formal characterization of epsilon serializability. IEEE Trans. Knowl. Data Eng., 7(6):997–1007, 1995.
8. Röhm U., Böhm K., Schek H.J., and Schuldt H. FAS – a freshness-sensitive coordination middleware for a cluster of OLAP components. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 754–765.
9. Shah S., Ramamritham K., and Shenoy P.J. Resilient and coherence preserving dissemination of dynamic data using cooperating peers. IEEE Trans. Knowl. Data Eng., 16(7): 799–812, 2004.
10. Sheth A.P. and Rusinkiewicz M. Management of interdependent data: specifying dependency and consistency requirements. In Proc. Workshop on the Management of Replicated Data, 1990, pp. 133–136.
11. Wiederhold G. and Qian X. Consistency control of replicated data in federated databases. In Proc. Workshop on the Management of Replicated Data, 1990, pp. 130–132.
12. Yu H. and Vahdat A. Design and evaluation of a conit-based continuous consistency model for replicated services. ACM Trans. Comput. Syst., 20(3):239–282, 2002.

Replicated Database Concurrency Control

Bettina Kemme
McGill University, Montreal, QC, Canada

Synonyms

Replica and concurrency control

Definition

In a replicated database, there exist several database servers each of them maintaining a (partial) copy of the database. Thus, each logical data item of the database has several physical copies or replicas. As transactions submit their read and write operations on the logical data items, the *replica control* component of the replicated system translates them into operations on the physical data copies. The *concurrency control component* of the replicated system controls the execution order of these operations such that the global execution obeys the desired correctness criteria. For a tightly coupled system with strong consistency requirements, 1-copy-serializability is the standard correctness criterion, requiring that the concurrent execution of transactions on the replicated data has to be equivalent to the serial execution of these transactions over a logical copy of the database. Thus, a global concurrency control strategy is needed. For loosely coupled systems with weak consistency requirements, each database server uses its local concurrency control mechanism. Inconsistencies are only resolved later via reconciliation techniques.

Key Points

Global concurrency control strategies for replicated databases can be centralized or distributed. In a centralized solution, there is a single or main concurrency control module which could be installed on one of the database servers or in an independent component (e.g., a middleware layer between clients and the individual database servers). This single scheduler decides on the global execution order. Using this architecture, standard concurrency control methods found in non-replicated database systems can be easily extended to the replicated environment. However, the single module becomes a single point of failure, and might lead to considerable message overhead. In a distributed solution, each database server has its own concurrency control component, and the different components have to work in a coordinated fashion in order to guarantee the correct global execution order.

The standard strict two-phase-locking (2PL) protocol can be extended very easily to a replicated system. In a centralized solution, there is no real difference to a non-replicated database. Before an operation on a data item is issued, a lock for this data item has to be acquired and all locks are released at commit time. In a distributed solution, each local lock manager performs strict 2PL. The difference is that locks are now set on the physical copies and not on the logical data item. As many replica control algorithms follow the read-one-copy/write-all-copies strategy it means that a write operation on a data item will lead to write locks on all database servers holding a copy of the data item. With this, a distributed deadlock might occur that involves only a single data item.

Optimistic concurrency control mechanisms are attractive in a replicated database system since they allow a transaction to be executed first locally at only one database server. Only at commit time, the modified data items are propagated to the other replicas and a validation phase checks whether a proper global execution order can be found. This keeps the communication overhead low.

In general, communication is an important issue in a replicated database. An option is to take advantage of advanced communication primitives, e.g., multicast protocols that provide delivery guarantees and a global total order of all messages. As all available database servers receive all messages in the same total order, this order can be used as a guideline to determine at each site locally the very same global serialization order.

Cross-references

- ► Concurrency Control – Traditional Approaches
- ► Data Replication
- ► One-Copy-Serializability
- ► Replica Control
- ► Replication Based on Group Communication
- ► Serializability
- ► Traditional Concurrency Control for Replicated Databases

Recommended Reading

1. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency control and recovery in database systems. Addison Wesley, Reading, MA, 1987.
2. Carey M.J. and Livny M. Conflict detection tradeoffs for replicated data. ACM Trans. Database Syst., 16(4):703–746, 1991.
3. Wiesmann M. and Schiper A. Comparison of database replication techniques based on total order broadcast. IEEE Trans. Knowl. Data Eng., 17(4):551–566, 2005.

Replication

Kenichi Wada
Hitachi Limited, Tokyo, Japan

Synonyms

Data Replication; Duplication

Definition

Replication (or data replication) is the process to make copy of a collection of data. Copied data made by replication is referred as replica (or data replica).

Replication can be categorized by data type to be copied. Most DBMSs support database replication for high availability or parallel transaction processing. Some filesystems support filesystem replication for high availability or workload distribution. Filesystem replication is commonly used in distributed filesystems. Volume replication can be processed by Logical Volume Manager, Filesystem or Storage system.

Most replication techniques try to keep replicas consistent and updated so that in case of system failures, the replica can be used to recover from the failure with minimal data loss.

Key Points

During data replication, original data is usually copied on an ongoing basis. The goal of this data replication is to provide high availability. There are two types of data replication in terms of data copy method: synchronous and asynchronous. Synchronous replication guarantees, "zero data loss" by means of atomic write operations (i.e., a write either completes on both sides or not at all). A write is not complete until there is acknowledgment for both original and replica. Most applications wait for a write transaction to complete before proceeding with further work, so synchronous replication may lower overall performance considerably.

In asynchronous replication, a write is complete as soon as the original is updated. The replica is usually updated after a small lag. This decreases write costs, but "zero data loss" is not guaranteed.

Data replication can be categorized by the layer of doing it. In case of Server-based replication, some entity (e.g., DBMS, Filesystem, Logical Volume Manager, and Application) in a server performs replication. Most Database replication and filesystem replication falls into this category.

Appliance-based replication is another category. Some appliance in a network connecting servers and storage systems makes replication. Most storage virtualization appliances have volume replication functions. File virtualization appliances may provide filesystem replication.

Storage-based replication is done by storage system. Block storage can provide volume replication and file storage such as NAS can provide filesystem replication.

Cross-references

▶ Backup and Restore
▶ Data Replication

Replication Based on Group Communication

FERNANDO PEDONE
University of Lugano, Lugano, Switzerland

Definition

Database replication based on group communication encompasses protocols that implement database replication using the primitives available in group communication systems. Most commonly, these replication protocols ensure strong data consistency (e.g., serializability, snapshot isolation), which translates into ordering requirements on the data operations. Protocols not based on group communication primitives should implement this ordering from scratch, typically in some ad hoc manner. Implementing ordering guarantees is not an easy task, notably in the presence of failures since it is usually difficult to determine the state a failed site was in when the crash occurred. Group communication primitives offer message reliability and ordering properties that simplify the design of replication protocols. As a consequence, database replication protocols based on group communication are usually highly modular.

Historical Background

Group communication was introduced in the 1980s. The Isis system pioneered work on group communication primitives that provide strong guarantees in the presence of failures [4], a concept that builds on the earlier abstraction of process groups to support one-to-many communication. Database replication based on group communication was introduced in the 1990s.

To understand how group communication influenced database replication, the first step is to provide a brief account of how group communication is related to replication in distributed systems in general. Many fault-tolerant distributed systems aiming at strong consistency are implemented according to the state-machine replication approach [10,15], sometimes called active replication. (Notice that while [15] extends state-machine replication to the context of failures, the concept was introduced in [10] for environments in which failures could not occur.) Essentially, the state-machine replication approach can be decomposed into two requirements, which orchestrate the dissemination of operations to replicas:

Agreement. Every non-faulty replica receives every operation.
Order. Every non-faulty replica processes the operations it receives in the same relative order.

Assuming that each replica can be made to behave as a state machine, when provided with the same operations in the same order, all replicas will produce the same results. The problem is then how to ensure the agreement and order properties. It turns out that group communication systems provide primitives that guarantee these properties. Thus, it comes perhaps as no surprise that group communication has become a basic building block for implementing state-machine replication, and fault-tolerant distributed systems.

Two observations from the mid-1990s help understand how group communication came into play in database replication. First, it was pointed out that transactions in replicated databases share common properties with group communication primitives [14]. For example, transaction isolation translates into ordering requirements among requests, a property typically available in group communication systems. Likewise, transaction atomicity is usually referred to as agreement in group communication systems. Second, it was shown that mechanisms typically used to provide strong consistency in distributed databases are inappropriate in replicated scenarios. For example, distributed two-phase locking has an expected deadlock rate that grows with the third power of the number of replicas [7]. The underlying argument from these results follows from the fact that out of order requests may get entangled, increasing the chances of deadlocks, a shortcoming that group communication primitives can easily solve.

As a consequence, researchers started looking at database replication protocols based on primitives that provide guarantees similar to those found in transactional systems.

Foundations

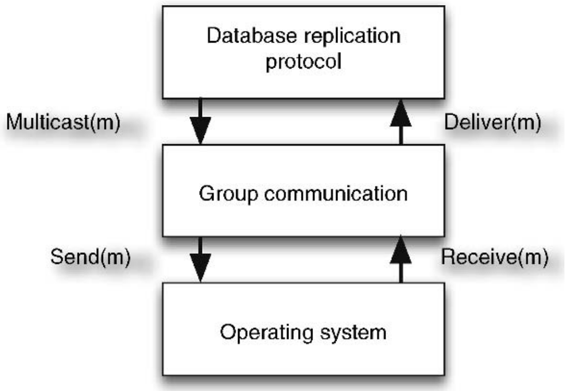
How can group communication help design database replication protocols? To answer this question group communication primitives are discussed and then a framework is presented to reason about the usefulness of these primitives in database replication. A recurrent approach to replication based on group communication is the deferred update model, which is presented next. In this discussion, unless stated otherwise, servers are assumed to have a full copy of the database.

Group Communication Primitives

The abstraction of groups (of servers) acts as a logical addressing mechanism in a replicated system, allowing the degree of replication and the identity of the individual servers to be ignored. For example, a group can abstractly represent the servers that replicate the database. Moreover, group communication primitives provide one-to-many communication with various powerful semantics, and hide much of the complexity of maintaining the consistency of the system and handling failures of replicated servers.

Group communication is usually characterized by two primitives: multicast(m, g) allows a message m to be forwarded to the members of group g; the message is passed to the application by means of the deliver (m) primitive. Multicast and deliver are implemented on top of simpler communication primitives (i.e., providing weaker guarantees). A multicast message is delivered to the application after the members of the group the message is addressed to have interacted, possibly exchanging point-to-point messages, to implement the properties ensured by the group communication (see Fig. 1).

Group communication primitives typically provide *message atomicity*: if a server in g delivers m, then every



Replication Based on Group Communication. Figure 1. Group communication primitives.

non-faulty server in g should also deliver m. Notice that if a server is down when a message m is multicast, it will not be able to deliver m (i.e., the server is faulty). If the server later recovers, it should deliver all messages it has missed. This is done using a state transfer mechanism, by which a recovering server receives the missed state from operational severs before it resumes its execution. State transfer is transparent to the application and implemented by the group communication system.

Another property typically ensured by group communication primitives concerns *message ordering*. Two such properties are *first-in-first-out (FIFO)* and *total order*. FIFO ensures that messages multicast by the same sender are delivered in the order they are multicast, that is, if a server executes multicast(m, g) followed by multicast(m', g), all non-faulty servers in g will deliver m first and then m'. Total order guarantees that all messages, regardless of the sender, are delivered in the same order by all non-faulty servers. If server s executes multicast(m, g) and server s' executes multicast(m', g), then m and m' should be delivered in the same order. Notice that FIFO is not a special case of total order. With total order, if a server multicasts m followed by m', the destinations can deliver m' before m, as long as they all deliver m and m' in the same order. Both properties can be combined in a *FIFO total order* property.

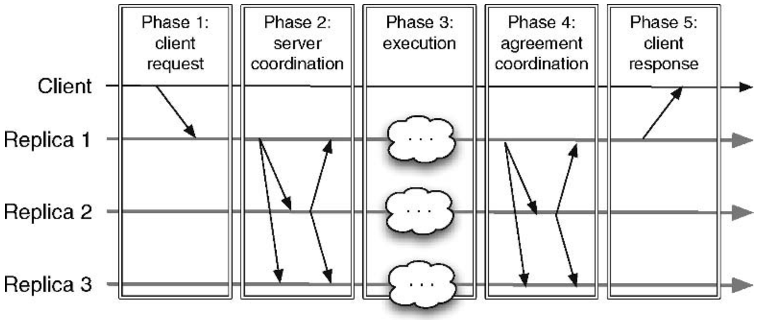
Groups can be *static* or *dynamic*. Static groups do not change their membership during the execution, although members can crash and later recover. Dynamic groups adapt the membership according to the state of their members, operational or crashed. Dynamic groups use a sequence of views, $v_0(g), v_1(g), \ldots, v_i(g), \ldots$, to manage membership. Each view $v_i(g)$ defines the composition of g at some time t, i.e., the members of the group that are operational at time t. Whenever a server in some view $v_i(g)$ is suspected to have crashed, or some server wants to join, a new view $v_{i+1}(g)$ is installed, which reflects the membership change.

Group communication primitives for dynamic groups can guarantee properties relating views to multicast messages. One such property is *sending view delivery*, according to which a message can only be delivered in the context of the view in which it was multicast. This means that if a server multicasts message m while in view $v_i(g)$, then m can only be delivered before view $v_{i+1}(g)$ is installed.

The following sections use two multicast primitives with different properties: abcast(m, g) ensures message atomicity and total order in the context of static groups; vscast(m, g) assumes dynamic groups and guarantees message atomicity, FIFO ordering, and sending view delivery properties. In both cases, messages are delivered to the application using deliver(m).

A Functional Model for Database Replication

Replication protocols can be generically decomposed into five phases, as shown in Fig. 2 [16]. Some protocols may skip some phases, while others may apply some of the phases several times during the execution of transactions. In Phase 1 the client contacts one or more database replicas to submit its requests. Phase 2 involves coordination among servers before executing requests. (Notice that message exchanges in Fig. 2 are illustrative only; different communication protocols may present different message patterns.) The actual execution of a request takes place in Phase 3; one or more servers may be involved in the execution of the operation. In Phase 4 servers agree on the result of the execution, usually to ensure atomicity, and in Phase 5 one or more servers send the result to the client. There



Replication Based on Group Communication. Figure 2. A functional model for database replication.

are basically two parts in which group communication primitives can help: during *server coordinaton* (Phase 2) and *agreement coordination* (Phase 4).

Database replication protocols can be *eager* or *lazy*, according to when update propagation takes place; and *update everywhere* or *primary copy*, according to who performs the updates. Group communication has traditionally been used by eager protocols, both in the case of update everywhere and primary copy. Some eager approaches are briefly reviewed next; a more detailed account can be found in [16].

Distributed two-phase locking (D2PL) is a traditional mechanism to coordinate servers during transaction execution. In replicated scenarios, implementing D2PL with point-to-point communication may lead to many distributed deadlocks [7]. If lock requests are propagated to servers using abcast, however, total order is ensured, and the probability of deadlocks is reduced [2]. In such a protocol, the client submits its requests to one database server (Phase 1) which abcasts them to all servers (Phase 2). Upon delivering the request, servers should make sure that conflicting locks are obtained in a consistent manner (Phase 3). Once the servers have executed the operation they send the response to the client (Phase 5). Commit and abort requests are processed like any other operation and so there is no need for agreement coordination (Phase 4). Notice that Phases 1–3,5 may be executed multiple times, once per request. If transactions can be predefined (e.g., storage procedures), then they can be abcast to all servers as a single request, and then Phases 1–3,5 are executed only once per transaction [3,11].

The deferred update approach is another form of eager update-everywhere replication mechanism based on group communication. In this case, the client selects one database server and submits all its requests to this server, without communicating with other servers (Phase 1). During the execution of the transaction, there is no synchronization among servers (i.e., no Phase 2), and only the selected server executes the transaction (Phase 3). Read-only transactions are committed locally by the selected server. Update transactions are propagated to all servers at commit time using abcast. The delivery of a terminating update transaction triggers a certification procedure used to ensure consistency. If the transaction passes the certification test its updates are committed against the database (Phase 4). Only then the reply is sent to the client (Phase 5). Several group communication-based replication protocols follow this approach, which is discussed in more detail in the next section.

Eager primary copy replication can also take advantage of group communication primitives. In this case clients interact only with the primary copy during the execution of the transactions (Phase 1). Therefore, there is no Phase 2. After the primary executes a request it sends the result to the client (Phase 3 executed by the primary only). Several requests from the client may be executed against the primary copy until commit is requested, at which point the primary communicates the new database state (e.g., redo logs) to the secondary copies (Phase 4). The secondaries apply the modifications to their local database. Communication between the primary and the secondaries is through vscast. FIFO ordering ensures that updates from the primary are received in the order they are sent. Sending view delivery guarantees correct execution in case of failure of the primary. For example, if the primary fails before all secondaries receive the updates for a certain request and another replica takes over as new primary, vscast ensures that updates sent by the new primary will be properly ordered with regard to the updates sent by the faulty primary.

Deferred Update Database Replication Protocols

Database replication protocols based on the deferred update technique, sometimes called certification-based protocols, are optimistic in that transactions execute on a single server without synchronization among servers. Abcast ensures that at termination transactions are delivered in the same order by all servers. The total order property together with a deterministic certification test guarantee that all servers agree on which transactions should be committed and which ones should be aborted.

The precise way in which the certification test is implemented and the information needed to implement it depend on the consistency criterion. Two typical consistency criteria are serializability and snapshot isolation. Serializability specifies that a concurrent execution of transactions in a replicated setting should be equivalent to a serial execution of the same transactions using a single replica. With snapshot isolation, transactions obtain at the beginning of their execution a “snapshot” of the database reflecting previously committed transactions; a transaction can commit as long as its writes do not intersect with the writes of the transactions that committed since the snapshot was taken.

The certification test may require transaction readsets and writesets. Transaction readsets and writesets refer to the data items the transaction reads and writes during its execution (e.g., the primary keys of the rows read or written). Let $RS(T)$ and $WS(T)$ denote, respectively, the readset and writeset of transaction T, and let $CC(T)$ be the set of transactions that executed concurrently with T. There are different ways to define the concurrent transactions set; typically, the set contains the transactions that committed after T started its execution, but before T itself committed. The propositions below abstractly define the conditions for committing a transaction according to serializability and snapshot isolation, respectively.

Serializability. T can commit if and only if $\forall T' \in CC(T) : RS(T) \cap WS(T') = \emptyset$
Snapshot isolation. T can commit if and only if $\forall T' \in CC(T) : WS(T) \cap WS(T') = \emptyset$

There are two general ways in which certification can be implemented in order to ensure serializability: with and without explicit information about transaction readsets. If information about data items read by the transaction is available, both the readsets and the writesets, together with the actual updates, are abcast to all servers as part of the termination of T. Upon delivering this message, each server individually evaluates the condition for serializability and determines whether T can be committed. When certifying T, each server builds $CC(T)$ based on transactions previously delivered, and in doing so, the test can be implemented deterministically, guaranteeing that all servers reach the same decision concerning committing transactions [12].

Certification without transaction readsets can be performed with an additional communication step among servers [9]. In the following, read operations are assumed to be performed before write operations. When a transaction requests to commit, only its writeset and the actual updates are abcast to all servers. After delivering the message, the server that executed the read operations tries to execute all write operations of the terminating transaction. Ongoing local transactions with read locks that conflict with the terminating transaction's write locks require special care: If the transaction has not been abcast yet, it is simply locally aborted; otherwise an abort message is abcast to all servers. Once the terminating transaction has acquired all its write locks, a commit message is abcast to all servers, which upon delivery commit the transaction.

Snapshot isolation can be implemented with a single round of messages (i.e., one abcast per transaction only) without readsets [9]. At commit time the writeset and the updates of a committing transaction are abcast to all servers. Each server executes the certification test and decides individually whether the transaction can commit or not.

Key Applications

Database replication protocols based on group communication are usually used to guarantee high availability for OLTP applications. Some of the existing protocols have been proved to provide good scalability under read-intensive workloads. Given the frequent assumption that each participating server should have a full copy of the database (i.e., full replication), scalability under write-intesive workloads is limited.

Future Directions

Replication based on group communication has been a hot topic of research in the past years. Several protocols have been proposed, a few implemented, and some made available for free download. While the main tradeoffs are known now, some problems are still open.

One open problem concerns how to structure group communication-based database replication protocols. According to whether changes in the database engine are required or not, there are two possibilities: kernel- and middleware-based architectures. Kernel-based protocols (e.g., [12,9]) take advantage of internal components of the database to increase performance in terms of throughput, scalability, and response time. For the sake of portability and heterogeneity, however, replication protocols should be independent of the underlying database management system. Even if the database internals are accessible, modifying them is usually a complex operation. As a consequence, middleware-based database replication (e.g., [6,13]) has received much attention in the last years. Such solutions can be maintained independently of the database engine, and can potentially be used in heterogeneous settings. Kernel-based protocols, on the other side, have more information about the data accessed by the transactions (e.g., readsets), which may result in more concurrency or less abort rate or both. In order to make its complexity more manageable, some efforts have tried to standardize kernel-based protocols [1].

Another problem subject to further investigation is partial replication. Full replication protocols have limited scalability under update-intensive workloads [8]. This is not a specific limitation of existing protocols, but an inherent characteristic of fully replicating the database on each server. Each new server added to a fully replicated system allows more clients to connect and submit transactions. If such transactions update the database, they will add load to every individual server. Partial replication does not suffer from the same problem since the degree of replication of each data item can be controlled. Thus, servers can be added to the system, improving the performance of data mostly read and without increasing the load of existing servers. It is not obvious how to implement partial replication in the context of group communication though and few proposals have considered it so far (e.g., [5]).

Cross-references

▶ Consistency Models for Replicated Data
▶ 1-Copy-Serializability
▶ Data Replication
▶ Replica Control

Recommended Reading

1. http://gorda.di.uminho.pt/
2. Agrawal D., Alonso G., Abbadi A.E., and Stanoi I. Exploiting atomic broadcast in replicated databases. In Proc. 3rd Int. Euro-Par Conference, 1997.
3. Amir Y. and Tutu C. From total order to database replication. In Proc. 22nd Int. Conf. on Distributed Computing Systems, 2002.
4. Birman K. The process group approach to reliable distributed computing. Commun. ACM, 36(12):37–53, 1993.
5. Camargos L., Pedone F., and Wieloch M. Sprint: a middleware for high-performance transaction processing. In Proc. Second European Conference on Systems Research, 2007.
6. Cecchet E., Marguerite J., and Zwaenepoel W. C-JDBC: flexible database clustering middleware. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004.
7. Gray J.N., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc. 1996 ACM SIGMOD Int. Conf. on Management of Data, 1996.
8. Jiménez-Peris R., Patiño-Martínez M., Alonso G., and Kemme B. Are quorums an alternative for data replication? ACM Trans. Database Syst., 28(3):257–294, 2003.
9. Kemme B. and Alonso G. A new approach to developing and implementing eager database replication protocols. ACM Trans. Database Syst., 25(3), 2000.
10. Lamport L. Time, clocks, and the ordering of events in a distributed system. Commun. ACM, 21(7):558–565, 1978.
11. Patiño-Martínez M., Jiménez-Peris R., Kemme B., and Alonso G. Middle-R: Consistent database replication at the middleware level. ACM Trans. Comput. Syst., 23(4), 2005.
12. Pedone F., Guerraoui R., and Schiper A. The database state machine approach. Distrib. Parall. Databases, 14(1):71–98, 2003.
13. Salas J., Jiménez-Peris R., Patiño-Martínez M., and Kemme B. Lightweight reflection for middleware-based database replication. In IEEE Int. Symp. on Reliable Distributed Systems, 2006.
14. Schiper A. and Raynal M. From group communication to transaction in distributed systems. Commun. ACM, 39(4):84–87, 1996.
15. Schneider F.B. Implementing fault-tolerant services using the state machine spproach: A tutorial. ACM Comput. Surv., 22 (4):299–319, 1990.
16. Wiesmann M., Pedone F., Schiper A., Kemme B., and Alonso G. Understanding replication in databases and distributed systems. In Proc. 20th Int. Conf. on Distributed Computing Systems, 2000.

Replication for High Availability

Bettina Kemme
McGill University, Montreal, QC, Canada

Synonyms

Fault-tolerance; Backup mechanisms

Definition

Replication is a common mechanism to increase the availability of a data service. The idea is to have several copies of the database, each of them installed on a different site (machine). Using replication, the data remains available as long as one site is running and accessible. Fault-tolerance is related to availability. A system is considered fault-tolerant if it continues to work correctly despite the failure of individual components. Replicating data and processes over several sites, the failure of any individual site can be masked since the tasks executed by the failed site can be transferred to one of the available sites. The terms high availability and fault-tolerance are often used interchangeably. However, fault-tolerance is stronger than a high availability solution since it expects the fault-tolerant system to behave exactly as a system where components never fail. This requires to make failures transparent to clients and typically means that all data copies have to be consistent at all times. This is difficult to achieve in systems where network connectivity is not always guaranteed. There, a high availability solution might allow users to access locally available data copies although it is possible that they do not reflect the latest updates.

Historical Background

High availability and fault-tolerance have received attention in many different communities covering high available hardware, fault-tolerant software systems and data availability. Redundancy is one of the major mechanisms to achieve the desired goal.

In the database community, looking for high availability solutions was the main reason to start research into database replication [3], while replication for scalability and performance were only explored later. The research community has mainly focused on the maintenance of transactional properties despite various kinds of failures. Apart from mere correctness, there exist many implementation alternatives that can have a large impact on the performance of the system, and current major database systems have developed a wide range of high availability solutions with different trade-offs.

Data availability has also been explored in many other data centric domains such as file systems [10, 12], mobile systems [6] or fault-tolerant processes [4].

Foundations

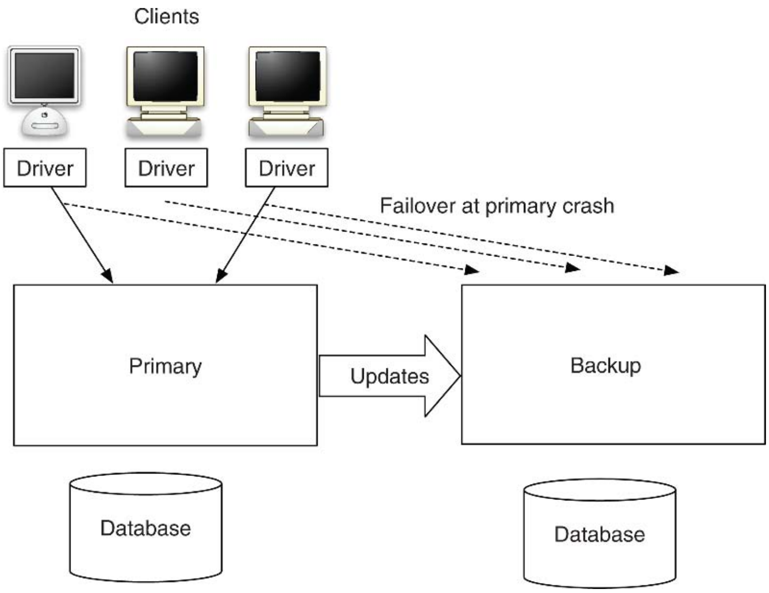
This entry focuses solely on availability for database systems. However, many of the issues and solutions discussed here are also valid outside the database domain.

Basic Fault-Tolerance Architecture

This discussion first looks at the case when a database system crashes. In this case, it does not perform any further actions and the data is no more available. Only after restart and recovery it becomes again operational. As the recovery procedure can take considerable time, a fault-tolerance solution maintains more than one copy of the database. In the following, the terms replica and site refer to a machine running a database management instance and controlling a copy of the complete database.

A fault-tolerance replication architecture has to tackle several issues. Firstly, when data is updated the data copies have to be kept consistent, resulting in additional measures, and thus *overhead during normal processing when no failures occur*. Secondly, when a replica fails, a *failover procedure* has to reconfigure the system. Thirdly, after a failure, the failed replica or a new replica have to be started and added to the system in order to handle future failures. For this, the joining replica has to receive the current state of the database from the available replicas. This third step is referred to as *replica recovery.*

As the most common fault-tolerance architecture uses primary-backup replication (Fig. 1) the following



Replication for High Availability. Figure 1. Primary-backup architecture.

discussion focuses on this architecture. There is one primary replica and one or more backup replicas in the system. All clients are connected to the primary replica which executes all transactions. In the case the primary fails, the clients have to be reconnected to one of the backups which takes over the tasks of the primary.

Execution While No Failures Occur The primary executes all transactions and decides on a serialization order (via the concurrency control module). At specific timepoints, the replication module of the primary replica propagates updates to the backup replica(s). Each backup replica applies the changes in an order that conforms to the serialization order determined by the primary.

Propagation can be done in several ways. In a *2-safe* approach, the changes performed by a transaction are propagated before the transaction commits, and an agreement protocol, such as a 2-Phase-Commit protocol, guarantees atomicity (the transaction either commits or aborts at all participating replicas). In a *1-safe* approach, the primary can commit a transaction before update propagation is completed.

1-safe propagation is faster than 2-safe propagation. However, if the primary fails after committing a transaction but before propagating its changes then the transaction is lost because the backup that becomes the new primary has no information about it. Neither atomicity nor durability of the transaction is given. Recovery of this transaction can be complex. In other context, 1-safe propagation is also often referred to as lazy, asynchronous or optimistic replication, while 2-safe propagation is often referred to as eager or synchronous replication.

There are several other design choices in regard to propagation. For instance, a message could be sent per update operation, per transaction or even per set of transactions. Furthermore, the update message could contain the physically changed records (e.g., taken from the log used for local recovery) or the update statement.

Changes can be applied at the backup at different timepoints. A backup is called *hot-standby* if it immediately applies any changes it receives to its own local database. Therefore, at the time of the failover, the backup is immediately operational and can accept client requests. However, it requires a powerful backup because of the high overhead during normal processing. A *cold-standby* defers applying the changes, e.g., to a timepoint when it is idle or when it actually becomes the new primary at failover time. This results in low overhead during normal processing. Thus, a cold-standby could be a less-expensive machine. However, failover will likely take longer and performance will be compromised until the powerful machine has completed recovery and can become again the primary.

Failover If there is more than one backup, an agreement protocol can be used to decide which of the backups takes over as new primary.

As part of the failover procedure, the clients have to be reconnected to the new primary. This reconnection is typically implemented within the driver software (e.g., JDBC driver) running on the client side (see Fig. 1). The driver is connected to the primary but knows the backup(s). When it looses connection to the primary, it connects to the backups to see who is the new primary.

Furthermore, the new primary, if is has not yet done so, has to apply all the changes of committed transactions that it received from the old primary before the crash. It aborts transactions for which it has already received some information but not all. A difficult issue is the handling of transactions that were active at the time of the failure. If the client had not yet submitted the commit request, the driver can simply throw an abort exception for this transaction before it connects to the new primary. This is correct since independently of being 1-safe or 2-safe, the new primary has neither committed nor is involved in a 2PC for this transaction. It is up to the client to resubmit the transaction (as would be the case for transactions involved in deadlocks). A different situation arises if the application program has already submitted the commit request, and the driver is waiting for the confirmation from the primary when the failure occurs. In this case, the transaction might already be committed at the backups. The driver has to ask the new primary for the outcome of the transaction and inform the client accordingly with a commit confirmation or an abort exception.

Replica Recovery Replica recovery is the task of integrating a new or failed replica into the replicated system. The recovery procedure can be performed *online* or *offline*. In offline recovery, transaction processing is halted in the system, the joining replica receives the accurate state of the data, and then transaction processing can resume. The problem is that availability is compromised. Online recovery, in contrast, performs recovery without stopping transaction processing in the rest of the system.

Data transfer strategies. There are several possibilities to provide a recovering replica with the latest state of the database [6]. One possibility is to transfer the entire database state to the recovering replica. This approach is simple but leads to unnecessary data transfer if large parts of the data did not change during the downtime of the recovering replica. Thus, an alternative solution first determines for each data item, whether this data item was actually changed during the downtime. Only if this is the case, the current version is transferred to the recovering replica. While this might decrease the size of data to be transferred it needs additional measures during normal processing and recovery to determine the changed objects. Instead of the data items, one can also transfer the log with all updates that the recovering replica has missed. Then, the recovering replica applies these missing updates to its database. The later two approaches are only possible for failed replicas that later recover while completely new replicas need a complete state transfer.

Recovery Procedure. A failed replica or a new replica typically joins as a backup. It can become the primary once recovery has completed (e.g., if it is the most powerful machine). The joining replica can receive the state from the primary or from another backup.

Using online recovery, the state transfer takes place concurrently to transaction processing at the primary. Thus, one has to make sure that the recovering replica does not miss any transactions. That is, for a given transaction, the updates performed by this transaction at the primary are either reflected by the state transfered through recovery, or are propagated by the primary to the new backup after recovery has completed.

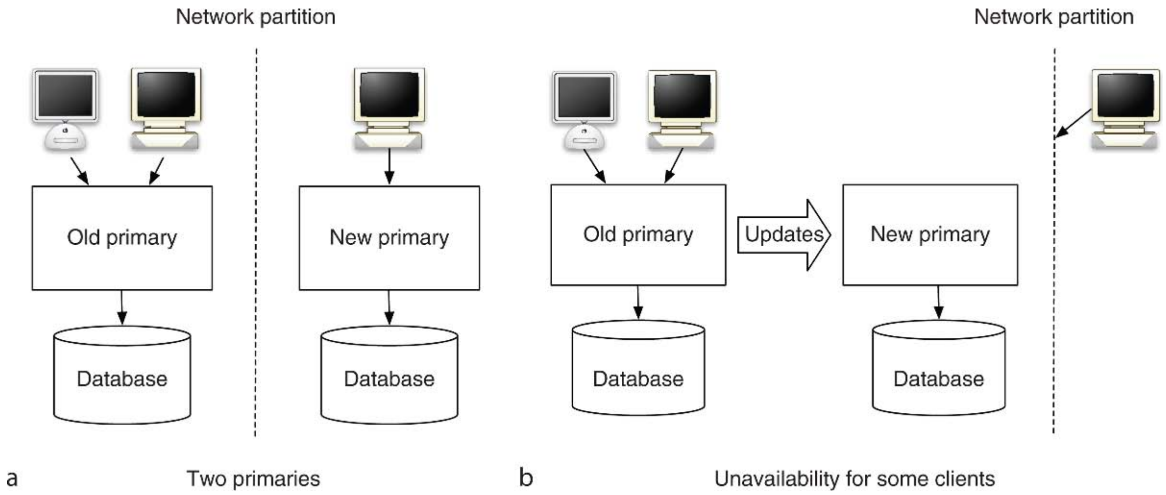
Combining Scalability and Fault-Tolerance

Many recent approaches exploit replication for both scalability and fault-tolerance by allowing transaction execution to be distributed across all available replicas (typically a LAN cluster). The protocols follow a read-one-write-all-available (ROWAA) replication strategy [2] where read operations are executed at one replica (which allows load-balancing) and write operations are executed at all replicas that are currently available (to keep copies consistent). In fact, the primary-backup approach is also a ROWAA approach but all clients are connected to the primary and transactions are only executed there. In these cluster replication solutions [9], when a replica fails, it is excluded from the system. Clients served by the replica are transferred to available replicas, and writes are only performed on the remaining available replicas. Failover is complicated by the fact of ongoing transaction execution during the failover period.

Network Partitions

Site crash is not the only failure type that can occur. A network problem might (temporarily) partition the replicas, allowing each replica to only communicate with a subset of other replicas. Network partitions occur seldom in LANs but are common in WANs or wireless networks. They are often transient and only hold for short periods of time. In general, it is impossible for a replica to determine whether another replica that is not reachable has failed, is currently not connected, or is simply slow and overloaded making it temporarily non-responsive. A correct fault-tolerance solution needs to work correctly despite this ambiguity. Standard primary-backup replication does not work correctly if network partitions occur. Figure 2 depicts two cases. When a network partition occurs between primary and backup (Fig. 2a), the backup and all clients within the partition of the backup will suspect the primary to have failed leading to both partitions having one primary serving local clients. If only clients but not the backup are disconnected from the primary (Fig. 2b), the system continues to run correctly, but from the disconnected clients' point of view, the system is unavailable.

Quorums One major approach to handle network partitions is to use quorums [7,13]. A read operation has to access a read quorum of replicas while a write operation has to access a write quorum of replicas. A quorum could, e.g., be the majority of replicas. Quorums are defined such that any two write operations on the same data item access at least one common replica. This is needed to serialize write operations and assure that any given write quorum has at least one replica with the latest version of the data item. Each read quorum must overlap with each write quorum. Therefore, each read operation is guaranteed to read at least one replica with the latest version. The beauty of



Replication for High Availability. Figure 2. Network partitions. (a) Two primaries. (b) Unavailability for some clients.

quorums is that they handle both replica failures and network partitions in the same way. It does not matter whether a client cannot reach a particular replica because it is down or disconnected, as long as the client can access a quorum of replicas, the operation can succeed. Recovery does not need any particular actions. While a replica is down it will not participate in any operations. Once it restarts it can be accessed and build part of a quorum. However, its data versions are outdated. Thus, any read operation will not read the data versions of this replica but the most current one from other replicas in the quorum. When the replica participates in a write quorum for a data item, the write operation will automatically update the value of the data item to the latest version.

Despite these attractive properties, quorums have not been used in database systems so far because of the complexity of handling database operations (e.g., complex SELECTs) on a quorum of replicas. The entry *Quorum Systems* discusses quorums in detail.

A possibility is to combine the strengths of ROWAA and quorums [1]. A connected group of replicas processes transactions according to ROWAA if the group fulfills a quorum requirement. A quorum is defined so that it is impossible that two partitioned groups of replicas can fulfill the quorum requirement. Group membership is dynamically adjusted whenever failures, restarts or network reconfigurations occur.

1-Safe (Lazy) Replication in WANs A further approach to address network partitions is to trade data accuracy with availability. It generally follows the ROWAA technique and performs update propagation lazily after commit (i.e., 1-safe). Each read operation only needs to access one copy of the data item, typically the one that is the closest. Thus, queries are usually fast and high availability is given since only one replica needs to be reachable.

For update transactions, one approach is to still have a primary replica and execute all update transactions at the primary. The primary then propagates the updates to all available secondary replicas sometime after commit. Having a single primary for update transactions allows for a globally correct serialization order. Propagating lazily allows the primary to commit a transaction without caring whether the other replicas are available. However, the secondaries can have outdated data, and thus local read operations might read stale data. Furthermore, if a client cannot reach the primary, its update transactions cannot be executed.

A second approach allows an update transaction to execute and commit at any replica, typically the closest, (update everywhere). This provides high availability for update transactions. However, data copies might diverge requiring some form of conflict detection and reconciliation, and read operations can read inconsistent data. Still, for many WAN applications, this is the only feasible option. Also, in mobile environments with planned periods of disconnection, this is the only way to provide availability to the mobile units.

In these lazy strategies, recovery is done incrementally. When a site wants to propagate an update but cannot reach a replica, it stores the update locally on persistent storage. When the replica is again available, the update will be propagated. That is, propagation is typically done via a persistent queue guaranteeing that an update is propagated and applied exactly once despite transient failures.

Other Failure Types

There exist other failures that are not considered here. For instance message loss can be handled by the underlying communication system.

Key Applications

Basically all commercial database systems provide a high availability solution based on primary-backup replication. Furthermore, other data replication strategies are also often supported and can be deployed for high availability and performance reasons. Most businesses that have update intensive workloads and require 24/7 availability deploy one or more of these solutions.

Outside the database domain, lazy (1-safe) replication is used both for fast local access and availability by file systems [10,12], web-servers [11] and cooperative applications such as distributed calendars.

Replication for fault-tolerance has also become popular in application servers. They represent the middle-tier in modern mutli-tier architectures, and maintain various kinds of data such as session information or cached versions of data stored in persistent storage. The challenges are similar as in pure database replication as the access to this data is transactional [5,14].

Future Directions

As applications are built increasingly on component-based architectures, combining software and hardware from many different vendors, it becomes a challenging task to assure that the system as a whole provides fault-tolerance. According to the end-to-end paradigm, the levels closest to the application might be the only ones able to completely implement the required level of fault-tolerance. On the other hand, lower levels might be able to ensure fault-tolerance more efficiently.

Cross-references

▶ Data Replication
▶ Logging and Recovery
▶ Optimistic Replication and Resolution
▶ Quorum Systems
▶ Replica Control
▶ Traditional Concurrency Control for Replicated Databases
▶ Two-Phase Commit
▶ WAN Data Replication

Recommended Reading

1. Abbadi A.E. and Toueg S. Availability in partitioned replicated databases. In Proc. 5th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1986, pp. 240–251.
2. Bernstein P.A. and Goodman N. An algorithm for concurrency control and recovery in replicated distributed databases. ACM Trans. Database Syst., 9(4):596–615, 1984.
3. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency control and recovery in database systems. Addison Wesley, Reading, MA, 1987.
4. Budhiraja N., Marzullo K., Schneider F.B., and Toueg S. The primary-backup approach. In Distributed Systems S. Mullender (ed.). (2nd Edition), Addison Wesley, Reading, MA, 1993, pp. 199–216.
5. DeCandia G., Hastorun D., Jampani M., Kakulapati G., Lakshman A., Pilchin A., Sivasubramanian S., Vosshall P., and Vogels W. Dynamo: Amazon's highly available key-value store. In Proc. 21st ACM Symp. on Operating System Principles, 2007, pp. 205–220.
6. Gray J., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
7. Jiménez-Peris R., Patiño-Martínez M., Alonso G., and Kemme B. Are quorums an alternative for data replication? ACM Trans. Database Syst., 28(3):257–294, 2003.
8. Kemme B., Bartoli A., and Babaoglu Ö. Online reconfiguration in replicated databases based on group communication. In Proc. IEEE Int. Conf. on Dependable Systems and Networks, 2001, pp. 117–130.
9. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Middleware based data replication providing snapshot isolation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 419–430.
10. Satyanarayanan M., Kistler J.J., Kumar P., Okasaki M.E., Siegel E.H., and Steere D.C. Coda: A highly available file system for a distributed workstation environment. IEEE Trans. Comput., 39(4):447–459, 1990.
11. Sivasubramanian S., Szymaniak M., Pierre G., and van Steen M. Replication for web hosting systems. ACM Comput. Surv., 36 (3):291–334, 2004.
12. Terry D.B., Theimer M., Petersen K., Demers A.J., Spreitzer M., and Hauser C. Managing update conflicts in Bayou, a weakly connected replicated storage system. In Proc. 15th ACM Symp. on Operating System Principles, 1995, pp. 172–183.
13. Thomas R.H. A majority consensus approach to concurrency control for multiple copy databases. ACM Trans. Database Syst., 4(2):180–209, 1979.
14. Wu H. and Kemme B. Fault-tolerance for stateful application servers in the presence of advanced transactions patterns. In Proc. Int. Symp. on Reliable Distributed Systems, 2005, pp. 95–108.

Replication for Scalability

Ricardo Jiménez-Peris, Marta Patiño-Martínez
Universidad Polytecnica de Madrid, Madrid, Spain

Synonyms

Scale out; Cluster replication; Scalable database replication

Definition

One of the main uses of data replication is to increase the scalability of databases. The idea is to have a cluster (of possibly inexpensive) nodes, to replicate the data across the nodes, and then distribute the load among them. In order to be scalable, the more nodes are added to the system, the higher the achievable throughput should be. The scale reached today is on tens of nodes (i.e., below 100 nodes). Communication is not an issue since CPU and IO overheads are dominant. The approach in the last years has been to learn from the traditional approaches but change some fundamentals so that the limitations of these traditional approaches are avoided.

In order to attain scalability each transaction should not be fully processed by every replica. This depends on how transactions are mapped to replicas. For read only transactions, it is easy to avoid redundant processing since they can be executed at any single replica. Update transactions are more challenging, since updates should be reflected at all replicas where there are copies of the updated data items. This should be achieved in an atomic way across the replicated system what makes scalability challenging. Under the traditional approaches, atomicity was attained by means of distributed locking (for isolation) and two-phase-commit (for failure atomicity), what resulted in solutions lacking scalability.

A second aspect introduced by updates is their overhead. With symmetric update processing no scalability is achieved for updates, since all replicas pay the cost of fully executing the transaction. However, for asymmetric processing some scalability is possible, since only one replica pays the cost of fully executing the transaction, whilst the others just pay the cost of propagating and installing the resulting updated tuples. Another of the major factors that impact the scalability is the degree of replication. Under full replication, all replicas keep a full copy of the database. This means that an update transaction executed at one of the replicas should propagate the resulting updated tuples to all other replicas. This update propagation overhead increases its relative weight with an increasing number of replicas, what inherently limits the scalability. Some research efforts pursue partial replication to overcome the scalability limit of full replication.

There are a number of additional factors that influence scalability. One of such factors is the consistency being provided. The traditional consistency criterion for replicated databases, 1-copy-serializability, constrains the potential concurrency therefore limiting the scalability. A lot of research has targeted to relax consistency in order to increase the scalability.

Historical Background

Traditional textbook approaches for data replication [3] were concerned with consistency and did not provide any scalability. The paper from Gray et al. [9] provided analytical evidence of this lack of scalability triggering a large body of research performed during the last decade on scalable database replication.

One set of approaches that aim at attaining scalability were based on lazy replication [8,4]. In order to provide consistency, some of these lazy approaches constrained update propagation to attain 1-copy-serializability [4].

A second batch of research explored scalable eager (synchronous) replication. The seminal approaches in this direction were Postgres-R [12] and the database state machine [17]. These approaches explored optimistic concurrency control under a white box (or kernel-based) replication approach and triggered the research performed in the last decade around scalable data replication. Due to the complexity of white box approaches, middleware-based replication was proposed to simplify the engineering of replication. The seminal approach to middleware-based replication was Middle-R [22,16] that explored scalable pessimistic replica control. The middleware approach from Middle-R became very popular and today it is one of the main approaches to engineer database replication [1,20,15,5]. Middleware approaches took two different flavors, those based on group communication [22,16,15] and those based on schedulers [1,5,20].

Another batch of research has looked at trading off some consistency in favor of scalability. There are two main families within this batch: those stemming from lazy approaches and those coming from eager replication. On the lazy replication side, consistency was relaxed through the concept of freshness. Freshness quantifies consistency, e.g., by the potential number of missed updates [8]. On the eager replication side, consistency was relaxed by resorting to isolation levels lower than serializability. The most popular one has been snapshot isolation [13] provides full 1-copy correctness (1-copy-snapshot-isolation, 1CSI) for snapshot isolation replicated databases. 1CSI is provided by a number of replication protocols both under update-everywhere [13] and primary-copy [20]. Another consistency criterion proposed based on snapshot isolation is Generalized Snapshot Isolation (GSI) [7]. It enables queries to see older snapshots that are prefix consistent. More recently, snapshot isolation has also been exploited in the context of lazy approaches in [6] and also for multi-tier architectures [18].

The fourth batch of research has tried to overcome the lack of scalability of full replication. The paper [11] demonstrated analytically the scalability limits of full replication. Two ways to overcome this inherent scalability limitation are quorum-based replication and partial replication. Quorum-based replication can improve the scalability for extreme update workloads, but introduces a number of technical problems, the most important ones related to the scalability of predicate reads [11]. Partial replication approaches have aimed at improving scalability by limiting the number of copies of each data item [21,19].

Foundations

One of the uses of database replication is to increase the scalability of the system. The use of data replication to scale is also known as *scale-out* approach, in which the scalability increases by adding new nodes. This contrasts with the *scale-up* approach in which a system scales by substituting the current node with a more powerful computer. This section discusses the major factors that influence scalability of data replication, namely: how to attain atomicity, the mapping of transactions to replicas, and the consistency criterion.

Attaining Atomicity and Isolation

The traditional way to attain transactional properties in data replication [3] was based on using distributed locking to serialize transactions in the same order (isolation at replicated level) and using two-phase-commit (2PC) to guarantee that either all replicas commit the transaction or none (failure atomicity). Gray et al. [9] showed analytically the lack of scalability of traditional approaches. Basically, distributed locking has a probability of deadlock that increases exponentially with the number of transactions and 2PC does not scale and produces high response times. Two trends were followed to escape from the bottlenecks of traditional approaches. Lazy protocols commit transactions without waiting for transaction update propagation, that is, updates are propagated lazily or asynchronously with respect to transaction commit at the replica that processed the transaction. Alternatively, eager protocols propagate transaction updates atomically using alternative ways to enforce atomicity more scalable than 2PC.

Lazy protocols have mainly resorted to two approaches to attain some levels of consistency: primary-copy (aka single-master) and multi-master data replication. The *primary-copy* approach simplifies the consistency problem by enabling updates only at a single replica known as primary. All the other replicas, known as secondaries, only allow the execution of read-only queries. In this way, the consistency problem boils down to installing the updates from the primary in FIFO order or a more relaxed order that guarantees the same relative order of conflictive transactions. A popular replication protocol providing primary-copy is [20].

Primary-copy replication has its bottleneck in the primary that has to fully process all update transactions. As an alternative, *multi-master* replication has been proposed in which update transactions can be processed by any replica. Different approaches have been adopted for multi-master replication. In some approaches the way update transactions are propagated is constrained to guarantee high levels of consistency [4]. In others consistency is relaxed by providing quantitative levels of consistency such as freshness. A more detailed discussion on freshness is provided later in this text. Leganet is one of the systems with a lazy replication protocol providing freshness [8].

Eager protocols aim at high levels of qualitative consistency. Most eager protocols are either based on group communication or on a scheduler. *Approaches based on group communication* use atomic multicast to enforce atomicity in a more scalable way than 2PC. Atomic multicast provides failure atomicity by guaranteeing that all or none of the replicas receive messages and also guarantees that all replicas receive the messages in the same relative order. In this way, each replica is able to schedule the messages containing updates in an order consistent with respect to the common total order. There are a number of data replication protocols based on group communication such as Postgres-R [12], database state machine [17], and Middle-R [16,13].

The *scheduler-based approach* has also been proposed for attaining consistency in eager protocols. The idea is to use a node as transaction scheduler. This scheduler is in charge of enforcing consistency. It labels transactions with a sequence number and forwards them to the replicas. In this way, replicas are able to order conflictive transactions in the same way and atomicity can also be enforced by preventing gaps in the transaction processing. Some of the database replication systems based on scheduler are conflict-aware data replication [1] and C-JDBC [5].

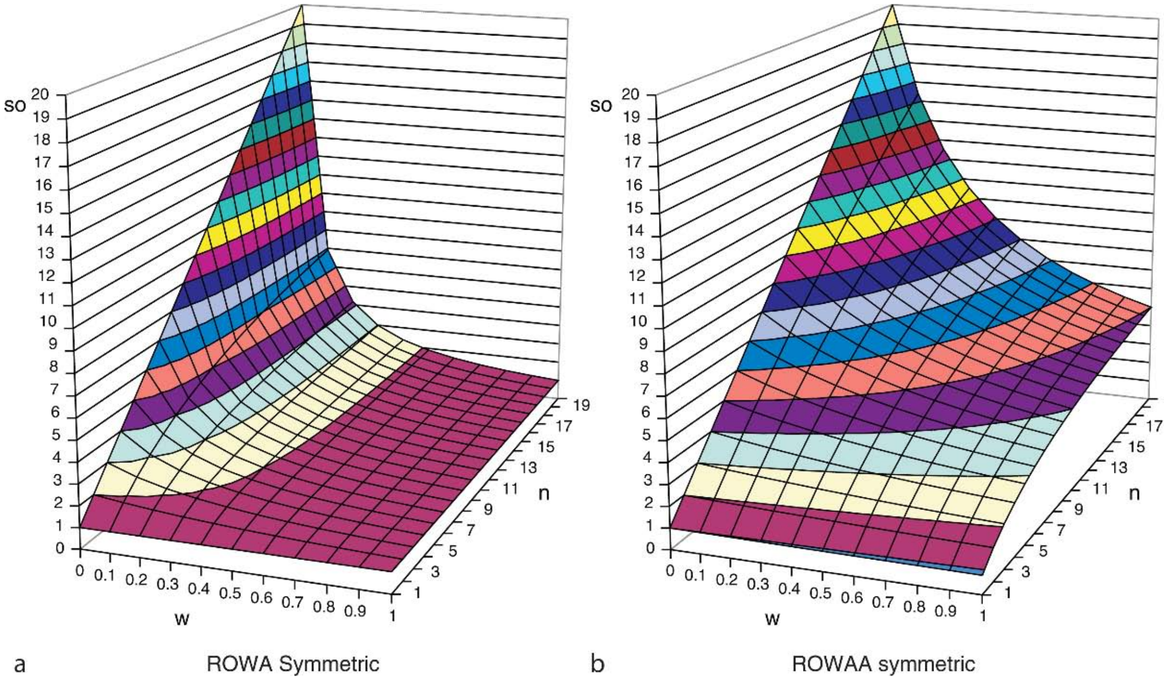
Transaction Mapping

The mapping of transactions to replicas is crucial in attaining scalability. There are several issues related to the transaction mapping: which replicas processes read-only transactions (queries), how update transactions are processed, and how many copies of each data item are kept.

The first issue is *how read-only transactions (queries) are managed*. Since queries only read data, for fully replicated systems it becomes possible to execute queries at any single replica, whilst update transactions are fully executed at all replicas. This mapping is known as read-one write-all-available approach (ROWAA) [3]. It enables to scale under read workloads, since for reads the load is shared among replicas. However, this approach fails to scale even with a small percentage of update transactions [11]. ROWAA has been used by the early replication protocols that are surveyed in [3].

The second issue is *how update transactions are processed*. Update transactions can be executed at only one of the replicas, as far as the other replicas get the resulting updates and install them. This mapping is known as *asymmetric update processing*, in contrast with *symmetric update processing* in which update transactions are fully executed by all replicas. This mapping also enables sharing the load introduced by update transactions, enabling scalability with update workloads [11].

Figure 1 compares the scalability of symmetric and asymmetric processing under ROWAA. The x-axis shows the fraction of writes (1.0 = 100% writes), the y-axis shows the number of nodes. The z-axis shows the scale-out, i.e., how many times the throughput of a single-node system is multiplied by the replicated system. Figure 1 shows that symmetric processing only achieves a high scale-out for very low values of w. With asymmetric processing, assuming that installing updates has around 15% of the costs of executing



Replication for Scalability. Figure 1. Scalability of symmetric and asymmetric processing (wo = 0.15 for asymmetric).

the update transaction itself ($wo = 0.15$), the scale-out is relatively high even for higher update rates.

A different mapping that has been proposed to further increase the scalability of update workloads is based on quorum systems. Quorum systems enable to write just in a subset of the replicas (write quorum). This has the associated tradeoff that reads should also be performed on a subset of replicas (read quorum). Quorums can compete with ROWAA when the percentage of updates in the workload is very high (80–100%) [11]. However, they only work for transactions accessing individual objects. When combined with collections of objects such as tables, queries become too expensive for being practical.

The third issue is *how many copies of each data item are kept (aka degree of replication)*. The paper [11] demonstrated analytically the scalability limits of full replication as a function of the ratio of the cost of fully executing the transaction and installing the resulting updates termed write overhead, *wo*.

With typical write overheads of 0.15 and below, the scalability becomes very reasonable for a few tens of nodes. However, full replication does not scale beyond that. The reason is that the overhead introduced by the update propagation and installation consumes higher and higher fractions of the capacity of the full system what finally is translated in consuming any extra additional capacity to process updates from the existing replicas.

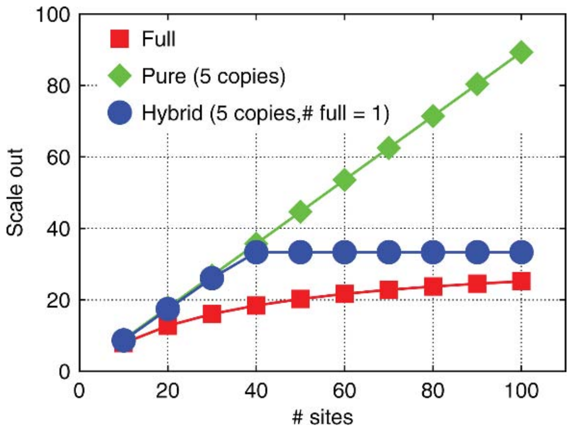
The alternative to scale beyond the limits of full replication is to use partial replication. Partial replication, however, also introduces other challenges. If there is no single replica with the full database, what is called pure partial replication, transactions become inherently distributed. As an alternative, if there is application knowledge available, it can be exploited to guarantee that there will be at least a replica that can execute a transaction fully locally. A potential solution is to have hybrid partial replication in which there are some replicas containing the full database and some replicas storing a fraction of the database. The former enables to avoid distributed transactions, and the latter enables to scale. Unfortunately, this approach, although it scales significantly better than full replication, reaches a limit of scalability. This limit is reached when the full replicas are saturated with update installation from partial replicas (even if they do not process any local transaction). The way to scale beyond hybrid partial replication is to use pure partial replication, what implies to be able to process replicated distributed transactions in an efficient way. The scalability of partial replication, both hybrid and pure, has been studied recently both analytically and empirically [21].

Figure 2 compares the scale-out of full replication, pure partial replication where each data item has five copies equally distributed among the nodes, and hybrid replication where one node has a copy of the entire database, and each data item has a total of five copies. The update workload in this case is 20% one can clearly see, that only pure replication has no scalability limit while the others are limited by the fact that all updates have to be performed on all copies, and there is at least one server that has copies of all data items.

Consistency Criterion

Another of the major factors influencing the scalability of replication is the consistency criterion. Some replication protocols provide a qualitative consistency. These protocols are usually *eager* protocols in which replica control is tightly integrated with update propagation to guarantee the qualitative consistency criterion. Other protocols have aimed at *quantitative consistency*, typically *lazy* protocols. In lazy protocols replica control is more relaxed and typically offers quantitative consistency, and in some cases simply eventual consistency.

Qualitative consistency for data replication provides a formal definition of the attained consistency. The consistency criterion can be seen as the extension of the



Replication for Scalability. Figure 2. Scalability of full versus partial data replication.

isolation concept of centralized systems to a replicated setting. Isolation formalizes the consistency for concurrent executions in a centralized system. The replication consistency criterion extends this formal notion to the consistency of the concurrent transaction execution in a replicated system. The traditional criterion for replicated databases has been 1-copy-serializability (1CS) [3]. This criterion states that the concurrent execution of a set of transactions in a replicated system should be equivalent to the execution in a serializable centralized system. 1CS has been the only criterion used till very recently. 1CS, as its centralized counterpart, serializability, constrains the potential concurrency in the system by making reads and writes conflicting. Constraining the potential concurrency is harmful for scalability (even for performance in a centralized system) since it might prevent to fully utilize the available capacity in the replicated system. In order to overcome the scalability limitation of 1CS, some researchers have explored snapshot isolation (SI) as isolation notion on which to build a replication consistency criterion. 1-copy snapshot-isolation [13] (1CSI) provides the notion of 1-copy correctness underlying in 1CS for a SI database. That is, the concurrent execution of a set of transactions in a replicated system should be equivalent to a centralized (1-copy) execution in a centralized SI database. The main advantage of SI is that reads and writes do not conflict. The only conflicts are between writes on the same tuples that are rare in most applications. 1CSI enables replica control protocols that are based on SI databases. This means that they do not have contention problems due to read-write conflicts, what results in higher scalability. Snapshot isolation is currently being used for different replica control protocols, lazy primary-copy replication [20], eager update-everywhere [13], lazy replication [6], and application server replication [18].

Quantitative consistency aims at increasing the scalability by relaxing the consistency enabling queries to see outdated data bounded quantitatively. It is typically used in lazy propagation schemes. Freshness criteria bound how much the value a query reads differs from the actual value of the data item. Freshness, e.g., could bound the number of updates a query has missed or the time since the copy read was last updated. The concept of freshness has been exploited by many different systems, and has been used in the context of primary-copy [20], multimaster [8], or application server replication [2]. The entry *Consistency Models for Replicated Data* provides a more detailed discussion on the different qualitative and quantitative consistency levels that exist.

Key Applications

Database replication has a wide number of applications. In principle any database that reaches saturation can be substituted by a replicated database to scale out. *Enterprise data centers* are certainly one of the main targets of database replication in order to scale for high loads typical of these systems. *Web farms* hosting dynamic content require the use of a database for storing it. In this kind of systems the main bottleneck is precisely the database and therefore, it can benefit from replicated databases. Grid systems are able to scale for applications based on the paradigm of bag of tasks, problems that can be split in a myriad of small subproblems that can be solved independently. In some cases, some grid applications require database access that is what finally becomes the bottleneck of the system. By employing a replicated database as a *data grid*, these grid applications can increase their scalability. A recent and increasingly important trend is *Software as a Service (SaaS)*. In this new paradigm, applications are hosted at a remote data center such as Google. SaaS aims for providing transparent scalability by means of self-provisioning. In this way, the hosted application can increase the number of replicas as needed depending on the received load and paying only for the resources it needs. SaaS platforms require an underlying scalable storage system such as a replicated database. Another area in which database replication is becoming a competitive solution is edge computing. *Edge computing* aims at moving web contents closer to clients located at the edge of Internet. Centralized solutions to edge computing are not adequate since distant clients observe high latencies. With edge computing, the contents is cached or replicated at data centers geographically close to the client to mask the latency. Early approaches only managed to reduce the latency of static contents. Recently, it has been shown that by using database replication in wide area networks it becomes possible also to mask the latency for dynamic contents [23].

Cross-references

► Data Replication

Recommended Reading

1. Amza C., Cox A.L., and Zwaenepoel W. Distributed versioning: consistent replication for scaling back-end databases of dynamic

content web sites. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2003.

2. Bernstein P.A., Fekete A., Guo H., Ramakrishnan R., and Tamma P. Relaxed-currency serializability for middle-tier caching and replication. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 599–610.
3. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison Wesley, 1987.
4. Breitbart Y., Komondoor R., Rastogi R., Seshadri S., and Silberschatz A. Update propagation protocols for replicated databases. In Poc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
5. Cecchet E., Marguerite J., and Zwaenepoel W. C-JDBC: flexible database clustering middleware. In Proc. USENIX 2004 Annual Technical Conference, 2004.
6. Daudjee K. and Salem K. Lazy database replication with snapshot isolation. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 715–726.
7. Elnikety S., Zwaenepoel W., and Pedone F. Database replication using generalized snapshot isolation. In Proc. 24th Symp. on Reliable Distributed Syst., 2005, pp. 73–84.
8. Gançarski S., Naacke H., Pacitti E., and Valduriez P. The leganet system: freshness-aware transaction routing in a database cluster. Inf. Syst., 32(2):320–343, 2007.
9. Gray J., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Poc. ACM SIGMOD Int. Conf. on Management of Data, 1996.
10. Jiménez-Peris R., Patiño-Martínez M., Alonso G., Kemme B. Scalable Database replication middleware. In Proc. 22nd Int. Conf. on Distributed Computing Systems, 2002.
11. Jiménez-Peris R., Patiño-Martínez M., Alonso G., and Kemme B. Are quorums an alternative for data replication. ACM Trans. Database Syst., 28(3):257–294, 2003.
12. Kemme B. and Alonso G. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000.
13. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Middleware based data replication providing snapshot isolation. In Poc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
14. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Enhancing edge computing with database replication. In Proc. 26th Symp. on Reliable Distributed Syst., 2007.
15. Muñoz-Escoí F.D., Pla-Civera J., Ruiz-Fuertes M.I., Irún-Briz L., Decker H., Armendáriz-Iñigo J.E., and de Mendívil J.R.G. Managing transaction conflicts in middleware-based database replication architectures. In Proc. 25th Symp. on Reliable Distributed Syst., 2006, pp. 401–420.
16. Patiño-Martínez M., Jiménez-Peris R., Kemme B., and Alonso G. Middle-R: consistent database replication at the middleware level. ACM Trans. Computer Syst., 23(4):375–423, 2005.
17. Pedone F., Guerraoui R., and Schiper A. The database state machine approach. Distributed and Parallel Databases, 14(1): 71–98, 2003.
18. Perez-Sorrosal F., Patiño-Martínez M., Jiménez-Peris R., and Kemme B. Consistent and scalable cache replication for multi--tier J2EE applications. In Proc. ACM/IFIP/USENIX 8th Int. Middleware Conf., 2007, pp. 328–347.
19. Pinto A.L., Oliveira R., Moura F., and Pedone F. Partial replication in the database state machine. In IEEE International Symposium on Networking Computing and Applications, 2001, pp. 298–309.
20. Plattner C. and Alonso G. Ganymed: scalable replication for transactional web applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004.
21. Serrano D., Patiño-Martínez M., Jiménez-Peris R., and Kemme B. Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation. In IEEE Pacific Rim Dependable Computing Conference, 2007, pp. 328–347.
22. Serrano D., Patiño-Martínez M., Jiménez-Peris R., Kemme B. An Autonomic Approach for Replication of Internet-based services. In Proc. 27th Symp. on Reliable Distributed Syst., 2008.

Replication in Multi-Tier Architectures

Ricardo Jimenez-Peris, Marta Patiño-Martinez
Universidad Polytecnica de Madrid, Madrid, Spain

Synonyms

Cloud computing; Scale out; Cluster replication; Scalable replication; Application server clustering; SOA replication

Definition

Modern middleware systems are commonly used in multi-tier architectures to enable separation of concerns. For each tier, a specific component container is provided, tailored to its mission, web interface, business logic, or persistent storage. Data consistency across tiers is guaranteed by means of transactions. This entry focuses on the main three tiers: web, application server, and database tiers.

Middleware systems are at the core of enterprise information systems. For this reason they require high levels of availability and scalability. Replication is the main technique to achieve these two properties. First middleware replication approaches addressed the replication of individual tiers and focused initially on providing availability and later, on scalability. However, an integral approach is needed to provide availability and scalability of multi-tier systems. Replicating an individual tier increases its availability and might increase its scalability as well. However, replicating a single tier is not sufficient for providing data availability and scaling for the whole system; the non-replicated tiers eventually become a single point of failure and/or a potential bottleneck. A potential solution is to combine independently replicated tiers. However, due to their independent replication, they are not aware of the replication of each other; this results in inconsistency problems in the event of failures.

Recent research has introduced new approaches to replication of multi-tier architectures. The two main replication architectures are horizontal and vertical replication. In horizontal replication, the tiers are replicated independently, then measures are taken to make at least one of them aware of the other to deal in a consistent way with failures and failovers. In vertical replication, a set of tiers (typically the application and database tiers) are replicated as a unit, and the replication logic is encapsulated in the upper tier (typically the application server). Another important aspect of replication of multi-tier systems is related to the definition of the correctness criterion, i.e., the conditions a replicated multi-tier architecture should fulfill in order to provide consistency. There are two important criteria in this area, namely, exactly-once semantics and 1-copy correctness.

Historical Background

The first wave of research on the replication of multi-tier architectures focused solely on the availability of individual tiers, mainly the application server and database tiers. In the first wave, the theoretical basis for process replication was set [22]. The state machine approach was proposed to guarantee the consistency of replicated servers. This approach stated that a server could be consistently replicated by guaranteeing that all server replicas receive the requests in the same order and the server deterministically processed them. This determinism requirement was interpreted as the server had to be sequential, which was too restrictive for real servers that are multi-threaded in nature.

The efforts on the database tier concentrated on how to attain data consistency in the advent of failures and how to characterize data consistency. These initial efforts led to lock-based data replication approaches. These approaches were not scalable but they enabled the development of correctness criterion. The criterion characterizing correctness was 1-copy-serializability, i.e., replication should be semantically transparent, the replicated database should allow only those executions that had an equivalent in a non-replicated (1-copy) system.

A second wave of research on middleware replication addressed the fault-tolerance of real systems in the context of CORBA, which led to the (fault-tolerant) FT-CORBA standard. These approaches were based on the theoretical foundations of the state machine and tried to solve the difficulties of engineering replication in real middleware systems [2,7,16]. A new line of research opened in this wave enabled the use of the state machine approach for multi-threaded servers. A seminal paper [11] showed that it was possible to devise a scheduler that can guarantee deterministic behavior of multi-threaded servers. Later approaches studied how to increase the potential concurrency [3].

The efforts on the database tier focused on how to attain scalability. Some approaches relaxed consistency in the quest for scalability relying on lazy replication approaches (see the entry *Replication for Scalability* for a deeper view) [5,17]. Other approaches aimed at attaining scalability while preserving full consistency [1,12,18,19]. The three later approaches leveraged group communication to provide consistency in any failure scenario.

The third wave of research on middleware replication focused on dealing with specific aspects of multi-tier architectures, more concretely, on how to deal with transactional processing in a consistent way, the consistency criteria, and how to attain scalability. The issue of the lack of transactional consistency in replication approaches for multi-tier architectures was then raised [8]. In the context of FT-CORBA, it was studied how to connect a replicated CORBA application server with a shared database guaranteeing exactly once semantics [25], therefore providing the same semantics as the non-replicated system. Also in the J2EE context transactional consistency has been studied. First approaches replicated session state in the application server replicas sharing a common database [24] and enforced transactional consistency in this context.

More recently, replication of multiple tiers has also been studied, e.g., in a vertical replication approach [21]. That is, an application server and database server pair is the replication unit. Database servers are centralized and are not aware of the replication. The application server encapsulates the replication logic transparently to the database. The application server replicas interact among them to enforce consistency (i.e., 1-copy-serializability). Additionally, in this approach transactions are not aborted in the advent of failures from the client perspective, that is, it provides high transaction availability. Another recent approach has considered the issue of replicating application servers accessing multiple databases [14]. In this approach a primary-backup model is adopted. Clients interact with the primary application server. The most general case in which the application server accesses multiple databases is considered. This case is more complex since it involves dealing with two-phase-commit (2PC) and the failover of the 2PC coordinator (the transaction manager of the application server). The primary application server accesses multiple databases to process client requests. When the client transaction ends, 2PC is started and coordinated by the primary application server. The primary checkpoints session state as well as transaction management information to the backups. The approach is able to handle the primary failure and enforce transaction consistency for distributed transactions, providing transaction manager (2PC coordinator) failover.

On the correctness side, it was studied how to guarantee exactly once semantics in multi-tier architectures in which the application server tier is stateless [9]. That is, how to guarantee transaction atomicity and exactly once semantics in the advent of a failover in replicas of stateless application servers. A thorough study of the different approaches for multi-tier replication that guarantee transactional consistency was presented in [13].

On the database tier, the third wave of research has focused on boosting the scalability beyond a few tens of replicas attained in the second wave. Two aspects were mainly addressed: How to overcome the scalability limits of 1-copy-serializability and full replication. 1-copy-serializability limits significantly the potential concurrency of the system which had an impact on the attainable scalability. This resulted in approaches based on a more relaxed consistency criterion, 1-copy-snapshot-isolation [15]. The scalability limits of full replication were studied analytically in [10] which led to the exploration of partial replication. Partial replication has resulted in boosting the scalability of full data replication [23], although it has some complexities such as how to deal with data partitioning and distributed transactions.

Nowadays, integral approaches for the replication of multi-tier architectures are becoming common. Research in the area is aiming at enhancing the scalability of the seminal approaches to replication of multi-tier systems. More concretely [4] studies how to replicate the application server tier with a shared database and boost the scalability by relaxing data currency through freshness constraints. Another recent approach is looking at the scalable replication of application server and database tiers [20]. In order to overcome the scalability bottleneck of serializability, this approach is based on 1-copy-snapshot isolation. Additionally, it deals for the first time with the issue of the cache consistency at the application server for relaxed isolation levels, in particular, for snapshot isolation.

Foundations

This section reviews the main replication approaches for the different tiers. Both approaches (replicating a single tier and replicating multiple tiers) will be considered.

The web tier is simple to replicate due to its stateless nature. On the other hand, it has to deal with a very specific aspect that lies in the availability and scalability of the connectivity to the Internet. This has resulted in multiple approaches to virtualize URLs and IP addresses. IP virtualization enables providing transparent failover since the client sees a single logical IP despite the fact that two physical IPs might get involved in the processing of its requests upon a failover. IP virtualization also enables to provide transparent load balancing since a network switch can transparently route requests to a logical IP to different physical IPs to balance the load across different sites. A thorough survey on the replication of the web tier can be found in [6].

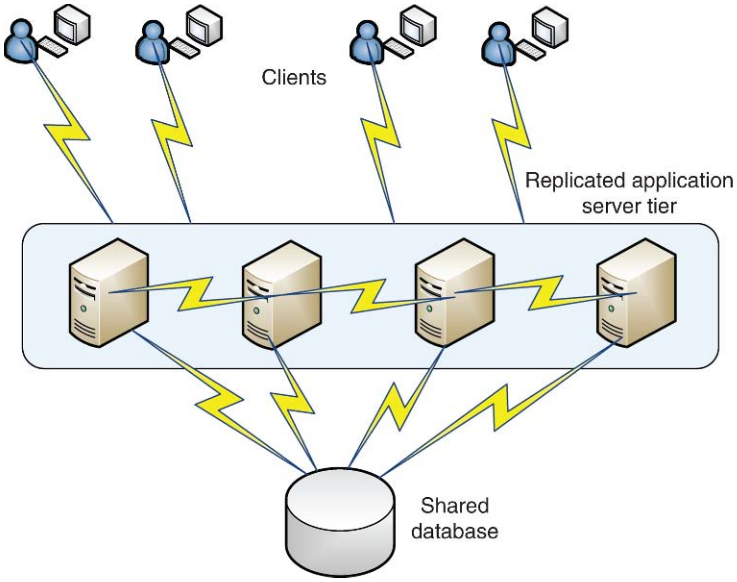
The replication of the application server tier is more challenging due to its stateful nature and also due to its support for transactional processing. It is possible to distinguish between replication for availability and replication for scalability. Replication for availability aims at providing fault-tolerance but typically without taking care of scalability, even providing negative scalability (a performance below the one of the non-replicated server). Replication for scalability, in addition to providing availability, it is able to increase the system capacity by increasing the number of replicas.

Replication for availability approaches rely on the state machine approach [22]. The state machine replication is based on two basic principles: (i) All replicas receive requests in the same global order; (ii) Each replica behaves deterministically and then, given the same input request sequence produces the same output sequence. Therefore, replication protocols for availability rely on a method for ordering client requests in the same order at all replicas to attain the required global order, typically, using group communication and total order multicast. Then, they remove the non-determinism from the application. In this class, it is possible to find all the efforts around FT-CORBA such as Eternal [16].

One of the main sources of non-determinism is multi-threading. Most of the approaches simply tackle with single-threaded (sequential) servers to guarantee determinism. However, multi-threading is a necessary feature for real-world servers. Some research has been conducted in how to guarantee determinism of replicas in the presence of multi-threading [3,11]. This research has produced different multi-threading schedulers that enforce deterministic scheduling enabling the replication of multi-threaded servers. The seminal approach [11] showed that by using a deterministic scheduler and setting a deterministic method to schedule requests integrated in the deterministic scheduler it became possible to guarantee the determinism of multi-threaded servers. Later on some other deterministic schedulers have been proposed aiming at increasing the real concurrency [3].

An important issue that needs to be dealt with in the replication of application servers is how to preserve transactional consistency [8]. The paper [9] proposes e-transactions to preserve consistency and identifies exactly once semantics as a key consistency criterion for replication of transactional multi-tier architectures. e-Transactions also provide a protocol for satisfying exactly-once semantics for replicated stateless application server tiers with a shared database (see Fig. 1). Exactly-once semantics states that each transaction should be executed exactly once despite failures and re-executions due to failovers.

A more evolved approach [24] deals with the replication of stateful application servers also with a shared database. In this approach, the relationship between request and transaction is not constrained to be 1 to 1 as in e-transactions, instead, it can be arbitrary. That is, it can be 1:1, N:1, and 1:N. In [24] it is proposed a protocol providing exactly-once semantics [9] despite failures. The protocol intercepts database accesses labeling them with a global sequence identifier and the associated client and request identifier. Each database requests is multicast to the other replicas with the associated information. In the advent of a failure of the primary replica of the application server, a backup replica will take over. Those clients that have not received a reply to their last request will resubmit it to the new primary. The new primary will execute resubmitted requests intercepting the requests submitted to the database. This interception takes care of executing the database requests in the same order as in the old primary in order to guarantee a



Replication in Multi-Tier Architectures. Figure 1. Horizontal replication with a replicated application server tier and a shared database.

deterministic execution and exactly once semantics. New requests will be delayed till the requests received from the former primary are replayed.

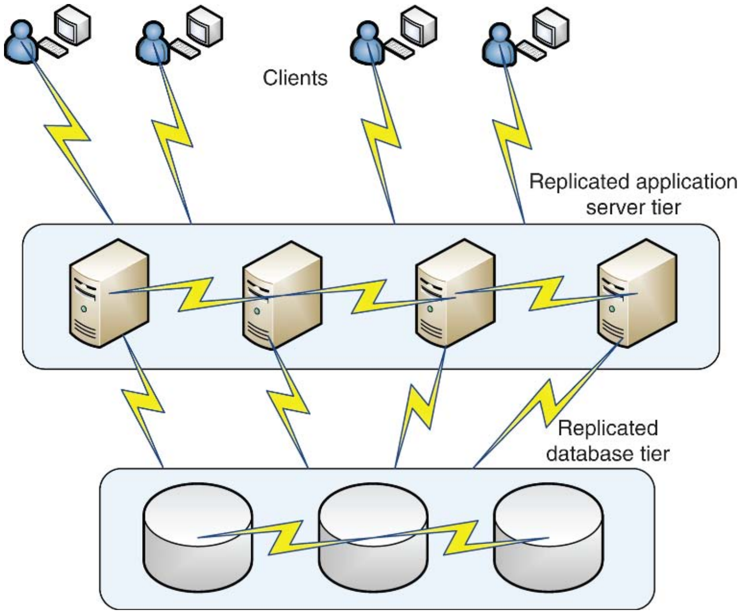
All the aforementioned approaches deal exclusively with the replication of the application server tier, which results in the database becoming a single point of failure and performance bottleneck. More recently, research has been conducted to deal with the replication of both the application server and database tiers guaranteeing consistency despite failures. Two alternative architectures have been considered: horizontal and vertical replication [13].

In horizontal replication each tier is replicated independently (see Fig. 2). However, the integration of two independently replicated tiers results in inconsistency problems in the advent of failures [13]. In order to avoid inconsistencies the replicated tiers sholud become aware of each other to handle failovers consistently. However, this is typically unrealistic since it implies the cooperation of two different vendors very often competitors in the market. [13] proposes two solutions to attain consistent failover by incorporating awareness of replication in only one of the two tiers, either the application server or the database tier.

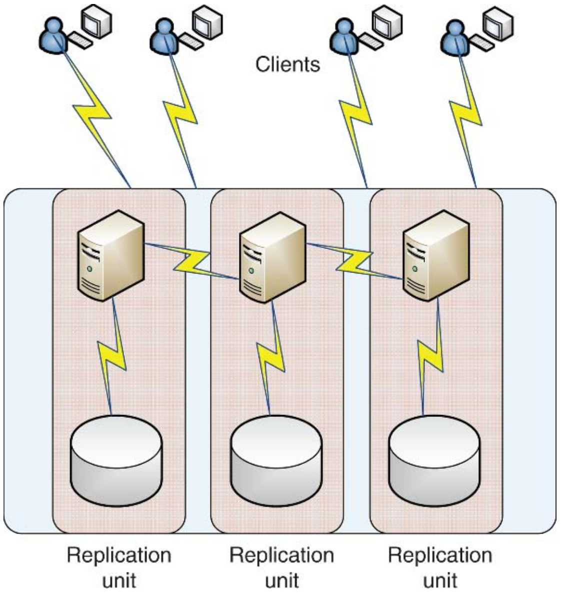
Vertical replication lies in taking as replication unit an application server - database server pair [21] (see Fig. 3). The database server is not aware of the replication. The application server encapsulates all the replication logic. In the vertical replication approach [21] both changes to session (session beans in the context of J2EE) and persistent state (entity beans in the context of J2EE) are captured. The two kinds of changes are propagated to the backup replicas at the end of each client request to provide *highly available transactions*, that is, transactions that do not abort despite failovers.

Both horizontal and vertical replication architectures are also used to attain scalability. Horizontal replication has mainly focused on replicating the middleware tier to replicate sessions while sharing a single database. In these approaches, consistency is attained through the shared database that serializes concurrent database accesses from different replicas. The replication of sessions enables to increase the scalability of applications in which the bottleneck is located in the application server, they are CPU intensive (e.g., image processing). However, these approaches fail to scale when the bottleneck is on the database, which is the most common case in practice.

Vertical approaches have aimed at increasing the scalability independently of whether they are application server or database intensive whilst providing strong consistency [4,20]. To date there are two different approaches to scalable vertical replication: [20] that provides strong consistency and [4] that provides relaxed consistency.



Replication in Multi-Tier Architectures. Figure 2. Horizontal replication with two replicated tiers.



Replication in Multi-Tier Architectures. Figure 3. Vertical replication. Replicated application server and database tiers.

[4] uses a master-slave approach and aims at scaling read-intensive workloads. The master processes all update transactions that get reflected in the underlying database and takes care of propagating a sequence of committed updates to the slaves. Each slave updates the cache and also commits the updates to the database in commit order. Read-only queries can be executed at any replica and can read stale data. The system can be configured to set the staleness or freshness upon which each transaction should be executed.

On the other hand, [20] is an update-everywhere approach (all replicas can process update transactions) and aims at scaling both read and write workloads. Unlike [4,20], provides strong consistency, namely, 1-copy-snapshot-isolation. [20] firstly provides a consistency criterion for application server caching, named cache-transparency. An application server is cache-transparent if it allows the same executions as the application server with the cache disabled. Cache transparency extends the isolation notion of databases to a multi-tier architecture. This is crucial since current application servers may work incorrectly with databases running on some relaxed isolation levels, such as, snapshot isolation. Secondly, it guarantees 1-copy correctness for the multi-tier server which provides replication transparency. [20] builds on 1-copy-snapshot isolation to obtain higher scalability levels since 1-copy-serializability has some strict scalability limits as discussed earlier. Basically, [20] uses multi-versioning in the cache of the application server replicas that is synchronized with the snapshots of the underlying database and among replicas in order to provide cache and replication transparency, respectively.

Cross-references

▶ Data Replication

Recommended Reading

1. Amza C., Cox A.L., and Zwaenepoel W. Distributed versioning: consistent replication for scaling back-end databases of dynamic content web sites. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2003.
2. Baldoni R. and Marchetti C. Three-tier replication for FT-CORBA infrastructures. Software Practice & Experience, 33(8):767–797, 2003.
3. Basile C., Kalbarczyk Z., and Iyer R.K. Active Replication of Multithreaded Applications. IEEE Trans. Parallel and Dist. Syst., 17(5):448–465, 2006.
4. Bernstein P.A., Fekete A., Guo H., Ramakrishnan R., and Tamma P. Relaxed-currency serializability for middle-tier caching and replication. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 599–610.
5. Breitbart Y., Komondoor R., Rastogi R., Seshadri S., and Silberschatz A. Update propagation protocols for replicated databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
6. Cardellini V., Casalicchio E., Colajanni M., and Yu P.S. The state of the art in locally distributed Web-server systems. ACM Comput. Surv., 34(2):263–311, 2002.
7. Felber P., Guerraoui R., and Schiper A. The Implementation of a CORBA Object Group Service. Theory & Practice of Object Systems, 4(2):93–105, 1998.
8. Felber P. and Narasimhan P. Reconciling Replication and Transactions for the End-to-End Reliability of CORBA Applications. In Proc. Int. Symp. on Distributed Objects and Applications, 2002.
9. Frølund S. and Guerraoui R. e-Transactions: End-to-End Reliability for Three-Tier Architectures. IEEE Trans. Software Eng., 28(4):378–395, 2002.
10. Jiménez-Peris R., Patiño-Martínez M., Alonso G., and Kemme B. Are Quorums an Alternative for Data Replication. ACM Trans. Database Syst., 28 (3), 2003.
11. Jiménez-Peris R., Patiño-Martínez M., and Arevalo S. Deterministic Scheduling for Transactional Multithreaded Replicas. In Proc. IEEE Int. Symp. on Reliable Distributed Systems, 2000, pp. 164–173.
12. Kemme B. and Alonso G. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000.
13. Kemme B., Jiménez-Peris R.R., Patiño-Martínez M.M., and Salas J. Exactly once interaction in a multi-tier architecture. In Proc. of

VLDB Workshop on Design, Implementation and Deployment of Database Replication, 2005.

14. Kistijantoro A.I., Morgan G., and Shrivastava S.K. Enhancing an Application Server to Support Available Components. IEEE Trans. Software Eng., SE-34(4):531–545, 2008.
15. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Middleware based Data Replication providing Snapshot Isolation. In Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD), 2005.
16. Moser L.E., Melliar-Smith P.M., Narasimhan P., Tewksbury L., and Kalogeraki V. The eternal system: an architecture for enterprise applications. In Proc. Int. Enterprise Distributed Object Computing Conf., 1999, pp. 214–222.
17. Pacitti E. and Simon E. Update propagation strategies to improve freshness in lazy master replicated databases. VLDB J., 8(3), 2000.
18. Patiño-Martínez M., Jiménez-Peris R., Kemme B., and Alonso G. Middle-R: Consistent Database Replication at the Middleware Level. ACM Trans. Computer Syst., 23(4):375–423, 2005.
19. Pedone F., Guerraoui R., and Schiper A. The Database State Machine Approach. Distrib. Parall. Databases, 14(1):71–98, 2003.
20. Perez-Sorrosal F., Patiño-Martínez M., Jiménez-Peris R., and Kemme B. Consistent and scalable cache replication for multi-tier J2EE applications. In Proc. ACM/IFIP/USENIX 8th Int. Middleware Conf., 2007, pp. 328–347.
21. Perez-Sorrosal F., Patiño-Martínez M., Jiménez-Peris R., and Vuckovic J. Highly available long running transactions and activities for J2EE applications. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2006.
22. Schneider F.B. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. ACM Comput. Surv., 22 (4):299–319, 1990.
23. Serrano D., Patiño-Martínez M., Jiménez-Peris R., and Kemme B. Boosting Database Replication Scalability through Partial Replication and 1-Copy-Snapshot-Isolation. In Proc. IEEE Pacific Rim Dependable Computing Conference, 2007, pp. 328–347.
24. Wu H. and Kemme B. Fault-tolerance for Stateful Application Servers in the Presence of Advanced Transactions Patterns. In Proc. IEEE Int. Symp. on Reliable Distributed Systems, 2005, pp. 95–108.
25. Zhao W., Moser L.E., and Melliar-Smith P.M. Unification of Transactions and Replication in Three-Tier Architectures Based on CORBA. IEEE Transactions on Dependable and Secure Computing, 2(1), 2005.

Report Writing

► Summarization

Representation

► Icon

Reputation

► Trust in Blogosphere

Reputation and Trust

Zoran Despotovic
NTT DoCoMo Communications Laboratories Europe,
Munich, Germany

Synonyms

Feedback systems; Word of mouth

Definition

Trust means reliance on something or someone's action. As such, it necessarily involves risks on the side of the subject of trust, i.e., trustor. The main goal of a trust management system is to reduce the involved risks. Reputation systems present a possible solution to do that. They use relevant information about the participants' past behavior (feedback) to encourage trustworthy behavior in the community in question. The key presumptions of a reputation system are that the participants of the considered online community engage in repeated interactions and that the information about their past doings is informative of their future performance and as such will influence it. Thus, collecting, processing, and disseminating the feedback about the participants' past behavior is expected to boost their trustworthiness.

Key Points

The goal of a reputation system is to encourage trustworthy behavior. It is up to the system designer to define what trustworthy means in her specific setting. There are two possible ways to do this: *signaling and sanctioning reputation systems*. In a signaling reputation system, the interacting entities are presented with signals of what can go wrong in the interactions if they behave in specific ways. Having appropriate signals, the entities should decide what behavior is most appropriate for them. An important assumption of the signaling reputation systems is that the involved entities do not change their behavior in response to a change of their reputation. As an example, the system may just provide a prospective buyer with indications of the probability that the seller will fail to deliver a purchased item. This probability is the main property of the seller. It can change with time, but independently of the seller's reputation.

The other possibility is *sanctioning reputation systems*. The main assumption they make is that the involved entities are aware of the effect the reputation has on their benefits and thus adjust their behavior dynamically as their reputation changes. The main task of a reputation system in this case is to *sanction* misbehavior through providing correlation between the feedback the agent receives and the long-run profit she makes.

Cross-references

- ▶ Distributed Hash Table
- ▶ P2P Database
- ▶ Peer-to-Peer System
- ▶ Similarity and Ranking Operations
- ▶ Social Networks
- ▶ Trust in Blogosphere

Request Broker

Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA

Synonyms

Object request broker; Event broker; Storage broker

Definition

A Request Broker is a software manifestation of the Broker architectural pattern [3] that deals primarily with coordinating requests and responses, and managing resources among communicating entities in a distributed system. A Request Broker is usually found as part of middleware, which are layers of software that sit between applications, and the underlying operating systems, hardware and networks.

Historical Background

The mid to late 1980s established the TCP/IP protocol suite as the *de facto* standard suite of protocols for building networked applications. Contemporary operating systems provided a number of application programming interfaces (APIs) for network programming. The famous among these were the *socket* API that were tailored towards building TCP/IP-based applications. This era also saw the widespread use of killer TCP/IP-based applications, such as the File Transfer Protocol (FTP).

Despite the success of the socket API to build distributed applications, there were a number of challenges involved in developing these applications. First, the socket API was tedious to use and incurred a number of accidental complexities stemming from the use of type-unsafe C-language data types. Second, application programmers were responsible for handling the marshaling and demarshaling of the data types transferred between the communication entities, which involved a number of challenges. Notable among these were the need to address the byte ordering, and word size and padding issues arising from the heterogeneity in the hardware architectures. Third, server applications were able to provide only one primary functionality due to the lack of an object-oriented service abstraction. Finally, client applications had to explicitly bind to a location-dependent service, which made the applications inflexible and brittle.

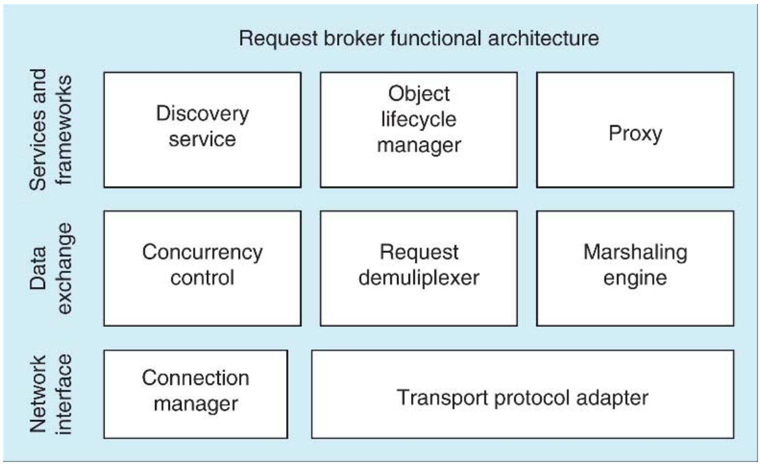
Every distributed application that was developed had to reinvent the wheel and address these complexities. There was a compelling need to overcome these challenges by factoring out the commonly occurring patterns of network programming and provide them within reusable frameworks. This led to the notion of middleware [1,2], which provide reusable capabilities in one or more layers of software that sit between the application logic, and the operating systems, hardware and networks. A Request Broker is at the heart of such a middleware and performs a number of functions on behalf of the applications.

Request Brokering capabilities at the middleware layer started appearing with the advent of the Remote Procedure Call (RPC). Sun RPC was among the earlier middleware platforms that illustrated Request Brokers. Others that emerged thereafter included the Distributed Computing Environment (DCE), the Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (RMI), Distributed Component Object Model (DCOM) and .NET Remoting.

Foundations

A Request Broker implements the Broker architectural pattern [3]. The primary objectives of a request broker are to decouple clients and servers of a distributed application and provide reusable services, such as concurrency management, connection management, seamless transport-level networking support, data marshaling, and location transparency. Figure 1 illustrates the commonly found functional blocks [5] in a Request Broker discussed below.

- *Proxy* – A Request Broker allows the separation of interface from implementation thereby decoupling the client of a service from the implementation of the service. A service is often described using interface definition languages to define the interfaces, the operations they support and data types that can be exchanged. Versioning of interfaces can also be provided within these descriptions. A proxy gives an illusion of the real implementation to the client.
- *Discovery services* – A Request Broker manages service discovery on behalf of clients. Some form of a service description, such as a URL or service name, is used by the broker to lookup a potentially remote implementation that offers the service. The broker will return to the client application a handle to the external service in the address space of the client so that the client can seamlessly invoke services on the handle via the Proxy. Depending on whether *pass by reference* or *pass by value* semantics are used, the execution of client requests may occur remotely on the server machine or locally in the client's address space, respectively.
- *Marshaling engine*– A Request Broker often specifies an encoding scheme or a serialization format for representing application data. Often there exist tools that read the interface descriptions of services and synthesize code that can marshal and unmarshal all the data types that are defined as serializable in the interface descriptions. This tool-generated code, which is usually called a *stub*, is linked into the fabric of the Request Broker, which manages data marshaling and unmarshaling on behalf of the application.
- *Concurrency control* – A server application may require finer-grained control on concurrency for scalability and performance. Request Brokers often provide sophisticated concurrency control mechanisms, such as *thread pools*, to handle application requests in a scalable and concurrent manner.
- *Object lifetime manager* – A server application may host multiple different services to optimally utilize resources. Often these services are provided in the form of objects implemented in a programming language. The Request Broker must manage several objects simultaneously in the system according to the policies dictating their lifetimes, e.g., transient or persistent. Other policies that tradeoff between memory footprint and performance include the *activate-on-demand* policy, which activates a service only on demand and for the duration of the request. This conserves resources but impacts performance.
- *Request demultiplexing and dispatching* – A broker may manage several hundreds of objects at a time. When requests arrive at a server, they must be efficiently demultiplexed and dispatched to the right object implementation.
- *Connection management and Transport Adapter* – Since applications are distributed they must communicate over different networking protocols. Request Brokers often enable applications to



Request Broker. Figure 1. Request broker functional architecture.

configure the choice of protocol to use for communication. For example, the unpredictable behavior of TCP/IP will not suffice for real-time applications in which case special transport protocols must be used. But the application must be shielded from these differences. Transport adapters [4] can provide these capabilities. For the transport protocol used, appropriate connection management capabilities are required. For example, connections may need to be purged periodically to conserve resources.

Apart from the functional architecture, a taxonomy of the capabilities provided by a Request Broker can be developed along the following orthogonal dimensions.

(1) *Communication models* – This dimension includes different classifications, such as (i) Remote Procedure Calls (RPC) versus Message Passing; (ii) Synchronous versus Asynchronous communications; (iii) Request/Response versus Anonymous Publish/Subscribe semantics; (iv) Client/Server versus Peer-to-Peer.

(2) *Location transparency* – This dimension includes mechanisms, such as object references or URLs, used by the Request Broker to hide the details of the service.

(3) *Type system support* – This dimension includes the richness of the data types that can be exchanged between the communicating entities, and the semantics of data exchange. For example, Request Brokers can support passing objects by value or by reference or both.

(4) *Interoperability and portability* – This dimension includes the degree of heterogeneity supported by the Request Brokers. For example, technologies such as CORBA are both platform- and language-independent, which makes them widely applicable but requires complex mapping between the platform- and language-independent representations to platform- and language-specific artifacts. Often this impacts the richness of data types that can be exchanged. On the other hand some technologies are language-dependent, e.g., Java RMI, or platform-dependent, e.g., DCOM and .NET Remoting.

(5) *Quality of service (QoS) support* – This dimension includes the capabilities provided by the Request Broker to support different QoS requirements of applications. Some brokers may be tailored to support real-time support while others are customized for persistence and transaction support.

Key Applications

Request Brokers are at the heart of distributed computing, and span a wide range of application domains including telecommunications, finance, healthcare, industrial automation, retail, grid computing, among others. Request Brokers with enabling technologies to support real-time applications have also been used to build distributed real-time and embedded systems found in domains, such as automotive control, avionics mission computing, shipboard computing, space mission computing, among others. Request Brokers are also found in systems that require management and scheduling of storage resources, or in large, event-based or content-management systems.

Future Directions

As applications become more complex, heterogeneous, and require multiple different and simultaneous quality of service properties, such as real-time, fault tolerance and security, the responsibilities of the Request Broker increase substantially. Brokering capabilities themselves will need to be distributed requiring coordination among the distributed brokers. Supporting multiple QoS properties will require design-time tradeoffs due to the mutually conflicting objectives of each QoS property. Run-time resource management will be a key in supporting the QoS properties since applications are increasingly demanding autonomic capabilities.

Experimental Results

Experimental research on Request Brokers is proceeding along the discussion articulated in the future trends. Service oriented computing is requiring Request Brokers to move beyond simple client-server or peer-to-peer computing to more advanced scenarios where the brokering capabilities are required across entire application workflows.

URL to Code

The following URLs provide more information on different Request Broker technologies and sample code.

CORBA is standardized by the Object Management group (http://www.omg.org).

Java RMI is a technology of Sun Microsystems (http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp).

DCOM and .NET Remoting are technologies from Microsoft (http://msdn2.microsoft.com/en-us/library/ms809340.aspx and http://msdn2.microsoft.com/en-us/library/2e7z38xb.aspx, respectively).

DCE is a technology standardized by the Open Group (http://www.opengroup.org/dce/).

Cross-references

- ▶ CORBA
- ▶ DCE
- ▶ DCOM
- ▶ .NET Remoting
- ▶ RMI
- ▶ Service Oriented Architecture
- ▶ Subsumed by Windows Communication Framework

Recommended Reading

1. Bakken D.E. Middleware. In Encyclopedia of Distributed Computing. J. Urban and P. Dasgupta (eds.). Kluwer, Dordrecht, 2001.
2. Bernstein P.A. Middleware: a model for distributed system services. Commun. ACM, 39(2):86–98, 1996.
3. Buschmann F., Meunier R., Rohnert H., Sommerlad P., and Stal M. Pattern-Oriented Software Architecture – A System of Patterns. Wiley, New York, 1996.
4. Gamma E., Helm R., Johnson R., and Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1995.
5. Schmidt D.C. Evaluating Architectures for Multi-threaded CORBA Object Request Brokers. Commun. ACM Special Issue on CORBA, 41(10):54–60, October 1998.

Residuated Lattice

VILÉM NOVÁK
University of Ostrava, Ostrava, Czech Republic

Synonyms

Structure of truth values

Definition

The residuated lattice is a basic algebraic structure accepted as a structure of *truth values* for fuzzy logic and fuzzy set theory. In general, it is an algebra

$$\langle L, \vee, \wedge, \otimes, \rightarrow, 0, 1\rangle$$

where L is a support, $\vee$ and $\wedge$ are binary lattice operations of join and meet, 0 is the smallest and 1 is the greatest element. The $\otimes$ is additional binary operation of *product* that is associative and commutative, and $a \otimes 1 = a$ holds for every $a \in L$. The $\rightarrow$ is a binary *residuation* operation that is adjoined with $\otimes$ as follows:

$$a \otimes b \le c \quad \text{if and only if} \quad a \le b \rightarrow c$$

for arbitrary elements $a, b, c \in L$. The residuation operation is a generalization of the classical implication.

Key Points

The residuated lattice is naturally ordered by the classical lattice ordering relation defined by

$$a \le b \quad \text{if and only if} \quad a \wedge b = a.$$

In general, of course, there can exist incomparable elements in L. A typical property of the residuation operation is $a \rightarrow b = 1$ iff $a \le b$. In words: $a \le b$ iff the degree of implication $a \rightarrow b$ is equal to 1.

A typical example of residuated lattice is the *standard Łukasiewicz MV-algebra*

$$\langle [0, 1], \max, \min, \otimes, \rightarrow, 0, 1\rangle$$

where $a \otimes b = \max\{0, a+b-1\}$ is Łukasiewicz product (conjunction) and $a \rightarrow b = \min\{1, 1-a+b\}$ is Łukasiewicz implication.

Another widely used residuated lattice is an MTL-algebra, where $L = [0,1]$, $\otimes$ as some left continuous *t-norm* (see TRIANGULAR NORMS) and $\rightarrow$ is the corresponding residuation.

Every boolean algebra is a residuated lattice. Thus, a special case of residuated lattice is also the boolean algebra for classical logic

$$\langle \{0, 1\}, \vee, \wedge, \otimes, \rightarrow, 0, 1\rangle$$

where $\otimes = \wedge$ (i.e., $\otimes$ coincides with minimum) and $\rightarrow$ is the classical boolean (material) implication.

Both $\wedge$ as well as $\otimes$ are natural interpretations of logical conjunction. This means that there are two, in general different, conjunctions in fuzzy logic. In classical logic they coincide, though. The residuation $\rightarrow$ is natural interpretation of implication.

Negation is defined by

$$\neg a = a \rightarrow 0.$$

This operation coincides with classical negation in boolean algebra for classical logic, i.e., $\neg 0 = 1$ and $\neg 1 = 0$. In the standard Łukasiewicz MV-algebra (based on $[0,1]$) this operation reduces to $\neg a = 1 - a$.

The *biresiduation* operation is defined by

$$a \leftrightarrow b = (a \rightarrow b) \wedge (b \rightarrow a).$$

This operation coincides with with classical equivalence in boolean algebra for classical logic. In fuzzy logic, it is used as a natural interpretation of logical equivalence. In standard Łukasiewicz MV-algebra it gives $a \leftrightarrow b = 1 - |a - b|$, $a, b \in [0,1]$.

There are many more special kinds of residuated lattices. Except for boolean algebra, the most important are BL-algebra, Gödel algebra, product algebra and MV-algebra.

Cross-references

- ▶ Fuzzy Relation
- ▶ Fuzzy Set
- ▶ Triangular Norms

Recommended Reading

1. Esteva F. and Godo L. Monoidal t-norm based logic: towards a logic for left-continuous t-norms. Fuzzy Sets Syst., 124:271–288, 2001.
2. Hájek P. Metamathematics of Fuzzy Logic. Kluwer, Dordrecht, 1998.
3. Klement E.P., Mesiar R., and Pap E. Triangular Norms. Kluwer, Dordrecht, 2000.
4. Novák V. and Perfilieva I. Močkoř J. Mathematical Principles of Fuzzy Logic. Kluwer, Boston/Dordrecht, 1999.
5. Gottwald S. A Treatise on Many-Valued Logics. Research Studies, Baldock, Herfordshire, 2001.

Resource Allocation Problems in Spatial Databases

Donghui Zhang, Yang Du
Northeastern University, Boston, MA, USA

Synonyms

Facility-location problem

Definition

Assume that a franchise plans to open one or more branches in a state. How shall the locations of the new branches be allocated to maximally benefit the customers? Depending on whether some branches already exist and how to quantify the benefits to the customers, there are multiple forms of such resource allocation problems. In spatial databases, distance plays an important role. A customer is assumed to always visit the closest branch. Therefore it is beneficial to a customer if a new branch is opened at a location closer than her closest existing branch. The *max-inf optimal-location query* assumes the existence of a set of sites (already opened franchise branches), and aims to find a new location within a given area which benefits the largest number of customers. The *min-dist optimal-location query* also assumes the existence of a set of sites and aims to find a location for a new site which is optimal; but here the optimality is defined as minimization of the average distance from each customer to the nearest site. Compared with its max-inf counterpart, the min-dist optimal-location query takes into account the saved distance for each customer. The *k-medoid query* assumes that there does not exist any p as residential blocks) that minimize the average distance from every customer to the nearest picked location.

Historical Background

There exists an extensive literature in operations research on resource allocation problems, named the *facility location problems*. The most widely studied version is the *uncapacitated facility location (UFL)* problem.

Given a set of customers, a set of potential sites, a non-negative cost for opening each site, and a non-negative service cost between each site and a customer, the *UFL* problem is to find a subset of the potential sites so that the total cost (opening cost plus service cost) is minimum. Here, the term "uncapacitated" refers to the assumption that there is no limit on the number of customers that each site can serve.

The k-medoid problem is a variation of the UFL problem, where the set of customers and sites are identical, the opening cost of each site is zero, the service cost between a site and a customer is the distance between them, and there is an additional constraint that exactly *k* sites will be opened. The k-means problem is a related problem, where the potential sites can be anywhere.

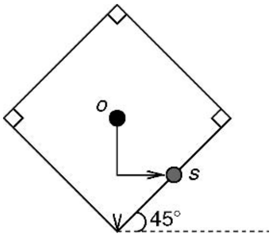
Facility location problems are (typically) NP-hard. Therefore research has focused on approximate algorithms with low computational complexity and small approximation errors. A recent survey of approximate algorithms for facility location problems appeared in [5]. Unlike spatial database research, the operations research assumes that all objects fit in memory and existing approaches typically scan through the dataset multiple times.

Foundations

When considering resource allocation problems in a large spatial database, the data usually reside in secondary storage and are indexed by spatial index structures so that scanning through all objects is avoided. The research goal is not to derive good asymptotic complexities, but to design efficient algorithms which, when applied to real datasets, incur a small number of I/O operations.

For both the max-inf and min-dist optimal-location queries under L_1 distance, there exist straightforward $O(n^2)$ solutions which find *exact* answers. Note that this differentiates the optimal-location queries from the NP-hard facility-location problem. However, such $O(n^2)$ algorithms are prohibitively slow in large spatial databases. Exploiting spatial index structures, [1] and [6] proposed solutions which are much faster (sub-linear in practice).

Similarly, the k-medoid query was examined in the context of spatial databases by utilizing spatial index structures. In particular, [2] relies on the clustering of higher-level index entries in an R-tree to avoid scanning through all objects.



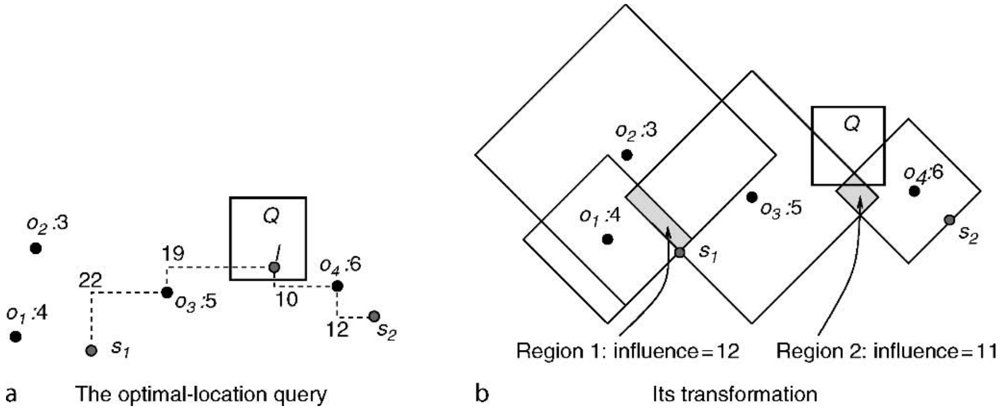
Resource Allocation Problems in Spatial Databases. Figure 1. The *nn_buffer* of an object (or customer) under L_1 metric is a diamond (*rotated square*).

Max-Inf Optimal-Location Query

The max-inf optimal location (OL) query aims to find the location of a new site which benefits (or influences) the largest number of customers. Here the *influence* of a location is the number of customers which are closer to the location than to any existing site. To answer the max-inf OL query under L_1 metric, Du et al. [1] introduced the concept of *nn_buffer* and reduced the problem into finding a location with maximum overlap among *nn_buffers* of objects. Formally, the *nn_buffer* of an object is defined as follows.

Definition *Given an object o and its closest site s, the nn_buffer of o is a contour such that* $\forall l$ *on the contour,* $d(l, o) = d(o, s)$.

In other words, a location l is inside $o.nn\_buffer$ if and only if o is closer to l than to any site. As shown in Figure 1, the *nn_buffer* of an object o under L_1 metric is a diamond. The object o contributes to the influence of a location l, if and only if l is inside the *nn_buffer* of o. Furthermore, if the coordinates are rotated by 45° counter-clockwise, the *nn_buffers* become axis-parallel squares. Therefore, given a query region Q, an optimal location is a location l inside Q which maximizes the total weight of overlapped *nn_buffers*. Here the weight of an object tells how important an object is. For instance, if the object is an apartment complex, its weight can be the number of residents living in it. Figure 2 gives an example of the query with four objects and two sites and its corresponding



Resource Allocation Problems in Spatial Databases. Figure 2. In (a), l is an optimal location, with influence 11. The transformation in (b) shows that any location in the intersection between Q and region 2 is an optimal location.

transformation. Based on this observation, three solutions are proposed to answer the max-inf OL query under L_1 metric.

The first one is an R-tree based solution. It assumes that an R-tree is used to index the set O of objects. Furthermore, the R-tree is augmented with some extra information. Every object stores the L_1 distance to its closest existing site, and every index entry stores the maximum such distance of objects in the sub-tree. The solution follows two steps. The first step is to retrieve those objects from the R-tree whose *nn_buffers* intersect with Q. The objects are identified in increasing order of X coordinate in the rotated coordinate, even though the R-tree was built in the original coordinate. The second step is a plane-sweep process, which consumes the objects streamed in from the first step and computes the weight of overlapped *nn_buffers*. A naive plane-sweep solution has $O(n^2)$ cost, where n is the number of retrieved objects. However, it can be improved to $O(n \log n)$ by using a data structure called *aggregation SB-tree*. After processing all retrieved objects, the algorithm reports the location with maximum overlap as the answer.

The second solution is based on a specialized aggregation index called the *OL-tree*. The *OL-tree* is a balanced, disk-based, dynamically updateable index structure extended from the k-d-B-tree. This index is built in the rotated coordinate. Objects to be inserted into the OL-tree are *nn_buffers* versus points in the original k-d-B-tree. Local optimal location information in each subtree is stored along with the index entry referencing the sub-tree. Such information enables efficient processing algorithms that can answer the max-inf optimal-location query without examining all *nn_buffers* intersecting the query region Q. In the best case, the algorithm only needs to examine the root node of the tree.

These two solutions have interesting tradeoffs between storage cost and query efficiency. The R-tree based solution utilizes the existing R-tree structure. The storage cost is linear to the number of objects. However, as the objects are not pre-aggregated, a query needs to examine all objects whose *nn_buffers* intersect with the query range Q. If Q has large size, the query performance is poor. On the other hand, the OL-tree is a specialized aggregation index, whose space overhead is much higher, but provides faster query processing.

The third solution combines the benefits of the previous two approaches. As in the R-tree based solution, it uses an R-tree to store the objects. But to guide the search, it uses a small, in-memory OL-tree-like index structure. This index is named the *Virtual OL-tree (VOL-tree)*. It resembles the top levels of an OL-tree and it does not physically store any *nn_buffer*. A leaf entry plays a similar role to an index entry. It corresponds to a spatial range, and it logically references a node that stores (pieces of) *nn_buffers* in that range. If necessary, these *nn_buffers* can be retrieved from the R-tree dynamically. To prune the search space, the VOL-tree stores some (but not all) aggregated information of the *nn_buffers* in each level. This solution provides the best trade-off between space overhead and computational costs, as shown experimentally in [1].

Min-Dist Optimal-Location Query

The min-dist OL query aims to find the location of a new site which minimizes the average distance from each customer to the nearest site. Zhang et al. [6] proposed a progressive algorithm to find the exact answer of the Min-Dist OL query under L_1 metric. The algorithm works in two steps. In the first step, it limits the candidate locations to finite locations, which are the intersections of certain horizontal and vertical lines. The second step is a recursive partition-and-refine step. In this step, it partitions the query range Q into a few cells (by using some of the vertical and horizontal lines), and calculates $AD(\cdot)$ for the corners of these cells. Here $AD(\mathrm{l})$ is denoted as the average distance from each customer to the nearest site after opening a new site l. The smaller $AD(\mathrm{l})$ is, the better the new location l is. For each cell, the algorithm estimates the lower-bound of $AD(\cdot)$ among all locations in the cell and prunes the cell if its lower bound is larger than the minimum $AD(\cdot)$ already found. It repeatedly partitions the unpruned cells into smaller cells to further refine the result, until the actual optimal location is found.

To understand how the candidate locations are limited to finite locations, consider the example in Figure 3. The black dots are the objects and the thick-bordered rectangle is the query region Q. The shadowed region is composed of the horizontal extension of Q and the vertical extension of Q. The following theorem guarantees that the optimal location can be limited to finite candidates.

Theorem *Consider the set of horizontal (and vertical) lines that gothrough some object in the horizontal (and vertical) extension of Q or go through some corner of Q.*

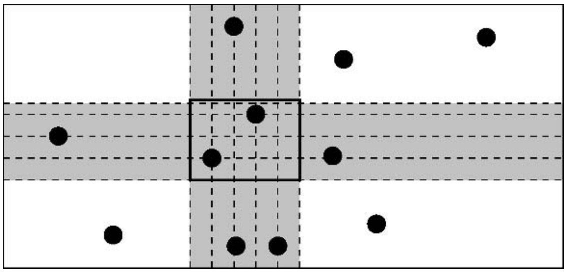
The min-dist optimal location under L_1 metric can be found among the intersection points of these lines.

In order to prune some cells, the second step of the algorithm utilizes a novel lower-bound estimator for $AD(\cdot)$ of all locations in a cell. The following theorem describes how the estimator works.

Theorem *Let the corners of a cell C be* c_1, c_2, c_3, and c_4, *where* $\overline{c_1 c_4}$ *is a diagonal. Let the perimeter of C be p.*

$$\max\{\frac{AD(c_1)+AD(c_4)}{2},\frac{AD(c_2)+AD(c_3)}{2}\}-\frac{p}{4}$$

is a lower bound of AD(l) for any location $l \in C$. The progressive algorithm, which starts with one cell (the query region Q) and keeps partitioning it into smaller cells and trying to prune cells from the processing queue, has two advantages. First, it avoids computing $AD(\cdot)$ for all candidate locations, as all candidate locations inside a pruned cell are ignored. Second, it responds fast. An approximate answer is reported at the very beginning (after computing $AD(\cdot)$ for the four corners of Q), within a guaranteed error bound. As the algorithms runs, the candidate optimal location and the associated error rate are improved progressively, until the exact optimal location is found.



Resource Allocation Problems in Spatial Databases. Figure 3. The candidate locations are limited to the intersections of the dashed lines.

Disk-Based k-Medoid Query

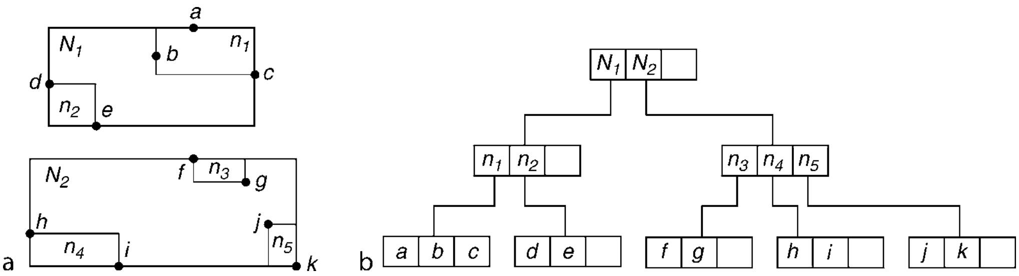
Mouratidis et al. [3] studied the k-medoid problem in large spatial databases. They assume that the data objects are spatial points indexed by an R-tree and the service cost between two points is defined by the distance between them. They propose the TPAQ (Tree-based PArtition Querying) algorithm that achieves low CPU and I/O cost. The TPAQ algorithm avoids reading the entire dataset by exploiting the grouping properties of the existing R-tree index.

Initially, TPAQ traverses the R-tree in a top-down manner, stopping at the topmost level that provides enough information for answering the given query. In the case of the k-medoid problem, TPAQ finds the topmost level with more than (or equal to) k entries. For instance, if $k = 3$ in the tree of Figure 4, TPAQ stops at level 1, which contains five entries, n_1 through n_5.

Next, TPAQ groups the entries of the partitioning level into k slots (i.e., groups.) To utilize the grouping properties of the R-tree index, TPAQ augments each retrieved entry n_i with a weight w and a center c. Here, the weight w is the number of points in the subtree referenced by n_i and the center c is the geometric centroid of the entry, assuming that the points in the sub-tree are uniformly distributed. To merge the initial entries into exactly k groups, TPAQ utilizes space-filling curves to select k seed entries which capture the distribution of points in the dataset. Then, each remaining entry is inserted into the slot whose weighted center is the closest to its center.

The final step of TPAQ is to pick k medoid objects, one from each group; TPAQ reports the weighted center of each group as the corresponding medoid.

Since the k-medoid problem is NP-hard, like any other practical algorithm TPAQ can only provide an approximate answer to the query. Experiments show that, compared to previous approaches, TPAQ achieves



Resource Allocation Problems in Spatial Databases. Figure 4. R-tree example.

comparable or better quality, at a small fraction of the cost (seconds as opposed to hours).

In [3], the authors also extended the above method to solve the medoid-aggregate query, where k is not known in advance. Given a user-specified parameter T, the medoid-aggregate query determines the smallest value of k and computes k-medoids such that the average distance from each object to the nearest medoid is smaller than T. The solution extends from TPAQ in two ways. First, without knowing k in advance, the R-tree top-down traversal stops at the level decided by the spatial extents and the expected cardinality of the entries. Second, multiple passes over the initial entries might be required to find the proper way of grouping the entries into slots.

Key Applications

Location-Based Services

Research on resource allocation problems is expected to help enhance the performance of location-based services based on the geographic proximity of clients to potential facilities of interest.

Spatial Decision Making

The efficient solutions to the resource allocation problems can provide valuable information to decision making. For example, to help provide candidate locations of a new branch.

Future Directions

One future research direction is to extend the solutions to other practical distance metrics. The existing solutions to both the max-inf and min-dist OL queries assume L_1 distance. Extending to L_2 (i.e., Euclidean) distance and road network distance is desirable. Similarly, extending the existing k-medoid solution to handle road network distance is interesting. Another future direction is to extend the queries to allow both the pre-existence of certain sites and the ability to find multiple locations. Existing solutions to the OL queries are limited to only one optimal location, and existing solutions to the k-medoid query are limited to zero existing site. Relaxing/avoiding these limitations will lead to more practical problems and solutions. The third future direction is to consider moving objects instead of static ones. In this case, the optimal locations and k-medoids should be continuously monitored over a set of moving objects [3,4].

Cross-references

▶ Nearest Neighbor Query
▶ Reverse Nearest Neighbor

Recommended Reading

1. Du Y., Zhang D., and Xia T. The optimal-location query. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005, pp. 163–180.
2. Mouratidis K., Papadias D., and Papadimitriou S. Medoid queries in large spatial databases. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005, pp. 55–72.
3. Papadopoulos S., Sacharidis D., and Mouratidis K. Continuous medoid queries over moving objects. In Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases, 2007, pp. 38–56.
4. U L.H., Mamoulis N., and Yiu M.L. Continuous monitoring of exclusive closest pairs. In Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases, 2007, pp. 1–19.
5. Vygen J. Approximation algorithms for facility location problems (lecture notes). Technical Report, University of Bonn, Germany, 2005, pp. 1–59.
6. Zhang D., Du Y., Xia T., and Tao Y. Progressive computation of the min-dist optimal-location query. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 643–654.

Resource Description Framework

Michael Weiss
Carleton University, Ottawa, ON, Canada

Synonyms

RDF

Definition

RDF (Resource Description Framework) is a language for making statements about resources, i.e., for representing metadata [7]. Here, the term resource is intentionally used very broadly in the sense of any entity that can be uniquely identified by a URI (Universal Resource Identifier). This includes traditional web resources (such as web pages or images), as well as physical objects (such as devices) and humans about which the statements are made.

Historical Background

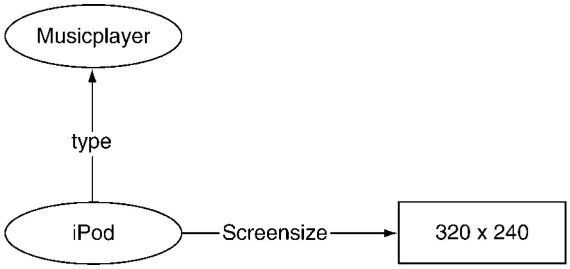
In 1998, Tim Berners-Lee described RDF in his vision for how metadata should be represented on the Semantic Web [4]. The first W3C Recommendation of RDF as a standard for encoding metadata was published in 1999. Several related efforts include the Meta Content Format (MCF) developed at Apple between 1995 and 1997, and the Dublin Core metadata standard [5] originally defined in 1995.

Foundations

Resource Description Framework

RDF is a core component of the layered specification of the semantic web (also known as the "layered cake"). It has its basis in the theory of semantic networks. The relationships between the elements of an RDF model can be presented in a number of ways: as a directed labeled graph (helpful to visualize the model), as object-attribute-value or resource-property-value triplets, or through an XML binding.

Figure 1 shows a simple example of an RDF graph. Oval nodes represent resources, and arrows between them represent properties. Property values that are themselves resources are shown as ovals, while literal values (such as strings) are shown as boxes. This figure makes two statements about an iPod music player: it is a type of music player, and it has a screen size of 320 × 240 pixels. The same information is shown as a set of resource-property-value triplets in Fig. 2. The resource `iPod` has two properties, `rdf:type` and `profile:ScreenSize`. The value of the property `rdf:type` is the type `MusicPlayer`, and the volume of the property `profile:ScreenSize` is the strip 320 × 240.



Resource Description Framework. Figure 1. Example of an RDF graph.

Resource	*Property*	*Value*
`iPod`	`rdf:type`	`MusicPlayer`
`iPod`	`profile:ScreenSize`	`320x240`

Resource Description Framework. Figure 2. Resource-property-value triplets.

Finally, Fig. 3 shows a corresponding XML binding. The XML binding starts with several namespace declarations, one for the RDF syntax, the other for the RDF schema that defines a vocabulary for describing mobile devices. This schema describes that a `MusicPlayer` is a kind of `MobileDevice`, and that `MobileDevice` resources have certain properties such as `ScreenSize`. The description of the `iPod` resource itself is contained in an `rdf:Description` element. It specifies a unique id through the `rdf:id` property, and a list of property values. As Fig. 2 illustrates, an `rdf:Description` element can contain multiple statements. However, it is not required that these statements are made within the same description. In fact, this allows a resource to be described in a decentralized manner. For example, the description of an iPod above could be extended – in a separate document – to make a statement about the iPod's owner without affecting the original description.

Key Applications

Another key application is FOAF (Friend of a Friend), a vocabulary for describing people and their relationships. A FOAF description can also contain information such as the organizations and projects a person works for and contributes to, documents they have created, and images that show them (Brickley & Miller, 2007). Information about a person can be distributed, leveraging the capability of RDF for decentralized representation. A person's

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:profile="http://www.examples.org/profile/mobile-device#">
  <rdf:Description rdf:ID="iPod">
    <rdf:type rdf:resource="#MusicPlayer"/>
    <profile:ScreenSize>320x240</profile:ScreenSize>
  </rdf:Description>
</rdf:RDF>
```

Resource Description Framework. Figure 3. XML binding for RDF.

own FOAF profile on their home page might only list the projects they work on and some of their friends. Other sites can provide additional information about that person using FOAF, for example, the page of a conference could list images of conference attendees. RDF has been applied to a variety of uses: annotating documents (e.g., Dublin Core [5]), representing device profiles (e.g., CC/PP [6]) and recently as exchange format in web services (e.g., MusicBrainz [9]). A popular use is in the RDF Site Summary (RSS) format, one of several RSS formats for lightweight content syndication in blogs [8].

Cross-references

▶ Dublin Core
▶ Metadata
▶ RDF Schema
▶ XML

Recommended Reading

1. Allemang D. and Hendler J. Semantic Web for the Working Ontologist: Modeling in RDF, RDFS and OWL. Morgan Kaufmann, 2008.
2. Berners-Lee T. What the semantic web can represent. Available online at: http://www.w3.org/DesignIssues/RDFnot.html, 1998.
3. Brickley D. and Guha R.V. RDF vocabulary description language: RDF schema. W3C recommendation. Available online at: http://www.w3.org/TR/rdf-schema/, 2004.
4. Brickley D. and Miller L. FOAF Vocabulary Specification. Available online at: http://xmlns.com/foaf/spec/, 2007.
5. Hillman D. Using dublin core. DCMI recommended resource. Available online at: http://dublincore.org/documents/usageguide/, 2005.
6. Klyne G., Reynolds F., Woordrow C., Ohto H., Hjelm J., Butler M., and Tran L. Composite capability/preference profiles (CC/PP): structure and vocabularies 1.0. W3C recommendation. Available online at: http://www.w3.org/TR/CCPP-struct-vocab/ 2004.
7. Manola F. and Miller E. RDF primer. W3C recommendation. Available online at: http://www.w3.org/TR/rdf-primer/, 2004.
8. RSS-DEV Working Group. RDF site summary (RSS) 1.0. Available online at: http://web.resource.org/rss/1.0/, 2000.
9. Swartz A. MusicBrainz: a semantic web service. IEEE Intell. Syst., 17(1):76–77, 2002.

Resource Description Framework (RDF) Schema (RDFS)

Vassilis Christophides
University of Crete, Heraklion, Greece

Synonyms

Conceptual schemas

Definition

An RDF schema (RDFS) is represented in the basic RDF model and provides (i) *abstraction* mechanisms, such (multiple) class or property subsumption and (multiple) classification of resources; (ii) *domain and range* class specifications to which properties can apply; (iii) *documentation facilities* for names defined in a schema.

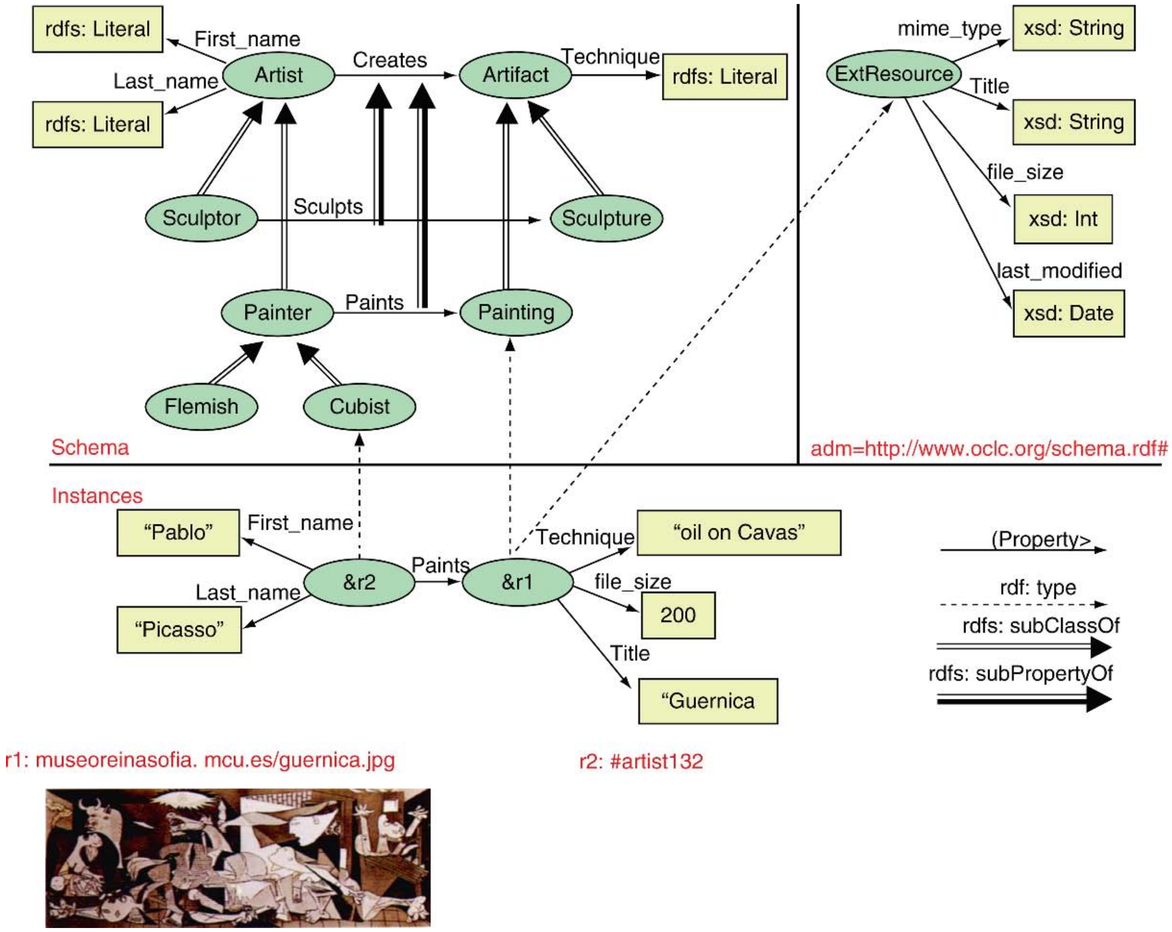
RDF/S follow the W3C design principles of interoperability, evolution and decentralization. In particular, it is possible to interconnect in an extensible way resource descriptions (by superimposing different statements using the same resource URIs) or schema namespaces (by reusing or refining existing class and property definitions) regardless of their physical location on the Web.

Key Points

Over the last decade, RDF and its accompanying RDFS specifications has been the subject of an extensive collaborative design effort. ([1]http://www.w3.org/RDF) RDF/S was originally developed as an application-neutral model to represent various kinds of descriptive information about Web resources, i.e., metadata. The W3C published the RDF Schema as Candidate Recommendation in 2000. Under the boost of the Semantic Web for transforming the Web into a universal medium for data, information, and knowledge exchange, refined versions of the RDF/S family of specifications have been published in 2004 [1–5]. As a matter of fact, RDF/S serves today as general purpose languages for consistent encoding, exchange and processing of available Web content, through a variety of syntax formats (XML or others). Their current design has been strongly influenced by recent advances in knowledge representation and aims to provide a simple *semantic layer* to the Web (no negation), as a base for more advanced query answering and reasoning services.

To present the core RDF/S modeling primitives, the example of a catalog of a cultural Portal (see Fig. 1) is used. To build this catalog, various cultural resources (e.g., Web pages of cultural sites) must be described from both a Portal and a Museum curator perspective. Figure 1 relies on an almost standard graphical notation, where nodes represent RDF/S resources (circles) or literals (rectangulars) while edges represent either user-defined (single arrows) or build-in (double and dashed arrows) properties.

The lower part of Fig. 1 depicts the description of an image (e.g., museoreinasofia.mcu.es/guernica.



Resource Description Framework (RDF) Schema (RDFS). Figure 1. A cultural Portal Catalog in RDF/S.

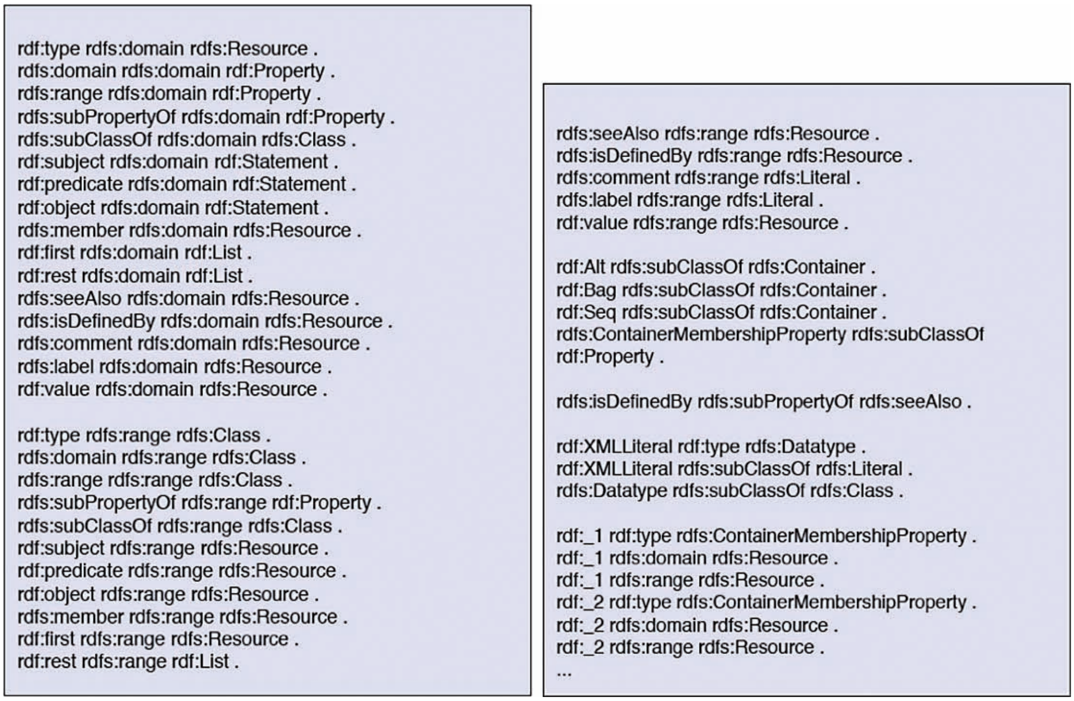
jpg) available on the Web. Hereforth the prefix “&” is used to abbreviate the involved resource URIs (e.g., &**r1**). In a first place, &**r1** is described from a Portal perspective as instance of the class named adm: ExtResource (uniqueness of names is ensured by using as prefix the corresponding schema URIs, like adm or cult in the example). More precisely, the statement (a triple in the RDF jargon) <&**r1**,rdf:type,adm:ExtResource> asserts that the resource &**r1** (subject) is of type (predicate) adm:ExtResource (object). Additionally &**r1** is stated to have two properties: one with name title and value the string “Guernica” (triple <&**r1**,adm:title,Guernica>) and, the other, with name file_size and value the integer 200 (triple < &**r1**, adm:file_size,200>). Resource &**r1** is also asserted as an instance of the class named Painting (triple <&**r1**,rdf:type,cult:Painting>) having a property technique with the literal value (string) “oil on canvas” (triple <&**r1**,cult:technique,oil on canvas>). &**r1** is further described by considering additional resources as for instance, &**r2** whose identifier (#artist132) is local to the Portal, i.e., not universally accessible in the Web. Resource &**r2** is then classified under Cubist and has a property first_name with value “Pablo” and a property last_name with value “Picasso.” Finally, it is asserted that &**r2** and &**r1** are related thought the property paints (triple < &**r2**,cult:paints,&**r1**>).

The upper part of Fig. 1 depicts two RDF/S schemas (i.e., the namespace URIs **cult** = http://www.icom.com/schema.rdf# and **adm** = http://www.oclc.org/schema.rdf#) which define the classes and properties (i.e., the names) employed by the previous resource descriptions. In schema **cult**, the property creates(triple < cult:creates,rdf:type,rdf:Property>), is defined with domain the class Artist (triple < cult:creates,rdf:domain,cult:Artist>) and range the class Artifact (triple < cult:createsrdf:range,cult:Artifact>).

Note that properties are binary relations used to represent *attributes* of resources (e.g., technique with range a literal type (XML Schema datatypes could be used in this respect.) as well as *relationships* between resources (e.g., creates with range a class of resources). Furthermore, both classes and properties can be organized into *taxonomies* carrying inclusion semantics (multiple subsumption is also supported). For example, the class Painter subsumes Cubist (<cult:Cubist, rdfs:subClassof, cult:Painter>) while the property paints is subsumbed by creates (<cult:paints,rdfs:subPropertyof,cult:creates>). According to the RDFS semantics [3], rdfs:subClassof (or rdfs:subPropertyof) relations are transitive (and reflexive), thus enabling one to infer more triples (not depicted in Fig. 1.) than those explicitly stated (e.g., < cult:Cubist, rdfs:subClassof,cult:Artist > < &**r2**,rdf:type, cult:Painter > < &**r2**,rdf:type,cult:Artist > etc.).

To summarize, RDF/S properties are by default are *unordered* (e.g., there is no order between the properties first_name and last_name), *optional* (e.g., the property mime-type is not used) or with *multi-occurrences* (e.g., &**r2** may have two properties paints). Cardinality constraints for property domains/ranges (as well as inverse properties and Boolean class expressions) can be captured in more expressive ontology languages such as OWL. Furthermore, utility properties like rdfs:label, rdfs:comment, rdfs:isDefinedBy and rdfs:seeAlso are also available for documenting the development of a schema. Although not illustrated in Fig. 1, RDF/S also support structured values called *containers* for grouping statements, namely rdf:Bag (i.e., multi-sets) and rdf:Sequence (i.e., tuples), as well as, higher-order statements (i.e., *reified statements* whose subject or object can be another RDF statement). Figure 2 summarizes RDF/S axiomatic triples.

The RDF/S modeling primitives are reminiscent of knowledge representation languages (like Telos). Compared to traditional object or relational database models, RDF/S blurs the distinction between schema and instances. RDF/S schemas are *descriptive* (and not prescriptive designed by DB experts), *interleaved with the instances* (i.e., may cross abstraction layers when a resource is related through a property with a class) while may be *large* (compared to the size of instances). In particular, unlike objects (or tuples) RDF/S resources *are not strongly typed*:



```
rdf:type rdfs:domain rdfs:Resource .
rdfs:domain rdfs:domain rdf:Property .
rdfs:range rdfs:domain rdf:Property .
rdfs:subPropertyOf rdfs:domain rdf:Property .
rdfs:subClassOf rdfs:domain rdfs:Class .
rdf:subject rdfs:domain rdf:Statement .
rdf:predicate rdfs:domain rdf:Statement .
rdf:object rdfs:domain rdf:Statement .
rdfs:member rdfs:domain rdfs:Resource .
rdf:first rdfs:domain rdf:List .
rdf:rest rdfs:domain rdf:List .
rdfs:seeAlso rdfs:domain rdfs:Resource .
rdfs:isDefinedBy rdfs:domain rdfs:Resource .
rdfs:comment rdfs:domain rdfs:Resource .
rdfs:label rdfs:domain rdfs:Resource .
rdf:value rdfs:domain rdfs:Resource .

rdf:type rdfs:range rdfs:Class .
rdfs:domain rdfs:range rdfs:Class .
rdfs:range rdfs:range rdfs:Class .
rdfs:subPropertyOf rdfs:range rdf:Property .
rdfs:subClassOf rdfs:range rdfs:Class .
rdf:subject rdfs:range rdfs:Resource .
rdf:predicate rdfs:range rdfs:Resource .
rdf:object rdfs:range rdfs:Resource .
rdfs:member rdfs:range rdfs:Resource .
rdf:first rdfs:range rdfs:Resource .
rdf:rest rdfs:range rdf:List .
```

```
rdfs:seeAlso rdfs:range rdfs:Resource .
rdfs:isDefinedBy rdfs:range rdfs:Resource .
rdfs:comment rdfs:range rdfs:Literal .
rdfs:label rdfs:range rdfs:Literal .
rdf:value rdfs:range rdfs:Resource .

rdf:Alt rdfs:subClassOf rdfs:Container .
rdf:Bag rdfs:subClassOf rdfs:Container .
rdf:Seq rdfs:subClassOf rdfs:Container .
rdfs:ContainerMembershipProperty rdfs:subClassOf
rdf:Property .

rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .

rdf:XMLLiteral rdf:type rdfs:Datatype .
rdf:XMLLiteral rdfs:subClassOf rdfs:Literal .
rdfs:Datatype rdfs:subClassOf rdfs:Class .

rdf:_1 rdf:type rdfs:ContainerMembershipProperty .
rdf:_1 rdfs:domain rdfs:Resource .
rdf:_1 rdfs:range rdfs:Resource .
rdf:_2 rdf:type rdfs:ContainerMembershipProperty .
rdf:_2 rdfs:domain rdfs:Resource .
rdf:_2 rdfs:range rdfs:Resource .
...
```

Resource Description Framework (RDF) Schema (RDFS). Figure 2. RDF/S Axiomatic Triples.

- *RDF/S Classes do not define object or relation types*: an instance of a class is just a resource URI without any value/state (e.g., &**r1** is an instance of `Painting` regardless of any property associated to it);
- *RDF Resources may be instances of different classes* not necessarily pair wise related by subsumption: the instances of the same class may have associated quite different properties (e.g., see the properties of &**r1** which is multiply classified under the classes `ExtResource` and `Painting`);
- *RDF/S Properties are self-existent individuals* (i.e., decoupled from class definitions) which *may also be related through subsumption* (e.g., the property creates).

In addition, less rigid data models, such as those proposed for semi-structured databases, when they are not totally schemaless (such as OEM, UnQL), they cannot certainly exploit the RDF class (or property) subsumption taxonomies (as in the case of YAT). Finally, XML DTDs and Schemas have substantial differences from RDF schemas: (i) they cannot represent directed label *graphs* (Formally speaking, *RDF graph*s are not quite classical directed labeled graphs. First, a resource (e.g., paints) may occur both as a predicate (e.g., < &**r2**, cult:paints,&**r1**>) and a subject (e.g., < `cult:paints, rdf:domain, cult:Painter`>) of a triple. This compromises one of the more important aspects of graph theory: the intersection between the nodes and arcs labels must be empty. Second, in an RDF graph a predicate (e.g., `rdfs:subPropertyof`) may relate other predicates (<`cult:paints,rdfs:subPropertyof,cult:creates`>). Thus, the resulting structure is not a graph in the strict mathematical sense, because the set of arcs must be a subset of the Cartesian product of the set of nodes. There is an ongoing research on formalizing RDF using adequate graph models (e.g., bipartite graphs, directed hypergraphs).) (vs. rooted labeled *trees*); (ii) they cannot distinguish between *entity labels* (e.g., `Artist`) and *relationship labels* (e.g., `creates`); and (iii) they *constrain the structure* of XML documents, whereas an RDF/S schema simply *defines the vocabulary of class and property names* employed in RDF descriptions.

Cross-references

▶ Ontologies
▶ Resource Description Framework
▶ Semantic Web

Recommended Reading

1. Beckett D. RDF/XML syntax specification (revised). W3C recommendation. Available online at: http://www.w3.org/TR/rdf-syntax-grammar/, 2004.
2. Brickley D. and Guha R.V. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation. Available online at: http://www.w3.org/TR/rdf-schema/, 2004.
3. Hayes P. RDF Semantics. W3C Recommendation. Available online at: http://www.w3.org/TR/rdf-mt/, 2004.
4. Klyne G. and Carroll J. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation. Available online at: http://www.w3.org/TR/rdf-concepts/, 2004.
5. Manola F. and Miller E. RDF Primer W3C Recommendation. Available online at: http://www.w3.org/TR/rdf-primer/, 2004.

Resource Identifier

GREG JANÉE
University of California-Santa Barbara, Santa Barbara, CA, USA

Synonyms

Document identifier; UUID; GUID; Uniform resource identifier; URI

Definition

In a networked information system, a *resource identifier* is a compact surrogate for a resource that can be used to identify, retrieve, and otherwise operate on the resource. An identifier typically takes the form of a short textual string. An identifier must be *resolved* to yield the associated resource.

Key Points

Resource identifiers can be broadly characterized as either *locations*, which identify resources by where they reside, or *names*, which identify resources by properties intrinsic to the resources [2]. This distinction is not absolute, and identifiers can exhibit characteristics of both classes. Nevertheless, the distinction is useful in defining the relationship between identifiers and resources. Consider:

> *Can two distinct, yet identical resources have the same identifier?*
>
> *If a resource changes, must its identifier change?*

If the answer to these questions is yes, then the identifiers should be considered names; if no, locations. To take two well-known examples, International Standard

Book Numbers (ISBNs) are names, while HTTP URLs on the World Wide Web are locations.

Uniqueness

Uniqueness is the property that an identifier resolves to a single resource. The converse property – that every resource is identified by a single identifier, i.e., that identifier "aliasing" is avoided – is generally desirable, but is often not enforceable in systems that allow free generation of identifiers.

Broadly speaking, two approaches have been employed to guarantee uniqueness. The first is to incorporate into each identifier unique characteristics of the identified resource, for example a content-based signature, or characteristics of the context in which the resource and/or identifier system reside, for example a network address and timestamp. UUIDs incorporate both types of characteristics. The second approach is to acquire identifiers from an "authority" that maintains a centralized store of previously generated identifiers (identifier–resource associations are often stored as well). For scalability such systems are often arranged hierarchically so that a root authority, located at a well-known address, may delegate identifier generation and resolution requests to distributed sub-authorities. DNS and the Handle system are two well-known examples of this approach.

Persistence

Persistence is the property that an identifier continues to reference the associated resource over time. Strictly speaking, persistence is not a property of an identifier, or even a property at all; it's an outcome of the commitment of the operator of the identifier resolution system. A persistent identifier system is one that attempts to address known risks to persistence.

The risk of identifier breakage due to resource movement is universally mitigated by employing indirection: identifiers identify intermediate quantities which are maintained by resource owners to track current resource locations. In principle the indirection may be hidden from users, but for scalability reasons it is typically exposed. For example, the persistent uniform resource locator (PURL) system employs HTTP's redirection mechanism. The risk of breakage due to resource renaming has been mitigated in some systems by issuing so-called "semantics-free" identifiers; for example, DOIs are strings of digits with no external referent. However, the benefit of this approach must be balanced by the inscrutability of such identifiers to humans. Other notable persistent identifier systems include OpenURLs, which identify objects by metadata constraints, i.e., by intrinic resource properties; "robust hyperlinks," which append content-based signatures to locations, specifically URLs; and archival resource keys (ARKs), which incorporate a protocol for obtaining resource persistence guarantees and policies.

Other Properties

Additional desirable properties of resource identifiers include global scope, global uniqueness, extensibility, machine readability, recognizability in text, and human transcribability [3]. Identifiers that are subject to transcription errors may benefit from having error-correcting codes incorporated into them.

Cross-references

- ▶ Citation
- ▶ Digital Signatures
- ▶ Distributed Architecture
- ▶ Object Identity

Recommended Reading

1. Hilse H.-W. and Kothe J. (2006). Implementing persistent identifiers: overview of concepts, guidelines and recommendations. London/Amsterdam: Consortium of European Libraries and European Commission on Preservation and Access. http://nbn-resolving.de/urn:nbn:de:gbv:7-isbn-90-6984-508-3-8
2. Jacobs I. and Walsh N. (eds.) (2004). Architecture of the World Wide Web, Volume One. http://www.w3.org/TR/webarch/
3. Sollins K. and Masinter L. Functional Requirements for Uniform Resource Names. IETF RFC 1737, 1994. http://www.ietf.org/rfc/rfc1737.txt

Resource Scheduling

▶ Query Load Balancing in Parallel Database Systems

Restart Processing

▶ Crash Recovery

Restricted Data

▶ Statistical Disclosure Limitation For Data Access

Result Display

Catherine Plaisant
University of Maryland, College Park, MD, USA

Synonyms

Result display; Result overview; Preview; Data visualization

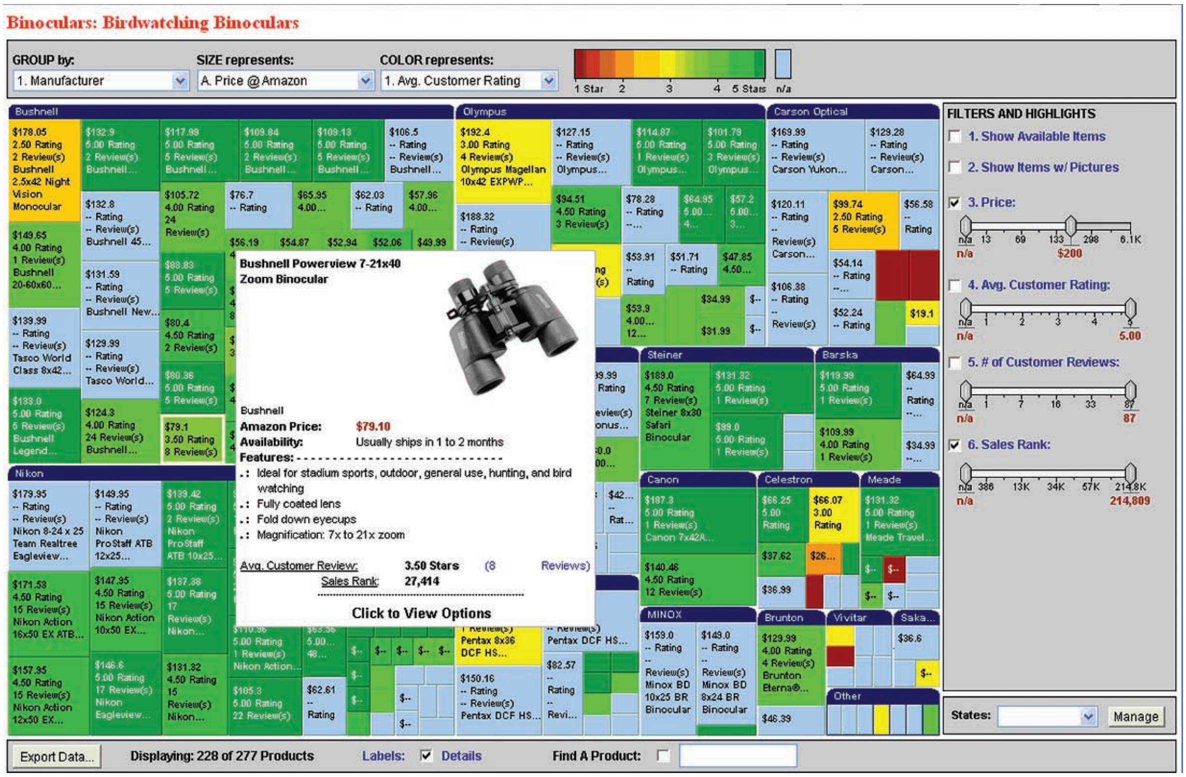
Definition

After formulating and initiating a search in a database, users review the results. The complexity of this task varies greatly depending on the users' needs, from selecting one or more top ranked items to conducting a complex analysis of the results in the hope of discovering an unknown phenomena. Displaying results includes providing and overview of the results and previews of items, manipulating visualizations, changing the sequencing of the results, adjusting the size of the results, clustering results by topic or attribute values, providing relevance feedback, examining individual items, and presenting explanatory messages such as restating the initial query.

Key Points

Displaying results is part of a dynamic and iterative decision-making process in which users initiate queries, review results and refine their queries. Users scan objects rapidly to determine whether to examine them more closely, or move on in the dataset. This process continues until the information need is satisfied, or the search is abandoned [3]. A visual result display typically includes interactive widgets to further filter the results, blurring the boundary between search and result display, e.g., dynamic queries entirely blend searching and result display [1].

The visual display of results relies on previews and overviews of the items returned by the search [2]. Graphical overviews indicate scope, size or structure and help gauge the relevance of items retrieved. Those



Binoculars: Birdwatching Binoculars

GROUP by: 1. Manufacturer
SIZE represents: A. Price @ Amazon
COLOR represents: 1. Avg. Customer Rating

Bushnell Powerview 7-21x40 Zoom Binocular

Bushnell
Amazon Price: $79.10
Availability: Usually ships in 1 to 2 months
Features:
.: Ideal for stadium sports, outdoor, general use, hunting, and bird watching
.: Fully coated lens
.: Fold down eyecups
.: Magnification: 7x to 21x zoom

Avg. Customer Review: 3.50 Stars (8 Reviews)
Sales Rank: 27,414

Click to View Options

FILTERS AND HIGHLIGHTS
☐ 1. Show Available Items
☐ 2. Show Items w/ Pictures
☑ 3. Price: n/a – $200
☐ 4. Avg. Customer Rating: n/a – 5.00
☐ 5. # of Customer Reviews: n/a – 87
☑ 6. Sales Rank: n/a – 214,809

States: Manage

Export Data... Displaying: 228 of 277 Products Labels: ☑ Details Find A Product: ☐

Result Display. Figure 1. After querying for birdwatching binoculars, users can review the results of their query using the Hive Group's treemap. Each box corresponds to a pair of binoculars and the size of the box is proportional to its price. Green boxes are best sellers, gray indicates unavailability. Results are grouped by manufacturer. Using the sliders on the right, users can filter results e.g., showing only items under $200.

overviews can vary from simple bar charts displaying the distribution of results over important attributes such as size or type, or consist of specialized visualizations such as interactive geographical maps, timelines, node link diagrams, conceptual topic maps etc. When no natural representation exist, more abstract overviews of the results can be used e.g., a treemap (Fig. 1). Multiple overviews are tightly coupled to facilitate synchronized browsing in multiple representations.

Previews consist of samples or summaries and help users select a subset of the results for detail review. Multimedia database require specialized preview mechanisms such as thumbnail browsers or video summaries allowing zooming and scene skipping. Both previews and overviews help users define more productive queries as they learn about the content of the database.

Users should be given control over what the size of the result set is, which fields are displayed, how results are sequenced (alphabetical, chronological, relevance ranked, and how results are clustered (by attribute value, by topics). One strategy involves automatic clustering and naming of the clusters for example in Vivisimo. Studies show that clustering according to more established and meaningful hierarchies such as the open directory might be effective. Translations may be proposed. Finally users need to gather information for decision making, therefore results need to be saved, annotated, sent by email or used as input to other programs such as visualization and statistical tools.

Cross-references

▶ Information Retrieval
▶ Video Querying
▶ Visualization

Recommended Reading

1. Ahlberg C. and Shneiderman B. Visual information seeking: tight coupling of dynamic query filters with starfield displays. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1994, pp. 313–317.
2. Greene S., Marchionini G., Plaisant C., and Shneiderman B. Previews and overviews in digital libraries: designing surrogates to support visual Information-seeking, J. Am. Soc. Inf. Sci., 51 (3):380–393, 2000.
3. Shneiderman B. and Plaisant C. Designing the User Interface, (4th edn.). Addison-Wesley, 2005.

Result Overview

▶ Result Display

Result Ranking

▶ Web Search Relevance Ranking

Retrieval Models

▶ Web Search Relevance Ranking

Retrieval Models for Text Databases

▶ Structured Text Retrieval Models

Retrospective Event Processing

Opher Etzion
IBM Research Lab-Haifa, Haifa, Israel

Definition

Retrospective event processing is the detection of patterns on past events i.e., not done when the event occur, this can be done as part of existing event processing.

Historical Background

While the concept of "event pattern" typically refers to the events on-the-move, there are cases, in which it is required to use these patterns on past events, this typically happens in one of the following cases:

1. *Situation Reinforcement:* An event pattern designates the possibility that a business situation has occurred; in order to provide positive or negative reinforcement, as part of the on-line pattern detection, there is a need to find complementary pattern in order to assert or refute the occurrence of the situation.
2. *Retrospective Context.* In regular cases, contexts to start and look for patterns start with a certain event, and go forward until either some event occurs, or the context expires. There are cases in which the arrival of an event indicates the end of the context, however the context has not been monitored in the forward-looking way, and need to be monitored backwards.
3. *Patterns as queries.* Patterns are higher-level abstractions relative to SQL queries; in some cases

it is more convenient to use patterns, as higher-level languages on top of queries.

The issue of querying past information have been introduced in the area of *temporal databases* in which dealt with maintaining, querying and even updating past (and future) information. This included regular database queries or updates, however provided the infrastructure for storing past events.

The emerging of the event processing area have lead to the development of *event processing patterns* that initially was applied on the current events, retrospective event processing is the next logical step in getting the patterns on the past. It should be noted that there are three approaches to implementation of retrospective pattern language:

1. Extending pattern language to look at retrospective contexts
2. Adding patterns as an extension to SQL
3. Providing a hybrid language

Foundations

Taking the approach of extending event processing patterns for past information, this materializes in the following areas:

- *Retention policies:* in order to enable retrospective processing, event should be available beyond the original context of their processing, this leads to issues such as retention policies and vacuuming policies.
- *Grid storage:* one of the emerging areas in storing events and states for pre-determined term is storing events on in-memory stores on the grid.
- *Extending the notion of context:* the notion of temporal context should be extended to include past time intervals.
- *Automatic translation to SQL:* Assuming that the events are stored on a database, the pattern language should be translated to SQL. Again, it should be noted that an alternative approach is to include pattern extensions to SQL.

Key Applications

Use Cases for Situation Reinforcement

Anti-Money Laundering A person that has deposited (in aggregate) more than $20,000 within a single working day is a SUSPECT in money laundering. To reinforce the suspicion the following retrospective patterns are sought:

- There has been a period of week within the last year in which the same person has deposited (in aggregate) $50,000 or more and has withdrawn (in aggregate) at least $50,000 within the same week.
- The same person has already been a "suspect" according to this definition within the last 30 business days.

If any of these patterns are satisfied – the event "confirmed suspect" is derived.

The Greedy Seller Alert An electronic trade site provides the opportunity to customers to offer items for sale, but letting them conduct a bid, and provide bid management system (using a CEP system, of course). One of the services it provides to the customer is "alert on expensive sales":

If there have been at least two bidders, however, and none of them have matched the minimum price of the seller then this may be an indication of "too expensive bid."

To reinforce it, if at least two-thirds of the past bids of the same sellers have also resulted in a "too expensive bid" situation, then reinforce and send the seller a notification "you are too greedy."

Monitored Patient Alert A patient is hooked up to multiple monitors, the monitors are uncorrelated and each of them issues an alert when a certain threshold is passed (either up or down), results in most cases being false alarms. The physician can set up a "global monitoring system" which checks a pattern over recent time, e.g.,

- An alert has been given from the blood pressure monitor
- and Reinforcement condition:
  - If fever is more than 103F despite medication taken less than 2 hours ago, and blood pressure is strictly increasing in the last five measures then alert nurse.

Use Cases for Retrospective Contexts

Smart Retail The detected pattern is – "no item of a certain product reached the checkout in the last hour." This is a pattern that is detected hourly. If it is detected there is a retrospective context opened for this hour to check two retrospective patterns:

- If during that hour more than five customers took an item of that product from the shelf, but returned it after they have taken a competitive product (the information can be obtained by RFID tag on each item, an RFID reader for putting and removing items from the shopping cart).
- If no customer has taken an item of the product from the shelf, and did not take a competitor's product either.

Luggage Handling The reported event is – Luggage did not arrive.

Retrospective context – start at luggage check-in time.

- Collect all events related to this luggage (using the tag reading at various points).
- If no events found – notify the source airport to trace in their video tracking system.

Utilities Billing System Identified situation – customer has not paid 30 days after due date.

- Find out over the last billing cycle – has the customer addressed the customer center around issues with this bill, and obtain status.
- Look at the customer billings over the last year – determine maximal and average days of late in paying.
- Look at customers with the same Zip-code over the last billing cycle and determine the percentage of non-payment, late-payment (to determine if they are mail has significant delays in that area).

Use Cases for Patterns as Queries

Stock Trends Find all stocks that during the last month have satisfied the following conditions:

- The stock closing values at the end of the day were strictly increasing over a period of five consecutive working days, anywhere during this month.
- The stock value in the beginning at the end of the 5 days value was at least 30% more than its value at the beginning of the 5 days period.

Fraud Detection in On-Line Gaming Determine the case in which one commits identity theft, on the expense of the victim, consistently loose money to his partners in on-line poker game as a way of fraud (the example came from Oracle).

- Find a "consistent looser" – in a session that includes at least 30 poker games, in which a person loses in all the games, and the total loss is more than $20K.
- Find a person who has been a "consistent looser" at least twice, on distinct set of games.

Inventory Management The level of inventory is determined according to consumption prediction, to improve the inventory management, there are:

- Pre-planned orders that may be cancelled if the level of inventory does not justify order.
- Emergency orders that are requested if inventory goes below threshold in an unpredicted time.

The following patterns are requested:

- Find products for which "emergency order" has been requested at least twice during a period of the period of 30 days that ended in a date Last-Day (variable name) and no cancellation of pre-planned order during that period.
- Find products in which within 30 days there were cancellation of pre-planned order followed by an emergency order.
- Find products in the monitored period which all pre-planned orders have been cancelled, and no emergency orders occurred.

Cross-references

- ▶ Complex Event
- ▶ Context
- ▶ Event and Pattern Detection over Streams
- ▶ Event Pattern Detection
- ▶ Temporal Database

Recommended Reading

1. Arasu A., Babu S., and Widom J. The CQL continuous query language: semantic foundations and query execution. VLDB J., 15(2):121–142, 2006.
2. Deng M., Prasad Sistla A., and Wolfson O. Temporal conditions with retroactive and proactive updates. In Proc. 1st Int. Workshop on Active and Real-Time Database Syst., 1995, pp. 122–141.
3. Etzion O., Gal A., and Segev A. Retroactive and proactive database processing. In Proc. 4th Int. Workshop on Research Issues on Data Eng., 1994, pp. 126–131.
4. Gal A. and Etzion O. A multiagent update process in a database with temporal data dependencies and schema versioning. IEEE Trans. Knowl. Data Eng., 10(1):21–37, 1998.
5. Luckham D, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, 2002.

6. Won J. and Elmasri R. Representing retroactive and proactive versions in bi-temporal databases. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 85–94.
7. Yang Y., Pierce T., and Carbonell J.G. A study of retrospective and on-line event detection. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 28–36.

Reverse Nearest Neighbor Query

Dimitris Papadias[1], Yufei Tao[2]

[1]Hong Kong University of Science and Technology, Hong Kong, China

[2]Chinese University of Hong Kong, Hong Kong, China

Synonyms

Reverse nearest neighbor search; RNN query

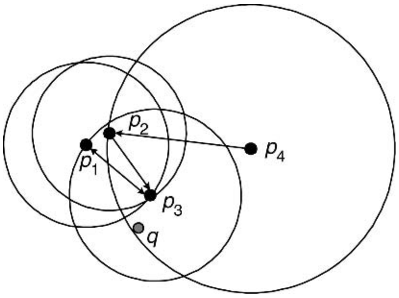
Definition

Given a multi-dimensional dataset P and a point q, a *reverse nearest neighbor* (RNN) query retrieves all the points $p \in P$ that have q as their nearest neighbor. The set RNN(q) of reverse nearest neighbors of q is called the *influence set* of q. Formally, RNN(q) = $\{p \in P \mid \neg \exists p' \in P$ such that $dist(p,p') < dist(p,q)\}$, where *dist* is a distance metric (Euclidean distance is assumed in the following examples).

The definition can also be extended to *reverse k nearest neighbors* (R*k*NN). Specifically, a R*k*NN query retrieves all the points $p \in P$ that have q as one of their k nearest neighbors. In this case, R*k*NN (q) = $\{p \in P \mid dist(p,q) \leq dist(p,p_k)$, where p_k is the k-th NN of $p\}$.

Historical Background

Reverse nearest neighbor queries were proposed in [4] and have received considerable attention due to their importance in several applications involving decision support, resource allocation, profile-based marketing, etc. Figure 1 shows an example R2NN query. The dataset P contains 4 points each associated with a circle covering its two nearest neighbors (NNs), e.g., the two NNs of p_4 (p_2, p_3) are in the circle centered at p_4. The result of a R2NN query q includes the "owners" of the circles that contain q, i.e., R2NN(q) = $\{p_3, p_4\}$. Let kNN(q) be the set of k nearest neighbors of q. Note that $p \in k$NN(q) does not necessarily imply $p \in$ RkNN(q), and vice versa. For instance, 2NN(q) = $\{p_1, p_3\}$, but p_1 does not belong to R2NN(q). On the other hand, although $p_4 \in$ R2NN(q), p_4 is not in 2NN(q).



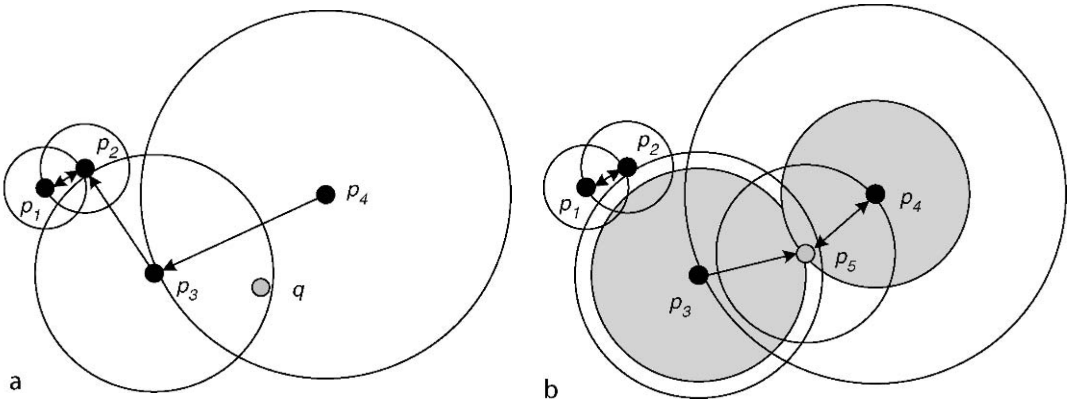
Reverse Nearest Neighbor Query. Figure 1. 2NN and R2NN examples.

Other versions of the problem include: (i) *continuous* RNN [1], where P contains linearly moving objects with fixed velocities, and the goal is to retrieve all RNNs of q for a future interval; (ii) *stream* RNN [5], where data arrive in the form of streams, and the goal is to report aggregate results over the RNNs of a set of query points, and (iii) *bichromatic* RNN [10] where, given two data sets P_1, P_2 and a query point $q \in P_1$, the goal is to find all the points $p_2 \in P_2$ that are closer to q than to any other object in P_1, i.e., $d(q, p_2) < d(p_1, p_2)$ for any $p_1 \in P_1$ and $p_1 \neq q$.

Foundations

Algorithms for RNN processing can be classified in two categories depending on whether they require preprocessing, or not. For simplicity, this section describes all methods for single RNN retrieval in 2D space, assuming that the dataset P contains points indexed by an *R-tree*. Their applicability to arbitrary values of k and dimensionality will be discussed at the end of the section.

The first RNN method, *KM* (Algorithms are referenced according to the author initials.) [4], pre-computes for each data point p its nearest neighbor NN(p). Then, it represents p as a *vicinity circle* (p, *dist*(p,NN(p))) centered at p with radius equal to the Euclidean distance between p and its NN. The MBRs of all circles are indexed by an *R-tree*, called the RNN-tree. Using the RNN-tree, the reverse nearest neighbors of q can be efficiently retrieved by a point location query, which returns all circles that contain q. Figure 2(a) illustrates *KM* using four data points, each associated with a vicinity circle. Since q falls in the circles of p_3 and p_4, the result of the query is RNN(q) = $\{p_3, p_4\}$. Because

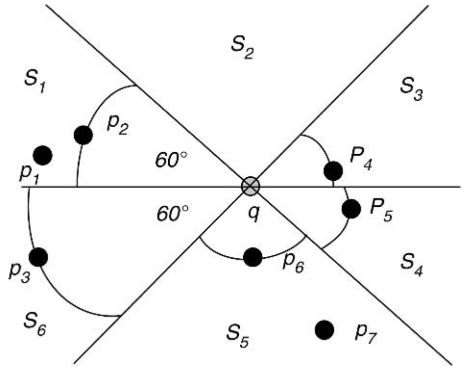


Reverse Nearest Neighbor Query. Figure 2. Illustration of *KM*.

the RNN-tree is optimized for RNN, but not NN search, *KM* uses an additional (conventional) *R-tree* on the data points for nearest neighbors and other spatial queries.

In order to avoid the maintenance of two separate structures, *YL* [13] combines the two indexes in the RdNN-tree. Similar to the RNN-tree, a leaf node of the RdNN-tree contains vicinity circles of data points. On the other hand, an intermediate node contains the MBR of the underlying points (not their vicinity circles), together with the maximum distance from every point in the subtree to its nearest neighbor. As shown in the experiments of [13], the RdNN-tree is efficient for both RNN and NN queries because, intuitively, it contains the same information as the RNN-tree and has the same structure (for node MBRs) as a conventional R-tree. *MVZ* [7] is also based on pre-computation. The methodology, however, is applicable only to 2D spaces and focuses on asymptotical worst case bounds (rather than experimental comparison with other approaches).

The problem of *KM*, *YL*, *MVZ*, and all techniques that rely on pre-processing, is that they cannot deal efficiently with updates. This is because each insertion or deletion may affect the vicinity circles of several points. Consider Fig. 2(b), where a new point p_5 needs to be inserted in the database. First, a RNN query is performed to find all objects (in this case p_3 and p_4) that have p_5 as their new nearest neighbors. Then, the vicinity circles of these objects are updated in the index. Finally, the update algorithm computes the NN of p_5 (i.e., p_4) and inserts the corresponding circle. Similarly, each deletion must update the vicinity circles of the affected objects. In order to alleviate the problem, Lin et al. [6] propose a method for bulk insertions in the RdNN-tree.

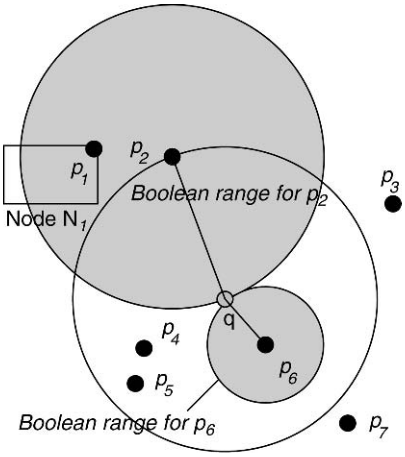


Reverse Nearest Neighbor Query. Figure 3. Illustration of *SAA*.

SAA [9] eliminates the need for pre-computing all NNs by utilizing some interesting properties of RNN retrieval. Consider Fig. 3, which divides the space around a query q into six equal regions S_1 to S_6. Let p be the NN of q in some region S_i; it can be proved that (i) either $p \in \text{RNN}(q)$ or (ii) there is no RNN of q in S_i. For instance, in Fig. 3 the NN of q in S_1 is point p_2. However, the NN of p_2 is p_1. Consequently, there is no RNN of q in S_1 and it is not necessary to search further in this region. The same is true for S_2 (no data points), S_3, S_4 (p_4, p_5 are NNs of each other) and S_6 (the NN of p_3 is p_1). The actual result is $\text{RNN}(q) = \{p_6\}$. Based on the above property, *SAA* adopts a two-step processing method. First, six constrained NN queries [2] retrieve the nearest neighbors of q in regions S_1 to S_6. These points constitute the *candidate* result. Then, at a second step, a *nearest neighbor query* is applied to find the NN p' of each candidate p.

If $dist(p,q) < dist(p,p')$, p belongs to the actual result; otherwise, it is a false hit and discarded.

The number of regions to be searched for candidate results increases exponentially with the dimensionality, rendering *SAA* inefficient even for three dimensions. *SFT* [8] follows a different approach that: (i) finds (using an R-tree) the K NNs of the query q, which constitute the initial candidates; (ii) it eliminates the points that are closer to some other candidate than q; (iii) it applies *boolean range queries* on the remaining candidates to determine the actual RNNs. Consider, for instance, the query of Fig. 4 assuming that K (a system parameter) is 4. *SFT* first retrieves the 4 NNs of q: p_6, p_4, p_5 and p_2. The second step discards p_4 and p_5 since they are closer to each other than q. The third step uses the circles $(p_2, dist(p_2,q))$ and $(p_6, dist(p_6,q))$ to perform two boolean ranges on the data R-tree. The difference with respect to conventional range queries is that a boolean range terminates immediately when (i) the first data point is found, or (ii) the entire side of a node MBR lies within the circle. For instance, N_1 contains at least a point within the range. Thus, p_2 is a false hit and *SFT* returns p_6 as the only RNN of q. The major shortcoming of the method is that it may incur false misses. In Fig. 4, although p_3 is a RNN of q, it does not belong to the 4 NNs of the query and will not be retrieved.

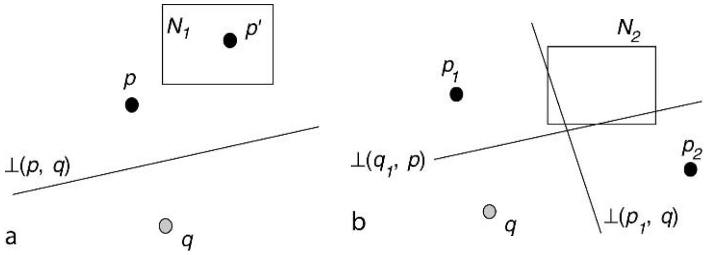


Reverse Nearest Neighbor Query. Figure 4. Illustration of *SFT*.

Similar to *SAA* and *SFT*, *TPL* [11] follows a filter-refinement framework. As opposed to *SAA* and *SFT* that require multiple queries for each step, the filtering and refinement processes are combined into a single traversal of the R-tree. In particular, *TPL* traverses the data R-tree and retrieves potential candidates in ascending order of their distance to the query point q because the RNNs are likely to be near q. Each candidate is used to prune node MBRs (data points) that cannot contain (be) candidates. For instance, consider the perpendicular bisector $\perp(p,q)$ between the query q and an arbitrary data point p as shown in Fig. 5(a). The bisector divides the data space into two half-planes: $\mathrm{PL}_q(p,q)$ that contains q, and $\mathrm{PL}_p(p,q)$ that contains p. Any point (e.g., p') in $\mathrm{PL}_p(p,q)$ cannot be a RNN of q because it is closer to p than q. Similarly, a node MBR (e.g., N_1) that falls completely in $\mathrm{PL}_p(p,q)$ cannot contain any candidate.

In some cases, the pruning of an MBR requires multiple half-planes. For example, in Fig. 5(b), although N_2 does not fall completely in $\mathrm{PL}_{p_1}(p_1,q)$ or $\mathrm{PL}_{p_2}(p_2,q)$, it can still be pruned since it lies entirely in the *union* of the two half-planes. In general, if p_1, p_2,...,p_{n_c} are candidate results, then any node whose MBR falls inside $\cup_{i=1\sim n_c}\mathrm{PL}_{p_i}(p_i,q)$ cannot contain any RNN result. The filter step terminates, when there are no more candidates inside the remaining (i.e., non-pruned data space). Each pruned entry is inserted in a *refinement set* S_{rfn}. In the refinement step, the entries of S_{rfn} are used to eliminate false hits.

Table 1 summarizes the properties of each algorithm. Pre-computation methods cannot efficiently



Reverse Nearest Neighbor Query. Figure 5. Illustration of half-plane pruning in *TPL*.

Reverse Nearest Neighbor Query. Table 1. Summary of algorithm properties

	Dynamic data	Arbitrary dimensionality	Exact result	*Arbitrary k*
KM, YL	No	Yes	Yes	No
MVZ	No	No	Yes	No
SAA	Yes	No	Yes	Yes
SFT	Yes	Yes	No	Yes
TPL	Yes	Yes	Yes	Yes

handle updates. *MVZ* is suitable only to 2D spaces, while *SAA* is practically inapplicable for three or more dimensions. *SFT* incurs false misses, the number of which depends on the parameter *K*: a large value of *K* decreases the false misses but increases significantly the processing cost. Regarding the applicability of the existing algorithms to arbitrary values of *k*, precomputation methods only support a specific value (typically equal to 1), used to determine the vicinity circles. *SFT* can be adapted for retrieval of R*k*NN by setting a large value of *K* ($\gg k$) and replacing the boolean with *count* queries (that return the number of objects in the query range instead of their actual ids). *SAA* can be extended to arbitrary *k* as discussed in [11]. *TPL* can handle dynamic data for arbitrary values of *k* and dimensionality.

Key Applications

A number of applications for RNN can be found in [4] and [14]. Examples include:

Profile-Based Marketing

Assume that a real estate company keeps profiles of its customer set *P* based on their goals, i.e., each customer is a point in a vector space defined by the features of interest (e.g., house area, neighborhood etc). When a new estate *q* enters the market, a RNN query could retrieve the clients for which *q* constitutes the closest match to their interests.

Decision Support Systems

Consider that a franchise wants to open a new branch at location *q* so that it attracts a large number of customers from competitors based on proximity. This can be modeled as a bichromatic RNN query where P_1 corresponds to the competitor set and P_2 to the customer dataset. The result for a potential location *q* is the set of customers that are closer to *q* than any competitor.

Peer-to-Peer Systems

Assume that a new user *q* enters a P2P system. A RNN query retrieves among the existing users, the ones for which *q* will become their new NN based on the network latency. In a collaborative environment, *q* would inform such users about its arrival, so that they could address future requests directly to *q*, minimizing the network cost. Furthermore, the set RNN(*q*) reflects the potential workload of *q*; thus by knowing this set, each peer could manage/control its available resources.

Future Directions

Stanoi et al. [10] solve bichromatic RNN queries using *R-trees* to prune the search space. Benetis et al. [1] extend the *SAA* algorithm for continuous RNN queries. Yiu et al. [14] deal with reverse nearest neighbors in large graphs. Tao et al. [12] focus on RNN processing in *metric spaces*. Kang et al. [3] discuss the continuous evaluation of RNN queries in highly dynamic environments.

Experimental Results

Tao et al. [11] contains a comprehensive comparison of *SAA*, *SFT* and *TPL*. Each of the following references, except for [7], also contains an experimental evaluation of the proposed algorithm.

Data Sets

A common benchmark for RNN queries in the Euclidean space is the Tiger dataset: *http://www.census.gov/geo/www/tiger/*

In addition, the DBLP graph has been used for RNN queries in large graphs [14], while road networks have been applied in [12] and [11].

Cross-references

- ▶ Metric Space
- ▶ Nearest Neighbor Query
- ▶ R-Tree (and Family)

Recommended Reading

1. Benetis R., Jensen C., Karciauskas G., and Saltenis S. Nearest neighbor and reverse nearest neighbor queries for moving objects. VLDB J., 15(3): 229–250, 2006.

2. Ferhatosmanoglu H., Stanoi I., Agrawal D., and Abbadi A. Constrained nearest neighbor queries. In Proc. 7th Int. Symp. Advances in Spatial and Temporal Databases, 2001.
3. Kang J., Mokbel M., Shekhar S., Xia T., and Zhang D. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 806–815.
4. Korn F. and Muthukrishnan S. Influence sets based on reverse nearest neighbor queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 201–212.
5. Korn F., Muthukrishnan S., and Srivastava D. Reverse nearest neighbor aggregates over data streams. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 814–825.
6. Lin K., Nolen M., and Yang C. Applying bulk insertion techniques for dynamic reverse nearest neighbor problems. In Proc. Int. Conf. on Database Eng. and Applications, 2003, pp. 290–297.
7. Maheshwari A., Vahrenhold J., and Zeh N. On reverse nearest neighbor queries. In Proc. Canadian Conf. Computational Geometry, 2002, pp. 128–132.
8. Singh A., Ferhatosmanoglu H., and Tosun A. High dimensional reverse nearest neighbor queries. In Proc. Int. Conf. on Information and Knowledge Management, 2003.
9. Stanoi I., Agrawal D., and Abbadi A., Reverse nearest neighbor queries for dynamic databases. In Proc. SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp. 44–53.
10. Stanoi I., Riedewald M., Agrawal D., and Abbadi A. Discovery of influence sets in frequently updated databases. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 99–108.
11. Tao Y., Papadias D., and Lian X. Reverse *k*NN search in arbitrary dimensionality. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 744–755.
12. Tao Y., Yiu M., and Mamoulis N. Reverse nearest neighbor search in metric spaces. IEEE Trans. Knowl. Data Eng., 18(9): 1239–1252, 2006.
13. Yang C. and Lin K. An index structure for efficient reverse nearest neighbor queries. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 482–495.
14. Yiu M., Papadias D., Mamoulis N., and Tao Y. Reverse nearest neighbors in large graphs. IEEE Trans. Knowl. Data Eng., 18(4): 540–553, 2006.

Reverse Nearest Neighbor Search

► Reverse Nearest Neighbor Query

RF

► Relevance Feedback
► Relevance Feedback for Text Retrieval

Rich Media

► Video

Right-Time Data Warehousing

► Active and Real-Time Data Warehousing

Risk-Utility Tradeoff

► Statistical Disclosure Limitation For Data Access

Rewriting Queries using Views

Chen Li
University of California-Irvine, Irvine, CA, USA

Definition

Given a query on a database schema and a set of views over the same schema, the problem of query rewriting is to find a way to answer the query using only the answers to the views. Rewriting algorithms aim at finding such rewritings efficiently, dealing with possible limited query-answering capabilities on the views, and producing rewritings that are efficient to execute.

Historical Background

Query rewriting is one of the oldest problems in data management. Earlier studies focused on improving performance of query evaluation [9], since using materialized views can save the execution cost of a query. In 1995, Levy et al. [10] formally studied the problem and developed complexity results. The problem became increasingly more important due to new applications such as data integration, in which views are used widely to describe the semantics of the data at different sources and queries posed on the global schema. Many algorithms have been developed, including the bucket algorithm [11] and the inverse-rules algorithm [7,15]. See [8] for an excellent survey.

Foundations

Formally, a query Q_1 is *contained* in a query Q_2 if for each instance of their database, the answer to Q_1 is always a subset of that to Q_2. The queries are *equivalent* if they are contained in each other. Let T be a database schema, and $\mathcal{V}$ be a set of views on T. The *expansion* of a query P using the views in $\mathcal{V}$, denoted by P^{exp}, is obtained from P by replacing all the views in P with their corresponding base relations. Given a query Q on T, a query P is called a *contained rewriting* of query Q using $\mathcal{V}$ if P uses only the views in $\mathcal{V}$, and P^{exp} is contained in Q as queries. P is called an *equivalent rewriting* of Q using $\mathcal{V}$ if P^{exp} and Q are equivalent as queries.

Examples: Consider a database with the following three relations about students, courses, and course enrollments:

```
Student(sid, name, dept);
Course(cid, title, quarter);
Take(sid, cid, grade).
```

Consider the following query on the database:

```
Query Q1: SELECT C.title, T.grade
FROM Student S, Take T, Course C
WHERE S.dept = 'ee' AND S.sid = T.sid AND
T.cid = C.cid;
```

The query asks for the titles of the courses taken by EE students and their grades. Queries and views are often written as conjunctive queries [4]. For instance, the above query can be rewritten as:

```
Q1(T, G) :- Student(S, N, ee), Take(S,
C, G), Course(C, T, Q).
```

Lower-case arguments (such as "ee") are used for constants, upper-case arguments (such as "T") for variables. The right-hand side of the symbol "`:-`" is the *body* of the query. It has three *subgoals*, each of which is an occurrence of a relation in the body. The constant "`ee`" in the first subgoal represents the selection condition. The variable `S` shared by the first two subgoals represents the join between the relations `Student` and `Take` on the student-id attribute. The variables `T` and `G` in the head of the query, which is the left-hand side of the symbol "`:-`", represent the final projected attributes.

Consider the following materialized views defined on the base tables:

```
Views: V1(S, N, D, C, G) :- Student(S, N,
D), Take(S, C, G);
 V2(S, C, T) :- Take(S, C, G), Course(C,
T, Q).
```

The SQL statement for the view definition of $V1$ is the following:

```
CREATE VIEW V1 AS
 SELECT S.sid, S.name, S.dept, T.cid,
T.grade
 FROM Student S, Take T
 WHERE S.sid = T.sid;
```

This view is the natural join of the relations `Student` and `Take`. Similarly, view $V2$ is the natural join of the relations `Take` and `Course`, except that the attributes about grades and quarters are dropped in the final results. The following is a rewriting of the query $Q1$ using the two views.

```
answer(T, G) :- V1(S, N, ee, C, G), V2(S,
C, T).
```

This rewriting takes a natural join of the two views on the attributes of student ids and course ids, then does a projection on the title and grade attributes. This rewriting can always compute the answer to the query on every instance of the base tables. In particular, after replacing each view in the rewriting with the body of its definition, the rewriting becomes the following expansion:

```
answer(T, G) :- Student(S, N, ee), Take
(S, C, G),
 Take(S, C, G'), Course(C, T, Q').
```

`G'` and `Q'` are fresh variables introduced during the replacements. This expansion is equivalent to the query, thus the rewriting is an equivalent rewriting of the query.

Now, assume in the definition of $V2$, there is another selection condition on the quarter attribute. The following is the view definition:

```
V2'(S, C, T) :- Take(S, C, G), Course(C,
T, fall2006).
```

That is, it only includes the information about the courses offered in the fall quarter of 2006. If only views $V1$ and $V2'$ are given, then the following is a rewriting of the query $Q1$:

```
answer(T, G) :- V1(S, N, ee, C, G),
V2'(S, C, T).
```

In particular, its expansion, which is obtained by replacing each view with the body of its definition, is the following:

```
answer(T, G) :- Student(S, N, ee), Take
(S, C, G),
 Take(S, C, G'), Course(C, T,
fall2006).
```

This expansion is contained in the original query, thus this rewriting is a contained rewriting of the query *Q*1. It is not an equivalent rewriting, since it does not include information about courses offered in other quarters. On the other hand, each fact in the answer to this rewriting is in the answer to the original query.

Suppose the view definition of *V*2 does not have the attribute about course ids. Then using this modified view and *V*1, there is no rewriting of the query, since the modified view does not have the course id to join with view *V*1. As another example, if the view definition of *V*1 does not keep the grade information, the following is the new view:

```
V1'(S, N, D, C) :- Student(S, N, D), Take
(S, C, G).
```

Using this new view and the original view *V*2, there is no rewriting to answer the query, since the views do not provide any information about grades, which is requested by the query. All these examples show that, when deciding how to answer a query using views, it is important to consider the conditions in the query and the views, including their selections, joins, and projections.

Algorithms. There are two classes of algorithms for rewriting queries using views: the first one includes the bucket algorithm [11] and its variants, and the second one includes the inverse-rules algorithm [7,15]. Notice that the number of possible rewritings of a query using views is exponential in the size of the query. Here the main idea of the bucket algorithm is explained using the running example, in which the query *Q*1 needs to be answered using the views *V*1 and *V*2. Its main idea is to reduce the search space of rewritings by considering each subgoal in the query separately, and deciding which views could be relevant to the query subgoal.

The bucket algorithm has two steps. In step 1, for each subgoal in the query, the algorithm considers each view definition, and checks if the body (definition) of the view also includes a subgoal that can be used to answer this query subgoal. For each view, if it includes a subgoal that can be unified with the query subgoal, and the query and the view are compatible after the unification, the corresponding head of the view definition is added to the bucket of this query subgoal. The following shows the buckets for the three query subgoals.

```
Student(S, N, ee): {V1(S, N, ee, C',
G')};
Take(S, C, G): {V1(S, N', D', C, G)};
Course(C, T, Q): {V2(S', C, T)}.
```

Each primed variable is a fresh variable introduced in the corresponding unification process. The bucket of the second query subgoal does not include the view *V*2 because the query subgoal requires the grade information be included in the answer, while the corresponding grade information in the view subgoal is not exported in the head of *V*2.

In step 2, the algorithm selects one view from each bucket, and combines the views from these buckets to construct a contained rewriting. The following is a contained rewriting:

```
Q1(T, G) :- V1(S, N, ee, C', G'), V1(S,
N', D', C, G), V2(S', C, T).
```

The final output of the algorithm is the union of contained rewritings in order to maximize the set of answers to the query using the views, since these rewritings could produce different pieces of information.

One main advantage of the bucket algorithm is that it can prune those views that do not contribute to a condition in the query, thus it can reduce the number of candidate rewritings to be considered. One limitation of the algorithm is that each query subgoal introduces a view in a rewriting. For instance, in the example above, view *V*1 could be used to answer the first two query subgoals. But the algorithm needs to use three view instances in each candidate rewriting, which requires more postprocessing steps to simplify this rewriting. In addition, the algorithm does not use the fact that if a view can be used to cover a query subgoal using a view variable that is not exported in the head of the view, then the view has to cover all the query subgoals that use the corresponding query variable. Based on these observations, a new algorithm, called MiniCon, was developed to make the rewriting process significantly more efficient [14]. A similar idea was used in the shared-bucket-variable (SVB) algorithm [13].

In some cases, especially in the context of data integration, where a view is a description of the content at a data source, the views could have limited query capabilities. For instance, imagine the case where the view *V*1 above is a materialized table, such that it can be accessed only if a student id is provided to the table, and the table can return its information about that student id. The table does not accept arbitrary queries such as "return all records," or "retrieve all information about students from the CS department." These limitations on the views present new challenges for the development of query-rewriting algorithms. The problem in this setting was studied in [16]. It is shown that the the inverse-rules algorithm [7] can handle such restrictions with minor modifications.

Other algorithms have been developed to study variants of the query-rewriting problem. The CoreCover algorithm [2] was developed for the problem of generating an *efficient* equivalent rewriting efficiently. There was also a study [1] for the case where the query and the views can have comparison conditions such as `salary` > `30K` and `year <= 2004`. The work in [6] studied how to compute a set of views with a minimal size to compute the answers to a set of queries. In some settings, applications need to find a rewriting called "maximally contained rewriting," which can compute the maximal set of answers to the query using the views. The problem is also different depending on whether the closed-world assumption is taken (as in data warehousing, in which each materialized view is assumed to include all the facts satisfying the view definition) or the open-word assumption is taken (as in data integration, in which each view includes a subset of the facts satisfying the view definition). In the literature there is another related problem called "query answering." See [3] for a comparison between "query rewriting" and "query answering."

Key Applications

The problem of rewriting queries using views is related to many data-management applications, including information integration [12,18], data warehousing [17], and query optimization [5].

Cross-references

▶ Answering Queries Using Views
▶ Closed-World Assumption (CWA)
▶ Data Integration
▶ Data Warehouse
▶ Global-as-View (GAV)
▶ Local-as-Views (LAV)
▶ Open-World Assumption (OWA)
▶ Query Containment
▶ Query Optimization

Recommended Reading

1. Afrati F.N., Li C., and Mitra P. Answering Queries Using Views with Arithmetic Comparisons. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2002, pp. 209–220.
2. Afrati F., Li C., and Ullman J.D. Generating Efficient Plans Using Views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 319–330.
3. Calvanese D., Giacomo G.D., Lenzerini M., and Vardi M.Y. View-Based Query Processing: On the Relationship Between Rewriting, Answering and Losslessness. In Proc. 10th Int. Conf. on Database Theory, 2005, pp. 321–336.
4. Chandra A.K. and Merlin P.M. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In Proc. 9th Annual ACM Symp. on Theory of Computing, 1977, pp. 77–90.
5. Chaudhuri S., Krishnamurthy R., Potamianos S., and Shim K. Optimizing Queries with Materialized Views. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 190–200.
6. Chirkova R. and Li C. Materializing views with minimal size to answer queries. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2003, pp. 38–48.
7. Duschka O.M. and Genesereth M.R. Answering Recursive Queries Using Views. In Proc. ACM SIGACT-SIGOPS 16th Symp. on the Principles of Dist. Comp., 1997, pp. 109–116.
8. Halevy A.Y. Answering queries using views: A survey. VLDB J., 10(4):270–294, 2001.
9. Larson P.Å. and Yang H.Z. Computing Queries from Derived Relations. In Proc. 11th Int. Conf. on Very Large Data Bases, 1985, pp. 259–269.
10. Levy A., Mendelzon A.O., Sagiv Y., and Srivastava D. Answering Queries Using Views. In Proc. 14th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 1995, pp. 95–104.
11. Levy A., Rajaraman A., and Ordille J.J. Querying Heterogeneous Information Sources Using Source Descriptions. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 251–262.
12. Li C. Query Processing and Optimization in Information-Integration Systems. Ph.D. Thesis, Computer Science Dept., Stanford Univ., 2001.
13. Mitra P. An algorithm for answering queries efficiently using views. In Proc. the 12th Australasian Database Conf. 2001, pp. 99–106.
14. Pottinger R. and Levy A. A Scalable Algorithm for Answering Queries Using Views. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000.
15. Qian X. Query folding. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 48–55.
16. Rajaraman A., Sagiv Y., and Ullman J.D. Answering Queries Using Templates with Binding Patterns. In Proc. 14th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 1995, pp. 105–112.
17. Theodoratos D. and Sellis T. Data warehouse configuration. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 126–135.
18. Ullman J.D. Information Integration Using Logical Views. In Proc. 6th Int. Conf. on Database Theory, 1997, pp. 19–40.

RMI

Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA

Synonyms

Remote method invocation

Definition

Java Remote Method Invocation (RMI) [1,2] is a Java language-based technology to achieve distributed computing among distributed Java virtual machines.

Key Points

Java RMI is a Java language-dependent technology for distributed computing. It provides seamless distributed communication between Java virtual machines. RMI uses object serialization and offers true object-oriented polymorphism even across distributed address spaces. Since RMI is based on Java, it brings the power of safety, concurrency and portability to distributed applications. In developing their applications, programmers must explicitly indicate which interfaces will be available as a remote service by extending the java.rmi.Remote interface.

A special quality of RMI is its ability to dynamically load new objects into an address space. For example, if a remote service has undergone change and extended its capabilities, it is feasible for RMI to dynamically load the new class in the client's address space.

Another attractive feature of RMI is its ability to allow entire behaviors of objects to be sent to remote entities. At the remote end, it is then feasible to activate a local copy of the passed object with its behavior. These techniques are useful in load balancing and faster response times.

RMI can allow clients behind firewalls to contact remote servers. This capability enables clients to reside within applets. RMI also provides interoperability with other broker technologies, such as CORBA, by supporting RMI over CORBA IIOP.

Cross-references

- ▶ Client-Server Architecture
- ▶ CORBA
- ▶ DCE
- ▶ DCOM
- ▶ J2EE
- ▶ .NET Remoting
- ▶ Request Broker
- ▶ SOAP

Recommended Reading

1. Sun Microsystems. Java Remote Method Invocation. 1996.
2. Sun Developer Network. Remote Method Invocation. Available at: http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp.

RNN Query

▶ Reverse Nearest Neighbor Query

Road Network Databases

▶ Road Networks

Road Networks

Cyrus Shahabi
University of Southern California, Los Angeles,
CA, USA

Synonyms

Spatial network databases; Road vector data; Road network databases

Definition

In *vector* space, the distance between two objects can be computed as a function of the components of the vectors representing the objects. A typical distance function for multidimensional vector spaces is the well-known Minkowski metric, with the Euclidean metric as a popular case for two-dimensional space. Therefore, the distance computation in multidimensional vector spaces is fast because its complexity depends on the number of dimensions which is limited to two or three in geospatial applications. However, with *road-networks*, the distance between two objects is measured by their *network distance*, i.e., the length of the shortest path through the network edges that connects the two locations. This computationally expensive network-based metric mainly depends on the connectivity of the network. For example, representing a road network as a graph with e weighted edges and v vertices, the complexity of the Dijkstra algorithm to find the minimum weighted path between two vertices is $O(e + vLogv)$. Therefore, several distance-based queries, such as nearest-neighbor queries, cannot be performed efficiently in road-networks. One option is to estimate the distance between two objects by their Euclidean distance. Unfortunately, as shown in [13], the Euclidean distance is not a good approximation of the network distance with real-world road-networks. Therefore, in the past several years, many studies have been investigating new techniques to index road-networks and/or pre-compute distances in order to expedite distance-based query processing on road-networks.

Historical Background

The main challenge with query processing in road-networks is the high complexity of network distance computation. This is important since many applications, especially those dealing with moving objects, require frequent, fast and on-the-fly computation of distances between a query point and several points of interests. The earliest work that studied this challenge of fast distance computation in road-network databases is by Shahabi et al. [13]. In this paper, the authors proposed an embedding technique to transform the road network to a high dimensional space in which fast Minkowski metrics can be utilized for distance measurement. The results are still approximation of the actual distances but much more accurate than using Euclidean distances on the points' geographical coordinates.

Besides this transformation-based approach to perform fast network distance computation, three other main approaches are based on either indexing the network or pre-computing the network distances or a hybrid of the two. A prominent work in the indexing category is by Papadias et al. [10], which proposes an architecture for road-networks that uses a disk-based network representation. Their approach is based on the fact that the current algorithms (e.g., Dijkstra) for computing the distance between a query object *q* and an object *O* in a network will automatically result in the computation of the distance between *q* and the objects that are (relatively) closer to *q* than *O*. This approach applies an optimized network expansion algorithm with the advantage that the network expansion only explores the objects that are closer to *q* and computes their distances to *q* during expansion. The advantages of this approach are: (i) it offers a method that finds the exact distance in networks, and (ii) the architecture can support other spatial queries like range search and closest pairs. There are other studies that similar to this work try to combine traditional spatial access methods with some sort of network representation and expansion method such as [3,5,7]. The main disadvantage of these network-expansion approaches is that they perform poorly when the objects are not densely distributed in the network because then they require to retrieve a large portion of the network for distance computation. A rather different indexing approach is proposed in [4], termed distance signature. This approach categorizes the distances between objects and network nodes into groups and then encodes these groups.

A representative work for pre-computation approaches is the work by Sankaranarayanan et al. [12], which proposes a framework called SILC for computing the shortest distance between vertices on a spatial network. The proposed framework pre-computes the shortest path between all pairs of vertices, which in turn results in fast distance computation. Examples of other studies that used some sort of pre-computation to expedite network distance computation are [6,2].

A hybrid approach that combines an indexing technique with pre-computation is proposed in [8]. This approach is based on partitioning and then indexing a large network to small *network Voronoi* regions, and then pre-computing distances both within and across the regions.

There are also many variations in the applications and query types on road-networks that require special-purpose index structures and/or query processing algorithms for efficient network distance computation. Some of these applications and query-types are reviewed below.

As mentioned, the fast computation of network distance becomes more critical when objects are moving. Most of the aforementioned techniques can indeed be used for querying points of interests from a moving object. However, some optimization can be performed if one wants to update the query results as the query point moves. One main representative query type here is the Continuous *k*-nearest-neighbor (C-*k*NN) queries. C-*k*NN on road-networks maintains the *k* nearest neighbors in network distance as the query object moves on the road network, which has its own unique challenges as opposed to C-*k*NN in Euclidean spaces. Sample studies focusing on moving objects in road-networks are [1,11].

Other novel applications include [9], which tries to address NN queries by proposing a novel travel time network that integrates both spatial networks and real-time traffic event information. In [15], the authors propose and solve Aggregate nearest neighbor queries (ANN) in the context of large road networks. In [14], the authors study the novel problem of optimal sequenced route (OSR) query in both vector and road-network spaces. The OSR query tries to find a route of minimum length starting from a given source location and passing through a number of typed locations in a specific sequence imposed on the types of the locations. The paper proposes a pre-computation approach to OSR query by exploiting the geometric properties of the solution space and relating it to additively weighted Voronoi diagrams.

Foundations

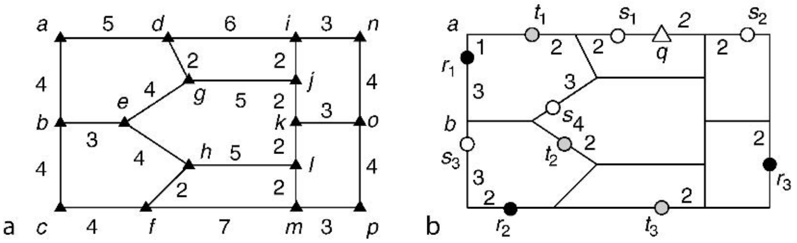
A popular class of queries in geospatial applications is the class of distance-based queries. To answer these queries, the distance between one or more query points or regions with some points or areas of interests must be computed. A frequently used member of this class is the *k* nearest neighbor (*k*NN) query where the *k* closest points to a query point are requested. For example, many car navigation systems provide the feature to ask for the *k* closest gas stations to the vehicle's current location. One way to answer this query is to estimate the distance by computing the Euclidean distance between the vehicle's geographical coordinates (i.e., latitude and longitude) and all the gas stations' coordinates in the vicinity. The problem with this approach is that the Euclidean distance corresponds to the *air-distance* between the vehicle and the gas station. Basically, if the car could have flown, then the Euclidean distance would have been the accurate distance between the car and the gas station. Unfortunately, most cars are restricted to the underlying road network. Hence, the actual distance between the car and the gas station depends on the connectivity of the underlying network and usually is very different than the Euclidean distance. In [13], it is shown that in real-world road-networks, Euclidean distance does not yield a good approximation of the network distance.

Formal Definition of the Problem

The challenge with computing network-distance is that the complexity of its computation depends on the number of vertices and edges of the underlying network, which is normally very large for real road-networks. Now imagine computing this distance continuously from a moving vehicle to several gas stations and one would appreciate the efforts in developing new techniques to compute this distance as fast as possible. To formally define this problem, first a formal model to represent road networks is provided and then using this model, the concept of network-distance is defined more accurately.

A road-network can be modeled as a weighted graph. Consider the weighted undirected (Many spatial networks consist of directed edges and hence must be modeled as directed graphs. Throughout this entry, undirected graphs are used for simplicity.) graph $G = (V, E)$ as the two sets V of vertices, and $E \subseteq V \times V$ of edges. Each edge of E, directly connecting vertices u and v, is represented as the pair $[u, v]$. Each vertex v represents a 2-d point $(v.x, v.y)$ in a geometric space (e.g., an intersection in a road network). Hence, each edge is also a line segment in that space (e.g., a road segment). A numeric weight (cost) w_{uv} is associated with the edge $[u, v]$. In road-networks, this is the distance or the travel time between intersections u and v. N refers to the space of points located on the edges/vertices of graph G. For a point $p \in N$ located on the edge $[u, v]$, $w_{up} = \frac{|up|}{|uv|} w_{uv}$ where $|uv|$ is the Euclidean distance between u and v. Figure 1a shows the graph model of a road network including the vertex set $V = \{a,\ldots,p\}$. Each edge of the graph is labeled by its weight. Figure 1b shows points $s_1,\ldots,s_4$, $r_1,\ldots,r_3$, and $t_1,\ldots,t_3$ on the edges of the same graph. As shown in the figure, point $r_1 \in N$ corresponds to the weights $w_{r_1 a} = 1$ and $w_{r_1 b} = 3$.

The main challenge with query processing in road-networks is the expensive cost of computing the



Road Networks. Figure 1. (a) Graph model of a road network, (b) 10 points of interest on the edges of the graph.

network distance between two points. Hence, *network distance* for road-networks is formally defined below.

Definition 1: Given a graph G, a *path P* from $p_1 \in N$ to $p_2 \in N$ is an ordered set $P = \{p_1, v_1,...,v_n, p_2\}$ consisting of a sequence of connected edges from p_1 to p_2. Here, p_1 and p_2 are located on the edges $[u, v_1]$ and $[v_n, w]$, respectively. Also, v_i is connected to v_{i+1} by the edge $[v_i, v_{i+1}]$ for $1 \leq i < n$. As shown in Fig. 1b, $P = \{t_1, a, b, s_3\}$ is a path from t_1 to s_3.

Definition 2: Given a path $P = \{p_1, v_1,...,v_n, p_2\}$, *Path Cost* of P, $pcost(P)$, is defined as the sum of the costs of all edges in P. Formally, for the path P,

$$pcost(P) = w_{p_1 v_1} + \sum_{i=1}^{n-1} w_{v_i v_{i+1}} + w_{v_n p_2}$$

In Fig. 1b, the cost of path $P = \{t_1, a, b, s_3\}$ is calculated as $pcost(P) = 3 + 4 + 1 = 8$. For the points $p_1, p_2 \in N$, $P_{p_1 p_2}$ is used to denote the *shortest path* from p_1 to p_2 in G; the path $P = \{p_1,...,p_2\}$with minimum cost $pcost(P)$.

Definition 3: Given the two points p_1 and p_2 in N, the *network distance* between p_1 and p_2, $D_n(p_1, p_2)$, is the cost of the shortest path between p_1 and p_2 (i.e., $D_n(p_1, p_2) = pcost(P_{p_1 p_2})$). For instance, $D_n(t_1, s_3) = 8$. The network distance $D_n(.,.)$ is non-negative and obeys identity, symmetry and the triangular inequality. Hence, together with N, it forms a metric space.

As discussed in the background section several techniques have been proposed to expedite the computation of this network-distance for efficient processing of distance-based queries. Here, one hybrid approach is reviewed which combines indexing the space (using voronoi diagrams) with a distance pre-computation approach and has shown to be superior in performance to its competitors [8].

A Voronoi-Based Solution for Road-Networks

A comprehensive solution for spatial queries in road-networks must fulfill the following real-world requirements: (i) be able to incorporate the network connectivity to provide exact distances between objects, (ii) efficiently answer the queries in real-time in order to support distance-based queries (e.g., *k*NN) for moving objects, (iii) be scalable in order to be applicable to usually very large networks, (iv) be independent of the density and distribution of the points of interest, (v) be adaptive to efficiently cope with database updates where nodes, links, and points of interest are added/deleted, and (vi) be extendible to consider query constraints such as direction or range.

In [8], the authors proposed a novel approach that fulfills the above requirements by reducing the problem of distance computation in a very large network, in to the problem of distance computation in a number of much smaller networks plus some additional table lookups.

The main idea behind this approach, termed Voronoi-based Network Nearest Neighbor (VN3), is to first partition a large network in to smaller/more manageable regions. This is achieved by generating a first-order *network* Voronoi diagram over the points of interest. Each cell of this Voronoi diagram is centered by one point of interest (e.g., a restaurant) and contains the nodes that are closest to that object in *network* distance (and not the Euclidean distance). Next, the intra and inter distances for each cell are pre-computed. That is, for each cell, the distances between all the edges (or border points) of the cell to its center are pre-computed. In addition, the distances across the border points of the *adjacent* cells are also pre-computed. This will reduce the pre-computation time and space by localizing the computation to cells and handful of neighbor-cell node-pairs.

Now, to find the *k* nearest-neighbors of a query object *q*, first the first nearest neighbor is found by simply locating the Voronoi cell that contains *q*. This can be easily achieved by utilizing a spatial index (e.g., R-tree) that is generated for the Voronoi cells. In [8], it is shown that the next nearest neighbors of *q* are within the adjacent cells of the previously explored ones, which can be efficiently retrieved from a lookup table. Next, the intra-cell pre-computed distances are utilized to find the distance from q to the borders of the Voronoi cell of each candidate, and finally the inter-cell pre-computed distances are used to compute the actual network distance from *q* to each candidate. The local pre-computation nature of VN3 also results in low complexity of updates when the network is modified.

Key Applications

The applications of distance-based queries on road-networks are numerous. In emergency response, the first responders may want to find *k* closest hospitals to a crisis area (*k*NN query). In urban planning, one may need to find the set of parks that are closest to a set of houses (known as *spatial skyline queries*). In location-based services, a group of mobile users want to find a meeting location where traveling towards which minimizes their total travel distance (Aggregate-NN query). In Location-based Services (LBS), a driver wants to find all the gas stations within its 4-mile distance (spatial range queries). In OnLine map services such as Yahoo! Maps, Google Earth or Microsoft Virtual Earth, one may want to minimize distance when planning a day trip to a shopping center, a restaurant and a movie theater (OSR query). In all these applications and queries, the accurate and fast computation of network distance given the underlying road network is critical.

Future Directions

Even though, as reviewed in this entry, several techniques for fast computation of network distances have been proposed, there are still no real-world deployment of these or any other techniques to enable accurate and fast network distance computation for spatial queries. This may be attributed to the fact that many users can tolerate or are used to the inaccuracy in distance computations. However, as the collected geospatial data becomes more accurate and the users become more sophisticated, the competition between different geospatial services would lead into the adaptation of a simple but effective technique to provide accurate network distance for query processing. Therefore, a technique that can easily be integrated into the current information infrastructure used by geospatial applications is of substantial importance.

On the research front, new queries and applications for multidimensional spaces are often proposed in academic conferences and journals. Most of these queries are applicable to geospatial applications and road-networks. Therefore, a rather effortless way of finding a new research topic is to take any of these new queries and extend it to work in the road-network space.

Finally, adding other attributes that would affect the distance or time-to-travel in road-networks renders most if not all of the current solutions insufficient. For example, one can consider efficient on-the-fly computation of network distance in the presence of traffic flow, road direction, road closure, road elevation, or navigational data such as costs of left-turns, right turns, U-turns, and stops.

Cross-references

- ▶ Rtree
- ▶ Spatial Data Analysis
- ▶ Spatial Data Mining
- ▶ Spatial Data Types
- ▶ Spatial Indexing Techniques
- ▶ Spatial Network Databases
- ▶ Spatial Operations and Map Operations
- ▶ Spatio-Temporal Trajectories
- ▶ Voronoi Diagrams

Recommended Reading

1. Almeida V.T.D. and Güting R.H. Indexing the Trajectories of Moving Objects in Networks, GeoInformatica, 9(1):33–60, 2005.
2. Cho H.-J. and Chung C.-W. An efficient and scalable approach to cnn queries in a road network. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 865–876.
3. Güting H., de Almeida T., and Ding Z. Modeling and querying moving objects in networks. VLDB J., 15(2):165–190, 2006.
4. Hu H., Lee D.L., and Lee V.C.S. Distance indexing on road networks. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 894–905.
5. Hu H., Lee D.L., and Xu J. Fast nearest neighbor search on road networks. In Advances in Database Technology, Proc. 10th Int. Conf. on Exdending Database Technology, 2006, pp. 186–203.
6. Huang X., Jensen C.S., and Saltenis S. The islands approach to nearest neighbor querying in spatial networks. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005, pp. 73–90.
7. Jensen C.S., Kolářvr J., Pedersen T.B., and Timko. I. Nearest neighbor queries in road networks. In Proc. 11th ACM Int. Symp. on Advances in Geographic Inf. Syst., 2003, pp. 1–8.
8. Kolahdouzan M.R. and Shahabi C. Voronoi-based k nearest neighbor search for spatial network databases. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 840–851.
9. Ku W.-S., Zimmermann R., Wang H., and Wan C.-N. Adaptive nearest neighbor queries in travel time networks. In Proc. 13th ACM Int. Symp. on Geographic Inf. Syst., 2005, pp. 210–219.
10. Papadias D., Zhang J., Mamoulis N., and Tao Y. Query processing in spatial network databases. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 790–801.
11. Pfoser D. and Jensen C.S. Indexing of network constrained moving objects. In Proc. 11th ACM Int. Symp. on Advances in Geographic Inf. Syst., 2003, pp. 25–32.
12. Sankaranarayanan J., Alborzi H., and Samet H. Efficient query processing on spatial networks. In Proc. 13th ACM Int. Symp. on Geographic Inf. Syst., 2005. pp. 200–209.
13. Shahabi C., Kolahdouzan M.R., and Sharifzadeh M. A road network embedding technique for k-nearest neighbor search in moving object databases. In Proc. 10th ACM Int. Symp. on Advances in Geographic Inf. Syst., 2002, pp. 94–100.

14. Sharifzadeh M. and Shahabi C. Processing optimal sequenced route queries using voronoi diagrams. GeoInformatica, 12 (4):411–433, 2008.
15. Yiu M.L., Mamoulis N., and Papadias D. Aggregate nearest neighbor queries in road networks. IEEE Trans. Knowl. Data Eng., 17(6):820–823, 2005.

Road Vector Data

▶ Road Networks

Robot

▶ Web Crawler Architecture

ROC

▶ Receiver Operating Characteristic

Rocchio's Formula

Ben He
University of Glasgow, Glasgow, UK

Definition

Rocchio's formula is used to determine the query term weights of the terms in the new query when Rocchio's relevance feedback algorithm is applied.

Key Points

In 1971, Rocchio proposed a classical query expansion algorithm based on the Vector Space model [1]. The basic algorithm assumes that the user identifies a set R of relevant documents and a set N of non relevant documents and the improved query is the result of a linear combination of the mean frequencies tf of the terms in the original query and in these two sets (*the centroids of R and N*), that is the weight of each term in the new query is:

$$qtf_m = \alpha.qtf + \beta.\sum_{d\in R} tf - \gamma.\sum_{d\in N} tf$$

Feedback variations assume that positive feedback exhibits a much clear impact on the reformulation of the query. Also positive feedback can be applied to expand query without the explicit feedback from the user (*blind relevance feedback*). Blind relevance feedback can be obtained in four steps:

1. All documents are ranked for the given query using a particular Information Retrieval model, for example the TF-IDF term weighting of the vector space model. This step is called *first-pass retrieval.* The user identifies a set R of relevant documents and a set N of non relevant documents.
2. A weight qtf_{exq} is assigned to each term appearing in the set of the k highest ranked documents. In general, qtf_{exq} is the mean of the weights provided by the Information Retrieval model, for example the TF-IDF weights, computed over the set of the k highest ranked documents.
3. The vector of query terms weight is finally modified by taking a linear combination of the initial query term weights qtf used for the first-pass retrieval and the new weight qtf_{exq}, that is:

$$qtw_m = qtf + \beta \cdot qtf_{exq} \qquad (1)$$

4. To remove noisy terms from the expanded query, automatic query expansion techniques usually selects only the highest informative terms from the set of top-ranked documents. The informativeness of a term is determined by the terms with highest weights assigned in step 2.

Cross-references

▶ Query Expansion Models
▶ Relevance Feedback

Recommended Reading

1. Rocchio J. Relevance Feedback in Information Retrieval. Prentice-Hall, Englewood Cliffs, NJ, USA, 1971, pp. 313–323.

Role Based Access Control

Yue Zhang, James B.D. Joshi
University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

RBAC; Role based security

Definition

Access control is a security service responsible for defining which subjects can perform what type of operations on which objects. A subject is typically an active entity such as a user or a process, and an object is an entity, such as a file, database table or a field, on which the subject can perform some authorized operations. A permission indicates the mode of operation on a particular object.

Role based access control (RBAC) involves controlling access to computer resources and information by (i) defining *users*, *roles*, and *permissions*, and (ii) assigning users and permissions to roles. A user can create a *session* in which he/she can activate a subset of the roles he/she has been assigned to and use the permissions associated with the activated roles. RBAC approach is based on the understanding that a user's access needs are defined by the roles that he/she plays within his/her organization. In general, a role is considered as a group of permissions. RBAC approach also uses *role-role relation*, known as *role-hierarchy*, to provide permission inheritance semantics, and *constraints* on the assignment relations and the activation of roles to capture various access control requirements.

Historical Background

The origin of the concept of roles can be traced back to organizational theory much earlier than the advent of computerized information systems. However, it was primarily in the early 1990s that the security researchers and practitioners became interested in adopting the notion of a role to address the access control issues for information systems. In particular, in 1992, Ferraiolo and Kuhn showed that the existing mandatory and discretionary access control (MAC and DAC) approaches were inadequate in addressing the complex and diverse access control needs of various organizations and proposed the use of a Role based approach. While the MAC approach uses the predefined set of system rules to control accesses to resources, thus not giving users discretionary power to grant the rights they have on objects (e.g., files, databases) to other users/subjects, the DAC approach allowed the users to grant the permissions that they have on objects to others freely. Later, the initial work by Ferrailolo and Kuhn was followed by Nyanchama and Osborn's role graph model in 1995 [9], and later by the seminal paper by Sandhu, Coyne, Feinstein and Youman in 1996 [13], where they defined a family of RBAC models each with different sets of capabilities, known as the RBAC96 model. This model later evolved into the NIST RBAC model. The NIST model was later modified into the ANSI/INCITS (ANSI/ICITS stands for American National Standards. Institute and International Committee for Information Technology Standards) RBAC standard in 2004 [2]. The significant interest in this area can be seen by the establishment of the ACM Workshop on RBAC in 1996 which later evolved into the current ACM Symposium in Access Control Method and Technologies (SACMAT). Several extensions of the RBAC model have been proposed and several newer RBAC issues have been identified over the last one decade. A key contribution is also related to the demonstration by Osborn et al. that the RBAC96 model can also be configured to represent the MAC and DAC policies [10], establishing its usefulness as a uniform model for addressing very diverse set of access control needs.

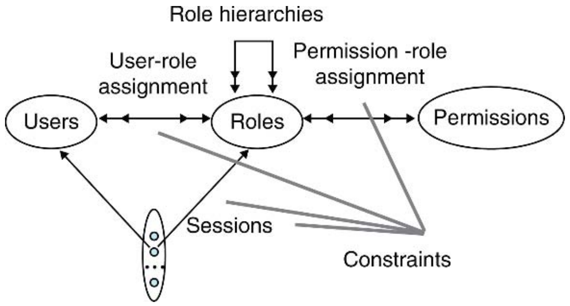
Foundations

Role is a prevalent organizational concept and it identifies the various job functions and responsibilities within an organization. As organizational information systems has become a crucial component for carrying out organizational job functions, it has motivated the use of roles that users within the organization play to define what accesses should be authorized to them so that they can carry out their job functions and responsibilities efficiently. Based on the premise that the role represents the set of permissions that are needed for carrying out the job functions, various RBAC approaches have been proposed. RBAC96 model is the most widely recognized initial model which has evolved into the ANSI/ICITS RBAC standard. The RBAC96 family of models is briefly described next followed by a discussion on other extensions made to it and its standardized version.

Figure 1 depicts the RBAC96 family of models which include $RBAC_0$, $RBAC_1$, $RBAC_2$, and $RBAC_3$. $RBAC_0$ is the base model containing only basic elements; $RBAC_1$ augments $RBAC_0$ with role hierarchy; $RBAC_2$ augments $RBAC_0$ with constraints; and $RBAC_3$ combines $RBAC_1$ and $RBAC_2$, and provides the most complete set of features.

$RBAC_0$ Base Model

In the base model $RBAC_0$, the key elements are the sets users (U), roles(R), permissions (P) and sessions (S).

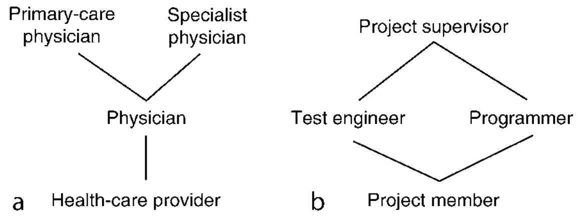


Role Based Access Control. Figure 1. RBAC96 model.

Various relations are defined among them, as depicted in Fig. 1. A *user* in this model is a human being but can also be generalized to include intelligent autonomous agents such as robots, immobile computers, or even networks of computers. A *role* typically represents a job function within the organization with some authority and responsibility. The objects are data objects as well as other computer resources within a computer system. RBAC96 model only supports "positive permissions" that grants accesses to objects, and does not support "negative permissions" that deny accesses. Constraints, as defined in $RBAC_2$, are used as a mechanism to achieve the denial of accesses.

The *user assignment* (*UA*) and *permission assignment* (*PA*) relations shown in Fig. 1 are many-to-many relations. A user can be a member of many roles, and a role can have many users. Similarly, a role can have many permissions and the same permission can be assigned to many roles. The placement of a role as an intermediary to enable a user to exercise a permission provides much greater control over access configuration and review than does directly relating users to permissions.

A *session* is essentially a mapping of one user to possibly many roles, i.e., a user can establish a session and activate within it some subset of roles that he/she has been assigned to. The set of permissions that can be used by a user is the union of the permissions assigned to the roles that the user activates in one session. The association between a user and the activated set of roles within a session remains constant for the life of the session. A user may have multiple sessions open at the same time and each session may have a different combination of assigned roles activated in it. This feature of $RBAC_0$ supports the principle of least privilege that involves use of the minimal set of permissions needed for a particular task. A user who is a member of several roles can invoke any subset of these within a session to complete a task. The concept of a session is equivalent to that of the traditional notion of a *subject* in the access control literature. A subject (or session) is a unit of access control, and a user may have multiple subjects (or sessions) with different permissions [13].



Role Based Access Control. Figure 2. Sample Role Hierarchies.

$RBAC_1$: RBAC with Role Hierarchy

$RBAC_1$ introduces role hierarchies (*RH*) on top of $RBAC_0$ to indicate which roles can inherit which permissions from which other roles. Role hierarchies are a natural way of structuring roles to reflect an organization's lines of authority and responsibility. Figure 2 depicts some example of role hierarchies. By convention senior roles are shown toward the top of these diagrams, and junior roles toward the bottom. For example, in Fig. 2(a), the junior-most role is *health-care provider*. The *physician* role is senior to *health-care provider* and thereby inherits all permissions from *health-care provider*. In addition to the permissions inherited from the *health-care provider* role, the *physician* role can have other permissions assigned directly to it. Inheritance of permissions is transitive. For example, in Fig. 2(a), the *primary-care physician* role inherits permissions from the *physician* and *health-care provider* roles. *Primary-care physician* and *specialist physician* both inherit permissions from the *physician* role. Figure 2(b) illustrates multiple inheritances of permissions, where the *project supervisor* role inherits from both *test engineer* and *programmer* roles. Mathematically, these hierarchies are partial orders. A partial order is a reflexive, transitive and anti-symmetric relation.

$RBAC_2$: RBAC with Constraints

$RBAC_2$ introduces the concept of constraints. Constraints are an important aspect of RBAC and are sometimes argued to be a principal motivation for using RBAC. Constraints are a powerful mechanism for capturing higher-level organizational security policies. Constraints can be applied to the *UA* and *PA* relations, as well as to the association between a user and its activated set of roles within a session.

The most frequently mentioned constraint in the context of RBAC is the *mutually exclusive roles* (MER) constraint that is used to enforce Separation of Duty (SoD) requirements. For instance, a user can be restricted to assume at the most one role in a mutually exclusive role set. This supports separation of duty requirements where a person should not be able to assume two different roles to carry out critical steps of a task. Consider two mutually exclusive roles, *accounts-manager* and *purchasing-manager*. Mutual exclusion in terms of *UA* specifies that one individual cannot be a member of both these roles. Mutual exclusion in terms of *PA* specifies that the same permission cannot be assigned to both these roles. For example, the permission to issue checks should not be assigned to both these roles. Normally such a permission would be assigned to the *accounts-manager* role only. The mutual exclusion constraint on *PA* helps prevent the permission from being unintentionally or intentionally assigned to the *purchasing-manager* role.

Another example of a user assignment constraint is that a role can have a maximum number of members. For instance, the number of roles to which and individual user can belong to could also be limited. These are called *cardinality constraints*. Similarly, the number of roles to which a permission can be assigned can have cardinality constraints to control the distribution of powerful permissions.

RBAC$_3$: The Consolidated Model

RBAC$_3$ combines RBAC$_1$ and RBAC$_2$ to provide both role hierarchies and constraints. There are several issues that arise by bringing these two concepts together.

Constraints can be applied to the role hierarchy itself. For example, constraints on *RH* can limit the number of senior (or junior) roles that a given role may have. Two or more roles can also be constrained to have no common senior (or junior) role. These kinds of constraints are useful in situations where the authority to change the role hierarchy has been decentralized, but the chief security administrator desires to restrict the cases in which such changes can be made.

Benefits of the RBAC Approach

The RBAC approach has been recognized for several beneficial features. First, it supports the principle of least privilege which has been considered very important for better security of systems. When RBAC is used, a user's access requirements can be easily changed when his role is changed within his organization. All that should be done is removing him as a member of his current role and assigning him to the new role. Also, the total number of assignment relationships that needs to be maintained in RBAC is $n_r(n_u + n_p)$ where n_r, n_u, and n_p represent the number of roles, users and permissions, respectively, in a system. In general subject-object based authorization system will need to maintain $n_u.n_p$ subject-to-permission associations. Role hierarchy makes security administration significantly easy as it eliminates the need for explicitly assigning the permissions to multiple roles. Constraints can be used to capture various types of policies including very important SoD requirements. It has been shown that by configuring constraints and relationships in RBAC, one can configure an RBAC system to express the traditional DAC and MAC policies [10]. Because of its capability to capture very diverse sets of requirements, it has been considered as a very promising approach to address emerging multidomain security problems where various domains with different access control policies need to securely interact with each other.

RBAC Standards

The NIST RBAC model was the first attempt towards establishing an RBAC standard. Compared to the RBAC96 models, the most distinct feature in NIST RBAC is the clear specification of incremented 4-level RBAC models. The first level, called the core RBAC is similar to the RBAC$_0$ model with the explicit specification of user-role view where the roles assigned to a specific user and the users assigned to a specific role can be determined. The second level, called the hierarchical RBAC model is based on the RBAC$_1$ model and includes the separation of general hierarchy (the same as in RBAC$_1$) and the restricted hierarchy, where the structure of the hierarchy is restricted to a certain type, such as, a tree or an inverted tree structure. The third level is the same as the constraint RBAC. The fourth level further adds the permission-role view where the roles assigned to a specific permission and the permissions assigned to a specific role can be determined. Later, the NIST model was further refined to establish the ANSI/INCITS RBAC standard.

Administration Models for RBAC

In large organizational systems the number of roles can be in the hundreds or thousands. Managing these roles and their relationships is a daunting task that is often highly centralized. A key issue is how the benefits of the RBAC model can be used to construct an administration model for managing the RBAC policies. Several researchers have addressed this issue, including the Administrative RBAC (ARBAC) family of models by Sandhu et al. [12] and Scoped ARBAC (SARBAC) family of models by Crampton et al. [6].

RBAC Extensions

The RBAC96 model and its standardized versions also have been extended in several ways to address emerging applications. One notable among the emerging requirements is the need to capture context based access control requirements. Several extensions of RBAC models have been made to address such a need. Notable among these are extensions of RBAC to capture temporal and/or location context. Temporal RBAC (TRBAC) model [3] and its generalized version Generalized TRBAC (GTRBAC) [7] are main work related to temporal extensions of the RBAC model. While these capture the time context, several works have tried to address the need of capturing location context within an RBAC framework. Covington et al. propose a Generalized RBAC (GRBAC) model, where they introduce the concept of environment roles, which are roles that can be activated based on the value of conditions in the environment where the request has been made. Bertino et al. [4] has recently proposed the GEO-RBAC model that integrates RBAC with a spatial model based on the OpenGIS system. A significant aspect of the GEO-RBAC model includes (i) its specification of role schema that are location-based, and (ii) the separation of role schema and role instances to provide different authorizations for different logical and physical locations. Location and time-based RBAC (LoT-RBAC) model was proposed by Chandran et al. [5] to address the access control requirements of highly mobile, dynamic environments to provide both location and time based control. Significant work has also been done to try to develop specification languages for RBAC models. This includes the inclusion of RBAC profile in OASIS's XACML. Joshi et al. propose X-RBAC that generalizes X-GTRBAC, which is the XML-based language for specifying the GTRBAC policies. Other extensions to the RBAC work include developing better constraint frameworks and analyzing their complexity issues.

Support for SoD constraints is a major issue for the RBAC model. Various SoD constraints proposed in the literature include: Static SoD, Dynamic SoD, History Based SoD, Object Based SoD, Operational SoD, Order Dependent/Independent SoD. The ANSI RBAC standard, however, only supports the basic SoD constraints. Several researchers have developed RBAC extensions to support these advanced SoD constraints. Ahn et al. have proposed the RCL2000 language for specifying several role-based SoD constraints [1]. Joshi et al. have proposed time-based SoD constraints in [8].

RBAC approach has also been shown to be very beneficial for multi-domain security which refers to the need to facilitate secure interactions among multiple security domains with very diverse set of access control requirements. Several researchers have recently focused on using RBAC to address the multi-domain security challenge [11,14]. One approach uses mapping roles from multiple domains to ensure secure cross--domain accesses. Such inter-domain role mapping problem has been recently formalized and solutions for both tightly coupled environments as well as lightly coupled environments are being sought [11,14]. Some researchers have employed RBAC delegation models to address such multi-domain sharing issues [15]. Several RBAC delegation models have been proposed that use roles as the central entity in the process of delegating rights from one user to another.

Key Applications

RBAC has been used in operating systems, databases and applications. In particular, RBAC is very promising for large scale application environments and enterprises. Emerging systems and applications have more context and content based access control requirements as can be seen in mobile and peer to peer applications and several extensions of RBAC has been motivated by such requirements.

Future Directions

Future applications will require efficient access control techniques to protect large number of objects and manage huge number of privileges that may need to be given to unknown users. Furthermore, interactions among multiple domains with different access control policy requirements are crucial in emerging applications. RBAC has been found to be promising for these emerging access control requirements. Several researchers have advocated use of RBAC to address large scale enterprise security as well as multidomain security. Work related to multidomain security using RBAC approach is in its initial stages of development. Policy verification and evolution management issues are crucial for access control in large and dynamic systems but little work currently have addressed these.

URL to Code

NIST provides an implementation of some RBAC versions http://csrc.nist.gov/rbac/.
ANSI/INCITS website: http://csrc.nist.gov/groups/SNS/rbac/standards.html.

Cross-references

► Access Control Policy Languages
► Discretionary Access Control
► Mandatory Access Control
► Security Policies
► Temporal Access Control

Recommended Reading

1. Ahn G. and Sandhu R. Role-based authorization constraints specification. ACM Trans. Inf. Syst. Secur., 3(4):207–226, 2000.
2. American national standard for information technology (ANSI). Role based access control. ANSI INCITS 359–2004, February 2004.
3. Bertino E., Bonatti P.A., and Ferrari E. TRBAC: a temporal role-based access control model. ACM Trans. Inf. Syst. Secur., 4(3):191–233, 2001.
4. Bertino E., Catania B., Damiani M.L., and Perlasca P. GEO-RBAC: A spatially aware RBAC. In Proc. 10th ACM Symp. on Access Control Models and Technologies, 2005, pp. 29–37.
5. Chandran S.M. and Joshi J.B.D. LoT RBAC: a location and time-based RBAC model. In Proc. 6th Int. Conf. on Web Information Systems Eng., 2005, pp. 361–375.
6. Crampton J. and Loizou G. Administrative scope: a foundation for role-based administrative models. ACM Trans. Inf. Syst. Secur., 6(2):201–231, 2003.
7. Joshi J.B.D., Bertino E., Latif U., and Ghafoor A. A generalized temporal role-based access control model. IEEE Trans. Knowl. Data Eng., 17(1):4–23, 2005.
8. Joshi J.B.D., Shafiq B., Ghafoor A., and Bertino E. Dependencies and separation of duty constraints in GTRBAC. In Proc. 8th ACM Symp. on Access Control Models and Technologies, 2003, pp. 51–64.
9. Nyanchama M. and Osborn S.L. The role graph model. In Proc. 1st ACM Workshop on Role-Based Access Control, 1995.
10. Osborn S., Sandhu R., and Munawer Q. Configuring role-based access control to enforce mandatory and discretionary access control policies. ACM Trans. Inf. Syst. Secur., 3:85–106, 2000.
11. Piromruen S. and Joshi J.B.D. An RBAC framework for time constrained secure interoperation in multi-domain environment. In Proc. IEEE Workshop on Object-oriented Real-time Dependable Systems, 2005, pp. 36–45.
12. Sandhu R., Bhamidipati V., and Munawer Q. The ARBAC97 model for role-based administration of roles. ACM Trans. Inf. Syst. Secur., 2(1):105–135, 1999.
13. Sandhu R.S., Coyne E.J., Feinstein H.L., and Youman C.E. Role-based access control models. IEEE Comput., 29(2):38–47, 1996.
14. Shafiq B., Joshi J.B.D., Bertino E., and Ghafoor A. Secure interoperation in a multi-domain environment employing RBAC policies. IEEE Trans. Knowl. Data Eng., 17(11):1557–1577, 2005.
15. Zhang L., Ahn G., and Chu B. A role-based delegation framework for healthcare information systems. In Proc. 7th ACM Symp. on Access Control Models and Technologies, 2002, pp. 125–134.

Role Based Security

► Role Based Access Control

Rollback

► Crash Recovery
► Logging and Recovery

Rollback Operator

► Timeslice Operator

Rotation

► Dynamic Graphics

Rotation Estimation

► Cross-Validation

Rough Computing

► Decision Rule Mining in Rough Set Theory

Rough Set Theory (RST)

► Decision Rule Mining in Rough Set Theory

Rough Set Theory, Granular Computing on Partition

► Deductive Data Mining Using Granular Computing

Rounding

► Microdata Rounding

Row-Level Locking

► B-Tree Locking

Row-Versioning

► Snapshot Isolation

R-Precision

Nick Craswell
Microsoft Research Cambridge, Cambridge, UK

Definition

For a given query topic *Q*, *R*-precision is the precision at *R*, where *R* is the number of relevant documents for *Q*. In other words, if there are *r* relevant documents among the top-*R* retrieved documents, then *R*-precision is $\frac{r}{R}$.

Key Points

R-precision is defined as the proportion of the top-*R* retrieved documents that are relevant, where *R* is the number of relevant documents for the current query. This requires full knowledge of a query's relevant set, and will be a shallow evaluation for a query with few relevant documents and a deep evaluation for a query with many relevant documents. Rank cutoff *R* is the point at which precision and recall are equal, since at that point both are $\frac{r}{R}$.

Cross-references

► Average R-Precision
► Precision
► Precision at n
► Precision-Oriented Effectiveness Measures

RSJ Model

► Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model

Rtree

► Tree-based Indexing

R-Tree (and Family)

Apostolos N. Papadopoulos[1], Antonio Corral[2],
Alexandros Nanopoulos[1], Yannis Theodoridis[3]
[1]Aristotle University, Thessaloniki, Greece
[2]University of Almeria, Almeria, Spain
[3]University of Piraeus, Piraeus, Greece

Definition

The R-tree is an indexing scheme that has been originally proposed towards organizing spatial objects such as points, rectangles and polygons. It is a hierarchical data structure suitable to index objects in secondary storage (disk) as well as in main memory. The R-tree has been extensively used by researchers to offer efficient processing of queries in multi-dimensional data sets. Queries such as *range*, *nearest-neighbor* and *spatial joins* are supported efficiently leading to considerable decrease in computational and I/O time in comparison to previous approaches. The R-tree is capable of handling diverse types of objects, by using approximations. This means that an object is approximated by its minimum bounding rectangle (MBR) towards providing an efficient filtering step. Objects that survive the filtering step are inspected further for relevance in the refinement step. The advantages of the structure, its simplicity as well as its resemblance to the B$^+$-tree "persuaded" the database industry to implement it in commercially available systems in addition to research prototypes.

Historical Background

The R-tree index was proposed by Guttman [6] in 1984 in order to solve an organization problem regarding rectangular objects in VLSI design. Later on, the structure was revised to become even more efficient and to adapt to the particular problem. The set of variants comprise the so called *R-tree family* of access methods.

The first variant, the R$^+$-tree, was proposed in 1987 [15]. The main difference between the two schemes is that while in the R-tree the MBR of an object is placed to one leaf node only, in the R$^+$-tree the MBR may break to several smaller MBRs and stored in different leaf nodes. The motivation behind this new index is that large MBRs may cause performance degradation. However, storage utilization decreases because of the MBR decomposition.

The next most important variation, the R*-tree [1], retains the properties of the original R-tree and provides better strategies for inserting MBRs. New heuristics are offered towards improving the shape of the tree during insertions and node splits. Performance evaluation results have shown that the R*-tree offers significantly better performance in *range query* processing in comparison to R-trees and R$^+$-trees. The tradeoff is that more time is spent during insertions and deletions of objects, since the heuristics used require more computational overhead.

Another important variation of the R-tree is the Hilbert R-tree [8]. It is a hybrid structure based on the R-tree and the B$^+$-tree. Actually, it is a B$^+$-tree with geometrical objects being characterized by the Hilbert value of their centroid. The structure is based on the Hilbert space-filling curve. According to the authors' experimentation in [8], Hilbert R-trees were proven to be the best dynamic version of R-trees as of the time of publication. The term *dynamic* is used to denote that insertions and deletions are allowed in the underlying data set.

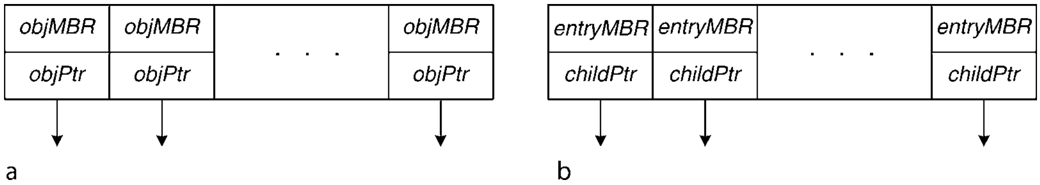
The aforementioned contributions focused on the use of heuristics during tree construction with the aim to provide a well-formed structure towards better query processing. However, by inserting objects one-by-one two problems may arise: (i) the shape of the structure may not be compact and (ii) the storage utilization will never reach 100% due to the rules applied in node splits. In cases where objects are known in advance, a bottom-up building process (also called bulk-loading) may be applied in contrast to the usual top-down tree construction. The first contribution in this area is due to Roussopoulos and Leifker in [13] who proposed the use of packed R-trees. Along the same lines, Kamel and Faloutsos [7] proposed a packed version of the R-tree where packing is performed by the use of the Hilbert space filling curve. Other packed variants have been proposed in [3,4,9].

The R-tree inspired subsequent work on handling high-dimensional data. It has been observed by many researchers that the performance of R-trees degrades rapidly when the number of dimensions increases above a threshold (10 or 15). To tackle this dimensionality curse problem a number of R-tree variations appeared that show better scaling capabilities for high-dimensional spaces. Some of these contributions are the TV-tree [10], the X-tree [2] and the A-tree [14].

Due to space limitations, the description of more variations (actually these are more than 70) is not feasible. A more detailed presentation of the R-tree family can be found in [11].

Foundations

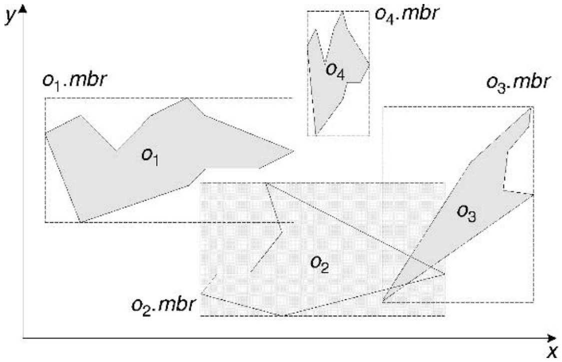
Let $\mathcal{O}$ be a set of objects in the two-dimensional space. Each object $o_i \in \mathcal{O}$ is represented by its MBR, which is the minimum rectangle that completely encloses the object. The MBR of object o_i is denoted by $o_i.mbr$. Since MBRs are forced to be orthogonal with respect to the axes, each MBR is completely defined by its lower-left and upper-right corners. The R-tree organizes data in a hierarchical manner. It is a height-balanced tree where object MBRs are hosted in leaf nodes, whereas internal nodes contain auxiliary entries to guide the search process. More formally, each leaf node contains pairs of the form (*objMBR*, *objPtr*) where *objMBR* is the MBR of an object, and *objPtr* is the pointer to the object's detailed information (geometry and/or other attributes). Each internal node contains entries of the form (*entryMBR*,*childPtr*) where *entryMBR* is the MBR enclosing all descendants in the corresponding subtree and *childPtr* is the pointer to the subtree. The format of leaf and internal nodes is illustrated in Fig. 1.



R-Tree (and Family). Figure 1. Format of leaf and internal nodes. (a) Format of a leaf node. (b) Format of an internal node.

Each R-tree node corresponds to one disk page which has a limited capacity. The number of entries in each internal node determines the tree *fanout*, which is the maximum number of subtrees that can emanate from an internal node. Let M_{int} and M_{leaf} denote the maximum number of entries that can be hosted in an internal node and a leaf node, respectively. To guarantee acceptable storage utilization, a minimum number of entries per node must also be defined. Therefore, the tree construction procedure forces that each internal node (leaf) must contain no less than m_{int} (m_{leaf}) entries. In the case where the R-tree stores MBR of objects, then evidently $M_{int} = M_{leaf}$ and $m_{int} = m_{leaf}$. However, there are cases where the above equalities do not hold. For example, when the R-tree stores point objects, then the number of objects in a leaf node increases, since a point requires less data for representation than a rectangle. For ease of illustration, for the rest of the article it is assumed that the R-tree stores rectangles, which means that the minimum and the maximum number of entries is the same for all tree nodes (the symbols m and M are used, respectively). Therefore, each tree node contains at least $m \geq M/2$ entries and at most M entries. A violation of this rule is only allowed for the root of the tree, which may contain less than m entries.

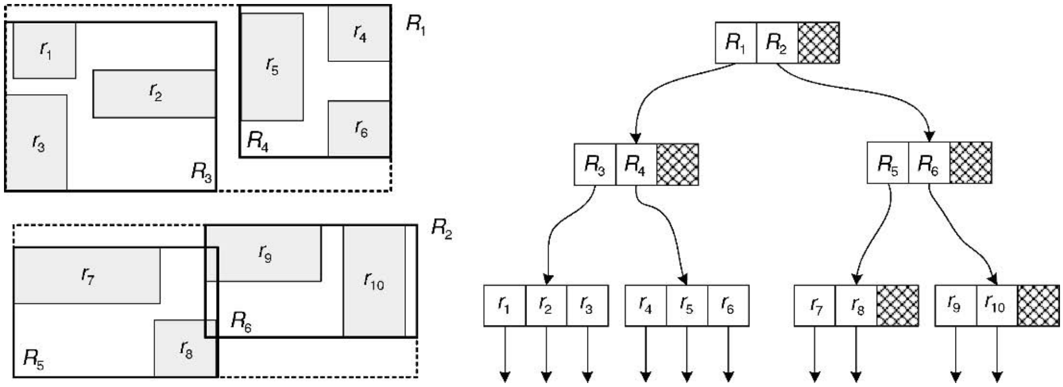
Since MBR is an approximation of the original object, intersection tests must be performed in two steps to guarantee correctness of results. Figure 2 depicts three polygonal objects with their corresponding MBRs. Searching for objects intersected by o_2, it is evident that both o_1 and o_3 may be contained in the final answer, because $o_2.mbr$ intersects both $o_1.mbr$ and $o_3.mbr$. This information is easily obtained by performing efficient intersection tests for rectangular objects. However, to produce the final result the answer requires further refinement by considering the detailed geometric characteristics of objects o_1, o_2 and o_3. This refinement step involves more complex geometric computations and it is applied only for candidate objects. Evidently, if an MBR does not intersect $o_2.mbr$ (e.g., $o_4.mbr$) there is no need to investigate it further.



R-Tree (and Family). Figure 2. Intersection of objects and MBRs.

Figure 3 shows a set of objects in the 2-d space (left) and the corresponding R-tree index (right). For simplicity only the MBRs of objects are shown using the symbols r_1 through r_{12}. The R-tree is composed of seven nodes. There are four leaf nodes containing the object MBRs, and three internal nodes containing MBRs that enclose all the descendants in the corresponding subtree. For example, the MBR R_1 which is hosted at the root, encloses MBRs R_3 and R_4. Moreover, R_3 encloses the object MBRs r_1, r_2 and r_3. In this example, it is assumed that each node can host up to three entries. However, in real implementations this number is significantly higher and depends on the page size and the number of dimensions.

It is evident that the R-tree for a collection of object is not unique. The shape of the tree depends significantly on the insertion order. As it has been pointed out previously, there are two ways to build an R-tree: (i) by individual insertion of objects and (ii) by bulk loading. In dynamic data sets, where objects may be



R-Tree (and Family). Figure 3. R-tree example.

inserted and deleted in an ad hoc manner, the first method is applied. In the sequel, the insertion process is discussed briefly.

Insertions in an R-tree are handled similarly to insertions in a B^+-tree. In particular, the R-tree is traversed to locate an appropriate leaf node L to accommodate the new entry. The selection of L involves a number of internal node selections towards determining an appropriate path from the root to the most convenient leaf. It is important to guarantee that after the insertion, the tree will be in a good shape. Towards this goal, the insertion process tries to select the next node in the insertion path, aiming at low MBR enlargement, because the size of MBRs is directly connected to search efficiency. The new entry is inserted in L and then all nodes within the path from the root to L are updated accordingly (adjustment of MBRs). In case the found leaf cannot accommodate the new entry because it is full (it already contains M entries), then it is split into two nodes. Splitting in R-trees is different from that of the B^+-tree, because it considers different criteria. Guttman in his original paper [6] proposed three different split policies:

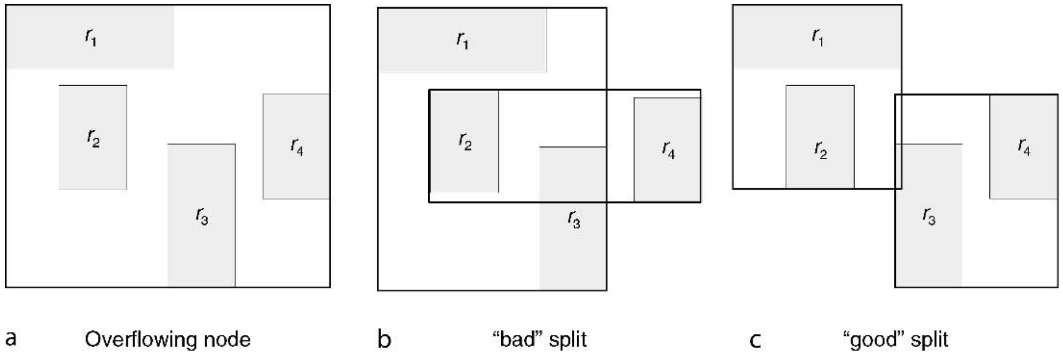
Linear Split. Choose two objects as seeds for the two nodes, where these objects are as far apart as possible. Then consider each remaining object in a random order and assign it to the node requiring the smallest enlargement of its respective MBR.

Quadratic Split. Choose two objects as seeds for the two nodes, where these objects if put together create as much dead space as possible (*dead space* is the space that remains from the MBR if the areas of the two objects are ignored). Then, until there are no remaining objects, insert the object for which the difference of dead space if assigned to each of the two nodes is maximized in the node that requires less enlargement of its respective MBR.

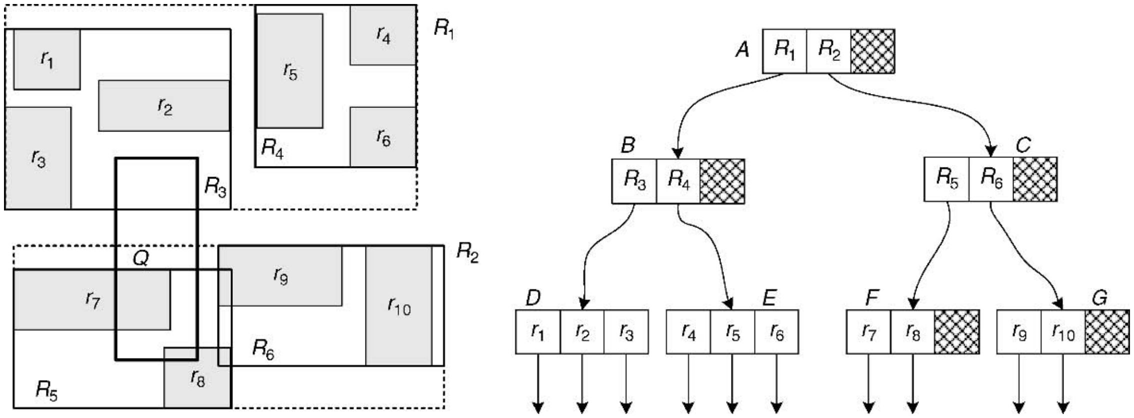
Exponential Split. All possible groupings are exhaustively tested and the best is chosen with respect to the minimization of the MBR enlargement.

Guttman suggested using the quadratic algorithm as a good compromise between insertion speed and retrieval performance. The linear split policy is the most efficient but the resulting R-tree does not keep its nice characteristics, whereas the exponential split policy although it takes the best split decision it requires significant computational overhead. Albeit the split policy being used, splits may propagate upwards up to the root node. If there is a split in the root, a new root node is created and the height of the tree increases. To demonstrate the importance of the splitting policy an example is given in Fig. 4. Figure 4a depicts an overflowing node, assuming that at most three entries can be stored. A "bad" split is shown in Fig. 4b whereas Fig. 4c depicts a "good" split choice. The first split should be avoided since the quality of the resulting nodes degrades (large MBRs with a lot of overlap), whereas the second is more preferable (small MBRs with small overlap).

Deletions are performed by first searching the tree to locate the corresponding leaf L which contains the deleted object. After the removal of the entry from L the node may contain fewer than m entries (node underflow). The handling of an underflowing node is different in the R-tree, compared with the case of B^+-tree. In the latter, an underflowing case is handled by merging two sibling nodes. Since B^+-trees index one-dimensional data, two sibling nodes will contain consecutive entries. However, for multi-dimensional data, this property does not hold. Although one still



R-Tree (and Family). **Figure 4.** Splitting example.



R-Tree (and Family). **Figure 5.** Range query example using an R-tree.

may consider the merging of two R-tree nodes that are stored at the same level, reinsertion is more appealing for the following reasons:

1. Reinsertion achieves the same result as merging. Additionally, the algorithm for insertion is used. Moreover, the pages required during reinsertion are likely to be available in the buffer memory, because they have been retrieved during the search of the deleted entry.
2. As described, the insertion process tries to maintain the good quality of the tree during the query operations. Therefore, it sounds reasonable to use reinsertion, because the quality of the tree may degrade after several deletions.

In all R-tree variants that have appeared in the literature, tree traversals for any kind of operations are executed in a way similar to the one applied in the original R-tree. An exception is the R^+-tree which decomposes an object to smaller parts and therefore, the search process requires some modifications in comparison to other variants. Basically, the dynamic variations of R-trees differ in how they perform splits and how they handle insertions in general.

To demonstrate the way queries are executed, a simple range query example is given. Figure 5 shows the region of interest *Q* and the query asks for all objects that intersect *Q*. The nodes of the R-tree are labeled using the symbols *A* through *G*. The first accessed node is *A* (the root). The entries of *A* are tested for intersection with *Q*. Evidently, *Q* intersects both R_1 and R_2, therefore both subtrees require further investigation. The next accessed node is *B* which contains the MBRs R_3 and R_4. Among them, only R_3 intersects the region of interest, which means that node *D* should be accessed next. Node *D* contains the object MBRs r_1, r_2 and r_3 and none of them intersects *Q*. At this point, the search process backtracks to node *B* and since no eligible entries are found it backtracks to node *A*. Next, node *C* is accessed which contains the

MBRs R_5 and R_6. Only R_5 intersects the region of interest, and therefore node F is accessed next. By inspecting the entries of F it is evident that both object MBRs r_7 and r_8 intersect Q, and both are included in the final answer. The search process backtracks to node C and since no more promising branches can be followed, it backtracks to node A. At this point, all promising branches have been examined and the range query execution terminates. The object MBRs that intersect Q are r_7 and r_8. Note that if r_7 and r_8 correspond to the real objects then no further actions should be taken. Otherwise, the detailed geometry of these objects must be tested for intersection with Q (refinement).

In the previous lines, the main issues related to R-tree construction and search have been briefly discussed. The interested reader is directed to [11] for an exhaustive list of algorithmic techniques regarding R-trees and related structures.

Key Applications

Geographic Information Systems

R-tree is an excellent choice for indexing spatial data sets. Algorithms have been proposed to answer fundamental query types like *range queries*, *nearest-neighbor queries* and more complex queries like *spatial joins* and *closest pairs* using R-trees to index the underlying data. Therefore, the structure is equipped with all necessary tools to organize and index geographic information.

Location-Based Services

Many location-aware algorithmic techniques are based on the R-tree index or its enhancements. Queries involving the current location of moving objects are handled efficiently by R-tree variants and therefore these schemes are a convenient tool for organizing objects that potentially change their location.

Multimedia Database Systems

Since the R-tree index is capable of organizing multidimensional data, it can be utilized as an indexing scheme for multimedia data (e.g., images, audio). A common approach to organize multimedia data is to represent the complex multimedia information by using feature vectors. These feature vectors can be organized by means of R-trees, to facilitate similarity search towards multimedia retrieval by content [5].

Future Directions

R-trees have been successfully applied to offer efficient indexing support in diverse disciplines such as query processing in spatial and spatio-temporal databases, multi-attribute data indexing, preference query processing to name a few. An important research direction towards more efficient query processing in modern systems is to provide efficient indexing schemes towards distributed processing. A significant effort towards this direction has been reported in [12], which proposes a distributed R-tree indexing scheme. Taking into account that huge volumes of multi-attribute data are scattered across different systems, such distributed schemes are expected to offer enormous help towards efficient query processing. The challenge is to provide efficient implementations of the corresponding centralized algorithms developed so far, since the methods used for centralized structures are likely to fail when applied to distributed data.

Experimental Results

The interested reader will find a plethora of performance evaluation results in the corresponding literature. Usually, when a new indexing scheme is proposed it is experimentally compared to other methods. Although these comparisons are not based on a common framework, they reveal the advantages and disadvantages of the indexing schemes under study. Usually, the comparison is based on the number of disk accesses and the computational time required to process queries.

Data Sets

Performance evaluation results regarding R-trees and related indexing schemes are produced based on real-life as well as on synthetically generated data sets with diverse distributions. The interested reader can browse the data sets hosted in http://www.rtreeportal.org to view some representative real-life data sets that are consistently being used by researchers for comparison purposes. Synthetically generated data sets follow different distributions (e.g., uniform, normal, Zipf) and their use provide additional hints for the index performance. Moreover, the use of synthetically generated data sets offers the flexibility to choose the cardinality of the data set, the distribution and the size of the objects (e.g., extent of MBRs). Hence, the comparison among indexing schemes is more reliable.

URL to Code

R-tree portal (http://www.rtreeportal.org) contains the code for most common spatial access methods (mainly R-tree and variations), as well as data generators and several useful links for researchers and practitioners interested in spatial database issues.

Cross-references

- ▶ Closest Pair Query
- ▶ Nearest-Neighbor Query
- ▶ Range Query
- ▶ Spatial Join

Recommended Reading

1. Beckmann N., Kriegel H.P., Seeger B. The R.*-tree: an efficient and robust method for points and rectangles. In ACM SIGMOD Conf. on Management of Data, 1990, pp. 322–331.
2. Berchtold S., Keim D.A., and Kriegel H.P. The X-tree: an index structure for high-dimensional data. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 28–39.
3. Chen L., Choubey R., and Rundensteiner E.A. Bulk-Insertions into R-trees using the small-tree-large-tree approach. In Proc. 6th Int. Symp. on Advances in Geographic Inf. Syst., 1998, pp. 161–162.
4. Choubey R., Chen L., and Rundensteiner E.A. GBI – A generalized R-tree bulk-insertion strategy. In Proc. 6th Int. Symp. Advances in Spatial Databases, 1999, pp. 91–108.
5. Faloutsos C. Searching Multimedia Databases by Content. Kluwer, Dordecht, 1996.
6. Guttman A. R-trees: a dynamic index structure for spatial searching. In ACM SIGMOD Conf. on Management of Data, 1984, pp. 47–57.
7. Kamel I. and Faloutsos C. On Packing R-trees. In ACM Int. Conf. on Information and Knowledge Management, 1993, pp. 490–499.
8. Kamel I. and Faloutsos C. Hilbert R-tree – an Improved R-tree using fractals. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 500–509.
9. Leutenegger S., Edgington J.M., and Lopez M.A. STR – A Simple and Efficient Algorithm for R-tree Packing. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 497–506.
10. Lin K., Jagadish H.V., and Faloutsos C. The TV-Tree: An index structure for high-dimensional data. VLDB J. 3, 1994, 517–542.
11. Manolopoulos Y., Nanopoulos A., Papadopoulos A.N., and Theodoridis Y. R-trees: Theory and Applications. Springer, Berlin Heidelberg New York, 2006.
12. du Mouza C., Litwin W., and Rigaux P. SD-Rtree: A Scalable Distributed R-tree. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 296–305.
13. Roussopoulos N. and Leifker D. Direct spatial search on pictorial databases using packed R-trees. ACM SIGMOD Rec. 14 (4):17–31, 1985.
14. Sakurai Y., Yoshikawa M., Uemura S., and Kojima H. Spatial indexing of high-dimensional data based on relative approximation. VLDB J. 11(2):93–108, 2002.
15. Sellis T., Roussopoulos N., Faloutsos C. The R+-tree: a dynamic index for multidimensional objects. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 507–518.

Rule Bases

▶ Closed Itemset Mining and Nonredundant Association Rule Mining

Rule-based Classification

Anthony K. H. Tung
National University of Singapore, Singapore, Singapore

Definition

The term rule-based classification can be used to refer to any classification scheme that make use of IF-THEN rules for class prediction. Rule-based classification schemes typically consist of the following components:

- *Rule Induction Algorithm* This refers to the process of extracting relevant IF-THEN rules from the data which can be done directly using sequential covering algorithms [1,2,5–7,9,12,14–16] or indirectly from other data mining methods like decision tree building [11,13] or association rule mining [3,4,8,10].
- *Rule Ranking Measures* This refers to some values that are used to measure the usefulness of a rule in providing accurate prediction. Rule ranking measures are often used in the rule induction algorithm to prune off unnecessary rules and improve efficiency. They are also used in the class prediction algorithm to give a ranking to the rules which will be then be utilized to predict the class of new cases.
- *Class Prediction Algorithm* Given a new record with unknown class, the class prediction algorithm will predict the class of the new record based on the IF-THEN rules that are output by the rule induction algorithm. In many cases where multiple rules could be matched by the new case, the rule ranking measures will be used to either select the best best matching rule based on the ranking or to compute an aggregate from the multiple matching rules in order to arrive at a final prediction.

Historical Background

Earlier rule-based classification methods includes AQ [7], CN2 [1] and more recently RIPPER [2]. These methods induce rules using the sequential covering algorithm where. Rules are learned one at a time. Decision tree classification methods like C4.5 [13] can also be considered as a form of rule-based classification. However, decision tree induction involved parallel rule induction, where rules are induced at the same time. Even more recently, advances in association rule mining had made it possible to mine association rules efficiently in order to build a classifier [3,4,8,10]]. Such an approach can also be considered as rule-based classification.

Foundations

The discussion first looks at how IF-THEN rules can be used for classification before proceeding to look at their ranking measures and induction algorithms.

1. Using IF-THEN Rules for Classification

An IF-THEN rule is typically an expression of the form *LHS* ⇒ *RHS* where *LHS* is a set of conditions that much be meet in order to derive a conclusion represented by *RHS*. In much literature, *LHS* is called the *antecedent* of the rule and *RHS* is called the *consequent* \of the rule. For rule-based classification, the rule antecedent typically consists of a conjunction(AND)of attribute tests while the rule consequent is a class value. An example of a IF-THEN rule will be:

$$\begin{aligned}&(\text{No. of years} \geq 6) \text{ and } (\text{Rank}\\ &\quad = \textit{Associate Professor}) \Rightarrow (\text{Tenured} = \text{YES})\end{aligned}$$

A rule is say to *cover* a record if the record matches all the antecedent conditions in the rule. Given a new record with unknown class value, rules which cover the record will be used to determine the class value of the record. In the case where only one rule matches the tuple, the class value at the consequent of such a rule will be assigned as the predicted class for the tuple. On the other hand, it is also possible for none of the rules to match, in which case a *default class value* will be assigned.

For more complex situation in which multiple rules are matched, there are usually two approaches:

(i) Top Rule Approach In this approach, all the rules that matched the new record are ranked based on the rule ranking measures. The consequent of the rule that is rank top based on this approach will be the predicted class value of the record.

(ii) Aggregation Approach In the aggregation approach, rules that match the new record are separated into groups based on their consequent. For each of the rule group with the same consequent, an aggregated measure will be computed based on the rule ranking measure for each rule in the group. Each group of rules are then ranked based on their aggregated measure and the consequent of the rule group that are ranked highest will be the predicted class value for the new record.

Note that the two approaches described above are not mutually exclusive. For example it is possible to pick the top-k rules based on the first approach and then apply the aggregation approach on the top-k rules so as to determine the final predicted class value.

2. Rule Ranking Measures

Rule ranking measures are important components in a rule-based classification scheme because of two reasons. First, they can improve the efficiency of constructing and using the classifier. Rules that are not deemed to be useful based on these measures can be pruned off during rule induction making the process more efficient and also reducing the number of rules that must be processed during classification. Second, they can enhance the effectiveness of a rule-based classifier by removing rules which have weak prediction power.

Among the various rule ranking measure, the most basic ones are *coverage*, *accuracy* and *length of the rule*. Let n be the number of record in the training database, n_r be the number of records that match both the antecedent and consequent of a rule r and n_{lr} be the number of records that are covered by r. The coverage of a rule r, denotes as $cov(r)$ is defined to be n_{lr}/n while the accuracy of rule r is defined to be n_r/n_{lr} and denote as $acc(r)$. A rule should have high coverage and accuracy in order to be useful for classification. Besides coverage and accuracy, the length of a rule, $len(r)$, which is the number of terms at the antecedent of r, is also important in determining the usefulness of a rule in a rule-based classification scheme.

More complex rule ranking measures try to integrate both coverage and accuracy. They include *information gain* and *likelihood ratio*. Information gain was proposed in FOIL (First Order Inductive Learner) for comparing rules whose antecedents are subset/superset of each other. Let r' be a rule which

Algorithm 1 Generic Sequential Covering Algorithm

```
Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.
Input: D, a database of class-labelled records; Attvals, the set of all attributes and
their possible values.
Output: A set of IF-THEN rules. Method:
    Rule set = { };
    for each class c do
        repeat
            Rule = Learn OneRule (D, Att vals, c);
            remove tuples covered by Rule from D;
        until terminating condition;
        Rule set = Rule set + Rule; // add new rule to rule set
    return Rule Set;
```

antecedent is a superset of *r*. FOIL assess the information gain of r' over r to be

$$FOIL\_Gain = p' \times \left(\log_2 \frac{p'}{p' + n'} - \log_2 \frac{p}{p + n}\right) \quad (1)$$

where p', n', p' and n are the number of records that are of positive class and negative class and which are covered by r' and r respectively. FOIL_gain favors rules that have high accuracy and cover many positive tuples.

Besides, information gain, one can also use a statistical test of significance to determine if the apparent effect of a rule is not attributed to chance but instead indicates a genuine correlation between attribute values and classes. The test compares the observed distribution among classes of tuples covered by a rule with the expected distribution that would result if the rule made predictions at random. Given *m* classes, let p_i be the probability distribution of class i within the database and $p_i(r)$ be the probability distribution of class i within the set of records that are covered by *r*. The likelihood ratio is computed as:

$$Likelihood\_Ratio = 2\sum_{1}^{m} p_i(r) \log(\frac{p_i(r)}{p_i}) \quad (2)$$

The likelihood ratio is then used to perform a significant test against a χ^2 distribution with m-1 degrees of freedom. The higher the likelihood ratio is, the more likely that there is a significant difference in the number of correct predictions made by the rule in comparison with a random guess that following the class probability distribution of the database.

3. Rule Induction

Various forms of rule induction can be performed for rule-based classification. Here, the *sequential covering* algorithm will be described.

Algorithm 1 presents a generic algorithm for sequential covering rule induction. The algorithm iteratively learn one rule from the database and then remove all records that are covered by the rule before learning the next rule. This is done repeatedly until a terminating condition is met. The terminating condition can varies across different algorithms but is typically linked to the fact no more interesting rules can be induced once many of the records are removed from the database. All rules that are induced during the process are then output.

Each single invocation of LearnOneRule typically involve a greedy search for interesting rule based on the rule ranking measures. This is done by searching for attribute conditions which will improve the rule ranking measure when they are appended to the antecedent of the rule. A single attribute condition that best improve the measure is appended each time and this is done repeatedly until the measure cannot be improved. The rule will then be return as the output for LearnOneRule.

Key Applications

Rule-based classification has been very popularly used in machine learning for classification of data. It is applicable whenever other classification scheme is applicable.

Future Directions

Despite efforts to reduce the number of rules in a rule-based classifier, the number of rules being used are still substantially higher than what a human can handle. More studies on new interestingness measure and visualization techniques are needed to further enhance the interpretability of a rule-based classifier.

Cross-references

▶ Associative Classifiers

▶ Association Rule Mining

▶ Decision Tree Induction

▶ Sequential Covering Algorithm BBb

Recommended Reading

1. Clark P. and Niblett T. The CN2 induction algorithm. Mach. Learn., 3(4):261–283, 1989.
2. Cohen W. Fast effective rule induction. In Proc. 12th Int. Conf. on Machine Learning, 1995, pp. 115–123.
3. Cong G., Tan K., Tung A., and Xu X. Mining top-K covering rule groups for gene expression data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 670–681.
4. Cong G. Tung A.K.H., Xu X., Pan F., and Yang J. FARMER: finding interesting rule groups in microarray datasets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 143–154.
5. Domingos P. The RISE system: conquering without separating. Tools with Artificial Intelligence, 1994. In Proc. 6th IEEE Int. Conf. on Tools with Artificial Intelligence, 1994, pp. 704–707.
6. Furnkranz J. and Widmer G. Incremental reduced error pruning. In Proc. 11th Int. Conf. on Machine Learning, 1994, pp. 70–77.
7. Hong J., Mozetic I., and Michalski R. AQ15: Incremental Learning of Attribute-Based Descriptions from Examples: The Method and User's Guide. Reports of the Intelligent Systems Group, ISG, pp. 86–5.
8. Liu B., Hsu W., and Ma Y. Integrating Classification and Association Rule Mining. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998.
9. Major J. and Mangano J. Selecting among rules induced from a hurricane database. J. Intell. Inform. Syst., 4(1):39–52, 1995.
10. Pan F., Cong G., and Tung A.K.H. CARPENTER: Finding closed patterns in long biological datasets. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003.
11. Quinlan J. Simplifying decision trees. Int. J. Man–Machine Studies, 27(3):221–234, 1987.
12. Quinlan J. Learning logical definitions from relations. Mach. Learn., 5(3):239–266, 1990.
13. Quinlan J. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA, 1993.
14. Quinlan J. and Cameron-Jones R. FOIL: A Midterm Report. In Proc. European Conf. on Machine Learning, 1993.
15. Smyth P. and Goodman R. An information theoretic approach to rule induction from databases. IEEE Trans. Knowl. Data Eng., 4(4):301–316, 1992.
16. Weiss S. and Indurkhya N. Predictive Data Mining: A Practical Guide. Morgan Kaufmann, Los Altos, CA, 1998.

S@n

► Success at n

Safety and Domain Independence

Rodney Topor
Griffith University, Nathan, QLD, Australia

Synonyms

Finiteness

Definition

The values in the relations of a relational database are elements of one or more underlying sets called domains. In practical applications, a domain may be infinite, e.g., the set of natural numbers. In this case, the value of a relational calculus query when applied to such a database may be infinite, e.g., $\{n \mid n \geq 10\}$. A query Q is called *finite* if the value of Q when applied to any database is finite.

Even when the database domains are finite, all that is normally known about them is that they are some finite superset of the values that occur in the database. In this case, the value of a relational calculus query may depend on such an unknown domain, e.g., $\{x \mid \forall y R(x, y)\}$. A query Q is called *domain independent* if the value of Q when applied to any database is the same for any two domains containing the database values or, equivalently, if the value of Q when applied to a database contains only values that occur in the database.

The term *safe* query has been used ambiguously in the literature. Often safe queries have been identified with finite queries. Sometimes safe queries have been members of a large, simple, decidable class of queries that are guaranteed to be finite, or, in other cases, domain independent. The use of word *safe* is preferred to denote a large, simple, decidable class of queries that are guaranteed to be domain independent and hence, normally finite.

Obviously, it is desirable that queries be finite and domain independent. Unfortunately, the classes of finite queries and domain independent queries are undecidable, which leads to a search for decidable classes of queries that can represent all, or as many as possible, finite (resp., domain independent) queries.

Historical Background

DiPaola [3] and independently Vardi [11] recognized the desirability that queries be domain independent and proved that the class of domain independent queries was undecidable.

Many researchers then attempted to define decidable classes of queries that were guaranteed to be domain independent. This work was summarized by Topor [9], Kifer [6], Ullman [10], and Abiteboul *et al.* [1]. Many different names such as range-restricted, allowed, safe, with subtle differences, were used in these definitions. Ullman [10], Van Gelder and Topor [12], and Abiteboul *et al.* [1] gave algorithms for translating queries in these classes into relational algebra for efficient evaluation.

Other researchers such as Escobar-Molano *et al.* [4], Hull and Su [5] and Suciu [8] attempted to define decidable classes of queries, *safe* queries, that were guaranteed to be finite in the presence of functions (e.g., arithmetic functions) over infinite domains (e.g., the natural numbers).

Whether or not there is a decidable class of queries that can express every finite (resp., domain independent) query depends critically on the particular set of functions on the domains. Stolboushkin and Taitslin [7] showed that, for many common domains, there is a decidable class of queries that can express every finite (resp. domain independent) query, but that there do exist domains for which there is no such decidable class. Benedikt and Libkin [2] extended and generalized these results to a wider class of domains.

Foundations

Following standard practice in the literature, assume that every database is defined over a single domain. The use of multiple domains complicates the presentation without introducing any substantially new concepts.

A *domain* $D = (U, O)$ consists of an underlying set U and a set of operations O on the set. The set may be infinite or finite. The operations may be represented as (infinite) relations over the set. Technically, each operation must be decidable. Often, one also wants the first-order theory of the domain to also be decidable. These conditions are satisfied in most common cases.

Examples of such domains include (a) any finite set of symbols possibly with an equality operator, (b) the set of natural numbers with addition and linear order operators, and (c) the set of finite strings over some finite alphabet with concatenation and lexicographic order operators.

A *database scheme* S is a finite set of pairs $\{(S_i, p_i)| 1 \leq i \leq k\}$, where each S_i is a relation name with arity $p_i \geq 1$.

Given a domain $D = (U, O)$, *a database (instance) I* of a scheme $\{(S_i, p_i) \mid 1 \leq i \leq k\}$ over D is a family of finite sets $\{R_i \mid 1 \leq i \leq k\}$, where each $R_i \subseteq U^{p_i}$.

A *query* over a database scheme is a first-order formula constructed from the relations in the scheme and the operators (and constants) in its domain. That is, the domain relational calculus is used as our query language.

A query Q over a database scheme S and domain $D = (U, O)$ is called *finite* if, for every database instance I of S over D, the value of Q when applied to I is a finite relation over U.

For example, over the domain of natural numbers, the query $Q_1 = \{n \mid n + 1 \leq 10\}$ is finite, but the query $Q_2 = \{n \mid 10 \leq n + 1\}$ is infinite.

It is desirable that queries be finite so that they may be composed and so that their results may be displayed.

The *active domain* of a database I is the finite set of domain elements that occur in the relations of I.

A query Q over a database scheme S and domain $D = (U, O)$ is called *domain independent (d.i.)* if, for every database instance I of S over D, the value of Q when applied to I contains only elements in the active domain of I. Equivalently, a query Q is domain independent if and only if, for every database instance I, and for every two extensions U_1 and U_2 of the active domain of D, the value of Q when applied to I over U_1 equals the value of Q when applied to I over U_2.

For example, over a finite domain of symbols without equality, the query $Q_3 = \{x \mid \exists y(P(x) \vee R(y))\}$ is *not* domain independent because, when applied to any database instance over this domain, the value consists all elements x that occur in P if R is empty, and *all* elements of the domain otherwise.

It is desirable that queries be domain independent so that their values are predictable despite the possibly unknown underlying domain.

It is natural to ask about the difference between finiteness and independence.

Clearly, if the domain is finite, all queries are finite, but the query Q_3 above is still not domain independent.

However, if the domain is infinite and the only operation on the domain is equality, then a query is finite if and only if it is domain independent [7].

Further, if the domain is infinite, even if there are operations other than equality, every domain independent query is also finite (as the active domain of every database instance is finite).

More interestingly, if the domain is the set of natural numbers and the only operation on the domain is linear order, then the query

$$Q_4 = \{x \mid \forall y(\Delta(y) \to x > y) \\ \wedge \forall y(y < x \to \exists z(\Delta(z) \wedge z \geq y)),$$

where $\Delta(y)$ is true if and only if y is in the active domain of the database, defines the smallest integer greater than all the active domain elements, and is hence finite but not domain independent [7].

Next, it is natural to ask whether it is possible to effectively recognize finite or domain independent queries. The answer is no. By reduction from standard undecidable problems in first-order logic, DiPaola [3] and, independently, Vardi [11] showed that, over any infinite domain, if the database scheme contains at least one relation of arity 2 or more, the classes of finite and domain independent queries are both undecidable. This undecidability result extends to domains such as the natural numbers with addition and linear order operations.

However, Benedikt and Libkin [2] show that finiteness (resp., d.i.) *is* decidable for Boolean combinations of conjunctive queries for a large class of domains over the real numbers.

Given that it is not possible to recognize whether or not a given query is finite (resp., d.i.) over a given domain D, two further questions arise naturally. First, is there is a *decidable* class of queries over D that can express all finite (resp., d.i.) queries over D? That is, is there a decidable class C of queries over D such that, for every finite (resp., d.i.) query Q there exists an equivalent query Q' in C? Second, can a large, simple, decidable class C' of queries be defined such that every query in C' is finite (resp., d.i.)? Historically, the second question was considered first, but they were considered in the order given.

For many domains, there is a decidable class of queries that can express every finite (resp., d.i.) query. Stolboushkin *et al.* [7] say that the finite (resp. d.i.) queries hence have an "effective syntax" for such domains. Examples of domains for which the finite (resp., d.i.) queries have an effective syntax include (a) an infinite domain of symbols in which the only operation is equality, (b) the domain of natural numbers with only the linear order operation, (c) the domain of natural numbers with the addition and linear order operations (Pressburger arithmetic), and (d) the domain of finite strings over a finite alphabet with the lexicographic order operation [7].

However, Stolboushkin and Taitslin [7] show that it is possible to construct an (artificial) domain with decidable operations and decidable first-order theory that does *not* have an effective syntax for the finite (resp., d.i.) queries. They also give an example of a domain with *undecidable* first-order theory (arithmetic) that has an effective syntax for the finite (resp., d.i.) queries.

Given the undecidability results above, many researchers, *e.g.*, [1,6,9,10]) have defined specific, decidable classes of queries that are guaranteed to be finite (resp. d.i.). Researchers have attempted to make these classes both simple and as large as possible. In many cases, they showed that queries in their class could express *all* finite (resp. d.i.) queries. These classes were given names such as safe, range-restricted, allowed, and many others. Most researchers restricted attention to domains with equality as the only operation, but some extended their work to the domain of the natural numbers with arithmetic and linear order operations, and some, *e.g.*, [2,4,5]) considered more arbitrary domains. Others, *e.g.*, [9]) applied these ideas to the domain independence of deductive databases.

The basic idea of these definitions is to ensure that every free variable in the query is somehow bound to an element in the active domain of the database or, in the presence of nontrivial operations, to one of a finite number of domain elements. In the absence of operations, this is typically done by ensuring that every free or existentially quantified variable in a query occurs positively in its scope, every universally quantified variable occurs negatively in its scope, and that the same free variables occur in every component of a disjunction. For example, the query $\{x \mid P(x) \wedge \forall y(Q(x, y) \rightarrow R(x, y))\}$ is safe according to these ideas. The equality operation is used to propagate a positive variable (or constant) from one side of the equality to the other. For example, the query $\{x \mid P(y) \wedge x = y\}$ is safe according to this idea. Finiteness dependencies are used with arithmetic operations over the natural numbers of concatenation operations over strings to propagate a positive variable (or constant) from one or more positions in the operation to one or more other positions. For example, the query $\{x \mid P(z) \wedge x + y = z \wedge \neg Q(y)\}$ is safe according to this idea. With deductive databases or, equivalently, with Datalog queries, it is necessary to require that every variable in the head of a rule occurs positively in the body of a rule and that the body of a rule is itself safe. In every case, the details of the definitions are too complicated to present here.

Finally, as relational calculus queries are evaluated in real database systems by translation into relational algebra, many researchers have studied techniques for translating safe queries (as defined in the previous two paragraphs) into equivalent relational algebra expressions. (A basic result that many researchers proved is that the class of safe queries is equivalent to the class of relational algebra queries.) These translations typically involve a sequence of transformations into increasingly restricted forms, until the translation into relational algebra is direct. Again, the details of the transformations too complicated to present here. See [1,10,12] for more information.

Key Applications

The concepts of finiteness, domain independence and safety are fundamental to our understanding of database queries over different domains. Tools that generate queries over specific domains should only generate safe queries. Query processors should check that input queries over specific domains are safe and should report warnings otherwise.

Future Directions

These questions of finiteness, domain independence and safety are well-understood and largely resolved for relational databases. However, additional work extending these ideas and methods to other data models and query languages may still be required.

Cross-references

- ▶ Complete Query Languages
- ▶ Conjunctive Query Language
- ▶ Constraint Query
- ▶ Query Language
- ▶ Relational Algebra
- ▶ Relational Calculus
- ▶ Relational Model

Recommended Reading

1. Abiteboul R., Hull R., and Vianu V. Foundations of Databases, Chapter 5. Addison-Wesley, Reading, MA, 1995, pp. 70–104.
2. Benedikt M. and Libkin L. Safe constraint queries, SIAM J. Comput., 29:1652–1682, 2000.
3. DiPaola R.A. The recursive unsolvability of the decision problem for the class of definite formulas. J. ACM, 16(2):324–327, 1969.
4. Escobar-Molano M., Hull., and Jacobs D. Safety and translation of calculus queries with scalar functions. In Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1993, pp. 253–264.
5. Hull R. and Su J. Domain independence and the relational calculus. Acta Inform., 31:513–524, 1994.
6. Kifer M. On Safety, Domain Independence, and Capturability of Database Queries (Preliminary Report). In Proc. 3rd Int. Conf. on Data and Knowledge Bases, 1988, pp. 405–415.
7. Stolboushkin A.P. and Taitslin M.A. Finite queries do not have effective syntax, Inform. Comput., 153(1):99–116, 1996.
8. Suciu D. Domain-independent queries on databases with external functions. Theor. Comput. Sci., 190(2):279–315, 1998.
9. Topor R.W. Domain independent formulas and databases. Theor. Comput. Sci., 52(3):281–306, 1987.
10. Ullman J.D. Principles of Database and Knowledge-Base Systems, Volume I, Sections 3.2 and 3.8. Computer Science Press, 1988, pp. 100–106 and 145–156.
11. Vardi M.Y. The decision problem for database dependencies. Inform. Process. Lett., 13(5):251–254, 1981.
12. Van Gelder A. and Topor R.W. Safety and translation of relational calculus aueries. ACM Trans. Database Syst., 16(2): 235–278, 1981.

Sagas

Kenneth Salem
University of Waterloo, Waterloo, ON, Canada

Definition

A saga [3] is a sequence of atomic transactions $T_1,...,T_n$ for which the following execution guarantee is made. Either the component transactions T_i will all commit in the order

$$T_1\ T_2,\ldots,T_n$$

in which case that saga is said to have committed, or one of the transaction sequences

$$T_1,\ldots,T_j\ C_j,\ldots,C_1$$

will be executed (for some $0 \leq j < n$), in which case the saga is said to have aborted. The transactions C_i are *compensating transactions* for the corresponding saga transactions T_i. Each transaction T_i in a saga must have a corresponding compensating transaction, which is responsible for undoing the T_i's effects.

Key Points

A saga is a type of extended transaction model [1] Each component transaction in a saga is executed atomically, but the saga itself is not atomic. Effects of the component transactions are visible to other operations as soon as those transactions commit, which may be well before the saga has finished.

The saga model guarantees execution of a compensating transaction for each component transaction that has already committed at the time that a saga aborts. This guarantee is known as *semantic atomicity* [2]. The saga model does not specify the nature of these compensations. Rather, the definition or identification of appropriate compensations is an application-specific task.

Sagas were originally proposed as a weaker substitute for the traditional atomic transaction model in situations for which atomicity would be expensive to enforce, e.g., when the traditional transaction would be long-running. Sagas and other extended transaction models have since found a variety of applications, such as workflow systems.

Cross-references

- ▶ Compensating Transactions
- ▶ Extended Transaction Model

- ▶ Open Nested Transaction Models
- ▶ Semantic Atomicity
- ▶ Workflow Management

Recommended Reading

1. Chrysanthis P.K. and Ramamritham K. Synthesis of extended transaction models using ACTA. ACM Trans. Database Syst., 19(3):450–491, 1994.
2. Garcia-Molina H. Using semantic knowledge for transaction processing in a distributed database. ACM Trans. Database Syst., 8(2):186–213, 1983.
3. Garcia-Molina H. and Salem K. Sagas. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 249–259.

Samba

▶ Storage Protocols

Sampling

▶ Matrix Masking

Sampling Techniques for Statistical Databases

Amarnath Gupta
University of California-San Diego, La Jolla, CA, USA

Definition

A sampling technique is a method by which one inspects only a small portion of data from a database to reduce the time to compute an aggregate query, but simultaneously ensuring that result computed on the sample faithfully represents the true results of the query for the entire data population.

Example: Acceptance-Rejection sampling (AR sampling) is sampling technique.

Key Points

Sampling is used in a database for different reasons such as (i) to estimate the results of aggregate queries (e.g., SUM, COUNT, or AVERAGE), (ii) to retrieve a sample of records from a database query for subsequent processing, (iii) for internal use by the query optimizer for selectivity estimation, (iv) to provide privacy protection for records on individuals contained in statistical databases. It has been determined that fixed size random sampling of data does not yield a true representation of the population. *Acceptance/rejection (A/R) sampling* is used to construct weighted samples in which the inclusion probabilities of a record are proportional to some arbitrary weight. *Reservoir sampling* is a form of sequential scan sampling algorithms which are used on files of unknown size to perform on-the-fly sampling from the results of a query. Methods have been developed to perform sampling not only from raw records, but also from B+-trees, hash structures, spatial data structures and so on.

Cross-references

- ▶ On-line Analytical Processing
- ▶ Privacy
- ▶ Secure Database Development
- ▶ Summarizability

Recommended Reading

1. Olken F. and Rotem D. Random sampling from databases: a survey. Stat. Comput., 5:25–42, 1995.

SAN

▶ Storage Area Network

SAN File System

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Synonyms

Shared-Disk File System

Definition

The term SAN file system refers to a file system which transfers file data directly to/from a storage device through a SAN. A SAN file system often has the capability of coherency control such that multiple servers may share the file system volume and simultaneously access files stored in the volume. The term shared disk file system is also used to refer to a SAN file system.

Key Points

In contrast to local file systems, SAN file systems allow multiple servers to share file system volumes directly and to access the same file simultaneously. To achieve this, the SAN file system has the capability of coherency control between servers. File system software running on each server may cache file data in a main memory buffer. Suppose that two servers, A and B, have cached the same file X. If A and B were to update X independently at the same time, two versions of X might be generated. The SAN file system needs to let each server be aware of the other servers. When server A updates a fragment of the file X, server B must be informed of a message of cache invalidation or changed information regarding the file X. Some SAN file systems exchange mutual exclusion messages to synchronize write accesses between servers.

The beneficial property of SAN file systems is high performance in comparison with network file systems such as NFS and CIFS. SAN file systems can directly transfer file data between servers and storage devices e.g., on a high-speed Fibre Channel network. In addition, SAN file systems do not need a central file server, which often becomes a bottleneck.

A variety of SAN file systems have been proposed and some of them have been deployed mainly into cluster computer systems often used to run scientific calculations. Recently, SAN file systems are also used as back-end file systems for large-scale enterprise NAS systems.

Cross-references

▶ Storage Network Architectures

Recommended Reading

1. Barrios M., Jones T., Kinnane S., Landzettel M., Al-Safran S., Stevens J., Stone C., Thomas C., and Troppens U. Sizing and tuning GPFS. IBM Redbook. SG24–5610–00, 1999.
2. Burns R.C., Rees R.M., and Long D.D.E. Semi-preemptible locks for a distributed file system. In Proc. 19th IEEE Int. Performance, Computing and Communications Conf., 2000, pp. 397–404.
3. Soltis S.R., Ruwart T.M., and O'Keefe M.T. The global file system. In Proc. 15th NASA Goddard Conference on Mass Storage Systems, 1996, pp. 319–342.

SARBAC

▶ Administration Model for RBAC

SAS

▶ Storage Protocols

SATA

▶ Storage Protocols

SBQL

▶ Stack-Based Query Language

SCA

▶ Service Component Architecture (SCA)

Scalable Classification Tree Construction

▶ Scalable Decision Tree Construction

Scalable Database Replication

▶ Replication for Scalability

Scalable Decision Support Systems High Performance Data Warehousing

▶ Parallel and Distributed Data Warehouses

Scalable Decision Tree Construction

Johannes Gehrke
Cornell University, Ithaca, NY, USA

Synonyms

Scalable classification tree construction; Scalable top-down decision tree construction; Tree-structured classifier

Definition

Decision trees are popular classification models. Decision trees are usually contructed greedily top-down from a training dataset. In many modern applications, the training dataset is very large and thus decision tree construction algorithms that scale with the size of the training dataset are needed.

Historical Background

Decision trees, in particular classification trees, have a long history both in the statistics [4] and the machine learning communities [12,13]. Scalability was not much a concern until the advent of data mining brought training datasets that were orders of magnitude larger than in traditional applications in machine learning and statistics.

Scalability concerns in classification started with the work by Agrawal et al. who presented an interval classfier that generated classification functions that distinguishes the different groups of training records based on their class label [1]. A follow-up paper introduces scalable construction of classification models as one of the three important classes of database mining problems [2], the other two being associations and sequences. The first scalable classification tree construction algorithm in the literature was SLIQ [9], which was then quickly followed by more algorithms that improved performance and allowed scaling up a more general class of algorithms from the machine learning and statistics literature [5,6,14–17].

Foundations

The input to a classification or regression problem is a dataset of *training records* (also called the *training database*). Each record has several attributes. Attributes whose domain is numerical are called *numerical attributes*, whereas attributes whose domain is not numerical are called *categorical attributes*. A *categorical* attribute takes values from a set of categories. Some authors distinguish between categorical attributes that take values in an unordered set (*nominal* attributes) and categorical attributes having ordered domains (*ordinal* attributes).

There is one distinguished attribute called the *dependent attribute*. The remaining attributes are called *predictor attributes*; they are either numerical or categorical. If the dependent attribute is categorical, the problem is referred to as a *classification problem* and the dependent attribute is called the *class label*. The elements of the domain of the class label attribute will also be denoted as *class labels*; the meaning of the term class label will be clear from the context. If the dependent attribute is numerical, the problem is called a *regression problem*. This entry concentrates on classification problems.

The goal of classification is to build a concise model of the distribution of the dependent attribute in terms of the predictor attributes. The resulting model is used to assign values to a database where the values of the predictor attributes are known but the value of the dependent attribute is unknown. This entry surveys research on scalable classification tree construction from the database literature. An excellent survey of other aspects of decision tree construction can be found in Murthy [11].

Problem Definition

Let $X_1,\ldots,X_m$, C be random variables where X_i has domain $\mathtt{dom}(X_i)$; assume without loss of generality that $\mathtt{dom}(C) = \{1,2,\ldots,J\}$. A *classifier* is a function

$$d : \mathtt{dom}(X_1) \times \ldots \times \mathtt{dom}(X_m) \mapsto \mathtt{dom}(C).$$

Let $P(X',C')$ be a probability distribution on $\mathtt{dom}(X_1) \times\ldots\times \mathtt{dom}(X_m)\times \mathtt{dom}(C)$ and let $t = \langle t.X_1,\ldots,t.X_m, t.C\rangle$ be a record randomly drawn from P, i.e., t has probability $P(X', C')$ that $\langle t.X_1,\ldots,t.X_m\rangle \in X'$ and $t.C \in C'$. Define the *misclassification rate* R_d of classifier d to be $P(d(\langle t.X_1,\ldots,t.X_m\rangle) \neq t.C)$. The training database D is a random sample from P, the X_i correspond to the predictor attributes and C is the class label attribute.

A *decision tree* is a special type of classifier. It is a directed, acyclic graph T in the form of a tree. The focus here is on binary decision trees although these techniques can be generalized to non-binary decision trees. If a node has no outgoing edges it is called a *leaf node*, otherwise it is called an *internal node*. Each leaf node is labeled with one class label; each internal node n is labeled with one predictor attribute X_n called the *splitting attribute*. Each internal node n has a predicate q_n, called the *splitting predicate* associated with it. If X_n is a numerical attribute, q_n is of the form $X_n \leq x_n$, where $x_n \in \mathtt{dom}(X_n)$; x_n is called the *split point* at node n. If X_n is a categorical attribute, q_n is of the form $X_n \in Y_n$ where $Y_n \subset \mathtt{dom}(X_n)$; Y_n is called the *splitting subset* at node n. The combined information of splitting attribute and splitting predicates at node n is called the *splitting criterion* of n. An example training database is shown in Fig. 1, and a sample classification tree is shown in Fig. 2.

With each node $n \in T$ there is assciated a predicate

$$f_n : \mathtt{dom}(X_1) \times \ldots \times \mathtt{dom}(X_m) \mapsto \{\mathtt{true}, \mathtt{false}\},$$

called its *node predicate* as follows: For the root node n, $f_n \overset{\text{def}}{=} true$. Let n be a non-root node with parent p whose splitting predicate is q_p. If n is the right child of p, define $f_n \overset{\text{def}}{=} f_p \wedge q_p$; if n is the right child of p, define $f_n \overset{\text{def}}{=} f_p \wedge \neg q_p$. Informally, f_n is the conjunction of all splitting predicates on the internal nodes on the path from the root node to n. Since each leaf node $n \in T$ is labeled with a class label, n encodes the classification rule $f_n \rightarrow c$, where c is the label of n. Thus the tree T encodes a function $T : \mathtt{dom}(X_1) \times \ldots \times \mathtt{dom}(X_m) \mapsto \mathtt{dom}(C)$ and is therefore a classifier, called a *decision tree classifier*. (Both the tree as well as the induced classifier will be denoted by T; the semantics will be clear from the context.) For a node $n \in T$ with parent p, the *family of tuples* F_n is the set of records in D that follows the path from the root to n when being processed by the tree, formally

$$F_n \overset{\text{def}}{=} \{t \in D : f_n(t)\}.$$

Also define $F_n{}^i$ for $i \in \{1,\ldots,J\}$ as the set of records in F_n with class label i, formally

$$F_n^i \overset{\text{def}}{=} \{t \in D : f_n(t) \wedge t.C = i\}.$$

The problem of classification tree construction can now be stated formally: Given a dataset $D = \{t_1,\ldots,t_n\}$ where the t_i are independent random samples from an unknown probability distribution P, find a decision tree classifier T that minimizes the misclassification rate $R_T(P)$.

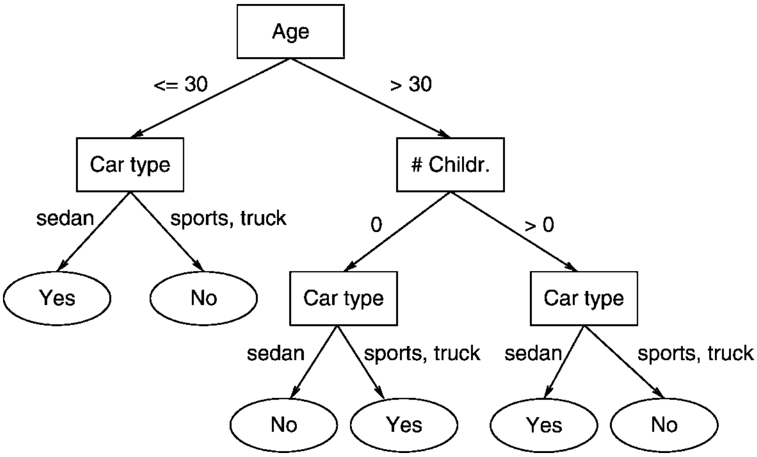
A classification tree is usually constructed in two phases. In phase one, the *growth phase*, an overly large decision tree is constructed from the training data. In phase two, the *pruning phase*, the final size of the tree T is determined with the goal to minimize R_T. It is possible to interleave growth and pruning phase for performance reasons as in the PUBLIC Pruning Method described later in this entry. Nearly all decision tree construction algorithms grow the tree top-down in the following greedy way: At the root node n, the training database is examined and a splitting criterion for n is selected. Recursively, at a non-root node n, the family of n is examined and from it a splitting criterion is selected. This schema is depicted in Fig. 3.

During the tree growth phase, two different algorithmic issues need to be addressed. The first issue is to devise an algorithm such that the resulting tree T minimizes R_T; this part of the overall decision tree construction algorithm is called the *split selection method*. The second issue is to devise a *data access method* for data management in the case that the training database is very large. During the pruning phase a third issue arises, namely how to find a good estimator $\widehat{R_T}$ of R_T and how to efficiently calculate $\widehat{R_T}$.

A popular class of split selection methods are *impurity-based* split selection methods [4,12]. Impurity-based split selection methods find the splitting criterion by minimizing a concave *impurity function* $\mathtt{imp}_\theta$ such as the entropy [12] or the $\mathtt{gini}$-index [4]. (Arguments for the concavity of the impurity function can be found in Breiman et al. [4].) The most popular split selection methods such as CART [4] and C4.5 [12] fall

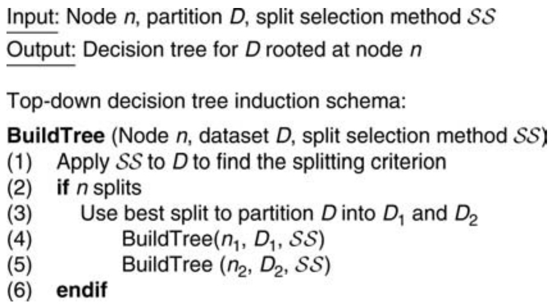
Record Id	Car	Age	Children	Subscription
1	sedan	23	0	yes
2	sports	31	1	no
3	sedan	36	1	no
4	truck	25	2	no
5	sports	30	0	no
6	sedan	36	0	no
7	sedan	25	0	yes
8	truck	36	1	no
9	sedan	30	2	yes
10	sedan	31	1	yes
11	sports	25	0	no
12	sedan	45	1	yes
13	sports	23	2	no
14	truck	45	0	yes

Scalable Decision Tree Construction. Figure 1. Example training database.



Scalable Decision Tree Construction. Figure 2. Magazine Subscription Example Classification Tree.

into this group. At each node, all predictor attributes X are examined and the impurity $\texttt{imp}_\theta$ of the best split on X is calculated. The final split is chosen such that the combination of splitting attribute and splitting predicates minimizes the value of $\texttt{imp}_\theta$.



```
Input: Node $n$, partition $D$, split selection method $\mathcal{SS}$
Output: Decision tree for $D$ rooted at node $n$

Top-down decision tree induction schema:
BuildTree (Node $n$, dataset $D$, split selection method $\mathcal{SS}$)
    Apply $\mathcal{SS}$ to $D$ to find the splitting criterion
    if $n$ splits
        Use best split to partition $D$ into $D_1$ and $D_2$
            BuildTree($n_1$, $D_1$, $\mathcal{SS}$)
            BuildTree ($n_2$, $D_2$, $\mathcal{SS}$)
    endif
```

Scalable Decision Tree Construction. Figure 3. Classification tree construction.

Data Access

There exist many scalable data access methods for classification tree construction. Some of the issues in scalable decision tree construction are introduced by briefly discussing one method, RainForest [6].

An examination of the split selection methods in the literature reveals that the greedy schema can be refined to the generic *RainForest Tree Induction Schema* shown in Fig. 4. A broad class of split selection methods, namely those that generate splitting criteria involving a single splitting attribute, proceed according to this generic schema. Split selection methods that generate linear combination splits cannot be captured by RainForest. Consider a node n of the decision tree. The split selection method has to make two decisions while examining the family of n: (i) It has to select the splitting attribute X, and (ii) it has to select the splitting predicates on X. Once decided on the splitting criterion, the algorithm is recursively applied to each of the children of n. Denote by $\mathcal{SS}$ a representative split selection method.

Note that at a node n, the utility of a predictor attribute X as a possible splitting attribute is examined independent of the other predictor attributes: The *sufficient statistics* are the class label distributions for each distinct attribute value of X. Define the *AVC-set* of a predictor attribute X at node n to be the projection of F_n onto X and the class label where counts of the individual class labels are aggregated. Denote the AVC-set of predictor attribute X at node n by $\text{AVC}_n(X)$. (The acronym AVC stands for **A**ttribute-**V**alue, **C**lasslabel.) To give a formal definition, let $a_{n,X,x,i}$ be the number of records t in F_n with attribute value $t.X = x$ and class label $t.C = i$. Formally,

$$a_{n,X,x,i} \overset{\text{def}}{=} |\{t \in F_n \ :\ t.X = x \ \wedge\ t.C = i\}|.$$

For a predictor attribute X, let $S \overset{\text{def}}{=} \texttt{dom}(X) \times \mathbf{N}^J$ where $\mathbf{N}$ denotes the set of natural numbers. Then

$$\text{AVC}_n(X) \overset{\text{def}}{=} \{(x, a_1,\ldots,a_J) \in S : \exists t \in F_n \ : \\ (t.X = x \ \wedge\ \forall i \in \{1,\ldots,J\} \ :\ a_i = a_{n,X,x,i}\ )\}.$$

Define the *AVC-group* of a node n to be the set of the AVC-sets of all predictor attributes at node n. Note that the size of the AVC-set of a predictor attribute X at

```
RainForest refinement to the schema in Fig 3:
(1a)  for each predictor attribute X
(1b)    Construct the AVC-set of X
(1c)    Call SS.find_best_partitioning(AVC-set of X)
(1d)  endfor
(2a)  SS.decide_splitting_criterion();
(2b)  if n splits...
```

Scalable Decision Tree Construction. Figure 4. RainForest refinement.

Car	Subscription	
	Yes	No
sedan	5	2
sports	0	4
truck	1	2

Age	Subscription	
	Yes	No
23	1	1
25	1	2
30	1	1
31	1	1
36	0	3
45	2	0

Scalable Decision Tree Construction. Figure 5. Rainforest AVC-sets.

Car	Subscription	
	Yes	No
sedan	3	0
sports	0	3
truck	0	1

Age	Subscription	
	Yes	No
23	1	1
25	1	2
30	1	1

Scalable Decision Tree Construction. Figure 6. Rainforest AVC-sets of the left child of the root node.

node n depends only on the number of distinct attribute values of X and the number of class labels in F_n.

As an example, consider the training database shown in Fig. 1. The AVC group of the root node is depicted in Fig. 5. Assume that the root node splits as shown in Fig. 2. The AVC-group of the left child node of the root node is shown in Fig. 5 and the AVC-group of the left child node of the root node is shown in Fig. 6.

If the training database is stored inside a database system, the AVC-set of a node n for predictor attribute X can be retrieved through a simple SQL-query:

```
SELECT      D.X, D.C, COUNT(*)
FROM        D
WHERE       f_n
GROUP BY    D, X, D.C
```

In order to construct the AVC-sets of all predictor attributes at a node n, a UNION-query would be necessary. (In this case, the SELECT clause needs to retrieve also some identifier of the attribute in order to distinguish individual AVC-sets.) Graefe et al. observe that most database systems evaluate the UNION-query through several scans and introduce a new operator that allows gathering of sufficient statistics in one database scan [7].

Based on this observation, there exist several algorithms that construct as many AVC-sets as possible in main memory while minimizing the number of scans over the training database. As an example of the simplest such algorithm, assume that the complete AVC-group of the root node fits into main memory. Then the tree can be constructed according to the following simple schema: Read the training database D and construct the AVC-group of the root node n in-memory. Then determine the splitting criterion from the AVC-sets through an in-memory computation. Then make a second pass over D and partition D into children partitions D_1 and D_2. This simple algorithm reads the complete training database twice and writes the training database once per level of the tree; more sophisticated algorithms are possible [6]. Experiments show that RainForest outperforms SPRINT on the average by a factor of three. Note that RainForest has a large memory requirement: RainForest is only applicable if the AVC-group of the root node fits in-memory (this requirement can be relaxed through more sophisticated memory management [6]).

Tree Pruning

The pruning phase of classification tree construction decides on the tree of the right size in order to prevent overfitting to minimize the misclassification error $R_T(P)$. In bottom-up pruning, in the tree growth phase the tree is grown until the size of the family of each leaf node n falls below a user-defined threshold c; the pruning phase follows the growth phase. Examples of bottom-up pruning strategies are cost-complexity pruning, pruning with an additional set of records called a test set [4], and pruning based on the MDL-principle [10]. In top-down pruning, during the growth phase a statistic s_n is computed at each node n, and based on the value of s_n, tree growth at node n is continued or stopped [13]. Bottom-up pruning results usually in trees of higher quality [4,8,13], but top-down pruning is computationally more efficient since no parts of the tree are first constructed and later discarded.

The section describes the PUBLIC pruning algorithm [14], an algorithm that integrates bottom-up pruning into the tree growth phase; thus PUBLIC preserves the computational advantages of top-down pruning while preserving the good properties of top-down pruning. PUBLIC uses pruning based on the MDL principle [10], which sees the classification tree as a means to encode the values of the class label attribute given the predictor attributes X_1,..., X_m. The MDL principle states that the "best" classification tree is the tree can be encoded with the least number of bits. Thus there needs to be an encoding schema that allows encoding of any binary decision tree. Given an encoding schema, a classification tree can be pruned by selecting the subtree with minimum code length.

In the MDL encoding schema for binary splitting predicates from Mehta et al. [10], each node requires one bit to encode its type (leaf or intermediate node). An intermediate node n needs to encode its splitting criterion, consisting of the splitting attribute X (log m bits since there are m predictor attributes) and splitting predicate. Let X be the splitting attribute at node n and assume that X has v different attribute values. If X is a numerical attribute, the split will be of the form $X \leq c$. Since c can take $v - 1$ different values, encoding of the split point c requires $\log(v - 1)$ bits. If X is a categorical attribute, the split will be of the form $X \in Y$. Since Y can take $2^v - 2$ different values, the encoding requires $\log(2^v - 2)$ bits. Denote the cost of a split at a node n by $C_{Split}(n)$. For a leaf node n, Metha et al. show that the cost of encoding the leaf is [10]:

$$C_{leaf}(n) = \sum_i n_i \log \frac{|F_n|}{|F_n^i|} + \frac{k-1}{2} \log \frac{|F_n|}{2} + \log \frac{\pi^{k/2}}{\Gamma(k/2)}.$$

Given this encoding schema, a fully grown tree can be pruned bottom-up by deciding for each node whether it should be pruned or whether it should remain [10].

The PUBLIC algorithm integrates the building and pruning phase by computing a lower bound $L(n)$ on the MDL-cost of any subtree rooted at a node n. A trivial lower bound is $L(n) = 1$ (for the encoding of n). Rastogi and Shim give in their PUBLIC algorithms more sophisticated lower bounds including the following:

PUBLIC Lower Bound [14]

Consider a (sub-)tree T with $s > 1$ nodes rooted at node n. Then the cost $C(n)$ of encoding T has the following lower bound:

$$C(n) \geq 2 \cdot (s-1) + 1 + (s-1) \cdot \log m + \sum_{i=s+1}^{k} |F_n^i|$$

With this lower bound the MDL Pruning Schema can be used even during top-down tree construction. PUBLIC distinguishes two different types of leaf nodes: "True" leaf nodes that are the result of pruning or that cannot be expanded any further, and "intermediate" leaf nodes n, where the subtree rooted at n might be grown further. During the growth phase, the PUBLIC Pruning Schema is executed from the root node of the tree. Rastogi and Shim show experimentally that integration of pruning with the growth phase of the tree results in significant savings in overall tree construction. More sophisticated ways of integrating tree growth with pruning in addition to tighter lower bounds are possible [14].

Key Applications

Classification has a wide range of applications, including scientific experiments, medical diagnosis, fraud detection, credit approval, and target marketing.

Cross-references

▶ Classification
▶ Data Mining

Recommended Reading

1. Agrawal R., Ghosh S.P., Imielinski T., Iyer B.R., and Swami A.N. An interval classifier for database mining applications. In Proc. 18th Int. Conf. on Very Large Data Bases, 1992. pp. 560–573.
2. Agrawal R., Imielinski T., and Swami A.N. Database mining: a performance perspective. IEEE Trans. Knowl. Data Eng., 5(6): 914–925, 1993.
3. Alsabti K., Ranka S., and Singh V. Clouds: a decision tree classifier for large datasets. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998, pp. 2–8.
4. Breiman L., Friedman J.H., Olshen R.A., and Stone C.J. Classification and regression trees. Wadsworth, Belmont, 1984.
5. Gehrke J., Ganti V., Ramakrishnan R., and Loh W.-Y. BOAT – Optimistic decision tree construction. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 169–180.
6. Gehrke J., Ramakrishnan R., and Ganti V. Rainforest – a framework for fast decision tree construction of large datasets. Data Min. Knowl. Dis., 4(2/3):127–162, 2000.
7. Graefe G., Fayyad U., and Chaudhuri S. On the efficient gathering of sufficient statistics for classification from large

SQL databases. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998, pp. 204–208.

8. Lim T.-S., Loh W.-Y., and Shih Y.-S. A comparison of prediction accuracy, complexity, and training time of 33 old and new classification algorithms. Mach. Learn., 48:203–228, 2000.
9. Mehta M., Agrawal R., and Rissanen J. SLIQ: A fast scalable classifier for data mining. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996.
10. Mehta M., Rissanen J., and Agrawal R. MDL-based decision tree pruning. In Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining, 1995.
11. Murthy S.K. Automatic construction of decision trees from data: a multi-disciplinary survey. Data Min. Knowl. Dis., 2(4): 345–389, 1998.
12. Quinlan J.R. Induction of decision trees.Mach. Learn., 1:81–106, 1986.
13. Quinlan J.R. C4.5: Programs for Machine Learning. Morgan Kaufman, 1993.
14. Rastogi R. and Shim K. PUBLIC: a decision tree classifier that integrates building and pruning. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 404–415.
15. Shafer J., Agrawal R., and Mehta M. SPRINT: a scalable parallel classifier for data mining. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996.
16. Sreenivas M.K., AlSabti K., and Ranka S. Parallel out-of-core decision tree classiers. In Advances in Distributed and Parallel Knowledge Discovery. Kargupta H. and Chan P. (eds.). AAAI. 2000, pp. 317–336.
17. Srivastava A., Han E., Kumar V., and Singh V. Parallel formulations of decision-tree classication algorithms. Data Min. Knowl. Dis., 3(3), 1999.

Scalable Replication

▶ Replication in Multi-Tier Architectures

Scalable Top-Down Decision Tree Construction

▶ Scalable Decision Tree Construction

Scale Out

▶ Replication for Scalability
▶ Replication in Multi-Tier Architectures

Scale-Out Databases

▶ Process Structure of a DBMS

Scale-Up Databases

▶ Process Structure of a DBMS

Scaling

▶ Zooming Techniques

Scanning

▶ Browsing

Scene Change Detection

▶ Video Segmentation

Scheduler

Nathaniel Palmer
Workflow Management Coalition, Hingham,
MA, USA

Synonyms

Workflow scheduler; Queuing mechanism

Definition

The mechanism that identifies and initiates the sequence for activities and work items are executed.

Key Points

A Scheduler initiates work assignments based on precedence relationships and the state of a workflow instance. A Scheduler is not a "queue" which is typically a set of work items or activities waiting to be scheduled, however, a queue of items may be managed by the Scheduler. The role of the Scheduler is to minimize queue time and optimize executive efficiency.

Cross-references

▶ Activity
▶ Work Item

Scheduling

▶ Concurrency Control – Traditional Approaches

Scheduling Policies

▶ Scheduling Strategies for Data Stream Processing

Scheduling Strategies for Data Stream Processing

Mohamed Sharaf[1], Alexandros Labrinidis[2]
[1]Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada
[2]Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

Operator scheduling; Continuous query scheduling; Scheduling policies

Definition

In a Data Stream Management System (DSMS), data arrives in the form of continuous streams from different data sources, where the arrival of new data triggers the execution of multiple continuous queries (CQs). The order in which CQs are executed in response to the arrival of new data is determined by the CQ scheduler. Thus, one of the main goals in the design of a DSMS is the development of scheduling policies that leverage CQ characteristics to optimize the DSMS performance.

Historical Background

The growing need for *monitoring applications* [8] has forced an evolution on data processing paradigms, moving from Database Management Systems (DBMSs) to Data Stream Management Systems (DSMSs) [4,11]. Traditional DBMSs employ a store-and-then-query data processing paradigm, where data are stored in the database and queries are submitted by the users to be answered in full, based on the current snapshot of the database. In contrast, in DSMSs, monitoring applications register continuous queries which continuously process unbounded data streams looking for data that represent events of interest to the end-user.

The data stream concept permeated the data management research community in the mid- to late 90's, with general-purpose research prototypes of data stream management systems materializing shortly afterwards, for example Aurora[8], TelegraphCQ[10] and STREAMS[5].

Scheduling is one of the fundamental research challenges for effective data stream management systems; as such, it has received a lot of attention, with early works on scheduling in 2003 [2,9].

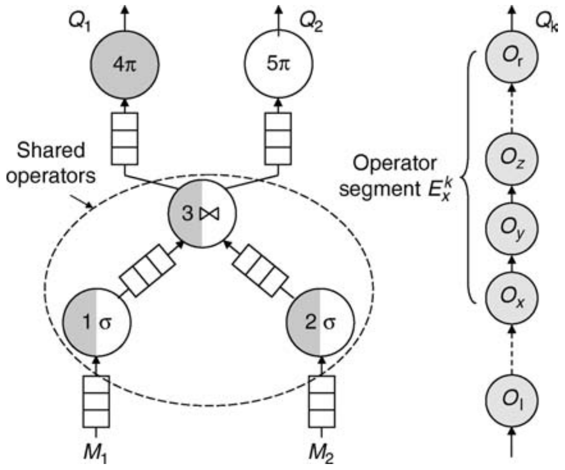
Foundations

System Model

A continuous query evaluation plan can be conceptualized as a data flow tree [2,8], where the nodes are operators that process tuples and edges represent the flow of tuples from one operator to another (Fig. 1). An edge from operator O_x to operator O_y means that the output of O_x is an input to O_y. Each operator is associated with a *queue* where input tuples are buffered until they are processed.

Multiple queries with common sub-expressions are usually merged together to eliminate the repetition of similar operations. For example, Figure 1 shows the global plan for two queries Q_1 and Q_2. Both queries operate on data streams M_1 and M_2 and they share the common sub-expression represented by operators O_1, O_2 and O_3, as illustrated by the half-shaded pattern for these operators.

A *single-stream query* Q_k has a single *leaf* operator Q_l^k and a single *root* operator Q_r^k, whereas a *multi-stream* query has a single root operator and more than one leaf operators. In a query plan Q_k, an



Scheduling Strategies for Data Stream Processing. Figure 1. Continuous queries plans.

operator segment $E^k_{x,y}$ is the sequence of operators that starts at O^k_x and ends at O^k_x. If the last operator on $E^k_{x,y}$ is the root operator, then that operator segment is simply denoted as E^k_x. For example, in Fig. 1, $E^1_1 = <O_1, O_3, O_4>$, whereas $E^2_1 = <O_1, O_3, O_5>$.

In a query, each operator $O_x{}^k$ (or simply O_x) is associated with two parameters:

1. *Processing cost* or *Processing time* (c_x) is the amount of time needed to process an input tuple.
2. *Selectivity* or *Productivity* (s_x) is the number of tuples produced after processing one tuple for c_x time units. s_x is less than or equal to 1 for a filter operator and it could be greater than 1 for a join operator.

Multiple CQ Scheduling

At the arrival of new data, the MCQ scheduler decides the execution order of CQs, or more precisely, the execution order of operators within CQs. The execution order is decided with the objective of optimizing the DSMS performance under certain metrics. Towards this, the scheduler assigns a priority to each operator and operators are executed according to these priorities.

For a single-stream query Q_k which consists of operators $<O^k_l,...,O^k_x, O^k_y,...,O^k_r>$ (Fig. 1), the function for computing the priority of operator O^k_x typically involves one or more of the following parameters:

- *Operator Global Selectivity (S^k_x) is the number of tuples produced at the root O^k_r after processing one tuple along operator segment E^k_x.*

$$S^k_x = s^k_x \times s^k_y \times ... \times s^k_r$$

- *Operator Global Average Cost* ($\overline{C}^k_x$) is the expected time required to process a tuple along an operator segment C^k_x.

$$\overline{C}^k_x = (c^k_x) + (c^k_y \times s^k_x) + ... + (c^k_r \times s^k_{r-1} \times ... \times s^k_x)$$

If O^k_x is a leaf operator ($x = l$), when a processed tuple actually satisfies all the filters in E^k_l, then $\overline{C}^k_l$ represents the ideal total processing cost or time incurred by any tuple *produced* or *emitted* by query Q_k. In this case, $\overline{C}^k_l$ is denoted as T_k:

- *Tuple Processing Time* (T_k) is the ideal total processing cost required to produce a tuple by query Q_k.

$$T_k = c^k_l + ... + c^k_x + c^k_y + ... + c^k_r$$

The exact priority function depends on the performance metric to optimize, and in turn on the employed scheduling strategy.

Metrics and Strategies

Response Time: Processing a tuple by a CQ might lead to discarding it (if it does not satisfy some filter predicate) or it might lead to producing one or more tuples at the output, which means that the input tuple represents an event of interest to the user who registered the CQ. Clearly, in DSMSs, it is more appropriate to define response time from a data/event perspective rather than from a query perspective as in traditional DBMSs. Hence, the *tuple response time* or *tuple latency* is defined as follows:

Definition 1

Tuple response time, R_i, for tuple t_i is $R_i = D_i - A_i$, where A_i is t_i's arrival time and D_i is t_i's output time. Accordingly, the average response time for N tuples is: $\frac{1}{N}\sum_{i=1}^{N} R_i$.

For a single CQ over multiple data streams, the *Rate-based* policy (*RB*) has been shown to improve the average response time of tuples processed by that CQ [17].

For multiple CQs, the Aurora DSMS [9], uses a two-level scheduling strategy where *Round Robin (RR)* is used to schedule queries and *RB* is used to schedule operators within the query. The work in [14] proposes the *Highest Rate* policy (*HR*) which extends the *RB* to schedule both queries and operators. Basically, *HR* views the network of multiple queries as a set of operators and at each scheduling point it selects for execution the operator with the highest priority (i.e., output rate).

Specifically, under *HR*, each operator O^k_x is assigned a value called *global output rate* (GR^k_x). The output rate of an operator is basically the expected number of tuples produced per time unit due to processing one tuple by the operators along the operator segment starting at O^k_x all the way to the root O^k_r. Formally, the output rate of operator O^k_x is defined as follows:

$$GR^k_x = \frac{S^k_x}{\overline{C}^k_x} \qquad (1)$$

where S^k_x and $\overline{C}^k_x$ are the operator's global selectivity and global average cost as defined above. The intuition underlying *HR* is to give higher priority to operator paths that are both productive and inexpensive. In other words, the highest priority is given to the operator paths with the minimum latency for producing one tuple.

Slowdown: Under a heterogeneous workload, the processing requirements for different tuples may vary significantly and average response time is not an appropriate metric, since it cannot relate the time spent by a tuple in the system to its processing requirements. Given this realization, other on-line systems with heterogeneous workloads such as DBMSs, OSs, and Web servers have adopted *average slowdown* or *stretch* [13] as another metric. This motivated considering the stretch metric in [14].

The definition of slowdown was initiated by the database community in [12] for measuring the performance of a DBMS executing multi-class workloads. Formally, the slowdown of a job is the ratio between the time a job spends in the system to its processing demands [13]. In a DSMS, the slowdown of a tuple is defined as follows [14]:

Definition 2

The slowdown, H_i, for tuple t_i produced by query Q_k is $H_i = \frac{R_i}{T_k}$, where R_i is t_i's response time and T_k is its ideal processing time. Accordingly, the average slowdown for N tuples is: $\frac{1}{N}\sum_{i=1}^{N} H_i$.

Intuitively, in a general purpose DSMS where all events are of equal importance, a simple event (i.e., an event detected by a low-cost CQ) should be detected faster than a complex event (i.e., an event detected by a high-cost CQ) since the latter contributes more to the load on the DSMS.

The *HR* policy schedules jobs in descending order of output rate which might result in a high average slowdown because a low-cost query can be assigned a low priority since it is not productive enough. Those few tuples produced by this query will all experience a high slowdown, with a corresponding increase in the average slowdown of the DSMS.

The work in [14] proposes the *Highest Normalized Rate (HNR)* policy for minimizing the slowdown in a DSMS. Under HNR, each operator O_x^k is assigned a priority V_x^k which is the *weighted rate* or *normalized rate* of the operator segment E_x^k that starts at operator O_x^k and it is defined as:

$$V_x^k = \frac{1}{T_k} \times \frac{S_x^k}{\overline{C}_x^k} \qquad (2)$$

The *HNR* policy, like *HR*, is based on output rate, however, it also emphasizes the ideal tuple processing time in assigning priorities. As such, an inexpensive operator segment with low productivity will get a higher priority under *HNR* than under *HR*.

Worst-Case Performance: It is expected that a scheduling policy that strives to minimize the average-case performance might lead to a poor worst-case performance under a relatively high load. That is, some queries (or tuples) might starve under such a policy. The worst-case performance is typically measured using *maximum response time* or *maximum slowdown* [7].

Intuitively, a policy that optimizes for the worst-case performance should be pessimistic. That is, it assumes the worst-case scenario where each processed tuple will satisfy all the filters in the corresponding query.

The work in [14] shows that the traditional *First-Come-First-Serve (FCFS)* minimizes the maximum response time. Similarly, it shows that the traditional *Longest Stretch First (LSF)* [1] optimizes the maximum slowdown.

Average- vs. Worst-Case Performance: On one hand, the average value for a QoS metric provided by the system represents the expected QoS experienced by any tuple in the system (i.e., the average-case performance). On the other hand, the maximum value measures the worst QoS experienced by some tuple in the system (i.e., the worst-case performance). It is known that each of these metrics by itself is not enough to fully characterize system performance.

The most common way to capture the trade-off between the average-case and the worst-case performance is to measure the ℓ_2 norm [6]. For instance, the ℓ_2 norm of response times, R_i, is defined as:

Definition 3

The ℓ_2 norm of response times for N tuples is equal to $\sqrt{\sum_1^N R_i^2}$.

The definition shows that the ℓ_2 norm considers the average in the sense that it takes into account all values, yet, by considering the second norm of each value instead of the first norm, it penalizes more severely outliers compared to the average metrics.

In order to balance the trade-off between the average- and worst-case performance, the *Balance Slowdown (BSD)* and the *Balance Response Time (BRT)* policies have been proposed in [14]. To avoid starvation, the two policies consider the amount of time an operator O_x^k has been waiting for scheduling (i.e., W_x^k). Specifically, under *BSD*, each operator O_x^k is assigned a priority value V_x^k which is the product of the operator's normalized rate and the current highest slowdown of its pending tuples. That is:

$$V_x^k = \left(\frac{S_x^k}{\overline{C}_x^k T_k}\right)\left(\frac{W_x^k}{T_k}\right) \tag{3}$$

As such, under *BSD*, an operator is selected either because it has a high weighted rate or because its pending tuples have acquired a high slowdown.

Application-Specific QoS: Aurora also proposes a QoS-aware scheduler which attempts to satisfy application-specified QoS requirements [9]. Specifically, under that QoS-aware scheduler, each query is associated with a QoS graph which defines the utility of stale output.

Given, a QoS graph, the scheduler computes for each operator a *utility* value which is basically the slope of the QoS graph at the tuple's output time. The scheduler also computes for each operator its *urgency* value which is an estimation of how close is an operator to a critical point on the QoS graph where the QoS changes sharply. Then, at each scheduling point, the scheduler chooses for execution the operators with the highest utility value and among those that have the same utility, it chooses the one that has the highest urgency.

Memory Usage: Multi-query scheduling has also been exploited to optimize metrics beyond QoS. For example, *Chain* is a multi-query scheduling policy that optimizes memory usage in order to minimize space requirements for buffering tuples [2]. Towards this, for each query plan, *Chain* constructs what is called a *progress chart*. A progress chart is basically a set of segments where the slope of each segment represents the rate of change in the size of a tuple being processed by a set of consecutive operators along the query plan. Given that progress chart, at each scheduling point, *Chain* schedules for execution the tuple that lies on the segment with the steepest slope. The intuition is to give higher priority to segments of operators with higher tuple consumption rate which will lead to quickly freeing more memory.

Quality of Data (QoD): Another metric to optimize is Quality of Data (*QoD*). For instance, the work in [15] proposes the *freshness-aware* scheduling policy for improving the QoD of data streams, when QoD is defined in terms of freshness. The proposed scheduler exploits the variability in query costs, divergence in arrival patterns, and the probabilistic impact of selectivity in order to maximize the freshness of output data streams.

Multiple-Objective Scheduling: In DSMSs, and in computer systems in general, it is often desirable to optimize for multiple metrics at the same time. However, those metrics might be in conflict most of the time. This motivated the proposals of schedulers that are able to balance the trade-off between certain conflicting metrics.

For instance, the work in [3] attempts to balance the trade-off between memory usage and latency by formalizing latency requirements as a constraint to the *Chain* scheduler. This formulation lead to the *Mixed* policy which can be viewed as a heuristic strategy that is intermediate between Chain and FIFO. Specifically, *Mixed* is tuned via a parameter where a high value of that parameter causes *Mixed* to behave more like *FIFO*, whereas a lower value makes it behave more like *Chain*.

In another attempt towards multiple-objective scheduling, the work in [16] proposes *AMoS* which is an Adaptive Multi-objective Scheduling selection framework. Given several scheduling algorithms, AMoS employs a learning mechanism to learn the behavior of the scheduling algorithms over time. It then uses the learned knowledge to continuously select the algorithm that has statistically performed the best.

Scheduler Implementation: To ensure the applicability of scheduling policies in DSMSs, a low-overhead implementation is needed in order to reduce the amount of computation involved in computing priorities. For static policies (i.e., policies where an operator priority is constant over time), priorities are computed only once when a query is registered in the DSMS which naturally leads to a low-overhead implementation. Examples of such static policies include *HR*, *HNR*, and *Chain*. On the other hand, for dynamic policies where priority is a function of time, the priority of each operator should be re-computed at each instant of time. Such a naive implementation renders that class of policies very impractical. This motivated several approximation methods for efficient implementation of dynamic policies to balance the trade-off between scheduling overhead and accuracy. For instance the work in [9] proposes using bucketing as well as pre-computation for an efficient implementation of the QoS-aware scheduling in Aurora. Similarly, [14] proposes using search space reduction and pruning methods in addition to clustered processing of continuous queries.

Key Applications

There is a plethora of applications that require data stream management systems and, as such, proper scheduling strategies. The most well-known class of applications is that of *monitoring applications*[8], be it environmental monitoring (e.g., via sensor networks), network monitoring (e.g., by collecting router data), or even financial monitoring (e.g., by observing stock-market data). In all such cases, the sheer amount of input data precipitates the use of the data stream processing paradigm and proper scheduling strategies.

Cross-references

- ▶ Adaptive Query Processing
- ▶ Adaptive Stream Processing
- ▶ Data Stream
- ▶ Event Stream
- ▶ Stream Processing
- ▶ Stream-Oriented Query Languages and Operators
- ▶ Streaming Applications

Recommended Reading

1. Acharya S. and Muthukrishnan S. Scheduling on-demand broadcasts: New metrics and algorithms. In Proc. 4th Annual Int. Conf. on Mobile Computing and Networking, 1998.
2. Babcock B., Babu S., Datar M., and Motwani R. Chain: operator scheduling for memory minimization in data stream systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
3. Babcock B., Babu S., Datar M., Motwani R., and Thomas D. Operator scheduling in data stream systems. VLDB J., 13(4), 2004.
4. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and Issues in Data Stream Systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002.
5. Babu S. and Widom J. Continuous queries over data streams. ACM SIGMOD Rec., 2001.
6. Bansal N. and Pruhs K. Server scheduling in the L_p norm: a rising tide lifts all boats. In Proc. 35th Annual ACM Symp. on Theory of Computing, 2003.
7. Bender M.A., Chakrabarti S., and Muthukrishnan S. Flow and stretch metrics for scheduling continuous job streams. In Proc. 9th Annual ACM -SIAM Symp. on Discrete Algorithms, 1998.
8. Carney D., Cetintemel U., Cherniack M., Convey C., Lee S., Seidman G., Stonebraker M., Tatbul N., and Zdonik S. Monitoring streams: a new class of data management applications. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
9. Carney D., Cetintemel U., Rasin A., Zdonik S., Cherniack M., and Stonebraker M. Operator scheduling in a data stream manager. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003.
10. Chandrasekaran S., Cooper O., Deshpande A., Franklin M.J., Hellerstein J.M., Hong W., Krishnamurthy S., Madden S., Raman V., Reiss F., and Shah M.A. TelegraphCQ: continuous dataflow processing for an uncertain world. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
11. Golab L. and Özsu M.T. Issues in data stream management. ACM SIGMOD Rec., 32(2):5–14, 2003.
12. Mehta M. and DeWitt D.J. Dynamic memory allocation for multiple-query workloads. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993.
13. Muthukrishnan S., Rajaraman R., Shaheen A., and Gehrke J.E. Online Scheduling to Minimize Average Stretch. In Proc. 40th Annual Symp. on Foundations of Computer Science, 1999.
14. Sharaf M.A., Chrysanthis P.K., Labrinidis A., and Pruhs K. Efficient Scheduling of Heterogeneous Continuous Queries. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.
15. Sharaf M.A., Labrinidis A., Chrysanthis P.K., and Pruhs K. Freshness-Aware Scheduling of Continuous Queries in the Dynamic Web. In Proc. 8th Int. Workshop on the World Wide Web and Database, 2005.
16. Sutherland T., Pielech B., Zhu Y., Ding L., and Rundensteiner E. A. An adaptive multi-objective scheduling selection framework for continuous query processing. In Proc. Int. Database Engineering and Applications Symp, 2005.
17. Urhan T. and Franklin M.J. Dynamic pipeline scheduling for Improving Interactive Query Performance. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.

Schema Evolution

John F. Roddick
Flinders University, Adelaide, SA, Australia

Definition

Schema evolution deals with the need to retain current data when database schema changes are performed. Formally, *Schema Evolution* is accommodated when a database system facilitates database schema modification without the loss of existing data, (q.v. the stronger concept of *Schema Versioning*) (Schema evolution and schema versioning has been conflated in the literature with the two terms occasionally being used interchangeably. Readers are thus also encouraged to read also the entry for *Schema Versioning*.).

Historical Background

Since schemata change and/or multiple schemata are often required, there is a need to ensure that extant data either stays consistent with the revised schema or is explicitly deleted as part of the change process. A database that supports schema evolution supports this transformation process.

The first schema evolutioning proposals discussed database conversion primarily in terms of a set of transformations from one schema to another [10]. These transformations focused on the relational structure of the database and included name changing, changing the membership of keys, composing and decomposing relations both vertically and horizontally and so on. In all cases only one schema remained and all data (that still remained) was coerced (ie. copied from one type to another) to the new structure.

Schema evolution has also been covered in the proposals to manage issues such as data coercion [5,12], authority control [2] and query language support [9].

Foundations

Schema evolution is related to the view-update problem, discussed in-depth when the relational model was introduced [1], and is strongly linked to the notion of information capacity [4,7]. Specifically, non-loss evolution can only be guaranteed when the information capacity of the new schema exceeds that of the existing schema. Formally, if $I(S)$ is the set of all valid instances of S, then for non-loss evolution $I(S_{new}) \supseteq I(S_{old})$. One novel solution is the integration of schema evolution with the database view facilities. When new requirements demand schema updates for a particular user, then the user specifies schema changes to a personal view, rather than to the shared base schema [8].

Once a schema change is accepted, the common procedure is for the underlying instances to be coerced to the new structure. Since the old schema is obsolete, this presents few problems and is conceptually simple. However, results in an inability to reverse schema amendments. Schema versioning support provides two other options (q.v.).

Four classes of schema evolution can be envisaged. Each type brings different problems.

1. *Attribute Evolution* occurs when attributes are added to, deleted from, or renamed in a relation. Issues here include the values to be ascribed to attributes in tuples stored under a new version that does not possess the attribute.
2. *Domain Evolution* occurs when the domain over which an attribute is defined is altered. Issues here include implying accuracy that does not exist in existing data when, for example, attributes defined as integers are converted to reals, and in truncation when character fields are shortened.
3. *Relation Evolution* occurs when the relational structure is altered through the definition, deletion, decomposition or merging of a relation. Such changes are almost always irreversible.
4. *Key Evolution* occurs when the structure of a primary key is altered or when foreign keys are added or removed. The issues here can be quite complex. For example, removing an attribute from a primary key may not violate the primary key uniqueness constraint for current data (the amendment can be rejected if it does) but in a temporal database may still do so for historical information.

Note that one change may involve more than one type of evolution, such as changing the domain of a key attribute.

These changes may also be reflected in the conceptual model of the system. For example, the addition of an entity in an EER diagram would result in the addition of a relation in the underlying relational model; deleting a 1-to-many relationship would remove a foreign key constraint, and so on.

Key Applications

Schema changes are linked to either error correction or design change. It is therefore useful if the design decisions can be consulted and the users can interact with schema changes at a high level. One way is to propagate requirements changes to database schemas [3] or provide better support for metadata management by providing a higher level view in which models can be mapped to each other [6].

In order to quantify the types of schema evolution, Sjøberg [11] investigated change to a database system over 18 months, covering 6 months of development and 12 months of field trials. A more recent study complements this by following the changes in an established database system over many years [13].

Future Directions

The major directions for schema versioning research have moved from low-level handling of syntactic elemental changes (such as adding an attribute or demoting an index attribute) to more model-directed semantic handling of change (such as propagating changes in a conceptual model to a database schema) [3]. Research has also moved from schema evolution to the more complex problem of providing versions of schema.

Cross-references

- ▶ Conceptual Modeling
- ▶ Schema Versioning
- ▶ Temporal Algebras
- ▶ Temporal Query Languages

Recommended Reading

1. Bancilhon F. and Spyratos N. Update semantics of relational views. ACM Trans. Database Syst., 6(4):557–575, 1981.
2. Bretl R., Maier D., Otis A., Penney J., Schuchardt B., Stein J., Williams E.H., and Williams M. The GemStone data management system. In Object-Oriented Concepts, Databases and Applications. W. Kim and F. Lochovsky (eds.). ACM, New York, NY, USA, 1989, pp. 283–308.
3. Hick J.M. and Hainaut J.L. Database application evolution: a transformational approach. Data Knowl. Eng., 59(3): 534–558, 2006.
4. Hull R. Relative information capacity of simple relational database schemata. Soc. Ind. Appl. Math., 15(3):856–886, 1986.
5. Kim W. and Chou H.T. Versions of schema for object-oriented databases. In Proc. 24th Int. Conf. on Very Large Data Bases, 1988, pp. 148–159.
6. Melnik S., Rahm E., and Bernstein P.A. Rondo: a programming platform for generic model management. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 193–204.
7. Miller R., Ioannidis Y., and Ramakrishnan R. The use of information capacity in schema integration and translation. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 120–133.
8. Ra Y.G. and Rundensteiner E.A. A transparent schema-evolution system based on object-oriented viewtechnology. IEEE Trans. Knowl. Data Eng., 9(4):600–624, 1997.
9. Roddick J.F. SQL/SE – a query language extension for databases supporting schema evolution. ACM SIGMOD Rec., 21(3):10–16, 1992.
10. Shneiderman B. and Thomas G. An architecture for automatic relational database system conversion. ACM Trans. Database Syst., 7(2):235–257, 1982.
11. Sjøberg D. Quantifying schema evolution. Inf. Softw. Technol., 35(1):35–44, 1993.
12. Tan L. and Katayama T. Meta operations for type management in object-oriented databases – a lazy mechanism for schema evolution. In Proc. First Int. Conf. on Deductive and Object-Oriented Databases, 1989, pp. 241–258.
13. de Vries D. and Roddick J.F. The case for mesodata: an empirical investigation of an evolving database system. Inf. Softw. Technol., 49(9–10):1061–1072, 2007.

Schema Evolution in Process Management Systems

▶ Workflow Evolution

Schema Evolution in Workflow Management Systems

▶ Workflow Evolution

Schema Mapping

Ariel Fuxman[1], Renée J. Miller[2]
[1]Microsoft Research, Mountain View, CA, USA
[2]University of Toronto, Toronto, ON, Canada

Synonyms

Mapping

Definition

The problem of establishing associations between data structured under different schemas is at the core of many data integration and data sharing tasks. *Schema mappings* establish semantic connections between schemas. Given a source schema **S** and a target schema **T**, a *schema mapping* $\mathcal{M}$ is a specification of a relation between instances of **S** and instances of **T**. Given an instance of the source I and an instance of the target J that satisfy the mapping, say that $(I, J) \models \mathcal{M}$. Research on schema mapping has focused on the formal specification of schema mappings, the semantics of mappings, along with techniques for creating schema mappings.

Historical Background

Schema mappings have been developed primarily to solve two different problems, each of which has led to a substantial body of research: *data integration* [11] and *data exchange* [4]. In both problems, one is given a source schema **S** (or a set of source schemas) and an instance I of **S**, along with a target schema **T**, which is sometimes called a global schema. A user knows the schema of the target and would like to retrieve source data by posing queries on the target. In data integration, queries posed on the global schema are translated, at query time, to the schema(s) of the local data source (s) and answered using source data. Data integration is sometimes called *virtual data integration* to emphasize that the translated source data are not materialized in the target. In contrast, in data exchange the goal is to translate the source data into a target instance that conforms to the target schema and reflects the source data as accurately as possible. Target queries are then answered using the materialized target instance.

For both problems, schema mappings are used to describe the semantic relationship between the schemas and their instances, and to determine how queries are translated (in data integration) and what is the best target instance to materialize (in data exchange).

Foundations

Semantics of Schema Mappings

Schema mappings establish semantic connections between schemas. These connections can be represented formally using logical formulas. Assume the existence of two schemas, called the source schema **S**, and the target schema **T**. The specifications of **S** and **T** may include a set of constraints that instances of the schema must satisfy. A *schema mapping* $\mathcal{M}$ is a specification of a relation between instances of **S** and instances of **T**. Given an instance of the source I and an instance of the target J that satisfy the mapping, one could say that $(I, J) \models \mathcal{M}$. A *mapping setting* is a triple $\mathcal{S} = \langle \mathbf{S}, \mathbf{T}, \mathcal{M} \rangle$. In order to give semantics to a mapping setting, it is assumed that an instance of the source is given, and the goal is to reason about the instances of the target that satisfy the constraints imposed by the schema mapping. That is, given an I, are would like to reason about all J such that $(I, J) \models \mathcal{M}$. These target instances are called *solutions*. (This terminology comes from the data exchange literature, and was introduced by Fagin et al. [4]. However, the same concepts are equally applicable to data integration, where the target schema may be referred to as the global or mediated schema.).

Definition

[solution] Let $\mathcal{S} = \langle \mathbf{S}, \mathbf{T}, \mathcal{M} \rangle$ be a mapping setting. Let I be an instance over the source schema **S**. One could say that an instance J over the target schema **T** is a *solution* for I in $\mathcal{S}$ if $\langle I, J \rangle \models \mathcal{M}$.

A mapping may be a function which defines a single target instance J for each source instance I. If **T** consists of a single relation, then any view over **S** defines such a function. Alternatively, in many industrial mapping tools, a mapping is a program which, given an instance of **S**, outputs an instance of **T**. But in general, a mapping does not need to be a function and there are clear advantages in having a declarative specification language for mappings. Declarative mapping specifications permit easier reasoning about the relationship between mappings, something that is essential in designing, maintaining, and evolving mappings.

The most commonly used specification for schema mappings are source-to-target tuple-generating-dependencies (TGDs) which have the form

$$\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})),$$

where $\phi_{\mathbf{S}}(\mathbf{x})$ is a conjunction of atomic formulas over **S** and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over **T**. TGDs have been generalized in some approaches to permit more general source ($\phi_{\mathbf{S}}$) and target ($\psi_{\mathbf{T}}$) queries (formulas) in the mapping.

Since a solution is any target instance that satisfies the mapping, there may be more than one solution for a given source instance. This fact must be accounted for in the semantics of query answering. The prevalent semantics adopted in the literature is based on the notion of *certain answers*. This semantics takes the conservative approach of returning only the answers that are valid in *every* solution.

Definition

[certain answer] Let $\mathcal{S}$ be a mapping setting. Let I be a source instance such that there exists some solution for I in $\mathcal{S}$. Let q be a query. We one could that a tuple **t** is a *certain answer* to q in $\mathcal{S}$, denoted $\mathbf{t} \in \mathbf{certain}(q, I, \mathcal{S})$, if for every solution J for I in $\mathcal{S}$, it is the case that $\mathbf{t} \in q(J)$.

In addition to certain answers, other semantics have been explored in the literature, such as epistemic interpretations and probabilistic notions.

To illustrate the notions introduced so far, let **S** be a schema with relation symbol `Country(person,country)`. Let **T** be a schema with relation symbols `Home(person,city)` and `Loc(city, country)`. As an example of a mapping setting, let $\mathcal{S} = \langle \mathbf{S}, \mathbf{T}, \mathcal{M} \rangle$, where $\mathcal{M}$ consists of the following source-to-target TGD:

$$\forall p, cou.(\mathbf{S}.Country(p, cou) \rightarrow \\ \exists cit.\mathbf{T}.Home(p, cit) \wedge \mathbf{T}.Loc(cit, cou))$$

There may be more than one solution for a given instance I. For example, let I ={*Country*(*john*, *canada*)}. Consider J_1 = {*Home*(*john*, *toronto*), *Loc* (*toronto*, *canada*)} and J_2 = {*Home*(*john*, *montreal*), *Loc*(*montreal*, *canada*)}. It is easy to see that both J_1 and J_2 are solutions for I in $\mathcal{S}$. The reason for this is that the mapping states that the people and countries of the source must be in the target, but the city is left unspecified.

Now, consider a query q_1 that retrieves all people from the database. Let $q_1(p) = \exists cit : \mathbf{T}.Home(p,cit)$. Since $\mathcal{M}$ must be satisfied by all solutions, there are tuples *Home*(*john*,*c*) and *Loc*(*c*,*canada*) for some city *c*, in every solution for *I*. Thus, $(john) \in q_1(J)$, for every solution *J*. Say that (*john*) is a certain answer to q_1, and denote this by $(john) \in \mathbf{certain}(q_1, I, \mathcal{S})$. Next, consider a query q_2 that returns all cities. Let $q_2(cit) = \exists p : \mathbf{T}.Home(p,cit)$. In this case, there are no certain answers to q_2. To see why, notice that there is no tuple $\mathbf{t}$ such that $\mathbf{t} \in q_2(J_1) \cap q_2(J_2)$. The intuition is that John is the only person in the database, but different solutions may assign him a different city (as long as it is within Canada).

Types of Schema Mappings

Sound, Complete, and Exact Mappings

A common assumption in the data integration and data exchange literature is that the mappings consist of implications, where each side of the implication contains relation symbols coming from the same schema. This results in the following three types of mappings [6,11]. Let ϕ_s be a formula over the source schema, and ψ_t be a formula over the target schema. *Sound mappings* are rules of the form $\phi_s(\mathbf{x}) \rightarrow \psi_t(\mathbf{x})$; *complete mappings* are of the form $\psi_t(\mathbf{x}) \rightarrow \phi_s(\mathbf{x})$; and *exact mappings* are of the form $\phi_s(\mathbf{x}) \leftrightarrow \psi_t(\mathbf{x})$. (Variable quantifiers are omitted for generality.)

There is a substantial body of work on sound mappings. Such systems are sometimes called *open* because the mapping specifies what source data *must* be in the solutions, but it does not give negative information (i.e., it does not specify what *must not* be in any solution). A setting $\mathcal{S}$ containing only sound mappings is referred to as an *open mapping setting*. If *J* is a solution for an instance *I* in an open setting $\mathcal{S}$, and J' is such that $J \subseteq J'$, then J' is a solution for *I* in $\mathcal{S}$. As an example, consider an open setting with the following sound mapping.

$$\forall p, cou.\mathbf{S}.Country(p, cou) \rightarrow \\ \exists cit.\mathbf{T}.Home(p, cit) \wedge \mathbf{T}.Loc(cit, cou)$$

Let I = {*Country*(*john*,*canada*)}. Let J = {*Loc*(*calgary*, *canada*), *Home*(*john*,*ottawa*), *Loc*(*ottawa*,*canada*)}, which is a solution for *I* in $\mathcal{S}$. The tuple *Loc*(*calgary*, *canada*) does not seem to be related to the sources, and the mapping does not *force* its addition to the solution. However, it does not forbid its inclusion in the solution either. In fact, one could add any arbitrary tuple to *J*, and still have a solution for *I*.

Research on data integration and data exchange has focused primarily on open settings. One reason for this is that, given a source instance, open settings always have a solution. From a practical standpoint, open settings are better suited than settings containing exact or complete mappings in dynamic or autonomous environments, where new sources may be added independently of other sources. With open settings, sources can be described without requiring any knowledge of the other sources, or their relationship to the target. In particular, consider the problem of adding a new data source to an existing data integration (or exchange) system. With open settings, it is not necessary to change any of the existing mappings. A mapping for a new data source cannot conflict with existing mappings.

In contrast, in mapping settings containing complete or exact rules, the addition of a new source may lead to conflicts resulting in a setting for which there may be no solutions. In fact, even for a single setting the use of complete or exact mappings may sometimes preclude the existence of a solution. The problem of deciding the existence of a solution for mapping settings has been studied by Fuxman et al. [5]. As an example of a case in which there may be no solution, consider a setting $\mathcal{S} = \langle \mathbf{S}, \mathbf{T}, \mathcal{M} \rangle$, where $\mathcal{M}$ consists of the following rules.

$$\forall p, cou.\mathbf{S}.Country(p, cou) \rightarrow \\ \exists cit.\mathbf{T}.Home(p, cit) \wedge \mathbf{T}.Loc(cit, cou) \qquad (1)$$

$$\forall p, cit, cou.\mathbf{T}.Home(p, cit) \wedge \mathbf{T}.Loc(cit, cou) \rightarrow \\ \mathbf{S}.Capital(cit, cou) \qquad (2)$$

For the source instance I = {*Country*(*john*, *canada*), *Capital*(*washington*,*us*)}, there is no solution in $\mathcal{S}$. To see this, assume that there is a solution *J* for *I* in $\mathcal{S}$. By rule (1) of $\mathcal{M}$, *J* has tuples *Home*(*john*, *c*) and *Loc*(*c*, *canada*), for some city *c*. By rule (2) of $\mathcal{M}$, *I* is required to have a tuple *Capital*(*canada*, *c*) which it does not. Intuitively, there is no solution for *I* in $\mathcal{S}$ since the source has no information on what city is the capital of Canada. Notice the effect of rules (1) and (2) on the solutions. The former is a sound mapping and specifies what *must* be in the solutions; the latter is a complete mapping and constrains what can be in the solutions.

Global-as-View and Local-as-View

In many data integration systems, each relation (or element) of the target schema is defined in terms of the source schemas. In many early systems, there was also an implicit assumption that the schema mapping was a function (for example a view). This approach is known as *global-as-view*. An alternative approach, known as *local-as-view*, was later proposed in which each relation of the source schema is defined in terms of the target schema. For relational systems, these notions can be defined formally as follows [11].

- In *global-as-view* systems (GAV), mappings are of the form $\forall \mathbf{x}.\phi_s(\mathbf{x}) \leftrightarrow R_t(\mathbf{x})$, where R_t is a relation symbol from $\mathbf{T}$ and $\phi_s(\mathbf{x})$ is a formula over $\mathbf{S}$ (i.e., rather than a formula, there is just one atom on the right-hand-sidewith no repeated variables). (Note that GAV mappings as originally defined, were typically exact mappings, though more recently sound GAV mappings have also been studied [11].)
- In *local-as-view* systems (LAV), mappings are of the form $\forall \mathbf{x}.R_s(\mathbf{x}) \rightarrow \psi_t(\mathbf{x})$, where R_s is a relation symbol from $\mathbf{S}$ and $\psi_t(\mathbf{x})$ is a formula over $\mathbf{T}$ (i.e., rather than a formula, there is just one atom on the left-hand-side with no repeated variables).

Recall that queries are posed in terms of the target (global) schema. For this reason, query answering in GAV is easy: the mapping indicates explicitly how to retrieve data from the sources. In particular, given a query q over the target schema, it suffices to *unfold* q using the mapping in order to obtain a rewriting q' of q (i.e., a query that computes the certain answers to q). As an example, consider the following GAV setting $\mathcal{S} = \langle \mathbf{S}, \mathbf{T}, \mathcal{M} \rangle$, where $\mathcal{M}$ consists of the following rule.

$$\forall cit, cou \exists p.\mathbf{S}.Capital(cit, cou) \wedge \\ \mathbf{S}.Country(p, cou) \leftrightarrow \mathbf{T}.Loc(cit, cou)$$

Suppose that a user wants to retrieve the cities and countries from relation Loc. Thus, she issues the following query over the global schema:

$$q(cit, cou) = \mathbf{T}.Loc(cit, cou)$$

In order to obtain a rewriting of q, it suffices to replace $Loc(cit, cou)$ by its definition in $\mathcal{M}$:

$$q'(cit, cou) = \exists p.\mathbf{S}.Capital(cit, cou) \wedge \\ \mathbf{S}.Country(p, cou)$$

Query processing in LAV is more involved than in GAV. The reason is that queries are written over the target schema, but a LAV mapping associates views to relations of the source schemas. Thus, the unfolding strategy no longer works in this case; and it is not immediate how to rewrite queries over the source schema. In general, the complexity of query answering in LAV is higher than in GAV. The intuitive explanation is that in GAV, given a source instance *I*, it suffices to concentrate on a single solution *J*, whereas in LAV there may be many such solutions.

Rather than unfolding, the problem of reformulating a target query using a LAV mapping boils down to the problem of *answering queries using views* [12].

For many classes of open schema mappings (including source-to-target TGDs) query answering has tractable *data complexity*. For example, query answering is tractable when the schema mappings and queries are conjunctive. In contrast, if complete or exact rules are allowed, the same problem becomes coNP-complete. To show the jump in complexity, consider Figs. 1 and 2 (due to Abiteboul and Duschka [1]). The former gives results for open LAV mappings. The latter gives results under the "closed world assumption", where mappings consist of exact LAV mappings of the form $\forall \mathbf{x} : R_s(\mathbf{x}) \leftrightarrow \psi_t(\mathbf{x})$, where $R_s(\mathbf{x})$ is a relation from the source, and $\psi_t(\mathbf{x})$ is a formula over the target. The results are given for different logical languages used for the mappings and queries: conjunctive queries (CQ), conjunctive queries with inequalities ($CQ^{\neq}$), union of conjunctive queries (UCQ), Datalog, and first-order logic (FO). In addition to these results, the problem of query answering in open mapping settings has also been studied for other query languages (e.g., description logics) and on other data models (e.g., semi-structured models).

mapping			query		
	CQ	$CQ^{\neq}$	UCQ	Datalog	FO
CQ	PTIME	coNP	PTIME	PTIME	undec.
$CQ^{\neq}$	PTIME	coNP	PTIME	PTIME	undec.
UCQ	coNP	coNP	coNP	coNP	undec.
Datalog	coNP	undec.	coNP	undec.	undec.
FO	undec.	undec.	undec.	undec.	undec.

Schema Mapping. Figure 1. Complexity of query answering using open LAV mappings.

mapping			query		
	CQ	$CQ^{\neq}$	UCQ	Datalog	FO
CQ	coNP	coNP	coNP	coNP	undec.
$CQ^{\neq}$	coNP	coNP	coNP	coNP	undec.
UCQ	coNP	coNP	coNP	coNP	undec.
Datalog	undec.	undec.	undec.	undec.	undec.
FO	undec.	undec.	undec.	undec.	undec.

Schema Mapping. Figure 2. Complexity of query answering using exact LAV mappings.

Schema Mappings in Peer Data Sharing

The success of peer-to-peer (P2P) technology in the domain of file exchange motivated the research community to consider peer-to-peer architectures for data sharing. In a P2P system, participants (peers) rely on one another for service, blurring the distinction between clients and servers, source and target. P2P systems are founded on the principles of *peer autonomy* and *decentralized coordination*. As a result, peers do not have a global view of the system. Rather, global behavior emerges from local interactions.

In a Peer Data Management System (PDMS), each peer has a schema that describes the structure of its data, and can establish connections (typically specified as schema mappings) with other peers in order to exchange data [2,7]. A PDMS is expected to satisfy the desirable properties of P2P systems. For example, the requirement of decentralized coordination precludes the existence of a central catalog. Rather, knowledge about schemas and mappings should be distributed among the peers.

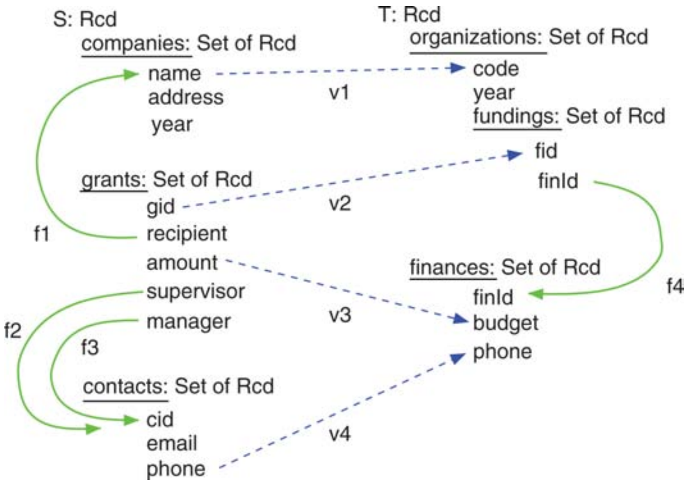
A PDMS should support an arbitrary network of mappings among peers. More importantly, it should be able to exploit the transitive relationships of the network during query answering. A PDMS is essentially a directed graph whose nodes are individual mapping settings, and whose arcs correspond to mappings that relate the schemas in the network. More precisely, there is an arc from $\mathbf{P}_1$ to $\mathbf{P}_2$ if there is a sound rule in some peer setting $\langle \mathbf{P}_1, \mathbf{P}_2, \mathcal{M} \rangle$ or a complete rule in some peer setting $\langle \mathbf{P}_2, \mathbf{P}_1, \mathcal{M} \rangle$. It turns out that the topology of this graph has a direct impact on query answering. In particular, Halevy et al. [7] showed the undecidability of the problem of obtaining the certain answers for a PDMS of arbitrary topology, where conjunctions are used for the mapping rules and the queries. Contrast this to the case of a single mapping setting with exact mappings, which is not undecidable, but coNP-complete and tractable for open settings. Fuxman et al. [5] studied this problem for a special case of PDMS, called Peer Data Exchange, and gave a class of mappings and queries for which the problem is decidable under the existence of cycles. Calvanese et al. [3] proposed an alternative semantics for query answering, based on epistemic interpretations, for which obtaining the certain answers in a PDMS of arbitrary topology is decidable and, in some cases, tractable. An alternative approach involves specifying schema mappings using *mapping tables* which specify how data values are mapped between peers [10]. Research is on-going on what are good mapping formalisms to support PDMS.

Creating Schema Mappings

Creating a schema mapping between independently designed schemas can be a tremendous challenge. Schemas that are designed independently, even if they represent the same or similar information may use different names and structures to describe the same or similar data. Designing schema mappings by hand is known to be a very difficult task requiring expert users familiar with both source and target schemas. Even experts can often make errors leading to specifications that omit information or produce incorrect answers to target queries.

To help automate this task, Milo and Zohar proposed the use of *schema matchings* (or matchings) which indicate potential associations between elements within different schemas [14]. A matching is most often represented as a set of pairs of schema attributes from two different schemas. Schema matchings can be semi-automatically inferred by using a variety of matching tools. These tools use schema and data characteristics such as lexical similarities, structural proximity, data values, etc. to infer potential matches between attributes of different schemas. Matchings, however, do not represent the full semantic relationship between schemas and their instances.

Figure 3 shows two schemas that represent information about companies and grants. The leftmost schema (the source **S**) is a relational schema (with three tables `companies`, `grants`, and `contacts`), presented in a nested relational representation that is used as a common platform for modeling relational and XML schemas. The curved lines $f1$, $f2$ and $f3$ in the figure represent either foreign keys or simple inclusion dependencies, specified as part of the schema (or discovered using a dependency or constraint miner).



Schema Mapping. Figure 3. A matching between source and target schemas.

The schema on the right (the target **T**), records the funding (`fundings`) that an organization (`organizations`) receives, nested within the respective organization element. The amount of each funding is recorded in the `finances` record along with a contact phone number (`phone`).

Figure 3 indicates, using the dotted lines v_1 through v_4, a matching entered by a schema expert or discovered by a matching tool. Due to the heterogeneity and the different requirements under which the two databases were developed, the same real world entity (for example, a company 'IBM') may be represented in very different ways in the two databases, and structures that appear to be the same may actually model different concepts. In our example, the matching v_1 indicates that what is called a company `name` in the first schema, is referred to as an organization `code` in the second. On the other hand, both schemas have an element `year`, but there is no match between these attributes indicating that they likely do not represent the same concept. For instance, element `year` in the source schema may represent the time the company was founded, while in the target it may represent the time the company had its initial public offer.

Note that matchings are far from sufficient to tell us how the data instances of **S** and **T** are related. While the matching may indicate that `companies.name` data should appear in the `organizations.code` attribute of the target and that `grants.gid` data should appear in `fundings.fid`, it does not tell us which grant should be associated with which company. Similarly, if one relies solely on the matching, one could map `grants.gid` to `fundings.fid` and leave the `finId` attribute value empty (since there is no matching for `finId`). If one does this, in the target data, there will be no way to associate a funding record with a finance record. However, the foreign key f_4 indicates there is a real world relationship between these concepts.

The generation of schema mappings have been considered in a number of research projects and industrial tools. The Clio project [13] was the first mapping system to exploit logical reasoning about the semantics embedded in the schemas and their instances to help automate mapping creation. In this work, the mapping discovery process has been referred to as *query discovery* in that the goal is to discovery a query over the source ($\phi_\mathbf{S}$), a query over the target ($\psi_\mathbf{T}$), and their relationship, in order to create a set of possible source-to-target TGDs: $\forall \mathbf{x}(\phi_\mathbf{S}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_\mathbf{T}(\mathbf{x}, \mathbf{y}))$ [15]. These mappings can then be shown to a mapping designer (visually or using data examples) who can decide if they correctly represent the relationship between source and target instances.

To illustrate Clio's approach, consider our example schemas. In the target, the nesting structure within `organizations` indicates that there is a real-world relationship between organizations and their fundings– that is, the association of a specific funding record with a specific organization has some natural semantics in the domain. For example, the nesting may represent fundings given to (or alternatively given by) an organization. Hence, in creating a mapping, Clio will consider related associations in the source. In the example, the data that matches organizations and fundings comes from companies and grants (via the matches v_1 and v_2). There is a relationship between these two tables represented by the inclusion dependency on grants.recipient. Hence, by ignoring v_3 and v_4 for the moment, Clio will suggest the following source-to-target TGD to a user. (Notation is slightly abused and let $\underline{F}$ represent identifiers for sets nested inside of organizations.)

$$\begin{aligned}&\forall n, \forall d, \forall y, \forall g, \forall a, \forall s, \forall m\\ &companies(n, d, y), grants(g, n, a, s, m) \;\rightarrow\\ &\exists y', \underline{F}, f, organizations(n, y', \underline{F}), \underline{F}(g, f)\end{aligned} \tag{3}$$

Notice that this mapping retains the semantic association between companies and their grants, and uses this to associate organizations with a set of related fundings. This mapping is correct if these two associations represent the same real world association, something a user must verify. By extending this example to include v_3, Clio will see that there is an association between gid and amount in the source (because these values are paired in the same record) and consider possible ways of associating the matched attributes (in this case fid and budget) in the target. These attributes are not in the same record in the target, but are associated through a foreign key on finId. To maintain the source association in the target, Clio creates a mapping containing a target join on finId. Finally, if one considers how to create finances tuples in the target, notice that there are two possible ways of associated the related source data (grants.amount and contacts.phone) – using a join on supervisor and cid, or using a join on manager and cid. These joins represent different semantic associations and Clio will create mappings corresponding to each and let the user decide which (if any) of these associations should be preserved in the target data. One of the mappings created by Clio, which uses the source association represented by f_2, is illustrated below.

$$\begin{aligned}&\forall n, \forall d, \forall y, \forall g, \forall a, \forall s, \forall m, \forall e, \forall p\; companies(n, d, y)\\ &\wedge grants(g, n, a, s, m) \wedge contacts(s, e, p) \;\rightarrow\\ &\exists y', \underline{F}, f, organizations(n, y', \underline{F}) \wedge \underline{F}(g, f) \wedge\\ &finances(f, a, p)\end{aligned} \tag{4}$$

Clio's mapping discovery algorithm is based on an extension of standard relational dependency inference (based on the chase) to nested relational schemas. The schemas may be relational or nested relational containing source and target TGDs (e.g., inclusion dependencies) and egds (e.g., functional dependencies).

Data Translation

Mapping tools that create declarative mappings provide a way of translating these specifications into programs (transformation code) that given a source instance produce a single target instance for data exchange [15,16], Industrial mapping systems, such as Altova Mapforce (http://www.altova.com/products/mapforce/data_mapping.html), Stylus Studio (http://www.stylusstudio.com), or Aqualogic (http://www.bea.com/aqualogic) are often visual programming systems which compile visual specifications of mappings into executable code including SQL, XSLT, Java or C.

Within this area, there has been a great deal of work on producing data transformation code that is modular and efficient [9,16]. For schema mappings that permit many solutions, a decision must be made as to what is the "best" solution to materialize. Notice that there may be target data (for example, organizations.year, or fundings.finId attributes from Fig. 3) that do not correspond to any source data. It may not be sufficient to simply fill in null values for this information. Consider the fundings.finId and finances.finId from Fig. 3. If one fills both with null values, it is possible to join fundings with finances to find the budget of a specific funding. As an alternative, to maintain the association between the source values grants.gid and grants.amount as this data are translated into the target, one option is to create identifiers (using Skolem functions) that represent the desired association [8]. The Clio project was the first to consider systematically how to create Skolem functions that fill in missing target data specifically for data exchange [15].

Key Applications

Schema mappings are foundational to enabling data integration, data exchange, schema evolution, and data translation (between data models). Applications of schema mappings include Enterprise Information Integration (EII), e-commerce, object-to-relational wrappers, XML-to-relational mapping, data warehousing, and portal design tools.

Cross-references

- ► Answering Queries Using Views
- ► Certain Answers
- ► Data Exchange

- ▶ Peer Data Management System
- ▶ Peer-to-Peer Data Integration
- ▶ Schema Mapping Composition
- ▶ Schema Matching

Recommended Reading

1. Abiteboul S. and Duschka O.M. Complexity of answering queries using materialized views. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 254–263.
2. Bernstein P.A., Giunchiglia F., Kementsietsidis A., Mylopoulos J., Serafini L., and Zaihrayeu I. Data management for peer-to-peer computing: a vision. In Proc. 5th Int. Workshop on the World Wide Web and Databases, 2002.
3. Calvanese D., De Giacomo G., Lenzerini M., and Rosati R. Logical foundations of peer-to-peer data integration. In Proc. 23rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2004, pp. 241–251.
4. Fagin R., Kolaitis P.G., Miller R.J., and Popa L. Data exchange: semantics and query answering. Theor. Comput. Sci., 336(1):89–124, May 2005.
5. Fuxman A., Kolaitis P.G., Miller R.J., and Tan W.-C. Peer data exchange. ACM Trans. Database Syst., 31(4):1454–1498, 2006.
6. Grahne G. and Mendelzon A.O. Tableau techniques for querying information sources through global schemas. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 332–347.
7. Halevy A., Ives Z., Suciu D., and Tatarinov I. Schema mediation in peer data management systems. In Proc. 9th Int. Conf. on Data Engineering, 2003, pp. 505–518.
8. Hull R. and Yoshikawa M. ILOG: declarative creation and manipulation of object identifiers. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 455–468.
9. Jiang H., Ho H., Popa L., and Han W.S. Mapping-driven xml transformation. In Proc. 16th Int. World Wide Web Conf., 2007, pp. 1063–1072.
10. Kementsietsidis A., Arenas M., and Miller R.J. Mapping data in peer-to-peer systems: Semantics and Algorithmic Issues. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 325–336.
11. Lenzerini M. Data integration: a theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
12. Levy A.Y., Mendelzon A.O., Sagiv Y., and Srivastava D. Answering queries using views. In Proc. 14th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1995, pp. 95–104.
13. Miller R.J., Haas L.M., and Hernández M. Schema mapping as query discovery. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 77–88.
14. Milo T. and Zohar S. Using schema matching to simplify heterogeneous data translation. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 122–133.
15. Popa L., Velegrakis Y., Miller R.J., Hernández M.A., and Fagin R. Translating web data. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 598–609.
16. Shu N.C., Housel B.C., Taylor R.W., Ghosh S.P., and Lum V.Y. EXPRESS: a data eXtraction, processing, amd restructuring system ACM Trans. Database Syst., 2(2):134–174, 1997.

Schema Mapping Composition

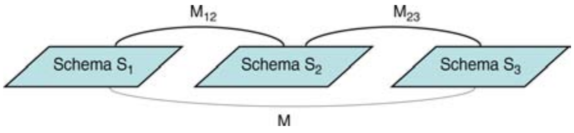
Wang-Chiew Tan
University of California-Santa Cruz, Santa Cruz,
CA, USA

Synonyms

Mapping composition; Semantic mapping composition

Definition

A *schema mapping* (or *mapping*) is a triple $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$, where $\mathbf{S}_1$ and $\mathbf{S}_2$ are relational schemas with no relation symbols in common and Σ is a set of formulas of some logical formalism over $(\mathbf{S}_1, \mathbf{S}_2)$. An *instance* of $\mathcal{M}$ is a pair (I, J) where I is an instance of $\mathbf{S}_1$ and J is an instance of $\mathbf{S}_2$ such that (I, J) satisfies every formula in the set Σ. The set of all instances of $\mathcal{M}$ is denoted as $\text{Inst}(\mathcal{M})$.



Let $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ be two consecutive mappings such that there are no relation symbols in common between any two schemas of $\mathbf{S}_1$, $\mathbf{S}_2$ and $\mathbf{S}_3$. A mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma)$ is a *composition* of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ if $\text{Inst}(\mathcal{M}) = \text{Inst}(\mathcal{M}_{12}) \circ \text{Inst}(\mathcal{M}_{23})$. In other words, $\text{Inst}(\mathcal{M})$ is the set of all pairs (I, J) such that I is an instance of $\mathbf{S}_1$, J is an instance of $\mathbf{S}_3$ and there exists an instance K of $\mathbf{S}_2$ such that $(I, K) \in \text{Inst}(\mathcal{M}_{12})$ and $(K, J) \in \text{Inst}(\mathcal{M}_{23})$.

Historical Background

Mappings are widely used in the specification of relationships between data sources in applications such as data integration, data exchange, and peer data management systems. The model management framework introduced by Bernstein et al. [3], where the primary abstractions are models and mappings between models, can be used to model these applications. Several operators for manipulating mappings between models were introduced in this framework. Among them, the *composition operator* is one of the most fundamental operators for manipulating mappings between models.

Madhavan and Halevy [10] first studied the problem of composing mappings between relational schemas (mappings are also called *semantic mappings* in [10]). They gave a definition of the semantics of the composition operator in the context where mappings are specified by sound *Global-Local-As-View (GLAV)* formulas [9]. Sound GLAV formulas are the most widely used and studied form of mappings in data integration systems and they are equivalent to *source-to-target tuple generating dependencies (s-t tgds)* [13]. (See Foundations section for a definition of s-t tgds.) The Madhavan and Halevy semantics of composition is different from the one stated above; The set of formulas that specifies the composition $\mathcal{M}$ of two successive mappings, $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$, is relative to a class of queries Q that is defined over the schema $\mathbf{S}_3$. This means that for every query q in Q, the certain answers of q according to $\mathcal{M}$ coincide with the certain answers of q that would be obtained by applying the consecutive mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$.

The Madhavan and Halevy notion of composition is termed a composition that is *certain answer adequate for* Q in [8]. Fagin et al. [8] showed that while certain answer adequacy may be sufficient for obtaining the certain answers of any query in Q, whether via Madhavan and Halevy's notion of composition $\mathcal{M}$ or through the successive mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$, there may be formulas that are logically inequivalent to Σ of $\mathcal{M}$ that are also certain answer adequate for Q. In other words, there can be two mappings $\mathcal{M}$ and $\mathcal{M}'$ that are both certain answer adequate for the consecutive mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ with respect to the class of queries Q, but the sets of formulas Σ and Σ' of $\mathcal{M}$ and $\mathcal{M}'$, respectively, are logically inequivalent. It was also shown in [8] that certain answer adequacy is a rather fragile notion. A mapping $\mathcal{M}$ that is a composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ with respect to a class Q of conjunctive queries may no longer be a composition of the same two mappings when Q is extended to the class of conjunctive queries with inequalities.

Fagin et al. [8] introduced a definition of composition that is based entirely on the set-theoretic composition of instances of the two successive mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$. (See the Definition section for the set-theoretic definition given in [8].) Unlike the definition of composition given by Madhavan and Halevy, Fagin et al.'s [8] definition is not relative to a class of queries. Furthermore, the set of formulas in $\mathcal{M}$ that defines the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ is unique up to logical equivalence. Hence, with Fagin et al.'s [8] notion of composition, $\mathcal{M}$ is referred to as *the* composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$. It was also shown in [8] that the composition $\mathcal{M}$ is always certain answer adequate for $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ for *every* class of queries.

The results established in [8] were based on mappings specified by s-t tgds. In other words, Fagin et al. [8] assumes that the sets of formulas Σ_{12} and Σ_{23} in the successive mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$, respectively, are finite sets of s-t tgds. A subsequent paper by Nash et al. [12] also studied the composition operator where mappings are specified by *embedded dependencies*. Embedded dependencies are more general than s-t tgds and can model constraints such as keys. Among the results established in [12] is an algorithm that computes the composition of two successive mappings specified by embedded dependencies. Their algorithm may not terminate in general and the authors characterized sufficient conditions on the input mappings for which their composition algorithm is guaranteed to produce a composition of the input mappings. An implementation of the composition operator that extends the composition algorithm of [12] is described in [2].

Foundations

In [8], mappings are specified by finite sets of s-t tgds which are equivalent to sound GLAV assertions used in [10]. A *s-t tgd* is a first-order formula of the form:

$$\forall \mathbf{X}(\phi_S(\mathbf{X}) \rightarrow \exists \mathbf{y} \psi_T(\mathbf{x}, \mathbf{y})),$$

where $\phi_S(\mathbf{x})$ is a conjunction of atomic formulas over the schema $\mathbf{S}$ and where $\psi_T(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over the schema $\mathbf{T}$. Every variable in $\mathbf{x}$ and $\mathbf{y}$ must appear in ϕ_S and ψ_T respectively. However, some variables in $\mathbf{x}$ need not appear in ψ_T. A *full source-to-target tuple generating dependency (full s-t tgd)* is a special s-t tgd of the form

$$\forall \mathrm{x}(\phi_S(\mathbf{x}) \rightarrow \psi_T(\mathbf{x}),$$

where no existentially-quantified variables appears on the right-hand-side of the s-t tgd. As before, $\phi_S(\mathbf{x})$ is a conjunction of atomic formulas over **S** and $\psi_T(\mathbf{y})$ is a conjunction of atomic formulas over **T**. Every variable in **x** must appear in ϕ_S.

Example 1. Let $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{12})$ be two successive mappings. The schema $\mathbf{S}_1$ consists of a binary relation symbol `Takes`, the second schema $\mathbf{S}_2$ consists of two binary relation symbols $\texttt{Takes}_1$ and `Student`, and the third schema $\mathbf{S}_3$ consists of a single binary relation symbol `Enrollment`. The formulas in Σ_{12} and Σ_{23} are s-t tgds stated below:

$$\sum{}_{12} = \{\forall_n \forall_c (\textbf{Takes}(n, c) \rightarrow \textbf{Takes}_1(n, c)),$$
$$\forall_n \forall_c (\textbf{Takes}(n, c) \rightarrow \exists s\ \textbf{Student}(n, s))\}$$

$$\sum{}_{23} = \{\forall_n \forall_s \forall_c (\textbf{Student}(n, s) \wedge \textbf{Takes}_\mathbf{1}(\mathbf{n}, \mathbf{c})),$$
$$\textbf{Enrollment}(s, c)))\}$$

Observe that the first s-t tgd in Σ_{12} and the s-t tgd in Σ_{23} are full s-t tgds. The s-t tgds in Σ_{12} state that the $\textbf{Takes}_1$ relation in $\mathbf{S}_2$ contains the **Takes** relation in $\mathbf{S}_1$ and that every student with name *n* who takes a course *c* (in **Takes**) has a associated tuple in **Student** with name *n* and some student id *s*. The s-t tgd in Σ_{23} states that every student with name *n* and student id *s* (in **Student**) who is also taking a course *c* (in $\textbf{Takes}_1$) must have a corresponding tuple (*s*, *c*) that associates the student id *s* with the course *c* in the **Enrollment** relation of $\mathbf{S}_3$.

Recall from the definition that the composition $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ is a mapping $\mathcal{M}$ that captures exactly the set of instances $\text{Inst}(\mathcal{M}_{12}) \circ \text{Inst}(\mathcal{M}_{23})$. The set of instances $\{(I_1, I_3) \mid (I_1, I_3) \in \text{Inst}(\mathcal{M}_{12}) \circ \text{Inst}(\mathcal{M}_{23})\}$ is called the *composition query* of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$. Among the issues investigated by Fagin et al. [8] in composing mappings under this semantics are:

- Is the language of s-t tgds always sufficient to define the composition of two successive mappings?
- What is the complexity of the instances associated with the composition query of two successive mappings?
- What is a right language for composing mappings?
- How does data exchange and query answering behave in the chosen language for composing mappings?

Composing s-t TGDs: Definability and Complexity. The answer to the first question above is no. It was shown in [8] that if $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ are specified by finite sets of full s-t tgds and s-t tgds respectively, then the composition of $\mathcal{M}_{12}$ with $\mathcal{M}_{23}$ is always definable by a finite set of s-t tgds. Furthermore, the associated composition query is in PTIME. However, if $\mathcal{M}_{12}$ is specified by a set of s-t tgds, not necessarily full, then the composition of $\mathcal{M}_{12}$ with $\mathcal{M}_{23}$ may not always be definable by a finite set of s-t tgds. For instance, the composition of the successive mappings in Example 1 is not definable by any finite set of s-t tgds. However, the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ is definable by an infinite set of s-t tgds and, in fact, definable by a first-order formula. As a consequence, the composition query of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ is a PTIME query.

Fagin et al. [8] further showed that there exists successive mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ where the composition query of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ is NP-complete and the composition of the two mappings is not definable in least fixed-point logic LFP. The mappings used are such that $\mathcal{M}_{12}$ is specified by a finite set of s-t tgds each having at most one existentially-quantified variable and $\mathcal{M}_{23}$ consists of only one full s-t tgd. Essentially, the NP-hardness result is obtained by a reduction from 3-Colorability to the composition query of two fixed mappings with the above properties. They showed that the composition query of mappings specified by finite sets of s-t tgds is always in NP.

Second-Order TGDs. Since the composition query of mappings specified by finite sets of s-t tgds is always in NP, it follows from Fagin's theorem [5] that the composition of the two mappings is always definable by an existential second-order formula, where the existential second-order variables are interpreted over relations on the union of the set of values in I_1 with the set of values in I_3. Here, $(I_1, I_3) \in \text{Inst}(\mathcal{M}_{12}) \circ \text{Inst}(\mathcal{M}_{23})$. Fagin et al. [8] showed that, in fact, the composition of mappings specified by finite sets of s-t tgds is always definable by a restricted form of existential second-order formula, called *second-order tgds* (SO tgds).

SO tgds are s-t tgds that are extended with existentially quantified functions and with equalities. SO tgds are the "right" language for composing mappings because they form the smallest well-behaved extension to the class of s-t tgds that is closed under conjunction and composition. In addition, as explained later, SO tgds also possess good properties for data exchange and query answering. The precise definition of SO tgds is given next, after the definition of *terms*.

Given a collection **x** of variables and a collection **f** of function symbols, a *term (based on* **x** *and* **f***)* is defined recursively as follows: (i) Every variable in $\mathbf{x}$ is a term. (ii) If f is a k-ary function symbol in $\mathbf{f}$ and $t_1,\ldots,t_k$ are terms, then $f(t_1,\ldots,t_k)$ is a term. Let $\mathbf{S}$ be a source schema and $\mathbf{T}$ a target schema. A *second-order tgd (SO tgd)* is a formula of the form:

$$\exists \mathbf{f}((\forall \mathbf{x_1}(\phi_1 \rightarrow \psi_1)) \wedge \ldots \wedge (\forall \mathbf{x_n}(\phi_n \rightarrow \psi_n))),$$

where

1. Each member of $\mathbf{f}$ is a function symbol.
2. Each ϕ_i is a conjunction of
   - atomic formulas of the form $S(y_1,\ldots,y_k)$, where S is a k-ary relation symbol of schema $\mathbf{S}$ and $y_1,\ldots,y_k$ are variables in $\mathbf{x_i}$, not necessarily distinct, and
   - equalities of the form $t = t'$ where t and t' are terms based on $\mathbf{x_i}$ and $\mathbf{f}$.
3. Each ψ_i is a conjunction of atomic formulas $T(t_1,\ldots,t_l)$, where T is an l-ary relation symbol of schema $\mathbf{T}$ and $t_1,\ldots,t_l$ are terms based on $\mathbf{x_i}$ and $\mathbf{f}$.
4. Each variable in $\mathbf{x_i}$ appears in some atomic formula of ϕ_i.

Each subformula $\forall \mathbf{x}_i(\phi_i \rightarrow \psi_i)$ is a *conjunct* of the SO tgd. The last condition is a safety condition similar to that made for s-t tgds. As an example, the formula $\exists f \forall x \forall y(S(x) \wedge (y = f(x)) \rightarrow T(x, y))$ is not a SO tgd because the safety condition is violated. In particular, the variable y does not appear in an atomic formula on the left-hand-side of the formula. As another example, the formula

$$\exists f \forall_n \forall_c(\mathtt{Takes}(n, c) \rightarrow \mathtt{Enrollment}(f(n), c)$$

is a SO tgd. Recall that the composition of the successive mappings in Example 1 is not definable by any finite set of s-t tgds. Fagin et al. [8] showed that the SO tgd above defines the composition of the two mappings given in Example 1. Hence, existentially quantified function symbols are a necessary extension to the language of s-t tgds for composing mappings. Intuitively, the SO tgd states that if a student with name n takes a course c in **Takes**, then the student id of n, which is denoted by the function *f(n)*, is associated with c in **Enrollment**. An example of SO tgds where equalities are involved is described next.

Example 2. Let $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ be two successive mappings. The first schema $\mathbf{S}_1$ consists of a unary relation symbol **Emp**, the second schema $\mathbf{S}_2$ consists of a binary relation symbol $\mathbf{Mgr}_1$, and the third schema $\mathbf{S}_3$ consists of a binary relation symbol **Mgr** and a unary relation symbol **SelfMgr**. The s-t tgds in Σ_{12} and Σ_{23} are:

$$\sum\nolimits_{12} = \{\forall e(\mathbf{Emp}(e) \rightarrow \exists m\, \mathbf{Mgr}_1(e, m))\}$$

$$\sum\nolimits_{23} = \{\forall e \forall m(\mathbf{Mgr}_1(e, m) \rightarrow \mathbf{Mgr}(e, m)), \\ \forall e\, (\mathbf{Mgr}_1(e, e) \rightarrow \mathbf{SelfMgr}(e))\}$$

Intuitively, the s-t tgd in Σ_{12} asserts that every employee e in **Emp** must have a manager m and this association can be found in $\mathbf{Mgr}_1$. The first s-t tgd in Σ_{23} asserts that the **Mgr** relation contains the $\mathbf{Mgr}_1$ relation and the second s-t tgd in Σ_{23} asserts that **SelfMgr** contains employees who are their own managers according to the $\mathbf{Mgr}_1$ relation.

Fagin et al. [8] showed that the following SO tgd defines the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$.

$$\exists f(\forall e\, (\mathbf{Emp}(e) \rightarrow \mathbf{Mgr}(e,\ f(e)) \wedge \forall e\, (\mathbf{Emp}(e) \\ \wedge (e = f(e)) \rightarrow \mathbf{SelfMgr}(e)))$$

They also showed that the above SO tgd is not logically equivalent to any finite or infinite sets of SO tgds without equalities. In other words, the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ is not definable by SO tgds without equality. Hence, equalities are a necessary extension to the language of s-t tgds for composing mappings.

The extensions of s-t tgds with function symbols and equalties in SO tgds are necessary to compose mappings specified by s-t tgds. Fagin et al. [8] also showed that SO tgds are closed under composition. This means that the composition of two mappings, each specified by an SO tgd, is another SO tgd.

Example 3. An illustration of the algorithm described in [8] for composing two mappings, based on Example 2, is described next. The algorithm takes as input two mappings, $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$, specified by s-t tgds and returns as output a mapping $\mathcal{M}_{13}$ that defines the composition of the two input mappings.

The first step of the algorithm is to transform the s-t tgds in Σ_{12} and Σ_{23} into SO tgds by introducing Skolem functions to replace existentially quantified variables. For example, Σ_{12} and Σ_{23} will now become Σ'_{12} and Σ'_{23} respectively.

$$\sum\nolimits'_{12} = \{\exists f \forall e\, (\mathbf{Emp}(e) \rightarrow \mathbf{Mgr}_1(e, f(e)))\}$$

$$\sum\nolimits_{23}' = \{\forall e \forall m(\mathbf{Mgr}_1(e,m) \rightarrow \mathbf{Mgr}(e,m)),\\ \forall e(\mathbf{Mgr}_1(e,e) \rightarrow \mathbf{SelfMgr}(e))\}$$

In particular, observe that Σ'_{12} now consists of an SO tgd with function *f(e)* that denotes the manager of employee *e*. The arguments of f consist of all universally quantified variables in the s-t tgd. The next step of the algorithm combines Σ'_{12} with Σ'_{23} to obtain Σ_{13}' by replacing atomic formulas on the left-hand-side of conjuncts of SO tgds in Σ'_{23} with atomic formulas from $\mathbf{S}_1$ through conjuncts of SO tgds in Σ'_{12}. In the running example, Emp(e_0) →Mgr$_1$(e_0, f(e_0)) is combined with Mgr$_1$(e,m) →Mgr(e,m) to obtain

$$\mathbf{Emp}(e_0) \wedge (e = e_0) \wedge (m = f(e_0)) \rightarrow \mathbf{Mgr}(e,m))$$

and **Emp**(e_1) → **Mgr**$_1$(e_1, f(e_1)) is combined with **Mgr**$_1$(e, e) → **SelfMgr**(e) to obtain

$$\mathbf{Emp}(e_1) \wedge (e = e_1) \wedge (e = f(e_1)) \rightarrow \mathbf{SelfMgr}(e)$$

Observe that the equalities generated by this step has the form $y = t$ where y is a variable in Σ'_{23} and t is a term based on the variables and functions of Σ'_{12}. The next step of the algorithm removes variables from Σ'_{23} according to the equalities. For example, e is replaced with e_0 in the first formula and e is replaced with e$_1$ in the second formula to obtain the following SO tgds:

$$\mathbf{Emp}(e_0) \wedge (m = f(e_0)) \rightarrow \mathbf{Mgr}(e_{0,} m)$$

$$\mathbf{Emp}(e_1) \wedge (e_1 = f(e_1)) \rightarrow \mathbf{SelfMgr}(e_1)$$

At this point, the variable m is replaced with f(e_0) to obtain the following SO tgds:

$$\mathbf{Emp}(e_0) \rightarrow \mathbf{Mgr}(e_0, f(e_0))$$

$$\mathbf{Emp}(e_1) \wedge (e_1 = f(e_1)) \rightarrow \mathbf{SelfMgr}(e_1)$$

Finally, when no more variables of Σ'_{23} can be replaced, the following SO tgd is returned.

$$\exists f(\forall e(\mathbf{Emp}(e) \rightarrow \mathbf{Mgr}(e, f(e))) \wedge \forall e(\mathbf{Emp}(e)\\ \wedge (e = f(e)) \rightarrow \mathbf{SelfMgr}(e)))$$

Data Exchange and Query Answering. Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$ be a mapping. Given a finite instance*I*over the schema $\mathbf{S}_1$, the *data exchange problem* [7] is to construct a finite instance J over the schema $\mathbf{S}_2$ such that (*I, J*) satisfies all the formulas specified in Σ. Such an instance J is called a *solution* of I under the mapping $\mathcal{M}$. If Σ is specified by a finite set of s-t tgds, many solutions for I under $\mathcal{M}$ may exist in general because s-t tgds underspecify the data exchange process in general. In [7], the classical *chase* procedure [1,11] has been used to construct *universal* solutions of I under a mapping $\mathcal{M}$. Universal solutions are the most general type of solutions in the following sense: If J is a universal solution for *I*under $\mathcal{M}$, this means that J is a solution for I under $\mathcal{M}$ with the additional property that J has a *homomorphism* into every solution for I under $\mathcal{M}$. Intuitively, an instance K has a homomorphism into an instance K′ if K can be embedded in K' (modulo the renaming of nulls that occur in K). It was shown in [7] that universal solutions can be computed in polynomial time when the mapping is fixed. Universal solutions are desirable not only because they are the most general, but also because they can be used to compute the certain answers of unions of conjunctive queries that are posed against the schema $\mathbf{S}_2$ in polynomial time. If q is a k-ary query over $\mathbf{S}_2$, then the certain answers of q with respect to an instance I over $\mathbf{S}_1$, denoted as $certain_{\mathcal{M}}(q, I)$, is the set of all k-tuples t of constants from I such that t ∈ $q(J)$ for every solution J of I under $\mathcal{M}$. It was shown in [7] that if J is a universal solution for I under $\mathcal{M}$ and q is a union of conjunctive queries, then $certain_{\mathcal{M}}(q, I)$ can be computed as follows: (i) Evaluate $q(J)$ and then (ii) discard tuples from $q(J)$ that contain nulls. The remaining tuples obtained from this process, denoted as $q(J)_{\downarrow}$, form the certain answers of q with respect to I.

In the case where there are two or more successive mappings specified by s-t tgds and only the target instance over the last schema is of interest, one approach to obtain a universal solution of the last schema when given an instance I over the first schema is to perform a series of data exchanges (using the chase procedure [7]) starting from I according to the sequence of mappings. The final target instance that is arrived through this process is a universal solution for I for the sequence of mappings. (The series of data exchanges produces a universal solution. This result can be found in Proposition 7.2 of [6].) Obviously, one drawback of this approach is the unnecessary construction of potentially many intermediate instances which are not of interest. Another approach that avoids the construction of intermediate instances altogether is to first compose the sequence of mappings to obtain a composed mapping over the first source schema and the final target schema. After this, data are exchanged (by using the chase procedure) according to the composed mapping. However, as described earlier, the language of s-t tgds may no longer be sufficient for defining the composition of two or more successive mappings. Instead, SO tgds are needed to describe the composition of successive mappings in general. In [8], the classical chase technique is extended to SO tgds. They showed that chasing with SO tgds is again a polynomial time procedure and that the chase with SO tgds produces a universal solution as in s-t tgds. Hence, a universal solution for the final target schema can be computed by simply chasing I over the composed mapping. As a consequence, the certain answers of unions of conjunctive queries that are posed over the target schema of a mapping specified by SO tgds can also be computed in polynomial time: First, a universal solution J of I is computed by chasing I with the mapping. After this, compute $q(J)_{\downarrow}$, as described earlier.

Key Applications

One important application of composing mappings is schema evolution. Consider the figure shown in the Definition section. If $\mathbf{S}_3$ is an evolved schema of $\mathbf{S}_2$ and the mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ are given, then it is possible to derive the direct relationships between $\mathbf{S}_1$ and $\mathbf{S}_3$ by composing $\mathcal{M}_{12}$ with $\mathcal{M}_{23}$. See [14] for an application of composition to schema evolution.

Another important application of composition is to optimize the migration of data through a sequence of mappings. An end-to-end mapping is first assembled from a sequence of two or more mappings by composition before data are migrated through the assembled mapping. The benefit of using an end-to-end mapping for migrating data from the first schema to the last schema in the sequence of mappings is the potential savings from the sequence of unnecessary data migration steps through the intermediate schemas along the sequence of mappings. For a similar reason, the end-to-end mapping could also be used to optimize query rewriting. Referring back to the figure in the Definition section, if a query that is posed against the schema $\mathbf{S}_3$ needs to be rewritten into a query against the schema $\mathbf{S}_1$, it is potentially rewarding to first compose the sequence of mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ and then reason about the rewriting through the composition rather than through the sequence of mappings $\mathcal{M}_{23}$ and $\mathcal{M}_{12}$.

Composed mappings can also be used as an abstraction for a sequence of data migration steps. A recent work [4] on Extract-Transform-Load (ETL) systems illustrates this point. An ETL script can be modeled as a network of mappings describing the flow of data from a source to a target. By composing various sequences of mappings in the network of mappings, an abstraction of the overall ETL transformation can be achieved.

Cross-references

- ▶ Data Exchange
- ▶ Schema Mapping

Recommended Reading

1. Beeri C. and Vardi M.Y. A Proof Procedure for Data Dependencies. J. ACM, 31(4):718–741, 1984.
2. Bernstein P.A., Green T.J., Melnik S., and Nash A. Implementing Mapping Composition. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 55–66.
3. Bernstein P.A., Halevy A.Y., and Pottinger R. A Vision of Management of Complex Models. ACM SIGMOD Rec., 29(4):55–63, 2000.
4. Dessloch S., Hernández M., Wisnesky R., Radwan A., and Zhou J. Orchid: Integrating Schema Mapping and ETL. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 1307–1316.
5. Fagin R. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In Complexity of Computation, SIAM-AMS Proceedings, Vol. 7, R.M. Karp (ed.), 1974, pp. 43–73.
6. Fagin R. Inverting Schema Mappings. ACM Trans. Database Syst., 32(4):24, 2007.
7. Fagin R., Kolaitis P.G., Miller R.J., and Popa L. Data Exchange: Semantics and Query Answering. Theoretical Computer Science, 336(1):89–124, 2005.
8. Fagin R., Kolaitis P.G., Popa L., and Tan W.C. Composing Schema Mappings: Second-Order Dependencies to the Rescue. ACM Trans. Database Syst., 30(4):994–1055, 2005.
9. Lenzerini M. Data Integration: A Theoretical Perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
10. Madhavan J. and Halevy A.Y. Composing Mappings Among Data Sources. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 572–583.
11. Maier D., Mendelzon A.O., and Sagiv Y. Testing Implications of Data Dependencies. ACM Trans. Database Syst., 4(4): 455–469, 1979.
12. Nash A., Bernstein P.A., and Melnik S. Composition of Mappings Given by Embedded Dependencies. ACM Trans. Database Syst., 32(1):4, 2007.
13. Popa L., Velegrakis Y., Miller R.J., Hernández M.A., and Fagin R. Translating Web Data. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 598–609.
14. Yu C. and Popa L. Semantic Adaptation of Schema Mappings when Schemas Evolve. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 1006–1017.

Schema Matching

Anastasios Kementsietsidis
IBM T.J. Watson Research Center, Hawthorne,
New York, USA

Synonyms

Attribute or value correspondence

Definition

Schema matching is the problem of finding potential associations between elements (most often attributes or relations) of two schemas. Given two schemas S_1 and S_2, a solution to the schema matching problem, called a *schema matching* (or more often a matching), is a set of *matches*. A match associates a schema element (or a set of schema elements) in S_1 to (a set of) schema elements in S_2. Research in this area focuses primarily on the development of algorithms for the discovery of matchings. Existing algorithms are often distinguished by the information they use during this discovery. Common types of information used include the schema dictionaries and structures, the corresponding schema instances (if available), external tools like thesauri or ontologies, or combinations of these techniques. Matchings can be used as input to *schema mappings* algorithms, which discover the semantic relationship between two schemas.

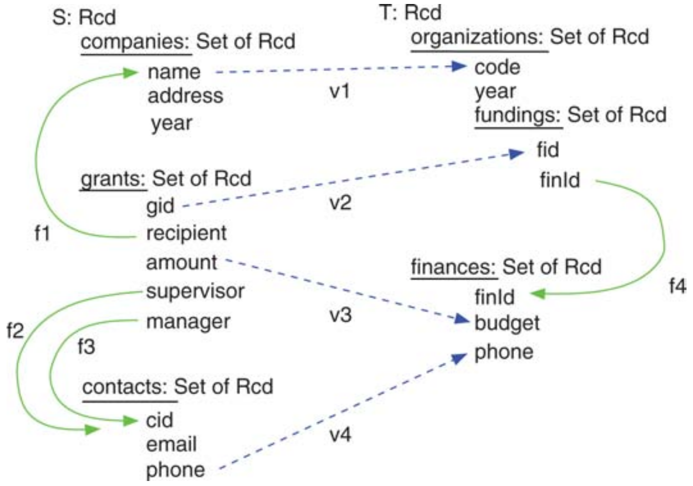
Historical Background

A *schema matching* is most often a binary relation between the elements of two schemas, but may, in a few approaches, be a relation between sets of elements in different schemas. In general, a matching represents a potential semantic relationship, but does not specify the semantics. For example, a matching between attributes $S_1.A$ and $S_2.B$ indicates that there may be some semantic relationship between these attributes. Examples of possible semantic relationships include subset relationships (e.g., all values of $S_1.A$ are also values of $S_2.B$) or has-a relationships (e.g., each value of $S_2.B$ has-a $S_1.A$ value). Consider the matches (v_1 through v_4) in Fig. 1. The matching helps in understanding the possible relationship between the schemas, but is not sufficient to determine how to transform data or queries from one schema to another. In contrast, a *schema mapping* (or mapping) is a specification of the semantic relationship between schemas [8]. The discovery of matchings between elements of different schemas has been studied for decades, most notably in the context of the *schema integration* problem [1]. A solution to the schema integration problem presumes the ability to discover elements in the various schemas that are potentially *semantically related*, including those that may represent the same real-world concepts. Schema matching algorithms attempt to find candidate elements that may have a semantic relationship, though notably, they do not attempt to specify (or differentiate) the semantics of the relationship. These algorithms have been motivated by the presence of naming or structural differences (referred to as *conflicts*) among schemas that have been developed independently. Such differences are due to the fact that a real-world concept might have a different name or representation in different schemas. Schema integration deals with the development of methodologies to *discover* matchings in the presence of conflicts, but the main focus is on how each methodology *resolves* such conflicts (e.g., through schema transformations) so that the real-world concept is *uniquely* represented in the global schema. Indeed, from the five generic schema integration steps in each methodology, identified in [1], three of them deal with the resolution of conflicts. On the other hand, schema matching deals exclusively with the development of algorithms to *discover* matchings (see [9] for a survey of matching approaches).

Foundations

A matching between a pair of schemas S_1 and S_2 is typically a binary relation between the elements of the two schemas. In such cases, the so-called *local cardinality* of the matching is said to be one-to-one (1:1). Some algorithms consider matchings between sets of elements and, in the terminology of [9], are said to have a many-to-many local cardinality (see Figs. 2 and 3). In most approaches, the matching is between individual attributes of the schemas (Fig. 1). Matching algorithms compute a matching between S_1 and S_2 through a process which can abstractly be described by the following steps:

1. Consider (possibly all) pairs of elements s_1, s_2 with $s_1 \in S_1$ and $s_2 \in S_2$.
2. Compute a *score* indicating the confidence in the validity of each match between s_1 and s_2.

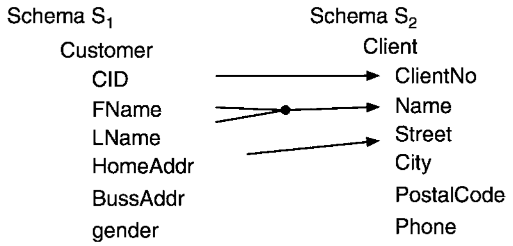


Schema Matching. Figure 1. A matching between source and target schemas.

3. Compute a matching by filtering and selecting a subset of the matching elements of the previous step.

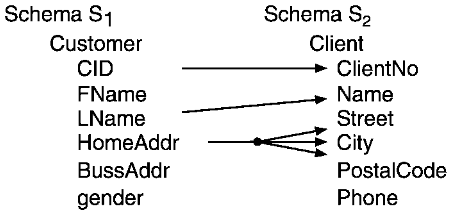
Existing matching algorithms differ on how they implement each of these steps. Each implementation needs to make some key decisions in each step, hence the substantial diversity of existing solutions. During the first step, an important consideration is whether every possible pair of schema elements will be considered as a candidate for a match [6], or whether there is a more sophisticated mechanism in place to prune the potentially unrelated element pairs considered [3]. Many approaches assume that an element can be associated with at most one element in another schema (a restriction referred to as (1:1) global cardinality in [9]). For matching algorithms that only associate elements (not sets of elements), this means the resulting matching is a simple 1:1 relation over the schema elements.

The second step is probably the most important, and there is a huge space of alternatives for the computation of scores. Score computation may take into account the name and types of schema elements, the structure of schemas and the corresponding nesting depths of elements, instance-level information (if instances of the schemas are available) like value ranges and patterns (for example, the frequency or position of substrings appearing in attribute values), or it may combine various types information to compute a score. Matching algorithms are commonly classified by the type of information used during scoring computation (see the classification in [9]). The term *individual* matcher is commonly used to describe algorithms that consider only a single (or a limited) type of information during score computation. Individual matchers are further classified into *schema-based* and *instance-based* depending on the type of information (i.e., schema versus instance) used during this phase. In contrast to individual matchers, *hybrid* or *composite* matchers rely on several types of information during score computation, where each type essentially corresponds to a different individual matcher. While hybrid matchers combine the results of multiple individual matchers in a *prespecified* manner, composite matchers are more flexible and allow for a dynamic composition of individual matchers which can be customized for the specific schemas being matched.



Schema Matching. Figure 2. A matching with *n*:1 local cardinality.

In the final step, there are two key considerations which influence the selection of matching elements that will comprise the result matching. First, the selection of matching elements is influenced by the supported *cardinality* which determines whether, or not, sets of



Schema Matching. Figure 3. A matching with 1:*n* local cardinality.

elements (for example, Street, City and PostalCode as a set as in Fig. 3) are considered in the matching.

The second consideration in this final step relates to how matching elements are selected based on their score. A common approach is to select matching elements whose scores are above a certain threshold and then select the matching elements with the maximum score, among the alternatives [6]. While such an approach results in matchings that are locally optimal, a more sophisticated approach considers maximizing the cumulative score of the matching elements in a matching [7].

State of the Art

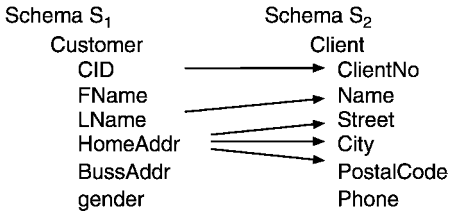
There are many approaches to schema matching, so we offer a non-comprehensive overview of some of the representative approaches.

Cupid [6] is one of the first hybrid schema matching algorithms proposed in the context of model management. The algorithm considers initially every possible pair of elements in the two input schemas and thus its local cardinality is 1:1. It computes a linguistic and structural similarity score between these elements from which a weighted mean is computed using these two scores. The selection of matching elements in the resulting matching is performed by using a threshold over the computed scores and the supported global cardinality is 1:1, although it is suggested that matchings with global cardinality 1:*n* can also be supported (Fig. 4).

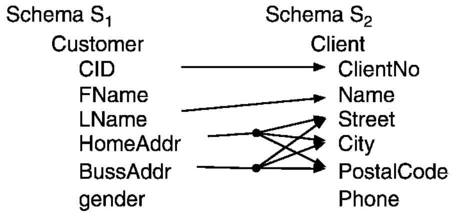
Coma [4] is a composite matcher with an extensible library of single and hybrid matchers. For example, Cupid might become one of the matchers used by Coma. Both the local and global cardinality is 1:1, and each *component* matcher of COMA computes a score between every pair of elements in the input schemas. Being a composite matcher, emphasis in Coma is given on how the results of component matchers are combined and four alternative strategies are proposed to this end. The four strategies compute the score of a matching element by taking the max, min, average or weighted sum of scores, computed for this element, of the component matchers. Experiments with these four strategies show that average gives the best results on the schemas tested.

The majority of matchers discover matchings whose local cardinality is 1:1. The iMap [3] matcher in contrast emphasizes the discovery of complex matching elements between two schemas, i.e., matchings with a local cardinality of *n*:1. For each element in the target schema, iMap employs a set of *specialized searchers* to discover candidate sets of elements in the source schema that together can be associated to the target schema element. Examples of the matchers supported are a numerical matcher, which discovers matches between elements containing numerical values; a categorical matcher, for categorical attributes; a unit conversion matcher; and a date matcher. In terms of global cardinality, iMap supports *n*:1 matchings (Fig. 5), since it allows an element to participate in more than one complex matching. An interesting feature of iMap is that apart from discovering (candidate) matchings, it also provides a module which traces the key decisions made by the system during matching discovery and it can therefore present the reasoning behind a suggested matching, in a human understandable format.

Kang and Naughton [5] make the interesting observation that matching algorithms often rely on *interpreting* the element names and values in the two input schemas, that is, they assume that the names used to described the same real-world concept or entity are syntactically and semantically related (e.g., a relational column named COLOR in S_1 versus one called PAINT in S_2). Therefore, when different element names are used, for the same elements in the two schemas (e.g., the former column is called CID in S_1 and PMS (PMS stands for the Pantone Color Matching System used in various industries.) in S_2), or different data encodings are used for the same real-world domain (e.g., different encodings for colors), then existing matching algorithms fail to discover appropriate matching elements. To this end, Kang and Naughton propose an instance-based matching algorithm that does not interpret values. Their matcher relies on the well-known notions of entropy and mutual-information, from Information Theory, to discover a matching between two input schemas. In a nutshell, their approach consists of two main steps. First, for each input



Schema Matching. Figure 4. A 1:1 local and 1:*n* global cardinality matching.



Schema Matching. Figure 5. A 1:*n* local and *n*:1 global cardinality matching.

schema, it computes the mutual information between each pair of attributes within the schema. The second step considers each possible matching with local and global cardinality of 1:1 and computes a score for this matching, a computation that takes into account the mutual information and entropy of the matched elements.

Key Applications

Schema Mapping, Schema Integration

Future Directions

Schema matchings represent potential associations between a pair of schema elements (or between two sets of schema elements). Matching algorithms do not discover the meaning of the association. A match between elements $S_1.A$ and $S_2.B$ may be discovered because these attributes contain similar values, because their names have a small edit-distance, their names are related in a domain ontology, or for numerous other reasons. Hence, matchings by themselves are not directly useful until they have been interpreted, either by a human, or a system designed to infer the semantic relationship between the elements. The most common example of the latter are schema mapping algorithms, which are designed to infer the semantic relationship between two schemas.

Cross-references

▶ Information Integration
▶ Metadata
▶ Schema Mapping

Recommended Reading

1. Batini C., Lenzerini M., and Navathe S.B. A comparative analysis of methodologies for database schema integration. ACM Comput. Surv., 18(4):323–364, 1986.
2. Bernstein P.A., Halevy A.Y., and Pottinger R.A. A vision for management of complex models. ACM SIGMOD Rec., 29 (4):55–63, 2000.
3. Dhamankar R., Lee Y., Doan A., Halevy A., and Domingos P. iMap: Discovering complex semantic matches between database schemas. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
4. Do H. and Rahm E. Coma – a system for flexible combination of schema matching approaches. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 610–621.
5. Kang J. and Naughton J.F. On schema matching with opaque column names and data values. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 205–216.
6. Madhavan J., Bernstein P.A., and Rahm E. Generic schema matching with cupid. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 49–58.
7. Melnik S., Garcia-Molina H., and Rahm E. Similarity flooding: a versatile graph matching algorithm. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 117–128.
8. Miller R.J., Haas L.M., and Hernández M.A. Schema matching as query discovery. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 77–88.
9. Rahm E. and Bernstein P.A. A survey of approaches to automatic schema matching. VLDB J., 10(4):334–350, 1994.

Schema Normalization

▶ Design for Data Quality

Schema Tuning

Philippe Bonnet[1], Dennis Shasha[2]
[1]University of Copenhagen, Copenhagen, Denmark
[2]New York University, New York, NY, USA

Definition

Schema tuning is the activity of organizing a set of table designs in order to improve overall query and update performance.

Historical Background

Table design entails deciding which tables to implement and which attributes to put in those tables. Other sections of this encyclopedia (design theory, normalization theory) discuss a mathematical model of table design to eliminate redundancy. Sometimes however redundancy can be good for performance, so database tuners must consider the possibility of a principled incorporation of redundancy.

Foundations

Normalization tends to break up the attributes of an application into separate tables. Consider the normalized schema consisting of two tables:

Blog(blog_id, author_id, title, numreaders) and *Author*(author_id, author_city).

If one frequently wants to associate blogs with the city of their authors, then this table design requires a join on author_id for each of these queries. A denormalized alternative is to add author_location to *Blog*, yielding *Blog*(blog_id, author_id, product, numreaders, author_city) and *Author*(author_id, author_city).

The *Author* table avoids anomalies such as the inability to store the location of an author whose blogs are perhaps temporarily offline.

Comparing these two schemas, one can see that the denormalized schema requires more space and more work on insertion of a blog. On the other hand, the denormalized schema is much better for finding the authors in a particular city.

The tradeoff of space plus insertion cost vs. improved speeds for certain queries is the characteristic one in deciding when to use a denormalized schema. Good practice suggests starting with a normalized schema and then denormalizing sparingly.

Redundant Tables

The previous example showed that redundancy can be helpful. The form of redundancy there was to repeat an association between two fields (in this case between authors and their address) for every blog.

One may also consider a completely redundant table.

For example:

Blog(blog_id, author_id, product, numreaders, author_city)

Author(author_id, author_city).

City_Agg(city, totalreaders)

This improves performance if one frequently wants to know the total readers per author city, but imposes an update time as well as a small space overhead. The trade-off is worthwhile in situations where many aggregate queries are issued and an exact answer is required.

Key Applications

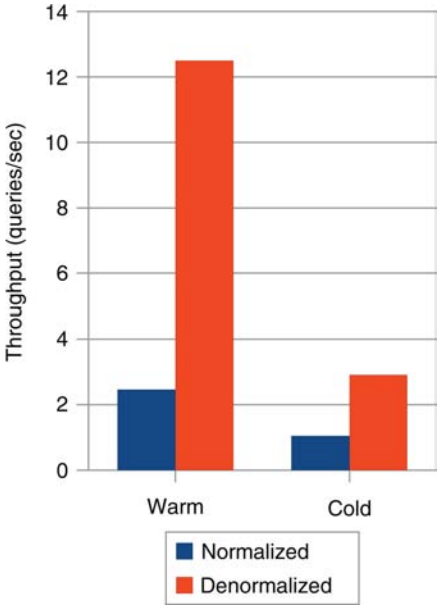
Schema tuning is relevant for all applications, but it is especially important for complex multi-table queries, particularly involving aggregates. Data warehousing applications typically include denormalized, redundant schemas, because data warehouses can be engineered to be updated at off hours and then intensively queried during the work day.

Experimental Results

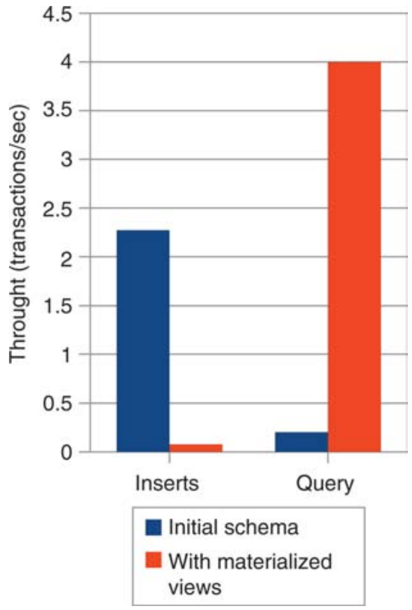
Denormalization

This experiment compares the performance impact of denormalization in the example presented above. Consider a query that finds the author in a given city. That query requires a join in the normalized schema whereas it requires a simple lookup in the denormalized schema.

Figure 1 presents the performance figures running this example on MySQL 6.0. The author table is



Schema Tuning. Figure 1. Denormalization experiment on MySQL 6.0.



Schema Tuning. Figure 2. Materialized view experiment on MySQL 6.0.

populated with 100,000 tuples, and the blog table with 50,000. Note that the denormzalized schema provides a significant speed-off whether the cache is cold (in which case IOs are issued) or warm (the data already resides in the database cache).

Materialized Views

This experiment illustrates the trade-off between query speed-up and insert slow-down when using a materialized view. Consider the schema from the example above. The code includes a trigger in MySQL that maintains the materialized view when data are inserted in the blog table. The experiment includes the insertion throughput as well as query throughput. The query find the total number of readers per city.

Figure 2 shows the expected trade-off. Insertions are much slower with the materialized view due to trigger execution. Queries are much faster, however, because the query requires a join and an aggregate computation under the initial schema, whereas the query requires only a simple lookup when using the materialized view.

URL to Code and Data Sets

Denormalization experiment: http://www.databasetuning.org/sec = denormalization

Materialized view experiment: http://www.databasetuning.org/sec = materalized_views

Cross-references

- ▶ Application-level Tuning
- ▶ Clustering Index
- ▶ Design Theory
- ▶ Normalization Theory
- ▶ Performance Monitoring Tools

Recommended Reading

1. Celko J. and Joe Celko's. SQL for Smarties: Advanced SQL Programming (3rd Edn.). Morgan Kaufmann, San Fransisco, CA, 2005.
2. Kimball R. and Ross M. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (2nd Edn.). Wiley, New York, NY, 2002.
3. Shasha D. and Bonnet P. Database Tuning: Principles, Experiments and Troubleshooting Techniques. Morgan Kaufmann, San Fransisco, CA, 2002.
4. Tow D. SQL Tuning. OReilly, North Sebastopol, CA, 2003.

Schema Versioning

John F. Roddick
Flinders University, Adelaide, SA, Australia

Definition

Schema versioning deals with the need to retain current data, and the ability to query and update it, through alternate database structures. (The structure of a database is held in a *schema* (pl. *schemata* or *schemas*). Commonly, particularly in temporal databases, these schemata represent the historical structure of a database but this may not always be the case.) *Schema Versioning* requires not only that data are not lost in schema transformation but also requires that all data are able to be queried, both retrospectively and prospectively, through user-definable version interfaces. *Partial schema versioning* is supported when data stored under any historical schema may be viewed through any other schema but may only be updated through one specified schema version – normally the current or active schema. (Schema evolution and schema versioning has been conflated in the literature with the two terms occasionally being used interchangeably.)

Historical Background

Multiple versions of a database schema may exist for a number of reasons. First, as a result of changes in system functionality and the external environment, the structure of a database system might change over time but the historical shape of the database might need to be retained. Second, future versions might be created to develop and test later versions of a system. Third, more than one schema may be required in parallel to access the same data in a number of ways. Temporal databases, because of their requirement to maintain the context of historical information, are particularly affected by schema change.

The idea of schema versioning was introduced in the context of OODBs with a number of systems implementing techniques to handle multiple schema (such as Encore [12], Gemstone [7] and Orion [1]), including those that might be required for reasons other than simple historical succession. For example, parallel, alternate schema might be required to conceptualize an idea from a number of semantically consistent but different perspectives. In particular, polymorphism was suggested as a mechanism for providing some stability when faced with changing schema [6].

In order to maintain long-established concepts such as soundness and completeness, algebraic extensions have also been discussed [3]. More recently, schema versioning has also been considered in the context of spatio-temporal databases [9] and meta-data management [2,4].

Foundations

Schema versioning is closely related to the concepts of schema integration and data integration – all deal with the problems of accessing data through schema that were not used when the data were originally stored. However, the idea of maintaining multiple schemata, and allowing data to be accessed through them, raises a number of issues.

- What is the significance of a difference between two schema (or two databases) and therefore what is the informational cost of the change?
- What are the atomic operations of schema translation or transformation and what happens to the data during these operations?
- Are there any modelling techniques that can be used?
- Are there any other side-effects or opportunities (for instance in query language support)?

Types of Schema Evolution

As outlined elsewher, four forms of schema evolution can be envisaged - attribute, domain, relation and key evolution. Moreover, one change may involve more than one type of evolution, such as changing the domain of a key attribute and may also be reflected in the conceptual model of the system. Importantly for schema versioning, the inverse function for each of these must be considered. For example, when a schema (merely) evolves by vertically splitting a relation in two with data being suitably transformed, for schema versioning to be allowed, active transformation functions must be provided if the old schema is still to be utilized.

Practical and Theoretical Limits of Schema Versioning

It has been shown that in order to update data stored under two different schemata using the opposite schemata, they must have equivalent information capacity – all valid instances of some schema S_1 must be able to be stored under S_2 and vice-versa [5]. Specifically, $S_1 \equiv S_2$ if $I(S_1) \rightarrow I(S_2)$ is bijective where $I(S)$ is the set of all valid instances of S. This means that, in theory, *full* schema versioning across nonequivalent versions of a schema is unattainable and much research in the area adopts the weaker concept of *partial schema versioning* in which data stored under any historical schema may be viewed through any other schema but may only be updated through one specified schema version - normally the current or active schema.

However, in practice, many schema changes that expand or reduce the information capacity of a schema can be done without loss of information. This is the case, for example, for domains defined too large for any of the data, or for the creation of subclass relations from a single relation where the subclass type attribute already exists. It is a common practice, where there is some ambiguity in the requirements definition of a system, to allow for a larger schema capacity – some of which may never materialize and, as a result, changes to schemata to adhere to the data actually collected are not uncommon. For example, allowing for time and date when only date is recorded in practice.

Thus the limits for *practical schema versioning* in a database $\mathfrak{D}$ are that $S_1 \overset{p}{\equiv} S_2$ (S_1 and S_2 have practical equivalent information capacity) if $I'(\mathfrak{D}| S_1) \rightarrow I'(\mathfrak{D}| S_2)$ is bijective where $I'(\mathfrak{D}|S_\mathrm{n})$ is the set of all instances of S_n inferrable from $\mathfrak{D}$ given the constraints of S_n. This means that whether the integration of two schema is possible is dependent on the data held as well as the schema definition and while this makes the ability to undertake wholesale change less predictable, it may provide an acceptable level of support in many practical situations.

Completed Schemas

In order to make all data for a relation available without the need to issue multiple queries, each targeting different time periods, the concept of a *completed schema*, *C*, can be employed that includes all attributes that have ever been defined over the life of a relation. The domain of each attribute in *C* is considered syntactically general enough to hold all data stored under every version of the relation and the implicit primary key of *C* is defined as the maximal set of key attributes for the relation over time. Depending on the mechanism used to implement schema versioning, the completed schema can then be used by a series of view functions. For example, in Fig. 1, V_{t_5} maps the completed schema C to a subset of the attributes in a schema S_{t_5} active during t_5. A converse view function W_{t_2} maps from S_{t_2} to C.

Thus the data stored during t_2 may be mapped to the format specified during t_5 through invocation of $V_{t_5}(W_{t_2}(S_{t_2}))$.

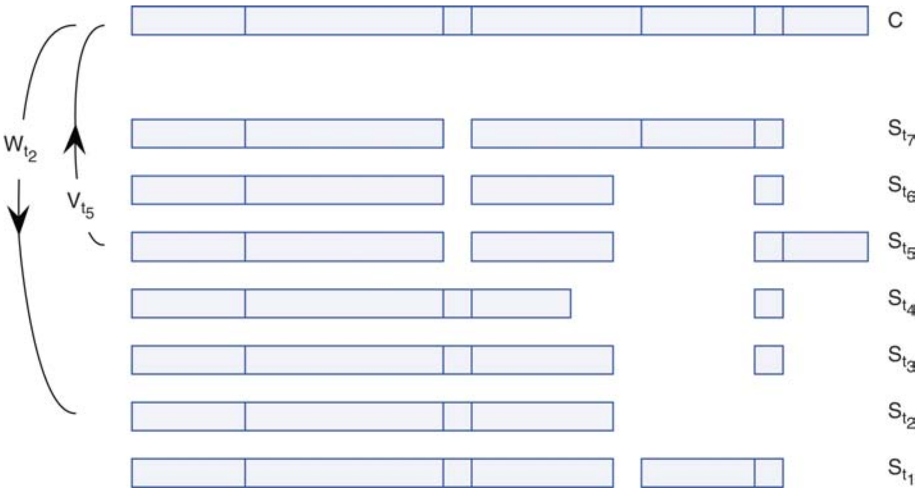
Query Language Support

Support for schema versioning does not yet exist in commercially available query languages. However, the TSQL2 proposal [10] and an earlier SQL/SE proposal [8] outlined some parts of the solution. As examples of such extensions:

- Reference to the *completed schema* can be included to provide access to all data;
- The specification of the schema could be done either through the specification of a global *schema-time* as in TSQL2, which would be useful for SQL embedded in a program with the schema-time set to compile time, or explicitly as part of the query;
- Attribute definition might be able to be tested by adding a test to see if a value was missing because it was *not defined* rather than being merely *null*;
- The language may also include meta-data queries such as the ability to ask what version of the schema a given piece of data adheres to.

Instance Amendment

For schema versioning in which the old schemata are still considered valuable, once a schema change is accepted, there are three options regarding the change to existing data. First, the underlying instances may be coerced to the new structure. While conceptually simple, this may result in lost information and an inability to reverse schema amendments. Secondly, data are retained in the format in which it was originally stored. This retains information content at the expense of more complex (and slower) translation of data when needed. Third, data are initially retained in the format in which it was originally stored but is converted when amended. While the most complex option, it has the advantage of identifying data that has not been amended since the schema change.



Schema Versioning. Figure 1. Versions of Schemata over time.

Future Directions

Schema versioning research has moved from low-level handling of syntactic elemental changes to more model-directed semantic handling of change. There are a number of other issues that make schema versioning non-trivial. Some of these represent future issues to be investigated.

- Many schema change requirements involve composite operations and thus a mechanism for schema level commit and rollback functions could be envisaged which could operate at a higher level to the data level commit and rollback operations.
- Access rights considerations are particularly a problem in object-oriented database systems. Consider, for example, a change to a class (e.g., Employees) from which attributes are inherited to a sub-class (e.g., Engineers) for which the modifying user has no legitimate access. Any change to the definition of attributes inherited from the superclass can be considered to violate the access rights of the subclass. Moreover, in some systems ownership of a class does not imply ownership of all instances of that class.
- In temporal databases the concept of vacuuming (q.v.) allows for the physical deletion of temporal data in cases where the utility of holding the data are outweighed by the cost of doing so [11]. Similar consideration must be given to the deletion of obsolete schema definitions, especially in cases where no data exists adhering to either that version (physically) or referring, through its transaction-time values, to the period in which the definition was active.

Cross-references

▶ Conceptual Modelling
▶ Schema Evolution
▶ Temporal Algebras
▶ Temporal Evolution
▶ Temporal Query Languages

Recommended Reading

1. Kim W., Ballou N., Chou H.T., Garza J.F., and Woelk D. Features of the orion object-oriented database system. In Object-Oriented Concepts, Databases and Applications. W. Kim and F. Lochovsky (eds.). ACM Press, New York, 1989, pp. 251–282.
2. Madhavan J. and Halevy A.Y. Composing mappings among data sources. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 572–583.
3. McKenzie L. and Snodgrass R. Schema evolution and the relational algebra. Inf. Syst., 15(2):207–232, 1990.
4. Melnik S., Rahm E., and Bernstein P.A. Rondo: a programming platform for generic model management. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 193–204.
5. Miller R. Ioannidis Y. and Ramakrishnan R. The use of information capacity in schema integration and translation. In Proc. 29th Int. Conf. on Very Large Data Bases, 1993, pp. 120–133.
6. Osborn S. The role of polymorphism in schema evolution in an object-oriented database. IEEE Trans. Know. Data Eng., 1(3):310–317, 1989.
7. Penney D. and Stein J. Class modification in the gemstone object-oriented DBMS. In Proc. 1987 Conf. on Object-Oriented Programming Systems, Languages, and Applications, 22(12): 111–117, 1987.
8. Roddick J.F. SQL/SE - a query language extension for databases supporting schema evolution. ACM SIGMOD Rec., 21 (3):10–16, 1992.
9. Roddick J.F., Grandi F., Mandreoli F., and Scalas M.R. Beyond schema versioning: a flexible model for spatio-temporal schema selection. Geoinformatica, 5(1):33–50, 2001.
10. Roddick J.F. and Snodgrass R. Schema versioning support, Chapter 22. In The TSQL2 Temporal Query Language. R. Snodgrass (ed.). Kluwer, Boston, 1995, pp. 427–449.
11. Skyt J., Jensen C.S., and Mark L. A foundation for vacuuming temporal databases. Data Know. Eng., 44(1):1–29, 2003.
12. Zdonik S. Version management in an object-oriented database. In Proc. Int. Workshop on Adv. Programming Env., 1986, pp. 405–422.

Scientific Databases

AMARNATH GUPTA
University of California-San Diego, La Jolla, CA, USA

Definition

Scientific data refers to data that arise from scientific experiments, instruments, analytical tools, and computations. A chemistry experiment, for example, can yield data about the experimental setup, the pressure and temperature conditions under which the experiment was set up, measured variable like the heat released, initial and final masses the ingredients and products of the experiment, and so forth. The output of an instrument like a radio-telescope, after running signal processing algorithms, will produce "images" of the radio-frequency sources in a part of the sky that the telescope was looking at. A biologist, after obtaining the image of a dye-filled nerve cell, uses image analysis software to produce a set of measurements that reflect the structure of the cell and its subparts. Recently, environmental sensors are cast in oceans and send real-time data on ocean temperature, salinity, oxygen content and other parameters. A scientific database refers to an information management framework need to store, organize, index, query, analyze, maintain, and mine such heterogeneous scientific data.

Historical Background

Investigation in data management techniques for scientific data started with studying file organization principles that are tuned toward specific kinds of scientific data [5], where the problem explored was to develop a multi-query indexing scheme for scientific records. Discipline-specific systems with data retrieval capabilities were also developed. Coughran [2] describes a systems called Hydrosearch for worldwide hydrographic data from oceans that supported range queries like:

```
OCEAN = PACIFIC
OUTPUT = REPORT
LATHEM = N
LONHEM = W
LATDEG GEQ 31 AND LATDEG LEQ 33
LONDEG GEQ 121 AND LONDEG LEQ 123
MONTH GEQ 3 AND YEAR = 63 OR MONTH
LEQ 2
MXDPTH GEQ.98 DBOT
```

Around the same time, the National Laboratories dealing with a wide category of data related to energy research, recognized that "such diverse data applications as material compatibility, laser fusion, magnetic fusion, test, equation of state, weather, environmental and demographic data, has an acute need for a Scientific Data Base Management System (SDBMS)...The large volume of data, the numeric values within an epsilon of accuracy, the unknown data relationships, the changing requirements, coupled with the overall goal of extracting new intelligence from the raw data, dictate a data base system tailored toward scientific applications. Such an SDBMS should support scientific data types, a relational end user view, an interactive user language, interfaces to graphical and statistical packages, a programming language interface, interfaces to existing facilities, extensibility, portability, and use in a distributed environment" [1]. The system in [1] was developed on the CODASYL model [7].

It was in the 1980s that these different efforts started to take a coherent form where some general characteristics of scientific data management were identified. It was recognized through papers like [10] that unlike business data that is usually defined by a data schema and values conforming to the schema, scientific data can come with a measurement framework, a metadata specification, and often a summarization framework. In [11] the database requirements for scientific data were characterized, and a new conference started in 1986 devoted to the issues managing scientific and statistical data.

Foundations

As alluded to in the previous paragraph, scientific data are usually more than the values of attributes – they are often accompanied by additional descriptors which together specify the semantics of the data and therefore determines how the data can be interpreted for query and analytical operations. Some broad categories of these additional components that specify the context of the raw scientific data are described below.

Measurement Framework

A measurement framework is a specification of the setting of the experimental data. For time-series data, it may be the sampling frequency. For spatial data represented as a raster, it includes the resolution of the grid, and how the measurement is obtained per grid (e.g., once at the center of the grid, average of n measurements within the grid ...). For finite element data such as data from a fluid mechanics model, it may be the nature and regularity of the mesh over which data are recorded. The measurement framework is important in understanding the semantics of the data. If there are two raster data sets containing the measured temperature of two overlapping regions such that (i) one has a finer resolution than another, and (ii) one has the average-of-the-grid semantics, while the other has a single-sample-at-center semantics, how can one define a join operation to combine the two data sets? One cannot simply define a band join without considering a way to homogenize the data sets before they can be joined. Another aspect of the measurement framework is an assessment of the uncertainty associated with the measured, estimated or computed data. When data are associated with uncertainty, the traditional data models do not suffice – probabilistic or uncertainty-aware data models and query evaluation techniques are needed.

Metadata Framework

Metadata refers to descriptors that provide additional semantics beyond the value of an attribute. These include the unit of measurement, the precision and accuracy of the data value, the uncertainty associated with the temporal or spatial position at which the data are taken, the experimental setting including whether the data are absolute or relative to any other reference, what if any computational corrections should be made on the data before it can be delivered to an end user. Metadata also covers constraint statements that limit the allowable domain of data values, or additional conditions that must be satisfied for the data to be interpreted. These may range from simple encoding schemes that specify "out of range" or "unknown" values, to multi-attribute constraints like "data are valid only if the cloud cover coefficient at the location is less than 0.2." For data that are produced by computational algorithms (such as simulations of natural phenomena), the metadata also consists of the parameter settings of the algorithms, which must be taken into account to interpret, compare and analyze the data.

Summarization Framework

Scientific data are often voluminous due to high degree of sampling, or the total time or space over which data are acquired. In many applications, the total amount of data are too much and non-informative, and the scientists maintain only summarized versions of the data. For example, one may keep only weekly average temperature obtained from satellites; alternatively, one might keep the information only when there is a significant local change in the data. Since this is a common practice in many scientific disciplines, specification of how the data was summarized is a form of information that needs to go together with the data itself.

Heterogeneity of Types

An important characteristic of scientific data is the wide range of complexity and heterogeneity of the data types that are needed to model the applications.

Complexity and Heterogeneity of Formats

Distinct from the issue of data types, scientific data demonstrates a wide variety of formats for the same kind of data. In some domains, there is a lack of a single standard, and vendors of instruments that provide data, or vendors of software that manipulate the data define their own formats to facilitate their respective needs for data generation, analysis and visualization. For example, biological pathways, which are essentially graphs with node and edge attributes are represented differently by software such as Cytoscape [9], PATIKA [3], Pathway Studio [6] and the standardization effort called BioPAX (www.biopax.org) In other domains, there is more standardization. But the formats are very complex. HDF (Hierarchical Data Format) is a complex scientific data format for storing multidimensional data, raster data and tables. It is also designed to be self-describing and contains additional metadata. The multiplicity of supported models and the embedded metadata requires special data management tools [4] to be developed for indexing and querying HDF data. A consequence of this format heterogeneity and complexity is that interoperability of scientific data remains a research challenge.

Data Management Issues in Scientific Databases

Traditional Issues In 1985, [11] identified a number of data management issues that pertain to scientific data management. Many of these issues that hold equally well today are:

- *Data Volume and Compression*: Much of scientific data are multidimensional. While the data sets can be very large, the fraction of the multidimensional space that is occupied by the data is smaller. This brings up the need to compress the data as well as to choose a data organization that will exploit the sparsity of data. Further, data manipulation and query evaluation techniques that utilize the compressed data or a new data organization are needed. Managing large-scale scientific data is now considered to be an important challenge. A notable project in this area is led by the Stanford Linear Accelerator (SLAC – http://www.slac.stanford.edu/) that is attempting to build a data management system for a petabyte of data.
- *Data Structures*: With new scientific data types, there is a need for new data structures and access methods. In recent years, a number of index structures have been proposed for multidimensional data. For example, Zhang et al. [12] proposed a data structure for sampling multidimensional data; Rotem et al. [8] have developed bitmap indexes for very large-scale multidimensional data.
- *New Operations*: Data manipulation and search in scientific databases need operators that go beyond

traditional relational or tree manipulation algebras. It requires new operations such as sampling, neighborhood searching in metric space, estimation and interpolation operators for sampled data over a dense data space, novel join methods for complex data types.

- *Analysis Support*: The ultimate goal of scientific data acquisition and storage is some form of analysis and derivation of scientific truth. Scientists, the primary users of the database, are not often willing to learn complex query languages – instead, they want to query the data as part of their analytical tasks. The management of the entire analysis process require analysis-friendly user interfaces that are sufficiently expressive but not overly complex, a way to facilitate repeated use of the same query with differing parameters, management of long-running queries, handling large volumes of intermediate data, and optimal execution of an entire analytical workflow.
- *Quality Management*: Quality awareness of data is important when the data collected in a database comes from any error prone process. For scientific data, errors and approximations arise often due to factors like resolution limits of instruments, malfunctioning of devices, unforeseen environmental or experimental confounding factors, biases introduced by sampling, approximations used by preprocessing computations and, of course, human error. The problem gets compounded when a data product is derived from an existing data product. In many cases, data for a given application may come from data sources with different quality and "believability." Query languages, evaluation techniques and analysis for scientific data needs to be quality-aware, and give a user the ability to filter data based on its quality and integrity.

Recent Trends More recently, the scientific data management community has identified newer challenges over and above the issues above. Some of them are:

- *Annotation Management*: Annotating data is a common practice in science. An annotation is a piece of user-imposed data that references an arbitrary data element in an existing data store. One can annotate a block of data with a statement about its quality; one may annotate a fragment of data with information of its provenance (i.e., where the data was obtained from and how it was transformed before it appeared in its present form); one may annotate data by tagging it with keywords or terms from an ontology so that it can be easily related to other data. Annotation management attempts to create a uniform way to store the annotation and their referent data so that both the primary data and the annotations can be queried together.
- *Semantics*: In many domain sciences, the semantics of data is not adequately represented in data repositories. This makes it very difficult to one user to interpret data from another user, and even harder to combine multiple kinds of data together. This recognition has led to a renewed interest in developing semantic data models for scientific data. As part of this effort ontologies are being created to standardize and define the terms and the inter-term relationships in a discipline using standards like the Web Ontology Language, OWL, so that data producers can either use the ontological terms to represent their data, or map their existing data to the ontologies. At the same time, efforts are underway to develop query, integration and data mining techniques that make use of the semantic framework.

Cross-references

▶ Annotation Management
▶ Data Types in Scientific Data Management

Recommended Reading

1. Birss E.W., Jones S.E., Ries D.R., and Yeh J.W. Scientific data base management at Lawrence Livermore Laboratory: needs and a prototype system. Technical Report UCRL-80146; CONF-771062–1, Lawrence Livermore Lab, California University, 1977.
2. Coughran E. HYDROSEARCH, an easy-to-use retrieval system for hydrographic station data. OCEANS, 7:418–421, 1975.
3. Demir E., Babur O., Dogrusoz U., Gursoy A., Nisanci G., Cetin-Atalay R., and Ozturk M. PATIKA: an integrated visual environment for collaborative construction and analysis of cellular pathways. Bioinformatics, 18(7):996–1003, 2002.
4. Gosink L., Shalf J., Stockinger K., Wu K., and Bethel W. HDF5-FastQuery: accelerating complex queries on HDF datasets using fast bitmap indices. In Proc. 18th Int. Conf. on Scientific and Statistical Database Management, 2006, pp. 149–158.
5. Ikeda H. and Naito M. Evaluation of a combinatorial file organization scheme of order one. In Proc Study on Scientific Database Management Systems, 1979, pp. 195–199 (in Japanese).
6. Nikitin A., Egorov S., Daraselia N., and Mazo I. Pathway studio – the analysis and navigation of molecular networks. Bioinformatics, 19(16):2155–7, 2003.

7. Olle T.W. The Codasyl Approach to Data Base Management. Wiley, Chichester, UK, 1978.
8. Rotem D., Stockinger K., and Wu K. Minimizing I/O costs of multi-dimensional queries with bitmap indices. In Proc. 18th Int. Conf. on Scientific and Statistical Database Management, 2006, pp. 33–44.
9. Shannon P., Markiel A., Ozier O., Baliga N.S., Wang J.T., Ramage D., Amin N., Schwikowski B., and Ideker T. Cytoscape: a software environment for integrated models of biomolecular interaction networks. Genome Res., 13(11):2498–2504, 2003.
10. Shoshani A., Olken F., and Wong H.K.T. Characteristics of scientific databases. In Proc. 10th Int. Conf. on Very Large Data Bases, 1984, pp. 147–160.
11. Shoshani A. and Wong H.K.T. Statistical and scientific database issues. IEEE Trans. Softw. Eng., 11(10):1040–1047, 1985.
12. Zhang X., Kurc T., Saltz J., and Parthasarathy S. Design and analysis of a multi-dimensional data sampling service for large scale data analysis applications. In Proc. 20th Int. Parallel and Distributed Processing Symp., 2006.

Scientific Knowledge Bases

▶ Biomedical Scientific Textual Data Types and Processing

Scientific Medicine

▶ Evidence Based Medicine

Scientific Query Languages

▶ Query Languages for the Life Sciences

Scientific Visualization

Ronald Peikert
ETH Zurich, Zurich, Switzerland

Definition

Scientific visualization [1] provides graphical representations of numerical data for their qualitative and quantitative analysis. In contrast to a fully automatic analysis (e.g., with statistical methods), the final analytic step is left to the user, thus utilizing the power of the human visual system. Scientific visualization differs from the related field of information visualization in that it focuses on data that represent samples of continuous functions of space and time, as opposed to data that are inherently discrete.

The challenge in scientific visualization is to cope with massive data, which cannot be presented to the user in an unprocessed way for several reasons:

1. Volumetric data, i.e., data given on a three-dimensional domain, occlude each other. This problem becomes even more severe if data are not scalars, but vectors or even tensors.
2. Visualization should provide a global picture of the spatial and temporal behavior of the data, but also allow for interactive exploration of details.
3. There can be multiple data (different physical quantities, multiple data channels, etc.) at each point in the domain.
4. Visualization of scientific data should also include visualization of their uncertainty.
5. The amount of raw data often exceeds limitations of processor speed, transfer rates, memory size, and display resolution.

Applications of scientific visualization cover a wide spectrum of science and engineering disciplines. Currently, some of the most active fields are medical and biomedical image data, simulation and measurement data from fluid or solid mechanics, molecular data, data from geology and geophysics, astronomy, weather and climate.

Key Points

Scientific visualization evolved in the 1980s from earlier graphing techniques when 3D computer graphics opened new ways of displaying numerical data. The abstraction from the application domain led to interdisciplinary visualization software systems with a modular dataflow architecture. This approach is still successful [2], since for many visualization tasks, the semantics of the data is less relevant than mathematical properties such as the discretization type or the categorization into scalars, vectors, and tensors.

For volumetric scalar data, important visualization techniques are isosurfaces and direct volume rendering. For vector fields examples are arrow glyphs, integral lines (streamlines, streaklines) and texture advection. Tensor fields are visualized with glyphs (ellipsoids, superquadrics) or tensor lines. In the special case of diffusion tensor MRI data, fiber tracking techniques are applied. Scalar, vector, and tensor fields all are amenable to topology-based visualization, which provides both the singularities and a segmentation of the domain into regions of "similar data behavior." More general, feature extraction and feature tracking techniques aim at reducing the data complexity and providing the viewer with only the most salient information. Features (e.g., edges, ridges, flow structures) are typically defined in terms of data and their derivatives. User-defined feature definitions are possible in visualization systems built on the linked views paradigm, where simultaneous views of both physical and data space are available all of which allow for interactions such as data coloring and subsetting. The visualization of very large data requires optimization techniques including multi-resolution, parallel and out-of-core algorithms, as well as view-dependent visualization.

Cross-references

- ▶ Data Visualization
- ▶ Visualization Pipeline

Recommended Reading

1. Hansen C.D. and Johnson C.R. (eds.). Visualization Handbook. Academic Press, San Diego, CA, 2004.
2. Schroeder W., Martin K., and Lorensen B. The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics Kitware, Inc., New York, 2006.

Scientific Workflows

Bertram Ludäscher, Shawn Bowers,
Timothy McPhillips
University of California-Davis, Davis, CA, USA

Synonyms

In silico experiment; Grid workflow

Definition

A *scientific workflow* is the description of a process for accomplishing a scientific objective, usually expressed in terms of *tasks* and their *dependencies*. Typically, scientific workflow tasks are computational steps for scientific simulations or data analysis steps. Common elements or stages in scientific workflows are acquisition, integration, reduction, visualization, and publication (e.g., in a shared database) of scientific data. The tasks of a scientific workflow are organized (at design time) and orchestrated (at runtime) according to dataflow and possibly other dependencies as specified by the workflow designer. Workflows can be designed visually, e.g., using block diagrams, or textually using a domain-specific language.

Historical Background

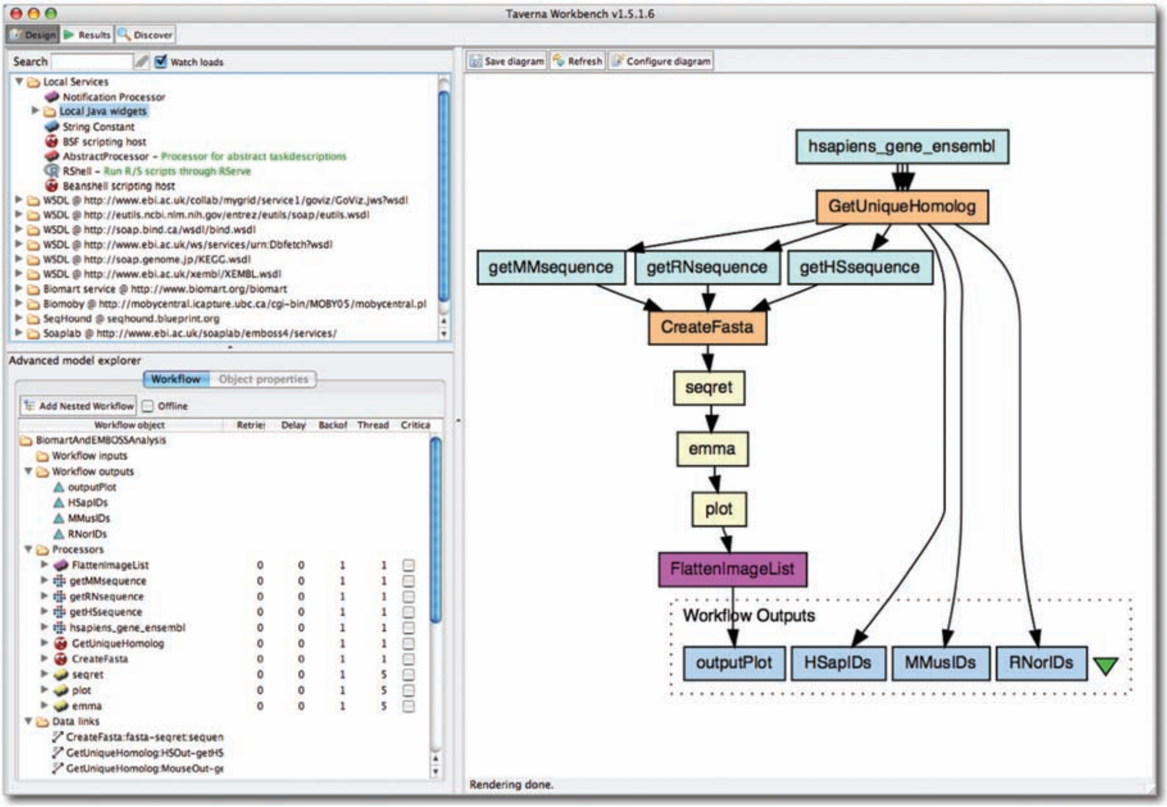
Workflows have a long history in the database community and in business process modeling, in which case they are sometimes called *business workflows* to distinguish them from scientific workflows. The database community realized early [10] that scientific data management has different characteristics from more traditional business data management. Early work on scientific workflows within the database community took a database-centric view by defining data models and query languages suitable for scientific experiment management systems. The MOOSE data model and FOX query language have their roots in the late 1980's [5] and early 1990's [13] and gave rise to the ZOO experiment management environment [6], an early system based on an underlying object-oriented database. Another pioneering work that emphasized the importance of workflow concepts in scientific data management is WASA, a *Workflow-based Architecture for Scientific Applications* [8]; the related publication [12] introduced the term "scientific workflow" and contrasted such workflows with office automation and business workflows. An early benchmark comparing different database architectures for scientific workflow applications is LabFlow-1 [1].

Other roots of scientific workflow systems include *problem solving environments*, which emerged in the nineties in the computational sciences community as intuitive tools to "solve a target class of problems for scientific computing" [4], and *laboratory information management systems* (LIMS) [9], which can be seen as special scientific workflow systems that are used in a laboratory environment for the management of samples, instrument-based measurements, and other functions, including data analysis and workflow automation. Similar to many scientific workflow systems, problem solving environments and LIMS sometimes employ a visual programming paradigm to link together components. An early, if not the first, visual language that allowed simple interfacing with lab instruments was G in LabVIEW1.0, released in 1986 for the Apple Macintosh. Modern incarnations of LIMS can include functions of enterprise resource planning (ERP) systems and thus go beyond the scope of current scientific workflow systems.

With the advent of *e-Science* as a paradigm, scientific workflow research and development has seen a major resurgence. Similar to the related term *cyberinfrastructure*, e-Science brings together computational techniques and tools from the computational sciences, distributed and high-performance computing, databases, data analysis, visualization, sharing, and collaboration. There are now a number of new open source as well as commercial scientific workflow systems available and under active development. For example, a special journal issue of *Concurrency and Computation: Practice and Experience* covers a number of systems, including Kepler, Taverna, and Triana among others [2]. For a high-level overview and attempt at a classification of current scientific workflow systems see [14], which includes also references to many other systems, such as Askalon, Pegasus/DAGMan, Karajan, etc.

Foundations

Science is an exploratory process involving cycles of observation, hypothesis formation, experiment design and execution. Today, scientific knowledge discovery is increasingly driven by data analysis and computational methods, e.g., due to ever more powerful instruments for observation and the use of commodity clusters for high-performance scientific computing and simulations in the computational sciences. Scientific workflows can be applied during various phases of the larger science process, specifically modeling and automation of computational experiments, data analysis, and data management. The results from workflow runs can yield new data and insights and thus may lead to affirmation,



- Local Services
  - Notification Processor
  - Local Java widgets
  - String Constant
  - BSF scripting host
  - AbstractProcessor – Processor for abstract taskdescriptions
  - RShell – Run R/S scripts through RServe
  - Beanshell scripting host
- WSDL @ http://www.ebi.ac.uk/collab/mygrid/service1/goviz/GoViz.jws?wsdl
- WSDL @ http://eutils.ncbi.nlm.nih.gov/entrez/eutils/soap/eutils.wsdl
- WSDL @ http://soap.bind.ca/wsdl/bind.wsdl
- WSDL @ http://www.ebi.ac.uk/ws/services/urn:Dbfetch?wsdl
- WSDL @ http://soap.genome.jp/KEGG.wsdl
- WSDL @ http://www.ebi.ac.uk/xembl/XEMBL.wsdl
- Biomart service @ http://www.biomart.org/biomart
- Biomoby @ http://mobycentral.icapture.ubc.ca/cgi-bin/MOBY05/mobycentral.pl
- SeqHound @ seqhound.blueprint.org
- Soaplab @ http://www.ebi.ac.uk/soaplab/emboss4/services/

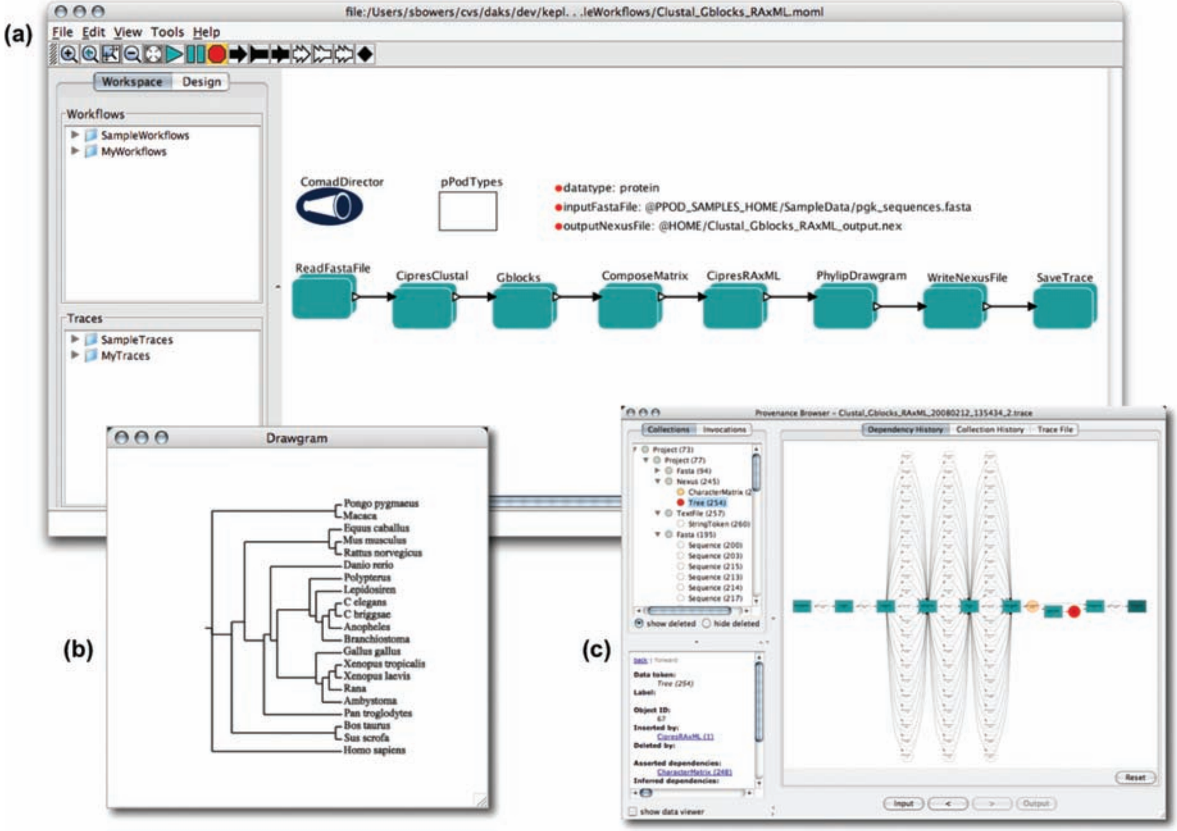
Workflow object	Retrie	Delay	Backof	Thread	Critica
BiomartAndEMBOSSAnalysis					
Workflow inputs					
Workflow outputs					
outputPlot					
HSapIDs					
MMusIDs					
RNorIDs					
Processors					
FlattenImageList	0	0	1	1	□
getMMsequence	0	0	1	1	□
getRNsequence	0	0	1	1	□
getHSsequence	0	0	1	1	□
hsapiens_gene_ensembl	0	0	1	1	□
GetUniqueHomolog	0	0	1	1	□
CreateFasta	0	0	1	1	□
seqret	0	0	1	5	□
plot	0	0	1	5	□
emma	0	0	1	5	□
Data links					
CreateFasta:fasta-seqret:sequen					
GetUniqueHomolog:HSOut-getHS					
GetUniqueHomolog:MouseOut-ge					

Scientific Workflows. Figure 1. Example workflow represented in the Taverna workflow system. This workflow extracts gene IDs from human chromosome 22 with mappings to disease functions and homologues in mouse and rat; fetches base pairs of the associated DNA sequences; combines the sequences into a FASTA file; performs a multiple sequence alignment; and renders the result. The workflow uses three soaplab-based analysis operations (seqret, emma, plot) that run on the EBI compute cluster.

modification, or refutation of a given hypothesis or experiment outcome.

Scientific workflow systems automate the execution of scientific workflows, and may additionally assist in workflow design, composition, and the management and sharing of workflow descriptions. Other important functions include support for workflow execution monitoring, for recording and querying provenance information, for workflow optimization (e.g., exploiting dataflow and concurrency information for parallel execution), and for fault-tolerant execution. These additional features also distinguish a scientific workflow systems approach from more traditional script-based solutions in which such functionality is usually not provided. Workflow provenance information can be used, e.g., to facilitate the interpretation, debugging, and reproducibility of scientific analyses. An increasing number of scientific workflow systems now offer support for various forms of provenance. One can distinguish *data provenance*, i.e., the processing history of data, and provenance information describing the *workflow evolution*, i.e., the history of changes of a workflow definition and the parameter settings used for a particular workflow instance.

Scientific workflows are often visually represented as directed graphs (Figs. 1 and 2) linking atomic tasks or composite components, so-called *subworkflows.* Tasks can include native functions of the workflow system, but often correspond to invocations of localapplications, remote (web) services, or subworkflows. Scientific workflows differ from conventional programming in that the workflows are often more coarse-grained and involve wiring together of pre-existing components and specialized algorithms. Figure 1



●datatype: protein
●inputFastaFile: @PPOD_SAMPLES_HOME/SampleData/pgk_sequences.fasta
●outputNexusFile: @HOME/Clustal_Gblocks_RAxML_output.nex

◉ show deleted ○ hide deleted

Data token:
Tree (254)
Label:
Object ID:
67
Inserted by:
CipresRAxML (1)
Deleted by:
Asserted dependencies:
CharacterMatrix (248)
Inferred dependencies:

□ show data viewer

Scientific Workflows. Figure 2. Example scientific workflow in the Kepler system: (a) user interface for creating, editing, and executing scientific workflows; (b) a visual representation of the data product (a phylogenetic tree) computed by a workflow run; and (c) a viewer for navigating the data provenance (lineage) captured in an execution trace. This workflow uses a combination of local and remote (web) services to perform multiple sequence alignment and phylogenetic tree inference on input DNA sequences.

shows a simple bioinformatics workflow in the Taverna system, consisting of multiple (soaplab) services.

There is currently no standard *scientific workflow language*, and standards from related communities (e.g., BPEL4WS) have not found widespread adoption in the scientific workflow community. For example, job-based *grid workflows* are often represented as directed acyclic graphs (DAGs), which are then scheduled on a computational grid or cluster computer according to the implied task dependencies. In this *model of computation*, each task is executed only once per workflow run and task scheduling amounts to finding a *topological sort* for the partial order implied by the DAG. Other more sophisticated models of computation consider tasks as independent and continuously executing processes which can receive and send many different data items per workflow run. Scientific workflow systems that support such models of computation may thus be used for *data stream processing* and *continuous queries*. Similar to business workflows, formal approaches such as *Petri nets* can be used to describe scientific workflow execution semantics. However, the dataflow models of computation of many scientific workflow systems can exhibit both task- and pipeline-parallelism where token order is important. A standard computation model for such dataflow systems is the *Kahn Process Network* model. The structurally simple linear Kepler workflow in Fig. 2 is achieved via a special model of execution, implemented by a so-called *director*. (Kepler inherits from the underlying Ptolemy II system the capability to use distinct directors at different workflow modeling levels and thus to combine different models of computation in a single workflow.) The COMAD (*Collection-Oriented Modeling And Design*) director in Fig. 2 specifies that workflow components work on a continuous, XML-like data stream which passes through all components eventually. Each component is configurable to compute only on certain (tagged) data collections. Results are injected back into the stream. The resulting more linear workflows are easy to comprehend and evolve over time, another important advantage over script-based solutions.

Key Applications

Scientific workflows now span virtually all areas of the natural sciences. Bioinformatics is a particularly active application area (cf. Figs. 1 and 2), but the spectrum of disciplines employing scientific workflow systems is much wider and includes particle physics, chemistry, neurosciences, ecology, geosciences, oceanography, atmospheric sciences, astronomy and cosmology, among others.

URL to Code

A number of open source scientific workflow systems are available, among them:

Kepler: `http://www.kepler-project.org`
Taverna: `http://taverna.sourceforge.net`
Triana: `http://www.trianacode.org`

For a list including many other systems, see `http://www.extreme.indiana.edu/swf-survey/`.

Cross-references

- ▶ Business Process Modeling
- ▶ (Business) Workflow
- ▶ Data Analysis
- ▶ Dataflow
- ▶ Problem Solving Environment
- ▶ Provenance
- ▶ Visual Programming

Recommended Reading

1. Bonner A.J., Shrufi A., and Rozen S. LabFlow-1: a database benchmark for high-throughput workflow management. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996, pp. 463–478,
2. Fox G.C. and Gannon D. (eds.) Concurrency and Computation: Practice and Experience. Special Issue: Workflow in Grid Systems, 18(10), 2006.
3. Gil Y., Deelman W., Ellisman W., Fahringer T., Fox G., Gannon D., Goble C., Livny M., Moreau L., and Myers J. Examining the challenges of scientific workflows. Computer, 40(12): 24–32, 2007.
4. Houstis E., Gallopoulos E., Bramley R., and Rice J. Problem-solving environments for computational science. IEEE Comput. Sci. Eng., 4(3):18–21, 1997.
5. Ioannidis Y.E. and Livny M. MOOSE: modeling objects in a simulation environment. In IFIP Congress, G.X. Ritter (ed.). North-Holland, 1989, pp. 821–826.
6. Ioannidis Y.E., Livny M., Gupta S., and Ponnekanti N. ZOO: a desktop experiment management environment. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 274–285.
7. Ludäscher B. and Goble C. (eds.) ACM SIGMOD Rec., 34(3): 44–49, September 2005. Special Issue on Scientific Workflows.
8. Medeiros C.B., Vossen G., and Weske M. WASA: a workflow-based architecture to support scientific database applications. In Proc. 6th Int. Conf. Database and Expert Syst. Appl., 1995, pp. 574–583.
9. Nakagawa A.S. LIMS: Implementation and Management. The Royal Society of Chemistry, Thomas Graham House. The Science Park Cambridge CB4 4WF, 1994.

10. Shoshani A., Olken F., and Wong H.K.T. Characteristics of scientific databases. In Proc. 10th Int. Conf. on Very Large Data Bases, 1984, pp. 147–160.
11. Taylor I., Deelman E., Gannon D., and Shields M. (eds.) Workflows for e-Science: Scientific Workflows for Grids. Springer, Berlin, 2007.
12. Wainer J., Weske M., Vossen G., and Medeiros C.B. Scientific workflow systems. In Proc. NSF Workshop on Workflow and Process Automation in Information Systems: State of the Art and Future Directions, 1996.
13. Wiener J.L. and Ioannidis Y.E. A moose and a fox can aid scientists with data management problems. In Proc. Fourth Int. Workshop on Database Programming Languages, 1993, pp. 376–398.
14. Yu J. and Buyya R. A taxonomy of scientific workflow systems for grid computing. ACM SIGMOD Rec., 34(3):44–49, September 2005. Special Issue on Scientific Workflows.

Score Propagation

► Propagation-Based Structured Text Retrieval

Screen Scraper

Harald Naumann
Vienna University of Technology, Vienna, Austria

Synonyms

Screen scraping; Data extraction; Screen wrapper

Definition

A screen scraper is a program which extracts relevant data from the visual user interface of an application. Input data are commonly represented using text-only or graphically enhanced tables, lists and forms, tailored to a human audience. Scraping is the task of collecting data from its presentation, not directly from its source for lack of access. The scraper output has a structured and machine-readable format, where extracted data are usually annotated with its semantics (metadata), suitable for automatic post-processing. The process can be thought of as reverse-engineering a data store from its presentation, abstracting content from layout. Using this approach, application data are taken from the human-oriented screen output rather than the application's hidden proprietary data structures.

Key Points

Traditionally, screen scrapers have been used to interface legacy systems residing on old mainframes, which often host critical data processing applications. Although both hardware and software are obsolete, they cannot be replaced for various reasons. Screen scraping offers a cost-effective alternative to access and leverage underlying data stores. Typical applications include capturing emulated IBM 3270 screens (a widely used text-based protocol for dumb terminals). Combined with macros to enable navigation throughout different screens, scrapers can be used to integrate with modern architectures.

Common text scraping methods make heavy use of syntactic tools such as regular expressions to identify relevant data. Recently, more semantic approaches have been researched that furthermore allow scraping from unstructured documents such as PDF using generic document understanding techniques supplemented by domain-specific knowledge modeled with ontologies. Layout and table recognition can be performed on a visual level using top-down segmentation (recursive X-Y cut), bottom-up clustering as well as probabilistic graph-matching algorithms [1]. Identified document segments can then be classified using semantically designed rules in order to annotate the original document with its implicit structure.

Another key area is web scraping, which locates data by exploiting the explicit underlying layout markup (HTML) of its presentation. As a means to build interfaces (APIs) for web sites not available otherwise, scrapers also serve as the basis for state-of-the art semantic web applications called web mashups, such as MIT's SIMILE project [2]. Web scrapers filter relevant content, serializing it in annotated XML format. The main complexity issue arising with all scraper types is coping with change: scrapers are said to "break," when data presentation changes substantially. Visual IDEs can assist scraper design, offering lower maintenance effort compared to purely programmatical solutions.

Cross-references

► Information Extraction
► Web Data Extraction
► Wrapper Generator
► Wrapper Maintenance

Recommended Reading

1. Hassan T. and Baumgartner R. Intelligent text extraction from PDF documents. In Proc. Int. Conf. on Intelligent Agents, Web Technologies and Internet Commerce, 2005, pp. 2–6.
2. Huynh D., Mazzocchi S., and Karger D. Piggy bank: experience the semantic web inside your web browser. In the Fourth Int. Semantic Web Conf., 2005.

Screen Scraping

▶ Languages for Web Data Extraction
▶ Screen Scraper

Screen Wrapper

▶ Screen Scraper

SCSI Target

Kaladhar Voruganti
Network Appliance, Sunnyvale, CA, USA

Definition

In SCSI protocol, the server which provides the storage is known as target. There can be multiple targets in a storage controller. Each target can offer access to either a single volume or multiple volumes. The volumes being offered by a storage target are mapped into LUNs by the host operating system.

Key Points

Storage controllers, JBOD (just a bunch of disks in an enclosure), direct attached disks, and storage virtualization boxes can all act as SCSI targets. Other types of storage media that can support the SCSI protocol can also act as SCSI targets. The transport protocol encapsulating the SCSI commands dictate the uniqueness of the SCSI target and initiator identifiers.

Cross-references

▶ LUN
▶ LUN Mapping
▶ Storage Protocols
▶ Volume

SDI, Selective Dissemination of Information

▶ Information Filtering

Search Advertising

▶ Web Advertising

Search Engine Caching and Prefetching

▶ Web Search Result Caching and Prefetching

SDC Score

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona, Catalonia

Definition

Statistical disclosure control (SDC) methods for microdata can be ranked based on information loss, disclosure risk or a combination of both. An SDC score is a combination of information loss and disclosure risk measures used to rank methods.

Key Points

The construction of an SDC score combining information loss and disclosure risk was first proposed in [1,2]. For each method M and parameterization P, the following score is computed:

$$Score(\mathbf{V},\mathbf{V}') = \frac{IL(\mathbf{V},\mathbf{V}') + DR(\mathbf{V},\mathbf{V}')}{2}$$

where IL is an information loss measure, DR is a disclosure risk measure and $\mathbf{V}'$ is the protected dataset obtained after applying method M with parameterization P to an original dataset $\mathbf{V}$.

In the above references, IL and DR were computed using a weighted combination of several information loss and disclosure risk measures. With the resulting score, a ranking of a set of masking methods (and their parameterizations) was obtained. Yancey et al. [3] later followed the same approach to rank a different set of methods using a slightly different score.

To illustrate how a score can be constructed, the particular score used by [2] is next described. Let X and X' be matrices representing original and protected datasets, respectively, where all attributes are numerical. Let V and R be the covariance matrix and the correlation matrix of X, respectively; let $\bar{X}$ be the vector of attribute averages for X and let S be the diagonal of V. Define V', R', $\bar{X}'$, and S' analogously from X'. The Information Loss (IL) is computed by averaging the mean variations of $X - X'$, $\bar{X} - \bar{X}'$, $V - V'$, $S - S'$, and the mean absolute error of $R - R'$ and multiplying the resulting average by 100. Thus, the following expression is obtained for information loss:

$$IL = \frac{100}{5}\left(\frac{\sum_{j=1}^{p}\sum_{i=1}^{n}\frac{|x_{ij}-x'_{ij}|}{|x_{ij}|}}{np} + \frac{\sum_{j=1}^{p}\frac{|\bar{x}_j-\bar{x}'_j|}{|\bar{x}_j|}}{p} + \frac{\sum_{j=1}^{p}\sum_{1\le i\le j}\frac{|v_{ij}-v'_{ij}|}{|v_{ij}|}}{\frac{p(p+1)}{2}} + \frac{\sum_{j=1}^{p}\frac{|v_{jj}-v'_{jj}|}{|v_{jj}|}}{p} + \frac{\sum_{j=1}^{p}\sum_{1\le i\le j}|r_{ij} - r'_{ij}|}{\frac{p(p-1)}{2}}\right)$$

The expression of the overall score is obtained by combining information loss and information risk as follows:

$$Score = \frac{IL + \frac{(0.5DLD+0.5PLD)+ID}{2}}{2}$$

Here, DLD (Distance Linkage Disclosure risk) is the percentage of correctly linked records using distance-based record linkage, PLD (Probabilistic Linkage Record Disclosure risk) is the percentage of correctly linked records using probabilistic linkage, ID (Interval Disclosure) is the percentage of original records falling in the intervals around their corresponding masked values and IL is the information loss measure defined above.

Based on the above score, it turned out that, for the benchmark datasets and the intruder's external information they used in [2], two good performers among the set of methods and parameterizations they tried were: (i) rankswapping with parameter p around 15; (ii) multivariate microaggregation on unprojected data taking groups of three attributes at a time.

Cross-references

▶ Disclosure Risk
▶ Inference Control in Statistical Databases
▶ Information Loss Measures
▶ Microaggregation
▶ Microdata
▶ Rank Swapping
▶ Record Matching

Recommended Reading

1. Domingo-Ferrer J., Mateo-Sanz J.M., and Torra V. Comparing SDC methods for microdata on the basis of information loss and disclosure risk. In Pre-proceedings of ETK-NTTS'2001, 2001, pp. 807–826.
2. Domingo-Ferrer J. and Torra V. A quantitative comparison of disclosure control methods for microdata. In P. Doyle, J.I. Lane, J.J.M. Theeuwes, and L. Zayatz (eds.). Confidentiality, Disclosure and Data Access: Theory and Practical Applications for Statistical Agencies. North-Holland, Amsterdam, 2001, pp. 111–134.
3. Yancey W.E., Winkler W.E., and Creecy R.H. Disclosure risk assessment in perturbative microdata protection. In J. Domingo-Ferrer (ed.). Inference Control in Statistical Databases. LNCS, Vol. 2316. Springer, 2002, pp. 135–152.

Search Engine Metrics

BEN CARTERETTE
University of Massachusetts Amherst, Amherst,
MA, USA

Synonyms

Evaluation measures; Performance measures

Definition

Search engine metrics measure the ability of an information retrieval system (such as a web search engine) to retrieve and rank relevant material in response to a user's query. In contrast to database retrieval, relevance in information retrieval depends on the natural language semantics of the query and document, and search engines can and do retrieve results that are not relevant. The two fundamental metrics are *recall*, measuring the ability of a search engine to find the relevant material in the index, and *precision*, measuring its ability to place that relevant material high in the ranking. Precision and recall have been extended and adapted to many different types of evaluation and task, but remain the core of performance measurement.

Historical Background

Performance measurement of information retrieval systems began with Cleverdon and Mills in the early 1960s with the Cranfield tests of language indexing devices [4,3]. Prior to that, retrieval systems had been measured primarily by their efficiency; as with databases, it was implicitly assumed that any document matching the query was relevant. Cleverdon and Mills recognized that information retrieval is not like database retrieval. Queries can be under- or over-specified, polysemy can confound the relationship between query and document, the wrong word can be chosen for a concept with many names, and so on. Results that are not relevant to the user's request will be returned and results that are relevant will not be returned; there is a need to measure how often this can be expected to happen in general.

Cleverdon and Mills identified two primary dimensions on which to evaluate performance: the proportion of relevant material retrieved (the *recall ratio*) and the proportion of retrieved material that is relevant (originally the *relevance ratio*, later *precision*). Part of the goal of the Cranfield tests was to measure how different indexing strategies affected recall and precision. To this end, Cleverdon and Mills assembled a collection of 1,100 papers in high speed aerodynamics and asked the authors to list the research questions that inspired the paper. Each of the cited references was then judged for relevance to each of the questions. The resulting set of data – a collection of documents, a set of questions or queries, and judgments of the relevance of each document to each query – is called a *test collection*, and the use of test collections for information retrieval evaluation is now referred to as the Cranfield methodology.

Through his extensive evaluations of the SMART retrieval system in the 1960's and 1970's using the Cranfield collection and methodology, Gerald Salton cemented precision and recall as the primary evaluation metrics for search engine performance [8]. He additionally offered extensions and refinements to these basic measures: normalized precision and recall, precision-recall curves to demonstrate the tradeoff between the two, interpolated precision at standard levels of recall, average precision over different levels of recall or different queries.

The early 1990s saw the formation of the Text REtrieval Conference (TREC) and the first evaluations over hundreds of thousands of full-text documents rather than the tens of thousands of abstracts that had previously been the standard in research [12]. The TREC collections are large and heterogeneous, and thus are a prime proving ground for any automatic retrieval technique. However, with an order-of-magnitudes increase in the number of documents, it became impossible to know every relevant document, and thus to know recall with certainty. TREC also motivated the birth of a field of research on "meta-evaluation," the evaluation of performance measures themselves, the evaluation of test collections, and the estimation of retrieval measures.

TREC also led the way in defining and providing models for the evaluation of new retrieval tasks. Some of the tasks studied and evaluated at TREC over the years include routing, multimedia retrieval, cross-language retrieval, and passage retrieval. Closely related to TREC are conferences on machine translation, summarization, and document understanding. Precision- and recall-based metrics such as the BLEU score [5] have become standard in these fields as well.

The 1990's also saw the explosive growth of the web, which over the past 15 years has grown into a collection of billions of documents, and within which search is a multi-million dollar industry. Accurate measures of performance are more important than ever, as millions of dollars are at stake when decisions are made based on those measures.

Foundations

Automatic text retrieval systems such as web search engines return results in the form of a ranked list, with the documents most likely to be relevant to the user's request at the top. Bad results can be returned if a query is over- or under-specified, a word with multiple meanings included in the query, or a word chosen to represent a concept that is in the index but not under that word, the ranked list will be "polluted" with nonrelevant results. To understand the extent to which this happens and how to fix it, it is necessary to evaluate the ability of the system to retrieve and rank relevant material independent of other factors affecting the utility of the search engine such as interface design or response time.

The two primary dimensions on which to evaluate a ranked list are its ability to find the indexed relevant material (recall) and its ability to rank that relevant material highly (precision). Formally, precision and recall are defined for a given rank cut-off in terms of binary relevance – each document is either relevant or not. Considering everything above the cut-off to be "retrieved" and everything below it to be "not retrieved" and comparing to the relevance of each document produces a 2 × 2 contingency table, as shown in Fig. 1.

Precision at rank n is defined as the proportion of relevant documents in the top n retrieved:

$$\text{precision@n} = \frac{\begin{array}{c}\text{number of documents relevant \&}\\ \text{retrieved in the top } n\end{array}}{\text{number retrieved}}$$

Recall at rank n is the proportion of all relevant documents in the index retrieved in the top n:

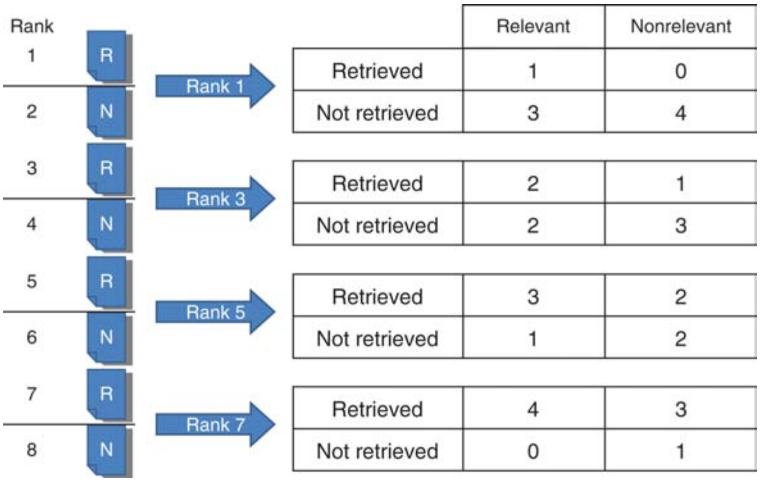
$$\text{recall@n} = \frac{\begin{array}{c}\text{number of documents relevant \&}\\ \text{retrieved in the top } n\end{array}}{\text{total number relevant}}$$

this means that the engine was able to find every relevant document without ever confusing a nonrelevant document for relevant.

As Fig. 2 shows, precision-recall curves appear jagged, as each new relevant document increases both precision and recall. The curve can be smoothed into a non-increasing curve by *interpolating* precision: k equally-spaced points of recall are chosen, and the interpolated precision at the ith point is defined as the highest precision at any point of recall greater than or equal to that point [7]. The interpolated curve demonstrates the trade-off between recall and precision: retrieving more documents increases recall, but it also brings more nonrelevant material into the ranking, decreasing precision. Figure 2 shows an example of an 11-point (recall = 0,0.1,...,1) interpolated curve.

Besides precision and recall (which readers may also know as "positive predictive value" and "sensitivity" respectively) there are many other statistics that can be calculated on a 2 × 2 contingency table as in Fig. 1: specificity, χ^2, mutual information, and accuracy, among others; besides the precision-recall curve, there are other curves that can be plotted over varying cut-off values, the ROC curve being the most famous. The utility of these to information retrieval is limited: they depend on counts of "true negatives," i.e., nonrelevant documents that were not retrieved. When retrieving documents over a large heterogeneous collection such as the web, nonretrieved nonrelevant documents make up the vast majority of indexed pages – to a close approximation, 100% of the index. Thus the difference in one of these statistics for any two rankings is negligible, and certainly not distinguishable from chance.

A number of statistics that summarize the precision-recall curve have been invented over the years. The most common is *average precision*, the area under the precision-recall curve (originally the interpolated curve, now more commonly the non-interpolated curve).



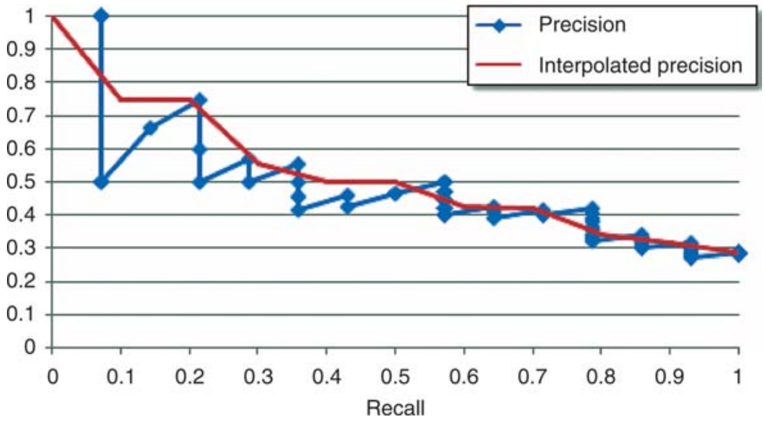
	Relevant	Nonrelevant
Retrieved	1	0
Not retrieved	3	4

Retrieved	2	1
Not retrieved	2	3

Retrieved	3	2
Not retrieved	1	2

Retrieved	4	3
Not retrieved	0	1

Search Engine Metrics. Figure 1. A ranking of eight documents, four of which have been judged relevant (**R**) and four nonrelevant (**N**). Cut-offs at ranks one, three, five, and seven produce the 2 × 2 tables shown.



Search Engine Metrics. Figure 2. An example precision-recall curve, along with its 11-point interpolated curve.

Another is *R-precision*, the point at which recall and precision are equal. These are both measures that reward systems for having both high recall and high precision, but in a nonlinear fashion. The F-measure is more linear: it is the weighted harmonic mean of precision and recall (weights are chosen depending on the relative importance of precision vs. recall); max-F is the highest of all such values.

Additionally, there are a number of other metrics in the literature that are based on the fundamental ideas behind precision and recall. One that has found widespread use in web measurement is *discounted cumulative gain* (DCG) [5]. Since it can handle graded relevance judgments, it is more flexible than precision as traditionally defined. *Normalized DCG* (NDCG) incorporates a recall component into DCG by dividing it by the best possible DCG for the query.

Relevance Judgments

As described above, calculating metrics requires judgments of the relevance of each document to each query. Precision requires a judgment on every retrieved document to the query. Recall requires that every document that is in the index and relevant to the query has been identified; thus until every document in the collection has been judged, the possibility remains that recall is being overestimated.

In Cleverdon and Mills' original experiments, judgments were made on how relevant a cited reference was to the research questions that inspired the citing paper [4]. They were made by the authors of the citing papers, the ones who came up with the research questions to begin with. To fill out the set needed for precise recall computation, additional judgments were made by students working for Cleverdon.

With the shift to much larger, much more heterogeneous document collections that was inaugurated by the National Institute for Standards and Technology (NIST) at the TREC conferences, it became impossible to judge every document to every query. Instead, the *pooling* method [10] was adopted: a set of queries is sent to participating sites without relevance judgments; sites run the queries through their retrieval systems and return the resulting ranked lists to NIST. The top N documents retrieved by each system are pooled, and the entire pool judged for relevance. Although this results in a small fraction of the total collection being judged, it is a biased sample that ensures that most of the documents that are likely to be retrieved by any system will be judged. Zobel has shown that although relevant documents are missed using this method (and thus recall overestimated), it is more than satisfactory for evaluation when the goal is comparing two or more different systems [13]. Additional work has shown that reliable comparisons can be made with very few judgments; even when judgments needed to calculate precision are missing, system comparisons can often be made with high confidence, and even when confidence is not high, the degree of confidence can be reliably estimated [2,1].

The judgments acquired for TREC are typically binary – relevant or not – or trinary – highly relevant, relevant, or not relevant – but binarized for evaluation. This is appropriate for the tasks studied at TREC, which tend to emphasize recall. For many types of web searches, recall is significantly less important than precision. For example, the query “microsoft” may return Microsoft’s corporate web page, pages about Microsoft software, pages about court cases Microsoft is involved in, and pages about Microsoft’s stock activity. All of these are relevant to some user’s need, but it is unlikely that all of them are relevant to the same need. Since the query is so broad, the best ranking would probably put Microsoft’s home page at rank 1. But if relevance is binary, any of those pages would be considered equally relevant, and a page about a small drop in Microsoft stock on a certain date could appear at rank 1 without affecting precision or recall, even though the user’s utility is clearly negatively affected.

To resolve this, web judgments are often made on a graded scale. Examples of graded scales include the “highly relevant,” “relevant,” and “nonrelevant” sometimes used at TREC; “highly relevant,” “relevant,” “maybe relevant,” “nonrelevant” to allow for some uncertainty on the part of the judge, or the five-point scale originally used by Cleverdon and Mills. There is a trade-off between finer performance distinctions and judgment quality, however: as more categories are added, it becomes harder to define what exactly distinguishes one category from another, and as a result the judgments become less reliable. Even with the binary judgments and highly-specified information needs used at TREC, there is a fair amount of disagreement about what is relevant [11]; when moving to finer scales and trying to infer user’s needs on the basis of a 1–3 word query, disagreement may skyrocket.

Hypothesis Testing and Relative Performance

Measures like average precision and NDCG defy easy interpretation. What does it mean for a system to have an NDCG of 0.69? Thus the goal of performance measurement is often to compare the performance of two engines, one of which may be a minor modification of the other. But a small difference in performance can occur simply by chance. A decision based on such a difference should take into account the probability that it is “real,” i.e., whether it is unlikely to have occurred only due to random factors.

Estimating this probability involves taking a random sample of queries likely to be input to the system. For the web, the sample can be obtained from search logs. The measure of interest is computed for each query, and some test statistic computed over the set. The ideal test statistic should have high power to detect the “real” differences when they exist.

There has been some debate over which test statistic (and therefore which hypothesis test) is applicable to information retrieval. If the same sample of queries can be treated as a random sample to either engine and both engines index the same documents, paired tests provide more powerful analysis [7]. The sign test makes no assumptions about the distribution of metrics like NDCG over queries, but is not very powerful. The Wilcoxon sign rank test, which has been popular, also makes no distributional assumptions, but as a test for difference in median has limited power to detect differences in mean performance. The t-test is a powerful test for detecting differences in means. Although it requires some distributional assumptions that may not hold in practice, it is robust to violations of those assumptions, and therefore is probably the best test to use when at least 25 queries can be sampled [13].

Key Applications

Measuring Search Engine Performance

Precision, recall, and DCG measure how well the engine ranks documents independent of other factors that can influence users’ opinions, such as interface, extra tools, and so on. Each metric measures a different aspect of performance with varying degrees of fineness.

Comparing Search Engines

Metrics allow the comparison of two different search engines or two variations on a baseline ranking algorithm. The statistical significance of differences can be evaluated and used to make decisions about development and deployment.

Optimizing Search Engine Performance

Search engine algorithms can be optimized to maximize performance on one or more of these metrics.

Future Directions

There are many open problems in search performance measurement: how to evaluate personalized search (in which results are tailored to the user), how to evaluate novelty (ensuring that the same information is not duplicated in results), how to use context in evaluation, and so on.

A challenge of web evaluation is the ever-changing nature of the query stream and the indexed documents [9]. The distribution of queries changes frequently, and there is always a long tail of queries that only appear in the logs once. As a result, queries need to be resampled and reevaluated constantly. Web pages disappear or fall out-of-date frequently, and judgments should be kept accordingly up-to-date. Finally, changes in the search engine's interface or its underlying algorithms can affect the way users interact with it, making comparisons between engines separated by long time periods difficult if not impossible.

Finally, there is still more work to be done on understanding how missing relevance judgments affect conclusions that can be drawn from evaluations.

Data Sets

The TREC test collections described in this article are available from NIST at http://trec.nist.gov/.

Cross-references

- ▶ Average Precision
- ▶ Discounted Cumulated Gain
- ▶ F-Measure
- ▶ Information Retrieval
- ▶ MRR
- ▶ Relevance
- ▶ R-Precision
- ▶ Web Page Quality Metrics
- ▶ Web Search Relevance Ranking

Recommended Reading

1. Aslam J.A., Pavlu V., and Yilmaz E. A statistical method for system evaluation using incomplete judgments. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 541–548.
2. Carterette B., Allan J., and Sitaraman R.K. Minimal test collections for retrieval evaluation. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 268–275.
3. Cleverdon C.W. The cranfield tests on index language devices. In Readings in Information Retrieval. K.S. Jones and P. Willett (eds.). Morgan Kaufmann, 1967, pp. 47–59.
4. Cleverdon C.W. and Mills J. The testing of index language devices. In Readings in Information Retrieval. K.S. Jones and P. Willett (eds.). Morgan Kaufmann, 1963, pp. 98–110.
5. Kekalainen J. and Jarvelin K. Using graded relevance assessments in ir evaluation. JASIST, 53:1120–1129, 2002.
6. Papineni K., Roukos S., Ward T., and Zhu W.J. BLEU: a method for automatic evaluation of machine translation. In Proc. 40th Annual Meeting of the Assoc. for Computational Linguistics, 2002, pp. 311–318.
7. van Rijsbergen C.J. Information Retrieval. Butterworths, London, UK, 1979.
8. Salton G. and Lesk M.E. Computer evaluation of indexing and text processing. In Readings in Information Retrieval. K.S. Jones and P. Willett (eds.). Morgan Kaufmann, 1967, pp. 60–84.
9. Soboroff I. Dynamic test collections: measuring search effectiveness on the live web. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 276–283.
10. Sparck J.K. and van Rijsbergen C.J. Information retrieval test collections. J. Doc., 32(1):59–75, 1976.
11. Voorhees E. Variations in relevance judgments and the measurement of retrieval effectiveness. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 315–323.
12. Voorhees E.M. Harman D.K. (eds.). TREC: Experiment and Evaluation in Information Retrieval. MIT, Cambridge, MA, USA, 2005.
13. Zobel J. How reliable are the results of large-scale information retrieval experiments? In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 307–314.

Search Engine Query Result Caching

▶ Web Search Result Caching and Prefetching

Search Ranking

▶ Web Search Relevance Ranking

Searching Compressed XML

▶ Managing Compressed Structured Text

Searching Digital Libraries

PANAGIOTIS G. IPEIROTIS
New York University, New York, NY, USA

Synonyms

Federated search

Definition

Searching digital libraries refers to searching and retrieving information from remote databases of digitized or digital objects. These databases may hold either the metadata for an object of interest (e.g., author and title), or a complete object such as a book or a video.

Historical Background

The initial efforts to standardize and facilitate searching of digital libraries date back to the 1970s, when the development of the Z39.50 protocol started. The Z39.50 protocol is an ANSI standard and defines how to search and retrieve items from a remote database catalog. The Z39.50 protocol was widely deployed within library environments, allowing users to perform searches to remote libraries.

With the advent of the Web, libraries started digitizing and making contents available on the Web, and the Z39.50 protocol started losing its importance. Many libraries made their content "searchable" through standard Web forms, allowing users to search and retrieve content using simply a Web browser. However, due to the lack of a link structure, the contents of the libraries remained "hidden" from the modern search engine crawlers, forming part of the "Hidden-Web" (also known as Deep Web, or Invisible Web). Searching across multiple Hidden Web databases, despite the tremendous progress since 2000, is still an open research problem.

However, achieving interoperability across all Web databases is inherently harder than achieving interoperability across library databases, which are relatively more homogeneous. Therefore, a set of efforts focused on introducing protocols to facilitate integrating and searching digital libraries. The Open Archives Initiative focused on defining a protocol for exporting metadata about the objects in the collections hosted by each library. The SRU protocol aims to modernize the Z39.50 by making it similar to modern Web services. Such efforts allow programmers to leverage their existing skills and develop easier tools for the library market.

Foundations

Digital libraries host a variety of digital objects, including, but not limited to, textual documents, images, sounds, videos, or even multimodal objects that combine the above. The concept of searching digital libraries may refer either to the action of searching a *single* digital library or to the action of searching across *multiple* digital libraries.

Searching a *single* digital library typically refers to the action of searching and browsing the contents of the underlying relational, textual, or multimedia database.

Searching across *multiple* digital libraries is a concept that evolved significantly over the years. The development of these efforts is broadly divided in three periods:

- *The pre-Web period (late 1970s–mid 1990s)*: Development of the Z39.50 standard.
- *The early-Web period (mid 1990s–early 2000s)*: Emergence of the Web, and increased accessibility of libraries over the Web.
- *The Web-services period (early 2000s–now)*: Definition of protocols for Web services, and development of library-focused search and discovery protocols.

The Pre-Web Period

The first attempts to define a standardized, common protocol for searching library databases date back to the 1970s. Then, the "Linked Systems Project" examined how to provide support for standardized access method to a small set of homogeneous, bibliographic databases. This effort led to the formation of a NISO committee in 1979, which after years of efforts defined the "American National Standard Z39.50, Information Retrieval Service Definition and Protocol Specifications for Library Applications" in 1987. The protocol was later revised in 1992, in 1995, and in 2003. (See [11] for a detailed history and timeline of the development of Z39.50.)

The Z39.50 protocol was designed as a client-server protocol, defining how the client can search and retrieve information from a remote database. The protocol supports a significant number of actions, including searching across individual fields, such as author, abstract, title, and so on. Unfortunately, the protocol did not mandate the implementation of several aspects of the specifications, allowing the developers to choose the aspects of the protocol to implement. This led to unexpected behavior of some systems, as the same query, executed over the same underlying content, could return very different results, depending on the implementation. Furthermore, the extremely heavy specification made it difficult for vendors to develop systems that were fully compatible with each other.

The Early-Web Period

The emergence of the Web changed significantly the way that digital libraries make their content available. Many libraries, perhaps encouraged by the *Digital Libraries Initiative* in 1994, started digitizing and making their content available over the Web. This meant that user could simply visit the Web site of a library and then, using simply Web forms, could query and browse the holdings of the library.

A significant fraction of these new digital libraries are only accessible via a search interface and the ability to browse through a static hyperlink structure is often missing. This means that the contents of these libraries are "hidden" from search engines, since traditional crawlers, which discover new pages by following links, cannot discover the contents of the library. Such libraries are part of the *hidden-Web* [2]. On the other hand, libraries that provide a link structure for accessing their holdings, are part of the *surface Web*, which is accessible by using general search engines, such as Google.

For libraries with content available as part of the *surface Web*, the common model for searching is through vertical search engines. The vertical search engines create topically-focused indexes of the material available on the Web by using *focused crawlers* [4] to identify and index the pages about a given topic. Under this model, the distributed digital libraries become searchable through a centralized search interface that indexes the remotely stored content. When a user issues a query, the vertical search engine identifies the most relevant pages in the index and returns to the user the URLs of the pages, which are stored remotely.

For libraries with *hidden Web* content, the typical way of searching their contents is through *metasearchers*. A complete metasearcher has to perform the following tasks:

- Discover the available digital libraries. This involves crawling the Web to identify pages with Web forms that are search interfaces for underlying databases [5].
- Understand the capabilities of the available query interface [1,13,16].
- Characterize the contents of the underlying database, typically by extracting a small sample of the stored contents through query-based sampling. The characterization may involve classifying the database into a topic hierarchy [6], extracting a statistical summary of the content [3,8], or it may involve keeping the actual sample as a surrogate for the contents of the database [7,15].
- Use the database characterization to select the most promising databases for evaluating a given query [9,15].
- Evaluate the queries in the selected databases, retrieve, and merge the results from multiple databases into a single list [14].

An alternative approach to the distributed search technique adopted by metasearchers is to try to download *all* contents of a hidden Web database [12]. Once all the contents of the remote digital libraries are retrieved and stored locally, the problem of searching multiple digital libraries is reduced to the problem of searching a single, centralized database. One of the issues in this case is the need to periodically refresh the local copy with the most recent contents of the remote database [10].

The Web-Services Period

During the early-Web period, the problem of integrating and searching across digital libraries was similar to the problem of integrating Web databases at large. The vision of the *semantic Web* promised a solution for this problem, and the implementation of a *Web services* framework was a first step towards this direction.

Inherently, though, the library integration problem is much easier than the problems involved in the full implementation of the semantic Web. Therefore, a set of niche solutions were developed for the library integration problem, focusing on the one hand on library-specific needs, but building on top of the existing tools for general Web services that are being developed and rapidly improved.

One of the first attempts to make effortless the discovery of the contents of a library database was the development of the *Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)*. This protocol defines how a library can export metadata descriptions of its holdings. Then, *metadata harvesters* can easily collect the contents of the database and make these contents searchable through a centralized search interface. The OAI-PMH protocol is now widely adopted by many libraries and a set of OAI registries facilitate even further the discovery of libraries that support this protocol. Notably, major search engines, such as Google and Yahoo! also support the protocol, as an alternative of the *sitemaps protocol*. This support allows libraries to be an integral part of the general Web and at the same time use a protocol developed and customized for their own needs.

Beyond OAI, there are also attempts to modernize the Z39.50 protocol and make it part of the larger family of Web protocols. First, the *Bath profile* specifies the exact query syntax that Z39.50 clients should use, so that clients can interpret the results returned by Bath-compliant Z39.50 servers. A more significant development is the agreement for the *Search/Retrieval via URL (SRU) protocol*. SRU is a standard XML-focused search protocol for Internet search queries that uses *Contextual Query Language (CQL)* for representing queries. The SRU uses the REST protocol and introduces a standard method for querying library databases, by simply submitting URL-based queries. For example, consider the following URL-encoded query:

http://z3950.loc.gov:7090/voyager?version=1.1&operation=searchRetrieve&query=dinosaur&maximumRecords=10

This example is a search for the term "dinosaur," requesting that at most ten records to be returned. The SRU protocol is easy to support and implement, and is familiar to programmers that also use such syntax to interact with other popular Web services.

Key Applications

Digital libraries are increasingly becoming part of everyday life. The book digitization projects undertaken by corporations (e.g., Google, Microsoft) and by many universities will generate enormous digital archives accessible over the Web. Similarly, the high-quality holdings of the existing libraries are becoming increasingly accessible over the Web, allowing users to reach easier authoritative sources of information.

Cross-references

- ▶ Bioinformatics Data Management
- ▶ Digital Libraries
- ▶ Health Informatics Databases
- ▶ Metadata Management
- ▶ Multimedia Databases
- ▶ Multimedia IR
- ▶ Querying over Data Integration Systems
- ▶ Scientific Databases
- ▶ Semantic Web and Ontology
- ▶ Semi-structured Text Retrieval
- ▶ Structured and Semi-structured Document Databases
- ▶ Text Retrieval
- ▶ Web Search and Crawl
- ▶ Web Services and Service Oriented Architecture

Recommended Reading

1. Bergholz A. and Chidlovskii B. Using query probing to identify query language features on the web. In Distributed Multimedia Information Retrieval, In Proc. SIGIR 2003 Workshop on Distributed Information Retrieval, 2004, pp. 21–30.
2. Bergman M.K. The deep Web: surfacing hidden value. J. Electron. Pub., 7(1), August 2001.
3. Callan J.P. and Connell M. Query-based sampling of text databases. ACM Trans. Inf. Syst., 19(2):97–30, 2001.
4. Chakrabarti S., van den Berg M., and Dom B. Focused crawling: a new approach to topic-specific web resource discovery. Comput. Netw., 31(11–16):1623–1640, May 1999.
5. Cope J., Craswell N., and Hawking D. Automated discovery of search interfaces on the web. In Proc. 14th Australasian Database Conf., 2003, pp. 181–189.
6. Gravano L., Ipeirotis P.G., and Sahami M. QProber: a system for automatic classification of hidden-web databases. ACM Trans. Inf. Syst., 21(1):1–41, January 2003.
7. Hawking D. and Thomas P. Server selection methods in hybrid portal search. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 75–82.
8. Ipeirotis P.G. and Gravano L. Distributed search over the hidden web: hierarchical database sampling and selection. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 394–405.
9. Ipeirotis P.G. and Gravano L. When one sample is not enough: improving text database selection using shrinkage. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 767–778.
10. Ipeirotis P.G., Ntoulas A., Cho J., and Gravano L. Modeling and managing content changes in text databases. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 606–617.
11. Lynch C.A. The Z39.50 information retrieval standard. D-Lib Mag., 3(4), April 1997.
12. Ntoulas A., Zerfos P., and Cho J. Downloading textual hidden web content by keyword queries. In Proc. ACM/IEEE Joint Conf. on Digital Libraries, 2005.
13. Raghavan S. and García-Molina H. Crawling the hidden web. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 129–138.
14. Si L. and Callan J. A semisupervised learning method to merge search engine results. ACM Trans. Inf. Syst., 21(4):457–491, 2003.
15. Si L. and Callano J. Modeling search engine effectiveness for federated search. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 83–90.
16. Zhang Z., He B., and Chang K.C.-C. Understanding web query interfaces: best-effort parsing with hidden syntax. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 107–118.

Second Normal Form (2NF)

Marcelo Arenas
Pontifical Catholic University of Chile, Santiago, Chile

Synonyms

2NF

Definition

Let $R(A_1,\ldots,A_n)$ be a relation schema and Σ a set of functional dependencies over $R(A_1,\ldots,A_n)$. An attribute A_i ($i \in \{1,\ldots,n\}$) is a *prime* attribute if A_i is an element of some key of $R(A_1,\ldots,A_n)$. Then specification (R, Σ) is said to be in Second Normal Form (2NF) if for every nontrivial functional dependency $X \rightarrow A$ implied by Σ, it holds that A is a prime attribute or X is not a proper subset of any (candidate) key for R [1].

Key Points

In order to avoid update anomalies in database schemas containing functional dependencies, 2NF was introduced by Codd in [1]. This normal form is defined in terms of the notions of prime attribute and key as shown above. For example, given a relation schema $R(A, B, C)$ and a set of functional dependencies $\Sigma = \{A \rightarrow B\}$, it does not hold that $(R(A, B, C), \Sigma)$ is in 2NF since B is not a prime attribute and A is a proper subset of the key AC. On the other hand, $(S(A, B, C), \Gamma)$ is in 2NF if $\Gamma = \{A \rightarrow B, B \rightarrow C\}$, since A is a key (and thus it is not a proper subset of any candidate key) and B is not contained in any (candidate) key for S.

It should be noticed that relation schema $S(A, B, C)$ above is in 2NF if $\Gamma = \{A \rightarrow B, B \rightarrow C\}$, although this schema is not in 3NF. In fact, 3NF is strictly stronger than 2NF; every schema in 3NF is in 2NF, but there exist schemas (as the one shown above) that are in 2NF but not in 3NF.

Cross-references

- ▶ Boyce-Codd Normal Form
- ▶ Fourth Normal Form
- ▶ Normal Forms and Normalization
- ▶ Third Normal Form

Recommended Reading

1. Codd E.F. Further Normalization of the Data Base Relational Model. In Data base systems. Englewood Cliffs, N.J. Prentice-Hall, 1972, pp. 33–64.

Secondary Index

Yannis Manolopoulos[1], Yannis Theodoridis[2], Vassilis J. Tsotras[3]
[1]Aristotle University of Thessaloniki, Thessaloniki, Greece
[2]University of Piraeus, Piraeus, Greece
[3]University of California-Reverside, Riverside, CA, USA

Synonyms

Non-clustering index

Definition

A tree-based index is called a *secondary index* if the order which it maintains on the search-key values is *not* the same as the order of the file which it indexes. For example, consider a relation R with some numeric attribute A taking values over an (ordered) domain D. Assume that relation R is *not* physically stored on the values of attribute A (i.e., relation R is either stored as a heap – an unordered file, or is ordered on another attribute). Furthermore, assume that a tree-based index (e.g., B + -tree) has been created on attribute A. Then this index is secondary.

Key Points

Tree-based indices are built on numeric attributes and maintain an order among the indexed search-key values. They are further categorized by whether their search-key ordering is the same with the file's physical order (if any). Note that a file may or may not be ordered. Ordered is a file whose records are stored in pages according to the order of the values of an attribute. Obviously, a file can have at most a single such order since it is physically stored once. For example, if the *Employee* relation is ordered according to the *name* attribute, the values in the other attributes will not be in order. A file stored without any order is called an unordered file or heap. An index built on any non-ordering attribute of a file is called *secondary* (or non-clustering) while an index built on the ordering attribute of a file is called *primary* (clustering).

Since the actual data record can be anywhere in the file, the secondary index needs an extra level of indirection, namely, a pointer to the actual position of a record with a given value in the relation file. In other words, a secondary index only clusters references to records (in the form of <value, pointer> fields), but *not* the records themselves. This extra indirection from a leaf page of a secondary index to the actual position of a record in a file has important subsequences on optimization. Consider, for example, a secondary index (B + -tree) on the *ssn* attribute of the *Employee* relation (which assume is ordered by the *name* attribute). A query that asks for the salaries of employees with *ssn* in the range *[x, y]* can facilitate the B + -tree on *ssn* to retrieve references to all records in the query range. Assume there are 1,000 such *ssn* values in the *Employee* file. Since the actual *Employee* records must be retrieved (so as to report their salaries), each such reference needs to be materialized by possibly a separate page I/O (since the actual records can be in different pages of the *Employee* file).

A relation can have several indices, on different search-keys; among them, at most one is primary (clustering) index while the rest are secondary ones.

Cross-references

▶ Access Methods
▶ B+-Tree
▶ Index Sequential Access Method (ISAM)
▶ Indexing

Recommended Reading

1. Elmasri R.A., and Shamkant N.B. Fundamentals of Database Systems (5th edn.). Addisson-Wesley, Reading, MA, 2007.
2. Manolopoulos, Theodoridis Y. and Tsotras Y. Vassilis J. Advanced Database Indexing. Kluwer, Dordecht, 1999.
3. Ramakrishnan and Raghu Gehrke. Johannes Database Management Systems (3rd edn.). McGraw-Hill, NY, 2003.

Secret-Key Encryption

▶ Symmetric Encryption

Secure Data Outsourcing

Barbara Carminati
University of Insubria, Varese, Italy

Synonyms

Secure third-party data management

Definition

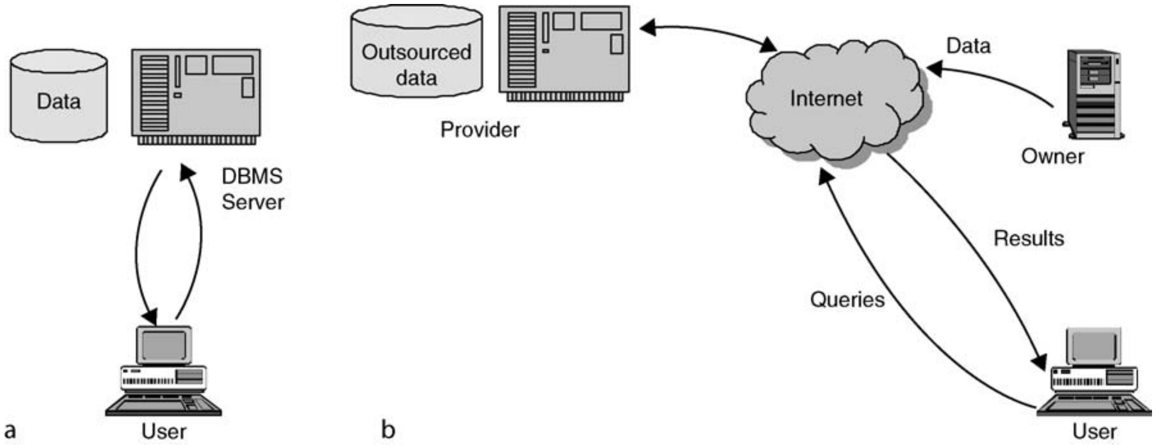
Data outsourcing is a new, emerging data management paradigm in which the owner of data is no longer totally responsible for its management. Rather, a portion of data is outsourced to external providers who offer data management functionalities. Secure data outsourcing is a discipline that investigates security issues associated with data outsourcing.

Historical Background

At present, service outsourcing is a paradigm widely used by many companies and organizations to achieve better service by delegating some of their business functions to external specialized service providers. A natural evolution of this paradigm is the recent emergence of *data outsourcing*. With this strategy, a company is no longer completely responsible for its own data management. Rather, it outsources some of its data functionalities to one or more external data management service providers (such as efficient query processing or large storage capability). Data outsourcing clearly leads to a range of security issues, because data owners have the potential to lose control over the data that is outsourced. Thus, the challenge is to ensure the highest level of security when data are managed by external service providers. With this aim, several research groups have started to investigate and propose mechanisms to achieve *secure data outsourcing*.

Foundations

In general, the enforcement of data outsourcing requires examination of new, challenging issues. With traditional, well-known client server architecture (see Fig. 1a), data owners manage the DBMS and directly answer user queries. Data outsourcing relies on third-party architecture (see Fig. 1b), in which data owners outsource their data (or portions of it) to one or more service providers. In real world environments, it cannot be assumed that third parties always operate according to the data owner's security policies. By contrast, to achieve secure data outsourcing, one needs to define techniques that satisfy the main security properties even in the presence of an *untrusted* third party – that is, a provider that could maliciously modify or delete the data it manages by, for instance, inserting fake records or sending data to unauthorized users. Several researchers have focused on this problem and have developed different proposals. However, before illustrating the techniques proposed to date, it is



Secure Data Outsourcing. Figure 1. (a) Two-party; (b) third-party architecture.

necessary first to identify the main security requirements in secure data outsourcing.

Security Requirements in Secure Data Outsourcing

In third-party architectures, potentially untrusted providers manage data. For this reason, secure data outsourcing must examine novel security issues as well as reexamine the traditional ones. The following presents the main security issues studied so far in secure data outsourcing.

Privacy: If a third-party architecture is adopted, a user could be concerned about his/her privacy for any query processing that is performed by a third-party provider. This is due to the fact that by simply tracking a user's queries, an untrusted provider could infer sensitive information about the user (for instance, the user's preferences). For this reason, the privacy of the submitted queries needs to be protected. To ensure access privacy, a provider should not be able to know the details of the query, but should be able to process it.

Authenticity and Integrity: Ensuring authenticity and integrity in third-party architecture enables a user, upon receiving some data from a provider, to verify that the data received has been in fact generated by the data owner and not modified by the provider. (*Integrity* also has an additional meaning, that is, ensuring that unauthorized users have not modified the data; however, since data outsourcing is mainly conceived for read-only data access, this definition of *integrity* is not considered in this entry.) In traditional architectures, both authenticity and integrity are ensured by means of digital signatures. When a user submits a query, the data owner re-evaluates it and digitally signs the query result. Then, the query result together with its digital signature is sent to the user, thus enabling the user to verify the query's authenticity and integrity. However, in third-party architectures, traditional signature techniques cannot be used. A provider may return to the user only selected portions of the signed data in answer to the query evaluation. Thus, a user that is provided with only these portions is not able to validate the owner's digital signature, which has been generated on the whole data. To cope with these requirements, alternative ways must be found to sign outsourced data digitally so that a user is able to validate the digital signature even if he/she has only received selected portions of the signed data.

Completeness: Third-party architectures introduce a further novel security requirement, called completeness. If satisfied, this property ensures a user that the answer received by third-party service providers to a query genuinely contains all of the data answering to the submitted query. (In the literature, some works refer to this property as *query correctness*, by also implying authenticity and integrity requirements.)

Confidentiality: Data confidentiality means ensuring that data are disclosed only to authorized users. However, it is obvious that when data are outsourced, confidentiality requirements are not limited to users, but extended also to providers. Thus, confidentiality in data outsourcing acquires a twofold meaning. The first deals with protecting the owner's data from access by a malicious or untrusted provider, and is referred to as *confidentiality wrt the provider*. A further confidentiality requirement, hereafter called *confidentiality wrt users*, refers to the protection of data from unauthorized user access on the basis of the access control policies stated by the data owners. In traditional client-server architectures, this requirement is enforced by access control mechanisms, called *reference monitors*, which mediate each user request by authorizing only those in accordance with the owner's access control policies. This type of solution can hardly be applied in data outsourcing, since it implies the delegation of the reference monitor tasks to a potentially untrusted publisher. For this reason, alternative solutions should be devised for access control enforcement when data services are outsourced.

Techniques for Secure Data Outsourcing

Many research groups have investigated security property enforcement in data outsourcing, resulting in several proposals for different security requirements. The following presents the main results proposed so far, grouped according to the security properties addressed by each.

Access Privacy: Private Information Retrieval protocols (PIR, for short), first introduced in [5], are one of the most relevant results of investigations into the problem query privacy protection. The underlying idea of PIR protocols is that several replications of the same database are available in different servers (i.e., providers). To preserve access privacy, the user submits different queries to each different server, defined so that by combing the servers' answers, the user is able to obtain the information desired, but by analyzing the submitted query, each server is unable to infer the actual interest of the user.

Authenticity and Integrity: These are the first properties that have been investigated in secure data outsourcing. The aim is to devise alternative digital signature schemes for signing the data to be outsourced, enabling users to validate the signature even if users are only provided with selected portions of the signed data. Several schemes have been proposed so far, exploiting different strategies for achieving this result. In particular, two of the most widely used techniques are Merkle trees and aggregate signatures. The following presents both of these concepts by introducing some of the related proposals.

Merkle Trees. Merkle proposed a method to authenticate, with a unique signature, a set of messages $\{m_1,\ldots,m_n\}$, by at the same time enabling an intended verifier to authenticate a single message without the disclosure of the other messages. The proposed solution exploits a binary tree, where each leaf contains the hash values of a message in $\{m_1,\ldots,m_n\}$, whereas internal nodes enclose the concatenation of the hash values corresponding to its left and right children (see Merkle tree entry for more details). The root node of the resulting binary hash tree can be considered the digest of all messages, and thus it can be digitally signed by using a standard signature technique. The main benefit of this method is that a user is able to validate the signature by having a subset of messages, provided that he/she receives a set of additional hash values. Indeed, by having hash values of the missing messages, a user is able to build up locally the binary hash tree and thus is able to validate the signature. Merkle trees have been used in several computer areas. However, the first work to exploit them in data outsourcing was the one by Devanbu et al. [7], which adapts these trees to relational data to prove the completeness, authenticity, and integrity of query answers. According to this approach, for each relation *R*, a different Merkle tree is generated in such a way that leaves contain hash values of tuples in *R*. Then, when a user submits a query on *R*, the provider replies to him/her with the tuples answering the submitted query together with the signature generated on the root node of the corresponding Merkle tree. Moreover, the provider also sends the user the hash values of the tuples of *R* not included in the result set. These additional hash values enable the user locally to generate the Merkle tree of *R* and to validate the signature. A similar approach has been proposed to authenticate query results in edge computing [14]. Here, besides signing only the root of the Merkle tree, all leaves as well as all of the internal nodes are also signed. This leads to a reduced number of hash values to be returned to users to enable them to verify owner signatures. Merkle trees have also been investigated for secure data outsourcing of XML documents [1,6]. In these approaches, Merkle trees are generated in a different way wrt those computed over relational data, i.e., a flat list of tuples. Here, the challenge is how to generate a Merkle tree that exploits the hierarchical organization of an XML document. In both these approaches, a hash value is univocally associated with the document root through a recursive bottom-up computation on its structure. The digest is computed by associating a hash value with each node *n* of the XML document, computed by also taking into account the hash values of its children and the attributes in addition to contents of the n itself. Thus, the digest of the whole document is the hash value of the root of the document.

Aggregate signatures. Boneh et al. [2] introduced the notion of aggregate signatures, that is, a signature scheme able to aggregates n distinct signatures generated by n distinct data owners into a unique digital signature. The main advantage of this scheme is that the validation of this unique digital signature implies the validation of each component signature. The aggregate signature has been used by Mykletun et al. in [12] to ensure authenticity and integrity of outsourced relational data. According to this approach, given a relation R, the data owner generates a different signature for each tuple in R. These signatures, together with the corresponding tuples, are outsourced to the provider. Then, when a user submits a query on relation R, the provider evaluates the query on R and aggregates all of the signatures corresponding to the tuples in the result set into a unique signature. Therefore, as result of the submitted query, the third party returns to the user the resulting aggregate signature, as well as the tuples answering the query. The properties of aggregate signatures assure the user that if the aggregate signature generated by the third party is valid, then all of the signatures generated by the owner on the tuples in the result set are also valid, which proves tuples' authenticity and integrity.

Completeness: Researchers have investigated the completeness property in combination with authenticity and integrity requirements. As a consequence, the main solutions proposed for ensuring this property exploit the same techniques used for authenticity and integrity – that is, Merkle trees and aggregate signatures. First, let introduce how completeness can be ensured by exploiting Merkle trees. In particular, [7] considers the problem of completeness of answers to range queries. The proposed solution's underlying idea is that given a range query whose predicate is against attribute a, the Merkle tree is generated over tuples sorted according to a values. Then, when a user submits a range query containing a predicate against a, the provider returns two additional tuples to the user together with the result set answering the range query – that is, the tuples precedent and subsequent to the lower and upper bound of the result set. These two values are then used to verify the owner's signature. Since the signature is computed on a Merkle tree generated over sorted tuples, if the user validates the owner's signature, he/she is ensured that the received precedent and subsequent tuples really precede and follow the lower and upper bounds of the result set and that no tuples are omitted from the result set, which proves the completeness of the answer. In contrast, the approach to ensure completeness that has been proposed by Mykletun et al. [12] modifies an aggregation signature scheme in such a way to include in the signature of a tuple t, the hash value of tuples preceding t according to all possible sorts defined on a the relation's attributes, obtaining a so-called signature chain. Thus, when a user submits a range query, the provider also inserts into the result set the boundary tuples, i.e., the tuples preceding the upper and lower bound of the result set, as well as their aggregated signatures. By means of these signatures, a user is therefore able to prove that the third party has not omitted any tuple.

Notice that to ensure completeness, both these solutions require the user to be sent two additional tuples wrt the result set answering the range query. This could lead to some confidentiality breaches, since these additional tuples could contain sensitive data. To overcome this problem, Pang et al. [13] proposed an approach exploiting signature chains that does not disclose more tuples than those in the result set.

Confidentiality: Several research groups have investigated confidentiality issues in data outsourcing. Most have focused on confidentiality wrt providers, whereas only the approach proposed in [3] also investigates confidentiality wrt users in the context of outsourcing XML documents. It is interesting to note, however, that all these solutions share a common underlying idea by which the data owners provide outsource service providers with an encrypted version of the data to manage, without providing them with the corresponding decryption keys. Consequently, the third party is unable to access and to misuse outsourced data. This obviously ensures confidentiality wrt providers. Also, to provide assurance of confidentiality wrt users, the authors of [3] proposed a selective encryption for XML documents in which different portions of the same document are encrypted with different encryption keys. More precisely, the owner encrypts all portions of the data to which the same policies apply with the same key. Then, the owner provides each user with all and only the keys corresponding to the portions of data that the user is allowed to access. This selective key distribution ensures that each user is able to access all and only the portions of data for which there is an access control policy authorizing the access. Obviously, applying these solutions requires addressing an interesting problem – that is, how to enable providers to evaluate queries on encrypted data without accessing them. In recent years, this problem has been deeply investigated by several researchers, with the results of different proposals. The following sections introduce some of them by grouping them according to the underlying data model.

Querying encrypted relational data. The most relevant solution to querying encrypted relational data has been proposed by Hacigumus et al. [10,11], where binning techniques and privacy homomorphic encryption are exploited to execute SQL queries over encrypted relations. The first step is to introduce how binning techniques enable a service provider to evaluate selection queries. The underlying idea in [10,11] is that, given a relation *R*, the data owner partitions the domain of each attribute in *R* into distinguished intervals, to which the data owner assigns a different id. Then, for each tuple *t* in *R*, the data owner outsources to providers its encryption complemented with the ids associated with the intervals to which *t*'s attribute values belong. According to this approach, when a user intends to submit a query, he/she rewrites the query's conditions in terms of interval ids, thus enabling the provider with the ability to evaluate them without accessing the data. For instance, the condition "Salary =200K" is rewritten as "Salary =*id*(200K)," where *id*(200K) is the id of the partition containing the value 200K. Moreover, to enable third parties to evaluate aggregate functions over encrypted data, in [11] Hacigumus et al. exploited privacy homomorphisms (PH) to calculate some arithmetic operations directly on encrypted data. More precisely, PH functions are used to encrypt attributes of R, on which it is expected to do some aggregations. The data owner outsources the encrypted tuples of R and the corresponding partition ids together with attributes encrypted by PH functions. In [11], it is shown that by having the privacy homomorphisms of attributes to be aggregated, the third party is able to evaluate aggregate functions directly over them.

Querying encrypted textual data. In 2000, Song et al. proposed a first cryptographic scheme that supports searching words on encrypted textual data [16]. (In this context, the textual data consists of a set of encrypted words.) According to this scheme, the third party is provided with a ciphered version of words to be managed. These ciphered words are generated as follows: first, each word is symmetrically encrypted with a single secret key *k*; then, each resulting encrypted word is XORed with a different pseudorandom number. Since different occurrences of the same encrypted word are XORed with a different pseudorandom number, information about the word distribution cannot be inferred by analyzing the distribution of the encrypted words. Each user is provided with the secret key *k*, and the used pseudorandom numbers. (The scheme proposed in [16] is defined in such a way that users are able to compute pseudorandom numbers locally without any interaction with the data owner.) By having this information, therefore, when a user intends to ask the third party for a keyword *W*, the user first generates the encrypted word using the secret key *k*, and then computes the XOR of the result with the corresponding pseudorandom number. The user then submits the obtained ciphered word to the third party, which sequentially scans all ciphered words to search for the one matching the one submitted. Thus, this scheme allows the third party to search for a keyword *W* directly on the ciphered data without gaining any information on the clear text or on the required keyword *W*. In 2003, Eu-Jin Gon proposed an alternative solution for searching keywords in an encrypted document [8], based on indexes. According to this approach, a different index is associated with each document to be encrypted. These indexes, called security indexes, are based on Bloom filters and have the property to store hidden information about the keywords contained within the corresponding document. According to this scheme, the owner outsources the encrypted documents and the corresponding security indexes to service providers, which are then able to search for a keyword by simply accessing the indexes. A similar approach has been devised in [4], which proposes dictionary-based keyword indexes.

However, all of this work has the limitation that third parties are able to identify only documents matching with a given keyword, but are not able to support more expressive searches, such as Boolean combinations of keywords. A first step to overcome this limitation has been done by Golle et al. in [9], which proposes a public key scheme to support conjunctive keywords searches.

Key Applications

Data outsourcing offers several benefits. One of the most relevant is related to cost reduction. Indeed, the company pays only for services that it uses from providers, which are generally significantly less than the cost implied by deployment, installation, maintenance, and upgrades of DBMSs. Moreover, the data management services offered by specialized providers are more competitive than the ones provided by the company itself. A further benefit of data outsourcing is its scalability, since a company can outsource its data to as many providers as it needs according to the amount of data and the number of managed users, avoiding that provider might become a bottleneck for the system. All these benefits make secure data outsourcing suitable for a wide range of applications in different data domains. For geographical data, for instance, the secure data outsourcing paradigm can be adopted to support geomarketing services. A data owner can outsource some of its geographical data (for instance, maps at various levels of details) to a publisher that provides them to customers based upon different registration fees or different confidentiality requirements (for instance, maps of some regions that cannot be distributed to everyone because they show sensible objectives).

Future Directions

Given the attention that the data outsourcing paradigm is receiving, it is expected that secure data outsourcing will be intensely investigated in the future. Besides proposing more efficient strategies for the security requirements considered so far, it is necessary to consider further challenging security issues, like those related to user privacy and ownership protection. Moreover, more consideration must be given to complex data outsourcing scenarios to enable users to manipulate the outsourced data rather than just simply reading it.

Cross-references

- ► Access Control
- ► Data Encryption
- ► Digital Signatures
- ► Merkle Trees

Recommended Reading

1. Bertino E., Carminati B., Ferrari E., Thuraisingham B., and Gupta A. Selective and authentic third-party distribution of XML documents. IEEE Trans. Knowl. Data Eng., 16(10):1263–1278, 2004.
2. Boneh D., Gentry C., Lynn B., and Shacham H. Aggregate and verifiably encrypted signatures from bilinear maps. In Proc. Advances in Cryptology, 2003.
3. Carminati B., Ferrari E., and Bertino E. Securing XML data in third-party distribution systems. In Proc. Int. Conf. on Information and Knowledge Management, 2005.
4. Chang Y. and Mitzenmacher M. Privacy preserving keyword searches on remote encrypted data, Cryptology ePrint Archive, Report, 2004.
5. Chor B., Goldreich O., Kushilevitz E., and Sudan M. Private information retrieval. In Proc. Symp. on Foundations of Computer Science, 1995.
6. Devanbu P., Gertz M., Kwong A., Martel C., Nuckolls G., and Stubblebine S.G. Flexible authentication of XML documents. In Proc. 8th ACM Conf. on Computer and Communications Security, 2001.
7. Devanbu P., Gertz M., Martel C., and Stubblebine S.G. Authentic third-party data publication. In Proc. 14th Annual IFIP WG 11.3 Working conference on Database Security, 2000.
8. Goh E. Secure Indexes, Cryptology ePrint Archive, Report 2003/216, 2003.
9. Golle P., Staddon J., and Waters B. Secure conjunctive keyword search over encrypted data. In Proc. the Applied Cryptography and Network Security Conf., 2004.
10. Hacigumus H., Iyer B., Li C., and Mehrotra S. Executing SQL over encrypted data in the database service provider model. In Proc. 9th Int. Conf. on Database Systems for Advanced Applications, 2002.
11. Hacigumus H., Iyer B., Li C., and Mehrotra S. Efficient execution of aggregation queries over encrypted relational databases. In Proc. 9th Int. Conf. on Database Systems for Advanced Applications, 2004.
12. Mykletun E., Narasimha M., and Tsudik G. Authentication and integrity in outsourced databases. In Proc. 11th Annual Symp. on Network and Distributed System Security, 2004.
13. Pang H., Jain A., Ramamritham K., and Tan K. Verifying completeness of relational query results in data publishing. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
14. Pang H. and Tan K. Authenticating query results in edge computing. In Proc. 20th Int. Conf. on Data Engineering, 2004.
15. Rivest R., Adleman L., and Dertouzos M. On data banks and privacy homomorphisms. In Foundations of Secure Computation, Richard J. Lipton, David P. Dobkin, Anita K. Jones (eds.). Academic press, 1978, pp 169–178.
16. Song D.X., Wagner D., and Perrig A. Practical techniques for searches on encrypted data. In Proc. IEEE Symp. on Security and Privacy, 2000.

Secure Database Design

► Secure Database Development

Secure Database Development

Jan Jurjens[1], Eduardo B. Fernandez[2]
[1]The Open University, Buckinghamshire, UK
[2]Florida Atlantic University, Boca Raton, FL, USA

Synonyms

Secure DBMS development; Secure database design

Definition

This entry considers how to build secure database system software. In particular, it describes how to build a general-purpose database management system where security is an important design parameter. For the database community, the words secure database design may refer to the schema design to produce a database for a specific application with some level of security properties. There is a large amount of literature on this latter subject and a related entry in this encyclopedia (Database security). This entry concentrates mostly on how to build the software of a DBMS such that it exhibits security properties, which is called secure database development. Both approaches are contrasted so that the reader can decide which one of these problems applies to their specific case but more space is dedicated to the general secure database development problem.

Historical Background

While there is a large number of papers on security models including authorization and other security aspects of databases [2,4,5], there is little work on how to implement a secure Database Management System (DBMS). It is true that many proposals for secure multilevel databases include details of implementation but most of them are ad hoc architectures that cannot be generalized to databases using different models or even to other multilevel databases with different requirements. Of the books on database security, [7] had several chapters on how to build secure relational database systems, and later [4] included also multilevel models. Those books do a good job of indicating the architectural units of such systems and their general requirements. However, software development aspects are not discussed in detail. It appears that [10] is the only work discussing these aspects explicitly.

Foundations

There are two aspects to the problem of developing software for secure databases: building a general (application-independent) secure DBMS and building a database system which is part of a secure application. These two problems are first briefly defined and then discussed in more detail. Other approaches and possible system architectures are also considered.

In the first approach the DBMS is just a complex software application in itself and a general secure software methodology can be applied without or with little change. Object-oriented applications typically start from a set of use cases, which define the user interactions with the system under development. In this particular case, use cases would define the typical functions of a DBMS, e.g. search, query, and update, and security would be included as part of its development life cycle. The DBMS would follow an appropriate model, e.g. Role-Based Access Control (RBAC), selectable in the design stage, which defines security constraints for the functions defined by the use cases. In some cases, it may be possible to support more than one security model. This would result in a secure DBMS where security would be a general nonfunctional requirement. The approach results in a general-purpose secure DBMS, where nothing is known about the specific applications that will be executed by its users. The DBMS itself is the application. The secure development methodologies of [6,13] and others are applicable here.

Another view is the one from a designer who needs to build a specific user application (or type of application) that includes a DBMS as part of its architecture, e.g. a financial system (most applications require a database but the degree of security needed may vary). This is discussed in [4,9,10,11]. In this case, the DBMS is rather ad hoc and tailored to the level of security desired for the specific type of application. For example, [11] separates the requirements into three types: functional, security, and database. Typically, these approaches emphasize how to define and enforce a set of application-specific rules that follow some security model and how to reflect them in the schema and other parts of the DBMS. Most of these studies emphasize the security of the database schema or some specific sections without much concern for the rest of the application. A methodology such as [6] or [13] can also be applied here, the DBMS being one of the architectural levels of a system that implements a specific application, although these methodologies have little to say about the contents of the specific rules that are needed in the schema (only their safe storage but not their consistency or security).

An interesting problem that applies to both approaches is the mapping from the conceptual security model (that may apply to a collection of DBMSs) to the authorization system of a specific database; for example, security constraints defined in a conceptual UML model defining authorizations in terms of classes must be mapped to an SQL-based authorization system which defines authorizations in terms of relations. Clearly, whatever is defined in the common conceptual model must be respected in the DBMS authorization system, although this latter may add further constraints related to implementation aspects.

General Secure Database Systems

In this case, as indicated earlier, the DBMS is a complex application requiring a general high level of security. There are several methodologies for this purpose and two of them are described below. A methodology for secure software development should include appropriate tools and provide a unified and consistent approach through all the life cycle stages. Ideally, a methodology should use a Model-Driven Development approach, where transformations between development stages are based on corresponding metamodels. Since the resulting software is independent of the access control model adopted, it does not provide for special requirements of the model; for example, multilevel models typically require data labeling. This means that the resulting software would be less secure than an ad hoc design (unless the multilevel model was the target in the example). Because of the generality of the resultant DBMS it may be difficult to prove formally security properties. An early approach in this direction was based on adding security functions to a general-purpose DBMS, e.g. INGRES or System/R.

Secure Database Development using Patterns

A methodology to build secure systems is presented in [6]. A main idea in the proposed methodology is that security principles should be applied at every stage of the software lifecycle and that each stage can be tested for compliance with those principles. Another basic idea is the use of patterns at each stage. A pattern is an encapsulated solution to a recurrent problem and their use can improve the reusability and quality of software.

Domain analysis stage: A business model is defined. Legacy systems are identified and their security implications analyzed. Domain and regulatory constraints are identified and use as global policies. The suitability of the development team is assessed, possibly leading to added training. This phase may be performed only once for each new domain or team. The need for specialized database architectures should be determined at this point. The approach (general DBMS or application-oriented system) should also de defined at this stage.

Requirements stage: Use cases define the required interactions with the system. Each activity within a use case is analyzed to see which threats are possible. Activity diagrams indicate created objects and are a good way to determine which data should be protected. Since many possible threats may be identified, risk analysis helps to prune them according to their impact and probability of occurrence. Any requirements for degree of security should be expressed as part of the use cases.

Analysis stage: Analysis patterns can be used to build the conceptual model in a more reliable and efficient way. The policies defined in the requirements can now be expressed as abstract security models, e.g. access matrix. The model selected must correspond to the type of application; for example, multilevel models have not been successful for medical applications. One can build a conceptual model where repeated applications of a security model pattern realize the rights determined from use cases. In fact, analysis patterns can be built with predefined authorizations according to the roles in their use cases. Patterns for authentication, logging, and secure channels are also specified at this level. Note that the model and the security patterns should define precisely the requirements of the problem, not its software solution. UML is a good semi-formal approach for defining policies, avoiding the need for ad-hoc policy languages. The addition of OCL (Object Constraint Language) can make the approach more formal.

Design stage: When one has defined the policies needed, one can select mechanisms to stop attacks that would violate them. A specific security model, e.g. RBAC, is now implemented in terms of software units. User interfaces should correspond to use cases and may be used to enforce the authorizations defined in the analysis stage. Secure interfaces enforce authorizations when users interact with the system. Components can be secured by using authorization rules for Java or .NET components. Distribution provides another dimension where security restrictions can be applied. Deployment diagrams can define secure configurations to be used by security administrators. A multilayer architecture is needed to enforce the security constraints defined at the application level. In each level, one can use patterns to represent appropriate security mechanisms. Security constraints must be mapped between levels.

The persistent aspects of the conceptual model are typically mapped into relational databases. The design of the database architecture is done according to the requirements from the uses cases for the level of security needed and the security model adopted in the analysis stage. Two basic choices for the enforcement mechanism include query modification as in INGRES and views as in System R. A tradeoff is using an existing DBMS as a Commercial Off-the-Shelf (COTS) component, although in this case security will depend on the security of that component.

Implementation stage: This stage requires reflecting in the code the security rules defined in the design stage. Because these rules are expressed as classes, associations, and constraints, they can be implemented as classes in object-oriented languages. In this stage one can also select specific security packages or COTS, e.g., a firewall product or a cryptographic package. Some of the patterns identified earlier in the cycle can be replaced by COTS (these can be tested to see if they include a similar pattern). Performance aspects become now important and may require iterations. As indicated, a whole DBMS could be such component.

An important aspect for the complete design is assurance. Experience shows that one can verify each pattern used but this does not in general verify their combination. One can however still argue that since one has used a careful and systematic methodology with verified and tested patterns, the design should provide a good level of security. The set of patterns can be shown to be able to stop or mitigate the identified threats.

Secure Database Development using UMLsec

A general methodology for developing security-critical software which in particular can be used to develop secure DBMSs has been proposed in [13]. It makes use of an extension of the Unified Modeling Language (UML) to include security-relevant information, which is called UMLsec. The approach is supported by extensive automated tool-support for performing a security analysis of the UMLsec models against the security requirements that are included [14] and has been used in a variety of industrial projects [3]. The UMLsec extension is given in form of a UML profile using the standard UML extension mechanisms. Stereotypes are used together with tags to formulate the security requirements and assumptions. Constraints give criteria that determine whether the requirements are met by the system design, by referring to a precise semantics of the used fragment of UML. The security-relevant information added using stereotypes includes security assumptions on the physical level of the system, security requirements related to the secure handling and communication of data, and security policies that system parts are supposed to obey. The UMLsec tool-support can be used to check the constraints associated with UMLsec stereotypes mechanically, based on XMI output of the diagrams from the UML drawing tool in use. There is also a framework for implementing verification routines for the constraints associated with the UMLsec stereotypes. Thus advanced users of the UMLsec approach can use this framework to implement verification routines for the constraints of self-defined stereotypes. The semantics for the fragment of UML used for UMLsec is defined using so-called UML Machines, which is a kind of state machine which is equipped with UML-type communication mechanisms. On this basis, important security requirements such as secrecy, integrity, authenticity, and secure information flow are defined.

Applications Including Secure Databases

Since this approach is tailored to the application, one can add the required level of security using formal proofs when necessary. Specialized operating system and hardware are also possible and may be needed to reach the required level of security. High-security systems require faithful application of basic security principles; for example, multilevel databases apply complete mediation. Databases work through transactions and a concurrency control system serializes transactions to prevent inconsistencies. High-security multilevel databases also require that the concurrency control system preserves security. The methods described in the last section still apply here, except that additional requirements must be considered. Because of this, these approaches are discussed in less detail, describing only two recent papers that contain references to past work.

Designing Secure Databases using OCL

An approach to designing the content of a security-critical data base uses the Object Constraint Language (OCL) which is an optional part of the Unified Modeling Language (UML). More specifically, [8] presents the Object Security Constraint Language V.2. (OSCL2), which is based in OCL. This OCL extension can be used to incorporate security information and constraints in a Platform Independent Model (PIM) given as a UML class model. The information from the PIM is then translated into a Platform Specific Model (PSM) given as a multilevel relational model. This can then be implemented in a particular Database Management System (DBMS), such as Oracle9*i* Label Security. These transformations can be done automatically or semi-automatically using OSCL2 compilers. Related to this, [9] presents a methodology that consists of four stages: requirements gathering; database analysis; multilevel relational logical design; and specific logical design. Here, the first three stages define activities to analyze and design a secure database. The last stage consists of activities that adapt the general secure data model to one of the most popular secure database management systems: Oracle*9i* Label Security. They later extended the approach to data warehouses, multidimensional databases, and on-line analytical processing applications.

In both cases, a particular multilevel database system, meaning a set of users organized in levels, compartments, and groups, is given access to specific items of a relational database, according to the characteristics of those items, which also include levels, compartments, and groups. A set of rules describes the allowed access of users to data items. The secure metamodel is stored in the labels of each row or user definition. As indicated, the extra requirements can be superimposed in a general secure software development methodology.

Other Approaches to Secure Software Development with Applicability to Databases

There are other approaches to developing security-critical software which can be applied to developing secure databases and database management systems.

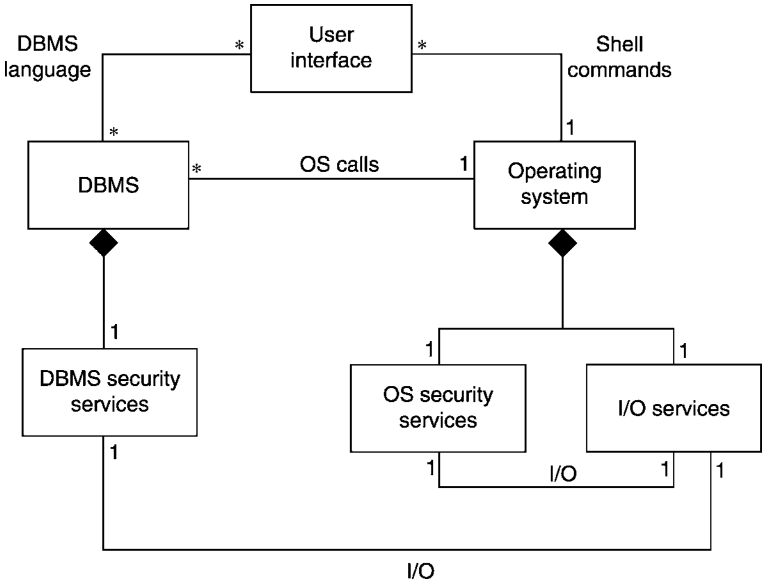
[12] presents an approach for the predicative specification of user rights in the context of an object oriented use case driven development process. It extends the specification of methods by a permission section describing the right of some actor to call the method of an object. The syntactic and semantic framework is first-order logic with a built-in notion of objects and classes provided with an algebraic semantics. The approach can be realized in OCL.

[1] presents an approach to building secure systems where designers specify system models along with their security requirements and use tools to automatically generate system architectures from the models, including complete, configured access control infrastructures. It includes a combination of UML-based modeling languages with a security modeling language for formalizing access control requirements.

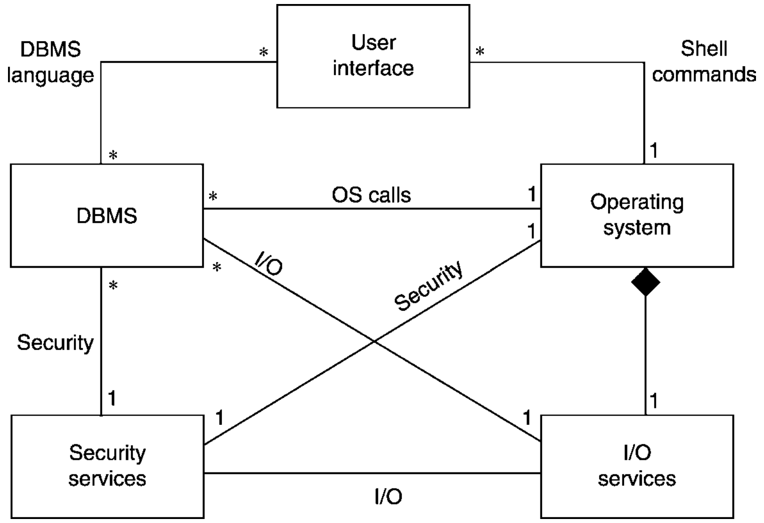
[15] presents an approach based on the high-level concepts and modeling activities of the secure Tropos methodology and enriched with low level security-engineering ontology and models derived from the UMLsec approach.

System Architecture for Security

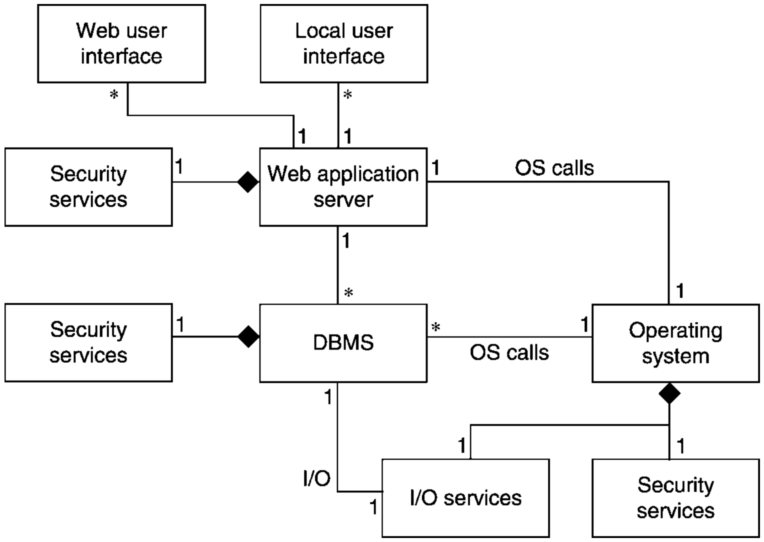
Whichever approach is used, there are basically three general architectural configurations to include security functions:



Secure Database Development. Figure 1. Standard placement of security services.



Secure Database Development. Figure 2. Common security services.



Secure Database Development. Figure 3. Architecture using a Web Application Server.

1. Figure 1 shows the standard approach. Here the DBMS and the operating system have their own set of security services.
2. Figure 2 shows a way to unify the design of the DBMS with the design of the OS, using an I/O and file subsystem and a security subsystem to be used by both the DBMS and the OS.
3. Figure 3 is an extension of the standard approach where a Web Application Server (WAS) unifies security for several databases. The WAS applies a common conceptual model to the information and can integrate different types of databases.

These configurations can be used in either of the approaches discussed earlier. Within each configuration it is possible to use security kernels and virtual machines.

Key Applications

Clearly, the first approach makes sense when the objective is a secure DBMS product, since it is not possible to know what user applications will be supported in the future. The only choice is then to build a system which is as secure as possible within these constraints and within a reasonable cost.

In the second case, the type of application to be supported is known. This gives the designers the flexibility of choosing an appropriate existing database system, as done in [9], or to build the DBMS to reach the required degree of security. If the complete DBMS is to be built, the first approach is appropriate, using as parameter the degree of security.

Cross-references

▶ Access Control Administration Policies
▶ Access Control Policy Lauguages
▶ Architecture-Conscious Database System
▶ Application Server
▶ Authentication
▶ Authorization
▶ Concurrency Control – Traditional Approaches
▶ Data Stream Management Architectures and Prototypes
▶ Data Warehouse Life-Cycle and Design
▶ Data Warehouse Security
▶ Database Design
▶ Database Security
▶ DB Middleware
▶ DBMS
▶ Discretionary Access Control
▶ Distributed Database Design
▶ Distributed Database Systems
▶ Distributed DBMS
▶ Mandatory Access Control
▶ Metamodel
▶ Object Constraint Language
▶ Object Data Models
▶ Object-Role Modeling
▶ Privacy
▶ Process Life Cycle
▶ Process Structure of a DBMS
▶ Role Based Access Control

Recommended Reading

1. Basin D.A., Doser J., and Lodderstedt T. Model driven security: from UML models to access control infrastructures. ACM Trans. Softw. Eng. Methodol., 15(1):39–91, 2006.
2. Bertino E. and Sandhu R. Database security – Concepts, approaches, and challenges. IEEE Trans. Dependable Sec. Comput., 2(1):2–19, 2005.
3. Best B., Jurjens J., and Nuseibeh B. Model-based security engineering of distributed information systems using UMLsec. In Proc. 29th Int. Conf. on Software Eng., 2007, pp. 581–590.
4. Castano S., Fugini M., Martella G., and Samarati P. Database Security. Addison-Wesley, 1994.
5. Fernandez E.B., Gudes E., and Song H. A model for evaluation and administration of security in object-oriented databases. IEEE Trans. Knowl. Database Eng., 6(2):275–292, 1994.
6. Fernandez E.B., Larrondo-Petrie M.M., Sorgente T., and VanHilst M. A methodology to develop secure systems using patterns, Chapter V. In Integrating Security and Software Engineering: Advances and Future Vision, H. Mouratidis, P. Giorgini (eds.). IDEA Press, 2006, pp. 107–126.
7. Fernandez E.B., Summers R.C., and Wood C. Database Security and Integrity (Systems Programming Series). Addison-Wesley, 1981.
8. Fernández-Medina E. and Piattini M. Extending OCL for secure database development. In Proc. Int. Conf. on the Unified Modeling Language, 2004, pp. 380–394.
9. Fernández-Medina E. and Piattini M. Designing secure databases. Inf. Softw. Technol., 47(7):463–477, 2005.
10. Fugini M. Secure database development methodologies. In Database Security: Status and Prospects, C.E. Landwehr (ed.). Elsevier, 1987, pp. 103–129.
11. Ge X., Polack F., and Laleau R. Secure Databases: an Analysis of Clark-Wilson Model in a Database Environment. In Proc. 16th Int. Conf. on Advanced Information Systems Eng., 2004, pp. 234–247.
12. Hafner M. and Breu R. Towards a MOF/QVT-Based Domain Architecture for Model Driven Security. In Proc. 9th Int. Conf. Model Driven Eng. Lang. and Syst., 2006.
13. Jurjens J. Secure Systems Development with UML. Springer, New York, 2004.
14. Jurjens J. Sound methods and effective tools for model-based security engineering with UML. In Proc. 27th Int. Conf. on Software Eng., 2005, pp. 322–331.
15. Mouratidis H., Jürjens J., and Fox J. Towards a comprehensive framework for secure systems development. In Proc. 18th Int. Conf. on Advanced Information Systems Eng., 2006, pp. 48–62.

Secure Database Systems

▶ Multilevel Secure Database Management Systems

Secure Datawarehouses

▶ Data Warehouse Security

Secure DBMS Development

▶ Secure Database Development

Secure Hardware

▶ Trusted Hardware

Secure Multiparty Computation Methods

Murat Kantarcioğlu[1], Jaideep Vaidya[2]
[1]University of Texas at Dallas, Richardson, TX, USA
[2]Rutgers University, Newark, NJ, USA

Definition

The problem of preserving privacy while allowing data analysis can be attacked in many ways. One way is to avoid disclosing data beyond its source while still constructing data mining models equivalent to those that would have been learned on an integrated data set. This follows the approach of Secure Multiparty Computation (SMC). SMC refers to the general problem of computing a given function securely over private inputs while revealing nothing extra to any party except what can be inferred (in polynomial time) from its input and output. Since one can prove that data are not disclosed beyond its original source, the opportunity for misuse is not increased by the process of data mining.

The definition of privacy followed in this line of research is conceptually simple: no site should learn anything new from the *process* of data mining. Specifically, anything learned during the data mining process must be derivable given one's own data and the final result. In other words, nothing is learned about any other site's data that is not inherently obvious from the data mining result. In the context of data mining, the approach followed in this research has been to select a type of data mining model to be learned and develop a protocol to learn the model while meeting this definition of privacy.

Historical Background

Privacy-preserving data mining can be defined as the problem of how to mine data when it is not possible to see it. Two seminal papers [1,9] first considered this problem and proposed different ways to attack it. Both looked at the problem of constructing decision trees from distributed data in a privacy-preserving manner. Agrawal and Srikant [1] proposed a randomization approach based on perturbing the input data and reconstructing the distribution. Lindell and Pinkas [9] proposed a cryptographic solution based on secure multiparty computation. This entry describes the second approach following the application of secure multiparty computation methods to data mining.

Secure Multiparty Computation (SMC) originated with Yao's Millionaires' problem [15]. The basic problem is that two millionaires would like to know who is richer, with neither revealing their net worth. Abstractly, the problem is to simply compare two numbers, each held by one party, without either party revealing its number to the other. Yao [15] presented a generic circuit evaluation based solution for this problem as well as generalizing it to any efficiently computable function restricted to two parties. Goldreich et al. [6] generalized this to multi-party computation and proved that there exists a secure solution for any functionality. There has been significant theoretical work in this area. The restriction of polynomially time bounded passive adversaries has been removed. Similarly, work has been extended to active adversaries, as well as mobile adversaries. While much effort has been due to efficiency reasons, it is completely infeasible to directly apply the theoretical work from SMC to form secure protocols for privacy-preserving data mining.

Thus, work in privacy-preserving data mining has focused on creating specialized efficient solutions in the context of data mining. Starting with the work of Lindell and Pinkas [9], secure methods have been proposed for various tasks such as association rule mining [7,13], clustering [8], classification [2], and outlier detection [12]. [14] gives a good overview of much of this work.

Foundations

The basic ideas used in SMC based privacy-preserving data mining techniques are now illustrated using a commonly deployed public key encryption technique called homomorphic encryption [11]. More formally, let $E_{pk}(.)$ denote the encryption function with public key pk and $D_{pr}(.)$ denote the decryption function with private key pr. A secure public key cryptosystem is called additively homomorphic if it satisfies the following requirements: (i) Given the encryption of m_1 and m_2, $E_{pk}(m_1)$ and $E_{pk}(m_2)$, there exists an efficient algorithm to compute the public key encryption of $m_1 + m_2$, denoted $E_{pk}(m_1 + m_2) := E_{pk}(m_1) +_h E_{pk}(m_2)$. (ii) Given a constant k and the encryption of m_1, $E_{pk}(m_1)$, there exists an efficient algorithm to compute the public key encryption of km_1, denoted $E_{pk}(km_1) := k \times_h E_{pk}(m_1)$.

Using the homomorphic encryption technique, one can easily develop many secure protocols. For example, consider the case where three sites S_1, S_2 and S_3 want to add their private values (resp.) v_1, v_2, and v_3 to learn $v_1 + v_2 + v_3$ securely. A simple protocol for the above task using homomorphic encryption can be given as follows: S_1 creates a homomorphic encryption public and private key pair, and sends the public key to S_2 and S_3. In addition, S_1 computes $e_1 = E_{pk}(v_1)$ and sends e_1 to S_2. Using the homomorphic encryption scheme, S_2 can calculate $e_2 = e_1 +_h E_{pk}(v_2)$ and can send e_2 to S_3. Similarly, S_3 can calculate $e_3 = e_2 +_h E_{pk}(v_3) = E_{pk}(v_1 + v_2) + E_{pk}(v_3)$. Finally, S_1 can decrypt the e_3 to compute $D_{pr}(e_3) = v_1 + v_2 + v_3$. If all the parties follow the protocol exactly, it can be shown that nobody learns anything other than the final result. The obvious question is what happens when the parties do not follow the protocol exactly. Clearly S_3 can collaborate with S_1 to learn the private value v_2 because if S_3 sends the message e_2 to S_1 then S_1 can compute $D_{pr}(e_2) - v_1 = (v_1 + v_2) - v_1 = v_2$ to learn v_2.

The above example indicates that when considering privacy, one must first model the different adversarial behaviors that an attacker can assume. The SMC literature defines two basic adversarial models:

Semi-Honest: Semi-honest (or Honest but Curious) adversaries follow the protocol faithfully, but can try to infer the secret information of the other parties from the data they see during the execution of the protocol.

Malicious: Malicious adversaries may do anything to infer secret information. They can abort the protocol at any time, send spurious messages, spoof messages, collude with other (malicious) parties, etc.

While the semi-honest model may seem questionable for privacy (if a party can be trusted to follow the protocol, why would they not be trusted with the data?), it does meet several practical needs for early adoption of the technology. Consider the case where credit card companies jointly build data mining models for credit card fraud detection. In many cases the parties involved already have authorization to see the data (e.g., the theft of credit card information from CardSystems involved data that CardSystems was expected to see during processing). The problem is that *storing* the data brings with it a responsibility (and cost) of protecting that data; CardSystems was supposed to delete the information once the processing was complete. If parties could develop the desired models without seeing the data, then they are saved the responsibility (and cost) of protecting it. Also the simplicity and efficiency possible with semi-honest protocols will help speed adoption so that trusted parties are saved the expense of protecting data other than their own. As the technology gains acceptance, malicious protocols will become viable for uses where the parties are not mutually trusted.

In either adversarial model, there exist formal definitions of privacy [5]. Informally, the definition of privacy is based on equivalence to having a trusted third party perform the computation. This is the gold standard of secure multiparty computation. Imagine that each of the data sources gives their input to a (hypothetical) trusted third party. This party, acting in complete isolation, computes the results and reveals them. After revealing the results, the trusted party forgets everything it has seen. A secure multiparty computation approximates this standard: no party learns more than it would in the trusted third party approach.

One fact is immediately obvious: no matter how secure the computation, some information about the inputs may be revealed. This is a result of the computed function itself. For example, if one party's net worth is $100,000, and the other party is richer, one has a lower bound on their net worth. This is captured in the formal SMC definitions: any information that can be inferred from one's own data and the result can be revealed by the protocol. Thus, there are two kinds of information leaks; the information leak from the function computed irrespective of the process used to compute the function and the information leak from the specific process of computing the function. Whatever is leaked from the function itself is unavoidable as long as the function has to be computed. In secure computation, the second kind of leak is provably prevented. There is *no* information leak whatsoever due to the process.

While the generic secure multi-party computation methods exist, they pose significant computational problems. The challenge of privacy-preserving distributed data mining is to develop algorithms that have reasonable computation and communication costs on real-world problems, and prove their security with respect to the SMC definition. The typical approach taken is to reduce the large domain problem to a series of smaller sub-tasks and to use secure cryptographic protocols to implement those smaller sub-tasks.

In the following, the common secure sub-protocols used in privacy-preserving distributed data mining are now described. As far as possible, for each sub-protocol, a version using only homomorphic encryption is described. Unless otherwise stated, all the sub-protocols are secure in the semi-honest model with no collusion, and all the arithmetic operations are defined in some large enough finite field.

Following the description of the subprotocols, it is shown how different algorithms could be implemented using these secure sub-protocols. Since these common building blocks are quite general, using the Composition theorem [5], they can be combined to create new privacy preserving algorithms in the future.

Secure Sum

Secure Sum securely calculates the sum of values from individual sites. As seen above, homomorphic encryption can easily be used to secure compute the sum local values. Assuming three or more parties and no collusion, a more efficient method can be found in [7].

Secure Comparison / Yao's Millionaire Problem

Assume that two sites, each having one value, want to compare the two values without revealing anything else other than the comparison result. Secure Comparison methods can be used to solve the above problem. To the best of our knowledge, secure circuit evaluation based approaches still provide the best performance [15].

Dot Product Protocol

Securely computing the dot product of two vectors is another important sub-protocol required in many privacy-preserving data mining tasks. Many secure dot product protocols have been proposed in the past. Among those proposed techniques, the method of Goethals et al. [4] is quite simple and provably secure. It is now briefly described.

The problem is defined as follows: Alice has a n-dimensional vector $\vec{X} = (x_1,\ldots,x_n)$ while Bob has a n-dimensional vector $\vec{Y} = (y_1,\ldots,y_n)$. At the end of the protocol, Alice should get $r_a = \vec{X} \cdot \vec{Y} + r_b$ where r_b is a random number chosen from uniform distribution that is known only to Bob, and $\vec{X} \cdot \vec{Y} = \sum_{i=1}^{n} x_i \cdot y_i$. The key idea behind the protocol is to use a homomorphic encryption system that can be used to perform arithmetic operations over encrypted data. Using such a system, it is quite simple to build a dot product protocol. If Alice encrypts her vector and sends in encrypted form to Bob, using the additive homomorphic property, Bob can compute the dot product. The specific details can be found in [4].

Oblivious Evaluation of Polynomials

Another important sub-protocol required in privacy-preserving data mining is the secure polynomial evaluation protocol. Consider the case where Alice has a polynomial P of degree k over some finite field $\mathcal{F}$. Bob has an element $x \in \mathcal{F}$ and also knows k. Alice would like to let Bob compute the value $P(x)$ in such a way that Alice does not learn x and Bob does not gain any additional information about P (except $P(x)$). This problem was first investigated by [10]. Subsequently, there have been more protocols improving the communication and computation efficiency as well as extending the problem to floating point numbers.

Privately Computing ln *x*

For entropy measures used in data mining, one must be able to privately compute ln x, where $x = x_1 + x_2$ with x_1 known to Alice and x_2 known to Bob. Thus, Alice should get y_1 and Bob should get y_2 such that $y_1 + y_2 = \ln x = \ln(x_1 + x_2)$. One of the key results presented in [9] was a cryptographic protocol for this computation. One point to note is that ln x is *Real* while general cryptographic tools work over finite fields. Therefore, ln x is actually multiplied with a known constant to make it integral. The basic idea behind computing random shares of $\ln(x_1 + x_2)$ is to use the Taylor approximation for ln x. Thus, shares for the Taylor approximation are actually computed. The actual details of the protocol, as well as the proof of security, can be found in [9].

Secure Intersection

Secure Intersection methods are useful in data mining to find common rules, frequent itemsets etc., without revealing the owner of the item. Many algorithms have been developed for calculating Secure Set Intersection. For example, [13] provides an efficient solution. However, a secure set intersection protocol that utilizes secure polynomial evaluation [3] is described below. Let us assume that Alice has set $X = \{x_1,\ldots,x_n\}$ and Bob has set $Y = \{y_1,\ldots,y_n\}$. Our goal is to securely calculate $X \cap Y$. By representing set X as a polynomial and using polynomial evaluation, Alice and Bob can calculate $X \cap Y$ securely.

Secure Set Union

Secure union methods are useful in data mining to allow each party to give its rules,decision trees etc. without revealing the owner of the item. Union of items can be easily evaluated using SMC methods if the domain of the items is small. Each party creates a binary vector (where the ith entry is 1 if the ith item is present locally). At this point, a simple circuit that *or's* the corresponding vectors can be built and securely evaluated using general secure multi-party circuit evaluation protocols. However, in data mining, the domain of the items are usually very large, potentially infinite. This problem can be overcome using approaches based on commutative encryption [7].

Key Applications

This section overviews how different sub-protocols described above could be used to create various privacy-preserving distributed data mining (PPDM) algorithms for different data models. In each of the discussed PPDM algorithms general data mining functionality is reduced to a computation of secure sub-protocols.

In the following discussion, horizontal partitioning of data implies that different sites collect the same set of information about different entities. Vertical data partitioning implies that different sites collect different features of information for the same set of entities. While this entry does not explicitly discuss arbitrary partitioning, the building blocks presented above are actually useful even in that case.

Classification

In the first work on privacy-preserving distributed data mining on horizontally partitioned data [9], the goal is to securely build an ID3 decision tree where the training set is horizontally distributed between two parties. The basic idea is that finding the attribute that maximizes information gain is equivalent to finding the attribute that minimizes the conditional entropy. The conditional entropy for an attribute for two parties can be written as a sum of the expression of the form $(v_1 + v_2) \times \log(v_1 + v_2)$. The authors use the secure log algorithm, secure polynomial evaluation, and secure comparison sub-protocols to securely calculate the expression $(v_1 + v_2) \times \log(v_1 + v_2)$ and show how to use this function for building the ID3 securely. Correspondingly, decision trees for vertically partitioned data can also be built if the attribute with maximum entropy gain can be found. If the class attribute is present with all parties, this can be easily done. But even when the class attribute is only present with one of the parties, the secure scalar product protocol can be used to compute counts of transactions having certain attribute values and class values. This can then be used to compute the information gain for the attribute, and thus to decide the best attribute. Naïve Bayes classifiers can also be built for both horizontally and vertically partitioned data using combinations of the secure sum, secure scalar product and secure comparison primitives. [14] provides more details.

Association Rule Mining

The essential problem in association rule mining is the problem of finding frequent itemsets (meeting some support threshold). Once frequent itemsets are found, it is easy to find association rules meeting certain confidence thresholds. For horizontally partitioned data, [7] showed that for every candidate itemset, the support at each site can be computed locally. Now, a secure sum followed by a secure comparison is sufficient to evaluate if a candidate itemset is indeed frequent. However, this still requires the knowledge of the candidate itemsets. [7] uses some additional techniques (such as Secure Union) to ensure that candidate itemsets contributed by each site are also kept secret along with their support values. Similarly, [13] shows that the problem of finding frequent itemsets in vertically partitioned data can be reduced to the problem of securely computing the scalar product of multiple vectors (or equivalently as the problem of finding the size of the intersection set). Once this is done, finding globally valid association rules is quite simple.

Clustering

Several solutions for privacy-preserving clustering have been proposed. Lin et al. [8] propose a privacy preserving EM algorithm for secure clustering of horizontally partitioned data. EM clustering is an iterative algorithm. Each iteration consists of an expectation (E) step followed by a maximization (M) step. In the E-step, the expected value of the cluster membership for each entity is determined. In the M-step, the each cluster distribution parameters are re-estimated to maximize the likelihood of the data, given the expected estimates of the membership. Lin et al. [8] show that computing the cluster parameters at each iteration can be easily done via secure summation once the total number of objects is known. Once computed, the cluster parameters are assumed to be public (i.e., known to all parties). Therefore each party can then locally assign its entities to the appropriate clusters. This is repeated until the algorithm converges or until a sufficient number of iterations have been carried out. Similar solutions exist for vertically partitioned data as well.

Outlier Detection

The goal of outlier detection is to find anomalies or outliers in the data. This requires a definition/metric of outlyingness. Many such metrics (with varying degrees of sophistication) have been defined in the statistical literature. One of the simplest metrics is that of $DB(p,d)$ outliers. Under this definition, an entity e in the dataset DB is said to be an outlier if more than p percentage of the entities in the dataset DB are farther than distance d from e. Thus, to figure out if an entity is an outlier, several tasks need to be performed: first, the distance of this entity to other entities must be computed; next, one must check if the number of farther entities is more than the given threshold. Vaidya and Clifton [12] show how to do this for both horizontally and vertically partitioned data. The key primitives used are the secure sum, secure comparison and secure dot product primitives. More detail can be found in [12].

Future Directions

Now that the key concepts behind secure multiparty computation methods have been presented, it is necessary to discuss some of the problems and challenges still open in this area. Inherently, the primary challenge with secure multiparty computation techniques lies with efficiency. Even with cryptographic accelerators and faster machines, since data mining is typically done over millions of transactions, this cost significantly balloons up. Even for other application areas, more efficient protocols are clearly needed. One alternative that has not been well explored is that of approximation. Instead of computing the exact results, it may make a lot more sense to compute approximations of the final results, especially if it gives huge efficiency improvements. This will be critical for development of real solutions in this area.

Cross-references

- ► Horizontally Partitioned Data
- ► Privacy-Preserving Data Mining
- ► Vertically Partitioned Data

Recommended Reading

1. Agrawal R. and Srikant R. Privacy-preserving data mining. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 439–450.
2. Du W. and Zhan Z. Building decision tree classifier on private data. In C. Clifton and V. Estivill-Castro (eds.). IEEE Int. Conf. on Data Mining Workshop on Privacy, Security, and Data Mining, 2002, pp. 1–8.
3. Freedman M.J., Nissim K., and Pinkas B. Efficient private matching and set intersection. In Proc. Int. Conf. Theory and Application of Cryptographic Techniques, 2004.
4. Goethals B., Laur S., Lipmaa H., and Mielikäinen T. On secure scalar product computation for privacy-preserving data mining. In Proc. the Seventh Annual Int. Conf. in Information Security and Cryptology, 2004, pp. 104–120.
5. Goldreich O. The Foundations of Cryptography, vol. 2, General Cryptographic Protocols. Cambridge University Press, London, 2004.
6. Goldreich O., Micali S., and Wigderson A. How to play any mental game – a completeness theorem for protocols with honest majority. In Proc. 19th ACM Symp. on the Theory of Computing, 1987, pp. 218–229.
7. Kantarcoğlu M. and Clifton C. Privacy-preserving distributed mining of association rules on horizontally partitioned data. IEEE Trans. Knowl. Data Eng., 16(9):1026–1037, 2004.
8. Lin X., Clifton C., and Zhu M. Privacy preserving clustering with distributed EM mixture modeling. Knowl. Inf. Syst., 8(1):68–81, 2005.
9. Lindell Y. and Pinkas B. Privacy preserving data mining. J. Cryptol., 15(3):177–206, 2002.
10. Naor M. and Pinkas B. Oblivious transfer and polynomial evaluation. In Proc. Thirty-First Annual ACM Symp. on Theory of Computing, 1999, pp. 245–254.
11. Paillier P. Public-key cryptosystems based on composite degree residuosity classes. In Proc. Int. Conf. Theory and Application of Cryptographic Techniques, 1999, pp. 223–238.
12. Vaidya J. and Clifton C. Privacy-preserving outlier detection. In Proc. 2004 IEEE Int. Conf. on Data Mining, 2004, pp. 233–240.
13. Vaidya J. and Clifton C. Secure set intersection cardinality with application to association rule mining. J. Comput. Security, 13(4):593–622, November 2005.
14. Vaidya J., Clifton C., and Zhu M. Privacy-Preserving Data Mining, vol. 19 of Advances in Information Security, 1st edn. Springer, Berlin, 2005.
15. Yao A.C. How to generate and exchange secrets. In Proc. 27th IEEE Symp. on Foundations of Computer Science, 1986, pp. 162–167.

S

Secure Third-Party Data Management

► Secure Data Outsourcing

Secure Transaction Processing

Indrakshi Ray
Colorado State University, Fort Collins, CO, USA

Definition

Secure transaction processing refers to execution of transactions that cannot be exploited to cause security breaches.

Historical Background

Research in making transaction processing secure has progressed along different directions. Most of the early research in this area focused in processing multilevel transactions suitable for military applications. Such applications are characterized by having a set of security levels which are partially ordered using the dominance relation. The requirement is that information can flow from a dominated level to a dominating level but all other flows are considered to be illegal. The traditional concurrency control and recovery algorithms cannot be used for processing transactions in military applications because they cause illegal information flow. Most research in this area involved developing architectures, concurrency control and recovery mechanisms to prevent illegal information flow. Subsequently, researchers have also looked into the problem of processing real-time secure transactions. These transactions must satisfy real-time requirements together with preventing illegal information flow.

In the commercial sector, subsequent research focused on how to deal with the effect of malicious transactions that may have compromised the integrity of the database. The malicious transactions can damage one or more data items. Other transactions reading from these committed data items help spread the damage. Traditional recovery mechanisms cannot undo the effects of committed transactions. The research in this area focused on developing efficient techniques that will remove the effects of malicious transactions while minimizing the impact on good transactions.

Foundations

A comprehensive survey on the work in multilevel secure (MLS) databases was performed by Atluri et al. [11]. Some of the important results discussed in this paper are enumerated. An MLS database environment is characterized by a set of security levels L that are partially ordered by the relation $\prec$. $\prec$ is the dominance relation between classes and it is transitive, reflexive and anti-symmetric. For any two levels, $l_i, l_j \in L$, if level $l_i \prec l_j$, then l_j is said to dominate l_i. In this case, l_j and l_i are referred to as dominating and dominated level respectively. If neither $l_i \prec l_j$, nor $l_j \prec l_i$, then l_i and l_j are said to be incomparable. Each data object o in an MLS an environment is associated with a security classification, denoted by $l(o)$, where $l(o) \in L$. Each user u is also cleared to some security level $l(u)$, where $l(u) \in L$. A user u cleared to security level $l(u)$ can log in at any security level l', where l' is dominated by $l(u)$. All processes, including transactions, initiated during a session inherit the security level at which the user has logged in.

MLS systems allow information to flow from dominated to dominating levels but all other information flows are considered illegal. Direct information flow occurs by virtue of transactions reading and writing data items. When a transaction reads a data item, information flows from the data item to the transaction. Similarly, when a transaction writes a data item, information flows from the transaction to the data item. Such direct illegal information flow is prevented by using the simple security property and the $\star$-property of the Bell-LaPadula (BLP) model [12]. Simple security property states the condition under which a transaction can read a data item. A transaction T may read a data item O only if the security level of the data item, denoted by $L(O)$, is dominated by the security level of the transaction, denoted by $L(T)$, that is, only when $L(O) \prec L(T)$. During the read operation, information flows from the data item to the transaction. Thus, when $L(O) \prec L(T)$, information flows from dominated level to the dominating one. $\star$-property states the condition under which a transaction T can write a data item O. The write operation is allowed only when the security level of the data object, denoted by $L(O)$, is dominated by that of the transaction, denoted by $L(T)$. In other words, the write operation is allowed only when $L(T) \prec L(O)$. In this case the information flows from the dominated level to the dominating level. However, the $\star$-property does not prevent a transaction operating at the dominated level from corrupting data items at the dominating level. Thus, for reasons of integrity, a modified form of $\star$-property, known as restricted $\star$-property, is used in practice. The restricted $\star$-property allows a transaction T to write a data item O only if the security levels of the transaction is the same as that of the object, that is, $L(T) = L(O)$. The properties stated in the BLP model are not adequate in preventing illegal information flows that occur through indirect means. One such example is the covert channels. A covert channel is an information flow mechanism within a system that is based on the use of system resources; it is not intended for communication between the regular users of the system. Unfortunately, the traditional transaction processing mechanisms can be exploited to establish a covert channel.

First, the discussion describes how the concurrency control protocols that are used in traditional transaction processing systems can be used to establish a covert channel. Two concurrency control mechanisms are considered: two-phase locking (2PL) and timestamp ordering (TO). 2PL requires transactions to acquire read lock (write lock) before reading (writing) a data item. A read lock on a data item can be acquired if no other transaction has a write lock on the same data item. A write lock can be acquired if no other transaction has any lock on the data item. The locks acquired by a transaction must eventually be released. Moreover, once a transaction releases a lock, it can no longer lock any other data item. TO requires each transaction t to have a unique timestamp $ts(t)$. Each data item x is associated with a read timestamp $rts(x)$ and a write timestamp $wts(x)$ that denotes the timestamp of the latest transaction that have read and written x respectively. The operations are executed on a first-come-first-serve basis. If the execution of an operation does not violate the serialization order specified by the timestamps of the transactions, it is executed. If not, the operation is not allowed and the transaction is aborted.

Suppose this MLS database system is associated with two security levels *low* and *high* where the level *high* dominates *low*. The transactions initiated by a *high* user can read all data items and write high data items in accordance with the security and the restricted-⋆ properties of the BLP model. The transactions initiated by a *low* user can read and write low data items only. Suppose there are two transactions T_h and T_l that have decided to collude and there are no other transactions executing in the system. The security level of T_h and T_l are high and low respectively. The database has a data item x whose security level is *low* - both transactions have decided to communicate by accessing this data item: T_h will read the data item x and T_l will write the data item. Assume that the concurrency control mechanism uses two-phase locking (2PL). When T_h wants to read data item x, a read lock is placed on x which prohibits T_l from acquiring a write lock on x. Thus, T_h can selectively issue lock request on x to transmit information. T_l can measure the delay in acquiring lock and interpret the information. Thus, a covert communication channel has been established between T_l and T_h. A similar problem occurs if a timestamp-based protocol is used. Suppose $ts(T_l) < ts(T_h)$. If T_l attempts to write x after T_h has read it, the write operation is rejected and T_l is aborted. Here again, a high transaction can selectively cause a low transaction to abort and communicate information. Thus, concurrency control mechanisms in an MLS database must not only ensure serializability but must also eliminate such illegal information flows.

Researchers have proposed several concurrency control algorithms for processing transactions in an MLS database. The algorithms are dependent on the underlying architecture of the DBMS. First, the discussion focuses on the algorithms developed for the kernelized architecture. Some of the early solutions proposed are by Schaefer [35], Lamport [22], and Reed and Kanodia [34]. In these solutions, the transactions are allowed to proceed. However, before a transaction can commit it is validated. Thus, if a transaction at the dominating level has read some data item which has been updated by a transaction at the dominated level, the dominating level transaction must abort. Such algorithms will cause starvation of transactions at the dominating levels. Keefe and Tsai [20] proposed a protocol based on multiversion timestamp ordering. Although this protocol ensures serializable histories without causing starvation of transactions at the dominating levels, it has several problems. First, it requires a large number of versions to be maintained. Second, transactions at the dominating levels read stale data. Third, performance is an issue. Subsequent research [2,4,27] focused on limiting the number of versions to two. The idea is that transactions reading at the dominated level read from the snapshot, while those writing data at their own level write it on the current state.

Researchers have also proposed solutions that are suitable for replicated architectures. In such architectures, an MLS DBMS is constructed from several single-level DBMSs. The DBMS at level l will contain a copy of every data item that a transaction at level l can access. The first protocol for this architecture was proposed by Jajodia and Kogan [17]. The protocol assumes that the set of security levels are totally ordered. Transactions are submitted to a global transaction manager (GTM) who is responsible for forwarding the transactions to the corresponding DBMSs. The updates made by the transactions must also be propagated to the DBMSs at the dominating levels. Two transactions that conflict at level l must be submitted at the dominating level l' in the order in which they commit. However, when transactions do not conflict, they are sent in an arbitrary order to the dominating levels. This may result in nonserializable histories as pointed out by Kang and Keefe [18]. Costich [14] improves upon the Jajodia-Kogan protocol in the following manner. First, it reduces the amount of trust required to implement GTM. Second, it does not require the security levels to form a total order. However, McDermott, Jajodia, and Sandhu [26], illustrate that certain security posets cause the protocol to deadlock and block update projections and produce non-serializable executions. Subsequently, researchers [3–5,18] have characterized the posets that create this problem. An example will help illustrate the problem. Suppose there are the following security levels: A, B, C, AB, BC, AC, and ABC. The dominance relationship between the levels is as follows: $A \prec AB$, $A \prec AC$, $B \prec AB$, $B \prec BC$, $C \prec AC$, $C \prec BC$, $AB \prec ABC$, $BC \prec ABC$, and $AC \prec ABC$. Three update transactions T_1, T_2 and T_3 are submitted at levels A, B, and C respectively. These transactions execute at the DBMS at these levels and are then propagated to dominating levels. Thus, T_1, T_2 must execute in the DBMS at level AB, T_2, T_3 must execute at level BC, and T_3, T_1 must execute at level AC. Suppose T_1 is serialized before T_2 in level AB, T_2 is serialized before T_3 in level BC, and T_3 is serialized before T_1 in level AC. Now these updates must be propagated to level ABC. The DBMS at level ABC must respect the serialization orders of the dominated levels. Here is a deadlock situation because the orders are conflicting. This problem is solved by ordering transactions according to the timestamps generated when the transactions commit for the first time and using a conservative TO protocol which ensures that update projections are never aborted.

The transactions discussed so far have a single security level associated with them and are termed single-level transactions. These transactions can read at multiple security levels but can update data at one security level only. The problem with single-level transactions is that they cannot preserve integrity constraints spanning multiple security levels. Thus, for these databases serializability is an overly restrictive correctness criterion [16,25]. Jajodia and Atluri [16] have proposed weaker notions of correctness, such as, levelwise serializability, one-item read serializability, and pairwise serializability for MLS databases having single-level transactions. Other researchers have proposed the notion of multilevel transactions. A multilevel transaction is associated with a set of security levels. It allows the transaction to read and write data items at multiple security levels. A multilevel transaction is composed of a set of subtransactions. Each subtransaction is associated with a single security level and performs operations following the simple security and restricted-⋆ property of the BLP model. Ideally, a multilevel transaction must have the properties of atomicity, consistency, isolation, and durability and it also should not cause any illegal information flow by virtue of its execution. However, it is often not possible to guarantee atomicity without causing illegal information flow [13,36]. Towards this end, Blaustein et al. [13] defines varying degree of atomicity that can be achieved by multilevel transactions. Ray, Ammann, and Jajodia [31] provide a notion of semantic atomicity which is suitable for multilevel transactions. The application containing the transactions are formally analyzed to give assurance of the satisfaction of this property. This work does not use serializability as the correctness criterion but uses the notion of semantic correctness.

Protocols for distributed transaction also may have to be modified for multilevel secure databases. Consider, for instance, the early prepare protocol that ensures atomicity for distributed transactions. When a transaction T_i is submitted, the coordinator decomposes it into subtransactions, say, T_{ij} and T_{ik}, which are distributed to the participants at sites j and k for processing. The participants execute the subtransactions and replies to the coordinator with a prepare/no vote. If all the sites have responded with a prepare vote, the transaction will be committed and the coordinator sends a commit message to all the participants. Now suppose that the transaction T_i is executing on a MLS database. Since T_i is a distributed transaction, it is possible that the subtransaction T_{ij} has finished its execution and entered the prepare state before T_{ik} completes. Some other subtransaction say T_{mj} at the dominated level may want to write a data item, say x, that has been read by T_{ij}. To prevent a covert channel, the lock on x must be released by T_{ij}. Since T_{ik} is executing at a different site, it is possible that T_{ik} will acquire a lock after T_{ij} releases the lock. This will violate the two phase locking rule and may result in non-serializable executions.

This motivated Atluri, Bertino, and Jajodia [8] to propose a new protocol called secure early prepare (SEP). The coordinator decomposes the transaction T_i into subtransactions T_{i1}, T_{i2},...,T_{in} and sends them together with their security levels to the participants. The participants on completing their work successfully responds with a yes vote as before. However, if the participant has read a data item at the dominated level, it also sends a read-low indicator bit. If none of the participants have read low data items, the transaction proceeds like the early prepare protocol. However, if at least one participant has read a data item at the dominated level, additional rounds of message are necessary. In such cases, the coordinator sends a confirm message to all participants who have read data items at dominated levels. If the participant has not released any locks so far, it responds with a confirmed message. Otherwise it responds with a non-confirmed message. When the coordinator receives confirmed messages from all the participants who have read data items at dominated levels, the coordinator sends a commit message to all the participants. Otherwise, an abort message is sent. In a subsequent work [8], the authors propose an optimization to SEP that avoids some unnecessary aborts caused by SEP and also reduces the number of messages. Ray et al. [32] also attempts to improve upon SEP – instead of aborting subtransactions that have read from low data items, it rolls back the subtransaction to an earlier savepoint and reexecutes it. On successful reexecution, the participant sends a yes message. When all subtransactions have responded with a yes message, the transaction is committed. Otherwise, it is aborted.

Some researchers [1,19,37] have also looked into secure real-time transaction processing. The goal in such systems is to prevent illegal information flow as well maintain the timing constraints required for real-time applications. When both security constraints and real-time constraints cannot be satisfied, some works [1,37] trade-off security in order to improve the performance. Others [15,19] do not compromise security for the sake of performance. George and Haritsa [15] propose a concurrency control mechanism in which data conflicts are resolved in favor of the dominated level. Within a given level, data conflicts are resolved in favor of the earliest transaction deadline. Kang et al. [19] improve upon the work presented by George and Haritsa [15] by providing guarantees on average/transient miss ratios. A separate work [30] describes a new concurrency control protocol, known as, multiversion locking protocol with freezing, for processing secure real-time transactions.

Researchers have also investigated the impact of multilevel security on extended transaction models, such as workflows. A workflow is characterized as having a set of tasks and dependencies specified between the tasks. In an MLS workflow, each task is associated with a single security level and is allowed to read and write data items provided they obey the BLP rules. However, the dependencies between tasks at different levels may cause illegal information flow. Towards this end, Atluri et al. [10] have proposed an approach that redesigns the dependencies such that illegal information flow does not occur. A separate work [9] argues about how mandatory and discretionary access controls can be enforced in a workflow.

Secure transaction processing in non-MLS database systems focused on survivability. Attacks will occur in spite of sophisticated prevention mechanisms. The issue is how to identify the attack, confine it, assess the damage caused by it, and repair the damage in a timely manner. One of the early works in damage detection and recovery is by Ammann et al. [7]. After an attack occurs, the data items are marked with different colors to indicate the severity of the damaged. The authors define a notion of consistency for databases in which some data may have been damaged. Clean data must satisfy the integrity constraints defined over them. Damaged data must satisfy a set of relaxed integrity constraints. The authors classify transactions into three categories: *attack transactions*, *normal transactions* and *countermeasure transactions*. Attack transactions damage data items. Normal transactions sometimes help spread the damage. The normal transaction access protocol determines how the damage is propagated and ensures that the database satisfy the consistency constraints. The countermeasure transactions detect and repair the effects of an attack. These transactions must execute as trusted processes. Detection transactions change the marking of data items whereas repair transactions alter the value of data items. When an attack has occurred, the state of the database before the execution of attack transactions can be retrieved using snapshots. Towards this end, the paper proposes a technique by which snapshots can be generated while the database is servicing normal transactions.

Survivability issues and repair from malicious attacks have received attention in the database context. Ammann et al. [6] propose repair algorithms for traditional database systems that help to recover from the damage caused by malicious transactions. A two pass static algorithm is proposed where the first pass scans the log forward to locate all malicious and suspect tasks and the second pass goes backward from the end of the log to undo all malicious and suspect tasks. They also proposed a dynamic repair algorithm that continues to accept new transactions while repair is taking place. Panda et al. [21] have also proposed a number of algorithms on damage assessment and repair; some of these store the dependency information in separate structures so that the log does not have to be traversed for damage assessment and repair. Ray et al. [33] improve upon the time taken to assess the damage by using a dependency graph to store the dependencies and using depth-first search to retrieve the affected transactions. Liu and Jajodia [24] present a multi-phase damage confinement model. In the initial confinement phase, an estimation is done with respect to the damaged items. The estimation may not be accurate. The authors propose several schemes about damage confinement. The first one maintains timestamps. The initial confinement confines all data items that were updated after the commitment of the bad transactions. A damage assessor unconfines data items that are written by transactions not dependent on the bad transactions. This simple scheme causes damage leakage because a data item unconfined by an unaffected transaction can be updated by an affected transaction. Thus, even temporarily releasing this data item can cause damage spreading. The second scheme takes care of this problem. Confining the data items and later unconfining them usually takes some time. To reduce the relaxation latency, the authors propose a third scheme which uses transaction access patterns to unconfine data items that were not affected by the bad transaction. Panda and Giordano [28] provide two techniques for performing damage assessment and recovery. The first algorithm does detection and recovery simultaneously and is not very efficient. Moreover, new transactions are blocked until recovery is complete. These two shortcomings are removed in the second algorithm. Most of the work on damage assessment are based on transaction dependency approach. In these approaches, the goal is to identify affected transactions which must be undone and then re-executed. However, this may involve unnecessary undoing and redoing of operations. This is because not all operations of an affected transaction are influenced by a bad transaction. Towards this end, Panda and Haque [29] propose a damage assessment technique based on data dependency approach – only the affected operations are undone and redone.

Damage assessment in distributed databases has also been studied by several researchers. Liu and Hao [23] propose a damage assessment technique for distributed database that incurs a high communication overhead. Zuo and Panda [40] propose two approaches for damage assessment in distributed databases. The first one is a peer-to-peer approach which does not require a coordinator to perform damage assessment. When a site knows that it has some global affected transactions, it sends a multicast message to other sites which were involved with these global transactions. The other sites on receiving this message may identify some more global affected transactions which are then broadcast. The process continues until no more new global affected transactions are detected. This approach incurs high communication overhead. The other approach requires a coordinator. There are three variants of this other approach. In the first one, known as receive and forward, the coordinator keeps information about all global transactions. Each site manager sends a list of global affected transaction identifiers to the coordinator. The coordinator informs the other sites of these affected transactions. The other sites check whether or not any new transactions are affected. If not, a clear message is sent to the coordinator. Otherwise, the identity of the affected transactions are sent. The process stops when all the other sites send a clear message. The second coordinator-based approach incurs less communication overhead. When a malicious transaction is identified, the coordinator requests other sites for their local dependency graphs. The coordinator builds a global dependency graph using this information. The global graph is used for identifying affected transactions. The third coordinator-based approach relies on sites sending their graphs periodically to the coordinator. In this approach, the local graphs are not merged.

Yu, Liu and Zang [38] describe an algorithm for on-line attack recovery of workflows. The algorithm tries to build the list of redo and undo tasks, after an independent Intrusion Detection System reports malicious tasks. They also relax the restriction of executing order that exist in an attack recovery system; they introduced multi-version data objects to reduce unnecessary blocks in order to reduce degradation of performance in recovery. However, like in most existing papers, the authors only pay attention to restore consistency for data objects, while they do not analyze the correct actions needed in repair for different control-flow dependencies. In repair for advanced transactions, one needs to ensure that constraints of all control-flow dependencies are satisfied, and one should not treat all types of control-flow dependencies in the same manner – need to distinguish different types of control-flow dependencies and adopt different treatment to enforce these dependencies during repair. This issue is addressed in a subsequent work [39].

Key Applications

Critical information, such as health records, financial records are stored in the databases of an organization. Organizations must also interact with each other and share critical information to accomplish a particular mission. Security and privacy breaches involving such critical information have disastrous consequences. Thus, there is a need to formalize the concept of secure information flow both in the context of military and commercial databases. Ideally, mechanisms that enforce the secure information flow policies are needed. Automated capabilities that will track information flow and detect and thwart illegal flows are also needed. Since it is impossible to protect against all kinds of security and privacy breaches, it is also necessary to design systems that can automate to the extent possible the detection and repair from an attack.

Cross-references

- ▶ Data Confidentiality
- ▶ Data Integrity Services
- ▶ Information Flow
- ▶ Mandatory Access Control
- ▶ Multilevel Secure Database Management System
- ▶ Transaction Processing

Recommended Reading

1. Ahmed Q. and Vrbsky S. Maintaining security in firm real-time database systems. In Proc. 14th Annual Computer Security Applications Conference, 1998.
2. Ammann P., Jaeckle F., and Jajodia S. A two-snapshot algorithm for concurrency control in secure multi-level databases. In Proc. IEEE Symp. on Security and Privacy, 1992, pp. 204–215.
3. Ammann P. and Jajodia S. Distributed timestamp generation in planar lattice networks. ACM Trans. Comput. Syst., 11(3): 205–225, 1993.
4. Ammann P. and Jajodia S. An efficient multiversion algorithm for secure servicing of transaction reads. In Proc. First ACM Conf. on Computer and Communication Security, 1994, pp. 118–125.
5. Ammann P., Jajodia S., and Frankl P. Globally consistent event ordering in one-directional distributed environments. IEEE Trans. Parallel Distrib. Syst., 7(6):665–670, 1996.
6. Ammann P., Jajodia S., and Liu P. Recovery from malicious transactions. IEEE Trans. Knowl. Data Eng., 14:1167–1185, 2002.
7. Ammann P., Jajodia S., McCollum C., and Blaustein B. Surviving information warfare attacks on databases. In Proc. IEEE Symp. on Security and Privacy, 1997.
8. Atluri V., Bertino E., and Jajodia S. Degrees of isolation, concurrency control protocols, and commit protocols. In Proc. IFIP WG11.3 Working Conf. on Database Security, 1995, pp. 259–274.
9. Atluri V. and Huang W.K. Enforcing mandatory and discretionary security in workflow management systems. J. Comput. Secur., 5(4):303–340, 1997.
10. Atluri V., Huang W.K., and Bertino E. A semantic-based execution model for multilevel secure workflows. J. Comput. Secur., 8(1):3–41, 2000.
11. Atluri V., Jajodia S., Keefe T.F., McCollum C., and Mukkamala R. Mutilevel secure transaction processing: status and prospects. In Proc. 10th IFIP WG11.3 Working Conf. on Database Security, 1996.
12. Bell D.E. and LaPadula L.J. Secure computer system: unified exposition and multics interpretation. Tech. Rep. MTR-2997, MITRE Corporation, 1975.
13. Blaustein B.T., Jajodia S., McCollum C.D., and Notargiacomo L. A model of atomicity for multilevel transactions. In Proc. IEEE Symp. on Research in Security and Privacy, 1993, pp. 120–134.
14. Costich O. Transaction processing using an untrusted scheduler in a multilevel database with replicated architecture. In Proc. IFIP WG11.3 Working Conf. on Database Security, 1992, pp. 173–190.
15. George B. and Haritsa J. Secure concurrency control in firm realtime databases. Distrib. Parallel Databases, 5:275–320, 1997.
16. Jajodia S. and Atluri V. Alternative correctness criteria for concurrent execution of transactions in multilevel secure databases. In Proc. IEEE Symp. on Security and Privacy, 1992, pp. 216–224.
17. Jajodia S. and Kogan B. Integrating an object-oriented data model with multilevel security. In Proc. IEEE Symp. on Security and Privacy, 1990, pp. 76–85.
18. Kang I. and Keefe T. Transaction management for multilevel secure replicated databases. J. Comput. Secur., 3:115–145, 1995.
19. Kang K., Son S., and Stankovic J. STAR: secure real-time transaction processing with timeliness guarantees. In Proc. 23rd IEEE Real-Time Systems Symp., 2002.
20. Keefe T. and Tsai W. Multiversion concurrency control for multilevel secure databases. In Proc. IEEE Symp. on Security and Privacy, 1990, pp. 369–383.

21. Lala C. and Panda B. Evaluating damage from cyber attacks: a model and analysis. IEEE Trans. Syst. Man Cybern. A, 31(4): 300–310, 2001.
22. Lamport L. Concurrent reading and writing. Commun. ACM, 20(11):806–811, 1977.
23. Liu P. and Hao X. Efficient damage assessment and repair in resilient distributed database systems. In Proc. 15th IFIP WG11.3 Working Conf. on Data and Application Security, 2001, pp. 75–89.
24. Liu P. and Jajodia S. Multi-phase damage confinement in database systems for Intrusion Tolerance. In Proc. 14th IEEE Computer Security Foundations Workshop, 2001.
25. Maimone W. and Greenberg I. Single-level multiversion schedulers for multilevel secure database systems. In Proc. 6th Annual Computer Security Applications Conf., 1990, pp. 137–147.
26. McDermott J., Jajodia S., and Sandhu R. A single-level scheduler for replicated architecture for multilevel secure databases. In Proc. 7th Annual Computer Security Applications Conf., 1991, pp. 2–11.
27. Pal S. A locking protocol for multilevel secure databases providing support for long transactions. In Proc. 10th IFIP WG11.3 Working Conf. on Database Security, 1996, pp. 183–198.
28. Panda B. and Giordano J. Reconstructing the database after electronic attacks. In Proc. 12th IFIP WG11.3 Int. Working Conf. on Database Security, 1998.
29. Panda B. and Haque K.A. Extended data dependency approach: a robust way of rebuilding database. In Proc. 2002 ACM Symp. on Applied Computing, 2002.
30. Park C., Park S., and Son S. Multiversion locking protocol with freezing for secure real-time database systems. IEEE Trans. Knowl. Data Eng., 14(5):1141–1154, 2002.
31. Ray I., Ammann P., and Jajodia S. A semantic-based model for multi-level transactions. J. Comput. Secur., 6(3):181–217, 1998.
32. Ray I., Bertino E., Jajodia S., and Mancini L. An advanced commit protocol for MLS distributed database systems. In Proc. 3rd ACM Conf. on Computer and Communications Security, 1996, pp. 119–128.
33. Ray I., McConnell R., Lunacek M., and Kumar V. Reducing damage assessment latency in survivable databases. In Proc. 21st British National Conf. on Databases, 2004.
34. Reed D. and Kanodia R. Synchronizations with event counts and sequencers. Commun. ACM, 22(5):115–123, 1979.
35. Schaefer M. Quasi-synchronization of readers and writers in a multi-level environment. Tech. Rep. TM-5407/003, System Development Corporation, 1974.
36. Smith K.P., Blaustein B.T., Jajodia S., and Notargiacomo L. Correctness criteria for multilevel secure transactions. IEEE Trans. Knowl. Data Eng., 8(1):32–45, 1996.
37. Son S., Mukkamala R., and David R. Integrating security and real-time requirements using covert channel capacity. IEEE Trans. Knowl. Data Eng., 12(6):865–879, 2000.
38. Yu M., Liu P., and Zang W. Multi-version attack recovery for workflow systems. In Proc. 19th Annual Computer Security Applications Conf., 2003, pp. 142–151.
39. Zhu Y., Xin T., and Ray I. Recovering from malicious attacks in workflow systems. In Proc. 16th Int. Conf. Database and Expert Syst. Appl., 2005.
40. Zuo Y. and Panda B. Damage discovery in distributed database systems. In Proc. 18th IFIP WG11.3 Working Conf. on Data and Applications Security, 2004.

Security Services

Athena Vakali
Aristotle University, Thessaloniki, Greece

Synonyms

Authentication; Data confidentiality; Data integrity services

Definition

Given a set of local or distributed resources to be protected, a security service is a task (or set of tasks) that coherently performs processing or communication on behalf of the underlying system infrastructure, in order to support and employ several security requirements of both the system and the data sources. Such requirements involve authentication, PKI accessing etc. over the underlying resources. Security services typically implement portions of security policies and are implemented via particular processes which are called security mechanisms.

Key Points

Security services are well documented and described in X.800 documentation [1] for almost all the layers from the physical up to the application layer, whereas focus on security services applied on the Web is given in [2]. Physical and data layer security services focus on support confidentiality at various levels, namely at the connection, the traffic flow (both full and limited) and they are designed for peer to peer or multi-peer communications. At the network and transport layers security services involve authentication, access control, traffic flow, confidentiality and they are provided together or separately.

At the application layer, from X.800 the core security services are identified and they may be supported either singly or in combination, to employ access control for enforcing authentication, data confidentiality, data integrity and non-repudiation. As highlighted in [2], these involve:

1. Services for subjects/clients and resources identities: to verify the identity of the subject who requests a source access and prevent malicious client attempts. Therefore, such services may be either authentication services (to verify an identity claimed by/for an entity) or Nonrepudiation services (to prevent either sender or receiver from denying a transmitted message).
2. Services for subjects/clients authorizations over resources: to manage relationships among clients and protected resources. Therefore, such services involve either access control services (for protecting system resources against unauthorized access) or data privacy (for protecting data against unauthorized disclosure-data confidentiality and changes-data integrity.

Cross-references

▶ Authentication
▶ Database Security
▶ Secure Database Development

Recommended Reading

1. Recommendation X.800, Security architecture for open systems, Interconnection for CCITT applications. http://fag.grm.hia.no/IKT7000/litteratur/paper/x800.pdf
2. Stoupa K. and Vakali A. Policies for Web Security Services, Chapter III. In Web and Information Security, E. Ferrari, B. Thuraisingham (eds.). Idea Group, USA, 2006.

Segmentation

▶ Cluster and Distance Measure

Selection

Cristina Sirangelo
University of Edinburgh, Edinburgh, UK

Definition

Given a relation instance R over set of attributes U and a condition F, the selection $\sigma_F(R)$ returns a new relation over U consisting of the set of tuples of R which satisfy F. The condition F is an atom of the form $A = B$ or $A = c$, where A and B are attributes in U, and c is a constant value.

The generalized selection allows more complex conditions: F can be an arbitrary boolean combination of atoms of the form $A = B$ or $A \neq B$ or $A = c$ or $A \neq c$. Moreover, if a total order is defined on the domain of attributes, more general comparison atoms of the form $A \alpha B$ or $A \alpha c$ are allowed, where α ranges over $\{=, \neq, <, >, \leq, \geq\}$.

Key Points

The selection is one of the basic operators of the relational algebra. It operates by "selecting" rows of the input relation. A tuple t over U satisfies the condition $A = B$ if the values of attributes A and B in t are equal. Similarly t satisfies the condition $A = c$ if the value of attribute A in t is c. Satisfaction of generalized selection atoms is defined analogously.

As an example, consider a relation *Exams* over attributes (*course-number, student-number, grade*), containing tuples $\{(EH1, 1001, A), (EH1, 1002, A), (GH5, 1001, C)\}$. Then $\sigma_{grade=A \wedge course-number = EH1}(Exams)$ is a relation over attributes (*course-number, student-number, grade*) with tuples $\{(EH1, 1001, A), (EH1, 1002, A)\}$.

In the case that a relation schema is only specified by a relation name and arity, the result of the selection is a new relation having the same arity as the input one, containing the tuples which satisfy the selection condition. In this case the selection atoms are expressions of the form $j = k$ or $j = c$ (or $j \alpha k$ and $j \alpha c$ in the generalized selection). Here j and k are positive integers bounded by the arity of the input relation, identifying its j-th and k-th attribute, respectively.

Cross-references

▶ Relation
▶ Relational Algebra

Selective XML Dissemination

▶ XML Publish/Subscribe

Selectivity Estimation

Evaggelia Pitoura
University of Ioannina, Ioannina, Greece

Synonyms

Selectivity estimation; Cost estimation

Definition

For each query, there are many equivalent execution plans. To choose the most efficient among these different query plans, the optimizer has to estimate their cost. Computing the precise cost of each plan is usually not possible without actually evaluating the plan. Thus, instead, optimizers use statistical information stored in the DBMS, such as the size of relations and the depth of the indexes, to estimate the cost of each plan. For large databases, this cost is dominated by the number of disk accesses.

Since, in general, the cost of each operator depends on the size of its input relations, it is important to provide good estimations of their selectivity, that is, of their result size.

Key Points

During query optimization, the optimizer enumerates potential execution plans for each query and evaluates their cost in order to choose the less expensive among them. In general, computing the exact cost of each query plan is not possible without actually evaluating the plan. Instead, optimizers make use of statistical information stored in the DBMS catalog to estimate the cost of each plan. Such statistics include the number of tuples in each relation, the size of each tuple and the number of distinct values that appear in each relation. The cost of a plan can be measured in terms of different resources, such as CPU, I/O, buffer utilization, and, in the case of parallel and distributed databases, communication costs. However, in large databases, the cost is usually dominated by disk accesses.

The cost of each plan is computed by combining the estimations of the cost of each of the operators appearing in the plan. Since the cost of an operator depends mainly on the sizes of its input relations, it is central to have good estimates of the result size of each operator that is going to be used as input to the next operator in the plan.

Take, for example, a selection condition consisting of a number of predicates. Each predicate has a reduction factor, which is the relative reduction in the number of result tuples caused by this predicate. There are many heuristic formulas for the reduction factors of different predicates. In general, they depend on the assumption of uniform distribution of values and independence among the various relation fields. More accurate reduction factors can be achieved by maintaining more accurate statistics, for example in the form of histograms or multidimensional histograms. The accuracy of the estimates also depends on how frequently the available statistics are updated to reflect the current database state.

Cross-references

- ▶ Evaluation of Relational Operators
- ▶ Query Optimization
- ▶ Query Plan

Recommended Reading

1. Chaudhuri S. An overview of query optimization in relational systems. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 34–43.
2. Ramakrishnan R. and Gehrke J. Database Management Systems. McGraw-Hill, New York, 2003.
3. Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., and Price T.G. Access path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.

Selectivity for Predictive Spatio-Temporal Queries

▶ Spatio-Temporal Selectivity Estimation

Self-Maintenance of Views

Himanshu Gupta
Stony Brook University, Stony Brook, NY, USA

Definition

A data warehouse is a collection of materialized views derived from base relations that may not reside at the warehouse. It is important to keep the views up to date in response to changes to the base relations. *Self-maintenance* of views involves maintaining the views, using information that is strictly local to the warehouse: the view definitions and the view contents. Such self-maintenance of views (whenever possible) is more efficient than incremental maintenance or recomputation of views.

Key Points

In a data warehouse, views are computed and stored in the database to allow efficient querying and analysis of the data. These views stored at the data warehouse are known as *materialized views*, and are defined in terms of the base relations residing in data sources that may or may not be local to the warehouse. To keep the views consistent with the base data, any change reported by the data sources must be reflected in the views. In response to the changes at the base relations, the view can be either recomputed from scratch, or incrementally maintained, or self-maintained.

Incremental maintenance of a view involves propagating the changes at the source onto the view so that the view reflects the changes. While maintaining these views incrementally is often significantly more efficient than recomputing them from scratch (as done in most current data warehouses), it can still be expensive. For instance, in response to an update to a base relation, incremental maintenance of views defined as a join may involve looking up the non-updated base relations, which may reside in external sources. Some of these base relations may even be unavailable when the view needs to be maintained. Further, since base relations are independently updated, they may be read in an inconsistent state, often resulting in erroneous view updates. Thus, in data warehousing environments where maintenance is performed locally at the warehouse, an important incremental view-maintenance issue is how to minimize external base data access.

Motivated by the above, another approach of maintaining a view is called *self-maintenance.* In view self-maintenance, a view is maintained using information only local to the datawarehouse, viz., the view definition and the view contents. In general, a view may not be self-maintainable. However, in certain cases, a view can still be maintained using only a specified subset of the base relations. With the above approach of self-maintenance, it is possible to minimize the cost to maintain the data warehouse, shorten the time window during which the warehouse is inconsistent with the updated data sources, and avoid view update anomalies due to asynchronous base data updates.

There can be two notions [2] of self-maintainability (SM): the *compile-time SM* where a view is self-maintainable independently of the view's contents and the contents of the base relations, and under all updates of a certain type. The *runtime SM* is when a specific view is self-maintainable under a specific update and given the contents of a specified subset of base relations. Note that the runtime approach is more aggressive in the way that it may succeed in maintaining a view when the compile-time approach may fail. Below, works on run-time self-maintainability approach are described.

An important question is to determine whether a view is (run-time) self-maintainable. In other words, test for self-maintainability requires whether a unique new state is guaranteed, given an update to the base relations, an instance of the views, and an instance of a subset of the base relations. A second question is how to bring the view up to date using only the given information. Together, these two questions define the *view self-maintenance problem.*

The authors in [1,5] give self-maintainability conditions for views that are select-project-join queries with no self-joins and for single-relation insertions or deletions. Huyn in [3] solved the problem more efficiently than [1] for select-project-join views with no self-joins for single insertions. In particular, Huyn generate SQL queries that test whether a view is self-maintainable and update the view if it is. More specifically, he shows that for insertion updates and conjunctive queries: (i) the self-maintainability test are extremely simple queries that look for certain tuples in the view to be maintained, (ii) these tests can be generated from just the view definition using a simple algorithm based on the concept of "Minimum Z-Partition," and (iii) view self-maintenance can also be expressed as simple update query over the view.

In a follow-up work [4], Huyn considers the view self-maintenance problem in the presence of multiple views, under arbitrary mixes of insertions and deletions. In particular, for conjunctive queries, he gives an algorithm that generates, at view definition time, the query expressions required to maintain the view in response to base updates. He generalizes his techniques to the problem of generalized self-maintenance problem, where in addition to the warehouse views, one is also given access to some of the base relations. In general, he provides better insight into the problem by showing that view self-maintainability can be reduced to the problem of deciding query containment.

Cross-references

- ▶ Data Warehouse
- ▶ View Maintenance
- ▶ Views

Recommended Reading

1. Gupta A. and Blakeley J.A. Using partial information to update materialized views. Inf. Syst., 20(9):641–662, 1995.
2. Gupta A. and Mumick I.S. Maintenance of materialized views: problems, techniques, and applications. IEEE Data Eng. Bull., 18(2):3–18, 1995.
3. Huyn N. Efficient view self-maintenance. In Proc. Workshop on Materialized Views (VIEWS), 1996.
4. Huyn N. Multiple-view self-maintenance in data warehousing environments. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997.
5. Tompa F.W. and Blakeley J.A. Maintaining materialized views without accessing base data. Inf. Syst., 13(4):393–406, 1988.

Self-Management Technology in Databases

Surajit Chaudhuri[1], Gerhard Weikum[2]

[1]Microsoft Corporation, Redmond, WA, USA

[2]Max-Planck Institute for Informatics, Saarbrueken, Germany

Synonyms

Self-managing database systems; Autonomic database systems; Self-tuning database systems; Auto-administration and auto-tuning of database systems

Definition

The total cost of ownership (TCO) for a database-centric information system is dominated by the expenses for highly skilled human staff in order to deploy, configure, administer, monitor, and tune the database system. Self-management technology for databases aims to automate these tasks to the largest possible extent and throughout the entire life-cycle of the information system. This involves many dimensions that determine the system performance and availability such as: workload analysis, capacity planning, physical database design, database statistics management for query optimization, load control, memory management, system-health monitoring, failure diagnosis and root-cause identification, configuration of backup procedures and other self-healing capabilities. The self-managing capabilities can be incorporated in a system using a number of architectural options. For example, such capabilities can be either built into the system itself and integrated with its normal functionality, or provided through external tools that for the database engine. The latter approach is often referred to as DB advisors, DB assistants, or DB wizards.

The notion of self-management for database systems comprises a wide spectrum of issues, and it is tempting to consider the partial automation of all database-related human activity as facets of self-management, for example, information integration. However, this entry defines self-management technology to be confined to system issues that arise with the operation of a database engine, thus excluding the tasks that do not directly affect the engine's operation.

Historical Background

Needs for tuning tools and certain self-managing capabilities have been around for decades. For example, analytic models for capacity planning has a long tradition in the mainframe world [13], and methods for incremental online reorganization of storage and indexing systems can be seen as early forms of self-management. Selected issues of automatic tuning have been addressed already in the 1980's, most notably, on index selection [11].

In the late 1980's and throughout the 1990's, both database system functionality and workloads became much richer and increased the complexity of system management. Together with the proliferation of database systems across a wide spectrum of IT applications, this created a shortage of sufficiently skilled system administrators and tuning experts. At the same time hardware and software-licensing costs were rapidly decreasing, so that human staff for system management became the key factor in total cost of ownership (TCO). These trends alerted the database-system industry in the mid 1990's and led to intensive research and development initiatives at all of the major system vendors.

Early work on more comprehensive strategies and principles of automatic tuning included the Comfort project at ETH Zurich [18,19] and the "DBMS Autopilot" work at the University of Wisconsin [2]. These projects were in turn inspired by prior work towards adaptive and self-tuning operating systems [16], and particularly focused on resource control for dynamically evolving, mixed workloads. Starting in the mid 1990's interest in self-management was revived by the AutoAdmin project, which then became the leading initiative in this area [5,8,9]. AutoAdmin initially focused on physical database design, but subsequently also considered many other issues such as adaptive statistics management, system monitoring, and online tuning techniques. Good overviews of the progress achieved in the past ten years, in terms of both research contributions and product impact, are given by [1,8,9].

Foundations

General Framework

The overriding goal of self-management is to operate the database system at a satisfactory level of performance and service quality – at every point in time regardless of load peaks or shifts in workload characteristics. Specifically, the system should automatically adjust its configuration to evolving workloads. To this end, the key issue is to understand for each component of the system the dependencies that relate the system configuration and workload properties to the resulting performance measures:

$$\textit{configuration} \times \textit{workload} \rightarrow \textit{performance}$$

All three parameters of this relation need to be interpreted broadly:

- *System configuration* includes the *hardware setup* (number and speed of processor cores, memory size, number and characteristics of disks and access channels, etc.), the *software setup* at system-boot time (e.g., thresholds for lock escalation or memory-pressure handling), the *physical database design* (indexes, materialized views, etc.), the *backup or replication procedures* and their parameters (e.g., frequency and granularity of backups, consistency protocols for replication), and also the system's *run-time adaptation policies* to handle newly arising conditions (e.g., the scheduling policy and workload-class priorities, the memory allocation policy, etc.) as well as the system's *exception-handling policies* (e.g., load shedding under memory pressure).
- *Workload characteristics* include the types of queries, update operations, transactions, and workflows that access the database (i.e., the *workload classes*), their frequencies, their arrival rates during specific periods (e.g., main business hours vs. weekend vs. end-of-fiscal-year processing), their arrival patterns (e.g., typical sequences of different queries), their co-occurrence patterns (e.g., online transactions concurrently with certain batch jobs), the distribution of query parameter values, and so on.
- *Performance measures* include the *throughput* and *response times* (or properties of response-time distributions like quantiles) of different workload classes, but also metrics that capture the system's *dependability*, namely, *reliability* (e.g., probability of losing data by a permanent failure such as double or triple disk failures), *availability* (i.e., probability of being able to service requests at any timepoint in the presence of transient outages), *performability* (i.e., the performance level that can be sustained in degraded configurations, e.g., when servers or data replicas are temporarily unavailable), and ultimately, even capabilities to react to the system environment such as *resilience to security attacks*, graceful handling of denial-of-service attacks, countering attempts to breach data privacy, and so on.

If one had a reasonably accurate and complete model of the configuration-workload-performance relation, a self-managing system could, in principle, solve the following "inverse problem":

Given specified goals for performance measures and the workload properties, find the lowest-cost configuration that satisfies the performance goals.

This would have to be solved dynamically whenever the workload exhibits major changes and necessitates adaptive reconfiguration. The goals in this setting should cover throughput, response time (e.g., a 95th quantile of at most 1 second), availability (e.g., expected downtime per year of at most 10 minutes), and other measures, leading to a more general notion of *service level agreements (SLA)* or *quality-of-service (QoS) guarantees*. The specified SLA/QoS requirements would drive the self-management procedures.

All three elements of the configuration-workload-performance function can refer to an entire system or to individual *components*. The latter is highly preferable: the configuration parameters are then restricted to the ones that are relevant for the specific component at hand, and the same modularization holds for workload properties and performance measures. At the system level, the configuration is the union of the component-specific parameters, the workload at the system level is decomposed into workload properties for each component, and the performance observed at the component level is aggregated into system-level measures. As a consequence, predicting the system-wide performance requires *composability* of workload-characterization models and, most crucially, of performance-prediction models for the underlying components.

An inherent difficulty in achieving self-manageability is that many of the input parameters are not known a priori, may change rapidly thus posing difficulties to parameter estimation techniques, and are inherently difficult to model and thus bound to be incompletely captured. In particular, workload modeling is an extremely difficult task: queries, for example, can be modeled at different resolution levels from SQL statements down to internal operator trees (or actually, pipelined DAGs) or storage-level access operations; arrival rates can refer to steady state (long-term averages) or a specific look-ahead time horizon like the next hour or next minute; the dependencies in the arrival patterns of different query or transaction types can be determined by statistical correlations or by causal flow models via static program analysis of the application code.

Although the above discussion presents only a conceptual model, this framing does provide useful guidelines for approaching specific self-management issues. The salient aspects of the configuration-workload-performance framework and the general aim of solving for a suitable configuration suggest a methodology that is best characterized as an *observe-predict-react* cycle:

- *Observe:* Workload characteristics need to be observed as the system is running, so as to estimate parameters of the workload model or models. Observations may need to be collected at different resolution levels and time scales, and they must track drifts and anomalies in the workload evolution. This calls for a form of *introspection* or *self-monitoring*.
- *Predict:* As the current configuration is known and given the outcome of the observation step, one can aim to assess the performance in the near-term future, either with the current configuration as well as with various alternative configurations that could potentially improve performance. This calls for a notion of *what-if analysis*.
- *React:* Occasionally the system configuration needs to be reconsidered, in view of the observed workload changes and projected near-term performance. In the case that explicit performance goals are given, the search for and assessment of new configurations could be triggered whenever one of the goals is violated (or about to be violated) with the current configuration. In the case without explicit goals, one or more objective functions need to be specified, so that whenever there is a significant loss in the objective function(s), the reaction step is invoked. The react step needs a smart search strategy to identify promising new configurations.

The observe-predict-react approach can be applied to *different time scales* based on the specific self-management task at hand from long-term capacity planning, which may be done relatively infrequently, down to real-time decisions, for example, about memory management, which requires all three steps at the resolution of minutes or even seconds to handle sudden memory-pressure situations. The three steps may be implemented as tools outside the database engine, or could be incorporated into the engine at *different integration levels*. Typically, short time scales mandate deeper integration, whereas long-term decisions could be made by external advisor tools.

Self-Management Paradigms

Ideally, the models and strategies for self-managing systems would follow a unified principle such as mathematical optimization theory, but such a "grand solution" is not in reach today. Instead, there are many diverse results on a variety of specific self-management issues; the state of the art is best characterized by a number of self-management paradigms: approaches that work well on paradigmatic example problems but bear the potential for being generalized into more broadly applicable principles. In the following, several such paradigms will be introduced, each with a general characterization and one or more exemplary use cases.

Trade-off Elimination: Tuning knobs exist because of trade-offs: there is no algorithm (or data structure) that performs near-optimal under all possible workloads, and therefore, systems are equipped with different options. However, sometimes it is possible to design an algorithm or a strategy that performs very well across the full spectrum of workloads, and this algorithm should be knob-free or only have second-order parameters which are uncritical or easy to set once across (almost) all workloads [19]. Such an algorithm effectively eliminates the trade-off. Examples are modern cache-replacement algorithms (LRU-k, ARC, etc.) or B + -tree indexes for both exact-match lookups and range scans. In the case of quantitative tuning parameters such as disk block sizes or striping units, similar considerations may lead to robust techniques that can effectively eliminate knobs.

Static Optimization: Some self-tuning problems can be cast into mathematical optimizations with statically given input parameters. In principle, this opens up solutions based on combinatorial optimization methods such as branch-and-bound. Physical database design falls into this category; planning backup or replication procedures is another example. However, it is crucial to obtain input parameters and evaluate the objective function in a way that preserves the actual behavior of the database system. A major lesson from the research on physical design automation [8,9] is that a hand-crafted cost model, for assessing the quality of a particular design configuration, is bound to be inappropriate no matter how detailed and seemingly accurate it may be. The crux is that a separate cost model does not consider the actual behavior of the engine's query optimizer in selecting indexes for particular queries. Instead, the practically viable solutions references the query optimizer's cost model each time they need to assess the benefit of a design configuration for the given workload. To limit the computational overhead of these calls, thoroughly designed techniques for what-if assessment are needed (so as to estimate the benefit of a design configuration without actually building indexes). In addition, great care must be taken in the enumeration of candidate configurations. Although this is an offline optimization, the combinatorial explosion of the search space could easily lead to unacceptable run-time.

Stochastic Optimization: In some situations, albeit dealing with an offline optimization problem, the input parameters can only be characterized as random variables or by means of stochastic processes. System capacity planning falls into this category, for example, deciding how many disks are needed or how big a shared database cache should be in order to satisfy throughput and response time goals. For such problems, the established methodology is stochastic modeling, most importantly, queuing theory as the non-linear effects of resource contention under multi-user load are most critical [13,15]. For tractability, stochastic models often need to make simplifying assumptions, and this entails the crucial issue of how accurate the model's predictions can still be. This concern can be addressed in two ways: (i) using more advanced mathematics to capture also non-standard situations or combining analytic models with simulations (e.g., to capture realistic inter-arrival time distributions rather than simply postulating an exponential distribution), and (ii) using stochastic models as a building block in a more comprehensive approach, where even somewhat inaccurate relative predictions are beneficial towards configuration decisions.

Online Optimization: Many self-managing problems obtain their inputs – workload parameters – only dynamically (but then accurately, not stochastically) and need to optimize configuration parameters as the workload evolves. These situations typically entail periodic or even continuous re-optimization, possibly in an incremental manner, and face tight timing constraints for finding the solution [2,3]. Memory governing for workspaces (e.g., for hash-joins or sorting) and database statistics management (e.g., for multidimensional histograms) fall into this category, the former having a time horizon of minutes or seconds, the latter with a typical reconsideration cycle of days but then requiring expensive database accesses. Inspired by online optimization theory, a possible approach to tackle these problems is to identify fast approximation techniques to obtain a viable solution. However, the details of the specific approach heavily depend on the problem at hand.

Feedback Control: When it is very difficult, if not impossible, to capture some aspect of the configuration-workload-performance relation in a causal model, the paradigm of feedback control loops offers a principled alternative even in the absence of causal understanding. Dynamic load control that adjusts the multiprogramming level (MPL, i.e., the maximum number of concurrent threads) of a server falls into this category. These methods consist of an admission control, to avoid overload effects that may result in performance thrashing, and a cancellation control to shed load when performance goals can no longer be met. This is crucially important especially for memory management, but potentially also for lock management and other resources. The main challenge that must be addressed here is that of sudden load surges: bursty arrivals of requests that require adjustment of MPL settings for smoother load (possibly even with workload-class-specific MPL limits). These transient effects are inherently difficult to model analytically, even with stochastic models. Feedback loops treat the MPL settings as control variables that are adjusted based on simple differential equations over the measured performance values, the desired performance goals, and the control values. Control theory may be harnessed to ensure stability properties. Feedback control has been shown to work well for sufficiently simple functionality such as Web application servers [10].

Statistical Learning: Machine-learning models that use statistics for regression of continuous functions or classification with discrete labels are becoming increasingly attractive for self-tuning tasks [12,17]. Modern statistical learning methods can handle large numbers of input parameters (high-dimensional multivariate models) and can determine the most influential factors or can learn a predictor for an output variable. The only input is prior observations, sometimes along with manually assigned labels for training (if needed). In the context of self-managing database systems, the data are the system's event log: fine-grained information about configuration values (whenever they change), performance measurements, and also exceptions and potential problem situations. Statistical learning on this data can be used for diagnosis and root-cause analysis. A word of caution is in order, though: even if, in principle, all kinds of functions, labelings, and rankings can be learned, it is still an art to design the learning models and their feature spaces in the right way. So just like all the other self-management paradigms, statistical learning is not a panacea in itself.

Infrastructure

Self-management technology requires capabilities for adaptation, introspection, and self-healing. This in turn calls for a rich infrastructure in or around the database engine, so as to gather the necessary data and provide the mechanisms that will be enacted by the self-managing strategies. In the last few years, all modern database systems have addressed these requirements and provide rich facilities in this regard:

- Many internal algorithms (e.g., hash-joins or sorting) are *resource-adaptive*: they can continue running even if their resources, like workspace memory, are reduced at run-time. (But this alone does not provide self-management; in addition, strategies are needed for deciding when and how to reduce or increase resource assignments.)
- For *continuous self-monitoring*, light-weight techniques have been developed to identify and trace relevant events (e.g., exceptions raised because of memory shortage) and aggregate large amounts of monitoring data. The difficulty that these techniques have successfully overcome is to do all this with very low run-time overhead so as not to adversely affect normal system operation [7].
- For self-healing, techniques have been added to isolate abnormal behavior and re-initialize potentially affected system components [4]. These approaches are eased by the fact that database systems generally follow the transactional paradigm for data accesses. By aborting ongoing transactions and sessions, resetting software components is greatly simplified. A similar argument holds for fail-over techniques among redundant servers (with data replication or shared disk storage), and this even works over large geographic distances to provide disaster recovery.

Future Directions

Self-management is an important topic across all kinds of computer systems. Ultimately, all IT systems should strive to become as easily usable as household appliances such as washing machines or TV sets. In some areas, the vision of administration-less and trouble-free solutions has almost been achieved, most notably, in storage systems [20]. However, database systems, with their very powerful languages like SQL and XQuery and their application-specific extensibility, have a much richer functionality than storage and thus face a much harder challenge.

In fact, database systems and their application workloads have become so complex that self-management is no longer just a desirable capability but will be a vital necessity in the long run. But despite the good progress on many issues of this theme, the quest for overriding principles has not yet achieved any breakthrough. This is partly due to the difficulty of the problems (and the quest is certainly ongoing); on the other hand, one could conjecture that the problem may, to some extent, be ill-defined given today's very sophisticated system architecture with their overwhelming richness of features and the limited set of abstractions.

This concern is reflected in considerations on componentizing database systems into building blocks with narrow interfaces and much fewer tuning choices [6,14], in building specialized data-management engines for particular application domains, or drastically limiting the engine's options and functionality. Virtualization of resources is another trend towards simplifying system management, but its impact on database engine tuning is still unclear. A breakthrough may require radical departures from today's architectures as well as rethinking the functionality that is offered by the database system. Similar to the "design-for-recovery" position of [4], a completely new *design-for-manageability* approach may be needed.

Cross-references

▶ Database Tuning using Combinatorial Search
▶ Database Tuning using Online Algorithms
▶ Database Tuning using Trade-off Elimination

Recommended Reading

1. Ailamaki A (ed.) Special issue on self-managing database systems. IEEE Data Eng. Bull., 29(3):2006.
2. Brown K.P., Mehta M., Carey M.J., and Livny M. Towards automated performance tuning for complex workloads. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994.
3. Bruno N. and Chaudhuri S. To tune or not to tune? A lightweight physical design alerter. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 499–510
4. Candea G., Brown A.B., Fox A., and Patterson D.A. Recovery-oriented computing: building multitier dependability. IEEE Comput., 37(11):60–67, 2004.
5. Chaudhuri S., König A.C., and Narasayya V.R. SQLCM: a continuous monitoring framework for relational database engines. In Proc. 20th Int. Conf. on Data Engineering, 2004.
6. Chaudhuri S. and Narasayya V.R. An efficient cost-driven index selection tool for microsoft SQL server. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997.
7. Chaudhuri S. and Narasayya V. Self-tuning database systems: a decade of progress. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
8. Chaudhuri S., Narasayya V., and Syamala M. Bridging the application and DBMS profiling divide for database application developers. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
9. Chaudhuri S., and Weikum G. Rethinking database system architecture: towards a self-tuning RISC-style database system. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000.
10. Diao Y., Hellerstein J.L., Parekh S.S., Griffith R., Kaiser G.E., and Phung D.B. A control theory foundation for self-managing computing systems. IEEE J. Select. Areas Commun., 23(12): 2213–2222, 2005.
11. Finkelstein S.J., Scholnick M., and Tiberio P. Physical database design for relational databases. ACM Trans Database Syst., 13(1):91–128, 1988.
12. Jiang N., Villafane R., Hua K.A., Sawant A., and Prabhakara K. ADMiRe: an algebraic data mining approach to system performance analysis. IEEE Trans. Knowl. Data Eng., 17(7):888–901, 2005.
13. Lazowska E.D., Zahorjan J., Scott Graham G., and Sevcik K.C. Quantitative system performance: computer analysis using queuing network models, Prentice-Hall, Englewood, Cliffs, NJ, 1984.
14. Lightstone S. Seven software engineering principles for autonomic computing development. Innovations in Syst. and Softw. Eng., 3(1):71–74, 2007.
15. Menasce D.A. and Almeida V.A.F. Capacity Planning for Web Performance. Metrics, Models and Methods: Metrics, Models and Methods, Prentice-Hall, 2001.
16. Reiner D.S. and Pinkerton T.B. A method for adaptive performance improvement of operating systems. In Proc. 18th ACM Symp. on Operating System Principles, 1981.
17. Stillger M., Lohman G.M., Markl V., and Kandil M. LEO – DB2's LEarning Optimizer. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
18. Weikum G., Hasse C., Moenkeberg A., and Zabback P. The COMFORT automatic tuning project. Inform. Syst., 19(5): 381–432, 1994.
19. Weikum G., Moenkeberg A., Hasse C., and Zabback P. Self-tuning database technology and information services: from Wishful thinking to viable engineering. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
20. Wilkes J., Golding R.A., Staelin C., and Sullivan T. The HP AutoRAID hierarchical storage system. ACM Trans. Comput. Syst., 14(1):108–136, 1996.

Self-Managing Database Systems

▶ Self-Management Technology in Databases

Self-Tuning Database Systems

▶ Self-Management Technology in Databases

Semantic Analysis of Video

▶ Video Content Analysis

Semantic Data Integration for Life Science Entities

Ulf Leser
Humboldt University of Berlin, Berlin, Germany

Synonyms

Object identification; Data fusion; Duplicate detection; LSID

Definition

An entity is the representation of a (not necessarily physical) real-world object, such as a gene, a protein, or a disease, within a database. To integrate information about the same entities from different databases, these representations must be analyzed to uncover the corresponding underlying objects. This process is called entity identification. A variation of entity identification is *duplicate detection*, which analyses two or more entities to determine whether they represent the same real-world object or not. Finally, *data fusion* is the process of generating a single, homogeneous representation from multiple, possibly inconsistent entities that represent the same real-world object.

When entities have globally unique *keys*, such as ISBN numbers in the case of books, entity identification and duplicate detection are simple. However, in life science databases, one usually has only descriptive information, such as the name or the sequence of a gene, which does not suffice to uniquely identify real-world objects. A *homonym* is a single name (or an ID) that is identifies multiple, different objects. For instance, the term "ACE" may reference many different proteins, such as "angiotensin converting enzyme" or "acetylcholinesterase." *Synonyms* are multiple names (or IDs) given to the same object.

Entity identification is particularly important in data curation, which is the (often manual) process of distilling a comprehensive description of a complex object from multiple data sources.

Historical Background

Duplicate detection has a long tradition in *census databases* where multiple representations of a person have to be identified to ensure reliable statistics. In the life sciences, the problem is particularly difficult, because many biological objects are much less stringently defined than, say, a human being. For instance, it is not clear whether two copies of a gene sequence on different sections of a chromosome should be considered the same gene or not, or if two highly similar and functionally identical genes in different species should be considered as different genes or not. At the other extreme, one often treats a gene and the protein it encodes as one object, especially in prokaryotes where differential splicing is almost non-existent, leading to a strict 1:1 relationship between genes and proteins. Thus, although a (DNA or protein) sequence is the fundamental identifying property of many biological objects, it does not always lead to unique entity identification.

First calls for establishing world-wide standards to identify biological objects appeared in the early 1990s, when the Human Genome Project (HGP) quickly increased the amount of available information on genes and other molecular objects [4]. The HGP from the very start was an internationally distributed effort with no central organization that could have enforced consistent naming of objects or assignment of IDs to objects.

Standardization efforts were initiated in the domains of genes, proteins, and clones. Other areas where identification of objects is an important issue are small molecules, diseases, and species. One way to achieve standardized names was the installation of committees to define naming conventions for biological objects, such as the HUGO Gene Nomenclature Committee. On the other hand, some databases have become the de-facto standards for some types of biological objects, and their IDs are now commonly used as identifiers. An example is the use of UniProt-IDs as identifiers for proteins.

However, in many areas no standards exist or standards are not commonly used. Thus, many areas are facing the problem of identifying duplicates. The predominant approach is the comparison of biological sequences, i.e., DNA or protein sequences, using various variations of *edit distance* calculations. However, recent scientific discoveries, such as the importance of differential splicing (one gene forming several proteins) or the existence of paralogs (highly similar genes in the same genome with different function) render the pure usage of sequence similarity insufficient. Today, the method of choice for performing duplicate detection in a life sciences database therefore depends on the specific domain and the scientists' and databases' particular understanding of the biological object.

Foundations

Identity Versus Similarity

The entity identification problem exists for all types of objects in the life sciences. It is particularly pressing for genes and proteins. For most purposes one considers a gene to be a (not necessarily continuous) stretch of DNA, i.e., a sequence of the four nucleic acids, on a chromosome, which is – by some complex regulation mechanism – at times first transcribed into RNA and then translated into a protein. A protein is a molecule consisting of a linear chain of amino acids forming a complex 3D structure. The translation procedure moves through the RNA and appends for each triple of nucleic acids one amino acid to the growing protein chain; thus, the translation of a given gene sequence into a protein is unique. The function of a protein mostly depends on the topology of its 3D structure and the properties of atoms exhibited at the surface of the structure [3].

The revelation of the function(s) of a protein is a crucial task in Bioinformatics and especially important for drug development. Because similar structure may hint toward similar function (independent of the species), a standard procedure for revealing the function of a human protein is to experimentally analyze the structure of a similar protein in another well-studied species, such as mouse or fruit-fly. Because the sequence of a gene determines the sequence of its protein(s), which in turn determines the structure of the protein, the same principle holds for genes: Two genes with similar sequence quite likely code for proteins with similar function.

With the function being the most important attribute of a gene/protein, the predominant question for semantic integration of genes and proteins is one of *similarity of sequences* rather than identity. The similarity of two biological sequences usually is measured by their (weighted) *edit distance* [9]. Two genes are considered as identical if their sequences are very similar, with the exact definition of "very similar" being a subject of much debate. The fundamental tool for entity identification and duplicate detection of genes and sequences therefore still is sequence comparison, using tools such as BLAST.

Naming Standards

Once a group of genes or proteins have been identified as "one real-world" object, one has to select a common name or ID for this group. For human genes, the Human Genome Organization (HUGO) installed the HUGO Gene Nomenclature Committee (HGNC), which is responsible for assigning names to human genes. Gene names are often complex, multi-term phrases that might include parts of the name of the person who discovered the gene, the phenotype it produces when a defect occurs, an inherited disease the gene is associated to, the chromosome and species where it first was detected, etc. However, the work of the HGNC is still widely ignored (especially in scientific publications). For other species, similar bodies often were more successful in terms of standardization. The reason for their success is (i) the number of researchers working on a particular other species is usually much smaller than for humans, (ii) the number of genes in other species is much smaller (e.g., Yeast ~4,000 genes, human ~22,000 genes), and (iii) data collection was centralized early on; (see the Mouse Genome Database MGD for an example). Furthermore, certain database identifiers are commonly used to denote genes, especially those from the EntrezGene database.

No established naming convention exists for human proteins. The problem is worse than for genes, because on average every human gene codes for ~8 proteins by means of differential splicing. Very often database identifiers are used instead of spoken names. De-facto standards are IDs from the Protein Data Bank (PDB) and the Uniprot Knowledge Base (formerly known as SwissProt).

For DNA sequencing, chromosomes are broken into pieces, which are cultivated, copied, and distributed inside living bacteria, called clones. Sets of clones are gathered in libraries that often contain hundreds of thousands of clones. Physically, a library is a set of so-called plates with 10–100 dwells, each containing a particular clone. To avoid duplication of work, clones must be identified uniquely. However, after generation nothing is known about a particular clone except the dwell where it is hosted. Researchers therefore identify clones by the plate number and column/row number inside the library. Since re-distribution of clones into different, specialized libraries is commonplace, there are clones with more than 20 different IDs of this form (see the Genome Database (GDB) for examples).

To overcome the problem of object names, the *Object Management Group* (OMG) recently has defined the LSID – Life Science Identifiers – standard for "*persistent, location-independent, resource identifiers for uniquely naming biologically significant resources including species names, concepts, occurrences, genes or proteins, or data objects that encode information about them.*" An LSID identifier consists of a network identifier (usually the fixed term *urn:lsid*), an authority identifier (who defines this name), a namespace identifier, an object identifier, and a revision identifier (see discussion below on object versions). Whether or not this standard will be accepted by the community remains to be seen.

Evolution of Names

A particular problem with names and IDs is how to keep them stable and consistent. For instance, the question of which (of multiple) names of a particular gene is used in the literature is highly influenced by a kind of social-scientific "fashion" [10]. In biological databases, objects need to be merged and deleted, rendering existing IDs inconsistent. Furthermore, new findings may completely change what is known about a gene while keeping the ID unchanged, which makes previous studies based on the now outdated knowledge obsolete. This problem is usually solved by implementing a particular versioning model, which distinguishes major and versioned IDs; major ID always point to the most current version and a new version of the object is created with every update.

Key Applications

Semantic Integration of Entities

Duplicate detection is vital to ensure high data quality in integrated databases. It consists of two steps. First, multiple representations of same real-world entities have to be discovered. Second, for each group of duplicates, a uniform representation must be found [8]. Both steps are particularly difficult in the Life Sciences, because object definitions are vague (see above) and most biological data are obtained by complex experiments and are notoriously noisy [2].

Therefore, both tasks most often are performed manually, a process called *data curation.* It is common that large biological databases employ professional curators whose task it is to read new publications and to convert the most important information into some semi-structured database entry. There are also forms of community curation, where the correction of errors in databases through a web interface is possible for registered users [5].

The general problem with curation is that it is very costly and highly subjective. At the same time, deciding on the correct value given two diverging experimental results is usually impossible without further experiments. Consequently, a typical approach to data integration in this field is to omit the second step. The resulting architecture has been called "entity-based" or "multidimensional." Different sources containing information about the same set of entities are considered as dimensions of the entities and are integrated into a schema similar to a *star schema* in *Data Warehouses* [11]. The advantage and disadvantage is that the different views on entities are reported to the user in a logically separated manner; thus, she has the ability but also the obligation to select the appropriate values herself. Alternatively, mixtures of manual and automatic data fusion are used (see [1] for an example).

Entity Identification in Text

A related problem is the identification of object names in scientific publications, i.e., in English sentences. Named Entity Recognition (NER) is the problem of judging for a given set of terms within a document whether they form a gene name. A standard technique for solving NER is the usage of *classification* based on machine learning algorithms. Named Entity Normalization (NEN) is the problem of assigning a unique name to an entity in text once it has been identified as such. NEN typically is tackled using large *dictionaries* of names and some kind of fuzzy string matching method. See [6,7] for recent surveys on both tasks.

URL to Code

BLAST tool for search similar sequences in database: http://www.ncbi.nlm.nih.gov/BLAST/

EntrezGene database: http://www.ncbi.nlm.nih.gov/sites/entrez

HUGO Gene Nomenclature Committee: http://www.genenames.org/

Mouse Genome Database: http://www.informatics.jax.org/

OMG Life Science Research Task Force: http://www.omg.org/lsr/

Protein Data Bank (PDB): http://www.pdb.org/

Uniprot knowledge base: http://www.uniprot.org/

Cross-references

- ► Data Deduplication
- ► Information Integration
- ► Information Integration Techniques for Scientific Data
- ► Metadata Management and Resource Discovery
- ► Object Identity
- ► Scientific Databases

Recommended Reading

1. Bhat T.N., Bourne P., Feng Z., Gilliland G., Jain S., Ravichandran V., Schneider B., Schneider K., Thanki N., and Weissig H, et al.

The PDB data uniformity project. Nucleic Acids Res., 29(1):214–218, 2001.

2. Brenner S.E. Errors in Genome Annotation. Trends Genet., 15(4):132–133, 1999.
3. Gibson G. and Muse S.V. A Primer of Genome Science. Sinauer Associates, Sunderland, MA, 2001.
4. Karp P.D. Models of identifiers. In Proc. Second Meeting on Interconnection of Molecular Biology Databases. Cambridge, UK, 1995.
5. Kingsbury D. Consensus, common entry, and community curation. Nat. Biotechnol., 14:679, 1996.
6. Krauthammer M. and Nenadic G. Term identification in the biomedical literature. J. Biomed. Inform., 37(6):512–526, 2004.
7. Leser U. and Hakenberg J. What Makes a Gene Name? Named Entity Recognition in the Biomedical Literature. Briefings in Bioinformatics, 6(4):357–369, 2005.
8. Müller H., Naumann F., and Freytag J.-C. Data quality in genome databases. In Proc. Conf. on Information Quality, 2003.
9. Smith T.F. and Waterman M.S. Identification of common molecular subsequences. J. Mol. Biol., 147:195–197, 1981.
10. Tamames J. and Valencia A. The success (or not) of HUGO nomenclature. Genome Biol., 7(5):402, 2006.
11. Trissl S., Rother K., Müller H., Koch I., Steinke T., Preissner R., Frömmel C., and Leser U. Columba: an integrated database of proteins, structures, and annotations. BMC Bioinformatics, 6:81, 2005.

Semantic Data Model

David W. Embley
Brigham Young University, Provo, UT, USA

Synonyms

Conceptual model; Conceptual data model

Definition

A *semantic data model* represents data in terms of named sets of objects, named sets of values, named sets of relationships, and constraints over these object, value, and relationship sets. The *semantics* of a semantic data model are the intensional declarations: the names for object, value, and relationship sets that indicate intended membership in the various sets and the declared constraints that the data should satisfy. The *data* of a semantic data model is extensional and consists of instances of object identifiers and values for object and value sets and of *m*-tuples of instances for *m*-ary relationship sets. The *model* of a semantic-data-model instance describes intensionally a real-world domain of interest. The modeling components of the semantic data model specify the modeling elements from which a real-world model instances can be built.

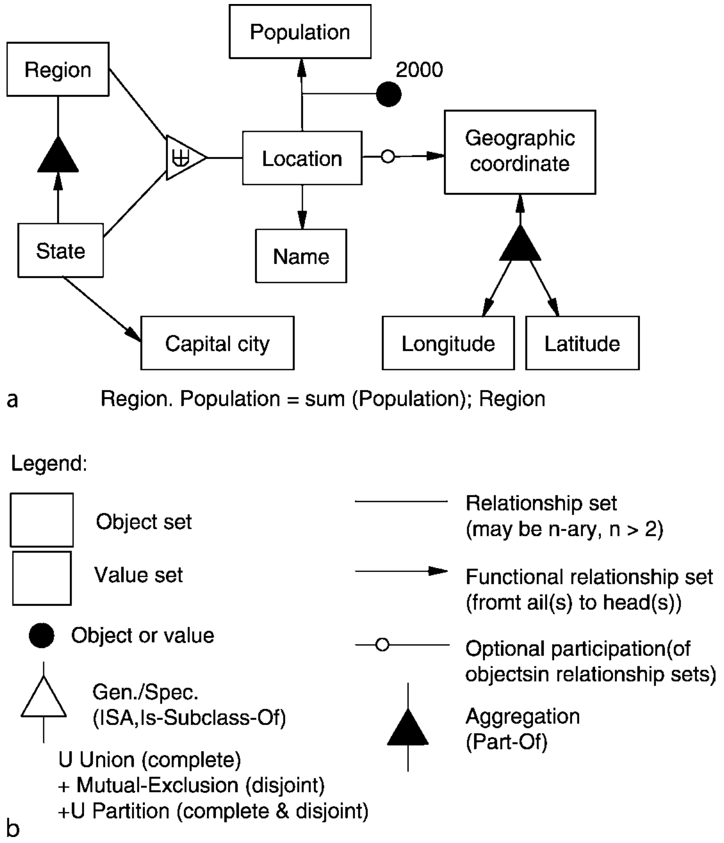
For a general description of semantic data models, see [1]. This article describes the generic properties of semantic data models and presents a representative collection of early semantic data models.

Key Points

Figure 1a gives a sample semantic data model. Semantic data models use graphical symbols to represent data semantics. Each semantic data model, however, has its own set of graphical symbols. The graphical symbols in Fig. 1b are meant to be generic-representative of the symbols used and illustrative of the kinds of data sets and constraints included in typical semantic data models.

The legend in Fig. 1b tells what each symbol means.

- A box with a solid border designates a set of objects. In Fig. 1a *State* designates the set of U.S. states (e.g., the state of California) and *Region* designates areas within the U.S. (e.g., the region of states in the Northeastern part of the U.S.).
- A box with a dashed boarder represents a set of values. In Fig. 1a *Capital City* designates the names of the capital cities of the U.S. states (e.g., "Sacramento" for California) and *Longitude* designates the set of longitude values for the geographic coordinates of locations (e.g., 120°4.9′W for the longitudinal part of the centerpoint of California).
- A large filled-in dot represents a single object or value. In Fig. 1a the object stands for the year 2000. Single objects or values can be thought of as singleton object or value sets – an object set or value set with one object.
- Relationship-set names can appear explicitly or can be a composition of the names of connected object and value sets. In Fig. 1a the names are all compositions (e.g., *State-CapitalCity* names the binary relationship set between *State* and *Capital City*). In general, relationship sets are *m*-ary ($n \geq 2$) (e.g., the *Location-2000-Population* relationship set is ternary).
- Constraints on relationship sets include functional/non-functional and mandatory/optional constraints. An arrowhead designates a functional relationship set from its tail(s) as domain space (s) to its head(s) as range space(s) (e.g., *Location-GeographicCoordinate* is functional from *Location*


Population
Region
2000
Location
Geographic coordinate
State
Name
Capital city
Longitude
Latitude
a
Region. Population = sum (Population); Region
Legend:
Object set
Value set
Object or value
Gen./Spec. (ISA,Is-Subclass-Of)
U Union (complete)
+ Mutual-Exclusion (disjoint)
+U Partition (complete & disjoint)
Relationship set (may be n-ary, n > 2)
Functional relationship set (fromt ail(s) to head(s))
Optional participation(of objectsin relationship sets)
Aggregation (Part-Of)
b

Semantic Data Model. Figure 1. Sample semantic data model instance.

to *Geographic Coordinate*). A small "o" near a connection between an object (value) set and a relationship set allows for optional participation of the objects in the object set (values in the value set) in relationships in the relationship set (e.g., the "o" near *Location* in the *Location-GeographicCoordinate* relationship set makes the participation of location objects in the relationship set optional – a geographic coordinate for a location such as Northeast need not have a geographic coordinate).

- Generalization/specialization constraints, represented by a triangle, designate the specialization sets attached to the base of the triangle as subsets of the generalization set attached to the apex of the triangle (e.g., both the set of states and the set of regions are subset of the set of locations). If a union symbol (∪) appears in the triangle, the generalization is a union of the specializations. If a mutual-exclusion symbol (+) appears, the specializations are pairwise non-intersecting. And if a partition symbol (⊎) appears, both a union and mutual-exclusion constraint hold, making the constraint be a partition (e.g., in the semantic-data-model instance in Fig. 1, *Region* and *State* are mutually exclusive and the union constitutes *Location* – all the locations of interest for the semantic-data-model instance).
- Aggregation constraints, represented by a filled-in triangle, designate sub-part/super-part constraints. In Fig. 1a several states make up a region, and a longitude and latitude together constitute a geographic coordinate. The functional constraints associated with the aggregations allow a state to be part of at most one region and require that a longitude and latitude together correspond to one and only one geographic coordinate.

Formally, a populated semantic-data-model instance is an interpretation for a first-order language. Each object set and value set is a unary predicate (e.g., *State*(x) and *Latitude*(x)), and each m-ary relationship set is an m-ary predicate (e.g., *State-StateCapital*(x, y) and *Location-GeographicCoordinate*(x, y)). The domain for the interpretation is the set of object identifiers and values in the populated semantic-data-model instance (or more generally the set of potential object identifiers and potential values for the semantic-data-model instance). The constraints are closed, well-formed formulas (e.g., $\forall x(\mathit{State}(x) \Rightarrow \exists! y \mathit{State\text{-}State\ Capital}(x, y))$). A populated semantic-data-model instance whose data satisfies all the constraints is said to be a *model* – a valid semantic-data-model instance.

Cross-references

- ▶ Entity Relationship Model
- ▶ Extended Entity-Relationship Model
- ▶ Hierarchical Data Model
- ▶ Network Data Model
- ▶ Object Data Models
- ▶ Object-Role Modeling
- ▶ Ontology
- ▶ Unified Modeling Language

Recommended Reading

1. Peckham J. and Maryanski F. Semantic data models. ACM Comput. Surv., 20(3):153–189, September 1988.

Semantic Image Retrieval

▶ Annotation-based Image Retrieval

Semantic Inference in Audio

▶ Audio Content Analysis

Semantic Mapping Composition

▶ Schema Mapping Composition

Semantic Matching

Fausto Giunchiglia, Pavel Shvaiko,
Mikalai Yatskevich
University of Trento, Trento, Italy

Definition

Semantic matching: given two graph representations of ontologies G1 and G2, compute N1 × N2 *mapping elements* $\langle ID_{i,j}, n1_i, n2_j, R'\rangle$, with $n1_i \in$ G1, $i = 1,\ldots,$N1, $n2_j \in$ G2, $j = 1,\ldots,$N2 and R' the strongest *semantic relation* which is supposed to hold between the *concepts at nodes* $n1_i$ and $n2_j$.

A *mapping element* is a 4-tuple $\langle ID_{ij}, n1_i, n2_j, R\rangle$, $i = 1,\ldots,$N1; $j = 1,\ldots,$N2; where ID_{ij} is a unique identifier of the given mapping element; $n1_i$ is the i-th node of the first graph, N1 is the number of nodes in the first graph; $n2_j$ is the j-th node of the second graph, N2 is the number of nodes in the second graph; and R specifies a semantic relation which is supposed to hold between the concepts at nodes $n1_i$ and $n2_j$.

The *semantic relations* are within *equivalence* ($=$), *more general* ($\sqsupseteq$), *less general* ($\sqsubseteq$), *disjointness* ($\perp$) and *overlapping* ($\sqcap$). When none of the above mentioned relations can be explicitly computed, the special *idk* (I don't know) relation is returned. The relations are ordered according to decreasing binding strength, i.e., from the strongest ($=$) to the weakest (*idk*), with more general and less general relations having equal binding power. The semantics of the above relations are the obvious set-theoretic semantics.

Concept of a label is the logical formula which stands for the set of data instances or documents that one would classify under a label it encodes. *Concept at a node* is the logical formula which represents the set of data instances or documents which one would classify under a node, given that it has a certain label and that it is in a certain position in a graph.

Historical Background

An ontology typically provides a vocabulary that describes a domain of interest and a specification of the meaning of terms used in the vocabulary. Depending on the precision of this specification, the notion of ontology encompasses several data and conceptual models, for example, classifications, database schemas, or fully axiomatized theories. In open or evolving systems, such as the semantic web, different parties would, in general, adopt different ontologies. Thus, just using ontologies, just like using XML, does not reduce heterogeneity: it raises heterogeneity problems to a higher level. Ontology matching is a plausible solution to the semantic heterogeneity problem faced by information management systems. Ontology matching aims at finding correspondences or mapping elements between semantically related entities of the input ontologies. These mapping elements can be used for various tasks, such as ontology merging, query answering, data translation, etc. Thus, matching ontologies enables the knowledge and data expressed in the matched ontologies to interoperate [6].

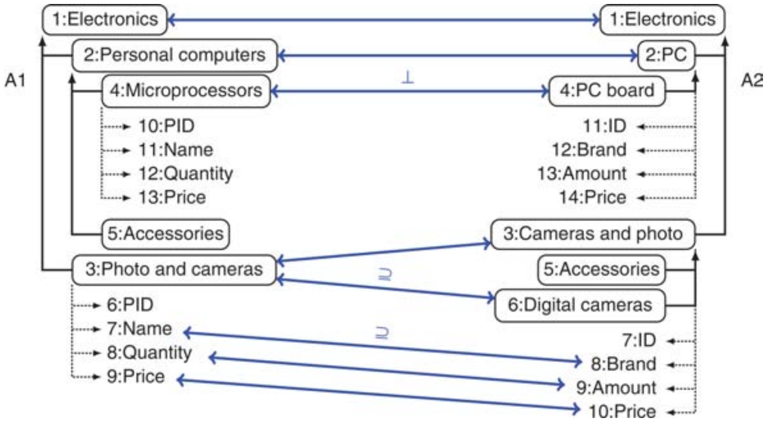
Many diverse solutions of matching have been proposed so far, see [7,17,18] for recent surveys which addressed the matching problem from different perspectives, including databases, artificial intelligence and information systems; while the major contributions of the last decades are provided in [2,14,19]. Some examples of individual approaches addressing the matching problem can be found in [4,5,8,15,16]. (See http://www.ontologymatching.org for a complete information on the topic.) Finally, ontology matching has been given a book account in [6]. This work provided a uniform view on the topic with the help of several classifications of the available methods, discussed these methods in detail, etc. In particular, the matching methods are primarily classified and further detailed according to (i) the input of the algorithms, (ii) the characteristics of the matching process and (iii) the output of the algorithms.

The work in [10] mixed the process dimension of matching together with the output dimension and classified matching approaches into *syntactic* and *semantic*. Syntactic are those approaches that rely on purely syntactic matching methods, e.g., edit distance between strings, tree edit distance. The semantic category, in turn, represents methods that work with concepts and compare their meanings in order to compute mapping elements. However, these have been also constrained by a second condition dealing with the output dimension: syntactic techniques return coefficients in the [0 1] range, while semantic techniques return logical relations, such as equivalence, subsumption (and justified by deductive techniques for instance). The work in [4] provided a first implementation of semantic matching.

Foundations

In order to motivate the matching problem two simple XML schemas are used. These are represented as trees in Fig. 1 and exemplify one of the possible situations which arise, for example, when resolving a schema integration task. Suppose an e-commerce company A1 needs to finalize a corporate acquisition of another company A2. To complete the acquisition, databases of the two companies have to be integrated. The documents of both companies are stored according to XML schemas A1 and A2, respectively. A first step in integrating the schemas is to identify candidates to be merged or to have taxonomic relationships under an integrated schema. This step refers to a process of ontology (schema) matching. For example, the elements with labels *Personal_Computers* in A1 and *PC* in A2 are the candidates to be merged, while the element with label *Digital_Cameras* in A2 should be subsumed by the element with label *Photo_and_Cameras* in A1.

Consider semantic matching as first motivated in [10] and implemented within the S-Match system [13]. Specifically, a schema-based solution is discussed,



Semantic Matching. Figure 1. Two simple XML schemas. The XML elements are shown in rectangles with rounded corners, while attributes are shown without them. Numbers before the labels of tree nodes are the unique identifiers of the XML elements and attributes. In turn, the mapping elements are expressed by arrows. By default, their relation is =; otherwise, these are mentioned above the arrows.

where only the schema information is exploited. It is assumed that all the data and conceptual models, e.g., classifications, database schemas, ontologies, can be generally represented as graphs. This allows for the statement and solution of a *generic (semantic) matching problem* independently of specific conceptual or data models, very much along the lines of what is done, for example, in Cupid [15].

The semantic matching takes as input two graph representations of ontologies and returns as output logical relations, e.g., equivalence, subsumption (instead of computing coefficients rating match quality in the [0 1] range, as it is the case with other approaches, e.g., [15,16]), which are supposed to hold between the nodes in the graphs. The relations are determined by (i) expressing the entities of the ontologies as logical formulas, and (ii) reducing the matching problem to a logical validity problem. In particular, the entities are translated into logical formulas which explicitly express the concept descriptions as encoded in the ontology structure and in external resources, such as WordNet. (http://wordnet.princeton.edu/.) This allows for a translation of the matching problem into a logical validity problem, which can then be efficiently resolved using (sound and complete) state of the art satisfiability solvers.

Consider tree-like structures, e.g., classifications, and XML schemas. Real-world ontologies are seldom trees, however, there are (optimized) techniques, transforming a graph representation of an ontology into a tree representation, e.g., the graph-to-tree operator of Protoplasm [3]. From now on it is assumed that a graph-to-tree transformation can be done by using existing systems, and therefore, the focus is on other issues instead.

Consider Fig. 1. "C" is used to denote concepts of labels and concepts at nodes. Also "$C1$" and "$C2$" are used to distinguish between concepts of labels and concepts at nodes in tree 1 and tree 2, respectively. Thus, in A1, $C1_{\mathit{Photo\_and\_Cameras}}$ and $C1_3$ are, respectively, the concept of the label *Photo_and_Cameras* and the concept at node 3. Finally, in order to simplify the presentation whenever it is clear from the context, it is assumed that the formula encoding the concept of label is the label itself. Thus, for example in A2, $\mathit{Cameras\_and\_Photo}_2$ is a notational equivalent of $C2_{\mathit{Cameras\_and\_Photo}}$.

The algorithm inputs two ontologies and outputs a set of mapping elements in four macro steps. The first two steps represent the pre-processing phase. The third and the fourth steps are the element level and structure level matching, respectively. (Element level matching techniques compute mapping elements by analyzing entities in isolation, ignoring their relations with other entities. Structure level techniques compute mapping elements by analyzing how entities are related together.)

Step 1. For all labels L *in the two trees, compute* concepts of labels. The labels at nodes are viewed as concise descriptions of the data that is stored under the nodes. The meaning of a label at a node is computed by taking as input a *label*, analyzing its real-world semantics, and returning as output a *concept of the label*, C_L. Thus, for example, $C_{\mathit{Cameras\_and\_Photo}}$ indicates a shift from the natural language ambiguous label *Cameras_and_Photo* to the concept $C_{\mathit{Cameras\_and\_Photo}}$, which codifies explicitly its intended meaning, namely the data which is about cameras and photo. Technically, concepts of labels are codified as propositional logical formulas [9]. First, labels are chuncked into *tokens*, e.g., *Photo_and_Cameras* → ⟨*photo*,*and*,*cameras*⟩; and then, *lemmas* are extracted from the tokens, e.g., *cameras* → *camera*. Atomic formulas are WordNet *senses* of lemmas obtained from single words (e.g., cameras) or multiwords (e.g., digital cameras). Complex formulas are built by combining atomic formulas using the connectives of set theory. For example, $C2_{\text{Cameras\_and\_Photo}} = \langle \mathit{Cameras}, \mathit{senses}_{\text{WN\#2}} \rangle \sqcup \langle \mathit{Photo}, \mathit{senses}_{\text{WN\#1}} \rangle$, where sensesWN#2 is taken to be disjunction of the two senses that WordNet attaches to *Cameras*, and similarly for *Photo*. The natural language conjunction "and" has been translated into the logical disjunction "⊔."

Step 2. For all nodes N *in the two trees, compute* concepts at nodes. During this step the meaning of the positions that the labels at nodes have in a tree is analyzed. By doing this, concepts of labels are *extended* to *concepts at nodes*, C_N. This is required to capture the knowledge residing in the structure of a tree, namely the context in which the given concept at label occurs. For example, in A2, by writing C_6 it is meant the concept describing all the data instances of the electronic photography products which are digital cameras. Technically, concepts of nodes are written in the same propositional logical language as concepts of labels. XML schemas are hierarchical structures where the path from the root to a node uniquely identifies that node (and also its meaning). Thus, following an *access criterion* semantics, the logical formula for a concept at node is defined as a conjunction of concepts of labels located in the path from the given node to the root. For example, $C2_6 = Electronics_2 \sqcap Cameras\_and\_Photo_2 \sqcap Digital\_Cameras_2$.

Step 3. For all pairs of labels in the two trees, compute relations *among atomic* concepts of labels. Relations between concepts of labels are computed with the help of a library of element level semantic matchers. These matchers take as input two atomic concepts of labels and produce as output a semantic relation between them. Some of them are re-implementations of the well-known matchers used, e.g., in Cupid. The most important difference is that these matchers return a semantic relation (e.g., $=$, $\sqsupseteq$, $\sqsubseteq$), rather than an affinity level in the [0 1] range, although sometimes using customizable thresholds.

The element level semantic matchers are briefly summarized in Table 1. The first column contains the names of the matchers. The second column lists the order in which they are executed. The third column introduces the matcher's approximation level. The relations produced by a matcher with the first approximation level are always correct. For example, $name \sqsupseteq brand$ returned by the *WordNet* matcher. In fact, according to WordNet *name* is a hypernym (superordinate word) of *brand*. In WordNet *name* has 15 senses and *brand* has 9 senses. Some sense filtering techniques are used to discard the irrelevant senses for the given context, see [13] for details. Notice that matchers are executed following the order of increasing approximation. The fourth column reports the matcher's type, while the fifth column describes the matcher's input. As from Table 1, there are two main categories of matchers. *String-based* matchers have two labels as input. These compute only equivalence relations (e.g., equivalence holds if the weighted distance between the input strings is lower than a threshold). *Sense-based* matchers have two WordNet senses as input. The *WordNet* matcher computes equivalence, more/less general, and disjointness relations. The result of step 3 is a matrix of the relations holding between atomic concepts of labels. A part of this matrix for the example of Fig. 1 is shown in Table 2.

Step 4. For all pairs of nodes in the two trees, compute relations *among* concepts at nodes. During this step, initially the tree matching problem is reformulated into a set of node matching problems (one problem for each pair of nodes). Then, each node matching problem is translated into a propositional validity problem. Semantic relations are translated into propositional connectives in an obvious way, namely: equivalence ($=$) into equivalence ($\leftrightarrow$), more general ($\sqsupseteq$) and less general ($\sqsubseteq$) into implication ($\leftarrow$ and $\rightarrow$, respectively) and disjointness ($\perp$) into negation ($\neg$) of the conjunction ($\wedge$). The criterion for determining whether a relation holds between concepts at nodes is the fact that it is entailed by the premises. Thus, it is necessary to prove that the following formula:

$$axioms \longrightarrow rel(context_1, context_2) \qquad (1)$$

is valid, namely that it is *true* for all the truth assignments of all the propositional variables occurring in it. $context_1$ is the concept at node under consideration in tree 1, while $context_2$ is the concept at node under consideration in tree 2. *rel* (within $=$, $\sqsubseteq$, $\sqsupseteq$, $\perp$) is the semantic relation (suitably translated into a propositional connective) to be proved to hold between $context_1$ and $context_2$. The *axioms* part is the conjunction of all the relations (suitably translated) between atomic concepts of labels mentioned in $context_1$ and $context_2$. The validity of formula (1) is checked by proving that its negation is unsatisfiable. Specifically, it is done, depending on a matching task, either by using ad hoc reasoning techniques or standard propositional satisfiability solvers.

Semantic Matching. Table 1. Element level semantic matchers

Matcher name	Execution order	Approximation level	Matcher type	Schema info
WordNet	1	1	Sense-based	WordNet senses
Prefix	2	2	String-based	Labels
Suffix	3	2	String-based	Labels
Edit distance	4	2	String-based	Labels
Ngram	5	2	String-based	Labels

From the example in Fig. 1, trying to prove that $C2_6$ is less general than $C1_3$, requires constructing formula (2), which turns out to be unsatisfiable, and therefore, the less general relation holds.

$$\begin{aligned}&((Electronics_1 \leftrightarrow Electronics_2) \wedge (Photo_1 \leftrightarrow Photo_2)\wedge\\&(Cameras_1 \leftrightarrow Cameras_2)\wedge\\&(Digital\_Cameras_2 \rightarrow Cameras_1)) \wedge (Electronics_2\wedge\\&(Cameras_2 \vee Photo_2) \wedge Digital\_Cameras_2)\wedge\\&\neg(Electronics_1 \wedge (Photo_1 \vee Cameras_1) \qquad (2)\end{aligned}$$

A part of this matrix for the example of Fig. 1 is shown in Table 3.

Finally, notice that the algorithm returns N1 × N2 correspondences, therefore the cardinality of mapping elements is *one-to-many*. Also, these, if necessary, can be decomposed straightforwardly into mapping elements with the *one-to-one* cardinality.

Key Applications

Semantic matching is an important operation in traditional metadata intensive applications, such as *ontology integration*, *schema integration*, or *data warehouses*. Typically, these applications are characterized by heterogeneous structural models that are analyzed and matched either manually or semi-automatically at design time. In such applications matching is a prerequisite of running the actual system. A line of applications that can be characterized by their dynamics, e.g., *agent communication*, *peer-to-peer information sharing*, *web service composition*, is emerging. Such applications, contrary to traditional ones, require (ultimately) a run time matching operation and take advantage of more explicit conceptual models [18].

Semantic Matching. Table 2. The matrix of semantic relations holding between atomic concepts of labels

	$Cameras_2$	$Photo_2$	$Digital\_Cameras_2$
$Photo_1$	*idk*	=	*idk*
$Cameras_1$	=	*idk*	⊒

Semantic Matching. Table 3. The matrix of semantic relations holding between concepts at nodes (the matching result)

	$C2_1$	$C2_2$	$C2_3$	$C2_4$	$C2_5$	$C2_6$
$C1_3$	⊑	*idk*	=	*idk*	⊒	⊒

Future Directions

Future work includes development of a fully-fledged *iterative* and *interactive* semantic matching system. It will improve the quality of the mapping elements by iterating and by focusing user's attention on the critical points where his/her input is maximally useful. Initial steps have already been done in this direction by discovering automatically *missing background knowledge* in ontology matching tasks [11]. Also, an *evaluation methodology* is needed, capable of estimating quality of the mapping elements between ontologies with hundreds and thousands of nodes. Initial steps have already been done as well; see [1,12] for details. Here, the key issue is that in these cases, specifying reference mapping elements manually is neither desirable nor feasible task, thus a semi-automatic approach is needed.

Experimental Results

In general, for the semantic matching approach, there is an accompanying experimental evaluation in the corresponding references. Also, there is the Ontology Alignment Evaluation Initiative (OAEI), (http://oaei.ontologymatching.org/.) which is a coordinated international initiative that organizes the evaluation of the increasing number of ontology matching systems. The main goal of the Ontology Alignment Evaluation Initiative is to be able to compare systems and algorithms on the same basis and to allow anyone for drawing conclusions about the best matching strategies. From such evaluations, matching system developers can learn and improve their systems.

Data Sets

A large collection of datasets commonly used for experiments can be found at:http://oaei.ontologymatching.org/.

URL to Code

The OntologyMatching. org contains links to a number of ontology matching projects which provide code for their implementations of the matching operation: http://www.ontologymatching.org/.

Cross-references

► Data Integration
► Data models (including semantic data models)
► Mobile and ubiquitous data management

Recommended Reading

1. Avesani P., Giunchiglia F., and Yatskevich M. A large scale taxonomy mapping evaluation. In Proc. Fourth Int. Semantic Web Conf., 2005, pp. 67–81.
2. Batini C., Lenzerini M., and Navathe S. A comparative analysis of methodologies for database schema integration. ACM Comput. Surv., 18(4):323–364, 1986.
3. Bernstein P., Melnik S., Petropoulos M., and Quix C. Industrial-strength schema matching. ACM SIGMOD Rec., 33(4):38–43, 2004.
4. Bouquet P., Serafini L., and Zanobini S. Semantic coordination: a new approach and an application. In Proc. Second Int. Semantic Web Conf., 2003, pp. 130–145.
5. Doan A., Madhavan J., Dhamankar R., Domingos P., and Halevy A.Y. Learning to match ontologies on the Semantic Web. VLDB J., 12(4):303–319, 2003.
6. Euzenat J. and Shvaiko P. Ontology Matching. Springer, 2007.
7. Gal A. Why is schema matching tough and what can we do about it? ACM SIGMOD Rec., 35(4):2–5, 2006.
8. Gal A., Anaby-Tavor A., Trombetta A., and Montesi D. A framework for modeling and evaluating automatic semantic reconciliation. VLDB J., 14(1):50–67, 2005.
9. Giunchiglia F., Marchese M., and Zaihrayeu I. Encoding classifications into lightweight ontologies. J. Data Semantics, 8:57–81, 2007.
10. Giunchiglia F. and Shvaiko P. Semantic Matching. Knowl. Eng. Rev., 18(3):265–280, 2003.
11. Giunchiglia F., Shvaiko P., and Yatskevich M. Discovering missing background knowledge in ontology matching. In Proc. 17th European Conf. on Artificial Intelligence, 2006, pp. 382–386.
12. Giunchiglia F., Yatskevich M., Avesani P., and Shvaiko P. A large scale dataset for the evaluation of ontology matching systems. Knowl. Eng. Rev., 23:1–22, 2008.
13. Giunchiglia F., Yatskevich M., and Shvaiko P. Semantic matching: algorithms and implementation. J. Data Semantics, 9:1–38, 2007.
14. Larson J., Navathe S., and Elmasri R. A theory of attributed equivalence in databases with application to schema integration. IEEE Trans. Software Eng., 15(4):449–463, 1989.
15. Madhavan J., Bernstein P., and Rahm E. Generic schema matching with Cupid. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 48–58.
16. Noy N. and Musen M. The PROMPT suite: interactive tools for ontology merging and mapping. Int. J. Hum. Comput. Stud., 59 (6):983–1024, 2003.
17. Rahm E. and Bernstein P. A survey of approaches to automatic schema matching. VLDB J., 10(4):334–350, 2001.
18. Shvaiko P. and Euzenat J. A survey of schema-based matching approaches. J. Data Semantics, 4:146–171, 2005.
19. Spaccapietra S. and Parent C. Conflicts and correspondence assertions in interoperable databases. ACM SIGMOD Rec., 20(4):49–54, 1991.

Semantic Modeling and Knowledge Representation for Multimedia Data

Edward Y. Chang
Google Research, Mountain View, CA, USA

Synonyms

Multimedia information retrieval; Image/Video/Music search

Definition

Semantic modeling and knowledge representation is essential to a multimedia information retrieval system for supporting effective data organization and search. Semantic modeling and knowledge representation for multimedia data (e.g., imagery, video, and music) consists of three steps: *feature extraction*, *semantic labeling*, and *features-to-semantics mapping*. Feature extraction obtains perceptual characteristics such as color, shape, texture, salient-object, and motion features from multimedia data; semantic labeling associates multimedia data with cognitive concepts; and features-to-semantics mapping constructs correspondence between perceptual features and cognitive concepts. Analogically to data representation for text documents, improving semantic modeling and knowledge representation for multimedia data leads to enhanced data organization and query performance.

Historical Background

The principal design goal of a multimedia information retrieval system is to return data (images, video clips, or music) that accurately match users' queries (for example, a search for pictures of a deer). To achieve this design goal, the system must first comprehend a user's query concept thoroughly, and then find data in the low-level input space (formed by a set of perceptual features) that match the concept accurately. For traditional relational databases, a query concept is explicitly specified by a user using SQL. For multimedia information retrieval, however, articulating a query concept (e.g., a deer) using low level features (e.g., color, shape, texture, and salient-object features) is infeasible.

Semantic modeling and knowledge representation thus plays a key role in query-concept formulation and query processing for a multimedia query.

The QBIC system [8] introduced in 1995 is the first query-by-example system. QBIC uses color histograms to represent an image/video clip; two images/clips containing similar color histograms are considered to be similar. Such knowledge representation for multimedia data is clearly inadequate. In the subsequent 5 years, many researchers in the *signal processing* and *computer vision* communities proposed techniques to extract perceptual features, such as textures, shapes, and segments of objects, for improve image representation (see [13] for a survey). At the same time, the query-by-example paradigm was applied also to music retrieval.

Query by *just one example* was soon discovered insufficient to represent a query concept. Relevance feedback, a query refinement technique developed by the information retrieval community in the 1970's [12], was then borrowed to provide additional examples to augment the shortcoming of knowledge under-representation. In 2001, the work of [14] showed that relevance feedback could be much improved by using the *kernel methods* [1] with *active learning*. The kernel methods project data from their input space formed by perceptual features to a much higher (possibly infinite) dimensional space, where a linear classifier can be learned to separate desired data (with respect to the query) from the others. The kernel methods enjoy both rich semantic modeling (the linear class boundary in a high-dimensional space represents a non-linear boundary in the input space) and computational efficiency (computation is performed in the projected, linear space). Active learning is applied to select the most ambiguous and diversified training instances along the class boundary to query the user for labels. Once these training instances have been labeled, maximal information is gained for refining the class boundary. This process of active learning continues until the search result is satisfactory. In order to further improve the effectiveness of query-concept learning through active learning, keywords (tagged by users [9] or obtained from query logs [10]) were subsequently integrated into the semantic modeling and knowledge representation framework.

Over a decade of research since QBIC, though productive, has not yielded a large-scale real-world deployment of multimedia information retrieval system. The key reason is that semantic modeling and knowledge representation for multimedia data is intrinsically inter-disciplinary. Its success demands collaborative effort from researchers of *signal processing*, *computer vision*, *machine learning*, and *databases*. Recent works in addressing issues of perceptual similarity [11] and scalability in statistical learning [5] are inter-disciplinary approaches that hold promises to lead to a Web-scale deployment. The survey conducted by [6] provides a complementary view on the historical background.

Foundations

Semantic Modeling

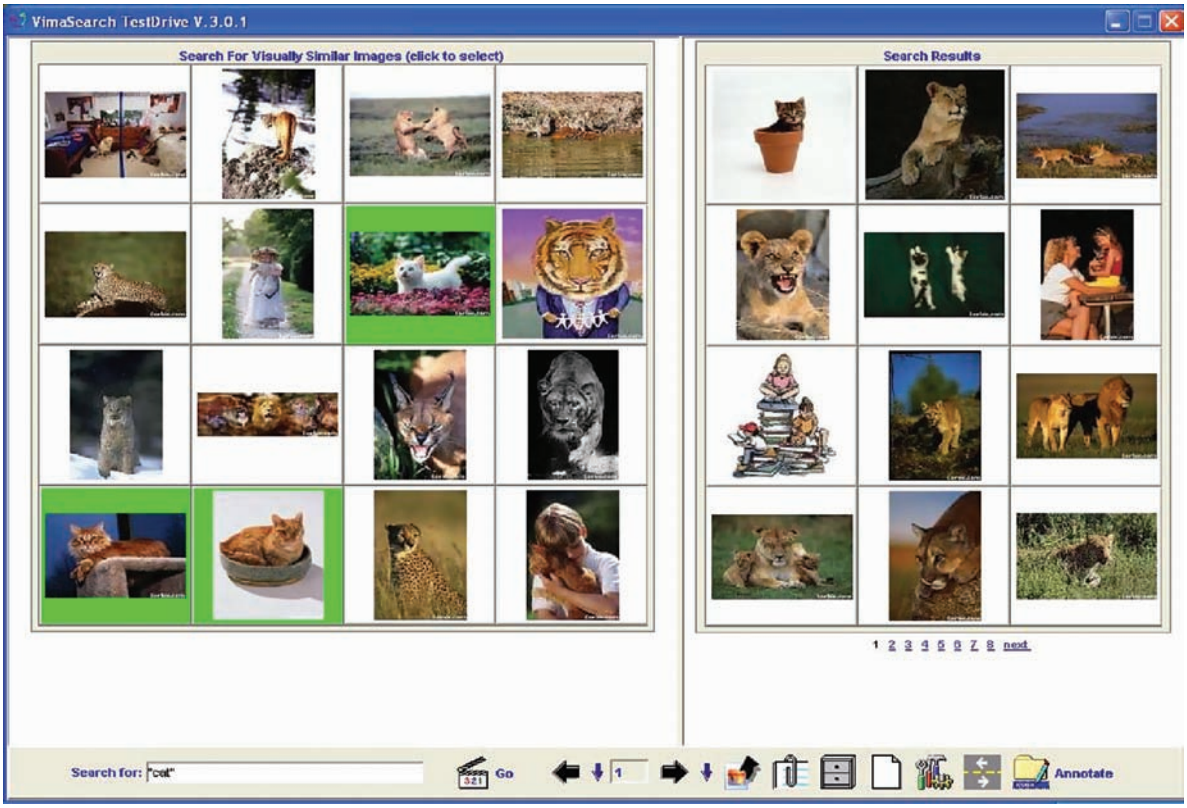
There are two realistic ways for users to specify a multimedia query semantic: *query by keywords* and *query by examples*. In order to support query by keywords, *semantic annotation* provides data with semantic labels (for example, landscape, sunset, animals, and so forth). Several researchers (e.g., [2]) have proposed semi-automatic annotation methods to propagate keywords from a small set of annotated images to the other images. Although semantic annotation can provide some relevant query results, annotation is often subjective and narrowly construed. When it is, query performance may be compromised. To thoroughly understand a query concept, with all of its semantics and subjectivity, a system must obtain the target concept from the user directly via *query-concept learning*. Semantic annotation can assist, but not replace, query-concept learning.

Both semantic annotation and query-concept learning require mapping features to semantics. This semantic modeling consists of three steps. First, a set of perceptual features (e.g., color, texture and shape) is extracted from each training instance. Second, each training feature-vector $\mathbf{x}_i$ is assigned semantic labels g_i. Third, a classifier $f(.)$ is trained by a supervised learning (or semi-supervised learning) algorithm, based on the labeled instances, to predict the class labels of a query instance $\mathbf{x}_q$. Given a query instance $\mathbf{x}_q$ represented by its low-level features, the semantic labels g_q of $\mathbf{x}_q$ can be predicted by $g_q = f(\mathbf{x}_q)$. (About how multimedia data and knowledge can be represented is discussed in the Knowledge Representation section.)

At first it might seem that traditional supervised learning methods could be directly applied to perform *semantic annotation* and *query-concept learning*. Unfortunately, traditional learning algorithms are not adequate to deal with the technical challenges posed by these two tasks. To illustrate, let D denote the number of low-level features. Let N denote the number of training instances, N^+ the number of positive training instances, and N^- the number of negative training instances ($N = N^+ + N^-$). And let U denote the number of unlabeled instances in the repository. Three major technical challenges arise:

1. *Scarcity of training data.* The features-to-semantics mapping problem often comes up against the $D > N$ challenge. For instance, in the query-concept learning scenario, the number of low-level features that characterize an image (D) is greater than the number of training instances that a user can provide (N) via her query history or relevance feedback. The theories underlying "classical" data analysis are based on the assumptions that $D < N$, and N approaches infinity. But when $D > N$, the basic methodology which was used in the classical situation is not similarly applicable [7].
2. *Imbalance of training classes.* The target class in the training pool is typically outnumbered by the non-target classes ($N^- > > N^+$). When the prior of the non-target class dominates the target class, a class prediction favors the non-target class. This skew can substantially reduce recall in search performance [16].
3. *Scalability.* A typical value of D can be in the order of hundreds, and U can be millions or even billions. Scalability challenges arise in at least two areas. First, searching data among U instances in a high-dimensional space is inefficient [3]. Second, when $U > > N$, training data may under-represent the knowledge required to model semantics.

Effective techniques for addressing the above challenges are inter-disciplinary. The *signal processing* and *computer vision* communities devise algorithms to extract useful features to represent multimedia data. The *machine learning* community develops models that can map features to semantics both effectively and


VimaSearch TestDrive V.3.0.1
Search For Visually Similar Images (click to select)
Search Results
1 2 3 4 5 6 7 8 next
Search for: "cat"
Go
1
Annotate

Semantic Modeling and Knowledge Representation for Multimedia Data. Figure 1. Cat query initial screen.

efficiently. The *database* community improves indexing, metadata fusion, and query processing techniques to deal with scalability issues. All these endeavors may consult experts in *neural processing* or *cognitive science* (e.g., [15]) to develop representations and models that fit human perception.

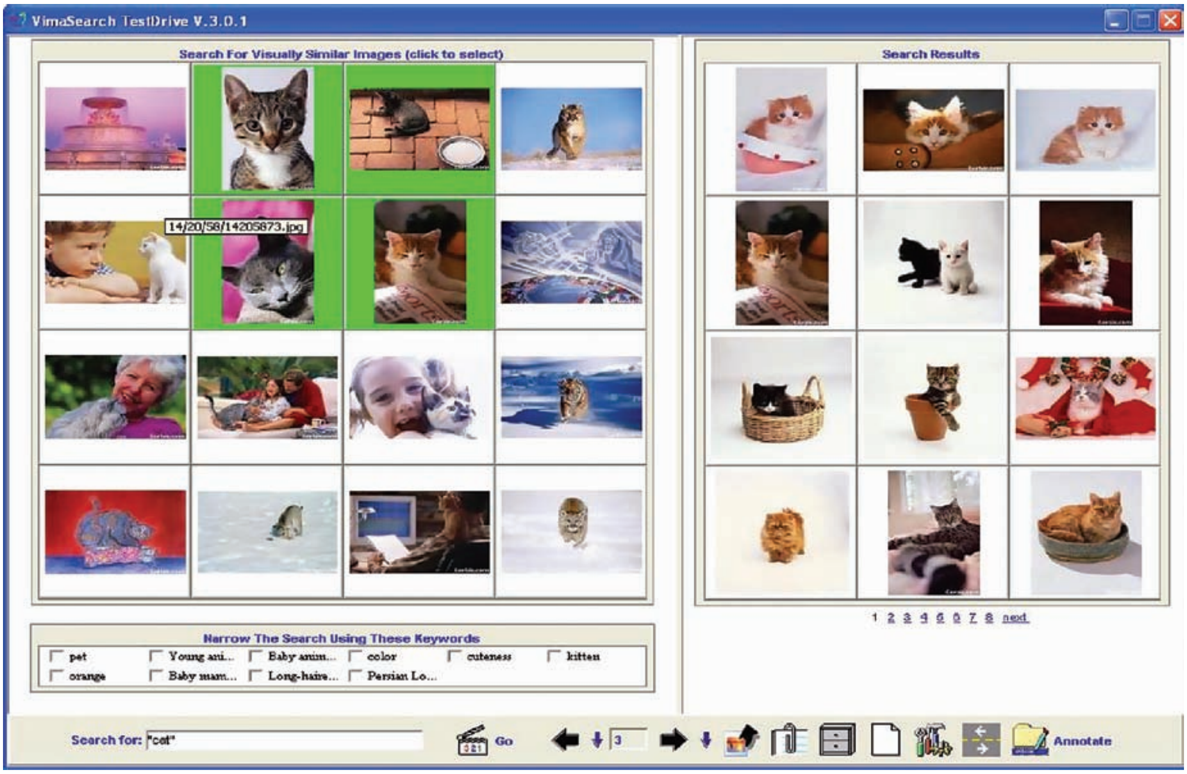
Knowledge Representation

As mentioned, a piece of multimedia data can be represented at two levels: low-level features and high-level semantics/concepts. A set of low-level features consists of perceptual features, and these features can be put in the form of a vector or a bag. High-level concepts are organized into an ontology structure, depicting relationship between concepts. In between, descriptors can be formulated either explicitly or implicitly to provide building blocks for low-level to high-level mapping and reasoning. For instance, a high-level ski concept can be formed by descriptors of snow, ski equipment, and people. Each of these descriptors is in turn composed of color, texture, shape, or salient-point features. Texts when available can be used to augment low-level perceptual features (e.g., using word "white" to depict the color of the mountain), to label descriptors (e.g., snow), or to directly annotate high-level semantics/concepts (e.g., ski). Statistical methods such as SVMs and Latent Semantic Analysis techniques (e.g., LDA [4]) can be employed to perform mapping between the three levels.

Efforts of standardizing knowledge representation have been embarked on for over a decade by academia and industry. For instance, digital cameras save JPEG files with EXIF (Exchangeable Image File) data. EXIF records camera settings, scene information, and time (and location where a photo is taken in the near future). DOLCE devises descriptive ontology for linguistic and cognitive engineering. MPEG-7 proposes different description granularity to depict multimedia data. Standard knowledge representation is essential for supporting metadata exchange and system interoperability.

Key Applications

The launches of photo and video sharing sites such as Flickr, Google Photos, and YouTube between 2002 and 2008 renewed the interest on multimedia data management. The following applications are in high demand to manage large-scale multimedia data repositories:



Search for: "cat"

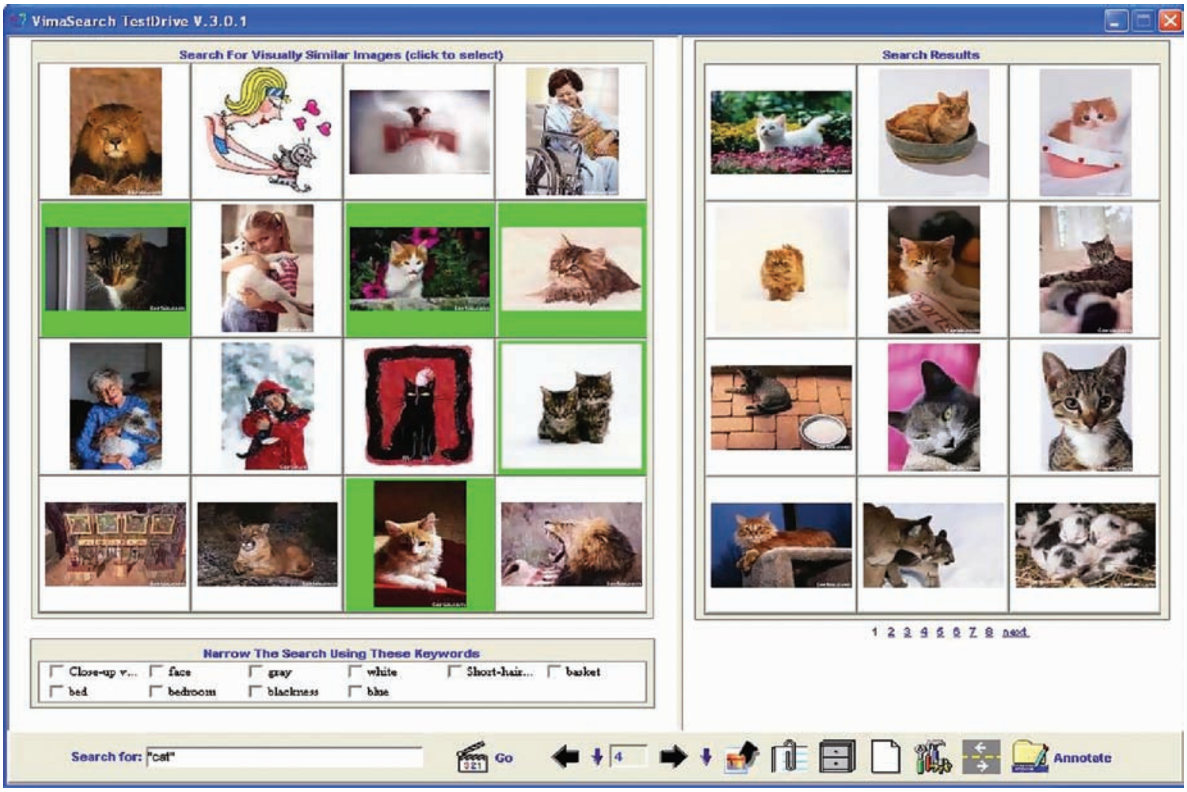
Semantic Modeling and Knowledge Representation for Multimedia Data. Figure 2. Cat query after iteration #2.

1. Content-based Video, Image, Music Search Engines
2. Copy Right Infringement Detection
3. Multimedia Digital Libraries
4. Semi-automatic Photo/Video Annotation/ Classification

An application scenario is used to illustrate how aforementioned science fundamentals can improve multimedia information retrieval. Figures 1–3 show an example query using a *Perception-based Image Retrieval* (PBIR) prototype developed at UC Santa Barbara. The figures demonstrate how a query concept is learned in an iterative process by the PBIR search engine to improve search results. The user interface shows two frames. The frame on the left-hand side is the feedback frame, on which the user marks images relevant to his or her query concept. On the right-hand side, the search engine returns what it interprets as matching this far from the image database.

Most images were annotated by users. To query "cat," one first enters the keyword *cat* in the query box to get the first screen of results in Fig. 1. The right-side frame shows a couple of images containing domestic cats, but several images containing tigers or lions. This is because many tiger/lion images were annotated with "wild cat" or "cat." To disambiguate the concept, the user clicks on a couple of domestic cat images on the feedback frame (left side, in gray/green borders). The search engine refines the class boundary accordingly, and then returns the second screen in Fig. 2. In this figure, the images in the result frame (right side) have been much improved. All returned images contain a domestic cat or two. After performing another couple of rounds of feedback to make some further refinements, more satisfactory results are shown in Fig. 3.

This example illustrates three critical points. First, keywords alone cannot retrieve images effectively because words may have varied meanings or senses. This is called the *word-aliasing* problem. Second, the number of labeled instances that can be collected from a user is limited. Through three feedback iterations, it is possible to gather just *16* × *3* = *48* training instances, whereas the feature dimension of this dataset is more than one hundred. Since most users would not be willing to give more than three iterations of feedback,


VimaSearch TestDrive V.3.0.1
Search For Visually Similar Images (click to select)
Search Results
1 2 3 4 5 6 7 8 next
Narrow The Search Using These Keywords
□ Close-up v... □ face □ gray □ white □ Short-hair... □ basket
□ bed □ bedroom □ blackness □ blue
Go
4
Annotate

Search for: "cat"

Semantic Modeling and Knowledge Representation for Multimedia Data. Figure 3. Cat query after iteration #3.

the system encounters the problem of scarcity of training data. Third, the negatives outnumber the relevant or positive instances being clicked on. This is known as the problem of imbalanced training data. Besides, there are a large number of images in the repository. To achieve real-time performance in query refinement and in search, efficiently indexing schemes are needed to reduce search space.

Future Directions

Major advancements in three areas are necessary before large-scale multimedia systems can be realistic: *accurate and efficient object segmentation*, *scalable statistical learning*, and *high-dimensional indexing*. For details please consult the section of Foundations.

Cross-references

▶ Nearest Neighbor Query in Spatio-temporal Databases

Recommended Reading

1. Aizerman M.A., Braverman E.M., and Rozonoer L.I. Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control, 25:821–837, 1964.
2. Barnard K. and Forsyth D. Learning the semantics of words and pictures. Int. Conf. on Computer Vision, 2:408–415, 2000.
3. Beyer K., Goldstein J., Ramakrishnan R., and Shaft U. When is Nearest Neighbor meaningful. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 217–235.
4. Blei D.M., Ng A., and Jordan M. Latent Dirichlet allocation. J. Machine Learning Res. 3, 2003.
5. Chang E.Y. et al. Parallelizing support vector machines on distributed computers. In Proc. Advances in Neural Inf. Proc. Syst. 20, Proc. 21st Annual Conf. on Neural Inf. Proc. Syst., 2007.
6. Datta R., Joshi D., Li J., and Wang J.Z. Image retrieval: ideas, influences, and trends of the new age. ACM Comput. Surv., 40(65), 2008.
7. Donoho D.L. Aide-Memoire. High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality (American Math. Society Lecture). In Math Challenges of the 21st Century Conference, 2000.
8. Flickner M. et al. Query by Image and Video Content: QBIC system. IEEE Comput., (28), 1995.
9. Goh K., Chang E.Y., and Lai W.-C. Concept-dependent multimodal active learning for image retrieval. In Proc. 12th ACM Int. Conf. on Multimedia, 2004, pp. 564–571.
10. Hoi C.-H. and Lyu M.R. A novel log-based relevance feedback technique in content-based image retrieval. In Proc. 12th ACM Int. Conf. on Multimedia, 2004, pp. 24–31.
11. Li B. and Chang E.Y. Discovery of a perceptual distance function for measuring image similarity. ACM Multimedia Syst. J. (Special Issue on Content-Based Image Retrieval), 8(6):512–522, 2003.
12. Rocchio J.J. Relevance feedback in information retrieval. In The SMART Retrieval System – Experiments in Automatic Document Processing, G. Salton (ed.). Prentice-Hall, Chapter 14, 1971, pp. 313–323.
13. Rui Y., Huang T.S., and Chang S.-F. Image retrieval: current techniques, promising directions and open issues. J. Visual Commn. Image Representation, 1999.
14. Tong S. and Chang E.Y. Support vector machine active learning for image retrieval. In Proc. 9th ACM Int. Conf. on Multimedia, 2001, pp. 107–118.
15. Tversky A. Features of similarity. Psychol. Rev., 84:327–352, 1997.
16. Wu G. and Chang E.Y. KBA: Kernel Boundary Alignment considering imbalanced data distribution. IEEE Trans. Knowl. Data Eng., 17(6):786–795, 2005.

Semantic Modeling for Geographic Information Systems

Christine Parent[1], Stefano Spaccapietra[2], Esteban Zimányi[3]

[1]University of Lausanne, Lausanne, Switzerland

[2]EPFL, Lausanne, Switzerland

[3]Free University of Brussels, Brussels, Belgium

Synonyms

Conceptual modeling; Geographical databases; GIS; Conceptual modeling for Geographic Information System; Conceptual modeling for Spatio-temporal applications

Definition

Semantic modeling denotes the activity of designing and describing the structure of a data set using a *semantic data model*. Semantic data models (also known as *conceptual data models*) are data models whose aim is to provide designers with modeling constructs and rules that are well suited for representing the user's perception of data in the application world, abstracting from implementation concerns. They contrast with *logical* and *physical data models*, whose aim is to organize data in a way that is easily manageable by a computer. The most popular semantic data models are UML, a de facto standard, and ER (Entity-Relationship), still widely used in many design methodologies and favored by the academic community.

Semantic models were first created in the database community in the 1980s. They started to be developed for Geographical Information Systems (GIS) in the 1990s. Their aim is the same as for traditional databases, to free GIS users from the specificities of system-oriented data models and proprietary file formats (e.g., spaghetti and topological data models, triangulated irregular network (TIN) models, shape files, and raster models). While a number of semantic data models for geographic data have been developed, rapidly the focus has shifted from supporting spatial data to supporting data with both spatial and temporal features, leading to the development of several spatiotemporal semantic data models. Despite the fact that current GIS and Database Management Systems (DBMS) provide poor support for temporal features, semantic modeling advocates that space and time aspects are intrinsically correlated in the application world.

Historical Background

Most people consider the 1976 paper by Peter Chen [5], defining the basic ideas of Entity-Relationship modeling, as the foundational milestone for semantic modeling. The paper had indeed an enormous effect on the database design community, leading to considerable developments to further extend the semantic capabilities of the approach. It took more than 15 years to see the same idea spreading in the academic GIS community with, for example, the 1993 MODUL-R formalism [4], which extended with spatial data the ER approach used in the leading French design methodology, Merise. Further work on MODUL-R eventually resulted in the Perceptory UML-based approach and tool [3]. Semantic models for GIS bloomed in the 1990s, basically splitting into approaches stemming from the object-oriented paradigm (e.g., [6,16]) and approaches following the ER or the UML paradigm (e.g., MADS [10], STER [15], GeoUML [2]). A survey of many spatial data models may be found in [12]. The industrial and application world has also developed GIS data modeling specifications to help promoting interoperability between different systems and different applications. The Open Geospatial Consortium (OGC) and the International Standards Organization (ISO) have produced specifications supporting conceptual modeling for data with spatial (and some temporal) features (http://www.opengeospatial.org/).

Thanks to the development of ubiquitous and mobile computing on the one hand, and of sensors and GPS technologies on the other hand, large-scale capture of the evolving position of mobile objects has become technically and economically feasible. This opened new perspectives for a large number of applications (e.g., from transportation and logistics to ecology and anthropology) built on the knowledge of objects' movements. Typical examples of moving objects include cars, persons, and planes equipped with a GPS device, animals bearing a transmitter whose signals are captured by satellites, and parcels tagged with RFIDs. This fostered the interest in spatio-temporal models, rather than purely spatial or purely temporal models at the logical and semantic levels. Güting's approach [8] defined a set of data types and associated operators for moving objects (points and surfaces), which allows one to record, for example, the changing geometry of pollution clouds and flooding waters. At the semantic level, examples of spatiotemporal models include MADS [10], Perceptory [3], STUML [15], STER [15], and ST USM [9]. Extending the limited capabilities of commercial data management systems, some research prototype systems [1,11] do provide nowadays support for storing and querying the position of a moving object all along the lifespan of the object. The latest developments in this domain are the management of trajectories, which adds a semantic interpretation to the movement of objects of kind moving point [13]. Trajectory management is important in many application domains, e.g., for addressing traffic management issues, building social models of people's movements within a city, and optimizing the localization of resources (e.g., communication antennas, shops, advertisement panels) that have to be available to moving customers.

Foundations

Requirements for Semantic Modeling of Spatial Data

Semantic modeling of spatial data requires concepts for the description of both the discrete and the continuous view of space, in a seamlessly integrated way. The *discrete view* (or *object-based view*) is the one that sees space as filled by objects with a defined location and shape. Parts of space where no object is located are considered as empty. This view typically serves application requests asking where certain objects are located, or which objects are located in a given surface. On the other hand, the *continuous view* (or *field-based view*) is the one that sees space as a continuum, holding properties whose values depend on the location in space but not on any specific object (i.e., the value for the property is given by a function whose domain is a spatial extent). Typical examples where this view applies are the recording of continuous phenomena such as temperature, altitude, soil coverage, etc. Both views are important for applications, which may use one or the other, or both simultaneously.

Assuming the discrete view, any traditional database schema can be enriched to become a spatiotemporal database schema by including the description of the spatial and/or temporal properties of the real-world phenomena represented in the schema. Consider, for instance, a Building object type, with properties name, address, usage, architect, and owner. Adding positional information on the geographic location of the building (e.g., its coordinates in some spatial reference system) turns Building into a spatial object type. If one adds information characterizing the existence of the building in time (e.g., when construction was first decided, when construction started, when it was completed, when it was abandoned, and when it was demolished), Building becomes a temporal object type. Space and time are independent dimensions. Some data may have spatial features, some may have temporal features, some may have both, and some may have none.

Objects, be they spatial or not, can have spatial properties, i.e., properties whose value domain is composed of spatial values rather than alphanumeric values. Spatial values conform to spatial data types (see the entry in this encyclopedia), e.g., point, line, polyline, surface. For example, a Building object type can have a property nearestFireStation whose value for each building is the geographic location of the nearest fire station, e.g., a spatial value composed of two spatial coordinates defining a point.

Most basic types for space are Point, Line, and Surface (and volume for 3D databases). However, applications may require more than simple spatial data types. Some spatial objects have extents (the term "extent" denotes the set of points that an object occupies in space) that are made up of a set of elementary extents. For example, an archipelago is a set of surfaces; many coastal countries do have islands too; and facility networks may be represented by connected sets of lines. Moreover, some spatial objects have complex extents made up of a heterogeneous set of spatial values. For example, an avalanche zone is described by a surface and a set of oriented lines describing, respectively, its maximal extent and the usual avalanche paths. Similarly, a river may be described by lines when its bed is narrow and by surfaces when it is broad. Therefore, the set of spatial data types should include types for homogeneous or heterogeneous collections, like PointSet, LineSet, SurfaceSet, or SpatialHeterogeneousSet. The whole set of spatial data types is organized into a generalization hierarchy with generic data types, in order to support spatial object types whose extent may be of different types depending on the instance. For example, the object type City may contain larges cities represented by a surface and small ones represented by a point. The spatial extent of City could then be described by a generic spatial data type that would contain points and surfaces. The Open Geospatial Consortium (OGC) has defined such a hierarchy of spatial data types.

Geographical applications often need to enforce spatial or temporal constraints between spatial or temporal features. For example, harbors should be located along water bodies and bridges on roads or railways. Therefore, a spatial data model should support constructs allowing designers to specify constraints that will be automatically enforced by the system. A first kind of construct is the spatial (or temporal) relationship type. They link two spatial (and/or temporal) object types and bear a spatial (and/or temporal) condition that the linked objects must obey. Typically, conditions express topological relationships (e.g., inclusion, disjointedness, overlapping), metric relationships (e.g., based on distance), orientation relationships (e.g., North of), or the temporal predicates defined by Allen (e.g., during). Applications may need two different kinds of these spatial and temporal relationships:

- Spatially/temporally constraining relationship types: Users can link two spatial objects by a spatially constraining relationship only if their spatial/temporal extents abide by the condition.
- Derived spatial/temporal relationship types: The system automatically creates the instances of the relationship for all couples of objects that satisfy the condition.

Moving and deforming objects may also be linked by spatial relationships. For instance, an aeronautic database may need recording the trajectories of planes when they cross storms, the two being moving objects. In these cases, the condition of the relationship type is spatiotemporal: It bears on the location and the time.

Applications may also need constraints between composite and component elements. For example, a spatial aggregation relationship may enforce that the extent of the composite object is made up by the union of the extents of the component objects, as in a spatial aggregation linking the spatial object types Country and District. Another example is restricting the spatial (or temporal) values of attributes to be within the spatial (or temporal) extent of the object to which they belong. For example, the values of the spatial attribute major Cities (a multivalued attribute of type Point) of the object type Country should be within the spatial extent of the country. This kind of constraint may be frequent, but it is not always the case. Refer for example to the Building spatial object type with the spatial attribute nearestFireStation. Therefore, the data model should not automatically and implicitly enforce these constraints. It should provide designers with a means for explicitly specifying which constraints should be enforced.

The modeling of the continuous view of space requires another construct for properties that are defined on a spatial extent and whose value depends on the exact location (point) of the spatial extent. The spatial extent may be the whole space covered by the database or a specific extent. For example, the water quality of a river exists only in the spatial extent of the river course. On the other hand, temperature, soil, and land coverage are information that exist and may be measured (if relevant) at any point of the geographical space covered by the database. *Field-based* models are well suited for applications that perceive the real world exclusively through varying properties. For the many applications that use both the discrete and continuous views, several spatial data models provide a predominant discrete view (i.e., based on spatial objects) in addition to a special construct for representing varying properties, the *space-varying attribute*, which is a function from a spatial extent to a range of values. Any object and relationship, be it spatial or not, should be able to bear space-varying attributes. Moreover, the range of space-varying attributes may be simple (e.g., elevation) or complex (e.g., weather composed of temperature, pressure, and rainfall), monovalued (e.g., altitude) or multivalued (e.g., insects in forests, assuming this information is captured using a space unit large enough to be the home of several kinds of insect, e.g., using cells of 1 m^2).

Another important requirement for space modeling is the ability to describe data at different granularity or resolution, for example to be able to support applications working with maps at different scales.

Finally, an essential requirement is the ability to model spatial features of a phenomenon irrespectively of the fact that the phenomenon has been modeled as an object, a relationship, or an attribute. This orthogonality of the space modeling dimension with the data structure modeling dimension is what avoids making the designs in the two dimensions dependent on each other.

Survey of Current Semantic Modeling Approaches

Semantic models are typically developed in the academic world. For example, MADS [10] has been purposely developed to match all the requirements discussed in the previous section. MADS belongs to the extended ER family of models. Its distinguishing feature is the full support for multiple perceptions and multiple representations of the same real-world objects. Another distinguishing feature of MADS is its support of explicit relationships equipped with topological and synchronization constraints. Multiple perceptions and representations are also supported, to a more limited extent, by Perceptory [3], an UML extension targeted to support spatiotemporal analysis in a data-warehousing framework. STUML [15] and GeoUML [2] are other UML-based approaches, although without multi-perception support. Other spatiotemporal approaches include ST USM [9], very similar to MADS but emphasizing support of multigranularity, and STER [15], another extended ER formalism which supports both valid and transaction time but, compared to MADS, is weaker in data structures. Most of these academic proposals have been implemented in prototypes, but, with the exception of Perceptory, they have not yet turned in commercial products. These proposals deal with 2D data.

A very different approach, known as spatial constraint database modeling, relies on mathematical equations to define spatial extents. Some existing prototype systems (e.g., DEDALE [7]) use this approach.

Key Applications

Semantic modeling is an essential capability for organizations that need to develop a database that provides different applications and different categories of users with different sets of data, possibly organized in different ways. Designing a database in such a complex environment is a very challenging task, as has been extensively proven in traditional data management. Adding spatial features makes the design task even more complex, in particular since this inevitably leads to adding also temporal features. Indeed, what most applications in the geographical domain need to analyze is the temporal evolution of the spatial features of interest. Cartographic applications are the most traditional ones, but today the focus is rather on all kinds of planning and forecasting services to citizens and the society at the municipal, regional, and statewide levels. Examples of such services include environmental control management and global warming. Given the cost of developing databases for these applications and the need for people in charge (politicians and managers) to be successful, it is of the highest importance that the design of an operational database is carried out using the most suitable tools. Semantic modeling is the key to a successful design that determines what data are needed, to be complemented afterwards in the implementation phase by addressing performance aspects in order to guarantee that the data can be used effectively.

Semantic modeling is also the key to all data exchange, reuse, and integration efforts. Whether in database terms, as discussed here, or in ontological terms, semantic modeling is the kernel of the semantic web.

Future Directions

The economic trend towards worldwide enterprise operation and the technical trend towards web-based interoperability will significantly increase the complexity of GIS and the challenges designers will have to overcome. A key help in this context will come from ontologies about spatiotemporal application domains. These ontologies provide a common semantic basis to build repositories of domain knowledge that go beyond traditional enterprise boundaries. In this perspective, the current focus on ontology-assisted semantic modeling and ontology-assisted data integration is leading research into a fruitful direction [14].

In a complementary effort, ontologies and geographic markup languages facilitate the integration of geographical knowledge coming from multiple sources available through the Web. This will contribute to significantly enhance geographical knowledge, benefiting from geo-content actually hidden in Web pages.

Cross-references

- ▶ Data Models
- ▶ Database Design
- ▶ Field-Based Spatial Modeling
- ▶ Geographic Information System
- ▶ Multiple Representation Modeling
- ▶ Spatial Data Types
- ▶ Topological Data Models
- ▶ Topological Relationships

Recommended Reading

1. Almeida V.T., Güting R.H., and Behr T. Querying moving objects in SECONDO. In Proc. 7th Int. Conf. on Mobile Data Management, 2006, pp. 47–51.
2. Belussi A., Negri M., and Pelagatti G. GeoUML: A geographic conceptual model defined through specialization of ISO TC211 standards. In Proc. Tenth EC GI & GIS Workshop, ESDI State of the Art, 2004.
3. Brodeur J., Bédard Y., and Proulx M.J. Modelling geospatial application database using UML-based repositories aligned with international standards in geomatics. In Proc. 8th ACM Symp. on Adavances in Geographic Inf. Syst., 2000, pp. 39–46.
4. Caron C. and Bédard Y. Extending the individual formalism for a more complete modeling of urban spatially referenced data. Comput. Environ. Urban Syst., 17(4):337–346, 1993.
5. Chen P.P. The entity-relationship model: toward a unified view of data. ACM Trans. Database Syst., 1(1):9–36, 1976.
6. Egenhofer M.J. and Frank A.U. Object-oriented modeling for GIS. J. Urban Reg. Inf. Syst. Assoc., 4(2):3–19, 1992.
7. Grumbach S., Rigaux P., Scholl M., and Segoufin L. The DEDALE prototype. In Constraint Databases, Springer, 2000, pp. 365–382.
8. Güting R.H., Böhlen M.H., Erwig M., Jensen C.S., Lorentzos N.A., Schneider M., and Vazirgiannis M. A foundation for representing and querying moving objects. ACM Trans. Database Syst., 25(1):1–42, 2000.
9. Khatri V., Ram S., and Snodgrass R.T. On augmenting database design-support environments to capture the geo-spatio-temporal data semantics. Inf. Syst., 31(2):98–133, 2006.
10. Parent C., Spaccapietra S., and Zimányi E. Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach. Springer, 2006.
11. Pelekis N., Theodoridis Y., Vosinakis S., and Panayiotopoulos T. Hermes – A framework for location-based data management. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 1130–1134.
12. Rios Viqueira J.R., Lorentzos N.A., and Brisaboa N.R. Survey on spatial data modelling approaches. In Spatial Databases: Technologies, Techniques and Trends, Y. Manolopoulos, A. Papadopoulos, M. Vassilakopoulos (eds.). Idea Group, 2005, pp. 1–22.
13. Spaccapietra S., Parent C., Damiani M.L., Macedo J., Porto F., and Vangenot C. A conceptual view on trajectories. Data Knowl. Eng., 65(1):126–146, 2008.

14. Sugumaran V. and Storey V.C. The role of domain ontologies in database design: an ontology management and conceptual modeling environment. ACM Trans. Database Syst., 31(3):1064 –1094, 2006.
15. Tryfona N., Price R., and Jensen C.S. Spatiotemporal conceptual modeling. vol. 2520, In Spatiotemporal Databases: The Chorochronos Approach (chapter 3), Lecture Notes in Computer Science, vol. 2520, 2003, pp. 79–116.
16. Worboys M., Hearnshaw H., and Maguire D. Object-oriented data modelling for spatial databases. Int. J. Geogr. Inf. Syst., 4(4):369–383, 1990.

Semantic Overlay Networks

George Anadiotis, Spyros Kotoulas, Ronny Siebes
VU University Amsterdam, Amsterdam,
The Netherlands

Synonyms

Semantic overlays; SONs

Definition

Semantic Overlay Networks are types of *Overlay Networks* where the topology is formed according to the resources (i.e., services or data provided) of the participants. This is done on the basis of similarity between participants, which is calculated by means of resource metadata exchange (e.g. keywords, term vectors, concepts from ontologies, histograms).

Historical Background

Semantic Overlay Networks were introduced in 2002, by H. Garcia-Molina and A. Crespo [3]. The motivation was to give an alternative to inefficient search methods for overlay networks, one that would provide more relevant results in less time and using less resources (mainly bandwidth).

Foundations

The original idea of Semantic Overlay Networks [3] was to organize the nodes that participate in a network into many different overlays, based on the content that the nodes are contributing and some classification scheme for this content. As one overlay is created for each class, the classification to be used is rather important for the operation of the system. Having classifications whose distribution is skewed would result in overlays that are not efficient (either too big or too small to be of real value). Classification as used in [3] entails hierarchy, so it is in fact a taxonomy.

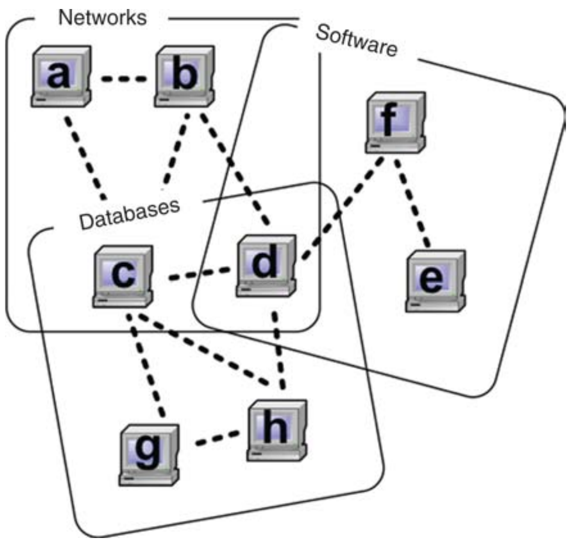
After choosing an appropriate taxonomy, content needs to be classified (either manually or automatically). This is performed massively for all content belonging to incoming nodes, so then the results of content classification are used to assign nodes to overlays, by choosing the one(s) that is the best match according to the majority of the content's classification. Different techniques and policies can be applied here (for example the predecessors and the descendants of a class are also considered), and nodes can be assigned to more than one overlay.

Finally, in order to be able to answer queries, queries are also subjected to the same classification. When they have been classified, they are subsequently forwarded only to the appropriate overlay. Results can be incremental, meaning that the query can also be sent to overlays that correspond to predecessors of the querys assigned class (i.e., more general classes) in order to retrieve additional matches.

In Fig. 1, one can see an example of three semantic overlay networks, as described above. Each node may belong to more than one semantic overlay (e.g., node d belongs to all semantic overlays) and has to maintain a number of connections for each one of them.

Inter and intra overlay routing (i.e., finding the appropriate overlay and routing within the overlay) presents a series of challenges which, along with other optimizations, have been elaborated by recent research.

In [6], the metadata used to extract similarity are (schema-less) XML documents extracted from peer



Semantic Overlay Networks. Figure 1. Three overlay networks.

content, and supported queries are extended from keyword to path queries that exploit the structure of XML documents. Peers maintain specialized data structures that summarize large collections of documents (filters – specialized extensions of Bloom filters). Each node maintains a local filter that summarizes the documents stored locally. Nodes are organized in hierarchies (trees) based on similarity of local filters; non-leaf nodes also contain merged filters summarizing the documents of its children, or in the case of root nodes, other root nodes as well.

With this organization, nodes belonging to the top levels receive more load and responsibilities, thus, the most stable and powerful nodes should be located to the top levels of the hierarchies. When receiving a query, nodes use their local filter to find results and then forward the query to their sub-tree.

In [12], the notion of *peer schemas* is defined. Peer schemas are virtually defined schemas that represent a peer's view of the world and are used for purposes of querying and mapping. Relations between peer schemas are called peer relations. Peers also contribute data to the system in the form of stored relationships, which correspond to the peer's local view of the data. Mappings are utilized in order to translate queries between different semantic networks. There are two types of mappings, namely mappings that relate two or more peer schemas (peer descriptions) as well as mappings that relate a stored schema to a peer schema (storage descriptions).

Sending queries only to peers that might provide answers is achieved using a two-fold approach: on the one hand, there is a query reformulation algorithm that works by combining global-as-view (*GAV*) and local-as-view (*LAV*) approaches and selectively applying unfolding and rewriting techniques to the original query. Since however the algorithm is only able to exploit information pertaining to schema mappings and not actual data stored at the peers, a (centralized) index structure that allows simple value lookup with partial match over structured attributes is also used. Participating peers upload data summaries as well as peer mappings to the index, thus enabling the index engine to correlate attributes from different peers and provide a simple type of schema mapping.

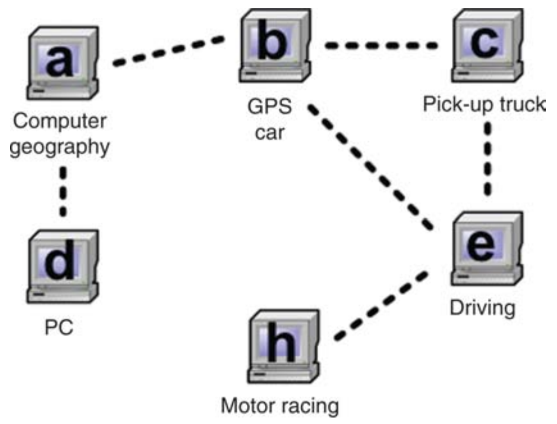
In [9], a variation on the approach presented in [6] is given. Instead of schema-less XML documents, metadata consist of RDF statements abiding to different RDFS schemas. There are normal peers (P) and super peers (SP) that normal peers attach to, as well as index structures (SP/P and SP/SP) that utilize frequency statistics used to define similarity measures responsible for clustering peers to super-peers, thus making them dynamic. Furthermore, an efficient topology is maintained for communication in the super peer network (HyperCup), which also offers mediation services between different peer schemas.

In [2], a layered architecture for SONs is proposed, comprising of a knowledge infrastructure layer and a communication infrastructure layer. Here fully blown ontologies are used instead of taxonomies or schemas. Each peer is able to store data as well as a peer ontology. When a peer sends a query Q over the H3 network, the request goes through a query processing module for rewriting in terms of the ontological description of target concept(s). The rewritten query is then forwarded to a semantic routing module that sends it only to peers that may provide results semantically related to the kind of concepts requested (semantic neighbors).

In order to choose the semantic neighbors, semantic routing exploits the services of a knowledge manager module to retrieve ontology location links to the peers whose contents are semantically related to the target concept(s) in the query. Location links are returned to the semantic routing module, which uses them for query routing. When a peer's routing module receives a query, it forwards it to the query processing manager, where it is analyzed and processed. If no matching concepts are found, the query is discarded and no reply is returned. Otherwise the query answer is composed and forwarded to the semantic routing module which sends back the reply to the requesting peer.

In some systems, SONs are not discretely clustered and are formed on per-peer basis. While in the original approach by [3] there is a separate SON for each concept, a category of systems use pair-wise peer interactions. They are used to construct a topology enriched with content information about every neighbor, and dynamically determine routing according to this information, instead of broadcasting on the entire SON.

In [5,10], the SON is built using *advertising*. Peers exchange descriptions of their content with their neighbors. Furthermore, they keep advertisements that are similar to theirs, and thus, a semantic topology is formed. An example of such a topology is given in Fig. 2. Peers keep neighborhood relations based on their content descriptions. Thus, locality is improved. Furthermore, each peer is aware of the description of the



Semantic Overlay Networks. Figure 2. Peer neighborhood relations.

content of its neighbors. Thus, it can forward queries to the peers which are more likely to have relevant content. Research in the context of [10] indicates that either maintaining neighbor relations according to description similarity or forwarding queries to the peers with description most similar to the query perform much better than flooding approaches.

It is important to note a category of systems that use both a semantic and a structured overlay. P-Search [11] uses Latent Semantic Indexing *LSI* to extract vector representations of documents. These vectors are mapped into a multi-dimensional space maintained by a Content Addressable Network (CAN), a type of structured overlay. Nodes within this overlay participate in an an additional semantic overlay used for content-directed search.

Gridvine [1] uses semantic overlays to store, query and make mappings between RDFS schemas. They are split into triples and are indexed by subject, predicate and object using the P-Grid structured overlay. Each peer may define and use its own RDFS schema, so naturally incompatibilities in terms of resource description/query interpretation may appear. To cope with such incompatibilities, the notion of schema translations is introduced. A schema translation is a mapping between two different schemas. Since RDFS does not support schema mapping, these translations are encoded using OWL.

This is a very powerful feature, as it enables the gradual forwarding of requests from the originating peer to peers for which no direct schema translation exists. This is achieved through a procedure called Semantic Gossiping, in which each peer that receives a query expressed in a certain schema examines available translation links and evaluates (by performing syntactic and semantic analysis) if and where it should forward this query. In addition, schema inheritance is also supported, which enables peers to not only define their own schemas, but also either directly reuse or extend existing schema hierarchies. Sets of peers that share the same schema are called semantic neighborhoods.

Finally, other state-of-the-art features in SONs include observing past queries submitted/answers received in order to judge semantic proximity [13], proactively acquiring semantic links through gossiping [14], and applying ant-colony heuristics to improve semantic routing [8].

Key Applications

Semantic overlay networks use an order of magnitude less messages than flooding overlays. They enable Internet-scale systems, by providing the infrastructure for efficient search over large numbers of hosts. Despite keen interest in the area by the scientific community, none of the aforementioned systems has been commercially deployed yet.

The OpenKnowledge project (http://www.openk.org.) works on a P2P system where web-services and workflows, annotated by keywords, can be shared. The system uses a SON to store and retrieve them efficiently and to find peers that execute the webservices.

Bibster [5] is a P2P application for sharing bibliographic items. These items are annotated by concepts from an ontology. The SON is used to route queries, which are also concepts from the same ontology, to the peers that semantically match the query.

Another proposed application of SONs is using them to cluster content providers on the WWW in order to facilitate a comprehensive distributed search engine [4].

Future Directions

An interesting problem concerns optimizing multi-attribute search. Simply joining the results of single attributes is often not scalable, especially when the single attributes individually result in many answers.

Another topic is to do efficient information retrieval by using SONs. Currently, most resource discovery systems do not rank the results according to the relevance: either they match or they do not match.

Current peer-to-peer discovery systems using SONs fail to solve privacy infringement issues and are actually more vulnerable than centralized approaches. This is because, due to the nature of SONs, peers "know" about the content of other peers, which may be undesirable.

Distributed reasoning over a P2P network is another interesting topic where SONs may be of help, for example to cluster consistent parts of knowledge. Especially in the Semantic Web area, there is a desire to have an efficient (RDF) triple storage where reasoning is done via shared or local schema's. Some first solutions like Unistore (http://www.p-grid.org/publications/applications.html.) and Gridvine [1] are based on storing the triples in *DHTs*, which leads to many messages because each triple leads at least to three *DHT* storage- or lookup messages.

Cross-references

- ▶ Data Semantics
- ▶ DHT
- ▶ GAV
- ▶ LAV
- ▶ LSI
- ▶ Peer-to-Peer Overlay

Recommended Reading

1. Aberer K., Cudré-Mauroux P., Hauswirth M., and Pelt T.V. GridVine: Building Internet-Scale Semantic Overlay Networks. In McIlraith et al. [7], pp. 107–121.
2. Castano S., Ferrara A., Montanelli S., Pagani E., and Rossi G. Ontology-addressable contents in p2p networks. In Proc. First Workshop on Semantics in Peer-to-Peer and Grid Computing, 2003.
3. Crespo A. and Garcia-Molina H. Semantic Overlay Networks for P2P Systems, Technical Report 2003–75, Stanford University, InfoLab, 2003.
4. Doulkeridis C., Nørvåg K., and Vazirgiannis M. DESENT: Decentralized and Distributed Semantic Overlay Generation in P2P Networks, IEEE Jour. Selected Areas in Commun., 25 (1):25–34, 2007.
5. Haase P., Broekstra J., Ehrig M., Menken M., Mika P., Olko M., Plechawski M., Pyszlak P., Schnizler B., Siebes R., Staab S., and Tempich C. Bibster - A Semantics-Based Bibliographic Peer-to-Peer System. In McIlraith et al. [7], pp. 122–136.
6. Koloniari G. and Pitoura E. Content-Based Routing of Path Queries in Peer-to-Peer Systems. In Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 29–47.
7. McIlraith S.A., Plexousakis D., and van Harmelen F. (eds.), The Semantic Web. In Proc. 3rd Int. Semantic Web Conf., 2004.
8. Michlmayr E., Pany A., and Kappel G. Using Taxonomies for Content-based Routing with Ants. In Proc. Workshop on Innovations in Web Infrastructure, 2006.
9. Nejdl W., Wolpers M., Siberski W., Schmitz C., Schlosser M., Brunkhorst I., and Löser A. Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks. In Proc. 12th Int. World Wide Web Conf., 2003.
10. Siebes R. and Kotoulas S. pRoute: Peer selection using shared term similarity matrices. Web Intelligence and Agent Systems, 5(1):89–107, 2007.
11. Tang C., Xu Z., and Dwarkadas S. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. Tech. rep., HP Labs, 2002.
12. Tatarinov I., Ives Z., Madhavan J., Halevy A., Suciu D., Dalvi N., Dong X.L. Kadiyska Y., Miklau G., and Mork P. The Piazza peer data management project. ACM SIGMOD Rec., 32(3):47–52, 2003.
13. Tempich C., Staab S., and Wranik A. Remindin': semantic query routing in peer-to-peer networks based on social metaphors. In Proc. 12th Int. World Wide Web Conf., 2004, pp. 640–649.
14. Voulgaris S., Kermarrec A.M., Massoulie L., and Van Steen M. Exploiting Semantic Proximity in Peer-to-peer Content Searching. In Proc. 10th Int. Workshop on Future Trends in Distributed Computing Systems, 2004.

Semantic Overlays

▶ Semantic Overlay Networks

Semantic Web

Grigoris Antoniou[1,2], Dimitris Plexousakis[1,2]
[1]Foundation for Research and Technology-Hellas (FORTH), Heraklion, Greece
[2]University of Crete, Heraklion, Greece

Definition

The central idea of the Semantic Web initiative is to enrich Web content by machine-processable semantics. The approach is based on the following ideas:

1. Use meta-data (data about data) as semantic annotations
2. Use ontologies to describe knowledge needed to understand collections of Web information. The semantic annotations are linked to such ontologies
3. Use logic-based techniques to process and query collections of meta-data and ontologies

In the current Semantic Web work, two main goals can be distinguished.

Interpretation 1: The Semantic Web as the Web of Data

In the first interpretation, the main aim of the Semantic Web is to enable the integration of structured and semi-structured data sources over the Web. The main recipe is to expose data-sets on the Web enriched with semantic annotations, to use ontologies to express the intended semantics of these data-sets, in order to enable the integration and unexpected re-use of these data.

A typical use case for this version of the Semantic Web is the combination of geo-data with a set of consumer ratings for restaurants in order to provide an enriched information source.

Interpretation 2: The Semantic Web as an enrichment of the current Web

In the second interpretation, the aim of the Semantic Web is to improve the current World Wide Web. Typical use cases here are improved search engines, dynamic personalization of Web sites, and semantic enrichment of existing Web pages.

The source of the required semantic meta-data in this version of the Semantic Web is mostly claimed to come from automatic sources: concept extraction, named-entity recognition, automatic classification, etc. More recently, the insight is gaining ground that the required semantic markup can also be produced by social mechanisms of communities that provide large-scale human-produced markup.

Historical Background

The Semantic Web sprang as a vision approximately 10 years after the birth of the World-Wide Web. Not surprisingly, it was the inventor of the WWW that shaped the vision of the Semantic Web, which in turn gave rise to the entire research field.

Up until that stage, the Web was (and still is to a great extent) purely about syntax, a specific syntax geared towards homogenizing the way in which information is presented to human users via a browser. As revolutionary as the concept may have been, it was making content available only for human consumption as the interpretation of the content relied on implicit semantics. In other words, meaningful representation of content was not possible in HTML or its precursor SGML. Information and its presentation were mixed in the form of HTML documents, many of which generated automatically by applications. The Web made it easy to fetch any Web page from any server, on any platform through a uniform interface. HTML has many benefits: it is simple, textual, portable, easily searchable by keyword-based search engines and connects pieces of information together through hypertext links. The browser is the universal application. If written properly, normal HTML markup may reflect document presentation, but it cannot adequately represent the semantics & structure of data. Newer applications require more than the publishing of HTML documents; data must be made available on the Web for use by Web-enabled applications.

XML was the incarnation of the paradigm shift on the Web: a new standard that could be easily generated and consumed by applications, facilitating data exchange across platforms and organizations, transforming the Web from a collection of documents to a collection of data published as documents. XML gained popularity very fast. It resembles HTML in that it is easy to read and learn, it is universal, portable and at the same time extensible and more flexible than HTML. However, XML cannot address all interoperability requirements as it only provides the means for solving syntactic heterogeneity problems. The challenge is to address the inherent structural but foremost semantic heterogeneities that are encountered on the Web. Modern applications need more than data on the Web; they need semantics on the Web. Applications themselves evolve into services on the Web that may exploit semantics.

The main motivation behind the Semantic Web (or Web of meaning) vision is to make vast amounts of information resources (data, documents, programs) available along with various kinds of descriptive information, i.e., metadata. Better knowledge about the meaning, usage, accessibility or quality of web resources considerably facilitates automated processing of available Web content/services especially when metadata are described in a form that is precise, human-readable and machine-interpretable. The Semantic Web enables syntactic and semantic/structural interoperability among independently-developed Web applications, allowing them to efficiently perform sophisticated tasks for humans. At the same time, it enables Web resources (data & applications) to be accessible by their meaning rather than by keywords and syntactic forms.

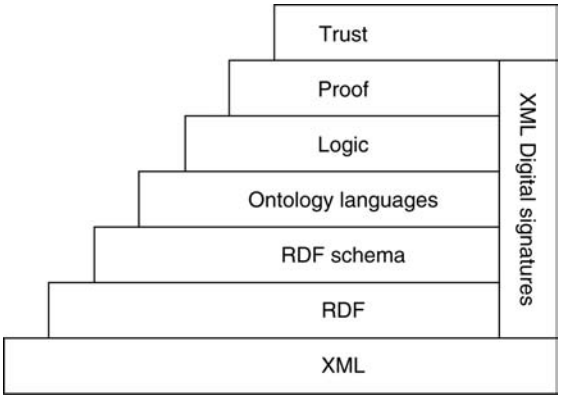
Foundations

The Semantic Web approach is based on the use of *semantic annotations* to describe the meaning of certain parts of Web information. For example, the Web site of a hotel could be suitably annotated to distinguish between hotel name, location, category, number of rooms, available services etc. Such metadata can facilitate the automated processing of the information on the Web site, thus making it accessible to machines.

However, the question arises as to how the semantic annotations of different Web sites can be combined, if everyone uses terminologies of their own. The solution lies in the organization of vocabularies in so-called *ontologies*. Recommended Reading to such shared vocabularies allow interoperability between different Web resources and applications. For example, an ontology of hotel classifications in a given country could be used to relate the rating of certain hotels. And a geographic ontology could be used to determine that Crete is a Greek island and Heraklion a city on Crete. Such information would be crucial to establish a connection between a requester looking for accommodation on a Greek island, and a hotel advertisement specifying Heraklion as the hotel location.

The development of the Semantic Web proceeds in steps, each step building a layer on top of another. The basic layered design is shown in Fig. 1, which is outlined below.

1. The bottom layer comprises *XML*, a language that lets one write structured Web documents with a user-defined vocabulary. XML is particularly suitable for sending documents across the Web, thus supporting syntactic interoperability
2. *RDF* (Resource Description Framework) is a basic data model, like the entity-relationship model, for writing simple statements about Web objects (resources). The RDF data model does not rely on XML, but RDF has an XML-based syntax. Therefore it is located on top of the XML layer
3. *RDF Schema* provides modeling primitives, for organizing Web objects into hierarchies. RDF Schema is based on RDF. RDF Schema can be viewed as a primitive language for writing ontologies
4. But there is a need for more powerful *ontology languages* that expand RDF Schema and allow the representations of more complex relationships between Web objects. Ontology languages, such as OWL (Ontology Web Language), are built on the top of RDF and RDF Schema
5. The *logic layer* is used to enhance the ontology language further and to allow writing application-specific declarative knowledge. Rule languages are the most popular logical languages used in Semantic Web applications
6. The *proof layer* involves the actual deductive process, as well as the representation and exchange of proofs in Web languages, for purposes such as explanation provision and proof validation
7. Finally, *trust* will emerge through the use of digital signatures, and other kind of knowledge, based on recommendations by agents that can be trusted, or rating and certification agencies and consumer bodies



Semantic Web. Figure 1. The semantic web tower.

RDF Basic Features

The language of RDF allows one to write statements. A statement consists of three parts (subject, predicate, object) and is often referred to as a triple. A triple of the form (x, P, y) corresponds to the logical formula P(x, y), where the binary predicate P relates the object x to the object y; this representation is used for translating RDF statements into a logical language ready to be processed automatically in conjunction with rules.

There are other ways to describe an RDF document, using a graphical and an XML representation.

RDF Schema Basic Features

In RDF, Web resources are individual objects. In RDFS, objects sharing similar characteristics are put together to form *classes*. Examples for classes are hotels, airlines, employees, rooms, excursions etc. Individuals belonging to a class are often referred to as *instances* of that class. For example, John Smith could be an instance of the class of employees of a particular hotel.

Binary *properties* are used to establish connections between classes. For example, a property *works_for* establishes a connection between employees and companies. Properties apply to individual objects (instances of the classes involved) to form RDF statements, as seen above.

The application of predicates can be restricted through the use of *domain and range restrictions*. For example, the property *works_for* can be restricted to apply only to employees (domain restriction), and to have as value only companies (range restriction).

Classes can be put together in hierarchies through the *subclass relationship*: a class C is a subclass of a class D if every instance of C is also an instance of D. For example, the class of island destinations is a subclass of all destinations: every instance of an island destination (e.g., Crete) is also a destination.

The hierarchical organization of classes is important due to the notion of *inheritance*: once a class C has been declared a subclass of D, every known instance of C is *automatically* classified also as instance of D. This has far-reaching implications for matching customer preferences to service offerings. For example, a customer may wish to make holidays on an Indonesian island. On the other hand, the hotel Noosa Beach advertises its location to be Bali. It is not necessary (nor is it realistic) for the hotel to add information that it is located in Indonesia and on an island; instead, this information is inferred by the ontology automatically.

Key Applications

This section provides a bird's eye survey of key application areas. It should be noted that a healthy uptake of Semantic Web technologies is beginning to take shape in the following areas:

1. Knowledge management, mostly in intranets of large corporations
2. Data integration (Boeing, Verison and others)
3. e-Science, in particular the life-sciences
4. Convergence with Semantic Grid

If one considers the profiles of companies active in this area, they will see a distinct transition from small start-up companies such as Aduna, Ontoprise, Network Inference, Top Quadrant (to name but a few) to large vendors such as IBM (their Snobase ontology Management System}, HP (with their popular Jena RDF platform), Adobe (with their RDF-based based XMP meta-data framework), and Oracle (now lending support for RDF storage and querying in their prime database product).

However, besides the application areas listed above, there is also a noticeable lack of uptake in some other areas. In particular, promises in the areas of

1. e-commerce
2. Personalization
3. Large-scale semantic search (on the scale of the World Wide Web, not limited to intra-nets),
4. Mobility and context-awareness

are largely unfulfilled, though there is significant ongoing activity in these directions.

A pattern that seems to emerge between the successful and unsuccessful application areas is that the successful areas are all aimed at *closed communities* (employees of large corporations, scientists in a particular area), while the applications aimed at the general public are still in the laboratory phase at best. The underlying reason for this could well be the difficulty of dealing with multiple ontologies and mappings among them.

Future Directions

At present, Semantic Web research focuses, among others, on:

1. Rule languages and their interaction or integration with ontology languages (RDF and OWL)
2. Scalable storage and retrieval systems
3. Knowledge and ontology evolution and change
4. Mapping mechanisms between different ontologies

A number of items on the research agenda are hardly tackled, but do have a crucial impact on the feasibility of the Semantic Web vision. In particular:

1. The mutual interaction between machine-processable representations and the dynamics of social networks of human users
2. Mechanisms to deal with trust, reputation, integrity and provenance in a (semi-) automated way
3. Inference and query facilities that are sufficiently robust to work in the face of limited resources (be it either computation time, network latency, memory or storage space), and that can make intelligent trade-off decisions between resource use and output-quality

Cross-references

▶ Interoperation of NLP-based Systems with Clinical Databases
▶ Ontology
▶ OWL: Web Ontology Language
▶ Resource Description Framework
▶ Resource Description Framework (RDF) Schema (RDFS)
▶ World Wide Web Consortium

Recommended Reading

1. Antoniou G. and van Harmelen F. A Semantic Web Primer (2nd ed.). MIT Press Cambridge, MA, 2008.
2. Staab S. and Studer R. (eds.). Handbook on Ontologies (2nd ed.). Springer, New York, 2008.
3. Berners-Lee T., Hendler J., and Lassila O. The Semantic Web. Sci. Am., 284 (May 2001): 34–43.
4. REASE. Available at: ubp.l3s.uni-hannover.de/ubp.
5. www.SemanticWeb.org.
6. www.w3.org/2001/sw/.
7. www.ontology.org.
8. The International Semantic Web Conference (http://iswc.semanticweb.org/).
9. Journal of Web Semantics (www.elsevier.com/locate/websem).

Semantic Web Query Languages

James Bailey[1], François Bry[2], Tim Furche[2], Sebastian Schaffert[3]

[1]University of Melbourne, Melbourne, VIC, Australia

[2]University of Munich, Munich, Germany

[3]Salzburg Research, Salzburg, Austria

Synonyms

Web query languages; Ontology query languages

Definition

A number of formalisms have been proposed for representing data and meta data on the Semantic Web. In particular, RDF, Topic Maps and OWL allow one to describe relationships between data items, such as concept hierarchies and relations between the concepts. A key requirement for the Semantic Web is integrated access to data represented in any of these formalisms, as well the ability to also access data in the formalisms of the "standard Web," such as (X)HTML and XML. This data access is the objective of *Semantic Web query languages*. A wide range of query languages for the Semantic Web exist, ranging from (i) pure "selection languages" with only limited expressivity, to fully-fledged reasoning languages, and (ii) from query languages restricted to a certain data representation format, such as XML or RDF, to general purpose languages that support multiple data representation formats and allow simultaneous querying of data on both the standard and Semantic Web.

Historical Background

The importance of Semantic Web query languages can be traced back to the roots of the Semantic Web itself. In its original conception, Tim Berners-Lee viewed the Semantic Web as allowing Web-based systems to take advantage of "intelligent" reasoning capabilities [4]:

> ► The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users .... For the Semantic Web to function, computers must have access to structured collections of information and sets of inference rules that they can use to conduct automated reasoning.

As the representation format for the Semantic Web has grown to cover XML, RDF, Topic Maps and OWL, there has been a corresponding growth in query languages that support access to each of these kinds of data.

Foundations

A number of techniques have been developed to facilitate powerful data retrieval on the Semantic Web. This article follows the classification and taxonomy given in [1], which provides a comprehensive survey of the area. Several categories of query languages can be distinguished, according to the format of the Semantic Web data they can retrieve:

1. Query languages for XML
2. Query languages for Topic Maps
3. Query languages for RDF
4. Query languages for OWL

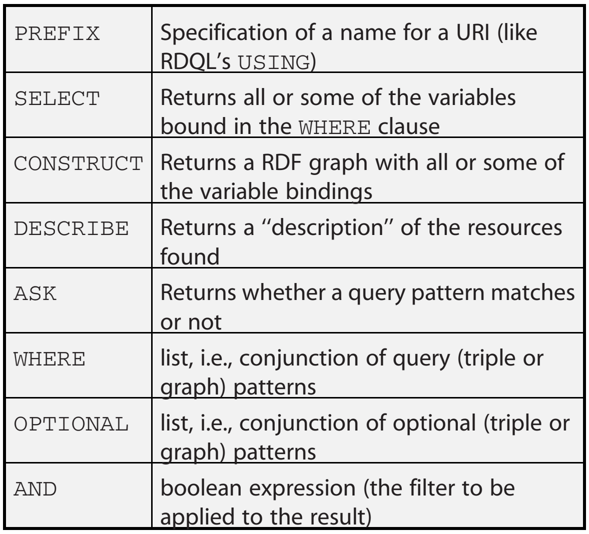
XML Query Languages: Although not a primary format, it is possible to specify information on the Semantic Web using XML. Hence query languages for XML are applicable to Semantic Web data. Most query and transformation languages for XML specify the structure of the data to retrieve using either of two approaches. In the navigational approach, path-based queries over the XML data are specified and the W3C standardized languages XPath, XSLT and XQuery are well known instances of this scheme. In the example based approach, query patterns are specified as "examples" of the XML data to be retrieved. Languages of this kind are mainly research languages, with some well known representatives being XML-QL [7] and Xcerpt [3,15].

Topic Maps Query Languages: Several different query languages for Topic Maps data exist, with representatives being tolog [9], AsTMA [2] and Toma [11].

tolog was selected as the initial straw man for the ISO Topic Maps Query Language and is inspired from logic programming, also having SQL style constructs. AsTMa is a functional query language, in the style of XQuery, whereas Toma combines both SQL syntax and path expressions for querying.

RDF Query Languages can be grouped into several families, that differ in aspects such as data model, expressivity, support for schema information, and type of queries. Principal among these families is the "SPARQL Family." This originated with the language SquishQL [12], which evolved into RDQL [12] and then was later extended to the language SPARQL [14]. These languages all "regard RDF as triple data without schema or ontology information unless explicitly included in the RDF source." SPARQL currently has W3C Candidate Recommendation status as being the "Query Language for RDF." In particular, SPARQL has facilities to:

1. Extract RDF subgraphs
2. Construct a new RDF graph using data from the input RDF graph queried
3. Return "descriptions" of the resources matching a query part
4. Specify optional triple or graph query patterns (i.e., data that should contribute to an answer if present in the data queried, but whose absence does not prevent an answer being returned 5. Test the absence, or non-existence, of tuples. The general format of a SPARQL query is:



PREFIX	Specification of a name for a URI (like RDQL's USING)
SELECT	Returns all or some of the variables bound in the WHERE clause
CONSTRUCT	Returns a RDF graph with all or some of the variable bindings
DESCRIBE	Returns a "description" of the resources found
ASK	Returns whether a query pattern matches or not
WHERE	list, i.e., conjunction of query (triple or graph) patterns
OPTIONAL	list, i.e., conjunction of optional (triple or graph) patterns
AND	boolean expression (the filter to be applied to the result)

Another family of languages for RDF, the "RQL family," consists of the language RQL [10], and its extensions such as SeRQL [5]. Common to this family is support for the combination of both data and schema querying. The RDF data model which is used slightly deviates from the standard data model for RDF and RDFS, disallowing cycles in the subsumption hierarchy and requiring both a domain and a range to be defined for each property. RQL itself has a large number of features and choices in syntactic constructs. This results in a complex, yet powerful language, which is far more expressive than other RDF query languages, especially those of the SPARQL family.

A number of other types of query languages for RDF also exist, using alternative paradigms. These include query languages using reactive rules, such as Algae [13] and deductive languages such as TRIPLE [6] and Xcerpt [15,3]. The last of these is noteworthy, as it combines querying on both the Standard Web (HTML/XML), with querying on the Semantic Web (e.g. RDF, TopicMaps) and also allows pattern-based, incomplete specification of queries.

OWL Query Languages: Query languages for OWL are still in their infancy compared to those for RDF. OWL-QL [8] is a well known language for querying OWL data and is an updated version of the DAML Query language. Its design targets the assistance of query-answering dialogues between computational agents on the Semantic Web. Unlike the RDF query languages, it focuses on the querying of schema rather than instance data. An RDF language such as SPARQL may of course be used to query OWL data, but it is not well suited to the task, since it is not designed to be aware of OWL semantics.

Several themes emerge from considering the design of the various Semantic Web Query languages [1].

- *Choice of querying paradigm*: Semantic Web query languages express basic queries using either the path based (navigational) or logic based (positional) paradigm.
- *Choice of variable type*: When Semantic Web query languages have variables, they almost always are logical variables, as opposed to variables in imperative programming languages.
- *Provision of Referential Transparency and Answer-Closure*. Referential Transparency (i.e., within the same scope, an expression always means the same), a well known trait of declarative languages, is striven

for by Semantic Web query languages. Answer closedness is a property that allows answers to queries themselves to be used as input to queries and is a key design principle of the languages SPARQL and Xcerpt.

- *Degree of Incompleteness*: Many Semantic Web query languages offer a means for incomplete specifications of queries, a reflection of the semi-structured nature of data on the Semantic Web.
- *Reasoning Capabilities*. Interestingly, but not surprisingly, not all XML query languages have views, rules, or similar concepts allowing the specification of other forms of reasoning. Surprisingly, the same holds true of RDF query languages. Many authors of RDF query languages see deduction and reasoning to be a feature of an underlying RDF store offering materialization, i.e., completion of RDF data with derivable data prior to query evaluation. This is surprising, because one might expect many Semantic Web applications to access not only one RDF data store at one Web site, but instead many RDF data stores at different Web sites and to draw conclusions combining data from different stores.

Key Applications

Like classical query languages such as SQL, the first key application of Semantic Web query languages is the efficient and scalable access, classification, analysis and transformation of large collections of data in a Web format such as XML, RDF, OWL, or Topic Maps. Whereas classical query languages are most often used for accessing a single, centralized database, Semantic Web query languages need to be able to access also remote databases and data sources. This opens up new application scenarios, potentially utilizing any of the vast number of the data sources available on the Web.

For example, one might query researcher and publication information integrated over various sources, such as DBLP, Citeseer, IEEE and Cordis, combine that data with course and lecturer information from the Semantic Web School and then even further correlate it with the US census data. All these resources would be far too large to download individually and query locally, but they provide interfaces known as *endpoints*, that can be used to select the relevant portions via a Semantic Web query interface. Another example application is the W3C Amaya browser, which can be used to enrich Web pages visited by a user, with annotations contained in remote data sources. The annotations relevant to a given Web page are accessed by querying an annotation server using Algae [13], an RDF query language similar to SPARQL. In such scenarios, the ability of RDF (and to some extent, XML) to define the names and concepts used in a database, reason about them and to map them to names and concepts used in another database, is essential. This clearly separates the use of Semantic Web query languages from the use of classical query languages for centralized databases.

Increasingly, current Web applications (often referred to as Web 2.0 applications) contain a Javascript-based user interface which is separate from the data processed by the application itself. Thus, the user interface can be loaded once and data then requested from the origin server or other data sources on the Web as required. Web query languages for XML, RDF, JSON and Topic Maps are now becoming recognized as the ideal interfaces between the client user interfaces of Web 2.0 applications and data sources, since they can target just the data that is needed in the current state of the application. Web query languages allow flexible, but fine-grained access to the required data, rather than the coarse-grained access provided by other solutions.

Future Directions

Most RDF query languages are RDF-specific, and even specifically designed for one RDF serialization, which of course limits their applicability. It is to be hoped that in the future, there will be an evolution towards data format "versatile" languages, capable of easily accommodating XML, RDF, Topic Maps and OWL, without requiring "serialization consciousness" from the programmer.

The method of query evaluation in current Semantic Web query languages is either backtracking-free logic programming (as used by positional languages) or set-oriented functional query evaluation. It seems likely these two paradigms may converge in future Semantic Web query languages. Language engineering issues, such as abstract data types and static type checking, modules, polymorphism, and abstract machines, have not yet made their way into Semantic Web query languages, as they did not in database query languages. This situation opens avenues for promising research of great practical, as well as theoretical relevance.

Data Sets

There are a number of SPARQL endpoints that can be browsed on the Web. These provide RDF data which can be viewed and then queried using a SPARQL client:

- The 2000 US Census Data endpoint: http://www.rdfabout.com/demo/census/
- The Semantic Web School endpoint: http://sparql.semantic-web.at/
- A compilation of endpoints including DBLP, Citeseer, IEEE and Cordis: http://www.rkbexplorer.com/

A collection of concrete query language use cases for accessing RDF data can be found in the W3C RDF Use Case document at http://www.w3.org/TR/rdf-dawg-uc/. A use case collection is also included in [2].

URL to Code

The D2R Server is a utility for publishing relational databases on the Semantic Web and can be found at: http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2r-server/

Annotea is a project that aims to assist collaboration via shared semantic meta-data. The Annotea-Server with Amaya Browser and Algae QL can found at: http://www.w3.org/2001/Annotea/

Cross-references

▶ Ontology
▶ OWL: Web Ontology Language
▶ Resource Description Framework
▶ Resource Description Framework (RDF) Schema (RDFS)
▶ Semantic Web
▶ Topic Maps
▶ XML
▶ XPath/XQuery
▶ XSL/XSLT

Recommended Reading

1. Bailey J., Bry F., Furche T., and Schaffert S. Web and semantic Web query languages: a survey. In Reasoning Web, LNCS 3564, Springer, 2005, pp. 35–133.
2. Barta R. AsTMa 1.3 language specification. Technical report, Bond University, 2003.
3. Berger S., Bry F., Furche T., Linse B., and Schroeder A. Beyond XML and RDF: the versatile Web query language Xcerpt. In Proc. 15th Int. World Wide Web Conf., 2006, pp. 1053–1054.
4. Berners-Lee T., Hendler J., and Lassila O. The Semantic Web—a new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American, 2001, 29–37.
5. Broekstra J. and Kampman A. SeRQL: a second generation RDF query language. In Proc. SWAD-Europe Workshop on Semantic Web Storage and Retrieval, 2003.
6. Decker S., Sintek M., Billig A., Henze N., Dolog P., Nejdl W., Harth A., Leicher A., Busse S., Ambite J.L., Weathers M., Neumann G., and Zdun U. TRIPLE – an RDF rule language with context and use cases. In Proc. Rule Languages for Interoperability, 2005.
7. Deutsch A., Fernandez M., Florescu D., Levy A., and Suciu D. A query language for XML. Comput. Netw., 31(11–16):1155–1169, 1999.
8. Fikes R., Hayes P., and Horrocks I. OWL-QL – a language for deductive query answering on the semantic Web. J. Web Semant., 2(1):19–29, 2004.
9. Garshol L.M. Tolog – a topic maps query language. In Proc. 1st Int. Workshop on Topic Maps Research and Applications, 2005, pp. 183–196.
10. Karvounarakis G., Magkanaraki A., Alexaki S., Christophides V., Plexousakis D., Scholl M., and Tolle K. Querying the Semantic Web with RQL. Comput. Netw. ISDN Syst. J., 42(5):617–640, August 2003.
11. Lacher M. and Decker S. RDF, topic maps, and the semantic web. Markup Lang. Theory and Pract., 3(3):313–331, December 2001.
12. Miller L., Seaborne A., and Reggiori A. Three implementations of SquishQL, a simple RDF query language. In Proc. Int. Semantic Web Conf., 2002, pp. 423–435.
13. Prud'hommeaux E. Algae RDF Query Language. http://www.w3.org/2004/05/06-Algae/, 2004.
14. Prud'hommeaux E. and Seaborne A. SPARQL query language for RDF. Candidate recommendation, W3C, June 2007, http://www.w3.org/TR/rdf-sparql-query/.
15. Schaffert S. and Bry F. Querying the Web reconsidered: a practical introduction to Xcerpt. In Proc. Extreme Markup Languages, 2004.

Semantic Web Services

David Martin
SRI International, Menlo Park, CA, USA

Definition

Semantic Web Services (SWS) is a research area developing theory, technology, standards, tools, and infrastructure for working with distributed, networked services. As its name indicates, SWS has arisen from the cross-fertilization of challenges and approaches from the Web services and Semantic Web areas. The central theme of SWS is the enrichment of Web services technology with knowledge representation and reasoning technologies (including but not limited to those associated with the Semantic Web). The starting point for most SWS approaches is the use of expressive, declarative descriptions of the elements of dynamic distributed computation, with a particular focus on services, processes that are encapsulated by services, and message-based conversations between service providers and consumers. (Depending on the approach, other relevant concepts might include goals, transactions, roles, commitments, mediators of various kinds, etc.) These descriptions, in turn, are seen as the basis for fuller, more flexible automation of service provision and use, and the construction of more powerful components, architectures, tools and methodologies for working with services. In most cases, descriptions are expressed in a formal logical framework allowing for the use of well-understood reasoning procedures.

Many SWS researchers have articulated a broad and ambitious long-term vision of a Web where support for shared *activities* is as central as support for shared *information*. Many view SWS, developed to its full potential, as a technology foundation for distributed autonomous agents (and much SWS work draws on earlier work on agent-based systems). Another important theme in SWS is the development of a unified, comprehensive representation framework (often making use of ontologies) that can provide a foundation for a broad range of activities throughout the Web service lifecycle, including design and development, publication in registries, discovery and selection, negotiation and contracting, composition of services, monitoring and recovery from failure, and so forth.

Key Points

SWS research, as a distinct field, began in earnest in 2001. In that year, the initial release of OWL for Services (OWL-S) [5] became available. Other major initiatives began not long thereafter, including the Web Services Modeling Ontology (WSMO) [4], the Semantic Web Services Framework (SWSF) [2], WSDL-S [1], and the Internet Reasoning Service [3]. Many individual researchers and small teams have also done much valuable work, sometimes drawing on one of these larger efforts, sometimes not.

A fair amount of work in SWS has been focused on two central problems. Given a service request and a collection of service descriptions, *service discovery* is the problem of identifying those services that can satisfy the request, and possibly ranking them according to some measure of suitability. Given a goal to be satisfied and a collection of service descriptions, *service composition* is the problem of finding a procedure composed of service invocations that will achieve that goal. It should be emphasized, however, that SWS is a broad field with many challenging problems, of which these two are mentioned as illustrations.

Important application areas for SWS have included business (e.g., automated or partially automated discovery and use of needed services, enactment and composition of business processes and workflow, supply chain management, contracting, formation of virtual organizations, etc.), e-Government, and e-Science.

A few SWS standards activities have occurred. For example, the World Wide Web Consortium (W3C) has published a set of extensions to the Web Services Description Language (WSDL), known as Semantic Annotations for WSDL (SAWSDL), which makes it possible to associate elements of WSDL specifications with elements defined in a SWS framework (not defined by SAWSDL). W3C also hosts workshops and study groups to consider the suitability of various aspects of SWS for standardization.

Cross-references

▶ Semantic Web
▶ Web Services

Recommended Reading

1. Akkiraju R., Farrell J., and Miller J, et al. Web Service Semantics – WSDL-S, vol. 1.0, tech. note, Apr. 2005.
2. Battle S, Bernstein A, and Boley H, et al. Semantic Web Services Framework (SWSF) Overview, 2005.
3. Cabral L., Domingue J., and Galizia S, et al. IRS-III: a broker for semantic web services based applications. In Proc. 5th International Semantic Web Conference, 2006, pp. 201–214.
4. Fensel D., Lausen H., and Polleres A, et al. Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer, New York, 2006.
5. Martin D., Burstein M., and McDermott D, et al. Bringing semantics to web services with OWL-S. World Wide Web J., 10(3): 243–277, 2007.

Semantic-based Retrieval

▶ Multimedia Information Retrieval Model

Semantic Atomicity

Greg Speegle
Baylor University, Waco, TX, USA

Definition

Let T be a transaction composed of subtransactions S_0, $S_1,\ldots,S_{n-1}$. Let C_0, $C_1,\ldots,C_{n-1}$ be a set of *compensating transactions*, such that C_i compensates for the corresponding S_i. T is *semantically atomic* iff all S_i have committed, or for all S_i that have committed, C_i has also committed. A schedule (or history) ensures semantic atomicity if all transactions are semantically atomic. If T requires compensating transactions, then the resulting database is semantically equivalent to one in which T did not execute at all, but it is not guaranteed to be identical. Typically, two database states are equivalent if they both satisfy all of the database constraints.

Historical Background

Semantic Atomicity is first defined in [6], with the use of *countersteps* to remove parts of a failed transaction executing in a distributed database environment, without rolling back the entire transaction. The "step" grew in complexity to a subtransaction with the introduction of Sagas [7]. This required a corresponding increase in the complexity of the counter-measure, now called *compensating transactions*. Within Sagas, subtransactions are allowed to interleave with other transactions, and an execution is correct if every subtransaction commits, or the corresponding compensating transaction is executed. In [10], transactions are extended to transaction programs with input and output constraints, thus allowing formal representations of the capabilities and requirements for compensating transactions. Semantic atomicity has been applied in multidatabases[4], multilevel secure databases [2], workflows [3], and real-time database systems [16].

Foundations

Under traditional database correctness criteria, either all of the updates performed by a transaction must be committed to the database, or none of them should be. This is the *atomicity* requirement in ACID (atomicity, consistency, isolation and durability), the standard correctness criteria for database transactions. Atomicity has potentially far-reaching consequences. Consider the scenario where transaction T_1 updates some data, and transaction T_2 reads the updated, but not committed, value. If T_1 fails, then the update must be removed from the database, which implies T_2 cannot have read that value, and thus, T_2 must be aborted as well. This situation is called *cascading aborts*.

Within traditional database applications, atomicity (and consequently cascading aborts) is the correct criterion. Transactions in this environment are very short (only a few operations) and access only a few data items. In fact, the traditional database environment avoids cascading aborts by using a protocol called strict two-phase locking [8], which forces T_2 to wait until T_1 commits before it can read anything written by T_1.

The success of database management systems and the explosion of electronic data has pushed DBMSs (database management systems) into non-traditional applications. In these applications, transactions can be very long and very complex (e.g., design applications [9] and workflows [3]). In these applications, cascading aborts are unacceptable, but it is also unacceptable to force long-duration waits with two-phase locking. There are many efforts in the literature to solve this problem by exploiting the semantic information of the transactions (see e.g., [1,5,7,11,12,17]).

These solutions focus on relaxing one or more of the ACID properties while still ensuring the correct execution of the DBMS. This relaxation can only occur by using additional information (semantics) not exploited in traditional database systems. *Semantic atomicity* focuses on relaxing the atomicity requirement by using additional information in the form of *compensating transactions* for any transaction that is executing in the system [7]. A compensating transaction uses application specific information to restore the database consistency constraint for any failed transaction.

Additionally, semantic atomicity is often used in environments where transactions are nested [14], which means transactions are divided into subtransactions. One way to prevent cascading aborts is to prohibit the exposure of the results of a subtransaction beyond the transaction itself. However, with semantic atomicity, these intermediate results can be visible to other transactions and cascading aborts can still be avoided. This is one of the benefits of the long duration transaction model called *Sagas* [7].

Semantic atomicity allows transactions to read dirty data if all failed transactions meet the following compensation requirement:

Definition: Let T be a transaction, H be the set of all transactions concurrently executing with T (excluding T), and let C be the compensating transaction for T. Let D represent the database state resulting from executing THC on the database, and D' be the database state resulting from executing H alone. A transaction has been compensated if $D \equiv D'$.

In general, two database states are equivalent if they both satisfy all database consistency constraints. Likewise, it must be the case that the database state seen by H is consistent, otherwise the execution of some of the transactions in H is unpredictable. It is also possible for subtransactions to have reduced consistency requirements, both in terms of the state of the database before transactions execute, and in the database state after the execution is finished [10]. In these cases, the compensating transaction does not have to completely restore the database state, but it must create a database state that allows the other subtransactions to continue.

An example can clarify the benefits of semantic atomicity. Consider a business process in which a company manufactures widgets. Some of the widgets are sold after they are manufactured, and others are ordered in advance. The database consistency constraint is that the number of widgets in production must be at least twice the number of widgets that have been pre-ordered. Let P and O be data items in the database such that P is the number of widgets in production, and O is the number of ordered widgets. The constraint is that $P \geq 2 \times O$. Furthermore, assume a purchase transaction, T, is composed of two parts – order placement (S_0) and payment (S_1). Finally, assume the current database state has $O = 10$ and $P = 23$.

An order comes in for 3 additional widgets. Since the company wants to produce the needed widgets as soon as possible, the database is updated right away by having transaction S_0 set O to 13, and correspondingly, increasing P to 26. Under semantic atomicity, S_0 is free to commit. As soon as P is increased, a workflow transaction W begins the manufacturing process for three additional widgets.

Now the customer decides the price for the three widgets is too high, and cancels the order. As a result, T has failed. Under traditional atomicity, the update by T would rollback, and therefore the execution of W would also abort. Note that aborting W may not even be possible, depending on the state of widget production. Under semantic atomicity, a compensating transaction C_O is executed instead. C_O performs a rollback on the value of O, but leaves P unchanged. Therefore, W can continue execution. Thus, consistency is restored and the schedule is semantic atomic, even though W would *never execute* without T. The database states are equivalent (both are consistent), but not identical.

One great but not immediately obvious benefit of semantic atomicity is that subtransactions can commit as soon as possible, thus externalizing the effects of the subtransactions right away. This is because a later failure cannot cause a cascading abort. The appropriate compensating transaction is executed, and the transaction system continues forward. As a result, long duration transaction systems do not have to impose long duration waits to avoid cascading aborts. Thus, the two primary disadvantages for ensuring ACID with advanced transactions (long waits and large amounts of lost work) are prevented.

Unfortunately, compensating transactions cannot always be applied automatically. Consider the example from before, but assume the constraint is $2 \times O \leq P \leq 2.5 \times O$. The compensation for S_0 cannot simply rollback O, as the resulting state would be inconsistent. One possible solution would require the compensation of W, perhaps removing a widget from production. Whether or not this is possible would depend on the application semantics. Alternatively, S_0 cannot be compensated, and thus semantic atomicity reduces to traditional atomicity. In this case, T would not commit until payment is received, and the extra widgets would not begin production until the order is committed.

The complexity of using application semantics in building compensating transactions prevents the common deployment of semantic atomicity. In [10] these problems are studied in detail, with the creation of *operations* which are arbitrarily complex modifications to a single database entity. These operations are allowed local variables, thereby resembling functions in traditional programming languages. The operations are combined into *transaction programs*, which include conditional statements and statement blocks. Within this model, several aspects of compensating transactions are explored, such as compensation when the database states must be identical (not just equivalent) and compensation when some transaction program must follow the compensated-for transaction (called unsound).

Another key issue mentioned in [10] is the requirement that compensating transactions do not fail. Although this can be ensured during normal database operations (e.g., using a deadlock avoidance mechanism for compensating transactions), system failures cannot be prevented. The solution to this problem requires logging the internal state of the compensating transaction as well as any database modifications. During recovery, incomplete compensating transactions are not aborted, but are continued from the saved internal state, similar to the notion of compensating log records in ARIES [13].

Note that semantic atomicity is distinctly different from the concept of a *savepoint*. A savepoint does not expose the updates of a transaction to outside processes (called externalized operations in [10]), while semantic atomicity supports this. Likewise, when a traditional database transaction performs a rollback to a savepoint, any other transaction which has read the aborted updates, must also abort. Thus, savepoints do not prevent cascading aborts.

Key Applications

Semantic atomicity is appropriate for any application where the benefit of avoiding cascading aborts without long-duration waits is greater than the difficulty of creating the compensating transactions. Examples include:

1. Multidatabases [4] where each site can commit a subtransaction without the over head of two-phase commit
2. Application services such as the Microsoft Phoenix project [3] which Support applications surviving database failures in part by using semantic information
3. Workflows [3] where processes need to begin as soon as possible, and often cannot be aborted
4. Web services [15] where actions are performed by loosely coupled systems
5. Real-time systems [16] where the ability to predict transaction length is greatly improved by avoiding cascading aborts and by allowing early commits
6. Secure databases [2] where cascading aborts can cause covert information exchange

Although not an application per se, support for compensating transactions, and thereby semantic atomicity, has been included in the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Business Activity (WS-Business-Activity) standard released in July 2007. Certainly, this will increase the number of commercial applications using semantic atomicity.

Cross-references

- ▶ ACID
- ▶ Atomicity
- ▶ Compensating Transactions
- ▶ Extended Transaction Models and ACTA
- ▶ Nested Transactions
- ▶ Open-Nested Transaction Model
- ▶ Sagas
- ▶ Workflows

Recommended Reading

1. Ahmed E. (ed.). Database transaction models for advanced applications. Data Management Systems. Morgan Kaufmann, Los Altos, CA, 1992.
2. Ammann P., Jajodia S., and Ray I. Ensuring atomicity of multilevel transactions. In Proc. IEEE Symp. on Research in Security and Privacy, 1996, pp. 74–84.
3. Breitbart Y., Deacon A., Schek H.-J., Sheth A., and Weikum G. Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows. ACM SIGMOD Rec., 22(3):23–30, 1993.
4. Breitbart Y., Garcia-Molina H., and Silberschatz A. Overview of multidatabase transaction management. VLDB J., 1(2):181–240, 1992.
5. Chrysanthis P.K. and Ramamritham K. Synthesis of extended transaction models using ACTA. ACM Trans. Database Syst., 19(3):450–491, 1994.
6. Garcia-Molina H. Using semantic knowledge for transaction processing in a distributed database. ACM Trans. Database Syst., 8(2):186–213, June 1983.
7. Garcia-Molina H. and Salem K. Sagas. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 249–259.
8. Gray J., Lorie R., Putzolu G., and Traiger I. Granularity of locks and degrees of consistency in a shared database. In Readings in Database Systems, Morgan Kaufmann, 1998, pp. 94–121.
9. Korth H.F., Kim W., and Bancilhon F. On long duration CAD transactions. Inf. Sci., 46:73–107, October 1988.
10. Korth H.F., Levy E., and Silberschatz A. A formal approach of recovery by compensating transactions. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 95–106.
11. Korth H.F. and Speegle G. Formal aspects of concurrency control in long-duration transaction systems using the NT/PV model. ACM Trans. Database Syst., 19(3):492–535, 1994.
12. Lynch N., Merritt M., Wiehl W., and Fekete A. Atomic transactions. Data Management Systems. Morgan Kaufmann, 1994.
13. Mohan C., Haderle D., Lindsay B., Pirahesh H., and Schwarz P. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. Database Syst., 17:94–162, March 1992.
14. Moss J.E.B. Nested Transactions – An Approach to Reliable Distributed Computing. MIT Press, Cambridge, MA, 1985.

15. Puustjarvi J. Using advanced transaction and workflow models in composing web services. In Adv. Comput. Sci. Technol., 2007.
16. Soparkar N., Levy E., Korth H.F., and Silberschatz A. Adaptive commitment for distributed real-time transactions. In Proc. Int. Conf. on Information and Knowledge Management, 1994, pp. 187–194.
17. Weikum G. Principles and realization strategies of multilevel transaction management. ACM Trans. Database Syst., 16(1): 132–180, 1991.

Semantics-based Concurrency Control

Krithi Ramamritham[1], Panos K. Chrysanthis[2]
[1]Indian Institute of Technology Bombay, Mumbai, India
[2]University of Pittsburgh, Pittsburgh, PA, USA

Definition

Specifications of data contain semantic information that can be exploited to increase concurrency. For example, two insert operations on a multiset object commute and hence, can be executed in parallel; further, regardless of whether one operation commits, the other can still commit. Applying the same rule, two push operations on a stack object do not commute and hence cannot be executed concurrently. Several schemes have been proposed for exploiting the semantics of operations have to provide more concurrency than obtained by the conventional classification of operations as *reads* or *writes*.

Key Points

In most semantics-based protocols, conflicts between operations is based on commutativity, an operation o_i which does not commute with other uncommitted operations will be made to wait until these conflicting operations abort or commit. Some protocols use operations' return value commutativity, wherein information about the results of executing an operation is used in determining commutativity, and some use the arguments of the operations in determining whether or not two operations commute. An example of the former, two increment operations on a counter object commute as long as they do not return the new or old value of the counter. An example of the latter, two insert operations on a set object commute as long as they do not insert the same item.

In the scheme reported in [1], non-commuting but *recoverable* operations are allowed to execute in parallel; but the order in which the transactions invoking the operations should commit is fixed to be the order in which they are invoked. If o_j is executed after o_i, and o_j is *recoverable relative to* o_i, then, if transactions T_i and T_j that invoked o_i and o_j respectively commit, T_i should commit before T_j. Thus, based on the recoverability relationship of an operation with other operations, a transaction invoking the operation sets up a dynamic commit dependency relation between itself and other transactions. If an invoked operation is not recoverable with respect to an uncommitted operation, then the invoking transaction is made to wait. For example, two pushes on a stack do not commute, but if the push operations are forced to commit in the order they were invoked, then the execution of the two push operations is serializable in commit order. Further, if either of the transactions aborts the other can still commit.

In [2] authors make an effort to discover, from first principles, the nature of concurrency semantics inherent in objects. Towards this end, they identify the dimensions along which object and operation semantics can be modeled. These dimensions are then used to classify and unify existing semantic-based concurrency control schemes. To formalize this classification, a graph representation for objects that can be derived from the abstract specification of an object is proposed. Based on this representation, which helps to identify the semantic information inherent in an object, a methodology is presented that shows how various semantic notions applicable to concurrency control can be effectively combined to improve concurrency. A new source of semantic information, namely, the ordering among component objects, is exploited to further enhance concurrency. Lastly, the authors present a scheme, based on this methodology, for *deriving* compatibility tables for operations on objects.

Cross-references

► ACID Properties
► Concurrency Control – Traditional Approaches

Recommended Reading

1. Badrinath B.R. and Ramamritham K. Semantics-based concurrency control: beyond commutativity. ACM Trans. Database Syst., 17(1):163–199, 1991.
2. Chrysanthis P.K., Raghuram S., and Ramamritham K. Extracting concurrency from objects: a methodology. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991.

Semijoin

Kai-Uwe Sattler
Technical University of Ilmenau, Ilmenau, Germany

Synonyms

Semijoin filter; Hash filter join; Bit vector join; Bloom filter join; Bloom join

Definition

Semijoin is a technique for processing a join between two tables that are stored sites. The basic idea is to reduce the transfer cost by first sending only the projected join column(s) to the other site, where it is joined with the second relation. Then, all matching tuples from the second relation are sent back to the first site to compute the final join result.

Historical Background

The semijoin technique was originally developed by Bernstein et al. [3] as part of the SDD-1 project as a reduction operator for distributed query processing. The idea of applying hash filtering was proposed by Babb [1] as well as by Valduriez [9] particularly for specialized hardware (content addressed file stores and distributed database machines respectively). The theory of semijoin-based distributed query processing was presented in [2]. In [10] semijoins are also exploited for query processing on multiprocessor database machines. Results of detailed experimental work on semijoins in distributed databases were first reported by Lu and Carey [6] as well as by Mackert and Lohman [7].

Foundations

Semijoin is a join processing technique which was originally developed for distributed databases. A semijoin is the "half of a join" and is particularly useful as a reduction operator.

Relational Definition

Given two relations $R(A,B)$ and $S(C,D)$ with the join condition $R.A = S.C$ the semijoin $R \ltimes S$ is defined as follows:

$$R \ltimes_{A=C} S = \pi_{attr(R)}(R \bowtie_{A=C} S)$$

where $attr(R)$ denotes the set of attributes in R. The semijoin has two important characteristics:

1. It is a reducing operator, because $R \ltimes_{A=C} S \subseteq R$.
2. It is asymmetric, i.e., $R \ltimes_{A=C} S \neq S \ltimes_{A=C} R$.

Semijoin Filtering

The obvious approach of processing a join between a relation R stored at site 1 and S stored at site 2 is to ship the smaller relation to the other site and compute the join locally. This is also called "ship whole" approach. However, for computing the join one or both of relations can be replaced by a semijoin with the other relation, i.e.,:

$$\begin{aligned} R \bowtie_{A=B} S &= (R \ltimes_{A=C} S) \bowtie_{A=C} S \\ &= R \bowtie_{A=C} (S \ltimes_{C=A} R) \\ &= (R \ltimes_{A=C} S) \bowtie_{A=C} (S \ltimes_{C=A} R) \end{aligned}$$

In each case the semijoin acts as a reducer operation just like a selection operator. Which variant is chosen for the actual join processing has to decided by estimating the costs.

The principled approach of the semijoin filtering can be formulated in the following algorithm:

1. At site 1 compute $R' := \pi_A(R)$ and send it to site 2
2. At site 2 process the semijoin $S' := S \ltimes_{C=A} R'$ Note, relation S' contains only tuples matching the join condition and will appear in the final result. Furthermore, the result relation provides only the attributes from S
3. Send relation S' to site 1
4. At site 1, $R \bowtie_{A=C} S'$ is computed producing a result equivalent to $R \bowtie_{A=C} S$

In Fig. 1 the process is illustrated using an example.

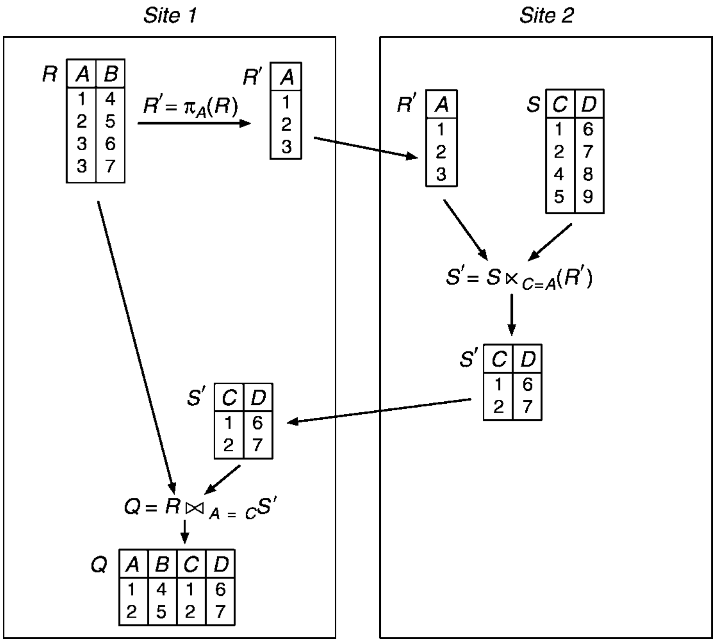
In order to estimate the benefit of the semijoin compared to the "ship whole" approach it is sufficient to consider only the transfer costs. Let C denote the cost for transfer a data unit and $size(R) = |R| \cdot width(R)$ the size of the relation derived from the cardinality $|R|$ and the size $width(R)$ of a tuple in data units. Then, the cost for the ship whole strategy is

$$C \cdot size(S)$$

assuming S is the smaller relation ($size(S) < size(R)$). For the semijoin the main costs are in step 1 and step 3, i.e.,

$$C \cdot size(\pi_A(R)) + C \cdot size(S \ltimes_{C=A} R)$$

Comparing these costs, one can observe that the semijoin approach is better if



Semijoin. Figure 1. Example of semijoin processing.

$$size(\pi_A(R)) + size(S \ltimes_{C=A} R) < size(S)$$

More exactly due to $width(S \ltimes_{C=A} R) = width(S)$ the semijoin is the better choice if $|S \ltimes_{C=A} R| < |S|$, i.e., if the semijoin is really a reducer. At the other hand, the ship whole approach is better if nearly all tuples of S contribute to the join result. In this case, the semijoin has the disadvantage of the additional transfer of $\pi_A(R)$.

Thus, a decision for one of these join strategies requires an estimation of the join selectivity factor SF. For the semijoin the following approximation

$$SF_{R \ltimes_{A=C} S} = \frac{|\pi_C(S)|}{|distinct(C)|}$$

was proposed by [5], where $|distinct(C)|$ denotes the number of distinct values in attribute C.

Bit Vector Filtering

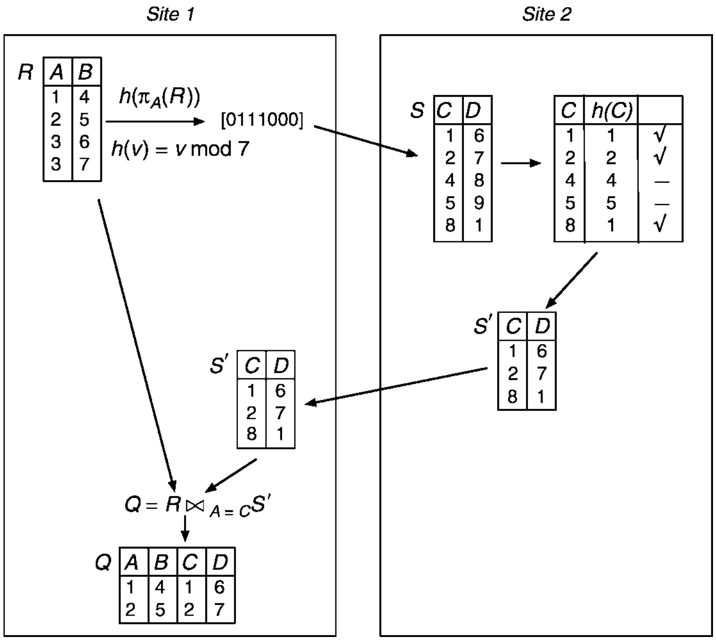
The effort for step 1 of the semijoin can be further reduced by sending only a compact bitmap representation of the column values instead of $\pi_A(R)$. This bitmap or bit vector is built using a hash function [4] and, thus, the approach is called bit vector filtering, hash filter join or bloom join.

For a hash function $h(v)$ returning values $0...n$ a bit vector $B[0...n]$ containing $n+1$ bits initially set to 0 is required. For each value $v \in \pi_A(R)$ the corresponding bit $B[h(v)]$ is set to 1. Instead of processing the semijoin $S \ltimes R$ at site 2, this bit vector and the hash function are used to probe the tuples of S for matching with the join values of R, i.e., if for a value v' of the join attribute B the corresponding bit is set: $B[h(v')] = 1$. The whole process is shown in the following algorithm:

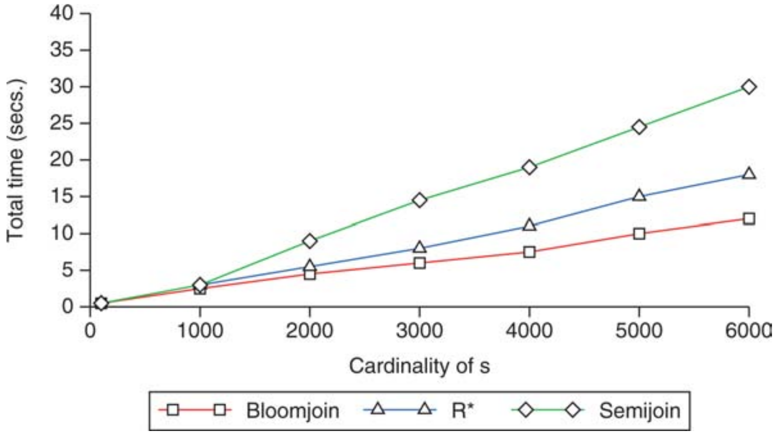
1. At site 1: for each $v \in \pi_A(R)$ set $B[h(v)] = 1$ and sent B to site 2
2. At site 2: derive $S' = \{t \in S | B[h(t.C)] = 1\}$
3. Sent S' to site 1
4. At site 1: $R \bowtie_{A=C} S'$ is computed producing a result equivalent to $R \bowtie_{A=C} S$

This algorithm is illustrated by an example in Fig. 2 using a simple hash function $h(v) = v \bmod 7$. Applying h to column A of relation R produces the vector of seven bits ([0111000]) which is used to probe the S-tuples at site 2 by computing $h(C)$. Note, that $\surd$ indicates a match and $-$ a non-match.

Note that a hash function is usually not injective and therefore the problem of collision occurs, i.e., for different values $v_1 \neq v_2$ one can have $h(v_1) = h(v_2)$. Thus, useless tuples are sent to site 1 in step 3 which will not contribute to the final result, e.g. the tuple with $C = 8$ in this example. This problem can be mitigated by choosing a bit vector of an appropriate length. An alternate



Semijoin. Figure 2. Example of hash filter join.



Semijoin. Figure 3. Comparison of semijoin algorithms.

approach is to use multiple hash functions $h_1,\ldots,h_k$ together with the associated bit vectors $B_1,\ldots,B_k$ and to set the bits for a value v in each bit vector:

$$B_1[h_1(v)] = 1, B_2[h_2(v)] = 1,\ldots,B_k[h_k(v)] = 1$$

All these bit vectors are sent to site 2 and used there for probing. A tuple $t \in S$ qualifies only to be a candidate tuple if all bits are set to 1, i.e., if the result of the bitwise AND is 1. It can be shown that with an increasing k the collision probability comes close to 0.

Key Applications

The main application of semijoin techniques is distributed join processing, where the semijoin acts as a reducer. Though, experimental work has shown that the computational overhead is typically higher than the savings in transfer cost, particularly the hash filter strategy is often an attractive alternative.

Variants of the semijoin are also used for processing queries in heterogeneous databases where a component database provides only limited query capabilities, e.g. selections with parameters (also called bindings). If a set of tuples is sent to the component database as binding parameter, this corresponds in fact to the semijoin strategy.

Finally, semijoins are also useful for processing star queries in data warehouses. Here, the semijoin technique is exploited for joining each dimension table with the fact table (or more exactly an index on the fact table) in order to collect the rowids of the fact tuples. Then, the intersection of all rowid sets is computed which is finally used to retrieve the tuples from the fact table.

Experimental Results

Mackert and Lohman [7] report results of an experimental analysis of the performance of distributed join strategies in the R* system. Though, the experiments were conducted on a hardware which was up-to-date in the eighties (e.g., a highspeed network with 4 Mbit/s effective transfer rate), the general trend of the results is still valid.

Figure 3 shows the results of a comparison of several strategies for computing $R \bowtie S$ where the cardinality of R was $|R| = 1{,}000$ and the cardinality of S varied from 100 to 6,000.

In this experiment, the hash filter join clearly outperformed the other strategies. Only for small cardinalities where the inner relation S fits into the buffer, the semijoin has advantages. The third join variant was the R* strategy of shipping one relation to the other site and exploiting local indexes for join processing.

Cross-references

▶ Distributed Join
▶ Evaluation of Relational Operators
▶ Semijoin Program

Recommended Reading

1. Babb E. Implementing a relational database by means of specialized hardware. ACM Trans. Database Syst., 4(1):1–29, 1979.
2. Bernstein P.A. and Chiu D-M.W. Using semi-joins to solve relational queries. J. ACM, 28(1):25–50, 1981.
3. Bernstein P.A., Goodman N., Wong E., Reeve C.L., and Rothnie Jr. Query processing in a system for distributed databases (SDD-1). ACM Trans. Database Syst., 6(4):602–625, 1981.
4. Bloom B.H. Space/time trade-offs in hash coding with allowable errors. Commun. ACM, 13(7):422–426, 1970.
5. Hevner A.R. and Yao S.B Query Processing in distributed database systems. IEEE Trans. Software Eng., 5(3):177–182, 1979.
6. Lu H. and Carey M. Some experimental results on distributed join algorithms in a local network. In Proc. 11th Int. Conf. on Very Large Data Bases, 1985, pp. 229–304.
7. Mackert L.F. and Lohman G. R* optimizer validation and performance evaluation for local queries. In Proc. ACM SIGMOD Int. Conf. on Management on Data, 1986, pp. 4–95.
8. Özsu M.T. and Valduriez P. Principles of distributed database systems, 2nd edn. Prentice-Hall, 1999.
9. Valduriez P. Semi-join algorithms for distributed database machines. In J.-J. Schneider (ed.). Distributed Data Bases. North-Holland, 1982, pp. 23–37.
10. Valduriez P. and Gardarin G. Join and semi join algorithms for a multiprocessor database machine. ACM Trans. Database Syst., 9(1):133–161, 1984.

Semijoin Filter

▶ Semijoin

Semi-Structured Data

▶ Semi-structured Data Model

Semi-Structured Query Languages

▶ Semi-structured query language

Semijoin Program

Stéphane Bressan
National University of Singapore, Singapore,
Singapore

Synonyms

Semijoin reducer

Definition

A semijoin program is a query execution plan for queries to distributed database systems that uses semijoins to reduce the size of relation instances before they are transmitted and further joined. Yet the reduction itself requires that a projection of the relation instances involved in the join onto the join attributes be transmitted. The maximum amount of reduction can be achieved by a semijoin program called a full reducer. Full reducers that do not require the computation of a fixpoint exist for acyclic queries. Fully reducing relation instances is rarely beneficial. However semijoin programs partially reducing selected relation instances may be an effective optimization when the dominant cost of query execution is communication. Considering semijoin programs considerably increases the distributed query optimization search space.

Historical Background

Semijoin programs were first introduced to improve the performance, input/output operations and communication of database applications running on database machines [7] and on parallel database machines [10]. These machine were possibly equipped with specialized hardware such as filters efficiently implementing semijoins. SDD-1 [3] is the first distributed database management system making use of semijoin programs for query optimization. Bernstein and Chiu, in [1], review the algorithms for these early applications. In these approaches, reduction of relation instances and the other steps of global and local optimization are conducted as successive phases of the distributed query optimization. Stoker et al. [8], revisit semijoin programs for modern applications and empirically evaluate their usefulness. The authors propose dynamic programming query optimization algorithms that integrate the selection of selected semijoin reducers with join ordering into a single phase.

The hyper-graph representation of relational queries and the notion of acyclic queries is from Fagin [5]. Bernstein et al. introduce comparable notions [2] and define the notion of full reducer. Several authors, see for instance [11] and, more recently [6], have discussed the relationship between reducers and constraint satisfaction problems in the context of database query optimization.

The textbook [4] gives a good overview of semijoin programs and their use in early distributed database systems. It describes the use of semijoin programs in SDD-1 while the presentation in [9] emphasizes the notions of reducers, full reducers, and reduction algorithms.

Foundations

Cyclic and Acyclic Query Hyper-Graphs

For the sake of simplicity, queries are considered which contain natural joins only. The hyper-graph representing such queries is composed of vertices corresponding to attributes and hyper-edges corresponding to relations.

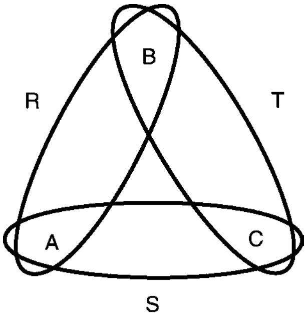
Given, for instance, the three relations $R(A, B)$, $S(A, C)$ and $T(C, B)$, the query $R \bowtie (S \bowtie T)$ is the natural join of R with the natural join of S and T. Its hyper-graph is represented on Fig. 1.

In addition, consider the relation $U(C, D)$. The query $R \bowtie (S \bowtie U)$ is the natural join of R with the natural join of S and U. Its hyper-graph is represented on Fig. 2.

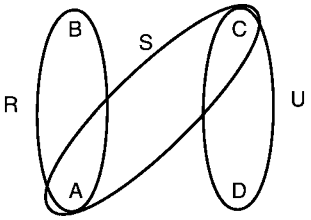
An *ear* of a hyper-graph is a hyper-edge that contains a vertex that does not belong to any other hyper-edge. R and U in Fig. 2. are ears. Notice that removing one ear may create new ears. If all hyper-edges of a hyper-graph can be removed by iteratively removing ears, the hyper-graph is said to be acyclic. The hyper-graph of Fig. 2. is acyclic. The ears U, S and R can be removed in this order, for instance. The hyper-graph of Fig. 1. is cyclic: none of the hyper-edges is an ear.

Reducer and Full Reducer

A reducer for a relation R with respect to a query Q is a program of semijoins with other relations in Q applied to R, such that R can be replaced by the result of the program in Q without changes in the result of the



Semijoin Program. Figure 1. Hyper-graph representing $R \bowtie (S \bowtie T)$.



Semijoin Program. Figure 2. Hyper-graph representing $R \bowtie (S \bowtie U)$.

query *Q*. In other words, the semijoins possibly remove some tuples that do not contribute to the query *Q*. The program $R \ltimes S$ is a reducer of *R* in both queries of Figs. Fig 1. and Fig 2, respectively. The reader can verify that with the instances of *R* and *S* given in Fig. 3, the tuple *R* (1, 2) is removed from *R* by the semijoin. This tuple neither contributes to the result of the query of Fig. 1 nor to the result of the query of Fig. 2.

A full reducer is a reducer that eliminates all the tuples that do not contribute to the result of the query for any instance of the relations in the query. The program $R \ltimes (S \ltimes U)$ is a full reducer of *R* for the query of Fig. 2. For the instances of relations *R*, *S* and *U* given in Fig. 3, it reduces *R* to the instance given in Fig. 4.

Fully reducing *R* for the query of Fig. 1. requires a fixpoint program that iteratively applies semijoin until no more tuples can be eliminated. It is clear that the number of iterations in the fixpoint depends on the actual instances. For the instances *R*, *S* and *T* in Fig. 3, the query of Fig. 1. denotes an empty result. *R* can only be reduced to the empty relation by if one iteratively applies the semijoins $S \ltimes T$, $R \ltimes S$ and $T \ltimes R$ until a fixpoint is reached. In the example the fully reduced instance of *R* is empty. This is not always the case.

R	
A	B
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9

S	
A	C
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8

T	
C	B
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8

U	
C	D
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8

Semijoin Program. Figure 3. Instances of *R*, *S*, *T*, and *U*.

$R \ltimes (S \ltimes U)$	
A	B
3	4
4	5
5	6
6	7
7	8
8	9

Semijoin Program. Figure 4. *R* fully reduced.

There always exists a full reducer for acyclic queries while cyclic queries always require a fixpoint iteration.

Semijoins, Distributed Query Optimization and Semijoin Programs

Given two relation instances *R* and *S* located on two different servers, the computation of the join $R \bowtie S$ would normally consist in sending a copy of either one of the two instances from the server where it resides to the other server. Depending on the attributes of *S*, on the join condition and on the selectivity of the join, the volume of data transferred might be reduced significantly if the projection of *S* onto the join attributes is shipped to the server on which *R* is located (or conversely without loss of generality) where they can be used to compute the semijoin $R \ltimes S$. The result is shipped to the site of *S* where the original join can be computed.

Example instances of the relations *R*(*A*, *B*) and *S* (*A*, *C*) allocated to server *S*1 and *S*2, respectively, are given in Fig. 5.

The natural join of *R* and *S* is equivalent to the following expressions.

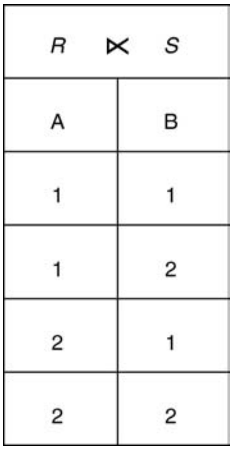
$$(R \bowtie \pi_A(S)) \bowtie S = (R \bowtie S) \bowtie S$$

The projection $\pi_A(S)$ is executed on *S*2. The result is sent to *S*1 and used to compute the semijoin $(S \ltimes R)$. The result of the semijoin for the instances of Fig. 5. is given in Fig. 6. The result of the semijoin is sent to *S*2. The join $(R \ltimes S) \bowtie S$ is computed on *S*2.

R	
A	B
1	1
1	2
2	1
2	2
3	1
3	2
4	1
4	2

S	
A	C
1	1
1	2
2	3
2	4
5	1
5	2
6	3
6	4

Semijoin Program. Figure 5. The instance of relation *R* is on site *S*1 and the instance of relation *S* is on site *S*2.



R ⋉ S	
A	B
1	1
1	2
2	1
2	2

Semijoin Program. Figure 6. Semijoin of *R* and *S*.

This plan is valuable if the volume of data in the projection of *S* on *A* plus the volume of data in the results of its join with R is significantly smaller than the volume of data in *S*.

$$volume(\pi_A(S)) + volume(S \bowtie (\pi_A\,(S) \bowtie R))volume(S)$$

In the example, if each integer is 4 bytes, by using a semijoin, 4 × 4 + 4 × 2 × 4 bytes = 48 bytes are transmitted, instead of 8 × 2 × 4 bytes = 64 bytes.

Semijoin programs are implementation of (full or partial) reducers by means of semijoins. Their traditional application to distributed query optimization consists in the reduction of fragments before fragments are shipped from one server to another, as illustrated in the example above. It is rarely beneficial to fully reduce relations, even for acyclic queries.

Considering semijoin programs for query optimization significantly increases the number of candidate query execution plans and, therefore, the search space of the optimization algorithm.

Key Applications

Semijoin programs have been designed and used for the optimization of queries in early distributed database systems, at a time when communication, i.e., data transmission, was the dominant cost. Although semijoin programs fell in desuetude, the authors of [8] argue that they cannot only significantly reduce communication cost in modern distributed database systems, but also allow a better utilization of resources for some applications in both centralized and distributed systems.

Cross-references

▶ Bloom Join

▶ Distributed Join

▶ Distributed Query Processing

▶ Semijoin

Recommended Reading

1. Bernstein P.A. and Chiu D.-M. W. Using semi-joins to solve relational queries. J. ACM., 28(1):25–40, 1981.
2. Bernstein P.A. and Goodman N. Power of natural semijoins. SIAM J. Comput., 10(4):751–771, 1981.
3. Bernstein P.A., Goodman N., Wong E., Reeve C.L., and Rothnie J.B. Query processing in a system for distributed databases (SDD-1) ACM Trans. Database Syst., 6(4):602–625, 1981.
4. Ceri S. and Pelagatti G. Distributed databases: Principles and systems. McGraw-Hill, 1984.
5. Fagin R. Degrees of acyclicity for hypergraphs and relational database schemes. J. ACM, 30(3):514–550, 1983.
6. Lal A. and Choueiry B.Y. Constraint processing techniques for improving join computation: a proof of concept. In Proc. 1st Int. Symp. on Applications of Constraint Databases, 2004, pp. 149–167.
7. Ozkarahan E.A., Schuster S.A., and Sevcik K.C. Performance evaluation of a relational associative processor ACM Trans. Database Syst., 2(2):175–195, 1977.
8. Stocker K., Kossmann D., Braumandl R., and Kemper A. Integrating semi-join-reducers into state of the art query processors. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 575–584.
9. Ullman J.D. Principles of Database and Knowledge-Base Systems, Vol. II. Computer Science, 1989.
10. Valduriez P. and Gardarin G. Join and semijoin algorithms for a multiprocessor database machine. ACM Trans. Database Syst., 9(1):133–161, 1984.
11. Wallace M., Bressan S., and Provost T.L. Magic checking: constraint checking for database query optimization. In Proc. ESPRIT WG CONTESSA Workshop on Constraint databases and Applications. LNCS, Vol. 1034. Springer, 1995, pp. 148–166.

Semijoin Reducer

▶ Semijoin Program

Semi-Streaming Model

▶ Graph Mining on Streams

Semi-Structured Data

Serge Abiteboul
INRIA-Saclay, Ile-de-France, Orsay, Cedex, France

Synonyms

XML (almost)

Definition

A semi-structured data model is based on an organization of data in labeled trees (possibly graphs) and on query languages for accessing and updating data. The labels capture the structural information. Since these models are considered in the context of data exchange, they typically propose some form of data serialization, i.e., a standard representation of data in files. Indeed, the most successful such model, namely XML (that is promoted by the W3C), is often confused with its serialization syntax. XML equipped with query/update language [10] is a semi-structured data model.

Semi-structured data models are meant to represent from very structured to very unstructured information, and in particular, irregular data. In a structured data model such as the relational model [9], one distinguishes between the type of the data (schema in relational terminology) and the data itself (instance in relational terminology). In semi-structured data models, this distinction is blurred. One sometimes speaks of schemaless data although it is more appropriate to speak of self-describing data. Semi-structured data may possibly be typed. For instance, tree automata have been considered for typing XML. However, semi-structured data applications typically use very flexible and tolerant typing or sometimes no typing at all.

Historical Background

Before the Web, publication of electronic data was limited to a few scientific and technical areas. With the Web and HTML, it rapidly became universal. HTML is a format meant for presenting documents to humans. However, a lot of the data published on the Web is produced by machines. Moreover, it is more and more the case that Web data are consumed by machines. Since HTML is not appropriate for machine processing, this lead in the 1990's to the development of *semi-structured data models* and most importantly of a new standard for the Web, namely XML. The use of a semi-structured data model as a standard for *data representation* and *data exchange* on the Web brought important improvement to the publication and reuse of electronic data by providing a simple syntax for data that is machine-readable and at the same time, human readable (with the help of the so-called "style-sheets").

Semi-structured data models may be viewed, in some sense, as bringing together two cultures that were for a long while seen as irreconcilable, document systems (with notably SGML [8]) and database systems (with notably relational systems [9]). From a model perspective, there are many similarities with the object database model [5]. Indeed, like XML, the object database model is also based on trees, provides an object API, comes equipped with a query language and offers some form of serialization. A main difference is that the very rigorous typing of object databases was abandoned in semi-structured data models.

The articulation of the notion of semi-structured data may be traced to two simultaneous origins, the OEM model at Stanford [3,6] and the UnQL model at U. Penn [4].

Specific data formats had been previously proposed and even became sometimes popular in specific domains, e.g. ASN.1 [7]. The essential difference between data exchange formats and semi-structured data models is the presence of high level query languages in the latter. A query language for SGML is considered in [2]. Languages for semi-structured data models such as [3,4] then paved the way for languages for XML [10].

Foundations

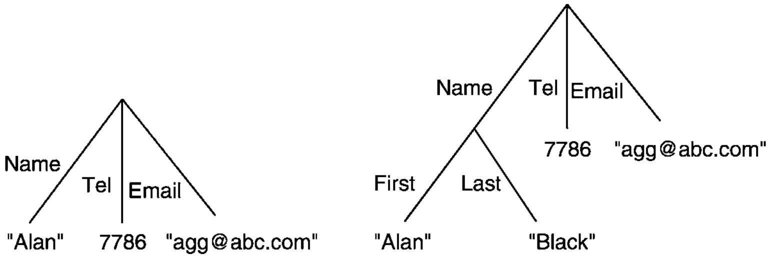
One can start with an idea familiar to Lisp programmers of association lists, which are nothing more than label-value pairs and are used to represent record-like or tuple-like structures:

```
{name: "Alan," tel: 2157786, email:
"agb@abc.com"}
```

This is simply a set of pairs such as `name: "Alan"` consisting of a label and a value. The values may themselves be other structures as in:

```
{name: {first: "Alan," last: "Black"},
tel: 2157786,
email: "agb@abc.com"}
```

This data may be represented graphically with nodes denoting object, connected by edges to values,



Semi-Structured Data. Figure 1. Tree representation.

see Fig. 1. Departing from the usual assumption made about tuples or association lists that labels are unique, duplicate labels may be allowed as in:

```
   {name: "alan,'' tel: 2157786, tel: 2498762 }
```

The syntax makes it easy to describe sets of tuples as in:

```
   {person: {name: "alan," phone: 3127786,
email: "agg@abc.com"},
   person: {name: "sara," phone: 2136877,
email: "sara@math.xyz.edu"},
   person: {name: "fred," phone: 7786312,
email: "fds@acme.co.uk"}}
```

Furthermore, one of the main strengths of semi-structured data is its ability to accommodate variations in structure, e.g., all the `Person` tuples do not need to have the same type. The variations typically consist of missing data, duplicated fields or minor changes in representation, as in the following example:

```
   {person: {name: "alan," phone: 3127786,
email: "agg@abc.com"},
   person: &314
     {name: {first: "Sara," last: "Green"},
     phone: 2136877,
     email: "sara@math.xyz.edu,"
     spouse: &443
   person: &443
     {name: "fred," Phone: 7786312, Height:
183,
      spouse: &314}}
```

Observe how identifiers (here &443 and &314) and references are used to represent graph data. It should be obvious by now that a wide range of data structures, including those of the relational and object database models, can be described with this format.

As already mentioned, in semi-structured data, the conscious decision is made of possibly not caring about the type the data might have, and serialize it by annotating each data item explicitly with its description (such as `name, phone`, etc). Such data are called *self-describing*. The term *serialization* means converting the data into a byte stream that can be easily transmitted and reconstructed at the receiver. Of course self-describing data wastes space, since these descriptions are repeated for each data item, but more interoperability is achieved, which is crucial in the Web context.

There have been different proposals for semi-structured data models. They differ in choices such as labels on nodes versus on edges, trees versus graphs, ordered trees versus unordered trees. Most importantly, they differ in the languages they offer.

Key Applications

The main applications of semi-structured data models are found on the Web.

First, semi-structured data models and XML are very useful for data *publication*. XML is also serving as a universal *data exchange* format in a wide variety of fields, from bioinformatics to e-commerce. It presents the advantage compared to previous formats that it comes equipped with an array of available software such as parsers or programming interfaces. Also, the flexibility of the typing in semi-structured data models turns out to be essential for *data integration*, and in particular in the integration of heterogeneous data in mediator systems.

Cross-references

▶ Document representations (incl. native and relational)
▶ Semi-Structured Data Model
▶ W3C XML Query Language
▶ XML
▶ XML Types

Recommended Reading

1. Abiteboul S., Buneman P., and Suciu D. Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann, 1999.
2. Abiteboul S., Cluet S., Christophides V., Milo T., Moerkotte G., and Simeon J. Querying documents in object databases. Int. J. Digit. Libr., 1(1):5–19, 1997.
3. Abiteboul S., Quass D., McHugh J., Widom J., and Wiener J. The Lorel query language for semistructured data. Int. J. Digit. Libr., 1(1):68–88, 1997.
4. Buneman P., Davidson S., and Suciu D. Programming constructs for unstructured data. In Proc. 5th Int. Workshop on Database Programming Languages, 1995.
5. Cattell R.G.G. The Object Database Standard: ODMG-93. Morgan Kaufmann Publishers, 1994.
6. Papakonstantinou Y., Garcia-Molina H., and Widom J. Object exchange across heterogeneous information sources. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 251–260.
7. Specification of Abstraction Syntax Notation One (ASN.1), ISO Standard 8824, Information Processing System, 1987.
8. Standard Generalized Markup Language (SGML), ISO 8879, 1986.
9. Ullman J.D. Principles of Database and Knowledge-Base Systems, Vol. I: Classical Database Systems. Computer Science, 1988.
10. XQuery. XQuery 1.0: An XML query language. http://www.w3.org/TR/Xquery

Semi-Structured Data Model

DAN SUCIU
University of Washington, Seattle, WA, USA

Synonyms

Semi-Structured data

Definition

The semi-structured data model is designed as an evolution of the relational data model that allows the representation of data with a flexible structure. Some items may have missing attributes, others may have extra attributes, some items may have two ore more occurrences of the same attribute. The type of an attribute is also flexible: it may be an atomic value, or it may be another record or collection. Moreover, collections may be heterogeneous, i.e., they may contain items with different structures. The semi-structured data model is self-describing data model, in which the data values and the schema components co-exist. Formally:

Definition 0.1

A semi-structured data instance is a rooted, directed graph in which the edges carry labels representing schema components, and leaf nodes (i.e., nodes without any outgoing edges) are labeled with data values (integers, reals, strings, etc.).

There are two variations of semi-structured data, depending on how one interprets equality. In the object exchange model (OEM) introduced in Tsimmis [8], each node in the data has its own identity, and thus two data instances are "equal" if and only if they are isomorphic, and this corresponds to the bag semantics of collections. In the value-based model introduced in UnQL [2], two data graphs are equal if they are bisimilar; this corresponds to the set semantics of collections.

A variation of the semi-structured data model is one in which labels are placed on nodes rather than edges.

Historical Background

The term *semi-structured data* was introduced by Luniewski et al. in 1993 in a system called Rufus [6,9]. In 1995, Papakonstantinou et al. introduced a data model for semi-structured data (called *object exchange model*, OEM) for a system for integrating heterogeneous databases called Tsimmis [5,8]. In 1995, Buneman et al. introduced a data model for biological data where equality is based on bisimulation [1–3]. The connection between the semi-structured data model and XML was described in 1999 by Deutsch et al., who proposed a query language for XML called XML-QL [4].

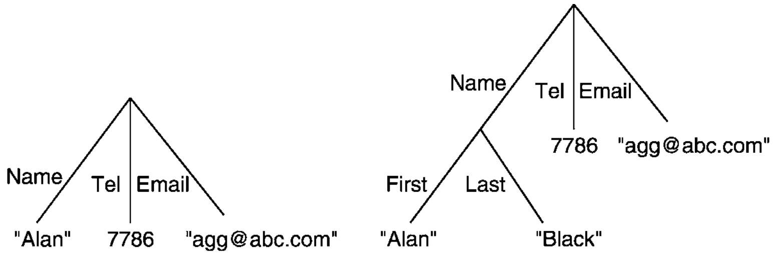
Foundations

Semi-structured data are *schema-less* or *self-describing*. Both the data and its schema is described directly using a simple syntax for sets of label-value pairs, similar to association lists in Lisp. For example:

```
{name: "Alan", tel: 2157786, email:
"agb@abc.com"}
```

This is simply a set of pairs such as `name: "Alan"` consisting of a label and a value. The values may themselves be other structures as in

```
{name: {first: "Alan", last: "Black"},
tel: 2157786,
email: "agb@abc.com"
}
```



Semi-Structured Data Model. Figure 1. Graph representations of simple structures.

One may represent this data graphically as a node that represents the object, connected by edges to values, see Fig. 1.

Unlike in traditional tuples or association lists the labels are not necessarily unique, and duplicate labels are allowed, as in:

```
{name: "alan, tel: 2157786, tel:
2498762}
```

The syntax makes it easy to describe sets of tuples as in

```
{person:
  {name: "alan", phone: 3127786,
  email: "agg@abc.com"},
person:
  {name: "sara", phone: 2136877,
  email: "sara@math.xyz.edu"},
person:
  {name: "fred", phone: 7786312,
  email: "fds@acme.co.uk"}
}
```

Person tuples do not necessarily have to be of the same type. One of the main strengths of semi-structured data is its ability to accommodate variations in structure. While in principle semi-structured data could become a completely random graph, data instances that are usually found in practice are "close" to some type, and have only minor variations from that type. The variations typically consist of missing data, duplicated fields or minor changes in representation, as in the example below.

```
{person:
  {name: "alan", phone: 3127786,
  email: "agg@abc.com"},
person:
  {name: {first: "Sara", last:
  "Green"},
  phone: 2136877,
  email: "sara@math.xyz.edu"
  },
person:
  {name: "fred", Phone:
  7786312 Height: 183}
}
```

Representing relational databases. It is easy to represent every relational database as a semi-structured data, which happens to have a regular structure. For example the relational database instance:

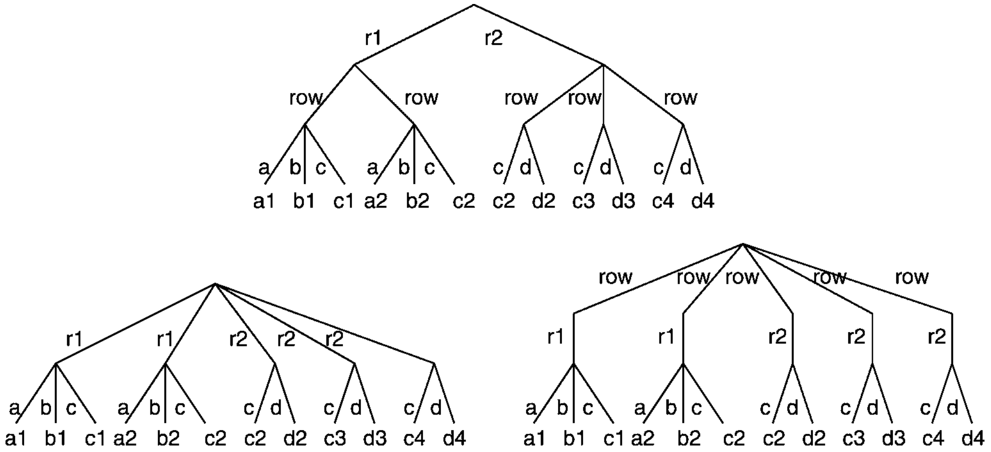
r1	a	b	c
	a1	b1	c1
	a2	b2	c2

r2	c	d
	c2	d2
	c3	d3
	c4	d4

can be described as a set of rows:

```
{r1: {row: {a: a1, b: b1,  c: c1},
      row: {a: a2, b: b2,  c: c2}
     },
r2: {row: {c: c2,  d: d2},
     row: {c: c3,  d: d3},
     row: {c: c4,  d: d4}
    }
}
```

It is worth noting that this is not the only possible representation of a relational database. Figure 2 shows tree diagrams for the syntax given above and for two other representations of the same relational database.


r1
r2
row
row
row
row
row
a
b
c
a
b
c
c
d
c
d
c
d
a1 b1 c1 a2 b2 c2 c2 d2 c3 d3 c4 d4
r1
r1
r2
r2
r2
a
b
c
a
b
c
c
d
c
d
c
d
a1 b1 c1 a2 b2 c2 c2 d2 c3 d3 c4 d4
row
row
row
row
row
r1
r1
r2
r2
r2
a
b
c
a
b
c
c
d
c
d
c
d
a1 b1 c1 a2 b2 c2 c2 d2 c3 d3 c4 d4

Semi-Structured Data Model. Figure 2. Three representations of a relational database.

Representing object databases. Consider for example the following collection of three persons, in which `Mary` has two children, `John` and `Jane`. Object identities may be used to construct structures with references to other objects.

```
{person: &o1{name: "Mary",
             age: 45,
             child: &o2,
             child: &o3
            },
person: &o2{name: "John",
            age: 17,
            relatives: {mother: &o1,
            sister: &o3}
            },
person: &o3{name: "Jane",
            country: "Canada",
            mother: &o1
            }
}
```

The presence of a label such as `&o1` before a structure binds `&o1` to the identity of that structure. This makes it possible to use that label – as a value – to refer to that structure. In this graph representation it is allowed to build graphs with shared substructures and cycles, as shown in Fig. 3. The name `&o1`, `&o2`, `&o3` are called *object identities*, or oid's. In this figure, arrows are placed on the edges to indicate the direction, which is no longer implicit, like in the tree-like structure.

The object exchange model (OEM) was explicitly defined for the purpose of integrating heterogeneous data sources in Tsimmis. An OEM object is a quadruple `(label, oid, type, value)`, where `label` is a character string, `oid` it the object's identifier, `type` is either `complex`, or some identifier denoting an atomic type (like `integer`, `string`, `gif-image`, etc.). When `type` is `complex`, then the object is called a *complex object*, and `value` is a set (or list) of oid's. Otherwise the object is an *atomic object* and the `value` is an atomic value of that type. Thus OEM data are essentially a graph, but in which labels are attached to nodes rather than edges.

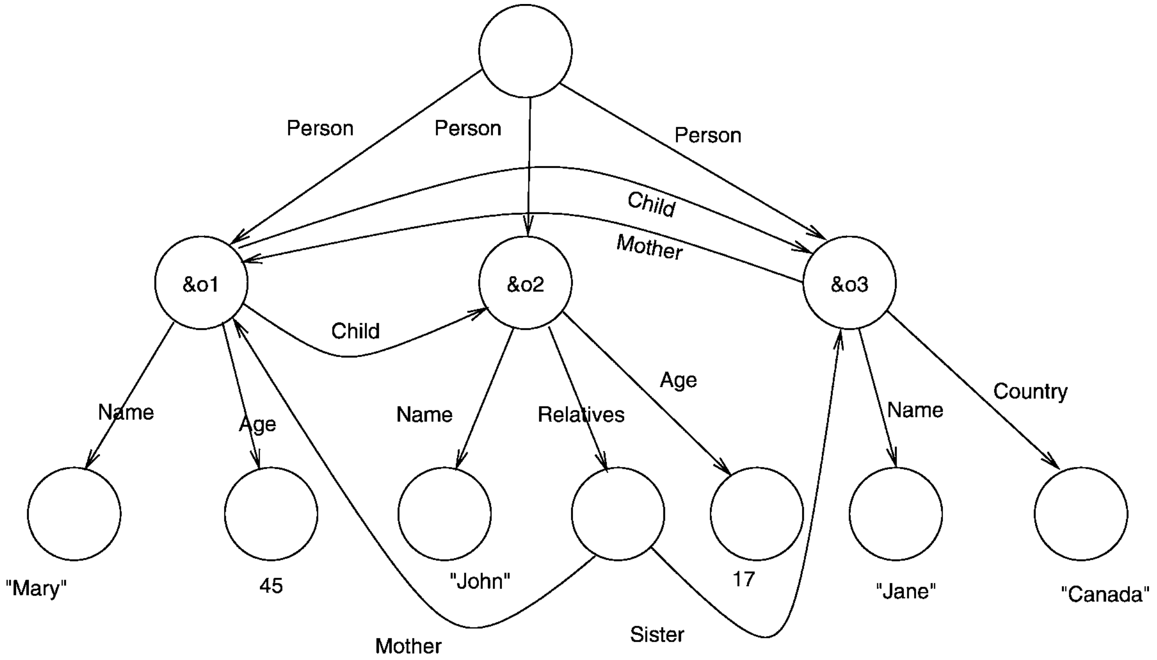
Equality in semi-structured data. A shallow notion of equality simply checks whether two object identifiers are the same, or two data values are equal. Beyond that a deep notion of equality is needed, which addresses the following question: given two semi-structured data instances (i.e., two graphs), do they represent the same data? This question is fundamental in query optimization, since it allows replacement of one query expression with another if the instances they return are equal.

Two notions of deep equality have been considered. One is *graph isomorphism*: two data instances are equal if there exists an isomorphism that preserves the edge labels and the data values.

Definition 0.2

An isomorphism *between two semi-structured data instances D_1, D_2 is a function f mapping the nodes of D_1 to the nodes of D_2 such that:*

1. *f is a bijection.*
2. *f maps the root of D_1 to the root of D_2.*



Semi-Structured Data Model. Figure 3. A cyclic structure.

3. *If there exists an edge from a node x to a node y in D_1 then there exists an edge from the node f (x) to the node f (y) in D_2 and both edges have the same label.*
4. *Conversely, if there exists an edge from f (x) to f (y) in D_2 then there exists an edge* (It follows from the previous condition that the two edges have the same label.) *from x to y in D_2.*
5. *If a leaf node x in D_1 is labeled with a data value v then the node f (x) in D_2 is labeled with the same data value* (It follows from the previous conditions that $f(x)$ is also a leaf node.)*v.*

In the first interpretation two data instances are equal if there exists an isomorphism between them. For example the two instances $\{a : 3, b : 5, c : 7, b : 9\}$ and $\{b : 5, b : 9, a : 3, c : 7\}$ are equal, because they are isomorphic. On the other hand, the instances $\{a : 3, a : 3, b : 5\}$ and $\{a : 3, b : 5\}$ are not equal. Thus, when restricted to collections this notion of equality corresponds to the bag semantics.

The second notion of equality is based on *bisimulation*: two data instances are equal if there exists a bisimulation:

Definition 0.3

A bisimulation *between two semi-structured data instances D_1, D_2 is a relation $R(x, x')$ between the nodes in the two instances s.t.*

1. *If r_1 is the root node in D_1 and r_2 is the root node in D_2, then $R(r_1, r_2)$.*
2. *If $R(x, x')$ holds, and D_1 contains an edge (x, y) with label a, then there exists an edge (x', y') labeled a in D_2, and $R(y, y')$ holds.*
3. *Symmetrically, if $R(x, x')$ holds and D_2 contains an edge (x', y') with label a then there exists an edge (x, y) in D_1 labeled a, and $R(y, y')$ holds.*
4. *If $R(x, x')$ holds and the node x in D_1 is a leaf node, and is labeled with the atomic value v then the node x′ in D_2 is also a leaf node and labeled with the same atomic value v (the symmetric property follows automatically).*

In the second interpretation, two semi-structured data instances are said to be equal if there exists a bisimulation between them. For example $\{a : 3, a : 3, b : 5\}$ is equal to $\{a : 3, b : 5\}$ because, denoting r, n_1, n_2, n_3 the nodes in the first graph and r', n_1', n_2' the nodes in the second graph, the relation $R(r, r')$, $R(n_1, n_1')$, $R(n_2, n_1')$, $R(n_3, n_2')$ is a bisimulation. Thus, the equality based on bisimulation corresponds to the set semantics on collections.

If two data instances are isomorphic, then there always exists a bisimulation between them; the converse does not always hold. Checking whether two data instances are isomorphic is a computationally hard problem. By contrast, checking if two data instances are bisimilar can be done efficiently [7].

Edge vs. node labeled graphs. The model described here is that of an *edge-labeled graph.* A minor variation is one in which nodes are labeled, and this has gained a lot of popularity since the introduction of XML.

Key Applications

The initial motivation for the introduction of semi-structured data was to support the integration of heterogeneous data, and to model non-standard data formats, especially in the bioinformatics domain, s.a. ACEDB and ASN.1. After the introduction of XML, this became the main application of semi-structured data.

Cross-references

▶ Semi-Structured Data
▶ Semi-Structured Query Languages
▶ XML

Recommended Reading

1. Buneman P., Davidson S., and Suciu D. Programming constructs for unstructured data. In Proc. Workshop on Database Programming Languages, 1995.
2. Buneman P., Davidson S., Hillebrand G., and Suciu D. A query language and optimization techniques for unstructured data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 505–516.
3. Buneman P., Fernandez M., and Suciu D. UNQL: A query language and algebra for semistructured data based on structural recursion. VLDB J., 9(1):76–110, 2000.
4. Deutsch A., Fernandez M., Florescu D., Levy A., and Suciu D. A query language for XML. In Proc. 8th Int. World Wide Web Conference, 1999, pp. 77–91.
5. Garcia-Molina H., Papakonstantinou Y., Quass D., Rajaraman A., Sagiv Y., Ullman J., and Widom J. The TSIMMIS project: integration of heterogeneous information sources. J. Intell. Inf. Syst., 8(2):117–132, March 1997.
6. Luniewski A., Schwarz P., Shoens K., Stamos J., and Thomas J. Information organization using Rufus. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 560–561.
7. Paige R. and Tarjan R. Three partition refinement algorithms. SIAM J. Comput., 16:973–988, 1987.
8. Papakonstantinou Y., Garcia-Molina H., and Widom J. Object exchange across heterogeneous information sources. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 251–260.
9. Shoens K., Luniewski A., Schwarz P., Stamos J., and Thomas II J. The Rufus system: Information organization for semi-structured data. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 97–107.

Semi-Structured Database

▶ Graph Database

Semi-Structured Database Design

Gillian Dobbie[1], Tok Wang Ling[2]
[1]University of Auckland, Auckland, New Zealand
[2]National University of Singapore, Singapore, Singapore

Synonyms

XML Database Design

Definition

From a requirements document, a database designer distills the real world constraints and designs a database schema. While the design process for structured data is well defined, the design process for semi-structured data is not as well understood. What is a "good" design for semi-structured databases that captures real world constraints, prevents data redundancy and update anomalies, and allows typical queries to execute quickly?

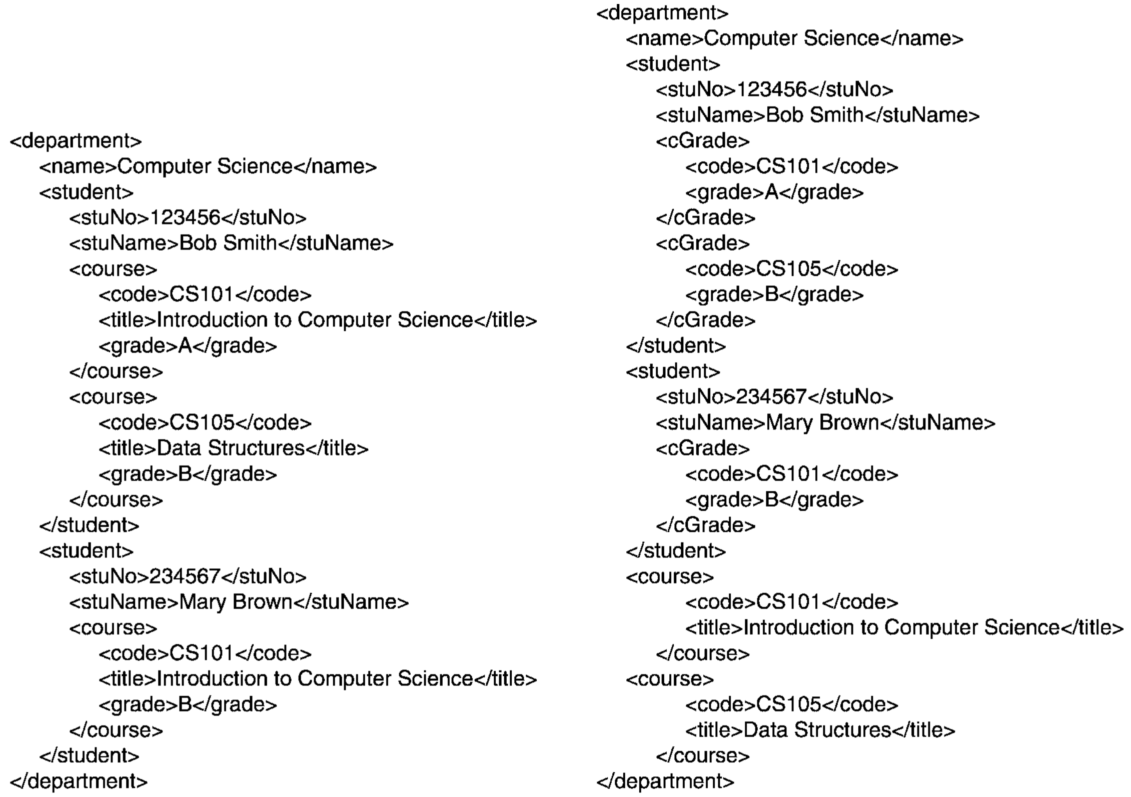
Historical Background

There was a lot of research into the design of relational databases in the 1970s, and it was found that the design of relational databases involves a trade off between the speed of execution of queries and the updating anomalies caused by maintaining redundant data when updates occur. During logical schema design normalization algorithms are used to reduce redundancy, and during physical design to improve performance some redundancy may be reintroduced, views can be created over the schema, and indexes may be introduced.

Semi-structured data differs from relational data in a number of ways: it is hierarchical, the queries that are posed are more complex, a database may or may not have a schema, and there is no generally agreed upon mathematical foundation. Because of these differences, there is a need for a different design process for semi-structured databases.

Foundations

Consider the XML document in Fig. 1a. It models the courses that students are taking within a department, and the grade of each student taking a course. What is the best way to organise this information? There are various possibilities, such as modeling *course* as a subelement of *student*, or modeling *student* as a subelement of *course*. While both of these options



```
<department>
   <name>Computer Science</name>
   <student>
      <stuNo>123456</stuNo>
      <stuName>Bob Smith</stuName>
      <course>
         <code>CS101</code>
         <title>Introduction to Computer Science</title>
         <grade>A</grade>
      </course>
      <course>
         <code>CS105</code>
         <title>Data Structures</title>
         <grade>B</grade>
      </course>
   </student>
   <student>
      <stuNo>234567</stuNo>
      <stuName>Mary Brown</stuName>
      <course>
         <code>CS101</code>
         <title>Introduction to Computer Science</title>
         <grade>B</grade>
      </course>
   </student>
</department>
```

a An XML Document

```
<department>
   <name>Computer Science</name>
   <student>
      <stuNo>123456</stuNo>
      <stuName>Bob Smith</stuName>
      <cGrade>
         <code>CS101</code>
         <grade>A</grade>
      </cGrade>
      <cGrade>
         <code>CS105</code>
         <grade>B</grade>
      </cGrade>
   </student>
   <student>
      <stuNo>234567</stuNo>
      <stuName>Mary Brown</stuName>
      <cGrade>
         <code>CS101</code>
         <grade>B</grade>
      </cGrade>
   </student>
   <course>
         <code>CS101</code>
         <title>Introduction to Computer Science</title>
      </course>
   <course>
         <code>CS105</code>
         <title>Data Structures</title>
      </course>
</department>
```

b A normalized XML Document

Semi-Structured Database Design. Figure 1. An original and normalized XML Document.

seem quite natural, Fig. 1a demonstrates that these options involve repeating information. The details of the course, i.e., title in this example, are repeated for each student that takes that course. Consequently, if the title of a course changes, it must be updated in every *title* element of the course. Brandin [2] describes a simple way to define XML documents, in which the modeler designs simple forms entailing the information to be modeled, and uses the form headings as tags and the entries on the form as data. While the simplicity of methods like this is appealing, they can lead to similar anomalies that arise with relational databases.

In order to ensure a "good" design for semi-structured databases, the following steps must be followed:

1. Choose a data model that is able to represent the semantics necessary for modeling semi-structured data.
2. Capture the semantics of the data that will be stored, either by:
   (a) Extracting the schema from a set of documents and discovering the semantics in a data model, or
   (b) Studying the constraints in the real world and capturing them in a data model.
3. Reorganize the schema into a normalized schema to avoid replication of data in the XML documents.
4. Consider the typical query set and reorganize the schema to improve the performance of typical queries, perhaps by introducing controlled replication of data.
5. Consider the users of the system and define views over the data for individual users or groups of users.

A data model for semi-structured database design needs to model the logical structure of the schema from a real world perspective, much like an ER diagram [4], while also modeling the physical aspects that are representative of XML documents, such as the hierarchical relationship between elements. This enables a designer to first model the real world constraints on the data, and using the same diagram, capture the extra constraints that are introduced with XML. For example, consider again the scenario captured in Fig. 1a. From a real world perspective the data model must model the relationship between students and courses, and from an XML perspective the data model must be able to capture whether student should be modeled as a subelement of course or vice-versa. The data model must also capture the participation constraints between elements, because they also change the way the data are modeled. For example, there is a many-to-many relationship between courses and students and this will be modeled differently than a one-to-many relationship say between department and employees. In order to capture these requirements, the data model must be able to model n-ary relationship sets, cardinality, participation and uniqueness constraints, ordering, irregular and heterogeneous structures, for both data- and document-centric data. One such data model is ORA-SS (Object Relationship Attribute Data Model for Semi-Structured Data) [6].

One of the advantages of semi-structured data is that it is self-describing, and so it does not require a schema. However, after XML was introduced, it was soon realized that schemas offer many benefits, and as a consequence schema languages have been defined for XML recently [3,8]. Some semantics of a document can be extracted from the schema if one exists or extracted from the document. In [7,9] and Chap. 4 of [6], the authors have described how an approximate schema can be extracted from semi-structured data. The algorithms generally follow two steps. The first step extracts the structural information, such as elements and their subelements. The second step infers semantics from the original document, such as key attributes, attributes of object classes vs attributes of relationship types, and participation constraints. Normalization is used to identify and reduce anomalies, and a key ingredient in normalization algorithms is functional dependencies. Consider the real world constraints that should be modeled in the document in Fig. 1a. There is a many-to-many relationship between elements *student* and *course*, attributes *stuNo* and *stuName* belong to *student*, attributes *code* and *title* belong to *course*, and attribute *grade* belongs to the relationship type between *student* and *course*. These relationships may be modeled as functional dependencies, such as $stuNo \rightarrow stuName$, $code \rightarrow title$, and $\{stuNo, code\} \rightarrow grade$. Yu and Jagadish [10] describe a system, DiscoverXFD, which given an XML document, discovers XML functional dependencies and data redundancies. However, multi-valued dependencies (MVDs) are also very important in relational database design, such as in the definition of 4NF. Like with relational databases, functional dependencies cannot describe all redundancy in XML databases, so MVDs must also be considered.

The motivation for normalization algorithms for semi-structured databases is similar to that for relational databases, namely the identification of

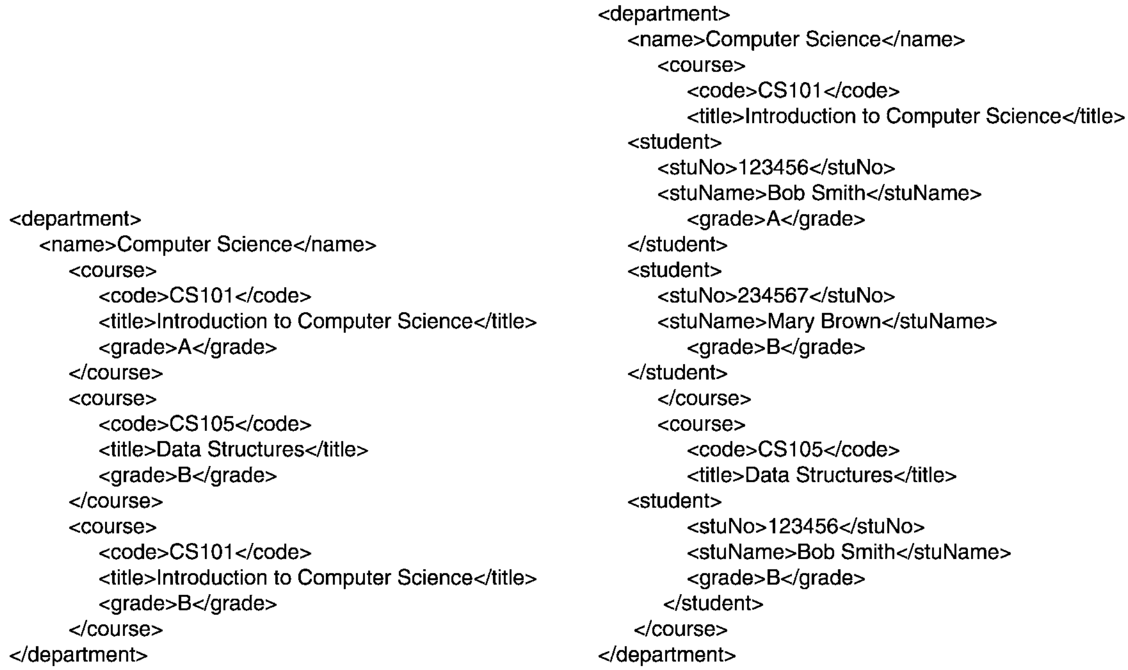
```
<department>
   <name>Computer Science</name>
   <student>
      <stuNo>123456</stuNo>>
      <stuName>Bob Smith</stuName>
       <C_G>
          <course>CS101</course>
          <grade>A</grade>
       </C_G>
       <C_G>
          <course>CS105</course>
          <grade>B</grade>
       </C_G>
   </student>
   <student>
      <stuNo>234567</stuNo>
      <stuName>Mary Brown</stuName>
       <C_G>
          <course>CS101</course>
          <grade>B</grade>
       </C_G>
   </student>
```

```
<course>
   <code>CS101</code>
   <title>Introduction to Computer Science</title>
    <S_G>
       <student>123456</student>
       <grade>A</grade>
    </S_G>
    <S_G>
       <student>234567</student>
       <grade>B</grade>
    </S_G>
  </course>
  <course>
    <code>CS105</code>
    <title>Data Structures</title>
     <S_G>
        <student>123456</student>
        <grade>B</grade>
      </S_G>
  </course>
</department>
```

Semi-Structured Database Design. Figure 2. A symmetric relationship in an XML Document for *department*.

redundant data that lead to update anomalies. The normalization algorithms must recognize the hierarchical structure of semi-structured data, the participation constraints of both parents and children in hierarchical relationships, and whether an attribute is an attribute of an object class or the attribute of a relationship type. The algorithm described in [1] converts an arbitrary DTD into a well-designed one, using path functional dependencies. They show that the normal form that they define, XNF, generalizes BCNF for XML documents. This algorithm works with the semantics available in the DTD along with additional functional dependencies, however it does not consider MVDs, so documents in XNF may still contain redundancy. Using the semantics expressed in ORA-SS, it is possible to capture the multi-valued attributes of object classes and relationship types, avoiding the redundancy because of the existence of multi-valued attributes or MVDs. Note each multi-valued attribute of an object class is a MVD, e.g. if a student can have many hobbies, *studNo* → *hobby*. As hobby is not involved in any functional dependencies, XNF will not be able to detect such redundancy but the normalization algorithm defined in [6] deals with this case. Figure 1b shows a normalized version of the XML document shown in Fig. 1a. Note that the details of each course are no longer stored for each student that takes the course. These details are extracted and stored in a *course* element which is a sibling to the *student* elements. A new element, *cGrade*, is introduced to record the grade a student scored in a course. The element *code* in *cGrade* is effectively a reference to *course*.

During physical database design, the database schema is tuned to ensure that queries, which are likely to be asked often, will run faster. This can be done by reintroducing redundant data in a controlled way, considering how indexes will improve execution times, or by introducing views. There has not been a lot of work carried out in the area of physical database design or database tuning for semi-structured databases. When reintroducing redundant data, it is necessary to consider both the cost of storage and the cost of updating the redundant data. Consider for example, the XML document in Fig. 1a. The information that is repeated is the title of the course, and it is repeated for every student taking the course. In this case, the title of a course is not likely to be



```
<department>
    <name>Computer Science</name>
        <course>
            <code>CS101</code>
            <title>Introduction to Computer Science</title>
            <grade>A</grade>
        </course>
        <course>
            <code>CS105</code>
            <title>Data Structures</title>
            <grade>B</grade>
        </course>
        <course>
            <code>CS101</code>
            <title>Introduction to Computer Science</title>
            <grade>B</grade>
        </course>
</department>
```

a An invalid view over the XML Document in Figure1a

```
<department>
    <name>Computer Science</name>
        <course>
            <code>CS101</code>
            <title>Introduction to Computer Science</title>
    <student>
        <stuNo>123456</stuNo>
        <stuName>Bob Smith</stuName>
            <grade>A</grade>
    </student>
    <student>
        <stuNo>234567</stuNo>
        <stuName>Mary Brown</stuName>
            <grade>B</grade>
    </student>
        </course>
        <course>
            <code>CS105</code>
            <title>Data Structures</title>
    <student>
            <stuNo>123456</stuNo>
            <stuName>Bob Smith</stuName>
            <grade>B</grade>
         </student>
     </course>
</department>
```

b An valid view over the XML Document in Figure 1a

Semi-Structured Database Design. Figure 3. Views over the XML Document in Fig. 1a.

updated very often. If users were expected to ask a query often that lists the students name and the titles of all the courses a student is taking, then with this profile the designer would opt to retain this level of redundancy. In fact, *title* is an example of a relatively stable attribute, that is an attribute that is updated infrequently. For relatively stable attributes or relatively stable relationship types [5] one needs to consider only the cost of extra storage since these data are not updated often. For all replications ensure that the necessary controls are put in place to maintain the integrity of the data. For example, for relatively stable attributes and relationship types, it is necessary to enforce that replicated data are consistent when the data are inserted.

In some instances, it is not obvious whether to embed one object class inside another or vice versa. Consider the relationship in Fig. 1a. If it is equally likely that users will ask for all the students by course, and all the courses by student then the best design may be a symmetric relationship type. For the pairing required for symmetric relationship types, a reference is duplicated. The consistency of this duplication must be enforced when information is inserted, deleted or updated. A symmetric relationship type is modeled in Fig. 2. In this case, within the *student* element a reference to *course* is stored and in the *course* element a reference to *student* is stored. The *grade* is stored in both *student* and *course.* This approach is similar to a bidirectional relationship with physical pairing in IBM's Information Management System (IMS).

When views are defined, one of the challenges is ensuring that the view over the underlying data is valid. None of the systems available today check the validity of a view against the semantics of the underlying data. In [6], a set of operators is described that can be used to define views, along with guidelines of how the operators can be used to produce valid views. For example, a view over the XML document in Fig. 1a is shown in Fig. 3a. This view is very simple, and was created by extracting all the course elements from the original document. However, this view is invalid because although it lists course codes with their titles, each *course* element also has a grade attached to it. Having removed the context of student, the information about who the grade was assigned to is lost. The operators that are needed to create views are select, drop, join, and swap. The select operator filters elements for which a particular condition is true and does not alter the schema. For example, a view over the XML document in Fig. 1a can be created using the select operator of all students and courses where the student achieved an "*A*" grade. The drop operator drops elements or attributes from the source schema. For example, it is possible to create a view by dropping the student elements from the XML document in Fig. 1a. However, it is important that if an element which represents an object class is dropped, then all its attributes or attributes of relationship types in which it participates must also be dropped, otherwise it is possible to create an invalid view such as that shown in Fig. 3a. Note that *grade* is an attribute of the relationship type between the object classes *student* and *course* and should be removed when student is dropped. The join operator joins two elements where one contains a foreign key of another. For example, if the underlying schema is that shown in Fig. 2 it would be possible to create a view that joins the course and student elements in Fig. 2 to improve the performance of queries that ask for the courses taken by a student with a specific *stuNo.* In order to guarantee the validity of the drop operator it is necessary to ensure that all ancestors and descendents of the object class being joined are dealt with appropriately. The swap operator exchanges the position of a parent element with a child element. For example, the XML document in Fig. 1a embeds *course* elements in *student* elements. In order to answer the query that asks for the students taking a particular course, it is possible to use the swap operator to create a view where *student* elements are embedded in *course* elements. When the swap operator is used, the relationship types must be updated, and the attributes that belong to the relationship types must also be moved to the appropriate place. The relationship types that need to be updated are those among the elements involved in the swap. Figure 3b shows a valid view over the XML document in Fig. 1a, created by swapping the object classes *student* and *course.*

Key Applications

The motivation for studying this area is to ensure that databases that are designed to store semi-structured data will always contain consistent data, and will always provide answers to queries in an acceptable time. The people that will benefit most from this research are the users of semi-structured database systems.

Future Directions

There are a number of key areas that deserve further investigation:

- Definition of a standardized data model for semi-structured data
- Investigation and comparison of the normalization algorithms that have been proposed for semi-structured data
- Investigation and experimentation of physical database design techniques for semi-structured data
- Case studies that show how well the research works in practice
- Cost models for queries over XML databases

Cross-references

► Entity Relationship Model
► Normal Form ORA-SS Schema Diagrams.
► Object Relationship Attribute Model for semi-structured Data
► Database Design
► Semi-structured Data Models
► XML
► Semi-structured Query Languages

Recommended Reading

1. Arenas M. and Libkin L. A normal form for XML documents. ACM Trans. Database Syst., 29(1):195–232, 2004.
2. Brandin C. Information Modeling with XML. In A. Chaudhri, A. Raschid, and R. Zicari (ed.). XML Data Management. Addison-Wesley, 2003, pp. 3–17.
3. Bray T., Paoli J., and Sperberg-McQueen C.M. Extensible markup language (XML) 1.0., 2nd edn. October 2000.
4. Chen P.P. The entity-relationship model – toward a unified view of data. ACM Trans. Database Syst., 1(1):9–36, 1976.
5. Ling T.W., Goh C.H., Lee M.-L. Extending classical functional dependencies for physical database design. Inf. & Software Tech., 38(9):601–608, 1996.
6. Ling T.W., Lee M.L., and Dobbie G. Semistructured Database Design. Springer, 2005.
7. Nestorov S., Abiteboul S., and Motwani R. Extracting schema from semistructured data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 295–306.
8. Thompson H.S., Beech D., Maloney M., and N. Mendelson (eds.). XML Schema Part 1: Structures. May 2001. http://www.w3.org/TR/xmlschema-1.
9. Wang Q.Y., Yu J.X., and Wong K.F. Approximate graph schema extraction for semi-structured data. In Advances in Database Technology, Proc. 7th Int. Conf. on Extending Database Technology, 2000, pp. 302–317.
10. Yu C. and Jagadish H.V. Efficient discovery of XML data redundancies. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 103–114.

Semi-Structured Query Languages

Dan Suciu
University of Washington, Seattle, WA, USA

Synonyms

Semi-Structured query languages

Definition

A query language for semi-structured data allows a user to extract items from a semi-structured data instance, or to transform it into another semi-structured data instance. The first type of expressions are called *queries*, the latter kind of expressions are interchangeably called *queries* or *transformations*. Query languages can be classified along several dimensions:

1. *Expressive power.* What kind of queries or transformations can they express?
2. *Querying vs. restructuring.* Some query languages allow only the extraction of items from the data, others allow the data to be transformed.
3. *Compositionality.* Can the output of a query be used as input in another query expressed in the language, and is the composed transformation still expressible in the same language? Query languages that are restricted to extraction only are not compositional, because the type of their outputs are not semi-structured data instances. Transformation languages may fail to be compositional if the composition of two queries cannot be expressed in the same language.

Historical Background

Query languages for semi-structured data were introduced almost as early as the data model itself. Lorel [1] was one of the first query language for semi-structured data, and it introduced the concept of regular path expressions for navigating paths with partially known structure. The language UnQL [2] emphasized transformations over queries, and introduced structural recursion as a central paradigm for transforming semi-structured data. MSL [5] introduced Skolem functions in a logic-based language for transforming semi-structured data: these were later used extensively in Strudel [4], a system for declarative specification of Web sites. The first language to apply the principles of semi-structured data query languages to XML was XML-QL [3].

Foundations

Path expressions. A node sequence of edge labels l_1. l_2,...,l_n is called a *path expression*. In its most general form the path expression denotes all pairs of nodes (x, y) such that there exist edges (x, l_1, v_1),(v_1, l_2, v_2),..., (v_{n-1}, l_n, y). When used in a query language, x is either specified explicitly or assumed by default to be the root node. Thus, a path expression denotes a *set of nodes*, namely the set of nodes y, given the fixed x.

A *regular path expression* is

$$e ::= l \mid \epsilon \mid - \mid e\text{‘.’}e \mid \text{‘(’}e\text{‘)’} \mid e\text{‘|’}e \mid e\text{‘}_*\text{’} \mid e\text{‘+’} \mid e?$$

where l ranges over label, e over expressions and ε is the empty expression. The expression $e*$ stands for 0 or more repeats of e, i.e., for

$$\epsilon \mid e \mid e.e \mid e.e.e \mid e.e.e.e \mid \ldots$$

Also, e+ stands for one or more repeats, and e? for 0 or one occurrence of e.

(Regular) path expression form a simple query language for semi-structured data, or can be embedded in a simple query syntax to define such a language. For example the query below returns all nodes X accessible by the path `biblio.book.author`.

```
% Query q1
select X
from biblio.book.author X
```

Note that a query language consisting of path expressions is not compositional, since queries return sets of nodes, not pieces of semi-structured data.

Patterns. A convenient way to combine multiple path expressions is through patters. Consider the following query, returning all titles that were published before 1979:

```
% Query q2:
select X
from biblio.paper T, T.title X, T.year Y
where Y < 1979
```

Note that here the expression `T.title` starts from the node `T` rather than from the root. The query can be written more concisely as:

```
% Query q3:
select X
from {biblio: {paper: {title: X, year:
  Y}}}
where Y < 1979
```

Here `biblio: paper: title: X, year:Y` is a *pattern.*

Constructors. To return semi-structured data rather than nodes a query language needs to have *constructors.* For example the expressions `authorResult:X` and `result:{title X, author Z}` are constructors below:

```
% Query q4:
select authorResult:X
from biblio.book.author X
```

```
% Query q5:
select result:{title: X, author: Z}
from biblio.paper T, T.title X, T.year
  Y, T.author Z
where Y < 1979
```

More complex constructors can built by nesting subqueries inside constructors, as in:

```
% Query q6:
select row:( select author: Y
  from X.author Y)
from biblio.book X
```

```
% Query q7:
select row: ( select author: Y, title: T
from X.author Y
  X.title T)
from biblio.paper X
where "Roux" in X.author
```

Declarative semantics. The formal semantics of a query is given in three steps. In the first step a set of bindings to all variables is computed, which results in a relation that has one column for every variable in the query, and one row for every binding of these variables. In the second step the predicates in the `where` clause are applied to select a subset of the rows. In the third step, the constructor is applied to each remaining row. For example in query `q6` above, the effect of the first step is to construct a relation of tuples of the form (t, x, y, z), where t, x, y, z are nodes or atomic values in the input databases. The second step selects only those tuples for which z < 1979. The third step constructs a partial graph for each remaining tuple (t, x, y, z) consisting of a root (common among all these graphs), an edge labeled `result`, and from there two more edges labeled `title` and `author`.

Skolem functions. A Skolem function takes as input some arguments and constructs as output a fresh new node. The essential property of a Skolem function is that if called again at a later time, on the same arguments, then it returns the same node for those argument, not a new one. This allows for duplicate elimination, grouping, and the construction of cyclic outputs. The query below is a standard grouping query that groups publications by their years:

```
% Query q8:
select resultYear f(Y): {paper:
  {title: X, author: Z}
from biblio.paper T, T.title X, T.year
  Y, T.author Z
```

Here *f*(*Y*) is the Skolem function. Without it, the query would construct a separate `resultYear` node for every binding of the variables `T`, `X`, `Y`, and `Z`. The Skolem function determines the query to construct only one node for every year, thus performing a duplicate elimination on years.

The combination of Skolem functions and regular path expressions leads to transformation languages that are not compositional. Consider a semi-structured data instance that represents a binary table *R*:

```
R: { row: {a: 4, b:8},
   row: {a: 3, b:4},
   row: {a: 3, b:9},
   ...
}
```

It is known that the relational calculus cannot express the transitive closure of *R*. The same holds for the language described so far, since its only addition to the relational calculus consists of regular path expressions, but on this simple data instance these expressions can only be applied to very short paths, namely `R.row.a` and `R.row.b`, and therefore do not give extra power. However, the transitive closure can be expressed by first transforming the relation *R* into a graph, i.e., making all atomic values into nodes, then using a regular expression on this graph to compute the transitive closure. The two queries are

```
% Query q9:
select node f(X): {value: X, next: f(Y)}
from row T, T.a X, T.b Y
```

```
% Query q10:
select result: {a: X, b:Y}
from node U, U.value X, U.next* V, V.
value Y
```

The first query constructs for every value *x* of the `a` attribute a new node *f*(*x*) with two outgoing edges: one to a leaf holding the value *x*, and the other to the node *f*(*y*). Thus, the edges from *f*(*x*) to *f*(*y*) in the output graph materialize the implicit graph given by the binary relation *R*. The second query uses the regular expression `next*` to compute the transitive closure on this graph.

Key Applications

See applications for semi-structured data.

Cross-references

- ▶ Indexing Semi-Structured Data
- ▶ Semi-Structured Data
- ▶ Semi-structured Data
- ▶ Semi-structured Data Model
- ▶ Top-k XML Query Processing
- ▶ XML
- ▶ XML Tree Pattern, XML Twig Query
- ▶ XPath/XQuery

Recommended Reading

1. Abiteboul S., Quass D., McHugh J., Widom J., and Wiener J. The Lorel query language for semistructured data, 1996. http://www-db.stanford.edu/lore/.
2. Buneman P., Davidson S., and Suciu D. Programming constructs for unstructured data. In Proc. Workshop on Database Programming Languages, 1995.
3. Deutsch A., Fernandez M., Florescu D., Levy A., and Suciu D. A query language for XML. In Proc. 8th Int. World Wide Web Conference, 1999, pp. 77–91.
4. Fernandez M., Florescu D., Kang J., Levy A., and Suciu D. Catching the boat with Strudel: experience with a Web-site management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 414–425.
5. Papakonstantinou Y., Abiteboul S., and Garcia-Molina H. Object fusion in mediator systems. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 413–424.

Semi-Structured Text Retrieval

▶ Biomedical Scientific Textual Data Types and Processing

Semi-Supervised Classification

▶ Semi-Supervised Learning

Semi-Supervised Clustering

▶ Clustering with Constraints

Semi-Supervised Learning

Sugato Basu
Google Inc, Mountain View, CA, USA

Synonyms

Semi-supervised classification

Definition

In machine learning and data mining, supervised algorithms (e.g., classification) typically learn a model for predicting an output variable (e.g., class label for classification) from some supervised training data (e.g., data instances annotated with both features and class labels). These algorithms use various techniques of increasing the accuracy of predicting the training data labels, by minimizing a loss function that measures the prediction error on the training data. They also use different regularization methods to ensure that the model does not overtrain on the training data, thereby having good prediction performance on unseen test data.

In semi-supervised learning, unlabeled data (i.e., data instances with only features) are used along with the labeled training data, in an effort to improve the accuracy of the models on the training data as well as provide better generalization performance on unseen data. This paradigm is very useful in practical applications, where unlabeled data are generally available in more abundance (since in most cases unlabeled data are easier to collect) and at a substantially lower cost (e.g., less human effort) than labeled data.

In the published literature, the term semi-supervised learning usually refers to semi-supervised classification. However it can also refer to semi-supervised regression, where unlabeled data are used to improve the performance of a regression algorithm that predicts a real valued output instead of a class label. A related area of research is semi-supervised clustering, where small amounts of supervision is used to improve the performance of clustering algorithms on unlabeled data.

Historical Background

As outlined in [3] (see Chap. 1 for a detailed history of this area, with more references), the earliest known work [10] that suggested using unlabeled data to improve classification performance is self-learning or self-training. The method first trains a classifier on just the labeled data. In each successive step, the trained model predicts the labels on the unlabeled data instances and uses a part of that (e.g., the labels on which it has highest prediction confidence) as additional labeled data for training in the following step.

Transductive learning [12] is an idea closely related to semi-supervised learning – it uses only the features of the test data instances as unlabeled data to improve the performance of the classifier learned on the labeled data. The main difference between transductive learning and semi-supervised learning is that the latter is an inductive method, i.e., it can generalize to give predictions on new data, while transduction can only give predictions for the given finite set of test data. The relative benefits of these two approaches are well compared in Chap. 25 of [3].

In the 1970s, researchers started to investigate the use of unlabeled data in different classification algorithms, e.g., Fisher's linear discriminant [6], using iterative refinement algorithms like expectation maximization (EM) [5]. The 1990s saw a significant increase of research in semi-supervised learning, especially related to different application areas in text and natural language processing [2,4,7,8,13]. This was accompanied by theoretical analysis of semi-supervised learning algorithms, notably using the PAC learning framework [9].

For detailed historical background and overview of recent work, readers are referred to the excellent surveys [11,14] and book [3] on semi-supervised learning.

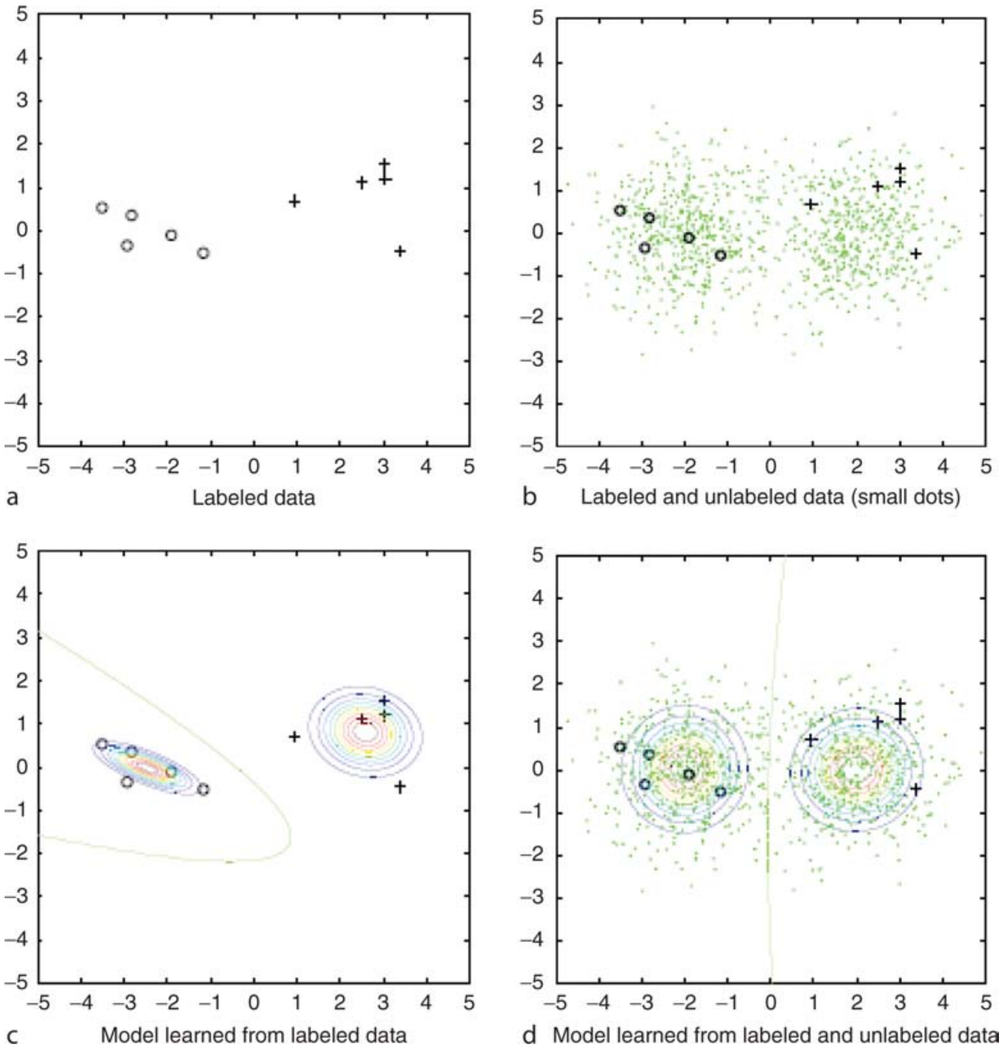
Foundations

To concretize the concepts introduced in the definition, consider that the features of the ith data instance in the training data X is represented by the vector x_i, and the corresponding output variable is $y_i \varepsilon Y$. In classification, Y is a set of labels, while in regression it a set of real-values. In semi-supervised learning, the training set X consists of two components: (i) the supervised data X_l, where the data instances x_l are annotated with the corresponding y values; (ii) the unsupervised data X_u, where the data instances x_u do not have corresponding y values.

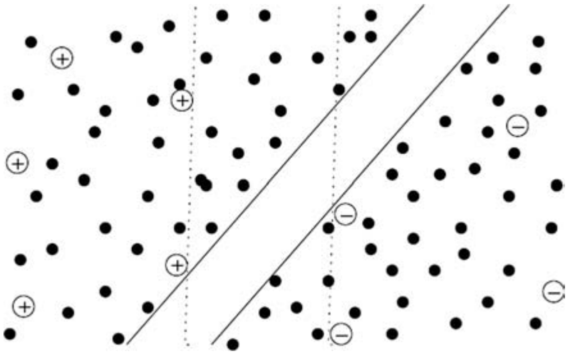
There are different types of semi-supervised learning methods, some of which are listed below:

1. *Generative models*. They estimate the class-conditional probability of the data $p(x|y)$, and use the unlabeled data X_u as additional information to improve the model fitting (as shown in Fig. 1). For example, [8] first trains a naive-Bayes classifier C_l on the labeled data X_l and then uses EM to re-train the classifier on the unlabeled data X_u, using their estimated class probabilities $p(y_u|x_u)$ of the unlabeled data instances (estimated by C_l) as fractional class labels during the EM iterations.

2. *Low density separation*. Models following this principle push the decision boundary of the classifier away from the high-density regions of the unlabeled data. For example, [7] trains a transductive support vector machine (SVM) that maximizes the margin of the SVM classifier on both labeled and unlabeled data – it first trains a classifier C_l using only the labeled data X_l, and then uses the predictions $p(y_u|x_u)$ of C_l on the unlabeled data X_U to retrain a new SVM. The



Semi-Supervised Learning. Figure 1. Shows the effect of using unlabeled data in semi-supervised classification. The class boundary of the generative model-based classifier (d) trained using both the labeled and unlabeled data are quite different from the one trained using just the labeled data (c) [14].



Semi-Supervised Learning. Figure 2. Shows the effect of using unlabeled data in learning a SVM classifier. The *dotted line* shows the classifier trained on only the labeled training data, while the *solid line* is the classifier trained on both labeled and unlabeled data [14].

process is repeated iteratively, increasing the weights of the unlabeled data in each iteration, thereby forcing the classifier decision boundary to pass through the low-density regions of X_u (as shown in Fig. 2).

3. *Graph-based methods.* These approaches encode the data as nodes in a graph (labeled data instances have class labels associated with the nodes), with edges encoding pairwise distance between data instances. They operate under the assumption that the original (high-dimensional) data can be projected to a low-dimensional manifold without significant loss of information, and use the unlabeled points for different types of regularization, e.g., classify unlabeled points using label propagation algorithms that prefer smoothness of inferred classes across edges [15].

There are various other methods of semi-supervised learning that employ different techniques of using unlabeled data to improve supervised learning, e.g., co-training [2], semi-supervised manifold embedding [1], which are outlined in the book and surveys on semi-supervised learning mentioned in the historical background section.

Key Applications

Semi-supervised learning has been applied successfully to various applications in text analysis (e.g., document categorization), natural language processing (e.g., word sense disambiguation), bioinformatics (e.g., protein classification, protein function prediction) image analysis (e.g., handwritten digit/character recognition, face detection, facial expression recognition), and speech recognition (e.g., speaker identification).

Cross-references

- ▶ Classification
- ▶ Clustering with Constraints

Recommended Reading

1. Belkin M. and Niyogi P. Semi-supervised learning on manifolds. Technical Report, The University of Chicago, TR-2002-12, 2002.
2. Blum A. and Mitchell T. Combining labeled and unlabeled data with co-training. In Proc. 11th Annual Conf. on Computational Learning Theory, 1998, pp. 92–100.
3. Chapelle O., Schölkopf B., and Zien A. (eds.). Semi-supervised learning. MIT Press, Cambridge, MA, 2006.
4. Collins M. and Singer Y. Unsupervised models for named entity classification. In Proc. Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora, 1999.
5. Dempster A.P., Laird N.M., and Rubin DB. Maximum likelihood from incomplete data via the EM algorithm. J. R. Stat. Soc. B, 39:1–38, 1977.
6. Hosmer D.W. Jr. A comparison of iterative maximum likelihood estimates of the parameters of a mixture of two normal distributions under three different types of sample. Biometrics, 29(4):761–770, 1973.
7. Joachims T. Transductive inference for text classification using support vector machines. In Proc. 16th Int. Conf. on Machine Learning, 1999, pp. 200–209.
8. Nigam K., McCallum A., Thrun S., and Mitchell T. Learning to classify text from labeled and unlabeled documents. In Proc. 11th National Conf. on AI, 1998, pp. 792–799.
9. Ratsaby J. and Venkatesh S.S. Learning from a mixture of labeled and unlabeled examples with parametric side information. In Proc. Eighth Annual Conf. on Computational Learning Theory, 1995, pp. 412–417.
10. Scudder H.J. Probability of error of some adaptive pattern-recognition machines. IEEE Trans. Inf. Theory, 11:363–371, 1965.
11. Seeger M. Learning with labeled and unlabeled data. Technical Report, Edinburgh University, 2001.
12. Vapnik V.N. and Chervonenkis A. Theory of pattern recognition [in Russian]. Nauka, Moscow, 1974.
13. Yarowsky D. Unsupervised word sense disambiguation rivaling supervised methods. In Proc. 23rd Annual Meeting of the Assoc. for Computational Linguistics, 1995, pp. 189–196.
14. Zhu X. Semi-supervised learning literature survey. Computer Sciences Technical Report TR 1530, University of Wisconsin Madison, 2006.
15. Zhu X., Ghahramani Z., and Lafferty J. Semi-supervised learning using Gaussian fields and harmonic functions. In Proc. 20th Int. Conf. on Machine Learning, 2003.

S

Sense and Respond Systems

▶ Event Driven Architecture

Sensitivity

► Precision and Recall

Sensor Network Systems

► Event Driven Architecture

Sensor Networks

Ramesh Govindan
University of Southern California, Los Angeles,
CA, USA

Synonyms

Wireless sensor networks; Embedded networked sensing; Sensornet

Definition

A sensor network is a collection of small battery-operated embedded devices each with a built-in processor, sensors, and wireless communication capability. It usually has one or more *gateways*, or embedded computers, connected to an external communications infrastructure (e.g. the Internet or the cellular network). Either the embedded devices or the gateways may be mobile. The purpose of a sensor network is to collaboratively sense the environment, and convey (a possibly condensed version of) the sensed information to one or more human users via the gateways. More advanced sensor networks may autonomously react to the sensed information and effect an action, such as activating a camera or a light switch.

Historical Background

As early as 1977, researchers appear to have discussed, in significant depth, the various issues in building a distributed wireless sensor network. These discussions were part of the ARPA Sensor Net project, and foreshadowed many of the issues that sensor network researchers have been more recently addressing. Moreover, with great prescience, project participants enumerated many application scenarios that are relevant even today: airborne deployment of sensors, wireless surveillance, tracking, and so on.

Perhaps the first concrete instance of a wireless sensor node was the WINS prototype developed by Pottie and Kaiser [2]. The development of these prototypes made significant advances in low-power electronics and device integration. Moreover, the WINS project anticipated the need for sophisticated network self-configuration in wireless sensor networks.

However, sensor networking really blossomed as a separate discipline following the "Simple Systems" DARPA ISAT study led by Deborah Estrin, and the subsequent establishment of a DARPA-funded program networked sensing called SenseIT. The SenseIT program explored a number of important early research directions in the field, including collaborative signal processing, energy-efficient data dissemination, and the application of database concepts to data management in sensor networks.

Foundations

The technological constraints imposed by battery-operated wireless sensors have resulted in five categories of research questions that have been explored by the sensor networks community:

1. *Energy-awareness.* Since sensor nodes operate on batteries, the lifetime of a sensor network is determined by how quickly individual sensor nodes drain their energy resources. This line of research examines techniques for increasing network lifetime through careful energy management or harvesting.
2. *Communication reliability.* Wireless communication is notoriously unpredictable, and sensor networks are often deployed in heavily obstructed environments. As such, being able to reliably retrieve sensed data, while still being energy-efficient, is a significant challenge.
3. *Spatio-temporal awareness.* Information generated by sensors is useful only when it is associated with a spatio-temporal context that indicates when and where the data was generated. To do this, sensor nodes need accurate positioning and time synchronization technologies. Unfortunately, these nodes are often deployed in locations (indoors, or in foliage) where the Global Positioning System (GPS) is unavailable.
4. *Programmability.* Sensor networks have a very large number of potential applications, and there is a dire need for novel programming paradigms that

hide the complexity imposed by the previous challenges, yet promote reusable and easily understood systems.

5. *Security and Data integrity.* Because sensors may be deployed in unattended and harsh environments, there is critical need for software that ensures the security of the data generated by the sensors, and the overall integrity or correctness of the data.

The following paragraphs summarize research in these areas.

Energy-awareness. Perhaps most widely studied, research in improving network lifetime has proceeded on two fronts. There has been exploratory work on technologies for harvesting energy from light sources, from vibration, or by using mobility to find a nearby power source. This line of research aims to renew the supply of energy at a node. More extensively studied is the direction in which careful algorithms and systems design techniques are used to conserve energy. Specifically, it has been experimentally determined that communication and sensing are among the primary contributors to energy usage. To conserve communication energy, researchers have explored techniques like *data aggregation* [6,11] where sensor readings are processed and possibly compressed within the network before being transmitted. In addition, techniques employed at various software layers (medium access, routing, and application) that turn the radio off or put the node to seek during periods of inactivity have also been studied. To conserve sensing energy, researchers have proposed exploiting correlations between sensors, or using one sensor to trigger another higher-energy sensor.

Communication reliability. Wireless packet transmissions are susceptible to losses or corruption due to environmental noise and collisions from concurrent transmissions. To overcome losses due to packet corruption, researchers have explored careful coding techniques that allow for packet reconstruction using relatively small amounts of information. Another line of research has explored the construction of high-quality routing paths which attempt to reduce the likelihood of corruption [15]. To ensure reliable end-to-end delivery, researchers have explored the combined use of hop-by-hop recovery and end-to-end retransmissions. To reduce packet loss due to contention, a line of work has explored *congestion control* techniques that dynamically adapt a node's sending rate to avoid saturating the channel capacity [13]. These techniques essentially sample the current channel conditions, and adjust node sending rates according to feedback control laws that ensure stability and efficiency.

Spatio-temporal awareness. When a sensor node is deployed, it faces two important questions: "Where am I?" and "What is the time?" Without answers to these questions, the data generated by sensor networks becomes meaningless. Since GPS cannot be assumed to work well in the kinds of obstructed environments that sensor networks are likely to be deployed in, researchers have explored a wide variety of methods for *localization* and *time synchronization.* For the former problem, a variety of devices (such as ultrasound, radio, lasers) have been used to estimate the distances between two nodes, and a variety of techniques ranging from multi-lateration and least-squares regression to multidimensional scaling have been used to take these estimates and place nodes in a coordinate system [9]. For the latter problem, a similar approach has been followed: methods to locally synchronize clocks between neighboring nodes use message transmission and processing latencies and carefully compensate for clock drift and skew [12]; a second class of methods attempts to synchronize clocks across the entire network while minimizing error accumulation at every hop.

Programmability. Programming these tiny sensors because of the platform constraints imposed by form factor and lifetime requirements: energy, processor, memory, and wireless communication bandwidths are all constraints that affect application development. The literature on programming sensor networks has focused on how to relieve the burden of the programmer in dealing with these constraints. Starting from seminal work on event-driven operating systems [5], researchers have moved on to higher-level abstractions that simplify programming. These include specialized abstractions for programming individual nodes such as virtual machines [10], state machines, neighborhood communication abstractions, and so on. More recently, researchers have turned towards specialized programming languages for expressing the behavior of the network as a whole [8]. An important thread in this line of research is the application of database techniques to program sensor networks: this thread is discussed below.

Security and data integrity. More recently, research in sensor networks has turned towards techniques to ensure the integrity of data produced by the sensors and transmitted across the network. Fundamentally, sensors can produce erroneous results, and the data transmitted by sensors can be inadvertently or maliciously corrupted while in transit. Techniques to address these problems have examined encryption and authentication mechanisms for medium access [7] and routing, as well as statistical techniques for detecting outliers in sensor data or for being able to quantify the confidence attributable to a received result.

Data-Centric Techniques in Sensor Networks

An important development in the history of sensor networking has been the emergence of *data-centricity* – the use of programming abstractions that specify attributes or characteristics of data, rather than the nodes at which data may be found. Two views of data-centricity in sensor networks have emerged: the *database view* which views the network as a virtual relational table, and the networking view which views the network as a distributed hash table [14].

More prominently explored, research on the database view has been motivated by three challenges: first, the data being generated by sensors is continuously changing; second, because of quantization effects and variable spatial density, these data are inherently approximate; and third, given the energy constraints, it is infeasible to extract large volumes of data from the network. These constraints have motivated research on approximate, aggregate, continuous queries, on the virtual relational table resulting from data generated by the various sensors. This setting provides ample scope for exploring query optimization techniques. These optimization techniques exploit user-specified bounds on result error, correlations among different types of sensors, or models of the sensor field in order to reduce communication and sensing cost.

The networking view has explored one-shot point and enumeration queries by modeling the sensor network as a distributed hash table. This approach uses random or locality preserving hashing of generated sensor data to a geographic location in a two-dimensional coordinate system. The data are stored at the corresponding location. This approach enables the construction of innovative distributed indexing structures, such as, for example, those that support multidimensional range queries. Overall, this approach tends to have higher communication costs except in regimes with relatively high query rates (such as, for example, queries generated by programs running within the network).

Key Applications

The following are some examples of potential application areas for sensor networks.

1. *Atmospheric*. In this application, sensors measure the concentration of pollutants or carbon dioxide, as well as other atmospheric conditions (temperature, humidity, wind direction and speed). These dense measurements can give a much clearer picture of local variations in atmospheric conditions.
2. *Habitats*. Sensors can be used to measure light, temperature, humidity, and photo-synthetically active radiation across a forest floor or a forest canopy transect. This gives a detailed picture of spatio-temporal variations in these quantities, and can help biologists understand the effect of these variations on local distributions of plant and animal life.
3. *Water*. Sensor nodes deployed on buoys on the surface of the lake or near the seashore and measuring temperature gradients and chlorophyll beneath the lake's surface can help marine biologists study the dynamics of plankton populations. These populations can significantly affect marine life and thereby have huge economic impact.
4. *Soil*. A network of sensors deployed in soil can measure temperature, light, humidity and contaminant flow. Such measurements can aid precision agriculture for improving crop yields and ensuring better land management.
5. *Man-made structures*. Integrity and energy-expenditure of structures. Sensors deployed on buildings and measuring light and temperature conditions, as well as ambient vibrations, can help understand (and control) energy usage in largebuildings as well as assess the integrity of these structures. Similar sensor networks can also be used to measure activity in seismically active areas (e.g., volcanos).

Future Directions

In the first 5–8 years of its existence, sensor networks were driven by a technological push. Advances in miniaturization made it possible to envision networks of small devices, and these advances prompted many of the research directions described above. Now that the potential of sensor networks is well established, the next phase of research has to deliver reliable, manageable systems that provide ease of programmability. Thereafter, advances in sensor networks will be driven by experiences obtained from large-scale deployments.

Cross-references

- ▶ Continuous Queries in Sensor Networks
- ▶ Data Acquisition and Dissemination in Sensor Networks
- ▶ Data Aggregation in Sensor Networks
- ▶ Data Compression in Sensor Networks
- ▶ Data Estimation in Sensor Networks
- ▶ Data Fusion in Sensor Networks
- ▶ Data Storage and Indexing in Sensor Networks
- ▶ Database Languages for Sensor Networks
- ▶ Query Optimization in Sensor Networks

Recommended Reading

1. Akyildiz I., Su W., Sankarasubramaniam Y., and Cayirci E. A survey on sensor networks. IEEE Commun. Mag., 40(8):102–114, 2002.
2. Asada G., Dong T., Lin F., Pottie G., Kaiser W., and Marcy H. Wireless integrated network sensors: low power systems on a chip. In Proc. European Solid State Circuits Conference, 1998.
3. Culler D.E. and Hong W. Wireless sensor networks – introduction. Commun. ACM, 47(6):30–33, 2004.
4. Estrin D., Govindan R., and Heidemann J. Embedding the internet: introduction. Commun. ACM, 43(5):38–41, 2000.
5. Hill J., Szewczyk R., Woo A., Hollar S., Culler D., and Pister K. System archtecture directions for networked sensors. SIGPLAN Not., 35(11):93–104, 2000.
6. Intanagonwiwat C., Govindan R., and Estrin D. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In Proc. 6th Annual Int. Conf. on Mobile Computing and Networking, 2000, pp. 56–67.
7. Karlof C., Sastry N., and Wagner D. TinySec: link-layer encryption for tiny devices. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 162–175.
8. Kothari N., Gummadi R., Millstein T., and Govindan R. Reliable and efficient programming abstractions for wireless sensor networks. In Proc. SIGPLAN Conf. on Programming Language Design and Implementation, 2007, pp. 10–13.
9. Langendoen K. and Reijers N. Distributed Localization in Wireless Sensor Networks: A Quantitative Comparison. Technical Report PDS-2002-003, Technical University, Delft, November 2002.
10. Levis P. and Culler D. Maté: a tiny virtual machine for sensor networks. In Proc. 10th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 2002, pp. 85–95.
11. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. TAG: Tiny AGgregate queries in ad-hoc sensor networks. In Proc. 5th USENIX Symp. on Operating System Design and Implementation, 2002, pp. 131–146.
12. Maróti M., Kusy B., Simon G., and Lédeczi Á. The flooding time synchronization protocol. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 39–49.
13. Rangwala S., Gummadi R., Govindan R., and Psounis K. Interference-aware fair rate control in wireless sensor networks. In Proc. ACM SIGCOMM Symp. on Network Architectures and Protocols, 2006.
14. Woo A., Madden S.,and Govindan R. Networking support for query processing in sensor networks. Commun. ACM, 47(6):47–52, 2004.
15. Woo A., Tong T., and Culler D. Taming the underlying challenges of reliable multihop routing in sensor networks. In Proc. 1st Int. Conf. on Embedded Networked Sensor Systems, 2003.

Sensornet

▶ Sensor Networks

Sentiment Analysis

▶ Opinion Mining

SEQUEL

▶ SQL

Sequence Data Mining

▶ Temporal Data Mining

Sequenced Semantics

MICHAEL H. BÖHLEN[1], CHRISTIAN S. JENSEN[2]
[1]Free University of Bozen-Bolzano, Bolzano, Italy
[2]Aalborg University, Aalborg, Denmark

Definition

Sequenced semantics make it possible to generalize a query language statement on a non-temporal database to a temporal query on a corresponding temporal, interval-timestamped database by applying minor syntactic modifications to the statement that are independent of the particular statement. The semantics of such a generalized statement is consistent with considering the temporal database as being composed of a sequence of non-temporal database states. Sequenced semantics takes into account the interval timestamps of the argument tuples when forming the interval timestamps associated with result tuples, as well as permits the use of additional timestamp-related predicates in statements.

Key Points

A question that has intrigued temporal database researchers for years is how to systematically generalize non-temporal query language statements, i.e., queries on non-temporal databases, to apply to corresponding temporal databases. A prominent approach is to view a temporal database as a sequence of non-temporal databases. Then a non-temporal statement is rendered temporal by applying it to each non-temporal database, followed by integration of the non-temporal results into a temporal result. Sequenced semantics formalizes this approach and is based on three concepts: S-reducibility, extended S-reducibility, and interval preservation. These topics are discussed in turn.

The ensuing examples assume a database instance with three relations:

Employee

ID	Name	VTIME
1	Bob	5–8
3	Pam	1–3
3	Pam	4–12
4	Sarah	1–5

Salary

ID	Amt	VTIME
1	20	4–10
3	20	6–9
4	20	6–9

Bonus

ID	Amt	VTIME
1	20	1–6
1	20	7–12
3	20	1–12

S-Reducibility

S-reducibility states that the query language of the temporally extended data model must offer, for each query q in the non-temporal query language, a *syntactically similar temporal query* q^t that is its natural generalization, i.e., q^t is snapshot reducible to q, and q^t is syntactically identical to S_1qS_2. The goal is to make the semantics of temporal queries easily understandable in terms of the semantics of the corresponding non-temporal queries. The strings S_1 and S_2 are independent of q and are termed *statement modifiers* because they change the semantics of the entire statement q that they enclose.

In the following examples, statements are prefixed with the modifier `SEQ VT` [2]. This modifier tells the temporal DBMS to evaluate statements with sequenced semantics in the valid-time dimension. These examples illustrate that S-reducible statements are easy to write and understand because they are simply conventional SQL statements with the additional prefix `SQL VT`. Writing statements that compute the same results, but without using statement modifiers, can be very difficult [3].

```
SEQ VT SELECT * FROM EMPLOYEE;
SEQ VT
  SELECT ID
  FROM EMPLOYEE AS E
  WHERE NOT EXISTS (
    SELECT *
    FROM SALARY AS S
    WHERE E.ID = S.ID);
```

The first query returns all `Employee` tuples together with their valid time – this corresponds to returning the content of `Employee` at each state. The second query determines the time periods when an employee did not get a salary. It returns $\{\langle 3, 1{-}3\rangle, \langle 3, 4{-}5\rangle, \langle 3, 10{-}12\rangle, \langle 4, 1{-}5\rangle\}$. Conceptually the enclosed statement is evaluated on each state of the database. Computationally, the interval 6–9 is subtracted from the interval 4–12 to get the intervals 4–5 and 10–12.

Extended S-Reducibility

S-reducibility is applicable only to queries of the underlying non-temporal query language and does not extend to queries with explicit references to time. Consider the following queries.

```
SEQ VT
  SELECT E.ID
  FROM Employee AS E, Salary AS S
  WHERE E.ID = S.ID
  AND DURATION(VTIME(E)) >
  DURATION(VTIME(S));
```

```
SEQ VT
  SELECTE.ID,VTIME(S), VTIME(E)
  FROMEmployeeAS E, Salary AS S
  WHEREE.ID = S.ID;
```

The first query constrains the temporal join to tuples in `Employee` with a valid time that is longer than the valid time of the salary tuple it shall be joined with. This condition cannot be evaluated on individual non-temporal relation states because the timestamp is not present in these states. Nevertheless, the temporal join itself can still be conceptualized as a non-temporal join evaluated on each snapshot, with an additional predicate. The second query computes a temporal join as well, but also returns the original valid times. Again, the semantics of this query fall outside of snapshot reducibility because the original valid times are not present in the non-temporal relation states.

DBMSs generally provide predicates and functions on time attributes, which may be applied to, e.g., valid time, and queries such as these arise naturally. Applying sequenced semantics to statements that include predicates and functions on time offers a higher degree of orthogonality and wider ranging temporal support.

Interval Preservation

Coupling snapshot reducibility with syntactical similarity and using this property as a guideline for how to semantically and syntactically embed temporal functionality in a language is attractive. However, S-reducibility does not distinguish between different relations if they are snapshot equivalent. This means that different results of an S-reducible query are possible: the results will be snapshot equivalent, but will differ in how the result tuples are timestamped. As an example, consider a query that fetches and displays the content of the `Bonus` relation. An S-reducible query may return the result {⟨1, 20, 1−6⟩, ⟨1, 20, 7−12⟩, ⟨3, 20, 1−12⟩}. If Bob received a 20K bonus for his performance during the first half of the year and another 20K bonus for his performance during the second half of the year and Pam received a 20K bonus for her performance during the entire year, this is the expected result. This is also the result supported by the three tuples in the example instance displayed above. However, S-reducibility does not distinguish this result from any other snapshot equivalent result. With S-reducibility a perfectly equivalent result would be {⟨1, 20, 1−12⟩, ⟨3, 20, 1−6⟩, ⟨3, 20, 7−12⟩}.

Interval preservation settles the issue of which result should be favored out of the many possible results permitted by S-reducibility. When defining how to timestamp tuples of query results, two possibilities come to mind. Results can be coalesced. This solution is attractive because it defines a canonical representation for temporal relations. A second possibility is to consider lineage and preserve, or respect, the timestamps as originally entered into the database [1]. Sequenced semantics requires that the default is to preserve the timestamps – being irreversible, coalescing cannot be the default.

Cross-references

▶ Nonsequenced Semantics
▶ Snapshot Equivalence
▶ Temporal Coalescing
▶ Time Interval
▶ Valid Time

Recommended Reading

1. Böhlen M.H., Busatto R., and Jensen C.S. Point- versus interval-based temporal data models. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 192–200.
2. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. ACM Trans. Database Syst., 25(4):48, 2000.
3. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, 1999.

Sequential Patterns

Jianyong Wang
Tsinghua University, Beijing, China

Synonyms

Frequent subsequences

Definition

A *sequence database* $D = \{S_1, S_2, \ldots, S_n\}$ for sequential pattern mining consists of n input sequences (where $n \geq 1$), and an *input sequence*

$S_i = \langle e_{i_1}, e_{i_2},\ldots,e_{i_m}\rangle (1 \le i \le n)$ is an ordered list of m events (where $m \geq 1$). Each *event* e_{i_j} $(1 \le i \le n, 1 \le j \le m)$ is a non-empty set of items. Given two sequences, $S_a = \langle e_{a_1}, e_{a_2},\ldots,e_{a_k}\rangle$ and $S_b = \langle e_{b_1}, e_{b_2},\ldots,e_{b_l}\rangle$, if $k \le l$ and there exist integers $1 \leq x_1 < x_2 < \ldots < x_k \leq l$ such that $e_{a_1} \subseteq e_{b_{x_1}}, e_{a_2} \subseteq e_{b_{x_2}},\ldots,e_{a_k} \subseteq e_{b_{x_k}}$, S_b is said to *contain* S_a (or equivalently, S_a is said to be contained in S_b). The number of input sequences in D that contain sequence S is called the *support* of S in D, denoted by $sup^D(S)$. Given a user-specified minimum support threshold *min_sup*, S is called a *sequential pattern* (or a *frequent subsequence*) in D if $sup^D\ (S) \geq min\_sup$. If there exists no proper supersequence of a sequential pattern S with the same support as S, S is called a *closed sequential pattern* (or a *frequent closed subsequence*) in D. Furthermore, a sequential pattern S is called a *maximal sequential pattern* (or a *frequent maximal subsequence*) if it is not contained in any other sequential pattern. The problems of *sequential pattern mining*, *closed sequential pattern mining*, and *maximal sequential pattern mining*, are to find all frequent subsequences, all frequent closed subsequences, and all frequent maximal subsequences from input sequence database D, respectively, given a user-specified minimum support threshold *min_sup*.

Historical Background

Similar to association rule mining, sequential pattern mining was initially motivated by the decision support problem in retail industry and was first proposed by Rakesh Agrawal and Ramakrishnan Srikant in [1]. Later on, it was applied to other domains. Some recent research work further validated its utility in various applications, such as identifying outer membrane proteins, automatically detecting erroreous sentences, discovering block correlations in storage systems, identifying copy-paste and related bugs in large-scale software code, API specification mining and API usage mining from open source repositories, frequent subsequence-based XML document clustering, sequence-based XML query pattern mining for effective caching, and Web log data mining.

In the seminal paper on sequential pattern mining, three algorithms were introduced [1]. Among these algorithms, AprioriSome and DynamicSome were proposed for mining maximal sequential patterns, while AprioriAll was designed for mining all sequential patterns. The same authors later generalized the sequential pattern mining problem by allowing time constraints, sliding time window, and taxonomies, and proposed a new algorithm, GSP [10]. The inefficiency of AprioriAll mainly stems from its computationally expensive data transformation operation which transforms each transaction to a set of frequent itemsets in order to find sequential patterns. As GSP overcomes the drawbacks of AprioriAll, it is much faster than AprioriAll. In [4], Jiawei Han et al. proposed the FreeSpan algorithm, which shows better performance than GSP. In [15], Mohammed J. Zaki adopted the vertical data representation for sequential pattern mining and devised an efficient algorithm, SPADE, which fully exploits the lattice search techniques and some join operations. Another state-of-the-art sequential pattern mining algorithm is PrefixSpan, which was proposed by Jian Pei et al. [8]. It adopts a projection-based, sequential pattern growth approach to avoiding the traditional candidate-generation-and-test paradigm, thus improves the algorithm efficiency. In [3], Jay Ayres et al. designed another sequential pattern mining algorithm, SPAM. This algorithm integrates a depth-first search strategy with some effective pruning techniques, uses a vertical bitmap data representation, and can incrementally output frequent subsequences in an online fashion.

All the preceding algorithms except AprioriSome and DynamicSome mine the complete set of sequential patterns. One problem with sequential pattern mining is that it may generate too many redundant patterns, which also impedes the algorithm efficiency. One popular solution to this problem is to mine closed sequential patterns only, which usually leads to not only a more compact yet complete result set but also better efficiency. In [14], Xifeng Yan and Jiawei Han presented the CloSpan approach, which incorporates several effective pruning methods into the PrefixSpan framework and achieves much better performance than PrefixSpan. Recently, another closed sequential pattern mining algorithm, BIDE, was proposed in [12] by Jianyong Wang et al. It integrates a new pattern closure checking scheme and a new pruning technique with the PrefixSpan framework, and is both runtime and memory efficient.

Foundations

The biggest challenge faced by sequential pattern mining is the combinatorial explosion problem. To alleviate this problem, researchers have tried various ways. In the following some factors which may have an impact on the efficiency of a sequential pattern mining (or closed sequential pattern mining) algorithm are summarized. These factors mainly include the data representation format, pattern enumeration framework (i.e., search strategy), search space pruning techniques, and pattern closure checking scheme.

Sequence Data Format

The input sequence data can be represented in two alternative formats. The *horizontal representation* is a natural bookkeeping of the input sequences. Each sequence consists of an ordered list of events, while each event is recorded as a list of items (which are supposed in most algorithms to be sorted according to a certain order, say the lexicographical order). AprioriAll, GSP, PrefixSpan, CloSpan, and BIDE are typical examples adopting the horizontal representation. Note that AprioriAll needs to first find the frequent itemsets and transform each event to the set of frequent itemsets contained in the event in order to find sequential patterns. To compute the support of a subsequence, the horizontal format based algorithms need to scan the database, which is computationally expensive. To assist support counting, these algorithms devise some special mechanisms. For example, GSP introduces the hash-tree data structure, while PrefixSpan, CloSpan, and BIDE use a projection-based approach to shrink the part of database that needs to be scanned.

In the *vertical representation*, the database is represented as a set of items, where each item is recorded as a set of pairs of sequence identifier (SID) and event identifier (EID) containing the item. SPADE and SPAM use the vertical format. With the vertical representation, the support-counting can be performed using simple join operations with temporal ordering constraint. To improve the efficiency of support counting, SPAM proposes to use vertical bitmaps to represent the sequence database. Each item is converted to a bitmap, which has a bit for each event in the database. If an event contains an item, the corresponding bit regarding the event and item is set to one; otherwise, it is set to zero. Based on the transformed bitmap sequence representation, efficient support counting can be easily achieved using bitwise AND operations of bitmaps.

Search Strategy

Given an input sequence database D and a minimum support threshold *min_sup*, the set of sequential patterns are deterministic, and can be organized into a lexicographic frequent sequence tree structure. Suppose *min_sup*=2, database D contains three input sequences, ⟨(A B D)(B C D)(A)⟩, ⟨(B) (A B E) (B)⟩, and ⟨(A B) (B C D)⟩ (here the commas separating each pair of adjacent events in the same sequence are omitted), respectively, and there exists a lexicographic ordering among the set of distinct items A ≤B ≤C ≤D ≤E. For a prefix subsequence S_p, it can be extended in two ways, namely, sequence-extension and itemset-extension. The *sequence-extension* extends S_p by a new event containing a single item, while *itemset-extension* adds a new item to the last event of S_p and the new item must be lexicographically larger than any item of the last event of S_p. Assume both sequence-extension and itemset-extension are performed in lexicographic ordering and itemset-extension is performed before sequence-extension for the same prefix sequence. Then, the lexicographic sequence tree structure of the frequent subsequences in the running example is shown in Fig. 1. Note that each node in the tree shows a sequential pattern and its corresponding support (i.e., the number after the colon), and all the patterns at the same level have the same length (namely, they contain the same number of items). An edge which links a parent node P at level k to a child node C at level (k+1) indicates that the pattern at C is directly extended from the prefix pattern at node P, either by sequence-extension or by itemset-extension.

Once the sequential patterns are organized into a lexicographic tree structure, one can choose a tree traversal strategy for sequential pattern mining. The two popular search strategies are breadth-first search and depth-first search. In the *breadth-first search* method, frequent subsequences are mined in a level-wise manner, that is, before mining patterns with length (k+1), one needs to first mine all patterns with length k. In contrast, a *depth-first search* method traverses the sequence tree in depth-first order. GSP adopts the breadth-first search strategy to enumerate the sequential patterns, PrefixSpan, SPAM, CloSpan, and BIDE choose the depth-first search paradigm, while SPADE supports both breadth-first search and depth-first search methods.

Search Space Pruning

One of the most crucial optimization considerations to improve the efficiency of a data mining algorithm is to devise some effective search space pruning techniques. Based on some heuristics, if some parts of the search space are already known to be futile in generating sequential patterns, they should be found and pruned as quickly as possible. Perhaps the most well-known property used for designing pruning methods in frequent pattern mining is the *Apriori* property (also known as the downward closure property or anti-monotone property). It states in the sequence mining setting that all the subsequences of a frequent sequence must be also frequent, or equivalently, a sequence must be infrequent if it contains an infrequent subsequence. All the existing sequential pattern mining algorithms have exploited the Apriori property in different ways.

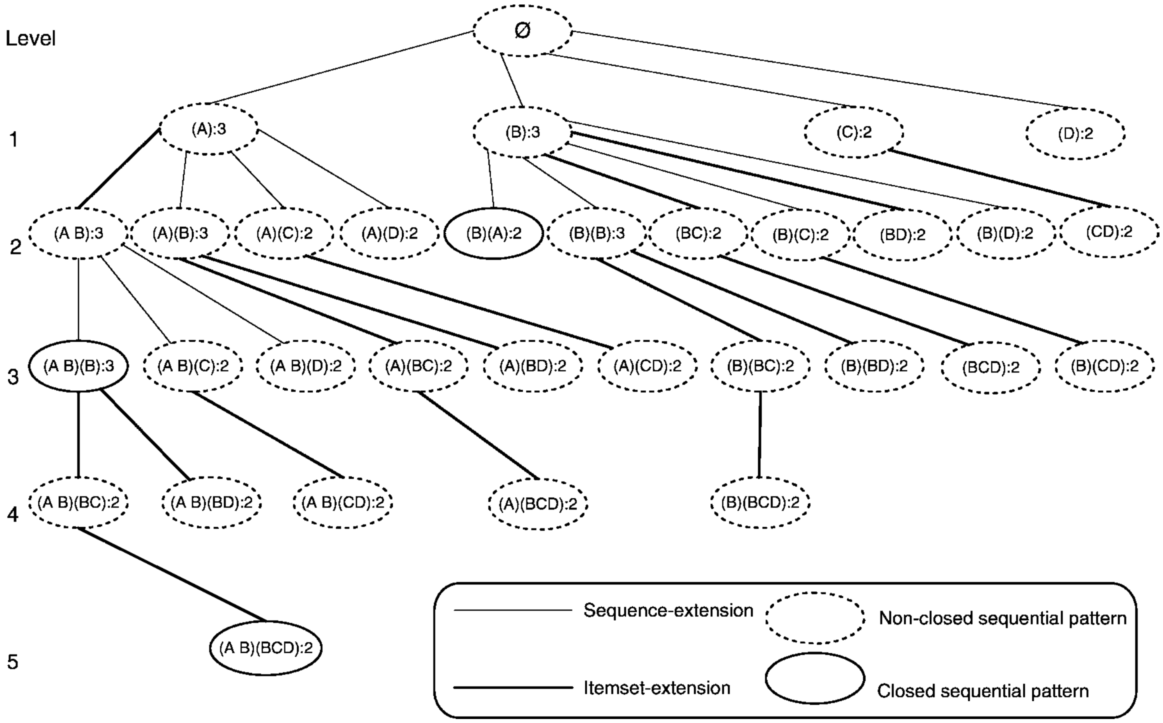
From Fig. 1 one can see that only three out of the 31 frequent subsequences are closed, namely, (B)(A):2, (AB)(B):3, and (AB)(BCD):2, and many subtrees in Fig. 1 contain no closed sequential pattern. An efficient closed sequential pattern mining algorithm should avoid traversing the subtrees containing no closed patterns, which leaves room to further prune the search space. Both CloSpan and BIDE adopt some optimization techniques. The pruning methods proposed in CloSpan are listed as follows.

- *Common prefix pruning*: if there exists a common prefix, all sequences beginning with a proper subsequence of this prefix cannot be closed.
- *Partial-order pruning*: if an item “a” always occurs before item “b” in all sequences, any sequence beginning with “b” cannot be closed.
- *Equivalent projected DB pruning*: given two subsequences, s and s′, where s is a proper subsequence of s′ and they have equivalent projected database, any sequence beginning with s cannot be closed.

The concepts of common prefix, partial-order, and equivalent projected database can be found in [7]. The BIDE algorithm proposes a single but effective pruning technique called *BackScan* search space pruning. Given a prefix S_p, if ∃ i (i is a positive integer and is no greater than the length of S_p) and there exists any item that appears in each of its ith semimaximum periods, S_p can be safely pruned. The interested readers are referred to [12] for more details.

Pattern Closure Checking Scheme

The optimization methods proposed in CloSpan and BIDE are very effective in pruning the unpromising parts of the search space, however, they cannot assure that each discovered sequential pattern is closed. For closed sequential pattern mining algorithms, one still



Sequential Patterns. Figure 1. The lexicographic sequence tree structure of the frequent subsequences in the running example.

needs to devise some methods to check if a sequential pattern is closed or not. In CloSpan, all the candidate closed sequential patterns are maintained in a tree data structure. CloSpan eliminates the non-closed patterns in a post-processing phase and adopts the hashing technique to accelerate the pattern closure checking. When the number of candidate sequential patterns is large, the pattern tree structure may consume non-trivial space. In [12], BIDE adopts a so-called *BI-Directional Extension* closure checking scheme. One big advantage of this new scheme is that it avoids maintaining the set of candidate closed sequential patterns, and thus saves space. If there exists no *forward-S-extension* item, *forward-I-extension* item, *backward-S-extension* item, nor *backward-I-extension* item with respect to a prefix sequence S_p, S_p is a closed sequence, otherwise, S_p must be non-closed. The definitions of a forward-S-extension item, a forward-I-extension item, a backward-S-extension item, and a backward-I-extension item can be found in [8].

Key Applications

In recent years sequential pattern mining witnessed many applications, which roughly fall into the following categories.

Frequent Subsequence-based Classifier

In [9], the authors used an efficient implementation of generalized suffix tree to mine a set of frequent subsequences with a minimum length constraint, and built the rule-based classifier and SVM classifier based on the discovered frequent subsequences. Their performance results demonstrate that the frequent subsequence-based classifier achieves high accuracy in identifying outer membrane proteins. Recently the authors of [11] proposed a method to build associative classification rules from frequent subsequences returned by a variant of the PrefixSpan algorithm. Their performance study shows that sequence-based classification rules are very helpful in automatically detecting erroreous sentences.

Operating System and Software Engineering

In [5,6], the authors adopted the CloSpan algorithm to mine closed sequential patterns, which have been shown very useful in discovering block correlations in storage systems and identifying copy-paste and related bugs in large-scale software code. In [7,13], the BIDE algorithm was used to discover the set of frequent closed subsequences with the purpose of API specification mining and API usage mining from open source repositories.

Frequent Subsequence-based XML Data Management

Sequential pattern mining has also been widely applied in semi-structured data management. In this application, an XML document is first converted to a sequence instead of a tree structure. Then some kinds of constrained frequent subsequences are mined, which can be exploited to accelerate XML query or XML document clustering. The performance results in [2] demonstrate that the frequent subsequence-based XML clustering algorithm XProj achieves better clustering quality than previous algorithms.

Web Log Data Mining

Some researchers have also applied sequential pattern mining algorithms in mining salient patterns (e.g., contiguous sequential patterns) from Web log data.

Experimental Results

Some experimental results on sequential pattern mining can be found in [3,8,10,15], which compares the efficiency of some typical sequential pattern mining algorithms including GSP, SPADE, PrefixSpan, and SPAM, while [14] and [12] present the performance study of two closed sequential pattern mining algorithms, CloSpan and BIDE.

Data Sets

The IBM synthetic dataset generator can generate sequence datasets and can be found from the link of http://www.cs.rpi.edu/∼zaki/software/. The popularly used real sequence datasets include some Web log data, and protein sequence datasets.

URL to Code

The code for PrefixSpan and CloSpan algorithms can be found from the Illini Mine portal, http://dm1.cs.uiuc.edu/protected/im, the code for SPADE algorithm can be downloaded from the link of http://www.cs.rpi.edu/∼zaki/software/, while the code for BIDE algorithm can be traced from http://dbgroup.cs.tsinghua.edu.cn/wangjy/.

Cross-references

- ► Apriori Property and Breadth-First Search Algorithms
- ► Closed Itemset Mining and Non-redundant Association Rule Mining

- ▶ Frequent Graph Patterns
- ▶ Frequent Itemsets and Association Rules
- ▶ Frequent Partial Orders
- ▶ Pattern-Growth Methods

Recommended Reading

1. Agrawal R. and Srikant R. Mining sequential patterns. In Proc. 11th Int. Conf. on Data Engineering, 1995.
2. Aggarwal C.C., Ta N., Wang J., Feng J., and Zaki M.J. XProj: a framework for projected structural clustering of XML documents. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007.
3. Ayres J., Gehrke J., Yiu T., and Flannick J. Sequential pattern mining using a bitmap representation. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002.
4. Han J., Pei J., Mortazavi-Asl B., Chen Q., Dayal U., and Hsu M.C. FreeSpan: frequent pattern-projected sequential pattern mining. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000.
5. Li Z., Chen Z., Srinivasan S., and Zhou Y. C-Miner: mining block correlations in storage systems. In Proc. 3rd USENIX Conf. on File and Storage Technologies, 2004.
6. Li Z., Lu S., Myagmar S., and Zhou Y. CP-Miner: finding copy-paste and related bugs in large-scale software code. IEEE Trans. Software Eng., 32(3):176–192, 2006.
7. Lo D. and Khoo S.C. SMArTIC: towards building an accurate, robust and scalable specification miner. In SIGSOFT FSE., 2006.
8. Pei J., Han J., Mortazavi-Asl B., Pinto H., Chen Q., Dayal U., and Hsu M.C. PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern-growth. In Proc. 17th Int. Conf. on Data Engineering, 2001.
9. She R., Chen F., Wang K., Ester M., Gardy J.L., and Brinkman F.S.L. Frequent-subsequence-based prediction of outer membrane proteins. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003.
10. Srikant R. and Agrawal R. Mining sequential patterns: generalizations and performance improvements. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996.
11. Sun G., Liu X., Cong G., Zhou M., Xiong Z., Lee J., and Lin C.Y. Detecting erroreous sentences using automatically mined sequential patterns. In Proc. 45th Annual Meeting of the Assoc. for Computational Linguistics, 2007.
12. Wang J., Han J., and Li C. Frequent closed sequence mining without candidate maintenance. IEEE Trans. Knowl. Data Eng., 19(8):1042–1056, 2007.
13. Xie T. and Pei J. Data mining for software engineering. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006.
14. Yan X., Han J., and Afshar R. CloSpan: mining closed sequential patterns in large databases. In Proc. SIAM International Conference on Data Mining, 2003.
15. Zaki M.J. SPADE: an efficient algorithm for mining frequent sequences. Machine Learn., 42:31–60, 2001.

Serializability

K. Vidyasankar
Memorial University of Newfoundland, St. John's,
NL, Canada

Synonyms

Correctness criterion for concurrent executions

Definition

A database is a collection of data items operated on concurrently by several programs. A set of values for the data items is called a *database state*. It is *consistent* if the values satisfy the *integrity constraints* specified for the database. Arbitrary interleaving of the executions of the programs may affect the consistency of the database. The executions that preserve the consistency are called *correct* executions. The notion of serializability helps to identify correct executions. It is based on the transaction concept: *a transaction* is a partially ordered set of atomic steps that constitute an execution of a program, with the property that, when executed alone, it transforms a consistent database state into another consistent one. Examples of atomic steps are read and write of a data item, increment and decrement on a counter object, enqueue and dequeue on a queue object, etc. The sequence of steps in an execution is called a *history*. A *serial* history corresponds to a serial, that is, non-interleaving, execution of the transactions. Any (concurrent) execution whose effect is "equivalent" to that of some serial execution of the same set of transactions is called *serializable*. Clearly, any serial execution of a set of transactions preserves consistency, and hence is correct. Therefore serializable executions are also correct. Serializability theory concerns finding properties of histories which are serializable. This knowledge is useful for designing schedulers that will allow only those concurrent executions that are correct.

Historical Background

After the initial exposure to the problem in [2], several characterizations of serializability were obtained, for example, in [2,6,9,11], and the theory was extended in several directions.

One main direction was defining different notions of serializability corresponding to different interpretations of 'equivalence' to serial histories. Examples are *view-serializability*, where the values read by all the transactions and the final values of all the data items are the same as in some serial execution, *state-serializability* where only the final values of the data items need be the same and the general *S-serializability* where only the values read by a specified subset S of the transactions need be the same.

Another main direction was the study of serializability under some constraints [11]. Here, the transaction order in an equivalent serial history must satisfy certain constraints which are usually derived from some syntactic properties of the histories. In general, checking whether a history is serializable is NP-complete. Constraints help to identify subclasses of serializable histories that have a polynomial membership test.

A third direction dealt with relaxing the unit of atomicity. Instead of taking all the steps of a transaction as forming a single unit preserving consistency of the database, groups of several subsets of the steps were considered as atomic units, and interleaving constraints were applied to them. Also, the units of atomicity were allowed to be different for different transactions, giving rise to the *relative atomicity* notion. Hierarchical grouping of the steps led to the study of *nested transactions* [8]. Non-hierarchical groupings were studied, for example, in [3,13].

Another extension was the study of *multi-version serializability* [10]. Here, the value written by each write step of the transactions is kept as a separate version, and a read operation can read any of the versions of the corresponding data item. This notion allows more concurrent executions as correct ones compared to the single version serializability.

Orthogonal developments have been the study and applicability of serializability theory in different architectures (from centralized to distributed databases, multi-databases, mobile environments and recently Internet) and with different objects (from simple data items to complex objects, *B*-trees, design objects, workflows, business processes and Web services). Executions that "slightly" deviate from being serializable and hence are "almost correct" have also been studied. Serializability theory initially focused only on concurrency. Later, recovery aspects were combined and unified theories studied. Details and references can be found in [15].

Foundations

Serializability Characterization

In the following, a simple transaction model is considered where (i) each atomic step is either a *read step* or a *write step*, that reads or writes the value of a data item, respectively, and (ii) each data item is accessed by at most one read step and at most one write step in a transaction, and if both these steps do occur, then the read step occurs before the write step. The theory can easily be extended to other transaction models, and with other atomic steps.

A transaction is *write-only* if it does not have any read steps, and *read-only* if it does not have any write steps. Let D be the set of data items in a database, and $\{T_1,\ldots,T_n\}$ be a set of transactions. For convenience, a hypothetical write-only initial transaction T_0 which writes the initial values of all the data items, and a hypothetical read-only final transaction T_f which reads the values of all the data items after all the transactions have completed are added. The transaction T_0 helps to set up a consistent initial database state, and T_f helps to argue about the final database state. Let T be $\{T_0, T_1,\ldots,T_n, T_f\}$. A read step (write step) of transaction T_i reading (writing) the data item X is denoted $R_i[X](W_i[X])$. A set of read steps, unrelated by the partial order, reading a subset C of D and occurring together, is denoted $R_i[C]$. For example, $R_i[\{X, Y\}]$ denotes the unrelated read steps $R_i[X]$ and $R_i[Y]$ occurring together in any order; for brevity, it is written as $R_i[X, Y]$, ignoring the curly brackets. Similar notation is followed for the write steps.

A *history* h of T is a sequence of the steps of T representing the execution of the transactions in a possibly interleaved fashion, starting with $W_0[D]$ and ending with $R_f[D]$. A history is *serial* if there is no interleaving, that is, once a transaction starts executing, it finishes without any other transaction executing some step in between. A transaction T_j reads X from transaction T_i in a history h if $W_i[X]$ is the last write X step before $R_j[X]$ in h. The *reads-from* relation *rf* of a history h is defined as: $rf(h) = \{(T_i,X,T_j)$: T_j reads X from $T_i\}$. Two histories h and h' (of the same set of transactions) are *equivalent* if $rf(h) = rf(h')$. A history h is *serializable* if there is a serial history h' equivalent to h. (This is the *view-serializability* notion [11].)

Example: Consider the history h_1, for $D = \{X, Y\}$: $W_0[D]R_1[X]R_2[X]R_3[Y]W_2[Y]W_3[Y]W_4[X,Y]R_f[D]$. The reads-from relation $rf(h_1)$ is $\{(T_0, X, T_1),(T_0, X, T_2),(T_0, Y, T_3),(T_4, X, T_f),(T_4, Y, T_f)\}$. The history h_1 has the same reads-from relation as the following serial history h_2, corresponding to the serial order $(T_0, T_1, T_3, T_2, T_4, T_f)$: $W_0[D]R_1[X]R_3[Y]W_3[Y]R_2[X]W_2[Y]W_4[X,Y]R_f[D]$ and hence is serializable.

A write X step of transaction T_i (also the value that is written) is *useless* in h if no transaction reads this value, that is, there is no T_j for which (T_i, X, T_j) is in $rf(h)$; otherwise, it is *useful.*

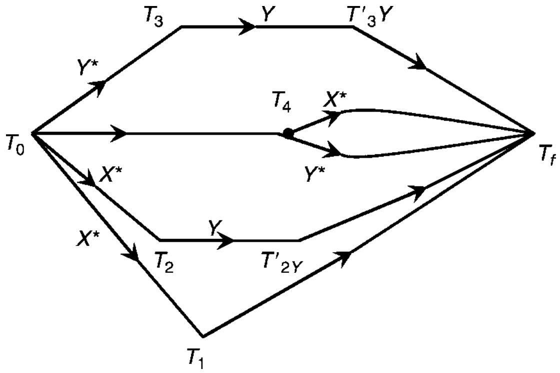
The *history graph* of h, denoted $H(h)$, is a directed graph constructed as follows.

The vertex set of $H(h)$ is $T \cup T'$ where $T' = \{T'_{iX} : T_i$ has a useless write X in $h\}$.
The set T' is a set of dummy transactions, one for each useless write in h.
The edge set of $H(h)$ has the following:

- An edge labeled X from T_i to T_j, for each (T_i, X, T_j) in $rf(h)$
- An edge labeled X from T_i to T'_{iX} for each useless write X of T_i, for all T_i
- An unlabeled edge from each vertex in T' to T_f
- An unlabeled edge from each read-only transaction other than T_f to T_f
- An unlabeled edge from T_0 to each write-only transaction other than T_0

The transaction name itself is used to denote the vertex corresponding to that transaction in the graph. An edge α from T_i to T_j is denoted (T_i, T_j). Here, T_i is the *positive end* $p\alpha$ of α, and T_i is the *negative end* $n\alpha$ of α. The edge α is also referred to as an *outdirected* edge of T_i and an indirected edge of T_j. A *source* is a vertex with no indirected edges, and a *sink* is a vertex with no outdirected edges. An *X-edge* refers to an edge labeled X. The labeled edges incident to T' are useless; all other labeled edges are useful.

The history graph for h_1 is given in Fig. 1. In the graph, useful edges are indicated by a star (*).



Serializability. Figure 1. History graph $H(h_1)$.

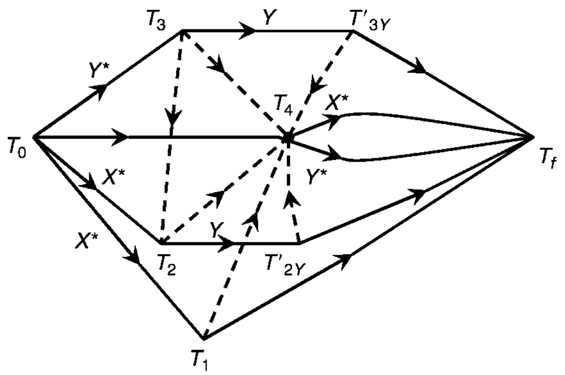
Clearly, two histories h and h' are equivalent iff $H(h) = H(h')$. It is easy to verify that the history graph of a serial history is acyclic. Hence, the history graph of a serializable history will also be acyclic; any topological sort of the vertices will give a serial order of the transactions in an equivalent serial history. (The dummy transactions are ignored.) However, $H(h)$ being acyclic is not sufficient for h to be serializable. For example, the history $W_0[X]R_1[X]R_2[X]W_2[X]W_1[X]R_f[X]$ gives an acyclic history graph, but neither of the two possible topological sorts, (T_0,T_1,T_2,T_f) and (T_0,T_2,T_1,T_f), gives a serial history which has the same reads-from relation as the original one. Thus for h to be serializable, $H(h)$ must satisfy some other property in addition to being acyclic. That property is characterized in the following, after some more terminology is introduced.

The *coboundary* δS of a subset S of the vertex set of a graph G is the set of edges each with one end in S and the other not in S. By a *coboundary of G* is meant a set of edges that is a coboundary of some subset of the vertex set of G. A *cutset* is a minimal nonnull coboundary. (A set is *minimal* with a given property if it has that property but none of its proper subsets has that property.) An edge α of a coboundary δS is *outdirected* if $p\alpha$ is in S; it is *indirected* if $n\alpha$ is in S. A coboundary (cutset) δS is *outdirected* if all its edges are outdirected, and similarly it is *indirected* if all its edges are indirected; in either case, it is *directed.* For a data item X in D, a cutset is *X-unsafe* if it is directed and it contains a useful X-edge α and another (useful or useless) X-edge β such that $p\beta \neq p\alpha$. A cutset is *unsafe* if it is X-unsafe for some X in D. Any cutset which is not unsafe is *safe.* Note that all undirected cutsets are safe.

A *transaction precedence graph* of h, denoted $TP(h)$, is a graph that contains (i) $H(h)$ and possibly some additional, unlabeled, edges and (ii) no unsafe cutsets.

Theorem: *A history h is serializable iff there exists an acyclic TP(h)* [12].

A $TP(h)$ can be obtained from $H(h)$ by repeatedly applying the following construction: for each X-unsafe cutset with useful X-edge α and another X-edge β, add an unlabeled edge from $n\alpha$ to $p\beta$, or from $n\beta$ to $p\alpha$, thus making this cutset undirected. Either of these edges is an *exclusion edge.* A transaction precedence



Serializability. Figure 2. Transaction precedence graph *TP* (h_1).

graph for the history graph in Fig. 1 is shown in Fig. 2. The added exclusion edges are shown in broken lines. For instance, in H (h_1), $\delta\{T_0, T_2\}$ is Y -unsafe: it is outdirected with edges $\{(T_0, T_3),(T_0, T_4),(T_0, T_1), (T_2, T'_{2Y})\}$, it has one useful Y-edge (T_0, T_3) and one useless Y-edge (T_2, T'_{2Y}). The exclusion edge (T_3, T_2) makes the cutset undirected and therefore safe. The cutset $\delta\{T_0,T_2,T_3\}$ is directed and has two Y -edges, but is safe since both Y-edges are useless.

Note that, for each unsafe cutset, at most one exclusion edge can be added, to get an acyclic *TP* (h). (Adding both edges will create a directed cycle.) Thus there are two choices for an exclusion edge. If there are n unsafe cutsets, then there are 2^n choices for adding n exclusion edges. With each such choice, the resulting graph can be checked for acyclicity in polynomial time. However, because of the number of choices, checking whether there exists an acyclic *TP* (h) for a given H (h), in brute-force manner, will take exponential time. It has been shown that a better time is unlikely. That is, checking whether a history is serializable is an NP-complete problem. Hence, serializability under some additional constraints have been studied so as to get polynomial membership test. The constraints limit the serial histories which can be taken as correct executions.

Another Serializability Characterization

First, another characterization of serializability that depicts the nature of the constraints for polynomial membership test is given. It is based on the following property:

In an acyclic *TP* (h), for any set Ψ of useful X-edges no two of which have the same positive end,

(a) There is a directed path that contains all the edges of Ψ.
(b) For each useless X-edge β,there is a directed path that contains β and all the edges of Ψ.

In the following, a transaction that writes X will be called an X-writer. It is *useful* (*useless*) X-writer if the write X step is useful (useless) in h. Note that for some X and Y, the same transaction could be a useless X-writer but a useful Y -writer. The property instrumental to the characterization is the following:

In an acyclic *TP* (h), for each X in D,

(a) There is a directed path from T_0 that contains all the useful X-writers.
(b) For each useless X-writer T_a , there is a directed path from T_0 that contains all the useful X-writers and T_a.

A *weak order* on X-writers in h is an irreflexive partial order such that

(a) It totally orders all the useful X-writers and T_0.
(b) For any useless X-writer T_a, it totally orders all useful X-writers, T_0 and T_a.

A *weak order* in h is the union of weak orders on X-writers in h, one for each X in D.

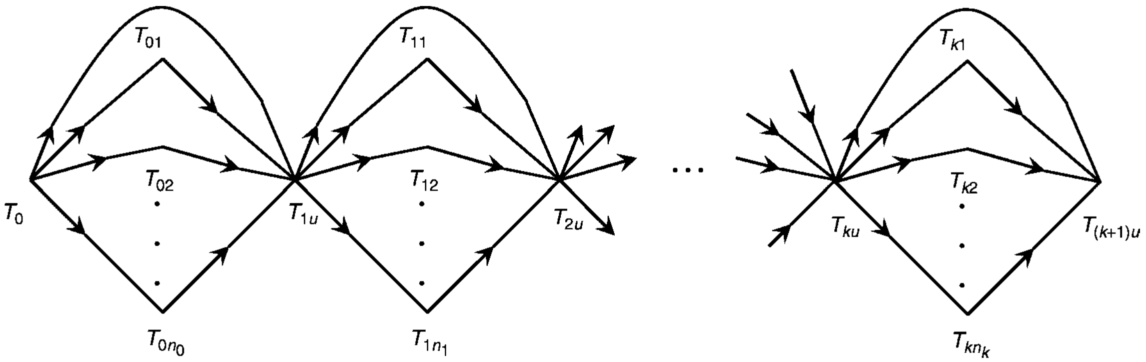
Figure 3 depicts a weak order on X-writers for some X. Here, T_{1u},T_{2u},... are useful X-writers,T_0 may or may not be useful, and all others are useless X-writers. A weak order q can be added to *TP* (h) by means of unlabeled edges, one for each element of q. The resulting graph is called $TP[q](h)$.

Theorem. *A history h is serializable iff for some weak order q in h there exists an acyclic TP[q](h)* [12].

For any weak order q, in each unsafe cutset of H (h) with useful X-edge α and another X-edge β, the weak order edges will introduce a directed path between $p\alpha$ and $p\beta$ and this will induce a directed cycle with one of the exclusion edges; therefore only the other exclusion edge can possibly be added to get an acyclic $TP[q](h)$ graph. Acyclicity of a graph can be checked in polynomial time. Therefore, checking whether there exists an acyclic $TP[q](h)$, for a given q, can be done in polynomial time.

Serializability under Constraints

For read or write steps O and O', OO'-constraints are defined as: if $O_i[X]$ of transaction T_i occurs before $O'_j[X]$ of transaction T_j in h, for some X in D, then



Serializability. Figure 3. Weak order on *X*-writers.

T_i must be serialized before T_j. Depending on whether each of these steps is a read (R) or a write (W), the constraints are classified as *WW*-, *WR*-, *RW*- and *RR-constraints*. In addition, *WRW-constraints* are defined as the union of *WR*- and *RW*-constraints. It turns out that *WRW*-constraints impose exactly a weak order *q*. *WW*-constraints are more stringent than *WRW*-constraints; they impose a total order on all the *X*-writers whereas the latter impose a total order only on useful *X*-writers. Hence there are histories which are serializable under *WRW*-constraints but not under *WW*-constraints. For histories in which there are no useless writes (on any data item), *WRW*-constraints are equivalent to *WW*-constraints. Serializability under *WRW*-constraints, and similarly *WW*-constraints, can be checked in polynomial time. It turns out that *WR*- and *RW*-constraints, individually, do not induce a weak order on *h*. They leave the choices of exclusion edges for some unsafe cutsets open, and checking membership under each of them is NP-complete.

Serializability under *WW*-constraints corresponds exactly to serializability under the union of *WW*-, *WR*- and *RW*-constraints. It also corresponds exactly to *conflict preserving serializability CPSR* [1], also called *conflict serializability* [15]. Here, the constraints are stated in terms of conflicts: two steps *conflict* if they are on the same data item and at least one of them is a write, and two transactions *conflict* if they have conflicting steps.

A constraint that is not based on conflicts is: if a transaction T_i finishes execution before a transaction T_j starts, in the given history *h*, then T_i should appear before T_j in an equivalent serial order. The histories serializable under these constraints are called *strictly serializable* (also *order-preserving serializable* [1,9]).

Different Notions of Serializability

In the serializability definition that the reads-from relation of the given history be the same as that of a serial history, the reads of *all* transactions were considered. This notion has been called *view-serializability* [11]. If only the reads of T_f are considered, then *state-serializability* notion is obtained. Here the final database state is consistent but some transactions may see (read from) an inconsistent database state. A general notion is *S-serializability* where a subset *S* of *T* sees a consistent database state. This is characterized as follows:

A step *O* is *immediately-useful*-to another step O' if either O' reads the value (of a data item) written by *O*, or (i) *O* is a read of some data item, (ii) O' is a write of another (perhaps different) data item, (iii) both *O* and O' belong to the same transaction and (iv) *O* precedes O' in the transaction partial order. The *useful*-to is the transitive closure of immediately-useful-to. With respect to a specified subset *S* of *T*, a step of T_i is *S-useful* if it is useful to some transaction in *S*; otherwise, it is *S-useless*.

The *reads-from relation of h with respect to S*, denoted *rf* (*h*, *S*), is defined as *rf* (*h*, *S*) = {(T_i, *X*, T_j): T_j reads *X* from T_i, and $R_j[X]$ is *S*-useful}. Two histories *h* and *h*′ are *S-equivalent* if *rf* (*h*, *S*) = *rf* (*h*′, *S*). A history *h* is *S-serializable* if there is a serial history *h*′ *S*-equivalent to *h*.

Relative Atomicity

Serializable executions have been considered as correct executions on the premise that a transaction is a unit of atomicity, and any serial execution is correct. A serial execution is called a *basic* correct execution. This notion of atomicity has been found to be unnecessarily restrictive for many applications. Hence, researchers started refining atomicity units and hence accepting some non-serial histories as basic correct executions. An earliest refinement is in the definition of *nested transactions*, where a transaction consists of several subtransactions, each of which in turn consists of several subtransactions, and so on, with the traditional transactions at the lowest levels of the hierarchy. In one definition of nested transactions [8], each subtransaction is an atomic unit to other subtransactions which are siblings at the same level or their descendants.

The atomicity property in the nested transaction definition has two important characteristics: (i) an atomic unit may consist of *some*, not necessarily all, steps of a transaction; and (ii) some steps may constitute an atomic unit to *some* transactions, not to others. The atomicity property in sagas [5] has characteristic (i). A saga is a two-level nested transaction where each bottom level transaction is an atomic unit for every other transaction. The characteristic (ii) has been called *relative atomicity* [7]. The relative atomicity notion has been generalized in various stages. Garcia-Molina [4] defines *compatible* transactions as a set of transactions whose steps can interleave arbitrarily. If T_i and T_k are not compatible, then the entire transaction T_i is an atomic unit for T_k, and vice versa. If transactions T_i and T_j are compatible, then each step of T_i is an atomic unit for T_j, and vice versa. Then, (i) the steps of T_j can interleave with those of T_i arbitrarily and (ii) *any* number of steps of T_j can be executed after *any* step of T_i. The relative atomicity notion in [3] constrains property (i): T_j is allowed to interleave only at certain points in the execution of T_i, defined as the *breakpoints* of T_i with respect to T_j; but, whenever T_j is allowed, any number of its steps can be executed. With the *generalized relative serializability* notion of [13], the number of steps of T_j that can be executed at the individual breakpoints is restricted. Further, in the relative serializability notion, the above interleaving restrictions are only with respect to "dependent" steps; non-dependent steps are allowed to interleave anywhere in T_i. The precedence relation among the steps of the same transaction and conflict relations among the steps of different transactions contribute to the dependency. In all these proposals, any execution equivalent to any of the basic correct executions is considered as a correct execution. Lynch [7] extends the compatibility notion of [4] to subtransactions of nested transactions. Further extensions appear in [12].

Key Application

Serializability has become a de facto yardstick for arguing correctness of concurrent executions of programs. Serializability theory has revealed the amount of concurrency that can be achieved in correct executions and has helped in designing methods of achieving them. The theory has been extended from simple database operations to complex (and even non-electronic) activities, from individual to collaborative applications, and from centralized to highly distributed environments. The theory has also been enriched by adding application semantics along with the syntactic considerations.

Conflict serializability notion has served as a cornerstone to the design of database schedulers. The strength of the notion includes the following two nice properties: (i) it can be checked with a much simpler history graph where vertices correspond to transactions, edges correspond to conflicts, and acyclicity of the graph is a necessary and sufficient condition for the history to be conflict serializable; and (ii) if a history is conflict serializable then the projection of the history over any subset of the transactions is also conflict serializable. The class of histories allowed by the very popular *two-phase locking policy* is a subset of conflict-serializable histories.

Other serializability notions have been used in several advanced applications (for example in design databases [14]).

Cross-references

- ▶ ACID Properties
- ▶ Atomicity
- ▶ Concurrency Control – Traditional Approaches
- ▶ Multi-Version Serializability and Concurrency Control
- ▶ Transaction Management
- ▶ Transaction Models – the Read/Write Approach

Recommended Reading

1. Bernstein P.A., Shipman D.W., and Wong W.S. Formal aspects of serializability in database concurrency control. IEEE Trans. Software Eng., SE-5:203–215, 1979.

2. Eswaran K.P., Gray J.N., Lorie R.A., and Traiger I.L. The notion of consistency and predicate locks in a database system. Commun. ACM, 19:624–633, 1976.
3. Farrag A.A. and Özsu M.T. Using semantic knowledge of transactions to increase concurrency. ACM Trans. Database Syst., 14(4):503–525, 1989.
4. Garcia-Molina H. Using semantic knowledge for transactions processing in a distributed database. ACM Trans. Database Syst., 8(2):186–213, 1983.
5. Garcia-Molina H. and Salem K. Sagas. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 249–259.
6. Ibaraki T., Kameda T., and Minoura T. Serializability with constraints. ACM Trans. Database Syst., 12(3):429–452, 1987.
7. Lynch N.A. Multilevel atomicity – a new correctness criterion for database concurrency control. ACM Trans. Database Syst., 8(4):485–502, 1983.
8. Moss T.E.B. Nested Transactions: An Approach to Reliable Distributed Computing. Ph.D. Thesis, Technical Report MIT/LCS/TR-260, MIT Laboratory for Computer Science, Cambridge, Massachusetts, 1981.
9. Papadimitriou C.H. The serializability of concurrent database updates. J. ACM, 26:631–653, 1979.
10. Papadimitriou C.H. and Kanellakis P. On concurrency control by multiple versions. ACM Trans. Database Syst., 9:(1):89–99, 1984.
11. Vidyasankar K. Generalized theory of serializability. Acta Inf., 24:105–119, 1987.
12. Vidyasankar K. Unified theory of database serializability. Fundamenta Inf., 14(2):147–183, 1991.
13. Vidyasankar K. Generalized relative serializability. In Proc. of 9th Int. Conf. on Management of Data, 1998, pp. 313–327.
14. Vidyasankar K. and Dampney C.N.G. Version consistency and serializability in design databases. In Proc. 2nd Int. Conf. on Database Theory, 1988, pp. 368–382.
15. Weikum G. and Vossen G. Transactional information systems – theory, algorithms, and the practice of concurrency control and recovery. Morgan Kaufmann, 2002.

Service Bus

▶ Enterprise Service Bus (ESB)

Service Buses

▶ Interface Engines in Healthcare

Service Choreography

▶ Composed services and WS-BPEL

Service Component Architecture (SCA)

Allen Chan
IBM Toronto Software Lab, Markham, ON, Canada

Synonyms

SCA

Definition

The Service Component Architecture (SCA) [1] is a collaborative effort driven by a number of software vendors in the Open SOA (OSOA) [2] Collaboration group to facilitate the building of applications and systems based on service-oriented architecture. The final specification for SCA Version 1.0 was available as of March 21, 2007.

SCA is a set of specifications in the area of service composition, assembly, protocol bindings and policy definitions, where the Service Data Objects (SDO) [3] specification is used to specify how service data can be specified and manipulated. SDO provides a uniform access pattern for heterogeneous data sources, such as XML or relational databases. Although SCA and SDO can work independently of each other, they are often used together to provide a full end-to-end framework for defining SOA applications and systems.

Key Points

Conceptually, SOA is an architecture principle to enable software applications to be exposed as services. However, since SOA itself does not dictate how these services will be packaged or assembled, some of the benefits of SOA such as reuse, manageability and scalability become unpredictable. SCA can speed up SOA adoption by using the SCA Assembly Model to define how services can be declared, implemented and connected to each other.

The basic building block for SCA is an SCA Component, which can be used to declaratively describe the business *services* exposed by an *implementation*, and declare dependency on other business services as *references*. In addition, *bindings* can be applied to any *services* or *references* to describe how a client application can invoke an existing service or how an external service can be accessed, respectively.

SCA also supports a recursive composition model to support the creation of a composite SCA Component. Another aspect of the SCA Assembly Model is the SCA Policy Framework [2], which provides a way to capture the non-functional aspects of an SOA system.

The SCA specification provides a framework for the implementation of scalable and manageable SOA systems, such as the enterprise service bus (ESB).

Cross-references

▶ Enterprise Service Bus (ESB)
▶ Service Data Objects (SDO)
▶ Service Oriented Architecture (SOA)

Recommended Reading

1. Open SOA Collaboration, http://www.osoa.org/.
2. SCA Specification, Final Version 1.0, http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications.
3. SDO Specification, Final Version 2.1, http://www.osoa.org/display/Main/Service+Data+Objects+Specifications.
4. Spring Framework http://static.springframework.org/spring/docs/2.0.x/reference/index.html.

Service Composition

▶ Composition

Service Item

▶ Clinical Order

Service Orchestration

▶ Composed services and WS-BPEL

Service Order

▶ Clinical Order

Service Oriented Architecture

Serge Mankovski
CA Labs, CA Inc., Thornhill, ON, Canada

Synonyms

SOA

Definition

Service Oriented Architecture is a conceptual model for integration of software systems where system function is performed by coordinated invocation of services. In this model term service refers to *significant atomic* computational activity that can be invoked over a computer network. Service computational activity is significant in the sense that it is in order of several magnitudes more complex than a function invocation and atomic in the sense that it is a smallest element of functional decomposition in this model.

Historical Background

Historically IT systems are built in a competitive environment where each vendor is trying to develop best possible solutions to their customers and, at the same time, trying to build even more complex systems in attempt to serve any customer need and prevent customers looking for solutions from another vendor. Logic of this process led to development of IT systems that either did not have any means for integration with other systems, or at most had interfaces necessary for integration between within the brand. At the same time companies using IT systems to conduct business were trying to integrate systems in attempt to streamline operations, increase utilization of IT asset and achieve business critical functionality. These competing forces were shaping landscape IT for several decades. Over time there were a number of successful attempts to build large scale integration of IT systems, but more often than not, these integrated solutions themselves were becoming silos in its own right. A notable example of this process was emergence of electronic data interchange (EDI) system that was under development from mid 1960s almost at the same time when DARPA started work leading to development of the Internet. By the time when Tim Berners-Lee developed a first web browser in 1990, EDI was already well established as a successful model of integration of IT functionality across multiple companies.

It is perhaps not possible to pinpoint when exactly SOA way of thinking started. Impact and acceptance of SOA became more evident after successes standardization of technologies that were necessary within the SOA model. Experience and lessons learned by the industry from several decades of development, deployment and operation of various integration architectures, emergence of XML as a universal data interchange format, emergence of new open standards, and tremendous success of Internet have build technological foundation for SOA adoption. Broad adoption started when critical mass of standards comprising SOA stack has matured. At the same time Open Source Development phenomena made implementations of the standards broadly available. It removed barriers created by proprietary implementations and lowered barrier to entry for new users of the technology. Growth in scope, diversity and rapid pace of change in business requirements and demand business agility created awareness of SOA benefits in business community.

Foundations

SOA as an integration architecture is concerned with all aspects of interactions between IT systems. SOA postulates that a basic element of SOA architecture is service.

SOA system is comprised of a number of *services* deployed over a computer network. *Service* is a basic building element of SOA. Notion of a service is similar to notion of object in object oriented programing, but it is more coarse-grained. For example, a service can represent an important function of an IT system or even entire system all together. Each service has an *interface*. Service interface is *metadata*, or data about data, needed to *invoke* service operation.

In respect to invocation SOA distinguishes two roles – Service Consumer and Service Producer. Service Consumer is a system invoking service and Service Producer performs the service. Service Producer invokes service by message to an instance of Service Producer. Upon receiving an invocation conforming to the service interface, Service Producer executes requested operation using parameters provided by the invocation. Service execution might also include invocation of other services within the SOA system and hence Service Producer can also play role of Service Consumer in respect to another service. When Service Producer completes execution of the operation it replies to Service Consumer with an acknowledgement of successful completion or data containing results of the invocation or indication of fault. This reply from Service Producer to Service Consumer indicates completion of service execution.

It is important to note that service interface is an abstraction that does not take into account details of the network data transport needed to invoke the service or details of service implementation. Use of the interface abstraction allows for definition of service interactions at the higher level of abstraction than in any other integration architecture. In particular, it allows for definition of service interactions at design time and achieves high degree of flexibility regarding service implementation, location, time, and transport protocol needed for service invocation.

Since interface is separated from service instance it is necessary to associate interface with a service instance in a process that is called *service binding*. SOA allows to perform service binding at any time of SOA system life time. In particular, it can be done at run time by means of *service discovery* and *service lookup*. Service discovery is a process of discovery of services capable to perform a necessary function. Service lookup is a process of finding a service *end point reference* based on a service name, name of a service interface, or the interface metadata itself. End point reference is a piece of metadata that must contain a protocol dependent service address along with optional parameters and session identifier.

Notion of service interface is important from the software engineering point of view because allows for separation of the function performed by service from implementation of service itself. This constitutes a good software engineering practice leading to more robust system design. It also enables very powerful notion of service *orchestration*. Service orchestration is a process of delivering a *composite service* function by coordinated invocation of other services. It is usually done within an orchestration engine that executes the service in accordance with the process definition describing logic and sequence of invocation of orchestrated services along with the necessary *data transformation*. Data transformation is an activity of modifying syntax of the data exchanged between the services to accommodate their interface requirements.

SOA as a conceptual model accommodates wide variations. Any of the architectural concepts highlighted above can be omitted, except concept of service. This is perhaps why it is called service-oriented architecture.

In respect to services there are at least two major types of services:

1. SOAP services use WS-* stack of Web Service Standards. These services are based on a number of international standards developed with W3C and OASIS standardization committees and make extensive use of XML.
2. RESTful services emerged from the world of Open Source. They make use of HTTP protocol and

became “standard-de-facto” for web based services and mush-ups.

Within SOA, there is a wide degree of variation in respect to use of metadata. SOAP services make extensive use of metadata: XML Schema for message syntax, WSDL and Schema for interface definition, WSDL for binding, UDDI for lookup, BPEL for orchestration. RESTful services use HTTP verbs for defining actions and use HTTP for data transport. They do not have formal definition of data syntax beyond URL syntax, no notion of binding, lookup, discovery and orchestration and hence no metadata associated with them.

In respect to invocation there are three types of invocation:

1. *Synchronous* invocation. This form of invocation passes control of execution from Service Consumer to Service Producer and it does not return to Service Consumer until service is completed. This type of invocation is common within both SOAP and RESTful services.
2. *Asynchronous* invocation. In this form of invocation Service Consumer does not need to wait for Service Producer to complete service invocation. Service Consumer can carry on performing its own function, but it requires capability on the part of the Service Consumer to receive message indicating completion of the service. This type of invocation is often done by use of a messaging system. This type of invocation is more common among SOAP services.
3. *Enterprise Services Bus (ESB) based* invocation. In this form of invocation Enterprise Service Bus performs function of discovery, lookup, binding, messaging, data transformation and orchestration. ESB-based invocation can be performed synchronously or asynchronously, but in this case service invocation can be done using abstract Service Producer interface. Service invocation is passed to ESB that performs Service Producer lookup and binding and invocation. If necessary, ESB performs data transformation. If requested service is a composite service, ESB performs necessary orchestration.

Variation in respect to service discovery, lookup and binding range from systems fully bound at design time to systems using run-time semantic-based mechanisms employing artificial intelligence methods and techniques. Systems performing mission critical function tend to drift towards design time binding. SOA systems aiming to accommodate high rate of changes tend to drift towards sophisticated run-time binding.

There is a wide variation in use and purpose of the SOA systems themselves. WS-* based service architecture tend to be used for enterprise integration. They often use ESB as a back-bone carrying majority of the business related data. It becomes a focal point where enterprise policies can be enforced and formal audit necessary for proving business compliance can be conducted. Because of this highly visible position of SOA systems within enterprise it gives rise to notion of *SOA Governance*. SOA Governance is an on-going activity within an enterprise maximizing leverage of the SOA infrastructure for business purposes. SOA Governance has two distinctive aspects. One aspect refers to ensuring that all aspects of an enterprise functioning within SOA are performing their functions in expected manner by means of enforcing and auditing compliance with business policies, practices. In this aspect SOA provides means for automated support and tractability of decisions and actions performed within enterprise. Automated decision and action support within SOA is achieved by use of service orchestration. SOA Orchestration provides automation of business processes, automatic routing of documents, enforcement of timely document processing, notification of non-compliance, and change management. Tractability of decision and actions is achieved by retaining of service invocation data within the SOA infrastructure. It allows for cross-referencing and correlation of logging data retained with the services and allows reconstructing entire picture of business activity at any point of time. It ensures that at any point of time there is a means of checking if business activities were performed in accordance with law, regulations and in adherence to best practices.

Another meeting of SOA Governance relates to operation of the SOA system itself. Services within SOA have a certain degree of freedom and can change independently from each other as long as interfaces remain unchanged. However they are not completely independent because it is often not possible make them completely independent and there is still some degree of dependence between the services. This requires some level of control over the degree in which services can vary. SOA Governance makes sure that any change within the system does not destabilize or jeopardize business function performed by the system. It is accomplished by imposing policies to restrict degree of changes in behavior of services, ensure that services continue to perform within established service levels, managing deployment of new services and maintaining operation of the existing services. This from of governance also uses the same automation and tractability infrastructure as the other one.

Key Applications

Enterprise Application Integration, Business Process Optimization.

Future Directions

Deployment of SOA without changing business processes does not produce the same level of return on investment as if deployment is accompanied by changes in the business processes tailored to take advantage of the SOA system. On the other hand changes in business processes trigger changes in the SOA system. In the future, it would be necessary to develop a methodology for SOA deployment. This methodology would have to cover both technical and business aspects as well as provide foundation for understanding of economic impact and, ultimately, quantify return on investment associated with SOA deployment.

Cross-references

- ▶ BPEL
- ▶ Enterprise Application Integration
- ▶ Enterprise Service Bus
- ▶ Messaging Systems
- ▶ Mush-up
- ▶ OASIS
- ▶ Open Source
- ▶ RPC
- ▶ SOAP
- ▶ UDDI
- ▶ W3C
- ▶ Web Services
- ▶ WSDL

Recommended Reading

1. OASIS Reference Model for Service Oriented Architecture, http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf
2. Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Thomas Erl. The Prentice-Hall Service Oriented Computing Series, 2005.
3. Understanding Enterprise SOA. Eric Pulier, Hugh Taylor, Manning, 2005.
4. W3C Web Services Glossary, http://www.w3.org/TR/ws-gloss/

Service Request

▶ Clinical Order

Session

Sameh Elnikety

Microsoft Research, Cambridge, UK

Synonyms

Database interaction

Definition

A database session is sequence of interactions between a client and a database server. The session captures the state of the client's in-flight SQL commands.

Key Points

Session state may contain database objects, such as temporary relations, which are accessible only within the session. For efficiency, some database engines maintain session state per connection rather than per client. In this case, it is called connection state.

A client expects to see the effects of its previous updates to the database. This concept is called session consistency [1] and is illustrated in the following example. A client issues a transaction to buy a book. Then, it sends a subsequent transaction to see the list of ordered books. Session consistency requires the list to contain that book. Session consistency is trivial to implement in a centralized database system, but becomes harder in a distributed database system.

Cross-references

- ▶ Connection
- ▶ Strong Consistency Models for Replicated Data

Recommended Reading

1. Daudjee K. and Salem K. Lazy database replication with ordering guarantees. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 424–435.

Set Abstraction

▶ Comprehensions

Set-Difference

▶ Difference

Shape Descriptors

▶ Feature-Based 3D Object Retrieval

Shared-Disk File System

▶ SAN File System

Shared-Disk Architecture

Patrick Valduriez
INRIA, LINA, Nantes, Cedex, France

Definition

In the shared-disk architecture, only the disks are shared by all processors through the interconnection network. The main memory is not shared: each processor exclusive (non-shared) access to its main memory. Each processor-memory node is under the control of its own copy of the operating system. Since any processor can cache the same disk page, a cache coherency mechanism is necessary.

Key Points

Shared-disk requires a cache coherency mechanism which allows different nodes to cache a consistent disk page. This function is hard to support and requires some form of distributed lock management. The most notable parallel database system which uses shared-disk is Oracle, with an efficient implementation of a distributed lock manager for cache consistency.

Shared-disk has a number of advantages: lower cost, good extensibility, availability, load balancing, and easy migration from centralized systems. The cost of the interconnection network is significantly less than with shared-memory since standard bus technology may be used between processor nodes. Given that each processor has enough main memory, interference on the shared disk can be minimized. Thus, extensibility can be better, typically up to a hundred processors. Since memory faults can be isolated from other nodes, availability can be very good. Load balancing is relatively easy as a query at any node can access all data on the shared disks. Finally, migrating from a centralized system to shared-disk is relatively straightforward since the data on disk need not be reorganized.

However, shared-disk suffers from complexity and potential performance problems. It requires distributed database system protocols, such as distributed locking and two-phase commit which are complex. Furthermore, maintaining cache consistency can incur high communication overhead among the nodes. Finally, access to a shared-disk is a potential bottleneck.

Cross-references

▶ Parallel Data Placement
▶ Parallel Query Processing
▶ Query Load Balancing in Parallel Database Systems

Shared-Disk Databases

▶ Process Structure of a DBMS

Shared-Everything

▶ Shared-Memory Architecture

Shared-Everything Databases

▶ Process Structure of a DBMS

Shared Health Record

▶ Electronic Health Records (EHR)

Shared-Memory Architecture

Patrick Valduriez
INRIA, LINA, Nantes Cedex, France

Synonyms

Shared-everything

Definition

In the shared-memory architecture, the entire memory, i.e., main memory and disks, is shared by all processors. A special, fast interconnection network (e.g., a high-speed bus or a cross-bar switch) allows any processor to access any part of the memory in parallel. All processors are under the control of a single operating system which makes it easy to deal with load balancing. It is also very efficient since processors can communicate via the main memory.

Key Points

Shared-memory is the architectural model adopted by recent servers based on symmetric multiprocessors (SMP). It has been used by several parallel database system prototypes and products as it makes DBMS porting easy, using both inter-query and intra-query parallelism.

Shared-memory has two advantages: simplicity and load balancing. Since directory and control information (e.g., lock tables) are shared by all processors, writing database software is not very different than for single-processor computers. In particular, inter-query parallelism is easy. Intra-query parallelism requires some parallelization but remains rather simple.

Load balancing is also easy to achieve since it can be achieved at run-time by allocating each new task to the least busy processor.

However, shared-memory has three problems: cost, limited extensibility and low availability. The main cost is incurred by the interconnection network which requires fairly complex hardware because of the need to link each processor to each memory module or disk. With faster processors, conflicting accesses to the shared-memory increase rapidly and degrade performance. Therefore, extensibility is limited to a few tens of processors, typically up to 16 for the best cost/performance. Finally, since memory is shared by all processors, a memory fault may affect several processors thereby hurting availability. The solution is to use duplex memory with a redundant interconnect which makes it more costly.

Cross-references

- ► Parallel Data Placement
- ► Parallel Query Processing
- ► Query Load Balancing in Parallel Database Systems

Shared-Nothing Architecture

Patrick Valduriez
INRIA, LINA, Nantes, Cedex, France

Synonyms

Distributed architecture

Definition

In the shared-nothing architecture, each node is made of processor, main memory and disk and communicates with other nodes through the interconnection network. Each node is under the control of its own copy of the operating system and thus can be viewed as a local site (with its own database and software) in a distributed database system. Therefore, most solutions designed for distributed databases such as database fragmentation (called partitioning in parallel databases), distributed transaction management and distributed query processing may be reused.

Key Points

As opposed to symmetric multiprocessor (SMP), shared-nothing is often called massively parallel processor (MPP). Many research prototypes and commercial products have adopted the shared-nothing architecture because it has the best scalability. The first major parallel DBMS product was Teradata which could accommodate a thousand processors in its early version in the 1980s. Other major DBMS vendors, except Oracle, have provided shared-nothing implementations.

Shared-nothing has three main advantages: low cost, high extensibility, and high availability.

The cost advantage is better than that of shared-disk which requires a special interconnection network for the disks. By easing the smooth incremental growth of the system by the addition of new nodes, extensibility can be better (in the thousands of nodes). With careful partitioning of the data on multiple disks, almost linear speedup and linear scale up could be achieved for simple workloads. Finally, by replicating data on multiple nodes, high availability can be also achieved.

However, shared-nothing is much more complex than either shared-memory or shared-disk.

Higher complexity is due to the necessary implementation of distributed database functions for large numbers of nodes, in particular, data placement. Load balancing is more difficult to achieve because it relies on the effectiveness of database partitioning. Unlike shared-memory and shared-disk, load balancing is decided based on data location and not the actual load of the system. Furthermore, the addition of new nodes in the system presumably requires reorganizing the database to deal with the load balancing issues.

Cross-references

▶ Parallel Data Placement
▶ Parallel Query Processing
▶ Query Load Balancing in Parallel Database Systems

Shared-Nothing Databases

▶ Process Structure of a DBMS

Shot Boundary Detection

▶ Video Shot Detection

Shot Segmentation

▶ Video Segmentation

Shotcut Detection

▶ Video Segmentation

SI

▶ Snapshot Isolation

Side-Effect-Free View Updates

Yannis Velegrakis
University of Trento, Trento, Italy

Definition

A view is an un-instantiated relation. The contents of its instance depend on the view query and the instances of the base tables. For that reason, an update issued on the view cannot be directly applied on the view instance. Instead, it has to be translated into a series of updates on the base tables so that when the view query is applied again on the modified base table instances, the result of the view update command will be observed on the view instance. Unfortunately, it is not always possible to find an update translation such that the change observed on the view instance is the one and only the one specified by the view update command. When this happens for a view update translation, the translation is said to have no *side-effects*. To fully exploit the updateability power of views, it is desired to be able to find update translations that have no side-effects.

Historical Background

Updates on the views were introduced almost simultaneously with views. Their importance has been recognized by Codd himself. In fact, one of the twelve rules that Ed Codd [2] introduced to define what a real relational database is, was referring to the ability of the views to be updateable. In particular, the sixth rule was:

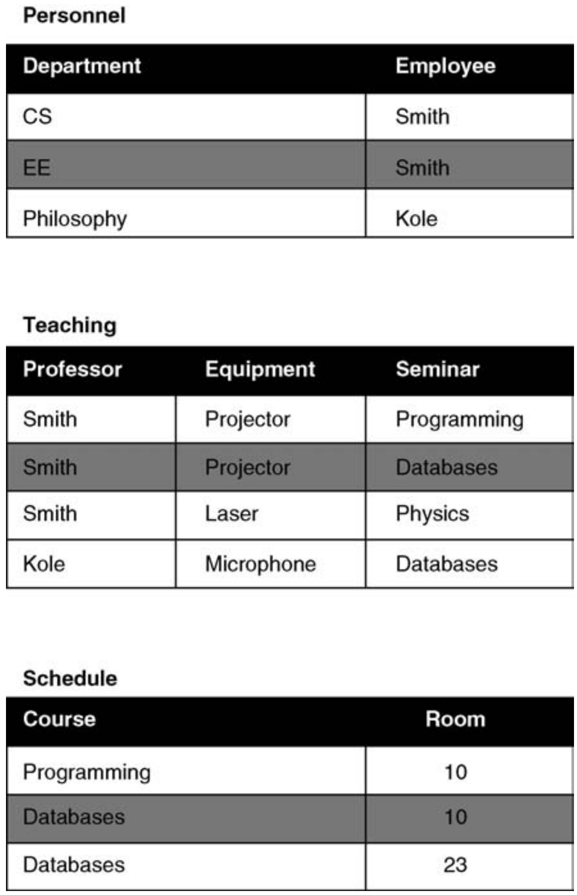
"All views that are theoretically updatable must be updatable by the system".

The term *theoretically updateable* is referring to the ability of finding side-effect-free translations of the view updates.

Foundations

The problem of side-effect-free updates is based on the problem of *Updates through views*. Referring to Figure 1 of that entry, the update translation W of a view update U on a view V is said to have no side-effects if U $(Q_V(I)) = Q_V(W(I))$, where I is the database instance and Q_V is the view query of the view V.

To better realize the problem of side-effect-free view updates, consider a database instance that consists of the 3 tables of Figure 1.



Personnel

Department	Employee
CS	Smith
EE	Smith
Philosophy	Kole

Teaching

Professor	Equipment	Seminar
Smith	Projector	Programming
Smith	Projector	Databases
Smith	Laser	Physics
Kole	Microphone	Databases

Schedule

Course	Room
Programming	10
Databases	10
Databases	23

Side-Effect-Free View Updates. Figure 1. Three base Tables.

Suppose that a view V_1 is defined on top of these three tables through the following view query:

```
select *
from Personnel P, Teaching T, Schedule S
where P.Employee = T.Professor and
  T.Seminar = S.Course
```

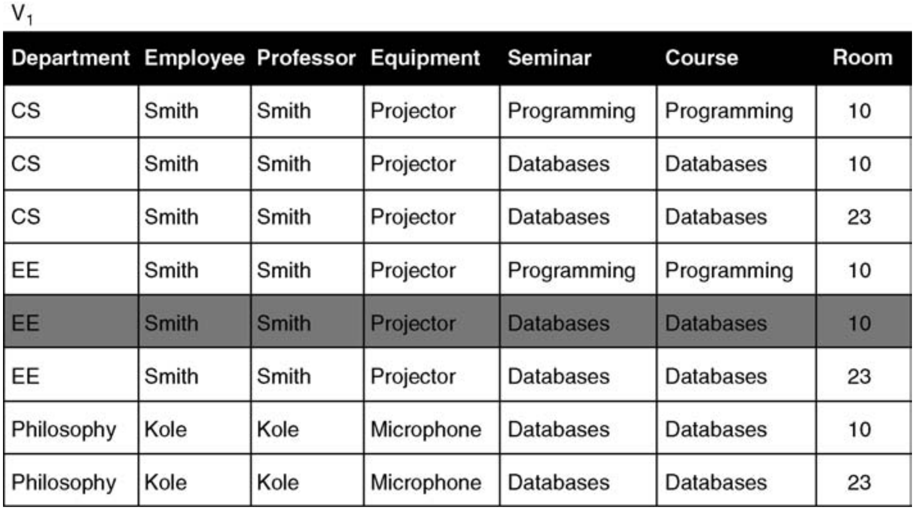
The instance of the view will be the relation illustrated in Figure 2.

Consider now an update command on this view that requests the deletion of the tuple t_d:[EE, Smith, Smith, Projector, Databases, Databases, 10]. Tuple t_d appears in the view instance due to the join of the 3 tuples [EE, Smith], [Smith, Projector, Databases] and [Databases, 10] of the base tables Personnel, Teaching and Schedule. Deletion of any (or all) of these tuples will achieve the desired result of deleting tuple t_d from the view. However, any such deletion will have additional effects in the view instance. For instance, the removal of tuple [EE, Smith] from Personnel will also eliminate the view tuples that are immediate before and after t_d. Similar observations can be made for the tuples in the other two base relations. In fact, for the particular update, it can be shown that there is no change that can be made on the base tables to achieve the desired tuple deletion without any additional changes, i.e., side-effects, in the view instance.

Side-effects are not observed only on deletions but also on insertions. For instance, consider the update command that requests the insertion of tuple t_i:[Economy, Smith, Smith, Projector, Databases, Databases, 10] in V_1. For the appearance of t_i in the view V_1 to be justified, tuples [Economy, Smith], [Smith, Projector, Databases] and [Databases, 10] need to exist in the instances of Personnel, Teaching, and Schedule, respectively. The last two are already there, but not the first. The translation of the insert command on the base tables will insert [Economy, Smith] in Personnel. Unfortunately, due to the value "Smith" in its attribute *Employee*, tuple [Economy, Smith] will be able to join with every other tuple of table Teaching that has value "Smith" in the attribute *Professor*. This will introduce additional tuples in the V_1 instance that the insert command did not request.

Base tables, i.e., tables that have been defined through the "create table" command, have standalone instances, thus, update commands on them can be implemented without side-effects by simply modifying their materialized instance accordingly. If views are to be used as any other table, a view needs to show the same behavior as base tables. This means that update commands need to have translations that generate no side-effects in the view instance. It would have been really surprising for an application or a user that is not aware that a relation she is interacting is actually a view, to request the deletion (or insertion) of a tuple and then see additional tuples disappearing from the view (or appearing in it).

To cope with the view update translation side-effects one option is to leave the burden to the database administrator who defines the view. During the view definition, the administrator is responsible to specify not only the view query but also how exactly each update is translated to updates on the base tables [7] and make sure that side-effects will not occur. The drawback of this option is that it requires a lot of knowledge and experience from the administrators. If the administrator determines that for a



V_1

Department	Employee	Professor	Equipment	Seminar	Course	Room
CS	Smith	Smith	Projector	Programming	Programming	10
CS	Smith	Smith	Projector	Databases	Databases	10
CS	Smith	Smith	Projector	Databases	Databases	23
EE	Smith	Smith	Projector	Programming	Programming	10
EE	Smith	Smith	Projector	Databases	Databases	10
EE	Smith	Smith	Projector	Databases	Databases	23
Philosophy	Kole	Kole	Microphone	Databases	Databases	10
Philosophy	Kole	Kole	Microphone	Databases	Databases	23

Side-Effect-Free View Updates. Figure 2. The view V_1 instance.

given view update there is no translation that has no side-effects, she can make the view not to accept this kind of updates, or allow the side-effects to happen if she believed that this is the semantically correct behavior.

Instead of letting the administrator deciding whether an update should be allowed or not, an alternative solution is to develop methods to perform this test automatically. Based on this idea, Keller [4] developed five criteria to characterize the correctness of a view update translation. The first of these criteria requires the translation to have no side-effects. A consequence of this is that keys of the base relations have to appear in the views, i.e., cannot be projected out. This reduces the cases in which side-effects may appear in the views, but does not completely eliminate them. Keller studied the different choices that exist when translating updates on select, project, select-project and select-project-join views, and provided algorithms for update translation for each case. These algorithms are guaranteed to respect his 5 criteria. For a given update on the view, there may be more than one algorithms that can be used, i.e., more than one possible translations. Which one to be used is a decision that is provided by the database administrator at the moment of view definition [5].

Dayal and Bernstein [3] introduced the notion of the *view-trace* and the *view-dependency* graph. They are graphs that model the dependencies between the attributes of the base tables as determined by the schema and the view definitions. Through them one can determine whether there is an update translation that has no side-effects. For each update on the view, either a unique side-effect-free translation is found and is applied, or the update is not allowed to occur. This approach eliminates the need of an administrator involvement, but cannot be applied in cases in which side-effect generated translations are allowed to occur if they are semantically meaningful.

A different method to determine the kind of updates a view can accept without generating side-effects is through the *constant complement.* Two views are considered complementary if given the state of each view there is a unique corresponding database state. This means that when the instance of one of these views changes (due to an update) while the instance of the other is kept constant, then there is a unique database instance from which the instances of the two views are generated. In other words, the correct translation of the view update is unique [1]. Unfortunately, given a view V, finding its view complement has been shown to be NP-complete even for views with very simple view definition queries.

In summary, it is not always guaranteed that there is a unique side-effect-free translation of a view update. In practical situations, updates are typically allowed on the views. If there are more than one translations of a view update, some criterion may be used to select one of these translations, e.g., the minimality of the changes in the database instance, or some specific parameters that the data administrator specified at the time of view definition. Another approach is to disallow updates on the view that have more than one translation. In the presence of side-effect generating translations, one approach is to allow the translation to take place, allowing that way the side-effects to appear on the view, or to disallow the update completely.

But what would happen in cases in which an update on the view is absolutely necessary, as is the case of an application that can access the database only through a view interface without being aware of the fact that the relation it is accessing is actually a view and with the need to perform updates on it as it would have done if it was a base table? An idea proposed by Kotidis et al. [6] is the following. When an update command is issued on the view, the change in the view instance must be exactly the one described by the update. However, any change on the base table should not take place unless it is implied by the semantics of the view query and the update command. For instance, the deletion of the tuple [EE, Smith] from Personnel as a translation of the delete command for the view tuple [EE, Smith, Smith, Projector, Databases, Databases, 10] mentioned above, would have implied that the reason that the view tuple is deleted is that Smith stopped being affiliated with the EE department. However, neither the semantics of the update command, nor the semantics of the view query imply something like that. Similar claims can be done for tuples [Smith, Projector, Databases] and [Databases, 10] of tables Teaching and Schedule, respectively. For the specific view tuple deletion, the claim is that no change should be observed in the instances of the three base tables, but tuple [EE, Smith, Smith, Projector, Databases, Databases, 10] will be removed from the view instance. This behavior will only be possible if one can accept views whose instances are not exclusively determined by the results of their view queries on the base tables, but also from the update commands that have been issued on them.

Key Applications

Achieving side-effect-free updates on the views is of great importance for systems that provide access to their data through views, but at the same time need to hide from their users or the applications that use the system the fact that they are dealing with views and not actual relations.

Cross-references

▶ Provenance
▶ Updates through Views

Recommended Reading

1. Bancilhon F.B. and Spyratos N. Update Semantics of Relational Views. ACM Trans. Database Syst., 6(4):557–575, 1981.
2. Codd E.F. Is Your DBMS Really Relational? Computer-World, 1985.
3. Dayal U. and Bernstein P. On the correct translation of update operations on relational views. ACM Trans. Database Syst., 8(3):381–416, 1982.
4. Keller A.M. Algorithms for translating view updates to database updates for views involving selections, projections, and joins. In Proc. 4th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1985, pp. 154–163
5. Keller A.M. Choosing a view update translator by dialog at view definition time. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 467–474.
6. Kotidis Y., Srivastava D., and Velegrakis Y. Updates through views: a new hope. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
7. Rowe L.A. and Shoens K.A. Data abstractions, views and updates in Rigel. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 71–81.

Sight

▶ Visual Perception

Signal Transduction Networks

▶ Biological Networks

Signature Files

Mario A. Nascimento
University of Alberta, Edmonton, AB, Canada

Definition

A signature file allows fast search for text data. It is typically a very compact data structure that aims at minimizing disk access at query time. Query processing is performed in two stages: filtering, where false-negatives are guaranteed to not occur but false-positives may occur and, query refinement, where false-positives are removed.

Historical Background

Efficient and effective text indexing is a well-known and long-standing problem in information retrieval. While inverted files are nowadays a *de facto* standard for text indexing, in the early days, its storage overhead was not acceptable for larger datasets. In addition, accessing an inverted file on disk would require a relatively large number of (expensive) disk seeks. The main motivation for signature files is to allow fast filtering of text using a linear scan of the signature file for finding text segments that may contain the queried term(s). Given that the found segments may be false-positives, a refinement step is required before the final correct answer is returned. The main compromise in signature files lies in how to build signatures for terms and for text segments that allow low storage overhead, fast disk access, and minimizes the ratio of false-positives.

Foundations

Let T be a text to be indexed that is divided into non-overlapping blocks T_i each containing b contiguous terms. For each block T_i a binary signature $S(T_i)$ is built. Assume a hashing function $h(.)$ that takes as an argument a term and returns a signature of length B. The signature for a block T_i can then be obtained by performing a bitwise-OR of the signatures of the terms in that block typically excluding the terms in the stop-list. The signature file for T is then the set of signatures for its $\mathrm{T_i}$. A query term Q is also mapped into a signature $h(Q)$. At query time each block signature $S(T_i)$ is compared to the query's signature $h(Q)$. Denoting the bitwise-AND operator by "&" one can show that if $S(T_i)$ & $h(Q) = h(Q)$ then block T_i may contain the query term Q and is considered a candidate answer. However, in order to guarantee that only correct answers are returned, all candidate answers must be refined to ensure that they do contain query term Q. The reason for such false-candidates is that when a block signature is built, a particular bitstring matching the query's signature may appear as an incidental combination of different signatures.

Consider the sample text T: "To be, or not to be: that is the question" (punctuation marks can be ignored without loss of generality). Assume further that a hashing function $h(.)$ is such that it hashes all the terms in this text as shown in Table 1.

The resulting block signatures will depend on the value chosen for b. Table 2 shows the resulting blocks T_i and their respective block signatures $S(T_i)$ assuming $b = 2$ and the hashing signatures shown in Table 1.

If Q = "question," then T_5 satisfies the criterion $S(T_5)$ & $h(Q) = h(Q)$. In this particular case, T_5 indeed contains Q and the text is selected as a true-positive answer. Consider now the case where Q = "to." In this *case* $(S(T_1)$ & $h(Q)) = (S(T_3)$ & $h(Q)) = (S(T_5)$ & $h(Q)) = h(Q)$. In query refinement the actual blocks T_1, T_3 and T_5 need to be read from disk and inspected. While T_1 and T_3 do contain query term Q thus being true-positives, block T_5 does not, i.e., it is a false-positive. This illustrates the major drawback of signature files. While one can safely discard a block T_i if $S(T_i)$ & $h(Q) = h(Q)$, the same is not true otherwise. Without the query refinement step the query's answer is prone to contain false-positives, which is typically not acceptable. Several factors need to be considered when aiming at minimizing the probability of false-positives. For instance, the length of the produced signatures (B), the number of terms per text block (b) and the number of bits randomly set in the signatures (which is denoted by n). For instance, for an optimal selection of $n = B \ln(2)/b$ then the false-positive probability is $1/2^n$ [1].

There are other important issues to be considered in addition to minimizing the false-positive probability, and thus the overhead of query refinement, when using signature files. In principle the query signature could be also be the result of a bitwise-OR between several terms, hence in principle leading to the possibility of querying for phrases. Unfortunately, phrase queries are not well supported by signature files. This is due to the fact that the bitwise-OR operation used to produce the block and query signatures does not preserve any notion of order among the terms

Signature Files. Table 1. Sample term signatures

Term	To	Be	or	not	that	is	the	question
H(term)	100100	011000	010010	101000	001100	010001	100001	000110

in the block. Therefore, two blocks with very different term ordering will have the exactly same block signature, which is an obvious problem. Another related problem occurs when terms in a phrase query occur in the boundaries of blocks. For instance, consider if one searches for Q = "not to," which would yield $h(Q)$ = 101100. Using the block signature illustrated in Table 2, one would find that while T_1 and T_2 would be candidate blocks all would be considered false-positives in the query refinement and one would be unable to find that the queried phrase is indeed in T. It would be missed altogether for being in the boundaries of the constructed text blocks.

Although less popular, negation queries, where one searches for a document not containing a query term, can also be answered using signature files. Recall that in the discussion above if $S(T_i)$ had at least the same bits set as $h(Q)$ then T_i could be answer, pending further checking. If a block T_i does not have at least the same bits as $h(Q)$ set is definitely an answer to the negation query. However, if it has those same bits set, it may still be an answer because those bits may have independently been set by other terms. Thus, in this case further checking is required as well. As an illustration consider the term Q = "writer" and assume that $h(Q)$ = 100010. Since $S(T_i)$ & $h(Q) \neq h(Q)$ for i = 1, 3 and 4 one can be sure that those blocks do not contain Q and therefore, up to that point, T could be an answer to query Q. However the fact that $S(T_i)$ & $h(Q) = h(Q)$ for i = 2 and 5 does not necessarily mean that those blocks contain Q. In fact, only upon checking the text of the blocks one could verify that they do not and consequently classify the text as satisfying the query.

It should be clear by now that the crucial issue in dealing with signature files is to minimize the overhead of query refinement. The more blocks needed to be further check the closer the performance will be to that of reading the whole dataset, which is obviously not desirable. This issue can be addressed by using lengthier signatures possibly through more complex hashing schemes. Due care is needed though as excessively long signatures detract from the claimed low storage overhead yielded by the signature files.

Signature Files. Table 2. Block signatures for "To be, or not to be: that is the question"

Block # (i)	Terms in the block (T_i)	Block Signature $S(T_i)$
1	To be	111100
2	or not	111010
3	to be	111100
4	that is	011101
5	the question	100111

Despite its limitations above, signature files offer the possibility of very efficient search on disk. Given that the only operation necessary is to read and compare binary signatures in a linear fashion, no relatively expensive disk seeks are necessary during the file scan. Furthermore, the inspection of the signature blocks can be implemented very efficiently in memory. Updating the indexed texts can also be carried out with low overhead in a fairly straightforward way.

Bit-Slice Signature File

Even though the framework presented above is effective, fairly efficient and relatively simple to implement it can be further optimized by exploring the layout of the signature file. The typical layout is to have all block signatures written contiguously in a file. At query time, each and every block is read from the disk and compared to the query signature, regardless of which bits are set in the same. The bit-slice signature aims at reducing disk access per query by exploiting the fact that likely few bits are set in the query signature and in the text blocks.

In order to illustrate the idea, take Table 2 and transpose it, i.e., instead of having one row per signature, one will have one row per bit, i.e., the jth row will contain the jth bit of every block signature, ordered from most to less significant. Table 3 shows the resulting table after "transposing" Table 2.

Notice that the idea when checking whether $S(T_i)$ & $h(Q) = h(Q)$ is to look for common set bits in corresponding positions. For instance, when searching for a block that contains Q = "or" one needs to inspect the second and fifth bits since $h(Q)$ = 010010. Using the bitslice S_2 one can infer that blocks T_1, T_2, T_3 and T_4 have the 2nd bit set and therefore they could contain Q if they also have the 5th bit set. Similarly using S_5 one can infer that blocks T_2, T_4 *and* T_5 could contain Q is their 2nd bit is also set. If S_2 & S_5 is computed, one obtains the index to the blocks that need be verified. Interestingly enough, this is equivalent to performing the intersection of the two sets of terms obtained by inspecting each bit individually. In this case S_2 & S_5 = 01010 and hence blocks T_2 and T_4 are candidate blocks. Note that as before, candidate blocks need to be further refined as false-positives are still possible.

While the query processing using plain signature files would require a linear scan of the whole signature, using bitslice signature this is typically not the case. The tradeoff to be considered is that now one needs to perform (relatively expensive) random disk access in both the bitslice file and the signature files. As well, the storage overhead for the bitslice file, nearly as large as the signature file itself needs to be taken into account. Just like the more straightforward case, a number of parameters need to be considered to produce efficient bitslice signature files.

Signature Trees

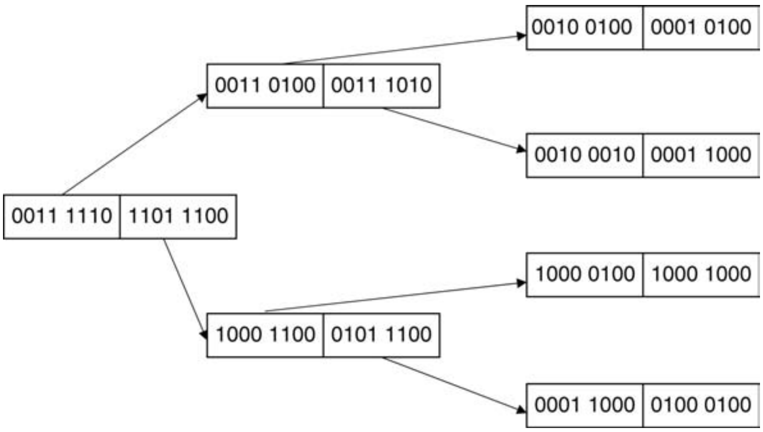
As an alternative to simple flat signature files one can also arrange the signatures in a hierarchical balanced tree structure, similar to a B$^+$-tree. Once again, the idea is to trade the few disk seeks required by a linear scan of the signature file by additional seeks that would allow pruning, hence not reading large portions of the signatures.

Signature Files. Table 3. Bitslices for the block signatures in Table 2

Bit # (*i*)	Bitslice (S_i)
1	11101
2	11110
3	11110
4	10111
5	01011
6	00011

Assume that a given text has already been broken into blocks T_i and their signatures $S(T_i)$ obtained as discussed above. A signature-tree (or S-tree as it is called in [4]) can be constructed as follows. A set of block signatures are clustered together and stored in a disk page. The number of signatures per page depends primarily on the signature size and the page size. Each entry in such a node is a signature $S(T_i)$ and points to the actual text block T_i, which, as before will be needed to check for false-positives. Note that at this point it would be trivial to process a query by simply traversing all constructed nodes, which would function as a regular signature file. Fortunately, it is possible to avoid reading many of these blocks if a tree structure is used on top of the block signature nodes.

A tree can be constructed by creating an upper layer of nodes that will point to the first layer (which will become leaves of the tree). Each entry in a non-leaf node points to a leaf node containing entries for signatures $S(T_i)$, $S(T_j)$,...$S(T_k)$, thus its entry signature will be the bitstring $(S(T_i) \mid S(T_j) \mid ... \mid S(T_k))$, where "|" denotes the bitwise-OR operator. The same reasoning can be applied recursively replacing the leaf nodes with the nodes of the current upper level. The sample tree depicted in Fig. 1, assuming an eight bit block signature and a disk page that can fit two signatures, illustrates the result of this process using the set



Signature Files. Figure 1. A sample signature-tree. The actual block signatures are at leaf level and point to the respective text block (not shown for simplicity).

of signatures at the leaf level. For instance, the entry 0011 1110 in the root node points to a node containing signatures 0011 0100 and 0011 1010 (indeed 0011 0100 | 0011 1010 = 0011 1110). Similarly the entry 0011 0100 points to the leaf node containing signatures 0010 0100 and 0001 0100.

An important issue is how to cluster signatures in the leaf nodes and, similarly how to group nodes under a single entry in the upper levels of the tree and so on and so forth until the root node. Since one of the possible criteria for this clustering task is tightly related to how the query is processed, the latter is discussed first.

Query processing starts by traversing the signature tree down from the root node choosing which subtree to traverse based on the probability that the subtree contains a candidate block. Assume that a node N has m (signature) entries N_i and it is pointed to by a parent node P under an entry with signature $P_i = (N_1 \mid N_2 \mid ... \mid N_m)$. The query starts with P_i being each of the root entries. If P_i & $h(Q) \neq h(Q)$ then it is certain that the subtree pointed by that entry cannot contain a candidate block and it is discarded. Given the way the tree is constructed the bits set are propagated from the leaf nodes up to the root entries, therefore if a given block in that subtree contained $h(Q)$ the root of that subtree entry would necessarily contain $h(Q)$. All is needed now is to repeat the same reasoning using as the new root the node pointed to by the candidate root entry. Note that all candidate subtrees need be traversed and, when finally reaching the leaf level, query refinement step is still required.

As an illustration consider the signature tree in Fig. 1 and assume $h(Q)$ = 1000 0100. Starting from the entries in the root node one can discard the upper subtree in the Figure since 0011 1110 & $h(Q) \neq h(Q)$. The lower subtree needs to be traversed given that 1101 1100 & $h(Q) \neq h(Q)$. However, at that point and the expense of one single disk access, half of the signatures can be safely discarded, illustrating the pruning power of signature trees. Of the two subtrees only the one pointed by first entry (1000 1100) needs to be read. Of the two block signatures found in the corresponding leaf nodes, only the one corresponding to signature 1000 0100 needs to be retrieved for the mandatory refinement step.

Given the query processing reasoning above if too many bits are set per block signature, the entries in the non-leaf nodes will quickly have too many bits set and therefore be unable to help pruning the traversal of the tree. If too many subtrees are traversed, the savings of not reading all signatures is bound to be offset by the additional disk seeks. Clearly the less bits are set higher up in the tree the more selective the traversal will be. This provides a criterion for cluster signatures within a node. Let $W(S_i)$ be defined as a function that returns the number of bits set in signature S_i. Clearly, for any pair of signatures S_i and S_j, $W(S_i \mid S_j) \geq W(S_i) + W(S_j)$. Thus the driving criteria for clustering signatures together is to minimize the value of $W(.)$ over the bitwise-OR'ed signatures. Just as in the case of plain signature files and bitslice signature files, a number of parameters have to be set in order to obtain efficient signature trees.

Key Applications

Text indexing and search.

Cross-references

- ▶ B+-Tree
- ▶ Disk
- ▶ File Systems
- ▶ Hash Functions
- ▶ Inverted Files
- ▶ Text Indexing and Retrieval

Recommended Reading

1. Baeza-Yates R.A and Ribeiro-Neto B.A. Modern information retrieval. ACM Press/Addison-Wesley, 1999.
2. Christos F. Access methods for text. ACM Comput. Surv., 17(1):49–74, 1985.
3. Justin Z., Alistair M., and Kotagiri R. Inverted files versus signature files for text indexing. ACM Trans. Database Syst., 23(4):453–490, 1998.
4. Uwe D. S-tree: a dynamic balanced signature index for office retrieval. In Proc. 9th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1986, pp. 77–87.
5. William B.F. and Baeza-Yates R.A. Information Retrieval: Data Structures & Algorithms. Prentice-Hall, 1992.
6. Witten I.H, Alistair M., and Bell T.C. Managing Gigabytes: Compressing and Indexing Documents and Images (2nd ed.). Morgan Kaufman, 1999.

Signatures

▶ Digital Signatures

Similarity and Ranking Operations

Michael Huggett
University of British Columbia, Vancouver,
BC, Canada

Synonyms

Association; Correlation; Matching; Proximity; Ordering; Relevance

Definition

Similarity and ranking operations are fundamental to *information searching*, in which a user generates a *query* phrase of one or more words that reflects an *information need*. The query is used to find related items that satisfy that need.

Similarity operations quantify the *resemblance* or *alikeness* between two information objects. An *information object* is a conceptual unit, most typically described as a *document*, but also taking the form of a term, phrase, paragraph, page, section, chapter, article, book, or script, etc. Information objects may be printed or digital.

Similarity judgments may be subjective (performed by a user) or algorithmic (performed by a computer). Depending on the method of evaluation, alikeness judgments may be semantic (i.e., within the meaning of a document), structural (i.e., within parts of speech, position in a document, or pattern of links between documents), or statistical (i.e., within correlations between document attributes). Statistical methods that model the distribution of terms in a corpus are most common in information retrieval (IR).

In typical IR systems, documents are preprocessed to extract representative *keyword* terms. The terms of a user's query are then compared with the keywords of each document to find the best matches. Given a document that satisfies an information need, its keywords may themselves be used to query for related documents.

Ranking operations are based on the numeric result of a similarity measure: given a query representing an information need, the documents that are scored as most similar to the query are most highly ranked. Rank is typically presented to the user as a list of document descriptors (esp. titles) sorted monotonically in decreasing order of similarity score.

Historical Background

The history of information retrieval is parallel to and largely separate from that of database research, since the goal of IR is to find semantically related information in an arbitrary corpus of unstructured documents, whereas the traditional relational database model searches within specified value ranges in pre-defined fields. Information retrieval produces "best guess" matches for a given keyword query, whereas relational databases return items that are *true* of a (typically boolean) query statement. Information retrieval is uncertain and probabilistic; its similarity and ranking operations are vital to this distinction.

Notions of similarity and ranking are fundamental to every-day information needs, as intelligent agents (e.g., people, animals) define target objects (e.g., food) that meet certain goals (e.g., survival), and then seek objects in the environment that meet these goals. As such, the topics of similarity and ranking have been well-discussed by cognitive scientists who study the structure of knowledge (e.g., Tversky, 1975), and before them by philosophers from the Enlightenment (e.g., Hume) back to Ancient Greece (e.g., Aristotle).

By the start of the Information Age in the mid-20th century, discrete mathematics was already the basis of information retrieval research. Proposed techniques included the comparison of attribute lists between documents, called *association* (Yule, 1912); ranking as a process of ordinal measurement (Stevens, 1946); *distance metrics* that quantify the differences between strings, originally used for error detection (Hamming, 1950); the automatic indexing of documents based on the statistical distribution of their terms (Luhn, 1957); and the weighting of terms to improve retrieval accuracy (Maron and Kuhns, 1960).

Beginning in the 1960s, these techniques were refined and combined into the *associative* and *probabilistic* approaches that form the basis of current practice—although earlier explorations can claim to have produced the first operational information retrieval system (e.g., Goldberg's *Statistical Machine* as the first electronic system, c. 1927, and Tillett's *QUEASY* as the first system on a general-purpose computer, c. 1953).

The associative approach was consolidated and evaluated exhaustively in the first viable product of modern information retrieval: the SMART system (Salton, 1966), which continues to be much imitated. The probabilistic approach thereafter established a solid theoretical foundation that had been lacking in earlier work, and over a series of refinements improved retrieval accuracy to become a standard benchmark (Robertson & Sparck, 1976). Some researchers believe the probabilistic approach represents the future of information retrieval.

Foundations

Similarity refers primarily to alikeness based on shared attributes; it is sometimes described as *association*, *relatedness* or *relevance*. Computational similarity operates by comparing lists of representative terms derived from information objects. Comparisons are either between a query and document, or between two documents. In the latter case, if a user has an exemplar document in hand that represents an information need, that exemplar may be used to "give me more like this one".

Automatic similarity and ranking operations require information objects in digital form; paper documents may be digitized using optical character recognition (OCR). Similarity comparisons are necessarily automatic in systems that index and organize large corpora of documents. Some systems anticipate significant human interaction, but may not be practical since human similarity valuations are slower, and inter-evaluator consistency is poor.

Information retrieval systems typically encode documents as *term vectors*. Each document in a corpus is assigned a term vector. Words in the document are typically *stemmed* to pool together related words such as *stemmer*, *stemmed*, *stemming*, etc. under a single unique root term, e.g., *stem*. Each unique term is assigned a specific cell in the term vector that contains a numerical value. *Binary vectors* show a '1' in a cell if the document contains the term, '0' otherwise. *Weighted vectors* use either integer values to record the number of times each term appears in the document, or real-number values that express the degree to which the term is representative of the document. All the term vectors in the same corpus use the same (binary or weighted) numbering scheme.

All the term vectors in the same corpus are configured the same way. Two types of configuration are typical. The first type reserves a cell for each unique term that appears in the corpus, and numerical values are assigned only to cells for terms that appear in the document. In a large corpus, document vectors can contain many zeroes, and thus be inefficiently *sparse*.

The second type of configuration involves the use of document *keywords*: a small set of discriminating terms from within the document that represent its content. Keywords are fundamental to *indexing*, whose goal is to identify terms that best identify a document with respect to other documents in a corpus. Using a term-weighting algorithm, a score is calculated for each term in each document, and each document is assumed to be best described by its highest-scoring terms. These terms are chosen as the document's keywords and low-scoring terms are ignored.

The use of such automatically-extracted keywords provides some advantages. Similarity scores can be calculated more efficiently with a small number of representative keywords than by using all the terms in a document. Keywords also provide a human-readable summary of document content without requiring that all documents be read and evaluated manually. Keywords require less space for term storage: a document term vector is a compact mapping from each keyword to its numerical value. Weighted keywords are standard in IR; more complicated methods than this seldom justify the additional complexity and difficulty (Cleverdon, Mills, & Keen 1966).

Ranking refers to any function that follows the *Probability Ranking Principle*, which states that IR systems are most effective when the documents that they retrieve in response to a query are ordered in decreasing probability of relevance to the user's information need. Ranking is always performed as a result of some user-generated query.

Associative Similarity

In associative similarity, document term vectors define a metric space: a corpus with *n* unique keywords can be described as a Euclidean *n*-space, in which each document vector describes a point, line, or hyperplane. Together, the document vectors of a corpus comprise a matrix subject to algebraic methods for space division and summarization. Associative similarity is also commonly known as the *vector space model*.

The associative approach finds documents most similar to a query by matching the terms of the query vector against all of the document term vectors in the corpus. The matching process typically uses an index to retrieve candidate document vectors. The documents that generate the highest similarity scores are assumed to be those that best meet the user's information need as represented by the query. Documents are then presented to the user in a ranked list sorted in decreasing order of similarity score.

There are several associative vector methods that can be applied to binary and weighted document vectors to generate a similarity score. The simplest method counts the number of found terms in common between two binary vectors: this produces a *simple matching coefficient*, which is useful under the assumption that *any* matching is important, for example when two objects are assumed to be incomplete descriptions of each other. In practice, this assumption proves to be coarse and inaccurate, and most similarity operations normalize the simple matching coefficient by the lengths of the input vectors. For weighted vectors, the inner product of two vectors is used as the equivalent of the simple matching coefficient, and sum-squares of the vectors are used for normalization. Table 1 shows binary and weighted versions of four common similarity measures. The weighted cosine method is especially popular, and represents a geometrically accurate interpretation of correlation.

In practice, these methods are virtually interchangeable, since all their scores increase monotonically given the same input data. However, it is notable that for a given method and query vector, multiple target documents with significantly different term vector compositions may generate the same similarity score. Furthermore, the effective domain of similarity scores in *n*-dimensional space can vary dramatically between different similarity functions. Such results suggest that different similarity operations may be more or less appropriate for corpora with different types of term distribution (Jones & Furnas, 1987) [2].

Probabilistic Retrieval

Probabilistic retrieval is a decision-theoretic process based on the idea that for each query, documents should be retrieved if they maximize the probability that they are *relevant* to the query, $P(relevance|document)$. Whereas associative similarity measures are largely ad hoc, probabilistic retrieval is based on an explicit theoretical framework. The intuition is that users will find documents relevant if the documents have a certain distribution of attributes, and that the probability of relevance depends on how closely a document's distribution of attributes matches the distribution sought by the user.

Since the probabilistic model depends on the presence or absence of terms, documents are represented with binary term vectors, writing a '1' in each cell that a given term is present, '0' otherwise. In its simplest form, the model assumes that errors of false-positive choices (deemed relevant when irrelevant) and false-negative choices (deemed irrelevant when actually relevant) are negligible.

For a given document represented by term vector X, probability of relevance $P(R)$ and probability of irrelevance $P(\bar{R})$, the document is judged relevant if $P(R|X) > P(\bar{R}|X)$, i.e., if the following discriminant function $g(X)$ is greater than 1:

$$g(X) = \frac{P(R|X)}{P(\bar{R}|X)}$$

Using Bayes' Law, this is rewritten as:

$$g(X) = \frac{P(X|R)P(R)}{P(X|\bar{R})P(\bar{R})} \approx \frac{P(X|R)}{P(X|\bar{R})}$$

since $P(R)$ and $P(\bar{R})$ are constant for each document. The calculation of $P(X|R)$ and $P(X|\bar{R})$ depend on the probabilities of each term in the document. For each term, the following table is used, with N the number of documents in the corpus, R the number of documents relevant to the query, n the number of documents that contain the term, and r the number of relevant documents that contain the term:

	Relevant	Non-relevant	
documents w/ term	r	$n-r$	n
documents w/o term	$R-r$	$N-n-(R-r)$	$N-n$
	R	$N-R$	N

Thus for each binary term x_i in X that matches a term in the query, the relevance function is re-written as a ratio of relevant to non-relevant portions:

$$g(X) \approx \frac{P(X|R)}{P(X|\bar{R})} = \frac{\prod_i P(x_i|R)}{\prod_i P(x_i|\bar{R})}$$

$$= \sum_i x_i log \frac{r_i/(R-r_i)}{(n-r_i)/(N-n-R+r_i)}$$

This simplifies to $g(x) \approx \sum_i x_i log \frac{N-n_i}{n_i}$ in the absence of relevance information, and approximates further to

Similarity and Ranking Operations. Table 1. Four popular methods for calculating the similarity between two objects. The binary vectors use a simple count of matching keywords, given their attribute vectors *X* and *Y*. The weighted vectors hold term weights calculated by an indexing algorithm; (w_X, w_Y) indicates the inner product of the vector weights, and $\Sigma {w^2}_A$ indicates the sum of squares of all weights of vector *A*.

Binary vectors		Weighted vectors	
Jaccard $\frac{\lvert X\cap Y\rvert}{\lvert X\cup Y\rvert}$	Dice $\frac{2\lvert X\cap Y\rvert}{\lvert X\rvert+\lvert Y\rvert}$	Jaccard $\frac{(w_X,w_Y)}{\Sigma w_X^2+\Sigma w_Y^2-\Sigma(w_X w_Y)}$	Dice $\frac{2(w_X,w_Y)}{\Sigma w_X^2+\Sigma w_Y^2}$
cosine $\frac{\lvert X\cap Y\rvert}{\sqrt{\lvert X\rvert^2\times\lvert Y\rvert^2}}$	overlap $\frac{\lvert X\cap Y\rvert}{min(\lvert X\rvert,\lvert Y\rvert)}$	cosine $\frac{(w_X,w_Y)}{\sqrt{\Sigma w_X^2 \Sigma w_Y^2}}$	overlap $\frac{(w_X,w_Y)}{min\left(\Sigma w_X^2,\Sigma w_Y^2\right)}$

$g(x) \approx \sum_i x_i log \frac{N}{n_i}$ for $n_i \ll N$, noting that the log factor in this last equation is identical to the common form for *inverse document frequency* (IDF). Once $g(X)$ is calculated, documents can be ranked by their relevance score.

The crucial factor in probabilistic retrieval is how to estimate *relevance*. The problem is that *R* is initially unknown, since there are no retrieved documents. To gain a notion of what may actually satisfy a user's information need, a common technique has been to estimate term relevance based on distributions of terms in the corpus. One approach is to simply use term IDF scores, and to assume that all query terms have an equal probability of appearing in relevant documents. Another approach is to use a mixture of two Poisson distributions to characterize the distribution of each term among relevant and irrelevant documents in the corpus.

Once the model has made some retrievals based on these relevance estimates, the user can provide *relevance feedback* by stating explicitly whether a retrieved document is relevant or irrelevant. This feedback is used to refine the model's term weight parameters incrementally. Relevance feedback is a direct approach to system personalization, but may not be practical with large corpora, or with systems with many (perhaps anonymous) users with different information needs. Relevance feedback also imposes a *burden of decision* that some users may wish to avoid, and its use reminds us that information retrieval is not a purely objective science: it is subject to often ill-defined information need. Users themselves may not be aware of their goal or able to describe their need, other than by a vague feeling of relevance.

Although probabilistic retrieval has been described here in its simplest form, further developments have addressed the troublesome assumption of *term independence* endemic to information retrieval, and accuracy of relevance scores has been improved by incorporating parameters for document length and *term frequency*.

Related Areas

Information retrieval deals primarily with textual information objects, although the advent of digital images and videos has led to techniques for quantifying the alikeness of non-textual media. Traditional textual similarity and ranking operations may be applied to non-textual media if they can first be interpreted into textual-symbolic form, such as by using machine-learning techniques to generate keyword descriptions of photographs. Whatever the source media, if symbolic (esp. alphanumeric) descriptive attributes can be extracted in a robust, consistent manner, then the media can be compared based on those attributes.

Alternatively, where many users share a common information space, they may add their own *tags* to information objects. Tagging is a form of *social filtering* where semantic relations between objects emerge from the collective valuations of a group of interested individuals. The resulting *tag cloud* acts as an organic index, preferentially retrieving objects strongly represented by a conjunction of query terms.

The World Wide Web, with its search engines, is by far the most popular information retrieval system: a text-based information medium with many millions of users and linked pages. Correspondingly, it has become an increasingly popular domain for information retrieval research and development. Although the term-based similarity methods discussed above can be used to compare the content of documents, other effective methods have taken advantage of the semantic collaboration that gives the Web its structure: the many Web authors who link to pages that they consider relevant to their own. Web ranking algorithms (such as HITS and PageRank) observe which pages are highly-connected *hubs* and *authorities*; these pages are retrieved preferentially for a given query.

It is often useful to calculate the similarity between documents of a corpus, whether to find documents related to an exemplar document, or as a prelude to clustering operations. Since calculating similarity scores on demand in a large corpus can be expensive (particularly where corpus-wide sums are necessary, as with *TF*IDF*), pair-wise similarity scores between documents may be stored in memory for later rapid retrieval. One approach is to create a similarity matrix—triangular if the similarity property is assumed to be commutative, but twice as large otherwise.

Another common approach is to store relations between documents in an *associative similarity network* (ASN). Documents act as the nodes of a network; links between pairs of documents represent relatedness, and are typically weighted with a real-valued similarity score. As many remotely-related documents could be linked with trivial weights, nodes are only linked for similarity scores above some threshold. In addition to fast nearest-neighbour retrieval, an ASN also provides opportunities for interactive navigation through the network, following a *semantic gradient descent* moving toward clusters of documents that better support a user's information need. ASNs can be seen as an example of the *cluster hypothesis*, which states that similar documents will be relevant to the same query: after a document has been retrieved with a term-based query, other documents similar to that document are already linked and immediately available for further inspection.

Key Applications

Classification; Thesaurus construction; Recommendation systems; Similarity (nearest-neighbour) search; Summarization and Information Filtering.

Cross-references

- ▶ BM25
- ▶ Classification
- ▶ Clustering for Post-Hoc Information Retrieval
- ▶ Dimensionality Reduction
- ▶ Index Creation and File Structures
- ▶ Indexing and Similarity Search
- ▶ Multimedia Information Retrieval
- ▶ Probability Ranking Principle
- ▶ Relevance Feedback for Text Retrieval Model
- ▶ Stemming
- ▶ Term Weighting
- ▶ TF*IDF
- ▶ Web Information Retrieval Models

Recommended Reading

1. Harman D. Ranking algorithms. In Information Retrieval: Data Structures & Algorithms, Chap. 14, W.B. and Frakes R. Baeza-Yates Prentice-Hall, Upper Saddle River, NJ, 1992, pp. 363–392.
2. Jones W.P. and Furnas G.W. Pictures of relevance: A geometric analysis of similarity measures. J. Am. Soc. Inf. Sci., 38(6):420–442, 1987.
3. Rasmussen E.M. Clustering algorithms. In Information Retrieval: Data Structures & Algorithms, Chap. 16, W.B. Frakes and R. Baeza-Yates (eds.) Prentice-Hall, Upper Saddle River, NJ, 1992, pp. 419–442.
4. Salton G. and McGill M. An Introduction to Modern Information Retrieval. McGraw-Hill, New York, NY, 1983.
5. van Rijsbergen C.J. Information Retrieval. Butterworths, London, 1979.

Similarity in Video

▶ Video Querying

Similarity Measure

▶ Image Retrieval

Similarity-based Data Partitioning

▶ Database Clustering Methods

Simplicial Complex

ANDREW U. FRANK
Vienna University of Technology, Vienna, Austria

Synonyms

CW complex; Polyhedron; Cell complex

Definition

A simplicial complex is a topological space constructed by gluing together dimensional simplices (points, line segments, triangles, tetrahedrons, etc.).

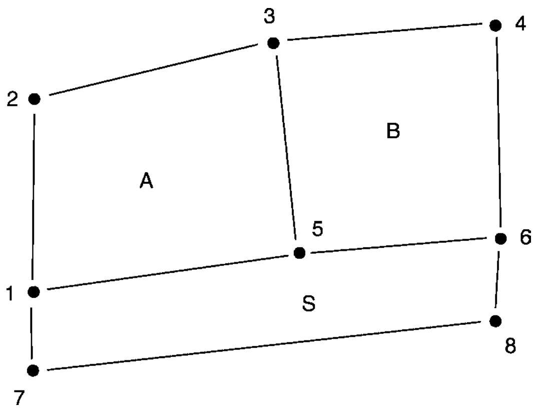
A simplicial complex *K* is a set of simplices *k*, which satisfies the two conditions:

1. Any face of a simplex in *K* is also in *K*
2. The intersection of any two simplices in *K* is a face of both simplices (or empty)

Historical Background

Raster (field) or vector (object) are the two dominant conceptualizations of space. Applications focusing on object with 2 or 3 dimensional geometry structure the storage of geometry as points, lines, surfaces, and volumes and the relations between them; a classical survey paper discussed the possible approaches mostly from the perspective of Computer Aided Design (CAD) where individual physical objects are constructed [10].

The representation of geographic information, e.g., maps, introduces consistency constraints between the objects; consider the sketch of a few cadastral parcels (lots) and the adjoining street (Fig. 1). Land, in this case 2 dimensional space, is divided into lots, such that the lots do not overlap and there are no gaps between them; this is called a partition (definition next section). Corbett [2] proposed to check that a sequence of line segments around a face closes and that the left neighbor of line segment and the right neighbors of the following line segment around a point is the same face; these two conditions are dual to each other (Fig. 2). This duality is the foundation of the DIME (dual independent map encoding) schema to store 2D line geometry for areas.



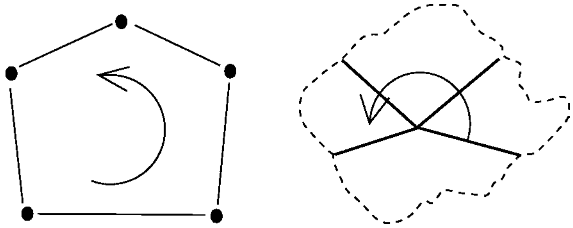
Simplicial Complex. Figure 1. Cadastral parcels provide an example of a simplicial complex.

Every line of a graph, which represents a partition, is related to a start and an end point and to two adjacent faces (Fig. 3). Such data structures were typical for the 1980s; implemented originally with network and later relational DBMS. They did not perform acceptably fast with large Geographic Information System data, mostly because geometric operations do not translate to database operations directly (the so-called impedance mismatch of record oriented programming and tuple oriented database operations [7]), most obvious when checking geometric consistency. As late as 1985, all commercial programs to compute the overlay of two partitions, which is one of the most important operations in geographic information processing, failed.

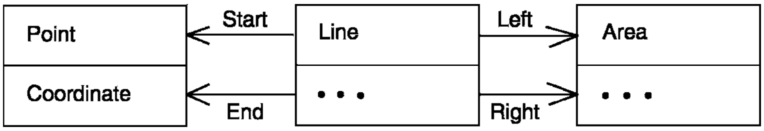
In 1986, Frank observed that simplicial (and possibly cell) complexes enforced exactly the consistency constraints required by the large class of applications that manage geometry as 2D or 3D partitions [5]. A commercial implementation became available, designed concurrently by Herring (then with Intergraph). Alternative approaches to manage the geometry of partitions without explicit representation of topology and to reconstruct topology when required were often used, but cause difficulties, because of the fundamental limitations of approximative numerical processing.

Foundations

Topology, specifically the theory of homotopy, provides the mathematical theory to program geometric operations. Homotopy captures the notion that multiple metric (coordinative) descriptions of a single geometry may be different but represent "essentially" the



Simplicial Complex. Figure 2. The two consistency checks: following the line segments around a face and following the line segments around a point.



Simplicial Complex. Figure 3. An UML object diagram for a database schema for partitions.

same geometry. Figure 1 can be transformed continuously to Fig. 4 but not to Fig. 5.

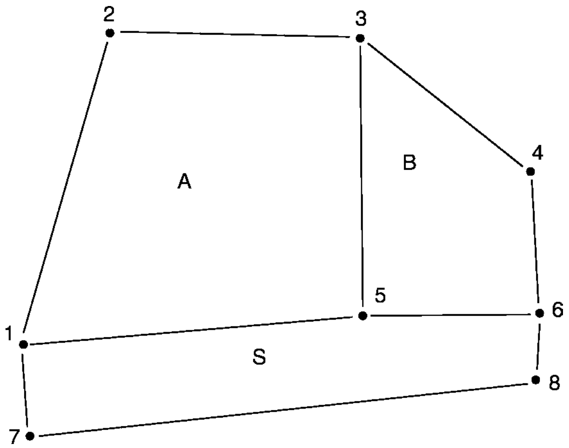
Homotopy creates equivalence classes for geometric figures. Many applications are interested in exactly these equivalence classes and benefit from the achieved abstraction that leaves out imprecisions caused, e.g., by measurements or approximative numerical processing.

Topology studies the invariants of space under continuous (homeomorphic) transformations, which preserve neighborhoods. Algebraic topology, also called combinatorial topology [1], studies invariants of spaces under homotopy with algebraic methods. The perspective of point set topology, which sees geometric figures as (infinite) sets of points is not practical for programming and the discretization of geometry achieved through algebraic topology is crucial: the unmanageable infinite sets are converted into countable objects, namely points, lines between points, and faces bounded by the boundary lines. Algebraic topology studies different "spaces" like Fig. 4 and Fig. 5 (both are embedded in ordinary 2D space, but the embedding is not in focus in algebraic topology).

The complexity of operations on arbitrary cells of a partition can be reduced by forcing a triangulation; all elements are then convex! Figure 1 is a cell complex and the corresponding simplicial complex is Fig. 6.

Algebraic topology studies simplices and their relations: A simplex is the simplest geometric figure in each dimension. A zero dimensional simplex (0-simplex) is a point, a one dimensional simplex (1-simplex) is a straight line segment, a two dimensional simplex (2-simplex) is a triangle, a three dimensional simplex (3-simplex) a tetrahedron, etc. n + 1 points in general position define an n-simplex. Each n-simplex consists of (is bounded) by $(n+1)$ $(n-1)$-simplexes: a line (1-simplex) is bounded by 2 0-simplices (points), etc. Simplices can be oriented; the oriented 1-simplex 2–3 (in Fig. 1) is different from the oriented 1-simplex 3–2 (Fig. 7).

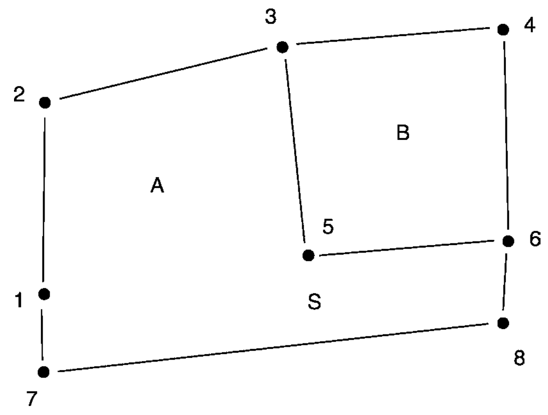
A k-simplicial complex K is a complex in which at least one simplex has dimension k and none a higher dimension. A homogeneous (or pure) k-complex K is a complex in which every simplex with dimension less than k is the face of some higher dimension simplex in K. For example, a triangulation is a homogeneous 2-simplicial complex, a graph is a homogeneous 1-simplicial complex. Homogeneous simplicial complexes are models of partitions of space and used therefore to model geographic spatial data. Whitehead gave for so-called CW-complexes a slightly more general, more categorical definition mostly used in homotopy theory.



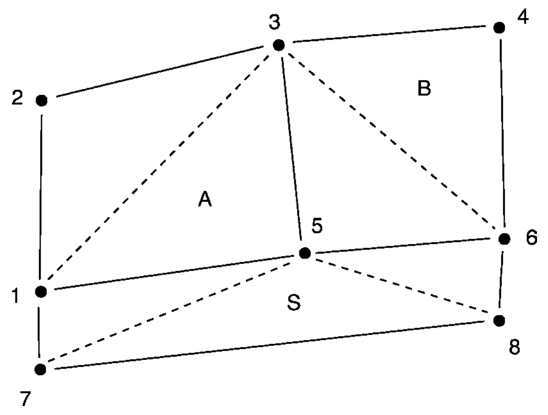
Simplicial Complex. Figure 4. A deformed, but homotopic, copy of Fig. 1.

Four operations are important for simplicial complexes: the *closure* of a set of simplices S is the smallest complex containing all the simplices; it contains all the faces of every simplex in S. The *star* of a set of simplices S is the set of simplices in the complex that have simplices in S as faces. The *link* of a set of simplices S is a kind of boundary around S in the complex. The *skeleton* of simplicial complex K of dimension k is the subcomplex of faces of dimension k-1 in K.

Simplicial complexes can be represented as chains, which are lists of the ordered simplices included in the complex. Chains can be written as polynomials with integer factors for the simplices included in the complex, e.g., the 2-chain of the 2-complex in Fig. 1 is $K = 1\ \ A + 1\ \ B + 1\ \ S$.



Simplicial Complex. Figure 5. Metric is preserved, but the figure is not homotopic to Fig. 1, because elements are missing.



Simplicial Complex. Figure 6. The geometry of Fig. 1 triangulated.

The boundary operator δ applied to a k-simplex gives the set of k-1-simplices, which form the boundary of the simplex; for example, the boundary of a 1-simplex gives the two 0-simplices, which are start and end point of the line, one taken with positive, the other with negative orientation. The boundary operator is applied to a chain by applying it to every oriented simplex in the chain. The boundary of a closed simplicial complex is 0; in general, the boundary of the boundary is 0.

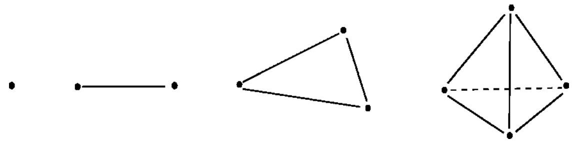
$$\begin{aligned}
\mathrm{\delta A} &= l_{12} + l_{23} + -l_{15} \\
\delta(\delta \mathrm{A}) &= \delta l_{12} + \delta l_{23} + \delta l_{35} - \delta l_{15} \\
&= p_r - p_2 + p_2 - p_3 - p_s - p_r + p_s \\
&= 0
\end{aligned}$$

The boundary operator is important to deduce the topological 4- and 9-intersection (Egenhofer) relations between two subcomplexes, of the same complex [3,4]. Chains and boundary operator are easy to implement with list operators and often it is sufficient to generalize the code for operations on polynoms.

The theory of simplicial complexes can be generalized to cell complexes. Cells are homomorph to simplices, but can have arbitrary form; a 2-cell can have an arbitrary number of nodes in its boundary.

From an application point of view, it is often important that objects do not overlap and all of space is accounted for. The concept of a *partition* captures this idea; a partition of a space S is a set of subsets of the space, such that

- All subsets cover all of space (jointly exhaustive): $\bigcup_i s_i = s$



Simplicial Complex. Figure 7. The simplices of 0, 1, 2, and 3 dimensions.

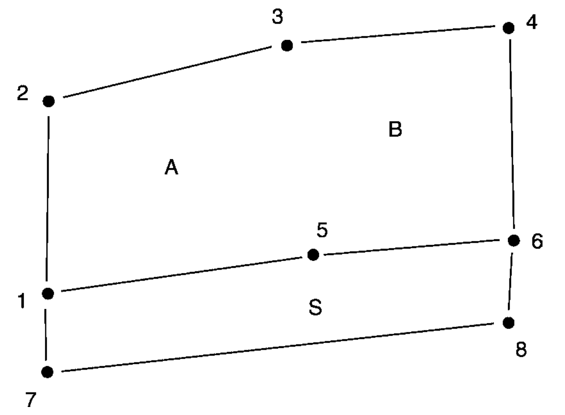
- No two subsets overlap (pairwise disjoint): $s_i \cap s_j = \emptyset$ *for* $i \neq j$.

These two properties are sometimes abbreviated as JEPD.

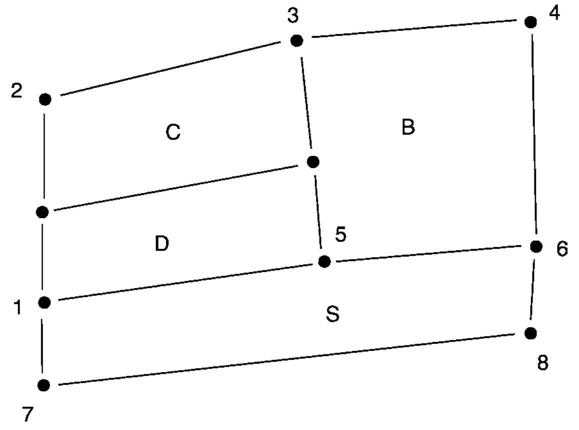
Partitions are changed by the Euler operations, *glue* and *split*, which maintain the Euler characteristic of the surface; the Euler characteristic is computed as $\chi = V - E + F$, where V is the number of nodes (vertices), E is the number of edges and F is the number of faces. From Fig. 1 with $\chi = 8–10 + 3 = 1$ merging two parcels obtains Fig. 8 with $\chi = 8–9 + 2 = 1$ or Fig. 9 where parcel A is split into parcel C and D with $\chi = 10–13 + 4 = 1$.

Consistency of these operations is difficult to check in cell complexes if "islands" occur as in Fig. 10, which is realistic for many application areas. The problem is avoided by triangulation and therefore simplicial complexes are an effective representation for maintainable geometric data describing partitions.

Simplicial complexes are triangulations of 2 dimensional space; they contain more objects than a partition represented as cells, but operations to maintain consistency in a triangulation are faster and simpler to program. The representation of a simplicial or cell complex



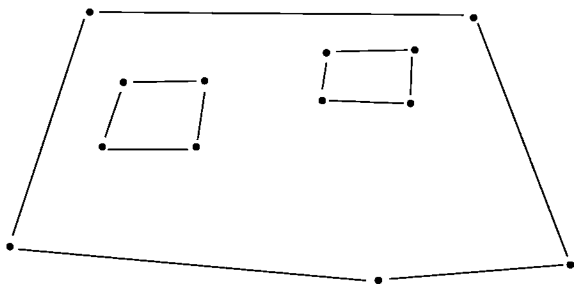
Simplicial Complex. Figure 8. A and B merged.



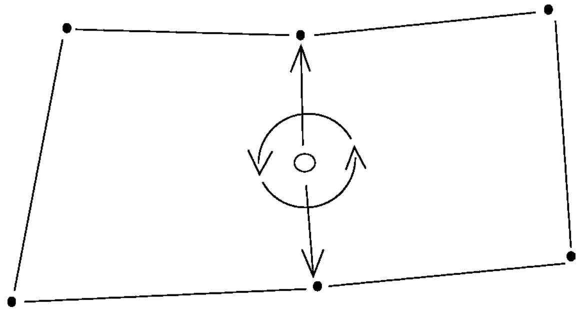
Simplicial Complex. Figure 9. A subdivided in C and D.

requires the explicit representation of the boundary and converse co-boundary relation. The schema used initially (Fig. 3) contains redundancy (which is used in Corbett's tests for consistency) and is therefore difficult to maintain. Popular today are schemes with half edges (Fig. 11), where a half-edge points to the starting node and the corresponding other half edge or quad edges [6] (Fig. 12), where each quad-edge points to the next quad-edge and either a boundary node or face; in a quad-edge structure, the boundary graph and its dual are maintained in a well-defined algebra with a single operation *splice*. For example, taking Fig. 1 as a boundary graph (primal) the dual is Fig. 13, which shows adjacency between faces.

Quad edges represent efficiently without redundancy a much larger universe, namely partitions of orientable manifold. The Euler operations *glue* and *split* can be efficiently implemented and maintain a simplicial or cell complex. The geometry can be represented as



Simplicial Complex. Figure 10. A parcel with "islands."



Simplicial Complex. Figure 11. Two half edges, pointing to adjacent nodes.

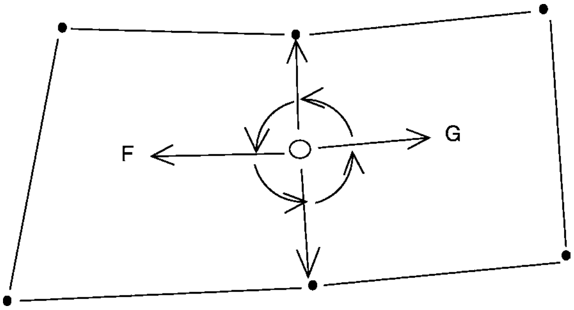
generalized maps, for which efficient implementation using relational databases has been reported [9].

Key Applications

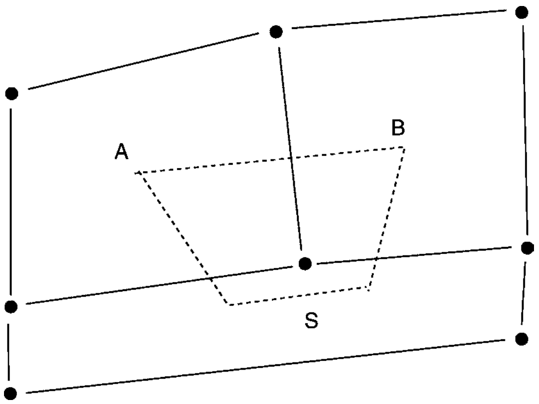
Many applications include geometric descriptions of objects; Computer Aided Design for mechanical and civil engineering are important, but also Geographic Information Systems, with many special applications like Utility Mapping for cities, Cadastral Maps to show ownership of land, but also car navigation systems, are popular examples.

Management of partitions is central for Geographic Information Systems (GIS); 2D partitions are wisely used for land ownership parcels, soil types, etc. Increasingly 3D models of cities and buildings are built to produce visualizations for virtual trips. Town planning applications expect that changes in 3D models over time can be visualized, which requires 4 (3 spatial plus one temporal) dimensions.

Management of the geometry of partitions of 3D space is important for CAD (Computer Aided Design), used for architecture, civil engineering but also mechanical engineering. Image processing intended to



Simplicial Complex. Figure 12. Four quad edges give one edge and point to adjacent nodes and faces.



Simplicial Complex. Figure 13. The dual graph of Fig. 1 (dashed) shows the neighbor relations.

produce 3D representations of the environment is using hierarchically structured partitions and needs effective operations to subdivide these.

A generalizable approach to storing and maintaining geometry in a database integrates for many application areas the treatment of geometric data with other data. Approaches based on the theory of simplicial or cell complexes are now available as plug-ins to convert general purpose DBMS to spatial databases. They replace earlier systems where geometric data was managed in proprietary file structures and the connection between geometry and descriptive data established only in the application program.

Future Directions

Besides efforts to enhance the performances of implementations three major research goals stand out:

1. Efficient solutions for 3D data; required for example to build 3D city models and to construct operations for consistently updating these [12]
2. Generalization to *n*-dimensions to include temporal data, especially 2 and 3 dimensional geometry and time required to include time related data, movement and, in general, processes in CAD and GIS applications [11]
3. Hierarchical structures to have partitions at one level of resolution (e.g., countries of the world) and then allow subdivision (e.g., regions, departments, counties, towns) [13]

A fully general application independent, *n*-dimensional and hierarchical representation that supports Euler operations effectively within data stored in a database is the implied goal of research in the first decade of the twenty-first century.

Cross-references

▶ Geographic Information System
▶ Topological Data Models
▶ Topological Relationships

Recommended Reading

1. Alexandrov P.S. Combinatorial Topology Volumes 1, 2 and 3. Dover Publications, Inc., Mineola, New York, 1960.
2. Corbett J.P. Topological Principles in Cartography, Bureau of the Census, US Department of Commerce, 1979.
3. Egenhofer M. and Herring J.R. A mathematical framework for the definition of topological relationships. In Proc. Fourth Int. Symp. on Spatial Data Handling, 1990.
4. Egenhofer M.J. and Franzosa R.D. On the equivalence of topological relations. Int. J. Geogr. Inf. Syst., 9(2):133–152, 1995.
5. Frank A.U. and Kuhn W. Cell graph: a provable correct method for the storage of geometry. In Proc. Second Int. Symp. on Spatial Data Handling, 1986.
6. Guibas L.J. and Stolfi J. A language for bitmap manipulation. ACM Trans. Grap., 1(3):191–214, 1982.
7. Härder T. New approaches to object processing in engineering databases. In Proc. 1986 Int. Workshop on Object-Oriented Database Systems, 1986.
8. Levin B. Objecthood: an event structure perspective. In Proc. Chicago Linguistic Society 35, 1999, pp. 223–247.
9. Lienhardt P. Extensions of the notion of map and subdivision of a three-dimensional space. In Proc. 5th Annual Symp. on Theoretical Aspects of Computer Science, 1988.
10. Requicha A. Representation for rigid solids: theory, methods and Systems. ACM Comp. Surv., 12(4):437–464, 1980.
11. Sellis T, et al. (eds.) Spatiotemporal Databases: The Chorochronos Approach. LNCS, vol. 2520. Springer, Berlin, 2003.

12. Thompson R.J. Towards a Rigorous Logic for Spatial Data Representation. Doctoral thesis, Delft, NCG, 2007.
13. Timpf S. Hierarchical Structures in Map Series. Ph.D thesis, Technical University Vienna, Vienna, 1998.

Simulated Data

▶ Synthetic Microdata

Single Instancing

▶ Deduplication

Single Instruction Multiple Data (SIMD) Parallelism

▶ Intra-operator Parallelism

Singular Value Decomposition

Yanchun Zhang, Guandong Xu
Victoria University, Melbourne, VIC, Australia

Synonyms

SVD transformation; Latent semantic indexing; Principle component analysis

Definition

The SVD definition of a matrix is illustrated as follows [1]: For a real matrix $A = [a_{ij}]_{m\times n}$, without loss of generality, suppose $m \geq n$ and there exists SVD of A (shown in Fig. 1):

$$A = U\begin{pmatrix}\Sigma_1\\ 0\end{pmatrix}V^T = U_{m\times m}\sum{}_{m\times n}V^T_{n\times n}$$

where U and V are orthogonal matrices $U^TU = I_m, V^TV = I_n$. Matrices U and V can be respectively denoted as $U_{m\times m} = [u_1, u_2,\ldots,u_m]_{m\times m}$ and $V_{n\times n} = [v_1, v_2,\ldots,v_n]_{n\times n}$, where $\mathrm{u_i}, (i = 1,\ldots,m)$ is a m-dimensional vector $u_i = (u_{1i}, u_{2i},\ldots,u_{mi})^T$ and $v_j, (j = 1,\ldots,n)$ is a n-dimensional vector $v_j = (v_{1j}, v_{2j},\ldots,v_{nj})^T$. Suppose rank(A) = r and the single values of A are diagonal elements of Σ as follows:

$$\sum = \begin{bmatrix}\sigma_1 & 0 & \cdots & 0\\ 0 & \sigma_2 & \ddots & \vdots\\ \vdots & \ddots & \ddots & 0\\ 0 & \cdots & 0 & \sigma_n\end{bmatrix} = diag(\sigma_1, \sigma_2, \cdots \sigma_m),$$

where $\sigma_i \geq \sigma_{i+1} > 0$, for $1 \leq i \leq r-1$; $\sigma_j = 0$, for $j \geq r+1$, that is

$$\sigma_1 \geq \sigma_2 \geq \cdots \sigma_r \geq \sigma_{r+1} = \cdots = \sigma_n = 0$$

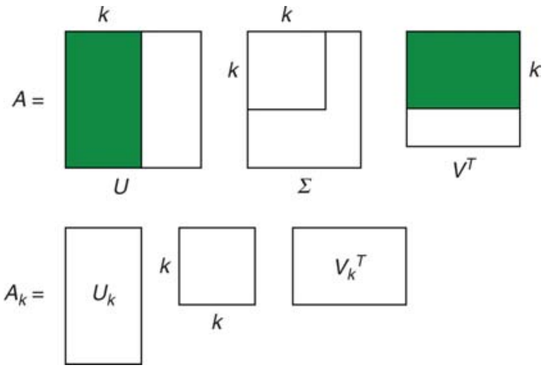
For a given threshold ε $(0<\varepsilon<1)$, choose a parameter k such that $(\sigma_k - \sigma_{k+1})/\sigma_k \geq \varepsilon$. Then, denote $U_k = [u_1, u_2,\ldots,u_k]_{m\times k}$, $V_k = [v_1, v_2,\ldots,v_k]_{n\times k}$, $\sum_k = diag(\sigma_1, \sigma_2,\ldots,\sigma_k)$, and

$$A_k = U_k\sum{}_k V_k^T$$

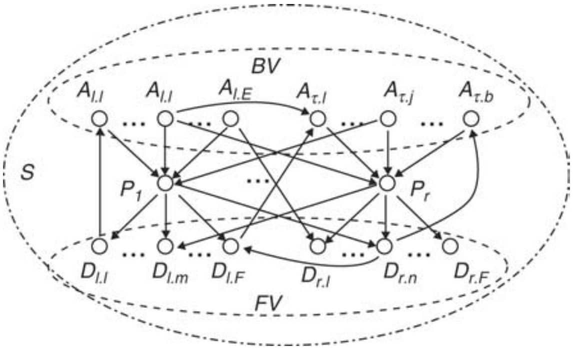
As known from the theorem in algebra [1], A_k is the best approximation matrix to A and conveys main and latent information among the processed data. This property makes it possible to find out the underlying semantic association from original feature space with a dimensionality-reduced distance computational cost, in turn, is able to be used for latent semantic analysis.

Key Points

Singular Value Decomposition (SVD) algorithm could be considered as a useful means for applications of data engineering and knowledge discovery, such as Web search, image and document retrieval and Web data mining. For example, finding the closely relevant pages to a given page can be carried out by manipulating



Singular Value Decomposition. Figure 1. SVD transformation of matrix and its approximation.



Singular Value Decomposition. Figure 2. Page source structure for the given page *u*.

SVD operation on a constructed page source to reveal the latent linkage relationships from them [3]. In order to avoid high cost of similarity computations and keep the minimum loss of linkage information contained in the feature space, SVD algorithm is applied to not only reduce the dimensionality of original feature, which leads to less computational costs, but also capture the semantic similarity among web pages, which is unseen intuitively. The algorithm is working as follows: (i) construct a web page space A (page source) for the given page u from link topology on the web. The page source is represented as a directed graph with edges indicating hyperlinks and nodes standing for web objects (shown in Fig. 2); (ii) since the topological relationships amongst the page source is expressed in a linkage matrix, manipulating SVD results in decomposition of original feature space $A = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}$; (iii) From the SVD theorem, the best approximation matrix A_k contains main linkage information among the pages, and makes it possible to filter those irrelevant pages; (iv) by selecting a threshold of similarity, the relevant pages in page source to the given page are eventually found.

While SVD algorithm is usually used in conventional latent semantic analysis (LSA) techniques, some variants of LSA have been proposed recently in the context of Web information processing and text mining. Apart from the difference at theoretical formulation, the common characteristics of these methods are to map the original feature space, which is utilized to model the co-occurrence observation, into a new dimensionality-reduced feature space, and maintain the maximum approximation of the original feature distance with the converted feature space. For example, Probabilistic Latent Semantic Analysis (PLSA) model is an representative of such kinds of approaches [2]. For instance, [3] proposed a PLSA-based Web usage mining approaches for Web recommendation. In this collaborative recommendation scheme, user task-oriented access patterns are extracted from Web log files, in turn, are used to predict user's likely interested Web content via referring the navigational preference of other users, who exhibit like-minded access task.

Cross-references

▶ Database Clustering Methods
▶ Principal Component Analysis

Recommended Reading

1. Datta B. Numerical Linear Algebra and Application. Brooks/Cole Publishing Company, Pacific Grove, CA, 1995.
2. Hofmann T. Latent semantic models for collaborative filtering. ACM Trans. Inf. Syst., 22(1):89–115, 2004.
3. Zhang Y., Yu J.X., and Hou J. Web Communities: Analysis and Construction. Springer, Berlin, 2006.

Sketch

▶ AMS Sketch
▶ Structure Indexing

SMI-S

▶ Storage Management Initiative-Specification

Snapshot

▶ Point-in-Time Copy (PiT Copy)

Snapshot Data

▶ Atelic Data

Snapshot Equivalence

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Temporally weak; Weak equivalence

Definition

Informally, two tuples are *snapshot equivalent* or *weakly equivalent* if all pairs of timeslices with the same time instant parameter of the tuples are identical.

Let temporal relation schema *R* have *n* time dimensions, D_i, $i = 1,\ldots,n$, and let τ^i, $i = 1,\ldots,n$ be corresponding timeslice operators, e.g., the valid timeslice and transaction timeslice operators. Then, formally, tuples *x* and *y* are snapshot equivalent if

$$\forall t_1 \in D_1 \ldots \forall t_n \in D_n(\tau^n_{t_n}(\ldots(\tau^1_{t_1}(x))\ldots) \\ = \tau^n_{t_n}(\ldots(\tau^1_{t_1}(y))\ldots))$$

Similarly, two relations are snapshot equivalent or weakly equivalent if at every instant their snapshots are equal. Snapshot equivalence, or weak equivalence, is a binary relation that can be applied to tuples and to relations.

Key Points

The notion of weak equivalence captures the information content of a temporal relation in a point-based sense, where the actual timestamps used are not important as long as the same timeslices result. For example, consider the two relations with just a single attribute: {(a, [3,9]} and {(a, [3,5]), (a, [6,9])}. These relations are different, but snapshot equivalent.

Both "snapshot equivalent" and "weakly equivalent" are being used in the temporal database community. "Weak equivalence" was originally introduced by Aho et al. in 1979 to relate two algebraic expressions [1,2]. This concept has subsequently been covered in several textbooks. One must rely on the context to disambiguate this usage from the usage specific to temporal databases. The synonym "temporally weak" does not seem intuitive—in what sense are tuples or relations weak?

Cross-references

▶ Temporal Database
▶ Time Instant
▶ Timeslice Operator
▶ Transaction Time
▶ Valid Time
▶ Weak Equivalence
▶ Point-Stamped Temporal Models

Recommended Reading

1. Aho A.V., Sagiv Y., and Ullman J.D. Efficient optimization of a class of relational expressions. ACM Trans. Database Syst., 4(4):435–454, 1998.
2. Aho A.V., Sagiv Y., and Ullman J.D. Equivalences among relational expressions. SIAM J. Comput., 8(2):218–246, 1979.
3. Gadia S.K. Weak temporal relations. In Proc. 4th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1985, pp. 70–77.
4. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer-Verlag, 1998, pp. 367–405.

Snapshot Isolation

Alan Fekete
University of Sydney, Sydney, NSW, Australia

Synonyms

SI; Row-versioning

Definition

Snapshot Isolation is a multi-version concurrency control approach that is widely used in practice. A transaction *T* that operates under Snapshot Isolation never observes any effects from other transactions that overlap *T* in duration; instead *T* sees values as if it were operating on a private copy or snapshot of the database, reflecting all other transactions that had committed before *T* started. In Snapshot Isolation, the system will not allow both of two transactions to commit if they overlap in duration and modify the same data item. Snapshot Isolation prevents many well-known anomalies (such as Lost Updates and Inconsistent Reads) that are also prevented by Serializability, but it does not guarantee that all executions will be Serializable. Snapshot Isolation allows reads to occur without delay or blocking caused by concurrent updates, and also updates are never blocked by concurrent readers, so Snapshot Isolation often gives the transactions better throughput than traditional concurrency control based on Two-Phase Locking.

Historical Background

The idea of providing a multi-version concurrency control algorithm based on reading from a private snapshot has been wide-spread since at least 1982, Chan et al. [4] provided a detailed discussion of how this could work, considering many practical issues. However, in this early period, the idea was only considered for read-only transactions. That is, any transaction that included writes was treated with a different concurrency control, in order to ensure serializability.

In 1995, Berenson et al. [2] introduced the term Snapshot Isolation, and explained the algorithm for transactions that include write operations. They did so in the context of a paper about ambiguities in how the SQL Standard defines Isolation. As well as describing the algorithm, this paper showed that it could allow non-serializable executions, though it did not allow any of the particular erroneous phenomena that had been considered in the SQL Standard. Also in 1995, Oracle 7 introduced the use of the algorithm when transactions request "Isolation Level Serializable" [9]; previous releases of Oracle had not provided any transaction-level consistency, but rather they had each SQL statement within a transaction see a different state of the database. In 1999, Oracle obtained a US patent (number 5870758) on their approach.

Considerable research has been devoted to understanding the properties of executions allowed by Snapshot Isolation. Bernstein et al. [3] showed how to reason about whether transactions preserve individual integrity constraints when run with Snapshot Isolation. In 2005, Fekete et al. [8] published a theory to show which sets of transactions are guaranteed to run serializably on a DBMS platform that uses Snapshot Isolation (and therefore that every possible integrity constraint is preserved); some of these checks were automated by Jorwekar et al. [10]. Several papers [3,7] look at issues that arise when some transactions run with Snapshot Isolation, while other transactions use other concurrency control techniques. An abstract characterization of the isolation provided by Snapshot Isolation was suggested by Adya [1].

Since 2000, the issues of storing replicated data in sites each using Snapshot Isolation has been studied extensively. Different approaches are explored by many researchers [5,6,11,12,13,16].

Foundations

Like other concurrency control techniques, Snapshot Isolation (hereafter abbreviated SI) responds to requests from clients for reading and writing data items that are stored by the DBMS engine. Each request could be either performed immediately or delayed for a while, or the requesting transaction might be aborted. SI is a multi-version mechanism, so when performing a request to read an item, the engine might return a value other than the current value of the item (that is, it might return a value that had been written in some earlier transaction, not necessarily the value written by the transaction that most recently altered the item).

The SI mechanism is defined by two properties. The first property (that explains the term "Snapshot") determines which value is to be returned in a read operation. If transaction T reads a data item x, then the value returned by the system is whichever value was written by the most recently committed transaction, among all the transactions that wrote the item x and also committed before T started. There is one exception to this rule: if T itself writes the item x, and later requests to read x, then T will see the value it wrote itself. The second property of SI is sometimes called "First Committer Wins"; it requires the engine to prevent the situation where there are two concurrent (Transactions T and U are concurrent if their duration overlaps, that is there is some intersection between the interval from the start of T to its completion, and the interval from the start of U to its completion.) transactions that both write to the same item, and that both commit.

To illustrate the way SI controls concurrency, consider the sequence of operations shown in Schedule 1. In presenting this and later schedules, this entry uses the standard notation where each operation has a subscript that indicates which transaction performs the operation, so r_i is a read within T_i. As well as subscripts on the operations, each data item has versions that are indicated by subscripts, where the version of item x produced by transaction T_i is represented as x_i. Thus when T_5 writes x, this is indicated by $w_5[x_5]$, and if T_3 later does a read of x that returns the value that was written by T_5, one includes $r_3[x_5]$ in the schedule. (Readers might mistakenly think that the subscript indicates the version order; thus they think that x_2 represents the second version of x. This is not the case in the notation used here.) The usual notation is extended by representing the start of transaction T_i as b_i (while the completion of the transaction is c_i in the case of a commit, or a_i when the transaction aborts)

$$b_1 r_1[x_0] r_1[y_0] c_1 b_2 w_2[x_2] a_2 b_3 r_3[x_1] r_3[y_1] w_3[x_3] \\ b_4 r_4[x_1] r_4[y_1] w_3[y_3] r_3[x_3] c_3 c_4 \quad (1)$$

In Schedule 1, notice that the snapshot used by T_4 includes the changes made by T_1 (which committed before T_3 started), but not those made by T_2 which aborted, nor those made by T_3 which is concurrent with T_4. In particular, when T_4 reads x, it sees the version x_1 which was written by T_1, even though there is already a more recent version x_3. In this schedule one also can find an example where T_3 reads x twice, having modified the item in between; the second time T_3 reads x, it sees its own version.

SI is an attractive concurrency control mechanism for many reasons. It usually performs well, and in particular it does not suffer from the delays that can reduce throughput in locking-based concurrency control. For example, if a large slow transaction T is reading many items, in order to calculate some complicated statistics, traditional two-phase locking takes read locks on many items, and holds these while T is running; through this long period, other transactions which want to change those items will be blocked if locking is used for concurrency control. Under SI, in contrast, the large slow transaction T does not take read locks, and concurrent transactions can update the items.

The database literature has identified a number of anomalies that can occur from uncontrolled concurrency. For example, the Inconsistent Read phenomenon happens when a transaction T sees some but not all of the changes made by another transaction U. Under SI, this can't happen: if U committed before T starts, then the effects of U are all in the snapshot used when T reads, while if U is still running when T starts, then none of the changes made by U are in the snapshot. For example, in Schedule 1 the snapshot used when T_4 reads includes both changes made by T_1 (to x and also to y) and none of the changes made by T_3. Another famous phenomenon is Lost Update. An example of this occurs when two transactions both read an item, and both produce new values that increment what they read; if both these transactions commit, the final value of the item will be incremented by one instead of by two (as would happen in a serial, non-interleaved execution). If the database platform is using SI, the First Committer Wins property will prevent Lost Update (as one of the transactions will be required to abort). For example, in Schedule 2, T_2 is not allowed to commit (since T_1 and T_2 are concurrent and both have written the same item x, the property says that they cannot both commit).

$$b_1 r_1[x_0] w_1[x_1] b_2 c_1 r_2[x_0] w_2[x_2] a_2 \quad (2)$$

Despite preventing the well-known concurrency control anomalies, SI does not ensure that all executions are serializable. Schedule 3 shows an anomaly called "Write Skew."

$$b_1 r_1[x_0] r_1[y_0] b_2 r_2[x_0] r_2[y_0] w_1[x_1] w_2[y_2] c_1 c_2 \quad (3)$$

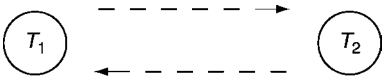
In Schedule 3, the First Committer Wins property is not effective, because the concurrent transactions do not have any item that both of them write (T_1 writes x and T_2 writes y). The lack of serializability in this schedule can result in data corruption. For example, suppose x and y are data items that represent the balance in two different bank accounts, and suppose a business rule requires the sum of the balances to be positive. Suppose initial values are $x_0 = 100$ and $y_0 = 200$, and T_1 is reducing x by 150 (aborting if it sees insufficient funds in the combined balance), while T_2 reduces y by 175 (again, aborting if there is not enough in the total of the balances). One sees that each transaction, run alone, preserves the business rule; however the Schedule 3 is possible with SI and yet it produces a final state where x is -50, and y is 25, violating the integrity of the data according to the business rule.

Because many developers think that correct isolation ought to be what SI does (namely, a transaction does not see any effects of concurrent transactions), it is worth explaining why this is not so. Correct isolation ("serializable" execution) means that the outcome is just like in a serial or batch execution. In a batch execution, between any pair of transactions, one will come first, and so the other will see its effects. Thus if neither of two transactions sees the other, this is not like a batch execution. The Write Skew example shows that two transactions may each decide to take some action like removing money from a bank account, where it is acceptable for one to make the change, but not when the other has already done so. When neither sees the other, they might both make the change and commit.

The Schedule 3 is not serializable, and this can be proved because it has a multi-version serialization graph with a cycle. In the literature, there are several variant definitions of multi-version serialization graph (MVSG) for a schedule; this entry uses the one in the text by Weikum and Vossen [15]. MVSG is defined for a given schedule and a version order $\ll$ which relates every pair of versions of the same data item. When working with SI, the version order is always taken as the order of the commits of the transactions that wrote the versions; that is one defines $x_i \ll x_j$ when T_i commits before T_j. MVSG has nodes for the transactions, and there is an edge from T_i to T_j in the following three circumstances: (i) the schedule contains $r_j[x_i]$ for some item x, (ii) $x_i \ll x_j$ and the schedule contains $r_k[x_j]$ for some x and k, or (iii) the schedule contains $r_i[x_k]$ and $x_k \ll x_j$, for some x and k. It turns out that for understanding the behavior of SI, it is important to pay attention to particular edges in the MVSG: those which go between concurrent transactions. Call such an edge vulnerable, and draw it with a dashed line in the multi-version serialization graph. Notice that the First Committer Wins rule means that there can never be a vulnerable edge between two transactions if there is some data item to which both transactions write. The Snapshot property means that if there is an edge from T_i to T_j because of an operation $r_j[x_i]$, then T_i must have committed before T_j started and so the edge is not vulnerable. Thus the only vulnerable edges arise from conflicts where one transaction reads an item which the concurrent transaction writes. The MVSG for the Schedule 3 above is in Fig. 1.

Even though SI can allow executions that are not serializable, these executions are not observed often. There are some sets of application programs which never give rise to a non-serializable execution when running with SI as the concurrency control mechanism. For example many of the standard benchmark suites, such as TPC-C [14], generate only serializable schedules. It can be proved [8] that in any schedule allowed by SI, if there is a cycle in the MVSG, then the cycle contains two *consecutive* vulnerable edges. Given a set of transactions T_1, T_2 etc, one can draw a static dependency graph SDG, which is a directed graph whose nodes are transactions, with an edge from T_i to T_j if it is possible to find a schedule h with some of these transactions, so that MVSG(h) has an edge from T_i to T_j. Furthermore, one says that the edge in SDG is *vulnerable* if there is a schedule h where the edge in MVSG(h) is vulnerable. Note that MVSG(h) depends on the schedule h which shows how the transactions interleave, but SDG can be found from the set of separate transactions. Because for any schedule h, MVSG(h) is a subset of SDG, it follows that if SDG has no cycle with consecutive vulnerable edges, then MVSG(h) also has no cycle with consecutive vulnerable edges, and so h is serializable. Thus, a set of transactions will always interleave in serializable executions under SI, provided that SDG does not have any cycle with consecutive vulnerable edges.

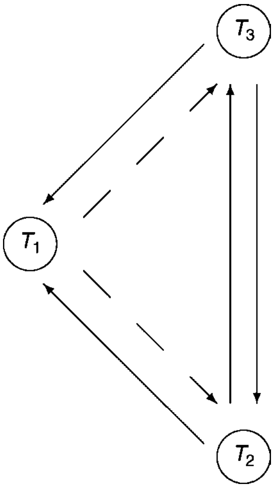
As an example, consider the transactions In Fig. 2. For these transactions, the SDG is shown in Fig. 3. The only vulnerable edges in this SDG are from T_1 to T_2 and from T_1 to T_3. The edge from T_2 to T_1 is not vulnerable because T_1 has no write operations (and under SI, a vulnerable edge can only come from a read-to-write conflict), and similarly T_3 to T_1 is not vulnerable. The edges between T_2 and T_3 are not vulnerable (in either direction) because both transactions write the item x, and so the First Committer Wins property of SI prevents these transactions both committing if they are concurrent. Thus there are no consecutive vulnerable edges at all in SDG, and so every execution of T_1, T_2 and T_3 will be serializable when they run on a platform using SI for concurrency control. To use these ideas in practice, one needs to deal with application code that contains parameterized SQL statements, and complicated control flow; the techniques needed are discussed in [8,10].



Snapshot Isolation. Figure 1. MVSG for Schedule 3.

$$\begin{aligned} T_1 &= r[x]r[y]r[z] \\ T_2 &= r[x]r[y]w[x]w[y] \\ T_3 &= r[x]r[z]w[x]w[z] \end{aligned}$$

Snapshot Isolation. Figure 2. Transactions with every execution serializable.



Snapshot Isolation. Figure 3. SDG for Transactions from Fig. 2.

What can the database administrator do if they have an application which will run on a platform where SI is the concurrency control mechanism, and yet the application is made up of transactions that are not guaranteed to have serializable executions on such a platform? The natural approach is to alter the application code, without changing the meaning of each transaction, so that the changed transactions are certain to execute serializably. This means changing programs so as to make some edges from the SDG be not vulnerable. Two techniques are known to change transactions T_i and T_j where the edge from T_i to T_j is vulnerable. One can materialize the conflict, by creating a new table called say Conflict, and including in both T_i and T_j an update of a particular row in this table. Alternatively, one can sometimes leave T_j unaltered, and introduce an identity write into T_i. That is, perform "SET x=x" in an UPDATE statement which is added to the code of T_i, to affect whichever data item (row) is the one which T_i reads and T_j writes.

In conclusion, SI is a concurrency control mechanism that has many attractive features. It usually gives quite good throughput, since a read operation is never delayed by other transactions that are changing the data, and updates are not delayed when other transactions have read the data they want to change. The outdated versions that are used in SI, to respond to read requests, are often available anyway, because they are kept to support rollback recovery. SI prevents many bad executions; it can't suffer from Lost Update or Inconsistent Read or Phantoms. The way SI works is easy to understand, and indeed many articles have just assumed that being "isolated" means "not seeing any changes made by concurrent transactions" (as happens in SI). However, SI does not enforce that every execution will be serializable. Developers and users need to be aware that when SI is used, it is possible that transactions can interleave in ways that make the data invalid according to some business rule which is obeyed by every transaction running alone.

Key Applications

Snapshot Isolation is used as a concurrency control mechanism in a wide range of common platforms. For example, Microsoft SQL Server 2005 offers it when a user chooses to invoke "SET TRANSACTION ISOLATION LEVEL SNAPSHOT." It is similarly available as a separate isolation level in Interbase and Oracle Berkeley DB. Other platforms, such as PostgreSQL (since version 7) and Oracle, use SI when the client chooses "SET ISOLATION LEVEL SERIALIZABLE" even though SI does allow non-serializable executions. SI is very useful in managing replicated data. One can combine individual databases which use SI, to act transparently as a global one-copy database. This is easier than to combine traditional locking databases to provide one-copy serializability. Many research prototypes combine SI with consistent replication, however these ideas are not widely used in practice yet.

Future Directions

The main focus of current research with SI is in replicated data management. There are many issues that arise when data are replicated between sites some or all of which use SI rather than traditional locking for local concurrency control. Many different systems have been designed and evaluated but no clear winner has yet emerged, so research is continuing. Another topic that needs more understanding is the performance of SI, in particular to understand which characteristics of an application domain make SI perform better or worse than other concurrency control techniques.

Cross-references

- ▶ Concurrency Control – Traditional Approaches
- ▶ Consistency Models for Replicated Data
- ▶ Multi-Version Serializability and Concurrency Control
- ▶ Replication for Scalability

▶ Serializability
▶ SQL Isolation Levels

Recommended Reading

1. Adya A. Weak consistency: a generalized theory and optimistic implementations for distributed transactions (PhD thesis). Technical Report MIT/LCS/TR-786, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA, 1999.
2. Berenson H., Bernstein P.A., Gray J., Melton J., O'Neil E.J., and O'Neil P.E. A critique of ANSI SQL isolation levels. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 1–10.
3. Bernstein A.J., Lewis P.M., and Lu S. Semantic conditions for correctness at different isolation levels. In Proc. 16th Int. Conf. on Data Engineering, 2000, pp. 57–66.
4. Chan A., Fox S., Lin W.-T.K., Nori A., and Ries D.R. The implementation of an integrated concurrency control and recovery scheme. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1982, pp. 184–191.
5. Daudjee K. and Salem K. Lazy database replication with snapshot isolation. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 715–726.
6. Elnikety S., Zwaenepoel W., and Pedone F. Database replication using generalized snapshot isolation. In Proc. 24th IEEE Symp. on Reliable Dist. Syst., 2005, pp. 73–84.
7. Fekete A. Allocating isolation levels to transactions. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 206–215.
8. Fekete A., Liarokapis D., O'Neil E., O'Neil P., and Shasha D. Making snapshot isolation serializable. ACM Trans. Database Syst., 30(2):492–528, 2005.
9. Jacobs K. Concurrency control: transaction isolation and serializability in SQL92 and Oracle7. Technical Report A33745 (White Paper), Oracle Corporation, 1995.
10. Jorwekar S., Fekete A., Ramamritham K., and Sudarshan S. Automating the detection of snapshot isolation anomalies. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 1263–1274.
11. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Middleware based data replication providing snapshot isolation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 419–430.
12. Plattner C. and Alonso G. Ganymed: scalable replication for transactional web applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004, pp. 155–174.
13. Schenkel R. and Weikum G. Integrating snapshot isolation into ransactional federation. In Proc. Int. Conf. on Cooperative Inf. Syst., 2000, pp. 90–101.
14. Transaction Processing Performance Council. TPC Benchmark C Standard Specification, Revision 5.0. 2001. URL: http://www.tpc.org/tpcc/
15. Weikum G. and Vossen G. Transactional information systems: Theory, algorithms, and the practice of concurrency control and recovery. Morgan Kaufmann, Los Altos, CA, USA, 2002.
16. Wu S. and Kemme B. Postgres-R(SI): combining replica control with concurrency control based on snapshot isolation. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 422–433.

SNIA

▶ Storage Network Industry Association

Snippet

Marcus Herzog[1,2]
[1]Vienna University of Technology, Vienna, Austria
[2]Lixto Software GmbH, Vienna, Austria

Synonyms

Web widget; Macro; Module; Capsule; Mini; Flake

Definition

A snippet is a chunk of re-usable source code. In the context of Web programming, a snippet refers to a chunk of re-usable HTML source code, along with all relevant resources such as stylesheets and scripts applied within the context of the snippet. In the context of Web information extraction, a snippet is a subset of the available information items that can be extracted from the Web page.

Key Points

The term snippet originates from the domain of text editors, where snippets refer to chunks of source code which can be organized for copy and paste usage. Snippet management allows for viewing, editing, sorting, and storing snippets in a repository of re-usable source code fragments. The overall goal of snippets is to ease the process of writing code by reducing the manual effort to type in source code and to re-use existing lines of code.

Snippets can be classified according to the complexity of the interaction process: static, dynamic, and scriptable snippets. A static snippet is a fixed chunk of text that can be inserted at the cursor position. This operation is similar to a cut-and-paste operation well known from text editors. Dynamic snippets contain some dynamic elements which are filled in on insertion of the snippet into the main document. Scriptable snippets take this dynamic concept one step further by not only allowing for filling in placeholders, but by providing means to compute the values of placeholders, e.g., by applying a transformation operation on a placeholder value.

In Web programming, snippets are often used when assembling a web page from pre-existing building blocks. This is very popular in constructing social network home pages or other types of personal Web 2.0 applications such as Blogs. In this context snippets are often referred to as e.g., Web widgets, minis, or flakes, depending on the framework in which the snippet is programmed. In Web programming a snippet is already more like a mini application which can be reused in the context of a Web application, e.g., a portal such as iGoogle or MyYahoo.

In Web data extraction [1] the concept of snippet is used to refer to a particular part of the Web page which is extracted and transformed into an information item. Here the emphasis is on the re-use of existing data or content which is transformed into a presentation-independent representation, e.g., XML document format. The goal is to re-use exiting data in the context of new applications which assemble data snippets from various sources and provide additional value by relating the content extracted from these independent sources.

Cross-references

▶ Blogging
▶ Re-Usable Code
▶ Re-Usable Information Item
▶ Web Programming

Recommended Reading

1. Baumgartner R., Flesca S., and Gottlob G. Visual web information extraction with lixto. In Proc. 27th int. Conf. on Very Large Data Bases, 2001, pp. 119–128.

Snowflake Join Schema

▶ Snowflake Schema

Snowflake Schema

Konstantinos Morfonios, Yannis Ioannidis
University of Athens, Athens, Greece

Synonyms

Snowflake join schema

Definition

A *snowflake schema* has one "central" table whose primary key is compound, i.e., consisting of multiple attributes. Each one of these attributes is a foreign key to one of the remaining tables, which may, in turn, have some of its non-key attributes each be a foreign key to yet another, different table. This continues recursively with the remaining tables, until they are exhausted, forming chains or trees of foreign key dependencies rooted at the "central" table, i.e., each table in the schema (except the "central" table) is pointed to by exactly one such foreign key. (In the above, without loss of generality, we make the assumption that all tables except the "central" table have simple primary keys. This is usually the case in almost all practical situations, as for efficiency, these keys are often generated, *surrogate keys*.)

Key Points

Many data warehouses (see definitional entry for *Data Warehouse*) that represent the multidimensional conceptual data model in a relational fashion [1,2] store their primary data as well as the data cubes derived from it in snowflake schemas, as an alternative to *star schemas*. As in star schemas, the "central" table and the remaining tables of the definition above correspond, respectively, to the *fact table* and the *dimension tables* that are typically found in data warehouses. Each *fact* (tuple) in the fact table consists of a set of numeric *measures*, comprising the objects of analysis, and a set of *dimensions*, which uniquely determine the set of measures. The remaining tables store the attributes of the aforementioned dimensions at different levels of granularity.

Unlike star schemas, snowflake schemas can explicitly capture hierarchies in the dimensions, with each table in each chain (or tree path) of foreign key dependencies corresponding to one level of one such hierarchy. For instance, dimension `Store` in the example below contains values at different levels of detail, forming the hierarchy `Street`→`City`→`State`. On the contrary, star schemas capture all levels of a hierarchical dimension in a single, de-normalized table. Starting from a star schema (usually in Second Normal Form), one may generate the corresponding snowflake schema (usually in Third Normal Form at least) by normalization, decomposing the dimensions into multiple tables. Accordingly, star schemas lend themselves to simpler and usually faster queries, while snowflake schemas are easier to maintain and require less space.

For example, consider a data warehouse of a retail chain with many stores around a country. The dimensions may be the products sold, the stores themselves with their locations, and the dates, while the numeric measures may be the number of items and the total monetary amount corresponding to a particular product sold in a particular store on a particular date. The relevant snowflake schema, with the product, store, and date dimensions normalized, is shown below, where `SalesSummary` is the fact table, primary keys are in italics, and each attribute of the fact-table primary key as well as each non-key 'Id' attribute of the other tables is a foreign key.

`SalesSummary(`*ProductId*, *StoreId*, *DateId*, `NumOfItems, TotalAmount)`

`Product(`*ProductId*, `ProdName, ProdDescr, CategoryId, UnitPrice)`

`Category(`*CategoryId*, `CategoryDescr)`

`Store(`*StoreId*, `StreetId)`

`Street(`*StreetId*, `Street, CityId)`

`City(`*CityId*, `City, StateId)`

`State(`*StateId*, `State)`

`Date(`*DateId*, `Date, MonthId)`

`Month(`*MonthId*, `Month, YearId)`

`Year(`*YearId*, `Year)`

Cross-references

- ▶ Cube Implementations
- ▶ Data Warehouse
- ▶ Dimension
- ▶ Hierarchy
- ▶ Measure
- ▶ Multidimensional Modeling
- ▶ Star Schema

Recommended Reading

1. Chaudhuri S. and Dayal U. An overview of data warehousing and OLAP technology. ACM SIGMOD Rec., 26(1):65–74, 1997.
2. Kimball R. and Ross M. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling. Wiley, New York, NY, USA, 2nd edn., 2002.

SOA

▶ Service Oriented Architecture

SOA Replication

▶ Replication in Multi-Tier Architectures

SOAP

Eric Wohlstadter
University of British Columbia, Vancouver,
BC, Canada

Definition

SOAP [1] is an application-level protocol standard used to transport messages in distributed systems. The standard was defined and is maintained by the XML Protocol Working Group of the World Wide Web Consortium. SOAP is commonly used in the context of Web services. SOAP messages are encoded using XML and intended to carry XML encoded application data.

Key Points

SOAP provides a standard to separate infrastructure related data from application data for XML based messages. SOAP messages are known as "envelopes," which contain both a header, for infrastructure data, and a body for application data. The infrastructure which handles messages for applications is referred to as a "SOAP node." This role is commonly filled by some middleware platform. The SOAP protocol dictates the rules for the proper processing of messages by nodes on behalf of applications; this includes processing of header information and handling of faults.

The header processing rules are designed to make it easy to interpose network intermediaries between the sender and receiver of messages. The SOAP specification mentions that these intermediaries could be used for purposes such as "security services, annotation services, and content manipulation services." The specification of header information used by specific kinds of intermediaries is left to other specifications commonly known as the WS-* proposals.

The SOAP specification is intended to be extensible so that different rules for message processing can be described in further specifications. These rules are called "message exchange patterns." SOAP provides details of patterns for simple synchronous and asynchronous message exchange, which can be used for the purpose of remote procedure calls. The specification mentions but does not provide details for other more stateful patterns such as conversational exchanges and peer-to-peer message routing.

When a SOAP node is unable to process a message, an error message, called a SOAP fault is issued. Several descriptive fault types are provided by the specification as well as the conditions under which each type should be used.

SOAP provides the foundation of a Web services stack. SOAP messages are commonly layered on top of the Hypertext Transfer Protocol (HTTP). This tends to make SOAP services easier to deploy behind network firewalls; although, some critics have argued this is an abuse of HTTP. Since XML messages tend to be much larger than their binary counterparts, SOAP provides guidelines for using a binary encoding of a SOAP message body.

SOAP was originally intended as the "Simple Object Access Protocol," and its designers intended it to be used with traditional distributed object technologies such as remote method invocation. When SOAP became popular for Web services the acronym was dropped because Web service interfaces are agnostic as to whether object-oriented implementations are used.

Cross-references

- ► RMI
- ► Web Services
- ► W3C

Recommended Reading

1. SOAP Version 1.2, Part 1: Messaging Framework (2nd edn.). W3C Recommendation. http://www.w3.org/TR/soap12-part1/

Social Applications

Maristella Matera
Politecnico di Milano University, Milan, Italy

Synonyms

Web 2.0 applications; Collaborative software

Definition

Web applications, characteristic of the Web 2.0, that allow users to share data and interact with other users.

Key Points

The advent of the Web 2.0 has empowered the Web clients, thus providing users with richer and more complex interaction capabilities. The development of communication and interaction tools has therefore emerged, giving raise to computer-mediated communication and to collaborative approaches to content creation, based on new applications, such as social networking, file sharing, instant messaging, and blogs – just to mention few.

The advantage from the user experience perspective is that Web users do not play only the role of passive actors accessing information, but they become creators of the contents published by Web applications.

Very often, such new applications also foster the creation of *online communities*, i.e., groups of people that interact via Web-based communication media and cooperatively create contents.

Cross-references

- ► Visual Interaction
- ► Web 2.0/3.0

Social Networks

Felix Schwagereit, Steffen Staab
University of Koblenz-Landau, Koblenz, Germany

Definition

A social network is a social structure made of *actor*s, which are discrete individual, corporate or collective social units like persons or departments [19] that are tied by one or more specific types of *relation* or interdependency, such as friendship, membership in the same organization, sending of messages, disease transmission, web links, airline routes, or trade relations. The actors of a social network can have other attributes, but the focus of the social network view is on the properties of the relational systems themselves [19]. For many applications social networks are treated as graphs, with actors as nodes and ties as edges. A *group* is the finite set of actors the ties and properties of whom are to be observed and analyzed. In order to define a group it is necessary to specify the network boundaries and the sampling. *Subgroup*s consist of any subset of actors and the (possible) ties between them.

The science of social networks utilizes methods from general network theory and studies real world networks as well as structurally similar subjects dealing e.g., with information networks or biological networks.

Historical Background

The science of social network analysis comprises methods from social sciences, formal mathematical, statistical and computing methodology [19]. The first developments of scientific methods were empirically motivated and date back to the late nineteenth century. Jacob Moleno developed methods to facilitate the understanding of friendship patterns within small groups in the 1920s and 1930s. Other pioneers in the field of social networks were Davis, who studied social circles of women in an unnamed American city and Elton Mayo, who studied social networks of factory workers. Many of the current formal concepts (e. g., density, span, connectedness) had been introduced in the 1950s and 1960s as ways to describe social structures through measures. Another important milestone was an experiment Stanley Milgram conducted in 1967. In Milgram's experiment, a sample of US individuals were asked to reach a particular target person by passing a message along a chain of acquaintances. The average length of successful chains turned out to be about five intermediaries or six steps of separation.

Early research on social networks was limited to small networks with up to a few hundred actors, which could be examined visually. With increased computational power for data acquisition and management, networks may now comprise several millions of actors.

Foundations

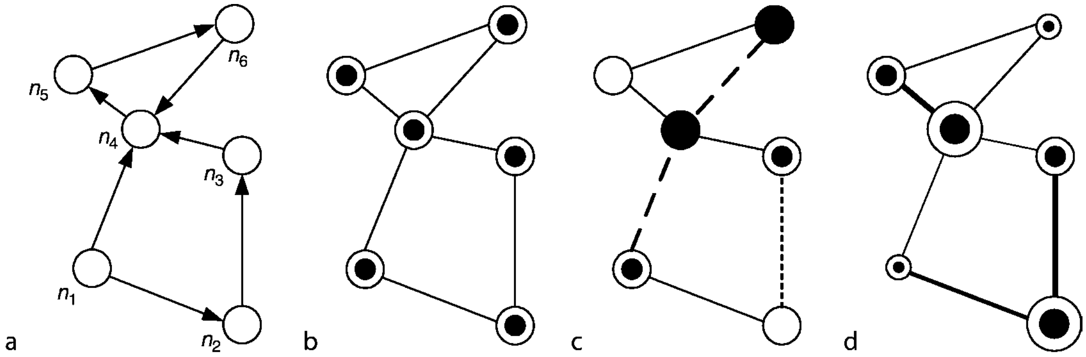
Types

The simplest type of network consists of only one set of actors and one relation representing one type of ties between the actors. More complex networks can be composed of different types of actors (*multi-mode*) and different relations (*multi-relational*). Furthermore the actors and ties between them can have assigned properties, which are mostly numerical. Ties can have a direction, which makes the network a directed graph. Figure 1 shows a selection of network types [12]. Network (i) is a directed network in which each edge has a direction; (ii) is an undirected network with only one type of actors; (iii) is a network with several types of actors and relations; (d) shows a network with different weights for actors and ties.

Of special interest in science of social networks are *bipartite graphs* [12] which contain actors of two types and ties connecting only actors of different types. They are called *affiliation networks* because they are suitable to express the membership of people (one type of actors) in groups (the second type of actors).

Notation

The common notation for social networks is the *sociometric notation* [19]. Simple social networks with one relation and only one group of actors (like the one shown above) are represented as a matrix, called *sociomatrix* or *adjacency matrix*. For one relation X, let $\mathbf{X}$ be the corresponding matrix. This matrix has g rows and g columns. The value at position x_{ij} denotes whether there exists a tie from the ith element of the social network to the jth element. An example sociomatrix for the social network (a) in Fig. 1 is shown in Table 1.



Social Networks. Figure 1. Types of social networks.

Social Networks. Table 1. Sociomatrix for network (a) in Fig. 1

	n_1	n_2	n_3	n_4	n_5	n_6
n_1	-	1	0	1	0	0
n_2	0	-	1	0	0	0
n_3	0	0	-	1	0	0
n_4	0	0	0	-	1	0
n_5	0	0	0	0	-	1
n_6	0	0	0	1	0	-

For more complex networks, like multi-mode and/or multi-relational social networks, tensors may be used instead of matrices [18].

Measures

Measures have been developed in order to formalize local and global properties for social networks. Local and global properties of social networks describe ego-centric properties of individual actors and socio-centric properties of the network as a whole, respectively. Furthermore, subsets of actors (subgroups) can be determined. The following paragraphs contain an outline of several basic concepts.

Socio-centric Properties In order to compare different social networks in size and structure the following basic measures have been established.

- *Number of Actors: g*
- *Number of Ties: m*
- *Mean Standarized Degree (Density)*: $z = \frac{\sum C_D(n_i)}{g(g-1)}$
- *Mean Actor-Actor Distance / Characteristic Path Length*: $l = \frac{1}{\frac{1}{2}g(g+1)} \sum_{i \geq j} d(n_i, n_j)$
- *Diameter*: is the longest Distance between all pairs of nodes of a given network. The distance $d(n_i, n_j)$ between a pair of nodes n_i and n_j in the network is the length of the geodesic (which is the shortest path between the two nodes).

Ego-centric Properties The identification of the "most important" or "prominent" actor was one of the primary goals of social network analysis [19]. Therefore various measures were developed to quantify "importance" of actors and subgroups for a given social network. The following measures can be calculated for simple undirected graphs of social networks.

- *Actor Degree Centrality* is the count of the number of ties to other actors in the network. The relevance of this measure is based on the assumption that an actor, which has more connections than other actors can be considered more active and therefore important. The actor degree centrality is calculated from sociomatrix **X** as follows:

$$C_D(n_i) = \sum_j x_{ij}$$

- *Actor Closeness Centrality* is the degree to which an individual is close to all other individuals in a network (directly or indirectly). Therefore an actor is central if it can quickly (that means by relying on so few mediatorsas possible) interact with all other actors. The index of actor closeness-centrality is:

$$C_C(n_i) = \left[\sum_{j=1, j \neq i}^{g} d(n_i, n_j) \right]^{-1}$$

where d(n_i, n_j) is the length of the geodesic of actor i and actor j. To allow comparisons between different networks actor closeness can be standardized:

$$C'_C(n_i) = \frac{g-1}{\left[\sum_{j=1,\ j \neq i}^{g} d(n_i, n_j) \right]}$$

- *Actor Betweenness Centrality* is the degree to which an individual lies between other individuals in the network. Therefore it is based on the assumption that all other actors lying in between have a certain amount of control on the interaction relying on them. So the betweenness of an actor is higher if more of the possible interactions rely on it as mediator. In order to calculate betweeness centrality two other measures are needed: g_{jk}, the number of geodesics linking two actors j and k; as well as $g_{jk}(n_i)$, which is the number of geodesics linking two actors that contain the actor i:

$$C_B(n_i) = \sum_{j<k} g_{jk}(n_i) / g_{ik}$$

For comparisons the measure can be normalized:

$$C'_B(n_i) = C_B(n_i) / [(g-1)(g-2)/2]$$

Subgroups In most social networks actors organize themselves in subgroups or cliques, which have their own values, sub-cultures, and structures. Therefore several methods to define and recognize certain kinds of subgroups were developed [19].

- A *Clique of size k* is a subgroup consisting of *k* many actors which are all adjacent to each other.
- An *n-Clique* is a subgroup with the property that the distance (length of the geodesic) between all actors is no greater than n and there is no actor with a distance equal or less than n outside the n-clique. An n-clique with $n = 1$ is equal to a normal clique.
- A *k-Core* is a subgroup with each actor is adjacent to at least n other actors in the subgroup.
- A *Cluster* is a subgroup consisting of actors which are similar to each other. The similarity (structural equivalence) of two actors can be defined with criteria like euclidean distance or correlation based on vectors of a sociomatrix. Similarity based clusters in undirected networks are usually created by using agglomerative or divisive hierarchical clustering methods [14]. For clustering directed networks methods like directed spectral clustering [7] can be used. In general graph theory there exist methods for partitioning graphs which can also be applied to graphs of social networks. One of these methods is the min-max cut algorithm which pursues the goal of minimizing the similarity between subgraphs while maximizing the similarity within each subgraph. Other clustering approaches are based on methods for finding densely connected subgroups by the calculation of special clustering coefficients or by comparing the number of connections within a subgroup with the number of connections to outside actors.

Topological Properties

Small-World Topology The small-world model [20] is a well studied distribution model of actors and ties, since it has interesting properties and features. Due to the fact that networks often have a geographical component to them it is reasonable to assume that geographical proximity will play a role in deciding which actors are connected. So in a small-world network each actor is connected to actors in its near neighborhood. Other connections between more distant actors (long-range connections) are infrequent and have a low probability. The probability for each actor of having a degree k follows a power law $p_k \sim k^{-\alpha}$ with α as constant scaling exponent. Despite the fact that long-range connections occur only sporadicly the diameter of small-world networks is exponentially smaller than their size, being bounded by a polynomial in $\log g$, where g is the number of nodes. In other words, there is always a very short path between any two nodes [8].

The discovery that real world social networks might have small-world characteristics explains the importance of this model. So it can be observed that the chain of social acquaintances required to connect one arbitrary person to another arbitrary person anywhere in the world is generally short. This concept gave rise to the famous phrase six degrees of separation after a 1967 small-world experiment by Stanley Milgram. Academic researchers continue to explore this phenomenon. A recent electronic small-world experiment [5] at Columbia University showed that about five to seven degrees of separation are sufficient for connecting any two people through e-mail. Other applications of the small-world model are investigations of iterated games, diffusion processes or epidemic processes [12].

Creation of Networks

Artificially generated graphs allow comparison with real datasets and by analyzing and comparing their properties they give insights into the inner structure of social networks. They also allow for the generation of (overlay) network structures on top of existing information structures.

Several procedures are known to generate social networks from scratch. A *Poisson random graph* is the simplest way to construct a social network. This is simply done by connecting each pair of actors with the probability of p. The result of this procedure is a network with a Poisson degree distribution ($p_k = \frac{\lambda^k}{k!} e^{-\lambda}$). Since this distribution is unlike the highly skewed power-law distributions of real world networks other methods have been proposed [12].

One of the important methods is known as *preferential attachment* [1]. In this model, new nodes are added to a pre-existing network, and connected to each of the original nodes with a probability proportional to the number of connections each of the original nodes already had. I.e., new nodes are more likely to attach to hubs than peripheral nodes or in other words the “rich-get-richer”. Statistically, this method will generate a power-law distributed small-world network (that is, a scale-free network).

Since there is evidence that the preferential attachment model does not show all the properties real world networks obey, like increasing of the average degree and shrinking of the diameter on growing of a network, other models have been proposed [9]. The *Community Guided Attachment*, which is based on a decomposition of actors into a nested set of subgroups, such that the difficulty of forming new links between subgroups increases with the size of the subgroups. In the *Forest Fire Model* new actors are attached to the network by burning through existing ties in epidemic fashion.

Key Applications

Distributed Information Management

Social routing allows to route efficiently in peer-to-peer networks without knowledge about the global network structure. This routing with local knowledge can be achieved by regarding the network as a social network and exploiting several properties of social networks like small-world characteristics [8,10].

Information Replication in information networks can improve scalability and reliability. By performing social network clustering on these structures prefetching of content can be improved [15].

Information Extraction

Name disambiguation is a technique for distinguishing person names in unsupervised information frameworks (e.g., web pages), where unique identifiers can not be assumed [2].

Ontology Extraction methods can be performed on social network structures like communities and their folksonomies. This approach is based on the assumption that individual interactions of a large number of actors might lead to global effects that could be observed as semantics [11].

Social Recommendations

*Social networking portal*s like Xing or LinkedIn allow users to express their relationships to other users and to provide personal information. This social network can be used e.g., for finding a short path to persons in special positions by identifying the geodesic to them [16].

Filtering, recommendations and inferred trust can be improved by taking into account the social networks all relevant actors are involved. So e.g., the trustworthiness of Bob can be inferred from a social network by Alice even if both are not directly known to each other [6].

Viral marketing is the strategy to let satisfied customers distribute advertisements (e.g., video clips) by recommendation or forwarding to other potential customers they know. Viral marketing campaigns are usually started by sending the advertisements to actors holding central positions in social networks in order to facilitate a rapid distribution [13,17].

Future Directions

For the future of the social network science many areas remain insufficiently explored [12]. Many properties of social networks have been studied in the past decades. But the scientific community is still lacking the whole picture which shows what the most important properties for each application are. Especially generalized propositions (e.g., "Are more centralized organizations more efficient?") about the structure of social networks need further verification across a large number of networks [19]. Another important direction of future research is to improve the understanding of the dynamics in and the evolution of social networks [3]. In order to archive this new and more sophisticated models of social networks have to be developed. New kinds of data including more complex structures and new properties of actors or relations demand further generalization of current models. An example of these more complex structures are multiple relations which connect more than two actors.

Data Sets

- Enron Email dataset (http://www.cs.cmu.edu/~enron/ and http://www.enronemail.com/) contains about 600,000 Email messages belonging to 156 users. It was made public during the legal investigation concerning the Enron corporation.
- The Internet Movie Data Base (IMDB) (http://www.imdb.com/interfaces/) is a collection of data about movies (about 400,000) and actors (about 900,000). Especially the affiliation network of the co-appearance of actors in the same movie is subject of several studies. (cf. "The Oracle of Bacon" http://oracleofbacon.org/)
- Digital Bibliography & Library Project (DBPL) collects the bibliographic information on major computer science journals and proceedings (currently

about 950,000 articles). Similar to the IMDB the co-authorship can be used to generate affiliation networks. (dataset http://dblp.uni-trier.de/xml/)

- Southern Woman Dataset, which was collected in the 1930s is published in the classical study of Davis [4], a pioneer of social network analysis. It contains the attendance at 14 social events by 18 women in an unnamed US city.

URL to Code

Tools and Libraries

(cf. http://www.insna.org/software/index.html):

- Jung: http://jung.sourceforge.net/
- Pajek: http://vlado.fmf.uni-lj.si/pub/networks/pajek/default.htm
- UCINET: http://www.analytictech.com/ucinet/ucinet.htm

Conference Series

- International Sunbelt Social Network Conferences: http://www.insna.org/sunbelt/index.html

Journals

- Social Networks: http://www.innsa.org/pubs/connections/index.html
- CONNECTIONS: http://www.insna.org/indexConnect.html
- Journal of Social Structure: http://www.cmu.edu/joss/

Cross-references

▶ Biological Networks
▶ Cluster and Distance Measure
▶ Clustering Overview and Applications
▶ Graph
▶ Hierarchial Clustering
▶ Web Characteristics and Evolution

Recommended Reading

1. Barabási A.L. and Albert R. Emergence of scaling in random networks. Science, 286:509–512, 1999.
2. Bekkerman R. and McCallum A. Disambiguating Web appearances of people in a social network. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 463–470.
3. Berners-Lee T., Hall W., Hendler J., Shadbolt N., and Weitzner D.J. Creating a science of the Web. Science, 313:769–771, 2006.
4. Davis A., Gardner B.B., and Gardner M.R. Deep South. The University of Chicago Press, 1941.
5. Dodds P.S., Muhamad R., and Watts D. An experimental study of search in global social networks. Science, 301:827–829, 2003.
6. Golbeck J. and Hendler J.A. Inferring binary trust relationships in Web-based social networks. ACM Trans. Internet Techn., 6(4):497–529, 2006.
7. Huang J., Zhu T., and Schuurmans D. Web communities identification from random walks. In Proc. Joint European Conf. on Machine Learning and European Conference on Principles and Practice of Knowledge Discovery in Databases, 2006.
8. Kleinberg J. Navigation in a small world. Nature, 406:845, 2000.
9. Leskovec J., Kleinberg J., and Faloutsos C. Graph evolution: Densification and shrinking diameters. ACM Trans. Knowl. Discov. Data, 1(1):2, 2007.
10. Löser A., Staab S., and Tempich C. Semantic Social Overlay Networks. IEEE Journal on Selected Areas in Communication, 25(1):5–14, 2007.
11. Mika P. Social Networks and the Semantic Web. Springer, 2007.
12. Newman M.E.J. The Structure and Function of Complex networks. SIAM Rev., 45(2):167–256, 2003.
13. Richardson M. and Domingos P. Mining knowledge-sharing sites for viral marketing. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 61–70.
14. Scott J. Social Network Analysis: A Handbook. Sage, 2000.
15. Sidiropoulos A., Pallis G., Katsaros D., Stamos K., Vakali A., and Manolopoulos Y. Prefetching in content distribution networks via web communities identification and outsourcing. World Wide Web J., 11(1):39–70, 2008.
16. Staab S., Domingos P., Mika P., Golbeck J., Ding L., Finin T.W., Joshi A., Nowak A., and Vallacher R.R. Social networks applied. IEEE Intell. Syst., 20(1):80–93, 2005.
17. Subramani M.R. and Rajagopalan B. Knowledge-sharing and influence in online social networks via viral marketing. Commun. ACM, 46(12):300–307, 2003.
18. Sun J., Tao D., and Faloutsos C. Beyond streams and graphs: dynamic tensor analysis. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006, pp. 374–383.
19. Wasserman S. and Faust K. Social network analysis. Cambridge University Press, Cambridge, 1994.
20. Watts D.J. and Strogatz S.H. Collective dynamics of “small-world” networks. Nature, 393:440–442, 1998.

Software Transactional Memory

Keir Fraser
University of Cambridge, Cambridge, UK

Definition

Software transactional memory (STM) is a method of concurrency control in which shared-memory accesses are grouped into transactions which either succeed or fail to commit in their entirety. STM provides applications programmers with an alternative to mutual-exclusion locks which avoids many of the latter's pitfalls, including risk of deadlock, unnecessary serialization, and priority inversion. Many STMs are themselves implemented using lock-free programming methods, although this is not a hard-and-fast rule.

Key Points

A software transactional memory (STM) is a software library or programming-language feature which provides application programmers with an interface for allocating and accessing shared-memory variables [3]. These variables are accessible in a concurrency-safe manner without resorting to classical concurrency-management techniques such as mutual exclusion. This is achieved by grouping accesses into transactions which execute in isolation and then atomically succeed or fail in their entirety.

The application programmer chooses when transactions should start and end, rather like choosing when to acquire and release mutexes in a conventional multi-threaded program, to ensure consistency of application data structures. The STM implementation is responsible for ensuring that transactions execute in isolation and coomit atomically. Thus transactional memory guarantees the same ACID properties as classical database transactions, with the exception of durability.

The benefits of a transactional interface to shared memory are numerous. Traditional mutexes, when used conservatively, can lead to unnecessary serialization of operations that do not otherwise conflict. When a finer-grained approach is taken, involving multiple locks with individually smaller scope, the programmer must take care to avoid subtle deadlock scenarios. STM is perhaps the most promising of the proposed lock-free techniques which eschew traditional mutual exclusion and hope to enable the average programmer to implement scalable multi-threaded applications in mainstream languages [1]. Hence, although still in its infancy and an ongoing topic of research, STM is being viewed eagerly by an industry looking for salvation from the complexity of optimizing for modern multi-core systems [2].

Cross-references

- ► Performance Analysis
- ► Transaction

Recommended Reading

1. Fraser K. and Harris T. Concurrent programming without locks. ACM Trans. Comput. Syst., 25(2), 2007.
2. Saha B., Adl-Tabatabai A., Hudson R., Minh C., and Hertzberg B. McRT-STM: a high performance software transactional memory system for a multi-core runtime. In Proc. 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2006, pp. 187–197.
3. Shavit N. and Touitou D. Software transactional memory. In Proc. ACM SIGACT-SIGOPS 14th Symp. on the Principles of Dist. Comp., 1995, pp. 204–213.

SONs

► Semantic Overlay Networks

Sort-Merge Join

Jingren Zhou
Microsoft Research, Redmond, WA, USA

Synonyms

Merge join

Definition

The sort-merge join is a common join algorithm in database systems using sorting. The join predicate needs to be an equality join predicate. The algorithm *sorts* both relations on the join attribute and then *merges* the sorted relations by scanning them sequentially and looking for qualifying tuples.

Key Points

The sorting step groups all tuples with the same value in the join attribute together. Such groups are sorted based on the value in the join attribute so that it is easy to locate groups from the two relations with the same attribute value. Sorting operation can be fairly expensive. If the size of the relation is larger than the available memory, external sorting algorithm is required. However, if one input relation is already clustered (sorted) on the join attribute, sorting can be completely avoided. That is why the sort-merge join looks attractive if any of the input relations is sorted on the join attribute.

The merging step starts with scanning the relations *R* and *S* and looking for matching groups from the two relations with the same attribute value. The two scans start at the first tuple in each relation. The algorithm advances the scan of R as long as the current R tuple has an attribute value which is less than that of the current S tuple. Similarly, the algorithm advances the scan of S as long as the current S tuple has an attribute value which is less than that of the current R tuple. The algorithm alternates between such advances until an R tuple $\mathcal{R}$ and an S tuple $\mathcal{S}$ with $\mathcal{R}.r = \mathcal{S}.s$. The join tuple $\{\mathcal{R}, \mathcal{S}\}$ is added to result.

There could be several R tuples and several S tuples with the same attribute value as the current tuples $\mathcal{R}$ and $\mathcal{S}$. That is, several R tuples may belong to the current $\mathcal{R}$ group since they all have the same attribute value. The same applies to the current $\mathcal{S}$ group. Every tuple in the current $\mathcal{R}$ group joins with every tuple in the current $\mathcal{S}$ group. The algorithm them resumes scanning R and S, beginning with the first tuples that follow the group of tuples that are just processed.

When the two relations are too large to be held in available memory, one improvement is to combine the merging step of external sorting with the merging step of the join if the number of buffers available is larger than the total number of sorted runs for both R and S. The idea is to allocate one buffer page for each run of R and one for each run of S. The algorithm merges the runs of R, merges the runs of S, and joins (merges) the resulting R and S streams as they are generated.

```
Algorithm 1: Sort-Merge Join: $R \bowtie_{R.r=S.s} S$
  // sorting step
  Sort the relation $R$ on the attribute $r$;
  Sort the relation $S$ on the attribute $s$;
  // merging step
  $\mathcal{R}$ = first tuple in R;
  $\mathcal{S}$ = first tuple in S;
  $\mathcal{S}'$ = first tuple in S;
  while $\mathcal{R} \neq eof$ and $\mathcal{S}' \neq eof$ do
      while $\mathcal{R}.r < \mathcal{S}'.s$ do
          $\mathcal{R}$ = next tuple in R after $\mathcal{R}$;
      end
      while $\mathcal{R}.r > \mathcal{S}'.s$ do
          $\mathcal{S}'$ = next tuple in S after $\mathcal{S}'$;
      end
      $\mathcal{S} = \mathcal{S}'$;
      while $\mathcal{R}.r == \mathcal{S}'.s$ do
          $\mathcal{S} = \mathcal{S}'$;
          while $\mathcal{R}.r == \mathcal{S}.s$ do
              add $\{\mathcal{R}, \mathcal{S}\}$ to result;
              $\mathcal{S}$ = next tuple in S after $\mathcal{S}$;
          end
          $\mathcal{R}$ = next tuple in R after $\mathcal{R}$;
      end
      $\mathcal{S}' = \mathcal{S}$;
  end
```

Cross-references

- ▶ Evaluation of Relational Operators
- ▶ External Sorting
- ▶ Parallel Join Algorithms

Recommended Reading

1. Mishra P. and Eich M.H. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, 1992.

Source

- ▶ Provenance
- ▶ Provenance in Scientific Databases

Space-Filling Curves

Mohamed F. Mokbel[1], Walid G. Aref[2]
[1]University of Minnesota, Minneapolis, MN, USA
[2]Purdue University, West Lafayette, IN, USA

Synonyms

Distance-preserving mapping; Locality-preserving mapping; Multi-dimensional mapping; Linearization

Definition

A space-filling curve (SFC) is a way of mapping the multi-dimensional space into the one-dimensional space. It acts like a thread that passes through every cell element (or pixel) in the multi-dimensional space so that every cell is visited exactly once. Thus, a space-filling curve imposes a linear order of points in the multi-dimensional space. A D-dimensional space-filling curve in a space of N cells (pixels) of each dimension consists of $N^D - 1$ segments where each segment connects two consecutive D-dimensional points. There are numerous kinds of space-filling curves (e.g., Hilbert, Peano, and Gray). The difference between such curves is in their way of mapping to the one-dimensional space, i.e., the order that a certain space-filling curve traverses the multi-dimensional space. The quality of a space-filling curve is measured by its ability in preserving the locality (or relative distance) of multi-dimensional points in the mapped one-dimensional space. The main idea is that any two D-dimensional points that are close by in the

D-dimensional space should be also close by in the one-dimensional space.

Key Points

Space-filling curves are discovered by Peano [3] where he introduces a mapping from the unit interval to the unit square. Hilbert [1] generalizes the idea to a mapping of the whole space. Following Peano and Hilbert curves, many space-filling curves are proposed, e.g., [4]. Space-filling curves are classified into two categories: recursive space-filling curves (RSFC) and non-recursive space-filling curves. An RSFC is an SFC that can be recursively divided into four square RSFCs of equal size. Examples of RSFCs are the Peano SFC, the Gray SFC, and the Hilbert SFC. For the past two decades, recursive space-filling curves have been considered a natural method for locality-preserving mappings. Recursive space-filling curves are special case of fractals [2]. Mandelbrot [2], the father of fractals, derived the term fractal from the Latin adjective fractus. The corresponding Latin verb frangere means "to break" or "to fragment." Thus, fractals divide the space into a number of fragments, visiting the fragments in a specific order. Once a fractal starts to visit points from a certain fragment, no other fragment is visited until the current one is completely exhausted. By dealing with one fragment at a time, fractal locality-preserving mapping algorithms perform a local optimization based on the current fragment.

Cross-references

▶ High-Dimensional Indexing
▶ Space Filling Curves for Query Processing
▶ Spatial Indexing Techniques

Recommended Reading

1. Hilbert D. Ueber stetige abbildung einer linie auf ein flashenstuck. Math. Ann., 459–460, 1891.
2. Mandelbrot B.B. Fractal geometry of nature. W. H. Freeman, New York, 1977.
3. Peano G. Sur une courbe qui remplit toute une air plaine. Math. Ann., 36:157–160, 1890.
4. Sagan H. Space Filling Curves. Springer, Berlin Heidelberg New York, 1994.

Space Partitioning

▶ Indexing and Similarity Search

Space Segmentation

▶ Indexing and Similarity Search

Space-Filling Curve

▶ Fractal

Space-Filling Curves for Query Processing

Mohamed F. Mokbel[1], Walid G. Aref[2]
[1]University of Minnesota, Minneapolis, MN, USA
[2]Purdue University, West Lafayette, IN, USA

Synonyms

Distance-preserving mapping; Locality-preserving mapping; Multi-dimensional mapping; Linearization

Definition

Given a query *Q*, a one-dimensional index structure *I* (e.g., B-tree), and a set of *D* dimensional points, a space-filling curve *S* is used to map the *D* dimensional points into a set of one-dimensional points that can be indexed through *I* for an efficient execution of query *Q*. The main idea is that space-filling curves are used as a way of mapping the multi-dimensional space into the one-dimensional space such that existing one-dimensional query processing and indexing techniques can be applied.

Historical Background

Although space-filling curves were discovered in 1890 [14], their use in query processors has emerged only in the last two decades as it is mainly motivated by the emergence of multi-dimensional applications. In particular, space-filling curves have been used as a mapping scheme that supports spatial join algorithms [13], spatial access methods [2,7], efficient processing of range queries [1,6], and nearest-neighbor queries in [8]. Numerous algorithms are developed for efficiently generating different space-filling curves that include recursive algorithms for the Hilbert SFC [4,15], recursive algorithms for the Peano SFC [15], table-driven algorithms for the Peano and Hilbert SFCs [4]. The clustering and mapping properties of various space-filling curves have been extensively studied in the literature (e.g., see [10,12]).

Foundations

Mapping Scheme

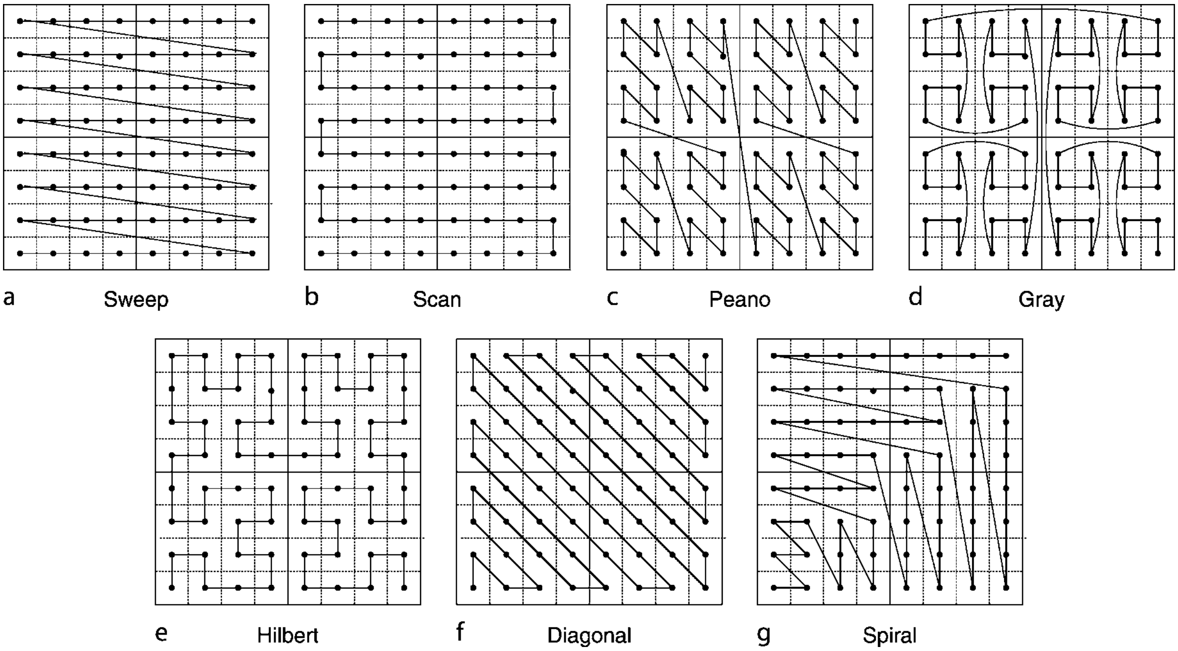
Figures 1 and 2 give examples of two- and three-dimensional space-filling curves with grid size (i.e., number of points per dimension) eight and four, respectively. Space-filling curves are classified into two categories: *recursive* space-filling curves (RSFC) and *non-recursive* space-filling curves. An RSFC is an SFC that can be recursively divided into four square RSFCs of equal size. Non-recursive space-filling curves include the Sweep SFC (Figs. 1a and 2a), the Scan SFC (Figs. 1b and 2b), the Diagonal SFC (Fig. 1f), and the Spiral SFC (Fig. 1g). Recursive space-filling curves include the Peano SFC (Figs. 1c and 2c), the Gray SFC (Figs. 1d and 2d), and the Hilbert SFC (Figs. 1e and 2e). Table 1 gives the first 16 visited points for the Peano, Gray, and Hilbert space-filling curves.

The Peano SFC

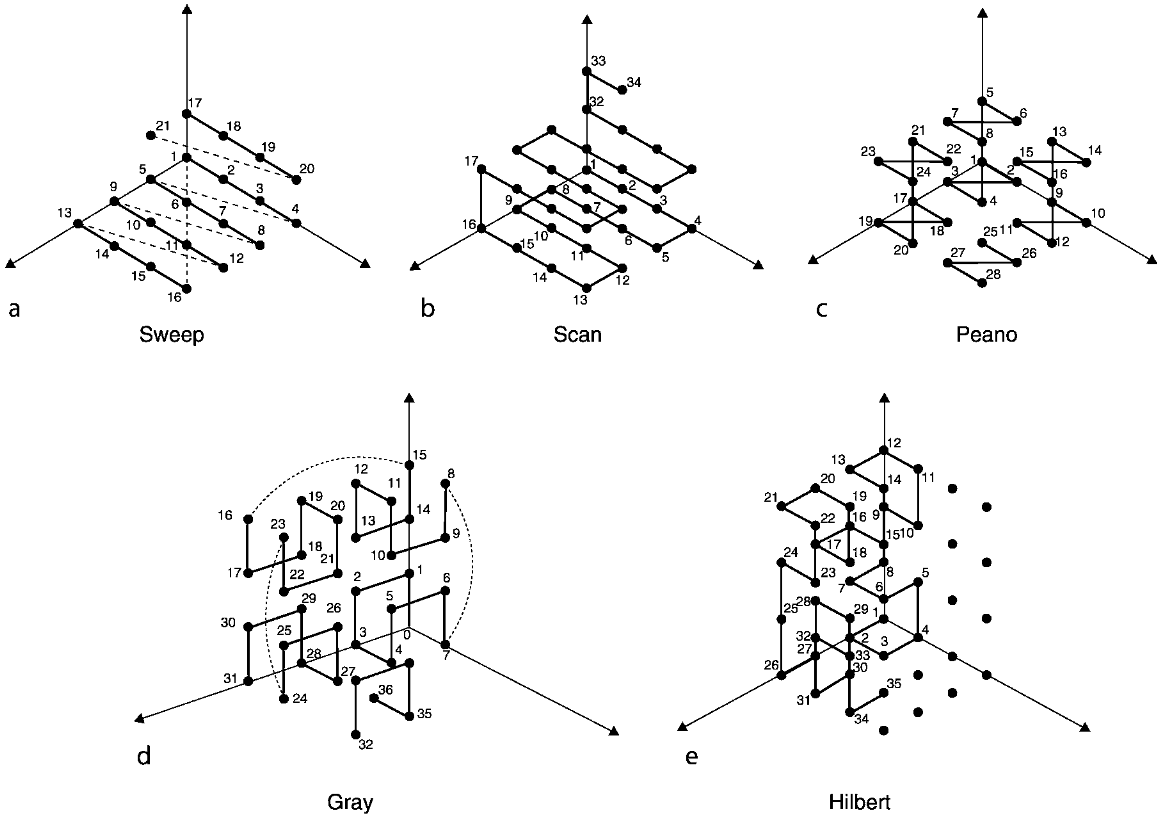
The Peano SFC (Figs. 1c and 2c) is introduced by Peano [14] and is also termed Morton encoding, quad code, bit-interleaving, N-order, locational code, or Z-order. The Peano SFC is constructed recursively as in Fig. 3. The basic shape (Fig. 3a) contains four points in the four quadrants of the space. Each quadrant is represented by two binary digits. The most significant digit is represented by its x position while the least significant digit is represented by its y position. The Peano SFC orders space quadrants in ascending order (00, 01, 10, 11). Figure 3b contains four blocks of Fig. 3a at a finer resolution and is visited in the same order as in Fig. 3a. Similarly, Fig. 3c contains four blocks of Fig. 3b at a finer resolution.

The Gray SFC

The Gray SFC (Figs. 1d and 2d) uses the Gray code representation [5] in contrast to the binary code representation as in the Peano SFC. Figure 4 gives the



Space-Filling Curves for Query Processing. Figure 1. Two-dimensional space-filling curves.



Space-Filling Curves for Query Processing. Figure 2. Three-dimensional space-filling curves.

recursive construction of the Gray SFC. The basic shape (Fig. 4a) contains four points in the four quadrants of the space. The Gray SFC visits the space quadrants in ascending order according to the Gray code (00, 01, 11, 10). Figure 4b is constructed by having the first and fourth blocks as those of Fig. 4a, while the second and the third blocks are the rotation of the blocks in Fig. 4a by 180^0. Similarly, Fig. 4c is constructed from two blocks of Fig. 4b at a finer resolution and two blocks of the rotation of Fig. 4b by 180^0.

The Hilbert SFC

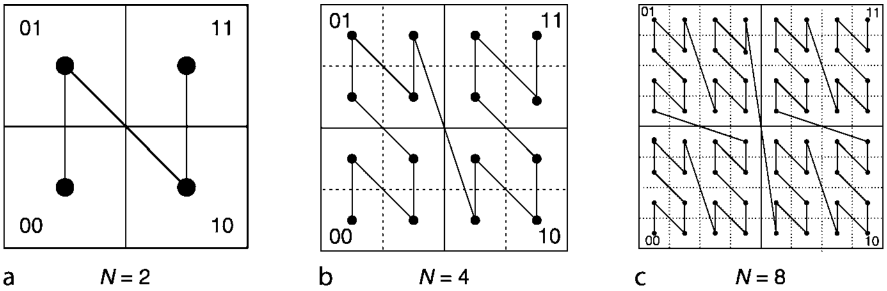
Figure 5 gives the recursive construction of the Hilbert SFC. The basic block of the Hilbert SFC (Fig. 5a) is the same as that of the Gray SFC (Fig. 4a). The basic block is repeated four times at a finer resolution in the four quadrants, as given in Fig. 5b. The quadrants are visited in their gray order. The second and third blocks in Fig. 5b have the same orientation as in Fig. 5a. The first block is constructed from rotating the block of Fig. 5a by 90^0, while the fourth block is constructed by rotating the block of Fig. 5 by -90^0. Similarly, Fig. 5a is constructed from Fig. 5b.

Segment Types

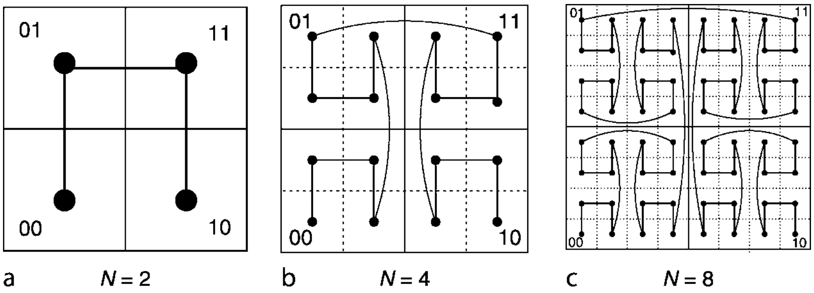
A space-filling curve consists of a set of segments. Each segment connects two consecutive multi-dimensional points. Five different types of segments are distinguished, namely, *Jump*, *Contiguity*, *Reverse*, *Forward*, and *Still*. A *Jump* segment in an SFC is said to happen when the distance, along any of the dimensions, between two consecutive points in the SFC is greater than one. Similarly, a *Contiguity* segment in an SFC is said to happen when the distance, along any of the dimensions, between two consecutive points in the SFC is equal to one. On the other side, a segment in an SFC is termed a *Reverse* segment if the projection of its two consecutive points, along any of the dimensions, results in scanning the dimension in decreasing order. Similarly, a segment in an SFC is termed a *Forward* segment if the projection of its two consecutive points,

Space-Filling Curves for Query Processing. Table 1. The first 16 traversed points by two-dimensional Peano, Gray, and Hilbert space-filling curves

Point	Peano	Gray	Hilbert	Point	Peano	Gray	Hilbert
0	(0,0)	(0,0)	(0,0)	8	(2,0)	(3,3)	(2,2)
1	(0,1)	(0,1)	(0,1)	9	(2,1)	(3,2)	(3,2)
2	(1,0)	(1,1)	(1,1)	10	(3,0)	(2,2)	(3,3)
3	(1,1)	(1,0)	(1,0)	11	(3,1)	(2,3)	(2,3)
4	(0,2)	(1,3)	(2,0)	12	(2,2)	(2,0)	(1,3)
5	(0,3)	(1,2)	(3,0)	13	(2,3)	(2,1)	(1,2)
6	(1,2)	(0,2)	(3,1)	14	(3,2)	(3,1)	(0,2)
7	(1,3)	(0,3)	(2,1)	15	(3,3)	(3,0)	(0,3)



Space-Filling Curves for Query Processing. Figure 3. The Peano SFC.

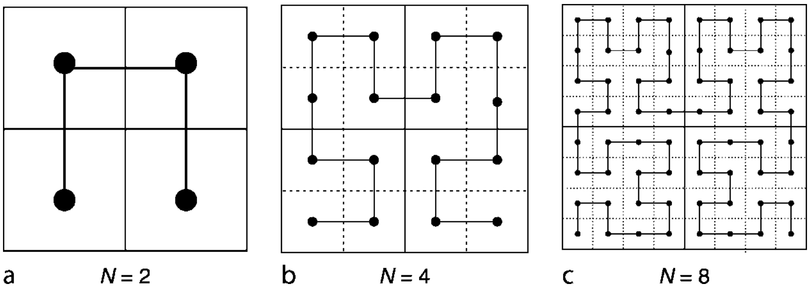


Space-Filling Curves for Query Processing. Figure 4. The Gray SFC.

along any of the dimensions, results in scanning the dimension in increasing order. Finally, a segment in an SFC is termed a *Still* segment when the distance, along any of the dimensions, between the segment's two consecutive points in the SFC is equal to zero. Closed formulas to count the number of *Jump*, *Contiguity*, *Reverse*, *Forward*, and *Still* segments along each dimension can be found in [10].

Irregularity

An optimal locality-preserving space-filling curve is one that sorts multi-dimensional points in ascending order for all dimensions. However, in reality, when a space-filling curve attempts to sort the points in ascending order according to one dimension, it fails to do the same for the other dimensions. A good space-filling curve for one dimension is not necessarily good for the other dimensions. In order to measure the mapping quality of a space-filling curve, the concept of *irregularity* has been introduced as a measure of goodness for the order imposed by a space-filling curve [11]. Irregularity introduces a quantitative measure that indicates the non-avoidable reverse order imposed by space-filling curves for some or all



Space-Filling Curves for Query Processing. Figure 5. The Hilbert SFC.

dimensions. Irregularity is measured for each dimension separately, and gives an indicator of how a space-filling curve is far from the optimal. The lower the irregularity, the better the space-filling curve. The irregularity is formally defined as: For any two points, say P_i and P_j, in the D-dimensional space with coordinates $(P_i.u_0,P_i.u_1,...,P_i.u_{D-1})$, $(P_j.u_0,P_j.u_1,...,P_j.u_{D-1})$, respectively, and for a given space-filling curve S, if S visits P_i before P_j, an irregularity occurs between P_i and P_j in dimension k iff $P_j.u_k < P_i.u_k$. Closed formulas to count the number of irregularities for various space-filling curves can be found in [11].

Key Applications

Pre-processing for Multi-dimensional Applications: Multimedia Databases, GIS,and Multi-dimensional Indexing

Mapping the multi-dimensional space into the one-dimensional domain plays an important role in applications that involve multi-dimensional data. Multimedia databases, Geographic Information Systems (GIS), QoS routing, and image processing are examples of multi-dimensional applications. Modules that are commonly used in multi-dimensional applications include searching, sorting, scheduling, spatial access methods, indexing, and clustering. Considerable research has been conducted for developing efficient algorithms and data structures for these modules for one-dimensional data. In most cases, modifying the existing one-dimensional algorithms and data structures to deal with multi-dimensional data results in spaghetti-like programs to handle many special cases. The cost of maintaining and developing such code degrades the system performance. Mapping from the multi-dimensional space into the one-dimensional domain provides a pre-processing step for multi-dimensional applications. The pre-processing step takes the multi-dimensional data as input and outputs the same set of data represented in the one-dimensional domain. The idea is to keep the existing algorithms and data structures independent of the dimensionality of data. The objective of the mapping is to represent a point from the D-dimensional space by a single integer value that reflects the various dimensions of the original space. Such a mapping is called a locality-preserving mapping in the sense that, if two points are near to each other in the D-dimensional space, then they will be near to each other in the one-dimensional space.

Network-Attached Storage Devices NASDs

Writing efficient schedulers is becoming a very challenging task, given the increase in demand of such systems. Consider the case of network-attached storage devices (NASDs) [3] as a building block for a multimedia server. NASDs are smart disks that are attached directly to the network. In a multimedia server, a major part of a NASD function goes towards fulfilling the real-time requests of users. This involves disk and network scheduling with real-time constraints, possibly with additional requirements like request priorities, and quality-of-service guarantees. NASDs requirements can be mapped in the multi-dimensional space and a SFC-based scheduler is used. The type of space-filling curve used in NASD scheduling is determined by its requirements. For example, in NASD, if reducing the number of requests that lose their deadlines is more important than increasing the disk or network bandwidth, then the real-time deadline dimension of the scheduling space will be favored. As a result, a space-filling curve with intentional bias is favored.

Multimedia Disk Scheduling

Consider the problem of disk scheduling in multimedia servers [9]. In addition to maximizing the bandwidth of the disk, the scheduler has to take into consideration the real-time constraints of the page requests, e.g., as in the case of video streaming. If clients are prioritized based on quality-of-service guarantees, then the disk scheduler might as well consider the priority of the requests in its disk queue. Writing a disk scheduler that handles real-time and QoS constraints in addition to maximizing the disk bandwidth is challenging and a hard task. Scheduler parameters can be mapped to space dimensions and an SFC-based scheduler is used. The reader is referred to [9] to get more insight about the applicability of the irregularity in multi-media disk schedulers.

Future Directions

Future directions for space-filling curves include: (i) exploiting new multi-dimensional applications that can make use of the properties of space-filling curves, (ii) analyzing the behavior of various space-filling curves in high-dimensional space, (iii) providing automated modules with the ability of choosing the appropriate space-filling curve for a given application, and (iv) developing new space-filling curves that are tailored to specific applications.

Cross-references

▶ High-Dimensional Indexing
▶ Space-Filling Curves
▶ Spatial Indexing Techniques

Recommended Reading

1. Faloutsos C. Gray codes for partial match and range queries. IEEE Trans. Software Eng, 14(10):1381–1393, October 1988.
2. Faloutsos C. and Rong Y. Dot: a spatial access method using fractals. In Proc. 7th Int. Conf. on Data Engineering, 1991, pp. 152–159.
3. Gibson G., Nagle D., Amiri K., Butler J., Chang F.W., Gobioff H., Hardin C., Riedel E., Rochberg D., and Zelenka J. File server scaling with network-attached secure disks. In Proc. 1997 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Comp. Syst., 1997, pp. 272–284.
4. Goldschlager L.M. Short algorithms for space-filling curves. Software–Prac. Exper., 11(1):99–100, 1981.
5. Gray F. Pulse code communications. US Patent 2632058, 1953.
6. Jagadish H.V. Linear clustering of objects with multiple attributes. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 332–342.
7. Kamel I. and Faloutsos C. Hilbert r-tree: an improved r-tree using fractals. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 500–509.
8. Liao S., Lopez M.A., and Leutenegger S.T. High dimensional similarity search with space-filling curves. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 615–622.
9. Mokbel M.F., Aref W.G., El-Bassyouni K., and Kamel I. Scalable multimedia disk scheduling. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 498–509.
10. Mokbel M.F., Aref W.G., and Kamel I. Analysis of multi-dimensional space-filling curves. GeoInformatica, 7(3):179–209, September 2003.
11. Mokbel M.F. and Aref W.G. Irregularity in multi-dimensional space-filling curves with applications in multimedia databases. In Proc. Int. Conf. on Information and Knowledge Management, 2001, pp. 512–519.
12. Moon B., Jagadish H.V., Faloutsos C., and Salz J. Analysis of the clustering properties of hilbert space-filling curve. IEEE Trans. Knowl. Data Eng., 13(1):124–141, 2001.
13. Orenstein J.A. Spatial query processing in an object-oriented database system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 326–336.
14. Peano G. Sur une courbe qui remplit toute une air plaine. Math. Ann., 36:157–160, 1890.
15. Witten I.H. and Wyvill B. On the generation and use of space-filling curves. Software–Prac. Exper., 3:519–525, 1983.

Space-Span (in Part)

▶ Context

Spamdexing

▶ Web Spam Detection

Span

▶ Time Interval

Sparse Index

Mirella M. Moro[1], Vassilis J. Tsotras[2]
[1]Federal University of Rio Grande do Sul,
Porte Alegre, Brazil
[2]University of California-Riverside, Riverside,
CA, USA

Synonyms

Non-dense Index

Definition

Consider a tree-based index on some numeric attribute *A* of a relation *R*. If an index record (of the form <*search-key, pointer*>) is created for *some* of the values that appear in attribute *A*, then this index is *sparse.*

Key Points

Tree-based indices are built on numeric attributes and maintain an order among the indexed search-key values. Hence, they provide efficient access to the records of a relation by attribute value. Consider for example an index built on attribute *A* of relation *R.* The leaf pages of the index contain *index-records* of the form <*search-key, pointer*>, where search-key corresponds to a value from the indexed attribute *A* and *pointer* points to the respective record in the indexed relation *R* with that attribute value. If not all distinct values that appear in *R.A* also appear in index records, this index is *sparse*, otherwise it is called *dense.*

A sparse index needs a way to access even the relation records with values that do not directly appear in the index. Hence it is required that the indexed relation is *ordered* according to the values of the indexed attribute *A*; in this way the relation order can be used to access values not directly indexed by the sparse index.

Tree-indices are further categorized by whether their search-key ordering is the same with the relation file's physical order (if any). If the search-key of a tree-based index is the same as the ordering attribute of a (ordered) file then the index is called *primary.* An index built on any non-ordering attribute of a file is called *secondary.* Hence a primary index is also sparse while a secondary index should also be dense.

A dense index is typically larger than a sparse index (since all search-key values are indexed) and thus requires more space. It also needs to be updated for every relation update that involves the attribute value being indexed.

Cross-references

- ▶ Access Methods
- ▶ B+-Tree
- ▶ Index Sequential Access Method (ISAM)
- ▶ ISAM
- ▶ Primary Index

Recommended Reading

1. Elmasri and Ramez Navathe. Shamkant B. Fundamentals of Database Systems (5th edn.). Addison-Wesley, Reading, MA, 2007.
2. Manolopoulos, Yannis Theodoridis, and Yannis Tsotras. Vassilis J. Advanced Database Indexing. Kluwer, Dordecht, 1999.
3. Silberschatz, Avi Korth, and Henry F. S. Sudarshan Database System Concepts (5th edn.). McGraw-Hill, NY, 2006.

Spatial Access Methods

▶ Spatial Indexing Techniques

Spatial Analysis

▶ Spatial Data Analysis

Spatial and Spatio-Temporal Data Models and Languages

Markus Schneider
University of Florida, Gainesville, FL, USA

Definition

A *data model* provides a formalism consisting of a notation for describing data of interest and of a set of operations for manipulating these data. It abstracts from reality and provides a generalized view of data representing a specific and bounded scope of the real world. In the context of databases, a data model describes the organization, that is, the structure, of a database. In the context of complex objects like video, genomic, and multimedia objects, a data model describes a type system consisting of data types, operations, and predicates. Spatial and spatio-temporal data models are of this second kind. A *spatial data model* is a data model defining the properties of and operations on static objects in space. These objects are described by *spatial data types* like *point* (for example, representing the locations of cities in the US), *line* (for example, describing the ramifications of the Nile Delta), and *region* (for example, depicting school districts). Operations on spatial data types include, for instance, the geometric *intersection*, *union*, and *difference* of spatial objects, the computation of the *length* of a line or the *area* of a region, the test whether two spatial objects *overlap* or *meet*, and whether one object is *north* or *southeast* of another object. A *spatio-temporal data model* is a data model representing the temporal evolution of spatial objects over time. These evolutions can be discrete, that is, they happen from time to time (for example, the change of the boundary of a land parcel) or continuous, that is, they happen permanently and smoothly (for example, the devastating trajectory of a hurricane). In the continuous case, one speaks about *moving objects* and represents them by *spatio-temporal data types* like *moving point* (for example, recording the route of a cell phone user), *moving line* (for example, representing the boundary of a tsunami), and *moving region* (for example, describing the motion of an air polluted cloud). Operations on spatio-temporal data types comprise, for instance, the spatio-temporal *intersection*, *union*, and *difference* of moving objects, the computation of the *trajectory* of a moving point as a line object, the determination of the *location* of a moving object at a particular time, the calculation of a moving object during a given set of intervals, and the test whether a moving point *enters* or *crosses* a moving region. *Spatial* and *spatio-temporal query languages* enable the user to query databases enhanced by these concepts.

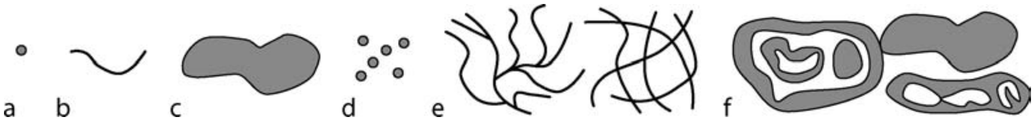
Historical Background

The interest to store geometric data in databases began in the late 1970s. Due to the increasing success of relational databases, the first approach has been to decompose a spatial object recursively into its constituent parts until they can be stored in tables. For example, this approach decomposes a polygon into its set of segments. Each segment is decomposed into a pair of points. A point is decomposed into a pair of two float numbers. Float numbers are a DBMS data type and can be stored in a table. This approach has revealed a number of fundamental drawbacks. Since all lines and polygons are decomposed into their constituent parts scattered as tuples over a relation, a spatial object is not treated as an entity or unit but only *corresponds* to a collection of tuples. Since this approach is based on standard domains and has no concept of spatial data types, it cannot provide and support any meaningful geometric operations. A more detailed discussion can be found in [7].

Classical research on time-varying geometric data has focused on *discrete* changes of spatial objects over time. For example, cadastral applications deal with the management of land parcels whose boundaries can change from time to time due to specific legal actions such as splitting, merging, or land consolidation. Political boundaries can suddenly disappear, as the reunification of West and East Germany shows. Different approaches have been proposed to model these discrete changes. One of them is to enhance *temporal databases* [12] with spatial data types. Each discrete change leads to a new stored snapshot with a modified spatial object in the temporal database. Another approach [15] keeps a single version of each spatial object only but annotates each of its components (for instance, a point or a segment) with a temporal element indicating the period of validity or existence of this component. Hence, discrete changes of a spatial object are registered within the object.

Foundations

Spatial data types form the basis of a large number of data models and query languages for spatial data. They are extensively leveraged by *spatial databases* [9] and embedded as attribute data types into their data models, that is, in the same way as standard data types such as *integer*, *real*, and *string*. The geometric types are designed as *abstract data types*, that is, the internal structure of a spatial object is hidden from the user, and its features can only be retrieved by (abstract) operations on this object. In this manner, they provide a high-level view of geometric data. One can distinguish the older generation of *simple* spatial data types and the newer generation of *complex* spatial data types, depending on the spatial complexity the types are able to model. In the two-dimensional space, simple spatial data types only provide simple object



Spatial and Spatio-Temporal Data Models and Languages. Figure 1. Examples of a simple point object (a), a simple line object (b), a simple region object (c), a complex point object (d), a complex line object (e), and a complex region object (f).

structures like single points, continuous lines, and simple regions (Fig. 1a–c). However, from an application perspective, simple spatial data types have turned out to be inadequate abstractions for spatial applications since they are insufficient to cope with the variety and complexity of geographic reality. From a formal perspective, they are not closed under the geometric set operations *intersection*, *union*, and *difference*. This means that these operations applied to two simple spatial objects can produce a spatial object that is *not* simple. Complex spatial data types solve these problems. They provide universal and versatile spatial objects and are closed under geometric set operations. They allow objects with multiple components, region components that may have holes, and line components that may model ramified, connected geometric networks (Fig. 1d–f).

As an example, in a relational setting, states, cities, and rivers are represented in the following relations:

```
states(sname: string, area: region)
cities(name: string, population: integer,
location: point)
rivers(name: string, route: line)
```

Queries can then be formulated by employing operations and predicates on spatial attribute values within an extended standard database query language such as SQL, leading to *Spatial SQL* [1]. Assume that the following operations and predicates are available:

area:	*region*	→ *real*
inside:	*point* × *region*	→ *bool*
intersection:	*line* × *region*	→ *line*
length:	*line*	→ *real*
meet:	*region* × *region*	→ *bool*

The predicates *inside* and *meet* represent *topological relationships* [8] that characterize the relative position between spatial objects. The operation *length* is a numerical function computing the length of a *line* object. The operation *intersection* computes the part of a *line* object intersecting a *region* object.

One can now pose queries: What is the total population of the cities in France?

```
select sum(c.pop) as total
from   cities as c, states as s
where  c.location inside s.area and
s.name = 'France'
```

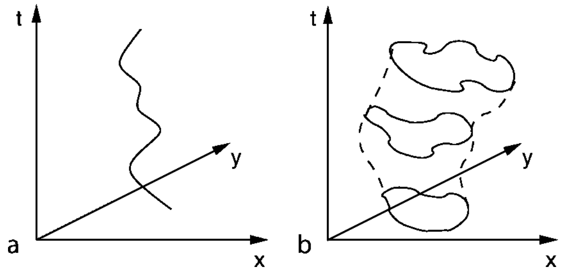
Compute the part of the river Rhine that is located within Germany and determine its length.

```
select intersection(r.route, s.area) as rhine,
       length(intersection(r.route,
       s.area)) as len
from   rivers as r, states as s
where  r.name = 'Rhine' and s.name = 'Germany'
```

Make a list that shows for each state the number of its neighbor states, and their total area.

```
select   s.name, count(*),
sum(area(t.area))
from     states as s, states as t
where    s.area meet t.area
group by s.name
```

Spatio-temporal data types enable the user to describe the dynamic behavior of spatial objects over time. The dynamic behavior refers to the continuous change of the locations of spatial objects over time. That is, the spatial objects move, and they are therefore called *moving objects*. They are stored in special spatio-temporal databases called *moving objects databases* [6]. In the same way as spatial data types, spatio-temporal data types are also designed as abstract data types and embedded as attribute types into a DBMS data model. Spatio-temporal data types are available for moving points (type *mpoint* for short), moving lines (*mline*), and moving regions (*mregion*). In case of moving regions, one can also represent the change of their extent and shape over time. Conceptually, a moving point is a function *f*: *time* → *point*, a moving line is a function *f*: *time* → *line*, and a moving region is a function *f*: *time* → *region*. For example, for a moving region this means that at each time instant an object of type *region* has to be returned. Geometrically, moving objects correspond to the three-dimensional shapes. In case of a moving point it is a three-dimensional line (Fig. 2a) and in case of a moving region it is a volume (Fig. 2b). One can distinguish moving objects databases that model and query the history of movement for spatio-temporal analysis [2,5] and moving objects databases that model, predict, and query current and future movement [10,11]. In the latter case, location updates require a balancing of update costs and



Spatial and Spatio-Temporal Data Models and Languages. Figure 2. Examples of a moving point object (a) and a moving region object (b).

imprecision [14] and introduce the feature of uncertainty [13].

As an example, consider relations describing the movements of airplanes or storms:

```
flight(id: string, from: string,
to: string, route: mpoint)
weather(id: string, kind:
string, area: mregion)
```

One can pose queries by employing operations and predicates on spatio-temporal attribute values within an extended standard database query language such as SQL, leading to *Spatio-Temporal Query Language* (*STQL*) [3]. Assume that the following operations and predicates are available:

deftime:	*mpoint*	→ *periods*
Disjoint:	*mpoint* × *mregion*	→ *bool*
distance:	*mpoint* × *mpoint*	→ *mreal*
Inside:	*mpoint* × *mregion*	→ *bool*
intersection:	*mpoint* × *mregion*	→ *mpoint*
meet:	*point* × *region*	→ *bool*
min:	*mreal*	→ *real*
trajectory:	*mpoint*	→ *line*

The function *deftime* returns the set of time intervals when a moving point is defined. The *spatio-temporal predicate* [3,4,6] *Disjoint* checks whether a moving point and a moving region are disjoint for some period. The function *distance* computes the distance between two moving points and is a real-valued function of time, captured here in a data type *mreal* for *moving reals*. The spatio-temporal predicate *Inside* tests whether a moving point is located inside a moving region for some period. The operation *intersection* returns the part of a moving point whenever it lies inside a moving region, which is a moving point again. The topological predicate *meet* checks whether a point object is located on the boundary of a region object. The function *min* yields the minimal value assumed over time by a moving real.

One can now pose queries: Find all flights from Frankfurt that are longer than 5,000 kms.

```
select id
from   flight
where  from = 'FRA' and length
(trajectory(route)) > 5000
```

Retrieve any pairs of airplanes, which, during their flight, came closer to each other than 500 m.

```
select f.id, g.id
from   flight as f, flight as g
where  f.id <> g.id and min(distance
   (f.route, g.route)) < 0.5
```

At what time was flight TB691 within a snowstorm with id RS316?

```
select   deftime(intersection(f.route,
         w.area))
from     flight as f, weather as w
where    f.id = 'TB691' and w.id = 'RS316'
```

Which are the planes that ran into a hurricane and had to traverse it?

```
select f.id, w.id
from   flight as f, weather as w
where  w.kind = 'hurricane' and
       f.route Disjoint >> meet >>
       Inside >> meet >> Disjoint w.area
```

The term *Disjoint* >> *meet* >> *Inside* >> *meet* >> *Disjoint* is a spatio-temporal predicate that is composed of a *temporal sequence* of the basic spatio-temporal predicates *Disjoint* and *Inside* as well as the topological predicate *meet*. The *temporal composition operator* is indicated by the symbol >>. The query above searches for a spatio-temporal pattern in which a plane is disjoint from a hurricane for some period, then meets the boundary of the hurricane at a time instant, is inside the hurricane for some period, meets the boundary of the hurricane again at a time instant, and is disjoint again from the hurricane for some period. The alternating sequence of topological predicates that hold for some period or for some time instant is characteristic for composite spatio-temporal predicates.

Key Applications

Spatial data models containing spatial data types, operations, and predicates are a universal and general concept for representing geometric information in all kinds of spatial applications. They have found broad acceptance in spatial extension packages of commercially and publicly available database systems as well as in geographic information systems. Further, all applications in the geosciences (for example, geography, hydrology, soil sciences) as well as many applications in government and administration (for example, cadastral applications, urban planning) already benefit from them. Independent studies have shown that about 80% of all data have spatial features (like geometric attributes) or a spatial reference (like an address). Thus, it is not surprising that independent international studies have predicted that geoinformation technology will belong to the most important and promising technologies in the future, besides biotechnology and nanotechnology.

The usage of moving objects in databases and especially in geographic information systems is still in its infancy since it is a relatively new technology. But increasingly, applications like location management, GPS-equipped PDAs, phones, and vehicles, navigation systems, RFID-tag tracking, sensor networks, hurricane research, and national security show interest in moving objects databases.

Cross-references

- ▶ Spatial Data Types
- ▶ Spatio-temporal Trajectories
- ▶ Temporal Database

Recommended Reading

1. Egenhofer M.J. Spatial SQL: a query and presentation language. IEEE Trans. Knowl. and Data Eng., 6(1):86–94, 1994.
2. Erwig M., Güting R.H., Schneider M., and Vazirgiannis M. Spatio-temporal data types: an approach to modeling and querying moving objects in databases, Geoinformatica, 3(3): 265–291, 1999.
3. Erwig M. and Schneider M. Developments in spatio-temporal query languages. In Proc. IEEE International Workshop on Spatio-Temporal Data Models and Languages, 1999, pp. 441–449.
4. Erwig M. and Schneider M. Spatio-Temporal Predicates. IEEE Trans. Know. and Data Eng., 14(4):1–42, 2002.
5. Güting R.H., Böhlen M.H., Erwig M., Jensen C.S., Lorentzos N.A., Schneider M., and Vazirgiannis M. A foundation for representing and querying moving objects. ACM Trans. Database Syst., 25(1):1–42, 2000.
6. Güting R.H. and Schneider M. Moving Objects Databases. Morgan Kaufmann, San Fransisco, CA, USA, 2005.
7. Schneider M. Spatial Data Types for Database Systems – Finite Resolution Geometry for Geographic Information Systems. LNCS 1288. Springer, Berlin Heidelberg New York, 1997.
8. Schneider M. and Behr T. Topological relationships between complex spatial objects. ACM Trans. Database Syst., 31(1): 39–81, 2006.
9. Shekar S. and Chawla S. Spatial Databases: A Tour. Prentice-Hall, Englewood Cliffs, NJ, USA, 2003.
10. Sistla A.P., Wolfson O., Chamberlain S., and Dao S. Modeling and Querying Moving Objects. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 422–432.
11. Sistla A.P., Wolfson O., Chamberlain S., and Dao S. Querying the uncertain position of moving objects. In: Temporal Databases: Research and Practice. O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399. Springer, Berlin Hiedelberg New York, 1998, pp. 310–337.
12. Tansel A.U., Clifford J., Gadia S., Jajodia S., Segev A., and Snodgrass R.T. (eds.). Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, 1993.
13. Trajcevski G., Wolfson O., Hinrichs K., and Chamberlain S. Managing uncertainty in moving objects databases. ACM Trans. Database Syst., 29(3):463–507, 2004.
14. Wolfson O., Chamberlain S., Dao S., Jiang L., and Mendez G. Cost and Imprecision in Modeling the Position of Moving Objects. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 588–596.
15. Worboys M.F. A unified model for spatial and temporal information. Comput. J., 37(1):25–34, 1994.

Spatial Anonymity

PANOS KALNIS, GABRIEL GHINITA
National University of Singapore, Singapore, Singapore

Synonyms

Spatial k-anonymity; Privacy-preserving spatial queries; Anonymity in location-based services

Definition

Let U be a user who is asking via a mobile device (e.g., phone, PDA) a query relevant to his current location, such as "find the nearest betting office." This query can be answered by a Location Based Service (LBS) in a public web server (e.g., Google Maps, MapQuest), which is not trustworthy. Since the query may be sensitive, U uses encryption and a pseudonym, in order to protect his privacy. However, the query still contains the exact location, which may reveal the identity of U. For example, if U asks the query within his residence, an attacker may use public information (e.g., white pages) to associate the location with U. Spatial k-Anonymity (*SKA*) solves this problem by ensuring that an attacker cannot identify U as the querying user with probability larger than $1/k$, where k is a user-defined anonymity requirement. To achieve this, a centralized or distributed anonymization service replaces the exact location of U with an area (called Anonymizing Spatial Region or ASR). The ASR encloses U and at least $k-1$ additional users. The LBS receives the ASR and retrieves the query results for *any* point inside the ASR. Those results are forwarded to the anonymization service, which removes the false hits and returns the actual answer to U.

Historical Background

The embedding of positioning capabilities (e.g., GPS) in mobile devices has triggered several exciting applications. At the same time, it has raised serious concerns [1] about the risks of revealing sensitive information in location based services (LBS). An untrustworthy LBS may use public knowledge to relate a set of query coordinates to a specific user, even if the user-id is removed. In practice, users are reluctant to access a service that may disclose their political/religious affiliations or alternative lifestyles. Furthermore, users might be hesitant to ask innocuous queries such as “find the restaurants in my vicinity” since, once their identity is revealed, they may face unsolicited advertisements. Spatial k-Anonymity (SKA) aims at solving this problem.

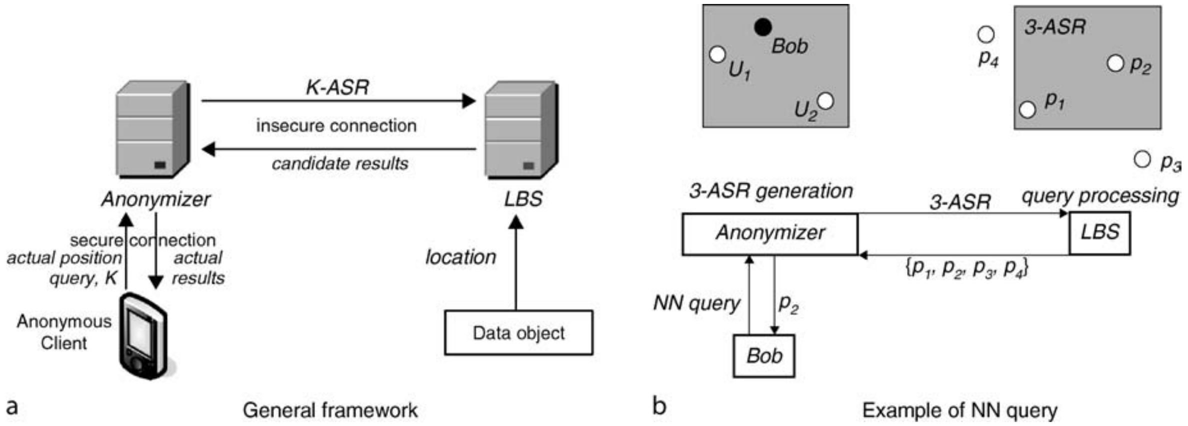
k-Anonymity [15] has been used in relational databases for publishing census, medical and voting registration data (often called microdata). A relation satisfies k-anonymity if every tuple is indistinguishable from at least $k-1$ other tuples with respect to a set of quasi-identifier (QI) attributes. QIs are attributes (e.g., date of birth, gender, zip code) that can be linked to publicly available data to identify individuals. In the context of location based services, the k-Anonymity concept translates as follows: given a query, an attack based on the query location must not be able to identify the query source with probability larger than $1/k$.

A straightforward method is to pick $k-1$ random users and forward k independent queries (including the real one) to the LBS. This method achieves SKA because the query could originate from any client with equal probability $1/k$. However, depending on the value of k, a potentially large number of locations are transmitted and processed by the LBS. Also, the exact locations of k users are revealed, which is undesirable in many applications.

Most of the existing work adopts the framework of Fig. 1a, which assumes a trusted server, called *anonymizer*. Users access the anonymizer through a secure connection and periodically report their position. A querying user U sends his location-based query to the anonymizer, which removes the user-id and transforms the location of U through a technique called cloaking. Cloaking hides the actual location by an *anonymizing spatial region* (k-ASR or ASR), which is an area that encloses U, as well as at least $k-1$ other users. The anonymizer then sends the ASR to the LBS, which returns a set of *candidate* results that satisfy the query condition for any possible point inside the ASR.

Figure 1b presents an example, where Bob asks for the nearest betting office. Bob forwards his request to the anonymizer, together with his anonymity requirement k. Assuming that $k = 3$, the anonymizer generates a 3-ASR (shaded rectangle) that contains Bob and two other users U_1, U_2 (the anonymizer knows the exact locations of all users). Then, it sends this 3-ASR to the LBS, which finds all betting offices that can be the nearest-neighbor (NN) of any point in the 3-ASR (the LBS does not know where Bob is). This candidate set (i.e., $\{p_1, p_2, p_3, p_4\}$) is returned to the anonymizer, which filters the false hits and forwards the actual NN (in this case p_2) to Bob. Even if an attacker knows the location of Bob and the other users, she can only ascertain that the query originated from Bob with probability $1/3$.

The privacy of user locations has also been studied in the context of related problems. *Probabilistic Cloaking* [2] does not apply the concept of SKA; instead, the ASR is a closed region around the query point, which is independent of the number of users inside. Given an ASR, the LBS returns the probability of each candidate result satisfying the query, based on its location with respect to the ASR. Kamat et al. [10] propose a model for sensor networks and examine the privacy characteristics of different sensor routing protocols. Hoh and Gruteser [8] describe techniques for hiding the



Spatial Anonymity. Figure 1. Framework and example for spatial *k*-anonymity.

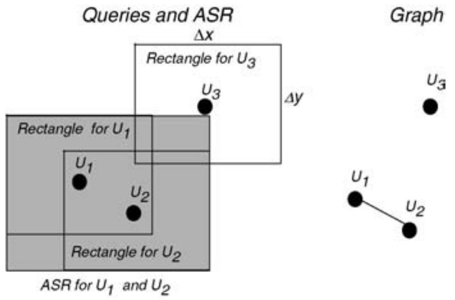
trajectory of users in applications that continuously collect location samples.

Foundations

In order to solve the SKA problem, the following assumptions about the capabilities of the attacker apply:

1. The attacker intercepts the ASR, which implies that either the LBS is not trustworthy, or the communication channel between the anonymizer and the LBS is not secure.
2. The attacker knows the cloaking algorithm used by the anonymizer. This is common in the security literature where algorithms are typically public.
3. The attacker can obtain the current locations of all users. This assumption is motivated by the fact that users may often issue queries from locations (e.g., home, office), which may be identified through physical observation, triangulation, telephone catalogs etc. However, it is difficult to model the exact amount of knowledge an attacker can gain. Therefore, the third assumption dictates that the anonymization method should be provably secure under the worst-case scenario.
4. The attacker uses current data, but not historical information about movement and behavior patterns of particular clients (e.g., a user often asking a particular query at a certain location or time). Therefore, SKA is defined only for *snapshot*, but not for continuous queries.

Given these assumptions, a spatial cloaking algorithm is said to be *secure*, if for any *U* and any *k* the probability of identifying *U* as the querying user is at most 1/*k*.



Spatial Anonymity. Figure 2. Example of *Clique Cloak*.

In addition to being secure, spatial cloaking should be efficient and effective. *Efficiency* means that the CPU and I/O cost of generating the ASR (at the anonymizer) should be minimized for better scalability and faster service. *Effectiveness* refers to the area of the ASR, which should also be minimized. Specifically, a large ASR incurs high processing overhead (at the LBS) and network cost (for transferring numerous candidate results from the LBS to the anonymizer). The rest of this article discusses several representative algorithms for the SKA problem.

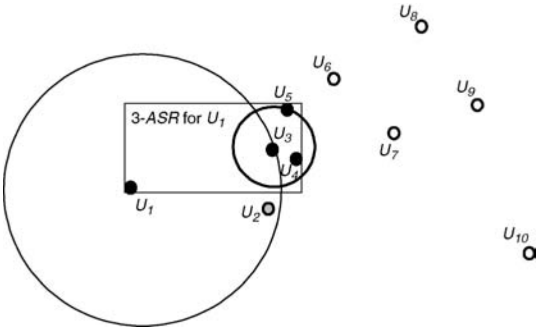
In *Clique Cloak* [5], each query defines an axis-parallel rectangle whose centroid lies at the user location and whose extents are Δx, Δy. Figure 2 illustrates the rectangles of three queries located at U_1, U_2, U_3, assuming that they all have the same Δx and Δy. The anonymizer generates a graph where a vertex represents a query: two queries are connected if the corresponding users fall in the rectangles of each other. Then, the graph is searched for cliques of *k* vertices and the minimum bounding rectangle (*MBR*) of the corresponding user areas, forms the ASR sent to the LBS. Continuing the example of Fig. 2, if $k = 2$, U_1 and U_2 form a 2-clique and the MBR of their respective rectangles is generated so that both queries are processed together. On the other hand, U_3 cannot be processed immediately, but it has to wait until a new query (generating a 2-clique with U_3) arrives. *Clique Cloak* allows users to specify a temporal interval Δt such that, if a clique cannot be found within Δt, the query is rejected. Therefore, *Clique Cloak* may affect the quality of service, as some queries may be delayed or completely rejected. The algorithms discussed next do not suffer from this drawback.

Simply generating an ASR that includes k clients is not sufficient for SKA. Consider for instance an algorithm, called *Center Cloak* in the sequel, that given a query from U, finds his $k - 1$ closest users and sets the ASR as the MBR that encloses them. In fact, a similar technique is proposed in [4] for anonymization in peer-to-peer systems (i.e., the ASR contains the query issuing peer and its $k - 1$ nearest nodes). However, by construction, the querying user U is often closest to the ASR center. Thus, a simple "center-of-ASR" attack would correctly guess U with probability that far exceeds $1/k$, especially for large k values.

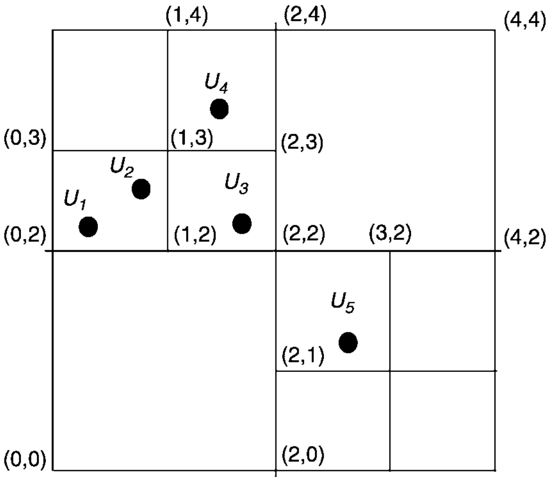
Nearest Neighbor Cloak (*NN-Cloak*) [9] is a randomized variant of *Center Cloak*, which is not vulnerable to the *center-of-ASR attack*. Given a query from U, *NN-Cloak* first determines the set S_0 containing U and his $k - 1$ nearest users. Then, it selects a random user U_i from S_0 and computes the set S_1, which includes U_i and his $k - 1$ nearest neighbors (NN). Finally, *NN-Cloak* obtains $S_2 = S_1 \cup U$. This step is essential, since U is not necessarily among the NNs of U_i. The ASR is the MBR enclosing all users in S_2. Figure 3 shows an example of *NN-Cloak*, where U_1 issues a query with $k = 3$. The two NNs of U_1 are U_2, U_3, and $S_0 = \{U_1, U_2, U_3\}$. *NN-Cloak* randomly chooses U_3 and issues a 2-NN query, forming $S_1 = \{U_3, U_4, U_5\}$. The 3-ASR is the MBR enclosing $S_2 = \{U_1, U_3, U_4, U_5\}$. It is not vulnerable to the *center-of-ASR attack* since the probability of U being near the center of the ASR is at most $1/k$ (due to the random choice).

In another approach called *Casper* [12], the anonymizer maintains the locations of the clients using a pyramid data structure, similar to a Quad-tree [14], where the minimum cell size corresponds to the anonymity resolution. Once the anonymizer receives a query from U, it uses a hash table on the user-id pointing to the lowest-level cell c where U lies. If c contains enough users (i.e., $|c| \geq k$), it becomes the ASR. Otherwise, the horizontal c_h and vertical c_v neighbors of c are retrieved. If the union of c with c_h or c_v contains at least k users, the corresponding union becomes the ASR. Else, *Casper* retrieves the parent of c and repeats this process recursively. Figure 4 shows an example. Cells are denoted by the coordinates of their lower-left and upper-right points. Assume a query q with $k = 2$. If q is issued by U_1 or U_2, the ASR is cell $\langle(0,2),(1,3)\rangle$. If q is issued by U_3 or U_4, the ASR is the union of cells $\langle(1,2),(2,3)\rangle \cup \langle(1,3),(2,4)\rangle$. Finally, if q is issued by U_5, the ASR is the entire data space.

Interval Cloak [7] is similar to *Casper* in terms of both the data structure used by the anonymizer (i.e., a



Spatial Anonymity. Figure 3. Example of *NN-Cloak*.



Spatial Anonymity. Figure 4. Example of *Casper*.

Quad-tree) and the cloaking algorithm. The main difference is that *Interval Cloak* does not consider neighboring cells at the same level when determining the ASR, but ascends directly to the ancestor level. For instance, a query with $k = 2$ issued by U_3 or U_4, would generate the ASR $\langle(0,2),(2,4)\rangle$ (instead of $\langle(1,2),(2,4)\rangle$ for *Casper*).

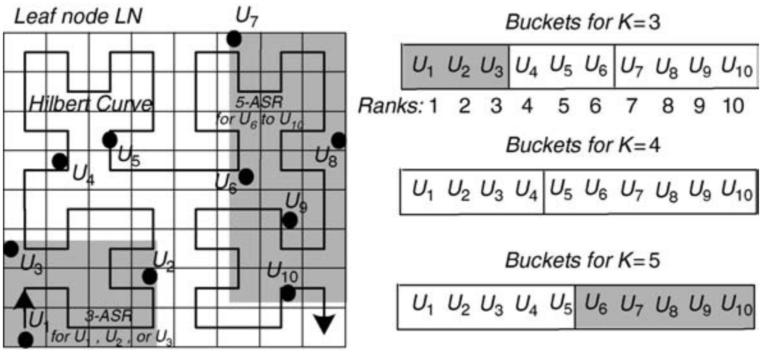
An important observation is that *Casper* and *Interval Cloak* are secure only for uniform data [8]. In the example of Fig. 4, although U_1 to U_4 are in the 2-ASR of U_5, U_5 is not in the 2-ASR of any of those users. Consequently, an attacker that detects an ASR covering the entire space can infer with high probability that it originates from U_5 (by assumption 3, in the worst case the attacker may know the locations of all users). *NN-Cloak* also faces similar problems. Kalnis et al. [9] identified reciprocity as a sufficient property for secure algorithms. Formally:

Definition [Reciprocity]. *Let U be any user who is issuing a query with anonymity degree k, a set of users AS called anonymizing set, and a region ASR which encloses the users in AS. AS satisfies the reciprocity property if (i) it contains U and at least k − 1 additional users, and (ii) every user in AS also generates the same anonymizing set AS for the given k.*

The only existing reciprocal (i.e., secure) algorithm is *Hilbert Cloak* [9], which has also been implemented on a peer-to-peer system for distributed anonymization [6]. *Hilbert Cloak* sorts the users according to their Hilbert value. The Hilbert space filling curve [13] transforms the multi-dimensional coordinates of each user U into an 1-D value $H(U)$. Figure 5 illustrates the Hilbert curve for a 2-D space using a 8 × 8 space partitioning. A user U is assigned the value $H(U)$ of the cell that covers him. If two users are near each other in the 2-D space, they are likely to be close in the 1-D transformation. Given a query with anonymization degree k, *Hilbert Cloak* assigns the first k users (in the Hilbert order) to the first bucket, the next k users to the second bucket and so on. Following this approach, each bucket contains exactly k users, except for the last one that may include up to $2k - 1$. Let $r(U)$ be the rank of U in the Hilbert order. The bucket b_U of U contains all clients whose ranks are in the range $[s,e]$, where $s = r(U) - (r(U) - 1) \bmod k$ and $e = s + k - 1$ (unless b_U is the last bucket). The example of Fig. 5 contains 10 users, whose user-ids are ordered according to their Hilbert value. Consider a query from U_7 with $k = 5$. The rank of U_7 is $r(U_7) = 7$. The bucket containing U_7 starts at $s = 7 - 6 \bmod 5 = 6$ and ends at $e = 10$, i.e., it contains all users from U_6 to U_{10}. Its ASR is the MBR (shaded rectangle at the upper-right corner) covering the corresponding users. Any query with $k = 5$ originating from these users will generate the same b_U and ASR, thus guaranteeing reciprocity. It must be noted that *Hilbert Cloak* constructs on-the-fly only b_U, as the remaining buckets are irrelevant to the query. Figure 5 illustrates another ASR (shaded rectangle at the lower-left corner) for a query with $k = 3$ originating from one of U_1 to U_3.

Key Applications

In recent years, positioning devices (e.g., GPS) have gained tremendous popularity. Navigation systems are already widespread in the automobile industry and, together with wireless communications, facilitate exciting new applications. General Motors OnStar system, for example, supports on-line rerouting to avoid traffic jams



Spatial Anonymity. Figure 5. Example of *Hilbert Cloak*.

and automatically alerts the authorities in case of an accident. More applications based on the user locations are expected to emerge with the arrival of the latest gadgets (e.g., iPAQ hw6515, Mio A701), which combine the functionality of a mobile phone, PDA and GPS receiver. For such applications to succeed, the privacy and confidentiality issues are of paramount importance.

Future Directions

Existing methods focus on snapshot queries. An interesting problem concerns continuous SKA [3]. In this setting, a client poses a long running query about its surroundings (e.g., "find the nearest gas station"), whose results are updated as the client moves. The cloaking algorithm should generate a continuously changing ASR in a way that does not reveal information about the user through inspection of the individual ASR snapshots. Another future direction involves the elimination of the anonymizer layer by employing private information retrieval techniques. Khoshgozaran and Shahabi [11] present an initial approach which employs 1-D transformation and encryption to conceal both the spatial data and the queries from the LBS.

Experimental Results

Compared to *Interval Cloak*, *Casper* is better both in terms of efficiency (i.e., ASR generation cost) and effectiveness (i.e., ASR size). *NN-Cloak* is worse than *Casper* in terms of efficiency, but it is considerably better in terms of effectiveness. The choice of cloaking algorithm depends on the application characteristics. If, for instance, the anonymizer charges clients according to their usage and the LBS is a public (i.e., free) service, efficiency is more important. On the other hand, if the LBS imposes limitations (e.g., on the number of results, processing time, etc) effectiveness becomes the decisive factor. Finally, *Hilbert Cloak* is very similar to *Casper* both in terms of efficiency and effectiveness. However, *Hilbert Cloak* is the only provably secure method.

Cross-references

- ▶ Anonymity
- ▶ Location-Based Services
- ▶ Nearest Neighbor Query
- ▶ Privacy
- ▶ Quadtrees (and Family)
- ▶ Space-Filling Curve

Recommended Reading

1. Beresford A.R. and Stajano F. Location privacy in pervasive computing. IEEE Pervasive Comput., 2(1):46–55, 2003.
2. Cheng R., Zhang Y., Bertino E., and Prabhakar S. Preserving user location privacy in mobile data management infrastructures. In Proceedings of Privacy Enhancing Technologies, 2006, pp. 393–412.
3. Chow C.-Y. and Mokbel M.F. Enabling private continuous queries for revealed user locations. In Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases, 2007, pp. 258–275.
4. Chow C.-Y., Mokbel M.F., and Liu X. A peer-to-peer spatial cloaking algorithm for anonymous location-based services. In Proc. 14th ACM Int. Symp. on Geographic Inf. Syst., 2006, pp. 171–178.
5. Gedik B. and Liu L. Location privacy in mobile systems: a personalized anonymization model. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2005, pp. 620–629.
6. Ghinita G., Kalnis P., and Skiadopoulos S. PRIVE: anonymous location-based queries in distributed mobile systems. In Proc. 16th Int. World Wide Web Conference, 2007, pp. 371–380.
7. Gruteser M. and Grunwald D. Anonymous usage of location-based services through spatial and temporal cloaking. In Proc. 1st Int. Conf. Mobile Systems, Applications and Services, 2003, pp. 31–42.
8. Hoh B. and Gruteser M. Protecting location privacy through path confusion. In Proc. 1st Int. Conf. on Security and Privacy for Emerging Areas in Communication Networks, 2005.
9. Kalnis P., Ghinita G., Mouratidis K., and Papadias D. Preventing location-based identity inference in anonymous spatial queries. IEEE Trans. Knowledge and Data Eng., 19(12): 1719–1733, 2007.
10. Kamat P., Zhang Y., Trappe W., and Ozturk C. Enhancing source-location privacy in sensor network routing In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2005, pp. 599–608.
11. Khoshgozaran A. and Shahabi C. Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases, 2007, pp. 239–257.
12. Mokbel M.F., Chow C. Y., and Aref W.G. The new Casper: query processing for location services without compromising privacy. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 763–774.
13. Moon B., Jagadish H.V., and Faloutsos C. Analysis of the clustering properties of the hilbert space-filling curve. IEEE Trans. Knowledge and Data Eng., 13(1):124–141, 2001.
14. Samet, H. The Design and Analysis of Spatial Data Structures. Addison-Wesley, New York, 1990.
15. Sweeney L. k-Anonymity: a model for protecting privacy. Int. J. Uncertain., Fuzziness Knowledge-based Syst., 10(5):557–570, 2002.

Spatial Autocorrelation

▶ Spatial Data Mining

Spatial Data

▶ Query Evaluation Techniques for Multidimensional Data

Spatial Data Analysis

Michael F. Goodchild
University of California-Santa Barbara, Santa Barbara, CA, USA

Synonyms

Spatial analysis; Geographical data analysis; Geographical analysis

Definition

Methods of data analysis perform logical or mathematical manipulations on data in order to test hypotheses, expose anomalies or patterns, or create summaries or views that expose particular traits. Data often refer to specific locations in some space. To qualify as spatial, the locations must be known and must affect the outcome of the analysis. While many spaces might be relevant, including the space of the human brain or the space of the human genome, the history of spatial data analysis is dominated by location in geographic space, in other words location on or near the surface of the Earth. Thus, geographical and spatial are often essentially synonymous. More formally, spatial data analysis can be defined as a set of techniques devised for the manipulation of data whose outcomes are not invariant under relocation of the objects of interest in some space. The term *exploratory spatial data analysis* (ESDA) describes an important subset that emphasizes real-time interaction, the creation of multiple views of data, the search for patterns and anomalies, and the generation of new hypotheses as opposed to the formal testing of existing ideas. The term *spatial data mining* describes another important subset that emphasizes the analysis of very large volumes of spatial data.

Historical Background

Berry and Marble [2] made one of the earliest efforts to assemble a systematic review of methods of spatial data analysis, drawing on a literature that had accumulated for many decades. Their interest was sparked in large part by what later became known as the Quantitative Revolution in Geography, a paradigm shift that originated at the University of Washington in the late 1950s and spread rapidly as the original group of graduate students found faculty positions. Bunge [3] summarized the core concept: that the analysis of patterns of phenomena on the Earth's surface could lead to a set of formal theories about the behavior of human and natural systems, and that the discovery of such theories would put the discipline of geography on a sound scientific footing. Substantial progress was made in the 1960s, particularly in the study of patterns of settlement and economic activity, and in the study of such physical phenomena as meandering rivers and stream channel networks.

Beginning in the 1960s, the development of geographic information systems (GIS) provided a major impetus, by creating a simple structure in which methods of spatial data analysis could be implemented. By the 1980s, GIS had become a popular and rapidly growing software application, with a flourishing industry and tools to enable spatial data analysis, along with the necessary techniques for data acquisition, editing, and display. Today, GIS is often portrayed as an engine for spatial data analysis, and many new techniques have been added to what are now literally thousands of methods. GIS finds application in virtually all disciplines that deal with the surface and near-surface of the Earth, ranging from ecology and geology to sociology and political science [7]. It is extensively used in logistics, in planning and public decision making, in military and intelligence applications, and in the management of utility networks.

While the use of computers to perform spatial data analysis was already well established in the 1960s, ESDA emerged rather later, when the graphics and interactive capabilities of computers had advanced sufficiently. By the early 1990s, researchers were developing novel ways of linking multiple views using the windowing techniques that emerged at that time, and exploiting the high-resolution graphics that became available on standard personal computers. Today, interactive tools inspired by ESDA are widely available in GIS products, and more specialized software is also available (see, for example, GeoDa, http://geoda.uiuc.edu).

Interest in spatial data mining has grown in the past decade, driven in part by the increasing availability of very large volumes of spatial data. For example, it is now routine to capture the location and time of use of credit and debit cards, and to apply sophisticated algorithms in an effort to detect fraudulent use. Heavy use of spatial data analysis is made by intelligence agencies, based on software that can examine telephone and email traffic and detect references to places.

Foundations

Several approaches have been devised for organizing the thousands of techniques that qualify as spatial data analysis. Perhaps the commonest, represented by several recent textbooks and by the organization of some GIS user interfaces, is based on a taxonomy of spatial data types. Very broadly, one can capture variation within a space using either raster or vector structures; a raster structure is created by dividing the space into discrete, regularly shaped elements and describing the contents of each, while vector structures describe each feature present in the space as either a point, line, area, or volume, with associated attributes.

Tomlin [9] and others have systematized the analysis of raster data in schemata described as map algebras, image algebras, or cartographic modeling, and several GIS have adopted these schemata in their user interfaces. In one such schema the analysis of raster data are described as either focal, local, zonal, or global: focal operations are performed independently on the contents of each cell; local operations are performed on a cell and its immediate neighborhood; zonal operations apply to contiguous patches of cells with identical descriptions; and global operations apply to the entire raster.

To date a similarly simple systematization of vector operations has not been achieved. Instead, several textbooks (e.g., [1,5]) organize methods of vector-based analysis according to the types of features being analyzed, focusing in turn on points, lines, areas, and volumes. For example, techniques for the analysis of sets of points might determine the degree of dispersion of the points; search for anomalous clusters; or find a shortest tour through the points. Some texts also provide descriptions of methods for the analysis of relationships or interactions between features. For example, retailers and traffic engineers commonly use methods of spatial data analysis to predict the numbers of trips expected between home neighborhood areas and such destination points as shopping centers or places of work.

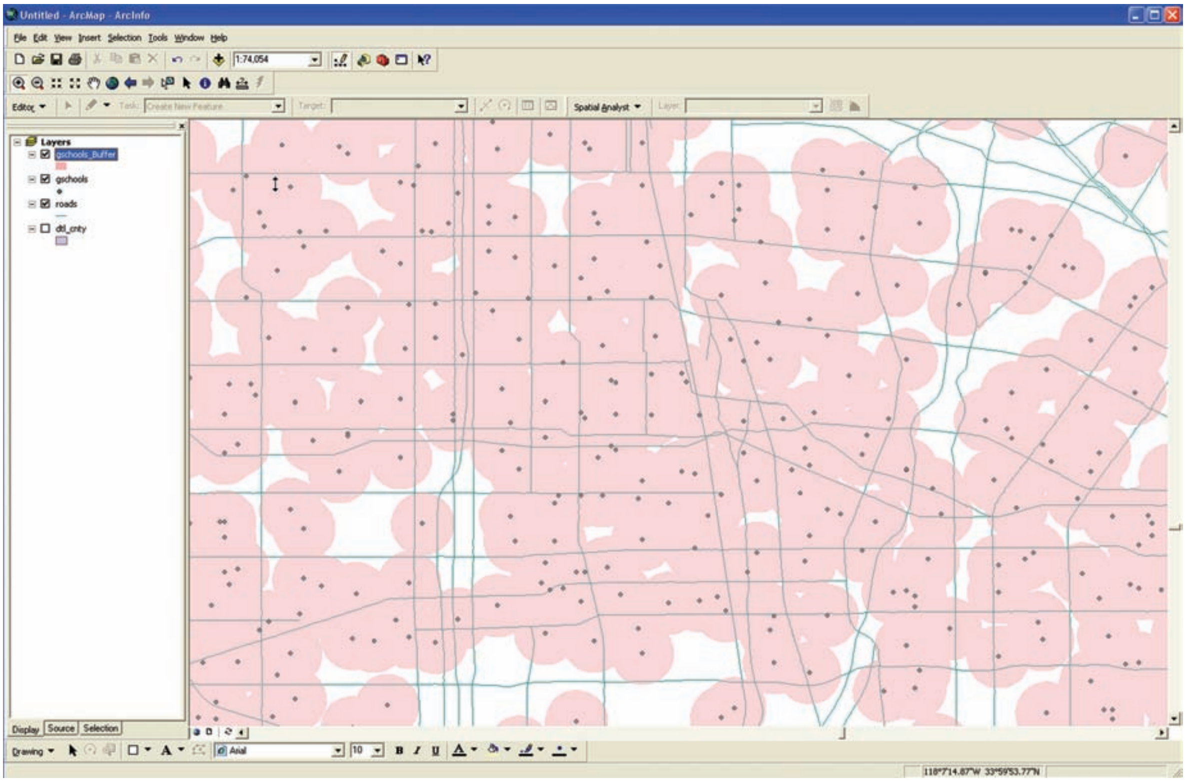
Longley et al. [7] use a different organizing scheme that is designed to be more strongly related to user motivation, and to overcome some of the ambiguities inherent in an emphasis on data type. Their scheme assigns techniques to six categories, ordered by increasing conceptual sophistication: query and reasoning, measurement, transformation, descriptive summary, optimization, and hypothesis testing.

Query and reasoning functions rely on the presentation of alternative views to the user. For example, a set of data on average income by US state might be presented as a map, as a table, as a histogram, and as a scatterplot in which average income is graphed against another variable such as percent with more than high-school education. The user gains insight by examining the alternative views, by querying specific values, or by selecting data items in one window and seeing them highlighted in the other windows.

Measurement functions represent one of the earliest motivations for GIS. Manual methods for obtaining measures of such properties as area, length, slope, or shape from maps are notoriously inaccurate, tedious, and time-consuming, whereas it is trivial to obtain them from digital representations. Nevertheless, digital representations are only approximations or generalizations of real phenomena, and many estimates exhibit representation-related biases.

Transformation functions obtain new objects, or new properties of those objects. They include many key GIS functions, including buffering (the geometric dilation of points, lines, areas, or volumes), overlay (the computation of intersections between objects), and interpolation (the use of data from sample locations to estimate values at locations where no samples were taken). Figure 1 shows an example of buffering, using half-mile circles around points representing the schools of part of Los Angeles. The example was motivated by proposals to ban registered sex offenders from living within a specified distance of a school.

Descriptive summaries include the widest range of spatial data analysis techniques. Standard univariate statistics such as the mean, median, mode, standard deviation, and variance have equivalents in multidimensional spaces. Figure 2 shows the two-dimensional equivalents of the mean and standard deviation

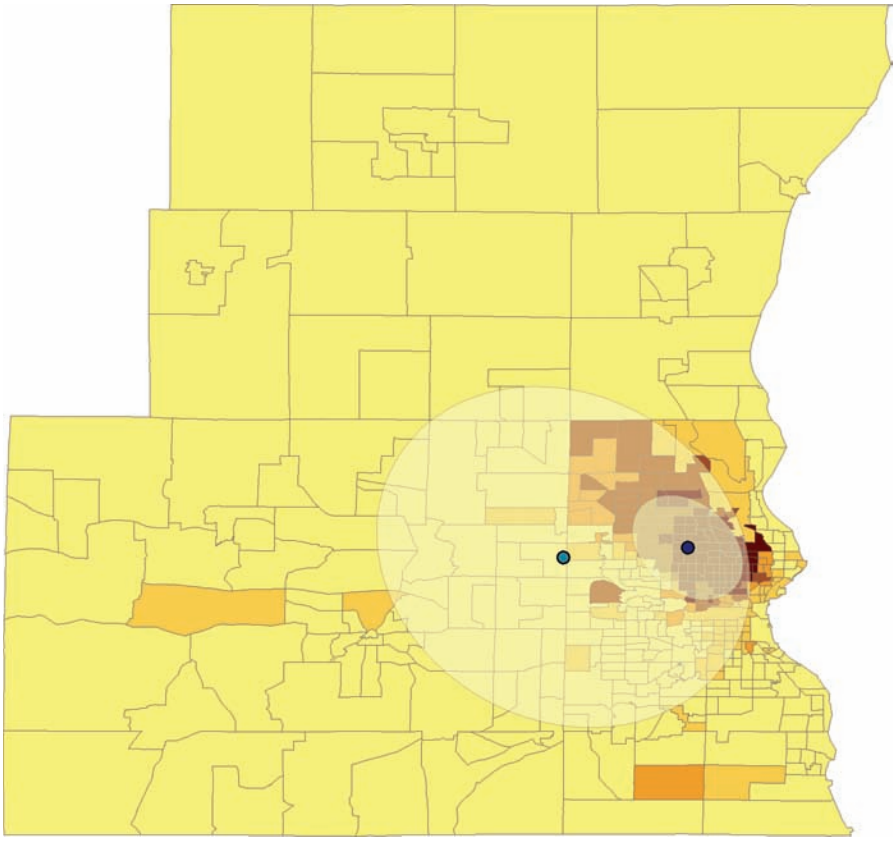


Spatial Data Analysis. Figure 1. The buffer operation. Half-mile buffers have been drawn around points representing the locations of schools in an area of central Los Angeles. Such buffers are often required by legislation; this example was motivated by a proposal to ban registered sex offenders from living within a prescribed distance of schools.

applied to the black and white populations of Milwaukee, using data by census tract. A suite of summary statistics have been devised for measuring *spatial dependence*, a key property of many spaces based on the observation that measurements of many properties taken close together tend to be more similar than measurements taken far apart. The fields of *spatial statistics* and *geostatistics* are both based on this property, and provide ways of addressing it explicitly. *Spatial heterogeneity*, or the tendency for the properties of spaces to vary widely from one area to another, is also the subject of many forms of descriptive summary. The rapidly evolving field of *local* or *place-based* summaries (e.g., [4]) addresses the spatial heterogeneity property directly, arguing that it is more important to determine how the results of spatial data analysis vary from one area to another than to attempt to extract single, global results. Another suite of summary statistics addresses the *fragmentation* of landscapes, with particularly strong applications in ecology.

Methods of *optimization* focus on the design of spatial pattern rather than on its analysis. They include methods for optimum location of points (e.g., retail stores, schools), lines (e.g., roads, pipelines), and areas (e.g., political voting districts), as well as on the design of optimum routes (e.g., for delivery vehicles or school buses).

Finally, methods of *hypothesis testing* address the process of inference, by which analysts reason from the analysis of a sample to conclusions about the larger world represented by the sample. Such methods are well known in statistical analysis, encompassing many well-known statistical tests, significance levels, and the formulation of null hypotheses. Unfortunately the application of such methods to spatial data is confounded by two major issues. First, it is rare for a sample of objects to be representative of any larger and well-defined set; instead, spatial data analysis is commonly applied to all of the objects that exist in a given study area. Second, it is also rare for objects distributed in space to be selected



Spatial Data Analysis. Figure 2. Two-dimensional equivalents of the mean and standard deviation. The larger ellipse shows the dispersion of the white population of Milwaukee around its centroid; the smaller ellipse shows the greater concentration of the city's black population. The map shows percent black, using 1990 data by census tract.

independently from any larger set; instead, the property of spatial dependence virtually ensures that nearby objects will have some degree of similarity.

Key Applications

As noted earlier, techniques of spatial data analysis can be applied to virtually all spaces, and all phenomena distributed within such spaces. Nevertheless, the vast majority of applications are found in geographic space, that is, the space defined by the surface and near-surface of the Earth, at spatial resolutions ranging from sub-meter to global.

Many important applications have derived from the need to understand the mechanisms of disease, and particularly its transmission within human populations. The work of Dr. John Snow on cholera [8] is often cited as the seminal example, but today methods of spatial data analysis are routinely used to scan data on such diseases as cancer, searching for anomalous clusters and thus for potential causal mechanisms. Spatial data analysis has been central to the study of outbreaks of new diseases such as West Nile virus and SARS.

Spatial data analysis has also been central to the study of landscape change, and related phenomena of urban sprawl, deforestation, desertification, and habitat destruction. Such analyses are often based on snapshots of landscape obtained from Earth-orbiting satellites, and can form the basis for sophisticated models of landscape change that can be used to investigate the future effects of management alternatives.

Transportation applications are also particularly rich. Methods of spatial data analysis are routinely used to model traffic patterns, and to evaluate planning options, including new roads, mass transit, and congestion pricing. The possibility of real-time tracking of vehicles using GPS has recently given this field new impetus.

Future Directions

The insights that can be obtained from spatial data analysis are limited by its essentially cross-sectional nature – the need to draw inferences from snapshots obtained at one point in time. It is difficult, for example, to ascribe cause when no information is available about change through time. Thus there is great interest in the development of an improved suite of methods for *spatiotemporal* data analysis. In the past, the lack of suitable data has been a major impediment, but today vast new sources are becoming available as the result of developments in satellite remote sensing, GPS tracking, and Internet-based data sharing.

GIS owes much of its original stimulus to the paper map, which is of necessity flat. At global scales, analysis based on flattened or *projected* views of the Earth's surface can be misleading, and there is therefore strong interest in developing methods of spatial data analysis for the Earth's curved surface. This interest has been stimulated in part by the recent emergence of *virtual globes*, including Google Earth.

Cross-references

- ▶ Geographic Information System
- ▶ Spatial Data Mining
- ▶ Spatial Data Types
- ▶ Spatial Operations and Map Operations

Recommended Reading

1. Bailey T.C. and Gatrell A.C. Interactive Spatial Data Analysis. Longman, 1995.
2. Berry B.J.L. and Marble D.F. Spatial Analysis: A Reader in Statistical Geography. Prentice Hall, Englewood Cliffs, NJ, 1968.
3. Bunge W. Theoretical Geography. University of Lund. Gleerup, Sweden, 1966.
4. Fotheringham A.S., Brunsdon C., and Charlton M. Geographically Weighted Regression: The Analysis of Spatially Varying Relationships. Wiley, 2002.
5. Haining R.P. Spatial Data Analysis: Theory and Practice. Cambridge University Press, UK, 2003.
6. Johnson S. The Ghost Map: The Story of London's Most Terrifying Epidemic and How It Changed Science, Cities, and the Modern World. Riverhead, 2006.
7. Longley P.A., Goodchild M.F., Maguire D.J., and Rhind D.W. Geographic Information Systems and Science. Wiley, New York, 2005.
8. O'Sullivan D., Unwin D.J. Geographic Information Analysis. Wiley, 2003.
9. Tomlin C.D. Geographic Information Systems and Cartographic Modeling. Prentice Hall, Englewood Cliffs, NJ, 1990.

Spatial Data Mining

Shashi Shekhar, James Kang, Vijay Gandhi
University of Minnesota, Minneapolis, MN, USA

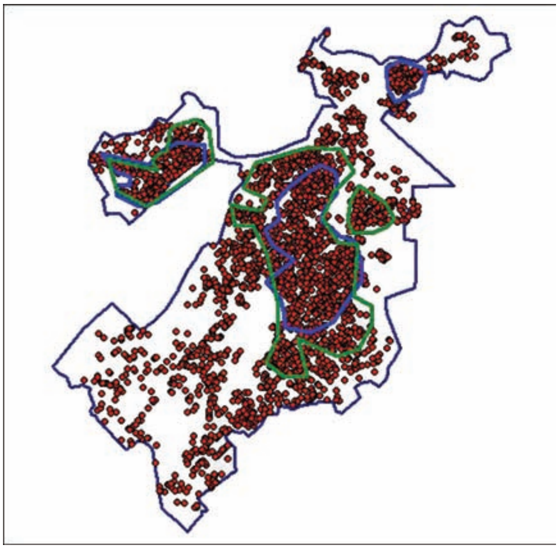
Synonyms

Spatial data analysis; Spatial statistics; Co-locations; Spatial outliers; Hotspots; Location prediction; Spatial autocorrelation

Definition

Spatial data mining is the process of discovering non-trivial, interesting, and useful patterns in large spatial datasets. The most common spatial pattern families are co-locations, spatial hotspots, spatial outliers, and location predictions.

Figure 1 gives an example of a spatial hotspot pattern for burglary related crimes in the Boston, MA area. In this figure, each point depicts a burglary event in the year 1999. The dark blue and green shapes in the figure represent the discovered hotspots or the source of this type of crime. Notice that discovering these hotspots is a non-trivial process due to the irregular size and spatial shape of the pattern. In addition, not all incidents contribute to the hotspot. Discovery of these patterns is very useful and interesting to public safety professionals



Spatial Data Mining. Figure 1. Spatial hotspots of crimes in Boston, MA (best viewed in color) (courtesy: NIJ, Infotech).

as they plan police patrols and social interventions to reduce future crime incidents in the area.

Historical Background

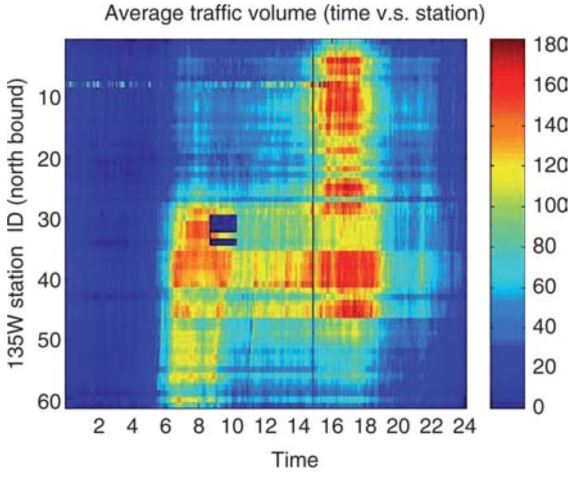
Spatial data mining research began several decades ago when practitioners and researchers noticed that critical assumptions in classical data mining and statistics were violated by spatial datasets. First, whereas classical datasets often assume that data are discrete, spatial data were observed to reside in continuous space. For example, classical data mining and statistical methods may use market-basket datasets (e.g., history of Walmart's transactions), where each item-type in a transaction is discrete. However, "transactions" are not natural in continuous spatial datasets, and decomposing space across transactions leads to loss of information about neighbor relationships between items across transaction boundaries. In addition, spatial data often exhibits heterogeneity (i.e., no places on the Earth are identical), whereas classical data mining techniques often focus on spatially stationary global patterns (i.e., ignoring spatial variations across locations). Finally, one of the common assumptions in classical statistical analysis is that data samples are independently generated. However, this assumption is generally false when analyzing spatial data, because spatial data tends to be highly self-correlated. For example, people with similar characteristics, occupation and background tend to cluster together in the same neighborhoods. In spatial statistics [1] this tendency is called spatial auto-correlation. Ignoring spatial auto-correlation when analyzing data with spatial characteristics may produce hypotheses or models that are inaccurate or inconsistent with the data set. Thus, classical data mining algorithms often perform poorly when applied to spatial data sets. Better methods are needed to analyze spatial data to detect spatial patterns.

Foundations

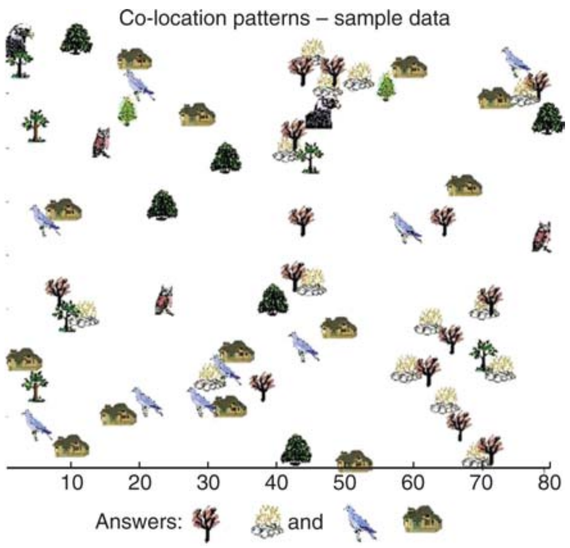
The spatial data mining literature has focused on four main types of spatial patterns: (i) spatial outliers, which are spatial locations showing a significant difference from their neighbors; (ii) spatial co-locations, or subsets of event types that tend to be found more often together throughout space than other subsets of event types; (iii) location predictions, that is, information that is inferred about locations favored by an event type based on other explanatory spatial variables; and (iv) spatial hotspots, unusual spatial groupings of events. The remainder of this section presents a general overview of each of these pattern categories.

A *spatial outlier* is a spatially referenced object whose non-spatial attribute values differ significantly from those of other spatially referenced objects in its spatial neighborhood. Figure 2 gives an example of spatial outliers, detected in traffic measurements for sensors on highway I-35W (North bound) in the Minneapolis-St. Paul area, for a 24-h time period. Station 9 may be considered a spatial outlier as it exhibits inconsistent traffic flow compared with its neighboring stations. Once a spatial outlier is identified, one may proceed with diagnosis. For example, the sensor at Station 9 may be diagnosed as malfunctioning. Spatial attributes are used to characterize location, neighborhood, and distance. Non-spatial attribute dimensions are used to compare a spatially referenced object to its neighbors. Spatial statistics literature provides two kinds of bi-partite multidimensional tests, namely graphical tests and quantitative tests. Graphical tests, such as Variogram clouds and Moran scatterplots, are based on the visualization of spatial data and highlights spatial outliers. Quantitative methods provide a precise test to distinguish spatial outliers from the remainder of data.

Spatial co-location pattern discovery finds frequently co-located subsets of spatial event types given a map of their locations. Figure 3 gives an example map with two examples of spatial co-locations. Readers are encouraged to determine for themselves the co-located pairs of spatial event types in Fig. 3. The answers provided



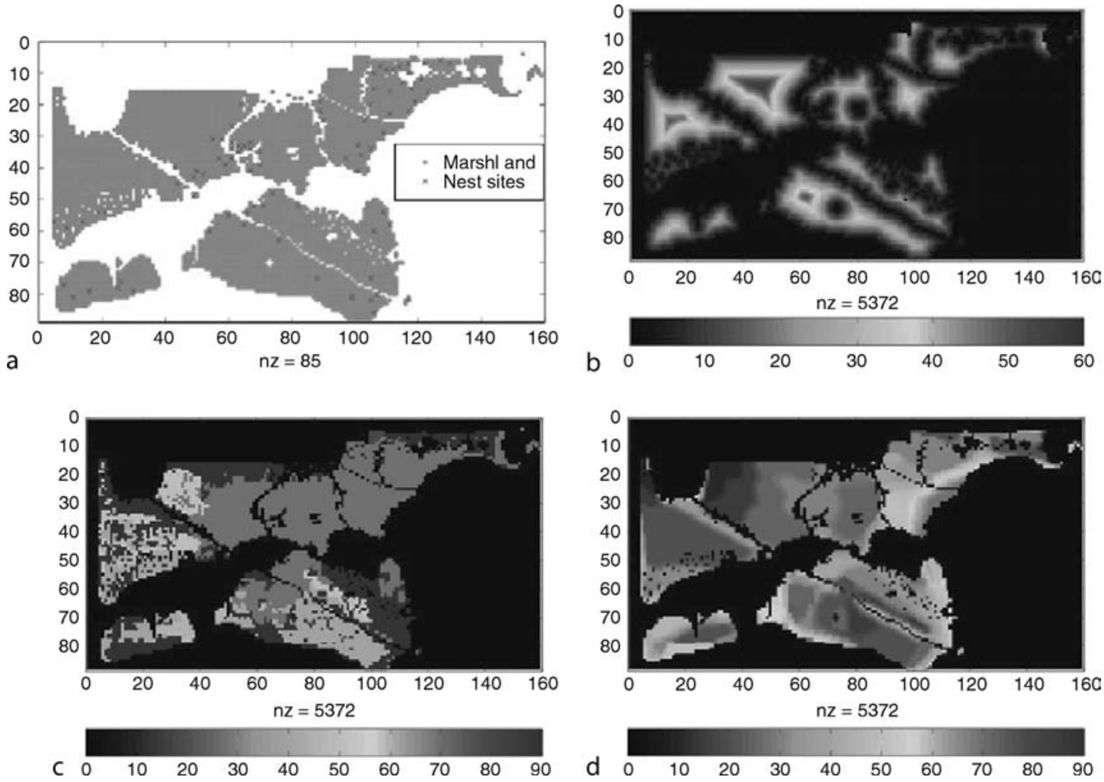
Spatial Data Mining. Figure 2. Spatial outlier (station ID 9) in traffic volume data (best viewed in color).



Spatial Data Mining. Figure 3. An example of spatial co-location patterns.

there show that *trees* and *fire* tend to co-occur together across the spatial region, as well as the pattern *bird* and *house*. Spatial co-location is a generalization of a classical data mining pattern-family called association rules, since transactions are not natural in spatial datasets, and partitioning space across transactions leads to loss of information about neighbor relationships between items near transaction boundaries. Additional details about co-location interest measures, e.g., participation index and K-functions, and mining algorithms are described in [2].

Location prediction is concerned with the discovery of a model to infer preferred locations of a spatial phenomenon from the maps of other explanatory spatial features. For example, ecologists may build models to predict habitats for endangered species using maps of vegetation, water bodies, climate, and other related species. Figure 4 gives an example of a dataset used in building a location prediction model for red-winged



Spatial Data Mining. Figure 4. (a) Learning dataset: The geometry of the Darr wetland and the locations of nests. (b) The spatial distribution of *distance to open water*. (c) The spatial distribution of *vegetation durability* over the marshland. (d) The spatial distribution of *water depth*.

blackbirds in the Darr and Stubble wetlands on the shores of Lake Erie in Ohio, USA. This dataset consists of nest location, distance to open water, vegetation durability and water depth maps. Classical prediction methods may be ineffective in this problem due to the presence of spatial auto-correlation. Spatial data mining techniques that capture the spatial auto-correlation of nest location such as the Spatial Autoregression Model (SAR) [1] and Markov Random Fields based Bayesian Classifiers (MRF-BC) are used for location prediction modeling. A comparison of these methods is discussed in [6].

Spatial Hotspots are unusual spatial groupings of events that tend to be much more closely related than other events. Examples of spatial hotspots can be incidents of crime in a city or outbreaks of a disease. Hotspot patterns have properties of clustering as well as anomalies from classical data mining. However, hotspot discovery [11] remains a challenging area of research due to variation in shape, size, density of hotspots and underlying space (e.g., Euclidean or spatial networks such as roadmaps). Additional challenges arise from the spatio-temporal semantics such as emerging hotspots, displacement etc.

Key Applications

Spatial data mining and the discovery of spatial patterns has applications in a number of areas. Detecting spatial outliers is useful in many applications of geographic information systems and spatial databases, including the domains of public safety, public health, climatology, and location-based services. As noted earlier, for example, spatial outlier applications may be used to identify defective or out of the ordinary (i.e., unusually behaving) sensors in a transportation system (e.g., Fig. 1). Spatial co-location discovery is useful in ecology in the analysis of animal and plant habitats to identify co-locations of predator-prey species, symbiotic species, or fire events with fuel and ignition sources. Location prediction may provide applications toward predicting the climatic effects of *El Nino* on locations around the world. Finally, identification of spatial hotspots can be used in crime prevention and reduction, as well as in epidemiological tracking of disease.

Cross-references

- ▶ Data Mining
- ▶ Geographic Information System
- ▶ Spatial Network Databases
- ▶ Spatio-temporal Databases
- ▶ Spatio-temporal Data Mining

Recommended Reading

1. Cressie N.A. Statistics for Spatial Data (Revised Edition). Wiley, New York, NY, 1993.
2. Huang Y., Shekhar S., and Xiong H. Discovering co-location patterns from spatial datasets: a general approach. IEEE Trans. Knowl. Data Eng., 16(12):1472–1485, 2004.
3. Kou Y., Lu C.T., and Chen D. Algorithms for spatial outlier detection. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 597–600.
4. Longley P.A., Goodchild M., Maquire D.J., and Rhind D.W. Geographic Information Systems and Science. Wiley, 2005.
5. Mamoulis N., Cao H., and Cheung D.W. Mining frequent spatio-temporal sequential patterns. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2005, pp. 82–89.
6. Shekhar S., Schrater P., Vatsavai R., Wu W., and Chawla S. Spatial contextual classification and prediction models for mining geospatial data. IEEE Trans. Multimed. (special issue on Multimedia Databases), 4(2):174–188, 2002.
7. Shekhar S. and Chawla S. A Tour of Spatial Databases. Prentice Hall, 2003.
8. Shekhar S., Lu C.T., and Zhang P. A unified approach to detecting spatial outliers. GeoInformatica, 7(2):139–166, 2003.
9. Shekhar S., Zhang P., Huang Y., and Vatsavai R. Trend in spatial data mining. In Data Mining: Next Generation Challenges and Future Directions, H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.). AAAI/MIT Press, 2003.
10. Solberg A.H., Taxt T., and Jain A.K. A Markov random field model for classification of multisource satellite imagery. IEEE Trans. Geosci. Remote Sens., 34(1):100–113, 1996.
11. US Department of Justice - Mapping and Analysis for Public Safety report. Mapping Crime: Understanding Hot Spots, 2005 (http://www.ncjrs.gov/pdffiles1/nij/209393.pdf).

Spatial Data Types

Markus Schneider
University of Florida, Gainesville, FL, USA

Synonyms

Geometric data types

Definition

Data types are a well known concept in computer science (for example, in programming languages or in database systems). A *data type* defines a set of homogeneous values and the allowable operations on those values. An example is a type *integer* representing the set of 32-bit integers and including operations such as addition, subtraction, and multiplication that can be performed on integers. *Spatial data types* or *geometric data types* provide a fundamental abstraction for modeling the geometric structure of objects in space as well as their relationships, properties, and operations. They are of particular interest in *spatial databases* [4,8,12] and *Geographical Information Systems* [4]. One speaks of *spatial objects* as values of spatial data types. Examples are two-dimensional data types for *points* (for example, representing the locations of lighthouses in the U.S.), *lines* (for example, describing the ramifications of the Nile Delta), *regions* (for example, depicting air-polluted zones), *spatial networks* (for example, representing the routes of the Metro in New York), and *spatial partitions* (for example, describing the 50 states of the U.S. and their exclusively given topological relationships of adjacency or disjointedness) as well as three-dimensional data types for *surfaces* (for example, modeling the shape of landscapes) or *volumes* (for example, representing urban areas). Operations on spatial data types include *spatial operations* like the geometric *intersection*, *union*, and *difference* of spatial objects, *numerical operations* like the *length* of a line or the *area* of a region, *topological relationships* checking the relative position of spatial objects to each other like *overlap*, *meet*, *disjoint*, or *inside*, and *cardinal direction relationships* like *north* or *southeast*.

Historical Background

In the late 1970s, the interest to store geometric data into databases arose. The success and efficiency of relational database technology for standard applications, which is rooted in its simple data model, its high-level query languages, and its well understood underlying theory, has led to many proposals to transfer this technology directly to geometric applications and to explicitly model the structure of spatial data as relations (tables). The consequence is that the user conceives spatial data in tabular form, just the same as standard data, and that a spatial object is represented by several or even many tuples. An example of such a relation schema is *RelName*(*id* : *integer*, x_1 : *integer*, y_1 : *integer*, x_2 : *integer*, y_2 : *integer*, *type* : *string*, <other information>) where x_1, y_1, x_2, and y_2 are the coordinates of a point or a line segment. The flag *type* indicates whether a tuple describes a point, a single line, a line segment of a line, or a line segment of a polygon. The value *id* denotes the object identifier.

This approach has revealed a number of fundamental drawbacks. Since all lines and polygons are decomposed into a set of line segments (tuples) scattered over a relation, a spatial object is not treated as an entity or unit but only *corresponds* to several tuples. This is different compared to values of standard data types. A second drawback is that the approach forces the user to model complex spatial objects in flat, independent relations. Since the representations of spatial data occurs on a very low level and is exclusively based on standard domains like integers, strings, and reals (while the user has originally intended to deal with points, lines, or polygons (regions)), an adequate treatment of spatial data is impeded. Although the facilities of the query language of a DBMS are available, they are only of limited use. Since such a language is based on standard domains and has no concept of spatial data types, it cannot provide and support any meaningful geometric operations. A more detailed discussion can be found in [9].

Foundations

The numerous deficiencies of the approach of modeling spatial data as relations, have resulted in the assessment that this approach is unsuitable to manage spatial data in a clean and efficient manner, and that a high-level view of spatial objects is essential. This has led to the design of spatial data types that are represented as *abstract data types*, thus provide such a high-level view, and can be used as attribute data types in a database schema in the same way as standard data types like *integer, float*, or *string*. That is, the internal structure of a spatial object is hidden from the user, and its features can only be retrieved by (abstract) operations on this object.

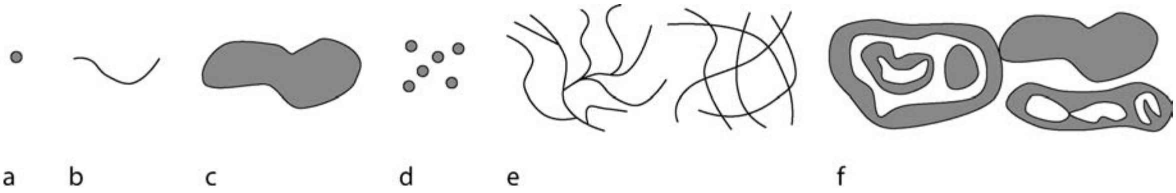
One can distinguish different kinds of spatial data types. *Universal spatial data types* either only provide a single generic spatial data type called *spatial*, and therefore do not consider the dimensionality and shape of spatial objects, or they provide the types *spatial_0*, *spatial_1*, *spatial_2*, and *spatial_3* and thus, consider the dimensionality but not the shape of spatial objects [6]. Another conceptual model for spatial data types is based on mathematical abstractions called *point sets*. The user is supplied with the concept that each spatial object consists of an infinite set of points that can be described by finite means. The approach in [7] introduces a type POINT-SET for point sets together with a collection of geometric operations. Aspects like dimensionality and shape of an object are not considered. A further approach of modeling spatial objects is that of using *half planes* [13], where each half plane is defined by a *half plane segment*. A half plane segment uniquely determines a straight line which is given by an inequality, passes this segment and forms the one-sided boundary of a half plane. For constructing a polygonal region, an appropriately arranged sequence of intersection operations (conjunction of inequalities) defined on half planes is employed. This concept is the precursor of so-called *constraint spatial databases*. Most popular and fundamental abstractions of spatial objects fall into the category of *structure-based spatial data types*. These data types organize space into points, lines, regions, surfaces, volumes, spatial partitions, spatial networks, and similarly structured entities. Thus, this approach considers the structural shape and spatial extent of spatial objects, that is, their geometry. Spatial data types for points, lines, and regions have, for example, been considered in [1,5,6,9,11,14], for surfaces and volumes in [10], for spatial partitions in [3], and for spatial networks in [12].

Structure-based spatial data types have prevailed and form the basis of a large number of data models and query languages for spatial data. They have also found broad acceptance in spatial extension packages of commercially and publicly available database systems, as well as in Geographical Information Systems. One can distinguish the older generation of *simple* spatial data types and the newer generation of *complex* spatial data types, depending on the spatial complexity the types are able to model. In the two-dimensional space, simple spatial data types only provide simple object structures like single points, continuous lines, and simple regions (Fig. 1a–c). However, from an application perspective, simple spatial data types have turned out to be inadequate abstractions for spatial applications, since they are insufficient to cope with the variety and complexity of geographic reality. From a formal perspective, they are not closed under the geometric set operations intersection, union, and difference. This means that these operations applied to two simple spatial objects can produce a spatial object that is not simple. Complex spatial data types solve these problems. They provide universal and versatile spatial objects and are closed under geometric set operations. They allow objects with multiple components, region components that may have holes, and line components that may model ramified, connected geometric networks (Fig. 1d–f).

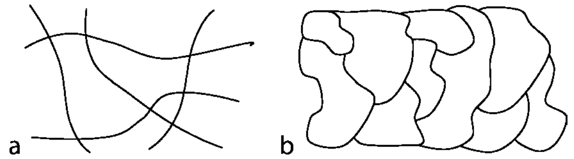
Even more complex structure-based spatial data types are spatial networks and spatial partitions. They are the essential components of maps. A *spatial network* (Fig. 2a) can be viewed as a spatially embedded graph which consists of a set of point objects representing its nodes and a set of line objects describing the geometry of its edges. Examples are highways, rivers, public transport systems, power lines, and phone lines. A *spatial partition* (Fig. 2b) is a set of region objects together with the topological constraint that any two regions either meet or are disjoint. The neighborhood relationship is of particular interest here since region objects may share common boundaries. Examples are states, school districts, crop fields, and land parcels. Both in spatial networks and in spatial partitions, their components (line objects, region objects) are annotated with thematic data like state name, unemployment rate, and parcel id.

Spatial operations manipulate spatial objects. They take spatial objects as operands and return either spatial objects or scalar values (like Boolean or numerical values) as results. One can classify them into the following categories:

Spatial predicates returning Boolean values. A spatial relationship is a relationship between two or more spatial objects. A *spatial predicate* compares two spatial objects with respect to some spatial relationship and



Spatial Data Types. Figure 1. Examples of a simple point object (a), a simple line object (b), a simple region object (c), a complex point object (d), a complex line object (e), and a complex region object (f).



Spatial Data Types. Figure 2. Examples of a spatial network (a) and a spatial partition (b).

thus conforms to a binary relationship returning a Boolean value. Spatial predicates can be classified into three subcategories. *Topological predicates* characterize the relative position of spatial objects towards each other and are preserved under topological transformations such as translation, rotation, and scaling; they do not depend on metric concepts like distance. Examples are the well known predicates *equal, disjoint, coveredBy, covers, overlap, meet, inside*, and *contains* between two simple regions. *Metric predicates* use measurements such as distances. For example, the predicates *in_circle* and *in_window* test if a spatial object is located within the scope of a predefined circle or rectangle. *Directional predicates* like *north* or *southeast* compare the cardinal direction of a target object with respect to a reference object.

Spatial operations returning numbers. These operations compute metric properties of spatial objects and return a number. Examples are the operations *area* and *perimeter* computing the corresponding values of a region object, the operation *length* calculating the total length of a line object, the operation *diameter* determining the largest distance between any two object components, the operation dist computing the minimal distance between two spatial objects, and the *operation cardinality* yielding the number of components of a spatial object.

Spatial operations returning spatial objects. These operations return spatial objects as results and can be subdivided into object construction operations, which construct new objects from existing objects, and object transformation operations, which transform one or more spatial objects into a new spatial object. The *object construction operations* include, for example, the *geometric set operations union, intersection*, and *difference*, which satisfy closure properties, the operation *convex_hull*, which constructs the smallest convex region (polygon) enclosing a finite collection of points, the operation *boundary*, which returns the boundary of a region object as a line object, the operation *box*, which determines the minimal, axis-parallel rectangle (called *minimal bounding box* or *rectangle*) that bounds a spatial object, and the operation *components*, which extracts the vertices of a line object. Examples of *object transformation operations* are the operation *extend*, which takes a spatial object *s* and a real number *r* as operands and creates a polygonal region that is a spatial extension of *s* with distance *r* from *s* (also known as *buffer zoning*), the operation rotate, which rotates a spatial object around a point, and the operation translate, which moves a spatial object by a defined vector.

Spatial operations on spatial networks and spatial partitions. An important operation on spatial networks is the *shortest_path* operator. It computes the route or path of minimum distance between a source and a destination. An important operation on spatial partitions is the *overlay* operation. It takes two spatial partitions modeling different themes as operands, lays them transparently on top of each other, and combines them into a new spatial partition by intersection. A large collection of other operations is available for both kinds of structures.

A brief example illustrates the embedding of a spatial data type into a relation schema and the posing of a spatial query. Consider the map of the 50 states of the USA. Besides its thematic attributes like name and population, each state is also described by a geometry which is a region. Cities can be represented as points, that is, one is here interested in their location and not so much in their extent. As thematic attributes, one could be interested in their name and population. In the following two relation schemas, the spatial data types *point* and *region* are used in the same way as attribute data types as standard data types.

```
states(sname: string, spop: integer, ter-
ritory: region)
cities(cname: string, cpop: integer, loc:
point)
```

A query could ask for all pairs of city names and state names where a city is located in a state. This can then be formulated as a *spatial join*:

```
select cname, sname
from cities, states
where loc inside territory
```

The term *inside* is a topological predicate testing whether a point object is located inside a region object.

Key Applications

Spatial data types are a universal and general concept for representing geometric information in all kinds of spatial applications. Hence, they are not only applicable to a few key applications. In principle, all applications in the geosciences (for example, geography, hydrology, soil sciences) and Geographical Information Systems, as well as many applications in government and administration (for example, cadastral application, urban planning), can benefit from them. Independent studies have shown that about 80% of all data have spatial features (like geometric attributes) or a spatial reference (like an address). Thus, it is not surprising that independent international studies have predicted that geoinformation technology will belong to the most important and promising technologies in the future, besides biotechnology and nanotechnology.

Cross-references

- ▶ Cardinal Direction Relationships
- ▶ Dimension-Extended Topological Relationships
- ▶ Simplicial Complex
- ▶ Spatial Operations and Map Operations
- ▶ Three-Dimensional GIS and Geological Applications
- ▶ Topological Relationships

Recommended Reading

1. Clementini E. and Di Felice P. A model for representing topological relationships between complex geometric features in spatial databases. Inf. Syst., 90(1–4):121–136, 1996.
2. Egenhofer M.J. Spatial SQL: a query and presentation language. IEEE Trans. Knowl. Data Eng., 6(1):86–94, 1994.
3. Erwig M. and Schneider M. Partition and Conquer. In Proc. Third International Conference on Spatial Information Theory, 1997, pp. 389–408.
4. Güting R.H. An introduction to spatial database systems. VLDB J., 3(4):357–399, 1994.
5. Güting R.H. and Schneider M. Realm-based spatial data types: the rose algebra. VLDB J., 4:100–143, 1995.
6. Güting R.H. Geo-relational algebra: a model and query language for geometric database systems. In Advances in Database Technology, Proc. 1st Int. Conf. on Extending Database Technology, 1988, pp. 506–527.
7. Manola F. and Orenstein J.A. Toward a general spatial data model for an object-oriented DBMS. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 328–335.
8. Rigaux P., Scholl M., and Voisard A. Spatial Databases – With Applications to GIS. Morgan Kaufmann Publishers, 2002.
9. Schneider M. Spatial Data Types for Database Systems – Finite Resolution Geometry for Geographic Information Systems, Vol. LNCS 1288. Springer, 1997.
10. Schneider M. and Weinrich B. An abstract model of three-dimensional spatial data types. In Proc. 12th ACM Int. Symp. on Geographic Inf. Syst., 2004, pp. 67–72.
11. Schneider M. and Behr T. Topological relationships between complex spatial objects. ACM Trans. Database Syst., 31(1):39–81, 2006.
12. Shekar S. and Chawla S. Spatial Databases: A Tour. Prentice-Hall, 2003.
13. Scholl M. and Voisard A. Thematic map modeling. In Proc. 1st International Symposium on Advances in Spatial Databases, 1989, pp. 167–190.
14. Worboys M.F. and Bofakos P. A canonical model for a class of areal spatial objects. In Proc. Third International Symposium on Advances in Spatial Databases, 1993, pp. 36–52.
15. Worboys M.F. and Duckham M. GIS: A Computing Perspective. CRC, 2004.

Spatial Graph Databases

▶ Spatial Network Databases

Spatial Indexing Techniques

Yannis Manolopoulos[1], Yannis Theodoridis[2],
Vassilis J. Tsotras[3]

[1]Aristotle University of Thessaloniki, Thessaloniki, Greece

[2]University of Piraeus, Piraeus, Greece

[3]University of California-Riverside, Riverside, CA, USA

Synonyms

Spatial access methods

Definition

A Spatial Index is a data-structure designed to enable fast access to spatial data. Spatial data come in various forms, the most common being points, lines, and regions in n-dimensional space (practically, $n = 2$ or 3 in GIS Geographical Information System applications). Typical "selection" queries include the *spatial range query* ("find all objects that lie within a given query region"), and the *spatial point query* ("find all objects that contain a given query point"). In addition, multi-dimensional data introduce spatial relationships (such as overlapping and disjointness) and operators (e.g., nearest neighbor), which need to be efficiently supported as well. Example queries are the *spatial join query* ("find all pairs of objects that intersect each other") and the *nearest neighbor query* ("find the five objects nearest to a given query point"). It should be noted that traditional indexing approaches (B^+-trees, etc.) are not appropriate for indexing spatial data; the basic reason is the lack of total ordering, which is an inherent characteristic in a multi-dimensional space. As a result, specialized access methods are necessary.

Historical Background

Many applications (VLSI, CAD/CAM, GIS, multimedia) need to represent, store and manipulate spatial data types, such as points, lines, and regions in *n*-dimensional space. Although the representation of this type of data may be straightforward in a traditional database system (e.g., a 2-dimensional point may be represented as a pair of *x*- and *y*- numeric values), spatial relationships (e.g., overlapping) and operators (e.g., nearest neighbor) need to be efficiently supported as well. These spatial relationships and operators have led to a variety of interesting and more complex queries like spatial joins, nearest neighbors etc. As a result, specialized access methods have been proposed in order to quickly answer the above complex queries, as well as spatial range/point queries.

Given the characteristics of spatial data, for each spatial operator the query object's geometry needs to be combined with each data object's geometry. Nevertheless, the processing of complex geometry representations, usually polygons, is very expensive in terms of CPU cost. For that reason, the object geometries are approximated (typically by Minimum Bounding Rectangles –MBRs), and these approximations are then stored in underlying indices while the actual geometry is stored separately. As a result, a two-step procedure is involved during query processing, consisting of a *filter step* and a *refinement step*. The question that arises is how the object approximations (MBRs) are organized in order to answer the hits and the candidates, i.e., the result of the filter step.

Various spatial indices have been proposed in the literature and can be divided in two categories: indices designed for multi-dimensional points, and indices for multi-dimensional regions. Examples in the first category are the LSD tree [7], the Grid File [10], the hB-tree [9], the Buddy Tree [14] and the BV-tree [3]. The major representatives in second category are the R-tree [5] and the Quadtree [2], and their variants.

Given the complexity of the indexing problem and the different requirements of the multiple applications that index spatial data, it is not clear which the best index is. Nevertheless, R-tree implementations have found their way into commercial DBMSs. This is mainly due to their simplicity and ease of implementation (their structure is an adaptation of the B^+-tree for spatial data), as well as their robust performance for many applications.

Foundations

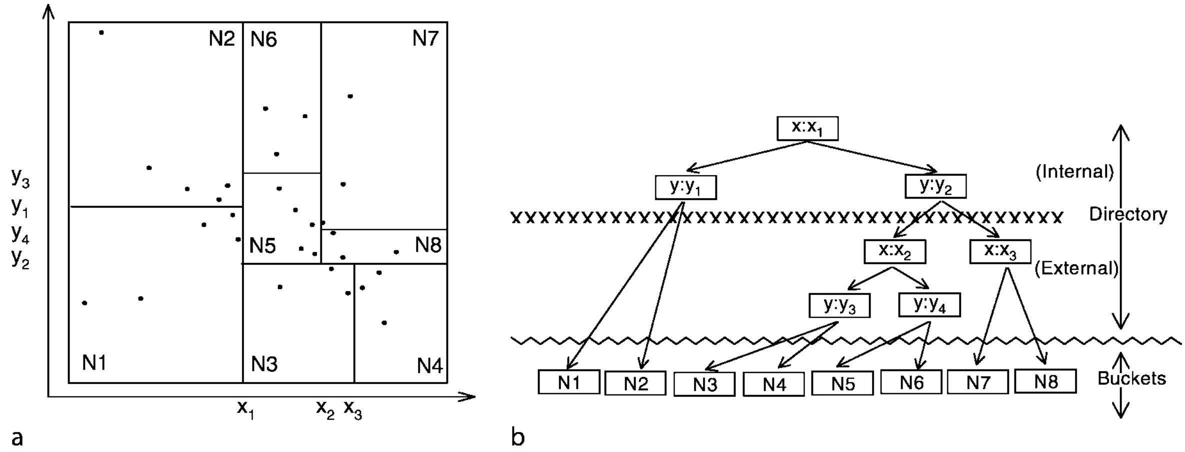
This section first discusses indices for multi-dimensional points, while the description of major indices for multidimensional (non-point) regions follows.

Indices for Multi-Dimensional Points

The LSD-tree (Local Split Decision tree), proposed in [1], maintains a catalog that separates space in a collection of (non-equal sized) disjoint subspaces using the extended k-d tree structure. New entries are inserted into the appropriate bucket. When an overflow happens then the bucket is split and the information about the partition line (split dimension and split position) is stored in a directory. Thus the overall structure of the LSD-tree consists of data buckets and a directory tree. The directory tree is kept in main memory until it grows more than a threshold; then a sub-tree is stored in an external catalogue in order for the whole structure to remain balanced (an example appears in Fig. 1). Inserting a new entry (point) in the LSD-tree is straightforward since nodes are disjoint. However, the target node may overflow due to an insertion; a split procedure then takes place.

The LSD-tree is a *space-driven structure*, i.e., it decomposes the complete workspace. Other members of this family include the Grid File [10] and the hB-tree [9]. On the other hand, *data-driven structures* only cover those parts of the workspace that contain data objects. Examples are the Buddy Tree [14] and the BV-tree [3].

The Grid File is an access method comprising of two separate parts: (i) the *directory*, and (ii) the *linear scales*. The Grid File imposes a grid on the indexed multidimensional attribute space. Each cell in this grid corresponds to one data page. The data points that "fall" inside a given cell are stored in the cell's corresponding page. Each cell must thus store a



Spatial Indexing Techniques. Figure 1. The LSD-tree.

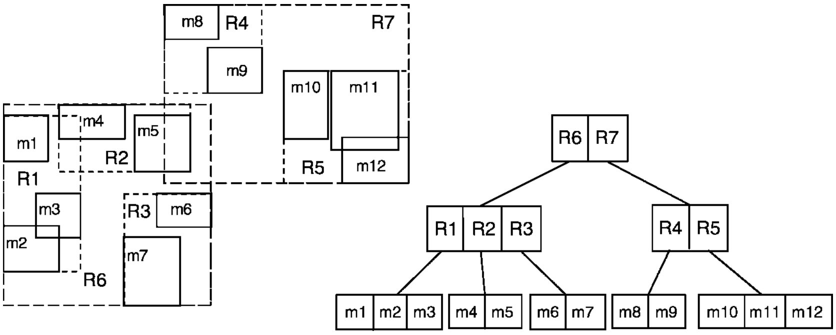
pointer to its corresponding page. This information is stored in the Grid File's *directory*. The information of how each dimension is divided (and thus how data values are assigned to cells) is kept in the *linear scales*. The Grid File can be thought as a multidimensional extension of hashing. As a result, exact match queries take only two disk accesses one for the directory and one for the data page.

Indices for Multi-Dimensional Regions

As with point indexing, two different approaches (data-driven and space-driven) have been proposed for indexing regions as well. The main representatives are the R-tree [5] and the Quadtree, [2,13], which were later followed by dozens of variants. In the sequel, the two structures are presented in detail. The reader is referred to a recent exhaustive survey [4] for further reading on their variants.

R-trees were originally proposed [5] as a direct extension of B$^+$-trees in n-dimensional space. The data structure is a height-balanced tree that consists of intermediate and leaf nodes. A leaf node is a collection of entries of the form (*o_id*, *R*) where *o_id* is an object identifier, used to refer to an object in the database, and *R* is the MBR minimum bounding rectangle approximation of the data object. An intermediate node is a collection of entries of the form (*ptr*, *R*) where *ptr* is a pointer to a lower level node of the tree and *R* is a representation of the minimum rectangle that encloses all MBRs of the lower-level node entries. Let M be the maximum number of entries in a node and let $m \leq M/2$ be a parameter specifying the minimum number of entries in a node. An R-tree satisfies the following properties: (i) every leaf node contains between m and M entries unless it is the root; (ii) for each entry (*o_id*, *R*) in a leaf node, *R* is the MBR minimum bounding rectangle approximation of the object represented by *o_id*; (iii) every intermediate node has between m and M children unless it is the root; (iv) for each entry (*ptr*, *R*) in an intermediate node, *R* is the smallest rectangle that completely encloses the rectangles in the child node; (v) the root node has at least two children unless it is a leaf; and (vi) all leaves appear at the same level. As an example, Fig. 2 illustrates several MBRs minimum bounding rectangle m_i and the corresponding R-tree built on these rectangles (assuming maximum node capacity $M = 3$).

In order for a new entry E to be inserted into the R-tree, starting from the root node, the child that needs minimum enlargement to include E is chosen (ties are resolved by choosing the one with the smallest area). When a leaf node N is reached, E is inserted into that, probably causing a split if N is already full. In such a case, the existing entries together with E are redistributed in two nodes (the current and a new one) with respect to the minimum enlargement criterion. In the original paper [6] three alternatives were proposed in order to find the two groups: an exhaustive, a quadratic-cost, and a linear-cost split algorithm. The processing of a point or range query with respect to a query window q (which could be either point or rectangle, respectively) is straightforward: starting from the root node, several tree nodes are traversed down to the leaves, depending on the result of the overlap operation between q and the corresponding node rectangles. When the search algorithm reaches



Spatial Indexing Techniques. Figure 2. The R-tree.

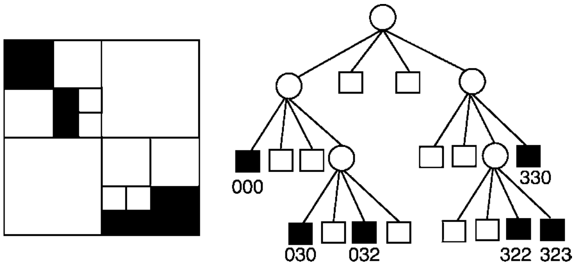
the leaf nodes, all data rectangles that overlap the query window *q* are added to the answer set. Regarding *k*-nearest-neighbor queries, [12] proposed customized branch-and-bound algorithms for R-trees.

After Guttman's proposal, several researchers proposed their own improvements on the basic idea. Roussopoulos and Leifker [11] proposed the *Packed* R-tree for bulk loading data in an R-tree. Objects are first sorted in some desirable order (according to the low-*x* value, low-*y* value, etc.) and then the R-tree is bulk loaded from the sorted file and R-tree nodes are packed to capacity. Note that the above techniques allow node "overlapping": MBRs of different nodes can overlap. Since no disjointness is guaranteed, during a search multiple paths of the R-tree may be traversed. An efficient variation, namely the R^+-tree, was proposed by Sellis et al. [15]. To preserve disjointness among node rectangles, the R^+-tree uses a "clipping" technique that duplicates data entries when necessary. However, the penalty is a (possibly high) increase in space demand due to the replication of data, which, in turn, degenerates search performance. Generally speaking, clipping techniques are ideal for point queries because a single path should be traversed, while range queries tend to be expensive, when compared with the overlapping techniques.

Later, Beckman et al. [1] and Kamel and Faloutsos [8] proposed two R-tree-based methods, the R*-tree and the Hilbert R-tree, respectively, which are currently considered to be the most efficient members of the R-tree family in terms of query performance. The R*-tree uses a rather complex but more effective grouping algorithm to split nodes by computing appropriate area, perimeter, and overlap values while the Hilbert R-tree actually stores Hilbert values at the leaf level and ranges of those values at the upper levels, similarly to the B^+-tree construction algorithm. In addition, a "lazy" split technique is followed, where overflow entries are evenly distributed among sibling nodes and only when all those are full, a new node (hence, split) is created.

The *Region Quadtree* [2] is the most popular member in the Quadtree family. It is used for the representation of binary images, that is $2^n \times 2^n$ binary arrays (for a positive integer *n*), where a "1" ("0") entry stands for a black (white) picture element. More precisely, it is a degree four tree with height *n*, at most. Each node corresponds to a square array of pixels (the root corresponds to the whole image). If all of them have the same color (black or white) the node is a leaf of that color. Otherwise, the node is colored grey and has four children. Each of these children corresponds to one of the four square sub-arrays to which the array of that node is partitioned. It is assumed here, that the first (leftmost) child corresponds to the upper left sub-array, the second to the upper right sub-array, the third to the lower left sub-array and the fourth (rightmost) child to the lower right sub-array, denoting the directions NW, NE, SW, SE single ended, respectively. Figure 3 illustrates a Quadtree for an 8 × 8 pixel array. Note that black (white) squares represent black (white) leaves, whereas circles represent internal nodes (also, *grey* ones).

Region Quadtrees, as presented above, can be implemented as main memory tree structures (each node being represented as a record that points to its children). Variations of Region Quadtrees have been developed for secondary memory. *Linear Region Quadtrees* [8] are the ones used most extensively. A linear Quadtree representation consists of a list of values where there is one value for each black node of the pointer-based Quadtree. The value of a node is an address describing



Spatial Indexing Techniques. Figure 3. The Quadtree.

the position and size of the corresponding block in the image. These addresses can be stored in an efficient structure for secondary memory (such as a B$^+$-tree). There are also variations of this representation where white nodes are stored too, or variations which are suitable for multicolor images. Evidently, this representation is very space efficient, although it is not suited to many useful algorithms that are designed for pointer-based Quadtrees. The most popular linear implementations are the FL Fixed Length (Fixed Length), the FD Fixed Depth (Fixed length – Depth) and the VL Variable Length (Variable Length) linear implementations [13]. Techniques for computing various kinds of geometric properties have also been developed. Connected component labeling, polygon coloring and computation of various types of perimeters fall in this category. Finally, many operations on images have been developed. For example, point location, set operations on two or more images (intersection, union, difference, etc.), window clipping, linear image transformations and region expansion.

Other region Quadtree variants have appeared in the literature mainly for indexing non-regional data. MX Quadtrees are used for storing points seen as black pixels in a Region Quadtree. PR Quadtrees are also used for points. However, points are drawn from a continuous space, in this case. MX-CIF Quadtrees are used for small rectangles. Each rectangle is associated with the Quadtree node corresponding to the smallest block that contains the rectangle. PMR Quadtrees are used for line segments. Each segment is stored in the nodes that correspond to blocks intersected by the segment. A detailed presentation of these and other region Quadtree variants is given in [8].

Key Applications

Geographic Information Systems (GIS) deal extensively with the management of 2- and 3- dimensional spatial data. For example, a map typically contains point objects (locations of interest), line objects (road segments, highways, rivers, etc.) as well as region objects (lakes, forests, etc.) GIS use spatial indexing as a means to provide fast access to large amounts of spatial data.

Multimedia Systems manage multimedia objects like images, text, audio, video, etc. A typical query in such systems is the *similarity* query (i.e., find objects that are similar to a query object according to some measure). To answer these queries, each multimedia object is abstracted by a set of multidimensional points (features). These multidimensional points are then indexed by a spatial index. Similarly, the query object is represented by a multidimensional point. The similarity query is then answered as a nearest neighbor query (i.e., find the nearest neighbor(s) to the point that represents the query).

The World-Wide Web has also provided new applications for geographic related queries and thus spatial indexing. Users can now find maps, driving directions, etc. through specialized web sites that typically offer the ability to perform spatial queries.

Location-based services provide querying capabilities based on the location of the user (e.g., “find the cheapest gas station within 5 miles of my car”). As the user moves in space, the results of the queries change. Such queries typically have a spatial component and spatial (and spatio-temporal) indexes are used to provide fast response.

CAD systems use spatial objects to store surfaces and bodies of design objects (for e.g., the wings or the wheels of an airplane). Typical spatial queries involve the proximity of spatial objects, their overlap etc. Related queries (but mainly in the 2-dimensional space) are also relevant for VLSI design systems; here the layout of a chip involves various rectangular regions and overlap and proximity queries are of importance.

Computer Games also involve many spatial searches. In such an environment, players move around in a 3-dimensional space and need to be able to see parts of (partially hidden) objects, various triggers are initiated if a player passed over them, or an explosion needs to identify the nearby objects that are affected. Spatial indexing is used to improve such query response.

Medical Imaging also involves large amounts of 2- and 3-dimensional spatial data. Consider for example X-rays, or, magnetic resonance imaging (MRI) brain scans. Again, proximity, overlap and related spatial queries are of interest.

Experimental Results

In general, for every presented method, there is an accompanying experimental evaluation in the corresponding reference.

Data Sets

A large collection of real spatial datasets, commonly used for experiments, can be found at *R-tree-portal* (URL: http://www.rtreeportal.org/).

URL to Code

R-tree portal (see above) contains the code for most common spatial and spatio-temporal indexes, as well as data generators and several useful links for researchers and practitioners in spatio-temporal databases. Similarly, the Spatial Index Library [6] provides a general framework for developing various spatial indices (URL: http://dblab.cs.ucr.edu/spatialindexlib).

Cross-references

- ▶ B-Tree
- ▶ GIS
- ▶ Grid File (and family)
- ▶ Nearest Neighbor Query
- ▶ Quadtrees (and family)
- ▶ R-Tree (and family)
- ▶ Spatial Join

Recommended Reading

1. Beckmann N., Kriegel H.-P., Schneider R., and Seeger B. The R*-tree: an efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 322–331.
2. Finkel R.A. and Bentley J.L. Quad Trees: a data structure for retrieval on composite keys. Acta Informatica, 4(1):1–9, 1974.
3. Freeston M.A. General solution of the n-dimensional B-tree problem. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp 80–91.
4. Gaede V. and Guenther O. Multidimensional access methods. ACM Comput. Surv., 30(2):170–231, 1998.
5. Guttman A. R-trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
6. Hadjieleftheriou M., Hoel E., and Tsotras V.J. SaIL: A spatial index library for efficient application integration. GeoInformatica, 9(4):367–389, 2005.
7. Henrich A., Six H.-W., and Widmayer P. The LSD tree: spatial access to multidimensional point and non point objects. In Proc. 15th Int. Conf. on Very Large Data Bases, 1989, pp. 43–53.
8. Kamel I. and Faloutsos C. Hilbert R-tree: an improved R-tree using fractals. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 500–509.
9. Lomet D.B. and Salzberg B. The hB-tree: a multiattribute indexing method with good guaranteed performance. ACM Trans. Database Syst., 15(4):625–658, 1990.
10. Nievergelt J., Hinterberger H., and Sevcik K.C. The grid file: an adaptable symmetric multikey file structure. ACM Trans. Database Syst., 9(1):38–71, 1984.
11. Roussopoulos N. and Leifker D. Direct Spatial Search on Pictorial Databases Using Packed R-trees. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1985, pp. 17–31.
12. Roussopoulos N., Kelley S., and Vincent F. Nearest neighbor queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 71–79.
13. Samet H. The Design and Analysis of Spatial Data Structures. Addison-Wesley, 1990.
14. Seeger B. and Kriegel H.-P. The Buddy-tree: an efficient and robust access method for spatial database systems. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 590–601.
15. Sellis T., Roussopoulos N., and Faloutsos C. The R^+-tree: a dynamic index for multidimensional objects. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 507–518.

Spatial Information System

- ▶ Geographic Information System
- ▶ Three-Dimensional GIS and Geological Applications

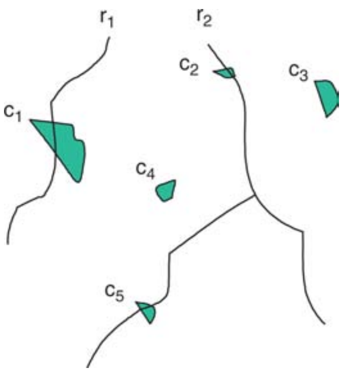
Spatial Join

NIKOS MAMOULIS
University of Hong Kong, Hong Kong, China

Definition

The spatial join is one of the core operators in spatial database systems. Efficient spatial join evaluation is important, due to its high cost compared to other queries, like spatial selections and nearest-neighbor searches. A binary (i.e., pairwise) spatial join combines two datasets with respect to a spatial predicate (usually overlap/intersect). A typical example is "find all pairs of cities and rivers that intersect." For instance, in Fig. 1 the result of the join between the set of cities $\{c_1, c_2, c_3, c_4, c_5\}$ and rivers $\{r_1, r_2\}$, is $\{(r_1, c_1), (r_2, c_2), (r_2, c_5)\}$.

The query in this example is a spatial intersection join. In the general case, the join predicate could be a combination of topological, directional, and



Spatial Join. Figure 1. Graphical example of a spatial intersection join.

distance spatial relations. Apart from the intersection join, variants of the distance join have received considerable attention, because they find application in data analysis tasks (e.g., data mining, clustering). Given two sets R and S of spatial objects (or multidimensional points), and a distance function $dist()$, the e-distance join (or else similarity join [1]) returns the pairs of objects $\{(r, s) : r \in R, s \in S, dist(r, s) \leq e\}$. A closest pairs query [2] returns the set of closest pairs $CP = \{(r, s) : r \in R, s \in S\}$, such that $dist(r, s) \leq dist(r', s')$, for all $r' \in R, s' \in S : (r', s') \notin CP$.

Historical Background

The first spatial join methods [3,4] assume that both inputs are indexed by some spatial access method (e.g., R-trees). The latest spatial join techniques do not rely on pre-existing indices [5–11]. Such situations may arise when at least one input is an intermediate result of a preceding operator. Consider for instance the query "find all rivers of width larger than 20 m, which intersect a forest." If there is a large percentage of narrow rivers, it might be natural to process the selection part of the query before the spatial join. In such an execution plan, even if there exists a spatial index on rivers, it is not employed by the join algorithm.

Table 1 classifies the spatial join techniques according to the assumption they make on pre-existing indices for the joined inputs. Methods of the first column can be applied only when both inputs are indexed (e.g., two relations Forests and Rivers are joined with respect to a spatial predicate). The second column includes algorithms suitable when only one input is indexed by an R-tree (e.g., Forests are joined with Rivers wider than 20 m). Join algorithms in the last column can be used in cases when both inputs are not indexed (e.g., Forests that intersect some City are joined with Rivers wider than 20 m). Most of the spatial join techniques focus on the filter step of the query. The refinement step (i.e., testing the exact geometry of objects against the join predicate) is applied independently of the algorithm, used for the filter step to the pairs that pass it, afterwards.

Spatial Join. Table 1. Classification of spatial join methods

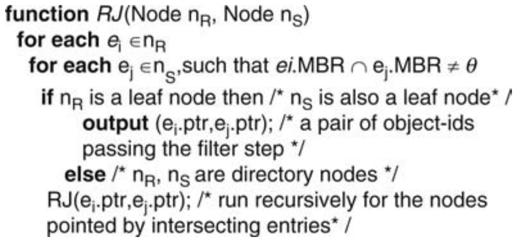
Both inputs are indexed	One input is indexed	Neither input is indexed
• Transformation to z-values and use of B-trees [4]	• Indexed nested loops	• Spatial hash join [7]
• Synchronized tree traversal [3]	• Seeded tree join [8]	• Partition-based spatial merge join [11]
	• Build a second R-tree and match it with the existing [10, 11]	• Size separation spatial join [6]
	• Sort and match [10]	• Sweeping-based spatial join [5]
	• Slot-index spatial join [9]	

Foundations

Early Spatial Join Algorithms

Most early spatial join algorithms apply transformation of objects in order to overcome difficulties due to their spatial extent and dimensionality. The first known spatial join algorithm [4] uses a grid to regularly divide the multidimensional space into small blocks, called pixels, and uses a space-filling curve (z-ordering) to order them. Each object is then approximated by the set of pixels intersected by its MBR, i.e., a set of z-values. Since z-values are one-dimensional, the objects can be dynamically indexed using relational index structures like the B^+-tree. The spatial join is then performed in a sort-merge fashion. The performance of the algorithm depends on the granularity of the grid; larger grids can lead to finer object approximations, but also increase the space requirements. Other approaches transform the MBRs of the objects into higher dimensional points and use k-d-trees or grid-files to index the points. The join is then performed by the use of these data structures in a similar way as relational multi-attribute joins.

The R-Tree Join R-tree Join (RJ) [2], often referred to as *tree matching or synchronous traversal*, computes the spatial join of two relations provided that they are both indexed by R-trees [5]. RJ synchronously traverses both trees, starting from the roots and following entry pairs which intersect. Let n_R, n_R be two directory (non-leaf) nodes of the R-trees that index relations *R* and *S*, respectively. RJ is based on the following observation: if two entries $e_i \in n_R$ and $e_j \in n_S$ do not intersect, there can be no pair (o_R, o_S) of intersecting objects, where o_R and o_S are under the sub-trees pointed by e_i and e_j, respectively. A simple pseudo-code for RJ that outputs the result of the filter spatial join step (i.e., outputs pairs of objects whose MBRs intersect) is given in Fig. 2. The pseudo-code assumes that both trees have the same height, yet it can be easily extended to the general case by applying range queries to the deeper tree when the leaf level of the shallow tree is reached.



```
function RJ(Node $n_R$, Node $n_S$)
 for each $e_i \in n_R$
  for each $e_j \in n_S$,such that $ei$.MBR $\cap$ $e_j$.MBR $\neq \theta$
   if $n_R$ is a leaf node then /* $n_S$ is also a leaf node* /
       output ($e_i$.ptr,$e_j$.ptr); /* a pair of object-ids
       passing the filter step */
     else /* $n_R$, $n_S$ are directory nodes */
    RJ($e_i$.ptr,$e_j$.ptr); /* run recursively for the nodes
    pointed by intersecting entries* /
```

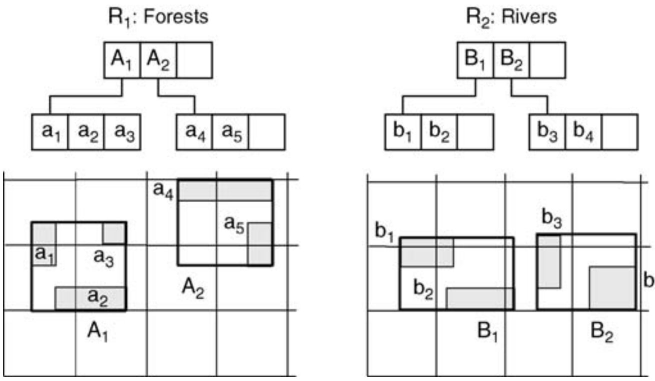
Spatial Join. Figure 2. The R-tree Join (RJ) Algorithm.

Figure 3 illustrates two datasets indexed by R-trees. Initially, RJ is run taking the tree roots as parameters. The qualifying entry pairs at the root level are (A_1, B_1) and (A_2, B_2). Notice that since A_1 does not intersect B_2, there can be no object pairs under these entries that intersect. RJ is recursively called for the nodes pointed by the qualifying entries until the leaf level is reached, where the intersecting pairs (a_1, b_1) and (a_2, b_2) are output.

Two optimization techniques can be used to improve the CPU speed of RJ [2]. The first (search space restriction) reduces the quadratic number of pairs to be evaluated when two nodes n_R, n_S are joined. If an entry $e_R \in n_R$ does not intersect the MBR of n_S (that is the MBR of all entries contained in n_S), then there can be no entry $e_S \in n_S$, such that e_R and e_S overlap. Using this fact, space restriction performs two linear scans in the entries of both nodes before RJ, and prunes out from each node the entries that do not intersect the MBR of the other node. The second technique, based on the plane sweep paradigm [15], applies sorting in one dimension in order to reduce the cost of computing overlapping pairs between the nodes to be joined. Plane sweep also saves I/Os compared to nested loops, because consecutive computed pairs overlap with high probability.

Algorithms That Do Not Consider Indexes

The most straightforward and intuitive algorithm that can be used to join two relations that are not indexed is the Nested Loops Join. This method can be applied for any type of joins (spatial, non-spatial) and condition predicates (topological, directional, distance, etc.).



Spatial Join. Figure 3. Two datasets indexed by R-trees.

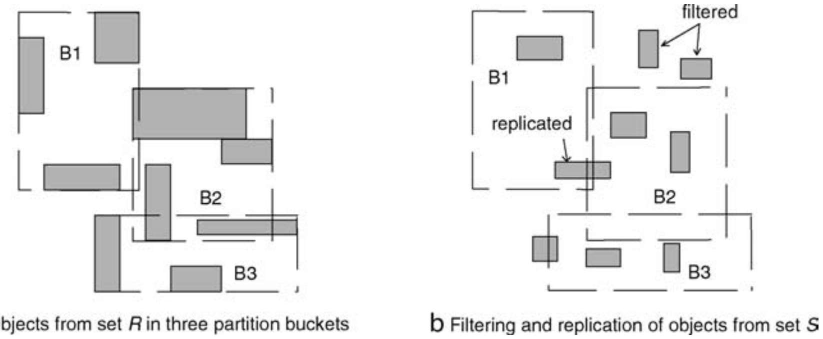
On the other hand, nested loops is the most expensive algorithm, since its cost is quadratic to the size of the relations (assuming that R and S have similar sizes). In fact, evaluation can be performed much faster. Spatial join algorithms for non-indexed inputs process the join in two steps; first the objects from both inputs are preprocessed in some data structures, and then these structures are used to quickly match objects that cover the same area. The algorithms differ in the data structure they use and the way the data are preprocessed.

Spatial Hash Join The Spatial Hash Join (HJ) [9] has common features with the relational hash-join algorithm. Set R is partitioned into K buckets, where K is decided by system parameters, such that the expected number of objects hashed in a bucket will fit in memory. The initial extents of the buckets are determined by sampling. Each object is inserted into the bucket whose bounding box is enlarged the least after the insertion. Set S is hashed into buckets with the same extent as R's buckets, but with a different insertion policy; an object is inserted into all buckets that intersect it. Thus, some objects may go into more than one bucket (*replication*), and some may not be inserted at all (*filtering*). The algorithm does not ensure partitions of equal number of objects from R, as sampling cannot guarantee the best possible slots. Equal sized partitions for S cannot be guaranteed in any case, because the distribution of the objects in the two datasets may be totally different. Figure 4 shows an example of two datasets, partitioned using HJ.

After hashing set S into buckets, the two bucket sets are joined; each bucket B_i^R from R is matched with the corresponding bucket B_i^S from S that covers the same spatial region. For this phase, a single scan of both sets of buckets is required, unless for some pairs of buckets none of them fits in memory. If one bucket fits in memory, it is loaded and the objects of the other bucket are matched with it in a nested-loops fashion. If none of the buckets fits in memory, an R-tree is dynamically built for one of them, and the bucket-to-bucket join is executed in an indexed nested-loop fashion.

Partition Based Spatial Merge Join Partition-based Spatial Merge Join (PBSM) [14] is also based on the hash join paradigm. The space, in this case, is regularly partitioned using an orthogonal grid, and objects from both datasets are hashed into partitions corresponding to grid cells, replicating wherever necessary. Figure 5a illustrates a regular space partitioning incurred by PBSM and some data hashed into the partitions. Objects hashed into the same partitions are then joined in memory using plane sweep. If the data inserted in a partition do not fit in memory, the algorithm recursively repartitions the cell into smaller parts and redistributes the objects. Since data from both datasets may be replicated, the output of the algorithm has to be sorted in order to remove pairs reported more than once.

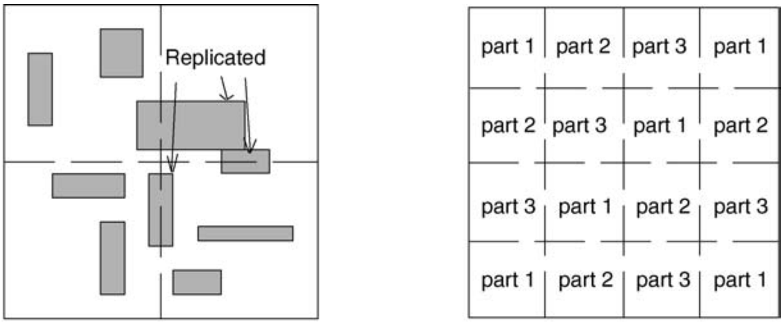
When the data to be joined are skewed, some partitions may contain a large percentage of the hashed objects, whereas others very few objects, rendering the algorithm inefficient. In order to evenly distribute the data in the partitions and efficiently handle skewed data, a spatial hash function is introduced. The cells of the grid are assigned to partitions according to this function and the space covered by a partition is no longer continuous, but consists of a number of scattered tiles. Figure 5b shows such a (round-robin like) spatial hash function.



a Objects from set R in three partition buckets

b Filtering and replication of objects from set S

Spatial Join. Figure 4. The partitioning phase of HJ algorithm.



part 1	part 2	part 3	part 1
part 2	part 3	part 1	part 2
part 3	part 1	part 2	part 3
part 1	part 2	part 3	part 1

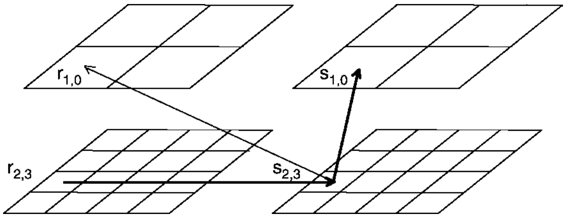
a Four partitions and a set of hashed objects

b A spatial hash function

Spatial Join. Figure 5. Regular partitioning by PBSM.

Size Separation Spatial Join Another algorithm that applies regular partitioning, like PBSM, but avoids object replication is Size Separation Spatial Join (S^3J) [6]. S^3J uses a hierarchical space decomposition. L partition layers of progressively larger resolution are introduced; the layer at level l partitions the space into $4l$ cells. A rectangle is then assigned to the topmost layer where it is not intersected by a grid line. This method achieves separation of the data according to their size. The rectangles in each layer are then sorted according to the Hilbert value of their MBRs center. A synchronized scan of the layer files is finally performed and the rectangles from dataset R in a partition at level l are joined with all partitions of dataset S that intersect it at levels $0,...,l$. A partition from S is joined with partitions from R at levels $0,...,l - 1$. The Hilbert values of the data inside a layer determine the order of the join, avoiding scanning a partition more than once. Figure 6 shows two partition layers of both datasets. Partition $r_{2,3}$ is joined with $s_{2,3}$ and $s_{1,0}$, and partition $s_{2,3}$ is joined with $r_{1,0}$.

S^3J also maintains a dynamic spatial bitmap which, after partitioning the first set, indicates the cells at each layer that contain at least one rectangle, or cover same area with cells at other layers that contain at least one rectangle. This bitmap can be used during the partitioning of the second set to filter entries that cannot intersect any rectangle of the first set. If a rectangle from set S is to be hashed into a partition cell and the bitmap entry of the cell is zero, the hashed rectangle is filtered out.



Spatial Join. Figure 6. Size separation spatial join.

Scalable Sweeping-Based Spatial Join The Scalable Sweeping-based Spatial Join (SSSJ) [1] is a relatively simple algorithm that is based on plane sweep. Both datasets are sorted according to the lower bound of their projection on an axis (e.g., the x-axis), and some variant of plane sweep (e.g., the *forward-sweep* algorithm described before) is applied to compute the intersection pairs. SSSJ is based on the square-root rule: the expected number of rectangles in a dataset R that intersect the sweep line is $\sqrt{|R|}$, where $|R|$ is the total number of rectangles in R. SSSJ initiates an internal memory plane sweep algorithm. If it runs out of memory, i.e., the rectangles intersected by the sweep line do not fit in memory, the space is dynamically partitioned by stripes parallel to the sorted axis, the rectangles are hashed into the stripes, and plane sweep is recursively executed for each stripe.

Single-Index Join Methods

Methods in this class can be applied when one input is not indexed. Such situations often arise when processing complex queries, where another operator precedes the spatial join. Notice that in this case index-based methods cannot directly be applied, because the intermediate result is not supported by any index. Also, algorithms that consider non-indexed inputs could be expensive. All single-index join methods were proposed after RJ, and they assume that the indexed input is supported by an R-tree. Most of them build a second structure for the non-indexed input and match it with the existing tree.

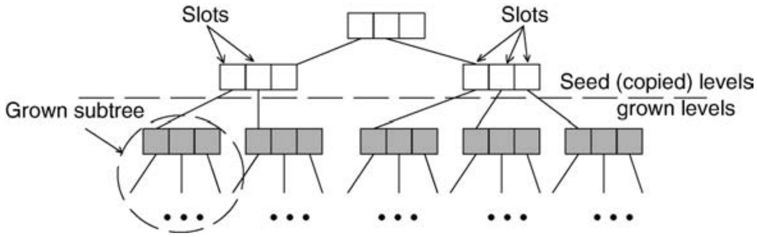
Indexed Nested Loops Join In accordance to the equivalent algorithm for relational joins, the Indexed Nested Loops Join (INLJ) applies a window query to the existing R-tree for each rectangle from the non-indexed set. This method can be efficient only when the non-indexed input is very small. Otherwise, the large number of selection queries can incur excessive computational overhead and access a large number of index pages.

Seeded Tree Join Let R be a dataset indexed by an R-tree and S be a non-indexed dataset. The Seeded Tree Join algorithm (STJ) [10] builds an R-tree for S, using the existing for R as a seed, and then applies RJ to match them. The rationale behind creating a *seeded* R-tree for the second input, instead of a normal R-tree, is the fact that if the new tree has similar high-level node extents with RA, this would lead to minimization of overlapping node pairs during tree matching. Thus, the seeded tree construction algorithm creates an R-tree which is optimal for the spatial join and not for range searching. The seeded tree construction is divided into two phases: the *seeding* phase and the *growing* phase. At the seeding phase, the top k levels (k is a parameter of the algorithm) of the existing R-tree are copied to formulate the top k levels of the new R-tree for S. The entries in the lowest of these levels are called slots. After copying, the slots maintain the copied extent, but they point to empty (null) sub-trees. During the growing phase, all objects from S are inserted into the seeded tree. A rectangle is inserted under the slot that contains it, or needs the least area enlargement. Figure 7 shows an example of a seeded tree structure. The top $k = 2$ levels of the existing R-tree are copied to guide the insertion of the second dataset.

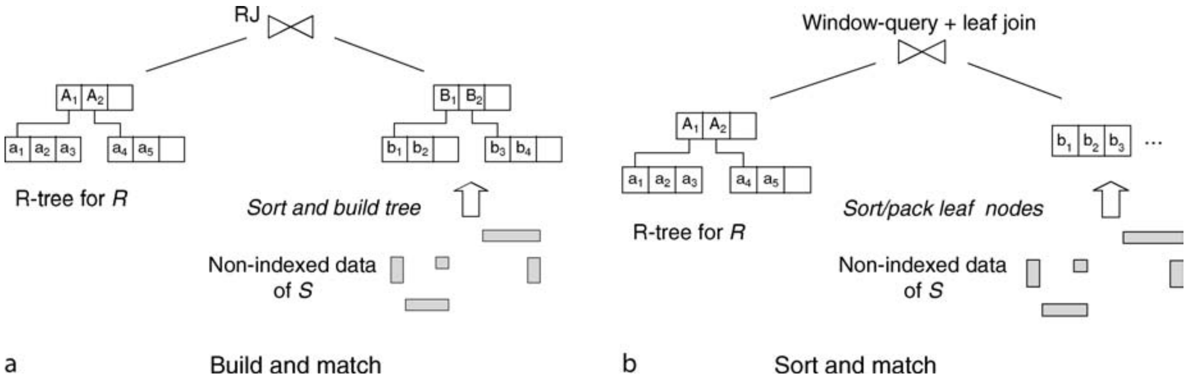
Build and Match Building a packed R-tree using bulk loading can be much more efficient in terms of both CPU time and I/O than constructing it incrementally. Moreover, packed R-trees have a minimum number of nodes and height, and could be very efficient for range queries and spatial joins. The Build and Match (BaM) method [13,14] first builds a packed R-tree for the non-indexed dataset S and then joins it with the existing tree of R, using RJ.

Sort and Match Sort and Match (SaM) [13] is an alternative of BaM, which avoids building a whole R-tree structure prior to matching. The algorithm employs an R-tree bulk-loading technique [8] to sort the rectangles from the non-indexed dataset S but, instead of building the packed tree, it matches each in-memory created leaf node with the leaf nodes from the R-tree of R that intersect it, using the structure of the tree to guide search. For each produced leaf node n_L at the last phase of STR, a window query using the MBR of n_L is applied on R's tree, in order to identify the leaf nodes there that intersect n_L. Plane sweep is then applied to match n_L with the qualifying leaves of R's tree. The matching phase of SaM is expected to be efficient, as two consecutive produced nodes will be close to each other with high probability, and there will be good utilization of the LRU buffer. Graphical examples of BaM and SaM are shown in Fig. 8.

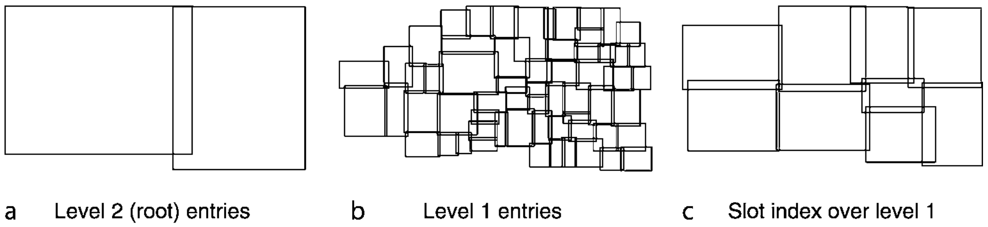
Slot Index Spatial Join The Slot Index Spatial Join [9] is a hash-based spatial join algorithm, appropriate for the case where only one of the two joined relations is indexed by an R-tree. It uses the existing R-tree to define a set of hash buckets. If K is the desired number of partitions (tuned according to the available memory), SISJ will find the topmost level of the tree such that the number of entries there is larger than or equal to K. These entries are then grouped into K (possibly overlapping) partitions called *slots*. Each



Spatial Join. Figure 7. A seeded tree.



Spatial Join. Figure 8. Algorithms based on bulk-loading.



Spatial Join. Figure 9. Entries of an R-tree and a slot index built over them.

slot contains the MBR of the indexed R-tree entries, along with a list of pointers to these entries. Figure 9 illustrates a three-level R-tree (the leaf level is not shown) and a slot index built over it. If $K = 9$, the root level contains too few entries to be used as partition buckets. As the number of entries in the next level is over K, they are partitioned in nine (for this example) slots. The grouping policy used by SISJ is based on the R*-tree insertion algorithm. After building the slot index, all objects from the non-indexed relation are hashed into buckets with the same extents as the slots. If an object does not intersect any bucket it is filtered; if it intersects more than one buckets it is replicated. The join phase of SISJ loads all data from the R-tree under a slot and joins them (in memory) with the corresponding hash-bucket from the non-indexed dataset (in a similar way as HJ).

Comparison of Spatial Join Algorithms

Since indexes can facilitate the spatial join operation, algorithms (like RJ) that are based on existence of indexes are typically more efficient compared to methods that do not rely on indexes. For example, RJ (which uses two R-trees) is expected to be more efficient than SISJ (which uses one R-tree), which is expected to be more efficient than HJ (which does not use trees).

RJ is the most popular index-based algorithm due to its efficiency and the fact that R-trees are becoming the standard access method in spatial database systems. Empirical and analytical studies have shown that the most efficient single-index methods are SISJ and SaM. Finally, conclusive results cannot be drawn about the relative performance of methods that do not consider indexes. S^{3}J is expected to be faster than PBSM and HJ when the datasets contain relatively large rectangles and extensive replication occurs in HJ and PBSM. On the other hand, this method uses sorting which is more expensive than hashing, in general. SSSJ is also based on sorting, thus it could be more expensive than hash-based methods. Furthermore, sort-based methods do not favor pipelining and parallelism of spatial joins. On the other hand, the fact that PBSM uses partitions with fixed extents makes it suitable for processing multiple joins in parallel, since the space partitions (and the local joins for them) can be assigned to different processors.

Key Applications

Spatial Database Systems

The spatial join is a core operation of Spatial Database Management Systems [4].

Geographic Information Systems

A fundamental operator in GIS is map overlay. Given two thematically different maps of the same region (e.g., elevation and political), this operator produces a join map that emphasizes on the overlaps of objects from both joined maps. Spatial join is the database operator used to produce this output.

Data Mining

In many applications that handle high dimensional data, an important analysis operation is to discover groups of objects that are close to each other in the multidimensional space. Examples of such mining tasks include "find stocks with similar movements" (time-series databases) and "find pairs of similar images" (multimedia databases). This type of clustering of complex data objects can be performed with the help of spatial joins with distance predicates [7]. Simply speaking, the original objects (e.g., images) are approximated by high dimensional feature vectors, and a spatial self-join is applied on this space to derive pairs of nearby objects, which are then postprocessed to larger groups (clusters).

Cross-references

▶ Hash Join
▶ Index Join
▶ Join
▶ Join Order
▶ Nested Loop Join
▶ Rtree
▶ Spatial Indexing Techniques

Recommended Reading

1. Arge L., Procopiuc O., Ramaswamy S., Suel T., and Vitter J.S. Scalable sweeping-based spatial join. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 570–581.
2. Brinkhoff T., Kriegel H.-P., and Seeger B. Efficient processing of spatial joins using r-trees. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 237–246.
3. Corral A., Manolopoulos Y., Theodoridis Y., and Vassilakopoulos M. Closest pair queries in spatial databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 189–200.
4. Güting R.H. An introduction to spatial database systems. VLDB J., 3(4):357–399, 1994.
5. Guttman A. R-trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
6. Koudas N. and Sevcik K.C. Size separation spatial join. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 324–335.
7. Koudas N. and Sevcik K.C. High dimensional similarity joins: algorithms and performance evaluation. IEEE Trans. Knowl. Data Eng., 12(1):3–18, 2000.
8. Leutenegger S.T., Edgington J.M., and Lopez M.A. Str: a simple and efficient algorithm for R-tree packing. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 497–506.
9. Lo M.-L. and Ravishankar C.V. Spatial hash-joins. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 247–258.
10. Lo M.-L. and Ravishankar C.V. The design and implementation of seeded trees: An efficient method for spatial joins. IEEE Trans. Knowl. Data Eng., 10(1):136–152, 1998.
11. Mamoulis N. and Papadias D. Slot index spatial join. IEEE Trans. Knowl. Data Eng., 15(1):211–231, 2003.
12. Orenstein J.A. Spatial query processing in an object-oriented database system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 326–336.
13. Papadopoulos A., Rigaux P., and Scholl M. A performance evaluation of spatial join processing strategies. In Proc. 6th Int. Symp. Advances in Spatial Databases, 1999, pp. 286–307.
14. Patel J.M. and DeWitt D.J. Partition based spatial-merge join. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 259–270.
15. Preparata F.P. and Shamos M.I. Computational Geometry - An Introduction. Springer, 1985.

Spatial k-Anonymity

▶ Spatial Anonymity

Spatial Network Databases

Betsy George, Shashi Shekhar
University of Minnesota, Minneapolis, MN, USA

Synonyms

Spatial graph databases

Definition

Spatial network databases render support for spatial networks by providing the necessary data model, query language, storage structure, and indexing methods. Spatial networks can be modeled as graphs where nodes are points embedded in space. One characteristic that distinguishes a spatial network database is the primary focus on the role of connectivity in relationships rather than the spatial proximity between objects. These databases are the kernel of many important applications, including transportation planning; air traffic control; water, electric, and gas utilities; telephone networks; urban management; utility network maintenance, and irrigation canal management. The phenomena of interest for these applications are structured as a spatial graph, which consists of a finite collection of the points (i.e., nodes), the line-segments (i.e., edges) connecting the points, the location of the points and the attributes of the points and line-segments. For example, a spatial network database storing a road network may store road intersection points and the road segments connecting the intersections (Fig. 1).

Foundations

Data Model of Spatial Networks

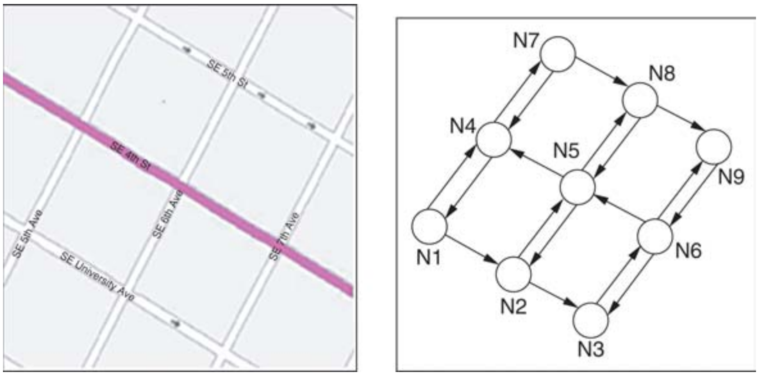
This section presents techniques related to the data modeling of spatial networks. The database design involves three steps, namely conceptual modeling, logical modeling and physical modeling.

Conceptual Data Model: The purpose of conceptual modeling is to adequately represent the data types, their relationships and the associated constraints. The Entity Relationship (ER) model, widely used in conceptual modeling, does not offer adequate features to capture the spatial semantics of networks. The most critical feature of spatial networks, namely the connectivity between objects can be expressed, using a graph framework. At the conceptual level, the pictogram enhanced ER (PEER) model [13] can be used. Figure 2 shows a PEER diagram for a spatial network. In a spatial graph, vertices represent road intersections and edges represent road segments. A path represents a street and consists of a series of edges.

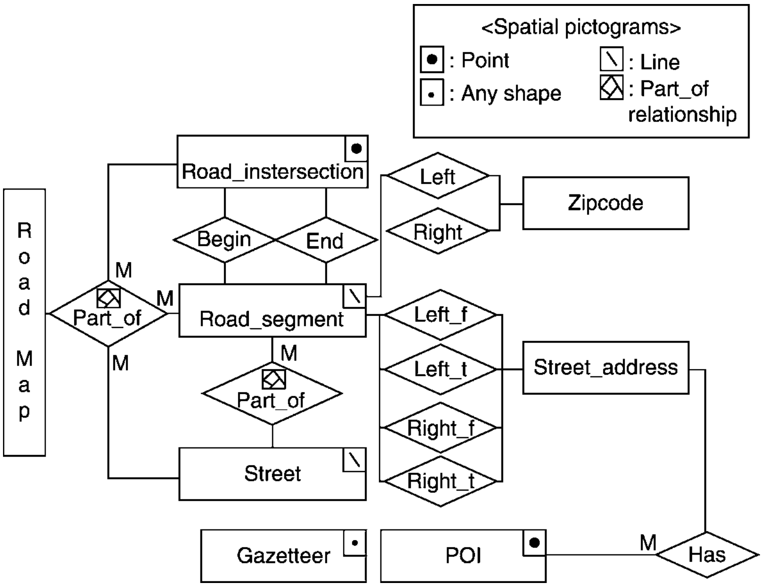
Labels and weights can be attached to vertices and edges to encode additional information such as names and travel times. Two edges are considered to be adjacent if they share a common vertex.

Modifications to the spatial network model have been proposed to make it more suitable in the context of some applications. For example, a simple node-edge network model might not be adequate to represent all features of a transportation network [9]. To address such limitations, various enhanced models have been proposed. One such model is the transportation data model (UNETRANS) that organizes the data model as three layers, namely (i) a reference network layer that represents the topological structure of the network, (ii) a route features layer that defines more complex features such as routes from the elements of the reference network layer, and (iii) the events layer that represents events such as traffic signs [2].

Logical Data Model: In the logical modeling phase, the conceptual data model is implemented using a commercial database management system. Among the various implementation models such as hierarchical, network, relational, object-relational data models and object-oriented models, the object-relational model has been gaining popularity in the representation of spatial applications. To model spatial network databases, graphs can be embedded into object-relational models. Shekhar and Chawla [11] lists some common graph operations used by spatial network applications,



Spatial Network Databases. Figure 1. A Road Map and its Spatial Network Representation. (a) A road map (b) Spatial Graph Representation (Source for Figure 1(a): http://maps.yahoo.com).



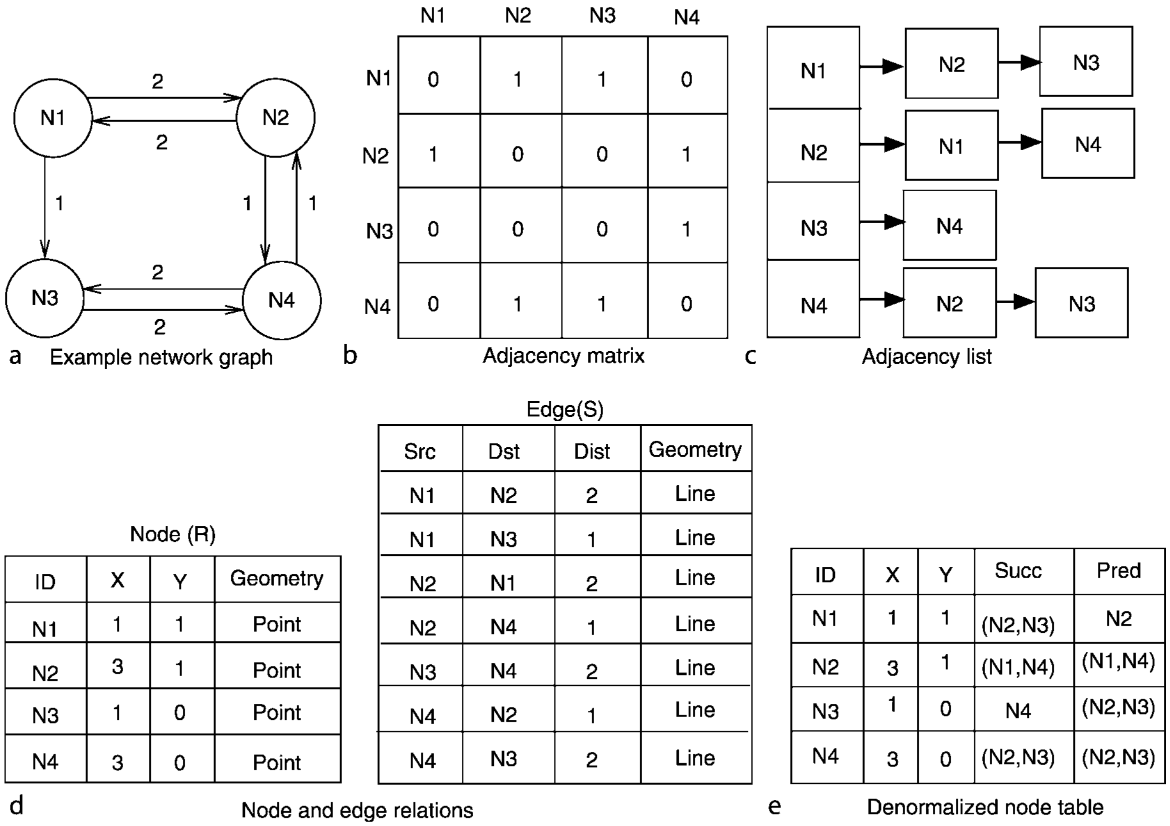
Spatial Network Databases. Figure 2. A PEER Diagram for Spatial Graph for a Road Network.

using a high-level object oriented notation that employs three fundamental classes in graphs, namely, Graph, Vertex, and Edge. Models such as GraphDB [5], which allow additional data types such as path, have also been proposed. A path class explicitly stores paths or routes in a graph, which contains the list of edges and nodes. In this model, an operator called "rewrite" can apply transformations to subsequences of heterogeneous sequences such as paths.

Physical Data Model: The physical data modeling phase deals with the actual implementation of the database application. Issues related to storage, indexing and memory management are addressed in this phase. Very often, queries that are posed on a network database such as a road map, involve route finding. This means the database must provide adequate support for network computations such as finding shortest paths. Figure 3 shows three representations of a graph. Adjacency-matrix and adjacency list are two well-known data structures used for implementing road networks represented as graphs [11]. In an adjacency-matrix, the rows and columns of a matrix represent the vertices of the graph. A matrix entry can be either 1 or 0, depending on whether there is an edge between the two vertices as shown in Fig. 3b. An adjacency list (shown in Fig. 3c) consists of an array of pointers. Each element of the array represents a vertex in the graph and the pointer points to a list of vertices that are adjacent to the vertex. Directed graphs can be implemented in the relational model using a pair of relations, one for the nodes and the other for the edges. The "Node" (R) and the "Edge" (S) relations are shown in Fig. 3d and a denormalized representation is shown in Fig. 3e. The denormalized representation of a node table contains the coordinates of the node, a list of its successors and a list of its predecessors. This representation is often used in shortest path computations.

A spatial access method called the Connectivity-Clustered Access Method (CCAM) was proposed in [12], which clusters the vertices of the graph based on graph partitions, thus providing an ordering based on connectivity.

Graph Algorithms: Frequent queries on a spatial network involve operations such as shortest path, nearest neighbor search, range search and closest pairs [10]. "Shortest" path algorithms find the least cost path between two nodes in a given graph. The cost of the path could be based on network distance, travel time or a user specified factor. Examples of popular shortest path algorithms are Dijkstra's algorithm and A* search. Nearest neighbor search algorithms find the point(s) closest to a given query point. Traditionally, the closest point was determined based on the Euclidean distances, which did not consider the network connectivity of the objects. However, in practice, trajectories of objects are usually constrained by an underlying spatial network such as a road network and hence algorithms that find nearest neighbors and closest pairs that



Adjacency matrix

	N1	N2	N3	N4
N1	0	1	1	0
N2	1	0	0	1
N3	0	0	0	1
N4	0	1	1	0

Node (R)

ID	X	Y	Geometry
N1	1	1	Point
N2	3	1	Point
N3	1	0	Point
N4	3	0	Point

Edge(S)

Src	Dst	Dist	Geometry
N1	N2	2	Line
N1	N3	1	Line
N2	N1	2	Line
N2	N4	1	Line
N3	N4	2	Line
N4	N2	1	Line
N4	N3	2	Line

ID	X	Y	Succ	Pred
N1	1	1	(N2,N3)	N2
N2	3	1	(N1,N4)	(N1,N4)
N3	1	0	N4	(N2,N3)
N4	3	0	(N2,N3)	(N2,N3)

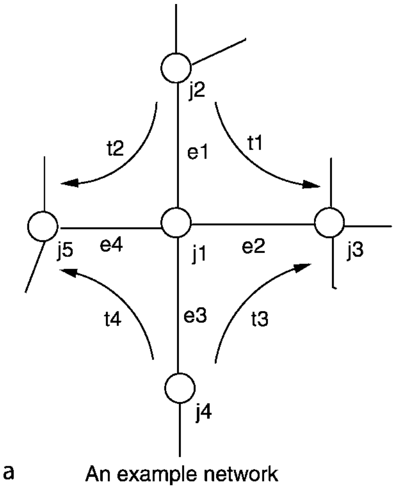
Spatial Network Databases. Figure 3. Three different representations of a graph.

consider network connectivity are critical in a spatial network database. Query processing algorithms that find nearest neighbors and closest pairs have been proposed [7,10].

Turn Restrictions: Turn restrictions are frequently encountered in road networks and they can affect the traversal in the network. A physical model that does not consider turn restrictions can lead to the computation of routes that are not entirely feasible. Turns have been modeled using a turn table where each turn restriction is represented as a row in the table that references the two associated edges [9]. Another proposed method to represent turn restrictions is node expansion [1]. The node that corresponds to a junction is expanded to a subgraph where permissible turns are represented as edges. This technique can lead to a substantial increase in the size of the network, which adversely affects the performance. Another method involves the transformation of the road network to a line graph where the edges in the original network are mapped to vertices in the line graph and the turns are represented as edges in the line graph [15].

A representation, consisting of a junction table, edge table and turn table was proposed in [6]. Every junction is represented as a row in the junction table. A row corresponding to a junction stores the edges that converge at the junction and the junctions connected to the given junction. The edge table stores edge identifiers and the junction where the edge originates (from-junction). A tuple in the turn table corresponds to a junction in the network. Each tuple consists of a junction identifier, and a triplet (turn identifier, first edge-id, last edge-id) corresponding to each turn associated with the given junction.

Figure 4 illustrates the representation of turn restrictions in a road network. Figure 4a shows a part of a road network around a junction *j1* where the edges *e1*, *e2*, *e3* and *e4* meet. The curved arrows indicate the permitted turns at the junction. For example, a turn is allowed from edge *e1* to edge *e2*. Figures 4b–4d show the edge, junction and turn tables respectively, corresponding to turn *t1* in the example *e4*) and the junctions connected to it (*j2*, *j3*, *j4*, and *j5*). The turn table shows the permitted turns at junction *j1* and the edges



id	from–jn
e1	j2
e2	j3
e3	j4
e4	j5

b Edge table

id	edge1	junc1	edge2	junc2	edge3	junc3	edge4	junc4
j1	e1	j2	e2	j3	e3	j4	e4	j5

c Junction table

id	Turn id1	First edge id1	Last edge id1	Turn id2	First edge id2	Last edge id2	Turn id3	First edge id3	Last edge id3	....
j1	t1	e1	e2	t2	e1	e4	t3	e3	e2	

d Turn table

Spatial Network Databases. Figure 4. Representation of Turn Restrictions (adapted from [6]).

that participate in each turn. For example, turn *t1* represents a turn from edge *e1* to edge *e2* as illustrated by the "*first edge id*" and "*last edge id*" entries in the turn table in Fig. 4d.

Key Applications

Location-Based Services

Spatial network databases are indispensable for any location-based service that involves route based queries [14]. Location-based services (LBS) provide the ability to find the geographical location of a mobile device and subsequently provide services based on that location. Spatial network databases play a key role in providing efficient query-processing capabilities such as finding the nearest facility (e.g., a restaurant) and the shortest path to the destination from a given location. Route-finding queries typically deal with route choice (shortest route to a given destination), destination choice (the nearest facility from the given location) and departure time choices (the time to start the journey to a destination so that the travel time is minimized). Though a significant amount of work has been done to find best routes and destinations, the problem of computing the best time to travel on a given route (time choice) needs further exploration.

Emergency Planning

One key step in emergency planning is to find routes in a road network to evacuate people from disaster-stricken areas to safe locations in the least possible time. This requires finding shortest routes from disaster areas to destinations. In metropolitan-sized transportation networks, manual computation of the required routes is almost impossible, making digital road maps integral to the efficient computation of these routes.

Future Directions

A significant fraction of queries that are posed on a road network involves finding the shortest path between a pair of locations. Travel times on the road segments very often depend on the time of day due to varying levels of congestion, thus making the shortest paths also time-dependent. Road networks need to be modeled as spatio-temporal networks to account for this time-dependence. Various models such as time-expanded networks [8] and time-aggregated graphs [4,3] are being explored in this context. A time expanded graph represents the time-dependence by copying the network for every time instant whereas in time aggregated graphs, the time-varying attributes are aggregated over edges and nodes.

Cross-references

▶ Graph
▶ Graph Database
▶ Road Networks

Recommended Reading

1. Anez J., de la Barra T., and Perez B. Dual graph representation of transport networks. Transport. Res., 30(3):209–216, 1996.
2. Curtin K., Noronha V., Goodchild M., and Grise S. ARCGIS Transportation Model (UNETRANS), UNETRANS Data Model Reference, December 2003.
3. George B. and Shekhar S. Time-aggregated graphs for modeling spatio-temporal networks – an extended abstract. In Proc. Workshops at Int. Conf. on Conceptual Modeling, 2006, pp. 85–99.
4. George B. and Shekhar S. Spatio-temporal network databases and routing algorithms: a summary of results. In Proc. 10th Int. Symp., Advances in Spatial and Temporal Databases, 2007, pp. 460–477.
5. Guting R.H. GraphDB: modeling and querying graphs in databases. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994.
6. Hoel E.G., Heng W.L., and Honeycutt D. High performance multimodal networks. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005.
7. Jensen C.S., Kolar J., Pederson T.B., and Timko I. Nearest neighbor queries in road networks. In Proc. 11th ACM Int. Symp. on Advances in Geographic Inf. Syst., 2003.
8. Kohler E., Langtau K., and Skutella M. Time-expanded graphs for flow-dependent transit times. In Proc. Tenth Annual European Symp. on Algorithms, 2002.
9. Miller H.J. and Shaw S.L. GIS-T Data Models, Geographic Information Systems for Transportation: Principles and Applications. Oxford University Press, Oxford, 2001.
10. Papadias D., Zhang J., Mamoulis N., and Tao Y. Query processing in spatial network databases. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003.
11. Shekhar S. and Chawla S. Spatial Databases: A Tour. Prentice Hall, Englewood Cliffs, NJ, 2002.
12. Shekhar S. and Liu D.R. CCAM: a connectivity-clustered access method for networks and network computations. IEEE Trans. Knowl. Data Eng., 9(1):102–119, 1997.
13. Shekhar S., Vatsavai R., Chawla S., and Burke T.E. Spatial pictogram enhanced conceptual data models and their translation to logical data models. In Proc. Int. Workshop on Integrated spatial databases, digital maps, and GIS, 1999.
14. Shekhar S., Vatsavai R., Ma X., and Yoo J. Navigation systems: a spatial database perspective, Chapter 3. In Location-Based Services, J. Schiller, A. Voisard (eds.). Morgan Kaufmann, 2004.
15. Winter S. Modeling costs of turns in route planning. GeoInformatica, 6(4):345–361, 2002.

Spatial Operations and Map Operations

MICHEL SCHOLL[1], AGNÈS VOISARD[2,3]
[1]Cedric-CNAM, Paris, France
[2]Fraunhofer Institute for Software and Systems Engineering (ISST), Berlin, Germany
[3]Free University of Berlin, Berlin, Germany

Synonyms

Map Algebra; Layer Algebra; Theme Algebra

Definition

Map operations refer to the operations that an end user performs on maps stored in a database. Map information is stored according to themes (for instance, cities, roads, or population), sometimes called layers in the GIS terminology. The maps considered here are stored in a vector format – as opposed to a raster format such as a grid of pixels – and can be 1 dimensional (e.g., a network of roads for a navigation system), 2 dimensional (e.g., a map of land-use for regional planning activities), or 2.5 dimensional if the elevation at certain locations is considered (for instance, the height of a building in an architecture project). A map is made of what is often called *geographic objects*. A geographic object (for instance, a city) has two parts, an alphanumeric one (e.g., its name and population) and a spatial one (e.g., a polygon), usually called *spatial object*. The alphanumeric attributes of a geographic object constitute its description. Map operations may use operations on spatial objects, commonly referred to as *spatial operations*. When describing the structure of a map – its description and its spatial part – together with its associated operations, one refers to a *map model*. The same applies to the structure and behavior of the spatial part of the geographic objects, leading to a *spatial model*.

Historical Background

When data are stored in a database, it is accessed through a query language, such as SQL. In the case of maps, however, SQL needs to be extended in order to consider their spatial component. With the emergence of GIS in the 80s, and also because maps are particular "non standard" entities, a set of operations to manipulate them was proposed by database researchers to describe these operations at a high level of abstraction, i.e., without considering SQL details. The idea was to define, at a conceptual level, unary or binary operations on maps that possibly take other arguments (alphanumerical or spatial) and that return maps, hence the term *algebra* often used in this context. Such operations also need spatial operations (geometric or topological) such as the intersection of polygons. Many proposals for lists of spatial operations were made. [8] is one of the first attempts to describe general map operations from a GIS view point. [7] proposed an extensible query language to the designer of geographic databases, independent of any underlying database model. The geo-relational algebra [2] was a pioneer approach, proposing an algebra based on the relational model that encompassed spatial operations. The SpatialSQL language [1] includes a list of spatial operations to be eventually used in conjunction with SQL. Operations on thematic layers were also proposed (e.g., [4]). The ROSE algebra [3] is a rich approach based on the relational model that allows extensible sets of functions. The OGIS Standard for SQL [5] from the Open GIS Consortium (OGC) focuses on spatial operations to be integrated in SQL and proposes an exhaustive list of such operations. Most current commercial approaches such as Oracle or ArcGIS from ESRI offer data types and operations that are OGIS compliant.

Foundations

This part focuses on end-user map operations that are performed in a database. In current applications, maps are usually stored in a relational database extended to abstract spatial data types. A kernel of elementary operations that can be combined in order to answer complex queries is presented here. The following list is coming from [2,6].

Map model. A map M is defined as a set of geographic objects: $M = \{g\}$ where a geographic object g is defined as follows: $g = \{<A>, S\}$, where $<A>$ is a list of alphanumerical attributes and S a spatial attribute.

Spatial model. The spatial attribute S of a geographic object corresponds to its associated geometric part (it also has topological relationships with other objects). It can be simple (for instance, one polygon for a lake) or complex, i.e., made of many parts (for instance, many polygons for a given country and its islands). A spatial attribute has a certain type and can be 0-, 1-, or 2-dimensional. Note that dimensions are often not mixed in a spatial attribute. An entity with a spatial type is usually called a spatial object and the referential used is the Euclidean plane.

The basic types of spatial objects that are usually considered are:

POINT	a 0-dimensional spatial object
LINE	a 1-dimensional spatial object made of segments
REG	a 2-dimensional spatial object made of polygons

as well as sets of these objects.

Spatial Operations

The operations presented below are primitives on spatial objects that are used in map operations. In the following, their signature is used for their short description (i.e., the type of arguments that these operations take and the one that they return). The spatial operations are presented here according to four groups: spatial predicates, spatial extractions, set operations, and geometric operations. Other classifications based for instance on the types of arguments are also sensible. Note also that the list given below is not exhaustive.

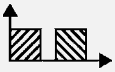
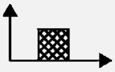
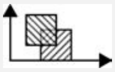
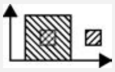
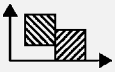
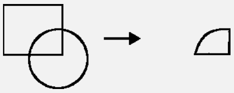
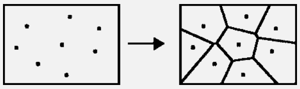
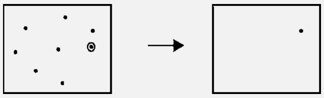
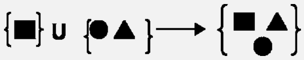
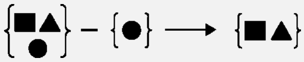
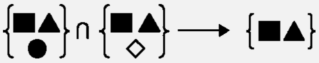
Map Operations

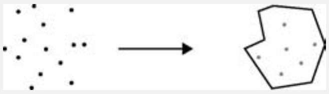
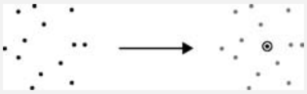
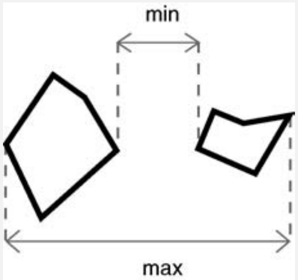
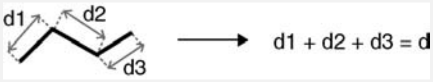
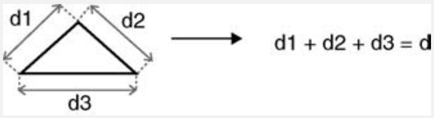
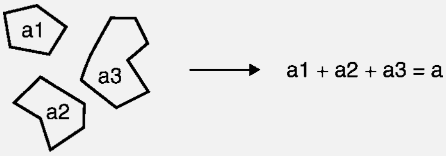
The operations described below constitute a common set of operations on maps. They are illustrated using the map of the 12 districts of Berlin, Germany, or a subset of it in some cases, and using the following schemas:

District (Name:STRING, Population:NUM, Area: REG)
CityDivision (Name:STRING, Area:REG)

Other Operations

The list of operations given above is not exhaustive but it corresponds to a kernel of common general operations on thematic maps. They are typical database operations performed in a GIS. Other GIS operations that are not detailed here include classification, zoom in, zoom out, as well as operations on layers stored in a raster form.

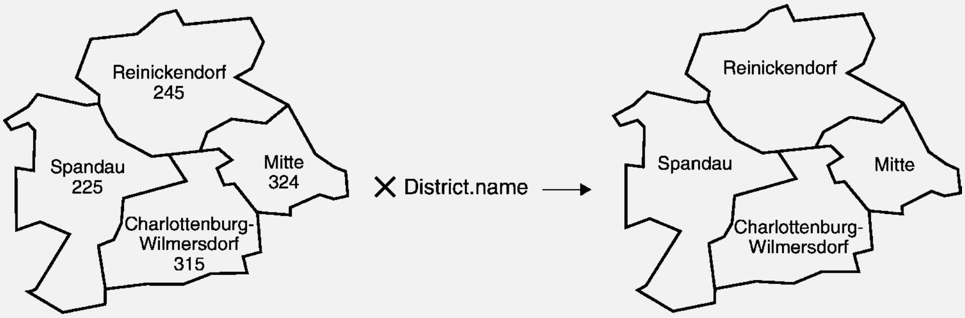
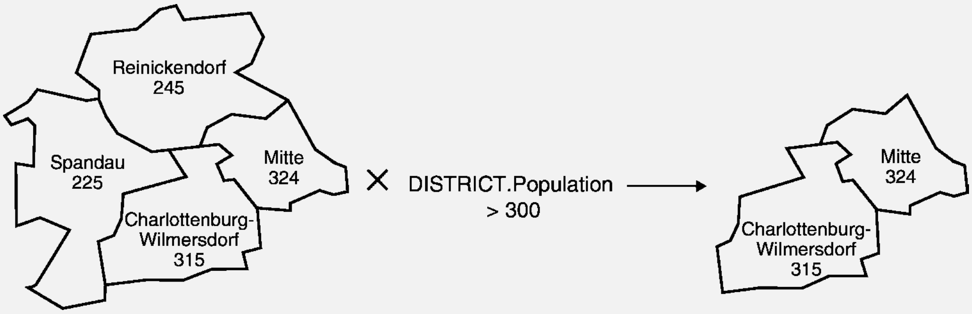
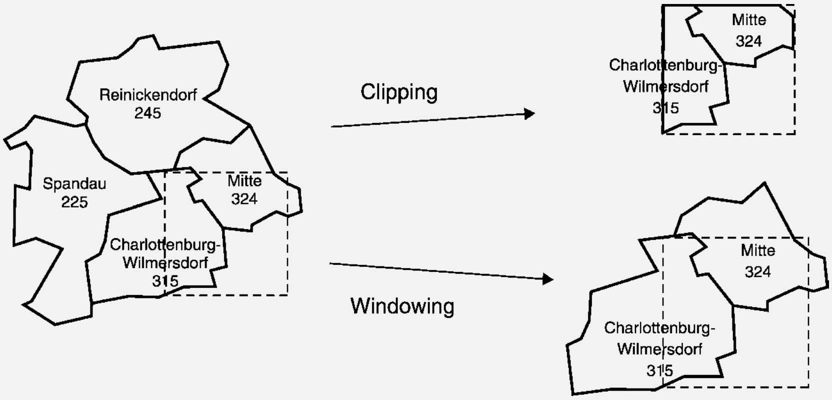
Group 1: Spatial Predicates		
In the following, BOOL represents a Boolean value (true or false) and REG is an abbreviation for the REGION type.		
Different 	Tests whether two spatial objects are different in the plane	
	Possible signatures:	*POINT × POINT → BOOL, LINE × LINE→ BOOL, REG × REG → BOOL*
Equal 	Tests whether two spatial objects are the same (i.e., have the same value in the plane)	
	Possible signatures:	*POINT × POINT → BOOL, LINE × LINE→ BOOL, REG × REG → BOOL*
Intersects 	Tests whether two spatial objects intersect	
	Possible signatures:	*LINE × LINE→ BOOL, LINE × REG→ BOOL, REG × REG → BOOL*
Inside/Outside 	Tests whether a spatial object is inside/outside a given region	
	Possible signatures:	*POINT × REG → BOOL, LINE × REG→ BOOL, REG × REG → BOOL*
Adjacent 	Tests whether two spatial objects are adjacent (i.e., have a common boundary)	
	Possible signatures:	*LINE × REG→ BOOL, REG × REG → BOOL*
Group 2: Spatial Extractions		
These operators transform their spatial input into another type of spatial object.		
Intersection 	Returns the intersection of two spatial objects	
	Possible signatures:	*LINE × LINE→ {POINT}, LINE × LINE → LINE, LINE × REG → {POINT}, LINE × REG → LINE, REG × REG → {POINT}, REG × REG → LINE, REG × REG → REG*
Voronoi 	Returns the Voronoi diagram of a region. Note that the returned set of regions has the particularity that the regions do not overlap	
	Signature:	*{POINT} × REG → {REG}*
Closest 	Returns the closest spatial object from a given object, taken from a set	
	Possible signatures:	*POINT × {POINT} → POINT, POINT × {LINE} → LINE, POINT × {REG} → REG, LINE × {POINT} → POINT, LINE × {LINE} → LINE, LINE × {REG} → REG, REG × {POINT} → POINT, REG × {LINE} → LINE, REG × {REG} → REG*
Group 3: Set Operations		
The following operations are common operations on sets of objects. Note again that a set only contains entities of the same type.		
SetUnion 	Returns the union of two sets of spatial objects, in the mathematical sense	
	Possible signatures:	*{POINT} × {POINT} → {POINT}, {LINE} × {LINE} → {LINE}, {REG} × {REG} → {REG}*
SetDifference 	Returns the difference of two sets of spatial objects	
	Possible signatures:	*{POINT} × {POINT} → {POINT}, {LINE} × {LINE} → {LINE}, {REG} × {REG} → {REG}*
SetIntersection 	Returns the intersection of two sets of spatial objects	
	Possible signatures:	*{POINT} × {POINT} → {POINT}, {LINE} × {LINE} → {LINE}, {REG} × {REG} → {REG}*

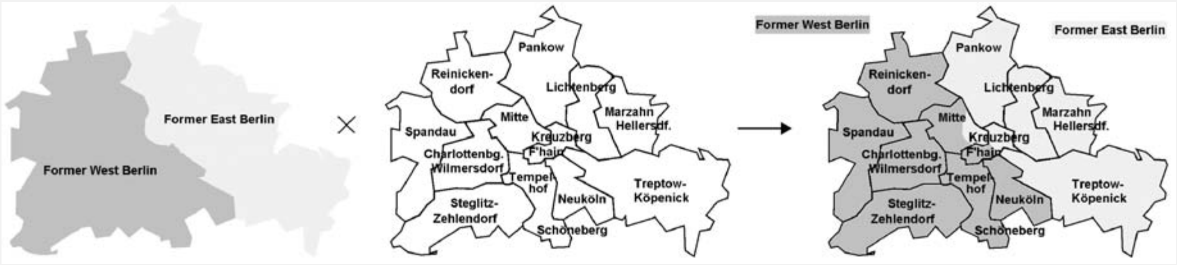
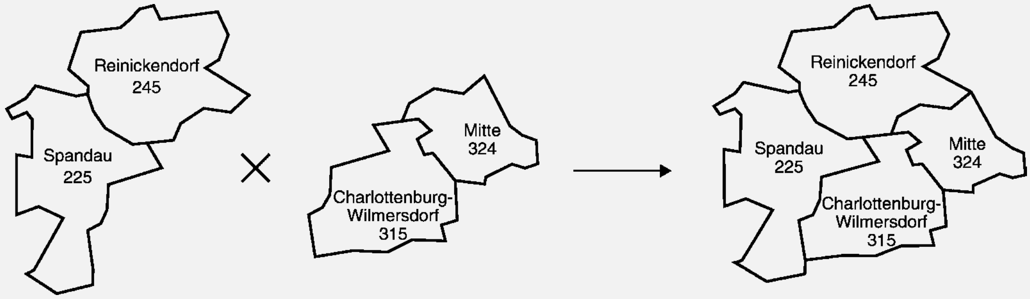
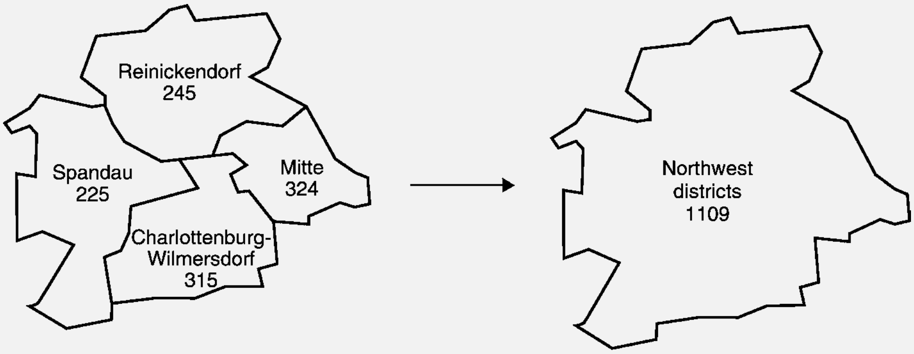
Group 4: Geometric Operations		
In the following, NUM represents a numerical value.		
Convex Hull 	Returns the region (polygon) that encompasses all the points given in the argument set	
	Signature:	*{POINT} → REG*
Center 	Returns the center of a set of points	
	Signature:	*{POINT} → POINT*
Min/Max Distance 	Returns the minimal (respect. maximal) distance between two spatial objects	
	Possible signatures:	*POINT × LINE → NUM, LINE × LINE → NUM, POINT × REG → NUM, LINE × REG → NUM, REG × REG → NUM*
Length 	Returns the length of a line	
	Signature:	*LINE → NUM*
Perimeter 	Returns the perimeter of a region. In case the region is composed of many polygons, it returns the sum of their respective perimeters	
	Signature:	*REG → NUM*
Area 	Returns the area of a region (total area if it is composed of many polygons)	
	Signature:	*REG → NUM*

Key Applications

Users likely to eventually use these map operations are end-users who need to get information from existing maps, for instance for the purpose of planning or geo-marketing. However, this conceptual approach – which moves away from implementation details – is targeted towards spatial database application designers who need to design appropriate application environments. Such environments should be easy to use, extensible, and adaptable to various application needs. They should, moreover, offer an efficient operation processing, however, this aspect is not handled by the conceptual approach presented here.

The key applications of this area concern thematic map manipulation (by the census bureau, city planners, local administrators, transportation managers, and so on) in order to perform statistics and analysis on data having a spatial dimension.

<table>
<tr><td>Map Projection</td><td colspan="2">Returns a map having a list of attributes given as argument and an unchanged spatial part</td></tr>
<tr><td></td><td>Signature:</td><td><i>map × &lt;A> → map, where <A> is a collection of alphanumerical attributes</i></td></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Map Selection</td><td colspan="2">Returns a map whose geographic objects satisfy the selection criteria given as argument (e.g., population greater than 300 thousand inhabitants)</td></tr>
<tr><td></td><td>Signature:</td><td><i>map × selection-criteria (<A>) → map, where selection-criteria (<A>) is a predicate on one or many alphanumerical attributes</i></td></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>Spatial Selections</td><td>- <i>Windowing (or region query):</i></td><td>Returns the map made of the original map whose objects intersect the region given as argument (often, a rectangle)</td></tr>
<tr><td></td><td>- Clipping:</td><td>Returns the part of the map that is exactly in the region given as argument</td></tr>
<tr><td></td><td>Signature:</td><td><i>map × REG→ map</i></td></tr>
<tr><td colspan="3">
</td></tr>
</table>

Map Overlay	Generates a new map from two (overlaid) maps, for instance a map of the former city division in Berlin and the map of districts. It uses the intersection operation on spatial objects. It creates new geographic objects as can be seen in the center of the newly created map. This operation is also called a spatial join in the database terminology	
	Signature:	*map* × *map* → *map*
	Note:	*For legibility reasons, the following map of districts does not include their population.*
 Former East Berlin Former West Berlin × Pankow Reinicken-dorf Lichtenberg Marzahn Hellersdf. Mitte Spandau Kreuzberg F'hain Charlottenbg. Wilmersdorf Tempelhof Steglitz-Zehlendorf Neukölln Treptow-Köpenick Schöneberg Former West Berlin Former East Berlin		
Map Union	Returns a map made of the two arguments	
	Signature:	*map* × *map* → *map*
 Reinickendorf 245 Spandau 225 × Mitte 324 Charlottenburg-Wilmersdorf 315 Reinickendorf 245 Spandau 225 Mitte 324 Charlottenburg-Wilmersdorf 315		
Fusion	Performs the geometric union of the spatial part of geographic objects that belong to the same map. Note the sum of the population in the example	
	Signature:	*map* → *map*
 Reinickendorf 245 Spandau 225 Mitte 324 Charlottenburg-Wilmersdorf 315 Northwest districts 1109		

Cross-references

- ▶ Geographic Information System
- ▶ OGC
- ▶ Semantic Modeling for Geographic Information Systems
- ▶ Spatial Data Types

Recommended Reading

1. Egenhofer M.J. Spatial SQL: a query and presentation language. IEEE Trans. Knowl. Data Eng., 6(1):86–95, 1994.
2. Güting R.H. Geo-relational algebra: a model and query language for geometric database systems. In Advances in Database Technology, Proc. 1st Int. Conf. on Extending Database Technology, 1988, pp. 506–527.
3. Güting R.H. and Schneider M. Realm-based spatial data types: the ROSE algebra. VLDB J., 4(2):243–286, 1995.
4. Hadzilacos T. and Tryfona N. Logical data modelling for geographical applications. Intl. J. Geogr. Inf. Sci., 10(2):179–203, 1996.
5. Open GIS Consortium. OpenGIS® Geographic objects implementation specification, 2007.

6. Rigaux P., Scholl M., and Voisard A. Spatial Databases – With Application to GIS, Chapter 3. Morgan Kaufmann/Elsevier, 2001.
7. Scholl M. and Voisard A. Thematic map modeling. In Proc. Int. Symp. on Spatial Databases, 1989, pp. 167–190.
8. Tomlin D. A Map algebra. In Proc. Harvard Computer Graphic Conf., 1983.

Spatial Outliers

▶ Spatial Data Mining

Spatial Referencing

▶ Georeferencing

Spatial Statistics

▶ Spatial Data Mining

Spatio-Temporal Approximation

▶ Spatiotemporal Interpolation Algorithms

Spatio-Temporal Benchmarking

▶ Real and Synthetic Test Datasets

Spatio-Temporal Data Generator

▶ Real and Synthetic Test Datasets

Spatio-Temporal Data Mining

NIKOS MAMOULIS
University of Hong Kong, Hong Kong, China

Synonyms

Data mining in moving objects databases

Definition

The extraction of implicit, non-trivial, and potentially useful abstract information from large collections of spatio-temporal data are referred to as spatio-temporal data mining. There are two classes of spatio-temporal databases. The first category includes timestamped sequences of measurements generated by sensors distributed in a map, and temporal evolutions of thematic maps (e.g., weather maps). The second class are moving object databases that consist of object trajectories (e.g., movements of cars in a city). A trajectory can be modeled as a sequence of (p_i, t_i) pairs, where p_i corresponds to a spatial location and t_i is a timestamp. The management and analysis of spatio-temporal data has gained interest recently, mainly due to the rapid advancements in telecommunications (e.g., GPS, Cellular networks, etc.), which facilitate the collection of large datasets of object locations (e.g., cars, mobile phone users) and measurement sequences (e.g., sensor readings). Mining tasks for moving object databases include detection and prediction of traffic jams, analyzing the movement behavior of animals, clustering or classification of moving objects according to their direction and/or speed, and identification of trends that associate the movement/speed of objects to their destination. In addition, from databases of measurement sequences, spatial relationships between correlated or anticorrelated sequences can be extracted (e.g., "sensors within 10 m from each other produce similar readings with high probability"), or build classification models to detect abnormal combinations of sensor readings. The analysis of spatio-temporal databases is challenging due to the vast amount of collected data, and the complexity of novel mining tasks. Special issues include the fuzzy and implicit nature of spatio-temporal relationships between objects, the complex geometry of spatial objects, the varying temporal nature of events (instantaneous vs. durable), the variability of spatio-temporal data (moving objects, evolution of spatial events or phenomena, etc.), and the multiple (spatial and temporal) resolution levels of abstraction.

Historical Background

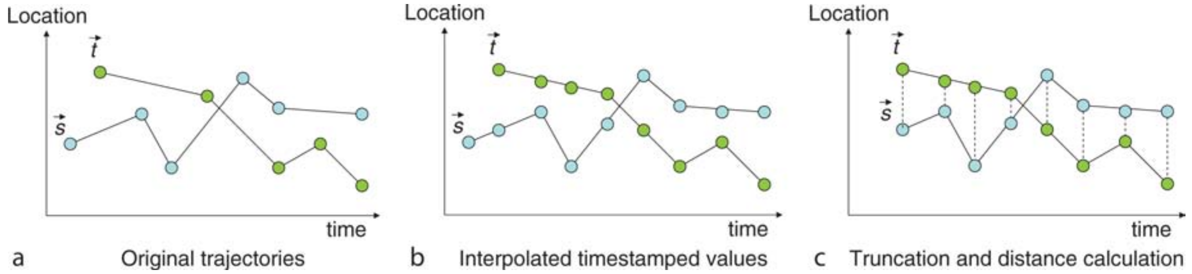
Data mining became a core field of database research in the 1990s [6]. Initial research focused on mining tasks (association analysis, classification, and clustering) applied on relational databases, or transactional, data that record sets of items purchased together. Two parallel streams of research were born soon after the first papers; data mining for temporal and spatial data. Temporal data mining focused on the extraction of sequential patterns from ordered transactional data or event sequences. Clustering and classification of time series, a classic problem in statistics, also triggered the interest of this research stream. The spatial relationships between associated events has been the main focus of spatial data mining. Spatio-temporal data mining is a rather new research field. Initially [4,11], temporal data mining techniques were applied for spatio-temporal data, after modeling the input as multi-dimensional temporal sequences. Lately, new problems, particular to this type of data have emerged, such as clustering multidimensional trajectories [12], clustering or pattern mining based on common subtrajectories [2,8] mining periodic patterns in moving object trajectories [9] and discovery of moving clusters with partial membership of objects during a cluster's lifetime [7].

Foundations

Clustering

Clustering is a classic data mining task that divides a set of objects into groups (clusters), such that the objects in the same cluster are similar to each other and objects in different clusters are dissimilar. Most clustering prototypes [6] can be applied for spatio-temporal data after (i) the objects to be clustered are well-defined, and (ii) a distance function between objects has been determined. The first prototype is partitioning-based clustering, where a set of k initial random partitions are iteratively refined. Characteristic algorithms in this class are the k-means and k-medoids algorithms. Hierarchical methods is another category of algorithms, where initially each object forms a cluster on its own and clusters are merged iteratively until a convergence criterion is met. Another popular clustering prototype is density-based clustering, where dense regions of nearby objects are iteratively merged. These general methods will not be discussed in detail here; instead, the focus will be on similarity measures for spatio-temporal data and on special definitions of clustering in this context.

Clustering can be performed on trajectories in order to classify moving objects into groups of similar movement behavior. The distance between two trajectories can be defined after modeling them as high-dimensional vectors, by some preprocessing if necessary. More specifically, if the timestamps of trajectories do not match, any missing timestamped values are generated by interpolation, such that the transformed trajectories correspond to vectors of the same length and they have the same timestamps (as regular as possible). In addition, if the sequences have different timespan they are shrunk (or extended) as necessary by truncation (or interpolation). Eventually, the distance between the resulting trajectories is measured using some appropriate distance function (e.g., Euclidean distance). Figure 1 illustrates this process. Apart from the Euclidean distance, other measures specific to (multidimensional) time series have been proposed. A popular distance measure is dynamic time warping (DTW) [1]. DTW allows elastic shifting of sequence in order to detect similar shapes with different phases. DTW aligns each element of one sequence to one or more elements of the other sequence by applying a dynamic programming process, similar to the edit distance computation between strings. Another method to assess the similarity between two time series is to find their longest common subsequence (LCSS) [3]. One definition of the LCSS between two vectors $\vec{s}$ and $\vec{t}$ is the pair of subsequences $\vec{s}' \in \vec{s}$ and $\vec{t}' \in \vec{t}$, such that (i) $\vec{s}'$ and $\vec{t}'$ have the same length, (ii) $dist(\vec{s}', \vec{t}') \leq \epsilon$, and (iii) $(\vec{s}', \vec{t}')$ are the longest subsequences that qualify the distance constraint



a Original trajectories

b Interpolated timestamped values

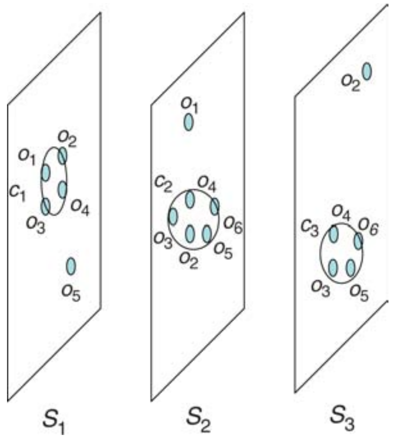
c Truncation and distance calculation

Spatio-Temporal Data Mining. Figure 1. Measuring distance between trajectories.

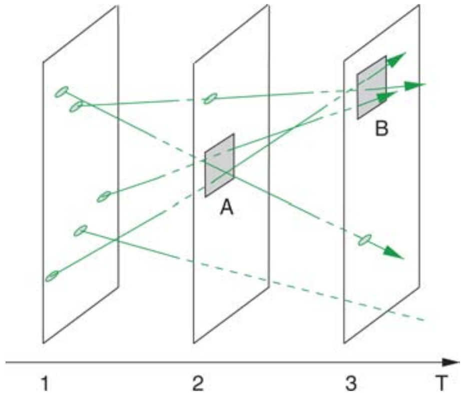
expressed in (ii). Here, *dist*() is a basis measure between sequences (e.g., Euclidean distance) and ε is a threshold quantizing adequate closeness. LCSS (like DTW) allows stretching of sequences in time (provided that a DTW-like measure is used as measure), and at the same time is more robust to noise, giving weight to the similar portions of the sequences and ignoring the very different parts. In [12] an appropriate definition of LCSS is given for multidimensional time series (i.e., trajectories), where a constraint for the maximum shift in time between common subsequences exists, and different stretching/shifting at different dimensions can also be performed in order to reach the best possible matching.

Instead of applying classic clustering algorithms with the help of an appropriate distance measure, [4] define clusters of trajectories by mixtures of regression models, which are extracted with the help of an EM algorithm. In a recent work, [8] clusters trajectories, after partitioning them to line segments. The clusters in this case are formed by similar line segments, and a trajectory may belong to more than one cluster.

A special definition of moving clusters is presented in [7]. The difference compared to the trajectory clusters defined above is that the identity of a moving cluster remains unchanged, while its location and content may change over time. For example, while a group of animals are migrating, some new animals may enter the group (e.g., those passing nearby the clusters trajectory or newborns), while some animals may leave the group (e.g., those attacked and eaten by lions). Formally, consider a set of moving objects in a long, timestamped history $H = \{t_1, t_2, ..., t_n\}$. A *snapshot* S_i of H is the set of objects and their locations at time t_i. Given a snapshot S_i, a standard spatial clustering algorithm can be employed to identify dense groups of objects in S_i which are close to each other. Let c_i and c_{i+1} be two such *snapshot clusters* for S_i and S_{i+1}, respectively. Then $c_i c_{i+1}$ is said to be a *moving* cluster if $\frac{|c_i \cap c_{i+1}|}{|c_i \cup c_{i+1}|} \geq \theta$, where θ $(0 < \theta \leq 1)$ is an integrity threshold for the contents of the two clusters. Intuitively, if two spatial clusters at two consecutive snapshots have a large percentage of common objects, then these are considered as a single cluster that moved between these two timestamps. Figure 2 shows an example of a moving cluster. S_1, S_2, and S_2 are three snapshots. In each of them there is a timeslice cluster (c_1, c_2, and c_3). Let $\theta = 0.5$. $c_1 c_2 c_3$ is a moving cluster, since $\frac{|c_1 \cap c_2|}{|c_1 \cup c_2|} = \frac{3}{6}$ and $\frac{|c_2 \cap c_3|}{|c_2 \cup c_3|} = \frac{4}{5}$ are both at least θ. Note that objects may enter or leave the moving cluster during its lifetime. Using this definition, Kalnis et al. [7] extended a density-based clustering algorithm into methods that discover moving clusters in a large database of moving object trajectories.



Spatio-Temporal Data Mining. Figure 2. Example of a moving cluster.



Spatio-Temporal Data Mining. Figure 3. Identifying areas of high object density.

Hadjieleftheriou et al. [5] proposed a framework for the discovery of areas with a high density of moving objects in the future, given the current locations and movements of a set of points, e.g., "Find all regions that will contain more than 500 objects, 10 min from now." The time interval during which the areas remain dense is also part of the mining task. Figure 3 (taken from [5]) shows a graphical example of such dense areas (A in timestamp 2 and B in timestamp 3), assuming that at least three objects must exist in an area of one square unit for the area to be considered dense.

The discovery of dense regions is done after defining a space-time 3D grid and merging neighboring dense cells in this grid. Note that this problem is different to classic clustering, where objects are grouped based on the whole history (or trend) of their movement.

Classification and Prediction

Classification of trajectories is usually performed by nearest neighbor (NN) classifiers [12]. Given a trajectory $\vec{s}$ of unknown label and a database D of labeled samples, such a classifier (i) searches in D for the k most similar time series to $\vec{s}$ and (ii) gives $\vec{s}$ the most popular label in the set of k returned time series. NN classifiers, like clustering algorithms, rely on an appropriate similarity function between trajectories. Depending on the application, one of the distance functions used for clustering (as discussed above) can be used.

A related task to classification is predicting the future movement of an object given its past locations. Regression models for one-dimensional time series can be extended for (multidimensional) moving object trajectories. The movement of objects is approximated by (mixtures of) functions, which in turn are used for prediction. Given the recent past movement of an object [10] propose a methodology that computes a recursive motion function; a concise form that captures a large number of movement types (e.g., polynomials, ellipses, sinusoids, etc.). A recursive function differs from classic regression functions in that it relates an objects location to those of the recent past.

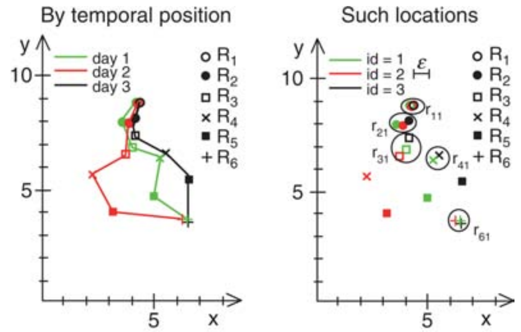
Pattern Extraction

There is limited work on extraction of patterns from spatio-temporal databases, which has been treated as a generalization of pattern mining in time series data. For example, [11] studied the discovery of frequent patterns related to changes of natural phenomena (e.g., temperature changes) in spatial regions. The locations of objects or the changes of natural phenomena over time are converted to categorical values. For instance, the map can be divided into spatial regions and replace the location of the object at each timestamp, by the region-id where it is located. Similarly, the change of temperature in a spatial region can be modeled as a sequence of temperature values. Continuous domains of the resulting time series data are discretized, prior to mining. In the case of multiple moving objects (or time series), trajectories are typically concatenated to a single long sequence. Then, an algorithm that discovers frequent subsequences in a long sequence (e.g., [13]) is applied.

In many applications, the movements obey periodic patterns; i.e., the objects follow the same routes (approximately) over regular time intervals. Objects that follow approximate periodic patterns include transportation vehicles (buses, boats, airplanes, trains, etc.), animal movements, mobile phone users, etc. For example, Bob wakes up at the same time and then follows, more or less, the same route to his work everyday. Periodic patterns can be thought of as (possibly non-contiguous) sequences of object locations that reappear in the movement history periodically.

Formally, let S be a sequence of n spatial locations $\{l_0,l_1,...,l_{n-1}\}$, representing the movement of an object (e.g., Bob) over a long history. Let $T \ll n$ be an integer called *period* (e.g., day, week, month). A *periodic segment s* is defined by a subsequence $l_i l_{i+1} ... l_{i+T-1}$ of S, such that i modulo $T = 0$. Thus, segments start at positions $0, T, ..., (\lfloor \frac{n}{T} \rfloor - 1) \cdot T$, and there are exactly $m = \lfloor \frac{n}{T} \rfloor$ periodic segments in S. A *periodic pattern P* is defined by a sequence $r_0 r_1 ... r_{T-1}$ of length T, such that r_i is either a spatial region or $*$. The *length* of a periodic pattern P is the number of non-$*$ regions in P. A segment s^j is said to *comply with* P, if for each $r_i \in P$, $r_i = *$ or s_i^j is *inside* region r_i. The *support* of a pattern P in S is defined by the number of periodic segments in S that comply with P. The same symbol P is used to refer to a pattern and the set of segments that comply with it. Let $min\_sup \leq m$ be a positive integer (*minimum support*). A pattern P is *frequent*, if its support is larger than min_sup. Patterns for which the regions r_i are too sparse are not interesting, therefore an constraint is imposed to the density of these regions. Let S^P be the set of segments that comply with a pattern P. Then each region r_i of P is *valid* if the set of locations $R_i^P := \{s_i^j \mid s^j \in S^P\}$ form a *dense cluster*.

The discovery of partial periodic patterns from a long trajectory has been studied in [9]. This process is performed in two phases. First, S is divided into T spatial datasets, *one for each offset* of the period T. Specifically, locations $\{l_i, l_{i+T}, ..., l_{i+(m-1) \cdot T}\}$ go to set R_i, for each $0 \leq i < T$ (m is the length of S). A spatial clustering algorithm is applied to discover clusters in each R_i. These clusters define periodic patterns of length 1. Figure 4 a shows the spatial datasets obtained after



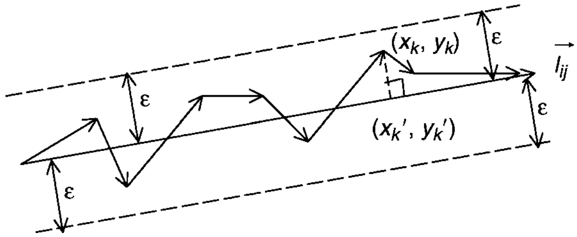
a *T*-based decomposition b Dense clusters in R_i's

Spatio-Temporal Data Mining. Figure 4. Locations and regions per periodic offset.

decomposing the trajectory of an object in three consecutive days (periods). A different symbol is used to denote locations that correspond to different periodic offsets and different colors are used for different segment-ids. Observe that a dense cluster r in dataset R_i corresponds to a frequent pattern, having * at all positions and r at position i. Figure 4b shows examples of five clusters discovered in datasets R_1, R_2, R_3, R_4, and R_6. These correspond to five 1-patterns (i.e., $r_{11}{}^{*****}$, $^*r_{21}{}^{****}$, etc.).

In the second phase, [9] extend the Apriori algorithm [6] to identify longer patterns level-by-level. In specific, Pairs $\langle P_1, P_2 \rangle$ of frequent $(k-1)$-patterns with their first $k-2$ non-* regions in the same position and different $(k-1)$-th non-* position create candidate k-patterns, the supports of which are counted at the next pass of the data sequence.

There has also been research on spatio-temporal pattern mining, where trajectories are regarded as sequences of locations (without giving any importance to the timestamps). In this case, the objective is to extract route-patterns of moving objects irrespectively to their speed. In this spirit, [2] define spatio-temporal patterns as sequences of line segments that form frequently followed routes by moving objects. A line simplification algorithm is used to approximate subsequences of trajectories by line segments. Figure 5 illustrates a subsequence s_{ij} which is approximated by a line segment $\vec{l}_{ij}$, such that the maximum distance of any point from s_{ij} to its projection on $\vec{l}_{ij}$ is at most ε. Simplified line segments of subsequences are clustered to form spatial regions (pattern elements) that can approximate a large number of subsequences. If a region approximates more than *min_sup* subsequences, then it forms a frequent movement pattern of length 1. A generalized frequent movement pattern is an m-length ordered sequence of pattern elements that is supported by (i.e., approximates) more than *min_sup* subsequences. The enumeration of frequent patterns is performed in two phases; first the frequent 1-patterns are identified with the help of the clustering algorithm that is based on line simplification; then, longer patterns are found by employing a substring tree which compresses overlapping sequences of pattern elements.



Spatio-Temporal Data Mining. Figure 5. Subsequence approximation.

Key Applications

Traffic Analysis

A motivating application of spatio-temporal data mining is to predict and analyze the causalities of traffic phenomena. Clustering or pattern extraction can help towards this purpose, since common routes that pass through the same map locations are likely the cause of traffic. Analyzing the causes of such clusters can lead to better transportation design for a city map.

Studying the Movement Behavior of Animals

With the help of GPS technology, the movements of animals can be tracked and analyzed. Identifying clusters or movement patterns can help in understanding the behavior of animals, such as the formulation and maintenance of herds, motion trends based on weather conditions, etc.

Video Analysis

The identification of objects and their movement behavior in video scenes is also an important application of spatio-temporal data mining. In this case, patterns of movement behavior can be extracted and analyzed.

For example, from a soccer game, one can extract the movement style of players. Or, from a martial arts video, one can analyze movement sequences of body parts and relate them to the objective of the subject.

Cross-references

▶ Data Mining
▶ Geometric Stream Mining
▶ Spatial and Spatio-Temporal Data Models and Languages
▶ Spatial Data Mining
▶ Spatio-Temporal Data Warehouses
▶ Spatio-Temporal Trajectories
▶ Temporal Data Mining

Recommended Reading

1. Berndt D. and Clifford J. Using dynamic time warping to find patterns in time series. In Proc. KDD Workshop, 1994.
2. Cao H., Mamoulis N., and Cheung D.W. Mining frequent spatio-temporal sequential patterns. In Proc. 2005 IEEE Int. Conf. on Data Mining, 2005, pp. 82–89.
3. Das G., Gunopulos D., and Mannila H. Finding similar time series. In Advances in Knowledge Discovery and Data Mining, 1st Pacific-Asia Conf., 1997, pp. 88–100.
4. Gaffney S. and Smyth P. Trajectory clustering with mixtures of regression models. In Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 63–72.
5. Hadjieleftheriou M., Kollios G., Gunopulos D., and Tsotras V.J. On-line discovery of dense areas in spatio-temporal databases. In Proc. 8th Int. Symp. Advances in Spatial and Temporal Databases, 2003, pp. 306–324.
6. Han J. and Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann, 2000.
7. Kalnis P., Mamoulis N., and Bakiras S. On discovering moving clusters in spatio-temporal data. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005, pp. 364–381.
8. Lee J.-G., Han J., and Whang K.-Y. Trajectory clustering: a partition-and-group framework. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 593–604.
9. Mamoulis N., Cao H., Kollios G., Hadjieleftheriou M., Tao Y., and Cheung D.W. Mining, indexing, and querying historical spatiotemporal data. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 236–245.
10. Tao Y., Faloutsos C., Papadias D., and Liu B. Prediction and indexing of moving objects with unknown motion patterns. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 611–622.
11. Tsoukatos I. and Gunopulos D. Efficient mining of spatiotemporal patterns. In Proc. 7th Int. Symp. Advances in Spatial and Temporal Databases, 2001, pp. 425–442.
12. Vlachos M., Gunopulos D., and Kollios G. Discovering similar multidimensional trajectories. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 673–684.
13. Zaki M.J. Spade: an efficient algorithm for mining frequent sequences. Machine Learning, 42(1/2):31–60, 2001.

Spatio-Temporal Data Reduction

▶ Compression of Mobile Location Data

Spatio-Temporal Data Types

RALF HARTMUT GÜTING
University of Hagen, Hagen, Germany

Synonyms

Data Types for Moving Objects

Definition

Abstract data types to represent time dependent geometries, in particular continuously changing geometries, or *moving objects*. The most important types are *moving point* and *moving region*.

Key Points

A *moving point* represents an entity for which only the time dependent position is of interest. A *moving region* describes an entity for which the time dependent location as well as the shape and extent are relevant. For example, moving points could represent people, vehicles such as cars, trucks, ships or planes, or animals; moving regions could be hurricanes, forest fires, spread of epidemic diseases etc. Moving point data may be captured by GPS devices or RFID tags; moving region data may result from processing sequences of satellite images, for example. Geometrically, moving points or moving regions exist in a 3D (2D + time) space, if the movement is modeled within the 2D plane; for moving points this can be easily extended to 4D (3D + time).

Beyond the most relevant types of moving point and moving region, to obtain a closed system there are related time dependent data types, such as real-valued functions or time dependent boolean values. To have a uniform terminology these types are also called *moving real* and *moving bool*, respectively. Static spatial data types such as *point*, *line* or *region*, and standard data types are also needed. The data types include suitable operations such as:

trajectory: *mpoint* → *line*	Projection of a moving point into the plane
inside: *mpoint* × *mregion* → *mbool*	When is a moving point inside a moving region
distance: *mpoint* × *point* → *mreal*	Distance between a moving and a static point

Cross-references

▶ Moving Objects Databases and Tracking

Recommeded Reading

1. Erwig M., Güting R.H., Schneider M., and Vazirgiannis M. Spatio-temporal data types: an approach to modeling and querying moving objects in databases. GeoInformatica, 3:265–291, 1999.

Spatio-Temporal Data Warehouses

Yufei Tao[1], Dimitris Papadias[2]
[1]Chinese University of Hong Kong, Hong Kong, China
[2]Hong Kong University of Science and Technology, Hong Kong, China

Synonyms

Spatio-temporal online analytical processing; Spatio-Temporal OLAP

Definition

Consider N regions R_1, R_2,...,R_N and a time axis consisting of discrete timestamps 1, 2,...,T, where T represents the total number of recorded timestamps (i.e., the length of history). The position and area of a region R_i may vary along with time, and its extent at timestamp t is denoted as $R_i(t)$. Each region carries a set of *measures* $R_i(t).ms$, also called the *aggregate data* of $R_i(t)$. The measures of regions change asynchronously with their extents. In other words, the measure of R_i ($1 \leq i \leq N$) may change at a timestamp t (i.e., $R_i(t).ms \neq R_i(t-1).ms$), while its extent remains the same (i.e., $R_i(t) = R_i(t-1)$), and vice versa.

A *spatio-temporal data warehouse* stores the above information, and efficiently answers the *spatio-temporal window aggregate* query, which specifies an area q_R and a time interval q_T of continuous timestamps. The goal is to return the aggregated measure $Agg(q_R, q_T, f_{agg})$ of all regions that intersect q_R during q_T, according to some distributive aggregation function f_{agg}, or formally:

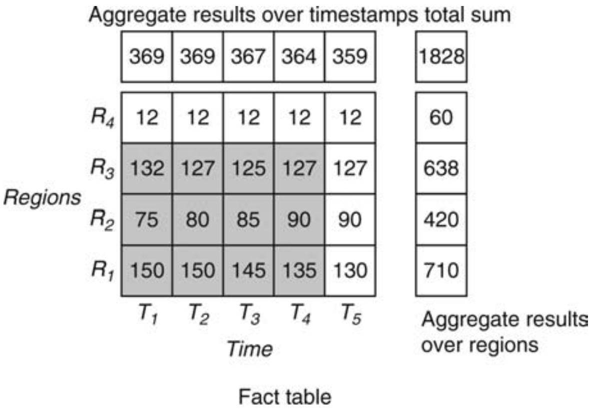
$$Agg(q_R, q_T, f_{agg}) = f_{agg}\{R_i(t).ms \\ |R_i(t) \text{ intersects } q_R \textit{ and } t \in q_T\}.$$

If q_T involves a single timestamp, the query is a *timestamp query*; otherwise, it is an *interval query*. An example of a timestamp window aggregate query is "find the total number of mobile users in the city center at 12 P.M." The query will summarize the number of users (measures) in all regions intersecting q_R = "city center" at q_T = "12 P.M."

Historical Background

The motivation behind spatio-temporal data warehouses is that many spatio-temporal applications require summarized results, rather than information about individual objects. As an example, traffic supervision systems monitor the number of cars in an area of interest, instead of their ids. Similarly, mobile phone companies use the number of phone-calls per cell in order to identify trends and prevent potential network congestion. Although summarized results can be obtained using conventional operations on individual objects (i.e., accessing every single record qualifying the query), the ability to manipulate aggregate information *directly* is imperative in spatio-temporal databases due to several reasons. First, in some cases personal data should not be stored due to legal issues. For instance, keeping historical locations of mobile phone users may violate their privacy. Second, the individual data may be irrelevant or unavailable, as in the traffic supervision system mentioned above. Third, although individual data may be highly volatile and involve extreme space requirements, the aggregate information usually remains fairly constant for long periods, thus requiring considerably less space for storage.

A considerable amount of related research has been carried out on *data warehouses* and *OLAP* (on line analytical processing) in the context of relational databases. The most common conceptual model for data warehouses is the multi-dimensional data view. In this model, each *measure* depends on a set of *dimensions*, e.g., *region* and *time*, and thus is a value in the multi-dimensional space. A dimension is described by a


Aggregate results over timestamps total sum
Regions
Time
Aggregate results over regions
Fact table

	T_1	T_2	T_3	T_4	T_5	
	369	369	367	364	359	1828
R_4	12	12	12	12	12	60
R_3	132	127	125	127	127	638
R_2	75	80	85	90	90	420
R_1	150	150	145	135	130	710

Spatio-Temporal Data Warehouses. Figure 1. A data cube example.

domain of values (e.g., days), which may be related via a hierarchy (e.g., day-month-year). Figure 1 illustrates a simple case, where each cell denotes the measure of a region at a certain timestamp. Observe that although regions are 2-dimensional, they are mapped as one dimension in the warehouse.

The *star schema* [6] is a common way to map a data warehouse onto a relational database. A main table (called *fact table*) F stores the multi-dimensional array of measures, while auxiliary tables $D_1, D_2,...,D_n$ store the details of the dimensions. A tuple in F has the form $<D_i[].key, M[]>$ where $D_i[].key$ is the set of foreign keys to the dimension tables and $M[]$ is the set of measures. OLAP operations ask for a set of tuples in F, or for aggregates on groupings of tuples. Assuming that there is no hierarchy in the dimensions of the previous example, the possible groupings in Fig. 1 include: (i) group-by Region and Time, which is identical to F, (ii)-(iii) group-by Region (Time), which corresponds to the projection of F on the *region-* (*time-*) axis, and (iv) the aggregation over all values of F which is the projection on the origin (Fig. 1 depicts these groupings for the aggregation function *sum*). The fact table together with all possible combinations of group-bys composes the *data cube* [2]. Although all groupings can be derived from F, in order to accelerate query processing some results may be precomputed and stored as *materialized views*.

A detailed group-by query can be used to answer more abstract aggregates. In the example of Fig. 1, the total measure of all regions for all timestamps (i.e., 1828) can be computed either from the fact table, or by summing the projected results on the *time* or *region* axis. Ideally, the whole data cube should be materialized to enable efficient query processing. Materializing all possible results may be prohibitive in practice as there are $O(2^n)$ group-by combinations for a data warehouse with n dimensional attributes. Therefore, several techniques have been proposed for the view selection problem in OLAP applications [1,4]. In addition to relational databases, data warehouse techniques have also been applied to spatial [3,10] and temporal [8] databases. All these methods, however, benefit only queries on a predefined hierarchy. An ad-hoc query not confined by the hierarchy, such as the one involving the gray cells in Fig. 1, would still need to access the fact table, even if the entire data cube were materialized.

Foundations

The next discussion describes several solutions to implementing a spatio-temporal data warehouse, assuming summation as the underlying aggregate function f_{agg}. Extensions to other aggregation functions (e.g., count, average) are straightforward.

- *Using a 3D aggregate R-tree*

The problem of a spatio-temporal window aggregate search can be regarded as a multi-dimensional aggregate retrieval in the 3D space (the spatial dimensions plus a time dimension) and solved using an aggregate R-tree (aR-tree). The aR-tree [5,9] is similar to a conventional *R-tree*, where each node also stores summarized information about the regions in each sub-tree. Whenever the extent or measure of a region changes, a new 3D box is inserted in a 3D version of the aR-tree, called the *a3DR-tree*. Using the example of Fig. 1, four entries are required for R_1: one for timestamps 1 and 2 (when its measure remains 150) and three more entries for the other timestamps. A spatio-temporal window aggregate query can also be modeled as a 3D box, which can be processed on the a3DR-tree, following the strategy of solving a range aggregate query on an aR-tree [5,9].

The problem with this solution is that it creates a new box duplicating the region's extent, even though it does not change. Since the measure changes are much more frequent than extent updates, the *a3DR-tree* incurs high redundancy. The worst case occurs for static regions: although the extent of a region remains constant, it is still duplicated at the rate of its measure changes. Bundling the extent and aggregate information in all entries significantly lowers the node fanout and compromises query efficiency, because more nodes must be accessed to retrieve the same amount of information. Note that redundancy incurs *whenever the extent and measure changes are asynchronous*, i.e., the above problem also exists when a new box is spawned because of an extent update, in which case the region's measure must be replicated.

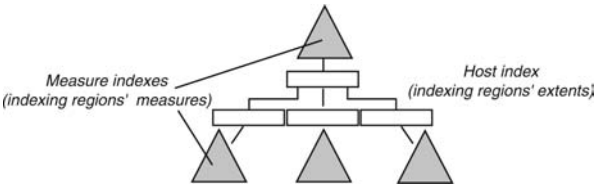
- *Using a data cube*

Following the traditional data warehouse approach, it is possible to create a data cube, where one axis corresponds to time, the other to regions, and keep the measure values in the cells of this two-dimensional table (see Fig. 1). Since the spatial dimension has no one-dimensional order, the table can be stored in the secondary memory ordered by time, and a B-tree index can be created to locate the pages containing information about each timestamp. The processing of a query employs the B-tree index to retrieve the pages (i.e., table columns) containing information about q_T; then, these regions (qualifying the temporal condition) are scanned sequentially and the measures of those satisfying q_R are aggregated.

Even if an additional spatial index on the regions exists, the simultaneous employment of both indexes has limited effect. Assume that first a window query q_R is performed on the spatial index to provide a set of ids for regions that qualify the spatial condition. Measures of these regions must still be retrieved from the columns corresponding to q_T (which, again, are found through the B-tree index). However, the column storage does not preserve spatial proximity, and hence the spatially qualifying regions are expected to be scattered in different pages. Therefore, the spatial index has some effect only on very selective queries (on the spatial conditions). Furthermore, recall that pre-materialization is useless, since the query parameters q_R and q_T do not conform to pre-defined groupings.

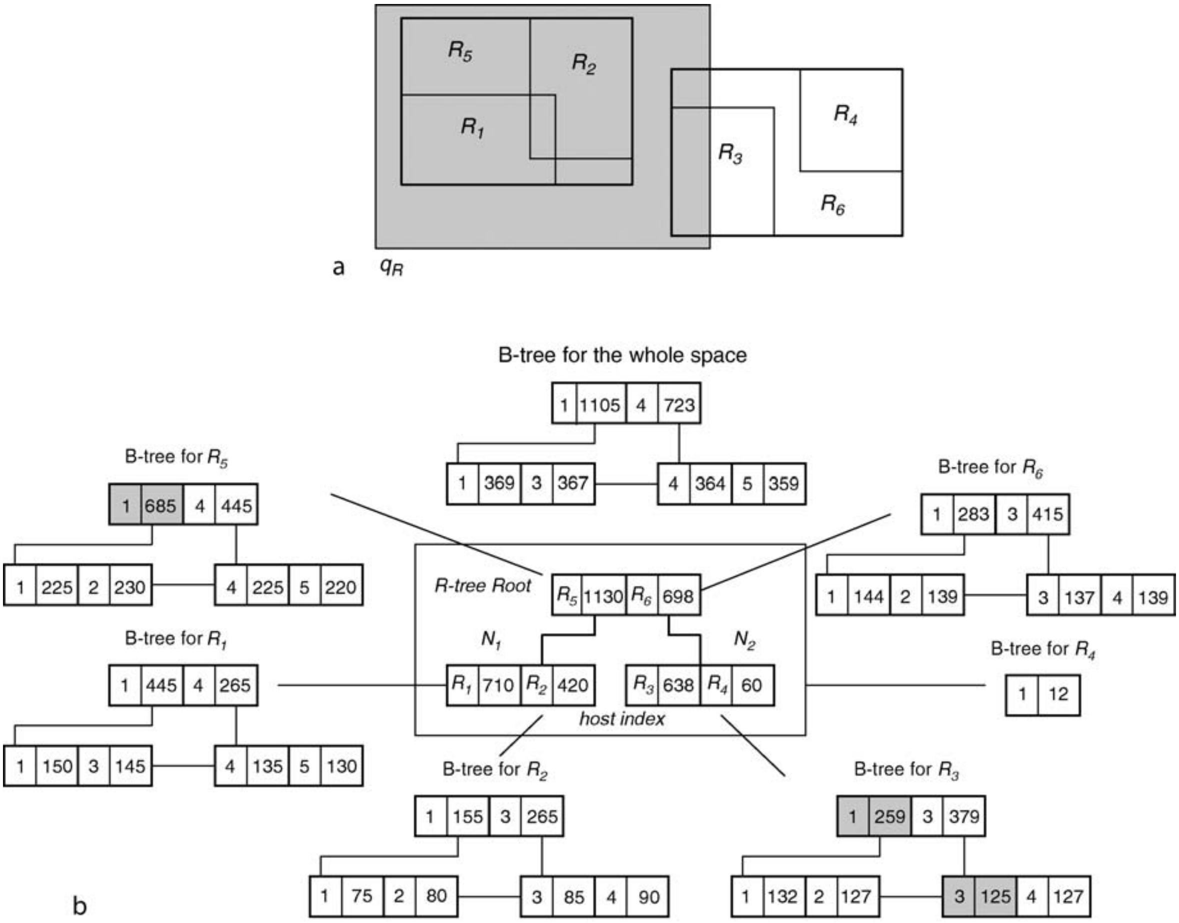
- *The aggregate R-B-tree*

Since (i) the extent and measure updates are asynchronous and (ii) in practice, measures change much more frequently than extents (which may even be static), the two types of updates should be managed independently to avoid redundancy. This implies the deployment of two types of indexes: (i) a *host index*, which is an aggregate spatial or spatio-temporal structure managing region extents, and (ii) numerous *measure indexes* (one for each entry of the host index), which are aggregate temporal structures storing the values of measures during the history. Figure 2 shows a general overview of the architecture [14]. Given a query, the host index is first searched, identifying the set of entries that qualify the spatial condition. The measure indexes of these entries are then accessed to retrieve the timestamps qualifying the temporal conditions. Since the number of records (corresponding to extent changes) in the host index is very small compared to the measure changes, the cost of query processing is expected to be low.



Spatio-Temporal Data Warehouses. Figure 2. A multi-index architecture for indexing spatio-temporal data warehouses.

The following discussion explains an instantiation of the architecture for solving window aggregate queries when the underlying data regions are static. The instantiation leads to a structure, called the *aggregate R- B-tree* (*aRB-tree*). It adopts an aR-tree as the host index, where an entry *r* has the form <*r*.MBR, *r*.aggr, *r*.pointer, *r*.btree >; *r*.MBR and *r*.pointer have the same semantics as a normal R-tree, *r*.aggr keeps the aggregated measure about *r* over the *entire history*, and *r*.btree points to an aggregate *B-tree* which stores the detailed measure information of *r* at concrete timestamps. Figure 3b illustrates an example using the data regions of Fig. 3a and the measures of Fig. 1. The number 710 stored with R-tree entry R_1, equals the sum of measures in R_1 for all 5 timestamps (e.g., the total number of phone calls initiated at R_1). The first leaf entry of the B-tree for R_1 (1, 150) indicates that the measure of R_1 at timestamp 1 is 150. Since the measure of R_1 at timestamp 2 is the same, there is no a special entry, but this knowledge is implied from the previous entry (1, 150). Similarly, the first root entry (1, 445) of the same B-tree indicates that the aggregated measure in R_1 during time interval [1,3] is 445. The topmost B-tree stores aggregated information about the whole space, and its role is to answer queries involving only temporal conditions (similar to that of the extra row in Fig. 1).



Spatio-Temporal Data Warehouses. Figure 3. A solution to static regions.

To illustrate the processing algorithms, consider the query "find the number of phone-calls initiated during interval $q_T = [1,3]$ in all cells intersecting the window q_R shown in Fig. 3a." Starting from the root of the R-tree, the algorithm visits the B-tree of R_5 since the entry is totally contained in q_R. The root of this B-tree has entries (1,685), (4,445) meaning that the aggregated measures (of all data regions covered by R_5) during intervals [1,3], [4,5] are 685 and 445, respectively. Hence, the contribution of R_5 to the query result is 685. The second root entry R_6 of the R-tree partially overlaps q_R, so its child node is visited, where only entry R_3 intersects q_R, and thus its B-tree is retrieved. The first entry of the root (of the B-tree) suggests that the contribution of R_3 for the interval [1,2] is 259. In order to complete the result, the algorithm will have to descend the second entry and retrieve the measure of R_3 at timestamp 3 (i.e., 125). The final result equals 685 + 259 + 125, which corresponds to the sum of measures in the gray cells of Fig. 3b.

Key Applications

Traffic Control

Traffic control systems require summarized information about areas of interest instead of the concrete vehicle ids. Furthermore, in most cases, approximate aggregation [11] is sufficient for tasks such as traffic jam detection and shortest path computation.

Mobile Computing

Measures such as the number of phone-calls per cell can help identify trends, prevent potential network congestion and achieve load balancing in mobile computing applications.

Sensor Systems

Spatio-temporal data warehouses can collect and store readings from geographically distributed sensors. As an example consider a pollution monitoring system, where the readings from several sensors are fed into a warehouse that arranges them in regions of similar or identical values. These regions should then be indexed for the efficient processing of queries such as "find the areas near the center with the highest pollution levels yesterday."

Future Directions

Tao and Papadias [13] discuss spatial aggregation techniques, i.e., when both the regions and their measures are static. Tao et al. [12] present a sketch-based aggregation technique that avoids counting the same object twice (*distinct counting problem*) during the computation of aggregates. A survey of spatio-temporal aggregation techniques can be found in [7].

Experimental Results

[14] contains an extensive set of spatio-temporal aggregation techniques for static and dynamic regions, and experimental (as well as analytical) comparisons.

Data Sets

Common benchmark datasets can be found at:

www.rtreeportal.org

Cross-references

► B+-Tree
► Data Warehouse
► On-Line Analytical Processing
► R-Tree (and Family)
► Star Schema

Recommended Reading

1. Baralis E., Paraboschi S., and Teniente E. Materialized view selection in a multidimensional database. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 156–165.
2. Gray J., Bosworth A., Layman A., and Pirahesh H. Data cube: a relational aggregation operator generalizing group-by, cross-tabs and subtotals. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 152–159.
3. Han J., Stefanovic N., and Koperski K. Selective materialization: an efficient method for spatial data cube construction. In Proc. Pacific-Asia Conf. on Knowledge Discovery and Data Mining, 1998, pp. 144–158.
4. Harinarayan V., Rajaraman A., and Ullman J. Implementing data cubes efficiently. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 205–216.
5. Jurgens M. and Lenz H. The Ra*-tree: an improved R-tree with materialized data for supporting range queries on OLAP-data. In Proc. Int. Workshop on Database and Expert Systems Applications, 1998, pp. 186–191.
6. Kimball R. The Data Warehouse Toolkit. John Wiley, New York, NY, 1996.
7. Lopez I., Snodgrass R., and Moon B. Spatiotemporal aggregate computation: a survey. IEEE Trans. Knowl. and Data Eng., 17(2):271–286, 2005.
8. Mendelzon A. and Vaisman A. Temporal queries in OLAP. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 242–253.
9. Papadias D., Kalnis P., Zhang J., and Tao Y. Efficient OLAP operations in spatial data warehouses. In Proc. 7th Int. Symp., Advances in Spatial and Temporal Databases, 2001, pp. 443–459.
10. Stefanovic N., Han J., and Koperski K. Object-based selective materialization for efficient implementation of spatial data cubes. IEEE Trans. Knowl. and Data Eng., 12(6):938–958, 2000.
11. Sun J., Papadias D., Tao Y., and Liu B. Querying about the past, the present and the future in spatio-temporal databases. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 202–213.
12. Tao Y., Kollios G., Considine J., Li F., and Papadias D. Spatio-temporal aggregation using sketches. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 214–225.
13. Tao Y. and Papadias D. Range aggregate processing in spatial databases. IEEE Trans. Knowl. and Data Eng., 16(12): 1555–1570, 2004.
14. Tao Y. and Papadias D. Historical spatio-temporal aggregation. ACM Trans. Inf. Syst., 23(1):61–102, 2005.

Spatio-Temporal Databases

► Moving Objects Databases and Tracking

Spatiotemporal Estimation

► Spatiotemporal Interpolation Algorithms

Spatio-Temporal Graphs

► Time Aggregated Graphs

Spatiotemporal Interpolation Algorithms

Peter Revesz
University of Nebraska-Lincoln, Lincoln, NE, USA

Synonyms

Moving objects interpolation; Spatio-Temporal approximation; Spatiotemporal estimation

Definition

Spatiotemporal interpolation is the problem of estimating the unknown values of some property at arbitrary spatial locations and times, using the known values at spatial locations and times where measurements were made. In spatiotemporal interpolation the estimated property varies with both space and time, with the assumption that the values are closer to each other with decreasing spatial and temporal distances.

Spatiotemporal interpolation is used in spatiotemporal databases, which record spatial locations and time instances together with other attributes that are dependent on space and time. For example, a spatiotemporal database may record the sales of houses in a town. The house sales database records the location, usually as the address of the house from which an (x, y) location can be easily found, by correlating the address with a map of the town, the calendar date when the sale occurred, the area and the sale price of the house.

Spatiotemporal interpolation is also used in moving objects databases to estimate the trajectory of moving objects.

Historical Background

Spatiotemporal interpolation is a generalization of spatial interpolation by the addition of a temporal dimension. While spatial interpolation is well-investigated [1], spatiotemporal interpolation in general, including interpolation of the trajectory of moving objects, is a relatively new area [5].

Spatiotemporal interpolation problems were either assumed to be just a sequence of spatial interpolation problems, called the *time slices approach*, or they were assumed to be as easily handled as adding one more spatial dimension, called the *extension approach*. Li and Revesz [10] pointed out problems with the time slices and the extension approaches, and proposed the *spatiotemporal product (ST-product)* approach. Gao and Revesz [2] described several *adaptive spatiotemporal interpolation* approaches that improve the ST-product method.

Moving objects carrying positioning devices, thus recording their positions in arbitrary spatial locations along with their corresponding timestamps, provide data for moving object databases. In moving object databases a variety of spatiotemporal interpolation methods can be applied, e.g., linear interpolation [1], which is the most commonly adopted, and polynomial functions and splines [6,8,11,16].

Foundations

There are several problems with the simple spatiotemporal interpolation approaches, like the time slices approach and the extension approach. Among these problems the following need to be mentioned.

Poor Estimation Accuracy: The time slices approach is not accurate in general. For example, a spatiotemporal interpolation problem is to estimate the price of houses based on sales data from sold houses in a town. To solve this problem the time slices approach would first select just the houses that were sold in one short time slice, for example on a single day or in a single week. Then the time slices approach would do a spatial interpolation on the selected data. The accuracy of this method is often poor, because there is simply not too many houses sold on a single day or in a single week to cover the town dense enough for accurate interpolation. Many town subdivisions may not have a single house sold on just one day.

Lack of Continuous Time: The time slices approach cannot deal with continuous time.

Non-Invariance to Scaling: The extension approach handles a k-dimensional spatial and one dimensional temporal interpolation problem as if it were a k+1-dimensional spatial interpolation problem. Li and Revesz [10] pointed out that the extension approach can also lead to problems because of the *non-invariance to scaling* of many spatial interpolation algorithms. Scaling invariance means that if the unit of measurement changes in one dimension, then the estimated value does not change. While many spatial interpolation algorithms are scaling invariant if the units change in each spatial dimension, few spatial interpolation algorithms are scaling invariant if the units change only in one dimension. For example, the inverse distance weighting (IDW) [14] spatial interpolation method is non-invariant to scaling in only one dimension. Such methods are difficult to use because it is difficult to decide what is the right unit of time given certain units of spatial distance. For example, IDW gives a different house price estimate if the unit of time is days than if the unit of time is weeks, even if all houses are always sold on Mondays. That is a strange phenomenon that makes IDW inherently awkward to use.

The spatiotemporal product (or ST-product) approach [10] improves the times slices method. Using the house price estimation problem, the ST-product approach can be explained as follows. For each house that is sold several times according to the sales spatiotemporal database, the ST-product approach first estimates its price based on a simple linear temporal interpolation between two consecutive sales. For example, if a house was sold for $200,000 on July 29th, 2000, and was sold for $250,000 on July 1st, 2002, then one can estimate that the price of the house rose $500 for each of the 100 weeks between the two sales.

After these temporal interpolations, a spatial interpolation is done for each week like in the time slices method. However, since a large percentage of the values for each week are filled in, the ST-product method uses a considerably greater density of houses in the spatial interpolation part than the time-slices method does. Hence, in general, the ST-product method is more accurate than the time-slices method. Further, the ST-product method can be applied without an exact calculation of the temporal interpolation values by using a temporal parameter t. Then the ST-product method yields an interpolation function of x, y, and t for any (x, y) location and time instance t, even in continuous time.

The spatiotemporal accuracy is increased further by *adoptive spatiotemporal interpolation* [2]. To explain the adoptive method, suppose $R(x, y, t, w)$ is a spatiotemporal relation where (x, y) is the location, t is the time instance, and w is the measured value. The key idea behind the adaptive method is that to interpolate the value w at any location (a, b) at time c, where a,b and c are constants, there are two main choices:

1. *Spatial Projection + 1D temporal interpolation:* Select from R the records with $x = a$ and $y = b$. Then use any 1D temporal interpolation method on the selected records.
2. *Temporal Projection + 2D spatial interpolation:* Select from R the records with $t = c$. Then use any 2D spatial interpolation method on the selected records.

When the spatial and temporal projections of R both contain enough number of records to do an interpolation at location (a, b) and time c, then one could choose either (1) or (2) as an option. These two options may give different interpolation values. Which one of the two estimates is more reliable? The *choice-based adaptive interpolation method* decides that question based on the *relationship strength measures* for space and time. Intuitively, the larger the relationship strength measure for space (or time), the more reliable is the estimation using a spatial (or temporal) interpolation. These measures are denoted by the following symbols:

$\mathcal{S}(a, b, c)$ – the relationship strength measure for space at location (a, b) and time c.
$\mathcal{T}(a, b, c)$ – the relationship strength measure for time at location (a, b) and time c.

Both measures are localized, hence the spatial relationship strength can be larger than the temporal relationship strength at some locations and times, while the reverse may be true at a different location and time within the same problem. The choice-based adaptive method works as follows whenever there is a choice:

$$\begin{cases} \text{Spatial interpolation} & \text{if } \mathcal{S}(\mathrm{a, b, c}) > \mathcal{T}(\mathrm{a, b, c}) \\ \text{Temporal interpolation} & \text{if } \mathcal{S}(\mathrm{a, b, c}) \leq \mathcal{T}(\mathrm{a, b, c}) \end{cases} \tag{1}$$

Both the ST-product and the choice-based adaptive methods first interpolate the missing values for the (a, b) locations for which several measurements at different times are known. While the ST-product method always preferred to do a temporal interpolation, the adaptive method at some spatiotemporal locations will prefer to do a spatial interpolation. This is the only reason why the two interpolation results may disagree. Finally, for both methods the non-measurement locations, that is, the locations that do not appear in the original data set, need to be estimated using some 2D spatial interpolation.

Gao and Revesz [2] suggested several spatial and temporal relationship strength measures. A simple measure is to use the inverse of the variance of the measured values in the spatial neighborhood of (a, b), which is defined as the nearest k other (a, b) locations for some integer k, as the spatial relationship strength. Similarly, one can use the inverse of the variance of the two measures at the earliest time following c, and latest time preceding c, as the temporal relationship strength. As an alternative to the choice-based adaptive interpolation approach, [2] also proposed a linear combination adaptive interpolation approach, which gives a weighted linear sum of the two estimates. The weights depend on the relative magnitudes of the spatial and temporal relationship strength measures.

Trajectory of Moving Objects: The estimation of the trajectory of moving objects can also be viewed as a spatiotemporal interpolation problem. Applying the above described ideas, in each sensor location the moving object can be either sensed with value one or not sensed and assigned a value zero. However, there are more advanced interpolation techniques, which associate each moving object's approximated position with an uncertainty factor [9,12,15].

Key Applications

Spatiotemporal interpolation has a growing number of applications, in areas such as the following:

1. *Real Estate Analysis:* In an experiment with a database of house sales in the town of Lincoln, Nebraska, USA, the ST-product method had less than a 9% mean absolute error in estimating house prices [10]. The surprising feature of the estimation was that nothing special was known about the houses except their locations, sizes in square feet, and prices. The estimation could be easily improved with a visit to the houses to check their conditions. The house price estimation can tell whether some given houses in given years were assessed too high or too low taxes.
2. *Weather Analysis:* Weather is a spatiotemporal phenomenon that shows several cyclical patterns. The ST-product spatiotemporal interpolation method was used for ground water level and drought analysis. Carbon dioxide concentrations over time is another application area.
3. *Epidemiology:* Spatiotemporal interpolation was also used to predict the spread of epidemics, for example, the spread of the West Nile Virus in the continental USA [13].
4. *Forest Fires:* The spread of forest fires can be predicted similar to the spread of epidemics.
5. *Traffic Accident Report:* In a traffic accident report spatiotemporal interpolation may be used to estimate the trajectories of the vehicles that were involved in the accident. An accurate estimation may be important to decide what caused the accident and to get insurance payments.

Future Directions

There are still several problems about the relationship of spatiotemporal interpolation and prediction of spatiotemporal phenomena. Spatiotemporal interpolation seems to work best if one is interested in a location that is in the middle of the space, and in a time that is in the middle of the time interval recorded. Predictions are at future times, hence they use specialized algorithms. For example, voting prediction is a research area in itself and has many specialized techniques different from the spatiotemporal interpolation approaches. For example, when one is trying to predict the outcome of an election in a voting district A, one cannot use the time-slices, the ST-product, and the adoptive interpolation approaches because there are no spatial neighbors in general. There are some exceptions to this rule. For example, if the neighboring voting districts have all already reported their election results while the votes in district A are still being (re)counted, then one can predict the outcome in voting district A using the results of the neighboring districts [3].

While voting problems are almost always prediction problems, on occasion, analysis of past election results may be also interesting, for example, to identify and help settle possible election fraud cases in the past. Predicting other spatiotemporal behaviors beside voting, for example, the number of customers in a town who would buy a certain product is also an important area of interest.

Spatiotemporal databases can be conveniently represented using *constraint databases* [7]. Constraint databases allow convenient handling of different types of interpolation data [4].

Cross-references

- ▶ Constraint Databases
- ▶ Databases
- ▶ Moving Object
- ▶ Moving Objects Databases and Tracking

Recommended Reading

1. Davis P.J. Interpolation and Approximation. Dover, NY, USA, 1975.
2. Gao J. and Revesz P. Adaptive spatiotemporal interpolation methods. In Proc. First Int. Conf. on Geometric Modeling, Visualization, and Graphics, 2005, pp. 1622–1625.
3. Gao J. and Revesz P. Voting prediction using new spatiotemporal interpolation methods. In Proc. 7th Int. Conf. on Digital Government Research, 2006, pp. 293–300.
4. Grumbach S., Rigaux P., and Segoufin L. Manipulating interpolated data is easier than you thought. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 156–165.
5. Güting R. and Schneider M. Moving Objects Databases. Morgan Kaufmann, Los Altos, CA, 2005.
6. Hadjieleftheriou M., Kollios G., Tsotras V.J., and Gunopulos D. Efficient indexing of spatiotemporal objects. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 251–268.
7. Kanellakis P.C., Kuper G.M., and Revesz P. Constraint query languages. J. Comput. Syst. Sci., 51(1):26–52, 1995.
8. Koubarakis M., Sellis T.K., Frank A.U., Grumbach S., Güting R. H., Jensen C.S., Lorentzos N.A., Manolopoulos Y., Nardelli E., Pernici B., Schek H.-J., Scholl M., Theodoulidis B., and Tryfona N. (eds.). Spatio-Temporal Databases: The CHOROCHRONOS Approach, LNCS 2520. Springer, 2003.
9. Kuijpers B. and Othman W. Trajectory databases: Data models, uncertainty and complete query languages. In Proc. 11th Int. Conf. on Database Theory, 2007, pp. 224–238.
10. Li L. and Revesz P. Interpolation methods for spatiotemporal geographic data. J. Comput. Environ. Urban Syst., 28(3):201–227, 2004.
11. Ni J. and Ravishankar C.V. Indexing spatio-temporal trajectories with efficient polynomial approximations. IEEE Trans. Knowl. Data Eng., 19(5):663–678, 2007.
12. Pfoser D. and Jensen C.S. Capturing the uncertainty of moving-object representations. In Proc. Int. Symp. on Large Spatial Databases, 1999, pp. 111–132.
13. Revesz P. and Wu S. Spatiotemporal reasoning about epidemiological data. Artif. Intell. Med., 38(2):157–170, 2006.
14. Shepard D.A. A two-dimensional interpolation function for irregularly spaced data. In Proc. 23rd ACM National Conf., 1968, pp. 517–524.
15. Trajcevski G., Wolfson O., Zhang F., and Chamberlain S. The geometry of uncertainty in moving objects databases. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 233–250.
16. Yu B., Kim S.H., Bailey T., and Gamboa R. Curve-based representation of moving object trajectories. In Proc. 8th Int. Database Engineering and Applications Symp., 2004, pp. 419–425.

Spatio-Temporal OLAP

▶ Spatio-Temporal Data Warehouses

Spatio-Temporal Online Analytical Processing

▶ Spatio-Temporal Data Warehouses

Spatio-Temporal Representation

▶ Spatio-Temporal Trajectories

Spatio-Temporal Selectivity Estimation

George Kollios
Boston University, Boston, MA, USA

Synonyms

Selectivity for predictive spatio-temporal queries

Definition

In spatio-temporal databases, the locations of moving objects are usually modeled as linear functions of time. Thus, the location of an object at time t is represented as $o(t) = o_s + o_v t$, where o_s is the initial location of the object at time $t = 0$ and o_v is its velocity. Given that the object moves in a $d-$dimensional space, $o(t)$, o_s, and o_v are $d-$dimensional vectors. In this setting, the selectivity estimation of spatio-temporal queries is defined as follows:

Given a database that stores the locations of moving objects and a spatio-temporal query, estimate the number of objects that satisfy the query.

There are two important types of queries in this environment: spatio-temporal window (or range) queries and spatio-temporal distance join queries. A spatio-temporal window query (*STWQ*) specifies a (static or moving) region q_S, a future time interval q_T, and a dataset of moving objects D, and retrieves all data objects that will intersect (or will be covered by) q_S during q_T. On the other hand, a spatio-temporal distance join query (*STDJQ*) assumes two datasets D_A and D_B of moving objects, a time period q_T and a distance q_d, and asks for the pairs of objects (o_a, o_b) that $o_a \in D_A$, $o_b \in D_B$ and are closer than q_d during the time interval q_T.

Historical Background

The problem of spatio-temporal selectivity estimation is related to spatio-temporal indexing and spatial selectivity estimation. In the spatio-temporal indexing problem the system must report all the objects that satisfy the query; here the system must just give an estimate on the number of such objects. In spatial selectivity estimation the main difference is that the objects are static and therefore easier to model and represent succinctly.

In particular, selectivity estimation for spatial queries is based on spatial histograms. A histogram partitions the data space into a set of buckets, and the object distribution in each bucket is assumed to be (almost) uniform. A bucket B stores the number $B.num$ of objects whose centroids fall in B, and the average extent $B.len$ of such objects. Consider a window query Q. Then, using an analysis based on uniformly distributed data [12,7], the expected number of qualifying objects in B is approximated by $B.num\frac{I.area}{B.area}$, where $I.area$ is the area of the intersection of the query Q with bucket B and $B.area$ the area of B. The total number of objects intersecting Q is estimated by summing the results of all buckets. Evidently, satisfactory estimation accuracy depends on the degree of uniformity of objects' distributions in the buckets. An example of a histogram construction that generates nearly uniform buckets for spatial datasets appeared in [1].

Another approach uses spatial sketches to estimate the selectivity of spatial queries and in addition provides probabilistic guarantees on the quality of the estimation [4]. In particular, given a dataset D of n spatial rectangles (points are assumed to be degenerated rectangles) it is possible to create a synopsis for each dataset, which has size poly-logarithmic to n and proportional to $\frac{1}{\epsilon^2}$, and provides answers that with high probability are $\pm\ \varepsilon$ away from the exact answer. The synopsis is based on AGMS sketches [2] extended to handle multidimensional intervals (rectangles) in poly-logarithmic space. Note that the histogram based techniques provide no guarantees on the estimation quality (e.g., in the worst case the error can be arbitrary large), but work well in most practical cases.

Foundations

To estimate the selectivity of STWQs efficiently two basic approaches have been proposed. One is based on random sampling and the other on spatio-temporal histograms. Moreover, the histogram based approach can be extended for STDJQs as well.

Selectivity Estimation for STWQ

The simplest approach is to use random sampling [6]. Given a dataset of n moving objects the method keeps a uniform random sample S of the set of moving objects that is used to estimate the result. Note that each time an object issues an update, the function that represents its location in the database must change. Therefore, a tuple must be deleted (the one that corresponds to the old function) and a new one must be inserted. To maintain a uniform random sample in such a dynamic environment a specialized solution must be used [5,9]. For a query Q, let r be the size of the result of Q over the random sample S. Then, the method estimates the result of Q as $r\frac{n}{|S|}$, where $|S|$ is the size of S. If the set of queries is known beforehand, then a technique that is based on stratified sampling can be applied which reduces the size of the sample and increases the accuracy of the estimator. This improved method is called Venn sampling and appeared in [10].

The other approach is based on spatio-temporal histograms. For $d-$dimensional moving points, the histogram is constructed in a $2d-$dimensional space, where d dimensions represent the coordinates of the moving objects at a reference time instant (e.g., $t = 0$) and the other d dimensions their velocity. Each bucket stores the number of objects and the minimum and maximum locations and velocities of these objects in each dimension. Furthermore, it is assumed that the objects are distributed uniformly inside the bucket. Based on that, a query estimation procedure is used to find the percentage of the objects in each bucket that are expected to intersect the query. Then, the (approximate) result of Q is obtained by summing the contributions of all buckets (see [3,6,11] for more details). The spatio-temporal histogram can also be dynamically maintained in the presence of object updates.

Selectivity Estimation for STDJQ

For estimating the size of a STDJQ between two sets of moving objects D_A and D_B, two spatio-temporal histograms are maintained, one for each set. Then, there are two approaches to estimate the size of the join. In one approach, the distance between objects is defined using the L_{max} norm, i.e., given two $d-$dimensional points a and b, their distance is: $dist(a, b) = \max_{i=1,2,\ldots,d}|a.x_i - b.x_i|$. Because of the independence between the dimensions under this distance, the selectivity of the join query can be estimated using the selectivity of each dimension independently of the others. Thus, the selectivity of a STDJQ Q is expressed as $Sel(Q) = \Pi_{i=1}^{d} Sel_i(Q)$. Furthermore, the $Sel_i(Q)$ can be estimated using the projections of the buckets to dimension i. More details can be found in [8].

In the other approach, the distance function is the L_2 metric (i.e., Euclidean distance), and a more complicated formula is used to estimate the selectivity of the query. The analysis is based on the following idea: consider a moving object p_1 with specific location and velocity and another object p_2 with fixed velocity. Also, assume that the location of p_2 is distributed uniformly in space. Based on these assumptions, it is possible to compute the probability that p_2 will satisfy the query. Moreover, using this probability, the selectivity of the SPDJQ can be estimated. Note that, this method can be used for both moving points and moving rectangles and can be extended to other L_p norms. The details of this technique appears in [11].

Key Applications

Selectivity estimation is used extensively in database query optimization. Therefore, the techniques developed for spatio-temporal selectivity estimation can be useful in systems that need to generate execution plans for spatio-temporal queries. Also, many application like traffic monitoring, sensor networks or mobile communications, require aggregate information about the locations of moving objects. Furthermore, getting the exact answer to these queries can be very expensive. In that case, fast approximate answers on spatio-temporal aggregation queries using the techniques discussed above is the best alternative.

Future Directions

Current techniques for spatio-temporal selectivity estimation use the assumption that objects move linearly over time. However, in many real life application this may not be a good approximation. It is an open problem how to extend the existing techniques to handle moving objects with non-linear motion functions.

Cross-references

► Indexing of the Current and Near-Future Positions of Moving Objects

► Spatio-Temporal Data Warehouses

Recommended Reading

1. Acharya S., Poosala V., and Ramaswamy S. Selectivity estimation in spatial databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 13–24.
2. Alon N., Gibbons P.B., Matias Y., and Szegedy M. Tracking join and self-join sizes in limited storage. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 10–20.
3. Choi Y.-J. and Chung C.-W. Selectivity estimation for spatio-temporal queries to moving objects. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 440–451.
4. Das A., Gehrke J., and Riedewald M. Approximation techniques for spatial data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 695–706.
5. Frahling G., Indyk P., and Sohler C. Sampling in dynamic data streams and applications. In Proc. Symp. on Computational Geometry, 2005, pp. 142–149.
6. Hadjieleftheriou M., Kollios G., and Tsotras V.J. Performance evaluation of spatio-temporal selectivity estimation techniques. In Proc. 15th Int. Conf. on Scientific and Statistical Database Management, 2003, pp. 202–211.
7. Pagel B.-U., Six H.-W., Toben H., and Widmayer P. Towards an analysis of range query performance in spatial data structures. In Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1993, pp. 214–221.
8. Sun J., Tao Y., Papadias D., and Kollios G. Spatio-temporal join selectivity. Inf. Syst., 31(8):793–813, 2006.
9. Tao Y., Lian X., Papadias D., and Hadjieleftheriou M. Random sampling for continuous streams with arbitrary updates. IEEE Trans. Knowl. Data Eng., 19(1):96–110, 2007.
10. Tao Y., Papadias D., Zhai J., and Li Q. Venn sampling: a novel prediction technique for moving objects. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 680–691.
11. Tao Y., Sun J., and Papadias D. Analysis of predictive spatio-temporal queries. ACM Trans. Database Syst., 28(4):295–336, 2003.
12. Theodoridis Y. and Sellis T. A model for the prediction of R-tree performance. In Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1996, pp. 161–171.

Spatio-Temporal Stream Processing

► Continuous Monitoring of Spatial Queries

Spatio-Temporal Trajectories

Elias Frentzos[1], Yannis Theodoridis[1],
Apostolos N. Papadopoulos[2]
[1]University of Piraeus, Piraeus, Greece
[2]Aristotle University, Thessaloniki, Greece

Synonyms

Moving object trajectories; Spatio-temporal representation

Definition

A spatio-temporal trajectory can be straightforwardly defined as a function from the temporal $I \subseteq \mathbb{R}$ domain to the geographical space $\mathbb{R}^2$, i.e., the 2-dimensional plane. From an application point of view, a trajectory is the recording of an object's motion, i.e., the recording of the positions of an object at specific timestamps.

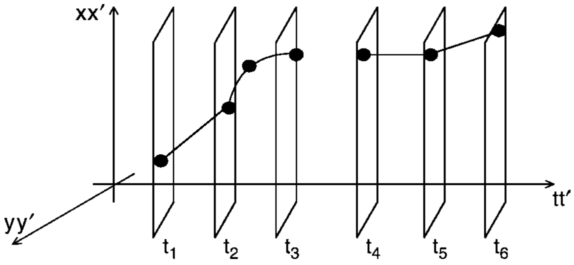
Generally speaking, spatio-temporal trajectories can be classified into two major categories, according to the nature of the underlying spatial object: (i) objects without area represented as moving points, and (ii) objects with area, represented as moving regions; in this case the region extent may also change with time. Among the above two categories, the former has attracted the main part of the research interest, since the majority of real-world applications involving spatio-temporal trajectories consider objects represented as points, e.g., fleet management systems monitoring cars in *road networks*.

Focusing on trajectories of moving points, while the actual trajectory consists of a curve, real-world requirements imply that the trajectory has to be built upon a set of sample points, i.e., the time-stamped positions of the object. Thus, trajectories of moving points are often defined as sequences of (x, y, t) triples:

$$T = \{(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots, (x_n, y_n, t_n)\},$$

where $x_i, y_i, t_i \in \mathbb{R}$, and $t_1 < t_2 < \ldots < t_n$, and the actual trajectory curve is approximated by applying *spatio-temporal interpolation* methods on the set of sample points; among the proposed in the literature *spatio-temporal interpolation* techniques, the notion of linear interpolation has been widely adopted, given that it is fast, natural, and easy to implement (Fig. 1).

There are several techniques developed merely for the management of spatio-temporal trajectories in databases: *spatio-temporal models and languages*, *indexing of historical spatio-temporal data*, advanced query processing, and trajectory summarization techniques.



Spatio-Temporal Trajectories. Figure 1. The spatio-temporal trajectory of a moving point: dots represent sampled positions and lines in between represent alternative interpolation techniques (linear vs. arc interpolation). Unknown type of motion can be also found in a trajectory (see $[t_3, t_4)$ time interval).

Historical Background

From a modeling perspective, the concept of spatio-temporal trajectories was introduced in some early works [6,7,10], which addressed the need for capturing the complete history of objects' movement. Clearly, as location data may change over time, the database must contain the whole history of this development. Thus, the DBMS should be allowed to go back in time at any particular timestamp, and to retrieve the state of the database at that time. More specifically, [11] models moving points (*mpoints*) and moving regions (*mregions*) as 3-dimensional (2D space + time) or higher-dimensional entities whose structure and behavior is captured by modeling them as abstract data types. Such types and their operations for spatial values changing over time can be integrated as base (attribute) data types into an extensible DBMS. Guting et al. [11] introduced a type constructor τ which transforms any given atomic data type a into a type $\tau(a)$ with semantics $\tau(a) = time \rightarrow a$. In this way, the two aforementioned basic types, namely *mpoint* and *mregion*, may be also represented as $\tau(point)$ and $\tau(region)$, respectively. Guting et al. [11] also provided an algebra with data types (such as moving point, moving region, moving real, etc.) together with a comprehensive set of operations, supporting a variety of queries of spatio-temporal trajectory data.

On another line of research, [14] first dealt with the special requirements that spatio-temporal trajectories

Spatio-Temporal Trajectories. Table 1. Classification of spatio-temporal queries

Query type		Operation
Coordinate-based		*Overlap, inside, etc.*
Trajectory-based	Topological	*Enter, leave, cross, bypass, etc.*
	Navigational	*Travelled distance, covered area, speed, heading, etc.*

pose to the database engine, in terms of efficient index structures and specific query processing techniques. In particular, [14] addressed the most commonly used queries over spatio-temporal trajectories and classified them according to Table 1. Based on the observation that many query types are trajectory-based (i.e., they require the knowledge of a significant part of the moving objects' trajectory), [14] proposed the Trajectory Bundle tree (TB-tree), based on the well-known *R-tree*, considered as a seminal work in the context of *Indexing Historical Spatio-temporal Data.*

Foundations

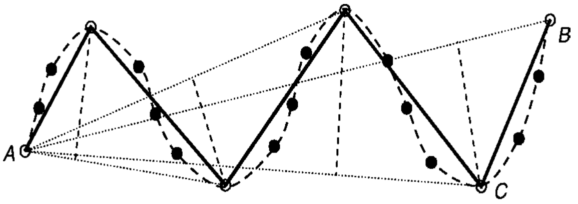
Spatio-temporal trajectories may be queried with a variety of operators, which are mainly extensions of existing spatial operators. Among them, the simple spatio-temporal range query, involving both spatial and temporal components over *R-tree* structures indexing spatio-temporal trajectories, is a straightforward generalization of the standard *R-tree FindLeaf* algorithm in the 3-dimensional space. *Nearest neighbor queries* have been also considered in the context of spatio-temporal trajectories; the algorithms over *R-trees* and variations (e.g., TB-tree) proposed in [8], are based on both depth-first and best-first *R-tree* traversals, similar to the algorithms used for nearest neighbor querying over spatial data. The proposed algorithms vary with respect to the type of the query object (stationary or moving point) as well as the type of the query result (historical continuous or not), thus resulting in four types of nearest neighbor queries.

Trajectory join has been also investigated in several papers motivated as an extension of the respective spatial operator. Bakalov et al. in [3] consider the problem of evaluating all pairs of trajectories between two datasets, during a given time interval, which, given a distance function, all distances between timely corresponding trajectory positions are within a given threshold. Then an approximation technique is used to reduce the trajectories into symbolic representations (strings) so as to lower the dimensionality of the original (3-dimensional) problem to one. Using the constructed strings, a special lower-bounding metric supports a pruning heuristic which reduces the number of candidate pairs to be examined. The overall schema is subsequently indexed by a structure based on the B-tree, requiring also minimal storage space. Another variation on the subject of joining trajectories is the closest-point-of-approach recently introduced in [2]. Closest-point-of-approach requires finding all pairs of line segments between two trajectories such that their distance is less than a predefined threshold. The work presented in [2] proposes three approaches; the first utilizes packed R-trees treating trajectory segments as simple line segments in the $d + 1$ dimensional space, and then employs the well known *R-tree spatial join* algorithm which requires carefully controlled synchronized traversal of the two *R-trees*. The second is based on a plane sweep algorithm along the temporal dimension, while the third is an adaptive algorithm which naturally alters the way in which it computes the join in response to the characteristics of the underlying data.

Much more challenging are the so called *similarity-based* queries over spatio-temporal trajectory data. Similarity search has been extensively studied within the time series domain; consequently, techniques addressed there, are usually extended in the spatio-temporal domain, in which spatio-temporal trajectories are considered as time series. Traditionally, similarity search has been based on the Euclidean Distance between time series, nevertheless having several disadvantages which the following proposals are trying to confront. In particular, in order to compare sequences with different lengths, [12] use the Dynamic Time Warping (DTW) technique that allows sequences to be stretched along the time axis so as to minimize the distance between sequences. Although DTW incurs a heavy computation cost, it is robust against noise. Moreover, in order to reduce the effect of its quadratic complexity on large time series, a lower bounding function along with a dedicated index structure has been proposed for pruning in [12]. Longest Common SubSequence (LCSS) measure [16] matches two sequences by allowing them to stretch, without rearranging the sequence of the elements, also allowing some elements to be unmatched (which is the main advantage of the LCSS measure compared to Euclidean Distance and DTW). Therefore, LCSS can efficiently handle outliers and different scaling factors. In [5], a distance function, called Edit Distance on Real Sequences (EDR), was introduced. EDR distance function is based on the edit distance, which is the number of insert, delete, or replace operations that are needed to convert a trajectory T into Q. In the respective experimental study presented in [5], EDR was shown to be more robust than DTW and LCSS over trajectories with noise. In order to speed up the similarity search between trajectories, both [16] and [5], rely on dedicated index structures, thus achieving pruning of over 90% of the total number of indexed trajectories. However, such approaches fail to utilize the most commonly available access methods for *Indexing Historical Spatio-temporal Data* such as *R-trees*, leading to additional overhead. In order to overcome this disadvantage, [9] employ the average Euclidean distance between two trajectories as a measure of their dissimilarity, and then, using *R-trees* indexing spatio-temporal trajectories provide a series of metrics and heuristics which efficiently prune the search space. These metrics are based (i) on the observation that an upper value for the speed of the spatio-temporal trajectories can provide lower and upper bounds of the average Euclidean distance between two trajectories, and (ii), on the fact that a best-first *R-tree* traversal on the $mindist(N,q)$ between a query trajectory q and a node N, provides a tighter lower bound for the average Euclidean distance. Finally, [9] provide an efficient algorithm for k-most similar trajectory search over *R-trees* indexing spatio-temporal trajectories.

In the indexing domain, a challenging line of research deals with network-constrained trajectories; this is due to the fact that the majority of the applications involving spatio-temporal trajectories deal with objects moving along *road networks* (i.e., cars, buses, trains). Following this observation, several specific access methods for objects moving in networks have been proposed; among them the most efficient is the Moving Objects in Networks tree (MON-tree) [1]. However, in order to exploit such network constrained indexes, the need for mapping the trajectories into the underlying network introduces the so called *map matching* problem. Specifically, the observation that raw trajectory positions are affected by the measurement error introduced by e.g., GPS, and, the sampling error being up to the frequency with which position samples are taken, reveals the problem of correctly matching such tracking data in an underlying map containing e.g., a *road network*. Currently, the state-of-the-art approach addressing the map matching problem is the one presented in [4], which proposes mapping the entire trajectory to candidate paths in the *road network* using the Fréchet distance, which can be illustrated as follows: suppose a human and her dog constrained to walk on two different curves, while they are both allowed to control their speed independently. Then, the Fréchet distance between the two curves is the minimal length of a leash that is necessary. The proposed global map-matching algorithms in [4] find a curve in the *road network* that is as close as possible to the given trajectory in terms of the Fréchet distance between them.

Last but not least, it is the need for compression techniques that arises due to the fact that all the ubiquitous positioning devices will eventually start to generate an unprecedented stream of time-stamped positions, leading to storage and computation challenges [13]. In this direction, [13] exploit existing algorithms used in the line generalization field, and present one *top-down* and one *opening window* algorithm, which can be directly applied to spatio-temporal trajectories. The *top-down* algorithm, named TD-TR, is based on the well known Douglas-Peucker algorithm (Fig. 2) originally used in the context of cartography. This algorithm calculates the perpendicular distance of each internal point from the line connecting the first and the last point of the polyline (line AB in Fig. 2) and finds the point with the greatest perpendicular distance (point C). Then, it creates lines AC and CB

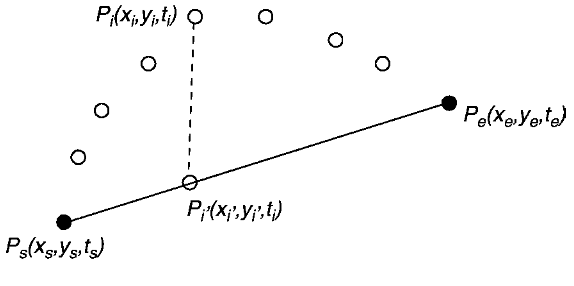


Spatio-Temporal Trajectories. Figure 2. Top-down Douglas-Peucker algorithm used for trajectory compression. Original data points are represented by closed circles [MB04] [13].

and, recursively, checks these new lines against the remaining points with the same method. When the distance of all remaining points from the currently examined line is less than a given threshold (e.g., all the points following *C* against line *BC* in Fig. 2) the algorithm stops and returns this line segment as part of the new – compressed – polyline. Being aware of the fact that trajectories are polylines evolving in time, the algorithm presented in [13] replaces the perpendicular distance used in the DP algorithm with the so-called *Synchronous Euclidean Distance* (SED), which is the distance between the currently examined point (P_i in Fig. 3) and the point of the line (P_s, P_e) where the moving object would lie, assuming it was moving on this line, at time instance t_i determined by the point under examination (P_i' in Fig. 3). The experimental study presented in [13] shows that such compression techniques introduce a small and manageable error, reducing at the same time the size of the dataset under 40% of its original size.

Key Applications

- *Location-based Services (LBS)* – Spatio-temporal trajectories are used in location-based services for determining the exact position of users, based on the map-matching solutions provided over trajectories.
- *Spatio-temporal Decision Support Systems (STDSS)* – A number of decision support tasks can exploit the presence of spatio-temporal trajectories. Traffic estimation and prediction systems. Analysis of traffic congestion conditions. Fleet management systems. Urban and regional planers analyzing the life courses of city residents. Scientists studying animal immigration habits.



Spatio-Temporal Trajectories. Figure 3. The Synchronous Euclidean Distance (SED): The distance is calculated between the point under examination (P_i) and the point P_i' which is determined as the point on the line (P_s, P_e) the time instance t_i [MB04] [13].

Future Directions

There are several research directions arising regarding spatio-temporal trajectory data management. For example, the problem of estimating the selectivity of a range query over historical trajectory data still remains open. More specifically, such a technique would have to deal with the *distinct counting* problem [15], which is also present in the context of *Spatio-temporal Data Warehouses*. This problem stands when an object samples its position in several timestamps inside a given query window resulting to be counted multiple times in the query result. Nevertheless, a selectivity estimation technique based on a space partitioning method, such as a histogram, would had to return the number of distinct trajectories contained inside the query region, summing the containment of several buckets; then trajectories appearing in several buckets would had to be counted only once.

Another interesting research direction appears when considering network-constraint trajectory compression; in particular, existing compression techniques do not consider that trajectories may be network-constraint, resulting in trajectories which after the compression may be invalid regarding the underline network. As such, future work should investigate on techniques which may produce compressed trajectories being still valid under the network constraints.

Experimental Results

In general, for every presented method, there is an accompanying experimental evaluation in the corresponding reference.

Data Sets

A collection of real spatio-temporal datasets, as well as links to generators for spatio-temporal trajectory data can be found at *R-tree portal* (URL: http://www.rtreeportal.org/).

Url to Code

R-tree portal (URL: http://www.rtreeportal.org/) contains the code for most common spatio-temporal indexes, as well as data generators and several useful links on spatio-temporal databases.

Cross-references

▶ Indexing Historical Spatio-Temporal Data
▶ Nearest Neighbor Query
▶ Road Networks
▶ Rtree
▶ Spatial
▶ Spatial and Spatio-Temporal Data Models and Languages
▶ Spatial Join
▶ Spatio-Temporal Data Warehouses
▶ Spatiotemporal Interpolation Algorithms

Recommended Reading

1. Almeida V.T. and Guting R.H. Indexing the trajectories of moving objects in networks. GeoInformatica, 9(1):33–60, 2005.
2. Arumugam S. and Jermaine C. Closest-point-of-approach join for moving object histories. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 86.
3. Bakalov P., Hadjieleftheriou M., Keogh E., and Tsotras V. Efficient trajectory joins using symbolic representations. In Proc. 6th Int. Conf. on Mobile Data Management, 2005, pp. 86–93.
4. Brakatsoulas S., Pfoser D., Salas R., and Wenk C. On map-matching vehicle tracking data. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 853–864.
5. Chen L., Özsu M.T., and Oria V. Robust and fast similarity search for moving object trajectories, In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 491–502.
6. Chomicki J. and Revesz P. A geometric framework for specifying spatiotemporal objects. In Proc. 6th Int. Workshop Temporal Representation and Reasoning, 1999, pp. 41–46.
7. Erwig M., Güting R.H., Schneider M., and Varziginannis M. Spatio-temporal data types: an approach to modeling and querying moving objects in databases. GeoInformatica, 3(3):265–291, 1999
8. Frentzos E., Gratsias K., Pelekis N., and Theodoridis Y. Algorithms for nearest neighbor search on moving object trajectories. Geoinformatica, 11(2):159–193, 2007.
9. Frentzos E., Gratsias K., and Theodoridis Y. Index-based most similar trajectory search. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 816–825.
10. Forlizzi L., Güting Nardelli E., and Schneider M. A data model and data structures for moving objects databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 319–330.
11. Guting R.H., Bohlen M.H., Erwig M., Jensen C.S., Lorentzos N.A., Schneider M., and Vazirgiannis M. A foundation for representing and querying moving objects. ACM Trans. Database Syst., 25(1):1–42, 2000.
12. Keogh E. Exact indexing of dynamic time warping. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 406–417.
13. Meratnia N. and By R. Spatiotemporal compression techniques for moving point objects. In Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 765–782.
14. Pfoser D., Jensen C.S., and Theodoridis Y. Novel approaches to the indexing of moving object trajectories. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 395–406.
15. Tao Y., Kollios G., Considine J., Li F., and Papadias D. Spatio-temporal aggregation using sketches. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 214–226.
16. Vlachos M., Kollios G., and Gunopulos D. Discovering similar multidimensional trajectories. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 673–684.

SPC Query

▶ Conjunctive Query

SPCU-Algebra

▶ Positive Relational Algebra

Specialization

▶ Abstraction

Specialization and Generalization

Bernhard Thalheim
Christian-Albrechts University Kiel, Kiel, Germany

Synonyms

Refinement; Abstraction; Hierarchies

Definition

Specialization and generalization are main principles of database modeling. Specialization is based on a refinement of types or classes to more specific ones. Generalization maps or groups types or classes to more abstract or combined ones. Typically, generalizations and specializations form a hierarchy of types and classes.

Key Points

Specialization introduces a new entity type by adding specific properties belonging to that type, which are different from the general properties of its more general type. Is-A associations specialize a type to a more specific one. Is-A-Role-Of associations consider a specific behavior of objects. Is-More-Specific-To associations specialize properties of objects of the more general type. The *student* type and the *customer* type are specializations of the *person* type. The *rectangle* type is specialized to the *square* type by adding restrictions. Different kinds of specialization may be distinguished: structural specialization which extends the structure, semantic specialization which strengthens type restrictions, pragmatic specialization which allows a separation of the different usage of objects in contexts, operational specialization which introduces additional operations, and hybrid specializations. Identification and other properties of objects of the special type can be inherited from the more general one. Methods applicable to objects of the more general one should be applicable to corresponding more special objects or specialized as well. Exceptions can be modeled by specializations. Specialization allows developers to avoid null values and to hide details from non-authorized users.

Generalization combines common features, attributes, or methods of types. It is based either on abstraction, on combination or on grouping. Generalization often tends to be an abstraction in which a more general type is defined by extracting common properties of one or more types while suppressing the differences between the subtypes. The subtypes can be virtually clustered by or generalized to or combined by a view to a general type. The *library's holding* type is a generalization of the *journal*, *book*, *preprint* and *PhD/Master thesis* types. The *occupation* type is a generalization of the *lawyer*, *merchant*, *teacher* and *banker* types. It is obtained by factoring out the commonalities among the specializations. Structural combination typically assumes the existence of a unifiable identification of all types. The *livestock* type combines the different types of farming. Generalization is represented by clusters of types. The cluster construct of the extended ER model represents common properties and abstractions. Identification of generalized objects is either inherited from the more special objects or built as an abstraction of the identification of the more special types. Generalizations often do not have their own methods.

Cross-references

► Database Fundamentals – Semantic Data Models

Recommended Reading

1. Ter Bekke J.H. Semantic Data Modeling. Prentice-Hall, London, 1992.
2. Thalheim B. Entity-Relationship Modeling – Foundations of Database Technology. Springer, Berlin Hiedelberg New York, 2000.

Specificity

Jovan Pehcevski[1], Benjamin Piwowarski[2]

[1]INRIA Paris Rocquencourt, Le Chesnay Cedex, France

[2]University of Glasgow, Glasgow, UK

Synonyms

Coverage

Definition

Specificity is a relevance dimension that describes the extent to which a document part focuses on the topic of request. In the context of structured text (XML) retrieval, a document part corresponds to an XML element.

Specificity is defined as the length ratio, typically in number of characters, of contained relevant to irrelevant text in the document part. Different Specificity values can be associated to a document part. These values are drawn from the Specificity relevance scale, which has evolved from a discrete multi-graded relevance scale to a continuous relevance scale.

Key Points

The Initiative for the Evaluation of XML Retrieval (INEX) has defined Specificity as a relevance dimension that uses values from its own relevance scale to express the extent to which an XML element focuses on the topic of request. Since 2002, different names and relevance scales were used for Specificity at INEX. It initially evolved because the relevance dimension was not sufficiently well defined, and later because the assessment procedure changed.

In 2002, Specificity was named coverage at INEX, which reflected the extent to which an XML element was focused on aspects of the information need (as represented by the INEX topic). The component coverage used a relevance scale comprising four relevance grades, from “no coverage,” “too large,” “too small,” to “exact coverage.” However, this dimension was used solely in 2002, partly because of the vagueness introduced in the terminology for its name, and partly because it has been subsequently shown that the INEX 2002 assessors did not particularly understand the notion of “too small” [1]. In particular, assessors understood “too small” as a measure of quantity while Specificity is more related to the concentration of relevant information. In 2003 and 2004, four grades were used for the Specificity relevance dimension at INEX, such that the extent to which an XML element may focus on the topic of request could range from “none” (0), to “marginally” (1), to “fairly” (2), or to “highly” (3) focused. An XML element was considered relevant only if its Specificity value was greater than zero.

From 2005 onwards, a highlighting assessment procedure was used at INEX to gather relevance assessments for the XML retrieval topics. The Specificity of an XML element is automatically computed as the ratio of highlighted to fully contained text, where the relevance values that can be associated to the element are drawn from a continuous relevance scale. These values are in the range between 0 and 1, where the value of 0 corresponds to an element that does not contain any highlighted text, while the value of 1 corresponds to a fully highlighted element.

With the highlighting assessment procedure, assessors are asked to highlight all the relevant information contained by returned XML documents. This results in a reduced cognitive load on the assessor, since in this case there is no need for the assessor to explicitly associate a Specificity value to a judged element. Studies of the level of assessor agreement, which used topics that were double-judged at INEX, have shown that the use of the new highlighting procedure further increases the level of assessor agreement compared to the level of agreement observed among assessors during previous years at INEX [2,3].

Cross-references

- ▶ Evaluation Metrics for Structured Text Retrieval
- ▶ Relevance

Recommended Reading

1. Kazai G., Masood S., and Lalmas M. A study of the assessment of relevance for the INEX 2002 test collection. In Proc. 26th European Conf. on IR Research, 2004, pp. 296–310.
2 Pehcevski J. and Thom J.A. HiXEval: highlighting XML retrieval evaluation. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, pp. 43–57.
3. Trotman A. Wanted: element retrieval users. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 63–69.

Spectral Clustering

Sergios Theodoridis[1],
Konstantinos Koutroumbas[2]
[1]University of Athens, Athens, Greece
[2]Institute for Space Applications and Remote Sensing, Athens, Greece

Synonyms

Graph-based clustering

Definition

Let X be a set $X = \{\mathbf{x}_1, \mathbf{x}_2,\ldots,\mathbf{x}_N\}$ of N data points. An *m-clustering* of X, is defined as the partition of X into m sets (*clusters*), $C_1,\ldots,C_m$, so that the following three conditions are met:

- $C_i \neq \emptyset,\ i = 1,\ldots,m$
- $\cup_{i=1}^{m} C_i = X$
- $C_i \cap C_j = \emptyset,\ i \neq j,\ i, j = 1,\ldots,m$

In addition, the data points contained in a cluster C_i are “more similar” to each other and “less similar” to the points of the other clusters. The terms “similar” and “dissimilar” depend very much on the types of clusters the user expects to recover from X. A clustering defined as above is known as hard clustering, to distinguish it from the fuzzy clustering case.

Historical Background

The essence of clustering is to “reveal” the organization of patterns into “sensible” groups. It has been used as a critical analysis tool in a vast range of disciplines, such as medicine, social sciences, engineering, computer science, machine learning, bioinformatics, data mining and information retrieval. The literature is huge and numerous techniques have been suggested over the years. A comprehensive introduction to clustering can be found e.g., in [14]. An important class of clustering algorithms builds around graph theory. Reference [9] is one of the first efforts in this direction. Points of X are assigned to the nodes of a graph.

Notions such as minimum spanning tree and directed trees have extensively been used to partition the graph into clusters (e.g., [14]). Spectral clustering is a more recent class of graph based techniques, which unravels the structural properties of a graph using information conveyed by the spectral decomposition (eigendecomposition) of an associated matrix. The elements of this matrix code the underlying similarities among the nodes (data points) of the graph (e.g., [3]). Among the earlier works on spectral clustering are [6,12]. Fiedler [4] was one of the first to show the application of eigenvectors in graph partitioning.

Foundations

In the sequel, the simplest task of partitioning a given data set, X, into two clusters, A and B, is considered. Let $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset R^l$, where the latter denotes the l-dimensional Euclidean space. According to the previous discussion, the following preliminary steps are in order:

- Construction of a graph $G(V, E)$, where each vertex of the graph corresponds to a point $\mathbf{x}_i$, $i = 1,2,\ldots,N$, of X. It is further assumed that G is undirected and connected. In other words, there exists at least one path of edges that connects any pair of points in the graph.
- Assignment of a weight $W(i, j)$ to each one of the edges of the graph, e_{ij}, that quantifies proximity between the respective nodes, v_i, v_j in G (For notational convenience in some places i is used instead of v_i). The set of weights defines the $N \times N$ weight matrix W, also known as *affinity* matrix, with elements

$$W \equiv [W(i,j)],\ \ i,j = 1,2,\ldots,N$$

The weight matrix is assumed to be symmetric, i.e., $W(i, j) = W(j, i)$. The choice of the weights is carried out by the user and it is a problem dependent task. A common choice is

$$W(i,j) = \begin{cases} \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right), & if \quad ||\mathbf{x}_i - \mathbf{x}_j|| < \epsilon \\ 0, & otherwise \end{cases}$$

where ε is a user-defined constant and $||\cdot||$ is the Euclidean norm in the l-dimensional space.

By the definition of clustering, $A \cup B = X$ and $A \cap B = \emptyset$. Once a weighted graph has been formed, the second phase in any graph-based clustering algorithm consists of the following two steps: (i) Choose an appropriate clustering criterion for the partitioning of the graph and (ii) Adopt an efficient algorithmic scheme to determine the partitioning that optimizes the previous clustering criterion.

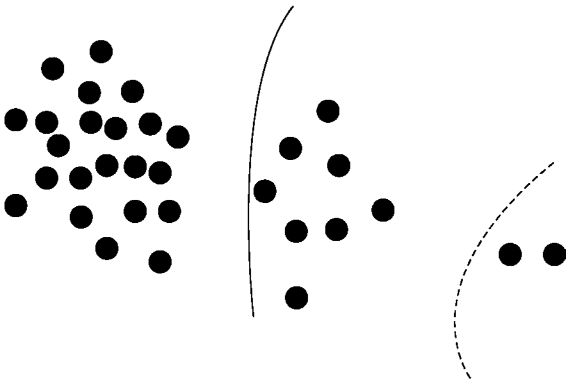
A clustering optimality criterion, that is in line with "common sense," is the so called *cut* [16]. If A and B are the resulting clusters, the associated *cut* is defined as:

$$cut(A,B) = \sum_{i \in A,\ j \in B} W(i,j) \tag{1}$$

Selecting A and B so that the respective $cut(A, B)$ is minimized means that the set of edges, connecting nodes in A with nodes in B, have the minimum sum of weights, indicating the lowest similarity between points in A and B. However, this simple criterion turns out to form clusters of small size of isolated points (least similar with the rest of the nodes). This is illustrated in Fig. 1. The minimum *cut* criterion would result in the two clusters separated by the dotted line, although the partition by the full line seems to be a more natural partitioning.

To overcome this drawback, the *normalized cut* criterion has been suggested in [12]. This is one of the most commonly used criteria in spectral clustering. The essence of this criterion is to minimize the *cut* and at the same time trying to keep the sizes of the formed clusters large. To this end, for each node, $v_i \in V$, in the graph G the index

$$D_{ii} = \sum_{j \in V} W(i,j) \tag{2}$$



Spectral Clustering. Figure 1. The dotted line indicates the clustering that is likely to favor the minimum cut criterion, while the full line indicates a more natural partitioning.

is defined. This is an index indicative of the "importance" of a node, v_i, $i = 1,2,\ldots,N$. The higher the value of D_{ii} the more similar the ith node is to the rest of the nodes. A low D_{ii} value indicates an isolated (remote) point. Given a cluster A, a measure of the "importance" of A is given by the following index

$$V(A) = \sum_{i \in A} D_{ii} = \sum_{i \in A,\ j \in V} W(i,j) \tag{3}$$

where $V(A)$ is sometimes known as the *volume* or the *degree* of A. It is obvious that small and isolated clusters will have a small $V(\cdot)$. The *normalized cut* between two clusters A,B is defined as

$$Ncut(A,B) = \frac{cut(A,B)}{V(A)} + \frac{cut(A,B)}{V(B)} \tag{4}$$

Obviously, small clusters correspond to large values (close to one) for the previous ratios, since in such cases $cut(A,B)$ will be a large percentage of $V(A)$.

Minimization of the $Ncut(A, B)$ turns out to be an NP-hard task. To bypass this computational obstacle, the problem will be reshaped to a form that allows an efficient approximate solution. To this end let ([1])

$$y_i = \begin{cases} \frac{1}{V(A)}, & if\ i \in A \\ -\frac{1}{V(B)}, & if\ i \in B \end{cases} \tag{5}$$

$$\mathbf{y} = [y_1, y_2, \ldots, y_N]^T$$

where T denotes the transpose operation. After some algebraic manipulations it can be verified that

$$\begin{aligned} &\left(\frac{1}{V(A)} + \frac{1}{V(B)}\right)^2 cut(A,B) \propto \\ &\frac{1}{2}\sum_{i \in V}\sum_{j \in V}(y_i - y_j)^2 W(i,j) = \mathbf{y}^T L \mathbf{y} \end{aligned} \tag{6}$$

where $\propto$ denotes proportionality and

$$L = D - W, \quad D \equiv diag\{D_{ii}\}$$

is known as the *graph Laplacian* matrix and D is the diagonal matrix having the elements D_{ii} across the main diagonal. It is also easily verified that

$$\mathbf{y}^T D \mathbf{y} = \frac{1}{V(A)} + \frac{1}{V(B)} \tag{7}$$

Combining (4), (6) and (7) it turns out that minimizing $Ncut(A, B)$ is equivalent with minimizing

$$J = \frac{\mathbf{y}^T L \mathbf{y}}{\mathbf{y}^T D \mathbf{y}} \tag{8}$$

subject to the constraint that $y_i \in \{\frac{1}{V(A)}, -\frac{1}{V(B)}\}$. Furthermore, based on the respective definitions, it can be shown that

$$\mathbf{y}^T D \mathbf{1} = 0 \tag{9}$$

where $\mathbf{1}$ is the N-dimensional vector having all its elements equal to 1. In order to bypass the computationally hard nature of the original task a relaxed problem will be solved: Eq. (8) will be minimized subject to the constraint of (9). The unknown "cluster labels," y_i, $i = 1,2,\ldots,N$, are now allowed to move freely along the real axis. Let

$$\mathbf{z} \equiv D^{1/2}\mathbf{y}$$

Then (8) becomes

$$J = \frac{\mathbf{z}^T \tilde{L} \mathbf{z}}{\mathbf{z}^T \mathbf{z}} \tag{10}$$

and the constraint in (9)

$$\mathbf{z}^T D^{1/2} \mathbf{1} = 0 \tag{11}$$

Matrix $\tilde{L} \equiv D^{-1/2} L D^{-1/2}$ is known as the *normalized graph Laplacian* matrix. It can easily be shown that $\tilde{L}$ has the following properties

- It is symmetric, real valued and nonnegative definite. Thus, as it is known from linear algebra, all its eigenvalues are non-negative and the corresponding eigenvectors are orthogonal to each other.
- $D^{1/2}\mathbf{1}$ is an eigenvector corresponding to zero eigenvalue. Indeed,

$$\tilde{L} D^{1/2} \mathbf{1} = 0$$

Obviously $\lambda = 0$ is the smallest eigenvalue of $\tilde{L}$, due to the non-negative definite nature of the matrix.

Furthermore, the ratio in (10) is the celebrated Rayleigh quotient for which the following hold (e.g., [5])

- The smallest value of the quotient, with respect to $\mathbf{z}$, is equal to the smallest eigenvalue of $\tilde{L}$ and it occurs for $\mathbf{z}$ equal to the eigenvector corresponding to this (smallest) eigenvalue.
- If the solution is constrained to be orthogonal to all eigenvectors associated with the j smaller

eigenvalues, minimization of the Rayleigh quotient results to the eigenvector corresponding to the next smallest eigenvalue, λ_{j+1} and the minimum value is equal to λ_{j+1}.

Taking into account i) the orthogonality condition in the constraint (11) and ii) the fact that $D^{1/2}\mathbf{1}$ is the eigenvector corresponding to the smallest eigenvalue $\lambda_0 = 0$ of $\tilde{L}$, it follows that:

The optimal solution vector $\mathbf{z}$ *minimizing the Rayleigh quotient in (10), subject to the constraint (11), is the eigenvector corresponding to the second smallest eigenvalue of* $\tilde{L}$.

In summary, the basic steps of the spectral clustering algorithm are:

- Given a set of points $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_N$ the weighted graph $G(V, E)$ is constructed. Then, the weight matrix W is formed by adopting a similarity rule.
- The matrices D, $L = D - W$ and $\tilde{L}$ are formed. The eigenanalysis of the normalized Laplacian matrix

$$\tilde{L}\mathbf{z} = \lambda\mathbf{z}$$

is performed and the computation of the eigenvector $\mathbf{z}_1$ corresponding to the second smallest eigenvalue λ_1 of $\tilde{L}$ is carried out. Then $\mathbf{y} = D^{-1/2}\mathbf{z}_1$ is computed.

- Finally, discretization of the components of $\mathbf{y}$ according to a threshold value takes place.

The final step is necessary since the components of the obtained solution are real-valued and our required solution is binary. To this goal, different techniques can be applied. For example, the threshold can be taken to be equal to zero. Another choice is to adopt the median value of the components of the optimum eigenvector. An alternative approach would be to select the threshold value that gives the minimum *cut*.

The eigenanalysis or spectral decomposition, as it is sometimes called, of an $N \times N$ matrix, using a general purpose solver, amounts to $O(N^3)$ operations. Thus, for large number of data points, this may be prohibitive in practice. However, for most of the practical applications the resulting graph is only locally connected, and the associated affinity matrix is a *sparse* one. Moreover, only the smallest eigenvalues/eigenvectors are required and also the accuracy is not of major issue, since the solution is to be discretized. In such a setting, the efficient Lanczos algorithm can be mobilized and the computational requirements drop down to approximately $O(N^{3/2})$.

So far, the partition of a data set into two clusters has been considered. If more clusters are expected, the scheme can be used in a hierarchical mode, where, at each step, each one of the resulting clusters is divided into two partitions. This is continued until a prespecified criterion is satisfied.

In the discussion above the focus was on a specific clustering criterion, i.e., the normalized cut, in order to present the basic philosophy behind the spectral clustering techniques. No doubt, a number of other criteria have been proposed in the related literature, e.g., [8,13]. In [15] a review and a comparative study of a number of popular spectral clustering algorithms is presented.

Key Applications

Spectral clustering has been used in a number of applications such as image segmentation and motion tracking [13,11], circuit layout [2], gene expression [8], machine learning [10], load balancing [7].

Cross-references

▶ Clustering Overview and Applications
▶ Graph
▶ Hierarchial Clustering
▶ Image Segmentation

Recommended Reading

1. Belikn M. and Niyogi P. Laplacian eigenmaps for dimensionality reduction and data representation. Neural Comput., 15(6):1373–1396, 2003.
2. Chan P., Schlag M., and Zien J. Spectral *k*-way ratio cut partitioning. IEEE Trans. Comput. Aided Des. Integrated Circ. Syst., 13:1088–1096, 1994.
3. Chung F.R.K. Spectral Graph Theory. American Mathematical Society, 1997.
4. Fiedler M. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. Czehoslovak Math. J., 25(100):619–633, 1975.
5. Golub G.H. and Van Loan C.F. Matrix Computations. John Hopkins, Baltimore, MD, USA, 1989.
6. Hagen L.W. and Kahng A.B. New spectral methods for ratio cut partitioning and clustering. IEEE Trans. Comput. Aided Des. Integrated Circ. Syst., 11(9):1074–1085, 1992.
7. Hendrickson B. and Leland R. Multidimensional spectral load balancing. In Proc. Fourth SIAM Conf. on Parallel Processing. 1993, pp. 953–961.
8. Kannan R., Vempala S., and Vetta A. On clusterings- good, bad and spectral. In Proc. 41st Annual Symp. on Foundations of Computer Science, 2000, pp. 367–377.
9. Ling R.F. On the theory and construction of *k*-clusters. Comput. J., 15:326–332, 1972.

10. Ng A.Y., Jordan M., and Weiis Y. On spectral clustering analysis and an algorithm. In Proc. 14th Conf. on Advances in Neural Information Processing Systems, 2001.
11. Qiu H. and Hancock E.R. Clustering and embedding using commute times. IEEE Trans. Pattern Anal. Mach. Intell., 29(11):1873–1890, 2007.
12. Scott G. and Longuet-Higgins H. Feature grouping by relocalization of eigenvectors of the proximity matrix. In Proc. British Machine Vision Conf., 1990, pp. 103–108.
13. Shi J. and Malik J. Normalized cuts and image segmentation. IEEE Trans. Pattern Anal. Mach. Intell., 22(8):888–905, 2000.
14. Theodoridis S. and Koutroumbas K. Pattern Recognition (4th edn.). Academic Press, 2008.
15. Verma D. and Meilă M. A comparison of spectral clustering algorithms. Technical report, UW-CSE-03-05-01, CSE Department, University of Washington, Seattle, 2003.
16. Wu Z. and Leahy R. An optimal graph theoretic approach to data clustering: Theory and its applications to image segmentation. IEEE Trans. Pattern Anal. Mach. Intell., 15(11):1101–1113, 1993.

Spider

► Web Crawler Architecture

Spidering

► Incremental Crawling

SPJRU-Algebra

► Positive Relational Algebra

Split

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

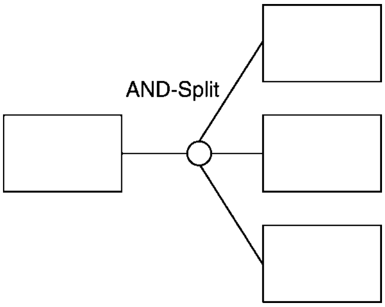
AND-split

Definition

A point within the workflow where a single thread of control splits into two or more parallel activities.

Key Points

The execution of parallel activities commences with an AND-Split and concludes with an AND-Join. For example, in a credit application process there may be a split in the workflow at which point multiple activities are completed separately (in parallel, if not simultaneously.) At an And-Split separate threads of control within the process instance are created; these threads will proceed autonomously and independently until reaching an And-Join condition. In certain workflow systems, all the threads created at an And-Split must converge at a common And-Join point (Block Structure); in other systems convergence of a subset of the threads can occur at different And-Join points, potentially including other incoming threads created from other And-split points.



Cross-references

► AND-Join
► OR-Split

Split Transactions

George Karabatis
University of Maryland, Baltimore County (UMBC), Baltimore, MD, USA

Definition

The split-transaction is an extended transaction model that introduces two new transaction management primitives/operations, namely, split and join. The split operation on a transaction T splits T and replaces it with two serialisable transactions; each one is later committed or aborted independently of the other. The inverse of split is the join operation on a transaction T which dissolves T by joining its results with a target transaction S.

Key Points

The concept of split transactions was introduced by Pu, Kaiser, and Hutchinson in [3] and later elaborated in [2] to support open-ended activities such as CAD/CAM projects, engineering type of applications, and software development. The syntax of the split-transaction operation on transaction T produces two new transactions A and B and dissolves T [3,2]:

```
Split-Transaction (
A: (AReadSet, AWriteSet, AProcedure),
B:  (BReadSet,  BWriteSet,  BProcedure))
```

where AReadSet, AWriteSet, BReadSet, BWriteSet are sets of data items accessed by A and B. AProcedure and BProcedure are the starting points of code where A and B will begin execution. There is no need to explicitly mention T in the arguments of the operation, as by definition the operation can only be executed within the body of transaction T. The syntax for a join operation on T to join S is: Join-Transaction (S: TID)

The split-transaction operation can be used to commit some work of a transaction early, or to distribute on-going work among several coworkers. On the contrary, the join-transaction is used to hand over and integrate results with a coworker [2]. These operations pertain to programmed transactions, and to actual open-ended activities with unpredictable developments (users determine the next operation to be executed in an adhoc manner). In the latter case, the read sets and write sets defined above are replaced with data sets on which application specific operations (e.g., edit or compile for software development) are applied; the AProcedure and Bprocedure are replaced with AUser and BUser specifying the users who take control of A and B. Thus, the definition changes to:

```
Split-Transaction (
A: (AReadSet, AWriteSet, AUser),
B: (BReadSet, BWriteSet, BUser))
```

There are additional operations such as: Split-Commit (transaction A is immediately committed, while B may be taken over by BUser; also Suspend (giving up control of a transaction) and Accept-Join-Transaction (a user executes this operation to accept responsibility of a joined transaction).

The main advantages of the restructuring operations (split/join transaction) on open-ended activities are:

- Adaptive recovery: committing resources that will not change
- Added concurrency: releasing committed resources or transferring ownership of uncommitted resources
- Serialisable access to resources by all activities

The split and join primitives were subsequently incorporated with nested transactions to produce combined transaction models [1].

Cross-references

▶ Database Management System
▶ Distributed Transaction Management
▶ Extended Transaction Models
▶ Serializability
▶ Transaction
▶ Transaction Management
▶ Transaction Manager

Recommended Reading

1. Chrysanthis P.K. and Ramamritham K. Synthesis of extended transaction models using ACTA. ACM Trans. Database Syst., 19 (3):450–491, 1994.
2. Kaiser G.E. and Pu C. Dynamic restructuring of transactions. In Database Transaction Models for Advanced Applications, A. K. Elmagarmid (ed.). Morgan Kaufmann Publishers, 1992, pp. 265–295.
3. Pu C., Kaiser G.E., and Hutchinson N.C. Split-transactions for open-ended activities. In Proc. 14th Int. Conf. on Very Large Data Bases, 1988, pp. 26–37.

SQL

Don Chamberlin
IBM Almaden Research Center, San Jose, CA, USA

Synonyms

SEQUEL; Structured Query Language

Definition

SQL is the world's most widely-used database query language. It was developed at IBM Research Laboratories in the 1970s, based on the relational data model defined by E. F. Codd in 1970. It supports retrieval, manipulation, and administration of data stored in tabular form. It is the subject of an international standard named Database Language SQL.

Historical Background

Early Language Development

In June 1970, E. F. Codd of IBM Research published a paper [4] defining the relational data model and introducing the concept of data independence. Codd's thesis was that queries should be expressed in terms of high-level, nonprocedural concepts that are independent of physical representation. Selection of an algorithm for processing a given query could then be done by an optimizing compiler, based on the access paths available and the statistics of the stored data; if these access paths or statistics should later change, the algorithm could be re-optimized without human intervention. In a series of papers, Codd proposed two high-level languages for querying relational databases, called relational algebra and relational calculus (also known as Data Sublanguage Alpha) [5].

The advantages of the relational model for application developers and database administrators were immediately clear. It was less clear whether an optimizing compiler could consistently translate nonprocedural queries into algorithms that were efficient enough for use in a production database environment. To investigate this issue, IBM convened a project called System R [1] at its research laboratory in San Jose, California. Between 1973 and 1979, this project designed and implemented a prototype relational database system based on Codd's ideas, testing and refining the prototype in several customer locations. SQL was the user interface defined by the System R research project. It later became the user interface for relational database products marketed by IBM and several other companies.

The principal goals that influenced the design of SQL were as follows:

1. SQL is a high-level, nonprocedural language intended for processing by an optimizing compiler. It is designed to be equivalent in expressive power to the relational query languages originally proposed by Codd.
2. SQL is intended to be accessible to users without formal training in mathematics or computer programming. It is designed to be typed on a keyboard. Therefore it is framed in familiar English keywords, and avoids specialized mathematical concepts or symbols.
3. SQL attempts to unify data query and update with database administration tasks such as creating and modifying tables and views, controlling access to data, and defining constraints to protect database integrity. In pre-relational database systems, these tasks were usually performed by specialized database administrators and required shutting down and reconfiguring the database. By building administrative functions into the query language, SQL helps to eliminate the database administrator as a choke point in application development.
4. SQL is designed for use in both decision support and online transaction processing environments. The former environment requires processing of complex queries, usually executed infrequently but accessing large amounts of data. The latter environment requires high-performance execution of parameterized transactions, repeated frequently but accessing (and often updating) small amounts of data. Both end-user interfaces and application programming interfaces are necessary to support this spectrum of usage.

The first specification of SQL was published in May 1974, in a 16-page conference paper [3] by Don Chamberlin and Ray Boyce, members of the System R project. In this paper, the language was named SEQUEL, an acronym for Structured English Query Language. The paper included a BNF syntax for the proposed language. This original paper presented only basic query features, without any facilities for data definition or update. However, the basic structure of the language, including query-blocks, grouping, set operations, and aggregating functions, has been consistent from this paper to the present day.

Over the course of the System R project, SEQUEL continued to evolve based on experience gathered by users and implementers. A much more complete description of the language was published in the IBM Journal of Research and Development in November 1976 [2], including data manipulation facilities (insert, delete, and update), a more complete join facility, facilities for defining tables and views, and database administration facilities including access control, assertions, and triggers. In 1977, because of a trademark issue, the name SEQUEL was shortened to SQL.

Although SQL was designed and prototyped at IBM Research, the language was published in the open literature, and the first commercial SQL product was released by a small company called Relational Software, Inc., in 1979. This product was named

Oracle, a name that was later adopted by the company, which is no longer small. The first IBM product based on SQL was called SQL/Data System, released in 1981, followed by DB2, released in 1983 on mainframes and eventually supported on many IBM platforms. SQL has now been implemented by all major database vendors and is available in a wide variety of operating environments. In addition to commercial database products, SQL implementations include several popular open-source products such as MySQL (http://www.mysql.com) and Apache Derby (http://db.apache.org/derby/).

Standards

Shortly after the first appearance of SQL in a commercial product, an effort was made to standardize the language. Over the years, the SQL standard has contributed to the growth of the database industry by defining a common interface for use by database vendors, application developers, and tools.

The first SQL standard, named "Database Language SQL," was published by the American National Standards Institute (ANSI) in 1986 (Standard No. X3.135-1986), and an identical standard with the same name was published by the International Standards Organization (ISO) in 1987 (Standard No. ISO 9075-1987). Over the years, ANSI and ISO have cooperated to keep their respective SQL standards synchronized as they have evolved through several versions.

The original standard, often called SQL-86, occupied just under 100 pages and included only simple queries, updates, and table definitions. It was followed in 1989 by a revised standard that added several kinds of constraints for protecting the integrity of stored data. This version of the standard comprised about 120 pages and is often referred to as SQL-89 ("Database Language SQL with Integrity Enhancement").

A major revision of the SQL standard, usually called SQL-92, was published by ANSI and ISO/IEC in 1992. This version improved the orthogonality of the language, allowing expressions to be used wherever tables or scalar values are expected. SQL-92 also added several new features, including date and time datatypes, set-oriented operators such as UNION and INTERSECT, standard catalog tables for storing metadata, and schema-evolution features such as ALTER TABLE. SQL-92 comprised about 600 pages. A conformance test suite for SQL-92 was developed in the United States by the National Institute of Standards and Technology (NIST). After certifying several conforming products, NIST discontinued SQL conformance testing in 1996.

Between 1992 and 1999, two specialized extensions to SQL-92 were published, named Call Level Interface (CLI) and Persistent Stored Modules (PSM). CLI defines a set of functions whereby programs written in languages such as C can dynamically connect to relational databases and execute SQL statements. PSM extends SQL with assignment statements, control-flow statements, and exception handlers, making it possible to implement some database applications entirely in SQL. PSM was an attempt to standardize the procedural extensions such as PL/SQL [9] and Transact-SQL [12] that had been added to SQL by several database vendors.

The next major update of the SQL standard occurred in 1999 and is usually called SQL:1999. This new version split the standard into several parts, incorporating CLI as Part 3 and PSM as Part 4. SQL:1999 introduced important new functionality including triggers, large objects, recursive queries, and user-defined functions. It placed major emphasis on object-relational functionality and on new features for on-line analytic processing (OLAP). Details of these and other recent additions to SQL are described below under "Advanced Features." The sum of all the parts of SQL:1999 exceeded 2,000 pages. Additional parts continue to be added to the SQL standard from time to time. The most recent major revision of the standard, known as SQL:2003, with all its parts, comprises more than 3,600 pages.

Over the years, the SQL standard has provided a controlled framework within which the language can evolve to correct its initial limitations and to meet changing user requirements. The standard has also served to focus the industry's attention and resources, providing a common framework in which individuals and companies could develop tools, write books, teach courses, and provide consulting services. The standard has been only partially successful in making SQL applications portable across implementations; this goal has been hampered by the fact that different vendors have implemented different subsets of the standard, and by the lack (since 1996) of a test suite to validate conformance of an implementation. The latest versions of the various parts of the standard can be obtained from ISO

[7] or from national standards organizations such as ANSI [8]. Jim Melton, editor of the SQL standard, has also published a two-volume reference book explaining the standard in a very accessible style [11, see also 10].

Foundations

Queries

SQL operates on data in the form of tables. Each table has a name and consists of one or more columns, each of which has a name and a datatype. The content of a table consists of zero or more rows, each of which has a value for each of the columns. The value associated with a given row and column may be an instance of the datatype of that column, or may be a special "null" value indicating that the value is missing (not available or not applicable). SQL statements, which may be queries or updates, operate on stored tables or on tabular "views" that are derived from stored tables. The result of a query is an unnamed virtual table. The result of an update is a change to the stored data, which is visible to subsequent statements. Generally, updates can be applied to a view only if each row in the view can be mapped uniquely onto a row of a stored table. This rule makes it possible to map updates on the view to updates on the underlying table.

An SQL query consists of one or more *query-blocks*. A query-block consists of several clauses, each of which begins with a keyword. Some of these clauses (keywords SELECT and FROM) are required, and others (keywords WHERE, GROUP BY, and HAVING) are optional. The examples in this article use upper-case keywords and lower-case names, although SQL is a case-insensitive language. The examples are based on two tables named PARTS and SUPPLIERS. The primary key of PARTS is PARTNO and the primary key of SUPPLIERS is SUPPNO. SUPPNO also appears as a foreign key in the PARTS table (see "Database Administration" below for definitions of primary key and foreign key.)

The following query-block illustrates a join of two tables. Conceptually, rows from the PARTS table are paired with rows from the SUPPLIERS table according to the criterion specified in the WHERE clause (SUPPNO's must match), and the resulting row-pairs are filtered by an additional condition (supplier's location must be Denver). From the surviving row-pairs, the SELECT clause specifies the columns that appear in the query result (in this case, the part number and the supplier name). Many different strategies are possible for executing this query; since SQL is a non-procedural language, choice of an execution strategy is left to an optimizing compiler.

```
SELECT p.partno, s.name
FROM parts p, suppliers s
WHERE p.suppno = s.suppno
AND s.location = 'Denver'
```

The following query-block illustrates grouping and aggregation. Conceptually, the rows of the PARTS table are partitioned into groups with matching SUPPNO values. The groups are then filtered by the HAVING clause, which applies a predicate based on group properties (in this case, retaining only groups that have at least ten rows.) Finally, the SELECT clause specifies the columns of the query result, which consists of one row for each group. In a grouping query, the selected expressions must consist of group properties (grouping keys or aggregating functions such as avg, sum, max, min, and count.) The SELECT clause can also specify names for the output columns. The query-block in this example returns, for each supplier that supplies at least ten parts, the average cost of all parts supplied by that supplier.

```
SELECT suppno, avg(cost) AS avgcost
FROM parts
GROUP BY suppno
HAVING count(*) >= 10
```

Many queries, including the above examples, consist of a single query-block. Query-blocks can also be combined to form larger queries. In the following example, a query-block serves as a subquery that computes a value used in another query-block. The subquery finds the maximum cost in the PARTS table, and the outer query-block returns the part numbers of the parts that have this maximum cost. The query-blocks are linked together by the IN keyword, denoting that the value in the WHERE-clause is tested for inclusion in the list of values returned by the subquery, which may in general return multiple values.

```
SELECT partno
FROM parts
WHERE cost IN
   (SELECT max(cost)
  FROM parts)
```

Another way in which query-blocks can be combined to form a larger query is by means of the set-operators UNION, INTERSECT, and EXCEPT, which compute the union, intersection, or difference of the sets of rows returned by two query-blocks (after eliminating duplicate rows). The datatypes of the rows returned by the respective query-blocks must be compatible. The following example computes the difference of two query-blocks to find the suppliers that supply no parts. It also includes an ORDER BY clause, which specifies an ordering for the rows in the query result. The ORDER BY clause applies to the whole query rather to an individual query-block.

```
  SELECT suppno
  FROM suppliers
EXCEPT
  SELECT suppno
  FROM parts
ORDER BY suppno
```

The first example above showed how the condition for joining two tables can be specified in a WHERE-clause. This method returns only data for which a matching row exists in both tables. Another join method called an *outer join* can be used to include rows from one of the joined tables that have no matching rows in the other table. The following example returns the supplier numbers of suppliers located in Denver, together with the parts that they supply, including Denver suppliers that supply no parts (the latter suppliers will appear in the query result with a null value in the PARTNO column):

```
SELECT s.suppno, p.partno
FROM suppliers s LEFT OUTER JOIN parts p
ON s.partno = p.partno
AND s.location = 'Denver'
```

Data Manipulation

The data manipulation facilities of SQL consist of the INSERT, DELETE, and UPDATE statements, which are used for inserting rows into a table, deleting rows from a table, and updating the values of rows in a table. The values to be used in data manipulation statements may be specified as constants or computed by subqueries, as illustrated by the following examples.

This example updates the PARTS table, increasing the cost of all the parts supplied by a certain supplier by 10%:

```
UPDATE parts
SET cost = cost * 1.1
WHERE suppno = '105'
```

The following example deletes from the SUPPLIERS table all the suppliers that supply no parts. This example illustrates a subquery that is linked to the outer DELETE statement by the keywords NOT EXISTS. Notice that the subquery depends on a value in a row supplied by the outer UPDATE statement (referred to as s.suppno); therefore the subquery must be executed repeatedly for each row in the table being updated. This kind of subquery is called a *correlated subquery.*

```
DELETE FROM suppliers s
WHERE NOT EXISTS
   (SELECT partno
  FROM parts
  WHERE suppno = s.suppno)
```

Database Administration

In addition to queries and data manipulation, SQL provides facilities for performing database administration tasks. These tasks fall into three general categories: data definition, access control, and active data features.

SQL data definition facilities include statements for creating or dropping tables and views, and for adding or deleting columns of existing tables. The definition of a view takes the form of an SQL query specifying how the view can be derived from stored tables and/or other views.

SQL access control facilities are based on the concepts of users, privileges, and roles. A privilege is the ability to perform an action (such as select, insert, delete, or update) on an object (such as a table or, in some cases, a column of a table). A role is a set of privileges that can be granted to a set of users. In general, the user who creates an object can grant privileges on that object to individual users and to roles, and can also revoke these privileges. When granting a privilege, the grantor can specify whether the grantee is authorized to pass along the privilege to additional users or roles.

SQL active data facilities include constraints and triggers. Constraints serve to enforce database integrity by limiting the kinds of updates that can be applied to a table. Important kinds of constraints include the following:

- NOT NULL constraints, which prohibit null values in a specific column.
- CHECK constraints, which specify a predicate that must not be false for any row of the table.

- PRIMARY KEY constraints, which require that the values in a specific set of columns called the *primary key* uniquely identify a row of the table.
- FOREIGN KEY constraints, which require that, for each row of the constrained table, a specific set of columns called the *foreign key* contain only combinations of values that are also found in the primary key of a related table called the *parent table*. The definer of a FOREIGN KEY constraint can specify what happens when a data manipulation statement attempts to violate the constraint (for example, the violating statement might have no effect; or deletion of a row from the parent table might cause the automatic deletion of related rows in the constrained table). Enforcement of foreign key constraints is said to protect the *referential integrity* of stored data.

A trigger is an action that is automatically invoked whenever a specific event, called the triggering event, occurs. The definer of a trigger specifies the following properties:

- The triggering event, which may be insert, delete, or update of rows (or, in some cases, columns) of a specific table.
- Whether the trigger is invoked before or after the triggering event becomes effective.
- If the triggering event affects multiple rows, whether the trigger is invoked once for each affected row or only once for the whole triggering event.
- An optional trigger condition: a predicate that must be true at the time of the triggering event in order for the trigger to be activated.
- The trigger body: one or more SQL statements that are automatically executed when the triggering event occurs and the trigger condition is true. The trigger condition and the trigger body have access to special variables that contain the data values before and after the triggering event.

Constraints and triggers are useful for specifying and enforcing the semantics of stored data. In general, it is preferable to specify a given semantic rule by means of a constraint rather than a trigger if possible, since constraints apply to all kinds of actions and provide maximum opportunities for optimization. On the other hand, some kinds of semantic rules (for example, "salaries never decrease") can only be specified by using triggers.

Advanced Features

Over the years, a great deal of functionality has been added to the SQL language. The full set of SQL features is far too large and complex to be explained here. The following are some of the major areas in which advanced functionality has been added to SQL:

- **Recursion:** A recursive query consists of an initial subquery that computes some preliminary results and a recursive subquery that computes additional results based on values that were previously computed. The recursive subquery is executed repeatedly until no additional results are computed. Recursion is useful in queries that search some space for an optimum result, such as "Find the cheapest combination of flight segments to travel from Shanghai to Copenhagen." Recursive queries were first defined in SQL:1999.
- **OLAP:** Online analytic processing (OLAP) is used by businesses to analyze large volumes of data to identify facts and trends that may affect business decisions. The GROUP BY clause and aggregating functions (sum, avg, etc.) of early SQL provided a primitive form of OLAP functionality, which was greatly extended in later versions of the language. For example, the ROLLUP facility enables a query to apply aggregating functions at multiple levels (such as city, county, and state). The CUBE facility enables data to be aggregated along multiple dimensions (such as date, location, and category) within a single query. The WINDOW facility allows aggregating functions to be applied to a "moving window" as it passes over a collection of data. These facilities, and others, were introduced by SQL:1999 and enhanced in subsequent versions of the standard.
- **Functions and procedures:** Originally, SQL supported a fixed collection of functions, which grew slowly over the years. SQL:1999 introduced a capability for users to define additional functions and procedures that can be invoked from SQL statements. (In this context, a procedure is simply a function that is invoked by a CALL statement and that is not required to return a value.) The bodies of user-defined functions and procedures can be written either in SQL itself or in a host language such as C or Java.
- **Object-relational features:** Early versions of SQL could process data conforming to a fixed set of

simple datatypes such as integers and strings. Over the years, a few additional datatypes such as dates and "large objects" were added. In the late 1990's, requirements arose for a more extensible type system. SQL:1999 introduced facilities for user-defined structured types and methods. These facilities support limited forms of object-oriented functionality, including inheritance and polymorphism.

- **Multi-media:** In 2000, the SQL Standard was augmented by a separate but closely-related standard called "SQL Multimedia and Application Packages" (ISO/IEC 13249:2000), often referred to as SQL/MM. This new standard used the object-relational features introduced by SQL:1999 to define specialized datatypes and methods for text, images, and spatial data.
- **XML-related features:** XML is an increasingly popular format for data exchange because it mixes metadata (tags) with data, making the data self-describing. The popularity of XML has led to requirements to store XML data in relational databases and to convert data between relational and XML formats. These requirements have been addressed by a facility called SQL/XML, which was introduced as Part 14 of SQL:2003 and was updated in 2006. SQL/XML includes a new XML datatype, a set of functions for converting query results into XML format, and a feature whereby SQL can invoke XQuery as a sublanguage for processing stored XML data.

Criticisms

Like most widely-used programming interfaces, SQL has attracted its share of criticism. Issues that have been raised about the design of SQL include the following:

- The earliest versions of SQL lacked support for some important aspects of Codd's relational data model such as primary keys and referential integrity. These concepts were added to the language in SQL-89, along with other integrity-related features such as unique constraints and check-constraints.
- The earliest versions of SQL had some ad-hoc rules about how various language features could be combined, and lacked the closure property because the columns of query results did not always have names. These problems were largely corrected by SQL-92.
- Null values are a complex and controversial subject. One of Codd's famous "twelve rules" requires relational database systems to support a null value, defined as a representation of missing or inapplicable information that is systematic and distinct from all regular values [6]. Some writers believe that the complexity introduced by null values outweighs their benefit. However, there seems to be no method for dealing with missing data that is free of disadvantages. The SQL approach to this issue has been to support a null value and to allow database designers to specify, on a column-by-column basis, where nulls are permitted. One benefit of this approach has been that null values have proven useful in the design of various language features, such as outer join, CUBE, and ROLLUP, that have been added during the evolution of SQL.
- Unlike Codd's definition of the relational data model, SQL permits duplicate rows to exist, either in a database table or in the result of a query. SQL also allows users to selectively prohibit duplicate rows in a table or in a query result. The intent of this approach is to give users control over the potentially expensive process of duplicate elimination. In some applications, duplicate rows may be meaningful (for example, in a point-of-sale system, a customer may purchase several identical items in the same transaction.) As in the case of nulls, the SQL approach has been to provide users with tools to allow or disallow duplicate rows according to the needs of specific applications.
- Another source of criticism has been the "impedance mismatch" between SQL and the host languages such as C and Java in which it is often embedded. Exchanging data between two languages with different type systems makes applications more complex and interferes with global optimization. One approach to this problem has been the development of computationally complete SQL-based scripting languages such as PSM.

Key Applications

SQL is designed to be used in a variety of application environments.

Most SQL implementations support an interactive interface whereby users can compose and execute ad-hoc SQL statements. In many cases, a graphical user interface is provided to display menus of available tables and columns and help the user to construct valid statements. These systems also usually support menu-based interfaces for administrative functions such as creating and dropping tables and views.

More complex applications usually involve use of both SQL and a host programming language. This requires a mapping between the type systems of SQL and the host language, and a well-defined interface for exchanging data between the two environments. Interfaces have been defined between SQL and C, Java, and many other host languages. These interfaces fall into two major categories:

- **Embedded SQL:** In this approach, SQL statements are embedded syntactically in the host program, and are processed by a precompiler that extracts the SQL statements and converts them to an optimized execution plan that is invoked when the host program is executed.
- **Call interfaces:** In this approach, the host program uses function calls to establish a connection to a database and to pass SQL statements to the database system for execution. Conversion of the SQL statements to optimized execution plans is done at runtime. The most widely-used interfaces of this type are ODBC (Open Database Connectivity) [13] and JDBC (Java Database Connectivity) [14]. Since ODBC and JDBC drivers exist for many popular relational database systems, these interfaces are widely used by applications that need to access remote databases or need to be compatible with database systems from multiple vendors.

Modern web-based database applications often employ a three-tiered architecture. The *client tier* handles user interaction and communicates with a remote server via a protocol such as HTTP. Application logic resides on this server, called the *mid-tier*. The application logic, in turn, accesses an SQL database using ODBC or JDBC. The database may be implemented on the mid-tier or on a separate machine called the *database tier* or “back-end.”

Cross-references

▶ Computationally Complete Relational Query Languages
▶ Database Trigger
▶ Expressive Power of Query Languages
▶ Java Database Connectivity
▶ Join
▶ Key
▶ Metadata
▶ Multi-Tier Architecture
▶ Null Values
▶ On-Line Analytical Processing
▶ Open Database Connectivity
▶ Query Language
▶ Query Optimization
▶ Query Processing
▶ Relational Algebra
▶ Relational Calculus
▶ Relational Integrity Constraints
▶ Relational Model
▶ SQL Isolation Levels
▶ SQUARE
▶ Transaction
▶ Views
▶ XPath/XQuery

Recommended Reading

1. Astrahan M.M. et al. System R: A relational approach to database management. ACM Trans. Database Syst., 1(2):97–137, 1976.
2. Chamberlin D. et al. SEQUEL 2: A unified approach to data definition, data manipulation, and control. IBM J. Res. Develop., 20(6):560–575, 1976.
3. Chamberlin D. and Boyce R. SEQUEL: A structured english query language. In Proc. ACM SIGFIDET Workshop, 1974, pp. 249–264.
4. Codd E.F. A relational model of data for large shared databanks. Commun. ACM, 13(6):377–387, 1970.
5. Codd E.F. A data base sublanguage founded on the relational calculus. In Proc. ACM SIGFIDET Workshop, 1971, pp. 35–68.
6. Codd E.F. Does Your DBMS Run by the Rules? ComputerWorld, 21 Oct., 1985. See also http://en.wikipedia.org/wiki/Codd’s_12_rules.
7. Database Language SQL. Standard ISO/IEC 9075-1, -2, etc. Available at: http://www.iso.ch.
8. Database Language SQL. Standard ANSI/ISO/IEC 9075-1, -2, etc. Available at: http://www.ansi.org.
9. Feuerstein S. and Pribyl B. Oracle PL/SQL Programming, O’Reilly, Sebastopol, CA, 2005.
10. Melton J. Advanced SQL:1999–Understanding Object-Relational and Other Advanced Features. Morgan Kaufmann, San Fransisco, CA, 2003.
11. Melton J. and Simon A.R. SQL:1999–Understanding Relational Language Components. Morgan Kaufmann, San Fransisco, CA, 2002.
12. Microsoft SQL Server Development Center. Transact-SQL Reference. http://msdn2.microsoft.com/en-us/library/ms189826.aspx.
13. Sanders R.E. ODBC 3.5 Developer’s Guide. McGraw-Hill, NY, 1998.
14. Sun Developer Network. JDBC Overview. http://java.sun.com/products/jdbc/overview.html.

SQL Isolation Levels

PHILIP A. BERNSTEIN
Microsoft Corporation, Redmond, WA, USA

Synonyms

Degrees of consistency; Degrees of isolation

Definition

A transaction is an execution of a well-defined set of read and write operations on shared data, which terminates with a commit operation that makes its updates permanent, or an abort operation that undoes its updates. Isolation levels define the situations in which a transaction can be affected by the execution of other transactions. In the ACID properties, isolation requires that transactions behave serializably, that is, as if they executed in a serial order with no interleaving. To obtain a serializable execution when many transactions are executing concurrently, a transaction's operations may be delayed and occasionally even rejected. This reduces the rate at which transactions execute. Users often regard this throughput reduction as unsatisfactory and therefore seek isolation levels that are less stringent than serializability, some of which are defined as part of the SQL language. These are the SQL Isolation Levels, which are called Read Uncommitted, Read Committed, Repeatable Read, and Serializable. They are derived primarily from the "degrees of consistency" originally presented in [3].

Key Points

The SQL isolation levels are defined in terms of the following types of interleavings, P1 – P3, that each isolation allows or prohibits. The interleavings are expressed by sequences of operations, using the notation $r_1[x]$ (respectively, $w_1[x]$) to represent the execution of a read (respectively write) operation by transaction T_1 on row x.

P1. Dirty Read – A dirty read is an operation that reads a value that was written by an uncommitted transaction. In the execution "$w_1[x]$ $r_2[x]$," $r_2[x]$ performs a dirty read. The problem is that transaction T_1 might abort after T_2 read row x, in which case T_2 has read a value of x that never existed.

P2. Non-repeatable Read – A transaction reads a row x, a second transaction writes into x and commits, and then the first transaction reads x again. In the execution "$w_0[x]$ $r_1[x]$ $w_2[x]$ c_2 $r_1[x]$," transaction T_1 has experienced a non-repeatable read, since it read one value of x before T_2 executed and a different value of x after T_2 committed.

P3. Phantom – A transaction T_1 reads a set of rows that satisfy a predicate P (such as a WHERE-clause in SQL). Before T_1 commits or aborts, a second transaction T_2 inserts, updates, or deletes rows that change the set of rows that satisfy P. Therefore, if T_1 re-executes its read, the read will return a different set of rows than it returned the first time. The rows that appear and disappear are called phantoms. This is the same situation as non-repeatable reads, except that the *set* of rows that T_1 retrieves is affected by T_2, not just their value. In the following execution, the row inserted by w_2 is a phantom:

- r_1[rows of the Employee table where Department = "Toy"]
- w_2[insert a new row in the Employee table where Department = "Toy"]
- r_1[rows of the Employee table where Department = "Toy"]

The SQL isolation levels are defined using the above three phenomena:

1. Read Uncommitted – All three phenomena (P1, P2, and P3) are allowed. The intent is to allow all interleavings of reads and writes by different transactions.
2. Read Committed – Dirty reads are prohibited. This ensures that each read operation reads a value that will not be undone because the transaction that wrote the value later aborts.
3. Repeatable Read – Dirty reads and non-repeatable reads are prohibited. The intent is to ensure that transaction executions are serializable except for phantom situations that arise.
4. Serializable – All three phenomena are prohibited. The intent is to ensure that transactions are truly serializable.

Each transaction may independently define its isolation level. This creates some difficulty in how a user should interpret the levels. For example, if a transaction runs as Serializable, it may still read data that was written by transactions running (say) as Read Committed.

SQL isolation levels have been criticized because there is a gap between the definition of the phenomena they prevent, and the intent of the isolation level as presented in the descriptions of (1) – (4) above. For details, see [1].

In principle, all transactions that perform updates should execute at the Serializable level. That way, if each transaction preserves database consistency, then each serial execution will preserve consistency. Hence a serializable execution will too. If update transactions execute at less than serializable level, then this consistency preservation guarantee is lost. Nevertheless, it is believed that most database applications execute at lower isolation levels than Serializable, typically Read Committed. This gives them higher throughput at the expense of some loss of correctness. One simulation study of transaction performance showed that transaction throughput is 2½–3 times higher with transactions executing at Read Committed level compared to Serializable level [2].

Cross-references

- ▶ ACID Properties
- ▶ Serializability
- ▶ Transaction
- ▶ Transaction Model

Recommended Reading

1. Berenson H., Bernstein P., Gray J., Melton J., O'Neil E., and O'Neil P. A critique of ANSI SQL isolation levels. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 1–10.
2. Bober P.M. and Carey M.J. On mixing queries and transactions via multiversion locking. In Proc. 8th Int. Conf. on Data Engineering, 1992, pp. 548–556.
3. Gray J., Lorie R.A., Potzulo G.R., and Traiger I.L. Granularity of locks and degrees of consistency in a shared database. In IFIP Working Conf. on Modelling in Data Base Management Systems, 1976, pp. 365–394. Reprinted in Readings in Database Systems (3rd edn.), M. Stonebraker and J. Hellerstein (eds.). Morgan Kaufmann, 1998, pp. 175–193.
4. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993, pp. 397–403.

SQL-Based Temporal Query Languages

Michael Böhlen[1], Johann Gamper[1],
Christian S. Jensen[2], Richard T. Snodgrass[3]
[1]Free University of Bozen-Bolzano, Bolzano, Italy
[2]Aalborg University, Aalborg, Denmark
[3]University of Arizona, Tucson, AZ, USA

Definition

More than two dozen extensions to the relational data model have been proposed that support the storage and retrieval of time-referenced data. These models timestamp tuples or attribute values, and the timestamps used include time points, time periods, and finite unions of time periods, termed temporal elements.

A temporal query language is defined in the context of a specific data model. Most notably, it supports the specification of queries on the specific form of time-referenced data provided by its data model. More generally, it enables the management of time-referenced data.

Different approaches to the design of a temporal extension to the Structured Query Language (SQL) have emerged that yield temporal query languages with quite different design properties.

Historical Background

A number of past events and activities that included the temporal database community at large had a significant impact on the evolution of temporal query languages. The 1987 *IFIP TC 8/WG 8.1 Working Conference on Temporal Aspects in Information Systems* [7] covered topics such as requirements for temporal data models and information systems, temporal query languages, versioning, implementation techniques, as well as temporal logic, constraints, and relations to natural language.

The 1993 *ARPA/NSF International Workshop on an Infrastructure for Temporal Databases* [8] gathered researchers in temporal databases with the goal of consolidating the different approaches to temporal data models and query languages. In 1993, the influential collection *Temporal Databases: Theory, Design, and Implementation* [11] was also published. This collection describes a number of data models and query languages produced during the previous 10 years of temporal database research.

Year 1995 saw the publication of the book *The TSQL2 Temporal Query Language* [9]. TSQL2 represents an effort to design a consensus data model and query language, and it includes many of the concepts that were proposed by earlier temporal data models and query languages. In 1995, the *International Workshop on Temporal Databases* [3] was co-located with the VLDB conference.

Then, in 1996, *SQL/Temporal: Part 7 of SQL3* was accepted. This was the result of an effort aimed at transferring results of temporal database research into SQL3. The first step was a proposal of a new part to SQL3, termed SQL/Temporal, which included the PERIOD data type. In 1997, the *Dagstuhl Seminar* *on Temporal Databases* took place [5]. Its goal was to discuss future directions for temporal database management, with respect to both research issues and the means to incorporate temporal databases into mainstream application development.

Foundations

A discrete and totally ordered time domain is assumed that consists of time instants/points. The term (time) "period" is used to denote a convex subset of the time domain. The term (time) "interval" then denotes a duration of time, which coincides with its definition in SQL. As a running example, the temporal relation *Rental* in Fig. 1(a) is used, which records car rentals, e.g., customer C101 rents vehicle V1234 from time 3 to time 5. Figures 1(b)–(e) show different representations of this relation, using the strong period-based, weak period-based, point-based, and parametric model, respectively. (In all the example relation instances, the conventional attribute(s) are separated from the timestamp attribute(s) with a vertical line.) The following queries together with their intended results build on the car rental example. These will serve for illustration.

Q1: *All rentals that overlap the time period* [7,9]. Query Q1 asks for all available information about rentals that overlap the period [7,9].

CustID	VID	T
C102	V1245	[5,7]
C102	V1234	[9,12]

Q2: *All 2-day rentals.* This query constrains the number of time points included in a time period and teases out the difference between the use of time points versus periods.

VID	T
V1245	[19,20]
V1245	[21,22]

Q3: *How many vehicles have been rented?* This is an example of an ordinary query that must be applied to each state of a temporal database. The non-temporal query is an aggregation. Thus, the result at a specific time point is computed over all tuples that are valid at that time point. (Note that some query languages don't return tuples when there is no data, e.g., when the `Cnt` is 0.)

Cnt	T
1	[3,4]
2	[5,5]
1	[6,7]
0	[8,8]
1	[9,12]
0	[13,18]
1	[19,20]
1	[21,22]

Q4: *How many rentals were made in total?* This is another aggregation query; however, the aggregation is to be applied independently of any temporal information.

Cnt
5

Q5: *List all (current) rentals.* This query refers to the (constantly moving) current time. It is assumed that the current time is 5.

CustID	VID
C101	V1234
C102	V1245

Approach I: Abstract Data Types – SQL/ATD

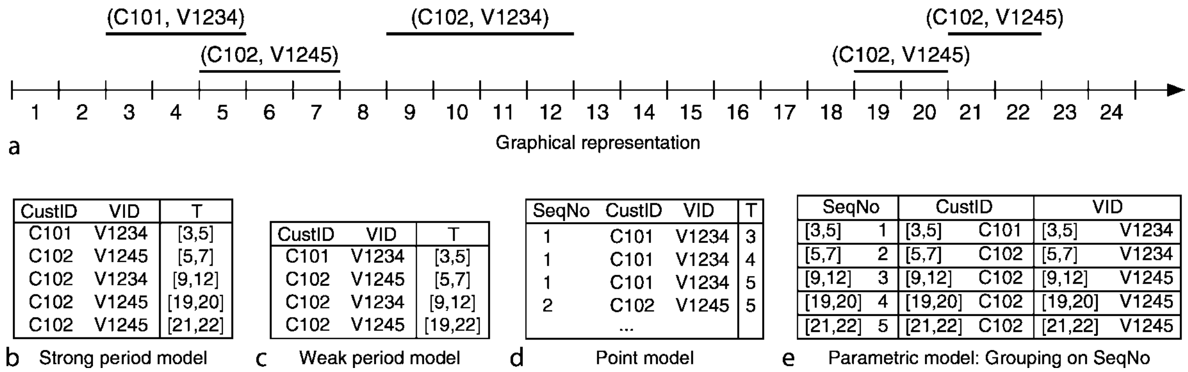
The earliest and, from a language design perspective, simplest approaches to improving the temporal data management capabilities of SQL have simply introduced time data types and associated predicates and functions. This approach is illustrated on the `Rental` instance in Figure 1(b).

$Q1^{\text{SQL/ATD}}$: `select * from Rental where T overlaps [7,9]`

$Q2^{\text{SQL/ATD}}$: `select VID, T from Rental where duration(T) = 2`

$Q4^{\text{SQL/ATD}}$: `select count(*) as Cnt from Rental`

$Q5^{\text{SQL/ATD}}$: `select CustID, VID from Rental where T overlaps [now,now]`



CustID	VID	T
C101	V1234	[3,5]
C102	V1245	[5,7]
C102	V1234	[9,12]
C102	V1245	[19,20]
C102	V1245	[21,22]

b Strong period model

CustID	VID	T
C101	V1234	[3,5]
C102	V1245	[5,7]
C102	V1234	[9,12]
C102	V1245	[19,22]

c Weak period model

SeqNo	CustID	VID	T
1	C101	V1234	3
1	C101	V1234	4
1	C101	V1234	5
2	C102	V1245	5
	...		

d Point model

SeqNo	CustID	VID
[3,5] 1	[3,5] C101	[3,5] V1234
[5,7] 2	[5,7] C102	[5,7] V1234
[9,12] 3	[9,12] C102	[9,12] V1245
[19,20] 4	[19,20] C102	[19,20] V1245
[21,22] 5	[21,22] C102	[21,22] V1245

e Parametric model: Grouping on SeqNo

SQL-Based Temporal Query Languages. Figure 1. Temporal relation *rental*.

The predicates on time-period data types available in query languages have been influenced by Allen's 13 period relationships [1], and different practical proposals for collections of predicates exist. For example, the overlaps predicate (as defined in the TSQL2 language) can be used to formulate Query Q1. Predicates that limit the duration of a period (Q2) and retrieve current data (Q5) follow the same approach.

Expressing the time-varying aggregation of Q3 in SQL is possible, but exceedingly complicated and inefficient. The hard part is that of expressing the computation of the periods during which the aggregate values remain constant. (This requires about two dozen lines of SQL with nested NOT EXISTS subqueries [10, pp. 165–166].) In contrast, counting the rentals independently of the time references is easy, as shown in Q4.

Adding a new ADT to SQL has limited impact on the language design, and extending SQL with new data types with accompanying predicates and functions is relatively simple and fairly well understood. The approach falls short in offering means of conveniently formulating a wide range of queries on period timestamped data, including temporal aggregation. It also offers no systematic way of generalizing a simple snapshot query to becoming time-varying. Shortcomings such as these motivate the consideration of other approaches.

Approach II: Folding and Unfolding – IXSQL

Another approach is to equip SQL with the ability to normalize timestamps. Advanced most prominently by Lorentzos [4,6] in the IXSQL language, the earliest and most radical approach is to introduce two functions: unfold, which decomposes a period-timestamped tuple into a set of point-timestamped tuples, and fold, which "collapses" a set of point-timestamped tuples into value-equivalent tuples timestamped with maximum periods. The general pattern for queries is then: (i) construct the point-based representation by unfolding the argument relation(s), (ii) compute the query on the period-free representation, and (iii) fold the result to obtain a period-based representation. The Rental relation in Fig. 1(b) is assumed.

```
Q3^IXSQL :select count(*) as Cnt, T
          from (select * from Rental
          reformat as unfold T)
          group by T reformat as fold T
```

The IXSQL formulations of Q1, Q2, Q4, and Q5 are essentially those of the ADT approach (modulo minor syntactic differences); specifically, normalization is not needed. The fold and unfold functions become useful for the temporal aggregation in Q3. The inner query unfolds the argument relation, yielding the point-based representation in Fig. 1(d), on which the aggregation is computed. The fold function then transforms the result back into a period-stamped relation, which, however, is different from the intended result because the last two tuples are merged into a single tuple (1,[19,22]). The combination of unfolding and folding yields maximal periods of snapshot equivalent tuples and does not carry over any lineage information.

SQL with folding and unfolding is conceptually simple and offers a systematic approach to formulating at least some temporal queries, including temporal queries that generalize non-temporal queries. It obtains the representational benefits of periods while avoiding the potential problems they pose in query formulation, since the temporal data are manipulated in point-stamped form. The fold and unfold functions preserve the information content in a relation only up to that captured by the point-based perspective; thus, lineage information is lost. This leaves some "technicalities" (which are tricky at times) to be addressed by the application programmer.

Approach III: Point Timestamps – SQL/TP

A more radical approach to designing a temporal query language is to simply assume that temporal relations use point timestamps. The temporal query language SQL/TP advanced by Toman [12] takes this approach to generalizing queries on non-temporal relations to apply to temporal relations. The point-timestamped Rental relation in Fig. 1(d) is assumed in the following.

```
Q1^SQL/TP: select distinct a.* from Rental a,
           Rental b where a.SeqNo = b.SeqNo
           and or (b.T = 7 or b.T = 8 or b.T = 9)
Q2^SQL/TP: select SeqNo, VID, T from Rental
           group by SeqNo having count(T) = 2
Q3^SQL/TP: select count(*) as Cnt, T
           from Rental group by T
Q4^SQL/TP: select count(distinct SeqNo) as Cnt
           from Rental
Q5^SQL/TP: select CustID, VID from Rental
           where T = now
```

Q1 calls for a comparison of neighboring database states. The point-based perspective, which separates the database states, does not easily support such queries, and a join is needed to report the original rental periods. The distinct keyword removes duplicates that are introduced if a tuple shares more than one time point with the period [7,9].

Duration queries, such as Q2, are formulated as aggregations and require an attribute, in this case SeqNo, that distinguishes the individual rentals. The strength of SQL/TP is in its generalization of queries on snapshot relations to queries on temporal relations, as exemplified by Q3. The general principle is to extend the snapshot query to separate database snapshots, which here is done by the grouping clause. SQL/TP and SQL are opposites when it comes to the handling of temporal information. In SQL, *time-varying* aggregation is poorly supported, while SQL/TP needs an additional attribute that identifies the real-world facts in the argument relation to support *time-invariant* aggregation (Q4).

The restriction to time points ensures a simple and well-defined semantics that avoids many of the pitfalls that can be attributed to period timestamps. As periods are still to be used in the physical representation and user interaction, one may think of SQL/TP as a variant of IXSQL where, conceptually, queries must always apply unfold as the first operation and fold as the last. To express the desired queries, an identifying attribute (e.g., SeqNo) is often needed. Such identifiers do not offer a systematic way of obtaining point-based semantics *and* a semantics that preserves the periods of the argument relations. The query "*When was vehicle V1245, but not vehicle V1234, rented?*" illustrates this point. A formulation using the temporal difference between the timestamp attributes does not give the expected answer {[6,7],[19,20],[21,22]} because the sequence number is not included. If the sequence number is included, the difference is effectively disabled. This issue is not only germane to SQL/TP, but applies equally to all approaches that use a point-based data model.

Approach IV: Syntactic Defaults – TSQL2

What may be viewed as syntactic defaults have been introduced to make the formulation of common temporal queries more convenient. The most comprehensive approach based on syntactic defaults is TSQL2 [9]. As TSQL2 adopts a point-based perspective, the Rental instance in Fig. 1(c) is assumed, where the periods are a shorthand representation of time points.

```
Q1^TSQL2: select * from Rental
          where valid(Rental) overlaps
          period '7-9'
Q2^TSQL2: select SeqNo, VID from Rental
          where cast(valid(Rental) as
          interval) = 2
Q3^TSQL2: select count(*) as Cnt from Rental
          group by valid(Rental) using instant
Q4^TSQL2: select snapshot count(*) as Cnt
          from Rental
Q5^TSQL2: select snapshot * valid(date 'now')
          from Rental
```

In TSQL2, a valid clause, which by default is present implicitly after the select clause, computes the intersection of the valid times of the relations in the from clause, which is then returned in the result. With only one relation in the from clause, this default clause yields the original timestamps as exemplified in Q1 and Q2. The cast function in Q2 maps between periods (e.g., [7–9]) and intervals (e.g., 3 days). The argument relation must be augmented by the SeqNo attribute (thus obtaining a relation with five tuples, as in Fig. 1(b)) for this query to properly return the 2-day rentals.

The default behavior of the implicit valid clause was designed with snapshot reducibility in mind, which shows nicely in the instant temporal aggregation query Q3. The grouping is performed according to the time points, not the original timestamps returned by valid(Rental). The using instant is in fact the default and could be omitted (added for clarity). As TSQL2 returns temporal relations by default, the snapshot keyword is used in queries Q4 and Q5 to retrieve non-temporal relations.

Well-chosen syntactic defaults yield a language that enables succinct formulation of common temporal queries. However, adding temporal support to SQL in this manner is difficult since the non-temporal constructs do not permit a systematic and easy way to express the defaults. It is challenging to be comprehensive in the specification of such defaults, and to ensure that they do not interact in unattractive ways. Thus, syntactic defaults lack "scalability" over language constructs.

Approach V: Statement Modifiers – ATSQL

ATSQL [2] introduces temporal statement modifiers to offer a systematic means of constructing temporal queries from non-temporal queries. A temporal query is formulated by first formulating the corresponding non-temporal query, and then prepending this query with a statement modifier that tells the database system to use temporal semantics. In contrast to syntactic defaults, statement modifiers are semantic in that they apply in the same manner to any statement they modify. The strong period-timestamped Rental instance in Fig. 1(b) is assumed in the following.

$Q1^{ATSQL}$: `seq vt select * from Rental`
`where T overlaps [7,9]`

$Q2^{ATSQL}$: `seq vt select VID from Rental`
`where duration(T) = 2`

$Q3^{ATSQL}$: `seq vt select count(*) as Cnt from`
`Rental`

$Q4^{ATSQL}$: `nseq vt select count(*) as Cnt`
`from Rental`

$Q5^{ATSQL}$: `select * from Rental`

Queries Q1 and Q2 can be formulated almost as in SQL. The seq vt ("sequenced valid time") modifier indicates that the semantics is consistent with evaluating the non-temporal query on a sequence of non-temporal relations, and ensures that the original timestamps are returned. Modifiers also work for queries that use period predicates, such as, e.g., Allen's relations, which cannot be used in languages of point-timestamped data models.

Query Q3 is a temporal generalization of a non-temporal query and can be formulated by prepending the non-temporal SQL query with the seq vt modifier. The modifier ensures that at each time point, the aggregates are evaluated over all tuples that overlap with that time point. Query Q4 is to be evaluated independently of the time attribute values of the tuples. This is achieved by using the nseq vt ("non-sequenced valid time") modifier, which indicates that what follows should be treated as a regular SQL query.

A query without any modifiers considers only the current states of the argument relations, as exemplified by Query Q5. This ensures that legacy queries on non-temporal relations are unaffected if the non-temporal relations are made temporal.

Statement modifiers are orthogonal to SQL and adding them to SQL represents a much more fundamental change to the language than, e.g., adding a new ADT or syntactic defaults. The notion of statement modifiers offers a wholesale approach to rendering a query language temporal: modifiers control the semantics of any query language statement. This language mechanism is independent of the syntactic complexity of the queries that the modifiers are applied to. It becomes easy to construct temporal queries that generalize snapshot queries.

Approach VI: Temporal Expressions – TempSQL

The notion of temporal expression was originally advocated by Gadia and is supported in the TempSQL language [11, p. 28ff], which is based on the parametric data model (see Fig. 1(e)). Relations in TempSQL consist of tuples with attribute values that are functions from a subset of the time domain to some value domain (specified as a pair of a temporal element, a finite union of time periods, and a value). The functions in the same tuple must have the same domain. The relations are keyed. If a set of attributes is a key, then no two tuples are allowed to exist in the relation that have the same range values for those attributes. Fig. 1(e) with the key `SeqNo` is assumed in the following.

$Q1^{TempSQL}$: `select * from Rental`
`where [[VID]]` $\cap$ `[7,9]` $\neq \emptyset$
$Q2^{TempSQL}$: `select VID from Rental`
`where duration([[VID]]) = 2`
$Q3^{TempSQL}$: `select count(*) as Cnt from Rental`
$Q5^{TempSQL}$: `select * from Rental`

Queries Q1 and Q2 can be formulated using temporal expressions. If X is an expression that returns a function from time to some value domain then $[[X]]$ is a temporal expression which returns the domain of X, i.e., the time when X is true. The result of Q2 is the relation $\{\langle[19,22]\ \text{V1245}\rangle\}$. For the aggregation query Q3, TempSQL automatically performs an instant temporal aggregation [11, p. 42]. A different query must be used to determine the time-invariant count in Q4. One possibility would be to formulate a query that first drops or equalizes all timestamps and then performs the above aggregation. For so-called current users, TempSQL offers built-in support for accessing the current state of a database, by assuming that the argument relations are the ordinary snapshot relations that contain the current states of the temporal relations. This is exemplified in Q5.

Temporal expressions as used in TempSQL, which return the temporal elements during which a logical expression is true, are convenient and often enable the elegant formulation of queries. Temporal expressions along with temporal elements fit well into the point-based framework. However, as of yet, little research has been done to further explore temporal expressions and to include them into query languages.

Key Applications

SQL-based temporal query languages are intended for use in database applications that involve the management of time-referenced data. Such applications are found literally in all data management application areas – in fact, virtually all real-world databases contain time-referenced data. SQL-based languages are attractive in comparison to other types of languages because SQL is used by existing database management systems.

Future Directions

While temporal query language support appears to be emerging in commercial systems, comprehensive temporal support is still not available in products.

Much research in temporal query languages has implicitly or explicitly assumed a traditional administrative data management setting, as exemplified by the car rental example. The design of temporal query languages for other kinds of data and applications, e.g., continuous sensor data, has received little attention.

Cross-references

▶ Allen's Relations
▶ Now in Temporal Databases
▶ Period-Stamped Temporal Models
▶ Point-Stamped Temporal Models
▶ Temporal Database
▶ Temporal Data Models
▶ Temporal Element
▶ Time Interval
▶ Time Period
▶ Temporal Query Languages
▶ TSQL2
▶ Valid Time

Recommended Reading

1. Allen J.F. Maintaining Knowledge about temporal intervals. Commun. ACM, 26 (11):832–843, 1983.
2. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. ACM Trans. on Database Syst., 25(4):407–456, 2000.
3. Clifford J. and Tuzhilin A. editors. Recent advances in temporal databases. In Proc. Int. Workshop on Temporal Databases, 1995.
4. Date C.J., Darwen H., and Lorentzos N. editors. Temporal Data and the Relational Model. Morgan Kaufmann publishers, 2002.
5. Etzion O., Jajodia S., and Sripada S. editors. Temporal Databases: Research and Practice, Volume 1399 of Lecture Notes in Computer Science. Springer Verlag, 1998.
6. Lorentzos N.A. and Johnson R.G. Extending relational algebra to manipulate temporal data. Inf. Syst., 13(3):289–296, 1988.
7. Rolland C., Bodart F., and Lèonard M. editors. Temporal aspects in information systems. In Proc. IFIP TC 8/WG 8.1 Working Conf. on Temporal Aspects in Information Systems, 1987.
8. Snodgrass R.T. editor. In Proc. Int. Workshop on an Infrastructure for Temporal Databases, 1993.
9. Snodgrass R.T. editor. The TSQL2 Temporal Query Language. Kluwer, 1995.

10. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann Publishers, San Francisco, CA, July 1999.
11. Tansel A., Clifford J., Gadia S., Jajodia S., Segev A., and Snodgrass R.T. Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, Publishing Company, Inc. 1993.
12. Toman D. Point-based temporal extensions of SQL and their efficient implementation. In [5], 1997, pp. 211–237.

SRM

► Storage Resource Management

Stability-based Validation of Clustering

► Clustering Validity

Stable Distribution

Ping Li
Cornell University, Ithaca, NY, USA

Synonyms

Lévy skew α-stable distribution

Definition

A random variable Z is said to follow a symmetric α-stable distribution[13,15], where $0 < \alpha \leq 2$, if the Fourier transform of its probability density function $f_Z(z)$ satisfies

$$\int_{-\infty}^{\infty} e^{\sqrt{-1}zt} f_Z(z)dt = e^{-d|t|^{\alpha}}, \quad 0 < \alpha \leq 2 \qquad (1)$$

where $d > 0$ is the scale parameter. This is denoted by $Z \sim S(\alpha, d)$.

There is an equivalent definition. A random variable Z follows a symmetric α-stable distribution if, for any real numbers, C_1 and C_2,

$$C_1Z_1 + C_2Z_2 \overset{d}{=} (|C_1|^{\alpha} + |C_2|^{\alpha})^{1/\alpha} Z, \qquad (2)$$

where Z_1 and Z_2 are independent copies of Z, and the symbol "$\overset{d}{=}$" denotes equality in distribution.

The probability density function $f_Z(z)$ can be obtained by taking inverse Fourier transform of 1. In particular, $f_Z(z)$ can be expressed in closed-forms when $\alpha = 2$ (i.e., the normal distribution) and $\alpha = 1$ (i.e., the Cauchy distribution).

Historical Background

The early comprehensive development of the stable distribution theory is credited to the French mathematician P. Lévy and the Soviet mathematician A. Ya. Khinchin, in the 1920s and 1930s[13,15]. Stable distributions have been widely used for modeling real-world data which exhibit *heavy-tailed* behaviors, for example, data that do not have finite variance or even finite mean.

In databases, data mining, and theoretical computer science, stable distributions have become an important tool for developing randomized dimension reduction algorithms, in particular, for efficiently computing summary statistics including distances, frequency moments, inner products and angles, in massive dynamic data sets. This type of technique is called *(stable) random projections*[1–4,7–9,11,14].

Pioneered by Alon, Matias, and Szegedy [2] in 1996, the method of *random projections* has been extensively applied in databases for approximating joint sizes and (l_2) Euclidean distances. Indyk and Motwani [8] in 1998 developed *local sensitive hashing (LSH)* using random projections, for efficiently searching for -approximate nearest neighbors in high-dimensional data. In 2004, Datar et al. [4] extended LSH for approximating nearest neighbors in the l_α ($0 < \alpha \leq 2$) norm. Indyk[7] in 2000 proposed using stable random projections to approximate the l_α distances and frequency moments in massive data streams. Cormode et al. [3] in 2002 proposed approximating the (l_0) Hamming distances in dynamic data using stable random projections with very small α.

The method of stable random projections eventually boils down to estimating the scale parameter of an α-stable distribution for a fixed α. This problem has been studied in statistics. For example, Fama and Roll[6] suggested estimators based on sample quantiles. Ping Li [11] in 2008 developed estimators based on the geometric mean and the harmonic mean.

Foundations

From the definitions (1) and (2), it follows that, if D random variables, r_1, r_2,...,r_D, are independent and identically distributed (i.i.d.), $r_i \sim S(\alpha, 1)$, then a linear combination of r_i's also follows an α-stable distribution. That is

$$c_1r_1 + c_2r_2 + \ldots + c_Dr_D \sim \\ S(\alpha, |c_1|^\alpha + |c_2|^\alpha + \ldots + |c_D|^\alpha), \tag{3}$$

for any real numbers c_1, c_2,...,c_D. This property is the foundation for stable random projections.

Stable Random Projections

The basic procedure of stable random projection is quite straightforward. Consider two D-dimensional vectors, $u_1 \in \mathbb{R}^D$ and $u_2 \in \mathbb{R}^D$. A matrix $\mathbf{R} = \{r_{ij}\}_{i=1}^{D}{}_{j=1}^{k} \in \mathbb{R}^{D\times k}$ is generated by sampling the entries from i.i.d. α-stable distributions, i.e., $r_{ij} \sim S(\alpha, 1)$. The matrix-vector multiplications, $v_1 = \mathbf{R}^{\mathrm{T}} \times u_1$ and $v_2 = \mathbf{R}^{\mathrm{T}} \times u_2$, result in two k-dimensional vectors, $v_1 \in \mathbb{R}^k$ and $v_2 \in \mathbb{R}^k$. The entries of v_1 and v_2 also follow α-stable distributions:

$$v_{1,j} \sim S\left(\alpha, \sum_{i=1}^{D} |u_{1,i}|^\alpha\right), \text{ i.i.d.} \quad \mathrm{j} = 1, 2,\ldots,\mathrm{k} \tag{4}$$

$$v_{2,j} \sim S\left(\alpha, \sum_{i=1}^{D} |u_{2}, i_{|}^{\alpha}\right), \text{ i.i.d.} \quad \mathrm{j} = 1, 2,\ldots,\mathrm{k} \tag{5}$$

$$v_{1,j} - v_{2,j} \sim S\left(\alpha, \sum_{i=1}^{D} |u_{1,i} - u_{2,i}|^\alpha\right), \text{ i.i.d.} \\ j = 1, 2,\ldots,k \tag{6}$$

The term, $\sum_{i=1}^{D} |u_{1,i}|^\alpha$, is the l_α norm (raised to the αth power) of the vector u_1; and in data stream computations, it is often referred to as the αth *frequency moment*. The term, $\sum_{i=1}^{D} |u_{1,i} - u_{2,i}|^\alpha$, is the l_α distance (raised to the αth power) between vectors u_1 and u_2. Many applications such as clustering, classification, nearest neighbor searching etc. only require pairwise distances of the data; and hence one might discard the original massive data after stable random projections, as long as one can estimate the distances from the stable samples.

Statistical Estimations

The method of stable random projections boils down to a statistical estimation problem. That is, given k i.i.d. samples $x_j \sim S(\alpha, d_{(\alpha)})$, $j = 1, 2,\ldots,k$, estimate the scale parameter $d_{(\alpha)}$. Listed below are various estimators, together with their estimation variances either exactly or asymptotically.

- *The arithmetic mean estimator, for* α = 2 *only*

$$\hat{d}_{(2),am} = \frac{1}{k}\sum_{j=1}^{k} |x_j|^2, \tag{7}$$

$$\mathrm{Var}\left(\hat{d}_{(2),am}\right) = \frac{2}{k} d_{(2)}^2. \tag{8}$$

- *The harmonic mean estimator, for small* α *only*

$$\hat{d}_{(\alpha),hm} = \frac{-\frac{2}{\pi}\Gamma(-\alpha)\sin\left(\frac{\pi}{2}\alpha\right)}{\sum_{j=1}^{k}|x_j|^{-\alpha}} \\ \left(k - \left(\frac{-\pi\Gamma(-2\alpha)\sin(\pi\alpha)}{\left[\Gamma(-\alpha)\sin\left(\frac{\pi}{2}\alpha\right)\right]^2} - 1\right)\right), \tag{9}$$

$$\mathrm{Var}\left(\hat{d}_{(\alpha),hm}\right) = d_{(\alpha)}^2 \frac{1}{k}\left(\frac{-\pi\Gamma(-2\alpha)\sin(\pi\alpha)}{\left[\Gamma(-\alpha)\sin\left(\frac{\pi}{2}\alpha\right)\right]^2} - 1\right) \\ + O\left(\frac{1}{k^2}\right). \tag{10}$$

As α approaches zero, in the limit,

$$\lim_{\alpha\to 0+} -\frac{2}{\pi}\Gamma(-\alpha)\sin\left(\frac{\pi}{2}\alpha\right) = 1, \\ \lim_{\alpha\to 0+}\left(\frac{-\pi\Gamma(-2\alpha)\sin(\pi\alpha)}{\left[\Gamma(-\alpha)\sin\left(\frac{\pi}{2}\alpha\right)\right]^2} - 1\right) = 1. \tag{11}$$

- *The geometric mean estimator*

$$\hat{d}_{(\alpha),gm} = \frac{\prod_{j=1}^{k}|x_j|^{\alpha/k}}{\left[\frac{2}{\pi}\Gamma\left(\frac{\alpha}{k}\right)\Gamma\left(1-\frac{1}{k}\right)\sin\left(\frac{\pi}{2}\frac{\alpha}{k}\right)\right]^k}. \tag{12}$$

$$\mathrm{Var}\left(\hat{d}_{(\alpha),gm}\right) = d_{(\alpha)}^2\left\{\frac{\left[\frac{2}{\pi}\Gamma\left(\frac{2\alpha}{k}\right)\Gamma\left(1-\frac{2}{k}\right)\sin\left(\pi\frac{\alpha}{k}\right)\right]^k}{\left[\frac{2}{\pi}\Gamma\left(\frac{\alpha}{k}\right)\Gamma\left(1-\frac{1}{k}\right)\sin\left(\frac{\pi}{2}\frac{\alpha}{k}\right)\right]^{2k}} - 1\right\} \tag{13}$$

$$= d_{(\alpha)}^2 \frac{1}{k}\frac{\pi^2}{12}\left(\alpha^2 + 2\right) + O\left(\frac{1}{k^2}\right). \tag{14}$$

- *The sample median estimator*

$$\hat{d}_{(\alpha),me} = \frac{\mathrm{median}\{|x_j|^\alpha, j = 1, 2,\ldots,k\}}{\mathrm{median}\{S(\alpha, 1)\}^\alpha}. \tag{15}$$

The estimation variance of the sample median estimator $\hat{d}_{(\alpha),me}$ cannot be expressed in closed-forms.

Compared with the geometric mean estimator, the sample median estimator is not as accurate when the sample size k is not very large. The sample median estimator, however, is more convenient to compute.

Sample Complexity

When $\alpha = 2$, the celebrated Johnson-Lindenstrauss (JL) Lemma [9] showed that k, the required number of projections, should satisfy $k = O\,(log\; n/\varepsilon^2)$ so that any pairwise l_2 distance among n data points can be approximated within a $1 \pm \varepsilon$ factor of the truth.

For general $0 < \alpha \leq 2$, it is proved[11] using the geometric mean estimator that the sample complexity should also be $k = O\,(log\; n/\varepsilon^2)$. The constants can be explicitly specified.

Sampling from Stable Distributions

Sampling from a stable distribution is in general quite expensive, unless $\alpha = 2$ or $\alpha = 1$. One procedure is described in [13, Proposition 1.71.1]. A random variable W_1 is sampled from a uniform distribution on the interval $(-\frac{\pi}{2}, \frac{\pi}{2})$; and a random variable E_1 is sampled from an exponential distribution with mean 1. W_1 and E_1 are independent. Then

$$\frac{\sin(\alpha W_1)}{\cos\,(W_1)^{1/\alpha}} \left(\frac{\cos((1-\alpha)W_1)}{E_1} \right)^{(1-\alpha)/\alpha} \tag{16}$$

is distributed as $S(\alpha, 1)$.

Under certain reasonable regularity assumptions on the original data, it is possible to simplify the sampling procedure by replacing the α-stable distribution $S(\alpha, 1)$ with a mixture of a symmetric α-Pareto distribution (with probability $0 < \beta \leq 1$) and a point mass at the origin (with probability $1 - \beta$), i.e.,

$$\begin{cases} P_\alpha & \text{with prob. } \frac{\beta}{2} \\ 0 & \text{with prob. } 1-\beta\,, \\ -P_\alpha & \text{with prob. } \frac{\beta}{2} \end{cases} \tag{17}$$

where P_α denotes an α-Pareto variable, i.e., $\mathbf{Pr}\,(P_\alpha > t) = \frac{1}{t^\alpha}$ if $t \geq 1$, and 0 otherwise. An α-Pareto distribution has the same tail behaviors as $S(\alpha, 1)$, but it is much easier to sample from. For example, given a random variable U drawn from a uniform distribution on the unit interval (0,1), then $1/U^{1/\alpha}$ follows an α-Pareto distribution.

If D random variables, r_1, r_2,...,r_D, are sampled i.i.d. from (17), then a linear combination of r_i's is asymptotically stable, as $D \to \infty$. That is [10],

$$c_1 r_1 + c_2 r_2 + \ldots + c_D r_D \Rightarrow \\ S\left(\alpha, \beta\Gamma(1-\alpha)\cos\left(\frac{\pi}{2}\alpha\right) \sum_{j=1}^{D} |c_i|^\alpha \right), \tag{18}$$

provided that the data, c_1, c_2,...,c_D, satisfy

$$\frac{\max_{1 \leq i \leq D} |c_i|}{\left(\sum_{i=1}^{D} |c_i|^\alpha\right)^{1/\alpha}} \to 0, \text{ as } D \to \infty. \tag{19}$$

The parameter β in (17) controls the sparsity of the projection matrix. Small β values considerably reduce the processing cost for conducting random projections. β should be chosen according to the data dimension D and the prior knowledge about the data. Some synthetic and real-world data experiments in [10] indicated that, when $\alpha = 1$, using (17) with $\beta < 0.\ 1$, achieved very similar estimation accuracy as using the exact stable distribution, even when D is not too large.

Key Applications

Stable distributions have been widely used for modeling real-world *heavy-tailed* data, arising in finance, economics, Internet traffic, computational Linguistics, and many other fields.

There have been numerous applications of stable random projections, in theoretical computer science, databases, data mining, data streams, and signal recovery[5].

Stable Random Projections for Dimension Reductions

Data mining and machine learning algorithms often assume a "data matrix" $\mathbf{A} \in \mathbb{R}^{n \times D}$, with n rows and D columns. For many algorithms, the data matrix $\mathbf{A}$ is utilized only through pairwise distances of $\mathbf{A}$ instead of the original data. A projection matrix $\mathbf{R} \in \mathbb{R}^{D \times k}$ is generated by sampling each entry from i.i.d. $S(\alpha, 1)$. The projected data matrix $\mathbf{B} = \mathbf{A} \times \mathbf{R} \in \mathbb{R}^{n \times k}$ contain enough information to approximately recover pairwise l_α distances of $\mathbf{A}$. The number of projections (sample size), should satisfy $k = O\,(log\; n/\varepsilon^2)$.

- The original data matrix $\mathbf{A}$ may be too large for physical memory, for example, $\mathbf{A}$ could be the term-by-document matrix at Web scale. Even if $\mathbf{A}$ may fit in memory, storing all pairwise distances of $\mathbf{A}$ in memory can be infeasible when $n > 10^6$. In contrast, the projected data matrix $\mathbf{B}$ may be small enough for the memory. Because **B** has only k columns, pairwise distances may be computed on demand.

- Computing all pairwise distances of **A** costs $O(n^2D)$. The cost is reduced to $O(nDk + n^2k)$ using stable random projections.
- When $\alpha = 2$, the projected data matrix **B** preserve not only the pairwise (squared) l_2 distances of **A** in expectations, but also the pairwise inner products of **A** in expectations. Some applications care about inner products more than distances. In databases, for example, counting the joint sizes can be viewed as computing inner products.

Stable Random Projections for Data Stream Computations

Consider the *Turnstile* model [12], which is a linear model for data streams. The input data stream $s_t = (i, I_t)$ arriving sequentially describes the underlying signal S, meaning $S_t[i] = S_{t-1}[i] + I_t$, $i = 1$ to D, where t denotes time. For example, S may represent the arriving IP addresses ($D = 2^{64}$) and $S_t[i]$ records the frequencies of IP address i. The term $\sum_{i=1}^{D} |S_t[i]|^{\alpha}$ is often referred to as the αth *frequency moment* of S_t. Due to the linearity of the *Turnstile* model, stable random projections can be applied for approximating the frequency moments.

Again, a random projection matrix $\mathbf{R} \in \mathbb{R}^{D\times k}$ is generated by sampling each entry r_{ij} from i.i.d. $S(\alpha, 1)$. A vector x of length k is initialized so that $x_j = 0$, for $j = 1$ to k. Then for each arriving tuple $s_t = (i, I_t)$, update $x_j \leftarrow x_j + r_{ij} \times I_t$ for $j = 1$ to k.

At any time t, the entries x_j, $j = 1$ to k, are i.i.d. samples from $S\left(\alpha, \sum_{i=1}^{D} |S_t[i]|^{\alpha}\right)$; and hence one can estimate the αth frequency moment $\sum_{i=1}^{D} |S_t[i]|^{\alpha}$. Due to the linearity, the same methodology can also be applied for approximating the difference between two streams.

In particular, when $\alpha \to 0+$, $\sum_{i=1}^{D} |S_t[i]|^{\alpha}$ approaches the Hamming norm of S_t, which is the total number of nonzero entries, sometimes referred to as the number of *distinct items*. Thus, stable random projections can provide the tool for approximating the Hamming norm (and Hamming distance) in dynamic streaming data, using very small α.

Cross-references

▶ Stream Mining

Recommended Reading

1. Achlioptas D. Database-friendly random projections: Johnson-Lindenstrauss with binary coins.. J. Comput. Syst. Sci., 66(4): 671–687, 2003.
2. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. In Proc. 28th Annual ACM Symp. on Theory of Computing, 1996, pp. 20–29.
3. Cormode G., Datar M., Indyk P., and Muthukrishnan S. Comparing data streams using hamming norms (how to zero in). In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 335–345.
4. Datar M., Immorlica N., Indyk P., and Mirrokn V.S. Locality-sensitive hashing scheme based on *p*-stable distributions. In Proc. 20th Annual Symp. on Computational Geometry, 2004, pp. 253–262.
5. Donoho D.L. Compressed sensing. IEEE Trans. Inform. Theory, 52(4):1289–1306, 2006.
6. Fama E.F. and Roll R. Parameter estimates for symmetric stable distributions.. J. Am. Stat. Assoc., 66(334):331–338, 1971.
7. Indyk P. Stable distributions, pseudorandom generators, embeddings, and data stream computation.. J. ACM, 53(3):307–323, 2006.
8. Indyk P. and Motwani R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In Proc. 30th Annual ACM Symp. on Theory of Computing, 1998, pp. 604–613.
9. Johnson W.B. and Lindenstrauss J. Extensions of Lipschitz mapping into Hilbert space. Contemp. Math., 26:189–206, 1984.
10. Li P. Very sparse stable random projections for dimension reduction in l_α $(0 < \alpha \leq 2)$ norm. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007.
11. Li P. Estimators and tail bounds for dimension reduction in l_α $(0 < \alpha \leq 2)$ using stable random projections. In Proc. 19th Annual ACM -SIAM Symp. on Discrete Algorithms, 2008.
12. Muthukrishnan S. Data streams: Algorithms and applications. Found. Trends Theor. Comput. Sci., 11:117–236, 2005.
13. Samorodnitsky G. and Taqqu M.S. Stable Non-Gaussian Random Processes. Chapman & Hall, 1994.
14. Vempala S. The Random Projection Method. American Mathematical Society, Providence, RI, 2004.
15. Zolotarev V.M. One-dimensional Stable Distributions. American Mathematical Society, Providence, RI, 1986.

Stack-based Query Language

Kazimierz Subieta
Polish-Japanese Institute of Information Technology,
Warsaw, Poland

Synonyms

SBQL

Definition

Stack-Based Query Language [1] is a query and programming language devoted to object-oriented database models. SBQL is the result of investigations into a uniform theoretical and conceptual basis for object-oriented query languages integrated with programming capabilities and abstractions, including database abstractions: updatable views, stored procedures and transactions. SBQL is developed according to the Stack-Based Architecture (SBA) [3,2] that is a conceptual frame for developing object-oriented query and programming languages. SBQL has the same role and meaning as object algebras, but it is much more universal and formally precise. SBQL deal with object store models that include complex objects, associations, classes, types, methods, inheritance, dynamic roles, encapsulation, polymorphism, semi-structured data, and other features. The functionality of SBQL includes all well-known query operators (selection, projection, navigation, path expressions, join, quantifiers, etc.), some less known operators (transitive closures, fixed-point equations, etc.), imperative (updating) statements integrated with queries, modules, procedures, functions and methods (with parameters being queries and recursive). SBQL deals with static strong type checking and with query optimization methods based on indices, rewriting rules and other techniques. Abstract implementation (a kind of operational semantics) is the basic paradigm of formal specification of SBQL semantics. It involves an abstract machine that employs abstract definitions of three internal data structures well-known in specification of programming languages: an *object store*, an *environmental stack*, and a *query result stack* (thus the Stack-Based Architecture). SBQL has been implemented within several research prototypes; the last and the most complete one is ODRA (Object Database for Rapid Applications).

Key Points

SBQL adopts a classical run-time mechanism of Programming Languages (PLs), with some improvements. The main syntactic decision is the unification of PL expressions and queries; queries remain the only kind of PL expressions. In SBQL there is no conceptual difference between expressions such as 2 + 2 and $(x + y) \times z$, and queries such as *Employee* where *Salary* = 1000 or (*Employee* where *Salary* = $(x + y)$ $*z$). *Name*. All such expressions/queries can be used as arguments of imperative statements, as parameters of procedures, functions or methods, as a return from a procedure, etc.

Semantically, SBQL is based on the classical *naming-scoping-binding* paradigm. Scopes are organized in an *environmental stack* with the "search from the top" rule. The operational semantics of query operators, programming constructs and procedures (functions, methods, views, etc.) is defined in terms of the three mentioned abstract data structures: *object store*, *environmental stack*, and *query results stack*. SBQL subdivides operators into algebraic and non-algebraic. Algebraic operators act only on the query result stack. The essence is non-algebraic operators, such as selection, projection, join, quantifiers, ordering, transitive closures, and iterations that are defined through the environment stack (with no reference to any object algebra or calculus). SBQL is extended by programming constructs, procedures, methods, modules, transactions and updatable views. SBQL has a (semi) strong static type checking and query optimizations. SBQL is implemented for different environments, including XML and workflow systems. SBQL is considered as a departure point for the new fourth generation database standard developed by OMG.

Cross-references

- ▶ Class
- ▶ Database Programming Language
- ▶ Inheritance
- ▶ OODB (Object-Oriented Database)
- ▶ Query Language
- ▶ Query Optimization
- ▶ Strong Typing
- ▶ Updatable View

Recommended Reading

1. SBQL web pages with a lot of resources and references. http://www.sbql.pl/.
2. Subieta K., Beeri C., Matthes F., Schmidt J.W. A stack-based approach to query languages. In Proc. 2nd East-West Database Workshop. 1994, pp. 159–180.
3. Subieta K., Kambayashi Y., Leszczyłowski J. Procedures in object-oriented query languages. In Proc. 21st Int. Conf. on Very Large Data Bases, 1995, pp. 182–193.

Staged Database Systems

▶ Staged DBMS

Staged DBMS

Stavros Harizopoulos
HP Labs, Palo Alto, CA, USA

Synonyms

Staged database systems

Definition

A Staged Database Management System (DBMS) is a database software architecture that optimizes data and instruction locality at all levels of the memory hierarchy in a computer system. An additional goal of Staged DBMS is to provide a robust and efficient platform for both parallelizing and pipelining database requests. The main principle of the Staged Database System design is to organize and assign software system components into self-contained stages; database request execution is broken into stages and sub-requests are group-processed at each stage. This allows for a context-aware execution sequence of requests that promotes reusability of both instructions and data, and also facilitates development of work sharing mechanisms, which has been a key application for StagedDB; work sharing is defined as any operation that reduces the total amount of work in a system by eliminating redundant computation or data accesses. Existing database systems can be converted to staged ones by carrying over their algorithms and mechanisms, and adapt those in a platform that supports staged execution.

Historical Background

Though the Staged DBMS architecture was proposed by Harizopoulos and Ailamaki in 2003 [3], one of the earliest prototype relational database systems, INGRES [12], also consisted of four “stages” (processes) that enabled pipelining; the reason for breaking up the DBMS software was main memory size limitations. Work in staged architectures re-emerged in the early 2000s, first by Larus and Parkes as a generic programming paradigm for building server applications [7], and subsequently by Welsh et al as a means for deploying highly concurrent internet services [13]. Initial prototypes of database systems developed at Carnegie Mellon University that followed the principles of StagedDB (Qpipe and Cordoba [4,5]), focused on the performance benefits of work sharing. A relational engine based on the Staged DBMS design can proactively coordinate same-operator execution among concurrent queries, thereby exploiting common accesses to memory and disks as well as common intermediate result computation.

Foundations

Modern commercial DBMS are typically built as a large piece of software that serves multiple requests using a thread-based concurrency model. Queries are handled by one or more threads (or processes) that follow the query execution plan up to its completion. This model implicitly defines a query execution sequence and a resource utilization schedule in the system. Whenever a thread blocks due to an I/O, an ungranted lock request, an internal synchronization condition, or due to an expiring CPU time quantum, the thread scheduler assigns the CPU to the next runnable thread of the highest priority. This context-switching mechanism creates a logical gap in the sequence of actions the DBMS performs. While a software developer can optimize the individual steps involved in a single query’s execution, she or he typically has no means of applying similar optimization techniques to a collection of multiplexed queries.

A staged database system consists of a number of self-contained software modules, each encapsulated into a *stage*. A stage is an independent server with its own queue, thread support, and resource management that communicates and interacts with the other stages through a well-defined interface. Stages accept *packets*, each carrying a query’s state and private data, perform work on the packets, and may enqueue the same or newly created packets to other stages. Each stage is centered around exclusively owned (to the degree possible) server code and data. There are two levels of CPU scheduling: local thread scheduling within a stage and global scheduling across stages. The StagedDB design promotes stage autonomy, data and instruction locality, and minimizes the usage of global variables.

A stage provides two basic operations, enqueue and dequeue, and a queue for the incoming packets. The stage-specific server code is contained within dequeue. The system works through the exchange of packets between stages. A packet represents work that the server must perform for a specific query at a given stage. It first enters the stage’s queue through the enqueue operation and waits until a dequeue operation removes it. Then, once the query's current state is restored, the stage specific code is executed. Depending on the stage and the query, new packets may be created and enqueued at other stages. Eventually, the stage code returns by either (i) destroying the packet (if done with that query at the specific stage), (ii) forwarding the packet to the next stage (i.e., from parse to optimize), or by (iii) enqueueing the packet back into the stage's queue (if there is more work but the client needs to wait on some condition). Queries use packets to carry their state and private data. Each stage is responsible for assigning memory resources to a query. In a shared-memory system, packets carry only pointers to the query's state and data structures (which are kept in a single copy). Each stage employs a pool of worker threads (the stage threads) that continuously call dequeue on the stage's queue, and one thread reserved for scheduling purposes (the scheduling thread). An analysis of scheduling tradeoffs in staged database systems along with a description of an initial implementation can be found in [2].

Key Applications

A key application for the Staged DBMS design has been detecting and exploiting work sharing opportunities at run-time inside a relational database engine. Traditional relational DBMS typically execute concurrent queries independently by invoking a set of operator instances for each query. To exploit common data retrievals and computation in concurrent queries, relational engines employ techniques ranging from constructing materialized views to optimizing multiple queries and sharing concurrent scans to the same table. These three techniques are briefly described next.

Materialized view selection [8] is typically applied to workloads known in advance, in order to speed up queries that contain common sub-expressions. Materialized views exploit commonality between different queries at the expense of potentially significant view maintenance costs. Tools for automatic selection of materialized views take such costs into account when recommending a set of views to create. The usefulness of materialized views is limited when the workload is not always known ahead of time or the workload requirements change frequently.

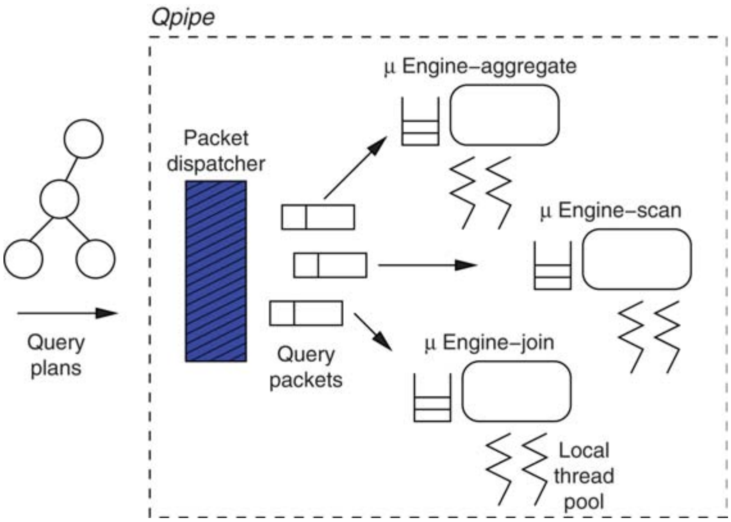
Multiple-query optimization (MQO) [10] identifies common sub-expressions in query execution plans during optimization, and produces globally-optimal plans. The detection of common sub-expressions is performed at optimization time, thus, all queries need to be optimized as a batch. In addition, to share intermediate results among queries, MQO typically relies on costly materializations. To avoid unnecessary materializations, a study described in [9] introduces a model that decides at the optimization phase which result can be pipelined and which needs to be materialized to ensure continuous progress in the system.

Shared scans allow multiple independent concurrent scans to the same table on disk to be synchronized, so that each new page fetched from disk is consumed by all scans that include the page in their range. This optimization applies to scans that can receive their input pages in any arbitrary order. Since queries interact with the buffer pool manager through a page-level interface, it requires a certain engineering effort to develop generic policies to coordinate current and future accesses from different queries to the same disk pages. Several commercial DBMSs (Teradata, Microsoft's SQL Server, IBM's DB2 [6]) and research prototypes [4,14] incorporate various forms of multi-scan optimizations.

A relational engine based on the Staged DBMS design complements the above-mentioned techniques, by proactively coordinating same-operator execution among concurrent queries, thereby exploiting common accesses to memory and disks as well as common intermediate result computation. Such staged relational engines are the academic prototypes QPipe and Cordoba [4,5], developed at Carnegie Mellon University.

To maximize data and work sharing at execution time, QPipe monitors each relational operator for every active query in order to detect overlaps. For example, one query may have already sorted a file that another query is about to start sorting; by monitoring the sort operator QPipe can detect this overlap and reuse the sorted file. Once an overlapping computation is detected, the system executes the corresponding operation only once, and simultaneously pipelines the results of the common operation to the interested parties, thereby avoiding materialization costs.

QPipe follows a "one-operator, many-queries" design philosophy. Each relational operator is promoted to a staged, independent micro-engine which manages a set of threads and serves queries from a queue (see Fig. 1). A packet dispatcher converts an incoming query plan to a series of query packets. Data flow between micro-engines occurs through dedicated



Staged DBMS. Figure 1. QPipe architecture: queries with the same operator queue up at the same micro-engine (for simplicity, only three micro-engines are shown).

buffers - similar to a parallel database engine. QPipe optimizes resource utilization by grouping requests of the same nature together, and by having dedicated micro-engines to process each group of similar requests. Every time a new packet queues up in a micro-engine, all existing packets are checked for overlapping work. On a match, each micro-engine can employ different mechanisms for data and work sharing, depending on the enclosed relational operator. Such mechanisms are described in detail in [4]. Subsequent work on QPipe produced the Cordoba prototype which is suited for execution on multicore CPUs. The tradeoffs of work sharing in highly parallel muticore chip designs are discussed in [5].

Future Directions

By the year 2005 it was apparent to microprocessor designers that performance increases in next generation CPUs would come by incorporating an increasing number of CPU cores on the same chip. If this trend continues to hold, then software designers will need to devise efficient solutions to take advantage of the (increasing) hardware-available parallelism. A preliminary study of bottlenecks for database systems on multicore CPUs (chip multiprocessors or CMPs) that was published in 2007 can be found in [1]. A future direction for Staged database systems is to exploit their inherent parallelism nature and apply it to CMP designs.

From a software engineering point of view, years of DBMS software development have lead to complex implementations that are increasingly difficult to extend, tune, and evolve. While software developers commonly organize code into separate components, the final product consists of tightly integrated and interdependent software modules, "glued" together to eliminate overheads and increase performance. It has been argued that such monolithic systems are "one size fits all" designs [11] that cannot possibly excel in all areas of data management. Another potential future direction for StagedDB is to allow several software components with specialized functionality to be transparently integrated and used inside a single system, thereby achieving high performance in several areas of data management without needing a number of different specialized architectures.

Cross-references

- ► Operator-level Parallelism
- ► Parallel Database

Recommended Reading

1. Hardavellas N., Pandis I., Johnson R., Mancheril N., Ailamaki A., and Falsafi B. Database servers on chip multiprocessors: limitations and opportunities. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007.
2. Harizopoulos S. Staged Database Systems. PhD Thesis, Computer Science Department, Carnegie Mellon University, 2005.
3. Harizopoulos S. and Ailamaki A. A case for staged database systems. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.

4. Harizopoulos S., Shkapenyuk V., and Ailamaki A. QPipe: a simultaneously pipelined relational query engine. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 383–394.
5. Johnson R., Hardavellas N., Pandis I., Mancheril N., Harizopoulos S., Sabirli K., Ailamaki A., and Falsafi B. To Share Or Not To Share? In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 351–362.
6. Lang C., Bhattacharjee B., Malkemus T., Padmanabhan S., and Wong K. Increasing buffer-locality for multiple relational table scans through grouping and throttling. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 1136–1145.
7. Larus J.R. and Parkes M. Using cohort-scheduling to enhance server performance. In Proc. General Track of the USENIX Annual Technical Conf., 2002, pp. 103–114.
8. Roussopoulos N. View indexing in relational databases. ACM Trans. Database Syst., 7(2):258–290, 1982.
9. Roy P., Seshadri S., Sudarshan S., and Bhobe S. Efficient and Extensible Algorithms for Multi Query Optimization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 249–260.
10. Sellis T.K. Multiple query optimization. ACM Trans. Database Syst., 13(1):23–52, 1988.
11. Stonebraker M., Bear C., Cetintemel U., Cherniack M., Ge T., Hachem N., Harizopoulos S., Lifter J., Rogers J., and Zdonik S. One size fits all? - Part 2: Benchmarking results. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 173–184.
12. Stonebraker M., Held G., Wong E., and Kreps P. The design and implementation of Ingres. ACM Trans. Database Syst., 1(3): 189–222, 1976.
13. Welsh M., Culler D., and Brewer E. Seda: an architecture for well-conditioned, scalable internet services. In Proc. 18th ACM Symp. on Operating System Principles, 2001, pp. 230–243.
14. Zukowski M., Héman S., Nes N., and Boncz P.A. Cooperative scans: dynamic bandwidth sharing in a DBMS. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 723–734.

Standard Effectiveness Measures

Ethan Zhang[1,2], Yi Zhang[1]
[1]University of California-Santa Cruz, Santa Cruz, CA, USA
[2]Yahoo! Inc., Santa Cruz, CA, USA

Synonyms

Evaluation in information retrieval

Definition

The standard effectiveness evaluation in information retrieval centers around determining how relevant are the documents retrieved to users' information needs. A user query represents the user's information need. A user information need is hidden, depending on what the user already knows, what the user wants to find out about, and what is the users constraint (time, format, price, recent, location). Recall and precision are the basis of relevance-based effectiveness measures. Other commonly used evaluation measures based on recall and precision include the F-measure, the 11-point precision-recall curve, and the average precision.

Historical Background

The first formal evaluation of information retrieval (IR) systems was conducted in the late 1950s by Cyrid Cleverdon at the College of Aeronautics in Cranfield, England [2]. These studies are often referred to as the Cranfield experiments and are the foundation of future research on IR evaluation. The Cranfield model requires an experimental setting with a test document collection, a set of queries, and relevance judgements that tell the relationships between the documents and the queries. Recall and precision are the effectiveness measures used in the Cranfield experiments, where the former is the percentage of the relevant documents that are retrieved and the latter is the percentage of the retrieved documents that are relevant.

TREC (Text REtrieval Conference) [3] is the most notable large IR evaluation project. The conference, co-sponsored by the National Institute of Standards and Technology (NIST) and the Advanced Research and Development Activity (ARDA) center of the U.S. Department of Defense, started in 1992 and convenes once a year. At the conference TREC participants from different research institutions evaluate and compare their IR systems on large test collections that consist of many gigabytes of documents. TREC usually uses the Cranfield model for evaluation setup and variations of recall and precision as effectiveness measures. The test collections and effectiveness measures used at TREC have been widely adopted by IR researchers as the standards for the evaluation of information retrieval systems.

Foundations

Experimental Setup

To evaluate an information retrieval (IR) system in the standard way, an experimental setting needs to be created that involves three things: i) a test document collection, ii) a set of information needs that are represented as queries, and iii) relevance judgements for each query-document pair. The relevance judgement is made by one or more human assessors who determine whether a document is *relevant* or *irrelevant* to a query. A document is considered relevant to a query if the document satisfies the user information need represented by the query. After each retrieval experiment for a given query, four document sets are formed and counted: retrieved and relevant documents (*tp*), retrieved and irrelevant documents (*fp*), relevant and nonretrieved documents (*fn*), and irrelevant and nonretrieved documents (*tn*). It is clearer to illustrate the four document sets in the confusion matrix in Table 1.

A basic assumption of the Cranfield model is that relevance judgement is available for each query-document pair in the collection. However, it is practically impossible to obtain exhaustive relevance judgements in an experiment with millions of documents, which is very common in information retrieval evaluations. In such cases, assessors usually only provide relevance judgements for the top-*n* results returned by several different search systems for each query. In some evaluation experiments, the retrieval systems perform a new search using new queries based on the judged documents and newly retrieved documents in the top-*n* results are then judged. This process is iterated until no new relevant documents are found in the top-*n* results.

Recall and Precision

Recall and precision are the basis of relevance-based retrieval effectiveness measures. They have been the most commonly used IR evaluation measures since the Cranfield experiments, and are considered the "gold standard" of IR evaluation by many researchers. Recall is the fraction of all relevant documents that have been retrieved by an IR system. Precision is the fraction of all retrieved documents that are relevant. In terms of the numbers in the confusion matrix, recall is

$$R = \frac{tp}{tp + fn}$$

and precision is

$$P = \frac{tp}{tp + fp}$$

Recall and precision trade off against each other. One can easily build a system with a recall value of one by returning all documents in the collection, but the precision in this case would be very low. In contrast, a search engine can only return a few documents that some user has judged to be relevant, which would possibly result in high precision but have low recall.

The fact that two numbers are used to evaluate an IR system allows researchers to emphasize on one measure versus the other in various circumstances. For web search engines, a typical user would like the first few search results to be relevant (high precision), however he/she usually does not have enough time to read every document that is relevant. On the other hand, a research scientist surveying a research topic would like to see every relevant document (high recall) while tolerating low precision to a certain degree.

Standard Effectiveness Measures. Table 1. Confusion matrix, where *tp*, *fp*, *fn* and *tn* correspond to the number of documents that fall into the corresponding category

	Relevant	Irrelevant
Retrieved	true positive (*tp*)	false positive (*fp*)
Not retrieved	false negative (*fn*)	true negative (*tn*)

F-Measure

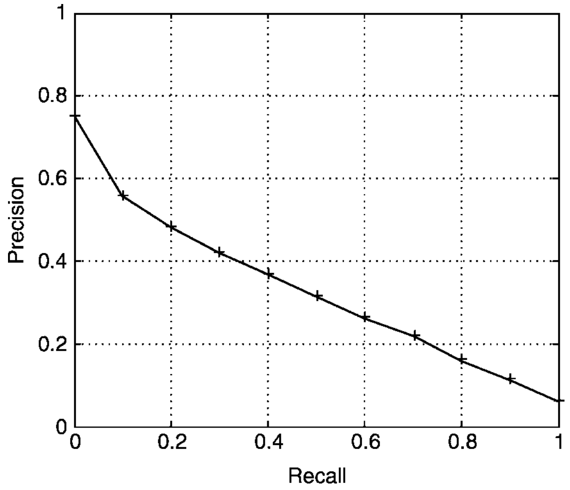
The *F-measure* is a single number that combines recall and precision. The measure is formally defined as the weighted harmonic mean of recall and precision, or

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}}$$

where $\alpha \in [0,1]$ is used to adjust the weighting of *R* and *P*. *F* is equivalent to recall when $\alpha = 0$ and precision when $\alpha = 1$. The commonly used F-measure has $\alpha = 1/2$ and weighs recall and precision equally. It can be written as $F_1 = \frac{2PR}{P+R}$. This measure can be viewed as a compromise between recall and precision. It is high only when both recall and precision are high.

Precision-Recall Curve

Recall and precision treat retrieved documents as a set, in which the order of the documents does not matter. However, most modern search systems return a ranked list of documents, where a document is ranked higher if the system believes it is more likely to be relevant to the query. For such systems, the *precision-recall curve* is a commonly used tool to evaluate the retrieval performance. The precision-recall curve plots precision as a function of recall. Standardly the precision is



Standard Effectiveness Measures. Figure 1. A 11-point precision-recall curve.

interpolated and plotted at 11 recall levels, r = 0.0,0.1,..,1.0. For multiple queries, the precisions at the same recall level can be averaged over all queries. Figure 1 shows the 11-point precision-recall curve of one of the best systems in TREC 8.

Average Precision

Average Precision (AP), which has become a standard effectiveness measure among the TREC community in recent years, is a combination of recall and precision for ranked retrieval results. Given a ranked list of documents for a single query, the average precision is the mean of the precision scores after each relevant document is retrieved.

$$Average\ Precision = \frac{\sum_r P@r}{R}$$

where r is the rank of each relevant document, R is the total number of relevant documents, and $P@r$ is the precision of the top-r retrieved documents. This measure is very sensitive to the rankings of the relevant document in the retrieval results, and therefore a good measure for tuning ranking algorithms. For one query, the average precision is approximately the area under the uninterpolated precision-recall curve.

ROC Curve

An alternative to the precision-recall curve is the *Receiver Operating Characteristics* (ROC) curve, which plots true positive rate (*sensitivity*) against false positive rate (1-*specificity*). In this setting, *sensitivity* is a synonym for recall, and false positive rate is $fp/(fp + tn)$ (therefore specificity is $tn/(fp + tn)$).

Key Applications

Retrieval effectiveness evaluation is an integral part of designing an IR system or algorithm. The standard effectiveness measures are used to compare a new system with benchmark systems and to justify the value of the new system. Although the relevance-based evaluation model has drawn criticism since the beginning, it is still the most widely accepted and used approach for IR evaluation. It is fair to say that the standard measures and their variations are used wherever information retrieval algorithms are built.

Data Sets

TREC has by far the largest data collections for information retrieval evaluation. The most often used collections are those that were built for the Ad Hoc retrieval track in the first 8 TREC conferences. In total these collections consist of 1.89 million documents. For each conference, TREC also collected relevance judgements for 50–100 information needs, which are called topics in TREC. There are relevance judgements for totally 450 information needs. Other TREC collections used for information retrieval evaluations are: TREC Web track collections, TREC terabyte track collections, TREC Blog Track collections, TREC Enterprise Track collections, TREC Filtering Track collections, TREC Genomics Track collections, TREC HARD Track collections, TREC Interactive Track collections, TREC Legal Track collections, TREC Novelty Track collections, TREC Robust Track collections, TREC Query Track collections, TREC Question Answering Track collections, and TREC SPAM Track collections. For details about the collections and different standard evaluation measures used for each collection, readers may refer to book [3] or visit TREC's web site at http://trec.nist.gov.

Cross-references

- ▶ Average Precision
- ▶ Effectiveness Involving Multiple Queries
- ▶ Eleven Point Precision-recall Curve
- ▶ F-Measure
- ▶ Information Retrieval
- ▶ Precision
- ▶ Precision at n
- ▶ Precision-Oriented Effectiveness Measures
- ▶ Recall

Recommended Reading

1. Baeza-Yates R. and Ribeiro-Neto B. Modern Information Retrieval. Addison Wesley, New York, NY, USA, 1999.
2. Cleverdon C.W. The significance of the Cranfield tests on index languages. In Proc. 14th Annual Int. Conf. on Research and Development in Information Retrieval, 1991, pp. 3–12.
3. Voorhees E.M. and Harman D.K. (eds.). TREC: Experiment and Evaluation in Information Retrieval. MIT, Cambridge, MA, USA, 2005.

Standing Query

▶ Continuous Query

Star Index

Theodore Johnson
AT&T Labs Research, Florham Park, NJ, USA

Synonyms

Star index; Join index; Join indices

Definition

A star index is a collection of join indices, one for every foreign key join in a star or snowflake schema.

Key Points

A common structure for a data warehouse is a fact table consisting of several *dimension* fields and several *measure* fields. To reduce storage costs, the fact table is often normalized into a *star* or a *snowflake* schema. Since most queries reference both the (normalized) fact tables and the dimension tables, creating a star index can be an effective way to accelerate data warehouse queries. The Red Brick data warehouse system has implemented star indices.

Cross-references

- ▶ Join Index
- ▶ Star Schema
- ▶ Snowflake Schema

Star Join Schema

▶ Star Schema

Star Schema

Konstantinos Morfonios, Yannis Ioannidis
University of Athens, Athens, Greece

Synonyms

Star join schema

Definition

A *star schema* has one "central" table whose primary key is compound, i.e., consisting of multiple attributes. Each one of these attributes is a foreign key to one of the remaining tables. Such a foreign key dependency exists for each one of these tables, while there are no other foreign keys anywhere in the schema. (In the above, without loss of generality, the assumption is made that all these other tables have simple primary keys. This is usually the case in almost all practical situations, as for efficiency, these keys are typically generated *surrogate keys*.)

Key Points

Most data warehouses that represent the multidimensional conceptual data model in a relational fashion [1,2] store their primary data as well as the data cubes derived from it in star schemas. The "central" table and the remaining tables of the definition above correspond, respectively, to the *fact table* and the *dimension tables* that are typically found in data warehouses. Each *fact* (tuple) in the fact table consists of a set of numeric *measures*, comprising the objects of analysis, and a set of *dimensions*, which uniquely determine the set of measures. The dimension tables are usually smaller than the fact table and store the attributes of the aforementioned dimensions.

For example, consider a data warehouse of a retail chain with many stores around a country. The dimensions may be the products sold, the stores themselves with their locations, and the dates, while the numeric measures may be the number of items and the total monetary amount corresponding to a particular product sold in a particular store on a particular date. The relevant star schema is shown below, where `SalesSummary` is the fact table, primary keys are in Italics, and each attribute of the fact-table primary key is a foreign key to one of the other tables.

`SalesSummary`(*ProductId*, *StoreId*, *DateId*, NumOfItems, TotalAmount)

Product(*ProductId*, ProdName, ProdDescr, Category, CategoryDescr, UnitPrice)

Store(*StoreId*, Street, City, State)

Date(*DateId*, Day, Month, Year)

If one were to draw the above as a graph, with tables as nodes and foreign keys as edges, or even as an ER diagram, with the fact table as a relationship and the dimension tables as entities, the resulting image is that of a star, with the fact table in the middle, hence, the name of these schemas.

Finally, note that dimensions often consist of several attributes organized in hierarchies. For instance, dimension Store in the example above contains values at different levels of detail, forming the hierarchy Street→City →State. As also shown in the example, star schemas capture all levels of a hierarchical dimension in a single, non-normalized table. An extension of the star schema that explicitly captures hierarchies in the dimensions is the *snowflake schema*.

Cross-references

▶ Cube Implementations
▶ Data Warehouse
▶ Dimension
▶ Hierarchy
▶ Measure
▶ Multidimensional Modeling
▶ Snowflake Schema

Recommended Reading

1. Chaudhuri S. and Dayal U. An overview of data warehousing and OLAP technology. ACM SIGMOD Rec., 26(1):65–74, 1997.
2. Kimball R. and Ross M. The data warehouse toolkit: The complete guide to dimensional modeling. Wiley, New York, NY, USA, 2nd edn., 2002.

Star Schema Modeling

▶ Multidimensional Modeling

State Query

▶ Timeslice Operator

State-based Publish/Subscribe

Hans-Arno Jacobsen
University of Toronto, Toronto, ON, Canada

Definition

State-based publish/subscribe is an instance of the publish/subscribe concept. However, it is distinguished from other publish/subscribe approaches by maintaining partial matching state when processing publications, whereas, traditionally, publish/subscribe treat publications as transient and does not manage matching state. State-based publish/subscribe support the detection of composite events, event correlation and complex event processing.

Key Points

In terms of publishing, subscribing, and decoupling, state-based publish/subscribe is no different from topic-based or content-based publish/subscribe. The main difference to the other publish/subscribe approaches is that state-based publish/subscribe treats publications as non-transient. A publication is processed by the publish/subscribe system and builds up partial matching state, contributes to existing partial matching state, triggers notifications if a match is complete, or is discarded, if no matching subscription exists. This is unlike in the other publish/subscribe approaches that treat publications as transient messages, where the arriving publication is matched against the subscriptions stored with the system and forwarded towards all matching subscribers, or dropped, if no matching subscription exists. In state-based publish/subscribe, the publish/subscribe system carries state across the processing of different publications. That is, various different publications arriving over time are correlated based on conditions expressed in subscriptions to result in matching subscriptions.

The publication data model is exactly the same as in a topic-based or a content-based publish/subscribe model, depending on the nature of the state-based approach, either topic-based, or content-based.

The subscription language model follows suite, but greatly extends the subscription language capabilities with means to express the correlation of publications. These capabilities are added to allow the application to express so called *composite subscriptions*. A composite subscription is the combination of several individual *atomic subscriptions* by means of an operator algebra that allows the developer to compose individual atomic subscriptions. An atomic subscription is a subscription in the publish/subscribe sense. It is referred to as atomic because it is matched by a single publication. A composite subscription defines a *composite event*. A composite event defines the set of events that have to occur in the specified constellation in order for the composite subscription to match. In publish/subscribe, the notion of event and publication are synonymous, while only the term composite event is used. Also, the term composite event is often used to refer to the composite subscription expression without differentiating between event and subscription.

There are large differences in the expressive power of subscription languages for specifying composite subscriptions. Common operators include the specification of composite-and (all specified events must occur in any order), composite-or (one of the events must occur), sequence operator (the specified sequence of events must occur and other events may or may not be interspersed with the sequence), and regular expression pattern (the specified pattern of events must occur). In addition, reference to time is included in many subscription languages to delimit the time a composite subscription can remain in a partial matching state before resetting to a completely unmatched state. More generally, various consumption policies attached to subscriptions express what should happen with partial matching state, as it accumulates in the system and new events arrive. For example, a consumption policy could express that a newly arriving event is correlated with the oldest or the newest event in a composite event that has accumulated a partial matching state holding many individual events already. Consumption policies are a powerful way to customize the matching behavior of state-based publish/subscribe systems.

In the state-based publish/subscribe model, the publish/subscribe matching problem is defined as follows: Given a set of subscriptions, *S*, and a sequence of events, *E*, as seen by the publish/subscribe system, determine the subscriptions in *S* that match under *E*. This formulation of the matching problem is different from the standard publish/subscribe matching problem that only looks at a single event e at a time. In state-based publish/subscribe, the matching algorithm has to manage partial matching state and correlated newly arriving events with already existing partial matching state stored in the algorithm's data structures.

State-based publish/subscribe is a fairly new subclassification of publish/subscribe, consequently few established standards and products exist that refer to this model. However, several research projects are experimenting with the above described schemes. For example, the PADRES [?,?] project is an example of a state-based publish/subscribe system. Moreover, many rule-based engine products, such as JESS and Drools are similar in conception to the above described functionality. There is a large product space of older and emerging rule-based systems that are applied to similar applications as state-based publish/subscribe. There is no clear dominant player at this point. The key difference between these approaches and the here described state-based publish/subscribe model, is the notion of a rule, which essentially is a composite subscription; except that rules follow a stricter if-then-else syntax model than composite subscriptions, which often only capture the antecedent part of a rule. Moreover, these is an emerging space of complex event processing, event correlation, or simply correlation technology, which also falls under the here defined space of state-based publish/subscribe systems. State-based publish/subscribe targets applications that need correlate events over time. Applications of this nature are event correlation for network management, system management and diagnostics, and business process execution and business activity monitoring. In these application scenarios, large numbers of events that in isolation are not useful, need to be correlated to detect higher-level composite events.

In the literature, the term state-based publish/subscribe is not used uniformly. Also, state-based publish/subscribe is just emerging as a separate model. The functionality provided by a state-based publish/subscribe system is very close to what rule-based systems offer. However, unlike rule-based systems, state-based publish/subscribe does not explicitly use rules to let developers model applications. Besides capturing state as the correlation of events, the management of publication state and subscription state are another important property of publish/subscribe systems, which falls into the subject spaces publish/subscribe model.

Cross-references

▶ Publish/Subscribe
▶ Subject Spaces

Recommended Reading

1. Fidler E., Jacobsen H.-A., Li G., and Mankovski S. The PADRES distributed publish/subscribe system. In Feature Extractions in Telecom. and Softw. Syst., S. Reiff-Marganie. and M. Ryan (eds.), IOS Press, 2005.
2. Li G., and Jacobsen H.-A. Composite subscriptions in content-based publish/subscribe systems. In Proc. ACM/IFIP/USENIX 6th Int. Middeware Conf., 2005.

Statistical Correctness

► Summarizability

Statistical Data Management

Amarnath Gupta
University of California-San Diego, La Jolla, CA, USA

Synonyms

Statistical database

Definition

A Statistical data management system is a data management system designed to explicitly handle so called "macro data," i.e., data computed by different forms of summarization, including grouping and classification, as first class objects. In a statistical data management system, there are data manipulation operators that "slice and dice" the macro data. In many statistical databases, one of the goals is to hide the micro data (i.e., the raw data records from which the macro data are computed) from user queries.

Example: A classical example of a statistical database is a database created for social or economic surveys. STORM [4] is a classical statistical data management system.

Key Points

Research on statistical databases started in the 1970s and flourished in the 1980s, predating OLAP. A number of systems developed based on both relational and object-oriented data models. Many of these systems developed a graph-based representation of statistical data where one could construct hierarchies of categories using different attributes, and place the summarized data under the suitable categories. One goal of query evaluation was to minimize redundancy of computation when answering a user query. A second important consideration in designing statistical DBMS is security so that the individual data records are not exposed to a malicious user. Several strategies are used to attain this security including (i) allowing the user to ask only aggregate queries, (ii) answering a query with a range of values instead of exact values, (iii) disallowing a user to make repeated increasingly specific queries that might expose actual data values.

Cross-references

► Privacy
► On-line Analytical Processing
► Secure Database Development
► Summarizability

Recommended Reading

1. Denning D.E. and Schlörer J. A fast procedure for finding a tracker in a statistical database. ACM Trans. Database Syst., 5 (1):88–102, 1980.
2. Ghosh S.P. Statistical relational tables for statistical database management. IEEE Trans. Softw. Eng., 12(12):1106–1116, 1986.
3. Rafanelli M. and Ricci F.L. Mefisto: a functional model for statistical entities. IEEE Trans. Knowl. Data Eng., 5(4):670–681, 1993.
4. Rafanelli M. and Shoshani A. STORM: a statistical object representation model. In Proc. 2nd Int. Conf. on Scientific and Statistical Database Management, 1990, pp. 14–29.
5. Shoshani A. OLAP and statistical databases: similarities and differences. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997, pp. 185–196.

Statistical Database

► Statistical Data Management

Statistical Decision Techniques

► Classification

Statistical Disclosure Control (SDC)

► Inference Control in Statistical Databases

Statistical Disclosure Limitation (SDL)

▶ Inference Control in Statistical Databases

Statistical Disclosure Limitation For Data Access

Stephen E. Fienberg, Jiashun Jin
Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

Confidentiality protection; Multiplicity; Privacy protection; Restricted data; Risk-utility tradeoff

Definition

Statistical Disclosure Limitation refers to the broad array of methods used to protect confidentiality of statistical data, i.e., fulfilling an obligation to data providers or respondents not to transmit their information to an unauthorized party. *Data Access* refers to complementary obligations of statistical agencies and others to provide information for statistical purposes without violating promises of confidentiality.

Historical Background

Starting in the early twentieth century, U.S. government statistical agencies worked to develop approaches for the protection of the confidentiality of data gathered on individuals and organizations. As such agencies also have a public obligation to use the data for the public good, they have developed both a culture of confidentiality protection and a set of statistical techniques to assure that data are released in a form that limits the identification of individual data providers [2]. In a now classic 1977 paper, Dalenius [3] described the probabilistic notion of a disclosure: "If the release of the statistics S makes it possible to determine the value [of confidential statistical data] more accurately than is possible without access to S, a disclosure has taken place." The ensuing statistical literature on disclosure limitation has built on this probabilistic notion.

Foundations

Privacy, Confidentiality, and Individual Identification

Massive databases and widespread data collection and processing offer enormous opportunities for statistical analyses, advances in the understanding of social and health problems, and benefits to society more broadly. But the explosion of computerized databases containing financial and healthcare records, and the vulnerability of databases accessible via the Internet, has heightened public attention and generated fears regarding the privacy of personal data. Identify theft and sensitive data disclosure may be just a click away from a new generation of computer users and potential intruders.

Data collected directly under government auspices or at public expense are in essence a public good; legitimate analysts wish to utilize the information available in such databases for statistical purposes. Thus, society's challenge is how to release the maximal amount of information without undue risk of disclosure of individually identifiable information. Assessing this tradeoff is inherently a statistical matter, as is the development of methods to limit disclosure risk. What distinguishes the field of statistical disclosure limitation from many other approaches to privacy protection is the ultimate goal of data access and enhanced data utility.

The term *privacy* is used both in ordinary language and in legal contexts with a multiplicity of meanings. Among these is the concept of privacy as "the right to be let alone," e.g., see Warren and Brandeis [14], and privacy in the context of data as the control over information about oneself. But privacy is personal and subjective, varies from one person to another, and varies with time and occasion depending on the context. It is even more difficult to define precisely the meaning of "privacy-preserving" with respect to databases, and the data pertaining to individual entities contained therein.

Confidentiality is the agreement, explicit or implicit, between data subject and data collector regarding the extent to which access by others to personal information is allowed. Confidentiality protection has meaning only when the data collector can deliver on its promise to the data provider or respondent. Confidentiality can be accorded to both individuals and organizations; for the individual, it is rooted in the right to privacy (i.e., the right of individuals to control the dissemination of information about themselves), whereas for establishments and organizations, there are more limited rights to protection, e.g., in connection with commercial secrets.

Disclosure relates to inappropriate attribution of information to a data provider or intruder, whether to an individual or organization. There are basically two types of disclosure, identity and attribute. An identity disclosure occurs if the data provider is identifiable from the data release. An attribute disclosure occurs when the released data make it possible to infer the characteristics of an individual data provider more accurately than would have otherwise been possible. The usual way to achieve attribute disclosure is through identity disclosure; first one identifies an individual through some combination of variables and then one associates with that individual values of other variables included in the released data.

Statistical disclosure limitation (SDL) is a set of techniques designed to "limit" the extent to which databases can be used to glean identifiable information about individuals or organizations. The dual goals of SDL are to assure that, based on released data, respondents can be identified only with relatively low probability, but also to release data that are suitable for non-identifiable analytical statistical purposes.

The Intruder

To protect the confidentiality of statistical data, one needs to understand what intruders or data snoopers want and how they may learn information about individuals in a database that require protection. Intruders may be those with legitimate access to databases and/or those who gain access to a database by breaking security measures designed to keep them out. In either case, one needs to distinguish among

- *Intruders with a specific target,* e.g., a friend or relative. The intruder may already know that the respondent is included in the database, will possess information about the target (e.g., height, weight, habits, income) and will search the database in order to learn additional information, e.g., drug and alcohol use.
- *Intruders in possession of data on multiple individuals whose goal is record linkage*, e.g., to build a larger database containing more individual information. Data consolidators or aggregators fit within this category.
- *Intruders without any specific target, whose goal is to embarrass the data owner.* The intruder may be an enemy agent or a "hacker" eager to demonstrate a capability of breaking through efforts to limit disclosure.

Data owners can be successful in protecting the confidentiality of released data if the intruder remains sufficiently uncertain about a protected target value after data release. Various authors in the SDL literature discuss confidentiality protection from the perspective of protecting against intruders or data snoopers, e.g., see [8,12], especially those using record linkage methods [11] for attempting to identify individuals in databases.

One may consider the intruder as someone engaged in a form of a large number of statistical tests, each at significance level α associated with an effort to identify an individual in the database. The data owner needs to account for this somehow. Some of the null hypotheses will eventually be rejected whether or not they are actually false, and thus there is a problem for the intruder as well. For the data owner, simply controlling the probability of erroneously identifying each respondent, at say 1%, is not enough. To ensure that the probability that at most one out of 1,000 individuals in a sample will be identified to be less than 1%, one in fact needs to assure that the probability of identifying *each* individual is no greater than $\approx 1\%/1000 = 10^{-5}$.

Statistical Analysis Methods for Protecting Privacy

Matrix Masking refers to a class of SDL methods used to protect confidentiality of statistical data, transforming an $n \times p$ (cases by variables) data matrix Z through pre- and post-multiplication and the possible addition of noise. The four most common forms of masking are:

1. *Sampling* clearly provides a measure of direct protection from disclosure provided that there is no information of which individuals or units are included in the sample. An intruder wishing to identify an individual in the sample and link that person's information to data in external files, using "key" variables such as age and geography available in both databases, needs to determine whether a record is unique in the sample, and if so, the extent to which a record that is unique in the sample is also unique in the population. For continuous variables, virtually all individuals are unique in the sample, and one needs to understand the probability that an intruder would correctly match records, e.g., in the presence of error in the key variables (e.g., see Fienberg et al. [8]). For categorical data, uniqueness corresponds to counts of "1" and various authors have shown, roughly speaking, that the probability that

an individual record that is unique in the sample is also unique in the population from which the sample was drawn equals the sampling fraction, n/N, e.g., see [7]. Thus for a sample of size 2,000 drawn from a population of 200,000,000 adults the sampling fraction is 2,000/200,000,000 or 0.00001. The bottom line therefore is that sampling protects, just not absolutely.

2. *Perturbation* is an approach to data masking in which the transformation involves random perturbations of the original data, either through the addition of noise or via some form of restricted randomization. The simplest form of perturbation is the addition of noise. Common forms for the noise are observations drawn from a normal distribution with zero mean or perhaps a double exponential, also centered at zero. Someone analyzing the resulting transformed data must statistically reverse the noise addition process using methods from the literature on measurement error models – this requires release of the parameters of the noise component, e.g., the error variance in the normal case. Other examples of perturbation include data swapping and related tabular adjustment approaches, e.g., see [9].
3. *Collapsing* is also referred to using the labels microaggregation and global recoding in the statistical literature [15], and *k*-anonymity in the computer science literature. In the statistical literature on tabular categorical data, collapsing across variables in a table produces a marginal table and a popular form of data release to protect confidentiality is the release of multiple marginal tables, especially when they correspond to the minimal sufficient statistics of a log-linear model. For more details, see [10].
4. *Synthetic data* are used to replace a database by a similar one, for which the individuals are generated through some statistical process. This can be achieved through the repeated application of data swapping, e.g., see [9], or the method known as multiple imputation, e.g., [13].

Implicit in all of these techniques is the notion that when masked data are released they can be used by responsible analysts to carry out statistical analyses so that they can reach conclusions similar to those that they would have reached had they analyzed the original data. This means that all of the details of the transformation, both stochastic and non-stochastic, must be made available to the user, a point not well understood in the computer science literature or by many statistical agencies. See the related discussion in [6]. Even when one has applied a mask to a data set, the possibilities of both identity and attribute disclosure remain, although the risks may be substantially diminished. Thus, one must still assess the extent of risk posed by the transformed data.

Putting SDL Methods to Use: Risk-Utility Tradeoff

If one is adding noise to a set of observations in order to protect confidentiality, how much noise is sufficient? And can one add too much? Clearly, too much noise will distort the data substantially and even if the details of the error variance are released the masking may impede legitimate statistical analyses of the data. The same is true for any of the methods of SDL. Thus, one faces a tradeoff between data protection and data utility, something that one can assess formally using statistical decision analysis and depict graphically, e.g., see the chapter by Duncan et al. in [5]. For a slightly less formal approach to the tradeoff for categorical data protection through the release of multiple margins, see [10].

A crucial but relatively rarely discussed aspect of the risk-utility tradeoff involves the issue of multiplicity, introduced above. For illustration, consider a data set with information on 2,000 individuals, for each of which records the diagnostic result of an HIV test, with 1 corresponding to a positive result and 0 to a negative one. To protect the confidentiality for those individuals with positive HIV test outcomes, the data owner adds noise to each record value.

Suppose an intruder wishes to identify the individuals corresponding to the proportion ε of "1"s $(0 < \varepsilon < 1)$ and the data owner attempts to protect the records by adding i.i.d. Gaussian noise $N(0, \sigma^2)$ to each data point. Clearly σ needs to be large enough to disguise some of the 1s and make them hard to distinguish from some of the "0′s. Suppose that the intruder wants to make sure that most of those identified as "1"s are indeed "1"s (otherwise the attack on the database would be unsuccessful). In statistical terms, this means that the intruder must control for the False Discover Rate (FDR), i.e., the rate of misclassified "1"s out of all those individuals labeled as being "1"s [2,1]: FDR = [#{Misclassified "1"s}]/[#{All classified "1"s}]. Consider an intruder who decides to set the FDR at 5% by picking a threshold σt and classifying any entry as a “1′ if the observed value exceeds the threshold. By elementary statistics, the number of misclassified “1′s is distributed as a binomial random variable, $B(n(1 - \varepsilon),\bar{\Phi}(t))$, and the number of correctly classified “1′s is distributed as $B(n\epsilon, \bar{\Phi}(1 - \frac{1}{\sigma}))$. Consequently, the associated $FDR \, \frac{n(1-\epsilon)\bar{\Phi}(t)}{n(1-\epsilon)\bar{\Phi}(t)+n\epsilon\bar{\Phi}(t-\frac{1}{\sigma})} \approx \left[1 + \left(\frac{\epsilon}{1-\epsilon}\right)\left(\frac{\bar{\Phi}(t-\frac{1}{\sigma})}{\bar{\Phi}(t)}\right)\right]^{-1}$, where $\bar{\Phi} = 1 - \Phi$ is the survival function of the standard normal distribution function.

Consider a high risk population where 50% of the individuals test positive for HIV, i.e., $\varepsilon = 1/2$ and suppose that the data owner chose $\sigma = 1$ as the noise variance to protect the data. To ensure that FDR $\leq$ 5%, the intruder needs to set $(\frac{\epsilon}{1-\epsilon})(\frac{\bar{\Phi}(t-\frac{1}{\sigma})}{\bar{\Phi}(t)}) = 19$, which yields $t \approx 3.132$. The threshold is high enough so that the chance for each of the individuals exhibiting a “1” to be correctly classified as “1” is $\bar{\Phi}(t - 1) = \bar{\Phi}(2.132) \approx 0.0165$, which seems not very large. But since $n = 2000$, the number of “0”s that are misclassified as “1”s is approximately $n(1 - \epsilon)\bar{\Phi}(3.132) \approx 2000 \times \frac{1}{2} \times 0.00087 = 0.87$, and the number of “1”s that are correctly classified as “1”s is approximately $n\epsilon\bar{\Phi}(2.132) = 2000 \times \frac{1}{2} \times 0.0165 \approx 16.5$. This says that the intruder is able to identify 17 records, out of which 16 are corrected classified!

Alternatively, one might ask about the probability that no more than k “1”s are correctly classified, i.e., $\sum_{j=0}^{k} \binom{n\epsilon}{j} p^j (1 - p)^{n\epsilon - j}$, $p \equiv \bar{\Phi}(2.132)$. For $k = 0, 3, 6, 9$, the probabilities are correspondingly 5.45×10^{-8}, 5.22×10^{-5}, 2.62×10^{-3}, and 0.031. To understand the implications of these values, consider $k = 9$. This says that with probability as high as 97%, 9 or more records that are actually “1”s are correctly identified as “1”! For many this might seem to be a worrisome situation, and it raises issues associated with the efficacy of adding noise that have not appeared in the statistical literature on confidentiality protection.

Suppose that the data come from a low risk population where only 5% or 100 individuals test positive for HIV, i.e., $\varepsilon = 0.05$, and the data owner uses a similar level of noise addition for confidentiality protection, i.e., $\sigma = 1$. Then to ensure that FDR $\leq$ 5%, the intruder needs to set $(\frac{\bar{\Phi}(t-\frac{1}{\sigma})}{\bar{\Phi}(t)}) = 361$, which yields $t \approx 6.22$. Correspondingly, $\bar{\Phi}(t) = 2.5 \times 10^{-10}$ and $\bar{\Phi}(t - \frac{1}{\sigma}) = 8.9 \times 10^{-8}$. The expected number of “0” that are misclassified as “1”s is approximately $n(1 - \varepsilon)\bar{\Phi}(6.22) = 2000 \times 0.95 \times 2.49 \times 10^{-10} \approx 4.7 \times 10^{-6}$, and the number of “1” that are correctly classified as “1”s is approximately $n\varepsilon\bar{\Phi}(5.22) = 2000 \times 0.05 \times 8.95 \times 10^{-8} \approx 8.95 \times 10^{-6}$. In this case, since $n\bar{\Phi}(6.22) \ll 1$, the approximation is inaccurate and one needs to take a different approach.

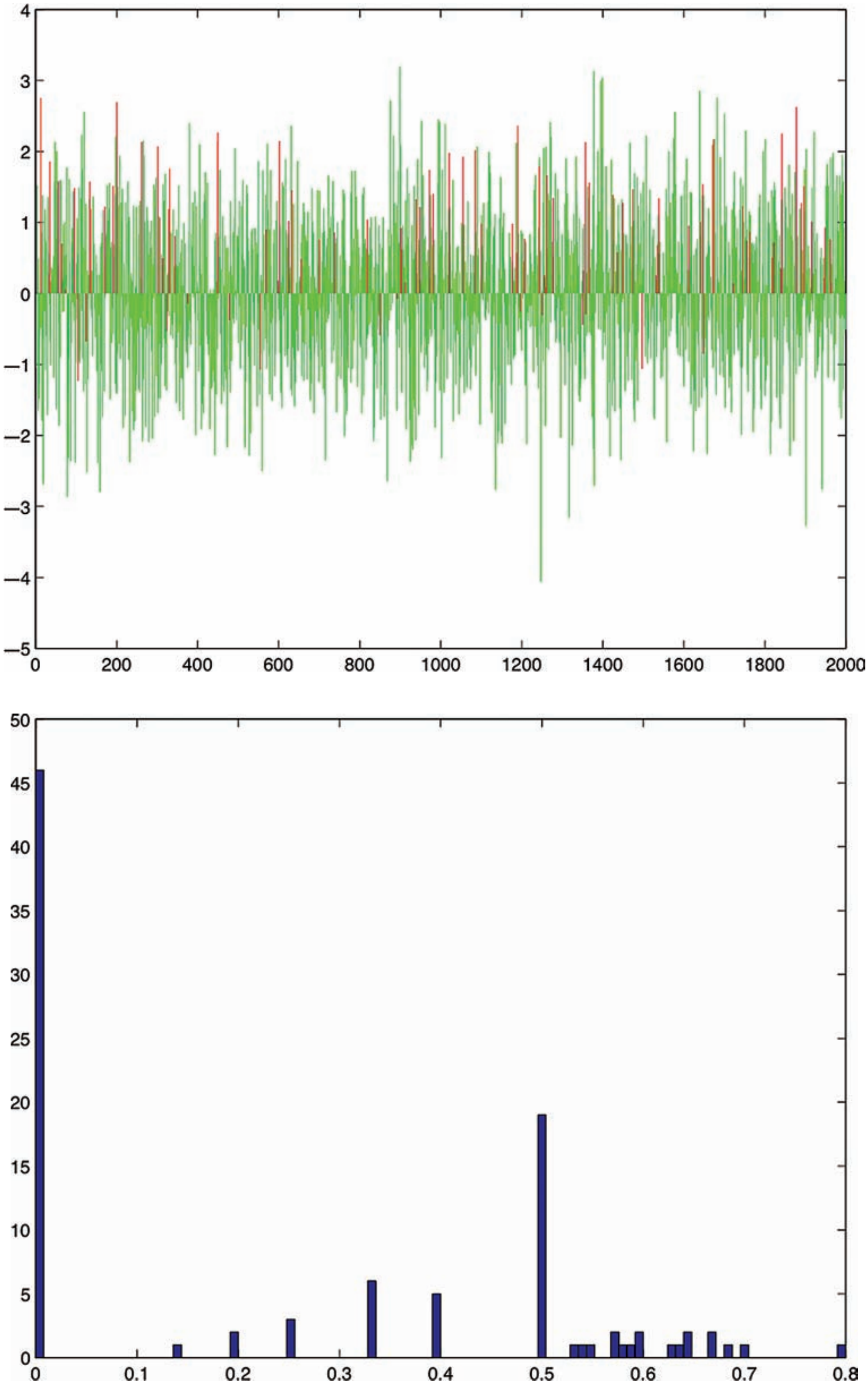
In fact, the proportion of true HIV cases is so small that the example falls into the so-called *very sparse regime* studied in detail in the multiple testing literature, see for example [1,4]. One phenomenon from that literature implies that when the noise level is relatively high, the extreme values are not necessary related to cases with positive HIV tests. Consider the following simulated data set with $n = 2{,}000$ cases, where 100 of them are HIV (equal to 1) and all others are non-HIV (equal to 0). The data owner adds independent standard Gaussian $N(0,1)$ noise to each value. Figure 1 shows the result where red correspond to cases with positive HIV tests, and green correspond to cases with negative HIV tests. The red values are larger than typical green ones, but not larger than all of them. In fact, among the largest 10 values, only 2 are red, with 8 are green.

This leads us to another interesting phenomenon from the statistical literature on the FDR. Let mFDR denote the minimum FDR across all possible thresholds t, mFDR = $min_{\{t\}}$ {FDR$_t$: FDR at the threshold t}. How small can mFDR be? Figure 1 shows the histogram of the mFDR values for 100 independent repetitions of the simulation experiment. More than half of the time, the mFDR value is no less than 15%, and sometimes it is as great as 50% and larger!

This simple example implies that with $\sigma = 1$, the noise level might be so large that the intruder cannot correctly identify any HIV cases. But from the perspective of the risk-utility tradeoff, one also needs to ask whether the noise level is so high that the data are no longer analytically useful. Thus one needs to ask: What is the largest noise variance that still allows for valid inferences, c.f., [1,4]. If the number of true HIV cases is

$$m = m_n = n^{1-\beta}, \qquad (1)$$

and the noise level is $\sigma = \sigma_n = \frac{1}{\sqrt{2r\log n}}$, where $0 < \beta$, $r < 1$ are parameters, then as n tends to ∞, there is a boundary, $r = \beta$, which separates the β-r plane into two



S

Statistical Disclosure Limitation For Data Access. Figure 1. Left Panel: Perturbed HIV data through addition of independent draws from $N(0,1)$. Those values associated with positive HIV tests are in red, and those with negative HIV tests are in green. Right Panel: 100 simulated mFDR values based on 100 simulation for $n = 2{,}000$ and $\varepsilon = .05$ and added noise from $N(0,1)$.

regions: the *classifiable* region and the *non-classifiable* region; In the interior of the classifiable region, asymptotically, it is possible to isolate completely the cases with positive HIV tests from those with negative ones. In fact, there is a threshold by which one can identify that subset of the data corresponding to positive HIV tests. On the one hand, almost every "identified" HIV case has a positive HIV test and the subset includes almost all the cases with positive HIV tests. In the interior of the non-classifiable region, by contrast, such isolation of cases is impossible. In fact, given any chosen threshold, either one situation or the other occurs!

For the example of $n = 2{,}000$ and $m = 100$ cases with positive HIV tests. take $\beta = 1 - log(m)/log(n) \approx 0.3941$ in model (1). In order not to have complete isolation of cases with HIV, one should take $\sigma_n > \frac{1}{\sqrt{2\beta \log(n)}} \approx 0.409$. For $\sigma_n = 0.5$, consider a repetition of the case of $n = 2{,}000$ and $\varepsilon = 1/2$, as well as the case of $n = 2{,}000$ and $\varepsilon = 0.05$. Thus $1/\sigma = 2$ and to control the FDR at 5%, one must evaluate $\frac{\epsilon}{1-\epsilon}(\frac{\bar{\Phi}(t-2)}{\bar{\Phi}(t)}) = 19$, which yields $t = 0.54$. Correspondingly, $n(1-\epsilon)\bar{\Phi}(t) = 2,000 \times \frac{1}{2} \times 0.029 \approx 29$, and $n\epsilon\bar{\Phi}(t-2) = 2,000 \times \frac{1}{2} \times 0.54 \approx 540$. Furthermore, for $k = 0{,}3{,}6.9$, the probability that no more than k "1"s are correctly classified are extremely small ($< 10^{-315}$). For the case of $n = 2{,}000$ and $\varepsilon = 0.05$. Similarly, one must similarly evaluate $\frac{\epsilon}{1-\epsilon}(\frac{\bar{\Phi}(t-2)}{\bar{\Phi}(t)}) = 19$, which yields $t = 3.62$. Correspondingly, $n(1 - \varepsilon)\bar{\Phi}(t) = 2{,}000 \times 0.95 \times 1.47 \times 10^{-4} \approx 0.28$, and $n\varepsilon\bar{\Phi}(t-2) = 2{,}000 \times 0.05 \times 0.053 \approx 5$. Furthermore, for $k = 0{,}3{,}6.9$, the probability that no more than k "1"s are correctly classified are 4.5×10^{-3}, 0.22, 0.73, 0.96. Take $k = 3$. The probability that more than three "1"s are correctly classified is about 78%.

This example illustrates how the multiplicity issue arises when an intruder tries to match many records with those in a database "protected" by matrix masking. Simply protecting each individual record with high probability is not enough; there remains a substantial chance for one or more record to be vulnerable to disclosure.

Summary

The accumulation of massive data sets and the rapid development of the Internet expanded opportunities for data analyses as well as created enormous challenges for privacy protection. This brief overview of the literature on statistical disclosure limitation has stressed four categories of approaches: sampling, perturbation, collapsing (aggregation), and the use of synthetic data. An overlooked issue in privacy protection is the notion of "multiplicity," which is present whenever one attempts to protect many records simultaneously, or a data intruder tries to match the records of multiple targets in a database simultaneously. Simply protecting each individual record with high probability does not automatically protect all records, and careful statistical measures for resolving the multiplicity issue are necessary.

Key Applications

The methods of statistical disclosure limitation outlined here are already in widespread use by government statistical agencies throughout the world. The volumes by Doyle et al. [5] and Willenborg and de Waal [15] summarize a number of the approaches and methodologies. In particular, census data released by most developed countries are protected using these methods.

Future Directions

The elaboration of approaches described here to deal with very large scale databases remains a challenge, especially in the face of demands for increased access to data and novel attacks on databases by intruders. This entry presents the first known application of ideas and results from the multiplicity literature to the problem of statistical disclosure limitation and risk-utility tradeoff. These ideas require further development and integration with the rest of the literature. And methodology dealing with the risk-utility tradeoff will clearly need to evolve in response to the evolution of intruder strategies to compromise databases.

Cross-references

- ▶ Data Perturbation
- ▶ Individually Identifiable Data
- ▶ Inference Control in Statistical Databases
- ▶ Matrix Masking
- ▶ Privacy
- ▶ Privacy-Preserving Data Mining
- ▶ Randomization Methods to Ensure Data Privacy

Recommended Reading

1. Abramovich F., Benjamini Y., Donoho D., and Johnstone I. Adapting to unknown sparsity by controlling the false discovery rate. Ann. Stat., 34:584–653, 2006.
2. Anderson M. and William Seltzer W. Challenges to the confidentiality of U.S. federal statistics, 1910–1965. J. Off. Stat., 23:1–34, 2007.

3. Benjamini Y. and Hochberg Y. Controlling the false discovery rate: A practical and powerful approach to multiple testing. J. Roy. Statist. Soc. B, 57:289–300, 1995.
4. Dalenius T. Towards a methodology for statistical disclosure control. Statistisk Tidskrift, 5:429–444, 1977.
5. Donoho D. and Jin J. Higher Criticism for detecting sparse heterogeneous mixtures. Ann. Stat., 32:962–994, 2004.
6. Doyle P., Lane J.I., Theeuwes J.J.M., and Zayatz L. (eds.). Confidentiality, Disclosure and Data Access: Theory and Practical Application for Statistical Agencies. Elsevier, New York, NY, USA, 2001.
7. Fienberg S.E. Confidentiality, privacy and disclosure limitation. In Encyclopedia of Social Measurement, Vol. 1. Academic Press, San Diego, CA, USA, 2005, pp. 463–469.
8. Fienberg S.E. and Makov U.E. Confidentiality, uniqueness and disclosure limitation for categorical data. J. Off. Stat., 14:485–502, 1998.
9. Fienberg S.E., Makov U.E., and Sanil A.P. A Bayesian approach to data disclosure: Optimal intruder behavior for continuous data. J. Off. Stat., 13:75–89, 1997.
10. Fienberg S.E., Makov U.E., and Steele R.J. Disclosure limitation using perturbation and related methods for categorical data (with discussion). J. Off. Stat., 14:485–502, 1998.
11. Fienberg S.E. and Slavkovic A.B. Preserving the confidentiality of categorical statistical databases when releasing information for association rules. Data Min. Knowl. Discov., 11:155–180, 2005.
12. Hertzog T.N., Scheuren F.J., and Winkler W.E. Data Quality and Record Linkage Techniques. Springer-Verlag, New York, NY, USA, 2007.
13. Lambert D. Measures of disclosure risk and harm. J. Off. Stat., 9:313–331, 1993.
14. Raghunathan T.E., Reiter J., and Rubin D.B. Multiple imputation for statistical disclosure limitation. J. Off. Stat., 19:1–16, 2003
15. Warren S. and Brandeis L. The right to privacy. Harvard Law Rev., 4:193–220, 1890.
16. Willenborg L. and de Waal T. Elements of Statistical Disclosure Control. Vol. 155. Lecture Notes in Statistics Springer-Verlag, New-York, NY, USA, 2001.

Steganography

Radu Sion
Stony Brook University, Stony Brook, NY, USA

Synonyms

Information hiding; Covert communication

Definition

Steganography (from the greek "steganos" – covered) is a term denoting mechanisms for hiding information within a "cover" such that, generally, only an intended recipient will (i) have knowledge of its existence, and (ii) will be able to recover it from within its cover. In modern digital steganography applications, the cover is often a multimedia object such as an image that is minorly altered in the steganographic process. Steganographic techniques have been deployed for millenia and several primitive war-time instances are described in the Histories of Herodotus of Halicarnassus, including a case of a message tattooed on the shaven head of a slave, which, when covered with grown hair acted as an effective "cover" when traversing enemy lines.

Key Points

Steganography versus Watermarking

A common trend of term misuse is associated with steganography. Specifically, many sources consider the term "watermarking" as equivalent. This is incorrect. There are fundamental differences, from both application perspectives and associated challenges. Steganography usually aims at enabling Alice and Bob to exchange messages in a manner as stealthy as possible, through a hostile medium where Malory could lurk. On the other hand, Digital Watermarking is deployed by a rights holder (Alice) as a court proof of rights over a Work, usually in the case when an adversary (Mallory) would benefit from using or selling that very same Work or maliciously modified versions of it. In Digital Watermarking, the actual value to be protected lies in the Works themselves, whereas pure steganography usually makes use of them as simple value "transporters." In Watermarking, Rights Assessment is achieved by demonstrating (with the aid of a "secret" known only to Alice – "watermarking key") that a particular Work exhibits a rare property ("hidden message" or "watermark"). For purposes of convincing the court, this property needs to be so rare that if one considers any other random Work "similar enough" to the one in question, this property is "very improbable" to apply (i.e., bound false-positives rate). It also has to be relevant, in that it somehow ties to Alice (e.g., by featuring the bit string "(c) by Alice"). There is a threshold determining the ability to convince the court, related to the "very improbable" assessment. This defines a main difference from steganography: from the court's perspective, specifics of the property (e.g., watermark message) are not important as long as they link to Alice (e.g., by saying "(c) by Alice") and, she can prove "convincingly" it is she who induced it to the (non-watermarked) original. In watermarking, the emphasis is on "detection" rather

S

than "extraction." Extraction of a watermark, or bits of it, is usually a part of the detection process but just complements the process up to the extent of increasing the ability to convince in court.

Fingerprinting

In this application of steganography, license violators are "tracked" by hiding uniquely identifying "fingerprints." If the Work would then be found in the public domain, the fingerprints can then be used to assess the source of the leak.

Recommended Reading

1. Watermarking World. Online at http://www.watermarkingworld.org/

Stemming

Chris D. Paice
Lancaster University, Lancaster, UK

Synonyms

Suffix stripping; Suffixing; Affix removal; Word conflation

Definition

Stemming is a process by which word endings or other affixes are removed or modified in order that word forms which differ in non-relevant ways may be merged and treated as equivalent. A computer program which performs such a transformation is referred to as a *stemmer* or *stemming algorithm*. The output of a stemming algorithm is known as a *stem*.

Historical Background

The need for stemming first arose in the field of information retrieval (IR), where queries containing search terms need to be matched against document surrogates containing index terms. With the development of computer-based systems for IR, the problem immediately arose that a small difference in form between a search term and an index term could result in a failure to retrieve some relevant documents. Thus, if a query used the term "explosion" and a document was indexed by the term "explosives," there would be no match on this term (whether or not the document would actually be retrieved would depend on the logic and remaining terms of the query).

The first stemmer for the English language to be fully described in the literature was developed in the late 1960s by Julie Beth Lovins [11]. This has now been largely superseded by the Porter stemmer [14], which is probably the most widely used, and the Paice/Husk stemmer [12]. Stemmers have also been developed for a wide variety of other languages.

Foundations

Definitions

In an IR context, the process of taking two distinct words, phrases or other expressions and treating them as semantically equivalent is referred to as *conflation*. The two expressions need not be precisely synonymous, but they must refer to the same core concept (compare "computed" and "computable"). In this article, the term "practically equivalent" is used to mean that, *for the purposes of a particular application*, the words may as well be taken as equivalent.

The term conflation is sometimes used as though it is equivalent to stemming, but it is in fact a much broader concept, since it includes (i) cases where the strings concerned are multi-word expressions, as in "access time" and "times for access", and (ii) cases where the strings are not etymologically related, as in "index term" and "descriptor". In case (i) special string matching techniques may be used, whereas in case (ii) reference to a dictionary or thesaurus is necessary. The present account deals exclusively with the conflation of etymologically related single words.

There are various possible approaches to word conflation, including the following.

1. *Direct matching*. In this method, the character sequences of two words are compared directly, and a similarity value is computed. The words are then considered to match if their mutual similarity exceeds a predefined threshold. To give a simple example, the first six letters of the words "exceeds" and "exceeded" are the same, so these words together contain 12 matching letters out of 15. Hence, a similarity of 12/15 = 0.80 can be computed. Use of a threshold (say, 0.70) allows a decision as to whether the words can be considered equivalent.

   With such a method, setting the threshold is problematic. Thus, the similarity between "exceeds" and "excess" is 0.62, which is below the stated threshold. However, allowing for this by lowering the threshold to 0.60 would cause "excess" and "except" (similarity 0.67) to be wrongly conflated.

2. *Lexical conflation.* In this case a thesaurus or dictionary is used to decide whether two words are equivalent. Obviously, this method can be used even for etymologically unrelated words. A problem here is obtaining a suitably comprehensive and up-to-date thesaurus, and one which explicitly lists routine variants such as plurals.
3. *Cluster-based conflation.* This method, investigated by Xu and Croft [15], involves creating clusters of practically equivalent words by analyzing the word-word associations in a large representative text corpus. Each query word is then supplemented by adding in the other words in its cluster. In contrast to method (2), the clusters created are specific to the text collection in question. However, the creation of the clusters can be very time-consuming.
4. *N-gram conflation.* In this method, each word is decomposed into a collection of *N*-letter fragments (*N*-grams), and a similarity is computed between the *N*-gram collections of two words; a threshold is then applied to decide whether the words are equivalent. This approach was pioneered by Adamson and Boreham [1], who used sets of *bigrams*, where $N = 2$. For example, after eliminating duplicates and sorting into order, "exceeds" can be represented by the bigram set {ce, ds, ed, ee, ex, xc} and "exceeded" by {ce, de, ed, ee, ex, xc}. Out of 7 distinct bigrams here, 5 are shared between the two words; hence a similarity of $5/7 = 0.712$ can be computed.
5. *Stemming.* Stemming refers to the removal of any suffixes (and sometimes other affixes) from an input word to produce a *stem*. Two words are then deemed to be equivalent if their stems are identical. This method is much favored because it is fast: all words can be reduced to stems on input to the system, and simple string matching used thereafter. The remainder of this article focuses on stemming in this narrow sense.

Stemming Algorithms

The most primitive type of stemming is *length truncation*, in which any word containing more than *N* letters is represented by its first *N* letters. Thus, using $N = 6$, "exceeds" and "exceeded" are both reduced to "exceed," though "excess" remains distinct. Most stemmers, however, use rules which test for specific endings which, if found, are removed or replaced.

It is possible to implement a set of stemming rules by encoding them directly as a computer program. This permits an arbitrary level of complexity in the tests and transformations used, but it makes the stemmer harder to design and modify. An alternative approach is to hold the rules in one or more tables, with a *stemming engine* designed to operate on those tables. This separation means that the same stemming engine can in principle be used with different rules on different occasions, depending on the particular stemming requirements. It also means that a given stemming engine can be used, with little or no adaptation, for a range of other languages.

In all of the stemmers to be described below, a stemming operation is subject to "acceptability" constraints. This ensures that if the ending of a word matches an ending in a table, the indicated action is only taken if relevant conditions are satisfied. These constraints vary from one stemmer to another (and sometimes from one ending to another). The Paice/Husk stemmer [12] uses a very simple constraint: an action only proceeds if the resulting stem will contain at least two letters, including at least one vowel. Thus, "string" cannot be transformed to "str" through a hit with an "-ing" rule.

It is important to note that, for the purposes of most applications, the stem returned by a stemmer need not be an actual word of the language. The essential desideratum is that the stem should be the same for all practically equivalent words, and different for all other words.

Stemming algorithms can be classified roughly as *single-stage*, *multi-stage* or *iterative*. Lovins' stemmer [11] is often described as a single-stage stemmer, since it uses a single table containing all the distinct endings which are to be removed. In this table, the rules are held in decreasing order of length, and the first matching rule is the one applied. This ensures that, if the table contains the endings "-mentary," "-ment" and "-ary," the word "documentary" and "document" are both reduced to "docu." If the "-ary" ending were tested first, "documentary" would simply be stemmed to "document."

In fact, Lovins' stemmer has a second (iterative) stage, using a table of 35 "recoding rules," which can adjust the stem returned by the first stage, e.g., by replacing double final consonants by single, and making other changes – thus,

admission → admiss → admis
admittance → admitt → admit → admis

The 290 endings listed for the original Lovins' stemmer are demonstrably inadequate, but a satisfactory rule set would need to be much longer, and would show considerable structural redundancy. The Paice/Husk stemmer [12] avoids this by taking an iterative approach, where long endings are removed or transformed in a series of actions using a much shorter table containing shorter endings. When one action has been activated and completed, the table may be entered again to see if another rule will fire. Thus, some endings are removed in several stages:

$$\text{sensibilities} \rightarrow \text{sensibility} \rightarrow \text{sensibil} \\ \rightarrow \text{sensibl} \rightarrow \text{sens}$$

The most popular stemmer for English is that devised by Martin Porter at Cambridge University [14]. This stemmer, which was designed to reflect the linguistic structure of suffixes in English words, proceeds through five main stages. Stage 1 deals with plurals, verb inflexions, and words ending with "-y"; stages 2–4 with all the major derivational endings; and stage 5 with "-e" removal and singling of final "-ll." Acceptability constraints are based on a quantity called the "measure" of the word, which is derived by scanning the consonant/vowel pattern of each word.

To facilitate the development of stemmers, Porter developed a special language known as Snowball (see URL below).

Krovetz developed a stemmer KSTEM which uses a machine readable dictionary to decide whether a stemmed form corresponds to an acceptable root form of the original word [9]. Despite careful refining of the algorithm to allow for a range of problem cases, IR performance was not consistently better than with Porter's algorithm.

The design of a stemmer for a new language can be a labor-intensive business. An attractive alternative is to generate a set of stemming rules automatically. Thus, Bacchin et al. have developed a probabilistic approach which uses a large representative corpus to determine the optimal splitting of words into "stems" and "derivations." They showed that, in terms of retrieval performance, their approach was about as good as Porter's algorithm for a range of European languages [5].

Prefixes and Infixes

In English, stemmers are usually designed for removing suffixes from words. The removal of "intimate" prefixes such as "intro-," "pro-" and "con-" generally results in words being wrongly conflated (consider "intro-duction," "pro-duction" and "con-duction"). However, there may be a case for removing looser prefixes such as "hyper-" or "macro-." Also, prefix removal may be desirable in certain domains with highly artificial vocabularies, such as chemistry and medicine.

As explained below, there are some languages in which removal or replacement of prefixes, or even infixes, is in fact essential.

Performance and Evaluation

Since stemmers were originally developed to aid the operation of information retrieval systems, it was natural that they were first assessed in terms of their effect on retrieval performance, as well as on "dictionary compression" rates. Researchers were frustrated to find that the effects on retrieval performance for English-language material were small and often negative [10]. Removal of "-s" and other regular inflectional endings might be modestly helpful, but use of heavier stemming could easily result in a loss of performance [7]. Work by Krovetz and by Hull showed that most benefit is obtained in cases where the document or the query is short [8,9].

Stemmers are not used only in IR systems, but in a wide range of natural language applications. A less "IR-oriented" general approach to measuring performance is to consider the number of actual stemming errors committed by an algorithm, and this forms the basis of a method developed by Paice [13]. Notice first that stemming errors are of two kinds: *understemming*, in which a pair of practically equivalent words are not conflated, and *overstemming*, in which two semantically distinct words are wrongly conflated. It is easy to see that these two types of error trade off against one another.

Paice's method makes use of a collection of distinct words (typically derived from an actual text source) which have been manually collected into groups, such that all the members of a group are practically equivalent. Two indices are computed based on a stemmer's treatment of pairs of words, which reflect the rate of understemming and of overstemming. Morerover, the resulting values are related to a baseline represented by length truncation (see above). This results in a general measure of accuracy called the "error rate relative to truncation," ERRT. Whilst this approach provides some insights into the activities of stemmers, it is unclear how such information should be used, though in future it might provide the basis for an optimization process. The use of human-defined target groups is a weak feature.

As a by-product, Paice's method yields a "stemming weight," which is the ratio of the overstemming to the understemming indices; a large stemming weight means that the stemmer is "heavy," "strong" or "aggressive." Frakes and Fox [6] present a series of other metrics related to stemming weight, as well as metrics for comparing stemmers one with another. These are all "behavior metrics," and do not relate directly to the actual accuracy of the stemming process.

Non-English Stemmers

Stemming is appropriate for most (though not all) natural languages, and appears to be especially beneficial for highly inflected languages [9]. There is neither space nor need to describe non-English stemmers here, except to note that some languages exhibit much greater structural complexity, and this warrants special approaches. Thus, a typical Arabic word consists of a root verb of three (or occasionally four or five) consonants (e.g., "k-t-b" for "to write"), into which various prefixes, infixes and suffixes are inserted to produce specific variant forms ("katabna": "we wrote" and "kitab": "book"). Some researchers have concentrated on extracting the correct root from a word [3], but Aljlayl and Frieder have demonstrated that better retrieval performance is obtained by using a simpler "light stemming" approach, in which only the most frequent suffixes and prefixes are removed [4]. Their results showed that extraction of roots causes unacceptable levels of overstemming.

Key Applications

As noted earlier, stemmers are routinely used in information retrieval systems to control vocabulary variability. They also find use in a variety of other natural language tasks, especially when it is required to aggregate mentions of a concept within a document or set of documents. For example, stemmers may be used in constructing lexical chains within a text. Stemming can also have a role to play in the standardization of data for input to a data warehouse.

Data Sets

Useful resources can be found on the two websites noted below.

URL to Code

Stemming algorithms and other resources may be obtained from the following websites:

http://www.snowball.tartarus.org/

http://www.comp.lancs.ac.uk/computing/research/stemming/.

Cross-references

▶ Lexical Analysis of Textual Data

Recommended Reading

1. Adamson G.W. and Boreham J. The use of an association measure based on character structure to identify semantically related pairs of words and document titles. Inf. Process. Manage., 10(7/8):253–260, 1974.
2. Ahmad F., Yusoff M., and Sembok M.T. Experiments with a stemming algorithm for Malay words. J. Am. Soc. Inf. Sci. Technol., 47(12):909–918, 1996.
3. Al-Sughaiyer I.A. and Al-Kharashi I.A. Arabic morphological analysis techniques: a comprehensive survey. J. Am. Soc. Inf. Sci. Technol., 55(3):189–213, 2004.
4. Aljlayl M. and Frieder O. On arabic search: Improving the retrieval effectiveness via a light stemming approach. In Proc. Int. Conf. on Information and Knowledge Management, 2002, pp. 340–347.
5. Bacchin M., Ferro N., and Melluci M. A probabilistic model for stemmer generation. Inf. Process. Manage., 41(1):121–137, 2005.
6. Frakes W.B. and Fox C.J. Strength and similarity of affix removal stemming algorithms. SIGIR Forum, 37(1):26–30, 2003 (Spring 2003).
7. Harman D. How effective is suffixing? J. Am. Soc. Inf. Sci., 42(1):7–15, 1991.
8. Hull D. A Stemming algorithms: a case study for detailed evaluation. J. Am. Soc. Inf. Sci., 47(1):70–84, 1996.
9. Krovetz R. Viewing morphology as an inference process. Artificial Intelligence, 118(1/2):277–294, 2000.
10. Lennon M., Pierce D.S., Tarry B.D., and Willett P. An evaluation of some conflation algorithms for information retrieval. J. Inf. Sci., 3:177–183, 1981.
11. Lovins J.B. Development of a stemming algorithm. Mech. Transl. Comput. Linguist., 11:22–31, 1968.
12. Paice C.D. Another stemmer. SIGIR Forum, 24(3):56–61, 1990.
13. Paice C.D. A method for the evaluation of stemming algorithms basedonerrorcounting. J. Am. Soc. Inf. Sci., 47(8):632–649, 1996.
14. Porter M.F. An algorithm for suffix stripping. Program, 14(3):130–137, 1980.
15. Xu J. and Croft W.B. Corpus-based stemming using coocurrence of word variants. ACM Trans. Inf. Syst., 16(1):61–81, 1998.

Step

▶ Activity

Stewardship

▶ Digital Curation

Stop-&-go Operator

Nikos Hardavellas, Ippokratis Pandis
Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

Non-pipelineable operator

Definition

A Stop-&-Go operator, or non-pipelineable operator, is a relational operator which cannot produce any result tuples unless it has consumed all of its input. A typical Stop-&-Go operator is the Sort operator. The usage of Stop-&-Go operators in the query execution plan limits the degree of operator-level parallelism.

Key Points

Some relational operators need to consume their entire input before they are able to produce tuples. These operators are called Stop-&-Go or non-pipelineable operators. A typical example of a Stop-&-Go operator is the Sort operator. To sort a set of tuples, the entire input set needs to be consumed before the operator can output the tuples in sorted order. There are many Stop-&-Go operators, such as various flavors of Join and Aggregation. For example, Hash Join is a Stop-&-Go operator because the Probe phase cannot start unless the Build phase has finished. Similarly, Sort-Merge Join is a Stop-&-Go operator because the Merge phase cannot start unless the Sort phase has finished.

The Stop-&-Go operators stop the flow of tuples in a pipelined execution, so their usage limits the degree of operator-level parallelism. Thus, a query optimizer that aims to achieve high levels of operator-level parallelism may choose to replace Stop-&-Go operators with more expensive – but pipelineable – operators [1,2] in the query execution plan.

Cross-references

▶ Hash Join
▶ Operator-Level Parallelism
▶ Pipelining
▶ Query Plan
▶ Sort-Merge Join

Recommended Reading

1. Graefe G. Encapsulation of parallelism in the volcano query processing system. In Proc. of the ACM SIGMOD Conf. on Management of Data, 1990, pp. 102–111.
2. Johnson R., Hardavellas N., Pandis I., Mancheril N., Harizopoulos S., Sabirli K., Ailamaki A., and Falsafi B. To share or not to share? In 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 351–362.

Stoplists

Edie Rasmussen
University of British Columbia, Vancouver, BC, Canada

Synonyms

Negative dictionary; Stopwords

Definition

Stoplists are lists of words, commonly called stopwords, which are not indexed in an information retrieval system, and/or are not available for use as query terms. A stoplist can be created by sorting the terms in a document collection by frequency of occurrence, and designating some number of high frequency terms as stopwords, or alternately, by using one of the published lists of stopwords available. Stoplists may be generic or domain specific, and are of course language specific. When a stoplist is used for indexing, as a document is added to the system, each word in it is checked against the stoplist (for example through dictionary lookup or hashing), and those which match are eliminated from further processing. In some systems, stopwords are indexed, but the stoplist is used to eliminate the words from processing when they are used as query terms.

Key Points

Hans Peter Luhn, in pioneering work on automatic abstracting, put forward the idea that certain words are too common to provide a significant discrimination value, instead contributing noise to the calculations, and should be excluded from consideration [6]. In his description of the processing needed to create Keyword-in-Context (KWIC) indexes, he described a “dictionary of insignificant words” which was to be excluded from processing. In his view, these insignificant words would include “articles, conjunctions, prepositions, auxiliary verbs, certain adjectives, and words such as “report,” “analysis,” “theory” and the like” [7]. This idea was incorporated in the 1960s in commercial KWIC indexes introduced by Biological Abstracts (BASIC) and Chemical Abstracts (Chemical Titles). At

Biological Abstracts the number of excluded terms varied, but grew to 1,000 words, although analysis showed that 14 words were enough to prevent 80% of the entries, and the tradeoff between reduction in (printed) index size and cost of dictionary lookup became a factor as the length of the stopword list increased [2]. The use of the term “stopword” seems to come from this application, where designation of a word as a stopword stops the corresponding index entries from being printed [9]. As electronic databases became available for searching, database vendors created lists of stopwords which were not indexed or available for use in searching. The lists used by commercial systems were usually quite short; for example, in the Dialog system, the list consists of only nine words: An, And, By, For, From, Of, The, To, With [1]. Other lists which were published and used in IR research contain several hundred words; for an example see Fox [3].

There have been dual arguments put forward for the use of a stopword list, or stoplist, in building an index. The first relates to efficiencies in storage and processing. Common words follow a Bradford distribution and therefore a relatively small number of words account for a relatively large number of word occurrences. Data will vary somewhat from one corpus to another, but a typical analysis might show, for instance, that six words account for 20% of a corpus or 250–300 words for 50% [3]. Therefore, removing these words from the inverted index in a text retrieval system significantly decreases the size of an uncompressed index, though it adds to the processing time needed to create the index since a dictionary lookup or other technique is needed to identify words as stopwords when the text is processed. However, Witten et al. [10] suggest that the storage savings are most obvious in an uncompressed index, and are much less significant if an appropriate compressed representation is used. Processing queries containing stopwords can also be expensive, since their frequency of occurrence results in very long lists of postings. However efficiencies in query processing can be introduced, such as sorting postings lists by term weight, so that processing can be terminated when term weights are small, as is the case with stopwords [8]. Therefore, current techniques can address to a large extent the problems associated with processing very common words in both indexes and queries.

The second rationale for using a stoplist is the claim put forward by Luhn, that these words have very little power for semantic resolution, and therefore may contribute noise rather than meaning for retrieval purposes. However, current term weighting techniques greatly reduce the contribution of common words in ranking functions, and there are many situations where an inability to use stopwords as query terms makes it difficult if not impossible to perform an effective search. There are classic examples of searches composed entirely of stopwords, such as “AT&T,” “To be or not to be,” or where a stopword is critical to the query, for example the “A” in “Vitamin A.” In other situations the removal of stopwords makes it impossible to adequately specify the query, for example, where common words are needed to clarify the relationship between terms. One approach, as used by Google for instance, is to index stopwords but to process them in queries only when the searcher specifically requests it in the query formulation, or when the query is composed only of stopwords [4]. This allows the stopwords to be used when they would be helpful, and ignores them when they are not, but it does require some knowledge of advanced search techniques on the part of the searcher. Overall, improved storage and compression techniques, term weighting schemes, and advanced query processing techniques significantly reduce the cost of including stopwords in a text retrieval system and arguments can be made for eliminating the stopword list [8,10].

Cross-references

▶ Index Creation and File Structures
▶ Lexical Analysis of Textual Data

Recommended Reading

1. Dialog Online Courses: Glossary of search terms. Available at: http://training.dialog.com/onlinecourses/glossary/glossary_life.html
2. Flood B.J. Historical note: the start of a stop list at Biological Abstracts. J Am Soc. Inf Sci., 50(12):1066, 1999.
3. Fox C. 1Lexical analysis and stoplists. In Information Retrieval: Data Structures and Algorithms, W.B. Frakes, R. Baeza-Yates, (eds.). Prentice-Hall, Englewood Cliffs, NJ, 1992, pp. 102–130.
4. Google Web Search Help Center. Search basics: use of common words. Available at: http://www.google.com/support/bin/answer.py?answer=981
5. Korfhage R.R. Information Storage and Retrieval. Wiley, NY, 1997.
6. Luhn H.P. The automatic creation of literature abstracts. IBM J. Res. Development, 2:157–165, 1958.
7. Luhn H.P. Keyword-in-context index for technical literature. Am. Doc., 11(4):288–295, 1960.

8. Manning C.D., Raghavan P., and Schütze H. Introduction to Information Retrieval. Cambridge University Press, Cambridge, UK, 2008.
9. Parkins P.V. Approaches to vocabulary management in permuted-title indexing of Biological Abstracts. In Proc. American Documentation Institute, 26th Annual Meeting, 1963, pp. 27–29.
10. Witten I.H., Moffat A., and Bell T.C. Managing Gigabytes: Compressing and Indexing Documents and Images (2nd ed.). Morgan Kaufmann, San Francisco, CA, 1999.

Stopwords

► Stoplists

Storage Access Models

Kaladhar Voruganti
Network Appliance, Sunnyvale, CA, USA

Synonyms

Database storage layer; Database physical layer

Definition

Database management systems provide storage that can be accessed via a query language interface and that can be updated under the control of a transaction management system. A database management system can reside on top of a file system (system management storage) or on top of raw block storage (direct managed storage), or on a combination of file system and block storage (hybrid model). There are advantages and disadvantages of using these different types of underlying storage.

Historical Background

Database management systems have historically managed data on disks by themselves. Over the past few years, the management functionality in file systems has steadily improved. In order to leverage this management functionality (like data backup and recovery), and make it easier for system administrators to manage their storage infra-structure in a uniform manner, database system vendors started to architect database systems so that they can also run on top of file systems. This, in turn, has now provided users with multiple storage alternatives. These alternatives are discussed in this section.

Foundations

A database is logically organized into multiple table spaces. A table space determines the location from where a table gets its storage space. Thus, database tables are created inside table spaces. Multiple tables can exist in a table space. A table space obtains its storage from either files or a directory of a file system or multiple raw block devices. Each raw block device or file or directory that provides storage to a table space is known as a container. Multiple containers can provide storage to a single table space. Separate table spaces exist for user data, user index data, temporary data, and log data. Similarly, separate table spaces also exist for system tables and catalog tables. A table space can be classified as one of the following:

System Managed Space (file system): This type of storage container is fully under the control of the file system. The advantages of system managed space are:

- *In-time Provisioning*: Typically, database administrators use system managed storage for managing temporary storage. This ensures that space is only allocated when needed, and re-used for other purposes when it is de-allocated. This is an advantage of system managed space over database managed space, where space is pre-reserved in the containers.
- *Leverage File System Utilities:* System managed storage files can leverage backup, migration, and all of the other file system utilities. Previously, this was a major advantage for system managed storage, but over the past few years, a lot of progress has been made in building block level data management utilities that can be leveraged by a database managed system.

Database Managed Space (Raw Storage): This type of storage container is fully under the control of the database management system. The advantages of database managed space are:

- *Better Performance:* Database managed storage offers better performance because of the following reasons:
  - Unlike in system managed space, the absence of file system logging also helps with the overall performance.
  - As file systems age, the underlying managed storage system can become fragmented. The absence of file system fragmentation also helps with the performance of database managed storage.

  - The lack of an intermediate file system buffer (in addition to the database buffer) in the I/O path, and the absence of contention in the file system buffer with other non-database applications, helps to improve the performance of database managed storage. With the recent emergence of direct I/O mechanisms, where the file system places data directly into the database buffer the disadvantages of an intermediate buffer have been reduced even in system managed storage space.
  - Each database containers resource is dedicated to that container, and thus, there is no contention for storage space with other non-database applications.
- *Storage Extensibility:* Database managed storage allows for dynamic addition of more storage containers to a table space. Thus, one does not have to a priori know the maximum size of the required storage space.
- *Concurrent Access:* Some file systems put a limit on the number of concurrent accesses on a file (container). These limitations are not present in database managed containers.

Hybrid Space (database managed file): In this type of storage container, the file system created file is given to the database management system to manage. It is a compromise between the above two types of containers. In these containers, there is no intermediate file system buffer but the size of the container is limited to the size limits of the created file. The performance of hybrid containers is almost as good as the database managed containers, and one can leverage the conventional file system provided backup/migration utilities.

Key Applications

OLTP applications that are performance sensitive typically use system managed storage. Applications store their temporary storage in system managed containers.

Cross-references

- ▶ Backup and Restore
- ▶ Buffer Management
- ▶ Database Concurrency
- ▶ Logging and Recovery

Recommended Reading

1. Mellish B., Aschoff J., Cox B., and Seymour D. IBM ESS and IBM DB2 UDB Working Together. IBM Redbook, SG24–6262–00, San Jose, CA, 2001.

Storage Area Network

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Synonyms

SAN

Definition

A Storage Area Network is a network whose main purpose is to transfer data between storage devices and servers and among storage devices. The term Storage Area Network can be a synonym of the term Storage Network, but differs in that the term Storage Area Network is usually identified with a network with block-level I/O services rather than file access services. More specifically, the term Storage Area Network is often used to refer to a network with Fiber Channel technology. However, SNIA released a more general definition in which the term Storage Area Network is not connected with any specific types of network connections. Under this definition, an Ethernet-based network infrastructure for mainly connecting storage devices could also be considered a Storage Area Network. The term Storage Area Network is often abbreviated to SAN. When the term SAN is used in connection with a specific network technology X, the use of a term "X SAN" is encouraged. A SAN based on Fiber Channel technology is sometimes referred to as Fiber Channel SAN. A SAN based on TCP/IP technology is often shortened to IP SAN. Despite the original meaning, the term SAN is sometimes identified with a storage system which is also implemented using a network.

Key Points

A SAN is a general network for connecting storage devices, but as a matter of fact, currently most SANs are implemented on top of Fiber Channel technology. A typical SAN is composed of Fiber Channel switches, storage devices such as disk arrays and tape libraries, and Fiber Channel host bus adapter (HBA) cards that are installed into servers. Alternative network technologies such as iSCSI, IFCP and FCIP are used mainly in entry-level storage systems or in wide-area network connections.

Cross-references

- ▶ Direct Attached Storage
- ▶ Network Attached Storage
- ▶ Storage Network Architectures

Recommended Reading

1. Clark T. Designing Storage Area Networks: A Practical Reference for Implementing Fibre Channel and IP SANs. Addison-Wesley, Reading, MA, 2003.
2. Storage Network Industry Association. The Dictionary of Storage Networking Terminology. Also available at: http://www.snia.org/.
3. Troppens U., Erkens R., and Müller W. Storage Networks Explained. Wiley, New York, 2004.

Storage Array

► Redundant Array of Independent Disks (RAID)

Storage Broker

► Request Broker

Storage Consolidation

HIROSHI YOSHIDA
Fujitsu Limited, Yokohama, Japan

Definition

The processes of centralizing the storage infrastructure resources of multiple servers to reduce management costs, achieve better service levels, and strengthen control over data.

Key Points

In small scale IT systems, each server has its own dedicated storage infrastructure (internal disks or DAS). However, as server numbers and the amount and importance of business data stored in the storage infrastructure increases, managing such dedicated storage infrastructure resources per server becomes difficult and expensive. To solve this problem, the dedicated infrastructure resources of servers are centralized to storage infrastructure resources shared by all servers, using storage networking technologies such as SAN and/or NAS.

Once dedicated storage infrastructure resources are consolidated to SAN and/or NAS resources, storage management operations can also be consolidated and centralized. For example, data backup is performed only once for the consolidated storage instead of individually for each server. This greatly reduces the cost of storage management. Another advantage is that expensive storage solutions such as disaster recovery using replicated data in remote sites can be shared by multiple servers with consolidated storage. The result is much improved data availability that can be achieved cost-effectively. In addition, from a data management and data security viewpoint; rather than having data spread over multiple servers, often managed by multiple administrators or divisions and based on disparate policies, data stored in consolidated storage can be managed based on more consistent policies.

Cross-references

► DAS
► ILM
► NAS
► SAN
► SRM
► Storage Network Architectures
► Storage Virtualization

Storage Controllers

► Storage Devices

Storage Devices

KALADHAR VORUGANTI
Network Appliance, Sunnyvale, CA, USA

Synonyms

Tapes; Tape libraries; CDs; DVDs; Optical storage; WORM; Storage controllers; NAS servers; Storage servers; Flash

Definition

One of the goals of database, file and block storage systems is to store data persistently. There are many different types of persistent storage devices technologies such as disks, tapes, DVDs, and Flash. The focus of this write-up is on the design trade-offs, from a usability standpoint, between these different types of persistent storage devices and not on the component details of these different technologies.

Historical Background

From a historical standpoint, tapes were the first type of persistent storage followed by disks, CDs, DVDs, and Flash. Newer types of memory technologies such as PRAM and MRAM are still in their infant stages. These newer non-volatile memory technologies promise DRAM access speeds and packaging densities, but these technologies are still too expensive with respect to cost/gigabyte.

Foundations

- *Tapes/Tape Libraries:* Tape readers/tape head, tape library, tape robot, and tape cartridge are the key components of a tape subsystem. Tapes provide the best storage packaging density in comparison to other types of persistent storage devices. Tapes do not provide random access to storage. Data on tapes can be stored either in compressed or uncompressed format. Unlike disks, tape cartridges can be easily transported between sites. Most organizations typically migrate data from older tape cartridges to newer tape cartridges once every 5 years to prevent data loss due to material degradation. One can employ disk based caches in front of tape subsystems in order to allow for tapes to handle bursty traffic. Tapes that provide Write-Once, Read Many (WORM) characteristics are also available. WORM tapes are useful in data compliance environments where regulations warrant guarantees that a piece of data has not been altered. DLT and LTO are currently the two dominant tape technologies in the market. Technology wise both these standards have minor differences. Finally, from a pure media cost standpoint, tapes are less expensive (cost per gigabyte) than disks and other forms of persistent media.
- *Disks/Storage Controllers/NAS Boxes:* Disks are the most widely used form of persistent storage media. Disks are typically accessed by enterprise level applications when they are packaged as part of the processing server box (direct attached storage model), or are part of a network attached storage box (NAS) and accessed via NAS protocols or, are packaged as part of a storage controller box and accessed via storage area network protocols (SAN). The current trend is for protocol consolidation, where the same storage controller provides support for both SAN and NAS protocols. Typically, the size of the storage controllers can vary from a few terabytes to hundreds of terabytes (refrigerator sized storage controllers). A storage controller typically consists of redundant processors, protocol processing network cards, and RAID processing adapter cards. The disks are connected to each other via either arbitrated loop or switched networks. Storage controllers also contain multi-gigabyte volatile caches. Disks are also packaged as part of laptops. There is a marked difference in the manufacturing process, and testing process between the enterprise class disks and commodity laptop class disks. Disks vary in their form factor, rotational speed, storage capacity, number of available ports, and the protocols used to access them. Currently, serial SCSI, parallel SCSI, serial ATA and parallel ATA, Fiber Channel, and SSA are the different protocols in use for accessing disks. Lower RPM and disk idle mode are new disk spin-down modes that allow disks to consume less power when they are not actively being used.
- *DVD/Juke Boxes:* DVDs and CDs are optical storage media that provide random access and WORM capabilities. Only recently, the multiple erase capacity of an individual CD, or DVD was less than the capacity of a single disk drive or tape cartridge. DVDs can store more data than a CD, and a high definition DVD can store more data than a DVD. There are numerous competing standards for CDs, DVDs and high definition DVDs, however, format agnostic DVD players and DVD writers are emerging. Usage of DVDs is more prominent in the consumer space rather than in the enterprise space. A juke box system allows one to access a library of CDs or DVDs. DVDs have slower access speeds than most types of disks.
- *Flash/SSDs/Hybrid Disks:* Flash is memory technology that has non-volatile characteristics. Flash memory has slower read times than DRAM. Moreover, it has much slower write times than DRAM. One has to perform an erase operation before one

can re-use a flash memory location. One can only perform a limited number of erase operations. Thus, the number of write operations determines the Flash memory life. SLC and MLC are the two different NAND flash technologies. SLC can be erased a greater number of times, and it has faster access times than MLC based flash. NAND flash has faster write and erase times than NOR flash. NOR flash has faster read times than NAND flash. NAND flash is used to store large amounts of data whereas NOR flash is used to store executable code. People are using MLC flash in cameras and digital gadgets, and are using SLC flash as part of solid state disks (SSDs). SSDs provide block level access interface (SCSI), and they contain a controller that performs flash wear leveling and block allocation. Hybrid disks that contain a combination of disks and Flash are emerging. Hybrid disks provide a Flash cache in front of the disk media. One typically can store meta-data or recently used data in the flash portion of hybrid disks to save on power consumption. That is, one does not have to spin-up the disk. Flash storage provide much better random access speeds than disk based storage.

Key Applications

Tapes are being used primarily for archival purposes because they provide good sequential read/write times. Disks are the media of choice for most on-line applications. Optical media (CDs, DVDs) are popular in the consumer electronic space. Flash based SSDs are popular for those workloads that exhibit random IOs. Disks are being used in Laptops, desktops and storage servers (SANs, NAS, DAS). Tape based WORM media and content addressable based disk storage are providing WORM media capabilities in tape and disk technologies, and thus, these technologies can be used to also store compliance/regulatory data.

Cross-references

▶ Backup and Restore
▶ Direct Attached Storage
▶ Network Attached Storage
▶ Storage Area Network

Recommended Reading

1. Anderson D., Dykes J., and Riedel E. More than an interface-SCSI versus ATA. In Proc. 2nd USENIX Conf. on File and Storage Technologies, 2003.
2. Toigo J. Holy Grail of Network Storage Management. Prentice Hall, Englewood Cliffs, NJ, 2003.
3. Voruganti K., Menon J., and Gopisetty S. Land below a DBMS. ACM SIGMOD Rec., 33(1):64–70, 2004.

Storage Grid

Kaladhar Voruganti
Network Appliance, Sunnyvale, CA, USA

Synonyms

Data grids; Content delivery networks; Peer to Peer network; Distributed databases; Cloud computing; Utility computing

Definition

In grid computing, storage and computing resources are geographically spread out and accessed via fast wide-area networks. Storage resources could either co-exist with computing resources, or they could exist separately from the computing resources. Databases, file systems or block storage devices can be accessed remotely across fast wide-area network based grids. A storage grid provides services for discovering storage resources, transferring data, recovering from unfinished data transfer failures, data authentication/encryption services, and data replication services for performance and availability purposes. Storage grids also typically provide the necessary mapping layers to access data from heterogeneous sources. Heterogeneity can be due to differences underlying system architectures, data formats, protocols used to access the data, and data organization. Finally, storage grids provide a global unified namespace across the resources, and one typically transfers large datasets (multi-terabytes) across the different nodes.

Key Points

Content Delivery Networks (CDNs) is a related area, where data are cached at secondary servers at various geographically distributed servers to cut down on data access latency. Peer-to-Peer networks is another related area where data are distributed across different peers in an ad-hoc manner and there is no central authority. Storage grids can be further classified based on the following criteria [1]:

Storage Grid Organization: Storage grid organization deals with how resources are organized in the system. Resources can be organized in a hierarchical manner, or in a monadic manner. In hierarchical model data exists at multiple sites. Each site in turn decides which of its children sites can have access to data. In a monadic grid data exists only at a single location and everyone accesses data from that location. Data sources in a grid can also be arranged in a federated model, where each site retains independent control over the data and its participation in the grid.

Data Transport: Data transport deals with transport issues, security issues, and fault-tolerance issues:

- *Transport:* Data can be transferred using protocols such as FTP and GridFTP. In addition, one could potentially employ overlay networks that provide caching functionality and control to the applications to directly control data transfer. Data can also be transferred via multiple parallel streams from one source location, and in a striped manner from multiple data source locations.
- *Security:* Authentication, encryption and authorization are the three security issues that are also applicable in grid environments.
- *Fault-Tolerance:* The primary fault-tolerance approaches are to restart a failed data transfer, or to have the ability to resume from the failed point. In some environments, if the destination node is not available, intermediate caches can temporarily store the data and then forward the data once the unavailable node comes back on line.

Data Replication and Storage: Data replication strategies can be classified in the following different ways:

- *Method:* Synchronous and asynchronous replication strategies are the two major classes of data replication mechanisms. In synchronous replication, updates are not acknowledged at the source until data has been successfully copied at the target locations. In asynchronous protocols, updates are immediately acknowledged at the source, and there is a lag in data consistency between the primary and secondary data copies.
- *Protocols:* Some grids employ open data transfer replication protocols, such as FTP or GridFTP. In open protocols, the catalog management becomes the responsibility of the application. Others employ a closed protocol which perform catalog management in an integrated manner.
- *Replication Granularity*: Data can be replicated at dataset, file, block, and database table level.
- *Replication Strategy:* Data can be replicated dynamically based on an objective function such as response time requirements, load balancing requirements or data consistency requirements, or data can be replicated statically based on an a priori schedule or on demand.

Resource Allocation and Scheduling: The goal of resource allocation and scheduling is to ensure that the data are located at the appropriate site in order to meet the performance and availability goals of the application. The settings for the following parameters can be varied in this regard:

- *Process Model:* The processes can be scheduled as independent tasks, as a bag of tasks or as part of a workflow. Workflow corresponds to a sequence of tasks, whereas, a bag of tasks correspond to executing the same task on different input parameters.
- *Objective Function:* Resource allocation and task scheduling is performed based on an objective function. The objective function tries to optimize load balancing, or business profit, or application performance. The object function is assigned at the task, bag of tasks or workflow level.
- *Scope of the Scheduler:* The scheduler decides whether to replicate/migrate data/process can try to optimize the utility function at the level of an individual application or at a more global community level.
- *Types of Tasks:* Data can be migrated, replicated, cached or remotely accessed in order to satisfy application's storage requirements. Alternately, the computation process can be migrated to the location where the data exists.

Storage grids are primarily used in scientific computing environments that deal with large amounts of data. It is usually not practical to replicate all of the data, and thus, grid architecture facilitates the remote access of large datasets across wide-area networks. However, variants of storage grid architectures such as CDNs are used to transfer streaming video, and P2P networks are used to shared audio and video data. Cloud computing is the new variant of utility computing where application, storage and server resources are managed by a service provider and clients remotely access these resources.

Cross-references

► Grid and Workflows
► Grid File

- ▶ Resource Scheduling
- ▶ Storage Protocols
- ▶ Storage Security

Recommended Reading

1. Venugopal S., Buyya R., and Ramamohan Rao K. A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing. ACM Comput. Surv., 38(1):1–53, 2006.

Storage Layer

▶ Storage Manager

Storage Management

Hiroshi Yoshida
Fujitsu Limited, Yokohama, Japan

Definition

The methods and tools used to manage storage devices (disk arrays, tape libraries, etc.), storage networking devices (fiber channel switches, etc.), storage-related components inside servers (host bus adaptors, etc.), and logical objects mapped on those devices (logical units, access paths, etc.). In general, the scope of storage management is limited to the management of storage infrastructure and does not handle the data stored in the infrastructure. The functions of storage management include device management, performance management, and problem management. Those functions are usually provided as software tools.

Historical Background

Storage management technologies have developed in parallel with the evolution of storage networking.

In the early 1990s, storage devices were used as DAS devices. Even in a DAS environment, storage management functions such as storage device management were required, and those functions were provided as dedicated software tools for specific vendors and/or devices, and those tools were often bundled with the hardware.

With the evolution of storage area networks (SANs) in the late 1990s, new requirements for storage management arose:

- Initially, SAN brought significant reduction of management cost through its storage consolidation capability. The storage capacity which could be managed by storage administrators was also greatly increased, compared to DAS environments. However, along with the growth of SAN, the number of connected devices and the amount and business importance of stored data increased; causing SANs to become more and more complex. This increased the "storage management gap," i.e., the gap between the decrease in storage hardware costs and the extreme increase in storage management costs. A reduction in the manpower required for storage management was urgently needed.
- To manage a SAN environment, both storage device management functions and network management functions, such as discovery, network configuration management, and network topology management, are necessary. In a networked environment, coping with hardware and software failures and performance problems also becomes much more difficult, compared to a DAS environment.
- SAN environments are usually constructed using products from multiple vendors. Storage management tools must cope with such multi-vendor environments. Therefore a new business model of providing software products which are independent from specific storage vendors' devices was established. This requirement also accelerated the standardization of the interfaces between storage devices and management software.

Foundations

General Classification of Storage Management Functions

Although storage management is necessary for all types of storage networking, the management of SANs is mainly discussed in the following description. This is because SANs are the most commonly used storage infrastructures, and feature the most typical management requirements. In general SAN environments, storage management is achieved by software tools running on a management server. Those tools provide functions such as device management, configuration management, performance management, and problem management.

- Device management and configuration management consists of functions to configure and to monitor storage devices (disk arrays, tape libraries, etc.), network devices (fiber channel switches, etc.), server components (fiber channel host bus adapters (HBAs), etc.), and relationship between those components. For example, disk array management provides the following functions:

– Monitoring and displaying the status of devices
– Creation/configuration/masking/mapping of logical units/logical unit numbers (LUNs)

Management of fiber channel switches and SAN configuration provides the following functions:

– Monitoring and displaying switch ports
– Collecting and displaying statistic information on switch ports
– Displaying SAN topology
– Configuration of zones
– Integrated and consistent control of multiple switches

As a SAN environment includes multiple levels of virtualization (e.g., disk arrays, volume managers, and virtualization network appliances) and access control features (zone, host affinity, and LUN mapping), the mapping of logical access paths on physical paths and the correlation between applications and physical storage devices tend to be complex. Configuration management should provide functions to visualize such mapping and correlation from the viewpoint of application and to configure multiple SAN components in a consistent manner.

- Performance management consists of functions to monitor, to analyze, and to display storage access performance based on statistic information collected from storage devices and fiber channel switches. It also includes functions to issue an alert in case that a specific parameter (e.g., device busy rate) exceeds the predefined threshold.
- Problem management consists of functions to monitor the status of storage devices as well as fiber channel switches and to notify the administrators and/or remote maintenance centers when problems such as hardware failures are detected.

"Storage management" is a very generic term and those elemental storage management functions are sometimes named "storage x management," e.g., "storage device management" and "storage configuration management." Note that the term "Storage resource management" represents a different concept from storage management mentioned here. It is usually used to indicate the functions to visualize and control usage of storage systems from the more application-aware or content-aware management viewpoint. It is described as a separate article.
Actual storage management software products include those typical functions as well as additional functions such as automation and the provision of integrated monitoring and operational views.

Elemental Technologies of Storage Management
Another aspect of understanding storage management is the technologies needed to implement storage management software. In general, the following internal functions are commonly required to implement storage management software.

- *Discovery* is a function to find storage devices in an SAN environment before knowing the topology of the SAN. When a new device such as a disk array system is connected to an SAN, it also has to be discovered.
- *Data collection* acquires necessary information, once storage devices are discovered. Information is collected through proprietary interfaces and/or standard interfaces such as SNMP (Simple Network Management Protocol) and SMI-S (Storage Management Initiative Specification). Data collection also stores that information in storage management repositories which are usually located in the storage management server. Collected information includes both information on the current status of storage resources such as configuration information and historical information such as accumulated performance information on storage devices.
- *Topology management* analyzes the topology of SANs based on the collected information. Topology management handles both physical network topology which represents the relationship between physical resources such as HBA ports, switch

ports, and storage device ports and logical network topology which represents the relationship between logical and/or virtualized resources such as LUNs, zones, and logical access paths.
- *Visualization* is a set of functions which display the information mentioned above on a management console and provide human interfaces to enable administrators to monitor and configure the storage resources easily.
- *Event processing* receives asynchronous events from storage resources, categorizes them based on the predefined policies, and notices administrators and/or remote maintenance centers if necessary, and records those events into event log files.
- *Security management* is a set of functions which are necessary to meet the appropriate security requirements. Security management includes authorization and access control features of administrators considering multiple administrative roles, single sign-on features among multiple storage management software products, management of credentials of managed storage resources for data collection, and logging functions for auditing all operations applied to the storage infrastructure to fulfill compliance requirements.

One important topic related to storage management implementation is how the interoperability between management software products and managed storage resources is achieved. At an early stage, storage management was implemented as management tools provided by individual storage vendors. Those tools were dedicated to the storage devices of respective vendors. Each storage vendor developed a proprietary interface and protocol which was applicable only to its storage devices and storage management software tools.

However, it became very common for datacenters to use storage devices provided by multiple vendors. Using multiple storage management tools with different looks-and-feels and manageability increased the management and labor costs in datacenters. This situation led to the new requirement that storage management software must be able to manage not only single vendor storage resources but multi-vendor storage resources. The goal was that every storage management software product could manage every storage resource. Standard interfaces and protocols which could be applied to the communication between management software and storage resources was crucial to achieving this goal.

Since the storage industry became aware early on of the importance of interoperability between products, standard interfaces and protocols for storage management were established as ISO standard SMI-S by the storage standards body SNIA (Storage Networking Industry Association).

Key Applications

Storage management is one of the essential features for administrative practice of computer system storage infrastructure. It allows administrators to configure, monitor, and control storage resources, particularly in SAN environments.

Storage management is also necessary as the basis of implementing higher level management, which is described as "management applications" later. For example, datacenter management requires storage management as part of its resource management as well as server management and network management. Another example is that information management requires storage management to manage the infrastructure in which information resides. In information lifecycle management, optimal storage devices are assigned to store information in accordance with its business value. Storage management is responsible for establishing the multiple storage device tiers which meet the different service level objectives and cost requirements.

Future Directions

The following areas will become more important in terms of storage management.

Integration of Management Software Including Storage Management

Currently a huge variety of management software products are used in large datacenters. Storage management software products must be integrated with those products.

To achieve consistent and automated resource management in a datacenter, storage management should be integrated with other resource management such as server and network management. This integration of resource management will provide capability such as the "provisioning" of a set of servers, storage devices, and network resources to an application. When administrators use this feature, they will not need to have detailed knowledge and skills in storage management, letting them manage storage devices in an SAN environment without regard to the SAN. This feature will be achieved through the cooperation of server management, network management, and storage management, which will configure internal connections between servers and storage devices automatically. Such integrated provisioning will greatly reduce management and labor costs in the datacenter.

Another example is integration with IT service management such as incident management, problem management, change management, release management, and configuration management. IT service management controls those management processes in a manner which is compliant to ITIL (Information Technology Infrastructure Library). In addition, configuration information needed for those management processes is stored in a CMDB (Configuration Management Database). To achieve integration with IT service management, storage management operations will have to be initiated as part of management processes by workflow managers, and configuration information on storage resources collected by storage management will also have to be federated with CMDB.

Visualization and Optimization from a Business Viewpoint

The business value proposition brought by IT systems becomes more and more important and needs to be clearly stated. On the other hand, to achieve business risk management, the impact of IT infrastructure outages on the customer's business also has to be clearly analyzed. To fulfill such requirements, the relationship between IT resources and business applications must be visualized. The IT resources must also be optimally configured and managed to achieve the performance and availability objectives of business applications. The most important key technology is dependency mapping between business applications and resources. Another important technology is policy management which allows customers to specify the criteria of system behavior based on their business requirements. Storage management is responsible for that capability with regard to storage resources.

Establishing Framework for Management Applications

In addition to the necessity of basic management capability such as the configuring and monitoring of storage devices, the importance of higher level "management applications" is continuously increasing. For example, such management applications include:

– Database performance management
– Resource provisioning
– Lifecycle management of storage resources and information lifecycle management
– Security management
– Automated management such as run book automation and autonomic management such as self configuration

Management applications are essential for achieving integrated management and business-aware management. The amount of management applications which are available on a storage platform is a key factor in achieving strategic use of storage and stored data in an enterprise.

In general, management applications monitor and control storage resources on the level of logical resources not on physical storage device level. In the main the information directly collected with current storage management tools, from storage hardware and the control functions of storage hardware, are sometimes too detailed for management applications. For example, a management application which provides provisioning capability is only aware of LUNs, their capacities, and the zoning configurations which restrict accessibility between servers and storage resources. There are also functions which many management applications use commonly to implement their storage-related capabilities. High-level interfaces which allow management applications to handle storage resources more logically, as well as a set of common components which help the development of management applications, are strongly required.

The storage industry has started to provide the higher-level interfaces and functional components as frameworks for management applications. One example is SNIA's attempt to standardize "management framework" as a higher level interface than the current SMI-S. It also includes common functional components such as discovery, data collection, topology management, data models, and policy management. Another example is "Aperi," which is the open source storage management software provided by open source community Eclipse. Aperi also provides the higher level management functions which are implemented on the SMI-S based storage management to management applications.

URL to CODE

The latest documents on SMI-S can be downloaded from the SNIA web site.

http://www.snia.org/tech_activities/standards/curr_standards/smi/

An open-source framework of storage management software can be downloaded from the Aperi project of Eclipse.

http://www.eclipse.org/aperi/

Cross-references

- ▶ DAS
- ▶ ILM
- ▶ LUN
- ▶ Mapping
- ▶ Masking
- ▶ SAN
- ▶ SMI-S
- ▶ SRM
- ▶ Storage Consolidation
- ▶ Storage Network Architectures
- ▶ Storage Networking Industry Association
- ▶ Storage Protocols
- ▶ Storage Virtualization
- ▶ Volume
- ▶ Zoning

Recommended Reading

1. Cummings, R. Storage Network Management, Storage Network Industry Association, 2004. Available at: http://www.snia.org/education/storage_networking_primer/stor_mngmnt/
2. Storage Network Industry Association. Storage Network Industry Association tutorials, 2007. Available at: http://www.snia.org/education/tutorials/

Storage Management Initiative-Specification

Hiroshi Yoshida
Fujitsu Limited, Yokohama, Japan

Synonyms

SMI-S

Definition

A standard storage management interface developed by the Storage Networking Industry Association (SNIA). SNIA describes SMI-S as follows: SMI-S defines a method for the interoperable management of a heterogeneous SAN, and describes the information available to a Web-Based Enterprise Management (WBEM) client from an SMI-S compliant Common Information Model (CIM) server and an object-oriented, XML-based, messaging-based interface designed to support the specific requirements of managing devices in and through SANs.

Key Points

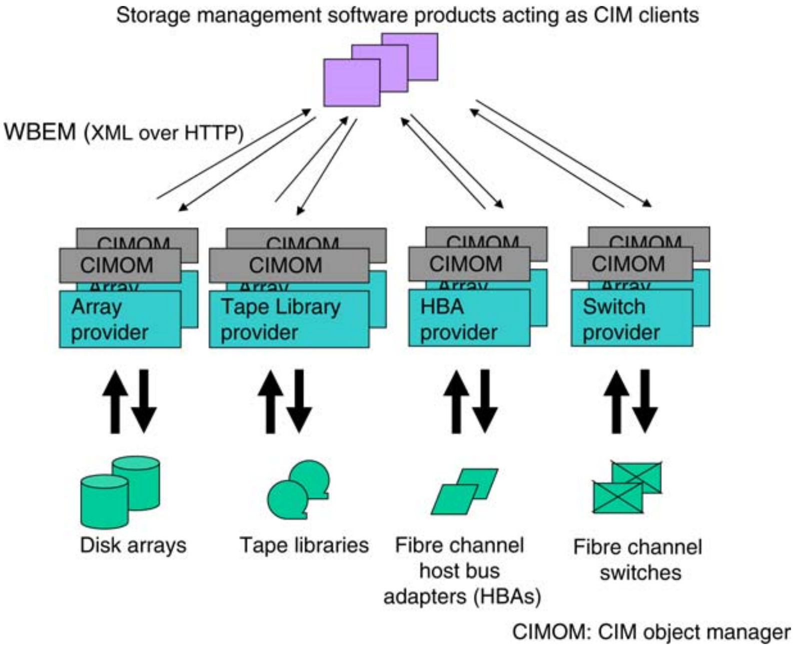
To implement storage management, methods to retrieve configuration information on storage components such as storage devices (disk arrays, tape libraries, etc.,), network devices (fiber channel switches, etc.,), and servers (hardware such as host bus adapters and software such as drivers and volume managers), plus methods to operate those components are necessary. To manage SAN environments composed of multi-vendor products, common methods of collecting information and operations must be standardized. In the standardization of those common methods, not only the standardized interfaces and protocols but also the semantics of the information and operations, i.e., object models of managed resources, must be defined.

To achieve SAN management in multi-vendor environments, the SNIA standardized on the Storage Management Initiative-Specification (SMI-S). Based on the Common Information Model (CIM) standardized by the Distributed Management Task Force (DMTF), SMI-S defines a common object model for each storage resource class as well as common methods of collecting information and operations using a Web-browser based management framework (Web-Based Enterprise Management, WBEM).

Figure 1 shows the architecture of SMI-S. It is based on the client-server model. Managed storage components act as CIM servers and storage management software tools act as CIM clients. CIM servers and CIM clients communicate by passing XML texts through HTTP. A SMI-S CIM server includes an HTTP server, an XML parser, an SLP agent used for discovery, and a CIM provider. The CIM provider implements actual CIM operations according to the profile defined for the corresponding storage resource class.

SMI-S version 1.3 includes the following storage management features and functionality:

Hardware Devices: SMI-S Providers
  Switches



Storage Management Initiative-Specification. Figure 1. The architecture of SMI-S.

- Arrays (fiber channel and iSCSI)
- Servers
- NAS devices
- Tape libraries
- Host profiles
- Host bus adapters

SMI-S Clients (software)

- Configuration discovery
- Provisioning and trending
- Security
- Asset management
- Compliance and cost management
- Event management
- Data protection

The standardization effort began in 1997 and was adopted formally by the SNIA as SMI-S in 2002. It was designated as a standard by ISO/IEC in January 2007.

Cross-references

▶ Storage Management
▶ Storage Networking Industry Association

Recommended Reading

1. Storage Network Industry Association. Storage management initiative-specification, 2007. Available at: http://www.snia.org/tech_activities/standards/curr_standards/smi/

Storage Manager

Goetz Graefe
Hewlett-Packard Laboratories, Palo Alto, CA, USA

Synonyms

Storage layer; Disk process

Definition

The storage manager is a software layer within a database management system. It relies on operating system primitives for I/O, synchronization, etc. and exposes records in storage structures such as B-trees and heaps. Its principal operations are creation and removal of storage structures as well as retrieval, search, scans, insertion, update, and deletion of records. For those operations, the storage layer provides concurrency control among threads and among transactions, as well as recovery from transaction, media, and system failures. Standard implementation techniques include locking and write-ahead logging. Using a buffer pool, records are made accessible in random access memory, although the permanent storage is on block-access media, typically on disk but possibly in flash memory.

The storage layer may support data compression, “blobs” (binary large objects), bitmaps in non-unique secondary indexes, hash indexes, multi-dimensional indexes, etc. It may also manage the catalogs needed to manage its objects, or it may leave metadata management to a higher software layer. For all the supported data structures, the storage layer provides utilities such as backup and restore, bulk insertions and deletions, logical and physical consistency check, defragmentation and other forms of reorganization, etc.

Historical Background

Design and scope of the storage layer are often similar to those of the RSS (research storage system) of the System R prototype. The basic architecture of the core functions has remained unchanged, including space management for records in pages, indexing and search, concurrency control and recovery. Many specific implementation techniques have changed, e.g., introduction of multi-dimensional indexes. Another change has been the transition from write-ahead logging with recovery from a read-only log to recovery by compensation of logical actions and guaranteed exactly-once execution of physical actions, both logged and even checkpointed during database recovery.

The architecture of utilities has also undergone some changes. The implementation of many storage layer functions now employs query execution and even query optimization, e.g., memory management policies or partitioning and pipelining for parallel execution. For example, creation of a new secondary index may scan existing secondary indexes rather than the primary index, or consistency check may aggregate facts gathered during a disk-order scan rather than navigate each index with many random I/O operations.

Sorting used to be a storage layer function because it was used almost exclusively for index creation, traditionally a storage layer function. Sorting can be used for many tasks, however, including complex query execution plans with merge join operations, etc., such that sort- and hash-based operations should both be part of the query processing layer.

The plethora of features and functions has led to complexity and high total cost of ownership for many installations. Extensible database management systems have not solved this problem, and the tension between "one size fits all" and "tailor-made" database systems might be resolved through factoring and mass-customization.

XML support is now available in the storage layer of many commercial database management systems, but further improvements in query processing over XML documents and databases will likely require further improvements of storage layer techniques.

In other words, as much as the storage layer may seem like a well-understood component of database technology, research, development, and competition continue unabated.

Foundations

This section describes a database system's storage layer by its external interfaces above and below, followed by internal components and data structures on disk and in memory, and completed by specific techniques for query processing, concurrency control and recovery, utilities, and catalogs.

Storage Layer Concepts

The most important services provided by a database system's storage layer revolve around indexes and records. Both of these are physical database concepts, quite different from logical database concepts such as table and row. For example, a single table may require many indexes due to redundant, secondary indexes as well as horizontal and vertical partitioning. Similarly, a single row might be represented by many records. Conversely, multiple tables may be clustered within a single index such that related information, e.g., about a purchase order and its line items, can be read, written, and buffered within a single disk page. Finally, a single record may contain information about multiple rows, e.g., a B-tree entry in a non-unique non-clustered index with a key value and a list of row identifiers.

The mapping between logical and physical concepts is usually provided by the relational layer and its query optimization component. Some implementations of the storage layer, however, provide some of this mapping, e.g., maintenance of all indexes for a row update, in order to maximize performance by minimizing the number of storage layer invocations.

Logical concepts include table, view, row, column, and domain; physical concepts include index, partition, record, and field. The word "key" is used both in the relational layer, where it restricts duplicates and null values, and in the storage layer, where it indicates index organization, sort order, or a search argument. It might be useful in some discussions to avoid the term "key" and to use "primary key," "foreign key," "index key," and "search key" instead.

Storage Layer Services

The storage layer provides access and updates for indexes and their records. Heaps can be thought of as indexes that map a record identifier to a record, with no capability to modify the record identifier. Indexes can be created and dropped. Records can be inserted, deleted, updated, scanned, and searched using an appropriate search key. Some indexes, e.g., heaps and clustered indexes, may generate unique row identifiers during insertion and possibly a new one during update. Such row identifiers can link all records representing a logical row, e.g., during retrieval using a non-clustered index or during deletion of a logical row.

All operations are part of transactions with concurrency control and recovery. The storage layer supports pre-commit for participation in coordinated commits in addition to the traditional immediate commit. The set of currently active transactions and their current state may be managed by the storage layer or by another component within the database management system.

If multiple threads invoke the storage layer at the same time, internal data structures are protected against concurrency problems using appropriate low-level synchronization.

If catalogs are managed by a higher software layer, the storage layer provides appropriate indexes and, most likely, special high-performance lock scopes and lock modes for metadata. For example, all metadata about a table may be covered by a single lock, and this lock may cover both the metadata and the data. Specifically, the weakest mode merely read-protects the metadata, whereas the strongest mode permits arbitrary changes to both metadata and data. Traditional read and write locks (shared and exclusive) imply read-protection on the metadata.

Storage Layer Requirements

A storage layer invokes two basic functions of the operating system: input/output and synchronization. Asynchronous I/O functions are valuable as otherwise they must be simulated with additional threads. Synchronization primitives that provide both shared and exclusive levels directly support the need of database management systems and their storage layers.

Storage Layer Components

The well-known components of a storage layer are the access methods (B-tree structure etc.), buffer pool, concurrency control and recovery. The less prominent components are memory management, disk space allocation, and catalogs. Asynchronous I/O could be separated into its own component, as could be latches, temporary structures (for sort- and hash-based operations), page structures (managing variable-length records), utilities for reorganization and consistency checks, backup and restore. Initial access to a database file, e.g., during recovery from a crash or when opening a database from a less-than-fully-trusted source, requires substantial logic and could be its own component, too.

Typically, multiple threads may invoke the storage layer at the same time; access and update of shared data structures within the storage layer is coordinated using latches. Those are equivalent to critical sections or locks in programming languages. Latch modes are typically shared and exclusive (read and write). Deadlock avoidance is required as latches do not participate in deadlock detection. Ordering latches based on levels is a standard technique to avoid deadlocks. Latches must not be retained while waiting for I/O to complete or while waiting for a database lock.

On-Disk Data Structures

The essential on-disk data structures are index for user data. This includes both primary, clustered indexes and secondary, non-clustered indexes on both tables and views. Additional data structures enable the essential ones: catalogs, free space management, and the recovery log. Each of those is special in their own way. For example, the recovery log is often mirrored on two devices in order to approximate the fiction of guaranteed stable storage.

Catalogs and their indexes often use the same on-disk format as user data. This is also possible for free space management, in particular if bitmap indexes are supported. The free space information is crucial during initial access to a database, i.e., system boot and recovery must be supported.

There is also the issue of free space management within each page. Each page typically consist of a page header, space for variable-length records, and an indirection vector that indicates the location of each record. A format version number in the page header enables incremental improvement of the database format without unloading and reloading entire databases. A “page LSN” (log sequence number) indicates the most recent log record pertaining to the page and is essential for exactly-once change application in modern recovery schemes.

Many database management systems and their storage layer support multiple index formats, e.g., hash indexes, heaps, multi-dimensional indexes, column storage, etc. All of them can be mapped to B-trees with reasonable efficiency in time and space, with a great savings in implementation effort. For example, implementation and testing for high scalability through a fine granularity of locking is a very substantial effort required for each storage structure.

In addition, bitmap indexes can be realized as a form of compression for non-unique non-clustered indexes. Even master-detail clustering (e.g., purchase orders and their line items) can be implemented relying merely on the sort order of B-trees and appropriate record formats. Finally, B-trees can be adapted to support "blobs" (binary large objects) for unstructured data. XML documents and other semi-structured data can be mapped to blobs plus traditional indexes for efficient search.

In-Memory Data Structures

The most prominent in-memory data structures within the storage layer are the lock manager's hash table and the buffer pool including buffers for the recovery log. Other data structures enable transaction management, checkpoints, device management, and asynchronous I/O.

In addition to the direct images of on-disk pages in the buffer pool, a storage layer may cache high-traffic data in data structures designed for fast in-memory access. Catalogs and bitmaps for free space management are obvious candidates. In order to achieve "in-memory performance" for user data and their indexes, interior B-tree nodes can be augmented in the buffer pool with in-memory pointers to child nodes also in the buffer pool, in a special form of pointer swizzling. In all cases, update propagation between cache and disk page images in the buffer pool must be ensured.

Query and Update Processing

The storage layer serves query and update processing but does not drive it. In addition to providing in-memory access to needed database pages, the storage layer speeds up query and update processing with prefetch, read-ahead, shared scans, and write-behind.

The storage layer may also provide automatic maintenance of non-clustered indexes. This design only applies to centralized systems or to parallel systems with local indexes, i.e., partitions of indexes aligned with the partitions of the table. This design forces row-by-row maintenance, although index-by-index maintenance can be required for correctness (certain updates of unique indexes) or for performance (sorting large sets of changes as appropriate for each index).

In very traditional designs, the storage layer may provide the navigation from non-clustered index to clustered index during query execution, but coupling these accesses inhibits many beneficial techniques such as covering a query with a non-clustered index alone, sorting references obtained from a non-clustered index, index intersection for conjunctive predicates and index union for disjunctive predicates, joining non-clustered indexes of the same table to cover a query not covered by any one index, and joining non-clustered indexes of two tables as a form of semi-join reduction.

Concurrency Control

Concurrency control is a very important service provided by the storage layer, both latching to protect in-memory-data structures from conflicting threads and locks to protect database contents from conflicting transactions. Alternatives to locking ("pessimistic concurrency control") include validation ("optimistic concurrency control") and versioning ("multi-version concurrency control"). Transactional memory may become an alternative to latching.

Locking and latching are described in detail elsewhere.

Logging and Recovery

Logging and recovery are also very important services provided by the storage layer, including transaction rollback, crash recovery, and media reconstruction. Since the units of recovery cannot be larger than the units of concurrency control, a high-concurrency system with key value locking or row-level locking cannot rely on page-based recovery provided by, for example, the file system or a network-attached storage service. The storage layer may provide log-shipping or continuous log-based replication using a "hot stand-by" database copy perpetually in recovery.

Logging and recovery are described in detail elsewhere.

Utilities

The broad term "utilities" covers all those operations needed for a complete database management system product but not directly associated with query processing and transaction processing. Examples include index creation and removal, defragmentation and other forms of reorganization, moving and partitioning data, backup and restore, statistics creation and update, consistency checks and repairs, etc. Catching up on deferred maintenance can apply to materialized views, indexes, statistics, caches, and replicas.

Utilities may be offline, online, i.e., permitting user transactions to read and modify database data while the utility is scanning or reorganizing them, or incremental, i.e., the utility operation's effects such as index creation become useful to user transactions in multiple discrete steps.

As many utility operation move data similar to a query execution plan, and since similar services are required such as memory management, partitioning and pipelining for parallel execution, etc., many utilities can be implemented using the query processing component.

Key Applications

A storage manager is useful in many applications, practically all applications that map collections of records to pages on persistent storage. This includes entertainment software such as music players, personal productivity applications such as e-mail clients, and server-side data management applications such as mail servers and database management systems. A storage manager that provides buffering and transactions for both records and large fields (such as pictures, sound tracks, videos, messages, and documents) is even more widely useful.

Future Directions

While the basics of access methods, buffer pool management, concurrency control and recovery are all well understood and documented, the need for further development continues.

Queuing

Some database management systems have integrated queuing into their feature set. It enables access patterns typical for workflow applications, electronic mail, and service-oriented architectures. Technical challenges include hotspots for both insertion and deletion as well as transaction semantics that link data records into messages and "conversations" among automatic processes and human users.

XML Support

XML is not only a message format but also a storage format, in particular for human-authored documents and in service-oriented architectures based on message passing. XML in databases creates challenges for storage, compression, fine-grained concurrency control and recovery, consistency enforcement and verification, and query processing. While initial research prototypes and even commercial implementations exist, their optimization and adaptation for large applications is not yet complete.

Transactional Memory

Forthcoming many-core processors require, for performance and power-efficiency, data structures and algorithms that permit very high degrees of parallelism as well as appropriate concurrency control among concurrent software threads. Transactional memory is a promising approach to these issues. Hardware-assisted transactional memory enables very fast execution of critical sections as well as guaranteed success based on automatic rollback and re-execution.

Self-Tuning, Self-Repair, Total Cost of Ownership

Perhaps the greatest challenge in the design and implementation of a storage layer is the total cost of ownership, i.e., the amount of human attention and trouble-shooting required to ensure the desired levels of availability, integrity, and performance. For example, can the storage layer software prevent data loss unless a user knowingly and deliberately accepts a risk? Such a storage layer would have to require, during initial deployment, that multiple storage devices with independent failures be specified, among many other things. Similarly, could the storage layer software assume all responsibility for tuning the set of indexes, or could there be a standard interface to other relevant components such as the relational layer within a relational database?

Cross-references

- ▶ Autonomic Computing
- ▶ B-Tree
- ▶ B-Tree Locking
- ▶ Buffer Pool
- ▶ Concurrency Control and Recovery
- ▶ Indexing
- ▶ Self-Tuning Database Systems
- ▶ Transaction

Recommended Reading

1. Carey M.J., DeWitt D.J., Franklin M.J., Hall N.E., McAuliffe M.L., Naughton J.F., Schuh D.T., Solomon M.H., Tan C.K., Tsatalos O. G., White S.J., and Zwilling M.J. Shoring up persistent applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 23(2):383–394, 1994.
2. Chamberlin D.D., Astrahan M.M., Blasgen M.W., Gray J., King W.F., Lindsay B.G., Lorie R.A., Mehl J.W., Price T.G., Putzolu G.R., Selinger P.G., Schkolnick M., Slutz D.R., Traiger I.L., Wade B.W., and Yost R.A. A history and evaluation of system R. Commun. ACM, 24(10):632–646, 1981.
3. Härder T. and Reuter A. Principles of transaction-oriented database recovery. ACM Comput. Surv., 15(4):287–317, 1983.
4. Hellerstein J.M., Stonebraker M., and Hamilton J.R. Architecture of a database system. Found. Trends Databases, 1(2):141–259, 2007.
5. Stonebraker M. Retrospection on a database system. ACM Trans. Database Syst., 5(2):225–240, 1980.

Storage Network Architectures

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Definition

Storage Network Architecture is the conceptual structure and logical organization of a network whose main purpose is to transfer data between storage devices and servers and among storage devices. The term Storage Network is identified with such a network, but is sometimes used to refer to a storage system communicating over a network. Related terms such as Storage Area Network and Network Attached Storage are described in separate entries. Note that usages of terms related to storage network architectures may depend on contexts at times.

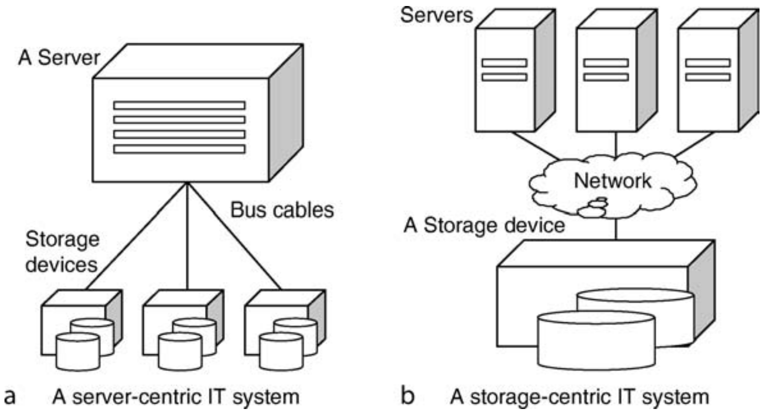
Historical Background

The first version of SCSI was released in 1986. SCSI then became deployed in many open systems, thus acquiring the position of the standard IO technology. However, after Fibre Channel technology was invented in the late 1990s, Fibre Channel rapidly extended its use in the market. Recent mid-range and high-end storage systems have deployed Fibre Channel as the standard IO technology. Alternative storage network technologies such as iSCSI may be available in entry-level systems. The SCSI bus technology has been replaced with Fibre Channel and iSCSI, but the SCSI protocol is still effectively used on top of such new network technologies.

Foundations

In conventional IT systems, storage devices such as disk drives, disk arrays, tape drives and tape libraries are connected only to a single server. Such an IT system is often referred to as a server-centric system, in which a storage device is considered a dedicated peripheral device of a server to which the storage device is connected. Small Computer System Interface (SCSI) is the main technology of network infrastructure used in the server-centric system. Storage devices are connected together to a server by SCSI bus cables and provide block-level I/O services to the server using the SCSI protocol. The scalability of such bus technologies is rather limited. A single SCSI cable can be at most 25 m long, and allows a maximum of 15 storage devices to be connected. Storage space which a single server can accommodate is thus severely limited. A server is not able to directly access storage devices connected to another server. Instead the server has to access such storage devices indirectly through a Local Area Network (LAN). The inflexibility of interconnection thus scatters and duplicates data management functions among multiple servers. Storage management may be optimized within each server, but should be far from global optimization of the entire system. Such issues were not visible when expensive microprocessors limited the system capability. However, recent technology innovations have been decreasing the cost of high-speed microprocessors and explosively expanding the volume of managed digital data. The system capability is then more likely to be limited by the storage system, so the issues of the conventional server-centric systems have become more obvious.

In contrast, the innovations of network technologies have given new possibilities of directly transferring data between storage devices and servers and among storage devices. All servers connected to a storage network are able to access all storage devices in the same network. Such a flexible connection helps storage devices to be consolidated in the network, thus isolating storage resource management from the server. That is, storage resources are placed and managed within a storage network, and servers are then positioned around the network. This type of IT system is often referred to as a storage-centric system. Figure 1 illustrates a server-centric system and a storage-centric system.



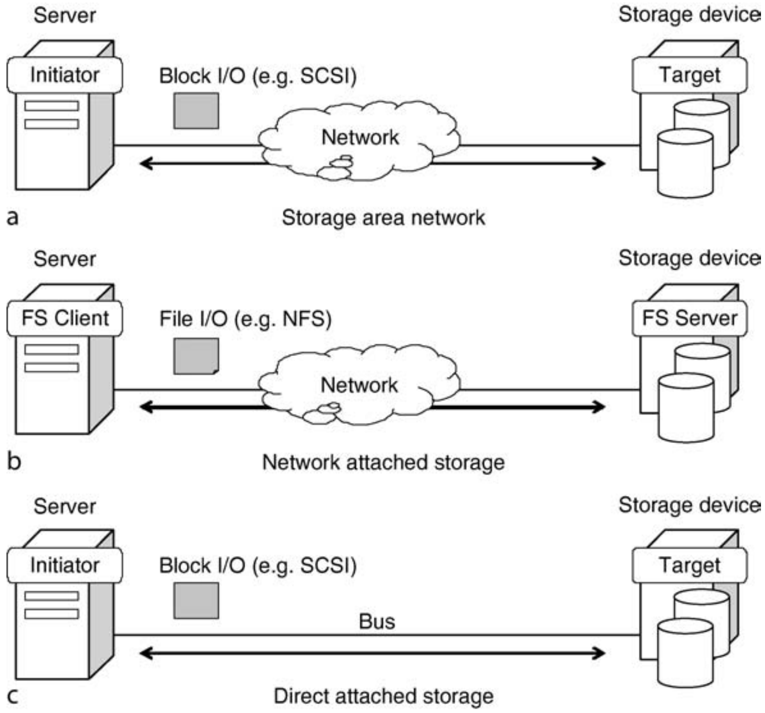
Storage Network Architectures. Figure 1. A server-centric IT system and a storage-centric IT system.

Storage networks have given system designers several alternatives for linking storage devices and servers. Currently available interconnections can be grouped into three architectures: Storage Area Network, Network Attached Storage and Direct Attached Storage. This categorization is widely accepted although it may not be formal. Figure 2 presents an illustrative comparison of these architectures. Separate entries give a formal definition of each storage network architecture.

A Storage Area Network, which is often abbreviated to SAN, is a network which mainly transfers data from/ to storage devices. A SAN is used to connect storage devices with servers and with other storage devices. A storage system which is implemented over a network is sometimes referred to as a SAN too. A SAN provides block-level I/O services. Most SANs are implemented on top of Fibre Channel technology, although alternative technologies such as internet SCSI (iSCSI) are available for SANs. A SAN which is implemented on Fibre Channel technology is sometimes referred to as Fibre Channel SAN.

Fibre Channel is a gigabit-level network technology, which is mostly used for SANs at present. The Fibre Channel protocol stack is divided into two parts. The lower part defines fundamental network infrastructure, which can be further subdivided into four layers (FC-0–FC-3). FC-0 defines transmission media such as optical/electrical interfaces, cables and connectors. FC-1 specifies signal encoding and decoding (8b/10b conversion and 40b ordered sets) and link controls. FC-2 defines frame formats, frame transmission management (sequences and exchanges) and flow controls. FC-3 defines common services such as multipathing, On top of these fundamental layers, the higher part (FC-4) defines protocol mappings to application protocols. When Fibre Channel is used for SANs, Fibre Channel Protocol for SCSI (FCP) maps the Fibre Channel infrastructure to the SCSI protocol. Fibre Channel is characterized by its powerful transmission capabilities such as serial transmission, low error rate and low latency. Processing of the Fibre Channel protocol is usually implemented in host bus adapter (HBA) cards at the hardware level so as to relieve servers' processors. These attributes are preferable for SANs, however, they increase the cost of Fibre Channel network devices. Fibre Channel SANs are thus deployed mainly in mid-range and larger IT systems. On the other hand, iSCSI is an approach to exploit the IP network infrastructure so that SANs can be installed and operated at much lower cost. iSCSI, which is placed on the top of the TCP/IP protocol stacks, encapsulates SCSI data in TCP/IP packets. At present, iSCSI is used for connecting servers and entry-level storage systems. Internet Fibre Channel Protocol (iFCP) and Fibre Channel over Internet Protocol (FCIP) are alternative approaches to use TCP/IP technology for SANs. These protocols can transmit Fibre Channel frames over IP networks. In contrast to iSCSI, iFCP and FCIP are mainly used for connecting remotely distant SAN islands. Note that Fibre Channel and iSCSI are network technologies that only replace the classical SCSI bus technology. The SCSI protocol is still utilized on top of such network technologies even in today's storage-centric systems.

Network Attached Storage is a storage device that is connected to a network and provides file access services. Network File System (NFS) and Common Internet File System (CIFS) are two major protocols used



Storage Network Architectures. Figure 2. Three storage network architectures.

for NAS networks. That is, NAS is used over IP network technology.

The history of NAS can be traced back to file sharing functions provided by operating systems. NFS and CIFS were developed for sharing files between servers in the conventional server-centric system. A file server, which exports file access services to other computers, is a type of implementation of a NAS device. Recent NAS devices are comprised of dedicated hardware and software because of the increasing demand on reliability and performance. A diskless NAS device, specifically a NAS device which has only controllers but contains no disk drives, is sometimes referred to as a NAS gateway or a NAS head. A NAS gateway/head can provide NAS clients with file access services to other storage devices that only export block-level I/O services. That is, a NAS gateway/head could be considered a service bridge between a SAN and a NAS network. Since NAS systems are based on TCP/IP technology, poor access performance is typically observed in comparison with a SAN. However, NAS systems have the strong benefit of exploiting existing IP network resources. The cost effectiveness of NAS systems has expanded their use especially in the entry-level markets.

The conventional storage system architecture, in which storage devices are connected to a single server via a SCSI bus cable, has been renamed to Direct Attached Storage after new storage network architectures such as SAN and NAS appeared. Direct Attached Storage is often abbreviated to DAS.

Key Applications

The flexible interconnection provided by storage networks enables storage devices to communicate with each other. In such a system, not necessarily all the functions need to be executed on server processors. Instead, executing some software codes on storage devices may be more efficient. Actually, storage developers are accommodating different applications into their storage devices. Below are described major storage network applications.

LAN-free Backup: in the server-centric storage architecture, a dedicated server connected to a LAN is usually responsible for creating and managing backup copies. The backup server reads data from other servers through the LAN and writes the data to the archiving storage, such as disk arrays and tape libraries, connected to the backup server. In contrast, the storage-centric architecture enables all the servers to access all the storage devices. The backup server can thus copy data directly from the source storage devices to the archiving storage device. Such LAN-free backup can be considered an approach of moving the copy traffic from the LAN toward the SAN.

Server-free Backup: server-free backup is a more advanced solution, in which storage devices or network devices connected in a storage network directly make a backup copy. Thus, the storage network does manage backup copies without any dedicated backup servers. A copy function which is incorporated in storage devices and/or network devices is often referred to as third-party copy.

Remote Replication: remote replication, which can be seen as a type of server-free backup, keeps a fresh backup copy in a remote data center. Communication between a local data center and a remote data center usually involves non-negligible latency. Two techniques are used for remote replication. Synchronous remote replication forwards a given write command to a remote storage device and then commits the command after the forwarded command is acknowledged by the remote device. This strict mode can synchronize data between the two storage devices all the time; no data would be lost even if a severe disaster damages the local storage device. However, high communication latency between the devices is likely to affect the response time of write commands, thus degrading application performance. In contrast, asynchronous remote replication commits a write command without waiting for the command to be acknowledged by the remote device. Inter-device communication latency would be invisible, but data coherence could not be guaranteed.

Data Sharing and Code Conversion: in a storage-centric environments, a storage device is shared by multiple servers. Sharing the data among multiple servers is also a natural approach. However, presentation forms of data usually depend on microprocessor architectures and operating systems. A file written by a server to a storage device may not be directly interpreted by another server. In the conventional server-centric system, the dedicated application running on the server converts data formats and character codes so that the application can interpret the data appropriately. Such code conversion facilities are being implemented in storage infrastructure. That is, when a server tries to read data stored in a storage system, the storage system converts and then exports the data to the server. Code conversion functions implemented in storage networks are deployed for downsizing from mainframes towards open systems and for file sharing between different types of machines.

SAN File system: a SAN file system is a file system which exports services for accessing files stored in storage devices connected to a SAN. A volume of a SAN file system is often shared among multiple servers. Thus, concurrency control is a key technology for a SAN file system. A SAN file system is deployed in many high-performance clusters and also in several high-end NAS systems.

Cross-references

- ▶ Direct Attached Storage
- ▶ IP Storage
- ▶ Network Attached Storage
- ▶ SAN File System
- ▶ Storage Area Network
- ▶ Storage Management

Recommended Reading

1. Benner A.F. Fibre channel for SANs. McGraw-Hill Professional, 2001.
2. Clark T. IP SANS: a guide to iSCSI, iFCP, and FCIP protocols for storage area networks. Addison-Wesley Professional, Reading, MA, 2001.
3. Clark T. Designing storage area networks: a practical reference for implementing fibre channel and IP SANs. Addison-Wesley, Reading, MA, 2003.
4. Robert W., Kembel R. W., and Cummings R. The fibre channel consultant: a comprehensive introduction. Northwest Learning Association, 1998.
5. Storage Network Industry Association. The Dictionary of Storage Networking Terminology. Available at: http://www.snia.org/
6. Troppens U., Erkens R., and Müller W. Storage networks explained. Wiley, London, 2004.

Storage Networking Industry Association

Hiroshi Yoshida
Fujitsu Limited, Yokohama, Japan

Synonyms

SNIA

Definition

A non-profit trade association dedicated to the development and promotion of standards, technologies, and educational services, to empower organizations in the management of information.

Key Points

The SNIA works toward these goals by forming and sponsoring technical work groups, producing a series of conferences, building and maintaining a vendor neutral technology center, and promoting activities that expand the breadth and quality of the storage and information management market. With seven regional affiliates spanning the globe, SNIA represents the voice of the storage industry on a worldwide scale.

Cross-references

- ► Storage Management
- ► Storage Management Initiative-Specification

Recommended Reading

1. Storage Network Industry Association, Storage Network Industry Association, 2007. http://www.snia.org/

Storage of Large Scale Multidimensional Data

Bernd Reiner[1], Karl Hahn[2]

[1]Technical University of Munich, Munich, Germany

[2]BMW AG, Munich, Germany

Synonyms

Hierarchical storage management; HSM; Multidimensional database management system; Raster data management

Definition

An identified major bottleneck today is fast and efficient access to and evaluation of high performance computing results. This contribution addresses the necessity of developing techniques for efficient retrieval of requested subsets of large datasets from mass storage devices (e.g., magnetic tape). Furthermore, the benefit of managing large spatio-temporal data sets, e.g., generated by simulations of climate models or physical experiments, with Data Base Management Systems (DBMS) will be shown. Such DBMS must be able to handle very large data sets stored on mass storage devices. This means DBMS need a smart connection to tertiary storage systems with optimized access strategies. HEAVEN (Hierarchical Storage and Archive Environment for Multidimensional Array Database Management Systems) is specifically designed and optimized for storing multidimensional array data on tertiary storage media.

Historical Background

Large-scale scientific experiments or supercomputing simulations often generate large amounts of multidimensional data sets. Data volume may reach hundreds of terabytes (up to petabytes). Typically, these data sets are permanently stored as files in an archival mass storage system, on up to thousands of magnetic tapes. Access times and/or transfer times of these kinds of tertiary storage devices, even if robotically controlled, are relatively slow. Nevertheless, tertiary storage systems are currently for the common state of the art storing such large volumes of data, because magnetic tapes are much cheaper than hard disk devices. This will also be the future trend. Furthermore, tapes are a good example for Green-IT. The generation of new data will increase extremely, because of new satellites, sensors, parameters etc. Consequently, scientists need more and more capacity for storing these large amounts of data, and tapes are well prepared for this task.

Concerning data access in the High Performance Computing (HPC) area, the main disadvantages are high access latency compared to hard disk devices and to have no random access. A major bottleneck for scientific applications is the missing possibility of accessing specific subsets of data. If only a subset of such a large data set is required, the whole file must be transferred from tertiary storage media. Taking into account the time required to load, search, read, rewind, and unload several cartridges, it can take many hours to retrieve a subset of interest from a large data set. Entire files (data sets) must be loaded from the magnetic tape, even if only a subset of the file is needed for further processing.

Furthermore, processing of data across a multitude of data sets, for example, time slices, is hard to support. Analysis of dimensions been contrary to storage patterns and requires network transfer of each required data set, implying a prohibitively immense amount of data to be shipped. Another disadvantage is that access to data sets is done on an inadequate semantic level. Applications accessing HPC data have to deal with directories, file names, and data formats instead of accessing multidimensional data in terms of area of interest and time interval. Examples of large-scale HPC data are climate-modeling simulations, cosmological experiments and atmospheric data transmitted by satellites. Such natural phenomena can be modeled as spatio-temporal array data of some specific dimensionality. Their common characteristic is that a huge amount of Multidimensional Discrete Data (MDD) has to be stored. For overcoming the above mentioned shortcomings, and for providing flexible data management of spatio-temporal data, HEAVEN (Hierarchical Storage and Archive Environment for Multidimensional Array Database Management Systems) was implemented [5].

Foundations

In order to implement smart management of large-scale data sets held on tertiary storage systems, HEAVEN combines the advantages of efficient retrieval and manipulation of data sets by using multidimensional array DBMS, and storing big amounts of data sets on tertiary storage media. This means the DBMS must be extended with easy to use functionalities to automatically store and retrieve data to/from tertiary storage systems without user interaction. A description of related work can be found in [5].

Such intelligent concepts are implemented within the European funded project ESTEDI, and integrated into the kernel of the multidimensional array DBMS RasDaMan (Raster Data Management). RasDaMan is designed for generic multidimensional array data of arbitrary size and dimensionality. In this context, generic means that functionality and architecture of RasDaMan are not tied to particular application areas. Figure 1 depicts the architecture of HEAVEN.

One can see the original RasDaMan architecture with the RasDaMan client, RasDaMan server and the underlying conventional DBMS (e.g., Oracle, which is used as a storage and transaction manager). The additional components for the tertiary storage interface are the Tertiary Storage Manager (TS-Manager), File Storage Manager and Hierarchical Storage Management System (HSM-System). The TS-Manager and File Storage Manager are included in the RasDaMan server. The HSM-System is a conventional product like TSM/HSM (Tivoli Storage Manager) from IBM. Such an HSM-System can be seen as a normal file system with unlimited storage capacity. In reality, the virtual file system of HSM-Systems is separated into a limited cache on which the user works (load or store his data), and a tertiary storage system with robot controlled tape libraries. The HSM-System automatically migrates or stages data to or from the tertiary storage media, if necessary.

Efficient Storage of Large Multidimensional Data

For overcoming the major bottleneck, i.e., the missing possibility of accessing specific subsets of data (MDD), the tiling concept of the RasDaMan DBMS is introduced. A MDD object consists of an array of cells of some base type (e.g., integer, float or arbitrary complex types), which are located on a regular multidimensional grid. An often discussed approach is chunking or tiling of large data [1,8,2]. Basically, chunking means the subdividing of multidimensional arrays into disjoint sub-arrays. Tiling is more general than chunking, because sub-arrays don't have to be aligned or have the same size. In RasDaMan, MDD can be subdivided into regular or arbitrary tiles. Consequently one MDD object is a set of multidimensional tiles. In RasDaMan, every tile is stored as one single Binary Large Object (BLOB) in the underlying relational DBMS. This makes it possible to transfer only a subset of large MDD from the DBMS (or tertiary storage media) to client applications, because access granularity is one singe tile. Also, the problem of inefficient access to data sets stored according to their generation process order is not any longer relevant with the tiling strategy of RasDaMan. This will mainly reduce access time and network traffic. The query response time scales with the size of the query box, not any longer with the size of MDD.

Data Export to Tertiary Storage Media

The export of data sets to tertiary storage media is two-tiered. The first step is the migration of the data sets from RasDaMan to the cache area (RAID-System) of the HSM-System. Transferring data sets from the hard disk of the underlying DBMS of RasDaMan to a RAID-System is very fast. Within a second step the migration from the cache area of the HSM-System to the tertiary storage media takes place. This process is performed by the HSM-System, and does not concern the RasDaMan system regarding I/O workload. During this process RasDaMan can execute another export process or user request in parallel.

Data Retrieval

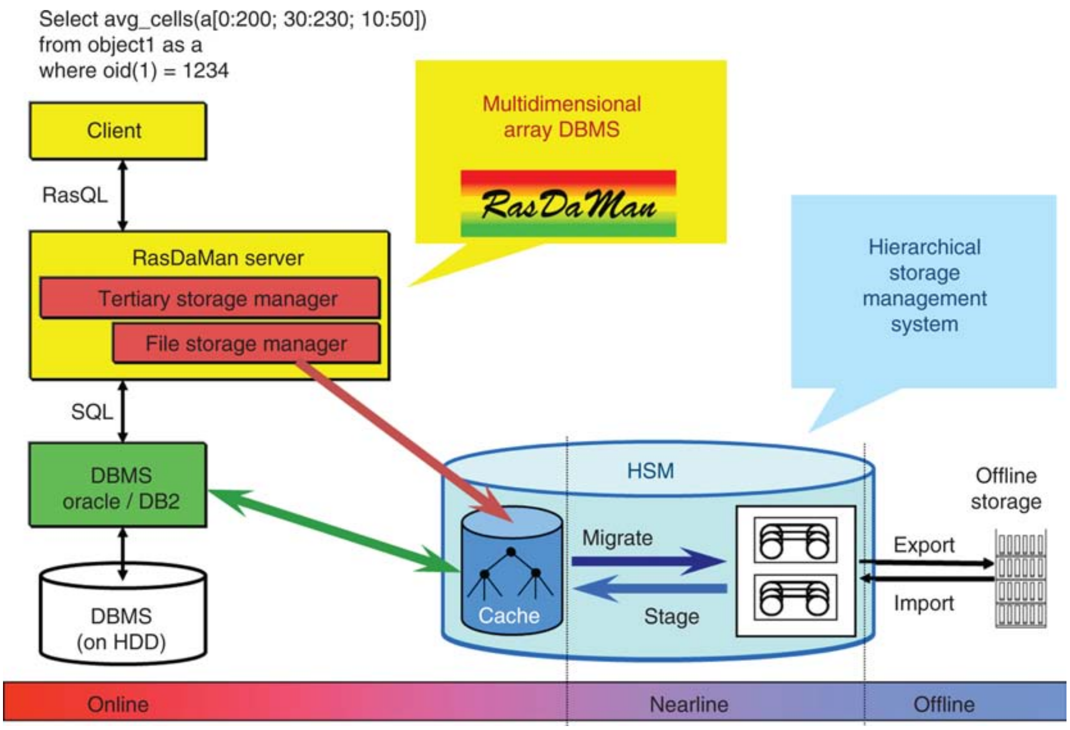
RasDaMan provides an algebraic query language RasQL, which extends SQL with powerful multidimensional operators like geometric, induced and aggregation operators. The primary benefit of such a complex query language is the minimization of data transfer between database server and client. Areas of interest can be specified with geometric operators, and complex calculations can be executed on the server side. Only the result is transferred to the client instead of the entire object [6,7]. With this feature, RasDaMan, overcomes the mentioned shortcoming of processing data across a multitude of data sets, because RasDaMan transfers only a minimum of data to the client. Furthermore, the query language RasQL provides data access on an adequate semantic level. Users can formulate queries such as: "average temperature on the earth surface of altitude y in the area of latitude x and longitude z."

With respect to data accessibility, three well-defined areas can be differentiated, i.e., online, nearline and offline area (Fig. 1). Data sets stored online means that data sets are stored on hard disk and, therefore, access time is very fast. Data sets stored on magnetic tape and stored in robot controlled libraries are called nearline data. Access time is much higher than with online access, but the process of data retrieval is done automatically. If data sets are stored in the offline area, user interaction is necessary for retrieving data sets (tapes are not robot controlled).

The new RasDaMan tertiary storage functionality is based on the TS-Manager module. If a query is executed, the TS-Manager knows (by metadata) whether the needed data sets are stored on hard disk (online area, DBMS or HSM-Cache) or on tertiary storage media. If the data sets are held on hard disk, the query will be processed very fast. If the data sets are stored on one or more tertiary storage media, the data sets must be imported into the database system (cache area for tertiary storage data) first. The import of data sets stored on tertiary storage media is done by the TS-Manager automatically whenever a query is executed and those data sets are requested. After the import process of the data sets is done, RasDaMan can handle the data sets in the normal way. The complexity of the RasDaMan storage hierarchy is completely hidden from the user.

Techniques for Reducing Tertiary Storage Access Time

The access time for tape systems is by order of magnitude slower than for hard disk devices. It is


Select avg_cells(a[0:200; 30:230; 10:50])
from object1 as a
where oid(1) = 1234

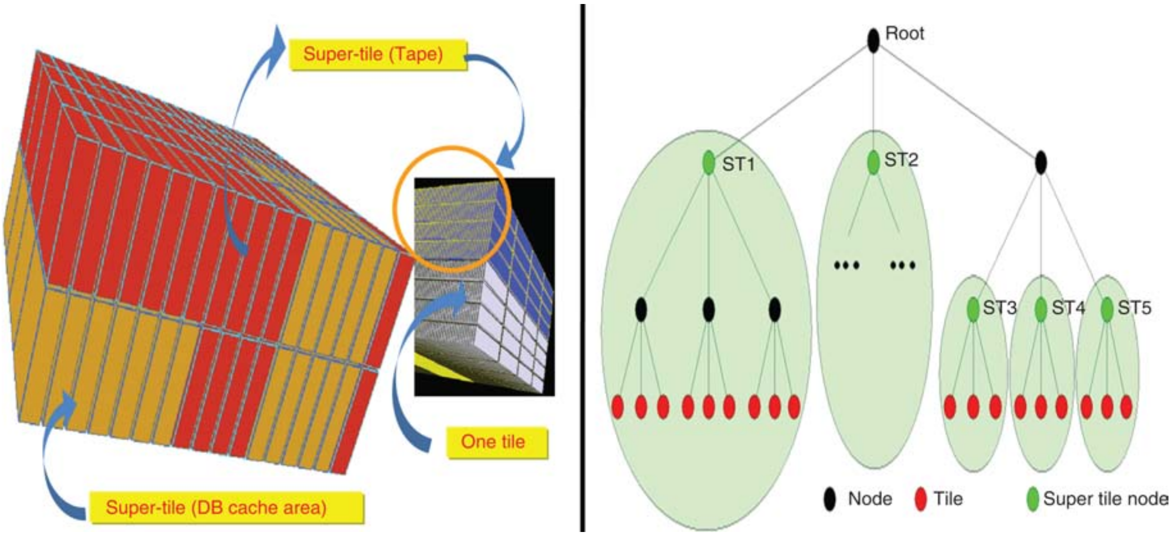
Storage of Large Scale Multidimensional Data. Figure 1. Extended RasDaMan architecture with tertiary storage connection.

important to use data management techniques for the efficient retrieval of arbitrary areas of interest from large data sets stored on tertiary storage devices. Hence, techniques that partition data sets into clusters based on optimized data access patterns and storage device characteristics, has to be developed. Therefore, methods for reducing tertiary storage access time, i.e., Super-Tile concept, data clustering and data caching are presented [5,6].

Super-Tile Concept In RasDaMan, DBMS tiles (BLOBs) are the smallest unit of data access. Typical sizes of tiles stored in RasDaMan range from 64 KB to 1 MB and are optimized for hard disk access [2]. Those tile sizes are much too small for data sets held on tertiary storage media. It is necessary to choose different granularities for hard disks and tape access because they differ significantly in their access characteristics. Hard disks have fast random access, whereas tape systems have sequential access with much higher access latency. The average access time for tape systems (20–180 s) is by order of magnitude slower than for hard disk drives (5–12 ms), whereas the difference between transfer rate is not significant [5,10]. For this reason, HEAVEN exploits the good transfer rate of tertiary storage systems, preserving the advantages of the tiling concept. The main goal is to minimize the number of media load and search operations.

It is unreasonable to increase the RasDaMan MDD tile size, because then RasDaMan would loose the advantage of reducing transfer volumes when accessing data on HDD. The solution is to introduce an additional data granularity as provided by the so-called Super-Tile. The main goal of the new Super-Tile algorithm is a smart combination of several small MDD tiles to one Super-Tile to minimize tertiary storage access costs. "Smart" means to exploit the good transfer rate of tertiary storage devices and to preserve advantages of other concepts like data clustering. The left side of Fig. 2 visualizes one three dimensional MDD with Super-Tile and tile granularity. An algorithm for computing Super-Tiles was developed which combines tiles of spatial neighborhood within the multidimensional object. For the realization, HEAVEN utilizes information of the RasDaMan R+ tree index [3,5].

The creation of the multidimensional index and the index access is no performance issue compared to data retrieval and data processing. Also, the primary criticism of the R-tree, that performance problems for very many dimensions occur, is not relevant for the application field. The used scientific data from various scientific fields (e.g., climat-modeling simulations, cosmological experiments, atmospheric data, earth observation, computational fluid dynamics) does not have more than five dimensions. Therefore, the integration of advanced multidimensional index methods, e.g., the bitmap index for scientific data proposed by Rishi Sinha and Marianne Winslet [9] or the UB tree proposed by Rudolf Bayer [4], was not considered.



Storage of Large Scale Multidimensional Data. Figure 2. Left: Visualization of one MDD with Super-Tile and Tile granularity. Right: Example R+ tree index of one MDD with Super-Tile nodes.

The conventional R+ tree index structure of the multidimensional DBMS was extended to handle such Super-Tiles stored on tertiary storage media. This means that information (whether tiles are stored on hard disk or on tertiary storage media) must be integrated into the index. Tiles of the same sub index of the R+ tree are combined into a Super-Tile and stored within a single file on tertiary storage medium (see right side of Fig. 2). Super-Tile nodes can exist on arbitrary levels of the R+ tree. Super-Tiles are the access (import/export) granularity of MDD on tertiary storage media, which preserve the advantages of the RasDaMan tiling concept (load minimum data) and exploit the good transfer rates of tertiary storage devices. More details about determining optimal file sizes on tertiary storage media can be found in [5].

Clustering Clustering is particularly important for tertiary storage systems where positioning time of the device is very high. The main goal is to minimize the number of search and media load operations and to reduce the access time of clusters read from tertiary storage system when subsets are needed. Clustering exploits the spatial neighborhood of tiles within data sets. Clustering of tiles according to spatial neighborhood on one disk or tertiary storage system, proceed one step further in the preservation of spatial proximity, which is important for the typical access patterns of array data, because users often request data using range queries, which implies spatial neighborhood.

The R+ tree index used to address tiles already defines the clustering of the stored MDD. With the developed Super-Tile concept intra Super-Tile clustering and inter Super-Tile clustering can be distinguished. The implemented algorithm for computing Super-Tiles maintains the predefined clustering of sub trees (of Super-Tile nodes) of the R+ tree index and achieves intra Super-Tile clustering (left side of Fig. 2). The export algorithm (export of Super-Tiles to tertiary storage) implements the inter Super-Tile clustering within one MDD. Super-Tiles of one MDD are written to tertiary storage media in the clustered order (predefined R+ tree clustering).

Caching In order to reduce expensive tertiary storage media access, the underlying DBMS of RasDaMan is used as a hard disk cache for data sets held on tertiary storage media. The general goal of caching tertiary storage data (Super-Tile granularity) is to minimize expensive loading, rewinding and reading operations from slower storage levels (e.g., magnetic tape). In the tertiary storage version of RasDaMan, requested data sets held on tertiary storage media are migrated to the underlying DBMS of RasDaMan (Fig. 1). The migrated Super-Tiles are now cached in the DBMS. After the migration, the RasDaMan server transfers only requested tiles from the DBMS to the client application.

The tertiary storage Cache-Manager evicts data (Super-Tile granularity) from the DBMS cache area only if necessary, i.e., the upper limit of cache size is reached. A special H-LRU (HEAVEN Least Recently Used) algorithm was developed, and together with the caching component of the HSM-System a caching hierarchy (Fig. 1) was built.

Conclusion

The main goal was the realization of optimized management of large-scale data sets stored on tertiary storage systems combined with access functionality like retrieval of subsets. Therefore, a multidimensional array DBMS for optimized storage, retrieval and manipulation of large multidimensional data was introduced. In order to handle hundreds of petabytes stored on tertiary storage media, an interface was presented connecting tertiary storage systems to the multidimensional array DBMS RasDaMan. The Hierarchical Storage and Archive Environment for Multidimensional Array Database Management Systems (HEAVEN) is specifically designed and optimized for storing multidimensional array data on tertiary storage media. For this reason, the query response time scales with the size of the query box and not with the size of the multidimensional data. This will dramatically reduce access time compared with the traditional access case.

Key Applications

HEAVEN is specifically designed for storing large-scale multidimensional array data (hundreds of terabytes) on tertiary storage media, and is optimized toward HPC. Addressed HPC areas are for example climate-modeling simulations, cosmological experiments and, atmospheric data transmitted by satellites.

Cross-references

- ▶ Archiving Experimental Data
- ▶ Clustering
- ▶ Data Partitioning

- ▶ Database Management System
- ▶ Disk
- ▶ Query Language
- ▶ Raster Data Management and Multi-Dimensional Arrays
- ▶ Rtree
- ▶ Storage Management

Recommended Reading

1. Chen L.T., Drach R., Keating M., Louis S., Rotem D., and Shoshani A. Efficient organization and access of multidimensional datasets on tertiary storage. Inf. Syst., 20(2), 1995.
2. Furtado P.A. Storage Management of Multidimensional Arrays in Database Management Systems, PhD Thesis, Technische Universität München, 1999.
3. Gaede V. and Günther O. Multidimensional access methods. ACM Comput. Surv., 30(2):170–231, 1998.
4. Ramsak F., Markl V., Fenk R., Zirkel M., Elhardt K., and Bayer R. Integrating the UB-Tree into a Database System Kernel. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 263–272.
5. Reiner B. HEAVEN – A Hierarchical Storage and Archive Environment for Multidimensional Array Database Management Systems, PhD Thesis, Technische Universität München, 2005.
6. Reiner B., Hahn K., Höfling G., and Baumann P. Hierarchical Storage Support and Management for Large-Scale Multidimensional Array Database Management Systems. In Proc. 13th Int. Conf. Database and Expert Syst. Appl., 2002, pp. 689–700.
7. Ritsch R. Optimization and Evaluation of Array Queries in Database Management Systems, PhD Thesis, Technische Universität München, 1999.
8. Sarawagi S. and Stonebraker M. Efficient organization of large multidimensional arrays. In Proc. 10th Int. Conf. on Data Engineering. 1994, pp. 328–336.
9. Sinha R.R. and Winslett M. Multi-resolution bitmap indexes for scientific data. ACM Trans. Database Syst., 32(3):2007.
10. Yu J. and DeWitt D. Processing satellite images on tertiary storage: a study of the impact of tile size on performance. In Proc. 5th NASA Goddard Conf. on Mass Storage Systems and Technologies, 1996.

Storage Power Management

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Definition

Storage Power Management is a process of improving the efficiency of electric power consumption of all the concerned storage resources including storage devices, storage controllers and storage network devices. Storage Power Management may sometimes cover related equipment such as power supplies and cooling apparatuses. The definition of "the efficiency" may depend on the situation; system designers and administrators often need to balance electric power reduction against performance degradation.

Historical Background

Electric power consumption of storage devices was discussed mainly with regard to battery-operated computing environments such as laptop PCs. However, due to the rapid growth of power consumption in data centers and the increased interest in environmental issues, much attention has recently been paid to electric power consumption of enterprise-level storage systems. Energy efficiency is recognized as a new direction for research and development of storage systems.

Foundations

Hard disk drives are main components of modern storage systems. In the disk drive, a spindle motor, which rotates metal platters at high speed, consumes most of the electric power. Gurumurthi et al. [3] report that the spindle motor can account for 81% of the power consumed by the disk. For a given disk drive, the power consumption of its spindle motor P theoretically relates to the angular velocity ω as

$$P = \frac{K_e^2 \omega^2}{R}$$

where Ke is a motor voltage constant and R is a motor resistance constant. In reality, since the spindle motor rotates the platters against air drag, the angular velocity ω may has a cubic or greater effect on the power consumption P. Disk drive manufacturers have increased the rotational speed to decrease the access latency and improve the transfer rate. Disk array developers have been accommodating a number of such high-speed disk drives into a single disk array enclosure. Thus, greater electric power consumption is often seen in enterprise-level storage systems.

Many commercial disk drives have a "stand-by" mode. While a disk drive is in the stand-by mode, its head is unloaded from the platters to the ramp and its spindle motor is completely suspended. The disk drive consumes much less electric energy in the stand-by mode than in the active (currently processing read/write requests) and idle (being able to start processing read/write requests immediately) modes. However, the transition to/from the stand-by mode involves non-negligible overhead of time and electric energy. Especially, spinning up a spindle motor to regular speed takes several to tens of seconds and consumes tens to hundreds of joules. Such a significant overhead associated with low-power modes is rarely seen in other computer components including processors and memory. Several commercial disk drives may have different low-power modes other than the stand-by mode. For example, a disk drive in a "low-rpm" mode may keep rotating the platters at lower speed with its head unloaded, accordingly consuming more power but involving less overhead.

The minimum time that the disk needs to be idle for the power saving achieved to exceed the control energy overhead is called "break-even time." The break-even time is a specific parameter of a disk drive. Assuming that the system could perfectly predict accesses that would be issued to a disk drive in the future, the system could spin down the disk drive after the disk becomes idle only if the idle period will be longer than its break-even time. In turn, the system could also spin up the drive in advance before new disk accesses are issued. This "oracle power management" gives the maximum possible energy saving for a given series of disk accesses. However, predicting the future disk accesses perfectly is impossible in reality. Alternative solutions have been studied so far.

Typical techniques of disk power management are based on idleness threshold. The simplest strategy is to spin down a disk drive to a low-power mode after a predetermined time has elapsed since the last disk access. This is based on the heuristic prediction that a disk drive is likely to continue to be idle if it has been idle for a long period. This traditional strategy is deployed in many commercial low-end disk drives, since it works effectively in end-user computing environments where the workload is dominated by interactive applications and users can accept reasonable spinning-up latency. More sophisticated techniques that try to tune the idleness threshold adaptively have been also investigated.

Dynamic Rotations Per Minute (DRPM) [3] is an attempt to exploit innovative disk drives that have the capability of dynamically changing the rotational speeds. Instead of completely spinning down the disk drive, this idea controls the rotational speeds adaptively by observing disk access performance. DRPM is helpful to balance between power and performance tradeoffs more flexibly compared with the conventional low-power modes. Gurumurthi et al. [3] validated potential benefits of DRPM techniques on a simulator. At present, disk drives that can change their rotational speeds are not commercially available but merely reported in papers.

In enterprise systems, a number of individual disk drives are incorporated into a disk array and managed by the array controller. Data layout among the disk drives is a key to power management of such systems. A disk array which has the capability of spinning down member disk drives is called Massive Array of Idle Disks (MAID). The original paper [2] of MAID investigated the caching strategy. Suppose that member disk drives of a given disk array can be divided to a small number of active disk drives and a large number of passive disk drives, and all the blocks that are exported to the server are originally located in the passive disk drives. By replicating hot blocks that are frequently accessed onto the active disks, the array can achieve long idle periods for the passive disks so that they may be spun down. Another paper [1] studied a block migration strategy called Popular Data Concentration (PDC), which clusters hot blocks onto particular disk drives in order to spin down the other drives.

This entry focuses on electric power consumption of disk drives, which are main components of modern storage systems. Other components such as RAID controllers and power supplies and other types of storage devices such as optical discs, electromagnetic tapes and solid-state memory should be more carefully studied in the future.

Key Applications

The rapid growth of electric power consumption has stimulated the economic demand of energy saving technologies. In addition, a variety of government-level restrictions and business-level standards are being considered to resolve or mitigate environmental issues. Much more attention is likely to be paid to Storage Power Management in the future.

Cross-references

- ▶ Deduplication
- ▶ Disk Power Saving
- ▶ Massive Array of Idle Disks

Recommended Reading

1. Carrera E.V., Pinheiro E., and Bianchini R. Conserving disk energy in network servers. In Proc. 17th Annual Int. Conf. on Supercomputing, 2003, pp. 86–97.

2. Colarelli D. and Grunwald D. Massive arrays of idle disks for storage archives. In Proc. 16th Annual Int. Conf. on Supercomputing, 2002, pp. 1–11.
3. Gurumurthi S., Sivasubramaniam A., Kandemir M., and Hubertus F. Reducing disk power consumption in servers with DRPM. IEEE Comput 36(12):59–66, 2003.
4. Hitachi Global Storage Technologies, Inc. Quietly Cool. White Paper, 2004.
5. Lu Y.-H. and Micheli G.D. Comparing System-Level Power Management Policies. IEEE Design Test Comput, 8(2):10–19, 2001.

Storage Protection

Kenichi Wada
Hitachi Limited, Tokyo, Japan

Synonyms

Data protection

Definition

Storage protection is a kind of data protection for data stored in a storage system. The stored data can be lost or becomes inaccessible due to, mainly, a failure in storage component hardware (such as a hard disk drive or controller), a disastrous event, an operator's mistake, or intentional alteration or erasure of the data.

Storage protection provides the underlying foundation for high availability and disaster recovery.

Historical Background

In 1956, IBM shipped the first commercial storage that had a hard disk drive. To protect data from bit errors on disk platters, the hard disk drive commonly uses cyclic redundancy check (CRC) and an error-correcting code (ECC).

CRC and ECC cannot protect data from a whole disk failure in which an entire disk becomes inaccessible (for example, because of a disk head crash). The IBM 3990, which was shipped in the 1980s, had the replication functionality in which two identical copies of data were maintained on separate media. This approach protected data from this kind of failure. Replication functionality can be implemented in many other layers of the computer system. Most DBMS support database replication. Some file systems and Logical Volume Managers have file or volume replication functionality. Further, many storage systems and storage virtualization appliances support volume replication functionality.

RAID (Redundant Array of Inexpensive Disks) is another technology for protecting data from whole disk failure. D. Patterson et al. published a paper "A Case for Redundant Arrays of Inexpensive Disks (RAID)" in June 1988 at the SIGMOD conference [6]. This paper introduced a five level data protection scheme. The term *RAID* was adopted from this paper, but currently RAID is an acronym for *Redundant Arrays of Independent Disks*. It is noted that the patent covering RAID level 5 technology was issued in 1978 [5].

RAID level 1 is a kind of replication. RAID level 2 to 5 can reduce the capacity required to protect data against disk drive failure than replication, but it is limited to protect disk drive failure. Replication, on the other hand, can be used to protect databases, file systems and logical volume. Further replication can be used for disaster recovery, if data are replicated remotely.

Foundations

Hard disk drives commonly use Reed-Solomon code [7] to correct bit errors. Data in hard disk drives is usually stored in fixed length blocks. Controllers in hard disk drives calculate ECC for each block and record it associated with the original data. When data are read, the controller checks data integrity using ECC. CRC can be used with ECC for detecting bit errors and/or reducing the possibility of correction error.

Most DBMS support database replication with master/slave relation between the original and the replica. The master process updates and transfer it to the slave. This type of replication can provide high availability to the client of the DBMS in case of storage system failures as well as server failures. Another type of database replication is multi-master, which is mostly used to provide high performance parallel processing. Both types can be either synchronous or asynchronous replication. In synchronous replication, updates made in original are guaranteed in the replica, note there may be some delay in asynchronous replication.

Volume replication by storage system is also widely accepted as data protection. There are synchronous and asynchronous replications, the same as database replication. Asynchronous volume replication is often used for long distance remote replication. It may prevent performance degradation caused by replication delay, but could cause some data loss in case of recovery. Synchronous replication, on the other hand, may provide no data loss recovery, but may cause performance degradation due to replication delay. Volume replication is also used within a local datacenter for online backup. Backup servers use replica volume for backup during original volume is online. To support this, a storage system can pause update delegation from original to replica volume.

RAID (Redundant Array of Independent Disks) is a set of disks from one or more commonly accessible disk subsystems, combined with a body of control software, in which part of the physical storage capacity is used to store redundant information about user data stored on the reminder of the storage capacity. The term *RAID* refers to a group of storage schemes that divide and replicate data among multiple disks, to enhance the availability of data at desired cost and performance levels. A number of standard schemes have evolved which are referred to as *levels*. Originally, five RAID levels were introduced [6], but many more variations have evolved. Currently, there are several sublevels as well as many non-standard levels. There are trade-offs among RAID levels in terms of performance, cost and reliability.

Key Applications

Storage protection is essential to achieve business continuity and legal compliance with adequate performance, cost, and reliability.

Cross-references

▶ Backup and Restore
▶ Checksum and Cyclic Redundancy Check Mechanism
▶ Continuous Data Protection
▶ Disaster Recovery
▶ Logical Volume Manager
▶ Point-in-Time Copy
▶ Redundant Arrays of Independent Disks
▶ Replication
▶ Write Once Read Many

Recommended Reading

1. ANSI. NFPA1600 Standard on Disaster/Emergency Management and Business Continuity Programs.
2. BSI. BS25999; Business Continuity Management.
3. Houghton A. Error Coding for Engineers. Kluwer Academic Publications, Hingham, MA, 2001.
4. Keeton K., Santos C., Beyer D., Chase J., and Wilkes J. Designing for disasters. In Proc. 3rd USENIX Conf. on File and Storage Technologies, 2004.
5. Ouchi N.K. System for recovering data stored in failed memory unit. US Patent 4,092,732, 1978.
6. Patterson D., Gibson G., and Katz R. A case for redundant arrays of inexpensive disks (RAID). In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988.
7. Sweeney P. Error Control Coding From Theory to Practice. Wiley, New York, 2002.
8. http://www.sec.gov/

Storage Protocols

Kaladhar Voruganti
Network Appliance, Sunnyvale, CA, USA

Synonyms

iSCSI; FCP; Parallel SCSI; SAS; ATA; SATA; NFS; CIFS; Samba

Definition

The emergence of networked storage has allowed organizations to de-couple their server and storage purchasing decisions. Thus, multiple application/database servers can share storage on the same network attached storage (NAS) server, or on a block storage controller via a storage area network protocol (SAN). This is unlike direct attached storage systems (DAS) where disks are connected to an application to a database server. In DAS environments, even if one only needs to add more storage, one has to also add more servers. Various types of storage networking protocols have emerged to support both NAS and SAN systems. Currently, the same storage boxes can support both distributed file (NAS) protocols such as NFS and CIFS, and block storage (SAN) protocols such as iSCSI and FCP (fiber channel). Most of the direct attached storage systems are moving from supporting parallel SCSI protocol to supporting serial ATA (SATA) or serial SCSI (SAS) protocols. The focus of this section is to set the context with respect to when they are used, and to compare the advantages and disadvantages of these different protocols.

Historical Background

Historically the fields of storage and networking were two separate fields. With the advent of network attached storage the fields of storage and networking have now converged. Many of the existing networking protocols are now also used to carry storage payload from storage devices to the application servers.

Foundations

- *SAN Protocol Analysis:* SAN protocols transfer SCSI commands and data over a transport protocol. SAN protocol is between SCSI initiators and SCSI targets. The initiators typically reside on the host (application server), and the targets reside on a storage controller box. The SCSI initiator can be software based or it can reside in a host bus adapter card. In the SAN protocol space the comparison is primarily between Fiber Channel and iSCSI. Previously, IBM's SSA protocol competed with Fiber Channel, but has been discontinued as it was not universally adopted by all the storage controller vendors.
  - *iSCSI:* iSCSI protocol transfers SCSI blocks and commands over TCP/IP networks. iSCSI protocol allows organizations to leverage their IP networks to transfer block storage data. Until the emergence of gigabit Ethernet networks, transfer of block storage data over Ethernet/TCP/IP stack proved to be not viable due to performance problems. Hardware TCP/IP off-load cards as well as multi-core CPU servers have alleviated the TCP/IP processing overhead. Software TCP/IP optimizations such as interrupt coalescing, and zero-copy optimizations have also reduced the CPU processing overhead. iSCSI based SANs have the following benefits over Fiber Channel based SANs:
    - *Interoperability:* Until recently, Fiber Channel devices from different vendors did not always interoperate with each other. Device interoperability certification is a much more elaborate and expensive process in the fiber channel world in comparison to the IP world.
    - *Distance:* IP networks have been designed to operate across large geographic distances. Fiber Channel networks have distance limitations and one typically needs channel extenders to extend their range.
    - *Cost:* An organization can leverage their IP network management expertise and devices for also transferring their storage traffic.
    - *Security:* The security protocols for IP networks have been well developed. Since Fiber Channel networks have been typically used behind fire-walls, the security aspects are being still developed.
  - *FCP (Fiber Channel):* Sending SCSI block commands and data over the Fiber Channel protocol is known as FCP. Fiber Channel protocol stack consists of physical, data link, network, and transport layer protocol specification. Fiber Channel protocol has the following benefits in comparison to TCP/IP stack:
    - *Performance:* Fiber Channel provides better performance than TCP/IP stack due to the following reasons:
      - *Hardware Offload:* The performance of first generation iSCSI HBAs was inferior to Fiber Channel HBAs. However, performance gap is being reduced in the newer generation iSCSI offload cards. The iSER and iWARP standards are trying to commoditize the RDMA standard for IP networks in order to bring down the cost of iSCSI offload cards. That is, these cards will be useful for additional protocols (not just iSCSI).
      - *Reservation Based Protocol:* Fiber Channel is a reservation based protocol instead of a retry based protocol. Thus, frames are not lost in Fiber Channel. The slow-start congestion control mechanism in TCP/IP is ill-suited for gigabit speed networks.

Currently, Fiber Channel protocol is the most commonly used SAN protocol in data centers.

- *Parallel versus Serial Protocols Analysis:* Parallel SCSI and ATA are two parallel protocols who have distance and device connectivity limitations. It is important to distinguish between the parallel SCSI transport protocol, and the SCSI block protocol which transfers data on top of the transport protocol. The physical wires for these parallel protocols are also quite wide and they make wiring a cumbersome process. Serial ATA (SATA), and Serial SCSI (SAS) have overcome the stringent distance and connectivity limitations of parallel SCSI and ATA transport mechanisms.

- *SCSI versus ATA Command Set Analysis:* The SCSI block protocol is transported over Fiber Channel (FCP), TCP/IP (iSCSI), and parallel SCSI, and serial SCSI (SAS) transport layers. The IDE/ATA protocol is transported over ATA and Serial ATA (SATA) transport mechanisms. It is important not to mix the block protocol with the underlying transport protocol. SCSI block and ATA block protocols have the following key differences:
  - *Command Queuing at the Device:* Previously SCSI protocol allowed for the queuing of multiple commands at the SCSI device, whereas, ATA protocol did not have this functionality. Recently, tagged command queuing functionality has been added to ATA protocol. Tagged command queuing also allows the device to optimize the order in which the queued commands are executed.
  - *Number of Connected Devices:* ATA protocol supported fewer devices per channel than the SCSI protocol. However, this limitation has been reduced in the serial ATA protocol with the emergence of port multipliers. Thus, 15 devices can be simultaneously supported using SATA.
  - *Checksums:* ATA protocol did not initially contain support for checksums on command and data. Later versions of the protocol have added checksum support for data.
  - *Hot-Plug of Devices:* ATA does not provide support for hot replacement of devices. Serial ATA has rectified this deficiency.
  - *Bus Mastering:* Previous versions ATA protocol did not allow to DMA data directly from the device into memory. However, recent versions of the protocol have overcome this limitation.
- *Block versus File-Based NAS Protocols:* NAS protocols provide a file level abstraction to the client applications, whereas SAN protocols provide a block level abstraction. NAS protocols allow for multiple hosts to share a file system namespace, whereas, SAN protocols allow multiple hosts to share a block level (SCSI device and LUN level) namespace. Previously, Fiber Channel based SANs provided better performance than IP based NAS protocols due to the implementation differences between the IP and Fiber Channel transport protocol stacks. However, this difference is disappearing due to the arrival of TCP/IP stack offload cards. In the past, NAS systems were not as scalable as SAN systems due to the absence of clustered NAS solutions but with the advent of clustered NAS solutions, this limitation has been also overcome.
- *NAS Protocol Analysis:* NFS and CIFS are the two primary NAS protocols. NAS protocols are between NAS clients that reside on the application/database server, and a NAS server that typically is a separate box that contains a file system and manages disks. NAS clients provide a file system interface to the host applications. NFS is used primarily in the Unix/Linux environment and it is an IETF standard, whereas, CIFS is used primarily in the Windows OS environment and its management/administrative APIs are proprietary (controlled by Microsoft). Samba is a popular protocol bundle that implements many different CIFS related protocols. A Samba server can act as an open-sourced CIFS server on Unix systems that makes Unix directories appear as Windows folders to Windows clients. With the emergence of NFSv4, many of the NFS deficiencies with respect to recovery management, caching, and security have been overcome and makes NFS a competitive protocol. NFSv4 has consolidated numerous protocols such as nfs, nlm, mountd, and nsm. NFSv4 is a stateful protocol, and it introduces the concept of delegation to allow for aggressive data caching at the clients [2].

Key Applications

SANs have been typically used by applications that want block level access to storage and are performance conscious. NAS solutions have been typically used by users who are more cost conscious and want to leverage their existing IP infra-structure and IP network management experience. In SANs, Fiber Channel and iSCSI protocols are typically used to connect application servers to the storage controller boxes. Fiber Channel, SATA and SAS protocols are typically used to connect the storage controller processor/cache complex to its backend arrays/disks. In DAS environments, the server is connected to its backend storage using Parallel SCSI, ATA, SATA or SAS. In NAS environments, NFS/CIFS are used to connect the application servers to the NAS server, and SATA, Fiber Channel or SAS is used to connect the NAS processor complex to the back-end disk arrays.

Cross-references

▶ Direct Attached Storage
▶ Network Attached Storage
▶ SAN File System
▶ Storage Area Network

Recommended Reading

1. Kaladhar V. and Prasenjit S. An analysis of three gigabit networking protocols for storage area networks. In Proc. 20th IEEE Int. Performance, Computing and Communications Conf., 2001.
2. Peter R., Li Y., Pawan G., Prasenjit S., and Prashant S. A performance comparison of NFS and iSCSI for IP-networked storage. In Proc. 3rd USENIX Conf. on File and Storage Technologies, 2004.

Storage Resource Management

Hiroshi Yoshida
Fujitsu Limited, Yokohama, Japan

Synonyms

SRM

Definition

The management of physical and logical storage resources, including storage elements, storage devices, appliances, virtual devices, disk volume and file resources. In most cases, storage resource management is achieved by software tools which indicate and manage the storage resource utilization in a storage networking environment.

Key Points

When multiple storage devices are connected to a SAN and shared by multiple servers, the space in each device cannot be used uniformly, e.g., one device may be almost full while another is underutilized. If the storage administrator is unaware of the correct space utilization status, inefficient resource utilization of the whole SAN environment can occur. This may cause unnecessary addition of new storage resources, or additional resources are not installed in time causing business applications to stop due to lack of required storage space.

To indicate and manage storage resource utilization, storage resource management software tools provide the following functions:

- Discovery and investigation of the utilization of all storage resources in a SAN, plus related integrated monitoring and integrated management operations.
- Analysis and reporting of storage resource utilization
- Analysis and estimation of movements in resource utilization, issuing alerts and/or execution of scripts for automation

Monitoring and indicating storage resource utilization from the business application viewpoint is an important requirement for storage resource management. To meet this requirement, storage resource management and storage configuration management, including dependency mapping between applications and storage devices, are often provided in combination.

Cross-references

▶ ILM
▶ LUN
▶ SAN
▶ Storage Consolidation
▶ Storage Management
▶ Storage Network Architectures
▶ Storage Virtualization

Recommended Reading

1. Storage Network Industry Association. Storage Network Industry Association tutorials, 2007. Available at: http://www.snia.org/education/tutorials/

Storage Security

Kaladhar Voruganti
Network Appliance, Sunnyvale, CA, USA

Synonyms

CAS; Authentication; Access control; WORM; Compliance; On-wire security; On-Disk security; Data corruption; Zoning; LUN masking; Port binding; Provenance; Watermarking

Definition

The definition of storage security has many facets, and some of the key requirements are:

Storage (and the appropriate manipulation capabilities) should only be accessible and visible to users with the appropriate permissions

Users should be notified if their data has been tampered with or altered either intentionally or unintentionally

Malicious users should not be allowed to access or tamper with other people's storage. If possible, the system should be able to catch malicious users.

User's data should be physically deleted at the appropriate time. That is, it should not be present past its intended life-time

Tracking unauthorized copying or access control delegations is also an important security concern.

Historical Background

Security has been a key computer science topic for many decades. Initially, people focused on authentication, encryption and access control for standalone systems. As distributed computing became popular, people started focusing on security within the context of network protocols. Now, with the emergence of utility computing, grid computing and cloud computing, storage is being accessed remotely. Thus, security issues pertaining to storage environment is now gaining importance. Many of the known security algorithms and techniques are now being re-used within the context of storage systems.

Foundations

The following different security mechanisms together provide the appropriate desired security functionality:

Storage Access Authentication: In order to prevent unauthorized users from accessing a storage device, there is usually an authentication mechanism employed (e.g., Kerberos, CHAP) to prevent unauthorized users from accessing the storage device. If the storage device is being accessed in a client-server manner, where the client resides on a database or application server, the client-server protocol needs to employ an authentication mechanism (e.g., Kerberos, or CHAP). In cases where both the host servers and the storage devices reside behind the firewall, client authentication is typically not performed.

On-Wire Security: When information flows between clients and storage servers outside the fire-wall protected domains, information is typically encrypted and sent over the wire to prevent eavesdropping. Data are also protected by a hash key to detect data tampering. IPSec security infra-structure provides authentication, data encryption and tampering detection support at the IP layer for internet environments. Data are typically not encrypted on the wire in intranet environments behind the firewall.

On-Disk Encryption: Data stored on persistent storage (disks, tapes etc) is typically not encrypted because disks usually reside in secure places. However, with the emergence of storage service providers, an organization's data are managed by others. Thus, many organizations prefer to encrypt their data before storing it at a remote location. It is very important to store and remember the keys that have been used to encrypt the data because loss of the key is as good as the loss of the data. Organizations also periodically re-encrypt data to prevent the use of brute-force approaches to break the keys being used to encrypt the data. AES, DES and 3DES are some of the encryption algorithms that are being currently used.

Access-Control: Access control mechanisms have been in place and are used extensively in operating systems to control access to various resource types. Access control typically amounts to controlling read/write access for individual users, groups and everyone to files, directories of files, volumes, and storage devices. Access control mechanisms also deal with delegation of control by secondary data owners to third parties who want to access the data.

Compliance: Organizations are being expected to adhere to many different types of government regulations such as:

- There should be only a single copy of a particular type of data.
- Data should be physically erased from the storage medium after so many years. That is, data cannot be logically deleted, and one should not be able to reconstruct the data from the physically deleted copy.
- Organization should be able to provide a guarantee that data has not been altered since its creation. Many organizations employ WORM (write-once, read-many) type of storage medium such as optical storage or WORM tapes or CAS systems to attain this capability.
- During audits data belonging to a certain topic should be available within a short specified period of time (for example, data should be available within 48 h). In some cases this precludes the storing of data on tape.

Data Corruption: Data can be corrupted either unintentionally due to device failure or malfunction, or it could be corrupted intentionally by a malicious user or a virus. In either case, data corruption has to be detected and subsequently corrected. Data corruption can be detected at the time of data creation, or it can be checked for by a periodic checking process such as virus scans or disk scrubbing. Data corruption can dealt with in the following ways:

- *Detection:* Data corruption can be detected (not corrected) using a hash function such as MD5 or SHA-1, checksum or CRC.
- *Detection and Correction:* Data corruption can be detected and corrected using error correction codes such as Reed-Solomon.
- *Correction:* Data corruption can be only corrected using erasure correcting codes such as a variant of Reed-Solomon codes.

Zoning: In storage area networks, zoning is a technique that is used to control which host port can have access to which storage device port. Zoning can be controlled at the switch firmware level (called hard zoning), by which one controls which switch input port can see which switch output port. Zoning can also be controlled at the Fiber Channel name service level (soft zoning), by which one controls which host server port can see which storage device port.

Port Binding: Port binding is the process by which one controls which port can be connected to a particular switch port. That is, one explicitly specifies the address of the host or storage device port that can exclusively transfer data to a particular switch port.

LUN Masking: The process of determining which host port can access which volume on the target via which target port is called as LUN masking. Thus, using this process one can control the access of storage volumes by the different hosts.

Provenance and Watermarking: With the emergence of peer to peer networks, data are getting copied at a very rapid rate often without the proper permission. In many cases, parts of the original document get copied and combined with new content. Thus, it is necessary to keep track of the trail of how data has been copied and modified from its source to its current location. In addition to keeping track of the original author and source of the data, it is also necessary to detect pirated copies of data. New digital document water-marking techniques are being developed to add hidden copyright notices to documents.

CAS: Content addressable storage (CAS) systems provide a new type of data access mechanism. In CAS systems one generates a hash value out of the data content. This hash value is subsequently used to uniquely access the data. In CAS systems, one typically does not overwrite existing data but instead a new copy of the data gets created when updates are made to existing data copies. CAS systems have been used to provide WORM media capabilities for disk based systems.

Key Applications

Until recently, storage systems have been based behind company fire-walls. Thus, authentication and encryption have not been a major issue. Access control and data corruption (due to viruses or device failures) have been the major forms of security/integrity checking processes. With the emergence of government compliance regulations, data compliance has become a very major issue for most organizations. With the emergence of third party archival or storage service provider paradigms, more importance is being given to storage authentication and encryption mechanisms.

Cross-references

▶ Access Control Administration Policies
▶ Access Control Policy Languages
▶ Asymmetric Encryption
▶ Authentication
▶ Data Encryption
▶ Database Security
▶ Discretionary Access Control
▶ Homomorphic Encryption
▶ Mandatory Access Control
▶ Message Authentication Codes
▶ Network Attached Secure Device
▶ Security Datawarehouses
▶ Security Services
▶ Symmetric Encryption
▶ Temporal Access Control
▶ XML Security

Recommended Reading

1. Riedel E., Kallahalla M., and Swaminathan R. A framework for evaluating storage system security. In Proc. 1st USENIX Conf. on File and Storage Technologies, 2002.

Storage Servers

► Storage Devices

Storage Systems

► Performance Analysis of Transaction Processing Systems

Storage Virtualization

Hiroshi Yoshida
Fujitsu Limited, Yokohama, Japan

Definition

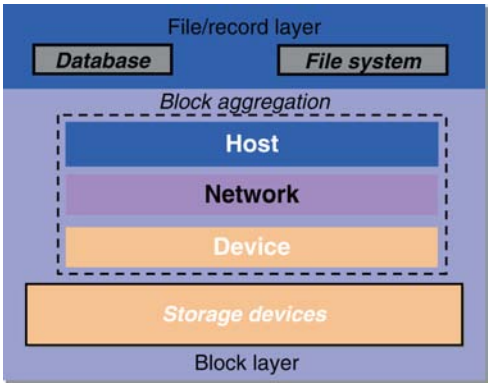
Storage virtualization is technology to build logical storage using physical storage devices. More precisely, SNIA defines storage virtualization as follows:

1. The act of abstracting, hiding, or isolating the internal function of a storage (sub) system or service from applications, compute servers or general network resources for the purpose of enabling application and network independent management of storage or data.
2. The application of virtualization to storage services or devices for the purpose of aggregating, hiding complexity or adding new capabilities to lower level storage resources.
3. Storage can be virtualized simultaneously in multiple layers of a system, for instance to create hierarchical storage manager like systems.

Key Points

Storage virtualization is used to provide an aggregation of data blocks (a logical volume) to applications running on servers. This aggregation of blocks may reside on a single storage device, may be spanned across multiple devices when it is too large to fit on a single device, may be mirrored to multiple storage devices to increase availability, or may be striped across multiple storage devices to enable parallel accesses.

Storage virtualization (block aggregation) can be implemented in various layers between the host server and the storage devices, i.e., in the host layer (software-based logical volume manager), in the network layer (switch-based virtualization appliance), and in the device layer (disk array). Figure 1 shows where block aggregation is implemented in the shared storage model defined by Storage Networking Association.



Storage Virtualization. Figure 1. Block aggregation in shared storage model by SNIA.

Storage virtualization technologies are now being extended to provide more capabilities. For example, multiple storage devices which reside in different locations can be virtualized as a single logical storage device with disaster recovery capability. Another example is the virtualization of inexpensive storage devices with high capacity together with expensive devices with high performance to form a single logical device where migration between the physical devices is done automatically. This provides a cost-effective storage system suitable for hierarchical storage management or information lifecycle management (ILM).

Cross-references

► ILM
► SAN
► SRM
► Storage Consolidation
► Storage Network Architectures
► Storage Networking Industry Association

Recommended Reading

1. Storage Network Industry Association. Storage Virtualization: the SNIA technical tutorials, 2007. Available at: http://www.snia.org/education/storage_networking_primer/stor_virt/
2. Storage Network Industry Association. The SNIA shared storage model, 2007. Available at: http://www.snia.org/education/storage_networking_primer/shared_storage_model/

Stored Procedure

Tore Risch
Uppsala University, Uppsala, Sweden

Definition

Modern relational query languages such as SQL provide general programming language capabilities in addition to the statements for searching and updating the database. A stored procedure is a user program written in a query language running inside the database server. Stored procedures often include side effects that update the database. This makes it possible to define general programs using the query language. These programs are called stored procedures and are executed inside the database server.

Key Points

In SQL, the user defines stored procedures as a schema manipulation statement using as CREATE PROCEDURE statement. A so defined procedure is immediately shipped to the database server where it is compiled and stored. Stored procedures can have side effects that change the state of the database. In order to prohibit searches that change the state of the database, in SQL stored procedures cannot be called from within queries, but only from applications or general SQL interfaces. SQL provides a special EXEC statement passing to the procedure when it is called.

The advantage with stored procedures are:

- Communication time is saved in case the communication with the application program is slow or unreliable.
- Common database centered updates and computations belong naturally to the database.

A disadvantage with stored procedures is that different vendors often provide different stored procedure languages. The SQL-PSM (Persistent Stored Modules) standard defines stored procedures in SQL.

Related to stored procedures (and part of SQL-PSM) are user defined functions (UDFs), which are user defined functions to perform common side-effect free computations. UDFs are allowed as expressions in queries.

Cross-references

▶ Query Language

S-Transactions

▶ Flex Transactions

Stream Data Analysis

▶ Stream Mining

Stream Mining

Jiawei Han, Bolin Ding
University of Illinois at Urbana-Champaign,
Champaign, IL, USA

Synonyms

Stream data analysis

Definition

Stream mining is the process of discovering knowledge or patterns from continuous data streams. Unlike traditional data sets, *data streams* consist of sequences of data instances that flow in and out of a system *continuously* and with varying update rates. They are *temporally ordered, fast changing, massive, and potentially infinite*. Examples of data streams include data generated by communication networks, Internet traffic, online stock or business transactions, electric power grids, industry production processes, scientific and engineering experiments, and video, audio or remote sensing data from cameras, satellites, and sensor networks. Since it is usually impossible to store an entire data stream, or to scan through it multiple times due to its tremendous volume, most stream mining algorithms are confined to reading only once or a small number of times using limited computing and storage capabilities. Moreover, much of stream data resides at a rather low level of abstraction, whereas analysts are often interested in relatively high-level dynamic changes, such as trends and deviations. Therefore, it is essential to develop online, multilevel, multidimensional stream mining methods. Stream mining can be considered a subfield of data mining, machine learning, and knowledge discovery.

Historical Background

There are extensive studies on stream data management and the processing of continuous queries in stream data [4]. Different from stream query processing, stream mining extracts patterns and knowledge from online stream data. It covers the topics of mining multidimensional stream statistics, frequent patterns, classification models, clusters, and outliers in online data streams. Substantial research on stream mining has appeared since only 2000, almost at the same time as the research on stream data management. However, lots of results have been generated in this line of research.

Foundations

Stream mining problems are challenging because of the following two reasons. (i) Stream data are massive, arriving with high speed, and updated frequently, so that one can neither store all the data nor scan the data repeatedly, and in the meantime, the response time is usually required to be short in applications; therefore, stream synopsis construction is popularly used to maintain a summary of stream data online using limited space without losing too much information. (ii) Stream data often evolves considerably over time, and for classification or clustering, one should often use biased sampling of the stream data to emphasize more recent behavior of the stream. In general, stream mining can be partitioned into four themes: (i) online computing multidimensional stream statistics, (ii) mining frequent items and itemsets over stream data, (iii) stream data classification, and (iv) clustering data streams.

For online computing multidimensional stream statistics, a multidimensional stream cube model was proposed by Chen et al. [6] in their study of multidimensional regression analysis of time-series data streams. A stream cube can be efficiently constructed based on (i) a tilted timeframe, where the finer granularity is used for more recent time, and the coarser granularity for more distant time, (ii) a minimal interest layer to register the minimal layer of the cube that is still of user's interest, and an observation layer for the cuboid that a user usually watches for trends or anomaly, and (iii) partial materialization that materializes the cuboids only along the popular drilling path, serving as a tradeoff between the storage space and online response time. Statstream, a statistical method for the monitoring of thousands of data streams in real time, was developed by Shasha and Zhu [15].

For mining frequent items and itemsets over stream data, one important issue is how to return approximate frequency counts for items or itemsets with limited buffer size for infinite data streams. Manku and Motwani [12] proposed Sticky Sampling algorithm and Lossy Counting algorithm for computing approximate frequency counts of items over data streams, and developed a Lossy Counting based algorithm for computing frequency counts of itemsets with the focus on system-level issues and implementation artifices. Another important issue is how to track frequent items dynamically. "Dynamically" means data streams consist of both "insertion" operations and "deletion" operations of items (imagine cars entering and exiting the parking lot). Cormode and Muthukrishnan [7] proposed algorithms for this problem using group testing and randomization techniques. Keeping track of frequent of items in such data streams arises in applications of both traditional databases and other domains, like telecommunication networks.

For stream data classification, the goal is to predict the class label or the value of new instances in the data stream, given some knowledge about the class membership or values of previous instances in the data stream. Since the distribution underlying the instances or the rules underlying their labeling may change over time, the class label or the target value to be predicted may change over time as well (stream data is evolving). This problem is referred to as *concept drift*. A major challenge in stream classification is how to construct highly accurate models with the existence of concept drift in stream data. Hulten et al. [10] developed an algorithm CVFDT, by integrating concept drift in time-changing data streams and a statistical measure, Hoeffding bound. Wang et al. [16] proposed an ensemble classifier to mine concept-drifting data streams. Aggarwal et al. [3] developed a *k*-nearest neighbor-based method for classify evolving data streams. Gao et al. [8] handled skewed distributions and proposed an ensemble-based framework that under-samples the overwhelming negative data, and repeatedly samples the scarce positive data for model construction with concept-drifting data streams.

For clustering data streams, in some applications, simply assume that the clusters are to be computed over the entire data stream, and view the stream clustering problem as a variant of single-scan clustering algorithms. For example, Guha et al. [9] gave space-efficient constant-factor approximation algorithms for the k-median problem in stream data using divide-and-conquer and randomization techniques; O'Callaghan et al. [14] proposed a *k*-median based stream clustering method by incrementally updating *k*-median centers. However, it is important to consider stream evolution over time besides the single-scan constraint and resource limitation. When stream data is evolving, the underlying clusters may also change considerably over time. If the entire data stream is used for clustering, the result is likely to be inaccurate. What's more, at one moment, users may wish to examine clusters occurring in different time periods (e.g., last week/month/year). Aggarwal et al. [2] proposed a *CluStream* framework for clustering evolving data streams by introducing a tilted time-frame, an online microclustering maintenance, and an offline query-based macro-clustering mechanism, to achieve efficiency and high clustering quality and to provide the flexibility to compute clusters over user-defined time periods in an interactive fashion.

Aggarwal [1] provides a comprehensive survey on stream data processing and stream mining with a collection of chapters on difference issues on stream mining.

Key Applications

There are broad applications of stream mining, of which only a few examples are illustrated.

Mining anomaly in network streams. Stream mining has been popularly used for mining anomaly in computer network or other stream data. For example, MAIDS (Mining Alarming Incidents in Data Streams) [5] is a system that explores tilted timeframe and stream cube for mining computer network anomaly.

Computing statistical measures over time series data streams. Another popular application is to compute statistical measures over time series streams, such as StatStream [15].

Integration of mobile computing with stream mining. Kargupta et al. [11] have developed VEDAS (Vehicle Data Stream Mining System) that allows continuous monitoring and pattern extraction from data streams generated onboard a moving vehicle.

Future Directions

There are many challenging issues to be researched further. Only a few are listed below.

Mining sophisticated patterns in data streams. Due to the single-scan constraint and resource limitation, the current stream pattern mining methods are confined to simple patterns, such as single items or some limited itemsets. Tasks for mining sequential patterns [13] and structured patterns are challenging but interesting for research.

Mining sophisticated data sets for advanced applications. Text (e.g., document) streams and video streams are important real-life stream mining applications. However, it is challenging to mine such streams because it requires both sophisticated text/video analysis and real-time resource constraints. Other advanced stream mining applications include spatial data streams, financial transaction data streams, and so on.

Experimental Results

There are many experimental results reported in numerous conference proceedings and journals.

Data Sets

UCI Machine Learning Repository: http://archive.ics.uci.edu/ml/datasets.html

URL to Code

RapidMiner (previously called YALE (Yet Another Learning Environment)), at http://rapid-i.com, is a free open-source software for knowledge discovery, data mining, and machine learning. It also features data stream mining, learning time-varying concepts, and tracking drifting concept.

MassDAL (Massive Data Analysis Lab) Public Code Bank, at http://www.cs.rutgers.edu/muthu/massdal-code-index.html, is a library of routines in C and Java for stream data and other massive data set analysis, including implementations of some published algorithms for finding frequent items over stream data.

Cross-references

- ▶ Association Rule Mining on Streams
- ▶ Classification in Streams
- ▶ Clustering on Streams
- ▶ Data Mining
- ▶ Event Stream
- ▶ Frequent Items on Streams

- ▶ Geometric Stream Mining
- ▶ Stream Similarity Mining
- ▶ Wavelets on Streams

Recommended Reading

1. Aggarwal C.C. Data Streams: Models and Algorithms. Kluwer Academic, 2006.
2. Aggarwal C.C., Han J., Wang J., and Yu P.S. A framework for clustering evolving data streams. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 81–92.
3. Aggarwal C.C., Han J., Wang J., and Yu P.S. On demand classification of data streams. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 503–508.
4. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and issues in data stream systems. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 1–16.
5. Cai Y.D., Clutter D., Pape G., Han J., Welge M., and Auvil L. MAIDS: Mining alarming incidents from data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 919–920.
6. Chen Y., Dong G., Han J., Wah B.W., and Wang J. Multi-dimensional regression analysis of time-series data streams. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 323–334.
7. Cormode G. and Muthukrishnan S. What's hot and what's not : tracking most frequent items dynamically. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2003, pp. 296–306.
8. Gao J., Fan W., Han J., and Yu P.S. A general framework for mining concept-drifting data streams with skewed distributions. In Proc. SIAM International Conference on Data Mining, 2007.
9. Guha S., Mishra N., Motwani R., and O'Callaghan L. Clustering data streams. In Proc. 41st Annual Symp. on Foundations of Computer Science, 2000, pp. 359–366.
10. Hulten G., Spencer L., and Domingos P. Mining time-changing data streams. In Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2001.
11. Kargupta H., Bhargava B., Liu K., Powers M., Blair P., Bushra S., Dull J., Sarkar K., Klein M., Vasa M., and Handy D. VEDAS: A mobile and distributed data stream mining system for real-time vehicle monitoring. In Proc. SIAM International Conference on Data Mining, 2004.
12. Manku G. and Motwani R. Approximate frequency counts over data streams. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 346–357.
13. Mendes L., Ding B., and Han J. Stream sequential pattern mining with precise error bounds. In Proc. 2008 IEEE Int. Conf. on Data Mining, 2008.
14. O'Callaghan L., Meyerson A., Motwani R., Mishra N., and Guha S. Streaming-data algorithms for high-quality clustering. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 685–696.
15. Shasha D. and Zhu Y. High Performance Discovery In Time Series : Techniques and Case Studies. Springer, 2004.
16. Wang H., Fan W., Yu P.S., and Han J. Mining concept-drifting data streams using ensemble classifiers. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 226–235.

Stream Models

Lukasz Golab
AT&T Labs-Research, Florham Park, NJ, USA

Definition

Conceptually, a *data stream* is a sequence of data items that collectively describe one or more underlying signals. For instance, a network traffic stream describes the type and volume of data transmitted among nodes in the network; one possible signal is a mapping between pairs of source and destination IP addresses to the number of bytes transmitted from the given source to the given destination. A stream model explains how to reconstruct the underlying signals from individual stream items. Thus, understanding the model is a prerequisite for *stream processing* and *stream mining*. In particular, the computational complexity of a data stream problem often depends on the complexity of the model that describes the input.

Historical Background

The stream models discussed in this article were introduced in [3] and extended in [7,8]. In addition to modeling a stream with respect to its underlying signal(s), there exist the following two related concepts. First, the stream computational model asserts that a stream algorithm must run in limited space and time, and can only make a small number of passes (often only one pass) over the data [8,6]. Furthermore, one can also model various statistical properties of a data stream, such as changes in the frequency distribution of the underlying signals or inter-arrival times of stream items (see, e.g., [9]).

Foundations

Basic Model Definitions

Consider a *data stream* S composed of individual items s_1,s_2,..., ordered by arrival time. Let A be a signal described by S. Assume that A is a function from a discrete and ordered domain to the range of reals; i.e., $A : [1...N] \rightarrow \mathbb{R}$. For instance, in the motivating example from above, the domain consists of IP address pairs. That is, $N = 2^{64}$ since an IP address is 32 bits long.

There are four models for representing A using individual stream items:

1. In the aggregate model, each stream item s_i corresponds to a range value for some domain value.
2. In the cash register model, each stream item s_i represents a domain value and a partial range value r_i, such that $r_i \geq 0$. Reconstructing the signal A involves aggregating all the r_i values corresponding to each domain value.
3. The turnstile model generalizes the cash register model by allowing any r_i to be negative. Thus, reconstructing the signal A involves adding/subtracting the contributions of stream items having positive/negative range values.
4. In the reset model, each stream item s_i corresponds to a range value and is understood to replace all previously reported range values for the given domain value.

Each of the four models defined above has an ordered and an unordered version. In the ordered version, stream items arrive over time in increasing order of the domain values. In the unordered version, the ordering of the domain does not correspond to the arrival order of stream items.

In terms of complexity and expressive power, the turnstile model is the most general, followed by the cash register model and the aggregate model, respectively. As a result, designing stream algorithms for the turnstile model is the most challenging. For instance, while many types of sketches have provable time and accuracy bounds in the turnstile and cash register models, *stream sampling* algorithms are typically applicable only in the cash register model [8] (effectively, the turnstile model allows deletions via negative range values, therefore it may not be possible to maintain a fixed-size sample of the stream over time). Additionally, geometric problems over streams (e.g., estimating the diameter of a stream of points) are difficult to solve in the turnstile model since previously seen points may be deleted in the future [8]. The reset model is also quite general and some algorithms in this model are more complex than in the other three models [8]. See [3,7,8] for examples of algorithms and complexity bounds for different stream models.

Examples and Extensions of Basic Models

Consider a network traffic stream S composed of IP packets. Each packet contains (among other things) the source IP address, destination IP address, and size. Define signal A_1 as a function from the source and destination address pairs to the total number of bytes exchanged by each pair (i.e., sums of sizes of all the packets sent between a given pair). Since many packets may be exchanged between two nodes and packets may arrive in random order, this example corresponds to the unordered cash register model.

For an example of an ordered cash register model, define S_2 to be the output stream of a pipelined query plan for the following query:

```
SELECT a1, a2, count(*)
FROM T
GROUP BY a1, a2
ORDER BY a1, a2
```

Define signal A_2 as a function from values of a_1 to the corresponding frequency counts (aggregated over all possible values of a_2). This example corresponds to the ordered cash register model – stream items arrive in the order of their domain values due to the `ORDER BY` clause, but must be aggregated on a_1 in order to reconstruct A_2.

Now suppose that the output stream of the above query is not ordered by a_1, but has the property that all the groups having the same value of a_1 are streamed out contiguously. This conforms to the contiguous unordered cash register model. Note that the ordered cash register model is always contiguous.

Next, suppose that stream S_3 is a pre-processed version of S, where each item s_i is a triple of the form: (source IP address, destination IP address, event), where the event field denotes the start or end of a connection between two nodes. Define signal A_3 as a function from the source and destination address pairs to the total number of open connections between each pair that have not yet ended. This corresponds to the unordered turnstile model since a stream item carrying an end-of-connection event decrements the total count of open connections for the given pair of nodes; furthermore nodes may open and close connections in arbitrary order.

In the above examples, the range of the signal corresponds to non-negative integers. A turnstile model whose signal has a non-negative range is said to be a strict turnstile model. For an example of a non-strict turnstile model, consider tracking the difference between the number of connections originating from two different IP addresses. Note that some sketch-based algorithms that work in the strict turnstile model do not apply in the non-strict version [8].

As in the cash register model, one may define a contiguous unordered (strict or non-strict) turnstile model; the ordered (strict or non-strict) turnstile model is always contiguous.

For an example of an aggregate model, suppose that stream S_4 is a pre-processed version of S, where each stream item denotes the total number of bytes exchanged between a given source-destination pair over a 5 min. *window*. Define signal A_4 as a function from the source and destination address pairs to the total number of bytes exchanged by each pair in the window. This gives rise to an ordered aggregate model if stream items are ordered by the source and destination addresses, and an unordered aggregate model otherwise. Note that S_4 may contain a concatenation of many instances of A_4, each corresponding to range values calculated over a particular 5 min. *window*.

Alternatively, one can model S_4 as carrying a single signal over time, call it A_5, whose domain is the same as that of A_4, but whose range is total number of bytes exchanged by each pair in the most recent 5 min. interval. This corresponds to the (unordered and non-contiguous) reset model because new items arriving on the stream (i.e., those corresponding to the most recent 5 min. window) replace old items having the same domain value.

Key Applications

The unordered cash register model is appropriate for applications where the incoming stream contains multiplexed data feeds from many sources (e.g., network traffic). However, the turnstile model must be used if the input (which may be a pre-processed version of a stream originally conforming to the cash register model) includes positive and negative range values. In particular, the turnstile model can represent a signal whose range values are computed over sliding *windows*. To see this, note that values that expire from the window may be modeled as new stream items with negative range values. Stream items whose purpose is to invalidate previously arrived items are often referred to as negative tuples [1,2,5].

On the other hand, the aggregate model is appropriate for many types of time series data, e.g., aggregated network traffic data generated every 5 min. In some cases, the reset model may also be suitable. Moreover, the reset model is useful in applications that process locations of moving objects over time. In these applications, moving objects (e.g., a fleet of delivery trucks) periodically report their current positions [7].

Future Directions

The four stream model described in this article can express a wide range of application scenarios. However, new *streaming applications*, such as *publish-subscribe over streams*, *XML-stream processing*, signal-oriented stream processing [4], and analysis of the results of large-scale scientific experiments, may require new models.

Cross-references

▶ Data Stream
▶ Stream Mining
▶ Stream Processing

Recommended Reading

1. Arasu A., Babu S., and Widom J. The CQL continuous query language: semantic foundations and query execution. VLDB J., 15(2):121–142, 2006.
2. Ghanem T., Hammad M., Mokbel M., Aref W., and Elmagarmid A. Incremental evaluation of sliding-window queries over data streams. IEEE Trans. Knowl. and Data Eng., 19(1):57–72, 2007.
3. Gilbert A., Kotidis Y., Muthukrishnan S., and Strauss M. Surfing wavelets on streams: one-pass summaries for approximate aggregate queries. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 79–88.
4. Girod L., Mei Y., Newton R., Rost S., Thiagarajan A., Balakrishnan H. and Madden S. The case for a signal-oriented data stream management system. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 397–406.
5. Golab L. and Özsu M.T. Update-pattern aware modeling and processing of continuous queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 658–669.
6. Henzinger M., Raghavan P., and Rajagopalan S. Computing on data streams. DIMACS Ser. Discrete Math. Theor. Comput. Sci., 50:107–118, 1999.
7. Hoffmann M., Muthukrishnan S., and Raman R. Streaming algorithms for data in motion. ESCAPE. Springer, Berlin Hiedelberg New York, 2007, pp. 294–304.
8. Muthukrishnan S. Data streams: algorithms and applications. Found. Trends Theor. Comput. Sci., 1(2):1–67, 2005.
9. Paxson V. and Floyd S. Wide-area traffic: the failure of Poisson modeling. IEEE/ACM Trans. Netw., 3(3):226–244, 1995.

Stream Processing

Michael Stonebraker[1,2]
[1]Massachusetts Institute of Technology, Cambridge, MA, USA
[2]StreamBase Systems, Inc., Lexington, MA, USA

Synonyms

Complex event processing (CEP); Event stream processing (ESP); Data stream processing

Definition

Stream processing refers to a class of software systems that deals with processing streams of high volume messages with very low latency. It is distinguished from business activity monitoring (BAM) or business process monitoring (BPM), in that the client of a stream processing application is often a program, rather than a human. Hence, the volume and latency requirements are often much more stringent.

Currently, stream processing is widely used in computing real-time analytics in e-trading, maintaining the state of massively multi-player Internet games, real-time risk analysis, network monitoring, and national security applications. In the future, the declining cost of sensor technology will create new markets for this technology including congestion-based tolling on freeways and prevention of lost children at amusement parks.

Key Points

There are three main technical approaches to stream processing at the present time.

- *Custom code*. Traditionally, stream processing applications have been hand-coded in a low-level programming language such as C or C++. The current trend is toward using one of the other two technologies to achieve lower development and maintenance cost.
- *Stream-oriented SQL*. Recent research activity has extended SQL with primitives for real-time operation. The main additions are the notion of real-time windows, over which SQL aggregates can be computed, facilities to perform pattern matching on sequences of messages, and primitives to deal with out-of-order data. There are now high-performance Stream-oriented SQL engines from several vendors (e.g., Coral8, http://www.coral8.com and StreamBase, http://www.streambase.com).
- *Rule engines*. The final approach is to utilize a high-performance implementation of a rule engine for stream processing. These systems are descendents of the rule engines found in expert systems in the 1980's and originally specified by the Artificial Intelligence community in pioneering work in the 1970s. Such systems contain rich pattern matching capabilities, but must be extended with aggregation and windowing constructs. Currently, there are a variety of commercial rule engines addressing the stream processing market.

In cases where real-time processing must be combined with access to historical data, Stream-oriented SQL enjoys a natural advantage. Both real-time and historical analysis can be done in a single paradigm (SQL), whereas a rule engine must switch paradigms to access historical data. On the other hand, where very sophisticated pattern matching is the main requirement, rule engines enjoy an advantage, due to their richer pattern matching capabilities.

In either case, achieving high performance and low latency requires a collection of implementation optimizations. Extensive compilation, often to machine code, will lower message processing overhead. In addition, some systems go to great lengths to remove scheduling overhead, by pre-computing what operation must be performed next, and then directly calling that operation from the current one, thereby removing both scheduling and message queuing overhead. In addition, implementations based on storing real-time state in a DBMS (either a conventional disk-based one or a main memory DBMS) are not likely to be successful, because of the inherent overhead of these class of products.

Over the course of the next decade, it is expected that stream processing products will enter the mainstream, thereby complementing other system software components such as application servers and DBMSs.

Cross-references

▶ Data Stream
▶ Data Stream Management Architectures and Prototypes
▶ Streaming Applications
▶ Stream-Oriented Query Languages and Operators

Recommended Reading

1. Stonebraker M., Çetintemel U., and Zdonik S. The 8 requirements of real-time stream processing. ACM SIGMOD Rec., 34(4):420–447, 2005.

Stream Query Processing

► Window-based Query Processing

Stream Sampling

Bibudh Lahiri, Srikanta Tirthapura
Iowa State University, Ames, IA, USA

Definition

Stream sampling is the process of collecting a representative sample of the elements of a data stream. The sample is usually much smaller than the entire stream, but can be designed to retain many important characteristics of the stream, and can be used to estimate many important aggregates on the stream. Unlike sampling from a stored data set, stream sampling must be performed online, when the data arrives. Any element that is not stored within the sample is lost forever, and cannot be retrieved. This article discusses various methods of sampling from a data stream and applications of these methods.

Historical Background

An early algorithm to maintain a random sample of a data stream is the *reservoir sampling* algorithm due to Vitter [15]. More recent random sampling based algorithms have been inspired by the work of Alon et al. [1]. Random sampling has for a long time been used to process data within stored databases – the reader is referred to [13] for a survey.

Foundations

A powerful sampling technique is *random sampling*, where random elements of the stream are selected into the sample. A random sample of a stream can be used in deriving *approximate* answers to aggregate queries such as *quantiles* [12] or *frequent elements* [11]. It can also be used in the estimation of the *selectivity of a query predicate*, which is defined as the fraction of the data items in the stream which satisfy the given user predicate. The intuition behind the above applications of random sampling is as follows. Suppose R is A uniform random sample of a data stream S. For any $A \subset S$, the size of A, $|A|$ can be estimated as $|A \cap R| \cdot |S| / |R|$. The accuracy of this estimate depends on the value of $|A \cap R|/|R|$. There are tradeoffs between the quality of the answer returned, the confidence in the answer, and the space taken by the sample.

There are two basic ways of generating a random sample of any data set – sampling without replacement and sampling with replacement. Consider a data stream with N elements and a sample size n. In random sampling with replacement, each element of the sample is chosen at random from among all N elements of the data set. It is possible that the same element is chosen more than once into the sample (though this is unlikely if the sample is much smaller than the data). A random sample without replacement is a randomly chosen subset of n elements from among all $\binom{N}{n}$ subsets of size n, thus ensuring that a data element appears no more than once in the random sample.

Reservoir Sampling

This technique, due to Vitter [15], allows the maintenance of a random sample of the stream of a particular target size in an online fashion. Suppose the objective is to maintain a random sample of n elements without replacement, from a stream of N elements, where N is not known a priori. Let the stream elements be $a_1,a_2,\ldots,a_N$.

A discussion of how to maintain a sample of a single element from the stream i.e., the case $n = 1$, can be a good point to begin. When a_1 arrives, it is always selected into the sample. For $i \geq 2$, element a_i is selected into the sample with probability $1/i$, i.e, a_2 is selected with a probability $1/2$, a_3 is selected with a probability $1/3$, and so on. Each time an element is selected into the sample, it replaces the existing element in the sample. It can be verified that the final element in the sample has an equal probability of being any of the N elements in the stream.

The above idea can be extended to maintain a random sample without replacement of size n as follows. The first n elements of the stream are (deterministically) included in the sample. For $t \geq n$, when a_{t+1} arrives, it is included in the sample with probability $n/(t+1)$. If an element is selected for inclusion in the sample, it replaces an element that is chosen uniformly at random from the currently existing elements in the sample. It is easy to verify that the resulting sample is equally likely to be any of the $\binom{t+1}{n}$ subsets of size n of the set $a_1,a_2,\ldots,a_{t+1}$. This algorithm is described in Fig. 1.

Note that if one wanted to sample n elements from a stream *with replacement*, this could be achieved by running n copies of the single element reservoir sampling algorithm. Further enhancements are possible to the algorithm in Fig. 1. In particular, instead of examining every element of the stream to see if it will be sampled, it is possible to directly generate the number of elements of the stream to be *skipped* before the next element that will be included in the sample. This can significantly reduce the number of stream elements to be examined by the sampling algorithm. For further details, the reader is referred to [15].

Sample and Count

This is a technique pioneered by Alon et al. [1], and is based on random sampling followed by counting. This technique has been applied in the estimation of frequency dependent statistics on a data stream in very small space. To see its use in the context of estimating the frequency moments of a data stream, consider a stream $S = a_1,a_2,\ldots,a_N$, where each $a_i \in \{1,2,\ldots,m\}$. For $1 \leq j \leq m$, let f_j denote the number of occurrences of j in stream S. For integral $k \geq 0$, the kth frequency moment of S, denoted by F_k is defined as follows.

$$F_k = \sum_{j=1}^{m} f_j^k$$

For $k \neq 1$, computing F_k exactly on a large data stream S is provably expensive space-wise. There are lower bounds showing that such an exact computation of F_k, or even an accurate *deterministic* approximation of F_k requires $\Omega(m)$ space, in the worst case. However, a randomized approximation to F_k (for $k \geq 2$) can be found as follows. First, choose a random element a_p from S (this can be done without a knowledge of N using the reservoir sampling technique). Then maintain the count $X = |\{q : q \geq p, a_q = a_p\}|$. In other words, count the number of re-occurrences of the element a_p in the portion of the stream that succeeds a_p (including a_p). Then, the random variable $Y = N\,[X^k - (X-1)^k]$ is an unbiased estimator of F_k, i.e., $E[\,Y\,] = F_k$. Further, it can also be shown that the variance of Y is small. For user defined parameters $0 < \varepsilon, \delta < 1$, this can be used to generate an estimator of F_k that is within a relative error of ε with probability more than $1 - \delta$ using small space. For exact space bounds, proofs and details, the reader is referred to [1].

The sample and count technique has also been used in accurate estimation of another frequency dependent aggregate, the *empirical entropy* of a data stream, in limited space. Consider a stream of integers, $S = a_1,a_2,\ldots,a_N$, where each $a_i \in \{1,2,\ldots,m\}$. For $1 \leq j \leq m$, let f_j denote the number of occurrences of j in S. The empirical entropy of the stream is defined as $\sum{}^{m}{}_{j=1} - (f_j/N) \log (f_j/N)$. The entropy of a stream yields valuable information about the amount of "randomness" within the stream, and is useful in many contexts in network monitoring. Chakrabarti et al. [3] present an algorithm for estimating the entropy of a stream. Their algorithm uses the sample and count technique, and yields a provably accurate estimate of the entropy (a randomized approximate estimate) using nearly optimal space.

Distinct Sampling

In some cases, it may be necessary to compute aggregates over all the *distinct* elements in the stream.

Input: Stream $a_1, a_2, \ldots, a_N$ where N is not known in advance; Sample Size n
Output: A sample $S[1,\ldots,n]$ of n elements chosen uniformly at random without replacement from the stream.

```
1. For $i$ from 1 to $n$, $S[i] \leftarrow a_i$ // The first $n$ elements are included in the sample
2. $t \leftarrow n$ // $t$ is the number of records processed so far
3. While (there are more stream elements)
    (a) $t \leftarrow (t+1)$
    (b) Let $s$ be a random number in the set {1, 2, 3,...., t}
```

Stream Sampling. Figure 1. Reservoir sampling for sampling without replacement from a stream.

Consider a stream of tuples (i,v) where i is an item identifier and v is the value. In database query optimization, a question of interest is often just an estimate of the number of distinct values for an attribute in a relation. In network monitoring, an objective may be tracking all those sources that have contacted a large number of distinct destinations in the recent past. For computing such aggregates over all distinct identifiers, a uniform random sample will not be useful. For example, suppose a stream of 10^8 elements had 1,000 distinct identifiers, but every identifier appeared exactly once in the stream, except for the "dominant identifier", which made up the remaining $10^8 - 999$ elements of the stream. Even if a fairly large random sample of this stream is collected, say a sample of 10^4 elements, the sample is likely to contain only the dominant identifier, and any estimate from this sample is likely to be extremely inaccurate. To derive an useful sample for estimating aggregates over distinct elements in a stream, Gibbons and Tirthapura [9] introduced a technique called *distinct sampling*.

In distinct sampling, the sampling is performed with the help of a randomly chosen *hash function*, rather than through independent random choices for each element. The hash function h is chosen before the stream elements are observed. Given a target sample size n, the algorithm in Fig. 2 maintains a random sample of all distinct elements of a stream S of size approximately n. For example, suppose the stream of 10^8 elements had 7,500 distinct identifiers. If the target sample size was 10^4, then all distinct elements would be included in the sample. If the target sample size was 10^3, then each distinct element would be included in the sample with probability p, where p is reduced until the resulting sample fits within the target sample size.

Let the stream S be an integer sequence $a_1,a_2,..,a_N$. The algorithm maintains a sample D of distinct elements, and a sampling level ℓ. At level ℓ, each distinct element is selected into the sample with probability $1/2^\ell$. For simplicity, assume a random hash function h is available such that $h(x)$ is a random real number which is uniformly chosen from the range (0,1), and the outputs of the hash function on different inputs are mutually independent. The analysis of the algorithm using more practical hash functions with limited independence (and integral outputs) is presented in [9].

The above distinct sampling algorithm assumed that all elements in the stream have the same weight – in general, this may not be true. For example, a user might be interested in computing the mean of the values received over all distinct identifiers in the stream – in this case, different elements should be weighted differently. The above algorithm was extended to the more general weighted case by Pavan and Tirthapura [14], who designed a "range-sampling" algorithm that allowed the weighted sampling problem to be reduced to the unweighted case.

Time-Decayed Sampling

The reservoir sampling (distinct sampling) algorithm maintains a sample of all elements (distinct elements) from the start of time, and such a sample is useful in

```
Input: Stream $S = a_1, a_2, \ldots, a_N$ where $a_i \in \mathcal{I}$ and $N$ is not known in advance; Maximum sample
size $n$; Hash function $h : \mathcal{I} \to (0,1)$.
Output: $D$, a random sample of the distinct elements of the stream, chosen without replacement;
The size of $D$ is guaranteed not to exceed $n$.

  1. Initialization: $D \leftarrow \phi$, $\ell \leftarrow 0$
  2. While there are more elements in $S$
      (a) Let $a$ be the next element in $S$
      (b) If $(h(a) < \frac{1}{2^\ell})$
            i. If $(a \notin D)$ then Add $a$ to $D$
           ii. If $(|D| > n)$ then || Overflow, discard approximately half the items in D by selecting
               every item currently in D with a probability 1/2.
               A. $\ell \leftarrow \ell + 1$
               B. For every $a \in D$, if $h(a) \geq \frac{1}{2^\ell}$ then discard $a$ from $D$.
```

Stream Sampling. Figure 2. Algorithm for maintaining a sample of all distinct elements of a stream.

computing aggregates over the entire data stream. However, in many cases the interest lies in computing *time-decayed* aggregates of data, where older elements must be discounted. For such aggregates, it is useful to have a sample where a more recent stream element has a higher probability of being included in the sample. For example, a typical *sliding-window* aggregate asks for an aggregate over all data elements that have appeared in a window of the last *W* elements of a stream – to answer such queries, it is useful to have a random sample of all elements within the current window. The problem with directly using any of the above algorithms to maintain a sample over a sliding window is that elements in the sample may expire, i.e., fall out of the window, and replacing them with enough new sampled elements may not be possible, causing the number of elements within the sample to become too small.

The reservoir sampling algorithm was extended to sliding windows by Babcock et al. [2], who presented probabilistic guarantees on the space taken by their sampling algorithm. Gibbons and Tirthapura [10] have extended the distinct sampling algorithm to sliding windows, by sampling the stream at multiple probabilities, and maintaining a fixed number of the most recent elements at each sampling probability. It is known [4] that for computing decayed aggregates, an arbitrary time-decay function such as polynomial or exponential decay can be reduced to sliding window decay. Thus, for any stream aggregate that can be estimated well using random samples, such as quantiles, frequent elements, distinct counts and sum, the above techniques can be used to estimate time-decayed aggregates over an arbitrary decay function.

Handling Deletions

Thus far, it has been assumed that a data stream is a sequence of additions to a data set (which is so massive that it is too expensive to store it explicitly). More generally, it is necessary to deal with a stream where each element is an *operation* on the data set, which may be an addition or a deletion of one or more elements. On such a stream of add/delete operations, a direct sampling algorithm such as reservoir sampling or distinct sampling may not perform well, since elements that currently belong in the sample may be deleted by a future stream operation, leading to a sample that is too small to give useful estimates. The basic stream sampling algorithms have been extended to handle such "update streams" (sometimes called "dynamic data streams") by Ganguly [7], Cormode et al. [5], and Frahling et al. [6].

Key Applications

Estimating Query Selectivity: The selectivity of a query on a database table is defined as the ratio of the number of records that match the query to the total number of records in the table. Suppose that the records appeared as a stream, and it was not possible to store the entire table on the disk (or disk access was too expensive). Then, a stream sampling algorithm can be used to maintain a random sample of all the records. The selectivity of the query on the random sample is an unbiased estimate of the selectivity of the query on the whole stream (i.e., the expected value of the estimate equals the actual selectivity). Of course, the larger the random sample, the more accurate is the estimate. Note that the query can be posed *after* the stream was observed, since the random sampling procedure was not sensitive to the query.

Network Monitoring: In the context of monitoring a TCP/IP network, a "network flow" is a unidirectional set of packets that arrive at the router on the same subinterface, have the same source and destination IP addresses, same transport layer protocol (TCP/UDP), same TCP/UDP source and destination ports and the same type of service (ToS) byte in the IP headers. A network monitoring tool is a software that constantly monitors the flows in a network and helps in network management tasks such as load balancing and fault management. Some network monitoring tools, for example, Random Sampled Netflow (by Cisco), Gigascope (by AT&T Research), and "Smart Sampling" (by AT&T), provide data for a subset of traffic in a router by processing only a random sample of the packet stream. Traffic sampling substantially reduces consumption of router resources while providing valuable network flow statistics.

Sensor Data Aggregation: A sensor network is a network of resource-constrained embedded devices, capable of computing and sensing, deployed to monitor environmental conditions like temperature, sound, vibration, pressure, light, etc. In a typical scenario, sensors collect readings periodically and send them to a base-station or a local cluster-head where computation/aggregation takes place. Data within a sensor network can be viewed as the union of multiple distributed streams, one per sensor node. Random samples of such distributed data streams can be used in computing key aggregates of sensor data streams, such as the mean, quantiles, frequent elements, of data. Transmitting the random samples rather than the entire observation stream can lead to savings in communication cost and hence, energy.

Cross-references

▶ AMS Sketch
▶ Data Aggregation in Sensor Networks
▶ Data Sketch/Synopsis
▶ Distributed Streams
▶ Frequency Moments
▶ Randomization Methods to Ensure Data Privacy
▶ Stream Mining

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. J. Comput. Syst. Sci., 58(1):137–147, 1999.
2. Babcock B., Datar M., and Motwani R. Sampling from a moving window over streaming data. In Proc. ACM-SIAM Symp. on Discrete Algorithms, 2002, pp. 633–634.
3. Chakrabarti A., Cormode G., and McGregor A. A near-optimal algorithm for computing the entropy of a stream. In Proc. ACM-SIAM Symp. on Discrete Algorithms, 2007, pp. 328–335.
4. Cohen E. and Strauss M. Maintaining time-decaying stream aggregates. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2003, pp. 223–233.
5. Cormode G., Muthukrishnan S., and Rozenbaum I. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 25–36.
6. Frahling G., Indyk P., and Sohler C. Sampling in dynamic data streams and applications. In Proc. 21st Annual ACM Symp. on Computational Geometry, 2005, pp. 142–149.
7. Ganguly S. Counting distinct items over update streams. Theor. Comput. Sci., 378(3):211–222, 2007.
8. Gibbons P. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 541–550.
9. Gibbons P. and Tirthapura S. Estimating simple functions on the union of data streams. In Proc. ACM Symp. on Parallel Algorithms and Architectures, 2001, pp. 281–291.
10. Gibbons P. and Tirthapura S. Distributed streams algorithms for sliding windows. Theor. Comput. Syst., 37:457–478, 2004.
11. Manku G.S. and Motwani R. Approximate frequency counts over data streams. In Proc. of the 28th Int. Conf. on Very Large Data Bases, 2002, pp. 346–357.
12. Manku G.S., Rajagopalan S., and Lindsay B.G. Random sampling techniques for space efficient online computation of order statistics of large datasets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 251–262.
13. Olken F. and Rotem D. Random sampling from databases - a survey. Stat. Comput., 5(1):43–57, 1995.
14. Pavan A. and Tirthapura S. Range-efficient counting of distinct elements in a massive data stream. SIAM J. Comput., 37(2):359–379, 2007.
15. Vitter J.S. Random sampling with a reservoir. ACM Trans. Math. Softw., 11(1):37–57, 1985.

Stream Similarity Mining

Erik Vee
Yahoo! Research, Silicon Valley, CA, USA

Synonyms

Distance between streams; Datastream distance

Definition

In many applications, it is useful to think of a datastream as representing a vector or a point in space. Given two datastreams, along with a distance or similarity measure, the distance (or similarity) between the two streams is simply the distance (respectively, similarity) between the two points that the datastreams represent. Due to the enormous amount of data being processed, datastream algorithms are allowed just a single, sequential pass over the data; in some settings, the algorithm may take a few passes. The algorithm itself must use very little memory, typically polylogarithmic in the amount of data, but is allowed to return approximate answers.

There are two frequently used datastream models. In the *time series model*, a vector, $\vec{x}$, is simply represented as data items arriving in order of their indices: x_1,x_2, x_3,.... That is, the value of the ith item of the stream is precisely the value of the ith coordinate of the represented vector. In the *turnstile model*, each arriving item signals an update to some component of the represented vector. So item (i,a) indicates that the value of the ith component of the vector is increased by a. For this reason, datastream items are typically written in the form $(i,x_i^{(j)})$ to indicate that this is the jth update to the ith component of the represented vector. The value of x_i is then the sum of $x_i^{(1)} + x_i^{(2)} + ...$ over all such updates. The update values may be negative; the special case when they are restricted to be nonnegative is sometimes called the *cash register model*.

One of the most commonly used measures for datastream similarity is the L_p distance between two streams, for $p \geq 0$. As in the standard definition, the L_p distance between points $\vec{x}, \vec{y}$ (hence, between streams representing those points) is defined to be $\sum_i |x_i^p - y_i^p|^{1/p}$. In the case that $p = 0$, the L_0 distance (sometimes called the Hamming distance) is taken to be the number of i such that $x_i \neq y_i$. For $p = \infty$, the L_∞ distance is $\max_i |x_i - y_i|$. Other measures include the Jaccard similarity, the edit distance, the earth-mover's distance, and the length of the longest common subsequence between the streams (viewed as sequences).

Historical Background

Although the earliest datastream-style algorithms were discovered some 30 years ago [11], the current resurgence of interest in datastreams began with the seminal paper of Alon et al. [2] in 1996. Implicit in their work is an algorithm for estimating the L_2 distance between streams. In 1999, Feigenbaum et al. [10] developed a datastreaming algorithm to approximate the L_1 distance between two streams. Building on this, Indyk [12] gave datastreaming algorithms to approximate the L_p distance between two datastreams, for all $p \in (0,2]$, utilizing the idea of p-stable distributions. Later, Cormode et al. [7] demonstrated an efficient algorithm for approximating the L_0 distance (i.e., Hamming distance). Sun and Saks [15] provide lower bounds for approximating L_p, for $p > 2$ (and including $p = \infty$), showing no datastream algorithm working in polylogarithmic space can approximate the L_p distance between two streams within a polylogarithmic factor. (The bounds are even stronger for p much larger than 2.)

Datar et al. [8] studied the sliding window model for datastreams, producing an algorithm that approximates the L_p distance between two windowed datastreams. Work by Datar and Muthukrishnan [9] gave an algorithm for approximating the Jaccard similarity between two datastreams in the sliding window model.

Foundations

Estimating the L_2 Distance

In their seminal paper, Alon et al. [2] provide a method for estimating F_2, the second frequency moment, of a datastream. As observed in [10,1], this method can easily be extended to produce a datastream algorithm to approximate the L_2 distance. The ideas are briefly outlined below.

Throughout, the datastreams considered have length n. For $i = 1,2,\ldots,n$, the variable X_i is defined to be an i.i.d. (independent and identically distributed) random variable taking on the value -1 or 1 with equal probability. Of course, a datastream algorithm cannot maintain all the values of each of the random variables in memory. This will be accounted for later; for now, an algorithm is presented assuming that there is random access to these values.

The datastreams vectors are represented in the turnstile model; $(x_1,\ldots,x_n)$ denotes the accumulated values of the first stream, and $(y_1,\ldots,y_n)$ denotes the accumulated values in the second stream. The algorithm simply maintains the value of $\sum_{i=1}^n X_i \cdot (x_i - x_y)$. This value is straightforward to maintain: If an item $(i, x_i^{(j)})$ arrives for some i, j, the value $X_i \cdot x_i^{(j)}$ is added to it. If an item $(i, y_i^{(j)})$ arrives, the value $X_i \cdot y_i^{(j)}$ is subtracted.

The algorithm focuses on the expected value of *the square* of this quantity:

$$\begin{aligned} &E\left[\left(\sum_{i=1}^n X_i \cdot (x_i - y_i)\right)^2\right] \\ &= E\left[\sum_{i=1}^n X_i^2 \cdot (x_i - y_i)^2\right. \\ &\quad \left. + \sum_{i \neq j} X_i X_j \cdot (x_i - y_i)(x_j - y_j)\right] \\ &= \sum_{i=1}^n (x_i - y_i)^2, \end{aligned}$$

where the last equality follows since $E[X_i] = 0$ and $X_i^2 = 1$ for all i, and all the random variables are independent. But this quantity is just the square of the L_2 distance between the two streams. Hence, the problem amounts to obtaining a good estimate of this expected value.

To do so, the above algorithm is run in parallel k times, for $k = \theta(1/\varepsilon^2)$. That is, it maintains the value $\sum_{i=1}^n X_i \cdot (x_i - x_y)$ for k different random assignments of the X_i. The algorithm then takes the average of their squares. For a given run t, this value is denoted $v^{(t)}$. To further ensure that the algorithm does not obtain a spurious estimate, the procedure is repeated ℓ times, for $\ell = \theta(\log(1/\delta))$. The algorithm then takes the median value over $\{v^{(1)}, v^{(2)}, \ldots, v^{(\ell)}\}$. A standard application of Chebyshev's Inequality shows that this estimates the square of the L_2 distance within a $(1 + \varepsilon)$ factor with probability greater than $1 - \delta$. (In total, this method maintains $k\ell$ values in parallel.)

Unfortunately, the procedure as described above produces and maintains values for n random variables. (In fact, due to the parallel repetitions, it actually needs $k\ell n$ random variables.) However, the technique only needed these variables to be four-wise independent. (Two-wise independence is needed for the expected value to be an unbiased estimator of the square of the L_2 distance; four-wise independence implies that the variance is small.) Hence, these fully independent random variables can be replaced with four-wise independent random variables, which is necessary for Chebyshev's Inequality to hold. These random variables can be pseudorandomly generated on the fly; the datastream algorithm thus only needs to remember a logarithmic-length seed for the pseudorandomly generated values. The full details are omitted here.

Estimating the L_p Distance: *p*-Stable Distributions

In 2000, Indyk [12], using many of the ideas in [2,10], extended the results to produce datastream algorithms for approximating the L_p distance between streams, for all $p \in (0,2]$. (Feigenbaum et al. were the first to produce a datastream algorithm for L_1 distance; their technique relied on their construction of pseudorandomly generated "range-summable" variables that were four-wise independent. Although similar in flavor to the result of [2], it is somewhat more complicated.) For convenience, the algorithm outlined below details the method for approximating the L_p norm of a single vector. Note, however, that in the turnstile model, it is a simple matter to produce the L_p distance between two streams (by simply negating all of the values in the second stream and finding the norm of their union). Indyk's method uses random linear projections, and relies on the notion of *p-stable distributions*.

A distribution $\mathcal{D}$ is *p-stable* if for all k real numbers $a_1,\ldots,a_k$, if $X_1,\ldots,X_k$ are i.i.d random variables drawn from distribution $\mathcal{D}$, then the random variable $\sum\ _i a_i X_i$ has the same distribution as $(\sum\ _i |a_i|^p)^{1/p} X$ for random variable X with distribution $\mathcal{D}$. There are two well-known p-stable distributions. The *Cauchy distribution*, with density function $\mu_C(x) = \frac{1}{\pi}\frac{1}{1+x^2}$, is 1-stable. The *Gaussian distribution*, with density function $\mu_G(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, is 2-stable. Although closed-form functions are not known for p-stable distributions for $p \neq 1,2$, Chambers et al. [4] provide a method for generating p-stable random variables for all $p \in (0,2]$. Throughout the rest of this discussion, $\mathcal{D}$ denotes a p-stable distribution, for some fixed p.

The method for approximating the L_p norm of a stream will now be outlined. As previously noted, this is easily modified to give the L_p distance between two streams. Throughout, the vectors are represented as in the turnstile model, and $(z_1,\ldots,z_n)$ denotes the vector represented by the datastream. As in the previous section, the n i.i.d. random variables $X_1,\ldots,X_n$ are generated first, this time drawn from p-stable distribution $\mathcal{D}$. A brief discussion of how to reduce the number of these variables appears later.

The algorithm simply maintains the value $\sum\ _i X_i z_i$. Again, these values are easy to maintain: If item $(i, z_i^{(j)})$ appears for some i,j, the algorithm adds the value $X_i z_i^{(j)}$ to the sum. As in the previous section, the algorithm gains better accuracy by repeating the procedure multiple times in parallel; in this case, the algorithm runs the procedure k times in parallel, for $k = \theta(\frac{1}{\epsilon^2}\log(1/\delta))$. The value of $\sum\ _i X_i z_i$ obtained in the ℓ-th run using this procedure is denoted $Z^{(\ell)}$.

The value $Z^{(\ell)}$ is a random variable itself. Since $\mathcal{D}$ is p-stable, it is the case that $Z^{(\ell)} = X^{(\ell)} \cdot (\sum\ _i |z_i|^p)^{1/p}$ for some random variable $X^{(\ell)}$ drawn from $\mathcal{D}$. then the output of the algorithm is

$$\frac{1}{\gamma}\, median\{|Z^{(1)}|, \ldots, |Z^{(k)}|\},$$

where γ denotes the median value of $|X|$, for X a random variable distributed according to $\mathcal{D}$. (The absolute value is taken for technical reasons. For instance, the median value of X is 0 when $\mathcal{D}$ is the Gaussian distribution, while the median value of $|X|$ is strictly greater than 0.) The value of the median of $\{|Z^{(1)}|,\ldots,|Z^{(k)}|\}$ is $(\sum\ _i |z_i|^p)^{1/p}$ times the median of $\{|X^{(1)}|,\ldots,|X^{(k)}|\}$. Hence, the above output is an approximation of $(\sum\ _i |z_i|^p)^{1/p}$, i.e., the L_p norm of the datastream, as needed. A more careful argument shows that this estimate is within a multiplicative factor $(1 \pm \varepsilon)$ of the true L_p norm, with probability greater than $1 - \delta$.

As in the previous section, Indyk observes that rather than storing the values of n i.i.d random variables, the values can be generated on the fly, using pseudorandom generators. The details are omitted here.

Cormode et al. [7] investigate the problem of estimating the L_0 norm. One of their key technical observations is that the L_p norm is a good approximation of the L_0 norm of the stream, for p sufficiently small. (In particular, they show the $p = \varepsilon/\log M$ is sufficient, where M is the maximum absolute value of any item in the stream.) Thus, the Hamming distance between two streams can be approximated using the same general algorithm that was described above.

Approximating Jaccard Similarity: Min-Wise Hashing

Another useful similarity measure between two streams is their *Jaccard similarity.* Given two datastreams in the time-series model, $a_1,a_2,...,a_n$ and b_1, $b_2,...,b_n$ denote their respective vectors. Further, A (and B) denotes the set of distinct elements appearing in the first stream (respectively, the second stream). The Jaccard similarity between the streams is given by $|A \cap B|/|A \cup B|$.

The first explicit study of the Jaccard similarity between two streams was given by Datar and Muthukrishnan [9]. Their paper examined the sliding window model, which is discussed further in the next section. However, a datastream algorithm in the standard model was given implicitly in the work of Cohen et al. [6], although the notion of datastreams is never mentioned in the paper.

The major technical tool uses *min-wise hashing*, or min-hashing [3,5]. For every subset A of $[n]$, the *min-hash* for A (with respect to π), denoted $h_\pi(A)$, is defined to be $h_\pi(A) = \min_{i \in A}\{\pi(i)\}$, where π denotes a permutation on $[n] =\{ 1,...,n\}$. The wonderful property of the min-hash is that, when π is chosen uniformly at random from the set of all permutations on $[n]$, for any two subsets A,B of $[n]$, it is the case that

$$Pr[h_\pi(A) = h_\pi(B)] = \frac{|A \cap B|}{|A \cup B|}.$$

This suggests the following algorithm.

The algorithm chooses π uniformly at random from the set of permutations on $[n]$. (The fact that storing π take $\theta(n \log n)$ space will be discussed momentarily.) For the first stream, the algorithm finds the value $h_\pi(A) = \min_{i \in A}\{\pi(i)\}$, where A is the set of distinct elements occurring in the first stream. This is simple to do in a datastreaming fashion: as each new a_j appears, the algorithm updates the min value if $\pi(a_j)$ is smaller than the min seen so far. Likewise, for the second stream, the algorithm finds the value $h_\pi(B)$, where B is the set of distinct elements occurring in the second stream. From the above, the probability that the two values are equal is precisely the Jaccard similarity between the two streams.

Of course, to obtain an accurate estimate of this probability, the algorithm needs to run the procedure multiple times. In this case, it will run the procedure in parallel k times, each with an independently chosen random permutation. (Here, $k = O(\varepsilon^{-3}\log(1/\delta))$.) The value ρ is defined to be the fraction of times (out of k) that the min values for the two streams coincide. That is, if $\pi_1,...,\pi_k$ are the k independently chosen random permutations, then

$$\rho = \frac{1}{k} \cdot \#\left|\{j : h_{\pi_j}(A) = h_{\pi_j}(B)\}\right|.$$

It is shown in [11] that with probability at least $1 - \delta$, the value ρ approximates the Jaccard similarity within multiplicative factor $(1 \pm \varepsilon)$.

In order for the above algorithm to be useable in a datastreaming context, it must be able to generate and store the necessary random permutations in small space. This is done using *approximately* min-wise independent hash functions. Although this introduces additional error, it can be done in small space and time. The reader is referred to [13] for more details.

Sliding Windows

In many applications, the data from streams becomes outdated or unnecessary quickly. To help understand this scenario better, researchers have proposed the *sliding window* model of datastreams. Here, the algorithm must maintain statistics (e.g., stream similarity), using only the last N items from the stream, for some N. This causes additional complications, since as each new item comes in, an old item is removed. Since memory is limited, algorithms cannot track which of these old items is disappearing. Still, there are datastream algorithms for both L_p distance and Jaccard similarity in the sliding window model.

In [8], Datar et al. define the sliding window model, and give a datastream algorithm for approximating the L_p distance between two streams (as well as several other datastream algorithms). Their technique uses what they call an *exponential histogram.* The histogram partitions the last N items (i.e., those items in the sliding window) into buckets; the last bucket may in fact contain items older than the last N. Each bucket maintains the necessary statistics for the items it contains. For instance, a bucket containing the items $a_s,a_{s+1},...,a_t$ would hold the L_p-sketch for those items. (Due to memory constraints, the bucket cannot actually maintain the values of all the items it holds.)

As new items come in, the algorithm merges old buckets to maintain the histogram structure, creating new buckets only for newly encountered items. The last bucket will eventually contain only items that do not appear in the N most recent, and will be removed from the histogram at this time. Datar et al. observe that the additional error in this windowed model, beyond that of the standard model, comes from the fact that the last bucket may contain items that are no longer in the N-item window. But the structure of the exponential histogram ensures that this error is not too large. Hence, they provide a general method for translating a wide range of datastream algorithms into windowed-datastream algorithms.

Datar and Muthukrishnan [9] study the problem of approximating the Jaccard similarity of two streams in the sliding window model. As in the non-windowed version, they use min-hashing as a primary tool. The main complication in the sliding window model is that maintaining the minimum value over a sliding window is hard. At a given time step t, the algorithm needs to know the value $\min_{i=t,\ldots,t-N+1}\{\pi_j(a_i)\}$, where π_j is a permutation chosen by the datastream algorithm in the standard model. Their solution is to maintain the value $\pi_j(a_i)$ for every *relevanti* $= t,\ldots,t - N + 1$. For instance, if $\pi_j(a_i) > \pi_j(a_{i+s})$ for some $s > 0$, then the value $\pi_j(a_i)$ will never be the minimum over the sliding window at any time; hence, it may be discarded. (Here, item a_i occurs earlier than a_{i+s}, thus item a_i will move out of the window before a_{i+s}.) Amazingly, with high probability, the number of relevant values that need to be maintained is at most $O(\log n)$. Hence, the standard datastream algorithm can be adapted to the sliding window model, using small space.

Lower Bounds for Stream Distance

The major technique for proving lower bounds utilize reductions from communication complexity. Here, only sketches of the very high level ideas are presented, with some of the main results cited.

An often-used communication complexity problem is DISJOINTNESS: Alice is given a set, A, and Bob is given a set, B. Neither knows what the other set is. They must communicate with each other by sending messages back and forth, until they decide whether $A \cap B$ is nonempty. (They are allowed to decide ahead of time the protocol they will use to communicate messages.) It has been shown that if the size of A and B is $\theta(n)$, the communication complexity (i.e., the number of bits that must be communicated in the worst case) is also at least $\theta(n)$ [14].

A datastream algorithm that calculates the distance between two streams can provide the basis for a communication complexity algorithm. A typical reduction gives a method for Alice to transform her set A into a datastream (without looking at set B). Likewise, the reduction gives a method for Bob to transform B into a datastream, without looking at A. Finally, the reduction guarantees that Alice's datastream and Bob's datastream are close if and only if $A \cap B$ is non-empty. Then Alice can begin running the datastream algorithm on her datastream. When it has processed her stream, the algorithm will have some memory bits indicating its current state. Alice sends a message to Bob, telling him that state. Bob can then finish running the datastream algorithm on his own datastream. If the algorithm indicates that the two streams are close, he knows $A \cap B$ is nonempty; otherwise, he knows that $A \cap B =$ (and may communicate this to Alice in one bit). Hence, Alice and Bob have solved their communication complexity problem. Since the original communication complexity problem took at least $\theta(n)$ bits, the datastream algorithm must also use at least this much memory. (In this case, showing that it cannot be space efficient.)

There is, of course, a great deal of technical work in providing the proper reductions; the difficulties are even greater when showing lower bounds for approximations. However, building on these ideas, Saks and Sun [15] show that approximating the L_∞ distance between two datastreams is impossible to do in sublinear space. In fact, their work shows that approximating within factor $n^{O(\varepsilon)}$ the L_p distance for any $p \geq 2 + \varepsilon$ requires space at least $n^{O(\varepsilon)}$. For p close to 2, this has very little practical implications, but the bounds become more meaningful for large p. Much simpler reductions show the impossibility of space-efficient datastream algorithms for approximating the length of the longest common subsequence between two datastreams (viewed as sequences).

Key Applications

Tracking Change in Network Traffic

The datastream algorithms outlined above allow one to take an entire day of network traffic and synopsize it using a small sketch. It is then possible to measure how different traffic is from day-to-day. Large changes in the network traffic can signal denial of service attacks or worm infestations.

Query Optimization

Most query-optimization techniques utilize data statistics to produce better plans. The L_2 norm is a useful measure for approximating join sizes, while the L_0 norm gives the number of distinct items in the stream.

Processing Genetic Data

Since genetic data consists of millions or billions of base pairs for an individual, it is useful to think of them as streams of data. The similarity of two base-pair sequences is a fundamental concept.

Data Mining

Often individual entities are represented by massive streams of data (e.g., phone calls from a large company, or IP addresses of users visiting a given web site, or items bought at a grocery store). Estimating the similarity between these streams can be a useful tool for identifying similar entities. As one example, it is possible to determine which web sites are most similar to each other, based on the IP addresses of their visitors.

Cross-references

- ▶ Approximation and Data Reduction Techniques
- ▶ Stream Data Management
- ▶ Stream Mining

Recommended Reading

1. Alon N., Gibbons P., Matias Y., and Szegedy M. Tracking join and self-join sizes in limited storage. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 10–20.
2. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. In Proc. 28th ACM Symp. on Theory of Computing, 1996, pp. 20–29.
3. Broder A., Charikar M., Frieze A., and Mitzenmacher M. Min-wise independent permutations. In Proc. of the 30th ACM Symp. on Theory of Computing, 1998, pp. 327–336.
4. Chambers J.M., Mallows C.L., and Stuck B.W. A method for simulating stable random variables. J. Am. Stat. Assoc., 71:340–344, 1976.
5. Cohen E. Size-estimation framework with applications to transitive closure and reachability. J. Comput. Syst. Sci., 55:441–453, 1997.
6. Cohen E., Datar M., Fujiwara S., Gionis A., Indyk P., Motwani R., and Ullman J. Finding interesting associations without support pruning. In Proc. 16th International Conf. on Data Engineering, 2000.
7. Cormode G., Datar M., Indyk P., and Muthukrishnan S. Comparing data streams using hamming norms. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 335–345.
8. Datar M., Gionis A., Indyk P., and Motwani R. Maintaining stream statistics over sliding windows. In Proc. 13th Annual ACM-SIAM Symp. on Discrete Algorithms, 2002, pp. 635–644.
9. Datar M. and Muthukrishnan S. Estimating rarity and similarity on data stream windows. In Proc. 10th European Symp. on Algorithms, 2002.
10. Feigenbaum J., Kannan S., Strauss M., and Viswanathan M. An approximate l_1-difference algorithm for massive data streams. In Proc. 40th Annual Symp. on Foundations of Computer Science, 1999.
11. Flajolet P. and Martin G. Probabilistic counting. In Proc. 24th Annual Symp. on Foundations of Computer Science, 1983, pp. 76–82.
12. Indyk P. Stable distributions, pseudorandom generators, embeddings and data stream computation. In Proc. 41st Annual Symp. on Foundations of Computer Science, 2000, pp. 189–197.
13. Indyk P. A small approximately min-wise independent family of hash functions. J. Algorithm., 38:84–90, 2001.
14. On the distributional complexity of disjointness. J. Comput. Sci. Syst., 2, 1984.
15. Saks M. and Sun X. The space complexity of approximating the frequency moments. In Proc. 34th ACM Symp. on Theory of Computing, 2002.

Streaming Algorithm

▶ One-Pass Algorithm

Streaming Applications

Yanif Ahmad, Uğur Çetintemel
Brown University, Providence, RI, USA

Synonyms

Stream-oriented applications; Continuous query processing applications

Definition

Streaming applications typically involve the processing of continuous data streams for the purposes of filtering, aggregation, correlation, transformation, pattern matching and discovery, and domain-specific temporal analytics. These applications often require such continuous processing to be performed with both high throughput and low latency, and are able to tolerate approximate results and forego some of the persistence requirements of standard database transaction processing applications.

Key Points

A large fraction of streaming applications are monitoring oriented: they involve the tracking of events or activities to identify and act upon situations (or patterns) of interest, either manually or automatically. This so-called "sense and respond" model requires query results to be generated in real-time (meaning low latency) as results lose their utility over time. As such, persistence of all the input data is often not an application requirement, unlike in traditional database applications. Thus, most input data can be simply discarded or, alternatively, asynchronously recorded for archival needs. As on-line data sources proliferate and consolidate, streaming applications need to deal with increasingly higher volume streams, which makes real-time operation especially challenging.

A flagship stream processing application is automated trading, which continually watches market streams (bids and asks) from financial feed providers (e.g., Reuters), evaluating sophisticated real-time patterns over them to identify arbitrage opportunities and automatically act on them. For automated trading, the desired processing latencies are in milliseconds (and continually decreasing) and the estimated peak input data rates are in 170,000 messages/s (as of July 2006), with rates roughly doubling every year [1]. Network monitoring is another application that has stringent real-time response needs under high data volumes: network elements (e.g., routers, gateways) are instrumented to log, summarize and forward traffic data, which is then analyzed to identify and automatically respond to online security attacks (e.g., denial of service attacks) and QoS problems (e.g., SLA violations). Another early streaming application is event detection in MMORPGs, where the virtual game world is continually monitored to identify oddities, semantic bugs and cheats.

Overall, streaming applications abound in various verticals including:

- Financial services: automated trading, market feed processing (cleaning, smoothing, and translation), smart order routing, real-time risk management and compliance (MiFID, RegNMS)
- Government and military: surveillance, intrusion detection and infrastructure monitoring, battlefield command and control
- Telecommunications: network management, quality of service (QoS)/service level agreement (SLA) management, fraud detection
- Web/E-business: click-stream analysis, real-time customer experience management (CEM)
- Entertainment: online gaming (online cheat, bug detection)
- Retail and logistics: automated supply-chain management
- Healthcare: patient monitoring
- Energy: power-grid/pipeline monitoring and control

Cross-references

► Data Stream Management Architectures and Prototypes
► Stream-oriented Query Languages and Operators

Recommended Reading

1. Options Price Reporting Authority (OPRA) Traffic Projections, http://www.opradata.com/specs/projections_2005_2006.pdf

Streaming Database Systems

► Event Driven Architecture

Stream-Oriented Applications

► Streaming Applications

Stream-Oriented Query Languages and Operators

Mitch Cherniack[1], Stan Zdonik[2]
[1]Brandeis University, Wattham, MA, USA
[2]Brown University, Providence, RI, USA

Synonyms

Continuous query languages

Definition

Many research prototypes and commercial products have emerged in the new area of stream processing. All of these systems support a language for specifying queries. A fundamental difference between a stream query language and a conventional query language like SQL is that stream queries are not one-time computations, but rather, they continue to produce answers as new tuples arrive on one or more input streams. Thus, queries are registered with the system and answers continue to evolve over time. This new assumption is crucial to understanding some of the technical differences that arise in stream query languages.

Most stream query languages try to extend SQL in one way or another. The form of these extensions can be either a purely textual extension of SQL or GUI, through which users can construct dataflow diagrams that connect extended versions of relational operators. These days, many systems provide both.

The most fundamental addition to a stream query language over their relational counterparts is the notion of a window. Windows produce finite structures (i.e., tables) from infinite structures (i.e., streams). Much of the technical detail of a streaming data model revolves around the specifics of how windows are formed. This will be discussed in detail below.

One of the biggest technical challenges in the implementation of such a stream query language is the ability to produce answers with minimum latency. The latency requirement is a reaction to the kinds of applications that stream processing was invented to address. In broad terms, these applications have to do with monitoring conditions on the input streams. Typically, in this setting, the value of the answer decays quickly.

The main thrust of this article is on the technical choices that must be faced by anyone who designs a stream query language. The main concepts with examples from major systems will be illustrated. There are also many related technologies and associated languages (e.g., XML streams, temporal databases, active databases) that are not discussed in this article. The focus is on extensions to the relational model and relational languages (e.g., relational algebra, SQL) that incorporate streams.

Historical Background

There has been significant work on stream query languages in the past. The academic languages include:

CQL [3,10] (from the STREAM project out of Stanford),

SQuAl [1,6] (from the Aurora/Borealis project out of Brandeis, Brown and MIT), and

ESL [4,13] (from the Atlas project out of UCLA).

The commercial languages include:

StreamSQL [11] (from Streambase),

CCL [9] (from Coral8),

EQL [7] (from Esper), and

StreaQuel [5,8] (from Truviso).

The commercial stream query languages are, in many cases, derived from the academic languages; StreamSQL is derived from SQuAl and CQL, CCL is derived from CQL, and StreaQuel is derived from a language of the same name from the Telegraph CQ project out of UC Berkeley. The material from which information about these languages was gleaned is listed at the end. Especially in the case of commercial languages, published documentation on these languages is sometimes incomplete. Thus, there may be omissions in the descriptions of the features of these languages that follow. For example, documentation on StreaQuel is especially scant and so the description for this language is likely to be incomplete.

Foundations

Stream query languages (both academic and commercial) primarily differ in how they approach the most fundamental requirements of stream processing. These requirements include:

1. *Language Closure*: Are the language's operators closed under streams? Or does the language support operators that convert streams to relations and/or relations to streams? The approach a language takes to closure reveals a lot about how tightly integrated is the language with a relational query language such as SQL.
2. *Windowing*: Does the language support first-class windows? (I.e., are windows namable, sharable and queryable?). Or are windows internal to the definition of stateful operations. And what kinds of windows can be expressed in the language?
3. *Correlation*: Does the language provide a way to correlate tuples (events) arriving on a stream with historical data, and with tuples arriving on a separate stream?
4. *Pattern Matching*: Does the language have a way to identify interesting subsequences of tuples on one or more streams?

Language Closure: The simplicity of the relational algebra/calculus is largely due to the closure property that says that all inputs and outputs of relational queries are relations. Beyond simplifying the type system, closure also ensures that the output of any one query can be input to another.

The various stream query languages compared here approach closure in one of two ways. Languages such as SQuAl and EQL define a language that is closed under streams. That is, every operator in these two languages accepts one or more streams as input and produces streams as output. (SQuAl includes two operators that have the side-effect of accessing a relation (*WRITESQL* and *READSQL*), but both return stream outputs.) The other languages in this list include both streams and relations in their type system. For example, CQL includes no stream-to-stream operations. Instead, operation on a stream demands that it first be converted into a relation (via windowing). Thereafter, all query operations are relation-to-relation operations as in SQL. If the desired result is a stream, the specialized operators *ISTREAM*, *DSTREAM* and *RSTREAM* produce stream outputs from relational inputs by (essentially) returning the log of changes to the relations in the order in which they occur. The other languages in this list (StreamSQL, CCL and StreaQuel) largely follow the CQL model, except that queries in these languages that contain exactly one unwindowed stream in the FROM clause are considered to be stream-to-stream. For example, in these languages, the query,

```
SELECT *
FROM S
WHERE p
```

returns a stream if S is a stream. In CQL, this query is assumed to be a syntactic shorthand for,

```
SELECT ISTREAM *
FROM S [∞]
WHERE p
```

and thus also returns a stream, but only as a result of first windowing S into a relation (see the subsection below on windowing) and then converting the result back into a stream.

Many of the query languages that include both relations and streams in their data model also define query operations that produce relations from streams and streams from relations. As mentioned previously, CQL produces streams from relations using windows, and the specialized operations *ISTREAM*, *DSTREAM* and *RSTREAM* produce relations from streams. StreamSQL, CCL and StreaQuel all include additional operations for producing relations from windows (*INSERT*, *UPDATE* and *DELETE*), which update prespecified relations with the arrival of each tuple on a stream in the same way that the equivalent SQL operations update a relation with an individual tuple. StreaQuel and StreamSQL also support *ISTREAM* and *DSTREAM* (and in the case of StreaQuel, *RSTREAM*) operations that produce streams from relations, as in CQL. No such operations are described in the publicly available literature on CCL as of the time of this writing.

Table 1 summarizes the closure properties of the academic and commercial languages studied in this report, as well as the operations the languages support (if any) for producing relations from streams and streams from relations. Note that all stream languages support windowing as a means of producing a relation from a stream. However, some of these languages (e.g., SQuAl, ESL, EQL) are still considered to be closed under streams because their window definitions are internal to the query's operation and not output.

Windowing: All stream query languages have some form of windowing to convert infinite streams into automatically maintained, time-varying relations. But different query languages vary in whether they support first-class windows, and in terms of the features of window definition that they support. Table 2 summarizes how various stream query languages support windows.

A first-class window is a window that can be named, shared and independently queries. Put simply, a query language supports first-class windows allows a window to be named and defined as the result of a statement in the query language, and subsequent queries can then access this window. Windows are not first-class if they are defined as part of a query, but are not visible outside of the execution of that query. First-class windows are typically supported in query languages that are closed under both streams and relations (i.e., CQL, StreamSQL, CCL and StreaQuel). Languages such as SQuAl, EQL and ESL that are closed under streams define windows internally within queries.

All windows are characterized as having a certain size, and advancing in some way (i.e., adding new

Stream-Oriented Query Languages and Operators. Table 1. Closure properties of stream query languages

Features		Closed under	Operations		
			Stream-to-stream	Stream-to-relation	Relation-to-stream
Academic Languages	CQL (STREAM)	Streams, Relations	No, but many queries assume use of ISTREAM	Windows	ISTREAM, DSTREAM, RSTREAM
	SQuAl (Aurora)	Streams	Default	Windows, WRITESQL	READSQL
	ESL (Atlas)	Streams	Default	Windows	-
Commercial Languages	StreamSQL (Streambase)	Streams, Relations	All queries with unwindowed stream in FROM clause	Windows, INSERT, UPDATE, DELETE	ISTREAM, DSTREAM
	CCL (Coral8)	Streams, Relations	All queries with unwindowed stream in FROM clause	Windows, INSERT, UPDATE, DELETE	-
	EQL (Esper)	Streams	Default	Windows	-
	StreaQuel (Truviso)	Streams, Relations	Default (from implicit use of ISTREAM and RSTREAM)	Windows, Active Tables	?

Stream-Oriented Query Languages and Operators. Table 2. Windowing support in stream query languages

Features			Windows (sizing)				Windows (movement)		Windows (other features)	
		First-class	Row-based	Time-based	Value-based	Sliding	Tumbling	Sampling	Top-K	Landmark
Academic Languages	CQL (STREAM)	Yes	Yes	Yes	No	Yes	Yes	X% SAMPLE	-	-
	SQuAl (Aurora)	No	Yes	Yes	Yes	Yes	Yes	RESAMPLE	BSORT	-
	ESL (Atlas)	No	Yes	Yes	No	Yes	Yes	Definable with UDA's	Definable with UDA's	Definable with UDA's
Commercial Languages	StreamSQL (Streambase)	Yes	Yes	Yes	Yes	Yes	Yes	-	EVICT MIN/ MAX	-
	CCL (Coral8)	Yes	Yes	Yes	No	Yes	Yes	-	KEEP LARGEST/ SMALLEST	Yes (KEEP FOR DURATION)
	EQL (Esper)	No	Yes	Yes	No	Yes	Yes	-	-	-
	StreaQuel (Truviso)	Yes	Yes	Yes	No	Yes	Yes	?	?	Yes

tuples and deleting old ones as new tuples arrive on a stream). Though each language has its own syntax for specifying how to size and advance a window, all of the languages discussed looked at support windows whose size is "row-based" (i.e., defined by the number of tuples contained in the window) or "time-based" (i.e., defined by the maximum time interval between any two tuples in the window). SQuAl and StreamSQL also support "value-based" windows over streams whose tuples are known to arrive in ascending order on some data field of the tuple. Value-based windows specify the maximum difference in value of that attribute between tuples in the same window. All languages studied in this article looked at support sliding by some query-specified amount (expressed in number of tuples, increment in time, or (in the case of SQuAl and StreamSQL), increment in value of the attribute over which the windowed stream is ordered. As well, all of the languages discussed here support "tumbling windows," which are windows that contain tuples that belong to exactly one window.

The last 3 columns of Table 2 show some of the window properties that are supported by some but not all of the query languages discussed here. For example, both CQL and SQuAl support a form of sampling to determine the contents of a window. In the case of CQL, a window defined with *SAMPLE* will consist of a subset of the most items that have arrived on the windowed stream. In the case of SQuAl, a window defined with *RESAMPLE* will use interpolation with a user-supplied function to fill-in missing values from the windowed stream. SQuAl, StreamSQL and CCL all support "top-k" (or "bottom-k") windows, which at any point in time, contain those tuples that have arrived on the windowed stream that have the maximum (minimum) values of some specified attribute. And CCL and StreaQuel both support "Landmark Windows"; so-named because these are windows which are fixed at the start-point with an end-point that advances as tuples arrive on the windowed stream. ESL has expressive user-defined aggregate support whereby aggregates are defined with SQL statements on an internal table that specify how to initialize, increment and return a final result, and this mechanism could be used to specify each of the window types described here.

Correlation: A key component of any stream query language is its support for correlating tuples appearing on a stream with either a repository of historical data, or with the tuples appearing on another stream. As Table 3 shows, all of the query languages studied here support both forms of correlation, though in different ways.

The correlation of a tuples on a stream with historical data are expressed in most languages by allowing exactly one stream to appear in a stream query's FROM clause. Then, either periodically (e.g., CQL) or upon the arrival of each tuple on this stream (e.g., StreamSQL), the query is reevaluated using the tuple(s) that have arrived on the stream since the last time the query was evaluated. ESL, StreamSQL, CCL, EQL and StreaQuel all allow FROM clauses to include one unwindowed stream for this purpose. A CQL query containing one unwindowed stream implicitly windows that stream using the "NOW" size directive, which says to create a window with all tuples that have arrived on the stream since the last time when all queries were reevaluated. Thus, the FROM clause of a CQL query always consists solely of relations including those resulting from windowing streams. SQuAl, which has a graphical rather than SQL-like notation, uses the operation, *READSQL* to correlate stream data with historical data.

Most query languages correlate the tuples on two streams by first windowing at least one (or in case of CQL, both) of the streams. The resulting query then is

Stream-Oriented Query Languages and Operators. Table 3. Correlation support in stream query languages

Features		Correlation	
		Stream-to-relation	Stream-to-stream
Academic Languages	CQL (STREAM)	No, but relation-to-relation correlation with window on most recent tuples in stream (NOW) often has same effect	Must window both streams
	SQuAl (Aurora)	READSQL	JOIN
	ESL (Atlas)	JOIN with Stream, Relation in FROM Clause	Must window one of the streams
Commercial Languages	StreamSQL (Streambase)	JOIN with Stream, Relation in FROM Clause	Must window one of the streams
	CCL (Coral8)	JOIN with Stream, Relation in FROM Clause	Must window one of the streams
	EQL (Esper)	JOIN with Stream, Relation in FROM Clause	Must window one of the streams
	StreaQuel (Truviso)	JOIN with Stream, Relation in FROM Clause	Must window one of the streams

Stream-Oriented Query Languages and Operators. Table 4. Pattern matching support in stream query languages

Features		Pattern matching	
		Multi-stream	Regular expression
Academic Languages	CQL (STREAM)	-	-
	SQuAl (Aurora)	-	-
	ESL (Atlas)	-	-
Commercial Languages	StreamSQL (Streambase)	Yes (MATCH)	Yes (PATTERN)
	CCL (Coral8)	Yes (MATCHING)	No
	EQL (Esper)	Yes (PATTERN)	Yes (PATTERN)
	StreaQuel (Truviso)	Yes (EVENT clause)	Simple (A B C)

either a join of two relations as in SQL, or a join of an unwindowed stream with a relation as described in the previous paragraph. StreamSQL also has a *GATHER* operation which performs key matching to match each tuple on an input stream with the single tuple it matches on each of the other input streams.

Pattern Matching: Pattern matching is a relatively new addition to stream query languages, and is only supported in the commercial languages examined in this study (though there exist some work in the academic literature on pattern matching on streams such as [12]). Pattern matching in stream query languages can take one of two forms:

Multi-stream pattern matching resembles joins between streams in that it correlates tuples appearing on separate streams according to the order in which they arrive. For example, this form of pattern matching might identify all cases where a particular stock received a bid quote (on a BIDS stream) without a corresponding ask quote (on an ASKS stream) within some specified time period. All of the commercial stream query languages examined here (StreamSQL, CCL, EQL and StreaQuel) support this form of pattern matching.

Single-stream pattern matching looks for a sequence pattern of tuples arriving on a single stream, and typically uses a rich pattern matching language based on regular expressions to express desired patterns. For example, this form of pattern matching might identify all sequence of quotes for a particular company that resulted in an "M-pattern" whereby the stock's price rises for a time, then falls, then rises again and then falls again [2]. This form of pattern matching is part of a general SQL standard proposal put forth by Streambase, Oracle and IBM, and documented in [13].

A summary of each query language's support for pattern matching is shown in Table 4.

In short, while there are several query languages for streams with both academic and commercial roots, these languages share much in common that identify the crucial requirements of stream processing. Specifically, all of these languages are closed either under streams, or under streams and relations; all have some notion of windowing to convert streams to relations; all provide ways to correlate tuples on a stream with both historical data and tuples on other streams, and all commercial languages support some form of pattern matching to identify interesting subsequences of tuples from one or more streams.

Key Applications

The applications of stream processing revolve around low-latency monitoring of physical or virtual items or events of interest. Examples include automated trading, network security monitoring, and event detection in massively multiplayer on-line games.

Cross-references

- ▶ Continuous Query
- ▶ Data Stream
- ▶ Event and Pattern Detection over Streams
- ▶ Punctuations
- ▶ Stream Processing
- ▶ Windows

Recommended Reading

1. Abadi D., Carney D., Cetintemel U., Cherniack M., Convey C., Lee S., Stonebraker M., Tatbul N., and Zdonik S. Aurora: a new model and architecture for data stream management. VLDB J., 12(2):120–139, 2003.

2. Anon.s Pattern matching in sequences of rows. SQL Standard Proposal, http://asktom.oracle.com/tkyte/row-pattern-recogniton-11-public.pdf, March, 2007.
3. Arvind A., Shivnath B., and Jennifer W. The CQL continuous query language: semantic foundations and query execution. VLDB J., 15(2):121–142, 2006.
4. Bai Y, Thakkar H., Luo C., Wang H., and Zaniolo C. A data stream language and system designed for power and extensibility. In Proc. Int. Conf. on Information and Knowledge Management, 2006, pp. 337–346.
5. Chandrasekaran S. and Franklin M. Streaming queries over streaming data. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 203–214.
6. Cherniack M. SQuAl: The Aurora [S]tream [Qu]ery [Al]gebra, Technical Reprt, Brandeis University, 2003.
7. Codehaus.org, Esper online documentation set, http://esper.codehaus.org/tutorials/tutorials.html, 2007.
8. Conway N. An introduction to data stream query processing. Slides from a talk given on May 24, 2007, http://www.pgcon.org/2007/schedule/attachments/17-stream_intro.pdf, 2007.
9. Coral8 Systems, Coral8 CCL Reference Version 5.1, http://www.coral8.com/system/files/assets/pdf/current/Coral8CclReference.pdf, 2007.
10. Jennifer W. CQL: a language for continuous queries over streams and relations. Slides from a talk given at the Database Programming Language (DBPL) Workshop, Potsdam, Germany, 2003. http://www-db.stanford.edu/~widom/cql-talk.pdf
11. Streambase Systems, StreamSQL online documentation set, http://streambase.com/developers/docs/latest/streamsql/index.html, 2007.
12. Wu E., Diao Y., and Rizvi S. High-performance complex event processing over streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 407–418.
13. Zaniolo C., Luo C., Wang H., Bai Y., and Thakkar H. An Introduction to the Expressive Stream Language (ESL), Technical Report, UCLA.

Strong Consistency Models for Replicated Data

ALAN FEKETE
University of Sydney, Sydney, NSW, Australia

Synonyms

Strong memory consistency; Copy transparency

Definition

If a distributed database system keeps several copies or replicas for a data item, at different sites, then a replica control protocol determines how the replicas are accessed. Some replica control protocols ensure that clients never become aware that the data are replicated. In other words, the system provides the transparent illusion of an unreplicated database. Such a system is described as offering a strong consistency model. 1-copy-serializability (q.v.) is the best-known strong consistency model.

Historical Background

Early work in the 1970s investigated a range of replica control mechanisms, usually with the intention of providing transparent serializability. In the early 1980s, Bernstein and colleagues formalized the concept of 1-copy-serialiability as a consistency model [1], with a careful proof technique [2] like that for single-site serializability. Herlihy [8] extended these ideas to replicating data types with general operations (not just read and write).

1996 marked the seminal paper by Gray et al. who used some simple performance models to show the scalability barriers for different system designs [7]. This inspired research on ways to gain 1-copy serializability within a lazy single-master replica control, through restrictions on the placement of copies, and/or the ordering of update propagation [3,4].

Since 2000, the prevalence of DBMS platforms with Snapshot Isolation concurrency control led to research on replication for these, especially by groups of researchers centered on Alonso and Kemme. The concept of 1-copy-SI was defined [10], and a proof theory was developed [9]. Variant consistency models were defined by considering different session properties [5,6].

Foundations

A major theme in the development of distributed databases has been *transparency*, that is, the clients should not have to change if they interact with a distributed system rather than a traditional, single-site database. Transparency applies to many aspects: table naming should be the same as for a single DBMS, queries should not need rewriting even if data is fragmented between sites, etc. The shift from a single-site system to a replicated one should be seen in better quality of service, but not in altered functionality. Since a single-site database has only one copy of each data item, a transparently replicated system will provide clients with the illusion of a single copy, hiding all evidence of the replication. A scheme for replica control (q.v.) that does this can be described as providing a strong consistency model. Other replica control mechanisms do not hide the fact of replication.

There are in fact several variants among strong consistency models, because there are several different isolation models used by different DBMS platforms, and because the formal definition of isolation doesn't always capture exactly the properties of an implementation. The next paragraphs describe the main strong consistency models that have been proposed for replicated data.

Replicated Serializability. The theory of concurrency control has an established notion of correct functionality for ACID transactions in a single-site DBMS: serializability (q.v.). This is defined by having execution equivalent to a serial (i.e., batch, non-overlapped) execution of the same transactions. When a replicated database gives to clients the transparent appearance of a single-site system with serializable transactions, one says that the replica consistency model is 1-copy-serializability (q.v.). That is, the operations of the transactions are indistinguishable from what happens if they are run serially in a database with only one site.

To illustrate the consistency model, consider a database with two logical items, x and y representing respectively the balance in the checking and savings bank accounts for a single customer. There is a single client C which submits two transactions. One client transaction $T_{1,C}$ is a transfer of two units of funds from y to x, and $T_{2,C}$ is a transaction to display the status of the customer's finances. Initial values are $x = 10$ and $y = 20$.

In (1) below is a sequence of events that might happen in a system with two sites A and B, using an eager locking-based read-one-write-all replica control. Notice how each client-submitted transaction has subtransactions at the local sites A and B; $T_{1,C}$ reads both items at site A, and then updates replicas at both sites; while $T_{2,C}$ reads the replica x^A and the replica y^B. The notation that is used in this and later examples is to indicate the event where a value 5 is written to the local replica of item x at site A, as part of transaction T_1, by $w_1[x^A,5]$. Here the subscript on the event type indicates the transaction involved, and the superscript on the item name indicates the site of the replica which is affected. The event where a client C running transaction T_3 has requested a read of the logical data item y, and the value 6 is returned, will be represented by $r_{3,C}[y,6]$. Note there is no superscript on the item since this is the client's view. Many consistency models need to refer to where transactions start and finish, so one also has events like $b_{3,\ C}$ for the start of transaction T_3 at client C, or $c_{3,\ C}$ for the commit of that transaction by the client, or indeed ${c_3}^A$ for the commit of the local subtransaction of T_3 which is running at site A.

$$\begin{gathered}
b_{1,C}\, b_1^A\, r_1[x^A, 10]\, r_{1,C}[x, 10]\, b_{2,C}\, b_2^A\, r_1[y^A, 20] \\
r_{1,C}[y, 20]\, w_1[x^A, 12]\, b_1^B\, w_1[x^B, 12]\, w_{1,C}[x, 12] \\
w_1[y^A, 18] w_1[y^B, 18]\, w_{1,C}[y, 18]\, c_1^A\, r_2[x^A, 12] \\
r_{2,C}[x, 12] b_2^B\, c_1^B\, c_{1,C}\, r_2[y^B, 18] \\
r_{2,C}[y, 18]\, c_2^A\, c_2^B\, c_{2,C}
\end{gathered} \tag{1}$$

When one hides the internal details (the events at the replicas), and only considers what the client sees, the relevant event sequence is

$$\begin{gathered}
b_{1,C}\, r_{1,C}[x, 10]\, b_{2,C}\, r_{1,C}[y, 20]\, w_{1,C}[x, 12] \\
w_{1,C}[y, 18]\, r_{2,C}[x, 12]\, c_{1,C}\, r_{2,C}[y, 18]\, c_{2,C}
\end{gathered} \tag{2}$$

This sequence is 1-copy-serializable, because the client sees the same as in a serial execution on a single-site DBMS, in the order $T_{1,C}$ then $T_{2,C}$.

Here is a sequence that might occur in a system where site A is the primary or master site, where all updates are initially done, and site B has secondary replicas which are updated through copier transactions that lazily apply the write sets of any update transaction. The copier that transmits values produced by $T_{1,\ C}$ from site A to site B will be denoted by ${T_\oplus}^B$; notice that in contrast to ${T_1}^B$ in the eager system modeled previously, ${T_\oplus}^B$ is not a subtransaction of any global client-submitted transaction but instead it can commit independently.

$$\begin{gathered}
b_{1,C}\, b_1^A\, r_1[x^A, 10]\, r_{1,C}[x, 10]\, r_1[y^A, 20]\, r_{1,C}[y, 20] \\
w_1[x^A, 12]\, w_{1,C}[x, 12]\, w_1[y^A, 18] \\
w_{1,C}[y, 18]\, c_1^A\, c_{1,C} b_{2,C} \\
b_2^B\, r_2[x^B, 10]\, r_{2,C}[x, 10]\, r_2[y^B, 20]\, r_{2,C}[y, 20]\, c_2^B \\
c_{2,C}\, b_\oplus^B\, w_\oplus[x^B, 12]\, w_\oplus[y^B, 18]\, c_\oplus^B
\end{gathered} \tag{3}$$

Again hiding the internal details, and considering only what the client sees, it is

$$\begin{gathered}
b_{1,C}\, r_{1,C}[x, 10]\, r_{1,C}[y, 20]\, w_{1,C}[x, 12] \\
w_{1,C}[y, 18]\, c_{1,C} \\
b_{2,C}\, r_{2,C}[x, 10]\, r_{2,C}[y, 20]\, c_{2,C}
\end{gathered} \tag{4}$$

This sequence is also 1-copy-serializable, since the values read are what could happen with an unreplicated system running the transactions serially in the order $T_{2,C}$ then $T_{1,C}$.

There is a detailed theory that allows one to prove that this property holds for schedules of certain read-one-write-all replica control mechanisms.

Session properties. The sequence of events shown in sequence (4) is indeed something that could happen according to the definition of serializable execution in a single-site DBMS, but it would never happen in a single site DBMS that used a concurrency control mechanism like two-phase locking. It might be very disturbing to be a client, who submits a transfer transaction $T_{1,C}$, learns that the transfer succeeded, and then uses $T_{2,C}$ to check the status of their accounts and is told that the initial balances are still unchanged. In the definition of serializability, it is enough that there exists some way to order the transactions and perform them serially; but in DBMS products, the concurrency control makes sure that the apparent serial order does not rearrange transactions unless they are actually concurrent (that is, unless they overlap). Any single-site system will not allow a situation where $T_{1,C}$ has completed, and then $T_{2,C}$ starts, but the apparent serial order has an inversion of the transaction order, with $T_{2,C}$ coming first. Thus it is often proposed to have explicit session properties [11] in a consistency model, to require that the apparent order have some relationship to what really happened. A very restrictive session requirement is *external consistency*; this means that whenever $T_{i,C}$ completes before $T_{j,D}$ starts, then the apparent serial order must contain $T_{i,C}$ ahead of $T_{j,D}$. A less restrictive property is *session consistency*, which says that the apparent order must not rearrange transactions from the same client, where one completes before the other starts. That is, whenever $T_{i,C}$ completes before $T_{k,C}$ starts, then the apparent serial order must contain $T_{i,C}$ ahead of $T_{k,C}$. But session consistency says nothing about the serialization order of $T_{i,C}$ and $T_{j,D}$ where $C \neq D$.

Replicated Snapshot Isolation. Several prominent single-site DBMS platforms provide isolation for transactions using a multiversion mechanism called Snapshot Isolation. This does not have exactly the properties of Serializability, but it avoids most of the known bad concurrency problems, and it seems to satisfy most application programmers. The key to this isolation level is that when a transaction reads an item, it sees the value which reflects all writes by other transactions which committed before the reading transaction started, but it does not see any effects of concurrent transactions.

When replicating data stored in platforms that offer Snapshot Isolation, a natural strong consistency model is to transparently appear like an unreplicated system running on the same sort of platform. This is the consistency model known as "1-copy-SI" [10]. Here is an example with concurrent client transactions $T_{3,C}$ and $T_{4,D}$, each of which reads two logical data items and increments one of them. The sequence (5) is something that a client could see in 1-copy-SI, but not in a system offering 1-copy-serializable consistency, because the values read are not the same as the serial order $T_{3,C}$ then $T_{4,D}$ (where $T_{4,D}$ would read $y = 21$), nor are they as in $T_{4,D}$ followed by $T_{3,C}$ (where $T_{3,C}$ would read x = 11).

$$\begin{aligned} & b_{4,D}\, r_{4,D}[x, 10]\, r_{4,D}[y, 20]\, b_{3,C}\, r_{3,C}[x, 10] \\ & r_{3,C}[y, 20]\, w_{4,D}[x, 11]\, w_{3,C}[y, 21]\, c_{3,C}\, c_{4,D} \end{aligned} \tag{5}$$

As the definition of Snapshot Isolation says that a read sees the effects of all transactions that commit before the reader's transaction started, this definition implicitly includes an external consistency session property. To allow more efficient replica control algorithms, some researchers have built systems which provide more permissive consistency models. For example, in Generalized Snapshot Isolation [6], for each transaction there is a snapshot-time, which could be somewhat before the transaction starts, and the transaction's reads see exactly the effects of those transactions that committed before the reader's snapshot-time. This allows the strange inversions where a client submits a transaction, learns of its success, and then submits another transaction that does not run in the expected state. Thus, an intermediate consistency model is Strong Session Snapshot Isolation [5], where inversions are allowed between non-concurrent transactions from different clients, but the snapshot-time for a transaction must be later than the commit of any previous transaction submitted by the same client.

Key Applications

1-copy serializability is most often provided through eager or synchronous propagation of updates among machines that are all in a single cluster. Some database engines provide options for replication internally, whereas others rely on replication tools that run along side the DBMS engine. So far, the model of 1-copy serializability with lazy propagation of updates, or the model of replicated snapshot isolation, are offered in research prototypes rather than among commercial products.

Future Directions

The algorithms known for replicated snapshot isolation may be attractive for practical use, since they can offer substantially higher performance than the algorithms for 1-copy serializability, and since programmers seem able to work with snapshot isolation in a single DBMS. Thus, this strong consistency model will probably become more widespread in the future.

Cross-references

► Data Replication

Recommended Reading

1. Attar R., Bernstein P.A., and Goodman N. Site initialization, recovery, and backup in a distributed database system. IEEE Trans. Software Eng., 10(6):645–650, 1984.
2. Bernstein P.A. and Goodman N. Serializability theory for replicated databases. J. Comput. Syst. Sci., 31(3):355–374, 1985.
3. Breitbart Y., Komondoor R., Rastogi R., Seshadri S., and Silberschatz A. Update propagation protocols for replicated databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
4. Chundi P., Rosenkrantz D.J., and Ravi S.S. Deferred updates and data placement in distributed databases. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 469–476.
5. Daudjee K. and Salem K. Lazy database replication with snapshot isolation. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 715–726.
6. Elnikety S., Zwaenepoel W., and Pedone F. Database replication using generalized snapshot isolation. In Proc. 22nd Symp. on Reliable Distributed Syst., 2005, pp. 73–84.
7. Gray J., Helland P., O'Neil P.E., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
8. Herlihy M. A quorum-consensus replication method for abstract data types. ACM Trans. Comput. Syst., 4(1):32–53, 1986.
9. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Middleware based data replication providing snapshot isolation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 419–430.
10. Plattner C. and Alonso G. Ganymed: Scalable replication for transactional web applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004, pp. 155–174.
11. Terry D.B., Demers A.J., Petersen K., Spreitzer M., Theimer M., and Welch B.B. Session guarantees for weakly consistent replicated data. In Proc. Int. Conf. on Parallel and Distributed Information Systems, 1994, pp. 140–149.

Strong Coupling

► Tight Coupling

Strong Memory Consistency

► Strong Consistency Models for Replicated Data

Structural Index

► Structure Indexing

Structural Summary

► Structure Indexing

Structure Indexing

MARIANO P. CONSENS
University of Toronto, Toronto, ON, Canada

Synonyms

Structural index; Structural summary; Path index; Dataguide; Synopsis; Sketch

Definition

Structure indexing creates summaries of the structure present in semi-structured data collections by grouping data items with similar structure, providing a mechanism to index such items. Since semi-structured data models are commonly represented by labeled graphs or trees (the XML data model being a prime example), structural indexes or summaries are naturally described as graphs where nodes represent sets of data items (called extents), and where edges represent structural relationships between the corresponding extents derived from the instance data. A concrete physical index can be created by selecting appropriate data structures to store the graph and the extents.

Structure indexing helps to find data items that satisfy structural constraints in queries by locating nodes in the structural summary graph that satisfy the query conditions (expecting far less summary nodes than data items), and then limiting query evaluation to data items in the relevant extents.

Structural summaries also provide a description of the structure present in the instance. This is in contrast with schemas, which prescribe structures that may or may not occur in an instance, but without giving an indication of the metadata that is actually present in a given collection. Note that it is possible to create a structural summary from an instance even when the instance does not conform to any schema.

Additional information can be attached to summaries (such as statistics related to nodes and relationships, and distributions of values associated with the extents), with applications in selectivity estimation and query optimization.

Historical Background

Structure indexing mechanisms were proposed as soon as the database community turned its attention to the problem of managing semi-structured data. Region Inclusion Graphs (RIG) and Region Order Graphs (ROG) [3] were proposed to help optimize the evaluation of region algebras (see [14]).

Representative Objects (RO) [10] and Dataguides [6] were motivated by a desire to describe the metadata present in semi-structured databases (modeled as labeled graphs), as well as to help in query optimization. The representative object of length 1 (1-RO) coincides with a RIG; both summaries group data items in the instance (nodes or regions, respectively) based on the label of an item. Representative objects of length k (k-RO) group instance nodes by the labels in the incoming paths of length k (or paths of arbitrary length in the case of a full representative object, or FRO). Dataguides create a summary of the path structure of a labeled graph database instance in which every label path starting at the root appears exactly once. Construction of a Dataguide is analogous to the conversion of a non-deterministic finite automaton (describing the language of the labels occurring in the instance) to a deterministic one. This construction is not unique, but a strong Dataguide is defined to be suitable as an index. For arbitrary graph data, Dataguides can in the worse case become exponentially larger than the actual instance. When instances are trees, as is the case for XML documents when links (such as IDREF) are not considered, the Dataguide size is bounded by the size of the instance.

The concept of bisimulation was introduced in the context of structural summaries by the T-index family [9]. In particular, the 1-index defines extents via partitioning the nodes of the instance using labeled bisimulation on the incoming paths. By creating a partition, bisimulation based summaries have a size bounded by the size of the instance. The F&B-Index [7] generalized the notion of bisimulation-based summaries to consider partitions created by both incoming and outgoing paths. AxPRE summaries [4] introduced a language for defining the neighborhood of interest to the bisimulation-based partitioning criteria. Depending on the axis path regular expression (AxPRE) used, all of the previously proposed bisimulation-based summaries can be defined (as well as entirely new ones).

There are also proposals that augment structural summaries with statistical information of the instance for selectivity estimation, including path/branching distribution and value distributions (e.g., XSketch [11], StatiX [5]).

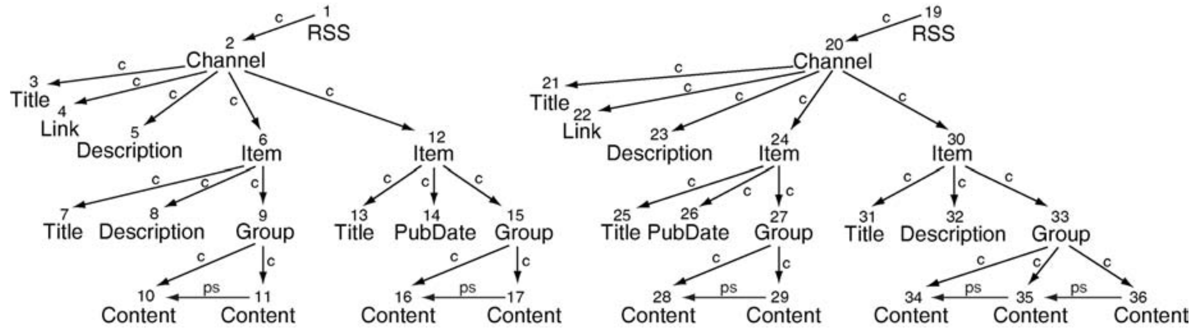
Most of the existing summary proposals define all the extents using the same criteria, hence creating homogeneous summaries. These summaries are based on common element paths (in some cases limited to length *k*), including incoming paths (e.g., representative objects [10], dataguides [6], 1-index [9], ToXin [13], A(k)-index [8]), both incoming and outgoing paths (e.g., F&B-Index [7]), or sequences of outgoing paths (e.g., Skeleton [1]). There are also heterogeneous summaries where summary nodes adapt their criteria to query workloads (APEX [2], D(k)-index [12]), or to statistics from the instance (XSketch [11]), or where the criteria can be given explicitly (AxPRE summaries [4]).

Foundations

A suitable graph-based model for semi-structured instances is introduced below and then partition-based structural summaries are defined. The discussion then generalizes this definition to AxPRE summaries and then presents a lattice with several bisimulation-based summaries in the literature.

Semi-structured Data Example

Consider an example XML instance that consists of two RSS documents, represented as two labeled graphs in Fig. 1. RSS documents are used to encode feeds that publish frequently updated Web content such as news headlines, blog entries, and podcasts. The feeds are organized into *channels*, which in turn contain a list of *items*. A number of elements (such as *title, link, description, pubDate*) can optionally appear within channels and/or items. The items may have *content*,



Structure Indexing. Figure 1. Sample XML instance with two RSS feeds.

appearing within *groups*, that refers to multimedia files (e.g., an audio file in a podcast). The labels in the XML nodes (which are numbered for ease of reference) identify the element, and the labels in the edges the relationship (or *axis* in the XPath data model) between the XML nodes. The edge from node 6 to 7 labeled *c* means that 7 is the child of 6 (or, considering the inverse relationship, that 6 is the parent of 7). Similarly, the edge from node 29–28 labeled *ps* means that 28 is the preceding sibling of 29 (another XPath axis that provides information about the XML document order among siblings).

Axis Graph Definition

An *axis* graph is defined to represent XML instances. The definition can be easily applied to other semi-structure data models where instances are represented by labeled graphs.

An axis graph $\mathcal{A} = (Inst, Axes, Label, \lambda)$ is a structure where *Inst* is a finite set of nodes, *Axes* is a set of binary relations $\{E_1,...,E_n\}$ in $Inst \times Inst$, *Label* is a finite set of node names, and λ is a function that assigns labels in *Label* to nodes in *Inst*. Edges in $\mathcal{A}$ are labeled by the name of the axes relations.

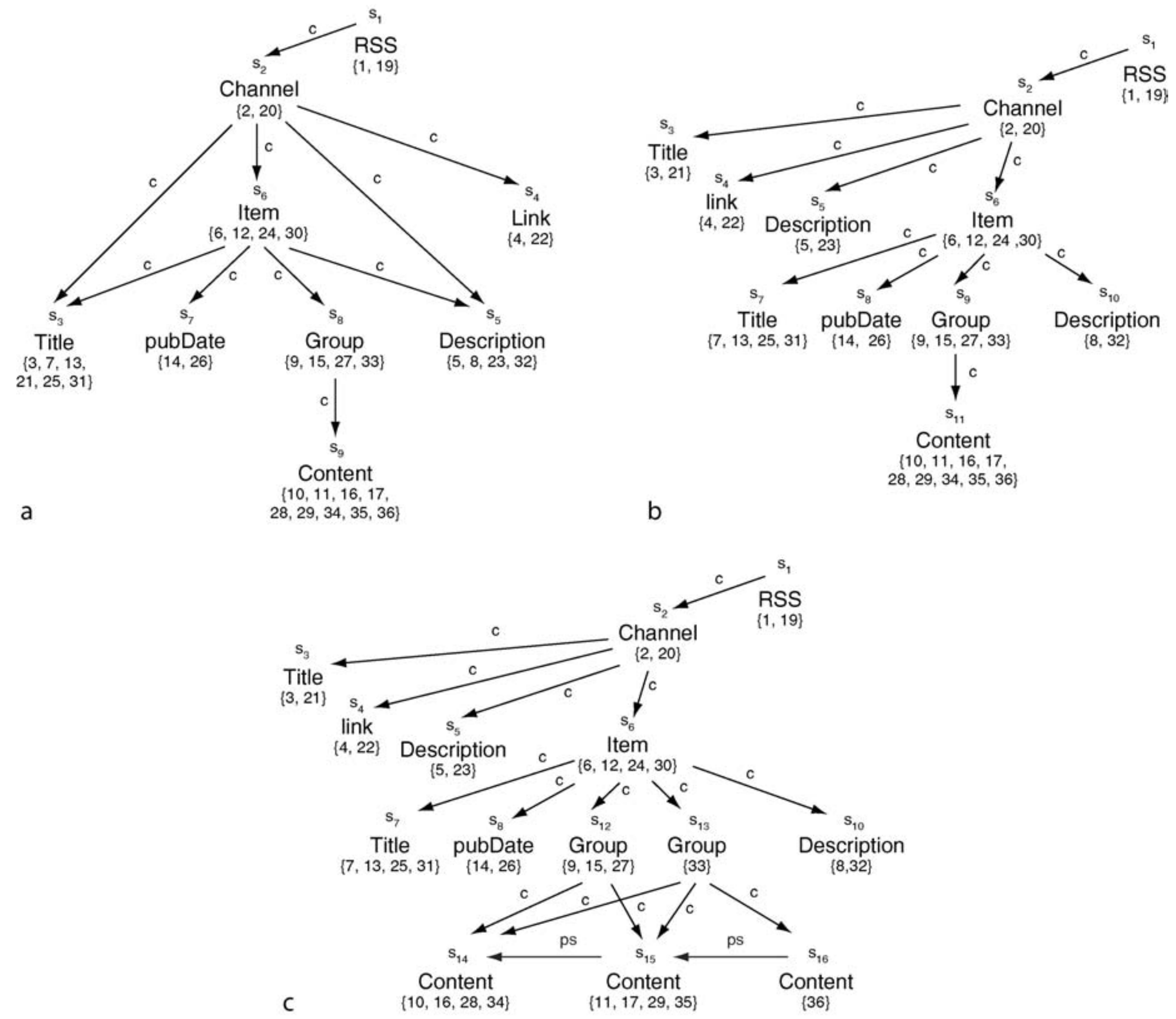
Structural Summary Definition

A structural summary of an axis graph $\mathcal{A}$ is another axis graph $\mathcal{S}(\mathcal{A}) = (Sum, Axes_{\mathcal{S}}, Label, \lambda_{\mathcal{S}})$ where summary nodes correspond to a partition of the nodes in $\mathcal{A}$ and summary edges are induced from the axes in $\mathcal{A}$. Each node $s \in Sum$ with $\lambda_{\mathcal{S}}(s) = l$ corresponds to a subset (called extent) in a partitioning of *Inst*, such that all the nodes in the extent have the same label $l \in Label$. Also, for every edge $E_i(n,m)$ with $n,m \in Inst$ and $E_i \in Axes$ there is $E'_i(n_s,m_s)$ with n_s, $m_s \in Sum$ and $E'_i \in Axes_{\mathcal{S}}$, such that n_s (respectively, m_s) is the summary node that has XML node n (respectively, m) in its extent.

Sample Structural Summaries Figure 2 shows three different structural summaries of the two RSS documents depicted in Fig. 1. The first summary, in Fig. 2a, is constructed by a partition of the XML instance nodes based solely on their labels, and as such there is a single summary node for each label in the instance. As such, the extent of the summary node s_9 labeled *description* consists of the subset {5,8,23,32} of all the XML nodes labeled *description* in the instance. Note that there are two edges labeled *c* incoming into the summary node s_9 since *description* appears both in *channels* and *items*.

The second summary, in Fig. 2b, is constructed from a partition that groups together XML nodes with the same ancestor elements. Note that there are two summary nodes (s_5 and s_{10}) labeled *description*, corresponding the two possible paths of labels to the root (*channel, RSS*, and *item, channel, RSS*). Observe that, for the specific instance in the example, grouping together nodes with just the same parents would produce the same summary. However, grouping together nodes with the same children would produce a different summary (e.g., there would be two summary nodes for *item*, since the *item* nodes in the XML instance would be partitioned into the sets {6,30} and {12,24} each with the same sets of children).

The third summary, in Fig. 2c, contains an heterogeneous summary, where subsets in the partition group XML nodes with different criteria. While most the summary nodes in the figure have extents containing XML nodes that use the same criteria as above (having the same ancestor elements), the summary nodes corresponding to *content* elements are



Structure Indexing. Figure 2. Summaries of the two RSS feeds; (a) label, (b) ancestors, (c) heterogeneous.

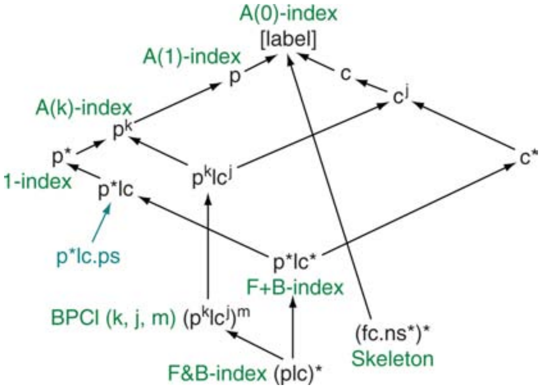
partitioned according to the previous siblings (the edges labeled *ps* in the instance) and the *group* elements are partitioned according to the partition of their *content* children. So there are three summary nodes s_{14},s_{15}, and s_{16} that group the first, second, and third *content* node siblings in the instance, and there are two summary nodes s_{12} and s_{13} corresponding, respectively, to those *group* elements that have a first and second *content* node as children, or those that have a first, second, and third *content* node as children.

AxPRE Summary Definition

An axis path regular expression (AxPRE) is a regular expression on the vocabulary of the names of the axes relations. The AxPRE Neighborhood $\mathcal{N}_{\alpha}(v)$ of an instance node $v \in \mathit{Inst}$ is the subgraph obtained by intersection with the prefix closed finite automaton corresponding to the AxPRE α.

A *labeled bisimulation* between two subgraphs $\mathcal{G}_1$ and $\mathcal{G}_2$ of an axis graph $\mathcal{A}$ is a symmetric relation ≈ such that for all $v \in \mathit{Inst}^{\mathcal{G}_1}$, $w \in \mathit{Inst}^{\mathcal{G}_2}$, $E_i^{\mathcal{G}_1} \in \mathit{Axes}^{\mathcal{G}_1}$, and $E_i^{\mathcal{G}_2} \in \mathit{Axes}^{\mathcal{G}_2}$: *(i)* if $v \approx \mathrm{w}$, then $\lambda(v) = \lambda(\mathrm{w})$; *(ii)* if $v \approx w$, and $\langle v, v' \rangle \in E_i^{\mathcal{G}_1}$, then $\langle w, w' \rangle \in E_i^{\mathcal{G}_2}$ and $v' \approx w'$.

An AxPRE summary is a structural summary where the partition is defined as follows: two nodes v,$w \in \mathit{Inst}$ belong to the same partition block iff there exists a *labeled bisimulation* ≈ between $\mathcal{N}_{\alpha}(v)$ and $\mathcal{N}_{\alpha}(w)$ such that $v \approx w$.



Structure Indexing. Figure 3. Summary lattice.

Sample AxPRE Summaries

The summary in Fig. 2a corresponds to the AxPRE summary with an empty AxPRE, where only condition (i) in the definition of labeled bisimulation applies and the result is a partition of the instance nodes according to their labels only.

The summary in Fig. 2b corresponds to the AxPRE summary with an AxPRE p^*, which is the AxPRE that creates node neighborhoods that consists of all the ancestors of the node. As discussed earlier, for the specific instance in the example, the AxPRE could also be p, but it can not be c.

The summary in Fig. 2c is an heterogeneous summary where most summary nodes have the partition defined according to the AxPRE p^*, but the summary nodes labeled *content* use $p^*|ps^*$ and those labeled *group* use $p^*|c.ps^*$.

Summary Lattice

Figure 3 shows a lattice with relationships among several AxPRE summaries that capture bisimilarity-based proposals mentioned earlier. The lattice represents the partition refinement relationship between the summaries. Each node in the lattice of Fig. 3 corresponds to a homogeneous summary defined by the AxPRE label, with an additional textual label for the corresponding summary name. A node is also used to represent an homogeneous AXPRE summary based on the AxPRE $p^*|c.ps^*$, which applied to only some of the summary nodes in the example in Fig. 2c.

Key Applications

Indexing, metadata description, query processing and optimization, selectivity estimation.

Future Directions

Structure indexing techniques can be extended to support additional data models and their associated query languages, and to further refine their adaptability to different workloads.

Cross-references

► Bisimulation
► Indexing
► Query Processing and Optimization in Object Relational Databases
► Selectivity Estimation
► Semi-structured Data
► Statistical Summaries
► XML
► XPath Data Model
► XPath/XQuery

Recommended Reading

1. Buneman P., Choi B., Fan W., Hutchison R., Mann R., and Viglas S. Vectorizing and querying large XML repositories. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 261–272.
2. Chung C.-W., Min J.-K., and Shim K. APEX: an adaptive path index for XML data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 121–132.
3. Consens M.P. and Milo T. Optimizing queries on files. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 301–312.
4. Consens M.P., Rizzolo F., and Vaisman A.A. AxPRE summaries: exploring the (semi-)structure of XML web collections. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 1519–1521.
5. Freire J., Haritsa J.R., Ramanath M., Roy P., and Simeon J. StatiX: making XML count. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 181–191.
6. Goldman R. and Widom J. Dataguides: enabling query formulation and optimization in semistructured databases. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 436–445.
7. Kaushik R., Bohannon P., Naughton J.F., and Korth H.F. Covering indexes for branching path queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 133–144.
8. Kaushik R., Shenoy P., Bohannon P., and Gudes E. Exploiting local similarity for indexing paths in graph-structured data. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 129–140.
9. Milo T. and Suciu D. Index structures for path expressions. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 277–295.
10. Nestorov S., Ullman J.D., Wiener J.L., and Chawathe S.S. Representative objects: concise representations of semistructured, hierarchial data. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 79–90.
11. Polyzotis N. and Garofalakis M.N. XSketch synopses for XML data graphs. ACM Trans. Database Syst., 31(3):1014–1063, 2006.

12. Qun C., Lim A., and Ong K.W. D(k)-index: an adaptive structural summary for graph-structured data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 134–144.
13. Rizzolo F. and Mendelzon A.O. Indexing XML data with ToXin. In Proc. of Fourth Int. Workshop on the Web and Databases, 2001, pp. 49–54.
14. Young-Lai M. and Tompa F.W. One-pass evaluation of region algebra expressions. Inform. Syst., 28(3):159–168, 2003.

Structure of Truth Values

► Residuated Lattice

Structure Weight

Mounia Lalmas
University of London, London, UK

Definition

In structured text retrieval, the structure of a text component may be used to estimate the relevance of that component. This is done by associating a weight to the structure reflecting its significance when estimating the relevance of the component for a given query.

Key Points

Associating weight to the structure of a component in itself is not new, and several investigations have been reported for whole document retrieval. This entry is concerned with structure weights in the context of structured text retrieval, where the aim is to exploit the document structure to return document components, instead of whole documents.

In structured text retrieval, not all document components will trigger the same user satisfaction when returned as answers to queries. In the context of structured documents mark-up in XML, some document components, i.e., XML elements, may not be appropriate to return because they are too small, or a tag type that does not contain informative content, nested too deep in the document logical structure, or for other reasons. When ranking XML elements, their structure (size, tag type, path, depth, etc.) may prove important. The importance of the element structure is captured through a weight, which can be binary.

Using binary weights means that an element is (value one) or is not (value zero) considered for indexing and retrieval. The decision can be made by looking at the DTD (document type definition) of the collection, past relevance data, and/or the requirements of the application and user scenario. In the selective indexing strategy [3], only elements of types that were found to contain relevant content for previous query sets (relevance data) are considered. Any elements with a length size less than a given threshold can also be ignored.

Weights can be assigned to characteristics of elements, such as length, depth, location in the document logical structure, and so on. For instance, within the language modelling framework, length has been used as a normalization parameter (weight) incorporated through a prior probability in the ranking formula [2].

With statistical approaches, the weights are estimated based on training data, such as past relevance data. The weights can be determined using machine learning, and then used in the ranking function. They can also be directly calculated based on the distribution of element characteristics. For example, in [1], the distribution of tag types is used in a way similar to the binary independence retrieval model (investigating the "presence" of tags in relevant and non-relevant elements) to estimate the element weights.

Cross-references

► Indexing Units
► Logical Structure
► Relationships in Structured Text Retrieval
► XML Retrieval

Recommended Reading

1. Gery M., Largeron C., and Thollard F. Probabilistic document model integrating XML structure. In Proc. 6th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2007, pp. 139–149.
2. Kamps J., de Rijke M., and Sigurbjörnsson B. Length normalization in XML retrieval. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 80–87.
3. Mass Y. and Mandelbrod M. Component ranking and automatic query refinement for XML retrieval. In Advances in XML Information Retrieval, Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, Revised Selected Papers, 2005, pp. 73–84.

Structured Data in Peer-to-Peer Systems

KAI-UWE SATTLER
Technical University of Ilmenau, Ilmenau, Germany

Synonyms

Peer data management; Peer database management; P2P database

Definition

A peer database management system is a peer-to-peer (P2P) system that manages structured data. Each node in such a system maintains data which conforms to user- or application-defined structures and can be accessed and retrieved efficiently. Examples of data structures are relations as set of tuples and hierarchical or tree-organized data such as XML data and documents. In contrast to unstructured data (e.g., text and binary objects) structured data can be retrieved by specifying logical conditions and further processed by operations such as set operations, aggregations and joins.

Historical Background

One of the origins of coordinator-free distributed data management were scalable distributed data structures (SDDS). One of these approaches called LH* [7] is an extension of linear hashing to distributed nodes, and supports key search as well as parallel operations like hash joins and scans.

Distributed hashtables (DHT) [2] for managing key/object pairs in very large and changing networks were introduced around 2001, e.g., Chord, CAN or P-Grid. Based on these DHTs several approaches for supporting more database-like functionality (e.g., multi-attribute queries, join and aggregation queries) have been developed. One of the first systems was PIER [5], other examples are RDFPeers [3] as well as the work by Triantafillou and Pitoura [12].

The idea of exploiting the P2P paradigm for data integration in Peer Data Management Systems (PDMS) was first presented in 2002/2003 by systems like Edutella [8], that uses RDF/RDFS as schema language and an own RDF-based query language, and Piazza [4] which is based on XML and XQuery.

Foundations

A widely accepted classification of P2P systems distinguishes between unstructured and structured systems. In the following, techniques for managing and querying structured data in each of these two classes of systems are described.

Unstructured P2P Systems

In an unstructured P2P system peers do not maintain information about the resources managed by other peers. This means that for answering a query the request has to be forwarded to its neighbors. Despite rather simple approaches with fixed schema (e.g., file-sharing systems) a typical example of an unstructured P2P system is a peer data integration system also called PDMS (Peer Data Management System).

A query in a PDMS can be issued at each peer. The peer processes the query locally but has to forward it (or portions of this query) to the neighbor peers which provide relations relevant to this query. For this purpose, the query is rewritten using the correspondence mappings. Depending on the formalism of mappings this is implemented by query unfolding (in case of GaV) or by techniques for answering queries using views (in case of LaV) known from data integration techniques. As long as the rewritten queries still contain views or subgoals instead of only base relations the query has to be further rewritten by the neighbor peer and to be forwarded to its neighbors. The remaining steps of optimization and execution follow the basic principle of distributed query processing. However, there are several ways for further optimizations, e.g., for query routing by using routing indexes or by exploiting transitivities of the correspondence paths.

Structured P2P Systems

In a structured P2P system each peer maintains information about the resources stored at other peers, e.g., in the form of a routing table. A typical example of a structured P2P system is a distributed hashtable (DHT) – a system where a logical key space is partitioned among all peers in order to manage key-object pairs $\langle k, v \rangle$.

In order to manage structured data comprising more than a key and a value component a mapping between the schema of a database relation or XML document and the $\langle k, v \rangle$ pair is needed. In the following, three possible alternatives are presented.

In the *horizontal* or tuple-based approach, each tuple is identified and inserted into the DHT by a resource identifier (or primary key) *oid*. The remaining attributes A_i with the values v_i are treated as a single object:

$$\langle oid, A_1 : v_1, A_2 : v_2, \ldots, A_n : v_n \rangle \Rightarrow$$
$$\langle h(oid), [A_1 : v_1, A_2 : v_2, \ldots, A_n : v_n] \rangle$$

This data organization scheme allows to retrieve tuples by their resource identifier. Note, that the individual attribute values are still available, but cannot be accessed directly using the DHT lookup feature. In this way, the DHT can be seen as an index structure for the database relation.

If efficient access to the other attributes A_i is needed, too, additional indexes have to be constructed by inserting additional pairs $\langle [A_i : v_i], oid \rangle$ into the DHT. Instead of using the *oid* as value it is also possible to store the whole tuple.

Furthermore, this approach can be easily extended to more than one relation by adding a name space or relation prefix to the resource identifier. An example system following the horizontal approach is the PIER system [5] which is based on the Bamboo DHT.

An alternative approach is a *vertical* data organization where each tuple is represented by a set of triples:

$$\langle oid, A_1 : v_1, A_2 : v_2, \ldots, A_n : v_n \rangle \Rightarrow \{\langle h(oid), A_1, v_1 \rangle,$$
$$\langle h(oid), A_2, v_2 \rangle, \ldots \langle h(oid), A_n, v_n \rangle\}$$

In order to be able to query all attributes each of these triples is inserted into the DHT using the resource identifier, the attribute and eventually the value component as key, i.e., $\langle h(oid), [A_i : v_i] \rangle$, $\langle h(A_i), oid \rangle$, $\langle h(v_i), oid \rangle$. If the DHT supports prefix search, the concatenation of $A_i \# v_i$ can be used as key to allow queries both for A_i as well as $A_i = v_i$. Advantages of this vertical data organization are first, that triples with similar values are stored at the same peer or at least in the neighborhood which simplifies joins and – in combination with an order-preserving hash function – range queries, and similarity queries. Second, there is no fixed schema for a given relation. Users can extend the schema to their needs by simply adding new triples for a given tuple. The disadvantage is the required storage overhead.

This approach is not restricted to relational data. RDF data which can be represented using triples of subject, predicate, and object can be directly mapped to this scheme. Examples of systems following this idea are RDFPeers [3] based on MAAN a multidimensional extension of Chord, and UniStore [6] which is built on top of P-Grid.

Assuming a P2P system for storing and querying XML documents instead of relations a third possible approach is to exploit the idea of *path indexing* [11]. A standard technique for indexing XML documents is suffix indexing of XML paths. Given a document identified by a URI and an XML path P in this document consisting of the elements $P = e_1/e_2/\ldots/e_n$ the following n suffixes can be derived

$$\begin{aligned} s_1 &= e_1/e_2/e_3/\ldots/e_n \\ s_2 &= e_2/e_3/\ldots/e_n \\ s_i &= \ldots \\ s_n &= e_n \end{aligned}$$

Now, based on these subpaths the keys for the DHT are calculated and the following pairs are inserted: $\langle h(s_1), [s_1, URI] \rangle$, $\langle h(s_2), [s_2, URI] \rangle$, etc.

Note, that this requires a DHT supporting prefix search.

An important question affecting the choice of the hash function is how to fragment a relation or document collection. A viable solution has to be found between the two extreme cases:

- Distribute tuples according to their relation identifier, i.e., each relation is completely stored at exactly one peer.
- Distribute tuples according to their resource identifier (oid) such that tuples are partitioned among a set of peers. However, in this case tuples of the same relation or with similar values of indexed attributes should be stored at neighboring peers in order to support range and nearest neighbor queries efficiently.

Appropriate fragmentation schemes are based on space-filling curves [1] and order preserving hash functions.

For querying structured data in a DHT both basic processing strategies *data shipping* and *query shipping* can be used. With data shipping the DHT is used only as a data storage: data are retrieved from the responsible peers which are identified by applying the hash functions to the value in the query predicate similarly to an index lookup or index scan in a classical DBMS.

Obviously, this also works for the path indexing approach, where the keys for the lookup operation are calculated from the path expression given in an XPath query. As an example consider a peer asking a path query *A// B/C/D//E*. By computing the longest subpath containing only the child axes (in this case *B/C/D*), the peers responsible for this key region can be identified and contacted for evaluating the remaining query locally.

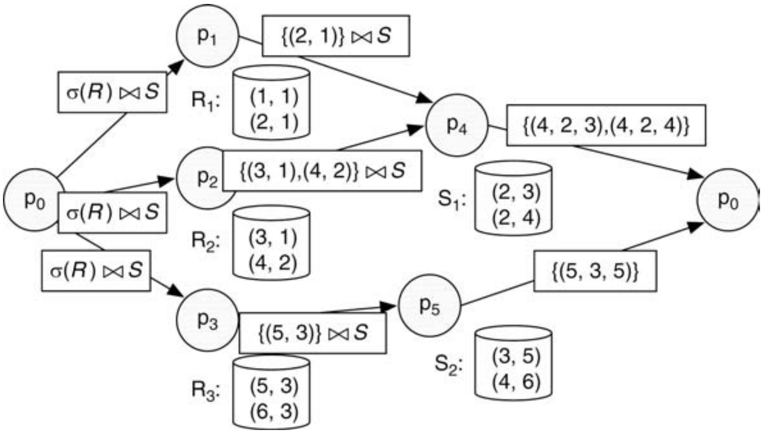
This approach can be extended to process more complex queries including joins. Based on the retrieved tuples of the first relation, the matching tuples of the second relation are retrieved from the DHT by applying the hash function to the join values. However, data shipping becomes inefficient if the query operators are not very selective. In this case, very expensive scans of many peers are needed which is impractical in large-scale systems.

For *query shipping* in a DHT each peer has to provide distributed query processing capabilities rather than only the put/get operations. A query is routed to the peer that is responsible for the data addressed by the operator that is to be executed next. These peers are determined by applying the hash function to the current intermediate results. In this way, the DHT is used both as a hash table for indexing and storing tuples as well as a content-addressable network for routing tuples and/or operators by values.

This idea was exploited in PIER for implementing different join strategies. A first strategy presented in [5] is a variant of the symmetric hash join between two relations R and S. Here, each peer storing tuples of R and S scans its local data and insert these tuples into a temporary table of the DHT. The peers responsible for the key space of this table execute the probing phase by joining the received tuples locally. A second strategy developed in PIER is a fetch matches variant which can be used if one of the relation is already hashed on the join attribute. In this case, the peers storing tuples of the second relation perform a local scan and retrieve the possibly matching tuples using the get operation of the DHT.

Further improvements can be achieved by applying the idea of symmetric semijoin, i.e., perform a local projection on the join attribute of both relations, followed by a symmetric hash join and feed the results into a fetch matches join to retrieve the remaining attribute values of the original relations. Finally, bloom filters can be used, too. Here, bloom filters are created by each peer responsible for R and S and inserted into temporary DHT table. The received filters are combined and sent to all peers storing the opposite relation. The experiments presented in [5], show that these strategies help to reduce the bandwidth consumption and response time particularly in case of low selectivity values.

In case of more complex queries consisting of sequences of operators query shipping may result in multiple instances of the plan that "travel" through the network, because a single query operator might involve tuples from different peers. This approach, which was initially proposed as *mutant query plans* [9] is illustrated in Fig. 1. Given a relation $R(A, B)$ stored at peers p_1, p_2, p_3 and a relation $S(B, C)$ stored at peers p_4, p_5. Peer p_0 submits the query $\sigma_{1<A<6}(R) \bowtie S$. The query is sent to $p_1 ... p_3$ which are identified by applying the hash function to the selection



Structured Data in Peer-to-Peer Systems. Figure 1. Mutant query plans in query shipping.

predicate. These peers evaluate the first part of the query ($\sigma_{1<A<6}(R)$) locally and replace this expression in the plan by the intermediate result. The modified plan is sent to the *S*-peers which are identified by applying the hash function on the *B* values of the intermediate result. Finally, p_4 and p_5 evaluate the remaining parts of the query and send the result back to the initiator.

The benefits of query shipping are exploiting computing resources of the peers as well as avoiding transfer of large datasets. However, query planning and execution is more difficult because the state of a processed query is spread over multiple peers.

Key Applications

A major application for DHT-based P2P database systems is public data management where information of a general interest, its structure and semantics is controlled by a large number of participants. Furthermore, the costs for providing the infrastructure should often be shared by the users in a fair manner. Examples of such applications are the management of public datasets in e-Sciences, e.g., genome data or data in astronomy, metadata and index data for the Semantic Web as well as specialized search engines, naming and directory services as well as social applications such as file/picture sharing, recommender systems or friend-of-a-friend networks. Note, that in these applications it is often not necessary to store the actual data itself in the DHT, but instead metadata or index data required for answering queries are publically managed.

The main application of unstructured peer data management system is data integration in large-scale, loosely-coupled scenarios.

Future Directions

Managing and querying structured data in P2P systems is a relatively new research area which raises several new challenges to established techniques, e.g., known from distributed database systems.

The first challenge is *scalability*, meaning the support of efficient query processing in networks of ten thousands or more nodes. Most of the research systems presented so far are based on simulation environments or a relatively small number of peers, e.g., in PlanetLab or similar platforms.

A second problem in P2P systems is the dynamic of the network (joining and leaving peers) as well as the unreliability of peers. Most existing approaches are best effort solutions which are unable to give guarantees wrt. result completeness, freshness or response time. Thus, an open issue is to estimate completeness of results in case of partial answers or to guarantee a certain *quality of service*.

Finally, in large P2P systems where no peer knows all other peers in the network *trustworthiness* are a further challenge. Particularly, if data are redistributed to other nodes in the network the original data producer wants to make sure that its data are not manipulated by the hosting peer. Furthermore, in order to achieve a fair balancing of load and avoiding overloaded peers the problem of rejecting requests and free riding by malicious peers has to be addressed, e.g., by incentive mechanisms.

Cross-references

▶ Distributed Join
▶ Distributed Query Processing
▶ P2P Data Integration
▶ P2P Overlay Networks

Recommended Reading

1. Andrzejak A. and Xu Z. Scalable, Efficient range queries for grid information services. In Proc. of Second IEEE Int. Conf. on Peer to Peer Computing, 2002, pp. 33–40.
2. Balakrishnan H., Kaashoek M.F., Karger D., Morris R., and Stoica I. Looking up Data in P2P Systems. Commun. ACM, 46(2):43–48, 2003.
3. Cai M. and Frank M. RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In Proc. 12th Int. World Wide Web Conference, 2004, pp. 650–657.
4. Halevy A., Ives Z., Mork P., and Tatarinov I. Piazza: data management infrastructure for semantic web applications. In Proc. 12th Int. World Wide Web Conference, 2003, pp. 556–567.
5. Huebsch R., Hellerstein J.M., Lanham N., Thau Loo B., Shenker S., and Stoica I. Querying the Internet with PIER. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 321–332.
6. Karnstedt M., Sattler K., Richtarsky M., Müller J., Hauswirth M., Schmidt R., and John R. UniStore: Querying a DHT-based Universal Storage. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 1503–1504.
7. Litwin W., Neimat M.-A., and Schneider D. LH* – a scalable, distributed data structure. ACM Trans. Database Syst., 21(4): 480–525, 1996.
8. Nejdl W., Wolf B., Qu C., Decker S., Sintek M., Naeve A., Nilsson M., Palmer M., and Risch T. Edutella: a P2P networking infrastructure based on RDF. In Proc. 11th Int. World Wide Web Conference, 2002, pp. 604–615.
9. Papadimos V. and Maier D. Mutant query plans. Inform. Software Technol., 44(4):197–206, 2002.

10. Risson J. and Moors T. Survey of Research towards Robust Peer-to-Peer Networks: Search methods. Comput. Networks, 50(17): 3485–3521, 2006.
11. Skobeltsyn G., Hauswirth M., and Aberer K. Efficient processing of XPath queries with structured overlay networks. In Proc. Int. Conf. on Cooperative Inf. Syst., 2005, pp. 1243–1260.
12. Triantafillou P. and Pitoura T. Towards a unifying framework for complex query processing over structured peer-to-peer data networks. In Proc. of Int. Workshop on Databases, Information Systems, and Peer-to-Peer Computing, 2003, pp. 169–180.

Structured Document Retrieval

Mounia Lalmas[1], Ricardo Baeza-Yates[2]
[1]University of London, London, UK
[2]Yahoo! Research, Barcelona, Spain

Synonyms

Structured text retrieval; Querying semi-structured data; Passage retrieval; XML retrieval; Focused retrieval

Definition

Structured document retrieval is concerned with the retrieval of document fragments. The structure of the document, whether explicitly provided by a mark-up language or derived, is exploited to determine the most relevant document fragments to return as answers to a given query. The identified most relevant document fragments can themselves be used to determine the most relevant documents to return as answers to the given query.

Key Points

The aim of this entry is to clarify different terminologies that have been used to refer to or are strongly related to structured retrieval and semi-structured data.

The term "structured document retrieval," which was introduced in the early to mid 1990s in the information retrieval community, refers to "passage retrieval" and "structured text retrieval." In passage retrieval, documents are first decomposed into passages (e.g., fixed-size text-windows of words, fixed discourses such as paragraphs, or topic segments through the application of a topic segmentation algorithm). Passages could themselves be retrieved as answers to a query, or be used to rank documents as answers to the query.

Structured text retrieval is concerned with the development of models for querying and retrieving from structured text, where the structure is usually encoded with the use of mark-up languages, such as SGML, and now predominantly XML. Indeed, text documents often display structural information. For example, a scientific article will have a so-called logical structure, such as an abstract, several sections, and subsections, each of which is composed of paragraphs. A book will have a so-called layout structure, such as pages and columns.

Structured text retrieval is to be contrasted to traditional text retrieval, where the latter is concerned with the retrieval of unstructured text – so-called "raw text" or "flat text." The use of the term "structured" in "structured text retrieval" is there to emphasize the interest in the structure. Furthermore, structured text retrieval aims to exploit the available structural information to return text fragments (e.g., XML elements) as opposed to entire text documents.

The term "semi-structured" comes mainly from the database community. Traditional database technologies, such as relational databases, have been concerned with the querying and retrieval of highly structured data (e.g., from a student table, find the names and addresses of those with a grade over 80 in a particular subject). Text documents marked-up, for instance, in XML are made of a mixture of highly structured components (e.g., year, author name) typical of database records, and loosely structured components (e.g., abstract, section). Database technologies are being extended to query and retrieve such loosely structured components, called semi-structured data. Databases that support this kind of data, mainly in the form of text with mark-up, are referred to as semi-structured databases, to emphasize the loose structure of the data and use "querying data" instead of "data retrieval."

From a terminology point of view, structured text retrieval and querying semi-structured data, in terms of end goals, are the same. The difference comes from the fact that in information retrieval, the structure is added, and in database, the structure is loosened. It should, however, be pointed out that research in information retrieval and databases with respect to accessing structured text (or semi-structured data in the form of text) have been concerned, because of historical reasons, with different aspects of the access process, e.g., ranking in information retrieval versus efficiency in databases. Nowadays, there is a convergence trend between the two areas (e.g., [1]).

In the late 1990s, the interest in structured document retrieval grew significantly due to the introduction of XML in 1998, which has now became the de-facto format standard for structured documents (or structured text, semi-structured data). Research on XML retrieval was further boosted with the set-up of INEX in 2002, the Initiative for the Evaluation of XML Retrieval, which allowed researchers to compare and discuss the effectiveness of models specifically developed for XML retrieval [2]. Nowadays, XML retrieval is almost a synonym for structured document retrieval, structured text retrieval, and querying semi-structured data.

Structured document retrieval, passage retrieval, structured text retrieval, querying semi-structured data, XML retrieval, all belong to what has recently been called "focused retrieval" [3]. Focused retrieval is concerned with returning the most focused results to a given query. Such focused results include passages, XML elements, and factoid answers (e.g., London being the capital of the UK).

Cross-references

- ► Document Databases
- ► INitiative for the Evaluation of XML Retrieval
- ► Integrated DB&IR Semi-Structured Text Retrieval
- ► Semi-Structured Data
- ► Structured Text Retrieval Models
- ► XML Retrieval

Recommended Reading

1. Amer-Yahia S., Case P., Rölleke T., Shanmugasundaram J., and Weikum G. In Report on the DB/IR panel at SIGMOD 2005. ACM SIGMOD Rec., 34(4):71–74, 2005.
2. Kazai G., Gövert N., Lalmas M., and Fuhr N. The INEX Evaluation Initiative. In Intelligent Search on XML Data, Applications, Languages, Models, Implementations, and Benchmarks. Springer, New York, NY, 2003, pp. 279–293.
3. Trotman A., Geva S., and Kamps J. Report on the SIGIR 2007 workshop on focused retrieval. SIGIR Forum, 41(2):97–103, 2007.

Structured Query Language

- ► SQL

Structured Text Retrieval

- ► Structured Document Retrieval
- ► XML Retrieval

Structured Text Retrieval Models

Djoerd Hiemstra[1], Ricardo Baeza-Yates[2]

[1]University of Twente, Enschede, The Netherlands

[2]Yahoo! Research, Barcelona, Spain

Synonyms

Retrieval Models for Text Databases

Definition

Structured text retrieval models provide a formal definition or mathematical framework for querying semi-structured textual databases. A textual database contains both content and structure. The content is the text itself, and the structure divides the database into separate textual parts and relates those textual parts by some criterion. Often, textual databases can be represented as marked up text, for instance as XML, where the XML elements define the structure on the text content. Retrieval models for textual databases should comprise of three parts: (i) a model of the text, (ii) a model of the structure, and (iii) a query language [4]: The model of the text defines a tokenization into words or other semantic units, as well as stop words, stemming, synonyms, etc. The model of the structure defines parts of the text, typically a contiguous portion of the text called element, region, or segment, which is defined on top of the text model's word tokens. The query language typically defines a number of operators on content and structure, such as set operators and operators like "containing" and "contained-by" to model relations between content and structure, as well as relations between the structural elements themselves. Using such a query language, the (expert) user can, for instance, formulate requests like "*I want a paragraph discussing formal models near to a table discussing the differences between databases and information retrieval.*" Here, "formal models" and "differences between databases and information retrieval" should match the content that needs to be retrieved from the database, whereas "paragraph" and "table" refer to structural constraints on the units to retrieve. The features, structuring power, and the expressiveness of the query languages of several models for structured text retrieval are discussed below.

Historical Background

The STAIRS system (Storage and Information Retrieval System), which was developed at IBM in the late 1950's, allowed querying of both content and structure. Much like today's On-line Public Access Catalogues, it was used to store bibliographic data in records with fields such as *keywords* and *title*, providing structured search, but no overlapping or hierarchical structures nor full text search. At the end of the 1980s, researchers at the University of Waterloo in Canada persued database support for the creation of an electronic version of the Oxford English Dictionary. This resulted in a number of models for querying and manipulating content and hierarchical structure such as the parsed strings model [10], PAT expressions [15], the containment model [5] and generalized concordance lists model [7]. Similar approaches were developed elsewhere, such as the proximal nodes model [13] and the nested region model [11]. The interest in structured text retrieval models has grown since the introduction of XML in 1998, and the emergence of standard data retrieval query languages for XML data. One might argue that the structured text retrieval approaches such as PAT expressions and proximal nodes mentioned above are predecessors of XPath. The success of XML in turn has influenced the work on structured retrieval models: XIRQL was proposed in 2000 [9] as an information retrieval extension of XML query languages (XIRQL is an extension of XQL, a predecessor of XPath). More recently, in 2004, NEXI and XQuery and XPath Full-Text have been proposed as query languages for structured text retrieval, as well as examples of structured text retrieval models for the respective query languages [2,12].

Foundations

There are several models of structured text retrieval. Since there is no consensus on how to structure a textual database, this entry addresses several modeling decisions following the taxonomy presented in [4]. The entry only addresses models for text databases, and not text retrieval models in relational databases as for instance provided by SQL/MM. Due to the success of XML, today XML retrieval is almost a synonym for structured text retrieval, although XML retrieval only addresses the explicit, single hierarchy case below.

Explicit vs. Implicit Structure

Most models use *explicit structure*, i.e., they define unambiguously what parts of the textual database are for instance "sections." These models require the database to be structured explicitly and unambiguously, or using terminology from markup languages: the models require the database to be well-formed. This allows easy modeling of nested regions, and powerful structural relationships such as the direct ancestor relationship (i.e., child and parent axis in XPath). The following query might be used in an explicit structure approach to retrieve sections that contain the word "databases":

```
section CONTAINING "databases"
```

Explicit structure is assumed by amongst others the proximal nodes model [13] and the full match model [2]. In systems that use *implicit structure*, however, structure is not explicitly distinguished from content. In these approaches the database is modeled as a sequence of tokens without distinguishing a word token from a markup token. A structural element should, therefore, be constructed at runtime by looking up the opening markup tokens, the closing tokens, and to return those regions starting with a opening token and ending with a closing token. The query above would then be formulated as [11] (here, the operator ". ." would be pronounced as "following"):

```
("<section>" .. "</section>") CONTAINING "databases"
```

So, the `section` element only exists at querying time. Semantically, the query is not different from a content-only query. For instance the query `("all" .. "equal") CONTAINING "created"` retrieves regions that start with the word "all," that end with the word "equal" and that contain the word "created," matching for instance the phrase "all men are created equal." Nested elements, or unbalanced tags are handled differently by several approaches. In the Generalized Concordance Lists (GCL) approach [7], nested sections will not be recognized by the system (instead two partially overlapped sections will be returned). In thenested region algebra approach [11] nested elements are returned properly. The approach is implemented as sgrep (structured grep) (http://www.cs.helsinki.fi/~jjaakkol/sgrep.html). The GCL approach was recently implemented in a research system called Wumpus (http://www.wumpus-search.org).

Static vs. Dynamic Structure

The use of implicit structure also implies the use of *dynamic structure*. A system that uses dynamic structure allows operations that define new elements or regions, i.e., elements or regions that were not previously in the database. In XQuery, this is done by element construction, but in some approaches dynamic structure is a natural consequence of the model. As an early example of dynamic structure consider the following bibliographic entry:

John Doe, "Crime," *Police* 6, 2028.

The entry is explicitly structured by the following grammar that functions as a database schema [10]:

```
entry   := author ',' title ',' journal ',' year '.' ;
author  := text ;
title   := ' "' text ' "' ;
journal := text digit+ ;
year    := digit digit digit digit;
text    := ( letter | ' ' ) + ;
```

A valid database instance contains data that conforms to the grammar. The instance takes the form of a parsed string or "p-string." Note that the schema does not distinguish the author's first name(s) from his surname, but this might be done at query time by introducing a small grammar fragment NameG that parses the author strings into given names and surnames:

```
NameG := { name := ( givenname ' ' )+ surname ;
givenname := letter + ;
surname := letter + ; }
```

The p-strings model provides a simple query language for adding additional grammar fragments. Suppose the bibliographic entry above is E, then the following query returns a p-string containing the author element with given name and surname explicitly identified.

(author *in* E) *reparsed by* NameG

The construct might be used to search for all authors with surname "Doe" that wrote a journal paper that mentions "grammar" in the title. The p-strings model uses regular expression matching as a core language primitive, and as such dynamic structure is more easily added than in for instance XQuery or XQuery Full-Text.

Single Hierarchy vs. Multiple Hierarchies

Although some fielded search methods use a flat structure to model text, the approaches considered here assume a hierarchical structure of the text database. The systems that use implicit structure introduced above assume a single hierarchy. Interestingly, many approaches assume multiple structural hierarchies on the same textual database. Each hierarchy might serve a different purpose. For instance, one hierarchy might represent the logical structure of the text, dividing it in chapters, sections, subsections, etc., whereas a second hierarchy might represent the lay-out structure in columns and pages; and a third layer might represent the results of a part-of-speech tagger, etc. Inside a single hierarchy, the structural elements are either disjoint or nested inside each other, but across hierarchies elements may partially overlap, i.e., a subsection might start half way a page, and end on the following page.

Some approaches relate single views in one query [13]. An interesting approach is suggested by Alink [1], who introduces additional XPath steps (select-narrow and select-wide) that navigate from one hierarchy to another. For instance, the following XQuery Full-Text-like query fragment navigates from the paragraph elements to another hierarchy with a Verb element that contains "killed," and to a hierarchy with a person element that contains "Abraham Lincoln":

```
$doc//paragraph[./select-narrow::Verb
ftcontains "killed" and ./select-narrow::
person ftcontains "Abraham Lincoln"]
```

The need for multiple hierarchies is for instance, addressed in the containment model [5], and the proximal nodes model [13]. In several publications, the hierarchies are called "stand-off annotation" or "offset annotation" to stress that the structural information (or annotations) are modeled separately from the textual data.

Exact Matching vs. Ranking

Many of the early structured text retrieval models do not consider ranked retrieval results, or if they do only as an afterthought, i.e., by ranking the retrieval results using a text-only query disregarding the structural conditions in the query [5]. A simple but powerful way to take the structure of the results into account is to apply a standard information retrieval model to the retrieved content, and then propagate element scores or aggregate term weights based on the text structure. In several of these approaches to ranking, propagation or aggregation is guided by weighting the paths to elements by so-called augmentation weights [9] or interpolation parameters [14], to model for instance that a title element is more likely to contain important information than a bibliography element. Instead of propagating or aggregating the scores from the leaf nodes, *algebraic* approaches include the ranking functionality inside each operator of the query language [2,12]. Ranking might also include relaxation of the query's structural conditions, for instance by relaxing complex queries step-wise to simpler queries [3].

In 2002, Fuhr and Lalmas [8] organized the first workshop of the Initiative for the Evaluation of XML Retrieval. The goal of INEX is to evaluate the quality of the retrieved results, and as such the quality of the ranking provided by the system taking both content and structure into account. The initiative provides a large testbed, consisting of XML documents, queries and relevance judgments on the data, where the relevance judgments are human judgments that define if an XML element is relevant to the query or not. With XML databases and extensions of XML query languages becoming a de-facto standard for structured text retrieval, ranking is one of the main remaining research challenges.

Key Applications

Systems based on structured text retrieval models can be applied to any problem that involves semi-structured text databases. Key applications of the approaches described in this section include: Managing and searching electronic dictionaries such as the Oxford English Dictionary [10,15], managing and searching electronic journals such as the journals of the IEEE [12,8,6], searching stageplays such as the collected works of William Shakespeare [7], and searching hard drives for digital forensics [1].

Cross-references

- ▶ Aggregation-based Structured Text Retrieval
- ▶ Information Retrieval Models
- ▶ NEXI
- ▶ Propagation-based Structured Text Retrieval
- ▶ XPath/XQuery
- ▶ XQuery Full-Text

Recommended Reading

1. Alink W. XIRAF: an XML information retrieval approach to digital forensics. Master's thesis, University of Twente, 2005.
2. Amer-Yahia S., Botev C., and Shanmugasundaram J. TeXQuery: a full-text search extension to XQuery. In Proc. 12th Int. World Wide Web Conference, 2004.
3. Amer-Yahia S., Lakshmanan L.V.S. and Pandit S. FleXPath: flexible structure and full-text querying for XML. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
4. Baeza-Yates R.A. and Navarro G. Integrating contents and structure in text retrieval. ACM SIGMOD Rec., 25(1):67–79, 1996.
5. Burkowski F.J. Retrieval activities in a database consisting of heterogeneous collections of structured text. In Proc. 15th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1992, pp. 112–124.
6. Carmel D., Maarek Y.S., Mandelbrod M., Mass Y., and Soffer A. Searching XML documents via XML fragments. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 151–158.
7. Clarke C.L.A., Cormack G.V., and Burkowski F.J. An algebra for structured text search and a framework for its implementation. Comput. J., 38:43–56, 1995.
8. Fuhr N., Gövert N., Kazai G., and Lalmas M. (eds.). In Proc. 1st Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2002.
9. Fuhr N. and Grossjohann K. XIRQL: a query language for information retrieval in XML. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 172–180.
10. Gonnet G.H. and Tompa F.W. Mind your grammar: a new approach to modelling text. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 339–346.
11. Jaakkola J. and Kilpeläinen P. Nested text-region algebra. Technical report, University of Helsinki, 1999.
12. Mihajlovic V., Blok H.E., Hiemstra D., and Apers P.M.G. Score region algebra: building a transparent XML-IR database. In Proc. Int. Conf. on Information and Knowledge Management, 2005, pp. 12–19.
13. Navarro G. and Baeza-Yates R.A. Proximal nodes: a model to query document databases by content and structure. ACM Trans. Inf. Syst., 15:400–435, 1997.
14. Ogilvie P. and Callan J. Hierarchical language models for XML component retrieval. In Advances in XML Information Retrieval, Lecture Notes in Computer Science 3493. Springer, 2005, pp. 224–237.
15. Salminen A. and Tompa F.W. PAT expressions: an algebra for text search. In Proc. COMPLEX 92, 1992, pp. 309–332.

Structured Text Retrieval Tasks

▶ Presenting Structured Text Retrieval Results

Subject Spaces

HANS-ARNO JACOBSEN
University of Toronto, Toronto, Ontario, Canada

Definition

Subject spaces are a model to formalize publish/subscribe-style interactions and generalize the publish/subscribe concept. Subject spaces subsume existing publish/subscribe models, such as the channel-based, the topic-based, the type-based, and the content-based publish/subscribe models. Subject spaces go beyond these models by permitting the treatment of publications and subscriptions symmetrically, extending publications to also include expressive filter predicates, introducing the notion of selective publishing, interpreting publications and subscriptions as either stateless or stateful, and generalizing matching to encompass a wide range of possible matching semantics.

Key Points

The subject spaces model is a unifying formal framework to specify, describe and analyze the publish/subscribe concept. Subject spaces encompass existing publish/subscribe models and allow the modeling of new aspects of publish/subscribe-style interactions. Informally, a *subject space* is the set of values a publisher can publish and a subscriber can subscribe to. The plurality of *spaces* refers to all sets of values, possibly overlapping, that publishers and subscribers publish and subscribe to in one particular instantiation of the model, respectively.

Subject spaces treat publications and subscriptions as symmetric constructs. That is, a publication must not be distinguished from a subscription and both are equally expressive. Thus, publications also contain predicates, such as range predicates, equality predicates, non-equality predicates, and string predicates found only in subscriptions in other models. The implications are that in subject spaces, both publications and subscriptions specify sets of values that can be interpreted as regions in space. Traditionally, only subscriptions specify regions in space, while publications are points in space.

In contrast to the content-based model, in subject spaces publications and subscriptions consist of predicates and attribute-value pairs, both of which are optional allowing one to model the respective counterpart in the content-based model.

Moreover, subject spaces treat publications and subscriptions as stateful entities. A publication, once published, is persisted in its associated subject space until explicitly revoked. Subsequent publish operations are updates to the previously published value. A sequence of publish operations can be interpreted as moving the published value through space. Subscriptions are interpreted in the same fashion.

Subject spaces generalize the matching of publications against subscriptions. Since both contain predicates and attribute-value pairs, a match between a publication and a subscription exists, if the attribute-value pairs of the publication satisfy the predicates of the subscription and the attribute-value pairs of the subscription satisfy the predicates of the publication. In addition, the interpretation of publications and subscriptions as regions in space enables a generalization of matching to encompass additional matching semantics such as overlap, containment, or closeness of publication and subscription regions in the space.

The symmetric treatment of publications and subscriptions enables a notion of *selective publishing*, whereby a publisher specifies predicates to declaratively specify a subset of potential receivers of published messages. This is in contrast to content-based publish/subscribe, where a publication is delivered to all subscribers with matching subscriptions. The symmetric treatment of publications and subscriptions further extends the conventional interpretation of publication, as mere data points, to ranges, regions in space, or other structures. This increases the expressiveness of the subject spaces model substantially, as opposed to other publish/subscribe models.

The state-persistent nature of publications and subscriptions avoids the problem of redundant notifications. This is because subject spaces distinguish between inserting a publication and updating the previously published value. A match occurs if the publication satisfies the subscription, but not at each update of the published values. Existing content-based publish/subscribe systems cannot model this difference and are therefore susceptible to the redundant notification problem. Subject spaces can model both state-persistent and stateless publish/subscribe. The subject spaces model can be used to formalize and model publish/subscribe-style interactions. Applications of concrete realizations of the subject spaces model are similar to applications of publish/subscribe. The increased expressiveness offered by subject spaces will likely enable new applications. For a more detailed treatment of the subject see for example the work by Leung and Jacobsen [1,2].

Cross-references

- ▶ Channel-Based Publish/Subscribe
- ▶ Content-Based Publish/Subscribe
- ▶ Publish/Subscribe
- ▶ State-Based Publish/Subscribe
- ▶ Topic-Based Publish/Subscribe
- ▶ Type-Based Publish/Subscribe

Recommended Reading

1. Leung H.K.Y. Subject space: a state-persistent model for publish/subscribe systems. In Proc. Conf. of the IBM Centre for Advanced Studies on Collaborative Research, 2002, p. 7.
2. Leung H.K.Y. and Jacobsen H.-A. Efficient matching for state-persistent publish/subscribe systems. In Proc. Conf. of the IBM Centre for Advanced Studies on Collaborative Research, 2003, pp. 182–196.

Subject-based Publish/Subscribe

▶ Topic-Based Publish/Subscribe

Subspace Clustering Techniques

Peer Kröger, Arthur Zimek
Ludwig-Maximilians University of Munich, Munich, Germany

Synonyms

Projected clustering; Oriented clustering; Correlation clustering; Bi-clustering; Co-clustering; Pattern based clustering

Definition

Cluster analysis aims at finding a set of subsets (i.e., a clustering) of a data set. A meaningful clustering reflects a natural grouping of the data. In high-dimensional data, irrelevant attributes and correlated attributes make any natural grouping hardly detectable. Specialized techniques aim at finding clusters in subspaces of a high-dimensional data space.

Historical Background

While different weighting of attributes was in use since clusters were derived by hand, the problem of finding a cluster based on a subset of attributes and a specialized solution was first described in 1972 by Hartigan [8]. However, triggered by modern capabilities of massive acquisition of high-dimensional data in many scientific and economic domains, and the first general approaches to the problem [2–4], research did not focus on the problem until 1998. An illustrative problem description and sketches of some earlier algorithms can be found in [12]. The more special topic of pattern-based clustering is covered in [11]. A general up-to-date overview is presented in [10].

Foundations

Different Challenges: The "Curse of Dimensionality"

High-dimensional data confronts cluster analysis with several problems. A bundle of problems is commonly addressed as the "curse of dimensionality". Aspects of this "curse" most relevant to the clustering problem are: (i) In general, any optimization problem becomes increasingly difficult with an increasing number of variables (attributes) [5]. (ii) The relative contrast of the farthest point and the nearest point converges to 0 with increasing data dimensionality [6,9], i.e., the discrimination between the nearest and the farthest neighbor becomes rather poor in high dimensional data spaces. (iii) Capabilities of automated data acquisition in many application domains leads to the collection of as many features as possible in the expectation that many of these features may sometimes provide useful insights. So, for the task at hand, in many problems there exist many irrelevant attributes in a data set. Since groups of data are defined by some of the attributes only, the remaining irrelevant attributes ("noise") may heavily interfere with the efforts to find these groups. (iv) Similarly, in a data set containing many attributes, some attributes will most probably exhibit correlations among each other (in varying complexity).

Many approaches try to alleviate the "curse of dimensionality" by applying feature reduction methods prior to cluster analysis. However, the second main challenge for cluster analysis of high dimensional data is the possibility, and even high probability, that different subsets or combinations of attributes may be relevant for different clusters. Thus, a global feature selection or dimensionality reduction method cannot be applied. Rather, it becomes an intrinsic problem of the clustering approach to find the relevant subspaces and to find clusters in these relevant subspaces. Furthermore, although correlation among attributes is often the basis for a dimension reduction, for many application domains it is a main part of the interesting information what correlations exist among which attributes for which subset of objects. As a consequence of this second challenge, the first challenge (i.e., the "curse of dimensionality") generally cannot be alleviated for clustering high dimensional data.

Different Solutions: Categories of Subspace Clustering Techniques

Subspace clustering techniques can be divided into three main families. In view of the challenges sketched above, any arbitrarily oriented subspace may be interesting for a subspace clustering approach. The most general techniques ("(arbitrarily) oriented clustering", "correlation clustering" (Note that the name "correlation clustering" relates to a different problem within the machine learning community.)) tackle this infinite search space. Yet most of the research in this field assumes the search space to be restricted to axis-parallel subspaces. Since the search space of all possible axis-parallel subspaces of a d-dimensional data space is still in $O(2^d)$, different search strategies and heuristics are implemented. Axis-parallel approaches mainly split into "subspace clustering" and "projected clustering". In between these two main fields a group of approaches is known as "pattern-based clustering" (also: "biclustering" or "co-clustering"). For these approaches, the search space is not necessarily restricted to axis-parallel subspaces, but on the other hand does not contain all arbitrarily oriented subspaces. The restrictions on the search space differ substantially between different approaches in this group.

Axis-Parallel Subspaces

To navigate through the search space of all possible axis-parallel subspaces and to find clusters in subspaces, mainly two strategies are implemented: the top-down approach and the bottom-up approach.

Following the top-down approach, an algorithm derives a cluster approximately based on the full-dimensional space, and refines the cluster by adapting the corresponding subspace based on the current selection of points. This means, a lower dimensional projection is sought for where the (iteratively refined) set of points clusters best. Thus, algorithms pursuing this approach are called "projected clustering algorithms" and, usually, assign each point to at most one subspace cluster. The first approach of this category is proposed in [2].

Bottom-up approaches start by single dimensions and search primarily for all interesting subspaces (i.e., subspaces containing clusters) as combinations of lower dimensional interesting subspaces (often this combination is translated to the frequent item set problem and, thus, based on the A priori property). Most of these approaches are therefore "subspace clustering algorithms", and can usually assign one point to different clusters simultaneously (i.e., subspace clusters may overlap). Their aim is to find all clusters in all subspaces. There are also "hybrid algorithms" following the projected clustering approach but allowing points to belong to multiple clusters simultaneously or, on the other hand, following the subspace clustering approach but not computing all clusters in all subspaces. The first approach in this category is proposed in [4].

In summary, approaches to axis-parallel subspace clustering handle the problem of irrelevant attributes (aspect (iii) of the "curse of dimensionality"). Bottom-up-approaches, additionally, tackle mostly the problem of poor discrimination of nearest and farthest neighbor (aspect (ii)).

Pattern-Based Clustering

Pattern-based clustering algorithms seek subsets of objects exhibiting a certain pattern on a subset of attributes. In the most-spread algorithms, this pattern is an additive model of the cluster, meaning, each attribute value within a cluster and within the relevant subset of attributes is given by the sum of a cluster mean value, and an adjustment value for the current object and an adjustment value for the current attribute. In general, covering a cluster with such an additive model is possible if the contributing attributes exhibit a simple linear positive correlation among each other. This excludes negative or complex correlations, thus restricting the general search space. Cluster objects reside sparsely on hyperplanes parallel to the irrelevant axes. Projected onto the relevant subspace, the clusters appear as increasing one-dimensional lines. In comparison to axis-parallel approaches, the generalization consists mainly in allowing the axis-parallel hyperplane to be sparse. Also the cluster in the projection subspace may remain sparse. The unifying property of all cluster members is the common pattern. This model has basically been introduced in [7].

Allowing sparseness in the spatial patterns is an interesting feature of this family of approaches, since this also alleviates aspects (ii) and (iii) of the "curse of dimensionality". Aspect (iv) is addressed partially.

Correlation Clustering

Correlation clustering approaches follow the most general model: Points forming a cluster can be located on an arbitrarily oriented hyperplane (i.e., subspace). These patterns occur if some attributes follow linear, but complex correlations among each other (i.e., one attribute may be the linear combination of several other attributes). The main point addressed by these approaches is therefore aspect (iv) of the "curse of dimensionality". The most widespread technique is the application of principal component analysis (PCA) on locally selected sets of points. Other techniques are the concept of the fractal dimension or applying the Hough transform on the data set. The first approach is proposed in [3]. The general model for this family of approaches is described in [1].

Key Applications

In many scientific and economic fields (like astronomy, physics, medicine, biology, archaeology, geology, geography, psychology, and marketing) vast amounts of high dimensional data are collected. To gain the full potential out of the gathered information, subspace clustering techniques are useful in all these domains. Pattern-based approaches are especially popular in microarray data analysis.

Future Directions

The different groups of subspace clustering techniques (subspace clustering, projected clustering, pattern-based clustering, correlation clustering) tackle different sub--problems of the "curse of dimensionality". There remain challenges for each of these problems. However, as a next-generation-type of approach, algorithms to tackle more and more aspects simultaneously can be expected.

Cross-references

▶ Apriori Property and Breadth-First Search Algorithms
▶ Clustering Overview and Applications
▶ Curse of Dimensionality
▶ Data Mining
▶ Dimensionality Reduction
▶ Feature Selection for Clustering

Recommended Reading

1. Achtert E., Böhm C., Kriegel H.-P., Kröger P., and Zimek A. Deriving quantitative models for correlation clusters. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006.
2. Aggarwal C.C., Procopiuc C.M., Wolf J.L., Yu P.S., and Park J.S. Fast algorithms for projected clustering. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
3. Aggarwal C.C., and Yu P.S. Finding generalized projected clusters in high dimensional space. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000.
4. Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998.
5. Bellman R. Adaptive Controll Processes. A Guided Tour. Princeton University Press, 1961.
6. Beyer K., Goldstein J., Ramakrishnan R., and Shaft U. When is "nearest neighbor" meaningful? In Proc. 7th Int. Conf. on Database Theory, 1999.
7. Cheng Y. and Chruch G.M. Biclustering of expression data. In Proc. 8th Int. Conf. Intelligent Systems for Molecular Biology, 2000.
8. Hartigan J.A. Direct clustering of a data matrix. J. Am. Stat. Assoc., 67(337):123–129, 1972.
9. Hinneburg A., Aggrawal C.C., and Keim D.A. What is the nearest neighbor in high dimensional spaces? In Proc. 26th Int. Conf. on Very Large Data Bases, 2000.
10. Kriegel H.P., Kröger P., and Zimek A. Clustering high dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. ACM Trans. Knowl. Discov. Data, 3(1), 2009.
11. Madeira S.C. and Oliveira A.L. Biclustering algorithms for biological data analysis: A survey, IEEE Trans. Comput. Biol. Bioinf., 1(1):24–45, 2004.
12. Parsons L., Haque E., and Liu H. Subspace clustering for high dimensional data: A review. SIGKDD Explorations, 6(1):90–105, 2004.

Subspace Selection

▶ Dimension Reduction Techniques for Clustering

Subsumed by Windows Communication Framework

▶ .NET Remoting

Success at n

Nick Craswell
Microsoft Research, Cambridge, UK

Synonyms

S@n

Definition

Success at n is an information retrieval relevance measure, equal to 1 if the top-n documents contain a relevant document and 0 otherwise. When averaged across multiple queries, the success rate at n indicates how often something relevant was retrieved within the top-n.

Key Points

A system with a high success rate will be one that rarely retrieves zero relevant documents. Therefore, success rate can be employed to monitor failure. The success rate of two systems may differ even if they have the same mean precision at n, because one system has higher variance than the other. Success at n models the satisfaction of a user who does not need to see many relevant documents, is prepared to view up to n results and is disappointed by a completely irrelevant top-n.

Cross-references

► Precision at n
► Precision-Oriented Effectiveness Measures

Succinct Constraints

Carson Kai-Sang Leung
University of Manitoba, Winnipeg, MB, Canada

Definition

Let `Item` be the set of domain items. Then, an itemset $SS_j \subseteq \texttt{Item}$ is a *succinct set* if SS_j can be expressed as a result of selection operation $\sigma_p(\texttt{Item})$, where σ is the usual selection operator and p is a selection predicate. A powerset of items $SP \subseteq 2^{\texttt{Item}}$ is a *succinct powerset* if there is a fixed number of succinct sets $SS_1,\ldots,SS_k \subseteq \texttt{Item}$ such that SP can be expressed in terms of the powersets of $SS_1,\ldots,SS_k$ using set union and/or set difference operators. A constraint C is *succinct* provided that the set of itemsets satisfying C is a succinct powerset.

Key Points

Succinct constraints [1,2] possess the following nice properties. For any succinct constraint C, there exists a precise "formula" – called a *member generating function* (*MGF*) – to enumerate *all* and *only* those itemsets that are guaranteed to satisfy C. Hence, if C is succinct, then C is pre-counting prunable. This means that one can directly generate precisely the itemsets that satisfy C – without looking at the transaction database *TDB* and even before counting the support (or frequency) of itemsets. In other words, whether an itemset S satisfies C or not can be determined based on the selection of items from the domain. Examples of succinct constraints include $max(S.Price) \leq \$120$ and $max(S.Price) \geq \$80$, which express that the maximum price of all items in an itemset S is at most \$120 and at least \$80 respectively. The set of itemsets satisfying the former can be expressed as $2^{\sigma_{Price\leq\$120}(\texttt{Item})}$; these itemsets can be enumerated by using only the items having prices at most \$120 – via the MGF $\{X \mid X \subseteq \sigma_{Price\leq\$120}(\texttt{Item}), X \neq \emptyset\}$. Similarly, the set of itemsets satisfying the second constraint $max(S.Price) \geq \$80$ can be expressed as $2^{\texttt{Item}} - 2^{\sigma_{Price<\$80}(\texttt{Item})}$; these itemsets can be enumerated by using the items having prices at least \$80 (i.e., mandatory items) and other items (i.e., optional items) – via the MGF $\{Y \cup Z | Y \subseteq \sigma_{Price\geq\$80}(\texttt{Item}), Y \neq \emptyset, Z \subseteq \sigma_{Price<\$80}(\texttt{Item})\}$.

Cross-references

► Frequent Itemset Mining with Constraints

Recommeded Reading

1. Lakshmanan L.V.S., Leung C.K.-S., and Ng R.T. Efficient dynamic mining of constrained frequent sets. ACM Trans. Database Syst., 28(4):337–389, 2003.
2. Ng R.T., Lakshmanan L.V.S., Han J., and Pang A. Exploratory mining and pruning optimizations of constrained associations rules. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 13–24.

Suffix Stripping

► Stemming

Suffix Tree

Maxime Crochemore[1,2], Thierry Lecroq[3]
[1] King's College London, London, UK
[2] University of Paris-East, Paris, France
[3] University of Rouen, Rouen, France

Synonyms

Compact suffix tries

Definition

The suffix tree $\mathcal{S}(y)$ of a non-empty string y of length n is a compact trie representing all the suffixes of the string.

The suffix tree of y is defined by the following properties:

1. All branches of $\mathcal{S}(y)$ are labeled by all nonempty suffixes of y $_\sqcup$.
2. Edges of $\mathcal{S}(y)$ are labeled by strings.
3. Internal nodes of $\mathcal{S}(y)$ have at least two children.
4. Edges outgoing an internal node are labeled by segments starting with different letters.
5. The segments are represented both by their starting position on y and their lengths.

The space sign $_\sqcup$ appended at the end of y avoids marking nodes, and implies that $\mathcal{S}(y)$ has exactly n leaves (number of non-empty suffixes).

All the properties then imply that the total size of $\mathcal{S}(y)$ is $O(n)$, which makes it possible to design a linear-time construction of the suffix tree.

Historical Background

The first linear time algorithm for building a suffix tree of a string of length n is from Weiner [15], but it requires quadratic space: $O(n \times \sigma)$ where σ is the size of the alphabet. The first linear time and space algorithm for building a suffix tree is from McCreight [13]. It works "off-line," inserting the suffixes from the longest one to the shortest one. A strictly sequential version of the suffix tree construction was described by Ukkonen [14]. When the alphabet is potentially infinite, the optimal construction algorithms of the suffix tree can be implemented to run in time $O(n \log \sigma)$, since they imply an ordering on the letters of the alphabet. On particular integer alphabets, Farach [5] showed that the construction can be done in linear time.

The minimization in the sense of automata theory of the suffix trie gives the suffix automaton. The suffix automaton of a string is also known under the name of *DAWG*, for *Directed Acyclic Word Graph*. Its linearity was discovered by Blumer et al. (see [1]), who gave a linear construction (on a fixed alphabet). The minimality of the structure as an automaton is from Crochemore [2], who showed how to build the factor automaton of a text with the same complexity.

The compaction of the suffix automaton gives the compact suffix automaton (see [1]). The compaction consists of removing all internal nodes with only one child and concatenating remaining successive edge labels. A direct construction algorithm of the compact suffix automaton was presented by Crochemore and Vérin [4]. The same structure arises when minimizing the suffix tree.

The suffix array of the string y consists of both the permutation of positions on the text that gives the sorted list of suffixes, and the corresponding array of lengths of their longest common prefixes (LCP). The suffix array of a string with the associated search algorithm based on the knowledge of the common prefixes is from Manber and Myers [12]. The suffix array can be built in linear time on integer alphabets (see [8–10]).

For the implementation of index structures in external memory, the reader can refer to Ferragina and Grossi [6].

Foundations

Suffix Trees

The suffix tree $\mathcal{S}(y)$ of the string $y = \texttt{ababbb}_\sqcup$ is presented in Fig. 1. It can be seen as a compaction of the suffix trie $\mathcal{T}(y)$ of y given in Fig. 2.

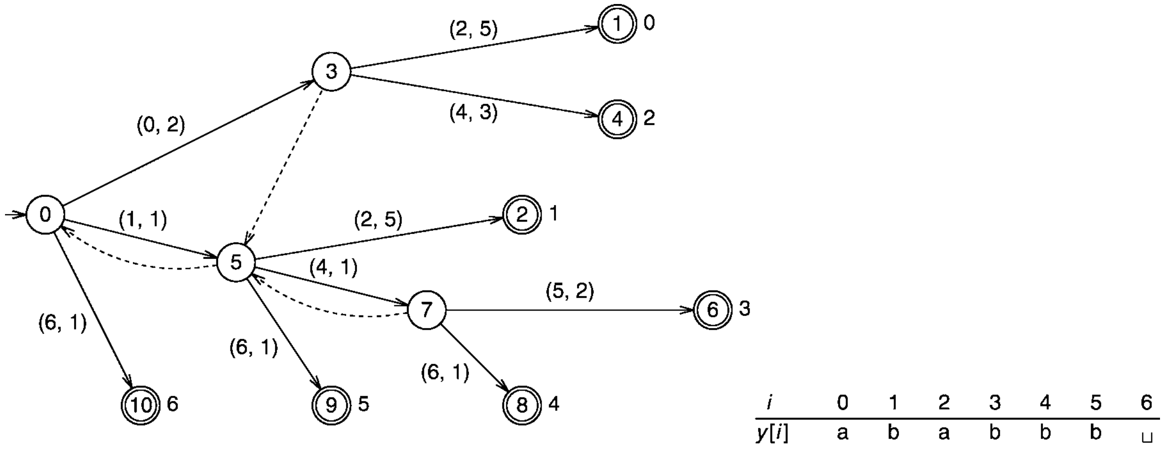
Nodes of $\mathcal{S}(y)$ and $\mathcal{T}(y)$ are identified with segments of y. Leaves of $\mathcal{S}(y)$ and $\mathcal{T}(y)$ are identified with suffixes of y. An output is defined, for each leaf, which is the starting position of the suffix in y.

The two structures can be built by successively inserting the suffixes of y from the longest to the shortest.

In the suffix trie $\mathcal{T}(y)$ of a string y of length n there exist n paths from the root to the n leaves: each path spells a different non-empty suffix of y. Edges are labeled by exactly one symbol. The suffix trie can have a quadratic number of nodes, since the sum of the lengths of all the suffixes of y is quadratic.

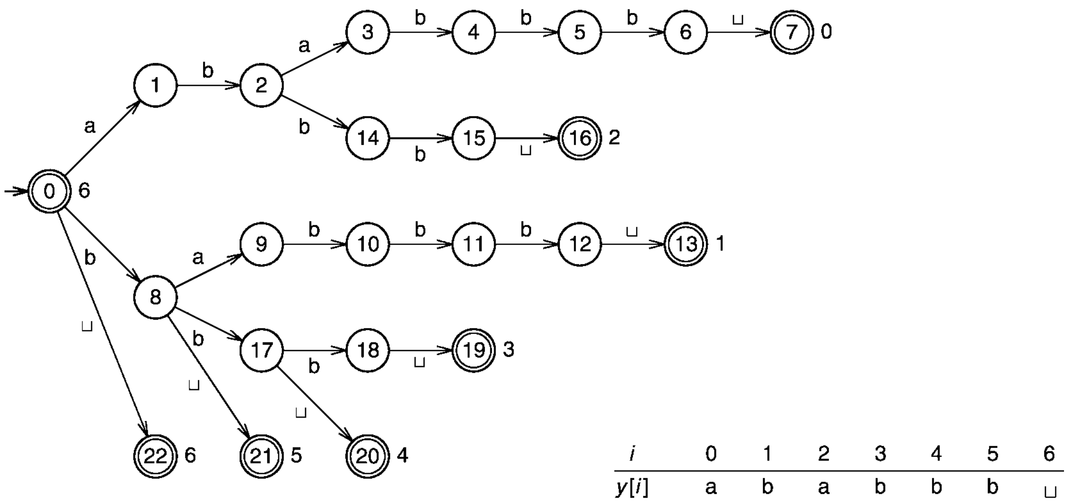
To get the suffix tree from the suffix trie, internal nodes with exactly one successor are removed. Labels of edges between remaining nodes are then concatenated. Edges are now labeled with strings. This gives a linear number of nodes, since there are exactly n leaves and since every internal node (called a fork) has at least two successors; there can be at most $n - 1$ forks. This also gives a linear number of edges.

Now, in order that the space requirement becomes linear, since the labels of the edges are all segments of y, a segment $y[i \mathinner{..} i + \ell - 1]$ is represented by the pair



i	0	1	2	3	4	5	6
$y[i]$	a	b	a	b	b	b	⊔

Suffix Tree. Figure 1. Suffix tree $\mathcal{S}(y)$ of ababbb. Nodes are numbered in the order of creation. The small number close to each leaf corresponds to the position of the suffix associated with the leaf.



i	0	1	2	3	4	5	6
$y[i]$	a	b	a	b	b	b	⊔

Suffix Tree. Figure 2. Suffix trie $\mathcal{T}(y)$ of ababbb. Nodes are numbered in the order of creation. The small number close to each leaf corresponds to the position of the suffix associated with the leaf.

(i, ℓ). Thus, each edge can be represented in constant space. This technique requires having y residing in the main memory.

Overall, there is a linear-number of nodes and a linear number of edges, each node and each edge can be represented in constant space, thus, the suffix tree requires linear space.

There exist several direct linear-time-construction algorithms of the suffix tree that avoid the construction of the suffix trie followed by its compaction.

The McCreight's algorithm [13] directly constructs the suffix tree of the string y by successively inserting the suffixes of y from the longest one to the shortest one. The insertion of the suffix of y beginning at position i (i.e., $y[i\ ..\ n-1]$) consists first of locating (creating it if necessary) the fork associated with the longest prefix of $y[i..n-1]$ common with a longer suffix of y: $y[0..n-1], y[1..n-1], \ldots,$ or $y[i-1..n-1]$. Let us call the head u this longest prefix and call the tail v the remaining part of the suffix such that $y[i..n-1] = uv$. Once the fork p associated with u has been located, it is enough to add a new leaf q labeled by i and a new edge labeled by $(i+|u|,|v|)$ from p to q to complete the insertion of $y[i..n-1]$ into the structure. The reader can refer to [3] or [7] for further details.

The linear time of the construction is achieved by using a function called suffix link, defined on the forks as follows: if fork p is identified with segment av, $a \in A$ and $v \in V^*$, then $s_y(p) = q$ where fork q is identified with v.

Suffix links are represented by dotted arrows in Fig. 1.

The suffix links create shortcuts that are used to accelerate head computations. If the head of $y[i - 1 \mathinner{..} n - 1]$ is of the form au ($a \in A$, $u \in V^*$) then u is a prefix of the head of $y[i \mathinner{..} n - 1]$. Therefore, using suffix links, the insertion of the suffix $y[i \mathinner{..} n - 1]$ consists first of finding the fork corresponding to the head of $y[i \mathinner{..} n - 1]$ (starting from suffix link of the fork associated with au), and then in inserting the tail of $y[i \mathinner{..} n - 1]$ from this fork.

Ukkonen algorithm [14] works on-line, i.e., it builds the suffix tree of y processing the symbols of y from the first to the last. It also uses suffix links to achieve a linear-time computation.

Weiner, McCreight and Ukkonen algorithms work in $O(n)$ time whenever $O(n \times \sigma)$ space is used. If only $O(n)$ space is used then the $O(n)$ time bound should be replaced by $O(n \times \log \min\{n, \sigma\})$. This accounts for the time to access a specific edge stored in each nodes. The reader can refer to [11] for specific optimized implementations of suffix trees.

For particular integer alphabets, when the alphabet of y is in the interval $[1 \mathinner{..} n^c]$ for some constant c, Farach [5] showed that the construction can be done in linear time.

Generalized suffix trees are used to represent all the suffixes of all the strings belonging to a set of strings.

Indexes

The suffix tree serves as a full index on the string: it provides a direct access to all segments of the string, and gives the positions of all their occurrences in the string. An index on y can be considered as an abstract data type whose basic set is the set of all the segments of y, and that possesses operations giving access to information relative to these segments. The utility of considering the suffixes of a string for this kind of application comes from the obvious remark that every segment of a string is the prefix of a suffix of the string.

Once the suffix tree of a text y is built, searching for x in y remains to spell x along a branch of the tree. If this walk is successful, the positions of the pattern can be output. Otherwise, x does not occur in y.

Any kind of trie that represents the suffixes of a string can be used to search it. However, the suffix tree has additional features which imply that its size is linear.

We consider four operations relative to the segments of a string y: membership, first position, number of occurrences and list of positions.

The first operation on an index is the membership of a string x to the index, that is to say the question to know whether x is a segment of y. This question can be specified in two complementary ways according to whether x is a segment of y or not. If x does not occur in y, it is often interesting in practice to know the longest beginning of x that is a segment of y. This is the type of usual answer necessary for the sequential search tools in a text editor.

The methods produce without large modification the position of an occurrence of x, and even the position of the first or last occurrence of x in y.

Knowing that x is in the index, other relevant information is constituted by its number of occurrences in y. This information can differently direct the ulterior searches.

Finally, with the same assumption as previously, complete information on the localization of x in y is supplied by the list of positions of its occurrences.

Suffix trees can easily answer these questions. It is enough to spell x from the root of $\mathcal{S}(y)$. If it is not possible, then x does not occur in y. Whenever x occurs in y, let w be the shortest segment of y that is such x is a prefix of w and w is associated with a node p of $\mathcal{S}(y)$. The number of leaves of the subtree rooted in the node p then gives the number of occurences of x in y. All these numbers associated with nodes of the tree can be precomputed in linear time. The smallest (respectively largest) of these leaves gives the position of the first (resp. last) position of x in y. The list of leaves numbers gives the list of the position of x in y (see [3]).

Key Applications

Suffix trees are used to solve string searching problems, mainly when the text into which several patterns have to be found is fixed. It is also used in other string related problems such as Longest Repeated

Substring, Longest Common Substring. It can be used to perform text compression. Word suffix trees can be used when processing natural languages, in order to represent only suffixes starting after separators such as space or line feed. Gusfield [7] gives many applications of suffix trees in computational biology.

Cross-references

► Tries

Recommended Reading

1. Blumer A., Blumer J., Ehrenfeucht A., Haussler D., Chen M.T., and Seiferas J. The smallest automaton recognizing the subwords of a text. Theor. Comput. Sci. 40(1):31–55, 1985.
2. Crochemore M. Transducers and repetitions. Theor. Comput. Sci. 45(1):63–86, 1986.
3. Crochemore M., Hancart C., and Lecroq T. Algorithms on Strings. Cambridge University Press, Cambridge, UK, 2007.
4. Crochemore M. and Vérin R. On compact directed acyclic word graphs. In Structures in Logic et Computer Science, LNCS 1261: 192–211, 1997.
5. Farach M. Optimal suffix tree construction with large alphabets. In Proc. 38th IEEE Annual Symp. on Foundations of Computer Science, 1997, pp. 137–143.
6. Ferragina P. and Grossi R. The string B-tree: A new data structure for string search in external memory et its applications. J. Assoc. Comput. Mach., 46:236–280, 1999.
7. Gusfield D. Algorithms on strings, trees and sequences. Cambridge University Press, Cambridge, UK, 1997.
8. Kärkkäinen J. and Sanders P. Simple linear work suffix array construction. In Proc. 30th Int. Colloquium on Automata, Languages, and Programming, 2003, pp. 943–955.
9. Kim D.K., Sim J.S., Park H., and Park K. Linear-time construction of suffix arrays. In Proc. 14th Annual Symp. Combinatorical Pattern Matching, 2003, pp. 186–199.
10. Ko P. and Aluru S. Space efficient linear time construction of suffix arrays. In Proc. 14th Annual Symp. Combinatorical Pattern Matching, 2003, pp. 200–210.
11. Kurtz S. Reducing the space requirement of suffix trees. Softw., Pract. Exp. 29(13):1149–1171, 1999.
12. Manber U. and Myers G. Suffix arrays: a new method for on-line string searches. SIAM J. Comput. 22(5):935–948, 1993.
13. McCreight E.M. A space-economical suffix tree construction algorithm. J. Algorithms 23(2):262–272, 1976.
14. Ukkonen E. On-line construction of suffix trees. Algorithmica 14(3):249–260, 1995.
15. Weiner P. Linear pattern matching algorithm. In Proc. 14th Annual IEEE Symp. on Switching et Automata Theory. Washington, DC, 1973, pp. 1–11.

Suffixing

► Stemming

Summarizability

Arie Shoshani
Lawrence Berkeley National Laboratory, Berkeley,
CA, USA

Synonyms

Summarization correctness; Statistical correctness

Definition

Summarizability is a property that assures the correctness of summary operations over *On-Line Analytical Processing* (OLAP) databases, which are akin to Statistical Databases [10]. Such databases are generally referred to as "summary databases," and have a data model based on one or more measures defined over the cross product of *dimensions*. For example, a bookstore company may have multiple stores in many cities. Assume that there is a database containing the stores revenues for books sold per day over the last three years. In such a database, "revenue" is a *measure*, and "book," "store," "day" are the dimensions that define the cross product over which the measure revenue is defined. A dimension in a summary database is said to be *summarizable* relative to a measure, if a summary statistic (sum, average, etc.) applied over the dimension produces correct results. For example, if summarization over all the books sold to obtain "total_revenues per store, per day" yields correct results, the dimension "book" is considered summarizable relative to the measure "revenue." There are certain conditions that have to hold in order to get correct results, which are discussed in the next section. Often, dimensions are organized into a *hierarchy* of categories. For example, days can naturally be organized into months, and months into years. Similarly, books can be organized by book-types (e.g., cooking, fiction, etc.). Summarization can then be applied to categories, such as summarizing over books and days to get "revenue per book-type, per store, per year." In such cases the summarizability property must apply to each category level of the category hierarchy of a dimension, for that dimension to be considered summarizable.

Historical Background

Statistical Databases, which were introduced in the 1980s [2], and OLAP databases, introduced in the 1990s [1,3,4] have a similar data model [1], but the issue of summarizability was not introduced until 1990 [9] and studied carefully until 1997 [7]. After that time, several authors have treated summarizability formally [5,6,8].

Foundations

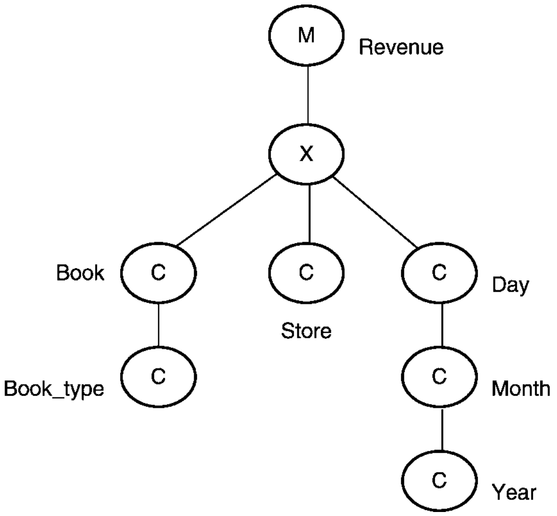
Next, examples that violate summarizability are presented, and using these examples the conditions for summarizability are stated. There are three such conditions that must hold in order for a dimension to be summarizable. Two of the conditions refer to the category levels in a dimension, and the third is a condition of the dimension relative to the measure. Next, the basic notation used throughout this document is introduced.

Notation

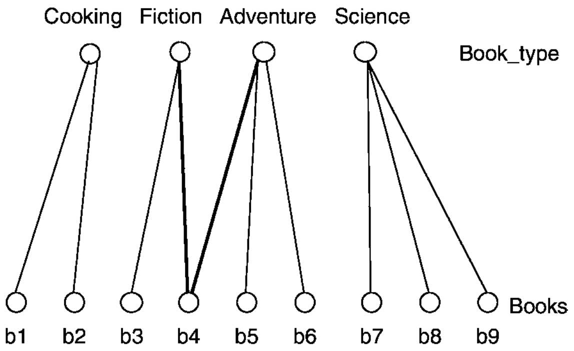
Consider, again, the example above of revenues per book, store, and day. Using a notation commonly used for such databases, this example database can be represented as "revenue (book, store, day)." For the category hierarchies of dimensions, the notation [C_1 -> C_2 -> ... C_i -> $C_{i\ +\ 1,\ ...}$ C_n] is used to represent a category hierarchy of a dimension of height n, starting from the more detailed level towards higher levels. Thus, for the example above, where the two hierarchies mentioned above are over the dimensions book and day, the database will be represented as:

"revenue ([book-> book-type], store, [day -> month -> year])."

These concepts and notation are shown graphically in Fig. 1, where the letters M, C, and X represent Measure, Category-level, and X-product (cross-product), respectively.



Summarizability. Figure 1. Revenue of books sold in a particular store on a particular day.



Summarizability. Figure 2. Books organized by book_type.

The "Disjointness" Condition

Consider the revenue database above. Suppose that the book-type set is (cooking, fiction, adventure, science, etc.). Most of the books will usually belong to a single book_type, but some could be categorized under two or more types. This is shown graphically in Fig. 2, for sales in a particular day for a particular store. As can be seen, there in one book, b4, which is classified under the categories "fiction" and "adventure." If the revenues by book_type are added to generate "revenue (book-type, store, day)," the totals will be incorrect, because if the revenues for all book-types are added, the revenue for book b4 is added twice. The reason is, of course, that book b4 belongs to two parent categories.

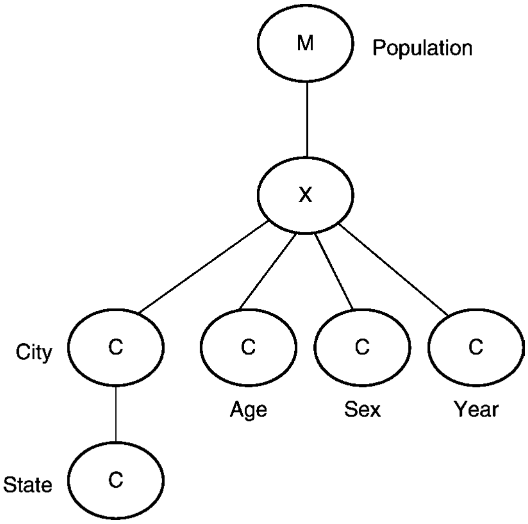
The disjointness condition states: given two consecutive category-levels C_low and C_high in a dimension, where [C_low -> C_high], the sets of lower-level categories elements that belong to each category element of the higher-level, must be disjoint. This condition can also be expressed as requiring that the category elements of the category levels form a "strict" hierarchy [8]. Yet another way to state this condition is to say that there must be a one-to-many relationship from C_high to C_low.

This seemingly simple observation is the source of incorrect statistics in many systems that do not enforce summarizability conditions. Under such conditions, summarization is still possible by special treatment, such as choosing to assign revenues equally to shared nodes (in the example above assigning half the revenue for book b4 to each of the "fiction" and "adventure" book types). However, this is not usually done. Note that summarization of book revenues to get, for example, "revenues (store, day)" will yield a correct result, since the category book_type is not involved.

The "Completeness" Condition

Completeness is a condition that holds if all the children of higher-level category elements exist. If some of the children are missing, then the summary to the higher level may be incorrect. Consider, for example, a database that contains "population (city, year, race, sex)." Suppose further that cities are organized by states, as shown in Fig. 3. In this database, if the population is summarized to the "state" level, the result of populations is obviously incorrect, since only populations of cities are taken into account and not populations of villages and small towns. However, if the measure was stated as "populations_in_cities" then the [city -> state] category mapping would be summarizable. This example shows that the second condition of completeness is relative to the measure semantics.

One way to overcome the "incompleteness" condition is to add instances that account for the missing elements in the category. In the population example, one can add for each state, in addition to the cities in the state, an instance that accounts for the population in all areas other than cities. Such a node can be labeled "other_areas" for example. If this is done, summarization to the state level would yield the correct summary population.



Summarizability. Figure 3. The population database is not summarizable to the "state" level.

Another way to determine if a category level satisfies the completeness condition can often be based on external knowledge. For example, suppose that the dimension "age" in Fig. 3 is organized into age_groups: (0–10) (10–20), ... ,(90–100). Is summarization to the age_group level correct in this case? It is only correct if there is external knowledge that this database does not contain people older that 100. Otherwise, a category (>100) has to be added in order to satisfy the completeness condition.

The "Measure Type" Condition

Consider the database in Fig. 3 again, where a higher category level is added to the dimension year: [year -> decade]. Obviously, population cannot be summarized to the decade level, since adding the yearly populations does not yield a meaningful measure for the decade. However, if the measure was "average population" per year (and "counts" were also recorded), the average population per decade could be calculated. Why is that? As another example, consider the book revenues database in Fig. 1. Obviously, the revenues can be summarized from days, to months, to years. However, if the measure was "number_of_unsold _books," this cannot be summarized (added) over the time dimension. The reason stems from the semantic behavior of "temporal aggregation."

In statistics the term "temporal aggregation" is used to describe the behavior of measures when aggregating over the time domain. The measures are classified into three types: "stock," "flow," and "value-per-unit." It turns out that these types behave differently when summarized over time, depending on the summary statistics used. In particular, a measure of "stock" type cannot be summed over the time domain, whereas a measure of "flow" type can be summed over the time domain. In the example discussed above, "population" is of type stock, and so is "number_of_unsold _books," and therefore they cannot be summed over the time dimension. In contrast, "book_revenues" is of type "flow," and therefore can be summed.

In general, measures of type "stock" refer to a state of the measure recorded at a particular point in time (such as inventory), while measures of type "flow" record values of events over a period of a time (such as sales). A measure of the type "value-per-unit" is similar to "stock" in that it is recorded at a particular point in time, but it has a per-unit value (such as the cost of a book). In [7] a table is given for temporal summarizability for each measure type for five common aggregation operators: min, max, sum, avg, and range. The table is reproduced in Fig. 4. As can be seem only the operator "sum" is not summarizable for the types "stock" and "value-per-unit." It turns out that "value-per-unit" is also not summarizable for non-temporal aggregation in the case of "sum," but all other cases are summarizable for non-temporal aggregation [7].

Summary

The conditions that are necessary to ensure correctness of aggregation operations over statistical and OLAP databases were presented. Such conditions are referred to as "summarizability conditions." Summarizability conditions apply to the dimensions of multidimensional data structures and to the category hierarchies in each dimension. Such conditions depend on the summary measure type, and whether the summarization is over the time domain. Note that the summarizability of each category level in a hierarchy is independent of the others; that is, some category levels can be summarizable while others are not.

Summarizability conditions are applicable to any database system that supports aggregation operations. While these conditions were described here in the context of the OLAP data model, these conditions apply to other models that do not express the multidimensional structures explicitly. In particular, it is possible to represent an OLAP schema as a relational schema, where the semantics of multidimensionality and category hierarchies are not explicit. In order to achieve correct results in aggregation operations from such relational databases, it is necessary to identify these (multidimensionality and category hierarchies) structures, and to make sure that the summarizability conditions hold. If all the conditions do not hold, aggregation operations should be avoided and/or refused.

	stock	flow	value/unit
min	yes	yes	yes
max	yes	yes	yes
sum	no	yes	no
avg	yes	yes	yes
range	yes	yes	yes

Summarizability. Figure 4. Temporal summarizability by measure type and function type.

Key Applications

It is often claimed that statistical operations can be inaccurate for various reasons. The better-known reason is summarization over null values. For example, taking an average over a set of values where some are null, will produce the wrong result if the null elements are represented as zeros, or if they are not discounted in the computation. Summarizability conditions are just as important, but are more subtle, semantically based, and are often overlooked. It is essential that such conditions are checked in any database that provides aggregation operators, including OLAP and relational database systems.

Future Directions

The three conditions described above for ensuring summarizability are necessary conditions. However, while it is believed that these conditions are also sufficient, this was not shown formally so far. Another aspect of future work is adding annotations to schemas as to whether summarizability conditions hold, and how to automate the checking of summarizability conditions dynamically in a database as data instances are entered into (or modified in) the database. Once summarizability is made part of the data model, it is necessary to enhance the aggregation operators to avoid summarization over non-summarizable data.

Cross-references

► Dimension
► Hierarchy
► Measure
► Multidimensional Modeling
► On-Line Analytical Processing
► Quality of Data Warehouses
► Query Processing in Data Warehouses
► Scientific Databases

Recommended Reading

1. Agrawal R., Gupta A., and Sarawagi S. Modeling multidimensional databases. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 232–243.
2. Chan P. and Shoshani A. Subject: a directory driven system for organizing and accessing large statistical databases. In Proc. 7th Int. Conf. on Very Data Bases, 1981, pp. 553–563.
3 Codd E.F., Codd S.B., and Salley C.T. Providing olap (online analytical processing) to user-analysts: An IT mandate, Codd and Associates technical report, 1993.
4. Gray J., Bosworth A., Layman A., and Pirahesh H. Data cube: a relational aggregation operator generalizing group-by, cross-tabs

and sub-totals. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 152–159.

5. Hurtado C.A., Gutiérrez C., and Mendelzon A. Capturing summarizability with integrity constraints in OLAP. ACM Trans. Database Syst., 30(3):854–886, 2005.
6. Hurtado C.A. and Mendelzon A.O. Reasoning about summarizability in heterogeneous multidimensional schemas. In Proc. 8th Int. Conf. on Database Theory, 2001, pp. 375–389.
7. Lenz H-J. and Shoshani A. Summarizability in OLAP and statistical data bases. In Proc. 9th Int. Conf. on Scientific and Statistical Database Management, 1997, pp. 132–143.
8. Pedersen T.B. and Jensen C.S. Multidimensional data modeling for complex data. In Proc. 15th Int. Conf. on Data Engineering, 336–345, 1999.
9. Rafanelli M. and Shoshani A. STORM: a statistical object representation model. In Proc. 2nd Int. Conf. on Scientific and Statistical Database Management, 1990, pp. 14–29.
10. Shoshani A. OLAP and statistical databases: similarities and differences. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997, pp. 185–196.

Summarization

Jimmy Lin
University of Maryland, College Park, MD, USA

Synonyms

Text/document summarization; Automatic abstracting; Distillation; Report writing

Definition

Summarization systems generate condensed outputs that convey important information contained in one or more sources for particular users and tasks. In principle, input sources and system outputs are not limited to text (e.g., keyframe extraction for video summarization), but this entry focuses exclusively on generating *textual* summaries from *textual* sources.

Historical Background

Summarization has a long history dating back to the 1960s, when researchers first started developing computer systems that processed natural language [6,12]. Following a number of decades with comparatively few publications, summarization research entered a new phase in the 1990s. A revival of interest was spurred by the growing availability of text in electronic formats and later the World Wide Web. The enormous quantities of information people come into contact with on a daily basis created a need for applications that help users cope with the proverbial information overload problem. Summarization systems attempt to address this need.

Foundations

Summarization is a broad and diverse field. Traditionally, it is considered a sub-area of natural language processing, but a significant number of innovations have their origins in information retrieval. This entry is organized as follows: first, various summarization factors are discussed. Next, a tripartite processing model for summarization systems is presented, which provides a basis for discussing general issues. Finally, selected summarization techniques are briefly overviewed.

Summarization Factors

To better understand summarization, it is helpful to enumerate its many dimensions – what Sparck Jones [19] calls “factors”. These factors provide a basis for understanding various automatic methods, and can be grouped into three broad categories: *input*, *purpose*, and *output*. What follows is meant to be an overview of important factors, and not intended to be exhaustive.

Input factors characterize the source of the summaries:

1. *Single versus multiple sources.* For example, one versus multiple reports of the same event.
2. *Genre* (categories of texts) and *Register* (different styles of writing). For example, dissertations versus blogs.
3. *Written versus spoken.* For example, newspaper articles versus broadcast news.
4. *Language.* Sources may be in multiple language.
5. *Metadata.* Sources may be associated with controlled vocabulary keywords, human-assigned category labels.
6. *Structure.* Source structure may be relatively straightforward (e.g., headings and sub-headings) or significantly more complex (e.g., email threads).

Purpose factors characterize the use of summaries (i.e., why they were created):

1. *Indicative versus informative versus evaluative.* Indicative summaries are meant to guide the selection of sources for more in-depth study, whereas informative summaries cover salient information in the sources at some level of detail (and is often meant to replace the original). Evaluative summaries

assess the subject matter of the source and the quality of the work (e.g., a review of a movie).

2. *Generic versus focused.* A generic summary places equal emphasis on different information contained in the sources and provides balanced coverage. Alternatively, a summary might be focused on an information need, i.e., created to answer a question.
3. *Task.* What will the summary be used for? For example, to help write a report or to make a decision.
4. *Audience.* Whom is the summary intended for? For example, experts, schoolchildren, etc.

Output factors characterize system output (note that the input factors are relevant here also, but not repeated):

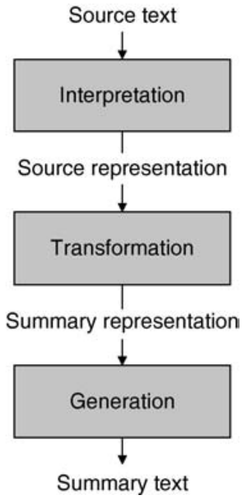
1. *Extractive versus abstractive.* Extractive summaries consist of text copied from the source material; typically, such approaches are based on shallow analysis. Abstractive summaries contain text that is system-generated, usually based on deeper analysis. Note that these approaches define a continuous spectrum, as many systems employ hybrid methods.
2. *Reduction, coverage, and fidelity.* Reduction, usually measured as a ratio between summary length and source length, is often inversely related to coverage, how much information of interest is preserved in the summary. The summary should also preserve source information accurately.
3. *Coherence.* Does the summary read fluently and grammatically, both syntactically and at the discourse level? For summaries not intended to be fluent prose (e.g., bullets), this factor is less important.

Input, purpose, and output factors together characterize the many dimensions of summarization and provide a basis for subsequent discussions. Note, however, that not all factors figure equally in current summarization systems – for a variety of reasons, the field has focused on some more than others.

Processing Model

Sparck Jones characterizes the process of summarization as a reductive transformation of source text to summary text through content condensation by selection and/or generalization of what is important in the source [19]. She proposes a tripartite processing model, shown in Figure 1, that serves as a framework for understanding how various summarization techniques fit together (see also [15] for a similar model). Systems first convert source text into the source representation, which is then transformed into the summary representation. Finally, the summary representation is realized as natural language text. Note that these stages do not necessarily map to system components, as the processing model only describes abstract processing tasks. Since this model does not prescribe specific representations or particular processing methods, it is sufficiently general to describe a wide variety of summarization systems while at the same time highlighting important differences.

As previously discussed, input may come from one or multiple sources (the term "documents" is used generically, recognizing that sources may also be speech, email, etc.). Single-document summarization is challenging because simple baselines are often very difficult to improve upon. For example, since news articles are typically written in the "inverse pyramid" style (most important information first), the first sentence or paragraph makes an excellent summary. Frequently, longer documents (e.g., reports) contain "executive summaries", which nicely capture important information in the documents. Multi-document summarization faces a different set of challenges, the most salient of which is the possibility of redundant information in the sources (e.g., multiple news articles



Summarization. Figure 1. A tripartite processing model for summarization.

about the same event). Frequently, the redundancy is not superficially obvious, but involves paraphrase (different syntactic structures, word choices, etc.). More complex are cases where the information partially overlaps or appears contradictory (e.g., different reports of death tolls). More generally, multi-document summarization requires systems to detect similarities and differences in text.

It is generally assumed that a summarization system is provided the source text. In cases where this is assumption is not met, information retrieval techniques may be used to first select the set of documents to summarize (from a larger collection of documents). However, since most systems assume that input sources are more or less relevant to the task at hand, they may not adequately cope with imperfect retrieval results.

The use of "representation" does not necessarily imply deep linguistic analysis or processing. In fact, most extractive summarization systems adopt a "bag of words" representation at both the source and summary end – that is, text is represented as a vector that has a feature for each word. This representation makes the obviously false assumption that word occurrences are independent and ignores the rich linguistic relationships present in text. Nevertheless, extractive techniques have proven to be effective in various summarization tasks.

With extractive techniques, generation is trivial since systems simply copy material from the source. However, pure extraction often leads to problems in overall coherence of the summary – a frequent issue concerns "dangling" anaphora. Sentences often contain pronouns, which lose their referents when extracted out of context. Worse yet, stitching together decontextualized extracts may lead to a misleading interpretation of anaphors (resulting in an inaccurate representation of source information, i.e., low fidelity). Similar issues exist with temporal expressions. Note that these problems become more severe in the multi-document case, since extracts are drawn from different sources. A general approach to addressing these issues involves post-processing extracts, for example, replacing pronouns with their antecedents, replacing relative temporal expression with actual dates, etc. Such techniques, however, can not be considered purely extractive (hence the observation that most systems are, in fact, hybrid).

In general, extractive systems can be characterized as "knowledge-poor", which is contrasted against "knowledge-rich" approaches. While not synonymous, abstractive methods tend to be associated with "knowledge-rich" approaches. They involve one or more of the following: detailed linguistic analysis on source text to produce richly annotated structures, incorporation of world knowledge to support the transformation process, or generation of fluent natural language text from abstract representations.

A canonical example of abstractive summarization involves integration with information extraction (IE) systems. Information extraction concerns the automatic identification and creation of template instances from natural language text based on some pre-defined structure. For example, a template for natural disasters might contain "slots" for type, damage, death toll, etc. An IE system would analyze text sources and automatically extract information to fill these templates, in effect, populating a structured database from free text. This process can be viewed as the interpretation stage in the summarization processing model, and the templates themselves can serve as the source representation. A summarization system can then combine information from multiple templates to generate a fluent summary (e.g., [18]).

Abstractive techniques face a number of major challenges, the biggest of which is the representation problem. Systems' capabilities are constrained by the richness of their representations and their ability to generate such structures – systems cannot summarize what their representations cannot capture. In limited domains, it may be feasible to devise appropriate structures, but a general-purpose solution depends on open-domain semantic analysis. Systems that can truly "understand" natural language are beyond the capabilities of today's technology.

Finally, coherence of system-generated text is one important output factor in summarization. Coherence is usually taken to mean fluent, grammatically correct prose that "reads well". This is a tall order, mainly because coherence is very difficult to operationalize. While humans can easily identify incoherent text, they have much more difficulty defining what makes a piece of text coherent. To make matters worse, multiple arrangements of segments might be equally coherent to a human. For extractive techniques, systems must devise an ordering of extracted segments and deal with "out-of-context" issues discussed above. For abstractive techniques, generation of fluent output from an abstract representation is sufficiently difficult that it is considered another sub-area in natural language processing. Although output coherence is a requirement in both single- and multi-document summarization, the latter presents more problems (particularly for extractive systems) given the variety of sources extracts.

Overview of Selected Techniques

Due to relatively easy access to corpora, most research in summarization over the past two decades has been on written news. As most summarization systems today are primarily extractive, these methods will occupy the bulk of this discussion.

Extractive techniques first segment source text into smaller segments (sentences, paragraphs, etc.), which are then scored according a variety of features, e.g., position in the text [6], term and phrase frequencies [12], lexical chains (degree of lexical-connectedness between various segments) [1], topics present in the text [16], or discourse prominence [14]. A widely adopted approach is to use machine learning techniques to determine the relative importance of various features (the earliest example being [10]).

The features discussed above are relevant for both single- and multi-document summarization, although their relative importance varies with the task. Historically, the summarization field focused on the single-document case first, and then subsequently moved on to multi-document summarization. This move required systems to explicitly model similarities and differences in text to address redundancy, paraphrase, entailment, contradiction, and related linguistic issues. One general approach involves clustering, as exemplified by the MEAD framework [16]. Documents are first clustered to find topics present in the sources. Clusters are represented by their centroids, which are used to rank extracts (along with other features). Maximal Marginal Relevance (MMR) [7] is another effective algorithm, specifically designed for query-focused summaries (i.e., summaries that address an information need). It iteratively selects candidate segments to include in the final summary, balancing relevance and redundancy at each iteration. Redundancy is computed by content similarity between each candidate and the current summary state (using cosine similarity) – thus, candidates containing words already in the summary are penalized. Note that neither MEAD nor MMR explicitly deals with linguistic relationships such as paraphrase, but that issue has been specifically addressed in other work [8].

After scoring and selecting segments from source documents, extractive systems must decide on an ordering in the final system output. Ideally, the output should constitute a coherent piece of text. Simple baselines for ordering segments include extraction order (i.e., by score), temporal order (based on metadata or temporal expressions), and order in source document (preserving source structure). While simple to implement, these techniques frequently yield disfluent summaries. Coherence can be improved by applying computational models of content and discourse [2]. Nevertheless, text structuring is a relatively underexplored area of summarization, particularly due to difficulty in evaluation. As a final note, one possible alternative is to abandon the assumption of summaries as fluent prose, and instead present users with a bulleted list of extracts.

Although open-domain abstractive summarization using deep semantic representations is beyond the current state of the art, a variety of successful abstractive techniques operating on syntactic structures have been developed. Most of these techniques involve parsing source documents and manipulating the resulting parse trees. One popular approach involves "trimming", or removing inessential structures from the parse tree [9,20] – for example, removing adjunct clauses that do not contribute much information. Other successful techniques include "splicing" fragments from multiple sentences (sometimes across multiple documents) – for example, embedding a simple sentence as a relative clause inside another [3,13]. Of course, these operations are not mutually exclusive. Syntactic manipulations are particularly helpful in multi-document summarization since sentences from different sources might partially overlap, e.g., a sentence contains both redundant and new information. In this case, syntactic operations can potentially deliver the best of both worlds, by eliminating redundant information and preserving new information. However, as Sparck Jones recently noted [19], there has been comparatively little work on abstractive summarization over the last decade.

Additional Readings

Beyond this entry, a number of additional sources are recommended for further reading: slides from a tutorial presentation at SIGIR 2004 [15] provide a good starting point. Special issues of the journal *Information Processing and Management* [19] and *Computational*

Linguistics [17] contain in-depth articles on selected topics. For details on specific summarization techniques, a good place to look is the online proceedings of the Document Understanding Conferences [4], an annual evaluation of summarization systems. A note on references in this entry: since a comprehensive bibliography is impossible due to space limitations, either representative early articles or recent ones are cited (in the latter case, the assumption is that the reader can trace citations backwards).

Key Applications

Summarization technology has a number of applications, many of which are outlined below:

Search result summarization. Search engines typically retrieve thousands of hits (if not more) in response to a user's query. Summarization systems can provide users with an overview of results to support information seeking.

Tools for analytical support. Summarization can be applied to support intelligence analysis, e.g., "prepare a report on recent insurgent activities in Basra", as well as similar activities such as investigative journalism and business intelligence.

Personal information agent. A personal information agent maintains a profile of the user's interest and proactively seeks out information (e.g., retrieving and summarizing relevant news items on a continuous basis).

Accessibility assistance. For example, a visually impaired person might make use of a screen reader augmented with summarization technology for greater efficiency.

Support for handheld devices. Handheld devices such as cell phones and PDAs with small screens could benefit from more condensed information.

Medical applications. Physicians struggle to keep current with the ever-increasing volume of medical literature. Summarization systems can be deployed to assist physicians, e.g., provide an overview of treatment options for a particular disease.

Summarization of meetings. Summarization technology can be coupled with speech recognizers to automatically generate "meeting minutes".

Future Directions

Current research in summarization can be characterized by three broad trends:

Increasing linguistic sophistication. Extractive techniques can benefit from richer features to characterize the appropriateness of a segment for inclusion in the summary – these features come from increasingly detailed linguistic analysis, enabled by advances in language processing technology. Of particular interest is the modeling of linguistic relations such as paraphrase, entailment, and contradiction. Separately, this task has been captured in the PASCAL recognizing textual entailment evaluations.

As discussed above, limitations of extractive methods can be addressed by incorporating abstractive techniques, e.g., manipulation of parse trees. Future developments appear to follow this trend, with increasingly richer representations (enabled by improvements in syntactic, semantic, discourse, and pragmatic analysis). In other words, abstractive summarization will likely be arrived at by successive approximations with hybrid techniques.

Exploration of different genres and domain-specific applications. Recently, researchers have become interested in "informal" text – a broad genre that includes emails, conversational speech, blogs, chat, SMS messages, etc. They are important because an increasing portion of our society's knowledge is captured in these channels. Furthermore, informal text push the frontiers of summarization technology by forcing researchers to develop more general and robust algorithms.

Integration with other language processing components. As technology matures, it becomes feasible to integrate summarization with other components to create more powerful applications. A few examples: integration with speech recognition to summarize TV broadcasts and meetings; integration with machine translation to summarize documents from multiple languages; integration with information retrieval and question answering to produce responses that answer complex questions.

Experimental Results

Summarization is fundamentally experimental in nature, as the effectiveness of different techniques cannot be derived from first principles. Thus, tools for assessing summary quality are critical to ensuring progress, and evaluation methods themselves represent an active area of research.

Methodologies for evaluating system output can be broadly classified into two categories: *intrinsic* and *extrinsic*. In an intrinsic evaluation, system output is directly evaluated in terms of a set of norms – for example, fluency, coverage of key ideas, or similarity to an "ideal" summary (see [19] for an overview). In particular, the last criteria has been operationalized in ROUGE [11], a commonly used automated metric that compares system output to a number of human-generated "reference" summaries. In contrast, extrinsic evaluations attempt to measure how summarization impacts some other task, for example, helping users determine if a document is relevant (see [5] and references therein). While more informative, extrinsic evaluations are much more difficult to conduct, since it often involves constructing realistic scenarios for summarization systems.

One of the most important driving forces behind summarization research is the existence of annual evaluations that provide a community-wide benchmark to assess progress. Two such evaluations are the Document Understanding Conferences [4] sponsored by the U.S. National Institute of Standards and Technology (NIST), and the NTCIR Project sponsored by Japan's National Institute of Informatics. Starting in 2008, DUC is replaced by the newly created Text Analysis Conference, also sponsored by NIST.

Data Sets

Instructions for obtaining data from the DUC and NTCIR evaluations can be found on their respective websites.

Cross-references

▶ Information Extraction
▶ Information Retrieval

Recommended Reading

1. Barzilay R. and Elhadad M. Using lexical chains for text summarization. In Proc. ACL/EACL Workshop on Intelligent Scalable Text Summarization, 1997.
2. Barzilay R. and Lee L. Catching the drift: Probabilistic content models, with applications to generation and summarization. In Proc. 2004 Human Language Technology Conf., 2004, pp. 113–120.
3. Barzilay R. and McKeown K.R. Sentence fusion for multidocument news summarization. Computat. Linguist., 31(3):297–327, 2005.
4. Document Understanding Conferences. http://duc.nist.gov/.
5. Dorr B.J., Monz C., President S., Schwartz R., and Zajic D. A methodology for extrinsic evaluation of text summarization: Does ROUGE correlate? In Proc. ACL 2005 Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization, 2005.
6. Edmundson H.P. New methods in automatic extracting. J. ACM, 16(2):264–285, 1969.
7. Goldstein J., Mittal V., Carbonell J., and Callan J. Creating and evaluating multi-document sentence extract summaries. In Proc. Int. Conf. on Information and Knowledge Management, 2000, pp. 165–172.
8. Hatzivassiloglou V., Klavans J.L., and Eskin E. Detecting text similarity over short passages: Exploring linguistic feature combinations via machine learning. In Proc. Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora, 1999.
9. Knight K. and Marcu D. Statistics-based summarization – step one: Sentence compression. In Proc. 12th National Conf. on AI, 2000, pp. 703–710.
10. Kupiec J., Pedersen J.O., and Chen F. A trainable document summarizer. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 68–73.
11. Lin C.Y. and Hovy E. Automatic evaluation of summaries using n-gram co-occurrence statistics. In Proc. 2003 Human Language Technology Conf., 2003, pp. 71–78.
12. Luhn H.P. The automatic creation of literature abstracts. IBM J. Res. Develop., 2(2):159–165, 1958.
13. Mani I., Gates B., and Bloedorn E. Improving summaries by revising them. In Proc. 27th Annual Meeting of the Assoc. for Computational Linguistics, 1999, pp. 558–565.
14. Marcu D. The Rhetorical Parsing, Summarization, and Generation of Natural Language Texts. PhD Thesis, University of Toronto, 1997.
15. Radev D.R. Text summarization. In Tutorial Presentation at the 27th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004.
16. Radev D.R., Blair-Goldensohn S., and Zhang Z. Experiments in single and multi-document summarization using MEAD. In Proc. 2001 Document Understanding Conf., 2001.
17. Radev D.R., Hovy E., and McKeown K. Introduction to the special issue on summarization. Computat. Linguist., 28(4):399–408, 2002.
18. Radev D.R. and McKeown K. Generating natural language summaries from multiple on-line sources. Computat. Linguist., 24(3):469–500, 1998.
19. Sparck Jones K. Automatic summarising: The state of the art. Inf. Process. Manage., 43(6):1449–1481, 2007.
20. Zajic D., Dorr B., Lin J., and Schwartz R. Multi-Candidate Reduction: Sentence compression as a tool for document summarization tasks. Inf. Process. Manage., 43(6):1549–1570, 2007.

Summarization Correctness

▶ Summarizability

Summary

▶ Data Sketch/Synopsis

Supervised Learning

► Classification

Support Vector Machine

HWANJO YU
University of Iowa, Iowa City, IA, USA

Synonyms

SVM

Definition

Support vector machines (SVMs) represent a set of supervised learning techniques that create a function from training data. The training data usually consist of pairs of input objects (typically vectors) and desired outputs. The learned function can be used to predict the output of a new object. SVMs are typically used for classification where the function outputs one of finite classes. SVMs are also used for regression and preference learning, for which they are called support vector regression (SVR) and ranking SVM, respectively. SVMs belong to a family of generalized linear classifier where the classification (or boundary) function is a hyperplane in the feature space. Two special properties of SVMs are that SVMs achieve (i) high generalization (Generalization denotes the performance of the learned function on testing data or "unseen" data that are excluded in training.) by maximizing the margin (Margin denotes the distance between the hyperplane and the closest data vectors in the feature space.), and (ii) support efficient nonlinear classification by kernel trick (Kernel trick is a method for converting a linear classifier into a non-linear one by using a nonlinear function to map the original observations into a higher-dimensional space; this makes a linear classification in the new space (or the feature space) equivalent to non-linear classification in the original space (or the input space).).

Historical Background

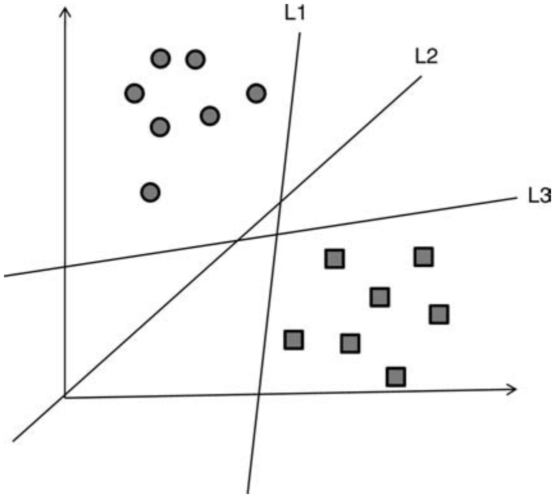
Vapnik developed the related concepts at 1979 and published an article in Russian that was translated to English at 1982. The first book introducing SVMs (written by him) were published at 1995. Since then, numerous literatures have been published including tutorials, journal articles, and books.

Foundations

SVMs were initially developed for classification [1] and have been extended for regression [4] and preference learning [3,5]. The initial form of SVMs are a binary classifier where the output of learned function is either positive or negative. A multiclass classification can be implemented by combining multiple binary classifiers using pairwise coupling method [2]. This section explains the motivation and formalization of SVM as a binary classifier, and the two key properties – margin maximization and kernel trick.

Motivation

Binary classification is to classify data objects into either positive or negative class. Each data object (or data point) is represented by a *n*-dimensional vector. Each of these data points belongs to only one of two classes. A *linear* classifier separates them with an "*n* minus 1" dimensional hyperplane. For example, Figure 1 shows two groups of data and separating hyperplanes that are lines in a two-dimensional space. There are many linear classifiers that correctly classify (or divide) the two groups of data such as L1, L2 and L3 in Fig. 1. In order to achieve maximum separation between the two classes, An SVM pick the hyperplane so that the margin, or the distance from the hyperplane to the nearest data point, is maximized. Such a hyperplane is likely to generalize better, meaning that the hyperplane not only correctly classify the given or training data points, but also is likely to correctly classify "unseen" or testing data points.



Support Vector Machine. Figure 1. Linear classifiers (hyperplane) in two-dimensional spaces.

Formalization

The data points D in Fig. 1 (or training set) can be expressed mathematically as follows:

$$D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_m, y_m)\}, \tag{1}$$

where $\vec{x}_i$ is a n-dimensional real vector, y_i is either 1 or -1 denoting the class to which the point $\vec{x}_i$ belongs. The SVM classification function $F(\vec{x})$ takes the form

$$F(\vec{x}) \Rightarrow \vec{w} \cdot \vec{x} - b. \tag{2}$$

$\vec{w}$ is the weight vector and b is the bias, which will be computed by SVM in the training process.

First, to correctly classify the training set, $F(\cdot)$ (or $\vec{w}$ and b) must return positive numbers for positive data points and negative numbers otherwise, that is, for every point $\vec{x}_i$ in D,

$$\begin{aligned} &\vec{w} \cdot \vec{x}_i - b > 0 \text{ if } y_i = 1, \text{ and} \\ &\vec{w} \cdot \vec{x}_i - b < 0 \text{ if } y_i = -1 \end{aligned}$$

These conditions can be revised into:

$$y_i(\vec{w} \cdot \vec{x}_i - b) > 0, \ \forall(\vec{x}_i, y_i) \in D. \tag{3}$$

If such a linear function F that correctly classifies every point in D or satisfies(3) exists, D is called *linearly separable.*

Second, F (or the hyperplane) needs to maximize the *margin.* Margin is the distance from the hyperplane to the closest data points. An example of such hyperplane is illustrated in Fig. 2. To achieve this, (3) is revised into the following (4).

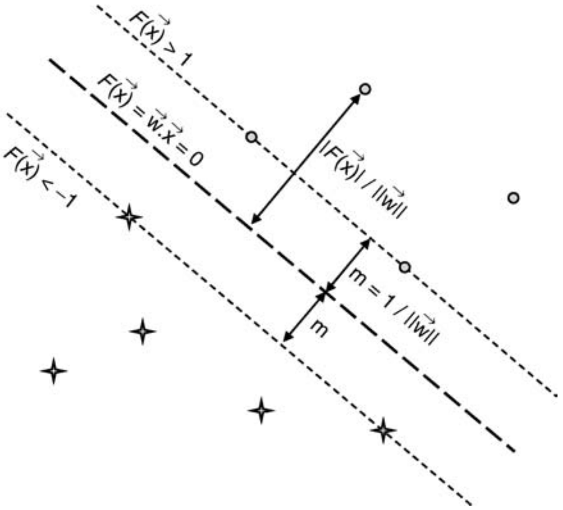
$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \ \forall(\vec{x}_i, y_i) \in D. \tag{4}$$

Note that (4) includes equality sign, and the right side becomes 1 instead of 0. If D is linearly separable, or every point in D satisfies (3), then there exists such a F that satisfies (4). It is because, if there exist such $\vec{w}$ and b that satisfy (3), they can be always rescaled to satisfy (4).

The distance from the hyperplane to a vector $\vec{x}_i$ is formulated as $\frac{|F(\vec{x}_i)|}{||\vec{w}||}$. Thus, the margin becomes

$$margin = \frac{1}{||\vec{w}||}. \tag{5}$$

because when $\vec{x}_i$ are the closest vectors, $F(\vec{x})$ will return 1 according to (4). The closest vectors, that satisfy (4) with equality sign, are called *support vectors.*


$F(\vec{x}) > 1$
$F(\vec{x}) = \vec{w}.\vec{x} = 0$
$F(\vec{x}) < -1$
$|F(\vec{x})| / ||\vec{w}||$
$m = 1 / ||\vec{w}||$
m

Support Vector Machine. Figure 2. SVM classification function: the hyperplane maximizing the margin in a two-dimensional space.

Maximizing the margin becomes minimizing $||\vec{w}||$. Thus, the training problem in SVM becomes a constrained optimization problem as follows:

$$\text{minimize}: \frac{1}{2}||\vec{w}||^2, \tag{6}$$

$$\text{subject to}: y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \quad \forall(\vec{x}_i, y_i) \in D. \tag{7}$$

The factor of $\frac{1}{2}$ is used for mathematical convenience.

However, the optimization problem will not have a solution if D is not linearly separable. To deal with such cases, *soft margin* SVM allows mislabeled data points while still maximizing the margin. The method introduces slack variables, ξ, which measure the degree of misclassification. The following is the optimization problem for soft margin SVM.

$$\text{minimize}: \quad L_1(w, b, \xi) = \frac{1}{2}||\vec{w}||^2 + C\sum_i \xi_i, \tag{8}$$

$$\begin{aligned} \text{subject to}: y_i(\vec{w} \cdot \vec{x}_i - b) &\geq 1 - \xi_i, \\ \forall(\vec{x}_i, y_i) &\in D. \end{aligned} \tag{9}$$

$$\xi \geq 0 \tag{10}$$

Due to the ξ in (9), data points are allowed to be misclassified, and the amount of misclassification or error will be minimized as well as the margin according to the objective function (8). C is a parameter that determines the tradeoff between the margin size and the amount of error in training.

This optimization problem is called quadratic programming (QP) problem, and it is the primal form of the QP. The primal form can be changed to the following dual form using the Lagrange multipliers.

$$\text{minimize}: L_2(\alpha)=\sum_i \alpha_i-\sum_i\sum_j \alpha_i\alpha_j y_i y_j K(\vec{x}_i,\vec{x}_j), \quad (11)$$

$$\text{subject to}: \sum_i \alpha_i y_i = 0 \qquad (12)$$

$$C \geq \alpha \geq 0 \qquad (13)$$

α constitute a dual representation for the weight vector such that

$$\vec{w} = \sum_i \alpha_i y_i \vec{x}_{i.} \qquad (14)$$

$K(\vec{x}_i, \vec{x}_j)$ in (11) is originally a dot product of the two vectors, that is, $\vec{x}_i \cdot \vec{x}_j$. However, the dot product can be replaced by a non-linear kernel function, which allows the algorithm to fit the maximum-margin hyperplane in the transformed feature space. The transformed feature space is usually high dimensional, and the hyperplane (or linear classifier) in the high-dimensional space becomes non-linear in the original input space. Computing the kernel function K is often done as fast as computing a dot product. In this way, the complexity of computing a non-linear function becomes the same as that of computing a linear function in SVM. This method for computing a non-linear function is called *kernel trick*. The following are popularly used non-linear kernels.

1. Radial basis function (RBF): $K(a, b) = exp(-\gamma||a - b||^2)$
2. Polynomial: $K(a, b) = (a \cdot b + 1)^d$
3. Sigmoid: $K(a, b) = tanh(\kappa a \cdot b + c)$

Once α is learned from the dual form, the linear function F in (2) becomes the following non-linear function.

$$F(\vec{x}) \Rightarrow \vec{w}\cdot\vec{x} - b \Rightarrow \sum_i \alpha_i y_i K(\vec{x}_i, \vec{x}) - b \qquad (15)$$

Key Applications

SVMs have been widely applied for object classification and pattern recognition such as text categorization, face detection in images, and handwritten digit recognition.

Cross-references

- ▶ Classification
- ▶ Regression

Recommended Reading

1. Burges C.J.C. A tutorial on support vector machines for pattern recognition. Data Min. Knowl. Discovery, 2:121–167, 1998.
2. Hastie T. and Tibshirani R. Classification by pairwise coupling. In Advances in Neural Information Processing Systems, 1998.
3. Herbrich R., Graepel T., and Obermayer K. (eds.) Large margin rank boundaries for ordinal regression. MIT Press, Cambridge, MA, 2000.
4. Smola A.J. and Scholkopf B. A tutorial on support vector regression. Technical Report, NeuroCOLT2 Technical Report NC2-TR-1998-030, 1998.
5. Yu H. SVM selective sampling for ranking with application to data retrieval. In Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2005.

Supporting Transaction Time Databases

David Lomet
Microsoft Research, Redmond, WA, USA

Synonyms

Temporal database; Multi-version database

Definition

The temporal concepts glossary maintained at http://www.cs.aau.dk/~csj/Glossary/ defines transaction time as: "The *transaction time* of a database fact is the time when the fact is current in the database and may be retrieved." A transaction time database thus stores versions of database records or tuples, each of which has a start time and an end time, delimiting the time range during which they represent the current versions of database facts. As each version is the result of transactions, the times associated with the version are the times for the transaction starting the version (the start time) and for the transaction ending the version (the end time). These transaction times are required to agree with the serialization order of the transaction, so that the database can present a transaction consistent view of the facts being stored.

Historical Background

Postgres was the first database system that supported transaction time databases [13]. It implemented a prototype relational database system using a versioning approach for recovery. This was then augmented with a version store based on the R-tree [4]. Subsequently, the TSB-tree was introduced [7], based on the WOB-tree time splitting [3], which provided an integrated approach to storing both current and historical data.

Commercially, Oracle and Rdb [5] database systems both support multi-version concurrency control, now called snapshot isolation. Oracle subsequently added a Flashback [11] feature that permitted access to historical versions, based on saving a linear history of their recovery versions. While this permits access to historical versions, it is mostly intended as an efficient means of providing an online backup for "point-in-time" recovery, a form of "media" recovery in which the database is recovered to a point just preceding a bad user transaction.

Temporal databases have been extensively studied [14], including not only transaction time but also valid time. Included as well are bi-temporal databases, where both valid and transaction time are supported. This work has clarified the conceptual issues and provided a common vocabulary of terms for describing work in the field.

Foundations

Transaction time databases usually offer the full range of database functionality that one would expect of a database storing only current facts, which is called a current time database. In addition, a transaction time database provides access to all prior facts as represented by versions of records that existed at some prior time. Several types of functionality can be envisioned.

1. Access to database facts "as of" some past time, which are called "as of" queries
2. Access to versions of a database fact in some time range, which are called "time travel" queries
3. Access to collections of database facts in some time range, i.e., "general transaction time queries"

The functionality provided by transaction time databases is important for applications such as time series data, regulatory compliance, repeatability of scientific experiments, etc. Further, a transaction time database can provide valuable system capabilities such as snapshot isolation concurrency control, recovery from bad user transactions, and recovery from media failure.

To support transaction time functionality, one needs to change the semantics of the data manipulation operations "update" and "delete." An update creates an additional version instead of overwriting the prior state, retaining both new and old versions of the data. A delete is strictly logical, providing an end time for the previously current version. In this way, the prior version persists so that queries about the database while this version was alive can be answered.

Database systems supporting transaction time data have been built that are based on the relational data model, though this is not a fundamental limitation, simply a pragmatic choice.

Implementation Approaches

There are two generic approaches to implementing transaction time databases, which are described here.

Layered Approach The premise of the layered approach is that application programmers cannot wait for vendors to build transaction time functionality into database products [15]. Rather, a middleware layer (MWL) is implemented that provides this functionality. The MWL processes data definition statements, adding timestamp fields to each record, processes data manipulation statements (queries plus updates) written in a language that exposes temporal functionality and translates them into equivalent ordinary SQL queries. Typically, start time and end time are added to each record of a transaction time table. The table may be organized by a clustering key that includes (user defined primary key, start time, end time). Thus, a record with a given user defined primary key is clustered next to its earlier versions, which makes "time travel" queries efficient, while implying that "as of" queries will have relatively poor performance. Grouping by time does not help much with "as of" performance, and usually compromises "time travel" performance.

Built-In Approach The built-in approach requires the ability to modify the database engine to provide transaction time support. While a significant barrier, building transaction time support into a database system can greatly improve performance, bringing it close to current time database performance. This improved performance is the result of optimizations that are possible (i) for update, when the timestamps need to be added to versions, (ii) for storage with simple forms of version compression, and (iii) for query, because specialized indexing is possible that improves data clustering. The rest of this article discusses the issues of built-in support and how to make this support perform well.

Managing Versions

Transaction time functionality requires dealing with the multiple versions of database facts that exist to express the states of the database over time. The built-in approach has more freedom than the MWL approach in how to achieve efficiency and performance. There are a number of issues, the more important ones being:

Timestamps Each historical version stored in a transaction time database has a begin time, at which it first became the current version, and an end time, at which it was either replaced by another version or deleted. Current versions have the usual start time, but have a special end time called "now" [2]. An MWL approach usually includes both start and end times with each version so that the SQL queries resulting from translating temporal queries are simple and efficient. With built-in support, most systems [6,13] store only the start time with each version, the end time being derived from the start time of the subsequent version. For a deletion, this next version can be a special "delete stub" version. For a query "as of" time T, the system then looks for the version of each record with the largest start time $\leq$ T.

Storing Versions on a Page The common approach for organizing pages for current time databases is called a slotted array. Each array element points to a record on the page. For B-trees, these records are maintained in B-tree key order. When adding temporal support to a current time database system, it is convenient to minimize the change required for current time functionality. This argues for retaining the slotted array, and back-linking versions where each version is augmented with a pointer to its preceding version. Then for each record accessed in a query, the system follows this backward chain to the first version with a timestamp $\leq$ T, the "as of" time requested.

Indexing Versions Being able to index historical versions by time is essential to avoid increasing costs for ever earlier query times [12]. Postgres [13] used the R-tree [4] for this. The TSB-tree [7] is a more specialized index that can, with an appropriate page splitting policy [1] provide guarantees about the performance of "as of" queries. Its special feature is the introduction of a time split [3] where the time interval of a full database page is partitioned, with record versions being assigned to the resulting pages whenever their lifetimes intersect the time interval of a resulting page. Thus, a version whose lifetime intersects both resulting pages will be replicated in both pages. The result is that, when combined with ordinary B-tree key splitting, each TSB-tree page contains all versions within a key-time rectangle of the search space. This enables identifying exactly which pages can contain answers to a temporal query. Despite the need for replicating versions, the space required for the versions remains linear in the number of unique versions.

Compressing Versions The way that versions are stored on a page and indexed makes compressing versions simple. Usually, an update changes only a small part of a record, perhaps only a single attribute. Thus, delta compression, where the compressed version represents the difference between one version and another that is adjacent in time order, can be very effective. Only the updated attribute together with location information and timestamp needs to appear in the compressed version. Backward delta's are to be preferred because this leaves the current time data uncompressed and hence unchanged, important both for compatibility and current time performance. Because time splits in the TSB-tree always replicate versions spanning the split time, and because splitting at current time is convenient, the last version in each page is always uncompressed, and this is preserved during a time split. Decompressing a version never needs information from any other page than the page upon which the version is stored.

Dealing with Timestamps

In a transaction time database, for a record identified in some way, its versions are distinguished by timestamps. The nature of timestamps, when they are chosen and included, how to optimize this process, and how to deal with user requests for transaction time are discussed below.

Nature of Timestamps Several forms of timestamp have been used for temporal support. Some systems use transaction identifiers (XIDs) instead of time, sometimes maintaining a separate table that maps XID to time. When versioning is limited, e.g., to only support multiversion concurrency control, an active transaction's XID may be mapped to a list of transactions (their XIDs) that committed before them, hence determining which transactions have updates that should be visible to the active transaction [5]. However, for more general functionality, system time is usually used. It may need to be augmented with a sequence number because its granularity may not be sufficient to completely distinguish every transaction's updates.

When to Timestamp A timestamp for a version must enable "as of" queries to always see a transaction consistent view of the data. This can be achieved when timestamp order agrees with the serialization order of transactions. If one chooses timestamps prior to updating, the timestamp can be added immediately to versions generated by the updating. However, early timestamp choice means that transactions that serialize differently must be aborted. Most implementations of transaction time functionality thus choose timestamps at commit, where the commit order is the same as the serialization order for the transactions. This means, however, that the timestamp is not available at time of update, and must be added later.

Lazy Timestamping When a transaction's timestamp is determined late in the transaction, e.g., at commit time, preceding updated records need to be revisited to add the timestamp. Typically, an XID is placed in an updated record at update time, to be replaced later by the system time. Eager timestamping replaces XID with time prior to transaction commit, logging this activity as another update. This can be costly, so a lazy approach is generally preferred in which XID is replaced by time after the transaction commits. The mapping from XID to system time must be maintained persistently, at least until the timestamping is complete, to ensure that replacing XID with time can continue after a possible system crash.

Impact of User Requested Time The SQL language supports a user's request for current (transaction) time within a query. It is essential that the user see a time that is consistent with the transaction timestamp used for updates of the transaction. Providing the user with a time for the transaction while the transaction is executing constrains the choice of timestamp when the transaction is committed [9]. A transaction must be aborted if a timestamp cannot be chosen that is consistent with the time provided to the user. To provide this, the system can exploit the fact that the user time request is usually not for the full precision of the timestamp, e.g., SQL DATE constrains only the date part of the timestamp. Further, remembering the largest timestamp on data that is seen by the transaction provides a lower bound for a possibly non-empty interval for timestamp choice.

Additional Uses

The versions maintained by a transaction time database can support a variety of other system uses.

Snapshot Isolation Recent versions can be used to provide snapshot isolation. With snapshot isolation (the default concurrency provided by Oracle), a transaction reads not the current data (which would be used for a serializable transaction) but a snapshot (version) current as of the start of the transaction or as of the first data read by the transaction. A transaction time database keeps these versions as well as possibly older versions. Thus, efficiently providing transaction time support also provides efficient snapshot isolation. The system may choose to garbage collect older versions more quickly when they are only used for concurrency control purposes.

Online Backup A database backup is simply an earlier state of the database. Transaction time databases make all earlier states accessible and queryable. To use an earlier transaction time database state as a backup for, e.g., media recovery for the current state, requires two things: (i) the earlier state needs to be on a separate medium than the current state; and (ii) the media recovery log needs to include all updates from the earlier state forward to the current state and itself be on a separate device. A transaction time database system can use time splits (which it would use in a TSB-tree) to move versions to separate backup media, and it can do this incrementally as well [8].

Bad User Transactions Occasionally, erroneous transactions commit, compromising the correctness of a database. Point-in-time recovery, where the database state is reset to an earlier time, just prior to the bad transaction, is the usual way of dealing with this. Conventional database backups used for this incur a long restore time followed by a roll forward to the just earlier time. A transaction time database lends itself greatly shortening the outage caused by this problem because earlier versions of the database are already maintained online.

Oracle Flashback [11] implements point-in-time recovery in this way. One can further limit the outage by identifying exactly which transactions should be removed from the database by tracking transaction read dependencies [10].

Key Applications

In addition to the system uses just described, transaction time databases are valuable for several applications, e.g., time series analysis, repeating experiments or analysis on historical data, and auditing and legal compliance.

Future Directions

Data stream processing is a new functionality that has several important applications requiring fast reaction to sequences of events, e.g., stock market data. This data may also be stored and more carefully analyzed. It is quite natural to think about stream data as transaction time data, and ask temporal queries of it.

Cross-references

- ▶ Concurrency Control
- ▶ Multi-Version Database
- ▶ Temporal Database
- ▶ Temporal Strata
- ▶ Transaction
- ▶ Transaction-Time Indexing

Recommended Reading

1. Becker B., Gschwind S., Ohler T, Seeger B., and Widmayer P. An asymptotically optimal multiversion B-tree. VLDB J., 5(4):264–275, 1996.
2. Clifford J., Dyreson C., Isakowitz T., Jensen C.S., and Snodgrass R.T. "On the semantics of "now" in databases," ACM Trans. Database Syst., 22(2):171–214, 1997.
3. Easton M. Key-sequence data sets on inedible storage. IBM J. Res. Dev., 30(3):230–241, 1986.
4. Guttman A. R-trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
5. Hobbs L. and England K. Rdb: A Comprehensive Guide. Digital, 1995.
6. Lomet D.B., Barga R., Mokbel M., Shegalov G., Wang R., and Zhu Y. Transaction time support inside a database engine. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 35.
7. Lomet D.B. and Salzberg B. Access methods for multiversion data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989, pp. 315–324.
8. Lomet D.B. and Salzberg B. Exploiting a history database for backup. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 380–390.
9. Lomet D.B., Snodgrass R.T., and Jensen C.S. Using the lock manager to choose timestamps. In Proc. Int. Conf. on Database Eng. and Applications, 2005, pp. 357–368.
10. Lomet D.B., Vagena Z., and Barga R. Recovery from "bad" user transactions. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 337–346.
11. Oracle. Oracle Flashback Technology (2005) http://www.oracle.com/technology/deploy/availability/htdocs/Flashback_Overview.htm
12. Salzberg B. and Tsotras V.J. A comparison of access methods for time-evolving data. ACM Comput. Surv., 31(2):158–221, 1999.
13. Stonebraker M. The design of the POSTGRES storage system. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 289–300.
14. Tansel U., Clifford J., Gadia S.K., Segev A., and Snodgrass R.T. Temporal databases: theory, design, and implementation. Benjamin/Cummings, 1993.
15. Torp K., Snodgrass R.T., and Jensen C.S. Effective timestamping in databases. VLDB J., 8(4):267–288, 2000.

Surfing

▶ Browsing in Digital Libraries

SVD Transformation

▶ Singular Value Decomposition

SVM

▶ Support Vector Machine

Switch

▶ OR-Split

Symbol Graph

▶ Symbolic Representation

Symbol Plot

▶ Symbolic Representation

Symbolic Graphic

▶ Symbolic Representation

Symbolic Representation

Hans Hinterberger
ETH Zurich, Zurich, Switzerland

Synonyms

Symbolic graphic; Symbol graph; Symbol plot

Definition

A written character or mark used to represent something; a letter, figure, or sign conventionally standing for some object, process etc. (Oxford English Dictionary). Examples are the figures denoting the planets, signs of the zodiac, etc. in astronomy; the letters and other characters denoting elements, etc. in chemistry, quantities, operations, etc. in mathematics, the faces of a crystal in crystallography.

In data visualization, the use of symbols allows the representation of multivariate data items, where each variate contributes to the symbol. The set of symbols may be displayed in an array, superimposed on coordinates to put extra information on a point plot or, if appropriate, on a geographical map.

When symbolic representations of information is used as a tool for thought or a form of communication, one distinguishes between abstract symbols where the graphical units are shapes formed by lines, and areas and depictive symbols (pictograms) where the graphical units are pictorial representations of objects and scenes.

Key Points

Symbols have considerable potential as an aid to support human cognition and as a medium for communication. To what extent symbols can successfully convey specific information and what their inherent limitations as a medium of communication are is far from being understood. There is nevertheless a useful and accumulating body of research dealing with the systematic application of symbolic representations.

One of the earliest researchers to treat symbolic representations in a theoretical context is the French cartographer Jacques Bertin. With his Semiology of Graphics [1] he organized the visual and perceptual elements of graphics according to the features and relations in data, distinguishing primarily between diagrams, networks, maps and symbols.

Among the first efforts to relate visual and perceptual research to the practical problems of designing information displays was the NATO Conference on Visual Presentation of Information in 1978. The contributions to this conference are published in [2], a book that deals with basic psychological issues as well as methods of evaluation of information design, with much room given to the use of symbols.

For a classification of symbols as used for data visualization and an overview for what type of functions different symbols are used the reader is referred to [3]. Tufte, in his series of books on information visualization, discusses with exemplary images and diagrams a wide range of different styles and techniques, good and bad, for the use of symbols, [4] is a starting point.

Cross-references

▶ Data Visualization
▶ Graph
▶ Table
▶ Thematic Map

Recommeded Reading

1. Bertin J. Semiology of Graphics (translation by W.J. Berg), University of Wisconsin Press, Madison, WI, 1983.
2. Easterby R. and Zwaga H. (eds.). Information Design, The Design and Evaluation of Signs and Printed Material, Wiley, London, 1984.
3. Harris R.L. Information Graphics: A Comprehensive Illustrated Reference, Oxford University Press, New York, 1999.
4. Tufte E.R. The Visual Display of Quantitative Information. Graphics Press, Cheshire, CT, 1983.

Symmetric Encryption

Ninghui Li
Purdue University, West Lafayette, IN, USA

Synonyms

Secret-key encryption

Definition

Symmetric encryption, also known as secret key encryption, is a form of data encryption where a single secret key is used for both encryption and decryption.

Key Points

Modern symmetric encryption algorithms are often classified into stream ciphers and block ciphers. In a stream cipher, the key is used to generate a pseudo-random key stream, and the ciphertext is computed by using a simple operation (e.g., bit XOR or modular addition) to combine the plaintext bits and the key stream bits. Many stream ciphers implemented in hardware are constructed using linear feedback shift registers (LFSRs). The use of LFSRs on their own, however, is insufficient to provide good security. Additional variation and enhancement are needed to increase the security of LFSRs. RC4 is the most widely-used software stream cipher and is used in popular protocols such as Secure Sockets Layer (SSL) (to protect Internet traffic) and WEP (to secure wireless networks).

A block cipher operates on large blocks of digits with a fixed, unvarying transformation. The Data Encryption Standard (DES) [1] algorithm uses blocks of 64 bits; the Advanced Encryption Standard (AES) [2] algorithm uses 128-bit blocks. When using a block cipher to encrypt a message, one needs to choose an encryption mode. Commonly used modes include the Electronic Codebook (ECB), Cipher-block chaining (CBC), Cipher feedback (CFB), Output feedback (OFB), and Counter (CTR).

Cross-references

- ▶ Asymmetric Encryption
- ▶ Data Encryption

Recommended Reading

1. Federal information processing standards publication 46-3: Data encryption standard (DES), 1999.
2. Federal information processing standards publication 197: Advanced encryption standard, November 2001.

Synchronization Component

▶ Concurrency Control Manager

Synchronization Join

▶ Workflow Join

Synchronizing Distributed Transactions

▶ Distributed Concurrency Control

Synchronous Join

▶ OR-Join

Synchronous Pipelines

▶ Iterator

Synopsis

- ▶ Structure Indexing
- ▶ Synopsis Structure

Synopsis Structure

Phillip B. Gibbons
Intel Labs Pittsburgh, Pittsburgh, PA, USA

Synonyms

Synopsis

Definition

A *synopsis structure* for a dataset S is any summary of S whose size is substantively smaller than S. Formally, its size is at most $O(|S|^{\varepsilon})$, where $|S|$ is the size (in bytes) of S, for some constant $\varepsilon < 1$.

Key Points

Synopsis structures are small, often statistical summaries of a data set. The term serves as an umbrella for any summarization structure of sufficiently small size, such as random samples, histograms, wavelets, sketches, top-k summaries, etc.

Synopsis structures are most commonly used in conjunction with data streams. The goal is to construct, in one pass over the data stream, a synopsis structure that can be used to answer any query from a prespecified class of queries. That is, at any point, a user may pose a query

Q on the data stream thus far, and a (typically approximate) answer to Q must be produced using only the current synopsis structure. Two key advantages of using a synopsis structure to answer queries are that the space overhead is low (a massive dataset can be summarized using only a small amount of space) and the response time is fast (e.g., disk accesses can be avoided altogether). Moreover, in the common setting of queries comparing or aggregating over a distributed collection of streams, only the small synopsis structures (and not the massive data streams) need to be communicated between the collection points in order to answer the query.

Synopsis structures are also used within relational and XML databases, both for query optimization and for approximate query answering.

Common metrics for evaluating a synopsis structure include (i) *Coverage:* the range and importance of the class of queries supported; (ii) *Answer quality:* the accuracy and confidence of its (approximate) answers; (iii) *Space footprint:* its size, where smaller is better and often polylog space is desired (i.e., space that is $O(\log^k(|S|))$ for some constant $k \geq 1$); (iv) *Per-item processing time:* the total time to process the dataset, normalized to the number of items in the dataset; and (v) *Query time:* the time to answer a query from the synopsis structure [1].

Sometimes *sketch* is used interchangeably with synopsis structure, but more typically "sketch" is restricted to synopses based on random projections.

The term was coined by Gibbons and Matias in 1995.

Cross-references

▶ Data Stream
▶ Histogram
▶ Random Sample
▶ Sketch
▶ Wavelets on Streams

Recommeded Reading

1. Gibbons P.B. and Matias Y. Synopsis data structures for massive data sets. DIMACS Series in Discrete Mathematics and Theoretical Computer Science: External Memory Algorithms, 1999.

Synthetic Data

▶ Matrix Masking

Synthetic Image

▶ Image

Synthetic Microdata

Josep Domingo-Ferrer
Universitat Revira i Virgili, Tarragona, Catalonia

Synonyms

Imputed data; Simulated data; Multiple imputation

Definition

Publication of synthetic – i.e., simulated – data is an alternative to masking for statistical disclosure control of microdata. The idea is to randomly generate data with the constraint that certain statistics or internal relationships of the original dataset should be preserved.

Key Points

The operation of the original proposal by Rubin [2] is next outlined. Consider an original microdata set X of size n records drawn from a much larger population of N individuals, where there are background attributes A, non-confidential attributes B and confidential attributes C. Background attributes are observed and available for all N individuals in the population, whereas B and C are only available for the n records in the sample X. The first step is to construct from X a multiply-imputed population of N individuals. This population consists of the n records in X and M (the number of multiple imputations, typically between 3 and 10) matrices of (B,C) data for the $N - n$ non-sampled individuals. The variability in the imputed values ensures, theoretically, that valid inferences can be obtained on the multiply-imputed population. A model for predicting (B,C) from A is used to multiply-impute (B,C) in the population. The choice of the model is a nontrivial matter. Once the multiply-imputed population is available, a sample Z of n' records can be drawn from it whose structure looks like the one of a sample of n' records drawn from the original population. This can be done M times to create M replicates of (B,C) values. The results are M multiply-imputed synthetic datasets. To make sure no original data are in the synthetic datasets, it is wise to draw the samples from the multiply-imputed population excluding the *n* original records from it.

There are other approaches to synthetic data generation based on bootstrap, Latin Hypercube sampling, Cholesky decomposition, etc. More information can be found in [1].

Cross-references

▶ Inference Control in Statistical Databases

▶ Microdata

Recommeded Reading

1. Hundepool A., Domingo-Ferrer J., Franconi L., Giessing S., Lenz R., Longhurst J., Nordholt E.S., Seri G., and De Wolf P.-P. Handbook on Statistical Disclosure Control. CENEX SDC Project, November 2006 (manuscript version 1.0). http://neon.vb.cbs.nl/CENEX/.
2. Rubin D.B. Discussion of statistical disclosure limitation. J. Off. Stat., 9(2):461–468, 1993.

System Catalog

▶ Data Dictionary

System R (R*) Optimizer

Mouna Kacimi, Thomas Neumann
Max-Planck Institute for Informatics, Saarbrücken, Germany

Definition

The System R Optimizer is the cost-based query optimizer of System R. It pioneered several optimization techniques, including using dynamic programming for bottom-up join tree construction, and the concept of interesting orderings for exploiting ordering in intermediate results. Later, it was generalized for distributed database systems in System R*.

Historical Background

System R is a database management system based on a relational data model that was proposed by E. F. Codd [4] in 1970. The system offers data independence by providing a high-level user interface through which the end user deals with data content rather than the underlying storage structures. In other words, users do not need to know how the tuples are physically stored and which access paths are available to write queries. Thus, data storage structures may change over time without users being aware of it, providing a high level of data independence and user productivity. Moreover, System R offers capabilities for database management in realistic and operational environments. Particularly, it supports multiple users concurrently accessing data, provides means for system recovery after hardware or software failures, supports different types of database use including ad hoc queries, programmed transactions and report generation. System R has been developed at the San Jose IBM Research Laboratory during three phases. First, *Phase Zero* of the project started in 1974 and ended in 1975. It involved the development of a high-level relational user language, called SQL. During this phase, a subset of SQL was implemented for one user at a time. One of the most challenging tasks of Phase Zero was the design of optimizer algorithms for efficient query processing. Second, *Phase One* took place from 1976 to 1977. It involved the development of a full-function multi-user version of System R. The multi-user prototype contained new subsystems, such as locking subsystems that prevent conflict of concurrent user accesses. Finally, *Phase Two* focused on evaluating the System R during 1978 and 1979. It was mainly composed of two parts: (i) evaluation of the system at the San Jose research Laboratory, and (ii) evaluation of the actual use of the system at a number of internal sites of IBM.

Foundations

This section is organized as follows. First, it starts by describing the role of the optimizer in processing SQL statements. Then, it describes the storage components used to access paths on the different relations. Next, it presents optimization techniques for single relations and joins in System R. Finally, it describes how these techniques are generalized to the distributed case of System R*.

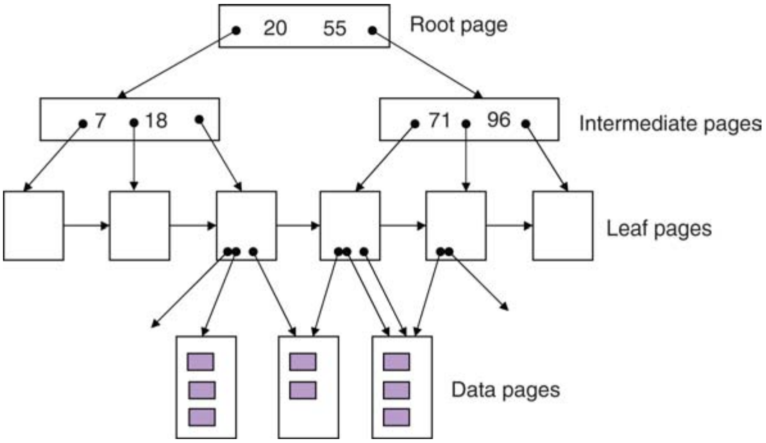
SQL Query Processing

An SQL statement is composed of one or multiple query blocks depending on whether the operands of the used predicates are simple values (of the from "*column operator value*") or queries (of the from "*column operator query*"). A *query block* is represented by: (i) a *SELECT* list containing the list of items to be retrieved, (ii) a *FROM* list containing the relation(s) references and (iii) a *WHERE* tree containing the boolean combination of simple predicates specified by the user. Processing SQL statements requires four main phases namely: parsing, optimization, code generation and execution. During the first phase, each SQL statement is sent to the *Parser* to check its syntax. If no errors are detected, the optimizer component is called in the second phase. Using the System R catalogs, the optimizer verifies the existence of all the relations and columns referenced, and collects information about them. It gets from the catalog the datatype and the length of each column, and use these information to check the semantic errors and type compatibility in both expressions and predicate comparisons of the SQL statement. In addition, the optimizer obtains, from the catalogs, statistics about the referenced relations and the access paths available on each of them and use these information for access path selection process. Then, the optimizer chooses a query plan, from a tree of alternate path choices, that is a minimum cost solution. This chosen plan is a tree represented in the Access Specification Language (ASL) [7]. In the third phase, the Code generator translates ASL trees into machine language code to finally execute the plan chosen by the optimizer in the fourth phase. When the code is executed, it accesses the Relational Storage System (RSS), via the Storage System Interface (RSI), to scan each of the query relations using the access paths chosen by the optimizer. Even though the RSS may be used for different purposes, here, we focus on its use for computing cost formulas and executing the code generated by the query processing in System R.

Relational Storage System

The Relational Storage System (RSS) provides underlying storage support for System R. Relations in the RSS are stored as a collection of tuples whose columns are physically contiguous. The storage space is logically organized into segments. Each relation resides within a single segment, however, a segment may contain one or more relations. A segment is composed of a set of equal-sized pages. Each tuple of a relation is stored within a single page and is assigned an identification of the relation to which it belongs. Note that a page may contain tuples from one or more relations.

To access tuples in a relation, an RSS scan is used. Along a given access path, a scan returns one tuple at a time. Two different types of scans can be distinguished. First, *Segment* scans find all the tuples of a given relation by examining the segment that contains the relation. All the non-empty pages of the segment are touched and the tuples belonging to the given relation are returned. Second, *Index* scans access a relation in value order using an index. An index is created on one or more columns of a relation. It is composed of one or more pages within the segment containing the relation. The pages of the index are separated from the pages containing the relation tuples. They are organized into a B-tree structure as shown in Fig. 1. Each page is a node of the B-tree that contains an ordered sequence of index entries. An entry of a non-leaf node consists of a $\prec$ *key, pointer* $\succ$ pair, where the pointer address another page in the same tree. Leaf pages contain sets of $\prec$ *key, identifier* $\succ$ pairs, where identifier indicates the tuple that contains the corresponding key. An



System R (R*) Optimizer. Figure 1. Example of B-tree index structure.

index scan does a sequential read along the leaf pages to get the tuple identifiers matching a given key. These identifiers are used to find and return the data tuples to the user in key value order.

When an index scan examines a relation, each page of the index is touched only once, but a data page may be touched more than once. This case may happen if a page contains tuples which are not close in the index ordering. An index is said to be clustered when the physical proximity of tuples in the same data page corresponds to the index key value. Therefore, not only each index page, but also each data page will be touched only once in a scan on that index. Additionally, to reduce the number of touched pages, starting and stopping key values can be specified when scanning tuples. Thus, only the tuples whose keys belong to the predefined interval are returned. Both index and segment scans may use a set of predicates, called also search arguments or SARGS. These predicates are of the form ("*column operator value*"). They are applied to tuples before they are returned to the RSI caller. Sargable predicates play an important role in reducing the cost by eliminating unnecessary RSI calls for tuples which can be rejected within the RSS.

Access Path Selection for Single Relations

This section describes how the optimizer chooses a query plan accessing a single relation. As presented previously, the optimizer gets from the System R catalog the access paths available on the relation referenced by the query. The cheapest access path is obtained by evaluating the cost for each available access path, i.e., each index on the relation plus a segment scan. The optimizer formulates a cost prediction given by the following formula:

$$Cost = Page\ Fetches + W \times (RSI\ calls)$$

where *Page Fetches* represent *I/O* requirement computed by the number of index pages fetched plus number of data pages fetched. *RSI calls* indicate the predicted number of tuples returned from the RSS. This number is a good approximation of *CPU* utilization since most of System R's CPU time is spent in RSS. The parameter *W* is an adjustable weighting factor between *I/O* and *CPU*.

To find the cheapest access plan for a single relation query, the optimizer needs to examine the cheapest *unordered* access path and the cheapest access path producing tuples in each *interesting* order. *Unordered* access path may produce tuples in some order, but the order is not *interesting*. In this case, the optimizer simply chooses the cheapest access path as query plan. By contrast, a tuple order is an *interesting order* if that order is specified by the query block using GROUP BY or ORDER BY clauses. In this case, the optimizer compares the cost of producing that interesting order to the cost of the cheapest unordered path plus the cost of sorting QCARD tuples into the proper order, where QCARD represents the query cardinality. The cheapest of these alternatives is chosen as the plan for the query block. In the following, a description is given of how the query cardinality (QCARD) and RSI calls are computed by the query optimizer.

During the query processing, the optimizer gets statistics on the relations of the query and access paths available on each relation. These statistics include the cardinality of the query relations, the number of segments holding the relevant relations and the fraction of their contained pages, the number of distinct keys and the number of pages of each index. The statistics are used by the optimizer to assign a *selectivity factor F* to each predicate of the WHERE tree. This selectivity factor indicates the expected fraction of tuples which will satisfy the predicate. More details on statistics and selectivity factors are given in [8]. The optimizer computes the query cardinality (QCARD) as the product of the cardinalities of every relation in the query block's FROM list times the product of all the selectivity factors of that query block's predicates. The number of RSI calls (RSICARD) is the product of the relation cardinalities times the selectivity factors of the sargable predicates, since the sargable predicates filter out tuples without returning across the RSS interface.

Access Path Selection for Joins

A join query in SQL combines tuples from more than one relation. Two join methods have been identified as optimal or nearly optimal in most cases. For simplicity, a description of how to join two relations is given, then, it is extended to *n* relations. A two-way join involves two relations respectively called *outer* relation and *inner* relation. A predicate that relates columns of two relations to be joined is called *join predicate*. The columns referenced in a join predicate are called *join columns*. Consider the following example:

```
SELECT Name, Location
FROM STUDENT, DEPARTMENT
WHERE STUDENT.Department=DEPARTMENT.Num
```

In this example, the outer and inner relations are respectively "STUDENT " and "DEPARTMENT". There is one join predicate that is "STUDENT.Departement = DEPARTMENT.Num." The join columns of this query are "STUDENT.Department" and "DEPARTMENT.Num".

The first supported join method is called *nested loops*. This method scans the outer and the inner relations in any order. The scan on the outer relation is opened and for each outer tuple obtained, a scan is opened on the inner relation to retrieve, one at a time, all the tuples of the inner relation that satisfy the join predicate. The cost of a nested loop join is computed from the costs of scans on single relations defined in the previous section and is given by the following formula:

$$\begin{aligned}&\text{C-nested-loop-join }(\text{path1},\ \text{path2})\\&\quad = \text{C-outer}(\text{path1}) + \text{N} \times \text{C-innter}(\text{path2})\end{aligned}$$

where *C-outer(path1)* indicates the cost of scanning the outer relation via path1, *C-inner(path2)* indicates the cost of scanning the inner relation, applying all applicable predicates, and *N* is the (product of the cardinalities of all relations R of the join so far) × (product of the selectivity factors of all applicable predicates).

The second supported join method is called *merging scans*. It requires the outer and the inner relations to be scanned in join column order. This means that join columns define *interesting* orders in addition of columns mentioned in GROUP BY and ORDER BY clauses. In case the join query contains more than one predicate, one of them is used as join predicate and the others are treated as ordinary predicates. If a relation has no index on the join column, it has to be sorted into a temporary list ordered by join column. By using ordering on join columns, the merging scan method avoids rescanning the entire inner relation for each tuple of the outer relation. The merging scan synchronizes the inner and the outer scans by matching join columns. In addition, it may take advantage of clustering on join column of the inner relation. Thus, the merging scan can remember where matching join groups are located since tuples having the same values on a join column are physically close to each other. The cost of a merge scan join can be divided into the cost of actually doing the join plus the cost of sorting the outer or inner relation if required. The cost of doing the merge is given by:

$$\begin{aligned}&\text{C-merge }(\text{path1},\ \text{path2}) = \text{C-outer}(\text{path1})\\&\quad + \text{N} \times \text{C-innter}(\text{path2})\end{aligned}$$

In case the inner relation is sorted into a temporary relation, the merging scans do not scan the entire relation looking for a match. Therefore, the cost of the inner scan can be significantly reduced comparing to nested-loop joins. The cost of the inner scan, in this case, is given by the following formula:

$$\begin{aligned}&\text{C-inner}(\text{sorted list}) = \text{TEMPPAGES}/\text{N}\\&\quad + \text{W} \times \text{RSICARD}\end{aligned}$$

where TEMPPAGES is the number of pages required to hold the inner relation. RSICARD is the number of RSI calls and W is an adjustable weighting factor between *I/O* and *CPU*.

When optimizing larger queries, these join methods are used as building blocks. The System R optimizer only considers linear join trees, where join operators may occur only on one side of other join operators. Join trees with more than two relations are therefore constructed by adding a new relation to an already existing join tree. For applying the methods described above, the join tree, which is treated like a composite relation, represents the *outer* relation and the relation being added to the join tree represents the *inner* relation. The optimizer uses a dynamic programming (DP) strategy (sketched in Fig. 2) to reduce the runtime complexity for join tree construction. It organizes the optimization by the size of the join tree, initializing the DP table with single relations and then constructing larger join trees by combining smaller join trees (S_l) with new relations (R_r). For each combination of relations the best join tree found so far is stored in the DP table, which reduces the search space from $O(n!)$ to $O(n2^n)$. Further, the System R optimizer considers only join trees where a join predicate between the smaller join tree and the new relation exists (i.e., it avoids cross-products). This further reduces the search space, for example to $O(n^3)$ when the relations are simply joined in a sequence.

An important aspect of this optimization strategy is the consideration of *interesting orders*. To find solutions for joining pairs of relations, the optimizer first finds access paths for each single relation in each interesting and non-interesting tuple ordering. Recall that interesting orders are defined by GROUP BY and

```
JoinOrder ($R = \{R_1,...,R_n\}$)
for each $R_i \in R$
    dpTable[{$R_i$}] = $R_i$
for each $1 < s \le n$ ascending // size of the join tree
    for each $S_l \subset R$, $R_r \in (R \backslash S_l)$ : $|S_l| = s \wedge$dpTable[$S_l$] $\neq \phi$
            if$\neg$($S_l$ can be joined with $R_r$) continue
            $p$ = dpTable[$S_l$]$\bowtie$dpTable[{$R_r$}]
            if dpTable[$S_l \cup \{R_r\}$] = $\phi \vee$ cost($p$)<cost(dpTable[$S_l \cup \{R_r\}$])
                dpTable[$S_l \cup$ {$R_r$} = p
return dpTable[{$R_1$,...,$R_n$}]
```

System R (R*) Optimizer. Figure 2. Dynamic programming strategy for join tree construction.

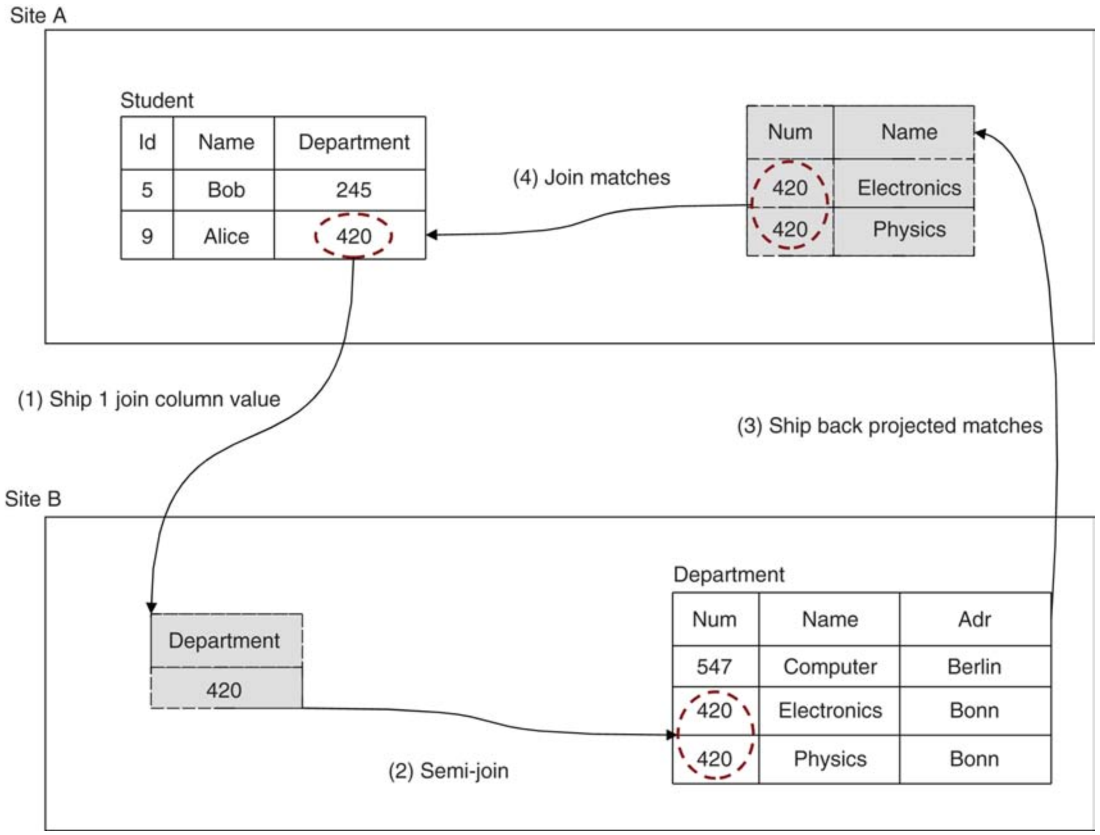
ORDER BY clauses and also by every join column. Next, the optimizer finds the best way for joining any two relations, and starts building larger join trees. For each join tree the order of the composite result is saved to allow for merge joins that would not require sorting the composite result. As the ordering can affect later operators, a plan can only be safely pruned if a cheaper one, which satisfies the same interesting ordering, is found. After join trees for all n relations have been constructed, the optimizer chooses the cheapest solution that gives the required order specified by the query. Consider an example of three relations R_1, R_2, R_3 in a query and the following join predicates $R_1.x = R_2.x$ and $R_2.x = R_3.x$. Assume that the costs of nested-loop and merge scan for the subquery $\{R_1, R_2\}$ are respectively C_1 and C_2, where C_1 is lower than C_2. Intuitively, when the optimizer looks for the best plan for $\{R_1, R_2, R_3\}$, it could consider the nested-loop method to join $\{R_1, R_2\}$ since it is the cheapest alternative. However, if the optimizer considers a merge scan to join $\{R_1, R_2\}$, the composite result will be sorted on x which may significantly reduce the cost of the join with R_3. Thus, the optimizer has to keep track of tuple orderings that can affect the execution plans for the given query to find the optimal join tree.

R* Optimizer

The optimization algorithms described previously have been extended to efficiently process queries in a distributed database management system (R*). In such environment, data needed by queries are stored in multiple sites. Two main factors distinguish query processing in System R from processing query in System R* [6]. First, the communication delays, and second, the possibility of concurrent processing on multiple sites. These two factors raise the importance of developing an R* optimizer to deal with increasing complexity of distributed query processing.

The distribution unit in R* is a relation and each relation is stored at one site. Figure 3 shows two relations STUDENT and DEPARTMENT stored in two different sites *A* and *B*. A query is called *distributed* if it refers to relations at sites other than the query site. The simplest form of a distributed query is a query that accesses a single relation at a remote site. To execute the query, a process at the remote site accesses the relation locally and ships the query result back to the query site. In case of a join query, the R* optimizer needs to chose a set of local and distributed parameters. The local parameters are the same as the one considered by the System R optimizer including the join method (nested-loop or merge scan), the order in which relations must be joined and the access path for each relation (index or segment scan). The distributed parameters include the choice of the join site, i.e., the site at which the join will take place, and the method for transferring a copy of the inner table to the join site, in case the inner table is not stored in the chosen join site.

R* optimizer can use different methods to transfer tuples from a site to another one. A straightforward strategy is to ship the entire relation to the join site and store it there in a temporary table. Alternatively, the R* optimizer can use other join methods such as semijoins, joins using hashing (Bloom) filters and joins using dynamically-created index. Semijoins, for example, help in reducing the number of transferred tuples by limiting the relevant domain. Specifically, only the tuples that could potentially match the join predicates are shipped to the join site. Figure 3 shows an example of semijoin procedure. First, it projects the outer relation to the join column and ships the results to the site of the inner relation. Second, it finds tuples from the inner relation that match the values received from the outer relation. Third, it ships a copy of the projected inner tuples to the join



System R (R*) Optimizer. Figure 3. Example of semijoin.

site. Last, it joins the received tuples to the outer relation.

Key Applications

The System R optimizer inspired many later optimizers, including the well known Starburst optimizer. Starburst uses a similar (though much more generalized) bottom-up constructive optimization technique, and eventually become the commercial database system DB2.

Cross-references

► Query Optimization (in Relational Databases)

Recommended Reading

1. Astrahan M.M., Blasgen M.W., Chamberlin D.D., Eswaran K.P., Gray J., Griffiths P.P., III W.F.K., Lorie R.A., McJones P.R., Mehl J.W., Putzolu G.R., Traiger I.L., Wade B.W., and Watson V. System R: Relational approach to database management. ACM Trans. Database Syst., 1(2):97–137, 1979.
2. Chamberlin D.D., Astrahan M.M., Blasgen M.W., Gray J., III W.F.K., Lindsay B.G., Lorie R.A., Mehl J.W., Price T.G., Putzolu G.R., Selinger P.G., Schkolnick M., Slutz D.R., Traiger I.L., Wade B.W., and Yost R.A. A history and evaluation of system R. Commun. ACM, 24(10):632–646, 1981.
3. Chamberlin D.D. and Boyce R.F. SEQUEL: A Structured English Query Language. In Proc. SIGMOD Workshop, Vol. 1. 1974, pp. 249–264.
4. Codd E.F. A relational model of data for large shared data banks. Commun. ACM, 13(6):377–387, 1970.
5. Gray J. Notes on data base operating systems. In Advanced Course: Operating Systems, 1978, pp. 393–481.
6. Lohman G.M., Mohan C., Haas L.M., Daniels D., Lindsay B.G., Selinger P.G., and Wilms P.F. Query processing in R*. In Query Processing in Database Systems, Springer, 1985, pp. 31–47.
7. Lorie R.A. and Nilsson J.F. An access specification language for a relational data base system. IBM J. Res. Dev., 23(3):286, 1979.
8. Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., and Price T.G. Access path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.

System Recovery

► Crash Recovery

T

Table

Hans Hinterberger
ETH Zurich, Zurich, Switzerland

Synonyms

Table; List; Matrix

Definition

A data structure to organize the tuples of a relation: {<$value_{ix}$, $value_{iy}$,...>, <$value_{jx}$, $value_{jy}$,...>, ...} in a structured way.

An arrangement of numbers, words, symbols or items of any kind, in columns and rows.

One-way table. The values of one or more variables (sets of data) are displayed either horizontally or vertically as a list. Each row or each column represents one data item.

Two-way table. Also known as a contingency table or cross tabulation (cross tab). The rows are labeled with values of one of two variables and the columns with values of the other. The cell at each row and column intersection displays the value of a characteristic of the data that is shared by the two variables when they have the value displayed in the row and column headings.

Multi-way table. The general idea of cross tabulation extended to more than two variables. The rows or columns or both are grouped by values of additional variables.

Key Points

In database terminology a (one-way) table is the data structure used to represent data that are organized with the relational model.

Two-way and multi-way tables are widely used tools to explore data by examining frequencies of observations that belong to specific categories on more than one variable.

In the context of data visualization, tables are an effective way to show exact numerical values and are usually better suited than graphical methods to represent small data sets. They assist users in making comparisons and provide them with a convenient way of storing data for rapid reference. Tables can be formatted in many different ways to suit the type of data, the purpose of the table and its intended use.

Ehrenberg [3] discusses tables in the context of readability, Bertin [1] considers tables as part of an information processing system, Card et al. [2] use them as a step in the process of mapping data to visual form while Harris [4] lists terms and key elements used in the design of tables.

The terminology for tables has also been influenced by the widespread use of table-based spreadsheet programs. One example is the notion of a *pivot table*, a practical feature to reorganize lists as user specified tables, particularly useful for cross tabulations.

Cross-references

► Chart
► Data Visualization
► Tabular Data

Recommended Reading

1. Bertin J. Graphics and Graphic Information-Processing. Walter de Gruyter, Berlin, New York, 1981.
2. Card S.K., MacKinlay J.D., and Shneiderman B. Readings in Information Visualization: Using Vision to Think. Morgan Kaufmann, San Francisco, CA, 1999.
3. Ehrenberg A.S.C. A Primer in Data Reduction. Wiley, Chichester, UK, 1982.
4. Harris R.L. Information Graphics: A Comprehensive Illustrated Reference, Oxford University Press, New York, 1999.

Table Design

► Physical Database Design for Relational Databases

Table Normalization

► Physical Database Design for Relational Databases

Tabular Data

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona, Catalonia

Synonyms

Table

Definition

From microdata, *tabular data* can be generated by crossing one or more categorical attributes. Formally, a table is a function

$$T : D(V_{i1}) \times D(V_{i2}) \times \cdots \times D(V_{il}) \rightarrow \mathbb{R} \text{ or } \mathbb{N}$$

where $l \leq t$ is the number of crossed categorical attributes V_{ij} and D(V_{ij}) is the domain of attribute V_{ij}.

Key Points

There are two kinds of tables: *frequency tables* that display the count of respondents at the crossing of the categorical attributes (in $\mathbb{N}$) and *magnitude tables* that display information on a numerical attribute at the crossing of the categorical attributes (in $\mathbb{R}$). For example, given some census microdata containing attributes "Job" and "Town," one can generate a *frequency table* displaying the count of respondents doing each job type in each town. If the census microdata also contain the "Salary," attribute, one can generate a magnitude table displaying the average salary for each job type in each town. The number n of cells in a table is normally much less than the number r of respondent records in a microdata file. However, tables must satisfy several linear constraints: marginal row and column totals. Additionally, a set of tables is called *linked* if they share some of the crossed categorical attributes: for example "Job" × "Town" is linked to "Job" × "Gender."

Cross-references

► Inference Control in Statistical Databases
► Microdata

Tamper-Proof Hardware

► Trusted Hardware

Tape Libraries

► Storage Devices

Tapes

► Storage Devices

Task

► Activity

Taxonomies

► Lightweight Ontologies

Taxonomy: Biomedical Health Informatics

Vipul Kashyap
Partners Healthcare System, Wellesley, MA, USA

Synonyms

Health informatics; Healthcare informatics; Biomedical informatics

Definition

Health informatics or medical informatics is the intersection of information science, computer science, and health care [2]. It deals with the resources, devices, and methods required to optimize the acquisition, storage, retrieval, and use of information in health and biomedicine. Health informatics tools include not only computers but also clinical guidelines, formal medical terminologies, and information and communication systems. Subdomains of (bio) medical or health care informatics include: clinical informatics, nursing informatics, imaging informatics, consumer health informatics, public health informatics, dental informatics, clinical research informatics, bioinformatics, veterinary informatics, pharmacy informatics, and healthcare management informatics. An alternative characterization refers to this field as biomedical informatics [1] takes a broader perspective including the application of computer and information science, informatics, cognitive science and human computer interaction in the practice of biological research, biomedical science, medicine and healthcare [3].

Foundations

This field is now characterized by means of a taxonomy comprising of the various subdomains identified above. The key sub domains are:

1. *Clinical Informatics* focuses on computer applications that address medical data (collection, analysis, representation), and is a combination of information science, computer science, and clinical science designed to assist in the management and processing of data, information and knowledge to support the practice and delivery of clinical care. Key activities covered by this area include Electronic Medical Records (EMRs), Decision Support Systems, Medical Data Mining, Hospital Information Systems and Laboratory Information Systems.
2. *Nursing Informatics* is the multidisciplinary scientific endeavor of analyzing, formalizing, and modeling how nurses collect and manage data, process data into information and knowledge, make knowledge-based decisions and inferences for patient care, and use this empirical and experiential knowledge in order to broaden the scope and enhance the quality of their professional practice. The scientific methods central to nursing informatics are focused on: (i) Using a discourse about motives for computerized systems, (ii) Analyzing, formalizing and modeling nursing information processing and nursing knowledge for all components of nursing practice: clinical practice, management, education and research, (iii) Investigating determinants, conditions, elements, models and processes in order to design, and implement as well as test the effectiveness and efficiency of computerized information, (tele)communication and network systems for nursing practice, and (iv) Studying the effects of these systems on nursing practice.
3. *Imaging Informatics* combines knowledge and capabilities from the fields of medicine, medical imaging (also known as diagnostic radiology), biomedical informatics and Information Technology. Imaging informatics includes the mining of information or knowledge from medical image databases, the use of technology to enhance the medical image interpretation process, and the utility of computer software to digitize medical imaging. The more commonly recognized areas of Imaging Informatics include Picture Archiving and Communications Systems (PACS), Radiology Information Systems (RIS), and Computer-Aided Detection and Diagnosis (CAD).
4. *Consumer Health Informatics* has been defined as a branch of medical informatics that analyses consumers' needs for information; studies and implements methods of making information accessible to consumers; and models and integrates consumers' preferences into medical information systems.
5. *Public Health Informatics* has been defined as the systematic application of information and computer science and technology to public health practice, research, and learning. It is distinguished from healthcare informatics by emphasizing data about populations rather than that of individuals. The activities of public health informatics can be broadly divided into the collection, storage, and analysis of data related to public health.
6. *Dental Informatics* is the understanding, skills and tools that enable the sharing and use of information to promote oral health and improve dental practice, research, education, and management. It encompasses electronic health records, CAD/CAM technology, diagnostic digital imaging and administrative information for all dentistry disciplines.
7. *Clinical Research Informatics* is concerned with the application of informatics theory and methods to design, conduct and improve clinical research and disseminate the knowledge gained. It overlaps considerably with the related rapidly developing domain of Translational Research Informatics. Clinical research is defined by the National Institutes of Health (NIH) as being comprised of studies and trials in human subjects that fall into the three sub-categories: (i) Patient-oriented research conducted with human subjects (or on material of human origin such as tissues, specimens and cognitive phenomena) for which an investigator (or

colleague) directly interacts with human subjects. It includes research on mechanisms of human disease, therapeutic interventions, clinical trials, or development of new technologies; (ii) Epidemiologic and behavioral studies; and (iii) Outcomes research and health services research.

8. *Translational Research Informatics* as defined by the NIH includes two areas of translation. One is the process of applying discoveries generated during research in the laboratory, and in preclinical studies, to the development of trials and studies in humans. The second area of translation concerns research aimed at enhancing the adoption of best practices in the community. Cost-effectiveness of prevention and treatment strategies is also an important part of translational science.
9. *Bioinformatics and Computational Biology* involves the use of techniques including applied mathematics, informatics, statistics, computer science, artificial intelligence, chemistry, and biochemistry to solve biological problems usually on the molecular level. The core principle of these techniques is the use of computing resources in order to solve problems on scales of magnitude far too great for human discernment. Research in computational biology often overlaps with systems biology. Major research efforts in the field include sequence alignment, gene finding, genome assembly, protein structure alignment, protein structure prediction, prediction of gene expression and protein-protein interactions, and the modeling of evolution.
10. *Pharmacy Informatics* is the application of computers to the storage, retrieval and analysis of drug and prescription information. Pharmacy informaticists work with pharmacy information management systems that help the pharmacist make excellent decisions about patient drug therapies with respect to medical insurance records, drug interactions, as well as prescription and patient information. Pharmacy informatics is the study of interactions between people, their work processes and engineered systems within health care with a focus on pharmaceutical care and improved patient safety.

Key Applications

This field is characterized by the key aspects and applications, some of which are discussed in more detail through cross-referenced encyclopedia entries:

- Systems and architectures for electronic medical records.
- Health information systems used for billing, scheduling and other financial operations.
- Information systems used for biomedical and clinical research.
- Decision support systems in healthcare, including clinical decision support systems.
- Standards (e.g., DICOM, HL7) and integration profiles (e.g., Integrating the Healthcare Enterprise) to facilitate the exchange of information between healthcare information systems.
- Information Models such as the HL7/RIM and the openEHR for storage and representation of clinical data in a standardized manner.
- Controlled medical vocabularies (CMVs) such as the Systematized Nomenclature of Medicine, Clinical Terms (SNOMED CT), Logical Observation Identifiers Names and Codes (LOINC), OpenGALEN Common Reference Model or the highly complex UMLS used to allow a standardized way of characterizing medication, laboratory results, and clinical findings.
- Use of hand-held or portable devices to assist providers with data entry/retrieval or medical decision-making, sometimes called mHealth.

Cross-references

▶ Biomedical Data/Content Acquisition, Curation
▶ Biomedical Image Data Types and Processing
▶ Biomedical Scientific Textual Data Types and Processing
▶ Clinical Data Acquisition, Storage and Management
▶ Clinical Data and Information Models
▶ Clinical Data Quality and Validation
▶ Clinical Decision Support
▶ Clinical Document Architecture
▶ Clinical Event
▶ Clinical Knowledge Repository
▶ Clinical Observation
▶ Clinical Order
▶ Computerized Physician Order Entry
▶ Data Privacy and Patient Consent
▶ Data, Text, and Web Mining in Healthcare
▶ Electronic Health Record
▶ Enterprise Terminology Services
▶ Evidence Based Medicine
▶ Executable Knowledge

- ▶ Implications of Genomics for Clinical Informatics
- ▶ Interface Engines in Healthcare
- ▶ Quality and Trust of Information Content and Credentialing
- ▶ Reference Knowledge
- ▶ Storage Management

Recommended Reading

1. Biomedical Informatics, http://en.wikipedia.com/wiki/Biomedical_informatics
2. Health Informatics, http://en.wikipedia.org/wiki/Health_informatics
3. Shortliffe E.H. and Cimino J.J. eds. Biomedical Informatics: Computer Applications in Health Care and Biomedicine, 3rd edn. Springer, New York, 2006.

Telic Distinction in Temporal Databases

Vijay Khatri[1], Richard T. Snodgrass[2], Paolo Terenziani[3]
[1]Indiana University, Bloomington, IN, USA
[2]University of Arizona, Tucson, AZ, USA
[3]University of Turin, Turin, Italy

Synonyms

Point-versus period-based semantics

Definition

In the context of temporal databases, telic (atelic) data are used to store telic (atelic) facts, and the distinction between telic and atelic data are drawn using the properties of downward and upward inheritance.

- *Downward inheritance*. The *downward inheritance* property implies that one can infer from temporal data d that holds at valid time t (where t is a time period) that d holds in any sub-period (and sub-point) of t.
- *Upward inheritance*. The *upward inheritance* property implies that one can infer from temporal data d that holds at two consecutive or overlapping time periods t_1 and t_2 that d holds in the union time period $t_1 \cup t_2$.

In temporal databases, the semantics of atelic data implies that both downward and upward inheritance holds; on the other hand, neither downward nor upward inheritance holds for telic data.

Historical Background

The distinction between telic and atelic facts dates back to Aristotle's categories [2] and has had a deep influence in the Western philosophical and linguistic tradition. In particular, the distinction between different classes of sentences (called aktionsart classes) according to their linguistic behavior and temporal properties is at the core of the modern linguistic tradition (consider, e.g., the milestone categorization by Vendler [11]). For instance, the upward and downward inheritance properties were used (without adopting terminology, which is imported from Shoham [7] and, more generally, from the artificial intelligence tradition) by Dowty [4] in order to distinguish between Vendler's accomplishments (telic facts) and states along with processes (atelic facts). Starting from the pioneering work by Bennet and Partee [3], several linguistic approaches have pointed out that the traditional point-based semantics, in which facts can be evaluated at each time point, properly applies only to atelic facts, while a period-based semantics is needed in order to properly cope with telic facts. Starting from Allen's milestone approach [1], the telic/atelic dichotomy has also played a major role in the area of artificial intelligence. In the field of temporal databases, the point-based vs. period-based dichotomy was initially related to representation and query evaluation issues (rather than to data semantics); later the connection was made between Aristotle's categories and the telic/atelic data semantics [9,10]. It is the emphasis on data semantics that renders "telic" and "atelic" the preferred term.

Foundations

The distinction between telic and atelic data regards the time when facts hold or occur, i.e., their valid time. The following discussion focuses on the temporal *semantics* of data and queries, independent of the *representation* that is used for time. Moreover, while in several database approaches the semantics of data is not distinguished from the semantics of the query, this presentation follows the logical tradition, stating that data has its own semantics independently of any query language and operators just in the same way in which a knowledge base of logical formulæ have their own semantics – usually expressed in model-theoretic terms.

Data Semantics

In the linguistic literature, most approaches classify facts (or sentences describing facts) according to their temporal properties. In particular, most approaches distinguish between telic and atelic facts, and prior research (see, e.g., [3]) points out that, while the point-based semantics is useful to cope with atelic facts, it is not suitable for telic ones, for which a period-based semantics is needed.

Point-based semantics of data: The data in a temporal relation is interpreted as a sequence of states (with each state a conventional relation, i.e., a set of tuples) indexed by points in time. Each state is independent of every other state.

Such temporal relations can be encoded in many different ways (data language). For example the following are three different encodings of the same information, within a point-based semantics, of John being married to Mary in the states indexed by the times 1, 2, 7, 8, and 9:

$$\text{(i)} < \text{John, Mary}||\{1,2,7,8,9\} >\in \text{R}$$

$$\text{(ii)} < \text{John, Mary}||\{[1-2],[7-9]\} >\in \text{R}$$

$$\text{(iii)} < \text{John, Mary}||[1-2] >\in \text{R and}$$
$$< \text{John, Mary}||[7-9] >\in \text{R}$$

Independently of the representation, the point-based semantics implies that the fact denoted by

<John, Mary> includes 5 individual states as follows:

$$\begin{aligned} 1 &\rightarrow \{< \text{John, Mary} >\}\\ 2 &\rightarrow \{< \text{John, Mary} >\}\\ 7 &\rightarrow \{< \text{John, Mary} >\}\\ 8 &\rightarrow \{< \text{John, Mary} >\}\\ 9 &\rightarrow \{< \text{John, Mary} >\} \end{aligned}$$

Notice that the point-based semantics naturally applies to atelic facts, since both downward and upward inheritance are naturally supported.

Period-based semantics of data: Each tuple in a temporal relation is associated with a multiset of time periods, which are the temporal extents in which the fact described by the tuple occur. In this case, time periods are atomic primitive entities in the sense that they cannot be decomposed. Note, however, that time periods can overlap, unlike time points.

For example, let <John || {[10–20]}> represent the fact that John started to build a house at time 10 and finished at time 20. If a period-based semantics is adopted, the period [10–20] is interpreted as an atomic (indivisible) one.

$$[10, 20] \rightarrow \{< \text{John} >\}$$

Note that a period-based semantics does not imply that John built the house in [12–15], or at the time point 12, or at any other time period other than [10–20]. As a consequence, the period-based semantics is naturally suited to cope with telic facts, for which (by definition) neither downward nor upward inheritance hold.

Although several query and data representation languages include time periods, most temporal database approaches adopt, explicitly or implicitly, the point-based semantics, interpreting a temporal database as a set of conventional databases, each one holding at a specific snapshot of time. This is the approach followed, e.g., by the Bitemporal Conceptual Data Model (BCDM) [5], a model that has been proven to capture the semantic core of many prior approaches in the temporal database literature, including the TSQL2 "consensus" approach [8]. While point-based semantics works perfectly when coping with atelic data, the problem with using it to cope with telic fact is illustrated by the following example.

Example

Phone calls are durative telic facts. For instance, if John made a call to Mary from time 10 to time 12, he didn't make it from 10 to 11. Similarly, two consecutive calls, one from 10 to 12 (inclusive) and the other from 13 to 15 (inclusive), are clearly different from a single call from 10 to 15. However, such a distinction cannot be captured at the semantic level, if the point-based semantics is used. In fact, the point-based semantics for the two phone calls of John is as follows:

$$\begin{aligned} 10 &\rightarrow \{< \text{John, Mary} >\}\\ 11 &\rightarrow \{< \text{John, Mary} >\}\\ 12 &\rightarrow \{< \text{John, Mary} >\}\\ \\ 13 &\rightarrow \{< \text{John, Mary} >\}\\ 14 &\rightarrow \{< \text{John, Mary} >\}\\ 15 &\rightarrow \{< \text{John, Mary} >\} \end{aligned}$$

Based on point-semantics, there is no way of grasping that two different calls were made. In other words, there is a loss of information. Note that such a loss of information is completely independent of the representation language used to model data. For instance, the above example could be represented as

$$\text{(i)} < \text{John, Mary}||\{10, 11, 12, 13, 14, 15\} > \in \text{R}$$

$$\text{(ii)} < \text{John, Mary}||\{[10-12], [13-15]\}\} > \in \text{R}$$

$$\text{(iii)} < \text{John, Mary}||[10-12] > \in \text{R and} \\ < \text{John, Mary}||[13-15] > \in \text{R}$$

But, as long as the point-based semantics is used, the data semantics is the one elicited above.

On the other hand, independently of the representation formalism being chosen, the semantics of telic facts such as phone calls is properly coped with if the period-based semantics is used. For instance, in the phone example, the semantics

$$[10-12] \rightarrow \{< \text{John, Mary} >\} \\ [13-15] \rightarrow \{< \text{John, Mary} >\}$$

correctly expresses the distinction between the two consecutive phone calls.

In an analogous way, period-based semantics is not suitable to model atelic facts. In short, both upward and downward inheritance holds for them, and the period-based semantics does not support such properties.

Terenziani and Snodgrass have proposed a two-sorted data model, in which telic data can be stored in telic relations (i.e., relations to be interpreted using a period-based semantics) and atelic data in atelic relations (i.e., relations to be interpreted using a point-based semantics) [9].

Query Semantics

It is interesting that the loss of information due to the treatment of telic data in a point-based (atelic) framework is even more evident when queries are considered. Results of queries should depend only on the data semantics, not on the data representation. For instance, considering the phone example above (and independently of the chosen representation), queries about the number or duration of phone calls would not provide the desired answers. For instance, the number of calls from John to Mary would be one, and a call from John to Mary would (incorrectly) be provided to a query asking for calls lasting for at least five consecutive units.

In order to cope with a data model supporting both telic and atelic relations, temporal query languages must be extended. Specifically, queries must cope with atelic relations, telic relations, or a combination of both.

Furthermore, linguistic research suggests a further requirement for telic/atelic query languages: *flexibility*. It is widely accepted within the linguistic community that while basic facts can be classified as telic or atelic, natural languages provides several ways to switch between the two classes. For instance, given a telic fact (such as "John built a house"), the progressive form (e.g., "John *was* build*ing* a house") coerces it into an atelic one, stripping away the culmination (and, in fact, "John was building a house" does not imply that "John built a house," i.e., that he finished it) [6]. For the sake of expressiveness, it is desirable that a database query language provides the same flexibility.

Queries About Atelic Data

As already mentioned above, most database approaches are interpreted (implicitly or explicitly) on the basis of the point-based semantics. Therefore, the corresponding algebraic operators already cope with atelic data. As an example, in BCDM, the union of two relations is simply obtained by taking the tuples of both relations, and "merging" the valid time of value equivalent tuples performing the union of the time points in their valid time. This definition is perfectly consistent with the "snapshot-by-snapshot" view enforced by the underlying point-based (atelic) semantics.

However, the algebrae in the literature also contain operators which contrast with such a "snapshot-by-snapshot" underlying semantics. Typical examples are temporal selection operators. For instance, whenever a duration is asked for (e.g., "retrieve all persons married for at least n consecutive time units"), the query implicitly relies on a telic view of data, in which snapshots are not taken into account independently of each others.

Queries About Telic Data

Algebraic operators on telic data can be easily defined by paralleling the atelic definitions, and considering that, in the telic case, the basic temporal primitives are not time points, but time periods [9]. For instance, telic union is similar to atelic one, except that the merging of valid times of value-equivalent tuples is performed by making the union of multisets of time periods, considered as primitive entities, e.g.,

$$\{[10-12],[13-15]\} \cup \{[10-14],[13-18]\}$$
$$=\{[10-12],[13-15],[10-14],[13-18]\}$$

Note that temporal selection operators perfectly fit with the telic environment. On the other hand, algebraic operators that intuitively involve a snapshot-by-snapshot view of data (e.g., Cartesian product, involving a snapshot-by-snapshot intersection between valid times) have an awkward interpretation in the telic context; for this reason, difference and "standard" Cartesian product have not been defined in the telic algebra in [9].

Queries Combining Telic and Atelic Data

In general, if a two-sorted data model is used, queries combining relations of both kinds are needed. In general, such queries involve the (explicit or implicit) coercion of some of the relations, to make the sort of the relations consistent with the types of the operators being used. For instance, the following query utilizes the example atelic relation modeling marriages and the telic one considering phone calls: Who was being married when John was calling Mary?

In such a case, the English clause "when" demands for an atelic interpretation: the result can be obtained by first coercing the relation about phone calls into an atelic relation, and then by getting the temporal intersection through the application of the atelic Cartesian product.

On the other hand, the query "list the marriage ceremonies that had a duration of more than 3 hours" requires a coercion of marriages into a telic relation, so that the whole valid time is considered as (a set of) time periods (instead as a set of independent points, as in the atelic interpretation), and the telic temporal selection operator can be applied.

In general, two coercion operators need to be provided [9]. Coercion from telic to atelic is easy: each time period constituting the (semantics of the) valid time is converted into the set of time points it contains, e.g., *to-atelic*({[10–12],[13–15]}) = {10,11,12,13,14,15}. Of course, since multisets of time periods are more expressive than sets of time points, such a conversion causes a loss of information. On the other hand, coercion from atelic to telic demands the formation of time periods out of sets of points: the output is the set of maximal convex time periods exactly covering the input set of time points, e.g., to-telic({10,11,12,13,14, 15}) = {[10–15]}.

Key Applications

Most applications involve differentiating between telic and atelic data.

Cross-references

- ▶ Atelic Data
- ▶ Period-Stamped Temporal Models
- ▶ Point-Stamped Temporal Models
- ▶ Temporal Query Languages

Recommended Reading

1. Allen J.F. Towards a general theory of action and time. Artif. Intell., 23:123–154, 1984.
2. Aristotle. The Categories, on Interpretation. Prior Analytics. Harvard University Press, Cambridge, MA, 1938.
3. Bennet M. and Partee B. Tense and Discourse Location in Situation Semantics. Indiana University Linguistics Club, Bloomington, 1978.
4. Dowty D. The effects of the aspectual class on the temporal structure of discourse, tense and aspect in discourse. Linguist. Philos., 9(1):37–61, 1986.
5. Jensen C.S. and Snodgrass R.T. Semantics of time-varying Iinformation. Inf. Syst., 21(4):311–352, 1996.
6. Moens M. and Steedman M. Temporal ontology and temporal reference. Comput. Linguist., 14(2):15–28, 1998.
7. Shoham Y. Temporal logics in AI: semantical and ontological considerations. Artif. Intell., 33:89–104, 1987.
8. Snodgrass R.T. (ed.). The Temporal Query Language TSQL2. Kluwer, Norwell, MA, 1995.
9. Terenziani P. and Snodgrass R.T. Reconciling point-based and interval-based semantics in temporal relational databases: a proper treatment of the Telic/Atelic distinction. IEEE Trans. Knowl. Data Eng., 16(4):540–551, 2004.
10. Terenziani P., Snodgrass R.T., Bottrighi A., Torchio M., and Molino G. Extending temporal databases to deal with Telic/Atelic medical data. Artif. Intell. Med., 39(2):113–126, 2007.
11. Vendler Z. Verbs and times. In Linguistics in Philosophy. Cornell University Press, New York, NY, 1967, pp. 97–121.

Telos

Manolis Koubarakis
University of Athens, Athens, Greece

Definition

Telos (From the Greek word τέλοςτΘ which means *end*; the object aimed at in an effort; purpose.) is a knowledge representation language designed especially to support the development of information systems. Telos is based on the premise that information system development is knowledge-intensive and that the main design goal of any language intended for the task should be to formally represent the relevant knowledge. Telos is founded on core concepts from data modeling and knowledge representation, and shares ideas with semantic networks and frame systems, semantic and object-oriented data models, logic programming and deductive databases. The main features of Telos include: a structurally object-oriented framework which supports aggregation, generalization and classification; a novel treatment of attributes as first class citizens in the language; a powerful way of defining meta-classes; an explicit representation of time; and facilities for specifying integrity constraints and deductive rules.

Historical Background

The research on Telos follows the paradigm of a number of software engineering projects initiated around the 1990s with the premise that software development is *knowledge-intensive* and that the primary responsibility of any language intended to support this task is to be able to *formally represent the relevant knowledge*. Thus, Telos was designed as a knowledge representation language that is intended to support software engineers in the development of information systems throughout the software lifecycle.

Telos has evolved from RML (a requirements modeling language developed in Sol Greenspan's Ph.D. thesis), and later CML (presented in the Master thesis of Martin Stanley at the University of Toronto). The main difference between RML and CML is that CML adopts a more sophisticated model for representing knowledge, and supports the representation of temporal knowledge and the definition of meta-classes. Telos is essentially a "cleaned-up" and improved version of CML which was originally defined and implemented in the Master thesis of Manolis Koubarakis at the University of Toronto. The original paper on Telos is [7]. Ontological and semantical issues for Telos are discussed in [10]. The history of knowledge representation languages for information systems development related to Telos is surveyed in [3]. An important dialect of Telos is O-Telos defined in the Ph.D. thesis of Manfred Jeusfeld at the University of Passau, and implemented in the ConceptBase system [4]. Since ConceptBase is the most mature implementation of Telos available today, this entry uses the ConceptBase syntax for Telos.

Foundations

The main (and essentially the only) concept of Telos is the *proposition*. Propositions are used to model any aspect of the application domain. Propositions have unique identities and are distinguished into *individuals* and *attributes*. Individuals are intended to represent entities in the application domain (concrete ones such as John Doe, or abstract ones such as the class of all persons). Attributes represent binary relationships between entities (concrete or abstract). Two special kinds of attribute propositions exist: instantiation propositions and specialization propositions. The proposition abstraction gives great flexibility to Telos users. Everything in the application domain that is represented by a proposition (e.g., an entity or a relationship) immediately becomes a first-class citizen of the knowledge base.

Propositions

Every proposition `p` consists of an *identifier*, a *source*, a *label* and a *destination*, denoted by the functions `id(p)`, `from(p)`, `label(p)` and `to(p)`. For example, the following are propositions:

```
P1: [..., Martin, ...]
P2: [..., "21 Elm Avenue," ...]
P3: [Martin, homeAddress, "21 Elm
    Avenue"]
P4: [..., Person, ...]
P5: [..., GeographicLocation, ...]
P6: [Person, address,
    GeographicLocation]
```

Propositions in Telos are what *objects* are in object-oriented formalisms but also what *statements* are in logic-based formalisms. Thus, an application can use the above propositions to represent the following pieces of knowledge:

- `P1`: There is somebody called Martin.
- `P2`: There is something called "21 Elm Avenue."
- `P3`: Martin lives in 21, Elm Avenue.
- `P4`: There is an abstract concept, the class of all persons.
- `P5`: There is an abstract concept, the class of all geographic locations.
- `P6`: Persons have addresses that are geographic locations.

`P1`, `P2`, `P4` and `P5` are individual propositions while `P3` and `P6` are attribute propositions. The source and destination components of an individual proposition are not important, thus they are shown as "...". Notice that while P1 and P2 represent concrete individuals, P4 represents an abstract one, the class of all persons. Similarly, P3 represents a concrete relationship (relating Martin with his address) while P6 represents an abstract one (relating the class of all persons with the class of all geographic locations).

Following are some examples of special propositions:

```
P7: [P1, *instanceOf, P4]
P8: [P3, *instanceOf, P6]
P9: [..., Employee, ...]
P10:[P9, *isA, P4]
```

P7 and P8 are *instantiation propositions*. P7 represents the fact that Martin is a member of the class of all persons. P8 represents the fact that the concrete relationship relating Martin with his address is an instance of the abstract relationship relating the class of all persons with the class of all geographic locations. Finally, P10 is a *specialization proposition* asserting that every employee is a person.

A graphical view of some of the above propositions is given in Fig. 1.

Organizing Propositions

Propositions (individual or attribute ones) can be organized along three dimensions: *decomposition/aggregation*, *instantiation/classification* and *specialization/generalization*.

The aggregation dimension enables one to see an entity of the application domain as a collection of propositions with a common proposition as source. For example, individual Martin can be seen to be the following aggregation:

```
{Martin,
 [Martin, age, 35],
 [Martin, homeAddress, "21 Elm
 Avenue"],
 [Martin, workAddress, "10 King's
 College Road"] }
```

The classification dimension calls for each proposition to be an instance of one or more generic propositions or classes. Classes are themselves propositions, and therefore instances of other, more abstract classes. For example, Person is a class and Martin is an instance of this class. Similarly,

```
[Person, address, GeographicLocation]
```

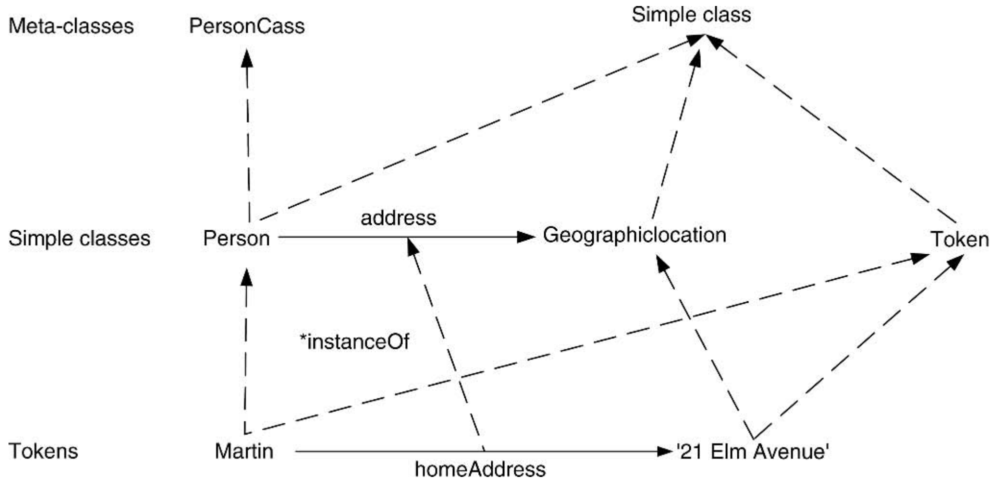
is a class and

```
[Martin, homeAddress, "21 Elm Avenue"]
```

is an instance of this class.

With respect to the classification dimension, propositions can be distinguished into:

- *Tokens:* propositions having no instances and intended to represent concrete entities in the application domain.
- *Simple classes:* propositions having only tokens as instances.



Telos. Figure 1. A graphical view of a set of Telos propositions.

- *Meta-classes:* propositions having only simple classes as instances.
- *Meta-meta-classes:* propositions having only meta-classes as instances.
- ...

Thus, classification in Telos defines an *unbounded* linear order of planes of ever more abstract propositions. Implementations restrict this unbounded hierarchy (e.g., ConceptBase restricts it to four levels: tokens to meta-meta-classes). There are also *ω-classes* with instances along more than one plane:

- `Proposition`. Contains all propositions as instances.
- `Class`. Contains all classes as instances.
- `Token`. Contains those individuals that may never have instances themselves.
- `SimpleClass`. Contains individuals that may have instances which are tokens.
- `MetaClass`. Contains individuals that may have simple classes as instances.
- `MetametaClass`. Contains individuals that may have meta-classes as instances.
- ...

Classification in Telos is a form of *weak typing*: the classes of which a structured object is an instance determine the kinds of attributes it can have optionally, and the properties it must satisfy. For example, by virtue of being an instance of `Person`, `Martin` can have attributes that are instances of the attribute class

```
[Person,address,GeographicLocation].
```

These *zero or more* attributes can have arbitrary labels, e.g., `homeAddress` and `workAddress`, but their values must be instances of `GeographicLocation`.

Finally, classes in Telos can be specialized along generalization or ISA hierarchies. For example, `Person` may have subclasses such as `Professor`, `Student`, and `TeachingAssistant`. Classes may form a partial order, rather than a tree (i.e., multiple inheritance is supported). Non-token attributes of a class are inherited by more specialized ones and can be refined. Inheritance in Telos is strict rather than default.

Interacting with Telos Knowledge Bases

A few examples of Telos are now given. The example application considered is the development of an information system to support organizing international scientific conferences. The knowledge to be represented in this case is about entities such as papers, authors, conferences etc.

The original definition of Telos in [7] defines the operations `TELL`, `UNTELL`, `RETELL` and `ASK` for interacting with a Telos knowledge base. These operations can be used to add new knowledge, discard existing knowledge, update existing knowledge and query a knowledge base, respectively. Implementations such as ConceptBase have followed the original definition and offer these (and other) operations. The above operations have Telos statements such as the following as their means of interaction with a knowledge base:

```
Individual p133 in Token, Paper with
      author
         firstAuthor: Stanley;
         secondAuthor: LaSalle;
         thirdAuthor: Wong
    title
         called: "The language Telos"
end
```

The above statement introduces an individual with name `p133`. The `in` clause specifies the classes of which `p133` is an instance (in this case, the predefined class `Token` and the application class `Paper`). The `with` clause introduces `p133`'s attributes. The first attribute of `p133` has label `firstAuthor` and is an instance of an attribute class which has source `Paper` and label `author` (the latter is denoted by the *attribute category* `author`). Before introducing individual paper `p133`, one might have defined the class of all papers using the following statement:

```
Individual Paper in SimpleClass with
      attribute
         author: Person;
         referee: Person;
         title: String;
         pages: Integer
end
```

A class definition prescribes the attributes that can be associated with its instances: `p133` can have `author`, `referee`, `title` and `page` attributes as seen previously, because it is an instance of class `Paper` that has these *attribute classes*. Moreover,

```
      [p133,firstAuthor,Stanley]
```

is an instance of attribute class

```
      [Paper,author,Person]
```

in exactly the same sense that p133 is an instance of Paper.

Once Paper has been defined, one can introduce specializations such as InvitedPaper using the isA clause of class definitions:

```
Individual AcceptedPaper in
SimpleClass isA Paper with
    attribute
      session: ConfProgrammeSession
end
```

AcceptedPaper inherits all attributes from Paper and adds a session attribute, to indicate the programme session during which the accepted paper will be presented.

Metaclasses

Metaclasses are a very powerful concept for modeling power and extensibility in Telos. It is the metaclass mechanism combined with its other features that makes Telos a powerful modeling language (one might wonder about this, since Telos offers only very simple primitives). From a modeling point of view, one can use Telos metaclasses in the following situations:

- To define concrete attributes of classes e.g., cardinality of a class. This is exactly the same to what a simple class does for its instances (tokens).
- To group together semantically similar classes of a domain in a generic way. For example, in the conference organization example, the classes Paper, Announcement, Letter, Memo could be grouped under the metaclass DocumentClass.
- To define concepts that are built-in in other frameworks e.g., necessary attributes, single-valued attributes etc.
- To do other forms of meta-level logical reasoning (again, for language expressibility).

The conference organization example is now revisited and defined:

```
DocumentClass in MetaClass with
    attribute
        source: AgentClass;
        content: SimpleClass;
        destination: AgentClass;
        cardinality: Integer
end
```

The class Paper can now be defined as follows:

```
Paper in DocumentClass with
      source
            author: Person;
      content
            title: String;
            abstract: String
      cardinality
            how_many: 120
end
```

Note that attribute categories such as source introduced in metaclass DocumentClass are then used to define attributes for the instance class Paper (this mechanism is the same along the instantiation hierarchy).

Integrity Constraints and Deductive Rules

Telos borrows the notions of integrity constraints and deductive rules from logic-based formalisms such as deductive databases. *Integrity constraints* are formulas that express conditions that knowledge bases should satisfy. They are used to express rich language or application semantics that cannot possibly be expressed only by the structural framework of Telos. *Deductive rules* are formulas that can be used to derive new knowledge. Integrity constraints and deductive rules in Telos are expressed in appropriately defined assertional languages that are subsets of first-order logic [4,7].

Integrity constraints and rules are defined as attributes of Telos classes that are instances of the built-in object Class. For example, the following Telos statement defines well-understood constraints and rules regarding employees, their managers and their respective salaries.

```
Class Employee with
  rule
    BossRule: $ forall e/Employee
    m/Manager (exists d/Department
    (e dept d) and (d head m))
    ==> (e boss m) $
  constraint
    SalaryBound: $ forall e/Employee
    b/Manager x,y/Integer (e boss b)
```

```
      and (e salary x) and (b salary y)
      ==> x <= y $
  end
```

Language Extensibility Through Metaclasses and Integrity Constraints

In Telos, one can use integrity constraints together with the metaclass mechanism to define concepts that are built-in in other representational frameworks. For example, in many object-oriented models one can constrain an attribute to be single-valued using some built-in construct of the model. In Telos, one can do this by using only the primitive mechanisms of the language as follows. First, one defines the class `Single`: (The syntax of this statement is from the original paper of Telos [7] (O-Telos allows one to specify the same thing in a slightly more complex way).)

```
  Class Single
    components [Class, single, Class]
    in AttributeClass, MetaClass with
    integrityConstraint
      : $ forall u/Single
           p,q/Proposition(p in u) and
           (q in u) and from(p)=from(q)
           ==> p=q $
  end
```

Then, one uses attribute class `Single` in the definition of class `Paper`:

```
  Individual Paper in SimpleClass with
      attribute
        author: Person;
        referee: Person
    single
        title: String;
        pages: Integer
  end
```

Now in every instance of `Paper`, a `title` attribute is constrained to be single-valued due to the integrity constrain and the instantiation relationships introduced by the above Telos statements.

Query Languages for Telos

The papers [4,7,11] give various query languages for Telos knowledge bases ranging from ones based purely on first-order logic [7] to ones exploiting the structurally object-oriented features of the language as well [4,11].

The paper [8] presents work on a knowledge base management system based on Telos.

Temporal Knowledge in Telos

The original paper of Telos [7] presents a powerful framework for representing and reasoning about temporal knowledge. In Telos, the history of an application domain is modeled by augmenting propositions with a *history time* i.e., an interval representing the time during which these facts are true in the application domain. Historical knowledge in Telos is allowed to be incomplete and a modification of Allen's interval algebra [1] is used to capture the relevant knowledge.

A knowledge base records essentially the *beliefs* of the system, which may be distinct from the actual state of the world at that time. So, for example, the title of a paper might have been changed in March, but the knowledge base is only told of it in May. Or one may make a correction to some previously told fact. Just like it represents the full history of an application domain, Telos also records the full history of its beliefs. For this reason, Telos represents *belief times*; these are intervals associated with every proposition in the knowledge base, which commence at the time when the operation responsible for the creation of the corresponding proposition was committed.

For efficiency reasons, implementations of Telos such as ConceptBase [4] have restricted the kinds of temporal knowledge that can be represented.

Telos and RDF

Telos is probably the pre-Web knowledge representation language most closely related to the Resource Description Framework (RDF) and the RDF Vocabulary Description Language or RDF Schema proposed by the W3C for representing knowledge about Web resources (see e.g., http://www.w3.org/TR/rdf-primer/). This relationship has been exploited by the prominent RDF query language RQL defined by ICS-FORTH [6] but also in the O-Telos-RDF proposal [9].

Key Applications

Telos was designed as a knowledge representation language that is intended to support software engineers in the development of information systems throughout the software lifecycle [7]. The strengths of the language made it the choice of many prominent research projects in Europe and North America including DAIDA [5], ITHACA [2] and others.

URL to Code

The most mature implementation of Telos is the ConceptBase system available at http://conceptbase.cc/.

Cross-references

- ▶ Meta Model
- ▶ Object Data Models
- ▶ RDF Schema
- ▶ Semantic Data Models

Recommended Reading

1. Allen J. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832–843, 1983.
2. Constantopoulos P., Jarke M., Mylopoulos J., and Vassiliou Y. The software information base: A server for reuse. VLDB J., 4(1):1–43, 1995.
3. Greenspan S.J., Mylopoulos J., and Borgida A. On formal requirements modeling languages: RML revisited. In Proc. 16th Int. Conf. on Software Eng., 1994, pp. 135–147.
4. Jarke M., Gallersdörfer R., Jeusfeld M.A., and Staudt M. ConceptBase – A deductive object base for meta data management. J. Intell. Inf. Syst., 4(2):167–192, 1995.
5. Jarke M., Mylopoulos J., Schmidt J.W., and Vassiliou Y. DAIDA: An environment for evolving information systems. ACM Trans. Inf. Syst., 10(1):1–50, 1992.
6. Karvounarakis G., Alexaki S., Christophides V., Plexousakis D., and Scholl M. RQL: A declarative query language for RDF. In Proc. 11th Int. World Wide Web Conference, 2002.
7. Mylopoulos J., Borgida A., Jarke M., and Koubarakis M. Telos: A language for representing knowledge about information systems. ACM Trans. Inf. Syst., 8(4):325–362, 1990.
8. Mylopoulos J., Chaudhri V.K., Plexousakis D., Shrufi A., and Topaloglou T. Building knowledge base management systems. VLDB J., 5(4):238–263, 1996.
9. Nejdl W., Dhraief H., and Wolpers M. O-Telos-RDF: A resource description format with enhanced meta-modeling functionalities based on O-Telos. In Proc. Workshop on Knowledge Markup and Semantic Annotation at the 1st Int. Conf. on Knowledge Capture, 2001.
10. Plexousakis D. Semantical and ontological consideration in Telos: A language for knowledge representation. Comput. Intell., 9:41–72, 1993.
11. Staudt M., Nissen H.W., and Jeusfeld M.A. Query by class, rule and concept. Appl. Intell., 4(2):133–156, 1994.

Temporal Access Control

Yue Zhang, James B. D. Joshi
University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

Time-based access control

Definition

Temporal access control refers to access control service that restricts granting of authorization based on time. The authorization may be given to a subject for a particular interval or duration of time or based on the temporal characteristics of the objects being accessed. Such a need arises from the fact that a subject's need to access a resource and the sensitivity (and hence the protection requirement) of the objects being accessed may change with time.

Historical Background

Work related to temporal access control has only a brief history and goes back to early 1990s. In many real-world situations, access to information and resources may have to be restricted based on time as the subject and object characteristics may change and so can the need for the subject to access the object. For example, in a hospital, the head of the hospital may need to grant the permissions related to a part-time doctor only during certain time intervals. Similarly, an external auditor may need to be given access to sensitive company data for a specific duration only.

Bertino et al.'s *Temporal Authorization Model* (TAM) is the first known access control model to support the time-based access control requirements in a discretionary access control (DAC) model [2]. TAM associates a periodicity constraint with each authorization indicating the valid time instants of the authorization. TAM also defines three derivation rules that allow an authorization to be derived based on another authorization. The TAM model, however, is limited to specifying the temporal interval for an entire authorization and does not consider the temporal characteristics of data/objects [1]. For example, there may be a need for be an authorization such as "a subject *s* is allowed to read object *o* one month after it has been created/written." Furthermore, the states of an object can change with time and access to such different object-states may need to be carefully specified. Atluri et al. propose a Temporal and Derived Data Authorization Model (TDAM) for a Web information portal [1]. An information portal mainly aims to provide access to data from different sources and hence the temporal characteristics of such data need to be properly captured in an access control policy [1]. TDAM uses a logic formula to capture time-related conditions to specify authorization rules and address the consistency issues.

Another significant work related to temporal access control can be seen within the context of the Role Based Access Control (RBAC) model. Bertino et al. proposed the Temporal Role Based Access Control Model (TRBAC) model by extending the existing RBAC model [3]. The TRBAC model supports the periodicity/interval constraints on the role enabling/disabling and uses triggers to define dependencies that may exist among basic role related events such as "*enable role*" and "*disable role*."

One limitation of the TRBAC model is that it only supports specification of a set of temporal intervals on the role enabling/disabling events. For example, the user-role and the role-permission assignments are not time-constrained in TRBAC, neither are the role activations. Joshi et al. proposed a General Temporal RBAC (GTRBAC) model to allow specification of more fine-grained temporal access control policies [5]. The GTRBAC model allows the interval and duration constraints on user-role assignment, role-permission assignment, and role enabling events. It also defines the duration and cardinality constraints on role activation. GTRBAC uses constraint enabling events to trigger the duration constraints and uses run-time requests to allow dynamic changes in the authorization states by administrators and users. GTRBAC uses triggers, first introduced in the TRBAC model, to support the dependencies among events. The other features of the GTRBAC include the temporal hybrid hierarchy and Separation of Duty (SoD) constraints.

More recently, work on access control approaches based on the location context and the integration of temporal and location based access control has emerged. Such work includes the GEO-RBAC model, the spatial-temporal access control model [4], the spatial-temporal RBAC model [7], the LRBAC model [6], and the location and time-based RBAC (LoT-RBAC) model.

Foundations

In this section, the existing time-based access control models mentioned above are reviewed in more detail, namely TAM, TDAM, the TRBAC model, and the GTRBAC model. The major features of each model are discussed below.

TAM: Temporal Authorization Model

TAM models the temporal context as a periodicity constraint. A periodicity constraint is of the form <[*begin*, *end*], *P*>; here, *P* represents a recurring set of intervals and the entire expression indicates every time instant in *P* between "begin" and "end." For example, "[1/1/1994, ∞], *Monday*" indicates every *Monday* starting 1/1/1994. The model uses the symbol ∞ in place of *P* to indicate all time instants. The temporal authorization in TAM associates such a periodicity constraint with a normal discretionary authorization. The authorization is valid at any time instant specified in the periodicity constraint. At any given time instant, if there are two authorization rules that try to grant and deny an operation on the same object, a conflict is said to occur. In such a case, the model uses the "*denials-take-precedence*" principle to favor negative authorizations.

It is possible that several authorizations have temporal dependencies among them. For example, suppose user u_1 grants a permission p to u_2 (authorization A_1), and u_2 further grants p to u_3 (authorization A_2). It is easy to see that A_2 can only be valid after A_1 becomes valid. Therefore, a requirement such as "u_2 is allowed to grant p to u_3 whenever he/she acquires p from u_1," can be specified as a derivation rule "A_2 WHENEVER A_1." TAM uses three such derivation rules to capture various relationships among different authorizations.

The derivation rules are of the form "[*begin*, *end*], *P*, *A* <*op*>, **A**", where, *A* is an authorization, **A** is a boolean expression on authorizations, and <*op*> is one of the following: WHENEVER, ASLONGAS, UPON. **A** is true at time instant *t*, if **A** evaluates to *true* by substituting each authorization in it by *true* if it is valid at *t*, and by *false* if not valid. For example, $\mathbf{A} = \neg\,(A_1 \text{ and } A_2)$ is *true* at time *t* if either or both of A_1 and A_2 is *false* at time *t*. ([*begin, end*], *P*, *A* WHENEVER **A**) specifies that one can derive *A* for each instant in $\Pi(P) \cap \{[t_b, t_e]\}$ (here, $\Pi(P)$ is the set of all time instants in *P*) for which **A** is valid. ([*begin*, *end*], *P*, *A* ASLONGAS **A**) specifies that one can derive *A* for each instant in $\Pi(P) \cap \{[t_b, t_e]\}$ such that **A** is valid for each time instant in $\Pi(P)$ that is greater than or equal to t_b and lesser than or equal to t_e. ([*begin*, *end*], *P*, *A* UPON **A**) specifies that *A* holds *true* for each instants in $\Pi(P) \cap \{[t_b, t_e]\}$ if there exists an instant $t' \varepsilon \Pi(P)$ that is greater than or equal to t_b and lesser than or equal to t_e such that **A** is valid at time t' in. The difference between WHENEVER and ASLONGAS is that the former only evaluates the authorization state for a time instant while the latter evaluates the history of the authorization states in a given time interval. The difference between ASLONGAS and UPON is that the former evaluates the authorization state for the entire time interval in the history while the latter only evaluates the authorization states at a time instant in the history.

Given a set of initial authorizations and rules, the derived authorizations may depend on the order in which the rules are evaluated. This is due to the existence of both positive and negative authorizations and the *denials-take-precedence* rule. To analyze this problem, the authors define the unique set of valid authorizations using the notion of critical set and propose a Critical Set Detection (CSD) algorithm to verify that a given set is critical [2].

TDAM: Temporal and Derived data Authorization Model

The TDAM model, as mentioned earlier, focuses on the temporal characteristics of data/objects. This is achieved by associating a formula τ instead of a simple temporal interval to each authorization. At any time instant, if τ is evaluated to be *true*, then the corresponding authorization is valid. By carefully designing the formula τ, the model can support specification of a fine-grained temporal access control policy. It is possible to use the temporal context of a single data item by making it a variable in τ. For example, assume a company maintains a database including the current and future predicted price for some goods; now, let each price of a data item d be associated with a time interval $[t_b, t_e]$ to indicate the temporal interval the specified price is predicted for. It is also possible to restrict a subject to read the current price only by specifying τ as "$t_b \leq t \leq t_e$" where t is the current time.

Similarly, the temporal dependency can also be indicated by τ. For example, consider the policy "a subject s is allowed to read object o one month after it has been created/written." Such a requirement can be specified by including τ as "$t \geq t_w + 1$ month" where t_w indicates the time when the object o is created/written and t is the current time.

TRBAC: Temporal Role Based Access Control Model

The TRBAC model allows the specification of temporal constraints to specify when a role can be enabled or disabled, and triggers to capture possible temporal dependencies among role events. The TRBAC model also uses periodic time expression instead of simple time interval to represent the periodicity constraint. In RBAC, the users can acquire the permissions only by activating enabled roles within a session; the model associates the periodic time expressions with role enabling/disabling events to define periodicity constraints.

Triggers constitute an important part of the TRBAC model, and they are used to capture temporal dependencies among RBAC authorization states. A trigger simply specifies that if some events occur and/or some role status predicates are evaluated to true then another event can occur, with a possible time delay. For example, assume that a hospital requires that when a part-time doctor is on duty a part-time nurse can also be allowed to log on to help the part-time doctor. Here, the trigger "enable *part-time doctor* → enable *part-time nurse*" can be used to capture such a requirement.

Bertino et al. introduces a soft notion of *safety* to formally characterize the efficiency and practicality of the model. In essence, because of triggers and other periodicity constraints, ambiguous semantics can be generated. They propose an efficient graph based analysis technique to identify such ambiguous policies so that a unique execution model can be guaranteed by eliminating any ambiguous specification. When such a unique execution model is ensured the policy base is said to be *safe.* Conflicts could also occur because of the opposing *enable* and *disable* role events. The TRBAC model uses *denial-takes-precedence* in conjunction with *priority* to resolve such conflicts.

GTRBAC: Generalized Temporal Role Based Access Control Model

Joshi et al. have proposed the GTRBAC model by extending the TRBAC model to incorporate more comprehensive set of periodicity and duration constraints and time-based activation constraints. The GTRBAC model introduces the separate notion of *role enabling* and *role activation*, and introduces role states. An *enabled* role indicates that a valid user can activate it, whereas a *disabled* role indicates that the role is not available for use. A role in *active* state indicates that at least one user has activated it. Such a distinction makes the semantics and implementation of any RBAC compatible system much clearer. Besides events related to user-role assignment, permission-role assignment and role enabling, the GTRBAC model also includes role activation/deactivation events. The model associates the temporal constraints with every possible event in the system. In particular, it uses the periodicity constraint to constrain the validity of role enabling events, user-role assignment events as well as the permission-role assignment events. As the role activation events are initiated by the users at their discretion, GTRBAC does not associate temporal constraints with activation events. Besides the periodic temporal constraints, GTRBAC also supports duration constraints. A duration constraint specifies how long should an event be valid for once it occurs. For example, one duration constraint could specify that once a role *r* has been enabled it should be in the enabled state for 2 hours. Note that the duration constraint has a non-deterministic start time and requires some other actions to initiate the start of the duration. For example, consider a duration constraint (2 hours, enable *r*). Here, when the "enable *r*" event occurs because of a trigger, the event becomes valid for 2 hours because of this constraint. At times, one may need to also enable duration or activation constraints – GTRBAC uses constraint enabling events to facilitate that.

The GTRBAC model also supports the temporal and cardinality constraints on role activation. Cardinality constraints have often been mentioned in the literature but have not been addressed much in the existing models. A cardinality constraint simply limits the number of activations (applies for assignments as well) within a given period of time. For example, GTRBAC supports limiting the total number of activations of a role or the maximum concurrent number of activations of a role in a given interval or duration. Furthermore, GTRBAC allows the activation constraint to be applied to all the activations of a role (per-role constraint) or applied to each activation of a role by a particular user (per-user constraint). The model uses the trigger framework introduced in the TRBAC model. In addition to these, the GTRBAC model also extends work on role hierarchy and constraints, and introduces hybrid hierarchy and time-based SoD constraints.

The GTRBAC model uses three conflict types (Type-1, Type-2, and Type-3) to categorize the different types of conflicting situations that may arise and provides a resolution technique using a combination of the following approaches: (i) *priority-based*, (ii) *denial-takes precedence*, and (iii) *more specific constraint takes precedence.*

Suroop et al. have recently proposed the LoTRBAC model by extending the GTRBAC model to address both time and location for security of mobile applications.

Key Applications

Temporal access control models are suitable for the applications where temporal constraints and temporal dependencies among authorizations are important protection requirements. One example is workflow systems that often have timing constraints on tasks and their dependencies. In particular, TAM is suitable for the system implementing the discretionary access control policies. TDAM is suitable for access control for data dissemination systems such as a web information portal where data have different states at different times. TRBAC and GTRBAC are suitable for large scale systems that have very fine-grained time-based access requirements.

Future Directions

With the development of networking and mobile technologies, context based access control is becoming very crucial. Time is very crucial context information, as is location. Hence, the work on temporal access control have provided a basis for looking at the intricacies of the context parameters that need to be used to restrict authorization decisions. In addition to context-based access, content-based access control is also becoming significant issues because of the growing need to dynamically identify content and make authorization decisions based on its semantics. Again, the work on temporal access control has provided a basis for capture the dynamic feature of the object content. Authorization models that capture context and content parameters using temporal access control are being pursued to develop more fine-grained access control models.

Cross-references

► Role Based Access Control

Recommended Reading

1. Atluri V. and Gal A. An authorization model for temporal and derived data: securing information portals. ACM Trans. Inf. Syst. Secur., 5(1):62–94, 2002.
2. Bertino E., Bettini C., Ferrari E., and Samarati P. An access control model supporting periodicity constraints and temporal reasoning. ACM Trans. Database Syst., 23(3): 231–285, 1998.
3. Bertino E., Bonatti P.A., and Ferrari E. TRBAC: a temporal role-based access control model. ACM Trans. Inf. Syst. Secur., 4(3):191–233, 2001.
4. Fu S. and Xu C.-Z. A coordinated spatio-temporal access control model for mobile computing in coalition environments. In Proc.

19th IEEE Int. Parallel and Distributed Processing Sym. – Workshop 17, vol. 18, 2005, p.289.2.

5. Joshi J.B.D., Bertino E., Latif U., and Ghafoor A. A generalized temporal role-based access control model. IEEE Trans. Knowl. Data Eng., 17(1):4–23, 2005.
6. Ray I., Kumar M., and Yu L. LRBAC: a location-aware role-based access control model. In Proc. 2nd Int. Conf. on Information Systems Security, 2006, pp. 147–161.
7. Ray I. and Toahchoodee M. A spatio-temporal role-based access control model. In Proc. 21st Annual IFIP WG 11.3 Working Conf. on Data and Applications Security, 2007, pp. 420–431.

Temporal Aggregation

Johann Gamper[1], Michael Böhlen[1], Christian S. Jensen[2]

[1]Free University of Bozen-Bolzano, Bolzano, Italy

[2]Aalborg University, Aalborg, Denmark

Definition

In database management, aggregation denotes the process of consolidating or summarizing a database instance; this is typically done by creating so-called aggregation groups of elements in the argument database instance and then applying an aggregate function to each group, thus obtaining an aggregate value for each group that is then associated with each element in the group. In a relational database context, the instances are relations and the elements are tuples. Aggregation groups are then typically formed by partitioning the tuples based on the values of one or more attributes so that tuples with identical values for these attributes are assigned to the same group. An aggregate function, e.g., *sum*, *avg*, or *min*, is then applied to another attribute to obtain a single value for each group that is assigned to each tuple in the group as a value of a new attribute. Relational projection is used for eliminating detail from aggregation results.

In temporal relational aggregation, the arguments are temporal relations, and the tuples can also be grouped according to their timestamp values. In temporal grouping, groups of values from the time domain are formed. Then an argument tuple is assigned to each group that overlaps with the tuple's timestamp, this way obtaining groups of tuples. When aggregate functions are applied to the groups of tuples, a temporal relation results. Different kinds of temporal groupings are possible: instantaneous temporal aggregation where the time line is partitioned into time instants/points; moving-window (or cumulative) temporal aggregation where additionally a time period is placed around a time instant to determine the aggregation groups; and span aggregation where the time line is partitioned into user-defined time periods.

Historical Background

Aggregate functions assist with the summarization of large volumes of data, and they were introduced in early relational database management systems such as System R and INGRES. During the intensive research activities in temporal databases in the 1980s, aggregates were incorporated in temporal query languages, e.g., the Time Relational model [1], TSQL [8], TQuel [9], and a proposal by Tansel [10]. The earliest proposal aimed at the efficient processing of (instantaneous) temporal aggregates is due to Tuma [12]. Following Tuma's pioneering work, research concentrated on the development of efficient main-memory algorithms for the evaluation of instantaneous temporal aggregates as the most important form of temporal aggregation [6,7].

With the diffusion of data warehouses and OLAP, disk-based index structures for the incremental computation and maintenance of temporal aggregates were investigated by Yang and Widom [14] and extended by Zhang et al. [15] to include non-temporal range predicates. The high memory requirements of the latter approach were addressed by Tao et al. [11], and approximate solutions for temporal aggregation were proposed. More recently, Vega Lopez et al. [13] formalized temporal aggregation in a uniform framework that enables the analysis and comparison of the different forms of temporal aggregation based on various mechanisms for defining aggregation groups. In a similar vein, Böhlen et al. [3] develop a new framework that generalizes existing forms of temporal aggregation by decoupling the partitioning of the time line from the specification of the aggregation groups.

It has been observed that expressing queries on temporal databases is often difficult with SQL, in particular for aggregation. As a result, temporal query languages often include support for temporal aggregation. A recent paper [2] studies the support for temporal aggregation in different types of temporal extensions to SQL. A subset of the temporal aggregates considered in the entry are also found in non-relational query languages, e.g., τXQuery [5].

Foundations

A discrete time domain consisting of a totally ordered set of time instants/points is assumed together with an interval-based, valid-time data model, i.e., an interval timestamp is assigned to each tuple that captures the time when the corresponding fact is true in the modeled reality. As a running example, the temporal relation *CheckOut* in Fig. 1 is used, which records rentals of video tapes, e.g., customer C101 rents tape T1234 from time 1 to 3 at cost 4.

Defining Temporal Aggregation

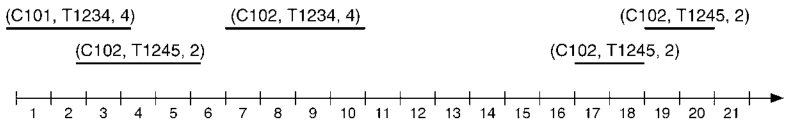
Various forms of temporal aggregation that differ in how the temporal grouping is accomplished have been studied. In *instantaneous temporal aggregation (ITA)* the time line is partitioned into time instants and an aggregation group is associated with each time instant *t* that contains all tuples with a timestamp that intersects with *t*. Then the aggregate functions are evaluated on each group, producing a single aggregate value at each time *t*. Finally, identical aggregate results for consecutive time instants are coalesced into so-called constant intervals that are maximal intervals over which all result values remain constant. In some approaches, the aggregate results in the same constant interval must also have the same lineage, meaning that they are produced from the same set of argument tuples. The following query, Q_1, and its result in Fig. 2a illustrate ITA: *What is the number of tapes that have been checked out?* Without the lineage requirement, the result tuples (1,[17,18]) and (1,[19,20]) would have been coalesced into (1,[17,20]). While, conceptually, the time line is partitioned into time instants, which yields the most detailed result, the result tuples are consolidated so that only one tuple is reported for each constant interval. A main drawback is that the result relation is typically larger than the argument relation and can be up to twice the size of the argument relation.

With *moving-window temporal aggregation (MWTA)* (first introduced in TSQL [8] and later also termed *cumulative temporal aggregation* [9,14]), a time window is used to determine the aggregation groups. For each time instant *t*, an aggregation group is defined as the set of argument tuples that hold in the interval $[t-w,\ t]$, where $w \geq 0$ is called a window offset. In some work [13], a pair of offsets w and w' is used, yielding a window $[t-w,\ t+w']$ for determining the aggregation groups. After computing the aggregate functions for each aggregation group, coalescing is applied similarly to how it is done for ITA to obtain result tuples over maximal time intervals. The following query, Q_2, and its result in Fig. 2b illustrate MWTA: *What is the number of tapes that have been checked out in the last three days?* To answer this query, a window is moved along the time line, computing at each time point an aggregate value over the set of tuples that are valid at some point during the last three days. While both ITA and MWTA partition the time line into time instants, the important difference is in how the aggregation groups for each time instant are defined.

Next, for *span temporal aggregation (STA)*, the time line is first partitioned into predefined intervals that are defined independently of the argument relation. For each such interval, an aggregation group is then given as the set of all argument tuples that overlap the interval. A result tuple is produced for each interval by evaluating an aggregate function over the corresponding aggregation group. The following query, Q_3, and its result in Fig. 2c illustrate STA: *What is the weekly number of tapes that have been checked out?* The time span is here defined as a period of seven days. Unlike in ITA and MWTA, in STA the timestamps of the result tuples are specified by the application and are independent of the argument data. Most approaches consider only regular time spans expressed in terms of granularities, e.g., years, months, and days.

The *multi-dimensional temporal aggregation (MDTA)* [3] extends existing approaches to temporal aggregation, by decoupling the definition of result groups and aggregation groups. A result group specifies the part of a result tuple that is independent of the actual aggregation (corresponds to the group by

CustID	*TapeNum*	*Cost*	*T*
C101	T1234	4	[1,3]
C102	T1245	2	[3,5]
C102	T1234	4	[7,10]
C102	T1245	2	[17,18]
C102	T1245	2	[19,20]



Temporal Aggregation. Figure 1. Tabular representation and graphical representation of temporal relation *CheckOut*.

Cnt	*T*
1	[1,2]
2	[3,3]
1	[4,5]
0	[6,6]
1	[7,10]
0	[11,16]
1	[17,18]
1	[19,20]

a Q_1: ITA

Cnt	*T*
1	[1,2]
2	[3,5]
1	[6,6]
2	[7,7]
1	[8,12]
0	[13,16]
1	[17,18]
2	[19,20]
1	[21,22]

b Q_2: MWTA

Cnt	*T*
3	[1,7]
1	[8,14]
2	[15,21]

c Q_3: STA

CntE	*CntC*	*T*
0	0	[1,7]
2	1	[8,14]
1	0	[15,21]
0	2	[15,21]

d Q_4: MDTA

Temporal Aggregation. Figure 2. Results of different forms of temporal aggregation.

attributes in SQL). Each result group has an associated aggregation group, namely the set of tuples from which the aggregated value is computed. In general, the grouping attributes of the tuples in an aggregation group might differ from the grouping attributes of the result group. For the specification of the result groups, two different semantics are supported: constant-interval semantics that covers ITA and MWTA and fixed-interval semantics that covers STA. The fixed-interval semantics supports the partitioning of the time line into arbitrary, possibly overlapping time intervals. The following query, Q_4, and its result in Fig. 2d illustrate some of the new features of MDTA: *For each week, list the number of expensive and the number of cheap checkouts during the preceding week?* (expensive being defined as a cost equal or greater than 4 and cheap as a cost equal or smaller than 2). The result groups are composed of a single temporal attribute that partitions the time line, the tuples in the associated aggregation groups do not have to overlap the timestamp of the result group, and two aggregates over different aggregation groups are computed for each result group.

Temporal Aggregation Processing Techniques

The efficient computation of temporal aggregation poses new challenges, most importantly the computation of the time intervals of the result tuples that depend on the argument tuples and thus are not known in advance.

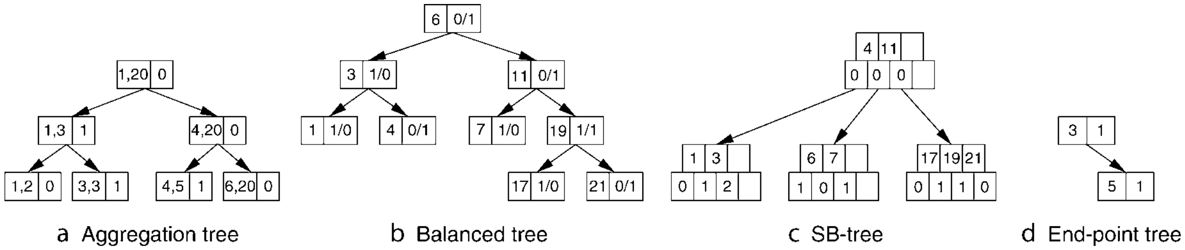
Two Scans The earliest proposal for computing ITA was presented by Tuma [12] and requires two scans of the argument relation – one for computing the constant intervals and one for computing the aggregate values over these intervals. The algorithm has a worst case running time of $O(mn)$ for m result tuples and n argument tuples.

Following Tuma's pioneering work, research concentrated on algorithms that construct main-memory data structures that allow to perform both steps at once, thus requiring only one scan of the argument relation.

Aggregation Tree The *aggregation tree* algorithm for ITA by Kline and Snodgrass [6] incrementally constructs a tree structure in main memory while scanning the argument relation. The tree stores a hierarchy of intervals and partial aggregation results. The intervals at the leaf nodes encode the constant intervals. Accumulating the partial results in a depth-first traversal of the tree yields the result tuples in chronological order. Figure 3a shows the tree for Query Q_1 after scanning the first two argument tuples. The path from the root to the leaf with time interval [3,3] yields the result tuple (2,[3,3]). The algorithm is constrained by the size of the available main memory, and it has a worst case time complexity of $O(n^2)$ for n argument tuples since the tree is not balanced.

An improvement, although with the same worst case complexity, is the *k*-ordered aggregation tree [6], which requires the argument tuples to be chronologically ordered to some degree. This allows to reduce the memory requirements by garbage collecting old nodes that will not be affected by any future tuples. Gao et al. [4] describe a number of parallel temporal aggregation algorithms that are all based on the aggregation tree.

Balanced Tree Moon et al. [7] propose the *balanced tree* algorithm for the main memory evaluation of ITA queries involving *sum*, *count*, and *avg*. As the argument tuples are scanned, their start and end times are stored in a balanced tree together with two values for each aggregate function being computed, namely the partial



a Aggregation tree

b Balanced tree

c SB-tree

d End-point tree

Temporal Aggregation. Figure 3. Different forms of tree structures for temporal aggregation.

aggregate result over all tuples that start and end here, respectively. An in-order traversal of the tree combines these values to compute the result relation. Whenever a node, *v*, is visited, a result tuple is produced over the interval that is formed by the time point of the previously visited node and the time point immediately preceding *v*. Figure 3b shows the balanced tree for Query Q_1. The aggregate value of the result tuple (2,[3,3]) is determined as $1 + (1 - 0) = 2$. Although the balanced tree requires less memory than the aggregation tree, it is constrained by the amount of available memory. For the *min* and *max* functions a merge-sort like algorithm is proposed. Both algorithms have $O(n \log n)$ time complexity for *n* argument tuples. To overcome the memory limitation, a bucket algorithm is proposed, which partitiones the argument relation along the time line and keeps long-lived tuples in a meta-array. Aggregation is then performed on each bucket in isolation.

SB-Tree Yang and Widom [14] propose a disk-based index structure, the *SB-tree*, together with algorithms for the incremental computation and maintenance of ITA and MWTA queries. It combines features from the segment tree and the B-tree and stores a hierarchy of time intervals associated with partially computed aggregates. To find the value of an aggregate at a time instant *t*, the tree is traversed from the root to the leaf that contains *t* and the partial aggregate values associated with the time intervals that contain *t* are combined. Figure 3c shows the SB-tree for Query Q_1. The value at time 8 results from adding 0 and 1 (associated with [10,11] and [7,11], respectively). The time complexity of answering an ITA query at a single time point is $O(h)$, where *h* is the height of the tree, and $O(h + r)$ for retrieving the result over a time interval, where *r* is the number of leaves that intersect with the given time interval. The same paper extends the basic SB-tree to compute MWTA. For a fixed window offset *w*, the timestamps of the argument tuples are extended by *w* to account for the tuples' contributions to the results at later time points. For arbitrary window offsets, a pair of SB-trees is required.

MVSB-Tree With the SB-tree, aggregate queries are always applied to an entire argument relation. The *multi-version SB-tree* (MVSB-tree) by Zhang et al. [15] tackles this problem and supports temporal aggregation coupled with non-temporal range predicates that select the tuples over which an aggregate is computed. The MVSB-tree is logically a sequence of SB-trees, one for each timestamp. The main drawbacks of this approach are: the tree might be larger than the argument relation, the range restriction is limited to a single non-timestamp attribute, and the temporal evolution of the aggregate values cannot be computed. Tao et al. [11] present two approximate solutions that address the high memory requirements of the MVSB-tree. They use an MVB-tree and a combination of B- and R-trees, respectively. These achieve linear space complexity in the size of the argument relation and logarithmic query time complexity.

MDTA Böhlen et al. [3] provide two memory-based algorithms for the evaluation of MDTA queries. The algorithm for fixed-interval semantics keeps in a *group table* the result groups that are extended with an additional column for each aggregate being computed. As the argument relation is scanned, all aggregate values to which a tuple contributes are updated. The group table contains then the result relation. The memory requirements only depend on the size of the result relation. With an index on the group table, the average runtime is *n* log *m* for *n* argument tuples and *m* result groups, the worst case being $O(nm)$ when each argument tuple contributes to each result tuple.

The algorithm for constant-interval semantics processes the argument tuples in chronological order and computes the result tuples as time proceeds. An *endpoint tree* maintains partial aggregate results that are computed over all argument tuples that are currently valid, and they are indexed by the tuples' end points. Figure 3d shows the endpoint tree for Query Q_1 after processing the first two argument tuples. When the third argument tuple is read, the result tuples (2,[3,3]) and (1,[4,5]) are generated by accumulating all partial aggregate values; the nodes are then removed from the tree. The size of the tree is determined by the maximal number of overlapping tuples, n_o. The average time complexity of the algorithm is nn_o. The worst-case complexity is $O(n^2)$, when the start and end points of all argument tuples are different and all tuples overlap.

Key Applications

Temporal aggregation is used widely in different data-intensive applications, which become more and more important with the increasing availability of huge volumes of data in many application domains, e.g., medical, environmental, scientific, or financial applications. Prominent examples of specific applications include data warehousing and stream processing. Time variance is one of four salient characteristics of a data warehouse, and there is general consensus that a data warehouse is likely to contain several years of time-referenced data. Temporal aggregation is a key operation for the analysis of such data repositories. Similarly, data streams are inherently temporal, and the computation of aggregation functions is by far the most important operation on such data. Many of the ideas, methods, and technologies from temporal aggregation have been and will be adopted for stream processing.

Future Directions

Future research work is possible in various directions. First, it may be of interest to study new forms of temporal aggregation. For example, a temporal aggregation operator that combines the best features of ITA and STA may be attractive. This operator should follow a data-driven approach that approximates the precision of ITA while allowing to limit the size of the result. Second, it is relevant to study efficient evaluation algorithms for more complex aggregate functions beyond the five standard functions for which most research has been done so far. Third, the results obtained so far can be adapted for and applied in related fields, including spatio-temporal databases where uncertainty is inherent as well as data streaming applications.

Cross-references

▶ Bi-Temporal Indexing
▶ Query Processing (in relational databases)
▶ Temporal Coalescing
▶ Temporal Data Mining
▶ Temporal Database
▶ Temporal Query Languages
▶ Temporal Query Processing

Recommended Reading

1. Ben-Zvi J. The Time Relational Model. Ph.D. thesis, Computer Science Department, UCLA, 1982.
2. Böhlen M.H., Gamper J., and Jensen C.S. How would you like to aggregate your temporal data? In Proc. 13th Int. Symp. on Temporal Representation and Reasoning, 2006, pp. 121–136.
3. Böhlen M.H., Gamper J., and Jensen C.S. Multi-dimensional aggregation for temporal data. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 257–275.
4. Gao D., Gendrano J.A.G., Moon B., Snodgrass R.T., Park M., Huang B.C., and Rodrigue J.M. Main memory-based algorithms for efficient parallel aggregation for temporal databases. Distrib. Parallel Dat., 16(2):123–163, 2004.
5. Gao D. and Snodgrass R.T. Temporal slicing in the evaluation of XML queries. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 632–643.
6. Kline N. and Snodgrass R.T. Computing temporal aggregates. In Int. Conf. on Data Engineering, 1995, pp. 222–231.
7. Moon B., Vega Lopez I.F., and Immanuel V. Efficient algorithms for large-scale temporal aggregation. IEEE Trans. Knowl. Data Eng., 15(3):744–759, 2003.
8. Navathe S.B. and Ahmed R. A temporal relational model and a query language. Inf. Sci., 49(1–3):147–175, 1989.
9. Snodgrass R.T., Gomez S., and McKenzie L.E. Aggregates in the temporal query language TQuel. IEEE Trans. Knowl. Data Eng., 5(5):826–842, 1993.
10. Tansel A.U. A statistical interface to historical relational databases. In Proc. Int. Conf. on Data Engineering, 1987, pp. 538–546.
11. Tao Y., Papadias D., and Faloutsos C. Approximate temporal aggregation. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 190–201.
12. Tuma P.A. Implementing Historical Aggregates in TempIS. M.Sc. thesis, Wayne State University, 1992.
13. Vega Lopez I.F., Snodgrass R.T., and Moon B. Spatiotemporal aggregate computation: a survey. IEEE Trans. Knowl. Data Eng., 17(2):271–286, 2005.

14. Yang J. and Widom J. Incremental computation and maintenance of temporal aggregates. VLDB J., 12(3):262–283, 2003.
15. Zhang D., Markowetz A., Tsotras V., Gunopulos D., and Seeger B. Efficient computation of temporal aggregates with range predicates. In Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2001, pp. 237–245.

Temporal Algebras

Abdullah Uz Tansel
Baruch College – CUNY New York, NY, USA

Synonyms

Historical algebras; Valid-time algebras; Transaction-Time algebras; Bitemporal algebras

Definition

Temporal algebra is a generic term for an algebra defined for a data model that organizes temporal data. A temporal data model may support Valid-time (the time over which a data value is valid), Transaction-time (time when a data value is recorded in the database), or both (Bitemporal). So an algebra can be defined for each case, a Valid-time relational algebra, a Transaction-time relational algebra, or a Bitemporal relational algebra, respectively. Temporal algebras include the temporal versions of relational algebra operations in addition to new operations for manipulating temporal data like Time-slice, Rollback, Temporal Coalesce, temporal restructuring operations, and others. For a Temporal algebra, it is desirable to be closed (common algebras are closed), a consistent extension of the relational algebra and to reduce to relational algebra when only current data are considered.

Historical Background

Temporal algebraic languages first appeared as extensions to the relational algebra in early 1980s, mostly Valid–time or Transaction-time algebras, followed by Bitemporal algebras. These extensions differed according to their operations and how temporal data are represented. McKenzie and Snodgrass surveyed temporal algebras and identified desirable criteria that are needed in a temporal algebra [10]. However, some of these criteria are conflicting.

Foundations

Temporal Algebra Basics

Let *T* represent the time domain which has a linear order under "≤". A time point (instant) is any element of *T*. A period is a consecutive sequence of time points. A temporal element is a set of disjoint maximal periods [6], and a temporal set is any set of time points. Any of these time constructs can be used to timestamp data values. A Bitemporal atom, <*Valid-Time*, *Transaction-time*, *Data*> asserts that *data* are valid during *Valid-time* and is recorded during *Transaction-time*. Either Valid-time or Transaction-time may be omitted and the result is a Valid-time Atom, or a Transaction-time atom, respectively.

A temporal relational algebra is closely related to how the temporal data (temporal atoms) are represented, i.e., the type of timestamps used, where they are attached (relations, tuples, or attribute values), and whether temporal atoms are kept atomic or broken into their components. In other words, time specification may be explicit or implicit. This in turn determines possible evaluation (semantics) of temporal algebra expressions. There are two commonly adopted approaches: (i) Snapshot (Point or Sequenced [1]) evaluation that manipulates the snapshot relation at each time point, like Temporal Logic [6,11,17]; (ii) Traditional (Nonsequenced [1]) evaluation that manipulates the entire temporal relation much like the traditional relational algebra. It is also possible to mix these approaches.The syntax and the operations of a temporal algebra are designed to accommodate a desired type of evaluation. Moreover, specifying temporal algebra operations at an attribute level, instead of tuples keeps the tuple structure intact after the operation is executed, so preserving the rest of a temporal relation that is beyond the scope of operation applied.

Let Q(A, B, C), R(A, D) and S(A, D) be temporal relations in attribute timestamping, whose attribute values are sets of temporal atoms except attribute A which has constant values. It is possibly a temporal grouping identifier [4] (or a temporal key). For the sake of simplicity, attributes B, C, and D are assumed to have one temporal atom in each tuple, i.e., they are already flattened. Q_t stands for the snapshot of temporal relation Q at time t. Temporal Algebras generally include temporal equivalents of traditional relational algebra operations. The five basic Temporal Algebra operations, $\cup^t$, $-^t$, π^t, σ^t, and $\times^t$ in snapshot evaluation are [2,4,5]:

- $R \cup^t S_t$ is $R_t \cup S_t$ for all t in T
- $R -^t S_t$ is $R_t - S_t$ for all t in T
- $\pi^t_{A1,A2,\ldots,An}$ (R) is $\pi_{A1,A2,\ldots,An}$ (R_t) for all t in T
- σ^t_F (R) is $\sigma_F(R_t)$ for all t in T; Formula F includes traditional predicates and temporal predicates like *Before*, *After*, *Overlaps*, etc
- $R \times^t Q$ is $R_t \times Q_t$ for all t in T

An important issue in Snapshot Evaluation is the homogeneity of the temporal relations. A temporal relation is homogenous if each tuple is defined on the same time, i.e., in a tuple, all the attributes have the same time reference [6]. If attributes in a tuple have different time references then a snapshot may have null values leading to complications. Thus, a Temporal Algebra may be homogenous, depending on the relations scheme on which it is defined.

In Temporal Algebras that use traditional evaluation, $\cup^t$, $-^t$, π^t, σ^t, and $\times^t$ may be defined exactly the same as the relational algebra operations or they have temporal semantics incorporated in their definitions. The temporal set operations may specially be defined by considering the overlapping timestamps in tuples. Two temporal tuples are value equivalent if their value components are the same, but their timestamps may be different. Let {(a1,<[2/07,11/07), d1>)} and {(a1, <[6/07,8/07), d1>)} be tuples in R and S, respectively. These two tuples are value equivalent. In case of $R \, U^t \, S$, value equivalent tuples are combined into one if their timestamps overlap. Considering the former tuples the result is {(a1,<[2/07,11/07), d1>)}. In case of $R -^t S$, the common portion of the timestamps for the value equivalent tuples is removed from the tuples of R. For the above tuples the result is {a1, <[2/07,6/07), d1>), (a1, <[8/07,11/07), d1>)}. Existence of value equivalent tuples makes query specification more complex but, query evaluation is less costly. On the other hand, eliminating them is also more costly. Temporal Coalescing operation combines value equivalent tuples into one tuple [2]. Temporal algebras may include aggregates.

The definition of Temporal Projection (π^t) is straightforward. However, it may generate value equivalent tuples much like the traditional projection operation creates duplicate tuples. Moreover, the projection operation may also be used to discard the time of a relation if it is explicitly specified. In the case of implicit time specification, it needs to be converted to an explicit specification before applying the projection operation. The formula F in the Selection operation ($\sigma^t_F(Q)$) may include time points, the end points of periods, and periods in temporal elements, or temporal predicates like *Before*, *After*, *Overlaps*, etc. It is possible to simulate the temporal predicates by conditions referring to time points or end points of periods.

Other temporal algebra operations such as Temporal Set Intersection or Temporal Join are similarly defined. There are different versions of Temporal Join. Intersection join is computed over the common time of operand relations (see the entry on temporal joins). For instance, if {(a1, <[1/07, 5/07), b1>, <[1/07, 4/07), c1>)} is a tuple in Q, the natural join ($Q \bowtie R$) contains the tuple {(a1, <[1/07, 5/07), b1>, <[1/07, 4/07), c1>, <[2/07,11/07), d1>)}. If this were an intersection natural join, times of the attributes in this tuple would be restricted to their common time period [2/07, 4/07). It is also possible to define temporal outer joins [11].

Temporal algebra operations are defined independent of time granularities. However, if operand relations are defined on different time granularities, a granularity conversion is required as part of processing the operation.

Algebras for Tuple Timestamping

In tuple timestamping relations are augmented with one column to represent time points, periods or temporal elements, or two columns to represent periods. Relation Q is represented as Q1(A, B, From, To) and Q2(A, C, From, To) where From and To are the end points of periods. Similarly, R(A, D, From, To) and S(A, D, From, To) correspond to the relations R and S, respectively. The tuple of Q given above is represented as the following tuples: (a1, b1, 1/07, 5/07) in Q1 and (a1, c1, 1/07, 4/07) in Q2. For accessing time points within a period snapshot evaluation may be used [6,17] or in case of Traditional Evaluation, attributes representing the end points of periods may be specified in operations. Another path followed is to define temporal expansion and contraction operations [9]. A period is expanded to all the time points included in it by temporal expansion and temporal contraction does the opposite, converts a sequence of time points to a period. Relation instances indexed by time points are used to define a temporal algebra by Clifford, Croker, and Tuzhilin [17].

Algebras for Attribute Timestamping

Timestamps are attached to attributes and N1NF relations are used and the entire history of an object is represented as a set of temporal atoms in one tuple. These temporal relations are called temporally grouped in contrast to temporally ungrouped relations that are based on tuple timestamping [17]. Naturally, temporally grouped algebras are more expressive than temporally ungrouped algebras and the former is more complex than the latter [17]. A temporal algebra that is based on snapshot evaluation and allows set theoretic operations on temporal elements is given in [6]. For the algebra expression e, the construct [[e]] returns the time over which e's result is defined [6] and it can further be used in algebraic expressions. An algebra, based on time points and lifespans, that uses snapshot evaluation is proposed in [3,5]. The nest and unnest operations for the transformations between 1NF and N1NF relations and operations which form and break temporal atoms are included in a temporal algebra [5,12,13]. N1NF Temporal relations and their algebras may or may not be homogonous [6,12,13].

Valid-time and Transaction-time Algebras

Most of the algebras mentioned above are Valid-time Relational Algebras. A Valid-time Relational Algebra includes additionally a Slice operation ($\varsigma\tau\Theta$) that is a redundant, but very useful operation. Let R be a Valid-time Relation and t be a time point (period, temporal element, or temporal set). Then, $\varsigma\tau\Theta_t$ (R) cuts a slice from R, the values that are valid over time t and returns them as a relation [1,5,12,13]. Common usage of the Slice operation is to express the "when" predicate in natural languages. Slice may also be incorporated into the selection operation or the specification of a relation to restrict the time of a temporal relation by a temporal element, i.e., R[t] [6]. Slice may be applied at an attribute level to synchronize time of one attribute by another attribute [5,12,13]. For instance, $\varsigma\tau\Theta_{B,C}$ (Q) restricts the time of attribute B by the time of attribute C. Applying the Slice operation on all the attributes by the same time specification returns a snapshot at the specified time.

A Transaction-time relational algebra includes a Rollback operation (τ), another form of Slice for rolling back to the values recorded at a designated time. Let R be a Transaction-time relation and t be a time point (period, temporal element, or temporal set). Then, τ_t (R) cuts a slice from R, the values that were recorded over time t and returns them as a relation [1,5,8]. Note the duality between the Valid-time relational algebra and the Transaction-time relational algebra; each has the same set of operations and an appropriate version of the slice operation.

Bitemporal Relational Algebras

Bitemporal algebra operations are more complicated than temporal algebras that support one time dimension only [1,8,15]. A Bitemporal algebra includes both forms of the Slice operation in addition to other algebraic operations. A Bitemporal query has a context that may or may not imply a Rollback operation [15]. However, once a Rollback operation is applied on a Bitemporal relation, Valid-time algebra operations can be applied on the result. It is also possible to apply Bitemporal Algebra operations on a bitemporal relation before applying a Rollback operation. In this case, the entire temporal relation is the context of the algebraic operations. In coalescing Bitemporal tuples, starting with Valid-time or Transaction-time may result in different coalesced tuples [8]. For the data maintenance queries, Valid-time needs to be coalesced within the Transaction-time.

Key Applications

Use of temporal algebra includes query language design [14], temporal relational completeness [16,17], and query optimization [14]. Some Relational algebra identities directly apply to temporal relational algebra whereas other identities do not hold due to the composite representation or semantics of temporal data [10]. Naturally, identities for the new temporal algebra operations and their interaction with the operations borrowed from the relational algebra need to be explored as well [5,7].

Cross-references

- ▶ Bitemporal Interval
- ▶ Bitemporal Relation
- ▶ Nonsequenced Semantics
- ▶ Relational Algebra
- ▶ Relational Model
- ▶ Sequenced Semantics
- ▶ Snapshot Equivalence
- ▶ Temporal Aggregates
- ▶ Temporal Coalescing
- ▶ Temporal Conceptual Models
- ▶ Temporal Data Models

- ▶ Temporal Element
- ▶ Temporal Query Languages
- ▶ Temporal Expression
- ▶ Temporal Homogeneity
- ▶ Temporal Joins
- ▶ Temporal Object-Oriented Databases
- ▶ Temporal Projection
- ▶ Temporal Query Optimization
- ▶ Temporal Query Processing
- ▶ Time Domain
- ▶ Time Interval
- ▶ Time Period
- ▶ Time Slice
- ▶ Transaction Time
- ▶ Value Equivalence
- ▶ Valid Time

Recommended Reading

1. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. ACM Trans. Database Syst., 25(4):407–456, 2000.
2. Böhlen M.H., Snodgrass R.T., and Soo M.D. Coalescing in temporal databases. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 180–191.
3. Clifford J. and Croker A. The historical relational data model (HRDM) and algebra based on lifespans. In Proc. 3th Int. Conf. on Data Engineering, 1987, pp. 528–537.
4. Clifford J., Croker A., and Tuzhilin A. On completeness of historical data models. ACM Trans. Database Syst., 19 (1):64–116, 1993.
5. Clifford J. and Tansel A.U. On an algebra for historical relational databases: two views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1985, pp. 247–265.
6. Gadia S.K. A homogeneous relational model and query languages for temporal databases. ACM Trans. Database Syst., 13(4):418–448, 1988.
7. Gadia S.K. and Nair S.S. Algebraic identities and query optimization in a parametric model for relational temporal databases. IEEE Trans. Knowl. Data Eng., 10(5):793–807, 1998.
8. Jensen C.S., Soo M.D., and Snodgrass R.T. Unifying temporal data models via a conceptual model. Inf. Syst., 19(7):513–547, 1994.
9. Lorentzos N.A. and Johnson R.G. Extending relational algebra to manipulate temporal data. Inf. Syst., 13(3):289–296, 1988.
10. McKenzie E. and Snodgrass R.T. Evaluation of relational algebras incorporating the time dimension in databases. ACM Comput. Surv., 23(4):501–543, 1991.
11. Soo M.D., Jensen C., and Snodgrass R.T. *1*An algebra for TSQL2. In TSQL2 Temporal Query Language, R.T. (ed.). R.T. Snodgrass (ed.). Kluwer Academic, Norwell, MA, 1995, pp. 505–546.
12. Tansel A.U. Adding time dimension to relational model and extending relational algebra. Inf. Syst., 11(4):343–355, 1986.
13. Tansel A.U. Temporal relational data model. IEEE Trans. Knowl. Database Eng., 9(3):464–479, 1997.
14. Tansel A.U., Arkun M.E., and Ozsoyoglu G. Time-by-example query language for historical databases. IEEE Trans. Softw. Eng., 15(4):464–478, 1989.
15. Tansel A.U. and Eren-Atay C. Nested bitemporal relational algebra. In Proc. 21st Int. Symp. on Computer and Information Sciences, 2006, pp. 622–633.
16. Tansel A.U. and Tin E. Expressive power of temporal relational query languages. IEEE Trans. Knowl. Data Eng., 9(1):120–134, 1997.
17. Tuzhilin A. and Clifford J. A temporal relational algebra as basis for temporal relational completeness. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 13–23.

Temporal Assignment

▶ Temporal Projection

Temporal Association Mining

▶ Temporal Data Mining

Temporal Coalescing

Michael Böhlen
Free University of Bozen-Bolzano, Bolzano, Italy

Definition

Temporal coalescing is a unary operator applicable to temporal databases that is similar to duplicate elimination in conventional databases. Temporal coalescing merges value-equivalent tuples, i.e., tuples with overlapping or adjacent timestamps and matching explicit attribute values. Tuples in a temporal relation that agree on the explicit attribute values and that have adjacent or overlapping timestamps are candidates for temporal coalescing. The result of operators may change if a relation is coalesced before applying the operator. For instance, an operator that counts the number of tuples in a relation or an operator that selects all tuples with a timestamp spanning at least 3 months are sensitive to temporal coalescing.

Historical Background

Early temporal relational models implicitly assumed that the relations were coalesced. Ben Zvi's Time Relational Model [13, Chap. 8], Clifford and Croker's Historical

Relational Data Model (HRDM) [13, Chap. 1], Navathe's Temporal Relational Model (TRM) [13, Chap. 4], and the data models defined by Gadia [13, pp. 28–66], Sadeghi [9] and Tansel [13, Chap. 7] all have this property. The term *coalesced* was coined by Snodgrass in his description of the data model underlying TQuel, which also requires temporal coalescing [10]. Later data models, such as those associated with HSQL [13, Chap. 5] and TSQL2 [11], explicitly required coalesced relations. The query languages associated with these data models generally did not include explicit constructs for temporal coalescing. HSQL is the exception; it includes a `COALESCE ON` clause within the select statement, and a `COALESCED` optional modifier immediately following `SELECT` [13, Chap. 5]. Some query languages that do not require coalesced relations provide constructs to explicitly specify temporal coalescing; VT-SQL [8] and ATSQL [2] are examples.

Navathe and Ahmed defined the first temporal coalescing algebraic operator; they called this COMPRESS [13, Chap. 4]. Sarda defined an operator called COALESCE [13, Chap. 5], Lorentzos' FOLD operator includes temporal coalescing [13, Chap. 3], Leung's second variant of a temporal select join operator TSJ_2 [13, Chap. 14] can be used to effect temporal coalescing, and TSQL2's representational algebra also included a coalesce operator [11].

In terms of performance and expressiveness Leung and Pirahesh provided a mapping of the coalesce operation into *recursive* SQL [6, p. 329]. Lorentzos and Johnson provided a translation of his FOLD operator into Quel [7, p. 295]. Böhlen et al. [3] show how to express temporal coalescing in terms of standard SQL and compare different implementations. SQL-based solutions to coalescing have also been proposed by Snodgrass [12] and Zhou et al. [14].

Foundations

Temporal databases support the recording and retrieval of time-varying information [13] and associate with each tuple in a temporal relation one or more timestamps that denote some *time periods*. The discussion assumes that each tuple is associated with a *valid time* attribute VT. This attribute is called the timestamp of the tuple. The timestamps are half open time periods: the start point is included but the end point is not. The non-timestamp attributes are referred to as the explicit attributes.

In a temporal database, tuples are uncoalesced when they have identical attribute values and their timestamps are either adjacent in time ("meet" in Allen's taxonomy [1]) or have some time in common. Consider the relations in Fig. 1. The relation records bonus payments that have been given to employees. Ron received two 2K bonuses: one for his performance from January 1981 to April 1981 and another one for his performance from May 1981 to September 1981. Pam received a 3K bonus for her performance from April 1981 to May 1981. Bonus1 is uncoalesced since the tuples for Ron have adjacent timestamps and can be coalesced. Bonus2 is coalesced. Coalescing Bonus1 yields Bonus2.

As with duplicate elimination in nontemporal databases, the result of some operators in temporal databases changes if the argument relation is coalesced before applying the operator [11]. For instance an operator that counts the number of tuples in a relation or an operator that selects all tuples with a timestamp spanning at least 3 months are sensitive to temporal coalescing.

In general, two tuples in a *valid time* relation are candidates for temporal coalescing if they have identical explicit attribute values (see *value equivalence* [10]) and have adjacent or overlapping timestamps. Such tuples can arise in many ways. For example, a projection of a coalesced temporal relation may produce an uncoalesced result, much as duplicate tuples may be produced by a duplicate preserving projection on a duplicate-free nontemporal relation. In addition, update and insertion operations may not enforce temporal coalescing, possibly due to efficiency concerns.

Bonus1

Name	Amount	VT
Ron	2K	[1981/01–1981/06)
Ron	2K	[1981/01–1981/05)
Pam	3K	[1981/05–1981/10)

Bonus2

Name	Amount	VT
Ron	2K	[1981/01–1981/10)
Pam	3K	[1981/04–1981/06)

Temporal Coalescing. Figure 1. Uncoalesced (Bonus1) and coalesced (Bonus2) valid time relations.

Thus, whether a relation is coalesced or not makes a semantic difference. In general, it is not possible to switch between a coalesced and an uncoalesced representation without changing the semantics of programs. Moreover, as frequently used database operations (projection, union, insertion, and update) may lead to potentially uncoalesced relations and because many (but not all) real world queries require coalesced relations, a fast implementation is imperative.

Temporal coalescing is potentially more expensive than duplicate elimination, which relies on an equality predicate over the attributes. Temporal coalescing also requires detecting if the timestamps of tuples overlap, which is an inequality predicate over the timestamp attribute. Most conventional DBMSs handle inequality predicates poorly; the typical strategy is to resort to exhaustive comparison when confronted with such predicates [5], yielding quadratic complexity (or worse) for this operation.

Implementing Temporal Coalescing

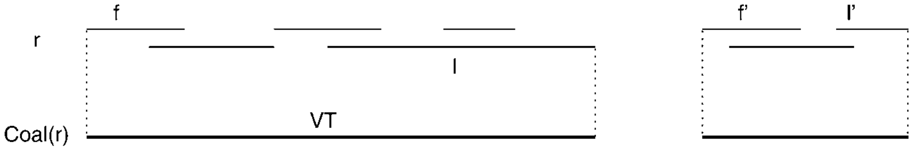
Temporal coalescing does not add expressive power to SQL. Assuming that time is linear, i.e., totally ordered, it is possible to compute a coalesced relation instance with a single SQL statement (see also [4, p. 291]). The basic idea is to use a join to determine the first (f) and last (l) time period of a sequence of *value equivalent* tuples with adjacent or overlapping timestamps as illustrated in Fig. 2.

The SQL code assumes that the *time period* is represented by start (*S*) and end (*E*) point, respectively. Besides start and end point there is an explicit attribute *c*. This yields a relation with schema *R*(*S*, *E*, *c*). Two subqueries are used to ensure that there are no temporal gaps (for example between l and f' is a temporal gap) and that the sequence is maximal (there is no tuple with a time period that starts before the start point of f and that temporally overlaps with f; there is no tuple with a time period that ends after the end point of l and that temporally overlaps with l), respectively.

```
SELECT DISTINCT f.S, l.E, f.c
FROM r AS f, r AS l
WHERE f.S < l.E
AND f.c = l.c
AND NOT EXISTS (SELECT *
                FROM r AS m
                WHERE m.c = f.c
                AND f.S < m.S AND
                m.S < l.E
AND NOT EXISTS  (SELECT *
                FROM r AS a1
                WHERE a1.c = f.c
                AND a1.S < m.S AND m.S
                <= a1.E))
AND NOT EXISTS (SELECT *
                FROM r AS a2
                WHERE a2.c = f.c
                AND (a2.S < f.S AND f.S
                <= a2.E OR
                   a2.S <= l.E AND l.E
                   < a2.E))
```

The above SQL statement effectively coalesces a relation. However, current database systems cannot evaluate this statement efficiently. It is possible to exploit the fact that only the maximal time periods are relevant. Rather than inserting a new tuple (and retaining the old ones) it is possible to update one of the tuples that was used to derive the new one. This approach can be implemented by iterating an update statement. The statement is repeated until the relation does not change anymore, i.e., until the fixpoint with respect to temporal coalescing is reached.

```
repeat
   UPDATE r l
   SET (l.E) =
     (SELECT MAX(h.E)
      FROM r h
      WHERE l.c = h.c
      AND l.S < h.S AND l.E >= h.S AND l.
      E < h.E)
```



Temporal Coalescing. Figure 2. Illustration of temporal coalescing.

```
    WHERE EXISTS (
      SELECT *
      FROM r h
      WHERE l.c = h.c
      AND l.S < h.S AND l.E >= h.S AND
      l.E < h.E)
  until fixpoint(r)
```

One means to further improve the performance is to use the DBMS as an enhanced storage manager and to develop main memory algorithms on top of it. Essentially, this means to load the relation into main memory, coalesce it manually, and then store it back in the database. If tuples are fetched ordered primarily by explicit attribute values and secondarily by start points it is possible to coalesce a relation with just a single tuple in main memory. The core of the C code of the temporal coalescing algorithm is displayed below. It uses ODBC to access the database.

```
SQLAllocEnv(&henv);
SQLAllocConnect(henv, &hdbc);
SQLConnect(hdbc,"Ora10g",
SQL_NTS,"scott",SQL_NTS,"tiger",
SQL_NTS;
SQLAllocHandle(SQL_HANDLE_STMT,
hdbc, &hstmt1)
SQLAllocHandle(SQL_HANDLE_STMT,
hdbc, &hstmt2)

/* initialize buffer curr_tpl (a one
tuple buffer) */
SQLExecDirect(hstmt1, "SELECT S, E, c
FROM r ORDER BY c, S", SQL_NTS))
curr_tpl.S = next_tpl.S;
curr_tpl.E = next_tpl.E;
curr_tpl.c = next_tpl.c;

/* open a cursor to store tuples back
in the DB */
SQLPrepare(hstmt2,  "INSERT  INTO
r_coal VALUES (?,?,?)", SQL_NTS)

/* main memory temporal coalescing */
while     (SQLFetch(hstmt1)     !=
SQL_NO_DATA){ /* fetch all tuples */
  if (curr_tpl.c == next_tpl.c &&
  next_tpl.S <= curr_tpl.E){
    /* value-equivalent and
    overlapping*/
    if (next_tpl.E > curr_tpl.E)
    curr_tpl.E = next_tpl.E;
  } else{
   /* not value-equivalent or non-overlapping */
   SQLExecute(hstmt2) /* store back
   current tuple */
   curr_tpl.S = next_tpl.S;
   curr_tpl.E = next_tpl.E;
   curr_tpl.c = next_tpl.c;
  }
}
SQLExecute(hstmt2) /* store back current tuple */
```

Key Applications

Temporal coalescing defines a normal form for temporal relations and is a crucial and frequently used operator for applications that do not want to distinguish between snapshot equivalent relations. Applications that allow to distinguish between snapshot equivalent relations have temporal coalescing as an explicit operator similar to duplicate elimination in existing database systems.

Cross-references

- ▶ Snapshot Equivalence
- ▶ Temporal Database
- ▶ Temporal Data Model
- ▶ Time Domain
- ▶ Time Interval
- ▶ Time Period
- ▶ Valid Time

Recommended Reading

1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 16(11):832–843, 1983.
2. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. ACM Trans. Database Syst., 25(4):48, 2000.
3. Böhlen M.H., Snodgrass R.T., and Soo M.D. Coalescing in temporal databases. In Proc. 22th Int. Conf. on Very Large Data Bases. 1996, pp. 180–191.
4. Celko J. SQL for Smarties: Advanced SQL Programming. Morgan Kaufmann, 1995.
5. Leung C. and Muntz R. Query Processing for Temporal Databases. In Proc. 6th Int. Conf. on Data Engineering, 1990, pp. 200–208.
6. Leung T.Y.C. and Pirahesh H. Querying Historical Data in IBM DB2 C/S DBMS Using Recursive SQL. In J. Clifford, A. Tuzhilin (eds.). Recent Advances in Temporal Databases, Springer, 1995.

7. Lorentzos N. and Johnson R. Extending relational algebra to manipulate temporal data. Inf. Syst., 15(3), 1988.
8. Lorentzos N.A. and Mitsopoulos Y.G. Sql extension for interval data. IEEE Trans. Knowl. Data Eng., 9(3):480–499, 1997.
9. Sadeghi R., Samson W.B., and Deen S.M. HQL – A Historical Query Language. Technical report, Dundee College of Technology, Dundee, Scotland, September 1987.
10. Snodgrass R.T. The temporal query language TQuel. ACM Trans. Database Syst., 12(2):247–298, June 1987.
11. Snodgrass R.T. (ed.). The TSQL2 Temporal Query Language. Kluwer Academic, Boston, 1995.
12. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, 2000.
13. Tansel A., Clifford J., Gadia S., Jajodia S., Segev A., and Snodgrass R.T. 1Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, Redwood City, California, 1993.
14. Zhou X., Wang F., and Zaniolo C. Efficient Temporal Coalescing Query Support in Relational Database Systems. In Proc. 17th Int. Conf. Database and Expert Syst. Appl., 2006, pp. 676–686.

Temporal Compatibility

Michael H. Böhlen[1], Christian S. Jensen[2], Richard T. Snodgrass[3]

[1]Free University of Bozen-Bolzano, Bozen-Bolzano, Italy

[2]Aalborg University, Aalborg, Denmark

[3]University of Arizona, Tucson, AZ, USA

Definition

Temporal compatibility captures properties of temporal languages with respect to the nontemporal languages that they extend. Temporal compatibility, when satisfied, ensures a smooth migration of legacy applications from a non-temporal system to a temporal system. Temporal compatibility dictates the semantics of legacy statements and constrains the semantics of temporal extensions to these statements, as well as the language design.

Historical Background

Since the very early days of temporal database research, the compatibility with legacy languages and systems has been considered, but the first comprehensive investigation was reported by Bair et al. [2]. Compatibility issues are common for work done in the context of systems and commercial languages, such as SQL or Quel. Theoretical or logic-based approaches usually do not explore compatibility notions since they tend to strictly separate temporal from nontemporal structures.

Foundations

Motivation

Most data management applications manage time-referenced, or temporal, data. However, these applications typically run on top of relational or object-relational database management systems (DBMSs), such as DB2, Oracle, SQL Server, and MySQL, that offer only little built-in support for temporal data management. Organizations that manage temporal data may benefit from doing so using a DBMS with built-in temporal support. Indeed, it has been shown that using a temporal DBSM in place of a non-temporal DBMS may reduce the number of lines of query language code by a factor of three, with the conceptual complexity of application development decreasing even further [13].

Then, what hinders an organization from adopting a temporal DBMS? A key observation is that an organization is likely to already have a portfolio of data management applications that run against a non-temporal DBMS. In fact, the organization is likely to have made very large investments in its legacy systems, and it depends on the functioning of these systems for the day-to-day operation of its business.

It should be as easy as possible for the organization to migrate to a temporal DBMS. It would be attractive if all existing applications would simply work without modification on the temporal DBMS. This would help protect the organization's investment in its legacy applications. The opposite, that of having to rewrite all legacy applications, is a daunting proposition.

However, this type of compatibility is only the first step. The next step is to make sure that legacy applications can coexist with new applications that actually exploit the enhanced temporal support of the new DBMS. These applications may query and modify the same (legacy) tables. It should thus be possible to add a new temporal dimension to existing tables, without this affecting the legacy applications that use these tables.

Next, the organization maintains a large investment in the skill set of its IT staff. In particular, the staff is skilled at using the legacy query language, typically SQL. The new, temporal query language should leverage this investment, by making it easy for the application programmers to write temporal queries against temporal relations.

Next four specific compatibility properties that aim to facilitate the migration from a non-temporal DBMS to a temporal DBMS are considered.

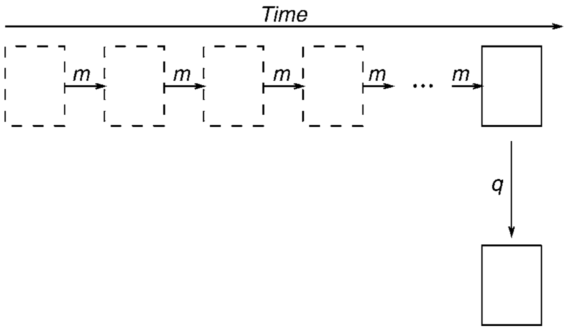
Upward Compatibility

The property of upward compatibility states that all language statements expressible in the underlying non-temporal query language must evaluate to the same result in the temporal query language, when evaluated on non-temporal data.

Figure 1 illustrates this property. In the figure, a conventional table is denoted with a rectangle. The current state of this table is the rectangle in the upper-right corner. Whenever a modification is made to this table, the previous state is discarded; hence, at any time, only the current state is available. The discarded prior states are denoted with dashed rectangles; the right-pointing arrows denote the modifications that took the table from one state to the next.

When a query *q* is applied to the current state of a table, a resulting table is computed, shown as the rectangle in the bottom right corner. While this figure only concerns queries over single tables, the extension to queries over multiple tables is clear.

As an example, consider a hypothetical temporal extension of the conventional query language SQL [9]. Upward compatibility states that (i) all instances of tables in SQL are instances of tables in this extension, (ii) all SQL modifications to tables result in the same tables when the modifications are evaluated according to the semantics of the extension, and (iii) all SQL queries result in the same tables when the queries are evaluated according to the extension.



Temporal Compatibility. Figure 1. Upward compatible queries.

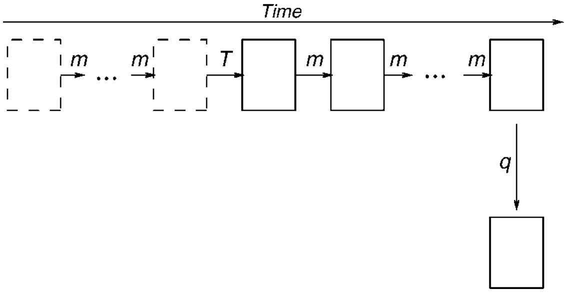
By requiring that a temporal extension to SQL is a strict superset (i.e., only *adding* constructs and semantics), it is relatively easy to ensure that the extension is upward compatible with SQL. TOSQL [1], TSQL [10], HSQL [11], IXSQL [7], TempSQL [5], and TSQL2 [12] were designed to satisfy upward compatibility.

While upward compatibility is essential in ensuring a smooth transition to a new temporal DBMS, it does not address all aspects of migration. It only ensures the operation of existing legacy applications and does not address the coexistence of these with new applications that exploit the improved temporal support of the DBMS.

Temporal Upward Compatibility

The property of temporal upward compatibility (TUC) addresses the coexistence of legacy and new applications. Assume an existing or new application needs support for the temporal dimension of the data in one or more of the existing tables that record only the current state. This is best achieved by changing the snapshot table to become a temporal table. It is undesirable to be forced to change the application code that accesses the snapshot table when that table is made temporal. TUC states that conventional queries on temporal data yield the same results as do the same queries on a conventional database formed by taking a timeslice at "now." TUC applies also to modifications, views, assertions, and constraints [2].

Temporal upward compatibility is illustrated in Fig. 2. When temporal support is added to a table, the history is preserved and modifications over time are retained. In the figure, the rightmost dashed state was the current state when the table was made temporal. All subsequent modifications, denoted again by



Temporal Compatibility. Figure 2. Temporal upward compatibility.

arrows, result in states that are retained, and thus are represented by solid rectangles. Temporal upward compatibility ensures that the states will have identical contents to those states resulting from modifications of the snapshot table. The query q is a conventional SQL query. Due to temporal upward compatibility, the semantics of this query must not change if it is applied to a temporal table. Hence, the query only applies to the current state, and a snapshot table results.

Most temporal languages were not designed with TUC in mind, and TOSQL [1], TSQL [10], HSQL [11], IXSQL [7], and TSQL2 [12] do not satisfy TUC. The same holds for temporal logics [4]. TempSQL [5] introduces a concept of different types of users, classical and system user. TempSQL satisfies TUC for classical users. ATSQL [3] was designed to satisfy TUC.

Snapshot Reducibility

This third property states that for each conventional query, there is a corresponding temporal query that, when applied to a temporal relation, yields the same result as the original snapshot query when applied separately to every snapshot state of the temporal relation.

Graphically, snapshot reducibility implies that for all conventional query expressions q in the snapshot model, there must exist a temporal query q^t in the temporal model so that for all db^t and for all c, the commutativity diagram shown in Fig. 3 holds.

This property requires that each query q (or operator) in the snapshot model has a counterpart q^t in the temporal model that is snapshot reducible with respect to the original query q. Observe that q^t being snapshot reducible with respect to q poses no syntactical restrictions on q^t. It is thus possible for q^t to be quite different from q, and q^t might be very involved. This is undesirable: the temporal model should be a straightforward extension of the snapshot model.

Most languages satisfy snapshot reducibility, but only because corresponding non-temporal and temporal statements do not have to be syntactically similar. This allows the languages to formulate for each nontemporal statement a snapshot reducible temporal statement, possibly a very different and complex statement.

Sequenced Semantics

This property addresses the shortcoming of snapshot reducibility: it requires that q^t and q be syntactically identical, modulo an added string.

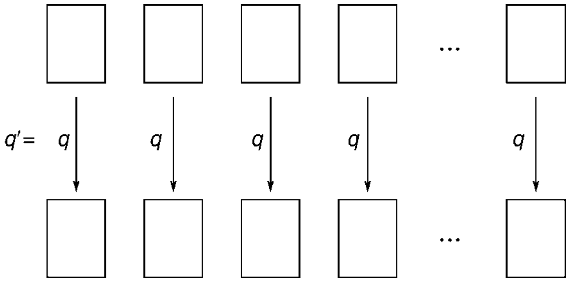
Figure 4 illustrates this property. This figure depicts a temporal query, q', that, when applied to a temporal table (the sequence of values across the top of the figure), results in a temporal table, which is the sequence of values across the bottom.

The goal is to ensure that an application programmer who is familiar with the conventional query language is able to easily formulate temporal generalizations of conventional queries using the temporal query language. This is achieved if a query q can be made temporal by simply adding a string to it. The syntactical similarity requirement of sequenced semantics makes this possible. Specifically, the meaning of q' is precisely that of applying the analogous *non-temporal* query q on each value of the argument table (which must be temporal), producing a state of the result table for each such application.

Most temporal languages do not offer sequenced semantics. As an exception, ATSQL [3] prepends the modifier SEQUENCED, together with the time dimension (valid time or transaction time, or both), to nontemporal statements to obtain their snapshot reducible generalizations. Temporal Logic [4] satisfies sequenced semantics as well: the original nontemporal statement yields sequenced semantics when evaluated over a corresponding temporal relation.

$$\begin{tikzcd}
db^t \arrow[r, "q^t"] \arrow[d, "\text{timeslice at } c"'] & q^t(db^t) \arrow[d, "\text{timeslice at } c"] \\
slice(db^t, c) \arrow[r, "q"'] & q(slice(db^t, c)) = slice(q^t(db^t), c)
\end{tikzcd}$$

Temporal Compatibility. Figure 3. Snapshot reducibility.



Temporal Compatibility. Figure 4. Sequenced semantics.

Key Applications

Temporal compatibility properties such as those covered here are important for the adoption of temporal database technology in practice. The properties are important because temporal technology is likely to most often be applied in settings where substantial investments have already been made in database management staff and applications. The properties aim at facilitating the introduction of temporal database technology in such settings.

Properties such as these are easily as crucial for the successful adoption of temporal database technology as is highly sophisticated support for the querying of time-referenced data.

Given the very significant decrease in code size and complexity for temporal applications that temporal database technology offers, it is hoped that other DBMS vendors will take Oracle's lead and incorporate support for temporal databases into their products.

Future Directions

Further studies of compatibility properties are in order. For example, note that temporal upward compatibility addresses the case where existing tables are snapshot tables that record only the current state. However, in many cases, the existing tables may already record temporal data using a variety of ad-hoc formats. The challenge is then how to migrate such tables to real temporal tables while maintaining compatibilities.

Next, it is felt that much could be learned from conducting actual case studies of the migration of real-world legacy applications to a temporal DBMS.

Cross-references

- ▶ Period-Stamped Temporal Models
- ▶ Point-Stamped Temporal Models
- ▶ Sequenced Semantics
- ▶ Snapshot Equivalence
- ▶ Temporal Database
- ▶ Temporal Data Models
- ▶ Temporal Element
- ▶ Temporal Query Languages
- ▶ Time Domain
- ▶ Time Interval
- ▶ Time Period
- ▶ Valid Time

Recommended Reading

1. Ariav G. A temporally oriented data model. ACM Trans. Database Syst., 11(4):499–527, December 1986.
2. Bair J., Böhlen M., Jensen C.S., and Snodgrass R.T. Notions of upward compatibility of temporal query languages. Bus. Inform. (Wirtschafts Informatik), 39(1):25–34, February 1997.
3. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. ACM Trans. Database Syst., 25(4):407–456, December 2000.
4. Chomicki J., Toman D., and Böhlen M.H. Querying ATSQL databases with temporal logic. ACM Trans. Database Syst., 26(2):145–178, June 2001.
5. Gadia S.K. and Nair S.S. Temporal databases: A prelude to parametric data. In Temporal Databases: Theory, Design, and Implementation, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R.T. Snodgrass (eds.). Benjamin/Cummings, Redwood City, CA, USA, 1993, pp. 28–66.
6. Jensen C.S., Soo M.D., and Snodgrass R.T. Unifying temporal data models via a conceptual model. Inf. Syst., 19(7):513–547, December1994.
7. Lorentzos N.A. and Mitsopoulos Y.G. SQL extension for interval data. IEEE Trans. Knowl. Data Eng., 9(3):480–499, 1997.
8. McKenzie E. and Snodgrass R.T. An evaluation of relational algebras incorporating the time dimension in databases. ACM Comput. Surv., 23(4):501–543, December 1991.
9. Melton J. and Simon A.R. Understanding the New SQL: A Complete Guide. Morgan Kaufmann, San Mateo, CA, USA, 1993.
10. Navathe S. and Ahmed R. Temporal extensions to the relational model and SQL. In Temporal Databases: Theory, Design, and Implementation, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R.T. Snodgrass (eds.). Benjamin/Cummings, Redwood City, CA, USA, 1993, pp. 92–109.
11. Sarda N. HSQL: A historical query language. In Temporal Databases: Theory, Design, and Implementation, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R.T. Snodgrass (eds.). Benjamin/Cummings, Redwood City, CA, USA, 1993, pp. 110–140.
12. Snodgrass R.T. The TSQL2 Temporal Query Language. Kluwer, Boston, USA, 1995.
13. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, San Francisco, CA, USA, July 1999.

Temporal Conceptual Models

Vijay Khatri
Indiana University, Bloomington, IN, USA

Definition

A *conceptual model* provides a notation and formalism that can be used to construct a high-level, implementation-independent description of selected aspects of the "real world," termed a miniworld. This process is called conceptual modeling, and the resulting description is referred to as a *conceptual schema*. Conceptual modeling is an important part of systems analysis and design. A *temporal conceptual model* provides a notation and formalism with built-in support for capturing temporal aspects of a miniworld during conceptual design.

Historical Background

Temporal applications need to represent data semantics not only related to "what" is important for the application, but also related to "when" it is important. The history of temporal conceptual models can be viewed in terms of two generations. The *first generation temporal conceptual models*, e.g., [2,14], provide support for only *user-defined time*; see Fig. 1a for an example. In contrast to the first generation, the *second generation temporal conceptual models*, e.g., [5,9,16], provide varying degree of support for temporal aspects; see Fig. 1b for an example. Because the first generation temporal conceptual models provide support for representation of only user-defined time, they may be thought of as "almost" time-agnostic conceptual models; on the other hand, second generation temporal conceptual models that support temporal semantics, e.g., event, state, valid time, transaction time, may be construed as time-aware conceptual models.

Foundations

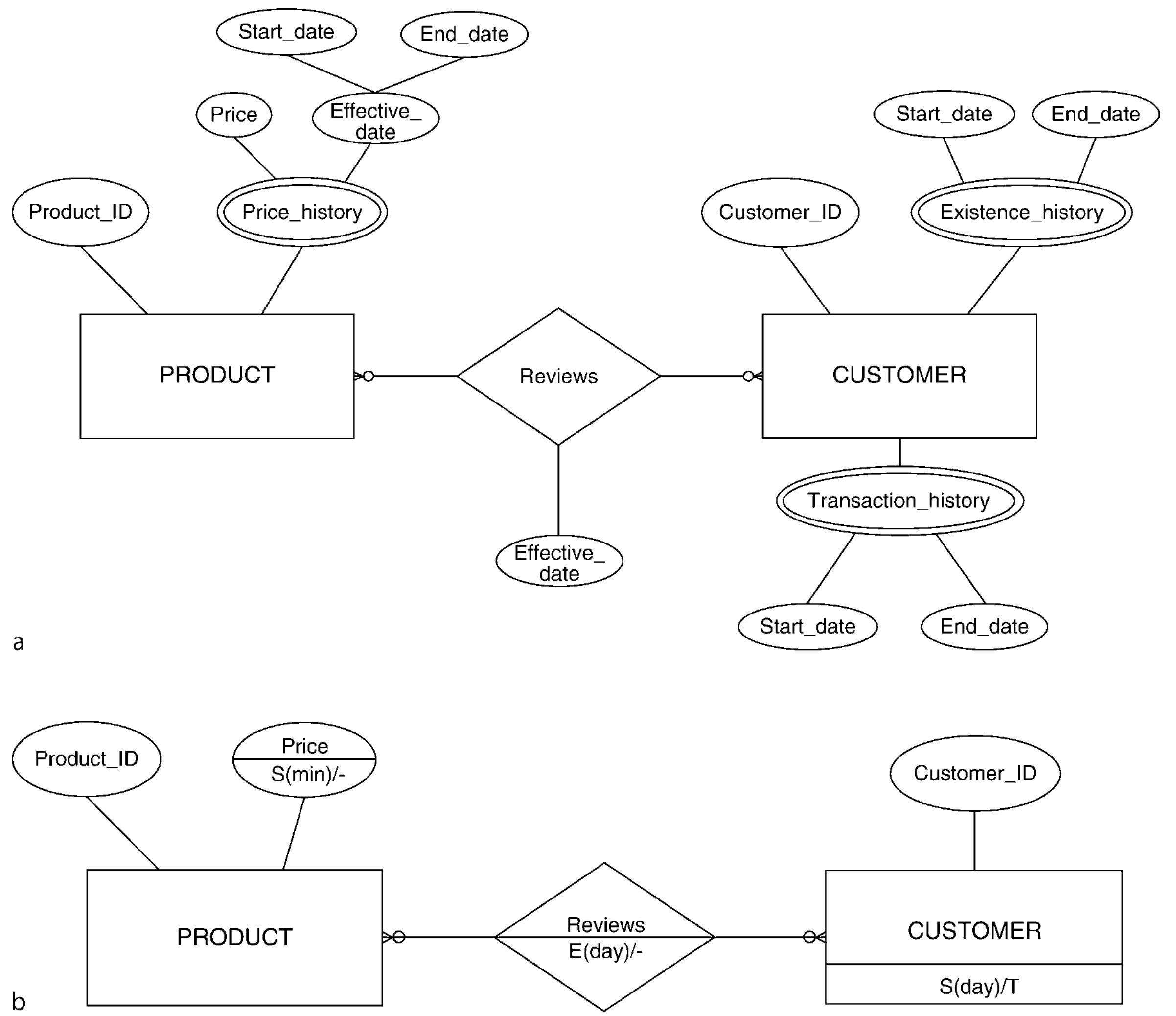
To highlight core concepts developed in the research related to temporal conceptual models, a framework of linguists is adopted that studies symbols with respect to three dimensions: syntactics, semantics and pragmatics [13]. The *syntactics* dimension includes formal relation between symbols, the *semantics* dimension involves the study of symbols in relation to the designatum (i.e., what the sign refers to) and the *pragmatics* dimension includes the relation between symbols and the interpreter.

In the following, a motivating example is employed to differentiate between first and second generation temporal conceptual models. Core concepts related to temporal conceptual models are described using syntactics, semantics and pragmatics.

Motivating Example

Figure 1a (first generation temporal conceptual model) provides an example of an ER schema that requires preserving the history of "prices" (of PRODUCT). Additionally, there is another entity type, CUSTOMER, whose existence needs to be modeled; the existence history needs to take into account both when the customer exists in the real world (Existence_history) and when the customer was added to the database (Transaction_history). Further, a CUSTOMER "reviews" PRODUCTs and the Effective_date on which the PRODUCT was reviewed needs to be captured as well. Because the first generation conceptual models do not provide a mechanism to represent temporal concepts, e.g., valid time, transaction time, event and state, these are all represented using only user-defined time. For example, the schema cannot differentiate Existence_history from Transaction_history, which are both represented simply as multi-valued attributes (double-lined ellipse). Additionally, the database analyst needs to make ad-hoc decisions related to granularity of a user-defined attribute such as Transaction_history. Start_date during implementation. As a result of the lack of a mechanism for directly mapping the miniworld to its representation, database designers are left to discover, design, and implement the temporal concepts in an ad-hoc manner.

The second generation temporal conceptual schema, referred to as the ST USM (geoSpatio-Temporal Unifying Semantic Model) schema [9] shown in Fig. 1b employs a textual string to represent temporal semantics. For example, "Price" is associated with valid time, which is represented as state ("S") with granularity of "min"(ute); further, the transaction time related to price is not relevant ("–"). The temporal semantics associated with "Price" are therefore represented by a textual string of valid time state ("S"), followed by a slash ("/"), followed by the specification of transaction time ("–"): "S (min)/–". Because both the existence (or valid) time and transaction time need to be recorded for the entity type, CUSTOMER, the annotation string for CUSTOMER is specified as "S(day)/T". Note that the granularity of transaction time is not specified



Temporal Conceptual Models. Figure 1. Examples of the two generations of temporal conceptual models.

because it is system-defined. A CUSTOMER "reviews" a PRODUCT at a certain point in time (event, E), captured to the granularity of day ("E (day)/–").

In summary, while the first generation temporal conceptual models provide a mechanism to represent only user-defined time, the second generation temporal conceptual models provide a mechanism to represent temporal data semantics. The readers are referred to Gregersen and Jensen [6] for a survey on various temporal conceptual models of the second generation.

Syntactics

Prior research has employed both graphical (see, for example, the TimeER Model [6]) and textual (see, for example, the Temporal Entity Relationship Model, TERM [11]) syntax to represent the temporal data semantics.

While some of graphical temporal conceptual models have changed the semantics of the constructs of conventional conceptual models (see, for example, [2]), others have proposed a new formalism for representing temporal aspects. The Temporal EER (TEER) Model [3] gave *new* meaning to *extant* ER modeling constructs such as the entity type, the attribute and the relationship; for example, each entity of an entity type is associated with a temporal element that represents the lifespan of the entity, i.e., the semantics of an entity type in a conventional conceptual model was changed. On the other hand, most of the graphical temporal conceptual models propose *new* constructs that represent the temporal aspects.

Prior research has employed two ways to graphically represent temporal aspects using new constructs; they are referred to as augmented (see, for example, the TimeER Model [6] and ST USM [9]) and standalone (see, for example, the Relationships, Attributes, Keys and Entities (RAKE) Model [4]). The *augmented approaches* construe second generation conceptual schemas as "constrained" first generation schemas. For example, ST USM employs a "shorthand" for temporal semantics that is represented as annotations (see, for example, Fig. 1b); however, the semantics of a second generation schema (ST USM schema) can be "unwrapped" using a first generation schema (USM schema) and a set of constraints. The readers are referred to [9] for examples of and procedure for "unwrapping" of the semantics of an annotated schema. In contrast, the *standalone approaches* suggest *new* constructs for representing the temporal aspects. The *augmented approaches* provide a mechanism for capturing temporal data semantics at the second level of abstraction; such approaches *deliberately* defer elicitation of the temporal data semantics ("when") from the first level of abstraction that focuses on "what" is important for the application. In contrast, the *standalone approaches* provide a single level of abstraction for representing both "what" and "when."

Having outlined different syntax adopted by various conceptual models, the temporal semantics that need to be captured in a temporal conceptual model are described next.

Semantics

Based on [8], definitions of different temporal aspects that need to be represented in a temporal conceptual model are outlined below.

An *event* occurs at a point in time, that is, it has no duration (for example, a special promotion for a product is scheduled on Christmas Eve this year (2007–12–24)), while a *state* has duration (for example, a certain price for a product is valid from 5:07 P.M. on 2005–11–11 to 5:46 P.M. on 2007–1–11).

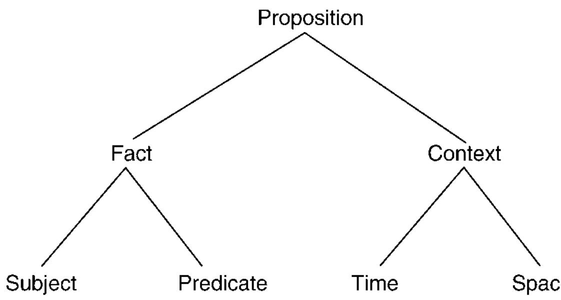
Facts can interact with time in two orthogonal ways, resulting in transaction time and valid time. *Transaction time* links a fact to the time that it becomes current in the database, and implies the storage of versions of data. The data semantics of transaction time associated with a fact require that the fact can exist in certain time periods in the past until *now* (state). *Valid time* is used to record the time at which a particular fact is true in the real world and implies the storage of histories related to facts. The data semantics of valid time associated with a fact imply that the fact can exist at certain points in time (events) or in certain time periods (states), in the past, the present, or the future.

Granularities, which are intrinsic to temporal data, provide a mechanism to hide details that are not known or not pertinent for an application. Day, minute, and second are examples of temporal granularities related to the Gregorian calendar. The price history for a manufacturing application may, for example, be associated with a temporal granularity of "day," while the representation of price history for a stock market application may require a temporal granularity of "minute" or even "second."

Pragmatics

Prior research suggests that "effective exchange of information between people and machines is easier if the data structures that are used to organize the information in the machine correspond in a natural way to the conceptual structures people use to organize the same information" [12]. Three criteria play a role in how an "interpreter" (users) interacts with "symbols" (conceptual schema): (i) "internal" representation; (ii) snapshot reducibility; (iii) upward compatibility. While snapshot reducibility and upward compatibility may be rooted in syntactics and semantics, they affect the pragmatic goal, comprehension.

Internal Representation All human knowledge is stored as abstract conceptual propositions. Based on propositions, Anderson and Bower's [1] Human Associative Model (HAM) represents information in the long-term memory as shown in Fig. 2. A *proposition*



Temporal Conceptual Models. Figure 2. Internal representation of human knowledge. (Adapted from HAM Model [1].)

is an assertion about the real world that is composed of a *fact* and *context* (associated with the fact). A *subject* and *predicate* correspond with a topic and a comment about the topic. For some applications, the context in which the fact is true can be the key to reasoning about the miniworld. This context in turn, is composed of *time* and *location* associated with the fact. Note that the "context" element is orthogonal to the "fact" element and specifies the temporal reality for which the fact is true. (This entry does not cover spatial aspects; see [9] for details on a spatio-temporal conceptual model.) An augmented approach that segregates "what" from "when" corresponds with the way humans naturally organize temporal information and should, thus, support comprehension of the schema.

Snapshot Reducibility Snapshot reducibility implies a "natural" generalization of the syntax and semantics of extant conventional conceptual models, e.g., the ER Model [2], for incorporating the temporal extension. Snapshot reducibility ensures that the semantics of a temporal model are understandable "in terms of" the semantics of the conventional conceptual model. Here, the overall objective is to help ensure minimum additional investment in a database analyst training.

For example, in a "conventional" conceptual model a *key attribute* uniquely identifies an entity (at a point in time). A *temporal key* implies uniqueness at *each point in time*. As may be evident, the semantics of a temporal key here are implied by the semantics of a key in a "conventional" conceptual model.

Upward Compatibility Upward compatibility refers to the ability to render a conventional conceptual schema temporal without impacting or negating that legacy schema, thus, protecting investments in the existing schemas. It also implies that both the legacy schemas and the temporal schemas can co-exist. Upward compatibility requires that the syntax and

Temporal Conceptual Models. Table 1. Summary of a sample of first and second generation temporal conceptual models

	Syntactics		Semantics			Pragmatics		
	Syntax	User-defined time	Valid time and transaction time	Event and state	Granularity	Consideration of internal representation	Upward compatibility	Snapshot reducibility
First Generation								
ER Model [2]	• "What": Graphical • "When": NA	Yes	No	No	No	NA	NA	NA
Second Generation								
TERM [11]	• "What": Textual • "When": Textual	Yes	Valid time only	Both	Yes	No	No	No
TERC+ [16]	• "What": Graphical • "When": Graphical	Yes	Valid time only	Both	No	No	Yes	Yes
TimeER [5]	• "What": Graphical • "When": Textual	Yes	Both	Both	No	No	Yes	Yes
ST USM [9]	• "What": Graphical • "When": Textual	Yes	Both	Both	Yes	Yes	Yes	Yes

semantics of the traditional conceptual model remain unaltered. An augmented approach that extends conventional conceptual models would ensure upward compatibility.

Summary

Because one of the important roles of conceptual modeling is to support user-database analyst interaction, the linguistics-based framework of evaluation is broader: it not only includes syntactics and semantics but also includes cognitive aspects in conceptual design (pragmatics). Table 1 summarizes the evaluation of a first generation and a few second generation temporal conceptual models.

Key Applications

There are several applications of this research, both for researchers and practitioners. (i) A temporal conceptual model can help support elicitation and representation of temporal data semantics during conceptual design. (ii) A temporal conceptual schema can, thus, be the basis for the logical schema and the database. (iii) A temporal conceptual modeling approach can be used as the basis for developing a design-support environment. Such a design support environment can be integrated with tools such as ERWin. (http://www.ca.com/us/products/product.aspx?id=260)

Future Directions

Future research should explore how the temporal schema can be used as the canonical model for information integration of distributed temporal databases. A temporal conceptual model should also be extended to incorporate schema versioning.

While an initial user study has been conducted [10], future research should further evaluate temporal conceptual modeling using, e.g., protocol analysis. Studies that address *how* problem solving occurs focus on "opening up the black box" that lies between problem-solving inputs and outputs; that is, such studies investigate what happens during individual problem solving (*isomorphic approach*) rather than simply observing the effects of certain stimuli averaged over a number of cases, as in traditional studies (*paramorphic approach*) [7]. The most common approach to opening up the black box is to examine the characteristics of the problem-solving process using protocol analysis.

Experimental Results

To evaluate the augmented temporal conceptual design approach, a user experiment [10] views conceptual schema comprehension in terms of matching the *external problem representation* (i.e., conceptual schema) with *internal task representation*, based on the theory of HAM and the theory of cognitive fit [15]. The study suggests that the similarity between annotated schemas (external representation) and the HAM model of internal memory results in cognitive fit, thus, facilitating comprehension of the schemas.

Cross-references

- ▶ Now in Temporal Databases
- ▶ Schema Versioning
- ▶ Sequenced Semantics
- ▶ Supporting Transaction Time Databases
- ▶ Temporal Data Models
- ▶ Temporal Granularity

Recommended Reading

1. Anderson J.R. and Bower G.H. Human Associative Memory. Washington, D.C.: V. H. Winston & Sons, 1973.
2. Chen P.P. The entity-relationship model – toward a unified view of data. ACM Trans. Database Syst., 1(1):9–36, 1976.
3. Elmasri R., Wuu G., and Kouramajian V. A temporal model and query language for EER databases. In Temporal Databases: Theory, Design and Implementation, A. Tansel (ed.). Benjamin/Cummings, Menlo Park, CA, 1993, pp. 212–229.
4. Ferg S. Modeling the time dimension in an entity-relationship diagram. In Proc. 4th Int. Conf. on Entity-Relationship Approach, 1985, pp. 280–286.
5. Gregersen H. and Jensen C. Conceptual Modeling of Time-Varying Information. TIMECENTER Technical Report TR-35, September 10, 1998.
6. Gregersen H. and Jensen C.S. Temporal entity-relationship models-A survey. IEEE Trans. Knowl. Data Eng., 11(3):464–497, 1999.
7. Hoffman P.J. The paramorphic representation of clinical judgment. Psychol. Bull., 57(2):116–131, 1960.
8. Jensen C.S., Dyreson C.E., Bohlen M., Clifford J., Elmasri R., Gadia S.K., Grandi F., Hayes P., Jajodia S., Kafer W., Kline N., Lorentzos N., Mitsopoulos Y., Montanari A., Nonen D., Peresi E., Pernici B., Roddick J.F., Sarda N.L., Scalas M.R., Segev A., Snodgrass R.T., Soo M.D., Tansel A., Tiberio R., and Wiederhold G. A consensus glossary of temporal database concepts – February 1998 Version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, and S. Sripada (eds.). Springer Berlin, 1998.
9. Khatri V., Ram S., and Snodgrass R.T. Augmenting a conceptual model with geospatiotemporal annotations. IEEE Trans. Knowl. Data Eng., 16(11):1324–1338, 2004.
10. Khatri V., Vessey I., Ram S., and Ramesh V. Cognitive fit between conceptual schemas and internal problem representations: the

case of geospatio-temporal conceptual schema comprehension. IEEE Trans. Profession. Commun., 49(2):109–127, 2006.

11. Klopprogge M.R. TERM: an approach to include the time dimension in the entity relationship model. In Proc. 2nd Int. Conf. on Entity-Relationship Approach, 1981, pp. 473–508.
12. Moens M. and Steedman M. Temporal ontology and temporal reference. Comput. Linguist., 14(2):15–28, 1988.
13. Morris C.W. Foundations of the theory of signs. In Int. Encyclopedia of Unified Science, vol. 1, 2nd edn. University of Chicago Press, 1955.
14. Ram S. Intelligent database design using the unifying semantic model. Inform. Manage., 29(4):191–206, 1995.
15. Vessey I. Cognitive fit: a theory-based analysis of graphs vs. tables literature, Decision Sci., 22(2):219–240, 1991.
16. Zimanyi E., Parent C., Spaccapietra S., and Pirotte A. TERC+: a temporal conceptual model. In Proc. Int. Symp. Digital Media Information Base, 1997.

Temporal Constraints

Peter Revesz
University of Nebraska-Lincoln, Lincoln, NE, USA

Definition

Temporal Constraints describe relationships among variables that refer somehow to time. A set of temporal constraints can be stored in a temporal database, which is queried by temporal queries during problem solving. For example, a set of temporal constraints may form some requirements, all of which must be satisfied during some scheduling problem.

Most interesting temporal constraints derive from references to time in natural language. Such references typically compare two *time points*, two *sets of time points*, or two *time intervals*. The literature on temporal constraints and this entry focuses on the study of these types of comparative or *binary* constraints.

Historical Background

The seminal work on temporal intervals is by Allen [1]. Difference Bounded Matrices (see the Section on Scientific Fundamentals) were introduced by Dill [3]. A graph representation of difference constraints and efficient constraint satisfaction problem-based solutions for consistency of difference constraints were presented by Dechter et al. [2]. A graph representation of gap-order constraints and an efficient algebra on them is presented by Revesz [11]. A graph representation of set order constraints and algebra on them is described in [12]. Addition constraints are considered in [12].

Temporal Constraints. Table 1. Consistency for conjunctions of temporal constraints

Case	Temporal constraints	Integers	Rationals
1.	After, Before, Equal, Inequal, AoE, BoE	$O(v + e)$	$O(v + e)$
2.	Addition	$O(ve)$	$O(v + e)$
3.	Inequal, Difference	NP-complete	$O(v^3)$

The complexity of deciding the consistency of conjunctions of integer addition constraints in Table 1 is from [9].

Periodicity constraint within query languages are considered by Kabanza et al. [4] and Toman and Chomicki [15]. Constraint databases [12,8] were introduced by Kanellakis et al. [5] with a general framework for constraints that includes temporal constraints. Indefinite temporal constraint databases were introduced by Koubarakis [6]. Linear cardinality constraints on sets were considered by Kuncak et al. [7] and Revesz [13].

Deciding the consistency of conjunctions of rational (or real) difference and inequality constraints was proven tractable by Koubarakis, but the $O(v^3)$ complexity result in Table 1 is from Péron and Halbwachs [10]. The NP-completeness result in Table 1 follows from [14].

Foundations

Temporal constraints on time points express temporal relationships between two time points, which are called also *time instances*. More precisely, let x and y be integer or rational variables or constants representing time points, and let b be an integer constant. Then some common temporal constraints on time points include the following:

$$\begin{array}{ll}
\textit{After}: & x > y \\
\textit{Before}: & x < y \\
\textit{Equal}: & x = y \\
\textit{Inequal}: & x \neq y \\
\textit{After or Equal}\ \ (\textit{AoE}): & x \geq y \\
\textit{Before or Equal}\ \ (\textit{BoE}): & x \leq y \\
\textit{After by at least } b\ \ (\textit{Gap}-\textit{Order}): & \\
& x - y \geq b \ \text{ where } \ b \geq 0 \\
\textit{Difference}: & x - y \geq b \\
\textit{Potential}: & x - y \leq b \\
\textit{Addition}: & \pm x \pm y \geq b
\end{array}$$

In the above table, the first six constraints are called *qualitative* constraints, and the last four constraints are called *metric* constraints because they involve a measure b of time units. For example, "a copyright form needs to be filled out before publication" can be expressed as $t_{copyright} < t_{publication}$, where $t_{copyright}$ is the time point when the copyright form is filled out and $t_{publication}$ is the time point when the paper is printed. This constraint could be just one of the requirements in a complex scheduling problem, for example, the process of publishing in a book a collection of research papers. Another temporal constraint may express that "the publication must be at least 30 days after the time of submission," which can be expressed by the constraint $t_{publication} - t_{submission} \geq 30$.

Temporal constraints on sets of time points express temporal relationships between two sets of time points. The domain of a set X is usually assumed to be the finite and infinite subsets of the integers. Common temporal constraints between sets of time points include the following:

$$\begin{array}{ll} \textit{Equal}: & X = Y \\ \textit{Inequal}: & X \neq Y \\ \textit{Contains}\ \ (\textit{Set Order}): & X \subseteq Y \\ \textit{Disjoint}: & X \cap Y = \emptyset \\ \textit{Overlap with } b \textit{ elements}: & \\ & |X \cap Y| = b \end{array}$$

where X and Y are set variables or constants, $\emptyset$ is the empty set, b is an integer constant, and $||$ is the *cardinality* operator. For example, "The Database Systems class and the Geographic Information Systems class cannot be at the same time" can be expressed as $T_{Database} \cap T_{GIS} = \emptyset$, where $T_{Database}$ is the set of time points (measured in hour units) the Database System class meets, and T_{GIS} is the set of time points the Geographic Information Systems class meets.

Temporal constraints on time intervals express temporal relationships between two time intervals. These types of temporal constraints are known as *Allen's Relations* [Allen's Relations] because they were studied first by J. F. Allen [1].

Other types of temporal constraints: The various other types of temporal constraints can be grouped as follows:

- *n-ary temporal constraints.* These generalize the binary temporal constraints to n number of variables that refer to time. While *temporal databases* typically use binary constraints, *constraint databases* [5] use n-ary constraints [Constraint Databases], for example linear and polynomial constraints on n time point variables.
- *Temporal periodicity constraints.* Periodicity constraints [4,15] occur in natural language in phrases such as "every Monday." These constraints are discussed separately in [Temporal Periodicity Constraints].
- *Indefinite temporal constraints.* The nature of a temporal constraint can be two types: definite and indefinite. Definite constraints describe events precisely, while indefinite constraints describe events less precisely leaving several possibilities. For example, "Ed had fever between 1 P.M. and 9 P.M. but at no other times" is a definite constraint because it relays each time instance whether Ed had a fever or not. On the other hand, "Ed had fever for some time during 1 P.M. and 9 P.M." is an indefinite constraint because it allows the possibility that Ed had fever at 5 P.M. and at no other times, or another possibility that he had fever between 1 P.M. and 4 P.M. and at no other times. Hence it does not relay whether Ed had fever at 5 P.M. Conjunctions of temporal constraints can be represented in a number of ways. Consider a scheduling problem where one needs to schedule the events $e_1,\ldots,e_6$. Suppose that there are some scheduling requirements of the form "some (second) event occurs after another (first) event by at least b days." For example, each row of the following table, which is a temporal database relation, represents one such scheduling constraint.

Scheduling_Requirements

Second_Event	First_Event	After_By
0	e_1	5
0	e_2	2
e_1	e_3	−2
e_1	e_5	−9
e_2	e_1	−6
e_3	e_2	3
e_3	e_4	−3
e_4	e_3	−5
e_5	e_4	3
e_5	e_6	3
e_6	0	1

Many queries are easier to evaluate on some alternative representation of the above temporal database relation. Some alternative representations that may be used within a temporal database system are given below.

Conjunctions of Constraints

Let x_1,...,x_6 represent,respectively, the times when events e_1,...,e_6 occur. Then the *Scheduling_Requirements* relation can be represented also by the following conjunction of difference constraints:

$$
\begin{aligned}
&0 - x_1 \geq 5,\\
&0 - x_2 \geq 2,\\
&x_1 - x_3 \geq -2, x_1 - x_5 \geq -9, x_2 - x_1 \geq -6, x_3 -\\
&x_2 \geq 3, x_3 - x_4 \geq -3, x_4 - x_3 \geq -5, x_5 - x_4 \geq 3,\\
&x_5 - x_6 \geq 3,\\
&x_6 - 0 \geq 1
\end{aligned}
$$

Labeled Directed Graphs

In general, the graph contains $n + 1$ vertices representing all the n variables and 0. For each difference constraint of the form $x_i - x_j \geq b$, the graph contains also a directed edge from the vertex representing x_j to the vertex representing x_i. The directed edge is labeled by b.

Difference Bound Matrices

Conjunctions of difference constraints can be represented also by *difference bound matrices (DBMs)* of size $(n + 1) \times (n + 1)$, where n is the number of variables. For each difference constraint of the form $x_i - x_j \geq b$, the DBM contains the value b in its (j, i)th entry. The default value is $-\infty$. For example, above set of difference constraints can be represented by the following DBM:

	0	x_1	x_2	x_3	x_4	x_5	x_6
0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	1
x_1	5	$-\infty$	-6	$-\infty$	$-\infty$	$-\infty$	$-\infty$
x_2	2	$-\infty$	$-\infty$	3	$-\infty$	$-\infty$	$-\infty$
x_3	$-\infty$	-2	$-\infty$	$-\infty$	-5	$-\infty$	$-\infty$
x_4	$-\infty$	$-\infty$	$-\infty$	-3	$-\infty$	3	$-\infty$
x_5	$-\infty$	-9	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
x_6	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	3	$-\infty$

A major question about temporal constraint formulas is whether they are *consistent* or *satisfiable.* A formula is consistent or satisfiable if and only if it has at least one substitution for the variables that makes the formula true. Otherwise, it is called *inconsistent* or *unsatisfiable.* For example, if the conjunction of difference constraints that describe the *Scheduling_Requirements* table is inconsistent, then there is no schedule of the events e_1,...,e_6 such that all the requirements are satisfied.

Table 1 summarizes some computational complexity results, in terms of v the number of vertices and e the number of edges in the graph representation.

Most complexity results translate deciding the consistency to classical problems on graphs with efficient and well-known solutions. Decher et al. [2] provides a translation to constraint satisfaction problems, which use efficient search heuristics for large sets of constraints. Many operations on DBMs can be defined. These operators include *conjunction* or *merge* of DBMs, *variable elimination* or *projection* of a variable from a DBM, testing *implication* of a DBM by a disjunction of DBMs, and *transitive closure* of a DBM [12].

Temporal Constraints on Time Intervals.

In general, deciding the consistency of a conjunction of temporal constraints on intervals is NP-complete. Many computational complexity results for temporal constraints on time intervals follow from the complexity results for temporal constraint on time points. In particular, any conjunction of pointisable temporal constraints on time intervals can be translated to a conjunction of temporal constraints on time points. After the translation, the consistency can be tested as before.

Key Applications

Temporal constraints are used in scheduling, planning, and temporal database querying [Temporal Databases Queries]. Temporal database queries usually take the form of SQL or Datalog combined with temporal constraints. Examples include Datalog with gap-order constraints [11] and Datalog with periodicity constraints [15].

Future Directions

There are still many open problems on the use of temporal constraints in temporal database query languages. An important problem is finding efficient indexing methods for conjunctions of temporal constraints. The combination of temporal constraints with spatial constraints is an interesting area within spatiotemporal databases [Spatiotemporal Databases] and constraint databases [12].

Cross-references

- ▶ Database Query Languages
- ▶ Indexing
- ▶ Temporal Dependencies
- ▶ Temporal Integrity Constraints
- ▶ Temporal Periodicity

Recommended Reading

1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832–843, 1983.
2. Dechter R., Meiri I., and Pearl J. Temporal constraint networks. Artif. Intell., 49(1-3):61–95, 1991.
3. Dill D.L. Timing assumptions and verification of finite-state concurrent systems. In Proc. Automatic Verification Methods for Finite State Systems, 1989, pp. 197–212.
4. Kabanza F., Stevenne J.-M., and Wolper P. Handling infinite temporal data. J. Comput. Syst. Sci., 51(1):1–25, 1995.
5. Kanellakis P.C., Kuper G.M., and Revesz P. Constraint query languages. J. Comput. Syst. Sci., 51(1):26–52, 1995.
6. Koubarakis M. The complexity of query evaluation in indefinite temporal constraint databases. Theor. Comput. Sci., 171(1-2):25–60, 1997.
7. Kuncak V., Nguyen H.H., and Rinard M.C. An algorithm for deciding BAPA: Boolean Algebra with Presburger Arithmetic. In Proc. 20th Int. Conf. on Automated Deduction, 2005, pp. 260–277.
8. Kuper G.M., Libkin L., and Paredaens J. (eds.). Constraint Databases. Springer, Berlin Heidelberg New York, 2000.
9. Lahiri S.K. and Musuvathi M. An efficient decision procedure for UTVPI constraints. In Proc. 5th Int. Workshop on Frontiers of Combining Systems, 2005, pp. 168–183.
10. Péron M. and Halbwachs N. An abstract domain extending difference-bound matrices with disequality constraints. In Proc. 8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation, 2007, pp. 268–282.
11. Revesz P. A closed-form evaluation for Datalog queries with integer (gap)-order constraints. Theor. Comput. Sci., 116 (1):117–49, 1993.
12. Revesz P. Introduction to Constraint Databases. Springer, Berlin Heidelberg New York, 2002.
13. Revesz P. Quantifier-elimination for the first-order theory of Boolean algebras with linear cardinality constraints. In Proc. 8th Conf. on Advances in Databases and Information Systems, 2004, pp. 1–21.
14. Rosenkrantz D.J. and Hunt H.B. Processing conjunctive predicates and queries. In Proc. 6th Int. Conf. on Very Data Bases, 1980, pp. 64–72.
15. Toman D. and Chomicki J. Datalog with integer periodicity constraints. J. Logic Program., 35(3):263–290, 1998.

Temporal Data Mining

Nikos Mamoulis
University of Hong Kong, Hong Kong, China

Synonyms

Time series data mining; Sequence data mining; Temporal association mining

Definition

Temporal data mining refers to the extraction of implicit, non-trivial, and potentially useful abstract information from large collections of temporal data. Temporal data are sequences of a primary data type, most commonly numerical or categorical values and sometimes multivariate or composite information. Examples of temporal data are regular time series (e.g., stock ticks, EEG), event sequences (e.g., sensor readings, packet traces, medical records, weblog data), and temporal databases (e.g., relations with time-stamped tuples, databases with versioning). The common factor of all these sequence types is the total ordering of their elements. They differ on the type of primary information, the regularity of the elements in the sequence, and on whether there is explicit temporal information associated to each element (e.g., timestamps). There are several mining tasks that can be applied on temporal data, most of which directly extend from the corresponding mining tasks on general data types. These tasks include classification and regression (i.e., generation of predictive data models), clustering (i.e., generation of descriptive data models), temporal association analysis between events (i.e., causality relationships), and extraction of temporal patterns (local descriptive models for temporal data).

Historical Background

Analysis of time series data has been an old problem in statistics [4], the main application being forecasting for different applications (stock market, weather, etc.) Classic statistical models for this purpose include autoregression and hidden Markov models. The term temporal data mining came along as early as the birth of data mining in the beginning of the 1990s. Soon after association rules mining in large databases [1] has been established as a core research problem, several researchers became interested in association analysis in long sequences and large temporal databases (see [12] for a survey). One big challenge in temporal data mining is the large volume of the data, which make traditional autoregression analysis techniques inapplicable. Another challenge is the nature of the data which is not limited to numerical-valued time series, but includes sequences of discrete, categorical, and composite values (e.g., sets). This introduces new, interesting types of patterns, like causality relationships between events in time, partial periodic patterns, and calendric patterns.

Foundations

Classic data mining tasks, like classification, clustering, and association analysis can naturally be applied on large collections of temporal data. Special to temporal databases, are the extraction of patterns that are frequent during specific temporal intervals and the identification of temporal relationships between values or events in large sequences. In the following, the above problems are discussed in more detail.

Classification and Clustering

Classification of time series is often performed by nearest neighbor (NN) classifiers [13]. Given a time series $\vec{s}$ of unknown label and a database $\mathcal{D}$ of labeled samples, such a classifier (i) searches in $\mathcal{D}$ for the k most similar time series to $\vec{s}$ and (ii) gives $\vec{s}$ the most popular label in the set of k returned time series. This process involves two challenges: definition of an appropriate *similarity* function to be used by the NN classifier and scalability of classification. The dissimilarity (distance) between two time series is typically quantified by their Euclidean distance or the dynamic time warping (DTW) distance. Like classification, clustering of time series can be performed by applying an off-the-shelf clustering algorithm [7] (e.g., k-means), after defining an appropriate distance (i.e., dissimilarity) function.

For sequences of categorical data, Hidden Markov Models (HMM) can be used to capture the behavior of the data. HMM can be used for classification as follows. For each class label, a probabilistic state transition model that captures the probabilities of seeing one symbol (state) after the current one can be built. Then a sequence is given the label determined by the HMM that describes its behavior best.

Prediction

For continuous-valued sequences, like time series, regression is an alternative to classification. Regression does not use a fixed set of class labels to describe each sequence, but models sequences as functions, which are more appropriate for predicting the values in the future. Autoregression is a special type of regression, where future values are predicted as a linear combination of recent previous values, assuming that the series exhibits a periodic behavior. Formally, an autoregressive model of order p for a time series $\vec{s} = \{s_1, s_2, \ldots\}$ can be described as follows:

$$s_i = e_i + \sum_{j=1}^{p} \phi_j s_{i-j},$$

where $\phi_j (1 \leq j \leq p)$ are the parameters of autoregression, and e_i is an error term. The error terms are assumed to be independent identically-distributed random variables (i.i.d.) that follow a zero-mean normal distribution. The main trend of a time series is commonly described by a *moving average* function, which is a smoothed abstraction of the same length. Formally, the moving average of order q for a time series $\vec{s} = \{s_1, s_2, \ldots\}$ can be described as follows:

$$MA(\vec{s})_i = e_i + \sum_{j=1}^{q} \psi_j e_{i-j},$$

where $\psi_j (1 \leq j \leq q)$ are the parameters of the model. By combining the above two concepts, a time series $\vec{s}$ can be described by an autoregressive moving average (ARMA) model:

$$s_i = e_i + \sum_{j=1}^{p} \phi_j s_{i-j} + \sum_{j=1}^{q} \psi_j e_{i-j},$$

Autoregressive integrated moving average (ARIMA) is a more generalized model, obtained by integrating an ARMA model. In long time series, periodic behaviors tend to be local, so a common practice is to segment the series into pieces with constant behavior and generate an autoregression model at each piece.

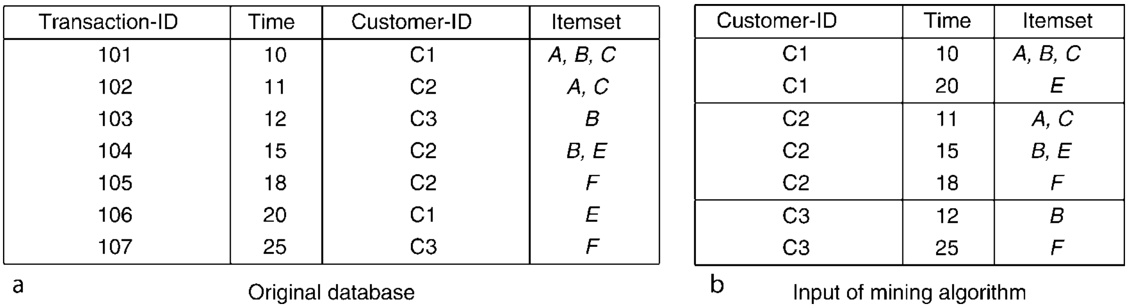
Association Analysis and Extraction of Sequence Patterns

Agrawal and Srikant [3] proposed one of the first methods for association analysis in timestamped transactional databases. A transactional database records timestamped customer transactions (e.g., sets of books bought at a time) in a store (e.g., bookstore) and the objective of the analysis is to discover causality relationships between sets of items bought by customers. An example of such a *sequential* pattern (taken from the paper) is "5% of customers bought 'Foundation,' then 'Foundation and Empire,' and then 'Second Foundation,' " which can be represented by {(Foundation),(Foundation and Empire),(Second Foundation)}. In general, sequential patterns are total orders of *sets of items* bought in the same transaction. For example, {(Foundation,Life),(Second Foundation)} models the buying of "Foundation" and "Life" at a single transaction followed by "Second Foundation" at another transaction. The patterns can be extracted by dividing the database that records the transaction history of the bookstore into groups, one per customer, and then treat each group as an ordered sequence. For example, the transactional database shown in Fig. 1a is transformed to the grouped table of Fig. 1b.

The algorithm for extracting sequential patterns from the transformed database is reminiscent to the Apriori algorithm for frequent itemsets in transactional databases [2]. It takes as input a minimum support threshold *min-sup* and operates in multiple passes. In the first pass, the items that have been bought by at least *min-sup* of the customers are put to a frequent items set L_1. Then, orderings of pairs of items in L_1 form a candidate set C_2 of level-2 sequential patterns, the supports of which are counted during the second pass of the transformed database and the frequent ones form L_2. A sequence adds to the support of a pattern if the pattern is contained in it. For example, the sequence $\{(A, C),(B, E),(F)\}$ of customer C2 in Fig. 1 adds to the support of pattern $\{(A),(F)\}$. In general, after L_k has been formed, the algorithm generates and counts candidate patterns of $k + 1$ items. These candidates are generated by joining pairs (s_1, s_2) of frequent k-sequences, such that the subsequence obtained by dropping the first item of s_1 is identical to the one obtained by dropping the last item of s_2. For example, $\{(A, B),(C)\}$ and $\{(B), (C, D)\}$ generate $\{(A, B), (C, D)\}$. Candidates resulting from the join phase are pruned if they have a subsequence that is not frequent.

Agrawal and Srikant also considered adding constraints when counting the supports of sequential patterns. For example, if "Foundation and Empire" is bought 3 years after "Foundation," these two books may be considered unrelated. In addition, they considered relaxing the rigid definition of a transaction by unifying transactions of the same customer that took place close in time. For example, if a customer buys a new book minutes after her previous transaction, this book should be included in the previous transaction (i.e., the customer may have forgotten to include it in her basket before). Parallel to Agrawal and Srikant, Mannila et al. [9] studied the extraction of frequent causality patterns (called episodes) in long event sequences. The main differences of this work are (i) the input is a single very long sequence of events (e.g., a stream of sensor indications), (ii) patterns are instantiated by temporal sliding windows along this stream of events, and (iii) patterns can contain sequential modules (e.g., *A* after *B*) or parallel modules (e.g., *A* and *B* in any order). An example of such an episode is "*C* first, then *A* and *B* in any order, then *D*". To compute frequent episodes Mannila et al. [9] proposed adaptations of the classic Apriori technique [2]. A more efficient technique for mining sequential patterns was later proposed by Zaki [14].

Han et al. [6] studied the problem of mining partial periodic patterns in long event sequences. In many



Transaction-ID	Time	Customer-ID	Itemset
101	10	C1	*A, B, C*
102	11	C2	*A, C*
103	12	C3	*B*
104	15	C2	*B, E*
105	18	C2	*F*
106	20	C1	*E*
107	25	C3	*F*

a Original database

Customer-ID	Time	Itemset
C1	10	*A, B, C*
C1	20	*E*
C2	11	*A, C*
C2	15	*B, E*
C2	18	*F*
C3	12	*B*
C3	25	*F*

b Input of mining algorithm

Temporal Data Mining. Figure 1. Transformation of a timestamped transactional database.

applications, the associations between events follow a periodic behavior. For instance, the actions of people follow habitual patterns on a daily basis (i.e., "wake-up," then "have breakfast," then "go to work," etc.). Given a long event sequence (e.g., the actions of a person over a year) and a time period (e.g., 24 h), the objective is to identify patterns of events that have high support over all the periodic time intervals (e.g., days). For this purpose, all subsequences corresponding to the activities of each periodic interval can be extracted from the long sequence, and a sequential pattern mining algorithm [3] can be applied. Based on this idea, an efficient technique for periodic pattern mining, which is facilitated by the use of a sophisticated prefix tree data structure, was proposed by Han et al. [6]. In some applications, the time period every when the patterns are repeated is unknown and has to be discovered from the data. Towards this direction, Cao et al. [5] present a data structure that automatically identifies the periodicity and discovers the patterns at only a small number of passes over the data sequence.

Temporal, Cyclic, and Calendric Association Rules

An association rule in a transactional database may not be strong (according to specific support and confidence thresholds) in the whole database, but only when considering the transactions in a specified time interval (e.g., during the winter of 2005). An association rule bound to a time interval, where it is strong, is termed temporal association rule [12]. Identification of such a rule can be performed by starting from short time intervals and progressively extending them to the maximum possible length where the rule remains strong.

Özden et al. [10] noticed that association rules in transactional databases (e.g., people who buy turkey they also buy pumpkins) may hold only in particular temporal intervals (e.g., during the last week of November every year). These are termed *cyclic* association rules, because they are valid periodically, at a specific subinterval of a cycle (e.g., year). Such rules can be discovered by identifying the periodic intervals of fixed granularity (e.g., week of the year), which support the associations.

Cyclic rules are assumed to be supported at *exact* intervals (e.g., the last day of January), and at *every* cycle (e.g., every year). In practice, a rule may be supported with some mismatch threshold (e.g., the last weekday of January) and only at the majority of cycles (e.g., 80% of the cycles). Accordingly, the "cyclic" rule concept was extended by Ramaswamy et al. [11] to the more flexible *calendric* association rule. A calendar is defined by a set of time intervals (e.g., the last 3 days of January, every year). For a calendric rule to be strong, it should have enough support and confidence in at least *min-sup*% of the time units included in the calendar. An algebra for defining calendars and a method for discovering calendric association rules referring to them can be found in Ref. [11].

Li et al. [8] proposed a more generalized framework for calendric association rules. Instead of searching based on a predetermined calendar, they automatically identify the rules and their supporting calendars, taken from a hierarchy of calendar concepts. The hierarchy is expressed by a relation of temporal generalizations of varying granularity, e.g., $R(year, month, day)$. A possible calendric pattern is expressed by setting to each attribute, either a specific value of its domain, or a wildcard value "$\ast$." For example, pattern ($\ast$, *Jan*, 30) means the 30th of January each year, while (2005, $\ast$, 30) means the the 30th day of each month in year 2005. By defining containment relationships between such patterns (e.g., ($\ast$, $\ast$, 30) contains all the intervals of (2005, $\ast$, 30)) and observing that itemset supports for them can be computed constructively (e.g., the support of an itemset in ($\ast$, $\ast$, 30) can be computed using its support in all (y, $\ast$, 30) for any year y), Li et al. [8] systematically explore the space of all calendric patterns using the Apriori principle to prune space (e.g., an itemset is not frequent in ($\ast$, $\ast$, 30) if it is infrequent in all (y, $\ast$, 30) for every year y).

Key Applications

Weather Forecasting

Temporal causality relationships between events can assist the prediction of weather phenomena. In fact, such patterns have been used for this purpose since the ancient years (e.g., "if swallows fly low, it is going to rain soon").

Market Basket Analysis

Extension of classic association analysis to consider temporal information finds application in market analysis. Examples include, temporal relationships between products that are purchased within the same period by customers ("5% of customers bought 'Foundation,' then 'Foundation and Empire' ") and calendric association rules (e.g., turkey is bought together with pumpkin during the last week of November, every year).

Stock Market Prediction

Time-series classification and regression is often used by financial analysts to predict the future behavior of stocks. The structure of the time series can be compared with external factors (such as pieces of news) to derive more complex associations that result in better accuracy in prediction.

Web Data Mining

The World Wide Web can be viewed as a huge graph where nodes correspond to web pages (or web sites) and edges correspond to links between them. Users navigate through the web defining sequences of page visits, which are tracked in weblogs. By analyzing these sequences one can identify frequent sequential patterns between web pages or even classify users based on their behavior (sequences of sites they visit and sequences of data they download).

Cross-references

▶ Association rules

▶ Spatial and Spatio-Temporal Data Models and Languages

▶ Temporal Periodicity

▶ Time Series Query

Recommended Reading

1. Agrawal R., Imielinski T., and Swami A.N. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 207–216.
2. Agrawal R. and Srikant R. Fast algorithms for mining association rules in large databases. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
3. Agrawal R. and Srikant R. Mining sequential patterns. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 3–14.
4. Box G.E.P. and Jenkins G. Time Series Analysis, Forecasting and Control. Holden-Day, 1990.
5. Cao H., Cheung D.W., and Mamoulis N. Discovering partial periodic patterns in discrete data sequences. In Advances in Knowledge Discovery and Data Mining, 8th Pacific-Asia Conf., 2004, pp. 653–658.
6. Han J., Dong G., and Yin Y. Efficient mining of partial periodic patterns in time series database. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 106–115.
7. Han J. and Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann, 2000.
8. Li Y., Ning P., Wang X.S., and Jajodia S. Discovering calendar-based temporal association rules. Data Knowl. Eng., 44(2):193–218, 2003.
9. Mannila H., Toivonen H., and Verkamo A.I. Discovery of frequent episodes in event sequences. Data Min. Knowl. Discov., 1(3):259–289, 1997.
10. Özden B., Ramaswamy S., and Silberschatz A. Cyclic association rules. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 412–421.
11. Ramaswamy S., Mahajan S., and Silberschatz A. On the discovery of interesting patterns in association rules. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 368–379.
12. Roddick J.F. and Spiliopoulou M. A survey of temporal knowledge discovery paradigms and methods. IEEE Trans. Knowl. Data Eng., 14(4):750–767, 2002.
13. Wei L. and Keogh E.J. Semi-supervised time series classification. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006, pp. 748–753.
14. Zaki M.J. Spade: an efficient algorithm for mining frequent sequences. Mach. Learn., 42(1/2):31–60, 2001.

Temporal Data Models

Christian S. Jensen[1], Richard T. Snodgrass[2]

[1]Aalborg University, Aalborg, Denmark

[2]University of Arizona, Tucson, AZ, USA

Synonyms

Valid-time data model; Transaction-time data model; Bitemporal data model; Historical data model

Definition

A "data model" consists of two components, namely a set of objects and a language for querying those objects [4]. In a temporal data model the objects vary over time, and the operations in some sense "know" about time. Focus has been on the design of data models where the time references capture valid time, or transaction time, or a combination of both (for bitemporal data).

Historical Background

Almost all real-world databases contain time-referenced data. Few interesting databases are entirely stagnant, and when the modeled reality changes, the database must be updated. Usually at least the start time of currently valid data are captured, though most databases also retain previous data.

Two decades of research into temporal databases have unequivocally shown that a *time-referencing table*, containing certain kinds of time-valued columns that capture one or more temporal aspects of data recorded in other columns, is completely different from this table, without the time-valued columns. Effectively designing, querying, and modifying time-referencing tables requires a different set of approaches and techniques. It is possible to handle such data within standard data models, generally at the expense of high data redundancy, awkward modeling, and unfriendly query languages. An alternative is a data model, probably an extension of an extant, non-temporal data model, that explicitly incorporates time, making it easier to express queries, modifications, and integrity constraints.

As an example, consider a primary key of the following relation, which records the current positions of employees, identified by their social security numbers: `EMP(EmpID, POSITION)`. The primary key is obviously `EmpID`. Now add `STARTTIME` and `STOPTIME` attributes. While a primary key of `(EmpID, STARTTIME)` seems to work, such a primary key will not prevent overlapping periods, which would allow an employee to have two positions at a point of time, which is problematic. Stating the primary key constraint properly requires a complex assertion containing a dozen lines of code with multiple sub-queries [3]. Referential integrity is even more challenging.

The last two decades have seen the introduction of a great many temporal data models, not just in the context of relational data, as in the above example, but also with object-oriented, logic-based, and semistructured data.

Foundations

Levels of Abstraction

Temporal data models exist at three abstraction levels: the *conceptual level*, in which the data models are generally extensions of the Entity-Relationship Model, the *logical level*, in which the data models are generally extensions of the relational data model or of an object-oriented data model, and, infrequently, the *physical level*, in which the data model details how the data are to be stored. In terms of prevalence, models at the logical level are by far the most numerous. However, it has been shown that several of the models that were originally proposed as logical data models are actually equivalent to the BCDM logical model, and should more properly be viewed as physical data models [2]. This entry is restricted to logical models, focusing on the objects that are subject to querying rather than the query languages. First examined is the association of time with data, as this is at the core of temporal data management.

Temporal Aspects of Data

A database models and records information about a part of reality, termed the modeled reality. Aspects of the modeled reality are represented in the database by a variety of structures, termed database entities. In general, times are associated with database entities. The term "fact" is used for any (logical) statement that can meaningfully be assigned a truth value, i.e., true or false.

The facts recorded by database entities are of fundamental interest, and a fundamental temporal aspect may be associated with these: the valid time of a fact is the times when the fact is true in the modeled reality. While all facts have a valid time by definition, the valid time of a fact may not necessarily be recorded in the database. For example, the valid time may not be known, or recording it may not be relevant. Valid time may be used for the capture of more application-specific temporal aspects. Briefly, an application-specific aspect of a fact may be captured as the valid time of another, related fact.

Next, the transaction time of a database entity is the time when the entity is current in the database. Like valid time, this is an important temporal aspect. Transaction time is the basis for supporting accountability and "traceability" requirements. Note that transaction time, unlike valid time, may be associated with any database entity, not only with facts. As for valid time, the transaction-time aspect of a database entity may or may not be captured in the database. The transaction-time aspect of a database entity has a duration: from insertion to deletion. As a consequence of the semantics of transaction time, deleting an entity does not physically remove the entity from the database; rather, the entity remains in the database, but ceases to be part of the database's current state.

Observe that the transaction time of a database fact, say *f*, is the valid time of the related fact, "*f* is current in the database." This would indicate that supporting transaction time as a separate aspect is redundant. However, both valid and transaction time are aspects of the content of all databases, and recording both of these is essential in many applications. In addition, transaction time, due to its special semantics, is particularly well-behaved and may be supplied automatically by the DBMS. Specifically, the transaction times of a fact stored in the database is bounded by the time the database was created at one end of the time line and by the current time at the other end.

The above discussion suggests why temporal data models generally offer built-in support for one or both of valid and transaction time.

Representation of Time

The valid and transaction time values of database entities are drawn from some appropriate time domain. There is no single answer to how to perceive time in reality and how to represent time in a database, and different time domains may be distinguished with respect to several orthogonal characteristics. First, the time domain may or may not stretch infinitely into the past and future. Second, time may be perceived as discrete, dense, or continuous. Some feel that time is really continuous; others contend that time is discrete and that continuity is just a convenient abstraction that makes it easier to reason mathematically about certain discrete phenomena. In databases, a finite and discrete time domain is typically assumed, e.g., in the SQL standards. Third, a variety of different structures have been imposed on time. Most often, time is assumed to be totally ordered.

Much research has been conducted on the semantics and representation of time, from quite theoretical topics, such as temporal logic and infinite periodic time sequences, to more applied questions such as how to represent time values in minimal space. Substantial research has been conducted that concerns the use of different time granularities and calendars in general, as well as the issues surrounding the support for indeterminate time values. Also, there is a significant body of research on time data types, e.g., time instants, time intervals (or "periods"), and temporal elements.

Data Model Objects

The management of temporal aspects has been achieved by building time into the data model objects. Here, the relational model is assumed, with a focus on valid time. One approach is to timestamp tuples with time instants, or points. Then a fact is represented by one tuple for each time point during which the fact is valid. An example instance for the EMP relation example is shown in Fig. 1.

A distinguishing feature of this approach is that (syntactically) different relations have different information content. Next, timestamps are atomic values that can be easily compared. Assuming a totally ordered time domain, the standard set of comparison predicates, =, ≠, <, >, ≤, and ≥, is sufficient to conveniently compare timestamps. The conceptual simplicity of time points comes at a cost, though. The model offers little support for capturing, e.g., that employee 2 was assigned to Sales during two contiguous periods [4,5] and [6,7], instead of during a single contiguous period [4,7].

It is important to note that the point model is not meant for physical representation, as for all but the most trivial time domains, the space needed when using the point model is prohibitive. The combination of conceptual simplicity and low computational complexity has made the point model popular for theoretical studies.

Another type of data model uses time periods as timestamps. This type of model associates each fact with a period that captures the valid time of the fact. Multiple tuples are needed if a fact is valid over disjoint periods. Figure 2 illustrates the approach.

The notion of snapshot equivalence, which reflects a point-based view of data, establishes a correspondence between the point-based and period-based models. Imagine that the last two tuples in the relation in Fig. 2 were replaced with the single tuple (2, Sales, [4,7]) to obtain a new relation. The resulting two relations are different, but snapshot equivalent.

EmpID	Position	T
1	Sales	3
1	Sales	4
1	Engineering	5
1	Engineering	6
2	Sales	4
2	Sales	5
2	Sales	6
2	Sales	7

Temporal Data Models. Figure 1. Point model.

Specifically, the new relation is a coalesced version of the original relation.

In some data models, the relations are taken to contain the exact same information. These models adopt a point-based view and are only period-based in the weak sense that they use time periods as convenient representations of (convex) sets of time points. It then also makes sense for such models to require that their relation instances be *coalesced*. This requirement ensures that relation instances that are syntactically different are also semantically different, and vice versa. In such models, the relation in Fig. 2 is not allowed.

In an inherently period-based model, periods carry meaning beyond denoting a set of points. In some situations, it may make a difference whether an employee holds a position for two short, but consecutive time periods versus for one long time period. Period-based models do not enforce coalescing and capture this distinction naturally.

Next, a frequently mentioned shortcoming of periods is that they are not closed under all set operations, e.g., subtraction. This has led to the proposal that temporal elements be used as timestamps instead. These are finite unions of periods.

With *temporal elements*, the same two semantics as for periods are possible, although models that use temporal elements seem to prefer the point-based semantics. Figure 3a and Figure 3b uses temporal elements to capture the example assuming the period-based semantics and point-based semantics, respectively. (As [4,5] ∪ [6,7] = [4,7], this latter period could have been used instead of [4,5] ∪ [6,7].)

EmpID	Position	T
1	Sales	[3, 4]
1	Engineering	[5, 6]
2	Sales	[4, 5]
2	Sales	[6, 7]

Temporal Data Models. Figure 2. Period (or interval) model.

Note that the instance in Fig. 3b exemplifies the instances used by the point-based bitemporal conceptual data model (BCDM) when restricted to valid time. This model has been used for TSQL2. The BCDM timestamps facts with values that are sets of time points. This is equivalent to temporal elements because the BCDM adopts a discrete and bounded time domain.

Because value-equivalent tuples are not allowed (this corresponds to the enforcement of coalesced relations as discussed earlier), the full history of a fact is contained in exactly one tuple, and one tuple contains the full history of exactly one fact. In addition, relation instances that are syntactically different have different information content, and vice versa. This design decision reflects the point-based underpinnings of the BCDM.

With temporal elements, the full history of a fact is contained in a single tuple, but the information in a relation that pertains to some real-world object may still be spread across several tuples. To capture all information about a real-world object in a single tuple, attribute value timestamping has been introduced. This is illustrated in Fig. 4, which displays the sample instance using a typical attribute-value timestamped data model.

The instance records information about employees and thus holds one tuple for each employee, with a tuple containing all information about an employee. An obvious consequence is that the information about a position cannot be contained in a single tuple. Another observation is that a single tuple may record multiple facts. In the example, the first tuple records two facts: the position type for employee 1 for the two positions, Sales and Engineering.

It should also be noted that different groupings into tuples are possible for this attribute-value timestamping model. Figure 5 groups the relation instance in Fig. 4 on the `POSITION` attribute, indicating that it is now the positions, not the employees, that are the objects in focus.

EmpIS	Position	T
1	Sales	[3, 4]
1	Engineering	[5, 6]
2	Sales	[4, 5]
2	Sales	[6, 7]

a Period view

EmpID	Position	T
1	Sales	[3, 4]
1	Engineering	[5, 6]
2	Sales	[4, 5] U [6, 7]

b Point view

Temporal Data Models. Figure 3. Temporal element model.

EmpID	Position
[3, 6] 1	[3, 4] Sales [5, 6] Engineering
[4, 7] 2	[4, 7] Sales

Temporal Data Models. Figure 4. Attribute-value timestamped model.

EmpID	Position
[3, 4] 1 [4, 7] 2	[3, 7] Sales
[5, 6] 1	[5, 6] Engineering

Temporal Data Models. Figure 5. Attribute-value timestamped model, grouped on POSITION.

Data models that timestamp attribute values may be temporally grouped. In a temporally grouped model, all aspects of a real-world object may be captured by a single tuple [1].

At first sight, that attribute-value timestamped model given above is temporally grouped. However, with a temporally grouped model, a real-world object is allowed to change the value for its key attribute. In the example, this means that the instance in Fig. 6 should be possible. Now observe that when grouping this instance on EmpID or POSITION, or both, it is not possible to get back to the original instance. Thus, temporally grouped tuples are not well supported. Clifford et al. [1] explore the notion of temporally grouped models in considerable depth.

Query Languages

Having covered the objects that are subject to querying, the last major aspect of a logical data model is the query language associated with the objects. Such languages come in several variants.

Some are intended for internal use inside a temporally enhanced database management system. These are typically algebraic query languages. However, algebraic languages have also been invented for more theoretical purposes. For example, an algebra may be used for defining the semantics of a temporal SQL extension. A key point is that an algebra is much simpler than is such an extension. Little is needed in terms of language design; only a formal definition of each operator is needed.

EmpID	Position
[3, 5] 1 [6, 6] 3	[3, 4] Engineering [5, 6]
[4, 7] 2	[4, 7] Sales

Temporal Data Models. Figure 6. Attribute-value timestamped model, temporally grouped.

Other query languages are targeted at application programmers and are thus typically intended to replace SQL. The vast majority of these are SQL extensions. Finally, languages have been proposed for the purpose of conducting theoretical studies, e.g., of expressive power.

Key Applications

Virtually all databases contain temporal information, and so virtually all database applications would benefit from the availability of data models that provide natural support for such time-varying information.

Future Directions

Rather than come up with a new temporal data model, it now seems better to extend, in an upward-consistent manner, existing non-temporal models to accommodate time-varying data.

Cross-references

- ▶ Period-Stamped Temporal Models
- ▶ Point-Stamped Temporal Models
- ▶ Probabilistic Temporal Databases
- ▶ Supporting Transaction Time Databases
- ▶ Temporal Access Control
- ▶ Temporal Compatibility
- ▶ Temporal Concepts in Philosophy
- ▶ Temporal Conceptual Models
- ▶ Temporal Constraints
- ▶ Temporal Database
- ▶ Temporal Indeterminacy
- ▶ Temporal Logical Models
- ▶ Temporal Object-Oriented Databases
- ▶ Temporal Query Languages
- ▶ Temporal XML
- ▶ Transaction Time
- ▶ Valid Time

Recommended Reading

1. Clifford J., Croker A., and Tuzhilin A. On completeness of historical relational query languages. ACM Trans. Database Syst., 19(1):64–16, March 1994.
2. Jensen C.S., Soo M.D., and Snodgrass R.T. Unifying temporal data models via a conceptual model. Inf. Syst., 19(7):513–547, December 1994.
3. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL, Morgan Kaufmann, San Francisco, CA, July 1999.
4. Tsichritzis D.C. and Lochovsky F.H. Data Models. Software Series. Prentice-Hall, 1982.

Temporal Data Warehousing

► Data Warehouse Maintenance, Evolution and Versioning

Temporal Database

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Historical database; Time-oriented database

Definition

A temporal database is a collection of time-referenced data. In such a database, the time references capture some temporal aspect of the data; put differently, the data are timestamped. Two temporal aspects are prevalent. The time references may capture either the past and current states of the database, yielding a transaction-time database; they may capture states of the reality being modeled by the data, yielding a valid-time database; or they may capture both aspects of the data, yielding a bitemporal database.

Historical Background

"Time" is a fundamental concept that pervades all aspects of daily life. As one indicator, a recent study by Oxford University Press found that the word "time" is the most commonly used noun in the English language. The nouns "year" and "day" rank third and fifth in the study.

Moving to a database context, the capture and representation of time-varying information go back literally thousands of years. The Egyptians and Syrians carved records into stone walls and pyramids of inventories of grain over many years. But it has been only in the last few decades that the advent of increasingly inexpensive and voluminous digital storage has enabled the computerized management of increasingly large volumes of time-referenced data.

Temporal databases have been the subject of intense study since the early 1980s. A series of bibliographies on temporal databases enumerates the thousands of refereed papers that have been written on the subject (the series started with Boulour's paper [3]; the most recent in the series is by Wu et al. [17]). There have also been more specialized bibliographies [2,9,13].

The first temporal database conference was held in Sofie Antipolis, France in 1987 [1]. Two workshops, the Temporal Database Workshop in Arlington, Texas in 1993 and in Zürich, Switzerland in 1995 [8], were held subsequently. The premier conferences on the topic are the International Symposium on Temporal Representation and Reasoning (*TIME*) (held annually since 1994), the International Symposium on Spatial and Temporal Databases (*SSTD*) (held biannually), and the Spatio-Temporal Database Management (*STDBM*) series (three up through 2006).

A seminal book collecting important results to date in this field appeared in 1993 [16]. Several surveys have been written on the topic [4–7, 10–12, 14, 15]. Temporal databases were covered in detail in an advanced database textbook [18].

Foundations

Time impacts all aspects of database technology, including database design (at the conceptual, logical, and physical levels) and the technologies utilized by a database management system, such as query and modification languages, indexing techniques and data structures query optimization and query evaluation techniques, and transaction processing.

The entries related to temporal databases go into more detail about these aspects. The following provides an organization on those entries (which are indicated in *italics*).

General Concepts

- Philosophers have thought hard about time (*temporal concepts in philosophy*).
- Two general temporal aspects of data attract special attention: *valid time* and *transaction time*.
- The *time domain* can be differentiated along several aspects: its structure, e.g., linear or branching; discrete versus continuous; bounded or infinite.
- Just as multiple versions of data may be stored, independently, the schemas can be versioned (*schema versioning*).
- The concept of "now" is important (*now in temporal databases*).

Temporal Data Models

- *Temporal conceptual models* generally extend an existing conceptual model, such as one of the variants of the Entity-Relationship model.
- *Temporal logical models* generally extend the relational model or an object-oriented model (*temporal object-oriented models*) or XML (*temporal XML*).
- Data can be associated with time in several ways: with time points (*point-stamped temporal models*) or time periods (*period-stamped temporal models*); these may capture valid and/or transaction time; and the associations of the data with the time values may carry probabilities (*temporal probabilistic models*).
- The time values associated with the data are characterized by their *temporal granularity*, and they may possess *temporal indeterminacy* and *temporal periodicity*.
- Data models incorporate *temporal constraints*, *temporal integrity constraints*, and *temporal dependencies*.

Temporal Query Languages

- Most *temporal query languages* are based on the relational algebra or calculus. Not surprisingly, much attention has been given to the design of user-level temporal query languages, notably *SQL-based temporal query languages*. For such languages, different notions of *temporal compatibility* have been an important design consideration.
- *Qualitative temporal reasoning* and *temporal logic in database query languages* provide expressive query facilities.
- *Temporal vacuuming* provides a way to control the growth of an otherwise append-only transaction-time database.
- *TSQL2* and its successor SQL/Temporal provided a way for many in the temporal database community to coordinate their efforts in temporal query language design and implementation.
- *Temporal query processing* involves disparate architectures, from *temporal strata* outside the conventional DBMS to adding native temporal support within a DBMS.
- *Supporting transaction time* in an efficient manner in the context of transactions is challenging and generally requires changes to the kernel of a DBMS.
- *Temporal algebras* extend the conventional relational algebra. Some specific operators (e.g., *temporal aggregation*, *temporal coalescing*, *temporal joins*) have received special attention.
- Temporal storage structures and indexing techniques have also received a great deal of attention (*temporal indexing*).
- *Temporal visual languages* have also been designed that present graphical user interfaces, as contrasted with the textual form of the temporal query languages mentioned previously.

Temporal Applications

- *Temporal access control* uses temporal concepts in database security.
- *Temporal data mining* has recently received a lot of attention.
- *Time series* has also been an active area of research.
- Other applications include temporal constraint satisfaction, support for planning systems, and natural language disambiguation.

The concepts of temporal databases are also making their way into research on data warehousing, OLAP, and data streams.

Key Applications

As storage costs decrease, more databases are retaining historical data. The dual of such a decrease is that the cost of deletion is effectively increasing, as the application then has to explicitly make the decision on what to retain and what to delete, and the DBMS has to revisit the data on disk in order to move it. Some have asserted that it may be simpler to simply disallow deletion (except for purging of unneeded records, termed *temporal vacuuming*) within a DBMS, rendering all databases by default temporal databases.

Commercial products are starting to include temporal support. Most notably, the Oracle database management system has included temporal support from its 9i version. Lumigent's LogExplorer product provides an analysis tool for Microsoft SQLServer logs, to allow one to view how rows change over time (a nonsequenced transaction-time query) and then to selectively back out and replay changes, on both relational data and the schema (it effectively treats the schema as a transaction-versioned schema). aTempo's Time Navigator is a data replication tool for DB2, Oracle, Microsoft SQL Server, and Sybase that extracts information from a database to build a slice repository, thereby enabling image-based restoration of a past slice; these are transaction time-slice queries. IBM's DataPropagator can use data replication of a DB2 log to create both before and after images of every row modification to create a transaction-time database that can be later queried.

Future Directions

While much progress has been made in all of the above listed areas, temporal database concepts and technologies have yet to achieve the desired levels of simplification and comprehensibility. While many of the subtleties of temporal data and access and storage thereof have been investigated, in many cases quite thoroughly, a synthesis is still needed of these concepts and technologies into forms that are usable by novices.

Given the simplifications of data management afforded by built-in support of time in database management systems, it is hoped that DBMS vendors will continue to enhance the temporal support in their products.

Cross-references

- ▶ Bi-Temporal Indexing
- ▶ Now in Temporal Databases
- ▶ Period-Stamped Temporal Models
- ▶ Point-Stamped Temporal Models
- ▶ Probabilistic Temporal Databases
- ▶ Qualitative Temporal Reasoning
- ▶ Schema Versioning
- ▶ SQL-Based Temporal Query Languages
- ▶ Supporting Transaction Time Databases
- ▶ Temporal Access Control
- ▶ Temporal Aggregation
- ▶ Temporal Algebras
- ▶ Temporal Coalescing
- ▶ Temporal Compatibility
- ▶ Temporal Concepts in Philosophy
- ▶ Temporal Conceptual Models
- ▶ Temporal Constraints
- ▶ Temporal Database
- ▶ Temporal Data Mining
- ▶ Temporal Data Models
- ▶ Temporal Dependencies
- ▶ Temporal Granularity
- ▶ Temporal Indeterminacy
- ▶ Temporal Integrity Constraints
- ▶ Temporal Joins
- ▶ Temporal Logic in Database Query Languages
- ▶ Temporal Logical Models
- ▶ Temporal Object-Oriented Databases
- ▶ Temporal Periodicity
- ▶ Temporal Query Languages
- ▶ Temporal Query Processing
- ▶ Temporal Strata
- ▶ Temporal Vacuuming
- ▶ Temporal Visual Languages
- ▶ Temporal XML
- ▶ Time Domain
- ▶ Time Series
- ▶ TSQL2

Recommended Reading

1. Rolland C., Bodart F., and Leonard M. (eds.). In Proc. Conf. on Temporal Aspects in Information Systems. North-Holland/Elsevier, 1987.
2. Al-Tara K.K., Snodgrass R.T., and Soo M.D. A bibliography on spatio- temporal databases. Int. J. Geogr. Inf. Syst., 8(1):95–103, January–February 1994.
3. Boulour A., Anderson T.L., Dekeyser L.J., and Wong H.K.T. The role of time in information processing: a survey. ACM SIGMOD Rec., 12(3):27–50,April 1982.
4. Böhlen M.H., Gamper J., and Jensen C.S. Temporal databases. In Handbook of Database Technology, J. Hammer, M. Schneider (eds.). Computer and Information Science Series. Chapman and Hall, to appear.
5. BöhlenM.H. and Jensen C.S. Temporal data model and query language concepts. In Vol. 4.Encyclopedia of Information Systems, Academic, New York, NY, USA, 2003, pp. 437–453.
6. Chomicki J. Temporal query languages: A survey. In Proc. 1st Int. Conf. on Temporal Logic, 1994, pp. 506–534.

7. Chomicki J. and Toman D. Temporal databases. In Handbook of Time in Artificial Intelligence, M.' Fisher et al. (eds.). Elsevier, Amsterdam, The Netherlands, 2005.
8. Clifford J. and Tuzhilin A. (eds.). In Proc. International Workshop on Temporal Databases: Recent Advances in Temporal Databases, 1995.
9. Grandi F. Introducing an annotated bibliography on temporal and evolution aspects in the World Wide Web. ACM SIGMOD Rec., 33(2):84–86, June 2004.
10. Jensen C.S. and Snodgrass R.T. Temporal data management. IEEE Trans. Knowl. Data Eng., 11(1):36–44, January/February 1999.
11. López I.F.V., Snodgrass R.T., and Moon B. Spatiotemporal aggregate computation: A survey. IEEE Trans. Knowl. Data Eng., 17(2):271–286, February 2005.
12. Özsoyoğlu G. and Snodgrass R.T., Temporal and real-time databases: A survey. IEEE Trans. Knowl. Data Eng., 7(4):513–532, August 1995.
13. Roddick J.F. and Spiliopoulou M. A bibliography of temporal, spatial and spatio-temporal data mining research. SIGKDD Explor., 1(1):34–38, January 1999.
14. Snodgrass R.T. Temporal databases: Status and research directions. ACM SIGMOD Rec., 19(4):83–89, December 1990.
15. Snodgrass R.T. Temporal databases. In Proc. Int. Conf. on GIS: From Space to Territory, 1992, pp, 22–61.
16. Tansel A., Clifford J., Gadia S., Jajodia S., Segev A., and Snodgrass R.T. (eds.). Temporal databases: Theory, design, and implementation. Database Systems and Applications Series. Benjamin/Cummings, Redwood City, CA, USA, March 1993, pp. 633+xx.
17. Wu Y., Jagodia S., and Wang X.S. Temporal database bibliography update. In Temporal Databases – Research and Practice. D. Etzion, S. Jajodla and S. Sripada (eds.). Springer, Berlin, 1998, pp. 338–367.
18. Zaniolo C., Ceri S., Faloutsos C., Snodgrass R.T., Subrahmanian V.S., and Zicari R. Advanced Database Systems. Morgan Kaufmann, San Francisco, CA, 1997.

Temporal Dependencies

JEF WIJSEN
University of Mons-Hainaut, Mons, Belgium

Definition

Static integrity constraints involve only the current database state. Temporal integrity constraints involve current, past, and future database states; they can be expressed by essentially unrestricted sentences in temporal logic. Certain syntactically restricted classes of temporal constraints have been studied in their own right for considerations of feasibility or practicality; they are usually called temporal dependencies. Most temporal dependencies proposed in the literature are dynamic versions of static functional dependencies.

Historical Background

Static dependencies (functional, multivalued, join, and other dependencies) have been investigated in depth since the early years of the relational model. Classical problems about dependencies concern logical implication and axiomatization. The study of a particular dependency class is often motivated by its practical importance in databases. This is undeniably the case for the notion of functional dependency (FD), which is fundamental in database design. A dynamic version of functional dependencies was first proposed by Vianu [7]. Since the mid 1990's, several other temporal variants of the notion of FD have been introduced.

Foundations

This section gives an overview of several temporal dependency classes proposed in the literature. With the exception of the notion of dynamic algebraic dependency (DAD) [2], all temporal dependencies here presented can be seen as special cases of the notion of constraint-generating dependency (CGD) [1]. The formalism of CGD, presented near the end, thus allows to compare and contrast different temporal dependency classes.

Functional Dependencies Over Temporal Databases

Since the semantics of temporal versions of FDs will be explained in terms of static FDs, the standard notion of FD is recalled next. All the following definitions are relative to a fixed set $U = \{A_1,...,A_n\}$ of *attributes*.

Definition A tuple over U is a set $t = \{A_1 : c_1,...,A_n : c_n\}$, *where each c_i is a constant. If $X \subseteq U$, then $t[X]$ denotes the restriction of t to X. A relation over U is a finite set of tuples over U.*

A functional dependency (FD) over U is an expression $X \rightarrow Y$ where $X,Y \subseteq U$. A relation I *over U satisfies the FD $X \rightarrow Y$ if for all tuples $s,t \in I$, if $s[X] = t[X]$, then $s[Y] = t[Y]$.*

When evaluating FDs over temporal relations, one may want to treat timestamp attributes different from other attributes. To illustrate this, consider the temporal relation *EmpInterval* with its obvious meaning.

EmpInterval

Name	Sex	Sal	Project	From	To
Ed	M	10K	Pulse	1	3
Ed	M	10K	Wizard	2	3
Ed	M	12K	Wizard	4	4

An employee has a unique sex and a unique salary, but can work for several projects. The salary, unlike the sex, may change over time. The relation *EmpInterval* is "legal" with respect to these company rules, because for any time point i, the *snapshot* relation $\{t[\mathit{Name}, \mathit{Sex}, \mathit{Sal}, \mathit{Project}] \mid t \in \mathit{EmpInterval}, t(\mathit{From}) \leq i \leq t(\mathit{To})\}$ satisfies *Name* →*Sex* and *Name* → *Sal*. Note that the relation *EmpInterval* violates the FD *Name* → *Sal*, because the last two tuples agree on *Name* but disagree on *Sal*. However, since these tuples have disjoint periods of validity, they do not go against the company rules.

Hence, the intended meaning of an FD expressed over a temporal relation may be that the FD must be satisfied at every snapshot. To indicate that *Name* →*Sal* has to be evaluated on snapshots, Jensen et al. [6] use the notation $\mathit{Name} \overset{\mathrm{T}}{\rightarrow} \mathit{Sal}$. The FD *Name* →*Sex* needs no change, because the sex of an employee should be unique not only at each snapshot, but also over time.

Chomicki and Toman [3] note that no special syntax is needed if tuples are timestamped by time points. For example, given the following point-stamped temporal relation, one can simply impose the classical FDs *Name* →*Sex* and *Name*, *T* →*Sal*.

EmpPoint

Name	Sex	Sal	Project	T
Ed	M	10K	Pulse	1
Ed	M	10K	Pulse	2
Ed	M	10K	Pulse	3
Ed	M	10K	Wizard	2
Ed	M	10K	Wizard	3
Ed	M	12K	Wizard	4

Vianu's Dynamic Functional Dependency [7]

Consider an employee table with attributes *Name*, *Sex*, *Merit*, and *Sal*, with their obvious meanings. The primary key is *Name*. Assume an annual companywide update of merits and salaries. In the relation shown next, every tuple is followed by its updated version. Old values appear in columns with a caron (ˇ), new values in columns with a caret (ˆ). For example, John Smith's salary increased from 10 to 12K. Such a relation that juxtaposes old and new values is called *action relation*.

⊢ old ⊣				⊢ new ⊣			
Nǎme	Sěx	Měrit	Sǎl	Nâme	Sêx	Mêrit	Sâl
John Smith	M	Poor	10K	John Smith	M	Good	12K
An Todd	F	Fair	10K	An Todd	F	Good	12K
Ed Duval	M	Fair	10K	Ed Duval	M	Fair	10K

The company's policy that "*Each new salary is determined solely by new merit and old salary*" can be expressed by the classical FD *Sǎl*, *Mêrit* →*Sâl* on the above action relation. In particular, since John Smith and An Todd agree on old salary and new merit, they must have the same new salary. Such FDs on action relations were introduced by Vianu [7] and called dynamic functional dependencies.

Definition For each $A_i \in U$, assume that $\check{A}_i$ and $\hat{A}_i$ are new distinct attributes. Define $\check{U} = \{\check{A}_1,\ldots,\check{A}_n\}$ and $\hat{U} = \{\hat{A}_1,\ldots,\hat{A}_n\}$. For $t = \{A_1 : c_1,\ldots,A_n : c_n\}$, define:

$$\check{t} = \left\{\check{A}_1 : c_1,\ldots,\check{A}_n : c_n\right\},\ \textit{a tuple over } \check{U};\ \textit{and}$$
$$\hat{t} = \left\{\hat{A}_1 : c_1,\ldots,\hat{A}_n : c_n\right\},\ \textit{a tuple over } \hat{U}$$

A *Vianu dynamic functional dependency* (VDFD) over U is an FD $X \rightarrow Y$ over $\check{U}\hat{U}$ such that for each $A \in Y$, XA contains at least one attribute from $\check{U}$ and one attribute from $\hat{U}$.

An update *over U is a triple $\langle I, \mu, J\rangle$, where I and J are relations over U and μ is a bijective mapping from I to J. The update $\langle I, \mu, J\rangle$* satisfies *the VDFD $X \rightarrow Y$ if the* action relation $\{\check{t} \cup \hat{s} \mid t \in I, s = \mu(t)\}$ *satisfies the FD $X \rightarrow Y$.*

The notion of VDFD directly extends to sequences of database updates. The interaction between dynamic VDFDs and static FDs is studied in [7].

Temporal Extensions of Functional Dependency Proposed by Wijsen [9, 10, 11, 12]

Instead of extending each tuple with its updated version, as is the case for VDFDs, one can take the union of the old relation and the new relation:

Name	Sex	Merit	Sal		
John smith	M	Poor	10K	⊤	
An todd	F	Fair	10K	old	
Ed duval	M	Fair	10K	⊥	⊤
John smith	M	Good	12K		new
An todd	F	Good	12K		⊥

The company's policy that "*Every change in merit (promotion or demotion) gives rise to a salary change*," is expressed by *Name*, *Sal* $\overset{\circ}{\rightarrow}$ *Merit* and means that the FD *Name*, *Sal* →*Merit* must be satisfied by the union of the old and the new relation. In particular, since Ed Duval's salary did not change, his merit cannot have changed either. Likewise, "*The sex of an employee cannot change*" is expressed by *Name* $\overset{\circ}{\rightarrow}$ *Sex*. The construct $\overset{\circ}{\rightarrow}$ naturally generalizes to database histories that involve more than two database states.

Definition A Wijsen dynamic functional dependency (WDFD) over *U* is an expression of the form $X \overset{\circ}{\rightarrow} Y$, where $X, Y \subseteq U$.

A database history *is a sequence* $\langle I_1,I_2,I_3,\ldots\rangle$, *where each* I_i *is a relation over U. This history* satisfies $X \overset{\circ}{\rightarrow} Y$ *if for every* $i \in \{1,2,\ldots\}$, $I_i \cup I_{i+1}$ satisfies $X \rightarrow Y$.

Although I_{i+1} can be thought of as the result of an update performed on I_i, there is no need to model a one-one relationship between the tuples of both relations, as was the case for VDFDs. VDFDs and WDFDs capture different types of constraints, even in the presence of some attribute that serves as a time-invariant tuple-identifier. This difference will be illustrated later on in the discussion of constraint-generating dependencies.

In practice, database histories will be be stored in relations with timestamped tuples. Like FDs, WDFDs can result in predictable (i.e., redundant) values. For example, if the following relation has to satisfy *Name*, *Sal* $\overset{\circ}{\rightarrow}$ *Merit*, then the value for the placeholder † must be equal to "Poor." Wijsen [9] develops temporal variants of 3NF to avoid data redundancy caused by WDFDs.

Name	City	Merit	Sal	From	To
Ed	Paris	Poor	10K	1	2
Ed	London	†	10K	3	4

WDFDs can be naturally generalized as follows: instead of interpreting FDs over unions of successive database states, the syntax of FD is extended with a binary relation on the time domain, called *time accessibility relation*, that indicates which tuple pairs must satisfy the FD.

Definition *A* time accessibility relation *(TAR) is a subset of* $\{(i,j) \mid 1 \leq i \leq j\}$. *A* generalized WDFD *over U is an expression* $X{\rightarrow}_{\alpha}Y$, *where* $X,Y \subseteq U$ *and* α *is a TAR. This generalized WDFD is satisfied by database history* $\langle I_1,I_2,I_3,\ldots\rangle$ *if for all* $(i,j) \in \alpha$, $s \in I_i$, $t \in I_j$, *if* $s[X] = t[X]$, *then* $s[Y] = t[Y]$.

If the TAR Next is defined by `Next` =$\{(1,1),(1,2),(2,2),(2,3),(3,3),(3,4),\ldots\}$, then $X{\rightarrow}_{\mathtt{Next}}Y$ and $X \overset{\circ}{\rightarrow} Y$ are equivalent. TARs can also capture the notion of time granularity. For example, `MonthTAR` can be defined as the TAR containing (i, j) whenever $i \leq j$ and time points *i*, *j* belong to the same month. Then, $Name{\rightarrow}_{\mathrm{MonthTAR}}$ *Sal* expresses that the salary of an employee cannot change within a month.

The temporal functional dependencies proposed in [11] extend this formalism with a notion of identity, denoted by λ, similar to object-identity. The identity is time-invariant and allows to relate old and new versions of the same object. For example, *Emp*: $\lambda{\rightarrow}_{\mathrm{Next}}Name$ means that the name of an employee object cannot change from one time to the next.

Trend dependencies [10,12] extend generalized WDFDs in still another way by allowing both equalities and inequalities. They can be seen as a temporal extension of the concept of *order dependency* introduced by Ginsburg and Hull [4]. For example, $(Name,=)\ {\rightarrow}_{\mathrm{Next}}(Sal,\leq)$ expresses that the salary of an employee cannot decrease. Technically, it is satisfied by a database history $\langle I_1,I_2,I_3,\ldots\rangle$ if for all $(i, j) \in$ Next, if $s \in I_i$ and $t \in I_j$ and $s(Name) = t(Name)$, then $s(Sal) \leq t(Sal)$.

Wang et al.'s Temporal Functional Dependency [8]

The dynamic versions of FDs introduced by Vianu and Wijsen can impose constraints on tuples valid at successive time points. Wang et al.'s notion of temporal functional dependency (TFD) concentrates on temporal granularity and compares tuples valid during the same granule of some temporal granularity. The main idea is captured by the following definitions that are simplified versions of the ones found in [8].

Definition *Assume a linear time domain* $(D, <)$. *Every nonempty subset of* D *is called a* granule. *Two distinct granules* G_1 *and* G_2 *are said to be* non-interleaved *if each point of either granule is smaller than all points of the other granule (i.e., either* $\forall d_1 \in G_1 \forall d_2 \in G_2 (d_1 < d_2)$ or $\forall d_1 \in G_1 \forall d_2 \in G_2 (d_2 < d_1)$). *A* granularity *is a set* $\mathcal{G}$ *of pairwise non-interleaved granules.*

Other granularity notions found in the literature often assume that granules are indexed by integers. Such index has been omitted here to simplify the formalism.

Common granularities are Month and Year. If granularity $\mathcal{G}$ is associated with temporal relation I, then all tuples of I must be timestamped by granules of $\mathcal{G}$. For example, all tuples in the following relation are timestamped by months. Practical labels, like Nov-2007, are used to denote granules of time points.

EmpMonth

Name	Sal	Position	T : Month
Ed	10K	Lecturer	Nov-2007
Ed	11K	Lecturer	Dec-2007
Ed	12K	Professor	Jan-2008

Definition *Assume a set* $U = \{A_1,\ldots,A_n\}$ *of attributes and a timestamp attribute* $T \notin U$. A timestamped tuple with granularity $\mathcal{G}$ *(or simply* $\mathcal{G}$-tuple) *over* U *is a set* $\{A_1\colon c_1,\ldots,A_n : c_n, T : G\}$, *where each* c_i *is a constant and* $G \in \mathcal{G}$. If $t = \{A_1 : c_1,\ldots,A_n : c_n, T : G\}$, *then define* $t[U] = \{A_1 : c_1,\ldots,A_n : c_n\}$ and $t(T) = G$. A timestamped relation with granularity $\mathcal{G}$ (or simply $\mathcal{G}$-relation) *over* U *is a finite set of* $\mathcal{G}$*-tuples over* U.

The TFD $\mathit{Name}\rightarrow_{\text{Month}}\mathit{Sal}$ expresses that the salary of an employee cannot change within a month. Likewise, $\mathit{Name}\rightarrow_{\text{Year}}\mathit{Position}$ expresses that the position of an employee cannot change within a year. Both dependencies are satisfied by the relation *EmpMonth* shown above. To check $\mathit{Name}\rightarrow_{\text{Year}}\mathit{Position}$, it suffices to verify whether for each year, the FD $\mathit{Name} \rightarrow \mathit{Position}$ is satisfied by the set of tuples whose timestamps fall in that year. For example, for the year 2007, the relation $\{t[\mathit{Name}, \mathit{Sal}, \mathit{Position}] \mid t \in \mathit{EmpMonth}, t(T) \subseteq 2007\}$ must satisfy $\mathit{Name} \rightarrow \mathit{Position}$. This is captured by the following definition.

Definition A temporal functional dependency (*TFD*) *over* U *is an expression* $X \rightarrow_{\mathcal{H}} Y$, *where* $X, Y \subseteq U$ *and* $\mathcal{H}$ *is a granularity. A* $\mathcal{G}$*-relation* I *over* U *satisfies* $X \rightarrow_{\mathcal{H}} Y$ *if for each granule* $H \in \mathcal{H}$, *the relation* $\{t[U] \mid t \in I, t(T) \subseteq H\}$ *satisfies the FD* $X \rightarrow Y$.

Wang et al. extend classical normalization theory to deal with data redundancy caused by TFDs. For example, since positions cannot change within a year, Ed must necessarily occupy the same position in Nov-2007 and Dec-2007. To avoid this redundancy, the information on positions must be moved into a new relation with time granularity Year, as shown above. After this decomposition, Ed's position in 2007 is only stored once.

Name	Sal	T : Month
Ed	10K	Nov-2007
Ed	11K	Dec-2007
Ed	12K	Jan-2008

Name	Position	T : Year
Ed	Lecturer	2007
Ed	Professor	2008

Constraint-Generating Dependencies [1]

Classical dependency theory assumes that each attribute of a relation takes its values in some uninterpreted domain of constants, which means that data values can only be compared for equality and disequality. The notion of *constraint-generating dependency* (CGD) builds upon the observation that, in practice, certain attributes take their values in specific domains, such as the integers or the reals, on which predicates and functions, such as $\leq$ and $+$, are defined. CGDs can thus constrain data values by formulas in the first-order theory of the underlying domain.

A *constraint-generating k-dependency* takes the following form in tuple relational calculus:

$$\forall t_1 \forall t_2 \ldots \forall t_k ((R_1(t_1) \wedge \ldots \wedge R_k(t_k) \wedge C[t_1,\ldots,t_k]) \Longrightarrow C'[t_1,\ldots,t_k])$$

where C and C' are arbitrary constraint formulas relating the values of various attributes in the tuples $t_1,\ldots,t_k$.

CGDs naturally arise in temporal databases: timestamp attributes take their values from a linearly ordered time domain, possibly equipped with arithmetic. Baudinet et al. [1] specifically mention that the study of CGDs was inspired by the work of Jensen and Snodgrass on temporal specialization and generalization [5]. For example, assume a relation R with two temporal attributes, denoted VT and TT. Every tuple $t \in R$ stores information about some event, where $t(TT)$ is the (transaction) time when the event was recorded in the database, and $t(VT)$ is the (valid) time when the event happened in the real world. The following CGD expresses that every event should be recorded within c time units after its occurrence:

$$\forall t(R(t) \Longrightarrow (t(VT) < t(TT) \wedge t(TT) \leq t(VT) + c)).$$

Given appropriate first-order theories for the underlying domains, CGDs can capture the different temporal dependencies introduced above. Assume a point-stamped temporal relation $Emp(Name, Merit, Sal, VT)$ with its obvious meaning. Assume that $Name$ provides a unique and time-invariant identity for each employee. Then, the VDFD $S\check{a}l, \hat{M}erit \rightarrow \hat{S}al$ can be simulated by the following constraint-generating 4-dependency. In this formula, the tuples s and s' concern the same employee in successive database states (likewise for t and t').

$$\forall s \forall s' \forall t \forall t' \left(\begin{pmatrix} Emp(s) \wedge Emp(s') \\ \wedge s(Name) = s'(Name) \\ \wedge s'(VT) = s(VT) + 1 \\ \wedge Emp(t) \wedge Emp(t') \\ \wedge t(Name) = t'(Name) \\ \wedge t'(VT) = t(VT) + 1 \\ \wedge s(Sal) = t(Sal) \\ \wedge s'(Merit) = t'(Merit) \\ \wedge s(VT) = t(VT) \end{pmatrix} \begin{matrix} \Rightarrow s'(Sal) = \\ t'(Sal)) \end{matrix} \right)$$

The WDFD $Name, Sal \overset{\circ}{\rightarrow} Merit$ can be expressed as a constraint-generating 2-dependency:

$$\forall t \forall t' \left(\begin{pmatrix} Emp(t) \wedge Emp(t') \\ \wedge t(Name) = t'(Name) \\ \wedge t(Sal) = t'(Sal) \\ \wedge (t'(VT) = t(VT) \vee t'(VT) \\ = t(VT) + 1) \end{pmatrix} \Rightarrow \begin{matrix} t(Merit) = \\ t'(Merit) \end{matrix} \right)$$

Assume a binary predicate $Month$ on the time domain such that $Month(t_1, t_2)$ is true if t_1 and t_2 are two time instants within the same month. Then, $Name \rightarrow_{\text{Month}} Sal$ can be expressed as a constraint-generating 2-dependency:

$$\forall t \forall t' \left(\begin{pmatrix} Emp(t) \wedge Emp(t') \\ \wedge t(Name) = t'(Name) \\ \wedge Month(t(VT), t'(VT)) \end{pmatrix} \Rightarrow \begin{matrix} t(Sal) = \\ t'(Sal) \end{matrix} \right)$$

Dynamic Algebraic Dependencies [2]

Consider a database schema containing $Emp(Name, Merit, Sal)$ and $WorksFor(Name, Project)$. A company rule states that "*The Pulse project only recruits employees whose merit has been good at some past time.*" The relational algebra expression E_0 shown below gets the workers of the Pulse project; the expression F_0 gets the good workers. In a consistent database history, if the tuple t is in the answer to E_0 on the current database state, then t is in the answer to F_0 on some (strict) past database state.

$$E_0 = \pi_{Name}(\sigma_{Project=\text{"Pulse"}} WorksFor)$$
$$F_0 = \pi_{Name}(\sigma_{Merit=\text{"Good"}} Emp)$$

Definition *A relational algebra query E is defined as usual using the named relational algebra operators* $\{\sigma, \pi, \bowtie, \rho, \cup, -\}$. *A* dynamic algebraic dependency *(DAD) is an expression of the form EF, where E and F are relational algebra queries over the same database schema and with the same output schema. A finite database history* $\langle I_0, I_1, \ldots, I_n \rangle$, where $I_0 = \{\}$, satisfies *the DAD EF if for each* $i \in \{1, \ldots, n\}$, *for each* $t \in E(I_i)$, *there exists* $j \in \{1, \ldots, i-1\}$ *such that* $t \in F(I_j)$.

The preceding definition uses relational algebra. Nevertheless, every DAD EF can be translated into temporal first-order logic in a straightforward way. Let $Now(\vec{x})$ and $Past(\vec{x})$ be safe relational calculus queries equivalent to E and F respectively, with the same free variables $\vec{x}$. Then the DAD EF is equivalent to the closed formula:

$$\forall \vec{x}(Now(\vec{x}) \Rightarrow \blacklozenge\, Past(\vec{x})).$$

It seems that the expressive power of DADs relative to other dynamic constraints has not been studied in depth. Notice that the simple DAD $\forall x(R(x) \Rightarrow \blacklozenge S(x))$

Name	Sex	Sal	Project	Day	Month	Year
Ed	M	10K	Pulse	29-Aug-2007	Aug-2007	2007
Ed	M	10K	Pulse	30-Aug-2007	Aug-2007	2007
Ed	M	10K	Pulse	31-Aug-2007	Aug-2007	2007
Ed	M	10K	Wizard	30-Aug-2007	Aug-2007	2007
Ed	M	10K	Wizard	31-Aug-2007	Aug-2007	2007
Ed	M	12K	Wizard	1-Sep-2007	Sep-2007	2007

is tuple-generating, in the sense that tuples in *R* require the existence of past tuples in *S*. The other temporal dependencies presented in this entry are not tuple-generating.

Bidoit and De Amo [2] study the existence of an operational specification that can yield all and only the database histories that are consistent. The operational specification assumes that all database updates are performed through a fixed set of update methods, called transactions. These transactions are specified in a transaction language that provides syntax for concatenation and repetition of elementary updates (insert a tuple, delete a tuple, erase all tuples).

Key Applications

An important motivation for the study of FDs in database courses is schema design. The notion of FD is a prerequisite for understanding the principles of "good" database design (3NF and BCNF). In the same line, the study of temporal functional dependencies has been motivated by potential applications in temporal database design. It seems, however, that one can go a long way in temporal database design by practicing classical, non-temporal normalization theory. As suggested in [3,8], one can put timestamp attributes on a par with ordinary attributes, write down classical FDs, and apply a standard 3NF decomposition. For example, assume a database schema $\{Name, Sex, Sal, Project, Day, Month, Year\}$.
The following FDs apply:

$$\begin{aligned} &Name \rightarrow Sex\\ &Name, Month \rightarrow Sal\\ &Day \rightarrow Month\\ &Month \rightarrow Year \end{aligned}$$

The latter two FDs capture the relationships that exist between days, months, and years. The standard 3NF synthesis algorithm finds the following decomposition – the last two components may be omitted for obvious reasons:

$$\begin{aligned} &\{Name, Project, Day\}\\ &\{Name, Sex\}\\ &\{Name, Sal, Month\}\\ &\{Day, Month\}\\ &\{Month, Year\} \end{aligned}$$

This decomposition, resulting from standard normalization, is also "good" from a temporal perspective. In general, the "naive" approach seems to prevent data redundancy in all situations where relationships between granularities can be captured by FDs. It is nevertheless true that FDs cannot capture, for example, the relationship between weeks and months. In particular, *Week* →*Month* does not hold since certain weeks contain days of two months. In situations where FDs fall short in specifying relationships among time granularities, there may be a need to timestamp by new, artificial time granularities in order to avoid data redundancy [8].

Cross-references

► Temporal Granularity
► Temporal Integrity Constraints

Recommended Reading

1. Baudinet M., Chomicki J., and Wolper P. Constraint-generating dependencies. J. Comput. Syst. Sci., 59(1):94–115, 1999.
2. Bidoit N. and de Amo S. A first step towards implementing dynamic algebraic dependences. Theor. Comput. Sci., 190 (2):115–149, 1998.
3. Chomicki J. and Toman D. Temporal databases. In M. Fisher, D.M. Gabbay, L. Vila (eds.). Handbook of Temporal Reasoning in Artificial Intelligence. Elsevier Science, 2005.
4. Ginsburg S. and Hull R. Order dependency in the relational model. Theor. Comput. Sci., 26:149–195, 1983.
5. Jensen C.S. and Snodgrass R.T. Temporal specialization and generalization. IEEE Trans. Knowl. Data Eng., 6(6):954–974, 1994.

6. Jensen C.S., Snodgrass R.T., and Soo M.D. Extending existing dependency theory to temporal databases. IEEE Trans. Knowl. Data Eng., 8(4):563–582, 1996.
7. Vianu V. Dynamic functional dependencies and database aging. J. ACM, 34(1):28–59, 1987.
8. Wang X.S., Bettini C., Brodsky A., and Jajodia S. Logical design for temporal databases with multiple granularities. ACM Trans. Database Syst., 22(2):115–170, 1997.
9. Wijsen J. Design of temporal relational databases based on dynamic and temporal functional dependencies. In Temporal Databases. J. Clifford A. Tuzhilin (eds.). Springer, Berlin, 1995, pp. 61–76.
10. Wijsen J. Reasoning about qualitative trends in databases. Inf. Syst., 23(7):463–487, 1998.
11. Wijsen J. Temporal FDs on complex objects. ACM Trans. Database Syst., 24(1):127–176, 1999.
12. Wijsen J. Trends in databases: Reasoning and mining. IEEE Trans. Knowl. Data Eng., 13(3):426–438, 2001.

Temporal Domain

▶ Lifespan
▶ Time Domain

Temporal Element

Christian S. Jensen[1], Richard Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Time period set

Definition

A *temporal element* is a finite union of *n*-dimensional time intervals. Special cases of temporal elements include *valid-time elements*, *transaction-time elements*, and *bitemporal elements*, which are finite unions of valid-time intervals, transaction-time intervals, and bitemporal intervals, respectively.

Key Points

Assuming an *n*-dimensional time domain, an interval is the product of *n* convex subsets drawn from each of the constituent dimensions.

Given a finite, one-dimensional time domain, a temporal element may be defined equivalently as a subset of the time domain. If the time domain is unbounded and thus infinite, some subsets of the time domain are not temporal elements. These subsets cannot be enumerated in finite space. For non-discrete time domains, the same observation applies.

Temporal elements are often used as timestamps. Unlike time periods, they are closed under the set theoretic operations of union, intersection, and complement, which is a very desirable property when formulating temporal database queries.

The term "temporal element" has been used to denote the concept of a valid-time interval. However, "temporal" is generally used as generic modifier, so more specific modifiers are adopted here for specific kinds of temporal elements. The term "time period set" is an early term for a temporal element. The adopted term has been used much more frequently.

Cross-references

▶ Bitemporal Interval
▶ Temporal Database
▶ Time Interval
▶ Time Period
▶ Transaction Time
▶ Time Domain
▶ Temporal Query Languages
▶ Valid Time

Recommended Reading

1. Gadia S.K. Temporal element as a primitive for time in temporal databases and its application in query optimization. In Proc. 13th ACM Annual Conf. on Computer Science, 1986, p. 413.
2. Gadia S.K. A homogeneous relational model and query languages for temporal databases. ACM Trans. Database Syst., 13(4):418–448, December 1988.
3. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.), Springer-Verlag, Berlin, 1998, pp. 367–405.

Temporal Evolution

▶ History in Temporal Databases

Temporal Expression

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

A *temporal expression* is a syntactic construct used, e.g., in a query that evaluates to a temporal value, i.e., an instant, a time period, a time interval, or a temporal element.

Key Points

Advanced by Gadia [1], a temporal expression is a convenient temporal query language construct.

First, any temporal element is considered a temporal expression. As Gadia uses a discrete and bounded time domain, any subset of the time domain is then a temporal expression. Next, an attribute value of a tuple in Gadia's data model is a function from the time domain to some value domain. Likewise, the attribute values of a tuple are valid during some temporal element. To illustrate, consider an (ungrouped) relation with attributes `Name` and `Position`. An example tuple in this relation is:

$$\begin{aligned}&(\langle[4,17]\ \mathit{Bill}\rangle,\\ &\quad\langle[4,8]\mathit{Assistant},\\ &\quad[9,13]\ \mathit{Associate}, [14,17]\mathit{Full}\rangle)\end{aligned}$$

Now let X be an expression that returns a function from the time domain to some value domain, such as an attribute value, a tuple, or a relation. Then the temporal expression $[[X]]$ returns the domain of X. Using the example from above, the temporal expression [[`Position`]] evaluates to [4,17] and the temporal expression [[`Position <> Associate`]] evaluates to $[4,8] \cup [14,17]$.

The terms "Boolean expression" and "relational expression" may be used for clearly identifying expressions that evaluate to Boolean values and relations.

Cross-references

- ► SQL-Based Temporal Query Languages
- ► Temporal Database
- ► Temporal Element
- ► Time Instant
- ► Time Interval
- ► Time Period

Recommended Reading

1. Gadia S.K. A homogeneous relational model and query languages for temporal databases. ACM Trans. Database Syst., 13(4):418–448, December 1988.
2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.), Springer-Verlag, Berlin, 1998, pp. 367–405.

Temporal Generalization

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

Temporal generalization comes in three guises. Consider a temporal database in which data items are timestamped with valid and transaction time. Temporal generalization occurs when weakening constraints hitherto applied to the timestamps. Used in this sense, temporal generalization is the opposite of temporal specialization.

Next, a temporal relation is generalized when new timestamps are being associated with its tuples. In larger information systems where data items flow between multiple temporal relations, items may accumulate timestamps by keeping their previous timestamps and gaining new timestamps as they are entered into new temporal relations. Thus, a tuple in a particular relation has multiple timestamps: a valid timestamp, a *primary* transaction timestamp, which records when the tuple was stored in this relation, one or more *inherited* transaction timestamps that record when the tuple was stored in previous relations, and one or more additional timestamps that record when the tuple was manipulated elsewhere in the system.

Finally, a more involved notion of temporal generalization occurs when a derived relation inherits the transaction timestamps from the relation(s) it is derived from.

By describing the temporal generalization that occurs in an information system, important semantics are captured that may be utilized for a variety of purposes. For example, a temporal relation may be queried, with specific restrictions, from a temporal relation that receives tuples with some delay from that relation. Another use is to increase the efficiency of query processing.

Key Points

The first notion of temporal generalization is simply the opposite of temporal specialization.

As an example of the second notion of temporal generalization, consider the following complex yet realistic scenario of a collection of temporal relations maintained by the transportation department of a state government. An *employee relation* is maintained on the workstation of each manager in this department, recording schedules, budgets, and salary levels for the employees under that manager. For the entire department, a single *personnel relation* is maintained on the administrative computer under the data processing group, which also maintains a *financial relation*. The bank, responsible for salary payments, maintains an *accounts relation*. Data items in the form of timestamped tuples move from the employee relation to the personnel relation and then to the financial relation and ultimately to the accounts relation, accumulating transaction timestamps each time they enter a new database. Each timestamp has a relationship with the other transaction timestamps and with the valid timestamp. These can be stated in the schema and utilized during querying to ensure accurate results.

As an example of the third notion of temporal generalization, consider process control in a manufacturing plant. Values from sensors that capture process characteristics such as pressure and temperature may be stored in temporal relations. This data may subsequently be processed further to derive new data that capture relevant aspects of the process being monitored at a higher level of abstraction. The original data, from which the new data was derived, may be stored together with the new data, to capture the lineage, or provenance, of that data. As a result, the new data inherits timestamps from the original data.

Cross-references

- ▶ Data Stream
- ▶ Temporal Database
- ▶ Temporal Specialization
- ▶ Transaction Time
- ▶ Valid Time

Recommended Reading

1. Jensen C.S. and Snodgrass R.T. Temporal specialization and generalization. IEEE Trans. Knowl. Data Eng., 5(6):954–974, December 1994.

Temporal Granularity

Claudio Bettini[1], X. Sean Wang[2], Sushil Jajodia[3]
[1]Università degli Studi di Milano, Milan, Italy
[2]University of Vermont, Burlington, VT, USA
[3]George Mason University, Fairfax, VA, USA

Synonyms

Time granularity; Temporal type

Definition

In the context of databases, a temporal granularity can be used to specify the temporal qualification of a set of data, similar to its use in the temporal qualification of statements in natural languages. For example, in a relational database, the timestamp associated with an attribute value or a tuple may be interpreted as associating that data with one or more granules of a given temporal granularity (e.g., one or more *days*). As opposed to using instants from a system-specific time domain, the use of user-defined granularities enables both more compact representations and temporal qualifications at different levels of abstraction. Temporal granularities include very common ones like *hours*, *days*, *weeks*, *months*, and *years*, as well as the evolution and specialization of these granularities for specific contexts or applications: *trading days*, *banking days*, *academic semesters*, etc.. Intuitively, a temporal granularity is defined by grouping sets of instants from a time domain into so-called *granules* in a rather flexible way with some mild conditions. For example, the granularity *business days* is defined as the infinite set of granules, each including the time instants composing one working day. A label, for example a date for a day granule, is often used to refer to a particular granule of a granularity. Answering queries in terms of a granularity different from the one used to store data in a database is not simply a matter of syntactic granularity conversion, but it involves subtle semantics issues.

Historical Background

Temporal granularities have always had a relevant role in the qualification of statements in natural languages, and they still play a major role according to a 2006 study by Oxford University. The study includes words "day," "week," "month," and "year" among the 25 most common nouns in the English language. Temporal granularities have also been used for a long time in computer applications, including personal information management, project management, scheduling, and more. Interestingly, in many situations, their use is limited to a very few common ones, their semantics is often simplified and sometimes confusing, and their management is hard-coded in applications with ad-hoc solutions. The database community seems to be a major driver in formalizing temporal granularities. One of the earliest formalizations was proposed in [5]. At the same time the AI community was investigating formalisms to represent calendar unit systems [9,10]. In the early 1990s, the relevant role played by time granularity and calendars in temporal databases, as well as the need to devise algorithms to manage granular data, became widely recognized by the research community, and some significant progress has been made [4, 11, 13, 15]. Some support for granularities was also included in the design of the temporal query language TSQL2. A comprehensive formal framework for time granularities to be applied in several areas of database research emerged in the mid-1990s, and has been progressively refined in the following years through the investigation of its applications in data mining, temporal database design, query processing, and temporal constraint reasoning [3]. This framework is based on a set-theoretic approach (partly inspired by [5]) and on an algebraic representation, and it includes techniques to compute basic as well as more complex operations on granules and granularities. The basic notions found a large consensus in the database community [1]. The use of logic to specify formal properties and to reason about granularities as defined in the above framework was investigated in [6]. Programming oriented support for integrating multiple calendars was provided in [12]. In the logic community, an independent line of research on representation and reasoning with multiple granularities investigated classical and non-classical logic extensions based on multi-layered time domains, with applications to the specification of real-time reactive systems. This approach is extensively described in [8]. More recently, the use of automata to represent granularities and to perform basic operations on them has been proposed [7]. This work, partly inspired by previously proposed string-based representation of granularities [15], has the benefit of providing compact representation of granularities. Moreover, decision procedures for some basic problems, such as granularity equivalence and minimization, can be applied directly on that representation.

Foundations

What follows is an illustration of the main formal definitions of temporal granularities and their relationships according to the set-theoretic, algebraic approach.

Definitions

A temporal granularity can be intuitively described as a sequence of time granules, each one consisting of a set of time instants. A granule can be composed of a single instant, a set of contiguous instants (time-interval), or even a set of non-contiguous instants. For example, the `September 2008 business-month`, defined as the collection of all the business days in September 2008, can be used as a granule. When used to describe a phenomena or, in general, when used to timestamp a set of data, a granule is perceived as a non-decomposable temporal entity. A formal definition of temporal granularity is the following.

Assume a time domain T as a set of totally ordered time instants. A *granularity* is a mapping G from the integers (the *index set*) to the subsets of the time domain such that:

(1) If $i < j$ and $G(i)$ and $G(j)$ are non-empty, then each element in $G(i)$ is less than all the elements in $G(j)$.

(2) IIf $i < k < j$ and $G(i)$ and $G(j)$ are non-empty, then $G(k)$ is non-empty.

Each non-empty set $G(i)$ in the above definition is called *granule*.

The first condition in the granularity definition states that granules in a granularity do not overlap and that their index order is the same as their time domain order. The second condition states that the subset of the index set for the granules is contiguous. Based on the above definition, while the time domain can be discrete, dense, or continuous, a granularity defines a countable set of granules; each granule is identified by an integer. The index set can thereby provide an "encoding" of the granularity in a computer. Two granules $G(i)$ and $G(j)$ are *contiguous* if there does not exist $t \in T$ such that $\forall s \in G(i)(s < t)$ and $\forall s \in G(j)(s > t)$. Independently, there may be a "textual representation" of each non-empty granule, termed its *label*, that is used for input and output. This representation is generally a string that is more descriptive than the granule's index. An associated mapping, the *label mapping*, defines for each label a unique corresponding index. This mapping can be quite complex, dealing with different languages and character sets, or can be omitted if integers are used directly to refer to granules. For example, "August 2008" and "September 2008" are two labels each referring to the set of time instants (a granule) corresponding to that month.

A granularity is *bounded* if there exist lower and upper bounds k_1 and k_2 in the index set such that $G(i) = \emptyset$ for all i with $i < k_1$ or $k_2 < i$.

The usual collections days, months, weeks and years are granularities. The granularity describing all years starting from 2000 can be defined as a mapping that takes an arbitrary index i to the subset of the time domain corresponding to the year 2000, $i + 1$ to the one corresponding to the year 2001, and so on, with all indexes less than i mapped to the empty set. The years from 2006 to 2010 can also be represented as a granularity G, with $G(2006)$ identifying the subset of the time domain corresponding to the year 2006, $G(2007)$ to 2007, and so on, with $G(i) = \emptyset$ for each $i < 2006$ and $i > 2010$.

The union of all the granules in a granularity G is called the *image* of G. For example, the image of `business-days-since-2000` is the set of time instants included in each granule representing a business-day, starting from the first one in 2000. The single interval of the time domain starting with the greatest lower bound of the image of a granularity G and ending with the least upper bound ($-\infty$ and $+\infty$ are considered valid lower/upper bounds) is called the *extent* of G. Note that many commonly used granularities (e.g., `days`, `months`, `years`) have their image equal to their extent, since each granule is formed by a set of contiguous elements of the time domain and each pair of contiguous indexes is mapped to contiguous granules.

Granularity Relationships

In the following, some commonly used relationships between granularities are given.

A granularity G *groups into* a granularity H, denoted $G \unlhd H$, if for each index j there exists a (possibly infinite) subset S of the integers such that $H(j) = \cup_{i \in S} G(i)$. For example, `days` groups into `weeks`, but `weeks` does not group into `months`.

A granularity G is *finer than* a granularity H, denoted $G \preceq H$, if for each index i, there exists an index j such that $G(i) \subseteq H(j)$. If $G \preceq H$, then H is *coarser than* G ($H \succeq G$).

For example, `business-days` is finer than `weeks`, while `business-days` does not group into `weeks`; `business-days` is finer than `years`, while `weeks` is not.

A granularity G *groups periodically into* a granularity H if:

1) $G \unlhd H$.
2) There exist $n, m \in \mathbf{Z}^+$, where n is less than the number of non-empty granules of H, such that for all $i \in \mathbf{Z}$, if $H(i) = \cup^k_{r=0} G(j_r)$ and $H(i+n) \neq \emptyset$, then $H(i+n) = \cup^k_{r=0} G(j_r + m)$.

The *groups periodically into* relationship is a special case of *groups into* characterized by a periodic repetition of the "grouping pattern" of granules of G into granules of H. Its definition may appear complicated, but it is actually quite simple. Since G groups into H, any non-empty granule $H(i)$ is the union of some granules of G; for instance, assume it is the union of the granules $G(a_1)$, $G(a_2)$,...,$G(a_k)$. The periodicity property (condition 2 in the definition) ensures that the n^{th} granule *after* $H(i)$, i.e., $H(i + n)$, if non-empty, is the union of $G(a_1 + m)$, $G(a_2 + m)$,...,$G(a_k + m)$. This results in a periodic "pattern" of the composition of n granules of H in terms of granules of G. The pattern repeats along the time domain by "shifting" each granule of H by m granules of G. Many common granularities are in this kind of relationship. For example, `days` groups periodically into `business-days`, with $m = 7$ and $n = 5$, and also groups periodically into `weeks`, with $m = 7$ and $n = 1$; `months` groups periodically into `years` with $m = 12$ and $n = 1$, and `days` groups periodically into `years` with $m = 14{,}697$ and $n = 400$. Alternatively, the relationship can also be described, by saying, for example, years is periodic (or 1-periodic) with respect to months, and years is periodic (or 400-periodic) with respect to days. In general, this relationship guarantees that granularity *H* can be finitely described in terms of granules of *G*. More details can be found in [3].

Given a granularity order relationship *g-rel* and a set of granularities, a granularity *G* in the set is a *bottom granularity* with respect to *g-rel*, if *G g-rel H* for each granularity *H* in the set.

For example, given the set of all granularities defined over the time domain ($\mathbf{R}$;$\leq$), and the granularity relationship $\preceq$ (finer than), the granularity corresponding to the *empty* mapping is the bottom granularity with respect to $\preceq$. Given the set of all granularities defined over the time domain ($\mathbf{Z}$;$\leq$), and the granularity relationship $\trianglelefteq$ (groups into), the granularity mapping each index into the corresponding instant (same integer number as the index) is a bottom granularity with respect to $\trianglelefteq$. An example of a set of granularities without a bottom (with respect to $\preceq$ or $\trianglelefteq$) is {weeks, months}.

Calendars are typically used to describe events or time-related properties over the same span of time using different granularities. For example, the Gregorian calendar comprises the granularities days, months, and years. Considering the notion of bottom granularity, a formal definition of calendar follows.

A *calendar* is a set of granularities that includes a bottom granularity with respect to $\trianglelefteq$ (groups into).

Defining New Granularities through Algebraic Operators

In principle, every granularity in a calendar can be defined in terms of the bottom granularity, possibly specifying the composition of granules through the relationships defined above. Several proposals have appeared in the literature for a set of algebraic operators with the goal of facilitating the definition of new granularities in terms of existing ones. These algebras are evaluated with respect to expressiveness, user friendliness, and ability to compute operations on granularities directly on the algebraic representation. Some of the operations that are useful in applications are inter-granule conversions; for example, to compute which day of the week was the k-th day of a particular year, or which interval of days was r-th week of that year, as well as conversions involving different calendars. In the following, the main operators of one of the most expressive calendar algebras [3] are briefly described. Two operators form the backbone of the algebra:

1. The *grouping* operator systematically combines a few granules of the source granularity into one granule in the target granularity. For example, given granularity days, granularity weeks can be generated by combining 7 granules (corresponding to Monday – Sunday) week = *Group*$_7$(day) if we assume that day(1) corresponds to Monday, i.e., the first day of a week.
2. The *altering-tick* operator deletes or adds granules from a given granularity (via the help of a second granularity) to form a new one. For example, assume each 30 days are grouped into a granule, forming a granularity 30-day-groups. Then, an extra day can be added for each January, March, and so on, while two days are dropped from February. The February in leap years can be similarly changed to have an extra day, hence properly representing month.

Other auxiliary operators include:

- *Shift* shifts the index forward or backward a few granules (e.g., the granule used to be labeled 1 may be re-labeled 10) in order to provide proper alignment for further operations.
- *Combine* combines all the granules of one granularity that fall into (using finer-than relationship) a second granularity. For example, by combining all the business days in a week, business-week is obtained.
- *Subset* generates a new granularity by selecting an interval of granules from a given granularity. For example, choosing the days between year 2000 and 2010, leads to a granularity that only contains days in these years.
- *Select* generates new granularities by selecting granules from the first operand in terms of their relationship with the granules of the second operand. For example, selecting the first day of each week gives the Mondays granularity.

More details, including other operators, conditions of applicability, as well as comparison with other algebra proposals can be found in [3]. The algebra directly supports the inter-granule and inter-calendar conversions mentioned above. Some more complex operations on granularities (e.g., temporal constraint propagation in terms of multiple granularities) and the verification of formal properties (e.g., equivalence of algebraic granularity representations) require a conversion in terms of a given bottom granularity. An efficient automatic procedure for this conversion has been devised and implemented [2]. The automaton-based approach may be a valid alternative for the verification of formal properties.

Key Applications

Temporal granularities are currently used in several applications, but in most cases their use is limited to very few standard granularities, supported by system calendar libraries and ad-hoc solutions are used to manipulate data associated with them. This approach often leads to unclear semantics, hidden mistakes, low interoperability, and does not take advantage of user-defined granularities and complex operations. The impact of a formal framework for temporal granularities has been deeply investigated for a number of application areas among which logical design of temporal databases, querying databases in terms of arbitrary granularities, data mining, integrity constraint satisfaction, workflows [3]. For example, when temporal dependencies in terms of granularities can be identified in the data (e.g., "salaries of employees do not change within a fiscal year"), specific techniques have been devised for the logical design of temporal databases that can lead to significant benefits. In query processing, it has been shown how to support the retrieval of data in terms of temporal granularities different from the ones associated to the data stored in the database, provided that assumptions on the data semantics are formalized, possibly as part of the database schema. Time distance constraints in terms of granularities (e.g., *Event2 should occur within two business days after the occurrence of Event1*) have been extensively studied, and algorithms proposed to check for consistency and solutions. Applications include the specification of classes of frequent patterns to be identified in time series, the specification of integrity contraints in databases, and the specification of constraints on activities durations and on temporal distance between specific events in workflows.

Temporal granularities also have several applications in other areas like natural language processing, temporal reasoning in AI, including scheduling and planning, and in computer logic, where they have been mainly considered for program specification and verification.

Future applications may include advanced personal information management (PIM). Time and location-aware devices coupled with advanced calendar applications, supporting user-defined granularities, may offer innovative personalized scheduling and alerting systems.

Experimental Results

Several systems and software packages dealing with temporal granularities have been developed, among which MultiCal, Calendar algebra implementation, GSTP Project, and TauZaman. Information about these systems can be easily found online.

Cross-references

- ▶ Temporal Constraints
- ▶ Temporal Data Mining
- ▶ Temporal Dependencies
- ▶ Temporal Periodicity
- ▶ Time Domain
- ▶ TSQL2
- ▶ Time Instant

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X. A glossary of time granularity concepts. In Temporal Databases: Research and Practice. O. Etzion, S. Jajodia, S. Sripada (eds.), 1998, pp. 406–413.
2. Bettini C., Mascetti S., and Wang X. Supporting temporal reasoning by mapping calendar expressions to minimal periodic sets. J. Artif. Intell. Res., 28:299–348, 2007.
3. Bettini C., Wang X.S., and Jajodia S. Time Granularities in Databases, Data Mining, and Temporal Reasoning. Springer, Berlin Heidelberg New York, 2000.
4. Chandra R., Segev A., and Stonebraker M. Implementing calendars and temporal rules in next generation databases. In Proc. 10th Int. Conf. on Data Engineering, 1994, pp. 264–273.
5. Clifford J. and Rao A. A simple, general structure for temporal domains. In Proc. IFIP TC 8/WG 8.1 Working Conf. on Temporal Aspects in Inf. Syst., 1987, pp. 23–30.
6. Combi C., Franceschet M., and Peron A. Representing and reasoning about temporal granularities. J. Logic Comput., 14(1):51–77, 2004.
7. Dal Lago U., Montanari A., and Puppis G. Compact and tractable automaton-based representations of time granularities. Theor. Comput. Sci., 373(1–2):115–141, 2007.
8. Euzenat J. and Montanari A. Time granularity. In M. Fisher, D. Gabbay, L. Vila (eds.). Handbook of Temporal Reasoning in Artificial Intelligence. Elsevier, 2005.

9. Ladkin P. The completness of a natural system for reasoning with time intervals. In Proc. 10th Int. Joint Conf. on AI, 1987, pp. 462–467.
10. Leban B., McDonald D., and Forster D. A representation for collections of temporal intervals. In Proc. 5th National Conf. on AI, 1986, pp. 367–371.
11. Lorentzos N.A. DBMS support for nonmetric measurement systems. IEEE Trans. Knowl. Data Eng., 6(6):945–953, 1994.
12. Urgun B., Dyreson C.E., Snodgrass R.T., Miller J.K., Kline N., Soo M.D., and Jensen C.S. Integrating multiple calendars using tau-ZAMAN. Software Pract. Exp., 37(3):267–308, 2007.
13. Wang X., Jajodia S., and Subrahmanian V.S. Temporal modules: an approach toward federated temporal databases. Inf. Sci., 82:103–128, 1995.
14. Weiderhold G., Jajodia S., and Litwin W. Integrating temporal data in a heterogeneous environment. In Temporal Databases: Theory, Design, and Implementation, Benjamin/Cummings, 1993, pp. 563–579.
15. Wijsen J. A string-based model for infinite granularities. In Spatial and Temporal Granularity: Papers from the AAAI Workshop. AAAI Technical Report WS-00-08, AAAI, 2000, pp. 9–16.

Temporal Homogeneity

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

Assume a temporal relation where the attribute values of tuples are (partial) functions from some time domain to value domains. A tuple in such a relation is *temporally homogeneous* if the domains of all its attribute values are identical. A temporal relation is temporally homogeneous if all its tuples are temporally homogeneous. Likewise, a temporal database is temporally homogeneous if all its relations are temporally homogeneous.

In addition to being specific to a type of object (tuple, relation, database), homogeneity is also specific to a time dimension when the time domain is multidimensional, as in "temporally homogeneous in the valid-time dimension" or "temporally homogeneous in the transaction-time dimension."

Key Points

The motivation for homogeneity arises from the fact that no timeslices of a homogeneous relation produce null values. Therefore, a homogeneous relational model is the temporal counterpart of the snapshot relational model without nulls. Certain data models assume temporal homogeneity, while other models do not.

A tuple-timestamped temporal relation may be viewed as a specific attribute-value timestamped relation. An attribute value *a* of a tuple with timestamp *t* is represented by a function that maps each value in *t* to *a*. Thus, models that employ tuple timestamping are necessarily temporally homogeneous.

Cross-references

- ▶ Lifespan
- ▶ SQL-Based Temporal Query Languages
- ▶ Temporal Database
- ▶ Time Domain
- ▶ Transaction Time
- ▶ Valid Time

Recommended Reading

1. Gadia S.K. A homogeneous relational model and query languages for temporal databases. ACM Trans. Database Syst., 13(4):418–448, December 1988.
2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.), Springer-Verlag, Berlin, 1998, pp. 367–405.

Temporal Indeterminacy

Curtis Dyreson
Utah State University, Logan, UT, USA

Synonyms

Fuzzy time; Imprecise time

Definition

Temporal indeterminacy refers to "don't know when" information, or more precisely, "don't know *exactly* when." The modifier 'temporally indeterminate' indicates that the modified object has an associated time, but that the time is not known precisely. The time when an event happens, when a time interval begins or ends, or even the duration of a period may be indeterminate. For example, the event of a car accident might be "sometime last week," the interval an airplane flight takes may be from "Friday to Saturday," or the duration a graduate student takes to write a dissertation may be "four to fifteen years."

The adjective ‘temporal’ allows parallel kinds of indeterminacy to be defined, such as spatial indeterminacy. There is a subtle difference between indeterminate and imprecise. In this context, indeterminate is a more general term than imprecise since precision is commonly associated with making measurements. Typically, a precise measurement is preferred to an imprecise one. Imprecise time measurements, however, are just one source of temporally indeterminate information.

Historical Background

Despite the wealth of research on adding incomplete information to databases, there are few efforts that address incomplete temporal information [6]. Much of the previous research in incomplete information databases has concentrated on issues related to null values, the applicability of fuzzy set theory, and the integration of various combinations of probabilistic reasoning, temporal reasoning, and planning.

In the earliest work on temporal indeterminacy, an indeterminate instant was modeled with a set of possible chronons [12]. Dutta next introduced a fuzzy set approach to handle events that can be interpreted to have multiple occurrences [5]. For example the event “Margaret’s salary is high” may occur at various times as Margaret’s salary fluctuates to reflect promotions and demotions. The meaning of “high” is incomplete, it is not a crisp predicate. In Dutta’s model all the possibilities for high are represented in a *generalized* event and the user selects some subset according to his or her interpretation of “high.” Generalized bitemporal elements were defined somewhat differently in a later paper by Kouramajian and Elmasri [11]. Bitemporal elements combine transaction time and valid time in the same temporal element, and can include a noncontiguous (i.e., indeterminate) set of noncontiguous possible times. Gadia et al. proposed an interesting model that intertwines support for value and temporal incompleteness [8]. By combining the different kinds of incomplete information, a wide spectrum of attribute values are simultaneously modeled, including values that are completely known, values that are unknown but are known to have occurred, values that are known if they occurred, and values that are unknown even if they occurred. Dyreson and Snodgrass proposed using probabilistic events to model temporal indeterminacy. In their model a time is represented as a probability distribution [7]. Probabilistic times were also comprehensively addressed by Dekhtyar et al. [4].

Reasoning with incomplete information can be computationally expensive. Koubarakis was the first to focus attention on the issue of cost [9,10]. He showed that by restricting the kinds of constraints allowed in representing the indeterminacy, polynomial time algorithms can be obtained. Koubarakis proposed a temporal data model with global and local inequality constraints on the occurrence time of an event. Another constraint-based temporal reasoner is LaTeR, which has been successfully implemented [2]. LaTeR similarly restricts the kinds of constraints allowed (to conjunctions of linear inequalities), but this class of constraints includes many of the important temporal predicates [3].

Temporal indeterminacy has also been addressed in non-relational contexts, for instance in object-oriented databases [1].

Foundations

There are (at least) three possible sources of indeterminacy in a statement with respect to time: (i) a discrepancy between the *granularity* of the temporal qualification and the occurrence time; (ii) an underspecification of the occurrence time when the granularities of the temporal qualification and the occurrence time coincide; and (iii) relative times.

As a first approximation, a statement is *temporally indeterminate* if the granularity of its reference to time (in the examples, the granularity of days) is coarser than the granularity of the time at which the denoted event(s) occur. Temporal indeterminacy as well as relativity of reference to time is mainly a qualification of a statement rather than of the event it denotes (that is, temporal indeterminacy characterizes the relationship between the granularity of the time reference of a statement and the granularity of an event’s occurrence time). It does not depend on the time at which the statement is evaluated. The crucial and critical point is the determination of the time granularity of the event occurrence time.

Generally, a statement whose reference to time has a granularity (e.g., days) which is temporally determinate with respect to every coarser granularity (e.g., months) and temporally indeterminate with respect to every finer granularity (e.g., seconds). But this general rule has exceptions since it does not take into account information about the denoted occurrence time. In particular, for a *macro-event* there exists a (finest) granularity at which its occurrence time can be specified, but with respect to finer granularities, the event as a whole does not make sense, and must, if possible, be decomposed into a set of components.

But not all cases of temporally indeterminate information involve a discrepancy between the granularity of the reference to time and the granularity of the occurrence time. Consider the sentence: "The shop remained open on a Sunday in April 1990 all day long." 'Days' is the granularity of both the time reference and the occurrence time. Nevertheless, this statement is temporally indeterminate because the precise day in which the shop remained open is unknown (it is known only that it is one of the Sundays in April 1990).

Statements that contain a relative reference to time are also temporally indeterminate, but the reverse does not hold: temporally-indeterminate statements can contain relative as well as absolute references to time. The statements "Jack was killed sometime in 1990" and "Michelle was born yesterday" contain absolute and relative references to time, respectively, but they are both temporally indeterminate.

The following example illustrates how temporal indeterminacy can be represented in a relational database. Consider the employment relation shown in Fig. 1 which is cobbled together from the (somewhat hazy) memories of several employees. Each tuple shows a worker's name, salary, department, and time employed (i.e., the valid time). The first tuple represents Joe's employment in Shoes. It is temporally indeterminate since the exact day when Joe stopped working in Shoes is not known precisely; the ending valid time is recorded as sometime in January 2005 and represented as the indeterminate event "`1 Jan 2005~31 Jan 2005`." Joe then went to work in Admin, which is also temporally indeterminate. Joe started working in Admin "After leaving Shoes" which is a temporal constraint on an indeterminate time. The third tuple represents Sue's employment history. She began working in Shoes sometime in the first half of January 2005 ("`1 Jan 2005~15 Jan 2005`") with a uniform probability for each day in that range (which is more information than is known about Joe's termination in Shoes, the probability is missing from that indeterminate event). Finally, Eve began working in Admin on some Monday in January 2005, but it is not known which Monday.

Querying temporal indeterminacy is more challenging than representing it. The two chief challenges are efficiency and expressiveness. Consider a query to find out who was employed on January 10, 2005 as expressed in a temporal version of SQL (e.g., TSQL2) below.

```
SELECT Name
FROM Employee E
WHERE VALID(E) overlaps "10 Jan 2005"
```

There are two well-defined limits on querying incomplete information: the *definite* and the *possible*. The definite answer includes only information that is known. On a tuple-by-tuple basis, determining which employment tuple definitely overlaps January 10, 2005 is straightforward: *none* definitely do. In contrast, every tuple *possibly* overlaps that day. It is efficient to compute both bounds on a tuple-by-tuple basis, but not very expressive. It would be more expressive to be able to find other answers that lie between the bounds. Probabilistic approaches seek to refine the set of potential answers by reasoning with probabilities. For instance, can it be computed who was *probably* employed (exceeding a probability of 0.5)? The probability that Sue began working before January 10, 2005 is 0.67 (the probability mass is uniformly distributed among all the possibilities). Since the probability mass function for Joe's termination in Shoes is missing, he can not be included in the employees who were probably employed.

Name	Dept.	Sal.	Valid time
Joe	Shoes	40K	[1 Jan 2003 – 1 Jan 2005~31 Jan 2005]
Joe	Admin	100K	[After leaving shoes – now]
Sue	Shoes	50K	[1 Jan 2005~15 Jan 2005 (uniform) – now]
Eve	Admin	90K	[A Monday in January 2005 – now]

Temporal Indeterminacy. Figure 1. A relation with temporal indeterminacy.

Most of the existing approaches have focused on improving the efficiency of computing probabilistic answers; the *usability* of probabilistic approaches has not yet been determined. It might be very difficult for users to interpret and use probabilities (should a user interpret "probably" as exceeding 0.75 or 0.5) so other approaches (e.g., fuzzy set approaches) may be needed to improve usability. A second research issue concerns reasoning with *intra-tuple* constraints. The definite answer given above is inaccurate. Even though it is not known exactly when Joe stopped working in Shoes or started working in Admin, it is known that he was employed in one or the other on January 10, 2005. The intra-tuple constraint in the second tuple represents this knowledge. Though intra-tuple constraints provide greater reasoning power, reasoning with them often has high computational complexity. Research continues in defining classes of constraints that are meaningful and computationally feasible. Finally, temporal indeterminacy has yet to be considered in new kinds of queries (e.g., roll-up in data warehouses and top-k queries) and new temporal query languages (e.g., τXQuery and TOWL).

Key Applications

The most common kinds of temporal indeterminacy are valid-time indeterminacy and user-defined time indeterminacy. Transaction-time indeterminacy is rarer because transaction times are always known exactly. Temporal indeterminacy can occur in logistics, especially in planning scenarios where project completion dates are typically inexact. In some scientific fields it is quite rare to know an exact time, for instance, archeology is replete with probabilistic times generated by radio-carbon dating, tree-ring analyses, and by the layering of artifacts and sediments. Indeterminacy can arise even when precise clocks are employed. In a network of road sensors, an "icy road" has an indeterminate lifetime as the change from non-icy to icy is gradual rather than instantaneous; "icy road" has a fuzzy starting and ending time.

Cross-references

- ▶ Probabilistic Temporal Databases
- ▶ Qualitative Temporal Reasoning
- ▶ Temporal Constraints
- ▶ Temporal Granularity

Recommended Reading

1. Biazzo V., Giugno R., Lukasiewicz T., and Subrahmanian V.S. Temporal probabilistic object bases. IEEE Trans. Knowl. Data Eng., 15(4):921–939, 2003.
2. Brusoni V., Console L., Terenziani P., and Pernici B. Extending temporal relational databases to deal with imprecise and qualitative temporal information. In Proc. Int. Workshop on Temporal Databases, 1995, pp. 3–22.
3. Brusoni V., Console L., Terenziani P., and Pernici B. Qualitative and quantitative temporal constraints and relational databases: theory, architecture, and applications. IEEE Trans. Knowl. Data Eng., 11(6):948–968, 1999.
4. Dekhtyar A., Ross R., and Subrahmanian V.S. Probabilistic temporal databases, I: algebra. ACM Trans. Database Syst., 26(1):41–95, 2001.
5. Dutta S. Generalized events in temporal databases. In Proc. 5th Int. Conf. on Data Engineering, 1989, pp. 118–126.
6. Dyreson C. A bibliography on uncertainty management in information systems. In Uncertainty Management in Information Systems: From Needs to Solutions. Kluwer Academic, Norwell, MA, 1997, pp. 415–458.
7. Dyreson C. and Snodgrass R.T. Supporting valid-time indeterminacy. ACM Trans. Database Syst., 23(1):1–57, 1998.
8. Gadia S.K., Nair S.S., and Poon Y.-C. Incomplete information in relational temporal databases. In Proc. 18th Int. Conf. on Very Large Data Bases, 1992, pp. 395–406.
9. Koubarakis M. Representation and querying in temporal databases: The power of temporal constraints. In Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 327–334.
10. Koubarakis M. The complexity of query evaluation in indefinite temporal constraint databases. Theor. Comput. Sci., 171(1–2):25–60, 1997.
11. Kouramajian V. and Elmasri R. A generalized temporal model. In Proc. Uncertainty in Databases and Deductive Systems Workshop, 1994.
12. Snodgrass R.T. Monitoring Distributed Systems: A Relational Approach. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1982.

Temporal Information Retrieval

▶ Time and Information Retrieval

Temporal Integrity Constraints

Jef Wijsen
University of Mons-Hainaut, Mons, Belgium

Synonyms

Dynamic integrity constraints

Definition

Temporal integrity constraints are integrity constraints formulated over temporal databases. They can express dynamic properties by referring to data valid at different time points. This is to be contrasted with databases that do not store past or future information: if integrity constraints can only refer to data valid at the current time, they can only express static properties. Languages for expressing temporal integrity constraints extend first-order logic with explicit timestamps or with temporal connectives. An important question is how to check and enforce such temporal integrity constraints efficiently.

Historical Background

The use of first-order temporal logic for expressing temporal integrity constraints dates back to the early 1980s (see for example [2]). Since the late 1980s, progress has been made in the problem of checking temporal integrity [3, 9, 11] without having to store the entire database history. This entry deals with general temporal integrity constraints.

Foundations

Integrity constraints, whether they are temporal or not, are an important component of each database schema. They express properties that, ideally, must be satisfied by the stored data at all times. If a database satisfies all the integrity constraints, it is called *consistent*. Integrity constraints are commonly expressed in a declarative way using logic. Declarative integrity constraints generally do not specify how to keep the database consistent when data are inserted, deleted, and modified. An important task is to develop efficient procedures for checking and enforcing such constraints.

Temporal databases store past, current, and future information by associating time to database facts. An integrity constraint can be called "temporal" if it is expressed over a temporal database. By relating facts valid at different points in time, temporal integrity constraints can put restrictions on how the data can change over time. This is to be contrasted with databases that do not store past or future facts: if integrity constraints can only refer to a single database state, they cannot capture time-varying properties of data.

1. Defining Temporal Integrity

While temporal integrity constraints can in principle be expressed as Boolean queries in whichever temporal query language, it turns out that temporal logic on timepoint-stamped data are the prevailing formalism for defining and studying temporal integrity.

1.1 Temporal, Transition, and Static Constraints Since first-order logic is the lingua franca for expressing non-temporal integrity constraints, it is natural to express temporal integrity constraints in temporal extensions of first-order logic. Such temporalized logics refer to time either through variables with a type of time points, or by temporal modal operators, such as:

Operator	Meaning
$\bullet p$	Property *p* was true at the previous time instant.
$\blacklozenge p$	Property *p* was true sometime in the past.
$\circ p$	Property *p* will be true at the next time instant.
$\Diamond p$	Property *p* will be true sometime in the future.

Satisfaction of such constraints by a temporal database can be checked if the question "Does (did/will) $R(a_1,...,a_m)$ hold at t?" can be answered for any fact $R(a_1,...,a_m)$ and time instant t. Alternatively, one can equip facts with time and ask: "Is $R(a_1,...,a_m \mid t)$ true?". Thus, one can abstract from the concrete temporal database representation, which may well contain interval-stamped facts [7]. Every time point t gives rise to a *(database) state* containing all facts true at t.

The following examples assume a time granularity of days. The facts *WorksFor*(John Smith, Pulse) and *Earns*(John Smith, 20K), true on 10 August 2007, express that John worked for the Pulse project and earned a salary of 20K at that date. The alternative encoding is *WorksFor*(John Smith, Pulse | 10 Aug 2007) and *Earns*(John Smith, 20K | 10 Aug 2007).

Although temporal integrity constraints can generally refer to any number of database states, many natural constraints involve only one or two states. In particular, *transition constraints* only refer to the current and the previous state; *static constraints* refer only to the current state.

The first-order temporal logic (FOTL) formulas (1)–(4) hereafter illustrate different types of integrity constraints. The constraint *"An employee who is dropped from the Pulse project cannot work for that project later on"* can be formulated in FOTL as follows:

$$\neg\exists x(\mathit{WorksFor}(x,\mathrm{Pulse}) \wedge \blacklozenge(\neg \mathit{WorksFor}(x,\mathrm{Pulse}) \wedge \bullet\, \mathit{WorksFor}(x,\mathrm{Pulse}))) \tag{1}$$

This constraint can be most easily understood by noticing that the subformula $\blacklozenge(\neg\,\mathit{WorksFor}(x,\ \mathrm{Pulse}) \wedge \bullet\, \mathit{WorksFor}(x,\ \mathrm{Pulse}))$ is true in the current database state for every employee x who was dropped from the Pulse project sometime in the past (this subformula will recur in the discussion of integrity checking later on). This constraint can be equivalently formulated in a two-sorted logic, using two temporal variables t_1 and t_2:

$$\neg\exists x\exists t_1\exists t_2((t_1 < t_2) \wedge \mathit{WorksFor}(x,\mathrm{Pulse}|t_2) \\ \wedge \neg \mathit{WorksFor}(x,\mathrm{Pulse}|t_1) \wedge \mathit{WorksFor}(x,\mathrm{Pulse}|t_1-1))$$

The constraint "*Today's salary cannot be less than yesterday's*" is a transition constraint, and can be formulated as follows:

$$\neg\exists x\exists y\exists z(\mathit{Earns}(x,y) \wedge \bullet(\mathit{Earns}(x,z) \wedge y < z)) \tag{2}$$

Finally, static constraints are illustrated. The most fundamental static constraints in the relational data model are primary keys and foreign keys. The constraint "*No employee has two salaries*" implies that employee names uniquely identify *Earns*-tuples in any database state; it corresponds to a standard primary key. The constraint "*Every employee in the WorksFor relation has a salary*" means that in any database state, the first column of *WorksFor* is a foreign key that references (the primary key of) *Earns*. These constraints can be formulated in FOTL without using temporal connectives:

$$\forall x\forall y\forall z(\mathit{Earns}(x,y) \wedge \mathit{Earns}(x,z) \rightarrow y = z) \tag{3}$$

$$\forall x\forall y(\mathit{WorksFor}(x,y) \rightarrow \exists z \mathit{Earns}(x,z)) \tag{4}$$

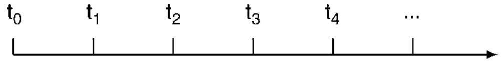
Formulas (3) and (4) show a nice thing about using FOTL for expressing temporal integrity: static constraints read as non-temporal constraints expressed in first-order logic.

1.2 Different Notions of Consistency The syntax and semantics of the temporal logic used in the previous section are defined next. Different notions of temporal constraint satisfaction are discussed.

Assume a countably infinite set **dom** of constants. In the following syntax, R is any relation name and each s_i is a variable or a constant:

$$C, C' ::= R(s_1,\ldots,s_m)|s_1 = s_2 \\ |C \wedge C'|\neg C|\exists x(C)|\circ C|\diamond C|\bullet C|\blacklozenge C$$

The connectives $\bullet$ and $\blacklozenge$ are called *past operators*; $\circ$ and $\diamond$ are *future operators*. A *past formula* is a formula without future operators; a *future formula* is a formula without past operators. Other modal operators, like the past operator **since** and the future operator **until**, can be added to increase expressiveness. The set of time points is assumed to be infinite, discrete and linearly ordered with a smallest element. Thus, the time scale can be depicted as follows:



Discreteness of time is needed in the interpretation of the operators $\circ$ and $\bullet$. Formulas are interpreted relative to an infinite sequence $H = \langle H_0, H_1, H_2, \ldots\rangle$, where each H_i is a finite set of facts formed from relation names and constants of **dom**. Intuitively, H_i contains the facts true at time t_i. Such a sequence H is called an *infinite (database) history* and each H_i a *(database) state*. The following rules define inductively what it means for a closed formula C to be *satisfied* by H, denoted $H \models_{\mathrm{inf}} C$.

$H, i \models_{\mathrm{inf}} R(a_1, \ldots, a_m)$	iff	$R(a_1, \ldots, a_m) \in H_i$
$H, i \models_{\mathrm{inf}} a_1 = a_2$	iff	$a_1 = a_2$
$H, i \models_{\mathrm{inf}} C \wedge C'$	iff	$H, i \models_{\mathrm{inf}} C$ and $H, i \models_{\mathrm{inf}} C'$
$H, i \models_{\mathrm{inf}} \neg C$	iff	not $H, i \models_{\mathrm{inf}} C$
$H, i \models_{\mathrm{inf}} \exists x(C)$	iff	$H, i \models_{\mathrm{inf}} C[x \mapsto a]$ for some $a \in$ **dom**, where $C[x \mapsto a]$ is obtained from C by replacing each free occurrence of x with a
$H, i \models_{\mathrm{inf}} \bullet C$	iff	$i > 0$ and $H, i-1 \models_{\mathrm{inf}} C$
$H, i \models_{\mathrm{inf}} \blacklozenge C$	iff	$H, j \models_{\mathrm{inf}} C$ for some j such that $0 \leq j < i$
$H, i \models_{\mathrm{inf}} \circ C$	iff	$H, i+1 \models_{\mathrm{inf}} C$
$H, i \models_{\mathrm{inf}} \Diamond C$	iff	$H, j \models_{\mathrm{inf}} C$ for some $j > i$

Note that the truth of each subformula is expressed relative to a *single* "reference" time point i (along with H). This characteristic allows efficient techniques for integrity checking [6] and seems crucial to the success of temporal logic. Finally:

$$H \models_{\mathrm{inf}} C \text{ iff } H, j \models_{\mathrm{inf}} C \text{ for each } j \geq 0$$

Consistency of infinite database histories is of theoretical interest. In practice, only a finite prefix of H will be known at any one time. Consider, for example, the situation where H_0 is the initial database state, and for each $i \geq 0$, the state H_{i+1} results from applying an update to H_i. Since every update can be followed by another one, there is no last state in this sequence. However, at any one time, only some *finite history* $\langle H_0,\ldots,H_n\rangle$ up to the most recent update is known, and it is of practical importance to detect constraint violations in such a finite history. It is reasonable to raise a constraint violation when the finite history obtained so far cannot possibly be extended to an infinite consistent history. For example, the constraints

$$\neg\exists x(\mathit{Hire}(x) \wedge \neg\Diamond(\mathit{Promote}(x) \wedge \Diamond\mathit{Retire}(x)))$$
$$\neg\exists x(\mathit{Retire}(x) \wedge \Diamond\mathit{Retire}(x))$$

express that all hired people are promoted before they retire, and that no one can retire twice. Then, the finite history $\langle\{\mathit{Hire}(\mathrm{Ed})\},\{\mathit{Hire}(\mathrm{An})\},\{\mathit{Retire}(\mathrm{Ed})\}\rangle$ is inconsistent, because of the absence of *Promote*(Ed) in the second state. It is assumed here that the database history is append-only and that the past cannot be modified.

Two different notions, denoted $\vDash_{\mathrm{pot}}$ and $\vDash_{\mathrm{fin}}$, of satisfaction for finite histories are as follows:

1. $\langle H_0,\ldots,H_n\rangle \vDash_{\mathrm{pot}} C$ if $\langle H_0,\ldots,H_n\rangle$ can be extended to an infinite history $H = \langle H_0,\ldots,H_n, H_{n+1},\ldots\rangle$ such that $H \vDash_{\mathrm{inf}} C$.
2. $\langle H_0,\ldots,H_n\rangle \vDash_{\mathrm{fin}} C$ if $\langle H_0,\ldots,H_n\rangle$ can be extended to an infinite history $H = \langle H_0,\ldots,H_n, H_{n+1},\ldots\rangle$ such that $H, i \vDash_{\mathrm{inf}} C$ for each $i \in \{0, 1,\ldots,n\}$.

Obviously, the first concept, called *potential satisfaction*, is stronger than the second one: $\langle H_0,\ldots,H_n\rangle \vDash_{\mathrm{pot}} C$ implies $\langle H_0,\ldots,H_n\rangle \vDash_{\mathrm{fin}} C$. Potential satisfaction is the more natural concept. However, Chomicki [3] shows how to construct a constraint *C*, using only past operators (● and ♦), for which $\vDash_{\mathrm{pot}}$ is undecidable. On the other hand, $\vDash_{\mathrm{fin}}$ is decidable for constraints *C* that use only past operators, because the extra states H_{n+1}, H_{n+2},... do not matter in that case. It also seems that for most practical past formulas, $\vDash_{\mathrm{pot}}$ and $\vDash_{\mathrm{fin}}$ coincide [6]. Chomicki and Niwiński [4] define restricted classes of future formulas for which $\vDash_{\mathrm{pot}}$ is decidable.

1.3 Expressiveness of Temporal Constraints The only assumption about time used in the constraints shown so far, is that the time domain is discrete and linearly ordered. Many temporal constraints that occur in practice need additional structure on the time domain, such as granularity. The constraint "*The salary of an employee cannot change within a month*" assumes a grouping of time instants into months. It can be expressed in a two-sorted temporal logic extended with a built-in predicate `month`(t_1, t_2) which is true if t_1 and t_2 belong to the same month:

$$\neg\exists x\exists y_1\exists y_2\exists t_1\exists t_2(\mathrm{month}\ (t_1, t_2) \wedge \mathit{Earns}(x, y_1|t_1)$$
$$\wedge\ \mathit{Earns}(x, y_2|t_2) \wedge y_1 \neq y_2).$$

Arithmetic on the time domain may be needed to capture constraints involving time distances, durations, and periodicity. For example, the time domain 0, 1, 2,... may be partitioned into weeks by the predicate `week`(t_1, t_2) defined by: `week`(t_1, t_2) if $t_1\backslash 7 = t_2\backslash 7$, where \ is the integer division operator. If $0 \leq t_2 - t_1 \leq 7$, then one can say that t_2 is *within a week from* t_1 (even though `week`(t_1, t_2) may not hold).

Temporal databases may provide two temporal dimensions for valid time and transaction time. Valid time, used in the preceding examples, indicates when data are true in the real world. Transaction time records the history of the database itself. If both time dimensions are supported, then constraints can explicitly refer to both the history of the domain of discourse and the system's knowledge about that history [10]. Such types of constraints cannot be expressed in formalisms with only one notion of time.

Temporal logics have been extended in different ways to increase their expressive power. Such extensions include fixpoint operators and second-order quantification over sets of timepoints. On the other hand, several important problems, such as potential constraint satisfaction, are undecidable for FOTL and have motivated the study of syntactic restrictions to achieve decidability [4].

This entry focuses on temporal integrity of databases that use (temporal extensions of) the relational data model. Other data models have also been extended to deal with temporal integrity; time-based cardinality constraints in the Entity-Relationship model are an example.

1.4 Constraints on Interval-stamped Temporal Data All constraints discussed so far make abstraction of the concrete representation of temporal data in a (relational) database. They only assume that the database can tell whether a given fact holds at a given point in time. The notion of finite database history (let alone infinite history) is an abstract concept: in practice, all information can be represented in a single database in which facts are timestamped by time intervals to indicate their period of validity. An interval-stamped relation is shown below.

Emp	*Sal*	*FromTo*
John Smith	10K	[1 May 2007, 31 Dec 2007]
John Smith	11K	[1 Jan 2008, 31 Dec 2008]

Following [10], constraints over such interval-stamped relations can be expressed in first-order logic extended with a type for time intervals and with Allen's interval relations. The following constraint, stating that "*Salaries of employees cannot decrease*," uses temporal variables i_1, i_2 that range over time intervals:

$$\forall x \forall y \forall z \forall i_1 \forall i_2 (Earns(x, y|i_1) \wedge Earns(x, z|i_2) \\ \wedge \text{before}(i_1, i_2) \Rightarrow y \leq z).$$

Nevertheless, it seems that many temporal integrity constraints can be most conveniently expressed under an abstract, point-stamped representation. This is definitely true for static integrity constraints, like primary and foreign keys. Formalisms based on interval-stamped relations may therefore provide operators like *timeslice* or *unfold* to switch to a point-stamped representation. Such "snapshotting" also underlies the *sequenced* semantics [8], which states that static constraints must hold independently at every point in time.

On the other hand, there are some constraints that concern only the concrete interval-stamped representation itself. For example, the interval-stamped relation *Earns* shown below satisfies constraint (3), but may be illegal if there is a constraint stating that temporal tuples need to be coalesced whenever possible.

Emp	*Sal*	*FromTo*
John Smith	10K	[1 May 2007, 30 Sep 2007]
John Smith	10K	[1 Oct 2007, 31 Dec 2007]
John Smith	11K	[1 Jan 2008, 31 Dec 2008]

2. Checking and Enforcing Temporal Integrity

Consider a database history to which a new state is added whenever the database is updated. Consistency is checked whenever a tentative update reaches the database. If the update would result in an inconsistent database history, it is rejected; otherwise the new database state is added to the database history. This scenario functions well if the entire database history is available for checking consistency. However, more efficient methods have been developed that allow checking temporal integrity without having to store the whole database history.

To check integrity after an update, there is generally no need to inspect the entire database history. In particular, static constraints can be checked by inspecting only the new database state; transition constraints can be checked by inspecting the previous and the new database state. Techniques for temporal integrity checking aim at reducing the amount of historical data that needs to be considered after a database update. In "history-less" constraint checking [3,9,11], all information that is needed for checking temporal integrity is stored, in a space-efficient way, in the current database state. The past states are then no longer needed for the purpose of integrity checking (though they may be needed for answering queries).

The idea is similar to Temporal Vacuuming and can be formalized as follows. For a given database schema **S**, let FIN_HISTORIES(**S**) denote the set of finite database histories over **S** and STATES(**S**) the set of states over **S**. Given a database schema **S** and a set **C** of temporal constraints, the aim is to compute a schema **T** and a computable function E : FIN_HISTORIES(**S**) → STATES(**T**), called *history encoding*, with the following properties:

- for every $H \in$ FIN_HISTORIES(**S**), the consistency of H with respect to **C** must be decidable from $E(H)$ and **C**. This can be achieved by computing a new set **C**′ of (non-temporal) first-order constraints over **T** such that for every $H \in$ FIN_HISTORIES(**S**), $H \models_{\text{temp}} \mathbf{C}$ if and only if $E(H) \models \mathbf{C}'$, where $\models_{\text{temp}}$ is the desired notion of temporal satisfaction (see the section on "Different Notions of Consistency"). Intuitively, the function E encodes, in a non-temporal database state over the schema **T**, all information needed for temporal integrity checking.
- E must allow an incremental computation when new database states are added: for every $\langle H_0,\ldots, H_n\rangle \in$ FIN_HISTORIES(**S**), the result $E(\langle H_0,\ldots, H_n\rangle)$ must be computable from $E(\langle H_0,\ldots,H_{n-1}\rangle)$ and H_n. In turn, $E(\langle H_0,\ldots,H_{n-1}\rangle)$ must be computable from $E(\langle H_0,\ldots,H_{n-2}\rangle)$ and H_{n-1}. And so on.

Formally, there must be a computable function $\Delta : \mathrm{STATES}(\mathbf{T}) \times \mathrm{STATES}(\mathbf{S}) \rightarrow \mathrm{STATES}(\mathbf{T})$ and an initial database state $J_{\mathrm{init}} \in \mathrm{STATES}(\mathbf{T})$ such that $E(\langle H_0 \rangle) = \Delta(J_{\mathrm{init}}, H_0)$ and for every $n > 0$, $E(\langle H_0,\ldots,H_n \rangle) = \Delta(E(\langle H_0,\ldots,H_{n-1} \rangle), H_n)$. The state J_{init} is needed to get the computation off the ground.

Note that the history encoding is fully determined by the quartet ($\mathbf{T}$, J_{init}, Δ,$\mathbf{C}'$), which only depends on $\mathbf{S}$ and $\mathbf{C}$ (and not on any database history).

Such history encoding was developed by Chomicki [3] for constraints expressed in Past FOTL (including the **since** modal operator). Importantly, in that encoding, the size of $E(H)$ is polynomially bounded in the number of distinct constants occurring in H, irrespective of the length of H. Chomicki's *bounded history encoding* can be illustrated by constraint (1), which states that an employee cannot work for the Pulse project if he was dropped from that project in the past. The trick is to maintain an auxiliary relation (call it *DroppedFromPulse*, part of the new schema $\mathbf{T}$) that stores names of employees who were dropped from the Pulse project in the past. Thus, *DroppedFromPulse*(x) will be true in the current state for every employee name x that satisfies $\blacklozenge(\neg \mathit{WorksFor}(x, \mathrm{Pulse}) \wedge \bullet \mathit{WorksFor}(x, \mathrm{Pulse}))$ in the current state. Then, constraint (1) can be checked by checking $\neg \exists x(\mathit{WorksFor}(x, \mathrm{Pulse}) \wedge \mathit{DroppedFromPulse}(x))$, which, syntactically, is a static constraint. Note incidentally that the label "static" is tricky here, because the constraint refers to past information stored in the current database state. Since history-less constraint checking must not rely on past database states, the auxiliary relation *DroppedFromPulse* must be maintained incrementally (the function Δ): whenever an employee named x is dropped from the Pulse project (i.e., whenever the tuple *WorksFor*(x, Pulse) is deleted), the name x must be added to the *DroppedFromPulse* relation. In this way, one remembers who has been dropped from Pulse, but forgets when.

History-less constraint checking thus reduces dynamic to static constraint checking, at the expense of storing in auxiliary relations (over the schema $\mathbf{T}$) historical data needed for checking future integrity. Whenever a new database state is created as the result of an update, the auxiliary relations are updated as needed (using the function Δ). This technique is suited for implementation in active database systems [5,11]: a tentative database update will trigger an abort if it violates consistency; otherwise the update is accepted and will trigger further updates that maintain the auxiliary relations.

The approach described above is characterized by ad hoc updates. A different approach is operational: the database can only be updated through a predefined set of update methods (also called transactions). These transactions are specified in a transaction language that provides syntax for embedding elementary updates (insertions and deletions of tuples) in program control structures. Restrictions may be imposed on the possible execution orders of these transactions. Bidoit and de Amo [1] define dynamic dependencies in a declarative way, and then investigate transaction schemes that can generate all and only the database histories that are consistent.

Although it is convenient to use an abstract temporal representation for specifying temporal constraints, consistency checks must obviously be performed on concrete representations. Techniques for checking static constraints, like primary and foreign keys, need to be revised if one moves from non-temporal to interval-timestamped relations [8]. Primary keys can be enforced on a non-temporal relation by means of a unique-index construct. On the other hand, two distinct tuples in an interval-timestamped relation can agree on the primary key without violating consistency. For example, the consistent temporal relation shown earlier contains two tuples with the same name John Smith.

Key Applications

Temporal data and integrity constraints naturally occur in many database applications. Transition constraints apply wherever the legality of new values after an update depends on the old values, which happens to be very common. History-less constraint checking seems particularly suited in applications where temporal conditions need to be checked, but where there is no need for issuing general queries against past database states. This may be the case in monitor and control applications [12].

Cross-references

- ▶ Interval-based Temporal Models
- ▶ Point-stamped Temporal Models
- ▶ Temporal Constraints

- ▶ Temporal Dependencies
- ▶ Temporal Logic in Database Query Languages
- ▶ Temporal Vacuuming

Recommended Reading

1. Bidoit N. and de Amo S. A first step towards implementing dynamic algebraic dependences. Theor. Comput. Sci., 190(2): 115–149, 1998.
2. de Castilho J.M.V., Casanova M.A., and Furtado A.L. A temporal framework for database specifications. In Proc. 8th Int. Conf. on Very Data Bases, 1982, 280–291.
3. Chomicki J. Efficient checking of temporal integrity constraints using bounded history encoding. ACM Trans. Database Syst., 20(2):149–186, 1995.
4. Chomicki J. and Niwinski D. On the feasibility of checking temporal integrity constraints. J. Comput. Syst. Sci., 51(3): 523–535, 1995.
5. Chomicki J. and Toman D. Implementing temporal integrity constraints using an active DBMS. IEEE Trans. Knowl. Data Eng., 7(4):566–582, 1995.
6. Chomicki J. and Toman D. Temporal logic in information systems. In Logics for Databases and Information Systems. J. Chomicki and G. Saake (eds.). Kluwer, Dordecht, 1998, pp. 31–70.
7. Chomicki J. and Toman D. Temporal databases. In M. Fisher, D. M. Gabbay, and L. Vila (eds.). Handbook of Temporal Reasoning in Artificial Intelligence. Elsevier Science, 2005.
8. Li W., Snodgrass R.T., Deng S., Gattu V.K., and Kasthurirangan A. Efficient sequenced integrity constraint checking. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 131–140.
9. Lipeck U.W. and Saake G. Monitoring dynamic integrity constraints based on temporal logic. Inf. Syst., 12(3):255–269, 1987.
10. Plexousakis D. Integrity constraint and rule maintenance in temporal deductive knowledge bases. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 146–157.
11. Sistla A.P. and Wolfson O. Temporal conditions and integrity constraints in active database systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 269–280.
12. Sistla A.P. and Wolfson O. Temporal triggers in active databases. IEEE Trans. Knowl. Data Eng., 7(3):471–486, 1995.

Temporal Joins

Dengfeng Gao
IBM Silicon Valley Lab, San Jose, CA, USA

Definition

A temporal join is a join operation on two temporal relations, in which each tuple has additional attributes indicating a time interval. The temporal join predicates include conventional join predicates as well as a temporal constraint that requires the overlap of the intervals of the two joined tuples. The result of a temporal join is a temporal relation.

Besides binary temporal joins that operate on two temporal relations, there are n-ary temporal joins that operate on more than two temporal relations. Besides temporal overlapping, there are other temporal conditions such as "before" and "after" [1]. This entry will concentrate on the binary temporal joins with overlapping temporal condition since most of the previous work has focused on this kind of joins.

Historical Background

In the past, temporal join operators have been defined in different temporal data models; at times the essentially same operators have even been given different names when defined in different data models. Further, the existing join algorithms have also been constructed within the contexts of different data models. Temporal join operators were first defined by Clifford and Croker [4]. Later many papers studied more temporal join operators and the evaluation algorithms. To enable the comparison of join definitions and implementations across data models, Gao et al. [7] proposed a taxonomy of temporal joins and then use this taxonomy to classify all previously defined temporal joins.

Foundations

Starting from the core set of conventional relational joins that have long been accepted as "standard" [11]: Cartesian product (whose "join predicate" is the constant expression TRUE), theta-join, equijoin, natural join, left and right outerjoin, and full outerjoin, a temporal counterpart that is a natural, temporal generalization of the set can be defined. The semantics of the temporal join operators are defined as follows.

To be specific, the definitions are based on a single data model that is used most widely in temporal data management implementations, namely the one that timestamps each tuple with an interval. Assume that the time-line is partitioned into minimal-duration intervals, termed *chronons* [5]. The intervals are denoted by inclusive starting and ending chronons.

Two temporal relational schemas, R and S, are defined as follows.

$$R = (A_1,\ldots, A_n, \mathrm{T}_s, \mathrm{T}_e)$$
$$S = (B_1,\ldots, B_m, \mathrm{T}_s, \mathrm{T}_e)$$

The A_i, $1 \leq i \leq n$, and B_i, $1 \leq i \leq m$, are the *explicit attributes* that are found in corresponding snapshot schemas, and T_s and T_e are the timestamp start and end attributes, recording when the information recorded by the explicit attributes holds (or held or will hold) true. T will be used as a shorthand for the interval $[T_s, T_e]$, and *A* and *B* will be used as a shorthand for $\{A_1,...,A_n\}$ and $\{B_1,...,B_n\}$, respectively. Also, *r* and *s* are defined to be instances of *R* and *S*, respectively.

Consider the following two temporal relations. The relations show the canonical example of employees, the departments they work for, and the managers who supervise those departments.

Employee

EmpName	Dept	T
Ron	Ship	[1,5]
George	Ship	[5,9]
Ron	Mail	[6,10]

Manages

Dept	MgrName	T
Load	Ed	[3,8]
Ship	Jim	[7,15]

Tuples in the relations represent facts about the modeled reality. For example, the first tuple in the Employee relation represents the fact that Ron worked for the Shipping department from time 1 to time 5, inclusive. Notice that none of the attributes, including the timestamp attributes T, are set-valued – the relation schemas are in 1NF.

Cartesian Product

The temporal Cartesian product is a conventional Cartesian product with a predicate on the timestamp attributes. To define it, two auxiliary definitions are needed.

First, *intersect* (*U*,*V*), where *U* and *V* are intervals, returns TRUE if there exists a chronon *t* such that $t \in U \wedge t \in V$, and FALSE otherwise. Second, *overlap* (*U*,*V*) returns the maximum interval contained in its two argument intervals. If no non-empty intervals exist, the function returns. To state this more precisely, let *first* and *last* return the smallest and largest of two argument chronons, respectively. Also let U_s and U_e denote the starting and ending chronons of *U*, and similarly for *V*.

$$overlap(U,V) = \begin{cases} [last(U_s, V_s), first(U_e, V_e)] \text{ if } last(U_s, V_s) \\ \qquad\qquad\qquad \leq first(U_e, V_e) \\ \emptyset \qquad\qquad\qquad\qquad \text{otherwise.} \end{cases}$$

The temporal Cartesian product, $r \times^{\mathrm{T}} s$, of two temporal relations *r* and *s* is defined as follows.

$$r \times^{\mathrm{T}} s = \{z^{(n+m+2)} \mid \exists x \in r\ \exists y \in s\ (intersect\ (x[\mathrm{T}], y[\mathrm{T}]) \wedge z[A] = x[A] \wedge z[B] = y[B] \wedge$$
$$z[\mathrm{T}] = overlap(x[\mathrm{T}], y[\mathrm{T}]) \wedge z[\mathrm{T}] \neq \phi)\}$$

The first line of the definition ensures that matching tuples *x* and *y* have overlapping timestamps and sets the explicit attribute values of the result tuple *z* to the concatenation of the explicit attribute values of *x* and *y*. The second line computes the timestamp of *z* and ensures that it is non-empty. The *intersect* predicate is included only for later reference – it may be omitted without changing the meaning of the definition.

Consider the query "Show the names of employees and managers where the employee worked for the company while the manager managed some department in the company." This can be satisfied using the temporal Cartesian product.

Employee × $^{\mathrm{T}}$ Manages

EmpName	Dept	Dept	MgrName	T
Ron	Ship	Load	Ed	[3,5]
George	Ship	Load	Ed	[5,8]
George	Ship	Ship	Jim	[7,9]
Ron	Mail	Load	Ed	[6,8]
Ron	Mail	Ship	Jim	[7,10]

The *overlap* function is necessary and sufficient to ensure *snapshot reducibility*, as will be discussed in detail later. Basically, the temporal Cartesian product acts as though it is a conventional Cartesian product applied independently at each point in time. When operating on interval-stamped data, this semantics corresponds to an intersection: the result will be valid during those times when contributing tuples from *both* input relations are valid.

Theta-Join

Like the conventional theta-join, the temporal theta-join supports an unrestricted predicate *P* on the explicit attributes of its input arguments. The temporal theta-join, $r \bowtie^{\mathrm{T}}_{\mathrm{P}} s$, of two relations *r* and *s* selects those tuples from $r \times^{\mathrm{T}} s$ that satisfy predicate $P(r[A],s[B])$. Let σ denote the standard selection operator.

The temporal theta-join, $r \bowtie^{\mathrm{T}}_{\mathrm{P}} s$, of two temporal relations *r* and *s* is defined as follows.

$$r \bowtie^{\mathrm{T}}_{\mathrm{P}} s = \sigma_{P(r[A],s[B])}(r \times^{\mathrm{T}} s)$$

Equijoin

Like snapshot equijoin, the temporal equijoin operator enforces equality matching between specified subsets of the explicit attributes of the input relations.

The temporal equijoin on two temporal relations *r* and *s* on attributes $A' \subseteq A$ and $B' \subseteq B$ is defined as the theta-join with predicate $P \equiv r[A'] = s[B']$:

$$r \bowtie^{\mathrm{T}}_{r[A']=s[B']} s \ .$$

Natural Join

The temporal natural join bears the same relationship to the temporal equijoin as does their snapshot counterparts. Namely, the temporal natural join is simply a temporal equijoin on identically named explicit attributes, followed by a subsequent projection operation.

To define this join, the relation schemas are augmented with explicit join attributes, C_i, $1 \leq i \leq k$, which are abbreviated by *C*.

$$R = (A_1,..,A_n, C_1,..,C_k, \mathrm{T}_s, \mathrm{T}_e)$$
$$S = (B_1,...,B_m, C_1,...,C_k, \mathrm{T}_s, \mathrm{T}_e)$$

The temporal natural join of *r* and *s*, $r \bowtie^{\mathrm{T}} s$, is defined as follows.

$$\begin{aligned}&r \bowtie^{\mathrm{T}} s = \{z^{(n+m+k+2)} \mid \exists x \in r \exists y \in s(x[C] = y[C] \wedge\\ &z[A] = x[A] \wedge z[B] = x[B] \wedge z[C] = y[C] \wedge\\ &z[\mathrm{T}] = overlap(x[\mathrm{T}],y[\mathrm{T}]) \wedge z[\mathrm{T}] \neq \phi)\}\end{aligned}$$

The first two lines ensure that tuples *x* and *y* agree on the values of the join attributes *C* and set the explicit attribute of the result tuple *z* to the concatenation of the non-join attributes *A* and *B* and a single copy of the join attributes, *C*. The third line computes the timestamp of *z* as the overlap of the timestamps of *x* and *y*, and ensures that *x*[T] and *y*[T] actually overlap.

The temporal natural join plays the same important role in reconstructing normalized temporal relations as does the snapshot natural join for normalized snapshot relations [10]. Most previous work in temporal join evaluation has addressed, either implicitly or explicitly, the implementation of the temporal natural join (or the closely related temporal equijoin).

Outerjoins and Outer Cartesian Products

Like the snapshot outerjoin, temporal outerjoins and Cartesian products retain *dangling tuples*, i.e., tuples that do not participate in the join. However, in a temporal database, a tuple may dangle over a portion of its time interval and be covered over others; this situation must be accounted for in a temporal outerjoin or Cartesian product.

The temporal outerjoin may be defined as the union of two subjoins, analogous to the snapshot outerjoin. The two subjoins are the temporal left outerjoin and the temporal right outerjoin. As the left and right outerjoins are symmetric, only the left outerjoin is defined here.

Two auxiliary functions are needed. The *coalesce* function collapses value-equivalent tuples – tuples with mutually equal non-timestamp attribute values [9] – in a temporal relation into a single tuple with the same non-timestamp attribute values and a timestamp that is the finite union of intervals that precisely contains the chronons in the timestamps of the value-equivalent tuples. *(Finite unions of time intervals are termed *temporal elements* [6].)* The definition of *coalesce* uses the function *chronons* that returns the set of chronons contained in the argument interval.

$$\begin{aligned}&coalesce(r) = \{z^{(n+2)} \mid \exists x \in r(z[A] = x[A] \Rightarrow chronons(x[\mathrm{T}]) \subseteq z[\mathrm{T}] \wedge\\ &\forall x'' \in r\ (x[A] = x''[A] \Rightarrow (chronons(x''[\mathrm{T}]) \subseteq z[\mathrm{T}]))) \wedge\\ &\forall t \in z[\mathrm{T}] \exists x'' \in r(z[A] = x''[A] \wedge t \in chronons(x''[\mathrm{T}]))\}\end{aligned}$$

The first two lines of the definition coalesce all value-equivalent tuples in relation *r*. The third line ensures that no spurious chronons are generated.

Now a function *expand* is defined that returns the set of maximal intervals contained in an argument temporal element, T. Prior to defining *expand* an auxiliary function *intervals* is defined that returns the set of intervals contained in an argument temporal element.

$$intervals(T) = \{[t_s, t_e] | t_s \in T \wedge t_e \in T \wedge$$
$$\forall t \in chronons([t_s, t_e])(t \in T)\}$$

The first two conditions ensures that the beginning and ending chronons of the interval are elements of T. The third condition ensures that the interval is contiguous within T.

Using *intervals*, *expand* is defined as follows.

$$expand(T) = \{[t_s, t_e] | [t_s, t_e] \in intervals(T) \wedge$$
$$\neg \exists [t_s', t_e'] \in intervals(T)(chronons([t_s, t_e]) \subset chronons([t_s', t_e']))\}$$

The first line ensures that a member of the result is an interval contained in T. The second line ensures that the interval is indeed maximal.

The temporal left outerjoin is now ready to be defined. Let R and S be defined as for the temporal equijoin. $A' \subseteq A$ and $B' \subseteq B$ are used as the explicit join attributes.

The temporal left outerjoin, $r \quad_{r[A']=s[B']} s$ of two temporal relations r and s is defined as follows.

$$r \quad_{r[A']=s[B']} s = \{z^{(n+m+2)} | \exists x \in coalesce(r) \exists y \in coalesce(s)$$
$$(x[A'] = y[B'] \wedge z[A] = x[A] \wedge z[\mathrm{T}] \neq \phi \wedge$$
$$((z[B] = y[B] \wedge z[\mathrm{T}] \in \{expand(x[\mathrm{T}] \cap y[\mathrm{T}])\}) \vee$$
$$(z[B] = \mathrm{null} \wedge z[\mathrm{T}] \in \{expand(x[\mathrm{T}]) - expand(y[\mathrm{T}])\}))) \vee$$
$$\exists x \in coalesce(r) \forall y \in coalesce(s)$$
$$(x[A'] \neq y[B'] \Rightarrow z[A] = x[A] \wedge z[B] = \mathrm{null} \wedge$$
$$z[\mathrm{T}] \in expand(x[T]) \wedge z[\mathrm{T}] \neq \phi)\}$$

The first four lines of the definition handle the case where, for a tuple x deriving from the left argument, a tuple y with matching explicit join attribute values is found. For those time intervals of x that are not shared with y, tuples with null values in the attributes of y are generated. The final three lines of the definition handle the case where no matching tuple y is found. Tuples with null values in the attributes of y are generated.

The temporal outerjoin may be defined as simply the union of the temporal left and the temporal right outerjoins (the union operator eliminates the duplicate equijoin tuples). Similarly, a temporal outer Cartesian product is a temporal outerjoin without the equijoin condition ($A' = B' = \phi$).

Temporal Joins. Table 1. Temporal join operators

Operator	Initial Citation	Taxonomy Operator	Restrictions
Θ-JOIN	[2]	Theta-join	None
EQUIJOIN	[2]	Equijoin	None
NATURAL-JOIN	[2]	Natural Join	None
TIME-JOIN	[2]	Cartesian Product	1
T-join	[8]	Cartesian Product	None
Cartesian product	[3]	Outer Cartesian Product	None
TE-JOIN	[13]	Equijoin	2
TE-OUTERJOIN	[13]	Left Outerjoin	2
EVENT-JOIN	[13]	Outerjoin	2
Valid-Time Theta-Join	[14]	Theta-join	None
Valid-Time Left Join	[14]	Left Outerjoin	None
GTE-Join	[15]	Equijoin	2, 3

Restrictions:
1 = restricts also the valid time of the result tuples
2 = matching only on surrogate attributes
3 = includes also intersection predicates with an argument surrogate range and a time range

Table 1 summarizes how previous work is represented in the taxonomy. For each operator defined in previous work, the table lists the defining publication, researchers, the corresponding taxonomy operator, and any restrictions assumed by the original operators.

In early work, Clifford [4] indicated that an INTERSECTION-JOIN should be defined that represents the categorized non-outer joins and Cartesian products, and he proposed that an UNION-JOIN be defined for the outer variants.

Reducibility

The following shows how the temporal operators reduce to snapshot operators. Reducibility guarantees that the semantics of snapshot operator is preserved in its more complex, temporal counterpart.

For example, the semantics of the temporal natural join reduces to the semantics of the snapshot natural join in that the result of first joining two temporal

$$\begin{tikzcd}
\underline{\text{Temporal relations}} & & \underline{\text{Snapshot relations}} \\
r,\, r' \arrow[rr, "\tau_t^{\mathrm{T}}"] \arrow[d, "\bowtie^{\mathrm{T}}"'] & & \tau_t^{\mathrm{T}}(r),\, \tau_t^{\mathrm{T}}(r') \arrow[d, "\bowtie"] \\
r \bowtie^{\mathrm{T}} r' \arrow[r, "\tau_t^{\mathrm{T}}"] & \tau_t^{\mathrm{T}}(r \bowtie^{\mathrm{T}} r') = & \tau_t^{\mathrm{T}}(r) \bowtie \tau_t^{\mathrm{T}}(r')
\end{tikzcd}$$

Temporal Joins. Figure 1. Reducibility of temporal natural join to snapshot natural join.

relations and then transforming the result to a snapshot relation yields a result that is the same as that obtained by first transforming the arguments to snapshot relations and then joining the snapshot relations. This commutativity diagram is shown in Fig. 1 and stated formally in the first equality of the following theorem.

The timeslice operation τ^{T} takes a temporal relation r as argument and a chronon t as parameter. It returns the corresponding snapshot relation, i.e., with the schema of r, but without the timestamp attributes, that contains (the non-timestamp portion of) all tuples x from r for which t belongs to $x[\mathrm{T}]$. It follows from the next theorem that the temporal joins defined here reduce to their snapshot counterparts.

Theorem 1

Let t denote a chronon and let r and s be relation instances of the proper types for the operators they are applied to. Then the following hold for all t:

$$\begin{aligned}
\tau_{\mathrm{t}}^{\mathrm{T}}(r \bowtie^{\mathrm{T}} s) &= \tau_{\mathrm{t}}^{\mathrm{T}}(r) \bowtie \tau_{\mathrm{t}}^{\mathrm{T}}(s) \\
\tau_{\mathrm{t}}^{\mathrm{T}}(r \times^{\mathrm{T}} s) &= \tau_{\mathrm{t}}^{\mathrm{T}}(r) \times \tau_{\mathrm{t}}^{\mathrm{T}}(s) \\
\tau_{\mathrm{t}}^{\mathrm{T}}(r \bowtie_{\mathrm{P}}^{\mathrm{T}} s) &= \tau_{\mathrm{t}}^{\mathrm{T}}(r) \bowtie_P \tau_{\mathrm{t}}^{\mathrm{T}}(s) \\
\tau_{\mathrm{t}}^{\mathrm{T}}(r \mathbin{⟕}^{\mathrm{T}} s) &= \tau_{\mathrm{t}}^{\mathrm{T}}(r) \mathbin{⟕} \tau_{\mathrm{t}}^{\mathrm{T}}(s) \\
\tau_{\mathrm{t}}^{\mathrm{T}}(r \mathbin{⟖}^{\mathrm{T}} s) &= \tau_{\mathrm{t}}^{\mathrm{T}}(r) \mathbin{⟖} \tau_{\mathrm{t}}^{\mathrm{T}}(s)
\end{aligned}$$

Due to the space limit, the proof of this theorem is not provided here. The details can be found in the related paper [7].

Evaluation Algorithms Algorithms for temporal join evaluation are necessarily more complex than their snapshot counterparts. Whereas snapshot evaluation algorithms match input tuples on their explicit join attributes, temporal join evaluation algorithms typically must in addition ensure that temporal restrictions are met. Furthermore, this problem is exacerbated in two ways. Timestamps are typically complex data types, e.g., intervals, requiring inequality predicates, which conventional query processors are not optimized to handle. Also, a temporal database is usually larger than a corresponding snapshot database due to the versioning of tuples.

There are two categories of evaluation algorithms. Index-based algorithms use an auxiliary access path, i.e., a data structure that identifies tuples or their locations using a join-attribute value. Non-index-based algorithms do not employ auxiliary access paths. The large number of temporal indexes have been proposed in the literature [12]. Gao et al. [7] provided a taxonomy of non-index-based temporal join algorithms.

Key Applications

Temporal joins are used to model relationships between temporal relations with respect to the temporal dimensions. Data warehouses usually need to store and analyze historical data. Temporal joins can be used (alone or together with other temporal relational operators) to perform the analysis on historical data.

Cross-references

- ▶ Bi-Temporal Indexing
- ▶ Temporal Algebras
- ▶ Temporal Data Models
- ▶ Temporal Database
- ▶ Temporal Query Processing

Recommended Reading

1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832–843, November 1983.

2. Clifford J. and Croker A. The historical relational data model (HRDM) and algebra based on lifespans. In Proc. 3th Int. Conf. on Data Engineering, 1987, pp. 528–537.
3. Clifford J. and Croker A. The historical relational data model (HRDM) revisited. In Temporal Databases: Theory, Design, and Implementation, Chap. 1, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R.T. Snodgrass (eds.). Benjamin/Cummings, 1993, pp. 6–27.
4. Clifford J. and Tansel A.U. On an algebra for historical relational databases: two views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1985, pp. 1–8.
5. Dyreson C.E. and Snodgrass R.T. Timestamp semantics and representation. Inf. Syst., 18(3):143–166, 1993.
6. Gadia S.K. A homogeneous relational model and query languages for temporal databases. ACM Trans. Database Syst., 13(4):418–448, 1988.
7. Gao D., Snodgrass R.T., Jensen C.S., and Soo M.D. Join operations in temporal databases. VLDB J., 14(1):2–29, 2005.
8. Gunadhi H. and Segev A. Query processing algorithms for temporal intersection joins. In Proc. 7th Int. Conf. on Data Engineering, 1991, pp. 336–3.
9. Jensen C.S. (ed.) The consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice. O. Etzion, S. Jajodia, S. Sripadi (eds.). Springer, Berlin Heidelberg New York, 1998, pp. 367–405.
10. Jensen C.S., Snodgrass R.T., and Soo M.D. Extending existing dependency theory to temporal databases. IEEE Trans. Knowl. Data Eng., 8(4):563–582, August 1996.
11. Mishra P. and Eich M. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, March 1992.
12. Salzberg B. and Tsotras V.J. Comparison of access methods for time-evolving data. ACM Comput. Surv., 31(2):158–221, June 1999.
13. Segev A. and Gunadhi H. Event-join optimization in temporal relational databases. In Proc. 15th Int. Conf. on Very Large Data Bases, 1989, pp. 205–215.
14. Soo M.D., Jensen C.S., and Snodgrass R.T. *1*An algebra for TSQL2. In the TSQL2 Temporal Query Language, Chap. 27, R.T. Snodgrass (ed.). Kluwer, Hingham, MA, 1995, pp. 505–546.
15. Zhang D., Tsotras V.J., and Seeger B. Efficient temporal join processing using indices. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 103.

Temporal Layer

▶ Temporal Strata

Temporal Logic

▶ Time in Philosophical Logic

Temporal Logic in Database Query Languages

Jan Chomicki[1], David Toman[2]

[1]State University of New York at Buffalo, Buffalo, NY, USA

[2]University of Waterloo, Waterloo, ON, Canada

Definition

The term “temporal logic” is used in the area of formal logic, to describe systems for representing and reasoning about propositions and predicates whose truth depends on time. These systems are developed around a set of *temporal connectives*, such as *sometime in the future* or *until*, that provide implicit references to time instants. First-order temporal logic is a variant of temporal logic that allows first-order predicate (relational) symbols, variables and quantifiers, in addition to temporal connectives. This logic can be used as a natural *temporal query language* for point-stamped temporal databases. A query (a temporal logic formula) is evaluated with respect to an *evaluation point (time instant)*. Each such point determines a specific database snapshot that can be viewed as a relational database. Thus, the evaluation of temporal logic queries resembles the evaluation of first-order (relational calculus) queries equipped with an additional capability to “move” the evaluation point using *temporal connectives*. In this way, it becomes possible to refer in a single query to multiple snapshots of a given temporal database. The answer to a temporal logic query evaluated with respect to all time instants forms a point-stamped temporal relation.

Historical Background

Temporal logic was developed (originally under the name of *tense logic* by Arthur Prior in the late 1950s), for representing and reasoning about natural languages. It was introduced to computer science by Amir Pnueli [8] as a tool for formal verification of software systems. The first proposal for using a temporal logic in the database context was by Sernadas [9]. Subsequently, de Castilho et al. [3] initiated the study of temporal-logic integrity constraints in relational databases. Tuzhilin and Clifford [12] considered temporal logic as a query language for temporal databases.

Foundations

Temporal Logic is a variant of *modal* logic, tailored to expressing statements whose truth is relative to an underlying time domain which is commonly a *linearly ordered* set of time instants. The modalities are expressed using natural-language statements of the form *sometime in the future*, *always in the future*, etc. and are captured in the syntax of the logic using *temporal connectives*.

Temporal logics are usually rooted in *propositional logic*. However, for the purposes of querying (single-dimensional, valid time) point-stamped temporal databases, *linear-time first-order temporal logic (FOTL)*, an extension of first-order logic (relational calculus) with *temporal connectives*, is used. More formally, given a relational schema ρ (of a snapshot of a point-stamped temporal database), the syntax of FOTL queries is defined as follows:

$$Q ::= r(x_{i_1},\ldots,x_{i_k}) \mid x_i = x_j \mid Q \wedge Q \mid \neg Q \mid \exists x.Q \mid Q \textbf{ since } Q \mid Q \textbf{ until } Q$$

for $r \in \rho$. The additional **since** and **until** connectives are the *temporal connectives*. The connectives completely *encapsulate* the structure of time: FOTL only refers to time *implicitly* using these connectives. In contrast, temporal relational calculus (TRC) uses *explicit* temporal variables and attributes to refer to time. Note that there are no restrictions on the nesting of the temporal connectives, the Boolean connectives, and the quantifiers.

The meaning of all temporal logic formulas is defined relative to a particular time instant called the *evaluation point*. Intuitively, the evaluation point can be viewed as representing the *current instant* or *now*. The standard first-order parts of FOTL formulas are then evaluated in the snapshot of the temporal database determined by the evaluation point. The temporal connectives make it possible to change the evaluation point, i.e., to "move" it to the past or to the future. In this way the combination of first-order constructs and temporal connectives allows to ask queries that refer to multiple snapshots of the temporal database. For example the query Q_1 **since** Q_2 asks for all answers that make Q_2 true sometime in the past and Q_1 true between then and now. Similarly, queries about the future are formulated using the **until** connective. More formally, answers to FOTL queries are defined using a *satisfaction relation* that, for a given FOTL query, links a temporal database and an evaluation point (time instant) with the valuations that make the given query true with respect to the snapshot of that database at that particular evaluation point. This definition, given below, extends the standard definition of satisfaction for first-order logic.

Definition [FOTL Semantics] Let *DB* be a point-stamped temporal database with a data domain D, a point-based time domain T_P, and a (snapshot) schema ρ.

The *satisfaction relation* $DB, \theta, t \models Q$, where Q is an FOTL formula, θ a valuation, and $t \in \mathrm{T}_P$, is defined inductively with respect to the structure of the formula Q: as shown in Figure 2. where $\mathbf{r}_j^{DB(t)}$ is the instance of the relation r_j in the snapshot $DB(t)$ of the database DB at the instant t.

The answer to an FOTL query Q over DB is the set of tuples:

$$Q(DB) := \{(t, \theta(x_1),\ldots,\theta(x_k)) : DB, \theta, t \models Q\},$$

where $x_1,\ldots,x_k$ are the free variables of Q.

Note that answers to FOTL queries are (valid-time) point-stamped temporal relations.

Other commonly used temporal connectives, such as $\diamond$ (*sometime in the future*), $\square$ (*always in the future*), $\blacklozenge$ (*sometime in the past*), and $\blacksquare$ (*always in the past*) can be defined in terms of **since** and **until** as follows:

$$\diamond X_1 := \text{true } \textbf{until } X_1 \qquad \blacklozenge X_1 := \text{true } \textbf{since } X_1$$
$$\square X_1 := \neg\diamond\neg X_1 \qquad \blacksquare X_1 := \neg\blacklozenge\neg X_1$$

For a discrete linear order, the $\circ$ (*next*) and $\bullet$ (*previous*) operators are defined as follows:

$$\circ X_1 := \text{false } \textbf{until } X_1 \qquad \bullet X_1 := \text{false } \textbf{since } X_1$$

The connectives **since**, $\blacklozenge$, $\blacksquare$, and $\bullet$ are called *past temporal connectives* (as they refer to the past) and **until**, $\diamond$, $\square$, and $\circ$ *future temporal connectives*.

Example. The sensor information about who enters or exits a room is kept in the relations *Entry* (Fig. 1a) and *Exit* (Fig. 1b). Consider the query Q_a: "*For every time instant, who is in the room Bell 224 at that instant?*" It can be written in temporal logic as:

$$\exists r.\ ((\neg Exit(r,p)) \textbf{ since } Entry(r,p)) \wedge r = \text{"Bell 224"}$$

The answer to Q_a in the given database is presented in Fig. 1c, under the assumption that time instants correspond to minutes.

a

Entry		
Time	Room	Person
8 : 30	Bell 224	John
9 : 00	Bell 224	Mary
11 : 00	Capen 10	John

b

Exit		
Time	Room	Person
10 : 30	Bell 224	John
10 : 00	Bell 224	Mary
12 : 00	Capen 10	John

c

Time	Person
8 : 31	John
...	...
10 : 30	John
9 : 01	Mary
...	...
10 : 00	Mary

Temporal Logic in Database Query Languages. Figure 1. The *Entry*, the *Exit*, and the *Who is in Bell 224* relations.

$DB, \theta, t \models r_j(x_{i1},\ldots,x_{ik})$	if $(\theta(x_{i1}),\ldots,\theta(x_{ik})) \in r_j^{DB(t)}$
$DB, \theta, t \models x_i = x_j$	if $\theta(x_i) = \theta(x_j)$
$DB, \theta, t \models Q_1 \wedge Q_2$	if $DB, \theta, t \models Q_1$ and $DB, \theta, t \models Q_2$
$DB, \theta, t \models \neg Q_1$	if not $DB, \theta, t \models Q_1$
$DB, \theta, t \models \exists x_i.Q_1$	if there is a $\in D$ *such that* $DB, \theta[x_i \mapsto a], t \models Q_1$
$DB, \theta, t \models Q_1$ **since** Q_2	if $\exists t_2.t_2 < t$ and $DB, \theta, t_2 \models Q_2$ and $(\forall t_1.t_2 < t_1 < t$ implies $DB, \theta, t_1 \models Q_1)$
$DB, \theta, t \models Q_1$ **until** Q_2	if $\exists t_2.t_2 > t$ and $DB, \theta, t_2 \models Q_2$ and $(\forall t_1.t_2 > t_1 > t$ implies $DB, \theta, t_1 \models Q_1)$

Temporal Logic in Database Query Languages. Figure 2. The satisfaction relation for FOTL.

Extensions

Several natural extensions of FOTL are obtained by modifying various components of the language:

Multiple temporal contexts (multi-dimensional TLs). Standard temporal logics use a single *evaluation point* to relativize the truth of formulas with respect to time. However, there is no principled reason to not use more than one evaluation point simultaneously. The logics taking this path are called *multidimensional temporal logics* or, more precisely, *n-dimensional* temporal logics (for $n \geq 1$). The satisfaction relation is extended, for a particular n, in a natural way, as follows:

$$DB, \theta, t_1,\ldots,t_n \models Q$$

where $t_1,\ldots,t_n$ are the evaluation points. In a similar fashion the definitions of *temporal connectives* are extended to this setting. Two-dimensional connectives, for example, seem to be the natural basis for temporal logic-style query language for the bitemporal data model. Unfortunately, there is no consensus on the selection of two-dimensional temporal operators.

An interesting variant of this extension are *interval temporal logics* that associate truth with *intervals* – these, however, can be considered *pairs* of time points [5,13].

More complex time domains. While most temporal logics assume that the time domain is equipped with a linear order only, in many practical settings the time domain has additional structure. For example, there may be a way to refer to *duration* (the distance of two time points). The linear-order temporal connectives are then generalized to *metric temporal connectives*:

$$DB, \theta, t \models Q_1 \ \mathbf{since}_{\sim m}\ Q_2 \quad \text{if } \exists \mathrm{t}_2.\mathrm{t} - \mathrm{t}_2 \sim \mathrm{m}$$
and
$$DB, \theta, t_2 \models Q_2$$
and
$$(\forall t_1.t_2 < t_1 < t \text{ implies } DB, \theta, t_1 \models Q_1)$$

$$DB, \theta, t \models Q_1 \ \mathbf{until}_{\sim m}\ Q_2 \text{ if } \exists \mathrm{t}_2.\mathrm{t}_2 - \mathrm{t} \sim \mathrm{m}$$
and
$$DB, \theta, t_2 \models Q_2$$
and
$$(\forall t_1.t_2 < t_1 < t \text{ implies } DB, \theta, t_1 \models Q_1)$$

for $\sim \in \{<, \leq, =, \geq, >\}$. Intuitively, these connectives provide means of placing constraints on how far in the past/future certain subformulas must be true. The resulting logic is then the *Metric First-order Temporal Logic*, a first-order variant of *Metric Temporal Logic* [7].

Example. To demonstrate the expressive power of Metric Temporal Logic, consider the query Q_b: "*For every time instant, who has been in the room Bell 224 at that instant for at least 2 hours?*":

$$\exists r.\ ((\neg Exit(r,p))\ \mathbf{since}_{\geq 2:00}\ Entry(r,p)) \wedge r = \text{"Bell 224"}$$

More powerful languages for defining temporal connectives. Another extension introduces a more powerful *language* for specifying temporal connectives over the underlying linearly-ordered time domain. Vardi and Wolper show that temporal logics with connectives defined using first-order formulas cannot express various natural conditions such as "every other day". To remedy this shortcoming, they propose temporal connectives defined with the help of *regular expressions* (ETL [15]) or *fixpoints* (temporal μ-calculus [14]). Such extensions carry over straightforwardly to the first-order setting.

More complex underlying query languages. Last, instead of relational calculus, temporal connectives can be added to a more powerful language, such as Datalog. The resulting language is called Templog [2]. With suitable restrictions, query evaluation in this language is decidable and the language itself is equivalent to Datalog$_{1S}$ [2].

Expressive Power

The **since** and **until** temporal connectives can be equivalently defined using *formulas* in the underlying theory of linear order as follows:

$$\begin{aligned} X_1 \textbf{ since } X_2 \quad &:= \exists t_2.t_0 > t_2 \wedge X_2 \wedge \forall t_1 \\ &\quad (t_0 > t_1 > t_2 \rightarrow X_1) \\ X_1 \textbf{ until } X_2 \quad &:= \exists t_2.t_0 < t_2 \wedge X_2 \wedge \forall t_1 \\ &\quad (t_0 < t_1 < t_2 \rightarrow X_1) \end{aligned}$$

where X_1 and X_2 are *placeholders* that will be substituted with other formulas to be evaluated at the time instants t_1 and t_2, respectively. This observation indicates that every FOTL query can be equivalently expressed in TRC. The explicit translation parallels the inductive definition of FOTL satisfaction, uniformly parameterizing the formulas by t_0. In this way, an atomic formula $r(x_1,\ldots,x_k)$ (where r is a non-temporal snapshot relation) becomes $R(t_0, x_1,\ldots,x_k)$ (where R is a point-timestamped relation), and so on. For a particular t_0, evaluating $r(x_1,\ldots,x_k)$ in $DB(t_0)$, the snapshot of the database DB at t_0, yields exactly the same valuations as evaluating $R(t_0, x_1,\ldots,x_k)$ in DB. The embedding of the temporal connectives uses the definitions above. For example, the embedding of the **since** connective looks as follows:

$$\begin{aligned} &\text{Embed}(Q_1 \textbf{ since } Q_2) = \exists t_2(t_0 > t_2 \wedge \\ &\quad \forall t_0(t_2 = t_0 \rightarrow \text{Embed}(Q_2)) \wedge \\ &\quad \forall t_1(t_0 > t_1 > t_2 \rightarrow \forall t_0(t_1 = t_0 \rightarrow \text{Embed}(Q_1)))). \end{aligned}$$

Note that t_0 is the only free variable in Embed(Q_1), Embed(Q_2), and Embed(Q_1 **since** Q_2). Thus, in addition to applying the (first-order) definition of the temporal connective, the embedding performs an additional *renaming* of the temporal variable denoting the evaluation point for the subformulas, because the only free variable outside of the expanded temporal connectives must be called t_0.

Additional temporal connectives can be defined using the same approach, as formulas in the underlying theory of the temporal domain. However, for linear orders – the most common choice for such a theory – Kamp [6] has shown that all the connectives that are first-order definable in terms of linear order can be readily formulated using **since** and **until**.

In addition, it is easy to see on the basis of the above embedding that all FOTL queries (and their subqueries) define point-stamped temporal relations. This *closure property* makes FOTL amenable to specifying operators for temporal relational algebra(s) over the point-stamped temporal model. On the other hand, many, if not most, other temporal query languages, in particular various temporal extensions of SQL, are based on TRC and use temporal variables and attributes to explicitly access timestamps. These languages do not share the above closure property. Surprisingly, and in contrast to the propositional setting, one can prove that query languages based on FOTL are strictly weaker than query languages based on TRC [1,11]. The query

SNAPSHOT EQUALITY: "are there two distinct time instants at which a unary relation R contains exactly the same values?"

cannot be expressed in FOTL. On the other hand, SNAPSHOT EQUALITY can be easily expressed in TRC as follows:

$$\exists t_1, t_2.t_1 < t_2 \wedge \forall x.R(t_1, x) \;\Leftrightarrow\; R(t_2, x).$$

Intuitively, the subformula "$\forall x.R(t_1, x) \;\Leftrightarrow\; R(t_2, x)$" in the above query requires the simultaneous use of *two* distinct evaluation points t_1 and t_2 in the scope of a universal quantifier, which is not possible in FOTL. The result can be rephrased by saying that FOTL and other temporal query languages closed over the point-stamped temporal relations fail to achieve (the temporal variant of) Codd's completeness. In particular, there cannot be a temporal relational algebra over the (single-dimensional) point-stamped temporal model that can express all TRC queries.

In addition, this weakness is inherent to other languages with *implicit access* to timestamps, provided the underlying time domain is a linear order. In particular:

1. Adding a finite number of additional temporal connectives defined in the theory of linear order (including constants) is not sufficient for expressing SNAPSHOT EQUALITY [11];
2. Introducing *multidimensional temporal connectives* [4], while sufficient to express SNAPSHOT EQUALITY, is still not sufficient to express all TRC queries [10]. This also means that in the bitemporal model, the associated query languages cannot simultaneously preserve closure with respect to bitemporal relations and be expressively equivalent to TRC;
3. Using temporal connectives that are defined by fixpoints and/or regular expressions (see the earlier discussion) is also insufficient to express SNAPSHOT EQUALITY. Due to their non-first-order nature, the resulting query language(s) are incomparable, in terms of their expressive power, to TRC [1].

The only currently known way of achieving first-order completeness is based on using temporal connectives defined over a time domain whose structure allows the definition of pairing and projections (e.g., integer arithmetic). In this way temporal connectives can use pairing to simulate an unbounded number of variables and in turn the full TRC. However, such a solution is not very appealing, as the timestamps in the intermediate temporal relations do not represent time instants, but rather (encoded) tuples of such instants.

Key Applications

The main application area of FOTL is in the area of temporal integrity constraints. It is based on the observation that a sequence of relational database states resulting from updates may be viewed as a snapshot temporal database and constrained using Boolean FOTL formulas. Being able to refer to the past states of the database, temporal logic constraints generalize dynamic constraints. Temporal logic is also influential in the area of design and analysis of temporal query languages such as temporal relational algebras.

Cross-references

▶ Bitemporal Data Model
▶ Datalog
▶ Point-Stamped Temporal Models
▶ Relational Calculus
▶ Temporal Algebras
▶ Temporal Integrity Constraints
▶ Temporal Query Languages
▶ Temporal Relational Calculus
▶ Time Domain
▶ Time Instant
▶ TSQL2
▶ Valid Time

Recommended Reading

1. Abiteboul S., Herr L., and Van den Bussche J. Temporal versus first-order logic to query temporal databases. In Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1996, pp. 49–57.
2. Baudinet M., Chomicki J., and Wolper P. Temporal deductive databases. In Temporal Databases: Theory, Design, and Implementation, Chap. 13, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R.T. and Snodgrass (eds.). Benjamin/Cummings, Reading, MA, 1993, pp. 294–320.
3. de Castilho J.M.V., Casanova M.A., and Furtado A.L. A temporal framework for database specifications. In Proc. 8th Int. Conf. on Very Data Bases, 1982, pp. 280–291.
4. Gabbay D., Hodkinson I., and Reynolds M. Temporal Logic: Mathematical Foundations and Computational Aspects. Oxford University Press, New York, 1994.
5. Goranko V., Montanari A., and Sciavicco G. A road map of interval temporal logics and duration calculi. J. Appl. Non-Classical Logics, 14(1–2):9–54, 2004.
6. Kamp J. Tense Logic and the Theory of Linear Order. PhD Thesis, University of California, Los Angeles, 1968.
7. Koymans R. Specifying real-time properties with metric temporal logic. Real-Time Systems, 2(4):255–299, 1990.
8. Manna Z. and Pnueli A. The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, Berlin, 1992.
9. Sernadas A. Temporal aspects of logical procedure definition. Inf. Syst., 5:167–187, 1980.
10. Toman D. On incompleteness of multi-dimensional first-order temporal logics. In Proc. 10th Int. Symp. Temporal Representation and Reasoning/4th Int. Conf. Temporal Logic, 2003, pp. 99–106.
11. Toman D. and Niwinski D. First-order queries over temporal databases inexpressible in temporal logic. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996, pp, 307–324.
12. Tuzhilin A. and Clifford J. A temporal relational algebra as a basis for temporal relational completeness. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 13-23.
13. van Benthem J. The Logic of Time. D. Reidel, 2nd edn., 1991.
14. Vardi M.Y. A temporal fixpoint calculus. In Proc. 15th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages, 1988, pp. 250–259.
15. Wolper P. Temporal logic can be more expressive. Inf. Contr., 56:72–99, 1983.

Temporal Logical Models

Arie Shoshani
Lawrence Berkeley National Laboratory, Berkeley, CA, USA

Synonyms

Historical data models

Definition

Temporal logical models refer to the logical structure of data that captures the temporal behavior and operations over such structures. The term "logical" is used to distinguish such temporal structures form the physical storage organization and implementation. For example, the behavior of temporal events and operations over them can be described logically in a way that is independent of the physical structure (e.g., linked lists) or indexing of the events. Temporal logical models include concepts of data values that are collected or are changed over time, such as continuous physical phenomena, a series of discrete events, and interval data over time. The challenge is one of having a single comprehensive model that captures this diversity of behavior.

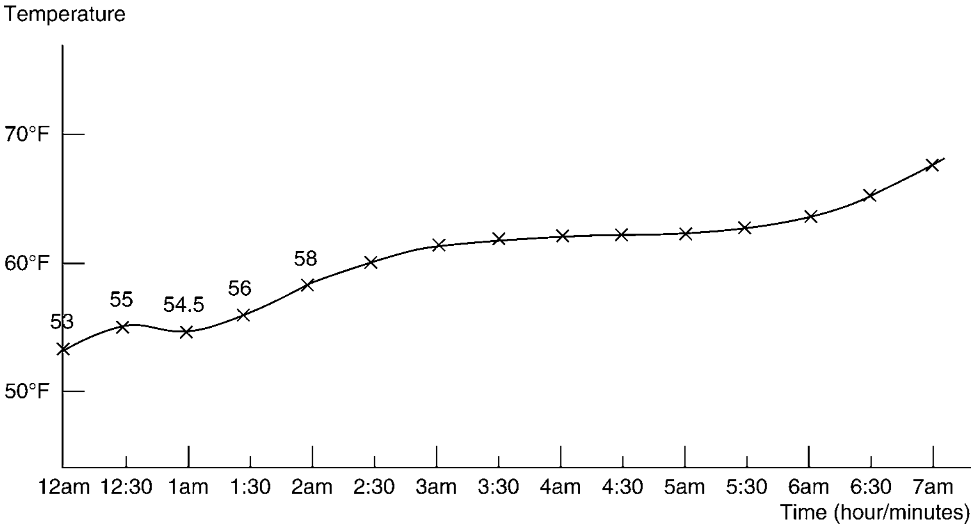
Historical Background

In the 1980s, several researchers focused on dealing with temporal data, both on the modeling concepts and on physical organization and indexing of temporal data. This led to the temporal database field to be established, and several books were written or edited on the subject (for example [4,5,12]). Since then, the subject continues to appear in specific application domains, or in combination with other concepts, such as spatio-temporal databases, and managing streaming data.

Foundations

The Treatment of Time in Database Systems

Time is a natural part of the physical world and an indispensable part of human activity, yet many database models treat temporal behavior as an afterthought. For example, weather (temperature, clouds, and storms) is a continuous phenomenon in time, yet it is treated as discrete events per day or per hour. In contrast, some human activities are fundamentally discrete events, such as salary which may change annually, but are treated as continuous concepts, where the salary is the same for the duration between the discrete events of salary changes. The main reason for the inconsistent treatment of time is that temporal objects and their semantics are not explicit in the data model. Consider for example, temperature measurements at some weather station as shown in Fig. 1. These are represented in conventional database systems (such as relational data models) as a two-part concept of time-of-measurement and value-of-measurement attributes, but the fact that the measurements are taken at evenly spaced intervals (e.g., every half an hour) and that the temperature represents a continuous



Temporal Logical Models. Figure 1. Continuous behavior of temperature measurements.

phenomenon is not captured. Consequently, if one asks what the temperature was at 12:35A.M, no such value exists. Furthermore, the interpolation function associated with getting this value is unknown. It could be a simple weighted averaging of the two nearest values, or a more sophisticated curve interpolation function.

Temporal Data Behavior

Temporal logical models are models designed to capture the behavior of temporal data sequences. First, some examples that illustrate the concepts that need to be captured by the model are presented.

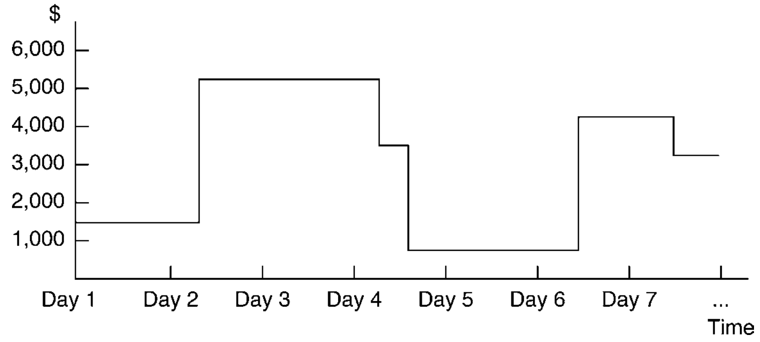
Example 1: wind velocity. Usually, the measurements of wind velocity are taken by devices at regular time periods, for example every hour. These are referred to as "time series." In this example, the measured quantity is not a single value, but has a more complex structure. It measures the direction of the wind and the velocity of the wind, which can be represented as a three-dimensional vector. The measured phenomenon is continuous, of course, but for this application it is determined by the database designers that a certain time granularity for queries is desired, such as values by minutes. Since the values are collected only hourly, an interpolation function must be provided and associated with this time sequence. The behavior is similar to the temperature behavior shown in Fig. 1, except that the measured values are three-dimensional vectors for each time point.

Example 2: bank account. The amount of money in the bank account changes when transactions take place. Money can be added or taken out of the account at irregular times. The value of the account is the same for the duration of time between transactions. This is shown in Fig. 2, where the granularity of the time points is in minutes. Note that the days shown should have precise dates in the database. Another aspect in this example is that in the case of a deposit of a check, funds may not be available until the check clears. Thus, there are two times associated with the deposit, the time of the transaction, and the time when funds are made available.

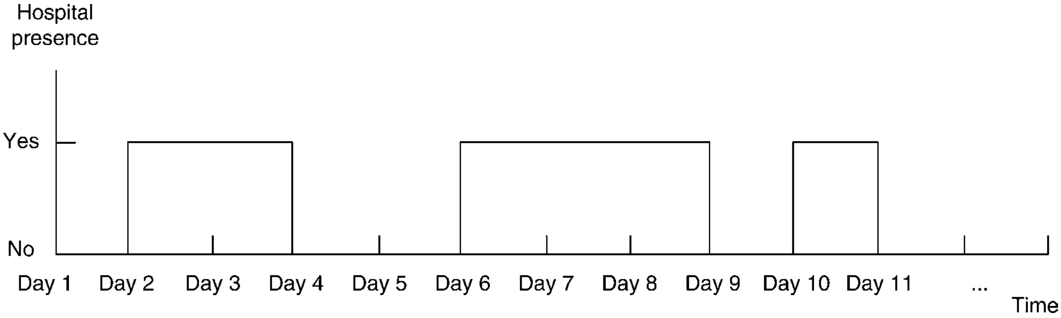
Example 3: hospitalization visits. Hospital visits of an individual occur typically at irregular times, and each can last a period of time, usually measured in days. The value associated with the hospital visit time sequence is Boolean; that is, only the fact that a visit took place or did not. This is an example where the concept of an interval must be represented in the data model. This is shown in Fig. 3, where the granularity is a day, and the interval durations span days. Here again, the days shown will have to have precise dates in the database.

Example 4: store revenue. Suppose that a store owner wishes to keep in a database the total revenue per day. The time sequence is regular, i.e., a time series (not counting days when the store is closed). The values per day do not represent continuous phenomena, but rather they are discrete in time, collected every day at the end of that day. This is the same as representing discrete events, such as the time of an accident, etc. In general, it does not make sense to apply interpolation to such a time sequence. However, if some data are missing, an interpolation rule could be used to infer the missing values. This is shown in Fig. 4. This is a time series, because only the days of business are shown (Monday – Friday).

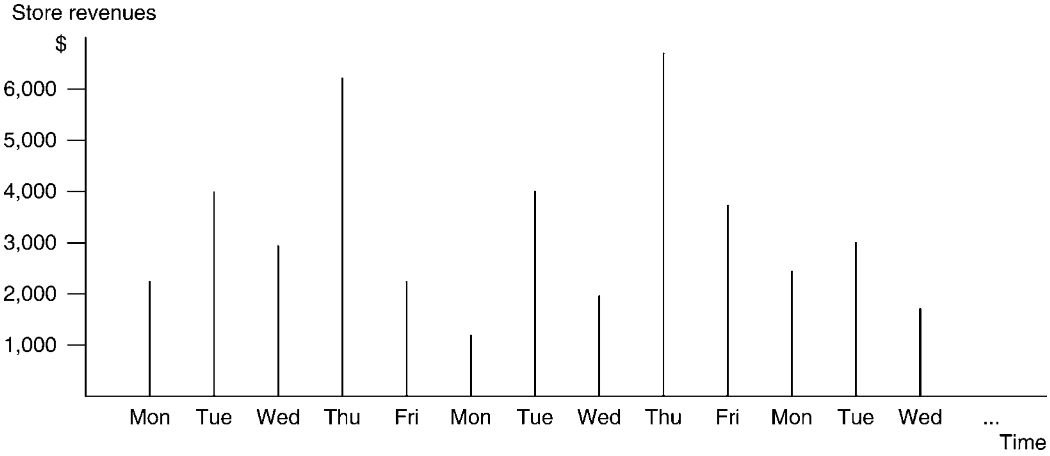
In the example above, only a single time sequence is shown, but there could be a collection of related time sequences. For example, time sequences of the quantity of each item sold in a store. For each item, there is a



Temporal Logical Models. Figure 2. Step-wise constant behavior of a bank account.



Temporal Logical Models. Figure 3. Interval behavior of hospital visits.



Temporal Logical Models. Figure 4. Discrete behavior of store revenues.

time sequence. However, all time sequences in this case have the same behavior, and they are collected in tandem per day. Such groups of related time sequences are referred to as "time sequence collections" [11]. As is discussed later, this concept is important for operations performed over collections of time sequences.

Behavioral Properties of Temporal Sequences

As is evident from the above examples, there are certain properties that can be specified to capture the logical behavior of temporal sequences. If such properties were supported by database systems, it is only necessary in such systems to store temporal data as time-value pairs in the general case, or simply ordered sequences of values for time series. These properties are discussed next, along with the possible category values they can assume.

Time-granularity: value and unit

The time-granularity indicates the time points for which data values can exist in the model. It is the smallest unit of time measure between time points in the time sequence. For example, if deposits and withdrawals to a bank account can be recorded with a minute precision, then the time granularity is said to be a minute. However, in cases where data values can be interpolated, an interpolation-granularity needs to be specified as well. For example, the temperatures shown in Fig. 1 are recorded every half an hour, and therefore the time granularity is 30 minutes, but given that values can be interpolated up to a minute precision, it is necessary to specify that the interpolation-granularity is a minute. This point is discussed further below in the section on "interpolation rule." Note that for regular time sequences (time series), it is often necessary to describe the origin of the time sequence, and the time granularity is relative to that origin. A formal treatment of time granularity can be found in [1].

1) Regularity: regular (time series), irregular

As mentioned above time series are regular sequences. They can be described by specifying the "time-step" between the time points. The time-step together with the "life span" (described next) specify fully the time points for which data values are expected to be provided. Because of its regular nature, it is not necessary to store the time points in the databases – these can be calculated. However, this is a physical implementation choice of the system, and the time values can be stored explicitly to provide faster access. Time series are prevalent in many applications, such as statistics gathering, stock market, etc.

Irregular time sequences are useful for event data that occurs in unpredictable patterns, such as bank withdrawals, visits to the library, or making a phone call. A typical phenomena in such sequences, is that most of the time points have no values associated with them. For example, suppose that the time granularity for recording phone calls is a minute. The time sequence of phone calls will typically be mostly empty (or null). For this reason, irregular time sequences are usually represented as time-value pairs in the database.

Life span: begin-time, end-time

The life span indicates for what period of time the time sequence is valid. The begin-time is always necessary, and has to be specified with the same precision of the time granularity. For example, if for the temperature time series in Fig. 1, the begin-time was January 1, 1:15 A.M, and the granularity was 30 minutes, then the time points will be 1:15 A.M, 1:45 A.M, 2:15 A.M, etc.

The life span end-time can be specified as "open-ended." That means that this time series is active.

Behavior type: continuous, step-wise-constant, interval, discrete.

These types were illustrated in the examples of the previous section. For example 1, on wind velocity, the type is continuous. For example 2, the amount available in a bank account, the type is step-wise-constant. For example 3, of hospital visits, the type is interval. For example 4, the store revenues per day, the type is discrete. Note that the interval type can be considered as a special case of step-wise-constant type having the Boolean values (0 or 1). Another thing worth noting is that discrete time sequences cannot be associated with an interpolation rule. The interpolation rules for the other types are discussed next.

Interpolation rule: interpolation-granularity, interpolation-function.

The interpolation-granularity has to be specified in order for the underlying system to enforce the data points for which interpolation can be applied in response to queries. Note that the interpolation-granularity has to be in smaller units than the time-granularity, and the number of interpolation-granularity points in a time-granularity unit must be an integer. For example, while temperature in the example of Fig. 1 has time-granularity of 30 minutes, the interpolation-granularity can be 5 minutes.

The interpolation-function for the step-wise-constant and interval types are straightforward, and are implied by the type. But, for a continuous type, an interpolation-function must be specified. It should be possible to provide the system with such a function for each continuous time sequence. If no interpolation-function is provided, the system can use a default function.

Value type: binary, numeric, character, multi-valued, etc.

This property of temporal sequences is no different from specifying attribute types in conventional database systems. The case of a binary type is special to interval events, and is not always supported by conventional system. Also, multi-valued or multi-component attributes are special requirements for more sophisticated time sequences that exist in scientific data, such as the wind velocity in example 2.

Transaction vs. valid time: transaction, valid

Similar to the bank account in example 2 where the deposit time was different from the time when funds are available, there are many examples where temporal data are recorded in the database before the data values are valid. This concept was explored extensively in [13], and referred to as "transaction time" and "valid time." Actually, there are situations where the transaction time can occur after the valid time for retroactive cases. For example, a salary raise can be added to a database in March of some year, but is retroactive to January of that year. This concept has led to an extensive literature on representing it as an extension of query languages, including a temporal extension to the SQL query language, referred to as TSQL [12]. It is worth noting that other concepts of multiple temporal dimensions were also introduced in the literature, in addition to transaction and valid times. For example, [3] introduced the concept of "event time" – times of the events that initiates and terminates the interval validity, and "availability time" – the time interval during which facts are available.

If multiple temporal dimensions are needed in the model, they can be thought of as multiple correlated time sequences. However, in general, each time dimension can have different properties. For example, the transaction time sequence for bank deposits can have a granularity of a minute, while the valid time for the available funds can be daily.

Operation over Temporal Data

Because of the inherent time order of temporal data, operations over them, such as "when," "preceding," "following," etc. are based on the time order. Similarly, the concept of a "time window" is natural. Various researchers have developed precise semantics to query languages by adding temporal operators to existing query languages, including relational query languages, such as SQL, relational algebras, such as QUEL, functional query languages, such as DAPLEX, deductive query languages, such a Datalog, and entity-relationship languages. Many such examples can be found in the books on temporal databases [4,5]. In order to explain the various operators, they are classified into the following four categories.

Predicate Operators Over Time Sequences Predicate operators refer to either specific times or a time interval. For specific times, the obvious predicates include "before," "after," "when," etc. But, in addition, there are operators that refer to the life span, such as "first, and "last." For time intervals, operators such as "during" or "interval" are used. Also, when specifying an interval, the keyword "now" is used to refer to time sequences that are active, such as "interval" (Jan 01, 2007, now). Note that the time values used must be expressed at the granularity of the time sequence (or the interpolation-granularity if interpolation is allowed). In some time sequences, it is useful to use an integer to refer to the nth time instance, such as t-33 to mean the 33rd time point in the time sequence. However, this is not included in most proposed query languages.

Another purpose of predicate operators is to get back the time periods where some condition on the data values hold. For example, suppose that a time sequences represents temperature at some location. The query "get periods where temperature >100" (the units are Prof.F) will return a (Boolean) interval time sequence, where the temperature was greater than 100. Note that "periods" is treated as a keyword.

Aggregation Operators Over Time Windows The usual statistical operators supported by database systems (sum, count, average, min, max) can be applied to a specific time window (t_begin, t_end), to the entire time sequence (first, last), or to the combinations (first, t_end), (t_begin, last). In general, "first" or "last" can be substituted by an instance number, such as "t-33" mentioned above. Here, again, the time has to be specified in the granularity of the time sequence.

Another way to apply operators over windows is to combine that with the "group by" concept. This is quite natural for temporal sequence that involve calendar concepts of month, day, minute, second, etc. For example, suppose that a time sequence represents daily sales. One can have a query "sum sales by month." This is the same as asking for multiple windows, each over the days in each month.

Aggregation Operators Over Time Sequence Collections In a typical database, time sequences are defined over multiple object instances. For example, one can have in an organization the salary history of all of its employees. Asking for the "average salary over all employees" over time requires the average operation to be applied over the entire collection of time sequences. This operation is not straight forward if all salary raises do not occur at the same time. This operation will generate a time sequence whose time points are the union of all the time points of the time sequences, where the average values are performed piecewise on the resulting intervals.

Similar to the case of aggregation over time windows, where the "group by" operation can be applied, it is possible to group by object instances in this case. For example, if employees are organized by departments, one can ask for "average salary over all employees per department."

Composition of Time Sequences Composition refers to algebraic operations over different time sequences. For example, suppose that in addition to salary history recorded for each employee in an organization, the history of commissions earned is recorded. In order to obtain "total income," the salary and the commission time sequences have to be added for each employee. This amounts to the temporal extension of algebraic operations on multiple attributes in non-temporal query languages.

Combinations of the Above Operators It is conceptually reasonable to combine the above operators in query languages. For example, it should be possible to have the aggregation and composition operators applied only to a certain time window, such as getting the "average salary over all employees for the last three years." Furthermore, it should be possible to apply a temporal operator to the result of another temporal operator. This requires that the result of operations over time sequences is either a time sequence or a scalar. If it is a time sequence, temporal operators can be applied. If it is a scalar (a single value) it can be applied as a predicate value. This requirement is consistent with other languages, such as the relational language, where the operation on relations always generates a relation or a scalar.

Additional Concepts There are many concepts introduced in the literature that capture other logical aspects of temporal operations and semantics. This broad literature cannot be covered here; instead, several concepts are mentioned next. Temporal specialization and generalization are explored in [6]. Unified models for supporting point-based and interval-based semantics are developed in [2,14]. It is argued in [8] that temporal data models have to include explicitly the concept of ordered data, and a formal framework for that is proposed. A single framework for supporting both time series and version-based temporal data are developed in [9]. There are also many papers that discuss how to efficiently support temporal operations (such as aggregation), see for example [7,10]. Finally, in the context of streaming data, temporal aggregation operations on time windows have in explored – see entries on "stream data management" and "stream mining."

Key Applications

Temporal data are ubiquitous. It naturally exists in applications that have time series data, such as stock market historical data, or history of transactions in bank accounts. In addition, it is a basic requirement of scientific databases collecting data from instruments or performing simulation over time steps. In the past, many databases contained only the current (most updated) data, such as current salary of employees, current inventories in a store or a warehouse, etc. The main reason for that was the cost of storage and efficiency of processing queries. One could not afford keeping all the historical data. More recently, as the cost of storage is plummeting, and compute engines are faster and can operate in parallel, historical data are routinely kept. While it is still worth keeping a version of current data for some applications for efficiency of access, many applications now use historical data for pattern and trend analysis over time, especially in data warehouse applications.

Future Directions

While a lot of research was done on temporal data, the concepts and operations over such data are only partially supported, if at all, in commercial and open source database system. Some support only the concept of date_time (it is complex enough, crossing time zones and century boundaries), but the support for properties of time sequences and operations over them are still not generally available. Building such database systems is still a challenge.

Cross-references

- ▶ Data Models
- ▶ Data Warehouse
- ▶ Database Design
- ▶ Query Language
- ▶ Spatial Network Databases
- ▶ Stream Data Analysis
- ▶ Stream Mining

Recommended Reading

1. Bettini C, Wang X.S., and Jajodia S., A general framework for time granularity and its application to temporal reasoning. Ann. Math. Artif. Intell., 22(1–2):29–58, 1998.
2. Chen C.X., and Zaniolo C. Universal temporal extensions for database languages. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 428–437.
3. Combi C., and Montanari A. Data models with multiple temporal dimensions: completing the picture. In Proc. 13th Int. Conf. on Advanced Information Systems Eng., 2001, pp. 187–202.

4. Etzion O, Jajodia S, and Sripada S.M., (eds.). Temporal Databases: Research and Practice. Springer, Berlin Heidelberg, 1998.
5. Tansel A.U., Clifford J., Gadia S.K., Jajodia S., Segev A., and Snodgrass R.T. Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, Redwood City, CA, 1993.
6. Jensen C.S., and Snodgrass R.T. Temporal specialization and generalization. IEEE Trans. Knowl. Data Eng., 6(6):954–974, 1994.
7. Kang S.T., Chung Y.D., and Kim M.-Y., An efficient method for temporal aggregation with range-condition attributes. Inf. Sci., 168(1–4):243–265, 2004.
8. Law Y.N., Wang H., and Zaniolo C. Query languages and data models for database sequences and data streams. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 492–503.
9. Lee J.Y., Elmasri R., and Won J. An integrated temporal data model incorporating time series concept. Data Knowl. Eng., 24 (3):257–276, 1998.
10. Moon B., López I.F.V., and Immanuel V. Efficient algorithms for large-scale temporal aggregation. IEEE Trans. Knowl. Data Eng., 15(3):744–759, 2003.
11. Segev A, and Shoshani A. Logical modeling of temporal data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 454–466.
12. Snodgrass R.T. The TSQL2 Temporal Query Language. Kluwer, Norwell, MA, 1995.
13. Snodgrass R.T., and Ahn I. Temporal databases. IEEE Comput., 19(9):35–42, 1986.
14. Terenziani P., and Terenziani R.T. Reconciling point-based and interval-based semantics in temporal relational databases: a treatment of the Telic/Atelic distinction. IEEE Trans. Knowl. Data Eng., 16(5):540–551, 2004.

Temporal Middleware

► Temporal Strata

Temporal Object-Oriented Databases

Carlo Combi
University of Verona, Verona, Italy

Definition

In a strict sense, a temporal object-oriented database is a database managed by an object-oriented database system able to explicitly deal with (possibly) several temporal dimensions of data. The managed temporal dimensions are usually valid and/or transaction times. In a wider sense, a temporal object-oriented database is a collection of data having some temporal aspect and managed by an object-oriented database system.

Historical Background

Research studies on time, temporal information, and object-oriented data started at the end of 1980s and continued in the 1990s. From the seminal work by Clifford and Croker on objects in time [4], several different topics have been discussed. Segev and Rose studied both the modeling issues and the definition of suitable query languages [12]; Wuu and Dayal showed how to use an object-oriented data model to properly represent several temporal aspects of data [14]; Goralwalla et al. studied the adoption and extension of an object-oriented database system, TIGUKAT, for managing temporal data, and finally proposed a framework allowing one to classify and define different temporal object-oriented data models, according to the choices adopted for representing time concepts and for dealing with data having temporal dimensions [8]. Schema evolution in the context of temporal object-oriented databases has been studied by Goralwalla et al. and by Edelweiss et al. [6,9]. Bertino et al. studied the problem of formally defining temporal object-oriented models [1] and, more recently, they proposed an extension of the ODMG data model, to deal with temporalities [2]. Since the end of 1990s, research has focused on access methods and storage structures, and on temporalities in object relational databases, which, in contrast to object-oriented databases, extend the relational model with some object-oriented features still maintaining all the relational structures and the related, well studied, systems. As for the first research direction, for example, Norvag studied storage structures and indexing techniques for temporal object databases supporting transaction time [10]. Recently, some effort has been done on extending object-relational database systems with some capability to deal with temporal information; often, as in the case of the work by Zimanyi and colleagues, studies on temporalities in object-relational databases are faced in the context of spatio-temporal databases, where temporalities have to be considered together with the presence of spatial information [15]. In these years, there were few survey papers on temporal object-oriented databases, even though they were considered as a part of survey papers on temporal databases [11,13]. Since the 1990s, temporal object-oriented databases have also been studied in some more application-oriented research efforts: among them are temporal object models and languages for multimedia and for medical data, as, for example, in [5,7].

Foundations

Object-oriented (OO) methodologies and technologies applied to the database field have some useful features – abstract data type definition, inheritance, complex object management – in modeling, managing, and storing complex information, as that related to time and to temporal information. Objects and times have, in some sense, a twofold connection: from a first point of view, complex, structured types could be suitably defined and used to manage both complex time concepts and temporal dimensions of the represented information; on the other side, object-oriented data models and languages may be suitably extended with some built-in temporal features. In the first case, the object-oriented technology is performed to show that complex concepts, as those related to time and temporal information, can be suitably represented and managed through types, generalization hierarchies, and so on. In the second case, the focus is slightly different and is on extending object-oriented models and languages to consider time-related concepts as first-class citizens: so, for example, each object, besides an object identifier, will have a lifespan, managed by the system.

Even though different approaches and proposals in the literature do not completely agree on the meaning of different terms used by the "object-oriented" community, there is a kind of agreement on the basic concepts of any object-oriented data model. An *object* may represent any entity of the real world, e.g., a patient, a therapy, a time interval. The main feature of an object is its *identity*, which is immutable, persists during the entire existence of the object, and is usually provided by the database system through an identifier, called OID (Object IDentifier). An object is characterized by a state, described by properties (*attributes* and *associations* with other objects) and by a behavior, defined by *methods*, describing modalities, by which it is possible to interact with the object itself. Objects are created as instances of a *type*; a *class* is the collection (also named *extent*) of all the objects of a given type stored into the database at a certain moment. A *type* describes (i) the structure of properties through attributes and associations, and (ii) the behavior of its instances through methods applicable to objects instances of that type.

To introduce the faced topics, a simple clinical database is used, storing data on symptoms and therapies of patients and will graphically represent its schema through the well-known UML notation for class diagrams, adopting the primitive types provided by the ODMG data model [3].

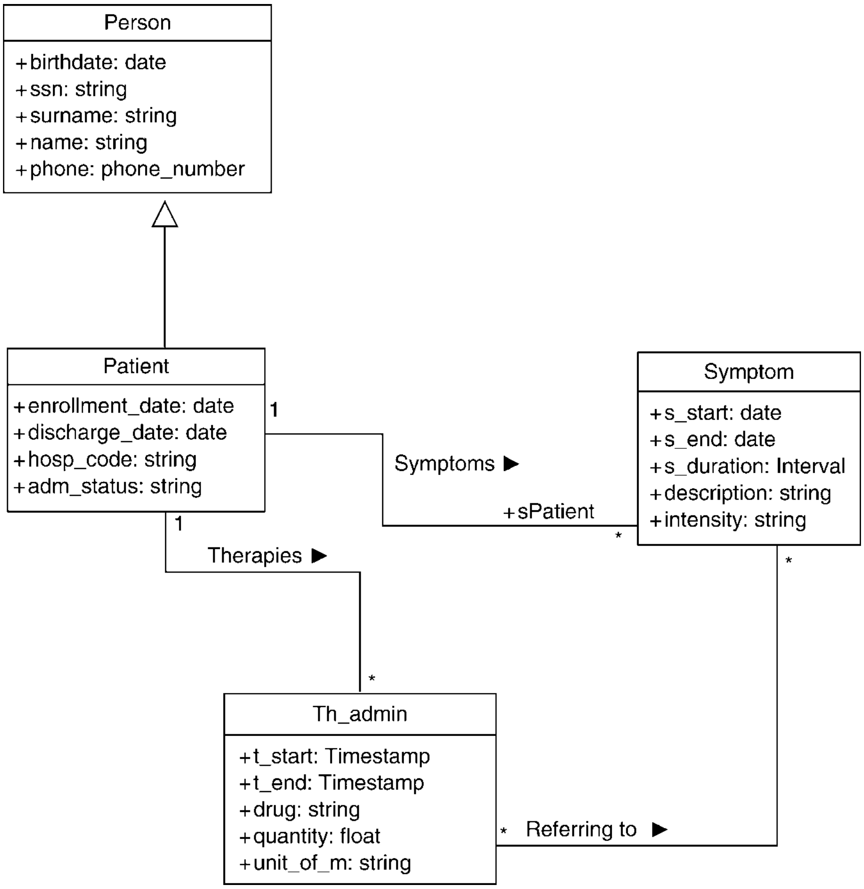
Time and Abstract Data Types

Through suitable abstract data types (ADTs), it is possible to represent several and different features of time primitives; more specifically ADTs can be suitably used to represent anchored (i.e., time points and time periods) and unanchored (i.e., durations) temporal primitives. Usually, any OO database system allows one to define ADTs, which can be possibly used to represent complex temporal concepts; moreover, any database system usually provides several built-in primitive and complex data types, allowing the user to represent the most usual time concepts.

Considering, for example, the clinical database depicted in Fig. 1, it is possible to observe different data types for time; according to the ODMG data model, the types Date and Timestamp represent time points corresponding to days and to seconds, respectively, while the type Interval stands for spans of time, i.e., distances between two time points.

According to this approach, as underlined by Wuu and Dayal [14], different types of time could be defined, starting from some general concept of time, through aggregation and generalization. For example, the ODMG type Timestamp could be viewed as a specialization of a supertype Point, having two operations defined, i.e., those for comparisons (= and >), as its behavior. Various time types can be defined as subtypes of Point. The two operations defined for Point are inherited, and could be suitably redefined for different subtypes. Indeed, properties of specific ordering relationships determine different time structures, e.g., total vs. partial, or dense vs. discrete. Moreover, through the basic structured types provided by the model, it is possible to build new types for time concepts. As an example, it could be important to specify a type allowing one to represent periods of time; this way, it is possible to represent in a more compact and powerful way the periods over which a therapy has been administered, a symptom was holding, a patient was hospitalized. Thus, in the database schema a new type Period will be used as part of types Patient, Th_admin, and Symptom, as depicted in Fig. 2.

ADTs may be used for managing even more complex time concepts [5], allowing one to deal with times given at different granularities or with indeterminacy in a uniform way.



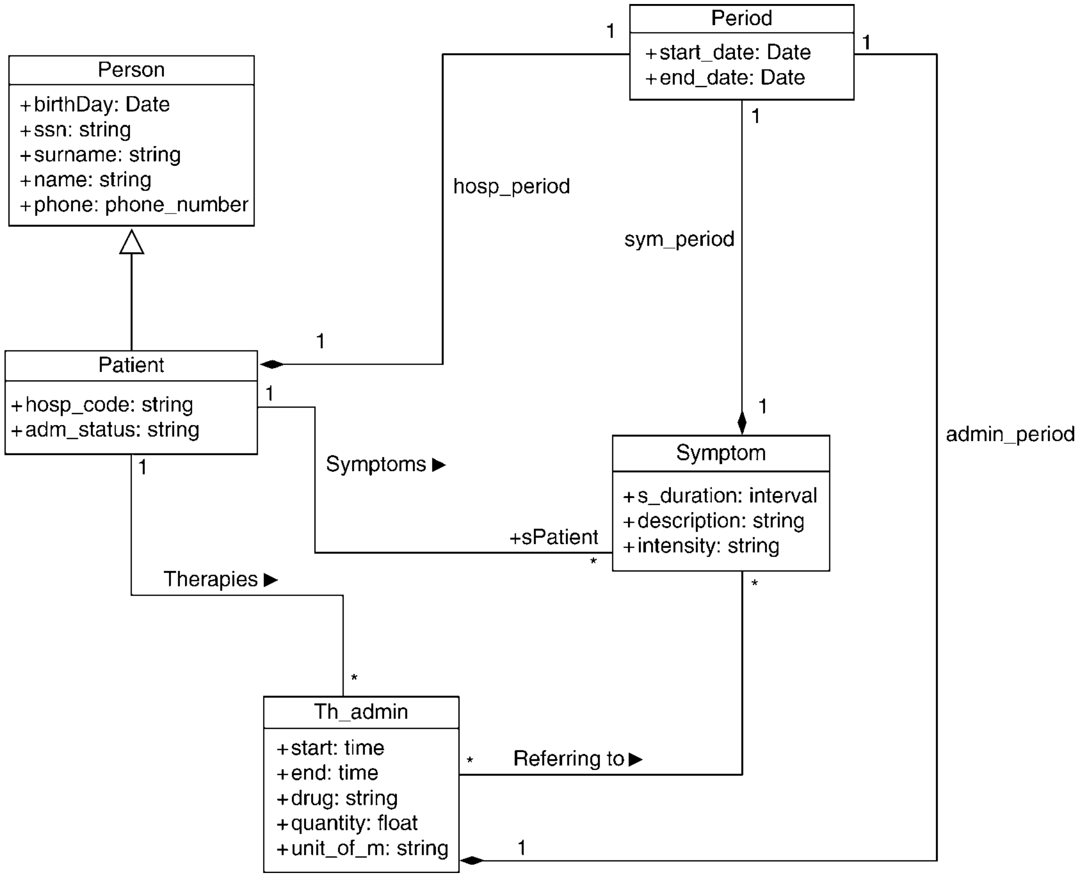
Temporal Object-Oriented Databases. Figure 1. Object-oriented schema of the database about patients' symptoms and therapies, using ODMG standard time types.

Temporal Object Data Models

Up to this point, the proposed solutions just use ADTs to represent complex time concepts; the sound association of time to objects for representing temporal data, i.e., time evolving information, according to the well-known temporal dimensions, such as valid and/or transaction times, is left to the user. Focusing, without loss of generality, on valid time, in the given example the semantics of valid time must be properly managed by the application designer, to check that symptoms occurred and therapies were administered when the patient was alive, that therapies related to symptoms were administered after that symptoms appeared, that there are no different objects describing the same symptom with two overlapping periods, and so on.

Managing this kind of temporal aspects in object databases has been the main focus of several research studies [13]: the main approaches adopted in dealing with the temporal dimension of data in object-oriented data models are (i) the direct use of an object-oriented data model (sometimes already extended to deal with other features as version management), and (ii) the modeling of the temporal dimensions of data through ad-hoc data models. The first approach is based on the idea that the rich (and extensible) type system usually provided by OO database systems allows one to represent temporal dimensions of data as required by different possible application domains. The second approach, instead, tries to provide the user with a data model where temporal dimensions are first-class citizens, avoiding the user the effort of modeling from scratch temporal features of data for each considered application domain.

General OO Models Using OO Concepts for Modeling Temporal Dimensions Among the proposed object-oriented systems able to deal with temporal

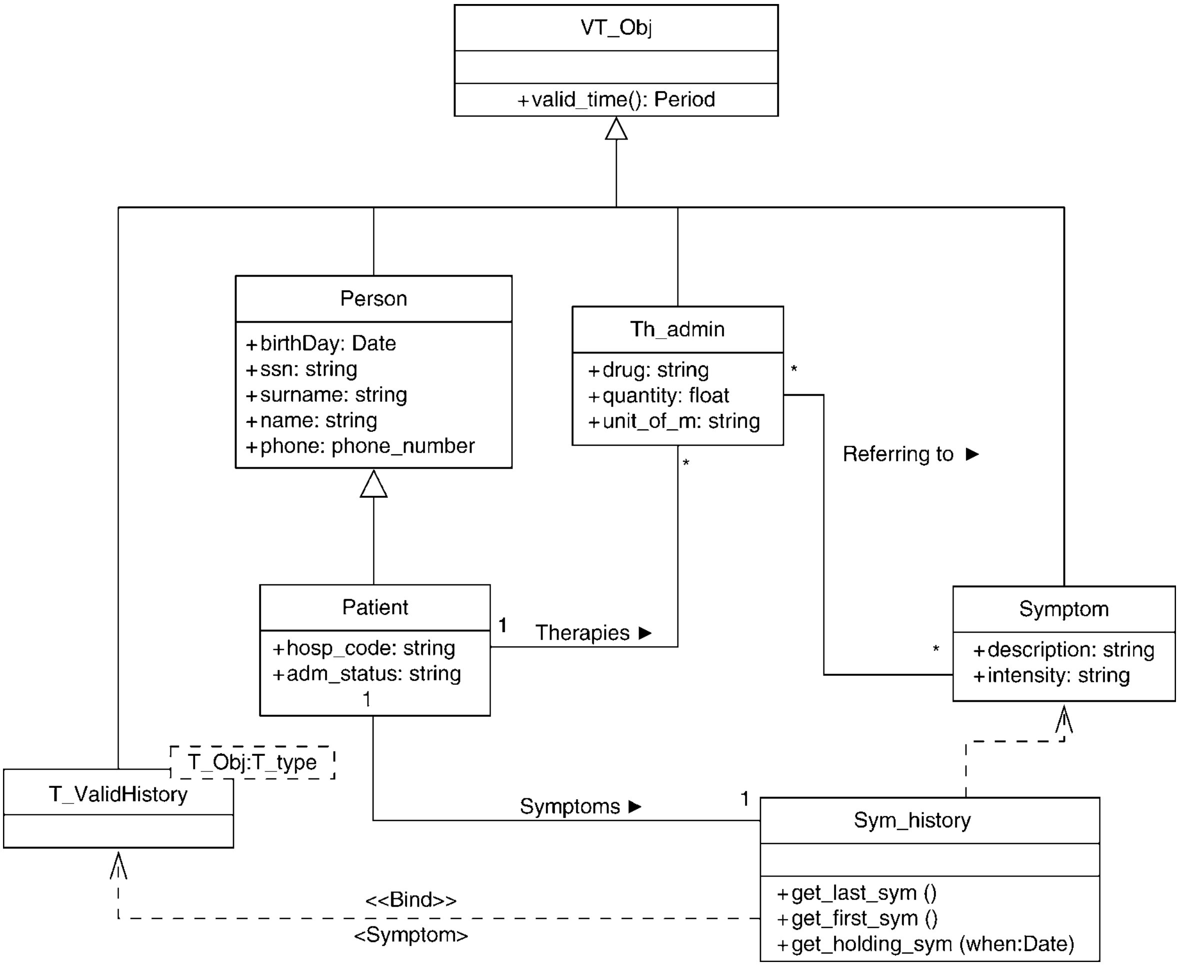


Temporal Object-Oriented Databases. Figure 2. Object-oriented schema of the database about patients' symptoms and therapies, introducing the type Period.

information, OODAPLEX and TIGUKAT adopt the direct use of an object-oriented data model [7,14]. In these systems suitable data types allow the database designer to model temporal information. For example, TIGUKAT models the valid time at the level of objects and of collections of objects. More particularly, for each single application it is possible to use the rich set of system-supported types, to define the real semantics of valid (or transaction) time.

According to this approach, the example database schema could be modified, to manage also the valid time semantics: as depicted in Fig. 3, objects having the valid time dimension are explicitly managed through the type VT_Obj, the supertype of all types representing temporal information, while the type Sym_History, based on the (template) type T_valid History is used to manage the set of objects representing symptoms.

OO Models Having Explicit Constructs for Temporal Dimensions of Data Besides the direct use of an OO data model, another approach which has been widely adopted in dealing with the temporal dimension of data by object-oriented data models consists of temporally-oriented extensions, which allow the user to explicitly model and consider temporal dimensions of data. As an example of this approach, Fig. 4 depicts the schema related to patients' symptoms and therapies: the temporal object-oriented data model underlying this schema allows one to manage the valid time of objects, often referred to as *lifespan* in the object-oriented terminology (this aspect is represented through the stereotype temporal and the operation lifespan() for all types); moreover, the temporal model allows the designer to specify that attributes within types can be time-varying according to different granularities: for example, the intensity of a symptom

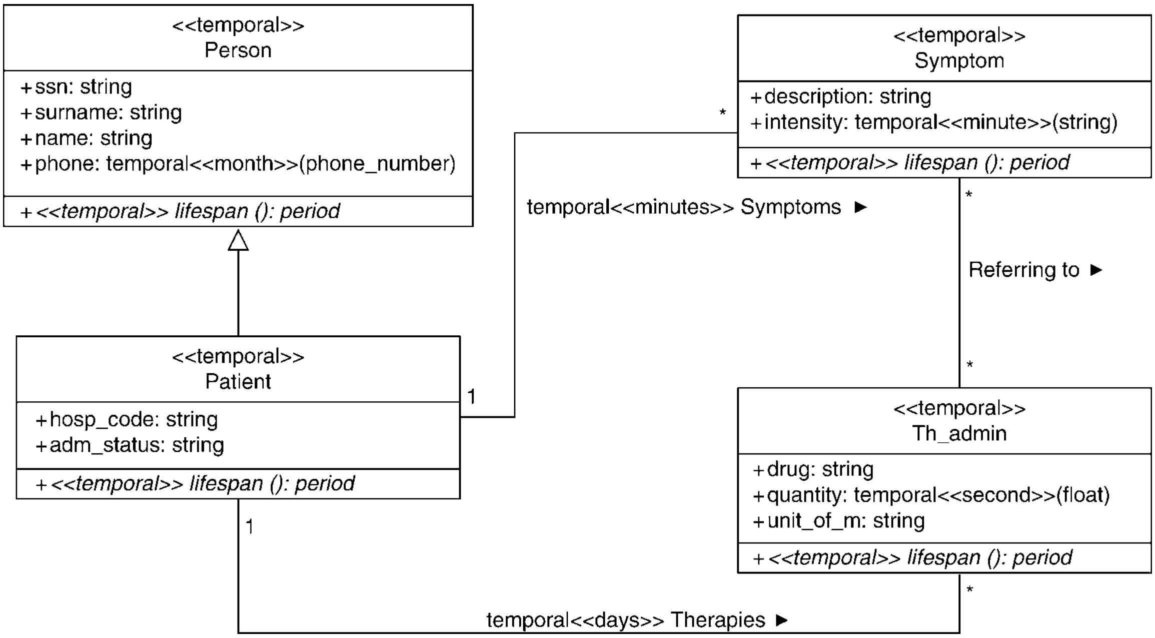


Temporal Object-Oriented Databases. Figure 3. Object-oriented schema of the database about patients' symptoms and therapies, modeling temporal dimensions through suitable types.

may change over time. Finally, temporalities can be represented even for associations, specifying the granularity to consider for them. Several constraints can be defined, and implicitly verified, when defining a temporal object-oriented data model. For example, in this case, any object attribute could be constrained to hold over some subinterval of the lifespan of the considered object: $\forall o \in$ Symptom (o.intensity.$VT() \subseteq o$.lifespan()), where $VT()$ returns the overall period over which the (varying) values of a temporal attribute have been specified. In a similar way, a temporal relationship is allowed only when both the related objects exist. As for the inheritance, objects could be allowed to move from some supertype to a subtype during their existence; the constraint here is that their lifespan as instances of a subtype must be a subinterval of their lifespan as instances of a supertype: $\forall o \in$ Patient (o.lifespan() $\subseteq$ ((Person)o).lifespan())

Temporal Object Query Languages

According to the approaches for object-oriented data models dealing with temporal information, even when querying temporal data, it is possible to either adopt a generic object-oriented query language and use directly its constructs for temporal data or extend an (atemporal) query language with some ad-hoc keywords and clauses to deal with temporal data in a more powerful and expressive way [3,5,12]. For both the approaches, the main focus is on querying data: data definition and data manipulation are usually performed through the object-oriented language provided by the database system [5].


<<temporal>>
Person
+ssn: string
+surname: string
+name: string
+phone: temporal<<month>>(phone_number)
+<<temporal>> lifespan (): period
<<temporal>>
Symptom
+description: string
+intensity: temporal<<minute>>(string)
+<<temporal>> lifespan (): period
temporal<<minutes>> Symptoms ►
Referring to ►
<<temporal>>
Patient
+hosp_code: string
+adm_status: string
+<<temporal>> lifespan (): period
<<temporal>>
Th_admin
+drug: string
+quantity: temporal<<second>>(float)
+unit_of_m: string
+<<temporal>> lifespan (): period
temporal<<days>> Therapies ►
1
1
*
*
*
*

Temporal Object-Oriented Databases. Figure 4. The temporal object-oriented schema of the database about patients' symptoms and therapies.

Temporal Object-Oriented Database Systems

Usually, temporal object-oriented database systems, offering a temporal data model and a temporal query language, are realized on top of object-oriented database systems, through a software layer able to translate temporalities in data and queries into suitable data structures and statements of the underlying OO system [5]. According to this point of view, only recently some studies have considered technical aspects at the physical data level; among them it is worth mentioning here the indexing of time objects and storage architectures for transaction time object databases [10].

Key Applications

Clinical database systems are among the applications of temporal object-oriented databases, i.e., the real world databases where the structural complexity of data needs for the object-oriented technology. Indeed, clinical database systems have to manage temporal data, often with multimedia features, and complex relationships among data, due both to the healthcare organization and to the medical and clinical knowledge. Attention in this field has been paid on the management of clinical data given at different granularities and with indeterminacy [5,7]. Another interesting application domain is that of video database systems, where temporal aspects of object technology have been studied for the management of temporal aspects of videos. Complex temporal queries have been studied in this context, involving spatio-temporal constrains between moving objects.

Future Directions

New and more intriguing topics have attracted the attention of the temporal database community in these last years; however, the results obtained for temporal object-oriented databases could be properly used in different contexts, such as that of temporal object-relational databases, which seem to attract also the attention of the main companies developing commercial database systems, and that of semi-structured temporal databases, where several OO concepts could be studied and extended to deal with temporalities for partially structured information (as that represented through XML data).

Cross-references

- ► Object-Oriented Data Model
- ► Object Query Language
- ► Object-Relational Data Model
- ► Spatio-temporal Data Models

- ▶ Temporal Granularity
- ▶ Temporal Indeterminacy
- ▶ Temporal Query Languages

Recommended Reading

1. Bertino E., Ferrari E., and Guerrini G. A formal temporal object-oriented data model. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996, pp. 342–356.
2. Bertino E., Ferrari E., Guerrini G., and Merlo I. T-ODMG: an ODMG compliant temporal object model supporting multiple granularity management. Inf. Syst., 28(8):885–927, 2003.
3. Cattel R.G.G. and Barry D.K. (eds.). The Object Data Standard: ODMG 3.0. Morgan Kaufmann, Los Altos, CA, 2000.
4. Clifford J. and Croker A. Objects in time. IEEE Data Eng. Bull., 11(4):11–18, 1988.
5. Combi C., Cucchi C., and Pinciroli F. Applying object-oriented technologies in modeling and querying temporally-oriented clinical databases dealing with temporal granularity and indeterminacy. IEEE Trans. Inf. Tech. Biomed., 1:100–127, 1997.
6. Galante R.M., dos Santos C.S., Edelweiss N., and Moreira A.F. Temporal and versioning model for schema evolution in object-oriented databases. Data Knowl. Eng., 53(2):99–128, 2005.
7. Goralwalla I.A., Özsu M.T., and Szafron D. Modeling medical trials in pharmacoeconomics using a temporal object model. Comput. Biol. Med., 27:369–387, 1997.
8. Goralwalla I.A., Özsu M.T., and Szafron D. An object-oriented framework for temporal data models. In Temporal Databases: Research and Practice. O. Etzion, S. Jajodia, S. Sripada (eds.), Springer, 1998, pp. 1–35.
9. Goralwalla I.A., Szafron D., Özsu M.T., and Peters R.J. A temporal approach to managing schema evolution in object database systems. Data Knowl. Eng., 28(1):73–105, 1998.
10. Nørvåg K. The vagabond approach to logging and recovery in transaction-time temporal object database systems. IEEE Trans. Knowl. Data Eng., 16(4):504–518, 2004.
11. Ozsoyoglu G. and Snodgrass R.T. Temporal and real-time databases: a survey. IEEE Trans. Knowl. Data Eng., 7(4):513–32, 1995.
12. Rose E. and Segev A. TOOSQL – A Temporal Object-Oriented Query Language. In Proc. 12th Int. Conf. on Entity-Relationship Approach, 1993, pp. 122–136.
13. Snodgrass R.T. Temporal Object-Oriented Databases: A Critical Comparison. In Modern Database Systems: The Object Model, Interoperability and Beyond. W. Kim (ed.). Addison-Wesley, 1995, pp. 386–408.
14. Wuu G.T.J. and Dayal U. A Uniform Model for Temporal and Versioned Object-oriented Databases. In Temporal Databases. A.U. Tansel, J. Clifford, S.K. Godia, A. Segev, R. Snodgrass (eds.), 1993, pp. 230–247.
15. Zimányi E. and Minout M. Implementing conceptual spatio-temporal schemas in object-relational dbmss. In OTM Workshops (2). LNCS, Vol. 4278, R. Meersman, Z. Tari, P. Herrero (eds.), 2006, pp. 1648–1657.

Temporal Periodicity

Paolo Terenziani
University of Turin, Turin, Italy

Definition

Informally, *periodic events* are events that repeat regularly in time (e.g., each Tuesday), and *temporal periodicity* is their temporal *periodic pattern* of repetition. A pattern is *periodic* if it can be represented by specifying a finite portion of it, and the duration of each repetition. For instance, supposing that day 1 is a Monday, the pair <*'day 2,' '7 days'*> may implicitly represent all Tuesdays.

A useful generalization of periodic patterns are *eventually* periodic ones, i.e., patterns that can be expressed by the union of a periodic pattern and a finite non-periodic one.

The above notion of periodic events can be further extended. For instance, Tuzhilin and Clifford [14] distinguish between *"strongly"* periodic events, that occur at equally distant moments in time (e.g., a class, scheduled to meet once a week, on Wednesday at 11A.M), *"nearly periodic"* events, occurring at regular periods, but not necessarily at equally distant moments of time (e.g., a meeting, that has to be held once a week, but not necessarily on the same day), and *"intermittent"* events, such that if one of them occurred then the next one will follow some time in the future, but it is not clear when (e.g., a person visiting "periodically" a pub). Most of the approaches discussed in the following cope only with "strongly" periodic events.

Finally, it is worth highlighting that "(periodic) temporal granularity," "calendar," and "calendric system" are notions closely related to temporal periodicity.

Historical Background

Temporal periodicity is pervasive of the world all around us. Many natural and artificial phenomena take place at periodic time, and temporal periodicity seems to be an intrinsic part of the way humans approach reality. Many real-world applications, including process control, data access control, data broadcasting, planning, scheduling, multimedia, active databases, banking, law and so on need to deal with periodic events. The problem of how to store and query periodic data has been widely studied in the fields of databases, logic, and artificial intelligence.

In all such areas it is widely agreed that, since many different data conventions exist, a pre-defined set of periodicities would not suffice. For instance, Snodgrass and Soo [12] have emphasized that the use of a calendar depends on the cultural, legal, and even business orientation of the users, and listed many examples of different calendric systems. As a consequence, many approaches to *user-defined* temporal periodicity have been proposed. The core issue is the definition of expressive *formal languages* to *represent* and *query* user-defined temporal periodicity. In particular, an *implicit* representation [2] is needed, since it allows one to cope with data holding at periodic times in a compact way instead of explicitly listing all the instances (extensions) of the given periodicity (e.g., all "days" in a possibly infinite frame of time). Additionally, also the *set-theoretic operations* (intersection, union, difference) on definable periodicities can be provided (e.g., in order to make the formalism a suitable candidate to represent periodic data in temporal databases). Operationally, also *mapping functions* between periodicities are an important issue to be taken into account.

Within the database community, the problem of providing an implicit treatment of temporal periodicity has been intensively investigated since the late 1980s. Roughly speaking, such approaches can be divided into three mainstreams (the terminology is derived from Baudinette et al. [2] and Niezette and Stevenne [9]):

(1) *Deductive rule*-based approaches, using deductive rules. For instance, Chomicki and Imielinsky [4] dealt with periodicity via the introduction of the successor function in Datalog;
(2) *Constraint*-based approaches, using mathematical formulae and constraints (e.g., [6]);
(3) *Symbolic* approaches, providing symbolic formal languages to cope with temporal periodicity in a compositional (and hopefully *natural* and *commonsense*) way (consider, e.g., [9,8]).

Tuzhilin and Clifford [14] have proposed a comprehensive survey of many works in such mainstreams, considering also several approaches in the areas of logics and of artificial intelligence.

Foundations

In the following, the main features of the three mainstreams mentioned above are analyzed.

Deductive Rule-Based Approaches

Probably the first milestone among *deductive rule-based* approaches is the seminal work by Chomicki and Imielinski [4], who used Datalog_{1S} to represent temporal periodicity. Datalog_{1S} is the extension to Datalog with the successor function. In their approach, *one* temporal parameter is added to Datalog_{1S} predicates, to represent time. For instance, in their approach, the schedule of backups in a distributed system can be represented by the following rules:

$$\mathrm{backup}(T + 24, X) \leftarrow \mathrm{backup}(T, X)$$

$$\mathrm{backup}(T, Y) \leftarrow \mathrm{dependent}(X, Y),\ \mathrm{backup}(T, X)$$

The first rule states that a backup on a machine should be taken every 24 hours. The second rules requires that all backups should be taken simultaneously on all dependent machines (e.g., sharing files). Of course, Datalog_{1S} programs may have infinite least Herbrand models, so that infinite query answers may be generated. However, in their later works Chomicki and Imielinski have provided a finite representation of them.

Another influential rule-based approach is based on the adoption of Templog, an extension of logic programming based on temporal logic. In this language, predicates can vary with time, but the time point they refer to is defined implicitly by temporal operators rather than by an explicit temporal argument [2]. Specifically, three temporal operators are used in Templog: *next*, which refers to the next time instant, *always*, which refers to the present and all the future time instants, and *eventually*, which refers to the present or to some future time instant.

Constraint-Based Approaches

While deductive rule-based approaches rely on deductive database theory, the approaches of the other mainstreams apply to relational (or object-oriented) databases.

Kabanza et al. [6] have defined a *constraint-based* formalism based on the concept of *linear repeating points* (henceforth lpr's). A lrp is a set of points {x(n)} defined by an expression of the form x(n) = c + kn where *k* and *c* are integer constants and *n* ranges over the integers.

A *generalized tuple* of temporal arity *k* is a tuple with *k* temporal attributes, each one represented by a lrp, possibly including *constraints* expressed by linear equalities or disequalities between temporal attributes. Semantically, a generalized tuple denotes a (possibly infinite) set of (ordinary) tuples, one tuple for each value of the temporal attributes satisfying the lrp's definitions and the constraints. For instance, the generalized tuple

$$(a_1,\ldots, a_n|[5+4n_1, 7+4n_2]\Lambda\, X_1 = X_2 - 2)$$

(with data part $a_1,\ldots,a_n$) represents the infinite set of tuples with temporal attributes X_1 and X_2, such that $X_1 = 5 + 4n_1$, $X_2 = 7 + 4n_2$, $X_1 = X_2–2$, for some integers n_1 and n_2, i.e.,

$$\{\ldots(a_1,\ldots,a_n|[1,3]), (a_1,\ldots,a_n|[5,7]),\\ (a_1,\ldots,a_n|[9,10]),\ldots\}$$

A *generalized relation* is a finite set of generalized tuples of the same schema.

In Kabanza et al. [6], the algebraic operations (e.g., intersection) have been defined over generalized relations as mathematical manipulations of the formulae coding lrp's.

A comparative analysis of deductive rule-based and constraint-based approaches has been provided, e.g., by Baudinet et al. [2], showing that they have the same *data expressiveness* (i.e., the set of temporal databases that can be expressed in such languages is the same). Specifically, as concerns the temporal component, they express *eventually periodic* sets of points (which are, indeed, points that can be defined by *Presburger Arithmetics* – see below). In such an approach, the *query* expressiveness and the computational complexity of such formalisms have also been studied.

Symbolic Approaches

In constraint-based approaches, temporal periodicity is represented through mathematical formulae. Several authors have suggested that, although expressive, such approaches do not cope with temporal periodicity in a "commonsense" (in the sense of "human-oriented") way, arguing in favor of a *symbolic* representation, in which complex periodicities can be compositionally built in terms of simpler ones (see, e.g., the discussions in [9,13]). A milestone in the area of symbolic approaches to temporal periodicity is the early approach by Leban et al. [8]. In Leban's approach, *collections* are the core notion. A *collection* is a *structured* set of *intervals* (elsewhere called *periods*). A base collection, partitioning the whole timeline (e.g., the collection of seconds), is used as the basis of the construction. A new partition P' of the timeline (called *calendar* in Leban's terminology) can be built from another partition P as follow:

$$P' =< P; s_0, \ldots s_{n-1} >$$

where $s_0,\ldots,s_{n-1}$ are natural numbers greater than 0. Intuitively, $s_0,\ldots,s_{n-1}$ define the number of periods of P whose unions yield periods of P′, intending that the union operation has to be repeated cyclically. For example, Weeks = {Days;7} defines weeks as aggregations of seven days. Analogously, months (not considering leap years for the sake of brevity) can be defined by the expression Months = {Days; 31,28,31,30,31,30,31,31,30,31,30,31}.

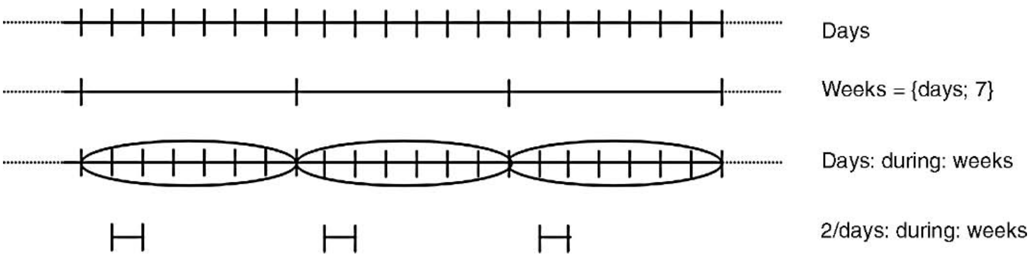
Two classes of operators, *dicing* and *slicing*, are defined in order to operate on collections.

The dicing operators provide means to further divide each period in a collection within another collection. For example, given the definitions of *Days* and *Weeks*, Day:during:Weeks breaks up weeks into the collection of days they contain. Other dicing operators are allowed (taken from a subset of Allen's relations [1]). Slicing operators provide a way of selecting periods from collections. For instance, in the expression 2/Day:during:Weeks the slicing operator "2/" is used in order to select the second day in each week. Different types of slicing operators are provided (e.g., -n/selects the *n-th* last interval from each collection).

Collection expressions can be arbitrarily built by using a combination of these operators (see, e.g., The examples in Figure 1).

While Leban's approach was mainly aimed to represent periodicity within knowledge bases (i.e., in artificial intelligence contexts), later on it has played an increasingly important role also in the area of temporal databases. Recently, Terenziani [8] has defined a temporal extension to relational databases, in which the valid time of periodic tuples can be modeled through (an extension of) Leban's language, and symbolic manipulation is used in order to perform algebraic operations on the temporal relations.

Another early symbolic approach which has been widely used within the database and artificial



Temporal Periodicity. Figure 1. Operators in Leban's language.

intelligence areas is the one by Niezette and Stevenne [9], who provided a formalism as well as the algebraic operations on it, mostly as an upper layer built upon *linear repeating points* [6]. A comparison between such a formalism and Leban's one, as well as an analysis of the relationships between the periodicities defined by such languages and *periodic granularities* has been provided by Bettini and De Sibi [3]. A recent influential symbolic approach, based on the notion of periodic granularities, has been proposed by Ning et al. [10].

An important issue concerns the formal analysis of the expressiveness (and of the semantics) of the implicit representation formalisms proposed to cope with temporal periodicity. *Presburger Arithmetics*, i.e., the first-order theory of addition and ordering over integers, is a natural reference to evaluate the expressiveness (and semantics) of such languages, because of its simplicity, decidability, and expressiveness, since it turns out that all sets definable in Presburger Arithmetics are *finite*, *periodic*, or *eventually periodic*. A recent comparison of the expressiveness of several constraint-based and symbolic approaches, based on Presburger Arithmetics, has been provided by Egidi and Terenziani [15].

While the above-mentioned approaches in [9,6,13] mainly focused on extending the relational model to cope with temporal periodicity, Kurt and Ozsoyoglu [7] devoted more attention to the definition of a *periodic temporal object oriented SQL*. They defined the temporal type of *periodic elements* to model strictly periodic and also eventually periodic events. Periodic elements consist of both an aperiodic part and a periodic part, represented by the repetition pattern and the period of repetition. They also defined the set-theoretic operations of union, intersection, complement and difference, which are closed with respect to periodic elements.

Moreover, in the last years, periodicity has also started to be studied in the context of moving objects. In particular, Revesz and Cai [11] have taken into account also periodic (called *cyclic* periodic) and eventually periodic (called *acyclic* periodic) *movements* of objects. They have proposed an extended relational data model in which objects are approximated by unions of *parametric rectangles*. Movement is coped with by modeling the x and y dimensions of rectangles through functions of the form f(t mod p), where t denotes time, p the repetition period, and mod the module function. Revesz and Cai have also defined the algebraic operations on the data model.

Key Applications

Temporal periodicity plays an important role in many application areas, including process control, data access control, office automation, data broadcasting, planning, scheduling, multimedia, active databases, banking, and so on. Languages to specify user-defined periodicities in the *queries* have been already supported by many approaches, including commercial ones. For instance, Oracle provides a language to specify periodicity in the queries to *time series* (e.g., to ask for the values of IBM at the stock exchange *each Monday*). Even more interestingly, such languages can be used in the *data*, in order to express (finite and infinite) *periodic valid times* in an implicit and compact way. However, such a move involves an in-depth revision and extension of "standard" database theory and technology, which have been partly addressed within the temporal database research community, but have not yet been fully implemented in commercial products.

Cross-references

- ▶ Allen's Relations
- ▶ Calendar
- ▶ Calendric System
- ▶ Temporal Granularity
- ▶ Time Series Query
- ▶ Valid Time

Recommended Reading

1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832–843, 1983.
2. Baudinet M., Chomicki J., and Wolper P. Temporal deductive databases. In Temporal Databases, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass (eds.). Benjamin/Cummings, 1993, pp. 294–320.
3. Bettini C. and De Sibi R. Symbolic representation of user-defined time granularities. Ann. Math. Artif. Intell., 30(1–4):53–92, 2000.
4. Chomicki J. and Imielinsky T. Temporal deductive databases and infinite objects. In Proc. 7th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1988, pp. 61–73.
5. Egidi L. and Terenziani P. A mathematical framework for the semantics of symbolic languages representing periodic time. Ann. Math. Artif. Intell., 46:317–347, 2006.
6. Kabanza F., Stevenne J.-M., and Wolper P. Handling infinite temporal data. In Proc. 9th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1990, pp. 392–403.
7. Kurt A. and Ozsoyoglu M. Modelling and querying periodic temporal databases. In Proc. 6th Int. Conf. and Workshop on Database and Expert Systems Applications, 1995, pp. 124–133.
8. Leban B., McDonald D.D., and Forster D.R. A representation for collections of temporal intervals. In Proc. 5th National Conf. on AI, 1986, pp. 367–371.
9. Niezette M. and Stevenne J.-M. An efficient symbolic representation of periodic time. In Proc. Int. Conf. on Information and Knowledge Management, 1992.
10. Ning P., Wang X.S., and Jajodia S. An algebraic representation of calendars. Ann. Math. Artif. Intell., 36(1–2):5–38, 2002.
11. Revesz P. and Cai M. Efficient querying and animation of periodic spatio-temporal databases. Ann. Math. Artif. Intell., 36(4):437–457, 2002.
12. Snodgrass R.T. and Soo M.D. *1*Supporting multiple calendars. In The TSQL2 Temporal Query Language, R.T. (ed.). R.T. Snodgrass (ed.). Kluwer, Norwell, MA, 1995, pp. 103–121.
13. Terenziani P. Symbolic user-defined periodicity in temporal relational databases. IEEE Trans. Knowl. Data Eng., 15(2): 489–509, 2003.
14. Tuzhilin A. and Clifford J. On periodicity in temporal databases. Inf. Syst., 20(8):619–639, 1995.

Temporal Projection

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Temporal assignment

Definition

In a query or update statement, *temporal projection* pairs the computed facts with their associated times, usually derived from the associated times of the underlying facts.

The generic notion of temporal projection may be applied to various specific time dimensions. For example, *valid-time projection* associates with derived facts the times at which they are valid, usually based on the valid times of the underlying facts.

Key Points

While almost all temporal query languages support temporal projection, the flexibility of that support varies greatly.

In some languages, temporal projection is implicit and is based the intersection of the times of the underlying facts. Other languages have special constructs to specify temporal projection.

The term “temporal projection” has been used extensively in the literature. It derives from the `retrieve` clause in Quel as well as the `SELECT` clause in SQL, which both serve the purpose of the relational algebra operator (generalized) projection, in addition to allowing the specification of derived attribute values.

The related concept called “temporal assignment” roughly speaking is a function that maps a set of time values to a set of values of an attribute. One purpose of a temporal assignment would be to indicate when different values of the attribute are valid.

Cross-references

- ▶ Projection
- ▶ SQL
- ▶ SQL-Based Temporal Query Languages
- ▶ Temporal Database
- ▶ Temporal Query Languages
- ▶ TSQL2
- ▶ Valid Time

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.), Springer-Verlag, Berlin, 1998, pp. 367–405.

Temporal Query Languages

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]University of Aalborg, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Historical query languages

Definition

A temporal query language is a database query language that offers some form of built-in support for the querying and modification of time-referenced data, as well as enabling the specification of assertions and constraints on such data. A temporal query language is usually quite closely associated with a temporal data model that defines the underlying data structures to which the language applies.

Historical Background

When the relational data model was proposed by Codd, he also proposed two query languages: the relational calculus and the relational algebra. Similarly, temporal data models are closely coupled with temporal query languages.

Most databases store time-referenced, or temporal, data. The ISO standard Structured Query Language SQL [4] is often the language of choice when developing applications that utilize the information captured in such databases. In spite of this, users realize that temporal data management is often challenging with SQL. To understand some of the difficulties, it is instructive to attempt to formulate the following straightforward, realistic queries, assertions, and modifications in SQL. An intermediate SQL programmer can express all of them in SQL for a database without time-referenced data in perhaps 5 minutes. However, even SQL experts find these same queries challenging to do in several *hours* when the data are time-referenced [6].

- An `Employee` table has three attributes: `Name`, `Manager`, and `Dept`. Temporal information is retained by adding a fourth attribute, `When`, of data type `PERIOD`. Attribute `Manager` is a foreign key for `Employee.Name`. This means that at each point in time, the `Manager` value of a tuple also occurs as a `Name` value (probably in a different tuple). This cannot be expressed via SQL's foreign-key constraint, which fails to take time into account. Formulating this constraint as an assertion is challenging.
- Consider the query "List those employees who are not managers." This can easily be expressed in SQL, using `EXCEPT` or `NOT EXISTS`, on the original, non-temporal relation with three attributes. Things are just a little harder with the `When` attribute; a `WHERE` predicate is required to extract the current employees. To formulate the query "List those employees who were not managers, and indicate when," `EXCEPT` and `NOT EXISTS` do not work because they do not consider time. This simple temporal query is hard to formulate, even for SQL experts.
- Consider the query "Give the number of employees in each department." Again, this is a simple SQL query using the `COUNT` aggregate when formulated on non-temporal data. To formulate the query on temporal data, i.e., "Give *the history of* the number of employees in each department," is very difficult without built-in temporal support in the language.
- The modification "Change the manager of the Tools department for 2004 to Bob" is difficult in SQL because only a portion of many validity periods are to be changed, with the information outside of 2004 being retained.

Most users know only too well that while SQL is an extremely powerful language for writing queries on the current state, the language provides much less help when writing temporal queries, modifications, and assertions and constraints.

Hence there is a need for query languages that explicitly "understand" time and offer built-in support for the management of temporal data. Fortunately, the outcomes of several decades of research in temporal query languages demonstrate that it is indeed possible to build support for temporal data management into query languages so that statements such as the above are easily formulated.

Foundations

Structure may be applied to the plethora of temporal query languages by categorizing these languages according to different concerns.

Language Extension Approaches

One attempt at bringing structure to the diverse collection of temporal query languages associates these languages with four approaches that emphasize how temporal support is being added to a non-temporal query language [1].

Abstract Data Types for Time From a design and implementation perspective, the simplest approach to improving the temporal data management capabilities of an existing query language is to introduce time data types with associated predicates and functions. This approach is common for query languages associated with temporal object-oriented databases [7].

Data types for time points, time intervals (periods), and for durations of time may be envisioned, as may data types for temporal elements, i.e., finite unions of time intervals. The predicates associated with time-interval data types are often inspired by Allen's 13 interval relationships. With reference to these, different sets of practical proposals for predicates have been proposed.

However, while being relatively easy to achieve, the introduction of appropriate time data types results only in modest improvements of the temporal data management capabilities of the query language.

Use of Point Timestamps An interval timestamp associated with a tuple in a temporal relational data model is often intended to capture the fact that the information recorded by the tuple is valid at each time point contained in the interval. This way, the interval is simply a compact representation of a set of time points. Thus, the same information can be captured by a single tuple timestamped with an interval and a set of identical tuples, each timestamped with a different time point from the interval (with no time points missing).

One attraction of intervals is that their representing is of fixed size. Another is that they appear to be very intuitive to most users — the notion of an interval is conceptually very natural, and people use it frequently in their daily thinking and interactions. In some respects, the most straightforward and simplest means of capturing temporal aspects is to use interval-valued timestamps.

However, the observation has also been advanced that the difficulty in formulating temporal queries on relations with interval-timestamped tuples stem exactly from the intervals – Allen has shown that there are 13 possible relations between a pair of intervals. It has been argued that a language such as SQL is unprepared to support something (an interval) that represents something (a convex set of time points) that it is not.

Based on this view, it is reasonable to equip an SQL extended with interval-valued timestamps with the ability to *unfold* and *fold* timestamps. The unfold function maps an interval-stamped tuple (or set of tuples) to the corresponding set of point-stamped tuples (set of tuples), and the fold function collapses a set of point-stamped tuples into the corresponding interval-stamped tuple(s). This way, it is possible to manipulate both point- and interval-stamped relations in the same language. If deemed advantageous when formulating a statement, one can effectively avoid the intervals by first unfolding all argument relations. Then the statement is formulated on the point-stamped relations. At the end, the result can be folded back into an interval-stamped format that lends itself to presentation to humans.

A more radical approach to designing a temporal query language is to completely abandon interval timestamps and use only point timestamps. This yields a very clean and simple design, although it appears that database modification and the presentation of query results to humans must still rely on intervals and thus are "outside" the approach.

The strength of this approach lies in its generalization of queries on non-temporal relations to corresponding queries on corresponding temporal relations. The idea is to extend the non-temporal query with equality constraints on the timestamp attribute of the temporal relation, to separate different temporal database states during query evaluation, thereby naturally supporting sequenced semantics.

Syntactic Defaults This approach introduces what may be termed syntactic defaults along with the introduction of temporal abstract data types, the purpose being to make the formulation of common temporal queries more convenient. Common defaults concern the access to the current state of a temporal relation and the handling of temporal generalizations of common non-temporal queries, e.g., joins. The temporal generalization of a non-temporal join is one where two tuples join if their timestamps intersect and where the timestamp of the result tuple is the intersection of the timestamps. Essentially, the non-temporal query is computed for each point in time, and the results of these queries are consolidated into a single result. The nature of the consolidation depends on the data type of the timestamps; if intervals are used, the consolidation involves coalescing.

The most comprehensive approach based on syntactic defaults is the TSQL2 language [5], but many of the earlier query languages that the creators of this language were attempting to consolidate also follow this approach. As an example, TSQL2 includes a default valid clause that computes the intersection of the valid times of the tuples in the argument relations mentioned in a statement's from clause, which is then returned in the result. So as explained above, the timestamp of a tuple that results from joining two relations is the intersection of the timestamps of the two argument tuples that produce the tuple. When there is only one argument relation, the valid clause produces the original timestamps.

Such defaults are very effective in enabling the concise formulation of common queries, but they also tend to complicate the semantics of the resulting temporal query language.

Semantic Defaults The use of semantic defaults is motivated in part by the difficulties in systematically extending a large and unorthogonal language such as SQL with syntactic defaults that are easy to understand and that do not interact in unintended ways. With this approach, so-called *statement modifiers* are added to a non-temporal query language, e.g., SQL, in order to obtain built-in temporal support [8].

It was argued earlier that statements that are easily formulated in SQL on non-temporal relations can be very difficult to formulate on temporal relations. The basic idea is then to make it easy to systematically formulate temporal queries from non-temporal queries. With statement modifiers, a temporal query is then formulated by first formulating the "corresponding" non-temporal query (i.e., assuming that there are no timestamp attributes on the argument relations) and then applying a statement modifier to this query.

For example, to formulate a temporal join, the first step is to formulate the corresponding non-temporal join. Next, a modifier is placed in front of this query to indicate that the non-temporal query is to be rendered temporal by computing it at each time point. The modifier ensures that the argument timestamps overlap and that the resulting timestamp is the intersection of the argument intervals. The attraction of using statement modifiers is that these may be placed in front of any non-temporal query to render that query temporal.

Statement modifiers are capable of specifying the semantics of the temporal queries unambiguously, independently of the syntactic complexity of the queries that the modifiers are applied to. This renders semantic defaults scalable across the constructs of the language being extended. With modifiers, the users thus need not worry about which predicates are needed on timestamps and how to express timestamps to be associated with result tuples. Further, the use of statement modifiers makes it possible to give more meaning to interval timestamps; they need no longer be simply compact representations of convex sets of time points.

Additional Characterizations of Temporal Query Languages

Several additional dimensions exist on which temporal query languages can be characterized. One is *abstractness*: is the query language at an abstract level or at a concrete level. Examples of the former are Temporal Relational Calculus and First-order Temporal Logic; an example of the latter is TSQL2.

Another dimension is *level*: is the query language at a logical or a physical level? A physical query language assumes a specific representation whereas a logical query language admits several representations. Examples of the former are examined in McKenzie's survey of relational algebras [3]; an example of the latter is the collection of algebraic operators defined on the Bitemporal Conceptual Data Model [2], which can be mapped to at least five representational models.

A third dimension is whether the query language supports a *period-stamped temporal model* or a *point-stamped temporal model*.

Other entries (indicated in *italics*) examine the long and deep research into temporal query languages in a more detailed fashion. *Qualitative temporal reasoning* and *temporal logic in database query languages* provide expressive query facilities. *Temporal vacuuming* provides a way to control the growth of a database. *TSQL2* and its successor SQL/Temporal provided a way for many in the temporal database community to coordinate their efforts in temporal query language design and implementation. *Temporal query processing* involves disparate architectures, from *temporal strata* outside the conventional DBMS to adding native temporal support within the DBMS. *Supporting transaction time* generally requires changes within the kernel of a DBMS. *Temporal algebras* extend the conventional relational algebra. Some specific operators (e.g., *temporal aggregation*, *temporal coalescing*, *temporal joins*) have received special attention. Finally, the

Oracle database management system includes support for valid and transaction time, both individually and in concert, in its extension of.

Future Directions

Given the substantial decrease in code size (a factor of three [6]) and dramatic decrease in conceptual complexity of temporal applications that temporal query languages offer, it is hoped that DBMS vendors will continue to incorporate temporal language constructs into their products.

Cross-references

- ▶ Abstract Versus Concrete Temporal Query Languages
- ▶ Allen's Relations
- ▶ Now in Temporal Databases
- ▶ Period-Stamped Temporal Models
- ▶ Point-Stamped Temporal Models
- ▶ Qualitative Temporal Reasoning
- ▶ Schema Versioning
- ▶ Sequenced Semantics
- ▶ Supporting Transaction Time Databases
- ▶ Temporal Aggregation
- ▶ Temporal Algebras
- ▶ Temporal Coalescing
- ▶ Temporal Database
- ▶ Temporal Data Models
- ▶ Temporal Joins
- ▶ Temporal Logic in Database Query Languages
- ▶ Temporal Object-Oriented Databases
- ▶ Temporal Query Processing
- ▶ Temporal Strata
- ▶ Temporal Upward Compatibility
- ▶ Temporal Vacuuming
- ▶ Temporal Visual Languages
- ▶ Temporal XML
- ▶ TSQL2

Recommended Reading

1. Böhlen M.H., Gamper J., and Jensen C.S. How would you like to aggregate your temporal data?. In Proc. 10th Int. Symp. Temporal Representation and Reasoning/4th Int. Conf. Temporal Logic, 2006, pp. 121–136.
2. Jensen C.S., Soo M.D., and Snodgrass R.T. Unifying temporal data models via a conceptual model. Inf. Syst., 19(7):513–547, December 1994.
3. McKenzie E. and Snodgrass R.T. An evaluation of relational algebras incorporating the time dimension in databases. ACM Comput. Surv., 23(4):501–543, December 1991.
4. Melton J. and Simon A.R. Understanding the New SQL: A Complete Guide. Morgan Kaufmann, San Mateo, CA, 1993.
5. Snodgrass R.T. (ed.). The TSQL2 Temporal Query Language. Kluwer, Boston, MA, USA, 1995.
6. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, San Francisco, CA, USA, 1999.
7. Snodgrass R.T. Temporal Object Oriented Databases: A Critical, Comparison, Chapter 19 in Modern Database System: The Object Model, Interoperability and Beyond, W. Kim, editor, Addison-Wesley/ACM Press, 1995, pp. 386–408.
8. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal Statement Modifiers, ACM Transactions on Database Sytems, 25(4): 407–456.

Temporal Query Processing

Michael Böhlen
Free University of Bolzano-Bozen, Bolzano, Italy

Definition

Temporal query processing refers to the techniques used by database management system to process temporal statements. This ranges from the implementation of query execution plans to the design of system architectures. This entry surveys different system architectures. It is possible to identify three general system architectures that have been used to systematically offer temporal query processing functionality to applications [6]: The *layered* approach uses an off-the-shelf database system and extends it by implementing the missing functionality in a layer between the database system and the applications. The *monolithic* approach integrates the necessary application-specific extensions directly into the database system. The *extensible* approach relies on a database system that allows to plug user-defined extensions into the database system.

Historical Background

In order to deploy systems that offer support for temporal query processing new systems must be designed and implemented. Temporal extensions of database systems are quite different from other database system extensions. On the one hand, the complexity of the newly added types, often an interval, is usually quite low. Therefore, existing access structures and query processing techniques are often deemed sufficient to manage temporal information [7,8,11]. On the other hand the temporal aspect is ubiquitous and affects all parts of a database system. This is quite different if extensions for multimedia, geographical or spatio-temporal data are considered. Such extensions deal with complex objects, for example, objects with complex boundaries, moving points, or movies, but their impact on the various components of a database system is limited.

A first overview of temporal database system implementations appeared in 1996 [2]. While the system architecture was not the focus, the overview describes the approached followed by the systems. Most systems follow the layered approach, including ChronoLog, ARCADIA, TimeDB, VT-SQL, and Tiger. A comprehensive study of a layered query processing architecture was done by Slivinskas et al. [10]. The authors use the Volcano extensible query optimizer to optimize and process queries. A monolithic approach is pursued by HDBMS, TDBMS, T-REQUIEM, and T-squared DBMS. A successful implementation of an index structure for temporal data has been done with the help of Informix's datablade technology [1].

Foundations

Figure 1 provides an overview of the different architectures that have been used to systematically provide temporal database functionality: the layered, monolithic, and extensible architectures. The gray parts denote the temporal extensions. The architecture balances initial investments, functionality, performance and lock-ins.

Functionalities

The different operators in a temporal database system have different characteristics with respect to query processing.

Temporal selection, temporal projection and temporal union resemble their non-temporal counterparts and they do not require dedicated new database functionality. Note though that this only holds if the operators may return uncoalesced relation instances [3] and if no special values, such as *now*, are required. If the operators must returned coalesced relations or if *now* must be supported the support offered by conventional database systems is lacking.

Temporal Cartesian product (and temporal joins) are also well supported by standard database systems. In many cases existing index structures can be used to index interval timestamps [7].

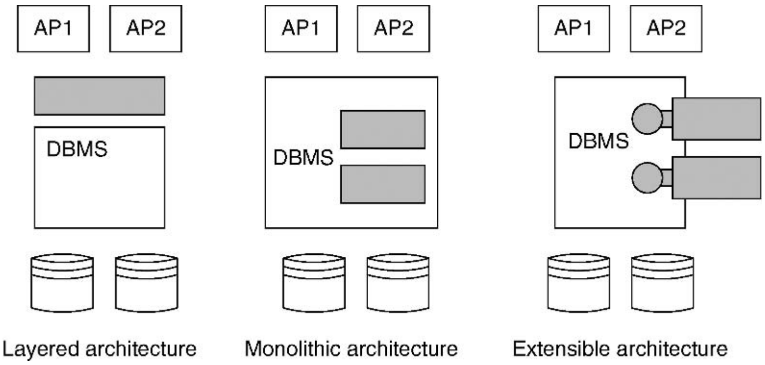
Temporal difference is more subtle and not well-supported by traditional database systems. It is possible to formulate temporal difference in, e.g., SQL, but it is cumbersome to do so. Algebraic formulations and efficient implementations have been studied by Dunn et al. [4].

Temporal aggregation is even worse than temporal difference. Formulating a temporal aggregation in SQL is a challenge for even advanced SQL programmers and yields a statement that current database systems cannot evaluate efficiently [11].

Temporal coalescing and temporal integrity constraints [11] are other examples of operations that current database system do not handle efficiently. *Temporal coalescing* is an algebraic operator and a declarative SQL implementation is inefficient.

The Layered Architecture

A common approach to design an extended database systems with new data types and operations for time-referenced data are to use an off-the-shelf database system and implement a layer on top providing data types



Temporal Query Processing. Figure 1. Illustration of architectural choices for temporal database systems and services for temporal applications. The database system with such a component is then used by different applications having similar data type and operation requirements. Database systems enhanced in this way exploit the standard data types and data model, often the relational model, as a basis. They define new data types and possibly a new layer that provides application specific support for data definition and query language, query processing and optimization, indexing, and transaction management. Applications are written against the extended interface.

The layered approach has the advantage of using standard components. There is a clear separation of responsibilities: application-specific development can be performed and supported independent of the database system development. Improvements in the database system component are directly available in the whole system with almost no additional effort. On the other hand, the flexibility is limited. Development not foreseen in the database system component has to be implemented bypassing the database system. The more effort is put into such an application-specific data management extension, the more difficult it gets to change the system and take advantage of database system improvements. Also (legacy) applications might access the database system over different interfaces. For such applications accessing the extended database system through a special purpose layer is not always an option.

The reuse of a standard database system is an advantage but at the same time also a constraint. The layer translates and delegates temporal requests to sequences of nontemporal request. If some of the functionality of the database systems should be extended or changed, e.g., a refined transaction processing for transaction time, this cannot be done easily with a layered approach. For the advanced temporal functionality the layered architecture might not offer satisfactory performance.

The Monolithic Architecture

Many systems that use a monolithic architecture have originally been designed as stand-alone applications without database functionality. The designers of a monolithic architecture then extend their system with database system functionality. They add query functionality, transaction management, and multi-user capabilities, thereby gradually creating a specialized database system. The data management aspects traditionally associated with database system and the application-specific functionality are integrated into one component.

Instead of adding general database system functionality to an application it is also possible to incorporate the desired application domain semantics into the database system. Typically, this is done by database companies who have complete control over and knowledge of their source code or by open source communities.

Because of the tight integration of the general data management aspects and the application specific functionality, monolithic systems can be optimized for the specific application domain. This results in good performance. Standard and specialized index structures can be combined for good results. Transaction management can be provided in a uniform way for standard as well as new data types. However, implementing a monolithic system is difficult and a big (initial) effort, since all aspects of a database system have to be taken into account. Another drawback is that enterprises tend to be reluctant to replace their database system, which increases the threshold for the adoption of the temporal functionality. With monolithic systems, there is is a high risk of vendor lock-ins.

The Extensible Architecture

Extensible database systems can be extended with application-specific modules. Traditional database functionality like indexing, query optimization, and transaction management is supported for new data types and functions in a seamless fashion.

The first extensible system prototypes have been developed to support non-standard database system applications like geographical, multimedia or engineering information systems. Research on extensible systems has been carried out in several projects, e.g., Ingres [12], Postgres[13], and Volcano [5]. These projects addressed, among other, data model extensions, storage and indexing of complex objects as well as transaction management and query optimization in the presence of complex objects. Today a number of commercial approaches are available, e.g., datablades from Informix, cartridges from Oracle, and extenders from DB2. A limitation is that the extensions only permit extension that were foreseen initially. A comprehensive temporal support might require support that goes beyond data types and access structures.

The SQL99 standard [9] specifies new data types and type constructors in order to better support advanced applications.

The extensible architecture balances the advantages and disadvantages of the layered and monolithic architectures, respectively. There is a better potential to implement advanced temporal functionality with a satisfactory performance than in the layered architecture. However, functionality not foreseen by the extensible database system might still be difficult to implement.

Key Applications

All applications that want to provide systematic support for time-varying information must choose one of the basic system architectures. The chosen architecture balances (initial) investments, future lock-ins, and performance.

Future Directions

The architecture determines the initially required effort to provide support for time-varying information and may limit functionality and performance. Changing from one architecture to another is not supported. Such transitions would be important to support the graceful evolution of temporal database applications.

Cross-references

- ▶ Temporal Data Model
- ▶ Temporal Database
- ▶ Temporal Strata

Recommended Reading

1. Bliujute R., Saltenis S., Slivinskas G., and Jensen C.S. Developing a datablade for a new index. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 314–323.
2. Böhlen M.H. Temporal database system implementations. ACM SIGMOD Rec., 24(4):16, December 1995.
3. Böhlen M.H., Snodgrass R.T., and Soo M.D. Coalescing in temporal databases. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 180–191
4. Dunn J., Davey S., Descour A., and Snodgrass R.T. Sequenced subset operators: definition and implementation. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 81–92.
5. Graefe G. and McKenna W.J. The volcano optimizer generator: extensibility and efficient search. In Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 209-218.
6. Koubarakis M., Sellis T.K., Frank A.U., Grumbach S., Güting R.H., Jensen C.S., Lorentzos N.A., Manolopoulos Y., Nardelli E., Pernici B., Schek H., Scholl M., Theodoulidis B., and Tryfona N. (eds.). Spatio-Temporal Databases: The CHOROCHRONOS Approach. Springer, Berlin, 2003.
7. Kriegel H.-P., Pötke M., and Seidl T. Managing intervals efficiently in object-relational databases. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 407-418.
8. Leung T.Y.C. and Muntz R.R. Stream processing: temporal query processing and optimization. In Tansel A., Clifford J., Gadia S., Jajodia S., Segev A., Snodgrass R.T. (eds.). Temporal Databases: Theory, Design, and Implementation, Benjamin/Cummings, 1993, pp. 329-355.
9. Melton J. and Simon A.R. Understanding the New SQL: A Complete Guide. Morgan Kaufmann, Los Altos, CA, 1993.
10. Slivinskas G., Jensen C.S., and Snodgrass R.T. Adaptable query optimization and evaluation in temporal middleware. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 127–138.
11. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, Los Altos, CA, 1999.
12. Stonebraker M. (ed.). The INGRES Papers: Anatomy of a Relational Database System. Addison-Wesley, Reading, MA, 1986.
13. The Postgresql Global. POSTGRESQL developer's guide.

Temporal Relation

▶ Bitemporal Relation

Temporal Relational Calculus

Jan Chomicki[1], David Toman[2]

[1]State University of New York at Buffalo, Buffalo, NY, USA

[2]University of Waterloo, Waterloo, ON, Canada

Synonyms

Two-sorted first-order logic

Definition

Temporal Relational Calculus (TRC) is a temporal query language extending the relational calculus. In addition to data variables and quantifiers ranging over a data domain (a universe of *uninterpreted constants*), temporal relational calculus allows *temporal variables* and quantifiers ranging over an appropriate time domain [1].

Key Points

A natural temporal extension of the relational calculus allows explicit variables and quantification over a given time domain, in addition to the variables and quantifiers over a data domain of uninterpreted constants. The language is simply the two-sorted version (variables and constants are temporal or non-temporal) of first-order logic over a data domain D and a time domain T.

The syntax of the two-sorted first-order language over a database schema $\rho = \{R_1,...,R_k\}$ is defined by the grammar rule:

$$\begin{aligned} Q ::= R(t_i, x_{i_1},..., x_{i_k}) \mid t_i < t_j \mid x_i = x_j \mid \\ Q \wedge Q \mid \neg Q \mid \exists x_i.Q \mid \exists t_i.Q \end{aligned}$$

In the grammar, t_i's are used to denote temporal variables and x_i's to denote data (non-temporal) variables. The atomic formulae $t_i < t_j$ provide means to refer to the underlying ordering of the time domain. Note that the schema ρ contains schemas of *time-stamped temporal relations*.

Given a point-timestamped database DB and a two-sorted valuation θ, the semantics of a TRC query Q is defined in the standard way (similarly to the semantics of relational calculus) using the *satisfaction relation* $DB, \theta \vDash Q$:

$$\begin{array}{ll} DB, \theta \vDash R_j(t_i, x_{i_1},..., x_{i_k}) & \text{if } R_j \in \rho \text{ and } (\theta(\mathrm{t_i}), \\ & \theta(x_{i_1}),..., \theta(x_{i_k})) \in R_j^{DB} \\ DB, \theta \vDash t_i < t_j & \text{if } \theta(t_i) < \theta(t_j) \\ DB, \theta \vDash x_i = x_j & \text{if } \theta(x_i) = \theta(x_j) \\ DB, \theta \vDash Q_1 \wedge Q_2 & \text{if} DB, \theta \vDash Q_1 \text{ and} \\ & DB, \theta \vDash Q_2 \\ DB, \theta \vDash \neg Q_1 & \text{if not } DB, \theta \vDash Q_1 \\ DB, \theta \vDash \exists t_i.Q_1 & \text{if there is } s \in T \text{ such} \\ & \text{that } DB, \theta[t_i \mapsto s] \vDash Q_1 \\ DB, \theta \vDash \exists x_i.Q_1 & \text{if there is } a \in D \text{ such} \\ & \text{that } DB, \theta[x_i \mapsto a] \vDash Q_1 \end{array}$$

where R_j^{DB} is the interpretation of the predicate symbol R_j in the database DB.

The *answer to a query Q over DB* is the set $Q(DB)$ of valuations that make Q true in DB. Namely, $Q(DB) := \{\theta_{|FV\ (Q)} : DB,\ \theta \vDash Q\}$ where $\theta_{|FV\ (Q)}$ is the restriction of the valuation θ to the free variables of Q.

In many cases, the definition of TRC imposes additional restrictions on valid TRC queries:

Restrictions on free variables: Often the number of free temporal variables in TRC queries can be restricted to guarantee closure over the underlying data model (e.g., a single-dimensional timestamp data model or the bitemporal model). Note that this restriction applies only to queries, not to subformulas of queries. **Range restrictions**: Another common restriction is to require queries to be range restricted to guarantee domain independence. In the case of TRC (and many other abstract query languages), these restrictions depend crucially on the chosen *concrete encoding* of temporal databases. For example, no range restrictions are needed for temporal variables when queries are evaluated over interval-based database encodings, because the complement of an interval can be finitely represented by intervals.

The schemas of atomic relations, $R_j(t_i, x_{i_1},..., x_{i_k})$, typically contain a single temporal attribute/variable, often in fixed (e.g., first) position: This arrangement simply reflects the choice of the underlying temporal data model to be the single-dimensional valid time model. However, TRC can be similarly defined for multidimensional temporal data models (such as the bitemporal model) or for models without a predefined number of temporal attributes by appropriately modifying or relaxing the requirements on the structure of relation schemas.

An interesting observation is that a variant of TRC, in which temporal variables range over *intervals* and that utilizes *Allen's interval relations* as basic comparisons between interval values, is equivalent to TRC over two-dimensional temporal relations, with the two temporal attributes standing for interval endpoints.

Cross-references

▶ Abstract Versus Concrete Temporal Query Languages
▶ Point-Stamped Temporal Models
▶ Relational Calculus
▶ Relational Model
▶ Temporal Logic in Database Query Languages
▶ Temporal Query Languages
▶ Temporal Relation
▶ Time Domain
▶ Time Instant
▶ TSQL2
▶ Valid Time

Recommended Reading

1. Chomicki J. and Toman D. Temporal databases. In Handbook of Temporal Reasoning in Artificial Intelligence. M. Fischer, D. Gabbay, and L. Villa Foundations of Artificial Intelligence. Elsevier, New York, NY, USA, 2005, pp. 429–467.

Temporal Restriction

▶ Temporal Specialization

Temporal Semi-Structured Data

▶ Temporal XML

Temporal Specialization

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Temporal restriction

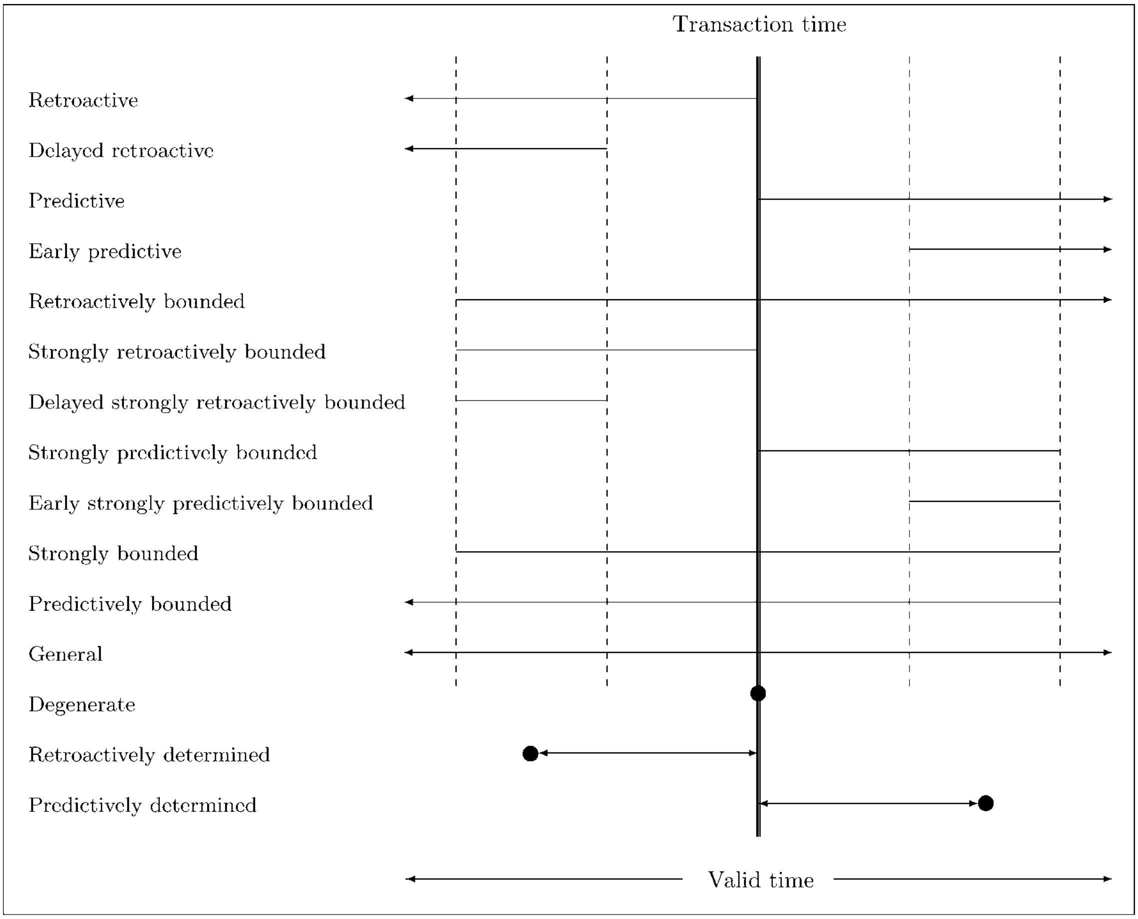
Definition

Temporal specialization denotes the restriction of the interrelationships between otherwise independent (implicit or explicit) timestamps in temporal relations. An example is a relation where tuples are always inserted after the facts they record were valid in reality. In such a relation, the transaction time of a tuple would always be after the valid time. Temporal specialization may be applied to relation schemas, relation instances, and individual tuples.

Key Points

Data models exist where relations are required to be specialized, and temporal specializations often constitute important semantics about temporal relations that may be utilized for, e.g., improving the efficiency of query processing.

Temporal specialization encompasses several kinds of specialization. One is based on the relationships



Temporal Specialization. Figure 1. Temporal Specialization based on isolated events – restrictions on the valid timestamp relative to the transaction timestamp. Adapted from Jensen and Snodgrass (1994).

between isolated events, and one based on inter-event relationships. Two additional kinds consider intervals instead of events, and one is based on the so-called completeness of the capture of the past database states.

The taxonomy based on *isolated events*, illustrated in Fig. 1, considers the relationship between a single valid time and a single transaction time. For example, in a *retroactive* relation, an item is valid before it is operated on (inserted, deleted, or modified) in the database. In a *degenerate* relation, there is no time delay between sampling a value and storing it in the database. The valid and transaction timestamps for the value are identical.

The interevent-based taxonomy is based on the *interrelationships* among multiple event timestamped items, and includes non-decreasing, non-increasing, and sequential. Regularity is captured through the categories of transaction time event regular, valid time event regular, and temporal event regular, and strict versions of these. The interinterval-based taxonomy uses Allen's relations.

To understand the last kind of specialization, which concerns the *completeness* of the capture of past states, recall that a standard transaction — time database captures all previously current states – each time a database modification occurs, a new previously current state is created. In contrast a valid-time database captures only the current database state. In-between these extremes, one may envision a spectrum of databases with incomplete support for transaction time. For example, consider a web archive that takes a snapshot of a collection of web sites at regular intervals, e.g., every week. If a site was updated several times during the same week, states would be missing from the database. Such incomplete databases are considered specializations of more complete ones.

Concerning the synonym, the chosen term is more widely used than the alternative term. The chosen term indicates that specialization is done with respect to the temporal aspects of the data items being timestamped. It is natural to apply the term temporal generalization to the opposite of temporal specialization. "Temporal restriction" has no obvious opposite term.

Cross-references

▶ Allen's Relations
▶ Bitemporal Relation
▶ Temporal Database
▶ Temporal Generalization
▶ Transaction Time
▶ Valid Time

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin, 1998, pp. 367–405.
2. Jensen C.S. and Snodgrass R.T. Specialized temporal relations. In Proc. 8th Int. Conf. on Data Engineering, 1992, pp. 594–603.
3. Jensen C.S. and Snodgrass R.T. Temporal specialization and generalization. IEEE Trans. Knowl. Data Eng., 5(6):954–974, 1994.

Temporal Strata

Kristian Torp
Aalborg University, Aalborg, Denmark

Synonyms

Temporal layer; Layered architecture; Temporal middleware; Wrapper

Definition

A temporal stratum is an architecture for implementing a temporal DBMS. The stratum is a software layer that sits on top of an existing DBMS. The layer translates a query written in a temporal query language into one or more queries in a conventional query language (typically SQL). The translated queries can then be executed by the underlying DBMS. The DBMS returns the result of the query/queries to the user directly or via the stratum. The core idea of the stratum is to provide new temporal query functionality to the users without changing the underlying DBMS. A temporal stratum can be implemented as a simple translator (temporal SQL to standard SQL) or as an advanced software component that also does part of the query processing and optimization. In the latter case, the temporal stratum can implement query processing algorithms that take the special nature of temporal data into consideration. Examples are algorithms for temporal join, temporal coalescing, and temporal aggregation.

Historical Background

Applications that store and query multiple versions of data have existed for a very long time. These applications have been implemented using, for example, triggers and log-like tables. Supporting multiple versions of data can be time consuming to build and computationally intensive to execute, since the major DBMS vendors only have limited temporal support. See [8] for details.

Parallel to the work in the software industry the research community has proposed a large number of temporal data models and temporal query languages. For an overview see [7]. However, most of this research is not supported by implementations. A notable exception to this is the Postgres DBMS that has built-in support for transaction time [6]. Postgres has later evolved into the popular open-source DBMS PostgreSQL that does not have transaction-time support. Most recently, the Immortal DB research prototype [2] has looked at how to built-in temporal support in the Microsoft SQL Server DBMS.

The DBMS vendors have not been interested in implementing the major changes to the core of their DBMSs that are needed to add temporal support to the existing DBMSs. In addition, the proposal to add temporal support to SQL3, called "SQL/Temporal" part 7 of the ISO SQL3 standard, has had very limited support. The temporal database research community has therefore been faced with a challenge of how to experimentally validate their proposals for new temporal data models and temporal query languages since it is a daunting task to build a temporal DBMS from scratch.

To meet this challenge it has been proposed to implement a temporal DBMS as a software layer on top of an existing DBMS. This has been termed a temporal stratum approach. Some of the first proposals for a temporal stratum mainly consider translating a query in a temporal query language to one or more queries in SQL [9].

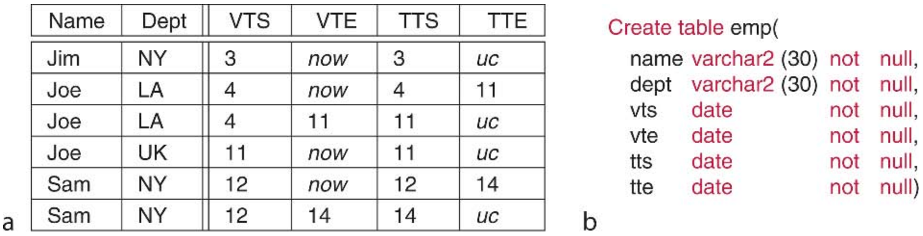
The temporal database research community has shown that some temporal queries are very inefficient to execute in plain SQL, either formulated directly in SQL or translated via a simple temporal stratum from a temporal query language to SQL. For this reason it has be researched how to make the temporal stratum approach more advanced such that it uses the underlying DBMS when this is efficient and does the query execution in the layer when the underlying DBMS is found to be inefficient [5,4].

Scientific Fundamentals

A bitemporal database supports both valid time and transaction time. In the following it is assumed that all tables have bitemporal support. Figure 1a shows the bitemporal table `emp`. The table has two explicit columns, `name` and `dept`, and four implicit timestamp columns: `VTS`, `VTE`, `TTS`, and `TTE`. The first row in the table says that Jim was in the New York department from the third (assuming the month of September 2007) and is still there, indicated by the variable *now*. This information was also entered on the third and is still considered to be the best valid information, indicated by the variable *uc* that means until changed.

It is straightforward to implement a bitemporal table in a conventional DBMS. The implicit attributes are simply made explicit. For the `emp` table an SQL table with six columns are created. This is shown in Fig. 1b. Existing DBMSs do not support variables and therefore the temporal stratum has to convert the variables *now* and *uc* to values in the domain of columns `VTE` and `TTE`. It has been shown that it is the most convenient to use the maximum value (9999-12-31) in the `date` domain for both *now* and *uc* [10].

Primary key and unique key constraints are quite complicated to implement in a temporal stratum.



Name	Dept	VTS	VTE	TTS	TTE
Jim	NY	3	*now*	3	*uc*
Joe	LA	4	*now*	4	11
Joe	LA	4	11	11	*uc*
Joe	UK	11	*now*	11	*uc*
Sam	NY	12	*now*	12	14
Sam	NY	12	14	14	*uc*

```
Create table emp(
  name varchar2 (30) not  null,
  dept varchar2 (30) not  null,
  vts  date          not  null,
  vte  date          not  null,
  tts  date          not  null,
  tte  date          not  null)
```

Temporal Strata. Figure 1. (a) The Bitemporal Table `emp` (b) SQL Implementation.

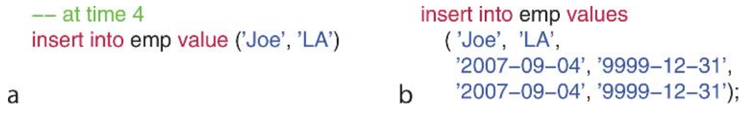
A temporal query language such as ATSQL [1] has to be temporal upwards compatible, i.e., all non-temporal SQL statements have to work as before. For primary keys this means that a primary key on a bitemporal table cannot be directly mapped to a primary key in the underlying DBMS. As an example, if the `emp` only has to store the current version of where employees are, the `name` column can be used as a primary key. However, since `emp` has bitemporal support there are now three rows in the example table where the name is "Joe." Due to space constraints, an example is not listed here. Please see [8] for a concrete example.

To look at how modifications are handled in a temporal stratum, first consider the temporal insert statement shown in Fig. 2a. This is a temporal upward compatible insert statement (it looks like a standard SQL statement). The mapping to SQL is shown in Fig. 2b. The columns `vts` and `tts` are set to the current date (fourth of September 2007). The *now* and *uc* variables are set to the maximum `date`. The result is the second row in Fig. 1a.

At time 11 Joe is updated from being in the LA department to the UK department. The temporal upwards compatible update statement for this is shown in Fig. 3a. This update statement is mapped to an SQL update of the existing row and two SQL insert statements. The update ends the current belief by updating the `TTE` column to the current date. The first SQL insert statement stores for how long it was believed that Joe was in the LA department. The second SQL insert statement stores the new belief that Joe is in the UK department. The temporal update corresponds to the updated second row plus the third and fourth row in Fig. 1a. A delete statement is mapped like the SQL update statement and the first SQL insert statement in Fig. 3b. In Fig. 1a a temporal insert of Sam at time 12 and a temporal delete of Sam at time 14 are shown as rows number 5 and 6. Note that the SQL delete statement is never used for mapping temporal modification statements. For a complete coverage of implementing temporal modification statements in a temporal stratum, please see [10].

It is possible to see past states of the database. In particular the following will look at the `emp` table as of the 13th. This is called a time slicing, i.e., the database is rewound to 13th to see the content of the database as of this date.

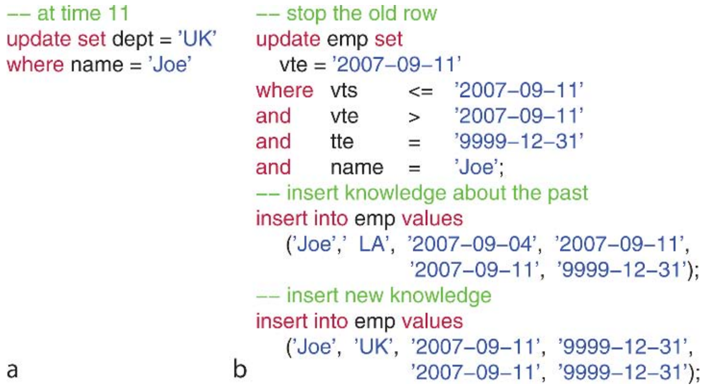
The ATSQL query in Fig. 4a is a sequenced query that selects the explicit attribute and the valid-time attributes (`vts` and `vte`). The equivalent SQL query translated by a temporal stratum is shown in Fig. 4b. The result of the query is shown in Fig. 4c (using *now*



```
-- at time 4
insert into emp value ('Joe', 'LA')
```

```
insert into emp values
  ( 'Joe',  'LA',
    '2007-09-04', '9999-12-31',
    '2007-09-04', '9999-12-31');
```

Temporal Strata. Figure 2. (a) Temporal Insert (b) Mapping to SQL.



```
-- at time 11
update set dept = 'UK'
where name = 'Joe'
```

```
-- stop the old row
update emp set
   vte = '2007-09-11'
where  vts      <=   '2007-09-11'
and    vte      >    '2007-09-11'
and    tte      =    '9999-12-31'
and    name     =    'Joe';
-- insert knowledge about the past
insert into emp values
    ('Joe',' LA', '2007-09-04', '2007-09-11',
                  '2007-09-11', '9999-12-31');
-- insert new knowledge
insert into emp values
    ('Joe', 'UK', '2007-09-11', '9999-12-31',
                  '2007-09-11', '9999-12-31');
```

Temporal Strata. Figure 3. (a) temporal update (b) mapping to SQL.

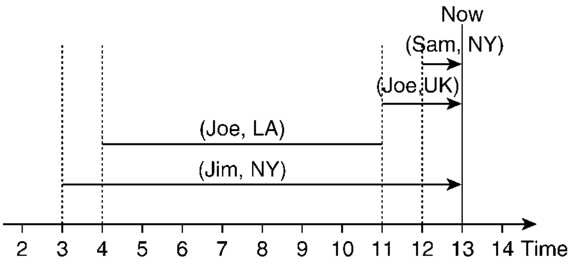
a
```
sequenced select *
from emp
as of '2007-09-13';
```

b
```
select name, dept, vts, vte
from   emp
where  tts   <=  '2007-09-13'
and    tte   >   '2007-09-13'
```

c

Name	Dept	VTS	VTE
Jim	NY	3	*now*
Joe	LA	4	11
Joe	UK	11	*now*
Sam	NY	12	*now*

Temporal Strata. Figure 4. Transaction-time slicing.



Temporal Strata. Figure 5. Visual representation of the content in Fig. 4c.

a
```
Sequenced validtime
select count (Name)
from emps
```

b

Count	VTS	VTE
1	3	4
2	4	11
2	11	12
3	12	*now*

Temporal Strata. Figure 6. Temporal aggregation

instead of the maximum date). The temporal SQL query is only slightly simpler than the equivalent standard SQL query.

To see the benefits of a temporal stratum it is necessary to look at more complicated queries. In the following, the focus is on temporal aggregation queries. Alternative complicated queries are temporal join or temporal coalescing.

Assume that a boss wants to see how many employees a company has had over (valid) time looking at the database as of the 13th. The result shown in Fig. 4c is used as an input for this query. For convenience it is assumed that this timeslice query is converted to a view call `empAt13`.

The content of the database is illustrated in Fig. 5, where the vertical dotted lines indicates the time where the result has to be split. The temporal query that expresses this is shown in Fig. 6a and the result of the query is shown in Fig. 6b. Note that the result is not coalesced.

To execute this query the temporal stratum has to find the constant periods, i.e., the periods where the count is the same. Here the temporal stratum can do either direct conversion to standard SQL or do part of the query processing in the stratum. The direct converted query is listed in Fig. 7. The standard SQL is very complicated compared to the equivalent temporal SQL query in Fig. 6a. The benefit for the user should be obvious. In line 1 of Fig. 7, the count and the valid-time start and end associated with the count are selected. In line 2 the view `empat13` from Fig. 4b is used. In addition, the `const_period` is introduced in lines 2–39. In line 40, only those periods that overlap the group currently being considered are included. In line 41, the groups are formed based on the valid-time start and valid-time end. Finally, in line 42 the output is listed in the valid-time start order.

The next question is then the efficiency of executing the query in the underlying DBMS or doing part of the query processing in the temporal stratum. It has been experimentally shown that for temporal aggregation, it can be up to ten times more efficient to do part of the query processing in a temporal stratum [5].

A different approach to adding temporal support to an existing DBMS is to use an extensible DBMS such as IBM Informix, Oracle, or DB2. This is the approach taken in [11]. Here temporal support is added to IBM Informix. Compared to a stratum approach it is not possible in the extension approach to use a new temporal SQL. The temporal extension are accessed by the user via new operators or function calls.

Key Applications

There is no support for valid-time or transaction-time in existing DBMSs. However, in Oracle 10g there is a flashback option [3] that allows a user to see the state

```
select count(name), const_period.vts as vts, const_period.vte as vte
from   empat13, (select t1.vts as vts, t1.vte as vte
                 from   empat13 t1
                 where  not exists ( select *
                                    from   empat13 t2
                                    where (t1.vts < t2.vts and t2.vts < t1.vte) or
                                          (t1.vts < t2.vte and t2.vte < t1.vte))
                 union
                 select t1.vts as vts, t2.vts as vte
                 from   empat13 t1, empat13 t2
                 where  t1.vts < t2.vts and t2.vts < t1.vte
                 and    not exists (select *
                                    from   empat13 t3
                                    where  (t1.vts < t3.vts and t3.vts < t2.vts) or
                                           (t1.vts < t3.vte and t3.vte < t2.vts))
                 union
                 select t1.vts as vts, t2.vte as vte
                 from   empat13 t1, empat13 t2
                 where  t1.vts < t2.vte and t2.vte < t1.vte
                 and    not exists (select *
                                    from   empat13 t3
                                    where  (t1.vts < t3.vts and t3.vts < t2.vte) or
                                           (t1.vts < t3.vte and t3.vte < t2.vte))
                union
                select t1.vte as vts, t2.vts as vte
                from   empat13 t1, empat13 t2
                where  t1.vte < t2.vts
                and    not exists (select *
                                   from   empat13 t3
                                   where  (t1.vte < t3.vts and t3.vts < t2.vts) or
                                          (t1.vte < t3.vte and t3.vte < t2.vts))
                union
                select t1.vte as vts, t2.vte as vte
                from   empat13 t1, empat13 t2
                where  t2.vts < t1.vte and t1.vte < t2.vte
                and    not exists (select *
                                   from   empat13 t3
                                   where  (t1.vte < t3.vts and t3.vts < t2.vte) or
                                          (t1.vte < t3.vte and t3.vte < t2.vte))) const_period
where empat13.vts <= const_period.vts and const_period.vte <= empat13.vte
group by const_period.vts, const_period.vte
order by const_period.vts
```

Temporal Strata. Figure 7. Temporal aggregation in standard SQL.

```
flashback table emp to timestamp (‘2007-09-01 08:00:00’);
```

Temporal Strata. Figure 8. A flashback in the Oracle DBMS.

of the entire database or a single table as of a previous instance in time. As an example, the flashback query in Fig. 8 will get the state of the `emp` table as of the 1st of September 2007 at 08.00 in the morning. The result can be used for further querying.

Future Directions

The Sarbanes-Oxley Act is a US federal law that requires companies to retain all of there data for a period of five or more years. This law may spur additional research with temporal databases and in particular temporal database architecture such as a temporal stratum.

Cross-references

- ► Databases
- ► Supporting Transaction Time
- ► Temporal Data Model

▶ Temporal Query Languages
▶ Temporal Query Processing

Recommended Reading

1. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. ACM Trans. Database Syst., 25(4):407–456, 2000.
2. Lomet D., Barga R., Mokbel M.F., Shegalov G., Wang R., and Zhu Y. Transaction time support inside a database engine. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
3. Oracle Corp. Oracle Flashback Technology. http://www.oracle.com/technology/deploy/availability/htdocs/Flashback_Overview.htm, as of 4.9.2007.
4. Slivinskas G. and Jensen C.S. Enhancing an extensible query optimizer with support for multiple equivalence types. In Proc. 5th East European Conf. Advances in Databases and Information Systems, 2001, pp. 55–69.
5. Slivinskas G., Jensen C.S., and Snodgrass R.T. Adaptable query optimization and evaluation in temporal middleware. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 127–138.
6. Stonebraker M. and Rowe L.A. The design of POSTGRES. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 340–355.
7. Snodgrass R.T. The TSQL2 Temporal Query Language. Kluwer Academic, Dordrecht, 1995.
8. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, 1999.
9. Torp K., Jensen C.S., and Snodgrass R.T. Stratum approaches to temporal DBMS implementation. In Proc. Int. Conf. on Database Eng. and Applications, 1998, pp. 4–13.
10. Torp K., Jensen C.S., and Snodgrass R.T. Effective timestamping in databases. VLDB J., 8(3–4):267–288, 2000.
11. Yang J., Ying H.C., and Widom J. TIP: a temporal extension to informix. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000.

Temporal Structure

▶ Time Domain

Temporal Type

▶ Temporal Granularity

Temporal Upward Compatibility

▶ Current Semantics

Temporal Vacuuming

John F. Roddick[1], David Toman[2]
[1]Flinders University, Adelaide, SA, Australia
[2]University of Waterloo, Waterloo, ON, Canada

Synonyms

Data expiration

Definition

Transaction-time temporal databases are inherently append-only resulting, over time, in a large historical sequence of database states. Data vacuuming allows for a strategic, and irrevocable, deletion of obsolete data.

Historical Background

The term *vacuuming* was first used in relation to databases in the Postgres database system as a mechanism for moving old data to archival storage [10]. It was later refined by Jensen and Mark in the context of temporal databases to refer to the removal of *obsolete* information [4] and subsequently developed into a comprehensive and usable adjunct to temporal databases [6,8,11]. Data expiration has also been investigated in the context of data warehouses by Garcia-Molina et al. [2] and others [9].

Foundations

In many applications, data about the past needs be retained for further use. This idea can be formalized, at least on the conceptual level, in terms of an append-only transaction-time temporal database or a history. However, a naive and unrestricted storage of all past data inevitably leads to unreasonable demands on storage and subsequently impacts negatively on efficiency of queries over such histories. Hence techniques that allow selective removal of *no longer needed* data have been developed and, at least in prototype systems, deployed.

The parts of the historical data that are to be retained/deleted are specified in terms of *vacuuming specifications*. These specifications state, for example, that data beyond certain absolute or relative time point is *obsolete* (as opposed to merely *superceded*) and thus can be removed. For example, the regulation

▶ *"Taxation data must be retained for the last 5 years"*

can be considered a specification of what data individuals must retain concerning their taxation returns and what can be discarded. However, one must be careful when designing such specifications as once a part of the history is deleted, it can no longer be reconstructed from the remaining data. Consider the alternative regulation

► *"Taxation data must be retained for past years, except for the last year."*

While this specification seems to suggest that the data of the last year can be discarded, doing so would lead to a problem in the following year (as this year's return won't be the last one any more). Hence, this specification is intuitively not well formed and must be avoided. Vacuuming specifications can be alternatively phrased in terms of what may be deleted, rather than what should be retained. For example, rather than

► *"Taxation data must be retained for the last 5 years"*

it would be better to rephrase it as

► *"Taxation data over 5 years old may be deleted."*

The reason for this is that vacuuming specifications indicate specific deletion actions and there is thus less chance of misinterpretation.

Another issue with vacuuming specifications relates to the granularity of data that is removed from the history. For example, if data items (such as tax receipts) are temporally correlated with other items that may appear in different, perhaps much older, parts of the history (such as investments), those parts of the history may have to be retained as well. Such considerations must be taken into account when deciding whether the specifications are allowed to refer to the complete history or whether selective vacuuming of individual data items is permitted. In both cases, issues of data integrity need to be considered.

Formal Vacuuming Specifications

A transparent way to understand the above issues is to consider the result of applying a vacuuming specification to a history (i.e., the *retained* data) to be a view defined on the original history.

Definition. Let $H = \langle S_0, S_1, \ldots, S_k \rangle$ be a history. The instances S_i represent the state of the data at time i and all states share a common fixed schema. $\mathbf{T}_H$ and $\mathbf{D}_H$ are used to denote the *active* temporal and data domains of H, respectively. A *vacuuming specification* is a function (a view) $E : H \rightarrow H'$, where H' is called the *residual history* (with some, potentially different but fixed schema).

The idea behind this approach is that the instance of the view represents the *result* of applying the vacuuming specification to the original history and *it is this instance that has to be maintained* in the system. While such view(s) usually map histories to other histories (sometimes called the *residual histories*), in principle, there is no restriction on the schema of these views nor on the language that defines the view. This approach allows us to address the two main questions concerning a vacuuming specification:

- **Is a given specification well formed?** The first question relates to anomalies such as the one outlined in the introductory example. Since only the instance of the view and not the original history itself is *stored*, a new instance of the view must be definable in terms of the current instance of the view $E(H)$ whenever a new state of the history S is created by progression of time. This condition can be formalized by requiring:
$$E(H; S) = \Delta(E(H), S)$$
for some function (query) Δ where $H;S$ is the extension of H with a new state S. To start this process, a constant, $\emptyset$, is technically needed to represent the instance of the view in the beginning (i.e., for an empty initial history). The condition above essentially states that the view E must be *self-maintainable* in terms of the pair $(\emptyset,\Delta)$. The pair $(\emptyset,\Delta)$ is called a *realization* of E.
- **What queries does a specification support?** The second question concerns which queries can be correctly answered over the residual histories. Again, for a query Q to be answerable, it must be the case that
$$Q(H) = Q'(E(H))$$
for some function (query) Q' and all histories H. Q' is a *reformulation* of Q with respect to E. This requirement states that queries preserved by the vacuuming process are exactly those that can be answered only using the view E.

In addition, for the approach to be *practical*, the construction of Δ and $\emptyset$ from E and of Q' from Q and E, respectively, must be effective.

Definition. A vacuuming specification represented by a self-maintainable view E is a *faithful history encoding for a query* Q if Q is answerable using the view E.

Given a vacuuming specification E over a history H that is self-maintainable using $(\emptyset,\Delta)$ and a query Q' that answers Q using E; the triple $(\emptyset,\Delta,Q')$ is then called the *expiration operator* of Q for H.

Space/Storage Requirements

Understanding vacuuming specifications in terms of self-maintainable materialized views also provides a natural tool for comparing different specifications with respect to *how well they remove unnecessary data*. This can be measured by studying the size of the instances of E with respect to several parameters of H:

- The size of the history itself, $|H|$,
- The size of the active data domain, $|\mathbf{D}_H|$, and
- The length of the history, $|\mathbf{T}_H|$.

In particular, the dependency of $|E(H)|$ on $|\mathbf{T}_H|$ is important as the progression of time is often the major factor in the size of H. It is easy to see that vacuuming specification with a *linear* bound in terms of $\mathbf{T}_T$ always exists: it is, e.g., the identity used to define both E and Q'. However, such a specification is not very useful and better results can be probably achieved using standard *compression algorithms*. Therefore the main interest is in two main cases defined in terms of $|\mathbf{T}_H|$:

1. Specifications bounded by O(1), and
2. Specifications bounded by $O(log(|\mathbf{T}_H|))$.

In the first case the vacuuming specification provides a *bounded encoding of a history*. Note that in both cases, the size of $E(H)$ will still depend on the other parameters, e.g., $|\mathbf{D}_H|$. This, however, must be expected, as intuitively, the more H refers to different constants (individuals), the larger $E(H)$ is likely to be (for example, to store the *names* of the individuals).

Vacuuming in Valid-Time Databases. In contrast to transaction-time temporal databases (or histories), valid-time temporal databases allow *arbitrary* updates of the temporal data. Hence information about future can be recorded and data about the past can be modified and/or deleted. This way, vacuuming specifications reduce to appropriate updates of the valid time temporal database.

Moreover, when allowing arbitrary updates of the database, it is easy to show that the only *faithful history encoding*s are those that are lossless (in the sense that H can be reconstructed from the instance of E).

Example. Consider a valid time temporal database H with a schema $\{R\}$ and a query Q asking *"return the contents of the last state of R recorded in H."* Then, for a vacuuming specification E to be a *faithful encoding* of H (w.r.t. Q), it must be possible to answer Q using only the instance of E after updating of H. Now consider a sequence of updates of the form *"delete the last state of R in H."* These updates, combined with Q, can *reconstruct* the contents of R for an arbitrary state of H. This can only be possible if E is lossless.

This, however, means that any such encoding must occupy roughly the same storage as the original database, making vacuuming useless. Similar results can be shown even for valid time databases in which updates are restricted to insertions.

Approaches to Vacuuming

The ability to vacuum data from a history depends on the expressive power of the query language in which queries over the history are formulated and on the number of the actual queries. For example, allowing an arbitrary number of *ad-hoc* queries precludes any possibility effective vacuuming of data, as finite relational structures can be completely characterized by first-order queries. Thus, for common temporal query languages, this observation leaves us with two essential options:

1. An *administrative* solution is adopted and a given history is vacuumed using a set of *policies* independent of queries. Ad-hoc querying of the history can be allowed in this case. However, queries that try to access already expired values (i.e., for which the view is not faithful history encoding) have to fail in a predefined manner, perhaps by informing the application that the returned answer may be only approximate, or
2. A query driven data expiration technique is used. Such a technique, however, can only work for a fixed set of queries *known in advance.*

Administrative Approaches to Vacuuming

One approach to vacuuming data histories, and, in turn, to defining *expiration operators*, can be based on *vacuuming specifications* that define query/application-independent policies. However, when data are removed from a history in such a way, the system should be able to characterize queries whose answers are not affected. A particular way to provide vacuuming specifications (such as through the ideas of Skyt et al. [6,8]) is using *deletion* (ρ) and *keep* (κ)

expressions. These would be invoked from time to time, perhaps by a vacuuming daemon. The complete specification may contain both deletion and keep specifications. For example:

$$\rho(EmpDep) : \sigma_{TT_{end} \leq NOW-1yr}(EmpDep)$$
$$\kappa(EmpDep) : \sigma_{EmpStatus='Retain'}(EmpDep)$$
$$\rho(EmpDep) : \sigma_{VT_{end} \leq NOW-7yrs}(EmpDep)$$

This specification states that unless the Employee has a status of “Retain,” all corrected data should be vacuumed after 1 year and all superceded data vacuumed after 7 years. For safety, keep specifications always override delete specifications (note that the ordering of the individual deletion and keep expressions is significant).

Vacuuming in Practice. Vacuuming specifications are generally given as either part of the relation definition or as a stand-alone vacuuming specification. In TSQL2, for example, a CREATE TABLE command such as:

```
CREATE TABLE EmpDep (
    Name              CHAR(30) NOT NULL,
    Dept              CHAR(30) NOT NULL,
AS TRANSACTION YEAR(2) TO DAY
VACUUM NOBIND (DATE ‘now - 7 days’);
```

specifies *inter alia* that only queries referencing data valid within the last 7 days are permissible [3] while

```
CREATE TABLE EmpDep ( ... )
  VACUUM DATE ‘12 Sep 2007’;
```

specifies that only query referencing any data entered on or after 12 September 2007 are permissible. The VACUUM clause provides a specification of what temporal range constitutes a valid query with the NOBIND keyword allowing DATE to be the date that the query was executed (as opposed to the date that the table was created).

An alternative is to allow tuple-level expiration of data. In this case, the expiration date of data are specified on insert. For example, in the work of Schmidt et al. [5] users might enter:

```
INSERT INTO EmpDep
VALUES (‘Plato’, ‘Literature’, ...)
EXPIRES TIMESTAMP ‘2007-09-12 23:59:59’;
```

to indicate that the tuple may be vacuumed after the date specified.

Application/Query-Driven Approaches

There are many applications that collect data over time but for which there are no *natural* or *a priori* given vacuuming specifications. However, it is still important to control the size of the past data needed. Hence, it is a natural question whether appropriate specifications can be derived *from the (queries in the) applications* themselves (this requires an *a priori* fixed *finite* set of queries – in the case of ad-hoc querying such a specification cannot exist. Formally, given a query language $\mathcal{L}$, a computable mapping of queries $Q \in \mathcal{L}$ to triples $(\emptyset, \Delta, Q')$, such that $(\emptyset, \Delta, Q')$ is an expiration operator for Q over H, has to be constructed. Figure 1 summarizes the results known for various temporal query languages and provides references to the actual techniques and proofs.

Key Applications

The major application domains for vacuuming are historical databases (that, being append only, need a mechanism to limit their size), logs (particularly those collected for more than one purpose with different statutes and business processes), monitoring applications (with rollback requirements) and garbage collection (in programming languages). Also, as *data streams* are essentially *histories*, the techniques and results

Temporal Query Language	Lower bound	Upper bound	Reference
Past FO Temporal Logic	bounded	$\mathrm{POLY}(\mathbf{D}_H)$	[1]
Past Temporal μ-Calculus	bounded	$\mathrm{POLY}(\mathbf{D}_H)$	[12]
Temporal Relational Calculus	bounded	$\mathrm{ELEM}(\mathbf{D}_H)$	[11]
Future Temporal μ-Calculus	$\Omega(\lvert\mathbf{T}_H\rvert)$	$O(\lvert H\rvert)$	[14]
Propositional Past TL w/duplicates	$\Omega(\lvert\mathbf{T}_H\rvert)$	$O(\lvert H\rvert)$	[14]
Past Temporal μ-Calculus w/bounded duplicates	$\Omega(\lvert\log(\mathbf{T}_H\rvert))$	$\Omega(\lvert\log(\mathbf{T}_H\rvert))$	[14]
Conjunctive Past TL Queries w/duplicates	$\Omega(\lvert\log(\mathbf{T}_H\rvert))$	$\Omega(\lvert\log(\mathbf{T}_H\rvert))$	[14]
Past TL Queries w/counting	$\Omega(\lvert\log(\mathbf{T}_H\rvert))$	$O(\lvert H\rvert)$	[13]

Temporal Vacuuming. Figure 1. Space bounds for residual histories.

developed for vacuuming and data expiration can be applied to query processing over data streams. In particular, expiration operators for a given query yield immediately a *synopsis* for the same query in a streaming setting. This observation also allows the transfer of the space complexity bounds.

Future Directions

Most approaches have concentrated on specifying what data to retain for given queries to continue to be answered perfectly. There are two other possibilities:

- Given a particular vacuuming specification and a query that is not supported fully by this specification, can the degree can this query be answered by the residual history be determined? Some suggestions are given by Skyt and Jensen [7] who propose that queries that may return results affected by vacuuming should also provide suggestions for an alternative, similar query.
- Given a requirement that certain queries should not be answered (e.g., for legal reasons), what would be the vacuuming specifications that would guarantee this, in particular in the conjunction with the issue of approximate answers above?

Both of these are areas for further research. Finally, most vacuuming research assumes a static schema definition (or at least, an overarching applicable schema definition). Having the versioning of schema while also handling the vacuuming of data is also an open problem.

Cross-references

▶ Point-Stamped Temporal Models
▶ Query Rewriting Using Views
▶ Schema Versioning
▶ Self-Maintenance of Views
▶ Synopses for Data Streams
▶ Temporal Query Languages

Recommended Reading

1. Chomicki J. Efficient checking of temporal integrity constraints using bounded history encoding. ACM Trans. Database Syst., 20(2):149–186, 1995.
2. Garcia-Molina H., Labio W., and Yang J. Expiring data in a warehouse. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 500–511.
3. Jensen C. Vacuuming. In The TSQL2 Temporal Query Language, Chapter 23, R. Snodgrass (ed.). Kluwer, New York, 1995, pp. 451–462.
4. Jensen C.S. and Mark L. A framework for vacuuming temporal databases. Tech. Rep. CS-TR-2516, University of Maryland at College Park, 1990.
5. Schmidt A., Jensen C., and Saltenis S. Expiration times for data management. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 36.
6. Skyt J. Specification-Based Techniques for the Reduction of Temporal and Multidimensional Data. Ph.D thesis, Aalborg University, Aalborg, Denmark, 2001.
7. Skyt J. and Jensen C.S. Vacuuming temporal databases. TimeCenter technical report TR-32, Aalborg University, 1998.
8. Skyt J., Jensen C.S., and Mark L. A foundation for vacuuming temporal databases. Data Knowl. Eng., 44(1):1–29, 2003.
9. Skyt J., Jensen C.S., and Pedersen T.B. Specification-based data reduction in dimensional data warehouses. In Proc. 18th Int. Conf. on Data Engineering, 2002, p. 278.
10. Stonebraker M. and Rowe L. The design of POSTGRES. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 340–355.
11. Toman D. Expiration of historical databases. In Proc. 8th Int. Symp. Temporal Representation and Reasoning, 2001, pp. 128–135.
12. Toman D. Logical data expiration for fixpoint extensions of temporal logics. In Proc. 8th Int. Symp. Advances in Spatial and Temporal Databases, 2003, pp. 380–393.
13. Toman D. On incompleteness of multi-dimensional first-order temporal logics. In Proc. 10th Int. Symp. Temporal Representation and Reasoning/4th Int. Conf. Temporal Logic, 2003, pp. 99–106.
14. Toman D. On construction of holistic synopses under the duplicate semantics of streaming queries. In Proc. 14th Int. Symp. Temporal Representation and Reasoning, 2007, pp. 150–162.

Temporal Value

▶ History in Temporal Databases

Temporal Visual Interfaces

▶ Temporal Visual Languages

Temporal Visual Languages

Ulrich Schiel[1], Sonia Fernandes Silva[2]
[1]Federal University of Campina Grande, Campina Grande, Brazil
[2]Etruria Telematica Srl, Siena, Italy

Synonyms

Temporal visual queries; Temporal visual interfaces

Definition

Database technology has evolved in order to be typically oriented towards a large set of non-expert users. While attempting to meet this need, textual query languages, such as SQL, have been replaced by visual query languages, which are based on visual representations of the database and direct manipulation mechanisms. Moreover, data characterized by the temporal dimension play an important role in modern database applications. Temporal Visual Languages are user-oriented languages that meet the specific requirements of querying and visualizing temporal data in an interactive and easy-to-use visual form.

Historical Background

The availability of graphical devices at low cost and the advent of the direct manipulation paradigm [10] have given rise in the last years to a large diffusion of visual user interfaces. Regarding the database area, databases are designed, created, and possibly modified by experts, but there are different kinds of users whose job requires access to databases, specifically for extracting information. However, traditional query languages for databases, such as SQL, are not very approachable for these users, for both their intrinsic syntactical complexity and the lack of a global view of the data of interest together with their interrelationships. Thus, visual interfaces for databases, in particular, the so-called Visual Query Systems (VQS) [4], have arisen as an evolution of the traditional query languages.

VQS include both a language to express queries in a visual form and a query strategy. They are oriented to a wide spectrum of users who generally ignore the inner structure of the accessed database and are characterized by several notable features, such as the availability of interactive visual mechanisms that facilitate the typical process of query formulation and refinement, without requiring users to have a previous knowledge of the database schema and to learn the syntax and semantics of a query language.

Moreover, it has been pointed out that modern database applications deal with temporal data such as banking, medical records, airline reservations, financial data, decision support systems, etc. Several proposals of temporal query languages have been carried out in the past years, where special clauses and predicates are added to the original language in order to deal with the temporal aspects. These languages increase the usability problems of the originating query languages, including quite complex syntax and a steep learning curve. For instance, considering specifically the temporal relational languages, the user must be familiar with concepts such as *tuple* and *attribute time-stamping*, temporal joins, as well as syntax and semantics of temporal predicates. As efforts were made to find new visual query mechanisms for accessing conventional databases, this should also be done for temporal databases.

In order to address this need, some proposals of temporal visual query languages have arisen in the last years. Initial proposals have mainly concentrated on the activity of query formulation, where temporal visual operators are interactively applied over diagrammatic representations of the database schemas exhibiting temporal features. However, most of these visual languages are not provided with formal syntax and semantics [13] and do not address the visual interaction with the query result. Trying to overcome this limitation, a variety of techniques for visualizing time-oriented data have been proposed [12], but they do not support query capabilities for extracting further information. In recent years, research efforts have been made in order to develop temporal visual interfaces in which the end-user could apply visual mechanisms for querying and visualizing temporal data in a single structure.

Foundations

Elements in temporal databases such as objects and their attributes may be temporal, meaning that the history of the successive values of a property is recorded or that the history of an object as a whole may be kept in the database. The valid time of a data element may be a single instant, a time interval or a set of disjoint time intervals.

The process of visual query formulation can be seen as constituted by three phases [13]: the user selects the part of the database he wants to operate on (location phase); then, he defines the relations/restrictions within the selected part in order to produce the query result (manipulation phase); finally, he operates on the query result (visualization phase). The same phases apply to a visual temporal query formulation.

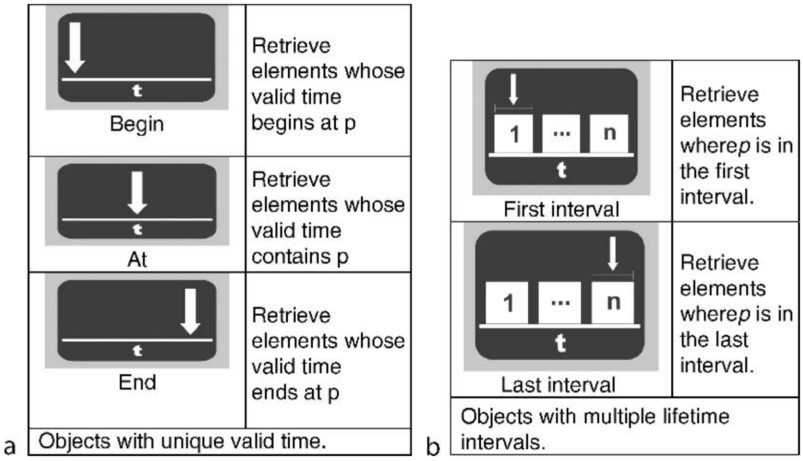
In the *location phase* the goal is the precise definition of the fragment of the database schema involved in the query, known as *query subschema*. Many approaches of temporal visual languages adopt the visualization of the database schemas as Entity-Relationship (ER) diagrams extended with temporal entities (classes) and temporal relationships. Following this approach, the selection of the subschema of interest may be done by selecting the classes, attributes, and relationships of interest [10], including the temporal ones. As an evolution of this ER-based approach a direct manipulation strategy has been proposed in [13], which adopts a "graphical notebook" metaphor for interacting with the database schema. Usability tests showed that this approach was enjoyed by the users, since they prefer to interact with something more familiar.

In the *manipulation phase*, the query subschema can be manipulated in several ways, according to the available query operators on which the detailed conditions of the query must be specified. A database query has two orthogonal components: *selection* and *projection.* In an analogy to a SQL SELECT-FROM-WHERE statement, the SELECT clause gives the data *projection* of the query result, the FROM clause states the *query subschema* of the location phase and the WHERE clause establishes the *selection* conditions on the database in order to retrieve the required data. Temporal queries encompass the possible combinations of current/temporal selection and current/temporal projection over time and data, resulting into nine different combinations beginning from data selection/data projection up to mixed selection/mixed projection [13]. For instance a query '*when did the employee Joseph move from department 1 to department 2, and what salary did he get at the new job?* ' is a data selection (Joseph and the two departments) and mixed projection (date of movement – time; new salary – data).

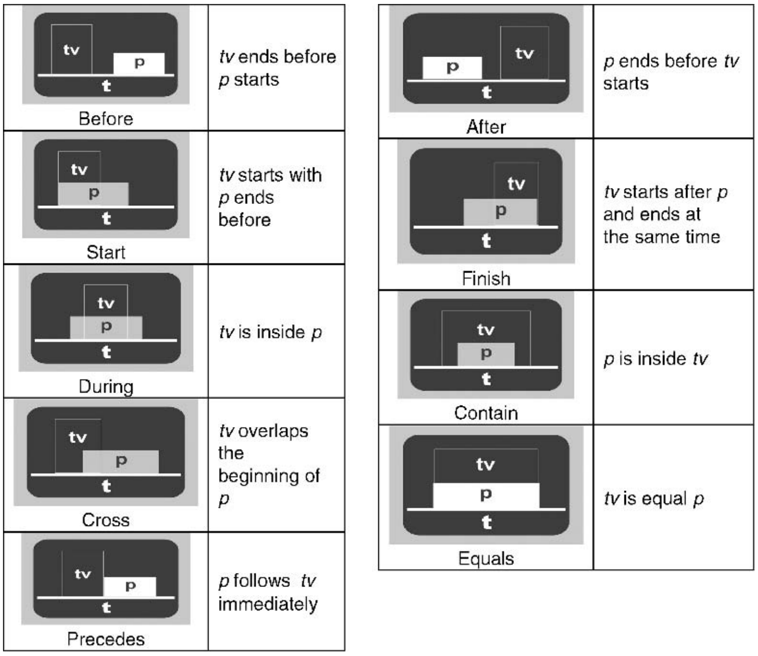
The question in temporal query processing is how to relate the (temporal-) data of the query to (temporal-) data of the database. This relationship is stated by comparing time-spans of the data lifetime that has been specified in the query with the corresponding lifetimes of objects or attributes in the database. This comparison may include special forms for expressing temporal restrictions or relations between the query and the database.

Temporal visual languages can use a set of visual representations for time and temporal relations, depending on the type of request: snapshot or slice [14]. Snapshot queries deal with facts that were valid at a particular time instant, whereas slices queries return facts that were valid over some period of time. Regarding snapshot queries, instants are visually represented as circles or small vertical bars. If *p* is a time instant (point) of the query and *t* is the valid time of an object (or attribute), the temporal relations between them can be visually represented as icons (Fig. 1).

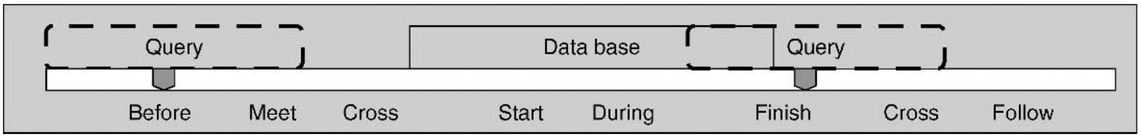
Regarding slice queries, the temporal relations between time intervals are based on Allen's Calculus of temporal intervals [3], which gives a complete set of possible relations between two (temporal) intervals. If both the database time *tv* and the query time *p* are intervals, the relational primitives of Allen can be visually represented as icons (Fig. 2). Instead of using icons for expressing temporal relations, a more intuitive


t
Begin
Retrieve elements whose valid time begins at p
t
At
Retrieve elements whose valid time contains p
t
End
Retrieve elements whose valid time ends at p
a
Objects with unique valid time.
1
...
n
t
First interval
Retrieve elements wherep is in the first interval.
1
...
n
t
Last interval
Retrieve elements wherep is in the last interval.
b
Objects with multiple lifetime intervals.

Temporal Visual Languages. Figure 1. Point/Interval relations as icons [5].



Temporal Visual Languages. Figure 2. Interval icons.



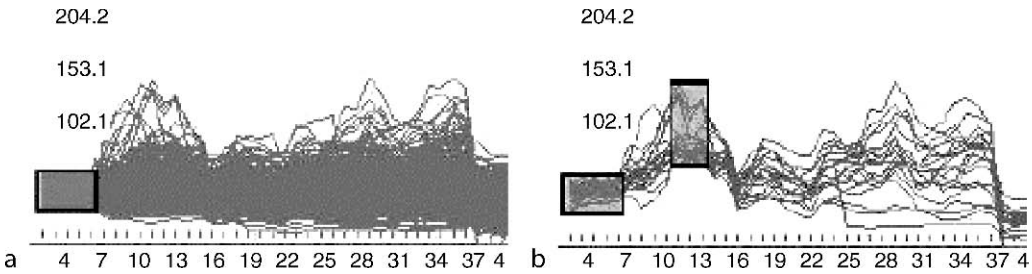
Temporal Visual Languages. Figure 3. Mobile slider [13].

form of expressing such relations is to visually represent intervals as horizontal slide bars in order to dynamically specify the predicates of Allen. For instance, Silva et al. [13] have proposed a mobile slider, as illustrated in Fig. 3.

However, one of the most difficult problems in designing a temporal visual language is to achieve both high expressive power and ease-of-use. For example, the two representations illustrated above cannot deal with situations involving more than two intervals. In this case, the logical expressions between temporal relations (involving conjunctions and disjunctions) should be addressed in this manipulation phase. Some proposals address such a need by adopting visual metaphors for complex representation of temporal relations [6, 8] in a temporal query formulation. Considering that the endpoints may be imprecise or variable, Chittaro and Combi [6] propose three alternative visual representations of flexible temporal intervals in order to deal with variable endpoints. Hibino and Rudensteiner [8] propose a bar with alternative endpoints, e.g., for specifying the temporal relation between two intervals starting at the same time but without restriction on the end, giving rise the logical expression *begins(A,B)* ∨ *equals(A,B)* ∨ *begins(B,A)*.

Moreover, other complex temporal relations such as temporal patterns should be also addressed. For instance, users might be interested in data related to events of arbitrary duration or events separated by arbitrary time gaps. The temporal visual language defined in [7] address such a need by presenting visual metaphors for expressing event-based queries, where constraints on events and inter-event time-spans are visually specified in different ways.

Finally in the *visualization phase*, the historical data retrieved from the query result is visualized for



Temporal Visual Languages. Figure 4. History with "TimeBox" [9].

interactive exploration and analysis. The most widely known visualization technique of time-oriented data are the interactive timeline [12], where the time is regarded as an ordinal axis in a bi-dimensional (2D) visualization and data are located at different positions along this time axis. Timelines and other visualization techniques can be categorized according to the generic criteria that address ontologies about the time, such as the temporal primitives that make up the time axis (time points and intervals), the temporal order (linear, cyclic or branching), etc. For instance, timeline visualization takes advantage of the linear ordered nature of time. An additional criterion is if time-oriented visualization supports the snapshot or slice views [12]. Other relevant criterion is the data tied to time axis [2], such as the data type which indicates if the data are abstract or spatial; the number of involved variables (univariate or multivariate) related to the data (temporal attributes and relationships); and its abstraction level (e.g., raw vs. aggregated data).

It is worth noting that for an effective exploration of data, visual techniques must be integrated with suitable interaction techniques, following the principle of visualization information mantra, defined in [11]: *overview first, zoom and filter, then details-on-demand*, where visual tools exploring the *dynamic query* approach [1] are well-known implementations of this principle. This means that starting from an overview of a large dataset, one may zoom and filter this overview to extract a data subset. Then, more details can be obtained from the selected data subset. In dynamic queries, the manipulation and visualization phases proceed iteratively in a visual query formulation. This means that, after visualization of a preliminary result, the user may interact with this result in order to refine the query.

Within this context, the visualization phase in temporal visual languages focuses on visual query and exploration of temporal trends and patterns within historical results from a preliminary query by using suitable interactive visualization techniques. When exploring such data, suitable interaction techniques such as the direct manipulation and brushing can be integrated with visual query capabilities. For instance, *TimeSearcher* [9] allow users to visually query and explore patterns in time-series data by using visual widgets for data filtering called "*TimeBoxes*." *TimeBoxes* are rectangular query locators that specify the region(s) in which the users are interested. They are placed and directly manipulated on a 2D timeline, with the region boundaries providing the query parameters, as illustrated in Fig. 4. The extent of the *Timebox* on the time (x) axis specifies the time period of interest, while the extent on the value (y) axis specifies a constraint on the range of data values of interest. In this case, a query is dynamically created by drawing a box on the timeline. Multiple timeboxes can be combined to specify conjunctive queries. Only data sets that match all of the constraints implied by the timeboxes are visualized.

Key Applications

Temporal Visual Languages may be integrated with Spatial Visual Languages for Geographic Information Systems. Other typical applications are virtual reality, moving objects, or multimedia systems, such as video and audio data. It can be also an important concern in Visual Analytics.

In the medical field, there are many different applications needing temporality of patient records, images, examination events, and so on.

The classic application fields of Temporal Databases are systems of planning data, or systems of historic data, such as banking account, economic data, meteorological data, business histories, and many others. Also in Decision Support Systems, such as OLAP – Online Analytical Processing, time is the most important dimension.

Another promising application is related to the document management, which is based on time-changing texts, such as legal data or instructional texts. For instance, a judgment support system based on jurisprudence must consider the temporal context of past judgments.

Cross-references

- ▶ Data Visualization
- ▶ Lifespan
- ▶ Temporal Database
- ▶ Temporal Query Languages
- ▶ TSQL2
- ▶ Visual Interaction
- ▶ Visual Interfaces
- ▶ Visual Query Language

Recommended Reading

1. Ahlberg C. and Shneiderman B. Visual information seeking: tight coupling of dynamic query filters with starfield displays. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1994, pp. 313–317.
2. Aigner W. et al. Visualizing time-oriented data – A systematic view. Comput. Graph., 31(3):401–409, 2007.
3. Allen J.F. Maintaining knowledge about temporal interval. Commun. ACM, 26(1):832–843, 1983.
4. Catarci T . et al. Visual query systems: analysis and comparison. J. Vis. Lang. Comput., 8(2):215–260, 1997.
5. Cavalcanti V.M.B., Schiel U., and Baptista C.S. Querying spatio-temporal databases using a visual environment. In Proc. Working Conf. on Advanced Visual Interfaces, 2006, pp. 412–419.
6. Chittaro L. and Combi C. Representation of temporal intervals and relations: information visualization aspects and their evaluation. In Proc. 8th Int. Symp. Temporal Representation and Reasoning, 2001, pp. 13–20.
7. Fails J.A., Karlson A., and Shahamat L. Visual Query of Multi-Dimensional Temporal Data. http://www.cs.umd.edu/class/spring2005/cmsc838s/assignment-projects/visual-query-of-temporal-data/Final-Paper-06.pdf.
8. Hibino S. and Rundensteiner E.A. User interface evaluation of a direct manipulation temporal query language. In Proc. 5th ACM Int. Conf. on Multimedia, 1997, pp. 99–107.
9. Hochheiser H. and Shneiderman B. Dynamic query tools for time series data sets, timebox widgets for interactive exploration. Inf. Vis., 3(1):1–18, 2004.
10. Shneiderman B. Direct manipulation, a step beyond programming languages. IEEE Comput., 16(8):57–69, 1983.
11. Shneiderman B. The eyes have it: a task by data type taxonomy for information visualizations. In Proc. IEEE Symp. on Visual Languages, 1996, pp. 336–343.
12. Silva S.F. and Catarci T. Visualization of linear time-oriented data: a survey. In Proc. 1st Int. Conf. on Web Information Systems Eng., 2000, pp. 310–319.
13. Silva S.F., Catarci T., and Schiel U. Formalizing visual interaction with historical databases. Inf. Syst., 27(7):487–521, 2002.
14. Silva S.F., Schiel U., and Catarci T. Visual query operators for temporal databases. In Proc. 4th Int. Workshop Temporal Representation and Reasoning, 1997, pp. 46–53.

Temporal Visual Queries

▶ Temporal Visual Languages

Temporal XML

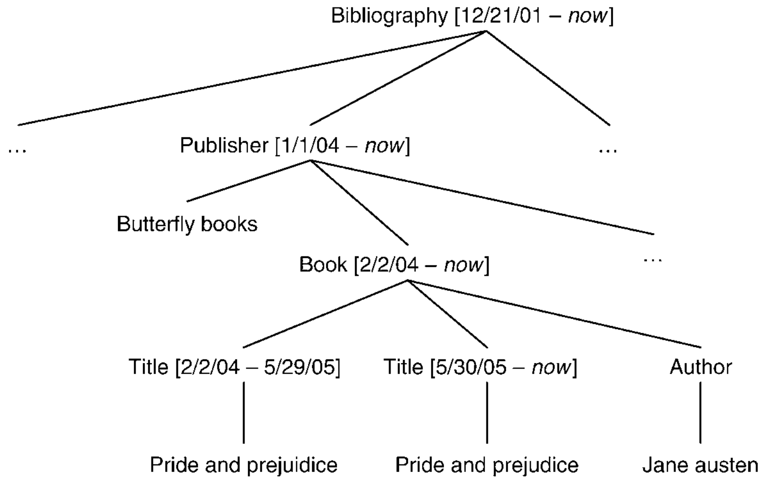
Curtis Dyreson[1], Fabio Grandi[2]
[1]Utah State University, Logan, UT, USA
[2]University of Bologna, Bologna, Italy

Synonyms

Temporal semi-structured data

Definition

Temporal XML is a *timestamped instance* of an XML data model or, more literally, an XML document in which *specially-interpreted* timestamps are present. In general, an XML data model instance is a tree or graph in which each node corresponds to an element, attribute, or value, and each edge represents the lexical nesting of the child in the parent's content. In temporal XML, a timestamp is added to some nodes or edges in the instance. The timestamp represents the lifetime of the node or edge in one or more temporal dimensions, usually valid time or transaction time. As an example, Fig. 1 shows a fragment of a temporal XML data model. The bibliographic data in the figure contains information about publishers, books, and authors. The figure also has timestamps that represent *when* each piece of data was entered into the data collection (i.e., the timestamps represent the *transaction-time* lifetime of each element). The bibliography began on Dec 21. 2001, and remains current (until *now*). Information about the Butterfly Books publisher was entered on Jan 1, 2004, and it started publishing a book by Jane Austen on Feb 2, 2004. The title of that book was originally misspelled, but was corrected on May 29, 2005. Alternatively, temporal XML is literally an XML document or data collection in which specially-interpreted timestamps, formatted



Temporal XML. Figure 1. A temporal XML fragment.

in XML, are included. Such a document yields a temporal XML data model instance when parsed.

Historical Background

XML is becoming an important language for data - representation and exchange, especially in web applications. XML is used to "mark-up" a data collection or document adding meaning and structure. The mark-up consists of elements inserted into the data. Usually an XML document is modeled as a tree in which each interior node corresponds to an element in the document and each leaf to a text value, attribute, or empty element. Temporal XML adds timestamps to the nodes and/or edges in the data model instance. The timestamps represent the lifetime of the nodes (edges).

Grandi has created a good bibliography of research in this area [8]. Chawathe et al. were the first to study time in an XML-like setting [2]. They encoded times in edge labels in a semi-structured database and extended the Lorel query language with temporal constructs. Dyreson et al. extended their research with collapsing and coalescing operators [5]. Grandi and Mandreoli presented techniques for adding explicit valid-time timestamps in an XML document [9]. Amagasa et al. next developed a temporal extension of the XML data model [1]. Following that, a range of temporal XML topics was investigated, from storage issues and indexing [3,11,12,13,14] to querying [6,7,12]. The timestamping of XML documents (or parts thereof) has also been considered in the more general context of versioning of XML documents [11,15]. Finally, schemes for validating and representing times in XML documents have also been considered [4,10].

Foundations

It is important to distinguish between "the representation in XML of a time" and "temporal XML." Times are common in many XML documents, especially documents that record the history of an enterprise. There is nothing special about the representation or modeling of these times. They would be modeled just the same as any other snippet of XML, e.g., represented within a <time> element. Temporal XML, on the other hand, is different. It models both the components within a document or data collection and their lifetimes. An instructive way to think about the difference is that temporal XML weds metadata in the form of timestamps to data contained in a document, i.e., to the elements or parts of the document that are annotated by the timestamps. Research in temporal XML builds on earlier research in temporal (relational) databases. Though many of the concepts and ideas carry over to temporal XML research, the ideas have to be adapted to the tree-like model of XML.

Many temporal XML data models impose a transaction-time constraint on the times along every path in a model instance: the timestamp of a child must be during (inclusive) the timestamp of its parent [1,4]. Said differently, no child may outlive its parent in transaction time. The reason for this constraint is that every snapshot of a temporal data model instance must be a single, complete, valid non-temporal XML data model instance. A non-temporal instance has a single root. But if in a temporal instance a child outlives its parent then, in some snapshot(s), the child represents a second root since it has no parent, thus violating a model property. In valid time it is more common to relax this constraint and model a temporal data collection as a sequence of forests where a child that outlives its parent is interpreted to mean that the child is the root in some snapshot(s) of some tree in the forest [10]. For instance, the valid time of Jane Austen's book Pride and Prejudice would extend from its time of publication (1813) to *now*, far exceeding the lifetime of its publication by Butterfly Books.

Another interesting situation is when a child moves among parents over time (for instance, in the data collection shown in Fig. 1 if the book Pride and Prejudice were published by two different publishers). A directed graph data model is better suited to modeling such movement as a node (e.g., the book) can have multiple incoming edges (e.g., an edge from each publisher) [5]. Various constraints have been proposed for relationships among the timestamps on nodes and edges in the graph.

Timestamps on nodes/edges in a data model instance changes query evaluation. At the core of all XML query languages (and different from SQL or relational query languages) are path expressions that navigate to nodes in a data model instance. In a temporal data model instance, a query has to account for the timestamps along each path that it explores. In general, a node is only available during the intersection of times on every node and edge in the path to it (though a node in a graph data model can be reached along multiple paths). Temporal XML queries can be evaluated using a sequenced semantics [7], that is, simultaneously evaluated in every snapshot or non-sequenced [6,14] where differences between versions can be extracted and paths between versions are directly supported by the data model.

Key Applications

Temporal XML can be used to model an evolving document or data collection. In many situations, "old" documents or document versions are still of use. For instance, in an industrial domain an airplane parts manufacturer has to retain part plan histories to produce parts for older planes, while in the legal domain a tax firm has to keep a complete history of tax laws for audits. Currently, the de facto method for storing old documents is an *archive*. An archive is a warehouse for deleted or modified documents. Archives can be site-specific or built for a number of sites, e.g., the Internet Archive. But the method to retrieve documents from an archive varies widely from site to site, which is problematic because then queries also have to vary. Moreover, archives typically only support retrievals of entire document versions, not a full range of temporal queries or version histories of individual elements. In contrast, temporal XML provides a basis for supporting a full range of temporal queries. Temporal XML can also be explicitly used to represent, store or view historical data, including structured data, or to encode multi-version documents. Multi-version documents are compact representations of XML documents which maintain their identity through modifications and amendments. A temporally consistent individual version or range of consecutive versions (timeslice) can be extracted by means of a temporal query. Temporal XML has also been proposed as a medium of communication with temporal relational databases in the context of traditional enterprise applications.

Future Directions

The future of temporal XML is tied to the continued growth of XML as an important medium for data storage and exchange. Currently, many sites promote XML by publishing data formatted in XML (e.g., genomic and proteomic data can be obtained in three, different XML formats from the National Center for Biotechnology Information (NCBI)). Building a temporal XML data collection by accumulating snapshots gathered from these sites is vital to answering queries such as "What new data has emerged over the past six months?" As search engines become more XML-aware, they could also benefit enormously from making time a relevant component in ranking resources, e.g., a search for "mp3 players" should lower the ranking of discontinued products. The growth of the Semantic Web may lead to XML being supplanted by new languages for knowledge representation such as the Ontology Web Language (OWL). Temporal extensions of these languages will not be far behind. OWL already has one such extension: the Time-determined Ontology Web Language (TOWL).

Cross-references

▶ Temporal Database
▶ Temporal Queries
▶ XML

Recommended Reading

1. Amagasa T., Yoshikawa M., and Uemura S. A data model for temporal XML documents. In Proc. 11th Int. Conf. Database and Expert Syst. Appl., 2000, pp. 334–344.
2. Chawathe S.S., Abiteboul S., and Widom J. Representing and querying changes in semistructured data. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 4–13.
3. Chien S.-Y., Tsotras V.J., and Zaniolo C. Efficient schemes for managing multiversion XML documents. VLDB J., 11(4):332–353, 2002s.
4. Currim F., Currim S., Dyreson C., and Snodgrass R.T. A tale of two schemas: creating a temporal XML schema from a snapshot schema with τ XSchema. In Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 348–365.
5. Dyreson C., Böhlen M.H., and Jensen C.S. Capturing and querying multiple aspects of semistructured data. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 290–301.
6. Dyreson C.E. Observing transaction-time semantics with TTXPath. In Proc. 2nd Int. Conf. on Web Information Systems Eng., 2001, pp. 193–202.
7. Gao D. and Snodgrass R.T. Temporal slicing in the evaluation of XML queries. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 632–643.
8. Grandi F. Introducing an annotated bibliography on temporal and evolution aspects in the World Wide Web. ACM SIGMOD Rec., 33(2):84–86, 2004.
9. Grandi F. and Mandreoli F. The valid web: an XML/XSL infrastructure for temporal management of web documents. In Proc. 1st Int. Conf. Advances in Information Systems, 2000, pp. 294–303.
10. Grandi F., Mandreoli F., and Tiberio P. Temporal modelling and management of normative documents in XML format. Data Knowl. Eng., 54(3):327–254, 2005.
11. Mitakos T., Gergatsoulis M., Stavrakas Y., and Ioannidis E.V. Representing time-dependent information in multidimensional XML. J. Comput. Inf. Technol., 9(3):233–238, 2001.
12. Rizzolo F. and Vaisman A.A. Temporal XML: modeling, indexing and query processing. VLDB J., 2007.
13. Wang F. and Zaniolo C. X-BiT: An XML-based Bitemporal Data Model. In Proc. 13th Int. Conf. on Entity-Relationship Approach, 2004, pp. 810–824.
14. Wang F. and Zaniolo C. An XML-based approach to publishing and querying the history of databases. World Wide Web, 8(3):233–259, 2005.
15. Wong R.K., Lam F., and Orgun M.A. Modelling and manipulating multidimensional data in semistructured databases. World Wide Web, 4(1–2):79–99, 2001.

Temporally Indeterminate Databases

▶ Probabilistic Temporal Databases

Temporally Uncertain Databases

▶ Probabilistic Temporal Databases

Temporally Weak

▶ Snapshot Equivalence
▶ Weak Equivalence

Term Expansion

▶ Query Expansion for Information Retrieval

Term Expansion Models

▶ Query Expansion Models

Term Frequency by Inverse Document Frequency

▶ TF*IDF

Term Frequency Normalization

▶ Document Length Normalization

Term Processing

▶ Lexical Analysis of Textual Data

Term Proximity

Vassilis Plachouras
Yahoo! Research Barcelona, Spain

Synonyms

Lexical affinities; Lexical relations

Definition

Term proximity is a form of term dependence based on the distance of terms in a document. A retrieval system using term proximity assigns a higher score to documents in which the query terms appear close to each other.

Key Points

Term proximity is a feature that partially captures the dependence of terms in documents. Information retrievals models are often based on the assumption that terms occur independently of other terms in a document. This assumption is only an approximation to allow the simple mathematical development of retrieval models. There have been, however, several efforts to introduce dependence of terms [4]. Most of the efforts to use term proximity in the past did not result in substantial improvements. Metzler and Croft [2] argued that this can be attributed to the small size of the test collections used in the past, as well as to the fact that previous models required estimating term dependencies for both the classes of relevant and non-relevant documents.

Metzler and Croft [2] proposed a model based on Markov Random Fields for term dependence using term proximity. They modeled full independence, sequential dependence that is equivalent to phrase search, and full dependence, where the dependence between any pair of query terms is computed. Mishne and de Rijke [3] also proposed a model in which every n-gram of the query is considered as a phrase, and it is evaluated on an index consisting of single terms. Their results show that improvements in early precision are obtained in the setting of Web search. In both models, term proximity is based on lexical relations [1]. Terms are said to be in a lexical relation if they appear often within a certain number of tokens of each other.

Cross-references

► Information Retrieval Models
► N-Gram Models

Recommended Reading

1. Maarek Y.S. and Smadja F.Z. Full text indexing based on lexical relations an application: software libraries. In Proc. 12th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1989, pp. 198–206.
2. Metzler D. and Croft B. A Markov random field model for term dependencies. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 472–479.
3. Mishne G. and de Rijke M. Boosting Web retrieval through query operations. In Proc. 27th European Conf. on IR Research, 2005, pp. 502–516.
4. Yu C.T., Buckley C., Lam K., and Salton G. A generalized term dependence model in information retrieval. Inform. Technol. R&D, 2:129–154, 1983.

Term Statistics for Structured Text Retrieval

Mounia Lalmas
Queen Mary, University of London, London, UK

Synonyms

Within-element term frequency; Inverse element frequency

Definition

Classical ranking algorithms in information retrieval make use of term statistics, the most common (and basic) ones being within-document term frequency, *tf*, and document frequency, *df*. *tf* is the number of occurrences of a term in a document and is used to reflect how well a term captures the topic of a document, whereas *df* is the number of documents in which a term appears and is used to reflect how well a term discriminates between relevant and non-relevant documents. *df* is also commonly referred to as inverse document frequency, *idf*, since it is inversely related to the importance of a term. Both *tf* and *idf* are obtained at indexing time. Ranking algorithms for structured text retrieval, and more precisely XML retrieval, require similar terms statistics, but with respect to elements.

Key Points

To calculate term statistics for elements, one could simply replace documents by elements and calculate so-called within-element term frequency, *etf*, and inverse element frequency, *ief*. This however raises an issue because of the nested nature of XML documents in particular. For instance, suppose that a section element is composed of two paragraph elements. The fact that a term appears in the paragraph necessitates that it also appears in the section. This overlap can be taken into account when calculating the *ief* value of a term.

In structured retrieval, in contrast to "flat" document retrieval, there are no a priori fixed retrieval units. The whole document, a part of it (e.g., one of its section), or a part of a part (e.g., a paragraph in the section), all constitute potential answers to queries. The simplest approach to allow the retrieval of elements at any level of granularity is to index all elements. Each element thus corresponds to a document, and *etf* and *ief* for each element are calculated based on the concatenation of the text of the element and that of its descendants (e.g., [4]).

With respect to the calculation of the inverse element frequency, *ief*, the above approach ignores the issue of nested elements. Indeed, the *ief* value of a term will consider both the element that contains that term and all elements that do so in virtue of being ancestors of that element. Alternatively, *ief* can be estimated across elements of the same type (e.g., [3]) or across documents (e.g., [1]). The former greatly reduces the impact of nested elements on the *ief* value of a term, but does not eliminate it as elements of the same type can be nested within each other. This approach can be extended to consider the actual path of an element, leading to so-called inverted path frequency. For example, in [2], this is defined as the combination of the *ief* values (as above calculated) with respect to each of the element types forming the path. The latter case, i.e., calculating *ief* across documents, is the same as using inverse document frequency, which completely eliminates the effect of nested elements.

Cross-references

▶ XML Retrieval

▶ Indexing Units

▶ Structure Weight

▶ Relationships in Structured Text Retrieval

Recommended Reading

1. Clarke C.L.A. Controlling overlap in content-oriented XML retrieval. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 441–448.
2. Grabs G. and Schek H.-S. ETH Zürich at INEX: flexible information retrieval from XML with PowerDB-XML. In Proc. 1st Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2002, pp. 141–148.
3. Mass Y. and Mandelbrod M. Component ranking and automatic query refinement for XML retrieval. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 73–84.
4. Sigurbjörnsson B., Kamps J., and de Rijke M. An element-based approach to XML retrieval. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2003, pp. 19–26.

Term Weighting

Ibrahim Abu El-Khair
Minia University, Minia, Egypt

Definition

Term weighting is a procedure that takes place during the text indexing process in order to assess the value of each term to the document. Term weighting is the assignment of numerical values to terms that represent their importance in a document in order to improve retrieval effectiveness [8]. Essentially it considers the relative importance of individual words in an information retrieval system, which can improve system effectiveness, since not all the terms in a given document collection are of equal importance. Weighing the terms is the means that enables the retrieval system to determine the importance of a given term in a certain document or a query. It is a crucial component of any information retrieval system, a component that has shown great potential for improving the retrieval effectiveness of an information retrieval system [7].

Historical Background

The use of word frequency dates back to G. K. Zipf and his well known law [14] for word distribution. The law indicates that there is a correlation between the frequency of a word and its rank, and their product is a constant.

$$r^*f = c$$

where: r is the rank of the word
f is the frequency of the word
c is a parameter/constant that depends on the text being analyzed.

Zipf indicates that this law is a way to express the within-document frequency weighting.

The use of word frequency as an indication of its significance in a given document was established almost 10 years later based on observations made by Luhn [4] when he was conducting an explanatory research on creating automatic abstracts in scientific documents. Based on these observations, Luhn proposed that the frequency of word occurrence in a document can be considered a useful measure of the word's significance in that document. The premise is that the author of a certain document repeats certain words as he/she presents his/her argument elaborating the subject of the document.

Looking at the word distribution in any given document shows that there are significant words and insignificant words. Luhn used Zipf's law as his null hypothesis [14] to enable him to specify two cut-off points to exclude all insignificant words, an upper point and a lower point. The upper cut-off eliminated the common words, and the lower point eliminated the rare words; both of which he considered extraneous in the document content. After Luhn excluded words above and below these two cut off points, the most useful range of words remained.

The 1960's saw several key developments in the field of information retrieval in general and the most notable were related to the development of the SMART system by Gerard Salton and his students, first at Harvard University and later at Cornell University. This system utilized weights for index terms based on their frequency [10]. The probabilistic approach to retrieval, with term weights based on probability of relevance, appeared in 1960 and since then it has been tested heavily with many variations [12].

The decade of the 1970's saw a breakthrough in the calculation of term weights used in retrieval systems. By then, it was confirmed that the significance of a certain term in a given document is determined using the term frequency of that term in the document. Sparck Jones [11] argued that the term frequency by itself is not sufficient enough to measure the importance of a term in a collection of documents. She suggested correlating the term frequency with its relative collection frequency, making the collection frequency a variable in retrieval. A significant development in the probabilistic based retrieval was also achieved in 1976 by Robertson and Sparck Jones [6].

Foundations

Storing, organizing, and retrieving information are the main functions of an information retrieval system. With the vast amounts of electronic information now available it is very hard to find the ideal information retrieval system that enables users to get what they want from a specific collection of documents. A considerable amount of research has addressed improvement in the effectiveness of retrieval systems, most of which is focused on finding appropriate indexing techniques. The indexing process in any retrieval system deals with assigning a set of index terms that represents the content of each document within a collection. Choosing the proper index terms is a primary issue in information retrieval systems, as they should be indicative of the content of a given document.

One of the most important procedures in the indexing process is assigning a value, or weight, to an index term in a document. It is a crucial component of any retrieval system, and one that has shown great potential for improving retrieval effectiveness [7]. By assigning a numerical value to a term representing its importance in the document, retrieval effectiveness can be improved [8]. Term weighting indicates how important each individual word is to the document and within the document collection.

The process of assigning term weights is affected by three major factors: term frequency, inverse document frequency, and document length.

Term Frequency

Following Luhn's observations [4], it is known that the significance of a certain term in a given document can be represented by the term frequency of that term in the document. Simply, if there is a document in which the word "database" occurs a hundred times, that document would potentially be more useful in response to a query containing "database" as a query term than a document in which the word appears only one or two times. Of course, the use of this factor alone in calculating the term weights in a collection of documents does not guarantee adequate retrieval performance. For example, there are very common words, sometimes referred to as stop words, which appear in the text but carry little meaning, serving only a syntactic function but not indicating subject matter [3]. They have a very high frequency and tend to diminish the impact of frequency differences among less common words, affecting the weighting process [2].

Inverse Document Frequency

The term frequency indicates the importance of the term in a given document, but knowing the term importance in a collection of documents is also significant. Term frequency was criticized as a method of determining term significance because in its simplest form, it treats all terms equally based on raw count, which does not take into account the term's discriminating power. To resolve this problem Sparck Jones [11] suggested the use of the relative collection frequency or inverse document frequency (IDF), making the frequency of the term in the collection as a whole a variable in retrieval. IDF places greater emphasis on the value of a term as a means of distinguishing one document from another than on its value as an indication of the content of the document itself.

Document Length

With the presence of long documents in the document collection handled by any retrieval system, it became harder to determine the importance of a term based only on the term frequency or the inverse document frequency or both. Even though the combination of them is a good weighting function, it overlooks the document length factor. Longer documents will have higher term frequencies because the terms tend to be repeated several times in the document, and thus will be high in the ranking during the retrieval process. Long documents are also likely to contain more unique terms which may affect the retrieval as well. More terms in a given document increases the possibility of matching between this document and multiple queries [7]. Applying a good normalization technique reduces the effect of long documents and makes the weighting function more effective. Another element to be taken into consideration with the document length factor is the removal of stop words, which changes the document length and subsequently affects the weighting process [2].

Term Weighting Schemes

The following is a brief and basic explanation of some of the major term weighting schemes available. It should be noted that each scheme has many variations and modifications that are not discussed.

TF*IDF A weighting function that depends on the term frequency (TF) in a given document calculated with its relative collection frequency or inverse document frequency (IDF). The term frequency emphasizes term significance in a given document, and inverse document frequency emphasizes term significance in the collection as a whole (TF*IDF).

BM25 A weighting function based on the traditional *Probabilistic Retrieval Model*. The basic principle is that a specific document could be judged relevant to a specific query, based on the assumption that the terms are distributed differently and independently in relevant and non relevant documents. The weight of a given term is calculated on the basis of the presence or absence of query terms in each document in the collection. Terms that have appeared in previously retrieved relevant documents for a given query should be given a higher weight than if they had not appeared in those relevant documents [12].

Language Modeling Language modeling (LM) is an extension of the probabilistic retrieval approach. It is a probabilistic mechanism for generating text, first applied by Andrei Markov at the beginning of the twentieth century to model letter sequences in works of Russian literature. It was also used by Claude Shannon in his models of letter sequences and word sequences, which he used to illustrate the implications of coding and information theory. At the end of the 1970's, LM was used successfully in speech recognition, which was its main application for many years [1]. In 1998 Ponte and Croft [5] were the first to apply language modeling to information retrieval. Their approach was to infer a language model for each document and estimate the probability of generating the query according to each of these models, and then rank the documents according to these probabilities. The results indicated an improvement in retrieval over the traditional TF*IDF, and there was further improvement when they used a smoothing function with their new approach.

Key Applications

Term weighting is a key process in any information retrieval system. It is the means that enables the system to determine the importance of any term in a certain document or a query.

Experimental Results

Experimentation in information retrieval has been an active area for over 40 years, and much of this research has focused on term weighting. Different schemes and variations with different retrieval models have been tested in order to find weighting schemes that perform effectively. In general, the schemes above and their variations have been tested extensively and evaluated (see the corresponding references). Many other weighting schemes were developed and used but without becoming widely adopted, either because the results were not effective enough or because of the complexity of the calculations or both. The Term Discrimination Value (TDV) model of indexing [9] is an example which is now seldom used because of its complexity and weak results.

Until the 1990s, experiments in the field of information retrieval in general and term weighting in particular were conducted on relatively small collections. With the beginning of TREC (http://trec.nist.gov) (Text REtrieval Conference) in 1992, large test collections became available for use by the IR community, making the results of experiments more credible and generalizable. Research in term weighting benefited, establishing the effectiveness of term weighting schemes such as BM25.

Cross-references

▶ BM25
▶ Information Retrieval
▶ Language Models
▶ Lexical Analysis of Textual Data
▶ TF*IDF
▶ Text Indexing Techniques

Recommended Reading

1. Hiemstra D. and de Vries A. Relating the New Language Models of Information Retrieval to the Traditional Retrieval Models (No. TR-CTIT-00-09). Centre for Telematics and Information Technology (CTIT), University of Twente, Amsterdam, Netherlands, 2000.
2. Korfhage R.R. Information Storage and Retrieval. John Wiley, New York, 1997.
3. Lancaster F.W. Indexing and Abstracting in Theory and Practice (2nd edn.). University of Illinois, Graduate School of Library and Information Science, Champaign, IL, 1998.
4. Luhn H.P. The automatic creation of literature abstracts. IBM J. Res. Dev., 2(2):159–165, 1958.
5. Ponte J.M. and Croft W.B. A language modeling approach to information retrieval. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 275–281.
6. Robertson S.E. and Sparck-Jones K. Relevance weighting of search terms. J. Am. Soc. Inf. Sci., 27(3):129–146, 1976.
7. Salton G. and Buckley C. Term-weighting approaches in automatic text retrieval. Inf. Process. Manage., 24(4):513–523, 1988.
8. Salton G. and McGill M. Introduction to Modern Information Retrieval. McGraw-Hill Book Company, New York, NY, 1983.
9. Salton G., Yang, C.S., and Yu, C.T. A theory of term importance in automatic text analysis. J. Am. Soc. Inf. Sci. Technol., 26(1):33–44, 1975.
10. Singhal A. Modern information retrieval: a brief overview. Bull. IEEE Comput. Soc. Tech. Comm. Data Eng., 24(4):35–43, 2001.
11. Sparck Jones K. A statistical interpretation of term specificity and its application in retrieval. J. Doc., 28:11–20, 1972.
12. Sparck Jones K., Walker S., and Robertson S.E. A probabilistic model of information retrieval: development and comparative experiments: Part I. Inf. Process. Manage., 36:779–808, 2000.
13. van Rijsbergen C.J. Information Retrieval (2nd edn.). Butterworths, London, 1979.
14. Zipf G.K. Human Behavior and Principle of Least Effort. Addison Wesley, Cambridge, MA, 1949.

Term-Document Matching Function

▶ Information Retrieval Models

Terminologic Languages

▶ Description Logics

Terminological Database

▶ Electronic Dictionary

Test Collection

Ben Carterette
University of Massachusetts Amherst, Amherst, USA

Synonyms

Corpus

Definition

A test collection is a standard set of data used to measure search engine performance. It comprises a set of queries, ideally randomly sampled from some space, a set of documents to be searched, and a set of judgments indicating the relevance of each document to each query in the set.

Key Points

The use of test collections for performance evaluation began with Cleverdon and Mills [1] and is today known as the Cranfield methodology. Test collections today are much larger than Cleverdon's Cranfield collection, consisting of millions of documents and tens of thousands of relevance judgments. The advantage of having standardized test collections is that experimental results can be compared across research groups and over time.

The National Institute of Standards and Technology (NIST), through their annual Text REtrieval Conferences (TREC), has led the way in providing test collections for information retrieval research. NIST has assembled large-scale test collections for many different retrieval tasks and types of documents and made these available to researchers so that the state of the art in retrieval can be constantly improved.

Cross-references

▶ Document Databases

Recommended Reading

1. Voorhees E.M. and Harman D.K. (eds.). TREC: Experiment and Evaluation in Information Retrieval. MIT, Cambridge, MA, USA, 2005.

Text Analytics

▶ Web Information Extraction

Text Categorization

Dou Shen
Microsoft Corporation, Redmond, WA, USA

Synonyms

Text classification

Definition

Text classification is to automatically assign textual documents (such as documents in plain text and Web pages) into some predefined categories based their content. Formally speaking, text classification works on an instance space X where each instance is a document d and a fixed set of classes $C = \{C_1,C_2,...,C_{|C|}\}$ where $|C|$ is the number of classes. Given a training set D_l of training documents $\langle d,C_i\rangle$ where $\langle d,C_i\rangle \in X \times C$, using a learning method or learning algorithm, the goal of document classification is to learn a classifier or classification function γ that maps instances to classes: $\gamma : X \rightarrow C$ [7].

Historical Background

Text classification, which is to classify documents into some predefined categories, provides an effective way to organize documents. Text classification dates back to the early 1960s, but only in the early 1990s did it become a major subfield of the information systems discipline. Recently, with the explosive growth of online textual data, text classification attracts more and more attention. A knowledge engineering approach is one of the most popular solutions before the late 1980s, which relies on manually defined rules encoding expert knowledge on how to classify documents under the given categories. After that, machine learning based methods became pervasive. These methods automatically build some automatic text classifiers by learning from a set of manually classified documents. The following sections will focus on machine learning based methods.

Foundations

Text categorization consists of several important components including document representation, dimensionality reduction, classification algorithms and performance evaluation, which are to be introduced in the followings sections. Readers are referred to [11] for more details.

Document Representation

Documents or Web pages cannot be directly interpreted by a classifier. Therefore, a proper representation approach is necessary to represent documents. Generally, a document d is usually represented as a vector of term weights $d = \langle w_1,w_2,...,w_{|V|}\rangle$, where V is the set of terms (sometimes called features) that occur at least once in the training document set D_l. This way is known as Vector Space Model (VSM) [7]. Different representation approaches vary in two issues: (i) different ways of understanding what a term is; (ii) different ways of computing term weights. For issue (i), a straightforward way is to identify terms with words. This is often called either the set-of-words or the bag-of-words approach to document representation, depending on whether weights are binary or not [11]. Although some previous work has found that representations more sophisticated than this are not significantly more effective [1], researchers still struggle to find better ways in the following four directions: (i) Represent a document by phrases [6]; (ii) Use the senses of words to represent a document [4]; (ii) Augment document representation by hidden concepts in a document; (iv) Employ language models, such as n-gram models [9], multigram models. For issue (ii), the weight can be binary (1 denoting presence and 0 absence of the term in the document) or nonbinary. For the nonbinary value of a term *t*, it can be either the Term Frequency (TF) of the term in a document or TFDIF as computed according to the following equation where $N(t,d)$ is the number of times the word *t* appears in *d*, $|D|$ is the size of the corpus, $n_{t,D}$ is the number of documents in D containing the word *t*:

$$w_t = N(t,d)*log(|D|/n_{t,D})$$

Sometimes, the document vectors are normalized by cosine normalization [11]. Using either binary or nonbinary values, either normalized or the original values depends on the classifier learning algorithm.

Generally, the bag-of-words representation is built based on the text in each document alone. However, it is not uncommon that some documents have certain relationships among them so that the text in one document can help enrich the text of its related documents to improve the classification results. The relationships among documents are quite obvious in the context of Web-page classification. For example, in [2], Glover et al. enrich Web pages by considering inbound links and words surrounding them. They come to the conclusion that the full-text of a Web page is not good enough for representing the Web pages for classification. They create virtual documents by incorporating anchor text and extended anchor text. The experimental results demonstrate that the virtual documents, especially when constructed through extended anchor text are of great help. In [13], the authors enhance the notion of virtual documents by complementing hyperlinks with implicit links which are extracted from query logs. Besides utilizing the links among documents, there are also works enriching documents by inserting features extracted from an existing knowledge base.

Dimensionality Reduction

In document classification, it is unavoidable to face the problem of high dimensionality. The original feature space consisting of the unique terms that occur in a moderate-sized document collection can reach hundreds of thousands of terms. Such high dimensionality prohibits the application of many classification algorithms such as neural networks and Bayes belief models [15]. Furthermore, some features in the native space do not contribute much to the document classification. Therefore, it is beneficial to conduct dimensionality reduction (DR). DR techniques consist of two groups, term selection and term generation. Term selection methods select a subset of terms from the native space while term generation methods obtain new features by combining or transforming the original ones. The methods belonging to the former group include Document Frequency (DF), Mutual Information (MI), Chi-Square and so on. The latter group of methods include Term Clustering, and Latent Semantic Indexing (LSI). A detailed analysis and comparison of these methods is presented in [11,15].

Related to feature selection, some works have tried to remove noise from documents. It is easy to imagine that some text in a document, especially in a Web page, is not related to the main topic of the documents. When judging the category of a document, only the main topic of the document should be considered and the irrelevant text should be removed. These works are different from conventional feature selection in that they process each document independently and the resultant feature space can still have high dimensionality. For example, in [5], Kolcz et al. use summarization as a feature selection method and apply a simple extraction-based technique with several heuristic rules. Different from Kolcz et al's work on pure-text document classification, [12] proposes to improve the Web-page classification performance by removing the noise through some summarization techniques.

Classification Algorithms

During the past few decades, a large number of categorization algorithms have been proposed for document classification such as naïve bayes [8], k-nearest neighbor, decision trees, regression models, neural networks, support vector machines [3], boosting and rule learning algorithms. The authors of [14] made a thorough comparison among these classifiers. In this section, two widely used text classification algorithms, Naive Bayes (NB) and Support Vector Machine (SVM) are briefly introduced.

Naïve Bayesian Classifier (NB) The Naïve Bayesian Classifier (NB) is a simple but effective text classification algorithm which has been shown to perform very well in practice [8]. The basic idea of NB is to use the joint probabilities of words and categories to estimate the probabilities of categories given a document. As described in [8], most researchers employ NB method by applying Bayes' rule:

$$P(C_j|d_i;\hat{\theta}) = \frac{P(C_j|\hat{\theta})\prod_{k=1}^{|V|}P(w_k|C_j;\hat{\theta})^{N(w_k,d_i)}}{\sum_{r=1}^{|C|}P(C_r|\hat{\theta})\prod_{k=1}^{|V|}P(w_k|C_r;\hat{\theta})^{N(w_k,d_i)}}$$

where $P(C_j|\hat{\theta})$ can be calculated by counting the frequency with each category C_j occurring in the training data; $|C|$ is the number of categories; $p(w_i|C_j)$ stands for probability that word w_i occurs in class C_j which may be small in training data, so the Laplace smoothing is chosen to estimate it; $N(w_k,d_i)$ is the number of occurrences of a word w_k in d_i; $|V|$ is the number of words in the training data.

Support Vector Machine (SVM) SVM is well founded in terms of computational learning theory and has been successfully applied to text categorization [3]. SVM operates by finding a hyper-surface in the space of possible inputs. The hyper-surface attempts to split the positive examples from the negative examples by maximizing the distance between the nearest of the positive and negative examples to the hyper-surface. Intuitively, this makes the classification correct for testing data that is near but not identical to the training data. There are various ways to train SVMs. The SVM^{light} system provided by Joachims [3] is one of the widely adopted and efficient implementations.

Performance Measures

Precision, recall and F1-measure are the most popular measures to evaluate the performance of document classification [10]. Precision (P) is the proportion of actual positive class members returned by the system among all predicted positive class members returned by the system. Recall (R) is the proportion of predicted positive members among all actual positive class members in the data. F1 is the harmonic average of precision and recall as shown below:

$$F1 = 2 \times P \times R/(P + R)$$

To evaluate the average performance across multiple categories, there are two conventional methods: micro-average and macro-average. Micro-average gives equal weight to every document; while macro-average gives equal weight to every category, regardless of its frequency [14].

Key Applications

Text classification has many applications [9], including email classification, text genre classification, topic identification, subjective sentiment classification and Web query classification.

Data Sets

There are several open data sets for text categorization. See the following for details:

20 Newsgroups: http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html

Reuters-21578: http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html

RCV1: http://www.daviddlewis.com/resources/testcollections/rcv1

Enron Email Dataset: http://www.cs.cmu.edu/ enron/

Query Classification Dataset: http://www.sigkdd.org/kddcup/index.php?section=2005&method=data

Cross-references

▶ Classification
▶ Information Retrieval (IR)
▶ Text Clustering

Recommended Reading

1. Dumais S., Platt J., Heckerman D., and Sahami M. Inductive learning algorithms and representations for text categorization. In Proc. Int. Conf. on Information and Knowledge Mangement, 1998, pp. 148–155.
2. Glover E.J., Tsioutsiouliklis K., Lawrence S., Pennock D.M., and Flake G.W. Using web structure for classifying and describing web pages. In Proc. 11th Int. Conf. World Wide Web Conference. 2002, pp. 562–569.
3. Joachims T. Text categorization with support vector machines: learning with many relevant features. In Proc. 10th European Conf. on Machine Learning, 1998, pp. 137–142.
4. Kehagias A., Petridis V., Kaburlasos V.G., and Fragkou P. A comparison of word- and sense-based text categorization using several classification algorithms. J. Intell. Inf. Syst., 21(3):227–247, 2003.
5. Kolcz A., Prabakarmurthi V., and Kalita J.K. String match and text extraction: summarization as feature selection for

text categorization, In CIKM'01: Proc. 10th ACM Int. Conf. on Information and Knowledge Management, 2001, pp. 365–370.

6. Lewis D.D. Representation quality in text classification: An introduction and experiment. In Proc. Workshop on Speech and Natural Language, 1990, pp. 288–295.
7. Manning C.D., Raghavan P., and SchÜZe H. Introduction To Information Retrieval. Cambridge University Press, 2007.
8. Mccallum A. and Nigam K. A comparison of event models for naive bayes text classication. In Proc. AAAI-98 Workshop on Learning for Text Categorization, 1998.
9. Peng F., Schuurmans D., and Wang S. Augmenting naive bayes classifiers with statistical language models. Inf. Retr., 7(3–4):317–345, 2004.
10. Rijsbergen C.V. Information Retrieval, 2nd edn. Butterworths, London, 1979.
11. Sebastiani F. Machine learning in automated text categorization ACM Comput. Surv., 34(1):1–47, 2002.
12. Shen D., Chen Z., Yang Q., Zeng H.-J., Zhang B., Lu Y., and Ma W.-Y. Web-page classification through summarization. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 242–249.
13. Shen D., Sun J.-T., Yang Q., and Chen Z. A comparison of implicit and explicit links for web page classification. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 643–650.
14. Yang Y. An evaluation of statistical approaches to text categorization. Inf. Retr., 1(1–2):69–90,1999.
15. Yang Y. and Pedersen J.O. A comparative study on feature selection in text categorization. In Proc. 14th Int. Conf. on Machine Learning, 1997, pp. 412–420.

Text Classification

► Text Categorization

Text Clustering

Hua Li
Microsoft Research Asia, Beijing, China

Definition

Text clustering is to automatically group textual documents (for example, documents in plain text, web pages, emails and etc) into clusters based on their content similarity. The problem of text clustering can be defined as follows. Given a set of n documents noted as *DS* and a pre-defined cluster number *K* (usually set by users), *DS* is clustered into *K* document clusters$DS_1, DS_2,\ldots,DS_k$, $(i.e, \{DS_1, DS_2,\ldots,DS_k\} = DS)$ so that the documents in a same document cluster are similar to one another while documents from different clusters are dissimilar [14].

Historical Background

Text clustering was initially developed to improve the performance of search engines through pre-clustering the entire corpus [2]. Text clustering later has also been investigated as a post-retrieval document browsing technique [1,2,7].

Foundations

Text clustering consists of several important components including document representation, text clustering algorithms and performance measurements. The readers should refer to [6,8,13] for more details.

Document Representation

The original representation of textual documents (like plain texts, web pages, emails and etc) could not be interpreted by text clustering algorithms directly. A proper document representation method is necessary for any text clustering algorithms. Vector Space Model [6] is generally used to represent a document *d* as a vector of term weights $d =< w_1, w_2,\ldots,w_{|V|} >$, where *V* is the set of terms (also named as features sometimes) that occur at least once in the document set *DS*. Different representation approaches vary in two issues: (i) different ways of understanding what a term is; (ii) different ways of computing term weights. For issue (i), a straightforward way is to identify terms with words. This is often called either the set-of-words or the bag-of-words approach to document representation, depending on whether weights are binary or not [11]. Some previous work has found that representations more sophisticated than this are not significantly more effective [5]. For issue (ii), the weight can be binary (1 denoting the presence and 0 absence of the term in the document) or non-binary. For the non-binary value, it can be either the Term Frequency (*TF*) of the term in a document or *TFIDF* as computed according to the following equation where *N(t, d)* is the number of the times the word *t* appears in *d*, $|D|$ is the size of the document corpus, $n_{t,D}$ is the number of documents in *D* containing the term *t*:

$$w_t = N(t, d) * \log(|D|/n_{t,D})$$

Cosine normalization is sometimes used to normalize the document vectors [11]. It would depend on the text clustering algorithms to choose proper term weight strategies.

Text Clustering Algorithms

Two categories can be used to organize all various clustering algorithms (most of the general clustering algorithms could be applied to text clustering tasks) developed in the past a few years: hierarchical and paritional approaches. The hierarchical algorithms generate successive clusters in a nested sequence. The partitional ones produce all clusters at one time.

In the following section, three popular clustering algorithms would be briefly introduced for readers to get primary impressions of basic clustering algorithms. Single-Link clustering [3] is one basic approach among hierarchical clustering algorithms category (http://en.wikipedia.org/wiki/Cluster_analysis). *K*-Means clustering [9] is one of the typical partitional algorithms which minimizes square error to generate clusters. Co-clustering [4] is a graphic theory based partitional clustering approach which is very popular in recent years. For more clustering algorithms, the readers can refer to [6].

Single-Link Clustering In the Single-Link clustering, the distance between two clusters is defined as the minimum of the distances of all linkages drawn from the two clusters, where the linkage is the criterion to determine the distance of pairs of patterns/points between two clusters while patterns/points are associated with them. One shortcoming of the Single-Link clustering is that it would suffer from a chaining effect [10] which has a tendency to produce clusters that are straggly or elongated [6].

The three main steps of Single-Link Clustering algorithm are as follows [6]:

1. With each pattern/point in its own cluster, construct a list of inter-pattern/point distances for all distinct N ordered pairs of patterns/points, and sort this list in ascending order.
2. Step through the sorted list of distances, forming for each distinct dissimilarity value d_k a graph on the patterns where pairs of patterns closer than d_k are connected by a graph edge.
   a If all the patterns are members of a connected graph, stop.
   b Otherwise, repeat 2.
3. The output of the algorithm is a nested hierarchy of graphs which can be cut at a desired dissimilarity level to form a clustering. The clusters would be identified by simply connected components in the corresponding graph.

***K*-Means Clustering** *K*-Means clustering algorithm is one of the simple but very efficient clustering algorithms, which allows it to run through large datasets. The main advantages of *K*-Means are (i) simplicity and efficiency; (ii) does not yield the same result with different run as the resulting clusters depend on the initial random assignments. The main disadvantage is that as it minimizes intra-cluster variance, *K*-means does not ensure the result has a global minimum of variance (http://en.wikipedia.org/wiki/Cluster_analysis).

K-Means algorithm is to cluster n objects (here textual documents) based on attributes (the document representation as vector space model) into K ($K < n$) partitions. It assigns each object to the cluster which has the nearest center. The center is defined as the average of all the objects in the cluster, which starts from a set of random initial centers. It assumes that the object attributes form a vector space and the objective for the algorithm to achieve is to minimize total intra-cluster variance or, the squared error function (http://en.wikipedia.org/wiki/Cluster_analysis):

$$V = \sum_{i=1}^{k} \sum\nolimits_{x_j \in S_i} (x_j - \mu_i)^2$$

where $S_i, i = 1, 2, \ldots, k$ are K clusters and μ_i is the center of cluster S_i.

The main steps of K-Means clustering algorithm are as follows [9]:

1. Setup the cluster number K;
2. Randomly generate K clusters and calculate the cluster centers, or directly generate K random points as cluster centers;
3. Assign each other points to the nearest cluster center;
4. Recalculate the new cluster centers after new points are clustered into the clusters;
5. Repeat 3 and 4 until some convergence criterion is met;

Co-Clustering In Co-Clustering method, the document collection would be modeled as a bipartite graph between document and words. That makes the clustering problem could be posed as a graph partitioning problem. Then Co-Clustering is developed as a spectral algorithm which could simultaneously yield a clustering of documents and words based on this document and word graph. The Co-Clustering algorithm uses the second left and right singular vectors of an appropriately scaled word-document matrix to yield good bipartitionings [4].

Performance Measurements

There are generally two types of measurements used to evaluate the performance of different text clustering algorithms. One is internal quality measure and the other is external quality measure. The authors of [12] had made a thorough introduction of various clustering algorithms measurements. The readers could refer to their work for more details. Here a brief introduction for both internal and external quality measurements would be introduced in the following.

Internal Quality Measure The internal quality measure is used to compare different sets of clusters without referring to external knowledge (like human labeled/known classes/categories). One approach of this kind of internal quality measurement is to calculate the "overall similarity" based on the pair-wise similarity of documents in a cluster [12].

External Quality Measure The external quality measure as naming is to leverage external knowledge as known classes (categories) to make comparisons with the generated clusters from the clustering algorithms. Entropy [12] is one external measure which provides a measure of "goodness" for un-nested clusters or for the clusters at one level of a hierarchical clustering. F-measure is another good example of external quality measure, which is more oriented toward measuring the effectiveness of a hierarchical clustering.

The readers should be aware that there are still many other different quality measures than those ones introduced here. The more important thing is that the performance of different clustering algorithms could vary substantially depending on which measure is applied [12].

Key Applications

Text clustering has many applications, including search results clustering, topic detection and tracking, email clustering, and etc.

Cross-references

- ► Document Clustering
- ► Information Retrieval
- ► Text Classification

Recommended Reading

1. Croft W.B. Organizing and Searching Large Files of Documents. Ph.D. Thesis, University of Cambridge, 1978.
2. Cutting D.R., Karger D.R., Pedersen J.O., and Tukey J.W. Scatter/gather: a cluster-based approach to browsing large document collections. In Proc. 15th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1992, pp. 318–329.
3. Day W.H. and Edelsbrunner H. Efficient algorithms for agglomerative hierarchical clustering methods. J. Classification, 1:1–24, 1984.
4. Dhillon I.S. Co-clustering documents and words using bipartite spectral graph partitioning, UT CS Technical Report #TR. Department of Computer Sciences, University of Texas, Austin, TX, 2001.
5. Dumais S., Platt J., Heckerman D. and Sahami M. Inductive learning algorithms and representations for text categorization. In Proc. 7th Int. Conf. on Information and Knowledge Management, 1998, pp. 148–155.
6. Jain A.K., Murty M.N., and Flynn P.J. Data clustering: a review. ACM Comput. Surv., 31(3):264–323, 1999.
7. Leouski A.V., and Croft W.B. An evaluation of techniques for clustering search results. Technical Report IR-76. Department of Computer Science, University of Massachusetts, Amherst, 1996.
8. Lewis D.D. Representation quality in text classification: an introduction and experiment. In Proc. Workshop on Speech and Natural Language, 1990, pp. 288–295.
9. MacQueen J.B. Some methods for classification and analysis of multivariate observations. In Proc. 5th Berkeley Symp. on Mathematical Statistics and Probability, 1967, pp. 281–297.
10. NAGY G. State of the art in pattern recognition. Proc. IEEE., 56:836–862, 1968.
11. Sebastiani F. Machine learning in automated text categorization. ACM Comput Surv., 34(1):147, 2002.
12. Steinbach M., Karypis G., and Kumar V. A comparison of document clustering techniques. Technique Report, University of Minnesota – Computer Science and Engineering, 2000.
13. van Rijsbergen C.J. Information Retrieval, 2nd edn. Butterworths, London, 1979.
14. Yoo I. and Hu X.H. A comprehensive comparison study of document clustering for a biomedical distal library Medline. In Proc. ACM/IEEE Joint Conf. on Digital Libraries, 2006, pp. 220–229.

Text Compression

Paolo Ferragina, Igor Nitto
University of Pisa, Pisa, Italy

Synonyms

Lossless data compression

Definition

Text Compression involves changing the representation of a file so that the (binary) compressed output takes less space to store, or less time to transmit, but still the original file can be reconstructed *exactly* from its compressed representation.

Key Points

The benefit of compressing texts in computer applications is threefold: it reduces the amount of memory to store a text, it reduces the time for transmitting the text over a computer network and, recently, it has been deployed to speed up algorithmic computations because they can better exploit the memory hierarchy available in modern PCs by reducing the disk access time, by increasing virtually the bandwidth and size of disk (or memory, cache), and by coming at a negligible cost because of the significant speed of current CPUs.

A text in uncompressed format, also called *raw* or *plain* text, is a sequence of symbols drawn from an alphabet Σ and represented in $\lceil \log_2 |\Sigma| \rceil$ bits each. Text compressors aim at storing a text in space less than its raw encoding by exploiting the redundancies possibly contained in it. Such redundancies might occur in the form of repetitions or frequently occurring patterns, which are not unlikely in texts generated by humans. Text compression is a *lossless* process because it allows restoring the original text from its compressed form by means of a proper *decompression* algorithm. Most of current approaches in text compression [4] can be classified into: *symbolwise*, *dictionary-based* and *transform-based*.

Symbolwise compressors encode the text one-symbol at time, by emitting a (variable length) codeword per individual symbol. They are further divided into two sub-families: *statistical* and *symbol-ranking*. Statistical compressors include some of the best compressors currently known, like PPM and DMC. They rely on two basic tools for the encoding task: a statistical model of the text and a statistical encoder. The statistical model serves to predict the probability of the next symbol given the previous portion of the text. If the prediction is conditioned on the last k occurred symbols, for some fixed constant k (typically ranging from 5 to 10), the model is said of *order k*. The model can be *static*, *semi-static*, or *dynamic*, according to the way the symbol probabilities are estimated from the scanning of the input text. The static construction assumes a fixed probability distribution for every input text. This is the simplest form of model construction but, since the model is independent of the input, it may poorly predict its symbols. The semi-static construction avoids this limitation by building the model via a preliminary scan of the input text. Unfortunately, a semi-static model must be passed to the decompressor as a part of the compressed file, thus increasing its size. As a result, statistical models are typically dynamic in that they are constructed *adaptively* as the input text is processed. This shares with the semi-static approach the advantage of being tailored on the input, but it additionally avoids the need of passing the model to the decompressor. The probability distribution provided by the model, whichever construction process is adopted, is eventually passed to the statistical encoder that determines the bit codeword associated to the next symbol. The principle used by all statistical encoding methods is to assign shorter codewords (even fraction of bits) to most probable symbols in order to minimize the average length of a codeword (this is typically called the *golden rule*). This is the essence of Huffman and Arithmetic encodings, also known as entropy encoders for their dependence on the entropy of symbols frequencies, a theoretical lower-bound on the number of bits emitted by any statistical encoder. The concatenation of all generated codewords gives the final compressed file of a statistical compressor.

Symbol-ranking compressors are still based on statistical prediction but, rather than estimating the probability distribution of the next symbol, they maintain the alphabet in a dynamic list, where symbols are sorted by decreasing likelihood of occurrence. Each symbol of the text is encoded with its rank in the list. If the prediction process is accurate enough, the distribution of ranks will be skewed around small values, and the resulting sequence will be well compressible through a statistical encoder. Symbol-ranking techniques have been rarely used as “stand-alone” compression methods but, like the well-known *Move-To-Front* (MTF) and *Inversion Frequency* encodings, they are often employed as a fundamental stage of more sophisticated compressors like the *Wheeler-Burrows* Transform (BWT), below.

While symbolwise compressors can encode only one symbol at a time, dictionary-based compressors can represent a *group* of symbols with one unique codeword. These compressors maintain a dictionary of strings, called *phrases*, each one identified by a distinct codeword. The input text is parsed into dictionary phrases which are then replaced by their corresponding (shorter) codewords. As for statistical models, dictionary construction schemes can also be classified into static, semi-static and dynamic; but the most significant difference is that, in the semi-static case, deciding which phrases to include in the dictionary is computationally difficult: in fact, computing the dictionary that maximizes compression is an NP-hard problem. An example of static dictionary is the Run-Length-Encoding method (`RLE`), in which the phrases are all possible runs of equal symbols, typically adopted in FAX transmissions and as a fundamental stage of the `BWT`-based compressors. An example of semi-static dictionary is the Huffword compressor, in which the dictionary is formed by all tokens extracted from an input text and phrases are Huffman-encoded according to their frequency of occurrence. Examples of dynamic-dictionary methods are the well-known `LZ77` and `LZ78` compressors which are implemented in many commercial softwares like `winzip`, `pkzip`, `ARJ`, the `.GIF` image format, etc..

The last class of compression algorithms considered is the one that *transforms* the input text in order to make it easier to compress by simpler coding schemes. The transformation must be reversible, lossless (e.g., a permutation of the input symbols), and efficiently computable and invertible. One notable example is the *Burrows-Wheeler* transform (BWT), which can be built and inverted in time linear in the length of the input text. The output of the `BWT` is a string in which symbols following the same context are grouped together, giving raise to clusters of nearly identical symbols. This feature makes redundancy in the input more accessible to simple coding schemes. The famous compression utility `bzip2`, currently available on most Linux distributions, is indeed based on the `BWT` and uses a proper combination of `MTF`, `RLE` and a statistical encoder to significantly squeeze the `BWT`-output. Besides its usage in pure text compression, the `BWT` has three other remarkable properties: it can be used to design a *compression booster* [1], that is, a tool for improving the performance of other compressors in a well-defined and measurable way; it can be used to derive novel and powerful transform-based compressors for various other data types [2,3], like `XML`, dictionaries and graphs (just to cite a few); and it is at the core of modern Compressed Full-text Indexes [3], that is, compressed representations of the input string which are efficiently searchable via arbitrary patterns.

Cross-references

- ▶ Data Compression in Sensor Networks
- ▶ Indexing Compressed Text
- ▶ XML Compression

Recommended Reading

1. Ferragina P., Giancarlo R., Manzini G., Sciortino M. Boosting textual compression in optimal linear time. J. ACM, 52(4): 688–713, 2005.
2. Ferragina P., Luccio F., Manzini G., Muthukrishnan S. Compressing and searching XML data via two zips. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 751–760.
3. Navarro G., Mäkinen V. Compressed full-text indexes. ACM Comput. Surv., 39(1), Article no. 2, 2007.
4. Salomon D., Data Compression: The Complete Reference, 4th edn., Springer, London 2007.

Text Data Mining

- ▶ Data, Text, and Web Mining in Healthcare
- ▶ Text Mining

Text Databases

- ▶ Document Databases

Text Extraction

▶ Biomedical Scientific Textual Data Types and Processing

Text Generation

Li Zhang[1], Jian-Tao Sun[2]

[1]Peking University, Beijing, China

[2]Microsoft Research Asia, Beijing, China

Synonyms

Natural language generation (NLG)

Definition

Text generation is a subfield of natural language processing. It leverages knowledge in computational linguistics and artificial intelligence to automatically generate natural language texts, which can satisfy certain communicative requirements.

Historical Background

Research work in the text generation field first appeared in the 1970s. Goldman's work on natural language generation from a deep conceptual base appeared in [2]. In the 1980s, more significant work was contributed in this field: McDonald saw text generation as a decision making problem [6], Appelt on language planning (1981), McKeown [8]. In the 1990s, a generic architecture for text generation was discussed, Reiter [10], Hovy [3]. Still today, variations on the generic architecture is a still a widely discussed question, Mellish et al. [9].

Foundations

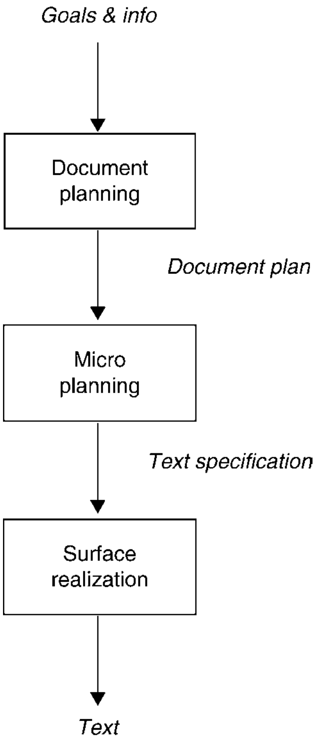
Text Generation, or Natural language generation (NLG), is usually compared with another subfield of natural language processing – natural language understanding (NLU), which is generally considered as the inverse process of the former. Because in a highly abstract level, NLG task synthesizes machine representation of information into natural language texts, while NLU task parses and maps natural language texts into machine representations. However, upon inspection at a more concrete level, they can hardly be seen as "opposite," because they are very different in problem sets, and by internal representations.

Text Generation System Architecture

Input and Output The input of text generation system is information represented in non-linguistic format, such as numerical, symbolical, graphical, etc. The output is understandable natural language in text format, such as messages, documents, reports, etc.

Architectures

The Generic Architecture Despite difference in application backgrounds and realization details, many of the current text generation systems followed a general architecture, which is known as the *Pipelined Architecture* or *Consensus Architecture*, usually described as in Fig. 1([11]; Edward Hovy also had a similar representation for this architecture).



Text Generation. Figure 1. A classical architecture for text generation.

As seen in the Fig. 1, the "Pipelined Architecture" describes a general strategy of tackling text generation problem from macro to micro, from inner structure organization to outer surface realization. Thus, language components such as paragraphs, sentences, and words will be coherently arranged together to meet certain communicative requirements.

The following are the detailed descriptions of the above stages:

Stage 1: Document Planning
Also known as Text Planning, Discourse Planning or Macro Planning). This includes:

- Content determination: Also know as content selection and organization, which is to discover and determine the major topics the text should cover, given a set of communicative goals and representations of information or knowledge.
- Document structuring: Determining the overall structure of the text/document. This structure categorizes and organizes sentence-leveled language components into clusters. The relationship

between different components inside a cluster can be explanatory, descriptive, comparative, causal, sequential, etc.

Stage 2: Micro Planning
Also know as Sentence Planning. This is to convert a document plan into a sequence of sentence or phrase specifications, including:

- Aggregation: To combine several linguistic structures (e.g., sentences, paragraphs) into a single and coherent structure. An example: Tomorrow will be cold. Tomorrow will be windy.
  →Tomorrow will be cold and windy.
- Lexicalization: To choose appropriate words from possible lexicalizations based on the communicative background. Examples: (i) buy, purchase, take, etc, (ii) a lot of, large amounts of, etc.
- Referring expression generation: To choose or introduce different means of reference for sentences, such as pronouns (*pronominalization*). There is usually more than one way to identify a specific object, for example: "Shakespeare, " "the poet and playwright, " "the Englishman, "and "he/him" can all point to the same object. Example: Andrew wanted to sing at the birthday party.
  →He wanted to sing at the birthday party.
  →The boy wanted to sing at the birthday party.

Stage 3: Surface realization
Also know as Speech Synthesis. This is to finally synthesize the text according the text specifications made in the previous stages.

- Structure realization: To mark up the text's surface structure, such as an empty line, or the boundaries between paragraphs, etc.
- Linguistic realization: To smooth the text by inserting function words, reorder word sequences, and select appropriate inflections and tenses of words, etc.

Other Architectures: Although the *Pipelined Architecture* provides a considerably articulate routine for text generation, it also provides predetermined restrictions for each stage in the process. Thus, the flexibility it can provide is limited, and is especially true for those sub-tasks in micro planning and surface realization stages. For example, the need for lexical selection can happen at any stage of the process. Thus, variations of the generic architecture and other methodologies have been discussed by many researchers (a recent discussion, Chris Mellish et al. [9]).

Key Applications

1. Routine documentation or information generation: examples of information are weather forecast descriptions, transportation schedules, accounting spreadsheets, expert system knowledge bases, etc. Examples of documentation are technical reports and manuals, business letters, medical records, doctor prescriptions, etc.
2. Literary writing: such as stories, poems, lyrics, couplets, etc. (Chinese couplet writer: generating a couplet sentence according to a given one. http://duilian.msra.cn).

Cross-references

- ▶ Text Summarization
- ▶ Text Representation
- ▶ Text Normalization
- ▶ Text Segmentation
- ▶ Text Analytics
- ▶ Text semantic Explanation

Recommended Reading

1. Dale R. Introduction to the special issue on natural language generation. Comput. Linguistics, 24 (3):346–353, 1998.
2. Goldman N.M. Computer Generation of Natural Language from a Deep Conceptual Base. Ph.D. thesis, Stanford University, CA, 1974.
3. Hovy E.H. Language generation, Chapter 4. In Survey of the State of the Art in Human Language Technology, G.B.Varile, A. Zampolli (eds.). Cambridge University Press, Cambridge, 1997, pp. 139–163.
4. Hovy E.H. Natural language generation. Entry for MIT Encyclopedia of Computer Science. MIT Press, Cambridge, MA, 1998, pp.585–588
5. Hovy E.H. Language generation. Entry for Encyclopedia of Cognitive Science, article 86. McMillan, London, 2000.
6. McDonald D.D. Natural Language Production as a Process of Decision Making Under Constraint. Ph.D. thesis, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1980.
7. McDonald D.D. *1*Natural language generation, Chapter 7. In Handbook of Natural Language Processing, Dale, R. H. Moisl, H. (eds.). Somers Marcel Dekker, New York, NY, 2000, pp. 147–180.
8. McKeown K.R. Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text. Cambridge University Press, Cambridge, 1985.
9. Mellish C, et al. A reference architecture for natural language generation systems. Nat. Lang. Eng., 12(1):1–34, 2006.

10. Reiter E. Has a consensus NL generation architecture appeared and is it psycholinguistically plausible? In Proc. 7th Int. Conf. on Natural Language Generation, 1994, pp. 163–170.
11. Reiter E. and Dale R. Building Natural Language Generation Systems. Cambridge University Press, Cambridge, 2000.

Text Index Compression

Gonzalo Navarro
University of Chile, Santiago, Chile

Synonyms

Inverted index; List; File compression

Definition

Text index compression is the problem of designing a reduced-space data structure that provides fast search of a text collection, seen as a set of documents. In Information Retrieval (IR) the searches to support are usually for whole words or phrases, either to retrieve the list of all documents where they appear (full-text searching) or to retrieve a ranked list of the documents where those words or phrases are most relevant according to some criterion (relevance ranking). As inverted indexes (sometimes also called inverted lists or inverted files) are by far the most popular type of text index in IR, this entry focuses on different techniques to compress inverted indexes, depending on whether they are oriented to full-text searching or to relevance ranking.

Historical Background

Text indexing techniques have been known at least since the 1960's (see, for example, the book *Automatic Information Organization and Retrieval*, 1968, by Gerard Salton, one of the pioneers in the area). Initially departing from the analog manual indexing process, where a short list of keywords was associated to each document from a collection, the increase in computational and storage capabilities quickly led to the so-called "full-text model", where every text word would be searchable (except for a few so-called "stopwords", which are too frequent and do not carry any meaning nor discriminative power, e.g., articles and prepositions). The so-called "inverted indexes" (or inverted lists, or inverted files), which are also modeled upon the traditional inverted index found at the end of books, have been since then the canonical model for indexing text collections. Most of the indexes used nowadays in Information Retrieval (IR) are variants of the inverted index. These mainly differ depending on the precise type of task that is to be carried out: Sometimes the application needs to find all the documents (and even exact positions within them) where some search terms appear; sometimes only a few "good" documents are wanted because the end user is a human.

Depending on the type of inverted index, it might take as little as 10% of extra space over the text size, or as much as 100% and even more. Nowadays the space, at least in secondary storage media, is extremely cheap and virtually unlimited. However, there are also extremely large text collections (for example the Web) where paying a significant amount of extra space for the index is not irrelevant. However, the most compelling reason to reduce the index size is the large gaps in the memory hierarchy: It is several orders of magnitude faster to transfer data from the main memory than from secondary storage. Hence, a reduction in space translates into an almost proportional increase in throughput at query processing, as the extra processing time for decompression is almost negligible. In networked environments, which are also becoming common to cope with the large demands in storage space and processing power, the same considerations apply with respect to network transfer time. For decades, CPU speeds have been increasing at an exponential rate, whereas disk speeds have not improved much. Moreover, new memory levels (caches) have appeared between the CPU and the main computer memory. This has only made it more and more attractive to design text indexes that fit in little space, even at the expense of needing more sophisticated decompression mechanisms. See [14,15] for a recent exhaustive coverage of the topic.

Foundations

An *inverted index* is formed by two main parts:

1. *Vocabulary.* Is the set of all the different *words* in the collection. The definition of what is a word may depend on the application. Not only does it involve delimiting them, but also determining which normalization processes will be carried out on them, e.g., upper/lower case conversion, removal of stopwords, stemming, etc. Once defined, the word becomes the unit of retrieval: one can search for words or for sequences of words (*phrases*), but not

for, say, a part of the word (although there are some exceptions in text retrieval systems offering extended functionalities which are not covered here, see e.g., [3]).

2. *Postings.* Is a list of "occurrences" of each vocabulary word in the collection. Depending on the application, the index might store the list of document identifiers where the word appears, or the list of exact positions (byte offsets or word offsets), or the list of document identifiers plus a "weight" associated to it, which computes according to some formula the importance of the word in that document, or the list of word positions plus data on the field where the word appears, or even color or font size of each occurrence.

For concreteness, this entry will assume that a policy for defining and normalizing words has been fixed, and will focus on the two most important type of posting lists: one storing exact word positions within documents (useful for full-text retrieval), and another storing document identifiers and weights (useful for relevance ranking). Because of the different processing needs, these lists might be stored in different orders. This is relevant because ordering is the key to inverted list compression.

A widely accepted statistical rule in IR [6] establishes that the vocabulary grows sublinearly with the collection size, more precisely as $v = O(n^{\beta})$ for some $0 < \beta < 1$ that depends on the collection. Different experiments show that β is actually around 0.5 [3], which means in practice that, even for very large collections, it is feasible to maintain the vocabulary in the main memory of the computer. For this reason, most of the efforts in index compression have focused on compressing the postings.

Inverted Indexes for Full-Text Retrieval

In full-text retrieval, queries are words or phrases and the task is to find *all* the documents where the word or phrase appears, possibly giving also the position(s) of the occurrences within each retrieved document. The most convenient index organization for word queries is a scheme storing basically all the answers to all the v possible queries: For each vocabulary word, the list of all the documents where it appears is stored in increasing order of document identifier. If exact occurrence positions are desired, and one does not want to sequentially scan the retrieved documents in order to find them, then the index must also store all the occurrence positions of each word within each document, also in increasing order. Because of the other types of queries that are usually necessary to support, it is usually advantageous to store the list of document identifiers contiguously and separated from the list of positions, so that the document identifiers alone can be retrieved with fewer disk accesses.

The important point is that the postings consist of *lists of increasing numbers* (document identifiers or occurrence positions). Moreover, for word queries, those lists are accessed from the beginning to the end. An obvious idea is to encode the *differences* between consecutive entries in the list. The longer the list, the smallest the differences between consecutive values. Because the number of occurrences of vocabulary words is usually highly skewed [3] (following a Zipf–Mandelbrot distribution [8]), there will be very long (and thus very compressible) lists and many short lists, where the long lists will contain a significant fraction of the whole data. Hence, a technique that represents those differences in a way that smaller numbers use a shorter representation should achieve compression. This requires that a different amount of bits is used to represent different numbers, as opposed to the classical data representation where a fixed number of bits is allocated for every possible number.

A first idea would be to use exactly the bits that the number uses, for example the binary representation of 9 is 1001_2, and hence one would like to represent it using 4 bits. The problem, of course, is that one cannot decode the individual numbers from the concatenation of their representations if one uses this method. For example, 10011101 could be 9,1,5 or 9,13. A large part of the research in inverted index compression refers to the design of *self-delimiting codes*, which (i) can be uniquely decoded from the concatenation of the codes, (ii) give shorter codes to smaller numbers, (iii) preferably can be efficiently decoded.

Several self-delimiting coding schemes are described in the book *Managing Gigabytes* [14], one of the best references on text index compression. Let $x > 0$ be a number to encode and $|x|$ the number of bits needed to code it (i.e., the highest bit set in the binary representation of x is at position $|x|$). Some of the most famous codes are described next.

1. Elias' γ-code emits $|x| - 1$ 0's followed by the $|x|$ bits representing x. For example, the code for

$x = 23 = 10111_2$ is 0000 10111 (which are here artificially separated in two parts). To decode the first number from a bit stream, one finds the next 1, and if one skipped $\ell - 1$ 0's before it, on reads the next ℓ bits starting from that 1. Those ℓ bits form x. This works because the representation of x always starts with a 1 (i.e., its most significant bit). Elias' γ representation of x takes $2|x| - 1 = 1 + 2\lfloor \log_2 x \rfloor$ bits. The representations of the first numbers are as follows: $1 = 1_2 \rightarrow 1$, $2 = 10_2 \rightarrow 010$, $3 = 11_2 \rightarrow 011$, $4 = 100_2 \rightarrow 00100$,...

2. Elias' δ-code first codes $|x|$ using Elias' γ-code, and then emits the $|x| - 1$ least significant bits of x (as one knows that the most significant bit is a 1). For example, the code for $x = 23 = 10111_2$, $|x| = 5 = 101_2$, is 00 101 0111. The length of the δ-code of x is $2||x|| - 1 + |x| - 1 = 1 + 2\lfloor \log_2(1 + \lfloor \log_2 x \rfloor) \rfloor + \lfloor \log_2 x \rfloor$. These codes are theoretically appealing because the space they require is $\log_2 x + O(\log \log x)$, which is asymptotically the same as just writing x. For example, it is not hard to prove that, using this scheme to encode all the word offsets of the occurrences, the whole index needs $nH_0 + O(n \log \log n)$ bits, where n is the number of words in the text collection and H_0 is its zero-order entropy seen as a sequence of words [9]. Thus the space is similar to that obtained when compressing the text using a good word-based zero-order compressor [4].
3. Rice codes are parameterized by a value r. The lowest r bits of x are represented in binary form (even if $|x| < r$), preceded by the unary representation of $\lfloor x/2^r \rfloor$. A different r value (the one yielding least space) can be chosen for each inverted list. An extension are Golomb codes, where instead of the quotient and remainder of the division by 2^r, any divisor d can be used. These are usually reported to achieve the least space consumption [14].

There are many other self-delimiting codes, such as Fibonacci codes [7], variable-byte codes [13], Dense codes [4], Simple codes [2], and many others. Some trade a slight loss in space for a large gain in decoding time.

Searching for Phrases Using Inverted Indexes

The described encodings of inverted lists must be decoded from the beginning, which as explained is appropriate to answer queries consisting of a single word. Phrase queries, however, are more complicated to deal with. If the index stores only document identifiers, all it can do is to intersect the list of documents of the different words, and then it is necessary to sequentially search the candidate documents for the exact phrases. If the index, instead, stores occurrence positions, it can look for the consecutive occurrences of the involved words within the same document. (This is especially convenient if the index stores word offsets rather than byte offsets of occurrences. Storing byte offsets, on the other hand, is more convenient to highlight the occurrences in the text.)

In both cases, a list intersection-like capability is necessary. Incidentally, note that another popular operation in inverted indexes is the conjunctive query, that is, find the documents where all the query words appear. Those are obviously solved by intersecting the lists of document identifiers as explained. Disjunctive queries (find the documents where some of the words appear) are solved by merging lists of document identifiers. Although there is not an especially clever way of merging lists other than traversing them all sequentially, much research has been done on the problem of efficiently intersection increasing lists of numbers. Although sequential intersection is an option as well, one can do better when one list is much shorter than the other, as it is frequently the case of the highly skewed length distributions that arise in natural language text collections [8].

Describing those techniques is not in the scope of this entry. A recent example of this research, which indeed takes compression into account, can be seen in [12]. In all cases the main idea is that one can search the longer list for the elements in the shorter list. This implies that some degree of random access is necessary in the encoded lists. For this sake, the natural choice [14] is to insert some absolute list values at regular intervals in the compressed list data, so that the search can be first carried out on those sampled values and only one chunk between consecutive samples must be decompressed. There is also some recent research on how those absolute values should be physically organized [5], as it turns out to be more convenient to store all the samples in contiguous form.

Note that, independently of the encoding technique used, the success of the compression depends on what is indexed. If one indexes the occurrences of all the text words, the space used by the index will be at best around 40% of that of the original (uncompressed) text, which can be reduced to around 20% by removing the stopwords. If only document identifiers are recorded, the space can be typically 10–20% depending on the size of the documents. A technique to achieve even less space, at the expense of higher time to solve queries, is called *block addressing* [10]. The idea is to cut the text into blocks, so that the index stores the blocks where each word appears. The index is used to filter out some blocks upon queries, and the others have to be sequentially traversed. Hence, the block size yields a space/time tradeoff, where reasonable performance on moderate-sized collections can be achieved with as little as 5% of index space overhead.

Finally, it is interesting to mention that there exist compressed indexes for what is (also) called "full-text searching", in the sense that the text is seen as a sequence of symbols and the index is able of retrieving any text substring (not only words or phrases) [9]. Those indexes were classically much larger than the text, and recent developments have shown that they can be compressed to sizes competitive to those of inverted indexes for word and phrase retrieval. Their search times are still larger than those for inverted indexes, although research is being carried out on applying them over a text regarded as a sequence of words (not letters). In this case they can also search only for words and phrases, but their space becomes even better than using inverted indexes, and performance for phrase searching is competitive.

Inverted Indexes for Relevance Ranking

Inverted indexes are also used to rank documents by "relevance" to a given query, so as to return a small set of those ranking higher. There are many formulas for computing relevance [3], yet the most popular ones build on two components: one is the *term frequency* $tf_{w,d}$, which is the number of times word w appears in document d, and the other is the *inverse document frequency* idf_w, which is the logarithm of the inverse of the fraction of the documents where word w appears. Note that, while idf_w is just one value per vocabulary word (and hence easy to maintain in main memory with the vocabulary), there is one $tf_{w,\ d}$ value per entry in the inverted index: For each word, the index stores the documents where it appears and the associated term frequency for each.

Because the relevance formula gives more weight to terms with higher term frequency, it is sensible to store the document identifiers of each word by decreasing term frequency value, rather than by increasing document identifier. This enables efficient algorithms that examine only a short prefix of the inverted lists [11]. The problem is that, although the decreasing term frequencies can be stored differentially, this is not possible anymore with the document identifiers accompanying them. Fortunately, because of Zipf–Mandelbrot law [8], it turns out that many of the term frequencies are small values, and therefore there are long runs of equal term frequencies, especially at the tail of the lists. Within those runs, one can still reorder the documents by increasing document identifier and encode them differentially. Recently, it has been shown that reducing the precision of the exact term frequency values is not only advantageous for compression purposes (which is obvious) but also for retrieval effectiveness [1].

Key Applications

Any application managing natural language text collections that have to be searched, and which are massive enough to discard sequential search as a solution, needs some kind of index. Index compression not only saves space, but more importantly, disk and network transfer time. A canonical example application are Web search engines.

Future Directions

Inverted index technology, even including compression, is rather mature. Still, it faces enormous efficiency challenges, especially those coming from Web search engines. The most active research fields related to inverted index compression are on encodings that permit fast decoding (e.g., [2]), and the interactions between information discarded to boost compression and the resulting retrieval quality of the index (e.g., [1]). The possibility of applying "true full-text indexes" [9] to natural language text is also extremely interesting, as it brings in radically new compression methods as well as algorithms for solving phrase queries. Yet, this trend is rather preliminary, and much research is needed to compete with inverted indexes in secondary memory scenarios.

Other challenges in inverted indexes, only mildly related to compression, are distributed indexes (how to split the collection and/or the indexes across multiple machines to boost performance), dynamic indexes (how to efficiently update the index when the collection changes), and extensions (how to cope with more complex queries, for example allowing approximate searches).

Experimental Results

Experiments can be found in the most recent cited papers, as most of them are of practical nature.

URL to Code

Probably the best known public domain implementation of compressed indexes was the *MG System* (http://www.cs.mu.oz.au/mg), closely related to the book *Managing Gigabytes* [14]. This system is now almost 10 years old, and is being replaced by its successor, *Zettair* (http://www.seg.rmit.edu.au/zettair).

Cross-references

▶ Inverted Indexes

Recommended Reading

1. Anh V. and Moffat A. Simplified similarity scoring using term ranks. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 226–233.
2. Anh V. and Moffat A. Improved word-aligned binary compression for text indexing., IEEE Trans. Knowl. Data Eng., 18(6): 857–861, 2006.
3. Baeza-Yates R. and Ribeiro-Neto B. Modern Information Retrieval. Addison-Wesley, Reading, MA, 1999.
4. Brisaboa N., Fariña A., Navarro G., and Paramá J. Lightweight natural language text compression., Inf. Retriev., 10:1–33, 2007.
5. Culpepper S. and Moffat A. Compact set representation for information retrieval. In Proc. 14th Int. Symp. String Processing and Information Retrieval, 2007, pp. 137–148.
6. Heaps H. Information Retrieval – Computational and Theoretical Aspects. Academic Press, New York, 1978.
7. Kautz W. Fibonacci codes for synchronization control. IEEE Trans. Inf. Theor., 11:284–292, 1965.
8. Mandelbrot B. An informational theory of the statistical structure of language. In Proc. Symp. on Applications of Communication Theory, 1952, pp. 486–500.
9. Navarro G. and Mäkinen V. Compressed full-text indexes. ACM Comput. Surv., 39(1):article 2, 2007.
10. Navarro G., Moura E., Neubert M., Ziviani N., and Baeza-Yates R. Adding compression to block addressing inverted indexes.. Inf. Retriev., 3(1):49–77, 2000.
11. Persin M., Zobel J., and Sacks-Davis R. Filtered document retrieval with frequency-sorted indexes. J. Am. Soc. Inf. Sci., 47 (10):749–764, 1996.
12. Sanders P. and Transier F. Intersection in integer inverted indices. In Proc. Workshop on Algorithm Engineering and Experiments, 2007.
13. Scholer F., Williams H., Yiannis J., and Zobel J. Compression of inverted indexes for fast query evaluation. In Proc. 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2002, pp. 222–229.
14. Witten I., Moffat A., and Bell T. Managing Gigabytes. 2nd edn. Van Nostrand Reinhold, New York, 1999.
15. Zobel J. and Moffat A. Inverted files for text search engines. ACM Comput. Surv., 38(2), 2006.

Text Indexing and Retrieval

Haoda Huang, Benyu Zhang
Microsoft Research Asia, Beijing, China

Synonyms

Document index and retrieval

Definition

Text indexing is a preprocessing step for text retrieval. During the text indexing process, texts are collected, parsed and stored to facilitate fast and accurate text retrieval. Text retrieval (also called document retrieval) is a branch of information retrieval in which the information is stored primarily in the form of text. Text retrieval is defined as the matching of some stated user query against a set of texts. As the result of text retrieval, texts are ranked and presented to the user according to their relevance with user query. User queries can range from a few words to multi-sentence full descriptions, which represent the user's information need.

Historical Background

Text indexing is the most fundamental part of a retrieval system. Over the past two decades, the corpus size of typical retrieval system has increased dramatically. The Text REtrieval Conference (TREC) (http://trec.nist.gov/) that started in 1992 only provides document collection consisting of less than 1 million texts in the 1990s. Today, the largest TREC test collection, named GOV2, is a 2004 crawl of the.gov top-level domain. GOV2 consists of approximately 25 million web pages (428GB of text). Moreover, a web search engine is believed to have far more items in its index. For example, in 2005, Yahoo claimed to have more than 20 billion items in its index, which is several orders larger than GOV2. The retrieval system is often required to return relevant document/text for a user query in a few seconds. Without an index, it is impossible for a retrieval system to achieve this task.

There are many indexing techniques. Among them, inverted index, suffix array, and signature are three typical examples. Signature files were popular in the 1980s, but since this scheme is inferior to inverted files in terms of speed, size and functionality, it is used less nowadays. Suffix trees are faster for phrase searches, but they are not easy to build and maintain. Finally, due to its simplicity and efficiency, inverted index is currently the most popular indexing scheme and is used for many applications.

Text retrieval is the core part of a retrieval system. There have been several decades of research on text retrieval, which has resulted in a fruitful range of beautiful, effective text retrieval models, such as Boolean model, vector model, and probabilistic model.

The Boolean model is one of the earliest models. Despite its simplicity, the Boolean retrieval model was popular among many large commercial information providers until the early 1990s. A noticeable advantage of the Boolean model is that in Boolean models, queries are represented in the form of a Boolean expression of terms, that is, an expression in which terms are combined with the operators AND, OR, and NOT. Many users prefer the Boolean query model because Boolean queries are precise and offer users greater control and transparency over the retrieved text. But the Boolean model also suffers from some major drawbacks. For example, the Boolean model judges a document to be either relevant or non-relevant in respect to a given query without any notion of grading scale, which does not achieve good performance in practice.

Different from the Boolean model, the vector model largely uses free text queries, which consist of one or more words but do not use operators to build up the query expressions. Query and documents are presented as a weighted vector over a fixed dictionary, and the documents are ranked by their Cos similarity with query. The vector model is better than Boolean model in that it is able to deal with cases in which documents only partially match the query. The famous tf-idf scheme for the vector model is very influential in the research of text retrieval.

Besides the simple Boolean model and vector model, probabilistic models have achieved great success regarding the text retrieval task. Probabilistic models are mainly guided by Probability Ranking Principle (PRP): "If a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data."

The Binary Independence Retrieval (BIR) model is one of the earliest probabilistic models for text retrieval. The model has other names, such as the Okapi model and the classical probabilistic model. Similar to the Boolean model, the BIR model documents are represented as binary vectors indexed over a fixed vocabulary. The representation ignores the number of times the term occurs, and is only able to capture whether or not a term occurs in a document. The 2-Poisson model was proposed to overcome this limitation. Under this model, documents are represented as vectors of term frequencies. Probabilistic models are not only solid from theoretical perspective, but also perform well in practice. An empirical, handcrafted approximation of the 2-Poisson model, called BM25, showed good performance over many data collections. It was first introduced at TREC in 1995, and is still a strong baseline for current text retrieval research.

The probabilistic language model, which was first applied to speech recognition and machine translation, has also been successfully applied to information retrieval in 1998 [4]. Since then, research to improving the language model has been an active area. Also, the probabilistic language model is a state-of-the-art text retrieval models for its robust, highly effective performance in the practice.

In all these models, documents are represented as a bag of words, where the exact ordering of the terms in a document is ignored but the number of occurrences of each term is material.

There are other models that go beyond the bag of words assumption, such as the Markov Random Field model, n-Gram language models, and the Indri Inference network model. The Markov Random Field model [2,3] for information retrieval, which was proposed in 2005 and improved in 2007, is demonstrated to be consistently and significantly better than probabilistic language model in several document collections, which shows that it is beneficial to incorporate dependency and features into text retrieval model.

Foundations

Among many text indexing schemes, the inverted index is the most popular. It could well support bag of words based text retrieval methods, like the tf-idf vector model, the BM25 model, and probabilistic language model. The process of building an inverted index is briefly described here. The major steps include:

1. Collecting the interested texts to form the text collection. For example, if the interested texts are the web pages on the internet, web pages should be crawled to form the text collection.
2. Tokenizing the text, which turns each document into a list of tokens. Generally, a text is represented as a continuous character stream. In this step, text is parsed and segmented into terms, digits, and email address and punctuations, such as ",", ".", "!", "?" and "-", are removed.
3. Doing linguistic preprocessing, which typically includes removing stemming and stop words, and then producing a list of indexing terms. Stemming refers to the process of reducing terms to their stems or root variants. For example, "computer," "computing," "compute" are reduced to "comput." For English, the most popular stemmer is Martin Porter's stemming algorithm. Stop words removal eliminates stop words from the documents. Common stop words include "a," "the," "of," "is" etc.
4. Creating the inverted index, which mainly consists of a dictionary and postings. The dictionary includes all the terms and the number of times they appear in the text collection. The postings include the text in which the terms appear. Thus the postings of a query could be obtained through the intersection operator of the query terms' postings.

With the support of the inverted index, many text retrieval models could be implemented efficiently. Here, the tf-idf vector model and probabilistic language model are taken as examples:

In the tf-idf vector model, either the query or document is represented as the weighted vector, with each component calculated as the product of term frequency (TF) and inverse document frequency (IDF) of the corresponding term. Term frequency is the number of times the term appears in a query or a document. The inverse document frequency of the term t is defined as

$$idf(t) = \log\left(\frac{N}{n_t}\right)$$

where N is the total number of texts in the collection and n_t is the number of documents with term t. The query vector is represented as $q = (tf_a(t_1) \cdot idf(t_1), tf_a(t_2) \cdot idf(t_2), \ldots, tf_a(t_n) \cdot idf(t_n))$, and the document vector is represented as $d = (tf_d(t_1) \cdot idf(t_1), tf_d(t_2) \cdot idf(t_2), \ldots, tf_d(t_n) \cdot idf(t_n))$. Then, documents are ranked by their Cos similarity with the query. The Cos similarity between the query and document is calculated as below:

$$sim(q, d) = \frac{\sum_{i=1}^{n} q_i \times d_i}{\sqrt{\sum_{i=1}^{n} q_i^2} \times \sqrt{\sum_{i=1}^{n} d_i^2}}$$

The TF and IDF together discriminate along two dimensions – informativeness (IDF) and aboutness (TF). The IDF component is used to discriminate between informative and non-informative query terms. Terms with high IDF occur rarely in the collection, and thus are considered more informative. The incorporation of TF component is based on the intuition that documents that contain more mentions of a term are more "about" this term.

The probabilistic language model for information retrieval is based upon the idea that a document is a good match to a query if the document model is likely to generate the query. Therefore, language model documents are ranked by query generation probability $p(q|d)$. There are several variant realizations for the language model. Among them, the query likelihood model [4] is the original and basic method for using language models in IR. The most common way to evaluate query generation probability $p(q|d)$ is using the multinomial unigram language model. Under this model, $p(q|d)$ is approximated as

$$p(q|d) = \prod_{w \in q} p(w|d)$$

Due to the data sparseness, the generation probability of query term absent in the document will be zero. Generally a smoothing technique is applied to overcome this problem. Please refer [6] for more details.

Key Applications

Text retrieval has many applications. It is used in digital libraries to help people quickly find desired books or articles. It could also be used in desktop search to help people instantly find documents, such as e-mail or other files, in a computer. Moreover, it is the fundamental basis of all internet search engines.

Future Directions

Most past research on text retrieval is based on the bag of words assumption, which has resulted in very fruitful models. These models have achieved rather good performance in the past, but appear to have reached a plateau; thus, their improvement has dwindled for several years. Recently, some text retrieval models (for example, Markov Random Field model for information retrieval) have tried to go beyond the bag of words assumption, and have achieved consistent and significant improvement. This indicates that it would be beneficial to incorporate dependency and features of documents into the text retrieval model; thus, more dedicated models may be developed in the future to improve retrieval performance further.

Data Sets

For testing the effectiveness of text indexing and retrieval strategies, TREC text datasets (http://trec.nist.gov/) are commonly used in the research community.

Cross-references

- ▶ Indexing
- ▶ Information Retrieval
- ▶ Inverse Document Frequency
- ▶ Term Frequency

Recommended Reading

1. Manning C.D., Raghavan P., and Schütze H. Introduction to Information Retrieval. Cambridge University Press, Cambridge, MA, 2008.
2. Metzler D. and Croft W.B. A Markov random field model for term dependencies. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 472–479.
3. Metzler D.A. Beyond bags of words: effectively modeling dependence and features in information retrieval, Ph.D. thesis, University of Massachussetts, 2007.
4. Ponte J. and Croft W.B. A language modeling approach to information retrieval. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 275–281.
5. Ricardo B.-Y. and Berthier R.-N. Modern Information Retrieval. Addison Wesley Longman, New York, NY, 1999.
6. Zhai C. and Lafferty J. A study of smoothing methods for language models applied to ad hoc information retrieval. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 334–342.

Text Indexing Techniques

Edleno Silva De Moura
Federal University of Amazonas, Manaus, Brazil

Definition

Text indexing is the act of processing a text in order to extract statistics considered important for representing the information available and/or to allow fast search on its content. Text indexing operations can be performed not only on natural language texts, but virtually on any type of textual information, such as source code of computer programs, DNA or protein databases and textual data stored in traditional database systems.

Historical Background

Efforts for indexing electronic texts are found in literature since the beginning of computational systems. For example, descriptions of Electronic Information Search Systems that are able to index and search text can be found in the early of 1950s [3].

In a seminal work, Gerard Salton wrote, in 1968, a book containing the basis for the modern information retrieval systems [5], including a description of a model largely adopted up to now for indexing texts, known as Vector Space Model. Other successful models for indexing texts were proposed since then, such as, for instance, Probabilistic Models and Language Models, which are discussed in detail in [2].

Text indexing operations have also received attention in the last decades as a basic operation in a large variety of applications. They are used in several basic algorithms related to data management, such as data integration and data disambiguation. Bioinformatics is another area where text indexing has been largely applied. In these cases, biological databases containing data such as DNA or protein information are indexed as texts in order to provide fast access to their content.

Foundations

The implementation of text indexing operations requires data structures for allowing fast access to the indexed information. The most used data structure for this purpose is known as *inverted index* or *inverted file*. An inverted index is a data structure composed of: (i) a vocabulary that contains all the distinct words indexed found in the text and (ii), for each word t of the vocabulary, a list that contains statistics about the occurrences of t in the text. Such list is known as the inverted list of t.

Inverted indexes allow fast search for statistics related to the distinct words found in a text. They are designed for using words as the search unit, which restricts their use in applications where words are not clearly defined or in applications where the system does not use words as the search unit. The statistics stored in an inverted index may vary according to the target application. Two non-exclusive alternatives usually found in literature are to record the position of all word occurrences in the text and to record general statistics about the word occurences in text units. Indexing all the word occurrences is useful in applications where positional information should be taken into account, such as when it is necessary to allow search for phrases or proximity queries. Inverted indexes that store word statistics are usually deployed in systems that adopt information retrieval models, such as the Vector Space Model, and in this case the text is divided into units of information, usually doucments. For instance, in web search engines, these units are the pages crawled from the web, while the whole set of pages compose the indexed text.

A third alternative to produce inverted indexes with intermediate space overhead and allow search for phrases and positional queries is to use a scheme known as block addressing [4]. Block addressing is a technique to reduce the space requirements of an inverted index. It was first proposed in a system called *Glimpse* [4]. The idea is that the text is logically divided into blocks, and the occurrences do not point to exact word positions but only to the blocks where the word appears. Space is saved because there are less blocks than text positions (and hence the pointers are shorter), and also because all the occurrences of a given word in a single text block are referenced only once.

Searching in a block addressing index is similar to search in a full inverted one. The pattern is searched in the vocabulary and a list of blocks where the pattern appears is retrieved. However, to obtain the exact pattern positions in the text, a sequential search over the qualifying blocks becomes necessary. The index is therefore used as a filter to avoid a sequential search over some blocks, while the others need to be checked. Hence, the reduction in space requirements is obtained at the expense of higher search costs.

The use of a block addressing scheme when indexing texts results in a tradeoff between index size and the computational costs for processing queries. The larger are the block units, the smaller is the final index overhead in terms of space. On the other hand, the larger are the block units, the larger are the portions of text that should be sequentially traversed when searching for a pattern in the text.

Although this combination of sequential search and index seems to produce less efficient search systems at first, block addressing indexes may achieve interesting combinations of cost for search time and space overhead. Block addressing was analyzed in [1], where it is analytically proved and experimentally verified that a block addressing index may yield sub-linear space overhead and, at the same time, sub-linear query time. Traditional inverted indexes pointing to words or documents achieve only the second goal. Empirical experiments performed in [?] indicate that in practice, $O(n^{0.85})$ space and query time can be obtained for exact queries in natural language texts.

Other data structures also adopted for indexing texts with the goal of allowing fast access to them are *signature files* and *suffix trees* [2,7]. Signature files are indexes that use a hash function to map the words found in a text to bit masks. The text is divided into blocks, and each block b is indexed by storing the result of a bit-wise OR operation over all the masks of the words that occur in b. This final mask represents the block in the signature file index. The search operation is performed through a bit-wise AND between the searched word and each block mask in the collection. Potential matches are reported whenever the bit-wise AND operation results in a number equal to the mask of the searched word. Note that false matches may arise, and thus the matched blocks should be inspected for confirming the occurrence of the word or not. Therefore, the signature file works as a filter for reducing the amount of text traversed in search operations.

The search in signature files is usually less efficient than the search when using inverted indexes. Further, inverted indexes support a larger set of search operations than signature files. For typical document indexing applications, signature files do not perform well compared to inverted indexes, being much larger and they are more expensive to build and update [8]. Thus, the use of inverted indexes in text indexing operations is more frequent than the use of signature files.

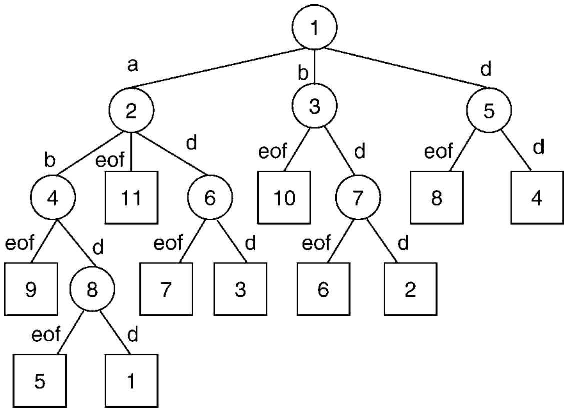
Another approach for indexing text databases regards the text as a long string. Suffix trees can index the text as a set of symbols according to the desired granularity of the search operations. For instance, by using suffix trees, it is possible to take all the characters of the text as index entries. This flexibility allows indexing texts that are written in languages where the words are not clearly separated from each other, such as it happens in some Asian languages, or even in texts where the concept of word does not exist, such as in DNA or protein databases.

Each position in the text is called a *suffix*. A suffix is defined by a starting position and extends to the right as far as needed or to the end of the text. Depending on the character sequences that are to be searched, the user must define the *index points*, i.e., which positions of the text will be indexed. In text databases, it is customary to index only word beginnings, but in many applications for text indexing it is necessary to index all the characters of the text. Suffix trees are data structures designed for indexing text using the suffix model, thus considering each entry point indexed in the text as a suffix of the whole text. In the suffix trees, each distinct path from the root to a leaf represents a unique suffix. For instance, consider the text fragment: "abadabadaba", which contains the suffixes described in Table 1.

Each suffix found in this example finishes with an implicit special symbol, marking the end of the text. Figure 1 describes the suffix tree for storing all the eleven suffixes of this text fragment. The circles in the trees contain internal nodes and their numbers indicate the position in the text where their children differ from each other. The square nodes are the leaves of the suffix tree and their numbers represent suffix ids (suffixes are usually represented by their position in the text). The letters marking each node in the tree indicate the value found on the positions where the node differ from their siblings nodes. The "eof" represents the end of the text is this example.

Text Indexing Techniques. Table 1. Suffixes found in the text fragment "abadabadaba"

Suffix	Suffix ID
"abadabadaba"	1
"badabadaba"	2
"adabadaba"	3
"dabadaba"	4
"abadaba"	5
"badaba"	6
"adaba"	7
"daba"	8
"aba"	9
"ba"	10
"a"	11



Text Indexing Techniques. Figure 1. Suffix tree for the text fragment "abadabadaba".

For instance, in the root node, the number 1 indicates that the suffixes are different in their first position. At this position, the suffixes may have value 'a', 'b' or 'c'. When searching for a suffix starting with an 'a', then the left most sub-tree should be taken in order to continue the search. In this case, the next comparison position to continue the search is position 2. The search continues until the current search position is longer than the search pattern, which means all the patterns in the sub-tree contain the search key; or until the current position in the search key does not match any child of the current node, which means the search key is not in the text.

Using this structure, it is possible to perform a variety of complex search operations at a relatively low computational cost. One of the drawbacks of using a suffix tree is the space required to store the index. If the space is a restriction, a more compact structure known as suffix array can be adopted. A suffix array is an array of pointers to each suffix indexed in the text. This array is sorted in the alphabetical order of the suffixes it represents and can then be used to perform search tasks. The use of suffix arrays increases the cost of search operations when compared to suffix trees. For instance, a given search pattern can be found in a suffix array at cost $O(log(n))$, where n is the number of suffixes in the text, while the cost for searching a simple pattern in a suffix tree is $O(m)$, where m is the size of the searched pattern, and is not affected by the number of suffix in the text.

A practical problem when using suffix trees or suffix arrays is the cost to build and maintain the indexes. Further, when searching for words inverted indexes are usually faster than suffix trees and suffix arrays. Exceptions may occur when it is necessary to process complex queries, such as regular expression patterns and search for word fragments.

Key Applications

Text indexing techniques have important practical applications, being of great importance in the construction of Web Search Engines, Web Directories and Enterprise Search systems. These techniques are also useful for allowing fast search on DNA or protein databases, which can also be treated as texts. In these cases, the text indexing techniques are adopted to accelerate more complex operations performed over the databases, such as the comparison of DNA fragments allowing small differences between them. Text indexing operations have also played an important role in several algorithms related to data management, such as data integration and data disambiguation.

Cross-references

- ▶ Inverted Files
- ▶ IR Retrieval Models
- ▶ Suffix Trees
- ▶ Text Retrieval

Recommended Reading

1. Baeza-Yates R. and Navarro G. Block-addressing indices for approximate text retrieval. J. American Soc. for Inf. Sci., 51(1): 69–82, 2000.
2. Baeza-Yates R. and Ribeiro-Neto B. Modern Information Retrieval. Addison Wesley, Reading, MA, 1999.
3. Luhn H.P. A statistical approach to mechanized encoding and searching of literary information. IBM Journal of Research and Development, 1(4):309–317, October 1957.
4. Manber U. and Wu S. Glimpse: A tool to search through entire file systems. In Proc. USENIX Winter 1994 Technical Conf., 1994, pp. 23–32.
5. Salton G. Automatic Information Organization and Retrieval. McGraw-Hill, New York, NY, 1968.
6. Salton G., Won A., and Yang C.S. A vector space model for automatic indexing. Inf. Retriev. Lang. Process., 18(11):613–620, November 1975.
7. Witten I., Moffat A., and Bell T. Managing Gigabytes, 2nd edn. Morgan Kaufmann, Los Altos, CA, 1999.
8. Zobel J., Moffat A., and Ramamohanarao K. Inverted files versus signature files for text indexing ACM Trans. Database Syst., 23(4):453–490, December 1998.

Text Mining

Yanli Cai[1], Jian-Tao Sun[2]
[1]Shanghai Jiao Tong University, Shanghai, China
[2]Microsoft Research Asia, Beijing, China

Synonyms

Knowledge discovery in text (KDT)

Definition

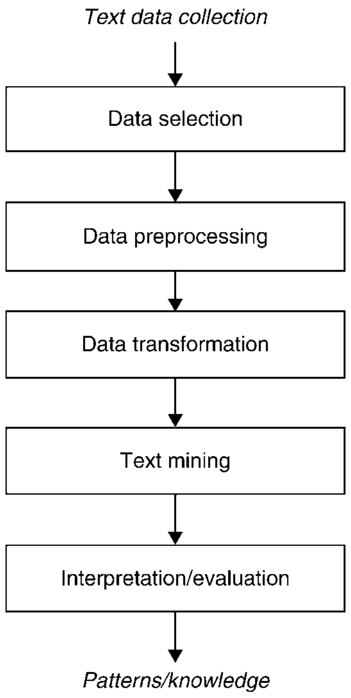
Text mining is the art of data mining from text data collections. The goal is to discover knowledge (or information, patterns) from text data, which are unstructured or semi-structured. It is a subfield of Data Mining (DM), which is also known as Knowledge Discovery in Databases (KDD). KDD is to discover knowledge from various data sources, including text data, relational databases, Web data, user log data, etc. Text Mining is also related to other research fields, including Machine Learning (ML), Information Retrieval (IR), Natural Language Processing (NLP), Information Extraction (IE), Statistics, Pattern Recognition (PR), Artificial Intelligence (AI), etc.

Historical Background

The phrase of *Knowledge Discovery in Databases (KDD)* was first used at 1st KDD workshop in 1989. Marti Hearst [4] first used the term of text data mining (TDM) and differentiated it with other concepts such as information retrieval and natural language processing.

Foundations

Typical steps of knowledge discovery in databases can be found in [7] by Usama Fayyad et al. The process of knowledge discovery in text data collections is similar, as shown in Fig. 1. Given one collection of text data as input, the goal is to extract patterns/knowledge from them. As the first step, data selection is to select proper data that will be processed and analyzed in the following steps. In the data preprocessing step, the task is to filter out noisy information and do some preliminary processes to facilitate the following steps, e.g., extracting name entities from the text data or part-of-speech tagging of the text data. In the data transformation step, the text data are converted to the format that is easy to be processed by mining algorithms, e.g., the format of vectors, sequences or inverted index tables. Text mining is to apply mining algorithms to find candidate patterns. In the interpretation/evaluation step, the candidate patterns produced in the former step are evaluated and the interesting ones are outputted as final knowledge.



Text Mining. Figure 1. Steps of knowledge discovery from text data.

Text preprocessing strongly affects the success of the outcome of text mining. Tokenization, or splitting the input into words, is an important first step that seems easy but is fraught with small decisions: how to deal with apostrophes and hyphens, capitalization, punctuation, numbers, alphanumeric strings, whether the amount of white space is significant, whether to impose a maximum length on tokens, what to do with non-printing characters, and so on [9]. It may be beneficial to perform some rudimentary morphological analysis on the tokens such as removing suffixes or representing them as words separate from the stem. Tokens may be standardized by using a dictionary to map different, but equivalent, variants of a term into a single canonical form. Once the input is tokenized, some level of syntactic processing is usually required. One operation is to remove stop words. Another is to identify common phrases and map them into single features. Tokenizing a document and discarding all sequential information yields the "bag of words." Additional linguistic preprocessing may also be needed [1]. Part-of-speech tagging (POS) determines the part of speech tag for each word, e.g., noun, verb, adjective. Text chunking aims at grouping adjacent words of a sentence. Word Sense Disambiguation (WSD) tries to resolve the ambiguity for individual words or phrases. Parsing produces a full parse tree for a sentence, which can identify the relation of each word in the sentence to all the others, and typically also its function in the sentence, e.g., subject, object.

Various statistical techniques and learning methods have been employed in text mining, including Naive-Bayes methods, support vector machines (SVM), decision trees, hidden Markov models (HMM), neural networks, etc. Major ways in which text is mined [1,9] are discussed in the following subsections.

Text Classification

Text classification is the assignment of text documents to predefined categories according to their contents. The pre-defined categories are symbolic labels with no additional semantics. When classifying a document, no information is used except for the document's content itself. A large number of feature selection and machine learning techniques have been applied to text classification. Typical approaches extract features from each document, and use the feature vectors as input to a scheme that learns how to classify documents. Using words as features and word occurrence frequencies as feature values, a model is built for each category. The documents in that category are positive examples and the remaining documents are negative ones. The model predicts whether or not that category is assigned to a new document based on the words in it, and their occurrence counts. Given a new document, each model is applied to determine which categories to assign. Automatic text classification has many practical applications, including indexing for document retrieval, automatically extracting metadata, word sense disambiguation by detecting the topics a document covers, and organizing and maintaining large catalogues of Web resources.

Text Clustering

Text clustering is unsupervised learning in which there is no predefined category or class, but groups of documents that belong together are sought. Clustering schemes not require training data to be preclassified, and the algorithms themselves are generally far more computation-intensive than supervised schemes. Usually the quality of clustering is considered better if the contents of the documents within one cluster are more similar and between the clusters more dissimilar.

Information Extraction

Information extraction is used to refer to the task of filling templates from natural language input, one of the principal subfields of text mining. A template is a composite structure with slots that are filled by individual pieces of structured information. A commonly-cited domain is that of terrorist events, where the template may include slots for the perpetrator, the victim, type of event, where and when it occurred, etc.

Machine learning has been applied to the information extraction task by seeking pattern-match rules that extract fillers for slots in the template. The rules can be expressed in pattern-action form, and the patterns comprise constraints on words in the surrounding context and the slot-filler itself. These constraints involve the words included, their part-of speech tags, and their semantic classes. An application of information extraction is extracting information from job ads such as those posted on Internet newsgroups.

The extracted information by information extraction can also be used in a subsequent step to learn rules that characterize the content of the text itself.

Document Summarization

A text summarizer strives to produce a condensed representation of its input, intended for human consumption. Useful distinctions can be made between different kinds of summaries: an extract picks certain key sentences scattered throughout the document. In contrast, an abstract contains material that is not present in the input, or at least expresses it in a different way. An indicative summary's main purpose is to suggest the contents of the article without giving away details of the article content. It can serve to entice the user into retrieving the full form [5]. Book jackets, card catalog entries and movie trailers are examples of indicative summaries. An informative summary is meant to represent the original document. Therefore it must contain all the pertinent information necessary to convey the core information and omit ancillary information. Another distinction is between a generic summary, aimed at a broad readership, and a topic-focused one, tailored to the requirements of a particular group of users. Summaries may also be produced from a single document or multiple documents [3]. In single-document summarization, features like word frequency, key phrases, sentence position, sentence length and uppercase words are used. Multi-document summarization extracts a single summary from multiple documents, and is used in the domain of news articles. It departs from single-document summarization since the problem involves multiple sources of information that overlap and supplement each other, being contradictory at occasions. So the key tasks are not only identifying and coping with redundancy across documents, but also recognizing novelty and ensuring that the final summary is both coherent and complete.

Key Phrase Extraction

Keywords and key phrases are attached to documents to give a brief indication of what they are about.

Key phrases are a useful form of metadata because they condense documents into a few pithy phrases that can be interpreted individually and independently of each other. In key phase extraction, all the phrases that occur in the document are listed and information retrieval heuristics are used to select those that seem to characterize it best. Most key phrases are noun phrases, and syntactic techniques may be used to identify these and ensure that the set of candidates contains only noun phrases. The heuristics used for selection range from simple ones such as the position of the phrase's first occurrence in the document to more complex ones such as the occurrence frequency of the phrase in the document versus its occurrence frequency in a corpus of other documents in the subject area.

Topic Detection and Tracking (TDT)

Topic Detection and Tracking (TDT) [8] refers to a variety of automatic techniques for discovering and threading together topically related material in streams of data such as newswire and broadcast news. The TDT research applications keep track of topics or events of interest, in a constantly expanding collection of multimedia stories. There are five research applications defined in the TDT Program. Story Segmentation detects changes between topically cohesive sections. Topic Tracking keeps track of stories similar to a set of example stories. Topic Detection builds clusters of stories that discuss the same topic. First Story Detection detects if a story is the first story of a new, unknown topic. Link Detection detects whether or not two stories are topically linked. Shared resources, such as TDT corpora, language resources and evaluation software, provide the necessary tools to build a TDT application. The TDT corpora consist of broadcast news and newswire texts sampled daily during most of 1998. The Linguistic Data Consortium (LDC) exhaustively annotated the corpora by identifying which stories discuss a predefined set of topics.

Opinion Mining

Opinion mining [8] refers to the research work of mining user opinion data. It is also known as sentiment analysis. Typical research problems include: (i) subjectivity/objectivity classification is to identify if the text data contains user opinion; (ii) sentiment classification is to predict the polarity of user sentiment (e.g., positive or negative); (iii) opinion summarization is to provide a condensed representation for a set of user opinion texts; (iv) opinion anti-spamming is to detect if the opinion data are written by review spammers.

Key Applications

Bioinformatics

Bioinformatics is the study of the information content and information flow in biological systems and processes [6]. Text mining can be applied to bioinformatics literatures for named entity recognition and relationship extraction. Named entity recognition identifies entities such as drugs, receptors, enzymes, toxins, genes and their features (box, chain, sequence, subunit, etc.). Relation extraction detects the relationship between a pair of entities such as the relationship between genes, protein, or other biological entities.

Email Spam Filtering

The explosive growth of unsolicited emails, more commonly known as spam, has been undermining constantly the usability of emails. One solution is provided by spam filters. Some spam filters use black lists and hand-crafted rules, which are not easy to adapt to new types of spam. On the other hand, the success of machine learning methods in text classification provides the possibility to compete with rule-based filters and quickly adapt to new types of spam. Many spam filters based on machine learning are using Naive-Bayes classifiers. A prominent example is Mozilla's email client. Different classifier methods such as SVM are also used.

Business Intelligence

Business intelligence (BI) is a broad category of applications and technologies for gathering, storing, analyzing, and providing access to data to help enterprise users make better business decisions. BI applications include decision support systems, query and reporting, online analytical processing, statistical analysis, forecasting, and data mining. People express their opinions and post reviews of products on the Web. Opinion mining [3] identifies whether users like or dislike a product and products summary from the reviews. Opinion mining can help manufactures to identify problems in their products, and also provides valuable information for placing advertisements in Web pages.

URL to Code

Tools for basic processes of text mining: http://nlp.stanford.edu/links/statnlp.html

Cross-references

- ► Biomedical Scientific Textual Data Types and Processing
- ► Data Cleaning
- ► Data Mining

- ▶ Information Extraction
- ▶ Information Retrieval
- ▶ Opinion Mining
- ▶ Text Categorization
- ▶ Text Clustering
- ▶ Text Summarization

Recommended Reading

1. Andreas H., Andreas N., and Gerhard P. A brief survey of text mining. J. Computat. Linguistics Lang. Technol., 20(1): 19–62, 2005.
2. Bing L. Web Data Mining: Exploring Hyperlinks, Contents and Usage Data. Springer, Berlin, 2007, pp. 411–447.
3. Dipanjan D. and Martins A.F.T. A Survey on Automatic Text Summarization. Literature Survey for the Language and Statistics II course at Carnegie Mellon University, November, 2007.
4. Hearst M. Untangling text data mining. In Proc. 27th Annual Meeting of the Assoc. for Computational Linguistics, 1999.
5. Informative and indicative summarization. Available at: http://www1.cs.columbia.edu/~min/papers/sigirDuc01/node2.html
6. Liebman M. Bioinformatics: an editorial perspective. Available at: (http://www.netsci.org/Science/Bioinform/feature01.html)
7. Usama F., Gregory P.-S., and Padhraic S. From data mining to knowledge discovery in databases. AI Mag., 17(3):37–54, 1996.
8. Wayne C.L. Multilingual topic detection and tracking: successful research enabled by corpora and evaluation. In Proc. Conf. on Language Resources and Evaluation, 2000.
9. Witten I.H. Text mining. In Practical Handbook of Internet Computing, M.P. Singh (eds.). Chapman and Hall/CRC Press, Boca Raton, FL, 2005, pp. 14-1–14-22.

Text Mining of Biological Resources

PADMINI SRINIVASAN
The University of Iowa, Iowa City, IA, USA

Synonyms

Knowledge discovery from biological resources; Hypothesis generation and exploration from biological resources; Literature-based discovery from biological resources

Definition

Text mining is about automatically or semi-automatically exploring hypotheses or new ideas from a set of resources. The mined hypotheses require further tests with methods native to the discipline, in this case with scientific methods in biomedicine. An overall goal in text mining is to support the intellectual activities of biomedical scientists as they explore new ideas using a collection of resources. Text mining is similar to data mining. But instead of mining a collection of well-structured data, text mining operates off semi-structured text collections. Current text mining efforts in biomedicine increasingly involve more structured data sources such as the Entrez Gene database maintained by the National Library of Medicine (NLM).

There is some diversity of opinion on the *kinds* of research that fall within the realm of text mining. As an example, some include text classification, which is about building models to predict one or more topical categories for a text. Still others include information extraction goals such as in research on identifying named entities, definitions and expansions of abbreviations in texts. Some go so far as to include text retrieval, i.e., research on finding documents relevant to a user query. Despite these diverse viewpoints, there is wide agreement that a core focus of text mining is on hypothesis generation and exploration based upon *implicit* connections present in the sources. Consistent with this viewpoint, Blagosklonny and Pardee [1] refer to text mining as conceptual biology, a field that fuels hypothesis-driven biomedical exploration. These hypotheses typically postulate relationships between at least two entities (or more generally concepts). The emphasis is also on novelty at least in the context of the mined sources. Note also that novelty itself eludes definition as it is difficult to pinpoint when and how an idea acquires the status of being *known*.

Historical Background

The motivation underlying biomedical text mining with its focus on hypothesis generation and exploration is that it is difficult for any one scientist or even a group of collaborators to keep abreast with research developments. This difficulty becomes compounded many times given the increasing importance of interdisciplinary perspectives to solve biomedical problems. Another motivation comes from the serendipitous nature of many scientific discoveries. Serendipity may be influenced by several intangibles, such as researcher intuition, prior experience and knowledge, including also the ability to creatively scan and combine the literature of multiple disciplines. With the biomedical literature growing at an astounding pace, there is a definite need for text mining tools to assist bioscientists gauge new ideas against prior research.

Text mining has its origins in the work of Swanson conducted during the mid 1980's. In 1986, Swanson mined the MEDLINE bibliographic database and proposed that fish oils may be used to treat Raynaud's disease [13]. Swanson observed from the literature that Raynauds is exacerbated by platelet aggregability, vasoconstriction, and blood viscosity. He also observed that fish oils reduce these phenomena. Combining these observations he postulated that fish oils may be beneficial for persons with Raynauds which was later corroborated by other scientists. The decade from the mid-1980's to the mid-1990's is marked by a series of papers by Swanson and his collaborator Smalheiser on using text mining to propose a variety of other hypotheses such as connections between estrogen and Alzheimer's disease [10]. These papers created a fertile for text mining research, especially in the context of MEDLINE. Remarkably, it was only in the mid-1990's that other researchers became seriously attracted to biomedical text mining. The first few papers were on automating several of the text mining steps in Swanson and Smalheiser's methodology [5,11]. Since then researchers have proposed other hypotheses such as viruses that may be used as bioweapons [14], possible therapeutic uses for substances such as thalidomide [16] and for turmeric (*Curcumin Longa*) [12], possible functional connections between genes as well as between genes and diseases [9]. Now past the 20 year mark, the text mining field is a fertile ground for research and development. It is mature to the point that several annual workshops affiliated with major conferences have been established. There is also a Journal of Biomedical Discovery and Collaboration that is dedicated to the field and finally there are recent reviews as for example on biomedical text mining tools [15].

Foundations

Resources

A central aspect to text mining in biomedicine is the use of MEDLINE, the bibliographic database produced by the National Library of Medicine. MEDLINE records contain a variety of fixed and variable length fields such as publication date, title, abstract, authors and author affiliations, chemical terms and subject representations in the form of assigned phrases. Phrases assigned to a record are selected from the MeSH (Medical Subject Headings) controlled vocabulary by trained indexers. Text mining systems sometimes differ in the MEDLINE field(s) used. Some methods use the free-text fields such as title and abstract [5] while others are based on controlled vocabulary fields such as MeSH and chemical terms (e.g., [11]). Systems using controlled vocabularies have the advantage of more precise vocabulary. But they take the risk of missing important information that may be present in the free-text alone. Systems using the free-text fields usually involve procedures for information extraction to identify key concepts that occur in the texts such as gene, protein, disease and drug name. In this regard research on natural language processing in the biomedical domain has had a huge influence on text mining. This direction is of increasing importance as other forms of textual collections such as patient records are brought into the fold of text mining.

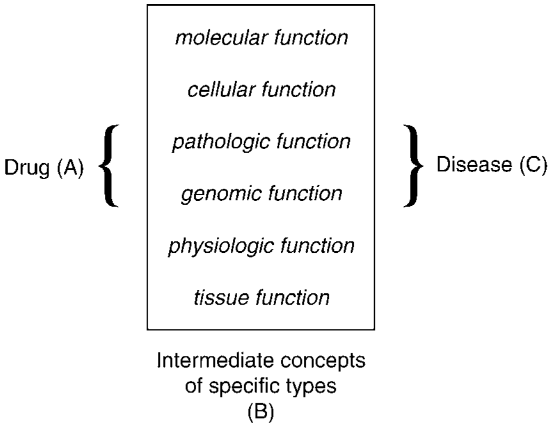
Many text mining approaches also avail of allied vocabulary resources such as the UMLS (Unified Medical Language System) also produced by the NLM. The UMLS offers a rich variety of conceptual and linguistic details and it also offers interconnections between the many general and specialized vocabularies arising from different subfields of biomedicine. Other core resources seen, especially in applications targeting bioinformatics, are Entrez Gene [3] and Gene Ontology [4]. Information about annotation links between GO terms and genes frequently accompanied by the MEDLINE records providing supporting evidence, are utilized in text mining. Beyond these core resources several others have been used for text mining such as DIP, the Database of Interacting Proteins [2].

Methods

There is a growing variety of text mining methods, orientations and applications. In general, methods appear to be selected or designed to fit the problem at hand. And since the space of text mining problems is broad (and growing), there is an almost bewildering array of text mining solutions. While this situation offers almost free rein to researchers and developers, it also makes it challenging to determine what methods (or aspects about methods) are most successful or most appropriate for a given problem or in a specific domain. Seemingly similar problems are sometimes addressed using significantly different approaches while certain approaches exhibit broader appeal.

Two established text mining approaches derive from Swanson and Smalheiser's early research. These are generally referred to as *open* discovery and *closed* discovery. The open discovery process is initiated with a single concept (A) and the goal is to identify one or more concepts (C) that are connected but only indirectly with A, i.e., through other intermediate concepts. In closed discovery, two concepts A and C are provided as input and the idea is to seek out novel connections between them.

Figure 1 displays these strategies. An example of open discovery represented by the figure is where the starting A concept is a specific drug and the user is interested in novel connections with C concepts representing diseases. Moreover, the intermediate (B) concepts could be constrained to concepts of particular types such as functions of different kinds, molecular, tissue etc. Or, they may be constrained to concepts designating genes already known to be associated with the disease. The novelty aspect is satisfied by ensuring that the A drug and the identified C disease have not yet been studied together. A typical approach implemented to satisfy this requirement is that the A-B concept(s) and B concept(s)-C connections should be found in disjoint portions of the literature collection or that the A-C connection is not recorded in an appropriate knowledge source. For closed discovery an example illustrated by the figure is where the user inputs both a specific drug (A) and a disease (C). The algorithm then looks for new connections, i.e., B concepts that tie the two together. Now it may be the case that no connections between the drug and the disease are known thus far or it may be that some connections are known and other novel ones are being sought. In all of these it can be seen that investigator participation is of value not only to specify the inputs but also to specify the kinds of outputs and to constrain the intermediate connection types of interest. Typically when multiple novel C concepts are discovered through open discovery or multiple B concepts discovered with closed these are ranked by some estimate of confidence.



Text Mining of Biological Resources. Figure 1. Open and closed discovery.

Many variations of these two basic strategies have been explored. One type of variation extends the transitive nature of these methods to allow for implicit connections ranging over longer distances, i.e., with longer connecting path lengths. For example in G2D [7] researchers start by connecting a disease to its pathological conditions and then to their co-occurring chemical terms in MEDLINE. These are in turn connected through co-occurrence with terms representing protein function from Gene Ontology (GO). These GO terms are then used to identify RefSeq [8] sequences through annotation links. Homology is then used to connect these with candidate sequences that are then constrained to those mapping to the same chromosomal region where the disease is mapped. Another key distinguishing feature across implementations of these text mining algorithms is that confidence estimates may be made in different ways. There is certainly variability in the particular confidence estimation functions used. But more generally, some utilize weights along the single best path while others utilize some function (such as a mean or median) computed over all the connecting paths. Still others, inspired by association rules, adopt the dual notions of confidence and support while several utilize symmetric measures exploiting concept co-occurrence based statistics.

Besides open and closed discovery, there are other text mining methods that rely on exploring graph properties. A graph may be constructed where the nodes represent biomedical objects and the links represent their interconnections. For example, gene networks have been studied by many researchers. Here links between pairs of genes may be directed to indicate influence of one gene over another and weighted by some estimate of degree of influence. Or, these may be undirected with link weight being a function of co-appearance in MEDLINE records. Such graphs may be studied for their structural properties including to identify core groups of objects (here genes).

Unexpected members of such groups may suggest ideas and lead to specific hypotheses and further research.

A recent trend is to embed text mining functions in systems designed with larger scope. iHOP is an example of such a system [6]. In such systems hypothesis generation becomes one of several sub goals. Wide ranging functions are offered by such systems such as easy connections between records of different databases and knowledge sources, literature retrieval and ranking as well as syntactic analysis of the text sentences to extract entities and relationships.

Key Applications

For pointers to current text mining applications the reader may explore sources such as the *Application Notes* sections of Bioinformatics (*Data and Text Mining* subsection); the *Software* and *Database* sections of BMC Bioinformatics; the *Web Server* issue of Nucleic Acid Research and similar sections of other biomedical journals.

Future Directions

Biomedical text mining is at a highly creative age with its scientific basis still in infancy. The area has been heavily influenced by research and development in several fields such as text retrieval, machine learning and computational linguistics. The rapid growth in terms of research papers is a strong indicator of its perceived potential.

There are several open problems in text mining. Certainly the relative merits of alternative methods, their generalizability and criteria for gauging relevance to specific application contexts are still open. There is also a need to obtain a deeper understanding of what is meant by a *novel* idea or hypothesis, as this is a key motivation for text mining research. Methods for evaluating text mining systems is also an open problem. Evaluations tend to be somewhat subjective and non standardized. A common strategy is to see if the algorithms can discover knowledge that is already recorded in sources such as the Database of Interacting Proteins [2]. Another strategy is to obtain the opinion of domain specialists on the connections found. Both strategies have their limitations. Also important is research on how to successfully integrate text mining systems into the work patterns of bioscientists. This will require a better understanding of research strategies used by bioscientists. Overall it remains to be seen how well text mining will address the needs of the biomedical research community. For the moment, at the very least, it is clear that text mining has significantly extended the frontiers of text based applications in biomedicine.

Cross-references

► Data Mining
► NLP Techniques for Biomedical Text Analysis and Mining
► Text Mining

Recommended Reading

1. Blagosklonny M.V. and Pardee A.B. Unearthing the gems. Nature, 416, 373, 2002.
2. Database of Interacting Proteins: http://dip.doe-mbi.ucla.edu/
3. Entrez Gene: http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene
4. Gene Ontology: http://www.geneontology.org/
5. Gordon M.D. and Lindsay R.K. Toward discovery support systems: A replication, reexamination, and extension of Swansons work on literature-based discovery of a connection between Raynauds and fish oil. J. Am. Soc. Inf. Sci., 47, 116–128, 1996.
6. iHOP: http://www.ihop-net.org/UniPub/iHOP/
7. Perez-Iratxeta C., Bork P., and Andrade M.A. Association of genes to genetically inherited diseases using data mining. Nat. Gene., 31(3):316–319, 2002.
8. RefSeq, http://www.ncbi.nlm.nih.gov/RefSeq/
9. Seki K. and Mostafa J. Discovering implicit associations between genes and hereditary diseases. Pacific Symp. Biocomput., 12:316–327, 2007.
10. Smalheiser N.R. and Swanson D.R. Linking estrogen to Alzheimers disease: an informatics approach. Neurology, 47:809–810, 1996.
11. Srinivasan P. Text mining: generating hypotheses from MEDLINE. J. Am. Soc. Inf. Sci. Technol., 55:396–413, 2004.
12. Srinivasan P. and Libbus B. Mining MEDLINE for Implicit Links between Dietary Substances and Diseases. Bioinformatics, 20 (Suppl 1):I290–I296, August 2004.
13. Swanson D.R. Fish oil, Raynauds syndrome, and undiscovered public knowledge. Persp. Biol. Med., 30:7–18, 1986.
14. Swanson D.R., Smalheiser N.R., and Bookstein A. Information discovery from complementary literatures: categorizing viruses as potential weapons. J. Am. Soc. Inf. Sci. Technol., 52:797–812, 2001.
15. Weeber M., Kors J.A., and Mons B.Online tools to support literature-based discovery in the life sciences. Brief. Bioinform., 6(3):277–286, 2005; doi:10.1093/bib/6.3.277
16. Weeber M., Vos R., Klein H., de Jong-Van den Berg L.T.W., Aronson A., and Molema G. Generating hypotheses by discovering implicit associations in the literature: a case report for new potential therapeutic uses for Thalidomide. J. Am. Med. Inform. Assoc., 10:252–259, 2003.

Text Representation

JUN YAN
Microsoft Research Asia, Haidian, China

Definition

Text representation is one of the fundamental problems in text mining and Information Retrieval (IR). It aims to numerically represent the unstructured text documents to make them mathematically computable. For a given set of text documents $D = \{d_i, i=1, 2,...,n\}$, where each d_i stands for a document, the problem of text representation is to represent each d_i of D as a point s_i in a numerical space S, where the distance/similarity between each pair of points in space S is well defined.

Historical Background

Mining the unstructured text data has attracted much attention of researchers in different areas due to its great industrial and commercial application potentials. A fundamental problem of text mining is how to represent the text documents to make them mathematically computable. Various text representation strategies have been proposed in the past decades for different application purposes such as text categorization, novelty detection and Information Retrieval (IR) [5]. This entry focuses on the text representation strategies specifically for IR applications.

Nowadays, the most commonly used text representation model in the area of Information retrieval is called as the Vector Space Model (VSM) [4,5]. It aims representing each text document by a numerical vector such that the similarity between vectors (documents) can be computed by different kernels. A simple and commonly used kernel is their normalized inner product, which is also known as the Cosine similarity. One of the commonly used VSM is the Bag of Words model (BOW). It uses all words appeared in the given document set D as the index of the document vectors. Under the BOW model, different term weighting schema give different text representation results. The simplest case of BOW is the Boolean model. It utilizes the binary vectors to represent text documents. In other words, if a term appears in a document, there has a "1" in the position, which corresponds to this term, in the document vector. Otherwise, the term weight is "0". As an extension of the Boolean model, Term Frequency Inversed Document Frequency (TFIDF) model was proposed. It uses real values which capture the term distribution among documents to weight terms in each document vector. However, there are many limitations in the traditional BOW text representation model. For example, (i) BOW ignores the within document term correlation such as the order of terms in a given document; (ii) the polysemy and synonymy problems can greatly decrease the IR performance in the TFIDF text representation model; and (iii) the TFIDF model cannot capture the semantics of documents for IR.

To solve the limitations of BOW model, various advanced text representation strategies have been proposed. The N-gram statistical language models [2] were proposed to capture the term correlation within document. However, the exponentially increasing data dimension with the increase of N limits the application of N-gram models. The Latent Semantic Indexing (LSI) [3] was proposed to reduce the polysemy and synonym problems. At the same time, LSI can also represent the semantics of text documents through the linear combination of terms, which is computed by the Singular Value Decomposition (SVD). However, the high complexity of SVD [1] make LSI seldom used in real IR tasks. In addition, some external resources such as the Wordnet and Wikipedia are recently used for solving the polysemy and synonym problems in text representation. Since the effectiveness of these external resources for text representation can only be learned in research papers and there still has no evidence to show their power in real IR applications, this article will not introduce their details. Motivated by the LSI, the Probabilistic Latent Semantic Indexing (PLSI) [6] is also proposed for representing the semantics of text documents. However, it is still limited by the computation complexity due to the increasing scale of Web data. As a summary, though various approaches have been proposed for solving the limitations of BOW, the BOW with TFIDF term weighting schema is still one of the most commonly used text representation strategies in real IR applications.

Beyond VSM, many propose to represent text documents in other formats instead of vectors or represent text documents through their meta-information. For instance, the text documents can be represented through neural network, through graphs and in tensor space model. The text documents in Web pages can be semantically represented by the search queries which have clicked these pages, the social bookmarks which have annotated these pages. However, most of them are used for solving specific text mining problems or used to enhance performance of some special IR tasks. In the next Section of this article, the details for text representation in two parts will be given, which are the traditional BOW text representation model with TFIDF term weighting schema and the semantic text representation through LSI.

Foundations

In the bag of words (BOW) model, a text document is represented as a collection of unordered terms. Given the document collection D={d_i, i=1, 2,...,n}, suppose there are m unique terms appeared in this collection (The stop words removal and stemming will be introduced later). Mathematically, this corpus of documents can be represented by a m by n matrix $S \in R^{m\times n}$. Each text document is denoted by a column vector $s_i, i = 1, 2,\ldots,n$ and each term is denoted by a row vector. The j^{th} entry of s_i is denoted by $s_{ji}, j = 1, 2,\ldots,m$. As an example, suppose the document collection D implies two documents,

d_1: He investigates the text representation approaches.
d_2: What is the meaning of text representation approach for text documents?

There are a list of 13 unique terms, which are,

“He, investigates, the, text, representation, approaches, What, is, meaning, of, approach, for, documents”.

The list of terms roughly represents the two documents by a 13 by 2 matrix. The problem is how to weight each entry of this matrix. Considering the simple Boolean model first. If a term appears in a document, its corresponding weight is 1; otherwise, it is 0. The transform of the matrix for the collection D is,

$$S^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

where the order of term index is the same as the term list given above, i.e., the first term is “He” and the last term is “documents.” There are three obvious problems in this text representation strategy: (i) not all terms have the physical meaning in representing text documents such as “the,” “is” etc; (ii) some terms such as “approach” and “approaches” are actually the same; and (iii) the importance of all the terms is treated as the same in this strategy. Can the difference of the term importance be reflected in text representation? The answers for these three questions correspond to three key steps in text representation, which are *stop words removal*, *stemming* and *TFIDF indexing*.

The stop words (or stopwords) is the name of the terms that should be filtered out before text documents indexing or natural language processing. There has no fixed stop words list for all text processing applications. Generally the stop words list will include the terms like “a,” “the,” “an,” etc. After removing the stop words, a stemming procedure is applied before the TFIDF indexing. The stemming aims at reducing inflected words to their stem. For example, “approaches” is stemmed to “approach,” “investigates” is stemmed to “investigate” and “representation” is stemmed to “represent.” Thus in the above example, the list of terms is reduced to,

“He, investigate, text, represent, approach, what, mean, document.”

Thus the two documents d_1 and d_2: can be represented by an 8 by 2 matrix. The transpose of it is,

$$S^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Until now, the problems (i) and (ii) have been solved. The last problem is how to tell the importance of different terms in text representation. One of the commonly used approaches is called as the TFIDF indexing. The TFIDF aims to assign a weight to each term according to a document in a collection or corpus. In other words, TFIDF aims to assign different weights for all entries s_{ji} in matrix S. An intuition is that the more times a term appears in a document, the more important is this term to this document. Thus the weight should increase proportionally to the number of times a term appears in the document. On the other hand, if a word appears in many documents in the corpus, the discriminative power of the term will be weak. Thus the weight is offset by the frequency of the word in the corpus. The former step is called the Term Frequency (TF). It can be counted directly from the documents. As an example, the TF of term stext” in d_2 is 2. A normalizing factor is always used for calculating the TF of a term in a document. Since in d_2 there are 7 terms after stop words removal and stemming. Among them, 2 of them are "text." Thus the TF for term "text" in d_2 is 2/7. The latter step is called the Inverse Document Frequency (IDF). It is a log function. The IDF for term t is,

$$\log \frac{n}{\#\text{document involve term } t}$$

Thus the TFIDF weighting schema can be as simple as TF*IDF. There are various variations of TFIDF text indexing. The major differences are how to normalize and smooth the weighting equation.

In the BOW model, the TFIDF indexing gives a way to weight the terms for text documents. However, this kind of term frequency based approaches cannot discover the semantics of the text documents. In the following several paragraphs, the Latent Semantic Indexing (LSI) is briefly introduced, which aims to discover the text semantics through linear combination of term weights. As introduced above, the term by document matrix is a sparse matrix whose rows correspond to terms and columns correspond to documents. LSI aims to transform the matrix S into a reduced matrix which can reflect the relation between the documents and some *concepts*. Thus the terms and documents are indirectly related through the concepts.

Mathematically, LSI aims to find a projection matrix $W \in R^{m \times p}$ such that the linear projection $y_i = W^T s_i \in R^p, i = 1, 2, \ldots, n$ can reflect the p semantic concepts implied by document d_i, where $p << m$. In other words, LSI assumes that the linear combination of terms can reflect the concepts in text documents. $W \in R^{m \times p}$ can give p different linear combinations for term weights vector s_i. Through the linear projection matrix W, the text document d_i which is represented by m terms' weights in vector s_i is represented by p concepts' weights in vector $y_{i.}$ The problem left is how to get the projection matrix W, i.e., how to get the weights for the linear combination of terms, from the matrix S. The calculating of W can be formulated from different perspectives. For example, it can be formulated as optimization problem through the essential relationship between LSI and Principal Component Analysis (PCA). It can also be formulated as the best low rank matrix approximation problem. Intuitively, LSI is computed through the matrix decomposition. Given the term by document matrix S, where s_{ji} stands for the weight of term j in document i. Assume that there exists a decomposition of matrix $S = U \Sigma V^T$, where U and V are orthogonal matrices and S is a diagonal matrix. This matrix decomposition is called as the Singular Value Decomposition (SVD). If preserve only the first p columns of U, it is the projection matrix $W \in R^{m \times p}$ for LSI. For details for LSI please refer to [3].

Key Applications

The text representation is the fundamental work for IR and text mining. Besides IR, it can also be used for text categorization, text clustering, topic detection and novelty detection etc.

Future Directions

The future directions of text representation problem can be roughly classified into two categories. The first is the large scale computation and the second is the semantic text representation. For the former, the goal is to make the text representation strategies which have high cost to be usable in real IR tasks. For example, one can develop the distributed infrastructure for N-gram model which can be used to enrich the BOW model if the computation of N-gram model is efficient enough. One can develop the incremental approximation or distributed computation algorithms for large scale LSI which can discover the semantic of documents in large scale IR systems. For the latter, besides LSI and PLSI, some semantic Web related research works give good information sources about how to discover the semantics of text documents.

Data Sets

For testing the effectiveness of text representation strategies, there are many commonly used text datasets in different scales. As some examples, the Reuters-21578 (http://www.daviddlewis.com/resources/testcollections/reuters21578/), RCV1 (http://jmlr.csail.mit.edu/papers/volume5/lewis04a/lewis04a.pdf), 20 Newsgroup (http://people.csail.mit.edu/jrennie/20Newsgroups/), Open Di rectory Project (ODP) (http://rdf.dmoz.org/) and the TREC text datasets (http://trec.nist.gov/data.html) are all commonly used for text mining or IR research.

Cross-references

- ▶ Stemming Algorithms
- ▶ Term Statistics
- ▶ Term Weighting
- ▶ Text Analytics

- ► Text Indexing & Retrieval
- ► Text Indexing Techniques
- ► Text Normalization
- ► Text Representation
- ► Text Semantic Explanation
- ► Tfxidf

Recommended Reading

1. Alter O., Brown PO., and Botstein D. Singular value decomposition for genome-wide expression data processing and modeling. In Proc. Natl. Acad. Sci. USA., 97:10101–10106.
2. Daniel J. and James H.M. Speech and Language Processing: An introduction to Natural Language Processing, Computational Linguistics, and Speech Processing. Prentice-Hall, Englewood Cliffs, NJ, 2000.
3. Deerwester S., Dumais S.T., Landauer T.K., Furnas G.W., and Harshman R.A. Indexing by latent semantic analysis. J. Soc. Inf. Sci., 41(6):391–407.
4. Gerard S.A. Theory of Indexing. Society for Industrial Mathematics, Philadelphia, PA, 1987.
5. Gerard S. and Michael J. Introduction to Modern Information Retrieval. McGraw-Hill, New York, 1983.
6. Thomas H. Probabilistic latent semantic indexing. In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 50–57.

Text Retrieval

► Information Retrieval

Text Segmentation

HAODA HUANG, BENYU ZHANG
Microsoft Research Asia, Beijing, China

Synonyms

Document segmentation

Definition

Text segmentation is a precursor to text retrieval, automatic summarization, information retrieval (IR); language modeling (LM) and natural language processing (NLP). In written texts, text segmentation is the process of identifying the boundaries between words, phrases, or some other linguistic meaningful units, such as sentences or topics. The term separated from such processing is useful to help humans reading texts, and are mainly used to assist computers to do some artificial processes as fundamental units, such as NLP, and IR.

Historical Background

Natural language processing (NLP) is an important research field. Its primary problem is how to segment text correctly. Various segmentation methods have emerged in the past decades for different kinds of language and applications. Text segmentation is language dependent (different language has its own special problems, which would be introduced later), corpus dependent, character-set dependent, and application dependent. Most existing text segmentation systems are language specific and corpus dependent.

This entry mainly focuses on traditional English and some Chinese text segmentation strategies. To do such segmentations, there are two major approaches: (i) Manually analyzing the characters of text to get some heuristic approaches; (ii) Doing annotations for the sample corpus with boundary information, then adopting some machine learning (ML) methods to learn from annotated corpus, and finally doing automatic text segmentation.

The problem of word segmentation (also known as tokenization), which is the process of dividing the sequence of characters into words by locating words boundaries, does not appear to be difficult for written languages that have explicit word boundary markers, such as English in which words are separated by white spaces. When such clues are not consistently retained in written languages, such as Chinese, in which sentences can be considered to be a character string, doing word segmentation should be the most important and essential part of text segmentation. On the contrary, sentence segmentation (which also can be referred to as sentence boundary detection, sentence boundary disambiguation (SBD), or sentence boundary recognition) is more difficult in English text segmentation, which must disambiguate punctuations that might denote sentence boundaries.

Based on previous analysis, for English text segmentation, words can be isolated by white spaces. So, the main problem of this kind of segmentation is how to recognize the boundaries of sentences, which involves resolving the use of ambiguous punctuation, such as periods, commas, and semicolons. Some recognizable tokens contain ambiguous punctuation, such as numbers (e.g., 1,236.78), dates (e.g., 02/05/96), acronyms (e.g., AT&T), and abbreviations (e.g., U.S.). Some example sentences containing ambiguous punctuations follow:

1. *Clairson International Corp. said it expects to report a net loss for its second quarter ended March 26 and doesn't expect to meet analysts' profit estimates of $3.0 to $4 million, or 1,276 cents a share to 1,279 cents a share, for its year ending Sept. 24.* (From the Wall Street Journal (1988))
2. *The contemporary viewer may simply ogle the vast wooded vistas rising up from Saguenay River and Lac St. Jean, standing in for the St. Lawrence River.* (From the Wall Street Journal (1991))
3. *The firm said it plans to sublease its current headquarters at 55 Water St. A spokesman declined to elaborate.* (From the Wall Street Journal (1987))

As can be seen, periods have been used three different ways in the first example - within a decimal *($3.0)*, in abbreviations (*Corp.* and *Sept.*), and at the end of the sentence. A comma is used in a number (*1,276 and 1,279*). Consider the second and third examples in which "*St.*" appears three times. The first two instances of "*St.*" do not denote the boundary of the sentence, whereas the last one delimits the sentence. At the same time, it is also essential to do word segmentation in English text segmentation. It is necessary to recognize whether a period within a number *($3.0)* should be a part of decimal or the end of a sentence – that is, should *$3.0* be parsed as a word "*/$3.0/*" or be parsed as "*/$3/. /0/*". The comma used in the number (*1,276 and 1,279*) also should be recognized as a part of number rather than a break of a sentence – that is, whether "*1,276*" should be broken to "*/1,276/*" rather than "*/1/, /276/*". Taking the third and first example sentences above, the first two "*St.*" should be segmented to "*/St./*". Though the last "*St.*" is also supposed to be segmented to "*/St./*", one must always realize that this period also stands for the end of sentence.

If there were a dictionary containing all kinds of words and including all the acronyms and abbreviations, it would be much easier to recognize words and do tokenization. In actuality, however, acronyms and abbreviations are emerging continuously and will never stop. In order to solve this problem, many approaches have been proposed. There are mainly two approaches: First, manual analysis often uses regular expression grammars and lexicon to solve those problems. The systems based on such methods are always called rule-based systems. In such systems, Lexicon is used to obtain the information of words, such as capitalization, spelling, and suffixes, to determine whether a word followed with a period should be a common word or a boundary signal of a sentence. Regular expressions are established for detecting special kinds of words, such as numbers and dates. Another way of performing sentence boundary disambiguation (SBD) is to use machine learning (ML) techniques, such as decision tree classifier, maximum entropy modeling, and neural network. The most essential information for Machine Learning systems to obtain are good features, such as word spelling, suffix, capitalization, and word classes. The difficulty is how to obtain a well annotated sample corpus with labels of word boundaries and sentence boundaries.

For Chinese text segmentation, sentences are always isolated by non-ambiguous punctuations, but words are character strings without explicit boundary markers. So, the main problem of Chinese text segmentation is how to recognize the boundaries of words. All the same, Word Sense Disambiguation is the primary issue in word segmentation. Three types of ambiguity string are present in Chinese text segmentation: overlapping ambiguity string, combinatorial ambiguity string, and hybrid ambiguity string. To solve those problems, various approaches have been proposed. The most typical solution is using forwards maximum matching (FMM) and backwards maximum matching (BMM) based on a dictionary.

Except for these two typical languages, there are still other types of languages with many different features of written text. Many more difficulties for each language have appeared.

In addition to language-dependent word segmentation and sentence segmentation, there is another segmentation that has emerged: topic segmentation, since a document always contains several topics and even one topic may have different aspects. Topic segmentation is much harder than word and sentence segmentation due to difficulties in detecting semantic units and dividing them into topics. Many approaches have been tried to deal with such problems [1,4]. Applications based on topic segmentation are abundant, such as document summarization, text processing, and NLP.

Foundations

Using regular expression grammars and lexicon to solve ambiguous punctuation is classic and to some extend useful. Different regular expressions have been defined for all kinds of tokens containing ambiguous punctuation. First, lexicon is used to find if a token exists in the dictionary or not. If it does not exist, a regular expression is used.

1. Numbers. Take "123,456.789%" as an example. The regular expression should be ((0–9]+,)*[0–9]+(.(0–9]+)*%? (ab means matching a, and then matching b.
   a + means one or more a would be matched.
   a* means zero or more a would be matched.
   a? means zero or one a would be matched.). Once this expression is used to recognize numbers, periods, commas and percent within a number, those ambiguous punctuations will not be parsed falsely as a boundary of a sentence.
2. Dates, such as "05/11/95". Heuristic regular expression would be [0–9][0–9]/[0–9][0–9]/[0–9][0–9].
3. Acronyms and Abbreviations, typically "U.S., St., Ash., and Corp." Regular Expression [A–Z]+[A–Za-z0–9]*.([A–Za–z0–9].)* will be used to find those acronyms and abbreviations. It should be noted that when an acronym or abbreviation is located at the end of a sentence, after the expression mentioned above has been used to parse such kind of sentences, an error would occur. For example, the sentence "*The firm said it plans to sublease its current headquarters at 55 Water St. A spokesman declined to elaborate.*" would be parsed to one sentence, although it should actually be split into two sentences "*/The firm said it plans to sublease its current headquarters at 55 Water St./*" and "*/A spokesman declined to elaborate./*".

Three typical kinds of tokens have been presented above. There are still many other types of words containing ambiguous punctuations. Many more regular expressions are required to deal with all of them. The set of expressions would be extremely huge. It is important to mention that these typical kinds of tokens are language dependent. This implies that different regular expression sets for each language would need to be established, which is a labor consuming enterprise. Also, the expressions are usually corpus related, which makes those rules unable to be ported across domains.

Because there are so many limitations in using regular expressions to segment words and sentences, as mentioned above, machine learning (ML) techniques have been taken into this field. These techniques can be retrained quickly for a new domain, a new corpus, a new language only if a well-annotated sample corpus has been given. Some systems also use part-of-speech (POS) information to improve performance [3].

In order to discuss how to deal with Chinese text segmentation, first, three kinds of ambiguity string should be introduced. (i) Overlapping Ambiguity String. It is defined as follows: ABC can be segmented into A/BC and AB/C, such as "和/平等" could be segmented into"和/平等" and "和平等" in different situations. In sentence "独立/自主/和/平等/独立/的/原则" and "讨论/战争/与/和平/等/问题", the segmentation of "和平等" is different. (ii) Combinatorial Ambiguity String. It is defined as follows: AB can be segmented into AB and A/B, such as "马上" should be segmented into "马上" in sentence "马上/过来" and should be "马/上" in sentence "他/骑/在/马/上". (iii) Hybrid Ambiguity String. It combines overlapping and combinatorial ambiguity. To figure out the first kind of ambiguity string problem, forwards maximum matching (FMM) and backwards maximum matching (BMM) are used to segment sentence separately. Such as sentence "独立/自主/和/平等/独立/的/原则", with FMM, it should be segmented into "独立/自主/和平/等/独立/的/原则"; with BMM, it should be "独立/自主/和/平等/独立/的/原则". When the result of segmentation with FMM and BMM is different, syntax, semantic, pragmatic information will be added to determine which segmentation should be chosen. To solve the combinatorial ambiguity string problem, FMM and Backwards Minimum Matching are adopted. For example with FMM, the sentence "他骑在马上" should be segmented into "他/骑/在/马上"; with BMM, it should be "他/骑/在/马/上". The following processing is the same as overlapping ambiguity string segmentation.

Key Applications

Text segmentation is an important aspect in developing text processing applications, such as Information Extraction, Document Summarization, Machine Translation (MT), Natural Language Processing, Information Retrieval, Language Modeling, and Speech Recognition.

Future Directions

There are roughly two directions for text segmentation in the future. One is text segmentation based on specific language. Another is its use in other fields, such as the rich transcription field. For the former, there are more than 200 languages in the world. Each has its own difficulties in doing text segmentation, such as word segmentation in Chinese text segmentation. These problems are do not have ideal solutions. For the latter, sentence boundary disambiguation (SBD) has attracted increased attention recently as a way to improve speech recognition output for better readability and downstream natural language processing and some subsequent tasks, such as speech translation and speech summarization.

Data Sets

Two main data sets are typically used for testing, evaluation and development in large amount of text segmentation and text processing tasks: Brown Corpus and the Wall Street Journal (WSJ) corpus – containing the Penn Treebank (Marcus, Marcinkiewicz, and Santorini, 1993). Texts in those corpora are all split into documents, paragraphs, and sentences and are annotated with POS information. The above information is necessary to develop and evaluate text segmentation systems.

Other data sets on Rich Transcription can be used to perform sentence boundary disambiguation, such as the Rich Transcription data sets provided by National Institute of Standards and Technology (NIST).

Cross-references

- ▶ Column Segmentation
- ▶ Text Retrieval
- ▶ Text Summarization
- ▶ Text Representation

Recommended Reading

1. Beeferman D., Berger A., and Lafferty J. Statistical models for text segmentation. Mach. Learn., 34(1–3):177–210, 1999.
2. Grefenstette G. and Tapanainen P. What is a word, what is a sentence? Problems of tokenization. In Proc. 3rd Conf. on Computational Lexicography and Text Research, 1994, pp. 7–10.
3. Mikheev A. Tagging sentence boundaries. In Proc. 1st Conf. on North American Chapter of the Association for Computational Linguistics, 2000, pp. 264–271.
4. Reynar J.C. and Marcus M.P. Topic segmentation: algorithms and applications. Ph.D. Thesis, University of Pennsylvania, Philadelphia, PA, 1998.

Text Semantic Representation

JUN YAN, JIAN HU
Microsoft Research Asia, Haidian, China

Definition

The classical text representation strategies aim to numerically represent the unstructured text documents to make them mathematically computable. With the rapid growth of information retrieval and text data mining research, the semantic text representation is attracting more and more attention. The problem is how to represent the text documents by explicit or implicit semantics instead of word occurrence in the document. The goals of semantic text representation are to improve the text clustering, classification, information retrieval and other text mining problems' performance.

Historical Background

In the past decades, semantic text representation has attracted much attention in the area of information retrieval and text data mining research. There have different ways for categorizing various semantic text representation strategies. This entry generally classifies the previous efforts for this problem into two categories: explicit semantic text representation and implicit semantic text representation.

The explicit semantic text representation aims to represent text documents by explicit readable sentences, key phrases or keywords, which can semantically describe the main topic of the given text documents. The related approaches can be further classified into automatic approaches and manual approaches. From the automatic approaches' perspective, the Text Summarization technologies aim at learning one or more sentences to represent a given text document; the Information Extraction technologies aim at extracting one or more key phrases for describing the semantic of a given text document. In addition, many previous works propose to annotate the Web pages, which is a special type of text documents, by search engine click through logs. With the rapid growth of Semantic Web (http://infomesh.net/2001/swintro/#furtherReading) in recent years, the automatic web page annotation and ontology learning, etc can all be utilized for automatically represent Web pages semantically. As some examples, the WordNet and Wikipedia have both been utilized for semantically enhance the text representation for various applications. From the manual approaches' perspective, there are many commercial systems target at semantic text representation by key phrases or key words. Two of the most representative examples of these commercial systems are delicious and flicker. Both of them aim to semantically represent the Web pages through manually assigned social annotation to a large number of Web pages.

To date, the work on integrating semantic background knowledge into text representation is quite few and the results are not good enough. Buenaga Rodriguez et al. and Urena Loez et al. successfully integrated the WordNet resource for a document categorization task. They improved classification results of Rocchio and Widrow-Hoff algorithms on Reuters corpus. In contrast some work utilized WordNet in a supervised scenario without employing WordNet relations such as hypernyms and associative relations. Meanwhile, they built the term vectors manually. Dave et al. has utilized WordNet synsets as features for document representation and subsequent clustering. That work did not perform word sense disambiguation and found that WordNet synsets decreased clustering performance in the experiments. Hotho et al. integrated WordNet knowledge into text clustering, and investigated word sense disambiguation strategies and feature weighting schema through considering the hypernym relations from WordNet. The experimental results on Reuters corpus show improvements compared with the best baseline. However, considering the few word usage contexts provided by WordNet, the word sense disambiguation effect is quite limited. Meanwhile, the enrichment strategy which appends or replaces document terms with their hypernym and synonym is overly simple. To solve the limitations of many previous work, Hu et al. proposed to enhance the text representation by Wikipedia.

On the other side, one of the most classical implicit semantic text representation approaches is known as the Latent Semantic Indexing (LSI) [2]. Nowadays, the most commonly used text representation model is called as the Vector Space Model (VSM) [3]. It aims to represent each text document by a numerical vector such that the similarity between vectors (documents) can be computed by different kernels. Latent Semantic Indexing (LSI) was originally proposed for dealing with the problem of *synonymy* and *polysemy* in the vector space model. In VSM, LSI represents the semantics of text documents through the linear combination of terms, which is computed by the Singular Value Decomposition (SVD) [1]. The reason why LSI is known as the implicit semantic text representation strategy is that the linear combination of keywords, which are semantics of the text documents, cannot be explained intuitively. A variety of tests and applications have been developed to validate its power in text representation. Special interests have been paid to investigate its ability in improving the IR performance. Besides general IR tasks, LSI has also been successfully applied for the cross-language retrieval and distributed information retrieval tasks. However, classical LSI suffers from the high computational cost involved in the Singular Value Decomposition (SVD), especially when applied to large scale text corpus. To avoid the costly computation, it has been proposed to use other strategies such as Semi-Discrete matrix Decomposition (SDD) and Concept Indexing (CI) instead of LSI for implicitly and semantically representing text documents. There are many variances of the classical LSI, the Probabilistic Latent Semantic Indexing (PLSI) [4] and the Supervised LSI are some of the examples.

Foundations

This entry mainly introduces one classical algorithm in each algorithm category. In the explicit semantic text representation category, the Wikipedia for text enrichment is introduced. On the other hand, in the implicit semantic text representation category, the traditional latent semantic indexing is introduced.

Wikipedia is a dynamic and fast growing resource – articles about newsworthy events are often added within few days of their occurrence. Each article in Wikipedia describes a single topic; its title is a succinct, well-formed phrase that resembles a term in a conventional thesaurus. Meanwhile, each article must belong to at least one category of Wikipedia. Hyperlinks between articles keep many of the same semantic relations as defined in international standard for thesauri, such as equivalence relation (synonymy), hierarchical relation (hypernym) and associative relation. However, as an open resource, it inevitable includes much noise. To make it a clean and easy-to-use as a thesaurus, a recent work proposed by Hu et al. first preprocess the Wikipedia data to collect Wikipedia concepts, and then explicitly derive relationships between Wikipedia based on the structural knowledge of Wikipedia.

Each title of a Wikipedia article describes a topic, and they are denoted as a concept. After process the Synonymy, Polysemy and Hypernymy, since each Wikipedia article contains a lot of hyperlinks, which express relatedness between them, The cosine similarity of article pairs in Wikipedia may reflect the relatedness between the two concepts. However the drawback of this measurement is the same as that of *BOW* approach, since it only considers terms appeared in text documents which have no semantic information. Another method to measure the relatedness between a pair of Wikipedia articles is to compare the similarity between outlinked categories of the two articles. It can be observed that if two articles share some out-linked categories, the concepts described in these two articles are most likely related. To get an overall relatedness of two Wikipedia concepts, the above two measures are linearly combined.

As introduced above, to represent text documents semantically, many previous approaches enriched text representation with external resources such as WordNet and ODP. The same to them, for the Wikipedia, first, the algorithm generates new features for each document in the dataset. The features can be synonym or hypernym for document terms or expanded features for terms, sentences and documents. Second, the generated new features replace or append to original document representation and construct new vector representation.

On the other hand, the classical LSI due to its effectiveness in the area of text data mining and information retrieval research. More details about other related approaches please refer to the recommended readings. In the bag of words (BOW) model, a text document is represented as a collection of unordered terms. Given the document collection $D = \{d_i,\ i = 1, 2,...,n\}$, suppose there are m unique terms appeared in this collection. Mathematically, this corpus of documents can be represented by a m by n matrix. $S \in R^{m\times n}$ Each text document is denoted by a column vector $s_i, i = 1, 2,...,n$ and each term is denoted by a row vector. The j^{th} entry of s_i is denoted by $s_{ji}, j = 1, 2,...,m$.

Mathematically, LSI aims to find a projection matrix $W \in R^{m\times p}$ such that the linear projection $y_i = W^T s_i \in R^p, i = 1, 2,...,n$ can reflect the p semantic concepts implied by document d_i, where $p << m$. In other words, LSI assumes that the linear combination of terms can reflect the concepts in text documents. $W \in R^{m\times p}$ can give p different linear combinations for term weights vector s_i. Through the linear projection matrix W, the text document d_i which is represented by m terms?' weights in vector s_i is represented by p concepts?' weights in vector $y_{i.}$ The problem left is how to get the projection matrix W, i.e., how to get the weights for the linear combination of terms, from the matrix S. The calculating of W can be formulated from different perspectives. For example, it can be formulated as optimization problem through the essential relationship between LSI and Principal Component Analysis (PCA). It can also be formulated as the best low rank matrix approximation problem. Intuitively, LSI is computed through the matrix decomposition. Given the term by document matrix S, where s_{ji} stands for the weight of term j in document i. Assume that there exists a decomposition of matrix $S = U\Sigma V^T$, where U and V are orthogonal matrices and Σ is a diagonal matrix. This matrix decomposition is called as the Singular Value Decomposition (SVD). If preserve only the first p columns of U, it is the projection matrix $W \in R^{m\times p}$ for LSI.

Key Applications

The semantic text representation has various different applications. Generally speaking, it is very important for the traditional text clustering, text categorization and information retrieval tasks. As some detailed examples, the LSI is generally used for relevance based information retrieval and text clustering. Documents summarization can be used for search results snippet generation. Key word extraction can be used as a component of the ontology learning of semantic Web. The social annotation is a fundamental work for semantic Web. As a summary, the semantic text representation is a fundamental problem for text mining and analysis.

Future Directions

The future directions of semantic text representation problem can be roughly classified into threefold. The first is how to develop novel semantic text representation approaches by designing novel algorithms and leveraging other meta-information. In more details, the Wordnet was first proposed to enhance the semantic text representation. Simultaneously, the search engine click-through log was proposed to enhance the text representation. Recently, the Wikipedia and social annotation were widely studied as the meta-information of text documents for semantic text representation. The remaining problem is whether other better data sources for text enrichment will be found. The second is the scalability issue for current strategies. For example, the SVD computation for LSI is highly expensive. A big problem is how to design scalable algorithm for LSI to deal with large scale data. The advanced algorithm could be approximated LSI, parallel computation and incremental computation etc. The third direction is how to apply the semantic text representation strategies for pushing the progress of semantic Web and other applications of semantic text representation. For example, one problem is how to enhance the text clustering or classification by semantic text representation.

Data Sets

For testing the effectiveness of semantic text representation strategies, there are many commonly used text datasets in different scales. As some examples, the Reuters-21578 (http://www.daviddlewis.com/resources/testcollections/reuters21578/), RCV1 (http://jmlr.csail.mit.edu/papers/volume5/lewis04a/lewis04a.pdf), 20 Newsgroup (http://people.csail.mit.edu/jrennie/20News, groups/), Open Directory Project (ODP) (http://rdf.dmoz.org/) and the TREC text datasets (http://trec.nist.gov/data.html) are all commonly used for text mining or IR research.

Cross-references

- ▶ Semantic Web
- ▶ Text Categorization
- ▶ Text Retrieval

Recommended Reading

1. Alter O., Brown PO., and Botstein D. Singular value decomposition for genome-wide expression data processing and modeling. In Proc. Natl. Acad. Sci. USA., 97:10101–10106.
2. Deerwester S., Dumais S.T., Landauer T.K., Furnas G.W., and Harshman R.A. Indexing by latent semantic analysis. J. Soc. Inf. Sci., 41(6):391–407.
3. Gerard S. and Michael J. Introduction to Modern Information Retrieval. McGraw-Hill Companies, 1983.
4. Thomas H. Probabilistic latent semantic indexing. In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 50–57.

Text Streaming Model

Ning Liu
Microsoft Research Asia, Beijing, China

Definition

Text streaming model (TSM) is one of the fundamental problems in streaming model and text mining. It aims to process a sequence of text data that comes at a rate. In this model, text data does not take the form of arriving in multiple, continuous, rapid, time-varying data streams. Input text streaming D_1, D_2, ... arrives sequentially, one by one, and the aim of TSM is to analysis the streaming text data.

Historical Background

Mining the streaming data has attracted much attention of researchers in different areas due to its great industrial and commercial application potentials. Specifically, text streaming data models are of interest to the machine learning and data mining community. The world wide web has many text data, such as web pages, news-feeds, emails and blogs. And most of them are classical text streaming data. Then, how to model text streaming data is important.

Traditional text data model is Vector Space Model (VSM). One of the commonly used VSM is the Bag of Words model (BOW), which index the document as a set of terms. This set of terms defines a space such that each distinct term represents the entries in that space. Since VSM represents the documents as a set of terms, this space can be viewed as a "document space." A numeric weight can then be assigned to each term in a given document, representing an estimate of the usefulness of the given term as a descriptor of the given documents. The weights assigned to the terms in a given documents can then be interpreted as the coordinates of the document in the documents space. Then, Term Frequency Inversed Document Frequency (TFIDF) model was proposed. It uses real values, which capture the term distribution among documents to weight terms in each document vector. Moreover, the N-gram statistical language model was proposed to model text data with VSM.

However, there are two challenges in the traditional VSM for text streaming data.

1. Update the data automatically
2. Do analyses of updated streaming data in acceptable time

Foundations

In the bag of words (BOW) model, a text document is represented as a collection of unordered terms. Given the document collection $D = \{d_i,\ i = 1,\ 2,...,n\}$, suppose there are m unique terms appeared in this collection (stop words removal and stemming are introduced later). Mathematically, this corpus of documents can be represented by a m by n matrix $S \in R^{m \times n}$. Each text document is denoted by a column vector $s_i, i = 1, 2,...,n$ and each term is denoted by a row vector. The j^{th} entry of s_i is denoted by $s_{ji}, j = 1, 2,...,m$. As an example, suppose the document collection D implies two documents,

– d_1: He investigates the text representation approaches.
– d_2: What is the meaning of text representation approach for text documents?

There is a list of 13 unique terms, which are,

► *"He, investigates, the, text, representation, approaches, What, is, meaning, of, approach, for, documents."*

Thus, one roughly represents the two documents by a 13 by 2 matrix. Consider the simple Boolean model first. In other words, if a term appears in a document, its corresponding weight is 1; otherwise, it is 0. The transform of the matrix for collection D is,

$$S_2^T = \begin{pmatrix} 1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,1 \end{pmatrix}$$

where the order of term index is the same as the term list given above, i.e., the first term is "He" and the last term is "documents."

For text streaming data, a new text is coming,

– d_3: Text mining is important in IR.

Now, the term list of $d_1$and d_2 should be changed. The problem is how the model streaming text data, which is how to update the indexing matrix.

$$S_s^T = \begin{bmatrix} S_2^T & 0 & 0 \\ 0 & s_{3i} & 0 \end{bmatrix}$$

Key Applications

The streaming text model is the fundamental work for IR and text mining. Besides IR, it can also be used for news categorization and RSS clustering etc.

Data Sets

For testing the effectiveness of text representation strategies, there are many commonly used text datasets in different scales. As some examples, the Reuters-21578, RCV1, 20 Newsgroup, Open Directory Project (ODP) and the TREC text datasets are all commonly used for streaming text model.

Recommended Reading

1. Abadi D., Carney D., Çetintemel U., Cherniack M., Convey C., Erwin C., Galvez E., Hatoun M., Maskey A., Rasin A., Singer A., Stonebraker M., Tatbul N., Xing Y., Yan R., and Zdonik S. Aurora: a data stream management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 666.
2. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and issues in data stream systems. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 1–16.

Text Summarization

DOU SHEN
Microsoft Corporation, Redmond, WA, USA

Synonyms

Document summarization

Definition

Text summarization is the process of distilling the most important information from a text to produce an abridged version for a particular task and user [9].

Historical Background

With more and more digitalized text being available, especially with the development of the Internet, people are being overwhelmed with data. How to help people effectively and efficiently capture the information from the data becomes extremely important. Many techniques have been proposed for this goal and text summarization is one of them.

Text summarization in some form has been in existence since the 1950s [8]. Two main influences have dominated the research in this area, as summarized by Mani in [10]. Work in library science, office automation, and information retrieval has resulted in a focus on methods for producing extracts from scientific papers, including the use of "shallow" linguistic analysis and the use of term statistics. The other influence has been research in artificial intelligence, which has explored "deeper" knowledge-based methods for condensing information. While there are a number of problems remaining to be solved, the field has seen quite a lot of progress, especially in the last decade, on extraction-based methods. This progress has been greatly accelerated by the rather spectacular advances in shallow natural language processing, and the use of machine learning methods which train summarization systems from text corpora consisting of source documents and their summaries. In the following sections, a brief introduction of different kinds of text summarization will be introduced, which is followed by an overview of the existing algorithms.

Foundations

Text summarization can be categorized along two different dimensions: abstract-based and extract-based. An extract-summary consists of sentences extracted from the document while an abstract-summary may employ words and phrases that do not appear in the original document but that are semantically meaningful [9]. The summarization task can also be categorized as either generic or query-oriented. A query-oriented summary presents the information that is most relevant to the given queries, while a generic summary gives an overall sense of the documents content [4]. In addition to single document summarization, which has been first studied in this field for years, researchers have started to work on multi-document summarization whose goal is to generate a summary from multiple documents that cover related information.

Most current text summarizers are extractive, since extraction is widely used, such as the snippets generated by search engines, while it is much easier than abstracting. Therefore, this entry focuses on extraction. For extractive summarizers (either for single-document summarization or multi-document summarization), they usually need to solve three problems: (i) Content selection, that is what should be selected from the text to form the summaries, which are most in the form of sentences or phrases; (ii) Information ordering, that is how to order the extracted sentences or phrases; (iii) Sentence realization, that is what kind of clean up to perform on the extracted sentences or phrases so they form a coherent summary. It is clear that the first problem is the critical one for extractive summarizers. The algorithms for this problem can be categorized as either unsupervised methods, or supervised methods. Unsupervised methods do not need any training data (pairs of a texts and the corresponding summaries). They calculate the importance of a sentence by considering the position of the sentence, the contained terms or phrases, the discourse centrality and so on. Supervised methods rely on a set of training data, which designs some features to capture the importance of sentences and a model can be learned from the training data to predict the importance of sentences. With this model, the most important sentences can be selected to form a summary.

Extractive Summarization Algorithms

Unsupervised Methods The simplest and straightforward way to select sentences is based on the sentences' locations. Generally speaking, certain locations of the text (titles, headings, the first sentence in each paragraph, etc.) tend to contain important information. Therefore, by simply taking sentences in these locations, a summary can be constructed, which forms a strong baseline, even better than other methods [2]. Besides locations of sentences, the importance of a sentence is indicated by cue phrases. For example, once a sentence contains the phrases like "To sum up," "In summary," it is more likely to be extracted as an summary sentence. On the contrary, some other phrases like "specifically," "in details" indicate that the sentence is not "abstract" enough to be a summary sentence. In [15], Teufel and Moens manually built a list of 1,423 cure phrases in a genre of scientific texts and each cue phrase has a (positive or negative) "goodness score," also assigned manually.

There are also some methods calculating the importance of sentences based term importance and the term importance can be estimated by its frequency. The system of Luhn [8] is an typical example of this kind. In Luhn's method, every sentence is assigned with a significance factor, and the sentences with the highest significance factor are selected to form the summary. In order to compute the significance factor of a sentence, it is necessary to build a "significant words pool" which is defined as those words whose frequency is between high-frequency cutoff and low-frequency cutoff that can be tuned to alter the characteristics of the summarization system. After this is done, the significant factor of a sentence can be computed in the following way: (i) set a limit L for the distance at which any two significant words could be considered as being significantly related. (ii) find out a portion in the sentence that is bracketed by significant words not more than L non-significant words apart. (iii) count the number of significant words contained in the portion and divide the square of this number by the total number of words within the portion.

In the above mentioned unsupervised methods, the sentences in a text are treated independently. In the following part, several more unsupervised methods are introduced, which exploit the relationship among sentences to some extent. The first one, as presented in [5], organizes the sentences in a text into a matrix and then apply a technique named Latent Semantic Analysis (LSA) [1] to derive the importance of sentences. A brief overview of LSA can make it easier to understand the LSA-based summarization method. LSA is based on singular value decomposition (SVD), a mathematical matrix decomposition technique that is applicable to text corpora as known by people. Given an m*n matrix $A = [A_1, A_2,...,A_n]$, with each column vector A_i representing the weighted term-frequency vector of sentence i in the document under consideration, the SVD is defined as:

$$A = U\Sigma V^T$$

where $U = [u_{ij}]$ is an m*n column-orthonormal matrix whose columns are called left singular vectors; $\Sigma = diag\ (\sigma_1,\sigma_1,...,\sigma_n)$ is an $n \times n$ diagonal matrix whose diagonal elements are non-negative singular values sorted in descending order. $V = [v_{ij}]$ is an $n \times n$ othonormal matrix whose columns are called right singular vectors.

As noted in [1], LSA is applicable in summarization for two reasons. First, LSA is capable of capturing and modeling interrelationships among terms by semantically clustering terms and sentences. Second, LSA can capture the salient and recurring word combination pattern in a document which describes a certain topic or concept. In LSA, concepts are represented by one of the singular vectors where the magnitude of the corresponding singular value indicates the importance of this pattern within the document. Any sentence containing this word combination pattern will be projected along this singular vector. The sentence that best represents this pattern will have the largest index value with this vector. Therefore, a summary can be constructed by collecting the sentences having largest index values over all concepts.

In [12], Mihalcea explicitly models the relationship among sentences by building a graph. In the graph, each node corresponds to a sentence and the weight of the edge linking two nodes is the similarity between the corresponding sentences. The direction of the edges can be decided by the appearance order of the sentences. After constructing the graph, Mihalcea employed some graph-based ranking algorithms like HITS and PageRank to decide the importance of a vertex (sentence) which can take into account the global information recursively computed from the entire graph. Finally, the sentences with highest ranking scores are selected to form a summary.

The third kind of summarization methods of exploiting sentence relationship is based on coherence relations. One example for the coherence relations is RST (rhetorical structure theory) relations. The RST relations are often expressed in terms of a satellite and a nucleus and nucleus sentences are more likely to be a summary sentence. More details of using RST for summarization can be found in [11].

Supervised Methods In the above mentioned unsupervised methods, each method estimates the importance of sentences based on some evidences from a certain aspect and then generate summaries based on the estimated importance. Therefore, a proper combination of these evidences may improve the generated summaries. Some supervised machine learning methods have been exploited for this goal. Among these methods, most of them treat the summarization task as a two-class classification problem at the sentence level, where the summary sentences are positive samples while the non-summary sentences are negative samples. After representing each sentence by a vector of features, a classification function such as Naive Bayes and Support Vector Machine can be trained [6]. Then for a new text, the trained classification function can be applied on each sentence in the text and decide whether it is a summary sentence. Although such methods are effective in most cases, they assume that the sentences are independent and classify each sentence individually without leveraging the relation among the sentences.

In order to address this shortcoming, some methods based on Hidden Markov Model (HMM) are exploited [3]. In Conroy et al.'s work [3], there are two kinds of states, where one kind corresponds to the summary states and the other corresponds to non-summary states. The observations are sentences that are represented by a vector of three features. Given the training data, the state-transition probabilities and the state-specific observation probabilities can be estimated by the Baum-Welch algorithm or an EM algorithm. Given a new document, the probability that a sentence corresponds to a summary state can be calculated. Finally, the trained model can be used to select the most likely summary sentences.

It is clear that such approaches can handle the positional dependence and feature dependence when the feature space is small by taking some special assumptions. However, the HMM based methods have two open problems. Firstly, when the feature space is large and the features are not independent or are even overlapping in appearance, the training process will become intractable. Therefore this approach cannot fully exploit the potential useful features for the summarization task due to the computational inefficiency. Secondly, the HMM based methods set the HMM parameters to maximize the likelihood of the observation sequence. By doing so, the approach fails to predict the sequence labels given the observation sequences in many situations because they inappropriately use a generative joint-model in order to solve a discriminative conditional problem when observations are given. In [14], the authors use Conditional Random Fields (CRF) to replace HMM for text summarization which avoids these problems.

Evaluation

For extractive summarization methods, there are two popular evaluation measurements. The first one is by Precision, Recall and F_1 which are widely used in Information Retrieval. For each document, the manually extracted sentences are considered as the reference summary (denoted by S_{ref}). This approach compares the candidate summary (denoted by S_{cand}) with the reference summary and computes the precision, recall and F_1 values as shown in the following equation:

$$p = \frac{|S_{ref} \bigcap S_{cand}|}{S_{cand}} \quad r = \frac{|S_{ref} \bigcap S_{cand}|}{S_{ref}} \quad F_1 = \frac{2pr}{p + r}$$

A second evaluation method is by the ROUGE toolkit, which is based on N-gram statistics [7]. This tool is adopted by DUC for automatic summarization evaluation that was found to highly correlate with human evaluations.

Key Applications

Text summarization has been applied in many fields. For example, the snippets generated by search engines such as Google, Live Search are successful examples. Another example is to generate summaries for hand-held devices, whose screens are usually small [10]. Besides these, text summarization has also been used as a preprocessing step for some text mining tasks such as Web-page classification, which is expected to catch the main content of the Web pages while removing the noises [13].

Data Sets

Document Understanding Conferences (2001–2007) provide an open data source for different kinds of summarization tasks: http://duc.nist.gov/. More useful URLs are at http://www.summarization.com/.

Cross-references

▶ Text Mining
▶ Text Generation
▶ Text Classification
▶ Summarization
▶ Topic Detection and Tracking

Recommended Reading

1. Berry M.W., Dumais S.T., and O'Brien G.W. Using linear algebra for intelligent information retrieval. SIAM Rev., 37(4): 573–595, 1995.
2. Brandowa R., Mitzeb K., and Rauc L.F. Automatic condensation of electronic publications by sentence selection. Inform. Process. Manage., 41(6):675–685, 1995.
3. Conroy J.M. and O'leary D.P. Text summarization via hidden markov models. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 406–407.
4. Goldstein J., Kantrowitz M., Mittal V., and Carbonell J. Summarizing text documents: sentence selection and evaluation metrics. In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, pp. 121–128.
5. Gong Y. and Liu X. Generic text summarization using relevance measure and latent semantic analysis. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 19–25.
6. Kupiec J., Pedersen J., and Chen F. A trainable document summarizer. In Proc. 18th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 68–73.
7. Lin C.-Y. and Hovy E. Automatic evaluation of summaries using n-gram co-occurrence statistics. In Proc. Human Lang. Tech. Conf. of the North American Chapter of Assoc. Comput. Linguistics, 2003, pp. 71–78.

8. Luhn H.P. The automatic creation of literature abstracts. IBM J. Res. Dev., 2(2), 1958.
9. Mani I. Advances in Automatic Text Summarization. MIT, Cambridge, MA, USA, 1999.
10. Mani I. Recent developments in text summarization. In Proc. 10th Int. Conf. on Information and Knowledge Management, 2001, pp. 529–531.
11. Marcu D. From discourse structures to text summaries. In Proc. ACL Workshop on Intelligent Scalable Text Summarization, 1997, pp. 82–88.
12. Mihalcea R. Language independent extractive summarization. In Proc. 20th National Conf. on AI and 17th Innovative Applications of AI Conf., 2005, pp. 1688–1689.
13. Shen D., Chen Z., Yang Q., Zeng H.-J., Zhang B., Lu Y., and Ma W.-Y. Web-page classification through summarization. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 242–249.
14. Shen D., Sun J.-T., Li H., Yang Q., and Chen Z. Document summarization using conditional random fields. In Proc. 20th Int. Joint Conf. on AI, 2007, pp. 2862–2867.
15. Teufel S. and Moens M. Sentence extraction as a classification task. In Proc. ACL Workshop on Intelligent Text Summarization. 1997, pp. 58–65.

Text Visualization

HAODA HUANG, BENYU ZHANG
Microsoft Research Asia, Beijing, China

Synonyms

Document visualization

Definition

Text visualization is a subarea of information visualization. The definition of information visualization is as follows: The use of computer-supported, interactive, visual representations of abstract data to amplify cognition [2]. Thus, the definition of text visualization is analogous: The use of computer-supported, interactive, visual representations of abstract text to amplify cognition. A more comprehensive and user-friendly definition is similar: The visual representation of text and its relationships.

Historical Background

In the early years, information such as texts and pictures were organized linearly: they were put in order and searched from beginning to end. But this is obviously not the most efficient way to organize information. It is well known that "A picture is worth a thousand words." People like to see news along with pictures, music, and even videos, rather than see purely raw text. Various technologies are also developed to help people graphically and visually represent their ideas, problems, challenges, solutions, and results. Today, information is not only presented one-dimensionally as in previous centuries, but also presented in two and more dimensions to help people understand the underlying idea clearly and thoroughly. Putting the right data in the right location and right format will greatly facilitate people seeking and understanding the information.

The history of text visualization is not very long. It started in the 1980s, when bandwidth and storage was very expensive. At that time, it was a luxury to play with advanced and real-time interactive graphics and visual effects. But with the rapid developments during the past several years, the standard PC platform provides high performance for processing videos and graphics and can present many advanced visual effects to users now. The 3D graphic interface has gradually dominated the gaming area, and 2D and 3D information visualization will become the mainstream of search engine retrieval and many other services.

Today, people are not satisfied with simply interacting with information in one dimension. Instead, they are more willing to search and manipulate their ideas and contents in multiple dimensions. This is becoming the main trend today and is also presenting a great challenge to traditional information organizations, such as libraries and museums. They are required to develop advanced 2D and 3D text visualization technologies to maintain their market share.

Foundations

Text visualization is about representing the underlying structure of a text or a group of texts. Text visualization offers several benefits for users. With text visualization, for example, users could have a view of texts in different levels of abstraction, could have a better view of the relationships between texts, and could have a high-level view of the topics in the text collection.

Generally, the ideas of many text visualization algorithms are borrowed from data analysis research areas while accounting for the specific properties of texts. These properties include the following [6,8]: "(i) High data dimensionality when using typical bag-of-words representation, where each word and each phrase represents on dimension in the data space. (ii) High redundancy, meaning that many dimensions can be easily merged into one dimension without losing much information. This is caused by the two properties of words, namely synonymy (different surface word forms having the same meaning – e.g., singer, vocalist) and hyponymy (one word denotes a subclass of an another – e.g., breakfast, is a subclass of a meal). (iii) Ambiguity between words in the cases where the same surface form of the word has different meanings (homonomy – e.g., the word "bank" can mean "river bank" or "financial institution") or in the cases where the same form has related meaning (polysemy – e.g., "bank" can mean "blood bank" or "financial institution"). (iv) Frequency of words (and phrases) follows power distribution. Appropriate weighting schemas (e.g., most popular being TFIDF) are used to normalize importance of the words to be able to work with the standard data analytic techniques."

Meanwhile, there are many types of texts, such as web documents, emails and news group postings, literature, and legal documents. In developing a text visualization algorithm or a system for a special type of text, the properties of the type of text also needs to be considered. For example, to visualize news articles, [8] have considered the following properties in their approach: (i) shorter documents; (ii) written by professionals; (iii) low number of mistakes; (iv) having good rhetorical structure; (v) rich information about people, companies, or phrase; and (vi) single documents as pieces of larger stories spanning over several documents.

Many text visualization algorithms adopt bag-of-words text representation, where text is viewed as a bin of independent words with term dependencies and with any other positional information of terms ignored. This simplification appears to be reasonable, since in many cases the efficiency of solving relevant problems does not degrade much. In the bag-of-words representation, each word is represented as a separate variable with a numeric weight. This numeric weight is often calculated using the famous TFIDF weighting schema: the weight is the multiply of term frequency and the inverse document frequency. The idea of TFIDF is very intuitive: if the word appears more times in a text, it would be more important; and if the word appears in fewer texts in the text corpus, it would be more important. In the bag-of-words representation, a text is represented as high-dimensional sparse vector, so it cannot be directly visualized. A clustering algorithm needs to be applied to indentify the relationships between texts and then to map them into 2D or 3D space. The cosine similarity measure is widely used to evaluate relationship between texts. There are also some other typical ways of text visualization using graphs or trees to present frequent co-occurrences of words and phrases. Typical visualization algorithms include graph based visualization and tiling based visualization as introduced in [4]:

Graph-based visualization has the following algorithm sketch: (i) First, documents are transformed into the bag-of-words sparse-vectors representation. Words in the vectors are weighted using TFIDF. (ii) Then, K-Means clustering algorithm is used to split the documents into K groups. Each group consists of similar documents and these are compared using cosine similarity. The K groups form a graph with groups corresponding to graph nodes and similar groups linked. Each group is represented by characteristic keywords. (iii) Finally, simulated annealing is used to draw a graph.

Tiling-based visualization has the following algorithm sketch: (i) First, documents are transformed into the bag-of-words sparse-vectors representation. Words in the vectors are weighted using TFIDF. (ii) Then, hierarchical top-down two-wise K-Means clustering algorithm is used to build a hierarchy of clusters. The hierarchy is an artificial equivalent of hierarchical subject index just like Yahoo. (iii) Finally, the leaf nodes of the hierarchy (bottom level) are used to visualize the documents. Each leaf is represented by characteristic keywords and each hierarchical splits the rectangular area into two sub-areas recursively.

Key Applications

WebSom [7] gives self-organizing Maps for Internet Exploration. An ordered map of the information space is provided: similar documents lie near each other on the map. The algorithm automatically organizes the documents onto a two-dimensional grid so that related documents appear close to each other.

ThemeScape [3] graphically displays images based on word similarities and themes in text. Themes within the document spaces appear on the computer screen as a relief map of natural terrain. The mountains indicate where themes are dominant, valleys indicate weak themes. Themes close in content will be close visually based on the many relationships within the text spaces. The algorithm is based on K-means clustering.

ThemeRiver [5] helps users identify time-related patterns, trends, and relationships across a large collection of documents. The themes in the collection are represented by a "river" that flows left to right through time. The theme currents narrow or widen to indicate changes in individual theme strength at any point in time.

The text representation is the fundamental work for IR and text mining. Besides IR, it can also be used for text categorization, text clustering, topic detection and novelty detection etc.

Future Directions

Although there have been many text visualization systems that have achieved much success in the past, there still exists many ways to improve them. One is to use natural language processing tools to do more detailed analysis or to improve the text summarization. Another may be to design more and better interaction tools for users to manipulate the texts. Advanced 2D and 3D graphics techniques may also be used to make the user interfaces friendlier.

Cross-references

- ► Data Mining
- ► Information Retrieval
- ► Text Segmentation

Recommended Reading

1. Baeza-Yates R. and Ribeiro-Neto B. Modern Information Retrieval. ACM Press, NewYork, NY, 1999.
2. Card S., Mackinlay J., and Shneiderman B. Readings in Information Visualization: Using Vision to Think. Academic Press, 1997.
3. Cartia. ThemeScape Product Suite. Available at: http://www.cartia.com/products/index.html
4. Grobelnik M. Text Visualization Tutorial.
5. Havre S., Hetzler E., Whitney P., and Nowell L. ThemeRiver: Visualizing thematic changes in large document collections. IEEE Trans. Vis. Comput. Graph., 8(1):9–20, 2002.
6. Jurafsky D. and Martin J.H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Prentice Hall, 2000.
7. Lagus K., Kaski S., and Kohonen T. Mining massive document collections by the WEBSOM method. Inf. Sci., 163 (1–3):135–156, 2004.
8. Marko G. and Dunja M. Visualization of news articles. In SIKDD 2004 at Multiconference IS. Ljubljana, Slovenia, 2004, pp. 12–15.

Text/Document Summarization

► Summarization

Text-based Image Retrieval

► Annotation-based Image Retrieval

TF*IDF

IBRAHIM ABU EL-KHAIR
Minia University, Minia, Egypt

Synonyms

Term frequency by inverse document frequency

Definition

A weighting function that depends on the term frequency (TF) in a given document calculated with its relative collection frequency (IDF). This weighting function is calculated as follows [1] Assuming that term *j* occurs in at least one document *d* (*dj* ≠ *0*), the inverse document frequency (idf) would be

$$\mathrm{Log}_2(\mathrm{N}/\mathrm{d}_j) + 1 = \log_2 \mathrm{N} - \log_2 \mathrm{d}_j$$

The ratio dj/N is the fraction of documents in the collection that contain the term. The term frequency-inverse document frequency weight (TF*IDF) of term *j* in document *i* is defined by multiplying the term frequency by the inverse document frequency:

$$\mathrm{W}_{ij} = f_{\mathrm{ij}}{}^*[\log_2 \mathrm{N} - \log_2 \mathrm{d}_j]$$

Where

N: number of documents in the collection

d_j: number of documents containing term *j*

f_{ij}: frequency of term *j* in document *i*

W_{ij}: is the weight of term *j* in document *i*

The use of the logarithm in the formula rather than the actual values of N and D_k moderates the effect of increasing the collection size and the effect of a high term frequency.

Key Points

The significance of a certain term in a given document is determined using the term frequency (TF) of that term in the document. Unfortunately, while term frequency is a good measure for term significance in one document, it is not an adequate measure for its significance in a collection of documents. The use of this factor alone in calculating the term weights in a collection of documents does not guarantee adequate retrieval performance. Some high frequency terms are not concentrated in a few particular documents, they are common in the whole collection which means that all these documents will be retrieved, affecting the performance of the information retrieval system.

The solution for this problem [4] is correlating the term frequency with its relative collection frequency (IDF), making the collection frequency a variable in retrieval. The use of collection frequency places a greater emphasis on the value of a term as a means of distinguishing one document from another than its value as an indication of the content of the document itself. Combining the two factors, TF and IDF enables the information retrieval system to exploit the good features of both. TF emphasizes term significance in a given document, and IDF emphasizes term significance in the collection as a whole; i.e., if the term was common in a document and rare in the collection it would be heavily weighted in both schemes. This way a term can distinguish certain documents from the remainder of the collection.

Even though this weighting function is a good indication of the term importance in a given set of documents it overlooks an important factor which is the document length. This problem may affect the weighting process because in real life documents have different lengths, and longer documents may have higher frequencies for a given term because it is repeated several times in the document. This increases the weight of the term and increases the possibility of retrieving these documents because of the higher weight of terms in them. Long documents also may have more different terms them which may affect the retrieval as well. More terms in a given document increases the possibility of matching between this document and multiple queries [3]. A possible way to overcome this problem is to incorporate a normalization factor for the document length to reduce its effect and make the weighting function more effective. A number of variant formulae for tf*idf weighting are given in [2].

Cross-references

- ▶ BM25
- ▶ Information Retrieval
- ▶ Term Weighting
- ▶ Text Indexing Techniques

Recommended Reading

1. Korfhage R.R. Information Storage and Retrieval. Wiley, New York, USA, 1997.
2. Manning C.D., Raghavan P., and Schütze H. Introduction to Information Retrieval. Cambridge University Press, Cambridge, UK, 2008.
3. Salton G. and Buckley C. Term-weighting approaches in automatic text retrieval. Inf. Process. Manage., 24(4):513–523, 1988.
4. Sparck J.K. A statistical interpretation of term specify and its application in retrieval. J. Doc., 28:11–20, 1972.

tgd

▶ Tuple-Generating Dependencies

Thematic Map

Hans Hinterberger
ETH Zurich, Zurich, Switzerland

Synonyms

Thematic map; Data map; Chart

Definition

The graphical representation of quantitative data or qualitative data for a given geographic area (e.g., regional unemployment rate or type of crops grown).

The many different methods that cartographers apply to create thematic maps are used to define a thematic map more precisely. The following five are found most often.

1. *Coropleth maps.* In coropleth maps the areas of the map are shaded or patterned in proportion

to the value of the data to be displayed for a particular area.

2. *Dot maps.* These maps use equal sized dots to show the presence of a feature at a particular geographic location and thus display spatial distributions. A dot need not be restricted to a single occurrence; it may indicate any number of entities.
3. *Proportional symbol maps.* When drawing proportional symbol maps, a cartographer selects a symbol (e.g., a circle or a bar), places it at the spot on the map to which the data apply and varies the symbol's size from place to place in proportion to the value of the variable that the symbol represents.
4. *Isarithmic maps.* These maps use contour lines to join points of equal value. Examples are barometric pressure lines in weather maps or elevation above sea level in topographic maps.
5. *Dasymetric maps.* These maps divide a geographic region with contour lines and shade or pattern the resulting regions in proportion to the value that applies to the region bounded by a contour line.

Topographic maps that provide reference to geographic features (political boundaries, roads, lakes, mountains) are generally not categorized as thematic maps.

Key Points

History. Edmund Halley (famous for discovering the comet bearing his name) is recognized as the author of the first thematic map. Drawn in 1686, it shows the direction of trade winds on a world map. Probably the best known example of using thematic maps for data analysis is John Snow's cholera map of 1855, a data display based on principles still applied in today's geographic information systems. These and more historical notes are summarized in [3]).

Usage. Thematic maps display the spatial distribution of data for a specific subject or a specific purpose. They represent information about particular locations in such a way that spatial patterns emerge. Frequently thematic maps are used to compare patterns on two or more maps. Examples are election results, occurrence of particular types of diseases, usage of agricultural land, climatic change over time and so on. Major contributions to thematic mapping come from the French cartographer Jacques Bertin ([1]). A more modern treatment of the topic can be found in [2].

Cross-references

- ▶ Data Visualization
- ▶ Chart

Recommended Reading

1. Bertin J. Graphics and Graphic Information-Processing. Walter de Gruyter, Berlin, New York, 1981.
2. Slocum T.A., McMaster R.B., Kessler F.C., and Howard H.H., Thematic Cartography and Geographic Visualization, 2nd edn. Pearson-Prentice Hall, Upper Saddle River, NJ, 2005.
3. Tufte E.R., The Visual Display of Quantitative Information. Graphics Press, Cheshire, CT, 1983.

Theme Algebra

▶ Spatial Operations and Map Operations

Thesauri Business Catalogues

▶ Lightweight Ontologies

Thiessen Polygons

▶ Voronoi Diagram

Third Normal Form

Marcelo Arenas
Pontifical Catholic University of Chile, Santiago, Chile

Definition

Let $R(A_1,...,A_n)$ be a relation schema and Σ a set of functional dependencies over $R(A_1,...,A_n)$. An attribute A_i ($i \in \{1,...,n\}$) is a *prime* attribute if A_i is an element of some key of $R(A_1,...,A_n)$. Then specification (R, Σ) is said to be in Third Normal Form (3NF) if for every nontrivial functional dependency $X \rightarrow A$ implied by Σ, it holds that X is a superkey for R or A is a prime attribute [2].

Key Points

In order to avoid update anomalies in database schemas containing functional dependencies, 3NF was introduced by Codd in [2]. This normal form is defined in terms of the notions of prime attribute and key as shown above. For example, given a relation schema $R(A, B, C)$ and a set of functional dependencies $\Sigma = \{AB \rightarrow C, C \rightarrow B\}$, it holds that $(R(A, B, C), \Sigma)$ is in 3NF since AB is a superkey and C is a prime attribute (given that AC is a key for R). On the other hand, $(S(A, B, C), \Gamma)$ is not in 3NF if $\Gamma = \{A \rightarrow B\}$, since A is not a superkey for S and B is not a prime attribute.

For every normal form two problems have to be addressed: how to decide whether a schema is in that normal form, and how to transform a schema into an equivalent one in that normal form. On the positive side, for every relation schema S there exists a database schema S' such that, S' is in 3NF and S' is a lossless and dependency preserving decomposition of S. Furthermore, schema S' can be generated efficiently by using the synthesis approach proposed in [1]. On the negative side, it is expensive to check whether a schema is in 3NF. It was shown by Jou and Fischer that this problem is NP-complete [3].

Cross-references

- ▶ Boyce-Codd Normal Form
- ▶ Fourth Normal Form
- ▶ Normal Forms and Normalization
- ▶ Second Normal Form (2NF)

Recommended Reading

1. Biskup J., Dayal U., and Bernstein P. Synthesizing independent database schemas. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 143–151.
2. Codd E.F. Further normalization of the data base relational model. In Proc. Data Base Systems. Prentice-Hall, Englewood Cliffs, NJ, USA, 1972, pp. 33–64.
3. Jou J. and Fischer P. The complexity of recognizing 3NF relation schemes. Inf. Process. Lett., 14(4):187–190, 1982.

Thread Lifecycle

▶ Process Life Cycle

Three-Dimensional GIS and Geological Applications

Martin Breunig
University of Osnabrueck, Osnabrueck, Germany

Synonyms

Spatial information system; Geoscientific information system

Definition

An information system for the in-/output, modeling, management, processing, analyzing and visualization of geoscientific data including geo-referenced three-dimensional geometric, topological and attribute data. The three-dimensional geometric data may consist of points/vertices (x,y,z-coordinates), curves, surfaces and polyhedra, respectively. The topological data may consist of nodes, edges, faces and solids, respectively. Typical attribute data are descriptions of geological strata, i.e., properties of strata such as "geological age," "soil type," "main components of the stratum" etc.

The implementation of a three-dimensional GIS provides data types, spatial access structures including geometric/topological algorithms and a spatial or visual query language for the modeling, management and analysis of geo-referenced three-dimensional data.

Historical Background

Three-dimensional GIS have two roots in the history of information systems. The first has its origin in the field of 3D modeling and visualization. 3D modeling and visualization systems usually provide a large collection of geometric and topological 3D algorithms. An example is the Discrete Smooth Interpolation (D.S.I.) algorithm of Jean-Laurent Mallet and others [12,13] used for creating and editing triangulated surfaces. Given an arbitrary mesh with an arbitrary set of vertices fixed by the user, D.S.I. assigns coordinates to the other nodes of the mesh, enabling the fixed vertices to be interpolated smoothly [12]. The second root has its origin in the field of *Geographical Information Systems* (GIS). GIS are inherently using both, spatial and attribute data, in their data management. However, standard Geographical Information Systems usually are not prepared to manage and process real 3D data. They only allow the visualization of 2.5D data representation such as digital elevation models. However, in this representation every (x,y)-coordinate may only have one z-coordinate. That is why Geographical Information Systems cannot be used to solve 3D geological problems. In particular, polyhedra and solids cannot be treated in Geographical Information Systems. In today's Geographical Information Systems, the third dimension is only treated as a thematic attribute such as the height value of an isoline map.

Relevant work in the field of three-dimensional GIS has been published by [2,5–7, 13,16,18,–20,22], and by other authors. For example, theoretical work on three-dimensional GIS and topological data models has been worked out by [15]. Literature about *spatial database systems* can be found in [8]. Spatial database systems can be extended for the management of three-dimensional GIS data. For example, an R-Tree can be used to access 3D data.

The term "three-dimensional GIS" (3D GIS) is not yet used in a standardized way. Unfortunately, the term "3D GIS" or "3D city model" is often used for information systems that only deal with surface and face data, but not with polyhedra and solids. Correctly, such information systems should be called "2.5D GIS," because they only allow to represent surfaces with the following property, valid for all of their points P(x, y, z): z = f (x, y). i.e., every point (x, y) of the surface has a different z-value.

First standardization efforts for the exchange of data between 3D GIS have been undertaken by the Open Geospatial Consortium [14] within the *Geography Markup Language* (GML).

Foundations

Spatial planning processes and the study of geoscientific processes often require three-dimensional information. This digital information must be modeled, managed, visualized and analyzed. A three-dimensional GIS is an information system that accomplishes these requirements. To work adequately, it needs 3D data types, 3D spatial access structures, and 3D geometric and topological algorithms to manage and process three-dimensional data of these applications. Furthermore, a user interface with a query language is required. A spatial database system may be embedded into a 3D GIS to support these requirements.

In a 3D GIS, objects of different dimension d ($0 \leq d \leq 3$) have to be processed. The 3D geometry of a geological object can be composed of sets of points, lines, surfaces and volumes, respectively. As a reference model for 3D geometric data types in three-dimensional GIS, *Simplicial Complexes* may be used: points, polylines, triangle nets and tetrahedron nets in three-dimensional space. They are often used in geological applications, because they well approximate 3D solids and surfaces formed by nature.

In a 3D topology model of a 3D GIS, the components of the objects are interpreted as a mesh of nodes, edges, faces, and solids that describes both the interior structure of the geological objects and their mutual neighborhood relationships in 3D space. As a reference model for 3D topological data types in three-dimensional GIS, cellular complexes [4,13] and Generalized Maps [10,11,13] respectively, may be used. They provide a general topological model treating 2D, 2.5D and 3D objects in a uniform way. Furthermore, they are based on the clear mathematical theory of algebraic topology.

To access 3D data, existing spatial access structures such as the R-Tree [9] or R*-Tree [3] may be applied. The spatial access structures can also be used internally within the processing of geometric 3D algorithms as a first filter step to compute on approximated geometries – 3D boxes are used most – to achieve better performance, for example to support the intersection of very large geometries such as sets of tetrahedron nets. For the analysis of 3D data, 3D GIS are using geometric and topological algorithms, e.g., for computing the distance between two objects, the intersection between two polyhedra, or the neighbor objects of a given geological object.

Three-dimensional GIS are subject of current research [1,21]. Concepts and prototypical software considering aspects of three-dimensional GIS such as architectural issues, spatial data modeling with constraints, efficient spatial data access and geological applications have been described in detail by many authors of the GIS and database communities (see Recommended Reading). First 3D GIS prototype systems are already used in some application domains of three-dimensional GIS such as geology, geophysics, archaeology, and early diagnosis of natural disasters. Also database vendors are integrating first simple 3D data types and operations in their products.

Key Applications

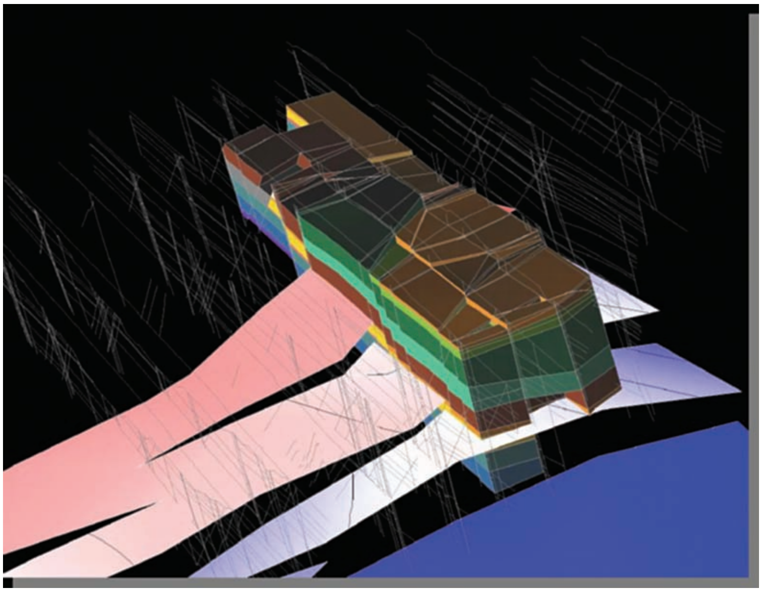
One of the most relevant applications of three-dimensional GIS is geology. In geology, the starting point of the examinations is the present condition of the earth's structure. Hence the three-dimensional geometric analysis of recent geological strata and solids is the key for further investigations concerning the interaction of former geological structures and processes [19]. Furthermore, consistent geometric 3D models and the use of GIS are the precondition for the production of digital geological maps. For example, the 3D models can be intersected with digital elevation models to improve maps or to define boundary conditions for 3D models. The computational expense of GIS, however, is high, because the third dimension is included.

Some of the relevant data in geological applications are sections, stratigraphic boundaries, and faults. Sections describe a mostly vertical intersection through the geological structure of an examination area, i.e., an estimation of the geological surfaces and solids including geological faults. A cross section through the series of sediments consists of a set of stratigraphic lines. Their geometry is mostly given by a point set in three-dimensional space.

The geometry of stratigraphic boundaries is based on stratigraphic lines in the sections where the stratum occurs. The stratigraphic surfaces are spread between the cross sections. Triangle nets in three-dimensional space can result from the triangulation of the surfaces between the point sets of the cross sections. Concerning topology, on each cross section the underlying and hanging stratum of each stratigraphic line can be identified.

Faults are modeled as surfaces, along which the tectonic deformation of the sediments took place. Usually the geometry of a fault consists of a triangle net in three-dimensional space. The modeling of strata and faults from cross section to cross section is an interpolation of new geometries from the sections.

Figure 1 shows an example of geological data managed by a three-dimensional GIS. During the modeling process of geologically defined geometries, a large amount of data are accumulated. Thus the handling of three-dimensional data in a spatial database system [8] is recommended for 3D GIS. for example, the efficient spatial access on a large set of geological objects is necessary to compute intersections between single strata and faults with different thematic attributes, respectively. Therefore, the implementation of efficient three-dimensional geometric algorithms has to be provided by the three-dimensional GIS. Checking the consistency of geometric



Three-Dimensional GIS and Geological Applications. Figure 1. Example of geological data in the Lower Rhine Basin, managed by a three-dimensional GIS (figure constructed in the group of Agemar Siehl, University of Bonn, with the GOCAD Software, Nancy, now distributed by Paradigm, UK).

3D models is essential for the geometric 3D reconstruction of geological structures and geological processes [19].

Cross-references

► Digital Elevation Models
► Geography Markup Language
► Simplicial Complex
► Spatial Network Databases

Recommended Reading

1. Abdul-Rahman A., Zlatanova S., and Coors V. (eds.). Innovations in 3D Geoinformation Systems, Lecture Notes in Geoinformation and Cartography, Springer, Heidelberg, 2006.
2. Balovnev O., Bode T., Breunig M., Cremers A.B., Müller W., Pogodaev G., Shumilov S., Siebeck J., Siehl A., and Thomsen A. The story of the GeoToolKit – an object-oriented geodatabase kernel system. Geoinformatica, 8(1):5–47, 2004.
3. Beckmann N., Kriegel H.-P., Schneider R., and Seeger B. The R*-tree: an efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 322–331.
4. Brisson E. Representing geometric structures in d dimensions: topology and order. In Proc. 5th Annual Symp. on Computational Geometry, 1989, pp. 218–227.
5. Coors V. and Zipf A (eds.). 3D-Geoinformationssysteme, Grundlagen und Anwendungen. Wichmann – Hüthig, Heidelberg, 2004.
6. GOCAD. http://www.gocad.org.
7. Götze H.J. and Lahmeyer B. Application of three-dimensional interactive modelling in gravity and magnetics. Geophysics, 53(8), 1988, pp. 1096–1108.
8. Güting R.H. An introduction to spatial database systems. VLDB J., 3(4):357–399, 1994.
9. Guttman A. R-Trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
10. Lienhardt P. Subdivision of n-dimensional spaces and n-dimensional generalized maps. In Proc. 5th Annual Symp. on Computational Geometry, 1989, pp. 228–236.
11. Lienhardt P. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. J. Comp. Geom. App., 4(3): 275–324, 1994.
12. Lévy B. and Mallet J.-L. Discrete Smooth Interpolation: Constrained Discrete Fairing for Arbitrary Meshes, ISA-GOCAD (Inria Lorraine/CNRS), ENSG, Vandoeuvre Nancy, http://www.earthdecision.com/news/white_papers/DSI.pdf.
13. Mallet J.L. Geomodelling. Oxford University Press, New York, NY, 2002.
14. OGC. http://www.opengeospatial.org.
15. Pigot S. A topological model for a 3D spatial information system. In Proc. 5th Int. Symp. on Spatial Data Handling, 1992, pp. 344–360.
16. Raper J. (Ed.). Three dimensional applications in geographical information systems. Taylor & Francis, London, 1989.
17. Samet H. The design and analysis of spatial data structures, Addison-Wesley, Reading, 1990.
18. Schaeben H., Apel M., v.d. Boogart G., Kroner U. GIS 2D, 3D, 4D, nD. Informatik-Spektrum, 26(3), 2003, pp. 173–179.
19. Siehl A. Construction of geological maps based on digital spatial models. Geol. Jb. A., 104:253–261, 1988.
20. Turner A.K. (ed.) Three-dimensional modeling with geoscientific information systems, Kluwer Academic, Dordrecht, 1991.
21. van Oosterom P., Zlatanova S., Penninga F., and Fendel E. (eds.) Advances in 3D geoinformation systems, Lecture Notes in Geoinformation and Cartography. Springer, Heidelberg, 2007.
22. Vinken R. Digital geoscientific maps – a research project of the DFG. In Proc. Int. Colloquium at Dinkelsbühl, Geolog. Jahrbuch A104, 1988, pp. 7–20.

Three-Dimensional Similarity Search

► Feature-Based 3D Object Retrieval

Three-Phase Commit

Yousef J. Al-Houmaily[1], George Samaras[2]
[1]Institute of Public Administration, Riyadh, Saudi Arabia
[2]University of Cyprus, Nicosia, Cyprus

Definition

Three-phase commit (3PC) is a synchronization protocol that ensures *global atomicity* of distributed transactions while alleviating the *blocking* aspect of 2PC (Two-Phase Commit) in the events of *site* failures. That is, 3PC never requires operational sites to wait (i.e., block) until a failed site has recovered.

Historical Background

3PC was one of the first attempts to resolve the blocking aspects of 2PC [6]. The main purpose of the protocol is to allow operational sites to continue transaction processing and reach agreement about the final status of transactions in spite of the presence of site failures. 3PC can tolerate any number of site failures (except for total sites' failures), assuming a highly reliable network (i.e., a network that never causes operational sites to be partitioned into more than one set of communicating sites, implying a network that never fails).

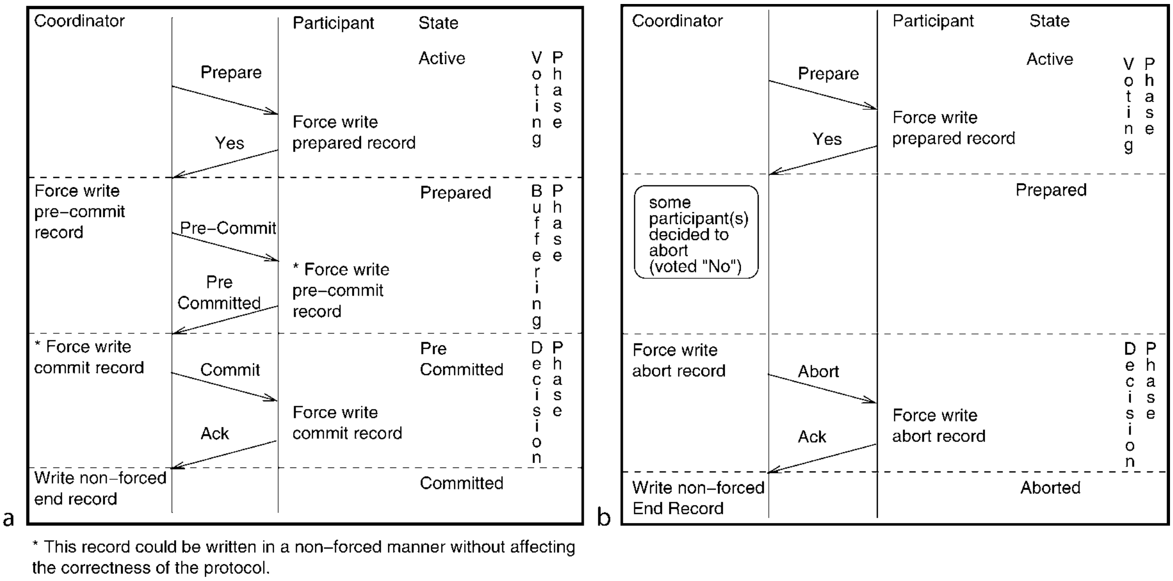
Foundations

In 2PC, a participant is *blocked* if it fails to communicate with the coordinator of a transaction while in a prepared-to-commit state. Blocking means that the participant cannot determine the final status of the transaction in the presence of a failure, rendering all resources held by the prepared-to-commit transaction at its site unusable by any other transaction until the final status of the transaction is resolved, i.e., the transaction is either committed or aborted. Blocking is inevitable in 2PC and it may occur either because of (i) a communication (link) failure or (ii) a coordinator's system crash. These two types of failures lead to blocking, in 2PC, even under the assumption that all participants remain operational and can communicate collaboratively to resolve the status of the prepared-to-commit transaction. For example, if a coordinator fails after all participating sites in a transaction's execution have entered their prepared-to-commit states; the participants (collectively) can neither commit nor abort the transaction. This is because the operational participants cannot be sure whether the coordinator had received all their votes and made a commit final decision just before it failed or it had received only some votes and did not have the chance to make the final decision before its failure and the transaction will be aborted by the coordinator when it recovers. Thus, the participants are blocked until the coordinator recovers.

The negative impact of blocking on the (i) overall system performance and (ii) availability of critical data on other transactions motivated the design of *non-blocking* atomic commit protocols (ACPs). In 3PC, an *extra (buffering) phase* is inserted between the two phases of 2PC to capture all the participants' intentions to commit, as shown in Fig. 1.

Dynamics of Three-Phase Commit

The basic idea behind the insertion of the buffering phase is to place the two (possible) reachable final states (i.e., the commit and the abort states) for a transaction apart from each other such that they cannot be reached from the *same* state. That is, if a final state can be reached from the current state of a site, then, the reachable final state can be either the commit or the abort state but not both. In 2PC, when a participant is in a prepared-to-commit state, both final states can be reached. Hence, if the coordinator fails, the participant cannot determine the final state for the transaction without any possible conflict in its decision with the coordinator. For this reason, the protocol is blocking. On the contrary and as shown in Fig. 1, the commit final state for a site (whether it is the coordinator or a participant), in 3PC, cannot be reached from the same


a
Coordinator
Participant
State
Active
Voting Phase
Prepare
Force write prepared record
Yes
Force write pre–commit record
Pre–Commit
Prepared
Buffering Phase
* Force write pre–commit record
Pre Committed
* Force write commit record
Commit
Pre Committed
Decision Phase
Force write commit record
Ack
Write non–forced end record
Committed
* This record could be written in a non–forced manner without affecting the correctness of the protocol.
b
Coordinator
Participant
State
Active
Voting Phase
Prepare
Force write prepared record
Yes
Prepared
some participant(s) decided to abort (voted "No")
Force write abort record
Abort
Decision Phase
Force write abort record
Ack
Write non–forced End Record
Aborted

Three-Phase Commit. Figure 1. The three-phase commit protocol.

state as the abort final state. In the former case, the commit state can be reached from the *pre-commit* state whereas, in the latter case, the abort state can be reached from the *prepared* state.

When a site is in a pre-commit state, it means that each of the other sites is at least in a prepared-to-commit state (Notice that the other sites might lag in their states because of system's delays such as queuing and network delays.). Thus, the pre-commit state is called a *committable state* since it implies that all participants have voted "yes" and the coordinator agreed on the commitment of the transaction. In 3PC, a *non-committable* state, i.e., a state that does not imply that all the participants have voted "yes," is not placed adjacent to a commit state. This is not the case in 2PC as the prepared state, which non-committable, is placed adjacent to a commit state.

The insertion of the buffering state makes the structure of 3PC to satisfy the two necessary and sufficient conditions for the construction of synchronous non-blocking ACPs within one state transaction, i.e., a structure where neither the coordinator nor any participant leads each other by more than one state transition during its execution of the protocol. That is, 3PC is synchronous within one state transaction that (i) does not contain a state that is adjacent to both a commit and an abort state, and (ii) it does not contain a non-committable state that is adjacent to a commit state.

Based on the above, if the coordinator of a transaction fails at any point during the execution of the protocol, the operational participants can collectively and deterministically decide the final status of the transaction. The decision is commit if any of the participants is in *at least* a pre-commit state (because it is not possible for the coordinator to have decided to abort). Otherwise, the decision is abort (because it could be possible for the coordinator to have decided to abort but not to commit). To reach an agreement on the final status of a transaction, there is a need for a termination protocol which is invoked when the coordinator fails.

Recovery in Three-Phase Commit

When a participant times out while waiting for a message from the coordinator, it means that the coordinator must have failed (or it is perceived as a coordinator failure). In this case, the participant initiates an election protocol to determine a *new* coordinator. One way to determine the new coordinator is based on sites' identification numbers such that the participant with the highest (or the lowest) number becomes the new coordinator. Once a new coordinator is elected, the participants exchange status information about the transaction. If the new coordinator finds the transaction in at least a pre-commit state at any participant, it commits (in its local log) the transaction; otherwise, it aborts the transaction. Then, this new coordinator proceeds to complete the 3PC for the transaction in all the other participants. If the new coordinator fails, the election process is repeated again.

When a participant starts recovering from a failure, it needs to determine the status of each prepared or pre-committed transaction as recorded in its log. Notice that a recovering participant cannot commit a transaction even if the participant is in a pre-commit state with respect to the transaction. This is because the operational sites might have decided to abort the transaction after the participant had failed if none of them was in a pre-commit state. In this case, the participant must ask the other sites about the final status of the transaction.

A final note on 3PC is about total sites' failure where there is a need to determine the last participant to have failed. This is because such participant is the only one which can decide the status of the transaction for the other participants. Determining the last participant that has failed could be implemented by maintaining an "UP" list at each participating site. This list contains the identities of operational participants as seen by the participant that is maintaining the list and is stored (in a non-forced or asynchronous manner) onto the stable log of the participant. Thus, the "UP" lists allow a set of participants to determine, upon their recovery from the total failure, whether they contain among themselves the last participant to have failed, reducing the number of participants that needs to recover before the transaction status can be resolved. Alternatively, all participants should recover and become operational again before the status of the transaction can be resolved.

Non-Blocking Commit Protocol Variants

As discussed above, blocking occurs in 2PC when the coordinator of a transaction crashes while the transaction is in its *prepared-to-commit* state at a participating site. In such a case, the participant is blocked until the coordinator of the transaction recovers. In general, all ACPs are susceptible to blocking. They just differ in the size of the window during which a site might be blocked and the type of failures that cause their blocking. Several ACPs have been designed to eliminate some of the blocking aspects of 2PC, besides 3PC, by adding extra coordination messages and forced log writes. These protocols can be classified into whether they preserve the prepared-to-commit state, such as cooperative 2PC, or allow *unilateral or heuristic* decisions in the presence of unbearable delays, such as IBM's presumed nothing (IBM-PrN).

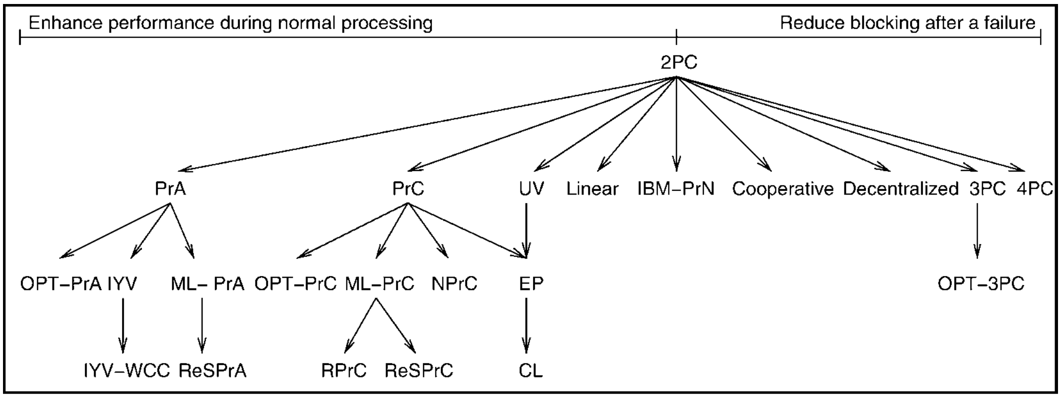
The *cooperative* 2PC (1981) reduces the *likelihood* of blocking in case of a coordinator's failure. In the cooperative 2PC, the identities of all participants are included in the prepare-to-commit message so that each participant becomes aware of the other participants. In the case of a coordinator's or a communication's failure, a participant does not block waiting until it reestablishes communication with the coordinator. Instead, it inquires the other operational participants in the transaction's execution about the final decision and if any of them has already received the final decision prior to the failure, it informs the inquiring participant accordingly.

The IBM-PrN (1990) is a 2PC variant that allows blocked participants to unilaterally commit or abort a transaction and detects atomicity violations due to conflicting heuristic decisions. In the event of atomicity violations, it reports any damage on transactions and data, simplifying the task of identifying problems that must be fixed. *Generalized presumed abort* (1994) is another IBM protocol that behaves like IBM-PrN when complete confidence in the final outcome and recognition of heuristic decisions is required and behaves like PrA during normal processing. Recent efforts to enhance commit protocols with heuristic decision processing resulted in the *allow-heuristics presumed nothing* (1996) commit protocol.

Other Atomic Commit Protocol Variants and Optimizations

Figure 2 shows some of the significant steps in the evolution of ACPs including the two most notable 2PC variants which are *presumed abort* (PrA) and *presumed commit* (PrC) [2]. The *new PrC* (NPrC) (1993) and *rooted PrC* (RPrC) (1997) protocols were proposed to reduce the log complexity of PrC further at the cost of slower recovery in the presence of failures. NPrC eliminates the initiation log record at the coordinator's site whereas RPrC eliminates the initiation log record at each cascaded coordinator when the *tree-of-processes* (or *multi-level transaction execution*) model is used.

In contrast to PrA and PrC variants, other 2PC variants have been proposed for specific environments. The common characteristic of these protocols is that they exploit the semantics of the communication networks, the database management systems and/or the transactions to enhance the performance of 2PC. For example, the *linear* 2PC (L2PC) reduces message complexity at the expense of time complexity compared to 2PC by assuming token-ring like networks. In L2PC, the participants are linearly ordered with the coordinator being the first in the linear order. The coordinator initiates the voting and each participant sends its "yes" vote to its successor in the linear order. The last participant in the order makes the decision and sends it to its predecessor and so on. In this way, L2PC maintains the same log complexity as 2PC, reduces



Three-Phase Commit. Figure 2. Some significant steps in the evolution of ACPs.

the message complexity of 2PC from "3" to "2n" while increasing the time complexity of 2PC from "3" to "2n" rounds, where n is the number of participants. In contrast to L2PC, *decentralized* 2PC (D2PC) reduces time complexity at the expense of message complexity which is $n^2 + n$ messages. In D2PC, the interconnecting communication network is assumed to be fully connected and efficiently supports the broadcasting of messages. In D2PC, two rounds of messages are required for each individual participant to make a final decision. During the first round, the coordinator broadcasts its vote (implicitly initiating commit processing) whereas, during the second one, all the participants broadcast their votes. Thus, each participant receives the votes of all the other participants, as well as the coordinator, and thereby, is able to independently conclude the final decision. By reducing the time complexity to two rounds, it becomes less likely for a participant, in D2PL, to be blocked during commit processing in the case of a coordinator's failure.

There are four transaction type specific 2PC protocols, all of which, when applicable, improve both the message and time complexities of 2PC by eliminating the explicit voting phase of 2PC. The *unsolicited-vote* protocol (UV) (1979) shortens the voting phase of 2PC assuming that each participant knows when it has executed the last operation for a transaction. In this way, a participant sends its vote on its own initiative once it recognizes that it has executed the last operation for the transaction. When the coordinator receives the votes of the participants, it proceeds with the decision phase. The *early prepare* protocol (EP) (1990) combines UV with PrC without assuming that a participant can recognize the last operation of a transaction. Every operation is, therefore, treated as if it is the last operation executing at the participant and its acknowledgment is interpreted as a "yes" vote. This means that a participant has to force write its log each time it executes an operation so that it can preserve the global atomicity of the transaction after a system crash. In contrast to EP which reduces time and message complexities at the expense of log complexity, the *coordinator log* (CL) and *implicit yes-vote* (IYV) protocols do not require force writing the log records, at the participants' sites, after the execution of each operation. Instead, they replicate the participants' logs at the coordinators' site. Hence, reducing log complexity compared to EP at the expense, however, of slower recovery. In CL, a participant does not maintain a local stable log and, therefore, it has to contact all coordinators in the system in order to recover after a failure. Moreover, a participant may need to contact the coordinator of a transaction during normal processing to maintain *write-ahead logging* (WAL) or to undo the effects of an aborting transaction. This is because the log of a participant is scattered across the coordinators' sites. In contrast, the log of a participant in IYV is partially replicated across the coordinators' sites. That is, only the *redo* records are replicated at the coordinators' sites while the *undo* records are stored locally. Thus, a participant, in IYV, never communicates with any coordinator to maintain WAL or to undo the effects of aborting transactions. Thus, in IYV, the replicated records are used only to recover a participant after a system's crash. Furthermore, IYV is based on PrA while CL is derived from PrC.

Existing 2PC variants are incompatible with each other and need to be made to interoperate in order to be integrated in (heterogeneous) multidatabase systems and the Internet. Thus, the continued research for more efficient ACPs has expanded to include the investigation of integrated ACPs. Efforts in this direction include the Harmony prototype system that integrates *centralized* participants that use centralized (asymmetric) ACPs with *centralized* participants that use decentralized (symmetric) ACPs (1991), the integration of *distributed* participants that use symmetric ACPs with *distributed* participants that use asymmetric ACPs (1994), and the *presumed any* protocol (1996) that integrates participants that use 2PC, PrA, or PrC. Besides that, recent efforts are targeted towards understanding the sources of incompatibilities [1] and the integration of ACPs in an adaptive manner to achieve higher system performance [8].

Several optimizations have been proposed that can reduce the costs associated with ACPs [5,2]. These include the *read-only, last agent, group commit, sharing the log, flattening the transaction tree* and *optimistic* optimizations. The read-only optimizations can be considered as the most significant ones, given that read-only transactions are the majority in any general database system. In fact, the performance gains allowed by the *traditional read-only* optimization provided the argument in favor of PrA to become the current choice of ACPs in the ISO OSI-TP (1998) and X/Open DTP (1996) distributed transaction processing standards, and commercial systems. The basic idea behind the read-only optimizations is that a read-only participant, a participant that has not performed any updates on behalf of a transaction, can be excluded from the decision phase of the transaction. This is because it does not matter whether the transaction is finally committed or aborted at a read-only participant to ensure the transaction's atomicity. In the traditional read-only optimization [4], a read-only participant votes "read-only" instead of a "yes" and immediately releases all the resources held by the transaction without writing any log records. A "read-only" vote allows a coordinator to recognize and discard the read-only participant from the rest of the protocol. The *unsolicited update-vote* (1997) is another read-only optimization that further reduces the costs associated with read-only participants. Not only that, but it incurs the same costs when used with both PrA and PrC, supporting the arguments for PrC to be also included in the standards.

The *last agent* optimization has been implemented by a number of commercial systems to reduce the cost of commit processing in the presence of a *single remote* participant. In this optimization, a coordinator first prepares itself and the nearby participants for commitment (fast first phase), and then delegates the responsibility of making the final decision to the remote participant. This eliminates the voting phase involving the remote participant. This same idea of delegating part of commitment (i.e., transferring the commit responsibilities) from one site to another has been also used to reduce blocking, for example, in open commit protocols (1990) and IYV with a commit coordinator (1996).

The *group commit* optimization has been also implemented by a number of commercial products to reduce log complexity. In the context of centralized database systems, a commit record pertaining to a transaction is not forced on an individual basis. Instead, a single force write to the log is performed when a number of transactions are to be committed or when a timer has expired. In the context of distributed database systems, this technique is used at the participants' sites *only* for the commit records of transactions during commit processing. The *lazy commit* optimization is a generalization of the *group commit* in which not only the commit records at the participants are forced in a group fashion, but *all* log records are lazily forced written onto stable storage during commit processing. Thus, the cost of a single access to the stable log is amortized among several transactions. The s*haring of log* between the transaction manager and data managers [5] at a site is another optimization that takes advantage of the sequential nature of the log to eliminate the need of force writing the log by the data managers.

The *flattening of the transaction tree* optimization is targeted for the tree-of-processes transaction model and is a big performance winner in distributed transactions that contain deep trees. It can reduce both the message and log complexities of an ACP by transforming the transaction execution tree of *any* depth into a two-level commit tree at commit initiation time. In this way, the root coordinator sends coordination messages directly to, and receives messages directly from, any participant. Thus, avoiding propagation delays and sequential forcing of log records. *Restructuring-the-commit-tree-around-update-participants* (RCT-UP) is an enhancement to the flattening technique that flattens only update participants (participants that have executed update operations on behave of the transaction), thereby, connecting them directly to the coordinator while leaving read-only participants connected in a multi-level manner. This is to reduce the effects of the communication delays on the overall system performance in systems that do not support simultaneous message multicasting to all participants.

Another optimization is *optimistic* (OPT) (1997) which can enhance the overall system performance by reducing blocking arising out of locks held by prepared transactions. OPT shares the same assumption as PrC, that is, transactions tend to commit when they reach their commit points. Under this assumption, OPT allows a transaction to *borrow* data that have been modified by another transaction that has entered a prepared-to-commit state and has not committed. A borrower is aborted if the lending transaction is finally aborted.

Key Applications

3PC has never been implemented in any commercial database system due to its cost, during normal transaction processing, compared to the other ACPs. This is besides its implementation complexity and, especially, the complexity of its termination protocol. Even with the added implementation complexity and cost, 3PC does not *completely* eliminate blocking since it is still susceptible to blocking in case of network partitioning. In fact there is no ACP that is non-blocking in the case of site as well as communication failures. This is an inherent characteristic of the *Byzantine Generals Problem*, the more general problem of atomic commitment. However, the protocol remains an instrumental theoretical result for understanding the behavior of ACPs and the limitations in solving the atomic commitment problem.

Cross-references

▶ Atomicity
▶ Distributed Database Systems
▶ Distributed Recovery
▶ Distributed Transaction Management

Recommended Reading

1. Al-Houmaily Y. Incompatibility dimensions and integration of atomic commit protocols. Int. Arab J. Inf. Technol., 5(4):2008.
2. Chrysanthis P.K., Samaras G., and Al-Houmaily Y. Recovery and performance of atomic commit processing in distributed database systems, Chapter 13. In Recovery Mechanisms in Database Systems, V. Kumar, M. Hsu (eds.). Prentice Hall, Upper Saddle River, NJ, 1998, pp. 370–416.
3. Lamport L., Shostak R., and Pease M. The byzantine generals problem. ACM Trans. Programming Lang. Syst., 4(3):382–401, 1982.
4. Mohan C., Lindsay B., and Obermarck R. Transaction Management in the R* Distributed Data Base Management System. ACM Trans. Database Syst., 11(4):378–396, 1986.
5. Samaras G., Britton K., Citron A., and Mohan C. Two-phase commit optimizations in a commercial distributed environment. Distrib. Parall. Databases, 3(4):325–361, 1995.
6. Skeen D. Non-blocking Commit Protocols. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1981, pp. 133–142.
7. Skeen D. and Stonebraker M. A Formal model of crash recovery in a distributed system. IEEE Trans. Softw. Eng., 9(3):219–228, 1983.
8. Yu W. and Pu C. A Dynamic two-phase commit protocol for adaptive composite services. Int. J. Web Serv. Res., 4(1), 2007.

Thresholding

▶ Image Segmentation

Tight Coupling

Serguei Mankovskii
CA Labs, CA Inc., Thornhill, ON, Canada

Synonyms

Strong coupling

Definition

Tight coupling indicates strong dependency between software components. The dependency is strong in the sense that two or more components have multiple and complex dependencies between internal states, data and functions of the components.

Key Points

Tight coupling often achieves high performance characteristics at the expense of flexibility and ease of maintenance. It is often justified for systems that are going to be used as a black box, with no expectation of ongoing maintenance or upgrade. The smaller a systems is, the more congruent its function is, the more likely it would justify tight coupling as a design principle. This is why one can find tight coupling in the components of larger systems. In this situation a number of tightly coupled components might be interacting using loosely coupled approach. This type of systems leads to robust designs where maintainability and flexibility is achieved by simply replacing one or more components. At the same time, each component performs in the best possible way because of tight coupling inside of it. This is the idea behind Service Oriented Architectures (SOA). This way SOA achieves balance between performance and flexibility. However it is not of the case for other architectures and because of that it is often beneficial to evaluate tradeoffs of both approaches during system design.

Cross-references

▶ Loose Coupling
▶ SOA
▶ Tight Coupling

Time Aggregated Graphs

Betsy George, Shashi Shekhar
University of Minnesota, Minnesota, Minneapolis, MN, USA

Synonyms

Spatio-temporal graphs; Time-dependent graphs; Time-dependent networks

Definition

A time aggregated graph [1,2] is a model that can be used to represent a spatio-temporal network. The topology and the attributes in a spatio-temporal network are typically time-dependent. Time aggregated graphs aggregate the time-dependent attributes over edges and nodes. This model facilitates the computation of path queries on a network accounting for the time-dependence of network parameters.

Key Points

Graphs have been extensively used to model spatial networks; weights assigned to nodes and edges are used to encode additional information. For example, the travel time between two intersections in a road network can be represented by the weight of the edge connecting the nodes that represent the intersections. In a real world scenario, it is not uncommon for these network parameters to be time-dependent. A time aggregated graph is a model that can capture the time-dependence of network parameters. In addition, the model captures the possibility of edges and nodes being absent during certain instants of time.

The model represents the time-variance of attributes by modeling them as time series.

Spatio-temporal networks have critical applications in domains such as transportation science. Developing storage efficient models for such networks that support the design of efficient query processing algorithms is important. Time aggregated graphs provide a means to model spatio-temporal networks and formulate algorithms that honor the time dependence of spatial networks.

Cross-references

▶ Spatio-Temporal Graphs
▶ Spatio-Temporal Networks

Recommended Reading

1. George B. and Shekhar S. Time-aggregated graphs for modeling spatio-temporal networks – an extended abstract. In Proc. Workshops at Int. Conf. on Conceptual Modeling, 2006, pp. 85–93.
2. George B. Kim S. and Shekhar S. Spatio-temporal network databases and routing algorithms: a summary of results. In Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases, 2007, pp. 460–477.

Time and Information Retrieval

Omar Alonso, Michael Gertz
University of California at Davis, Davis, CA, USA

Synonyms

Temporal information retrieval

Definition

Traditional information retrieval (IR) is concerned with models, algorithms, and architectures for the retrieval and ranking of documents from a document collection based on their relevance to search queries. In temporal information retrieval, expressions (words or phrases) that relate to instants in time, events, time periods, or other temporal descriptions are extracted from documents and handled in a special way to rank (and optionally group) the documents returned for a search query. Thus, in temporal information retrieval, temporal expressions extracted from documents play a special role in the overall relevance and in the organization and exploration of search results along timelines.

Historical Background

Research on using time information for retrieval and browsing activities is fairly recent. From a search perspective, there is previous work on placing search results in a timeline to facilitate the exploration of information [2,3,10]. A general overview of the basic idea of search result clustering using temporal document annotations obtained through a named-entity extraction approach has been outlined by Alonso and Gertz [4].

Research on temporal annotations has gained a lot of attention lately, and it is covered in great depth in the book edited by Mani et al. [8]. The work also includes discussions about tense and structural analysis and temporal reasoning techniques. The special issue on temporal information processing shows a wide range of current research directions and applications like question-answering and summarization [9].

News in particular have been the preferred information source for most of the related work in temporal information retrieval. Swan and Allan combine news topic detection and tracking with timelines as a browsing interface [14]. Automatic assignment of document event-time periods and automatic tagging of news messages using entity extraction is presented by Schilder and Habel in [11]. Their work also presents a temporal tagger along with its evaluation. There is very interesting work on adding time to applications like news for presenting temporal summaries as introduced by Allan et al. [1]. The work by Shaparenko et al. [12] concentrates on analyzing the development of a document collection over time and identifying temporal pattern. The work by Koen and Bender in Time Frames is one approach to augment news articles by extracting time information [7]. Recently, new research has emerged for future retrieval proposed by Baeza-Yates [5] where the idea of exploiting temporal information is developed for searching the future.

Foundations

Overview and Motivation

Time is an important dimension of any information space and can be very useful in information retrieval. Time and time measurements can help in outlining a particular historical period or establishing the context of a document. As an alternative to document ranking techniques like those based on popularity, time can be valuable for placing search results in a timeline for document exploration purposes. Current information retrieval systems and applications, however, do not take advantage of all the time information available within documents to provide better search results and thus to improve the user experience.

A quick look at any of the current search engines and information retrieval systems shows that the temporal viewpoint is restricted to sorting the search result represented in a hit list by date only. The date attribute is mainly the creation or last modified date of a Web page or document. In some cases it can be misleading, because the timestamp provided by a Web server or any other document management system may not be accurate. Other search applications provide a range date search as part of the advanced search options. Still, the search results are filtered based on the date attribute. For search purposes, the time axis is mainly constructed using that type of document metadata.

Even simple queries against Web search engines show that oftentimes organizing the documents in a hit list along some timeline can be helpful. For example, a query for "soccer world cup" against search engines now returns mostly pointers to documents that cover the recent event in Germany. But every soccer fan knows that this event happens every four years. Another example is "Iraq war;" here, results are primarily related to the latest events with little from the 1990's war. Clearly, it would be useful if a tool on top of a traditional retrieval system is more aware of the temporal information embedded in the documents and allows the user to have search results presented in different ways based on the temporal information. For this, it is essential to extract temporal information from documents and associate documents with points in time along well-defined timelines.

Time and Timelines

As the basis for associating points in time with documents, it is customary to assume a discrete representation of time based on the Gregorian Calendar, with a single day being an atomic time interval called a *chronon*. A base timeline, denoted T_d, is an interval of consecutive day chronons. For example, the sequence "March 12, 2002; March 13, 2002; March 14, 2002" is a contiguous subsequence of chronons in T_d. Contiguous sequences of chronons can be grouped into larger units called *granules*, such as weeks, months, years, or decades. A grouping based on a granule results in a more coarse-grained timeline, such as T_w based on weeks, T_m based on months, or T_y based on years. Examples of week chronons in T_w are "3rd week of 2005" or "last week of 2006." Depending on the type of underlying calendar, base timeline, and grouping of chronons, timelines of different time granularity can be constructed. Chronons from two timelines then can also be compared. For example, "March 18, 2002" (chronon in T_d) lies before "December 2006" (chronon in T_m). Timelines constructed in this way then serve as the basis to have temporal expressions in documents refer to chronons in one or more timelines.

Temporal Expressions

There is quite a lot of temporal information in any corpus of documents. For example, financial news tend to be rich in describing near future events; resume documents contain several references to the past in a very precise way; and project documentation involves phase milestones that are captured in time. However, what types of temporal information (besides a simple document timestamp) are there and how do they relate to timelines?

In general, with the textual content of a document, a set of *temporal entities* can be associated. A temporal entity describes a point in time, event, or time period at a conceptual level. The identification of such entities involves a linguistic analysis of the document, where approaches based on named-entity extraction determine so-called *temporal expressions*. A temporal expression is basically a sequence of tokens that represent an instance of a temporal entity. Contrary to other entities such as names and places, temporal entities can be represented as temporal expressions that are sequences of not necessarily contiguous tokens or words. Expressions can be mapped to temporal entities that are defined in some time ontology. Similar to the approach by Schilder and Habel [11], this discussion distinguishes between explicit, implicit, and relative temporal expressions. *Explicit temporal expressions* directly describe entries in some timeline, such as an exact date or year. For example, the token sequences "December 2004" or "September 12, 2005" in a document are explicit temporal expressions and can be mapped directly to chronons in a timeline (here T_m and T_d, respectively).

Depending on the underlying time ontology and capabilities of the named-entity extraction approach, even apparently imprecise temporal information, such as names of holidays or events can be anchored in a timeline. For example, the token sequence "Columbus Day 2006" in the text of a document can be mapped to the expression "October 12, 2006," or the sequence "Labor Day 2008" can be mapped to "September 1, 2008." Such types of temporal expressions whose mapping to entities relies on the capability of the underlying time ontology are called *implicit temporal expressions*.

Relative temporal expressions represent temporal entities that can only be anchored in a timeline in reference to another explicit or implicit, already anchored temporal expression (which, in the worst case, is the document timestamp). For example, the expression "today" alone cannot be anchored in any timeline. However, it can be anchored if the document is known to have a creation date. This date therefore can be used as a reference for that expression, which then can be mapped to a chronon. There are many instances of relative temporal expressions, such as the names of weekdays (e.g., "on Thursday") or months (e.g., "in July") or references to such points in time like "next week" or "last Friday."

Given a document collection $\mathcal{D}$, the temporal expressions that have been determined for each document $d \in \mathcal{D}$ can be represented in the form of a *temporal document profile*. For example, a profile then records for each type of expression the token sequence, position of the sequence in the document d, and the chronon to which the expression has been mapped. With a document, several expressions of different types and corresponding chronons (in different timelines) can be associated. The same expression and chronon can even appear several times in the same document, but then at different positions in the document. Each document at least has one explicit temporal expression, which is assumed to be the document timestamp. Next, an extraction approach for temporal expressions using existing tools is outlined.

Temporal Processing Pipeline

Given a document collection $\mathcal{D}$, the identification of the temporal expressions in each document $d \in \mathcal{D}$ is realized through a *document processing pipeline*, which includes a sequence of operations as follows. The first step is to extract the timestamp from the document. This can be the creation or last modified date for a file. In case of a Web page, one can rely on the information provided by the Web server. The second step is to run a part of speech tagger (POS tagger) on every document. A POS tagger returns the document with parts of speech assigned to each word/token like noun, verb etc. The tagger also tags sentence delimiters that later are needed for extracting the temporal expressions. The third step is to run a temporal expression tagger on the POS-tagged version of the document, which recognizes the extents and normalized values of temporal expressions [6]. This step extracts temporal expressions based on the TimeML standard and produces an XML document. The TimeML specification for temporal annotations seems to be a suitable approach here, because it has emerged as the standard markup language for events and temporal expressions in natural language [15]. The resulting XML document is the original document annotated by various information about the temporal expressions that have been determined for the original document. This information then can be used to construct the temporal document profile for the document, which, in turn, can be used for different time-centric document information retrieval and exploration approaches.

Document Retrieval

A fundamental property of any information retrieval system is the ability to help users find documents that satisfy their information needs. At a first glance, this looks pretty obvious but it is not, because users are not very expressive in describing what information they want. Furthermore, the information needs they specify can be ambiguous, making the retrieval task even harder. A user will judge whether or not a query result satisfies her information needs based on whether she considers the result to the search query relevant. The central idea underlying a temporal information retrieval approach is to utilize the temporal expressions that have been determined for each document in a given document collection $\mathcal{D}$ in order to group and/or rank search results based on the temporal information embedded in the documents. By using this approach, time plays a central role in the overall quality of the search results. Assume a standard information retrieval or search application that returns a hit list of n documents $L_q = \langle d_1,...,d_n \rangle$ for a search query q. The search application retrieves the result based on the relevance of the documents with respect to q using traditional metrics based on tf/idf and the distance of the query terms to the first token of temporal expressions in the documents. After all, *tense* happens at the sentence level so it is important to detect these "boundaries" with respect to the query q. There are several ways in which the temporal expressions in the documents in L_q can be used to group the documents using temporal aspects. In the following, only the general idea of these approaches is illustrated. For example, the following algorithm outlines how the usage of temporal expressions can help to group search results based on the temporal expressions determined from the documents in L_q.

1. Determine the document hit list $L_q = \langle d_1,...,d_n \rangle$ that satisfies the search query q, sorted by relevance of the documents.
2. Determine the temporal expressions $T = \{te_1,...,te_m\}$ for all the n documents in the hit list. Note that a document $d_i \in L_q$ can have several temporal expressions, represented in d_i's temporal document profile.
3. Choose a type of time granule g (e.g., year or month). Sort the temporal expressions in T using that granule as key. For example, the expression "September 1, 2007" then comes before the expression "August 2007," assuming a descending order.
4. For each granule of type g (e.g., a particular year or month), take all those documents from L_q that contain a temporal expression covered by that granule. Rank these documents using the distance between the query terms in q to the temporal expressions.
5. Display document groups using the granule type g as label in a timeline fashion in descending order. For example, if the granule type year has been chosen, for each year, there is a group of documents that contain a temporal expression related to that year (perhaps at a finer level of granularity). Note that instances of the granule type are based on a timeline.

If a document $d \in L_q$ contains several temporal expressions, this document can end up in different groups and at a different rank in each group. In general, the above approach organizes documents from a hit list along a timeline based on a time granule type. A group of documents related to a particular instant of a granule then can be organized further, i.e., based on a more fine-grained time granule.

Key Applications

Recently, exploratory search system have emerged as a specialization of information exploration to support serendipity, learning, and investigation, and, more generally, to allow users to browse available information. For such systems, taking advantage of temporal expressions embedded in the documents leads to a much richer framework for exploration. This is an important ingredient for the information forager who is trying to see the profit in terms of the interaction cost required to gain useful information from an information source. Users tend to prefer sources, in this case search engines, that are richer in good results. These good results involve adding important nuggets such as temporal information and relationships. The news domain is another area where search, presentation, and filtering can be greatly improved by using more temporal information. As an example, financial news about a company's Q4 earnings has a very precise meaning in a time-related context. Exploring and analyzing news by these types of temporal expressions can be very useful for particular application domains, for example, in the area of financial analysis.

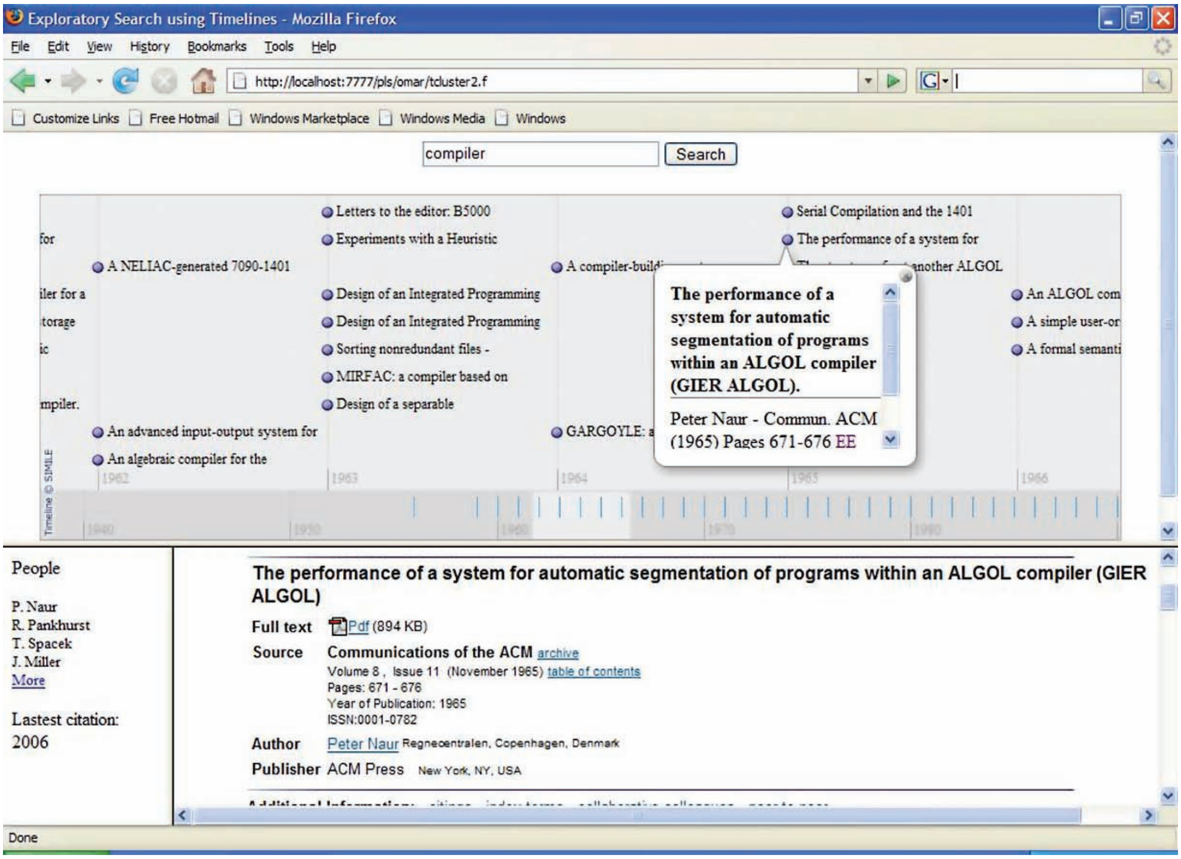
Timeline-Based Exploration

Current interfaces to search engines typically present search results sorted by the relevance of documents from a document collection to a search query. For this, the freshness of the information is considered an important part of the quality of the result. Temporal attributes in Web pages or documents such as date, however, are just viewed as some structured criteria to sort the result in descending order of relevance. At the time of this writing there is a new experimental feature for timelines as part of Google (`view:timeline`).

Another exploratory search prototype outlined in the following is a Web-based application that uses the SIMILE timeline toolkit for visualization and exploration purposes [13], here with respect to a document collection of journal articles from DBLP (http://www.informatik.uni-trier.de/~ley/db/). The interface is organized as follows. The main section takes half of the screen and contains the search box and the timeline. The timeline consists of two bands that represent different time scales (types of granules): decade and year. Both bands are synchronized such that panning one band also scrolls the other. The lower band (decade) is much smaller since the goal is to show the activity in a decade. The upper band shows all articles in a given year. Figure 1 shows the exploratory search interface in action for the query "compiler" against the collection. The system retrieves all journal articles that contain the term "compiler" in the title and returns a hit list clustered by year (in which the article appeared). Search results are anchored in the timeline, that is, documents (or rather the embedded temporal expressions) are linked to instants in time. If more than one article falls within a year, the order of the documents in such a group is based on each document's relevance to the query. Obviously, such an information exploration and visualization approach and tool on top of a more traditional search application can be extremely valuable in the context of temporal information retrieval.

Future Directions

Evaluations of the impact of temporal attributes for search and retrieval are needed to asses the importance of these techniques. This, of course, can only be


Exploratory Search using Timelines - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:7777/pls/omar/tcluster2.f
Customize Links
Free Hotmail
Windows Marketplace
Windows Media
Windows
compiler
Search
Letters to the editor: B5000
Experiments with a Heuristic
A NELIAC-generated 7090-1401
A compiler-build
Serial Compilation and the 1401
The performance of a system for
another ALGOL
for
iler for a
torage
ic
mpiler.
Design of an Integrated Programming
Design of an Integrated Programming
Sorting nonredundant files -
MIRFAC: a compiler based on
Design of a separable
An advanced input-output system for
An algebraic compiler for the
GARGOYLE: a
An ALGOL com
A simple user-or
A formal semanti
1962
1963
1964
1965
1966
1940
1950
1960
1970
1980
Timeline © SIMILE
Done

The performance of a system for automatic segmentation of programs within an ALGOL compiler (GIER ALGOL).

Peter Naur - Commun. ACM (1965) Pages 671-676 EE

People

P. Naur
R. Pankhurst
T. Spacek
J. Miller
More

Lastest citation:
2006

The performance of a system for automatic segmentation of programs within an ALGOL compiler (GIER ALGOL)

Full text Pdf (894 KB)

Source Communications of the ACM archive
Volume 8 , Issue 11 (November 1965) table of contents
Pages: 671 - 676
Year of Publication: 1965
ISSN:0001-0782

Author Peter Naur Regnecentralen, Copenhagen, Denmark

Publisher ACM Press New York, NY, USA

Time and Information Retrieval. Figure 1. Exploring research articles using the SIMILE timeline toolkit.

applied to those applications where the usage of time information has some (expected) benefit.

Cross-references

- ▶ Document Clustering
- ▶ Information Extraction
- ▶ Information Retreival
- ▶ Structured Document Retrieval
- ▶ Web Search Relevance Ranking

Recommended Reading

1. Allan J., Gupta R., and Khandelwal V. Temporal summaries of news topics. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 10–18.
2. Allen R.B. A focus-context browser for multiple timelines. In Proc. ACM/IEEE Joint Conf. on Digital Libraries, 2005, pp. 260–261.
3. Alonso O., Baeza-Yates R., and Gertz M. Exploratory search using timelines. In Proc. SIGCHI 2007 Workshop on Exploratory Search and HCI Workshop, 2007.
4. Alonso O. and Gertz M. Clustering of search results using temporal attributes. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 597–598.
5. Baeza-Yates R. Searching the future. In ACM SIGIR 2005 Workshop on Mathematical/Formal Methods in Information Retrieval, 2005.
6. GUTime. Available at: http://complingone.georgetown.edu/~linguist
7. Koen D.B. and Bender W. Time frames: temporal augmentation of the news. IBM Syst. J., 39(3–4):597–616, 2000.
8. Mani I., Pustejovsky J., and Gaizauskas R. (eds.). The Language of Time. Oxford University Press, New York, NY, USA, 2005.
9. Mani I., Pustejovsky J., and Sundheim B. Introduction to the Special Issue on Temporal Information Processing. ACM Trans. Asian Lang. Inform. Process., 3(1):1–10, March 2004.
10. Ringel M., Cutrell E., Dumais S.T., and Horvitz E. Milestones in time: the value of landmarks in retrieving information from personal stores. In Proc. IFIP TC13 Int. Conf. on Human-Computer Interaction, 2003, pp. 184–191.
11. Schilder F. and Habel C. From temporal expressions to temporal information: semantic tagging of news messages. In Proc. ACL 2001 Workshop on Temporal and Spatial Information Processing, 2001.
12. Shaparenko B., Caruana R., Gehrke J., and Joachims T. Identifying temporal patterns and key players in document collections. In Proc. IEEE ICDM Workshop on Temporal Data Mining: Algorithms, Theory and Applications, 2005, pp. 165-174.
13. SIMILE Timeline toolkit. Available at: http://simile.mit.edu/timeline/
14. Swan R. and Allan J. Automatic generation of overview timelines. In Proc. 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2000, pp. 49–56.
15. TimeML, markup language for temporal and event expressions. Available at: http://www.timeml.org/

Time Dependent Geometry

- ▶ Moving Object

Time Distance

- ▶ Time Interval
- ▶ Time Span

Time Domain

ANGELO MONTANARI[1], JAN CHOMICKI[2]
[1]University of Udine, Udine, Italy
[2]State University of New York at Buffalo, Buffalo, NY, USA

Synonyms

Temporal domain; Temporal structure

Definition

In its full generality, a time domain can be defined as a set of *temporal individuals* connected by a set of *temporal relations*. Different choices for the temporal individuals and/or the temporal relations give rise to different temporal ontologies.

In the database context, the most common temporal ontology takes *time instants* (equivalently, points or moments) as the temporal individuals and a *linear order* over them as the (unique) temporal relation [5]. In addition, one may distinguish between discrete and dense, possibly continuous, time domains and between bounded and unbounded time domains. In the discrete case, one may further consider whether the time domain is finite or infinite and, in the case of unbounded domains, one can differentiate between left-bounded, right-bounded, and totally unbounded domains. Moreover, besides linear time, one may consider *branching time*, where the linear order is replaced with a partial one (a tree or even a directed acyclic graph), or circular time, which can be used to represent temporal periodicity.

As for temporal individuals, time instants can be replaced with *time intervals* (equivalently, periods or anchored stretches of time) connected by (a subset of) Allen's relations `before`, `meets`, `overlaps`, `starts`, `during`, `equal`, and `finishes`, and their inverses or suitable combinations [7]. As in the case of instant-based domains, one may distinguish between discrete and dense domains, bounded and unbounded domains, linear, branching, and circular domains, and so on.

Finally, as most temporal database applications deal with both qualitative and quantitative temporal aspects, instant-based time domains are usually assumed to be isomorphic to specific numerical structures, such as those of natural, integer, rational, and real numbers, or to fragments of them, while interval-based ones are obtained as suitable intervallic constructions over them. In such a way, time domains are endowed with *metrical features*.

Historical Background

The nature of time and the choice between time instants and time intervals as the primary objects of a temporal ontology have been a subject of active philosophical debate since the times of Zeno and Aristotle. In the twentieth century, major contributions to the investigation of time came from a number of disciplines. A prominent role was played by Prior who extensively studied various aspects of time, including axiomatic systems of tense logic based on different time domains.

Nowadays, besides physics, philosophy, and linguistics, there is a considerable interest in temporal structures in mathematics (theories of linear and branching orders), artificial intelligence (theories of action and change, representation of and reasoning with temporal constraints, planning), and theoretical computer science (specification and verification of concurrent and distributed systems, formal analysis of hybrid temporal systems that feature both discrete and continuous components). A comprehensive study and logical analysis of instant-based and interval-based temporal ontologies, languages, and logical systems can be found in [2].

As for *temporal databases*, the choice of the time domain over which temporal components take their value is at the core of any application. In most cases, a discrete, finite, and linearly ordered (instant-based) time domain is assumed. This is the case, for instance, with SQL standards [9]. However, there is no single way to represent time in a database, as witnessed by the literature in the field. To model when something happened, time instants are commonly used; validity of a fact over time is naturally represented by the (convex) set of time instants at which the fact holds, the time *period of validity* in the temporal database terminology; finally, to capture processes as well as some kinds of temporal aggregation, time intervals are needed.

Foundations

Basics

The choice between time instants and time intervals as the basic time constituents is a fundamental decision step that all temporal systems have in common. In mathematics, the choice of *time instants*, that is, points in time without duration, is prevalent. Although quite abstract, such an solution turned out extremely fruitful and relatively easy to deal with in practice. In computer science, additional motivations for this choice come from the natural description of computations as possibly infinite sequences of instantaneous steps.

The alternative option of taking *time intervals*, that is, anchored stretches of time with duration, as temporal individuals seems to better adhere to the concrete experience of people. Physical phenomena as well as natural language expressions involving time can be more easily described in terms of time intervals instead of time instants. Nevertheless, the complexity of any systematic treatment of time intervals prevents many systems from the adoption of an interval-based ontology.

The instant and the interval ontologies are systematically investigated and compared in [2]. The author identifies the conditions an instant-based (resp., interval-based) structure must satisfy to be considered as an adequate model of time. Then, through an axiomatic encoding of such conditions in an appropriate language, he provides a number of (first-order and higher order) logical theories of both instant-based and interval-based discrete, dense, and continuous structures. Finally, he illustrates the strong connections that link the two time ontologies. In particular, he shows how interval-based temporal structures can be obtained from instant-based ones through the standard process of interval formation and how instant-based temporal structures can be derived from interval-based ones by a (non-trivial) limiting construction.

A metric of time is often introduced to allow one to deal with time distance and/or duration. In particular, a time metric is needed to define calendar times, such as those based on the commonly used Gregorian calendar.

Temporal Models and Query Languages

The choice of the time domain has an impact on various components of temporal databases. In particular, it influences temporal data models and temporal query languages.

As for *temporal data models*, almost all of them adopt an instant-based time ontology. Moreover, most of them assume the domain to be linear, discrete and finite. However, many variants of this basic structure have been taken into consideration [8]. Right-unbounded domains have been used to record information about the future. Dense and continuous domains have been considered in the context of temporal constraint databases, that allow one to represent large, or even infinite, sets of values, including time values, in a compact way. Branching time has been exploited in applications where several alternatives have to be considered in the future and/or past evolution of temporal data.

Many data models distinguish between absolute (anchored) and relative (unanchored) time values. Absolute time values denote specific temporal individuals. In general, they are associated with a time metric, such as that of calendar times. As an example, the 14th of September 2007 is an absolute time value that denotes a specific element of the domain of days in the Gregorian calendar. Relative time values specify the distances between pairs of time instants or the durations of time intervals. Absolute and relative time values can also be used in combination. As an example, the expression 7 days after the 14th of September 2007 denotes the 21st of September 2007.

As for *temporal query languages*, they typically assume that time is isomorphic to natural numbers. This is in agreement with the most common, *linear-time* dialect of temporal logic. In temporal constraint databases, however, the use of classical query languages like relational calculus or algebra accommodates a variety of time domains, including dense and continuous ones.

Time Domain and Granularity

Despite its apparent simplicity, the addition of the notion of time domain to temporal databases presents various subtleties. The main ones concern the nature of the elements of the domain. As soon as calendar times come into play, indeed, the abstract notion of instant-based time domain must be contextualized with respect to a specific *granularity* [3,4]. Any given granularity can be viewed as a suitable abstraction of the real time line that partitions it into a denumerable sequence of homogeneous stretches of time. The elements of the partition, granules in the temporal database terminology, become the individuals (non-decomposable time units) of a discrete time domain. With respect to the considered granularity, these temporal individuals can be assimilated to time instants. Obviously, if a shift to a finer granularity takes place, e.g., if one moves from the domain of months to the domain of days, a single granule must be replaced with a set of granules. In such a way, being instantaneous is not more an intrinsic property of a temporal individual, but it depends on the time granularity one refers to. A detailed analysis of the limitations of the temporal database management of instant-based time domains can be found in [9].

The Association of Time with Data

The association of the elements of the time domain with data is done by *timestamping*. A timestamp is a time value associated with a data object. In the relational setting, one distinguishes between attribute-timestamped data models, where timestamps are associated with attribute values, and tuple-timestamped data models, where timestamps are associated with tuples of values. As a third possibility, a timestamp can be associated with an entire relation/database.

Timestamps can be single elements as well as sets of elements of the time domain. Time instants are usually associated with relevant events, e.g., they can be used to record the day of the hiring or of the dismissal of an employee. (Convex) sets of time instants are associated with facts that hold over time. As an example, if a person E works for a company C from the 1st of February 2007 to the 31st of May 2007, one keeps track of the fact that every day in between the 1st of February 2007 and the 31st of May 2007, endpoints included, E is an employee of C.

Time intervals are needed to deal with situations where validity over an interval cannot be reduced to validity over its subintervals (including point subintervals) [10]. This is the case with processes that relate to an interval as a whole, meaning that if a process consumes a certain interval it cannot possibly transpire during any proper subinterval thereof. Examples are the processes of baking a cake or of flying from Venice to Montreal. This is also the case when validity of a fact at/over consecutive instants/intervals does not imply its validity over the whole interval. As an example, two consecutive phone calls with the same values are different from a single phone call over the whole period. The same happens for some kinds of temporal aggregation [4]. Finally, the use of time intervals is common in several areas of AI, including knowledge representation and qualitative reasoning, e.g., [1].

It is important to avoid any confusion between this latter use of intervals as timestamps and their use as compact representations of sets of time points (time periods in the temporal database literature). Time intervals are indeed often used to obtain succinct representations of (convex) sets of time instants. In such a case, validity over a time period is interpreted as validity at every time instant belonging to it. As an example, the fact that a person E worked for a company C from the 1st of February 2007 to the 31st of May 2007 can be represented by the tuple (E, C, [2007/02/01, 2007/05/31]) meaning that E worked for C every day in the closed interval [2007/02/01, 2007/05/31].

Key Applications

As already pointed out, the time domain is an essential component of any temporal data model, and thus its addition to SQL standards does not come as a surprise.

In SQL, time domains are encoded via *temporal data types* (they have been introduced in SQL-92 and preserved in SQL:1999). In SQL-92, five (anchored) time instant data types, three basic forms and two variations, are supported (DATE, TIME, TIMESTAMP, TIME WITH TIME ZONE, TIMESTAMP WITH TIME ZONE). In addition, SQL-92 features two (unanchored) data types that allow one to model positive (a shift from an instant to a future one) and negative (a shift from an instant to a past one) distances between instants. One can be used to specify distances in terms of years and months (the YEAR-MONTH INTERVAL type), the other to specify distances in terms of days, hours, minutes, seconds, and fractions of a second (the DAY-TIME INTERVAL type). As a matter of fact, the choice of using the word interval to designate a time distance instead of a temporal individual – in contrast with the standard use of this word in computer science – is unfortunate, because it confuses a derived element of the time domain (the interval) with a property of it (its duration). An additional (unanchored) temporal data type, called PERIOD, was included in the SQL/Temporal proposal for the SQL3 standard, which was eventually withdrawn. A period is a convex sets of time instants that can be succinctly represented as a pair of time instants, namely, the first and the last instants with respect to the given order.

SQL also provides *predicates*, *constructors*, and *functions* for the management of time values. General predicates, such as the equal-to and less-than predicates, can be used to compare pairs of comparable values of any given temporal type; moreover, the specific overlap predicate can be used to check whether two time periods overlap. Temporal constructors are expressions that return a temporal value of a suitable type. It is possible to distinguish datetime constructors, that return a time instant of one of the given data types, and interval constructors, that return a value of YEAR-MONTH INTERVAL or DAY-TIME INTERVAL types. As for functions, they include the datetime value functions, such as the CURRENT_DATE function, that return an instant of the appropriate type, the CAST functions, that convert a value belonging to a given (temporal or non temporal) source data type into a value of the target temporal data type, and the extraction functions, that can be used to access specific fields of instant or interval time values.

Future Directions

Despite the strong prevalence of instant-based data models in current temporal databases, a number of interesting problems, such as, for instance, that of temporal aggregation, motivate a systematic study and development of *interval-based data models*. Moreover, in both instant-based and interval-based data models intervals are defined as suitable sets of elements of an instant-based time domain. The possibility of assuming time intervals as the primitive temporal constituents of the temporal domain is still largely unexplored. Such an alternative deserves a serious investigation.

Cross-references

- ▶ Now in Temporal Databases
- ▶ Period-Stamped Temporal Models
- ▶ Point-Stamped Temporal Models
- ▶ Temporal Algebras
- ▶ Temporal Constraints
- ▶ Temporal Data Models
- ▶ Temporal Granularity
- ▶ Temporal Indeterminacy
- ▶ Temporal Periodicity
- ▶ Temporal Query Languages

Recommended Reading

1. Allen J. and Ferguson G. Actions and events in interval temporal logic. J. Logic Comput., 4(5):531–579, 1994.
2. van Benthem J. The Logic of Time. A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse, 2nd edn. Kluwer, Dordrecht, Holland, 1991.
3. Bettini C., Jajodia S., and Wang X.S. Time Granularities in Databases, Data Mining, and Temporal Reasoning. Springer, NJ, USA, 2000.
4. Böhlen M.H., Gamper J., and Jensen C.S. How would you like to aggregate your temporal data? In Proc. 13th Int. Symp. Temporal Representation and Reasoning, 2006, pp. 121–136.
5. Chomicki J. and Toman D. Temporal databases. In Chapter 14 of the Handbook of Temporal Reasoning in Artificial Intelligence, M. Fisher, D. Gabbay, L. Vila (eds.). Elsevier B.V., Amsterdam, The Netherlands, 2005, pp. 429–467.
6. Euzenat J. and Montanari A. Time granularity. In Chapter 3 of the Handbook of Temporal Reasoning in Artificial Intelligence, M. Fisher, D. Gabbay, L. Vila (eds.). Elsevier B.V., Amsterdam, The Netherlands, 2005, pp. 59–118.
7. Goranko V., Montanari A., and Sciavicco G. A road map of interval temporal logics and duration calculi. J. Appl. Non-Class. Logics, 14(1-2):9–54, 2004.
8. Montanari A. and Pernici B. Temporal reasoning. In Chapter 21 of Temporal Databases: Theory, Design and Implementation, A. Tansell, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R. Snodgrass (eds.). Database Systems and Applications Series. Benjamin/Cummings, Redwood City, CA, USA, 1993, pp. 534–562.
9. Snodgrass R.T. Developing time-oriented database applications in SQL. In Chapter 3 of Instants and Intervals. Morgan Kauffman, San Francisco, CA, USA, 2000, pp. 24–87.
10. Terenziani P. and Snodgrass R.T. Reconciling point-based and interval-based semantics in temporal databases: a treatment of the telic/atelic distinction. IEEE Trans. Knowl. Data Eng., 16(5):540–551, 2004.

Time Granularity

▶ Temporal Granularity

Time in Philosophical Logic

Peter Øhrstrøm, Per F. V. Hasle
Aalborg University, Aalborg, Denmark

Synonyms

Temporal logic; Logic of time

Definition

The aim of the study of time in philosophical logic is to provide a conceptual framework for an interdisciplinary study of the nature of time and to formalize and study various conceptions and systems of time. In addition, the introduction of time into logic has led to the development of formal systems, which are particularly well suited to represent and study temporal phenomena such as program execution, temporal databases, and argumentation in natural language.

Historical Background

The philosophy of time is based on a long tradition, going back to ancient thought. It is an accepted wisdom within the field that no attempt to clarify the concept of time can be more than an accentuation of some aspects of time at the expense of others. Plato's statement that time is the "moving image of eternity" and Aristotle's suggestion that "time is the number of motion with respect to earlier and later" are no exceptions (see [17]). According to St. Augustine (354–430) time cannot be satisfactorily described using just one single definition or explanation: "What, then, is time? If no one asks me, I know: if I wish to explain it to one that asketh, I know not." [5, p. 40] Time is not definable in terms of other concepts. On the other hand, according to the Augustinian insight, all human beings have a tacit knowledge of what time is. In a sense, the endeavor of the logic of time is to study important manifestations and structures of this tacit knowledge.

There were many interesting contributions to the study of time in Scholastic philosophy, e.g., the analysis of the notions of beginning and ending, the duration of the present, temporal ampliation, the logic of "while," future contingency, and the logic of tenses. Anselm of Canterbury (ca. 1033–1109), William of Sherwood (ca. 1200–1270), William of Ockham (ca. 1285–1349), John Buridan (ca. 1295–1358), and Paul of Venice (ca. 1369–1429) all contributed significantly to the development of the philosophical and logical analysis of time. With the Renaissance, however, the logical approach to the study of time fell into disrepute, although it never disappeared completely from philosophy.

However, the twentieth century has seen a very important revival of the philosophical study of time. The most important contribution to the modern philosophy of time was made in the 1950s and 1960s by A. N. Prior (1914–1969). In his endeavors, A. N. Prior took great inspiration from ancient and medieval thinkers and especially their work on time and logic.

The Aristotelian idea of time as the number of motion with respect to earlier and later actually unites two different pictures of time, the dynamic and the static view. On the one hand, time is linked to motion, i.e., changes in the world (the flow of time), and on the other hand time can be conceived as a stationary order of events represented by numbers. In his works, A. N. Prior logically analyzed the tension between the dynamic and the static approach to time, and developed four possible positions in regard to this tension. In particular, A. N. Prior used the idea of branching time to demonstrate that there is a model of time which is logically consistent with his ideas of free choice and indeterminism. (See [8, 189 ff.].)

After A. N. Prior's development of formalised temporal logic, a number of important concepts have been studied within this framework. In relation to temporal databases the studies of the topology of time and discussions regarding time in narratives are particularly interesting.

Foundations

In the present context, the following four questions regarding time in philosophical logic seem to be especially important:

1. What is the relation between dynamic and static time?
2. What does it mean to treat time as "branching"?
3. What is the relation between punctual and durational time (i.e., instants and durations)?
4. What is the role of time in storytelling (narratives)?

In the following, a brief introduction to each of these issues will be given.

Dynamical and Static Time: A-Theory vs. B-Theory

The basic set of concepts for the *dynamic* understanding of time are past, present, and future. In his very influential analysis of time the philosopher John Ellis McTaggart (1866–1925) suggested to call these concepts (i.e., the tenses) the A-concepts. The tenses are well suited for describing the flow of time, since the future will become present, and the present will become past, i.e., flow into past. The basic set of concepts for the *static* understanding of time are before/after and "simultaneous with." Following McTaggart, these are called the B-concepts, and they seem especially apt for describing the permanent and temporal order of events. The two kinds of temporal notions can give rise to two different approaches to time. First, there is the dynamic approach (the A-theory) according to which the essential notions are past, present and future. In this view, time is seen "from the inside." Secondly, there is the static view of time (the B-theory) according to which time is understood as a set of instants (or durations) ordered by the before-after relation. Here time is seen "from the outside." It may be said to be a God's eye-perspective on time.

There is also an ontological difference between the two theories. According to the A-theory the tenses are real whereas the B-theorists consider them to be secondary and unreal. According to the A-theory the Now is real and objective, whereas the B-theories consider the Now to be purely subjective.

The debate between proponents of the two theories received a fresh impetus with A. N. Prior's formal analysis of the problem. (See [9, 216 ff.]). According to the B-theory, time is considered to be a partially ordered set of instants, and propositions are said to be true or false at the instants belonging to the set. According to the A-theory, time is conceived in terms of the operators *P* (Past) and *F* (Future), which are understood as being relative to a "Now." A. N. Prior suggested a distinction between four possible grades of tenselogical involvement corresponding to four different views of how to relate the A-notions (past, present and future) to the B-notions ("earlier than"/"later than," "simultaneous with"):

1. The B-notions are more fundamental than the A-notions. Therefore, in principle, the A-notions have to be defined in terms of the B-notions.
2. The B-notions are just as fundamental as the A-notions. The A-notions cannot be defined in terms of the B-notions or vice versa. The two sets of notions have to be treated on a par.
3. The A-notions are more fundamental than the B-notions. All B-notions can be defined in terms

of the A-notions and a primitive notion of temporal possibility.

4. The A-notions are more fundamental than the B-notions. Therefore, in principle the B-notions have to be defined in terms of the A-notions. Even the notion of temporal possibility can be defined on terms of the A-notions.

A. N. Prior's four grades of tense-logical involvement represent four different views of time and also four different foundations of temporal logic. In fact, theory 1 is the proper B-theory and theory 3 and 4 are versions of the proper A-theory. Theory 2 is a kind of intermediate theory.

In theory 1, the tense operators, P (past) and F (future), can be introduced in the following way:

$$T(t, Fq) \equiv_{def} \exists t_1 : t < t_1 \wedge T(t_1, q)$$

$$T(t, Pq) \equiv_{def} \exists t_1 : t_1 < t \wedge T(t_1, q)$$

where $T(t, q)$ is read "q is true at t," and $t < t_1$ is read "t is before t_1."

In theory 3 and 4, A. N. Prior has shown how instants can be introduced as maximally consistent sets of tense-logical propositions and how the before-after relation can be consistently defined in terms of tense-logical concepts (i.e., A-notions).

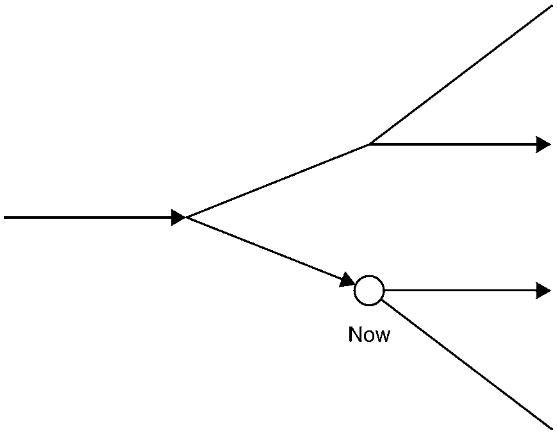
From a B-theoretical viewpoint, at any instant, an infinite number of propositions, including tensed ones, will be true about that instant. But from the A-theoretical point of view, precisely the infinite conjunction of the propositions in this set is a construction which, when called an "instant," makes the B-theoretical notion of "instant" secondary and derivable.

It should be noted, that whereas the A-theorist (theory 3 or 4) can translate any B-statement into his language, many A-statements cannot be translated into the B-language. For instance, there is no way to translate the A-statement "it is raining now in Aalborg" into the B-language. The "now" cannot be explained in terms of the B-language consisting of an ordered set of instants and the notion of a proposition being true at an instant. This asymmetry seems to be a rather strong argument in favor of the A-theory (i.e., A. N. Prior's theory 3 or 4).

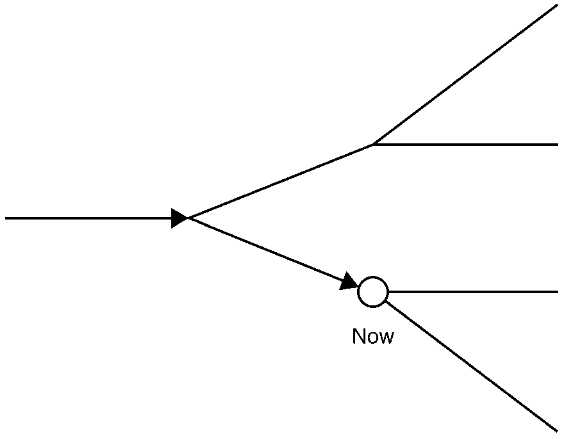
Linear vs. Branching Time

The idea of formalised branching time was first brought forward by Saul Kripke in a letter to A. N. Prior in 1958 [8, pp. 189–90]. Kripke's later development of the semantics for modal logics is well-known within computer science. But it has in fact been shown by Jack Copeland [3] that the kernel of the ideas published by Kripke were in fact present already in the work of Meredith and A. N. Prior in 1956.

The difference between A. N. Prior's theory 3 and 4 is important if time is considered to be branching. In theory 3, the notion of possibility is primitive. In theory 4, this notion can be derived from the tenses. But then it turns out to be very difficult to distinguish between the possible future, the necessary future and the "plain" future — e.g., between "possibly tomorrow," and "necessarily tomorrow" and just "tomorrow." In all obvious models constructed in accordance with A. N. Prior's theory 4, "tomorrow" is conflated either with "possibly tomorrow" or with "necessarily tomorrow." On the basis of theory 3, there is no difficulty in maintaining a difference between the three kinds of notions discussed. In a theory 3 model, one can refer not only to what happens in some possible future, $\Diamond Fq$, and to what happens in all possible futures, $\Box Fq$, but one can also refer to what is going to happen in the future, Fq, as something different from the possible as well as the necessary future. A branching time model with this property is said to be Ockhamistic, whereas a branching time model in which Fq is identified with $\Box Fq$ is said to be Peircean. Graphically, the two kinds of branching time models can be presented as in Figs. 1 and 2 respectively.



Time in Philosophical Logic. Figure 1. An Ockhamistic model of branching time. At every branching point there will be one possible future which is the true future.



Time in Philosophical Logic. Figure 2. A Peircean model of branching time. There is no difference between the status of the possible futures at any branching point.

Punctual vs. Durational Time

The notion of a "duration" is important within the study of time. Several logicians have tried to formulate a logic of durations. The medieval logician John Buridan (ca. 1295–1358) regarded the present as a duration and not as a point in time. One example which he considered was the sentence: "If a thing is moving, then it was moving." In his analysis Buridan suggested that the logic of tenses can be established in two different ways based on the durational structure of time. Either the tenses can be taken absolutely, in the sense that no part of the present time is said to be past or future. Or the tenses can be taken in the relative sense, according to which "the earlier part of the present time is called past with respect to the later, and the later part is called future with respect to the earlier." Buridan pointed out that if a thing is moving now, then there is a part of the present during which it is moving, and hence, it is moving in some part of the present, which is earlier than some other part of the present. Therefore, if the thing is moving, then it was moving (if the past is taken in the relative sense), i.e., $\text{moving}(x) \Rightarrow P(\text{moving}(x))$. For this reason, the above sentence must be accepted if the past is understood relatively, whereas it has to be rejected if the past tense is understood absolutely. The reason is that one could in principle imagine a beginning of a process of motion. (Details can be found in [8, 43 ff.].)

The first modern logician to formulate a kind of durational calculus was Walker [15]. Walker suggested a model according to which time is considered as a structure $(S, <)$, where S is a non-empty set of periods (also called "durations" or "intervals"). The "$a < b$"-relation is to be considered as "strict" in the sense that no overlap between a and b is permitted, and the ordering is assumed to be irreflexive, asymmetrical, and transitive. In addition, he considered the notion of overlap, which can be defined as:

$$a|b \equiv_{def} \neg(a < b \vee b < a)$$

Walker formulated an axiomatic system using the following two axioms:

Definition

(W1)$a|a$

(W2)$(a < b \wedge b|c \wedge c < d) \Rightarrow a < d$

Using a set-theoretic method, Walker demonstrated that it is possible to define instants in terms of durations, thus making it possible to view a temporal instant as a "secondary" construct from the logic of durations.

In 1972 Charles Hamblin [6] independently also put forth a theory of the logic of durations. He achieved his results using a different technique involving the relation:

$$a \text{ meets } b \equiv_{def} a < b \wedge \neg(\exists c : a < c \wedge c < b)$$

A decade later, James Allen [1], in part together with Patrick Hayes [2], showed that two arbitrary durations (in linear time) can be related in exactly 13 ways. It has been shown that all these durational theories are equivalent when seen from an ontological point of view. They all show that just as durations (temporal intervals) can be set-theoretically constructed from an instant-logic, it is also possible to construct instants mathematically from durations. In fact, all the durational theories put forth so far appear to give rise to the same ontological model.

The theories formulated by Walker, Hamblin, and Allen can all be said to be B-theoretical. But Buridan's studies already suggested that it is possible to take an A-theoretical approach to durational logic. In modern durational logic an idea similar to Buridan's absolute/relative distinction was introduced in 1980 by Peter Röper [13] and others (see [8, 312 ff]).

Time and Narratives

A narrative is a text which presupposes a kind of event structure, i.e., a story. The temporal order of the story is often called “told time.” In many cases the story can be represented as a linear sequence of events. However, even if the event structure of the system is linear, the discourse structure can be rather complicated, since the reader (user) can in principle be given access to the events in any order. The order in which the events are presented is often referred to as “telling time.” Keisuke Ohtsura and William F. Brewer [10] have studied some interesting aspects regarding the relation between the event structure (told time) and the discourse structure (telling time) of a narrative text.

Key Applications

The philosophy of time has typically been carried out for its own sake. In many cases philosophers and logicians have seen the study of time as intimately related to essential aspects of human existence as such. For this reason, the study of time within philosophical logic has been motivated by a fundamental interest in the concepts dealing with time themselves and not by the search for a possible application. Nevertheless, such fundamental studies of time have turned out to give rise to theories and models which are useful in many ways. For instance, A. N. Prior’s analysis of the systematic relation between the dynamic and the static approach to time led him to the invention of what is now called hybrid logic (http://hylo.loria.fr). In general, temporal logic has turned out to be very useful in artificial intelligence and in other parts of computer science.

A. N. Prior’s tense logic seems especially relevant for a proper description of the use of interactive systems. A description of such systems from a B-logical point of view alone cannot be satisfactory, since that would ignore the user’s “nowness” which is essential in relation to the user’s choices and thus to the very concept of interactivity. On the other hand, if a conceptual start is made from A. N. Prior’s tense logic (i.e., the A-logical point of view), all B-logical notions can be defined in terms of the A-language.

The need for an A-logical description becomes even clearer when turning to a temporal analysis of systems which are non-linear even from a B-logical perspective, for instance a game-like multimedia system. In her studies of narratives and possible-world semantics, Marie-Laure Ryan [14] has made it clear that such a system is not to be viewed as a static representation of a specific state of affairs. Rather, it contains many different narrative lines which thread together many different states of affairs. Thus it is the choices of the user which will send the history in case on its specific trajectory.

Cross-references

- ▶ Allen’s Relations
- ▶ Now in Temporal Databases
- ▶ Qualitative Temporal Reasoning
- ▶ Temporal Database
- ▶ Temporal Granularity
- ▶ Temporal Logic in Database Query Languages
- ▶ Temporal Logical Models
- ▶ Temporal Object-Oriented Databases
- ▶ Time Domain

Recommended Reading

1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 26:832–843, 1983.
2. Allen J.F. and Hayes J.P. A common-sense theory of time. In Proc. 9th Int. Joint Conf. on AI, 1985, pp. 528–531.
3. Copeland J. Meredith, Prior, and the history of possible world semantics. Synthese, 150(3):373–397, 2006.
4. Fraser J.T., Haber F.C., and Müller G.H. (eds.). The Study of Time, Vol. I. Springer, Berlin 1972.
5. Gale R., (ed.). The Philosophy of Time. Prometheus Books, New Jersey, 1968.
6. Hamblin C.L. Instants and intervals. In J.T. Fraser, F.C. Haber, G.H. Müller (eds.). The Study of Time, Vol. I. Springer, Berlin 1972, pp. 324–331.
7. Hasle P. and Øhstrøm P. Foundations of Temporal Logic – the WWW-site for Prior-studies. http://www.prior.aau.dk.
8. Øhrstrøm P. and Hasle P. Temporal Logic. From Ancient Ideas to Artificial Intelligence. Kluwer Academic, Dordrecht, 1995.
9. Øhrstrøm P. and Hasle P. The flow of time into logic and computer science. Bull. Eur. Assn. Theor. Comput. Sci., (82):191–226, 2004.
10. Ohtsuka K. and Brewer W.F. Discourse organization in the comprehension of temporal order in narrative texts. Discourse Processes, 15:317–336, 1992.
11. Prior A.N. Past, Present and Future. Oxford University Press, Oxford, 1967.
12. Prior A.N. Papers on Time and Tense, 2nd edn. Oxford University Press, Oxford, 2002.
13. Röper P. Intervals and tenses. J. Phil. Logic, 9:451–469, 1980.
14. Ryan M.-L. Possible Worlds, Artificial Intelligence, and Narrative Theory. Indiana University Press, 1991.
15. Walker A.G. Durées et instants. La Revue Scientifique, (3266):131 ff., 1947.
16. Whitrow G.J. Reflections on the concept of time. In The Study of Time, Vol. I. J.T. Fraser, F.C. Haber, G.H. Müller (eds.). Springer, Berlin, 1972, pp. 1–11.
17. Whitrow G.J. The Natural Philosophy of Time, 2nd edn. Oxford University Press, Oxford, 1980.

Time Instant

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Event; Moment; Time point

Definition

A time *instant* is a single, atomic time point in the time domain.

Key Points

Various models of time have been proposed in the philosophical and logical literature of time. These view time, among other things, as discrete, dense, or continuous.

Instants in the dense model of time are isomorphic to the rational numbers: between any two instants there is always another. Continuous models of time are isomorphic to the real numbers, i.e., they are dense and also, unlike the rational numbers, without "gaps."

A discrete time domain is isomorphic to (a possibly bounded subset of) the natural numbers, and a specific instant of such a domain then corresponds to some natural number.

The elements of a discrete time domain are often associated with some fixed duration. For example, a time domain can be used where the time elements are specific seconds. Such time elements are often called *chronons*. In this way, a discrete time domain can approximate a dense or continuous time domain.

A time domain may be constructed from another time domain by mapping its elements to granules. In this case, multiple instants belong to the same granule, and the same granule may therefore represent different instants. For example, given a time domain of seconds, a time domain of day-long granules can be constructed.

Concerning the synonyms, the term "event" is already used widely within temporal databases, but is often given a different meaning, while the term "moment" may be confused with the distinct terms "chronon" or "granule."

Cross-references

- ▶ Chronon
- ▶ Event
- ▶ Temporal Database
- ▶ Temporal Domain
- ▶ Time Domain
- ▶ Time Granularity
- ▶ Time in Philosophical Logic

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer, Berlin, 1998, pp. 406–413.
2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin, 1998, pp. 367–405.

Time Interval

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Duration; Span; Time distance; Time period

Definition

Definition 1:

A *time interval* is a convex subset of the time domain. A time interval may be open or closed (at either end) and can be defined unambiguously by its two delimiting time instants. In a system that models the time domain using granules, an interval may be represented by a set of contiguous granules.

Definition 2:

An *interval* is a directed duration of time. A duration is an amount of time with known length, but no specific starting or ending instants. For example, the duration "1 week" is known to have a length of 7 days, but can refer to any block of seven consecutive days. An interval is either positive, denoting forward motion of time, or negative, denoting backwards motion in time.

Key Points

Unfortunately, the term "time interval" is being used in the literature with two distinct meanings: as the time between two instants, in the general database research literature and beyond, and as a directed duration of time, in the SQL database language. The term "time period" is associated with the first definition above.

Definition 1 is recommended for non-SQL-related scientific work. Definition 2 is recommended for SQL-related work.

Concerning the synonyms, the unambiguous term "span" has been used previously in the research literature, but its use seems to be less widespread than "interval." While precise, the term "time distance" is also less commonly used. A "duration" is generally considered to be non-directional, i.e., always positive.

Cross-references

- ▶ Temporal Database
- ▶ Temporal Granularity
- ▶ Time Domain
- ▶ Time Instant
- ▶ Time Period
- ▶ Time Span

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer, Berlin, 1998, pp. 406–413.
2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin, 1998, pp. 367–405.
3. Lorentzos N.A. and Mitsopoulos Y.G. SQL extension for interval data. IEEE Trans. Knowl. Data Eng., 9(3):480–499, 1997.

Time Period

Nikos A. Lorentzos
Agricultural University of Athens, Athens, Greece

Synonyms

Time interval

Definition

If the *time domain* is a totally ordered set T = {t_1, t_2, t_3...} then *a time period* over T is defined as a convex subset of elements from T.

Example: If T = {d_1, d_2, d_3,...}, where d_i are consecutive dates, then [d_{10}, d_{20}] and [d_{30}, d_{80}] represent two time periods over T.

Key Points

In the area of *temporal databases*, a time period over T is usually defined as a distinct data type. Some researchers define a time period data type of the form [t_p, t_q]. Some others define such a data type of the form [t_p, t_q), i.e., its right end is closed.

Note that, initially, the term *time interval* was used instead of *time period*. This was later abandoned in order to avoid confusion, given that *interval* is a reserved word in SQL. Instead, time interval in today used with a different meaning (see *time interval*).

Cross-references

- ▶ Absolute Time
- ▶ Chronon
- ▶ Period-Stamped Temporal Models
- ▶ Time Domain
- ▶ Time Interval
- ▶ Temporal Granularity

Time Period Set

- ▶ Temporal Element

Time Point

- ▶ Time Instant

Time Quantum

- ▶ Chronon

Time Sequence

- ▶ History in Temporal Databases

Time Sequence Query

- ▶ Time Series Query

Time Sequence Search

► Time Series Query

Time Series

► History in Temporal Databases

Time Series Data Mining

► Temporal Data Mining

Time Series Database Querying

► Query by Humming

Time Series Query

Like Gao[1], X. Sean Wang[2]
[1]Teradata Corporation, San Diego, CA, USA
[2]University of Vermont, Burlington, VT, USA

Synonyms

Time sequence query; Time series search; Time sequence search

Definition

A time series query refers to one that finds, from a set of time series, the time series or subseries that satisfy a given search criteria. *Time series* are sequences of data points spaced at strictly increasing times. The search criteria are domain specific rules defined with time series statistics or models, temporal dependencies, similarity between time series or patterns, etc. In particular, similarity queries are of great importance for many real world applications like stock analysis, weather forecasting, network traffic monitoring, etc., which often involve high volumes of time series data and may use different similarity measures or pattern descriptions. In many cases, query processing consists of evaluating these queries in real-time or quasi-real time by using time series approximation techniques, indexing methods, incremental computation, and specialized searching strategies.

Historical Background

Time series queries play a key role in temporal data mining applications such as time series analysis and forecasting. It was in the recent years that these applications with massive time series data became possible due to the rapidly emerging query processing techniques, especially those for similarity queries. In 1993, Rakesh Agrawal et al. [1] proposed an indexing method for processing similarity queries in sequence databases. The key was to map time series to a lower dimensionality space by only using the first few Fourier coefficients of the time series, and building R*-trees to index the time series. This work laid out a general approach of using indexes to answer similarity queries with time series. In 1994, Christos Faloutsos et al. [6] extended this work to subsequence matching and proposed the GEMINI framework for indexing time series. In 1997, Davood Rafiei and Alberto Mendelzon [12] proposed a set of linear transformations on the Fourier series representation of a time series that can be used as the basis for similarity queries. Subsequent works have focused on new dimensionality reduction techniques [2,9,10], new similarity measures [4,5,14], and queries over streaming time series [3,7,15].

Foundations

Basic Concepts

A *time series* x is a sequence of data points spaced at strictly increasing times. The number of elements in the sequence is called the *length* of the time series. The data points are often real numbers, and therefore x can often be represented as a vector of real numbers, $x(n) = \langle x_{t_1},\ldots,x_{t_n}\rangle$, where n is the length of x, and each t_i is the timestamp associated with the data point with $t_i < t_{i+1}$ for $i = 1,\ldots,n-1$. The time intervals between successive data points are usually, but not always, assumed uniform, and hence $t_{i+1} - t_i$ is often assumed a constant for $i = 1,\ldots,n - 1$. When the specific t_i values are irrelevant but only signify the temporal order, x is also represented as $x(n) = \langle x_1,\ldots, x_n\rangle$. A subsequence of x that consists of consecutive elements is called a *subseries* or *segment* of *x*. Time series normally refer to those finite sequences of *static* elements. If the sequence has new elements continuously appended over time, they are specially called *streaming* time series [7].

Raw time series often need pre-processing to fit the application needs. It is possible to perform the following pre-processing to remove irregularities of time series. *Time regulation* is to convert a non-uniform time series to a uniform one with interpolation functions to obtain the elements at uniform time intervals. *Normalization* is to make the distance between two time series invariant to offset shifting and/or amplitude scaling. For example, given time series *x*, its mean $\bar{x}$ and standard deviation σ_x, the normalization function can be either $\tilde{x} = (x - \bar{x})$ or $\tilde{x} = (x - \bar{x})/\sigma_x$. *Linear trend reduction* is to remove the seasonal trend impact on time series, e.g., $\tilde{x}_{t_i} = x_{t_i} - (a{*}t_i + b)$ for all *i*. To reduce data noise, *smoothing techniques* such as moving average can also be applied.

Similarity is the degree of resemblance between two time series, and the choice of similarity measure is highly domain dependent. For applications without value scaling and time shifting concerns, a simple L_p-norm Distance, of which L_1 and L_2 are the well known Manhattan and Euclidean Distances, respectively, is sufficient. For other applications, more robust similarity measures may be needed, such as scale invariant distances (such as correlation), warping distances that allow an elastic time shifting (such as DTW or Dynamic Time Warping, Longest Common Subsequence and Spatial Assembling Distances [3]), Edit Distance With Real Penalty (ERP) [4], and model based and histogram based distances. Since most similarity measures are defined non-negative, and the smaller the values, the closer the time series, the notions of similarity and distance are often used interchangeably.

Example 1 (L_p-`norm Distances, a.k.a. Minkowski Distance`): Given time series $x(n)$ and $y(n)$, and positive integer *p*, let

$$L_p(x, y) = \sqrt[p]{\sum\nolimits_{i=1}^{n} |x_i - y_i|^p}\ .$$

A special case is given as $L_\infty(x, y) = \max\{|x_i - y_i|, i = 1,...,n\}$. Note when $p \to \infty$, $L_p(x, y) = L_\infty(x, y)$.

Example 2 (`DTW Distance`): Given time series $x(m)$ and $y(n)$, then recursively, for all $1 \le i \le m$ and $1 \le j \le n$

$$\begin{aligned} DTW(x(i), y(j)) = d(x_i, y_j) + \min\{&DTW(x(i-1),\\ &y(j-1)), DTW(x(i-1), y(j)),\\ &DTW(x(i), y(j-1))\}, \end{aligned}$$

where $d()$ is a distance function defined on two elements x_i and y_j, and $x(i)$ and $y(j)$ denote the prefixes of the time series $x(m)$ and $y(n)$ of lengths *i* and *j*, respectively. The base case of the above is when $i = j = 1$ in which case $DTW(x(1),y(1)) = d(x_1, y_1)$. When both *i* and *j* are 0, $DTW(x(0),y(0))$ is defined as 0. Otherwise, when either *i* or *j* is out of range, $DTW(x(i),y(j))$ is defined as $+\infty$. DTW is usually accompanied with one of the following global constraints, in regard to the two prefix lengths *i* and *j*.

`Sakoe-Chiba Band`: The allowed range of *i* and *j* in the definition above satisfies $|j - i| \le r$ for some $r \ge 0$.

`Itakura Parallelogram`: Use the constraint $g(i) \le j - i \le f(i)$ for *i* and *j*, where *f* and *g* are functions of *i* such that the allowed region for $j–i$ given by the constraint shows a parallelogram shape with two opposing corners at (0,0) and (m,n), respectively.

Example 3 (`Edit Distance With Real Penalty` [4]): Given time series $x(m)$ and $y(n)$, recursively for all $i \le m$ and $j \le n$, let

$$ERP(x(i), y(j)) = \min\begin{cases} ERP(x(i-1), y(j-1)) + d(x_i, y_j)\\ ERP(x(i-1), y(j)) + d(x_i, g)\\ ERP(x(i), y(j-1)) + d(g, y_j) \end{cases}$$

In the above, *g* is a constant value (and can be 0), $x(i) = y(j) = \langle g \rangle$ (i.e., a time series of length 1) is assumed for all $i \le 0$ and $j \le 0$, and $d(a, b) = |a - b|$. The base case of the above is when both argument time series are of length 1 in which case $ERP(x, y) = d(x_1, y_1)$. Intuitively, the constant *g* is used to fill a *gap* referring to an added element.

Time Series Query

Many forms of time series queries have been proposed over the years. Among them, one of the mostly used is *similarity search*, defined as follows: Given a set of candidate time series *X*, a similarity measure *D*, and a query series *y*, (i) find all the series in *X* whose distances to *y* are within a given threshold (near neighbor query), and (ii) find *k* series in *X* that are closest to

y (k-nearest neighbor query). Other queries are also possible, e.g., all pairs query that finds, in X, all pairs of series with distance below a given threshold. Besides similarity search, other types of queries include detecting elastic burst over streaming time series, retrieving values at any arbitrary time [13], etc. In the following, time series query refers to similarity search.

The time series query can be either *whole series matching* or *subseries matching*. The former refers to the query that concerns the whole time series, both for the query series y and each candidate series x in X. The latter concerns the subseries of all x in X. For example, given time series $y(l)$, for each $x(n) \in X$, the latter query may need to consider $x(i+1,i+l) = \langle x_{i+1},\ldots,x_{i+l}\rangle$ for all $0 \leq i \leq n-l$.

In the above definition, if the query object is a set of *patterns*, the query is called *pattern matching*. A pattern is an abstract data model of time series, often seen as a short sequence, representing a class of time series that have the same properties. For pattern matching, all the involved time series may be mapped to the space in which the patterns and the similarity measure are defined.

Like the notion of time series, time series queries by default refer to those with static time series. In case of streaming time series, the queries are often monitoring the subseries within a sliding widows, and need to be evaluated periodically or continually to identify the similar series or those with the given patterns [7,15].

Query Processing: Index-based Methods for Similarity Search

Due to large volumes of data and the complexity of similarity measures, directly evaluating time series queries is both I/O and CPU intensive. There are many approaches to process these queries efficiently. Among them, one is to convert time series to other data types (e.g., strings and DNA sequences), so that the corresponding search techniques (e.g., string matching and DNA sequence matching) can be applied. Another approach is to index time series based on their approximations (or *features*), which is detailed in the following.

Time series x of length n can be viewed as a point in an n-dimensional space. However, spatial access methods such as *kd*-tree and *R*-tree cannot be used to index the time series directly. The problem is due to the *dimensionality curse* – the performance of spatial access methods degrades rapidly once the dimensionality is above 16, while n is normally much larger than this number.

The general solution is to map time series to points in a lower N-dimensional space ($N << n$) and then construct the indexes in this space. The mapped points are called the time series *approximation*. Each dimension of the N-dimensional space represents one characteristic of the time series, such as mean, variance, slope, peak values, or a Fourier coefficient, at some coarse levels of time granularity and possibly within a shifted time window. Further, the domain of the N-dimensional space can be nominal so the time series approximation can be represented as a symbolic sequence [11].

Example 4 (Piecewise Aggregate/Constant Approximation [14,8]): Time series x of length mN can be mapped to a point in the N-dimensional space: $\bar{x} = (\bar{x}_1,\ldots,\bar{x}_N)$ where the value in the *ith* dimension is the mean over the *ith* segment of x, $\bar{x}_i = \frac{1}{m}\sum_{j=m(i-1)+1}^{mi} x_j$.

Example 5 (Line Fitting Approximation [11]): Given an alphabet A{"*up*", "*down*", "*flat*"}, define a mapping function $s(z) \in \mathcal{A}$ where z is a time-series of length m. Time series x of length mN can be mapped to a length-N symbolic sequence, $\langle s(x_1,\ldots,x_m), s(x_{m+1},\ldots,x_{2m}),\ldots, s(x_{(N-1)m+1},\ldots,x_{mN})\rangle$, e.g., ⟨"*up*", "*up*", ...,"*down*"⟩. Function s can be line fitting and, based on the slope of the fitting line, decides if the value is "*up*" or "*down*" etc.

A multi-step algorithm can be used to process a query. Take the k-nearest neighbor search as an example. First step: find k-nearest neighbors in the lower-dimensional index structure. Find the actual distance between the query series and the *kth* nearest neighbor. Second step: use this actual distance as a range query to find (in the lower-dimensional index) all the database points. Third step: calculate the actual distances found in step 2 and obtain the actual k-nearest neighbors. An improvement to this algorithm is to incrementally obtain nearest neighbors in the lower-dimensional approximation, and each time an actual distance is obtained, it is used to remove some database points that are returned by the range query (second step above).

The index-based methods need to guarantee soundness (no false alarms) and completeness (no false dismissals) in the query result. Soundness can be guaranteed by checking the original data as in step 3. The completeness can be guaranteed only if the chosen approximation method has the *lower-bounding property* [6]. That is, given a similarity measure D, for any candidate time series x and query time series y, let $\bar{x}$ and $\bar{y}$ be their lower-dimensional approximations and $D^{'}$ be the distance defined on $\bar{x}$ and $\bar{y}$, then $D^{'}(\bar{y}, \bar{x}) \leq D(y, x)$ must hold for any x and y.

Example 6 (Lower Bounding Approximation for Euclidian Distance): Method (1): apply an orthonormal transform (Fourier transform, Wavelet transform, and SVD) to both query and candidate time series and ignore many "insignificant" axes after the transform. The distance defined on the remaining axes gives the lower bounding approximation for Euclidian Distance [1]. Method (2): apply segmented mean approximation to both query and candidate time series. It is easy to see $\sqrt[p]{m}L_p(\bar{x}, \bar{y}) \leq L_p(x, y)$ for all p, while m is the factor of dimensionality reduction, i.e., x's length divided by $\bar{x}$'s. Since this lower-bounding approximation works for all p, one index tree can be used for all L_p-norm distances [8,14].

Example 7 (Lower Bounding Approximation for DTW Distance [9]): To derive a lower bounding approximation for DTW, approximate the candidate time series x using segmented mean approximation, and approximate the query time series y as follows. Let $y = \langle y_1,\ldots,y_{mN}\rangle$. Define $U = \langle U_1,\ldots,U_{mN}\rangle$ and $L = \langle L_1,\ldots,L_{mN}\rangle$ where $U_i = \max(y_{i-r},\ldots,y_{i+r})$ and $L_i = \min(y_{i-r},\ldots,y_{i+r})$ (r is the allowed range for $m - n$ in Sakoe-Chiba Band or Itakura Parallelogram). Sequences U and L form a bounding envelope that encloses y from above and below. Then reduce U and L to a lower dimension N, define $\hat{U} = \langle \hat{U}_1,\ldots,\hat{U}_N\rangle$ and $\hat{L} = \langle \hat{L}_1,\ldots,\hat{L}_N\rangle$, where $\hat{U}_i = \max(U_{(i-1)m+1},\ldots,U_{im})$ and $\hat{L}_i = \min(L_{(i-1)m},\ldots,L_{im})$, that is, $\hat{U}$ and $\hat{L}$ are piecewise constant functions that bound U and L, respectively. Let

$$LB\_PAA(y, \bar{x}) = \sqrt{m\sum_{i=1}^{N}\begin{cases}(\bar{x}_i - \hat{U}_i)^2 & \text{if } \bar{x}_i > \hat{U}_i;\\ (\bar{x}_i - \hat{L}_i)^2 & \text{if } \bar{x}_i < \hat{L}_i;\\ 0 & \text{otherwise.}\end{cases}},$$

then $LB\_PAA(y, \bar{x}) \leq DTW(y, x)$.

Query Processing: Similarity Search over Streaming Time Series

These queries are different from those with static time series, in that (i) having a sliding window or windows of multiple lengths at the same time; (ii) continuous monitoring; and (iii) incremental evaluation. In the following, consider the two problems: Euclidean distance or correlation monitoring among pairs of streams, and Euclidean distance or correlation monitoring between a stream time series and a time series database.

The first query problem is, given many streaming time series, how to find pairs of time series that have strong (positive or negative) correlations in the last sliding window [15].

The idea is to use the notion of "basic windows," similar to segmented mean application. Instead of mean, the coefficients of the Fourier transform of each segment is used to approximate the time series. Given $x = \langle x_1,\ldots,x_b\rangle$ and $y = \langle y_1,\ldots,y_b\rangle$, where b is the size of the basic window. If $x_i = \sum_{m=0}^{N-1} C_m^x f_m(i)$ and $y_i = \sum_{m=0}^{N-1} C_m^y f_m(i)$ is a family of orthogonal functions, then the inner product of x and y,

$$\begin{aligned} x{*}y &= \sum_{i=1}^{b} x_i y_i \\ &= \sum_{i=1}^{b}(\sum_{m=0}^{N-1} C_m^x f_m(i) \sum_{p=0}^{N-1} C_p^y f_p(i)) \\ &= \sum_{m=0}^{N-1}\sum_{p=0}^{N-1} C_m^x C_p^y (\sum_{i=1}^{b} f_m(i) f_p(i)). \end{aligned}$$

Note $\sum_{i=1}^{b} f_m(i)f_p(i)$ does not depend on x and y and can be pre-computed. From this, the inner product of two time series can be computed for each sliding window aligned with the basic windows (i.e., a sliding window must be the union of some basic windows). Fourier bases can be used as the f functions, and discrete Fourier transform (DFT) can compute the coefficients efficiently in an incremental way.

By only taking a few Fourier coefficients (small N), the approximate inner products and hence the Euclidean distance can be evaluated efficiently. To compute correlations, the normalized series of $\hat{x}_i = (x_i - \bar{x})/\sigma_x$ need only be considered, where $\bar{x}$ and σ_x are the mean and standard deviation of x over the sliding window.

A step further: since two series are highly correlated if their DFT coefficients are similar, an indexing structure can help to store the coefficients and look for series with high correlation only in series with similar coefficients.

The second query problem is, given a database of pattern time series, a streaming time series, and window size N, how to find the nearest neighbor of the streaming time series (using the last N values) in the database, at each time position [7].

The idea is a batch processing that uses fast Fourier transform (FFT) and its inverse to calculate the cross correlation of streaming time series and patterns at many time positions. Given $x = \langle x_1,\ldots,x_N \rangle$ and $y = \langle y_1,\ldots,y_N \rangle$, the circular cross correlation sequence is defined as

$$CirCCorr_d^{x,y} = \sum_{i=1}^{N} x_{(d+i-1)\ mod\, \mathrm{N}} y_i, \quad d = 1, 2, \ldots, N,$$

where d is the time lag. Let $\dot{x}$ and $\dot{y}$ be the DFT transforms of x and y respectively, then sequence $\langle \dot{x}_1 \dot{y}^*{}_1,\ldots, \dot{x}_N\, \dot{y}_N^* \rangle$ is the result of DFT transform of $CirCCorr^{x,y}$. Here $\dot{y}^*{}_i$ is the conjugate of $\dot{y}_i$.

With the CirCCorr, calculation of the Euclidean distances of a number of time positions can be done in a batch mode. This is faster than calculating the individual distances, as the batch process has time complexity $O(N\lg N)$, as compared to the direct computing of $O(Nl)$, where l ($l < N$) is the number of time positions covered by one batch processing. So it is profitable to wait for a few time steps and then find the nearest neighbors for these time steps all together. The longer the wait is, the more computation time saved. However, this causes a lengthening of the response time, i.e., a loss of the chance of finding the answer as early as possible. To overcome this, one may use a certain model to roughly predict the future values of the time series and apply the batch processing to compute all the Euclidean distance (or correlations) of many future time positions. When the actual values come, triangular inequality can filter out a lot of time series in the database that are not the nearest neighbor [7].

Key Applications

Market data analysis and trend predication, network traffic control, intrusion detection, temporal data mining.

Data Sets

1. UCR Time Series Classification/Clustering Page: http://www.cs.ucr.edu/~eamonn/time_series_data/
2. PhysioBank: Physiologic Signal Archives for Biomedical Research (including ECG and synthetic time series with known characteristics): http://www.physionet.org/physiobank/

URL to Code

1. Above URL 1.
2. ANN: A Library for Approximate Nearest Neighbor Searching: http://www.cs.umd.edu/~mount/ANN/

Cross-references

- ▶ Curse of Dimensionality
- ▶ Dimensionality Reduction
- ▶ Discrete Wavelet Transform and Wavelet Synopses
- ▶ High Dimensional Indexing
- ▶ Indexing and Similarity Search
- ▶ Nearest Neighbor Query
- ▶ Range Query
- ▶ R-tree (and Family)
- ▶ Sequential patterns
- ▶ Singular Value Decomposition
- ▶ Stream Similarity Mining
- ▶ Temporal data mining
- ▶ Top-K Selection Queries on Multimedia Datasets

Recommended Reading

1. Agrawal R., Faloutsos C., and Swami A.N. Efficient similarity search in sequence databases. In Proc. 4th Int. Conf. on Foundations of Data Organization and Algorithms, 1993, pp. 69–84.
2. Chan K.P. and Fu A.W.-C. Efficient time series matching by wavelets. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 126–133.
3. Chen Y., Nascimento M.A., Ooi B.C., and Tung A.K.H. Spade: on shape-based pattern detection in streaming time series. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 786–795.
4. Chen L. and Ng R. On the marriage of lp-norms and edit distance. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 792–803.
5. Das G., Gunopulos D., and Mannila H. Finding similar time series. In Principles of Data Mining and Knowledge Discovery, 1st European Symp., 1997, pp. 88–100.
6. Faloutsos C., Ranganathan M., and Manolopoulos Y. Fast subsequence matching in time-series databases, In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 419–429.
7. Gao L. and Wang X.S. Continuous similarity-based queries on streaming time series. IEEE Trans. Knowl. Data Eng., 17(10):1320–1332, 2005.
8. Keogh E.J. and Pazzani M.J. Scaling up dynamic time warping for datamining applications. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 285–289.
9. Keogh E.J. and (Ann) Ratanamahatana C. Exact indexing of dynamic time warping. Knowl. Inform. Syst., 7(3):358–386, 2005.
10. Korn F., Jagadish H.V., and Faloutsos C. Efficiently supporting ad hoc queries in large datasets of time sequences. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 289–300.

11. Qu Y., Wang C., Gao L., and Wang X.S. Supporting movement pattern queries in user-specified scales. IEEE Trans. Knowl. Data Eng., 15(1):26–42, 2003.
12. Rafiei D. and Mendelzon A. Similarity-based queries for time series data. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1997, pp. 13–25.
13. Revesz P., Chen R., and Ouyang M. Approximate query evaluation using linear constraint databases. In Proc. 8th Int. Symp. Temporal Representation and Reasoning, 2001, pp. 170–175.
14. Yi B.-K. and Faloutsos C. Fast time sequence indexing for arbitrary LP norms. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 385–394.
15. Zhu Y. and Shasha D. Statstream: statistical monitoring of thousands of data streams in real time. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 358–369.

Time Series Search

▶ Time Series Query

Time Span

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tuscon, AZ, USA

Synonyms

Time interval; Time distance

Definition

A span is a directed duration of time. A duration is an amount of time with known length, but no specific starting or ending instants. For example, the duration "1 week" is known to have a length of 7 days, but can refer to any block of seven consecutive days. A span is either positive, denoting forward motion of time, or negative, denoting backwards motion in time.

Key Points

Concerning the synonyms, the terms "time interval" is generally understood to denote an anchored span in the general community of computer science. Only in the SQL language does "time interval" denote a span. The term "span," which has only one definition, is thus recommended over "time interval" for works not related to the SQL language. This use is unambiguous.

A "duration" is generally considered to be non-directional, i.e., always positive. The term "time distance" is precise, but is longer.

Cross-references

▶ Fixed Span
▶ Temporal Database
▶ Time Instant
▶ Time Interval
▶ Variable Span

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, and S. Sripada (eds.). LNCS 1399, Springer, Berlin, 1998, pp. 406–413.
2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, and S. Sripada (eds.). LNCS 1399, Springer Verlag, Berlin, 1998, pp. 367–405.

Time Unit

▶ Chronon

Time-based Access Control

▶ Temporal Access Control

Time-based Window

▶ Windows

Time-Constrained Transaction Management

▶ Real-Time Transaction Processing

Time-Dependent Graphs

▶ Time Aggregated Graphs

Time-Dependent Networks

▶ Time Aggregated Graphs

Time-Line Clock

Curtis Dyreson
Utah State University, Logan, UT, USA

Synonyms

Clock; Base-line clock; Time-segment clock

Definition

In the discrete model of time, a *time-line clock* is defined as a set of *physical clocks* coupled with some specification of when each physical clock is authoritative. Each *chronon* in a time-line clock is a chronon (or a regular division of a chronon) in an identified, underlying physical clock. The time-line clock switches from one physical clock to the next at a *synchronization point.* A synchronization point correlates two, distinct physical clock measurements.

Key Points

A time-line clock is the clock for (concrete) times stored in a temporal database. A time-line clock glues together a sequence of physical clocks to provide a consistent, clear semantics for a time-line. Since the range of most physical clocks is limited, a time-line clock is usually composed of many physical clocks. For instance, a tree-ring clock can only be used to date past events, and the atomic clock can only be used to date events since the 1950s. Though several physical clocks might be needed to build a time-line, in some cases a single physical clock suffices. For instance SQL2 uses the mean solar day clock – the basis of the *Gregorian calendar* – as its time-line clock.

Cross-references

- ▶ Chronon
- ▶ Physical Clock
- ▶ Time Instant

Recommended Reading

1. Dyreson C.E. and Snodgrass R.T. Timestamp semantics and representation. Inf. Syst., 18(3):143–166, 1993.
2. Dyreson C.E. and Snodgrass R.T. The baseline clock. The TSQL2 Temporal Query Language. Kluwer, Norwell, MA, 1995, pp. 73–92.
3. Fraser J.T. Time: The Familiar Stranger. University of Massachusetts Press, Amherst, MA, 1987, p. 408.

Time-Oriented Database

▶ Temporal Database

Time-Segment Clock

▶ Time-Line Clock

Timeslice Operator

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Rollback operator; State query

Definition

The *valid-timeslice operator* may be applied to any temporal relation that captures valid time. Given also a valid-time element as a parameter, it returns the argument relation reduced in the valid-time dimension to just those time(s) specified by the valid-time element. The *transaction timeslice operator* is defined similarly, with the exception that the argument relation must capture transaction time.

Key Points

Several types of timeslice operators are possible. Some may restrict the time parameter to intervals or instants. Some operators may, given an instant parameter, return a conventional relation or a transaction-time relation when applied to a valid-time or a bitemporal relation, respectively; other operators may always return a result relation of the same type as the argument relation.

Oracle supports timeslicing through its flashback queries. Such queries can retrieve all the versions of a row between two transaction times (a key-transaction-time-range query) and allows tables and databases to be rolled back to a previous transaction time, discarding all changes after that time.

Concerning the synonyms, "rollback operator" is an early term that has since been abandoned. This term indicates that the result of a timeslice is a relation obtained by moving backwards in time, presumably from the current transaction time. This kind of result is less general than those that may be obtained using a timeslice operator. Specifically, this kind of result assumes a time parameter that extends from the beginning of the time domain to some past time (with respect to the current time). Similarly, “state query” suggests a less general functionality than what is actually offered by timeslice operators.

Cross-references

▶ Bitemporal Relation
▶ Temporal Database
▶ Temporal Element
▶ Temporal Query Languages
▶ Time Instant
▶ Time Interval
▶ Transaction Time
▶ TSQL2
▶ Valid Time

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin, 1998, pp. 367–405.

TIN

▶ Triangulated Irregular Networks

Tiny Aggregation (TAG)

▶ In-Network Query Processing

TinyDB

▶ In-Network Query Processing

TinySQL

▶ Database Languages for Sensor Networks

t-Norm

▶ Triangular Norms

Topic Detection and Tracking

Ning Liu
Microsoft Research Asia, Haidian, China

Definition

According to the definition at http://projects.ldc.upenn.edu/TDT/, Topic Detection and Tracking (TDT) is a multi-site research project to develop core technologies for a news understanding systems. Specifically, TDT systems discover the topical structure in unsegmented streams of news reporting as it appears across multiple media and in different languages. Some terms are defined below before the TDT problem is fully understood (The definitions are borrowed from Omid Dadgar’s work).

1. Event – An event is something that happens at some specific time and place, and the unavoidable consequences. Specific elections, accidents, crimes and natural disasters are examples of events.
2. Activity – An activity is a connected set of actions that have a common focus or purpose. Specific campaigns, investigations, and disaster relief efforts are examples of activities.
3. Story – A story is a newswire article or a segment of a news broadcast with a coherent news focus. They must contain at least two independent, declarative clauses.
4. Topic – The topic is defined as a seminal event or activity, along with all directly related events and activities.

With the definition of topic, the Topic Detection and Tracking can be known as to investigate the state of the art in finding and following new events in a stream of broadcast news stories. According to the Pilot-study, the original TDT problem consists of three major tasks: (i) segmenting a stream of data, especially recognized speech, into distinct stories; (ii) identifying those news stories that are the first to discuss a new event occurring in the news; and (iii) given a small number of sample news stories about an event, finding all following stories in the data stream.

Historical Background

The TDT study starts from 1996 through a Pilot-study [2] which aims to explore the approaches and performance baselines. After that, it quickly attracted much attention. Followed by the TDT2, TDT3 etc, increasing number of research works are focusing on the TDT problem. TDT2 in 1998 was the first major step in TDT after the pilot study since it established the foundation for the following works. It addresses the same three problems, which are segmentation, detection, and tracking with the original Pilot study. The evaluation procedures were modified and the volume and variety of data and the number of target topics were expanded. TDT2 attacked the problems introduced by imperfect, machine-generated transcripts of audio data. The TDT3 was used for the year 1999, 2000 and 2001 test. The TDT4 and TDT5 are used for the year 2002, 2003 and 2004 test respectively.

From the algorithms' perspective, start from the Pilot-study, many algorithms and applications about TDT have been proposed [1]. As some examples, in the Pilot-study, the Dragon approach, UMass approach and CMU approach were introduced for text segmentation. The same three approaches are used for new event detection. They are finally utilized for event tracking. As some recent progresses, Masaki et al. proposed the Topic Detection and Tracking for News Web Pages by cluster and SuffixTree [4]. He et al. proposed to conduct the topic detection and Tracking by topic sensitive language model [5]. Makkonen et al. proposed to utilizing the temporal information for topic detection and tracking. In the next section, the TDT problems is considered as several sub-problems. Some algorithms are summarized to address the sub-problems of TDT.

Foundations

According to the Pilot study [1], the TDT problem has the following several sub-problems (the major contents of this section is borrowed from the Pilot study).

The Segmentation Task

The segmentation task is defined to be the task of segmenting a continuous stream of text (including transcribed speech) into its constituent stories. To support this task the story texts from the study corpus will be concatenated and used as input to a segmenter. This concatenated text stream will include only the actual story texts and will exclude external and internal tag information. The segmentation task is to correctly locate the boundaries between adjacent stories, for all stories in the corpus.

The Detection Task

The detection task is characterized by the lack of knowledge of the event to be detected. In such a case, one may wish to retrospectively process a corpus of stories to identify the events discussed therein, or one may wish to identify new events as they occur, based on an on-line stream of stories. Both of these alternatives are supported under the detection task.

Retrospective Event Detection

The retrospective detection task is defined to be the task of identifying all of the events in a corpus of stories. Events are defined by their association with stories, and therefore the task is to group the stories in the study corpus into clusters, where each cluster represents an event and where the stories in the cluster discuss the event. It will be assumed that each story discusses at most one event. Therefore each story may be included in at most one cluster.

Online New Event Detection

The on-line new event detection task is defined to be the task of identifying new events in a stream of stories. Each story is processed in sequence, and a decision is made whether or not a new event is discussed in the story, after processing the story but before processing any subsequent stories). A decision is made after each story is processed. The first story to discuss an event should be flagged YES. If the story doesn't discuss any new events, then it should be flagged NO.

The Tracking Task

The tracking task is defined to be the task of associating incoming stories with events known to the system. An event is defined ("known") by its association with stories that discuss the event. Thus each target event is defined by a list of stories that discuss it.

To solve these sub-problems, various algorithms have been proposed. For the segmentation, as addressed by the Pilot study, there is a relatively small but varied body of previous work that has addressed the problem of text segmentation. This work includes methods based on semantic word networks, vector space techniques from information retrieval and decision tree induction algorithms. As for some classical algorithms, the Dragon's approach to segmentation is to treat a story as an instance of some underlying topic, and to model an unbroken text stream as an unlabeled sequence of these topics. In this model, finding story boundaries is equivalent to finding topic transitions. Given a text stream, a probability can be attached to any particular hypothesis about the sequence and segmentation of topics in the following way:

1. Transition from the start state to the first topic, accumulating a transition probability.
2. Stay in topic for a certain number of words or sentences, and, given the current topic, accumulate a selfloop probability and a language model probability for each.
3. Transition to a new topic, accumulating the transition probability. Go back to step 2.

A search for the best hypothesis and corresponding segmentation can be done using standard HMM techniques and standard speech recognition tricks.

After the segmentation work, Event *detection* is the problem of identifying stories in several continuous news streams that pertain to new or previously unidentified events. Using the same Dragon approach as example, Dragon's online and retrospective detection systems are applications of the clustering technology used to train background models for the segmenter. This technology is an implementation of a k-means clustering algorithm. The next step after event detection is the event tracking.

The TDT event tracking task is fundamentally similar to the standard routing and filtering tasks of Information Retrieval (IR). Given a few sample instances of stories describing an event (i.e., stories that provide a description of the event), the task is to identify any and all subsequent stories describing the same event. Event tracking is different from those IR tasks in that events rather than queries are tracked, and in that events have a temporal locality that more general queries lack. These differences shift the nature of the problem slightly but at the same time shift the possible solutions significantly. The narrowing of the scope of information filtering encourages modifications to existing approaches and invites entirely new approaches that were not feasible in a more general query centric setting.

Dragon's event tracker is an adaptation of its segmenter. As discussed there, the segmentation algorithm does segmentation and topic assignment simultaneously. In general, the topic labels assigned by the segmenter are not useful for classification, as they are few in number and do not necessarily correspond to categories a person would find interesting. However, by supplementing the background topic models with a language model for a specific event of interest, and allowing the segmenter to score segments against this model, it becomes possible for the segmenter to output a notification of an occurrence of that event in the news stream whenever it assigns that event model's label to a story. In this implementation, the topic models have the role of determining the background against which the event model must score sufficiently well to be identified.

Key Applications

TDT techniques have wide range of applications especially on the Web documents. As for the key applications, the major goal of TDT is to finding and following new events in a stream of broadcast news stories.

Future Directions

The future directions of TDT are in several fold. The first is to propose more effective algorithms for some classical problems such as monitoring streams of news in multiple languages (e.g., Mandarin) and media – newswire, radio, television, web sites or some future combination. On the other hand, due to the rapid growth of World Wide Web, the scale of the data for topic detection and tracking is getting larger and larger, thus more scalable algorithms are highly desired. Another direction for exploring is to find new online applications of the TDT problem.

Data Sets

The most commonly used corpus for TDT study start from the Pilot-study. And then the TDT2, TDT3, TDT2000, TDT2001, TDT4 and TDT5 were released. The details about the datasets, tasks and evaluation metrics can be found at http://projects.ldc.upenn.edu/TDT/. LDC is the provider of the corpus for the second phase of TDT and is currently developing the phase three corpus. As an example, some details about TDT2 are introduced.

Recommended Reading

1. Allan J. Topic Detection and Tracking. Kluwer, Norvell, MA, 2002.
2. Allan J., Carbonell J., Doddington G., Yamron J., and Yang Y. Topic detection and tracking pilot study final report. In Proc. DARPA Broadcast News Transcription and Understanding Workshop, 1998, pp. 194–218.
3. Makkonen J. and Ahonen-Myka H. Utilizing Temporal Expressions in Topic Detection and Tracking. In Proc. 7th European Conf. Research and Advanced Technology for Digital Libraries, 2003, pp. 393–404.
4. Mori M., Miura T., and Shioya I. Topic detection and tracking for news web pages. In Proc. 2006 IEEE/WIC/ACM Int. Conf. on Web Intelligence, 2006, pp. 338–342.
5. Ruifang H., Bing Q., Ting L., and Sheng L. The topic detection and tracking with topic sensitive language model. In Proc. Int. Conf. on Mutilingual Information Processing, 2005, pp. 324–327.

Topic Hierarchies

► Lightweight Ontologies

Topic Maps

James Caverlee
Texas A&M University, College Station, TX, USA

Definition

Topic Maps provide a standardized way to represent and interchange knowledge through the modeling of abstract concepts (called *topics*), the relationships among topics (called *associations*), and the connection between abstract concepts and real-world resources (called *occurrences*). By distinguishing the high-level topic space from real-world resources, Topic Maps may be used both as a semantic map among related concepts (in the topic space) and as a way to describe real-world resources (through occurrence mapping from the topic space into the resource space). Topic Maps have been formally standardized by the international standards body ISO.

Historical Background

The pre-cursors of what are now known as Topic Maps began in the early 1990s with an effort to merge independently created and maintained indexes for information sharing. This early work motivated the need for a more general and more useful knowledge description meta framework. By 1999, the original ISO standard for Topic Maps was published as ISO/IEC 13250 based primarily on SGML and the hypermedia linking language HyTime.

Foundations

Topic Maps support the modeling and exchange of knowledge based on a standardized framework centered around topics, occurrences, and associations [1,3,4]. These constructions serve as n-ary connections between items that express overarching semantics.

Using Topic Maps

According to the ISO standard, Topic Maps [2] are designed to facilitate knowledge representation and interchange by:

- Providing an abstract layer of topic-centered metadata over information resources for supporting navigational tools like indexes, glossaries, and citation systems.
- Linking topics in a clearly defined fashion so that users can navigate between them. Such linking can support thesaurus-like interfaces to disparate information stores.
- Supporting different "views" over a set of information resources by filtering information resources based on the metadata described in Topic Maps.
- Adding a structured layer over unstructured information resources in the form of a markup that is completely external to the original resources.

The Basics of Topic Maps

To represent and interchange knowledge in a standardized way, Topic Maps rely on three key concepts: (i) topics, which represent abstract concepts; (ii) associations, which model relationships among topics; and (iii) occurrences, which connect topics to real-world resources.

The most basic element in a topic map is a *topic*. In general, a *topic* is an abstract concept or subject of concern within the framework of Topic Maps. A topic can be used to represent people, places, events, organizations, Web pages, documents, or any other reasonable unit of interest. In an example universe of discourse, it may be appropriate to represent the author Jane Austen as a topic, as well as the concepts of author, person, novel, and so on. A topic may be associated with one or more names; in the running example, Jane Austen the topic may be referred to by multiple names, including Austen, Jane Austen, and jane-austen.

Using XTM, the topic for novel and the topic for Jane Austen (as an instance of a Writer) can be expressed as:

```
<topic id="novel">
   <baseName>
      <baseNameString>Novel
      </baseNameString>
   </baseName>
</topic>
```

```
<topic id="jane-austen">
   <instanceOf><topicRef xlink:href=
   "#writer"/></instanceOf>
   <subjectIdentity>
      <subjectIndicatorRef xlink:href=
      "{http://en.wikipedia.org/wiki/
      Jane_Austen}"/>
   <subjectIdentity>
   <baseName>
      <baseNameString>Austen, Jane 1775-
      1817</baseNameString>
   </baseName>
</topic>
```

Note that the Jane Austen topic includes a special *subject indicator* syntax that refers to the Wikipedia entry for Jane Austen. A subject indicator is a guideline for a human consumer of the topic map that, although Jane Austen the writer cannot be directly addressed by a URL, the Wikipedia article uniquely identifies her. In this way, the *subject indicator* serves as a *subject identifier* for the topic Jane Austen. By construction, two topics that have the same subject identifier must refer to the same abstract concept.

The second key component of Topic Maps is an *association*. An *association* is a relationship between topics in a topic map. For example, a topic for the author Jane Austen and a topic for the novel *Pride and Prejudice* (which was written by Jane Austen) could be linked by the association "written-by" which links an instance of an Author with an instance of a Novel. These instances serve as roles in the association. Since associations imply no directionality, the "written-by" association implicitly has a dual association "wrote." There are no limits on the number and nature of associations in a Topic Map, so the Topic Maps paradigm may be used to model complex and sophisticated domains, as well as simpler domains as in the Jane Austen example.

```
<association>
   <instanceOf><topicRef xlink:href="#
   written-by"/></instanceOf>
   <member>
      <roleSpec><topicRef xlink:
      href="#author"/> </roleSpec>
      <topicRef xlink:href="#jane_
      austen"/>
   </member>
   <member>
      <roleSpec><topicRef xlink:
      href="#novel"/> </roleSpec>
      <topicRef xlink:href="#pride_and_
   prejudice"/>
   </member>
</association>
```

Finally, an *occurrence* is a real representations of a topic. For example, a Web-accessible file of the book *Pride and Prejudice* is an occurrence of the topic of the same name. Similarly, the *Pride and Prejudice* topic may also occur in a scholarly article discussing the role of women in British literature that happens to mention *Pride and Prejudice*. In an XTM topic map, an occurrence must be a resource that is addressable using a Uniform Resource Identifier (URI) or may be placed inline as character data. Hence, an occurrence of Jane Austen could be an external resource like a Web page or an image, or a brief in-line description of the author.

In the following example, two of Jane Austen's works are referenced as occurrences:

```
<topic id="pride_and_prejudice">
   <instanceOf><topicRef xlink:href="#
   novel"/> </instanceOf>
      <baseName><baseNameString>Pride
      and Prejudice</baseNameString>
      </baseName>
```

```
  <occurrence>
    <instanceOf><topicRef  xlink:href=
    "#pdf-format"/></instanceOf>
    <resourceRef xlink:href="http://
    www.gutenberg.org/dirs/etext98/
    pandp12p2.pdf"/>
  </occurrence>
</topic>

<topic id="sense_and_sensibility">
  <instanceOf><topicRef xlink:href="
  #novel"/> </instanceOf>
  <baseName><baseNameString>Sense and
  Sensibility</baseNameString>
  </baseName>
  <occurrence>
    <instanceOf><topicRef xlink:href=
    "#pdf-format"/></instanceOf>
    <resourceRef xlink:href="http://
    www.gutenberg.org/dirs/etext94/
    sense11p.pdf"/>
  </occurrence>
</topic>
```

Extending the Basic Model

Topic Maps can be further refined through the use of *types* and *scope*.

A *type* is fundamentally a special kind of association between topics, used to indicate that one topic is an "instance-of" another topic. For example, since *Pride and Prejudice* is a book, there may also exist in the topic map an "instance-of" association between the topic *Pride and Prejudice* and a topic representing the concept of a book. A topic may have multiple types.

A *scope* provides additional contextual information about the elements of a Topic Map. Scope allows for the same topic map to exist at different levels of specification. Users of the topic maps can then decide if they are interested in things from any scope or only from one particular scope or subset of scopes. For example, scope can be used to provide localized names for topics – one name for English, one for Spanish, and one for French.

Merging Topic Maps

Since Topic Maps may be developed in a distributed or independent environment, one of the key features of Topic Maps is the notion of *merging*. Merging means that two topic maps with identical topics can have their associations and occurrences combined together to build a richer semantic model. In particular, two topics can be combined (or "merged") into a single topic containing the union of the types, the names, and the occurrences of the original two topics. This merged topic replaces the original two topics wherever they participate as a role in an association or serve as a topic type.

Using Topic Maps

The Topic Maps standard provides a reference point for the appropriate syntax and functionality of topic maps, leaving the implementation details of a topic maps processing engine to commercial and non-commercial applications and tools. Currently, there are topic map processing libraries in most popular languages like Java, C, Perl, and Python. In an effort to provide a uniform programmatic interface to topic maps, regardless of the particular language and platform, the Common Topic Map Application Programming Interface (TMAPI) has been recently developed. The TMAPI specification provides a base set of core interfaces for accessing and manipulating topic maps.

Key Applications

Web, Semantic Web, digital libraries, business-to-business exchange.

Cross-references

▶ Conceptual Schema Design
▶ RDF
▶ Semantic Web

Recommended Reading

1. Garshol L. Metadata? Thesauri? Taxonomies? Topic maps! Making sense of it all. J. Inf. Sci., 30(4):378–391, 2004.
2. International Organization for Standardization. ISO 13250-2003 Information technology – SBML applications – Topic maps. Available at: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38068
3. Park J. and Hunting S. (eds.). XML Topic Maps. Addison-Wesley, Boston, MA, USA, 2002.
4. Pepper S. The TAO of topic maps: finding the way in an age of infoglut. In Proc. XML Europe Conf., 2000.

Topical-Hierarchical Relevance

▶ Relevance

Topic-based Publish/Subscribe

Hans-Arno Jacobsen
University of Toronto, Toronto, ON, Canada

Synonyms

Subject-based publish/subscribe

Definition

Topic-based publish/subscribe is a communication abstraction that supports selective message dissemination among many sources and many sinks. Messages are associated with topics and are selectively routed to destinations with matching topic interests. Data sinks specify interest in receiving messages of a given topic and data sources publish messages on different topics. Topic-based publish/subscribe is an instance of the more general publish/subscribe concept.

Key Points

Topic-based publish/subscribe is an instance of the more general publish/subscribe concept. In the topic-based publish/subscribe model, a data source submits publication messages associated with a topic to the publish/subscribe system, while a data sink subscribes its interest in receiving messages of certain topics by submitting subscription expressions on available topics to the system. The kind of topics to publish or subscribe that exist is either out of band information and must be know to clients, or is dynamically discoverable by clients based on additional support provided by the system. For example, by subscribing to control channel topics, where the creation of new topics is announced. Topics are an integral part of the messages disseminated through the publish/subscribe system. The publish/subscribe system only knows how to interpret the topics, but not the rest of the publication message, which remains opaque to the system.

A publication message published to the topic-based publish/subscribe system is delivered to all subscribers with matching subscriptions. A subscription matches a publication if the topic associated with the publication message matches the subscription expression. In very simple realizations of this model, a topic is simply a string that represents a name, a subject, or a topic according to which messages are classified. In more sophisticated realizations, topics draw from a hierarchical topic space. The topic space is used to categorize messages. For example, in a market data dissemination scenario, messages may be classified according to the stock exchange, the traded commodity, and the kind of information disseminated. A message could for instance be associated with the following topic: `/NASDAQ/ABC-Inc/AskingPrice`. A subscriber can express interest in receiving messages of a specific topic, such as by subscribing to `/NASDAQ/ABC-Inc/AskingPrice`, or by defining a set of messages it is interested in, such as `/NASDAQ/ABC-Inc/*`, which indicates that the subscriber would like to receive any message published with topic `NASDAQ` and `ABC-Inc`.

As in the other publish/subscribe models, the topic-based publish/subscribe model decouples the interaction among publishing data sources and subscribing data sinks. The same decoupling characteristics as discussed under the general publish/subscribe concept apply here as well. Specific realizations of this model found in practice vary in the exact decoupling offered. To properly qualify as publish/subscribe, at least the anonymous communication style must exist. That is publishing clients must not be aware of who the subscribing clients are and how many subscribing clients exist, and vice versa. Thus, topic-based publish/subscribe enables the decoupled interaction of n sources with m sinks for $n, m \geq 1$.

In topic-based publish/subscribe, the publication data model is defined by the topics that can be associated with messages. Simplistic models allow the application developer to categorize messages by defining a flat topic space, simply a collection of topics. More sophisticated approaches allow the application developer to select topics from a hierarchical topic space. Whether flat or hierarchical, topics are often strings, possibly structured with separators for the hierarchical case. Some approaches additionally type the various levels of a hierarchical topic space allowing the application developer to use various operators supported by the type for expressing subscriptions. For example, in he above example, `AskingPrice` could be defined as `Integer`, to allow the subscriber to express a relational condition on the messages returned. This approach is close in expressiveness to the capabilities of content-based or type-based publish/subscribe, as the matching mechanism now also inspects the message content, i.e., the value associated with `AskingPrice`.

The subscription language model depends on the publication data model. A subscription expression defines the subscriber's interest in receiving messages.

Given a flat publication data model, subscribers can express interest in receiving messages of a given topic by specifying the exact topic or by specifying a regular expression that defines interest in a set of possible topics. For a hierarchical publication data model, any part of the hierarchy can be specified as interest by a subscriber in using a wildcard notation to select all messages published by the specified topics.

The publish/subscribe matching problem has the standard interpretation and is defined as determining the set of subscribers based on their subscription expression for a given publication message. This problem is solved over the topic space, which is much simpler than its content-based counter part.

Topic-based publish/subscribe systems are distinguished by the qualities of service the system offers to its clients, such as various degrees of reliability, topic persistence, message ordering constraints, message delivery guarantees, and message delivery latencies constraints. Topic-based publish/subscribe relates to channel-based publish/subscribe in that publishing a message to a channel is similar to associating a message with a topic, which could be the name or identity of the channel. However, in topic-based publish/subscribe this association is reflected in the message itself, while in channel-based publish/subscribe the association is indirect, reflected by selecting a channel, which must not be represented in the message. Topic-based publish/subscribe has more limited filtering capabilities than content-based publish/subscribe, as the message is opaque to the system, while in content-based publish/subscribe the message structure and content is used for determining the set of recipients of a message.

Examples that follow the topic-based publish/subscribe model are the information bus [3], TIBCO's RendezVous product [5], and the series of Web services standards: WS Topics, WS Base Notifications, WS Brokered Notifications [1]. Elements of channel-based publish/subscribe can also be found in the Java Messaging Service [2], the OMG Data Dissemination Service [4], and other messaging middleware. However, these systems are not directly following the topic-based model as described above; rather these approaches are enriched with elements of message queuing, channel-based publish/subscribe, and content-based publish/subscribe.

Topic-based publish/subscribe is intended to support applications that need to selectively disseminate messages from one or more data source to several data sinks, where the mapping of sources to sinks changes dynamically. That is not all sources always communicate with the same sinks. The mapping of which source communicates with which sink is represented through associating topics with messages and subscribing to topics. Most existing systems allow the application to dynamically change subscriptions to topics. Also, applications of topic-based publish/subscribe exist that use the topic as a message log. For these applications the filtering capabilities of the topic-based model is not so important, but message order guarantees, reliability of the queues underlying each topic, and low message delivery latencies are crucial. There are many applications that follow these characteristics. Examples include system integration, selective information dissemination, system management, and database replication.

The term topic-based publish/subscribe is not used uniformly. Abstractions that exhibit the above described functionality are also often referred to as subject-based publish/subscribe systems that offer subject-based addressing to the applications using the system. Subject-based addressing means that interacting applications address each other by publishing messages associated with subjects and by subscribing to subjects of interest. The term subject and topic are used synonymously. Based on the subscriptions registered with the system, the system determines the set of recipients for a given message, without needing explicit address information that identifies that a given message is to be sent to a given destinations. Also, many messaging systems exhibit part of the above described functionality and are simply referred to as messaging systems, message-oriented middleware, or message queuing systems.

Cross-references

▶ Channel-Based Publish/Subscribe
▶ Publish/Subscribe
▶ Type-Based Publish/Subscribe

Recommended Reading

1. Chappell D. and Liu L. (ed). Web Services Brokered Notification 1.2 (WS-BrokeredNotification), working draft 01 edition, July 2004.
2. Hapner M., Burridge R., and Sharma R. Java Message Service. Sun Microsystems, version 1.0.2 edition, November 9th 1999.
3. Oki B., Pfluegl M., Siegel A., and Skeen D. The information bus: an architecture for extensible distributed systems. In Proc. 14th ACM Symp. on Operating System Principles, 1993, pp. 58–68.

4. OMG. Data Distribution Service for Real-time Systems, version 1.2, formal/07-01-01 edition, January 2007.
5. TIBCO. TIBCO Rendezvous, software release 8.1 edition, April 2008.

Topic-Directed Web Crawling

▶ Focused Web Crawling

Top-k Queries in P2P Systems

▶ Approximate Queries in Peer-to-Peer Systems

Top-K Selection Queries on Multimedia Datasets

Amélie Marian
Rutgers University, Piscataway, NJ, USA

Synonyms

Ranked multimedia retrieval; Aggregation algorithms for middleware systems; Evaluation of fuzzy queries over multimedia systems

Definition

Traditionally, queries over structured (e.g., relational) data identify the exact matches for the queries. This exact-match query model is not appropriate for a multimedia dataset scenario where queries are inherently fuzzy – often expressing user preferences and not hard Boolean constraints – and are best answered with a ranked, or "top-*k*," list of the best matching objects. Efficient top-*k* query algorithms for such applications must take into account the specific challenges in accessing multimedia data. In particular, the query model should consider the access interfaces available to retrieve object attribute information, as well as the cost of retrieving this attribute information.

Historical Background

Content management in multimedia repositories is an important problem as more and more multimedia applications are developed. For example, digitization of photo and art collections is increasingly popular, multimedia mail and groupware applications are becoming widely available, and satellite images are being used for weather predictions. To access such large repositories efficiently, multimedia objects need to be queried via their attribute values, such as the date the multimedia object was authored, a free-text description of the object, and features like color histograms.

There are at least three major ways in which accesses to a multimedia repository differ from that of a structured database (e.g., a relational database). First, the data are inherently fuzzy: rarely does a user expect an exact match with the features of a multimedia object (e.g., color histogram). Rather, an object does not either satisfy or fail a condition, but has instead an associated grade of match [3,5]. Thus, an atomic query condition will not be a filter testing for an equality between two values (e.g., between a given color *c* and the color *O.c* of an object *O*) as is usually the case in an exact query model scenario, but instead will assign a score representing the grade of match between the two values (e.g., *GradeColor*(*c*,*O.c*)). Next, every condition on an attribute of a multimedia object may only be separately evaluated through calls to a system or index that handles that particular attribute. This is in contrast to a traditional database where, after accessing a tuple, all selection predicates can be evaluated on the tuple. Finally, the process of querying and browsing over a multimedia repository is likely to be interactive, and users will tend to ask for only a few best matches according to a ranking criterion.

Foundations

Existing query processing techniques for relational data cannot efficiently be applied to multimedia scenarios as object attribute information is often kept separate, possibly in different subsystems, and is typically expensive to retrieve. In addition, the fuzzy nature of the queries means that users are only interested in the best matches, making it unnecessary to evaluate every object.

Query Model

Consider a collection *C* of objects with attributes $A_1,\ldots,A_n$. A top-*k* query over collection C simply specifies target values for each attribute A_i. Therefore, a top-*k* query *q* is an assignment of values $\{A_1 = q_1,\ldots,A_n = q_n\}$ to the attributes of interest. The answer to the top-*k* query $q = \{A_1 = q_1,\ldots,A_n = q_n\}$ over a collection of objects C and for a scoring function is a list of the *k* objects in the collection with the highest score for the query. The final score that each object t in C receives for q is generally a function of a score for each individual attribute A_i of t. Typically, the scoring function that is associated with each attribute A_i is application-dependent. Top-k algorithms presented in the literature can be applied to a variety of aggregate scoring functions as long as they satisfy some monotonicity requirements [1,4,5,8].

Typically, multimedia attribute values (or scores) can only be accessed through specific interfaces. Two types of access to data, along with their associated costs, can be distinguished. The first type of access is sorted (or sequential) access, which allows retrieving objects through a list sorted by the objects' attribute scores (for instance, all images stored by degree of redness). The second type of access is random access, which allows to directly access the attribute score of a given object. A sorted access is usually cheaper than a random access as it can make use of sequential access to precomputed index structures. However, sorted access does require to access every object in the attribute's score order. The multimedia system may allow either sorted- or random-access, or both, for each attribute score, depending on the underlying subsystems.

Top-*k* Query Evaluation Algorithms

A naive brute-force top-k query processing strategy would consist of computing the score for the query for every object to identify and return k objects with the best scores. For large collections of objects, it is easy to see that this brute-force evaluation could be prohibitively expensive. Fortunately, the top-k query model provides the opportunity for efficient query processing, as only the best k objects need to be returned. Objects that are not part of the top-k answer, therefore, might not need to be processed. The challenge faced by top-k query processing techniques is then to identify the top-k objects efficiently, to limit the amount of processing done on non-top-k objects. To this end, various top-k query processing strategies have been presented.

The Threshold Algorithm

To process queries involving multiple multimedia attributes, Fagin et al. proposed a family of algorithms [3,4,5], developed as part of IBM Almaden's Garlic project. These algorithms can evaluate top-k queries that involve several independent multimedia "subsystems," each producing scores that are combined using arbitrary monotonic aggregation functions. The initial *FA* algorithm [3] was followed by "instance optimal" query processing algorithms over sources that allow for sorted accesses and possibly random accesses (*TA* algorithm) or only for sorted accesses (*NRA* algorithm) [4]. In later work, Fagin et al. [5] introduced the TA_z algorithm, a variation of *TA* that also handles sources that only provide random-access interfaces. These algorithms rely on making dynamic choices for scheduling index lookups during query execution in order to prune low-scoring candidate items as early as possible. *TA* does not need an unbounded buffer, and dynamically considers object grades to decide when to stop retrieving new objects. Specifically, *TA* stops retrieving new objects when it finds a threshold grade G such that (1) at least k objects with grade G or higher have been identified and (2) no unretrieved object can have a grade greater than G.

Nepal and Ramakrishna [10] and Güntzer et al. [6] presented variations of Fagin et al.'s *TA* algorithm [4] for processing queries over multimedia databases. In particular, Güntzer et al. [6] reduce the number of random accesses through the introduction of more stop-condition tests and by exploiting the data distribution.

While these algorithms are proved "instance optimal," i.e, the consider the minimum number of objects needed to correctly identify the top-k answer, they completely evaluate each object they consider.

Algorithms based on Expensive Predicates Evaluation

Some works have built upon the *TA* family of algorithms to further improve query processing efficiency by reducing the number of expensive random accesses.

Marian et al.'s *Upper* algorithm [8] picks the most *promising* object-attribute pair to process at any given time based on the result of previous accesses. The *Upper* algorithm requires keeping track of partially evaluated object score bounds. By interleaving the processing of objects, and discarding objects that are not fully evaluated, *Upper* results in significant savings in random access costs. Marian et al. also proposed *TA-EP*, an optimization of *TA* that exploits existing techniques for processing selections with expensive predicates. In addition, Marian et al. [8] proposes extending the *Upper* algorithm to efficient parallel evaluation.

Chang and Hwang [1] presented *MPro*, an algorithm that relies on identifying *necessary* probes to optimize the execution of *expensive predicates* for top-k queries. Unlike *Upper*, *MPro* always evaluate attributes in the same order for every object. Chang and Hwang also briefly discussed parallelization techniques for *MPro* and proposed the *Probe-Parallel-MPro* algorithm.

Filter/Restart Method

Algorithms based on Fagin et al.'s *TA* dynamically refine a threshold value *G* based on the status of evaluated objects. As a result, these algorithms can be processed continuously, until a solution is reached. In contrast, some techniques have focused on translating top-k queries into standard selection queries. While this approach allows using existing query evaluation and optimization implementations, it may require to "restart" a query, if the translation does not return at least k results.

In particular, Chaudhuri et al. built on Fagin's original *FA* algorithm and proposed a cost-based approach for optimizing the execution of top-k queries over multimedia repositories [2]. Their strategy translates a given top-k query into a selection (filter) query that returns a (hopefully tight) superset of the actual top-k tuples using data distribution information to estimate the value G that is expected to be the score of the k^{th} object. Ultimately, the evaluation strategy consists of retrieving the top-k' tuples from as few sources as possible, for some $k' \geq k$, and then probing the remaining sources by invoking existing strategies for processing selections with expensive predicates.

Using Pre-computed Views

Another approach to top-k query evaluation is to use precomputed top-k query indexes. Various top-k queries, with different scoring functions, are evaluated to create indexes, which are used whenever a new top-k query is entered. A challenge of such an approach is to correctly identify the most efficient index for the new query. The *PREFER* system [7] uses pre-materialized views to efficiently answer ranked preference queries over commercial DBMSs. *PREFER* precomputes a set of materialized views that provide guaranteed query performance and, for any new top-k query, selects a near optimal set of views under space constraints.

Handling Joins

Top-k query evaluation algorithms over arbitrary joins have been presented for multimedia applications [1,9]. They use a ranking function that combines individual tuple scores. These algorithm handle the possible explosion in the number of results resulting from the join operation.

Key Applications

Multimedia Search

Typical search queries in multimedia systems require for fuzzy matches on predicates that are expensive to evaluate as the corresponding information is not stored in indexes (e.g., similarity to a user-specified image). Top-k algorithms are designed to minimize the number of accesses to the data, only focusing on those that are needed to identify the best query answers.

Information Integration

In scenarios where many sources may be accessed to answer a query (e.g., web databases, legacy systems), using algorithms that are designed to minimize the number of these expensive remote accesses is an important aspect of query processing efficiency.

Future Directions

Research on top-k query processing in multimedia scenarios has so far focused mostly on efficiency. Relatively little attention has been devoted to evaluating the quality and usefulness of the resulting top-k answers. In contrast, the design of good scoring functions for (relatively unstructured) text documents has been the main focus of the IR community for the last few decades. Many lessons and techniques from IR can be applied to a more structured multimedia scenario.

Cross-references

▶ Multimedia Information Retrieval Model
▶ Multimedia Retrieval Evaluation
▶ Similarity and ranking operations

Recommended Reading

1. Chang K.C.-C. and Hwang S. Minimal probing: supporting expensive predicates for top-k queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 346–357.
2. Chaudhuri S., Gravano L., and Marian A. Optimizing top-k selection queries over multimedia repositories. IEEE Trans. Knowledge and Data Eng., 16(8):992–1009, August 2004.
3. Fagin R. Combining fuzzy information from multiple systems. In Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1996, pp. 216–226.
4. Fagin R., Lotem A., and Naor M. Optimal aggregation algorithms for middleware. In Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2001, pp. 102–113.

5. Fagin R., Lotem A., and Naor M. Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci., 66(4), 2003.
6. Güntzer U., Balke W.-T., and Kießling W. Optimizing multi-feature queries for image databases. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 419–428.
7. Hristidis V., Koudas N., and Papakonstantinou Y. PREFER: a system for the efficient execution of multi-parametric ranked queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 259–270.
8. Marian A., Bruno N., and Gravano L. Evaluating top-*k* queries over web-accessible databases. ACM Trans. Database Syst., 29(2):319–362, 2004.
9. Natsev A., Chang Y.-C., Smith J.R., Li C.-S., and Vitter J.S. Supporting incremental join queries on ranked inputs. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 281–290.
10. Nepal S. and Ramakrishna M.V. Query processing issues in image (multimedia) databases. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 22–29.

Top-k XML Query Processing

► Ranked XML Processing

Topological Data Models

Erik Hoel
Environmental Systems Research Institute, Redlands, CA, USA

Synonyms

Topology; Topological fabric; Topological data structure

Definition

Topology is defined as a mathematical model used to define the location of and relationships between geographical phenomena. These topological relationships are independent of distance or direction. Topology may depict connectivity of one entity to another; for example, an edge will have topological relationships to it's from and to nodes.

Topology is useful with spatial data because many spatial modeling or geoprocessing operations do not require geometric coordinate values. For example, to find the shortest path between two nodes requires a list of which edges connect to each other and the cost of traversing along each edge. Geometric coordinates are only necessary to draw the shortest path after it is calculated.

More generally, topology, in the context of spatial data, can have several other meanings:

- A mathematical model of features in space (e.g., nodes, edges, and faces).
- A physical data model for efficient representation of feature data.
- A mechanism that can be used to ensure data quality (e.g., no gaps or overlaps between polygons).
- A mechanism that allows the management of shared geometry.
- A mechanism that facilitates navigation between features using topological relationships (e.g., equal, disjoint, intersects, touches, crosses, within, contains, overlaps, and relate – the nine topological relationships in the dimensionally extended nine-intersection model [5]).

A topological data model is used to represent collections of features that are assembled into a topology (or topological fabric). Topological data models come in many different variants (as described in the following), but the central theme for each of the models is the storage and representation of spatial data that forms a topological fabric.

Historical Background

Topological data structures have been used to represent geographic information for over 40 years [3,14]. The topological model has been the basis of a number of operational systems (see, for example DIME [3], GIRAS [11], ODYSSEY [13], ARC/INFO [1], TIGRIS [7], and TIGER [9]). Many of these systems have been based on binary file and in-memory data structures and supported a single-writer editing model on geographic libraries organized as a set of individual map sheets or tiles.

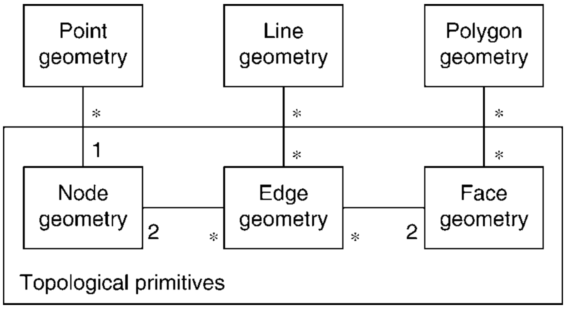
Topology has historically been viewed as a spatial data structure used primarily to ensure that the associated data forms a consistent and clean topological fabric. Topology is used most fundamentally to ensure data quality (e.g., no gaps or overlaps between polygons representing land parcels) and allow a GIS to more realistically represent geographic features. Topology allows one to control the geometric relationships between features and maintain their geometric integrity.

Foundations

Topological data structures for representing geographic information are a standard topic in geographic information science (see [6], for example, for an excellent definition of the mathematical theory underlying this information model). In general, the topological data model represents spatial objects (point, line, and area features) using an underlying set of topological primitives. These primitives, together with their relationships to one another and to the features, are defined by embedding the feature geometries in a single planar graph. Such datasets are said to be "topologically integrated."

The model associates one or more topological primitives (i.e., nodes, edges, and faces; or 0-cells, 1-cells, and 2-cells in the TIGER parlance) with spatial objects of varying geometry type (i.e., points, lines, and polygons respectively). More specifically, a feature with point geometry is associated with a single node element, a feature with line geometry is associated with one or more edge elements, and a feature with polygon geometry is associated with one or more face elements. This is depicted in Fig. 1 as the generic topology model.

There are additional relationships between the topological elements themselves as is also shown in Fig. 1. A node element may or may not be associated with a collection of edge elements. A face element may be associated with one or more edge elements. Finally, an edge element is associated with two node elements and two face elements. The relationships between nodes and faces may either be implicit or explicit.



Topological Data Models. Figure 1. Generic topology model.

The common representation of a topology is as a collection of topological primitives – i.e., nodes, arcs, and faces, with explicit relationships between the primitives themselves. For example, an arc would have a relationship to the face on the left, and the face on the right. With advances in GIS development, an alternative view of topology has evolved. Topology can be modeled as a collection of rules and relationships that, coupled with a set of editing tools and techniques, enables a GIS to more accurately model geometric relationships found in the world.

Topology, implemented as feature behavior and user specified rules, allows a more flexible set of geometric relationships to be modeled than topology implemented as a data structure. For example, older data structure based topology models enforce a fixed collection of rules that define topological integrity within a collection of data. The alternative approach (feature behavior and rules) allows topological relationships to exist between more discrete types of features within a feature dataset. In this alternative view, topology may still be employed to ensure that the data forms a clean and consistent topological fabric, but also more broadly, it is used to ensure that the features obey the key geometric rules defined for their role in the database.

Topological data structures, beginning with DIME in 1967 [3] have been used to represent features assembled into a topological fabric in a number of different ways over the past 40 years. In the following, the seven or so significant variants that have emerged during this period are described.

DIME Files

The US Census Bureau, as part of the New Haven Census Use Study of 1967, undertook the development of an explicit topological data model for their geographic data [3]. This system was called DIME (for Dual Independent Map Encoding) and was intended to facilitate the automation of detecting topological errors in the base geographic data. DIME files were based upon planar line segments defined by two endpoints. The line segments correspond to the Street Segment records as shown in Fig. 1. Each street segment was associated with a start and end node identifier (endpoint), left and right side block and tract identifiers, as well as an address range. Drawbacks of the DIME model include the need to perform searches in order to assemble polygons from the street segment records, or to determine all the segments sharing a given node.

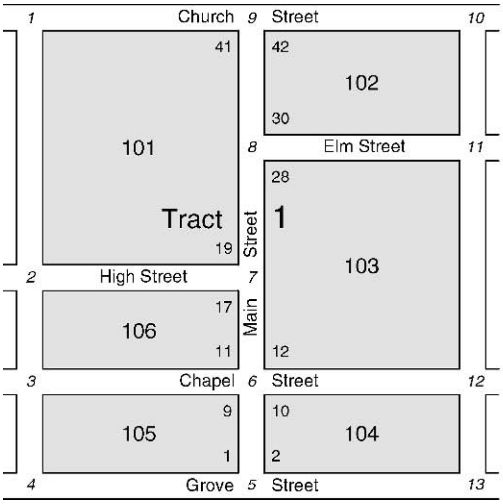
In Fig. 2, Main Street is represented as a collection of four records in the Street Segment Records. Each (line) segment is defined by two end points (there are no midspan shape points for the segment). The segment is associated with the start and end node, identifiers for the blocks and tracts on the left and right sides of the segment, as well as the low and high address ranges for each side of the segment. In addition, it is assumed that the segments are planar – segments may not cross each other.

The "dual independent" portion of the DIME model reflects the redundant nature of how the topology is represented. Topological fabrics can be represented using the collection of relationships between edges and nodes, or nodes and faces, or faces and edges. With DIME, the topological correctness may be verified using either block chaining or node chaining. Block chaining involves finding all the segment records that have a given block on the left or right side. By rearranging the from/to orientations of each segment (and the associated left/right oriented attributes) such that the block is on the right side of each segment, it is possible to chain the nodes. This involves walking from the "to node" of one segment to the "from node" of another segment, continuing until all segments are visited. If there is a topological problem, it will not be possible to chain the blocks in this manner [2].

Due to the dual nature of DIME, it is also possible to chain the nodes in order to find topological -problems. Specifically, for a given node, select all segments that have it as a "to" or "from" node. Then, after rearranging the from/to orientations of each segment such that the node is always in the "to node" position, one may chain all the blocks surrounding the node (moving from the "block right" record in one segment to the "block left" record in another segment). Thus, because of the dual independent encoding of DIME, one may use two independent mechanisms to detect topological problems.

POLYVRT

The Harvard Laboratory for Computer Graphics and Analysis developed a topological data structure, termed POLYVRT, that was intended to serve as a data structure to facilitate interchange between various other data models [14]. POLYVRT extended the representational capabilities of DIME to allow planar segment chains to exist between nodes. This allowed line detail to be efficiently handled. In addition, POLYVRT enables the user to readily flip between segment chains and polygons.



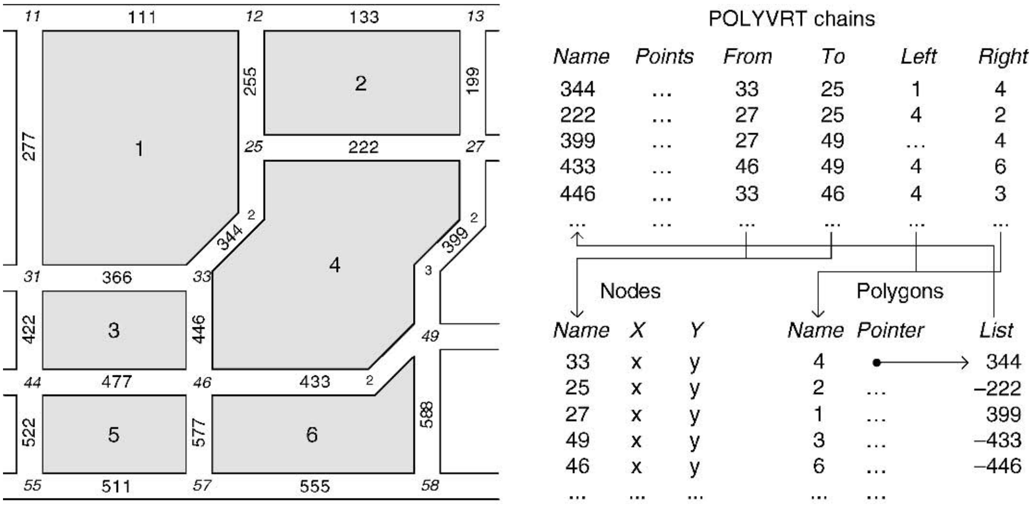
Census address coding guide records

Street	Tract	Block	Low Address	High Address
Main	1	102	30	42
Main	1	103	12	28
Main	1	104	2	10
Main	1	105	11	9
Main	1	106	11	17
Main	1	101	19	41

DIME Street segment records

Street	Start Node	End Node	Tract Left	Tract Right	Block Left	Block Right	LLow Addr.	LHigh Addr.	RLow Addr.	RHigh Addr.
Main	5	6	1	1	105	104	1	9	2	10
Main	6	7	1	1	106	103	11	17	12	18
Main	7	8	1	1	101	103	19	27	20	28
Main	8	9	1	1	101	102	29	41	30	42

Topological Data Models. Figure 2. Example of the DIME approach to storing topology (adapted from [3]).



POLYVRT chains

Name	Points	From	To	Left	Right
344	...	33	25	1	4
222	...	27	25	4	2
399	...	27	49	...	4
433	...	46	49	4	6
446	...	33	46	4	3
...	...	...	...	...	...

Nodes

Name	X	Y
33	x	y
25	x	y
27	x	y
49	x	y
46	x	y
...	...	...

Polygons

Name	Pointer	List
4	● →	344
2	...	–222
1	...	399
3	...	–433
6	...	–446
...	...	

Topological Data Models. Figure 3. Example of the POLYVRT approach to storing topology.

Figure 3 contains an analogous example as shown in Fig. 2 but using the POLYVRT representation as well as adding shape points to various segments (e.g., segments 344, 399, and 433). A POLYVRT chain record contains a unique identifier, a pointer to the shape points (termed Points), identifiers of the from and to nodes, and identifiers of the polygons on the left and right sides. Records in the nodes table contain a unique identifier and an (x, y) coordinate value. Entries in the Polygons table have a unique identifier and a pointer to a list of all associated chains that compose the boundary of the polygon (e.g., for polygon 4, the list contains chains 344, 222, 399, 433, and 446).

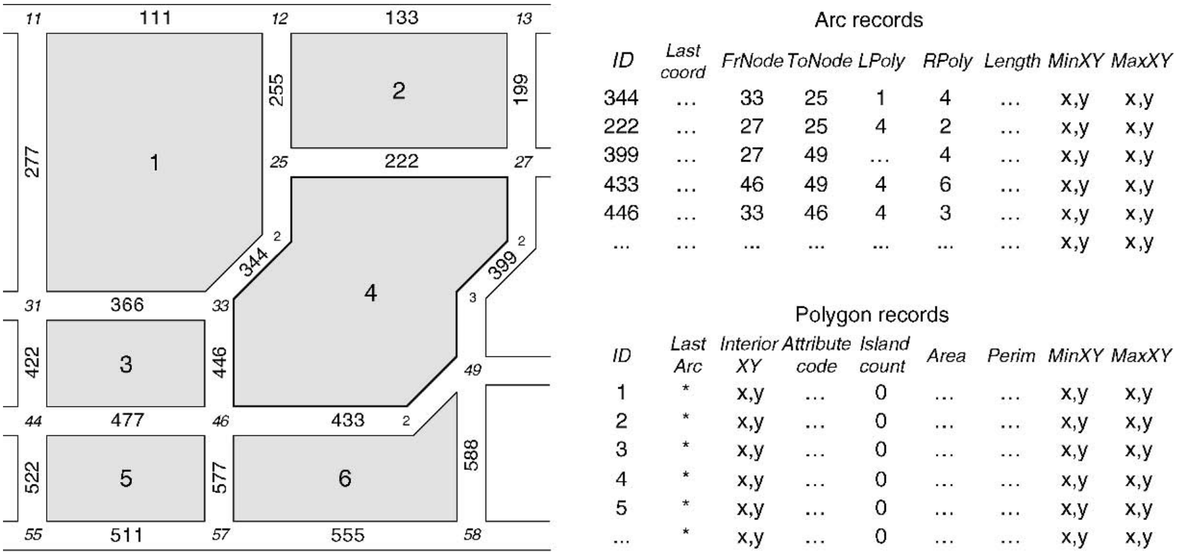
GIRAS

GIRAS (Geographical Information Retrieval and Analysis System) was developed by the US Geological Survey the mid 1970s [11]. The GIRAS topological data structure was motivated by the need to represent polygonal data. In addition, it was determined that it was more efficient to store certain types of data rather than recompute it. As a consequence, a large amount of ancillary data such as polygon perimeter and area was explicitly stored in the data structure. The data structure was based upon arcs and was considered a direct descendent of DIME and POLYVRT due to its topological similarities.

Figure 4 contains the same example dataset as shown in Figs. 2 and 3 except using the GIRAS topology model. The GIRAS model differs from DIME and POLYVRT as various attributes unrelated to storing the topology are maintained. This includes the length as well as bounding rectangle information for the Arc records (Length, MinXY, and MaxXY). In addition, the Arc records store the attribute codes of the left and right polygons (this is not represented in the figure). The Polygon records similarly store the area and perimeter of the polygon, the bounding rectangle information, as well as the attribute code and the island count (number of islands found within the polygon). Note depicted in the figure is the identifier of the enclosing polygon if the polygon serves as an island to another polygon. Finally, there are structures for representing the vertex coordinates of the arcs (e.g., LastCoord in the Arc Records table), as well as another structure that maps arcs to polygons (the FAP file in GIRAS).

TIGER

TIGER (Topologically Integrated Geographic Encoding and Referencing) was developed by the US Census Bureau during the 1980s as an evolution of the earlier DIME-file model [9]. New features and capabilities were added (e.g., curve points for linear features) to the model in order to create a comprehensive system that could support all of the various censuses. TIGER was first used for the 1990 Census of Population and Housing.



Arc records

ID	Last coord	FrNode	ToNode	LPoly	RPoly	Length	MinXY	MaxXY
344	…	33	25	1	4	…	x,y	x,y
222	…	27	25	4	2	…	x,y	x,y
399	…	27	49	…	4	…	x,y	x,y
433	…	46	49	4	6	…	x,y	x,y
446	…	33	46	4	3	…	x,y	x,y
…	…	…	…	…	…	…	x,y	x,y

Polygon records

ID	Last Arc	Interior XY	Attribute code	Island count	Area	Perim	MinXY	MaxXY
1	*	x,y	…	0	…	…	x,y	x,y
2	*	x,y	…	0	…	…	x,y	x,y
3	*	x,y	…	0	…	…	x,y	x,y
4	*	x,y	…	0	…	…	x,y	x,y
5	*	x,y	…	0	…	…	x,y	x,y
…	*	x,y	…	0	…	…	x,y	x,y

Topological Data Models. Figure 4. Example of the GIRAS approach to storing topology.

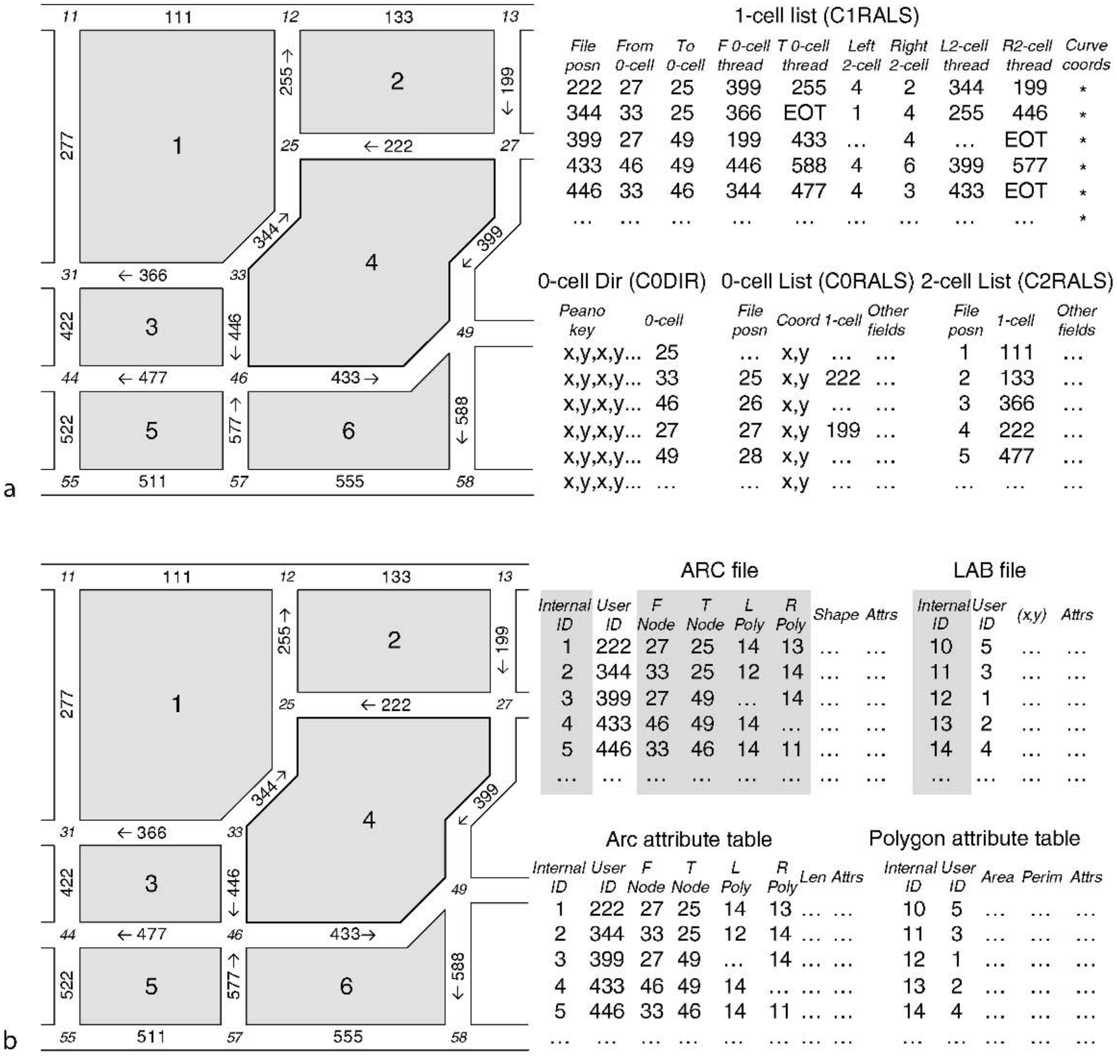
The topology model within TIGER was based upon the two dimensional network model of Corbett [6]. This model used three topological primitives termed 0, 1, and 2-cells (analogous to nodes, edges, and faces in other topological data models).

Figure 5 contains the same example dataset as shown in Figs. 2–4 except using the TIGER topology model (note – some liberty has been taken as in reality, the entries in the list tables are all consecutively numbered – this is not necessarily shown here; e.g., the file position in the C1RALS table). Within TIGER, 0 and 2-cells may be accessed via a directory mechanism. In the figure, the directory for 0-cells is shown (termed C0DIR). This directory contains a record for each 0-cell in the 0-cell List table (C0RALS). The directory entries are sorted by a simple Peano key (an alternating bit merging of longitude and latitude values for the associated point); this enables nearest point queries, etc. Each record in the directory table references a 0-cell in the 0-cell List table. The 0-cell List table is not geographically sorted. The 0-cell entries contain the x, y coordinate value of the point, and a pointer to the 1-cell List table (C1RALS) for the lowest value 1-cell (according to the file position) that is associated with the 0-cell at an endpoint. The entries in the 1-cell List table contain back-pointers to the from and to 0-cells, as well as threading pointer to other 1-cell records that enable a counter-clockwise traversal of all 1-cells associated with a given 0-cell (note that the last record in the thread contains a terminator – "EOT" in the figure). The 1-cell records also contain references to the 2-cells (faces) on the left and right sides as well as threading pointers to other 1-cells that enable the traversal of all 1-cells associated with a 2-cell. Finally, the 2-cell List (or C2RALS) table contains an entry for each 2-cell. Each entry contains a reference to the first 1-cell found in the C1RALS table that is associated with the 2-cell.

ARC/INFO Coverages

The motivating requirements behind the development of the ARC/INFO Coverage model were a model that had a strong theoretical basis (topology), as well as simplicity and efficiency (e.g., ability to support efficient geoprocessing functions such as polygon overlays and dissolves) [12].

The Coverage topological data model uses a collection of tables. End users are responsible for populating the ARC and LAB files. The entries in the ARC file correspond to line segments with optional attributes. In Fig. 6, the shaded portions of the ARC file correspond to system generated fields. Thus, the user specifies the user id field along with the shape (geometry) as well as any attributes. End users are also responsible for populating entries in the LAB (for label) file. Each label is used to specify the attributes



1-cell list (C1RALS)

File posn	From 0-cell	To 0-cell	F 0-cell thread	T 0-cell thread	Left 2-cell	Right 2-cell	L2-cell thread	R2-cell thread	Curve coords
222	27	25	399	255	4	2	344	199	*
344	33	25	366	EOT	1	4	255	446	*
399	27	49	199	433	…	4	…	EOT	*
433	46	49	446	588	4	6	399	577	*
446	33	46	344	477	4	3	433	EOT	*
…	…	…	…	…	…	…	…	…	*

0-cell Dir (C0DIR)

Peano key	0-cell
x,y,x,y...	25
x,y,x,y...	33
x,y,x,y...	46
x,y,x,y...	27
x,y,x,y...	49
x,y,x,y...	…

0-cell List (C0RALS)

File posn	Coord	1-cell	Other fields
…	x,y	…	…
25	x,y	222	…
26	x,y	…	…
27	x,y	199	…
28	x,y	…	…
…	x,y	…	…

2-cell List (C2RALS)

File posn	1-cell	Other fields
1	111	…
2	133	…
3	366	…
4	222	…
5	477	…
…	…	…

ARC file

Internal ID	User ID	F Node	T Node	L Poly	R Poly	Shape	Attrs
1	222	27	25	14	13	…	…
2	344	33	25	12	14	…	…
3	399	27	49	…	14	…	…
4	433	46	49	14	…	…	…
5	446	33	46	14	11	…	…
…	…	…	…	…	…	…	…

LAB file

Internal ID	User ID	(x,y)	Attrs
10	5	…	…
11	3	…	…
12	1	…	…
13	2	…	…
14	4	…	…
…	…	…	…

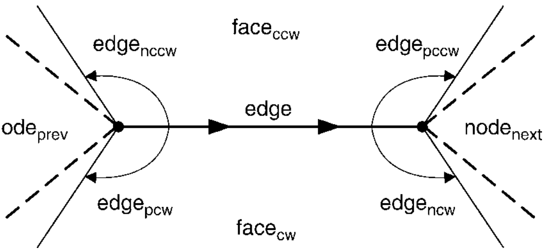
Arc attribute table

Internal ID	User ID	F Node	T Node	L Poly	R Poly	Len	Attrs
1	222	27	25	14	13	…	…
2	344	33	25	12	14	…	…
3	399	27	49	…	14	…	…
4	433	46	49	14	…	…	…
5	446	33	46	14	11	…	…
…	…	…	…	…	…	…	…

Polygon attribute table

Internal ID	User ID	Area	Perim	Attrs
10	5	…	…	…
11	3	…	…	…
12	1	…	…	…
13	2	…	…	…
14	4	…	…	…
…	…	…	…	…

Topological Data Models. Figure 5. Example of the TIGER approach to storing topology.

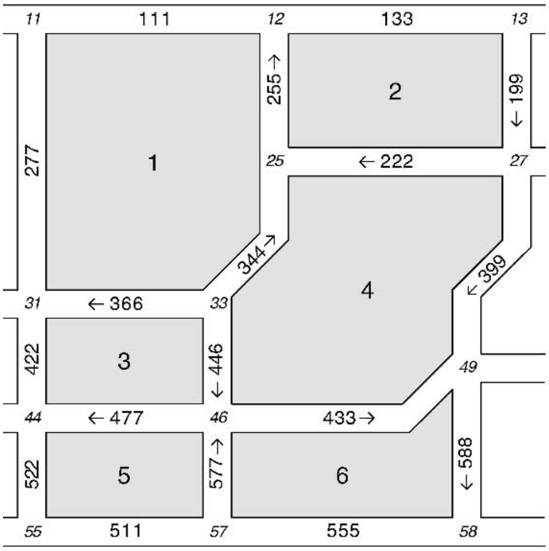


Topological Data Models. Figure 6. Example of the Coverage approach to storing topology (simplified).

that will be associated with a polygon in the topology. Following the population of the ARC and LAB files, the user will perform a topological integration and structuring of the data. The result of this is the population of the Arc Attribute Table (AAT) and the Polygon Attribute Table (or PAT). In addition, other fields in the ARC file are populated (i.e., the FNODE, TNODE, LPOLY, and RPOLY). There are other tables that form the Coverage data model that are not depicted in Fig. 6 (e.g., the Polygon Arc List, or PAL file) that provide additional explicit topological relationships.

Relational

Many topological data models stored in relational databases utilize a storage representation that relies upon a fixed length record. The winged-edge data structure [1], a fixed-length storage representation for representing the relationships between nodes, edges, and faces, simplified the task of representing



Edges

Edge ID	Start node	End node	Next LEdge	Prev LEdge	Next REdge	Prev REdge	LFace	RFace	Geom
344	33	25	255	–366	–222	–446	1	4	…
222	27	25	–344	–399	255	199	4	2	…
399	27	49	…	…	–433	–222	…	4	…
433	46	49	–399	446	588	577	4	6	…
446	33	46	433	–344	477	–366	4	3	…
…	…	…	…	…	…	…	…	…	…

Nodes

Node ID	Edge ID	Face ID	Geom
33	344	–	…
25	–222	–	…
27	399	–	…
49	–433	–	…
46	477	–	…
…	…	…	…

Faces

Face ID	Bdy Edge	Island EList	Island NList	Geom
1	255	–	–	…
2	–199	–	–	…
3	–446	–	–	…
4	–344	–	–	…
5	577	–	–	…
6	–577	–	–	…

Topological Data Models. Figure 7. Example of an edge and its four wings in the winged-edge data structure of Baumgart.

explicit topological data models within a relational database system. The edge record is the primal structure in this representation. For each edge, the identifiers of the start and end nodes (termed *node*$_{prev}$ and *node*$_{next}$ in Fig. 7), and the two faces, one in the clockwise traversal orientation (*face*$_{cw}$), and the other in the counterclockwise orientation (*face*$_{ccw}$) are represented. Finally, identifiers of four connected edges, two at each node, are stored. The four edges correspond to the previous and next edges found when traversing the two adjacent polygons in the clockwise (*edge*$_{pcw}$ and *edge*$_{ncw}$) and counterclockwise (*edge*$_{pccw}$ and *edge*$_{nccw}$) orientations.

A topological data model can be stored in a relational database. The topology model is represented using three tables – an edge table, a node table, and a face table. The edge table is essentially encoding the winged-edge data structure in addition to the geometry of the portion of the feature associated with the edge. The edge table utilizes negative identifiers to encode the orientation of the four edges found at the start or end nodes.

ArcGIS Geodatabase

The ArcGIS Geodatabase approach to modeling topology represents it as a collection of rules and relationships, coupled with a set of editing tools and techniques [8]. With the standard models (e.g., TIGER, Relational, etc.), it is possible to obtain topological primitives from feature geometry; similarly, it is possible to obtain feature geometry from topological primitives. Effectively, the geometry found in features is a dual representation of the geometry that would be found on the topological primitives. This topology model simplifies the generic explicit topology model and does not need to make both representations persistent. The process of topological integration (validation) results in vertex equality where features share underlying topological primitives. Given vertex equality, reconstruction of topological primitives is straightforward. Vertices on feature geometries in this scheme play the same role as that assigned to embedded foreign keys in data structures that explicitly model topological primitives.

Topological primitives and relationships are only instantiated during the process of topological validation or when required by the client application (this is similar to Intergraph's MGE where topology is selectively built but the topological primitives are not persisted in the RDBMS). The primary reason for this alternative approach is that it is easier (faster, more scalable) to recreate an index (e.g., the topological primitives) than to do all the bookkeeping necessary to make the topological primitives persistent and retrieve the primitives from the database while preserving the database transaction model. Additionally, it is frequently the case that the portion of the topological primitives necessary for an operation is small relative to the entire topology (e.g., editing a few block groups in a localized area within TIGER).

In order for this approach to be viable from a performance standpoint, it is critical that there exists a high performance topology engine that validates the portion of the topology in question as well as instantiate the topological primitives for the given collection of features within the topology [15].

At a high level, this topology model consists of a collection of feature classes (homogeneous collections of features), topology rules, and other metadata used to support the validation model. This metadata includes dirty areas (areas that have not been validated following updates or edits), topology errors, and the cluster tolerance (i.e., the distance range in which vertices and boundaries are considered identical or coincident). Topological integrity is defined with respect to a collection of topology rules. Topology rules are used to define constraints on the permissible topological relationships between features in one or more feature classes that participate in the topology. The collection of topology rules that are associated with the topology are selected on the basis of which topological relationships are important for the user's model.

The validation process is a fundamental operation on a topology performed by a topology engine. The validation process on a topology is responsible for ensuring that the first three of Milenkovic's five normalization rules [10] on all spatial objects participating in the topology are respected:

1. No two vertices are closer than ε.
2. No vertex is closer than ε to an edge of which it is not an endpoint.
3. No two edges intersect except at their endpoints.

In addition, the validation process is responsible for checking all specified topology rules and generating topology errors at locations where rules are violated.

Topology rules are checked when the topology is validated. When a topology rule is violated, a topology error object is generated. This topology error may be represented as a special type of feature that may itself be persisted. At a later point following the validation, the user may then review the topology error objects, and the error conditions may be corrected. Topology rule violations do not prevent the validation operation from completing successfully. Examples of topological rules that may be applied to polygon features include:

- The interiors of polygons in a feature class must not overlap (they may however share edges or vertices).
- Polygons must not have voids within themselves or between adjacent polygons (they may share edges, vertices, or interior areas).
- Polygons of one feature class must share all their area with polygons in another feature class (i.e., they must cover each other).

Key Applications

Cadastral databases, land use information systems, overlay processing, geoprocessing, topological analysis, dataset quality assurance/quality control (QA/QC).

Cross-references

▶ Geographic Information System
▶ Geographical Information Retrieval
▶ Network Data Model
▶ Spatial Data Analysis
▶ Spatial Data Types
▶ Spatial Network Databases
▶ Spatial Operations and Map Operations
▶ Topological Relationships

Recommended Reading

1. Baumgart B. A polyhedron representation for computer vision. In National Computer Conf., 1975, pp. 589–596.
2. Census Bureau. The DIME Geocoding System. Report No. 4, Census Use Study, US Department of Commerce, Bureau of the Census, 1970.
3. Cooke D. and Maxfield W. The development of a geographic base file and its uses for mapping. In Proc. 5th Annual Conf. Urban and Regional Information System Association, 1967, pp. 207–218.
4. Corbett J. Topological Principles in Cartography. Technical Paper 48. Bureau of the Census, Washington, DC, 1979.
5. Egenhofer M., Clementini E., and Di Felice P. Topological relations between regions with holes. Int. J. Geograph. Inform. Syst., 8(2):129–142, 1994.
6. Güting R. and Schneider M. Realm-based spatial data types: the ROSE algebra. VLDB J., 4(2):243–286, 1995.
7. Herring J. TIGRIS: topologically integrated geographic information system. In Proc. 8th Int. Symp. on Computer Assisted Cartography, 1987, pp. 282–291.
8. Hoel E., Menon S., and Morehouse S. Building a robust relational implementation of topology. In Proc. 8th Int. Symp. Advances in Spatial and Temporal Databases, 2003, pp. 508–524.
9. Marx R. The TIGER system: automating the geographic structure of the United States. In Introductory Readings in

Geographic Information Systems. Peuquet Marble Taylor & Francis, London, 1990.
10. Milenkovic V. Verifiable implementations of geometric algorithms using finite precision arithmetic. Artif. Intell., 37(1–3):377–401, 1988.
11. Mitchell W., Guptill S., Anderson K., Fegeas R., and Hallam C. GIRAS: A geographic information retrieval and analysis system for handling land use and land cover data: US Geological Survey Professional Paper 1059, GPO, Washington, DC, 1977.
12. Morehouse S. ARC/INFO: a geo-relational model for spatial information. In Proc. 7th Int. Symp. on Computer Assisted Cartography, 1985, pp. 388–397.
13. Morehouse S. and Broekhuysen M. ODYSSEY User's Manual. Laboratory for Computer Graphics and Spatial Analysis, Harvard Graduate School of Design, Cambridge, MA, 1982.
14. Peucker T. and Chrisman N. Cartographic data structures. Am. Cartograph., 2(1): 55–69, 1975.
15. van Roessel J. A new approach to plane-sweep overlay: topological structuring and line-segment classification. Cartograph. Geograph. Inform. Syst., 18(1), 1991.

Topological Data Structure

▶ Topological Data Models

Topological Fabric

▶ Topological Data Models

Topological Relationships

Paolino Di Felice, Eliseo Clementini
University of L'Aguila, L'Aguila, Italy

Definition

Topological relationships describe qualitative properties that characterize the relative position of spatial objects. *disjoint, meet, overlap*, and *inside* are few examples (Fig. 1).

Topology is considered the most primitive kind of spatial information, since a change in topology implies a change in other geometric aspects, while the opposite is not true. Generally speaking, topological properties are those that do not change after transformations like rotation, translation, scaling, and rubber sheeting.

Historical Background

Topological relationships have been studied extensively in a number of diverse disciplines since the beginning of the 1990s, achieving theoretical results which have constituted the basis for the definition of most of the topological operators today being part of the SQL dialects supported by commercial DBMSs (e.g., IBM-DB2, Oracle, PostGIS/PostgreSQL, ...). The implementations are all based on the OpenGIS Consortium specifications [8] and ISO/TC 211 standard.

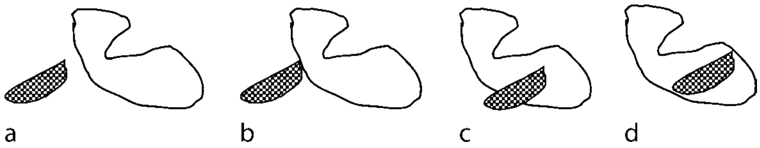
Foundations

The mathematical background behind the published contributions about topological relationships is constituted either by the *point set topology* or *spatial logic*. The study of topological relationships also depends on the embedding space, prevalently assumed to be the two-dimensional Euclidean space.

The major results that appeared in the literature can be clustered in the *three* main groups briefly discussed below.

Topological Relationships for Simple Objects

At the conceptual level, spatial objects can be modeled as *points*, *lines*, and *areas*. *Simple lines* are one-dimensional, continuous features embedded in the plane with two end points; *simple areas* are two-dimensional point sets topologically equivalent to a closed disc (Fig. 2).



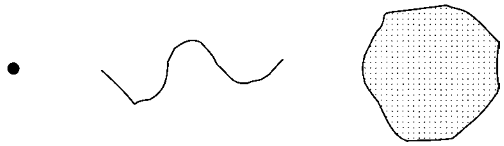
Topological Relationships. Figure 1. Examples of binary topological relationships between objects A and B (the biggest): (a) <A, *disjoint*, B>, (b) <A, *meet*, B>, (c) <A, *overlap*, B>, and (d) <A, *inside*, B>.

The two conceptual approaches, upon which almost all publications in this field have been based, are the *9-Intersection model* [5] and the *RCC model* [3]. Despite rather different foundations (the former relies on point set topology [6], the latter on spatial logic), both methods come to very similar results. Further relevant contributions belonging to this group are: [1,2,4].

The model proposed, in 1991, by Egenhofer and Franzosa, [4], for classifying topological relationships between pairs of 2D area features represents the starting point of the research in the field and the basis for the efficient implementation of the theory on top of commercial query languages [10]. Their classification is based on the cross intersection of the boundaries and interiors of the two features. The four values are collected into a two-by-two matrix, called the 4-intersection; while the approach is called the *4-IntersectionMethod* (4IM).

Topological Relationships for Complex Objects

An important advancement of the results about topological relationships, with respect to those based on the assumption of simple objects, was achieved by extending the definitions of point, line, and area, in order to take into account finite sets of isolated points as a single complex point feature, lines having separations, more than two end-points, and possibly self-intersections, and, finally, complex areas having both separations and holes (Fig. 3). In the reality, complex features are far more common than simple ones: [1,11].



Topological Relationships. Figure 2. Examples of a simple point, a simple line, and a simple area.

Topological Relationships for Objects with Vague Boundary

The models belonging to the previous two groups are applicable only to features whose geometry is exactly known (often called *crisp* spatial objects). Examples of crisp objects are mainly man-made artifacts like land parcels, buildings, and roads. But the reality reveals that the boundaries and extent of most spatial objects cannot be precisely determined. Examples of non-crisp features are: population density, vegetation, oceans, clouds, soil type, Spanish speaking areas, etc.

Three main alternatives have been proposed to model non-crisp spatial objects:

1. Models based on *fuzzy sets*. They allow a fine-grained modeling of vague spatial objects but are computationally rather expensive with respect to data structures and algorithms.
2. Models based on *rough sets*. They work with lower and upper approximations of spatial objects.
3. Models based on *crisp spatial objects*. They extend data models, type systems, and concepts for crisp spatial objects to vague spatial objects.

A discussion of the differences of these approaches can be found in [9], together with links to pertinent references.

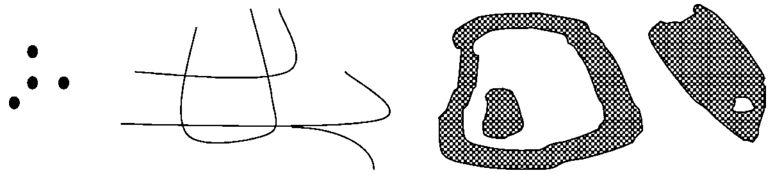
Table 1 summarizes the different subfields in the study of topological relationships discussed above.

Key Applications

The following applications are some examples among the many that benefit from dealing with topological relationships.

Geographic Information Systems (GISs)

Topological queries are relevant when dealing with spatial data. Today's GIS applications use a huge amount of spatial data. Formal models and efficient algorithms are of primary importance to reach optimal solutions.



Topological Relationships. Figure 3. Examples of a complex point, a complex line, and a complex area.

Topological Relationships. Table 1. The different subfields behind the study of topological relationships

Geometry	Boundary
Simple	Crisp
	Vague
Complex	Crisp
	Vague

Qualitative Spatial Reasoning (QSR)

Topological relations capture the everyday commonsense knowledge of space. QSR makes this knowledge explicit, so that, given appropriate reasoning techniques, a computer can make predictions about spatial relations in a qualitative manner, without recourse to an intractable or unavailable quantitative model [3].

Geospatial Semantic Web (GSW)

The Geospatial Semantic Web has become one of the most prominent research themes in geographic information science. In fact, the wide availability of geo-referenced data on the web would make it possible to index and query information based on the spatial attributes. This approach would be facilitated by using qualitative spatial relations, such as topological relations, since people are much more inclined to query web pages through natural language, instead of metric measurements.

Future Directions

Despite of the huge amount of theoretical studies about topological relationships done so far, there is still a lot of work to be done.

For example, the mapping of the topological relationships into operators to be included in future releases of spatial query languages, as well as the development of processing strategies for their efficient evaluation are still open issues. A notable contribution in this direction is constituted by a very recent paper by Praing and Schneider, [10].

Furthermore, major attention needs to be paid with respect to complex objects characterized by uncertainty in order to: a) identify suitable spatial data types for their modeling, b) design a minimal set of operations and predicates defined on top of them, and c) proceed to their integration into the query language of existing DBMSs.

Experimental Results

The IBM DB2 UDB system supports the modelling and the management of spatial data through the so-called *Spatial Extender* (briefly, SE) subsystem [7] which is fully conformant with the OGC Simple Features Specification for SQL [8], starting with version 8.2.

DB2 SE supports four different spatial data formats: a) Well-known text (WKT) representation, b) Well-known binary (WKB) representation, c) Shape representation, and d) Geography Markup Language (GML) representation.

Table below provides two examples according to the WKT text representation (the numerical values represent X-Y coordinates).

Geometry type	WKT representation	Comment
Point	`point(10,20)`	A 2D point
Polygon	`polygon((0 0,0 40,40 40,40 0,0 0))`	A 2D polygon

`{ST_Geometry, ST_Point, ST_LineString, ST_Polygon, ST_GeometryCollection, ST_MultiLineString,ST_MultiPolygon ST_MultiPoint}` is the set of geometric data types being part of the SQL/DDL language running under DB2 SE.

DB2 SE implements a long list of spatial functions conceptually grouped into five macro-categories: a) *data exchange format functions, b) comparison functions, c) functions that return information about properties of geometries, d) functions that derive new geometries from existing ones, and e) miscellaneous functions.* The *comparison functions* implement the topological operators.

In order to give the flavour of how they look like and how easily they can be called as part of SQL statements, a spatial database storing descriptive and spatial data about sites of interest and counties is taken into account.

The SQL/DDL scripts below provide the definition of the corresponding tables according to the DB2 SE syntax:

```
CREATE TABLE sitesOf_interest (id SMALLINT, geometry ST_POINT);
CREATE TABLE counties (id SMALLINT, geometry ST_POLYGON);
```

The SQL/DML scripts below insert 1 and 2 tuples into the previous tables, respectively:

```
   INSERT INTO sitesOf_interest (id,
geometry)
   VALUES (1, ST_Point(10,20,1)), (2,
ST_Point(41,41,1));
   INSERT INTO counties (id, geometry)
   VALUES (100, ST_Polygon('polygon ((0 0,0
40,40 40,40 0,0 0))', 1));
   INSERT INTO counties (id, geometry)
   VALUES (200, ST_Polygon('polygon ((1 1,1
40,40 40,40 1,1 1))', 1));
```

Notice that 1 identifies the *spatial reference system* for the resulting geometry.

The (incomplete) list of available *topological operators are:*

```
   ST_Disjoint, ST_Touches, ST_Equals,
ST_Contains, ST_Overlaps, ST_Crosses, etc.
```

An example of SQL usage of ST_Contains follows: Syntax:

```
   ST_Contains(geometry1,geometry2)
```

Meaning:

`ST_Contains` returns `1 if geometry1` contains `geometry2, 0` otherwise.

The query: *Determine the counties where the points of interest are located in.*

```
   SELECT poly.id AS polygon_id, pts.id AS
   point_id
   FROM sitesOf_interest pts, counties poly
   WHERE ST_Contains (poly.geometry, pts.
   geometry)
```

In summary, RDBMSs with spatial extensions like those featured by the IBM DB2 SE are:

- Reach of data types and functions to deal with geometry. This extends significantly the expressiveness of the relational data model and of SQL. It follows that writing ad hoc applications in high level programming languages, to make spatial analysis, is much easier than before
- Not problematic to be used for people accustomed to use SQL

Cross-references

▶ Dimension-Extended Topological Relationships
▶ Spatial Data Types
▶ SQL

Recommended Reading

1. Clementini E. and Di Felice P. A model for representing topological relationships between complex geometric features in spatial databases. Inf. Sci., 90(1–4):121–136, 1996.
2. Clementini E., Di Felice P., and van Oosterom P. A small set of formal topological relationships suitable for end-user interaction. In Proc. 3rd Int. Symp. Advances in Spatial Databases, 1993, pp. 277–295.
3. Cohn A.G., Bennett B., Gooday J., and Gotts N. RCC: a calculus for region based qualitative spatial reasoning. GeoInformatica, 1:275–316, 1997.
4. Egenhofer M.J. and Franzosa R. Point-set topological spatial relations. Int. J. Geogr. Inf. Syst., 5(2):161–174, 1991.
5. Egenhofer M.J. and Herring J. Categorizing binary topological relationships between regions, lines, and points in geographic databases. Technical report, Department of Surveying Engineering, University of Maine, 1991.
6. Gaal S. Point Set Topology. Academic Press, New York, NY, 1964.
7. IBM DB2. Spatial extender user's guide and reference (Vers. 8). 2004.
8. Open Geospatial Consortium. OpenGIS simple features specification for SQL. OpenGIS Project Document, 99–049, 1999.
9. Pauly A. and Schneider M. Topological predicates between vague spatial objects. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005, pp. 418–432.
10. Praing R. and Schneider M. Efficient implementation techniques for topological predicates on complex spatial objects. GeoInformatica, 2007.
11. Schneider M. and Behr T. Topological relationships between complex spatial objects. ACM Trans. Database Syst., 31(1): 39–81, 2006.

Topology

▶ Topological Data Models

Toponyms

▶ Gazetteers

Tour

▶ Dynamic Graphics

TP

▶ XML Tree Pattern, XML Twig Query

TP Monitor

▶ Transactional Middleware

TPQ

▶ XML Tree Pattern, XML Twig Query

Traditional Concurrency Control for Replicated Databases

Bettina Kemme
McGill University, Montreal, QC, Canada

Synonyms

Traditional replica and concurrency control strategies; Traditional data replication

Definition

Since the beginnings of distributed computing, the database community has developed strategies for replicated data management. The basic idea is that each "logical" data item has one or more physical data copies, also called replicas, that are distributed across the database servers in the system. Early work on data replication provided a framework to describe transactions in a replicated environment and developed concurrency control mechanisms to control their execution. The formalism and the techniques developed in this early work have been the foundations for much of the further research on database replication. It considered strong consistency requirements where the replicated system behaves similar to a non-replicated system. *Replica control* was introduced as the task of translating the read and write operations of transactions on logical data items into operations on the physical data copies. One-copy-serializability was developed as a first – and very strong – correctness criterion defining when the concurrent execution of transactions in a replicated system is equivalent to a serial execution of these transactions over a single logical copy of the database. Replica control was combined with *concurrency control mechanisms* in order to provide one-copy-serializable transaction execution.

Historical Background

Replication became a hot topic in the early 1980s. In their book "Concurrency Control and Recovery in Database Systems" [2], Bernstein et al. presented a thorough formalism to reason about the correctness of transaction execution and concurrency control mechanisms in central, distributed and replicated database systems. Their definitions of serializability and one-copy-serializability (1SR) are still used to reason about the correctness of transactional systems. Early work on replication took as baseline concurrency control strategies used in non-replicated or distributed databases, extended them and combined them with replica control in order to provide one-copy-serializability [2,3]. Furthermore, the correctness of execution despite site or communication failures has been analyzed thoroughly [1,4]. Work done in this early phase is very visible in textbooks on database systems and distributed systems [6], and builds part of the foundations of academic education in this area. In 1996, Gray et al. [5] indicated that these traditional approaches provide poor performance and do not scale as they commit transactions only if they have executed all their operations on all (available) physical data copies. Many advanced replication schemes have been developed since then. Nevertheless, they reuse many of the base techniques developed in the traditional replication algorithms.

Foundations

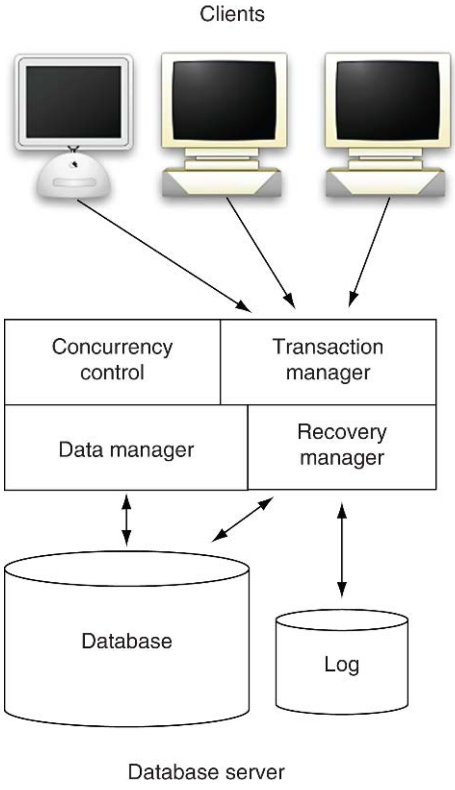
Transactions in a Non-Replicated System

The formalism that describes transactions and their execution in a replicated database is derived from the transaction model in a non-replicated system. In a non-replicated system, a database consists of a set of data items x,y,... . A transaction T_i is a sequence of read operations $r_i(x)$ and write operations $w_i(x)$ on data items that build a logical unit. The database system should provide the transactional properties atomicity, consistency, isolation and durability (see the entry *ACID properties*). Among them, atomicity and isolation require actions in a replicated system that go beyond the tasks in a non-replicated system.

Atomicity means that a transaction T_i either terminates with a commit operation (indicated as c_i) and all its write operations remain effective in the database, or with an abort operation (indicated as a_i), in which case all already executed write operations are undone before the transaction terminates.

Isolation requires that even if transactions execute concurrently in the system, each transaction should have the impression it executes isolated on the data. In particular, when two operations conflict, i.e., they are from different transactions, want to access the same data item and at least one is a write, the execution order matters. Given a set of transactions, a history describes the order in which the database server executes the operations of these transactions. The traditional correctness criterion in a non-replicated system is *serializability.* It requires a history to be equivalent to a serial history where the same transactions are executed serially one after the other. Equivalence typically refers to executing all conflicting operations in the same order.

The execution of transactions is controlled by several components of the database system (see Fig 1). The client starts a transaction and then submits the individual operations of the transaction (including a final commit or abort request). These requests are intercepted by the transaction manager which keeps track of active transactions. The individual read and write operations are forwarded to the concurrency control module that controls when operations are executed by the data manager in order to provide serializability. The recovery manager makes changes persistent at commit time or triggers undo operations in case of abort. In real database systems, clients submit SQL statements that can access and manipulate many records of different tables. However, the abstraction into simple read and write operations on data items allows for a clear and powerful reasoning framework.



Traditional Concurrency Control for Replicated Databases. Figure 1. Transaction components in non-replicated database systems.

Transaction Execution in a Replicated System

In a replicated database, there is a set of database servers *A*, *B*,..., also referred to as sites, and each logical data item *x* has a set of physical copies x^A, x^B, ... where the index refers to the server on which the copy resides. In full replication, each data item has a copy on each server, while using partial replication, it has only copies on a subset of the servers.

Execution Model As replication should be transparent to clients they continue to submit operations on the logical data items. Replica control has to map an operation $o_i(x)$, $o_i \in \{r,w\}$, of transaction T_i into operations on the physical copies of *x*, e.g., $o_i(x^A), o_i(x^B), \ldots$. Given the mapping for a set of transactions, when executing these transactions in the replicated system each database server *A* produces a local history showing the execution order of all the operations performed on the copies maintained by *A*.

The most common execution model for transactions in a replicated environment is to perform a read operation on one data copy while write operations update all copies. This is referred to as ROWA (or read-one-write-all). As most database applications typically have more read than write operations is makes sense to provide fast read access and only penalize write operations.

A problem of ROWA is that if one copy is not accessible, write operations cannot be performed anymore on the data item. In order to be able to continue even if failures occur, ROWAA (read-one-write-all-available) needs to be used. It does not require to perform updates on copies that are currently not available. An alternative are quorum protocols that require both read and write operations to access a quorum of copies.

Isolation One-copy-serializability was the first correctness criterion for replicated histories. The execution of a set of transactions in a replicated environment is one-copy-serializable if it is equivalent to a serial execution over a single, non-replicated (logical) database. Defining equivalence is not straightforward since the replicated system executes on physical copies while the non-replicated on the logical data items. The handling of failures further complicates the issue.

Atomicity Guaranteeing atomicity in a replicated system requires that all sites that have performed operations on behalf of a transaction agree on the outcome (commit or abort) of a transaction. Traditional replication solutions typically achieve atomicity by running a commit protocol at the end of a transaction. Commit protocols, such as the Two-Phase-Commit Protocol, are a special form of agreement protocol in a distributed system. The challenge here is to define a protocol that works correctly in the presence of crash and network failures.

This entry does not further look at failures but focuses on providing isolation in a ROWA system.

Replica and Concurrency Control in a Replicated System

There are many ways to distribute concurrency and replica control tasks across the system. Figure 2a shows a centralized architecture. Each individual server is only responsible for data and recovery management. Additionally, there is one central transaction manager, one concurrency control module and one replica control module in the system. Together, they control the execution of all operations in the system. They could be all located in one of the data servers, or, as shown in the figure, in a special middleware. Figure 2b shows a distributed architecture where each site has its own transaction manager, concurrency and replica control modules. Decisions on where and when to execute operations are made locally at each site. A hybrid approach is shown in Fig 2c. Each database server has its traditional transaction and concurrency control modules. A middleware layer controls transactions globally and performs replica control. It relies partially on the concurrency control modules of the individual servers. As this might not be enough for a globally correct execution, the middleware performs some additional scheduling. Other combinations of distribution are also possible.

Given a concurrency control mechanism developed for a non-replicated system, there exist many ways to extend it to a replicated system. The following depicts a few examples.

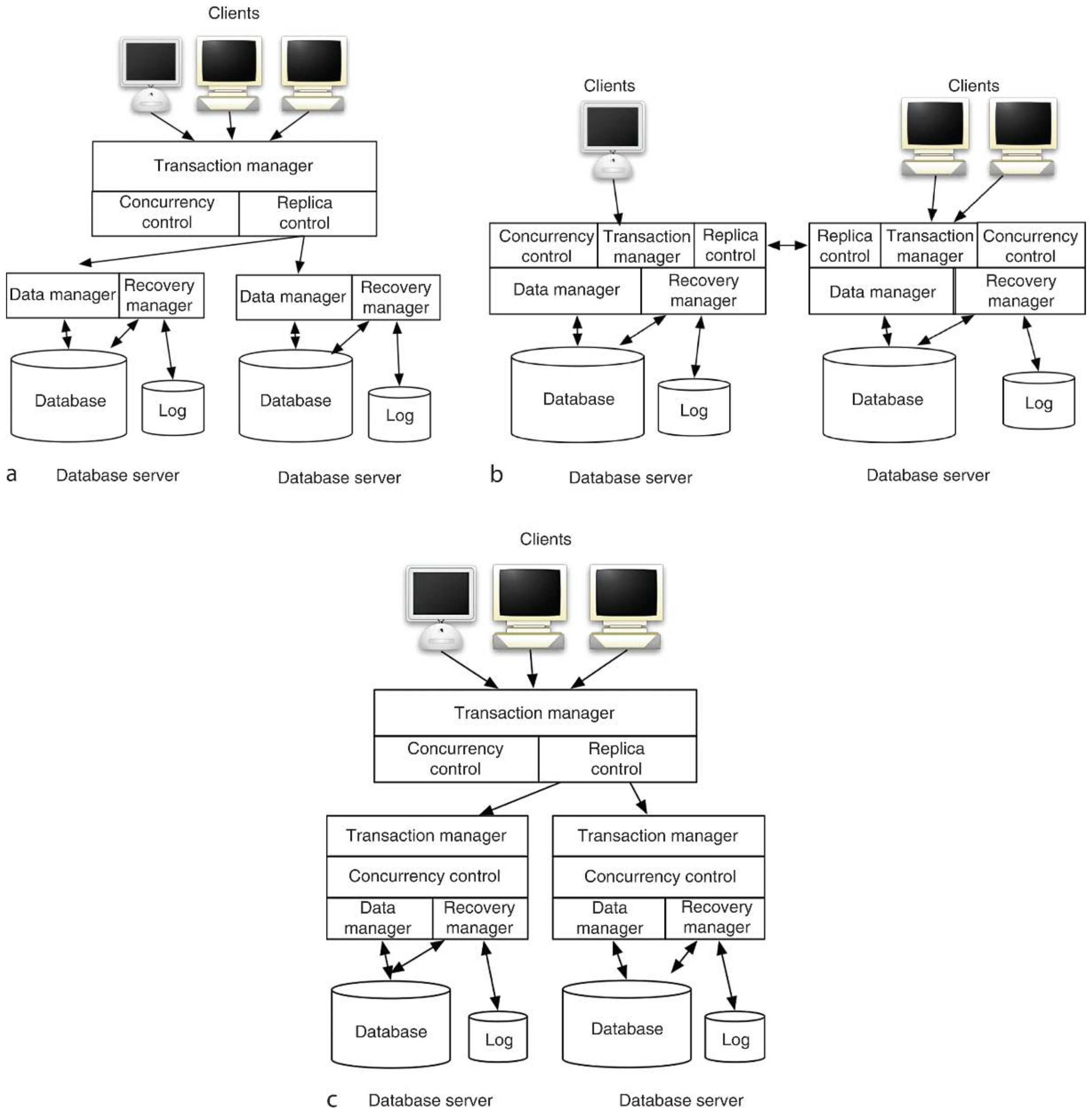
Strict Two-Phase Locking (S2PL) is probably the best known concurrency control mechanism. In this case, the concurrency control module implements a lock manager. Using S2PL, a transaction has to acquire a shared lock on data item x before performing a read on x, and an exclusive lock on x before writing x. An exclusive lock on x conflicts with other shared and exclusive locks on x. If a transaction requests a lock and another transaction holds a conflicting lock, the requesting transaction has to wait. Only when a transaction terminates it releases all its locks, which then can be granted to waiting transactions. S2PL guarantees serializability because the order in which locks are granted for the first pair of conflicting operations between two transactions determines the serialization order. Deadlocks can occur. A deadlock involving two transactions can happen if the transactions have two pairs of conflicting operations and execute them in different order.

Applying S2PL in a replicated system with a centralized architecture (Fig 2a), clients submit theiroperations to the global transaction manager. The transaction manager gets the appropriate lock via the lock manager and then the replica control module transfers each read operation to one database server with a copy of the data item, and write operations to all servers with copies. In the distributed architecture (Fig 2b), a client connects to any site. The execution of individual operations is illustrated in Fig 3a. The figure shows the message exchange between a client and two sites. When the client submits a read operation on logical data item x to the local server A, a local shared lock is acquired on the local physical copy ($sl(x^A)$) and the read operation executes locally. If it is a write operation, an exclusive lock (xl (x^A)) is acquired locally and the operation executes locally. At the same time, the operation is forwarded to the server B. B, upon receiving the request, acquires a lock on the local copy, performs the operation, and sends a confirmation back to A. When A has received all confirmations, it sends the confirmation back to the client.

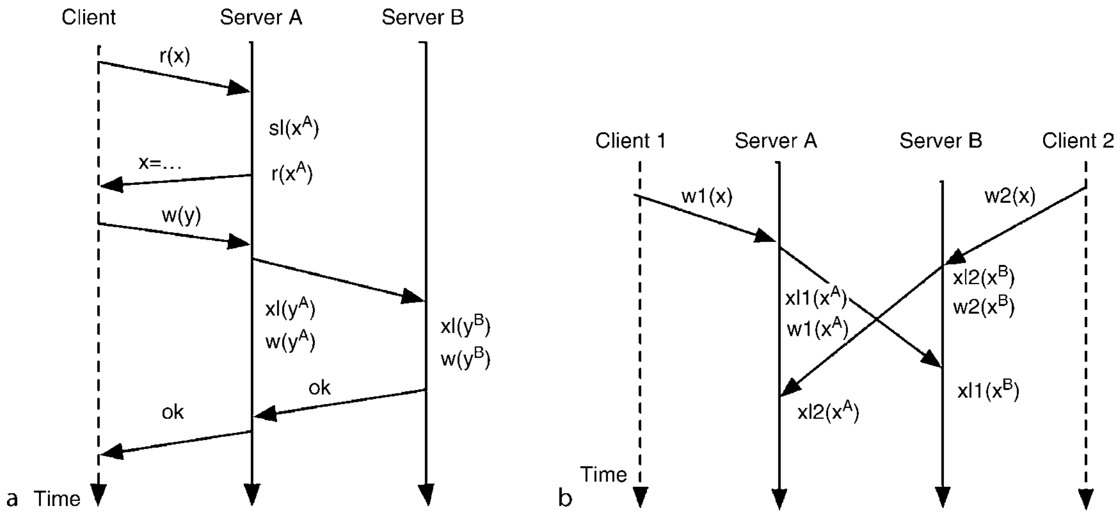
Architectural Comparison Comparing how well S2PL maps to the two architectures reflects well the principle trade-offs between a centralized and a decentralized architecture.

In favor of a centralized architecture. In principle, a central component makes the design of coordination algorithms often simpler. It directs the flow of execution, and has the *global knowledge* of where copies are located. One central concurrency control module serializes all operations. In the distributed architecture, the flow of execution is more complex as no single component has the full view of what is happening in the system.

While the centralized architecture acquires one lock per each operation on a logical data item, the distributed architecture acquires locks per data copies. Thus, a write operation involves many exclusive locks, adding to the complexity. The distributed architecture has the additional disadvantage of potential *distributed deadlocks*: there is no deadlock at any site locally but globally, a deadlock has occurred. Figure 3b depicts an example execution where a distributed deadlock occurs although the transactions both only access a single data item (something not even possible with the centralized architecture). *T*1 first acquires the lock on server *A* which forwards the request to server *B*. Concurrently,



Traditional Concurrency Control for Replicated Databases. Figure 2. Concurrency Control and Replica Control Architecture.



Traditional Concurrency Control for Replicated Databases. Figure 3. Distributed Transaction Execution.

*T*2 acquires the lock first on *B* and then requests it on *A*. At *A*, *T*2 has to wait for *T*1 to release the lock, on *B*, *T*1 waits for *T*2. The deadlock is distributed since no single server observes a deadlock. Such deadlocks need to be resolved via timeout or a deadlock detection mechanism, which in turn, could be implemented centrally or distributed.

In favor of a decentralized architecture. A central middleware is a potential bottleneck and a single point of failure. In contrast, in the distributed architecture, if ROWAA is used, the system can continue executing despite the failure of individual sites.

Furthermore, the middleware approach has an extra level of indirection. In the above example algorithm, this leads to four messages per read operation (a pair of messages between clients and middleware, and a pair between middleware and one database server). In contrast, the distributed architecture has two messages (between the client and one database server).

A further disadvantage is that the global controller does not have access to the data manager modules of the database servers. For example, assume clients submit SQL statements. The middleware cannot know what records will actually be accessed by simply looking at the SQL statement. Such information is only available during the execution. Thus, the central lock manager might need to set locks on the entire relation. In contrast, the distributed architecture can execute an SQL statement first locally, lock only the tuples that are updated, and then forward the update requests on the specific records to the other servers, allowing for a finer-grained concurrency control. Generally, a tighter coupling often allows for a better optimization.

Optimistic Concurrency Control A last example looks at optimistic concurrency control (OCC) [7]. In a non-replicated system, a write operation on *x* generates a local copy of *x*. A read on *x* either reads the local copy (if the transaction has previously written *x*) or the last committed version of *x*. At commit time of a transaction T_i, validation is performed. If validation succeeds, a write phase turns T_i's local copies into committed versions and T_i commits. Otherwise, T_i aborts. In the simplest form of OCC, validation and write phase are executed in a critical section. The validation order determines the serialization order. Therefore, validation of T_i fails if there is a committed transaction T_j that is concurrent to T_i (committed after T_i started), and T_j's writeset (data items written by T_j) overlaps with T_i's readset (data items read by T_i). As T_i validates after T_j it should be serialized after T_j, and thus, read what T_j has written. In the concurrent execution, however, it might have read an earlier version. Therefore, it needs to be aborted. Optimistic execution assumes conflicts are rare, and thus, it is sufficient to detect them at the end of transaction.

One possible implementation of OCC in a system using full replication uses the hybrid architecture of

Fig 2c. Upon the first operation of a transaction, the middleware starts a transaction and then executes all operations at one of the database servers. The local OCC of the server retrieves the latest committed versions and keeps track of local copies. At commit time the middleware retrieves the read- and writesets from the server at which the transaction executed, and performs validation. For that, it has to keep track of the writesets of previously committed transactions and use some timestamping mechanism to determine concurrent transactions. If validation succeeds, the write phase is triggered at all database servers with copies. Concurrency control is distributed: the middleware performs validation and ensures execution within a critical section while the local concurrency control is needed for the generation of local copies.

A distributed OCC strategy is proposed in [3]. It integrates validation into the commit protocol performed for atomicity. Therefore, a transaction can first execute completely locally at one database server and only at commit time communication takes place. This reduces the message overhead considerably.

Key Applications

Although the exact algorithms developed in this early phase of research are barely found in any system, the fundamental techniques behind the coordinated execution are used widely. For example, although few commercial solutions actually provide one-copy-serializability, the concept of ordering conflicting operations to some degree is common. Locking, timestamping, multi-version management, OCC, and distributed and centralized solutions are common place in the management of replicated data.

Experimental Results

Gray et al. [5] show that traditional approaches do not scale well. They analyzed the distributed locking approach and determined that the potential of deadlock increases quickly with the number of replicas in the system, the message overhead becomes too high, and transaction response times become too long. The goal of more recent research into replica and concurrency control has aimed at reducing the overhead by either providing lower levels of correctness or by developing more efficient ways to control the flow of execution in the system.

Cross-references

▶ ACID Properties
▶ Concurrency Control – Traditional Approaches
▶ Distributed Concurrency Control
▶ One-Copy-Serializabilty
▶ Replica Control
▶ Replicated Database Concurrency Control
▶ Replication based on Group Communication
▶ Replication for High Availability
▶ Replication for Scalability
▶ Transaction Models – the Read/Write Approach
▶ Two-Phase Locking

Recommended Reading

1. Bernstein P.A. and Goodman N. An algorithm for concurrency control and recovery in replicated distributed databases. ACM Trans. Database Syst., 9(4):596–615, 1984.
2. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison Wesley, Reading, MA, USA, 1987.
3. Carey M.J. and Livny M. Conflict detection tradeoffs for replicated data. ACM Trans. Database Syst., 16(4):703–746, 1991.
4. El Abbadi A. and Toueg S. Availability in partitioned replicated databases. In Proc. 5th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1986, pp. 240–251.
5. Gray J., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
6. Kindberg T., Coulouris G.F., and Dollimore J. Distributed Systems: Concepts and Design, 4th edn. Addison Wesley, Reading, MA, USA, 2005.
7. Kung H.T. and Robinson J.T. On optimistic methods for concurrency control. ACM Trans. Database Syst., 6(2):213–226, 1981.

Traditional Data Replication

▶ Traditional Concurrency Control for Replicated Databases

Traditional Replica and Concurrency Control Strategies

▶ Traditional Concurrency Control for Replicated Databases

Trajectory

Ralf Hartmut Güting
University of Hagen, Hagen, Germany

Synonyms

Trajectory; Moving point

Definition

Representation of a time dependent position observed over some period of time. Usually represented as a polyline in a 3D (2D + time) space for an object moving in the 2D plane.

Key Points

Trajectories describe complete histories of movement; they are stored in *moving objects databases*, sometimes called *trajectory databases* in the literature.

When operations are included, a trajectory corresponds to a value of a *moving point* data type. Queries on databases containing trajectories can be formulated using *moving object languages*. When uncertainty about an object's precise position is taken into account, an uncertain trajectory [3] results which can be viewed as a kind of cylindrical volume in the 2D + time space. There exists a lot of work on indexing trajectories [2]. Querying for trajectories similar to a given one is also an important research area [1].

Cross-references

- ▶ Moving Objects Databases and Tracking
- ▶ Spatio-Temporal Data Types

References

1. Chen L., Özsu M.T., and Oria V. Robust and fast similarity search for moving object trajectories. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 491–502.
2. Mokbel M.F., Ghanem T.M., and Aref W.G. Spatio-temporal access methods. Bull. TC Data Eng., 26(2):40–49, 2003.
3. Trajcevski G., Wolfson O., Hinrichs K., and Chamberlain S. Managing uncertainty in moving objects databases. ACM Trans. Database Syst., 29(3):463–507, 2004.

Trajectory Databases

▶ Moving Objects Databases and Tracking

Trajectory Indexing

▶ Indexing Historical Spatio-Temporal Data

Transaction

Gottfried Vossen
University of Münster, Münster, Germany

Synonyms

Transaction; ACID transaction

Definition

A transaction is a tool for application programmers to delegate the responsibility for preventing damage to data from threats such as concurrent execution, partial execution, or system crashes to the database system software; at the same time, application programmers retain the obligation to think about the impact on data consistency of the code they are writing, when executed alone and without failures. From a programmer's perspective, the power of the transaction paradigm hence lies in the fact that it reduces the task of concurrent failure-aware programming of the entire system to that of correct sequential programming of each application program separately. The transaction concept offers the ACID properties (short for *atomicity, consistency preservation, isolation*, and *durability*) and materializes through concurrency control and recovery. It is nowadays used beyond database systems.

Key Points

Database *transactions* go back to the work of Gray et al. [3,4] in the mid-1970s. Their development has been driven by applications where programs run against data stored in a single database system or a collection of such systems. There are many threats to the overall dependability of a system formed as a combination of databases and application programs; database transactions deal with the threats from concurrent execution, from incomplete execution (e.g., due to crashes or cancelations), and from system crashes that lose information from volatile buffers that has not yet been saved. The problem itself has not only been recognized in the context of database applications [1], yet has finally been solved by the notion of a transaction.

The key point of a transaction is that it comes with system-guaranteed properties collectively known as the *ACID properties* which considerably simplify the development of OLTP applications in that application programs can safely ignore a major portion of the system complexity. In particular, application programs are completely freed up from taking care of the issues of *concurrency*, i.e., effects that may result from concurrent or even parallel program executions and especially data accesses, and of *failures*, i.e., effects that would result from program executions being interrupted at random points due to process or computer failures.

In order to make this work, database systems offer the transaction concept as well as *transaction management*, commonly broken down into *concurrency control* for the synchronization of concurrent access to common data objects from multiple transactions, as well as into *recovery* for being able to restore a consistent state of the database after a crash. From a conceptual point of view, transactions can be modeled in various ways, which essentially boil down to the *page model* as well as the *object model* [5], and they have evolved from an abstraction concept into a system mechanism nowadays offered beyond database systems [2]. The page model considers transactions and transaction management at the syntactic level of disk pages that can either be read or written, while the object model considers them at a level where the semantics of operations on database objects can be taken into account.

Transactions are executed in interleavings called *schedules* or *histories*, which need to satisfy a correctness criterion that commonly comes in a form of *serializability*. Serializability is based on the perception that serial executions are correct and hence tries to make a non-serial execution "look" as if it was run serially. Similar approaches can be applied to both page-model as well as object-model transactions.

Cross-references

- ▶ ACID Properties
- ▶ Concurrency Control
- ▶ Crash Recovery
- ▶ Extended Transaction Models
- ▶ Serializability
- ▶ Transaction Management
- ▶ Transaction Manager
- ▶ Transaction Models – The Read/Write Approach

Recommended Reading

1. Davies C.T. Data processing spheres of control. IBM Syst. J., 17:179–198, 1978.
2. Elmagarmid A.K. Database Transaction Models for Advanced Applications. Morgan Kaufmann, San Francisco, CA, 1992.
3. Eswaran K.P., Gray J., Lorie R.A., and Traigerv I.L. The notions of consistency and predicate locks in a database system. Commun. ACM, 19:624–633, 976.
4. Gray J., Lorie R.A., Putzolu G.R., and Traiger I.L. Granularity of locks in a large shared data base. In Proc. 1st Int. Conf. on Very Large Data Bases, 1975, pp. 428–451.
5. Weikum G. and Vossen G. Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2002.

Transaction Chopping

DENNIS SHASHA
New York University, New York, NY, USA

Definition

Transaction chopping is a technique for improving the concurrent performance of a database system by reducing the time locks are held. The idea is to break up each transaction into smaller "pieces," such that each piece executes as a transaction, but the effect is as if the original transactions executed serializably.

Key Points

Imagine an application that locks the entire database and then accesses a few rows in a table that does not fit into memory. If one looked at the resource statistics, one would find low CPU consumption and low disk utilization, yet the throughput would be very bad. For example, if ten pages were accessed even at 1 ms per page, then throughput would be only 100 transactions per second. The point is that it is possible to slow down performance greatly just because of a poor locking strategy, even if there are plenty of resources.

One might consider a less extreme case: a transaction that processes an order. The transaction might check whether there is sufficient cash available, then add the appropriate quantities to inventory, subtract the value from cash, and commit the transaction. Because any application that invokes this transaction will access the "cash" data item, that data item may become a bottleneck. Transaction chopping is a technique for circumventing such bottlenecks by dividing transactions into smaller transactional pieces that will hold locks for only a short time, yet still preserve the serializability of the original transactions.

Assumptions

Transaction chopping makes the following main assumptions: (i) One can characterize all the transactions that will run in some time interval. The characterization may be parameterized. For example, one may know that some transactions update account balances and branch balances, whereas others check account balances. However, one need not know exactly which accounts or branches will be updated. (ii) The goal is to achieve the guarantees of serializability, while obtaining as much concurrency as possible. That is, one would like either to use degree 2 isolation, snapshot isolation, or to chop transactions into smaller pieces. The guarantee should be that the resulting execution be equivalent to one in which each original transaction executes serializably. (iii) If a transaction makes one or more calls to rollback, one knows when these occur and can arrange the transaction to move them towards the beginning.

Basic Definitions

A *chopping* partitions each T_i into *pieces* $c_{i_1}, c_{i_2}, \ldots c_{i_k}$. Every database access performed by T_i is in exactly one piece. A chopping of a transaction T is said to be *rollback-safe* if either T has no rollback statements or all the rollback statements of T are in its first piece. The first piece must have the property that all its statements execute before any other statements of T. This will prevent a transaction from half-committing and then rolling back. All transactions should be *rollback-safe*. Each piece will act like a transaction in the sense that each piece will acquire locks according to some standard method that guarantees the serializability of the piece.

For example, suppose T updates an account balance and then updates a branch balance. Each update might become a separate piece, acting as a separate transaction.

Correct Choppings

One may characterize the correctness of a chopping with the aid of an undirected graph having two kinds of edges. (i) *C edges*: C stands for *conflict*. Two pieces p and p' from different original transactions conflict if there is some data item x that both access and at least one modifies. In that case, one draws an edge between p and p' and label the edge C. (ii) *S edges*: S stands for *sibling*. Two pieces p and p' are siblings if they come from the same transaction T. In this case, draw an edge between p and p' and label the edge S.

The resulting graph is called the *chopping graph*. (Note that no edge can have both an S and a C label.) A chopping graph has an *SC-cycle* if it contains a simple cycle that includes at least one S edge and at least one C edge. A chopping of $T_1, T_2, \ldots, T_n$ is *correct* if any execution of the chopping is equivalent to some serial execution of the original transactions. "Equivalent" is in the sense of the serializability entry.

Theorem 1: *A chopping is correct if it is rollback-safe and its chopping graph contains no SC-cycle.*

Theorem 1 shows that the goal of any chopping of a set of transactions should be to obtain a rollback-safe chopping without an SC-cycle.

Conclusion

Transaction chopping is a method to enhance concurrency by shortening the time a bottleneck resource is held locked. It works well in practice and the theory extends naturally to snapshot isolation as well as read committed isolation levels.

Cross-references

- ► Concurrency Control
- ► Locking
- ► Transaction Execution

Recommended Reading

1. Fekete A., Liarokapis D., O'Neil E., O'Neil P., and Shasha D. Making Snapshot Isolation Serializable. ACM Trans. Database Syst., 30(2):492–528, 2005.
2. Shasha D. and Bonnet P. Database Tuning : Principles Experiments and Troubleshooting Techniques. Morgan Kaufmann, 2002.

Transaction Commit Time

► Transaction Time

Transaction Execution

► Concurrency Control – Traditional Approaches

Transaction Management

Gottfried Vossen
University of Münster, Münster, Germany

Synonyms

Concurrency control and recovery; Transaction scheduling; Transaction processing

Definition

Transaction management [2,6] refers to the tasks of processing multiple transactions issued by various clients of a database server in such a way that the ACID contract can be fulfilled, that is, the properties of *atomicity, consistency preservation, isolation,* and *durability* of each individual transaction can be guaranteed. Transaction management is generally understood as requiring serializability-based concurrency control as well as recovery from failures. Concurrency control is the task of scheduling transactions such that their serializability can be guaranteed, while recovery has to restore a consistent database state after a system or media failure. Assuming that the database server is in charge of the "C," the former guarantees the "I" in ACID, the latter the "A" and "D" properties. Transaction management has to be highly efficient, as modern transaction servers need to accommodate thousands of transactions per minute. This is achieved by a comprehensive combination and interplay of theoretical research and practical developments.

Historical Background

Transaction management emerged in the 1970s in early database management systems [5], and has become an indispensable part of a database server [1,3]. Its goal is to devise efficient algorithms for handling transactions, the essential "contract" between an application program and transactional server that combines a number of requests to the server into a logical unit. Over the years, transaction management has received theoretical underpinnings as well as system implementations in various ways, the former of which have allowed it to extend beyond simple database objects (pages) [7]. Indeed, transaction management can essentially be "positioned" at any level of abstraction within the functional layers of a data server, where the price to pay for expressiveness typically is efficiency. Nevertheless, transaction management has also been extended into areas beyond database management, among them operating systems and, more recently, programming languages.

Foundations

To recognize the essence of what transaction management is about, consider the operation of a bank that uses a relational database to keep track of its account business. The database may contain a table *Accounts* which describes bank accounts in terms of their account id, associated customer name, identification of the respective bank branch, balance, and possibly other data. Transactions in the bank are either *withdrawals* or *deposits* (debit/credit transactions) applied to *Accounts*, and these operations are often combined into *funds transfers*, i.e., withdrawals from one account and immediate deposit into another. With a huge number of clients potentially issuing simultaneous requests to the bank's database server, a *concurrent* execution of multiple debit/credit transactions is mandatory in order to exploit the server's hardware resources and to achieve processing speeds acceptable to clients. Concurrently executing transactions are typically modifying the underlying database of the banking application (table *Accounts*) frequently. In order to be able to ignore the potential fallacies of this concurrency, each transaction would ideally be executed as if there were no other transactions, but that would mean no concurrency. This tradeoff between concurrency for the sake of performance on the one hand, and potential sequential execution for the sake of correctness on the other, is one aspect of transaction processing, formally captured through notions of *serializability* and reconciled by the *concurrency control techniques* of a transactional server.

Figure 1 illustrates that concurrency may indeed be tricky, since it may have a disastrous impact on the consistency of the underlying data. Consider two debit/credit transactions t_1 and t_2 that are concurrently executed and that are both operating on the same account x (which could be the account id). To distinguish the two different instances of the local variable "balance" that temporarily holds the value of the account balance, they can be referred to as "balance1" for transaction t_1 and "balance2" for t_2. The first transaction then intends to withdraw $30, the second transaction intends to deposit $20, and it is assumed that the initial account balance is $100. The table in Fig. 1 shows those

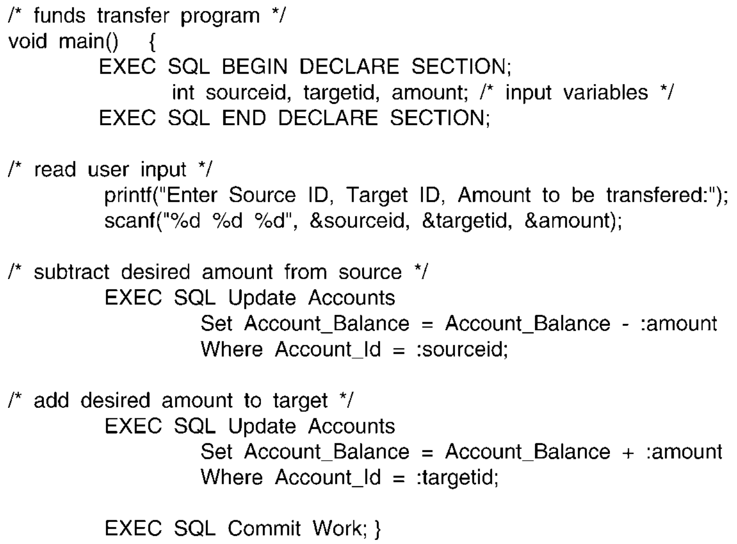
Transaction t_1	Time	Transaction t_2
/* balance1 = 0, x.Account_balance = 100, balance2 = 0 */		
Select account_balance Int:balance1 from account Where account_Id = x	1	
/* balance1 = 100, x.Account_balance = 100, balance2 = 0 */		
	2	Select account_balance Int:balance2 from account Where account_Id = x
/* balance1 = 100, x.Account_balance = 100, balance2 = 100 */		
balance1 = balance1 - 30	3	
/* balance1 = 70, x.Account_balance = 100, balance2 = 100 */		
	4	balance2 = balance2 + 20
/* balance1 = 70, x.Account_balance = 100, balance2 = 120 */		
Update account Set account_balance = :balance Where account_Id = x	5	
/* balance1 = 70, x.Account_balance = 70, balance2 = 120 */		
	6	Update account Set account_balance = :balance2 Where account_Id = x
/* balance1 = 70, x.Account_balance = 120, balance2 = 120 */		

Transaction Management. Figure 1. Concurrent transactions requiring concurrency control.

parts of the two transactions that read and modify the account record. Upon completion of the execution, the balance of account *x*, as recorded in the persistent database, will be $120, although it should be $90 after execution of the two transactions. Thus, the recorded data are incorrect, a kind of "anomaly" must be prevented, and concurrent executions must be treated with care. Similar anomalies can arise from transaction failures.

A second fundamentally important point is that the various accesses a transaction has to perform need to occur *in conjunction*: Once a transaction has begun execution, its data accesses should look to the outside world as an atomic operation which is either executed completely or not at all. This property of atomicity should be guaranteed even in a failure-prone environment where individual transactions or the entire database server may fail at an arbitrary point in time. To this end, a transactional server needs to provide *recovery techniques* to cope with failures. In addition to ensuring transaction atomicity, these techniques also serve to ensure the *durability* of a transaction's effects once the transaction is completed. The scenario shown in Fig. 2 illustrates this. It shows a program which transfers a given amount of money between two accounts, by first withdrawing it from a source account and then depositing it in a target account. The program is described in terms of SQL statements embedded into a C program. It is assumed that the funds transfer program has started executing and has already performed the withdraw operation (i.e., the first SQL `Update`). If there is a hardware or software failure that interrupts the program's execution at this point, the remaining second update operation will not be performed. Thus, the target account will not receive the money.

A recovery procedure, to be invoked after the system is restarted, will try to find out which updates were already made by ongoing transaction program executions and which ones were not yet done, and will try to fix the situation. However, implementing recovery procedures on a per-application case basis is a difficult task that is itself error prone because of its sheer complexity, especially because multiple transactions issued by different programs may have accessed the data at the time of the failure. So rather than programming recovery in an ad hoc manner for each application separately, a systematic approach is needed, as described, for example, in [9,2]. System-provided recovery ensures the atomicity of transactions and simplifies the understanding of the post-failure state of the data and the overall failure handling on the application side. In the sample scenario of Fig. 2, rather than being left with the inconsistent state in the middle



```
/* funds transfer program */
void main()    {
          EXEC SQL BEGIN DECLARE SECTION;
                  int sourceid, targetid, amount; /* input variables */
          EXEC SQL END DECLARE SECTION;

/* read user input */
           printf("Enter Source ID, Target ID, Amount to be transfered:");
           scanf("%d %d %d", &sourceid, &targetid, &amount);

/* subtract desired amount from source */
           EXEC SQL Update Accounts
                      Set Account_Balance = Account_Balance - :amount
                      Where Account_Id = :sourceid;

/* add desired amount to target */
           EXEC SQL Update Accounts
                      Set Account_Balance = Account_Balance + :amount
                      Where Account_Id = :targetid;

           EXEC SQL Commit Work; }
```

Transaction Management. Figure 2. Sample funds transfer that needs atomicity.

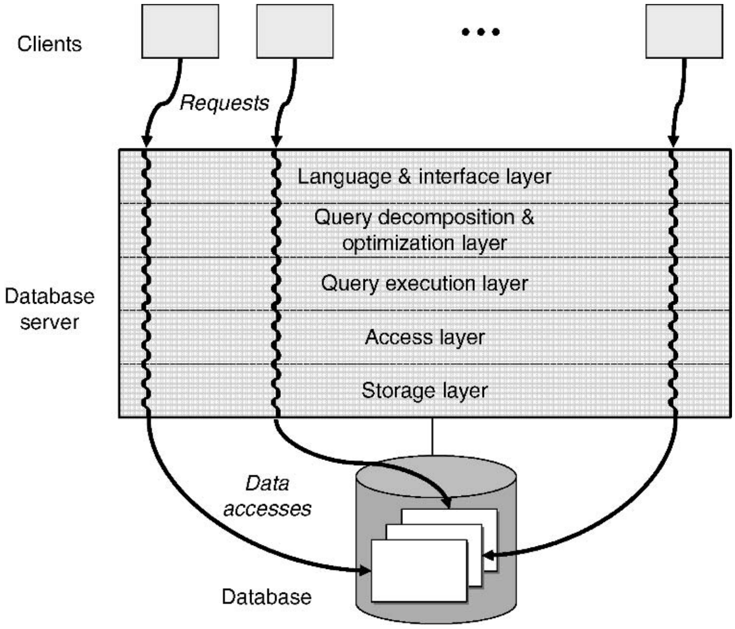
of the transaction, the system recovery will restore the state as of before the transaction began. On the other hand, if the transaction had already issued its "commit transaction" call, then the system would guarantee the durability of the transaction's complete funds transfer.

Guaranteeing the ACID properties of a transaction, which allow application developers to disregard concurrency and failures, are the major goal of transaction management; the means to accomplish this are concurrency control and recovery. These cornerstones for building highly dependable information systems can also be successfully applied outside the scope of *online transaction processing* (OLTP) applications.

Key Applications

Figure 3 shows the layered architecture shared by essentially all database servers in one form or another. When a client request arrives at the server, the server executes code that transforms the request into one or more operations at each of the underlying layers, ultimately arriving at a sequence of disk accesses (unless caching avoids the disk access). The *language and interface layer* makes various kinds of interfaces available to the application developer, usually in the form of APIs (e.g., SQL and ODBC). The *query decomposition and optimization layer* works on an internal, tree based representation of a request and is concerned with the further decomposition of the request into smaller units that can be directly executed; this is the level where also optimizations are carried out. The execution plan chosen is often represented as an operator tree where each operator can be directly mapped to a piece of server code. This code is provided by the *query execution layer*. Next, index structures and also the capabilities for accessing and manipulating data records are provided by the *access layer*. Finally, the *storage layer* is responsible for managing the pages of a database, including disk I/O and caching.

The layered architecture in Fig. 3 does not explicitly mention transaction management, for the simple reason that the functionality of transactional concurrency control and recovery can be tied to any of the five layers shown. For many applications that run simple transactions, each of which accesses a small portion of the data only, transaction management is commonly integrated into the access and storage layers. Applications in this category include classical OLTP scenarios such as banking (see above) or reservation systems, where high availability, high throughput, and high reliability are of utmost importance. However, transactions in electronic commerce applications, where client requests may span multiple databases as well as other information sources across enterprise boundaries, or transactions arising from an execution of business processes which typically take some form of semantic information (stemming from higher-level, for example SQL-type operations) into account can also benefit



Transaction Management. Figure 3. Functional database system layers.

from transaction concepts. Through appropriate transaction models and an appropriate adaptation of the relevant concurrency control and recovery algorithms to these models, transaction management can be applied in almost any area that needs to handle concurrent requests to shared resources in such a way that the ACID properties can be guaranteed [9].

Future Directions

With constant changes in network-centric computing, including the proliferation of Web services, long-running processes across organizational boundaries, large scale peer-to-peer publish-subscribe and collaboration platforms, and ambient-intelligence environments with large numbers of mobile and embedded devices, support for handling or even masking concurrency and component failures is critical in many modern application areas, but can no longer use traditional atomicity concepts alone. In open systems applications are constructed from pre-existing components, and these components and their configurations are not known in advance and they can change on the fly. Thus, it is crucial that atomicity properties of components become composable and allow for reasoning about the behavior of the resulting system. Transaction management will thus continue to require research for the foreseeable future.

Experimental Results

Since transaction management typically has to meet high efficiency requirements, experimental evaluation and comparison of algorithmic approaches to concurrency control and recovery has always been crucial, as is tuning of the transaction management component of a database server for the system administrator. Prominent references including implementation recipes as well as experimental evaluations are [4,8].

Cross-references

- ▶ Control
- ▶ Crash Recovery
- ▶ Database Server
- ▶ Performance Analysis of Transaction Processing Systems
- ▶ Transaction
- ▶ Two-Phase Locking

Recommended Reading

1. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, MA, 1987.
2. Bernstein P.A. and Newcomer E. Principles of Transaction Processing for the Systems Professional. Morgan Kaufmann, San Francisco, CA, 1997.
3. Cellary W., Gelenbe E., and Morzy T. Concurrency Control n Distributed Database Systems. North–Holland, Amsterdam, 1988.

4. Gray J. (ed.). The Benchmark Handbook for Database and Transaction Processing Systems. 2 edn. Morgan Kaufmann, San Francisco, CA, 1993.
5. Gray J., Lorie R.A., Putzolu G.R., and Traiger I.L. Granularity of locks in a large shared data base. In Proc. 1st Int. Conf. on Very Data Bases, 1975, pp. 428–451.
6. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1993.
7. Lynch N., Merritt M., Weihl W., and Fekete A. Atomic Transactions. Morgan Kaufmann, San Francisco, CA, 1994.
8. Shasha D. and Bonnet Ph. Database Tuning: Principles Experiments and Troubleshooting Techniques. Morgan Kaufmann, San Francisco, CA, 2002.
9. Weikum G. and Vossen G. Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2002.

Transaction Management in Distributed Database Systems

► Distributed Transaction Management

Transaction Manager

Andreas Reuter[1,2]
[1]EML Research gGmbH, Villa Bosch, Heidelberg, Germany
[2]Technical University Kaiserslautern, Kaiserslautern, Germany

Definition

The transaction manager (TxM) is a special resource manager that implements the resource type "transaction." It handles all the state transitions that a transaction can perform. For simple ACID transactions these are: begin, savepoint, rollback, abort, prepare commit; for more refined transaction models there will be additional states. The resource managers register with the TxM when they get employed by a transaction for the first time. Thus, the TxM keeps a record for each transaction of which resource managers have been involved with it. In the same vein, it stores which sessions each transaction has used in case it has performed operations on other nodes in a distributed system. The TxM closely interacts with the concurrency control manager who can make transaction wait for other transactions in case of access conflicts. It also interacts with the logging and recovery subsystem and the communications manager.

Key Points

The TxM keeps track of all active transactions. When a transaction starts, it is assigned a transaction ID that is unique across any distributed system and will never repeat. For each transaction the TxM stores which resource managers and which sessions are associated with it; for sessions the polarity (outgoing/incoming) is also important. For each state transition of a transaction, the TxM will orchestrate the proper protocol. The most important of these protocols is the two-phase commit (2PC) protocol. Depending on the structure of a transaction, the TxM will act as a coordinator or a participant – or both. Using the information about the participating resource managers and communication sessions, the TxM calls them at their proper callback entries to perform all the necessary actions to go, for example, into the prepared state. TxMs are often implemented as interpreters of the formal description of a state machine, so they can dynamically switch between different (optimizations of) commit protocols.

Upon recovery, the TxM is the first resource manager that is reconstructed by the recovery manager. When the recovery manager has completed its initial backward scan of the online log, the table of active transactions is completely recovered. If there are any in-doubt transactions left, the TxM starts handling that particular part of the 2PC in order to resolve those transactions.

Cross-references

► Communications Manager
► Concurrency Control Manager
► Logging/Recovery Subsystem
► Storage Resource Management
► Transactional Middleware

Recommended Reading

1. Gray J. and Reuter A. Transaction Processing – Concepts and Techniques; Morgan Kaufmann, San Mateo, CA, 1993.

Transaction Model

► Transaction Models – The Read/Write Approach

Transaction Models – the Read/Write Approach

Gottfried Vossen
University of Münster, Münster, Germany

Synonyms

Transaction model; Page model; Read/write model

Definition

The transaction concept essentially establishes an "ACID contract" in data-processing situations, and a transaction model is an abstraction concept that makes this concept amenable to realizations. Two fundamental models are the *page model* as well as the *object model*, where the former is an execution model and the latter is more a conceptual model. The page model is based on the perception that database operations ultimately are read or write operations on pages that need to be transferred between secondary storage and main memory or the database buffer. The model allows making all relevant notions (in particular interleavings of multiple transactions and schedule correctness) precise in a syntactic manner, and it forms the basis for verifying a variety of concurrency control algorithms that can be used in database management as well as other systems.

Historical Background

The read/write or page model of transactions goes back to the work of Jim Gray [5,6] and Kapali Eswaran et al. [4]. Related notions of atomic actions have been discussed by others around the same time, the mid- to late-1970s, including [8,9]. The page-model transaction concept became the subject of intensive theoretical studies, in particular in the work of Christos Papadimitriou and, independently, Phil Bernstein et al. around 1980 [2,3,10,11]. It is essentially valid in its original form until today, yet has received a number of extensions and variations over the years [13].

Foundations

The read/write model of database transactions is motivated by the observation that all operations on data (i.e., queries as well as updates) are eventually mapped into indivisible read and write operations on disk pages or on a page cache in main memory. Thus, to study the effects of concurrent executions, it essentially suffices to inspect the interleavings of the resulting page operations. The abstraction from higher-level data operations, such as SQL commands or method invocations on business objects, down to the view that a resulting transaction consists of reads and writes *only* is a strong one, yet suffices for many practical purposes. In fact, a comprehensive theory of concurrency control and of recovery can be built on it, which is directly applicable to real-world systems, albeit with some performance limitations.

Definition of a Transaction

To define the model of read/write transactions formally, a database is assumed to contain a (finite) set $D = \{x,y, z,...\}$ of (indivisible and disjoint) items (*pages*) with indivisible read and write operations. Data items are often denoted by small letters from the end of the alphabet, where indices are used if necessary (e.g., x_1,y_4). A transaction can then be defined as a total or a partial order of steps; for total orders, a formal definition is as follows: A *transactiont* is a (finite) sequence of steps (actions) of the form $r(x)$ or $w(x)$, written $t = p_1...p_n$, where $n < \infty$, $p_i \in \{r(x), w(x)\}$ for $1 \leq i \leq n$, and $x \in D$ for a given database D. Here, r stands for "read" and w for "write." Thus, a transaction abstracts from the details of a program execution and instead focuses on the sequence of read and write operations that results from that execution.

Each step occurring in a transaction can be uniquely identified so that two distinct transactions do not have steps in common. Let p_j denote the *j*th step of a given transaction, and let p_{ij} denote the *j*th step of transaction i in the presence of multiple transactions. Then the following terminology is common: If $p_j = r(x)$, the interpretation is that step j reads data item x; if $p_j = w(x)$, the interpretation is that step j writes data item x. A transaction is hence a purely syntactic entity whose semantics or interpretation is unknown. In the absence of information on the semantics of the program that launches a transaction, however, the best that can be done is to devise a *Herbrand semantics*, i.e., a "syntactic" interpretation of the steps of a transaction that is as general as possible: (i) In case $p_j = r(x)$, the current value of x is assigned to a local variable v_j. (ii) In case $p_j = w(x)$, a possibly new value, computed by the respective program, is written into x. Each value written by a transaction t potentially depends on the values of *all* data items that t has previously read; if t writes x,

x is the return value of an arbitrary but unknown function f_j applied to all values read before the respective write step. The Herbrand semantics of a read/write transaction thus is a technical vehicle, only useful for making various correctness notion for transaction interleavings precise.

The *total ordering* requirement to the steps of a transaction can be relaxed into a *partial ordering*; thus, a *transaction t* becomes a partial order of steps (actions) of the form $r(x)$ or $w(x)$, where $x \in D$ and reads and writes as well as multiple writes applied to the same data item are ordered. More formally, a *transaction* is a pair $t = (\mathrm{op}, <)$, where op is a finite set of steps of the form $r(x)$ or $w(x)$, $x \in D$, and $< \subseteq \mathrm{op} \times \mathrm{op}$ is a partial order on set op for which the following holds: If $\{p, q\} \subseteq \mathrm{op}$ s.t. p and q both access the same data item and at least one of them is a write step, then $p < q \vee q < p$. In other words, in the partial ordering of a transaction's steps, a read and write operation on the same data item or two write operations on the same data item need to be ordered. With "conflicting" steps inside a transaction being ordered, the "semantics" outlined above for totally ordered transactions carries over to partially ordered ones. For simplicity, however, most discussions of correctness criteria in the literature stick to total orders. Although read/write transactions are considered syntactic entities only, it is an advantage of this model that its theory can be developed in the absence of semantic information and hence can be used for *every possible interpretation* of the transactions. In other words, the read/write page model is fairly general despite its simple structure.

The model as described above allows a transaction to read or write the same data item more than once, as it is the case in the example $t = r(x)w(x)r(y)r(x)w(x)$. Here t reads and writes x twice, although it is reasonable to assume that the value of x remains available, after having been read the first time, in the local variables of the underlying program for as long as it is needed by t, and that only the last write step determines the final value of x produced by this transaction. To exclude "redundancies" of this kind, it is common to assume that (i) in each transaction each data item is read or written at most once, and (ii) no data item is read (again) after it has been written. The latter condition does not exclude the possibility of "blind writes," which is a write step on a data item that is not preceded by a read of that data item.

Schedules and Histories

The distinction between partial and total orderings is also appropriate for interleavings of transactions, i.e., for schedules and histories: Let $T = \{t_1,\ldots,t_n\}$ be a (finite) set of transactions, where each $t_i \in T$ has the form $t_i = (\mathrm{op}_i, <_i)$, with op_i denoting the set of operations of t_i and $<_i$ denoting their ordering, $1 \leq i \leq n$. A *history* for T is a pair $s = (\mathrm{op}(s), <_s)$ s.t. (i) $\mathrm{op}(s) \subseteq \mathrm{U}_{i=1}^{n}\ \mathrm{op}_i \cup \mathrm{U}_{i=1}^{n}\ \{a_i, c_i\}$ and $\mathrm{U}_{i=1}^{n}\ op_i \subseteq op(s)$, i.e., s consists of the union of the operations from the given transactions plus a termination operation, which is either a c_i (commit) or an a_i (abort), for each $t_i \in T$, (ii) $(\forall i, 1 \leq i \leq n)\ c_i \in \mathrm{op}(s) \Leftrightarrow a_i \notin \mathrm{op}(s)$, i.e., for each transaction, there is either a commit or an abort in s, but not both, (iii) $\mathrm{U}_{i=1}^{n} <_i \subseteq <_s$, i.e., all transaction orders are contained in the partial order given by s, (iv) $(\forall i, 1 \leq i \leq n)(\forall p \in \mathrm{op}_i) p <_s a_i$ or $p <_s c_i$, i.e., the commit or abort operation always appears as the last step of a transaction, and (v) any pair of operations $p,q \in \mathrm{op}(s)$ from distinct transactions accessing the same data item s.t. at least one is a write operation is ordered in s in such a way that either $p <_s q$ or $q <_s p$. A *schedule* is a prefix of a history. A history s is *serial* if for any two transactions t_i and t_j in it, where $i \neq j$, all operations from t_i are ordered in s before all operations from t_j or vice versa. Thus, a history (for partially ordered transactions) has to contain all operations from all transactions (i), needs a distinct termination operation for every transaction (ii), preserves all orders within the transactions (iii), has the termination steps as final steps in each transaction (iv), and orders "conflicting" operations (v). The view that two operations which access the same data item and of which at least one is a write operation are in conflict is identical to the notion of conflict that will shortly be used as the basis for a notion of serializability. The notions "schedule" and "history" are not always used in the sense defined here in the literature, but are often used as synonyms.

Schedule Correctness

For transaction executions, it is important to not only have a model of interleavings and their constituents, but also to fix a notion of schedule or history *correctness.* To this end, *conflict serializability* (CSR) is the notion which is most important for the practice of transactional information systems, in particular for designing scheduling algorithms and for building schedulers. CSR is computationally easy to test, and it has a number of interesting theoretical properties which can justify an exploitation of this concept in practice in a variety of ways. Finally, it can be generalized to other transaction models and different data settings. Conflict serializability is based on a simple notion of conflict which was mentioned already in connection with partially ordered histories, and which is appropriate for the syntactical nature of read/write transactions. Two data operations from distinct transactions are *in conflict* in a schedule if they access the same data item and at least one of them is a write. The notion of conflict gives rise to a notion of serializability: A history is *conflict serializable* if there exists a serial history with the same conflicts.

Commutativity of Operations

Conflict serializability can be characterized via acyclic conflict graphs, which gives rise to efficient testability. It is can also be characterized via *commutativity rules* for page model data operations. In these rules "$\sim$" means that the ordered pair of actions on the left-hand side can be replaced by the right-hand side, and vice versa: (C1) $r_i(x)r_j(y) \sim r_j(y)r_i(x)$ if $i \neq j$, i.e., two read steps $r_i(x)$ and $r_j(y)$, $i \neq j$, which occur in a schedule in this order and are adjacent (with no other operation in between), may be commuted. (C2) $r_i(x)w_j(y) \sim w_j(y)r_i(x)$ if $i \neq j$, $x \neq y$, i.e., a read and write step can be exchanged if they are from distinct transactions and access different data items. (C3) $w_i(x)$ $w_j(y) \sim w_j(y)w_i(x)$ if $i \neq j$, $x \neq y$, i.e., two write steps can be commuted if they refer to different data items. These commutativity rules can be applied to a given schedule or history in a stepwise fashion, for example to transform $s = w_1(x)r_2(x)w_1(y)w_1(z)r_3(z)w_2(y)w_3(y)$ $w_3(z)$ into $w_1(x)w_1(y)w_1(z)r_2(x)w_2(y)r_3(z)w_3(y)w_3(z)$, thereby proving it equivalent to the serial history $t_1t_2t_3$. The above transformations have implicitly assumed that operations in a schedule are totally ordered. With partial orders, one may need an additional transformation rule which simply states that two unordered operations can be arbitrarily ordered if they are non-conflicting.

The commutativity rules can be used for introducing another relation on schedules: Let s and s' be two schedules s.t. op(s) = op(s'). Define $s \sim s'$ if s' can be obtained from s by a single application of the commutativity rules to the steps of the schedule. Let "$^*\sim$" denote the reflexive and transitive closure of "$\sim$," i.e., $s *\sim s'$ if s' can be obtained from s by a finite number of applications of the rules. It turns out that "$\overset{*}{\sim}$" is an equivalence relation on the set of all schedules for a given set of transactions, and that finitely many applications of the rules may transform a given history into a serial one, as in the example above. More formally, a history s is *commutativity based reducible* if there is a serial history s' s.t. $s \overset{*}{\sim} s'$, i.e., s can be transformed into s' through a finite number of allowed transformation steps according to the commutativity rules. Therefore, a history s is commutativity based reducible iff s is CSR. An important application of this alternative characterization of schedule correctness is that it can immediately be generalized. Indeed, it is irrelevant to know of a schedule whether a step reads or writes or which data item it accesses, as long as it is known which steps of the schedule are in conflict. The latter suffices for deciding about the correctness of the schedules, even without any knowledge about the meaning of the operations [11]. Thus, the steps of a schedule or history to which a commutativity argument is applied may be of a completely different type, such as *increment* and *decrement* on a counter object, *push* and *pop* on a stack object, or *enqueue* and *dequeue* on a queue object. This observation can be exploited in the context of extended transaction models, where notions of serializability can be established that are based on semantic information.

The page model of transactions can also be extended to accommodate transaction recovery. Indeed, when a transaction aborts, the intuitive reaction of the underlying system is to "undo" the effects of that transaction, which can adequately be captured by inverse write operations. To execute an abort operation, the system would have to execute inverse writes in reverse order of their occurrence [1].

Key Applications

Key applications for transactions started out from debit/credit scenarios in banks and financial institutions, where multiple customer activities against shared accounts need to be synchronized, yet executed at a high throughput rate. In a debit/credit transaction, the activities are either *withdrawals* or *deposits*, and these are often combined into *funds transfers*. The typical structure of a debit/credit program is shown below, using SQL commands embedded into a C program. Note the distinction between local variables of the invoked program and the data in the underlying database that is shared by all programs.

```
/* debit/credit program */
void main()
{EXEC SQL BEGIN DECLARE SECTION;
 int accountid, amount;
  /* input variables */
 int balance; /* intermediate variable */
 EXEC SQL END DECLARE SECTION;
/* read user input */
 printf("Enter Account ID, Amount for
 deposit (positive) or withdrawal (nega-
 tive): ");
 scanf("%d %d", &accountid, &amount);
/* determine current balance of the ac-
count, reading it into a local variable
of the program */
 EXEC SQL
 Select Account_Balance Into :balance
 From Accounts
 Where Account_Id = :accountid;
/* add amount (negative for withdrawal) */
balance = balance + amount;
/* update account balance in the database */
 EXEC SQL Update Accounts
  Set Account_Balance = balance
  Where Account_Id = :accountid;
EXEC SQL Commit Work; }
```

The crucial situation is that, say, two transactions access the same account, where one makes a withdrawal and the other a deposit, i.e., both transactions write new values of the account balance. If unsynchronized, these transaction could overwrite each other's results, thereby leaving the account in an inconsistent state. Another crucial situation is that a transaction makes a withdrawal from one account and a deposit into another (i.e., a transfer); if this transaction crashes in the middle, i.e., after the withdrawal but before the deposit, money would be lost if the system is not capable of restoring the state prior to the withdrawal. The read/write model allows for a straightforward formalization of these situations, and efficient concurrency control as well as recovery procedures can be devised to handle them.

For scenarios like the one just sketched (in particular transfers that need to appear atomic to the outside world), there are numerous applications in today's information systems landscape of electronic business over the Internet, where client requests are rarely restricted to single data servers, but often span multiple databases and other information sources across enterprise boundaries, yet the mutual consistency of all this data is crucial and thus important to maintain. Then, the resulting transactions operate in a distributed system that consists of multiple servers, often with heterogeneous software. To go one step further, a *business process* is a set of *activities* (or steps) that belong together in order to achieve a certain business goal. Business processes are also at the heart of advanced, business-to-consumer (B2C) or business-to-business (B2B) services on the Internet, for example, electronic auctions (B2C) or supply chains (B2B). Typical examples would be the processing of a credit request or an insurance claim in a bank or insurance company, respectively, the administrative procedures for real estate purchase, or the "routing" of a patient in a hospital.

Future Directions

Transactions have originally been developed in the database system community, but their usage and potential benefits are by no means limited to database management. Not surprisingly, the transaction concept is also being explored in the operating systems and programming languages communities. Recent trends include, for example, enhancing the Java language with a notion of atomic blocks that can be defined for methods of arbitrary classes. This could largely simplify the management of concurrent threads with shared objects, and potentially also the handling of failures and other exceptions. The run-time environment could be based on an extended form of *software transactional memory*. Another important direction in ongoing research is to design and reason about guarantees that resemble transactions but are weaker than the ACID properties, especially with regard to the notion of isolation. A model that is widely deployed in industrial data management systems is *snapshot isolation* (based on keeping a versioned history of data items).

Experimental Results

The read/write model of transactions has in part been so successful since it allows for highly efficient implementations of concurrency control and recovery methods, and it allows for tuning a system on the fly. Surveys are provided by [7] and in particular [12].

Cross-references

- ▶ Atomicity
- ▶ Concurrency Control
- ▶ Extended Transaction Models
- ▶ Logging and Recovery
- ▶ Multi-version Serializability and Concurrency Control
- ▶ Serializability
- ▶ Software Transactional Memory
- ▶ Transaction
- ▶ Transaction Chopping

Recommended Reading

1. Alonso G., Vingralek R., Agrawal D., Breitbart Y., El Abbadi A., Schek H.-J., and Weikum G. Unifying concurrency control and recovery of transactions. Inform. Syst., 19:101–115, 1994.
2. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, MA, 1987.
3. Bernstein P.A., Shipman D.W., and Wong W.S. Formal aspects of serializability in database concurrency control. IEEE Trans. Software Eng., SE-5:203–216, 1979.
4. Eswaran K.P., Gray J., Lorie R.A., and Traiger I.L. The notions of consistency and predicate locks in a database system. Commun. ACM, 19:624–633, 1976.
5. Gray J. Notes on database operating systems. In: Operating Systems: An Advanced Course, LNCS, Vol. 60, R. Bayer, M.R. Graham, G. Seegmüller (eds.). Springer, Berlin Heidelberg New York, 1978, pp. 393–481.
6. Gray J. The transaction concept: virtues and limitations. In Proc. 7th Int. Conf. on Very Data Bases, 1981, pp. 144–154.
7. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1993.
8. Lampson B.W. Atomic transactions. In: Distributed Systems – Architecture and Implementation: An Advanced Course, LNCS, Vol. 105, B.W. Lampson, M. Paul, H.J. Siegert (eds.). Springer, Berlin Heidelberg New York, 1981.
9. Lomet D.B. Process structuring, synchronization, and recovery using atomic actions. ACM SIGPLAN Notices, 12(3):128–137, 1977.
10. Papadimitriou C.H. The Serializability of concurrent database updates. J. ACM, 26:631–653, 1979.
11. Papadimitriou C.H. The Theory of Database Concurrency Control. Computer Science, Rockville, MD, 1986.
12. Shasha D., Bonnet Ph. Database Tuning – Principles, Experiments, and Troubleshooting Techniques. San Francisco, CA, Morgan Kaufmann, 2003.
13. Weikum G. and Vossen G. Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2002.

Transaction Processing

- ▶ Application Recovery
- ▶ Transaction Management

Transaction Scheduling

- ▶ Transaction Management

Transaction Service

- ▶ Transactional Middleware

Transaction Time

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Registration time; Extrinsic time; Physical time; Transaction commit time; Belief time

Definition

A database fact is stored in a database at some point in time, and after it is stored, it remains current, or part of the current database state, until it is logically deleted. The *transaction time* of a database fact is the time when the fact is current in the database. As a consequence, the transaction time of a fact is generally not a time instant, but rather has duration.

The transaction time of a fact cannot extend into the future. Also, as it is impossible to change the past, meaning that (past) transaction times cannot be changed.

In the context of a database management system that supports user transactions, the transaction times of facts are consistent with the serialization order of the transactions that inserted or logically deleted them. Transaction times may be implemented using transaction commit times, and are system-generated and -supplied.

Key Points

A database is normally understood to contain statements that can be assigned a truth value, also called facts, that are about the reality modeled by the database and that hold true during some non-empty part of the time domain. Transaction times, like valid times, may be associated with such facts. It may also be noted that it is possible for a database to contain the following different, albeit related, facts: a non-timestamped fact and that fact timestamped with a valid time. The first would belong to a snapshot relation, and the second would belong to a valid-time relation. Both of these facts may be assigned a transaction timestamp. The resulting facts would then be stored in relations that also support transaction time.

A transaction time database is append-only and thus ever-growing. To remove data from such a database, *temporal vacuuming* may be applied.

The term "transaction time" has the advantage of being almost universally accepted and it has no conflicts with the other important temporal aspect of data, valid time.

The Oracle DBMS explicitly supports transaction time. Applications can access prior transaction-time states of their database, by means of transaction time-slice queries. Database modifications and conventional queries are temporally upward compatible.

Concerning the alternatives, the term "registration time" seems to be straightforward. However, this term may leave the impression that the transaction time is only the time instant when a fact is inserted into the database. "Extrinsic time" is rarely used. "Physical time" is also used infrequently and seems vague. "Transaction commit time" is lengthy, but more importantly, the term appears to indicate that the transaction time associated with a fact must be identical to the time when that fact is committed to the database, which is an unnecessary restriction. The term is also misleading because the transaction time of a fact is not a single time instant as implied. The term "belief time" stems from the view that the current database state represents the current belief about the aspects of reality being captured by the database. This term is used infrequently.

Cross-references

- ▶ Supporting Transaction Time Databases
- ▶ Temporal Compatibility
- ▶ Temporal Database
- ▶ Temporal Generalization
- ▶ Temporal Specialization
- ▶ Temporal Vacuuming
- ▶ Time Domain
- ▶ Timeslice Operator
- ▶ Transaction
- ▶ Transaction-Time Indexing
- ▶ User-Defined Time
- ▶ Valid Time

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. Springer-Verlag, Berlin, 1998, pp. 367–405.
2. Snodgrass R.T. and Ahn I. A taxonomy of time in databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1985, pp. 236–246.
3. Snodgrass R.T. and Ahn I. Temporal databases. IEEE Comput., 19(9):35–42, September 1986.

Transactional Business Processes

- ▶ Transactional Processes
- ▶ Workflow Transactions

Transactional Consistency in a Replicated Database

- ▶ One-Copy-Serializability

Transactional Middleware

Gustavo Alonso
ETH Zurich, Zurich, Switzerland

Synonyms

TP monitor; Object monitor; Application server; Transaction manager; Transaction service

Definition

Transactional Middleware is a generic term used to refer to the IT infrastructure that supports the execution of electronic transactions in a distributed setting. The best known form of transactional middleware is Transaction Processing Monitors (TP Monitors or TPM), which have been around for more than 3 decades (e.g., CICS of IBM). Today, TP Monitors are at the heart of most application servers and are a key component of any enterprise computing architecture. The main role of these systems is to run transactions, i.e., to support the illusion that certain distributed operations are executed atomically. This makes the design of complex systems easier for the programmer, who does not need to implement this functionality but can rely on the transactional middleware to ensure that groups of operations are executed in their entirety or not all, with the transactional middleware taking care of all the work necessary to do so. Historically, transactional middleware has often provided much more functionality than just transactions and, in many ways, they were some of the earliest forms of middleware. They were already used in early mainframes to connect different applications.

Historical Background

The background for transactional middleware is TP Monitors. TP Monitors originated with the first commercial applications running on mainframes and the need to provide additional functionality not provided by the operating system. As the needs of applications and developers flourished, so did the capabilities and features of TP Monitors. This is how TP Monitors became a general term to refer to complex infrastructure and collection of tools that provide not only transactional support but also specialized networking capabilities, batch jobs, job control languages, multithreading, transactional extensions to programming languages, load-balancing capabilities, transactional message queues, etc. When enterprise applications broke away from the mainframe and started running on mixed environments (e.g., a large database in the mainframe, workstations for processing, and PCs for the clients), TP Monitors proved to be the ideal platform to develop and operate such distributed systems. During the 1990s, TP Monitors dominated the middleware arena and were not only the most prominent but also the clearly dominant form of middleware in the market.

The conventional TP Monitors with extensive functionality were called TP-Heavy. These were standalone products were intended for the development of distributed, transactional applications with high performance requirements. A simpler form of TP Monitor was the so called TP-Light systems. Typically, these were extensions to database engines to support interactions with the database through RPC calls and the addition of some basic data processing capabilities to the database engine without having to resort to a full middleware layer such as that required by TP-Heavy approaches. The dominance and importance of TP-Monitors is illustrated by the struggles of CORBA systems to implement the Object Transactional Services described in the CORBA specification. Object Transactional Services are exactly the transactional functionality provided by a TP-Monitor and in many commercial systems it was implemented by using an already existing TP Monitor. Such combined systems enjoyed a short period of popularity under the name of Object Monitors or Object Transaction Monitors.

TP Monitors today are still widely used although the name TP Monitor has been replaced by others, more all-encompassing terms. For instance, applications servers have a TP Monitor as one of the central components. Platforms like .NET o J2EE also rely heavily on TP Monitor functionality. Hence, today these systems are collectively referred to as transactional middleware.

Foundations

To understand transactional middleware, one needs to understand that it is not just about transactions (or, at least, historically it was not just about transactions). Nevertheless, the core functionality of transactional middleware is to run distributed transactions in an efficient manner. Thus, the discussion starts by covering this aspect of transactional middleware and postpone the system issues till later.

Transactions are an abstraction used in database engines to implement the so called ACID properties (Atomicity, Consistency, Isolation, and Durability). Unlike what is commonly found in many textbooks, the vast majority of transactional middleware systems are not there to support the four ACID properties but only one: atomicity in distributed transactions and in the interaction between applications and databases. To the extent that records of the transaction executed are kept, transactional middleware can also provide durability but in a different sense than database durability and not as one of its main operational features. Some systems also supported isolation (e.g., Encina), but such systems run into the same difficulties that all other solutions for concurrent programming have faced and are still facing today.

Atomicity in the context of transactional middleware translates into running a 2 Phase Commit protocol among all the entities involved in executing parts of a transaction. The interactions and interfaces required to do so were standardized in the early 1990s as part of the X/Open models for transactions and distributed transactions, which also defined the XA interface. The XA interface describes how a transaction manager (the transactional middleware) interacts with a resource manager (e.g., a database) to run a 2 Phase Commit protocol. Once standardized, the XA interface allowed TP Monitors to execute transactions across heterogeneous systems, greatly simplifying the task of writing code integrating several systems.

The standard and generic procedure for running a transaction in a transactional middleware platform involves invoking some primitive that tells the middleware that a transaction needs to be started. This primitive typically hides a call to a transaction manager that records the transaction and return and identifier and a context for the transaction. The application then proceeds to make calls as part of this transaction. Each call is augmented with the transaction identifier and context, which allows the recipient of the call to realize that its execution has become part of a transaction and tells it of which one. The recipient of the call uses the transactional context it has received to register with the transactional manager. Part of this registration procedure also involves telling the transaction manager where the actual transaction will be executed (typically the part of the system that supports the XA interface). After registration, the transaction manager knows who is involved in running parts of the transaction, information that maintains in a transaction record that contains all participants in that transaction. When the original application invokes a commit on the transaction, the transaction manager uses the transaction record to start a 2 Phase Commit protocol between all the participants that have registered for this transaction. At the end of the protocol, the transaction manager informs the original application of the success or failure of the commit. If the commit was successful, the original application has the guarantee that the transaction has been executed and completed at all sites where it was invoked. If the commit fails, the original application has the guarantee that all side effects and changes made by the transaction have been rolled back. Of course, these guarantees hold only if the code that is being invoked does not cause any side effects or persistent changes outside the transaction (e.g., on a sub-system that does not support the XA interface) as those changes will not be rolled back for obvious reasons.

The system side of transactional middleware involves all the machinery necessary to run transactions plus a great deal of additional functionality to cope with distribution. Historically, transactional middleware also had to provide a wide range of functionality to compensate for the lack of support at the operating system level. A good example is support for multi-threading, which is necessary to allow the system to cope with concurrent requests to the same service. Other examples include name and directory services, load balancing, life cycle management for services, logging, domain definition, and authentication. For performance reasons, many commercial products also supported specialized network interfaces (e.g., to communicate with a mainframe) and provided additional sub-systems such as message queuing or transactional file systems. Providing such a wealth of functionality made transactional middleware, especially TP Monitors, very generic tools where sometimes the support for transaction management was not the primary reason to use the system. To a large extent, especially during the 1980s and the first half of the 1990s, transactional middleware was the most efficient way to build a distributed system and TP Monitors became not only middleware but also development platforms for distributed applications. A good example of this was Encina (the commercial version of Camelot [2]), which provided its own versions or C or C++ (Transactional C and Transactional C++).

Key Applications

High performance transaction processing (financial, on line trading, banking).

Application servers (web shops, multi tier systems).

Future Directions

TP Monitors are and will remain a key component of any enterprise computing solution. As the key functionality of TP Monitors (the ability to efficiently process large volumes of transactions) is embedded deeper and deeper within larger systems, some of the additional functionality that conventional TP Monitors always provided has migrated to other platforms (e.g., application servers) or become independent systems on their own (e.g., message brokers). Hence the generic name of transactional middleware when referring to such systems. A challenge in the future will be providing similar transactional semantics on Service Oriented Architectures where the interactions might be based on asynchronous messages instead of through blocking calls (RPC/RMI). What the proper transactional semantics are for an asynchronous enterprise bus or an event based architecture is a topic that is still open and needs attention.

Cross-references

- ► ACID Properties
- ► Advanced Transaction Models
- ► Transaction
- ► Transaction Tuning
- ► Two-Phase Commit

Recommended Reading

1. Bernstein P.A. and Newcomer E. Principles of Transaction Processing. Morgan Kaufmann, Los Altos, CA, 1997.
2. Eppinger J.L., Mummert L.B., and Spector A.Z (eds.). Camelot and Avalon. Morgan Kaufmann, Los Altos, CA, 1991.
3. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, (3rd edn.). Prentice Hall, Englewood Cliffs, NJ, 2009.
4. Reuter A. and Gray J. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, Los Altos, CA, 1993.
5. Weikum G. and Vossen G. Transactional Information Systems. Morgan Kaufmann, Los Altos, CA, 2001.

Transactional Processes

Heiko Schuldt
University of Basel, Basel, Switzerland

Synonyms

Transactional workflows; Transactional business processes

Definition

A *Transactional Process* is a partially ordered sequence of activities which is executed in a way that guarantees transactional consistency for all activities or a subset of them. Activities can be either transactional (e.g., they are again transactional processes or conventional database transactions) or non-transactional (e.g., invocations of application services). Activities are ordered by means of control flow and data flow dependencies.

Key Points

In most cases, transactional consistency for processes focuses on notions of atomicity which go beyond the "all-or-nothing" semantics of ACID database transactions. This is done by supporting a process designer in explicitly specifying how consistency has to be achieved. Models for transactional processes include support for failure handling which can either be integrated into control flow dependencies (i.e., distinguishing between regular execution and alternative executions in case of failures) or, in a rather programmatic style, as exception handlers. Failure handling also needs to take into account that certain activities can neither be compensated after having been successfully executed (i.e., to semantically undo their effects) nor be deferred until the end of a process, due to control flow dependencies. For this, backward recovery (partial compensation in case of failures) and forward recovery (alternative executions) need to be combined. Forward recovery also requires to make the state of a process persistent during execution so that it can be resumed in case of system failures. For applications in which also concurrent executions of transactional processes need to be controlled, isolation and atomicity can be jointly considered.

Commercial tools for transactional processes distinguish between microflows (short running processes where all activities are transactional) and macroflows (long-running processes). The former can actually be considered conventional database transactions and are executed using protocols such as two phase commit (2PC) while the latter have to be treated as transactional processes. Several Web services standards define protocols that allow to execute Web services in a transactional context and thus facilitate their integration into transactional processes.

Cross-references

- ► Distributed Transaction Management
- ► Generalization of ACID Properties
- ► Multilevel Transactions and Object-Model Transactions
- ► Transactional Middleware
- ► Workflow Transactions

Recommended Reading

1. Alonso G. Transactional business processes. In Process-Aware Information Systems. M. Dumas, W. Aalst, and van der A. ter Hofstede (eds.), Wiley, New York, 2005, pp. 257–278.
2. Leymann F. Supporting business transactions via partial backward recovery in workflow management systems. In Proc. Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'95), Informatik Aktuell, Dresden, Germany, March 1995. Springer Verlag, Berlin, 1995, pp. 51–70.
3. Schuldt H., Alonso G., Beeri C., and Schek H.-J. Atomicity and isolation in transactional processes. ACM Trans. Database Syst., 27(1):63–116, March 2002.
4. Zhang A., Nodine M., and Bhargava B. Global scheduling for flexible transactions in heterogeneous distributed database systems. IEEE Trans. Knowl. Data Eng., 13(3):439–450, 2001.

Transactional Workflows

- ► Transactional Processes
- ► Workflow Transactions

Transaction-Time Access Methods

► Transaction-Time Indexing

Transaction-Time Algebras

► Temporal Algebras

Transaction-Time Data Model

► Temporal Data Models

Transaction-Time Indexing

Mirella M. Moro[1], Vassilis J. Tsotras[2]
[1]The Federal University of Rio Grande do Sol, Porte Alegre, Brazil
[2]University of California-Riverside, Riverside, CA, USA

Synonyms

Transaction-time access methods

Definition

A transaction-time index is a temporal index that enables fast access to transaction-time datasets. In a traditional database, an index is used for selection queries. When accessing transaction-time databases, selection queries also involve the transaction-time dimension. The characteristics of the transaction-time axis imply various properties that such temporal index should have to be efficient. As with traditional indices, the performance is described by three costs: (i) storage cost (i.e., the number of pages the index occupies on the disk), (ii) update cost (the number of pages accessed to perform an update on the index; for example when adding, deleting or updating a record), and (iii) query cost (the number of pages accessed for the index to answer a query).

Historical Background

Most of the early work on temporal indexing has concentrated on providing solutions for transaction-time databases. A basic property of transaction-time is that it always *increases*. This is consistent with the serialization order of transactions in a database system. Each newly recorded piece of data are time-stamped with a new, larger, transaction time. The immediate implication of this property is that previous transaction times *cannot* be changed (since every new change must be stamped with a new, larger, transaction time). This is useful for applications where every action must be registered and maintained unchanged after registration, as in auditing, billing, etc. Note that a transaction-time database records the history of a database activity rather than "real" world history. As such it can "rollback" to, or answer queries for, any of its previous states.

Consider, for example, a query on a temporal relation as it was at a given transaction time. There are two obvious but inefficient approaches to support this query, namely the "copy" and "log" approaches. In the "copy" approach, the whole relation is "flushed" (copied) to disk for every transaction for which a new record is added or modified. Answering the above rollback query is simple: the system has to then query the copy that has the largest transaction-time less or equal to the requested time. Nevertheless, this approach is inefficient for its storage and update costs (the storage can easily become quadratic to the number of records in the temporal database and the update is linear, since the whole temporal relation needs to be flushed to disk for a single record update). In contrast, the "log" solution simply maintains a log of the updates to the temporal database. Clearly, this approach uses minimal space (linear to the number of updates) and minimal update cost (simply add an update record at the end of the log), but the query time is prohibitively large since the whole log may need to be traversed for reconstructing a past state of the temporal database. Various early works on transaction-time indexing behave asymptotically like the "log" or the "copy" approaches. For a worst-case comparison of these methods see [7]. Later on, two methodologies were proposed to construct more efficient transaction-time indices, namely the (i) *overlapping* [3,10] and (ii) (partially) *persistent* approaches [1,6,9,]. These methodologies attempt to combine the benefits of the fast query time from the "copy" approach with the low space and update costs of the "log" approach.

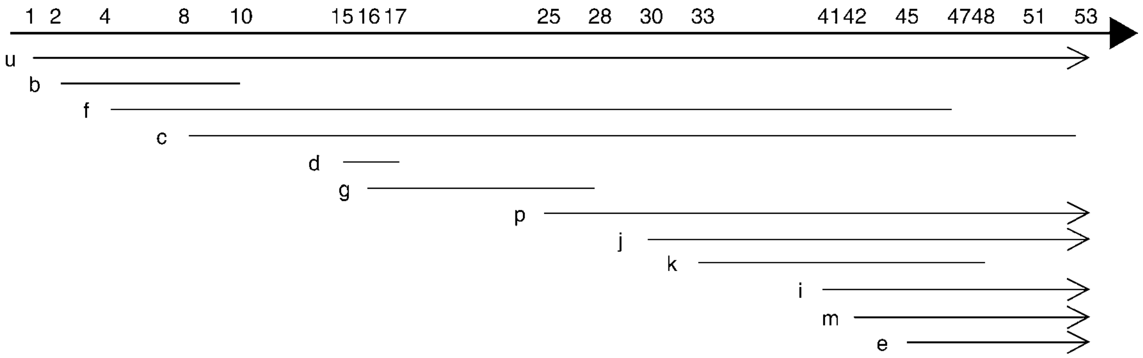
Foundations

The distinct properties of the transaction-time dimension and their implications to the index design are discussed through an example; this discussion has been influenced by [8]. Consider an initially empty collection of objects. This collection evolves over time as changes are applied. Time is assumed discrete and always increasing. A change is the addition or deletion of an object, or the value change of an object's attribute. A real life example would be the evolution of the employees in a company. Each employee has a surrogate (*ssn*) and a salary attribute. The changes include additions of new employees (as they are hired or re-hired), salary changes or employee deletions (as they retire or leave the company). Each change is time-stamped with the time it occurs (if more than one change happen at a given time, all of them get the same timestamp). Note that an object attribute value change can be simply "seen" as the artificial deletion of the object followed by the simultaneous rebirth (at the same time instant) of this object having the modified attribute value. Hence, the following discussion concentrates only on object additions or deletions.

In this example, an object is called "alive" from the time that it is added in the collection until (if ever) it is deleted from it. The set *s(t)*, consisting of all alive objects at time *t*, forms the state of the evolving collection at *t*. Figure 1 illustrates a representative evolution shown as of time *t* = 53. Lines ending to ">" correspond to objects that have not yet been deleted at *t* = 53. For simplicity, at most one change per time instant is assumed. For example, at time *t* = 10 the state is *s*(10) = *{u, f, c}*. The interval created by the consecutive time instants an object is alive is the "lifetime" interval for this object. Note that the term "interval" is used here to mean a "convex subset of the time domain" (and not a "directed duration"). This concept has also been named a "period" in this discussion however, only the term "interval" is used. In Fig. 1, the lifetime interval for object *b* is [2,10). An object can have many non-overlapping lifetime intervals.

Note that in the above evolving set example, changes are always applied to the current state *s(t)*, i.e., past states *cannot* be changed. That is, at time *t* = 7, the deletion of object *d* is applied to *s*(16) = *{u, f, c, d, g}* to create *s*(17) = *{u, f, c, g}*. This implies that, at time *t* = 54, no object can be retroactively added to state *s*(5), neither the interval of object *d* can be changed to become [15,25]. All such changes are not allowed as they would affect previous states and not the most current state *s*(53).

Assume that all the states *s(t)* of the above evolution need to be stored in a database. Since time is always increasing and the past is unchanged, a transaction time database can be utilized with the implicit updating assumption that when an object is added or deleted from the evolving set at time *t*, a transaction updates the database system about this change at the same time, i.e., this transaction has commit timestamp *t*. When a new object is added in the collection at time *t*, a record representing this object is stored in the database accompanied by a transaction-time interval of the form *[t, UC)*. In this setting, *UC* (Until Changed) is a variable representing the fact that at the time the object is added in the collection, it is not yet known when (if ever) it will be deleted from it. If this object is later deleted at time *t*', the transaction-time interval



Transaction-Time Indexing. Figure 1. An example of a transaction-time evolution.

of the corresponding record is updated to *[t, t')*. A real-world object deletion is thus represented in the database as a "logical" deletion: the record of the deleted object is still retained in the database, accompanied by an appropriate transaction-time interval. Since the past is kept, a transaction-time database conceptually stores, and can thus answer queries about, any past state *s(t)*.

Based on the above discussion, an index for a transaction-time database should have the following properties: (i) store past logical states, (ii) support addition/deletion/modification changes on the objects of the current logical state, and (iii) efficiently access and query any database state.

Since a fact can be entered in the database at a different time than when it happened in reality, the transaction-time interval associated with a record is actually related to the process of updating the database (the database activity) and may not accurately represent the times the corresponding object was valid in reality. Note that a valid-time database has a different abstraction, which can be visualized as a dynamic collection of "interval-objects." The term interval-object is used to emphasize that the object carries a valid-time interval to represent the temporal validity of some object property. Reality is more accurately represented if both time dimensions are supported. A bitemporal database has the characteristics of both approaches. Its abstraction maintains the evolution (through the support of transaction-time) of a dynamic collection of (valid-time) interval-objects.

Traditional indices like the B^+-tree or the R-tree are not efficient for transaction-time databases because they do not take advantage of the special characteristics of transaction time (i.e., that transaction time is always increasing and that changes are always applied on the latest database state). There are various index proposals that are based on the (partially) *persistent* data-structure approach; examples are the Time-Split B-tree (TSB) [6], the Multiversion B-tree (MVBT) [1], the Multiversion Access Structure [11], the Snapshot Index [9], etc. It should be noted that all the above approaches facilitate "time-splits": when a page gets full, current records from this page are copied to a new page (this operation is explained in detail below). Time-splits were first introduced in the Write-Once B-tree (WOBT), a B-tree index proposed for write-once disks [5]. Later, the Time-Split B-tree used time-splits for read-write media and also introduced other splitting policies (e.g., splitting by other than the current time, key splits etc.) Both the WOBT and TSB use deletion markers when records are deleted and do not consolidate pages with few current records. The MVBT uses the time splitting approach of the WOBT, drops the deletion markers and consolidated pages with few current records. It thus achieves the best asymptotic behavior and it is discussed in detail below. Among the index solutions based on the *overlapping* data-structure approach [2], the Overlapping B-tree [10] is used as a representative and discussed further.

For the purposes of this discussion, the so-called *range-timeslice* query is considered, which provides a key range and a specific time instant selection. For example: "find all objects with keys in range $[K_1, K_2]$ whose lifetimes contain time instant *t*." This corresponds to a query "find the employees with ids in range [100,..,500] whose entries were in the database on July 1, 2007." Let *n* be the total number of updates in a transaction-time database; note that *n* corresponds to the minimal information needed for storing the whole evolution. [7] presents a lower bound for answering a range-timeslice query. In particular, any method that uses linear space (i.e, $O(n/B)$ pages, where *B* is the number of object records that fit in a page) would need $O(\log_B n + s/B)$ I/O's to answer such a query (where an I/O transfers one page, and *s* corresponds to the size of the answer, i.e., the number of objects that satisfy the query).

*The Multiversion B-tree (MVBT)***:** The MVBT approach transforms a timeslice query to a partial persistence problem. In particular, a data structure is called *persistent* [4] if an update creates a new version of the data structure while the previous version is still retained and can be accessed. Otherwise, if old versions of the structure are discarded, the structure is termed *ephemeral*. *Partial* persistence implies that only the newest version of the structure can be modified to create a new version.

The key observation is that partial persistence "suits" nicely transaction-time evolution since these changes are always applied on the latest state $s(t)$ of the evolving set (Fig. 1). To support key range queries on a given $s(t)$, one could use an ordinary B^+-tree to index the objects in $s(t)$ (that is, the keys of the objects in $s(t)$ appear in the data pages of the B^+-tree). As $s(t)$ evolves over time through object changes, so does its corresponding B^+-tree. Storing copies of all the states that the B^+-tree took during the evolution of $s(t)$

is clearly inefficient. Instead, one should "see" the evolution of the B$^+$-tree as a partial persistence problem, i.e., as a set of updates that create subsequent versions of the B$^+$-tree.

Conceptually, the MVBT stores all the states assumed by the B$^+$-tree through its transaction-time evolution. Its structure is a directed acyclic graph of pages. This graph embeds many B$^+$-trees and has a number of root pages. Each root is responsible for providing access to a subsequent part of the B$^+$-tree's evolution. Data records in the MVBT leaf pages maintain the transaction-time evolution of the corresponding B$^+$-tree data records (that is, of the objects in $s(t)$). Each record is thus extended to include an interval [*insertion-time*, *deletion-time*), representing the transaction-times that the corresponding object was inserted/deleted from $s(t)$. During this interval the data-record is termed *alive*. Hence, the MVBT directly represents object deletions. Index records in the non-leaf pages of the MVBT maintain the evolution of the corresponding index records of the B$^+$-tree and are also augmented with insertion-time and deletion-time fields.

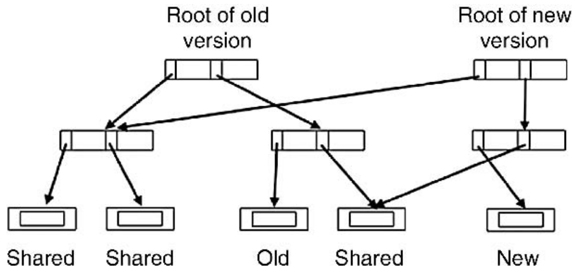
Assume that each page in the MVBT has a capacity of holding B records. A page is called *alive* if it has not been *time-split* (see below). With the exception of root pages, for all transaction-times t that a page is alive, it must have at least q records that are alive at t ($q < B$). This requirement enables clustering of the alive objects at a given time in a small number of pages, which in turn will minimize the query I/O. Conceptually, a data page forms a rectangle in the time-key space; for any time in this rectangle the page should contain at least q alive records. As a result, if the search algorithm accesses this page for any time during its rectangle, it is guaranteed to find at least q alive records. That is, when a page is accessed, it contributes enough records for the query answer.

The first step of an update (insertion or deletion) at the transaction time t locates the target leaf page in a way similar to the corresponding operations in an ordinary B$^+$-tree. Note that, only the latest state of the B$^+$-tree is traversed in this step. An update leads to a *structural change* if at least one new page is created. *Non-structural* are those updates which are handled within an existing page.

After locating the target leaf page, an insert operation at the current transaction time t adds a data record with a transaction interval of [t, UC) to the target leaf page. This may trigger a structural change in the MVBT, if the target leaf page already has B records. Similarly, a delete operation at transaction time t finds the target data record and changes the record's interval to [insertion-time, t). This may trigger a structural change if the resulting page ends up having less than q alive records at the current transaction time. The former structural change is called a *page overflow*, and the latter is a *weak version underflow* [1]. Page overflow and weak version underflow need special handling: a *time-split* is performed on the target leaf-page. The time-split on a page x at time t is performed by copying to a new page y the records alive in page x at t. Page x is considered *dead* after time t. Then the resulting new page has to be incorporated in the structure [1].

Since updates can propagate to ancestors, a root page may become full and time-split. This creates a new root page, which in turn may be split at a later transaction time to create another root and so on. By construction, each root of the MVBT is alive for a subsequent, non-intersecting transaction-time interval. Efficient access to the root that was alive at time t is possible by keeping an index on the roots, indexed by their time-split times. Since time-split times are in order, this root index is easily kept (this index is called the root* in [1]). In general, not many splits propagate to the top, so the number of root splits is small and the root* structure can be kept in main memory. If this is not the case, a small index can be created on top of the root* structure.

Answering a range-timeslice query on transaction time t has two parts. First, using the root index, the root alive at t is found. This part is conceptually equivalent to accessing $s(t)$ or, more explicitly, accessing the B$^+$-tree indexing the objects of $s(t)$. Second, the answer is found by searching this tree in a top-down fashion as in a B$^+$-tree. This search considers the record transaction interval. The transaction interval of every record returned or traversed should include the transaction time t, while its key attribute should satisfy the key query predicate. A range-timeslice query takes $O(\log_B n + s/B)$ I/O's while the space is linear to n (i.e., $O(n/B)$). Hence, the MVBT optimally solves the range-timeslice query. The update processing is $O(\log_B m)$ per change, where m is the size of $s(t)$ when the change took place. This is because a change that occurred at time t traverses what is logically a B$^+$-tree on the m elements of $s(t)$.



Transaction-Time Indexing. Figure 2. The Overlapping B-tree.

The Overlapping B-tree: Similar to the MVBT approach, the evolving set $s(t)$ is indexed by a B^+-tree. The intuition behind the Overlapping B-tree [2,10] is that the B^+-trees of subsequent versions (states) of $s(t)$ will not differ much. A similar approach was taken in the EXODUS DBMS [3]. The Overlapping B-tree is thus a graph structure that superimposes many ordinary B^+-trees (Fig. 2). An update at some time t creates a new version of $s(t)$ and a new root in the structure. If a subtree does not change between subsequent versions, it will be shared by the new root. Sharing common subtrees among subsequent B^+-trees is done through index nodes of the new B^+-tree that point to nodes of previous B^+-tree(s). An update will create a new copy of the data page it refers to. This implies that a new path is also created leading to the new page; this path is indexed under the new root.

To address a range-timeslice query for a given transaction time t, the latest root that was created before or at t must be found. Hence, roots are timestamped with the transaction time of their creation. These timestamps can be easily indexed on a separate B^+-tree. This is similar to the MVBT root* structure; however, the Overlapping B-tree creates a new root per version. After the appropriate root is found, the search continues traversing the tree under this root, as if an ordinary B^+-tree was present for state $s(t)$.

Updating the Overlapping B-tree involves traversing the structure from the current root version and locating the data page that needs to be updated. Then a copy of the page is created as well as a new path to this page. This implies $O(\log_B m)$ I/O's per update, where m is the current size of the evolving set. An advantage of the Overlapping structure is in the simplicity of its implementation. Note that except the roots, the other nodes do not involve any time-stamping. Such time-stamping is not needed because pages do not have to share records from various versions. Even if a single record changes in a page, the page cannot be shared by different versions; rather a new copy of the page is created. This, however, comes at the expense of the space performance. The Overlapping B-tree occupies $O(n \log_B n)$ pages since, in the worst case, every version creates an extra tree path. Further performance results on this access method can be found in [10].

Key Applications

The characteristics of transaction-time make such databases ideal for applications that need to maintain their past; examples are: billing, accounting, tax-related etc.

Cross-references

- ▶ Bi-Temporal Indexing
- ▶ B+-Tree
- ▶ Temporal Database
- ▶ Transaction Time
- ▶ Valid Time
- ▶ Valid-Time Indexing

Recommended Reading

1. Becker B., Gschwind S., Ohler T., Seeger B., and Widmayer P. An asymptotically optimal multiversion B-tree. VLDB J., 5(4):264–275, 1996.
2. Burton F.W., Huntbach M.M., and Kollias J.G. Multiple generation text files using overlapping tree structures. Comput. J., 28(4):414–416, 1985.
3. Carey M.J., DeWitt D.J., Richardson J.E., and Shekita E.J. Object and file management in the EXODUS extensible database system. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 91–100.
4. Driscoll J.R., Sarnak N., Sleator D.D., and Tarjan R.E. Making data structures persistent. J. Comput. Syst. Sci., 38(1):86–124, 1989.
5. Easton M.C. Key-sequence data sets on inedible storage. IBM J. Res. Dev., 30(3):230–241, 1986.
6. Lomet D. and Salzberg B. Access methods for multiversion data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989, pp. 315–324.
7. Salzberg B. and Tsotras V.J. A comparison of access methods for time-evolving data. ACM Comput. Surv., 31(2):158–221, 1999.
8. Snodgrass R.T. and Ahn I. Temporal databases. IEEE Comput., 19(9):35–42, 1986.
9. Tsotras V.J. and Kangelaris N. The snapshot index: an I/O-optimal access method for timeslice queries. Inf. Syst., 20(3):237–260, 1995.
10. Tzouramanis T., Manolopoulos Y., and Lorentzos N.A. Overlapping B+-trees: an implementation of a transaction time access method. Data Knowl. Eng., 29(3):381–404, 1999.
11. Varman P.J. and Verma R.M. An efficient multiversion access structure. IEEE Trans. Knowl. Data Eng., 9(3):391–409, 1997.

Transcriptional Networks

▶ Biological Networks

Transformation

▶ Mediation

Transformation Engines

▶ Interface Engines in Healthcare

Translation Lookaside Buffer (TLB)

▶ Processor Cache

Translingual Information Retrieval

▶ Cross-Language Mining and Retrieval

Tree Drawing

▶ Visualizing Hierarchical Data

Tree Pattern Queries

▶ XML Tree Pattern, XML Twig Query

Tree-based Indexing

Yannis Manolopoulos[1], Yannis Theodoridis[2], Vassilis J. Tsotras[3]
[1]Aristotle University of Thessaloniki, Thessaloniki, Greece
[2]University of Piraeus, Piraeus, Greece
[3]University of California-Riverside, Riverside, CA, USA

Synonyms

Index Sequential Access Method (ISAM); B+-tree; R-tree

Definition

Consider a relation R with some numeric attribute A taking values over an (ordered) domain D. A *range query* retrieves all tuples in R whose attribute A has values in the interval $[low, high]$. That is, $low \leq R.A \leq high$. To enable fast processing of range selection queries, an access method that maintains order is needed. Such an index has the form of a tree, where each node corresponds to a page. Leaf nodes contain (or index) the actual values of A, while index nodes provide ordered access to the nodes underneath. Examples of tree-based indexing are the B+-tree and the R-tree (for single- and multi-dimensional ranges, respectively).

Key Points

A major performance goal of a database management system is to minimize the number of I/O's (i.e., blocks or pages transferred) between the disk and main memory. One way to achieve this goal is to minimize the number of I/O's when answering a query. Consider for example relation *Employee* (*ssn, name, salary, dept, address*); the query: "Find the employees who reside in Santa Monica" references only a fraction of *Employee* records. It would be very inefficient to have the database system sequentially read all the pages of the *Employee* file and check the *address* field of each employee record for the value "Santa Monica." Instead the system should be able to locate the pages with "Santa Monica" employee records directly.

To allow such fast access additional data structures called access methods (or indices) are designed per database file. The term *search-key* is used to identify the attribute(s) on which the access method is built. There are two fundamental access methods, namely tree-based and hash-based indexing. They differ on the kind of queries that they can efficiently address. Tree-based indexing maintains the order of the search-key values. It is thus applicable to attributes that are *numeric* and hence it can be used to address *range* queries (and also *equality* queries, when the range query interval is reduced to one value). Hash-based indexing on the other hand does not assume any ordering; rather it is based on mapping the search-key values on a collection of buckets. Therefore it can only address *equality* (or *membership*) queries. Since a tree-based index maintains the order of the indexed values in its leaf pages, a (one-dimensional) range query is implemented as a search for the leaf page with the lower value of the range interval, followed by the accessing of sibling pages until a page that contains the higher value of the range interval is reached.

Tree-based indices are further categorized by whether their search-key ordering is the same with the file's logical order (if any). Note that a file may or may not be ordered according to the value of an (a sequence of) attribute(s). A file stored without any logical order is called an unordered file or heap. If the search-key of a tree-based index is the same as the ordering attribute of a (ordered) file then the index is called *primary*. Since such an index also clusters the values of the file, the term *clustering* index has also been used. The search-key of a primary index is usually the file's primary key, however this is not necessary. An index built on any non-ordering attribute of a file is called *secondary*. A relation can have several indices, on different search-keys; among them, at most one is primary (clustering) index and the rest are secondary ones (obviously, a file can have at most a single logical order since it is physically stored once).

Historically, the first tree-based index used in relational databases was the ISAM file (a static tree). Currently, the most widely used tree-based index is the B+-tree which is a dynamic structure (i.e., its size increases or decreases as the size of the indexed file changes by record insertions/deletions). As with any data structure, the performance of an access method is characterized by three costs, namely: query time, update time and space. Query time corresponds to the number of page accesses (I/O's) needed to answer a query. Update time counts the number of I/O's needed for updating the method when a record is updated, inserted or deleted. Space measures the number of pages occupied by the method's data structures. The B + -tree uses logarithmic update and query costs while its space requirements are linear to the size of the indexed file.

Cross-references

- ▶ Access Path
- ▶ Hash-based Indexing
- ▶ Index Creation and File Structures
- ▶ Primary Index
- ▶ Secondary Index
- ▶ Signature Trees
- ▶ Spatial Indexing Techniques
- ▶ Suffix Tree
- ▶ Text Indexing Techniques

Recommended Reading

1. Elmasri R.A., and Navathe S.B. Fundamentals of Database Systems (5th edn.). Addisson-Wesley, Reading, MA, 2007.
2. Manolopoulos, Theodoridis Y., Tsotras. Y., and Vassilis. J., Advanced Database Indexing. Kluwer, Dordecht, 1999.
3. Ramakrishnan R. and Gehrke J. Database Management Systems (3rd edn.). McGraw-Hill, NY, 2003.

Treemaps

Jean-Daniel Fekete
INRIA, LRI University Paris Sud, Orsay Cedex, France

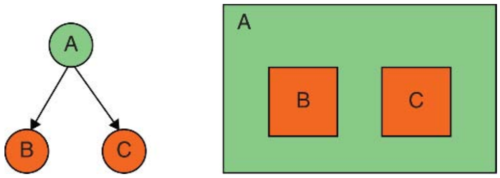
Definition

Treemaps [5] have been designed to visualize rooted trees using containment to express the hierarchy (Fig. 1 right). It is a space-filling technique and uses all the available space to render the tree. One important feature of Treemaps is their use of surface to represent an attribute value that adds up with the hierarchy. The number of leaves under a node is the simplest of these attributes. This number is one for a leaf and, for an interior node, it is the sum of the number of leaves of all its children.

This additive property is frequent on real trees. In an organizational chart, the salary of employees adds up, in an administrative geographical decomposition such as the one used by the census, population adds up, as well as income.

The traditional visual representation of rooted trees is called a *node-link diagram* (Fig. 1 left). Treemaps are not as good as standard node-link diagrams to represent the hierarchy depth on the tree; if an important task on the tree is to compare the depth of two nodes, node-link diagrams are more appropriate.

However, Treemaps can be applied on trees ranging from tens to hundred-thousands of nodes, much more



Treemaps. Figure 1. Rendition of a rooted tree as a node-link diagram on the left and as a Treemap on the right.

than standard node-link diagrams. Computing the layout of Treemaps is fast and their reading is easy with some training.

Historical Background

Treemaps were introduced by Shneiderman in 1992 [5]. They are restrictions of Euler diagrams to represent inclusion. Furthermore, when Euler diagrams are usually drawn with circles, ellipses, or curved shapes, Treemaps are drawn with rectangles to better use the screen real-estate.

The original Treemap algorithm is called "Slice and Dice" because at each level in the hierarchy, the children of a node cut the node's rectangle in slices horizontally or vertically, flipping direction at each level. Further refinements of Treemaps have tried to improve two main features: the aspect ratio of the rectangles and their order.

With the original "Slice and Dice" algorithm, the ratio between width and height of nodes can vary widely in the same Treemap. Some slices can become very thin while others can be almost square. Comparing visually the surface of rectangles with different aspect ratio is difficult for the human vision so one important improvement was to try to create nodes as square as possible. The most popular algorithm for achieving this property is due to van Wijk and is called "Squarified Treemap" [3].

Neither the "slice and Dice," nor the squarified algorithm maintains the order of children visually. When the leaf order is important, Bederson et al. have compared several algorithms and found that a simple variant of the squarified algorithm called "Ordered Treemaps" provided a good tradeoff between complexity and performance [2].

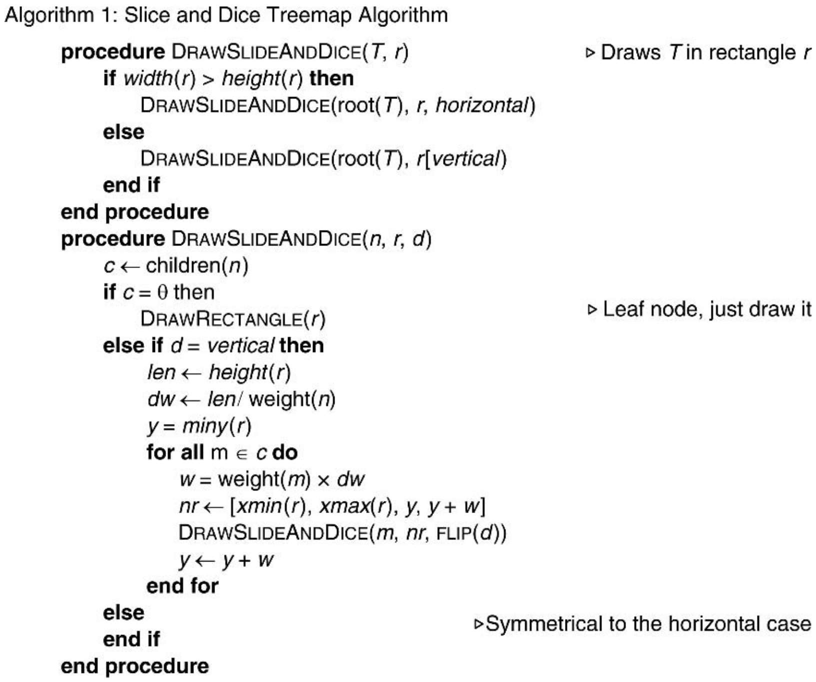
Further research have been conducted to study the limits of Treemaps in term of tree size and density [4] or to enhance their readability. Variant layouts have also been proposed, such as circular Treemaps (not space efficient) and Voronoï Treemaps [1] that are space efficient but costly to compute.

Foundations

As in all the visualization techniques, Treemaps visualizations consist in computing a *layout* for the tree – here a rectangle associated with each node – and drawing these rectangles using adequate visual attributes.

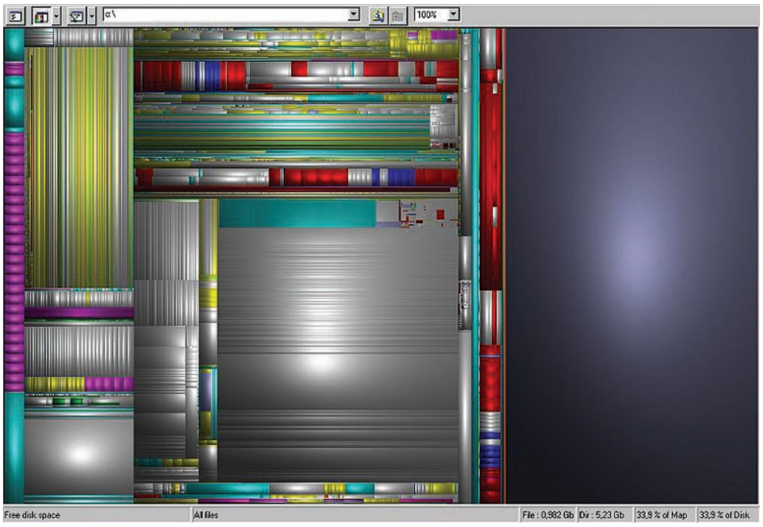
A rooted tree T is defined as a set of nodes $n \in T$ and three functions *root* , *parent* and *children* . It is convenient to define the element NIL to represent an undefined node. A rooted tree verifies the following axioms:

$$\text{parent}(T, n) \in \ T \cup \{\text{NIL}\}$$
$$\text{parent}(T, n) = \text{NIL} \Leftrightarrow n = \mathit{root}(T)$$
$$\text{children}(T, n) = \{c_1, c_2, \ldots c_k\} \in T^k$$
$$\text{parent}(T, n) = p \Leftrightarrow n \in \text{children}(T, p)$$

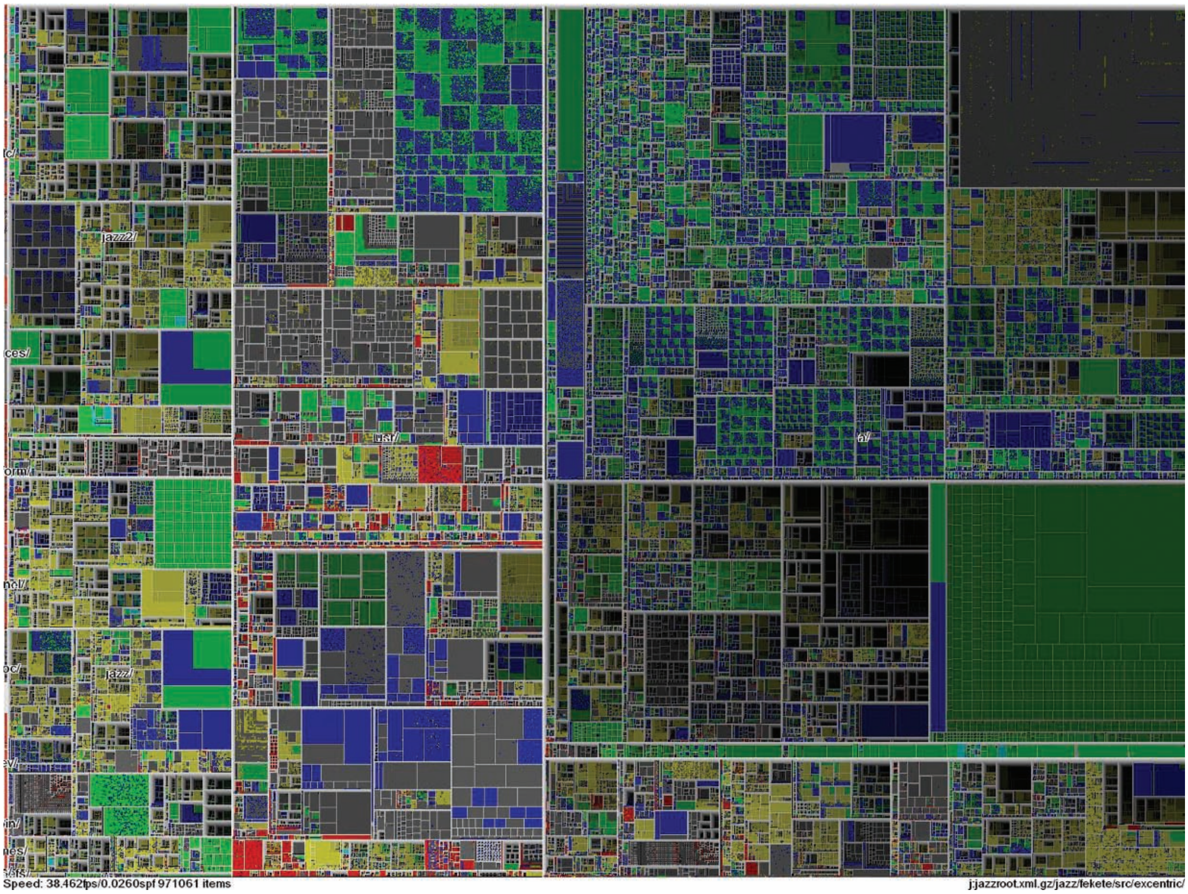


Algorithm 1: Slice and Dice Treemap Algorithm

```
procedure DrawSlideAndDice($T$, $r$)                         ▷ Draws $T$ in rectangle $r$
    if $width(r) > height(r)$ then
        DrawSlideAndDice(root($T$), $r$, $horizontal$)
    else
        DrawSlideAndDice(root($T$), $r[vertical]$)
    end if
end procedure
procedure DrawSlideAndDice($n$, $r$, $d$)
    $c \leftarrow$ children($n$)
    if $c = \theta$ then                                       ▷ Leaf node, just draw it
        DrawRectangle($r$)
    else if $d = vertical$ then
        $len \leftarrow height(r)$
        $dw \leftarrow len/$ weight($n$)
        $y = miny(r)$
        for all m $\in c$ do
            $w =$ weight($m$) $\times dw$
            $nr \leftarrow [xmin(r), xmax(r), y, y + w]$
            DrawSlideAndDice($m$, $nr$, flip($d$))
            $y \leftarrow y + w$
        end for
    else                                            ▷Symmetrical to the horizontal case
    end if
end procedure
```



Treemaps. Figure 2. SequoiaView using the Cushion Treemap rendition technique for a large file hierarchy.


jazz2/
jazz/
ces/
Speed: 38.462fps/0.0260spf 971061 items
j:jazzroot.xml.gz/jazz/fekete/src/excentric/

Treemaps. Figure 3. A Treemap of a one million file Web server, using intensity for depth perception and smooth shading to render the rectangles.

Algorithm 2: Squarified Treemap Algorithm

```
procedure DRAWSQUARIFIED($n$, $r$)
    if is Leaf ($n$) then                                   ▷ Leaf node, just draw it
        DRAWRECTANGLE($r$)
        return
    end if
    $scale \leftarrow width(r) * height(r)$ / weight($n$)
    $sorted$ = SORTBYDECREASINGWEIGHT(children($n$))
    while $sorted \neq nil$ do
        if $width(d) > height(`)$ then                       ▷ Create vertical strips
            $y = miny(r)$
            $len = height(r)$
            ($strip$, $sorted$) = SQUARIFY($sorted$, $len$, $scale$)
            $width \leftarrow \sum_{c \in strip}$ weight($c$) $\times$ $scale/ len$
            for all $child \in strip$ do
                $h$ = weight(child) $\times$ scale / width
                $nr \leftarrow [xmin(r), xmax(r), y, y + h]$
                DRAWSQUARIFIED($child$, $nr$)
                $y \leftarrow y + h$
            end for
            $r \leftarrow [xmin(r) + width, xmax(r), ymin(r), ymax(r)]$
        else
                                                    ▷ Symmetrical for the horizontal case
        endif
    endwhile
end procedure
function SQUARIFY($children$, $w$, $scale$)                  ▷ $w$ is the strip width
  $strip \leftarrow [pop(children)]$                          ▷ Strip starts with one node
  while children $\neq \emptyset$ do
    $c \leftarrow head(children)$
    if WORST($strip$, $w$, $scale$) $\leq$ WORST($strip + c$, $w$, $scale$) then
        $strip \leftarrow strip + c$
        $pop(children)$
        $c \leftarrow head(children)$
    else
        break
    end if
  end while
  return ($strip$, $children$)
end function
```

For convenience, define the degree of a node deg(T,n) to be the number of nodes in its children set and the *is Leaf*(T, m) predicate is deg(T, n) = 0.

Drawing a Treemap requires a positive function weight on the tree such that:

$$\begin{aligned} \text{weight}(T, n) = \sum_{c \in \text{children}(T,n)} \text{weight}(T, c) \wedge \\ \text{weight}(T, n) > 0 \end{aligned} \tag{1}$$

A rectangular portion of the screen is used to display the whole tree. The layout algorithm is recursive and uses the *weight* function to allocate space. The algorithm for "Slice and Dice" is simple (Algorithm 1). The "flip" function and the accessor function for the rectangle type should be self explanatory.

The squarified algorithm is more complex. It tries to assign a rectangle to each node with the ratio w/h close to 1. Computing the optimal configuration is equivalent to bin-packing and is known to be NP-complete so it uses a simple heuristics. It sorts the children by decreasing weights and fills the parent rectangle with strips of children. Each node rectangle is added to the strip until adding the next rectangle decreases the quality of the rectangles already in the strip. Then a new strip is created (see Algorithm 2). The quality is usually the worst aspect ratio of the rectangles in the strip. It is computed as follows: given R a list of areas and s their total sum:

$$worst(R, w) = \max_{r \in R}(\max(w^2 r/s^2, s^2/(w^2 r))$$

In the algorithm 2, a scale argument is passed to the function *worst* to transform the weights in surfaces since in the definition (equation 1), the weights are in arbitrary units. The computation of strips can be done incrementally as described in [3].

Sorted Treemaps only change two lines to the Algorithm 2: line 7 is removed so the nodes are in their natural order and line 9 is removed so the strips are always horizontal.

Key Applications

Treemaps have been used on any kinds of trees. They became visible to a large audience when the company SmartMoney used Treemaps to visualize a "map of the market" (still visible at www.smartmoney.com/mapofthemarket. Other applications include browsers for file systems where one can visualize the whole hierarchy and navigate on it by file sizes and using sophisticated color rendition for the rectangles, such as "cushion" (Fig. 2) or "smooth shading" (Fig. 3).

They are also very useful to visualize data tables using a flexible hierarchy. A set of column is chosen and ordered. The first level of the tree consists in creating a node for each values of the first column (partitioning according to the first column). The second level is built by sub-partitioning by values of the second column etc. This method is well suited to OLAP databases where the partitioning is already available. By interactively changing the column order, different Treemap views of the table can be explored.

Cross-references

- ▶ Data Visualization
- ▶ Dense Pixel Displays
- ▶ Graphic Information Processing
- ▶ Graphical Perception
- ▶ Hierarchical Data Model
- ▶ Hierarchy
- ▶ Human-Computer Interaction
- ▶ Visual Analytics
- ▶ Visual Data Mining
- ▶ Visual Representation
- ▶ Visualization for Information Retrieval
- ▶ Visualizing Hierarchical Data

Recommended Reading

1. Balzer M. and Deussen O. Voronoi treemaps. In Proc. IEEE Symp. on Information Visualization. 2005, pp. 48–56.
2. Bederson B.B., Shneiderman B., and Wattenberg M. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. ACM Trans. Graph., 21(4):833–854, 2002.
3. Bruls M., Huizing K., and van Wijk J.J. Squarified treemaps. In Data Visualization 2000, Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization, W. de Leeuw and R. van Liere (eds.). Springer, Vienna, 2000, pp. 33–42.
4. Fekete J.D. and Plaisant C. Interactive information visualization of a million items. In Proc. IEEE Symp. on Information Visualization, 2002, pp. 117–124.
5. Shneiderman B. Tree visualization with tree-maps: 2-d space-filling approach. ACM Trans. Graph., 11(1):92–99, 1992.

Tree-Structured Classifier

▶ Scalable Decision Tree Construction

Triangular Norms

Vilém Novák
University of Ostrava, Ostrava, Czech Republic

Synonyms

t-Norm

Definition

Triangular norms (briefly t-norms) are special binary operations $\mathbf{T} : [0,1]^2 \to [0,1]$. They are interesting for fuzzy logic because they preserve the fundamental properties of the logical conjunction "and" (to hold at the same time), namely commutativity, monotonicity, associativity, and boundedness and thus, they serve as a natural generalization of the classical conjunction in many-valued logical systems.

A concept associated with the t-norm is the triangular conorm (t-conorm) $\mathbf{S} : [0,1]^2 \to [0,1]$. This corresponds to the behaviour of truth values when joined by the logical connective "or."

Key Points

A t-norm is a binary operation $\mathbf{T} : [0,1]^2 \to [0,1]$ such that the following axioms are satisfied for all $a, b, c \in [0,1]$:

$$\begin{array}{ll}\text{(commutativity)} & a\ \mathbf{T}\ b = b\ \mathbf{T}\ a, \\ \text{(associativity)} & a\ \mathbf{T}(b\ \mathbf{T}\ c) = (a\ \mathbf{T}\ b)\mathbf{T}\ c, \\ \text{(monotonicity)} & a \leq b \quad \text{implies} \quad a\ \mathbf{T}\ c \leq b\ \mathbf{T}\ c, \\ \text{(boundary condition)} & 1\ \mathbf{T}\ a = a.\end{array}$$

The most important t-norms are *minimum*, *product* and *Łukasiewicz conjunction* defined by

$$a\ \mathbf{T}_{\mathrm{L}} b = \max\{0, a + b - 1\}.$$

A t-conorm is a binary operation $\mathbf{S} : [0,1]^2 \to [0,1]$, which is commutative, associative, monotone, and for all $a \in [0,1]$ it fulfils the following boundary condition:

$$0\ \mathbf{S}\ a = a.$$

The most important t-conorms are *maximum*, *probabilistic sum*

$$a\ \mathbf{S}_P\ b = a + b - ab$$

and *Łukasiewicz disjunction*

$$a\ \mathbf{S}_{\mathrm{L}} b = \min\{1, a + b\}.$$

A t-conorm is dual to the given t-norm **T** if $a\ \mathbf{S}\ b = 1 - (1-a)\ \mathbf{T}\ (1-b)$ holds for all $a, b \in [0,1]$. There are many classes of t-norms and their structure is extremely complicated.

Cross-references

- ▶ Approximate Reasoning
- ▶ Fuzzy IF-THEN Rules
- ▶ Fuzzy Relation
- ▶ Fuzzy Set
- ▶ Residuated Lattice

Recommended Reading

1. Klement E.P., Mesiar R., and Pap E. Triangular Norms. Kluwer, Dordrecht, 2000.

Triangulated Irregular Network

Leila De Floriani, Paola Magillo
University of Genova, Genova, Italy

Synonyms

Triangulated terrains; TIN

Definition

A Triangulated Irregular Network (TIN) is a special case of a Digital Elevation Model (DEM).

A terrain can be mathematically modeled as a function $z = f(x, y)$ mapping a point (x, y) in a domain D in the plane to its elevation value $f(x, y)$. In practice, the value of function f is known at a finite set S of points within D. A DEM provides an estimated value for function f at any point (x, y) of the domain, based on the values at the points of S. A DEM consists of a subdivision of the domain into cells and of a piece-wise interpolating function defined on such cells.

A TIN is a DEM in which the domain subdivision is a triangle mesh, i.e., a set T of triangles such that: (i) the set of vertices of T is S, (ii) the interiors of any two triangles of T do not intersect, (iii) if the boundaries of two triangles intersect, then the intersection is either a common vertex, or a common edge.

Usually, a linear interpolating function is defined on the triangles of T, thus providing a continuous terrain approximation. Given a triangle $t = P_1P_2P_3$ and a point $P = (x,y)$ inside triangle t, the following function estimates the elevation value z at P. Let $P_1 = (x_1,y_1,z_1)$ $P_2 = (x_2,y_2,z_2)$ and $P_3 = (x_3,y_3,z_3)$ be the coordinates of the three vertices of t. Then,

$$z = z_1 - (a(x - x_1) + b(y - y_1))/c,$$

where (a,b,c) are the components of the normal vector to the triangle $P_1P_2P_3$ in 3D space:

$$a = (y_1 - y_2)(z_1 - z_3) - (z_1 - z_2)(y_1 - y_3)$$
$$b = (z_1 - z_2)(x_1 - x_3) - (x_1 - x_2)(z_1 - z_3)$$
$$c = (x_1 - x_2)(y_1 - y_3) - (y_1 - y_2)(x_1 - x_3)$$

Key Points

TINs have been extensively studied in Geographic Information Systems (GISs), in Computational Geometry, and in Computer Graphics. Several data structures and algorithms for representing, constructing and manipulating triangle meshes have been proposed.

The quality of the terrain approximation provided by a TIN depends on the underlying triangle mesh. Note that a point set S does not define a unique triangle mesh. The most widely used triangle mesh is the *Delaunay* one, in which the circumcircle of each triangle does not contain any data point in its interior. This means that the triangles of a Delaunay mesh are as much equiangular as possible. It has also been proven that the use of a Delaunay mesh as the basis of a TIN improves the quality of the terrain approximation and enhances numerical stability in computations. Other triangulation criteria have been proposed which consider the triangles of the mesh in 3D space.

Often, not only points, but also lines need to be included in a TIN. Such lines may represent morphological terrain features (coast lines, rivers, ridges), man-made structures (roads, railways, gas lines), political or administrative boundaries, or contour lines. The Delaunay criterion has been modified to deal with lines in two different ways: (i) in the *constrained Delaunay triangulation*, the lines appear as triangle edges (but it may present sliver triangles); (ii) in the *conforming Delaunay triangulation*, each line is discretized by adding points on it (but a large number of points may need to be added).

TINs are used in multiresolution terrain modeling.

Cross-references

▶ Regular Entry on Digital Elevation Models (DEM)
▶ Regular Entry on Multiresolution Terrain Modeling

Recommended Reading

1. de Berg M., van Kreveld M., Overmars M., and Schwarzkopf O. Computational Geometry – Algorithms and Applications. 2nd ed. Springer-Verlag, Berlin, 2000.
2. De Floriani L., Magillo P., and Puppo E. Applications of computational geometry to Geographic Information Systems. Chapter 7 in Handbook of Computational Geometry, J.R. Sack, J. Urrutia (eds.). Elsevier Science, 1999, pp. 333–388.
3. van Kreveld M. Digital elevation models and TIN algorithms. In Algorithmic Foundations of Geographic Information Systems, M. van Kreveld, J. Nievergelt, T. Roos, P. Widmayer (eds.). Springer-Verlag, Berlin, 1997, pp. 37–78.

Triangulated Terrains

▶ Triangulated Irregular Networks (TIN)

Trie

Maxime Crochemore[1,2], Thierry Lecroq[3]
[1]King's College London, London, UK
[2]University of Paris-East, Paris, France
[3]University of Rouen, Rouen, France

Synonyms

Prefix tree

Definition

A trie is a rooted tree used for storing associative arrays where keys are usually strings. Edges are often labeled by individual symbols. Then common prefixes are factorized. Each node of the trie is associated with a prefix of a string of the set of strings: concatenation of the labels of the path from the root to the node. The root is associated with the empty string. Strings of the set are stored in terminal nodes (leaves) but not in internal nodes. A trie can be seen as a Deterministic Finite Automaton.

Tries can be compacted. To get a compact trie from a trie, internal nodes with exactly one successor are removed. Then labels of edges between remaining nodes are concatenated. Thus:

- Edges are labeled by strings.
- Internal nodes have at least two children.
- Edges outgoing an internal node are labeled by strings starting with different symbols.

Historical Background

Tries were first recommended by de la Briandais [1]. The word "trie" comes from information re*trie*val and was suggested by Fredkin [3]. Tries enable to store and retrieve information that consists of key-element pairs. Fredkin suggested, in 1960, that it is an alternative way of storage to unordered lists, ordered lists or pigeonholes. The reader can refer to [6] for further details on tries.

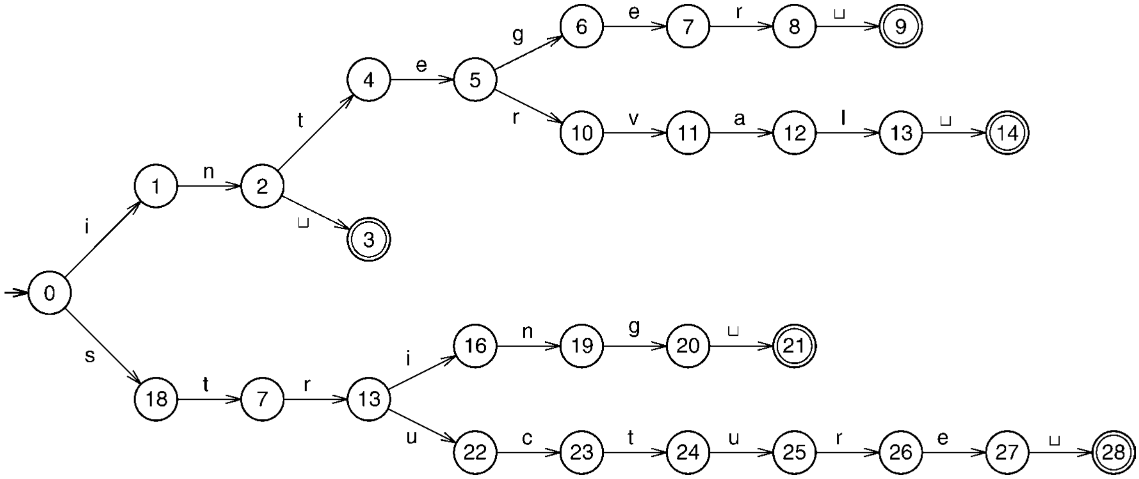
Foundations

In binary search trees, keys are stored in all the nodes of the tree and the search method is based on comparison between keys. In tries, keys are stored in the leaves and the search method involves left-to-right comparison of prefixes of the keys.

The trie of the set of strings $X = \{$`in`, `integer`, `interval`, `string`, `structure`$\}$ is presented Fig. 1. Note that the space sign ␣ has been added at the end of each of the strings of X so that no string of X is a prefix of another string of X. Then each string of X is associated with a leaf of the trie (not with an internal node). The trie for the set X can be implemented in linear space with respect to the total length of the strings in X. The compact trie for the set X can be implemented in linear space with respect to the total number of the strings in X, which dramatically reduces the size of the structure.

Consider that the strings are build over an alphabet of size σ.

Flajolet and Sedgewick [8] provide an average case analysis of tries.



Trie. Figure 1. Trie of X = {`in`, `integer`, `interval`, `string`, `structure`}.

Construction

The algorithm Trie(X) shown in Fig. 2, builds the trie containing all the strings in the set X. It uses a function target(p, a) that gives the successor of a node p for a symbol a or the value special nil if such a node does not exist. It works by inserting all the strings of X successively in the trie starting with the tree consisting with a single node (the root). Then for each string $x \in X$ it spells the longest prefix of x corresponding to an existing path from the root of the trie. When this longest prefix is found, it creates the nodes and the edges of the remaining suffix of x.

The sum of the lengths of all the strings of X is denoted by M. Then, the algorithm Trie(X) run in time $O(M)$ when the branching time from a node with a given symbol is constant or $O(M \times \log \sigma)$ when the branching time depends on the alphabet size (see Implementation below).

Searching

The algorithm IsInTrie(*root*, x), see Fig. 3, tests if the string x is present in the trie and consequently if the string x is a prefix of strings represented by the trie. It works, similarly to the creation of the trie, by spelling, from the root of the trie, the longest prefix of x corresponding to a branch in the trie. If this longest prefix is x itself, then the algorithm returns true and the string x belongs to the trie, otherwise the algorithm returns false and the string is not a prefix of any string in the set. The algorithm IsInTrie(*root*, x) works in time $O(|x|)$ or $O(|x| \times log\ \sigma)$ depending on the branching time.

```
Trie($X$)
  $root \leftarrow$ new node
  for each string $x \in X$ do
    $p \leftarrow root$
    for $i \leftarrow 0$ to $|x| - 1$ do
      $q \leftarrow$ Target $(p, x[i])$
      if $q =$ NIL then
        $q \leftarrow$ new node
        $Succ[p] \leftarrow Succ[p] \cup \{(X[i], q)\}$
      $p \leftarrow q$
    $info[p] \leftarrow x$
  return $root$
```

Trie. Figure 2. Algorithm that builds the trie containing of the string of a set X.

```
IsInTrie($p, x$)
  if $x$ is empty then
    return TRUE
  else $q \leftarrow$ Target$(p, x[0])$
    if $q =$ NIL then
      return FALSE
  else return IsInTrie$(q, x[1..|x| - 1])$
```

Trie. Figure 3. Algorithm that enables to test if a string x belongs to a trie.

Sorting

A trie can be used to sort a set of strings by doing the following: insert all the strings in the trie and output them in lexicographically increasing order by applying a pre-order traversal of the trie (respectively lexicographically decreasing order by applying a post-order traversal of the trie).

Implementation

There exist different possible representations of a trie, each with different branching times. It is possible to use a transition table whose size is the product of the number of nodes times the size of the alphabet. A trie can then be implemented in linear space with respect to the total length of all the strings it represents. The branching time, for finding the successor of a given node with a given symbol, is then constant.

The use of adjacency lists has the advantage to minimize the required space but the branching time depends on the alphabet size. It can be as low as $\log(\sigma)$ if the alphabet is ordered and outgoing edges are stored in a balanced search tree. A representation by binary tree is achieved by using a "first child – right sibling" representation of the trie.

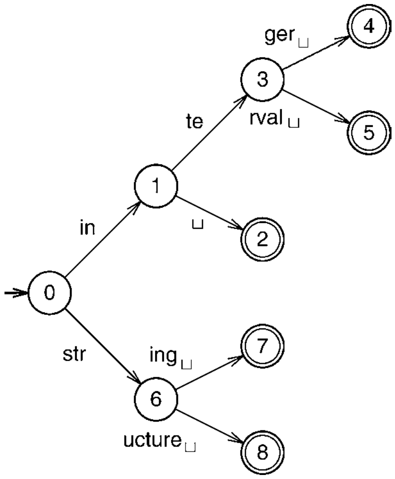
Hashing techniques can be used as a good trade-off between transition tables and adjacency lists. The branching time is then constant on average.

A mixed technique known as the "deep shallow" technique consists of representing the nodes up to a certain level k with a transition table and to use adjacency lists for the nodes with level greater than k. Most of the times nodes with small level have many successors while nodes with large level have only one successor.

However when storage space is an issue and tries do not entirely fit into the central memory, it is possible to use compact tries as described below. Accesses to individual edges are only a bit more involved than in non-compact tries.

Compact Tries

The compact trie of the set of strings X = {in, integer, interval, string, structure} is presented Fig. 4. It is obtained from the trie of Fig. 1 by removing internal nodes with exactly one successor. Then labels of edges between remaining nodes are concatenated.



Trie. Figure 4. Compact trie of X = {in⊔, integer⊔, interval⊔, string⊔, structure⊔}.

Patricia Trees

Morrison [7] designed specific compact tries known as PATRICIA trees. PATRICIA trees are efficient especially for extremely long variable-length strings. The PATRICIA tree of the set of strings X = {in⊔, integer⊔, interval⊔, string⊔, structure⊔} is presented Fig. 5. PATRICIA trees are binary trees that consider the binary encoding of the strings. In Fig. 5 nodes 0, 1 and 3 actually point to in⊔, nodes 4 and 7 point to integer⊔, node 8 points to interval⊔, nodes 2 and 5 point to string⊔ and node 6 points to structure⊔. Small numbers close to the nodes are skip numbers, they indicate on which bit the branching has to be decided. The skip number of node 0 is 1: it corresponds to bit at position 1 (bolded in columns 1, 18, 26, and 34 on Fig. 5), which is the leftmost difference in the bit strings representing the strings of X. The three strings in⊔, integer⊔ and interval⊔ possess a 0 so they are on the left and string⊔ and structure⊔ possess a 1 so they are on the right. The skip number of node 1 is 18: it corresponds to bit at position 18 (bolded on Fig. 5), where is the leftmost difference in the bit strings representing the three strings in⊔, integer⊔ and interval⊔. The string in⊔ has a 0 so it is on the left and integer⊔ and interval⊔ possess a 1 so they are on the right. The skip number of node 2 is 26: it corresponds to bit at position 26, (bolded on Fig. 5), where is the leftmost difference in the bit strings representing the two strings string⊔ and structure⊔. The string string⊔ has a 0 so it is on the left and structure⊔ possesses a 1 so it is on the right. The skip number of node 4 is 34: it corresponds to bit at position 34 (bolded on Fig. 5), where is the leftmost difference in the bit strings representing the two strings integer⊔ and interval⊔. The string interval⊔ has a 0 so it is on the left and integer⊔ possesses a 1 so it is on the right.

The skip number of a node (different from the root) in a PATRICIA tree is never larger than its parent skip number.

Decimal and binary ASCII codes of the symbols

		7	6	5	4	3	2	1	0
␣	32	0	0	1	0	0	0	0	0
a	97	0	1	1	0	0	0	0	1
c	99	0	1	1	0	0	0	1	1
e	101	0	1	1	0	0	1	0	1
g	103	0	1	1	0	0	1	1	1

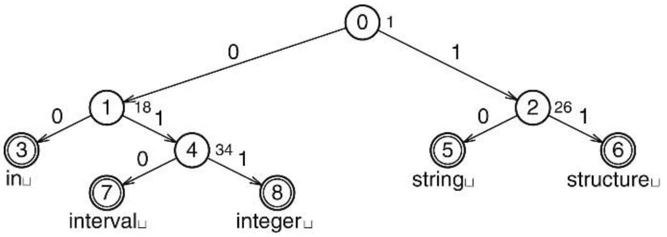
		7	6	5	4	3	2	1	0
i	105	0	1	1	0	1	0	0	1
l	108	0	1	1	0	1	1	0	0
n	110	0	1	1	0	1	1	1	0
r	114	0	1	1	1	0	0	1	0
s	115	0	1	1	1	0	0	1	1

		7	6	5	4	3	2	1	0
t	116	0	1	1	1	0	1	0	0
u	117	0	1	1	1	0	1	0	1
v	118	0	1	1	1	0	1	1	0

Strings of *X* as sequences of bits

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
in␣	1	0	0	1	0	1	1	0	0	1	1	1	0	1	1	0	0	0	0	0	0	1	0	0												
integer␣	1	0	0	1	0	1	1	0	0	1	1	1	0	1	1	0	0	0	1	0	1	1	1	0	1	0	1	0	0	1	1	0	0	1	1	0 ···
interval␣	1	0	0	1	0	1	1	0	0	1	1	1	0	1	1	0	0	0	1	0	1	1	1	0	1	0	1	0	0	1	1	0	0	1	0	0 ···
string␣	1	1	0	0	1	1	1	0	0	0	1	0	1	1	1	0	0	1	0	0	1	1	1	0	1	0	0	1	0	1	1	0	0	1	1	1 ···
structure␣	1	1	0	0	1	1	1	0	0	0	1	0	1	1	1	0	0	1	0	0	1	1	1	0	1	0	1	0	1	1	1	0	1	1	0	0 ···

PATRICIA tree of *X*



Trie. Figure 5. PATRICIA tree of *X* = {in␣, integer␣, interval␣, string␣, structure␣}.

The reader can refer to [4] for further details on PATRICA trees.

Key Applications

Tries are used for dictionary representation including spell checking systems or predictive text for mobile phones (T9 system for instance). They are used for text compression (Ziv and Lempel compression schemes), multiple string matching (or multi key search). They are at the basis of the Burstsort for sorting large sets of strings. Also, it should be mentioned that when *X* is the set of suffixes of a string, the structures that are then called Suffix Trie or Suffix Tree are intensively used for Pattern Matching [2,5].

Cross-references

▶ Data Dictionary
▶ Suffix Tree
▶ Text Compression

Recommended Reading

1. de la Briandais R., File searching using variable length keys. In Proc. Western Joint Computer Conference., 1959, pp. 295–298.
2. Crochemore M., Hancart C., and Lecroq T. Algorithms on Strings. Cambridge University Press, Cambridge, 2007.
3. Fredkin E. Trie memory. Commun. ACM, 3(9):490–499, 1960.
4. Gonnet G.H. and Baeza-Yates R. Handbook of Algorithms and Data Structures – In Pascal and C, 2nd edn. Addison-Wesley, 1991.
5. Gusfield D. Algorithms on strings, trees and sequences. Cambridge University Press, Cambridge, 1997.
6. Knuth D.E. The Art of Computer Programming, Volume 3: Sorting and Searching, 3rd edn. Addison-Wesley, 1997, Section 6.3: Digital Searching, pp. 492–512.
7. Morrison D.R. PATRICIA – Practical Algorithm to Retrieve Information Coded in Alphanumeric, J. ACM, 15(4):514–534, 1968.
8. Sedgewick R. and Flajolet Ph. An Introduction to the Analysis of Algorithms, Addison-Wesley, 1996.

Triggers

▶ Database Trigger
▶ ECA Rules

True Answer (Maybe Answer)

▶ Certain (and Possible) Answers

Trust and Reputation in Peer-to-Peer Systems

Zoran Despotovic
NTT DoCoMo Communications Laboratories Europe,
Munich, Germany

Synonyms

Feedback systems; Word of mouth

Definition

Trust means reliance on something or someone's action. As such, it involves risks on the side of the subject of trust, i.e., trustor. Reducing these risks is the main goal of a trust management system. A possible way to do this is through reputation management, i.e., reputation systems.

In a typical large scale online setting, be it on the Web or in P2P networks, it is necessary to learn more about prospective transaction partners prior to engaging in a transaction with them. The size of such systems makes it highly improbable to meet the same partner repeatedly, so own experience is of little use. The types of the performed transactions are often such that well-established forms of quality assurance (e.g., contracts) are highly inefficient. Under such circumstances, reputation systems ("word of mouth" [3]) turn out to be the only viable mechanism to encourage trustworthy behavior and guide people to decide whom to trust and to what degree. They do this through collecting, distributing, and aggregating feedback about the participants past behavior," as Resnick et al. [8] explain. The key presumptions of a reputation system are that the participants of the considered online community engage in repeated interactions and that the information about their past doings is informative of their future performance and as such will influence it. Thus, collecting, processing, and disseminating the feedback about the participants' past behavior is expected to boost their trustworthiness.

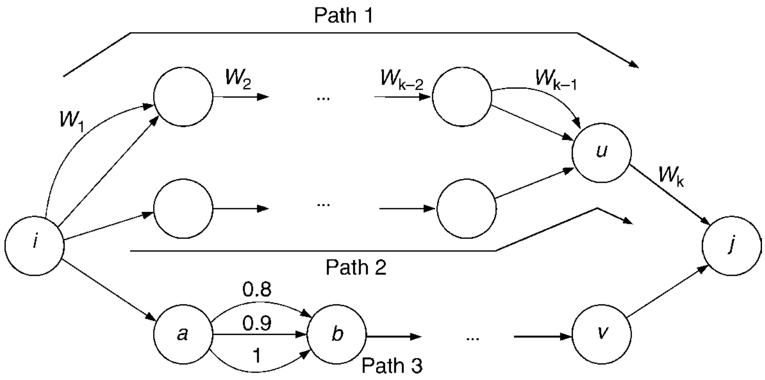
Historical Background

The concept of reputation is almost as old as human society. It was present in ancient times as a key enabler of trade and a broad range of other activities [3]. In the millennia to come, a new form of interaction between interested parties emerged, involving contractual agreements. When necessary, they are enforced by third parties, be it a local feudal sovereign in the medieval time or state, as nowadays. But the need for reputation did not disappear. Quite often, it is not possible to foresee what can go wrong, so that it can be specified in a contract. More important, binding contracts incur transaction costs which sometimes offset the prospective benefits from the interaction. In such cases, people resort to streamlining their interactions in informal ways, one of them being the use of reputation.

Reputation has long been a subject of study in economics. Economists find it vital. Many markets would not exist or they would be highly inefficient without reputation. Consider a market in which sellers sell goods of different qualities. Buyers cannot observe the quality of any good. Thus, they tend to undervalue high quality goods, as they may end up purchasing low quality. But then high quality sellers cannot achieve good prices and may withdraw from the market. So only low quality goods will be traded. This is what George Akerlof calls the "market for lemons" [1]. Information asymmetry between sellers and buyers is critical here. A mechanism to break it is needed. Reputation systems may be such a mechanism.

Today's perception of the P2P systems are old about eight years as of this writing. But that is quite enough for a turbulent history, which demonstrated the need for reputation management, among others. There were numerous reports of viruses spreading through well known P2P file sharing applications such as Gnutella or Kazaa. A quick inspection in Google reveals millions of entries returned to the query `(Gnutella OR Kazaa) AND virus`. As an example, Wired reports as of September 2004 that "forty-five percent of the executable files downloaded through Kazaa, the most popular file-sharing program, contain malicious code like viruses and Trojan horses, according to a new study."

Although these examples illustrate the point, they are benign in the sense that one can select different ways to download content, e.g., through well known web sites. But the P2P paradigm does not coincide with simple file swapping. It aims at making a more serious impact on the online world through offering a range of useful applications. No matter what these applications are, they will need reputation management. The reason is that P2P applications must be implemented through cooperation of their users, which can be unskilled or dishonest.



Trust and Reputation in Peer-to-Peer Systems. Figure 1. A trust graph.

Foundations

The core of any P2P reputation system is how it solves the following problem: how can a peer use the information on experiences between the peers to evaluate the trustworthiness of any other peer? A possible strategy might be as follows. The peer (call it also trustor) can ask its friends to report on their experiences with the unknown peer (trustee). However, the friends might not have any experience with the peer in question. This happens quite frequently in large scale systems, in which virtually every interaction is with a new peer, not seen before. As a result, the peer may search for some other unknown peers which happened to interact with the trustee. Their opinion might help. However, they might lie for whatever reason. So the problem is now how to assess the credibility of their reports. A possible solution is to continue with the search, this time looking for peers who interacted with the feedback providers and so on until enough peers are found with whom the trustor had enough experiences so that it knows their credibility. The whole process is depicted in Figure 1. The figure shows a set of peers, which are shown as vertices in the graph. The arcs represent the interactions among them, i.e., services they provide to each other. For example, peer *b* provided three services to peer *a*. The weights next to the arcs represent the level of satisfaction of the service consumer with the provided service. The structure that is formed in this way is called a trust graph.

There are two classes of reputation systems: *signaling and sanctioning reputation systems* [3]. They make different assumptions on the underlying behavior and also use different amounts of available reputation data, i.e., different fractions of the trust graph. In a signaling reputation system, the interacting entities are presented with signals of what can go wrong in the interactions if they behave in specific ways. Having appropriate signals, the entities should decide what behavior is most appropriate for them. An important assumption of the signaling reputation systems is that the involved entities do not change their behavior in response to a change of their reputation. As an example, the system may just provide a prospective buyer with indications of the probability that the seller will fail to deliver a purchased item. This probability is the main property of the seller. It can change with time, but independently of the seller's reputation.

The other possibility is *sanctioning reputation systems.* The main assumption they make is that the involved entities are aware of the effect the reputation has on their benefits and thus adjust their behavior dynamically as their reputation changes. The main task of a reputation system in this case is to *sanction* misbehavior through providing correlation between the feedback the agent receives and the long-run profit made. The distinction between signaling and sanctioning reputation systems is made explicit in the following discussion.

A typical signaling approach involves the following three-step procedure: (i) enumerating all paths from the trustor to the trustee, (ii) aggregating the feedback along the paths and (iii) merging the obtained values. There is a nice theory developed on this subject. It is due to Richardson et al. [9]. Consider a trust graph and assume that there is only one directed arc from *i* to *j*, for any pair of vertices *i* and *j*. Multiple arcs have been merged somehow. It is not important how exactly this merging is done. Consider the matrix $M \equiv [M_{ij}]_{i,j=1}^{N}$ (N is the number of the peers) corresponding to the trust graph and assume that it has been normalized so that for any $1 \leq i, j \leq N$: $0 \leq M_{ij} \leq 1$ and $\sum_{k=1}^{N} M_{ik} = 1$.

Define two binary operators: trust concatenation, the symbol $\circ$ will be used to denote it, and trust aggregation, denoted by $\diamond$. The former is applied on two consecutive edges in a chain of trust, while the latter applies to two chains. Simple multiplication and addition are good examples of these operators. Define now a matrix "multiplication" operation $\bullet$ as $C = A \bullet B$ such that $C_{ij} = \diamond(\forall k : A_{ik} \circ B_{kj})$. If $A = B \equiv M$, where M is the matrix representation of a given trust graph, then the interpretation of C_{ij} is aggregated trust that i puts on j over all chains of length 2. Again, if $\circ$ and $\diamond$ are ordinary multiplication and addition then $\bullet$ becomes the ordinary matrix multiplication. This is what [6] proposes.

Now, the most interesting result is that if $\diamond$ is commutative and associative and $\circ$ is associative and distributes over $\diamond$ then the aggregated value of all paths (of any length) between any pair of users can be obtained by the following simple algorithm:

$$Q^{(0)} = M, Q^{(k)} = M \bullet Q^{(k-1)} \text{ until } Q^{(k)} = Q^{(k-1)}. \tag{1}$$

The computation will converge after a finite number of steps if the matrix M (or the trust graph) is irreducible and aperiodic. It is important to see that the computation can be performed locally, after each iteration k all the peers can retrieve from their neighbors the currently computed opinions of those neighbors about all other peers and then do the computation of the step $k + 1$. It turns out that this algorithm requires at most $O(N^3)$ computations per peer.

In a similar vein, Xiong and Liu [10] compute the trustworthiness of a given peer as the average feedback about it weighted by the trustworthiness of the feedback originators themselves. This can be expressed by the formula:

$$t_j = \sum_{e \in incoming(j)} w_e \cdot \frac{t_{source(e)}}{\sum_{f \in incoming(j)} t_{source(f)}},$$

where $incoming(j)$ is the set of all edges ending at node j, w_e is the feedback belonging to the edge e and $t_{source(e)}$ the trustworthiness of the originator of this feedback. The formula can be computed by using an iterative computation, similar to (1).

[4] makes explicit the assumption about the peer behavior. The gain is a more efficient algorithm. There is a loss as well; the mechanism is not as robust as [10]. Assume that a peer can perform trustworthy or untrustworthy (1 or 0) in its interactions with others. More precisely, each peer's behavior is modeled as two Bernoulli random variable, i.e., each peer has innate probabilities of performing trustworthy when serving other peers (denote this probability θ_k for peer p_k) and reporting truthfully its experiences with other peers (let l_k be the probability for peer p_k). The distributions of these variables are independent across peers. Consider a peer p_j that interacted with peers $p_1,\ldots,p_n$ and its performances in these interactions were $x_1,\ldots,x_n$, where $x_i \in \{0,1\}$. When asked to report on peer p_j's performances witnesses p_1, $p_2,\ldots,p_n$ may misreport. This happens with their associated misreporting probabilities. The probability of observing report y_k from peer p_k can be calculated as:

$$\begin{aligned} P[Y_k = y_k] &= [l_k(1-\theta_j) \\ &+ (1-l_k)\theta_j]^{y_k}[l_k\theta_j + (1-l_k)(1-\theta_j)]^{1-y_k}. \end{aligned} \tag{2}$$

Given a sample of independent reports $y_1, y_2,\ldots,y_n$, the likelihood function of the sample is computed:

$$L(\theta_j) = P[Y_1 = y_1] \cdot P[Y_2 = y_2] \cdots P[Y_n = y_n]. \tag{3}$$

The final step is finding θ_j that maximizes this expression, i.e., the maximum likelihood estimate of the unknown probability. To do this, one needs to know l_k's in (1.3). [4] proposes a simple method to approximate them. Peer p_i deduce them from its own performances, by comparing own performances with the reports about them. If peer p_i has sufficiently many experiences with peers p_1, $p_2,\ldots,p_n$ as reporters of its performances then it can use them to estimate the misreporting probabilities of those peers. If not, then it can opt to estimate the misreporting probability at the level of the entire network. In this case, all l_k's have the same approximate value, denote it by l, and the maximum likelihood estimate of θ_j becomes $\frac{\bar{y}-l}{1-2l}$, where $\bar{y} = \frac{y_1+\cdots y_n}{n}$.

[10] and [6] on the one hand and [4], on the other, represent different tradeoffs between the computation efficiency and robustness.

So far, an important assumption was that the peer behavior is static. The peers were characterized by probability distributions whose parameters never change. This is an unrealistic assumption. It is dropped in sanctioning reputation systems. As a special case of these systems, game theoretic models of reputation go to the other extreme with respect to the peer behavior. They assume that the involved agents maximize their utilities across their entire lifetime. This means that, at any time instant, the agents condition their actions by the histories of previous play, both theirs and their opponents. The concept of repeated games deals with such long term interactions. However, the game-theoretic models of reputation need one more ingredient. Players are associated with different types (normally, every type assumes a different utility function of a player). Every player knows his type but the others are uncertain about which types of their opponents they face. The game-theoretic tool for modeling such uncertainties is that of games with incomplete information (Bayesian games). Fudenberg and Tirole [5] offer an extensive introduction to the subject.

There are a number of problems with a potential application of game-theoretic reputation models to P2P systems. One deals with human behavior. There are evidences that humans do not behave as utility maximizers (or if they do, it is hard to grasp their utilities within simple economic models). The other one is related to the complexity of game-theoretic models. The problem is that feedback has to be processed by the peers themselves. This leads to enlarged strategy spaces of the peers and complicates the task substantially.

Reputation models of evolutionary game theory present another area of active research. Their assumptions are as follows. A game, most often the celebrated Prisoner's dilemma, is played repeatedly among the players. The play is divided into epochs. In each epoch the players make a choice of (repeated game) strategies. When going to the next epoch, the choice of the players' strategies is biased by their scores in the previous epoch. Hence the name "evolutionary models." Axelrod [2] performed experiments in which he found that one strategy performs particularly well for the Prisoner's dilemma game. He called it "tit-for-tat" -a- it plays at any round whatever the opponent played in the previous round (and starts with cooperation). The setting for his experiments was that in each epoch the same two players played against each other. The question is how much his result extends to setting where the opponents are matched randomly, i.e., their interactions within an epoch are one-shot rather than long-term. Ohtsuki and Ywasa [7] offer an answer in which reputation data plays a critical role. They assume that a public label is associated with each player. All players can read the label, and all players except the owner of the label are allowed to change it. When a pair of players interacts, their labels are modified according to their actions. The behavior of a player can be described with two functions: the *action function* and the *assessment function*. The action function takes the label of self and the opponent and produces the decision to either cooperate or defect. The assessment function is executed after the actions of both agents have taken place. The assessment function takes the label of self, the label of the opponent and the action of the opponent and produces the new value for the opponent's label. There are 16 possible action functions and 256 possible assessment functions. This gives 4096 possible behaviors. [7] performed a systematic experimental study of these behaviors and found 8 of them (termed "the leading eight") evolutionary stable, see Table 1.

A population of agents using one of these strategies is able to sustain cooperation and drive out of existence any small population of defectors and/or reputation liars (i.e., players that set the labels to "bad" value even though their opponent cooperated).

Interestingly, there is a remarkable similarity between tit-for-tat and the leading eight strategies. The leading eight strategies exhibit all the properties of tit-for-tat, found to be important in Axelrod's experiments.

One cannot apply the above reasoning to P2P systems directly. There is a serious problem to solve.

assessment function:

	GG	BG	GB	BB
C	G	*	G	*
D	B	G	B	*

action function:

GG	GB	BG	BB
C	D	C	*

Trust and Reputation in Peer-to-Peer Systems. Table 1. The "leading eight" in the evolutionary indirect reciprocity game. G and B stand for good and bad reputation labels respectively. C and D stand for cooperation and defection. The GG, GB, BG, BB encode four possible states of the labels (of the two players). The three asterisks in the assessment function can take any value, hence eight possible assessment functions are possible (The value at the asterisk in the action function is uniquely determined given the choice of one of the eight assessment functions.)

How to maintain the reputation data when there is no trusted third party to do that, i.e., how to enforce the specific reputation aggregation strategy in a decentralized system? Problems like this constitute an active research area.

Key Applications

P2P applications are provided by participating peers collaboratively. For example, in a file sharing applications, any peer may provide a file that others can download. In a publish-subscribe application, anyone can act as supplier of information that the other peers can use. At the same time, peers are not well established and reputable institutions whose trustworthiness is not questionable. Most often, they are unknown individuals hidden behind meaningless identifiers. This means that reputation management is a natural need in P2P systems. Different P2P applications have varying degrees of need for reputation management, but literally all of them need it.

Interestingly, the operation of core P2P protocols can also benefit from reputation management. When routing messages, peers can take reputation of their neighbors into account and select only those which do not drop messages. This way, reputation management improves routing performance.

Cross-references

▶ Distributed Hash Table
▶ P2P Database
▶ Peer-to-Peer System
▶ Similarity and Ranking Operations
▶ Social Networks
▶ Trust in Blogosphere

Recommended Reading

1. Akerlof G. The market for “lemons”: quality uncertainty and the market mechanism. Quart. J. Econom., 84:488–500, 1970.
2. Axelrod R. The Evolution of Cooperation. Basic Books, New York, 1984.
3. Dellarocas C. The digitization of word-of-mouth: promise and challenges of online reputation systems. Manage. Sci., 49(10):1407–1424, October 2003.
4. Despotovic Z. and Aberer K. Probabilistic prediction of peers performances in P2P networks. Int. J. Eng. Appl. Artif. Intell., 18(7):771–780, Elsevier, October 2005.
5. Fudenberg D. and Tirole J. Game Theory. MIT, Cambridge, MA, USA, 1991.
6. Kamvar S., Schlosser M., and Garcia-Molina H. EigenRep: reputation management in P2P networks. In Proc. 12th Int. World Wide Web Conference, 2003, pp. 640–651.
7. Ohtsuki H. and Iwasa Y. How should we define goodness? – reputation dynamics in indirect reciprocity. J. Theor. Biol., 231:107–120, 2004.
8. Resnick P., Zeckhauser R., Friedman E., and Kuwabara K., Reputation systems. Commun. ACM, 43(12):45–48, 2000.
9. Richardson M., Agrawal R., and Domingos P. Trust management for the semantic web. In Proc. 2nd Int. Semantic Web Conf. SanibelIsland, FL, 2003, pp. 351–368.
10. Xiong L. and Liu L. Peertrust: supporting reputation-based trust in peer-to-peer communities. IEEE Trans. Knowl. Data Eng., 16(7):843–857, 2004.

Trust in Blogosphere

Nitin Agarwal, Huan Liu
Arizona State University, Tempe, AZ, USA

Synonyms

Reputation; Relationship of reliance

Definition

Trust can be defined as the relationship of reliance between two parties or individuals. *Alice* trusts *Bob* implies *Alice*’s reliance on the actions of *Bob*, based on what they know about each other. Trust is basically prediction of an otherwise unknown decision made by a party or an individual based on the actions of another party or individual. Trust is always directional and asymmetric. *Alice* trusts *Bob* does not imply *Bob* also trusts *Alice*.

From a sociological perspective, trust is the measure of belief of one party in another’s honesty, benevolence, and competence. Absence of any of these properties causes failure of trust. From a psychological perspective, trust can be defined as the ability of a party or an individual to influence the other. The more trusting someone is the more easily (s)he can be influenced.

The past several years witnessed significant changes in the interactions between the individuals and groups. Individuals flock on the Internet and engage in complex social relationships, termed as social networks. Social networking has changed the paradigm of interactions and content generation. Former information consumers are now producers (or, Prosumers). Social networking has given a humongous thrust to online communities, like Blogosphere. Blogosphere is the universe of all the blog sites which contains blog posts in reverse chronological order. Each blog post is a discourse of an individual's opinions, ideas, thoughts on some subject matter. These could be journals of personal experiences. Nonetheless, blog posts could be easily considered as a collection of semi-structured text. This opens up many research opportunities for existing text mining techniques to be adapted for this domain. Trust is highly important in a virtual world because of its low barriers to credibility. Profiles and identities could be easily faked and trust could be compromised, leading to severely critical losses, physical and/or mental.

Historical Background

Many existing works have identified the need for handling the trust aspect in social networks. Social networks can be further divided into friendship networks and the blogosphere. In social friendship networks it is important not only to detect the influential members or experts in case of knowledge sharing communities but also to assess to what extent some of the members are recognized as experts by their colleagues in the community. This leads to the estimation of trust and reputation of these experts. Some social friendship networks like Orkut allow users to assign trust ratings as a more explicit notion of trust. Whereas some websites have an implicit notion of trust where creating a link to a person on a webpage implies some amount of business trust for the person. In other cases, Trust and reputation of experts could be typically assessed as a function of the quality of their response to other members' knowledge solicitations. Pujol et al. [5] proposed a *NodeMatching* algorithm to compute the authority or reputation of a node based on its location in the social friendship network. A node's authority depends upon the authority of the nodes that relate to this node and also on other nodes that this node relates to. The basic idea is to propagate the reputation of nodes in the social friendship network. This is very similar to the *PageRank* and *HITS* algorithm in the traditional web search. However, authors point out the differences between their algorithm and *Pagerank* and *HITS*. For *PageRank* and *HITS* the transition probability matrix and variance-covariance matrix respectively have to be known previously, unlike *NodeMatching* algorithm. This becomes infeasible for very large graphs. Moreover, *PageRank* assumes a fixed graph topology by stratifying the range of transition probability which is different in *NodeMatching* which can automatically adapt to the topology since it depends upon the authority of the related nodes.

While Pujol et al. [5] proposed an approach to establish reputation based on the position of each member in the social friendship network [8], developed a model for reputation management based on the Dampster-Shafer theory of evidence in the wake of spurious testimonies provided by malicious members of the social friendship network. Each member of a social friendship network is called an agent. Each agent has a set of acquaintances a subset of which forms its neighbors. Each agent builds a model for its acquaintances to quantify their expertise and sociability. These models are dynamic and change based on the agent's direct interactions with the given acquaintance, interactions with agents referred to by the acquaintance, and on the ratings this acquaintance received from other agents. The authors point out a significant problem with this approach which arises if some acquaintances or other agents generate spurious ratings or exaggerate positive or negative ratings, or offer testimonies that are outright false. Yu and Singh [8] study the problem of deception using the Dampster-Shafer belief functions so as to capture uncertainty in the rankings caused by malicious agents. A variant of majority weighted function is applied to belief functions and simple deception models were studied to detect deception in the ratings.

Sabater and Sierra [6] propose a combination of reputation scores on three different dimensions. They combined reputation scores not only through social relations governed by a social friendship network, termed as social dimension but also past experiences based on individual interactions, termed as individual dimension and reputation scores based on other dimensions, termed as ontological dimension. For large social friendship networks it is not always possible to get reputation scores based on just the individual dimension, so they can use the social dimension and ontological dimension would enhance the reputation estimation by considering different contexts. The ontological dimension is very similar to the work proposed in [7], where the authors recommend collaboration in social friendship networks based on several factors. They explain the importance of context in recommending a member of social friendship network for collaboration.

In [2], authors consider those social friendship networking sites where users explicitly provide trust ratings to other members. However, for large social friendship networks it is infeasible to assign trust ratings to each and every member so they propose an inferring mechanism which would assign binary trust ratings (trustworthy/non-trustworthy) to those who have not been assigned one. They demonstrate the use of these trust values in email filtering application domain and report encouraging results. Authors also assume three crucial properties of trust for their approach to work: transitivity, asymmetry, and personalization. These trust scores are often transitive, meaning, if Alice trusts Bob and Bob trusts Charles then Alice can trust Charles. Asymmetry says that for two people involved in a relationship, trust is not necessarily identical in both directions. This is contrary to what was proposed in [8]. They assume symmetric trust values in the social friendship network between two members. Personalization of trust means that a member could have different trust values with respect to different members. Trust of a member is absolutely a personal opinion. Consolidating the trust scores for a member might not give a reasonable estimation, so authors propose trust propagation mechanism. Authors define source as the node which is seeking trust value of another node called sink. If there is a direct edge between source and sink then the value is directly transferred, otherwise the trust value is inferred based on the source's neighbors. Source polls each of its neighbors whom it has given a positive trust rating. The neighbors also use this procedure to compute the trust rating of the sink. Hence gradually sink's trust scores propagate to the source. They demonstrate the trust rating in filtering emails with the help of a prototype TrustMail and using Enron email dataset. (http://www.cs.cmu.edu/∼enron/). Guha et al. [3] proposed another trust propagation scheme in social friendship networks based on a series of matrix operations but they included the element of distrust along with the trust scores.

The works discussed above rely on one or the other form of network centrality measures (like degree centrality, closeness centrality, betweenness centrality, eigenvector centrality) to evaluate trustworthy nodes and how trust propagates in the network. Nonetheless, blog networks have very sparse trust information between different pairs of nodes. Using trust propagation approaches for such a sparse network would be highly inaccurate and unreliable. Although not much research has been published that exploits text mining to evaluate trust in Blogosphere, authors in [4] have proposed to use sentiment analysis of the text around the links to other blogs in the network. They study the link polarity and label the sentiment as "positive," "negative," or "neutral." This information mined from the blogs is coupled with Guha et al.'s [3] trust and distrust propagation approach to derive trust values between node pairs in the blog network. They further use this model to identify trustworthy nodes in the blog network and also identify clusters of like-minded blogs.

Foundations

Quantifying and computing trust in social networks is hard because concepts like trust are fuzzy, and is being expressed in a social way. The definitions and properties are not mathematical formalisms but social ones. The two main components of defining trust are belief and commitment. The extent to which someone trusts another is illustrated by the belief and trusted individuals maintain that with their commitment. Note that trust is highly subjective, nevertheless some characteristic properties are pointed:

Transitivity: Trust can propagate through different nodes following transitive property. However, the degree of trust does not remain same. It may decrease as the path length through which trust propagates increases.
Asymmetry: Trust is asymmetric, in the sense that if A trusts B then it is not necessary that B also trusts A. Some existing works relax this assumption and consider trust as symmetric.
Personalization: Trust is a personalized concept. Everyone has a different conception of trust with respect to some other individual. Assigning a global trust value to an individual is highly unrealistic. Trustworthiness of an individual is always evaluated with respect to some other individual.

Trust can be considered as binary-valued with 1 indicating trust and 0 indicating "not-trusted." Trust can also be evaluated as continuous-valued. Moreover, binary-valued trust is little more complicated than meets the eye. A value of 0 could be a little vague as it could represent both "no-opinion" or "distrust." To qualify this notion, often researchers use − 1 to represent distrust and 0 as missing value or "no-opinion." Researchers model the propagation of distrust the same way as the propagation of trust. Propagation of trust (T) and distrust (D) could be governed by the set of rules illustrated in Table 1. Here *A*, *B*, and *C* are

Trust in Blogosphere. Table 1. Rules for trust and distrust propagation

Propagation Scheme	Outcome	Comments
$A \overset{T}{\rightarrow} B \overset{T}{\rightarrow} C$	$A \overset{T}{\rightarrow} C$	*Transitivity*
$A \overset{T}{\rightarrow} B \overset{D}{\rightarrow} C$	$A \overset{D}{\rightarrow} C$	Don't trust someone who is distrusted by a person you trust.
$A \overset{D}{\rightarrow} B \overset{T}{\rightarrow} C$	$A \overset{D}{\rightarrow} C$	Don't trust someone who is trusted by a person you don't trust.
$A \overset{D}{\rightarrow} B \overset{D}{\rightarrow} C$	(1) $A \overset{T}{\rightarrow} C$	Enemy of yopur enemy is your friend.
	(2) $A \overset{D}{\rightarrow} C$	Don't trust someone who is not trusted by a person you don't trust.

different individuals and trust or distrust relationship between *A–B* and *B–C* is known. These rules help in inferring trust or distrust between *A–B*. Propagation of distrust is a little intricate. As shown in the Table 1, if *A* distrusts *B* and *B* distrusts *C* then *A* has reasons for either trusting *C* (enemy of enemy is a friend) or distrusting *C* (do not trust someone who is not trusted by someone else that is not trusted).

In case the link between *A* and *C*, like *B* is missing, which can be used to infer the trust between *A–C*, a different strategy could be used. Trust only if someone is trusted by *k* people, i.e., if *C* is trusted by a *k* number of people then *A* could trust *C*. Do not trust anyone who is distrusted by k' people, i.e., if *C* is distrusted by k' number of people then *A* could distrust *C*. Note that the thresholds *k* and k' could be learned from the data.

Key Applications

Trust in social networks has several applications. Trust and reputation based spam email filters have become popular after naïve spam email filters. The social network information of senders and recipients could be exploited to study trust among them and filter emails based on these values. Trust acts as lubricant that improves information flow and promotes frankness and honesty. Trust can also be helpful in online discussion forums where users look for informative and trustworthy answers to their questions. Giving trustworthy recommendations could also improve the customer retention policies.

Future Directions

Trust is a promising area of research in social networks, especially the blogosphere, where most of the assumptions from friendship networks are absent.

1. Social friendship networks assume initial trust values are assigned to the nodes of the network. Unless some social networking websites allow their members to explicitly provide trust ratings for other members, it is a topic of research and exploration to compute initial trust scores for the members. Moreover, in Blogosphere it is even harder to implicitly compute initial trust scores.
2. Social friendship networks assume an explicit relationship between members of the network. However, in Blogosphere there is no concept of explicit relationship between bloggers. Many times, these relationships have to be anticipated using link structure in the blogs or blogging characteristics of the bloggers.
3. Existing works of trust propagation algorithms assume an initial starting point. In Blogosphere, where both network structure and initial ratings are not explicitly defined, it is challenging to tackle the trust aspect. A potential approach could be to use influential members [1] of a blog community as the seeds for trustworthy nodes.
4. Since text mining has not been sufficiently exploited in Blogosphere domain, several promising research opportunities can be explored.

Data Sets

The following datasets are widely used by many researchers in this area:

A website (http://www.epinions.com/) that maintains trust values for all the available products/services provided by the customers.

Epinions: Movie Recommendation: Netflix (http://www.netflixprize.com/) provides movie recommendation dataset and what recommendations were followed by the customers. Research works have engineered this dataset to evaluate trust among customers.

Enron Email Dataset: A collection of emails that contains both genuine and spam emails. Researchers constructed social network between senders and recipients of the email and studied trust aspect.

Cross-references

▶ Actors/Agents/Roles

▶ Social Networks

▶ Trust and Reputation in Peer-to-Peer Systems

Recommended Reading

1. Agarwal N., Liu H., Tang L., and Yu P.S. Identifying the influential bloggers in a community. In Proc. Int. Conf. Web Search and Web Data Mining, 2008, pp. 207–218.
2. Golbeck J. and Hendler J. Inferring binary trust relationships in web-based social networks. ACM Trans. Inter. Tech., 6(4):497–529, 2006.
3. Guha R., Kumar R., Raghavan P., and Tomkins A. Propagation of trust and distrust. In Proc. 12th Int. World Wide Web Conference, 2004, pp. 403–412.
4. Kale A., Karandikar A., Kolari P., Java A., Finin T., and Joshi A. Modeling trust and influence in the blogosphere using link polarity. In Proc. 1st Int.'l AAAI Conf. on Weblogs and Social Media, 2007.
5. Pujol J.M., Sangesa R., and Delgado J. Extracting reputation in multi agent systems by means of social network topology. In Proc. 1st Int. Joint Conf. on Autonomous Agents and Multiagent Systems, 2002, pp. 467–474.
6. Sabater J. and Sierra C. Reputation and social network analysis in multi-agent systems. In Proc. 1st Int. Joint Conf. on Autonomous Agents and Multiagent Systems, 2002, pp. 475–482.
7. Terveen L. and McDonald D.W. Social matching: a framework and research agenda. ACM Trans. Comput.-Hum. Interact., 12(3):401–434, 2005.
8. Yu B. and Singh M.P. Detecting deception in reputation management. In Proc. 2nd Int. Joint Conf. on Autonomous Agents and Multiagent Systems, 2003, pp. 73–80.

Trusted Database Systems

▶ Multilevel Secure Database Management Systems

Trusted Hardware

Radu Sion
Stony Brook University, Stony Brook, NY, USA

Synonyms

Tamper-proof hardware; Secure hardware

Definition

Trusted Hardware is a broad term used to denote any hardware that has been certified to perform according to a certain set of requirements. Most often however, "trusted hardware" is discussed in adversarial contexts. The term has thus been somewhat hijacked to mean "tamper-proof" hardware, i.e., hardware designed to resist direct physical access adversaries. Often trusted hardware encompasses some cryptographic abilities, i.e., performing encryption and data authentication.

Key Points

Certification. The National Institute of Standards has established a set of standards for security requirements of cryptographic modules and specifically for physical properties and tamper-resistance thereof [2]. The FIPS 140–2 Level 4 certification is at present the highest-attainable hardware security in sensitive, non-classified domains. While a plethora of devices have undergone FIPS certification, the most common types of trusted hardware in use today are TPM micro-controllers and secure CPUs: ***TPM.*** The Trusted Platform Module (TPM) specifications of the Trusted Computing Group [3] define a micro-controller that stores keys, passwords and digital certificates. In actual instances TPMs are connected to the main circuitry of computing devices (such as PC motherboards) and ensure that the stored data are secure from external software attacks. It is important to note however, that a TPM "can only act as a 'lave' to higher level services and applications by storing and reporting pre-runtime configuration information. [...]. At no time can the TCG building blocks 'control' the system or report the status of applications that are running." This passive nature limits the TPM's utility in security paradigms that require active processing.

SCPUs. Secure CPUs (SCPUs) are a term used to denote general-purpose CPUs deployed in a certified tamper-proof enclosure. Instances include the IBM 4758 PCI and the newer IBM 4764 PCI-X cryptographic coprocessors [1]. The IBM 4764 is a PowerPC-based board and runs embedded Linux. The 4758 is based on a Intel 486 architecture and is preloaded with a compact runtime environment that allows the loading of arbitrary external certified code. The CPUs can be custom programmed. Moreover, they (4758 models 2 and 23 and 4764 model 1) are compatible with the IBM Common Cryptographic Architecture (CCA) API. The CCA implements common cryptographic services such as random number generation, key management, digital signatures, and encryption (DES/3DES, RSA). If physically attacked, the devices destroy their internal state (in a process powered by internal long-term batteries) and shut down in accordance with their FIPS 140-2 certification. It is important to note that SCPUs are generally one order of magnitude slower than main processors mainly due to heat dissipation constraints limiting the maximum allowable gate-density within the tamper-proof enclosure.

Cross-references

▶ Regulatory Compliance in Data Management

Recommended Reading

1. IBM Cryptographic Hardware. Online at http://www-03.ibm.com/security/products/, 2007.
2. NIST Federal Information Processing Standards. Online at http://csrc.nist.gov/publications/fips/, 2007.
3. Trusted Computing Group. Online at http://www.trustedcomputinggroup.org/, 2007.

TSQL2

Richard T. Snodgrass
University of Arizona, Tucson, AZ, USA

Definition

TSQL2 (*Temporal Structured Query Language*) is a temporal extension of SQL-92 designed in 1993–1994 by a committee comprised of Richard T. Snodgrass, Ilsoo Ahn, Gad Ariav, Don Batory, James Clifford, Curtis E. Dyreson, Ramez Elmasri, Fabio Grandi, Christian S. Jensen, Wolfgang Käfer, Nick Kline, Krishna Kulkarni, T. Y. Cliff Leung, Nikos Lorentzos, John F. Roddick, Arie Segev, Michael D. Soo and Suryanarayana M. Sripada. The goal of this language design committee was to consolidate past research on temporal query languages, by the committee members as well as many others, by developing a specification for a consensus language that could form a common core for research.

Historical Background

Temporal databases have been an active research topic since 1980. By the early 1990's, several dozen temporal query languages had been proposed, and many temporal database researchers felt that the time had come to consolidate approaches to temporal data models and calculus-based query languages to achieve a consensus query language and associated data model upon which future research could be based.

In April 1992, Richard Snodgrass circulated a white paper that proposed that a temporal extension to SQL be produced by the research community. Shortly thereafter, the temporal database community organized the ARPA/NSF International Workshop on an Infrastructure for Temporal Databases, held in Arlington Texas in June 1993 [3]. Discussions at that workshop indicated that there was substantial interest in a temporal extension to the conventional relational query language SQL-92 [2]. A general invitation was sent to the community, and about a dozen people volunteered to develop a language specification. Several people later joined the committee. The goal of this language design committee was to develop a specification for a consensus extension to SQL-92, termed the *Temporal Structured Query Language*, or TSQL2.

The group corresponded via electronic mail from early July 1993, submitting, debating, and refining proposals for the various aspects and elements of the language. In September 1993, the first draft specification, accompanied by 13 commentaries, was distributed to the committee. In December 1993, a much enlarged draft, accompanied by some 24 commentaries, was distributed to the committee. A preliminary language specification was made public in March 1994 [6], and a tutorial of the language appeared in September 1994 [7]. The final language specification and 28 commentaries were also made available via anonymous FTP in early October 2004. The final specification and commentaries appeared in a book [4] that was distributed at a temporal database workshop in summer of 1995, less than 2 years after the committee had been founded. TSQL2 is remarkable, and perhaps unique, in that it was designed entirely via electronic mail, by a committee that never met physically (in fact, no one on the committee has met every other committee member).

Work then commenced to incorporate elements and underlying insights of TSQL2 into SQL3. The first step was to propose a new part to SQL3, termed SQL/Temporal. This new part was accepted at the Ottawa meeting in January, 1995 as Part 7 of the

SQL3 specification [11]. A modification of TSQL2's PERIOD data type is included in that part.

The focus at that point changed to adding valid-time and transaction-time support to SQL/Temporal. Two change proposals, one on valid-time support and one on transaction-time support, were unanimously accepted by ANSI and forwarded to ISO [8,9]; a summary appeared shortly thereafter [10]. A comprehensive set of case studies [5] showed that while SQL-92 required 1,848 lines, only 520 lines of SQL/Temporal were required to achieve exactly the same functionality. These case studies showed that, over a wide range of data definition, query, and modification fragments, the SQL-92 version is *three* times longer in numbers of lines than the SQL/Temporal version, and many times more complex. In fact, very few SQL/Temporal statements were more than ten lines long; some statements in SQL-92 comprised literally dozens of lines of highly complex code. Due to disagreements within the ISO committee as to where temporal support in SQL should go, the project responsible for temporal support was canceled near the end of 2001. Nevertheless, concepts and constructs from SQL/Temporal have been implemented in the Oracle database management system, and other products have also included temporal support.

Oracle 9i includes support for transaction time. Its flashback queries allow an application to access prior transaction-time states of its database; they are transaction timeslice queries. Database modifications and conventional queries are temporally upward compatible. Oracle 10g extends flashback queries to retrieve all the versions of a row between two transaction times (a key-transaction-time-range query) and allows tables and databases to be rolled back to a previous transaction time, discarding all changes after that time. The Oracle 10g Workspace Manager includes the period data type, valid-time support, transaction-time support, support for bitemporal tables, and support for sequenced primary keys, sequenced uniqueness, sequenced referential integrity, and sequenced selection and projection, in a manner quite similar to that proposed in SQL/Temporal.

Foundations

The goals that underpinned the process that led to the TSQL2 language design are first considered, then the language concepts underlying TSQL2 are reviewed.

Design Goal for TSQL2

TSQL2 is a temporal query language, designed to query and manipulate time-varying data stored in a relational database. It is an upward-compatible extension of the international standard relational query language SQL-92.

The TSQL2 language design committee started their work by establishing a number of ground rules with the objective of achieving a coherent design.

- TSQL2 will be a *language* design.
- TSQL2 is to be a *relational* query language, not an object-oriented query language.
- TSQL2 should be consistent with existing standards, not another standard.
- TSQL2 should be comprehensive and should reflect areas of convergence.
- The language will have an associated algebra.

The committee enumerated the desired features of TSQL2; these guided the design of the language. The first batch concerned the data model itself.

- TSQL2 should not distinguish between snapshot equivalent instances, i.e., snapshot equivalence and identity should be synonymous.
- TSQL2 should support only one valid-time dimension.
- For simplicity, tuple timestamping should be employed.
- TSQL2 should be based on homogeneous tuples.
- Valid-time support should include support for both the past and the future.
- Timestamp values should not be limited in range or precision.

The next concerned the language proper.

- TSQL2 should be a consistent, fully upwardly compatible extension of SQL-92.
- TSQL2 should allow the restructuring of tables on any set of attributes.
- TSQL2 should allow for flexible temporal projection, but TSQL2 syntax should reveal clearly when non-standard temporal projections are being done.
- Operations in TSQL2 should not accord any explicit attributes special semantics. For example, operations should not rely on the notion of a key.

- Temporal support should be optional, on a per-table basis. Tables not specified as temporal should be considered as snapshot tables. It is important to be an extension of SQL-92's data model when possible, not a replacement. Hence, the schema definition language should allow the definition of snapshot tables. Similarly, it should be possible to derive a snapshot table from a temporal table.
- User-defined time support should include instants, periods, and intervals.
- Existing aggregates should have temporal analogues in TSQL2.
- Multiple calendar and multiple language support should be present in timestamp input and output, and timestamp operations. SQL-92 supports only one calendar, a particular variant of the Gregorian calendar, and one time format. The many uses of temporal databases demand much more flexibility.
- It should be possible to derive temporal and non-temporal tables from underlying temporal and non-temporal tables.

Finally, the committee agreed upon three features aimed at ease of implementation.

- TSQL2 tables should be implementable in terms of conventional first normal form tables. In particular, the language should be implementable via a data model that employs period-timestamped tuples. This is the most straightforward representation, in terms of extending current relational technology.
- TSQL2 must have an efficiently implementable algebra that allows for optimization and that is an extension of the conventional relational algebra, on which current DBMS implementations are based. The temporal algebra used with the TSQL2 temporal data model should contain temporal operators that are extensions of the operations in the relational algebra. Snapshot reducibility is also highly desired, so that, for example, optimization strategies will continue to work in the new data model.
- The language data model should *accept* implementation using other models, such as models that timestamp attribute values. The language data model should allow multiple representational data models. In particular, it would be best if the data model accommodated the major temporal data models proposed to date, including attribute timestamped models.

Language Concepts in TSQL2

The following is a brief outline of the major concepts behind TSQL2.

Time Ontology The TSQL2 model of time is bounded on both ends. The model refrains from deciding whether time is ultimately continuous, dense, or discrete. Specifically, TSQL2 does not allow the user to ask a question that will differentiate the alternatives. Instead, the model accommodates all three alternatives by assuming that an instant on a time-line is much smaller than a chronon, which is the smallest entity that a timestamp can represent exactly (the size of a chronon is implementation-dependent). Thus, an instant can only be approximately represented. A discrete image of the represented times emerges at run-time as timestamps are scaled to user-specified (or default) granularities and as operations on those timestamps are performed to the given scale.

An instant is modeled by a timestamp coupled with an associated scale (e.g., day, year, month). A period is modeled by a pair of two instants in the same scale, with the constraint that the instant timestamp that starts the period equals or precedes (in the given scale) the instant timestamp that terminates the period.

Base Line Clock A semantics must be given to each time that is stored in the database. SQL-92 specifies that times are given in UTC seconds, which are, however, not defined before 1958, and in any case cannot be used to date prehistoric time, as UTC is based in part on solar time. TSQL2 includes the concept of a *baseline clock*, which provides the semantics of timestamps. The baseline clock relates each second to physical phenomena and partitions the time line into a set of contiguous *periods*. Each period runs on a different clock. *Synchronization points* delimit period boundaries. The baseline clock and its representation are independent of any calendar.

Data Types SQL-92's datetime and interval data types are augmented with a *period* datetime, of specifiable range and precision. The range and precision can be expressed as an integer (e.g., a precision of 3 fractional digits) or as an interval (e.g., a precision of a week). Operators are available to compare timestamps and to compute new timestamps, with a user-specified precision. Temporal values can be input and output in user-specifiable formats, in a variety of natural languages. *Calendars* and *calendric systems* permit the application-dependent semantics of time to be incorporated.

A surrogate data are introduced in TSQL2. Surrogates are unique identifiers that can be compared for equality, but the values of which cannot be seen by the users. In this sense, a surrogate is "pure" identity and does not describe a property (i.e., it has no observable value). Surrogates are useful in tying together representations of multiple temporal states of the same object; they are not a replacement for keys.

Time-Lines Three time-lines are supported in TSQL2: user-defined time, valid time, and transaction time. Hence values from disparate time-lines can be compared, at an appropriate precision. Transaction-time is bounded by `initiation`, the time when the database was created, and until changed. In addition, user-defined and valid time have two special values, beginning and forever, which are the least and greatest values in the ordering. Transaction time has the special value until changed.

Valid and user-defined times can be indeterminate. In *temporal indeterminacy*, it is known that an event stored in a temporal database did in fact occur, but it is not known exactly *when* that event occurred. An instant (interval, period) can be specified as determinate or indeterminate; if the latter then the possible mass functions, as well as the generality of the indeterminacy to be represented, can be specified. The quality of the underlying data (termed its *credibility*) and the *plausibility* of the ordering predicates expressed in the query can be controlled on a per-query or global basis.

Finally, instant timestamps can be *now-relative*. A now-relative time of "*now* – 1 day," interpreted when the query was executed on June 12, 1993, would have the *bound* value of "June 11, 1993." The users can specify whether values to be stored in the database are to be bound (i.e., not now-relative) or unbound.

Aggregates The conventional SQL-92 aggregates are extended to take into account change across time. They are extended to return time-varying values and to permit grouping via a partitioning of the underlying time line, termed *temporal grouping*. Values can be *weighted* by their duration during the computation of an aggregate. Finally, a new temporal aggregate, `RISING`, is added. A taxonomy of temporal aggregates [4, Chap. 21] identifies 14 possible kinds of aggregates; there are instances of all of these kinds in TSQL2.

Valid-Time Tables The snapshot tables supported by SQL-92 continue to be available in TSQL2, which, in addition, supports *state* tables, where each tuple is timestamped with a *temporal element* that is a union of periods. As an example, the Employee table with attributes Name, Salary, and Manager could contain the tuple (Tony, 10,000, LeeAnn). The temporal element timestamp would record the maximal (non-contiguous) periods in which Tony made $10,000 and had LeeAnn as his manager. Information about other values of Tony's salary or other managers would be stored in other tuples. The timestamp is implicitly associated with each tuple; it is not another column in the table. The range, precision, and indeterminacy of a temporal element can be specified.

Temporal elements are closed under union, difference, and intersection. Timestamping tuples with temporal elements is conceptually appealing and can support multiple representational data models. Dependency theory can be extended to apply in full to this temporal data model.

TSQL2 also supports *event* tables, in which each tuple is timestamped with an *instant set*. As an example, a Hired table with attributes Name and Position could contain the tuple (LeeAnn, Manager). The instant set timestamp would record the instant(s) when LeeAnn was hired as a Manager. (Other information about her positions would be stored in separate tables.) As for state tables, the timestamps are associated implicitly with tuples.

Transaction-Time and Bitemporal Tables Orthogonally to valid time, transaction time can be associated with tables. The transaction time of a tuple, which is a temporal element, specifies when that tuple was considered to be part of the current database state. If the tuple (Tony, 10,000, LeeAnn) was stored in the database on March 15, 1992 (say, with an `APPEND` statement) and removed from the database on June 1, 1992 (say, with a `DELETE` statement), then the transaction time of that tuple would be the period from March 15, 1992 to June 1, 1992.

Transaction timestamps have an implementation-dependent range and precision, and they are determinate.

In summary, there are six kinds of tables: snapshot (no temporal support beyond user-defined time), valid-time state tables (timestamped with valid-time elements), valid-time event tables (timestamped with valid-time instant sets), transaction-time tables (timestamped with transaction-time elements), bitemporal state tables (timestamped with bitemporal elements), and bitemporal event tables (timestamped with bitemporal instant sets).

Schema Specification The CREATE TABLE and ALTER statements allow specification of the valid- and transaction-time aspects of temporal tables. The scale and precision of the valid timestamps can also be specified and later altered.

Restructuring The FROM clause in TSQL2 allows tables to be *restructured* so that the temporal elements associated with tuples with identical values on a subset of the columns are coalesced. For example, to determine when Tony made a Salary of $10,000, independent of who his manager was, the Employee table could be restructured on the Name and Salary columns. The timestamp of this restructured tuple would specify the periods when Tony made $10,000, information which might be gathered from several underlying tuples specifying different managers.

Similarly, to determine when Tony had LeeAnn as his manager, independent of his salary, the table would be restructured on the Name and Manager columns. To determine when Tony was an employee, independent of how much he made or who his manager was, the table could be restructured on only the Name column.

Restructuring can also involve *partitioning* of the temporal element or instant set into its constituent maximal periods or instants, respectively. Many queries refer to a continuous property, in which maximal periods are relevant.

Temporal Selection The valid-time timestamp of a table may participate in predicates in the WHERE clause by via VALID() applied to the table (or correlation variable) name. The transaction-time of a table can be accessed via TRANSACTION(). The operators have been extended to take temporal elements and instant sets as arguments.

Temporal Projection Conventional snapshot tables, as well as valid-time tables, can be derived from underlying snapshot or valid-time tables. An optional VALID or VALID INTERSECT clause is used to specify the timestamp of the derived tuple. The transaction time of an appended or modified tuple is supplied by the DBMS.

Update The update statements have been extended in a manner similar to the SELECT statement, to specify the temporal extent of the update.

Cursors Cursors have been extended to optionally return the valid time of the retrieved tuple.

Schema Versioning Schema *evolution*, where the schema may change, is already supported in SQL-92. However, old schemas are discarded; the data are always consistent with the current schema. Transaction time support dictates that previous schemas be accessible, which calls for *schema versioning*. TSQL2 supports a minimal level of schema versioning.

Vacuuming Updates, including (*logical*) deletions, to transaction time tables result in insertions at the physical level. Despite the continuing decrease in cost of data storage, it is still, for various reasons, not always acceptable that all data be retained forever. TSQL2 supports a simple form of *vacuuming*, i.e., physical deletion, from such tables.

System Tables The TABLE base table has been extended to include information on the valid and transaction time components (if present) of a table. Two other base tables have been added to the definition schema.

SQL-92 Compatibility All aspects of TSQL2 are pure extensions of SQL-92. The user-defined time in TSQL2 is a consistent replacement for that of SQL-92. This was done to permit support of multiple calendars and literal representations. Legacy applications can be supported through a default SQL92_calendric_system.

The defaults for the new clauses used to support temporal tables were designed to satisfy snapshot reducibility, thereby ensuring that these extensions constitute a strict superset of SQL-92.

Implementation During the design of the language, considerable effort was expended to ensure that the language could be implemented with only moderate modification to a conventional, SQL-92-compliant DBMS. In particular, an algebra has been demonstrated that can be implemented in terms of a period-stamped (or instant-stamped, for event tables) tuple representational model; few extensions to the conventional algebra were required to fully support the TSQL2 constructs. This algebra is snapshot reducible to the conventional relational algebra. Support for multiple calendars, multiple languages, mixed precision, and indeterminacy have been included in prototypes that demonstrated that these extensions have little deleterious effect on execution performance. Mappings from the data model underlying TSQL2, the bitemporal conceptual data model [1], to various representational data models have been defined [4].

Key Applications

TSQL2 continues to offer a common core for temporal database research, as well as a springboard for change proposals for extensions to the SQL standard.

Future Directions

Given the dramatic decrease in code size and complexity for temporal applications that TSQL2 and SQL/Temporal offers, it is hoped that other DBMS vendors will take Oracle's lead and incorporate these proposed language constructs into their products.

Url to Code

http://www.cs.arizona.edu/people/rts/tsql2.html This web page includes links to the ISO documents. http://www.sigmod.org/dblp/db/books/collections/snodgrass95.html

Cross-references

- ▶ Applicability Period
- ▶ Fixed Time Span
- ▶ Now in Temporal Databases
- ▶ Period-Stamped Temporal Models
- ▶ Span
- ▶ Schema Versioning
- ▶ Temporal Aggregation
- ▶ Temporal Algebras
- ▶ Temporal Compatibility
- ▶ Temporal Integrity Constraints
- ▶ Temporal Joins
- ▶ Temporal Logical Models
- ▶ Temporal Query Languages
- ▶ Temporal Vacuuming
- ▶ Time-Line Clock
- ▶ Transaction Time
- ▶ TUC
- ▶ Until Changed
- ▶ Valid Time
- ▶ Value Equivalence

Recommended Reading

1. Jensen C.S., Soo M.D. and Snodgrass R. T. Unifying Temporal Data Models via a Conceptual Model. Inf. Syst., 19(7):513–547, December 1994.
2. Melton J. and Simon A.R. Understanding the New SQL: A Complete Guide. Morgan Kaufmann, San Mateo, CA, 1993.
3. Snodgrass R.T. (ed.). In Proc. Int. Workshop on an Infrastructure for Temporal Databases, 1993.
4. Snodgrass R.T. (ed.). The TSQL2 Temporal Query Language. Kluwer Academic, 1995.
5. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, San Francisco, CA, July 1999.
6. Snodgrass R.T., Ahn I., Ariav G., Batory D.S., Clifford J., Dyreson C.E., Elmasri R., Grandi F., Jensen C.S., Käfer W., Kline N., Kulkarni K., Leung T.Y.C., Lorentzos N., Roddick J.F., Segev A., Soo M.D., and Sripada S.M., TSQL2 Language Specification. ACM SIGMOD Rec., 23(1):65–86, March 1994.
7. Snodgrass R.T., Ahn I., Ariav G., Batory D., Clifford J., Dyreson C.E., Elmasri R., Grandi F., Jensen C.S., Käfer W., Kline N., Kulkarni K., Leung T.Y.C., Lorentzos N., Roddick J.F., Segev A., Soo M.D., and Sripada S.M. A TSQL2 tutorial. ACM SIGMOD Rec., 23(3):27–33, September 1994.
8. Snodgrass R.T., Böhlen M.H., Jensen C.S. and Steiner A. Adding Transaction Time to SQL/Temporal. Change proposal, ANSI X3H2-96-502r2, ISO/IEC JTC1/SC21/ WG3 DBL MAD-147r2, November 1996.
9. Snodgrass R.T., Böhlen M.H., Jensen C.S. and Steiner A. Adding Valid Time to SQL/Temporal. change proposal, ANSI X3H2-96-501r2, ISO/IEC JTC1/SC21/ WG3 DBL MAD-146r2, November 1996.
10. Snodgrass R.T., Böhlen M.H., Jensen C.S., and Steiner A., Transitioning Temporal Support in TSQL2 to SQL3. In Temporal Databases: Research and Practice, O. E.zion, S. Jajodia, S.M. Sripada (eds.). Springer, Berlin, 1998, pp. 150–194.
11. Snodgrass R.T., Kulkarni K., Kucera H., and Mattos N. Proposal for a new SQL Part – Temporal. ISO/IEC JTC1/SC21 WG3 DBL RIO-75, X3H2-94-481, November 2, 1994.

Tug-of-War Sketch

▶ AMS Sketch

Tuning Concurrency Control

Philippe Bonnet[1], Dennis Shasha[2]
[1]University of Copenhagen, Copenhagen, Denmark
[2]New York University, New York, NY, USA

Synonyms

Lock tuning

Definition

Database systems implement concurrency control to give users the illusion that each transaction executes correctly, in isolation from all others. The concurrency-control algorithm in predominant use is two-phase locking. Tuning concurrency control consists in improving the performance of concurrent operations by reducing the number, duration and scope of the conflicts due to locking.

Historical Background

In 1976, Jim Gray et al. identified the fundamental concurrency control trade-off between correctness and performance. They discussed different lock granularities and introduced the notion of degrees of consistency.

Foundations

Database systems attempt to give users the illusion that each transaction executes in isolation from all others. The ANSI SQL standard, for example, makes this explicit with its concept of degrees of isolation. Full isolation or *serializability* is the guarantee that each transaction that completes will appear to execute one at a time, except that its performance may be affected by other transactions. Choosing a lower level of isolation will benefit performance, possibly at the cost of correctness. The value of serializability experiment (see below in experimental results) illustrates this performance/correctness trade-off. This entry discusses basic concurrency tuning techniques.

Leveraging Application Semantics

Efficient tuning often entails understanding application semantics. A frequently required feature is to assign consecutive key values to consecutive records (e.g., customer numbers, purchase order numbers). Consider a straightforward implementation.

In the following example, the COUNTER table contains the next value which is used as a key when inserting values in the ACCOUNT table.

```
begin transaction
  NextKey:=select nextkey from COUNTER;
  insert into ACCOUNT values (nextkey, 100, 200);
  update COUNTER set nextkey=NextKey+1;
end transaction
```

When the number of such transactions issued concurrently increases, COUNTER becomes a bottleneck because all transactions read and write the value of nextkey.

An alternative approach is to use the facilities that many systems offer that reduce the length of time counter locks are held. These facilities (sequences in Oracle, autoincrement in MySQL and identity in SQL Server, DB2 UDB and Sybase Adaptive Server) enable transactions to hold a latch (see the latch definitional entry) on the counter only while accessing the counter, rather than until the transaction completes. This eliminates the counter as a bottleneck but may introduce a small problem.

Consider an insert transaction T that increments the counter then aborts. Before T aborts, a second transaction T' may increment the counter further. Thus, the counter value obtained by T will not be associated with any data item. That is, there may be gaps in the counter values. Most applications can tolerate such gaps, but some cannot for legal reasons, e.g., tax authorities prefer that invoice numbers have no gaps.

Living Dangerously

Many applications live with less than full isolation due to the high cost of holding locks during user interactions. Consider the following full-isolation transaction from an airline reservation application:

```
Airline Reservation Transaction
  Begin transaction
    Retrieve list of seats available;
    Reservation agent talks with customer regarding
    availability;
    Secure seat.
  End transaction
```

The performance of a system built from such transactions would be intolerably slow, because each customer would hold a lock on all available seats for a flight while chatting with the reservations agent. This solution does, however, guarantee two conditions: (i) no two customers will be given the same seat, and (ii) any seat that the reservation agent identifies as available in view of the retrieval of seats will still be available when the customer asks to secure it.

Because of the poor performance, however, the following is done instead:

```
Loosely Consistent Airline Reservation
  Begin transaction
    Retrieve list of seats available;
    Reservation agent talks with customer regarding
    availability;
    Secure seat.
  End transaction
```

This design relegates lock conflicts to the secure step, thus guaranteeing that no two customers will be given the same seat. It does allow the possibility, however, that a customer will be told that a seat is available, will ask to secure it, and will then find out that it is gone.

General Rules of Thumb

By looking at blocking and (more rarely) deadlock statistics, an administrator or advanced application user can infer the existence of a concurrency control bottleneck. What should follow is careful analysis of the application to see (i) whether transactions can be redesigned to place accesses to hot items at the ends of the transactions, (ii) whether system facilities may help to reduce concurrency bottlenecks, (iii) or whether the application semantics allow a lesser form of concurrency correctness guarantee for the sake of performance. The general idea is to reduce the hold on the few critical resources that cause the concurrency bottleneck.

Key Applications

Concurrency control tuning is essential for applications having frequent modifications (inserts, deletes, and/or updates), because those applications entail lock conflicts.

Experimental Results

Value of Serializability

This experiment illustrates the correctness/performance trade-off associated to the two isolation levels, i.e., serializable and read committed. Consider a table of the form R(a int primary key, b int), on which an application executes two types of transactions: (i) a *sum* transaction that computes the sum of *b* values, and (ii) *swap* transactions that swap *b* values. The experiment consists in executing the two types of transactions concurrently. The parameters of the experiments are (i) the level of isolation, and (ii) the number of concurrent threads executing the swap transactions (note that the total number of transactions is kept constant throughout the experiment). Response time is measured for the total number of transactions.

The results presented below were obtained with MySQL SQL 6.0 (using InnoDB with a 1GB buffer pool), running on a Linux server equipped with an Intel Core 2 Duo processor (the cache is warm during this experiment). The SQL code and the data used for this experiment as well as the data set with the measurements are available at the URL listed at the end of this entry.

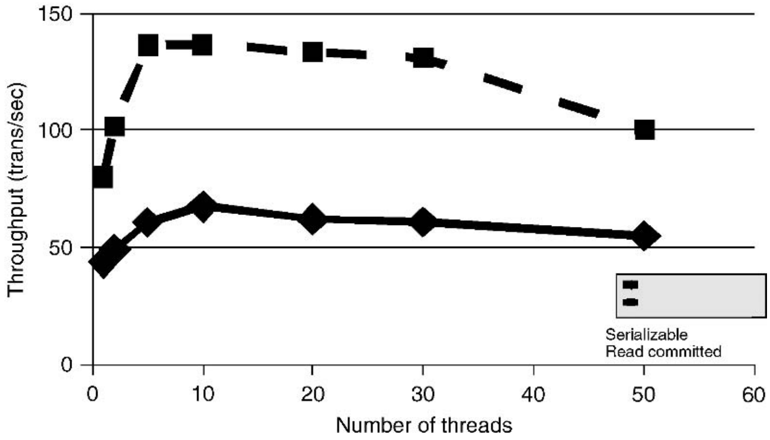
Interestingly, both the read committed and serializable isolation levels yield 100% correct answer regardless of the number of threads executing the *swap* transactions. The reason is that MySQL (like Oracle) implements snapshot isolation: the result of the *sum* transaction is obtained on the R table as it stood when that transaction began, i.e., the *swap* transactions do not impact the result of the *sum* transaction. When executed on database systems that do not implement snapshot isolation (e.g., SQLServer or DB2), only about 40% of the results were correct for the *sum* transaction [3].

Figure 1 traces throughput (i.e., total number of transactions/response time) for the serializable and read committed level as a function of the number of swap threads.

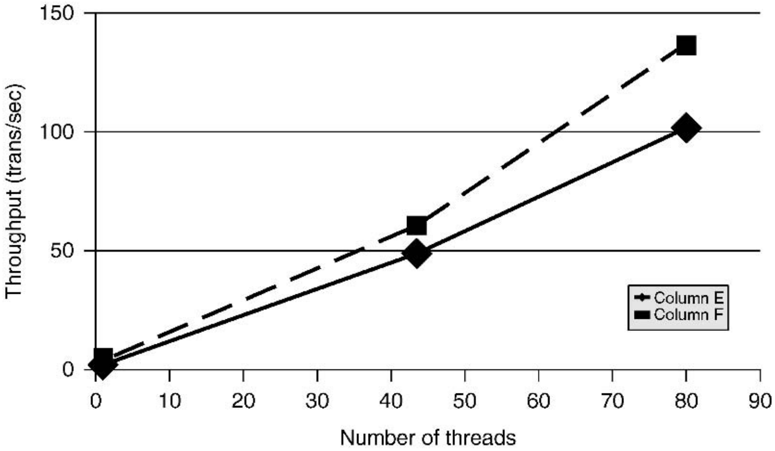
Read committed yields a higher throughput than serializable. The reason is that in the serializable isolation level, the *sum* transaction sets next-key locks while scanning the table, whereas at the read committed isolation level, the sum transaction relies on snapshot isolation, i.e., no read locks are used.

Counters

This experiment illustrates the benefit of using a system-defined counter as opposed to using an adhoc method based on a counter table. The experiment



Tuning Concurrency Control. Figure 1. Value of serializability experiment on MySQL 6.0.



Tuning Concurrency Control. Figure 2. Counter experiment on MySQL 6.0.

consists in running a fixed number of insert transactions concurrently. The number of threads used to run these transactions is the main parameter of this experiment. The experiment measures response time for the total number of transactions.

Figure 2 presents traces throughput (i.e., total number of transactions/response time) for the ad hoc and system implementation as a function of the number of swap threads. These results were obtained with the configuration used for the Value of Serializability experiment (see above). The SQL code and the data used for this experiment are available at the URL listed below.

In the experiment, the benefits of the system-based counter become more significant as the number of threads increases. The reason is that as the number of threads increase, the counter table becomes a hot spot and the benefits of using a latch released at the end of the statement (system method) over a lock held until the end of the transaction (ad hoc method) becomes significant. Note that the performance of the ad hoc method diminishes as the amount of processing in the transaction increases (i.e., as the time during which the lock is held increases).

URL to Code and Data Sets

Value of Serializability experiment: http://www.databasetuning.org/?sec=valueofserializability

Counter experiment: http://www.databasetuning.org/?sec=counter

Cross-references

- ▶ Concurrency Control
- ▶ Isolation
- ▶ Latching
- ▶ Performance Monitoring Tools
- ▶ Snapshot Isolation
- ▶ Transaction Chopping
- ▶ Two-Phase Locking

Recommended Reading

1. Bernstein P., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Boston, MA, 1987.
2. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1992.
3. Shasha D. and Bonnet P. Database Tuning: Principles, Experiments and Troubleshooting Techniques. Morgan Kaufmann, San Francisco, CA, 2002.
4. Weikum G. and Vossen G. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2001.

Tuning the Application Interface

▶ Application-Level Tuning

Tuple Relational Calculus

▶ Relational Calculus

Tuple-Generating Dependencies

RONALD FAGIN
IBM Almaden Research Center, San Jose, CA, USA

Synonyms

Equality-generating dependency (egd)

Definition

Tuple-generating dependencies, or *tgds*, are one of the two major types of *database dependencies* (the other major type consists of *equality-generating dependencies*, or *egds*).

To define tgds, the notion of an *atomic formula* is first needed, which is a formula of the form $P(x_1,...,x_k)$, where P is a k-ary relational symbol, and $x_1,...,x_k$ are variables, not necessarily distinct.

Then tgds are formulas of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}))$, where

1. $\phi(\mathbf{x})$ is a conjunction of atomic formulas, all with variables among the variables in $\mathbf{x}$.
2. every variable in $\mathbf{x}$ appears in $\phi(\mathbf{x})$ (but not necessarily in $\psi(\mathbf{x}, \mathbf{y})$).
3. $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas, all with variables among the variables in $\mathbf{x}$ and $\mathbf{y}$.

If $\mathbf{y}$ is empty, so that there are no existentially-quantified variables, then the tgd is called *full*.

Conditions (1) and (2) together are sometimes replaced by the weaker condition that $\phi(\mathbf{x})$ be an arbitrary first-order formula with free variables exactly those in $\mathbf{x}$.

Key Points

An example of a tgd is the formula

$$\forall x_1 \forall x_2 (R(x_1, x_1, x_2) \wedge S(x_2) \\ \rightarrow \exists y (R(x_1, y) \wedge T(x_2, y, x_1)).$$

Historically, tgds were introduced for the purpose of database normalization and design, with the first example being *multivalued dependencies* [2,5]. Fagin [3] defined the class of *embedded implicational dependencies*, which includes both tgds and egds, but he focused on the case where they are (i) unirelational (so that all atomic formulas involve the same relation symbol) and (ii) typed (so that no variable can appear in both the ith and jth position of an atomic formula if $i \neq j$). Beeri and Vardi [1] defined and named tgds and egds.

In recent years, tgds have been used to define *schema mappings* in *data exchange* [4], which describe how data structured under one schema (the *source schema*) is to be transformed into data structured under a second schema (the *target schema*). In this application, the atomic formulas in the premise $\phi(\mathbf{x})$ are all from the source schema, and the atomic formulas in the conclusion $\psi(\mathbf{x}, \mathbf{y})$ are all from the target schema.

Cross-references

- ▶ Data Exchange
- ▶ Database Dependencies
- ▶ Equality-Generating Dependencies

- ▶ Join Dependency
- ▶ Multivalued Dependency
- ▶ Normal forms and Normalization
- ▶ Schema Mapping

Recommended Reading

1. Beeri C. and Vardi M.Y. A proof procedure for data dependencies. J. ACM, 31(4):718–741, 1984.
2. Fagin R. Multivalued dependencies and a new normal form for relational databases. ACM Trans. Database Sys., 2(3):262–278, 1977.
3. Fagin R. Horn clauses and database dependencies. J. ACM, 29(4):952–985, 1982.
4. Fagin R., Kolaitis P.G., Miller R.J., and Popa L. Data exchange: semantics and query answering. Theor. Comput. Sci., 2005, pp. 89–124.
5. Zaniolo C. Analysis and Design of Relational Schemata for Database Systems. 1976. Ph.D. Dissertation, UCLA.

Twigs

- ▶ XML Tree Pattern, XML Twig Query

Two-Dimensional Shape Retrieval

Lei Chen
Hong Kong University of Science and Technology,
Kowloon, Hong Kong, China

Definition

Shape is an important image feature, it is the geometrical information of an object after removing position, scale and rotational effects [3]. A shape is often represented by the contour map extracted from the images. Given a query 2D shape, *2D shape retrieval* retrieves a ranked list of similar 2D shapes from a collection of 2D polygonal models (contour points) based on the shape characteristics.

Figure 1 gives an example object shapes, which are represented by the extracted contour maps.

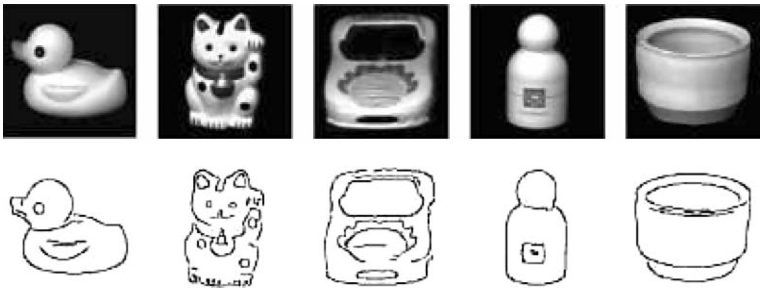
Historical Background

The study of shape retrieval can be traced back to 1980s. At that time, shape retrieval was treated as an key technique in object recognition of robot vision. Since then, shape retrieval has received much attention in the database domain due to its various application in biometrics, industry, medicine and anthropology.

Foundations

The shape retrieval problem is to retrieve shapes that are visually similar to the query shape. There are two key issues related to shape retrieval [6], *shape representation* and *similarity measure*. Given a shape, it should be represented in a form which is invariant to scaling, translation and rotation. For similarity measures, various measures are designed to meet application requirements. In fact, the two key issues are closely related to each other. Based on the different shape representations, different similarity measures are applied. Existing works represent shape in various ways including:

1. *Shape signature* is one-dimensional vector derived from the shape boundary coordinates. For example, the Euclidean distance between centroid point to the boundary points [8]. The shape of an object is represented by a set of normalized Fourier coefficients of the shape signature and the inner product



Two-Dimensional Shape Retrieval. Figure 1. Examples of shape of object represented by contour maps [4].

of the Fourier coefficients is used to measure the similarity between two shapes.

2. *Grid descriptor* is derived by overlaying the shapes with a coarse grid and assigning an '1' to a grid cell if more than 15% of the cell is covered by the shape, otherwise '0' [5]. Two shapes are compared by counting the number of different bits of corresponding normalized grid descriptors. The similarity measure conforms to human similarity perception, i.e., perceptually similar shapes have high similarity measure.
3. *Shape context* captures the distribution of the points relative to a reference point on the contour of an object, thus it offers a globally discriminative feature. Two shapes are similar if they have similar shape contexts. Therefore, the similarity between two shapes is computed by counting the number of matching corresponding points [2].
4. *Distance set* uses N nearest neighbors to represent each contour point [4]. The similarity between two shapes are measure by the cost of the cheapest correspondence relation of corresponding distance sets, which is computed by evaluating the minimum cost assignment in an associated bipartite graph.
5. *Curvature scale space* is formed by the positions of inflection points (x-axis) on the contours on every scale (y-axis) [7]. The shape representation is the positions of the maxima on these curves. The shape similarity is measures by relating the positions of the maxima of the corresponding curves.
6. *Symbolic features* refers to a shape which is represented in terms of multi-interval valued type features including shape centroid, extreme points, axis of least inertia with slope angle, and feature points on the axis of least inertia, etc. [1]. The similarity between two shapes is defined on these symbolic features as the average degree of similarity of all corresponding feature vector pairs.
7. *Shape space* refers a single point (vector) in a high-dimensional manifold. The vector is obtained by normalizing the landmark vector of the origin shape [9]. The similarity is measured as the geodesic distance between a shape and a model in the shape space.

In addition to the shape representation and similarity measure design, index structures should be built on the shape representations to allow shape similarity queries be answered efficiently.

Key Applications

Content Based Image Retrieval

Content-based image retrieval (CBIR), also known as query by image content (QBIC) and content-based visual information retrieval (CBVIR) is the application of computer vision to the image retrieval problem, that is, the problem of searching for digital images in large databases.

Visual Surveillance

Visual surveillance in dynamic scenes is an active research topics in computer vision. The aim of visual surveillance is to make it possible that the computer can watch or monitor a scene by automatic localization, tracking and recognition.

Future Directions

Effective and robust similarity measures and efficient indexing structures.

Cross-references

► Feature-Based 3D Object Retrieval
► Image Retrieval

Recommended Reading

1. Attalla E. and Siy P. Robust shape similarity retrieval based on contour segmentation polygonal multiresolution and elastic matching. Pattern Recognit., 38(12):2229–2241, 2005.
2. Belongie S., Malik J., and Puzicha J. Shape matching and object recognition using shape contexts. IEEE Trans. Pattern Anal. Mach. Intell., 24(4):509–522, 2002.
3. Dryden I.L. and Mardia K.V. Statistical Shape Analysis. Wiley, New York, 1998.
4. Grigorescu C. and Petkov N. Distance sets for shape filters and shape recognition. IEEE Trans. Image Process., 12(10): 1274–1286, 2003.
5. Lu G. and Sajjanhar A. Region-based shape representation and similarity measure suitable for content-based image retrieval. Multimedia Syst., 7(2):165–174, 1999.
6. Mehrotra R. and Gary J.E. Similar-shape retrieval in shape data management. Computer, 28(9):57–62, 1995.
7. Mokhtarian F. and Bober M. Curvature Scale Space Representation: Theory, Applications, and MPEG-7 Standardization. Kluwer, Norwell, MA, USA, 2003.
8. Zhang D. and Lu G. Evaluation of mpeg-7 shape descriptors against other shape descriptors. Multimedia Syst., 9(1):15–30, 2003.
9. Zhang J., Zhang X., Krim H., and Walter G.G. Object representation and recognition in shape spaces. Pattern Recognit., 36(5): 1143–1154, 2003.

Two-Phase Commit

Yousef J. Al-Houmaily[1], George Samaras[2]
[1]Institute of Public Administration, Riyadh, Saudi Arabia
[2]University of Cyprus, Nicosia, Cyprus

Definition

Two-phase commit (2PC) is a synchronization protocol that solves the *atomic commitment problem*, a special case of the *Byzantine Generals* problem. Essentially, it is used in distributed database systems to ensure *global atomicity* of transactions in spite of site and communication failures, assuming that a failure will be, *eventually*, repaired and each site guarantees atomicity of transactions at its local level.

Historical Background

2PC is the simplest and most studied *atomic commit protocol* (ACP). It was first published in [9] and [4]. Since then, the protocol has received much attention from the academia and industry due to its importance in distributed database systems, and the research has resulted in numerous variants and optimizations for different distributed database environments. These environments include main memory databases (e.g., [10]), real-time databases (e.g., [5]), mobile database systems (e.g., [12]), heterogeneous database systems (e.g., [1]), Web databases (e.g., [15]), besides traditional (homogeneous) distributed database systems (e.g., [13,3]).

Foundations

In a distributed database system, a transaction is decomposed into a set of *subtransactions*, each of which executes at a single participating database site. Assuming that each database site preserves atomicity of (sub)transactions at its local level, global atomicity cannot be guaranteed without taking additional measures. This is because without global synchronization a distributed transaction might end-up committing at some participating sites and aborting at others due to a site or a communication failure. Thus, jeopardizing *global* atomicity and, consequently, the consistency of the (distributed) database.

To achieve atomicity at the global level, there is a need for a synchronization protocol that ensures a unanimous final outcome for each distributed transaction and regardless of failures. Such a protocol is referred to as an *atomic commit protocol* (ACP). An ACP ensures that a distributed transaction is either *committed* and all its effects become persistent across all participating sites, or *aborted* and all its effects are obliterated as if the transaction had never executed at any site. This is the essence of the two-phase commit (2PC) protocol.

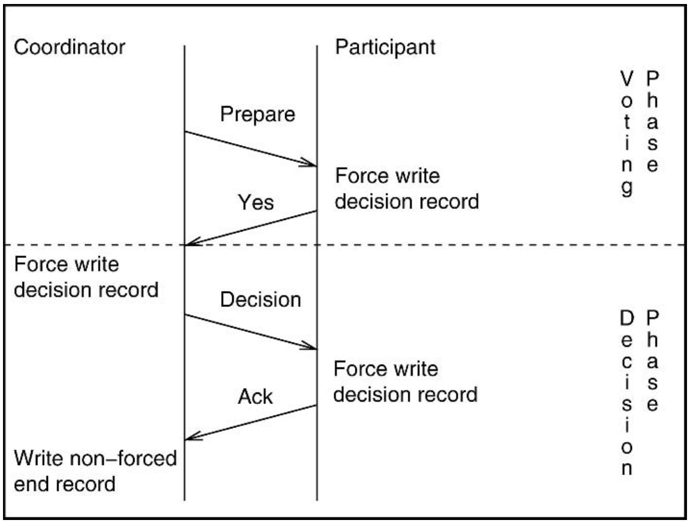
Dynamics of Two-Phase Commit

In 2PC, each transaction is associated with a designated site called the *coordinator* (or *master*). Although the coordinator of a transaction could be any of the sites participating in the transaction's execution, it is commonly the originating site of the transaction (i.e., the site where the transaction is first initiated). The rest of the sites are called *participants*, *subordinates*, *cohorts* or *slaves*. Once a transaction finishes its execution and indicates its termination point, through a commit primitive, to its coordinator, the coordinator initiates 2PC.

As the name implies, 2PC consists of two phases, namely a *voting phase* and a *decision phase*, as shown Fig. 1. During the voting phase, the coordinator requests all the sites participating in the transaction's execution to *prepare-to-commit* whereas, during the decision phase, the coordinator either decides to commit the transaction if *all* the participants are prepared to commit (voted "yes"), or to abort if any participant has decided to abort (voted "no"). On a commit decision, the coordinator sends out commit messages to *all* participants whereas, on an abort decision, it sends out abort messages to *only* those (required) participants that are prepared-to-commit (voted "yes").

When a participant receives a prepare-to-commit message for a transaction, it validates the transaction with respect to data consistency. If the transaction can be committed (i.e., it passed the validation process), the participant responds with a "yes" vote. Otherwise, it responds with a "no" vote and aborts the transaction, releasing all the resources held by the transaction.

If a participant had voted "yes", it can neither commit nor abort the transaction unilaterally and has to wait until it receives a final decision from the coordinator. In this case, the participant is said to be *blocked* for an indefinite period of time called *window of uncertainty* (or *window of vulnerability*) awaiting the coordinator's decision. When a participant receives the final decision, it complies with the decision, sends back an acknowledgement message (Ack) to the coordinator and releases all the resources held by the transaction.


Coordinator
Participant
Prepare
Force write decision record
Yes
Voting Phase
Force write decision record
Decision
Force write decision record
Ack
Decision Phase
Write non–forced end record

Two-Phase Commit. Figure 1. The two-phase commit protocol.

When the coordinator receives Acks from all the participants that had voted "yes," it *forgets* the transaction by discarding all information pertaining to the transaction from its protocol table that is kept in main memory.

The resilience of 2PC to failures is achieved by recording the progress of the protocol in the logs of the coordinator and the participants. The coordinator force writes a *decision* record prior to sending out its decision to the participants. Since a *forced write* of a log record causes a flush of the log onto a stable storage that survives system failures, the decision is not lost if the coordinator fails. Similarly, each participant force writes a *prepared* record before sending its "yes" vote and a *decision* record before acknowledging a decision. When the coordinator completes the protocol, it writes a non-forced *end* record in the volatile portion of its log that is kept in main memory. This record indicates that all (required) participants have received the decision and none of them will inquire about the transaction's status in the future. This allows the coordinator to (permanently) forget the transaction, with respect to 2PC, and garbage collect the log records of the transaction when necessary.

Recovery in Two-Phase Commit

Site and communication failures are detected by *timeouts*. When an operational site detects a failure, it invokes a recovery manager to handle the failure. In 2PC, there are four places where a communication failure might occur. The first place is when a participant is waiting for a prepare-to-commit message from the coordinator. This occurs before the participant has voted. In this case, the participant may unilaterally decide to abort the transaction. The second place is when the coordinator is waiting for the votes of the participants. Since the coordinator has not made a final decision yet and no participant could have decided to commit, the coordinator can decide to abort. The third place is when a participant had voted "yes" but has not received a commit or an abort final decision. In this case, the participant cannot make any unilateral decision because it is uncertain about the coordinator's final decision. The participant, in this case, is *blocked* until it re-establishes communication with the coordinator and, once re-established, the participant inquires the coordinator about the final decision and resumes the protocol by enforcing and, then, acknowledging the coordinator's decision. The fourth place is when the coordinator is waiting for the Acks of the participants. In this case, the coordinator re-submits its final decision to those participants that have not acknowledged the decision once it re-establishes communication with them. Notice that the coordinator cannot simply discard the information pertaining to a transaction from its protocol table or its stable log until it receives Acks from all the (required) participants.

To recover from site failures, there are two cases to consider: coordinator's failure and participant's failure.

For a coordinator's failure, the coordinator, upon its restart, scans its stable log and re-builds its protocol table to reflect the progress of 2PC for all the pending transactions prior to the failure. The coordinator has to consider only those transactions that have started 2PC and have not finished it prior to the failure (i.e., transactions that have decision log records without corresponding end log records in the stable log). For other transactions, i.e., transactions that were active at the coordinator's site prior to its failure without a decision record, the coordinator considers them as aborted transactions. Once the coordinator re-builds its protocol table, it completes the protocol for each of these transactions by re-submitting its final decision to all (required) participants whose identities are recorded in the decision record and waiting for their Acks. Since some of the participants might have already received the decision prior to the failure and enforced it, these participants might have already forgotten that the transaction had ever existed. Such participants simply reply with *blind* Acks, indicating that they have already received and enforced the decision prior to the failure.

For a participant's failure, the participant, as part of its recovery procedure, checks its log for the existence of any transaction that is in a prepared-to-commit state (i.e., has a prepared log record without a corresponding final decision one). For each such transaction, the participant inquires the transaction's coordinator about the final decision. Once the participant receives the decision from the coordinator, it completes the protocol by enforcing and, then, acknowledging the decision. Notice that a coordinator will be always able to respond to such inquires because it cannot forget a transaction before it has received the Acks of all (required) participants. However, there is a case where a participant might be in a prepared-to-commit state and the coordinator does not remember the transaction. This occurs if the coordinator fails after it has sent prepare-to-commit messages and just before it has made its decision. In this case, the coordinator will not remember the transaction after it has recovered. If a prepared-to-commit participant inquires about the transaction's status, the coordinator will *presume* that the transaction was aborted and responds with an abort message. This special case where an abort presumption is made about unremembered transactions in 2PC motivated the design of the *presumed abort* 2PC.

Underlying Assumptions

ACPs solve a special case of the problem of consensus in the presence of faults, a problem that is known in distributed systems as the *Byzantine Generals* problem [7]. This problem is, in its most general case, not solvable without some simplifying assumptions. In distributed database systems, ACPs solve the problem under the following general assumptions (among others that are sometimes ACP specific):

1. *Each site is sane*: A site is fail stop where it never deviates from its prescribed protocol. That is, a site is either operational or not but never behaves abnormally causing *commission* failures.
2. *Eventual recovery*: A failure (whether site or communication) will be, eventually, repaired.
3. *Binary outcome*: All sites unanimously agree on a single binary outcome, either commit or abort.

Performance Issues

There are three important performance issues that are associated with ACPs, which are as follows [3]:

1. *Efficiency During Normal Processing*: This refers to the cost of an ACP to provide atomicity in the absence of failures. Traditionally, this is measured using three metrics. The first metric is *message complexity* which deals with the number of messages that are needed to be exchanged between the systems participating in the execution of a transaction to reach a consistent decision regarding the final status of the transaction. The second metric is *log complexity* which accounts for the frequency at which information needs to be recorded at each participating site in order to achieve resiliency to failures. Typically, log complexity is expressed in terms of the required number of non-forced log records which are written into the log buffer (in main memory) and, more importantly, the number of forced log records which are written onto the stable log (on the disk). The third metric is *time complexity* which corresponds to the required number of rounds or sequential exchanges of messages in order for a decision to be made and propagated to the participants.
2. *Resilience to Failures*: This refers to the types of failures that an ACP can tolerate and the effects of failures on operational sites. An ACP is considered *non-blocking* if it never requires operational sites to

wait (i.e., block) until a failed site has recovered. One such protocol is 3PC.

3. *Independent Recovery*: This refers to the speed of recovery. That is, the time required for a site to recover its database and become operational, accepting new transactions after a system crash. A site can *independently* recover if it has all the necessary information needed for recovery stored locally (in its log) without requiring any communication with any other site in order to fully recover.

Most Common Two-Phase Commit Variants

Due to the costs associated with 2PC during normal transaction processing and the reliability drawbacks in the events of failures, a variety of ACPs have been proposed in the literature. These proposals can be, generally, classified as to enhance either (i) the efficiency of 2PC during normal processing or (ii) the reliability of 2PC by either reducing 2PC's blocking aspects or enhancing the degree of independent recovery. The most commonly pronounced 2PC variants are *presumed abort* (PrA) [11] and *presumed commit* (PrC) [11]. Both variants reduce the cost of 2PC during normal transaction processing, albeit for different final decisions. That is, PrA is designed to reduce the costs associated with aborting transactions whereas PrC is designed to reduce the costs associated with committing transactions.

In PrA, when a coordinator decides to abort a transaction, it does not force-write the abort decision in its log as in 2PC. It just sends abort messages to all the participants that have voted "yes" and discards all information about the transaction from its protocol table. That is, the coordinator of an aborted transaction does not have to write any log records or wait for Acks. Since the participants do not have to Ack abort decisions, they are also not required to force-write such decisions. After a coordinator's or a participant's failure, if the participant *inquires* about a transaction that has been aborted, the coordinator, not remembering the transaction, will direct the participant to abort the transaction (by presumption). Thus, as the name implies, if no information is found in the log of the coordinator of a transaction, the transaction is presumed aborted.

As opposed to PrA, in which missing information about transactions at a coordinator's site is interpreted as abort decisions, in PrC, a coordinator interprets missing information about transactions as commit decisions when replying to inquiry messages. However, in PrC, a coordinator has to force write a commit *initiation* record for each transaction before sending out prepare-to-commit messages to the participants. This record ensures that missing information about a transaction will not be misinterpreted as a commit after a coordinator's site failure without an actual commit decision is made.

To commit a transaction, the coordinator force writes a commit record to logically eliminate the initiation record of the transaction and then sends out commit messages. The coordinator also discards all information pertaining to the transaction from its protocol table. When a participant receives the decision, it writes a non-forced commit record and commits the transaction without having to Ack the decision. After a coordinator's or a participant's failure, if the participant *inquires* about a transaction that has been committed, the coordinator, not remembering the transaction, will direct the participant to commit the transaction (by presumption).

To abort a transaction, on the other hand, the coordinator does not write the abort decision in its log. Instead, the coordinator sends out abort messages and waits for Acks before discarding all information pertaining to the transaction. When a participant receives the decision, it force writes an abort record and then acknowledges the decision, as in 2PC. In the case of a coordinator's failure, the initiation record of an interrupted transaction contains all needed information for its recovery.

Table 1 summarizes the costs associated with the three 2PC variants for the commit as well as the abort case assuming a "yes" vote from each participant: "*m*" is the total number of log records, "*n*" is the number of forced log writes, "*p*" is the number of messages sent from the coordinator to each participant and "*q*" is the number of messages sent back to the coordinator. For simplicity, these costs are calculated for the flat (two-level) execution model in which, unlike the multi-level execution model, a participant never initiates (i.e., spawns) new (sub)transactions that execute at other participants, forming a tree of communicating participants.

Compatibility of 2PC Variants

ACPs are incompatible in the sense that they cannot be used (directly) in the same environment without conflicts. This is true even for the simplest and most

Two-Phase Commit. Table 1. The costs for update transactions in 2PC and its most commonly known two variants

	Commit decision						Abort decision					
	Coordinator			Participant			Coordinator			Participant		
2PC Variant	m	n	p	m	n	q	m	n	p	m	n	q
Basic 2PC	2	1	2	2	2	2	2	1	2	2	2	2
Presumed abort	2	1	2	2	2	2	0	0	2	2	1	1
Presumed commit	2	2	2	2	1	1	2	1	2	2	2	2

closely related variants such as the basic 2PC, PrA and PrC. The analysis of ACPs shows that incompatibilities among ACPs could be due to (i) the semantics of the coordination messages (which include both their meanings as well as their existence), or (ii) the presumptions about the outcome of terminated transactions in case of failures [1].

The *presumed any* (PrAny) protocol [2] interoperates the basic 2PC, PrA, and PrC. It was proposed in the context of multidatabase systems, a special case of heterogeneous distributed databases, to demonstrate the difficulties that arise when one attempts to interoperate different ACPs in the same environment and, more importantly, to introduce the "*operational correctness criterion*" and the notion of "*safe state*." Operational correctness means that all sites should be able, not only to reach an agreement but also, to forget the outcome of terminated transactions. On the other hand, the safe state means that, for any operationally correct ACP, the coordinator should be able to reach a state in which it can reply to the inquiry messages of the participants, in a consistent manner, without having to remember the outcome of terminated transactions forever.

In PrAny, a coordinator talks the language of the three 2PC variants and knows which variant is used by which participant. Based on that, it forgets a committed transaction once all PrA and 2PC participants Ack the commit decision, and forgets an aborted transaction once all PrC and 2PC participants Ack the abort decision. This is because only commit decisions are acknowledged in PrA whereas, in PrC, only abort decisions are acknowledged. However, unlike the other 2PC variants, in PrAny, a coordinator does not adopt a single presumption about the outcome of *all* terminated transactions. This is because, if it does so, the global atomicity of some transactions might be violated. For example, if the coordinator adopts for recovering purposes the abort presumption, it will respond with an abort message to a recovering PrC participant that inquires about a forgotten committed transaction. Similarly, if the coordinator adopts the commit presumption, it will respond with a commit message to a recovering PrA participant that inquires about a forgotten aborted transaction. Instead of using a single presumption, a coordinator in PrAny adopts the presumption of the protocol used by the inquiring participant. That is, if a participant inquires about a forgotten committed transaction, the participant has to be a PrC participant. This is because only PrC participants do not acknowledge commit decisions. Thus, the coordinator will reply with a commit message in accordance with PrC adopted by the participant. On the other hand, if a participant inquires about a forgotten aborted transaction, the participant has to be a PrA participant. This is because only PrA participants do not acknowledge abort decisions. Thus, the coordinator will reply with an abort message in accordance with PrA adopted by the participant. Knowledge about the used protocols by the participants could be recorded statically at the coordinator's site [2] or inferred dynamically by having a participant declares its used protocol in each inquiry message [15].

Key Applications

The use of 2PC (or one of its variants) is mandatory in any distributed database system in which the traditional atomicity property of transactions is to be preserved. However, the basic 2PC has never been implemented in any commercial database system due to its unnecessary costs compared to its two other most commonly known variants. Instead, PrA is considered the *de facto* standard in the industry and has been incorporated as part of the current X/Open DTP [14] and ISO OSI-TP [6] distributed transaction processing standards. PrA is chosen instead of PrC because (i) the cost of PrC for committing transactions is not symmetric with the cost of PrA for aborting transactions (which is highlighted in Table 1) and, more importantly, (ii) PrA is much cheaper to use with read-only transactions when complementing it with the traditional *read-only* optimization [13,3] (which is also part of the current database standards).

The above two reasons that favor PrA have been nullified with new 2PC variants and a read-only optimization called *unsolicited update-vote* (UUV). Thus, PrC is expected to become also part of future database standards, especially that the two variants can be incorporated in the same environment without any conflicts [1,15].

Cross-references

▶ Atomicity
▶ Distributed Database Systems
▶ Distributed Recovery
▶ Distributed Transaction Management

Recommended Reading

1. Al-Houmaily Y. Incompatibility dimensions and integration of atomic commit protocols. Int. Arab J. Inf. Technol., 5(4):2008.
2. Al-Houmaily Y. and Chrysanthis P. Atomicity with incompatible presumptions. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 306–315.
3. Chrysanthis P.K., Samaras G., and Al-Houmaily Y. Recovery and performance of atomic commit processing in distributed database systems, In Recovery Mechanisms in Database Systems, V. Kumar, M. Hsu (eds.). Prentice Hall, Uppersaddle River, NJ, 1998, pp. 370–416.
4. Gray J.N. Notes on data base operating systems. In Operating Systems – An Advanced Course. M.J. Flynn et al. (eds.), LNCS, Vol. 60, Springer, London, 1978, pp. 393–481.
5. Haritsa J., Ramamritham K., and Gupta R. The PROMPT real-time commit protocol. IEEE Trans. Parallel Distributed Syst., 11(2):160–181, 2000.
6. ISO. Open systems interconnection – Distributed transaction processing – Part 1: OSI TP Model. ISO/IEC, 10026–1, 1998.
7. Lamport L., Shostak R., and Pease M. The Byzantine generals problem. ACM Trans. Programming Lang. Syst., 4(3):382–401, 1982.
8. Lampson B. and Lomet D. A new presumed commit optimization for two phase commit. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 630–640.
9. Lampson B. and Sturgis H. Crash recovery in a distributed data storage system. Technical report, Computer Science Laboratory, Xerox Palo Alto Research Center, CA, 1976.
10. Lee I. and Yeom H. A single phase distributed commit protocol for main memory database systems. In Proc. 16th Int. Parallel and Distributed Processing Symp., 2002, pp. 14–21.
11. Mohan C., Lindsay B., and Obermarck R. Transaction management in the R* distributed data base management system. ACM Trans. Database Syst., 11(4):378–396, 1986.
12. Nouali N., Drias H., and Doucet A. A mobility-aware two-phase commit protocol. Int. Arab J. Inf. Technol., 3(1):2006.
13. Samaras G., Britton K., Citron A., and Mohan C. Two-phase commit optimizations in a commercial distributed environment. Distrib. Parall. Databases, 3(4):325–361, 1995.
14. X/Open Company Limited. Distributed Transaction Processing: Reference Model. Version 3 (X/Open Document No. 504), 1996.
15. Yu W. and Pu C. A Dynamic Two-phase commit protocol for adaptive composite services. Int. J. Web Serv. Res., 4(1):2007.

Two-Phase Commit Protocol

Jens Lechtenbörger
University of Münster, Münster, Germany

Synonyms

XA standard

Definition

The *Two-phase commit (2PC)* protocol is a distributed algorithm to ensure the consistent termination of a transaction in a distributed environment. Thus, via 2PC a unanimous decision is reached and enforced among multiple participating servers whether to commit or abort a given transaction, thereby guaranteeing atomicity. The protocol proceeds in two phases, namely the prepare (or voting) and the commit (or decision) phase, which explains the protocol's name.

The protocol is executed by a *coordinator* process, while the participating servers are called *participants*. When the transaction's initiator issues a request to commit the transaction, the coordinator starts the first phase of the 2PC protocol by querying – *via prepare messages* – all participants whether to abort or to commit the transaction. If all participants vote to commit then in the second phase the coordinator informs all participants to commit their share of the transaction by sending a *commit message*. Otherwise, the coordinator instructs all participants to abort their share of the transaction by sending an *abort message*. Appropriate log entries are written by coordinator as well as participants to enable restart procedures in case of failures.

Historical Background

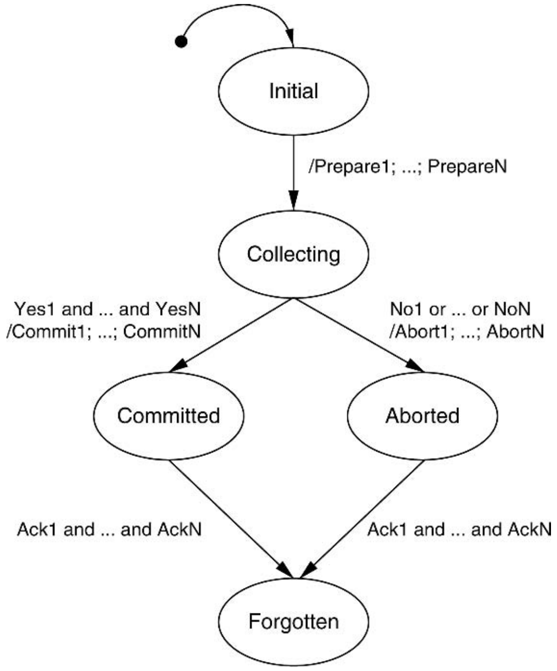
Essentially, the 2PC protocol is modeled after general contract law, where a contract among two or more parties is only established if all parties agree; hence, the underlying idea is well-established in everyday life. According to [3] the first known implementation in a distributed system was performed by Nico Garzado for the Italian social security system in the early 1970s, while the protocol's name arose in the mid 1970s. Early scientific presentations are given by Gray [2] and by Lampson and Sturgis [4]. Since then an API for the 2PC protocol has been standardized under the name XA within the X/Open Distributed Transaction Processing (DTP) model [7], and this API has been incorporated into several middleware specifications and implemented in numerous software components.

Foundations

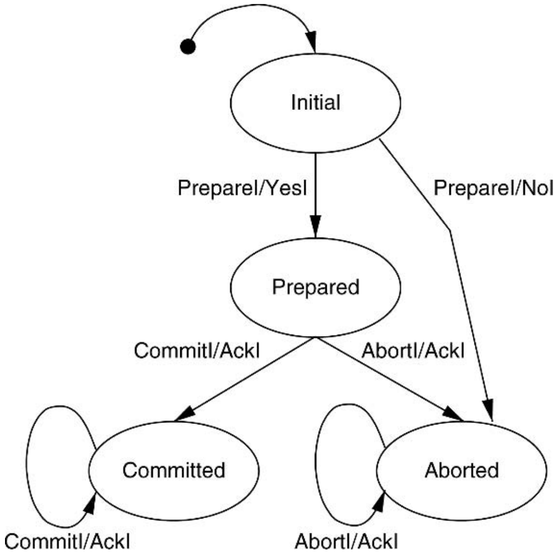
The 2PC protocol as described and analyzed in detail in [8] assumes that parts of a single (distributed) transaction involve resources hosted by multiple *resource managers* (e.g., database systems, file systems, messaging systems, persistent programming environments), which reside on possibly different nodes of a network and are called *participants* of the protocol. For every transaction one *coordinator* process, typically running on the node of that participant where the transaction was initiated, assumes responsibility for executing the 2PC protocol; alternative strategies for selecting (and transferring) the coordinator are discussed in [8]. The states through which coordinator and participants move in the course of the protocol are illustrated in Figs. 1 and 2, resp., and explained in the following. Such statecharts represent finite state automata, where ovals denote states, labeled arcs denote state transactions, and arc labels of the form "precondition/action" indicate that (i) the state transition is only enabled if the precondition is satisfied and (ii) the given action is executed when the state is changed.

Basic Protocol

As long as a transaction is still executing ordinary operations, coordinator as well as all participants operate in the Initial state. When the coordinator is requested to commit the transaction, it initiates the first phase of the 2PC protocol: To capture the state of the protocol's execution (which needs to be available in case of protocol restarts as explained below), the coordinator first forces a *begin log entry*, which includes a transaction identifier as well as a list of the transaction's participants, to a stable log. Afterwards, the coordinator sends a *prepare* message to every participant, enters the Collecting state and waits for replies.



Two-Phase Commit Protocol. Figure 1. Statechart for coordinator (given *N* participants).



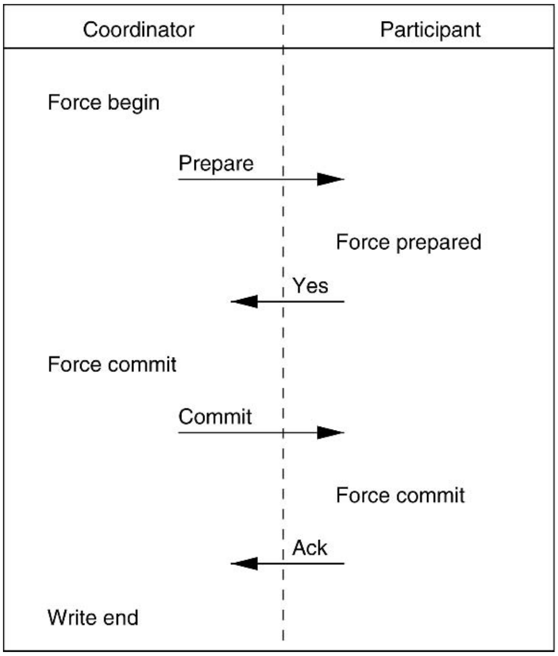
Two-Phase Commit Protocol. Figure 2. Statechart for participant *I*.

Upon receiving a prepare message, a participant decides whether it is able to commit its share of the transaction. In either case, suitable log entries for later recovery operations as well as a *prepared log entry* indicating the vote ("Yes" or "No") are forced to a stable log, before a response message containing the vote is sent back to the coordinator. In case of a No-vote, the participant switches into the Aborted state and immediately aborts the transaction locally. In case of a Yes-vote, the participant moves into the Prepared state. In the latter case the participant is said to be *in doubt* or *blocked* as it has now given up its local autonomy and must await the final decision from the coordinator in the second phase (in particular, locks cannot be released yet).

Once the coordinator has received all participants' response messages it starts the second phase of the 2PC protocol and decides how to complete the global transaction: The result is "Commit" if all participants voted to commit and "Abort" otherwise. The coordinator then forces a *commit* or *abort log entry* to the stable log, sends a message containing the final decision to all participants, and enters the corresponding state (Committed or Aborted).

Upon receipt of the decision message, a participant commits or aborts the local changes of the transaction depending on the coordinator's decision and forces suitable log entries for later recovery as well as a *commit* or *abort log entry* to a stable log. Afterwards, it sends an acknowledgment message to the coordinator and enters the corresponding final state (Committed or Aborted).

Once the coordinator has received all acknowledgment messages it ends the protocol by writing an *end log entry* to a stable log to enable later log truncation and enters the final state, Forgotten. The actions described for the overall process are summarized in Fig. 3 for the case of a transaction commit. (For multiple participants, the actions simply have to be duplicated; in case of abort, at least one of the participants votes "No", which implies that all occurrences of "commit" are replaced with "abort".)



Two-Phase Commit Protocol. Figure 3. Actions for transaction commit in the basic protocol.

Protocol Restart

The log entries seen so far are used to restart the 2PC protocol after so-called soft crashes of coordinators or participants, i.e., failures like process crashes which lead to a loss of main memory but which leave secondary storage intact. In particular, as participants always force log entries before sending replies, the coordinator never needs to resend messages for which replies have been received. Moreover, log truncation (garbage collection) may occur once all acknowledgment messages have arrived. Finally, every log entry uniquely determines a state, and the last log entry determines the most recent state prior to a failure. Clearly, failures in the final states (Forgotten for the coordinator and Committed or Aborted for a participant) do not require any action. For the remaining states, restart procedures are as follows:

If the coordinator fails in the Initial or the Collecting state, it simply restarts the protocol in the Initial state. (Coordinators writing received votes into the log could recover differently from the Collecting state.) If it fails in the Committed or in the Aborted state, it resends the decision message to all participants, and continues waiting for acknowledgments in the previous state.

If a participant fails in the Initial state it did not yet participate in the 2PC protocol and is free to decide arbitrarily when asked later on. If it fails in the Prepared state it either waits for the coordinator to announce the decision or actively queries the coordinator or other participants for the decision.

In addition to these restart procedures, coordinator and participants also need to be able to recover from message losses. To this end, standard timeout mechanisms are employed: Whenever a message is sent, a timer starts to run. If the timer expires before an appropriate answer is received, the message is simply resent (assuming that either original message or answer are lost; e.g., if the coordinator is missing some votes in the Collecting state, it resends a prepare message to every participant that did not answer in time). Finally, if repeated timeouts occur in the Collecting state the coordinator may decide to abort the transaction globally (as if an "Abort" vote was received), and a participant may unilaterally abort the transaction in the Initial state if no prepare message arrives.

Hierarchical and Flattened 2PC

New participants enter the 2PC protocol whenever they receive requests (e.g., to execute SQL statements) from already existing participants. In such a situation, the new participant can be regarded as child node of the requesting participant, and all such parent-child relationships form a *participant tree* with the transaction's initiator as root node. To execute the 2PC protocol, that tree may either be used directly or flattened as explained in the following.

For the *flattened* 2PC, one node in the participant tree, e.g., the root node, is chosen as coordinator, and this coordinator communicates directly with every participant contained in the tree to execute the basic 2PC protocol as described above. In contrary, in case of the hierarchical 2PC, the root node acts as global coordinator, the leaf nodes are ordinary participants, and the inner nodes are participants with respect to their parents as well as sub-coordinators for their children. Thus, when an inner node receives a 2PC message from its parent, the inner node first has to forward the message to its children before it responds on behalf of the *entire* subtree. For example, a prepare message is forwarded down the tree recursively, and an inner node first waits for all votes of its children before it decides, write a log entry, responds with a vote to the parent, and makes a transition to the Prepared (if all children voted to commit) or Aborted state.

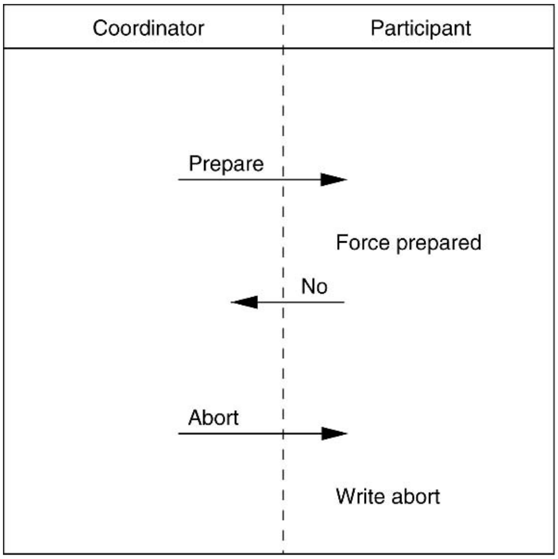
Optimizations

As the 2PC protocol involves costly operations such as sending messages and forcing log entries, several optimizations of the basic protocol have been proposed. In the following the most common variants based on *presumption* are sketched; further details and techniques such as real-only subtree optimization, coordinator transfer, and three-phase commit (3PC) to reduce blocking are presented in [8].

The key idea for presumption based optimizations is to write less log entries and send fewer messages in a systematic way such that in case of a failure the missing information can be compensated for by suitable presumptions concerning the transaction's state. As the basic protocol described above is not based on any presumptions, it is also called *presumed-nothing protocol*. In contrast, in the *presumed-abort protocol*, which aims to optimize the case of aborted transactions, the essential idea is to omit certain information concerning transaction aborts. If that information is needed but absent later on, abort is presumed. In fact, for the presumed-abort protocol the following information is omitted:

- The Coordinator's begin and abort log entries are omitted.
- The participants' abort log entries are not forced.
- Participants do not send acknowledgment messages before entering the Aborted state.

The actions required in case of a transaction abort are summarized in Fig. 4, which indicates significant



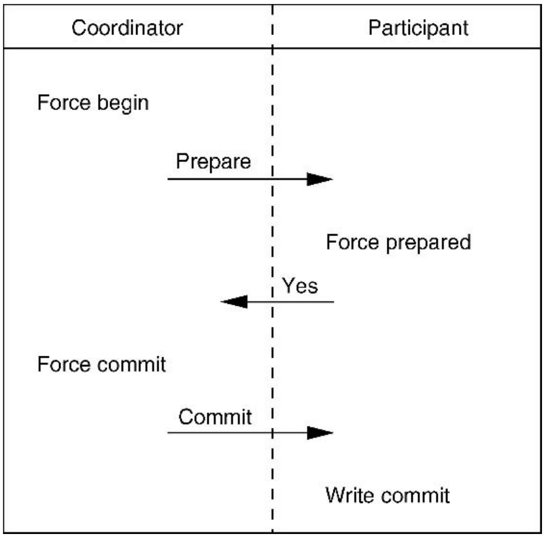
Two-Phase Commit Protocol. Figure 4. Actions for transaction abort in the presumed-abort variant.

savings when compared with the actions for the basic protocol shown in Fig. 3. In the presumed-abort variant, if a participant fails after receiving the abort decision from the coordinator and restarts without finding a log entry, it queries the coordinator for the decision. As the coordinator does not find the appropriate log entry (which has never been written) it presumes that the transaction should be aborted and informs the participant accordingly, which leads to a globally consistent decision.

Alternatively, in the *presumed-commit protocol*, which aims to optimize the case of committed transactions, the following information is omitted:

- The participants' commit log entries are not forced.
- Participants do not send acknowledgment messages before entering the Committed state.

The actions required in case of a transaction commit are summarized in Fig. 5, which again indicates significant savings in comparison to the basic protocol shown in Fig. 3. In this variant, log entries of committed transactions can be garbage collected as missing transactions are presumed to have committed. Thus, if a participant fails after receiving the commit decision from the coordinator and restarts without finding a log entry, it queries the coordinator for the decision. If the coordinator does not find any log entry it presumes that the transaction has committed and informs the participant accordingly, which leads to a globally consistent decision.



Two-Phase Commit Protocol. Figure 5. Actions for transaction commit in the presumed-commit variant.

Key Applications

While there is no single key application for the 2PC protocol, it is applicable wherever decentralized data needs to be shared by multiple participants under transactional guarantees, e.g., in e-commerce or e-science settings. More specifically, the 2PC protocol is widely implemented in database systems (commercial as well as open source ones), TP monitors, and message queue systems, where it is used in the background to provide atomicity for distributed transactions. In addition, the XA interface [7] for the protocol, more precisely for the hierarchical presumed-abort variant, has been adopted in the CORBA Transaction Service specified by the OMG [5] and is used as basis for the Java Transaction API (JTA) [6]. Furthermore, the 2PC protocol is also part of the Web Services Atomic Transaction specification [1] to enable the interoperable atomic composition of Web Service invocations.

Cross-references

▶ ACID Properties
▶ Distributed Transaction Management
▶ Logging and Recovery
▶ Transaction
▶ Transaction Management

Recommended Reading

1. Cabrera L.F. et al. Web services atomic transaction, 2005.
2. Gray J. Notes on database operating systems. In Operating Systems: An Advanced Course. Lecture Notes in Computer Science. R. Bayer, M.R. Graham, G. Seegmüller (eds.). 60, Springer, Berlin Heidelberg New York, 1978, pp. 393–481.
3. Gray J. and Reuter A. Transaction processing: concepts and techniques. Morgan Kaufmann, San Francisco, CA, 1993.
4. Lampson B.W. and Sturgis H. Crash recovery in distributed data storage systems. Technical Report, Xerox Palo Alto Research Center, Palo Alto, CA.
5. OMG Transaction Service, version 1.4. http://www.omg.org/technology/documents/formal/transaction_service.htm, 2007.
6. Sun Microsystems. Java Transaction API (JTA). http://java.sun.com/jta/; http://java.sun.com/jta/, 2007.
7. The Open GROUP Distributed Transaction Processing: The XA Specification. X/Open Company Ltd, ISBN 1 872630 24 3, 1991.
8. Weikum G. and Vossen G. Transactional information systems – theory, algorithms, and the practice of concurrency control and recovery. Morgan Kaufmann, San Francisco, CA, 2002.

Two-Phase Locking

Georg Lausen
University of Freiburg, Freiburg, Germany

Synonyms

Locking protocol; Isolation; Conflict serializability; Pessimistic scheduler

Definition

A locked transaction is a transaction which, in addition to read and write actions, contains lock and unlock operations to the data items. Lock and unlock operations enable a database system to control the order of read and write actions of a concurrent set of transactions. A locking policy is a set of rules which restrict the possible ways to introduce lock and unlock operations into a transaction. A locking policy is safe, if, whenever all the transactions conform to the policy, any history of the transactions is guaranteed to be serializable. Two-Phase Locking is a safe locking policy which is based on the simple rule saying a transaction is not allowed to further lock a data item once it has already unlocked some data item.

Historical Background

Two-Phase Locking was first described in [6]. Later the basic policy has been extended into several directions. In [1] ordered sharing of locks is proposed which allows more than one transaction to hold a lock on a data item as long as the actions are performed in the same order as the locks have been acquired. Altruistic locking has been proposed by [10]. A transaction may allow other transactions to access data items it has locked if it will not access these items later again. This protocol is designated to situations in which long and short transactions are running concurrently. Two-Phase Locking taking old values of a data item into account is described in [2]. Also hybrid protocols have been proposed which allow to apply other techniques than Two-Phase Locking simultaneously [5,8].

Foundations

Locking protocols are among the earliest mechanisms which have been developed for controlling a set of transactions to achieve serializability. The reason is not surprising, as locking is a very intuitive means to control transactions: the processing of an action may be either immediately allowed or delayed. Using locking, the critical actions for serializability are all those, whose order of processing might influence the effects of the transactions and thereby affect serializability. Such actions are called conflicting. Read and write actions are conflicting, whenever they refer to the same data item and at least one of them is a write action. The notion of serializability which is applicable under these assumptions is called conflict-serializability.

For the following, a database is a finite set $D = \{x, y, z, ...\}$ of disjoint data items. A transaction $t = (op_t, <_t)$ consists of a set of steps op_t which are assumed to be totally ordered by $<_t$. Read actions $r(x)$ and write actions $w(x)$ are steps, where x is the data item on which the respective action is processed. A history s of a set T of concurrent transactions is a pair $s = (op_s, <_s)$, where $op_s = \cup_{t \in T} op_t$ and $<_s$ is a total order on the steps in op_s which preserves $<_t$ for $t \in T$. $<_s$ also is called an interleaving of the transactions in T. For notational simplicity, total orders $<_t$ and $<_s$, $t \in T$, s a history of T, are also written as action sequences, where the total order is defined by considering a sequence from left to right. A history is called serializable, if it is equivalent to a serial, i.e., not interleaved history of the same set of transactions.

The decision whether or not a history is serializable can be based on an analysis of a so called conflict graph whose definition is based on the relative order of conflicting actions. Let $s = (op_s, <_s)$ be a history of a set of transactions T. The conflict graph $C(s)$ is a directed graph with set of nodes T and edges $t_i \rightarrow t_j$, $i \neq j$, whenever one of the following conditions is fulfilled:

$$\begin{array}{ll}
\textbf{RW}-\textbf{conflict:} & r_i(x) \in op_i, w_j(x) \in op_j, r_i(x) <_s w_j(x) \\
 & \text{and for all } w_k(x) \in op_s, i \neq k, j \neq k, \\
 & \text{there either holds } w_k(x) <_s r_i(x) \\
 & \text{or } w_j(x) <_s w_k(x), \\
\textbf{WR}-\textbf{conflict:} & w_i(x) \in op_i, r_j(x) \in op_j, w_i(x) <_s r_j(x) \\
 & \text{and for all } w_k(\mathrm{x}) \in op_s, i \neq k, j \neq k, \\
 & \text{there either holds } w_k(x) <_s w_i(x) \\
 & \text{or } r_{\mathrm{j}}(x) <_s w_k(x), \\
\textbf{WW}-\textbf{conflict:} & w_i(x) \in op_i, w_j(x) \in op_j, w_i(x) <_s w_j(x) \\
 & \text{and for all } w_k(x) \in op_s, i \neq k, j \neq k, \\
 & \text{there either holds } w_k(x) <_s w_i(x) \\
 & \text{or } w_{\mathrm{j}}(x) <_s w_k(x).
\end{array}$$

It is well known (e.g., [9,12]), whenever the conflict graph of a history is acyclic, then the history is serializable. However, this condition is not necessary, there may exist histories whose conflict graph is cyclic and still these histories are serializable. In practice, such histories are considered to be not of interest and, as a consequence, as correctness notion conflict-serializability (CSR) is used:

CSR : A history is conflict – serializable if and
only if its conflict graph is acyclic.

To keep the terminology simple, instead of conflict-serializability the notion of serializability will be used in the sequel. The serializability of a history $s = (op_s, <_s)$ of a set of transactions T depends on the relative ordering of the conflicting actions of the involved transactions. Consider two transactions t, t'. Let t, among others, contain actions p, q and t' actions p', q'. Now assume that p and p' are in conflict and q and q' as well. Let further $p <_s p'$ and $q' <_s q$. s is not serializable as the conflict graph $C(s)$ contains the cycle $t \rightarrow t' \rightarrow t$. This means that there cannot exist a serial schedule s^* containing t and t' with the same order on the conflicting actions. For such a schedule s^*, either $p <_{s^*} p'$ and $q <_{s^*} q'$, or $p' <_{s^*} p$ and $q' <_{s^*} q$; both orderings are not compatible to $<_s$. So why not enforce $q <_s q'$ once $p <_s p'$ has occurred? This means to ask for a transaction scheduler who is able to enforce certain relative orderings of conflicting actions such that not serializable orderings cannot occur. Two-Phase Locking, which is called 2PL for brevity in the sequel, is the most widely used technique which can be used to control the concurrent execution of transactions in a way which guarantees serializability.

To control a concurrent set of transactions 2PL uses lock and unlock operations. A locked transaction $t = (op_t, <_t)$ is a transaction which, in addition to the read and write actions, contains lock operations Lx and unlock operations Ux, where Lx, $Ux \in op_t$ and $<_t$ is a total order as before, however now ordering locks and unlocks as well. For a locked transaction $t = (op_t, <_t)$, whenever $p \in op_t$, $p \in \{r(x), w(x)\}$, the following conditions must hold: (i) $Lx, Ux \in op_t$, (ii) $Lx <_t Ux$ and (iii) $Lx <_t p <_t Ux$.

Let $t = r(x)w(x)r(y)w(y)$ be a transaction without any lock and unlock operations. Under the above conditions, there still exist many ways to insert lock and unlock operations into t, as it is demonstrated by the following examples:

$$\begin{aligned}
&(a) Lx\ r(x)\ w(x)\ Ux\ Ly\ r(y)\ w(y)\ Uy,\\
&(b) Lx\ r(x)\ w(x)\ Ly\ Ux\ r(y)\ w(y)\ Uy,\\
&(c) Lx\ r(x)\ w(x)\ Ly\ r(y)\ w(y)\ Ux\ Uy,\\
&(d) Lx\ Ly\ r(x)\ w(x)\ Ux\ r(y)\ w(y)\ Uy,\\
&(e) Lx\ Ly\ r(x)\ w(x)\ r(y)\ w(y)\ Ux\ Uy.
\end{aligned} \tag{1}$$

The idea of a lock operation Lx and the corresponding unlock operation Ux is to grant a transaction t the right of an exclusive access to data item x for the interval defined by the time of the processing of the lock and the processing of the succeeding unlock operation. The decision whether or not a lock can be granted typically is based on a so called lock table $\mathcal{L}$. Starting from an empty table $\mathcal{L}$, a lock operation Lx can only then be granted to a transaction t, if $\mathcal{L}$ does not contain an entry with respect to x. In that case (x, t) is inserted into $\mathcal{L}$. Otherwise transaction t has to wait until the condition is fulfilled. When later t processes the corresponding unlock operation Ux, the entry $(x, t)$is deleted from the lock table $\mathcal{L}$. Therefore, for any history s of locked transaction the following condition on locks and unlocks (LUL) must hold, where $t_1, t_2 \in T$:

LUL : If $L_1\, x <_s L_2\, x$, then $U_1\ x <_s L_2\, x$.

Locking by itself does not guarantee serializability. Consider the two transactions t_1, t_2:

$$\begin{aligned}
t_1 &: L_1x\ r_1(x)\ w_1(x)\ U_1x\ L_1y\ r_1(y)\ w_1(y)\ U_2y,\\
t_2 &: L_2x\ L_2y\ r_2(x)\ w_2(x)\ r_2(y)\ w_2(y)\ U_2x\ U_2y,
\end{aligned}$$

and a history s representing an interleaving of t_1 and t_2:

$$\begin{aligned}
s :& L_1x\ r_1(x)\ w_1(x)\ U_1x\ L_2x\ L_2y\ r_2(x)\\
& w_2(x)\ r_2(y)\ w_2(y)\ U_2x\ U_2y\ L_1y\ r_1(y)\ w_1(y)\ U_1y.
\end{aligned}$$

This schedule is not serializable; $w_1(x) <_s r_2(x)$ and $w_2(y) <_s r_1(y)$ force a cycle $t_1 \rightarrow t_2 \rightarrow t_1$ in the conflict graph which contradicts serializability. Therefore, rules are needed which define how locks and unlocks should be introduced into the single transactions such that, whatever other transactions are running concurrently, serializability is guaranteed. In other words, a locking policy has to be introduced. The most widely used locking policy is called Two-Phase Locking (2PL) and is expressed by the surprisingly simple sentence:

2PL : Whenever a transaction t has executed its *first* operation,no further lock operations *of* t are allowed.

All except the first transaction listed in (1) perfectly obey 2PL. The remaining four locked versions (b) – (e) implement different strategies for locking. Transaction (b) unlocks x as early as possible and locks y as late as possible. It therefore tries to minimize delay of other transactions by keeping the intervals of locked data items as small as possible. At a first glance, this seems to be an attractive approach. However, because of the early unlocking, another transaction may read the value written of x before the transaction has finished. Reading values of data items from transactions which have not yet reached their end is highly problematic for recovery reasons. Only after a transaction has executed its final commit successfully, a database system guarantees that the effects of the respective transactions are permanent and will survive system and transaction failures. Version (c) behaves more carefully and keeps all data items locked until the end of the transaction. The remaining versions (d) and (e) lock all data items in advance and unlock either as early as possible, respectively at the end of the transaction. Taking the possibility of failures into account, only versions (c) and (e) are acceptable. In practice, locks are kept until to the end of a transaction giving rise to the notion of strict 2PL. The following theorem states that 2PL indeed is a safe locking policy:

Theorem: *Let T be a set of transactions. If all* $t \in T$ *obey 2PL, then any history of the transactions in* T *is serializable.*

Proof: For any $t \in T$, the lock point of t is defined as t's last lock operation. Now consider a history s of the transactions in T such that all $t \in T$ obey 2PL. Under the assumption, that s is not serializable, a contradiction to 2PL can be derived thereby proving serializability of s which in turn proves the theorem.

If s is not serializable, then the conflict graph of $<_s$ contains a cycle, which, without loss of generality, is of the form $t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_k \rightarrow t_1$. An edge $t \rightarrow t'$ can only then be part of the cycle, when there exists a data item x such that t and t' perform conflicting actions on x. As all transactions use locks, transaction t' can only then execute its action on x, after t has processed the unlock Ux. Before processing the respective action on x, t' has to lock x. Applying this observation on all edges of the cycle the following restrictions on the order $<_s$ can be concluded, where $x_1,\ldots,x_k$ are data items:

$$\begin{aligned} &U_1x_1 <_s L_2x_1,\\ &\vdots\\ &U_{k-1}x_{k-1} <_s L_kx_{k-1},\\ &U_kx_k <_s L_1x_k. \end{aligned}$$

Let l_i be the lock point of transaction t_i, $1 \leq i \leq k$. From the structure of $<_s$ it follows $l_1 <_s l_2,\ldots,l_{k-1} <_s l_k$ and $l_k <_s l_1$. Therefore it holds $l_1 <_s l_1$, which is a contradiction to the total order $<_s$.

2PL is optimal in the sense that for any locked transaction t_1 which does not follow 2PL, a locked transaction t_2 can be constructed such that for $T = \{t_1,t_2\}$ there exists a history which is not serializable. Consider the lock and unlock operations of t_1 be given by $L_1x <_{t_1} U_1x <_{t_1} L_1y <_{t_1} U_1y$ and the lock and unlock operations of t_2 be given by $L_2x <_{t_2} L_2y <_{t_2} U_2y <_{t_2} U_2x$. t_2 follows the 2PL policy, however t_1 does not. If both transactions are running concurrently, then the following order of locks and unlocks being part of a history s may happen:

$$L_1x <_s U_1x <_s L_2x <_s L_2y <_s U_2y <_s U_2x <_s L_1y <_s U_1y.$$

Obviously, if between any pair of lock and unlock operations Lx, Ux and Ly, Uy there exist read and write actions to the respective data items, conflicts happen and history s is not serializable. Therefore, t_1 has also to follow 2PL when serializability has to be guaranteed for arbitrary sets of transactions T, $t_1 \in T$.

However, optimality in the above sense does not imply that in every serializable history of transactions without locks and unlocks, locks and unlocks can be inserted according to 2PL in a way not violating LUL. Consider a serializable history s of $T = \{t_1,t_2,t_3\}$ with order of actions

$$r_1(x) <_s r_2(x) <_s w_2(x) <_s r_3(y) <_s w_3(y) <_s w_1(y).$$

It is impossible to insert lock and unlock operations into the transactions in T such that any $t \in T$ obeys 2PL and s fulfills LUL. Because of $r_1(x) <_s r_2(x)$ it must hold $U_1x <_s L_2\ x$ and because of 2PL it must hold $L_1y <_s U_1\ x$. However, as $w_1(y) <_s U_1y$, this implies $L_1y <_s r_3(y) <_s w_3(y) <_s U_1y$. Therefore, either $L_3y <_s L_1y <_s U_3y$, or $L_1y <_s L_3y <_s U_1y$. Both orderings contradict the basic locking condition LUL. Therefore, 2PL is a safe locking policy, however some serializable histories of a set of transactions T may be excluded. In other words, 2PL can only accept a strict subset of the set of all serializable histories, in general.

The model for locking presented so far is overly restrictive, as it does not distinguish between read and write actions. Serializability of a history is only then an issue, when some of the involved transactions modify the database, i.e., perform write actions. Therefore, when locking it should be possible to distinguish between locks for read actions $L^R x$ and locks for write actions $L^W x$. If a transaction reads and writes a data item x, then a single write lock is in order. It should be possible that arbitrarily many transactions may lock a data item for reading as long as there is no concurrent transaction writing the same data item. For any history s of locked transaction $t_1, t_2 \in T$ the following conditions on locks and unlocks (LUL^{RW}) must hold, which do not impose any restrictions on the ordering of read locks $L_1{}^R x$ and $L_2{}^R x$:

$$\mathbf{LUL^{RW}}: \begin{array}{l} \text{If } L_1^R x <_s L_2^W x, \text{ then } U_1 x <_s L_2^W x. \\ \text{If } L_1^W x <_s L_2^R x, \text{ then } U_1 x <_s L_2^R x. \\ \text{If } L_1^W x <_s L_2^W x, \text{ then } U_1 x <_s L_2^W x. \end{array}$$

When locking is used, deadlocks may occur. Consider transactions t_1, t_2 as follows: $t_1 = L_1 x\ r_1(x)\ L_1 y\ w_1(y)\ U_1 x\ U_1 y$ and $t_2 = L_2 y\ r_2(y)\ L_2 x\ w_2(x)\ U_2 y\ U_2 x$. A prefix s' of a history s cannot be continued to a complete history containing all the steps of t_1 and t_2, if, for example, $L_1 x <_{s'} L_2 y$ and s' does not contain $U_1 x$. To complete the prefix, $L_2 x$ and also $L_1 y$ have to be considered for $<_s$. However this is not possible, because otherwise the condition LUL would be violated. The problem of deadlocks is inherent to locking and practical systems solve deadlocks by aborting one of the involved transactions [12].

The transaction model abstracts away many aspects of real transactions running in practical systems. In particular, transactions are defined explicitly by a set of steps and not by a program whose concrete execution on a certain state of the database will define the transaction. Assume a program to process all data items which fulfill a certain predicate p. A transaction t resulting from executing the program will read all data items for which p is true. To guarantee serializability, in advance to reading such a data item, the item has to be locked. If concurrently, however later before t has finished, another transaction t' inserts a new data item x into the database which also fulfills predicate p, t has neither read nor locked x. In such situations serializability cannot be guaranteed by 2PL as described so far, because the relevant set of data items has not been locked by t. This problem is known as the phantom problem. Practical systems have found ways to live with phantoms. Instead of only locking data items, locking is applied on the set of objects fulfilling a predicate, on complete relations or index intervals.

Key Applications

For organizations of any kind database systems have become indispensable to maintain the operational data. Applications in this context typically are dominated by a potentially huge number of rather short transactions. In such a setting, commonly called online transaction processing (OLTP), reliable and efficient processing of transactions is required. 2PL has become a de facto standard in database systems for OLTP applications. However, 2PL is based on blocking transactions and therefore system throughput may be severely affected by the locking protocol. Fortunately, over the years several guidelines have been developed [3,7,11] which help a database administrator to tune the programming of the transactions and the implementation of 2PL in a way such that the requisite efficiency of the overall system can be achieved.

Cross-references

- ▶ ACID Properties
- ▶ Concurrency Control – Traditional Approaches
- ▶ Locking Granularity and Lock Types
- ▶ Multi-Version Serializability and Concurrency Control
- ▶ Serializability
- ▶ SQL Isolation Levels
- ▶ Transaction Chopping
- ▶ Transaction Models – The Read/Write Approach

Recommended Reading

1. Agrawal D. and Abbadi A.E. Constrained shared locks for increased concurrency in databases. J. Comput. Syst., Sci.51:(1) 53–63, 1995.
2. Bayer R., Heller H., and Reiser A. Parallelism and recovery in database systems. ACM Trans. Database Syst., 5:(2)139–156, 1980.
3. Bernstein P.A. and Newcomer E. Principles of Transaction Processing for Systems Professionals. Morgan Kaufmann, San Francisco, CA, 1996.

4. Bernstein P.A., Shipman D.W., and Wong W.S. Formal Aspects of Serializability in Database Concurrency Control. IEEE Trans. Software Eng., SE-5:203–215, 1979.
5. Boral H. and Gold I. Towards a Self-adapting Centralized Concurrency Control Algorithm. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 18–32.
6. Eswaran K.P., Gray J.N., Lorie R.A., and Traiger I.L. The notion of consistency and predicate locks in a database system. Commun. ACM, 19:624–633, 1976.
7. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1993.
8. Lausen G. Concurrency Control in Database Systems: A Step towards the Integration of Optimistic Methods and Locking. In Proc. ACM Annual Conf., 1982, pp. 64–68.
9. Papadimitriou C.H. The Serializability of Concurrent Database Updates. J. ACM, 26:631–653, 1979.
10. Salem K., Garcia-Molina H., and Shands J. Altruistic locking. ACM Trans. Database Syst., 19:(1)17–165, 1994.
11. Shasha D. Database Tuning – A Principled Approach. Prentice-Hall, USA, 1992.
12. Weikum G. and Vossen G. 1Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Francisco, CA, 2002.

Two-Poisson model

Giambattista Amati
Ugo Bordoni Foundation, Rome, Italy,

Synonyms

Harter's model; Probabilistic model of indexing

Definition

The 2-Poisson model is a mixture, that is a linear combination, of two Poisson distributions:

$$\mathrm{Prob}(X = \mathbf{tf}) = \alpha \frac{\lambda^{\mathbf{tf}} e^{-\lambda}}{\mathbf{tf}\,!} + (1 - \alpha) \frac{\mu^{\mathbf{tf}} e^{-\mu}}{\mathbf{tf}\,!} \qquad [0 \leq \alpha \leq 1]$$

In the context of IR, the 2-Poisson is used to model the probability distribution of the frequency X of a term in a collection of documents.

Historical Background

The 2-Poisson model was given by Harter [5–7], although Bookstein [2,1] and Harter had been exchanging ideas about probabilistic models of indexing during those years. Harter coined the word "elite" to introduce his 2-Poisson model [5, pp. 68–74].

The origin of the 2-Poisson model can be traced back through all Luhn, Maroon, Damerau, Edmundson and Wyllys [3,4,5,6]. The first accounts on Poisson distribution modeling the stochastic behavior of functional words were given by Stone, Rubinoff and Damerau [3,11]. Stone, Rubinoff and Damerau observed that the words that have only a functional role in the text, can be modeled by a Poisson distribution.

Foundations

The 2-Poisson model is a probabilistic model of indexing, rather than a document retrieval model. The purpose of Harter's work is to identify the keywords likely to be informative for an arbitrary document that can be selected to build an index for a collection. Such words are called *specialty* words by Harter in contraposition to the other ones, the *non-specialty* ones, which instead are considered to occur at random in documents. In the works by Luhn, Maroon, Damerau, Edmundson and Wyllys it was observed that the divergence between the rare usage of a word across the document collection and the contrasting relative within-document frequency constitutes a revealing indication of the informative status of a word. Damerau suggests selecting the class of high status words of the index by making the assumption that Poisson distribution describes frequencies of words that are in the complementary class. If the Poisson probability of a term within a document is very small, then the word is marked as an index term. Obviously, not all words clearly fall either into one class or into the other. But nonetheless, many word tokens occur randomly in many documents while the same word tokens occur more densely and nonrational in a few documents. This set of documents called the *Elite set* of the term **t** is very likely to be the set of documents which extensively connects with the concept or the semantics related to the term. The Elite set $\mathbf{E_t}$ attracts the tokens with an expected rate $\lambda_{\mathbf{E_t}}$. Tokens fall randomly into the other documents with a lower rate $\lambda_{\overline{\mathbf{E_t}}}$. The final probability of occurrence of the term in any document is given by the mixture of these two Poisson distributions:

$$\mathrm{Prob}(X = \mathbf{tf}) = \alpha . \frac{e^{-\lambda_{\mathbf{E_t}}} \lambda_{\mathbf{E_t}}^{\mathbf{tf}}}{\mathbf{tf!}} + (1 - \alpha) . \frac{e^{-\lambda_{\overline{\mathbf{E_t}}}} \lambda_{\overline{\mathbf{E_t}}}^{\mathbf{tf}}}{\mathbf{tf!}} \qquad (1)$$

The probability of term **t** to appear **tf** times in a document **d** belonging to the Elite set $\mathbf{E_t}$ is given by the conditional probability

$$\begin{aligned}\mathrm{Prob}(\mathbf{d}\in\mathbf{E_t}|X=\mathbf{tf}) &= \frac{\mathrm{Prob}(X=\mathbf{tf},\mathbf{d}\in\mathbf{E_t})}{\mathrm{Prob}(X=\mathbf{tf})}\\ &= \frac{\alpha\cdot\frac{e^{-\lambda_{\mathbf{E_t}}}\lambda_{\mathbf{E_t}}^{\mathbf{tf}}}{\mathbf{tf!}}}{\alpha\cdot\frac{e^{-\lambda_{\mathbf{E_t}}}\lambda_{\mathbf{E_t}}^{\mathbf{tf}}}{\mathbf{tf!}}+(1-\alpha)\cdot\frac{e^{-\lambda_{\overline{\mathbf{E_t}}}}\lambda_{\overline{\mathbf{E_t}}}^{\mathbf{tf}}}{\mathbf{tf!}}}\end{aligned} \tag{2}$$

Thus, in the case that the word belongs to the non-specialty class:

$$\mathrm{Prob}(\mathbf{d}\in\mathbf{E_t}|X=\mathbf{tf}) = \frac{1}{1+\beta\cdot e^{\left(\lambda_{\mathrm{Et}}-\lambda_{\overline{\mathrm{Et}}}\right)}\left(\frac{\lambda_{\overline{\mathrm{Et}}}}{\lambda_{\mathrm{Et}}}\right)^{\mathbf{tf}}} \tag{3}$$

where $\beta=\frac{1-\alpha}{\alpha}$. The conditional probability of (3) is used by Harter to generate a ranking of the most informative words. However, the 2-Poisson model requires the estimation of three parameters for each word of the vocabulary, and this is a real drawback for any direct practical application of his model to term selection or term-weighting problems.

A last remark concerns the **N**-Poisson model, the generalization of the 2-Poisson model. Any probability distribution on $(0,\infty)$ can be defined as a mixing distribution of Poisons [10]. Therefore, it is true that every word follows a **N**-Poisson distribution for some N. **N**-Poisson models thus have a practical application only when N is small, that is the 2-Poisson model or the 3-Poisson model.

The 2-Poisson is illustrated with an example. A collection of ten documents contains a word **t** occurring respectively 4, 5, 6, 0, 1, 2, 0, 0, 0, 0 times within these ten documents, the first three documents having the highest frequency of term in the collection. A useful image for this configuration could be that the first three documents constitute an *elite* or *special* subset of documents for the term. Then, these data are fitted to the 2-Poisson model using the Expectation Maximization (EM) algorithm. To set the initial values α_0, λ_0 and μ_0 for the first step of the EM algorithm the assumption is that there is a Poisson distribution generating the frequency in the elite set of the word, with a mean term-frequency $\hat{\mu_0}=\frac{4+5+6}{3}=5$, and a second Poisson distribution generating the term-frequency in the rest of the collection with $\hat{\lambda}_0=\frac{0+1+2+0+0+0+0}{7}=0.4287$. Observe that the elite set is $\frac{3}{10}=0.3333$ of the entire collection, and thus $\hat{\alpha_0}=0.3333$ is initially set. Finally, the EM algorithm converges to the values $\hat{\alpha}=0.35$, $\hat{\mu}=4.5$ and $\hat{\lambda}=0.54$ with a confidence of 0.99. The probability that a document **d** belongs to the Elite set of a term occurring **tf** times in **d** is

$$\mathrm{Prob}(\mathbf{d}\in\mathbf{E_t}|X=\mathbf{tf}) = \frac{1}{1+1.86\cdot e^{3.96}0.12^{\mathbf{tf}}}$$

Key Applications

The 2-Poisson model is at the basis of two types of probabilistic models of IR: the BM25 model and the Divergence From Randomness models.

Cross-references

▶ BM25

▶ Divergence from Randomness Models

Recommended Reading

1. Bookstein A. and Kraft D. Operations research applied to document indexing and retrieval decisions. J. ACM, 24(3):418–427, 1977.
2. Bookstein A. and Swanson D. Probabilistic models for automatic indexing. J. Am. Soc. Inform. Sci., 25:312–318, 1974.
3. Damerau F. An experiment in automatic indexing. Am. Doc., 16:283–289, 1965.
4. Edmundson H.P. and Wyllys R.E. Automated abstracting and indexing–survey and recommendations. Commun. ACM, 4(5):226–234, May 1961. Reprinted in Readings in Information Retrieval, pp. 390-412. H. Sharp (ed.). New York, NY: Scarecrow; 1964.
5. Harter S.P. A probabilistic approach to automatic keyword indexing. PhD thesis, Graduate Library, The University of Chicago, Thesis No. T25146, 1974.
6. Harter S.P. A probabilistic approach to automatic keyword indexing. part I: On the distribution of specialty words in a technical literature. J. American Soc. for Inf. Sci., 26:197–216, 1975.
7. Harter S.P. A probabilistic approach to automatic keyword indexing. part II: An algorithm for probabilistic indexing. J. American Soc. for Inf. Sci., 26:280–289, 1975.
8. Luhn H.P. A statistical approach to mechanized encoding and searching of literary information. IBM Journal of Research and Development, 1:309–317, 1957.
9. Maron M.E. Automatic indexing: an experimental inquiry. J. ACM, 8:404–417, 1961.
10. Puri P.S. and Goldie C.M. Poisson mixtures and quasi-infinite divisibility of distributions. J. Appl. Probab., 16(1):138–153, 1979.
11. Stone D. and Rubinoff B. Statistical generation of a technical vocabulary. Am. Doc., 19(4):411–412, 1968.

Two-Sorted First-Order Logic

► Temporal Relational Calculus

Type Theory

► Data Types in Scientific Data Management

Type-based Publish/Subscribe

Hans-Arno Jacobsen
University of Toronto, Toronto, ON, Canada

Definition

Type-based publish/subscribe is an instance of the publish/subscribe concept, where publications are instances of application-defined types, subscriptions express interest in receiving publications of a specified type or sub-type, and publish/subscribe matching amounts to type conformance checking.

Key Points

The characterization of the type-based publish/subscribe class originated in the programming languages context with the objective of bridging the impedance mismatch in the integration of publish/subscribe abstractions into programming languages. For example, the representation of subscriptions as strings that are parsed, checked, and processed at runtime by the underlying publish/subscribe implementation, are frequent sources of errors that materialize too late due to the inability to properly type check the subscription string at the language level. To enable static type checking, type-based publish/subscribe includes subscriptions and publications as first class citizens into the language.

Publications become instances of language types. Arbitrary composite types, subject to the type definition capabilities of the host language, can be used to model application-defined events. Subscribers express interest in receiving publications of a specified type. Matching of publications against subscriptions amounts to type conformance checking. That is a publication matches a subscription, if the type of interest specified by a subscription conforms to the type of a publication, for some definition of type conformance. Matching is based on type determination, sub-type checking, and is-instance-of type checks. This is similar to topic-based publish/subscribe matching, except that a type is more general than a topic. Moreover, types may support operations that can model content-based predicates as tests on the instances of types. In this sense, type-based publish/subscribe can model content-based processing.

Several standards, such as the OMG Notification Service [2] and the OMG Data Dissemination Service [3] exhibit elements of type-based publish/subscribe, but do not directly follow the prescription of the type-based publish/subscribe model. The characterization of type-based publish/subscribe can be found in [1].

Cross-references

► Publish/Subscribe
► Topic-Based Publish/Subscribe

Recommended Reading

1. Eugster P. Type-based publish/subscribe: concepts and experiences. ACM Trans. Program. Lang. Syst., 29(1):6, 2007.
2. MG. Notification Service Specification, version 1.1, formal/04–10–11 edition, October 2004.
3. MG. Data Distribution Service for Real-time Systems, version 1.2, formal/07–01–01 edition, January 2007.

UML

► Unified Modeling Language

Uncertain Databases

► Probabilistic Databases

Uncertain Information

► Incomplete Information

Uncertainty in Events

Segev Wasserkrug
IBM Research, Haifa, Israel

Synonyms

Event uncertainty

Definition

Uncertainty in events is uncertainty regarding either the occurrence of an event, or uncertainty regarding the data values associated with an event. This uncertainty is a result of a gap between the actual occurrences of events in the real world, and the availability of knowledge regarding the events.

Historical Background

The first event-based systems were active databases, in which automatic actions were carried out as a result of database queries. This was done using the *ECA* (Event-Condition-Action) paradigm. However, in many such database applications, the events of interest were not the results of single queries, (e.g., insertion or deletion of data), but rather could be deterministically inferred from several such queries. To facilitate such inferences, event inference languages were defined. Initially such languages were specific to active databases (e.g., SNOOP and ODE). However, more general languages were developed suitable for implementing such deterministic inferences in any event-based system. These general languages resulted from the need to enable event driven behavior in a wide variety of application domains and became part of a wider area known as Complex Event Processing (CEP). As event driven applications became more complex, it became necessary to handle uncertainty regarding the occurrence and inference of events.

Foundations

For a system to implement event driven behavior, the system must be able to recognize all events of interest. However, in many cases, there is a gap between the actual occurrences of events to which the system must respond and the data generated by monitoring tools regarding these events. This gap results in uncertainty.

To understand this, consider a thermometer that generates an event whenever the temperature rises above 37.5°C. The thermometer is known to be accurate to within ±0.2°C. Therefore, when the temperature measured by the thermometer is 37.6°C, there is some uncertainty regarding whether the event has actually occurred.

Another gap between the actual occurrence of events and the information available to the system is caused by the following: The information regarding the occurrence of some events (termed *explicit events*) is signaled by event sources (e.g., monitoring tools such as the thermometer described above), while for other events, explicit notification is never sent (*non-explicit events*). An example of a non-explicit event is insider trading. Although insider trading either does or does not take place, no explicit signal regarding such an event is generated.

For an event-based system to respond to non-explicit events, in many cases the occurrence of these events must be inferred based on the occurrence of other events. (The events based on such inference are termed *inferred events*.) To facilitate such inferences, several event composition languages have been defined that make it possible to infer non-explicit events based on a set of complex temporal predicates. In many cases, however, such inference cannot be carried out with certainty. To see an example, consider again the case of insider trading. While a single large sale of stock may not be indicative of anything suspicious, such a sale, together with a sharp change in stock prices of the company due to an announcement in the press may infer insider trading. However, such insider trading cannot always be said to have occurred whenever a combination of a sale event and a stock decline event occur.

Even when the inference rules may be deterministically stated, uncertainty associated with explicit events may also propagate to inferred events. To see this propagation of uncertainty, consider a rule (defined in an event composition language) that states that event e_3 must be inferred whenever an event of type e_2 occurs after an event of type e_1. Moreover, assume that it is known that both an event of type e_1 and an event of type e_2 have occurred, but there is uncertainty about the exact time of their occurrence because the occurrence of e_1 is known to be between time 2 and time 5, and the occurrence of e_2 is known to be between time 3 and time 7. Note that even though the rule is deterministic, the uncertainty regarding the occurrence times of e_1 and e_2, and the fact that the occurrence of e_3 must be inferred based upon these occurrence times, results in uncertainty with regards to the occurrence of the event e_3.

Dimensions of Event Uncertainty

It is useful to classify the uncertainty according to two orthogonal dimensions: *element uncertainty* and *origin uncertainty.*

Element uncertainty refers to the fact that event-related uncertainty may involve one of two elements:

1. *Uncertainty regarding event occurrence*: Such uncertainty is associated with the fact that although the actual event occurrence is atomic, (i.e., the event either did or did not occur) the event-based system does not know whether or not this event has, in fact, occurred. An example of this is insider trading. At any point in time, insider trading either was or was not carried out by some customer. However, an event-driven system can probably never be certain whether insider trading actually took place.
2. *Uncertainty regarding event attributes*: Even in cases in which the event is known to have occurred, there may be uncertainty associated with its attributes. For example, while it may be known that an event has occurred at some point in time, its exact time of occurrence may not be precisely known.

Origin uncertainty pertains to the two types of events (explicit and inferred) that exist in an event-based system. Due to these two types of events, there are two possible origins for uncertainty:

1. *Uncertainty originating at the event source*: When an event originates at an event source, there may be uncertainty associated either with the event occurrence itself or the event's attributes, due to a feature of the event source. An example is the limited precision thermometer described previously, where uncertainty regarding an event occurrence (i.e., the temperature being above 37°C) is caused by the limited measuring accuracy of a thermometer (i.e., the reading being accurate only to within ±0.2°C).
2. *Uncertainty resulting from event inference*: Due to some events being inferred based on other events, uncertainty can propagate to the inferred events. This is demonstrated by the rule previously described, which is used to infer events of type e_3 based on events of types e_1 and e_2. In the aforementioned case, uncertainty regarding the occurrence of event e_3 resulted from uncertainty regarding the time that events e_1 and e_2 occurred.

Based on the above two dimensions, it is possible to define four types of event uncertainty. These are the following:

1. Uncertainty regarding event occurrence originating at an event source;
2. Uncertainty regarding event occurrence resulting from inference;

3. Uncertainty regarding event attributes originating at an event source; and
4. Uncertainty regarding event attributes resulting from event inference.

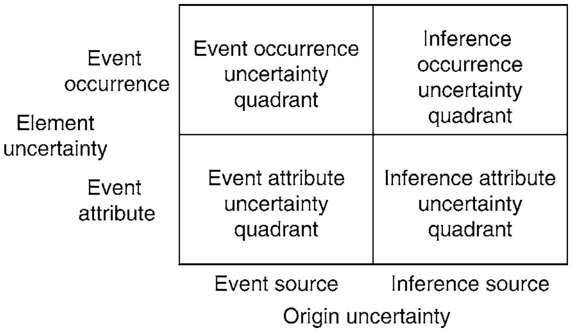
These uncertainty types are depicted as quadrants in Fig. 1.

Causes of Event Uncertainty

There are many possible causes for event uncertainty. In addition, the uncertainty causes for explicit events are different from the uncertainty causes of inferred events.

These are the sources of uncertainty for explicit events:

1. *An unreliable source*: An event source may malfunction, indicating that an event has occurred even if it has not. Similarly, the event source may fail to signal the occurrence of an event which has, in fact, occurred. A source may also transmit erroneous information regarding one (or more) of the event's attributes.
2. *An imprecise event source*: An event source may operate correctly, but still fail to signal the occurrence of events due to limited precision (or may signal events that did not occur). Such a source may also be the cause of imprecision regarding an event's attributes.
3. *Problematic communication medium between the event source and the event-based system*: Even if the event source has full precision and operates correctly 100% of the time, the communication medium between the source and the event-based system may drop indications of an event's occurrence, generate indications of events that did not occur, or alter information regarding the event's attributes.



		Origin uncertainty	
		Event source	Inference source
Element uncertainty	Event occurrence	Event occurrence uncertainty quadrant	Inference occurrence uncertainty quadrant
	Event attribute	Event attribute uncertainty quadrant	Inference attribute uncertainty quadrant

Uncertainty in Events. Figure 1. Event uncertainty types.

4. *Uncertainty due to estimates*: In some cases, the event itself (or its attributes) may be the result of a statistical estimate. For example, it may be beneficial to generate an event whenever a network Denial of Service (DoS) event occurs, where the occurrence of such a DoS event is generated based on some mathematical model. However, as the mathematical model may produce erroneous results, this event also has uncertainty associated with it.
5. *Clock synchronization in distributed systems*: This is a cause of uncertainty regarding the occurrence time of events in a distributed system. This is due to the fact that in distributed systems, the clocks of various nodes are usually only guaranteed to be synchronized to within some interval of a global system clock. Therefore, there is uncertainty regarding the occurrence time of events as measured according to this global system clock.

These are possible causes of uncertainty of inferred events:

1. *Propagation of uncertainty*: There are cases in which an inferred event can be deterministically inferred based on other events. Even in such cases, there may be uncertainty regarding the inferred event, resulting from uncertainty regarding the events based on which the inference is carried out.
2. *Uncertain inference*: There are cases in which the inference is inherently uncertain. One example of this is insider trading, where events denoting suspicious purchases and sales of stock coupled with rapid price changes only serve to indicate the possible occurrence of an insider trading event. In such cases, an insider trading event cannot be inferred with certainty based on such suspicious transactions. An additional prominent example is when the event driven application is required to predict the occurrence of future events.

Note that for both explicit and inferred events, uncertainty regarding a specific event may be caused by a combination of factors. For example, for a specific event, it is possible that both the event source and communication medium simultaneously corrupt the information sent regarding this event.

Handling Uncertainty in Events

There is a spectrum of possibilities by which such event uncertainty may be handled. One end of this spectrum has methods that explicitly or implicitly ignore the presence of uncertainty. The other end is a complete and formal treatment of such uncertainty. Details about these two extremes are provided below. Obviously, solutions which lie between these two extremes are also possible.

The methods that explicitly or implicitly ignore such uncertainty usually rely on deterministic event composition languages to enable the event driven functionality required. In many cases, applications that ignore the uncertainty rely on a human being to make the final decision regarding whether or not an event of interest occurred. An example of such an application is credit card fraud detection. In many such systems, the system is expected to recognize suspicious patterns of credit card transaction events that may indicate fraud. Such indications are then used to alert a human operator, whose role it is to establish whether such a fraud indeed took place.

A full and formal treatment of such uncertainty is required in applications where automatic actions must be carried out as a result of events of interest. In such cases, the framework for dealing with event uncertainty must include the following components:

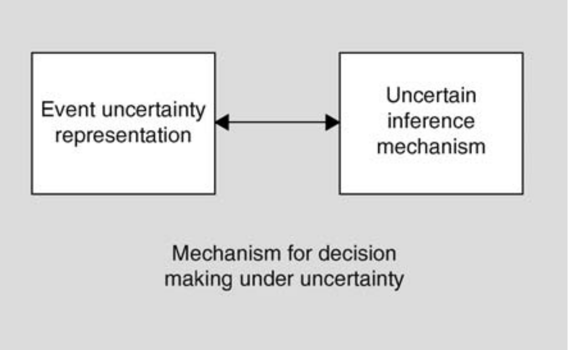
1. Mechanism for automatic decision making that can take uncertainty into account
2. Mechanism for enabling uncertain inference regarding events
3. Representation method of the uncertainty associated with each event

A depiction of such a framework appears in Fig. 2.

Example of Event Uncertainty Handling Framework

This section provides an example of an uncertainty handling framework that has the three components described in Fig. 2.

Underlying any framework for handling uncertainty in events is an uncertainty handling mechanism. Many such formalisms exist, including *Lower and Upper Probabilities*, *Dempster-Shafer Belief Functions*, and *Possibility Measures*. The framework discussed in this section is based on probability theory, which is the most well known framework for quantitative representation and reasoning about uncertainty.


Event uncertainty representation
Uncertain inference mechanism
Mechanism for decision making under uncertainty

Uncertainty in Events. Figure 2. Elements of event uncertainty handling framework.

In this framework, the uncertainty regarding an event is represented as follows: the event may have multiple sets of values associated with it. Each such set of values corresponds to possible attribute values of this event. The uncertainty is then quantified by probabilities for this set of values. For example, consider an event that either has not occurred, has occurred at time 5, or has occurred at time 10. This event has the set of values {*notOccurred*, 5, 10} associated with it. Furthermore, if the probability of the event not occurring is 0.5, the probability of the event having occurred at time 5 is 0.3 and the event having occurred at time 10 is 0.2. This is represented by {{*notOccurred*,0.5},{5,0.3},{10,0.2}}.

The framework enables uncertain inference by using a language that can specify uncertain rules together with an inference algorithm that enables the probabilities of interest to be calculated. The uncertain rules in the framework are of the form "If event e_1 and event e_2 occurred, then the probability of event e_3 occurring is 0.7." The inference framework ensures that the probabilities of the inferred events are taken into account in a manner consistent with probabilistic dependencies and independencies between the events.

The automatic decision making mechanism is based on utility theory.

Key Applications

Event driven functionality is required in almost all application domains. Therefore, the use of event-based systems is widespread. Two prominent examples of applications in which a complete treatment of uncertainty is required are security applications and sensor network applications.

In security applications, the response time available to respond to threats requires that automatic actions be carried out. An example is network security applications where it is important to respond quickly to Denial of Service (DoS) attacks. There is a need to provide a quick response (e.g., in milliseconds) to the threat, since waiting too long to respond may mean that the network may already be too saturated for a response to be of use. On the other hand, wrongly determining that a DoS attack has occurred and carrying out measures such as shutting down certain ports, may result in unjustifiable denial of service to legitimate network traffic. In such a case, an event driven framework for automatic decision making under uncertainty is required.

In sensor network applications, a large number of sensors continuously transmit relatively rudimentary data (in the form of events) to some central server. An example may be a monitoring application, in which sensors throughout a large office building constantly transmit the temperature of each room in the building. An application may then be required to detect the possibility of a fire occurring in the building based on the individual readings or to aggregate the individual readings.

Because of current technical limitations, each individual sensor is unreliable. Therefore, it is quite possible that a temperature reading is not transmitted by the sensor or that there is an error in the transmitted reading. Due to this unreliability, an explicit treatment of uncertainty is required in such domains.

Future Directions

While research in event-based systems and event driven applications has been ongoing for several years, relatively little research has been carried out on the explicit treatment of uncertainty in events. Moreover, most such research has been focused on specific domains such as sensor networks and security application. Therefore, much work remains in exploring new and formal approaches to such treatment. An additional important future direction is the implementation of a production level system that enables event inference under uncertainty in the general case.

Cross-references

- ▶ Active and Real-time Data Warehousing
- ▶ Atomic Event
- ▶ Complex Event
- ▶ Complex Event Processing
- ▶ Composite Event
- ▶ Event
- ▶ Event and Pattern Detection over Streams
- ▶ Event Driven Architecture
- ▶ Event Prediction
- ▶ Explicit Event
- ▶ Implicit Event

Recommended Reading

1. Balazinska M., Khoussainova N., and Suciu D. PEEX: extracting probabilistic events from rd data. In Proc. 24th Int. Conf. on Data Engineering, 2008.
2. Halpern J.Y. Reasoning About Uncertainty. MIT Press, Cambridge, MA, 2003.
3. Li C.-S., Aggarwal C., Campbell M., Chang Y.-C., Glass G., Iyengar V., Joshi M., Lin C.-Y., Naphade M., and Smith J.R. Epi-spire: a system for environmental and public health activity monitoring. In Proc. IEEE Int. Conf. on Multimedia and Expo., 2003.
4. Luckham D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Reading, MA, 2002.
5. Patton N.W. Active Rules in Database Systems. Springer, Berlin, 1999.
6. Wasserkrug S., Gal A., and Etzion O. A model for reasoning with uncertain rules in event composition systems. In Proc. 21st Annual Conf. on Uncertainty in Artificial Intelligence, 2005.

Uncertainty Management in Scientific Database Systems

NILESH DALVI
Yahoo! Research, Santa Clara, CA, USA

Definition

Scientific databases often deal with data that comes from multiple sources of varying quality, is heterogeneous, incomplete and inconsistent, and ridden with measurement errors. Uncertainty management deals with a set of techniques for modeling and representing the various uncertainties that arise in scientific data and to enable users to query the data. This entry describes the UII system [10] that addresses the issue of managing uncertainty in integrating scientific databases.

Historical Background

Distributed data integration is becoming increasingly popular in biomedical research and in scientific research in general. Its popularity is based on the realization that combining sources frequently lead to novel scientific discoveries that cannot be concluded from any single source in isolation. However, as more and more scientific data is shared and as tools are built to provide a common query interface for them, the scientists face the major problem of dealing with information overload [19]. There are several factors that affect the quality of answers to the user queries: the quality of the data sources themselves, the quality of links across data sources, the alignment of data across sources, and so on. The data sources have a large variation in the quality of their data, caused by the curation method, provenance and the experimental protocols used in data collection. Some databases are nicely linked, with one containing foreign keys to other, while others have to be linked by matching tuples over text fields. The scientific data across sources is often not aligned, due to experimental errors and lack of common standards. Specialized algorithms exist to align scientific data, e.g., the BLAST [16] sequence alignment algorithm for aligning potentially similar proteins, but these algorithms are speculative at best. As a consequence, the user is left with the daunting task of searching for relevant answers among a vast number of spurious results.

Uncertain data management [15] has seen a recent renewed interest in the databases community. This is because of the thrust from applications that span not just scientific databases but several others that include exploratory queries in databases, novel IR-style approaches to data integration, information extraction from Web, sensor networks applications and data privacy analysis. There are efforts to build general purpose uncertain data management systems, e.g., MystiQ [2] and Trio [18]. In parallel, there are several efforts in the community to develop scientific data management systems and tools to ease the tasks of scientists and facilitating the process of sharing and reusing experimental data. This entry describes the application of uncertain data management techniques to scientific databases. It gives several examples of specific uncertainties that arise in managing scientific data and describes the UII system [10] that addresses these issues.

Foundations

This section provides examples of data uncertainties that arise in scientific data.

Uncertainties in Scientific Data

The uncertainties in scientific data can be classified into two broad categories:

1. *Inherent Data Uncertainties* Inherent data uncertainties are attributes of the data itself and not artifacts of its representation. Data generated from laboratory experimental methods often have inherent uncertainties. To illustrate an extreme case, two-hybrid screening assays, which are used to detect protein interactions, have error rates estimated to be close to 50% [2]. Experimental data can also be generated from computational experiments. The BLAST algorithm [3] searches in a database for sequences similar to a query sequence. The similarity between any two sequences is measured by the BLAST evalue, which is the degree to which the pairing could occur by chance. Additionally, uncertainties can be rooted in the ever-evolving nature of biological knowledge itself. For example, GenBank references sequences (RefSeqs) are assigned status codes which refer to the amount of evidence and expert curation attributed to a given sequence and its function [4]. These codes range from "inferred" where there is little support for a given sequence, to "reviewed" where substantial evidence exists and has been vetted by a biological domain expert. Status codes for sequences change over time as evidence for them accumulates.
2. *Data Representation Uncertainties* Data representation uncertainties result from the mapping of real world information onto a computable representation of this information. At last count there were over 600 online data sources in molecular biology [5]. Unfortunately, for all the data that is available there are no common standards for representing it (in part due to the evolving nature of biomedical knowledge). The result is the decentralized and heterogeneous nature of biological data sources which is an underlying source of many data uncertainties. For instance, there is no common identifier for a biological object [6] which make it difficult to query across data sources (manually or otherwise), a task which is commonly performed. Linkages between data records may then require string matches on text fields rather than more reliable "foreign-key" relationships. Additionally, data sources tend to represent data in idiosyncratic fashion. For

example, GenBank uses RefSeq status codes to represent the level of evidence for a particular gene but the Gene Ontology (GO) uses evidence codes [7]. Given evidence from both sources, it is sometimes difficult to make comparisons, such as determining which code provides the greater weight of evidence.

General Purpose Systems for Managing Uncertainty

The problem of managing uncertainty in databases has a long history [1,3,6,9] with a recent renewed interest, and several systems for managing uncertain data have been proposed recently in the literature.

Trio The Trio [18] system being developed at Stanford is a data management system that supports uncertainty and lineage. The focus of the system is on the representation formalisms for uncertainty and lineage as well as query languages over such data. Trio extends the relational database model with (i) *alternative values*, where tuple attributes may be assigned a set of possible values rather than a single value, (ii) *maybe* ("?") annotations, specifying that a tuple may not exist, (iii) optional *numeric confidences* attached to alternatives and (iv) *lineage*, connecting tuple-alternatives to other tuple alternatives from which they were delivered. The Trio query language extends SQL to support querying based on tuple-alternatives as well as lineage.

MystiQ The MystiQ [2] system from University of Washington is another general-purpose system which supports various constructs for handling uncertainty that include probabilities associated with tuples, approximate predicates in SQL queries and *soft views* over uncertain data. The focus of the system is on efficient query evaluation, and uses various techniques like *safe query plans* [4] and Monte-carlo approximations [11].

Other Systems The Orion system [14] at Purdue looks at uncertainty given by continuous-valued probability distributions. This is an important problem which is specially relevant to managing uncertainty in sensor networks, where the actual sensor readings often have continuous-valued uncertainty associated with it. The work looks at supporting aggregate queries over such data items. There is also very interesting research [5,7,12] being carried out on extending probabilistic databases to represent correlated tuples and graphical models.

Similarly, there are several systems that exist for Scientific data management. However, several challenges must be met to combine the uncertain data management techniques with these systems. First, it requires an analysis of various sources of uncertainties that arise in Scientific data management, and a principled way to quantify and represent these uncertainties. Secondly, the uncertain data management systems primarily use SQL queries over relational data. This model is unsuitable [13] for scientific databases as the scientists using the system are unfamiliar with the concepts of databases and SQL queries and prefer a simpler interface to browse and explore the data. The rest of this entry describes the UII system[10] which is a specialized scientific data management system with support for uncertainty.

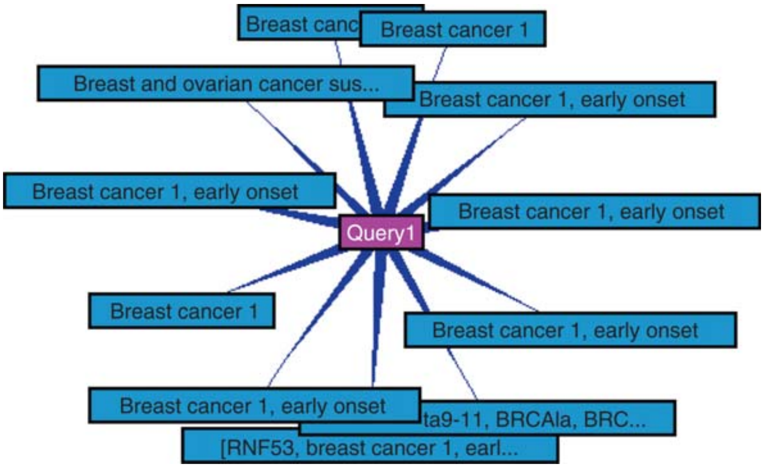
The UII System for Managing Uncertainty in Scientific Data

UII is built on top of the BioMediator system [13,17], which is a mediated-schema distributed data integration system being developed at the University of Washington. The query model of BioMediator is designed to address the need of biologists, which supports poorly specified, exploratory kind of queries.

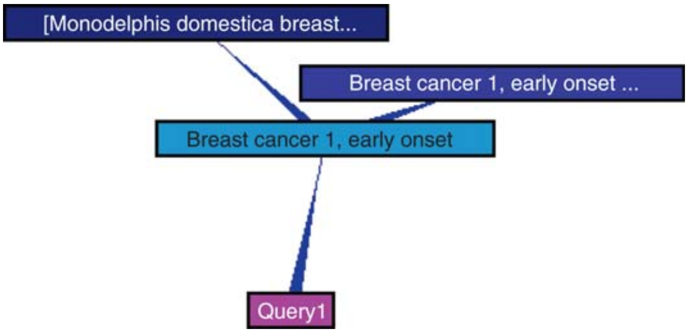
A user begins by issuing a simple query, called the *seed query*, which establishes the basic topic of interest. The user can then browse these results and explore any of the results further by finding information about it in other sources, a process called *query expansion*.

Figure 1 shows a sample seed query: `Gene` records containing a `Symbol` attribute with value `BRCA1`. The system determines which of the included data sources contain `Gene` information and then queries these sources for entries satisfying the specified constraints. The result is shown as a graph that contains one query node and several results node linked to it.

A user can expand a set of node by clicking them, which causes them to join with tuples from other data sources, using the information present in the nodes to be expanded. Such information may include foreign keys to other sources allowing direct look-ups. Often, in the absence of unique foreign keys, the joins involve partial matches over (multiple) free text fields. Figure 2 shows an example of a single step of query expansion. The one light node is expanded once resulting in two dark nodes.


Breast canc
Breast cancer 1
Breast and ovarian cancer sus...
Breast cancer 1, early onset
Breast cancer 1, early onset
Query1
Breast cancer 1, early onset
Breast cancer 1
Breast cancer 1, early onset
Breast cancer 1, early onset
ta9-11, BRCAla, BRC...
[RNF53, breast cancer 1, earl...

Uncertainty Management in Scientific Database Systems. Figure 1. A seed query.


[Monodelphis domestica breast...
Breast cancer 1, early onset ...
Breast cancer 1, early onset
Query1

Uncertainty Management in Scientific Database Systems. Figure 2. A single step in query expansion.

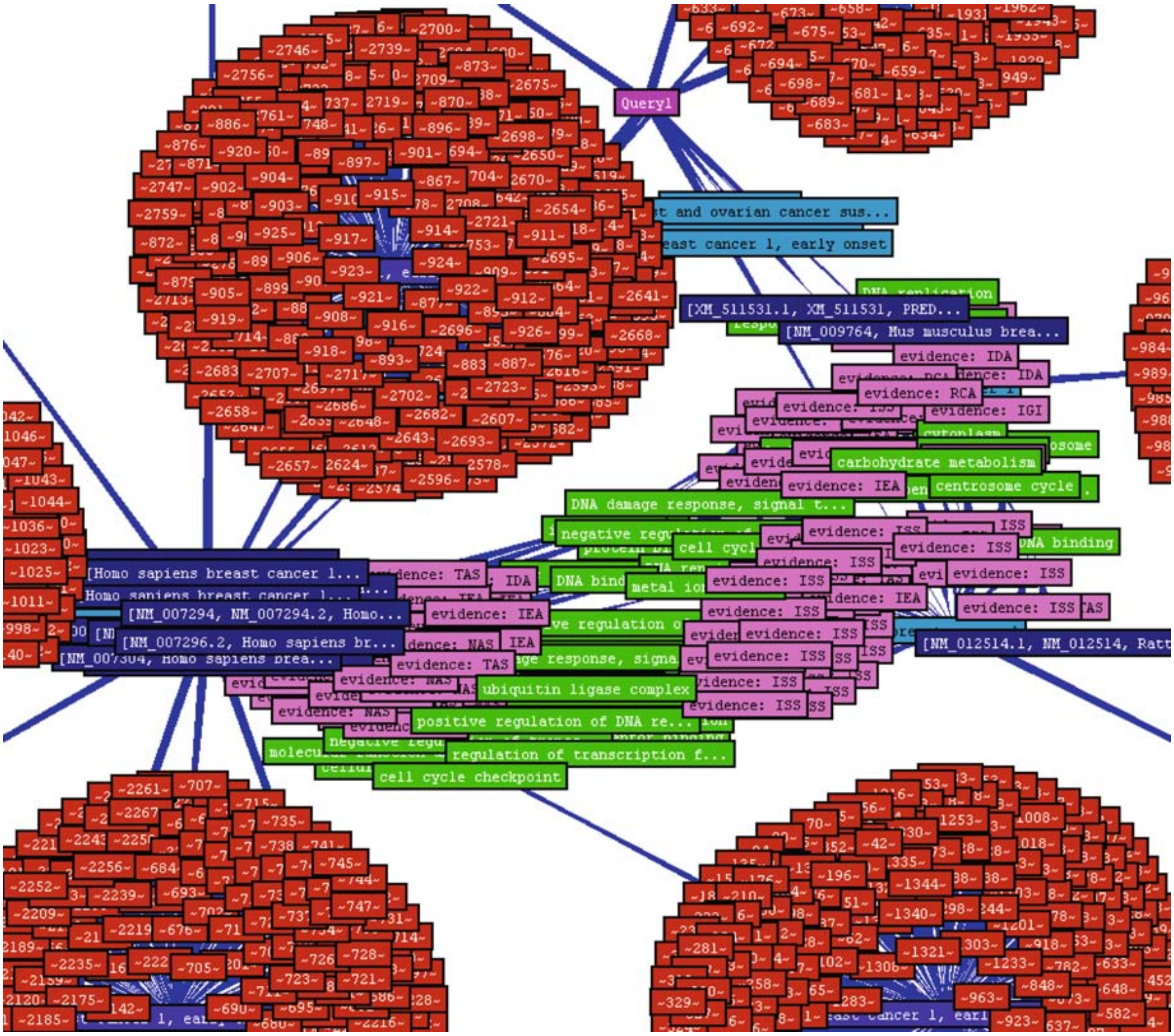
The browsing history of an user is abstracted in terms of a *browsing graph*. A browsing graph is a directed graph whose nodes are records from data sources, and edges correspond to node expansions. It has a distinguished query node, which in the seed query used to initiate the browsing. There can be multiple paths from the start node to a give node, which corresponding to multiple expansion paths to the same node.

Handling Uncertainty

The problem of searching for relevant query answers is exemplified by Fig. 3, which shows a fragment of browsing graph with over 5,000 results, derived by just two complete expansion steps (where in each step all the nodes are expanded) starting from the same seed query shown in Fig. 1. However, not all the nodes in the browsing graph are equally likely to be useful to the user. A node may be of low quality because of the data source that contributes that node. An expansion edge may be of low quality because it was joined using a partial match on a text field. The farther a node is from the start node, the less likely it is to be relevant to the user. The relevance of nodes to the user is assessed in two steps. In the first step, the browsing graph is annotated with *uncertainty metrics*, that describe the quality of each node and each edge in the graph. In the second step, these metrics are used to compute the relevance of each node to the user. The two steps are described below.

Uncertainty Metrics The following is a summary description of the four fundamental uncertainty metrics that capture all types of uncertainty in the UII system:

Ps Measure: Ps is a measure defined at the schema level. It is a quantification of user's prior belief in the quality of data records of a particular data type from a particular data source (e.g., Genes from Entrez Gene, Classifications from Entrez Gene, and Classifications from GO, are each assigned Ps values). For example, consider the comparison between proteins from


Query1
t and ovarian cancer sus...
east cancer 1, early onset
[XM_511531.1, XM_511531, PRED...
[NM_009764, Mus musculus brea...
DNA replication
evidence: IDA
evidence: RCA
evidence: IGI
carbohydrate metabolism
evidence: IEA
centrosome cycle
DNA damage response, signal t...
negative regu
cell cycl
evidence: ISS
DNA binding
[Homo sapiens breast cancer 1...
Homo sapiens breast cancer 1...
[NM_007294, NM_007294.2, Homo...
[NM_007296.2, Homo sapiens br...
[NM_007304, Homo sapiens brea...
evidence: TAS
evidence: NAS
DNA bind
metal ion
ubiquitin ligase complex
positive regulation of DNA re...
regulation of transcription f...
cell cycle checkpoint
[NM_012514.1, NM_012514, Ratt

Uncertainty Management in Scientific Database Systems. Figure 3. A small portion of a large BioMediator result set. The seed query plus two subsequent expansions produced over 5,000 results.

SwissProt and TrEMBL. SwissProt is a manually and carefully curated data source of protein functional information whereas TrEMBL contains only computational predictions which are deemed less reliable. The class of protein records from SwissProt therefore are assigned a higher Ps value than those from TrEMBL because SwissProt protein records are generally trusted to a greater degree.

Qs Measure: Qs is also defined at the schema level. It is a quantification of user's prior belief in the quality of links between to data types in two different data sources (e.g., Genes in Entrez Gene to Proteins in Entrez Protein). To elaborate, records in some sources, such as Gene records from Entrez Gene, contain references to records in another source, such as Protein records from Entrez Protein. In this example, these references are in the form of "accession" numbers which essentially correspond to unique identifiers (foreign keys). Records between other types and sources however may only be connected by non-foreign keys, e.g., text-string similarities such as is the case between Genes from Entrez Gene and Genes in OMIM. In this example, the relationship between Genes in Entrez Gene and Proteins in Entrez Protein is assigned a higher Qs value since these links are better in general than those between Genes in Entrez Gene and Genes in OMIM.

Pr Measure: This is defined at data level and is a quantification of user's belief in a particular data record. It is used to capture data uncertainties which differ between records of the same type and source.

Gene records in Entrez Gene, for example, are attributed with a Refseq status code which ranges in value from "inferred" to "reviewed." These status codes correspond to the amount of evidence for a given gene, therefore "reviewed" are assigned a higher Pr value than "inferred."

Qr Measure: Qr is also defined at the data level, and is a quantification of user's belief in a particular cross-reference (link) between two data records. It is dynamic (calculated at the time two linked results are returned by the system). For example, record cross-references using unique identifiers always receive a Qr of 1.0. Some records may reference others via the use of comparison algorithms such as BLAST. For BLAST cross-references, Qr scores are dynamically computed by converting the e-value from the BLAST algorithm into a numeric value between 0.0 and 1.0. BLAST comparisons that correspond to better matches between records (higher similarity) receive higher Qr values.

Probabilistic Query Evaluation Each query in UII system results in a set of nodes forming the browsing graph for the query. The objective is to find the relevance of each node in the browsing graph to the original seed query. This can be posed as a network reliability problem [8]. For each node in the graph, the quantity $Ps * Pr$ is interpreted as the probability that the node is correct. Similarly for each edge, $Qr * Qs$ is interpreted as the score that the edge is correct. Finally, the relevance of a node is simply the probability that the node is reachable from the seed query node in the browsing graph.

The exact relevance computation is an intractable problem [8]. However, the probabilities can be efficiently approximated to arbitrary precision using simulation algorithms. The following simulation algorithm can be employed, which is well suited for the needs of the problem. In a single pass, N trials (path traversals in the graph) are simulated where nodes and edges are included in the traversal with their associated probabilities. This is done by first storing a random N-bit vector with each node/edge, that denotes whether the node is in or out in the corresponding experiment. Finally, performing a single depth-first search and using these bit vectors, a final bit vector for each node is obtained that describes the trials where that node was reachable from the start node. The relevance for a node is then estimated using the quantity k/N, where k is the number of set bits in the node's final bit vector. The choice of N influences the error in the estimation, the larger the N the smaller the error. Also, for any fixed value of N, the larger the actual relevance of a node, the better the approximation. Thus, the simulation will correctly rank the most relevant answers, which the user cares about most, while the poorest results may be slightly out of order.

Key Applications

Uncertainty management techniques are fundamental to *Scientific Data Integration* applications.

Future Directions

Uncertainty is a fundamental issue in scientific databases, and there are several open and challenging problems which need to be addressed. One important problem is to assign meaningful probability scores to data in a principled way. While the current approach requires domain experts to specify Ps, Pr, Qs and Qr values, it would be more desirable to automatically ascribe them using machine learning techniques. Uncertainty management systems should also be able to relearn and update these scores based on users' feedback to query results. Another problem is to develop highly scalable and efficient query answering algorithms that go beyond the current simulation based techniques. A potential approach in this direction is to use the uncertainty scores to enable a focused expansion of the query rather than constructing the whole browsing graph a priori and ranking its nodes. Another important aspect of managing uncertainty in scientific data is keeping track of data provenance. As scientists use experimental data from different sources and their own experimental data to generate new data, it becomes important to track the provenance of data through these computing processes. The uncertainties in scientific data are often correlated in ways dictated by their provenance, and managing uncertainty along with the provenance information is a challenging and important future direction.

Cross-references

- ▶ Data Cleaning
- ▶ Data Integration
- ▶ Data Provenence
- ▶ Data Quality Models
- ▶ Inconsistent Databases
- ▶ Probabilistic Databases

- ▶ Provenence of Scientific Databases
- ▶ Record Linkage
- ▶ Record Matching

Recommended Reading

1. Barbará D., Garcia-Molina H., and Porter D. The management of probabilistic data. IEEE Trans. Knowl. Data Eng., 4(5):487–502, 1992.
2. Boulos J., Dalvi N., Mandhani B., Mathur S., Re C., and Suciu D. Mystiq: a system for finding more answers by using probabilities. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 891–893.
3. Cavallo R. and Pittarelli M. The theory of probabilistic databases. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 71–81.
4. Dalvi N. and Suciu D. Efficient query evaluation on probabilistic databases. In Proc. 26th Int. Conf. on Very Large Data Bases, 2004, pp. 864–875.
5. Deshpande A. and Sunita Sarawagi. Probabilistic graphical models and their role in databases. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 1435–1436.
6. Dey D. and Sarkar S. A probabilistic relational model and algebra. ACM Trans. Database Syst., 21(3):339–369, 1996.
7. Garofalakis M.N., Brown K.P., Franklin M.J., Hellerstein J.M., Wang D.Z., Michelakis E., Tancau L., Wu E., Jeffery S.R., and Aipperspach R. Probabilistic data management for pervasive computing: The data furnace project. IEEE Data Eng. Bull., 29(1):57–63, 2006.
8. Karger D.R. A randomized fully polynomial time approximation scheme for the all terminal network reliability problem. In Proc. 27th Annual ACM Symp. on Theory of Computing, 1995, pp. 11–17.
9. Lakshmanan L.V.S., Leone N., Ross R., and Subrahmanian V.S. Probview: a flexible probabilistic database system. ACM Trans. Database Syst., 22(3):419–469, 1997.
10. Louie B., Detwiler L., Dalvi N., Shaker R., Tarczy-Hornoch P., and Suciu D. Incorporating uncertainty metrics into a general-purpose data integration system. In Proc. 19th Int. Conf. on Scientific and Statistical Database Management, 2007, pp. 19–28.
11. Re C., Dalvi N., and Suciu D. Efficient top-k query evaluation on probabilistic data. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 886–895.
12. Sen P. and Deshpande A. Representing and querying correlated tuples in probabilistic databases. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 596–605.
13. Shaker R., Mork P., Brockenbrough J.S., Donelson L., and Tarczy-Hornoch P. The biomediator system as a tool for integrating biologic databases on the web. In Proc. Workshop on Information Integration on the Web, 2004.
14. Singh S., Mayfield C., Mittal S., Prabhakar S., Hambrusch S., and Shah R. Orion 2.0: native support for uncertain data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2008, pp. 1239–1242.
15. Suciu D. and Dalvi N. Foundations of probabilistic answers to queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 963.
16. Tatusova T.A. and Madden T.L. Blast 2 sequences – a new tool for comparing protein and nucleotide sequences. FEMS Microbiol. Lett., 174:247–250, 1999.
17. Wang K., Tarczy-Hornoch P., Shaker R., Mork P., and Brinkley J. Biomediator data integration: Beyond genomics to neuroscience data. In AMIA Fall 2005 Symposium Proceedings, 2005, pp. 779–783.
18. Widom J. Trio: a system for integrated management of data, accuracy, and lineage. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005.
19. Woods D.D., Patterson E.S., Roth E.M., and Christoffersen K. Can we ever escape from data overload? a cognitive systems diagnosis. Cogn. Technol. Work, 4(1):22–36, 2002.

Undo

- ▶ Logging and Recovery

Unicode

Ethan V. Munson
University of Wisconsin-Milwaukee, Milwaukee,
WI, USA

Definition

Unicode is an international standard for representing text characters. Unicode supports the scripts of most languages in wide use and has a flexible design that is capable of supporting all known human languages and all of their variant scripts. The development of the Unicode standard is coordinated by the Unicode Consortium.

Key Points

Unicode's development is motivated by the need to encode characters in all languages without conflicts between the encodings for different languages. Obviously, the achievement of this goal is fraught with technical and political complexities.

Unicode has several different encodings. The most widely used is the 8-bit, variable-width UTF-8 encoding, which permits the encoding of many European languages in an efficient one-byte form and is backward compatible with both the ASCII and ISO-8859-1 character sets. UTF-16 is a 16-bit, variable-width encoding that is more suitable to languages with many characters, such as Chinese, Japanese and Korean. UTF-32 is a four-byte, fixed-width encoding that encompasses all Unicode characters.

The Unicode Consortium is supported by many well-known computer and software manufacturers. For the members of the consortium, the system internationalization problem is of compelling importance.

Cross-references

▶ Document
▶ Document Representations (Inclusive Native and Relational)
▶ Unified Modeling Language (UML)
▶ XML

Unified Modeling Language

Martin Gogolla
University of Bremen, Bremen, Germany

Synonyms

UML; Unified modeling language; Unified modeling language

Definition

The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components [8].

Historical Background

The UML is based on earlier software design approaches, among them the Object Modeling Technique (OMT) [9], Object-Oriented Analysis and Design (OOAD) [1], and Object-Oriented Software Engineering (OOSE) [6] and other important techniques [5]. The UML was standardized in various versions by the Object Modeling Group (OMG) [8]. Authors of the predecessor techniques have published a comprehensive description of the UML [10]. Questions strongly related to UML are discussed at the UML and MODELS conferences [13]. Special issues with extended papers from that conference and other works strongly related to UML and modeling are published in [11].

UML is mainly a graphical language and offers different diagram types. Among the most important diagram types are the class, object, statechart, activity, sequence, communication, and use case diagram. UML includes the Object Constraint Language (OCL) which is a textual language for navigation in class diagrams and for expressing textual constraints like invariants, pre- and postconditions, or guard conditions. Many tools for UML, but fewer support for OCL is available, see e.g., [3,12], among other interesting work.

Foundations

UML Class and Object Diagrams

The main purpose of class diagrams is to capture the static structures and operations of a system. This section briefly explains the most fundamental features in class diagrams: classes and associations. More advanced features are also discussed.

Classes: A class is a descriptor for a set of objects sharing the same structure and behavior. In each (database) system state, a class is manifested by a set of objects. In the database context, concentration is upon structural aspects, although behavioral aspects may be represented in UML as well. Object properties can be described by attributes classified by data types like `String` or `Boolean`. Properties can also stem from roles in associations which connect classes.

Example: Figure 1 shows the classes `Supplier`, `Project`, and `Part` together with some basic attributes including their data types, e.g., one identifies `Supplier::Name:String` and `Project::Budget:Integer`. In this contribution, the general scheme for denoting properties (attributes and roles) is `Class::Property:PropertyType`.

Associations: An association represents a connection among a collection of classes and may be given a name. An association is manifested by a set of object connections, so-called links, sharing the same structure. A binary association can be defined between two different classes; objects of the respective classes play a particular role in the association. A binary association can also be defined on a single class; then objects of the class can play two different roles; such a binary association is called reflexive. A ternary association involves three roles. The notion n-ary association refers to a ternary or a higher-order association. Binary associations are usually shown with a simple line, and an n-ary association with a small rhomb-shaped polygon.

Example: Fig. 1 shows the binary association ProjectPart with roles project and part, the ternary association SupplierProjectPart with roles supplier, suppliedProject, and suppliedPart, and the reflexive association Component with roles parent and child.

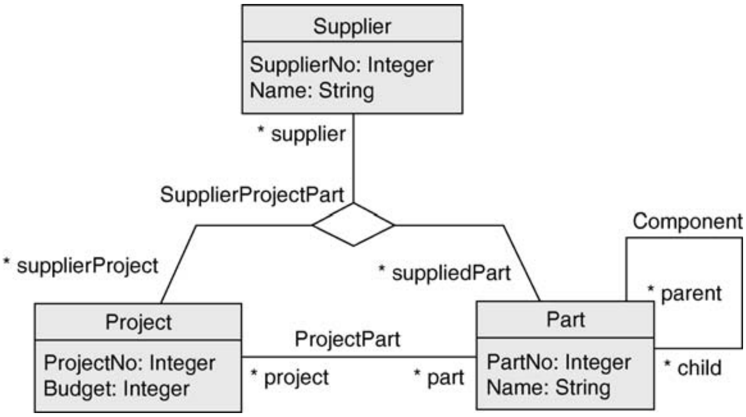
Objects and Links: Structural aspects in UML can also be represented in an object diagram showing objects, links, and attribute values as manifestations of classes, associations, and attributes. An object diagram shows an instantiation of a class diagram and represents the described system in a particular state. Underlining for objects and links is used in object diagrams in order to distinguish them clearly from class diagrams.

Example: Figure 2 shows an object diagram for the class diagram from Fig. 1. Objects, links, and attribute values correspond to the classes, associations, and attributes. There are two Project objects, two Supplier objects and five Part objects. Each Part object represents a piece of a software controller (Ctrl) being responsible for a particular portion of a car. The Component links express part-of relationships, for example, the Engine Code (engineCtrl) includes the Battery Code (batteryCtrl) and the Motor Code (motorCtrl).

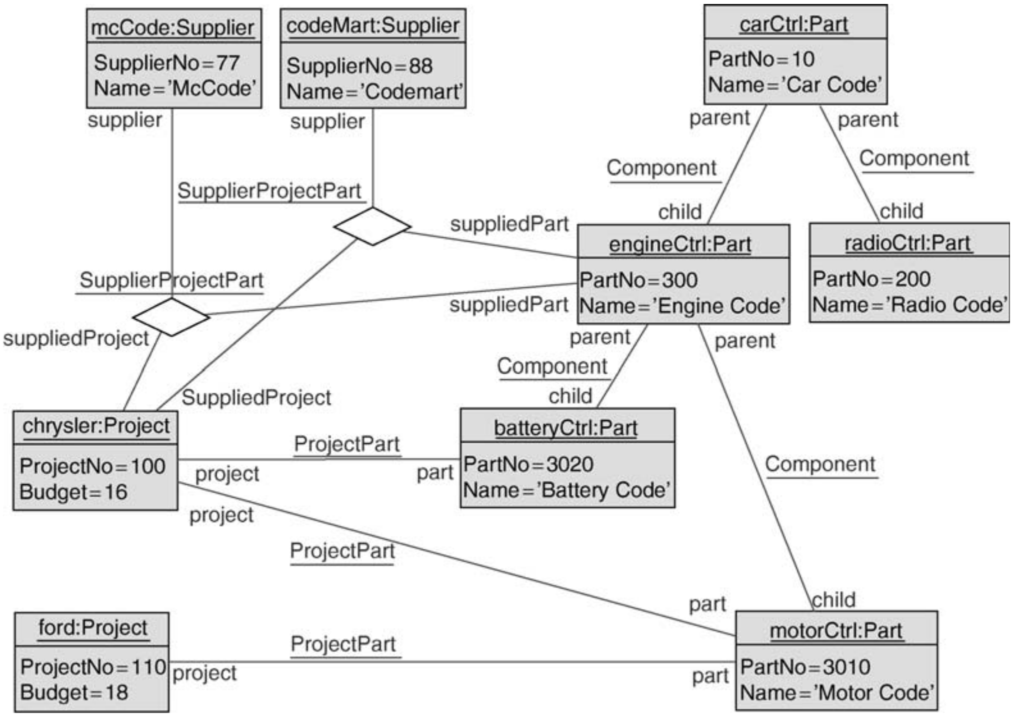
Roles: Proper roles must be specified on a class diagram in order to guarantee unique navigation. Navigation in a class diagram means to fix two classes and to consider a path from the first class to the second class by using association roles. The roles on the opposite side of a given class in an association also determine properties of the given class by navigating via the roles. Therefore, in UML the opposite side roles must be unique. Recall that properties can also come from attributes.

Example: Roles are captured on links. This is necessary in reflexive associations and in other situations, for example, if two associations are present between two given classes. For example in Fig. 2, if one considers the link between carCtrl and engineCrtl, without roles one could not tell which object plays the parent role and which one the child role. In the class diagram in Fig. 1, the class Project has two direct navigation possibilities to class Part: One via association ProjectPart and the other one via association SupplierProjectPart. One obtains therefore two properties of class Project returning Part objects: Project::part:Set(Part) from association ProjectPart and Project::suppliedPart: Set(Part) from association SupplierProjectPart. In the object diagram one obtains, for example, ford.part = Set{motorCtrl} as well as ford.suppliedPart = Set{}. In order to distinguish between these two navigations, the role name part and suppliedPart must be distinct.

Class Diagram versus Database Schema: In the database context, it is interesting to remark that the connection between a class diagram and its object diagrams resembles the connection between a database schema and its associated database states. The class diagram in general induces a set of object diagrams and the database schema determines a set of database states. Object diagrams and database states follow the general principles formulated in the class diagram and database schema, respectively. Because example object diagrams have to be displayed on a screen or on paper,



Unified Modeling Language. Figure 1. Example UML class diagram 1.



Unified Modeling Language. Figure 2. Example object diagram 1.

they tend to show less information than proper, large database states. They may however explain the principles underlying a class diagram pretty well if the examples are well chosen.

UML Class Diagram Features for Conceptual Schemas

Some of the more advanced features of conceptual database schemas in UML class diagrams are: object-valued, collection-valued and compound attributes, role multiplicities, association classes, generalizations, aggregations, compositions, and invariants.

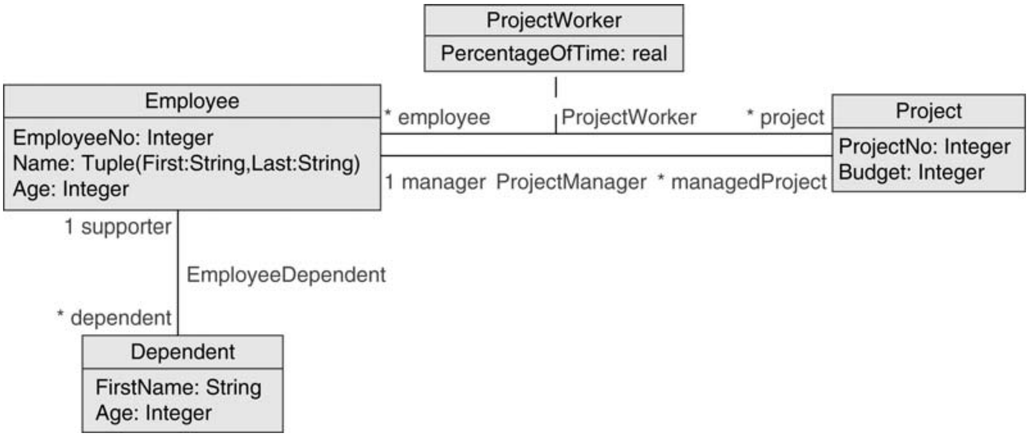
Object-Valued Attributes: Attributes in UML may not only be data-valued as above, but the attribute type may be a class as well which leads to object-valued attributes. Like associations, object-valued attributes also establish a connection between classes. However, an object-valued attribute is only available in the class in which it is defined. The information from that attribute is not directly present in the attribute type class. Thus an object-valued attribute may be regarded as a unidirectional association without an explicit name and where only one role is available.

Examples: The class diagram in Fig. 3 extends the class diagram in Fig. 1 by introducing the new classes `Employee`, `Dependent`, and `ProjectWorker`, and the associations `EmployeeDependent`, `ProjectManager`, and `ProjectWorker`. The fact that `ProjectWorker` is mentioned as a class as well as an association will be explained below. The object diagram in Fig. 4 shows an example state for the class diagrams from Fig. 3.

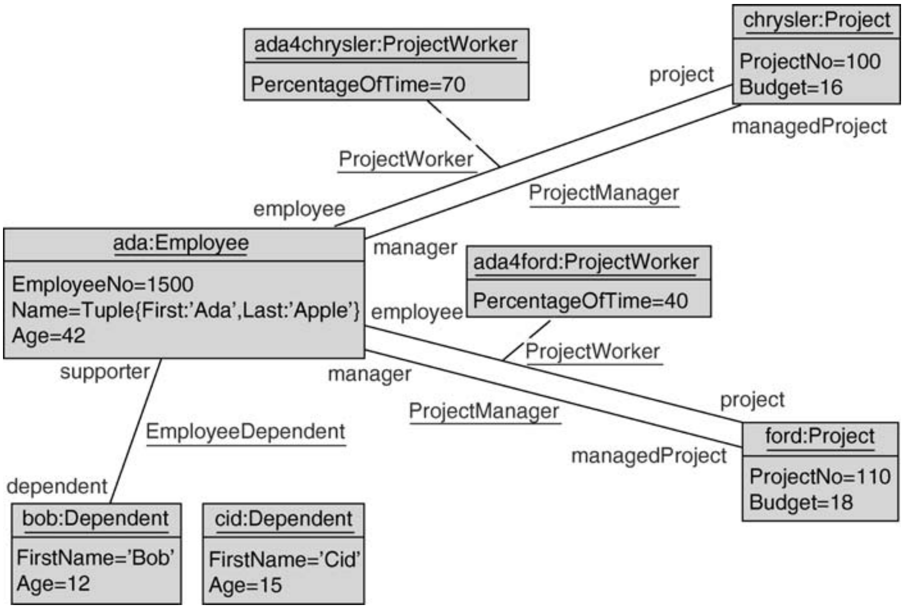
As an example for an object-valued attribute and as an alternative for the association `ProjectManager`, one could extend the class `Project` by an attribute `manager` with type `Employee`. This could be represented altogether as `Project::manager: Employee`.

Collection-Valued Attributes: The collection kinds set, bag, and sequence have already been introduced. These collection kinds can be used as type constructors on data types and classes. For building attribute types, the constructors may be nested.

Examples: An attribute could possess a type like `Set(Project)`. As an alternative for the association `ProjectManager` one could have one attribute `managedProject:Set(Project)` in the class `Employee` and another attribute `manager:Employee` in class `Project`. There is however a significant difference between the model with the association `ProjectManager` including the roles `manager` and



Unified Modeling Language. Figure 3. Example UML class diagram 2.



Unified Modeling Language. Figure 4. Example UML object diagram 2.

`managedProject` and the model with the two attributes `manager` and `managedProject`. In the model with the association, the roles `managedProject` and `manager` represent the same set of object connections, i.e., are inverse to each other:

```
    Employee.allInstances->forAll(e|e.
managedProject->forAll(p|p.manager=e))
Project.allInstances->forAll(p|p.manag-
er.managedProject->includes(p))
```

This is not required to hold in the model possessing the two attributes. In this case the two attributes `managedProject` and `manager` are independent from each other and may represent different sets of object connections.

Another useful application of collection-valued types are collections over the data types like the complex type `Set(Sequence(String))`. A value for an attribute typed in this way could be, for example, the complex value `Set{Sequence{'Rome','Euro'}, Sequence{'Tokyo','Yen'}}`.

Compound Attributes: Apart from using the collection constructors `Set`, `Bag`, and `Sequence` for attributes, one can employ a tuple constructor `Tuple`. A tuple has a set of components each possessing a component discriminator and a component type. The collection constructors and the tuple constructor may be nested in an orthogonal way.

Examples: The above value for the type Set(Sequence(String)) could be represented also with type Set(Tuple(Town:String,Currency:String)) and with the corresponding value Set{Tuple {Town:'Rome', Currency:'Euro'}, Tuple {Town:'Tokyo', Currency:'Yen'}}.

As a further example of a compound attribute using the Tuple constructor, one identifies in the class diagram in Fig. 3 the attribute Name in class Employee which is a compound attribute with type Tuple(First:String, Last:String).

Role Multiplicities: Associations may be restricted by specifying multiplicities. In a binary association, the multiplicity on the other side of a given class restricts the number of objects of the other class to which a given object may be connected. In a simple form, the multiplicity is given as an integer interval low..high (with low ≤high) which expresses that every object of the given class must be connected to at least low objects and at most high objects of the opposite class. The high specification may be given as * indicating no higher bound. A single integer i denotes the interval i..i and * is short for 0..*. The multiplicity specification may consist of more than one interval.

Examples: The multiplicity 1 on the role supporter indicates that an object of class Dependent must be linked to exactly one object of class Employee via the association EmployeeDependent.

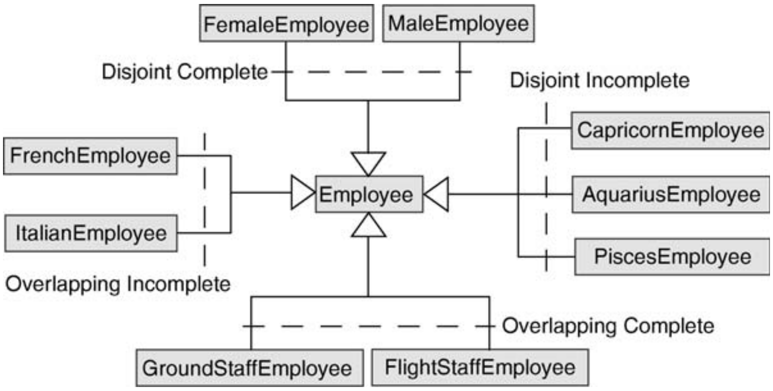
Association Classes: Associations may be viewed again as classes leading to the concept of an association class. Association classes are shown with a class rectangle and are connected to the association (represented by a line or a rhomb) with a dashed line. Association classes open the possibility of assigning attributes to associations.

Examples: The association ProjectWorker is modeled also as a class: ProjectWorker is an association class. This makes it possible to assign the attribute PercentageOfTime to the association ProjectWorker. In the class diagram, ProjectWorker is redundant as both the class name and the association name; the specification as the class name would be sufficient.

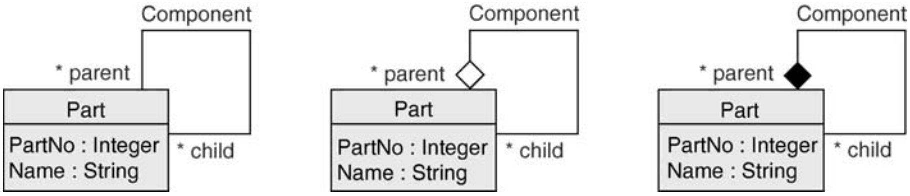
Generalizations: Generalizations are represented in UML with directed lines having an unfilled small triangle pointing to the more general class. Usually the more specific class inherits the properties from the more general class. Generalizations are known in the database context also as ISA (IS-A) hierarchies. In the programming language context often the notion of inheritance shows up. Viewed from the more general class its more specific classes are its specializations. In general, a class may have many specializations, and a class may have many generalizations. A set of generalizations may be restricted to be disjoint and a set of generalizations may be classified as complete. The classification disjoint means that any two specialized classes are not allowed to have a common instance. The label complete means that every instance of the general class is also an instance of at least one more specialized class. The explicit keywords overlapping and incomplete may be attached to sets of generalizations for which no respective restriction is made.

Examples: Figure 5 shows different specializations of the class Employee. The subclasses FemaleEmployee and MaleEmployee represent a disjoint and complete classification. The subclasses CapricornEmployee, AquariusEmployee, and PiscesEmployee classify employees according to their birthday (December 22–January 20, January 21–February 19, February 20–March 20, respectively). This classification is disjoint but incomplete. The subclasses GroundStaffEmployee and FlightStaffEmployeein the context of an airline company are labeled overlapping and complete, because each airline employee either works on the ground or during a flight and, for example, a flight accident is allowed to work on the ground during boarding and of course during the flight. The subclasses FrenchEmployee and ItalianEmployee are overlapping because employees may have two citizenships, but it is incomplete because, e.g., Swiss employees are not taken into account.

Aggregations: Part-whole relationships are available in UML class diagrams in two forms [6]. The first form represents a loose binding between the part and the whole, the second form realizes a stronger binding. Both forms can be understood as binary associations with additional restrictions. The first form called aggregation is drawn with a hollow rhomb on the whole side and is often called white diamond. The second form called composition is drawn with a filled rhomb on the whole side and is often called black



Unified Modeling Language. Figure 5. Different example generalizations and specializations in UML.



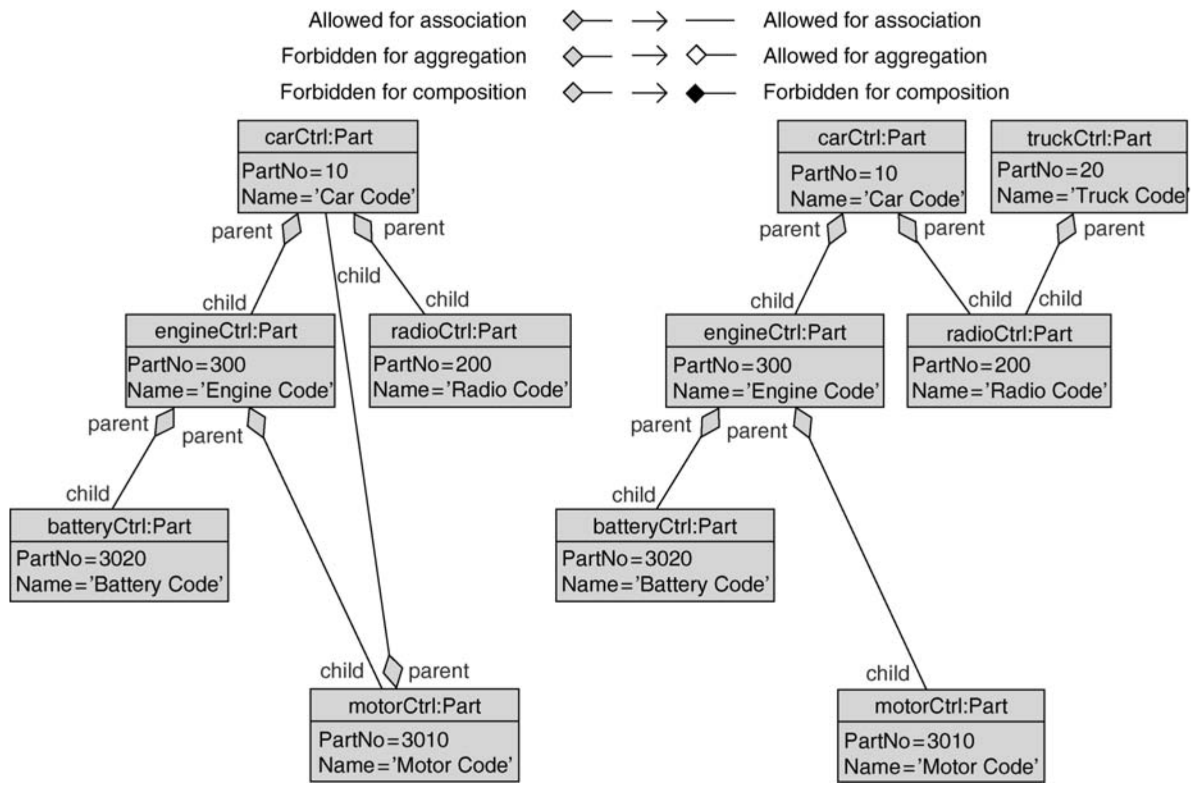
Unified Modeling Language. Figure 6. Component as association, aggregation, and composition.

diamond. The links in an object diagram belonging to a class diagram with a part-whole relationship must be acyclic if one regards the links as directed edges going from the whole to the part. This embodies the idea that no part can include itself as a subpart. Such cyclic links are allowed however for arbitrary associations. Part objects from an aggregation are allowed to be shared by two whole objects whereas this is forbidden for composition.

Examples: The class diagrams in Fig. 6 show on the left the association `Component` already introduced and on the right two alternatives in which the association is classified as an aggregation with a white diamond and as a composition with a black diamond. Recall that roles are essential in reflexive associations and therefore in reflexive part-whole relationships. Here the `parent` objects play the whole role and the `child` objects play the part role. The two object diagrams in Fig. 7 explain the differences between association, aggregation, and composition. The diamonds are shown as grey diamonds, a symbol which does *not* exist in the UML. If the grey diamond is substituted by a white diamond, the left object diagram is forbidden, because there is a cycle in the part-whole links which would mean that the object `carCtrl` is a part of itself. This would also hold for the other two objects on the cycle. Recall that if one would have a simple association instead of the grey diamond, this object diagram would be allowed. If the grey diamond is replaced by a white diamond, the right object diagram is an allowed object diagram. Here, the object `radioCtrl` is shared by the objects `carCtrl` and `truckCtrl`. Naturally, if the grey diamond would become an association, the right object diagram is allowed as well.

Compositions: Compositions pose further restrictions on the possible links in addition to the required acyclicity. Part objects from a composition cannot be shared by two whole objects. The table in Fig. 8 gives an overview on the properties of associations, aggregations, and compositions.

Examples: The discussion now turns to what happens in Fig. 7 if the grey diamond is substituted by a black diamond in order to represent compositions. If the grey diamond is replaced by a black diamond, the left object diagram is again forbidden, because there is a cycle in the part-whole links. If the grey diamond is replaced by a black diamond, the right object diagram is a forbidden object diagram for compositions, because sharing of objects is not allowed in that case.



Unified Modeling Language. Figure 7. Forbidden and allowed object diagrams for aggregation and composition.

	Acyclicity	Prohibition of sharing
Association	–	–
Aggregation	+	–
Composition	+	+

Unified Modeling Language. Figure 8. Overview on properties of associations, aggregations and compositions.

To show also a positive example for composition and aggregation, if the link from `motorCtrl` to `carCtrl` is removed in the left object diagram, it is a valid object diagram for compositions and aggregations.

Data Types And Enumeration Types: UML offers a collection of predefined data types with the usual operations on them. The data types include `Integer`, `Real`, `String`, and `Boolean`. Application dependent enumeration types can also be defined in a class diagram. The enumeration type name is followed by the list of allowed enumeration literals. Enumeration types can be used as attribute, operation-parameter or operation-return types.

Examples: Figure 9 shows two enumeration types useful in the context of the running example. The type `Gender` may represent the gender of an employee and the type `CivilStatus` its civil status.

Invariants: OCL allows invariants to be specified, i.e., conditions which must be true during the complete lifetime of an object (or perhaps more precisely, at least, in moments when no activity in the object takes place). Such invariants are implicitly or explicitly universally quantified OCL formulas introduced with the keyword `context`.

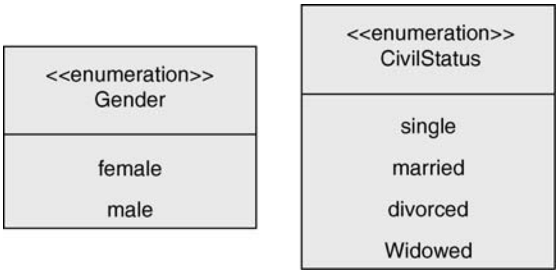
Example: In order to require that employees are atleast 18 years of age, one could state the following invariant.

context Employee inv EmployeeAreAtLeast18: Age>=18

This constraint has an implicit variable `self` of type `Employee` and is equivalent to:

context self:Employee inv EmployeeAreAtLeast18: self.Age>=18

Instead of `self` one could have used any other name for the variable, e.g., the variable `e`. The invariant



Unified Modeling Language. Figure 9. Enumerations in UML.

corresponds to the following OCL formula which must be true in all system states.

Employee.allInstances->forAll(selflself.age>=18)

Key Applications

UML class diagrams are widely used to describe structural aspects for databases (see [7,14], among other relevant papers). OCL constraints are applied for the specification of database integrity. OCL may also be regarded as an object-oriented navigation and query language.

Cross-references

▶ Data Model
▶ ER Model
▶ Extended Entity-Relatioinship Model
▶ Object Constraint Language
▶ Specialization and Generalization

Recommended Reading

1. Booch G. Object-Oriented Design with Applications. Benjamin-Cummings, Menlo Park, CA, 1991.
2. Chen P.P. The Entity-Relationship Model – Toward a Unified View of Data. ACM Trans. Database Syst., 1(1):9–36, 1976.
3. Gogolla M., Büttner F., and Richters M. USE: A UML-based Specification Environment for Validating UML and OCL. Sci. Comput. Program., 69:27–34, 2007.
4. Gogolla M. and Richters M. Expressing UML class diagrams properties with OCL. In Advances in Object Modelling with the OCL, T. Clark, J. Warmer (eds.). Springer, Berlin Heidelberg New York, 2001, pp. 86–115.
5. Harel D. Statecharts: a visual formalism for complex systems. Sci. Comput. Program., 8(3):231–274, 1987.
6. Jacobson I., Christenson M., Jonsson P., and Oevergaard G. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, Reading, MA, USA, 1992.
7. Marcos E., Vela B., and Cavero J.M. A methodological approach for object-relational database design using UML. Software Syst. Model., 2(1):59–75, 2003.
8. OMG (ed.). OMG Unified Modeling Language Specification. OMG, 2007. www.omg.org.
9. Rumbaugh J., Blaha M., Premerlani W., Eddy F., and Lorensen W. Object-Oriented Modeling and Design. Prentice-Hall, Englewood Cliffs (NJ), 1991.
10. Rumbaugh J., Booch G., and Jacobson I. The Unified Modeling Language Reference Manual, 2nd edn. Addison-Wesley, Reading, MA, USA, 2005.
11. SOSYM Editorial Board. Software and Systems Modeling. Springer, Berlin Heidelberg New York, 2007.
12. Toval J.A., Requena V., and Fernandez J.L. Emerging OCL tools. Software Syst. Model., 2(4):248–261, 2003.
13. UML and MODELS Steering Committee. International ACM/IEEE Conference on Model Driven Engineering Languages and Systems MODELS (previously ACM/IEEE International Conference on the Unified Modeling Language UML). http://www.modelsconference.org/.
14. Urban S.D. and Dietrich S.W. Using UML class diagrams for a comparative analysis of relational, object-oriented, and object-relational database mappings. In Proc. of 34th SIGCSE Technical Symp. on Computer Science Education, 2003, pp. 21–25.

Uniform Resource Identifier

▶ Resource Identifier

Union

Cristina Sirangelo
University of Edinburgh, Edinburgh, UK

Synonyms

Union

Definition

The union of two relation instances R_1 and R_2 over the same set of attributes U – denoted by $R_1 \cup R_2$ – is another relation instance over U containing precisely the set of tuples t such that $t \in R_1$ or $t \in R_2$.

Key Points

The union is one of the primitive operators of the relational algebra. It is a natural extension of the set union to relations; the additional restriction is that it can be applied only to relations over the same set of attributes. However the union of two arbitrary relations having the same arity can be obtained by first renaming the attributes of one of the two relations.

As an example, consider a relation *Students* over attributes (*number, name*), containing tuples {(1001, *Black*), (1002, *White*)}, and a relation *Employees* over attributes (*number, name*), containing tuples {(1001, *Black*), (1003, *Brown*)}. Then the union *Students* ∪ *Employees* is a relation over attributes (*number, name*) with tuples {(1001, *Black*), (1002, *White*), (1003, *Brown*)}.

In the absence of attribute names, the union is defined on two relations with the same arity. The output is a relation with the same arity as the input, containing the union of the sets of tuples in the two input relations.

The union operator is commutative and associative. The number of tuples in the output relation is bounded by the sum of the number of tuples in the input relations.

Cross-references

▶ Relation
▶ Relational Algebra
▶ Renaming

Uniqueness Constraint

▶ Key

Unnoticability

▶ Unobservability

Unobservability

Simone Fischer-Hübner
Karlstad University, Karlstad, Sweden

Synonyms

Unnoticability

Definition

Unobservability ensures that a user may use a resource or service without others, especially third parties, being able to observe that the resource or service is being used [1].

A corresponding, but more general definition is provided by [2]. Unobservability of an item of interest (e.g., a subject, messages, action) means that all uninvolved subjects cannot sufficiently distinguish whether or not it exists. Besides, anonymity of subjects involved in the item of interest is provided even against the other subjects involved in that item of interest.

Key Points

Whereas anonymity and pseudonymity protect the relationship of subjects to other items of interest (e.g., the fact that a specific user has sent a message), unobservability protects information about the very existence of the item of interest against uninvolved parties (e.g., the fact that a message was sent). With respect to the same attacker, if a subject's action is unobservable, then the user is also anonymous (see also [2]).

Cross-references

▶ Anonymity
▶ Pseudonymity
▶ Privacy
▶ Privacy-Enhancing Technologies

Recommended Reading

1. Common Criteria Project, Common Criteria for Information Technology Security Evaluation, Version 3.1, Part 2: Security Functional Requirements, www.commoncriteriaportal.org, September, 2006.
2. Pfitzmann A. and Hansen M. Anonymity, Unlinkability, Unobservability, Pseudonymity, and Identity Management – A Consolidated Proposal for Terminology, Version 0.29, available at: http://dud.inf.tu-dresden.de/Anon_Terminology.shtml (accessed on July, 2007).

Unsupervised Learning

▶ Cluster and Distance Measure
▶ Clustering Overview and Applications

Unsupervised Learning on Document Datasets

▶ Document Clustering

Until Changed

▶ Now in Temporal Databases

Update Propagation in Peer-to-Peer Systems

▶ Updates and Transactions in Peer-to-Peer Systems

Updates and Transactions in Peer-to-Peer Systems

Zachary Ives
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

Consistency in peer-to-peer systems; Update propagation in peer-to-peer systems

Definition

In recent years, work on peer-to-peer systems has started to consider settings in which data is updated, sometimes in the form of atomic transactions, and sometimes by parties other than the original author. This raises many of the issues related to enforcing consistency using concurrency control or other schemes. While such issues have been addressed in many ways in distributed systems and distributed databases, the challenge in the peer-to-peer context is in performing the tasks cooperatively, and potentially in tolerating some variation among the instances at different nodes.

Historical Background

Early peer-to-peer systems focused on sharing or querying immutable data and/or files. More modern uses of peer-to-peer technology consider settings in which dynamic data and updates are being made in a distributed, autonomous context. A question of significant interest is how to define consistency in this model, while still allowing at least some autonomy among the sites.

Foundations

The database community has only recently begun to consider peer-to-peer architectures for applications beyond query answering, with recent work in peer data exchange [10] and collaborative data sharing [16]. However, a popular model has been that of *distributed stream processing*, in which streams of data (usually interpreted as insertions of new data) are processed in a distributed or peer-to-peer network [1,7]. Additionally, work in the distributed systems community, particularly on file systems, must consider many issues related to distributed replication and consistency with the granularity of updates typically being at the level of the file, or custom to an application. (See the entry on *peer-to-peer storage.*)

The work on updates and transactions in peer-to-peer systems can be classified based on who is allowed to modify it, and how conflicting modifications are resolved. This can be divided into the following categories:

Single Owner/Primary Copy is a setting in which each data item that originates from some source peer *p* can *only* be modified by (or through) *p* – i.e., no other peers are allowed to directly modify that data.

Owner-Resolver protocols allow multiple peers to modify the data, and they typically rely on the owner to resolve any conflicts. If resolution is impossible, they "branch" the data into fully independent instances.

Consensus protocols allow multiple peers to modify the data, and some set of nodes works together to determine how to arbitrate for consistency.

Partial Divergence schemes handle conflicts in a way that results in multiple divergent copies of the data, but they operate at a finer level of granularity than divergent replica protocols, and they allow some portions of the data instance to remain shared even after "branching" two instances.

The remainder of this entry provides more detail on the different approaches and their common implementations.

Single-Owner/Primary Copy

In the simplest schemes, each data item is owned by a single source, which may update that data. Many other nodes may replicate the data but may not change it (except, perhaps, by going through the *primary copy* at the owner). This is sometimes referred to as the *single-writer, multiple readers* problem. In this type of scheme, the owner of the data uses a timestamp (logical or physical) to preserve the serial order of updates, or to arbitrate among different verions of the data. Since there is a single owner and a single clock, any node can look at the data and deterministically choose an ordering.

In the database world, peer-to-peer stream processing or filtering systems, e.g., ONYX [7] and PIER [11], can be considered systems in which each data item (tuple or XML tree) has a single owner. When new data arrives on the stream, this appends to or replaces the previous data items from the same source.

P2P file systems have also employed a single-owner scheme that provides very interesting properties. In particular, CFS [5], PAST [8], and similar systems build a filesystem layer over distributed hash tables (DHash and Pastry, respectively), where every version of the file is maintained in the distributed hash table. In these approaches, a user must ask for a file by name *and version*; the latest version can be obtained from the file's owner or from a trusted third party. This model is exemplified by CFS, which retrieves the file as follows: first, it cryptographically hashes the filename and version, receiving a key from which the file's current block map can be fetched. This map contains an ordered list of keys corresponding to the individual blocks in the requested version of the file. Each block can be fetched using its key; the key actually represents the hash of the block's content. This content-based hashing scheme for blocks has a two key benefits: (i) if two files or file versions share a page with identical content, CFS will store only *one* copy of the page and will use it in both files; (ii) CFS will employ caching as blocks are requested, and the cache lookup mechanism can be based on hashing, rather than naming, content.

Owner-Resolver

Coda [12] relaxes the single-owner scheme described above, in allowing data to be replicated throughout a network, and for changes to be made to the replicas. Coda's focus is on allowing updates in the presence of network partition: nodes might need to make changes without having access to the primary copy. Once connectivity is restored, the newly modified replica must be reconciled with the original data and any other changed replicas; Coda does this by sharing and replaying logs of changes made to the different replicas. If Coda determines that multiple concurrent changes were made, then activates an application-specific *conflict resolver* that attempts to resolve the conflicts. In the worst case, the data may need to be branched.

Bayou [9] uses a very similar scheme, except that changes to replicas are propagated in pairwise fashion across the network (an *epidemic protocol*) instead of sent directly to each file's owner. Nodes maintain logs of the updates they know about, including other nodes' updates; as they communicate, they exchange logs and merge them in pairwise fashion (this is an *epidemic protocol*). If conflicts occur, an application-specific *merge procedure* is triggered. Eventually the logs reach a primary node, which determines the final order of updates.

Building even further on the notion of epidemic protocols, work by Datta et al. [6] focuses on settings in which conflicts are unlikely to arise at all. It adopts an epidemic protocol that provides eventual consistency across the network. In this model, a peer that makes an update pushes a notification to subset of its neighboring peers, who may in turn forward to additional peers. This is likely to keep many of the peers "relatively" up to date. When a peer has not received an update in a while, or if it comes back online, it tries to determine the latest state by executing a "pull" request.

Consensus

Coda and Bayou allowed for concurrent updates, but relied on the holder of the primary copy to resolve conflicts. An alternative is to reconcile conflicts through some sort of voting or consensus scheme. Like Bayou, Deno [4] uses an epidemic protocol, in which nodes share information in pairwise fashion about new updates. Here, updates are grouped into transactions that are to be atomically committed or aborted. For each transaction that is not *blocked* (i.e., waiting for another transaction to complete), a distributed vote is executed to determine whether the transaction should be committed. Nodes gossip by exchanging information about transactions and votes; ordering information is maintained using version vectors [15]. If a majority of nodes vote for the transaction, rather than any other transactions that have conflicting updates, then the transaction is committed.

A number of filesystems, including BFS [3], OceanStore [13], Farsite [2], make use of *quorums* of nodes that manage the sequencing on updates: these nodes essentially serve very similarly to the single owner schemes described previously, in that they must be contacted for each update in sequence, and they define the serialization order of updates.

Partial Divergence

Two more recent works – one focused on filesystems, and the other on database instances – enable a scheme for managing inconsistency based on peers' individual *trust policies*.

Like CFS, Ivy [14] is a filesystem built over the DHash distributed hash table. However, Ivy provides NFS-like semantics including the ability to modify files, and it does so in a novel way. Ivy has one update log per peer, and such logs are made publicly available through DHash. As a peer modifies a file, it writes these changes to its own update log (annotating them with version vectors [15] so sequencing can be tracked). As the peer *reads* a file, it may consult *all* logs; but it may also ignore some logs, depending on its local trust policy. However, Ivy assumes that it is highly undesirable for each peer to get a different version of a file (because this would prevent sharing); hence, sets of peers share a *view* of the file system – all files in the view have the same consistent version.

Finally, the Orchestra [16] *collaborative data sharing system* focuses on sharing different database instances in loose collaborations: here, multiple peers wish to share updates with one another, but each peer may selectively override the updates it receives from elsewhere. Each peer may adopt its own *trust policies* specifying (in a partial order) how much it trusts the other peers. Orchestra is specifically motivated by scientific data sharing: for instance, organizations holding proteomics and genomics data wish to import data from one another, and to refresh this imported data; however, since the data is often unreliable, each peer may wish to choose the version from the site it trusts most, and then independently curate (revise, correct, and annotate) the resulting data.

In Orchestra, as in Deno, all updates are grouped into atomic transactions. Here, each peer uses its local trust policies to choose among conflicting transactions – such a transaction will be *accepted* by the peer, and all conflicting transactions will be *rejected*. If a transaction *X* from a particular peer is rejected, then all subsequent transactions from that same peer that directly modified the results of *X* are also transitively rejected. In this trust model, the common case is that transactions come to a target peer *C* from a trusted peer and does not conflict with any others; hence they are immediately applied to *C*. Otherwise, trust composes as follows: as a transaction is propagated from peer *A* to peer *B* to peer *C*, it is only applied at *C* if it is the most trusted transaction at each step; if multiple conflicting transactions can be applied at *C*, then the one most trusted by *C* is applied. Orchestra allows instances to "partially diverge" where there are conflicts, while still maintaining sharing for portions of the data where there are no conflicts.

Key Applications

Recent applications of peer-to-peer technologies include distributed network monitoring, distributed stream processing, and exchange of data among collaborating organizations that host scientific databases. In all of these settings, data may be frequently updated.

Future Directions

One of the most promising directions of future study is the interaction between transactions and *mappings* or conversion routines: in many real-world applications, data is being shared between sites that have different data representations. There is some work (e.g., that related to view maintenance) that shows how to translate updates across mappings; but there is no scheme for mapping *transactions*.

Cross-references

- ▶ Distributed Concurrency Control
- ▶ Distributed Databases
- ▶ Epidemic Algorithms
- ▶ Peer-to-Peer Storage
- ▶ Transaction

Recommended Reading

1. Balazinska M., Balakrishnan H., and Stonebraker M. Demonstration: Load management and high availability in the Medusa distributed stream processing system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
2. Bolosky W.J., Douceur J.R., Ely D., and Theimer M. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In Proc. 2000 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Comp. Syst., 2000.
3. Castro M. and Liskov B. Practical Byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst., 20(4): 2002.
4. Cetintemel U., Keleher P.J., Bhattacharjee B., and Franklin M.J. Deno: a decentralized, peer-to-peer object-replication system for weakly connected environments. IEEE Trans. Comput., 52(7):943–959, Jul 2003.
5. Dabek F., Kaashoek M.F., Karger D., Morris R., and Stoica I. Wide-area cooperative storage with CFS. In Proc. 18th ACM Symp. on Operating System Principles, 2001.
6. Datta A., Hauswirth M., and Aberer K. Updates in highly unreliable, replicated peer-to-peer systems. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2003.
7. Diao Y., Rizvi S., and Franklin M.J. Towards an Internet-Scale XML Dissemination Service. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.

8. Druschel P. and Rowstron A. PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility. In Proc. 8th Workshop on Hot Topics in Operating Systems, 2001.
9. Edwards W.K., Mynatt E.D., Petersen K., Spreitzer M.J., Terry D.B., and Theimer M.M. Designing and implementing asynchronous collaborative applications with Bayou. In Proc. 10th Annual ACM Symp. on User Interface Software and Technology, 1997.
10. Fuxman A., Kolaitis P.G., Miller R.J., and Tan W.C. Peer data exchange. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2005.
11. Huebsch R., Hellerstein J.M., Lanham N., Loo B.T., Shenker S., and Stoica I. Quering the Internet with PIER. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003.
12. Kisler J. and Satyanarayanan M. Disconnected operation in the coda file system. ACM Trans. Comput. Syst., 10(1), 1992.
13. Kubiatowicz J., Bindel D., Chen Y., Czerwinski S., Eaton P., Geels D., Gummadi R., Rhea S., Weatherspoon H., Weimer W., Wells C., and Zhao B. OceanStore: an architecture for global-scale persistent storage. In Proc. 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 2000.
14. Muthitacharoen A., Morris R., Gil T.M., and Chen B. Ivy: A Read/Write Peer-to-Peer File System. In Proc. 5th USENIX Symp. on Operating System Design and Implementation, 2002.
15. Parker Jr. D.S., Popek G.J., Rudisin G., Stoughton A., Walker B.J., Walton E., Chow J.M., Edwards D.A., Kiser S., and Kline C.S. Detection of mutual inconsistency in distributed systems. IEEE Trans. Software Eng., 9(3), 1983.
16. Taylor N.E. and Ives Z.G. Reconciling while tolerating disagreement in collaborative data sharing. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006.

Updates through Views

YANNIS VELEGRAKIS
University of Trento, Trento, Italy

Definition

Views are windows to a database. They provide access to only a portion of the data that is either of interest or related to an application. Views are relations without independent existence, as is the case of database tables created with the "create table" command. The contents of the instance of a view are determined by the result of a query on a set of database tables, typically referred to as *base tables*. Applications deal with views the same way they deal with any other relation. In fact, an application is rarely aware of whether a relation it is accessing is a base table or a view. This means that the application may issue updates on the view relation as it would have done with any other database table. Since the view instance depends on the instances of the base tables, to execute an update on the view one needs to find a number of base table modifications whose effect on the view instance is the modification described by the update command.

Historical Background

Views are one of the oldest concepts in computer science. They appeared almost at the same time with queries. In the early 1980s, as database vendors were moving towards the relational model, Codd introduced twelve rules that need to be satisfied by a database management system in order to be considered *relational*. Rule number six was referring to the concept of views and in particular, to the ability of the views to be updated when they are *theoretically updatable*, a concept that will be explained below.

The scientific literature contains different definitions (and uses) of the term "view," with two of them being dominant. The first is that a view is a relation whose instance depends (somehow) on the data of its base tables, and the second is that a view is a short-hand for a query. For query answering, those two definitions are equivalent, but for the purpose of updating, the definition that is considered makes a difference. They have different consequences. Considering the view as a query, implies that at any given point in time the instance of a view is fully specified by the results of the view query over the data, i.e., the set of possible instances of a view is the set of possible relations that can be generated as a result of the view query. A consequence of this is that the view might not be *theoretically* updatable, which means that there are updates on the view that cannot be translated to updates on the base tables. Recently, Kotidis et al. [8] have relaxed that requirement to accept views whose instance may be different from the result of their view query in order to allow arbitrary updates on the view.

Update propagation is one of the main issues in view updates. When multiple views share the same piece of data from the base tables, all these views should have consistent instances. This means that when some part of the data is updated, this change must propagate accordingly to the base tables and the other views. For the case of materialized views, and updates on the base tables, this is a well-studied problem, namely the view maintenance [10]. View maintenance has considered how to change the instances of the views in response to base table modifications.

However, updates on the views have not been studied to the same extend.

Foundations

A *database* D = <S, I> is a collection of data along with a description of its structure. The collection of data is referred to as the *database instance* I and the description of its data structures as the *database schema* S. In the relational model, a database instance is a set of relations, and the schema is the set of schemas of these relations. The contents of a relation, i.e., its instance, may be explicitly specified by a user or an application, or they may be derived from the contents of other relations. Relations of the first kind are those constructed using the "create table" command and are often referred to as *base tables*. Relations of the second kind are referred to as *views*. A view is accompanied by a query which is referred to as the *view query* and specifies how its contents will be derived from the contents of the base tables.

The term "update" on a relation implies an insertion, a deletion, or a value modification of one or more tuples. The literature often chooses to ignore modifications on the basis that a modification can be modeled as a deletion followed by an insertion. The same assumption is followed here.

Let U denote an update command on a relation R, |R| the instance of R and |U| the tuple(s) that need to be inserted in |R| (if U is an insert command) or deleted from |R| (if U is a delete command).

The *implementation* of an update U on a relation R is a new relation R′ such that:

- $|R|' = |R| - |U|$ if U is a delete command, and
- $|R|' = |U| \cup |R|$ if U is an insert command

In the case of a base table, the implementation of an update is straight forward (it is done by directly modifying the contents of the base table instance according to the update command). In the case of a view the situation is different. Since the view instance depends on the instances of its base tables an implementation is performed by translating the view update command into a series of updates on the base tables so that the requested update is observed in the view instance. Figure 1 provides a graphical representation of this notion. $Q_v(I)$ denotes the application of the view query Q_v on the database instance I. The result of this query is the instance |V| of the view V. Given an update command U on the view V, the *view update problem* is defined as the problem of finding an update W, referred to as the *translation* of U, on the instance I such that when the view query is applied on the new modified instance W(I), the new view instance $|V|' = Q_v(W(I))$ is an implementation of the update U.

For a given update U, there may be multiple possible translations. The following example illustrates such a situation. Consider a database instance that consists of a table *Personnel (Department, Employee)* with tuples {[Administration, Smith], [HR, Smith], [Research, Kole]} and the view

V_1: *select* Department *from* Personnel *where Employee* = "Smith"

The instance of view V_1 consists of the two tuples [Administration, Smith] and [HR, Smith]. Let U_1 be an update command that requests the deletion of tuple [Administration, Smith] from V_1. There are many possible changes that can be done on the instance of *Personnel* in order to make tuple [Administration, Smith] disappear from the instance of view V_1. The most obvious one is to simply delete tuple [Administration, Smith] from Personnel. Alternatively, one could update the value of its *Employee* attribute from "Smith" to *null* or to some value other than "Smith." Even changing the *Department* attribute from "Administration" to some other value "Y" will have the desired effect of removing tuple [Administration, Smith] from the view. Thus, each of the aforementioned changes constitutes an implementation of the view update U_1. They are not, however, equivalent. Each translation may be considered appropriate in different situations. The first translation, for instance, achieves the desired result and only this by deleting only one tuple from the database instance. The second achieves the result without reducing the number of tuples in the database instance, and the latter will cause the appearance of the additional tuple [Y, Smith] in the view, which makes the specific translation less appealing.

For the case of a view deletion, the first step in finding a view update translation is to find the provenance of the tuples that are to be deleted from the view, i.e., the tuples in the base tables that are responsible for the appearance in the view of the tuples that are to be deleted [4,5]. Researchers have concentrated their efforts in detecting the minimum number of changes that need to be made on the base tables in order to achieve the view update (*source-side-effect*). These changes, as mentioned previously through the example, may result in additional changes in the view instance apart from the one requested by the view update command. Thus, another important desideratum is to find translations that minimize the number of changes in the view (*view-side-effect*). In certain cases, these two desiderata may conflict.

Finding a translation of a view deletion that minimizes the changes in the base tables or/and the view is in general an NP-hard problem, even with monotone view queries. The complexity remains NP-hard even for the very restrictive classes of views with *select-project-join* or *select-join-union* view queries [4].

A different approach to the view update problem is to deal with it at the semantic level. This approach has been advocated by many early researchers [1]. Their position is that at the moment of view definition, the data administrator needs to specify how each update on the view will be translated to updates on the base tables. This is a safe approach that guarantees a semantically correct translation. However, this approach may be hard to apply in certain practical situations. One reason is that views may have been defined at a point in which updates were not of interest and the correct translation semantics may be difficult to infer. Furthermore, even if the data designer is aware of the existence of updates on the view, she may not be able to predict at the moment of view definition all the different semantics of the possible view update commands. For instance, the deletion of the [Administration, Smith] tuple in the example above may be simply because Smith stopped working for the Administration department, or because he moved from the Administration to another department. Another reason why this approach is difficult to implement is that views may have been introduced as replacements of tables in cases of physical or logical data reorganization, in which case again, more than one alternative translation may be semantically correct.

Keller [7] has provided a detailed classification of the different kinds of view update translations for projections, selections, select-project-join, and select-project-join-union views. Through an interactive procedure with the data administrator at the time of the view definition, the system tries to infer the right translation rules. At the other end, Dayal and Bernstein [6] have studied the classes of views for which the translation of an update is not ambiguous. For the case of translation ambiguity or the case in which it is possible to implement the update but with more changes in the view instance than those the view command requested, then the update may not be allowed.

$$\begin{tikzcd}
Q_V(I) \arrow[r, "U", ""{name=U, below}] & U(Q_V(I)) & Q_V(W(I)) \\
I \arrow[u] \arrow[rr, "W", ""{name=W}] & & W(I) \arrow[u]
\arrow[from=U, to=W]
\end{tikzcd}$$

Updates through Views. Figure 1. The updates through views problem.

A different approach in finding a view update translation is to reduce it to a constraint satisfaction problem [9]. Relational views can be represented as conditional tables and a view update can be translated into a disjunction of a number of constraint satisfaction problems (CSPs). Solutions to the CSPs correspond to possible translations. For the case of multiple candidate translations, semantic information to resolve ambiguity can be modeled as additional constraints.

The problem of updates through views appears also in other models, such as the object views or XML. Since most of the studies have been done for the relational model, one way to deal with the problem of updating object or XML views is to abstract it to the relational case [2,3].

Key Applications

Information Integration Systems. IIS are used to provide a unified view of data that is stored in multiple heterogeneous and physically distributed data sources. The majority of such systems are read-only, since data maintenance is considered to be an exclusive responsibility of the owner of each data source. This situation is gradually changing. Integrated views provide an excellent opportunity for detecting errors and inconsistencies among the different data sources. Once these inconsistencies are detected and corrected at the view level, the data in the sources will have to be accordingly updated.

Corporate Environments. Multiple different systems may operate in different parts of an organization, but centralized data management is crucial for decision making and policy implementations. Views provide the mean to propagate changes from the centralized level to the individual systems of the organization.

P2P. P2P systems use views to describe the relationships between the contents of one peer and its acquaintances. When a modification takes place in one of the peers, the views are used to determine its effects (if any) on the contents of its adjacent peers.

Database Management Systems. View support has been prevalent in Database Management Systems. They provide physical data independence, i.e., decoupling the logical schema from the physical design which is usually driven by performance reasons. By controlling access to the views, one can control access to different parts of the data and implement different security policies. It is not uncommon the case of databases that provide access to their data exclusively through views. For them the ability to perform arbitrary updates on the views is becoming a necessity since the only way to update the stored information is by issuing updates on the views.

WEB. The World Wide Web has evolved to be the world's largest database where organizations and individuals publish their information and data. Web users are then using applications and integration engines to query and retrieve that information. Since the users have no access to the individual sources, the web in its current form is a read-only system, restricting the potential of its million users. If the Web is to be brought from its read-only status into a full-fledged database, allowing updates on the web views to propagate to the sources is a fundamental requirement.

Cross-references

- ▶ Provenance
- ▶ Side-Effect-Free View Updates
- ▶ View Maintenance

Recommended Reading

1. Aaron B., Jeffrey A. Vaughan, and Benjamin C. Pierce. Relational lenses: a language for updatable views. In Proc. 27th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 338–347.
2. Barsalou T., Siambela N., Keller A.M., and Wiederhold G. Updating relational databases through object-based views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991, pp. 248–257.
3. Braganholo V.P., Davidson S.B., and Heuser C.A. On the updatability of XML views over relational databases. In Proc. 6th Int. Workshop on the World Wide Web and Databases, 2003, pp. 31–36.
4. Buneman P., Khanna S., and Tan W.C. On propagation of deletions and annotations through views. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 150–158.
5. Cui Y., Widom J., and Wiener J.L. Tracing the lineage of view data in a data warehousing environment. ACM Trans. Database Syst., 25(2):179–227, 2000.
6. Dayal U. and Bernstein P. On the correct translation of update operations on relational views. ACM Trans. Database Syst., 8(3):381–416, 1982.
7. Keller A.M. Choosing a view update translator by dialog at view definition time. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 467–474.
8. Kotidis Y., Srivastava D., and Velegrakis Y. Updates through views: a new hope. In Proc. 21st Int. Conf. on Dara Engineering, 2005.
9. Shu H. Using constraint satisfaction for view update translation. In Proc. 21st Int. Conf. on Data Engineering, 2005.
10. Widom J. Research problems in data warehousing. In Proc. Int. Conf. on Information and Knowledge Management, 1995, pp. 25–30.

URI

▶ Resource Identifier

Usability

Nigel Bevan
Professional Usability Services, London, UK

Synonyms

User centered design

Definition

A product is usable if the intended users can achieve their goals with effectiveness, efficiency and satisfaction in a specified context of use [8]. Usability is achieved by taking a user-centered approach to design, and thus ensuring that the product incorporates characteristics that support usability.

Historical Background

Usability was adopted as a technical term to replace the phrase "user friendly," which by the early 1980s had acquired undesirably vague and subjective connotations. It is a goal for product design in the scientific fields of HCI, human factors and ergonomics. In 1985

Gould and Lewis [5] described the central principles of what became known as user-centered design [15]: (i) early focus on users and tasks., (ii) empirical measurement, (iii) iterative design.

Usability has since grown into an established discipline with the Usability Professionals Association founded in 1991, and the landmark book by Nielsen on Usability Engineering [13] published in 1993.

Although the field of usability overlaps with human factors and ergonomics, it tends to be associated with systems that have discretionary users. Human factors and ergonomics have traditionally focused on how human capabilities can be integrated into pre-defined tasks and work systems. Usability is more typically concerned with designing applications to support users and tasks that are often not well-understood.

Foundations

The broad interpretation of usability is the user's experience of the quality of the system in use [1]: to what extent the user is successful in achieving their goals (effectiveness), in an acceptable amount of time (efficiency), without undesirable side-effects (safety), and in a way that satisfies the user? While this interpretation is preferred by people working in the field, in systems development usability is often interpreted more narrowly as ease of use (as in the 1991 definition of usability in ISO 9126 [10]). More recently it has been appreciated that the prerequisites for achieving usability as a goal include not only software and hardware with appropriate functionality, reliability and ease of use, but also usable data [12].

The broad interpretation of usability is thus a black box view at the system level, measured in terms of success in achieving goals for effectiveness, efficiency, safety and satisfaction (Fig. 1). It is a result of the user's interaction with the product, and is facilitated by appropriate product characteristics.

In the context of databases, the major product characteristics for usability are software and data.

Software Usability

There are many well-established design principles and standards, both for general software usability, and for web usability. At a high level they include heuristics such as [13]:

- Use simple and natural dialogue, and speak the users' language.
- Minimize user memory load.
- Be consistent in screen and task design.
- Provide feedback in response to user input.
- Provide shortcuts and clearly marked exits.
- Prevent errors and provide informative error messages, help and documentation.

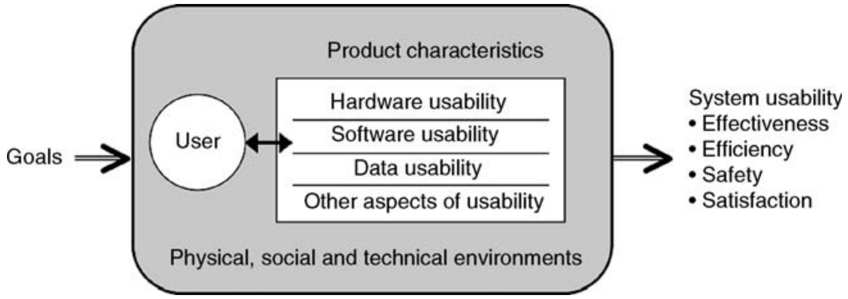
At a detailed level ISO 9241 [8] contains comprehensive interface guidelines, and the US Government has produced an authoritative and well-documented set of web design guidelines [18].

Data Usability

ISO is developing a model for data quality [7]. Data characteristics that contribute to overall system usability include completeness, depth of information, accuracy and understandability. Poor data quality makes it very difficult for users to achieve their goals, even if the user interface is easy to use.

For example, an online train timetable will have poor usability if the data is not usable, for example if:

- The destinations or types of ticket are *incomplete*.
- There is insufficient *depth of information* about the destinations or types of ticket



Usability. Figure 1. System usability.

- Any of the train times or ticket information are *inaccurate*
- The user cannot *understand* the names used for the destinations or types of ticket.

Key Applications

User Centered Design

A product will only be usable if the design and development process is based on an in depth understanding of the range and types of intended users, their task, and the physical, technical and social environments of use. Potential design solutions need to be continually evaluated from a user perspective from the earliest stages of design. The activities necessary to achieve user centered design are described at a high level in ISO 13407 [9], and in more detail in ISO 18529 [9].

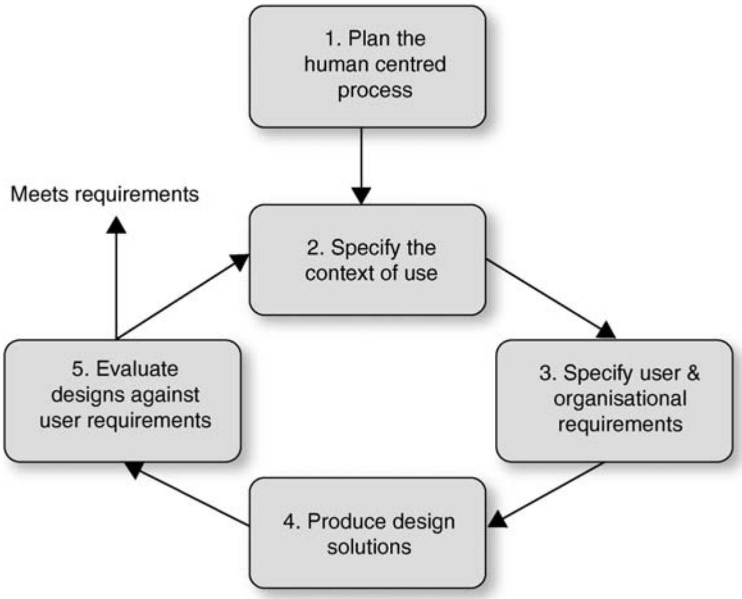
Key activities are illustrated in Fig. 2:

1. Planning. The nature and complexity of the user-centered activities should depend on the importance of usability, and a judgement on which particular activities are necessary for project success.
2. Context of Use. Detailed information about the context of use is an essential input to requirements, and provides a basis for prioritizing testing.
3. User and Organizational Requirements. These should be specified both at a high level in terms of user performance and satisfaction, and at a more detailed level of user interface characteristics.
4. Early design solutions should be produced as mock-ups, typically paper-based, simulations for exploratory testing [17]. Later in development, computer simulations can be used evaluate the user interface.
5. Designs should be evaluated from a user perspective using both expert and user-based methods, to obtain design feedback, and to establish whether requirements have been met.

Context of Use

The characteristics of the users, tasks and the organizational and physical environment define the context in which the system is used. It is important to gather and analyze information on the current context in order to understand and then specify the context that will apply in the future system. Analysis of existing or similar systems can provide information on a whole range of context issues including needs, problems and constraints that might otherwise be overlooked.

Methods for identifying the context range from collecting information in a workshop attended by the relevant stakeholders, to detailed field studies and task analysis.


1. Plan the human centred process
Meets requirements
2. Specify the context of use
5. Evaluate designs against user requirements
3. Specify user & organisational requirements
4. Produce design solutions

Usability. Figure 2. User centered design process.

User and Organizational Requirements

User needs should be identified in conjunction with identifying the context of use. The system requirements should include criteria for user performance, for minimization of adverse effects and for user satisfaction. These are most easily set in relation to baseline values for a comparable existing system. More detailed requirements should be developed iteratively incorporating feedback from early design solutions.

Design Solutions

Using simulations, mock-ups or prototype allows designers to communicate in a meaningful way what the proposed design is/would be like to users and other stakeholders. Paper prototypes can be very effective ingathering feedback [17]. The design should be consistent with established principles for software and data usability [8,12].

Usability Evaluation

Usability evaluation will reveal the strengths and weakness in the design solution and can indicate where the design should be improved.

Feedback from usability evaluation is particularly important because developers seldom have an intimate understanding of the user's perspective and work practices. Initial designs therefore very rarely fully meet user requirements. The cost of rectifying any divergence between the design and user needs increases rapidly as development proceeds, which means that user feedback should be obtained as early as possible. Without proper usability evaluation, a project runs a high risk of expensive rework to adapt a system to actual user needs.

Evaluation of overall system usability and of detailed product characteristics are complementary. Although a user-based evaluation is the ultimate test of usability, it is not usually practical to evaluate all permutations of user types, tasks and operational conditions. Evaluation of the detailed characteristics of the product or interactive system can anticipate and explain potential usability problems, and can be carried out before there is a working system. However evaluation of detailed characteristics alone can never be sufficient, as this does not provide enough information to accurately predict the eventual user behavior in every context.

The purpose of the evaluation may be primarily to obtain design feedback to identify and fix any obstacles to effective usability, or to validate the usability of a system. The most useful validation data are measures of user performance and satisfaction obtained from measuring system usability.

When to Use Methods for Design Feedback The most common type of usability evaluation is to improve a product by identifying and fixing usability problems. Evaluation of early mock-ups can also be used to obtain a better understanding of user needs and to refine requirements. An iterative process of repeated evaluation of prototypes can be used to monitor how closely the prototype designs match user needs. The feedback can be used to improve the design for further testing. Early evaluation reduces the risk of expensive rework. Usability evaluation is most effective when it involves a combination of expert and user-based methods.

When to Use Validation Methods In a more mature design process evaluation to obtain design feedback should be complemented by establishing usability requirements and testing whether these have been achieved by using a more formal method to validate usability. This reduces the risk of delivering a product that fails as a result of poor user performance.

Similar testing of an existing system can be used to provide baseline measures that can form the basis for usability requirements (i.e., objectives for human performance and user satisfaction ratings). A Common Industry Specification for Usability Requirements [14] has been developed to support iterative development and sharing of such requirements.

Validation tests at the end of development should have acceptance criteria derived from the usability requirements. Validation methods can be elaborated to also identify usability problems, but if prior iterative rounds of usability testing are performed, then typically there will be few usability surprises uncovered during this late stage testing.

Validity Usability testing should be carried out carefully and systematically, otherwise the resulting data may not be valid or reliable. O'Hara et al. [16] describe the need for the following types of validity:

- External validity: extent to which the context of use for the test is realistic.
- Construct validity: extent to which the measures are representative of user performance and satisfaction.

- Internal validity: extent to which the test is properly designed.
- Statistical validity: extent to which the statistical conclusions are valid.

Poor usability data may lead to poor design decisions and ultimately an error-prone and unsafe product. Therefore, usability testing protocols should be developed in collaboration with professional usability specialists. Non-specialists under the direction and training of professionals can handle the actual execution and reporting of the testing.

Cost Benefits

The objective of user centered design is to ensure that products can be used by real people to achieve their tasks in the real world. This requires not only easy-to-use interfaces, but also the appropriate functionality and support for real business activities and work flows. According to IBM [6], developing easy-to-use products "makes business effective. It makes business efficient. It makes business sense."

User centered design can reduce development and support costs, increase utilization, and reduce staff costs for employers [2]:

- Development costs can be reduced by fixing usability problems early in the development process, avoiding unnecessary functionality and minimizing documentation.
- Support costs can be reduced by minimizing help line, maintenance and training costs.
- Utilization of the system by individuals and organizations can be increased as a result of higher user success rates, productivity and satisfaction.

Usability of Digital Libraries

Digital libraries pose some particular challenges for usability, as a result of the large amount of information they contain, the difficult of determining what users want, and the need to cater for browsing and searching. Digital libraries demand more sophistication of query formulation than web search engines. Skills acquired in one library environment are often not easily transferred to another [3].

Cross-references

▶ Data Quality Dimensions
▶ Human-Computer Interaction
▶ Information Quality Assessment

Recommended Reading

1. Bevan N. Quality in use: meeting user needs for quality. J. Syst. Softw., 49(1):89–96, 1999.
2. Bevan N. Cost benefits framework and case studies. In Cost-Justifying Usability: An Update for the Internet Age, R.G. Bias, D.J. Mayhew (eds.). Morgan Kaufmann, San Francisco, CA, 2005.
3. Blandford A. and Buchanan G. Usability of digital libraries: a source of creative tensions with technical developments. IEEE-TCDL, Bulletin, 2003.
4. Dumas J.S. and Redish J.C. A Practical Guide to Usability Testing. Ablex Publishing Corporation, Norwood, NJ, 1993.
5. Gould J.D. and Lewis C. Designing for usability: key principles and what designers think. Commun. ACM, 28(3):300–311, 1985.
6. IBM. Cost Justifying Ease of Use, 2000. Available at: www-03.ibm.com/easy/page/23
7. ISO 13407. User centred design process for interactive systems. ISO, 1998.
8. ISO 9241. Ergonomic requirements for office work with visual display terminals (VDT)s (Pts 10–17). ISO, 1997–99.
9. ISO TR 18529. Human-centred lifecycle process descriptions. ISO, 2000.
10. ISO/IEC 9126. Software product evaluation – Quality characteristics and guidelines for their use. ISO, 1991.
11. ISO/IEC CD 25010. Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Quality model. ISO, 2008.
12. ISO/IEC FCD 25012. Software engineering – Software product quality requirements and evaluation (SQuaRE) – Data quality model. ISO, 2008.
13. Nielsen J. Usability Engineering. Academic Press, San Diego, CA, 1993.
14. NIST. Industry Usability Reporting, 2007. Available at: www.nist.gov/iusr
15. Norman D.A. and Draper S.W. User Centered System Design New Perspectives on Human–Computer Interaction. Lawrence Erlbaum Associates, Mahwah, NJ, 1986.
16. O'Hara J., Stubler W., Higgins J., and Brown W. Integrated System Validation: Methodology and Review Criteria (NUREG/CR-6393), 1995. Available at: www.bnl.gov/humanfactors/Publications.asp
17. Snyder C. Paper Prototyping the Fast and Easy Way to Define and Refine User Interfaces. Morgan Kaufmann, San Francisco, CA, 2003.
18. U.S. Department of Health and Human Sciences. Research-Based Web Design & Usability Guidelines, 2006. Available at: www.usability.gov/guidelines

User Centered Design

▶ Usability

User Classifications

► Lightweight Ontologies

User-Centred Design

► Human-Computer Interaction

User-Defined Time

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

The concept of *user-defined time* denotes time-valued attributes of database items with which the data model and query language associate no special semantics. Such attributes may have as their domains all domains that reference time, e.g., date and time. The domains may be instant-, period-, and interval-valued. Example user-defined time attributes include "birthday," "hiring date," and "contract duration." Thus, user-defined time attributes are parallel to attributes that record, e.g., salary, using domain "money," and publication count, using domain "integer." User-defined time attributes contrast transaction-time and valid-time attributes, which carry special semantics.

Key Points

The valid time and transaction time attributes of a database item are "about" the other attributes of the database item. The valid time records when the information recorded by the attributes is true in the modeled reality, and the transaction time captures when the data item was part of the current database state. In contrast, user-defined time attributes are simply "other" attributes that may be used for the capture of information with which valid time can be associated.

Conventional database management systems generally support a time and/or date attribute domain. The SQL2 standard has explicit support for user-defined time in its `datetime` and `interval` types.

Cross-references

► Temporal Database
► Transaction Time
► Valid Time

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS, vol. 1399, Springer-Verlag, Berlin, 1998, pp. 367–405.
2. Snodgrass R.T. and Ahn I. A taxonomy of time in databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1985, pp. 236–246.
3. Snodgrass R.T. and Ahn I. Temporal databases. IEEE Comput., 19(9):35–42, September 1986.

User-Level Parallelism

► Inter-query parallelism

Using Efficient Database Technology (DB) for Effective Information Retrieval (IR) of Semi-Structured Text

► Integrated DB&IR Semi-Structured Text Retrieval

Utility Computing

► Storage Grid

UUID

► Resource Identifier

Valid Time

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Real-world time; Intrinsic time; Logical time; Data time

Definition

The *valid time* of a fact is the time when the fact is true in the modeled reality. Any subset of the time domain may be associated with a fact. Thus, valid timestamps may be sets of time instants and time intervals, with single instants and intervals being important special cases. Valid times are usually supplied by the user.

Key Points

While other temporal aspects have been proposed, such as "decision time" and "event time," closer analysis indicates that the decision and event time of a fact can be captured as either a valid time or a transaction time of some related fact.

For example, consider a valid-time `Faculty` table with attributes `Name` and `Position` (that is, either Assistant Professor, Associate Professor, or Professor) that captures the positions of faculty members. The valid time then captures when a faculty member held a specific position.

The "decision time" of a faculty member holding a specific position would be the time the decision was made to hire or promote the faculty member into that position. In fact, there are generally several decision times for a promotion: recommendation by departmental promotion and tenure committee, recommendation by department chair, recommendation by college promotion and tenure committee, recommendation by Dean, and finally decision by Provost. Each of these decisions may be modeled as a separate fact, with an (event) valid time that captures when that decision was made.

Oracle explicitly supports valid time in its Workspace Manager. The support includes sequenced primary keys, sequenced uniqueness, sequenced referential integrity, and sequenced selection and projection.

The term "valid time" is widely accepted; it is short and easily spelled and pronounced. Most importantly, it is intuitive.

Concerning the alternatives, the term "real-world time" derives from the common identification of the modeled reality (opposed to the reality of the model) as the real world. This term is less frequently used. "Intrinsic time" is the opposite of extrinsic time. Choosing intrinsic time for valid time would require one to choose extrinsic time for transaction time. The terms are appropriate: The time when a fact is true is intrinsic to the fact; when it happened to be stored in a database is clearly an extrinsic property. However, "intrinsic" is rarely used. "Logical time" has been used for valid time in conjunction with "physical time" for transaction time. As the discussion of intrinsic time had to include extrinsic time, discussing logical time also requires the consideration of physical time. Both terms are used much less frequently than valid and transaction time, and they do not posses clear advantages over these. The term "data time" is probably the most rarely used alternative. While it is clearly brief and easily spelled and pronounced, it is not intuitively clear that the data time of a fact refers to the valid time as defined above.

Cross-references

- ▶ Nonsequenced Semantics
- ▶ Sequenced Semantics
- ▶ Temporal Database
- ▶ Temporal Generalization
- ▶ Time Instant
- ▶ Time Interval
- ▶ Time Period
- ▶ Transaction Time
- ▶ User-Defined Time
- ▶ Valid-Time Indexing

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin, 1998, pp. 367–405.
2. Snodgrass R.T. and Ahn I. A taxonomy of time in databases. In Proc. ACM-SIGMOD Int. Conf. on Management of Data, 1985, pp. 236–246.
3. Snodgrass R.T. and Ahn I. Temporal databases. IEEE Comput., 19(9):35–42, September 1986.

Validity (Satisfiability)

▶ Certain (and Possible) Answers

Valid-Time Access Methods

▶ Valid-Time Indexing

Valid-Time Algebras

▶ Temporal Algebras

Valid-Time and Transaction-Time Relation

▶ Bitemporal Relation

Valid-Time Data Model

▶ Temporal Data Models

Valid-Time Indexing

Mirella M. Moro[1], Vassilis J. Tsotras[2]
[1]Federal University of Rio Grande do Sol,
Porto Alegre, Brazil
[2]University of California-Riverside, Riverside,
CA, USA

Synonyms

Valid-time access methods

Definition

A valid-time index is a temporal index that enables fast access to valid-time datasets. In a traditional database, an index is used for selection queries. When accessing valid-time databases such selections also involve the valid-time dimension (the time when a fact becomes valid in reality). The characteristics of the valid-time dimension imply various properties that the temporal index should have in order to be efficient. As traditional indices, the performance of a temporal index is described by three costs: (i) storage cost (i.e., the number of pages the index occupies on the disk), (ii) update cost (the number of pages accessed to perform an update on the index; for example when adding, deleting or updating a record), and (iii) query cost (the number of pages accessed for the index to answer a query).

Historical Background

A valid-time database maintains the entire temporal behavior of an enterprise as best known now [13]. It stores the current knowledge about the enterprise's past, current or even future behavior. If errors are discovered about this temporal behavior, they are corrected by modifying the database. If the knowledge about the enterprise is updated, the new knowledge modifies the existing one. When a correction or an update is applied, previous values are not retained. It is thus not possible to view the database as it was before the correction/update. This is in contrast to a transaction-time database, which maintains the database activity (rather than the real world history) and can thus rollback to a past state. Hence, in a valid-time database the past can change, while in a transaction-time database it cannot.

The problem of indexing valid-time databases can be reduced to indexing dynamic collections of intervals, where an interval represents the temporal validity of a record. Note that the term "interval" is used here to mean a "convex subset of the time domain" (and not a "directed duration"). This concept has also been named a "period"; in this discussion, however, only the term "interval" is used.

To index a dynamic collection of intervals, one could use R-trees or related dynamic access methods. Relatively fewer approaches have been proposed for indexing valid-time databases. There have been various approaches proposed for the related problem of managing intervals in secondary storage. Given the problem difficulty, the majority of these approaches have focused on achieving good worst case behavior; as a result, they are mainly of theoretical importance.

For a more complete discussion the reader is referred to a comprehensive survey [12].

Foundations

The following scenario exemplifies the distinct properties of the valid time dimension. Consider a dynamic collection of "interval-objects." The term interval-object is used to emphasize that the object carries a valid-time interval to represent the temporal validity of some object property. (In contrast, and to emphasize that transaction-time represents the database activity rather than reality, note that intervals stored in a transaction time database correspond to when a record was inserted/updated in the database.)

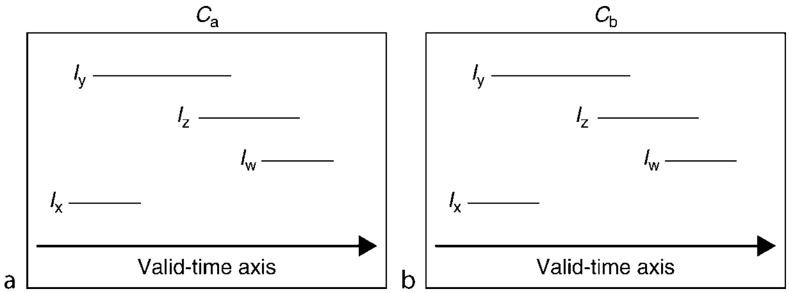
The allowable changes in this environment are the addition/deletion/modification of an interval-object. A difference with the transaction-time abstraction (the reader is referred to the Transaction-Time Indexing entry for more details) is that the collection's evolution (past states) is *not* kept. An example of a dynamic collection of interval-objects appears in Fig. 1. Assume that collection C_a has been recorded in some erasable medium and a change happens, namely object I_z is deleted. This change is applied on the recorded data and physically deletes object I_z. The medium now stores collection C_b, i.e., collection C_a is *not* retained. Note that when considering the valid time dimension, changes do not necessarily come in increasing time order (as is the case in transaction-time databases); rather they can affect *any* object in the collection. This implies that a valid-time database can correct errors in previously recorded data. However, only a single data state is kept, the one resulting after the correction is applied.

As a real-life example, consider the collection of contracts in a company. Each contract has an identity (*contract_no*) and an interval representing the contract's duration or validity. In collection C_a, there were four contracts in the company. But assume an error was discovered: contract I_z was never a company contract (maybe it was mistakenly entered). Then, this information is permanently deleted from the collection, which now is collection C_b.

The notion of time is now related to the valid-time axis. Given a valid-time instant, interval-objects can be classified as past, future or current as related to this instant, if their valid-time interval is before, after or contains the given instant. Valid-time databases can be used to correct errors anywhere in the valid-time domain (past, current or future) because the record of any interval-object in the collection can be changed, independently of its position on the valid-time axis. Note that a valid-time database may store records with the same surrogate but with non-intersecting valid-time intervals. For example, another object with identity I_x could be added in the collection at C_b as long as its valid-time interval does not intersect with the valid-time interval of the existing I_x object; the new collection will contain both I_x objects, each representing object I_x at different times in the valid-time dimension.

From the above discussion, an index used for a valid-time database should: (i) store the latest collection of interval-objects, (ii) support addition/deletion/modification changes to this collection, and (iii) efficiently query the interval-objects contained in the collection when the query is asked. Hence, a valid-time index should manage a *dynamic* collection of intervals.

Thus related is research on the interval management problem. Early work in main memory data-structures has proposed three separate approaches into managing intervals, namely, the Interval Tree, the Segment Tree and the Priority Search Tree [9]. Such approaches are focused on the "stabbing query" [6], i.e., given a collection of intervals, find all intervals



Valid-Time Indexing. Figure 1. Two valid-time databases.

that contain a given query point q. The stabbing query is one of the simpler queries for a valid-time database environment, since it does not involve any object key attribute (rather, only the object intervals).

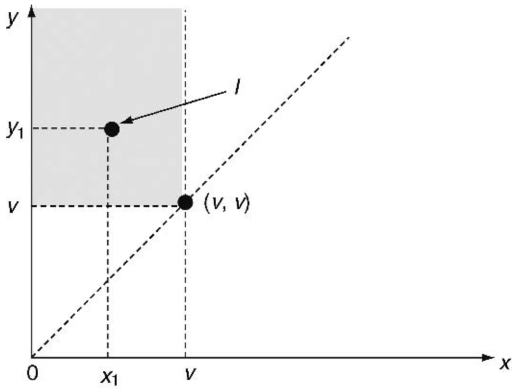
Various methods have been proposed for making these main-memory structures disk-resident. Among them are: the External Memory Interval Tree [1], the Binary-Blocked Interval Tree [4], the External Segment Tree [2], the Segment Tree with Path Caching [11] and the External Memory Segment Tree [1], the Metablock Tree [6], the External Priority Search Tree [5], and the Priority Search Tree with Path Caching [11]. Many of these approaches are mainly of theoretical interest as they concentrate on achieving a worst-case optimal external solution to the 1-dimensional stabbing query.

Recently, [8] presented the Relational Interval Tree (RI-tree) which is based on the original Interval Tree. In particular, the Interval Tree uses a backbone balanced binary tree structure that organizes the values of all the bounding points of the intervals. An interval (l, r) is registered in the highest node w of the backbone tree that it overlaps (that is, while descending the backbone tree, the first node w for which $l \leq w \leq r$). Each inner node w contains two sorted lists, one with the lower bounding points and one with the upper bounding points of the intervals assigned to this node. A "stabbing" query is answered by traversing one path of the backbone structure and accessing the lists in the nodes of this path. The RI-tree uses plain B+-trees to index these sorted lists, thus leading to fast query time. For n intervals stored in the RI-tree, the space used is linear $O(n/B)$ where B is the page capacity in records. If h corresponds to the height of the backbone structure, answering a stabbing query that returns s intervals takes $O(h \, logBn + s/B)$ page accesses.

Since intervals are 2-dimensional objects, a dynamic multi-dimensional index like an R-tree may be used. Moreover, the R-tree can also index other attributes of the valid-time objects, thus enabling queries involving non-temporal attributes as well. For example, "find contracts that were active on time t and had contract-id in the range $(r1,r2)$." While simple, the traditional R-tree approach may not always be very efficient. The R-tree will attempt to cluster intervals according to their length, thus creating pages with possibly large overlapping. It was observed in [7] that for data with non-uniform interval lengths (i.e., a large proportion of "short" intervals and a small proportion of "long" intervals), this overlapping is clearly increased, affecting the query and update performance of the index. This in turn decreases query and update efficiency.

Another straightforward approach is to transform intervals into 2-dimensional points and then use a Point Access Method (quad-tree, grid file, hB-tree, etc.). In Fig. 2, interval $I = (x_1,y_1)$ corresponds to a single point in the 2-dimensional space. Since an interval's end-time is always greater or equal than its start-time, all intervals are represented by points above the diagonal $x = y$. Note that an interval (x,y) contains a query time v if and only if its start-time x is less than or equal to v and its end-time y is greater than or equal to v. Then an interval contains query v if and only if its corresponding 2-dimensional point lies inside the box generated by lines $x = 0$, $x = v$, $y = v$, and $y = \infty$ (the shaded area in Fig. 2). This approach avoids the overlapping mentioned above. Long intervals will tend to be stored together in pages with other long intervals (similarly, for the short intervals). However, no worst case guarantees for good clustering are possible.

A related approach (MAP21) is proposed in [11]. An interval (l, r) is mapped to a point $z = (l \times 10^s) + r$, where s is the maximum number of digits needed to represent any point in the interval range. This is enough to map each interval to a separate point. A regular B$^+$-tree is then used to index these points. An advantage of this approach is that interval insertions/deletions are easy using the B$^+$-tree. To answer a stabbing query about q, the point closer but less than q is found among the points indexed in the B$^+$-tree, and then a sequential search for all intervals before q is performed. At worse, many intervals that do not intersect q can be found (this approach assumes that in

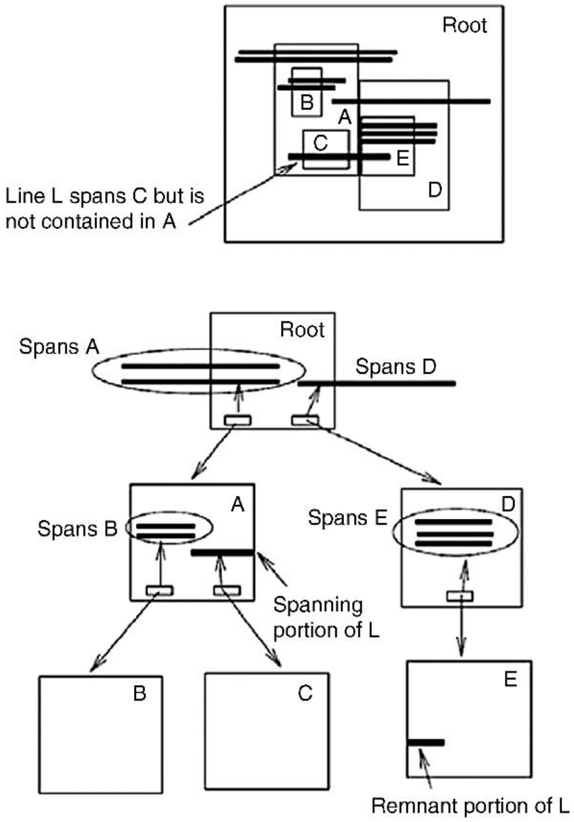


Valid-Time Indexing. Figure 2. An interval $I = (x_1,y_1)$ corresponds to a point in a 2-dimensional space.

practice the maximal interval length is known, which limits how far back the sequential search continues from *q*).

Another approach is to use a combination of an R-tree with Segment Tree properties. The Segment R-tree (SR-tree) was proposed in [11]. The SR-tree is an R-tree where intervals can be stored in both leaf and non-leaf nodes. An interval *I* is placed to the highest level node *X* of the tree such that *I* spans at least one of the intervals represented by *X*'s child nodes. If *I* does not span *X*, it spans at least one of its children but is not fully contained in *X*, then *I* is fragmented. Figure 3 shows an example of the SR-tree approach. The top rectangle depicts the R-tree nodes (root, A, B, C, D and E) as well as the stored intervals. Interval *L* spans the rectangle of node C, but is not contained in node A. It is thus fragmented between nodes A and E.

Using this idea, long intervals will be placed in higher levels of the R-tree. Hence, the SR-tree tends to decrease the overlapping in leaf nodes (in the regular R-tree, a long interval stored in a leaf node will "elongate" the area of this node thus exacerbating the overlap problem). However, having large numbers of spanning records or fragments of spanning records stored high up in the tree decreases the fan-out of the index as there is less room for pointers to children. It is suggested to vary the size of the nodes in the tree, making higher-up nodes larger. "Varying the size" of a node means that several pages are used for one node. This adds some page accesses to the search cost.



Valid-Time Indexing. Figure 3. An example of the SR-tree approach.

As with the R-tree, if the interval is inserted at a leaf (because it did not span anything) the boundaries of the space covered by the leaf node in which it is placed may be expanded. Expansions may be needed on all nodes on the path to the leaf, which contains the new record. This may change the spanning relationships since existing intervals may no longer span children, which have been expanded. In this case, such intervals are reinserted in the tree, possibly being demoted to occupants of nodes they previously spanned. Splitting nodes may also cause changes in spanning relationships as they make children smaller – former occupants of a node may be promoted to spanning records in the parent.

In contrast to the traditional R-tree, the space used by the SR-tree is no longer linear. An interval may be stored in more than one non-leaf nodes (in the *spanning* and *remnant* portions of this interval). Due to the use of the segment-tree property, the space can be as much as $O(n \log_B n)$. Inserting an interval still takes logarithmic time. However, due to possible promotions, demotions and fragmentation, insertion is slower than in the R-tree. Even though the segment property tends to reduce the overlapping problem, the (pathological) worst case performance for the deletion and query time remains the same as for the R-tree organization (that is, at worst, the whole R-tree may have to searched for an interval). The average case behavior is, however, logarithmic. Deletion is a bit more complex, as all the remnants of the deleted interval have to be deleted too. The original SR-tree proposal thus assumed that deletions of intervals are not that frequent.

The SR-tree search algorithm is similar to that of the original R-tree. It descends the index depth-first, descending only those branches that contain the given query point *q*. In addition, at each node encountered during the search, all spanning intervals stored at the node are added to the answer. To improve the performance of the structure, the *Skeleton SR-tree* has also been proposed [7], which is an SR-tree that prepartitions the entire domain into some number of regions. This pre-partition is based on some initial assumption on the distribution of data and the number of intervals to be inserted. Then the Skeleton SR-tree is populated with data; if the data distribution is changed, the structure of the Skeleton SR-tree can be changed too.

When indexing valid-time intervals, overlapping may also incur if the valid-time intervals extend to the ever-increasing *now*. One approach could be to use the largest possible valid-time to represent the variable *now*. In [3] the problem of addressing both the *now* and transaction-time Until-Changed (*UC*) variables is addressed by using bounding rectangles/regions that increase as the time proceeds. A variation of the R-tree, the GR-tree is presented. More details appear in [3].

Key Applications

The importance of temporal indexing emanates from the many applications that maintain temporal data. The ever increasing nature of time imposes the need for many applications to store large amounts of temporal data. Specialized indexing techniques are needed to access such data. Temporal indexing has offered many such solutions that enable fast access.

Cross-references

- ▶ Bi-temporal Indexing
- ▶ B+-Tree
- ▶ Temporal Database
- ▶ Transaction Time
- ▶ Transaction-Time Indexing
- ▶ Valid Time

Recommended Reading

1. Arge L. and Vitter J.S. Optimal dynamic interval management in external memory. In Proc. 37th Annual Symp. on Foundations of Computer Science, 1996, pp. 560–569.
2. Blankenagel G. and Gueting R.H. External segment trees. Algorithmica, 12(6):498–532, 1994.
3. Bliujute R., Jensen C.S., Saltenis S., and Slivinskas G. R-tree based indexing of now-relative bi-temporal data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 345–356.
4. Chiang Y.-J. and Silva C.T. External memory techniques for isosurface extraction in scientific visualization, external memory algorithms and visualization. In DIMACS Series in Discrete Mathematics and Theoretical Computer Science, J. Abello, J.S. Vitter (eds.). AMS, vol. 50, 1999, pp. 247–277.
5. Icking C., Klein R., and Ottmann T. Priority search trees in secondary memory. In Proc. Int. Workshop on Graph Theoretic Concepts in Computer Science, 1988, pp. 84–93.
6. Kanellakis P., Ramaswamy S., Vengroff D., and Vitter J.S. Indexing for data models with constraint and classes. In Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1993, pp. 233–243.
7. Kolovson C. and Stonebraker M. Segment indexes: dynamic indexing techniques for multi-dimensional interval data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991, pp. 138–147.
8. Kriegel H-P., Potke M., Seidl T. Managing intervals efficiently in object-relational databases. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000.
9. Mehlhorn K. Data Structures and Efficient Algorithms, Vol. 3: Multi-dimensional Searching and Computational Geometry. EATCS Monographs, Springer, 1984.
10. Nascimento M.A. and Dunham M.H. Indexing valid time databases via B^+-Trees. IEEE Trans. Knowl. Data Eng., 11(6):929–947, 1999.
11. Ramaswamy S. and Subramanian S. Path caching: a technique for optimal external searching. In Proc. 13th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1994, pp. 25–35.
12. Salzberg B. and Tsotras V.J. A comparison of access methods for time-evolving data. ACM Comput. Surv., 31(2):158–221, 1999.
13. Snodgrass R.T. and Ahn I. Temporal databases. IEEE Comput., 19(9):35–42, 1986.

Value Equivalence

NIKOS A. LORENTZOS
Agricultural University of Athens, Athens, Greece

Definition

In temporal databases, the scheme of a temporal relation S has the form S(**E, I**) where **E** and **I** represent two disjoint sets of attributes, termed by some authors *explicit* and *implicit*, respectively. *Explicit* are the attributes in which ordinary data are recorded, such as Employee_Id, Name, Salary etc. *Implicit* are the attributes in which either *valid time* (one or more) or *transaction time* is recorded. Given two tuples ($\mathbf{e}_1$, $\mathbf{i}_1$) and ($\mathbf{e}_2$, $\mathbf{i}_2$) with the scheme of S, it is said that there is a *value equivalence* between them if and only if $\mathbf{e}_1 = \mathbf{e}_2$. Equivalently, it is said that ($\mathbf{e}_1$, $\mathbf{i}_1$) and ($\mathbf{e}_2$, $\mathbf{i}_2$) are *value equivalent*.

Example: Consider one tuple, (Alex, 100, d400, d799), with scheme SALARY(Name, Amount, Valid-Time), indicating that Alex's salary was 100 on each of the dates d400, d401,...,d799. Let also another tuple of the same scheme be (Alex, 100, d500, d899). For the given scheme, **E** = {Name, Amount} and

I = {ValidTime}. Given that both tuples have the same value for Name and Amount, they are value equivalent.

Key Points

The term has been introduced in the context of temporal databases, mainly in order to ease discussion on issues such as *temporal coalescing*.

Cross-references

▶ Period-Stamped Temporal Models
▶ Temporal Coalescing

Variable Time Span

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Moving span

Definition

A span is *variable* if its duration is dependent on the assumed context.

Key Points

Given a specific setting, any span is either a fixed span or a variable span. An obvious example of a variable span is "1 month" in the Gregorian calendar. Its duration may be any of 28, 29, 30, and 31 days, depending on which particular month is intended. The span "1 h" is fixed because it always has a duration of 60 min.

Cross-references

▶ Fixed Time Span
▶ Temporal Database
▶ Temporal Granularity
▶ Time Interval
▶ Time Span

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer, Berlin, 1998, pp. 406–413.
2. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin, 1998, pp. 367–405.

VDM

▶ Visual Data Mining

Vector-Space Model

Massimo Melucci
University of Padua, Padua, Italy

Synonyms

VSM

Definition

The Vector-Space Model (VSM) for Information Retrieval represents documents and queries as vectors of weights. Each weight is a measure of the importance of an index term in a document or a query, respectively. The index term weights are computed on the basis of the frequency of the index terms in the document, the query or the collection. At retrieval time, the documents are ranked by the cosine of the angle between the document vectors and the query vector. For each document and query, the cosine of the angle is calculated as the ratio between the inner product between the document vector and the query vector, and the product of the norm of the document vector by the norm of the query vector. The documents are then returned by the system by decreasing cosine.

Historical Background

The use of vectors for modeling IR systems dates back to the early days of IR, especially as a tool for describing how a system should be designed and implemented. The popularity of the VSM is due to the intuitive yet formal view of indexing and retrieval – it should not come as a surprise that the VSM has attracted many researchers and newcomers since it was mentioned in [5], employed for indexing and clustering in [9], used as a framework for deciding whether a term should be stored in an index [10], and formulated in a comprehensive way in [6]. At the experimental level, the VSM has proved an effective and sound framework in retrieving documents in different languages, on different subjects, of different sizes, and of different media, thanks to a number of proposed and tested weighting schemes and applications (see [8] for a comprehensive evaluation). A perspective on the history of the VSM was illustrated in [2].

Despite its apparent simplicity, the mathematical properties of vector spaces can be used for advancing IR modeling. The hypothesis that the term-document frequency matrix contains information about the correlation among terms and among documents was cited in [7], stated in [13], developed in [6] and was further exploited in [1] in defining Latent Semantic Indexing (LSI). The latter is a technique based on Singular Value Decomposition (SVD) which aims at decomposing the term correlation matrix and at disclosing the principal components used to represent fewer independent concepts than many inter-dependent variables. A reconsideration of the potential of the vector spaces was presented in [12]. In that book, Hilbert's vector spaces, which is the mathematical tool of Quantum Mechanics, are used to see documents as vectors, relevance as a Hermitian operator, relevance statuses as the eigenvalues of this operator, and the computation of the probability of relevance of a document as the projection of the document vector onto the subspace of the eigenvalue of relevance. The idea of using the notion of basis of a vector space for representing context was proposed in [4].

Foundations

Basic Definitions

The VSM represents documents and queries as vectors of weights. A document vector is then

$$\mathbf{x} = (x_1,\ldots,x_k)$$

where x_i is the weight of index term i in the document and k is the number of unique index terms in the collection. Each weight is a measure of the importance of an index term in a document. A great deal of attention was paid in the past to the definition of the index term weights to be used in document vectors. An important example is

$$x_i = \mathrm{TF}_i$$

that is, the term frequency of i in the document. Of course, TF depends on the document. Another example is the binary weight,

$$x_i = \begin{cases} 1 & \text{if index term i occurs in the document} \\ 0 & \text{otherwise} \end{cases}$$

For an illustration of the term weighting schemes, see [8]. A query is represented as a vector of weights, too, for instance,

$$\mathbf{y} = (y_1,\ldots,y_k)$$

where y_i is the weight of index term i in the query. Each weight is a measure of the importance of an index term in a query – note that no Boolean operators are assumed. Also for queries, a great deal of attention has been paid in the past to the definition of the index term weights. An important example is

$$y_i = \mathrm{IDF}_i$$

where IDF stands for Inverse Document Frequency which is defined as

$$\mathrm{IDF}_i = \begin{cases} \log \frac{N}{n_i} & \text{if } i \text{ occurs in the query} \\ 0 & \text{otherwise} \end{cases}$$

where n_i is the number of documents indexed by i and N is the total number of documents in the collection. The binary weighting scheme can also be applied to queries.

At retrieval time, the documents are ranked by the cosine of the angle between the document vectors and the query vector. For each document and query, the cosine of the angle is calculated as the ratio between the inner product between the document vector and the query vector, and the product of the norm of the document vector by the norm of the query vector. The documents are then returned by the system by decreasing cosine. In a formal way,

$$\cos = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}|| \; ||\mathbf{y}||}$$

where

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{k} x_i \; y_i \qquad ||\mathbf{x}|| = \sqrt{\sum_{i=1}^{k} x_i^2}$$

$$||\mathbf{y}|| = \sqrt{\sum_{i=1}^{k} y_i^2}$$

When the weights are binary, i.e., $x_i, y_i \in \{0, 1\}$, the vectors are representations of term sets and an alternative ranking function is Jaccard's coefficient defined as

$$\frac{\sum_{i=1}^{k} x_i \; y_i}{\sum_{i=1}^{k} x_i + \sum_{i=1}^{k} y_i - \sum_{i=1}^{k} x_i \; y_i}$$

When term frequency and inverse document frequency are used for weighting terms in the document and in the query, respectively the inner product between document and query vectors is

$$\sum_{i=1}^{k} \mathrm{TF}_i \; \mathrm{IDF}_i$$

where

$$\mathrm{TF}_i \; \mathrm{IDF}_i$$

is called TF-IDF weighting scheme. When different documents are considered, the notations $\mathbf{x}_n = (x_{1,n},\ldots, x_{k,n})$ and $\mathrm{TF}_{i,n}$ IDF_i are used for distinguishing documents $n = 1,\ldots,N$.

Mathematical Development

The VSM for IR considers a document collection as a vector space defined over the real field. The definition of a vector is given with respect to a basis, namely, a set of linearly independent vectors, called basis vectors – a set of vectors is independent when no vector of this set can be defined as a linear combination of the other vectors in the same set. The dimension of the space is the number of basis vectors. It is a fact from Linear Algebra that every vector of a space of dimension k is a linear combination of k basis vectors – for an introduction to the vector spaces see, for example, [3].

The VSM represents a collection index as a basis of a real vector space and a distinct index term is a basis vector. Moreover, every object managed by an IR system, e.g., document, query, sentence, cluster centroids, or terms, is a vector of this space. As every vector is a linear combination of a basis of the vector space, in the VSM the vector of any object is a linear combination of the basis vectors that represent the index terms. When non-textual media are to be represented, the basis vectors are a representation of content descriptors, yet the philosophy of the model remains unchanged. A basis plays an important role in expressing term correlations and in Latent Semantic Indexing.

Since there are infinite bases in a space, every vector has infinite numerical representations. To be precise, let k be the dimension of the space and $\{\mathbf{t}_1,\ldots,\mathbf{t}_k\}$ be a basis for this space. The representation of a document or a query is then a vector $\mathbf{x}$ such that

$$\mathbf{x} = \sum_{i=1}^{k} a_i \; \mathbf{t}_i.$$

where the a_{i} are real numbers. The numerical representations are provided by the a_{i}'s with respect to the basis $\{\mathbf{t}_1,\ldots,\mathbf{t}_k\}$. This entails that a change of basis determines a change of the numerical representation of $\mathbf{x}$; for example,

$$\mathbf{x} = \sum_{i=1}^{k} b_i \; \mathbf{u}_i,$$

where $\{\mathbf{u}_1,\ldots,\mathbf{u}_k\}$ is a different basis and the b_{i}'s provide a different representation of $\mathbf{x}$ – it follows that a vector can be assigned infinite representations, all being equivalent.

According to the basic definitions, the a_{i}'s are the weights used for measuring the importance of the index terms in documents and queries. Thus, TF-IDF is a possible scheme for defining the a_{i}'s.

The ranking function is based on the cosine of the angle between the query vector $\mathbf{y}$ and a document vector $\mathbf{x}$. The documents are then ranked by (decreasing) cosine with respect to the query. Formally, the cosine of the angle between the two aforementioned vectors is:

$$\frac{\mathbf{x}^\top \cdot \mathbf{y}}{||\mathbf{x}|| \; ||\mathbf{y}||}$$

where (The vectors are assumed to be column vectors and the symbol $^\top$ denotes transposition from a column (row) vector to a row (column) vector.):

$$\mathbf{x} = \sum_{i=1}^{k} a_i \; \mathbf{t}_i \qquad \mathbf{y} = \sum_{j=1}^{k} c_j \; \mathbf{t}_j \qquad \mathbf{x}^\top \cdot \mathbf{y} = \sum_{l=1}^{k} x_l \; y_l$$

After a few simple passages,

$$\mathbf{x}^\top \cdot \mathbf{y} = \sum_{i=1}^{k} \sum_{j=1}^{k} a_i \; c_j \; \mathbf{t}_i^\top \cdot \mathbf{t}_j$$

where $\mathbf{t}_i{}^\top \cdot \; \mathbf{t}_j$ is a measure of correlation between the basis vectors. The cosine is used as ranking function instead of the inner product at the numerator because the norms at the denominator of the formula make document ranking independent of document size. In contrast, the inner product would place long documents at the top ranks, while relevant, short documents would be penalized; however, see [11]. When the index terms are assumed to be uncorrelated, the basis vectors are orthogonal, namely,

$$\mathbf{t}_i^\top \cdot \mathbf{t}_j = \begin{cases} ||\mathbf{t}_i||^2 & i = j \\ 0 & i \neq j \end{cases}$$

It is customary to assume that the basis vectors are orthonormal, therefore,

$$\mathbf{t}_i^\top \cdot \mathbf{t}_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

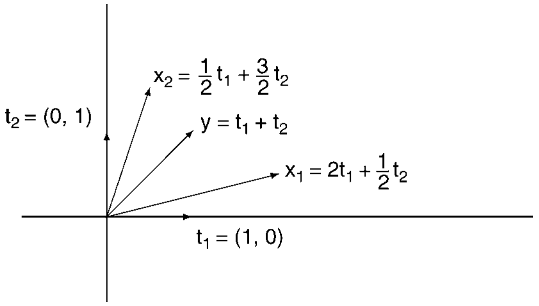
In practice, the basis is the canonical basis of a vector space, that is,

$$t_{i,j} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

where $t_{i,j}$ is the jth element of $\mathbf{t}_i$. In other words, the ith canonical basis vector has null elements except the ith element, which is 1. It follows that

$$\mathbf{x} = (a_1,\ldots,a_k)^\top \quad \mathbf{y} = (c_1,\ldots,c_k)^\top \quad \mathbf{x}^\top \cdot \mathbf{y} = \sum_{l=1}^{k} a_l\, c_l$$

when the canonical basis is used. Figure 1 gives a pictorial description of the VSM. The orthonormal basis ($\mathbf{t}_1$,$\mathbf{t}_2$) spans a two-dimensional spaces and every vector is thena linear combination of the basis vectors. In particular, $\mathbf{x}_1$ represents a document, $\mathbf{x}_2$ another document and $\mathbf{y}$ represents a query. After computing the cosine of the angle between the $\mathbf{x}_i$'s and $\mathbf{y}$,


$x_2 = \frac{1}{2}t_1 + \frac{3}{2}t_2$
$t_2 = (0, 1)$
$y = t_1 + t_2$
$x_1 = 2t_1 + \frac{1}{2}t_2$
$t_1 = (1, 0)$

Vector-Space Model. Figure 1. A pictorial description of the VSM.

$$\frac{\mathbf{y}^\top \cdot \mathbf{x}_1}{||\mathbf{y}||\,||\mathbf{x}_1||} = 0.86 \qquad \frac{\mathbf{y}^\top \cdot \mathbf{x}_2}{||\mathbf{y}||\,||\mathbf{x}_2||} = 0.89$$

If the basis is not orthogonal, for example, $\mathbf{t}_1 = (1, 0)^\top$ and $\mathbf{t}_2 = \left(\frac{1}{\sqrt{5}}, \frac{2}{\sqrt{5}}\right)^\top$, then, the numerical representation of the vectors and the cosines change; for example, the numerical representation of $\mathbf{x}_1$ is $\left(-\frac{1}{4}, \frac{5}{3}\right)^\top$.

Non-orthogonality has been employed for modeling relationships between index terms; for example, $\mathbf{t}_i{}^\top \cdot \mathbf{t}_j > 0$ for modeling that index term i is somehow related to index term j such that when i occurs, j also tends to occur – a negative value would model an inverse relationship. As a special, important case, non-orthogonality provides a principled way for modeling query expansion. Indeed, the cosine of the angle between $\mathbf{x}$ and $\mathbf{y}$ is in general affected by index term i when $a_i \neq 0$ or $c_i \neq 0$ – when the basis is orthogonal, the cosine is affected by i when $a_i \neq 0$ or $c_i \neq 0$ that is, when the index term occurs both in the document and in the query. Suppose, for example, $k = 2$ and the query contains term 1 only, that is, $c_1 = 1$ and $c_2 = 0$. Therefore,

$$\mathbf{x}^\top \cdot \mathbf{y} = a_1\; \mathbf{t}_1^\top \cdot \mathbf{t}_1 \; + a_2\; \mathbf{t}_2^\top \cdot \mathbf{t}_1$$

If $\mathbf{t}_2{}^\top \cdot \mathbf{t}_1 = 0.5$, that is, the terms are partially correlated, the distance between the document and the query is also affected by term 2, even though it does not occur in the query.

Latent Semantic Indexing (LSI) is a technique based on Singular Value Decomposition (SVD) which aims at finding a basis alternative to the canonical basis. The documents and the queries are then re-expressed with respect to this new basis. The potential of LSI is twofold. First, the document and the queries are vectors of a new space whose dimension is greatly reduced. Second, the new basis vectors no longer represent index terms, but sets of index terms that may be called "concepts," "clusters" or "aggregates" depending on the application domain.

Key Applications

The VSM has been subjected to an extensive evaluation and SMART is the best known experimental IR system. Thanks to the excellent results confirmed by several tests carried out within the Text Retrieval Conference (TREC), this model is presently the reference model for many applications whenever a best-match and weighted retrieval is required. The applications ranges in many tasks, including clustering, cross-language retrieval, summarization and personalization to name but a few. Salton and his colleagues' bibliography gives an idea of the broad range of applications.

Future Directions

The most challenging problem of Information Retrieval is to define a model that governs the complex interactions between the various components of a systems and the end users. An attempt in this direction has been made in [12], which has paved the way toward a new conception of the vector spaces and opened up some connections to other sciences.

Cross-references

- ▶ Boolean Model
- ▶ Clustering for Post Hoc Retrieval
- ▶ Indexing Units
- ▶ Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model
- ▶ Query Expansion Models
- ▶ Relevance Feedback
- ▶ Singular Value Decomposition
- ▶ Term Weighting

Recommended Reading

1. Deerwester S., Dumais S., Furnas G., Landauer T., and Harshman R. Indexing by latent semantic analysis. J. Am. Soc. Inform. Sci., 41(6):391–407, 1990.
2. Dubin D. The most influential paper Gerard Salton never wrote. Libr. Trends, 52(4):748–764, 2004.
3. Halmos P. Finite-Dimensional Vector Spaces. Undergraduate Texts in Mathematics, Springer, 1987.
4. Melucci M. A basis for information retrieval in context. ACM Trans. Inform. Syst., 26(3), 2008.
5. Salton G. Associative document retrieval techniques using bibliographic information. J. ACM, 10440–457, 1963.
6. Salton G. Automatic Text Processing. Addison-Wesley, 1989.
7. Salton G. Mathematics and information retrieval. J. Doc., 35(1):1–29, 1979.
8. Salton G. and Buckley C. Term Weighting Approaches in Automatic Text Retrieval. Inform. Process. Manage., 24(5):513–523, 1988.
9. Salton G., Wong A., and Yang C. A vector space model for automatic indexing. Commun. ACM, 18(11):613–620, 1975.
10. Salton G., Yang C., and Yu C. A theory of term importance in automatic text analysis. J. Am. Soc. Inform. Sci., 26(1):33–44, 1975.
11. Singhal A., Buckley C., and Mitra M. Pivoted Document Length Normalization. In Proc. 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1996. pp. 21–29.
12. van Rijsbergen C. The Geometry of Information Retrieval. Cambridge University Press, UK, 2004.
13. Wong S. and Raghavan V. Vector space model of information retrieval – a reevaluation. In Proc. 7th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1984, pp. 167–185.

Vertical Fragmentation

▶ Distributed Database Design

Vertically Partitioned Data

Jaideep Vaidya
Rutgers University, Newark, NJ, USA

Synonyms

Heterogeneously distributed data

Definition

Data is said to be vertically partitioned when several organizations own different attributes of information for the same set of entities. Thus, vertical partitioning of data can formally be defined as follows: First, define a dataset D in terms of the entities for whom the data are collected and the information that is collected for each entity. Thus, $D \equiv (E, I)$, where E is the entity set for whom information is collected and I is the feature set that is collected. Assume that there are k different sites, $P_1,\ldots,P_k$ collecting datasets $D_1 \equiv (E_1, I_1),\ldots,D_k \equiv (E_k,I_k)$ respectively. Therefore, data is said to be vertically partitioned if $E = \bigcap_i E_i = E_1 \bigcap\ldots\bigcap E_k$, and $I = \bigcup_i I_i = I_1 \bigcup\ldots\bigcup I_k$. In general, distributed data can be arbitrarily partitioned. Vertical partitioning can also be defined as a special case of arbitrary partitioning, where all of the partitions consist of information about the same set of entities.

Key Points

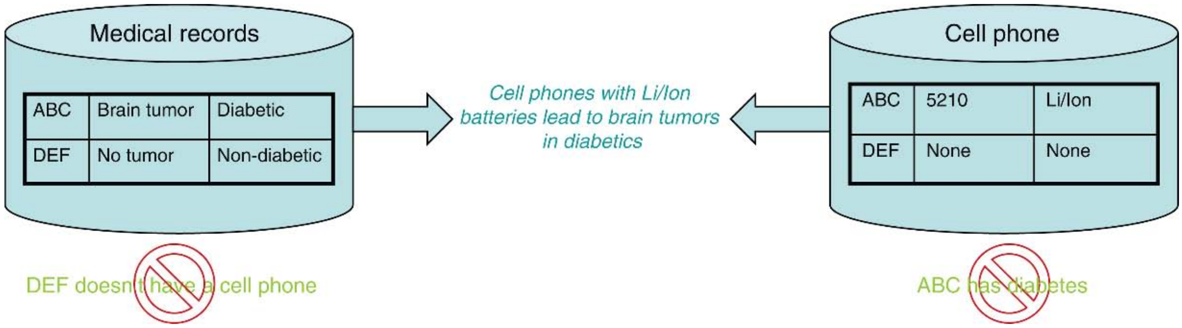
In general, data can be distributed in an arbitrary fashion. This means that different parties may own partial information about different sets of entities. While such arbitrary partitioning is possible, in practice, it rarely happens. Data is said to be vertically partitioned when different sites collect different features of data for the same set of entities. Integrating the local datasets gives the global dataset. Vertically partitioned data occurs naturally in many situations, and mining it can lead to unexpected insights. For example, consider the case of Ford and Firestone. Ford collects information about vehicles manufactured. Firestone collects information about tires manufactured. Vehicles can be linked to tires. This linking information can be used to join the databases. Why bring this up? In 2001, numerous accidents due to tread separation were reported. Initially both companies blamed each other. It turned out that it was only Ford Explorers with Firestone tires from the Decatur, Illinois plant, in specific situations that had these problems. If found out earlier, much loss could have been avoided. While both companies individually collect a lot of pertinent testing data, this was not shared due to commercial concerns. Mining of the global database had the potential to reveal this, which was otherwise impossible only with the local data.

Figure 1 demonstrates another example of vertical partitioning of data. There are two data sources – a hypothetical hospital/insurance company and a wireless service provider. The hospital collects medical records such as the type of brain tumor and diabetes (none if the person does not suffer from the condition). On the other hand, the wireless provider might collect other information such as the approximate amount of airtime used everyday, the model of the cellphone and the kind of battery used. Together, merging this information for common customers and running data mining algorithms might give completely unexpected correlations (for example, a person with Type I diabetes using a cell phone with Li/Ion batteries for more than 3 hours per day is very likely to suffer from primary brain tumors.). It would be impossible to get such information by considering either database in isolation. While this example is hypothetical, similar situations abound in practice.

In general, when data is vertically partitioned, more data significantly improves the quality of the models built from the dataset. Overall, the data analysis results are significantly more accurate and real. While this is also true of horizontally partitioned data (more data is always good), but it has a more critical impact with vertically partitioned data. This is because data from different parties give significantly different additional information about the entities. This is especially true with higher dimensional data.

The complexity of privacy-preserving data mining is also significantly increased due to the vertical partitioning of data. In contrast to horizontal partitioning of data, vertical partitioning of data raises several unique questions with respect to the way data is processed, results are obtained and shared. For example, consider the task of creating a decision tree classifier from some given vertically partitioned dataset. An immediate question is how is the final tree shared by the different parties? Since each party has knowledge of only some of the attributes, knowing the structure of the tree, and especially, knowledge of an unknown attribute and its breakpoints for testing – constitutes a violation of the privacy of the individual parties. Thus, completely revealing the tree is a bad idea from the security standpoint.

Indeed, in the best case, for no leakage of information, even the structure of the tree should be hidden, with an oblivious protocol for classifying a new instance. However, the cost associated with this is typically unacceptable. A compromise may be to


Medical records
Cell phones with Li/Ion batteries lead to brain tumors in diabetics
Cell phone
DEF doesn't have a cell phone
ABC has diabetes

ABC	Brain tumor	Diabetic
DEF	No tumor	Non-diabetic

ABC	5210	Li/Ion
DEF	None	None

Vertically Partitioned Data. Figure 1. A vertically partitioned dataset with hypothetical knowledge inferred from it.

cloak the attribute tests used in the tree while still revealing the basic structure of the tree. This way each site only need know the branch values for the decisions it makes, and the classification of a new instance would also have to be carried out in a distributed fashion by jumping from site to site. In general, it is necessary to carefully evaluate what may be the intermediate and final results, and only reveal as appropriate. Indeed, constructing the form of results is one of the major challenges, especially when one considers that multiple analyses may need to be carried out. [1] provides more details on problems, solutions and challenges in this area.

Cross-references

- ▶ Horizontally Partitioned Data
- ▶ Privacy-Preserving Data Mining
- ▶ Secure Multiparty Computation Methods

Recommended Reading

1. Vaidya J., Clifton C., and Zhu M. Privacy-Preserving Data Mining, 1st edn. Advances in Information Security 19, Springer-Verlag, Berlin, 2005.

Video

Ying Li
IBM T.J. Watson Research Center, Hawthorne,
NY, USA

Synonyms

Rich media; Multimedia

Definition

Video, which means “I see” in Latin, is an electronic representation of a sequence of images or frames, put together to simulate motion and interactivity. From the producer’s perspective, a video delivers information created from the recording of real events to be processed simultaneously by a viewer’s eyes and ears. For most of time, a video also contains other forms of media such as audio.

Video is also referred to as a storage format for moving pictures as compared to image, audio, graphics and animation.

Historical Background

Video technology was first developed for television systems, but it has been further developed in many formats to allow for consumer video recordings. Generally speaking, there are two main types of video: analog video and digital video. Analog videos are usually recorded as PAL (Phase Alternating Line) or NTSC (National Television System Committee) electric signals following the VHS (Video Home System) standard, and stored in magnetic tapes. Digital videos, on the contrary, are usually captured by digital cameras and stored in digital video formats such as DVD (Digital Versatile Disc), QuickTime and MPEG-4 (Moving Picture Experts Group).

Launched in September 1976, VHS became a standard format for consumer recording and viewing by the 1990s. Since then, it has dominated both home and commercial video markets. In March 1997, the DVD format was introduced to American consumers, which gradually pulled consumers away from VHS in following years due to its much better quality. In June 2003, the DVD’s market share exceeded that of the VHS for the first time. Since then, it has been steadily expanding its consumer market, and by July 2006, most major film studios have stopped releasing new movie titles in VHS format, opting for DVD-only releases. Now, VHS is gradually disappearing from both rental and retail stores, and DVD has dominated the whole commercial market. Nevertheless, VHS is still popular for home recording of television programs, due to the large installed base and the lower cost of VHS recorders and tape.

For the last few decades, as video technology quickly advances and the cost of storage devices rapidly decreases, digital videos have become widely available in diverse application areas such as medicine, remote sensing, entertainment, education and online information services. This has thus led to very active researches in various video-related areas.

Foundations

The last three decades have witnessed a significant amount of research efforts on various aspects of video technologies. Roughly speaking, they fall into the following three general categories: video representation, video content analysis, and video application. Specifically, video representation deals with the way a video is represented, in another word, the file format. Video content analysis, on the other hand, aims to automatically structurize and ultimately understand the video by analyzing its underlying content. Due to the difficult nature of this problem, such process usually involves the analysis of multiple media modalities including visual, audio and text information. Finally, video application applies what's learned from the analysis engine, and facilitates various types of content access including video browsing, summarization and retrieval. A brief discussion on each of these three research domains is given below.

Video Representation

A video sequence with accompanying sound track can occupy a vast amount of storage space when represented in digital format. As estimated in [6], a 1-min video clip could possibly occupy up to 448 MB. Consequently, compression has been playing an important role in modern schemes for video representation.

A wide variety of methods have been proposed to compress video stream. Nevertheless, almost all of them build their approaches upon the fact that video data contains both spatial and temporal redundancy. Specifically, to reduce the spatial redundancy, an intra-frame compression is applied which registers differences between parts of a single frame. Such a task is more closely related to image compression. Likewise, to reduce the temporal redundancy, an inter-frame compression is exploited which registers differences between neighboring frames. This involves discrete Cosine transform (DCT), motion compensation and other techniques.

Some popular video compression mechanisms include H.261, H.263, H.264, MPEG-1, MPEG-2, MPEG-4 and MJPEG (Motion-Joint Photographic Experts Group). Specifically, H.261 is a 1990 ITU-T (Telecommunication Standardization Sector of International Telecommunication Union) video coding standard originally designed for transmission over ISDN lines. Later on, H.263 and H.264 which provide more capabilities and mainly target at video-conferencing applications, were standardized in 1995 and 2003, respectively.

In 1998, the Moving Picture Experts Group (MPEG) was formed to establish an international standard for the coded representation of moving pictures and associated audio on digital storage media. Currently, there have been three established MPEG standards from this effort: MPEG-1, MPEG-2, and MPEG-4. Each of them targets at different commercial applications. For instance, MPEG-1 is usually used as the Video CD (VCD) format, MPEG-2 for High Definition Television (HDTV), and MPEG-4 for streaming video applications.

Finally, to facilitate mobile appliances such as digital cameras, MJPEG was developed in 1990s which uses intra-frame coding technology that is very similar to those used in MPEG-1 or MPEG-2. However, it does not use inter-frame prediction, which on one hand, results in a loss of compression capability, yet on the other hand, it makes the degree of compression capability independent of the amount of motion in the scene. Moreover, it also eases video editing as simple edits can now be performed at any frame.

Video Content Analysis

Video is a type of rich media as it often consists of other media types such as audio and text. Consequently, research on video content analysis can be grouped into three classes: visual content analysis, audio content analysis, and audiovisual content analysis. A general goal of video content analysis is to extract the underlying video structure so as to facilitate convenient and nonlinear content access. Yet a more aggressive goal is to automatically understand video semantics so as to support applications such as video summarization and retrieval that require an in-depth understanding of the video content.

Visual Content Analysis

As the name implies, visual content analysis concentrates on processing the visual part of the video signal. Most existing work in this area falls into two directions, with one focusing on analyzing the image or frame content in the spatial domain, and the other on exploiting the temporal relationships between frames.

Visual feature extraction at different semantic levels are usually the major focus of those work in the first category. Popular features such as color, texture and motion, are low-level features, which while easy to extract, are not able to capture the video semantics. Therefore, most of recent work focuses on extracting mid to high level visual features, which can not only be derived from lower level features, but also capable of bridging the semantic gap by revealing the underlying content semantics to a certain degree. Such features include various types of video objects and semantic concepts such as sports, military, explosion and sky. Machine learning and content modeling are two popularly applied approaches to achieve such goal.

On the other hand, there is also a significant amount of work that attempts to capture both video syntax and semantics by exploring its temporal structure. Specifically, video shot change detection is usually the very first step towards this goal, where the entire video sequence is segmented into a series of cascaded shots. Shot forms the building block of a video sequence, and contains a set of frames that are continuously taken. For a comprehensive survey on various shot detection algorithms, readers can refer to [3].

Based on the syntactic shot structure, higher-level visual content analysis such as scene detection or extraction, could be subsequently carried out. Figure 1 shows a typical video content structure, which is represented by a hierarchical tree. As shown, for an incoming video stream, it is first segmented into a sequence of *shots*. Then, one or more of its frames (called *keyframes*) can be extracted to represent its underlying content. The next step is to build semantic video scenes upon the extracted shot structure. Specifically, a video *scene* is defined as a collection of semantically related and temporally adjacent shots that depicts and conveys a high-level concept or story. Consequently, a common solution to scene extraction is to group semantically related shots into a scene [14]. This process is similar to grouping words into sentences, where a sentence starts conveying meanings to readers. Finally, there could be one or more steps further from scene detection. One such effort is on event detection, where an *event* is considered as important scenes that contain particular thematic topics of interest such as dialogs, actions or something abnormal [3,5].

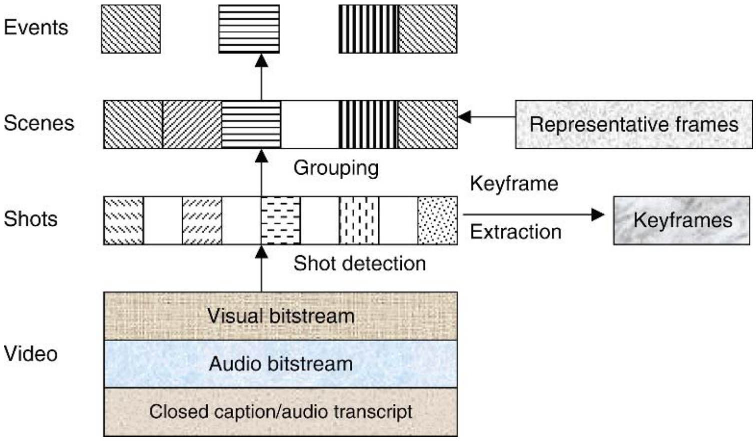
Audio Content Analysis

Audio, which includes voice, music and various kinds of environmental sounds, is an important type of media. A general approach to audio content analysis is to either classify or segment an audio clip into different homogeneous fragments with each fragment containing one particular audio type such as speech, music or silence [2]. Audio features such as short-time energy, short-time zero-crossing rate and short-time fundamental frequency, are usually extracted for this analysis purpose [15].

As a key ingredient of a video stream, the audio source could represent the story in a much simpler fashion than its pictorial counterpart. For instance, if people are only allowed to listen to a TV program without watching it, they can still get most of its meanings. Yet on the other hand, if they are only allowed to watch the program without listening to it, they may get lost easily. This shows that the audio source, especially the embedded speech cue, is critical to a human's content understanding. Moreover, audio is also essential to people's enjoyment of the video content.

Audiovisual Content Analysis

As automatic video content understanding is a very challenging task, thus most recent work tends to exploit all possible media cues to achieve such a goal. In particular, as audio and visual cues are two inseparable parts of a video stream, and usually complement with each other during the content creation, people tend to integrate them together during the content analysis.



Video. Figure 1. A hierarchical representation of video content.

A general solution to audiovisual content analysis is to first perform individual visual or audio content analysis to obtain two sets of analysis results, then combine them together using certain fusion rules or applying a probabilistic framework. Another popular way of media integration is to employ different information sources at different processing stages [4]. For instance, the visual cue is first employed to generate coarse-level results, then audio cues is introduced to refine the results. However, when and how to efficiently and effectively integrate multiple media sources still remains to be an open issue, and needs further study.

Video Application

Besides the large amount of research efforts on video content analysis, there are also many attentions on studying various video applications. After all, making the bulky and unstructured video content convenient and efficient to access, present, share, search and deliver is the ultimate goal of the entire research community in this area. Below, a brief introduction is given to three major types of video applications, namely, video indexing and browsing, video abstraction, and content-based video retrieval.

Video Indexing and Browsing

As analog to indexing a book, video indexing aims at facilitating non-linear access of the video content. It is thus very critical for efficient video retrieval. For instance, television and film archives usually contain a vast amount of audiovisual materials. If these materials are properly segmented and indexed, studios can conveniently produce a new video clip by finding and reusing some pre-produced segments. Moreover, in audiovisual libraries or family entertainment applications, it would be very desirable if the needed video segments from a large video collection could be quickly located.

Video browsing refers to the activity where a user watches through a video to get quick ideas of its underlying content [11]. Video browsing can also assist users in query forming for video retrieval purpose. For instance, when the user lacks a clear idea of what he wants from a large video collection, he can gradually conceptualize his needs by browsing through the video clips to find the one that stimulates his desire. Once such clip is located, he can further submit it as a query seed to obtain more related clips.

Video Abstraction

Video abstraction, as the name implies, generates a short summary for a long video document. Specifically, a video abstract is a sequence of still or moving images, representing the video content in a way such that the target party is rapidly provided with concise information about the content, while the essential message of the original is well preserved [12]. Video abstraction is primarily used for video browsing, and is an inseparable part of a video indexing and retrieval system.

Theoretically, a video abstract can be generated both manually and automatically, but due to the huge volumes of video data and limited human power, it is getting increasingly important to develop fully automated video analysis and processing tools so as to reduce human involvement in the video abstraction process.

There are two fundamentally different kinds of video abstracts: still- and moving-image abstracts. The still-image abstract, also known as static storyboard or *video summary*, is a small collection of salient images (also called keyframes) extracted or generated from the underlying video source. The moving-image abstract, also known as moving storyboard or *video skim*, consists of a collection of image sequences, as well as the corresponding audio abstract extracted from the original sequence. Thus, it is itself a video clip but is of a considerably shorter length.

There exist some significant differences between video summary and video skim. First, a video summary can be built much faster, since generally only visual information is utilized and no handling of audio and textual information is needed. Also, once composed, it can be displayed more easily since there are no timing or synchronization issues. Second, more salient images such as mosaics could be generated to better represent the underlying video content. Third, the temporal order of all extracted representative frames can be displayed in a spatial order so that the users are able to quickly grasp the video content. Finally, all extracted stills could be easily printed out when needed.

There are also advantages in using video skim. Compared to the still-image abstract, a moving-image abstract usually makes more sense to users since it preserves part of the original audio track. Moreover, the possibly higher computational effort during the abstracting process pays off during the playback time: it is usually more natural and more interesting for users to watch a trailer than watching a slide show, and in many cases, the motion is also information-bearing.

A comprehensive survey of various video summarization and video skimming techniques could be found in [3].

Content-Based Video Retrieval

By definition, a content-based video retrieval system (CBVRS) aims at assisting a human operator or user to retrieve a video sequence within a potentially large collection based on certain criteria. Generally speaking, there are three major aspects regarding a CBVRS, namely, *query formation*, *feature space selection*, and *similarity measurement*. In particular, the query formation deals with formulating a meaningful and clear query which faithfully captures what the user is looking for. Once the query is submitted, potential candidates will be evaluated within certain feature space, and the similarity between the query and candidate is measured. Such a process usually results in a ranked list of retrieved sequences sorted from the most similar to the least. Below gives a brief discussion on each of the above three aspects.

To formulate a query, the user can either submit still images (*a.k.a.* keyframes), short video sequence, textual keywords, or a combination of the above as the search seeds. Query formation is not as easy a task as it appears to be. One major reason for this is due to the semantic gap between the user's true intention and the constrained way of representing it. For instance, when the user submits a video sequence as a query example, he or she may mean to find some videos that present similar color schemes, motion, events, or objects, yet such intention is hard to be captured or understood by the system as it is embedded in the query and is highly subjective. To close such a semantic gap, one way is to use textual keywords in queries that precisely describe what the user is looking for. Nevertheless, this would require a thorough and correct textual annotation of the entire collection, which is not only very tedious but in fact, impossible for humans to perform. Recently, there have been some ongoing research efforts on automatic video annotation [10], yet many challenging issues still remain to be solved.

As another effort on bridging such a semantic gap, people have been introducing the so-called relevance feedback mechanism into the system, where the user interacts with the system by giving negative or positive feedback on the retrieved results [13]. The system then performs an online learning of the user's expectations by possibly adjusting or updating the weights of different features, the similarity metrics, and the learning algorithms. While there has been an increasing amount of work on this subject, how to design an efficient, effective and user-friendly relevance feedback system still deserves further study.

Once the query is formulated and submitted, the next step is to measure the similarity between the query example and possible candidates within certain feature space. Undoubtedly, finding a good set of features to represent the query is very critical to the success of the search. So far, various features at different semantic levels have been proposed including color, texture, motion, audio, object, scene, event etc. in both spatial and temporal domains [16].

Finally, a metric is required to measure the similarity, which produces a distance that would account for both the spatial and temporal differences between the two video sequences. Ideally, such metric should be jointly defined with the particular feature set that is applied in the system for a better result [1].

In a word, while at a first glance, a content-based video retrieval system is a natural extension of a content-based image retrieval system, it is in fact, far more complex due to the excessive dimensionality of the search space induced by the inherent temporal information in videos.

Key Applications

In October 1998, MPEG started a new work item called the "*Multimedia Content Description Interface*," or in short "MPEG-7," which aims to specify a standard set of descriptors and description schemes that can be used in describing various types of multimedia information [9]. This description shall be associated with the content itself, to allow fast and efficient search for materials of users' interests. The major objective of MPEG-7 is to make audiovisual material as searchable as text [8].

Various cases of professional and consumers applications have been identified and targeted by MPEG-7, which can be categorized into either *pull* or *push* applications [7]. Specifically, typical pull applications that are more related to video include: (i) storage and retrieval of video databases; (ii) delivery of pictures and video for professional media production, and

(iii) movie scene retrieval by memorable auditory events. In contrast, push applications follow a paradigm more akin to broadcasting and webcasting. Some key applications include: (i) user agent driven media selection and filtering; (ii) personalized television services; (iii) intelligent multimedia presentation; (iv) bio-medical applications; (v) remote sensing applications; (vi) semi-automated multimedia editing; (vii) educational applications, and (viii) surveillance applications.

Undoubtedly, with the amount of audiovisual information rapidly increasing and becoming widely available from many sources around the world, a lot more new applications will emerge so as to satisfy various urgent needs related to important academic, social and economic issues.

Cross-references

▶ Audio Content Analysis
▶ Content-Based Video Retrieval
▶ Image Retrieval
▶ Image Retrieval and Relevance Feedback
▶ Image Content Modeling
▶ Mid-to-High-Level Image Content Analysis
▶ Video Content Analysis
▶ Video Content Modeling
▶ Video Content Structure
▶ Video Representation
▶ Video Scene and Event Detection
▶ Video Segmentation
▶ Video Shot Detection
▶ Video Summarization
▶ Visual Content Analysis

Recommended Reading

1. Cheung S. and Zakhor A. Efficient video similarity measurement with video signature. IEEE Trans. Circ. Syst. Video Tech., 13(1):59–74, 2003.
2. Li Y. and Dorai C. SVM-based audio classification for instructional video analysis. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, 2004.
3. Li Y. and Kuo C.-C. Video Content Analysis Using Multimodal Information: for Movie Content Extraction, Indexing and Representation. Kluwer, MA, USA, 2003.
4. Li Y., Narayanan S., and Kuo C.-C. Content-based movie analysis and indexing based on audiovisual cues. IEEE Trans. Circ. Syst. Video Tech., 14(8):1073–1085, 2004.
5. Mahmood T.S. and Srinivasan S. Detecting topical events in digital video. In Proc. 8th ACM Int. Conf. on Multimedia, 2000, pp. 85–94.
6. Mitchell J., Pennebaker W., Fogg C., and LeGall D. MPEG Video Compression Standard. Chapman & Hall, New York, NY, USA, 1992.
7. MPEG Requirements Group, MPEG-7 Applications Document v.8, ISO/MPEG N2860, MPEG Vancouver Meeting, July 1999.
8. MPEG Requirements Group, MPEG-7 Context, Objectives and Technical Roadmap, ISO/MPEG N2861, MPEG Vancouver Meeting, July 1999.
9. MPEG Requirements Group, MPEG-7 Requirements Document V.15, ISO/MPEG N4317, MPEG Sydney Meeting, July 2001.
10. Nock H., Adams W., Iyengar G., Lin C., Naphade M., Neti C., Tseng B., and Smith J. User-trainable video annotation using multimodal cues. In Proc. 26th Annu. Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 403–404.
11. Oh J. and Hua K. Efficient and cost-effective techniques for browsing and indexing large video databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 415–426.
12. Pfeiffer S., Lienhart R., Fischer S., and Effelsberg W. Abstracting digital movies automatically. J. Vis. Comm. Image Represent., 7(4):345–353, 1996.
13. Yan R., Hauptmann A., and Jin R. Negative paeudo-relevance feedback in content-based video retrieval. In Proc. 11th ACM Int. Conf. on Multimedia, 2003, pp. 343–346.
14. Yeung M., Yeo B., and Liu B. Extracting story units from long programs for video browsing and navigation. In Proc. Int. Conf. on Multimedia Computing and Systems, 1996, pp. 296–305.
15. Zhang T. and Kuo C.-C. Audio content analysis for on-line audiovisual data segmentation. IEEE Trans. Speech Audio Process., 9(4):441–457, 2001.
16. Zheng W., Li J., Si Z., Lin F., and Zhang B. and Using high-level semantic features in video retrieval. In Image and Video Retrieval. Springer, Berlin Heidelberg, New York, 2006, pp. 370–379.

Video Abstraction

▶ Video Summarization

Video Analysis

▶ Video Content Analysis

Video Annotation

▶ Video Metadata

Video Chaptering

▶ Video Segmentation

Video Compression

▶ Video Representation

Video Content Analysis

ALEXANDER HAUPTMANN
Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

Video analysis; Video content processing; Semantic analysis of video

Definition

Video content analysis deals with the extraction of metadata from raw video to be used as components for further processing in applications such as search, summarization, classification or event detection. The purpose of video content analysis is to provide extracted features and identification of structure that constitute building blocks for video retrieval, video similarity finding, summarization and navigation. Video content analysis transforms the audio and image stream into a set of semantically meaningful representations. The ultimate goal is to extract structural and semantic content automatically, without any human intervention, at least for limited types of video domains. Algorithms to perform content analysis include those for detecting objects in video, recognizing specific objects, persons, locations, detecting dynamic events in video, associating keywords with image regions or motion patterns, identifying visible actions or behaviors, and labeling scenes, activities, genres. The analysis may be performed on single frames (images), sequences of frames indicating change or motion by the camera or the subject, audio analysis through speech recognition and non-speech sound characterization as well as any combinations of still image analysis, image sequence analysis and audio analysis.

Historical Background

Digital video has proliferated dramatically since the 1990s, ranging from ever-growing personal video collections to professional news and documentary archives. As bandwidth and disk space accessible to users is increasing, video is becoming the most rapidly proliferating type of data on the Internet. Fueled by the popularity of social media sharing in the Web 2.0 paradigm, there has been exponential increase in videos available on the internet. With the availability of large amounts of multimedia data, there comes an urgent need for good video content analysis.

The first attempts at image and video content analysis were made in the 1960s and 1970s, with applications mostly in video compression and videophones. By the mid 1990s, a number of research systems that could perform video content analysis on sequences longer than a few minutes were introduced. With the widespread introduction of fast computers in the twenty-first century, digital video content analysis has become common, but is frequently not accurate enough for widespread commercial use, and the more sophisticated approaches are still too computationally complex to be applied to every frame in the video.

Video content analysis has become the only feasible way to analyze larger video collections for search, summarization, navigation, and re-use in other applications. Social media tagging has allowed collections on the internet to be searchable through text tags and comments, but these searches rely on precision, and reasonable tagging, which is often not the case, especially for some of the less popular video files.

Foundations

Video content analysis algorithms are usually built within a machine learning framework, where the system is first provided with a set of training instances that represent the type of analysis to be performed (e.g., soccer goal scoring), as well as negative training instances (normal play) and any of a large set of machine learning techniques is used to automatically determine which video analysis features are useful to predict this higher level analysis. However, the most impressive results in video content analysis are achieved when judicious constraints, rules or heuristics exploit the particular characteristics of the analysis to be performed, limiting the number of features that are considered and improving the accuracy of the analysis or classification result.

V

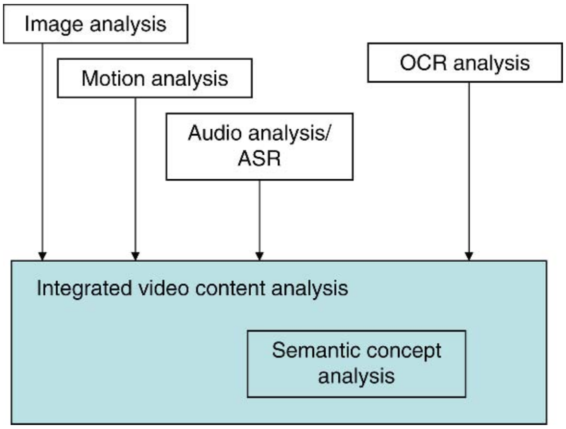
Much of the most interesting video content analysis is based on a fusion of analysis results at different levels of the different modalities (image, audio, motion, OCR). This type of content analysis has produced the most interesting results in sports video analysis, where the crowd and announcer noise indicates an interesting event on the field, while the image and motion analysis gives a clue to the type of event, and the OCR might indicate the players involved as well as the score.

Similarly, for broadcast video, a combination of image analysis for faces, audio analysis and camera analysis can determine which shots contain interviews with news subject, and OCR together with face recognition provides the identify of the person in the news. Video content analysis combining image similarity based on duplicate detection, identification of news studio shots and news anchor scenes can be used to provide the building blocks for news story segmentation. In order to summarize and browse through "scenes" or stories quickly, it is convenient to assign them to particular topic classes, which can again be done be integrating the results of different components of video content analysis.

In many ways, video content analysis fundamentally builds on image content analysis. Each frame in a video can be considered an image by itself, so all aspects of an image retrieval system can be applied. Text annotations may be available and can be searched as well. But video offers additional challenges and opportunities.

Video consists of additional types of information (e.g., audio sounds, human speech, motion) beyond what one might find in pure still image data. While a collection of a million images is considered quite large, just 10 h of video at 30 frames per second would provide a million frames (images). Of course, consecutive frames are often strongly related, providing high similarity and numerous slightly different versions of the same content in adjacent frames. However, the large amount of data, puts the burden on video content analysis to provide effective abstractions. The purpose of video content analysis is to provide extracted features and identification of structure that constitute building blocks for video retrieval, video similarity finding, summarization and navigation (Fig. 1).

Broadcast video consists of edited programs, whose structure can be exploited. Thus, news video consists of anchors, news stories, reporters, advertisements, etc. with each category providing context and constraint to what might be of use in further analysis. Videos can generally be partitioned temporally into keyframes, scenes, stories, and programs.



Video Content Analysis. Figure 1. Integrated video content analysis builds on Image, audio, motion and OCR analysis.

Partitioning Video

One of the earliest and most successful applications of video content analysis was the detection of shot boundaries. Shot boundary segmentation involves the identification of cuts, fades, and dissolves as typical editing effects added to video. Most techniques use sudden color, texture or edge changes and feature point tracks over various window sizes to determine shot breaks. In *movie or television video*, additional segmentation may aggregate several related shots into a scene or video paragraph, based on the similarity between sequences of shots.

In the case of *broadcast news video*, it is usually possible to segment a news story as a possible semantic unit, based on consistency in the text transcript, detection of news anchors, audio speaker analysis, and similarity in the news story footage. Special segmentation techniques involving black frames and rapid scene changes are used to identify commercial advertisements. Program or station logos, special transition effects, and recognition of distinctive backgrounds such as weather maps can also help determine news story boundaries.

Surveillance video has very different characteristics, since there are no editing artifacts, such as shot boundaries. Here, keyframe extraction is still desirable, but keyframes are usually selected from sections of video that contain motion, detected objects and people. In surveillance video, the best representative keyframe captures an unusual event or a person/object when it is most clearly visible (largest) in the frame.

Audio Analysis

Audio analysis provides one component of video content analysis, supplementing visual analysis. From the audio track, it is possible to automatically generate a transcript to enable text-based retrieval from spoken language documents. Audio analysis can also identify changes in speakers and identify individual speakers.

Speech Recognition for Video Content Analysis and Retrieval The consensus from a number of published experiments in the area of automatic speech recognition for broadcast retrieval applications is that as long as speech recognition has a word error rate less than 35%, then information retrieval from the transcripts of spoken documents is only 3–10% worse than information retrieval on perfect text transcriptions of the same documents. The most accurate recognizers to date produce a word error rate under 20% for broadcast news, suitable for very accurate text-based retrieval.

Speaker and Audio Type Identification Another component of video content analysis is the use of audio features and speaker labeling. Many systems use audio analysis techniques to automatically extract additional metadata providing a description of the content of the audio channel. Audio processing can be used to detect music and other non-speech sounds, e.g., laughter and applause. Audio processing techniques can also be used to distinguish male versus female speakers, so that the respective audio regions can be passed to more accurately trained gender-specific speech recognizers. Existing audio analysis systems also identify well-known speakers for which a significant amount of training data is available, or group speakers across different speaker turns. Thus, for example, the techniques could identify the US President as well as most news anchors. Infrequently appearing speakers can be tracked when reappearing at different times within a video broadcast, although not labeled by name. This speaker change classification can be used to help break the video into coherent segments, and in general to characterize the audio channel of the data. Thus, video where one talker speaks for a long time can be labeled as a "speech," whereas an "interview" consists of two people switching back and forth, while a "forum" discussion includes many speakers. For broadcast data, it is possible to detect which speaker remains present throughout the entire audio data, thus identifying the anchor or narrator.

Content Analysis from Individual Image Frames

Most image content analysis for video usually happens on extracted keyframes. The full palate of image content analysis techniques can be applied, including low-level image content analysis, mid/high-level content analysis, image content modeling, image salient point extraction and representation, image segmentation and images similarity analysis, as described in the relevant sections.

Metadata Extraction Unique to Video

There are several types of content analyses that are unique to video.

Motion Feature Analysis One additional feature type not present in images is related to *motion*. There are several types of motion estimators that can be extracted, the motion blocks typically used in MPEG-style compression, the overall kinetic energy in the image and a fine-grained motion analysis based on optical flow. MPEG motion vectors can be directly extracted from the compressed result of an MPEG encoding process. These motion vectors can be aggregated and used for object and camera motion assessment. Kinetic energy measures the pixel variation within the shot. Optical flow algorithms typically warp one frame into another frame. They estimate the movements of high intensity interest points between two image frames as the motion vectors. While expensive to compute, optical flow analysis tends to provide the most refined estimates of object motion.

Recognizing object motion in video sequences is an important component in many applications. For example, in computer vision systems it enables the identification and tracking of the objects that make up a scene; while in video data compression it provides a means of reducing redundancy – knowing the motion of an object allows its position in successive frames to be predicted, removing the need to retransmit identical frame data and leading to a reduction in the bit rate required to transmit the video. Other applications include the generation of panoramic images, and recovery of 3-dimensional structure.

Recovering detailed motion from video sequences is a non-trivial effort. Factors such as the ambiguity resulting from 3-D action recorded as a 2-D projection by the camera, poor contrast, and low spatial or temporal resolution, require sophisticated techniques to obtain reliable motion information. The range of approaches used includes affine models to estimate and track complex 2-D motions and techniques to deal with multiple component motion regions.

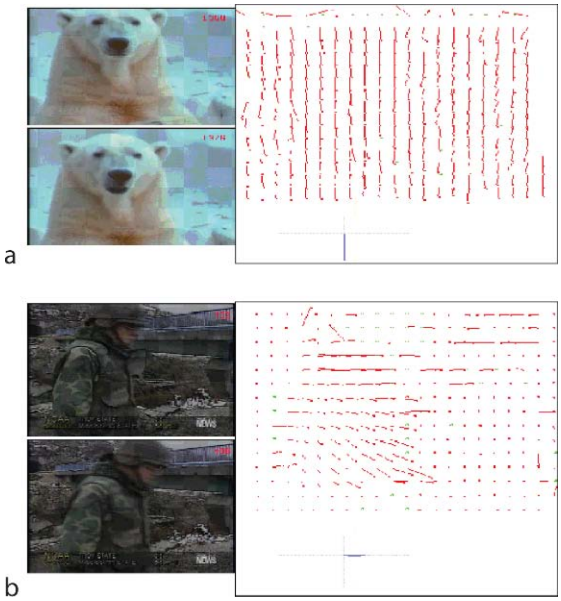
It has become popular in sports motion analysis to videotape subjects with markers placed on critical body parts, joints or equipment. The markers are used in each movement as references to the particular point of interest. Typically, the camera is carefully calibrated and distance markers placed in the field of view to allow accurate estimates of motion distances and velocities. This motion analysis allows customized systems to create models of motion and performance analysis relative to a specific sports activity. The motion may be translated into either 2-dimensional or 3-dimensional models and compared to reference motions.

Camera Motion Analysis One facet of video content analysis is concerned with analysis of camera motion, which includes pan (left/right), tilt (up/down), and zoom motions. More subtle analysis tries to distinguish dolly and boom motion, where the camera is physically moved from one place to another from the situation where the camera stays in place. Even further refinement of camera motion analysis tries to detect camera "shake," used for image stabilization. In principle, camera motion can also include rotation around a focal point; however this is a relatively infrequent event.

A wide variety of techniques exist for estimating the camera motion, most of which depend on the comparison between adjacent frames, based either on all pixels in the image or particularly salient pixels or regions. Since these pixels or regions only move slightly from frame to frame, their difference can be interpreted as a "track." From the overall patterns of tracks, models can be built to estimate if the motion pattern is more consistent with object motion, or camera shake, or camera pan/tilt/zoom, etc.

Most methods approach the problem of camera motion by deriving parameters proportional to the zoom, pan, rotate and tilt given a set of motion tracks. The methods usually determine an empirical threshold that classifies the derived parameters as representative for a particular camera motion. Optical flow analysis methods computed between consecutive images rely on tracking a large number of characteristic points across adjacent frames. Other methods use the MPEG motion vectors which encode one quantized motion vector per macro-block of pixels as an alternative to optical flow. Though these motion vectors are not directly equivalent to the true motion vectors of a particular pixel in the frame, there are typically enough motion vectors to robustly estimate camera motion parameters (Fig. 2).

Video OCR (VOCR) A somewhat different representation is derived by interpreting the text that is present in video images using optical character recognition (OCR). However, reading text present in the video stream requires a number of processing steps in addition to mere character recognition (see Fig. 3). First the text must be detected as present in a wide variety of scenes and backgrounds. Then it must be extracted from the image and finally converted into a binary (black and white) representation, since the commercially available OCR engines do not recognize colored text on variably colored background. The video OCR is further complicated because the text has very low resolution, frequently only about 10 pixels of height per character. Unlike text printed on white paper, the background of the image tends to be complex, with the character hue and brightness very near the background values. Among the solutions to these problems are



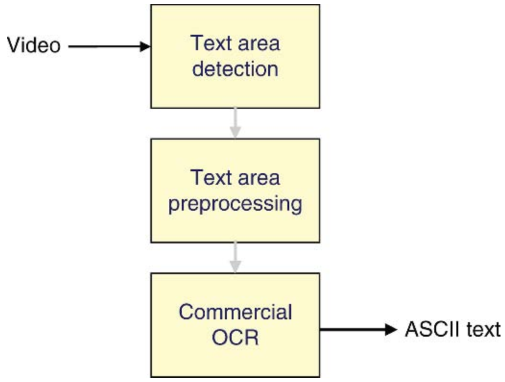
Video Content Analysis. Figure 2. Camera motion analysis for camera pan/tilt versus object motion using MPEG motion vectors.

interpolation filters, the integration of images across multiple frames and combinations of filters.

A text region detection is performed first, searching for horizontal rectangular structures of clustered sharp edges using variable orientation differential filtering techniques. Text boxes are identified based on their aspect ratio, absolute size and the fill factor of the bounding boxes.

Once a text area is detected, enhancement takes place. Multi-frame integration looks at the potential bounding boxes over several frames and finds the minimal (white) pixel values across that range. Potential text regions are sequentially filtered across consecutive frames, effectively increasing the resolution of each caption. Sub-pixel interpolation is performed to increase resolution without incurring jagged edges as artifacts (Fig. 4).

Adaptive thresholding on the gray-scale histogram is then used to create binarized black on white text before submitting it to an optical character recognition package. Systems will run OCR on multiple consecutive frames where text was detected, obtaining several nearly identical OCR results for a single occurrence of text on the screen that might last for a few seconds. These results can then be merged together for a single estimate. Once text is recognized, post-processing can correct some of the inevitable errors, e.g., using dictionary spelling correction.



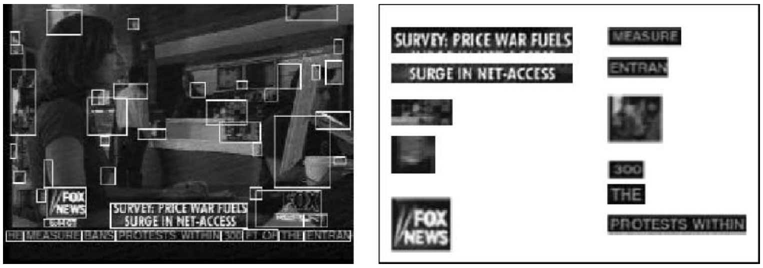
Video Content Analysis. Figure 3. Video OCR block diagram consisting of text area detection, text area preprocessing, and commercial OCR.

Combination of Audio Analysis, Image Analysis, and Motion Analysis for Semantic Content Analysis

Ultimately, video content analysis tries to achieve many of the same goals as mid-level and high level image analysis, except that the low level features available come from any number of modalities (audio, image, motion) and have already been performed at low, mid or high levels for this modality. The main task for video content analysis is then to fuse or combine these building blocks into a new characterization of the video.

Semantic Concepts as Mid or High-level Video Features. The effectiveness of low-level image analysis to represent image or video content is usually limited. The representations typically result in the notorious semantic gap between what users or applications expect and what content analysis systems can return, due to the systems' inability to capture the semantic meaning of the video content.

To provide a more semantic content analysis in video, an intermediate (mid to high-level) layer of hundreds of semantic concepts has been introduced in an effort to capture the semantic content of multimedia documents. The typical concepts includes a wide range of topics such as those related to people (face, anchor, etc.), objects (cars, buildings, bridges, graphics), location (outdoors, city, office setting), and genre (weddings, meetings, sports). Beyond the semantic concepts that are common to still image content analysis, semantic concepts for video can include temporally defined



Video Content Analysis. Figure 4. Candidate text regions as detected in a single frame of a video and some of the extracted individual text line.

events, such as an airplane landing, a vehicle stopping, etc. The successes of automatic semantic concept detection in recent years have demonstrated that a large number of high-level semantic concepts can be derived from the low-level multi-modal features with reasonable detection accuracy.

Detectors or classifiers for such semantic features are usually built off-line using large amounts of labeled training data combined with a number of machine learning approaches. Accuracy of the detectors depends critically on the number of training examples available as well as the difficulty of the semantic concept to be detected, e.g., a "left thumb" would be much more difficult to detect than a face. Face detection has been most effective for image and video retrieval, and significant added benefit can be gained from accurate face recognition, depending on the quality of the available images.

More recently, it has been proposed to combine these intermediate level semantic concepts into an ontology, which defines the relationships between concepts, their taxonomic structures and constraints. While researchers have shown some benefit to retrieval accuracy when using ontologies for limited domains, it is still unclear if a general-purpose ontology of visual concepts can be built and successfully applied. Similar efforts in the text analysis area have not yielded the expected results so far.

Key Applications

Search, browsing and summarization in video archives, real-time surveillance monitoring.

Cross-references

▶ Audio
▶ Audio Content Analysis
▶ Audio Segmentation
▶ Image Content
▶ Image Content Modeling
▶ Image Salient Points and Features
▶ Image Segmentation
▶ Image Similarity
▶ Low-Level Image Content Analysis (Color, Texture, Shape)
▶ Mid-to High-Level Image Content Analysis
▶ Video
▶ Video Content Modeling
▶ Video Content Structure
▶ Video Metadata
▶ Video Representation
▶ Video Scene and Event Detection
▶ Video Shot Detection

Recommended Reading

1. Chang S. and Sundaram H. Structural and semantic analysis of video. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2000, pp. 687–690.
2. Hanjalic A. Content-Based Analysis of Digital Video. Kluwer Academic, Boston, 2004.
3. Jay K.C. Video Content Analysis Using Multimodal Information: for Movie Content Extraction, Indexing and Representation. Kluwer Academic, Norwell, MA, USA, 2003.
4. Marques O. and Furht B. Content-Based Image and Video Retrieval. Kluwer Academic, Norwell, MA, USA, 2002.
5. Multimedia Image and Video Processing, L. Guan, S.Y. Kung, J. Larsen (eds.) Multimedia Image and Video Processing, CRC, Boca Raton, FL, USA, 1999.
6. Naphade M.R. and Smith J.R. On the detection of semantic concepts at TRECVID. In Proc. 12th Annu. ACM Int. Conf. on Multimedia, pp. 660–667.
7. Smeulders A.W., Worring M., Santini S., Gupta A., and Jain R. Content-based image retrieval at the end of the early years. IEEE Trans. Pattern Anal. Mach. Intell., 22(12):1349–1380, 2000.
8. Smith M.A. and Kanade T. Multimodal Video Characterization and Summarization. Kluwer, 2005. Series in Video Computing, Vol. 9.
9. Snoek C., Worring M., and Hauptmann A.G. Learning rich semantics from news video archives by style analysis. ACM Trans. Multimedia Comp., Comm., and Appl., 2(2):91–108, 2006.
10. The Informedia Digital Video project http://www.informedia.cs.cmu.edu
11. Worring M. and Snoek C.G. Semantic indexing and retrieval of video. In Proc. 14th Annu. ACM Int. Conf. on Multimedia, pp. 13–13.

Video Content Description

▶ Video Metadata

Video Content Modeling

Lei Chen
Hong Kong University of Science and Technology,
Hong Kong, China

Synonyms

Video data modeling

Definition

Video Content Modeling refers to representing the content of video data for search later on. Specifically, the content of video data includes the visual features, the temporal features, the contained objects, and the semantic concepts. With an effective modeling technique, people cannot only browse the video data, but also search the video with the specific features. Video content modeling is the basic for video data indexing and retrieval.

Historical Background

Video, as a popular type of multimedia, has been widely used by movie/TV industries and individuals. In the earlier 90s, people started search video data through annotated text information [20,21,23]. However, the low efficiency of manual annotation techniques prevents the text-based retrieval techniques applying to video data on a large scale. Thus, content-based video retrieval was proposed and studied. For the purpose of conducting effective and efficient search, significant works have been conducted on modeling video content. These techniques can be classified into three categories:

Segmentation-based models [24,18,12,8].
Video are recursively broken down into *scenes*, *shots* and *frames*. Key frames are extracted from shots and scenes to represent them, and visual features are extracted from key frames as indexes to key frames.

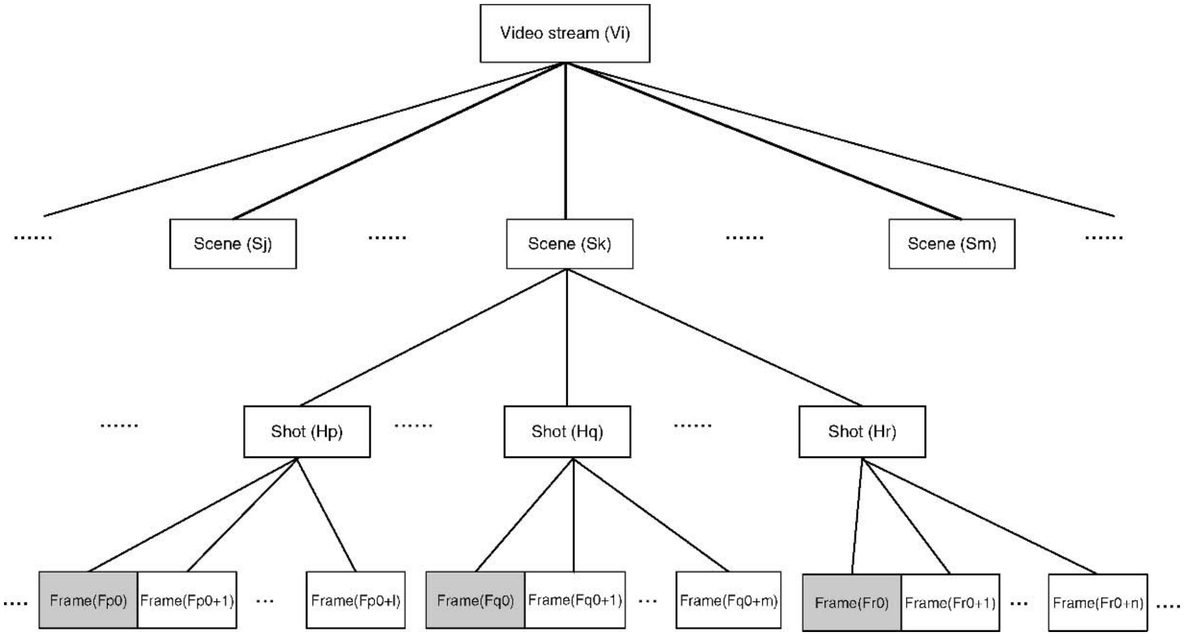
- Annotation-based models [20,23,22,14,3].
  In this model, a content description (annotation) layer is put on top of the video stream. Each annotation can be associated with a logical video sequence or physically segmented shots or scenes. An annotation describes the events or semantics of a video sequence.
- Salient object-based models [13,4,10,16,5,19]. Salient objects are extracted from video, and their audio-visual features and spatio-temporal relationships among them are described to express events or concepts.

Foundations

- Segmentation-based video data modeling:
  Modeling video data based on segmentation can be divided into three steps. First, video is segmented into *shots* (sometimes called *clips*); second, key frames are selected to represent the shots; finally, based on these shots, *scenes* or *story units* are constructed. During the last step, keyframes may also be selected to represent scenes and story units. Usually,visual features are extracted from the key frames as indexes to the video data. A video data model based on video segmentation possess a hierarchical structure (Fig. 1): a video stream contains several scenes or story units, each scene contains a set of shots and each shot contains asequence of video frames. The shaded frames in Fig. 1 are examples of key frames. The definition of shots, key frames and scenes are as follows [18]:
  - A shot is an unbroken sequence of frames recorded from a single camera.
  - A key frame is the frame selected from a shot which can represent the salient content of that shot.
  - A scene is a sequence of shots which conveys a concept or story.

The techniques for shot detection, key frame selection and scene construction, as briefly reviewed, follows:

- Video shot detection
  There are two basic types of video shot transitions: *abrupt* and *gradual* [24]. Abrupt transitions (cuts) occur when stopping and restarting cameras. Gradual transitions are introduced when two shots are artificially combined together by applying cinematic effects (fade-in, fade-out, dissolve, etc) [24]. Compared to the abrupt shot transitions, the gradual shot transitions are much more difficult to detect, because camera operations and big object movements may cause temporal variance similar to gradual shot transitions.
- Key frame selection
  Key frames provide a suitable abstraction and framework for video indexing, browsing and retrieval [1]. Users can get an overview of the video by only browsing the key frames. Furthermore, using key frames greatly reduces the amount of data required in constructing higher level video structure units (e.g., scenes, story units). Most of the key frame selection techniques are based on shot detection; in other words, key frames are mainly defined at the shot level. In order to select key frames, the video stream is first partitioned into shots, and different techniques are applied to extract key frames for each shot [25].



Video Content Modeling. Figure 1. A hierarchy structure of video data.

  - Scene construction from shots

    Individual shots are fundamental representation units of a video. However, in a normal video stream, there are thousands of shots, and usually a single shot conveys very little semantics. Shots can be grouped into higher-level segments to form *scenes*, which can convey much more semantics than shots [18]. All the scene construction algorithms follow similar steps: (1) Shots are clustered together based on the similarity which is computed from some visual features (e.g., color, texture, edge, etc.), (2) Clusters that are temporally close to each other are grouped into scenes.

- Annotation-based video data modeling

  Annotation-based video modeling techniques associate annotations (keywords or free text) to video sequences. These annotations describe the semantics of the video sequences. Based on the structure of annotations, an annotation-based video data model can be classified into: a *single value annotation structure* and an *attribute-value pair structure.*

  - Single value annotation structure approach

    In this model, annotations are associated to logical [20,23] or physical [11] frame sequences directly.

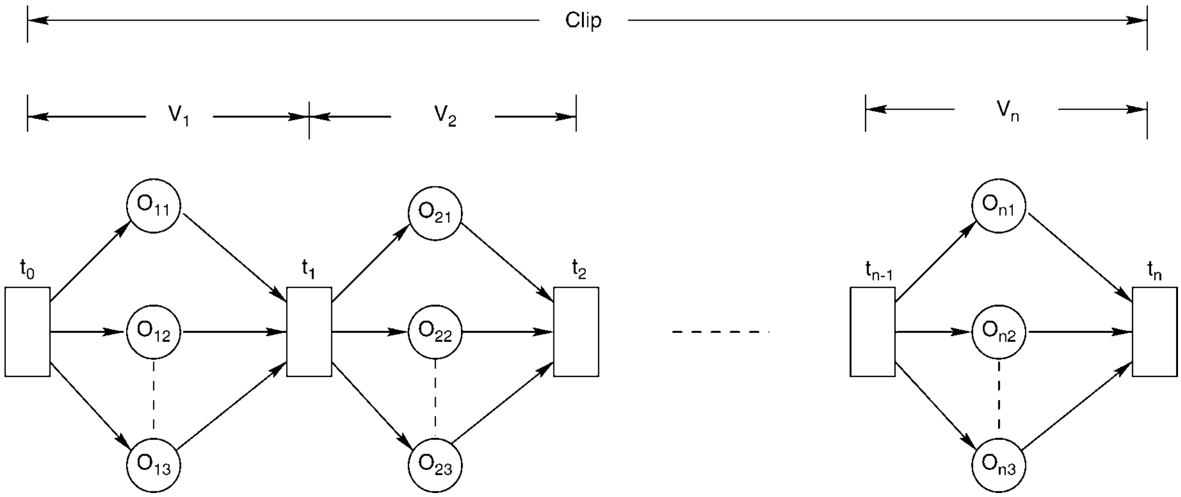
    - Annotations to logical frame sequences. Smith et al. [20] proposed a layered annotation representation model called the stratification model that segments the contextual information of the video. The stratification model divides the video sequence into a set of overlapping strata. A stratum consists of descriptive information such as title, keywords and the boundaries it represents. The strata may be contained in other strata and may encompass a multitude of other descriptions. The content information is derived by examining the union of all the contextual descriptions that are associated with it. Each stratum is logically independent and will make its contribution to the content when its descriptive information meets the user requirements.

    - Annotations to physical frame sequences

      Different from the stratification system which directly links the logical video segment to the raw video data at frame level, a generic video model [11] is developed which incorporates a segmentation-based video data model to associate text to a well defined structure. Hjelsvold's model supports sharing and reuse of the video data, temporal interval operations and structure abstraction. However, the arbitrary definition and reuse of a frame sequence introduces complicated relationships between structure units and thematic annotations.

- Attribute-value pair structure approaches
  Instead of using the single value for annotations, attribute-value pairs are adopted to annotate the logical video segments. Attributes define the types, ranges and semantics of the values [17].

- Salient object-based video modeling
  Events that happen in a video such as dialogs, explosions, car chases are considered basic semantic units that the video intends to express. Through analysis of characteristics of video content, the event can be expressed by describing spatio-temporal relationships among the *video objects* that appear in that video. The models of this category can be further divided into *motion segmentation-based* and *spatio-temporal relationship-based*.
  - Motion segmentation-based approaches
    Motion based approaches try to model video data through investigating motion directions, trajectories and velocities of video objects. These motion parameters are extracted together with video objects from video data and stored together. Motion segmentation-based approaches can befurther divided into *graph-based* approaches and *motion vector-based* approaches.
    - Graph-based approaches
      Approaches in this category use a graph to describe moving objects, their trajectories and spatio-temporal relations with other moving video objects [9,13,15].
    - Motion vector-based approaches
      Instead of using a graph, motion vector-based approaches model moving video objects with a set of motion vectors [6].
  - Spatio-temporal relationship-based approaches
    Motion segmentation-based approaches only address motion parameters of video objects. However, the spatial and temporal relationships among video objects (not only moving video objects [15,13]) possess many more semantics. Several approaches have been proposed and they can be classified into two categories: *spatio-temporal logic-based* and *graph model-based.*
    - Spatio-temporal logic-based approaches
      Del Bimbo et al. [4] propose a Spatial-Temporal Logic (STL) language to describe the spatio-temporal relationships among video objects within image sequences.
    - Graph model-based approaches
      Different graphs are used in these approaches to model the spatio-temporal relationships among the video objects. From the point of view of dealing with semantic heterogeneity of video data, Day et al. [10] propose a Video Semantic Directed Graph (VSDG), which is used to construct users' heterogeneous views of the video data. VSDG is created to model spatio-temporal interactions between video objects. An example of a VSDG is shown in Fig. 2. A video clip is composed of n video segments, labelled $V_1,V_2,...V_n$. Each video



Video Content Modeling. Figure 2. VSDG representation of a clip.

object $(O_{11},O_{12},...O_{nn})$ has attributes that describe the duration of its appearance in the video segments and its motion vector. Each rectangular node corresponds to an event in the video clip whenever a new physical object appears.

From the point of view of multimedia presentation, two other approaches have been proposed to model the spatio-temporal relationships among the video objects. HPNs [2] use Hierarchical Petri nets to capture the multi-level content of video data, which includes motion trajectories of moving objects and spatial temporal relationships among video objects. Chen et al. [7] model the salient objects, their content, and their spatio-temporal relationships in a hierarchy model and propose a video query language for users to search related contents.

Key Applications

Content Based Video Retrieval

Content-based video retrieval (CVIR) is the the problem of searching for videos that have similar content to the query video in a large database.

Video Databases

With a proper video content modeling technique, the video data can be stored in a database for efficient retrieval and analysis later on.

Cross-references

► Image Content Modeling

Recommended Reading

1. Aigrain P., Zhang H., and Petkovic D. Content-based representation and retrieval of visual media: A state-of-the-art review. Multimed. Tool. Appl., 3(3):179–202, 1996.
2. Al-Khatib W. and Ghafoor A. An Approach for Video Meta-Data Modeling and Query Processing. In Proc. 7th ACM Int. Conf. on Multimedia, 1999, pp. 215–224.
3. Bertini M., Bimbo A.D., and Torniai C. Automatic video annotation using ontologies extended with visual information. In Proc. 13th ACM Int. Conf. on Multimedia, 2005, pp. 395–398.
4. Bimbo A.D., Vicario E., and Zingoni D. Symbolic Description and Visual Querying of Image Sequences Using Spatio-Temporal Logic. IEEE Trans. Knowl. and Data Eng., 7(4):609–622, 1995.
5. Browne P. and Smeaton A.F. Video information retrieval using objects and ostensive relevance feedback. In Proc. 2004 ACM Symp. on Applied Computing, 2004, pp. 1084–1090.
6. Chang S.F., Chen W., Meng H.J., Urama H., and Zhong D. A fully automated content-based video search engine supporting spatiotemporal queries. IEEE Transaction on Circuits and Systems for Video Technology, 8(5):602–615, 1998.
7. Chen L., Oria V., and Özsu M.T. Modeling Video Data for Content Based Queries: Extending the DISIMA Image Data Model. In Proc. 9th Int. Conf. on Multimedia Modeling, 2003, pp. 169–189.
8. Cooper M. Video segmentation combining similarity analysis and classification. In Proc. 12th ACM Int. Conf. on Multimedia, 2004, pp. 252–255.
9. Courtney J.D. Automatic video indexing via ojbect motion analysis. Pattern Recognition, 30(4):607–625, 1999.
10. Day Y.F., S.D., Iino M., Khokhar A., and Ghafoor A. Object-oriented conceptual modeling of video data. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 401–408.
11. Hjelsvold R. and Midtstraum R. Modelling and Querying Video Data. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 686–694.
12. Lefèvre S., Holler J., and Vincent N. A review of real-time segmentation of uncompressed video sequences for content-based search and retrieval. Real-Time Imaging, 9(1):73–98, 2003.
13. Li J., Özsu M.T., and Szafron D. Modeling of moving objects in a video databas. In Proc. Int. Conf. on Multimedia Computing and Systems, 1997, pp. 336–343.
14. Martinez M. and Moran F. Authoring 744: Writing Descriptions to Create Content. IEEE MultiMedia, 10(4):94–99, 2003.
15. Nabil M., Ngu A.H.H., and Shepherd J. Modeling Moving Objects in Multimedia Database. In Proc. 8th Int. Conf. Database and Expert Syst. Appl., 1997, pp. 67–76.
16. Naphade M.R. and Huang T.S. Extracting semantics from audio-visual content: the final frontier in multimedia retrieva. IEEE Transactions on Neural Networks, 13(4):793–810, 2002.
17. Oomoto E. and Tanaka K. Ovid: Design and implementation of a video-object database system. IEEE Trans. Knowl. and Data Eng., 4(5):629–643, 1993.
18. Rui Y., Huang T.S., and Mehrotra S. Exploring Video Structure Beyond the Shots. In Proc. Int. Conf. on Multimedia Computing and Systems, 1992, pp. 237–240.
19. Shibata T., Kato N., and Kurohashi S. Automatic object model acquisition and object recognition by integrating linguistic and visual information. In Proc. 15th ACM Int. Conf. on Multimedia, 2007, pp. 383–392.
20. Smith T.G.A. and Davenport G. The stratification System: A Design Environment for Random Access Video. In Proc. Int. Workshop on Networking and Operating System Support for Digitial Audio and Video, 1992, pp. 250–261.
21. Smoliar S. and Zhang H. Content-based video indexing retrieval. IEEE Multimedia, 1(2):62–72, 1994.
22. Vendrig J. and Worring M. Interactive Adaptive Movie Annotation. IEEE MultiMedia, 10(3):30–37, 2003.
23. Weiss R., Duda A., and Gifford D.K. Composition and search with a video algebra. IEEE Multimedia, 1(2):12–25, 1994.
24. Zhang H., Kankanhalli A., and Smoliar S. Automatic partitioning of full-motion video. Multimedia Systems, 1:10–28, 1993.
25. Zhang H.J., Low C.Y., Smoliar S.W., and Wu J.H. Video parsing, retrieval and browsing: An integrated and content based solution. In Proc. 3rd ACM Int. Conf. on Multimedia, 1995, pp. 15–24.

Video Content Processing

▶ Video Content Analysis

Video Content Structure

Xian-Sheng Hua, Meng Wang
Microsoft Research Asia, Beijing, China

Synonyms

Video structuring; Video structure analysis

Definition

Mining video content structure is an elementary step of video content analysis. Direct access to a video without indexing is usually not an easy task, due to its length and unstructured format. On the other hand, analogous to text documents that can be decomposed into chapters, paragraphs, sentences and words, videos can be segmented into units like scenes, shots, and keyframes. The analysis of video content structure can be viewed as the process of hierarchically decomposing videos into units and building their relationships. Through such a process, a table-of-content can be constructed for each video, which facilitates the access and manipulations of the video data. For example, the keyframes extracted from the video can be used as its entries for indexing and browsing.

Historical Background

More and more *video* data has become available to ordinary users due to the advances in storage devices, networks and compression techniques. However, the efficient access to an unstructured video is a challenging task due to the huge number of frames. Video structuring is proposed to tackle this difficulty. In [14], Zhang et al. first proposed the scheme that partitions videos into shots, and then selects the first frame of each shot as the keyframe for indexing. From 2001, the National Institute of Standard and Technology (NIST) has established video shot detection as a evaluation task in the TREC Video Retrieval Evaluation (TRECVID) benchmark (http://www.itl.nist.gov/iaui/894.02/projects/trecvid/). In [10], Rui et al. proposed to group shots into scenes, such that the video content can be structured at a higher level. For several video genres, such as news and movies, "story" is also widely applied as the unit for content organization. Kim et al. [6] proposed a method to further segment shots into subshots in order to facilitate more detailed implementations. For each shot or subshot, one or more keyframes can be extracted to represent its content. Thus, as illustrated in Fig. 1, generally a video can be structured in a hierarchical form as "video → scenes → shots → subshots → keyframes," and it can also be segmented into stories if it belongs to certain genres, such as news video. The definitions of these terminologies are as follows:

Shot: a shot is an uninterrupted clip recorded by a single camera. It is a physical entity and often forms the building block of video content.

Scene: a scene is defined as a collection of semantically related and temporally adjacent shots, depicting and conveying a high-level concept or story. A scene usually consists of a series of consecutive shots that are recorded in the same location.

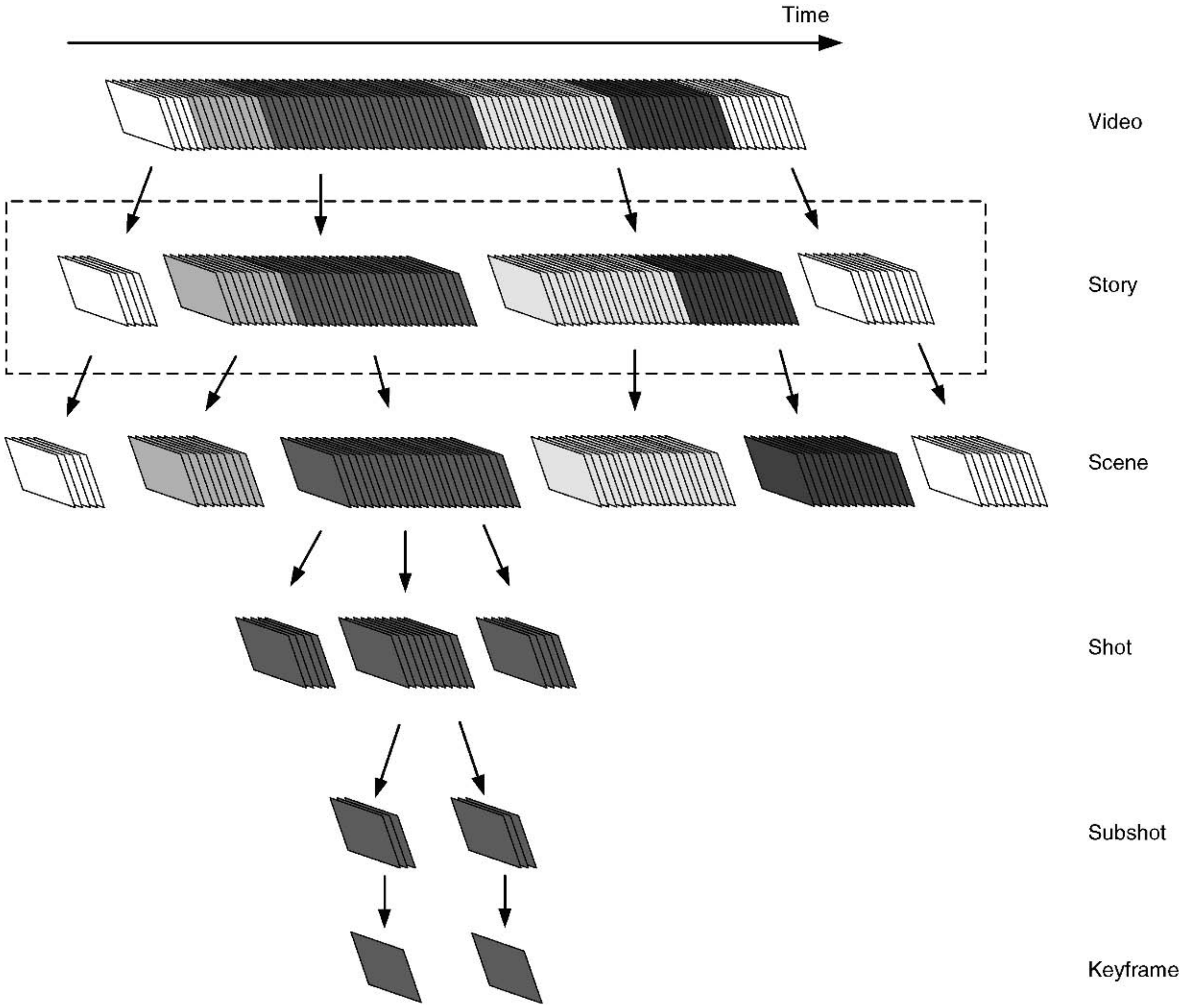
Story: a story is referred to as a clip that captures a continuous action or a series of events, and it may be composed of several scenes and shots. Note that the story lines are usually only clear for rigidly structured videos. Currently, most story identification methods are developed for news videos. Thus, here only "news story" is considered. A definition of a news story in TRECVID is "a segment of a news broadcast with a coherent news focus which contains at least two independent, declarative clauses."

Subshot: subshot is a segment within a shot that corresponds to a unique camera motion. A shot can be divided into one or more consecutive subshots according to the movement of the camera.

Keyframe: a keyframe is the frame which best represents the content of a shot or a subshot. According to the content complexity, one or more keyframes can be extracted for each shot or subshot. Keyframes can be used as the entries of the video data for manipulations, such as indexing and browsing.

Foundations

As previously introduced, the analysis of video content structure may involve five techniques, namely, shot detection, scene grouping, story identification, subshot segmentation, and keyframe extraction.



Video Content Structure. Figure 1. Hierarchical decomposition and representation of video content.

Shot Detection

Shot detection is the process of identifying the boundaries between two consecutive shots, such that the frame sequence can be grouped into a set of shots. According to the transition style of the consecutive shots, the shot boundaries can be mainly categorized into two types, i.e., cut and gradual transitions. Cut indicates that the change between the two shots is abrupt, whereas the gradual transition means that there is a gradual special effect in the transition of the two shots. Many different shot detection methods have been proposed (including the detection and categorization of gradual transitions), and a most straightforward approach to shot detection is to measure the change between every two consecutive frames and a shot boundary can be declared if there is a significant change. Yuan et al. have provided a comprehensive survey on shot detection in [13], and more details about this technique can be found in the entry *Video Shot Detection*.

Scene Grouping

Scene grouping is usually implemented based upon the results of shot detection. From a global point of view, scene grouping can be viewed as a shot *clustering* task. With an appropriate pairwise similar definition, many different clustering algorithms can be applied to accomplish this task. Intuitively, two criteria should be considered in a scene grouping algorithm, namely, content similarity and temporal continuity. Content similarity means that the shots within the same scene should have similar content, whereas the temporal continuity indicates that these shots should be close to each other temporally. The content similarity is usually defined based on a low-level feature space, and many different features can be applied, including visual, audio and text features (the text features can be extracted by several existing techniques such as “Automatic Speech Recognition” and “Optical Character Recognition”).

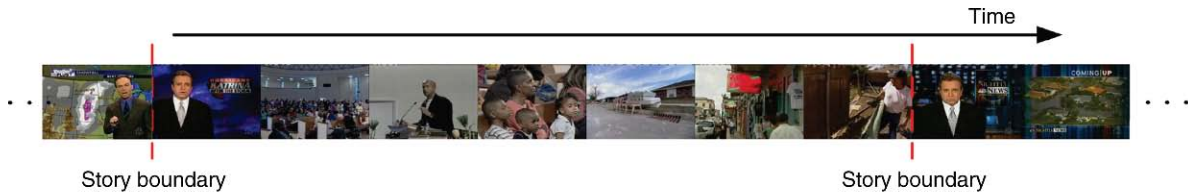
Gu et al. [3] have categorized the existing scene grouping methods into three approaches: merging-based, splitting-based, and model-based. The merging-based approach groups shots in a bottom-up way. In [10], the scene grouping is implemented via a two-step process: (i) assign the shots into groups according to their visual similarities and temporal continuities; and (ii) merge similar groups into a unified scene. Rasheed et al. proposed another two-step scene grouping process in [9]: (i) cluster shots according to Backward Shot Coherence (BSC); and (ii) merge the clusters into scenes based on an analysis of the shot length and the motion content in the potential scenes.

As opposed to the merging-based approach, splitting-based scene grouping methods are implemented in a top-down style, i.e., a video is split into a set of coherent scenes in turn. In [8,12], two different graph-based splitting methods are proposed. In these two methods, a video is represented by a graph, in which the vertices are denoted by the shots and the edges are determined by the similarities of the shots and their temporal localities. A graph is then partitioned into several subgraphs and each subgraph can be regarded as a scene. In [12], the graph is named as a Scene Transition Graph (STG), and it is partitioned into several subgraphs with the complete-link method, whereas in [8] the graph is named as a Shot Similarity Graph (SSG) and it is partitioned by the normalized cuts method.

Different from the above two approaches, model-based methods group shots into scenes with statistical models. Tan et al. [11] implemented scene grouping using the Gaussian Mixture Model (GMM), with each Gaussian component indicating the distribution of a scene, and the number of scenes can be determined by the Bayesian Information Criterion (BIC) method. Recently, Gu et al. [3] proposed an energy minimization scheme, in which the constraints of time and content of scene grouping are indicated by energy items. It is able to take both the global and local constraints into consideration, and the number of scenes can be established by the Minimum Description Length (MDL) principle.

Story Identification

Story identification needs more semantic understanding of video content (Fig. 2), and it is usually only applied for certain rigidly structured video genres such as news video. In TRECVID 2003 and 2004, news story identification was established as an evaluation task. The existing story identification methods can be mainly classified into two categories, i.e., rule-based and learning-based. The rule-based story identification methods are usually based on certain domain knowledge. For example, it has been observed that many news stories begin with an anchorperson shot and end with the start of another anchorperson shot. In [15], the pattern of news stories has been analyzed, and the task of story identification is accomplished via detecting the shots of certain types, such as anchorperson shot. However, such an approach highly depends on the adopted knowledge, and can hardly handle diverse video sources with different features and production rules. The learning-based approach is able to tackle this difficulty. In typical learning-based story identification methods, a set of story boundary candidates is first established (such as the shot boundaries and audio pauses), and then each candidate is classified as “story boundary” or not according to the model learned from a training set [1]. More details about these story identification methods can be found in [1,15] and references therein.



Video Content Structure. Figure 2. A typical news story in a video. The keyframes of the video have been illustrated, and the story boundaries can be identified according to the anchorperson analysis.

Subshot Segmentation

Subshot is a sub segment within a shot. Generally, a subshot is defined to contain a unique camera motion. Thus, subshot segmentation can be accomplished through camera motion detection. For example, consider a shot in which the camera moves as follows: zoomed out, then panned from left to right and zoomed in to a specific object, and then stopped. This shot then comprises three subshots, including one zoom out, one pan to right, and one zoom in. The camera motion between two adjacent frames can be estimated based on a two-dimensional affine model, in which the motion vector (v_x, v_y) at pixel (x, y) can be expressed as

$$\begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} a_1 \\ a_4 \end{pmatrix} + \begin{pmatrix} a_2 & a_3 \\ a_5 & a_6 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{1}$$

where $a_i(i = 1,2,...,6)$ denote the motion parameters. The motion parameters can be represented by a more meaningful set of terms as follows

$$\begin{cases} pan = a_1 \\ tilt = a_4 \\ zoom = \frac{a_2+a_6}{2} \\ rot = \frac{a_5-a_3}{2} \\ hyp = \frac{|a_2-a_6|+|a_3+a_5|}{2} \end{cases} \tag{2}$$

where *pan* corresponds to the pan movement of camera, *tilt* corresponds to tilt and boom, *zoom* corresponds to dolly and the change of focus, *rot* corresponds to roll, and *hyp* indicates that object motion is predominant. For more details about video motion analysis, please refer to [6].

Based on the analysis of camera motion, the subshot segmentation can be implemented and each subshot will be categorized into one of the following six classes: *pan*, *tilt*, *zoom*, *rot*, *objectmotion*, and *static*. For an individual frame, its motion category can be determined by thresholding the related terms in Eq. (2). However, it is observed that generally a camera movement will be maintained for a period of, say, at least a half second [6]. Thus, a typical subshot segmentation process consists of three steps, i.e., frame-level motion detection, segment-level motion detection, and post-processing. More details can be found in [6]. Figure 3 illustrates an example, in which the shot can be segmented into seven subshots.

Keyframe Extraction

Keyframes are the frames in the video sequence that can best preserve the content of shots or subshots. They can be used as the entries of the videos for access. The most widely-applied methods for this task can be categorized into two approaches, i.e., analysis-based and clustering-based. The analysis-based methods extract keyframes by analyzing video content, such as the quality and the attractiveness of frames. For example, in [7], Ma et al. adopted an attention model, and the frames that attract the most user attention are extracted as keyframes. In the clustering-based approach, a clustering process is carried out, and then the cluster centroids can be established as keyframes. In [4], Hanjalic et al. adopted a partitioning-based clustering method, and the number of clusters (i.e., the number of keyframes) can be determined by a cluster-validity approach.

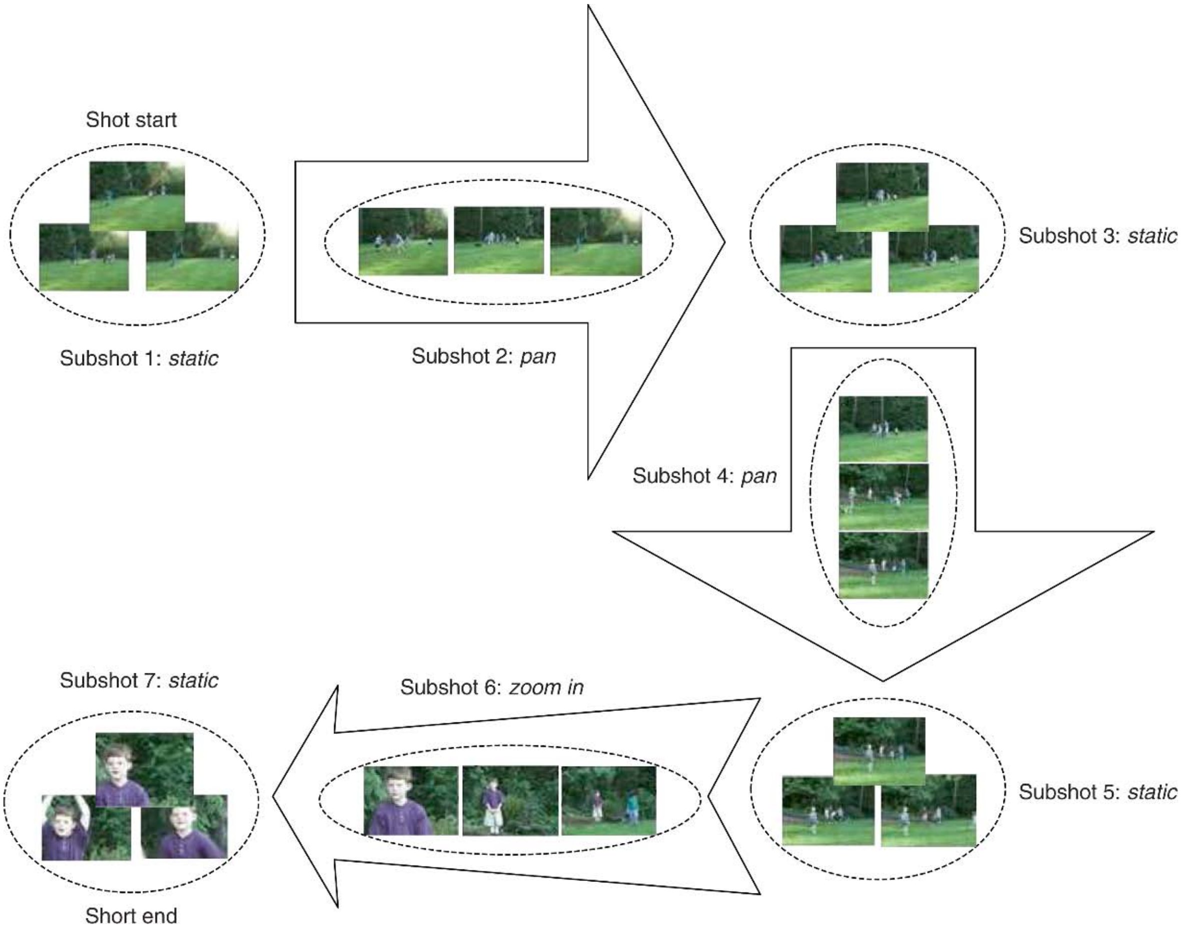
A recent development in keyframe extraction is to formulate it as a learning task [5]. It is observed that the representativeness of a video frame involves several elements, such as its image quality, user attention, and visual details. Thus, the frame representativeness can be modeled through a training set that regards these elements as features. This learning process is able to simulate a human's perception on keyframe extraction. The method in [5] builds a model based on four elements extracted from each frame, including *frame quality*, *visual details*, *content dominance* and *attention measurement*, and encouraging results have been reported in both subjective and objective evaluations.

Key Applications

As previously mentioned, analyzing video content structure is the first step of *video content analysis*, and it is thus a prerequisite for many video applications, such as *video abstraction*, *video summarization* and *content-based video retrieval.*

Future Directions

The existing video structuring methods are mainly carried out based on low-level features. However, from the above introduction it can be found that several structuring techniques involve the understanding of video content. For example, for scene grouping, it is better to group shots with coherent semantic concepts rather than those with close low-level features. Thus, it is believed that better structuring performance can be achieved by leveraging semantic features in the existing algorithms. This can be named as "semantic-based video structuring"



Video Content Structure. Figure 3. An illustrative example of subshot segmentation.

approach. For details about the application of semantic features (or semantic feature space), please refer to [2]. It is worth noting that structuring is currently always regarded as an independent and preliminary step of video semantic analysis. To use semantic features for structuring, the semantic analysis and structuring may have to be integrated or unified.

Experimental Results

Generally, for each presented method, there is an accompanying empirical evaluation in the corresponding reference. For shot detection and story identification, the results achieved in TRECVID can be found in the corresponding notebook papers which can be downloaded from the TRECVID website (http://www-nlpir.nist.gov/projects/tvpubs/tv.pubs.org.html).

Cross-references

- ▶ Clustering
- ▶ Content-Based Video Retrieval
- ▶ Video
- ▶ Video Abstraction
- ▶ Video Content Analysis
- ▶ Video Shot Detection
- ▶ Video Summarization

Recommended Reading

1. Chua T.-S., Chang S.-F., Chaisorn L., and Hsu W. Story boundary detection in large broadcast news video archives - techniques, experiences and trends. In Proc. 12th ACM Int. Conf. on Multimedia, 2004.
2. Ebadollahi S., Xie L., Chang S.-F., and Smith J.R. Visual event detection using multi-dimensional concept dynamics. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2006.

3. Gu Z., Mei T., Hua X.-S., Wu X., and Li S. EMS: Energy minimization based video scene segmentation. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2007.
4. Hanjalic A. and Zhang H.-J. An integrated scheme for automated video abstraction based on unsupervised cluster-validaty analysis. IEEE Trans. Circ. Syst. Video Tech., 9(8):1280–1289, 1999.
5. Kang H.-W. and Hua X.-S. To learn representativeness of video frames. In Proc. 13th ACM Int. Conf. on Multimedia, 2005.
6. Kim J.-G., Chang H.S., Kim J., and Kim H.M. Efficient camera motion characterization for MPEG video indexing. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2000.
7. Ma Y.F., Lu L., Zhang H.-J., and Li M. A user attention model for video summarization. In Proc. 10th ACM Int. Conf. on Multimedia, 2002.
8. Rasheed Z. and Shah M. Detection and representation of scenes in videos. IEEE Trans. Multimed., 7(6):1097–1105, 2005.
9. Rasheed Z. and Shah M. Scene detection in holleywood movies and tv shows. In Proc. Int. Conf. on Computer Vision and Pattern Recognition, 2005.
10. Rui Y., Huang T.S., and Mehrotra S. Constructing table-of-content for video. Multimed. Syst., 7:359–368, 1999.
11. Tang Y.-P. and Lu H. Model-based clustering and analysis of video scenes. In Proc. Int. Conf. Image Processing, 2002.
12. Yeung M., Yeo B., and Liu B. Segmentation of videos by clustering and graph analysis. Comput. Vis. Image Understand., 71(1):94–109, 1998.
13. Yuan J., Wang H., Xiao L., Zheng W., Li J., Lin F., and Zhang B. A formal study of shot boundary detection. IEEE Trans. Circ. Syst. Video Tech., 17:168–186, 2007.
14. Zhang H.-J., Kankanhalli A., and Smoliar S.W. Automatic paritioning of full-motion video. Multimed. Syst., 1: 10–28, 1993.
15. Zhang H.-J., Tan S.Y., and Smoliar S.W. Automatic parsing and indexing of news video. Multimed. Syst., 2(6):256–265, 1995.

Video Data Modeling

► Video Content Modeling

Video Format

► Video Representation

Video Indexing

► Video Sequence Indexing
► Visual Content Analysis

Video Metadata

Frank Nack
University of Amsterdam, Amsterdam,
The Netherlands

Synonyms

Video annotation; Video content description

Definition

Digital video is recorded in two different image capture formats: interlaced and progressive scan. Interlaced video establishes an image by recording alternating sets of lines, where one set of odd or even lines is referred to as a "field," and a consecutive pairing of two fields of opposite parity is called a frame. In a progressive digital video each frame is recorded in a distinct manner, with both fields being identical. Both operate at the same number of frames per second, which is in NTSC roughly 29 images and in Pal around 25 images per second. As of 2007, the highest resolution demonstrated for digital video generation is 33 megapixels (7,680 × 4,320) at 60 frames per second ("UHDV").

Metadata is data about data of any sort in any media, describing an individual datum, content item, or a collection of data including multiple content items. In that way, metadata facilitates the understanding, characterization, use and management of data.

Video metadata is structured, encoded data that describes content and representation characteristics of information-bearing video entities to facilitate the automatic or semiautomatic identification, discovery, assessment, and management of the described entities, as well as their generation, manipulation, and distribution.

Historical Background

In the late 1970s to the early 1980s of the twentieth century the first digital video production equipment, such as time base correctors (TBC) and digital video effects (DVE) units, were developed. This type of equipment was, however, not part of the general production flow, as the digitized and processed video still needed to be converted back to standard analog video.

Digital video was first introduced commercially in 1986 with the Sony D-1 format, which recorded an uncompressed standard definition component video signal in digital form. Due to its costs, the D-1 was primarily used by large television networks. Over the years it was replaced by cheaper systems, such as Sony's Digital Betacam, which were still expensive so that they were mainly used as a field recording format by professional television producers.

Consumer digital video first appeared in 1991 in the form of QuickTime, Apple's framework for time-based and streaming data. Shortly after, Microsoft's AVI format followed and then MPEG-1, MPEG-2 and MPEG-4 (the basis of MPEG-4 was QuickTime) formats gained popularity. The introduction of the DV tape in 1996 gave digital video another acceptance push as it allowed recording directly to digital data and thus simplified the editing process, allowing non-linear editing systems to be deployed wholly on desktop computers (e.g., FAST 601, Softimage DS, Appel's Final Cut line).

The work on video codec technology from the early 1990s allowed a great deal of research directed on computer environments that seek to interpret, manipulate or generate digital video either in a manual, semi-automatic, or automatic way [1–6]. In all these research projects the description of video content in form of metadata played a central role. The steady infiltration of those technological advances in everyday production finally allowed the general public to really include video into their everyday communication. Examples of such infiltrated technologies are simple non-linear video editing systems, environments for new media authoring (e.g., Director/Shockwave, Flash), and web authoring environments (e.g., Dreamweaver, Frontpage, and SMIL).

The rapid growth of professionally created and exploited video databases, as well as the slow but steady growth of user-generated video material on the web (the growth was much slower than the one of images), forced research to address similar problems as those already explored for digital image databases, namely how to efficiently search and then exploit video databases. Over the years a number of metadata standards have been developed, which address various aspects of video data, such as the Dublin Core Metadata Initiative (http://www.dublincore.org/), the Society for Motion Pictures and Television Engineering (SMPTE) (http://www.smpte.org/home/), the Moving Picture Expert Group (MPEG) (http://www.chiariglione.org/mpeg/), the TV-Anytime Consortium (http://www.tv-anytime.org/), and the International Press Telecommunications Council(IPTC) (http://www.iptc.org/pages/index.php). The common definition language between all these languages has been in one way or another the Extensible Markup Language (XML) [7], defined by W3C.

The most relevant initiative for addressing segmentation, indexing and content-based retrieval of digital video based on low-level features, usually automatically extracted from the content, is the TrecVid series (http://trecvid.nist.gov).

The by far largest contribution to digital video was provided by the three ISO standards MPEG-4 [8], MPEG-7 [9] and MPEG-21 [10]. With MPEG-4 the group entered the realm of media content, arisen due to the growing need for content manipulation and interaction. MPEG-4 expanded MPEG-1 to support video "objects," 3D content, low bitrate encoding and support for Digital Rights Management. Yet, MPEG-4 lacked the means to identify video objects on a semantic level – a necessary element for efficient search in and the manipulation of video. Addressing this problem, the MPEG-7 standard was started in the late 1990s. One core part of MPEG-7 (Part 3: Video – and also larger sections of Part 5: The Multimedia Description Schemes) is devoted to video annotation only. The beginning of the twenty-first century established an even faster exchange of multimedia data via the web, as higher bandwidth as well as access to high quality data became a commodity. Media businesses, such as the film industry, feared, due to peer-to-peer technology for their markets and requested strict digital rights management enforcement. MPEG reacted with MPEG-21, defined as the multimedia framework.

Since 2005 the web saw an exploitation of video content. The main reasons were:

- Easy access to digital video cameras
- The provision of Adobes Flash player, which can be easily integrated into web pages
- Easy ways of sharing enabling technology, of which YouTube (http://www.YouTube.com), among others, is the most well-known example

The relevance of YouTube is not so much grounded in its technology advances (YouTube plays back videos limited in size (320 × 240 pixel) and quality (a bitrate of around 314 kbit/s with a frame rate depending on the uploaded video), but rather on the distribution of content. YouTube lets their clientèle act as they please, without enforcing editorial decisions or odds and ends such as copyright. As many Web2.0 applications YouTube applies folksonomy tagging (also known as collaborative tagging, social classification, social indexing, and social tagging) a method of collaboratively creating and managing tags to annotate and categorize content. In folksonomy tagging metadata is not only generated by experts but mainly by creators and consumers of the content, where a tag is a keyword or term associated with or assigned to a piece of information (a picture, a map, etc.), which enables keyword-based classification and search. The advantage of tagging is its ease of use – creating a vocabulary based on freely chosen keywords instead of a controlled set of terms and structures. This approach, though highly popular, carries serious problems. Typically there is no information about the semantics of a tag, no matter if it is a single tag or a bag of tags. Additionally, different people may use drastically different terms to describe the same concept. This lack of semantic distinction can lead to inappropriate connections.

The success of YouTube, which consumed in 2007 as much bandwidth as the entire Internet in 2000, and similar sites let the W3C start a new video on the web initiative in 2008, which will address the access of video on the web (http://www.w3.org/2008/WebVideo/Activity.html).

Foundations

The description of video content is to some extent very similar to the one of images, simply because a video is the representation of single images over time. The following text, therefore, only addresses those issues of video content representation that are not already addressed in the "Image Metadata" entry of this encyclopedia. As applications like YouTube reestablished the question of rights management on a large scale, this section also addresses this problem in some detail.

The two essential aspects of video that enhance its "meaning making" over that of an image is the concept of a shot and the sequence of shots, or time and structure of time.

Shot Description

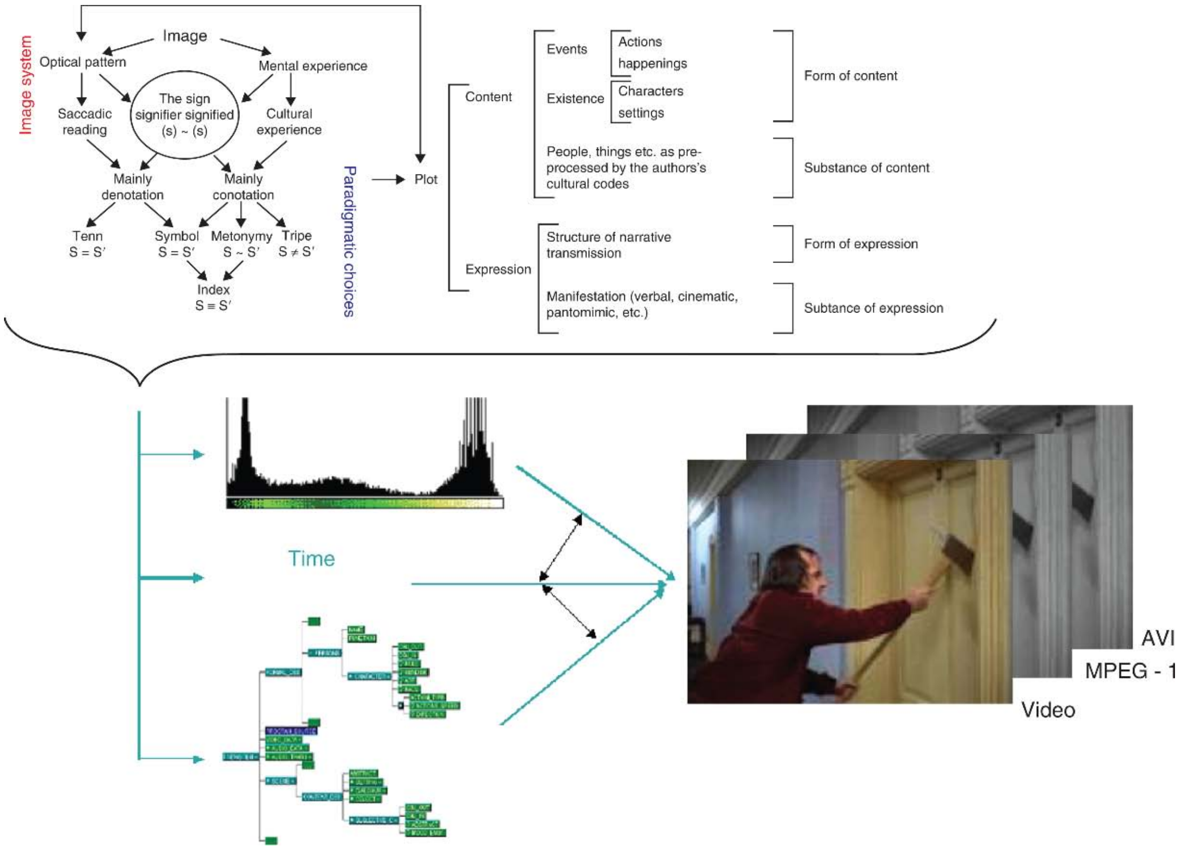
A shot is a single piece of film, however long or short, without cuts, exposed continuously. The significant additional element here is time, which provides the basis for the understanding of action, distance and the relationship among characters, based on the relationship between frames within a shot and their rhythmical variations. The compositional use of *focus*, for example, through which the foreground, middle ground or background are emphasized, is one way to guide the perception of a shot. If all planes are represented in focus, they are attributed with the same level of importance, whereas emphasis can be achieved by use of focus for a part of a frame. Of even stronger impact than focus is *camera movement* around the imaginary vertical axis (pan), the horizontal axis (tilt), and the longitudinal axis (distance from lens to the subject). The *tilt*, for example, presents the eye-level from which a scene is perceived and thus can affect the importance ascribed to an object (for example, high-angle shots may diminish the perceived importance of an object). The *tempo* of a shot can also provide information. The intense feeling of fast movement may excite, while calm movement expressed, for example, through the slow rolling of waves filmed from a static camera position, may encourage feelings of relaxation. Related to tempo, is the *perceived duration* of the shot. The actual duration of a long shot full of people and action may well be identical to one of the close-up of a face, and yet the latter will be perceived as being longer. Hence, the organization of perceived duration is more complex than the actual duration of a shot.

There are, on the perceptional level, various automatic methods to extract shot boundaries, namely the border between two different shots, forms of transitions between shots, as well as temporal or conceptual concepts (color, shape or texture-based). A good summary of these methods can be found in [11]. The advantage of automatic extraction is its low costs. The disadvantage, similar to the problems in automatic image indexing, is that it is exclusively organized around the sensory surface structures of media, i.e., the physical features, which are able to grant access to the representation of conceptual items but not to the higher semantics users wishes to access. Detailed description of schemata, that allow high-level semantic descriptions, namely for the essential content categories actor, appearance (features of a character), action (chronology of, or direction of, the action) location, and cinematographic devices (e.g., camera and lens movement, camera position, etc) are described in [3], which also establishes an icon based annotation method, and [4]. Both approaches make use of hierarchical schemata structures.

Important for all of these annotation forms, might they be automatic, semi-automatic or manual, is a proper representation of time. The current ways or representing time are either based on:

- Physical time (full clock value (e.g., 7:45:23.76, where the last two items present ms), partial clock values (any sort of short base notation), time count values (numbers with a additional type string, e.g., 10S for "ten seconds"), and time context values, which are represented in three parts: a date field (YYY:MM:DD), a time field, and a timezone field). These descriptions allow the location of an annotation based on the start- and end-frame.
- Logic descriptions, as best described by [12], which are useful for a symbolic representation of a time relation instantiation in a triple structure, as requested by the Resource Description Framework (RDF) [13].
- Or as a hierarchical ontological description, that describes time not on its physical form but rather on its relevance for the shot content, e.g., The presented epoch (e.g., seventeenth century), season (e.g., Summer), or daytime (e.g., Midday) [3].

The essential concepts of video metadata are summarized in Fig. 1. The upper left corner describes the possibilities of interpretation of an image, the upper right part outlines the structural elements of a story (of which the visual information forms a subpart). Both together form the schematic basis for the annotation process, which can be performed in an automatic, semi-automatic or manual fashion, as described in the lower left part of Fig. 1, which are performed on various instantiations of the same content material in form of differently coded digital video, as outlined in the lower right part.



Video Metadata. Figure 1. Essential metadata concepts for the description of video content.

Sequence

The final level of generating meaning with video material to be considered is the way in which content of a shot can be affected by other shots.

- The meaning of a shot depends on the context in which it is situated.
- A change in the order of shots within a scene changes the meaning of the shot as well as the meaning of the scene.

Important is that not every combination of shots creates a meaning, but there are restricted conventions that can help create larger meaningful entities. The key elements for creating meaning by joining shots are *assertions* and *associative cues* [14].

An assertion is the relationship between two elements. There are many different types of such relationships. For example, the description of an attribute (such as *red* for a car) could be as important as a simple action (two men shaking hands).

Associative cues result from the combinations of the indicators that make the creation of meaning possible. One essential cue was already discussed, namely human action. The other one is surrounding space. Most human activities, human roles or objects are associated with specific locations. The conceptualization of space is, therefore, an elementary principle of the analysis and organization of video material. The sequential structures built up through juxtaposition provide the complex and intricate syntax for film narrative. The two essential works for structural metadata concepts for video are described by Aguierre Smith et al. [1] and in the MPEG-7 part 5 [15].

Aguierre-Smith designed the Stratification System to support an anthropological video study in the state of Chiapas, Mexico. The idea was to provide a number of researchers with random access to a video archive, in which video could be annotated with complementary or even contradictory descriptions. The video material was stored on a laserdisc. The annotations used in the Stratification System were keywords organized in hierarchical classes which were implemented as directory trees in UNIX. The novel feature introduced by Aguierre-Smith was the multiple partially overlapping annotation, where each annotation is related to a precise time index (begin and end frame). To provide a visual representation of the distinct layers of the representation, the Stratification System used a histogram, where the keyword classes are displayed as buttons along the *y*-axis and the time code (frame numbers on the laserdisc) form the *x*-axis. Aguierre-Smith's stream-based content representation for video enables the dynamic development of context while maintaining the completeness of the original footage. The notion of multiple partially overlapping annotations establishes the Stratification System as a breakthrough in the effective representation of video content, despite its weaknesses, i.e., the keyword approach and the lack of a true representation of the semantics of the video.

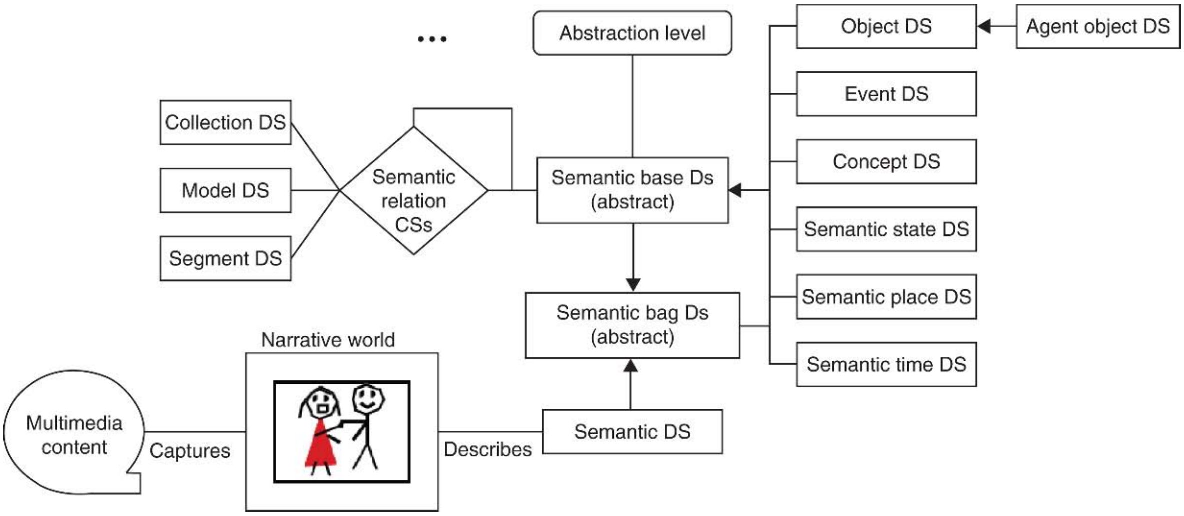
In MPEG-7's Part 5: Multimedia Description Schemes (MDS) the organization structure follows the overall hierarchical document structure, which at the end is instantiated as an XML schema file. Figure 2 visualizes description schemata that can be used to describe real-life concepts or narratives, which include objects, agent objects, events, concepts, states, places, times, and narrative worlds, all depicted by or related to video content. This means that the description schemata can be used to annotate the narrative in the video or the narrative world of the video, which then can be linked to the video.

The most relevant description schemata in Fig. 2 are:

- Semantic DS, which describes narrative worlds that are depicted by or related to the video content.
- Object DS, which describes objects in a narrative.
- AgentObject DS, which describes objects that are persons, organizations, or groups of people in a narrative.
- Event DS, which describes events in a narrative.
- Concept DS, which describes abstract semantic entities that cannot be described as abstractions of any objects, events, times, places, or states.
- SemanticState DS, which describes states or parametric attributes of semantic entities and semantic relations, at a given time or location in a narrative.
- SemanticPlace DS, which describes locations in a narrative.
- SemanticTime DS, which describes times in a narrative.

As often in MPEG-7, the description options are flexible, which allows the user to establish complex annotation structures. Yet, the taken approach also carries a number of problems, essentially:

- There are a great number of abstract elements, which are used to establish class structure. However,



Video Metadata. Figure 2. Illustration of the relationships of the tools for describing the semantics of video content.

abstract elements cannot appear in instantiations. When an element is declared to be abstract, a member of that element's substitutable class must appear in the instance document. To indicate that the derived type is not abstract, the XML namespace mechanism is used (xsi:type). Thus, a thorough understanding of schemata development is required, which makes instant schemata development for distinct domains hard, especially if the required schemata should cover simple descriptions, where the theoretical overhead is actually not required.

- The interlocked nature of schemata, providing an ontology-like yet general set of schemata for describing media semantics, makes it very difficult for a user to identify the appropriate schemata and to use them in isolation.
- Due to the lack of a fundamental data model the structures provided show inconsistencies and duplications, which makes manual schemata generation difficult.

There are, however, projects in real world domains, such as the TV Anytime Forum, that give an indication of how media-aware semantic structures, such as those provided by MPEG-7, will be used in the future. The TV Anytime Consortium (http://www.tvanytime.org) developed specifications for services based on consumer digital storage devices. The semantic structures, all written in XML Schema, are proprietary and cover the essential aspects of media description, i.e., content description, content referencing and location, rights management and protection, systems and transport. Though the TV Anytime schemata are similar to the equivalent structures in MPEG-7, they are less complex in their organizational structure. TV Anytime includes, for example, the MPEG-7 schemata on user-modeling, though without incorporating the complete MPEG-7 organizational overhead. Rather, TV Anytime uses MPEG-7 as a namespace and is thus able to incorporate only the required schemata.

Rights Management

MPEG-21 [10] addresses, among others, two important aspects when it comes to user interaction with video content (in fact MPEG-21 describes that for all types of multimedia): first the identification and declaration of video items, and second the description and protection of rights.

MPEG-21 is understood as a framework that supports users to access, exchange, trade and consume digital media. The basic unit for MPEG-21 is the Digital Item (DI) which can be distributed and on which transactions can be performed. The DI is defined as a "virtual container" for metadata (created based on MPEG-7) and content (in MPEG-1, MPEG-2 or MPEG-4 format). The Digital Item Declaration (DID) defines the resources (e.g., MPEG-4 files) and the related metadata (e.g., Dublin Core) which the author identified to be part of the DI. The Digital Identification framework then supports devices to uniquely identify various objects and items of the DI.

The provision of DIs only makes sense if there are mechanisms that allow to exploit those structures. For that MPEG-21 developed the Rights Expression

Language (REL), which is an XML-based machine-readable language that allows the specification of specific rights and conditions associated with the distribution of say video content. REL does not aim for replacing legal rights methods but rather specifies a grant specifying that a "principal" has a "right" over a "resource" under certain "conditions." REL is based on the Extensible Rights Markup Language (XCML) 2.0 which was selected by MPEG due to its expressiveness and unambiguaty. In addition MPEG-21 has designed the Rights Data Dictionary (RDD), which contains the key terms required to describe rights. Finally, the Intellectual Property Management and Protection Component (PMP) was established, which uses again metadata to allow the inclusion of protected and governed content into a DI. This mechanism facilitates terminals to process protected content. As MPEG-21 is at time of writing still relatively new the future has to show if that framework is functioning and thus will be accepted by the public.

Key Applications

The annotation of video is useful for the retrieval, reuse, manipulation, generation and distribution of video for domains, such as medicine, entertainment, distributed games, all sort of experience related applications and education.

Cross-references

- ▶ Image Metadata
- ▶ Multimedia Metadata

Recommended Reading

1. Adams B. Mapping the Semantic Landscape of Film: Computational Extraction of Indices through Film Grammar. Ph.D. thesis, Curtin University of Technology, Perth, 2003.
2. Aguierre Smith T.G. and Davenport G. The stratification system. a design environment for random access video. In Proc. ACM Workshop on Networking and Operating System Support for Digital Audio and Video, 1992.
3. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832–843, 1983.
4. Barry B. Mindfull documentary. Massachusetts Institute for Technology. Ph.D. thesis, MIT, Boston, 2005.
5. Davis M. Media Streams: Representing Video for Retrieval and Repurposing. Ph.D. thesis, MIT, Boston, 1995.
6. Extensible Markup Language (XML), http://www.w3c.org/XML/.
7. Gregory J.R. Some Psychological Aspects of Motion Picture Montage. Ph.D. University of Illinois, USA, 1961.
8. ISO MPEG-7 MDS Text of ISO/IEC 15938–5/FCD Information Technology – Multimedia Content Description Interface – Part 5: Multimedia Description Schemes, ISO/IEC JTC 1/SC 29/WG 11 N4242, 23/10/2001, 2001.
9. MPEG-4: ISO/IEC JTC1/SC29/WG11 N4668 March 2002, http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm.
10. MPEG-21: ISO/IEC JTC1/SC29/WG11/N5231 Shanghai, October 2002, http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm.
11. MPEG-7: ISO/IEC JTC1/SC29/WG11N6828 Palma de Mallorca, October 2004, http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.html.
12. Nack F. AUTEUR: The Application of Video Semantics and Theme Representation in Automated Video Editing. Ph.D. thesis, Lancaster University, 1996.
13. Nack F. and Putz W. Designing annotation before it's needed. In Proc. 9th ACM Int. Conf. on Multimedia, 2001, pp. 251–260.
14. Resource description Framework (RDF), http://www.w3c.org/RDF/.
15. Tonomura Y., Akutsu A., Taniguchi Y., and Suzuki G. Structured video computing. IEEE MultiMedia, 1(3):34–43, 1994.

Video Partitioning

▶ Video Segmentation

Video Querying

Ichiro Ide
Nagoya University, Nagoya, Japan

Synonyms

Image Querying; Query by Example; Similarity in Video; Near-duplicate Video Retrieval

Definition

Video querying is a way to issue a query for retrieving video data from a database without explicitly expressing the contents of the video-in-search by high-level semantics, such as keywords. It, instead, expects that a user issues a query as a video segment in hand. The system will then search and retrieve similar video data from the database. Since video data is composed of sequences of still frame images, the search for similar video data in a database necessarily requires a huge computation cost when comparing the query with the data in the database. Thus, the main issue when processing a video query is to reduce both the computation time for the search itself, and for the comparison of image sequences, to evaluate the similarity of the video segments. Video querying is an efficient way to issue a query when a user already has in hand an exact or an extremely relevant segment of the video in-search. For specific genres of video data, such as those obtained from a fixed surveillance camera or sports programs, querying based on certain motions, gestures or trajectories can be useful. This kind of querying can also be considered as a kind of video querying in the sense that it handles a query as a time-sequence transition of an interest point. In most other cases, keyword-based querying can be more efficient and realistic in general.

Historical Background

Although there are some academic works that make use of video segments as queries [3], video querying is still not widely employed for the purpose of video retrieval in general. It has, however, been a key technology for some real-world problems.

Most early works that make use of video segments as queries, aim to detect advertisements from a broadcast video stream; when a source video segment (an advertisement) is given, the system returns where it appears (repeatedly) in the archived long video stream [4,6]. This is often used by commercial sponsors to monitor if a television broadcaster has actually aired an advertisement at specific hours and/or for specific times, according to the contract.

Recently, the technology has started to be applied by copyright holders for monitoring video data illegally distributed on the internet. While the advertisement monitoring task requires exact matching, this task needs to handle near-duplicate video data, which are almost identical but may have slightly changed from the original data during the distribution process. The changes may include size, frame rate, compression, editing (ex. insertion of captions and logos) and so on.

The easiest approach to compare a pair of video data is to do so frame-by-frame. This approach is, however, highly expensive in computation cost; not only that the comparison needs to be done as many times as the number of the frames, but also the cost to compare each frame is itself relatively high compared to audio or text data. Thus, the key to realize this technology in a realistic speed is to reduce the size of the feature, and also to reduce the times of comparison.

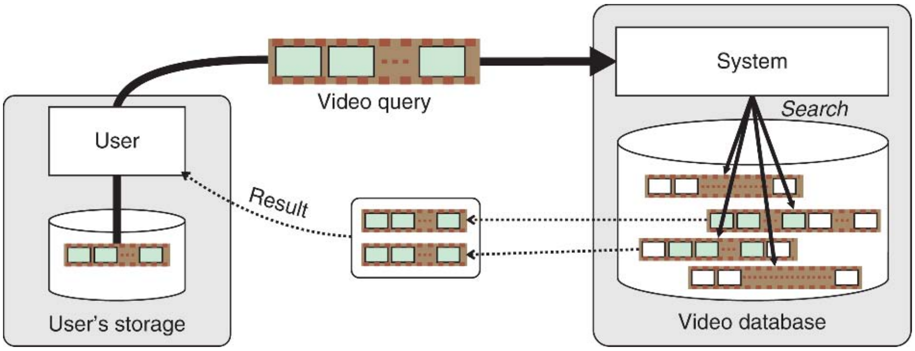
Foundations

Figure 1 shows the process flow of video querying. A user issues a query in the form of a video segment to a video database system. The query video segment should be readily available at the users hand, but it can also be provided externally through an interactive interface with a search function. For specific purposes such as action detection from fixed surveillance video, sports video, and so on, querying by motion, gesture, or trajectory [1, 2] can also be considered as a kind of video querying.

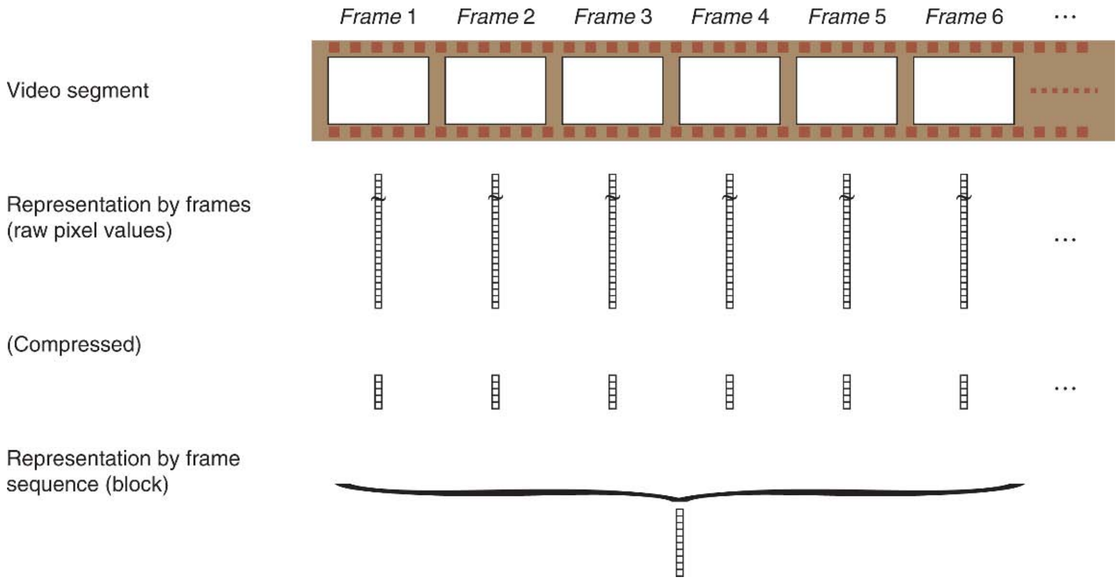
In order to process a video query, evaluation of the similarity between the query and the video data in the database is necessary.

The first question here is how to represent a video segment. Figure 2 shows an illustration of three representations explained below.

The simplest representation of a video segment is as a sequence of still frame images. Each image is represented by a feature vector composed of raw pixel values of all pixels in the frame image. It is, however, rarely the case



Video Querying. Figure 1. Process flow of video querying.



Video Querying. Figure 2. Representations of a video segment.

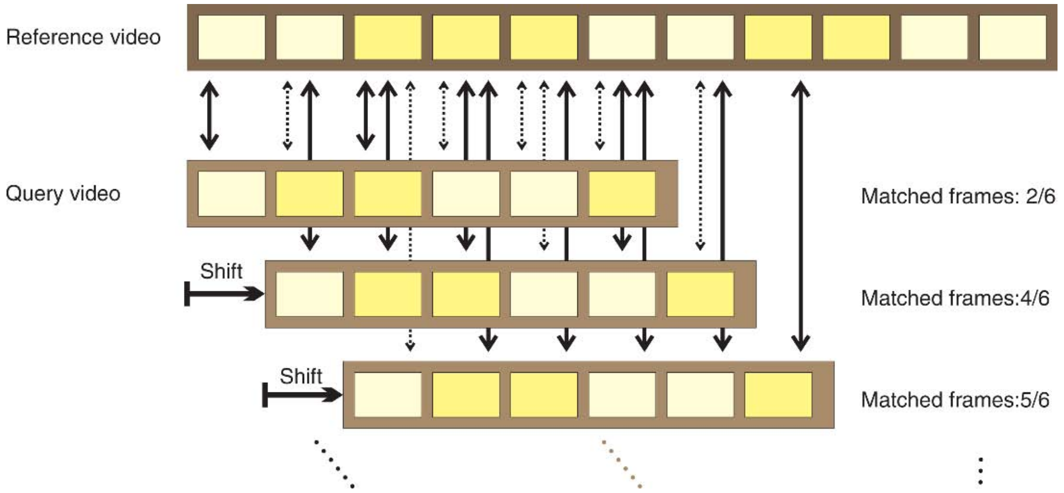
that exact pixel values between two video sequences are identical, due to noise, slight change of size / ratio, color adjustment, and so on. To cope with this problem, the image and the color resolutions are usually reduced, which is also the case for the original frame image used in the following two representations. Using raw pixel values is accurate, but the size of a feature vector tends to be large.

Since comparing large vectors is computationally expensive, it is better to reduce the size of a feature vector; the dimension of a feature space. Reduction of the dimension of a feature space can be done in various ways; from random sampling to hashing, signatures and transformations that preserve the original feature better by principal component analysis, and so on. On the other hand, the feature used to represent a frame can be more abstract features than the raw pixel values, such as edge, color histogram, or combinations of these features. In any case, representation in the compressed feature space does not usually guarantee the same distinguish ability as in the original feature space. When an identical result is needed, comparison in the original feature space after obtaining candidates in the compressed feature space is needed as a post-process in order to eliminate false positives. Note that in order to ensure that no false negative (oversight) exists among the candidates, the vector comparison method (distance measure) needs to theoretically guarantee it.

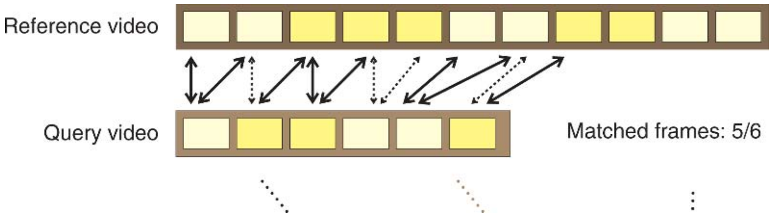
While the above two representations consider the feature of a video segment by a sequence of frame-wise features, it is more natural to represent a video segment by blocks of successive frames. In this case, image features from a fixed number of multiple successive frames represent a video segment. Ways to represent a block of frames can be a simple concatenation of frame-wise features (usually compressed afterwards), or can be more sophisticated features that make use of the redundancy of video data such as those used for video compression (MPEG and so on). This representation usually makes the comparison more robust and fast than the previous two representations, but it cannot represent, and accordingly compare and retrieve shorter segments than the block size.

The next question is how to compare the segments. Figures 3–5 show illustrations of the comparison methods explained below.

As shown in Fig. 3, the simplest method to compare the query video segment to the video data in the database (hereafter, “reference video”) is to compare it against all possible segments in the reference video. In this case, the query video is compared by shifting it frame by frame against the reference video until it reaches the end. For each iteration of the comparison, all frames are sequentially compared. This method is not robust even against slight change in the frame sequence; frame-rate, and editorial effects such as slow-motioning or fast-forwarding.



Video Querying. Figure 3. Simple frame-by-frame comparison.



Video Querying. Figure 4. Frame-by-frame comparison by continuous dynamic time warping (DTW), also known as DP-matching.

An improved version of this method that takes the above-mentioned weak-point in consideration, is the method that employs continuous Dynamic Time Warping (DTW; also known as DP-matching). This method allows the comparison of a query video frame to multiple (usually a fixed number of frames after the frame matched in the previous iteration) reference video frames. As shown in Fig. 4, the method could cope, to some extent, with temporal expansion and contraction in both sides of the comparison.

Although there are many works on speeding-up the above-mentioned methods, they essentially require large computation costs due to the frame-by-frame shift and comparison approach.

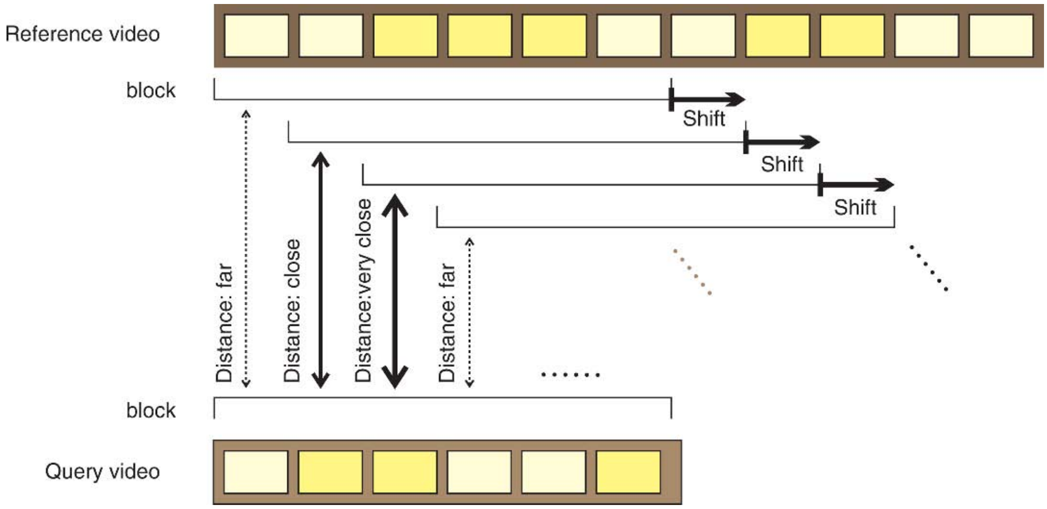
A different approach is to compare the video data as blocks of sequential frames. It does not necessarily have to be a simple concatenation of frame-wise features, but can be an integrated feature that represents the image features in the block. Block-wise comparison is robust to slight change of features in a frame sequence. Although in practice, it can tolerate time-wise expansion and contraction to some extent, it is mainly not due to the nature of the method, but rather due to the continuous nature of video contents. One of the fastest algorithm that takes this approach for video querying is the Time-Series Active Search method proposed by Kashino et al. [4], which reduces the computational cost by skipping the comparison under certain criteria.

Key Applications

As mentioned in the "Historical Background," the technology has been used by commercial sponsors of television programs and copyright holders of video contents.

Future Directions

Future applications include online retrieval of near-duplicate video data available on the internet. This application can be used not only for detecting illegal posting of copyrighted video data on the internet, but also for retrieval of web contents based on the video querying technology.



Video Querying. Figure 5. Comparison by block.

Another challenging task is to retrieve near-duplicate video segments from a video query based on feature points matching. This technology enables the retrieval of video segments shooting the same target from a different video camera, but current technology does not allow such processing in realistic time.

Cross-references

- ▶ Content-Based Video Retrieval
- ▶ Image Querying
- ▶ Multimedia Data Querying
- ▶ Principal Component Analysis

Recommended Reading

1. Aghbari Z., Kaneko K., and Makinouchi A. Content-trajectory approach for searching video databases. IEEE Trans. Multime., 5(4):516–531, 2003.
2. Chang S.-F., Chen W., Meng H.J., Sundaran H., Zhong D. VideoQ: An automated content based video search system using visual cues. In Proc. 5th ACM Int. Conf. on Multimedia, 1997, pp. 313–324.
3. Dimitrova N. and Abdel-Mottaleb M. Content-based video retrieval by example video clip. In Proc. Storage and Retrieval for Image and Video Database, 1997, pp. 59–70.
4. Kashino K., Kurozumi T., and Murase H. A quick search method for audio and video signals based on histogram pruning. IEEE Trans. Multime., 5(3):348–357, 2003.
5. Lienhart R., Effelsberg W., and Jain R. VisualGREP: a systematic method to compare and retrieve video sequences, Multime. Tool Appl., Kluwer, Hingham, MA, 10(1):47–72, 2000.
6. Lienhart R. Kuhmünch C., and Effelsberg W. On the detection and recognition of television commercials. In Proc. Int. Conf. on Multimedia Computing and Systems, 1997, pp. 509–516.

Video Representation

Ying Li
IBM T. J. Watson Research Center, Hawthorne, NY, USA

Synonyms

Video format; Video compression

Definition

Video representation, as the name implies, specifies a way of representing a video. While some work refers to video representation as the way to present or express video content through some extracted or summarized content units such as scenes or objects, the majority regard it more as the way the video content is stored. In other words, it is about *video format* which describes the sequence, structure and content of frames that create the moving video image, along with any possible audio or text (closed caption) information.

Historical Background

A video can be represented in either analog or digital formats. Typical analog formats include NTSC (National Television System Committee), PAL (Phase Alternating Line) and SECAM (Séquentiel couleur à mémoire, French for "Sequential Color with Memory,") which are commonly used in the domain of commercial broadcast. On the other hand, due to the rapid development of computer technologies, the continuously improved transmission rate and the increasing ubiquity of digital video capturing devices such as digital cameras and camcorders, digital videos have become widely available. This has entailed the development of various kinds of digital video formats suited for various purposes. Some of the most popularly used ones are DVD (Digital Versatile Disc), QuickTime, H.261, H.263, H.264, MPEG-1 (Moving Picture Experts Group), MPEG-2, and MPEG-4.

Digital video was first introduced commercially in 1986 with the Sony D-1 format, which recorded an uncompressed standard definition component video signal in digital form instead of the high-band analog forms that had been commonplace until then. Due to the expense, D-1 was used primarily by large television networks. It was eventually replaced by cheaper systems using compressed data.

Consumer digital video first appeared in the form of QuickTime around 1990, which is Apple Computer's architecture for time-based and streaming data formats. Also around the same time, the ITU-T (Telecommunication Standardization Sector of International Telecommunication Union) Video Coding Experts Group (VCEG) developed H.261 standard which aims for video transmission over ISDN (Integrated Services Digital Network) lines whose data rates are multiples of 64 kbit/s [6]. Based on the experience from H.261, H.263 was developed, which is a low-bitrate compressed format for video-conferencing [7]. Its first version was completed in 1995 and provided a suitable replacement for H.261 at all bit rates. The next enhanced compression mechanism developed by ITU-T VCEG is the H.264 standard, also known as AVC (Advanced Video Coding) and MPEG-4 part 10. H.264 provides a significant improvement in capability beyond H.263 [8]. Now, most new video-conferencing products include H.264 as well as H.263 and H.261 capabilities.

Along the same direction, but targeting at different commercial applications, the Moving Picture Experts Group (MPEG) was formed in 1998 to establish an international standard for the coded representation of moving pictures and associated audio on digital storage media. So far, there have been three established MPEG standards: MPEG-1, MPEG-2, and MPEG-4. Specifically, MPEG-1 was originally designed to achieve VHS-video quality at 1.5 Mbit/s data rate [2]. MPEG-1 video is usually used for the Video CD (VCD) format. Compared to MPEG-1, MPEG-2 targets at a very high bandwidth with up to 40 Mbits/s data rate, and is widely used as the format of digital television signals that are broadcast by terrestrial (over-the-air), cable, and direct broadcast satellite TV systems [4]. It also specifies the format of movies and other programs that are distributed on DVD and similar disks. In late 1998, an effort on MPEG-4 was initiated which added many new features such as VRML (Virtual Reality Modeling Language) support for 3D rendering, object-oriented composite files (including audio, video and VRML objects), support for externally-specified Digital Rights Management (DRM) and various types of interactivity, besides absorbing many features from MPEG-1 and MPEG-2 [5]. Major MPEG-4 applications include streaming media on web, video-phone and broadcast television.

Foundations

Basics of Video Representation

Frame is the fundamental building block of a video, which is basically a rectangular image consisting of a series of lines, known as scan lines. A scan line contains a given number of pixels, and a pixel is represented by a certain number of bits (*a.k.a.* bits per pixel or bpp). A frame can consist of two or more fields, which are sent sequentially and displayed over time to form a complete frame. This kind of assembly is known as interlace. An alternative way is to send the entire frame as a single entity, which is known as a progressive scan frame.

The basics that are needed to represent a video or to define a video format is listed below.

- *Video resolution*, which equals the frame size measured in pixels.
- *Aspect ratio*, which describes the dimensions of video screens and video picture elements or pixels. The screen aspect ratio of a traditional television screen is 4:3, while a high definition television is 16:9. On the other hand, while pixels on computer monitors are usually square, pixels used in digital videos usually have non-square aspect ratios.
- *Color space*, which specifies the video's color representation. For instance, the YUV color model where Y stands for the luma component (the brightness) and U and V are the chrominance (color) components, is used in the PAL, NTSC, and SECAM composite color video standards. Other popularly

used color models include RGB (Red, Green and Blue components) and YCbCr (Y for luma component, Cb and Cr for the blue and red chroma components).

- *Depth of color*, which indicates the number of distinct colors that could be represented by a pixel. This depends on the number of bits per pixel (bpp).
- *Interlacing* or *progressive*. Interlacing was invented as a way to achieve good visual quality within the limitations of a narrow bandwidth. The horizontal scan lines of each interlaced frame are numbered consecutively and partitioned into two fields: the odd field (upper field) consisting of the odd-numbered lines and the even field (lower field) consisting of the even-numbered lines. NTSC, PAL and SECAM all use interlaced formats. In contrast, with progressive scan systems, each refresh period updates all of the scan lines. This results in a higher perceived resolution and a lack of various artifacts that can make parts of a stationary picture appear to be moving or flashing.
- *Frame rate*, which indicates the number of frames per time unit of the video. PAL and SECAM standards specify 25 frames per second (fps), while NTSC specifies 29.97 fps. Film is shot at a slightly lower frame rate of 24 fps. Generally speaking, the minimum frame rate for achieving an illusion of moving images is about 15 fps.
- *Bit rate*, which measures the rate of the information content in a video stream. Bit rate is only defined for digital videos, and is usually quantified in units of bit per second (bit/s or bps) or Megabit per second (Mbit/s or Mbps). A higher bit rate generally indicates a better video quality. For instance, VCD has a bit rate of about 1 Mbps while DVD, which has a much better quality, has a bit rate of around 20 Mbps. Variable bit rate (VBR) is a strategy to maximize the video quality and minimize the bit rate. For fast motion scenes, VBR uses more bits than it does for slow motion scenes of similar duration to achieve a consistent visual quality. However, for real-time and non-buffered video streaming applications where the available bandwidth is fixed, *e.g.*, for video-conferencing delivered on channels of fixed bandwidth, a constant bit rate (CBR) must be used.

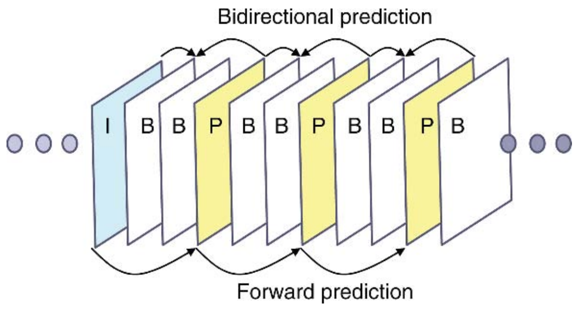
Video Compression

A video can occupy a vast amount of storage space when represented in digital form. For instance, suppose the video frame is of size 360 pixels by 288 pixels, and each pixel uses three color primaries with 8-bit precision for each color component, then each frame will occupy approximately 311 Kbytes. Further assume that the video is sent uncompressed at 24 frames/s, then the raw data rate for the video will be about 60 Mbps, which quickly amounts to 448 Mbytes for a 1 min video clip [10]!

Undoubtedly, there is a need of some techniques to compress or reduce the amount of video data so that videos can be conveniently and efficiently stored, transmitted and delivered, while retaining the original video quality as much as possible. Many research efforts on video compression started to emerge in 1990s. However, while there have been various kinds of compression techniques being proposed and standardized for various different application areas, most of them are based upon the same fact that video data contains both spatial and temporal redundancy. Specifically, to reduce the spatial redundancy, an intra-frame compression is applied which registers differences between parts of a single frame. On the other hand, to reduce the temporal redundancy, an inter-frame compression is exploited which registers differences between neighboring frames.

Below the basic spatial and temporal compression techniques that are applied in MPEG-1 video, are briefly discussed. Many other popularly used video compression standards such as H.261, H.263, H.264, MPEG-2 and MPEG-4, are more or less built upon similar techniques with certain improvements, extensions and modifications so as to meet different application requirements. These compression, together with the corresponding decompression algorithms are generally implemented in computer softwares as video codecs.

Frame Types in MPEG-1 There are basically three different types of frames in MPEG-1, namely, I-frame, P-frame and B-frame. Specifically, I-frames, also known as intra-coded frames, are coded independently without reference to any other frames. In contrast, P-frames (predictive-coded frames) obtain predictions from temporally preceding I- or P-frames in the sequence, which is known as forward prediction. Between the I- and P-frames, there may be zero or more bidirectionally predictive-coded frames, or B-frames. B-frames are interpolated between the preceding and/or upcoming I- or P-frames in the sequence. Figure 1 shows a group of pictures consisting



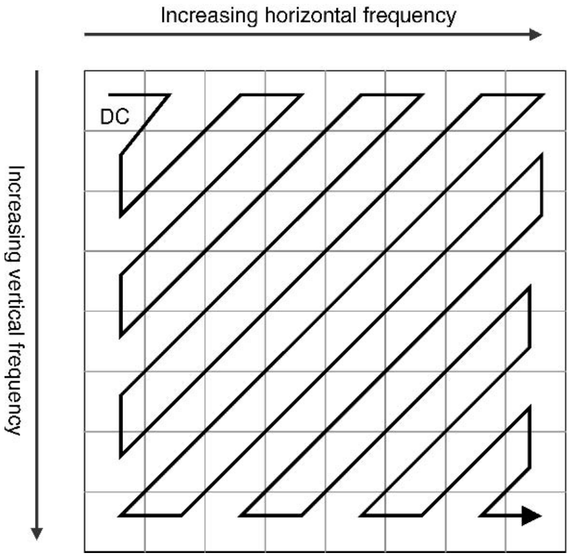
Video Representation. Figure 1. A typical group of pictures in display order in MPEG-1.

of one or more pictures of the three different types of frames.

Two different compression techniques are thus developed to code these three different types of frames. Specifically, intra-frame compression is applied to code I-frames, while inter-frame technique is applied to P- and B-frames. Both techniques are briefly discussed in the following sections.

Intra-Frame Compression *Macroblock*, which is a very important concept in MPEG coding, forms the basic building block of a coded frame. Specifically, a macroblock consists of a 16 sample array of luminance (grayscale) samples together with one 8 × 8 block of samples for each of two chrominance (color) components. The 16 sample array of luminance samples is actually composed of four 8 × 8 blocks of samples, which form the units of data that are actually fed to the compression models. Notice that a lower resolution is used for the chrominance blocks because the human eyes resolve high spatial frequencies in luminance better than in chrominance.

At the heart of both intra-frame and inter-frame coding in MPEG is a mathematical transform known as discrete cosine transform (DCT). DCT transform is a mapping function from time or space domain to frequency domain. Given an 8 × 8 image block, a DCT transform will convert it into 64 DCT coefficients. Among them, the first coefficient is called *DC coefficient* or DC term, and the others, *AC coefficients.* Specifically, the DC coefficient contains the block's average intensity, *i.e.*, the low frequency information, while the rest 63 AC coefficients contain high frequency information. Figure 2 shows the zigzag ordering which approximately orders the DCT coefficients in ascending spatial frequency.


Increasing horizontal frequency
Increasing vertical frequency
DC

Video Representation. Figure 2. Zigzag scanning order of DCT coefficients.

The advantages of applying DCT in the compression models are twofold: i) it de-correlates the original signal. That is, the 64 DCT coefficients are independent of each other, thus they can be coded separately; ii) it distributes the signal energy to only a small set of coefficients. Generally speaking, after the transform, many AC coefficients in high frequencies will be very small, which may be discarded without, or with little, loss of visual quality.

After each of the four 8 × 8 blocks of luminance samples and two 8 × 8 blocks of chrominance samples of a macroblock is processed by the DCT, a quantization process is carried out on the obtained DCT coefficients, which aims to represent the coefficients of high spatial frequencies with less precision. Specifically, it divides and truncates each of the transformed coefficients by individual quantization values. These values are usually given in a quantization matrix, which typically contains higher values towards the lower right, thus giving several of the less important coefficients at high spatial frequencies a zero value. This allows the encoder to selectively discard non-important high frequency activity that the human eye cannot readily perceive.

The next step is to code the resulted DC and AC coefficients. Specifically, as there could be some correlation between the DC coefficients of neighboring blocks, the DC term is coded separately from the AC terms using a predictive DPCM (differential pulse code modulation) technique. As for AC terms, they are first arranged qualitatively from low to high spatial frequency following the zigzag scan order as shown in Fig. 1. Such zigzag scan approximately orders the coefficients according to their probability of being zero. Finally, each nonzero AC coefficient is coded using a so-called *run-level* symbol structure where run refers to the number of zero coefficients before the next nonzero coefficient, and level refers to the amplitude of the nonzero coefficient. For more details on this part, please refer to [10].

Inter-Frame Compression Compared to the intra-frame compression, the inter-frame compression codes the difference between a frame and its reference frames, thereby exploiting the similarities or temporal redundancy from one picture to the next.

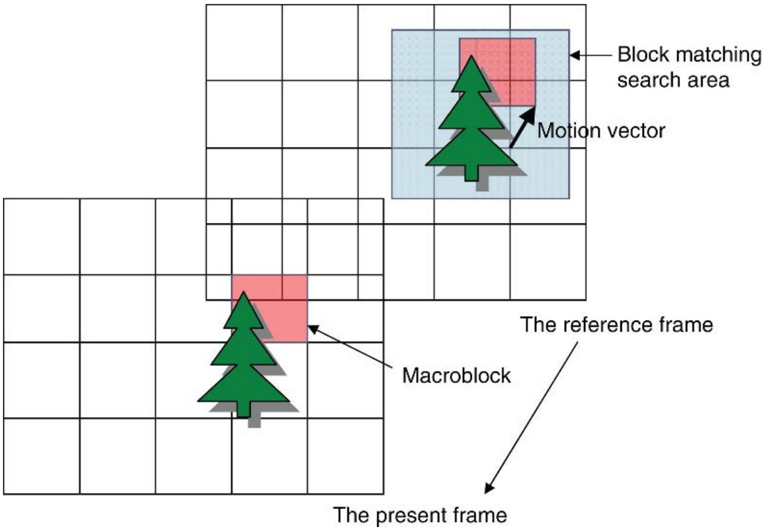
The most important concept in inter-frame compression is *motion estimation*. Specifically, motion estimation refers to the determination of the motion displacement between two frames which is expressed by motion vectors. In real video scenes, motion can be a complex combination of translation and rotation, which could be very difficult to estimate and may require large amount of processing. Nevertheless, translational motion could be easily estimated and has been successfully used for motion compensated coding.

There are two main classes of motion estimation techniques, namely, *pel-recursion* and *block matching*. Pel-recursion techniques are used primarily for systems where the motion vectors can vary from pixel to pixel. In contrast, in block matching techniques, there is only one single motion vector applied to a block of pixels. Due to its simple hardware realization and less computational complexity, the block matching approach has been more popularly used.

Figure 3 shows the basic idea of the block matching approach. Specifically, for each macroblock in the present frame, it is matched against a candidate block within a search area on the reference frame. If a good match is found based on certain criterion such as MSE (mean square error) and MAD (mean absolute distortion), the current macroblock will be represented by a motion vector pointing to the reference block, along with the DCT-coded residual. Otherwise, the macroblock will be intra-coded. Note that for B-frames, there are even more options for coding a macroblock, it could be forward predicted using the preceding I- or P-frame as the reference, backward predicted using the next I- or P-frame, and bidirectionally predicted using both preceding and succeeding I- or P-frames. Moreover, to achieve an even higher compression ratio, some macroblocks can be skipped.

Audio Compression

When a video stream also contains audio information, some kind of audio compression algorithm is usually



Video Representation. Figure 3. Block matching based motion estimation.

applied to condense the audio data, in addition to the aforementioned video compression. Generally speaking, many video codecs such as those introduced earlier (MPEG-1, MPEG-2, H.263 and H.264) also have corresponding specifications on encoding the embedded audio signals. For instance, MPEG-1 has defined three layers of audio coding in the ascending order of coding complexity: MPEG-1 Audio Layer 1 (MP1), Audio Layer 2 (MP2) and Audio Layer 3 (MP3) [3]. Below, a very high-level discussion on audio coding is given.

As with image or video compression, both lossy and lossless compression algorithms are used in audio compression. In both cases, techniques such as coding, pattern recognition and linear prediction are applied to reduce the amount of redundant information within the data. However, while lossless compression is good for the archival purpose, it generally produces much lower compression ratios due to the nature of audio waveforms and the fast-changing values of audio samples. Consequently, the lossy audio compression is much more popular. So far, it has found applications in video DVDs, digital television, streaming media, satellite and cable radio.

The key breakthrough in lossy audio compression is to rely on psychoacoustic models to identify the important audio signals that must be preserved, while throwing away unimportant ones. For instance, signals that can be perceived by the human auditory system must be kept, while perceptually irrelevant sounds could be ignored or coded with decreased accuracy as they are anyway very hard to hear. Typical examples of such sounds are those that have very high frequencies, or those that occur at the same time with other much louder sounds.

In particular, the following four human hearing characteristics, as recognized by the psychoacoustic model, are exploited in audio compression: *human hearing sensitivity*, *frequency masking*, *temporal masking* and *critical bands*. Specifically, by carefully conducting experiments, it is found that the general range of human hearing is from 20 Hz to 20 kHz, and the most sensitive range is from 2 kHz to 4 kHz. By frequency masking, it refers to the phenomenon in which a weak signal is made inaudible (*i.e.*, masked) by a simultaneously occurring stronger signal. In contrast, the temporal masking indicates the phenomenon in which a soft tone could not be heard immediately when there is a loud sound nearby. Finally, critical bands are obtained using a bandwidth classification scheme such that within each band, the width of frequency masking region is approximately uniform. As it could be seen that, by relying on these four characteristics, the compression algorithm could save bits on signals that are either out of human hearing ranges, temporally masked, or frequency masked within a critical band. During the actual coding, specific masking thresholds will be calculated by the psychoacoustic model to determine the audibility of each spectral component within particular critical band.

There are two general types of audio coding schemes, one in the frequency domain and the other, the time domain. Transform coding and subband coding belongs to the first category, while PCM (pulse code modulation), DPCM (differential pulse code modulation) and ADPCM (adaptive differential pulse code modulation) are in the second category.

Streaming Videos

Since the late 1990s, there have been some great advances in computer networking such as a higher network bandwidth, an increased access to networks, especially, the Internet, a widely accepted use of standard protocols and formats including TCP/IP (transmission control protocol/Internet protocol), HTTP (hyper text transfer protocol), and HTML (hyper text markup language), and the commercialization of the Internet. These advances, combined with increasingly powerful home computers and modern operating systems, have made streaming media practical and affordable for ordinary consumers.

By definition, streaming videos are the videos that are continuously received by and displayed to the end-users while the videos are being delivered from the providers. The name "streaming" refers to the delivery method of the medium rather than the medium itself. A video stream can be either *on demand* or *live*. On demand streams are stored on a server for a long period of time, and are available to be transmitted at a user's request. Live streams are only available at one particular time, such as the video stream of a live sporting event.

Some popular streaming video and streaming media technologies include Microsoft Windows Media, RealMedia from RealNetwork, Quicktime, MPEG-4, and Adobe Flash. And some major streaming video content providers include YouTube, Google Video, Netflix, Stage 6, and Metacafe.

Key Applications

With the proliferation of videos across the world, how to efficiently and effectively store, process, transmit, deliver and present them to end users has become an increasingly popular research topic. Video representation is undoubtedly one of these important issues. On one hand, the wide availability of videos has motivated the development of advanced video compression techniques, yet on the other hand, accomplishment achieved in the compression areas has made videos further distributed.

The benefits brought by efficient video representation are significant. Some popular applications that take advantage of advanced video compression techniques include: (i) HDTV and mobile broadcast terrestrial digital television (DTV); (ii) gaming consoles; (iii) network database services such as video library and video information provider; (iv) video on demand (VOD); (v) digital storage media such as CD-ROM videos, DVD movies and digital recorders; (vi) interactive communications such as two-way video phones, video-conferencing and videotex; and (vii) video surveillance.

Future Directions

Approximately every 3 to 4 years, a significantly new standard on video compression is developed. The latest generation is the MPEG-4 Part 10 advanced video coding (AVC). This is also known as the H.264 standard and is developed to achieve broadcast-quality video at much lower bit rates. The algorithm offers compression percentages that are twice as much as those offered by MPEG-4 Part 2 with the same image quality. It is also designed for a wide variety of consumer and pro AV (audiovisual) applications including TV set-top boxes and DVD players. MPEG-4 Part 10 (or H.264) improves older compression schemes by using significantly more sophisticated predictive coding, as well as variable block-size motion compensation and variable block-size integer discrete cosine transform [8]. However, the not-so-good news about MPEG-4 AVC is that it is extremely computation-intensive and requires a very high horsepower hardware platform to correctly implement it [9].

H.264 is now still in its preliminary stage in terms of adoption, although many vendors and analysts believe that it will eventually grab a big market share. It is expected that H.264 will be used by many digital video delivery networks and the new DVD standards such as high definition (HD)-DVD and Blu-Ray.

Another two directions to watch for future video compression are object-based video coding and scalable video coding (SVC). The basic idea of object-based video coding is to first segment a video scene into objects, then code them separately. MPEG-4 has actually explored such idea in its video codec, nevertheless, as object segmentation is a fairly challenging task and remains to be an open research topic, such object-based video coding still has a long way to go before it can truly take off. As for the scalable video coding, it is sort of a natural follow-on of H.264. In particular, it aims to provide a better way for dealing with wild cards such as disparate network types and different endpoint abilities, including display resolution and processing power [1].

In a word, while the network bandwidth has constantly increased in recent years, the emerging of high-definition videos effectively cancels out some of these gains. Moreover, considering the formidable high bit-rate of raw video data, video compression techniques are bound to be in great need for the future.

Cross-references

▶ Audio Representation
▶ Image Representation
▶ Video

Recommended Reading

1. International Standard ISO/IEC JTC1/SC29/WG11. N7315: Introduction to SVC extension of advanced video coding, 2005.
2. International Standard ISO/IEC 11172-2. Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s – Part 2: Video, 1993.
3. International Standard ISO/IEC 11172-3. Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s – Part 3: Audio, 1993.
4. International Standard ISO/IEC 13818-2. Information technology – Generic coding of moving pictures and associated audio information: Video, 2000.
5. International Standard ISO/IEC 14496-2. Information technology – Coding of audio-visual objects – Part 2: Visual, 2004.
6. International Telecommunication Union, Telecommunication Standardization Sector (ITU-T). ITU-T recommendation H.261: Line transmission of non-telephone signals – Video codec for audiovisual services at $p \times 64$ kbits, March 1993.
7. International Telecommunication Union, Telecommunication Standardization Sector (ITU-T). ITU-T recommendation H.263: Line transmission of non-telephone signals – Video coding for low bitrate communication, 2005.

8. International Telecommunication Union, Telecommunication Standardization Sector (ITU-T). ITU-T recommendation H.264: Advanced video coding for generic audiovisual services, 2005.
9. Kridel T. The future of video compression. Downloadable at: http://proav.pubdyn.com/Tech_Apps/Septemb912200532206PM.htm.
10. Mitchell J., Pennebaker W., Fogg C., and LeGall D. MPEG video compression standard. Chapman and Hall, New York, NY, USA, 1992.

Video Retrieval

► Video Sequence Indexing

Video Scene

► Video Scene and Event Detection

Video Scene and Event Detection

Noboru Babaguchi, Naoko Nitta
Graduate School of Engineering, Osaka University,
Osaka, Japan

Synonyms

Video scene and event extraction

Definition

A video scene, also called a Logical Story Unit [7] or simply a story unit, can be defined as a semantically related consecutive series of image frames that depicts and conveys a high-level concept such as *event*, *topic*, *object*, *location*, and *action*, which constitutes a story in a video. Especially, an event can be defined as an incident or situation, which occurs in a particular place during a particular interval of time, for example – homerun in a baseball game, actor's entrance on stage, car explosion on a highway, etc. Under these definitions, video scene and event detection is to find all video intervals corresponding to a specific event from a given video.

Historical Background

Video scene and event detection has been an active research area in the community of multimedia signal processing and computer vision, and has attracted much interest in many applications such as multimedia information retrieval, video archive indexing and management, real-time adaptive streaming, and security monitoring.

For example, in sports videos, various score event scenes, e.g., touchdown in American football, homerun in baseball, and 3-point shoot in basketball are actually what viewers want to watch. Video retrieval based on such events is a typical example of multimedia information retrieval. Real-time adaptive streaming can also be achieved by transmitting only these score event scenes with full motion audio-video, while transmitting other irrelevant video segments with less information such as only keyframes. In addition, *annotations* of each event scene about "Five Ws and One H" – who, when, where, what, why, and how – are of much importance because such data enables efficient and effective video archive indexing and management. Event detection provides when and what attributes, e.g., *the homerun* in *the top of the 8th inning*. As another example, the security monitoring can be realized by detecting unusual events in surveillance videos in order to reduce the volume of data presented to security personnel. As can be seen, video scene and event detection is one of the most important tasks in video content analysis.

Foundations

Video scene and event detection requires both video scene detection and event detection. Here, note that the target events can be categorized into two types: *known* and *unknown* events. For both types of event, shot detection, where a *shot* is defined as a set of contiguously recorded image frames, is usually the first step towards video scene and event detection. For unedited continuously captured videos, this can be realized by segmenting the video into fixed-length units. After that, there are mainly two types of approach: i) detecting events and scenes simultaneously based on heuristically/statistically determined rules, and ii) detecting scenes first and then classifying the detected scenes into predefined event classes. Both approaches can be applicable to detect known events, while the latter approach is usually used for detecting unknown events. More details are discussed below.

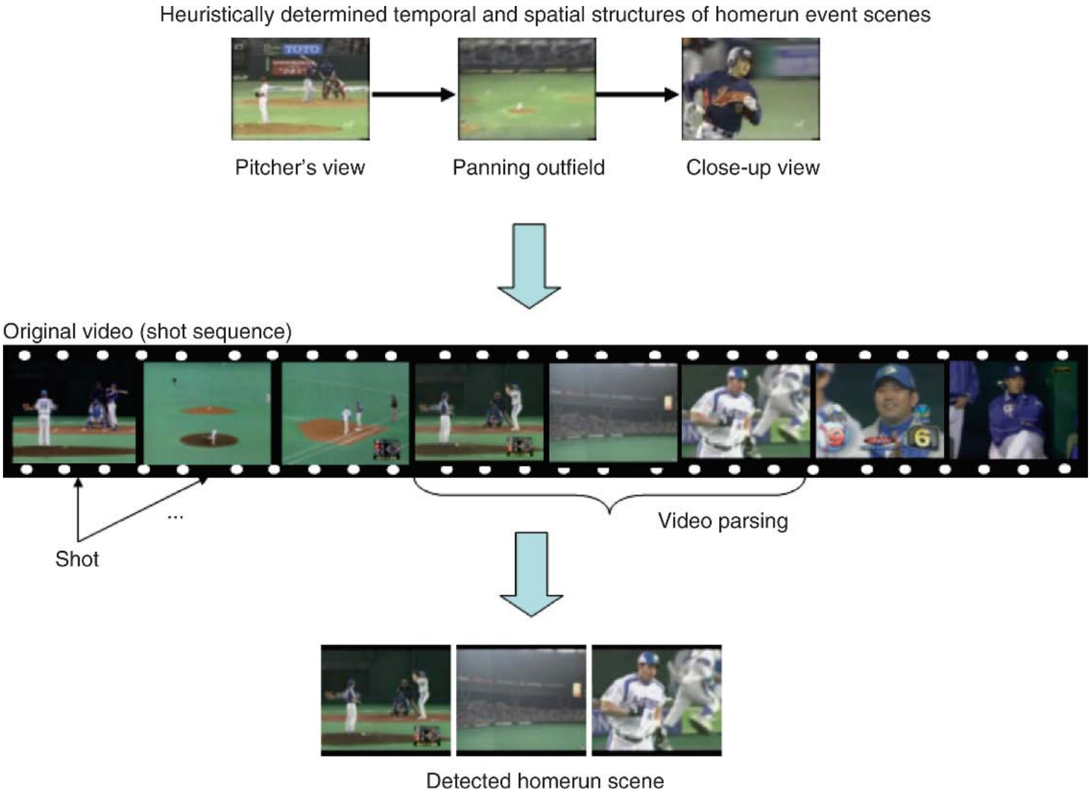
The primary event detection approach is to detect known event scenes based on heuristically determined rules. An example of known events is *homerun* in a baseball game. Broadcast baseball videos usually have

V

well-defined structures, for example, a typical home-run event usually consists of three or more shots, which starts from a pitcher's view, followed by a panning outfield and audience view in which the video camera tracks the flying ball, and ends with a global or close-up view of the player running to home base. These elemental shots consist of characteristic low- to mid-level image features such as color, texture, shape, motion, object trajectories, and human faces. Therefore, according to such *domain knowledge*, a series of certain types of shots is heuristically predefined as the temporal and spatial structures of the event scene, and then a video is parsed to find the corresponding event scenes as shown in Fig. 1.

It is noted that the approach discussed above focuses on single modality of the video data, in this case, the image stream. The single modality based approaches have low computational load, but the accuracy of event detection is low, as only using single modality is not able to fully characterize the events. For example, there are several cases for touchdown in American football games, such as the touchdown by pass, by running, and after turnover. It is not easy to construct a compact visual model that covers a number of cases of concern.

Alternatively, great emphasis has been placed on multimodality of the videos in recent years [2,3,8,11,13]. As is well known, the video data is composed of temporally synchronized *multimodal streams*: visual, auditory, text, and graphics streams, which are closely related to each other. The visual stream is a sequence of image frames, and the auditory stream is a mixture of a couple of auditory sources such as speech, music, and sound. In addition, the closed caption (CC) text, which is a transcript of the speech part of the auditory stream, can be viewed as the text stream for broadcast videos. The graphics stream is a sequence of overlays or video captions that have rich information about the content of the video data. The aim of multimodality based approaches is to improve the reliability and efficiency in analyzing the semantic content of videos. Although the computation for analyzing the visual stream is most costly, the use of other streams may be capable of reducing it. It should also be added that external metadata such as scenarios and web-cast game statistics is often used as one of the multimodal

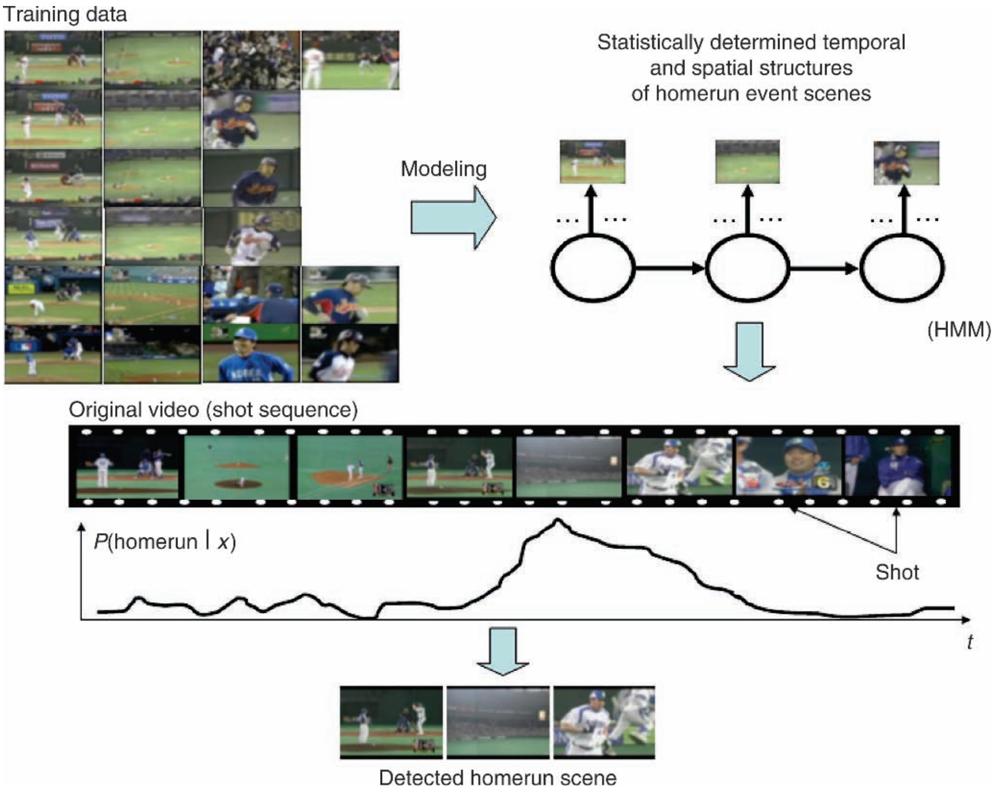

Heuristically determined temporal and spatial structures of homerun event scenes
Pitcher's view
Panning outfield
Close-up view
Original video (shot sequence)
Shot
...
Video parsing
Detected homerun scene

Video Scene and Event Detection. Figure 1. Heuristically determined rule-based approach for known events.

streams [4]; however, since the metadata is generally created independently from the video, it needs to be synchronized with other streams according to the semantic content at some point. Note that these multimodal streams are also used for other types of approaches described later.

These traditional rule-based approaches, where humans have to discover the domain knowledge and encode it into a set of programming rules, are too costly and incompetent for multimedia content analysis because knowledge for recognizing high-level events could be very complex, vague, or difficult to define. Therefore, on alternative solution is to use statistical models to learn the temporal and spatial structures of the event scene from training data which is given beforehand as shown in Fig. 2. Hidden Markov Models (HMMs), being a popular choice for probabilistic representation of sequences, are often used to learn these structures of a specific event and to detect the corresponding scenes from the video [15]. Several effective extensions to HMMs have been used to model more complex structures and to achieve better performance; especially, the coupled HMM (CHMM) has been developed to model interacting processes of multimodal streams.

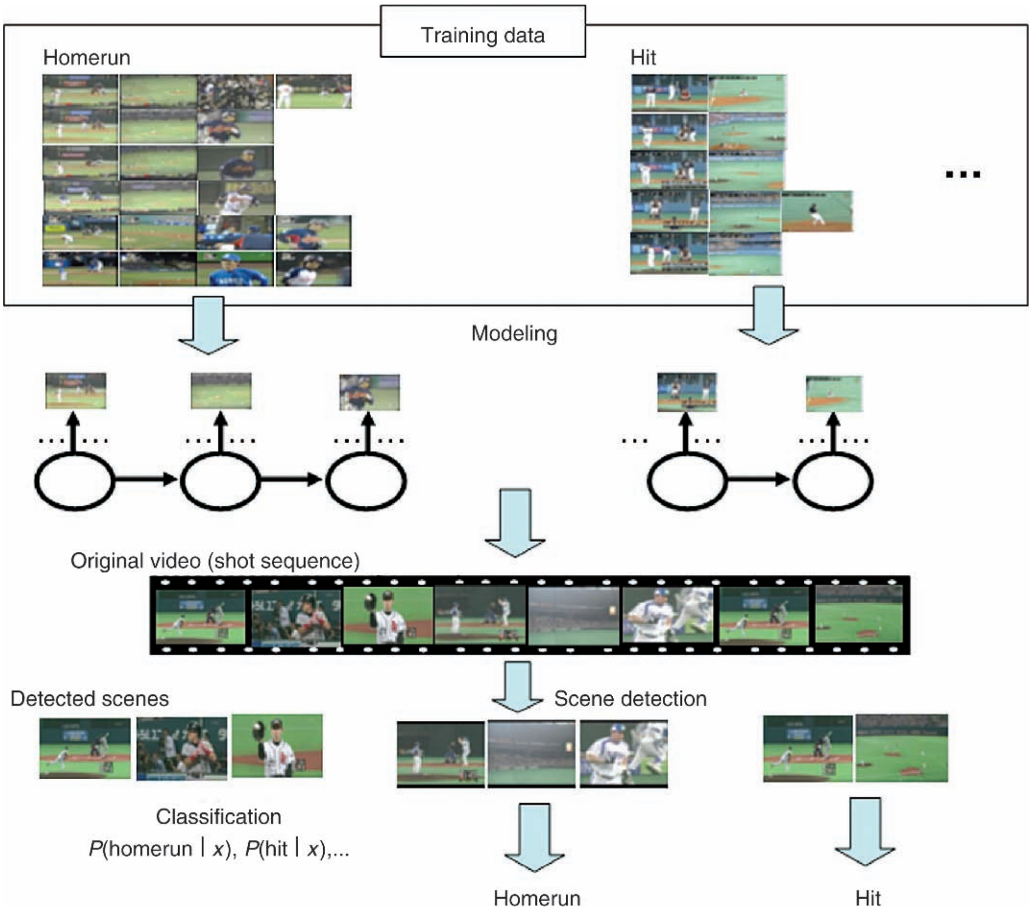
Now, there is a different type of solution for event detection, which is to firstly detect scenes and then classify the detected scenes into predefined event classes. In this case, semantically related shots are firstly grouped into a scene under the assumption that semantically related shots are usually temporally close to each other and have visual/audio similarity [7,9,10,12]. For example, if a person is talking in different shots, his/her speeches in these shots should present similar audio characteristics and these shots can be grouped as a scene. Then, the detected scenes are classified into a certain number of event classes based on the statistical model of each event [1]. More precisely, given a finite set of event classes $C = \{1,2,\ldots,K\}$ and a scene s, probabilistic classification methods typically compute the probabilities $P(k|s)$ that s belongs to class $k \in C$, and then classify



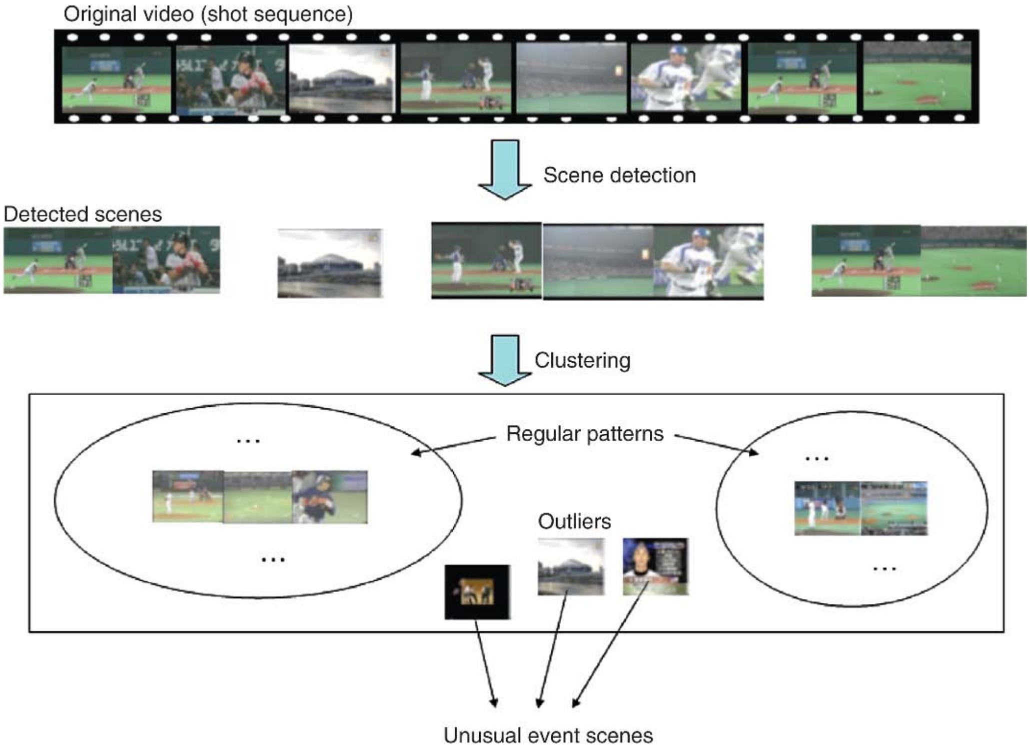
Video Scene and Event Detection. Figure 2. Statistical model-based approach for known events (with scene segmentation).

s into the class l that has the highest conditional probability $l = \mathrm{argmax}_k\, P(k|s)$. Figure 3 shows the schematic diagram of this approach. In general, there are two ways of learning $P(k|s)$: generative and discriminative. Discriminative models strive to learn $P(k|s)$ directly from the training set without the attempt to model the observation s. Generative models, on the other hand, compute $P(k|s)$ by first modeling the class-conditional probabilities $P(s|k)$ as well as the class probabilities $P(k)$, and then applying the Bayes' rule as follows: $P(k|s) \propto P(s|k)P(k)$. Popular generative models include HMMs, while representative discriminative models include Support Vector Machines (SVMs) [6].

Although high classification accuracy can often be achieved by these statistical model-based approach, the drawback is that the model developed for one application usually does not fit for others. For example, the homerun model developed for baseball games is obviously not suitable for detecting the 3-point shoot scenes in basketball videos. Similarly, when detecting unknown events, it is quite unnatural to have their statistical models created from the training data since there would be so much variety for unknown events. Recently, much work has been done to detect unusual events as unknown events. The basis of detecting unusual events is the notion of usual events and an underlying distance/similarity metric. The existing work mainly solves this problem by finding regular patterns, which are typically found with clustering operations on a given dataset, as usual events. Here, unsupervised classification methods automatically partition the given data set into the predefined number of clusters. Some methods are even able to automatically guess the optimal number of clusters into



Video Scene and Event Detection. Figure 3. Statistical model-based approach for known events (without scene segmentation).



Video Scene and Event Detection. Figure 4. Statistical model-based approach for unknown events.

which the given data set should be partitioned. Then, unusual events are detected by finding outliers that do not fit the current set of usual event clusters [5] as shown in Fig. 4.

Key Applications

Video Indexing Systems: are used to insert metadata about events for the corresponding video scenes so that they can be easily found at a later time.
Video Scene Retrieval Systems: are used to find one or more media clips that match an event-based user query, which can be specified in a variety of ways, such as text and some media examples.
Video Summarization Systems: are used to represent one media clip or a collection of clips in a condensed form. The presentation style includes the *storyboard*, where the representative images of all/important events are displayed in the spatial layout, and *skims*, where only the important event scenes are concatenated. Event metadata can help decide which part of the content shall be presented or emphasized in a concise and coherent manner.

Future Directions

Despite a great amount of research efforts, the success of video scene and event detection methods is limited due to the big *semantic gaps* between the low-level features used by these systems and the high-level semantics of events, that is, the low-level features such as color, texture, and motion are insufficient to capture the intrinsic nature of events even throughout the videos of the same genre, mainly due to the differences of their recorded situation. Therefore, a conclusion that can be drawn here is that the key to the success of video scene and event detection systems lies in the degree to which the semantic gaps can be bridged.

Experimental Results

Common evaluation measures for event detection are:

$$precision = \frac{the\ number\ of\ correctly\ detected\ events}{the\ number\ of\ all\ detected\ events}$$

$$recall = \frac{the\ number\ of\ correctly\ detected\ events}{the\ number\ of\ the\ events\ that\ should\ be\ detected}.$$

In general, for most existing methods, there are accompanying experimental evaluations with these measures; however, it is quite difficult to compare their performances since a different dataset is used in these experiments.

Data Sets

The TREC Video Retrieval Evaluation (TRECVID) [14], sponsored by the National Institute of Standards and Technology (NIST) and other U.S. government agencies, has been used as a benchmark dataset to encourage research in video content analysis. Although video scene and event detection has not been explicitly stated as their tasks, there are some related tasks, e.g., story segmentation can be considered as the equivalence to scene detection and high-level feature extraction includes the detection of events such as Monologue, Physical violence, People walking, and Airplane takeoff. Further usage of such benchmark datasets is expected for video scene and event detection in order to create objective measurements of the performance of each approach.

Cross-references

▶ Content-based Video Retrieval
▶ Video Content Analysis
▶ Video Content Modeling
▶ Video Content Structure
▶ Video Segmentation
▶ Video Sequence Indexing
▶ Video Shot Detection, Video Summarization

Recommended Reading

1. Adams B., Amir A., Iyengar G., Lin C.-Y., Naphade M., Neti C., and Smith J.R. Semantic indexing of multimedia content using visual, audio and text cues. EURASIP J. Appl. Signal Processing, 2:1–16, 2003.
2. Babaguchi N., Kawai Y., and Kitahashi T. Event based indexing of broadcasted sports video by intermodal collaboration. IEEE Trans. Multimedia, 4(1):68–75, 2002.
3. Babaguchi N. and Nitta N. Intermodal collaboration: a strategy for semantic content analysis for broadcasted sports video. In Proc. Int. Conf. Image Processing, 1:13–16, 2003.
4. Chua T.-S. and Xu H. Fusion of AV features and external information sources for event detection in team sports video. ACM Trans. Multimedia Comput. Commun. Appl., 2(1):44–67, 2006.
5. Goh K.-S., Miyahara K., Radhakrishan R., Xiong Z., and Divakaran A. Audio-visual event detection based on mining of semantic audio-visual labels. MERL, TR-2004-008, 2004.
6. Gong Y. and Xu W. Machine Learning for Multimedia Content Analysis. Springer, Berlin, 2007.
7. Hanjalic A., Lagendijk R.L., and Biemond J. Automated high-level movie segmentation for advanced video-retrieval systems. IEEE Trans. Circ. Syst. Video Techn., 9(4):580–588, 1999.
8. Hauptmann A.G. and Smith M.A. Text, speech, and vision for video segmentation: the informedia project. In Proc. AAAI Symp. on Computational Models for Integrating Language and Vision, 1995, pp. 90–95.
9. Li Y. and Kuo C.-C.J. Video content analysis using multimodal information: for movie content exraction, indexing and representation. Kluwer, Norwell, MA, USA, 2003.
10. Lienhart R., Pfeiffer S., and Effelsberg W. Video abstracting. Commun. ACM, 40(12):55–62, 1997.
11. Merlino A., Morey D., and Maybury M. Broadcast news navigation using story segmentation. In Proc. 5th ACM Int. Conf. on Multimedia, 1997, pp. 381–391.
12. Rui Y., Huang T.S., and Mehrotra S. Constructing table-of-content for videos. ACM Multimed. Syst. J., 7(5):359–368, 1999.
13. Sundaram H. and Chang S.-F. Computable scenes and structures in films. IEEE Trans. Multimedia, 4(4):482–491, 2002.
14. The National Institute of Standards and Technology (NIST). TREC video retrieval evaluation. 2001–2007, http://www-nlpir.nist.gov/projects/trecvid/
15. Xie L., Xu P., Chang S.-F., Divakaran A., and Sun H. Structure analysis of soccer video with domain knowledge and hidden Markov models. Pattern Recogn. Lett., 25(7):767–775, 2004.

Video Search

▶ Video Sequence Indexing

Video Segmentation

Nevenka Dimitrova, Lalitha Agnihotri,
Mauro Barbieri, Hans Weda
Philips Research, Eindhoven, The Netherlands

Synonyms

Video partitioning; Scene change detection; Shot-cut detection; Shot segmentation; Video shot-cut detection; Video chaptering; Logical story unit segmentation

Definition

Video (temporal) segmentation is the process of partitioning a video sequence into disjoint sets of consecutive frames that are homogeneous according to some defined criteria. In the most common types of segmentation, video is partitioned into *shots*, *camera-takes*, or *scenes*. A *camera take* is a sequence of frames captured by a video camera from the moment it starts capturing to the moment it stops. During *montage*, camera takes are trimmed, split, and inserted one after the other to compose an *edited version* of a video. The basic element of an edited video is called *shot*. A *shot* is a contiguous sequence of frames belonging to a single camera take in an edited video. Content-wise, shots usually possess some degree of visual uniformity. A *scene* is a group of contiguous shots that form a semantically meaningful set.

If a video V is represented as a finite sequence of frames $V = (f_1,\ldots,f_n)$, the temporal segmentation $S(V)$ of a video is a partition of the video into non-overlapping *video segments* v:

$$S(V) = \{v_1, v_2,\ldots,v_m\},$$

with $v_i \cap v_j = \phi$ and $\cup v_i = V \quad \forall i,j \in [1,\ldots,m]$, where a *video segment* is a finite sequence of consecutive frames $v = (f_p,\ldots,f_q\} \quad 1 \leq p \leq q \leq n.$

Historical Background

There have been many reports in the literature on methods for video shot-cut detection. The first report was published in 1992 (from a conference held in 1991) by Nagasaka et al., and described a way of indexing video objects by first detecting video cuts and then finding important objects in those frames [6]. Some of the reported methods worked on various ways of comparing pixel differences between frames. The methods presented in [5–7] operated in the spatial domain. In order to analyze an MPEG or motion JPEG stream, the frames in the stream would have to be fully decompressed, which was very computationally expensive. Later, other techniques have been applied in the compressed domain [2,12]. Some of these techniques require images obtained from an uncompressed stream to be fully compressed to take advantage of some of their special features. A method that was introduced by Swanberg et al. [7] actually went beyond the notions at that time that "cutting" and boundaries have to be at a consecutive frame level. The conceptual notion of "video parsing" was introduced where "video" is a multimedia language conveying more than the sum of the individual pixels and signals.

A model driven approach to digital video segmentation is presented by Hampapur and his colleagues in [5]. The paper deals with extracting features that correspond to cuts, spatial edits, and chromatic edits. The authors present extensive formal treatment of shot boundary identification based on models of video editing effects.

Arman et al. [1] have used a DCT approach on both JPEG and MPEG streams. For MPEG streams, only I-frames are analyzed. This implementation employed a two step approach. The first pass compares the images based on using selected DCT coefficients from selected blocks. Video frames are represented by a vector of this subset of DCT coefficients. Then the normalized inner product is subtracted from one another and compared to a threshold. If a potential cut is detected, the images can be decompressed for further processing.

A similar multi-pass approach has been used by Zhang et al. but their technique also analyzes the B and P frames in an MPEG stream [12]. The first pass compares the images based on DCT coefficients. The DCT comparison is based on a pair-wise block comparison algorithm which compares the DCT coefficients of corresponding blocks of consecutive video frames. For example, if block n in frame i has a DCT coefficient $C_{n,k}(i)$ where k is from 1 to 64 and n depends on the frame size, then the difference between two frames which are j frames apart is:

$$Diff_n = \frac{1}{64}\sum_{k=1}^{64}\frac{|C_{n,k}(i) - C_{n,k}(i+j)|}{\max(C_{n,k}(i), C_{n,k}(i+j))}$$

If $Diff_n$ is greater than some threshold t, then the block is different between the two frames. If the percentage of blocks changed is greater than some threshold t_b, it is considered to be a shot-cut. It is reported that t tends not to vary across video sources and can be easily experimentally determined. However, t_b is computed from the overall statistics for the values of the percentage of blocks that have changed which exceed the mean by approximately five standard deviations. In the second pass the number of motion vectors is compared to a threshold. If there are less motion vectors than some threshold, a shot-cut is determined to occur. This is based on the fact that if there are high residual errors, then there are no motion vectors, consequently, this block is sufficiently different and can not be predicted. If a large number of blocks can not be predicted, than it probably is a shot-cut. The performance of this method depends to a certain extent on the quality of the encoding scheme. An encoder with poor motion estimation may lead to poor shot-cut detection performances.

Yeo et al. have investigated using only the DC values of the DCT coefficients for comparing frames in the compressed domain [9]. They sum the DC difference between successive frames. If the difference is the maximum and n times larger than the next largest peak in a sliding window of frames, then it is a cut. They also detect "gradual transitions" such as dissolves and fades, by comparing each frame to the subsequent *k*-th frame over some time interval.

An alternative shot segmentation method was proposed by Zabih et al. in [11]. Their approach is based on the observation that, during shot-cuts, new edges appear far away from old ones, and old edges disappear in locations far away from new ones. Shot-cuts are detected by counting the number of edges *entering* and *exiting* pixels in consecutive frames.

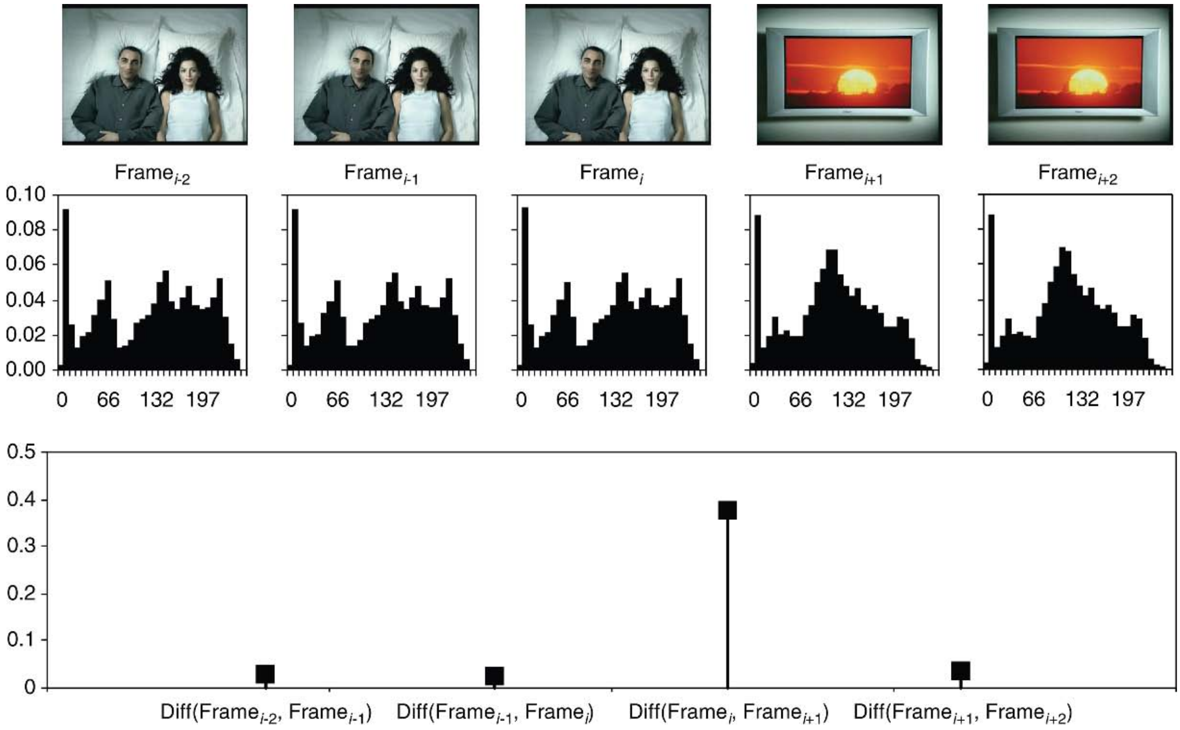
All of the above techniques have reported good results for shot-cut detection. However, a comparison of algorithms to detect shots boundaries has been performed by Boreczky and Rowe [2]. They selected and implemented some of the above algorithms. Their results showed that the simplest algorithms outperformed the more complicated methods and that DCT based algorithms had the lowest precision for a given recall.

Foundations

Shot-Cut Detection

Video segmentation aims at finding the temporal boundaries that separate visually homogeneous sequences of frames. The basic idea used by most methods for video segmentation is to compare subsequent frames for determining (based on some kind of threshold) when a significant change in the content occurs. For this purpose, it is necessary to define a metric for the comparison of subsequent frames and a thresholding method.

The key concept is visualized in Fig. 1. On the top of the figure, five frames across a shot boundary are shown. Below each frame, the corresponding luminance histogram is plotted. The plot shows on the horizontal axis the luminance values (from 0 to 255), and on the vertical axis the fraction of the pixels (from 0 to 1) in the frame image that have a particular luminance value. In this case, the metric for comparing two subsequent frames is the difference between the luminance histograms. The value of such distance is plotted in the graph at the bottom of the figure. As it can be seen in the histogram difference plot, the distance between the histograms of frames *i* and *i*+1 is considerably higher than



Video Segmentation. Figure 1. Example of a basic method for shot cut detection.

the differences between the histograms of other pairs of frames. The shot boundary located between the frames *i* and *i*+1 is easily detectable by means of a threshold located, for example, at 0.3.

A very simple metric for comparing frames is the *pixel-wise frame difference* consisting in the absolute difference of the intensity of the pixels between two consecutive frames. If $Y(x,y,n)$ and $Y(x,y,m)$ indicate the intensities of the pixels at position (x,y) in the frames n and m, the pixel-wise frame difference $\Delta_{n,m}$ is defined as follows:

$$\Delta_{n,\,m} = \sum_{x,\,y} |Y(x, y, n) - Y(x, y, m)|,$$

where the summation extends to the whole frame.

The pixel-wise frame difference is sensitive to noise, object and camera motion, and can lead to a high number of false detections. Better performances can be achieved by performing block-based differences instead of comparing single pixels.

Even better performances can be achieved by comparing image histograms. With respect to comparing pixels of consecutive frames, the histogram of a digital image increases robustness to noise, object and camera motion and is rotation invariant. Although image histograms do not retain any spatial information, they represent a good compromise between computational cost and performance for shot-cut detection.

If $H(i,\ n)$ represents the i-th histogram value for frame n and $H(i,\ m)$ represents the i-th histogram value for frame m, a simple but effective comparison metric is the *bin-to-bin* difference:

$$\Delta_{n,\,m} = \sum_{i} |H(i, n) - H(i, m)|$$

Shot-cuts are detected whenever the *bin-to-bin* difference exceeds a threshold. Other comparison metrics for histograms are the *intersection*, the *chi-square test* and the *correlation* defined as follows:

Histograms intersection	$\Delta_{n,\,m} = 1 - \sum_{i} \min(H(i,n), H(i,m))$
Chi-square test	$\Delta_{n,\,m} = \sum_{i} \frac{[H(i,n) - H(i,m)]^2}{[H(i,n) + H(i,m)]^2}$
Histograms correlation	$\Delta_{n,\,m} = 1 - \frac{\sum_{i} [H(i,n) - \mu_n][H(i,m) - \mu_m]}{\sigma_n \sigma_m}$

where μ_n and σ_n represent the mean and the standard deviation, respectively, of the histogram of the frame *n*.

The choice of a thresholding method is critical regardless of the comparison metric used. A too low threshold may create a high number of false detections, while a too high threshold may lead to many missed detections. A fixed threshold can be easily determined experimentally for one video. However, different videos might require different threshold values. To limit this problem, techniques have been proposed that link the threshold value to the mean and standard deviation of the distribution of frame differences across the whole video. Other methods use running averages in windows of multiple frames to detect shot-cuts.

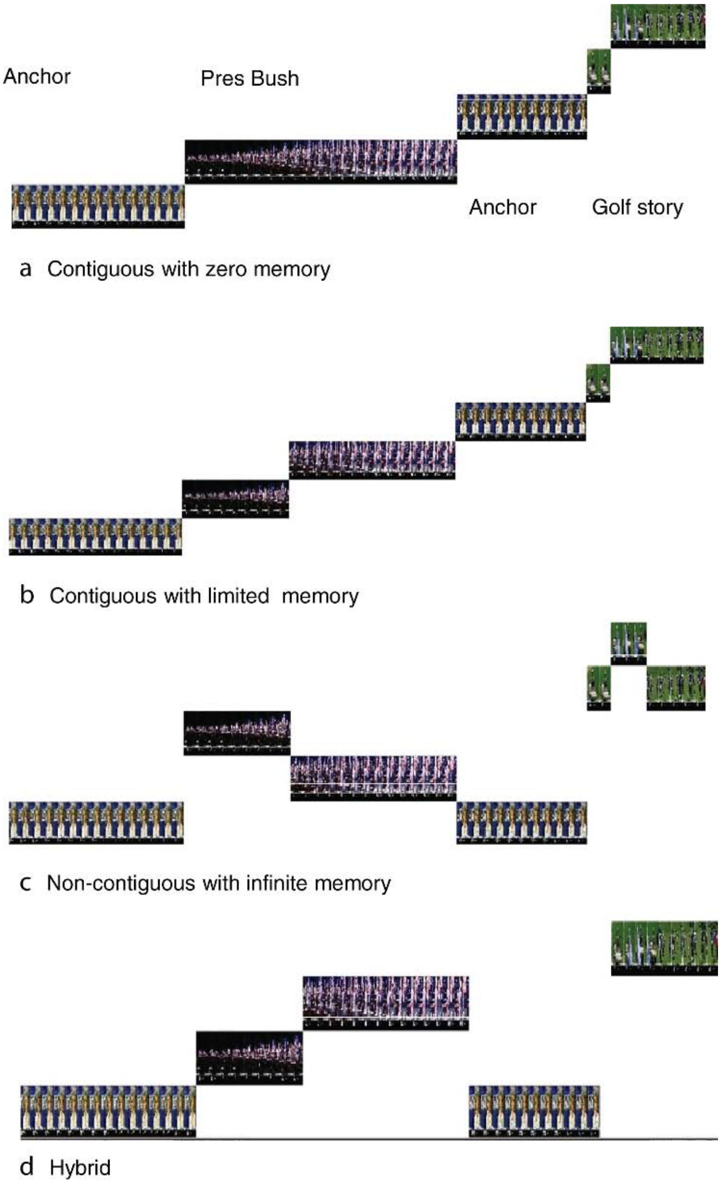
Scene Boundary Detection

The methods described above typically use the differences in contiguous frames to detect shot boundaries. In order to detect scene boundaries, groups of shots are determined that contain semantically related information such as the same physical location, or the same action. While a shot boundary is defined rather unambiguously, scene boundaries are much more difficult to capture. Different individuals tend to identify scenes in videos in different ways. This makes obtaining ground truth for the scenes a non-trivial task.

An overview and a benchmark of different methods for scene boundary detection are presented in [8]. Common techniques used for scene boundary detection use audio analysis and shot clustering [10].

One of the methods to automatically detect scene boundaries is by means of *superhistograms* [3]. Using this method, first the color histograms for individual frames are computed, and then they are merged into a single cumulative histogram called a *family histogram* based on a comparison measure. As new frames are added, the family histogram accumulates the new colors from the respective frames. However, if the histogram of a new frame is different from the previously constructed family histograms, a new family is formed. In the end, there are a few families of histograms to represent the entire video. This set of families is ordered with respect to the length of the temporal segment of video that they represent. The ordered set of family histograms is called superhistogram.

When computing the family histograms there are several dimensions to think about when making comparisons between frames based on (i) the amount of memory (time) (ii) contiguity of compared families, (iii) the representative structure for a family. Memory can be thought as one-dimension, that is, how much history of the computed family histograms is kept: a process that has "zero memory" and compares only the current frame with a previous frame histogram vs. a process with infinite memory that compares the current frame with all the previous family histograms. As shown in Fig. 2, the results can be quite different. Another dimension is time, which means whether contiguous or noncontiguous comparisons are made. The third dimension is the determination of a representative histogram for the family. This can be the average histogram or the histogram of a selected frame (e.g., the last frame). There are different


Anchor
Pres Bush
Anchor
Golf story
a Contiguous with zero memory
b Contiguous with limited memory
c Non-contiguous with infinite memory
d Hybrid

Video Segmentation. Figure 2. Merging strategies for creating family histograms.

merging strategies [15]: contiguous vs. noncontiguous comparison and comparison with a short term and longer term memory:

1. Contiguous with zero memory: A new frame histogram is compared with the previous frame histogram. This strategy allows to keep a pan sequence in a single family as the addition of new colors happens slowly and the difference between consecutive frames (in case the last frame is compared) is very small.
2. Contiguous with limited memory: Contiguous families are produced by comparing the new frame histogram only with the previous family histogram. For commercial detection applications such strategy needs to be used because, otherwise, many families would be created during the commercials when new colors are introduced with each shot. This enables the differentiation of commercials from regular programs where the families are usually longer.
3. Noncontiguous with infinite memory with family histogram comparison: This is useful to capture the global colors of a complete program. For applications for characterization of videos for clustering and retrieval, this strategy needs to be used as it captures all the colors present in the video in the least number of clusters possible. A new frame histogram is compared with all previous family within the same video.
4. Hybrid: First a new frame histogram is compared using (3) above and then the generated family histograms are merged using (1) or (2). This enables the combination of global information that is preserved when using the (3) strategy to the local information that is achieved by combining the families using strategies (1) or (2). In this case, the family histograms that are produced by (3) effectively become the frame histograms. These are now merged with each other to generate a hierarchical picture of the video. This enables to cluster colors that are repeated at regular intervals throughout the program while keeping pans etc. together in a single family.

An example of the results obtained using these four strategies is shown in Fig. 2 for a portion of a news video program.

Key Applications

Video segmentation is usually the first step in content-based automatic video indexing and retrieval. It is also widely used for video browsing and automatic video summarization. Additionally, another field of application is the object-based video compression. To correctly detect and track objects in video, a temporal segmentation step is in general needed.

Cross-references

► Video Representation
► Video Shot Detection
► Video Summarization

Recommended Reading

1. Arman F., Hsu A., and Chiu M-Y. Image processing on encoded video sequences. Multimed. Syst., 1(5):211–219, 1994.
2. Boreczky J.S. and Rowe L.A. Comparison of video shot boundary detection techniques. In Proc. SPIE 1996 Int. Symp. on Electrical Image Science and Technology. Storage and Retrieval for Image and Video Databases IV, vol. 2670, 1996, pp. 170–179.
3. Dimitrova N., Martino J., Agnihotri L., and Elenbaas H. Superhistograms for video representation. In Proc. Int. Conf. Image Processing, 1999, pp. 314–318.
4. Dimitrova N., Agnihotri L., and Jainschi R. Temporal video boundaries. In Video Mining, A. Rosenfeld, D. Doermann, and D. Dementhon (eds.). Kluwer, Boston, 2003, pp. 61–90.
5. Hampapur A., Jain R., and Weymouth T., Digital Video Segmentation. In Proc. 2nd ACM Int. Conf. on Multimedia, 1994, pp. 357–364.
6. Nagasaka A. and Tanaka Y. Automatic video indexing and full-video search for object appearances. In Visual Database Systems II, E. Knuth, L. Wegner (eds.). Elsevier, Amsterdam, The Netherlands, 1992, pp. 113–127.
7. Swanberg D., Shu C.F., and Jain R. Knowledge guided parsing and retrieval in video databases. In Proc. SPIE Storage and Retrieval for Image and Video Databases, vol. 1908, 1993, pp. 13–24.
8. Vendrig J. and Worring M. Systematic evaluation of logical story unit segmentation. IEEE Trans. Multimed., 4:492–499, 2002.
9. Yeo B. and Liu B. A unified approach to temporal segmentation of motion JPEG and MPEG compressed video. Multimed. Tools Appl., 1(1):81–88, 1995.
10. Yeung M. and Yeo B.-L. Video visualization for compact presentation and fast browsing of pictorial content. IEEE Trans. Circuits Syst. Video Technol., 7(5):771–785, 1997.
11. Zabih R., Miller J., and Mai K. A feature-based algorithm for detecting and classifying scene breaks. In Proc. 3rd ACM Int. Conf. on Multimedia, 1993, pp. 189–200.
12. Zhang H.J., Chien Y.L., and Smoliar S.W. Video parsing and browsing using compressed data. Multimed. Tools Appl., 1(1):89–111, 1995.

Video Sequence Indexing

Heng Tao Shen
The University of Queensland, Brisbane, QLD,
Australia

Synonyms

Video retrieval; Video search; Video indexing

Definition

A video is usually defined as a sequence of high-dimensional feature vectors. Video sequence indexing consists of describing the content of video sequences from a video database to allow effective and efficient search and retrieval. Given a query video sequence, video sequence indexing aims to find its similar video sequences from a video database quickly. Typically, it includes the following major components: effective summarization of the high-dimensional sequence, effective access method for indexing the obtained summarization, and efficient query processing method.

Historical Background

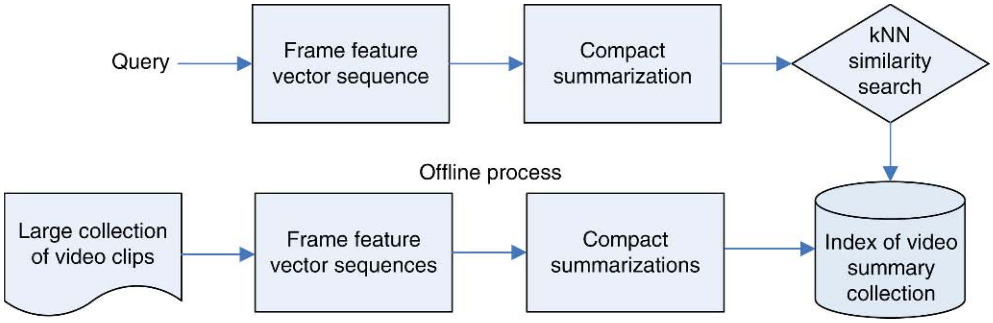
Video feature extraction and content analysis have been studied for several decades since the emergence of video data. Recently, video sequence indexing has attracted plenty of attention because of the huge amount of video data. With ever more heavy usage of video devices and advances in video processing technologies, the amount of video data has grown rapidly and enormously for various usages, such as filming, advertising, news video broadcasting, personal video archive, medical video data, and so on. Based on the statistics from Berkeley's "How Much Information" project 2003, 5,660 motion pictures are produced every year, amounting to almost 6,078 h. 21,264 television stations broadcast for about 16 h/day, producing 31 million hours of original programming annually, and huge amount of personal and organizational video data stocks are accumulating. Interestingly, the popularity of WWW enables enormous video data to be published and shared. The mainstream media is moving to the web. The media content online and video streaming is growing aggressively. While the pool of video data is super large, such as millions of videos currently available in web, video search engines provide users convenient ways for finding videos of their interests. Consequently, video sequence indexing has become a key part of the future of digital media [1].

The goal of video sequence indexing is to quickly find the similar videos to a query, and a video is usually represented by a sequence of high-dimensional frame feature vectors at a frame rate of 25–30/s. Due to the high complexity of video data, early work mainly focused on reducing the temporal redundancy of video data by identifying the shot which is the basic unit in video data. Shot detection is a fundamental task in video content manipulation. Representing the video by a small number of shots can reduce the data complexity in temporal dimension significantly. One central task in annual TRECVID evaluation is the shot boundary detection [2]. Many shot detection methods have been proposed and the field has matured [2,3]. Beside shot representation, recent work also considered the closeness of frames within a video across different shots and summarized similar frames into a compact representation [4,5]. Typically, temporal information is ignored for such aggressive summarizations. As the amount of video data increases explosively, there is also a trend to investigate various high-dimensional indexing methods to effectively manage and organize the video summaries for fast retrieval [5,6].

Foundations

Figure 1 shows the generic architecture for video sequence indexing. Given a collection of videos, their visual frame features, such as colour, tensity, and texture, is first extracted. Semantic features like objects and motions can also be further extracted from frame features. The sequence of frame features for each video is further summarized into compact representations which are indexed for fast retrieval. Given a query video, its features are first extracted and used to generate compact summarization. A similarity search is then performed on the indexed summary collection. The video similarity is approximated based on their compact representations.

To achieve efficient retrieval, most existing works emphasize on the summarization step to reduce the video data complexity and can be categorized into content-based and semantic-based approach. Content-based approach deals with frame visual features. In literature, two types of summarization techniques have been proposed: summarize the sequence as a statistical distribution and summarize the sequence into fewer representatives. The first type typically assumes that the frames are distributed in a model like Gaussian, or mixture of Gaussian [7]. In the second type, keyframe is often used [8]. Video signature introduces a


Query
Frame feature vector sequence
Compact summarization
kNN similarity search
Offline process
Large collection of video clips
Frame feature vector sequences
Compact summarizations
Index of video summary collection

Video Sequence Indexing. Figure 1. A generic architecture for video sequence indexing.

randomized summarization method which randomly selects a number of seed frames and assigns a small collection of closest frames to each seed [4]. However, the selection of seeds may sample non-similar frames from two almost-identical sequences. Video Triplet (ViTri) models a cluster of similar frames as a tightly bounded hypersphere described by its *position*, *radius*, and *density* [5]. The ViTri similarity is measured by the volume of intersection between two hyper-spheres multiplying the minimal density, i.e., the estimated number of similar frames shared by two clusters. The total number of similar frames is then estimated to derive the overall similarity between two video sequences.

The semantic-based approach typically handles the motion trajectory of objects in a video sequence. Motion trajectories suggest the temporal movement of objects. Moving objects are first detected and segmented, and the motion trajectories of objects are then detected by analyzing inter-frame correspondences. Videos are represented based on these raw motion trajectories within a shot [8,9]. However, this approach is limited by the accuracy of object detection and only applicable so applications where the objects can be clearly identified.

In content-based image retrieval, the similarity/ distance of two images is typically computed by the Euclidean distance. When extending the distance function to video sequences, many proposals have been proposed. One widely used video similarity measure is the approximate percentage of similar frames shared by two sequences [4,13]. Some studies also take temporal information into consideration. In [5], a time warping based approach is proposed to deal with video temporal variations, including local shifting. Hausdorff distance is used to measure the maximal dissimilarity between two shots [2]. The Earth Mover's Distance was introduced in computer vision to better approach human perceptual similarities. It models similarity as the amount of changes necessary to transform one image feature into another one. A tight and computationally simple approximation of the Earth Mover's Distance was proposed for matching video sequences [1]. Other measures widely used in time series, such as Longest Common Subsequence [15], Edit Distance and its extensions [3], can also be extended for comparing video sequences. However, all these measures need to compare most, if not all, frames pairwise. The full similarity computation requires storage of the entire sequence and time complexity is typically quadratic.

To index low-dimensional video trajectories, trajectories are typically first split into segments which are typically represented by Minimum Bounding Rectangles (MBR) [11]. The obtained MBRs can then be indexed by some tree structures like Trajectory Bundle Tree (TB-tree) [10]. Efficient query processing typically deploys the GEMINI-like framework which pruning the search space by the established lower bound lemma. When temporal information is not considered, existing high-dimensional indexing structures and their query processing methods can be directly applied [4,13].

Key Applications

There are many applications for video sequence indexing. Typically applications include video search, video monitoring, video copy detection, video annotation, video digital library, etc.

Future Directions

The indexing structures that are used to manage high-dimensional video sequences have not been well explored in the database community. There is the also an urgent need for online monitoring, indexing and searching, as the media content online and video streaming is growing aggressively.

Experimental Results

In general, for every presented method, there is an experimental study in the corresponding reference.

Data Sets

TRECVID provides reasonably large video data sets for testing and experimental purposes [14].

URL to Code

Published code for video sequence indexing is rather limited in the database society.

Cross-references

- ▶ Indexing
- ▶ Similarity Measure
- ▶ Time Series
- ▶ Video Summarization

Recommended Reading

1. Assent I., Wenning A., and Seidl T. Approximation Techniques for Indexing the Earth Mover's Distance in Multimedia Databases. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 11.
2. Chang H., Sull S., and Lee S. Efficient video indexing scheme for content-based retrieval. IEEE Trans. Circuits Syst. Video Technol., 1999.
3. Chen L., Özsu M.T., and Oria V. Robust and Fast Similarity Search for Moving Object Trajectories. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 491–502.
4. Cheung S. and Zakhor A. Efficient video similarity measurement with video signature. IEEE Trans. Circuits Syst. Video Technology, 2003.
5. Chiu C.-Y., Li C.-H., Wang H.-A. Chen C.-S., and Chien L.-F. A time warping based approach for video copy detection. In Proc. 18th Int. Conf. on Pattern Recognition, 2006, pp. 228–231.
6. Cotsaces C., Nikolaidis N., and Pitas I. Video shot detection and condensed representation: a review. IEEE Signal Process. Magaz., 23(2):28–37, 2006.
7. Iyengar G. and Lippman A. Distributional clustering for efficient content-based retrieval of images and video. In Proc. Int. Conf. Image Processing, 2000, pp. 81–84.
8. Keogh E.J., Palpanas T., Zordan V.B., Gunopulos D., and Cardle M. Indexing large human-motion databases. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 780–791.
9. Lee J., Oh J.-H., and Hwang S. STRG-index: spatio-temporal region graph indexing for large video databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 718–729.
10. Pfoser D., Jensen C.S., and Theodoridis Y. Novel Approaches in Query Processing for Moving Object Trajectories. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 395–406.
11. Rasetic S., Sander J., Elding J., and Nascimento M.A. A trajectory splitting model for efficient spatio-temporal indexing. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 934–945.
12. Sarukkai R.R. Video search: opportunities and challenges. Keynote Speech at ACM Multimedia Information Retrieval Workshop for ACM Multimedia Conference, 2005.
13. Shen H.T., Ooi B.C., Zhou X., and Huang Z. Towards effective indexing for very large video sequence database. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 730–741.
14. TRECVID. http://www-nlpir.nist.gov/projects/trecvid/. 2007.
15. Vlachos M., Hadjieleftheriou M., Gunopulos D., and Keogh E. Indexing multi-dimensional time-series with support for multiple distance measures. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 216–225.

Video Shot Detection

Chong-Wah Ngo
City University of Hong Kong, Hong Kong, China

Synonyms

Shot boundary detection; Camera break detection

Definition

Shot is a contiguous sequence of video frames with smooth and continuous camera motion. Shot can thus be viewed as an uninterrupted video frame sequence of space, time, and graphical configuration. The boundary between two shots is called a camera break (or video edit). Based on the transitional properties of video edits, there are two major types of camera breaks: *cut* and *gradual transition*. A camera cut is an instantaneous change from one shot to another. The gradual transition can further be categorized as *wipe* and *dissolve*. A wipe is the moving transition of a frame (or a pattern) across the screen that enables one shot to gradually replace another. A dissolve superimposes two shots, where one shot gradually appears while the other fades out slowly. Fade-in and fade-out can be considered as the special cases of dissolve, by replacing one of the shots as a constant image sequence (e.g., black image sequence). In contrast to cut, wipe and dissolve involve gradual transitions with no drastic changes between two consecutive frames, and hence, are relatively difficult to identify by computer.

Historical Background

Shot is regarded as the elementary unit of video structure, often serving as the basic component for video indexing, browsing and retrieval. Shot detection has been extensively studied since the emergence of the topic: content-based video retrieval in mid 1990. The early researches are unified under the theme of *video structuring and representation*, which targets for decomposing videos into smaller unit facilitating content analysis and understanding [15]. The scope of studies includes the detection of cut and gradual transition [15], detection efficiency [11], feasibility of thresholding [10], compressed versus uncompressed domain [11], and framework of detection [13].

Shot detection is mostly based on the analysis of visual information, or more precisely the changes of visual signal over time. For instance, an abrupt change of signal between frames can imply a camera cut. To date, there exist numerous approaches for shot detection which can be broadly classified according to the visual features being used and the type of camera breaks being analyzed. Based on feature, the existing approaches can be categorized as pixel-based, block-based, color-based, texture-based [14], motion-based [2], and slice-based [8]. These approaches can operate in either a compressed or uncompressed domain with slight modification. Uncompressed domain detection refers to the pixel-to-pixel processing of video frames to extract the desired features for analysis. Compressed domain detection refers to the direct processing of compressed information without explicitly decompressing video signals. The compressed features popularly used are DCT coefficients and MPEG motion vectors. Since the amount of information to be processed is relatively small, compressed domain detection has the advantage of speed efficiency. For instance, in [15], DC images which are the thumbnail version of original video frames are extracted directly from MPEG videos for rapid shot detection. In [15], motion vectors of P- and B- frames are extracted directly from MPEG videos for analysis.

Compared to cut, gradual transition (GT) is difficult to detect due to the absence of sharp signal changes. While cut normally involves only two frames, GT involves a sequence of frames ranging from several to even hundreds of frames. The first algorithm for GT detection is twin-comparison [15], which utilizes two thresholds to locate GTs. Another well-known algorithm is based on the video production model proposed in [1], which model the distribution of GT signals for detection by reversing the linear equation used to synthesized GTs in video production. GT detection has also recently been cast as a pattern analysis and learning problem. In [5–7,13], various classifiers are learned to recognize different types of GTs.

TRECVID has been the rendezvous where researchers meet and compete their shot detection algorithms since early 2000 [9]. The series of activities, over the past few years, have led to several sophisticated frameworks that perform reliably and excellently for the detection of camera cuts and GTs [9,13]. Based on the reported results, the best performing systems can always achieve more than 80% of recall and precision for cut and GT detection. Most systems also run faster than real-time.

Foundations

Cut

Camera cut is the most common type of shot boundary. In contrast to gradual transitions, most cuts involve no transitional frame. Thus, direct comparison between two frames is usually sufficient to attain satisfactory performance. A straightforward approach is to compute the mean absolute change of intensity between two frames at time t and t + 1. This simple approach, however, is sensitive to object and camera motion. An alternative version is to compute the energy changes of DC frames, which is the smaller and smoothed version of the original frames, extracted from MPEG videos to reduce the sensitivity [11]. In [8], the sequence of DC frames is further sub-sampled along the time dimension to form spatial-temporal slices. This approach detects cuts by analyzing the changes of temporal patterns in slices.

The most popular approach perhaps is via the comparison of intensity or color histograms between two contiguous frames. Color histogram quantizes the color space and captures the ratio of color components in a frame as a distribution. As a result, the histogram-based approaches are more invariant to local and global motion changes. To also capture spatial information in a color histogram, a more popular version is the block-based histogram: divide frames into blocks and compute a histogram for each block.

Other visual features being considered for cut detection include edge information, which captures the structural discontinuity of frames. In [14], edge change ratio is used to model the discontinuity by computing the number of edge pixels entering and existing between two frames. In addition, motion discontinuity is also studied in [2]. The cut detection is conducted by computing the global dominant motion of frames and accounting the number of points that do not fit the computed motion model. This approach can also be used for detecting gradual transitions.

Dissolve

Some existing works on the detection of dissolve transitions are based on the video production model [1]:

$$f(x,y,t) = (1-\alpha(x,y,t))g(x,y,t) + \alpha(x,y,t)h(x,y,t) \qquad (1)$$

where f is a dissolved frame superimposed by two frames g and h at time t. Typically, g and h are frames from two different shots. The transition function α characterizes, either linearly or non-linearly, how f is dissolved over time as a result of mixing g and h. Usually $0<\alpha(x, y, t) < 1$ with the condition $\alpha(x, y, t) \leq \alpha(x, y, t + 1)$. Since (1) is irreversible, apparently, detecting and classifying dissolves is a difficult task. To simplify the problem of detection, various assumptions have been made on (1). These assumptions can lead to plateau a effect [11] and parabolic curve of variance [1]. Take $\alpha(x, y, t) = \alpha(t)$, $g(x, y, t) = g(x, y)$ and $h(x, y, t) = h(x, y)$, (1) becomes

$$f(x,y,t) = (1-\alpha(t))g(x,y) + \alpha(t)h(x,y) \qquad (2)$$

In other words, f is a dissolved sequence of two static shots g and h in $t = [t_1, t_2]$. Let $\mathcal{F}(t) = f(x,y,t)$, by taking the frame difference, the following can be derived:

$$\frac{\mathcal{F}(t)-\mathcal{F}(t+k)}{\mathcal{F}(t-k)-\mathcal{F}(t)} = \beta(t,k) \qquad (3)$$

where $\beta(t,k) = \frac{\alpha(t+k)-\alpha(t)}{\alpha(t)-\alpha(t-k)} > 1$ and $t = [t_1 + k, t_2 - k]$. If $k > t_2 - t_1 + 1$, plateau effect will be exhibited and this effect can be exploited effectively for dissolve detection [11].

Equation(2) can be further simplified by assuming $\alpha(t)$ as a linear function, $\alpha(t) = \frac{t-t_1}{t_2-t_1}$ for instance. This terms leads to a formula in terms of variance:

$$\sigma_f(t) = (\sigma_g + \sigma_h)\alpha^2(t) - 2\sigma_g\alpha(t) + \sigma_g \qquad (4)$$

where $\sigma_f(t)$, σ_g and σ_h are the variances of, $g(x, y)$ and $h(x, y)$. Since $\sigma_f(t)$ is a concave upward parabolic curve, dissolves can be detected simply by locating parabolic curves [1,8]. The limitations of (3) and (4) are mainly due to the linearity assumption of α(t) and the static sequence assumption of shots g and h. As a result, most detectors are generally very sensitive to noise, camera and object motions.

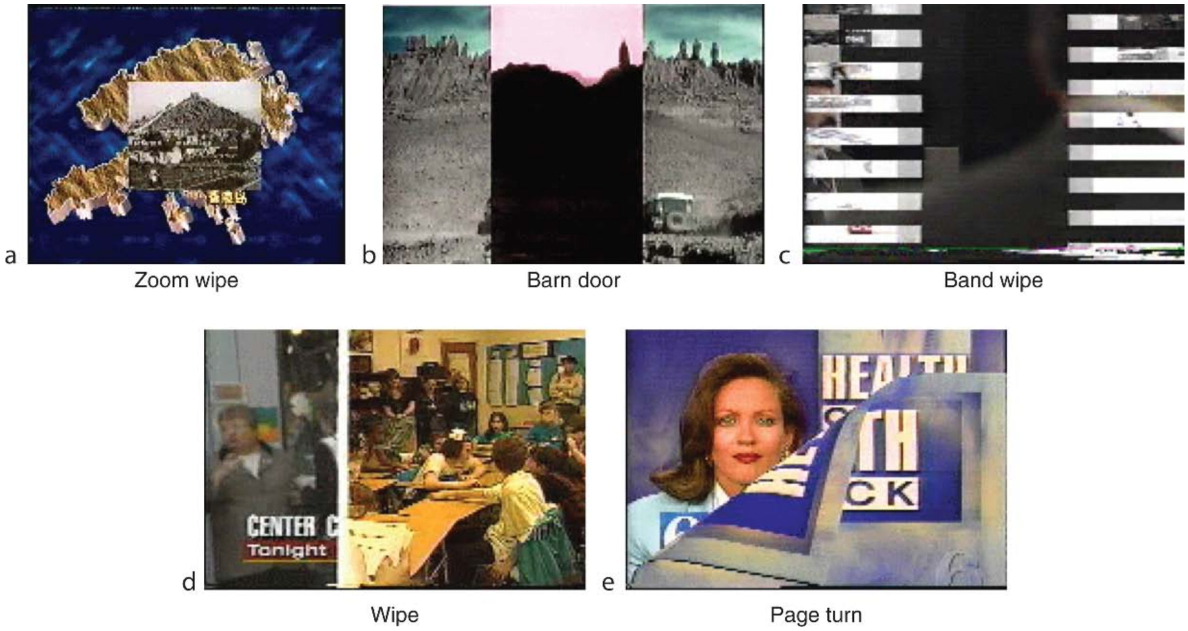
Recently, dissolve detection is also considered as a pattern classification problem, and various machine learning algorithms are brought in for this task [3]. For instance, in [6], multiple independent FSM (finite state machine) based detectors are used for detecting different types of shot boundaries. Together with classifiers, multiresolution analysis is also adopted by [5,7,13]. In [7], dissolve detection is reduced to the cut detection problem by projecting the spatio-temporal slices to lower temporal resolution scale. The detected cuts at lower resolution are further classified in the higher resolution space by machine learning. In [13], multi-resolution features are extracted for active learning of SVM (support vector machines) to detect dissolves.

Wipe

Wipe is also a common shot boundary transition frequently found, for example, in sport and news videos. There are lots of fancy wipe transitions being used by video editors today. Figure 1 shows a few examples of wipe. As each transition exhibits its own unique transition patterns, wipe patterns, compared to dissolves, are relatively hard for machines to learn. Early works are mostly based on frame differencing and edge detection to trace the moving boundary lines created by wipes. For instance, the works in [8] detect the edge patterns formed by wipes in spatial-temporal slices. The recent work in [4] analyzes the independence and completeness properties of wipe transitions. A cost function utilizing the MPEG motion vectors is derived to model these two properties for reliable wipe detection.

General Challenges

As a fundamental building block to support other video applications, shot detection has been extensively studied in the literature. The challenges that are not completely solved for cut detection include abrupt illumination change (e.g., flashlight) and large object/camera movement, which could trigger abrupt changes of video signal and generate false alarms. For gradual transition (GT), on the other hand, the



Video Shot Detection. Figure 1. Various wipe transition.

major difficulties are the slow evolution of signal that is hard to observe. Furthermore, the temporal duration can vary greatly while the patterns could appear similar to slow camera/object movement, making GT detection a very challenging problem. For both cut and GT detection, a fundamental issue is the thresholding heuristic – mostly relying on empirical setting. Over the years, the thresholding techniques have also been evolved from simple global threshold setting, to the adaptive thresholding [10] and learning-based heuristics [3,5,13].

Key Applications

Shot detection is a fundamental task of video content manipulation. Shots are often served as the basic unit for video browsing, indexing and search. An interesting application of shot detection is that, by knowing the type, speed and frequency of shot transitions, the pace, mood, style and highlight of the underlying video events can be modeled for affective computing.

Experimental Results

The evaluations commonly used are recall and precision, defined as:

$$\text{Precision} = \frac{\text{Number of Transitions Correctly Detected}}{\text{Number of Transitions Detected}} \qquad (5)$$

$$\text{Recall} = \frac{\text{Number of Transitions Correctly Detected}}{\text{Number of Transitionsarray}} \qquad (6)$$

Precision and recall are in the interval of [0,1]. Low precision hints the frequent occurrence of false positives, while low recall indicates the frequent occurrence of false negatives. As gradual transition involves multiple frames, frame precision and recall are also used in order to evaluate the accurate localization of transitions,

$$\text{Frame Precision} = \frac{\begin{array}{c}\text{Number of Frames Correctly Located}\\ \text{in the Detected Transitions}\end{array}}{\begin{array}{c}\text{Number of Frames Located in}\\ \text{the Detected Transitions}\end{array}} \qquad (7)$$

$$\text{Frame Recall} = \frac{\begin{array}{c}\text{Number of Frames Correctly Located}\\ \text{in the Detected Transitions}\end{array}}{\text{Number of Frames in the Transitions}} \qquad (8)$$

Data Sets

The popular datasets always experimented are TRECVID benchmarks [9].

Cross-references

▶ Scene Boundary Detection
▶ Sub-Shot Detection

Recommended Reading

1. Alattar A.M. Detecting and compressing dissolve regions in video sequences with a DVI multimedia image compression algorithm. Int. Symp. Circuits Syst., 1:13–16, 1993.
2. Bouthemy P., Gelgon M., and Ganansia F. A unified approach to shot change detection and camera motion characterization, IEEE Trans. Circuits Syst. Video Tech., 9(7):1030–1044, 1999.
3. Hanjalic A. Shot boundary detection: unraveled and resolved. IEEE Trans. Circuits Syst. Video Tech., 12(2):90–105, February 2002.
4. Li S. and Lee M.-C. Effective detection of various wipe transitions. IEEE Trans. Circuits Syst. Video Tech., 17(6), June 2007.
5. Lienhart R. Reliable dissolve detection. In Proc. SPIE Storage Retrieval Media Database, 2001.
6. Liu Z., Gibbon D., Zavesky E., Shahrary B., and Haffner P. AT&T Research at TRECVID 2006. In Proc. TREC Video Retrieval Evaluation, 2006.
7. Ngo C.W. A robust dissolve detector by support vector machine. In Proc. 11th ACM Int. Conf. on Multimedia, 2003.
8. Ngo C.W., Pong T.C., and Chin R.T. Video partitioning by temporal slice coherency. IEEE Trans. Circuits Syst. Video Tech., 11(8):941–953, August 2001.
9. TREC-Video, http://www-nlpir.nist.gov/projects/trecvid/.
10. Vasconcelos N. and Lippman A. Statistical models of video structure for content analysis and characterization. IEEE Trans. Image Process., 9(1):3–19, January 2000.
11. Yeo B.L. and Liu B. Rapid scene analysis on compressed video. IEEE Trans. Circuits Syst. Video Tech., 5(6):533–544, 1995.
12. Yu H. and Wolf W. A multi-resolution video segmentation scheme for wipe transition identification. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, 1998.
13. Yuan J., Wang H., Xiao L., Zheng W., Li J., Lin F., and Zhang B. A formal study of shot boundary detection. IEEE Trans. Circuits Syst. Video Tech., 17(2), 2007.
14. Zabih R., Miller J. Mai K., and Zabih R. et al., A feature-based algorithm for detecting and classifying scene break. In Proc. 3rd ACM Int. Conf. on Multimedia, 1995.
15. Zhang H.J., Kankanhalli A., Smoliar S., and Zhang H.J. et al. Automatic partitioning of full-motion video. ACM Multimedia Syst., 1(1):10–28, 1993.

Video Shot-Cut Detection

▶ Video Segmentation

Video Skimming

▶ Video Summarization

Video Structure Analysis

▶ Video Content Structure

Video Structuring

▶ Video Content Structure

Video Summarization

Chong-Wah Ngo, Feng Wang
City University of Hong Kong, Hong Kong, China

Synonyms

Video abstraction; Video skimming

Definition

Video summarization is to generate a short summary of the content of a longer video document by selecting and presenting the most informative or interesting materials for potential users. The output summary is usually composed of a set of keyframes or video clips extracted from the original video with some editing process. The aim of video summarization is to speed up browsing of a large collection of video data, and achieve efficient access and representation of the video content. By watching the summary, users can make quick decisions on the usefulness of the video. Dependent on applications and target users, the evaluation of summary often involves usability studies to measure the content informativeness and quality of a summary.

Historical Background

Due to the advance of web technologies and the popularity of video capture devices in the past few decades, the amount of video data is dramatically increasing. This creates a strong demand for efficient tools to browse, access, and manipulate large video collections. The most straightforward and simplest way for quick browsing of video content is to speed up the play of video by frame dropping or re-encoding. As studied in [6], the entire video could be watched in a shorter amount of time by fast playback with almost no pitch distortion using the time compression technology. This kind of approach is easy to implement and preserves most information while shortening the time of watching videos. However, without comprehensive understanding of the original video content, the selection of video materials is impossible, and thus almost all redundant information still has to be watched. This limits further shortening of summary duration and its enjoyability. Meanwhile, the ratio of time compression is also limited.

Since the 1990s, *video content analysis* has attracted a lot of research attention. Different approaches have been proposed for automatic structuring and understanding of video content. This enables video summarization by measuring and selecting the most informative materials based on more comprehension of the video content. Video summarization can be seen as a general term of video abstraction and video skimming. In some works, they are distinguished by little difference in the composition of the output. For instance, video skims are mainly composed of the short clips skimmed from the original video. Video summary and abstraction could be a set of keyframes, clips or even other media (e.g., texts) created to describe the video content. Nowadays, video summarization has become the key tool to facilitate efficient browsing and indexing of large video collections.

Foundations

In the past two decades, a lot of research efforts have been devoted to video summarization. A systematic review of these works can be found in [10]. According to the definition, a summary should be as short or concise as possible to support efficient video browsing. On the other hand, it should be informative so that as much interesting and useful materials as possible will be included. Thus, the key of video summarization is how to measure the importance of different video clips and select the most useful ones so as to achieve both elegancy and informativeness. For this purpose, there are mainly two issues concerned: *video content analysis* and summary generation by selecting appropriate materials. *Video content analysis* enables the understanding and comprehension of the original video content. Different features are then extracted to structure and describe the video. Based on *video content analysis*, a shorter summary is generated by selecting and presenting the most important materials to users. Meanwhile, to cope with the characteristics of different video domains, many algorithms are proposed to achieve better results for the summarization of specific video domains by employing domain-specific knowledge.

Features for Content Analysis

Along with the advances in *video content analysis*, more and more features are employed in video summarization. According to the features adopted for video content analysis, the algorithms can be categorized as shot detection based [15], motion based [3], audio based [9], and multi-modality based [4,11]. In recent years, most algorithms tend to integrate multiple features to better describe the video content. Besides the visual-audio features, especially with the dramatic increasing of web videos, the metadata, context, social, and log information are being explored for video summarization of video corpus, e.g., the video clips that are watched by the most users may be more important and interesting. What features to use is usually dependent on the requirements of specific applications.

Importance Measure for Summary Generation

How to decide the importance and usefulness of video segments are essential for summary generation. During the past two decades, different approaches have been proposed. The first kind of approach is based on *video shot detection.* Shot detection is one of the fundamental techniques for video structuring. In each segmented shot, one or several keyframes are selected and presented for users to effiently browse and index the video content in order to find the interesting video segments for further exploration. In [15], several keyframes are extracted by detecting visual changes using color histogram inside each shot. In [2], to eliminate the possible redundancy among extracted keyframes, fuzzy clustering and data pruning methods are employed based on color information to obtain a set of non-redundant keyframes as the summary of the video. These approaches just employ limited low-level visual features. For the selection of keyframes or video segments, the main idea is to reduce the redundancy and maximize the visual difference in the generated summary to show more information measure by low-level features.

In the second kind of approach, importance measure is based on some pre-defined rules. In [8], a set of rules are defined to select the most useful frames or video segments. For instance, images between similar scenes that are less than 5 seconds apart, are used for summarization; short successive shots are considered to introduce a more important topic. With prioritized video frame from each shot, another set of high order rules are then used to compose the selected clips for summary generation. For instance, the duration of each clip is at least 60 frames based on empirical and user studies of visual comprehension in short video sequences. Similar rules can also be used in query-based or domain-specific video summarization. For instance, given a soccer video, the users may be just interested in the shots of goal. In this case, to make a summary, the events of goal should be first detected and then shown to the users. In this kind rule-based approaches, the importance of video segment is manually assigned by the predefined rules. This may be useful for some specific video domains and applications. However, the rules are usually subjective. Considering the variations of video content, to make desirable summaries, more and more detailed rules are required to cope with different scenarios. This becomes almost impossible in general videos.

Today many approaches try to numerically measure the importance of video segments and model video summarization in a computational way. In [4], a user attention model utilizing a set of audio-visual features is proposed. Attention is a neurobiological conception. It implies the concentration of mental powers upon an object by close or careful observing or listening, which is the ability or power to concentrate mentally. When watching a video, human attention is always attracted by different information elements such as face, text, object and camera motion, and so on. In [4], attention models are first computed based on different visual/audio features. A user attention curve is then generated by linear combination fusion scheme. The crests of the curve are more likely to attract the viewers' attentions. Keyframes or video skims are extracted from these crests. Similarly, in [14], a perception curve that corresponds to human perception changes is constructed based on a number of visual features, including motion, contrast, special scenes, and statistical rhythm. The frames corresponding to the peak points of the perception curve are extracted for summarization. This kind of approach usually integrates multiple modalities from video documents and human perception cue to depict the curve of video importance numerically for summarization. However, there are still manually defined rules used such as the selection and the weights of different modalities.

Domain-Specific Video Summarization

According to the target video domains, video summarization can be categorized into general videos and domain-specific videos, such as music video, news video, sports video, and rushes video.

General video: The algorithms for general video summarization employ features available in all types of videos, and usually do not consider any domain-specific knowledge. In [5], a video is represented as a complete undirected graph and the normalized cut algorithm is employed to partition the graph into video clusters. The resulting clusters form a directed temporal graph and a shortest path algorithm is proposed to detect video scenes. The attention values are computed and attached to the scenes, clusters, shots and subshots. Thus, the temporal graph can describe the evolution and perceptual importance of a video. A video summary is generated from the temporal graph to emphasize both content balance and perceptual quality. The advantage of this kind of approach is that no prior knowledge is used and the algorithms can work for most videos. However, when watching videos of different domains, the users' attentions are quite different. By treating all videos in the same way, these algorithms usually cannot produce satisfactory results for videos of specific interest. Thus, a lot of works focus on specific video domains and make use of domain-specific knowledge in order to improve the summary qualities.

Music video: In music video, the music plays the dominant role. Thus, music video summarization is mainly based on music analysis. In [13], the chorus is detected and used as a thumbnail for music content. The analysis of visual content such as shot classification and text (lyrics) recognition are employed for music-visual-text alignment so as to make a meaningful and smooth music video summary.

Sports video: When watching sports videos, people are more interested in the exciting moments and great plays. Thus, the highlight detection and event classification are usually involved in order to make a good summary. In [1], a multi-level representation for sports video is proposed. Based on low-level features, different semantic shots (e.g., Audience, Player Close-up, and Player Following) are classified at mid-level. For different sports games, the corresponding interesting objects and events (such as the football and a goal event in a soccer video) are detected at high-level. The interesting events can then be selected for summary generation.

News video: In daily news videos, a topic is usually composed of a story chain during the topic evolution. The dependencies among related stories are important to summarize a news topic. A feasible solution for rapid browsing of news topics is by threading and autodocumenting news videos [12]. In [12], the duality between stories and textual-visual concepts is first exploited to cluster the stories of the same topic. The topic structure is then presented by exploring the textual-visual novelty and redundancy of stories. By pruning the peripheral and redundant news tories in the topic structure, a main thread is extracted for autodocumentary.

Rushes video: Rushes are the raw materials captured during video production. Being unedited, rushes contain a lot of redundant and junk information which is intertwined with useful stock footage. In [11], stock footage is first located by motion analysis, repetitive shot detection, and shot classification. The most representative video clips are then selected to compose a summary based on object and event understanding.

Key Applications

Video summarization can be used in many applications.

Multimedia Archives Indexing and Retrieval

Nowadays more and more video documents are being digitized and archived worldwide. Video summarization can be used to index and retrieve a large video collection, and thus facilitate efficient access of video content. For instance, the online summarization can support journalists when searching old video material, or when producing documentaries. Another example is the wide use of web videos, where the index could easily be realized by automatically generated video summaries.

Movie Marketing

In the broadcasting and filmmaking industries, a lot of raw footage (extra video, B-rolls footage) is used to generate the final products such as TV programs and movies. Twenty to forty times as much material may be shot as actually becomes part of the finished product. The "shoot-to-show" ratio, such as in BBC TV, ranges from 20 to 40. The producers see these large amount of raw footage as cheap gold mine. Video summarization can be used to find the stock footage from the heavily redundant materials for potential reuse.

Home Entertainment

For instance, one can easily have a brief overview of what happened in a television series that may have been missed.

Future Directions

Performance evaluation

Despite the advance achieved in video summarization, a fundamental problem, i.e., performance evaluation, is yet to be solved. Today, the evaluation is mainly carried out by subjective scoring based on the users' personal judgment or few predefined criterions. This limits effective comparison between different approaches.

Comprehensive understanding and modeling of video content

In the existing approaches, neither the rule-based nor computational models can fully grasp the content and the storyline of the video. Besides more powerful approaches for *video content analysis*, the modeling of multimedia information is also essential for video summarization.

From large-scale video summarization to search and browsing

The existing works mainly focus on the summarization of a single video document or a small video cluster, where the information inside a video or dependency between videos can be relatively easily modeled. With the dramatic increasing of video data (e.g., web videos), how to summarize a huge group of videos, for instance, by structuring the relationship among the diversity of videos is a big challenge. Besides video content dependency, context and social information are cues that could be explored for this type of summarization in large-scale. In addition, the exploitation of summarization for large-scale video search and browsing remains an area yet to be studied.

Experimental Results

In TRECVID Workshop 2007 on Rushes Summarization [7], there were 22 runs submitted for evaluation on the same benchmark. Different criteria such as IN (inclusion of groundtruth objects/events), EA (easy to understand), and RE (redundancy of the summary) were assessed. Runs generated by CityUHK, NII, and LIP6 were verified by [7] as significantly better than two baselines from CMU. The baselines were generated by evenly sampling frames and shot clustering, respectively.

Data Sets

Rushes Videos by TRECVID Workshop.

Since 2004, the annual TRECVID Workshop provides a benchmark for rushes video exploitation and summarization. The dataset is composed of 100 h's raw video materials produced during news video and film production.

Cross-references

▶ Video Content Analysis

References

1. Duan L.Y., Xu M., Chua T.S., Tian Q., and Xu C. A Mid-Level Representation Framework for Semantic Sports Video Analysis. In Proc. 11th ACM Int. Conf. on Multimedia, 2003.
2. Ferman A.M. and Tekalp A.M. Two-stage hierarchical video summary extraction to match low-level user browsing preerences. IEEE Trans. Multimedia, 5(2):244–256, 2003.
3. Liu T., Zhang H.J., and Qi F. A Novel Video Keyframe Extraction Algorithm based on Perceived Motion Energy Model. IEEE Trans. Circuits Syst. Video Tech., 13(10):1006–1013, 2003.
4. Ma Y.F., Lu L., Zhang H.J., and Li M. A user attention model for video summarization. In Proc. 10th ACM Int. Conf. on Multimedia, 2002.
5. Ngo C.W., Ma Y.F., and Zhang H.J. Video summarization and scene detection by graph modeling. IEEE Trans. Circuits Syst. Video Tech., 15(2):296–305, 2005.
6. Omoigui N., He L., Gupta A., Grudin J., and Sanocki E. Time-compression: system concerns, usage, and benefits. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1999.
7. Over P., Smeaton A.F., and Kelly P. The TRECVID 2007 BBC Rushes Summarization Evaluation Pilot. In TRECVID BBC Rushes Summarization Workshop in ACM Multimedia, 2007.
8. Smith M.A. and Kanade T. Video skimming and characterization through the combination of image and language understanding. In Proc. IEEE Int. Workshop on Content-Based Access of Image and Video Database, 1998.
9. Taskiran C.M., Pizlo Z., Amir A., Ponceleon D., and Delp E. Automated video program summarization using speech transcripts. IEEE Trans. Multimedia. 8(4):775–791, 2006.
10. Truong B.T. and Venkatesh S. Video abstraction: a systematic review and classification. ACM Trans. Multimedia Comput. Commun. Appl., 3(1), 2007.
11. Wang F. and Ngo C.W. Rushes video summarization by object and event understanding. In TRECVID Workshop on Rushes Summarization in ACM Multimedia Conference September 2007.
12. Wu X., Ngo C.W., and Li Q. Threading and autodocumenting in news videos. IEEE Signal Process. mag., 23(2):59–68, 2006.
13. Xu C., Shao X., Maddage N.C., and Kankanhalli M.S. Automatic music video summarization based on audio-visual-text analysis and alignment. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 361–368.
14. You J., Liu G., Sun L., and Li H. A multiple visual models based perceptive analysis framework for multilevel video summarization. IEEE Trans. Circuits Syst. Video Tech., 17(3), 2007.
15. Zhang H.J., Wu J., Zhong D., and Smoliar S.W. An integrated system for content-based video retrieval and browsing. Pattern Recogn., 30(4):643–658, 1997.

View Adaptation

Kenneth A. Ross
Columbia University, New York, NY, USA

Synonyms

Materialized view redefinition

Definition

Small changes to the definition of a materialized view are often needed in database systems. View adaptation is the process of keeping a materialized view up-to-date when the definition of the view is changed. View adaptation aims to leverage the previously materialized view to generate the new view, since the cost of rebuilding the materialized view from scratch may be expensive.

Key Points

View adaptation is related to the question of *answering queries using views*. The new view can be thought of as a query, with the old view available to help compute it. However, view adaptation also admits in-place changes that are not possible using a query-answering approach. For example, if a redefined view contains most but not all of the records from the original view, then view adaptation could be achieved by deleting the records that no longer qualify.

A variety of adaptation techniques are presented in [2,3], allowing changes to the SELECT, FROM, WHERE, GROUPBY and HAVING clauses. UNION and EXCEPT views are also considered. In some cases, multiple changes to a view definition can be handled in a single pass, without materializing intermediate results for each change. Experimental results show the value of adaptation, particularly the in-place methods, relative to recomputation. Extensions of view adaptation to more general contexts, such as distributed databases, have also been proposed [1,4].

Data warehouses and decision support systems often employ materialized views to speed up query processing. Applications in which a user can change queries dynamically and see the results fast, such as data visualization, data archeology, and dynamic query processing can also benefit from view adaptation.

Cross-references

▶ Answering Queries using Views
▶ Incremental View Maintenance
▶ Materialized Views

Recommended Reading

1. Bellahsene Z. View adaptation in the fragment-based approach. IEEE Trans. Knowl. Data Eng., 16(11):1441–1455, 2004.
2. Gupta A., Mumick I.S., Rao J., and Ross K.A. Adapting materialized views after redefinitions: techniques and a performance study. Inf. Syst., 26(5):323–362, 2001.
3. Gupta A., Mumick I.S., and Ross K.A. Adapting materialized views after redefinitions. In Proc. SIGMOD Conf. on Management of Data. San Jose, CA, 1995, pp. 211–222.
4. Mohania M.K. and Dong G. Algorithms for adapting materialized views in data warehouses. In Proc. Int. Symp. on Cooperative Database Systems for Advanced Applications, 1996, pp. 309–316.

View Definition

Yannis Kotidis
Athens University of Economics and Business,
Athens, Greece

Synonyms

View expression

Definition

The definition of a view consists of the name of the view and of a query, whose result is used to determine the content of the view.

Key Points

A view is a virtual relation. Its content depends on the evaluation of a query over a set of base tables or other views in the database. This query is part of the view definition and is, typically, recomputed every time the view is referenced. In some cases, for efficiency, the tuples of a view may be materialized as a separate table in the database.

In relational systems, a view is defined using the **create view** command:

create view < v > **as** < *query expression* >

The name of the view in the above example is < v > and the schema and content of the view are derived on-demand by the evaluation of < query expression >, which should be a legal expression supported by the database management system. Different vendor systems may impose some constraints on the form of < query expression >, for instance they may disallow references to temporary tables. When the data in the table(s) mentioned in < query expression > changes, the data in view < v > changes also.

Consider the following view definitions:

create view v1 as select Name, Age from Personnel where Department = "Sales"
create view v2 as select Wages.Name,Wages.Salary
from Personnel, Wages
where Personnel.Name = Wages.Name and Personnel.Department = "Sales"

The first expression defines a view termed *v1* that contains the name and age attributes from database table *Personnel*. The instance of view *v1* consists of the subset of personnel data restricted to those working at department *Sales*. View *v2* defined by the second expression, contains the result of the join between tables *Personnel* and *Wages* for employees working at the same department.

Cross-references

▶ View Maintenance Aspects
▶ View Unfolding
▶ Views

Recommended Reading

1. Adiba M.E. and Lindsay B.G. Database snapshots. In Proc. 6th Int. Conf. on Very Data Bases, 1980, pp. 86–91.

2. Dayal U. and Bernstein P. On the correct translation of update operations on relational views. ACM Trans. Database Syst., 8(3):381–416, 1982.
3. Gupta A., Jagadish H.V., and Mumick I.S. Data integration using self-maintainable views. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996, pp. 140–144.
4. Gupta H., Harinarayan V., Rajaraman A., and Jeffrey D.U. Index selection for OLAP. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 208–219.
5. Kotidis Y. and Roussopoulos N. DynaMat: a dynamic view management system for data warehouses. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 371–382.
6. Roussopoulos N. View indexing in relational databases. ACM Trans. Database Syst., 7(2):258–290, 1982.
7. Roussopoulos N. An incremental access method for viewCache: concept, algorithms, and cost analysis. ACM Trans. Database Syst., 16(3):535–563, 1991.

View Maintenance

Alexandros Labrinidis[1], Yannis Sismanis[2]
[1]Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA
[2]IBM Almaden Research Center, Almaden, CA, USA

Synonyms

View update; Materialized view maintenance

Definition

View maintenance typically refers to the updating of a *materialized view* (also known as a derived relation) to make it consistent with the base relations it is derived from. Such an update typically happens *immediately*, with the transaction that updates the base relations also updating the materialized views. However, such immediate updates impose significant overheads on update transactions that cannot be tolerated by many applications. *Deferred view maintenance*, on the other hand, allows the view to become inconsistent with its definition, and a *refresh operation* is required to establish consistency. Typically, under deferred maintenance, a view is *incrementally* updated only just before data is retrieved from it (i.e., on-demand, just before a query is performed on the view).

Historical Background

Early systems that supported views did so in their "pure form," i.e., by storing just the view definition and using query rewriting to take advantage of views in other queries [11].

Incremental view maintenance is introduced in [1] through a technique to efficiently detect relevant updates to materialized views, thus streamlining their maintenance.

Deferred view maintenance is introduced in [10] as a scheme for materializing copies of views on workstations attached to a mainframe that maintains a shared global database. The workstations update local copies of the views while processing queries. In [7], deferred view maintenance is defined as the application of incremental view maintenance whenever desired, unlike the immediate view maintenance, where any database update triggers the incremental view maintenance algorithm. [5] has a nice survey of view maintenance techniques.

Foundations

Algorithms and techniques for maintenance of materialized views can be classified according to three different criteria:

- Whether the view is recomputed from scratch or not: *recomputation* versus *incremental* maintenance.
- Whether the view is updated whenever the base data change or not: *immediate* versus *deferred* maintenance.
- Whether queries can be executed while the view is being updated or not: *online* versus *offline* maintenance.

All the above dimensions are typically orthogonal. We explain the different options below.

View Recomputation

Recomputing a materialized view from the base relations it is derived from is the most general technique of updating. As such, it can be applied on any type of view, regardless of the complexity of the query definition. The disadvantage is that, in most cases, such recomputation is costly, and, in many cases, the view can be updated *incrementally* instead, at a fraction of the cost.

Incremental View Maintenance

It is possible to update a materialized view *incrementally* for many types of view definitions (i.e., queries). One such class is the general case of SPJ views (i.e., views whose definition is just a select-project-join query).

For example, assume that we have a view V defined over two relations R and S through a natural join (i.e., $V = R \bowtie S$; for simplicity of the presentation we ignore the selection and projection operators). Further, let us assume that we have a set of deleted tuples from relation R, denoted as R_D; a set of inserted tuples into relation R, denoted as R_I (i.e., $R^{'} = R \cup R_I - R_D$). Also, assume a set of deleted tuples from relation S, denoted as S_D; and a set of inserted tuples into relation S, denoted as S_I (i.e., $S^{'} = S \cup S_I - S_D$). We trivially represent base relation updates as pairs of deletions and insertions.

Given the above, the updated version of V, i.e., $V^{'}$, should be $V^{'} = R^{'} \bowtie S^{'} = (R \cup R_I - R_D) \bowtie (S \cup S_I - S_D)$. By expanding this further, and grouping all the deletions from V as V_D and all the insertions to V as V_I, we have that: $V_D = (R_D \bowtie (S \cup S_I)) \bowtie ((R \cup R_I) \bowtie S_D)$, and $V_I = (R_I \bowtie S) \cup (R \bowtie S_I) \cup (R_I \bowtie S_I)$, so that $V^{'} = V \cup V_I - V_D$. This, incrementally computed formula, should be less costly to compute than recomputing the entire join from scratch.

The problem of incrementally updating materialized views is difficult in the general case, but there are additional classes of queries (i.e., besides SPJ views) that it can be solved for [6].

Immediate View Maintenance

The default way of updating materialized views is to do so *immediately*, i.e., batch together, in a single transaction, the updating of the base relations and the updating of the materialized views that are derived from these relations. However, many applications cannot tolerate this delay, especially if they are interactive and users are expecting an answer at transaction commit.

Deferred View Maintenance

Incremental deferred view maintenance requires (i) techniques for checking what views are affected by an update to the basic tables, (ii)auxiliary tables that maintain certain information like updates and deletes since the last view refresh and finally (iii) techniques for propagating the changes from the base tuples to the view tuples without fully recomputing the view relation.

First, Buneman in [2], proposes a technique for the efficient implementation of alerters and triggers that checks each update operation prior to execution to see whether it can cause a view to change. In [1], an efficient method for identifying updates that cannot possibly affect the views is described. Such *irrelevant updates* are then removed from consideration while differentially updating the views.

In [7], the hypothetical relations technique developed in [12] is adapted to the purpose of storing and indexing the deltas to the base tables. The main idea is to use a single table *AD* that stores deletions and insertions for the base tables (updates can be modeled as a deletion followed by an update). Whenever a view is accessed, the base tables and the *AD* table need to be accessed (in order to check for new or deleted tuples). A bloom filter however, is used to check if a tuple from the base relation exists in *AD* significantly reducing irrelevant accesses to *AD*.

In [4], the authors demonstrate that the ordering of the updates from the base tuples to the view tuples is critical and call this phenomenon *state bug*. Typically, an "incremental query" – during the refresh operation – avoids recomputing the full view and only incrementally computes the delta view to bring it up to date, based on updates/deletes made to the base tables. Such incremental queries can evaluated in two states: The *pre-update state*, where the base table updates have not been applied yet or the *post-update state* where changes have been applied. In most techniques a pre-update state is assumed which severely limits the class of updates and views considered. The post-update state allows for a much larger class of view to be deferred maintained, however direct application of pre-update techniques results in incorrect answers (state bug) and new techniques are proposed.

Offline View Maintenance

Typically, maintaining materialized views is done *offline*, without allowing queries to the materialized view to execute concurrently with the processing of the materialized view updates. This simplifies the view maintenance algorithms significantly, at the expense of delaying queries. Traditionally, in data warehousing environments [3], updates of materialized views are performed at night, thus minimizing the possibility of delaying user queries.

Online View Maintenance

The need of most companies for continuous operation (especially in the presence of the Web), has precipitated the need for *online view maintenance*, where queries can be answered while the materialized views are being updated.

In a centralized setting, this is typically achieved through some sort of multi-versioning, either as *horizontal redundancy*, where extra columns are added to hold the different versions [9], or as *vertical redundancy*, where extra rows are needed to hold the different versions [8]. In a distributed setting, this is typically achieved through determination of additional queries to ask of the data sources [13].

Key Applications

Materialized views help speed up the execution of frequently accessed queries, giving interactive response times to even the most complex queries. The cost of maintaining materialized views is typically amortized over multiple accesses (i.e., queries to the view). This has been utilized/transferred in many different application domains, from data warehousing to web data management. Beyond efficient algorithms and techniques to update materialized views, special attention has also been given to the *view selection* problem: how to identify which views should be materialized, and also to the issue of how to effectively use materialized views to answer other queries (i.e., by utilizing subsumption or caching).

Cross-references

▶ Recursive View Maintenance
▶ View Selection

Recommended Reading

1. Blakeley J.A., Larson P.Å., and Tompa F.W. Efficiently Updating Materialized Views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 61–71.
2. Buneman P. and Clemons E.K. Efficient Monitoring Relational Databases. ACM Trans. Database Syst., 4(3):368–382, 1979.
3. Chaudhuri S. and Dayal U. An overview of data warehousing and OLAP technology. ACM SIGMOD Rec., 26(1):65–74, 1997.
4. Colby L.S., Griffin T., Libkin L., Mumick I.S., and Trickey H. Algorithms for deferred view maintenance. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 469–480.
5. Gupta A. and Mumick I.S. Maintenance of materialized views: problems, techniques, and applications. IEEE Data Eng. Bull., 18(2):3–18, 1995.
6. Gupta A., Mumick I.S., and Subrahmanian V.S. Maintaining views incrementally. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 157–166.
7. Hanson E.N. A performance analysis of view materialization strategies. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 440–453.
8. Labrinidis A. and Roussopoulos N. A performance evaluation of online warehouse update algorithms. Tech. Rep. CS-TR-3954, Department of Computer Science, University of Maryland, 1998.
9. Quass D. and Widom J. On-line warehouse view maintenance. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 393–404.
10. Roussopoulos N. and Kang H. Principles and Techniques in the Design of ADMS±. IEEE Comp., 19(12):19–25, 1986.
11. Stonebraker M. Implementation of integrity constraints and views by query modification. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1975, pp. 65–78.
12. Woodfill J. and Stonebraker M. An implementation of hypothetical relations. In Proc. 9th Int. Conf. on Very Data Bases, 1983, pp. 157–166.
13. Zhuge Y., Garcia-Molina H., Hammer J., and Widom J. View maintenance in a warehousing environment. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 316–327.

View Maintenance Aspects

Antonios Deligiannakis
University of Athens, Athens, Greece

Definition

Database systems often define *views* in order to provide conceptual subsets of the data to different users. Each view may be very complex and require joining information from multiple *base relations*, or other views. A view can simply be used as a query modification mechanism, where user queries referring to a particular view are appropriately modified based on the definition of the view. However, in applications where fast response times to user queries are essential, views are often materialized by storing their tuples inside the database. This is extremely useful when recomputing the view from the base relations is very expensive. When changes occur to their base relations, materialized views need to be updated, with a process known as view maintenance, in order to provide fresh data to the user.

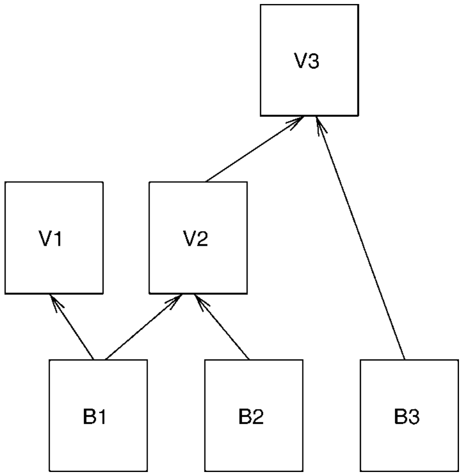
Historical Background

The use of relational views has long been proposed in relational database systems. The notion of materialized views, or *snapshots*, was first proposed in [1]. A snapshot represents the state of some portion of the database at the time when the snapshot was computed. Since the publication of [1], a large number of mechanisms for refreshing materialized views has been proposed. These mechanisms mainly involve either the time of refreshing the materialized views, or how such a refresh operation can be performed (i.e., through a re-computation of the view, or in an incremental manner). While each policy has its own advantages and drawbacks, ADMS was the first system that realized the importance of having multiple maintenance policies within the same database [11,12]. An excellent classification of the view maintenance problem was presented in [4].

Foundations

Materialized views are used in order to provide faster response times to user queries. Figure 1 depicts a sample *view-dependency graph* of three materialized views (namely, V1, V2 and V3) defined over three base relations (denoted as B1, B2 and B3). If a view V contains in its definition a reference to a base relation (or another view) B, then there exists a directed edge in the view-dependency graph from B to V. Given this explanation, please note that V3 in the sample view-dependency graph is expressed in terms of a base relation (B3) and another materialized view (V2). However, the ultimate set of base relations that are used to derive V3 are B1, B2 and B3.

When the underlying data of the base relations changes, the materialized views contain stale data, until the materialized view is refreshed. The view maintenance can be performed either through a re-computation of the view or, if possible, through an incremental procedure. An incremental update policy is often faster, and is thus in many cases desirable, if the base relations have been modified only by a small portion. However, when a large portion of some base relations is updated (i.e., most tuples of a base relation are deleted), then a complete re-computation may actually be faster. The process of maintaining a materialized view can be classified based on five main dimensions.



View Maintenance Aspects. Figure 1. An example of a view-dependency graph.

Time Dimension

Every update transaction incurs a time penalty, thus slowing down queries performed during the update process. On the other hand, some applications (i.e., applications regarding stock data) cannot tolerate viewing stale data.

Depending on when a materialized view is updated, three main policies for view maintenance can be defined [2]:

1. Immediate Views: Upon an update to a base relation, the materialized view is updated immediately. This is often accomplished using database triggers.
2. Deferred Views: Updating the materialized view is deferred until the first time when the view is queried.
3. Snapshot Views: The view is updated periodically (i.e., at daily intervals) by an asynchronous process.

Each of the aforementioned policies has some advantages and drawbacks, and each may be the policy of choice, depending on the characteristics of the targeted application. Immediate views incur an update penalty even if the updated tuples are never queried. Deferred views incur this update cost only at the first time when the view is queried after its base relations have been updated. Thus, deferred views incur a lower update overhead, but also worse query performance, since the query evaluation process may have to wait for some views to be updated, when compared to the immediate views policy. The snapshot views policy leads to faster query performance, since typically no updates are performed in parallel to the user queries, and to better update performance, since the updates are batched. Thus, snapshot views represent an attractive choice when the application may tolerate reading stale data.

Expressiveness of View Definition Language

Algorithms proposed for the maintenance of a materialized view can often handle only views expressed as a subset of SQL. For example, some techniques may only be able to handle views defined as select-project-join (SPJ) queries. However, a view definition may be much more complex, as it may contain (amongst others) duplicates, arithmetic operations, aggregate or ranking functions, nested subqueries, set operations such as calculating the union or difference amongst two sets, outer-joins, recursion etc.

View maintenance algorithms often seek to derive formulas for determining the update to a view, and thus avoid recalculating the entire view from scratch, based on the updates to its base relations. For example, for the sample dependency graph presented in Fig. 1, let view $V2$ be calculated as: $V2 = B1 \bowtie_{JC} B2$, where JC denotes the join condition in the definition of $V2$ between relations $B1$ and $B2$. If $\Delta B1$ and $\Delta B2$ represent the inserted (deleted) tuples to relations $B1$ and $B2$, then the inserted (deleted) tuples $\Delta V2$ to view $V2$ can be calculated as:

$$\Delta V2 = (\Delta B1 \bowtie_{JC} B2) \cup (B1 \bowtie_{JC} \Delta B2) \cup (\Delta B1 \bowtie_{JC} \Delta B2)$$

Updates can be modeled as deletions followed by insertions. Deriving such formulas for computing the incremental update operations on a view typically becomes harder, or even impossible, as the expressiveness of the view definition language is increased. Thus, maintenance algorithms often first check whether a view can be incrementally maintained, and then proceed to decide how to actually maintain it.

Available Information

When maintaining a materialized view, we do not always have access to both the materialized view and its base relations. For example, when one of the base relations represents a data stream, as in the case of Chronicle Views [7], the entire relation cannot be stored and is, thus, unavailable during the maintenance process. Thus, during the view maintenance process one may, or may not, have access to the materialized view itself, to its base relations, or to information regarding the presence of keys and referential integrity constraints. Depending on the amount of information available, different algorithms can be used for maintaining a view.

Supported Modifications

The view maintenance algorithms can also be classified based on types of modifications to the base relations that they can handle. Not all algorithms handle both insertions and deletions to base relations. Some algorithms handle sets of modifications (i.e., insertions and deletions) in a single pass, as in [6], while others require one pass for each different modification set. Some algorithms handle updates directly, while other algorithms model them as deletions followed by insertions. Finally, not all view maintenance algorithms can handle more complex modifications, such as modifications to the view definition. Even when such modifications are handled, the view maintenance can be achieved by different techniques that either recompute the view, or try to *adapt* the view, so that the old view can be used to materialize the new view [5].

Algorithm Applicability

View maintenance algorithms should be able to be applied to all possible data stored in the database tables, and to all instances of a particular modification (i.e., insertion, deletion, update etc). Techniques that operate correctly but only on specific database instances, or on specific only modification instances are less desirable.

Figure 2, originally presented in [4], summarizes some areas of the problem space for three of the five dimensions described above, namely for the expressiveness of the view definition language, for the available information and for the supported modifications dimensions. The remaining two dimensions have been omitted for ease of presentation.

Key Applications

Data Warehousing

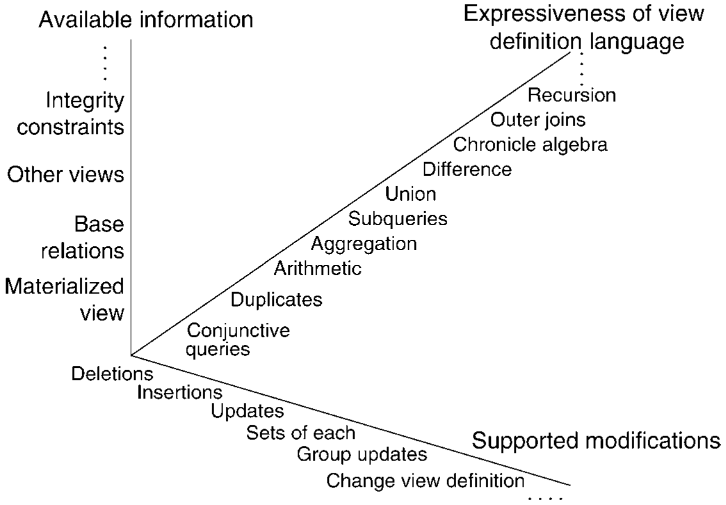
Materialized views are frequently used in data warehouses to provide personalized behavior to users, store frequently queried data derived from multiple relations, and to encapsulate different views of local or even remote data and databases. Materialization allows for significantly better query performance [8], while view maintenance, in the case of Immediate or Deferred Views, allows users to access fresh data.

Data Streams

Banking, billing, networking and stock applications often generate infinite streams of data. View maintenance algorithms can help answer complex queries over such infinite data streams without requiring access to the entire stream.

Caching

Cached data can be refreshed quickly using techniques developed for view maintenance when only a small portion of the cache has become stale. The cached



View Maintenance Aspects. Figure 2. The problem space, as presented in [4].

content may also involve dynamically generated web page fragments. In this case, the notion of WebViews [9,10] can be used to decide which such fragments to materialize, and how to determine a schedule for updating them.

Mobile Applications

If data needs to be transmitted in cases of bandwidth constraints, as in applications of mobile clients holding a cell phone or a GPS system, then only the data altered between the last transmission needs to be transmitted. Similar techniques that transmit only the changes in the computed data can also be used in other bandwidth-constrained applications as well, such as in transmitting smaller amounts of data to/from sensor nodes [3].

Cross-references

▶ Database Tuning and Performance
▶ Deferred View Maintenance
▶ Incremental View Maintenance
▶ Maintenance of Recursive Views
▶ Maintenance of Materialized Views with Outer-Joins
▶ Query Processing and Optimization in Object Relational Databases
▶ View Maintenance
▶ Viewcaches

Recommended Reading

1. Adiba B. and Lindsay B. Database snapshots. In Proc. 6th Int. Conf. on Very Data Bases, 1980, pp. 86–91.
2. Colby L., Kawaguchi A., Lieuwen D., Mumick I.S., and Ross K.A. Supporting multiple view maintenance policies. In Proc. ACM SIGMOD Conf. on Management of Data, 1997, pp. 405–416.
3. Deligiannakis A., Kotidis Y., and Roussopoulos N. Processing approximate aggregate queries in wireless sensor networks. Inf. Syst., 31(8):770–792, December 2006.
4. Gupta A. and Mumick I.S. Maintenance of materialized views: problems, techniques, and applications. IEEE Data Eng. Bull., Special Issue on Materialized Views and Data Warehousing, 18(2):3–19, June 1995.
5. Gupta A., Mumick I.S., Rao J., and Ross K.A. Adapting materialized views after redefinitions: techniques and a performance study. Inf. Syst., Special Issue on Data Warehousing, 16(5):323–362, July 2001.
6. Gupta A., Mumick I.S., and Subrahmanian V.S. Maintaining views incrementally. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 157–166.
7. Jagadish H.V., Mumick I.S., and Silberschatz A. View maintenance issues in the chronicle data model. In Proc. 14th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1995, pp. 113–124.
8. Kotidis Y. and Roussopoulos N. DynaMat: a dynamic view management system for data warehouses. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 371–382.
9. Labrinidis A. and Roussopoulos N. WebView materialization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 367–378.
10. Labrinidis A. and Roussopoulos N. Balancing performance and data freshness in Web database servers. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 393–404.
11. Roussopoulos N. The incremental access method of view cache: concept, algorithms, and cost analysis. ACM Trans. Database Syst., 16(3):535–563, September 1991.
12. Roussopoulos N. and Kang H. Principles and techniques in the design of ADMS±. IEEE Comput., 19(2):19–25, December 1986.

View Expression

► View Definition

View Update

► View Maintenance

View-based Data Integration

Yannis Katsis, Yannis Papakonstantinou
University of California-San Diego, La Jolla, CA, USA

Definition

Data Integration (or *Information Integration*) is the problem of finding and combining data from different sources. *View-based Data Integration* is a framework that solves the data integration problem for *structured* data by integrating sources into a single unified view. This integration is facilitated by a declarative *mapping language* that allows the specification of how each source relates to the unified view. Depending on the type of view specification language used, view-based data integration systems (VDISs) are said to follow the *Global as View (GAV)*, *Local as View (LAV)* or *Global and Local as View (GLAV)* approach.

Historical Background

Data needed by an application are often provided by a multitude of data sources. The sources often employ heterogeneous data formats (e.g., text files, web pages, XML documents, relational databases), structure the data in different ways and can be accessed through different methods (e.g., web forms, database clients). This makes the task of combining information from multiple sources particularly challenging. To carry it out, one has to retrieve data from each source individually, understand how the data of the sources relate to each other and merge them, while accounting for discrepancies in the structure and the values, as well as for potential inconsistencies.

The first to realize this problem were companies willing to integrate their structured data within or across organizations. Soon the idea of integrating the data into a single unified view emerged. These systems, to be referred to as view-based integration systems (VDISs) would provide a single point of access to all underlying data sources. Users of a VDIS (or applications) would query the unified view and get back integrated results from all sources, whereas the task of combining data from the sources and resolving inconsistencies would be handled by the system transparently to the applications.

The VDISs made their first appearance in the form of multidatabases and federated systems [11]. Subsequently, the research community dived into the problem of specifying the correspondence between the sources and the unified view. The outcome were three categories of languages to express the correspondence (GAV, LAV and GLAV), together with several related theoretical and system results. The industry also embraced the view-based integration framework by creating many successful VDISs (e.g., BEA AquaLogic, IBM WebSphere).

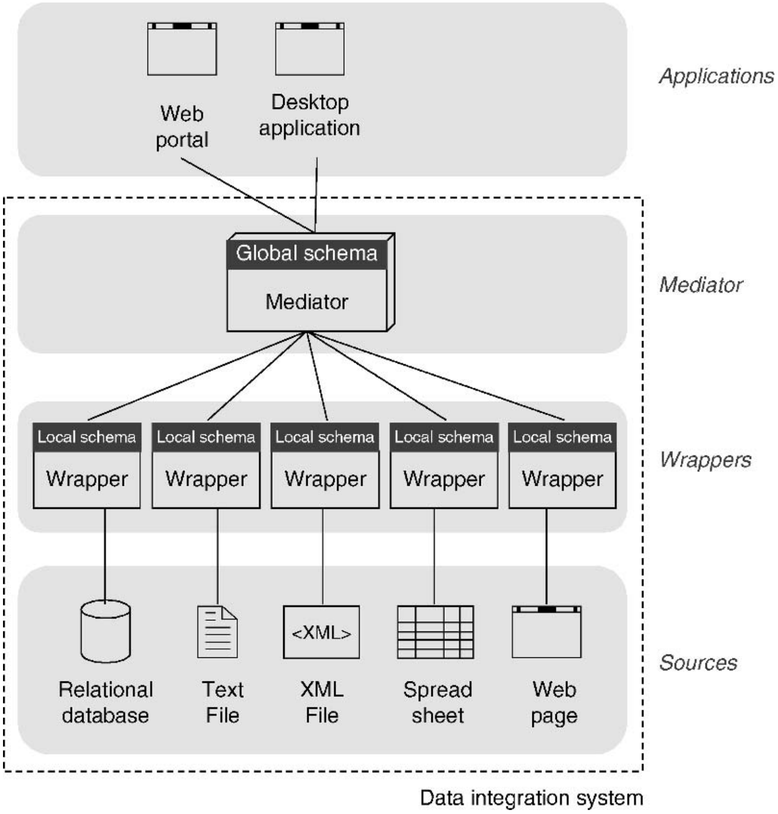
Foundations

Abstracting out the differences between individual systems, a typical VDIS conforms to the architecture shown in Fig. 1. Sources store the data in a variety of formats (relational databases, text files, etc.). Wrappers solve the heterogeneity in the formats by transforming each source's data model into a common data model used by the integration system. The wrapped data sources are usually referred to as local or source databases, the structure of which is described by corresponding local/source schemas. This is in contrast to the unified view exported by the mediator, also called global/target database. Finally, mappings expressed in a certain mapping language (depicted as lines between the wrapped sources and the mediator) specify the relationship between the wrapped data sources (i.e., the local schemas) and the unified view exported by the mediator (i.e., the global schema).

Given a VDIS, applications or users retrieve data from the sources indirectly by querying the global schema. It is the task of the mediator to consult the mappings to decide which data to retrieve from the sources, and how to combine them appropriately in order to form the answer to the user's query.

VDISs can be categorized according to the following three main axes:

1. *Common data model and query language.* The data model and query language that is exposed by the



View-based Data Integration. Figure 1. View-based Data Integration System (VDIS) architecture.

wrappers to the mediator and by the mediator to the applications. Commonly used data models include the relational, XML and object-oriented data model.

2. *Mapping language.* The language for specifying the relationship of sources with the view. Languages proposed in the literature fall into three categories; *Global as View (GAV)*, *Local as View (LAV)* and *Global and Local as View (GLAV)*. Being one of the most important components in a VDIS, these are discussed in detail below.
3. *Data storage method.* The decision on the place where the data are actually stored. The two extremes are the materialized and the virtual approach (see [14] for a comparison). In the materialized integration (also known as eager, in-advance or warehousing approach), all source data are replicated on the mediator. On the other hand, in the virtual mediation (e.g., Infomaster [6]) (or lazy approach), the data are kept in the sources and the global database is virtual. Consequently, a query against the global database cannot be answered directly, but has to be translated to queries against the actual sources. Finally, some systems employ hybrid policies, such as virtualization accompanied by a cache.

Specifying the Relationship of the Sources with the Unified View

To allow the mediator to decide which data to retrieve from each source and how to combine them into the unified view, the administrator of the VDIS has to specify the correspondence between the local schema of each source and the global schema through mappings.

The mappings are expressed in a language, corresponding to some class of logic formulas. Languages proposed in the literature fall into three categories: *Global as View (GAV)*, *Local as View (LAV)* and *Global and Local as View (GLAV)*. In GAV the global database (schema) is expressed as a function of the local databases (schemas). LAV on the other hand follows the opposite direction, with each local schema being described as a function over the global schema. Therefore, LAV allows to add a source to the system independently of other sources. Finally, GLAV is a generalization of the two. This section presents each of these approaches in detail and explains their implications on the query answering algorithms. Essentially, each of them represents a different trade-off between expressivity and hardness in query answering.

Running example. For ease of exposition, the following discussion employs a running example of integrating information about books. The example employs the relational data model for both the sources and the global database. Moreover, the query language used by the users to extract information from the global database, and in turn by the mediator to retrieve information from the sources, is the language of conjunctive queries with equalities ($CQ^{=}$); a subset of SQL widely adopted in database research.

Figure 2 shows the employed local and global schemas. Relations are depicted in italics and their attributes appear nested in them. For instance, the global schema $\mathcal{G}$ in Fig. 2b consists of two relations Book and Book_Price. Relation Book has attributes ISBN, title, suggested retail price, author and publisher, while Book_Price stores the book price and stock information for different sellers. Data is fueled into the system by two sources; the databases of the bookstore Barnes & Noble (B&N) and the publisher Prentice Hall (PH), with the schemas shown in Fig. 2a. Note that there are two versions of the PH schema, used in different examples.

Global as View (GAV)

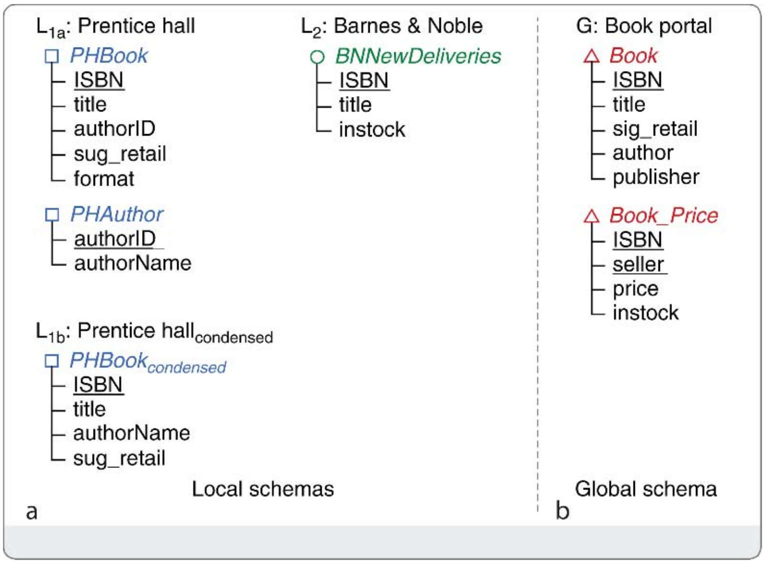
Historically, the first VDISs followed the Global as View (GAV) approach [7,12], in which the global schema is described in terms of the local schemas. In such systems the contents of each relation R in the target schema $\mathcal{G}$ are specified through a query (view) V over the combined schemas of the sources.

Thus, in GAV the correspondence between the local schemas and the global schema can be described through a set of mappings of the form:

$$V_i \rightarrow I(R_i)$$

one for each relation R_i of the global schema, where V_i is a query over the combined source schemas. $I(R_i)$ is the identity query over R_i (i.e., a query that returns all attributes of R_i). The symbol → can represent either query containment ($\subseteq$) or query equality ($=$). This leads to two different semantics, referred to in the literature as the open-world and the closed-world assumption, respectively.

Example: Consider the following two GAV mappings (For the examples, the identity query I over some relation is considered to return the attributes of that relation in the same order as they appear on the schema in Fig. 2).



View-based Data Integration. Figure 2. Local and global schemas of the running example.

$M_1 : V_1 \rightarrow I(Book)$
$M_2 : V_2 \rightarrow I(Book\_Price)$

where

$V_1(ISBN, title, sug\_retail, authorName, \text{"PH"})$:-
$PHBook(ISBN, title, authorID, sug\_retail, format)$,
$PHAuthor(authorID, authorName)$

and

$V_2(ISBN, \text{"B\&N"}, sug\_retail, instock)$:-
$PHBook(ISBN, title1, authorID, sug\_retail, format)$,
$BNNewDeliveries(ISBN, title2, instock)$

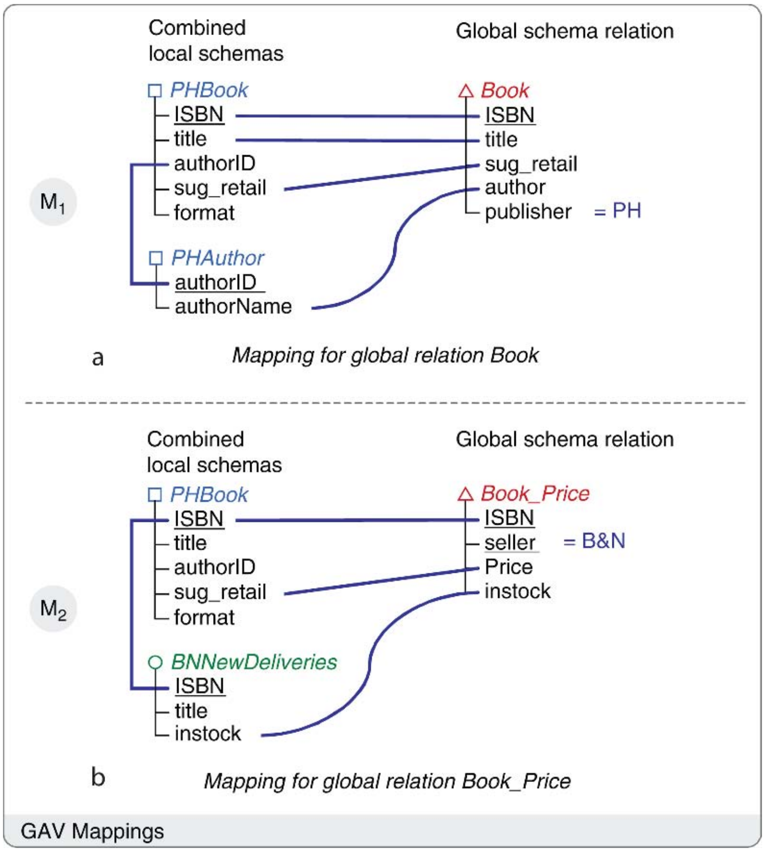
The mappings are graphically depicted in Fig. 3a and b, as described in [9]. This is similar to the way most *mapping tools* (i.e., tools that allow a visual specification of mappings), such as IBM Clio and MS BizTalk Mapper, display mappings.

Mapping M_1 intuitively describes how Book tuples in the global database are created. This is done by retrieving the ISBN, title and sug_retail from a PHBook tuple, the author from the corresponding PHAuthor tuple (i.e., the PHAuthor tuple with the same authorID as the PHBook tuple), and finally setting the publisher to "PH" (since the extracted books are published by PH).

Similarly, mapping M_2 describes the construction of the global relation Book_Price. This involves combining information from multiple sources: the price from the suggested retail price information provided by PH and the inventory information from B&N, because B&N's administrator knows that B&N's sells its books at the suggested retail price.

Query Answering in GAV. GAV mappings have a *procedural* flavor, since they describe how the global database can be constructed from the local databases. For this reason, query answering in GAV is straightforward, both in the materialized and in the virtual approach.

In the materialized approach, the source data are replicated in the global database by executing for each mapping $V_i \rightarrow I(R_i)$ the query V_i against the local databases and populating R_i with the query results.



View-based Data Integration. Figure 3. Example of GAV mappings.

Subsequently, an application query Q against the global schema is answered by simply running Q over the materialized global database.

On the other hand, in the virtual approach, data are kept in the sources and thus a query against the global schema has to be translated to corresponding queries against the local schemas. Due to the procedural flavor of GAV, this can be done through view unfolding (i.e., replacing each relational atom of the global schema in the query by the corresponding view definition). Intuitively, whenever a query asks for a global relation R_i, it will instead run the subquery V_i over the local schemas, which, according to the mapping $V_i \rightarrow I(R_i)$, provides the contents of R_i.

Advantages. The simplicity of the GAV rules together with the straight-forward implementation of query answering, led to the wide adoption of GAV by industrial systems. From the research sector representative GAV-based VDISs systems are MULTIBASE [11], TSIMMIS [5] and Garlic [2].

Disadvantages. GAV also has several drawbacks:

First, since the global schema is expressed in terms of the sources, *global relations cannot model any information not present in at least one source.* For instance, the Book relation in the example could not contain an attribute for the book weight, since no source currently provides it. In other words, the value of each global attribute has to be explicitly specified (i.e., in the visual representation all global attributes must have an incoming line or an equality with a constant).

Second, as observed in mapping M_2 of the running example, *a mapping has to explicitly specify how data from multiple sources are combined to form global relation tuples.* Therefore, GAV-based systems do not facilitate adding a source to the system independently of other sources. Instead, when a new source wants to join the system, the system administrator has to inspect how its data can be merged with those of the other sources currently in the system and modify the corresponding mappings.

Local as View (LAV)

To overcome the shortcomings of GAV, researchers came up with the Local as View (LAV) approach [7,12]. While in GAV the global schema is described in terms of the local schemas, LAV follows the opposite direction expressing each local schema as a function of the global schema. LAV essentially corresponds to the "source owners view" of the system by describing which data of the global database are present in the source. Using the same notation as in GAV, local-to-global correspondences can be written in LAV as a set of mappings:

$$I(R_i) \rightarrow U_i$$

one for every relation R_i in the local schemas, where U_i is a query over the global schema and I the identity query.

Example: Figure 4 shows the following two LAV mappings for the running example:

$M'_1 : I(PHBook_{condensed}) \rightarrow U_1$
$M'_2 : I(BNNewDeliveries) \rightarrow U_2$

where

$U_1(ISBN, title, author, sug\_retail)$:-
  $Book(ISBN, title, sug\_retail, author, \text{"}PH\text{"})$

and

$U_2(ISBN, title, instock)$:-
  $Book(ISBN, title, sug\_retail, author, publisher)$,
  $Book\_Price(ISBN, \text{"}B\&N\text{"}, sug\_retail, instock)$

For instance, M'_1 specifies that $PHBook_{condensed}$ contains information about books published by PH. Similarly, M'_2 declares that *BNNewDeliveries* contains the ISBN and title of books sold by B&N at their suggested retail price and whether B&N has them in stock.

In contrast to GAV mappings, LAV mappings have a *declarative* flavor, since, instead of explaining how the global database can be created, they describe what information of the global database is contained in each local database.

Advantages. LAV addresses many of GAV problems with the most important being that sources can register independently of each other, since a source's mappings do not refer to other sources in the system.

Disadvantages. LAV suffers from the symmetric drawbacks of GAV. In particular, it cannot model sources that have information not present in the global schema (this is the reason why the example above used the condensed version of PH's schema that did not contain the attribute format, which is not present in the global schema). Furthermore, due to LAV's declarative nature, query answering is non-trivial any more, as described next. Mainly because of its technical implications to query answering, LAV has been extensively studied in the literature. Representative LAV-based systems include Information Manifold [10] and the system described in [13].

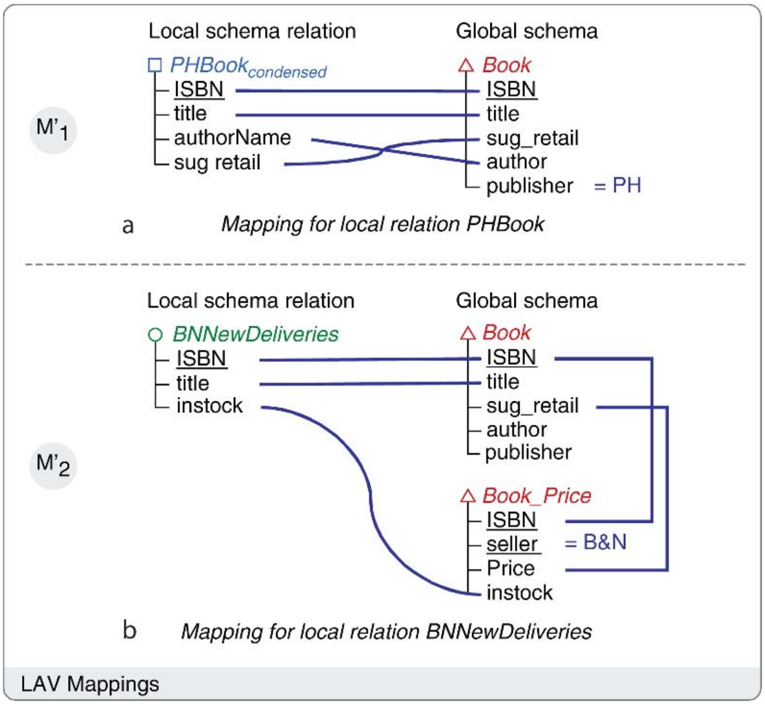
Query Answering in LAV. Since LAV mappings consist of an arbitrary query over the global schema, they may leave some information of the global database unspecified. For instance, mapping M'_2 above only states that B&N sells its books at the suggested retail price, without specifying the exact price. Thus, there might be infinitely many global databases that could be inferred from the sources through the mappings (each of them would make sure that each pair of Book and Book_Price tuples created from a BNNewDeliveries tuple share the same value for price, but each such global database might choose a different constant for the value). These databases are called *possible worlds*. Their existence has two important implications:

First, since a unique global database does not exist, it cannot be materialized and therefore LAV lends itself better to virtual mediation. However, there is still a way of replicating source information in a centralized place. This involves creating a "special" database that intuitively stores the general shape of all possible worlds. This "special" database is called *canonical universal solution* and can be built through procedures employed in data exchange [3].

Second, since many global databases exist, the query answering semantics need to be redefined. The standard semantics adopted in the literature for query answering in LAV-based systems are based on the notion of *certain answers* [1,8]. The certain answers to a query are the answers to the query, which will always appear regardless of which possible world the query is executed against (i.e., the tuples that appear in the intersection of the sets of query answers against each possible world). Intuitively, certain answers return information that *is guaranteed* to exist in *any* possible world.

Example: If in the integration system of Fig. 4 a query asks for all ISBNs that are sold by some seller at their suggested retail price, it will get back all ISBNs stored in relation BNNewDeliveries, because for each of them, any possible world will contain a pair of Book and Book_Price tuples with the same ISBN and the same prices (although these prices will have different values between possible worlds). On the other hand, if the query asks for all books sold at a specific price, it will not get back the ISBNs from BNNewDeliveries, because their exact prices are left unspecified by the mapping M'_2 and will therefore differ among possible worlds.

In order to compute the certain answers to a query in a virtual integration system following the LAV approach, the query against the global schema has to be translated to corresponding queries against the local schemas. This problem is called *rewriting queries using*



View-based Data Integration. Figure 4. Example of LAV mappings.

views (because the query over the global database has to be answered by using the sources which are expressed as views over it), and is also of interest to other areas of database research, such as query optimization and physical data independence (see [8] for a survey). In contrast to GAV, it is a non-trivial problem studied extensively by researchers.

Global and Local as View (GLAV)

To overcome the limitations of both GAV and LAV, [4] proposed a new category of mapping languages, called Global and Local as View (GLAV), which is a generalization of both GAV and LAV. GLAV mappings are of the form:

$$V_i \rightarrow U_i$$

where V_i, U_i are queries over the local and global schemas, respectively.

GLAV languages can trivially express both GAV mappings and LAV mappings by assigning to U_i a query returning a single global relation or to V_i a query asking for a single local relation, respectively. However, GLAV is a strict superset of both GAV and LAV by allowing the formulation of mappings that do not fall either under GAV or under LAV (i.e., mappings in which V_i and U_i both do not return just a single local or global relation).

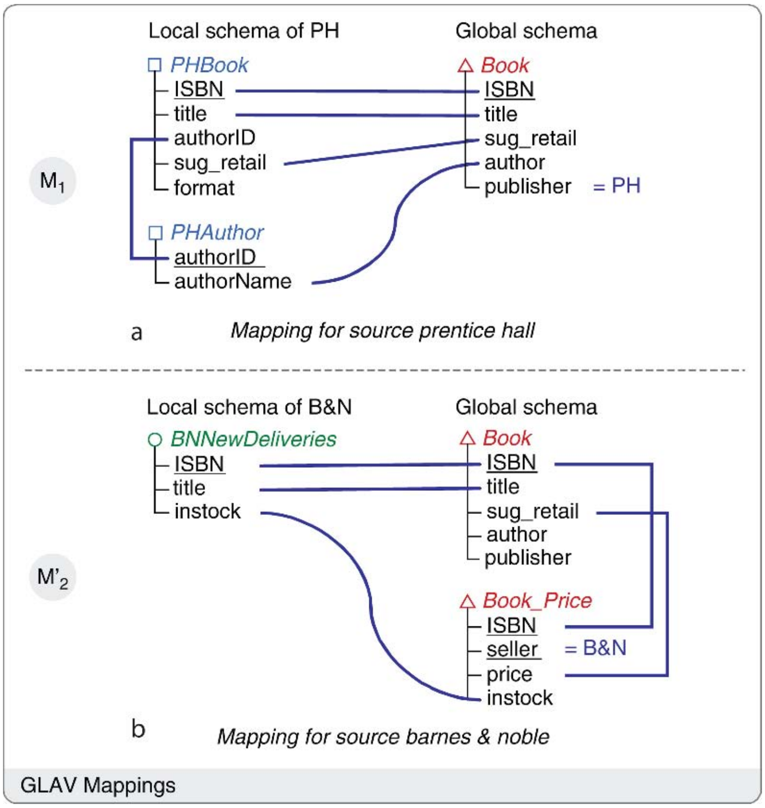
Example: Figure 5 shows two GLAV mappings. The first mapping is the GAV mapping M_1 first presented in Fig. 3a, while the second mapping is the LAV mapping M'_2 used in Fig. 4b.

Since U_i (a.k.a. the conclusion of the mapping) can be an arbitrary query over the global schema, GLAV allows the independent registration of sources in the same way as LAV. However, this also implies that the global database is incomplete. Consequently, query answering in GLAV is usually done under certain answer semantics, by extending query rewriting algorithms for LAV [15].

Alternatives to VDISs

Apart from the VDISs, research and industrial work led to many alternative approaches to data integration of structured data:

1. *Vertical Integration Systems* are specialized applications that solve the problem of data



View-based Data Integration. Figure 5. Example of GLAV mappings.

integration for a specific domain. For example, http://www.mySimon.com or http://www.rottentomatoes.com integrate price and movie information, respectively.

2. *Extract Transform Load (ETL) Tools* generally facilitate the actual migration of data from one system to another. When used for data integration they are closely tied to the problem of materializing the integrated database in a central data warehouse.

Compared to these solutions, VDISs offer a more general approach to data integration with the following advantages: (i) The relationships between the sources and the unified view are explicitly stated and not hidden inside a particular implementation, (ii) a general VDIS implementation can be used in many different domains, and (iii) a VDIS deployment can be easily utilized by many applications, as shown in Fig. 1. In a way VDISs are analogous to Database Management Systems, offering a general way of managing the data (which in VDISs are heterogeneous and distributed), independently of the applications that need those data.

Finally, another alternative to VDISs are Peer-to-Peer (P2P) Integration Systems that drop the requirement for a single unified view, allowing queries to be posed over any source schema. Although it is an active area of research, P2P systems have not yet been widely adopted in industry.

Key Applications

VDISs are used for integration of structured data in many different settings, including among others enterprises, government agencies and scientific communities.

Cross-references

▶ Information Integration
▶ Peer-to-Peer Data Integration
▶ Query Translation
▶ Query Rewriting Using Views

Recommended Reading

1. Abiteboul S. and Duschka O.M. Complexity of answering queries using materialized views. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 254–263.
2. Carey M.J., Haas L.M., Schwarz P.M., Arya M., Cody W.F., Fagin R., Flickner M., Luniewski A., Niblack W., Petkovic D., Thomas II J., Williams J.H., and Wimmers E.L. Towards heterogeneous multimedia information systems: The Garlic approach. In Proc. 5th Int. Workshop on Research Issues on Data Eng., 1995, pp. 124–131.
3. Fagin R., Kolaitis P.G., Miller R.J., and Popa L. Data exchange: Semantics and query answering. In Proc. Int. Conf. on Database Theory, 2002. pp. 207–224.
4. Friedman M., Levy A., and Millstein T. Navigational plans for data integration. In Proc. 16th National Conf. on AI and 11th Innovative Applications of AI Conf., 1999.
5. Garcia-Molina H.K., Papakonstantinou Y.K., Quass D.K., Rajaraman A.K., Sagiv Y.K., Ullman J.K., Vassalos V.K., and Widom J.K. The TSIMMIS approach to mediation: data models and languages. J. Intell. Inf. Syst., 8(2):117–132, 1997.
6. Genesereth M.R., Keller A.M., and Duschka O.M. Infomaster: An information integration system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997.
7. Halevy A. Logic-based techniques in data integration. In Logic Based Artif. Intell., 2000.
8. Halevy A.Y. Answering queries using views: A survey. VLDB J., 10(4):270–294, 2001.
9. Katsis Y., Deutsch A., and Papakonstantinou Y. interactive source registration in community-oriented information integration. In Proc. 34th Int. Conf. on Very Large Data Bases, 2008.
10. Kirk T., Levy A.Y., Sagiv Y., and Srivastava D. The information manifold. In Information Gathering from Heterogeneous, Distributed Environments, 1995.
11. Landers T. and Rosenberg R.L. An overview of MULTIBASE. Distributed systems, Vol. II: distributed data base systems table of contents, 1986, pp. 391–421.
12. Lenzerini M. Data integration: A theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002.
13. Manolescu I., Florescu D., and Kossmann D. Answering XML queries over heterogeneous data sources. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
14. Widom J. Research problems in data warehousing. In Proc. 27th Int. Conf. on Very Large Data Bases, 1995.
15. Yu C. and Popa L. Constraint-based XML query rewriting for data integration. In Proc. 27th Int. Conf. on Very Large Data Bases, 2004.

Views

Yannis Kotidis
Athens University of Economics and Business,
Athens, Greece

Definition

Views are virtual relations without independent existence in a database. The contents of the instance of a view are determined by the result of a query on a set of database tables or other views in the system.

Key Points

Over the years, there have been several definitions and uses for the term view. From one perspective, a view is a query (or a macro) that generates data every time it is invoked. The term view is also used in association with the derived data that is produced by the execution of the view query. Views have also been used as indexes, summary tables or combinations of both. They provide physical data independence, by decoupling the logical schema from the physical design, which is usually driven by performance reasons. By controlling access to the views, one can control access to different parts of the data and implement different security policies. Views are also used in integration systems in order to provide unified access to physically remote databases.

When the result of the view is stored as an independent table, the view is called materialized. Materialized views can be of two forms. In the first, the materialized table is treated as pure data, detached from the view definition that was used to derive its content. In the second, the view content is treated as derived data that needs to be properly maintained when update statements alter the state of the database, in a way that affects the instance of the view. This is called the view maintenance problem. In a symmetrical way, when a user or a program modifies a view (materialized or not) through an update statement, an implementation is required for translating the view update command into a series of updates on the base relations so that the requested update is observed in the view instance. This process is often termed update through views.

Cross-references

▶ Answering Queries Using Views
▶ Updates Through Views
▶ View Definition
▶ View Maintenance

Recommended Reading

1. Adiba M.E. and Lindsay B.G. Database snapshots. In Proc. 6th Int. Conf. on Very Data Bases, 1980, pp. 86–91.
2. Dayal U. and Bernstein P. On the correct translation of update operations on relational views. ACM Trans. Database Syst., 8(3):381–416, 1982.
3. Gupta A., Jagadish H.V., and Singh Mumick I. Data integration using self-maintainable views. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996, pp. 140–144.
4. Gupta H., Harinarayan V., Rajaraman A., and Ullman J.D. Index selection for OLAP. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 208–219.
5. Kotidis Y. and Roussopoulos N. DynaMat: a dynamic view management system for data warehouses. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 371–382.
6. Roussopoulos N. View indexing in relational databases. ACM Trans. Database Syst., 7(2):258–290, 1982.
7. Roussopoulos N. An incremental access method for ViewCache: concept, algorithms, and cost analysis. ACM Trans. Database Syst., 16:535–563, 1991.

Virtual Disk Manager

▶ Logical Volume Manager (LVM)

Virtual Health Record

▶ Electronic Health Records (EHR)

Virtual Partitioning

Marta Mattoso
Federal University of Rio de Janeiro, Rio de Janeiro, Brazil

Synonyms

VP

Definition

Virtual Partitioning (VP) is a distribution database design technique [2] that avoids physical partitioning table design. VP is adopted in Database Clusters to implement intra-query parallelism [3]. VP is based on database replication and aims at designing a table partition dynamically, according to each query specification. The idea is to take advantage of the current configuration of the parallel execution environment, such as the number of nodes available, the tables of the query, the current load, and the available table replicas. Intra-query parallelism can be obtained through VP by rewriting a query into a set of sub-queries to be sent to different virtual partitions. At each node, the local sequential DBMS processes the sub-query on the specified virtual partition of the table. VP requires an extra query processing phase to compose the final result from the partial ones produced by the sub-queries. This way, a database that is not physically partitioned can still be processed transparently through intra-query parallelism.

Key Points

VP is implemented through query rewriting [1]. Basically, a DBC rewrites the original query by adding range predicates to it, which originates as many sub-queries as the number of table replicas and nodes available. Then, each node receives a different sub-query. For example, let **Q** be a query on table `orders`:

```
Q: select sum (price)
   from orders
   where category = 'bolt' ;
```

A generic sub-query on a virtual partition is obtained by adding to Q's `where` clause the predicate "and `order_id` >=v1 and `order_id` < v2." By binding [v1, v2] to n subsequent ranges of `order_id` values, n sub-queries are generated, each for a different node on a different virtual partition of `orders`. These virtual partitions correspond to horizontally partitioned data. After the execution of all sub-queries each partial result would then be inserted into a temporary table, e.g., `tempResult`. Finally, to compose the result, the partitioned results need to be combined by an aggregate query. In this example: `select sum (sprice) from tempResult;`

For VP to be effective, the tuples of the virtual partition must be physically clustered according to the added range predicate. Ideally, there should be a clustered ordered index on table `orders` based on `order_id`, so that the DBMS will not scan tuples outside the VP.

Cross-references

▶ Clustering Index
▶ Data Partitioning
▶ Data Replication
▶ Distributed Database Design
▶ Horizontally Partitioned Data
▶ Intra-query Parallelism

Recommended Reading

1. Lima A.A.B., Mattoso M., and Valduriez P. Adaptive virtual partitioning for OLAP query processing in a database cluster. In Proc. 19th Brazilian Symp. on Database Systems, 2004, pp. 92–105.
2. Özsu T. and Valduriez P. Principles of Distributed Database Systems (2nd edn.). Prentice Hall, Upper Saddle River, NJ, 1999.
3. Röhm U., Böhm K., Scheck H.-J., and Schuldt H. FAS – A freshness-sensitive coordination middleware for a cluster of OLAP components. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 754–768.

Vision

▶ Visual Perception

Visual Analysis

▶ Visual Analytics
▶ Visual Data Mining

Visual Analytics

Daniel A. Keim, Florian Mansmann,
Andreas Stoffel, Hartmut Ziegler
University of Konstanz, Konstanz, Germany

Synonyms

Visual analysis; Visual data analysis; Visual data mining

Definition

Visual analytics is the science of analytical reasoning supported by interactive visual interfaces. Over the last decades, data was produced at an incredible rate. However, the ability to collect and store this data is increasing at a faster rate than the ability to analyze it. While purely automatic or purely visual analysis methods were developed in the last decades, the complex nature of many problems makes it indispensable to include humans at an early stage in the data analysis process. Visual analytics methods allow decision makers to combine their flexibility, creativity, and background knowledge with the enormous storage and processing capacities of today's computers to gain insight into complex problems. The goal of visual analytics research is thus to turn the information overload into an opportunity: Decision-makers should be enabled to examine this massive, multi-dimensional, multi-source, time-varying, and often conflicting information stream through interactive visual representations to make effective decisions in critical situations.

Historical Background

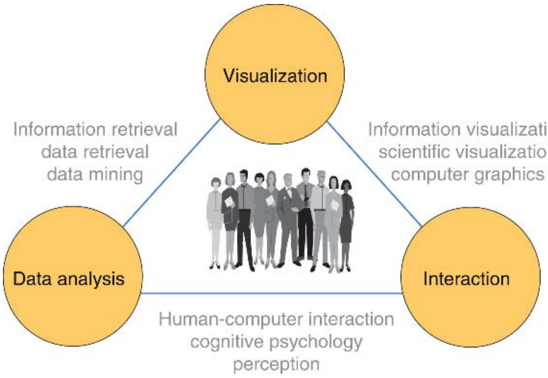
Automatic analysis techniques such as statistics and data mining developed independently from visualization and interaction techniques. However, some key thoughts changed the rather limited scope of the fields into what is today called visual analytics research. One of the most important steps in this direction was the need to move from confirmatory data analysis to exploratory data analysis, which was first stated in the statistics research community by John W. Tukey in his book "Exploratory data analysis" [7].

Later, with the availability of graphical user interfaces with proper interaction devices, a whole research community devoted their efforts to information visualization [1,2,5,8]. At some stage, this community recognized the potential of integrating the user in the KDD process through effective and efficient visualization techniques, interaction capabilities and knowledge transfer leading to *visual data exploration* or *visual data mining* [3]. This integration considerably widened the scope of both the information visualization and the data mining fields, resulting in new techniques and plenty of interesting and important research opportunities.

The term *visual analytics* was coined by Jim Thomas in the research and development agenda "Illuminating the Path" [6], which had a strong focus on Homeland Security in the United States. Meanwhile, the term is used in a wider context, describing a new multidisciplinary field that combines various research areas including visualization, human-computer interaction, data analysis, data management, geo-spatial and temporal data processing, and statistics [4].

Foundations

Visual analytics evolved from information visualization and automatic data analysis. It combines both former independent fields and strongly encourages human interaction in the analysis process as illustrated in Fig. 1. The focus of this section is to differentiate between visualization and visual analytics and thereby motivating its necessity. Thereafter, the visual analytics process is described and technical as well as social challenges of visual analytics are discussed.



Visual Analytics. Figure 1. Visual analytics as the interplay between data analysis, visualization, and interaction methods.

Visualization is the communication of data through the use of interactive interfaces and has three major goals: (i) presentation to efficiently and effectively communicate the results of an analysis, (ii) confirmatory analysis as a goal-oriented examination of hypotheses, and (iii) exploratory data analysis as an interactive and usually undirected search for structures and trends.

Visual analytics is more than only visualization. It can rather be seen as an integral approach combining visualization, human factors, and data analysis. Visualization and visual analytics both integrate methodology from information analytics, geospatial analytics, and scientific analytics. Especially human factors (e.g., interaction, cognition, perception, collaboration, presentation, and dissemination) play a key role in the communication between human and computer, as well as in the decision-making process. In matters of data analysis, visual analytics furthermore profits from methodologies developed in the fields of statistical analytics, data management, knowledge representation, and knowledge discovery. Note that visual analytics is not likely to become a separate field of study, but its influence will spread over the research areas it comprises [9].

Overlooking a large information space is a typical visual analytics problem. In many cases, the information at hand is conflicting and needs to be integrated from heterogeneous data sources. Often the computer system lacks knowledge that is still hidden in the expert's mind. By applying analytical reasoning, hypotheses about the data can be either affirmed or discarded and eventually lead to a better understanding of the data. Visualization is used to explore the information space when automatic methods fail and to efficiently communicate results. Thereby human background knowledge, intuition, and decision-making either cannot be automated or serve as input for the future development of automated processes. In contrast to this, a well-defined problem where the optimum or a good estimation can be calculated by non-interactive analytical means would rather not be described as a visual analytics problem. In such a scenario, the non-interactive analysis should be clearly preferred due to efficiency reasons. Likewise, visualization problems not involving methods for automatic data analysis do not fall into the field of visual analytics.

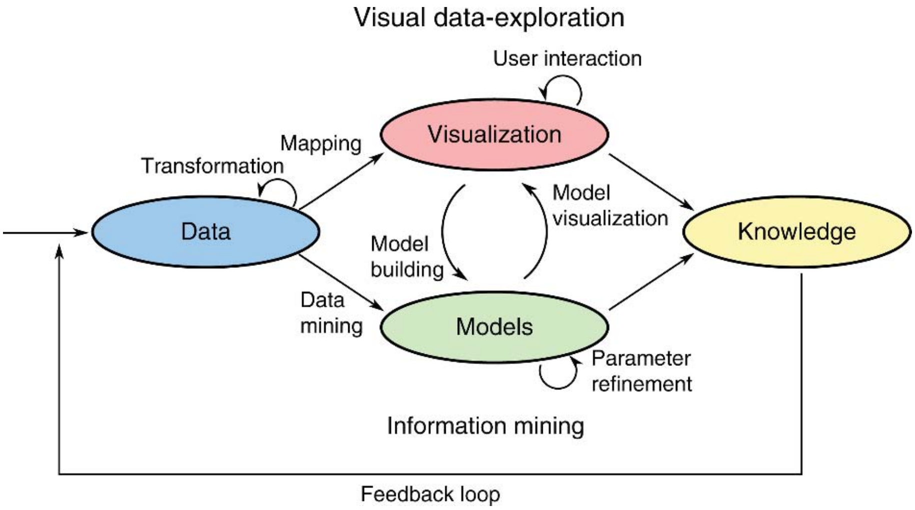
Visual Analytics Process

The visual analytics process is a combination of automatic and visual analysis methods with a tight coupling through human interaction in order to gain knowledge from data. Figure 2 shows an abstract overview of the different stages (represented through ovals) and their transitions (arrows) in the visual analytics process.

In many visual analytics scenarios, heterogeneous data sources need to be integrated before visual or automatic analysis methods can be applied. Therefore, the first step is often to preprocess and transform the data in order to extract meaningful units of data for further processing. Typical preprocessing tasks are data cleaning, normalization, grouping, or integration of heterogeneous data into a common schema.

Continuing with this meaningful data, the analyst can select between visual or automatic analysis methods. After mapping the data the analyst may obtain the desired knowledge directly, but more likely is the case that an initial visualization is not sufficient for the analysis. User interaction with the visualization is needed to reveal insightful information, for instance by zooming in different data areas or by considering different visual views on the data. In contrast to traditional information visualization, findings from the visualization can be reused to build a model for automatic analysis. As a matter of course these models can also be built from the original data using data mining methods. Once a model is created the analyst has the ability to interact with the automatic methods by modifying parameters or selecting other types of analysis algorithms. Model visualization can then be used to verify the findings of these models. Alternating between visual and automatic methods is characteristic for the visual analytics process and leads to a continuous refinement and verification of preliminary results. Misleading results in an intermediate step can thus be discovered at an early stage, which leads to more confidence in the final results.

In the visual analytics process, knowledge can be gained from visualization, automatic analysis, as well as the preceding interactions between visualizations, models, and the human analysts. The feedback loop



Visual Analytics. Figure 2. The *Visual Analytics Process* is characterized through interaction between data, visualizations, models about the data, and the users in order to discover knowledge.

stores this knowledge of insightful analyses in the system, and contributes to enable the analyst to draw faster and better conclusions in the future.

Technical and Social Challenges

While visual analytics profits from the increasing computational power of computer systems, faster networks, high-resolution displays, as well as novel interaction devices, it must be kept in mind that new technologies are always accompanied with a variety of technical challenges that have to be solved.

Dynamic processes in scientific or business applications often generate large streams of real-time data, such as sensor logs, web statistics, network traffic logs, or atmospheric and meteorological data. The analysis of such *large data streams* which can consist of terabytes or petabytes of data is one of the technical challenges since advances in many areas of science and technology are dependent upon the capability to analyze these data streams. As the sheer amount of data does often not allow to store all data at full detail, effective compression and feature extraction methods are needed to manage the data. Visual analytics aims at providing techniques that make humans capable of analyzing real time data streams by presenting results in a meaningful and intuitive way while allowing interaction with the data. These techniques enable quick identification of important information and timely reaction on critical process states or alarming incidents.

Synthesis of heterogeneous data sources is another challenge that is closely related to data streams, because real-world applications often access information from a large number of different information sources including collections of vector data, strings and text documents, graphs or sets of objects. Integrating these data sources includes many fundamental problems in statistics, machine learning, decision theory, and information theory. Therefore, the focus on scalable and robust methods for fusing complex and heterogeneous data sources is key to a more effective analysis process.

One step further in the analysis process, *interpretability and trustworthiness* or the ability to recognize and understand the data can be seen as one of the biggest challenges in visual analytics. Generating a visually correct output from raw data and drawing the right conclusions largely depends upon the quality of the used data and methods. A lot of possible quality problems (e.g., data capture errors, noise, outliers, low precision, missing values, coverage errors, double counts) can already be contained in the raw data. Furthermore, pre-processing of data in order to use it for visual analysis bears many potential quality problems (i.e., data migration and parsing, data cleaning, data reduction, data enrichment, up-/down-sampling, rounding and weighting, aggregation and combination). The concrete challenges are on the one hand to determine and to minimize these errors on the pre-processing side, and a flexible yet stable design of the visual analytics applications to cope with data quality problems on the other hand. From a technical point of view, the design of such applications should either be insensitive to data quality issues through usage of data cleaning methods or explicitly visualize errors and uncertainty to raise awareness for data quality issues.

In many scenarios, interpreting the raw data only makes little or no sense at all if it cannot be embedded in context. Research on *semantics* may derive this context from meta data by capturing associations and complex relationships. Ontology-driven techniques and systems have already started to enable new semantic applications in a wide span of fields such as bioinformatics, web services, financial services, business intelligence, and national security. Research challenges thereby arise from the size of ontologies, content diversity, heterogeneity as well as from computation of complex queries and link analysis over ontology instances and meta data.

Scalability in general is a key challenge of visual analytics, as it determines the ability to process large datasets by means of computational overhead as well as appropriate rendering techniques. Often, the huge amount of data that has to be visualized exceeds the limited amount of pixels of a display by several orders of magnitude. In order to cope with such a problem not only the absolute data growth and hardware performance have to be compared, but also the software and the algorithms to bring this data in an appropriate way onto the screen. As the amount of data is continuously growing and the amount of pixels on the display remains rather constant, the rate of compression on the display is steadily increasing. Therefore, more and more details get lost. It is an essential task of visual analytics to create a higher-level view onto the dataset, while maximizing the amount of details at the same time.

The field of *problem solving, decision science, and human information discourse* constitutes a further visual analytics challenge since it not only requires understanding of technology, but also comprehension of typical human capabilities such as logic, reasoning, and common sense. Many psychological studies about the process of problem solving have been conducted. In a usual test setup the subjects have to solve a well-defined problem where the optimal solution is known to the researchers. However, real-world problems are manifold. In many cases these problems are intransparent, consist of conflicting goals, and are complex in terms of large numbers of items, interrelations, and decisions involved. The dynamics of information that changes over time should not be underestimated since it might have a strong impact on the right decision. Furthermore, social aspects such as decision making in groups make the process even more delicate.

While many novel visualization techniques have been proposed, their wide-spread usage has not taken place primarily due to the users' refusal to change their daily working routines. *User acceptance* is therefore a further visual analytics challenge, since the advantages of developed tools need to be properly communicated to the audience of future users to overcome usage barriers, and to tap the full potential of the visual analytics approach.

Visual analytics tools and techniques should not stand alone, but should *integrate* seamlessly into the applications of diverse domains, and allow interaction with existing systems. Although many visual analytics tools are very specific (i.e., in astronomy or nuclear science) and therefore rather unique, in many domains (e.g., business or network security applications) integration into existing systems would significantly promote their usage by a wider community.

Finally, *evaluation* as a systematic analysis of usability, worth, and significance of a system is crucial to the success of visual analytics science and technology. During the evaluation of a system, different aspects can be considered such as functional testing, performance benchmarks, measurement of the effectiveness of the display in user studies, assessment of its impact on decision-making, or economic success to name just a few. Development of abstract design guidelines for visual analytics applications would constitute a great contribution.

Key Applications

Visual analytics is essential in application areas where large information spaces have to be processed and analyzed. Major application fields are *physics* and *astronomy*. Especially the field of *astrophysics* offers many opportunities for visual analytics techniques: Massive volumes of unstructured data, originating from different directions of space and covering the whole frequency spectrum, form continuous streams of terabytes of data that can be recorded and analyzed. With common data analysis techniques, astronomers can separate relevant data from noise, analyze similarities or complex patterns, and gain useful knowledge about the universe, but the visual analytics approach can significantly support the process of identifying unexpected phenomena inside the massive and dynamic data streams that would otherwise not be found by standard algorithmic means.

Monitoring *climate* and *weather* is also a domain which involves huge amounts of data, collected by sensors throughout the world and from satellites in short time intervals. A visual approach can help to interpret these massive amounts of data and to gain insight into the dependencies of climate factors and climate change scenarios, which would otherwise not be easily identified. Besides weather forecasts, existing applications visualize the global warming, melting of the poles, the stratospheric ozone depletion, as well as hurricane and tsunami warnings.

In the domain of *emergency management*, visual analytics can help determine the on-going progress of an emergency and identify the next countermeasures (e.g., construction of physical countermeasures or evacuation of the population) that must be taken to limit the damage. Such scenarios can include natural or meteorological catastrophes like flood or waves, volcanoes, storm, fire or epidemic growth of diseases (e.g., bird flu), but also human-made technological catastrophes like industrial accidents, transport accidents or pollution.

Visual analytics for *security* and *geographics* is an important research topic. The application field in this sector is wide, ranging from terrorism informatics, border protection, path detection to network security. Visual analytics supports investigation and detection of similarities and anomalies in large data sets, like flight customer data, GPS tracking or IP traffic data.

In *biology* and *medicine*, computer tomography, and ultrasound imaging for three-dimensional digital reconstruction and visualization produce gigabytes of medical data and have been widely used for years. The application area of bio-informatics uses visual analytics techniques to analyze large amounts of biological data.

From the early beginning of sequencing, scientists in these areas face unprecedented volumes of data, like in the Human Genome Project with three billion base pairs per human. Other new areas like Proteomics (studies of the proteins in a cell), Metabolomics (systematic study of unique chemical fingerprints that specific cellular processes leave behind) or combinatorial chemistry with tens of millions of compounds even enlarge the amount of data every day. A brute-force computation of all possible combinations is often not possible, but interactive visual approaches can help to identify the main regions of interest and exclude areas that are not promising.

Another major application domain for visual analytics is *business intelligence*. The financial market, with its hundreds of thousands of assets, generates large amounts of data every day, which accumulate to extremely high data volumes throughout the years. The main challenge in this area is to analyze the data under multiple perspectives and assumptions to understand historical and current situations, and then monitoring the market to forecast trends or to identify recurring situations. Other key applications in this area are fraud detection, detection of money laundering, or the analysis of customer data, insurance data, social data, and health care services.

Cross-references

▶ Cluster Visualization
▶ Comparative Visualization
▶ Data Mining
▶ Data Visualization
▶ Human-Computer Interaction
▶ Multidimensional Visualization Methods
▶ Multivariate Visualization Methods
▶ Parallel Visualization
▶ Scientific Visualization
▶ Visual Classification
▶ Visual Clustering
▶ Visual Content Analysis
▶ Visual Data Mining
▶ Visual Metaphor
▶ Visual On-Line Analytical Processing (OLAP)
▶ Visualization for Information Retrieval

Recommended Reading

1. Card S.W., Mackinlay J.D., and Shneiderman B. (eds.) Readings in Information Visualization: Using Vision to Think. Morgan Kaufmann, San Francisco, CA, USA, 1999.
2. Chen C. Information Visualization – Beyond the Horizon. Springer, Berlin, 2nd edn., 2004.
3. Keim D.A. Visual exploration of large data sets. Commun. ACM, 44(8):38–44, 2001.
4. Keim D.A. and Thomas J. Scope and challenges of visual analytics. Tutorial at IEEE Visualization Conf., 2007.
5. Spence R. Information Visualization – Design for Interaction. Pearson Education Limited, Harlow, England, 2nd edn., 2006.
6. Thomas J. and Cook K. Illuminating the path: Research and development agenda for visual analytics. IEEE-Press, Los Alamitos, CA, USA, 2005.
7. Tukey J.W. Exploratory data analysis. Addison-Wesley, Reading, MA, USA, 1977.
8. Ware C. Information Visualization – Perception for Design. Morgan Kaufmann, San Francisco, CA, USA, 1st edn., 2000.
9. Wong P.C. and Thomas J. Visual Analytics – Guest Editors' Introduction. IEEE Trans. Comput. Graph. Appl., 24(5):20–21, 2004.

Visual Association Rules

Li Yang
Western Michigan University, Kalamazoo, MI, USA

Synonyms

Association rule visualization

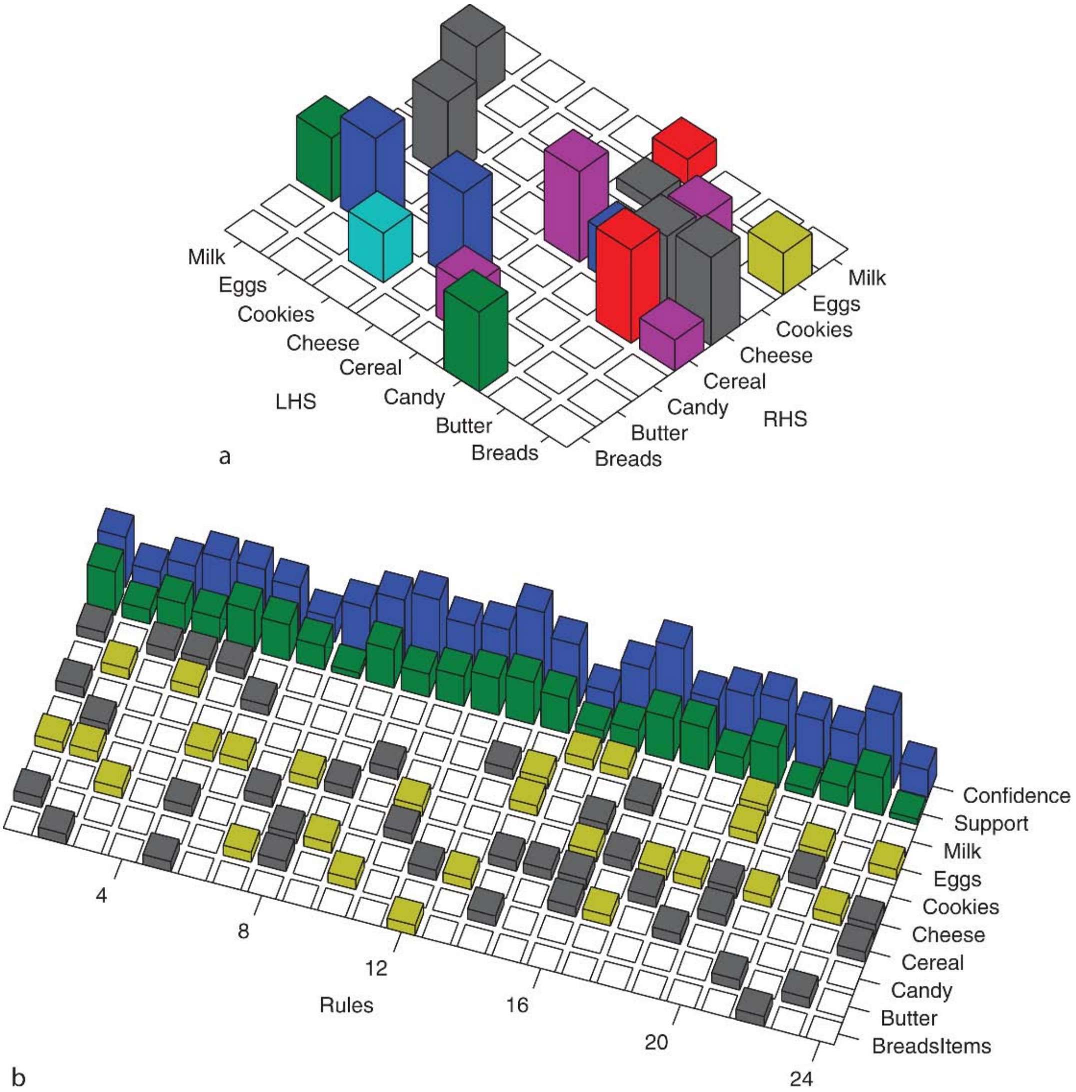
Definition

Association rule mining finds frequent associations between sets of data items from a large number of transactions. In market basket analysis, a typical association rule reads: *80% of transactions that buy diapers and milk also buy beer. The rule is supported by 10% of all transactions*. In this example, the 10% and 80% are called support and confidence, respectively. Depending on the user-specified minimum support and minimum confidence, association rule mining often produces too many rules for humans to read over. The answer to this problem is to select the most interesting rules. As interestingness is a subjective measure, selecting the most interesting rules is inherently human being's work. It is expected that information visualization may play an important role in managing a large number of association rules, and in identifying the most interesting ones.

Historical Background

An association rule reflects a many-to-many relationship. In lacking of an effective visual metaphor to display many-to-many relationships, information visualization has received a technical challenge in order to display and interact with many association rules. Most existing approaches have been designed to visualize association rules in restricted forms. The restriction can be on the number of items on each side of the rule, the type of the rule, the total number of items, or the total number of rules. These existing approaches can be classified into two categories: matrix-based approaches and graph-based approaches.

A rectangular matrix can be used to visualize one-to-one association rules, where the left-hand side (LHS) items of all rules are listed along one axis and the right-hand side (RHS) of all rules are listed along the other axis of the rectangular matrix. A rule could be shown as an icon in the corresponding cell. Figure 1a shows an example 3D visualization, where the height and color of each bar represent support and confidence values of the corresponding rule, respectively. The major restriction of this approach is that it allows only one item on each side of the rule. Wong et al. [9] gave an alternative approach of arranging axes, where one axis is used to list items and the other is used to list rules. A rule is then visualized by a 3D bar chart against all items in the rule. This approach is able to visualize many-to-many rules, as long as the number of rules is kept reasonably small. Figure 1b illustrates the approach where the color of each bar indicates whether the



Visual Association Rules. Figure 1. Two major matrix-based approaches: (a) LHS versus RHS and (b) Rule versus Item.

item is on the LHS or RHS of the rule. Support and confidence values of the rules are visualized alongside the bar charts. Another matrix-based approach is to use mosaic plots and double deck plots [4] to visualize the contingency table of a frequent itemset that gives rise to a many-to-one association rule. However, only many-to-one rules derived from a single frequent itemset can be visualized each time. Fukuda et al. [2] gave a matrix-based approach of visualizing numeric association rules, which contain two numeric attributes on the LHS and a Boolean attribute on the RHS. The approach displays the set of rules as a bitmap image.

Directed graph is another technique to depict associations among items. There are two alternatives to assign graph nodes:

1. Each node represents an itemset. An association rule is visualized as an edge from the node representing its LHS itemset to the node representing its RHS itemset. Such an approach has been used in IBM DB2 Intelligent Miner. A major problem is that there may easily be too many nodes to display.
2. Each node represents an item. An association rule is visualized as a bunch of edges from LHS items to an intermediate node and another bunch of edges from the intermediate node to RHS items. Such an approach may work well only when a few items and rules are involved. The graph can quickly turn into a tangled mess with as few as a dozen rules. Klemettinen et al. [5] introduced a rule visualizer that uses this approach.

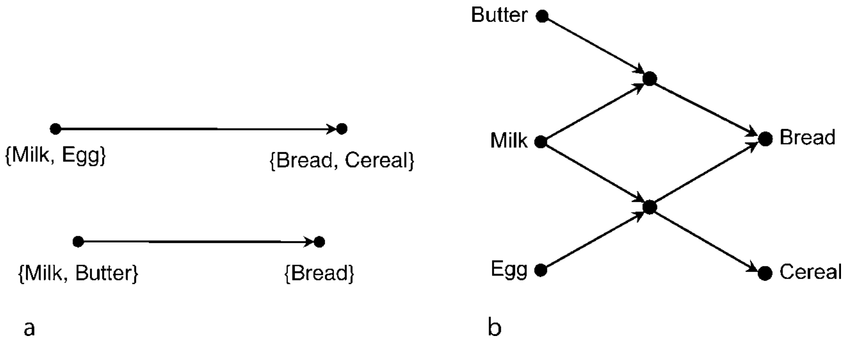
Figure 2 illustrates the two alternatives of visualizing association rules as directed graphs.

Visualization of association rules can be found in commercial data mining packages. SAS Enterprise Miner and SGI MineSet use matrix-based approaches and allow the user to visualize one-to-one association rules. IBM DB2 Intelligent Miner has a Associations Visualizer that uses the directed graph approach where each graph node displays an itemset.

Foundations

Association rule mining [1,3] is one of the mostly researched areas in data mining. Let $I = \{i_1, i_2,...,i_k\}$ be a set of items. A subset $A \subseteq I$ of items is called an itemset. Input data to association rule mining are n subsets $\{T_1, T_2,...,T_n\}$, called transactions, of I. Transaction T_i supports an itemset A if $A \subseteq T_i$. The percentage $P(A)$ of transactions that support A is called the support of A. A *frequent itemset* is an itemset whose support is no less than a minimum value specified by the user. An *association rule* is an expression $A \rightarrow B$ where A and B are itemsets and $A \cap B = \emptyset$. $P(A \cup B)$ is called the *support* of the rule. $P(A \cup B)/P(A)$ is called the *confidence* of the rule. The problem of association rule mining is to find all association rules whose supports are no less than a minimum support value and whose confidences are no less than a minimum confidence value. Frequent itemsets hold a well-known Apriori property: subsets of a frequent itemset are frequent. Efficient association rule mining algorithms [1] have been developed using the Apriori property for early pruning of search space.

The problem of too many discovered rules has also been studied. Klemettinen et al. [5] studied association rules at the border of frequent itemsets and used pattern templates to specify what the user wants to see. Testing criteria such as maximum entropy and



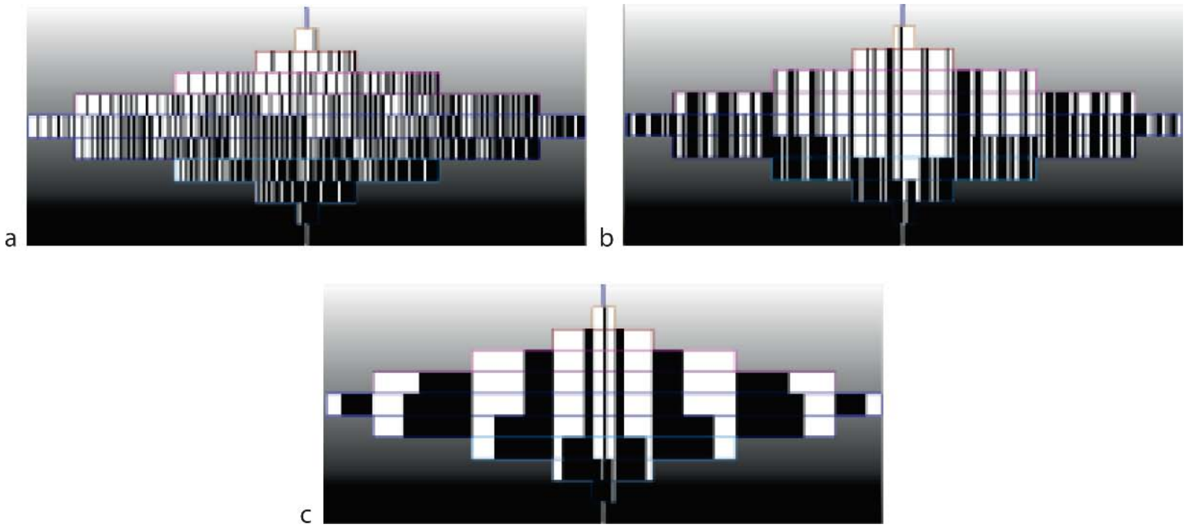
Visual Association Rules. Figure 2. Two major graph-based approaches: (a) Itemset as node and (b) Item as node.

chi-square(χ^2) have been suggested to replace the confidence test for the purpose of finding the most interesting rules. Liu et al. [7] have given a way of pruning and visualizing association rules with the presence of item taxonomy, by allowing a user to specify knowledge in terms of rules. Unexpected rules (that do not conform to the user's knowledge) are selected and visualized in a structured way.

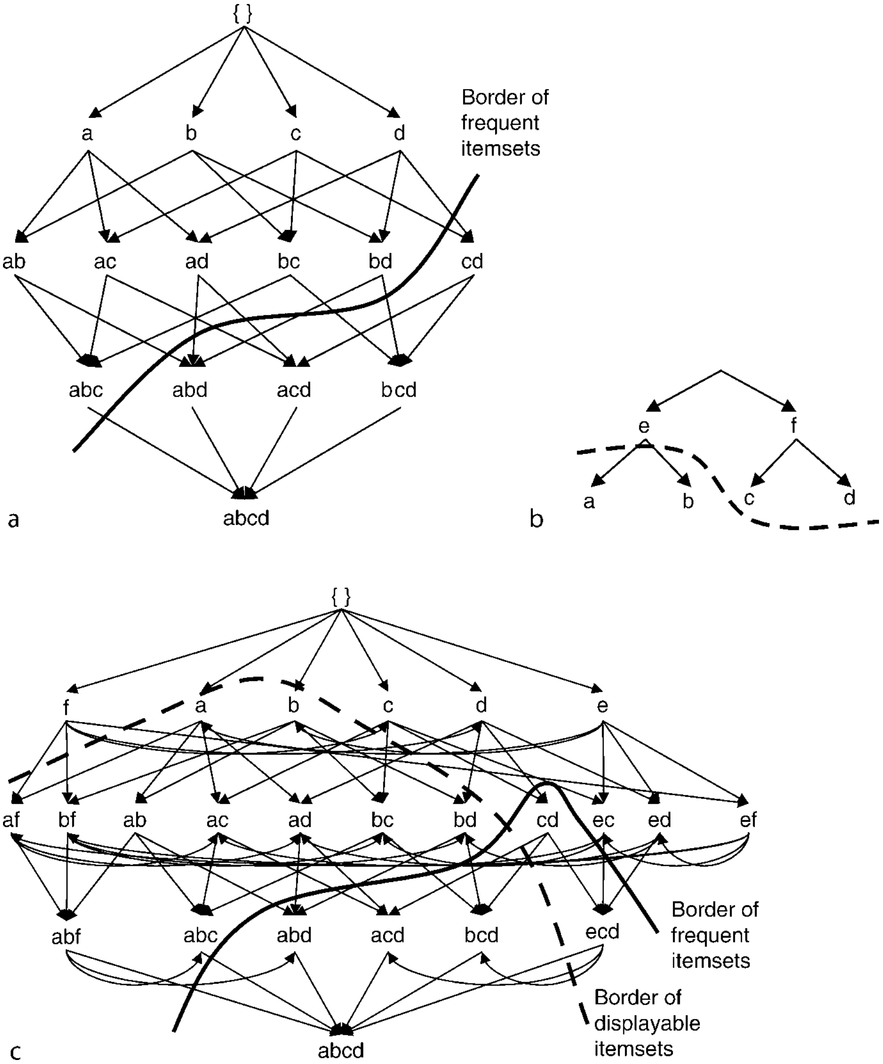
Association rules are difficult to visualize for several reasons. First, an association rule reflects a many-to-many relationship and there is no effective visual metaphor to display many many-to-one, never to say many-to-many, relationships. Second, frequent itemsets and association rules have inherent closure properties. For example, any subset of a frequent itemset is also frequent; if $a \rightarrow bc$ is a valid association rule, then $a \rightarrow b$, $a \rightarrow c$, $ab \rightarrow c$ and $ac \rightarrow b$ are all valid association rules. Third, the challenge is to visualize many such itemsets or rules. With the absence of an effective visual metaphor to present many-to-many relationships and to accommodate the closure properties, frequent itemsets and association rules pose fundamental challenges to information visualization.

Taking a function-theoretic view, the Apriori property basically says that frequent itemsets define an anti-monotone Boolean function $f(i_1,i_2,...,i_k)$ where $i_1,i_2,...,i_k$ are Boolean variables (items). As the function is anti-monotone, there is a border which separates 1's from 0's. Visualization of monotone Boolean functions has been studied by Kovalerchuk and Delizy in [6], although association rules are not their concern. Boolean vectors are displayed in multiple disk form where Boolean vectors of the same norm (equivalently, itemsets with the same number of items) are placed on the same disk. Figure 3 shows visualizations of an anti-monotone Boolean function $f(i_1,i_2,...,i_{10}) = \neg\, i_1$. Such a function is equivalent to frequent itemsets on $\{i_1,i_2,...,i_{10}\}$ where $\{i_2...i_{10}\}$ is a frequent itemset and $\{i_1\}$ is infrequent. In Fig. 3a, Boolean vectors are arranged on each disk in their numeric order. The visualization does not permit any real understanding of the border of frequent itemsets. Fig. 3b rearranges Boolean vectors on each disk so that all vectors on a Hansel chain are aligned vertically. A chain is a sequence of Boolean vectors such that each vector produces the next vector by changing a "0" element to "1," and Hansel chains provide a way to non-repeatedly visit all vectors. As the function is monotone on each chain, Fig. 3b does show a border, although the border is highly zigzagged. Figure 3c rearranges Hansel chains according to the position of the first "1" value within each chain and shows the border in a clearer way.

Frequent itemsets and association rules are often presented in a set-theoretic view. Given a set $I = \{i_1, i_2,...,i_k\}$ of items, its power set $\mathcal{P}(I)$ forms a lattice $<\mathcal{P}(I), \subseteq>$ where the subset relationship $\subseteq$ specifies the partial order. For an example set of items, $I = \{a, b, c, d\}$, this lattice can be illustrated in Fig. 4a. Frequent



Visual Association Rules. Figure 3. Visualizing the anti-monotone Boolean function $f(i_1,i_2,...,i_{10}) = \neg\, i_1$: (Courtesy of [6].)

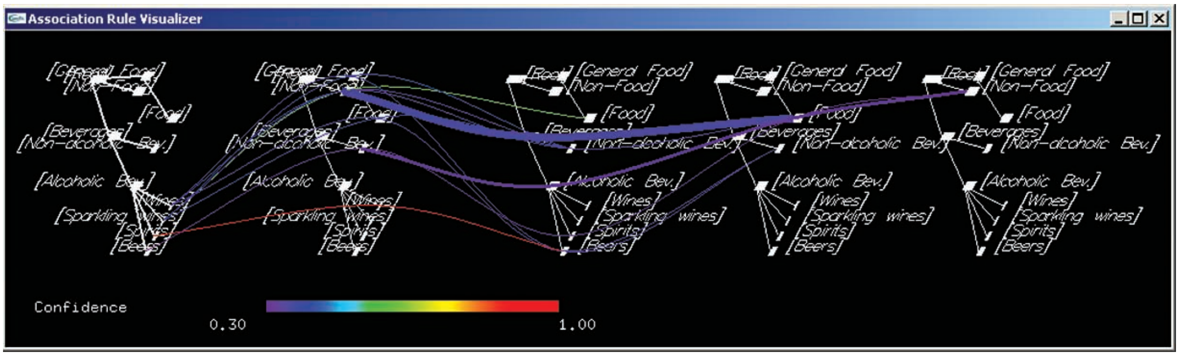


Visual Association Rules. Figure 4. Itemset lattice, item taxonomy tree, and generalized itemset lattice on $I = \{a, b, c, d\}$: (a) Itemset lattice, (b) An item taxonomy tree, and (c) Generalized itemset lattice. (Courtesy of [10].)

itemsets define a border on the lattice: if an itemset is frequent, so is every subset of it; if an itemset is infrequent, so is every superset of it. The objective of frequent itemset visualization is to visualize the border. In fact, this has been tried in the three visualizations in Fig. 3, which have visualized the whole lattice structure. Each disk in Fig. 3 displays $\binom{k}{l}$ itemsets where l is the disk level. Clearly, such an approach would fail when k is large.

In order to deal with the problem of long border of frequent itemsets, Yang has developed an approach [10] for visualizing generalized frequent itemsets and association rules by introducing user interaction with the border. Generalized association rules come from the introduction of item taxonomy. As shown in Fig. 4b, an item taxonomy is a directed tree whose leaf nodes are items and whose non-leaf nodes are item categories. Mining generalized association rules across item taxonomy was studied in [8]. Item taxonomy introduces another dimension of closure property. For example, an ancestor itemset of a frequent itemset is also frequent.

The presence of item taxonomy extends an itemset lattice to a generalized itemset lattice. The user can choose which items or item categories to display in an item taxonomy tree. The partially displayed item taxonomy tree induces a border of displayable itemsets in the generalized itemset lattice. For example, if the



Visual Association Rules. Figure 5. Visualizing association rules with item taxonomy. (Courtesy of [10].)

items *c, d, e, f* in Fig. 4b are displayed, this induces a border of displayable itemsets shown in Fig. 4c. Only non-redundant displayable frequent itemsets, for example, *ec* and *ed* in Fig. 4c, are visualized. In association rule visualization, only non-redundant rules derived from frequent displayable itemsets are visualized.

The non-redundant displayable frequent itemsets and association rules are visualized in 3D through parallel coordinates, where each parallel coordinate is replaced by a standing visualization of item taxonomy tree. An itemset or rule is visualized as a Bézier curve connecting all items in it. Parameters such as support and confidence can be mapped to graphical features such as width or color of the curve. Figure 5 gives a screen snapshot in the interactive visualization of many-to-many association rules, where association rules are aligned according to where the RHSs separate from the LHSs. In Fig. 5, the left two coordinates represent the LHSs of the rules and the right three coordinates represent the RHSs of the rules. The displayed item taxonomy tree can be expanded or shrunk by user interaction, and each change triggers a new set of itemsets or rules to be displayed.

Key Applications

As a standard feature in many data mining software packages, association rule visualization has been used widely in business intelligence and scientific research.

Future Directions

Although approaches have been developed to visualize association rules, none of them has gained predominant acceptance and can simultaneously manage a large number of rules with multiple items on both sides. Visual association rule research has many open problems. An obvious one is how to prune uninteresting itemsets and rules, a topic that has bothered researchers for years. Specific to visualization, open problems include how to visualize a large number of many-to-many rules and how to associate closure properties of itemsets or rules with intuitive visual presentations. Data visualization on the lattice structure has potential applications beyond visual association rules. For example, it can be used in the visualization of iceberg data cubes in data warehousing.

Cross-references

- ▶ Association Rules
- ▶ Data Visualization
- ▶ Visual Analytics
- ▶ Visual Data Mining

Recommended Reading

1. Agrawal R. and Srikant R. Fast algorithms for mining association rules. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 207–216.
2. Fukuda T., Morimoto Y., Morishita S., and Tokuyama T. Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 13–23.
3. Han J. and Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, USA, 2nd edn., 2005.
4. Hofmann H., Siebes A., and Wilhelm A. Visualizing association rules with interactive mosaic plots. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 227–235.
5. Klemettinen M., Mannila H., Ronkainen P., Toivonen H., and Verkamo I. Finding interesting rules from large sets of discovered association rules. In Proc. Int. Conf. on Information and Knowledge Management, 1994, pp. 401–407.
6. Kovalerchuk B. and Delizy F. Visual data mining using monotone Boolean functions. In Visual and Spatial

Analysis: Advances in Data Mining, Reasoning, and Problem Solving, B. Kovalerchuk, J. Schwing (eds.). Springer, Berlin, 2004, pp. 387–406.

7. Liu B., Hsu W., Wang K., and Chen S. Visually aided exploration of interesting association rules. In Advances in Knowledge Discovery and Data Mining, 3rd Pacific-Asia Conf., 1999, pp. 380–389.
8. Srikant R. and Agrawal R. Mining generalized association rules. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 407–419.
9. Wong P.C., Whitney P., and Thomas J. Visualizing association rules for text mining. In Proc. IEEE Symp. on Information Visualization, 1999, pp. 120–123.
10. Yang L. Pruning and visualizing generalized association rules in parallel coordinates. IEEE Trans. Knowl. and Data Eng., 17 (1):60–70, January 2005.

Visual Classification

Mihael Ankerst
Allianz, Munich, Germany

Synonyms

Cooperative classification

Definition

Decision trees have been successfully used for the task of classification. However, state-of the-art algorithms do not incorporate the user in the tree construction process. Through the involvement of the user in the process of classification, he/she can provide domain knowledge to focus the search of the algorithm and gain a deeper understanding of the resulting decision tree. In a cooperative approach, both the user and the computer contribute what they do best: the user specifies the task, focuses the search using his/her domain knowledge and evaluates the (intermediate) results of the algorithm. The computer, on the other hand, automatically creates patterns satisfying the specified user constraints. The cooperative approach is based on a novel visualization technique for multi-dimensional data representing their impurity with respect to their class labels.

Historical Background

The idea of visual classification has been built upon recent progress in the area of information visualization and decision trees.

Many different algorithms for learning decision trees have been developed over the last 20 years. For instance, *CART* [1] was one of the earliest systems which, in particular, incorporates an effective pruning phase. Their so-called minimum cost complexity pruning cuts off branches that have a large number of leaf nodes, yielding just a small reduction of the apparent error. *SLIQ* [2] is a scalable decision tree classifier that can handle both numerical and categorical attributes. In order to make SLIQ scalable for large training sets, special data structures called attribute lists are introduced, which avoid sorting the numerical attributes for each selection of the next split. Furthermore, a greedy algorithm for efficient selection of splits of categorical attributes is presented. An experimental evaluation demonstrates that SLIQ produces decision trees with state-of-the-art accuracy, and tree size with a much better efficiency for large training sets.

Visual representation of data as a basis for the human–computer interface has evolved rapidly in recent years. The increasing amount of available digital information in all kinds of applications has led to the challenge of dealing with both high dimensionality and large amounts of data. [3] gives a comprehensive overview over existing *visualization techniques* for large amounts of multidimensional data, which have no standard mapping into the Cartesian coordinate system.

Foundations

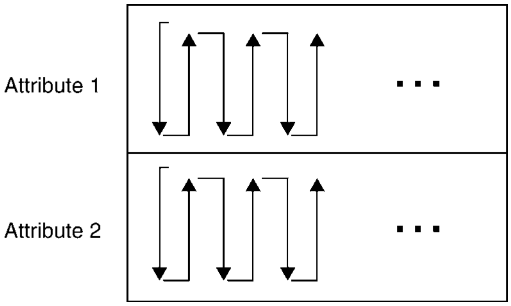
A decision tree classifier constructs a tree in a top-down fashion, performing a greedy search through the very large space of all possible decision trees. At each current node, the attribute that is most useful for the task of classification (with respect to the subset of all training objects having passed all tests on the path to the current node) is selected. Criteria such as the information gain or the gini index have been used to measure the usefulness of an attribute. Domain knowledge about the semantics of the attributes and the attribute values is not considered by the criteria. Note that greedy algorithms for decision tree construction do not allow to backtrack to a previous choice of an attribute when it finally turns out to be suboptimal.

V*isual classification* [4,5] is a user-centered approach to decision tree construction where the user and the computer can both contribute their strengths. The user provides domain knowledge and evaluates intermediate results of the algorithm, the computer automatically creates patterns satisfying user constraints and generates appropriate visualizations of these patterns. In this cooperative approach, domain knowledge of the user can direct the search of the algorithm. Additionally, by providing adequate data and knowledge visualizations, the pattern recognition capabilities of the human can be used to increase the effectivity of decision tree construction. Furthermore, the user gets a deeper understanding of the decision tree than just obtaining it as a result of an algorithm.

The three components of visual classification is visualizing the data, visualizing the decision tree and the integration of algorithms into decision tree construction.

Visualizing the data is done by the pixel-oriented bar technique. The bar visualization technique is performed as follows. Within a bar, the sorted attribute values are mapped to pixels in a line-by-line fashion according to their order. Each attribute is visualized independently from the other attributes in a separate bar. Figure 1 illustrates the method of the bar visualization for the case of two attributes. The amount of training data that can be visualized by the bar technique is determined by the product of the number of data records and the number of attributes.

Visualizing the decision tree is done by a visualization technique, such that each node is represented by the data visualization of the chosen splitting attribute of that node. For each level of the tree a bar is drawn representing all nodes of this level. The top level bar corresponds to the root node of the decision tree. On lower levels of the tree the number of records and thus the number of pixels is reduced if there are leaves in upper levels – leaves are underlined with a black line. Black vertical lines indicate the split points set in the current bar. On lower levels, partitions of the data inherited from upper levels are marked by white vertical lines at the same horizontal position as the original split point. Attribute and node information at the mouse pointer position (attribute name, attribute value, min., max. value and number of records in this node) is displayed on demand. Upon a mouse click the system switches back to the data visualization of the corresponding node.



Visual Classification. Figure 1. Illustration of bar visualization.

Compared to a standard visualization of a decision tree, a lot of additional information is provided which is very helpful in explaining and analyzing the decision tree:

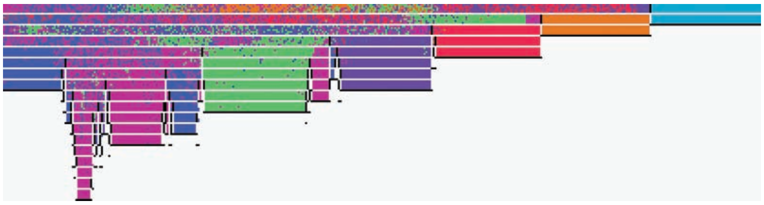
- Size of the node (number of training records corresponding to the node)
- Quality of the split (purity of the resulting partitions)
- Class distribution (frequency and location of the training instances of all classes)

Figure 2 illustrates the visualization of a decision tree for the Segment training data from the Statlog benchmark [6] having 19 numerical attributes.

The integration of algorithms into decision tree construction is solved in the following way.

Propose Split

For a set of attributes selected by the user, the attribute with the best split together with the optimum split point of this attribute is calculated and visualized. If a singleton attribute is specified as input, only the optimum split point for this attribute is determined. The function *propose split* turns out to be useful in two



Visual Classification. Figure 2. Illustration of knowledeg visualization.

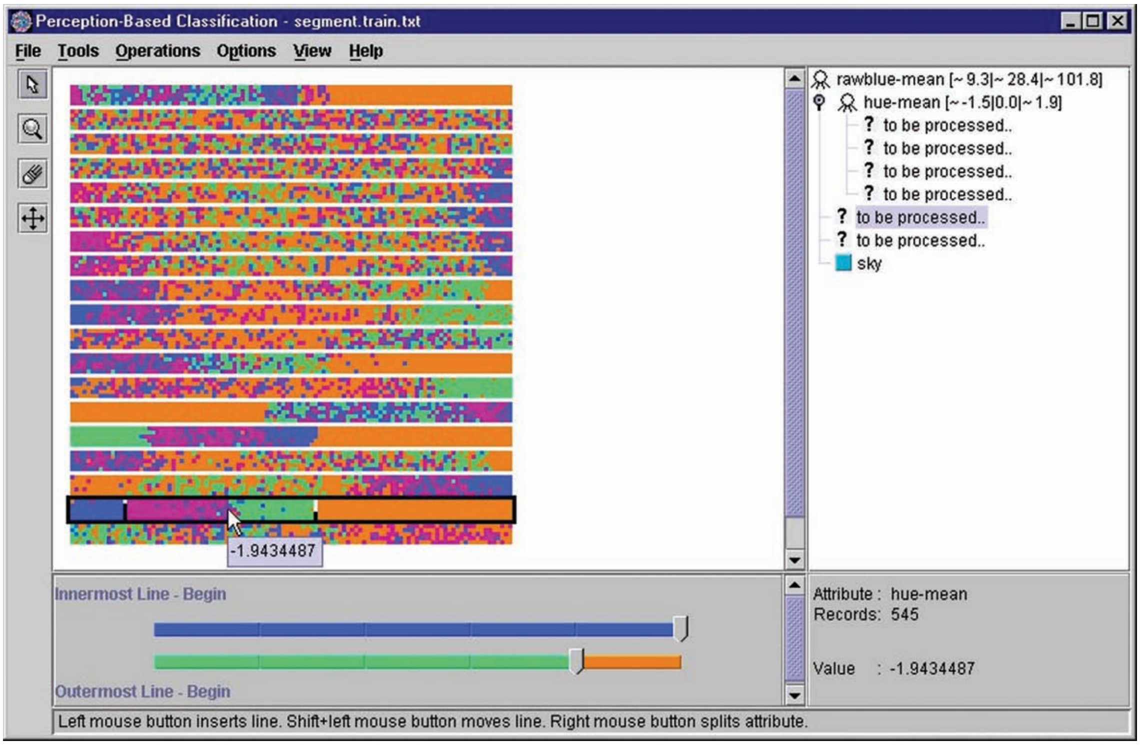
cases: first, whenever there are several candidate attributes with very similar class distributions and, second, when none of the attributes yields a good split which can be perceived from the visualization.

Look-Ahead

For some hypothetical split of the active node of the decision tree, the subtree of a given maximum depth is calculated and visualized with the new visualization technique for decision trees. This function offers a view on the hypothetical expansion up to a user specified number of levels, or until a user specified minimum number of records per node. If the lookahead function is invoked for a limited number of levels, it is very fast (some seconds of runtime) because no pruning is performed in this case. The *look-ahead* function may provide valuable insights for selecting the next split attribute. Without the look-ahead function, when there are several candidate attributes the user selects one of them, chooses one or several split points and continues the tree construction. If at a later stage the expanded branch does not yield a satisfying subtree, the user will backtrack to the root node of this branch and will remove the previously expanded subtree. He/she will select another candidate attribute, split it and proceed. Utilizing the new function, however, the user requests a look-ahead for each candidate attribute before actually performing a split. Thus, the necessity of backtracking may be avoided.

Expand Subtree

For the active node of the decision tree, the algorithm automatically expands the tree. Several parameters may be provided to restrict the algorithmic expansion such as the maximum number of levels and the minimum number of data records or the minimum purity per leaf node. The pruning of automatically created subtrees rises some questions. Should one only prune the subtree created automatically or prune the whole tree – including the subtrees created manually? According to the paradigm of the user as a supervisor, pruning is only applied to automatically created trees. It is important to distinguish two different uses of the *expand subtree* function: the maximum number of levels is either specified or it is unspecified. In the first case, the decision tree is usually post-processed by the user. No pruning is performed if a maximum number of levels is specified. Otherwise, if no maximum



rawblue-mean [~9.3|~28.4|~101.8]
hue-mean [~-1.5|0.0|~1.9]
? to be processed..
? to be processed..
? to be processed..
? to be processed..
? to be processed..
? to be processed..
sky

Attribute : hue-mean
Records: 545

Value : -1.9434487

Left mouse button inserts line. Shift+left mouse button moves line. Right mouse button splits attribute.

Visual Classification. Figure 3. PBC system.

number of levels is specified, the user wants the system to complete this subtree for him and pruning of the automatically created subtree is performed if desired. The function *expand subtree* is useful in particular if the number of records of the active node is relatively small. Furthermore, this function can save a lot of user time because the manual creation of a subtree may take much more time than the automatic creation.

The visual classification approach is implemented in the PBC system, as depicted in Fig. 3.

Key Applications

The visual classification approach can be applied to any domain where decision tree classification is applicable. The value of the approach is the deeper understanding of the data and the decision tree.

Cross-references

► Decision Trees
► Pixel-Oriented Visualization Techniques

Recommended Reading

1. Ankerst M., Elsen C., Ester M., and Kriegel H.-P. Visual classification: an interactive approach to decision tree construction. In Proc. 5th Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 392–396.
2. Ankerst M., Ester M., and Kriegel H.P. Towards an effective cooperation of the computer and the user for classification. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000.
3. Breiman L., Friedman J.H., Olshen R.A., and Stone P.J. Classification and Reggression Trees. Wadsworth, Belmont, CA, 1984.
4. Keim D.A. Visual database exploration techniques. Tutorial at Int. Conf. on Knowledge Discovery and Data Mining, 1997.
5. Mehta M., Agrawal R., and Rissanen J. SLIQ: A Fast Scalable Classifier for Data Mining. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996.
6. Michie D., Spiegelhalter D.J., and Taylor C.C. Machine Learning, Neural and Statistical Classification. Ellis Horwood, 1994. See also http://www.ncc.up.pt/liacc/ML/statlog/datasets.html.

Visual Clustering

Mike Sips
Stanford University, Stanford, CA, USA

Synonyms

Visual mining; Visual data mining

Definition

Synthesis of computational methods and interactive visualization techniques that represents a clustering structure, defined in higher dimensions to the human analyst in order to support the human analyst to explore and refine the clustering structure of high dimensional data spaces based on his/her domain knowledge.

Historical Background

The advancements made in computing technology over the last two decades allow both scientific and business applications to produce large data sets with increasing complexity and dimensionality. Automated clustering algorithms are indispensable for analyzing large n-dimensional data sets but often fall short to provide completely satisfactory results in terms of quality, meaningfulness, and relevance of the revealed clusters. With the increasing graphics capabilities of the available computers, researchers realized that an integration of the human into the clustering process based on visual feedbacks helps to improve the effectiveness of automated clustering algorithms. A synthesis of automated clustering algorithm and visualization usually does not only yield better clustering results, but also a higher degree of user satisfaction and confidence in the findings.

Foundations

Clustering is one of the basic data analysis tasks. It is a process of organizing data into similar groups. Unlike classification, clustering is unsupervised learning. In particular, the classes are unknown and no training set with pre-computed class labels is available. A clustering algorithm in general can be seen as an external source that labels the data.

Automated clustering algorithms are indispensable in analyzing large n-dimensional data spaces, but they often fall short in computing completely satisfactory results. In general, two interconnected reasons can be observed. First, many clustering algorithms use assumptions about specific properties of clusters either as built-in defaults or as input parameter settings. Automated clustering algorithms are very sensitive to these input parameters. Analysts often get different clustering results even for slightly different parameter settings. Useful parameter settings are in general hard to determine a priority because humans have huge difficulties in understanding the impact of different parameter settings on the revealed clusters; especially in n-dimensional data spaces. Second, large n-D data spaces often have skewed distributions. The density, distribution and shape of the clusters are quite diverse in different subspaces. Skewed distributions are in general difficult to reveal using just one global parameter setting.

A variety of different visual clustering approaches have been proposed. Visual clustering can be classified into four common approaches, based on their mechanism to incorporate the data analyst into the clustering process. Several exploratory data analysis systems allow an interactive exploration of a given clustering structure based on projections. The aim is to make a given clustering structure in n-D interpretable for the human. A few visual clustering approaches are extensions of an optimized clustering algorithm with advanced visualization techniques. Other approaches extract the clustering structure based on the analyst's feedback. In these scenarios, clusters are characterized based on the analyst's specific domain knowledge. More recently, novel data analysis techniques from related disciplines such as machine learning or statistics supporting visual clustering have been proposed. An interesting approach uses visualization to support the selection of subspaces that contain strong clusters. Another interesting approach is Self-Organizing Maps.

Visual Exploration of a Given Clustering Structure

Many interactive systems use orthogonal standard projections to visualize the given clustering structure of an n-dimensional data space. Orthogonal standard projections are widely used in exploratory data analysis, because they are easy to understand. Unlike in unlabeled data, the data item's class labels are exploited in these approaches to find interesting projections of the clustering structure, i.e., projections that preserve properties of the clusters.

The n23Tool proposed by Yang [15] uses 3–D cluster-guided projections [5]. The idea of cluster-guided projections is to find a 3–D subspace that neatly separates four given clusters. A cluster-guided projection is based on the following observation. Any combination of four distinct and non co-linear cluster centers defines a unique 3-D subspace. Moreover, such a subspace neatly separates the four clusters, and in addition it also preservers the inter-cluster distance between any two of the cluster centers. To support an efficient exploration of clusters in n-dimensional data spaces, the n23 tool combines cluster-guided projections with Grand Tour [3]. A cluster-guided tour allows an analyst to quickly get an impression of all clusters by smoothly moving through the space of cluster-guided projections.

Koren and Carmel [11] propose an alternative measure of goodness to preserve the clustering structure in projections. The basic idea is to weight the distances between points differently depending on whether they have the same class label. Given this objective function, the approach then searches for the best linear transformation of the n-dimensional data. This method has a significant advantage in comparison to traditional PCA or MDS, because it captures the cluster structure of the data, and in addition the intra-cluster shapes. One general problem with general projections and embeddings is that users may have trouble interpreting the display. Another problem is that these approaches work well whether the clusters are neatly separated in n–D or the data contains only small amounts of noise.

The Self-Organizing Map (SOM) is a neural network algorithm based on unsupervised learning (see [10] for further readings). The basic idea is to stretch a 1-D or 2-D neuron grid through the data. The lattice of the grid can be either rectangular or hexagonal. Each neuron is represented by an n-dimensional prototype vector. During the iterative training of a SOM, the neurons become the cluster means, and points closest to the neurons are considered to belong to that cluster. Intuitively, a SOM can be seen as a constraint k-means algorithm (the net of neurons is very flexible and folds onto the data clouds).

SOM's can be used to visualize the clustering structure of an n-dimensional data space. The 1-D or 2-D grids get wrapped into a flat 2-D layout [12]. An alternative visual representation of the clustering structure can be achieved by visualizing either the distances of each grid unit to its neighboring grid units or the similarity of grid units. In most cases, gray shading (U-Matrix) and geometric shapes (Distance-Matrix) are used to represent distances to neighboring grid units, and color to show similarity of grid units (similarity coloring). A detailed discussion of SOM-based data visualization techniques is presented in [14].

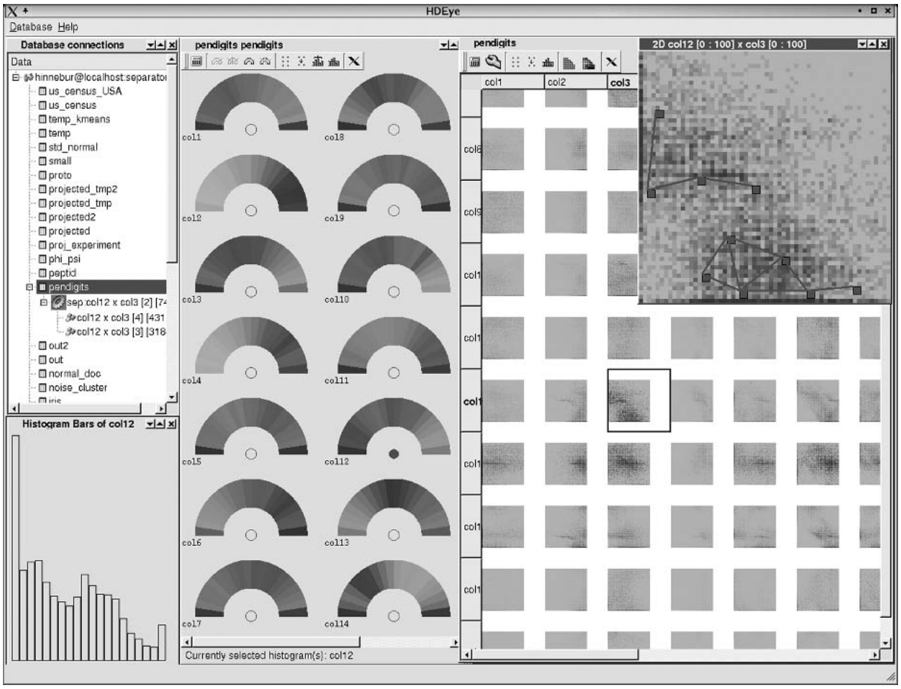
Extension of an Optimized Clustering Algorithm

An interesting extension of OptiGrid [8] with both an iconic visualization technique and kernel- density plots is HD-Eye [9]. OptiGrid is a density-based clustering algorithm that uses contracting projections and separators to build up an n-dimensional grid. Clusters are defined as highly populated grid cells.

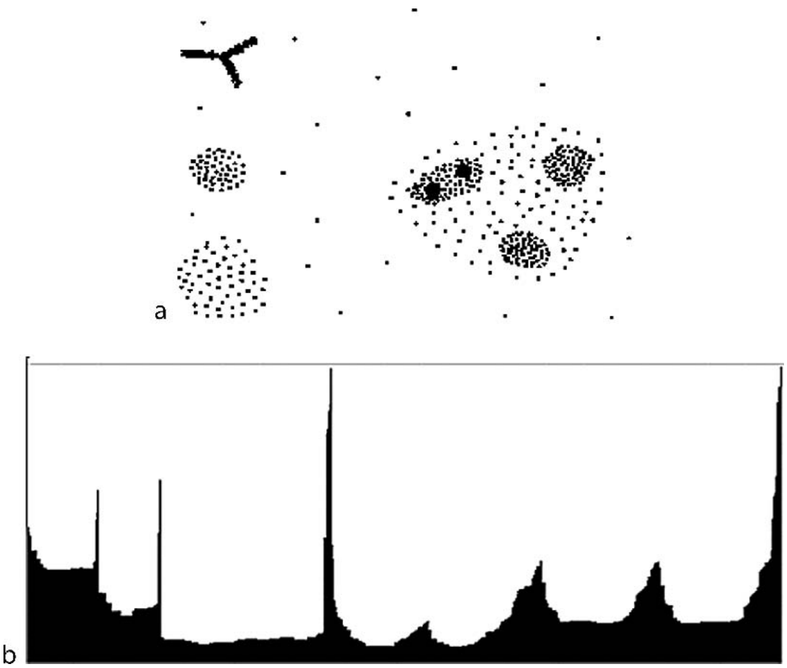
HD-Eye considers clustering as a partitioning problem. A projection is of potential interest in finding cluster separators whether it neatly separates a number of clusters in the projected space. Finding interesting projections and specifying good separators are two difficult problems; both are difficult to compute automatically. The basic idea of HD-Eye is to allow specific user feedbacks in two crucial steps of the partitioning process. To support the user in surveying the vast set of orthogonal projections in order to find potential useful projections, HD-Eye employs an abstract iconic display (Fig. 1). The shapes of the icons are determined based on the number of maxima of the data's density function, and how well the maxima are separated in the projections. The color of the icons represents the number of data items belonging to a maxima. Once a potential interesting projection has been selected, HD-Eye provides 1-D/2-D color-based density plots for a further analysis of the projection. Density plots easily allow the analyst to check whether a projection separates a number of clusters well.

The HD-Eye approach doesn't require that a projection neatly separates all clusters but at least two clusters. HD-Eye follows an iterative partitioning process. Based on his/her domain knowledge and intuition, the analyst interactively selects interesting projections from the iconic display. Then, the analyst defines useful separators within the density plots by drawing either a split line (linear partitions) or a split polygon (non-linear partitions). The analyst can easily follow the interactive partitioning process by inspecting the separator tree. After each interactive partitioning, the clusters get refined, i.e., highly populated regions based on the refined grid, and added into the separator tree.

Another interesting approach is OPTICS [2] (Fig. 2). OPTICS is an extension of the DBSCAN [6] algorithm and creates a one-dimensional ordering of the data items. The ordering of the data items represents its density-based clustering, which can be obtained from a broad range of input parameter



Visual Clustering. Figure 1. The HD-Eye interfaces has three main visual components – the iconic display showing properties of the projections, 1–D and 2–D kernel-density plots for a further analysis of selected projections, and the separator tree. Clockwise starting from the top: separator tree, iconic representation of 1–D projections, 1–D projection histogram, 1–D color-based density plots, iconic representation of multi dimensional projections and color-based 2D density plot. Figure is taken from the exploration of a large molecular biology data set. (used with permission of Alexander Hinneburg).



Visual Clustering. Figure 2. The figure shows an example data (Fig. 2a) and its reachability plot (Fig. 2b) – objects are on the x-axis together with their reachability values on the y-axis. (used with permission of Mihael Ankerst).

settings. Clusters in DBSCAN are determined based on ε and MinPt parameters, when ε is the radius of a local neighborhood around each data item, and MinPt determines the minimal number of data items in the local neighborhood. The basic observation of OPTICS is that given a constant MinPt value, density-based clusters with respect to a higher density (lower ε) are completely contained in density-based clusters with respect to a lower density (greater ε).

The ordering of the data items can be used to visualize how the data items are clustered. These visualizations are called reachability plots. The data items are visualized according to the clustering ordering on x-axis, together with their associated reachability distance on y-axis. Intuitively, data items sharing a common cluster are close to each other in the cluster ordering and have similar reachability distances. The reachability distance jumps whether a different cluster starts in the ordering. In addition, the ordering of the data items can be used to compute a cluster hierarchy (dendrogram).

Clustering Based on Visual User Feedback and Refinement

An interesting approach is presented by Aggarwal [1]. The basic idea is similar to HD-Eye, i.e., the search for projection that neatly separates clusters. In contrast to HD-Eye that uses an iconic display to show properties of the projections and then supports an interactive search for interesting projections, Aggarwal proposed an iterative cluster partitioning based an automated subspace detection algorithm.

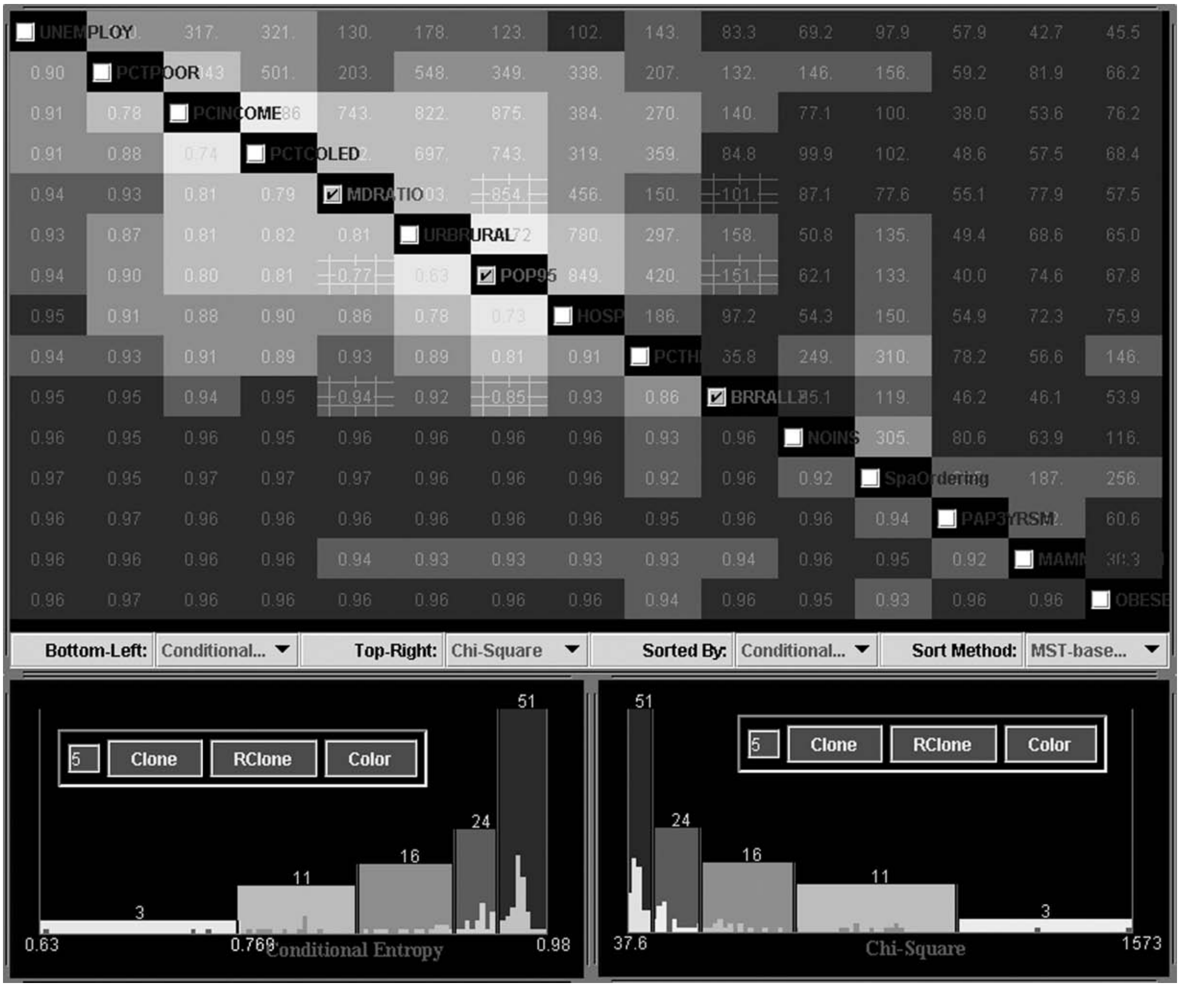
The idea of the algorithm is straightforward and follows an iteration process. In each iteration step, a number of data items called polarization points are randomly chosen from the data set. Next, an analytic method is used to find a lower-dimensional subspace that neatly groups the data items around the polarization points. Once an interesting projection has been found, HD-Eye and Aggarwal use kernel-density plots of the projected data in order to discover the clustering structure. In contrast to the HD-Eye approach that allows an interactive separation of clusters in the density plots, Aggarwal's approach separates clusters by choosing noise levels.

The iterative cluster partition process terminates whether all data items belong at least to one cluster. The final clustering structure is extracted in a frequent item analysis based on the recorded user selections in each partitioning step. To control the granularity of the cluster separation in the final clustering structure, the analyst can choose different noise levels for the same projection.

ClusterSculpture [13] is an interesting approach that allows an analyst to interactively refine an initially computed hierarchical clustering structure. The hierarchical clustering structure is described as a dendrogram. The analyst refines that dendrogram by either splitting or merging a number of nodes; if it's necessary the analyst might want to reorganize the whole dendrogram. ClusterSculpture supports the analyst in deciding whether a node should be split or merged by presenting the relationship between the attribute values and the clustering structure, as well as the distances of neighboring data items around a selected cluster.

ClusterSculpture provides three main visualizations. A dendrogram shows the cluster hierarchy, and it allows the analyst to quickly survey the clustering structure. The distribution of the attribute values in each dimension are shown in a point map. Each data item is visualized as a horizontal line with one colored pixel for each dimension. The color of each pixel represents the normalized attribute value in that dimension. Additionally, the data items are sorted along the y-axis to emphasize the clustering structure. The distances of neighboring data points around the leaf cluster closest to a selected data item is presented when the analyst selects a data item in the point map. Three different colors are used to represent the selected data item, and whether or not data items share the same neighbor with the selected data item.

A number of exploratory data analysis systems have interactive visualizations to support the analyst in finding clusters. GGobi [4] supports the analyst by providing a spin-and-brush mechanism. Starting with



☐ UNEMPLOY	0.	317.	321.	130.	178.	123.	102.	143.	83.3	69.2	97.9	57.9	42.7	45.5
0.90	☐ PCTPOOR	43	501.	203.	548.	349.	338.	207.	132.	146.	156.	59.2	81.9	66.2
0.91	0.78	☐ PCINCOME	86	743.	822.	875.	384.	270.	140.	77.1	100.	38.0	53.6	76.2
0.91	0.88	0.74	☐ PCTCOLED	2.	697.	743.	319.	359.	84.8	99.9	102.	48.6	57.5	68.4
0.94	0.93	0.81	0.79	☑ MDRATIO	03.	854.	456.	150.	101.	87.1	77.6	55.1	77.9	57.5
0.93	0.87	0.81	0.82	0.81	☐ URBRURAL	72	780.	297.	158.	50.8	135.	49.4	68.6	65.0
0.94	0.90	0.80	0.81	0.77	0.63	☑ POP95	849.	420.	151.	62.1	133.	40.0	74.6	67.8
0.95	0.91	0.88	0.90	0.86	0.78	0.73	☐ HOSP	186.	97.2	54.3	150.	54.9	72.3	75.9
0.94	0.93	0.91	0.89	0.93	0.89	0.81	0.91	☐ PCTH	35.8	249.	310.	78.2	56.6	146.
0.95	0.95	0.94	0.95	0.94	0.92	0.85	0.93	0.86	☑ BRRALL	5.1	119.	46.2	46.1	53.9
0.96	0.95	0.96	0.95	0.96	0.96	0.96	0.96	0.93	0.96	☐ NOINS	305.	80.6	63.9	116.
0.97	0.95	0.97	0.97	0.97	0.96	0.96	0.96	0.92	0.96	0.92	☐ SpaOrdering	[illegible]	187.	256.
0.96	0.97	0.96	0.96	0.96	0.96	0.96	0.96	0.95	0.96	0.96	0.94	☐ PAP3YRSM	2.	60.6
0.96	0.96	0.96	0.96	0.94	0.93	0.93	0.93	0.93	0.94	0.96	0.95	0.92	☐ MAMM	30.3
0.96	0.97	0.96	0.96	0.96	0.96	0.96	0.96	0.94	0.96	0.95	0.93	0.96	0.96	☐ OBESE

Bottom-Left: Conditional... Top-Right: Chi-Square Sorted By: Conditional... Sort Method: MST-base...

Visual Clustering. Figure 3. The figure shows a visualization of the entropy matrix of a real cancer data set with 72 dimensions. The two histograms at the bottom of the image allows the analyst to adjust the classification and coloring in an interactive fashion. (used with permission of Diansheng Guo).

an initial projection of the data, the analyst tours through the space of possible projections until an interesting projection is reached. The analyst then paints a cluster in that projection using an easy to distinguish color, and continues touring until no more clusters are revealed.

Visualization Used as Interactive Feature Selection for Clustering

An interesting subspace selection method that effectively identifies subspaces determining strong clusters is proposed by Guo [7]. The method is based on the computation of the pairwise conditional entropy values of 2-D subspaces. The pairwise conditional entropy values are visualized in the so-called entropy matrix, which allows the data analyst to easily understand relationships among dimensions (Fig. 3). Interesting multi-dimensional subspaces can be either automatically extracted or visually identified in the entropy matrix. The selected dimensions can be fed into a clustering algorithm in order to improve the effectiveness of the clustering results. Although designed as a feature selection method, the entropy matrix allows the analyst to get an understanding of the overall clustering structure of an n-dimensional data space.

Key Applications

Effective mining of large traditional relational databases, complex 2-D and 3-D multimedia databases, geo-spatial databases, and bio-databases.

Cross-references

- ▶ Automated Clustering Algorithms
- ▶ Exploratory Data Analysis
- ▶ Visual Analytics
- ▶ Visual Data Mining

Recommended Reading

1. Aggarwal C.C. A human–computer interactive method for projected clustering. IEEE Trans. Knowl. Data Eng., 16(4):448–460, 2004.
2. Ankerst M., Breunig M.M., Kriegel H.P., and Sander J. OPTICS: ordering points to identify the clustering structure. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 49–60.
3. Asimov D. The grand tour: a tool for viewing multidimensional data. SIAM J. Sci. Stat. Comp., 6(1):128–143, 1985.
4. Cook D. and Swayne D.F. Interactive and Dynamic Graphics for Data Analysis – With R and GGobi. Springer Science and Business Media, New York, NY, USA, 2007.
5. Dhillon I.S., Modha D.S., and Spangler W.S. Visualizing Class Structure of Multidimensional Data. In Proc. 30th Symp. on the Interface: Computing Science and Statistics, 1998, pp. 488–493.
6. Ester M., Kriegel H.P., Sander J., and Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. In Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, 1996, pp. 226–231.
7. Guo D. Coordinating computational and visual approaches for interactive feature selection and multivariate clustering. Inform. Vis., 2(4):232–246, 2003.
8. Hinneburg A. and Keim D.A. Optimal grid-clustering: towards breaking the curse of dimensionality in high-dimensional clustering. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 506–517.
9. Hinneburg A., Keim D.A., and Wawryniuk M. HD-Eye: visual mining of high-dimensional data. IEEE Computer Graphics and Applications, 19(5):22–31, 1999.
10. Kohonen T. Self-Organizing Maps. third edn., Springer Series in Information Science, 2001.
11. Koren Y. and Carmel L. Robust Linear Dimensionality Reduction. IEEE Trans. Vis. Comput. Graph., 10(4):459–470, 2004.
12. Kraaijveld M., Mao J., and Jain A. A nonlinear projection method based on kohonen's topology preserving maps. IEEE Trans. Neural Networks, 6(3):548–559, 1995.
13. Nam E.J., Han Y., Mueller K., Zelenyuk A., and Imre D. ClusterSculptor: A Visual Analytics Tool for High-Dimensional Data. In IEEE Symp. on Visual Analytics Science and Technology, 2007, pp. 75–82.
14. Vesanto J. Som-Based Data Visualization Methods. Intell. Data Analy., 2(3), 1999.
15. Yang L. Interactive exploration of very large relational datasets through 3D dynamic projections. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 236–243.

Visual Content Analysis

MARCEL WORRING, CEES SNOEK
University of Amsterdam, Amsterdam,
The Netherlands

Synonyms

Video indexing; Image indexing

Definition

Visual content analysis is the process of deriving meaningful descriptors for image and video data. These descriptors are the basis for searching large image and video collections. In practice, before the process starts, one applies image processing techniques which take the visual data, apply an operator, and return other visual data with less noise or specific characteristics of the visual data emphasized. The analysis considered in this contribution starts from here, ultimately aiming at semantic descriptors.

Historical Background

Analyzing the content of visual data using computers has a long history, dating back to the 1960s. Some initial successes prompted researchers in the 1970s to predict that the problem of understanding visual material would soon be solved completely. However, the research in the 80s showed that these predictions were far too optimistic. Even now, understanding visual data is still a major challenge.

In the 90s a new field emerged, namely content-based image retrieval (CBIR), where the aim is to develop methods for searching in large image archives. Where the computer vision community was publishing papers on the analysis of 10–50 images, the CBIR researchers started with 1,000 images or more. Nowadays, research papers consider data sets which are orders of magnitudes larger in size. The paper by Smeulders et al. [10] reviewed the state-of-the-art in 2000, almost all of the findings are still very relevant for current research.

The CBIR field has not only developed itself since then, it also had a major impact on the computer vision field. Large visual collections are also becoming the standard in computer vision. It also led to new methods for analyzing visual data. In early methods for understanding the content of the data pre-defined models were used. In current research, models are learned automatically from large sets of annotated examples, based on the appearance of objects in the visual data. This makes the computer vision algorithms more flexible and easier to adapt to new data sets.

Video content analysis started later than image analysis due to the high processing power required. Computer capacity, however, has increased substantially over the last 10 years making it feasible to process also large collections of video. Research in video content analysis started with camera shot segmentation and simple analysis. Soon the research community realized that the result of automatic speech recognition is an important, sometimes decisive, factor in video content analysis. Best results are obtained if the visual and audio modalities are analyzed in conjunction [11]. Relying on speech is fine for video programs with a clear speech signal, nicely separated from environmental sounds and music. More and more, however, people create video data without such clear speech, so the role of visual content has moved to the forefront in video content analysis again.

So, although visual content analysis started around 50 years ago, it is still an active research area with many research challenges ahead.

Foundations

Automatically extracting the semantics of visual data is a notoriously difficult problem which has become known as the semantic gap [10]:

Semantic gap: The semantic gap is the lack of coincidence between the information that one can extract from the sensory data, and the interpretation that the same data has for a user in a given situation.

The existence of the gap has various causes. One reason is that different users interpret the same visual data in a different way. This is especially true when the user is making subjective interpretations of the visual data. In the following part those subjective interpretations are not considered. However, also for objective interpretations like the presence of a car in a picture, developing automatic methods is still difficult. Consequently, their performance is not optimal. The difficulties are due to the large variations in appearance of visual data corresponding to one semantic concept. Cars, for example, come in different models, shapes, and color. The above causes are inherent to the problem. Visual analysis methods should address these fundamental problems.

There are also variations in appearance which are not due to the richness of the semantics. Varying the viewpoint, lighting and other circumstantial conditions in the recording of a scene will deliver different data, whereas the semantics have not changed. These variations induce the so called sensory gap, which is defined as [10]:

Sensory gap: The sensory gap is the gap between the object in the world and the information in an image recording of that scene.

One of the consequences of the semantic and sensory gap is the fact that recordings of semantically different objects might appear more similar than two recordings of the same object in different recording conditions. Methods are needed that are minimally affected by the sensory gap, while still being able to distinguish objects with different semantics. This is the essence of visual content analysis. General methods for doing the analysis are described next.

Methods for deriving the semantics of visual data are mostly based on a supervised learning scheme. In such a scheme, the developer provides the system with a large set of annotated examples. For all of the examples, the system extracts low-level features. The system then uses a machine learner to build a model for each semantic concept. Now, if a new image or video without annotation is given, the system again computes features. It then uses the model to derive a measure of the probability of the concept being present in the visual data. An overview of the general concept based video content analysis scheme is depicted in Fig. 1.

So, the two main things to consider are the features to extract and the machine learning scheme.

Features are many, see [10,11] for an extensive overview. A good starting point for a set of practical visual features is defined in the MPEG-7 standard [8]. There are different categories of features namely color features, texture features, shape features, and motion features, which will now be described.

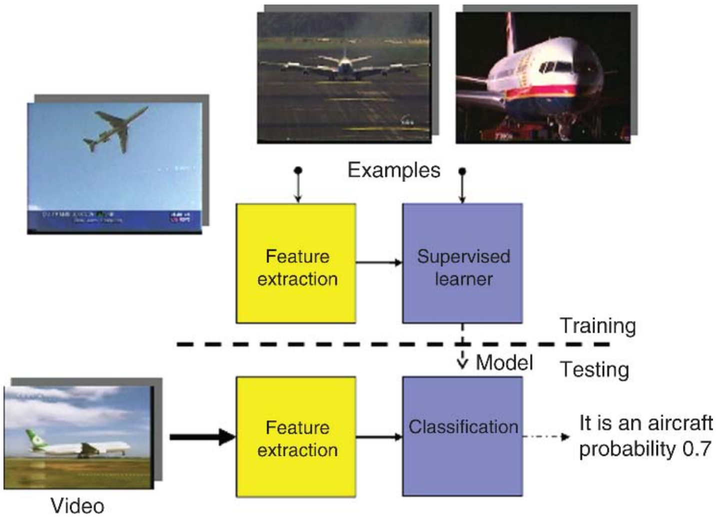
Color features in their simplest form are global descriptions of the visual content based on color histograms. There are numerous color spaces to choose from in which to compute the histogram, including standard *RGB*, the intuitive *HSV* space, the perceptually uniform *Lab* space, or the invariant set of color spaces in [4]. Methods which use more elaborate methods than histograms take the color layout into account, in addition to global characteristics.

Where color is a property which can be measured at every pixel in the visual data, texture is a measure which considers patterns in the image. Again, there is a variety of methods to choose from with varying characteristics in terms of invariance. Many can be traced back to statistics on co-occurrence of grey values in a local neighborhood, or filters emphasizing patterns in certain directions like Gabor filters. In [3], a convenient set of features for texture description has been defined based on natural statistics.

Color and texture often form the basis for segmenting the visual data in homogeneous regions. This process is known as weak segmentation as it is purely based on the visual data itself, not on the interpretation of the data. The latter is known as strong segmentation, which is delineating the contour of a semantic object in the image.

Strong segmentation is required before shape features can be measured. As a consequence, shape features are only applicable for those applications where the background can be controlled easily, like the recognition of logos or a set of objects on a turntable. A distinction is made here between descriptors that describe the object as a region, e.g., based on properties of the medial axis, and measures that take the contour as the basis, like the curvature scale space descriptor.

In most practical cases, strong segmentation is impossible, whereas global descriptors loose too much of



Visual Content Analysis. Figure 1. General scheme for concept-based learning in visual data using annotated examples.

the spatial information. Some methods therefore focus on weak segmentation, describing the resulting regions individually, or move to the detection of key points. These key points are characteristic points on the contours of objects, which in many cases capture the most important information of the object [7].

All of the above features are equally important for video data. However, for video one can also make use of the temporal dimension. One can analyze the motion pattern to find the camera motion, track regions or points of interest through the sequence and describe their tracks, or derive general measures of the activity in the scene. These additional information sources can enhance the interpretation of the data.

Feature extraction yields a feature vector for every image or video segment. As indicated, for interpretation of the data, state-of-the-art systems follow a supervised learning approach.

A good overview of learning methods is presented in [5]. Although there is a large variety of methods, the Support Vector Machine has become the standard choice in most visual indexing schemes [1,14].

The above considers the visual data in isolation, but often the data has associated text data like the automatic speech recognition results for video. This data can be used as an additional source to improve the understanding of the data. In fact, for some concepts the textual data gives almost all the clues to interpret the visual data, others perform best when only the visual data is used. Relations between the concepts detected can also provide essential information for some concepts. Therefore, a method like [13], which learns for a given concept the optimal analysis scheme through appropriate analysis of the data, is the best choice.

Key Applications

Video content analysis plays a role in various application areas, the major ones are considered here.

Broadcasting

The broadcasting field generates enormous amounts of audio-visual data, which should be stored for later reference and re-use. Manual annotation is laborious and expensive. Tools for automatic analysis are of great importance to deal with this flood of videos.

Narrow Casting

Producing video, up to now, has been limited to major film studios or specialized companies. Now, every organization, association or individual can create video and put it on the web. Even more than for broadcasting companies, narrow casters have limited resources to manually analyze the content of their videos.

Law Enforcement

Through the availability of cheap cameras, video content analysis plays a major role in law enforcement. There is a need for content analysis to detect and analyze illicit video material on confiscated hard disks containing hundreds of gigabytes of video data. In video surveillance, a single camera generates 7*24 h of video per week. Any visual content analysis which helps the operator in reducing the load of watching these video streams is of great importance.

Web Search

There is no need to explain that there is a tremendous amount of video available on the Internet. Video search on the web, however, is still mostly limited to keyword-based search for full video clips. Video content analysis is needed to delve into the actual content of all these videos out there.

Future Directions

Obviously the research in video content analysis has not reached its end yet. On the contrary, the community is only beginning to automatically understand the meaning of visual material. To bring the field a step further, researchers should tackle the large number of open questions out there.

Tangential contributions will introduce features with higher discriminatory power, while having better invariant properties, and new machine learning techniques, better suited for the specifics of visual data.

A new direction is tackling the major bottleneck in supervised learning, namely the amount of training examples needed, by employing user tagged visual data like Flickr. The latter annotations are less accurate than the current practice in supervised learning, but the amount of training samples is orders of magnitude larger.

Another promising direction is the use of capture time meta data, contextual information, and social networks to provide additional clues for interpreting the visual data.

Data Sets

The research community in video content analysis is very active. In recent years there was an important role for benchmarks encouraging research in the various subfields. They do so by providing large test collections and uniform evaluation procedures. The benchmarks provide a forum where researchers can compare their results, bringing the research a big step further every year.

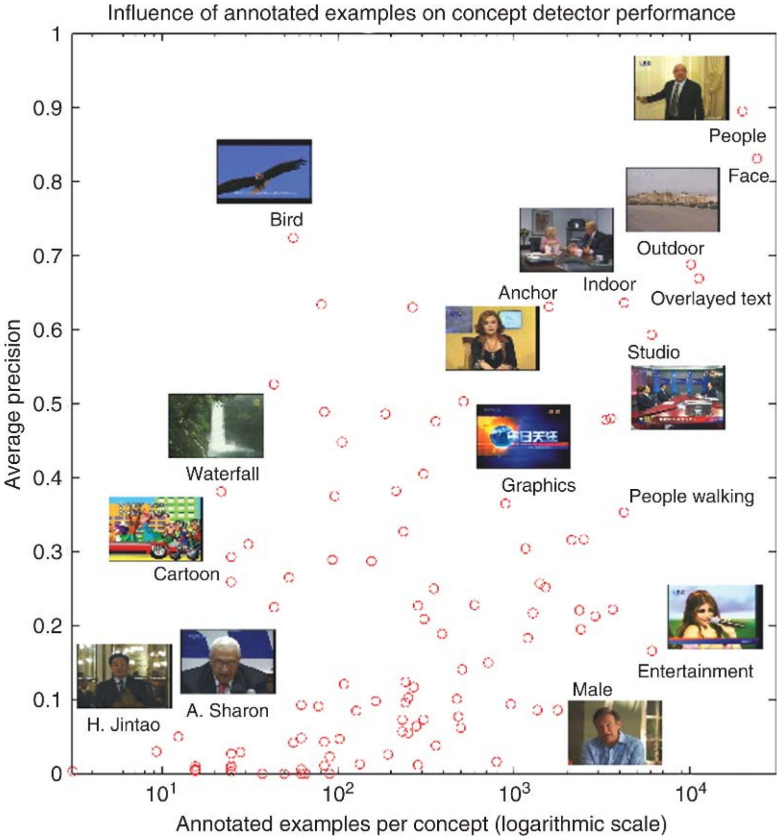
For visual content analysis of images the most elaborate benchmark is the PASCAL VOC challenge [2]. This benchmark contains image sets of several hundreds to thousands of images with a ground truth at the semantic level, a box containing the object, as well as complete outlines of the object in the scene. In this manner, it provides a challenging data set for testing.

Video analysis, especially for the purpose of search, has been studied for several years now in the TRECVID benchmark organized by NIST. In this benchmark, videos from mostly news and other broadcasts have been semantically annotated at the shot level [9].

Collections contain hundreds of hours of video material. Hence they not only pose a challenge at the analysis level, but also at the computer performance level. In addition, one can see that the analysis of such data sets requires expertise from different fields, making the threshold to work on such collections large. Therefore, the MediaMill Challenge [12] has taken the idea of making shared data sets available one step further. This resource not only contains annotations of the TRECVID data, but also a collection of five pre-cooked experiments. To allow an easy start on the topic, the results of each sub-step are provided. Later Columbia and the VIREO group made similar resources available [6,15]. The above efforts allow for visual database research without actually having to perform the visual analysis.

Experimental Results

The performance of video analysis algorithms for information retrieval is typically measured by the average precision. This is a single-valued measure proportional to the area under a recall-precision curve. This value is the average of the precision over all relevant judged



Visual Content Analysis. Figure 2. Indication of the state-of-the-art performance in video content analysis. Note the clear relation between the performance and the amount of annotated examples.

shots. Hence, it combines precision and recall into one performance value. To be precise, let $L^k = \{l_1, l_2, \ldots, l_k\}$ be a ranked version of the answer set A. At any given rank k let $R \cap L^k$ be the number of relevant results in the top k of L, where R is the total number of relevant results. Then average precision is defined by:

$$average\ precision = \frac{1}{R} \sum_{k=1}^{A} \frac{R \cap L^k}{k} \lambda(l_k), \qquad (1)$$

where indicator function $\lambda(l_k) = 1$ if $l_k \in R$ and 0 otherwise. As the denominator k and the value of $\lambda(l_k)$ are dominant in determining average precision, it can be understood that this metric favors highly ranked relevant results. To give an indication of the state-of-the-art, Fig. 2 gives an overview of results for TRECVID data using the features defined for the MediaMill challenge. Performance is shown as function of number of annotated examples as this has proven to be an important indicator for the quality of the result.

One can conclude that performance is not perfect, but a rapid increase in performance over the last years can be observed. Hence, one could expect that more and more reliable concept detectors will become available in the coming years.

Cross-references

- ▶ Content-Based Video Retrieval
- ▶ Decision Tree Classification
- ▶ Image Retrieval
- ▶ Multimedia Databases
- ▶ Multimedia Information Retrieval
- ▶ Video Scene and Event Detection
- ▶ Video Content Analysis

Recommended Reading

1. Chang C.C. and Lin C.J. LIBSVM: a library for support vector machines. 2001, software available at: http://www.csie.ntu.edu.tw/~cjlin/libsvm/.
2. Everingham M., van Gool L., Williams C., Winn J., and Zisserman A. The PASCAL visual object classes homepage. Available at: http://www.pascal-network.org/challenges/VOC/.
3. Gemert J., Geusebroek J., Veenman C., Snoek C., and Smeulders A. Robust scene categorization by learning image statistics in context. In Proc. Int. Workshop on Semantic Learning Applications in Multimedia, 2006.
4. Geusebroek J., Boomgaard R., Smeulders A., and Geerts H. Color invariance. IEEE Trans. Pattern Anal. Mach. Intell., 23(12):1338–1350, 2001.
5. Jain A., Duin R., and Mao J. Statistical pattern recognition: A review. IEEE Trans. Pattern Anal. Mach. Intell., 22(1):4–37, 2000.
6. Jiang Y.G., Ngo C.W., and Yang J. VIREO-374: LSCOM semantic concept detectors using local keypoint features. Available at: http://www.cs.cityu.edu.hk/~yjiang/vireo374/.
7. Lowe D.G. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis., 60(2):91–110, 2004.
8. Sikora T. The MPEG-7 visual standard for content description-an overview. IEEE Trans. Circ. Syst. Video Tech., 11:696–702, 2001.
9. Smeaton A. Large scale evaluations of multimedia information retrieval: The TRECVid experience. In Proc. 4th Int. Conf. Image and Video Retrieval, 2005, pp. 19–27.
10. Smeulders A., Worring M., Santini S., Gupta A., and Jain R. Content based image retrieval at the end of the early years. IEEE Trans. Pattern Anal. Mach. Intell., 22(12):1349–1380, 2000.
11. Snoek C. and Worring M. Multimodal video indexing: A review of the state-ofpascal-the-art. Multimed. Tool Appl., 25(1):5–35, 2005.
12. Snoek C., Worring M., van Gemert J.C., Geusebroek J.M., and Smeulders A. The challenge problem for automated detection of 101 semantic concepts in multimedia. In Proc. 14th ACM Int. Conf. on Multimedia, 2006.
13. Snoek C., Worring M., Geusebroek J., Koelma D., Seinstra F., and Smeulders A. The semantic pathfinder: Using an authoring metaphor for generic multimedia indexing. IEEE Trans. Pattern Analy. Mechine Intell., 28(10):1678–1689, 2006.
14. Vapnik V. The nature of statistical learning theory. Springer-Verlag, New York, NY, USA, 2nd edn., 2000.
15. Yanagawa A., Chang S.F., Kennedy L., and Hsu W. Columbia university's baseline detectors for 374 LSCOM semantic visual concepts. Columbia University, 2007, aDVENT technical report 222-2006-8.

Visual Data Analysis

- ▶ Visual Analytics
- ▶ Visual Data Mining

Visual Data Exploration

- ▶ Dense Pixel Displays
- ▶ Disclosure Risk

Visual Data Mining

SIMEON J. SIMOFF
University of Western Sydney, Sydney, NSW, Australia

Synonyms

Visual data analysis; Visual analysis; Visual discovery; Immersive data mining; VDM

Definition

Visual data mining (VDM) is the process of interaction and analytical reasoning with one or more visual representations of abstract data. The process may lead to the visual discovery of robust patterns in these data or provide some guidance for the application of other data mining and analytics techniques. It facilitates analysts in obtaining deeper understanding of the underlying structures in a data set. The process relies on the tight interconnectedness of tasks, selection of visual representations, the corresponding set of interactive manipulations, and respective analytical techniques. Discovered patterns form the information and knowledge utilized in decision making.

Historical Background

Visual exploration of large data sets had been used as a complementary technique to data mining in order to obtain additional information about the data set. Since the early 1990s there has been recognition of the need for specific visualization techniques for large data sets that enable data mining tasks. The term "visual data mining" emerged in the late 1990s, labeling such techniques combined with interactive functionality and some guidelines for sense making from different visual displays of the data and/or the output of data mining algorithms. Ankerst's and Niggemann's theses around the millennium considered the establishment of more systematic scientific and methodological foundations of the field. The 3D Visual Data Mining (3DVDM) project in Aalborg University added a new twist, placing the analyst literally "within" the visual representation of the data set in various stereoscopic virtual reality systems.

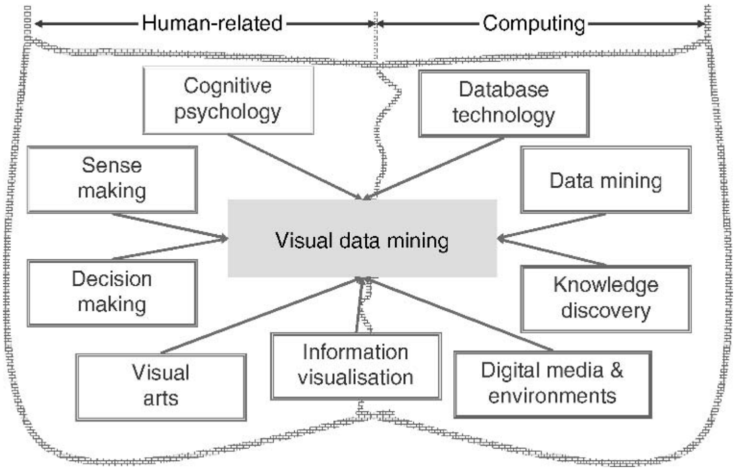
During 2001–2003 a series of workshops on visual data mining, conducted in conjunction with major KDD conferences on both sides of the Atlantic, bridged scholars in the fields of data mining and knowledge discovery in databases, information visualization, human-computer interaction as well as cognitive psychologists related to computing, industry practitioners, and developers of visual data mining tools. They established some common elements that need to be specified when developing methodologies for visual data mining, including: (i) the initial assumptions posed by the respective visual representations; (ii) the set of interactive operations over the respective visual representations and guidelines for their application and interpretation, and; (iii) the range of applicability and limitations of the visual data mining methodology. The majority of the works, published in the proceedings of these workshops after completion, have made their way into the books and special issues, referred in section "Recommended Reading". These and numerous subsequent research and development efforts led to the establishment of visual data mining as an essential component of visual analytics.

Foundations

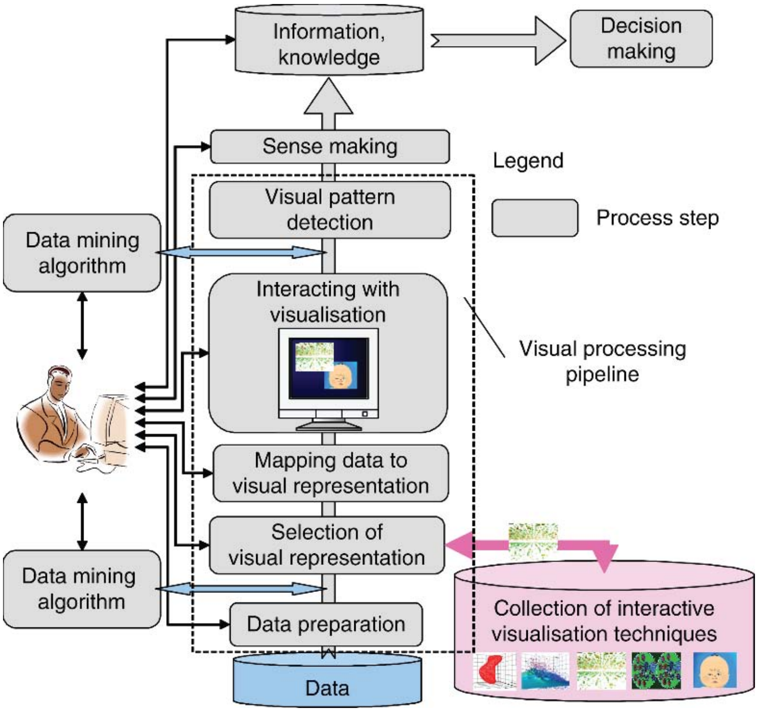
Visual data mining integrates the abilities of the human perceptual system for pattern recognition and human reasoning from images, with the data processing and display power of computers, aiming at capitalizing on the best of both worlds. From a disciplinary perspective, visual data mining emerged as the confluence of several disciplines, the main of which are grouped into two clusters in Fig. 1. On the right-hand side are those, primarily related to computing, when on the left-hand side are those primarily related to humans. From the perspective of scientific fundamentals, though not exhaustive, this list shows the diversity of sources that contribute to the methods and approaches developed in visual data mining.

The VDM process relies on the interactive visual processing pipeline, shown in Fig. 2. Each step within the pipeline involves interaction with the analyst and all the iterative loops in the process close via the analyst. Data mining algorithms can be used to assist the process before any visualization has been considered, and/or after a visual interaction with the data.

Being the core of this human-centred process, visual processing is constrained by the capacity of the human visual system to somewhat 10^6 to 10^7 observations. Large data sets easily surpass this constraint. Consequently, essential for visual data mining are low-dimensional visual representations of high dimensional data that (i) preserve relations between the data points in the high-dimensional space, and (ii) can make such relations visually detectable. Coupling of visualization algorithms with statistical techniques is one way it generating visual data representations that more accurately present the underlying properties of the data. Such visual representations are expected to fit consistently certain metaphors that are close to human mental models, as humans depict structures in the data by forming a mental model which captures only a gist of the information. These are important design considerations in visual data mining, as visualization



Visual Data Mining. Figure 1. Visual data mining as a confluence of disciplines.


Information, knowledge
Decision making
Sense making
Legend
Visual pattern detection
Process step
Data mining algorithm
Interacting with visualisation
Visual processing pipeline
Mapping data to visual representation
Selection of visual representation
Data mining algorithm
Data preparation
Collection of interactive visualisation techniques
Data

Visual Data Mining. Figure 2. Visual data mining is a human-centred interactive analysis and discovery process.

algorithms display data sets that usually lack inherent 2D and 3D semantics.

As manipulation of visual representations of the data set (or subsets of it) is essential in enabling visual pattern detection, the success of the process illustrated in Fig. 2 depends on: (i) the breadth of the collection of visualization techniques, (ii) the consistency of the design of these visual representations, (iii) the ability to remap interactively the data attributes to the parameters of the visual representations, (iv) the set of functions for interacting with the visualization, and (v) the capabilities that these functions offer in support of the sense-making process. From an analyst point of view, the suite of visual representations and interactive techniques is the toolbox for constructing "virtual worlds" from the data. The visual metaphors in such worlds are the vehicles to transfer meaning from the visual structure of the data into the semantics of the underlying problem. The transfer is enabled by the rules of behavior of the visualization elements that constitute a visual language. The consistent design of visual languages and techniques for interaction with different visual representations of a dataset take into account the specifics of human long-term and short-term memory, knowledge of interaction with virtual environments, including frames of reference, orientation, and navigation. These considerations are also central in order to support the sense-making process in Fig. 2 (the process itself has several iterative steps that are not shown in Fig. 2).

Dealing with large volumes of data means *scalability* and interactivity means *real-time response* of the visualization algorithms in the toolbox in Fig. 2, and the overall visual processing pipeline in order to analyze the details available in the large databases. Interactivity also means support of various visual data mining operations, for example, a visual drill down or visual clustering.

The presentation of the extracted information and knowledge during the decision making step in Fig. 2 usually requires a different visualization approach to the visual data mining step. Such separation of visual analysis tasks from the visual presentation of the results is reflected in contemporary visual data mining technology.

Key Applications

Visual data mining is applied in a wide range of areas of human endeavor, which generate large volumes of otherwise incomprehensible data. These include science, business, technology development, medicine, and healthcare. Chemistry and new drug discovery are examples of successful application of VDM methods. The applications in various fields provide feedback to the development of VDM technology, in terms of integration of 1D, 2D, 2.5D, and 3D visual representations, respective sets of interactive operations, and evaluations of the visual data mining capabilities and interfaces by the analysts in respective fields.

Special key application areas of visual data mining are the analysis of geospatial and spatio-temporal data sets, visual link analysis and visual text mining. Spatial data sets include spatially related attributes, which provides guiding information for the choice of visual representations and interaction methods. Visual link analysis is applied in areas where there is a need for identification of hidden links and chain of links between the attributes in a data set, for example, the fraud detection type of problems in different environments – government, corporate, insurance, retailing, and crime. These problems have the same feature: different cases of fraud may differ in terms of the patterns that reflect them in the data, and the nonvisual data mining and analytics methods may not be able to detect them. The advent of social computing has led to embedding visual link analysis technologies in the social computing systems. Visual text mining uses a variety of statistical techniques to relate groups of words with groups of documents, and then generates visual data summarizations using different metaphors.

Future Directions

Visual data mining, as a discovery mode of interaction with databases, plays a central role in visual analytics. Though interaction has been recognized, there is a strong demand on the development of interactive visualizations, which are fundamentally different from the static visualizations. Designed with the foundations of perceptual and cognitive theory in mind, and focused on supporting the processes and methods in visual data mining, these visual data representations are expected to be relevant to specific tasks and effective in terms of achieving the analytics goals. Within this direction essential are methods for: (i) design and evaluation of visualizations for visual data mining, including metrics for the evaluation of the interactivity of visualizations and their ability to facilitate discovery processes; (ii) visualization techniques of projections of high-dimensional data that preserve the statistical properties of the patterns in the original data; (iii) further integration of data visualization and sonification and development of interactive mining methods, including immersive ones, that operate with such data representations; and (iv) enabling VDM systems to support research strategies and inquiry styles of different data analysts.

Experimental Results

As a rule, research papers in visual data mining, reporting on novel visual analysis techniques, present the results from the application of respective techniques to several data sets. Papers, which explore human-computer interaction aspects, report the outcomes of observational case studies and usability experiments.

Data Sets

Data sets are available from the KDNuggets site (http://www.kdnuggets.com/datasets/index.html), government bureaus of statistics, research centers and other sources. For visual link analysis the Enron organizational e-mail data set is available at http://www.cs.cmu.edu/~enron/.

URL to Code

This selective rather than inclusive sample of tools is divided into two categories: (i) visual data exploration and mining tools, and (ii) general visualization platforms that can be used to develop visual data mining tools.

Visual Data Exploration and Mining

Open Source Xmdv Tool: Source: http://davis.wpi.edu/~xmdv/.

Offers five methods for displaying flat form data and hierarchically clustered data and variety of interactive operations.

GGobi: http://www.ggobi.org/

Offers similar variety of visualization and interaction methods.

VizRank: http://www.ailab.si:8088/supp/bi-vizrank

Offers informative scatter-plot projections in high dimensional class-labeled data.

3DVDM: http://www.inf.unibz.it/dis/projects/3dvdm/download.html

Offers tools for exploring data in 3D virtual reality environments.

Commercial Miner3D: http://www.miner3d.com/

Highly flexible visual data mining tool with many display and interaction methods.

NetMap Analytics: http://www.netmap.com.au/

Tool for visual link analysis. Separate visual analysis and visual presentation modes (the presentation mode bears some similarity to VisualLinks (http://www.visualanalytics.com)).

NetMiner: http://www.netminer.com/

A nice collection of visual metaphors for interactive analysis of graph structures in data sets.

IN-SPIRE: http://in-spire.pnl.gov/- visual text miner.

General Visualization Engines and Development Tools

Open Source VTK: http://www.vtk.org/

The Visualization ToolKit (VTK) software system for 3D computer graphics, image processing, and visualization.

Mondrian: http://moose.unibe.ch/tools/mondrian

Information visualization engine that lets the visualization be specified via a script, hence, can be utilized for development of specialized visual data mining tools.

ParaView: http://www.paraview.org/New/index.html

Visualization platform that supports distributed computational models for processing large data sets

Gapminder: http://www.gapminder.org

Based on the original Trendalyzer tool for knowledge extraction from interactive animation of statistical time-series, the development has been overtaken by Google.

Cross-references

- ▶ Data Mining
- ▶ Human-Computer Interaction
- ▶ Information Visualization
- ▶ Knowledge Discovery in Data Bases
- ▶ Visual Analytics
- ▶ Visual Clustering
- ▶ Visual Information Processing

Recommended Reading

1. Ankerst M. Visual Data Mining. Faculty of Mathematics and Computer Science, University of Munich, Munich, 2000.
2. Chen C. Information Visualization: Beyond the Horizon. Springer, London, 2004.
3. Chittaro L., Combi C., and Trapasso G. Data mining on temporal data: a visual approach and its clinical application to hemodialysis. J. Visual Lang. Comput., 14:591–620, 2003.
4. Demšar U.K. Investigating visual exploration of geospatial data: an exploratory usability experiment for visual data mining. Comput. Environ. Urban., 31:551–571, 2007.
5. de Oliveira F., Crisina M., and Levkowitz H. From visual data exploration to visual data mining: a survey. IEEE T. Vis. Comput. Gr., 9(3):378–394, 2003.
6. Isenberg P., Tang A., and Carpendale S. An exploratory study of visual information analysis. In Proc. SIGCHI Conf. on Human Facters in Computing Systems, 2008.
7. Keim D.A., Mansmann F., Schneidewind J., and Ziegler H. Challenges in visual data analysis. In Proc. Int. Conf. on Information Visualization, 2006.
8. Keim D.A. and North S.C. Visual data mining in large geospatial point sets. IEEE Comput. Graph., 24(5):36–44, 2004.

9. Keim D.A., Sips M., and Ankerst M. Visual data-mining techniques. In Visualization Handbook, Hansen Johnson Elsevier, Amsterdam, 2005, pp. 831–843.
10. Kovalerchuk B. and Schwing J. (eds.). Visual and Spatial Analysis: Advances in Data Mining, Reasoning, and Problem Solving. Springer, Dordrecht, 2004.
11. Niggemann O. Visual Data Mining of Graph-Based Data. Department of Mathematics and Computer Science, University of Paderborn, Paderborn, Germany, 2001.
12. Shneiderman B. Inventing discovery tools: combining information visualization with data mining, In Proc. Discovery Science, 2001, pp. 17–28.
13. Simoff S.J., Böhlen M., and Mazeika A. (eds.). Visual Data Mining: Theory, Techniques and Tools for Visual Analytics. Springer, Heidelberg, 2008.
14. Soukup T. and Davidson I. Visual Data Mining: Techniques and Tools for Data Visualization and Mining. John Wiley & Sons, London, 2002.
15. Thomas J.J. and Cook K.A. Illuminating the Path: The Research and Development Agenda for Visual Analytics. IEEE CS Press, Silver Spring, MD, 2005.

Visual Discovery

▶ Visual Data Mining

Visual Displays of Nonnumerical Data

▶ Visualizing Categorical Data

Visual Displays of Numerical Data

▶ Visualizing Quantitative Data

Visual Formalisms

David Harel, Shahar Maoz
The Weizmann Institute of Science, Rehovot, Israel

Definition

Visual formalisms are diagrammatic and intuitive, yet mathematically rigorous languages. Thus, despite their clear visual appearance, they come complete with a syntax that determines what is allowed, and semantics that determines what the allowed things mean. The main emphasis in the visuality is typically placed on topological relationships between diagrammatic elements, such as encapsulation, connectedness, and adjacency. Geometric and metric aspects, such as size, shape, line-style, and color, may also be part of the formalism. Icons can be used too. Such languages typically involve boxes and arrows, and are often hierarchical and modular. Visual formalisms are typically used for the design of hardware and software systems. This includes structural as well as more complex behavioral specifications.

Historical Background

Two of the oldest examples of visual formalisms are graphs and Venn diagrams, which are both originally due to Euler [7,8]. A graph, in its most basic form, is simply a set of points, called nodes or vertices, connected by lines, called edges or arcs. Its role is to represent a set of elements S and some binary relation R on them. The precise meaning of the relation R is part of the application and has little to do with the mathematical properties of the graph itself. Certain restrictions on the relation R yield special classes of graphs that are of particular interest, such as ones that are connected, directed, acyclic, planar, or bipartite. Graphs are used intensively in all branches of computer science; the elements represented by the nodes range from the most concrete (e.g., physical gates in a circuit diagram) to the most abstract (e.g., complexity classes in a classification scheme), and the edges have been used to represent many kinds of relations, e.g., logical, temporal, causal, or functional. Many of the fundamental algorithms in computer science are related to graphs. Some open problems are related to graphs too, such as the problem of whether graph isomorphism can be decided in polynomial time.

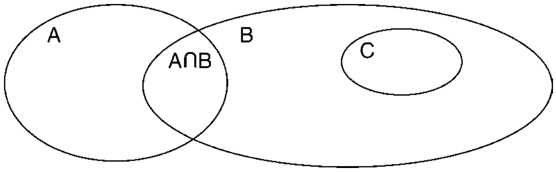
A hypergraph is a generalization of a graph, where the relation R is not necessarily binary (in fact, it need not have a fixed arity). The corresponding edges are called hyperedges. Hypergraphs have applications in database theory (see, e.g., [9]). The information conveyed by a graph (or a hypergraph) is nonmetric, and is captured by the purely topological notion of connectedness; shapes, locations, distances, and sizes, for example, have no significance.

Euler circles, or Venn diagrams (see [8,25] for the original work), represent sets using the fact that simple closed curves partition the plane into disjoint inside and outside regions. They thus give the topological notions of enclosure, exclusion, and intersection the obvious set theoretic meanings of being a subset of, being disjoint from, and having a nonempty intersection with, respectively (see Fig. 1). Typical uses of Venn diagrams include only a small number of circles. However, they can be expanded to *n*-set diagrams (see, e.g., [5]).

A higraph [17] is a formalism that further generalizes graphs, in the spirit of Euler circles. It adds not only hyperedges, but Euler/Venn-like topologically related "blobs" for the nodes, as well as a certain kind of partitioning and adjacency. It thus adds to the relation *R* notions of orthogonality and hierarchy, as well as multilevel and multinode edges (and hyperedges).

The term "visual formalism" appears to be first used in the work on statecharts [16,17], a language for the specification and design of reactive systems. Higraphs [17] are the underlying graphical representation for statecharts.

In general, visual formalisms provide mechanisms for higher levels of abstraction and thus help engineers in coping with the challenges posed by the ever increasing size and complexity of systems. The effectiveness of visual formalisms also relies on the power of human vision and cognitive perception in recognizing and memorizing visual properties, such as topological/spatial relationships between visual entities. They thus provide an appealing representation and communication medium. The advent of modern tools for graphical editing and display has also contributed to the increasing popularity of various visual formalisms among practitioners. Moreover, some tools, indeed go beyond plain editing and syntax correctness checks, and allow their users to take advantage of the rich semantics of some visual formalisms. Thus, some tools have automated procedures for languages that are sufficiently powerful, such as code generation and formal verification. These are directly applied to the visual artifacts and thus indeed exploit their full potential.

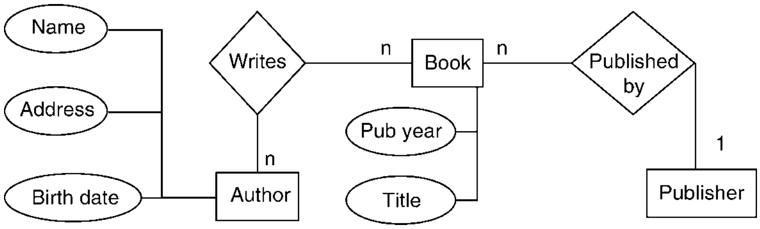


Visual Formalisms. Figure 1. An example of a Venn diagram.

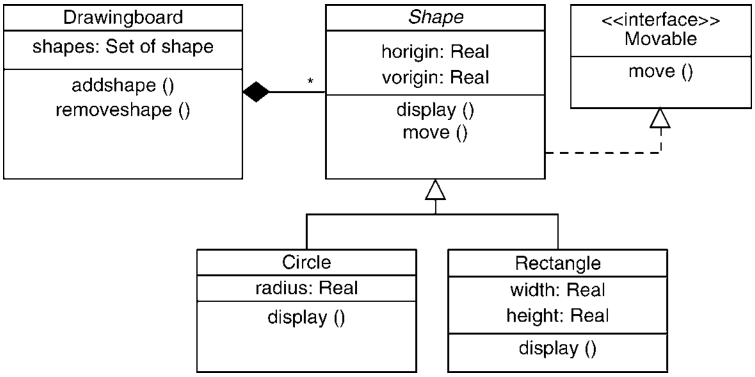
The Unified Modeling Language (UML) [23], an OMG standard formed in the late 1990s, is a collection of visual languages for software and systems specification, defined under a common meta model, and supported by many tools. The standard, however, does not come with satisfactory semantics and hence, by itself, cannot be regarded as a fully fledged visual formalism, though many researchers have provided semantics for many parts thereof.

Foundations

Some widely used visual formalisms include flowcharts, used for the representation of series of actions and decisions, mainly for simple imperative programs (see [14] for what seems to be the original usage thereof, and [12] for a fundamental paper on verification, which was carried out in the framework of flowcharts); data flow diagrams (DFD) (see, e.g., [13]), used to represent the flow of data through a system; entity relationship diagrams (ERD) (Fig. 2, and see [2] for the original work) and their modern application to objects, object model diagrams (OMD) and class diagrams (part of the UML [23], see Fig. 3); message sequence charts (MSC) [21] and their modal extension, live sequence charts [3,19], for inter-object



Visual Formalisms. Figure 2. An example of an entity relationship diagram.



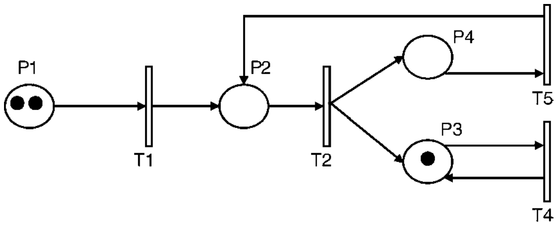
Visual Formalisms. Figure 3. An example of a class diagram.

scenario-based specification; the specification and description language (SDL) [20], for the specification and description of the behavior of reactive and distributed systems; Petri nets (see, e.g., [24], and see Fig. 4); statecharts [16,18] (see Fig. 5); and constraint diagrams [22], used to express logical constraints extended with quantification and navigation of relations.

Statecharts are a prime example of a visual formalism and are therefore shortly discussed below. However, the ideas presented are more general and have indeed been extended to other domains, such as security, databases, and more generally, UML-style behavioral and structural modeling.

In essence, statecharts are finite state machines drawn as state transition diagrams and extended with two key ideas: hierarchy and orthogonality. The hierarchy is introduced using a substating mechanism, and the orthogonality is introduced using a notion of simultaneity. Both are reflected in the graphics themselves, i.e., the hierarchy by encapsulation inspired by Euler/Venn diagrams, and the orthogonality by partitioning blobs into adjacent regions separated by dashed lines.

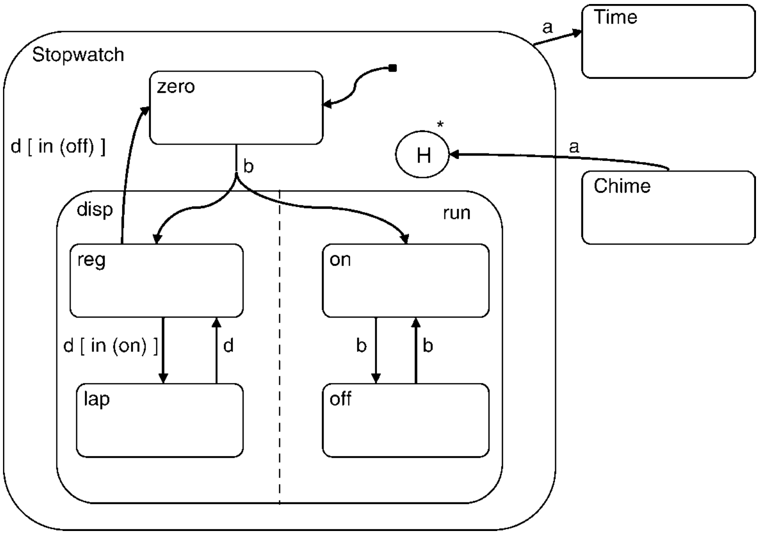
Topological features are a lot more fundamental than geometric ones, in that topology is a more basic branch of mathematics than geometry in terms of symmetries and mappings. One thing being inside another is more basic than it being smaller or larger than the other, or one being a rectangle and the other a circle. Being connected to something is more basic than being green or yellow or being drawn with a thick line or with a thin line. The brain understands topological features given visually much better than it grasps geometrical ones. The mind can see easily and immediately whether things are connected or not, whether one thing encompasses another, or intersects it, etc. (see, e.g., [15], for early work on this).



Visual Formalisms. Figure 4. An example of a Petri net.

Statecharts are not exclusively visual/diagrammatic. Their non-visual parts include, for example, the events that cause transitions, the conditions that guard against taking transitions, and actions that are to be carried out when a transition is taken. For these, statecharts borrow from both the Moore and the Mealy variants of state machines, in allowing actions on transitions between states or on entrances to or exits from states, as well as conditions that are to hold throughout the time the system is in a state.

Unfortunately, some people tend to take diagrams too lightly, finding it difficult to consider a collection of graphics serious or "formal" enough. A common misconception about formality is that textual and symbolic languages are inherently formal, whereas visual and diagrammatic languages are not, as if one could measure formality by how many Greek letters and mathematical symbols the language contains. This myth, together with the deviously deceptive simplicity of graphics, lead to the so called doodling phenomenon – a mindset that says that diagrams are what an engineer scribbles on the back of a napkin but that the real work is done with textual languages.



Visual Formalisms. Figure 5. An example of a statechart.

An interesting topic related to the use of visual formalisms concerns the layout of diagrams, and more generally, their usability and aesthetics. This includes the development of algorithms for the automatic layout of diagrams (see, e.g., [4]), as well as experimental research towards measuring the usability and effectiveness of different visual languages and the development of aesthetic principles.

Another research direction deals with the mechanisms of the language definitions themselves and with the development of new visual languages, for example, domain specific visual formalisms targeted for specific application areas.

Key Applications

Software and Systems Structural Specification

Object model diagrams, and more generally, other structural diagrams, are very popular and are used successfully by software and systems engineers. See the various structural diagrams of the UML [23] and SDL [20].

Reactive Systems Behavioral Specification

Petri nets, statecharts and variants of message sequence charts are successfully used in the design and development of reactive systems. For example, in embedded and real-time safety-critical systems in the automotive, avionics, and telecommunication industries. They are used across the development life cycle of systems, from requirements analysis to specification to simulation to code generation to testing, formal verification, and documentation. In recent years such languages have also been used for the modeling and analysis of biological systems as reactive systems [6,11], which appears to be an application area of great potential.

Data Modeling and Querying

Entity relationship diagrams (ERD) are used for data modeling, or more broadly, for the conceptual specification of databases. Hypergraphs have applications in database theory (see, e.g., [10]). Finally, visual formalisms are also used for the specification of database queries (see [1] for a survey).

Cross-references

▶ Activity Diagrams
▶ Temporal Visual Languages
▶ Visual Query Language

Recommended Reading

1. Catarci T., Costabile M.F., Levialdi S., and Batini C. Visual query systems for databases: a survey. J. Vis. Lang. Comput., 8(2): 215–260, 1997.
2. Chen P.P.-S. The entity-relationship model – toward a unified view of data. ACM Trans. Database Syst., 1(1):9–36, 1976.
3. Damm W. and Harel D. LSCs: Breathing Life into Message Sequence Charts. J. Form. Methods Syst. Des., 19(1):45–80,

2001. Preliminary version in P. Ciancarini, A. Fantechi and R. Gorrieri (eds.). In Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems, 1999, pp. 293–312.

4. Di Battista G., Eades P., Tamassia R., and Tollis I.G. Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall PTR, Upper Saddle River, NJ, USA, 1998.
5. Edwards A.W.F. Cogwheels of the mind: the story of Venn diagrmas. Johns Hopkins University Press, 2004.
6. Efroni S., Harel D., and Cohen I.R. Towards Rigorous Comprehension of Biological Complexity: Modeling, Execution and Visualization of Thymic T Cell Maturation. Gen. Res., 13(11): 2485–2497, 2003.
7. Euler L. Commentarii academiae scientiarum Petropolitanae, Vol. 8. 1741.
8. Euler L. Lettres il une Princesse d'Allemagne, Vol. 2. 1772. letters 102–108.
9. Fagin R. Degrees of acyclicity for hypergraphs and relational database schemes. J. ACM, 30(3):514–550, 1983.
10. Fagin R., Mendelzon A.O., and Ullman J.D. A simplified universal relation assumption and its properties. ACM Trans. Database Syst., 7(3):343–360, 1982.
11. Fisher J., Piterman N., Hubbard E.J.A., Stern M.J., and Harel D. Computational insights into C. elegans vulval development. In Proc. Natl. Acad. Sci., 102(6):1951–1956, 2005.
12. Floyd R.W. Assigning meanings to programs. In J.T. Schwartz (ed.). In Proc. Symposia on Appl. Math., Vol. 19. American Mathematical Society, 1967, pp. 19–32.
13. Gane C.P. and Sarson T. Structured Systems Analysis: Tools and Techniques. Prentice-Hall, Englewood, Cliffs, NJ, 1979.
14. Goldstine H.H. and von Neumann J. Planning and Coding of Problems for an Electronic Computing Instrument. Institute for Advanced Study, Princeton, N.J., 1947. Reprinted in von Neumann's Collected Works, Vol. 5 , A.H. Taub (ed.). Pergamon, London, 1963, pp. 80–151.
15. Green T.R.G. Pictures of Programs and Other Processes, or How to Do Things with Lines. Behav. Inform. Tech., 1:3–36, 1982.
16. Harel D. Statecharts: a visual formalism for complex systems. Sci. Comput. Program., 8:231–274, 1987.
17. Harel D. On visual formalisms. Commun. ACM, 31(5):514–530, 1988.
18. Harel D. and Gery E. Executable object modeling with statecharts. Computer, July 1997, pp. 31–42.
19. Harel D. and Marelly R. Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer, Berlin Heidelberg, New York, 2003.
20. ITU. ITU-T Recommendation Z.100: Specification and Description Language. Technical report, International Telecommunication Union, 1992.
21. ITU. ITU-T Recommendation Z.120: Message Sequence Charts. Technical report, International Telecommunication Union, 1996.
22. Kent S. Constraint diagrams: visualizing invariants in object-oriented models. In Proc. 12th ACM SIGPLAN Conf. on Object-Oriented Programming Systems, Languages & Applications, 1997, pp. 327–341.
23. Object Management Group (OMG). UML: Unified Modeling Language. Available at: http://www.omg.org.
24. Reisig W. Petri Nets: An Introduction, Monographs in Theoretical Computer Science. An EATCS Series, Vol. 4. Springer, Berlin Heidelberg, New York, 1885.
25. Venn J. Symbolic Logic. Macmillan and Co., London, 1881.

Visual Interaction

Maristella Matera
Politecnico di Milano University, Milan, Italy

Synonyms

Visual interaction design; Graphic design

Definition

In the field of Human-Computer Interaction (HCI), *Visual Interaction* refers to the adoption of user interfaces for interactive systems, which make use of visual elements and visual interaction strategies with the aim of supporting perceptual inferences instead of arduous cognitive comparisons and computations. The design of Visual Interaction focuses on the definition of interaction mechanisms through which (i) the users can perform actions on the interactive system by means of visual elements, and (ii) the system can provide feedback to the user, by visually representing the results of the computations triggered by the user actions. The flow of user actions and system feedback over time then has to be coordinated.

In the field of Databases, Visual Interaction refers to the adoption of visual interfaces that provide access to the collection of data stored in databases, by means of visual formalisms and mechanisms supporting both the presentation and the interaction with data.

Historical Background

The information proliferation that characterized the end of the last century led to an increased use of databases by a continuously growing number of users. As an effect of such a trend, the traditional paradigm to access databases, based on formal, text-based query languages, became inadequate for a large portion of inexperienced users.

In 1977, Moshe Zloof proposed QBE (Query By Example) [14], a pioneer query environment where a skeleton of a database relational schema was presented and, based on it, the users were then able to express queries by entering commands, example elements, and selection conditions in visual tables. QBE was the precursor of a new research line, fully emerged during 1990s on the wave of the consensus gained by interactive systems and visual interfaces. In fact, in that period, several research initiatives, both in the HCI and the database field, focused on enhancing the quality of the interaction with databases, by replacing the traditional, text-based query languages with visual paradigms. A new generation of visual tools, the so called Visual Query Systems (VQSs) (see [4] for a survey), was therefore proposed, which promoted visual formalisms and visual interaction mechanisms at the level of both the presentation of and the interaction with the information contained in databases. The most adopted visual representations were forms, diagrams, and icons, each one being used in accordance with the nature of the information to be represented or queried. Some systems also adopted 3D visualizations [1] or virtual reality techniques [12].

Following the first proposals for VQSs, which especially focused on the interaction with structured, record-based information, some visual query languages for XML were also proposed [2,9]. However, it was the Web that led to the major change in the interaction with databases, proposing a novel, easy-to-use visual paradigm, which suddenly allowed a huge number of users to access huge amounts of data distributed across several data sources.

The Web, today, provides the most widely adopted visual interface for accessing databases.

Foundations

In the interaction with database it is possible to recognize at least two different needs: (i) to let the users *understand* the database content, and (ii) to allow the users to *interact* with the content to extract the information they are interested in. From the visual interaction perspective, the main issue is therefore to find effective mechanisms to transmit to the users the information content of the database and to support users during the interaction with such content. This can result, for instance, in using different *visual metaphors*, defined as a mapping between a data model and a visual model [5], both for the understanding phase and for the interaction phase.

Several works focused on the effectiveness and efficiency of visual metaphors. Some of them (see for example [5,10]) dealt with the effectiveness of schema display, trying to define and formalize the notion of adequate metaphors for representing the database schema, thus supporting the understanding phase. Some other works also covered the effectiveness of the interaction phase. For example, in [3] authors pointed out the need for multi-paradigmatic systems, able to provide querying environments in which *adaptive interfaces* exploit several visual representations and interaction mechanisms depending on the user capabilities and tasks. In [11] the author instead specifically focused on the problem of automatically generating effective presentations of query results, adapting the data representation to the nature of the data themselves, and also to the tasks that such data are meant to support.

The majority of these works were characterized by stratified, multi-component architectures supporting the definition of proper visual formalisms. For example, in [10], authors propose a definition of the visual formalisms adopted for schema display characterized by (i) a *data model* that captures schemas, (ii) a *visual model* that captures visualizations, and (iii) a *visual metaphor* that provides the mapping between data and visual models. Such a stratified definition especially aimed at ensuring flexibility for the selection of the most appropriate metaphor for schema display, and also induced the formalization of many interesting properties. For instance, authors precisely defined the concept of correctness of the visual metaphors (no valid visual schema will map to an invalid data schema and vice versa) through a relationship between the constraints in the data model, and those in the visual model.

While the previous work mainly focused on schema display during the understanding phase, in [3] the authors propose a stratified multi-component structure for the architecture of a multi-paradigmatic VQS, supporting both the understanding and the interaction phase. The most notable features were:

1. The provision of *a formalism for expressing databases*, which is able to represent, in principle, a database expressed in any of the most common data models.
2. The construction and the management of *a user model*, which allows the system to propose to the users the most appropriate visual representation according to their skills and needs.

3. A *transformation layer*, defining the translations among the different available representations, in terms of both database content and visual interaction mechanisms, which allows the users to shift from one representation to another according to their preferences.

Given the always increasing relevance of XML-based data descriptions, in late 1990s some visual query languages for XML also started to be proposed, offering the advantage of letting users manipulate visual objects instead of managing the complexity of the underlying XML-base query languages (e.g., XQuery) [2]. However, even if such languages employed visual primitives, the proposed interaction paradigms still preserved a tight correspondence with the traditional (textual) query formulation. Therefore, their advantages have still to be proved.

An incisive step forward in the visual interaction with databases was, however, determined by the growth of the Internet. The World Wide Web suddenly allowed a huge number of users to easily retrieve and access a large amount of information available in different formats – from unstructured multimedia data to traditional relational data. The Web indeed introduced the *information browsing* paradigm, a very powerful (from the user experience point of view) visual interaction mechanism, based on the "one-click" selection of hyperlinks to explore the information space. This innovation immediately provided an easier alternative to the traditional, text-based way to access and query databases. Especially with the advent of *dynamic Web pages*, relational databases were fully integrated into the Web, and Web pages became a composition of "content units," each one corresponding to an application component in charge of querying at runtime the application data source, and displaying the extracted contents in the page. Additionally, search mechanisms were introduced. Form-based interfaces allowed the users to input (even complex) selection conditions that the Web application was able to transform into suitable SQL queries over the application contents; the query results were then returned into a new Web page, dynamically generated on the fly, typically in the form of an index of linked items.

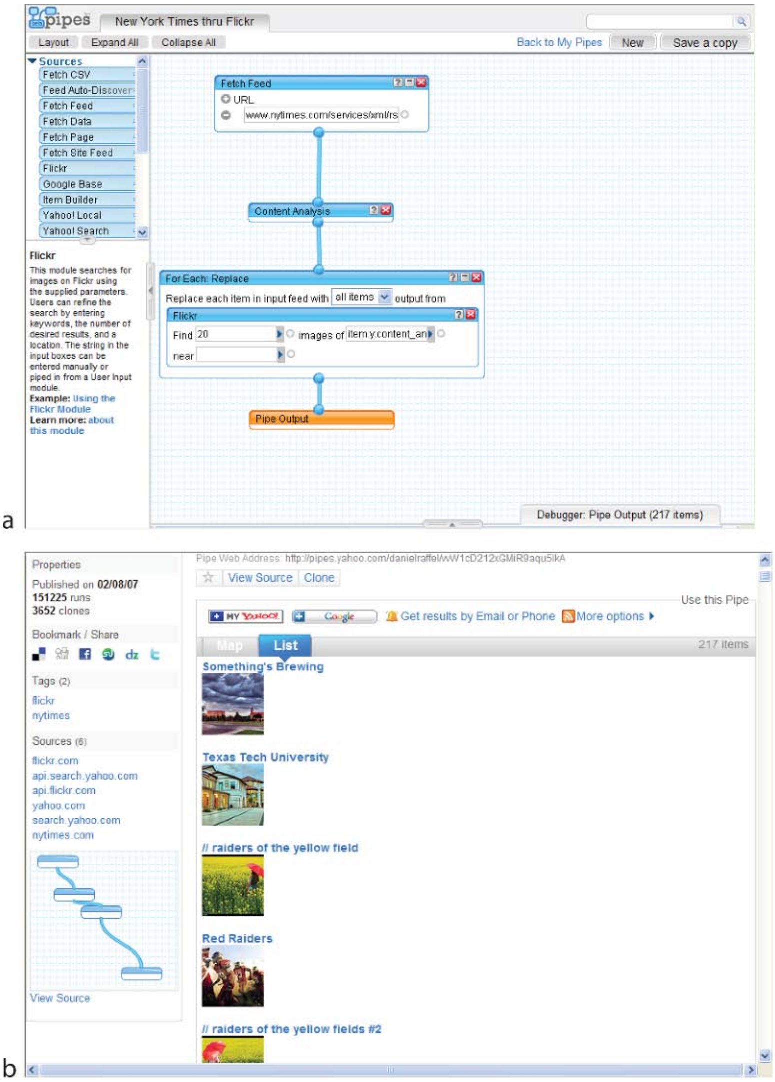
The Web thus definitely became a "de facto" visual interface for databases. The so called "data-intensive" Web applications gained momentum, offering Web interfaces over huge collections of data equipped with both navigation and search mechanisms, as it happened for on-line trading and e-commerce applications, digital libraries, and several other classes of Web applications interfacing large data sources. Such applications share with the initially proposed VQSs a stratified architecture, being characterized by three orthogonal dimensions:

1. The *application data*, collecting the contents to be published by the application.
2. The *hypertext interface*, defining the way in which content units extracted from the application data source are clustered within pages, as well as the navigation mechanisms allowing users to move along the different pages.
3. The *presentation style*, defining layout and graphic properties for the rendering of both contents and navigation mechanisms within pages.

Similarly to VQSs, such a stratified architecture ensures flexibility, by enabling the provision of multiple interfaces over the same content. Hypertext interfaces indeed can be considered *views* over the application data, and therefore several hypertext structures can be constructed over a same database [7] to satisfy the different requirements characterizing different classes of users, different access devices, or even different situations of use. Also, once a hypertext interface is defined, different presentation styles, defining the layout and the graphic properties of the constituent Web pages, can be applied on it. Therefore, especially in the case of adaptive Web applications, several combinations of hypertexts and presentation styles can be adopted to convey contents in the most appropriate modality with respect to the characteristics of the user or of the usage environment, possibly described in a user or a context model [6].

Recently, the so-called *data-driven* methods have been proposed for the design of the hypertext interfaces. They fund their approach on data modeling, and in particular on some properties of the data to be published in the hypertext interface, which can have impact on the effectiveness of the interface itself. In particular, the method proposed in [8] aims to enhance *content accessibility*, which is the ability of a Web application to offer effective and efficient access to its contents, supporting both (i) the identification of the most relevant content entities the application deals with and their mutual relationships (understanding phase), and (ii) the retrieval of the desired content instances (interaction phase). In Latins, "rem tene, verba sequentur" (manage content, words will follow). Rephrasing this motto, the idea behind this method is that if "Rem" is the information globally available to the application, and "Verba" is the hypertext interface enabling information identification and retrieval through navigation and search, one can say that organizing the application content in accordance with some proper modeling abstractions can induce the definition of hypertext interfaces able to effectively support the information access on the Web. Authors therefore propose some data modeling patterns, called *Web Marts*, which suggest a data organization where:

1. Some *core entities* reflect the most important concepts addressed by the application.
2. Some *detail entities* represent additional information that complete the description of the core concepts.
3. Some *access entities* then classify the core contents for facilitating their access.



Flickr

This module searches for images on Flickr using the supplied parameters. Users can refine the search by entering keywords, the number of desired results, and a location. The string in the input boxes can be entered manually or piped in from a User Input module.

Example: Using the Flickr Module

Learn more: about this module

Visual Interaction. Figure 1. An example of visual composition of a Yahoo Pipe (http://pipes.yahoo.com/pipes/) (a) and its corresponding rendering (b). The pipe accesses the New York Times homepage (*Fetch Feed* component), analyzes it (*Content Analysis* component), and uses the extracted keywords to find related pictures at the Flickr Web site (nesting a *For each Replace* component with a *Flickr* component).

By taking into account such data characterization, it is possible to achieve Web interfaces able to highlight the application core concepts, facilitating their retrieval by means of navigation and search mechanism defined on top of access entities, and support the browsing of the core concepts by means of navigation patterns defined on top of detail entities. In other words, the resulting hypertext interfaces are able to effectively support both the understanding of and the interaction with the Web application data.

Key Applications

Nowadays, principles of Visual Interaction design are essential for the construction of any system providing access to databases. Even more, they constitute the basis for the development of modern data-intensive Web applications, especially considering the modern Web 2.0 paradigm that, besides other advantages, also promotes more advanced interaction mechanisms, fully comparable to the mechanisms traditionally adopted for desktop interactive applications.

Future Directions

The current trend in the development of modern Web applications – and in particular of those applications commonly referred to as Web 2.0 applications – clearly points toward a high user involvement in the creation of contents. Users visually interact with Web sites not only as passive actors accessing information, but also as creators of the content instances published by the Web application. The so-called *social applications*, like MySpace, YouTube, or Wikipedia, are indeed living an indisputable success.

However, users are not only more and more being involved in content creation. The phenomenon of *Web mashups* is now emerging to provide Web environments where the users have the opportunity to create online applications starting from contents and functions that are provided by third parties. Typical components that may be mashed up, i.e., composed, are RSS/Atom feeds, Web services, content wrapped from third party Web sites, or programmable APIs (like the one provided by Google Maps). Online mashup tools, like Yahoo Pipes (http://pipes.yahoo.com/pipes/) or the Google Mashup Editor (http://editor.googlemashups.com/), and a number of other academic research efforts (see for example [13]) confirm this trend, which implies the addition of a further level of interaction in the Web: users are not only able to visually access data, but they are also able to visually compose the interfaces providing access to data. Figure 1, for example, shows the Yahoo pipes editor, where a simple application is visually modeled by composing some components that analyze the contents published in the New York Times home page, and then use the retrieved keywords to find related photos at another Web site (www flickr.com).

Since unskilled end users are supposed to “program” the necessary composition logic, special emphasis is therefore currently put on intuitive, visual composition environments, not requiring any programming effort. This seems to be one of the major challenges that will be addressed by future academic and industrial research efforts.

Cross-references

- ▶ Visual Interfaces
- ▶ Visual Metaphor
- ▶ Visual Query Language
- ▶ Web 2.0/3.0

Recommended Reading

1. Boyle J. and Gray P.M.D. Design of 3D metaphors for database visualisation. In Proc. IFIP 2.6 3rd Working Conf. on Visual Database Systems. Lausanne, Switzerland, 1995, pp. 157–175.
2. Braga D., Campi A., and Ceri S. XQBE (XQuery By Example): a visual interface to the standard XML query language. ACM Trans. Database Syst., 30(2):398–443, 2005.
3. Catarci T., Chang S.-K., Costabile M.F., Levialdi S., and Santucci G. A graph-based framework for multiparadigmatic visual access to databases. IEEE Trans. Knowl. Data Eng., 8(3): 455–475, 1996.
4. Catarci T., Costabile M.F., Levialdi S., and Batini C. Visual query systems for databases: a survey. J. Visual Lang. Computing, 8(2):215–260, 1997.
5. Catarci T., Costabile M.F., and Matera M. Which metaphor for which database? In People and Computers X, Proc. HCI’95 Conf., 1995, pp. 151–165.
6. Ceri S., Daniel F., Matera M., and Facca F.M. Model-driven development of context-aware web applications. ACM Trans. Internet Technol., 7(1), 2007.
7. Ceri S., Fraternali P., and Matera M. Conceptual modeling of data-intensive web applications. IEEE Internet Computing, 6(4):20–30, 2002.
8. Ceri S., Matera M., Rizzo F., and Demaldé V. Designing data-intensive web applications for content accessibility using web marts. Commun. ACM., 50(4):55–61, 2007.
9. Comai S., Damiani E., and Fraternali P. Computing graphical queries over XML data. ACM Trans. Inf. Syst., 19(4):371–430, 2001.

10. Haber E.M., Ioannidis Y.E., and Livny M. Foundations of visual metaphors for schema display. J. Intelligent Inf. Syst., 3(3/4):263–298, 1994.
11. Mackinlay J.D. Automating the design of graphical presentations of relational information. ACM Trans. Graphics, 5(2):110–141, 1986.
12. Massari A. and Saladini L. Virgilio: a VR-based system for database visualization. In Proc. Workshop on Advanced Visual Interfaces, 1996, pp. 263–265.
13. Yu J., Benatallah B., Saint-Paul R., Casati F., Daniel F., and Matera M. A framework for rapid integration of presentation components. In Proc. 16th Int. World Wide Web Conference, 2007, pp. 923–932.
14. Zloof M. Query By Example: a database language. IBM Syst. J., 16(4):324–343, 1977.

Visual Interaction Design

▶ Visual Interaction

Visual Interfaces

Tiziana Catarci
University of Rome, Rome, Italy

Synonyms

Graphical user interfaces; GUIs; Direct manipulation interfaces

Definition

Visual Interfaces are user interfaces that make extensive use of graphical objects (icons, diagrams, forms, etc.) that the user may directly manipulate on the screen through several kinds of pointing devices (including her/his fingers) and get an almost instantaneous feedback (near real-time interactivity). Information-intensive interfaces typically exploit one or more visual language, i.e., a language that systematically uses visual signs to convey a meaning in a formal way.

Historical Background

The importance of providing the user with an interactive environment where she/he could directly manipulate the computer content, and get an immediately graspable feedback out of her/his actions, was evident since the first demonstration of the Sketchpad system by Ivan Sutherland [7]. This system was the basis of his 1963 MIT Ph.D. thesis. SketchPad supported manipulation of visual objects using a light-pen, including grabbing objects, moving them, changing size, and using constraints. It contained the seeds of hundreds of important interface ideas. Later on, in the 1970s, many of the interaction techniques popular in direct manipulation interfaces, such as how objects and text are selected, opened, and manipulated, resulted from research at Xerox PARC. The concept of direct manipulation interfaces for everyone was envisioned by Alan Kay of Xerox PARC in a 1977 article about the Dynabook [4]. The first commercial systems to make extensive use of direct manipulation were the Xerox Star (1981) [6], the Apple Lisa (1982) [10], and Macintosh (1984) [9]. Ben Shneiderman at the University of Maryland coined the term "direct manipulation" in 1982, identified the components, and gave psychological foundations [5].

As the quantity of information available became larger and larger, as well as the potential user community of databases and other information sources, there was a growing need for user interfaces that require minimal technical sophistication and expertise by the users, and support a wide variety of information intensive tasks. For instance, information-access interfaces must offer great flexibility on how queries are expressed and how data are visualized.

The general idea was to exploit the well-known high bandwidth of the human-vision channel that allows both recognition and understanding of large quantities of information in no more than a few seconds. Thus, for instance, if the result of an information request can be organized as a visual display, or a sequence of visual displays, the information throughput is immensely superior to the one that can be achieved using textual support. User interaction becomes an iterative query-answer game that very rapidly leads to the desired final result. Conversely, the system can provide efficient visual support for easy query formulation. Displaying a visual representation of the information space, for instance, lets users directly point at the information they are looking for, without any need to be trained into the complex syntax of current query languages. Alternatively, users can navigate in the information space, following visible paths that will lead them to the targeted items. Again, thanks to the visual support users are able to easily understand how to formulate queries, and they are likely to achieve the task more rapidly and less prone to errors than with traditional textual interaction modes.

Foundations

One fruitful sign system conceived by humans for the purpose of storing, communicating and perceiving essential information is based on bidimensional visual signs, like those contained in pictures, photographs, geographic maps. Visual signs are characterized by a high number of sensory variables: size, intensity, texture, shape, orientation, and color, which all concur to provide details about the information to be communicated [1]. For instance, Tufte suggests many different purposes for which color may be employed: "...to label (color as noun), to measure (color as quantity), to represent or imitate reality (color as representation), and to enliven or decorate (color as beauty). In a geographical map, color labels by distinguishing water from stone and glacier from field, measures by indicating altitude with contour and rate of change by darkening, imitates reality with river blues and shadows hatchures, and visually enlivens the topography quite behind what can be done in black and white alone" [8]. Exploiting the multidimensionality of the visual representation allows people to perform in a single instant of perception a visual selection, in the sense that all the meaningful correspondences among the visualized elements are captured. Other constructions, for example a linear text, do not permit this immediate grasping, the entire set of correspondences may only be reconstructed in the user memory. This is also due to the fact that visual representations are processed by human beings primarily in parallel, while text is read sequentially.

Visual interfaces capitalize on all the above characteristics of envisioning the information, yet one can say that the basic paradigms adopted for enhancing visual interaction are:

1. The *direct manipulation* interaction
2. The *metaphor*
3. The *visual representation*

These three concepts, although separately investigated in the literature, at the present state are so strictly correlated to deeply influence each other, and constitute the foundations for the development of visual interfaces. All three concepts are covered in specific entries of this Encyclopedia.

A *visual language* may be defined as a language that systematically uses visual representations to convey a meaning in a formal way. Frequently, people have used visual languages, iconic languages and graphical languages as synonyms. If one follows the above definition they are all visual languages, but it may be more precise to call iconic languages only those using icons and pictures extensively, while diagrammatic (or graphical) languages are those primarily using diagrams (such as graphs, flow-charts, block-diagrams, etc). When speaking about visual languages, some people allude to languages handling visual objects that are visually presented, i.e., images or pictorial objects of which a visual representation is given, others intend languages whose constructs are visual [3]. Chang calls the first ones *visual information processing languages* and the others *visual programming languages.* In the first case, one deals with conventional languages enhanced with subroutine libraries to handle visual objects. Image processing, computer vision, robotics, office automation, etc. are the typical application domains for these languages. Visual programming languages handle objects that do not necessarily have a visual representation. Within the class of visual programming languages, a subclass exists which manages a particular kind of non-visual objects, namely data in databases; this subclass of languages is that of *visual query languages (VQLs),* i.e., query languages exploiting the use of visual representations.

While VQLs are usually adopted for query formulation, information visualization mechanisms are used for displaying the results that constitute the answer to a user request. Information visualization, an increasingly important field of Computer Science, focuses on visual mechanisms designed to communicate clearly to the user the structure of information, and allow her/him to make sense out of large quantities of data. Significant opportunities for information visualization may be found today in data mining and data warehousing applications, which typically access large data repositories. Also, the enormous quantity of information sources on the WWW also calls for visualization techniques.

Key Applications

Any kind of traditional database-related activity (e.g., database creation, querying, mining, updating) has been and could be further supported by the usage of suitable visual interfaces, equipped with effective visual representations, easy-to-use interaction mechanisms and data visualizations. For instance, information-discovery interfaces must support a collaboration between humans and computers, for example, for data mining. Due to humans' limited memory and cognitive abilities, the growing volume of available information has increasingly forced the users to delegate the discovery process to computers, greatly underemphasizing the key role played by humans. Discovery should be viewed as an interactive process in which the system gives users the necessary support to analyze terabytes of data, and users give the system the feedback necessary to better focus its search.

Visual interfaces are obviously also extremely important for novel database applications, dealing with non-traditional data, such as multimedia, temporal, geographical, etc.

Moreover, that the user interface has become a crucial component in the success of any modern (information intensive) application is surely witnessed by the extraordinary growth of the Web: a click on the mouse is all users need to traverse links across the world. However, this much easier access to a huge quantity of various information has created new problems to the users, related to information digestion and assimilation that are difficult to achieve if one lets unstructured floods of data collect in perceptually-rich and information-overabundant displays. The information flood can turn out to be useful only if it can be converted in a somehow structured information flow that users can consume. The Information Foraging theory has been indeed developed in order to evaluate the valuable knowledge gainable from a system in relation to the cost of the necessary interaction.

Future Directions

Issues that are being further explored in modern interfaces include how best to array tasks between people and computers, create systems that adapt to different kinds of users, and support the changing context of tasks. Also, the system could suggest appropriate discovery techniques depending on data characteristics as well as data visualizations, and help integrate what are currently different tools into a homogeneous environment. Modern interfaces use (and will further exploit in the near future) multiple modalities for input and output (speech and other sounds, gestures, handwriting, animation, and video) and multiple screen sizes (from tiny to huge), and have an "intelligent" component ("wizards" or "agents" to adapt the interface to the different wishes and needs of the various users).

However, a key challenge in interacting with current information-abundant electronic environments is to shift from data-centric computer systems, which have been characteristic of over 40 years of Computer Science, to task-centric ones. Indeed, current desktop oriented systems propose a mostly disconnected set of generic tools (word processor, e-mail reader, video and image visualizer, etc.). The fact that these tools are running on one system without being connected leads to awkward situations, such as one re-entering or copying the same data among the different applications. Nowdays, users have to cope with an ever-growing amount of information, which they have to manage and use in order to perform their everyday tasks. This trend forces users to focus more on managing their information rather than using it to accomplish their objectives. Managing information basically amounts to saving it, and possibly being able to find (and re-find) it for subsequent reuse. Given the over-abundance of available information, the process of saving, finding, and re-finding itself needs to be supported by algorithms and tools more powerful than standard OS file browsers. During recent years, the creation of such tools has been the ultimate goal of the so-called Personal Information Management (PIM) field [9]. However, not much has been done in order to cope with the original problem, i.e., to support the user in executing her/his tasks so as to achieve her/his goals. Whereas, it would definitely be beneficial for the user to move from a (static and rigid) object-centric world to a (dynamic and adaptive) task-centric one. Tasks are more of an abstract notion than a computer manageable entity. Hence, such a move requires designing a system that is able to interpret the user's aims and to support the execution of the user's tasks. Tasks need to be explicitly represented in the system both in their static aspects, i.e., the kind of information that they manipulate, the kind of programs involved in these manipulations, security and authentication issues that may arise, and in their dynamic ones, i.e., the sequences of actions that they require, the alternative choices that are given to the user, pre-conditions, post-conditions, and invariants for the various activities that are involved in the task. Both static and dynamic aspects are of direct interest to the user, hence, need to be expressed using explicit semantics that the user can share.

Experimental Results

The key point of any interactive system should be to support, at best, people in achieving their goals and performing their tasks. To be more precise, the interaction should favor an increase in efficiency of people performing their duties without this having to cause extra organizational costs, inconveniences, dangers and dissatisfaction for the user, undesirable impacts on the context of use and/or the environment, long periods of learning, assistance and maintenance. In literature, most of the above listed requirements are synthetically associated to the qualitative software characteristic called usability. Precise usability evaluation needs to be conducted at all stages in the system life cycle in order to: (i) provide feedback which can be used to improve design; (ii) assess whether user and organizational objectives have been achieved; and (iii) monitor long term use of the product or system. In the early stage of design, emphasis is placed on obtaining feedback that can be used as a guide in the design process, while later, when a realistic prototype is available, it is possible to assess whether user and organizational objectives have been achieved. Since in the early stages of design and development changes are less expensive than in later stages, evaluation has to be started as soon as the first design proposals are available. Depending on the development stage of the project, evaluation may be used to select and validate the design options that best meet the functional and user-centered requirements, elicit feedback and further requirements from the users or diagnose potential usability problems and identify needs for improvement in the system. Expert evaluation can be fast and economical. It is good for identifying major problems but not sufficient to guarantee a successful interactive system. Controlled experiments with actual users are a key technique to identify usability issues that might have been missed by the expert evaluation, since they are related with the real usage of the system and the knowledge of the application domain. Generally speaking, evaluation techniques vary in their degree of formality, rigor and user involvement depending on the environment in which the evaluation is conducted. The choice is determined by financial and time constraints, the stage of the development lifecycle and the nature of the system being developed.

Cross-references

▶ Data Visualization
▶ Diagram
▶ Direct Manipulation
▶ Form
▶ Human-Computer Interaction
▶ Icon
▶ Information Foraging
▶ Multimodal Interfaces
▶ Natural Interaction
▶ Result Display
▶ Usability
▶ Visual Metaphor
▶ Visual Data Mining
▶ Visual Query Language
▶ Visual Interaction
▶ Visual Representation
▶ WIMP Interfaces

Recommended Reading

1. Bertin J. Graphics and Graphic Information Processing. Walter de Gruyter, Berlin, 1981.
2. Catarci T., Dong X.L., Halevy A., and Poggi A. Personal Information Management, chap. Structure Everything. University of Washington Press, Seattle, WA, 2008.
3. Chang S.K. Principles of Pictorial Information Systems Design. Prentice-Hall, Englewood Cliffs, NJ, 1989.
4. Kay A. Personal dynamic media. IEEE Comput., 10(3):31–42, 1977.
5. Shneiderman B. Direct manipulation: a step beyond programming languages. IEEE Comput., 16(8):57–69.
6. Smith D.C., Harslem E., et al. The Star user interface: an overview. In Proc. 1982 National Computer Conference, 1982, pp. 515–528.
7. Sutherland I.E. SketchPad: a man-machine graphical communication system. In Proc. AFIPS Spring Joint Computer Conference. ACM, New York, NY, 1963.
8. Tufte E.R. Envisioning Information. Graphics Press, Cheshire, Connecticut, 1990.
9. Williams G. The Apple Macintosh computer. Byte, 9(2):30–54, 1984.
10. Williams G. The Lisa computer system. Byte, 8(2):33–50, 1983.

Visual Interfaces for Geographic Data

Robert Laurini
LIRIS, INSA-Lyon, Lyon, France

Synonyms

Cartography; Visualizing spatial data; Interactive capture; Interactive layout

Definition

Geographic data are in essence visual multimedia data. To enter this data with key-boards and to print them as tables of data are not very practical actions. For entry, special devices exist (theodolites, lasers, aerial photos, satellite images, etc.), whereas visual layout is currently named cartography or mapping. Visual interfaces have two facets, allowing the user to present their output as maps, possibly with very large printers, and to present spatial queries visually.

Historical Background

In the 1970s, the expression used was "computer-aided mapping" to emphasize the idea that printing maps was the sole scope of using computers. Then, in the 1980s, the more important aspect was considered to be structuring of the geographic database, so then the expression "Geographic Information Systems" was coined. From this period, research has been done on presenting maps as well as presenting queries. For centuries cartographers have created a large corpus of rules for manual mapping. Those rules were progressively integrated and enlarged to compose maps. For instance, zone hatching was considered a boring task when manually done, whereas with a computer, this task is straightforward. In contrast, name placement was considered as a relatively easy task, although in computing, this is still a challenge.

Foundations

It is common to say that "approximately 80% of all government data has some geographic component" (Langlois G., "Federal Computer Week," Jan. 08, 2001). This affirmation states the importance of geographic information not only for administration but also for companies, and not only for environmental and urban planning but also for geo-marketing, since all those data are stored in databases, or more specifically, geographic or spatial databases. In addition to non-spatial attributes, geographic objects are characterized by geometric shapes and coordinates, usually in two dimensions, but increasingly with three dimensions and time. Those characteristics imply three consequences:

1. The necessity of special data models for storing.
2. The necessity of distinguishing storage format and layout format.
3. The necessity of specialized interfaces for both querying against databases and presenting the results, essentially by means of cartography.

The scope of this overall presentation is to study those interfaces. But before presenting these interfaces, it is necessary to emphasize that geographic data are not entered via keyboard but are acquired by different devices, such as theodolites, aerial photos, satellite images, and more recently, GPS (Global Positioning System). These devices are essentially characterized by different resolutions and error levels. Just as theodolites can measure objects within accuracy of less than 0.1 mm, some satellite systems or GPS systems have accuracies of 100 m or less.

On the other hand, according to the size of the screens used and the size of the territory to be mapped, different scales must be used. In other words, the resulting map must be simplified or generalized more exactly in order to reach readability and completeness, as it passes from storage format to some other layout format.

This discussion will be organized as follows: after a short introduction, visual interfaces for cartographic output will be examined first, and then interfaces for querying. Then, a small section on new barriers for mobile handheld devices will conclude.

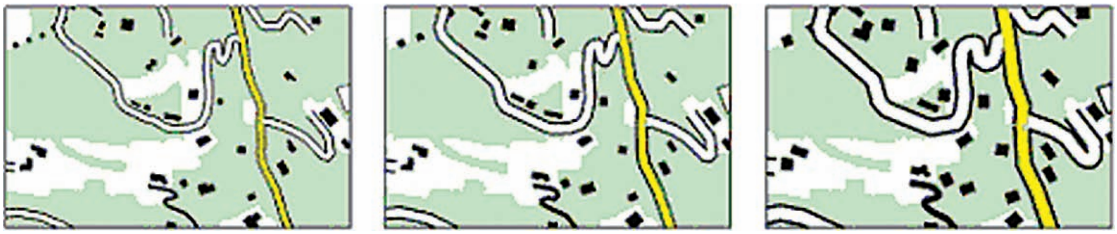
Visual Interfaces for Cartographic Output

A map is the classical way to respond to a spatial query or to layout the results of some spatial analysis. However, over the centuries, cartographers and geographers have elaborated upon sets of knowledge and know-how to make maps. One of the main problems is known as generalization, i.e., the way to simplify a map; for instance, in a geographic database, to have a country that is described with 1 million points/segments, but must be presented in a thumbnail with only 30 points/segments.

Another aspect is called graphic semiology, i.e., the way to select the symbology for mapping.

Generalities A fundamental rule in conventional cartography states that any object, once reduced after scaling to less than 0.1 mm cannot be mapped. For instance, an house or a road that is 10 m wide, at 1:1000 scale will be represented by 1 cm, whereas at 1:100,000 scale they will not be represented at all.

Another rule concerns details to be aggregated. An example would be two houses separated by a road. At some scale they will be represented separately, whereas at other scales they will be aggregated. As a consequence, at some scales a city is seen as a collection of separated houses, then as a set of city-blocks, then as a compact area, and finally as a point.



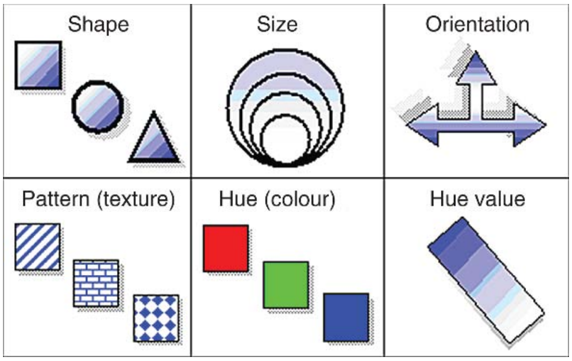
Visual Interfaces for Geographic Data. Figure 1. Example of generalization (1:25000, 1:35000, 1:50000). Source: http://recherche.ign.fr/labos/cogit/arGIGA.php [2].

Generalization By generalization, a piece-wise line can be simplified into a single piece, especially by using the Douglas-Peucker algorithm [3] or variants based on multi-agents systems [5]. Figure 1 gives some examples.

Graphic Semiology Graphic semiology, invented by J. Bertin [1], is the study of the meaning of graphics. In other words, it deals with the signification of drawings, the choice of captions, symbols and icons, together with a methodology to transmit visual messages. Six visual variables have been proposed for representing spatial objects (see Fig. 2) – shape, size, orientation, pattern, hue and value. For instance, to represent a church, a cross symbol (shape) can be used; the symbol's size can be selected according to the initial size of the church, etc.

Animation For some applications, animated cartography can be used, characterized by movement of objects, flickering, mutation, modification of shape or of color, velocity, etc. An excellent example is the animated map for weather forecast, in which some iconized clouds are slowly moving.

Chorems Chorems are a new way of representing schematized territories. Indeed, for several applications, it is not necessary to restitute the complete database contents, but rather to map the more important aspects. Those chorems were usually designed manually, but by means of spatial and spatio-temporal data mining, geographic patterns can be discovered and mapped. In other words, chorems are a new visual representation of geographic database summaries [4] and a way to represent geographic knowledge. The following example (Fig. 3) emphasizes the water problem in Brazil: it is easy to understand that a conventional river map (Fig. 3a) does not show the more crucial aspects as given in Fig. 3b with caption in Fig. 3c.



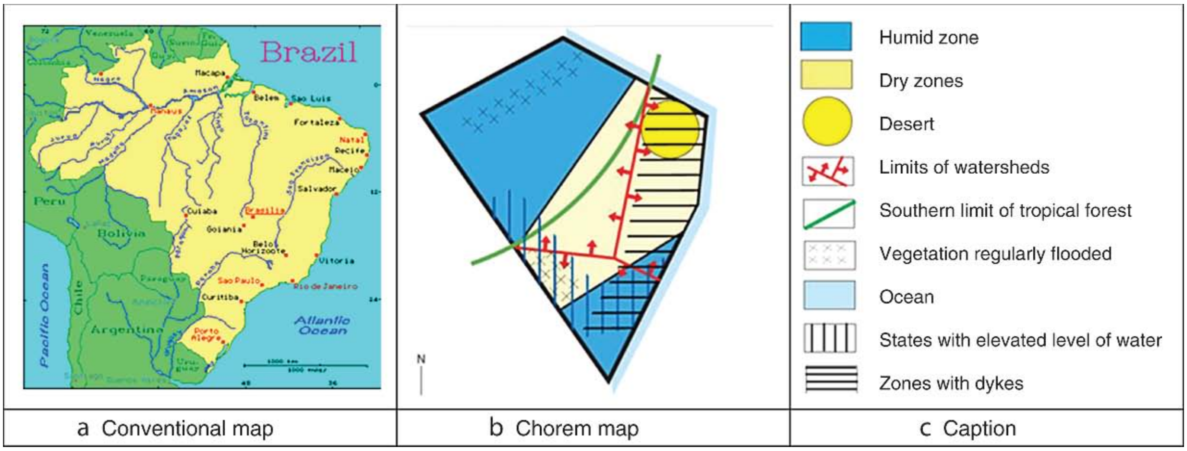
Visual Interfaces for Geographic Data. Figure 2. Bertin's Visual variables. Source: http://atlas.nrcan.gc.ca/site/english/learningresources/carto_corner/vis_var.gif/image_view.

Query Input

Visual interfaces for GIS are not only used for cartography, but also for query input, i.e., by means of interaction. Present systems can be classified into three categories:

1. Textual queries, such as by using spatial extensions of SQL/ORACLE
2. Tabular queries, by means of forms in which some queries are pre-programmed
3. Graphic queries based on widgets such as icons, mice, and clicks

Some basic queries such as point-in-polygon, region or buffer queries are usually given visually and interactively. However, more complex queries are also usually made, such as queries regarding

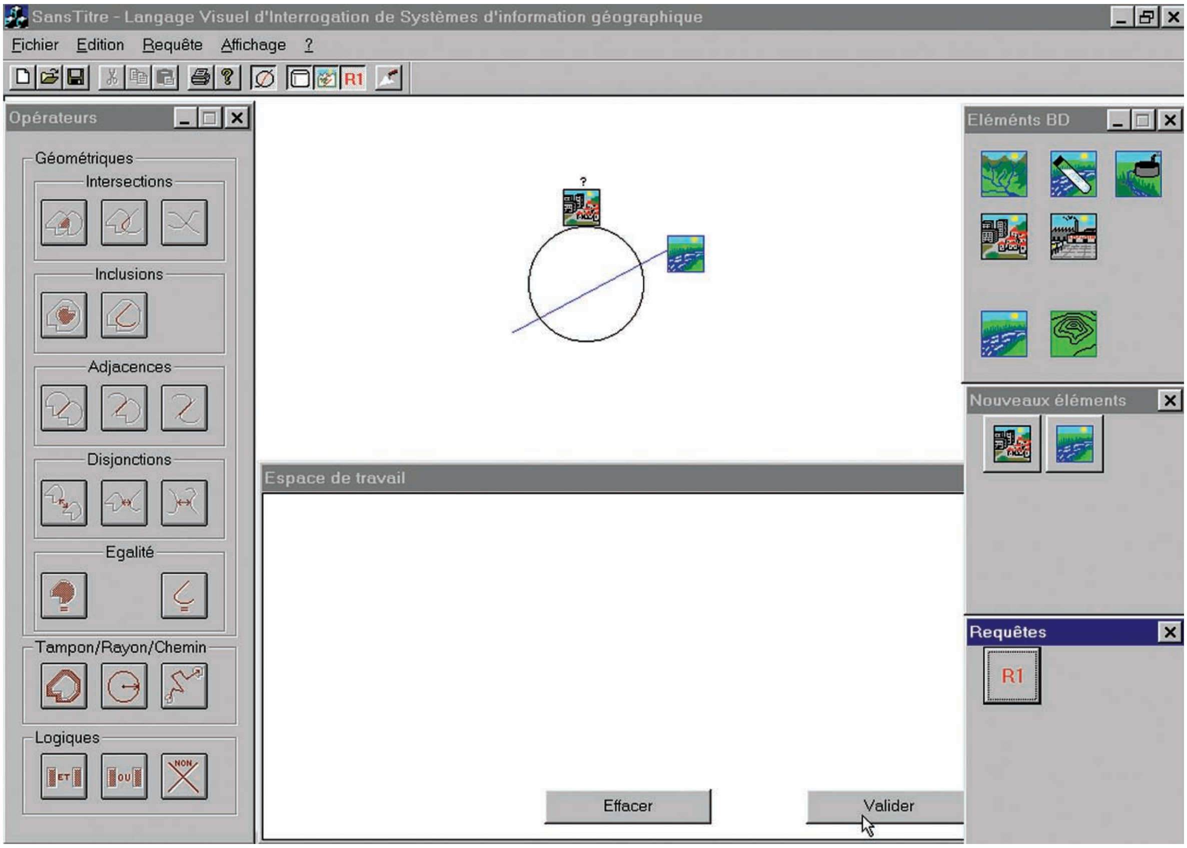


a Conventional map

b Chorem map

c Caption

Visual Interfaces for Geographic Data. Figure 3. The water problem in Brazil using a conventional river map (a) and a chorem map (b) issued from [5].

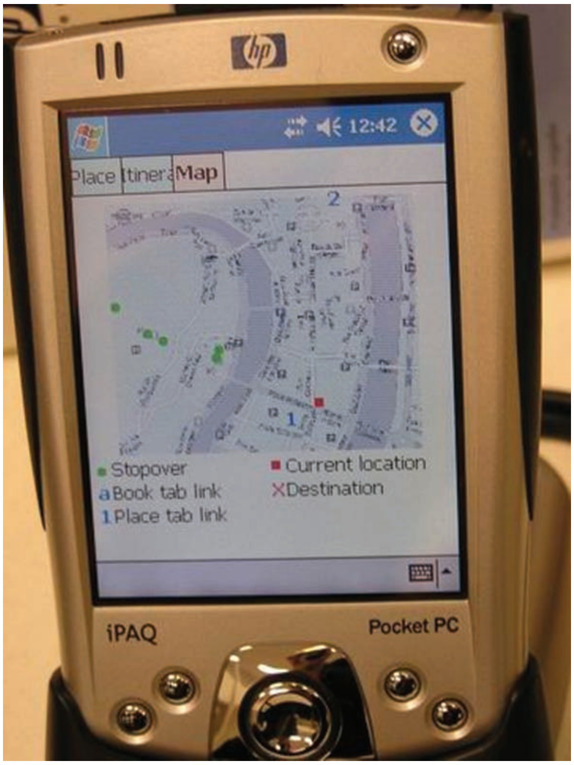


Visual Interfaces for Geographic Data. Figure 4. Example of a visual query asking for "all cities crossed by a river" [6].

intersection or adjacency by means of Egenhofer's spatial relation. For instance, the LVIS system [2] is a visual system based upon those relations (See Fig. 4).

However, a fourth method seems more interesting, based on a tangible table in which several persons can collaborate. Figure 5 shows such a table (from the Geodan Company); see [11] for details.

Visual Interfaces for Geographic Data. Figure 5. Example of a tangible table from the Geodan Company (http://www.geodan.nl).



Visual Interfaces for Geographic Data. Figure 6. Example of a map presented on a mobile device.

Final Remarks: Challenges for Small Mobile Devices

In the early 1970s, the main problem was producing maps, and then increasingly maps were seen as results of spatial queries or as results of spatial analysis techniques. However, as it is simple to lay out maps in conventional screens or in very big screens, the screen size of the new handheld mobile devices requires discovering new modes of representing maps and interacting with them, essentially for Location-Based Services. A very common example is the need to represent the way to go from one location to another location, as shown in Fig. 6.

These new mobile handheld devices will imply new techniques for visualizing geographic data, essentially due to screen size.

Key Applications

Any domains in cartography, from urban to environmental planning, geology, archaeology, real estate mapping, location-based services, etc.

Cross-references

- ▶ Cartography
- ▶ Geographical Databases
- ▶ Geographic Information System
- ▶ Spatial Network Databases
- ▶ Spatial Indexing Techniques
- ▶ Spatial Information System

Recommended Reading

1. Bertin J. Sémiologie graphique. La Haye, Mouton, 1970.
2. Bonhomme C., Trepied C., Aufaure M.A., and Laurini R. A visual language for querying spatio-temporal databases. In Proc. 7th Int. Symp. on Advances in Geographic Inf. Syst., 1999, pp. 34–39.
3. Cécile D. Généralisation Cartographique par Agents Communicants: Le modèle CartACom. PhD Dissertation, University Paris VI, 11/06/2004, 2004.
4. Del Fatto V., Laurini R., Lopez K., Loreto R., Milleret-Raffort F., Sebillo M., Sol-Martinez D., and Vitiello G. Potentialities of chorems as visual summaries of spatial databases contents.

In Proc. 9th Int. Conf. Visual Information Systems, 2007, pp. 537–548.

5. Douglas D. and Peucker T. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The Can. Cartographer, 10(2):112–122, 1973.
6. Kraak M.-J. and Brown A. Web Cartography. CRC, Boca Raton, FL, 2000, 208pp.
7. Kraak M.-J. and Omerling F. Cartography: Visualization of Geospatial Data (2nd edn.). Pearson Education, NJ, 2003, 205pp.
8. Lafon B., Codemard C., and Lafon F. Essai de chorème sur la thématique de l'eau au Brésil. http://histoire-geographie.ac-bordeaux.fr/espaceeleve/bresil/eau/eau.htm, 2005.
9. Laurini R. Information Systems for Urban Planning: A Hypermedia Co-operative Approach. Taylor and Francis, London, 2001, 308pp.
10. Laurini R. and Thompson D. Fundamentals of Spatial Information Systems. Academic Press, San Diego, CA, 1992.
11. van Borkulo E., Barbosa V., Dilo A., Zlatanova S., and Scholten H. Services for an emergency response system in The Netherlands. Second Symposium on Gi4DM, Goa, 2006.

Visual Metaphor

Maria Francesca Costabile[1], Alan F. Blackwell[2]
[1]University of Bari, Bari, Italy
[2]University of Cambridge, Cambridge, UK

Synonyms

Metaphor; Analogy

Definition

Metaphor is a figure of speech, whose essence is to make a comparison between things that are not literally the same. For example, saying "that man is a lion" asks the reader to imagine how a *topic* (the man) could be reinterpreted in terms of some *vehicle* (a lion).

In Graphical User Interfaces (GUIs), *visual metaphor* refers to a kind of analogy, by which designers present the user with an explanation of system behavior in terms of some image. In the early days of the GUI, attempts were made to present all computer behavior in terms of real world analogies, as in the case of the *desktop metaphor*. However, these large scale metaphors soon broke down, as the challenge of developing and maintaining whole systems of correspondence became apparent. Attempts to replicate the success of the desktop metaphor have failed [1]. In practice, UI designers now use the term to refer to visual conventions and genres that, although familiar, need not resemble any real-world objects (e.g., a dialog box or flowchart). The essential benefits of the original desktop metaphor are those provided by *direct manipulation*. Occasionally, real world analogies are successfully used to introduce real world models for new system behaviors (e.g., the shopping basket metaphor in e-commerce). In all cases, user acceptance is encouraged by the use of familiar visual forms, whether pictorial (an *icon* such as a picture of a shopping basket), or an abstract *visual formalism* (conventional layout of dialog boxes, conventional nodes and links for a flowchart).

Key Points

A common view in cognitive science is that all abstract thought is based on physical metaphors [3] (e.g., adding "up"). Computers also use expressions such as "cut and paste" to convey intended usage by analogy to existing technology. Once such figures of speech become widespread, they are unlikely to change. The best principle for visual metaphor design, therefore, is to present users with concepts and terminology that are familiar, whether from the physical world or from previous experience with computers. In creative product design, metaphor can also help designers to imagine novel forms. This more literary strategy is open to interpretation, and may provide less direct guidance to users of the final product.

In the database area, visual formalisms present abstract database concepts to users. GUIs and the growth of database users in the late 80s pushed towards the development of visual query systems, i.e., "systems for querying databases that use a visual representation to depict the domain of interest and express related queries" [2]. A visual representation combines visual formalisms (diagrams, icons, forms) that can be interpreted from experience of existing representation genres – the more familiar and appropriate, the easier it is for the user to understand the usage intended by the designer.

Visual metaphors create expectations about system functionality that, if not fulfilled, can disorientate users. Poor use of metaphor has caused some significant failures of software products. Since there are several ways of creating and interpreting metaphor, the effectiveness of any interface metaphor must be analyzed and empirically evaluated, firstly to ensure that it is consistent with principles of direct manipulation, and secondly to determine whether users recognize and are familiar with the intended visual formalism.

Cross-references

- ▶ Direct Manipulation
- ▶ Visual Formalisms
- ▶ Visual Interaction
- ▶ Visual Query Language
- ▶ Visual Representation

Recommended Reading

1. Blackwell A.F. The reification of metaphor as a design tool. ACM Trans. Comput. Hum. Interact., 13(4):490–530, 2006.
2. Catarci T, Costabile M.F., Levialdi S., and Batini C. Visual query systems for databases: a survey. J. Vis. Lang. Comput., 8:215–260, 1997.
3. Lakoff G. and Johnson M. Metaphors We Live By. University of Chicago Press, Chicago, 1980.

Visual Mining

▶ Visual Clustering

Visual Multidimensional Analysis

▶ Visual On-line Analytical Processing (OLAP)

Visual On-Line Analytical Processing (OLAP)

Marc H. Scholl, Svetlana Mansmann
University of Konstanz, Konstanz, Germany

Synonyms

Visual multidimensional analysis; Interactive visual exploration of multidimensional data

Definition

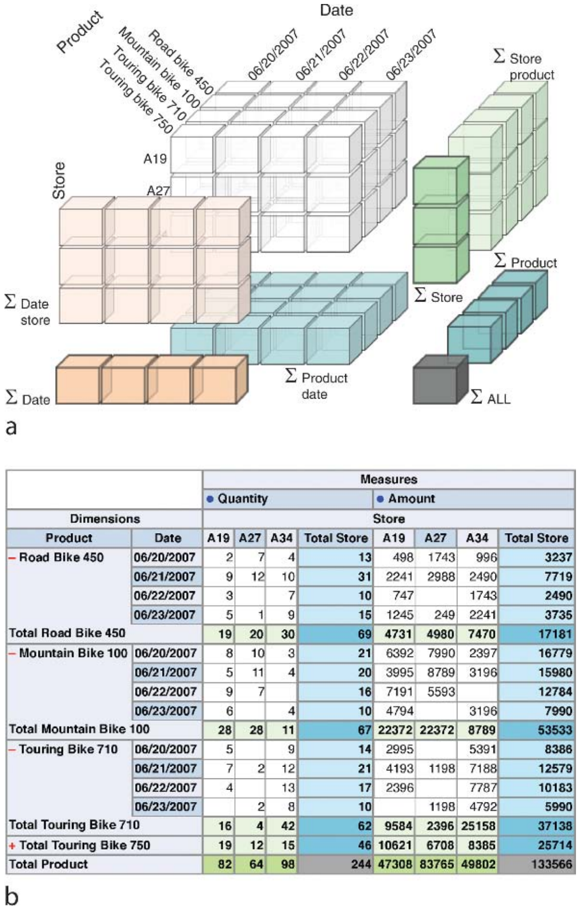
An umbrella term encompassing a new generation of OLAP (On-Line Analytical Processing) end-user tools for interactive ad-hoc exploration of large multidimensional data volumes. Visual OLAP provides a comprehensive framework of advanced visualization techniques for representing the retrieved data set along with a powerful navigation and interaction scheme for specifying, refining, and manipulating the subset of interest. The concept emerged from the convergence of BI (Business Intelligence) techniques and the achievements in the areas of Information Visualization and Visual Analytics. Traditional OLAP frontends, designed primarily to support routine reporting and analysis, use visualization merely for expressive presentation of the data. In the visual OLAP approach, however, visualization plays the key role as the method of interactive query-driven analysis. Comprehensive analysis includes a variety of tasks such as examining the data from multiple perspectives, extracting useful information, verifying hypotheses, recognizing trends, revealing patterns, gaining insight, and discovering new knowledge from arbitrarily large and/or complex data volumes. In addition to conventional operations of analytical processing, such as drill-down, roll-up, slice-and-dice, pivoting, and ranking, visual OLAP supports further interactive data manipulation techniques, such as zooming and panning, filtering, brushing, collapsing etc.

Historical Background

OLAP emerged in the mid-90s as a technology for interactive analysis of large volumes of accumulated and consolidated business data. The underlying multidimensional data model allows users to view data from different perspectives by shaping it into *multidimensional cubes* of measurable facts, or *measures*, as the cube's cells that are indexed by a set of descriptive categories, called *dimensions*. Member values within a dimension may be further arranged into a *classification hierarchy* (e.g., `city` → `state` → `country`) to enable additional aggregation levels. A data cube in a real-world application may hold millions of fact entries characterized by up to 20 dimensions, each supplied with a single or multiple classification hierarchies of various depth. Obviously, to gain insight into such huge and complex data, analysts need mechanisms for projecting the original set onto a two-dimensional (or at most three-dimensional) space and reducing it to a perceivable number of items.

First proposals on using visualization for exploring large data sets were not tailored towards OLAP applications, but addressed the generic problem of visual querying of large data sets stored in a database. Keim and Kriegel [3] proposed VisDB, a visualization system based on a new query paradigm. In VisDB, users are prompted to specify an initial query. Thereafter, guided by a visual feedback, they dynamically adjust the query, e.g., by using sliders for specifying range predicates on single attributes. Retrieved records are mapped to the pixels of the rectangular display area, colored according to the degree of their conforming to the specified set of selection predicates, and positioned according to a grouping or ordering directive.

A traditional interface for analyzing OLAP data is a pivot table, or cross-tab, which is a multidimensional spreadsheet produced by specifying one or more measures of interest and selecting dimensions to serve as vertical (and, optionally, horizontal) axes for summarizing the measures. The power of this presentation technique comes from its ability to summarize detailed data along various dimensions, and arrange the aggregates computed at different granularity levels into a single view, preserving the "part-of" relationships between the aggregates. Figure 1 exemplifies the idea of "unfolding" a three-dimensional data cube Fig. 1a into a pivot table Fig. 1b, with cells of the same granularity marked with matching background color in both representations. However, pivot tables are inefficient for solving non-trivial analytical tasks, such as recognizing patterns, discovering trends, identifying outliers, etc. [1,4,11]. Visualization has the power to save time and reduce errors in analytical reasoning by utilizing the phenomenal abilities of the human vision system to recognize patterns [2].



		Measures							
		● Quantity				● Amount			
Dimensions		Store							
Product	Date	A19	A27	A34	Total Store	A19	A27	A34	Total Store
– Road Bike 450	06/20/2007	2	7	4	13	498	1743	996	3237
	06/21/2007	9	12	10	31	2241	2988	2490	7719
	06/22/2007	3		7	10	747		1743	2490
	06/23/2007	5	1	9	15	1245	249	2241	3735
Total Road Bike 450		19	20	30	69	4731	4980	7470	17181
– Mountain Bike 100	06/20/2007	8	10	3	21	6392	7990	2397	16779
	06/21/2007	5	11	4	20	3995	8789	3196	15980
	06/22/2007	9	7		16	7191	5593		12784
	06/23/2007	6		4	10	4794		3196	7990
Total Mountain Bike 100		28	28	11	67	22372	22372	8789	53533
– Touring Bike 710	06/20/2007	5		9	14	2995		5391	8386
	06/21/2007	7	2	12	21	4193	1198	7188	12579
	06/22/2007	4		13	17	2396		7787	10183
	06/23/2007		2	8	10		1198	4792	5990
Total Touring Bike 710		16	4	42	62	9584	2396	25158	37138
+ Total Touring Bike 750		19	12	15	46	10621	6708	8385	25714
Total Product		82	64	98	244	47308	83765	49802	133566

Visual On-Line Analytical Processing (OLAP). Figure 1. Projecting a multidimensional data cube onto a pivot table. (a) A sample three-dimensional data cube for storing sales transactions as measures `Quantity` and `Amount` characterized by dimensions `Product`, `Date`, and `Store`. (b) A pivot table view of sales data (measures `Quantity` and `Amount`) broken down vertically by `Product` and `Date` and horizontally by `Store`.

OLAP interfaces of the current state of the art enhance the pivot table view by providing a set of popular business visualization techniques, such as bar-charts, pie-charts, and time series, as well as more sophisticated layouts, such as scatterplots, maps, treemaps, cartograms, matrices, grids, etc. and vendors' proprietary visualizations (e.g., decomposition trees or fractal maps). Some tools go beyond mere visual presentation of data and propose sophisticated approaches inspired by the findings in information visualization research. Prominent examples of advanced visual systems are Advizor by Visual Insights [1] and Tableau by Tableau Software [2].

Advizor implements a technique that organizes the data into three perspectives. A perspective is a set of linked visual components displayed together on the same screen. Each perspective focuses on a particular type of analytical task, such as (i) single measure view using a 3D multiscape layout, (ii) multiple measures arranged into a scatterplot, and (iii) anchored measures presented using techniques from multidimensional visualization (box plots, parallel coordinates, etc.).

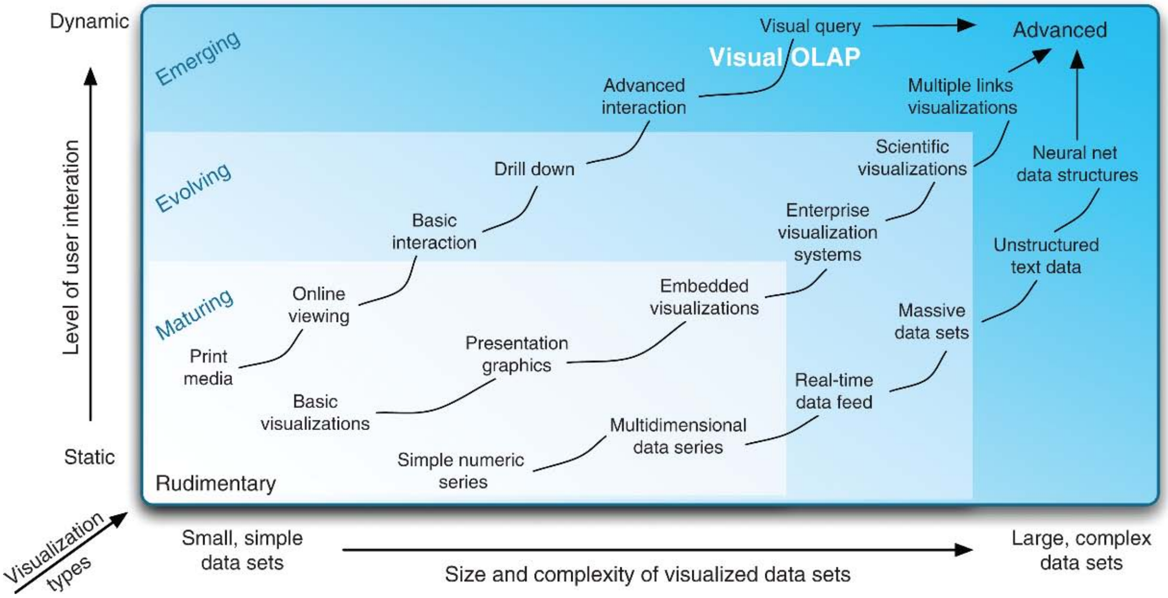
Tableau is a commercialized successor of Polaris, a visual tool for multidimensional analysis developed by a research team of Pat Hanrahan at Stanford University [12]. Polaris inherits the basic idea of the classical pivot table interface that maps aggregates onto a grid defined by dimension categories assigned to the grid's rows and columns. However, Polaris uses embedded graphical marks rather than textual numbers in the table cells. The types of supported graphics are arranged into a taxonomy, comprising rectangle, circle, glyph, text, Gantt bar, line, polygon, and image layouts.

Russom [10] summarizes the trends in business visualization software as a progression from rudimentary data visualization to advanced forms and proposes to distinguish three life-cycle stages of visualization techniques, such as maturing, evolving, and emerging, as depicted in Fig. 2. Within this classification, visual OLAP clearly fits into the emerging techniques for advanced interaction and visual querying.

Ineffective data presentation is not the only deficiency of conventional OLAP tools. Further problems are cumbersome usability and poor exploratory functionality. Visual OLAP addresses those problems by developing fundamentally new ways of interacting with multidimensional aggregates. A new quality of visual analysis is achieved by unlocking the synergy between the OLAP technology, information visualization, and visual analytics.

Foundations

A successful OLAP tool is capable of supporting a wide variety of analytical tasks. As mentioned in the previous



Visual On-Line Analytical Processing (OLAP). Figure 2. Trends in data visualization for business analysis (adopted with modifications from [7]).

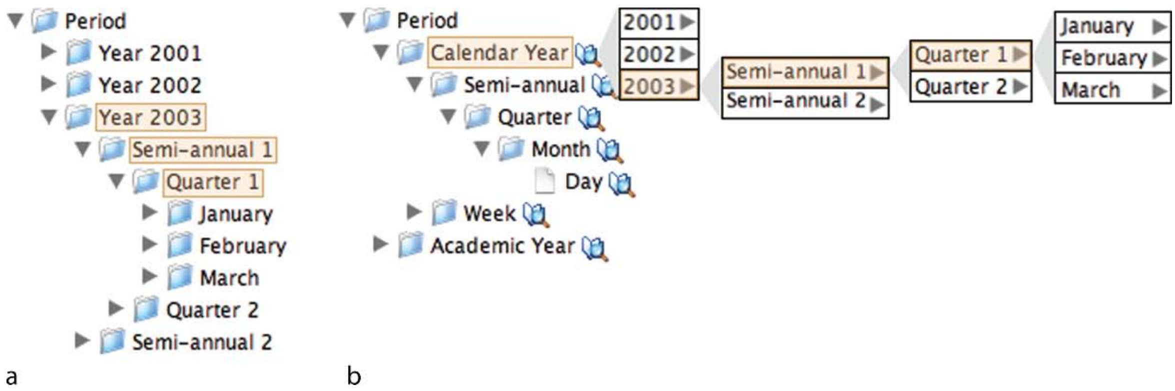
section, different tasks are best solved by applying different visual presentations. OLAP tools account for this diversity by providing a comprehensive framework, which enables users to interactively generate satisfactory visual presentations. The overall query specification cycle evolves by (i) selecting a data source of interest, (ii) choose a visualization technique (e.g., a scatterplot), and then (iii) mapping various data attributes to that technique's structural elements (e.g., horizontal and vertical axes) as well as to other visual attributes, such as color, shape, and size. The main elements of the framework are the navigation structure for visual querying of data sources, a taxonomy of available visual layouts attributes, and a toolkit of interaction techniques for dynamic query refinement and visual representation of the data. A unified framework is obtained by designing an abstraction layer for each element and providing mapping routines (e.g., navigation events to database queries, query results to a visual layout, etc.) implementing the interaction between different layers.

Visual Query Specification

Visual OLAP disburdens the end-user from formulating queries in the "raw" database syntax by allowing purely visual (i.e., by means of using a computer mouse) query specification. Data cubes are represented as a browsable structure whose elements can be queried by "pointing-and-clicking" and "dragging-and-dropping." Visual interface does not trade off advanced functionality for simplicity, it rather facilitates the process of specifying ad hoc queries of arbitrary complexity.

A common navigation paradigm is that of a file browser that represents the contents as a recursive containment of elements. The nodes in the navigation hierarchy may be of types *database*, *schema*, *table* (*cube*), *dimension*, *classification level*, and *measure*. In simplified configurations, the navigation may be limited to a single data cube and, thus, consist solely of dimensions and measures. Dimension hierarchies are presented as recursive nesting of their classification levels, thereby allowing users to browse either directly in the dimension's data or explore its hierarchy schema. In the former approach, denoted *extension-based*, the navigation tree of a dimension is a straightforward mapping of the dimension's data hierarchy: each hierarchical value is a node that can be expanded to access the contained next-level values. Alternatively, the navigation can explicitly display the dimension schema, with each level as a child node of its parent level. This so called *intension-based* approach is especially appreciated for power analysis and employing advanced visualization techniques. The latter navigation strategy becomes the only option when supporting multiple and heterogeneous dimension hierarchies [7] and a multi-cube join or drill-across [6]. Figure 3 shows the difference between instance-based and schema-based browsing for a hierarchical dimension Period.

To navigate in multidimensional aggregates and perform dimensional reduction to extract data for analysis, OLAP defines a number of standard multidimensional analysis operations incorporated into a visual framework in the form of navigation events and interaction options.



Visual On-Line Analytical Processing (OLAP). Figure 3. Browsing options for hierarchical dimensions: extension versus intension navigation. (a) dimension instances arranged in a navigation hierarchy. (b) hierarchy of dimension categories with on-demand data display and an option to switch to extension navigation.

User interactions are translated into valid database queries. From the user's perspective, querying is done implicitly by populating the visualization with data and incrementally refining the view. The output of any OLAP query is a data cube. Visual presentation is generated by assigning the cube's elements – measure values and their dimensional characteristics – to the visual variables of the display. A visualization technique is defined by the visual variables, or attributes, it employs and the way those constructs are combined. Examples of visual attributes are *position*, *shape*, *size*, *color*, and *orientation*. Typically, each of the above visual attributes is used to represent a single data field returned by a query. Various attributes behave differently with respect to the range, data type, and the number of values they can meaningfully represent.

The mapping of the cube's structure to the navigation scheme as well as the translation of navigation events into database queries rely on the metadata of the underlying data warehouse system. Metadata describes the structure of the cubes and their dimensions, measures, and applicable aggregation functions. In an advanced user interface, the analysts are able to define new measures in addition to the pre-configured ones. New measures are obtained by applying a different aggregate function (e.g., average, variance, count) or by specifying a more complex formula over a single or multiple measure attributes (e.g., computing a ratio between two aggregates).

Visualization Techniques

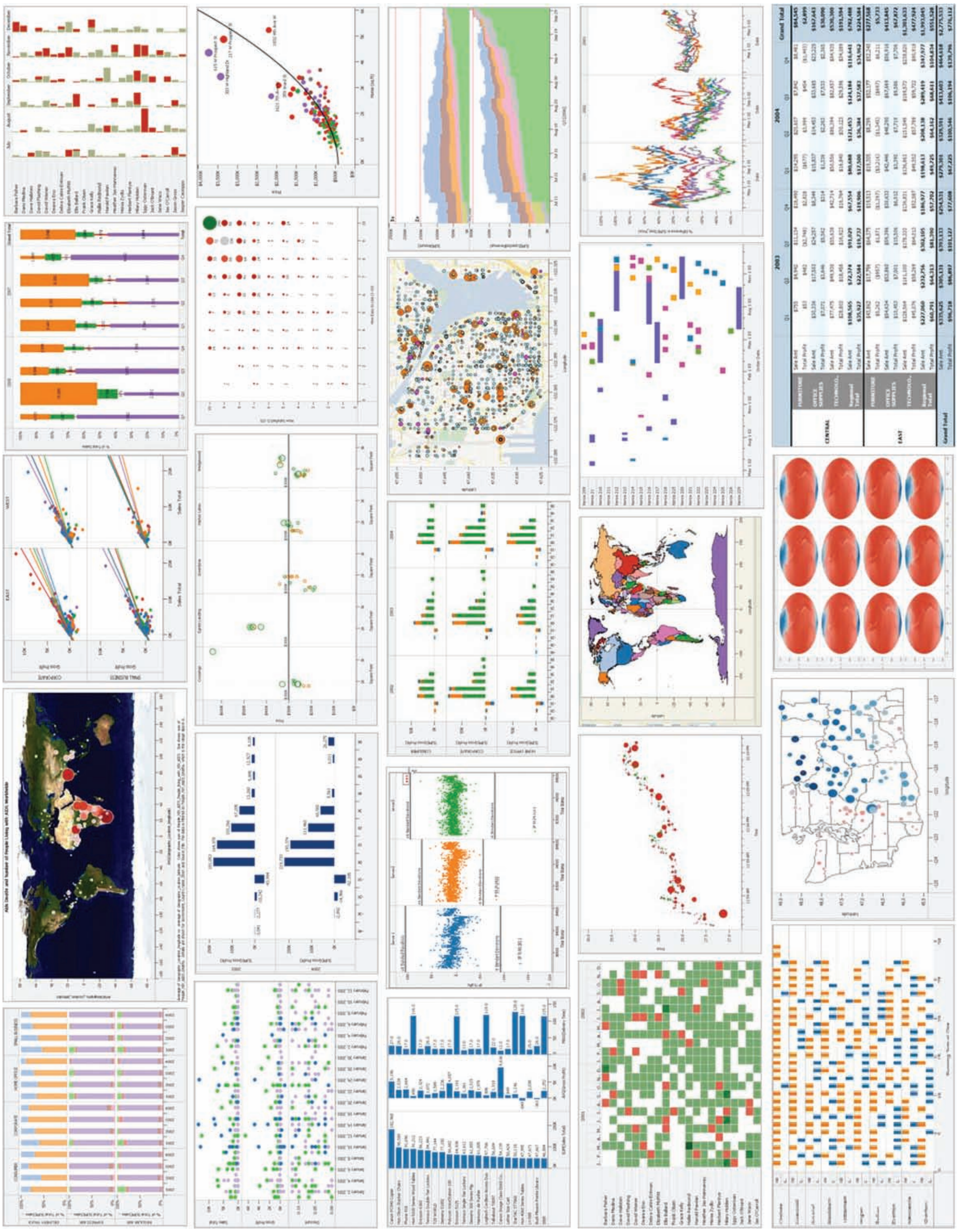
The task of selecting a proper visualization technique for solving a particular problem is by far not trivial as various visual representations (metaphors) may not only be task dependent, but also domain dependent. Successful visual OLAP frameworks need to be based on a comprehensive taxonomy of domains, tasks, and visualizations. The problem of assisting the analyst in identifying an appropriate visualization technique for a specific task is an unsolved issue in state-of-the-art OLAP tools. Typically, a user has to find an appropriate solution manually by experimenting with different layout options.

To support a large set of diverse visualization techniques and to enable dynamic switching from one technique to another, an abstraction layer has to be defined for specifying the relationships between the data and its visual presentation. Maniatis et al. propose an abstraction layer solution, called a *Cube Presentation Model* (CPM) [5], that distinguishes between two layers: a logical layer deals with data modeling and retrieval whereas a presentation layer provides a generic model for representing the data (normally, on a 2D screen). The entities of the presentation layer include points, axes, multicubes, slices, tapes, cross-joins, and content functions. The authors demonstrate how CPM constructs can be mapped to advanced visual layouts at the example of the Table Lens – a technique based on a cross-tabular paradigm with support for multiple zoomable windows of focus.

A common approach to visualization in OLAP application relies on a set of templates, wizards, widgets, and a selection of visual formats. Hanrahan et al. [2] argue, however, that an open set of questions cannot be addressed by a limited set of techniques, and choose a fundamentally different approach for their visual analysis tool Tableau: a declarative visual query language VizQL^{TM} offers high expressiveness and composability by allowing users to create their own visual presentations by means of combining various visual components. Figure 4 illustrates the visualization approach of Tableau by showing just a small subset of sophisticated visual presentations, created using simple VizQL statements, not relying on any pre-defined template layout.

The designers of Tableau deliberately restrict the set of supported visualizations to the popular and proven ones, such as tables, charts, maps, and time series, doubting general utility of exotic visual metaphors [2]. Thereby, Tableau's approach is constrained to generating grids of visual presentations of uniform granularity and limited dimensionality. Other researchers suggest that visual OLAP should be enriched by extending basic charting techniques or by employing novel and less known visualization techniques to take full advantage of multidimensional and hierarchical properties of the data [4,11,14,15]. Tegarden [15] formulates the general requirements of business information visualization and gives an overview of advanced visual metaphors for multivariate data, such as *Kiviat diagrams* and *Parallel Coordinates* for visualizing data sets of high dimensionality, as well as 3D techniques, such as *3D Scattergrams*, *3D line graphs*, *floors and walls*, and *3D map-based bar-charts*.

Another branch of visualization research for OLAP concentrates on developing *multiscale* visualization techniques capable of presenting the data at different levels of aggregation. Stolte et al. describe their



Visual On-Line Analytical Processing (OLAP). Figure 4. Examples of sophisticated multidimensional visualizations generated by simple VizQL statements (used by permission of Tableau Software, Inc.).

implementation of multiscale visualizations within the framework of the Polaris system [13]. The underlying visual abstraction is that of a *zoom graph* that supports multiple zooming paths, where zooming actions may be tied to dimension axes or triggered by a different type of interaction.

Lee and Ong propose a multidimensional visualization technique that adopts and modifies the Parallel Coordinates method for knowledge discovery in OLAP [4]. The main advantage of this technique is its scalability to virtually any number of dimensions. Each dimension is represented by a vertical axis and the aggregates are aligned along each axis in form of a bar-chart. The other side of the axis may be used for generating a bar-chart at a higher level of detail. Polygon lines adopted from the original Parallel Coordinates technique are used for indicating relationships among the aggregates computed along various dimensions (a relationship exists if the underlying sets of fact entries in both aggregates overlap).

Mansmann and Scholl concentrate on the problem of losing the aggregates computed at preceding query steps while changing the level of detail and propose to use hierarchical layouts for capturing the results of multiple decompositions within the same display [9]. The authors introduce a class of multiscale visual metaphors called *Enhanced Decomposition Tree*: the levels of the visual hierarchy are created by decomposing the aggregates along a specified dimension and the nodes contain the resulting sub-aggregates arranged into an embedded visualization (e.g., a bar-chart). Various hierarchical layouts and embedded chart techniques are considered to account for different analysis tasks.

Sifer presents a multiscale visualization technique for OLAP based on *coordinated views of dimension hierarchies* [11]. Each dimension hierarchy with qualifying fact entries attached as the bottom-level nodes is presented using a space-filling nested tree layout. Drilling-down and rolling-up is performed implicitly by zooming within each dimension view. Filtering is realized by (de-)selecting the values of interest at any level of dimension hierarchies, resulting either in highlighting the qualifying fact entries in all dimension views (*global context coordination*) or in eliminating the disqualified entries from the display (*result only coordination*).

A similar interactive visualization technique, called the *Hierarchical Dynamic Dimensional Visualization* (HDDV), is proposed in [14]. Dimension hierarchies are shown as hierarchically aligned barsticks. A barstick is partitioned into rectangles that represent portions of the aggregated measure value associated with the respective member of the dimension. Color intensity is used to mark the density of the number of records satisfying a specified range condition. Unlike in [11], dimension level bars are not explicitly linked to each other, allowing to split the same aggregate along multiple dimensions and, thus, to preserve the execution order of the dis-aggregation steps.

Key Applications

Visual OLAP should be considered an integral part of a BI architecture. The latter appears rather universal with respect to prospective application domains and comprises virtually all business and non-business scenarios that require quantitative analysis. Prominent business application areas are Financial Risk Management, Industrial Process Control, Operations Planning, Capital Markets Management, Network Monitoring, Marketing Analysis, Fraud/Surveillance Analysis, Portfolio Management, Customer/Product Analysis, Budget Planning, Operations Management, and Economic Analysis. Non-business applications are found primarily in government, healthcare, research, and academia.

Future Directions

At present, visual OLAP is lacking a unified formal model and query specification standard. Existing visual analysis frameworks are based on proprietary formalisms and models. Furthermore, advanced OLAP tools claim to turn visualization from the presentation layout into the method of data exploration. This claim implies the need for re-defining visualization as an instrument in terms of its structural components and interaction functions.

A pioneering initiative on addressing the above formalization issues by integrating visualization into a query language is a visual declarative data query language VizQL^{TM} developed at Tableau Software Inc. and released in 2006 [2]. However, VizQL is a proprietary solution and is limited to the visual table paradigm of the Tableau system. To be universally adopted, a new standard for visual exploration of OLAP data has to be open, flexible, and extendible to account for a wide range of visualization approaches.

A promising research direction is to evaluate a wealth of existing visualization techniques with respect to their applicability to multidimensional analysis and to identify classes of techniques efficient for solving particular analysis tasks. One of the major visualization challenges for OLAP is the ability to present a large number of dimensions on a display. An additional visual attribute for mapping a dimension could be *animation*, as found in Gapminder software for interactive data exploration using animated scatterplots where animation is used to show the evolution of values along the timeline [9].

Another emerging research direction is concerned with spatio-temporal analysis that employs specialized techniques for spatial and/or temporal exploration, such as maps, cartograms, times series, and calendar views. These techniques are aware of the rich semantics behind temporal and geographic dimensions of the data. Rivest et al. [8] propose SOLAP (spatial OLAP) as a visual platform for spatio-temporal analysis using cartographic and general displays. The authors define different types of spatial dimensions and measures as well as a set of specialized geometry-aware operators.

Cross-references

- ▶ Business Intelligence
- ▶ Cube
- ▶ Data Visualization
- ▶ Dimension
- ▶ Hierarchy
- ▶ Measure
- ▶ Multidimensional Modeling
- ▶ On-Line Analytical Processing
- ▶ Visual Interfaces

Recommended Reading

1. Eick S.G. Visualizing multi-dimensional data. ACM SIGGRAPH Comput. Graph., 34(1):61–67, 2000.
2. Hanrahan P., Stolte C., and Mackinlay J. Visual analysis for everyone: understanding data exploration and visualization. Tableau Software Inc., 2007. White Paper, http://www.tableau-software.com/docs/Tableau_Whitepaper.pdf
3. Keim D.A. and Krigel H.-P. VisDB: database exploration using multidimensional visualization. IEEE Comput. Graph. Appl., 14(5):40–49, 1994.
4. Lee H.-Y. and Ong H.-L. A new visualisation technique for knowledge discovery in OLAP. In Proc. First Pacific-Asia Conference on Knowledge Discovery and Data Mining, 1997, pp. 279–286.
5. Maniatis A.S., Vassiliadis P., Skiadopoulos S., and Vassiliou Y. Advanced visualization for OLAP. In Proc. ACM 6th Int. Workshop on Data Warehousing and OLAP, 2003, pp. 9–16.
6. Mansmann S. and Scholl M.H. Exploring OLAP aggregates with hierarchical visualization techniques. In Proc. 2007 ACM Symp. on Applied Computing, 2007, pp. 1067–1073.
7. Mansmann S. and Scholl M.H. Extending visual OLAP for handling irregular dimensional hierarchies. In Proc. 8th Int. Conf. Data Warehousing and Knowledge Discovery, 2006, pp. 95–105.
8. Rivest S., Bédard Y., and Marchand P. Toward better support for spatial decision making: defining the characteristics of spatial on-line analytical processing (SOLAP). Geomatica, 55(4): 539–555, 2001.
9. Rosling H., Rönnlund A.R., and Rosling O. New software brings statistics beyond the eye. In Proc. Organisation for Economic Co-operation and Development, 2006, pp. 522–530.
10. Russom P. Trends in Data Visualization Software for Business Users. DM Review, May 2000.
11. Sifer M. A visual interface technique for exploring OLAP data with coordinated dimension hierarchies. In Proc. Int. Conf. on Information and Knowledge Management, 2003, pp. 532–535.
12. Stolte C., Tang D., and Hanrahan P. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. IEEE Trans. Visual. Comput. Graph., 8(1):52–65, 2002.
13. Stoltec C., Tang D., and Hanrahan P. Multiscale visualization using data cubes. IEEE Trans. Visual. Comput. Graph., 9 (2):176–187, 2003.
14. Techapichetvanich K. and Datta A. Interactive visualization for OLAP, Part III. In Proc. Int. Conf. on Computational Science and its Applications, 2005, pp. 206–214.
15. Tegarden D.P. Business information visualization. Comm. AIS, 1(1), 1999, Article 4.

Visual Perception

Silvia Gabrielli

Bruno Kessler Foundation, Trento, Italy

Synonyms

Sight; Vision

Definition

In psychology, visual perception is the ability to transform visible light stimulus reaching the eyes into information supporting recognition processes and action. The various physical and processing components which enable a human to being to assimilate information from the environment are known as the visual system.

The act of seeing starts when the cornea and lens at the front of the eye focus an image of the outside world onto a light-sensitive membrane in the back of the eye, called the retina. The retina is actually the part of the brain which works as a transducer for the conversion of patterns of light into neuronal signals. Precisely, the photoreceptive cells of the retina detect the photons of light and respond by producing neural impulses. These signals are processed in a hierarchical fashion by different parts of the brain, such as the lateral geniculate nucleus, and the primary and secondary visual cortex of the brain.

The major problem in visual perception is that what is seen is not simply and always a translation of retinal stimuli (i.e., the image on the retina). Thus, different theories and experimental studies have been devised to explain what visual processing does to create what is actually perceived. It is worth stressing that visual perception involves the acquisition of knowledge, so it is not merely an optical process but it also entails cognitive activity.

Historical Background

The foundations of modern theories of vision were laid in the seventeenth century, when Descartes and others established the principles of optics, making it possible to distinguish between the physical properties of light, images and the psychological properties of the visual experience. It was proposed that visual perception involves adding information to that present in the retinal image in order to reach properties such as solidity of an object and meaning.

According to the empiricist position, developed primarily by von Helmholtz [8], vision originates from a form of unconscious inference: it is a matter of deriving a probable interpretation from incomplete data (a set of elementary sensations that are integrated and synthesized through a process of learning by association). Inference requires prior assumptions about the world, such as that light comes from above, or that objects are viewed from above and not below. The study of visual illusions (cases when the inference process goes wrong) has yielded a lot of insight into what sort of assumptions the visual system makes.

The unconscious inference hypothesis has recently been investigated in Bayesian studies of visual perception. Proponents of this approach consider that the visual system performs some form of Bayesian inference to derive a perception from sensory data. Models based on this idea have been used to describe various visual subsystems, such as the perception of motion or the perception of depth.

Gestalt psychologists (in the 1930s and 1940s), questioned the assumption that vision is derived from inference mechanisms and previous knowledge [10]. They considered visual perception to be direct and based on the detection of patterns or configurations, as organized wholes available in the perceptual field. According to the Gestalt *Laws of Organization* there are six main factors that determine how things are grouped in visual perception: proximity, similarity, closure, symmetry, common fate and continuity. The major problem with the Gestalt laws and theory is that they are *descriptive* of vision more than *explanatory.*

In Gibson's ecological theory of perception [4], the emphasis is placed on the role played by relations in the visual environment, which are embedded within the spatial and temporal distribution of stimuli an active observer can perceive. According to Gibson, no inference mechanism is needed to detect these *affordances*, that are directly available in the surrounding environment.

Computational approaches to the study of vision, such as Marr's work [5], have provided more detailed explanations of visual phenomena by building artificial intelligence models of the processes involved. Vision is considered as the process of forming a description or representation of what is in the scene from the retinal images. The system at work is modular and serial, consisting of a number of subprocesses, each one taking one representation and transforming it into another. It goes from the creation of the primal sketch (representing changes in light intensity occurring over space in the image and organizing these local descriptions onto a 2D representation of regions and boundaries) to the 2 1/2 D sketch (where the layout of objects surfaces, their distances and orientations relative to the observer are represented) to the creation of 3D model representations (specifying the solid shapes of objects matched against their corresponding representations in memory).

Foundations

According to a multidisciplinary study of visual perception based on perceptual psychology, neuroscience and computational analysis, the purpose of vision is to produce information about objects, locations and events in the world from imaged patterns of light reaching the viewer. Psychology uses the term 'distal stimulus' to refer to the physical world under observation and 'proximal stimulus' to refer to the retinal image. The function of vision is to create a description of aspects of the distal stimulus given the proximal stimulus. Although visual perception is said to be veridical when it produces accurate descriptions of the real world, in practice vision is better understood in the context of the cognitive and motor functions that it serves.

Vision systems create descriptions of the visual environment based on properties of the incident illumination. The human vision system is primarily sensitive to patterns of light rather than to the absolute magnitude of light energy. The eye does not operate as a photometer. Instead, it detects spatial, temporal and spectral patterns in the light imaged on the retina, and information about these patterns of light form the basis of visual perception. A system which measures changes in the light energy rather than the magnitude of energy has an ecological utility, since it makes it easier to detect patterns of light over large ranges in light intensity. This is also an advantage for computer graphics, which can make graphic displays work effectively by only producing similar patterns of spatial and temporal change to the real world.

Depth perception: Human beings can perceive the world as 3D, although images projected onto the retina are 2D (all points placed at different distances in space, but along the same directional axis, project onto a same point of the retina). Depth perception, also called stereopsis, relies mainly on binocular cues (cues that require input from both eyes) and on monocular cues (cues available from the input of just one eye), as well as on the synthetic integration performed by a person's brain based on the full field of view perceived with both the eyes.

Among the most important binocular cues there are: (i) retinal disparity, due to the distance between the two eyes which makes the projection of objects or scenes onto each retina slightly different. By using two images of the same scene obtained from slightly different angles, it is possible to triangulate the distance to an object with a high degree of accuracy. If an object is far away, the disparity of that image falling on both retinas will be small, if it is close the disparity will be large. (ii) accommodation, focusing on far away objects, the ciliary muscles stretch the eye lens, making it thinner. (iii) convergence, to focus on a same object the two eyes need to converge. The angle of convergence is larger when the eye is fixating objects that are far away.

Examples of monocular cues are: (i) focus, the lens of the eye can change its shape to bring objects at different distances into focus. Knowing at what distance the lens is focused when viewing an object means knowing the approximate distance to that object. (ii) perspective, the property of parallel lines converging at infinity allows people to reconstruct the relative distance of two parts of an object, or of landscape features. (iii) occlusion, the blocking of the sight of objects by others is also a clue which provides information about relative distance (the occluded object is perceived as more far away). (iv) peripheral vision, at the outer extremes of the visual field, parallel lines become curved, as in a photo taken through a fish-eye lens. This effect greatly enhances the viewer's sense of being positioned *within* a real, three dimensional space.

Perceptual constancies: To view the world by the exact image projected on the retina would mean to perceive it as very unstable. Objects within a person's field of vision would constantly be changing shape, size, position and color as s/he moved toward and away from them, because the distance and therefore the amount of reflected light detected by the retina would also be changing. Indeed, this is not the case, our surroundings are perceived as solid and stable since the world is not just "seen" but actively constructed from fragmentary perceptual data [4]. It is our brain that by means of perceptual constancies interprets the changing proximal stimulation to reach stability. For instance, in the case of *size* constancy our brain perceives an object in relation not only to its visual angle, but also to the perceived distance. Also, size constancy tells us that objects moving away from us are the same size even though their retinal image is getting smaller and this is because size constancy is a property of the perceptual field (it refers to the relationship between the object and its surrounding context, which remains stable). The same principle works in *form* constancy, where an object is perceives as the same notwithstanding the different shapes that are projected onto the retina as its angle changes. That is to say that perception of the partial view becomes equivalent to perception of the whole object. Also, in the case of *color* and *light* constancies, our retina is able to distinguish the different wavelengths of light entering the eye, however, our perception remains relatively stable since it is affected by our previous experiences or knowledge, as well as by the context one is dealing with (e.g., color and lightness of an object do not depend only on the absolute intensity of light, but also on their relationships with the stimulation arriving from the surrounding areas).

There are primarily two different ways in which visual perception and the perceptual process have been scientifically investigated. Psychophysical analysis has studied how a person's perception is related to the stimulus. As an example, *Stevens' power law* defines the relationship between the magnitude of a physical stimulus and its perceived intensity or strength [7]. The general form of the law is

$$\psi(I) = kI^a,$$

where I is the magnitude of the physical stimulus, ψ is the psychophysical function capturing sensation (the subjective size of the stimulus), a is an exponent that depends on the type of stimulation and k is a proportionality constant that depends on the type of stimulation and the units used.

A second way of studying perception considers its relation to the physiological processes that are occurring within the person's sensors and/or brain. This is called *physiological analysis* of the perceptual process, which measures, for example, some aspects of a person's brain activity when s/he looks at a visual stimulus. Magnetic resonance imaging (MRI) is a typical instrument used for mapping activation patterns in the human brain as a person watches a scene.

Due to the complexity of visual perception, and the intimate relation among its psychophysical, physiological and cognitive aspects, the best way of getting a complete view of this phenomenon is by cross referencing the different disciplines that have addressed its study.

Key Applications

Computer Graphics

Visual perception has become a key component of computer graphics, since it helps to achieve the development of high fidelity virtual environments in reasonable time by exploiting knowledge of how the human visual system works (e.g., rendering time can be saved by avoiding to compute those parts of a scene that the human will fail to notice).

Artificial Intelligence

The results of studies on human visual perception inspire the development of computational models of vision that, in turn, can be used to design and implement visual abilities within artificial intelligence systems (e.g., robots).

Visual Interfaces

Visual perception provides the scientific basis for the design of visual interfaces that can best match the requirements of their users, by enabling an easier and more transparent access and interaction with information.

Information Visualization

In the field of Information visualization, visual perception principles help to produce effective visualizations of non-geometric data retrieved from large document collections (e.g., digital libraries), the World Wide Web, and databases, with the aim of supporting users to make sense of information there contained and of enhancing their creative thinking.

Future Directions

Current theories and methods for the study of visual perception in psychophysics and neuroscience need to be integrated with information processing approaches in the attempt to model vision at a more abstract computational level. An interesting cross-fertilization is expected between neuroscience experimentation and computer vision theories. Particularly, for providing a detailed account of how the brain makes effective use of top-down resources (e.g., knowledge, memories etc.), creates predictions about forthcoming stimuli and constantly matches expectations against signals from the environment [6]. It is likely that as researchers in visual neuroscience and computer vision develop a better understanding of the role played by bottom-up and top-down information processing mechanisms and their interaction, better theories of visual perception can be developed.

Experimental Results

A considerable amount of experimental evidence exists for all the theories and approaches to the study of visual perception cited above. References to results on the physiology, psychology and ecology of vision are provided in [1,3]. Application of visual perception principles to the design of Information visualization and visual interfaces is discussed in [9].

Data Sets

An online laboratory providing a collection of demonstrations and experiments on visual perception can be found at: http://www2.psych.purdue.edu/~coglab/VisLab/welcome.html.

URL to Code

The portal (above) also intends to provide software that can be used in the study of visual science.

Cross-references

▶ Visual Interfaces

Recommended Reading

1. Bloomer, C.M. Principles of Visual Perception. Herbert, London, 1990.
2. Bruce V. and Green P.R. Visual Perception. Erlbaum, Hove, England, 1990.
3. Bruce V., Green P.R., and Georgeson M.A. Visual Perception: Physiology, Psychology, and Ecology. Psychology, New York, 2003.
4. Gibson J.J. The Ecological Approach to Visual Perception. Houghton Mifflin, Boston, 1979.
5. Marr D. and Vision W.H. Freeman, San Francisco, CA, USA, 1982.
6. Perception on Purpose. FP6-IST project N. 027268. Available at: http://perception.inrialpes.fr/POP/.
7. Stevens S.S. On the psychophysical law. Psychol. Rev., 64 (3):153–181, 1957.
8. von Helmholtz H. (Obituary). In Proc. Royal Soc. Lond. Royal Society, Great Britain. Taylor and Francis, London, 1854.
9. Ware C. Information Visualization: Perception for Design. Morgan Kaufmann, San Francisco, CA, USA, 2004.
10. Wertheimer M. Gestalt theory. In a Source Book of Gestalt Psychology, W.D. Ellis (ed. & trans.). Routledge and Kegan Paul, London (Original work published 1925), 1938, pp. 1–11.

Visual Query Language

Tiziana Catarci
University of Rome, Rome, Italy

Synonyms

Visual query system

Definition

Visual Query Languages (VQLs) are languages for querying databases that use a visual representation to depict the domain of interest and express related requests. VQLs provide a language to express the queries in a visual format, and they are oriented towards a wide spectrum of users, especially novices who have limited computer expertise and generally ignore the inner structure of the accessed database. Systems implementing VQLs are usually called Visual Query Systems (VQSs) [8].

Historical Background

The birth of VQLs was due to several needs, including: providing a friendly human-computer interaction, allowing database search by non-technical users, introducing a mechanism for comfortable navigation even in case of incomplete and ambiguous queries. It is worth noting that the real precursor of VQLs was QBE, already proposed by Moshe Zloof in 1977 [19].

QBE was really ahead of its time. Indeed, the Zloof's paper states: "the formulation of a transaction should capture the user's thought process. . . ." This is a quite common idea today, but at that time (1977) the research on user interfaces and human-computer interaction was still in its infancy. Approximately in the same period, Smith coined the term "icon" in his Ph.D. thesis [16] and Alan Kay introduced the idea of direct manipulation interfaces that are, in principle, usable by everyone [14]. It was necessary to wait until the beginning of 1980s for the first commercial systems making extensive use of direct manipulation, namely Xerox Star, Apple Lisa, and Macintosh.

It is worth noting that QBE is based not only on the usage of examples for expressing queries, but it also relies on the direct manipulation of relational tables inside a basic graphical user interface: an environment and an action modality which were quite unknown at that time, considering the almost simultaneous publication of Alan Kay's paper.

Another anticipatory idea is the incremental query formulation, i.e., ". . .the user can build up a query by augmenting it in a piecemeal fashion." Many papers still recommend to allow the user expressing the query in several steps, by composing and refining the initial formulations [8].

Moreover, QBE was the first proposal of query language in which the attention to the user interface is coupled with a rigorous definition of the language syntax and semantics and a careful study of the language expressive power. Unfortunately, many of the later proposals of visual query languages have emphasized the aspects related to user interaction and ease of learning and of use only, without also focusing on formal aspects, such as syntax, semantics and expressive power [8]. Such query languages are usually presented through examples of query formulation, making both the study of language expressiveness and the comparison with other languages difficult. The opposite is true for QBE, whose usability was also tested comparing it with SQL in several experiments, such as those reported in [15] and [17]. Finally, many of the QBE ideas are still up-to-date and it is amazing to note that QBE-like interfaces are nowadays adopted in commercial database systems, despite the current explosion of sophisticated visualizations and interaction mechanisms.

However, only later on, during the 1980s and early 1990s, VQLs received the greatest attention by the database community with the presentation of several diverse proposals for an effective visual and interactive query language.

Foundations

In [5], VQLs were reviewed and classified based on three criteria, namely:

- *What* can be done using the system, i.e., the *expressive power* of the environment.
- *How* the system may be used in order to build the query, i.e., the concept of *usability* as determined by the available strategies, the interaction and representation models.
- *Whom*, in terms of *classes of users*, the systems are addressed to.

The expressive power of a query system can be defined as the ability of the system to extract meaningful information from the database. More formally, according to [10], the expressive power can be based on the concept of *computable query*. Let U be a fixed countable set, called the universal domain, and let D(B), a subset of U, be a finite set which includes all the elements appearing in the database B. A *query* is a partial function giving an output (if any) which is a relation over D(B). A query Q is said to be *computable* if Q is partial recursive and satisfies a consistency criterion: if two databases are isomorphic, then the corresponding outputs of Q are also isomorphic (under the same isomorphism). In other words, the result of a query should be independent from the organization of the data in a database and should treat the elements of the database as uninterpreted objects. Excluding queries expressing undecidable problems, computable queries can be seen as the more general class of "reasonable" queries. Other meaningful classes of queries have been investigated in [11], giving rise to the so called Chandra's hierarchy. A significant class of queries inside the hierarchy is the one of first order queries, and the term *completeness* was used to indicate that a query language could express all the first order queries. However, it was apparent early that the amount of expressive power provided by such a language is not adequate for expressing useful queries, such as transitive closure and, more generally, fixpoint queries (i.e., queries obtained by augmenting the standard first order operators with the construct of least fixpoint). On the other hand, languages that friendly express queries simpler than first order queries are significant, since average database users make elementary requests.

As a consequence of the above remarks, the expressive power of the majority of visual query languages is lower or equal to the classes of first order or fixpoint queries. More precisely, most of the *iconic* languages fall in lowest level classes, since they are directed to a casual user, who is mainly interested in a friendly expression of simple queries (e.g., select-project queries). On the other hand, most of the *graphical* languages are equally or less expressive than relational algebra and very few are placed in the upper levels of the hierarchy (see [8]).

The notion of usability used in [5] is quite "database-oriented," and different from the ones used in the human-computer interaction (hci) community that is now commonly accepted. Indeed, in [5] usability was specified in terms of the models used in VQLs for denoting both data and queries, their corresponding visual representations and the strategies provided by the system in order to formulate the query. This definition does not reflect the hci view of usability as a software quality related to user perception and fruition of the software system. In particular, the data model has no significant impact on the user perception of the system. Whereas, visual representations and interaction strategies influence the user-system interaction, since they are important parts of the system interface (the only thing the user sees of a system).

As for the visual representation, the query representation is generally dependent on the database representation, since the way in which the query operands (i.e., data in the database) are presented constrains the query representation. For example, given a query on a relational database, the query may be formulated in terms of several representations, e.g., filling some fields in tables visualizing the relations, or following paths in a hypergraph that visualizes the relational schema. In this case the table and the hypergraph are two possible representations associated with the relational database. On the other hand, the visual representation used to display the query result can be different from the database representation. This is mainly due to the fact that what is visualized for the query purpose most often is the schema of the database, while the actual database instances constitute the query result to be displayed to the user.

Visual representations used in VQLs typically use forms, diagrams, icons or a combination of them. *Form*-based representations are the simplest way to provide the users with friendly interfaces for data manipulation. They are very common as application or system interfaces to relational databases, where the forms are actually a visualization of the tables. In query formulation prototypical forms are visualized for users to state the query by filling appropriate fields. In systems such as QBE [19], only the intensional part of relations is shown: the user fills the extensional part to provide an example of the requested result. The system retrieves whatever matches the example. In more recent form-based representations the user can manipulate both the intensional and the extensional part of the database.

Diagrammatic representations are also widely used in existing systems. Typically, diagrams represent data structures displayed using visual elements that correspond to the various types of concepts available in the underlying data model. Diagrammatic representations adopt as typical query operators the selection of elements, the traversal on adjacent elements and the creation of a bridge among disconnected elements.

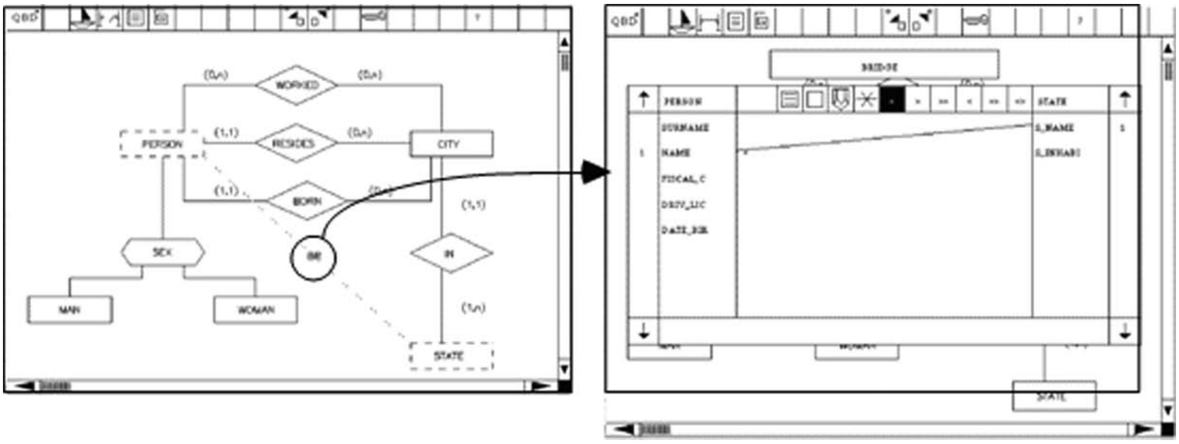
Iconic representations use icons to denote both the objects of the database and the operations to be performed on them. A query is expressed primarily by combining operand and operator icons. For example, icons may be vertically combined to denote conjunction (logical AND) and horizontally combined to denote disjunction (logical OR). In order to be effective, the proposed set of icons should be easily and intuitively understandable by most people. The need for users to memorize the semantics of the icons makes the approach manageable only for somehow limited sets of icons.

The *hybrid* representation is a combination of the above representations. Often, diagrams are used to describe the database schema, while icons are used either to represent specific prototypical objects or to indicate actions to be performed. Forms are mainly used for displaying the query result.

Similarly to most graphical user interfaces, the VQLs that have been developed so far have mainly stressed the user input aspects of the interaction and have given little thought to the visualization of output data. Conversely, an appropriate visualization of the query result allows the user to better capture the relationships amongst the output data, and some systems are progressing in this direction. For instance, AMAZE employs 3D graphs [7]. The data are shown as a 3D snapshot of the n-dimensional results. Different methods of result visualization are also planned to be available to the user. The Film Finder system [1] visualizes information about movies by means of *starfield displays*, which show database objects as small selectable spots (either points or 2D figures). The displayed data can be filtered by changing the range of values on both the Cartesian axes. The query result fits on a single screen and the system quickly, i.e., within a second, computes the new data display in response to the user's requests. This property, called near real-time interactivity, ensures high usability. A different approach is to use virtual reality techniques to present the query result with a simulation of a real environment (i.e., a virtual one) that depicts a situation familiar to the user. VQRH [12] is one of the systems that provide the user with several visual representations for both query formulation and result visualization. One possibility is to use 3D features to present the results in the simulated reality setting. For example, if the database refers to the books in a library, a virtual library can be represented in which the physical locations of the books are indicated by icons in a 3D presentation of the book stacks of the library.

Apart from the visual representation, any VQL is characterized by the way in which it allows the user to express his/her requests. Very often, the actual query specification is the second step of the user interaction, while there is a first step devoted to the understanding of the overall database content. This first phase is in general supported providing the user with browsing and/or filtering and zooming mechanisms.

Query formulation is the fundamental activity in the process of data retrieval. The query strategy *by schema navigation* has the characteristic of concentrating on a concept (or a group of concepts) and moving from it in order to reach other concepts of interest, on



Visual Query Language. Figure 1. Unconnected path in QBD* [19].

which further conditions may be specified. Such a strategy differs according to the type of path followed during the navigation (see Figure 1 for an example of unconnected path).

A second strategy for query formulation is *by subqueries*. In this case the query is formulated by composing partial results. The third strategy for query formulation is *by matching*. It is based on the idea of presenting the structure of a possible answer that is matched against the stored data.

The last strategy for query formulation is *by range selection*, allowing a search conditioned by a given range on multi-key data sets to be performed. The query is formulated through direct manipulation of graphical widgets, such as buttons, sliders, and scrollable lists, with one widget being used for every key. An interesting implementation of such a technique has been proposed in [1], and is called d*ynamic query*. The user can either indicate a range of numerical values (with a range slider), or a sequence of names alphabetically ordered (with an alpha slider). Given a query, a new query is easily formulated by moving the position of a slider with a mouse: this is supposed to give a sense of power but also of fun to the user, who is challenged to try other queries and see how the result is modified. Usually, input and output data are of the same type and may even coincide.

Key Applications

VQLs have been basically proposed to allow non-programmers to express database queries. In [5] it was also conjectured that, depending on the user class and the kind of task (i.e., query), certain VQLs are more appropriate than others. For instance, VQLs based on extensive use of icons and icon composition mechanisms [18], are typically more suited to express simple (select-project-join) queries and be used by naive users. Analysis underlying such statements were usually correct. Nevertheless, they needed to be substantiated by running user trials, and such experiments were (and still are) not so common in the VQL community. However, some of them have been carried on and basically support the hypothesis that different kinds of queries are better supported by different visual representations and interaction mechanisms. Whereas, results on the adequacy of the different visual representations with respect to the various classes of users are not so strong as it was conjectured.

Future Directions

VQLs mainly deal with traditional databases, i.e., databases containing alphanumeric data. However, in recent years the application realms of databases have raised a lot in terms of both number and variety of data types. As a consequence, specialized systems have been proposed for accessing such new kinds of databases, containing non-conventional data, such as images, videos, temporal series, maps, etc. Furthermore, the idea of information repository has been deeply influenced by the beginning of the Web age. Different visual systems have been proposed to cope with the need for extracting information residing on the Web.

Temporal Databases

There are a growing number of applications dealing with data characterized by the temporal dimension (e.g., medical records, biographical data, financial data, etc.). Still, visual interfaces for querying temporal databases have been less investigated than their counterpart in traditional databases. Typically, end-users of these data are competent in the field of the application but are not computer experts. They need easy-to-use systems able to support them in the task of accessing and manipulating the data contained in the databases. In this case, typical interactions with the data involve the visualization of some of their characteristics over some timeframe or the formulation of queries related with temporal events, such as the change of status of an employee or the inversion of the tendency of stock exchanges.

Geographical Databases

Geographical information is most naturally conveyed in visual format. Maps and diagrams (i.e., schematic maps such as a bus network map) are the core means in user interactions, both for querying the database and displaying the result of a query. A typical query would be "show me on a city map where is the post office that is closest to this location." The query itself would most likely be expressed using preformatted forms and menus to select the city and the reference location. The result would be a blinking or otherwise highlighted point in the displayed map. Once a map is displayed, as a result of a previous query or as an initial background screen in a query formulation interaction, the map can be used to specify a new query. This typically supports queries such as "give me more information on *this*," where the value of the *this* parameter is specified by pointing in some way to a location in the map (i.e., a point on the screen). Thus in some sense visual interaction is common practice in GIS systems, due to the intrinsically spatial reference that is associated to the data.

Web Visual Access

Nowdays, the Web is the widest information repository. However, to find the information of interest among the mass of uninteresting one is a very hard task. In order to help the user in retrieving information scattered everywhere in the Web, several proposals have been made by different research communities, such as those of database, artificial intelligence, and human-computer interaction (see, e.g., [13]), a limited amount of them relate to visual querying and information visualization.

Visually Querying Digital Libraries

The main purpose of a digital library (DL) is to facilitate the users in easily accessing the enormous amount of globally networked information, which includes preexisting public libraries, catalog data, digitized document collections, etc. Thus, it is fundamental to develop both the infrastructure and the user interface to effectively access the information via the Internet. The key technological issues are how to search and how to display desired selections from and across large collections. A DL interface must support a range of functions including query formulation, presentation of retrieved information, relevance feedback and browsing.

Experimental Results

As it was mentioned in the previous sections, user trials have been conducted to compare the usability of query languages. First a well-known definition of usability is recalled: "the extent to which a product can be used with efficiency, effectiveness and satisfaction by specific users to achieve specific goals in specific environments" [3]. More precisely, effectiveness refers to the extent to which the intended goals of the system can be achieved; efficiency is the time, the money, and the mental effort spent to achieve these goals; and satisfaction depends on how comfortable the users feel using the system.

In the case of VQLs the main goal is to extract information from the database by performing queries, and the accuracy in achieving such a goal is generally measured in terms of the accuracy of query completion (i.e., user's correctness rate when writing the queries). Measures of efficiency relate the level of effectiveness achieved at the expense of various resources, such as mental and physical effort, time, financial cost, etc. In principle, both the user's and the organization's point of view should be considered. However, the user's efficiency is most frequently measured in terms of the time spent to complete a query.

The above two measures (i.e., query accuracy and response time) can be evaluated quite precisely. Frequently this is done either by recording real users performing predefined tasks with the system and then analyzing the recorded data or by directly observing the user. The most common tasks are *query writing* and *query reading*, both of which are performed by investigating the relationships between database queries expressed in natural language and the same queries expressed in the system under study. In query writing, the question is: "Given a query in natural language how easily can a user express it through the query language statements?" The question for query reading is: "Given a query expressed through the query language statements, can the user express the query easily in natural language?" Moreover, other kinds of measures can be defined and evaluated, although with less precision.

Measures of satisfaction describe the comfort and acceptability of the overall system used. The learnability of a product may be measured by comparing the usability of a product handled by one user along a time scale. Measuring usability in different contexts can assess the flexibility of a product.

Usability of query languages has first been studied through the comparison between QBE and SQL [15,17]. The former study [15] showed better user performances when using QBE with respect to SQL, both in query reading and query writing tests. However, a later study [17] also comparing QBE and SQL took into account several factors, such as the use of the same database management system, a similar environment, etc. It is interesting to note that the query language type affected user performance only in "paper and pencil" tests, in which case QBE users had higher scores than SQL users. In on-line tests, the user's accuracy was not affected by the type of the language adopted, but the user's satisfaction was much greater with QBE, and his or her efficiency much better.

In [2], a language based on the previously-mentioned *dynamic queries* was tested against two other query languages, both providing form fill-in as the input method. One of these languages (called FG) has a graphical visualization output, and the other one (called FT) has a fully textual output. The alternative interfaces were chosen to find out which aspect of dynamic queries makes the major difference, either the input by sliders, or the output visualization. The tasks to be performed by the user concerned basically the selection of elements that satisfy certain conditions. However, the subjects were also asked to find a trend for a data property and to find an exception for a trend. The hypothesis that the dynamic query language would perform better than both the FG and the FT interface was confirmed. Similarly, the FG interface produced faster completion times than the FT interface. In particular, for the task of finding a trend, the possibility of getting an overview of the database (in the dynamic and FG interfaces) made the major difference. In searching for an exception, the dynamic interface performed significantly better than the FG and FT ones. This was due to the advantages offered by both the visualization and the sliders. The visualization allowed subjects to see exceptions easily when they showed up on the screen, and the sliders allowed them to quickly change the values to find the correct answer.

Other experiments have been conducted [17,18] to compare a diagrammatic query language, namely QBD*, against both SQL and QBI, an iconic query language. The overall objective of the studies was measuring and understanding the comparative effectiveness and efficiency with which subjects can construct queries in SQL or in the diagrammatic or iconic languages. The experiments were designed to determine if there is a significant correlation between 1) the query class and the query language type, and 2) the type of query language and the experience of the user. The subjects were undergraduate students, secretaries and professionals having different levels of expertise. The results of the comparison between QBD* and SQL confirmed the intuitive feeling that a visual language is easier to understand and use than a traditional textual language not only for novice users, but also for expert ones. The experts' errors when using SQL were mainly due to the need of remembering table names and using a precise syntax. Working with QBD*, users can gain from looking at the E-R diagrams. On the basis of the figures which have been obtained when comparing QBD* and QBI one can say that expert users perform better using the QBD* system, while a small difference exists concerning the performance of non-expert users (slightly better using QBI).

Cross-references

- ▶ Data Model
- ▶ Data Visualization
- ▶ Digital Libraries
- ▶ Expressive Power of Query Languages
- ▶ Geographic Information System
- ▶ Multimedia Databases
- ▶ Temporal Database
- ▶ Usability
- ▶ Query Language

Recommended Reading

1. Ahlberg C. and Shneiderman B. Visual information seeking: tight coupling of dynamic query filters with starfield displays. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1994, pp. 313–317.
2. Ahlberg C., Williamson C., and Shneidermann B. Dynamic queries for information exploration: an implementation and evaluation. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1992, pp. 619–626.
3. Angelaccio M., Catarci T., and Santucci G. QBD*: a graphical query language with recursion. IEEE Trans. Softw. Eng., 16:1150–1163, 1990.
4. Badre A.N., Catarci T., Massari A., and Santucci G. Comparative ease of use of a diagrammatic Vs. an iconic query language. In Interfaces to Databases J. Kennedy P.J. Barclay (eds.). Electronic Series Workshop in Computing, Springer, 1996.
5. Batini C., Catarci T., Costabile M.F., and Levialdi S. Visual Query Systems: A Taxonomy – nei. In Proc. 2nd IFIP W.G. 2.6 Working Conference on Visual Databases, 1991.
6. Bevan N. and Macleod M. Usability assessment and measurement. In The Management of Software Quality, M. Kelly (ed.). Ashgate Technical/Gower Press, Hampshire, UK, 1993.
7. Boyle J., Leishman S., and Gray P.M.D. From WIMP to 3D: the development of AMAZE. J. Vis. Lang. Comput. (Special issue on visual query systems), 7:291–319, 1996.
8. Catarci T., Costabile M.F., Levialdi S., and Batini C. Visual query systems for databases: a survey. J. Vis. Lang. Comput., 8(2):215–260, 1997.
9. Catarci T. and Santucci G. Diagrammatic vs Textual query languages: a comparative experiment. In Proc. IFIP W.G. 2.6 Working Conference on Visual Databases, 1995, pp. 57–85.
10. Chandra A.K. Programming primitives for database languages. In Proc. 8th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages, 1981, pp. 50–62.
11. Chandra A.K. Theory of database queries. In Proc. 7th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1988, pp. 1–9.
12. Chang S.K., Costabile M.F., and Levialdi S. Reality bites – progressive querying and result visualization in logical and VR spaces. In Proc. IEEE Symp. Visual Languages, 1994, pp. 100–109.
13. Geronimenko V. and Chen C. (eds.). Visualising the Semantic Web. Springer, Berlin, 2002.
14. Kay A. Personal dynamic media. IEEE Comput., 10(3): 31–42, 1977.
15. Reisner P. Query languages. In Handbook of Human-Computer Interaction, M. M. Helander (ed.). North-Holland, Amsterdam, 1988, pp. 257–280.
16. Smith D.C. Pygmalion: A Computer Program to Model and Stimulate Creative Thought. Birkhauser Verlag, Basel, 1977.
17. Yen M.Y. and Scamell R.W. A human factors experimental comparison of SQL and QBE. IEEE Trans. Softw. Eng., 19(4):390–402, 1993.
18. Tsuda K., Yoshitaka A., Hirakawa M., Tanaka M., and Ichikawa T. Iconic browser: an iconic retrieval system for object-oriented databases. J. Vis. Lang. Comput., 1(1):59–76, 1990.
19. Zloof M.M. Query-by-example: a database language. IBM Syst. J., 16(4):324–343, 1977.

Visual Query System

► Visual Query Language

Visual Representation

YANNIS IOANNIDIS
University of Athens, Athens, Greece

Synonyms

Graphical representation

Definition

The concept of "representation" captures the signs that stand in for and take the place of something else [5]. Visual representation, in particular, refers to the special case when these signs are visual (as opposed to textual, mathematical, etc.). On the other hand, there is no limit on what may be (visually) represented, which may range from abstract concepts to concrete objects in the real world or data items.

In addition to the above, however, the term "representation" is often overloaded and used to imply the actual process of connecting the two worlds of the original items and of their representatives. Typically, the context determines quite clearly which of the two meanings is intended in each case, hence, the term is used for both without further explanation.

Underneath any visual representation lies a mapping between the set of items that are being represented and the set of visual elements that are used to represent them, i.e., to display them in some medium. In order for a visual representation to be useful, the mapping must satisfy certain properties: it must be *expressive* as well as *effective* [1]. Expressiveness is related to the accuracy with which the visual representation captures the underlying items; an expressive mapping should neither lose any information nor lead to additional, irrelevant implications. Effectiveness is related to the quality of the interpretation of the visual representation by a human; an effective mapping should allow for a fast and unique interpretation of the underlying items, leaving no room for ambiguities or false impressions. Expressiveness and effectiveness are often enhanced significantly when the set of visual elements used to represent a set of (interrelated) items is familiar to the user (possibly from other domains), and the corresponding mapping is intuitive so that it invokes the correct interpretation of these items effortlessly.

The mapping that underlies a visual representation is often called a *Visual Metaphor* [4], as it realizes a "transfer" (metaphor is from the Greek word *μεταφορ ά* – metaphora, which means "transfer") of implicit and explicit characteristics of the visual elements to the items these represent.

The quality of visual representations depends on the visual elements used in it, the choice of which visual element is mapped to which item, and which visual element characteristic is mapped to which item characteristic, and any constraints that must be imposed on the mapping. A brief introduction to the above is the main topic of this entry.

Historical Background

Visual representation of the external world has been exercised by humans for thousands of years and, in recent history, this has extended to abstract worlds as well. Visual metaphors have been used so widely that human cognition is considered tightly interweaved, and sometimes even identified, with human vision. Work on visual representation of data and information dates as far back as the work of the Scottish political economist William Playfair, who was the first to use bar charts, pie charts, and other extremely intuitive and useful visual constructs that are used today for this purpose [6]. In recent times, many theories on the semantics of visual representations and how they may enhance human perception have been published, a great example being the work of Tufte [6,7].

Visual representation of information is extensively used in several types of computer systems, which are all using, either explicitly or implicitly, a set of visual elements and a mapping from them to data items being visually represented. Given that the variety of possibilities regarding both the elements and the mappings is endless, there have been a plethora of papers that have been written on visual representation, including some aggregate bodies of work that offer a comprehensive view of the entire field [2].

In the following section, a formal model for visual representation is presented. Although visual representations can be employed for any kind of abstract or concrete items, for ease of exposition and without loss of generality, the discussion focuses on representing data or information items.

Foundations

This section presents a formalism that captures the essence of visual representation and can be used as a means to classify and compare several approaches to the topic, and to explain some of the features of the relevant systems. For the sake of presentation clarity, it makes various simplifying assumptions and, therefore, its expressive power is not as high as some existing visual representations manifest. Nevertheless, it is expressive enough to capture a broad class of visual representations and characteristic enough to convey the main aspects of the concept overall.

Without loss of generality, any dataset is assumed to conform to the structure and constraints of some *data model* (in database terminology, that would be a *data schema*). Visualizations of such a dataset require the corresponding notion of a *visual model*, which is similar to a data model, except that its primitives are visual. In analogy to a dataset being an instance of a data model, a visual representation that is an instance of a visual model will be called a *visualset*.

In order for a visualset to obtain meaning and, therefore, become a visual representation of a dataset, it needs a mapping of its elements to the corresponding items of the dataset, so that users may see the former and understand the latter. Such a mapping is typically defined at the (data and visual) model level and propagated down to all instances of the models. Given this, such a mapping is called a (*visual*) *metaphor*.

Separation of the visual representation process into three distinct components (visual model, data model, and visual metaphor) has many benefits. It permits metaphors to be tested for correctness, evaluated, compared, and combined. Also, it permits systems, especially visual-representation and user-interface tools, to be evaluated with respect to (i) the quality of their visual models and metaphors, (ii) their flexibility with respect to defining or changing their models and metaphors, and (iii) the particular ways in which this is done.

Using the above three notions, the problem of visual representation of datasets may be stated more formally as follows. Given a data model $\mathcal{D}$, let $S(\mathcal{D})$ denote the set of valid datasets that can be constructed based on that model. Similarly, let $S(\mathcal{G})$ denote the set of visualsets that can be constructed based on a visual model $\mathcal{G}$. The goal is to establish a binary relation between $S(\mathcal{G})$ and $S(\mathcal{D})$, the metaphor, whose specific properties depend on the precise operations that users want to perform on visualsets. Specifically,

1. If users want to be able to use a visualset of $\mathcal{G}$ to view *any* dataset of $\mathcal{D}$ in its *entirety*, then an onto function must exist of the form $f \ : \ S(\mathcal{G}) \to \mathcal{S}(\mathcal{D})$, so that every dataset can be represented visually.
2. If, in addition, users want to be able to use a visualset of $\mathcal{G}$ to update a dataset of $\mathcal{D}$, then a total onto function must exist of the form $f \ : \ S(\mathcal{G}) \to \mathcal{S}(\mathcal{D})$, so that every visualset can be uniquely interpreted as a dataset.

Clearly, not all such functions *f* that satisfy the above properties are useful. Many are arbitrary mappings, with no obvious correspondence between the dataset and visualset. The goal is to establish a relationship between the members of $S(\mathcal{D})$ and $S(\mathcal{G})$ so that when users view a visualset, they can infer the dataset to which it maps. This implies that *f* should be derived from a correspondence between the individual components and features of the data and visual models, which would result in a structural similarity between datasets and visualsets, making it easier for users to understand the former through the latter.

Data and Visual Models

The formalism below (essentially a meta-model) identifies the key components and features of a large and interesting class of data and visual models, which serves as an important example of the concepts surrounding visual representation. In that formalism, every data or visual model $\mathcal{M}$ can be seen as a quadruple $\mathcal{M} \ = < \mathcal{P}, \mathcal{A}, \mathcal{V}, \mathcal{C} >$ defined as follows:

$\mathcal{P}$ is a finite set of classes (containers) of data items (resp., visual elements). Each such class *P* is associated with a (possibly infinite) set of unique ids $\mathcal{I}(P)$ that can be used to identify the members of class *P*.

$\mathcal{A}$ is a finite set of identifiers of attributes that data items (resp., visual elements) are expected to have, depending on the class they belong in. If $\mathcal{A}(P)$ represents the attributes of the members of class *P*, then $\mathcal{A} = \bigcup_{P \in \mathcal{P}} \mathcal{A}(P)$. For all $P \in \mathcal{P}$, each member of $\mathcal{A}(P)$ is of the form *P.A*, where *A* is the name of the corresponding attribute of the members of class *P*.

$\mathcal{V}$ is a (possibly infinite) set of identifiers of the possible values that the above attributes may have. If $\mathcal{V}(P.A)$ represents the possible values of attribute *P.A*, then $\mathcal{V} = \bigcup_{P \in \mathcal{P}} \bigcup_{P.A \in \mathcal{A}(P)} \mathcal{V}(P.A)$. For all $P.A \in \mathcal{A}(P)$, each element of $\mathcal{V}(P.A)$ is of the form $P.A == v$, where *v* is the corresponding potential value of attribute *P.A*. For an attribute *P.A*, $\mathcal{V}(P.A)$ may be equal to a set $\mathcal{I}()$ of ids of the members of some class.

$\mathcal{C}$ is a finite set of *constraints*, i.e., rules that must be satisfied by any dataset (resp., visualset) that conforms to $\mathcal{M}$. These constraints are formulas in some prespecified language $\mathcal{L}$ and refer to members of $\mathcal{P}$, $\mathcal{A}$, $\mathcal{V}$.

To draw an analogy, in relational database terms, the above describes relational tables ($\mathcal{P}$) with attributes ($\mathcal{A}$), the attributes' domains ($\mathcal{V}$), and integrity constraints ($\mathcal{C}$). It is conceivable that some data or visual model may not be representable in (any enhancement of) the above formalism. Most of the common models fall naturally in this formalism, however, so it is the one adopted in this entry.

Data models capture abstract organization of information and their creation is a classical database problem that is well understood. As an example, consider a very simple data model on *employees*. Each employee has a *name*, a *salary*, an *age*, and a *department*. The three possible departments are "toy", "candy", and "shoe". This data model is the quadruple $\mathcal{D} \ = < \mathcal{P}_\mathcal{D}, \mathcal{A}_\mathcal{D}, \mathcal{V}_\mathcal{D}, \mathcal{C}_\mathcal{D} >$, where $\mathcal{C}_\mathcal{D} = \emptyset$, and the data-item classes in $\mathcal{P}_\mathcal{D}$, their attributes in $\mathcal{A}_\mathcal{D}$, and their corresponding value sets in $\mathcal{V}_\mathcal{D}$ are given in the following table.

Data item class (*P*)	Attribute (*P.A*)	Attribute values ($\mathcal{V}(P.A)$)
Employee	Name	text
	Salary	[0, 10000]
	Age	[0, 100]
	Dept	{toy, candy, shoe}

Contrary to the situation with data models, creating suitable visual models is less straightforward, since they must reflect not only the (visual) information to be organized, but also the medium in which the models are expressed. To create visual item classes for a visual model, visual building blocks are needed, which may actually be combinations of standard visual elements, such as points, lines, regions, text, etc. (A more formal discussion of visual constructs may be found in [3].) Constraints in the visual model are used to hold compositions together. For example, if a visual element is a box with a piece of text in the center, then it is a composition of a region and a text item satistfying the constraint that the location of the text time (its center) is the same as the location of the region (its center).

As an example, consider a very simple visual model that supports the placement of objects in a two-dimensional data field. The only class of visual items is *point* (not in the mathematical sense), which is simply a region. This visual model is the quadruple $\mathcal{G} = < \mathcal{P}_\mathcal{G}, \mathcal{A}_\mathcal{G}, \mathcal{V}_\mathcal{G}, \mathcal{C}_\mathcal{G} >$, where again $\mathcal{C}_\mathcal{G} = \emptyset$, and the visual-item classes in $\mathcal{P}_\mathcal{G}$, their attributes in $\mathcal{A}_\mathcal{G}$, and their corresponding value sets in $\mathcal{V}_\mathcal{G}$ are given in the following table. For simplicity, only a subset of the attributes is shown.

Visual item class (*P*)	Attribute (*P.A*)	Attribute values ($\mathcal{V}(P.A)$)
Point	LocationX	plane-pixel-x
	LocationY	plane-pixel-y
	Size	{100 pixels}
	Shape	{square, oval, triangle}
	Color	{blue, red, green}

Visual Metaphors

A visual metaphor is defined as a correspondence between features of a visual and a data model. This correspondence induces a mapping between visualsets (instances of the visual model) and datasets (instances of the data model). Having the mapping be decomposed into mappings of the model features helps produce visualizations that, when viewed, allow the user to deduce the underlying dataset. Consider a data model $\mathcal{D} = < \mathcal{P}_\mathcal{D}, \mathcal{A}_\mathcal{D}, \mathcal{V}_\mathcal{D}, \mathcal{C}_\mathcal{D} >$ and a visual model $\mathcal{G} = < \mathcal{P}_\mathcal{G}, \mathcal{A}_\mathcal{G}, \mathcal{V}_\mathcal{G}, \mathcal{C}_\mathcal{G} >$. A metaphor includes correspondences between item classes ($\mathcal{P}_\mathcal{D}$ and $\mathcal{P}_\mathcal{G}$), their attributes ($\mathcal{A}_\mathcal{D}$ and $\mathcal{A}_\mathcal{G}$), and their attribute values ($\mathcal{V}_\mathcal{D}$ and $\mathcal{V}_\mathcal{G}$). These correspondences describe the meaning of visual model features relative to the underlying data model. To allow presentation flexibility, correspondences are permited between multiple features in the visual model and a single feature in the data model.

More formally, a *metaphor* T is an onto function from $\mathcal{G}$ to $\mathcal{D}$ (denoted by $T : \mathcal{G} \rightarrow \mathcal{D}$), which is the union of the following three onto functions:

1. Function $T_p : \mathcal{P}_\mathcal{G} \rightarrow \mathcal{P}_\mathcal{D}$
2. Function $T_a : \mathcal{A}_\mathcal{G} \rightarrow \mathcal{A}_\mathcal{D}$, where $T_a(P_G.A_G) = P_D.A_D$ only if $T_p(P_G) = P_D$
3. Function $T_v : \mathcal{V}_\mathcal{G} \rightarrow \mathcal{V}_\mathcal{D}$, where $T_v(P_G.A_G == v_G) = (P_D.A_D == v_D)$ only if $T_a(P_G.A_G) = P_D.A_D$

As an illustration of the above definition, the following metaphor *T* maps from the visual model to the data model discussed above.

A metaphor provides meaning to the features of a visual model by establishing a correspondence between them and features of a data model. The precise meaning is captured by *T*. For example, displaying a red square *point* implies the existence of an *employee* in the toy department. The metaphor must be such that users can correctly and unambiguously interpret a

x	$T_p(x)$	*x*	$T_a(x)$	*x*	$T_v(x)$
Point	Employee	point.locationX	employee.salary	point.locationX==x	employee.salary==(x*10)
		point.locationY	employee.age	point.locationY==x	employee.age==(x/5)
		point.color	employee.dept	point.color==“red”	employee.dept==“toy”
				point.color==“green”	employee.dept==“candy”
				point.color==“blue”	employee.dept==“shoe”
		point.shape	employee.dept	point.shape==“square”	employee.dept==“toy”
				point.shape==“oval”	employee.dept==“candy”
				point.shape==“triangle”	employee.dept==“shoe”

displayed visualset. This can be made more precise based on various mandatory and optional properties of T, i.e., being a function, onto, total, and 1-1.

As a minimum requirement, a metaphor T has been defined as an onto function: if it weren't onto, then some characteristics of a data model would not be captured visually; if it weren't a function, then a single visual construct would have multiple meanings and, therefore, would not be interpreted appropriately.

Beyond the above, a visual model may have features that do not carry any meaning and/or features that carry redundant meaning. Based on knowledge of T, users should be able to ignore the former and not be confused by the latter. In particular, if a metaphor is not total then some visual elements do not mean anything in the data model. This creates no problem if the metaphor will be used simply for visual display of datasets: unmapped features of the visual model will be used just for presentation. On the other hand, if the metaphor will be used for updating datasets displayed, then if a visual attribute $P_G.A_G$ is mapped by T_a, then all the values in $\mathcal{V}_\mathcal{G}(P_G.A_G)$ should be mapped as well, otherwise, one would be able to draw visualsets that are not translatable to datasets.

Also, for both retrieval and update, if a metaphor is not 1-1, then multiple visual elements have the same meaning in the data model. If T_p or T_v are not 1-1, then there is a choice of visual constructs that can be used, which should be left to the user or resolved via some default mechanism. If T_a is not 1-1, then there is redundancy: multiple visual attributes capture the same data attribute.

Functions that are not 1:1 establish equivalence classes among the features of the visual model, i.e., several features may have the same meaning. For example,

$$T_a(point.color) = T_a(point.shape) = employee.dept$$

specifies redundancy: visual attributes *point.color* and *point.shape* redundantly capture data attribute *employee.dept*. Value mappings are similar:

$$\begin{aligned} &T_v(\text{point.color} ==''\text{green}'') \\ &= T_v(\text{point.shape} ==''\text{oval}'') \\ &= (\text{employee.dept} ==''\text{candy}'') \end{aligned}$$

indicates that specific values of redundant visual attributes (e.g., *point.color* and *point.shape*) capture the same value of a data attribute.

Datasets, Visualsets, and Visual Representation

As defined above, a dataset or visualset may be considered as an instantiation of a data or visual model, respectively. In particular, a dataset S of a model $\mathcal{M}$ is defined as follows (similarly for a visualset):

1. For every $P \in \mathcal{P}$, there is a finite set $[P] \subseteq \mathcal{I}(\mathcal{P})$ of data items in class P that appear in dataset S.
2. For every $P.A \in \mathcal{A}$, there is a total function $[P.A] : [P] \rightarrow \mathcal{V}(P.A)$, which determines the value of the $P.A$ attribute for every data item in $[P]$.
3. For every $c \in \mathcal{C}$, there is a constraint $[c]$, constructed from c by replacing every $P \in \mathcal{P}$ by $[P]$ and every $P.A \in \mathcal{A}$ by $[P.A]$. All these constraints are satisfied by the above.

Given a metaphor $T : \mathcal{G} \rightarrow \mathcal{D}$, any dataset that conforms to data model $\mathcal{D}$ can be represented by a visualset that conforms to visual model $\mathcal{G}$ via a mapping of their contents that remains faithful to the metaphor. In particular, visual items in the classes of such a visualset represent data items in the corresponding classes (based on T) of the dataset. Similarly, the visual attributes and the values of these visual elements represent the data item attributes and their values as specified by T. Essentially, there is a function t induced by T that determines a mapping from any visualset that conforms to $\mathcal{G}$ to some dataset that conforms to $\mathcal{D}$. As with metaphor T, the induced function t can be extended so that its domain includes instantiations of constraints as well, i.e., $t([c])$ is valid for $c \in \mathcal{C}_\mathcal{G}$.

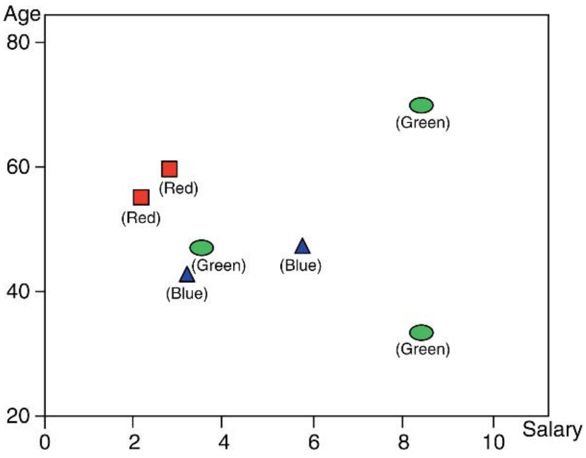
As a simple example of all the above, consider the following dataset that conforms to the data model described above:

employee(Jason, 2.1, 55, "toy")
employee(Iris, 2.6, 60, "toy")
employee(Nick, 3.2, 43, "shoe"
employee(Magda, 5.8, 47, "shoe")
employee(Chloe, 3.4, 46, "candy")
employee(Phoebe, 8.3, 31, "candy")
employee(Mike, 8.3, 70, "candy")

Figure below gives an example visualset that would be produced by applying the example metaphor above to this dataset. (The color of a point is indicated textually.) Clearly, real systems often deal with more complicated data and visual models and visual metaphors, but the underlying principles remain those highlighted in this exposition.

Key Applications

Visual representation is manifested in all applications with a (visual) user interface. Furthermore, having explicit declaration, storage, and manipulation of data and visual models as well as visual metaphors is essential to systems that offer to users the ability to modify the visual representations used, for reasons of personalization, quality control, or even plain variety.



Cross-references

- ▶ Data Visualization
- ▶ Visual Formalisms
- ▶ Visual Metaphor
- ▶ Visual Perception

Recommended Reading

1. Card S.K., Mackinlay J.D., and Shneiderman B. Information visualization. In Readings in Information Visualization: Using Vision to Think, 1999, pp. 1–34.
2. Card S.K., Mackinlay J.D., and Shneiderman B. Readings in Information Visualization: Using Vision to Think. Morgan Kaufman, Los Altos, CA, 1999.
3. Foley J.D., van Dam A., Feiner S.K., and Hughes J.F. Computer Graphics: Principles and Practice. Addison-Wesley, Reading, MA, 1990.
4. Haber E.M., Ioannidis Y., and Livny M. Foundations of visual metaphors for schema display. J. Intell. Inf. Syst., 3(3/4):263–298, 1994.
5. Mitchell W. Representation. In Critical Terms for Literary Study, Lentricchia F and McLaughlin T. (eds.), 2nd edn., Chicago, IL. University of Chicago Press, 1995.
6. Tufte E.R. The Visual Display of Quantitative Information. Graphics Press, Cheshire, CO, 1983.
7. Tufte E.R. Envisioning Information. Graphics Press, Cheshire, CO, 1990.

Visual Similarity

▶ Image Retrieval

Visual Web Data Extraction

▶ GUIs for Web Data Extraction

Visual Web Information Extraction

▶ GUIs for Web Data Extraction

Visualization for Information Retrieval

Jin Zhang
University of Wisconsin Milwaukee, Milwaukee,
WI, USA

Definition

Visualization for information retrieval refers to a process that transforms the invisible abstract data and their semantic relationships in a data collection into a visible display and visualizes the internal retrieval processes for users [14]. Basically, visualization for information retrieval consists of two components: visually presenting objects in a meaningful way in a defined visual environment or space, and visualizing information seeking process within the environment or space. Transformation of the invisible and abstract information and sophisticated semantic connections into a visual environment requires clearly defining a visual environment or space, pinpointing the objects in a data collection which are displayed in the visual environment, identifying significant connections among the objects, and projecting both the objects and the relationships onto the environment. The form and way of visualizing the internal retrieval process vary in different visualization models. Visualization of the internal retrieval process depends not only on a defined visual environment, but also on the information seeking paradigms such as query searching and browsing.

Many traditional information retrieval evaluation models such as the distance evaluation model, cosine evaluation model, ellipse evaluation model, etc. can be visualized in a visual space. Furthermore, new information retrieval evaluation models can be developed in the visual space.

The primary advantage of visualization for information retrieval is to make objects, object relationships, and information seeking transparent to end users. As a result, information retrieval becomes more intuitive and effective.

Visualization methods can be applied to various information retrieval models such as the Boolean based information retrieval model, the vector-based information model, etc.

Historical Background

A traditional information retrieval system like an *OPAC* (Online Public Access Catalog) system and search engine usually matches a user's query with document surrogates in a database, and returns the user with a linear retrieval results list. The results list includes relevant items retrieved from the database. The results list may be ranked against titles, authors, publishing time, or similarities. In nature an information retrieval process is an iterative process. Users of the information retrieval system are supposed to use the results list to make the relevance judgment decisions, subject relevance analysis among the retrieved items, and readjust the search strategies. Towards this aim, an information retrieval system should not only provide users with relevance information between a query and the retrieved items, but also relevance information among the retrieved items, in the retrieval results list. The former information is apparently important for information retrieval. The latter information, which is as valuable as the former information, would facilitate the users to further expand, revise, and modify their search strategies. Unfortunately, if an information retrieval system is equipped with a similarity ranking mechanism for its linear results list structure, it only provides users with the relevance information between a query and the retrieved items. The inherent weakness of the linear result list structure cannot offer relevance information among the retrieved items, let alone the degree to which they are relevant.

In a traditional information retrieval system, the user's information seeking process is discontinuous. After transferring a user's information need into a query and submitting it to the system, the user loses control over the internal information processing. The user regains control after the retrieval results are returned to the user. The user has no clues about how the query is matched with document surrogates, and how the final retrieval decision is made. In other words, the internal retrieval processing is a "black box" for the end user. The users cannot observe the internal processing and engage in it. However, if the user could participate in the internal retrieval processing, it would make information seeking more user-friendly, relevance judgment decision-making more accurate, and the retrieval process more natural.

Information retrieval basically consists of two important fronts: browsing and querying. They are equally important for information seeking. Each has its own advantages and disadvantages. Neither can replace the other. In a traditional information retrieval context where querying dominates an information retrieval process, browsing cannot be fully utilized to explore information due to the lack of a meaningful information browsing environment. Visualization for information retrieval opens a new avenue for users to seek for information in the two fronts.

Korfhage [6], as a pioneer and leading researcher in the field of visualization for information retrieval, made a significant contribution to the early research in the field. He introduced the important concept of the reference point, which can be used as a projection reference point when objects in a high dimensional space are projected onto a low dimensional visual space. He visually interpreted an information evaluation retrieval model in a visual space, and came up with new visualization models for information retrieval.

Foundations

Models for Multiple Reference Points

A reference point (or point of interest) is introduced to represent users' information needs. In a broad sense, a query is a kind of a reference point. A reference point can reflect a particular perspective of a user's information need. It can be a subject topic, user's search preference, a previous query, or any information related to the user's information need. Since a document usually involves multiple facets, multiple reference points can reveal more information about the document from multiple perspectives.

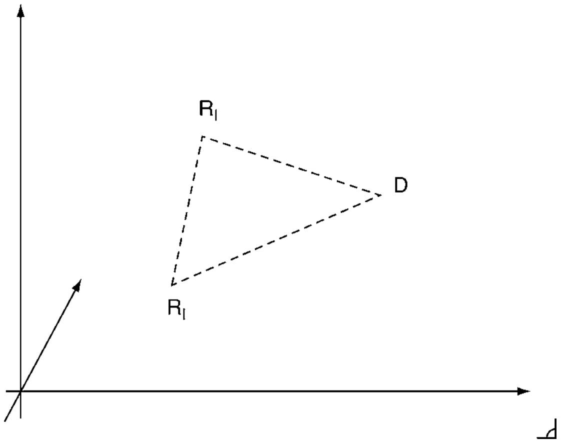
The visualization models for multiple reference points can be classified into two categories: models for fixed multiple reference points and models for movable reference points. In a model for fixed reference points [11] which is developed to visualize a sophisticated Boolean query, the vertices of a polygon represent the fixed reference points and the edges of the polygon have the same length. Within the polygon there are several concentric layers. The number of the concentric layers is equal to the number of the reference points or the number of vertices in the polygon. Each layer represents a different logic combination zone of the involved reference points. For instance, the first outer layer represents single reference point zone, the second outer layer represents the logic AND combination zone for two reference points, . . ., and the last central layer represents the logic AND combination zone for all reference points (or the final results of the query). Various icons which symbolize search results for the logic combination zones are located within the corresponding layers. The shape and the number of the icons vary in different layers so that they can be easily distinguished in the visual space. This multiple layer structure can display not only the final results of a query but also the itemized results at different combination levels. In addition, each vertex of the polygon can be defined as a sub-query. If that is the case, the vertex can extend to a new sub-polygon to visualize the sub-query results by connecting the vertex to the new sub-polygon. Thanks to this feature, the model can visualize a sophisticated Boolean query.

A model for movable reference points [9] works differently from the model for fixed reference points. It is apparent that in this model one of the most distinctive characteristics is that the involved reference points can be manipulated and placed in any place in the visual space by users. In other words, the reference points can be moved to a meaningful position. Because the model is developed based on the vector model instead of the Boolean model, the similarity between a document and any of the involved reference points can be accurately calculated by using a selected similarity measure. Basically, the location of a document in the visual space is affected directly by the strengths (similarities) between the document and all involved reference points. Consequently, after all reference points and documents are projected onto the visual space, the documents which are closer to a reference point are more relevant to the reference point. The uniqueness of the model relies on the fact that the strength between a document and a reference point is relative because the strength is measured by the ratio of the similarity between the document and the reference point to the similarities between the document and all reference points in the visual space. It is this relativity in conjunction with the fact that the similarities are not assigned to any coordinates of the visual space directly that makes the movability of the reference points possible in the visual space. That is, the positions of projected documents are determined by the locations of the reference points. The implication of the movability of the reference points on information retrieval is that users can select a reference point of interest, move it, and observe the object configuration change caused by the reference point movement in the visual space. If documents in the visual space are relevant to the moving reference point, they also move accordingly. Otherwise they stay still. Using this characteristic, users can effectively identify a group of related terms to a reference point in the visual space by selecting and moving it. This model can be applied to visualizing a returned results list from an information retrieval system, a full-text, and hyperlinks structures.

Euclidean Space Characteristics Based Models

Two of the most prominent characteristics of the Euclidean space are distance and direction. Both distance and direction are used to define and describe object locations in a space, and to demonstrate relationships among the objects in the space. When documents are organized in a vector space, they can be virtually described in a hyperspace where both the distance and direction characteristics are preserved. Furthermore, both the distance and direction have significance for information retrieval. It is natural and intuitive to utilize both distance and direction to develop information retrieval visualization models. By reducing the dimensionality of the high dimensional document vector space, spatial relationships among the documents in the vector space can be projected onto a low dimensional space to observe and manipulate the documents.

Euclidean space characteristics based models require two reference points to construct their low dimensional visual spaces. In the vector space if two reference points (R_1 and R_2) are clearly defined, a group of important parameters for a document (D) in the vector space can be calculated. The two visual distances (R_1D and R_2D) between the document and



Visualization for Information Retrieval. Figure 1. Display of the visual distances and visual sangles of a document.

the two reference points respectively, and the two visual angles ($\angle R_2R_1D$ and $\angle R_1R_2D$) formed by the three points (D, R_1 and R_2) can be calculated. For simplicity, D, R_1 and R_2 are displayed in a three dimensional space (see Fig. 1). If one of the two visual distances and one of the two visual angles are assigned to the Y-axis and X-axis, respectively, it constructs the visual space of the distance-angle based visualization model [15]. If the two angles are assigned to the Y-axis and X-axis, respectively, it constructs the visual space of the angle-angle based visualization model [13]. If the two distances are assigned to the Y-axis and X-axis, respectively, it constructs the visual space of the distance-distance based visualization model [8]. Observe that as long as these visual spaces are defined, any documents in the vector space can be effectively projected onto the visual spaces. The valid display areas vary in the models. They range from a close area to a semi-open area.

The uniqueness of these visualization models is reflected in the visualization of information retrieval evaluation models such as the distance evaluation model, cosine evaluation model, conjunction evaluation model, disjunction evaluation model, ellipse evaluation model, and oval evaluation model. In other words, they visualize the internal retrieval process in the visual spaces. For instance, in the vector space, the distance evaluation model corresponds to a hypersphere whose center is a query and radius is a retrieval threshold. Documents within the invisible sphere are regarded as the retrieved documents. This hyper sphere can be converted to a horizontal line in the distance-angle based low visual space if the query and the origin of the vector space are defined as the two reference points. That is because the distance from any point on the sphere to the center, which is one of the projection parameters, is a constant regardless of another projection parameter angle. Users can interact with the horizontal line in the visual space to control the size of the sphere in the vector space.

Visualization of Hierarchy Structures

As one important browsing mechanism, a hierarchy structure is widely used to organize and present a variety of information. A hierarchy structure can provide users with a structural categorical framework directing users to relevant information. Visualization techniques can enable users to navigate smoothly in a hierarchy structure by visualizing both global and local overviews of a hierarchy structure. Visualization technique enhances controllability and flexibility of information exploration and therefore alleviates the "disorientation" during navigation. ConeTree [10] is a visualization system that shows root, branch, and leaf nodes of a tree structure in a 3-D environment. Starting from the apex of a cone (the root of the tree), it fans out from left to right in a form of the cone. The child nodes are displayed along the cone base circumference which is a circle. The circle's diameter depends on the number of the child nodes on it. Any of these child nodes can extend to a new cone structure similar to its parent if the node has child nodes. In this way, an entire hierarchy structure can be presented visually. TreeMap [1] displays a hierarchy structure in a different way. Its visual space is a grid. A cell of the grid can include several structurally similar sub-grids. The nested sub-grids show the categories at different levels, each sub-grid corresponds to a new sub-category.

Visualization of Internet Information

The Web provides both challenge and opportunity for visualization for information retrieval. Information on the Web is huge, dynamic, diverse, and hyperlinked. It has become an indispensable source for people to search for information. One of the problems for information seeking on the Web is the notorious "lost in cyberspace" state. Visualization for information retrieval can alleviate, if not eliminate, the problem. Using information visualization techniques such as the hyperbolic method, people can effectively visualize the hidden hyperlink relationships among the connected web pages. In such a visualization environment, web pages (nodes) are connected by edges (hyperlinks). Users can easily recognize the visited paths, revisit the browsed web pages, and explore new paths in the visual space. A hyperbolic method is employed to visualize a subject directory where nodes are linked by hyperlinks [4].

A search engine can respond to a user query with an overwhelming number of related web pages. It is difficult for users to browse and identify the relevant web pages from the returned list. Information visualization techniques provide a unique way to solve the problem. In a cartographic visual space the returned web pages from multiple search engines are metaphorically presented as cities and the semantic link between two cities as a road [5]. The interactive map facilitates the decision-making on the relevance judgment. Grokker [3] categorizes all retrieved web pages and presents them in a group of circles in its visual space. A circle which represents a category includes multiple smaller circles which are its sub-categories. Users can drill down from the top of a category to the bottom (the web pages) by selecting corresponding circles in the visual space.

It is worth pointing out that there are many other information visualization approaches such as multidimensional scaling analysis [12], pathfinder associative networks [2], self-organizing maps [7], etc. which can be applied to information retrieval. A detailed discussion of visualization for information retrieval, comparisons among various approaches and their implications for information retrieval are discussed in Zhang's monograph on the topic [14].

Cross-references

- ▶ Information Retrieval
- ▶ Information Visualization

Recommended Reading

1. Asahi T., Turo D., and Shneiderman B. Using treemaps to visualize the analytic hierarchy process. Inf. Syst. Res., 6(4):357–375, 1995.
2. Dearholt D.W. and Schvaneveldt R.W. In Pathfinder Associative Networks: Studies in Knowledge Organization, R.W. Schvaneveldt (ed.). Ablex Publishing, Norwood, NJ, 1990, pp. 1–30.
3. Grokker. Available online at: http://www.grokker.com/ (retrieved October 29, 2007).
4. Inxight. Available online at: http://www.inxight.com/ (retrieved on October 29, 2007).
5. Kartoo. Available online at: http://www.kartoo.com/ (retrieved on October 29, 2007).
6. Korfhage R.R. To see or not to see – is that the query? In Proc. 14th Annu. Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1991, pp. 134–141.
7. Lin X. Map displays for information retrieval. J. Am. Soc. Inf. Sci., 48(1):40–54, 1997.
8. Nuchprayoon A. and Korfhage R.R. GUIDO: a visual tool for retrieving documents. In Proc. 1994 IEEE Comp. Soc. Workshop on Visual Languages, 1994, pp. 64–71.
9. Olsen K.A., Korfhage R.R., Sochats K.M., Spring M.B., and Williams J.G. Visualization of a document collection: the VIBE system. Inf. Process. Manag., 29(1):69–81, 1993.
10. Robertson G.G., Card S.K., and Mackinlay J.D. Information visualization using 3D interactive animation. Commun. ACM, 36(4):57–71, 1993.
11. Spoerri A. InfoCrystal: a visual tool for information retrieval and management. In Proc. Second Int. Conf. on Information and Knowledge Management, 1993, pp. 11–20.
12. White H.D. Visualizing a discipline: an author co-citation analysis of information science, 1972–1995. J. Am. Soc. Inf. Sci. Technol., 49(4):327–355, 1998.
13. Zhang J. TOFIR: a tool of facilitating information retrieval – introduce a visual retrieval model. Inf. Process. Manag., 37(4):639–657, 2001.
14. Zhang J. Visualization for Information Retrieval. Springer, Berlin, 2008.
15. Zhang J. and Korfhage R.R. DARE: distance and angle retrieval environment: a tale of the two measures. J. Am. Soc. Inf. Sci., 50(9):779–787, 1999.

Visualization Pipeline

Helwig Hauser[1], Heidrun Schumann[2]
[1]University of Bergen, Bergen, Norway
[2]University of Rostock, Rostock, Germany

Synonyms

Visualization reference model

Definition

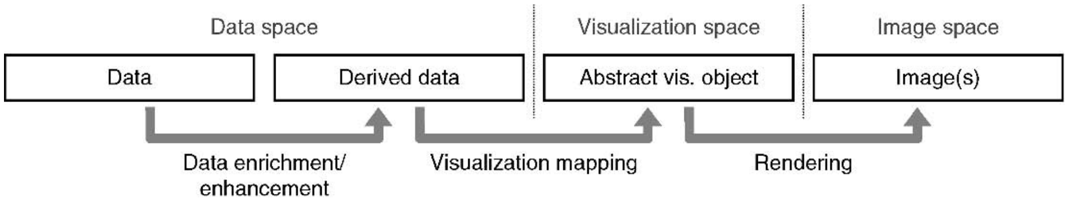
The *visualization pipeline* is a general model for a typical structure of a visualization process, which has been abstracted from earlier works in visualization in the late 1980s [3], and then adapted and extended several times [1, 2,...]. Starting out from *data* to be visualized and a particular visualization *task* at hand, a number of steps are processed along the visualization pipeline, including *data enhancement*, *visualization mapping*, and *rendering*, to eventually achieve a visualization of the data with the purpose of serving the given visualization task through effectiveness, expressiveness, and appropriateness [4].

Key Points

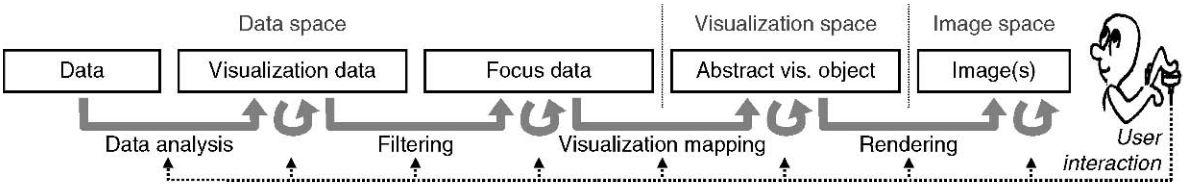
Data visualization is a part of computer science which provides expressive visual representations of data – through the appropriate application of computer graphics, often also in an interactive form – with the intention to effectively aid specific user tasks, including the exploration, analysis, and presentation of data. At this point only graphical data visualization is considered and other forms of data visualization such as sonification or the visualization through haptics are disregarded. Visualization establishes an efficient link between the user and her or his data, utilizing the enormous strengths of the human visual system with perception and cognition, and leading to a better understanding of the data through information extraction and knowledge crystallization.

To do so, the data is transformed from its original space, i.e., the *data space*, to an abstract *visualization space*, usually in two or three dimensions, where the visualization representation of the data is constructed. To eventually link up to the user, the visualization is rendered to an image or animation which is then displayed to the user. At least two mappings, i.e., the *visualization mapping* (from data space to visualization space) and *rendering* (from visualization space to image space) are required to fully accomplish this process. An example is the geometric representation of a 3D scalar data distribution by a discretized iso-surface, i.e., a mesh of triangles, which is then rendered to the screen by the use of computer graphics algorithms for visible surface detection and shading. In almost all cases, however, it is also necessary to first prepare and transform the data according to the needs of the visualization task at hand, such that visualization mapping and rendering can be applied afterwards. Haber and McNabb have already described this principal visualization model (as illustrated in Fig. 1) in 1990 [3].

About ten years after Haber and McNabb, Chi formulated the data state reference model for visualization [1], which formulates the visualization pipeline on the basis of transformation and stage operators, which – depending on their combination – result in various different possible instances of the visualization pipeline model. In 2004, dos Santos and Brodlie concluded that it is more meaningful to split the first step of the visualization pipeline into two [2]: *data analysis* and *filtering*. While the original data is initially prepared (once) for visualization, e.g., through interpolation, i.e., in a data-centric fashion, thereafter the user selects which data to actually visualize in the highly interactive and thus user-centric filtering step. Especially in the case of explorative visualization, where users search for priori unknown features in the data, the flexible interaction with the filtering step is crucial.



Visualization Pipeline. Figure 1. The visualization pipeline according to Haber and McNabb [3].



Visualization Pipeline. Figure 2. The visualization pipeline based on operators [1], with two steps in data space [2], and with user interaction.

The consideration of user interaction in general, altering operator parameters at all stages of the visualization pipeline, is an important aspect of the modern visualization pipeline (as also shown in Fig. 2).

Cross-references

▶ Data Visualization
▶ Visual Data Mining

Recommended Reading

1. Chi E.H. A taxonomy of visualization techniques using the data state reference model. In Proc. IEEE Symp. on Information Visualization, 2000, pp. 69–75.
2. dos Santos S. and Brodlie K. Gaining understanding of multivariate and multidimensional data through visualization. Computer & Graphics, 28(3):311–325, 2004.
3. Haber R.B. and McNabb D.A. Visualization idioms: A conceptual model for scientific visualization systems. In Visualization in Scientific Computing, G.M. Nielson, B. Shriver, L.J. Rosenblum (eds.). IEEE Computer Society, WA, USA, 1990, pp. 74–93.
4. Schumann H. and Müller W. Visualisierung – Grundlagen und allgemeine Methoden (in German). Springer-Verlag, Berlin, 2000.

Visualization Reference Model

▶ Visualization Pipeline

Visualizing Categorical Data

Ali Ünlü, Anatol Sargin
University of Augsburg, Augsburg, Germany

Synonyms

Visualizing categorical data; Graphics for discrete data; Visual displays of nonnumerical data; Plots for qualitative information

Definition

Categorical data are data recorded about units on variables which take values in a discrete set of categories. Examples of categorical variables are gender, citizenship, or number of children. Categorical variables can be dichotomous (two categories; e.g., gender) or polytomous (more than two categories; e.g., citizenship), and nominal (unordered categories; e.g., gender) or ordinal (ordered categories; e.g., number of children). Categorical data can be in case form (individual raw data vector recorded about each unit) or frequency form (tabulated data counting over the categories of the variables), and univariate or multivariate (including bivariate). Quantitative variables can be discretized to become categorical variables (e.g., using child and adult instead of exact age). Strictly speaking, all data may be considered categorical because of limited precision of measurement.

Visualizing categorical data, indeed visualization of data in general, seeks to (i) summarize the data, (ii) expose information and structure in the data, (iii) supplement the information available from analytic measures on the data, and (iv) suggest more adequate analytics for the data.

Key Points

Graphics for univariate categorical data are barcharts and stacked barcharts (for vertically drawn bars, all bars have the same width), and spineplots (modified barcharts where, for vertically drawn bars, all bars have the same height). The area of a bar represents the count for its category. Whereas (stacked) barcharts enable a good comparison of (cumulative) absolute counts, spineplots enable a direct comparison of proportions. When interactively highlighting a subgroup of units, spineplots allow visually comparing the proportions in different categories by looking at the heights of the highlighted areas [3]. Another popular, yet widely criticized, graphic is the pie chart (with partial, exploded, or perspective variants), for displaying shares.

Graphics for multivariate categorical data are mosaic plots and their variations equal binsize and doubledecker plots (for comparing highlighted proportions), fluctuation diagram (for identifying common combinations), and multiple barcharts (for comparing conditional distributions) [2]. Mosaic matrices, as a categorical analog of the scatterplot matrix, consist of all pairwise mosaic plots for two-way subtables [1]. Other powerful graphics are treemaps and trellis displays [2,3], for visualizing hierarchies and conditional structures, respectively. There are graphics specifically designed for bivariate categorical data, for instance, fourfold displays and sieve diagrams (parquet diagrams) [1]. For 2 × 2 tables, the fourfold display visualizes the association between variables in terms of the odds ratio, with confidence rings providing a visual test of whether the odds ratio differs significantly from 1. Sieve diagrams provide displays of the pattern of association in general $r \times c$ tables.

Influence and diagnostic plots for such categorical data models as logistic regression, loglinear, and logit models (e.g., half-normal probability plots) provide examples of model-based visualizations of categorical data [1,2]. These plots help to evaluate model fit. Dimension reduction methods such as correspondence analysis and biplots provide visualizations of associations in n-way tables in a small number of dimensions [1].

Cross-references

- ▶ Data Visualization
- ▶ Multivariate Visualization Methods
- ▶ Visual Analytics
- ▶ Visualizing Quantitative Data

Recommended Reading

1. Friendly M. Visualizing Categorical Data. SAS Institute, Cary, NC, 2000.
2. Hofmann H. Mosaic plots and their variants. In Handbook of Data Visualization, C.H. Chen, W. Haerdle, A.R. Unwin (eds.). Springer, Berlin Heidelberg New York, 2008.
3. Unwin A.R., Theus M., and Hofmann H. Graphics of Large Datasets. Springer, Berlin Heidelberg New York, 2006.

Visualizing Clustering Results

Alexander Hinneburg
Martin-Luther-University Halle-Wittenberg,
Halle/Saale, Germany

Synonyms

Dendrogram; Heat map

Definition

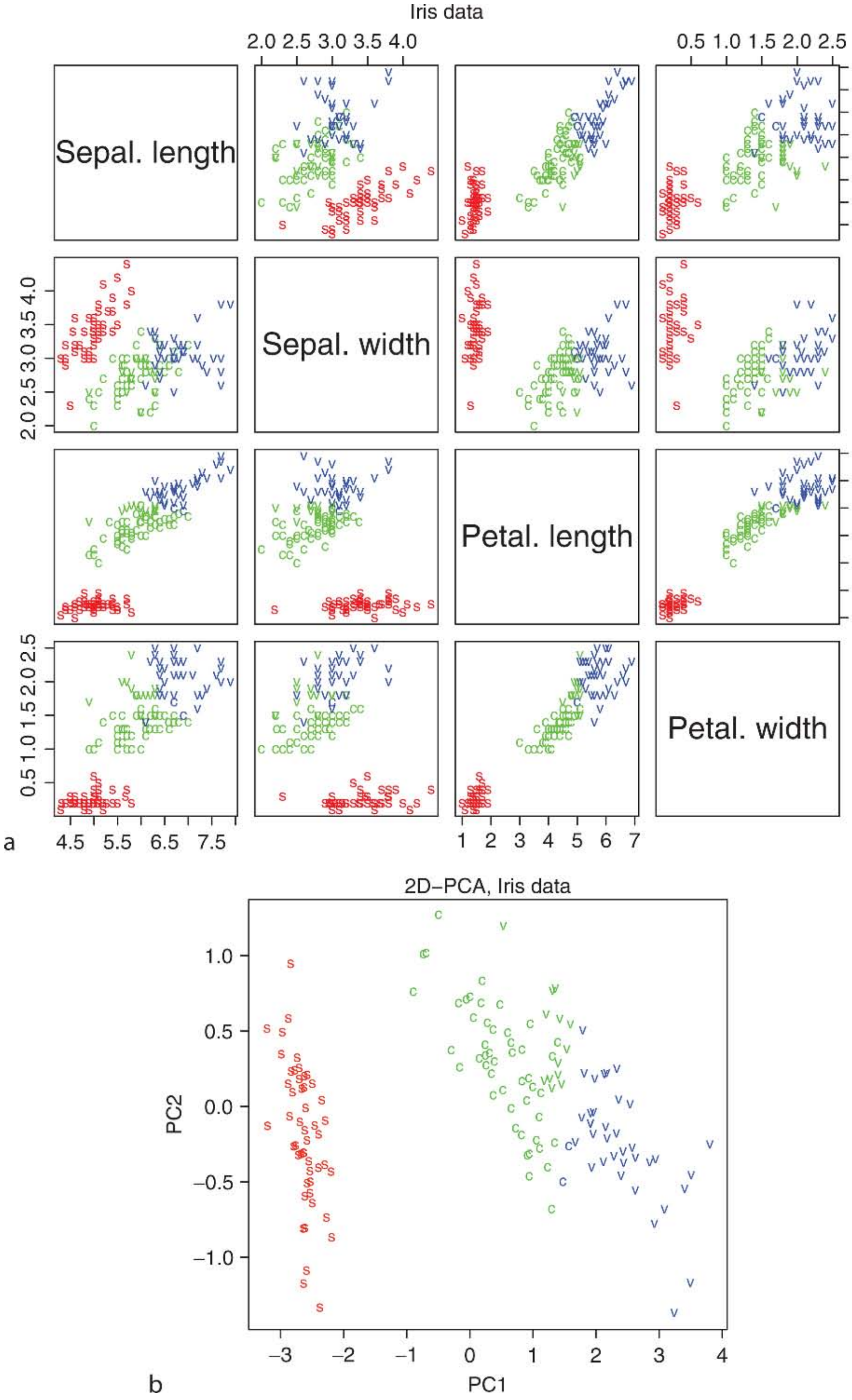
Visualizing clusters is a way to facilitate human experts in evaluating, exploring or interpreting the results of a cluster analysis. Clustering is an unsupervised learning technique, which groups a set of n data objects $D = \{x_1,...,x_n\}$ into clusters, so that objects in the same cluster are similar, and objects from different clusters are dissimilar to each other. The data can be available (i) as $(n \times n)$ matrix of similarities (or dissimilarities), and (ii) as $(n \times d)$ data matrix, which describes each data object by a d-dimensional vector. The second form has to be accompanied by a suitable similarity or dissimilarity measure, which computes for a pair of d-dimensional vectors a (dis)similarity score. A typical example of such a measure is the Euclidian metric. Clustering results may come in different forms: (i) as a partition of D, (ii) as a model, which summarizes properties of D and (iii) as a set of hierarchically nested partitions of D. Visualizations of those results focus one or several properties like the similarity relations between clusters and/or the underlying data objects, the components of the clustering model, or the representation of the data objects. Cluster visualization serves to determine semantics of clusters relevant to the application at hand by inspection, to check whether the cluster model and its parametrization (e.g., number of clusters) fits the data in a meaningful way, or to explore hierarchical relationships between clusters.

Historical Background

Clustering has been used as a technique to explore the structure of data. However, the results are abstract in nature, and in general the found cluster structure is not directly accessible to human experts as an overall picture. Data exploration is also facilitated by directly visualizing the data itself without clustering it.

Simple examples include the scatter plots technique of vector data, which shows 2D projections to all possible $d(d-1)/2$ pairwise attribute combinations. Independently computed results of partitioning clustering algorithms can be shown by labeling the points in the scatter plots according to cluster membership. An example is shown in Fig. 1a.

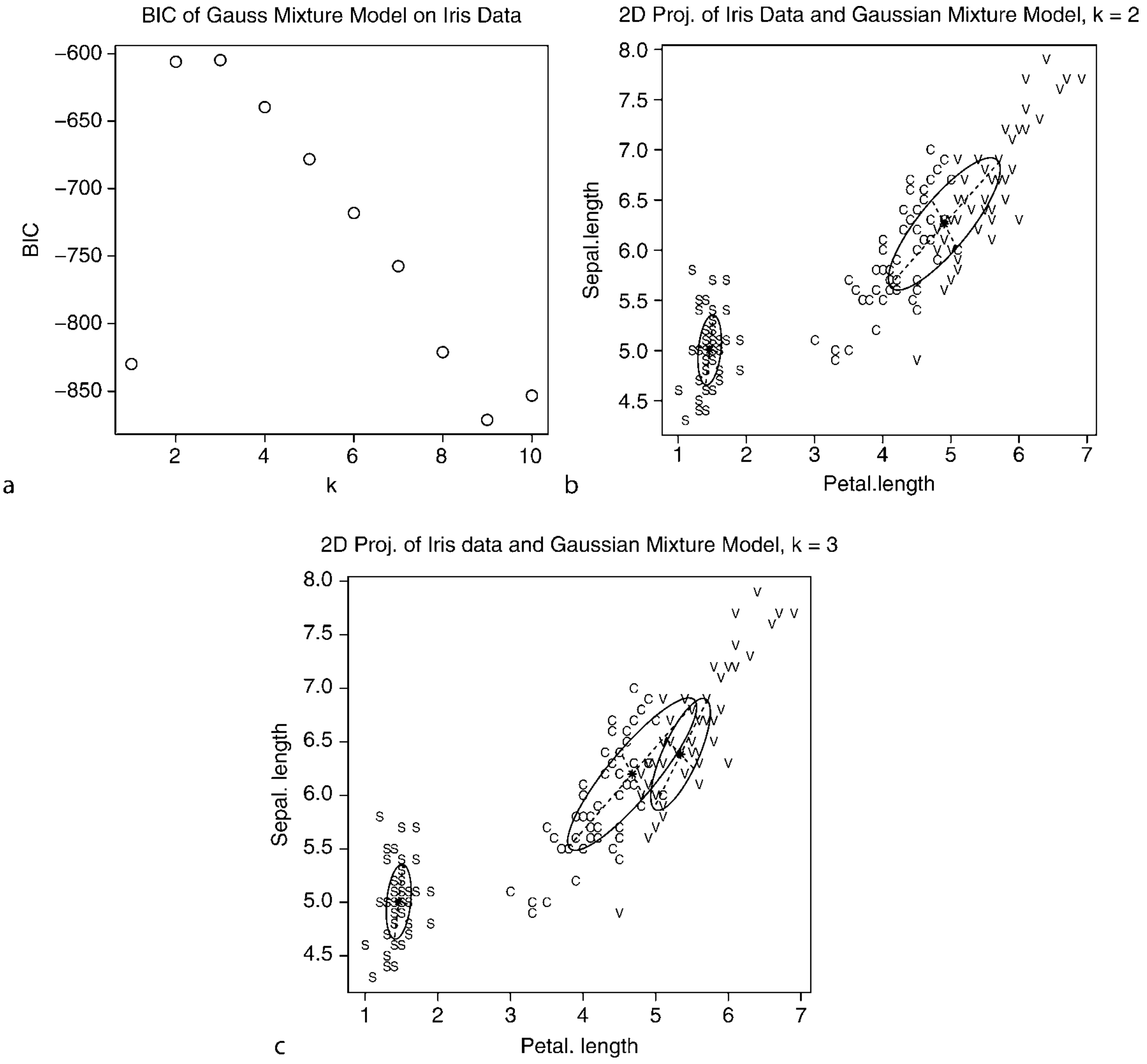
As the display of all 2D scatter plots requires quite a large view space, alternative techniques turn to dimensionality reduction to derive a single 2D projection of the vector data, which (approximately) optimizes some criterion. Dimensionality reduction techniques for data visualization include principle component analysis (PCA) using the first two principal components, classic multi-dimensional scaling (MDS), which performs PCA on the (dis)similarity matrix, isoMDS which minimizes the sum of squared differences between the original (dis)similarities and those in the projected 2D-space. New techniques of this sort include, FASTMAP [5] and latent probabilistic models like probabilistic and Bayesian PCA [4]. The optimization criteria differ from technique to technique but the results are always sets of 2D points. The main property is that



Visualizing Clustering Results. Figure 1. (a) Scatter plot the four-dimensional Iris data, (b) 2D-PCA of the Iris data. The classes Seritosa (S), Versicolor (C) and Virginica (V) are coded by letter and the membership of the clusters is coded by color *red*, *green*, *blue*.

dimensionality reduction and clustering are independently applied to the original data, and the results of both are combined in the visualization.

Another need for visualizing clustering results beside data exploration stems from cluster model evaluation and selection. Clustering methods require parameters, e.g., some algorithms need the number of clusters *k* as an input parameter. There is a large number of proposed validation measures to tackle the particular problem of choosing the right value for *k*. A typical example is the Bayesian information criterion (BIC), which balances the number of parameters with the model performance measured as likelihood. An example of BIC for a Gaussian mixture model applied to the Iris data is shown in Fig. 2a. Gaussian mixture models are a probabilistic extension of k-means. Note, that BIC gives no clear indication, that $k = 2$ or $k = 3$ is better. However, despite that measures like BIC are good indicators for model selection, they may also mislead the user [5] in cases where the model oversimplifies the underling data. In general, there is no broadly accepted measure, that solves the problem of model selection. So, cluster analysis often requires manual tuning of the algorithms to the application at hand. Visualization serves as a tool in the process of evaluating the performance of a clustering algorithm. However, due to its subjective nature, visualization is mainly used here as an auxiliary evaluation method.



Visualizing Clustering Results. Figure 2. (a) BIC of a Gaussian mixture model fitted to the Iris data for $k = 1,...,10$. The shown BIC-values are averages of 10 runs (higher BIC-values is better). (b,c) show clusterings for $k = 2,3$.

Foundations

A simple but powerful method to visualize the results of clustering is to visualize the cluster model itself. A straightforward example is *k*-means, which has been used to determined the clusters (shown by color) in Fig. 1a, b. Note, that in this case the cluster model, which consists of the cluster centers only, is shown indirectly by color coding the cluster membership. The cluster centers, which are the means of the clusters, can be visually approximated. An explicit visualization would be required to show the coordinates of the cluster centers and perhaps the decision boundaries of the clusters, which correspond to the edges of the Voronoi diagram induced by the cluster centers.

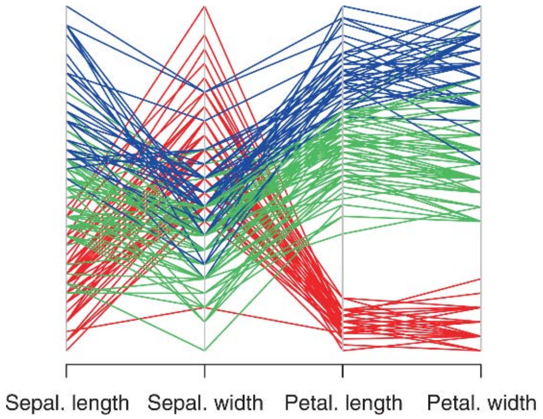
A more complex cluster model is the Gaussian mixture model, whose components consist of multi-dimensional Gaussian distributions. Such a distribution also has in addition to the center coordinates, parameters for the covariances between the dimensions. Figures 2b, c show some 2D-projections of the Gaussian models mixture models for two and three components fitted to the Iris data. Each component is visualized as an ellipse, whose center corresponds to the center of the Gaussian, and the orientation of the ellipse reflects the correlations between dimensions. Specifically, each ellipse shows the points which have the same probability density with respect to the corresponding Gaussian. The figures do not show the cluster memberships of the data points, because the Gaussian mixture model only gives a posterior distribution $P(c|\vec{x})$ for a point $\vec{x}$ and a cluster c, which allows several options for associating points to clusters. The most common method is to associate a point to the cluster with the maximum posterior. The shown visualizations combine both data and cluster models. However, there are many cases, when this scenario is not directly applicable. The reasons might be that the data does not allow a direct visualization in a scatter plot style, or the dimensionality of the vector data is too high to be easily reduced to 2D-projections by picking dimensions by hand. A typical example is document clustering, where the data objects are documents. Documents are either represented as sets of words or high dimensional word-count vectors. Both cases do not allow a straightforward two-dimensional visualization of the document collection.

A general method is parametric embedding (PE) [10], which takes the posterior distributions of the data objects with respect to some computed clusters, and fits a Gaussian mixture model of two-dimensional data points, so that the posteriors of the mixture model reflect the original posteriors. The model fitting of parametric embedding is done by minimizing the sum of Kullback–Leibler divergences between original posterior distributions and those from the two-dimensional Gaussian Mixture model. Note, that parametric embedding determines both the parameters for the two-dimensional Gaussian components of the mixture model, as well as the positions of the two-dimensional data points onto which the original data objects are mapped to. This technique allows the visualization the clustering results of most probabilistic clustering algorithms, regardless of the type and dimensionality of the data.

In the case where posterior probabilities are not available from the clustering algorithm, neighborhood component analysis (NCA) [8] can be applied for cluster visualization. NCA assumes vector data with given cluster (or class) labels and minimizes a stochastic version of the leave-one-one k-nearest-neighbor classification by learning a Mahalanobis metric. Therefore, NCA falls into the class of metric learning. Cluster visualization is achieved by restricting the rank of the learned Mahalanobis matrix to two or three. In such settings, the linearly transformed data can be visualized as scatter plots in two- or three-dimensional space.

An improvement of the PE technique is the combination of clustering and visualization [11], where the two-dimensional visualization acts like a regularizing prior distribution for the clustering model. The improvement is that the cluster membership values for each object are not fixed during the calculation of the visualization. So, both the parameter set for the clustering model and the parameters for the visualization are estimated simultaneously. In the case of high-dimensional data, the regularizing visualization component helps to avoid overfitting of the clustering model. Although, the experiments are done with a simple Gaussian mixture model (e.g., see Fig. 2), the improved technique could be used in combination with any probabilistic clustering model.

Instead of using scatter plots, which represent multi-dimensional data by two-dimensional points, parallel coordinates [1] can be used, which have no need for dimensionality reduction (e.g., see Fig. 3). Parallel coordinates visualize a multi-dimensional vector as a sequence of consecutive line segments, and visually scale up to 15 dimensions. The coordinates



Visualizing Clustering Results. Figure 3. K means clustering ($k = 3$) of the iris data shown in parallel coordinates.

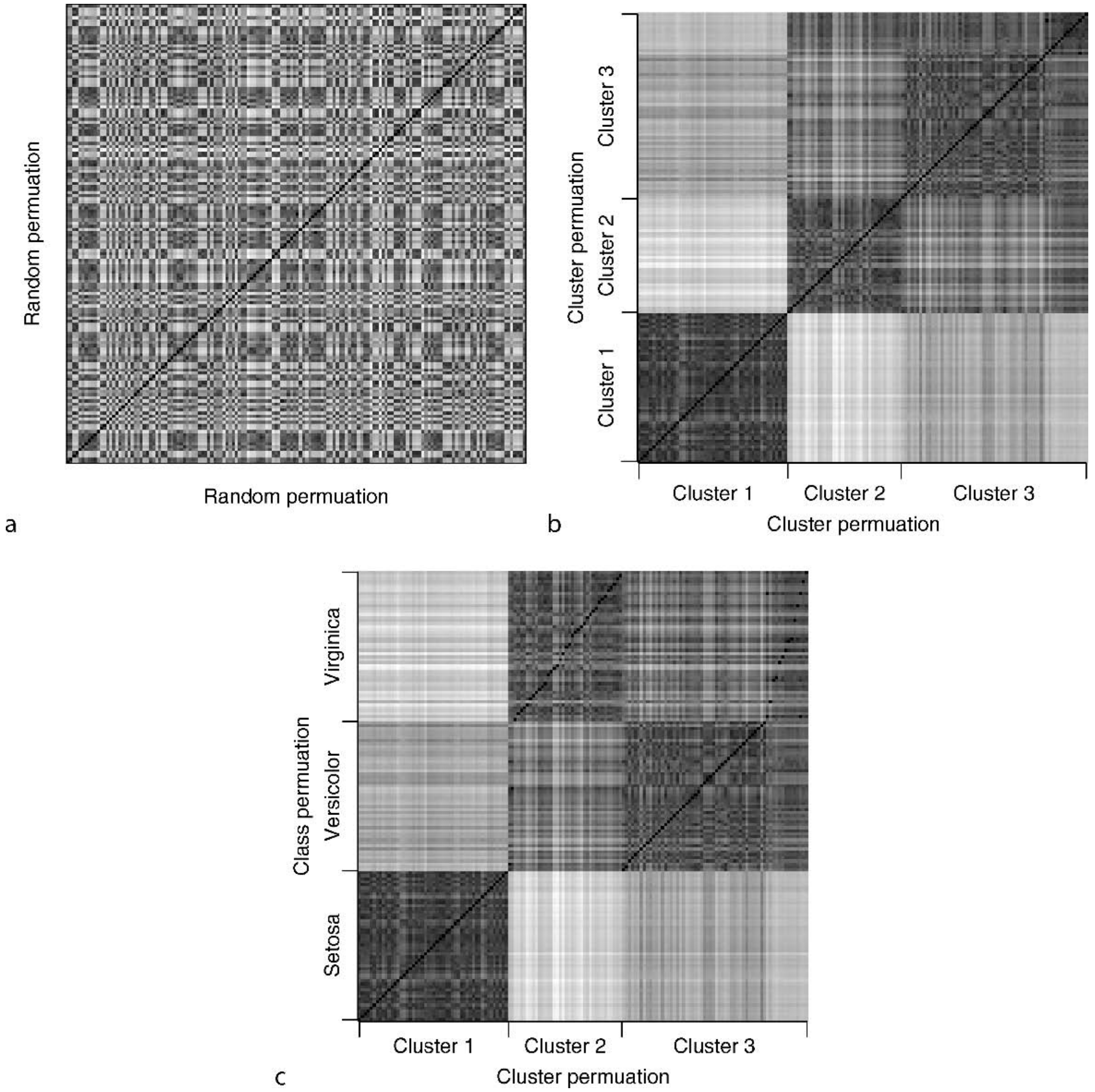
axes are represented by vertical lines. Simple visualizations show the clustering by color coding the cluster membership. A more advanced technique is proposed in [7]. Cluster centers are visualized by parallel coordinates. Additionally, the spans of the clusters projected to the attributes are also visualized by drawing opaque tubes of varying thickness around the line segments of the cluster centers.

So far, clusters have been visualized by showing a possibly transformed version of the data objects itself, together with some information about the clustering and/or the cluster model. A principle alternative to that approach is to visualize distance or similarity relations between the data objects and/or the derived clusters. In the next paragraphs, we explain the methods wrt. to object distances to make the discussion simpler, however, similarity measures could be used as well. The amount of distances between n data objects scales quadratically in n, which makes it difficult to visualize all pairwise distances of large data sets (e.g., $n \gg 1{,}000$). On the other side, visualizing distances or similarities is applicable to a much wider spectrum of data types, in contrast to the previous approaches, which assumed a given or derived representation of the data objects as vectors. Thus, visualizing relative information is attractive for looking at clusterings of sequences, trees, graphs, sets, and other non-vector like data.

The direct visualization of the distance matrix $\mathrm{D} \in \mathbb{R}^{n \times n}$ of a data set codes the values of a matrix entry d_{ij} by color or gray value of a pixels, while the two indices i and j are coded by the pixel's position. In order to visualize a clustering, the columns and rows are partially ordered into blocks according to cluster membership. In a case where the clustering is meaningful, the change of the column and row permutation from a random to the partial ordering induced by the clustering has a huge visual impact. In the ideal case, clusters form blocks which are placed along the diagonal of the matrix. Possible relations between clusters show up as off-diagonal blocks in such a plot. This technique also allows the comparison of a clustering with some ground truth, if available, by ordering the rows and columns according to clustering and class labels, respectively. Examples are shown in Fig. 4. As the distance values sometimes have some skewed distribution, it improves the visual results to apply a monotone sub(super)-linear transformation to the distance values before mapping them to color to increase (decrease) the visual contrast. In Fig. 4, the square roots of the distances are shown. Non-clustering methods to find a good permutation of the distance matrix are discussed in [9] under the term seriation.

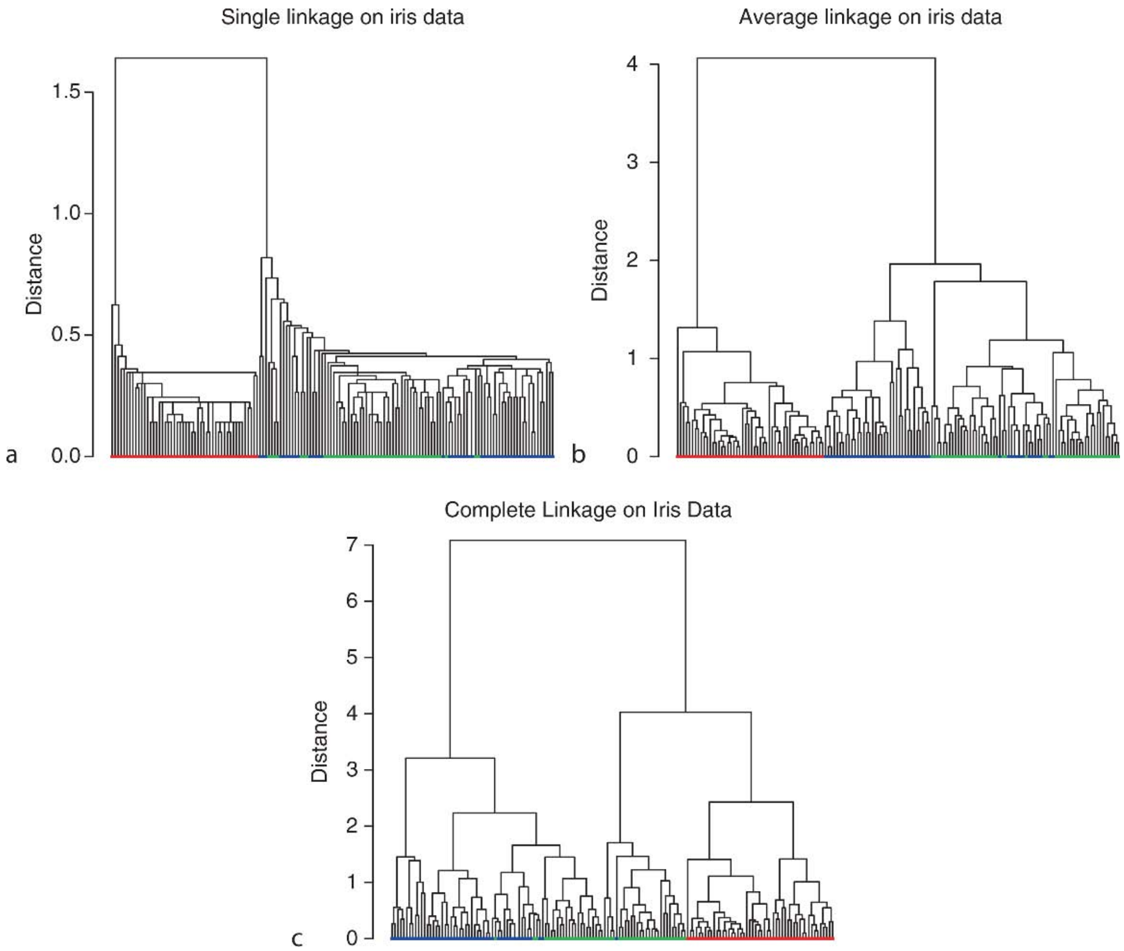
Hierarchical clustering is one of the earliest clustering methods. The generic, agglomerative, bottom-up algorithm starts with a clustering where each data object is its own cluster. In each step, the closest pair of clusters is found (i.e., the pair of clusters with smallest distance) and merged to form a new cluster. The distance between clusters C and C′ is induced by the distance between objects in different ways, e.g., single linkage $d(C, C') = \min_{x \epsilon C, x' \epsilon C'} d(x, x')$, average linkage $d(C, C') = 1/|C| \cdot |C'| \Sigma_{x \epsilon C} \Sigma_{x' \epsilon C'} d(x, x')$ or complete linkage $d(C, C') = \min_{x \epsilon C, x' \epsilon C'} d(x, x')$. After $\mathrm{n} - 1$ steps, a hierarchy of nested clusterings is determined, which is represented as a binary tree. The visualization of such a tree is called a dendrogram, when the height of an inner node is the distance between the merged clusters. Leaf nodes have zero heights. The height of a complete linkage dendrogram is the maximal distance between two objects, the height of a single linkage dendrogram is the maximal edge in the minimal spanning tree of the data, and the height of an average linkage dendrogram is something between the two other dendrograms. Figure 5 shows different dendrograms of the iris data.

Overall, a dendrogram shows $n - 1$ distances between clusters, which is much less distance information than the original distance matrix. So, a dendrogram can be seen as a lossy compressed form of



Visualizing Clustering Results. Figure 4. Distance matrix of the iris data: (a) in random order, (b) columns and rows are ordered according to a k-means clustering with $k = 3$, and (c) rows are ordered according to a k-means clustering with $k = 3$ and columns are ordered by ground truth class labels.

the distance matrix. The number of possible visualizations of the same dendrogram is 2^{n-1}, which can be seen by noting that the orientation of each inner node has two possible states. So, the super-exponential problem of finding a good visualization of the similarity matrix among $n!$ possibilities, is reduced to the exponential problem of finding a good dendrogram visualization among 2^{n-1}. Typically, heuristics are used to find a useful orientation of a dendrogram, but in [3,12] more principled optimization techniques are used to that end. The derived permutation of the leafs, which correspond to the data objects, is also used as a meaningful ordering of the rows and columns of the distance matrix [14]. In the special case of vector data, a similar visualization is derived by hierarchically clustering the rows and columns of the data matrix, which is built by concatenating the d-dimensional data vectors $x_1,...,x_n$ to a $n \times d$ matrix. The respective dendrograms for the row and the column set are shown at the borders of the matrix. Both variants are called heatmap. Examples of the iris data are shown in Fig. 6. These kinds of



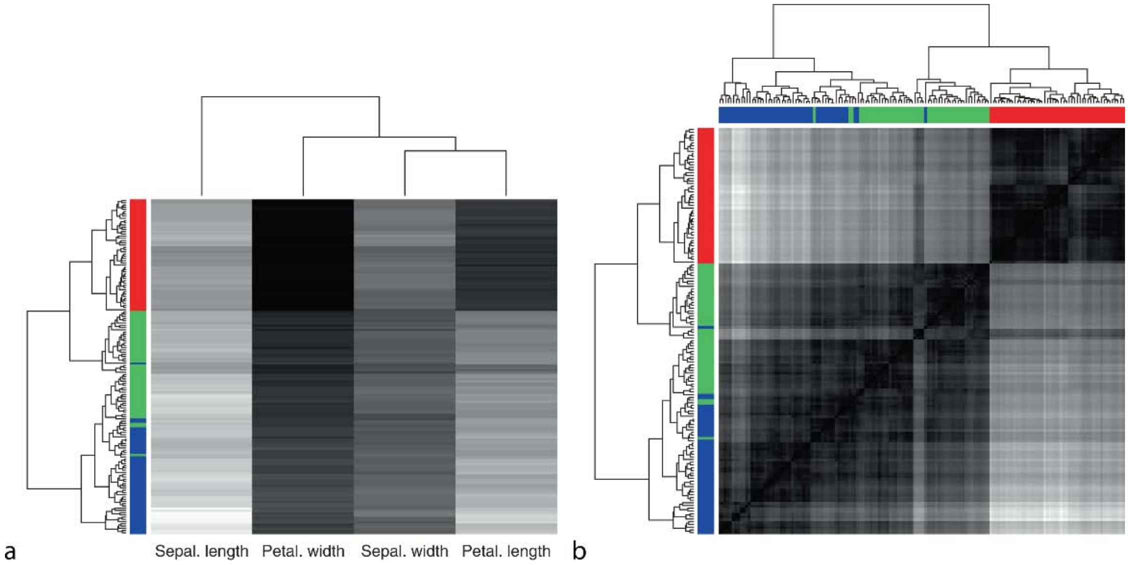
Visualizing Clustering Results. Figure 5. Dendrograms of the Iris data.

visualizations are especially popular in gene expression analysis.

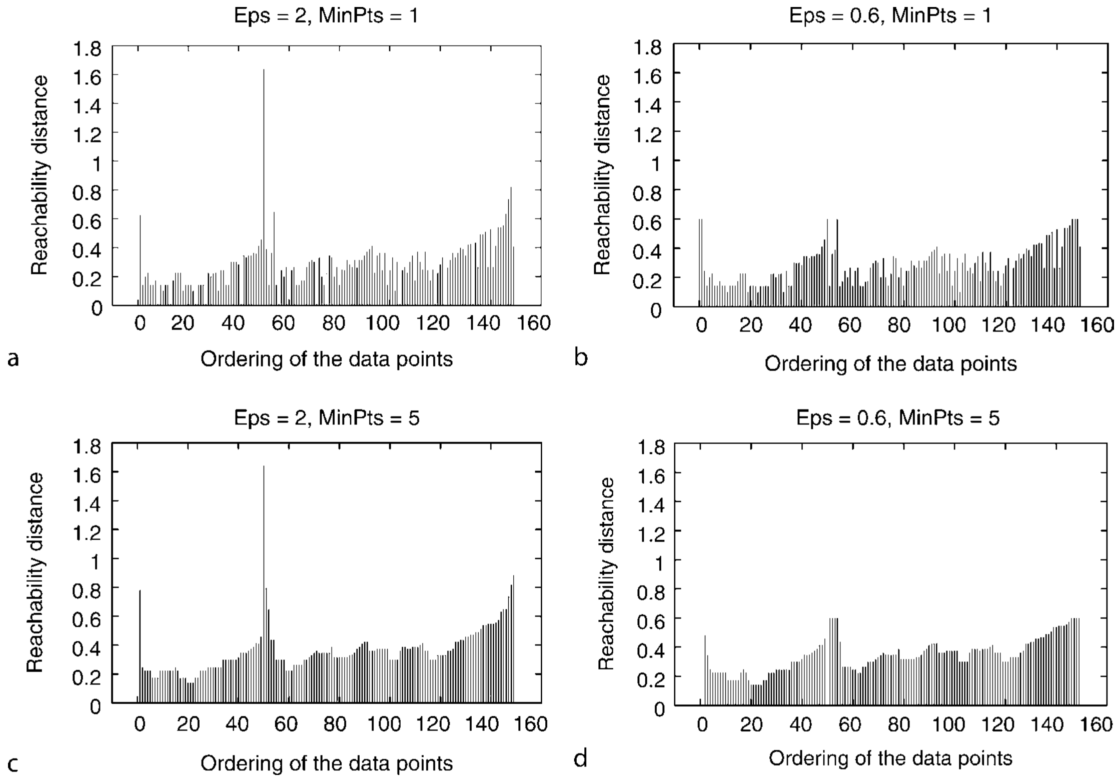
As the width of a dendrogram scales linearly with the data size, the clustering of a large data set is difficult to visualize with that technique. Therefore, heuristics are used to compute a horizontal cut in the dendrogram, and only the upper part is visualized. However, cutting at a constant height does not always produce the best result, thus new more flexible heuristics are discussed in [13].

In the case of single-linkage-like cluster hierarchies, the OPTICS approach [2] visualizes the lower part of the hierarchy with an alternative technique. OPTICS extends flat single linkage clustering. For visualization purposes it computes an ordering of the data points, which reveals nested clusters.

Flat single linkage clustering sees the data as a graph, with the data objects as nodes. An edge connects two nodes, when the corresponding data objects are closer than a given threshold (*EPS*). Clusters are the connected components of the graph. A drawback of single linkage clustering is that well separated clusters may be connected by chains of outliers and are merged by the algorithm. Therefore, OPTICS extends the basic single linkage approach by introducing an additional linkage constraint, the core object condition. The condition says that every object in a cluster is linked to at least one core object. A core object has at least *MINPTS* ≥ 1 other data objects in the neighborhood of a radius *EPS*. Graph theory says that connected components can be found by two equivalent algorithms, namely depth first search (DFS) and breath first search (BFS). BFS is mostly implemented by a first in first out (FIFO) queue. OPTICS uses a priority queue instead of a simple FIFO queue, which sorts the internal objects by ascending reachability distance. The reachability



Visualizing Clustering Results. Figure 6. Heatmaps of the Iris data showing (a) the data matrix and (b) the Euclidian distance matrix. The used dendrograms are computed by complete linkage.



Visualizing Clustering Results. Figure 7. OPTICS on Iris data.

distance of an object is defined as the distance to the *MINPTS*-nearest neighbor in the case of core objects, as the distance to the nearest core object in the case of border objects, which are linked to a core object, and infinity in the case of outliers. A good intuition of the reachability distance is to think of it as the inverse of data density. Objects with a low reachability distance have many neighbors, thus are in areas of high density. As the algorithm uses a priority queue, which outputs objects with a low reachability distance first, it focuses on objects in high density areas first. The mode of operation of OPTICS is that it starts randomly somewhere in a cluster, is then lead to the clusters center and works its way to the border. After a cluster is finished, it starts with the next one in the same manner.

The visualization technique of OPTICS displays the objects in the order they are processed by the algorithm, and shows the reachability distance for each object. By the mode of operation of OPTICS, this methods shows a cluster as a valley. The left border of the valley is formed by the random start, which is with high probability an object with some larger reachability distance. The bottom of the valley is formed by the center objects of the cluster, which have the smallest reachability distance. The right border of a valley is formed by the border objects of the cluster. In that case, the cluster is perfectly separated from the rest of the data, the priory queue becomes empty and the algorithm jumps randomly to some still unlabeled object and processes the next cluster. Nested cluster structure shows up as dents in the bottom of a larger valley. As OPTICS never links objects further away than *EPS*, it does not determine links between clusters, which are separated by more than *EPS*. Thus, the visualization shows only the lower part of the hierarchy. Examples of OPTICS plots are shown in Fig. 7.

Key Applications

The visualization of clustering results is mainly used for interpreting the results of clustering and presenting them to people from the application side. It is a good tool to put the derived clusters into the context of the application at hand by annotating the visualization with meta-information. Visualizing clustering results also facilitates the exploration of the data and the tuning of clustering algorithms.

URL to Code

The images in this article (except the OPTICS plots) are produced by the statistics package R. The code for the drawings can be found at http://www.informatik.uni-halle.de/~hinnebur/clusterVis.tar.gz.

Cross-references

- ▶ Clustering Overview and Applications
- ▶ Clustering Validity
- ▶ Visual Classification
- ▶ Visual Clustering
- ▶ Visual Data Mining

Recommended Reading

1. Alfred Inselberg. Parallel coordinates: VISUAL multidimensional geometry and its applications, Spring, 2007.
2. Ankerst M., Breunig M.M., Kriegel H.-P., and Sander J. Optics: Ordering points to identify the clustering structure. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 49–60.
3. Bar-Joseph Z., Gifford D.K., and Jaakkola T.S. Fast optimal leaf ordering for hierarchical clustering. Bioinformatics, 17(90001):22–29, 2001.
4. Bishop C. Pattern Classification and Machine Learning. Springer, New York, 2006.
5. Domingos P. Occam's two razors: The sharp and the blunt. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998, pp. 37–43.
6. Faloutsos C. and Lin K.I. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 163–174.
7. Fua Y.-H., Rundensteiner E.A., and Ward M.O. Hierarchical parallel coordinates for visualizing large multivariate data sets. In Proc. IEEE Conf. on Visualization, 1999.
8. Goldberger J., Roweis S.T., Hinton G.T., and Salakhutdinov R. Neighbourhood components analysis. In Advances in Neural Inf. Proc. Syst. 18, Proc. Neural Inf. Proc. Syst., 2005, pp. 513–520.
9. Hahsler M., Hornik K., and Buchta C. Getting Things in Order: An introduction to the R package seriation. http//:cran.at.r-project.org/web/packages/seriation/vignettes/seriation.pdf.
10. Iwata T., Saito K., Ueda N., Stromsten S., Griffiths T.L., and Tenenbaum J.B. Parametric embedding for class visualization. Neural Comput., 19(9):2536–2556, 2007.
11. Kaban A., Sun J., Raychaudhury S., and Nolan L. On class visualisation for high dimensional data: Exploring scientific data sets. In Proc. 9th Int. Conf. on Discovery Science, 2006.
12. Koren Y. and Harel D. A two-way visualization method for clustered data. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 589–594.
13. Langfelder P., Zhang B., and Horvath S. Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R. Bioinformatics, 24(5):719–720, 2008.
14. Strehl A. and Ghosh J. Relationship-Based Clustering and Visualization for High-Dimensional Data Mining. INFORMS J. Comput., 15(2):208–230, 2003.

Visualizing Hierarchical Data

Graham Wills
SPSS Inc., Chicago, IL, USA

Synonyms

Hierarchical graph layout; Visualizing trees; Tree drawing; Information visualization on hierarchies; Hierarchical visualization; Multi-level visualization

Definition

Hierarchical data is data that can be arranged in the form of a tree. Each item of data defines a node in the tree, and each node may have a collection of other nodes as *child nodes*. The relationship between the parent nodes and the child nodes forms a *tree network*. The formal definition of a tree is that the graph formed by the nodes and edges (defined between parent and child node) is both connected and contains no cycles. The following properties of a tree are of more practical use from the point of view of displaying visualizations:

- One node, called the *root* node, has no parent.
- All other nodes have exactly one parent.
- Nodes with no children are termed *leaf* nodes. Nodes with children are termed *interior* nodes.
- For all nodes in a tree, there is a single unique path up the tree going from parent to parent's parent and so on, which will terminate in the *root* node.
- The number of nodes on the path from a node to the root is termed its *depth*.

Although not strictly required, the vast majority of hierarchical data, and the main application area, consists of trees where the parent nodes define some form of aggregation on the child nodes, so that the data for the parent is equal to, or is expected to be close to, the aggregation of the data for that node's children. The relationship between parent and child is often described in terms of inclusion; it is common to state that nodes *contain* their children. This aspect is often brought out in visualizations.

A simple example of hierarchical data would consist of populations for the world, hierarchically broken down into sub-regions. At the top level, the root would consist of the world, with data being the population of the world. The next level could be the four main continents, each with their individual population, and each continent would have countries as children, each with their population counts. In each case, one would expect that the population of the parent node would roughly equal the population of the child nodes. If the data were collected at slightly different times, or from different sources, the populations for the continents might not exactly equal the sum of their children's populations, but one would expect it to be very close.

Note that in this example, every *leaf* node has the same *depth*. Although this is a common property in many applications, it is not a required property. In fact, if Antarctica is counted as a continent, it would have no contained countries, and the hierarchy would lose this property. If the level of the hierarchy is further increased, by dividing countries into regions, one would have very different levels. The United States might be divided into states and then into zip codes, whereas Vatican City would not be divided up at all.

Historical Background

Leonhard Euler is regarded as the founder of graph theory, publishing initially in the 1730s, but displays of trees as predate even this, especially displays of family trees, some of which survive from significantly earlier. However, the discipline of visualizing hierarchical data in a systematic fashion dates back only to the availability of computers. In 1992, the conference *International Symposium on Graph Drawing* commenced meeting and their conference proceedings, available from 1994, provide an excellent overall reference to the state of the art in this subject, although limited mainly to static graphs. From around 1990 onwards, increased attention has been paid to *interactive*, or *dynamic* visualization of hierarchical data, although information on such techniques is scattered across many disciplines. User interface controls for interacting with hierarchies became common with the advent of windowed operating systems; trees of folders and files are commonplace and techniques for filtering and pruning such views have seen significant research and user testing.

Foundations

There are a number of reasons why hierarchical data might be visualized, and it is important to identify the goal of any visualization, as different visualization techniques serve different goals. Common reasons to view hierarchies are:

- *Understanding Structure*: The goal is to understand the structure of the hierarchy; in the world population example, one might want to know how countries are distributed within continents, or to see how many major regions are within countries.
- *Understanding Data*: The goal is to understand the distribution of data across the hierarchy. In the example, one might want to know if each continent has a similar distribution of population, or if the lowest levels have similar populations (do zip codes

in the US, on average, have the same populations as the Vatican City?), or similar questions. Often, the goal is to understand the data within the context of the structure.

- *Summarizing large amounts of data*: The goal is to reduce information overload by providing a summary of low-level data into aggregated data. A hierarchy allows the user of a visualization to set the level of detail they want, by only showing data up to a certain level in the hierarchy.

There are two basic branches of visualization techniques for hierarchies. The first is based on a node-edge graph-layout approach which focuses attention on the structure and relationships, and the second on space-filling approaches, which focus attention on the relative sizes of nodes in the hierarchy. Each is discussed below.

Node-Edge Layouts

These displays are essentially a specialization of general graph layout techniques. Each node in the hierarchy is displayed as a small glyph, commonly a circle or a square. Data on the node can be represented by changing aesthetic attributes of the node such as its size, color, pattern, etc. Fig. 1 below gives an example of a traditional node-edge display for hierarchical data. Each node represents a country, and they are aggregated into a hierarchy where the net level up is a sociological grouping. The populations have been aggregated by a mean average, so the node representing the "new world" countries has been given the mean population of countries in the new world. It is clear that there is little difference between populations of countries when they are aggregated into these groupings.

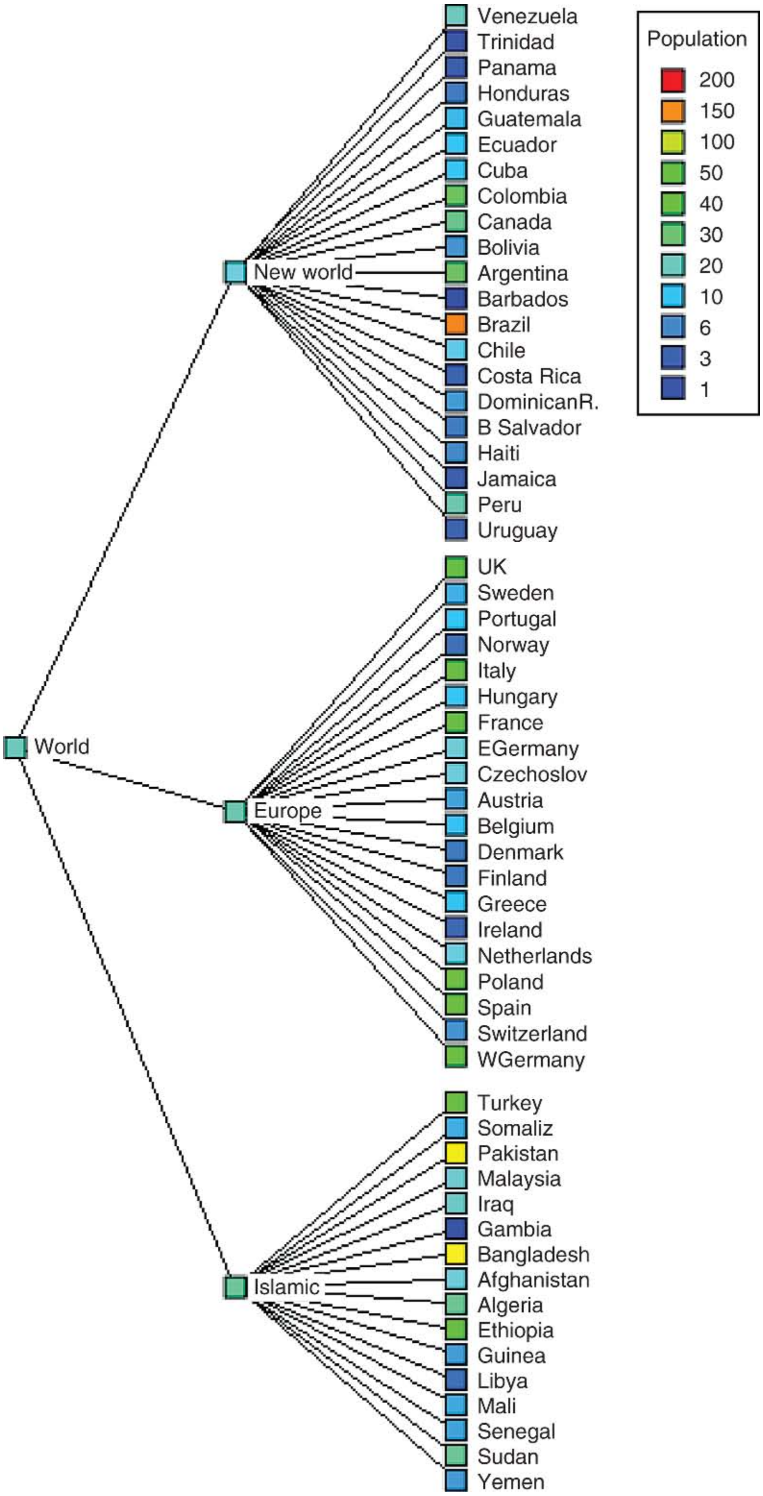
This visualization has the advantage that it is good for showing the *structure* of the hierarchy – one can see clearly how each many countries are in each group, and the distribution of the variable of interest. However, the display is not compact – a lot of space is wasted showing links having little information. Modifications to this basic display include the following:

- *Ordering the nodes within each parent.* The child nodes for each parent could be sorted by a variable. In Fig. 1, it would make sense to sort by the population to create a better overview of the distribution of that variable across countries within a group.
- *Displaying information on the edges.* In the above figure, the edges could be coded according to the similarity between the country and its group, using any of a number of statistical techniques to define such similarity. One such technique starts simply with data items and then generates a hierarchy by successively clustering items into groups based on similarity. This technique is called hierarchical clustering and is often visualized using a dendrogram as in Fig. 2. The dendrogram shows when groups are formed using the vertical dimension; the lower on the vertical dimension, the more similar the groups that were merged were. In Fig. 2, F and G were merged first and because they care the most similar pair, then A and B were formed into a group, then D and E. Then C was added to {A, B}, following which {D, E} was merged with {F, G}. The resulting groups {A, B, C}, {D, E, F, G}, {H} are only merged together at a much higher level of dissimilarity. The dendrogram therefore lets one see not only what clusters were created, but it also, by placing the merge information at a location in the vertical dimension proportional to the similarity of the groups being merged, allows one to see how good the resulting clusters are.

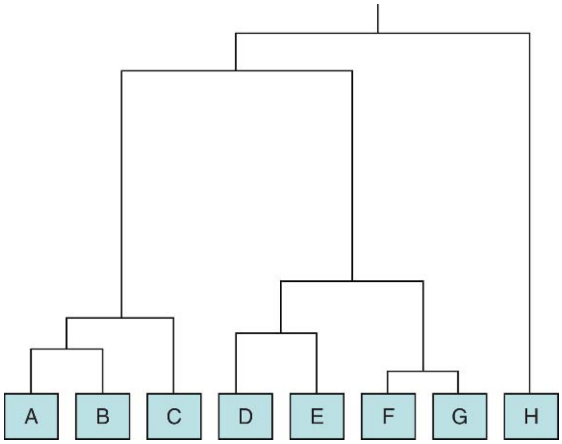
Space-Filling Layouts

In contrast to node-edge layouts, space-filling layouts take explicit advantage of the hierarchical nature of data directly. A space-filling layout is usable only when the main variable of interest is *summable.* As these techniques lay out areas in proportion to sizes, and parents visually include their children, a variable that can be summed is necessary. It is not possible to use space-filling layouts directly to show means, minimums or similar statistics. The base layout is limited to sums. It is, of course, possible to color or use some other non-size aesthetic for such techniques, but then the essential space-filling nature of the display is of little value.

Figure 3 shows the example data with a radial space-filling layout. Each level in the hierarchy is represented by a band at a set radius, with the root note in the center, and children outside their parents. The angle subtended by a node is proportional to the percentage of the entire population that this node represents. Children are laid out directly outside their parents, so parents "divide up" the space for their children according to the child sizes.. This is a typical



Visualizing Hierarchical Data. Figure 1. Standard Node-Edge layout for a hierarchical network.



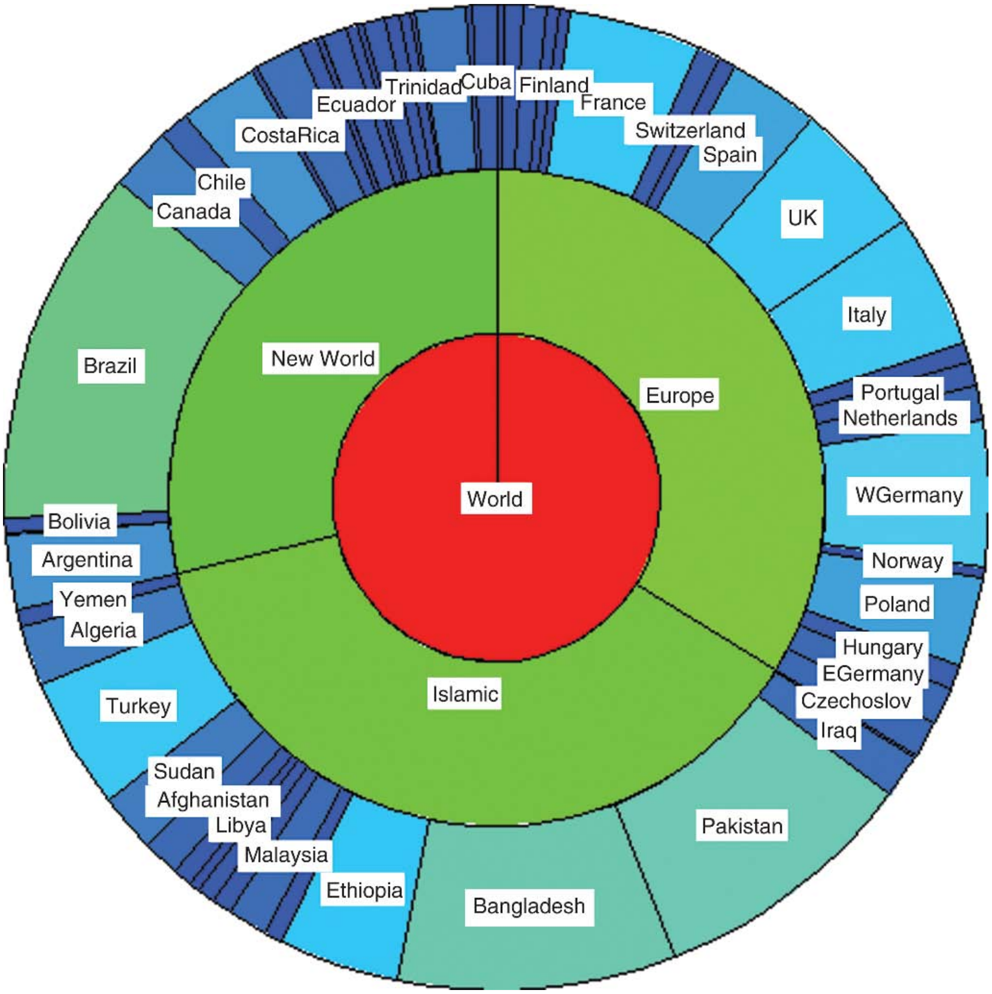
Visualizing Hierarchical Data. Figure 2. Dendrogram of a hierarchical clustering.

space-filling technique, many of which have been discovered in various disciplines. They share the following characteristics:

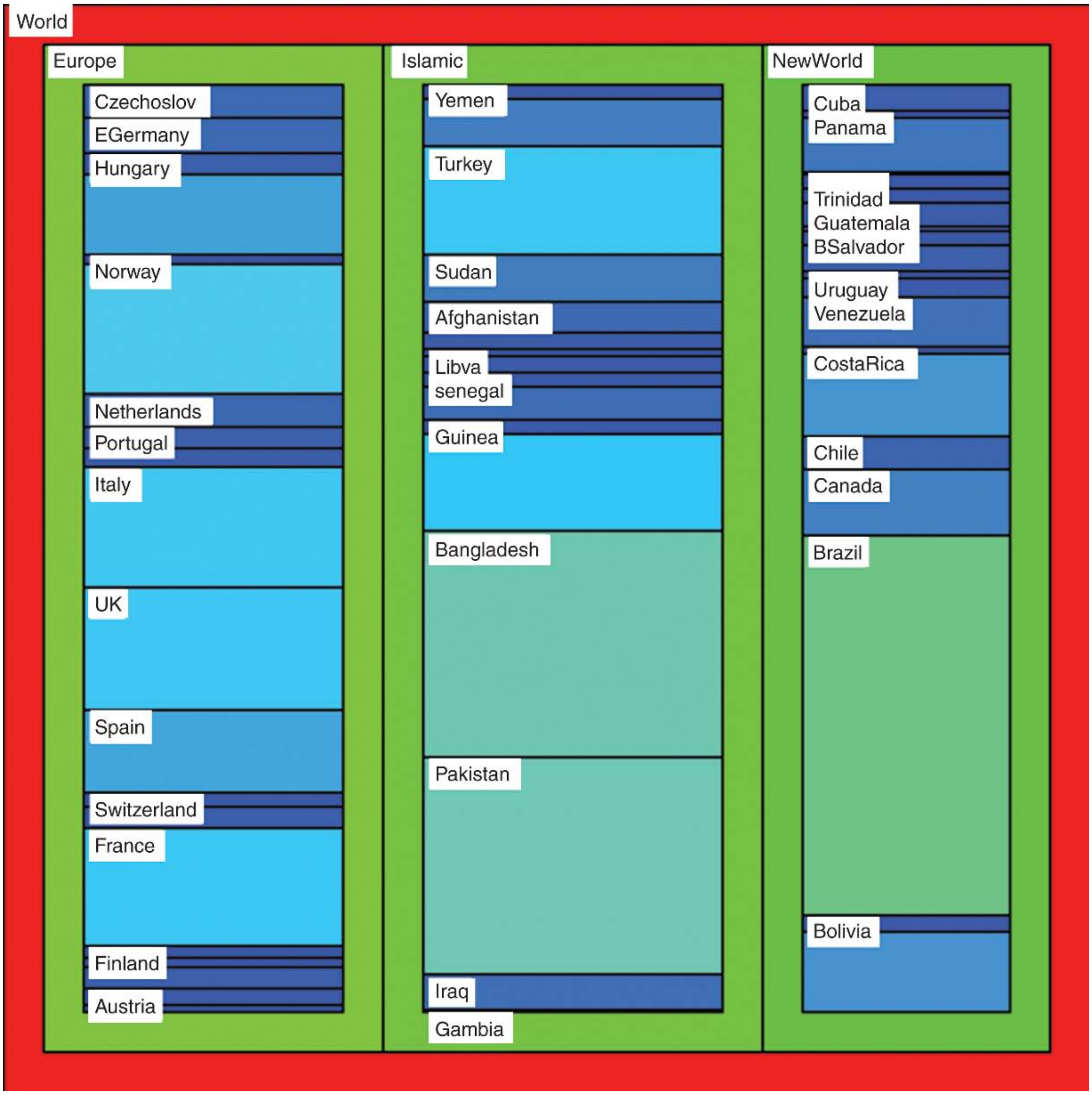
- The root node occupies 100% of the space, or dimension of interest (in this case, the radial dimension).
- Each node partitions its space according to the relative sizes of its children.

Figure 3 uses space more economically than Fig. 1, and also allows one to see sizes more clearly. It is generally to be preferred if the data support it. Compare this figure with Fig. 4, which uses a style of layout popularized under the name “TreeMap.”

The treemap, instead of allocating space for child nodes outside the parent node, lays them out directly



Visualizing Hierarchical Data. Figure 3. The Population data of Fig. 1, this time showing summed population, using a space-filling radial layout.



Visualizing Hierarchical Data. Figure 4. TreeMap version of Fig. 3.

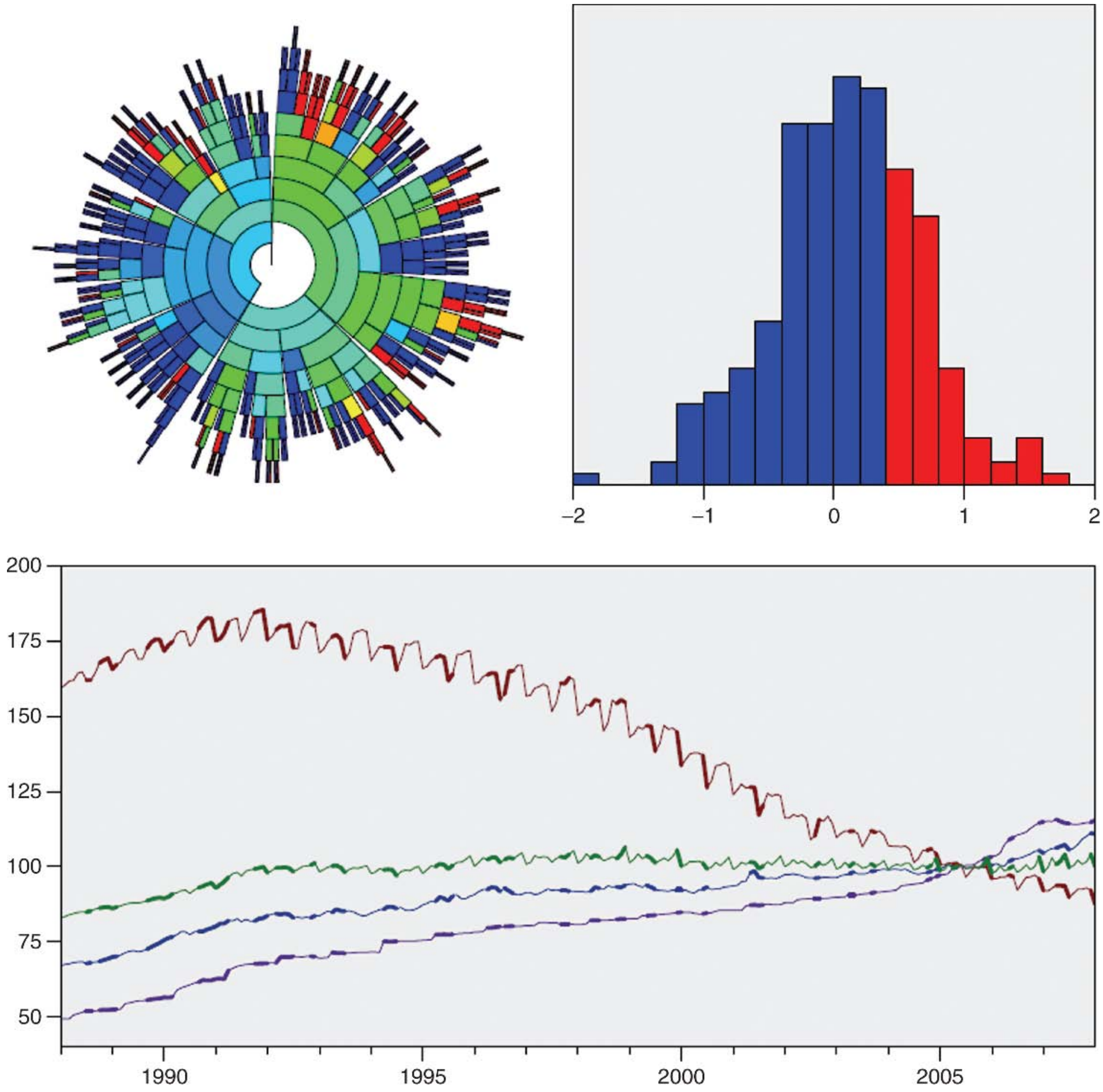
on top of the parent, thus making a maximally compact representation. This causes the immediate problem that there is then no way to distinguish the tree structure, since the children completely occlude the parents. Typically, as in Fig. 4, a small border is left around the child nodes to allow the tree structure to be ascertained, although that does distort the overall relationship between visual size and the sizes of the hierarchical data points. The TreeMap is not a good technique for learning about *structure* because of this issue, but because it is very compact it can perform quite well in the domain to which it is applicable: displaying relative sizes of items in the hierarchy when the structure of the hierarchy is well-known or of little interest. There are numerous different ways to render this figure, and care should be taken to avoid situations where some rectangles are skinny and others are flat; it is harder to make size judgments based on areas with widely differing aspect ratios than it is on arcs with different angles as in Fig. 3, for example. In general, the TreeMap should be treated as a specialist tool for hierarchies to which it is applicable, not as a general purpose hierarchical visualization.

Interactive Visualization of Hierarchical Data

The entry on Visualization of Network Data indicated some techniques for interacting with general networks. These can be applied to the special case of hierarchies, and of these the most applicable and important technique is that of brushing and linking.

Figure 5 demonstrates a number of techniques that have been detailed earlier, and adds interaction to the visualization. The base data are a set of consumer price indices (CPI) from the United Kingdom, collected monthly. A hierarchical clustering has been performed on the months, based on all the CPIs in the data set except the overall CPI. At the bottom is multiple time series chart showing four indices (food, clothing, housing, furniture). At the top right is a histogram showing residuals from a fitted time series mode of the overall CPI. Each view is linked to each other view, so that selecting a region of one chart highlights the corresponding months in all other views.

In this figure, the higher residuals have been selected, denoting months where the actual CPI was higher than the model predicted. This is shown in red in the



Visualizing Hierarchical Data. Figure 5. Linked Views of UK CPI data.

histogram, and in the time series view the segments corresponding to those months are shown with a thicker stroke. It appears that these residuals occur mainly on the downward part of regular cycles on the topmost time series (representing the clothing CPI).

In the hierarchical view, a generalization ahs been made to the linking. As the months are aggregated at higher levels of the hierarchy, these interior nodes in the hierarchical tree map may be partially selected. In this figure the selection percentage is shown with a rainbow hue map, with blue indicating completely unselected, red completely selected, and green half-selected. Intermediate hues indicate intermediate selection percentages. At the outer levels some fully selected clusters can be seen, but more interestingly, looking at the inner bands, there are strong differences in color, indicating some relationship between the distributions of the residuals for the overall CPI, and the distribution of clusters of the other CPIs. This indicates some form of inter-dependence between them should be investigated to improve the model.

Key Applications

Hierarchies are a very common data structure. Application areas include geographical data, which are invariably organized in hierarchies, data on organizations, such as businesses, military organizations, and political entities. Financial data is also often suitable. As well as *a priori* hierarchies, it is very common to generate hierarchies so as to aggregate data and allow higher level information to be viewed. Database systems like OLAP and other roll-up techniques can create hierarchies and allow users to move rapidly between levels to investigate data.

Data Sets

UK figures for CPI were retrieved from http://www.statistics.gov.uk/cpi/, and similar statistics should be generally available from most countries National Statistics office.

The world population data are a subset taken from the CIA's world fact book, available at https://www.cia.gov/library/publications/the-world-factbook/.

Cross-references

- ► OLAP
- ► Visualizing Network Data

Recommended Reading

1. Di Battista G., Eades P., Tamassia R., and Tollis I. Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall, Englewood Cliffs, NJ, USA, 1999.
2. Friendly M. The Gallery of Data Visualization. http://www.math.yorku.ca/SCS/Gallery/.

Visualizing Network Data

Graham Wills
SPSS Inc., Chicago, IL, USA

Synonyms

Graph layout; Network topology; Graph drawing; Information visualization on networks

Definition

A network is a set of nodes with edges connecting the nodes. When the graph defined by that set of nodes and edges has other associated data, the result is termed "network data." Visualizing Network Data is the process of presenting a visual form of that structure so as allow insight and understanding.

Historical Background

Although examples of informal drawing of networks and hand-drawn graph layouts can be found stretching back many decades, the discipline of visualizing network data in a systematic fashion dates back only to the availability of computers, with an early paper by Tutte in 1963 titled "How to Draw a Graph" being a prime example. In 1992, the conference *International Symposium on Graph Drawing* commenced meeting and their conference proceedings, available from 1994, provide an excellent overall reference to the state of the art in this subject. From around 1990 onwards, increased attention has been paid to *interactive*, or *dynamic* visualization of Network Data. These techniques have surfaced in a variety of different fields, including Statistical Graphics, Information Visualization, and many applied areas.

Foundations

The basic goal behind a successful static visualization of network data is simply stated: Producing a layout of nodes and edges with a bounded 2-dimensional (or, less commonly, 3-dimensional) region such that the resulting display portrays the structure of the network as clearly as possible. To achieve this goal the following topics must be addressed:

- Techniques for representing the nodes and edges
- Aesthetic criteria that define what is meant by a clear representation
- Layout Techniques
- Extensions to layout techniques to incorporate data on nodes and edges
- Interactive techniques to augment static layouts

Node and Edge Representation

The simplest form of display for a node is as a small glyph, commonly a circle or a square. It is simple and compact. Further, it is easy to add extra information to, such as color, pattern, label or other aesthetics. These can represent data on the nodes. The first three example layouts Fig. 1 below show nodes as circles. The last layout is different; it uses an extended representation to display a node, allowing the edges to be displayed as vertical lines. This representation is most common when the graph is *tree-like*, or more generally is a *hierarchical* network. When there is special known structure for a graph like this, it is possible to use representations that help elucidate such properties. For general network data, however, simple nodes are the most common and suitable choice.

Edge representation allows more freedom. If the edges are *directed*, so that they have a definite "from" and "to" node, then they are usually portrayed with arrows at the ends, unless the direction is obvious from the layout. Other common representational techniques are:

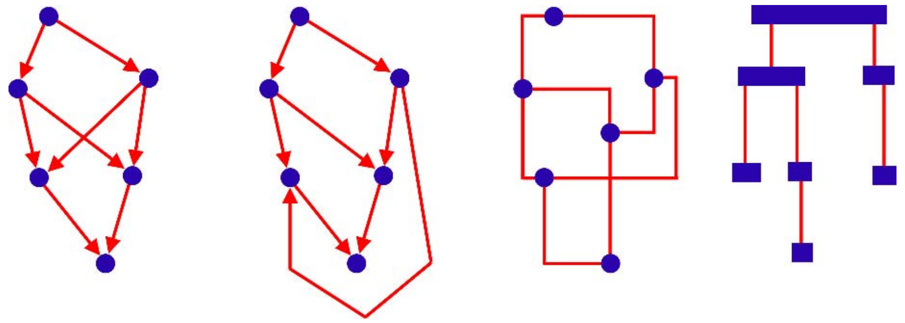
- *Straight Edge*. Figure 1(a) and Figure 1(d) shows edges as straight lines directly linking nodes. This has the advantage of being simple and making connections clear, but tends to result in more edge crossings.
- *Polyline/Paths*. Using paths instead of straight edges allows a reduction in the number of edge crossings in Fig. 1(c), but the resulting display is more complex.
- *Orthogonal Paths*. A style of representation where paths consist of orthogonal lines. This form of representation harkens back to printed circuit board layouts, an early application of graph layout.

Quality Criteria

The question of what makes a good layout has been approached by many authorities. In practice trade-offs must be made, and specific applications stress some criteria more than others. Those criteria that are most often considered important are given below:

- Minimize edge crossings
- Minimize the area needed to display
- Maximize the symmetry of the layout, both globally and locally
- Minimize number of bends in polyline layouts
- Maximize the angle between edges, both at nodes and when they cross
- Minimize total path lengths

There are also criteria that are suitable for specific graph types. For example, in a graph that is *directed* and *acyclic* (there is no path from nodes following edges that loops back on itself), one strong criterion is that the edges generally head in the same direction (all upward, or all downward, as in Fig. 1(a). Note that Fig. 1(b) does not exhibit this quality, an example of the trade-offs made when considering different layout algorithms.



Visualizing Network Data. Figure 1. Representations: (a) Straight-edge, (b) Polyline, (c) Orthogonal, (d) Hierarchical.

Layout Techniques

Layout techniques for general networks employ a variety of techniques, the most common of which are:

Planar Embeddings. A planar graph is one that has a 2-dimensional representation that has no edge crossings. It is also possible to represent any planar graph using only straight-line edges. Although testing and subsequent layout can be done in linear time, the algorithms for doing so are complex. When each vertex has at most four edges, orthogonal representations are possible, but minimizing the number of bends is *NP-hard*, and approximate algorithms are employed in practice.

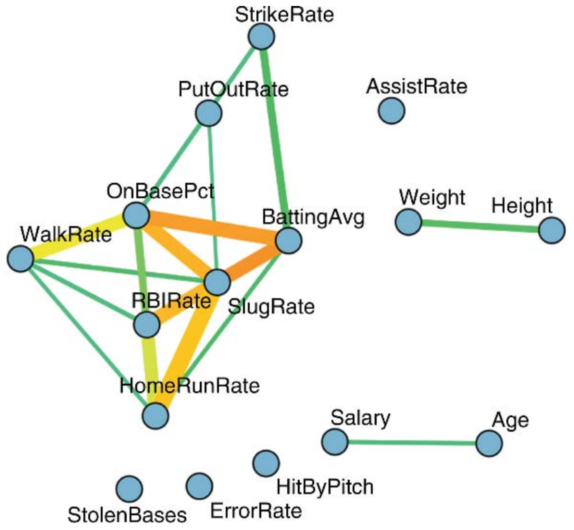
Planarization. If a graph is not planar, it can be made planar by adding "fake" nodes at the crossing points. One such technique would be to find the maximal planar subgraph and laying it out, then adding the additional edges and inserting the additional nodes at crossing points. The resulting graph is laid out using straight edges and then the fake nodes removed, leaving polylines connecting some remaining nodes.

Directed Embeddings. A similar technique is to extract a directed graph from the overall graph (by orienting the edges if necessary) and use a hierarchical drawing technique to place the nodes. A simple example would be finding a minimum spanning tree within the graph and laying it out directly.

Force-directed. Using a mixture of aesthetic criteria, the "energy" of a layout can be defined where low energy corresponds to a good layout. The resulting energy function can be minimized using techniques including randomization, simulated annealing, steepest descent, and simulation. Force-directed techniques are very general, and often can be implemented using iterative algorithms, which makes them amenable to a choice of stopping criteria that can deal with large networks more easily.

Layouts for Networks with Data

If, in addition to the network connections, one also has data on either or both of the nodes or the edges, standard *Information Visualization* techniques can be used to augment the network visualizations. The simplest augmentation is to map a variable for nodes or edges onto an aesthetic, such as color, size, shape, pattern, transparency or dashing. The basic act of labeling nodes is already an example of this use of aesthetics to convey information. Figure 2 shows an application to understanding correlation patterns between variables in a data set consisting of information on Major League Baseball players in 2004. Only correlations passing a statistical test of adequacy have been retained, resulting in a non-connected graph. The edges contain data on two measures of association between then variables connected by the edge, which have been mapped to aesthetics as follows:



Visualizing Network Data. Figure 2. Correlations between Variables; baseball player data, 2004.

- *Color*: The color represents the statistical significance of the association, with green being weak and red being strong.
- *Size*: The width of the edges indicates how strong the association is in the sense of how much of the variation with one variable can be explained by the other.

The overall layout has a straight-edge representation, and has been arrived at via a force-directed algorithm.

When the data are similar to the above, where one has a measure (or, in this case, two measures) of an edge's strength, the algorithms used should be modified so as to take the strength into account. Although this is possible for all algorithms, it is most simple to implement using force-directed techniques. The goal is to ensure that nodes that are highly associated with each other are close together, and that leads to the criterion that path length should be inversely proportional to edge weight. Reviewing the quality criteria above, some of the criteria can be modified for weighted networks as follows:

- Minimize the weighted summed deviation between path lengths and inverse edge weights.

- Penalize edge crossings with crossings involving strong edges penalized more than weak edges.
- Maximize the angle between edges, both at nodes and when they cross, penalizing angles involving strong edges more than weak edges.

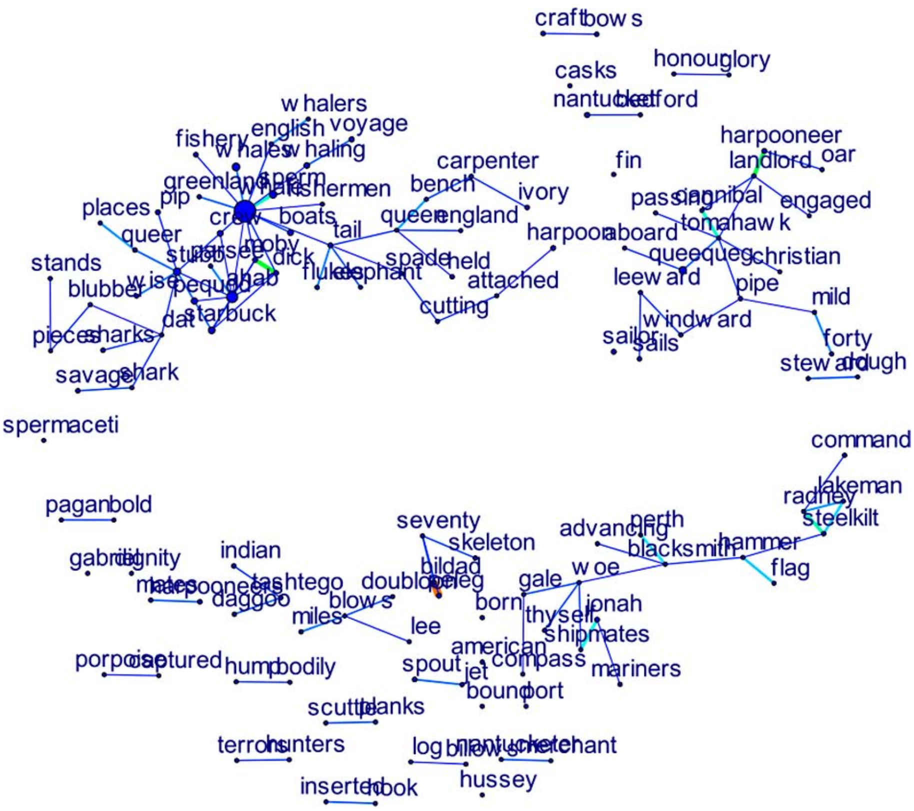
The layout of Fig. 2 was achieved by a force-directed algorithm under these constraints. The measures of association were derived by *Scagnostic* algorithms of Wilkinson and Wills.

Interactive Augmentations

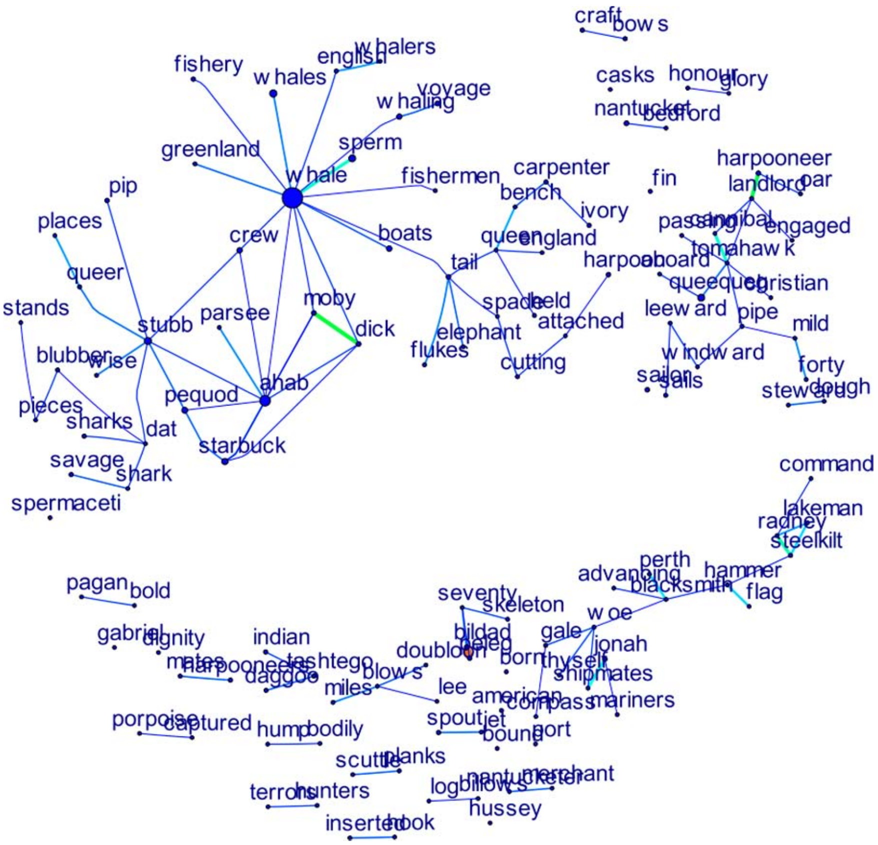
For small and medium-sized static networks, static layouts are adequate, but for larger data sets and for time-varying data, different techniques are required. These can roughly be divided into the following categories:

Distortion techniques. Suitable for large but not huge networks, distortion techniques allow users to place a focus point on a region of interest of the display, and the display redraws so as to magnify the area of interest and de-magnify the rest of the display. Since usable magnification levels can go no higher than a factor of about 5, this technique can improve the number of nodes that can be *visualized* by a maximum of about 25. Specific versions of these techniques include *fisheye* and *hyperbolic* transformations. Figure 3 and Figure 4 below illustrate a network showing associations between words in Melville's *Moby Dick*. The size of the nodes indicates word frequencies, and the links color indicates strength of association. The cluster at the top left is cluttered in Fig. 3, so an interactive fisheye is applied and the focus point is dragged to that cluster, allowing one to see that cluster in more detail, as shown in Fig. 4.

Brushing/Linking. Brushing and Linking techniques are generally applicable techniques for using one visualization in conjunction with another. In the basic implementation a binary pseudo-variable is added to the data set indicating the user's degree of interest. This variable is mapped to an aesthetic in each linked chart, and the user is allowed to drag a brush, or click on



Visualizing Network Data. Figure 3. Force-directed layout of important words in *Moby Dick*. Node sizes indicate word frequencies. Edges link words are commonly found close to each other.



Visualizing Network Data. Figure 4. Force-directed layout of important words in *Moby Dick*. A fisheye transformation has been placed over the dense cluster around the word "whale," magnifying that cluster.

areas of interest in one chart so as to set corresponding values of the "degree of interest" variable and highlight corresponding parts of all linked charts.

This technique is of particular value in the visualization of network data, as it allows any graph layout to be augmented with additional visualizations of data on nodes and links. A simple system for reducing visual complexity is to link histograms, bar charts or other summarized charts to a graph layout display, with the visibility of nodes and links based on the degree of interest. This allows users to select, for example, high values of one variable and then refine the selection by clicking on a bar of a categorical chart. The network display will then show only those items corresponding to the visually selected subset of rows.

A trivial application of this is simply to provide sliders for each variable associated with nodes and links, and by dragging the sliders define a subset of the data which is then displayed as a network display.

Animation. When data on networks varies over time, animating the results can provide insight into the nature of the variation. Visualization is particularly suitable for such data as modeling dynamically evolving networks is a hard problem, with no general models currently available. Hence visualization becomes an important early step in understanding the problem. Technically, animation is similar to brushing, and can be simulated in a brushing environment by making selections on a view of the time dimension. The main differences are in internal techniques to optimize for animation, and in the user interfaces provided for each.

Key Applications

Network Data Visualization is widely applicable. Communication networks such as telephony, internet and cellular are key areas, with particular emphasis on network security issues such as intrusion detection and fraud monitoring.

Future Directions

The field of network data visualization has been evolving steadily since its inception. The classic problem remains open; providing high-quality layouts for networks. It is known that most problems in this area are NP-hard, and so ongoing research will focus on approximate algorithms. Application areas are increasingly providing large networks with sometimes millions of nodes, which require new algorithms for such data. Further, evolutionary networks, in which the topology of the network itself changes over time, have become more important, and algorithms for evolving a layout smoothly from an old state to accommodate a new state are needed.

Data Sets

The baseball data can be found online at the baseball archive: http://www.baseball1.com. This contains a wealth of tables, and the example data used above was extracted from the tables found there. The text of *Moby Dick* can be found at many sites, including Project Gutenberg: http://www.gutenberg.org.

Cross-references

▶ Visualizing Hierarchical Networks

Recommended Reading

1. Battista G., Eades P., Tamassia R., and Tollis I. Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall, 1999.
2. Herman I., Melancon G., and Marshal M.S. Graph visualization and navigation in information visualization: a survey. IEEE Trans. Vis. Comput. Graph., 6(1), 2000.
3. International Symposium on Graph Drawing, http://graphdrawing.org/

Visualizing Quantitative Data

Anatol Sargin, Ali Ünlü
University of Augsburg, Augsburg, Germany

Synonyms

Visualizing quantitative data; Graphics for continuous data; Visual displays of numerical data

Definition

Quantitative data are data that can be measured on a numerical scale. Examples of such data are length, height, volume, speed, temperature or cost.

A quantitative variable can be transformed into a categorical variable by grouping, for example weight can be divided into underweight, normal weight and overweight. The inverse transformation may not be possible.

Quantitative data can be categorical or continuous. This entry only concentrates on continuous data. Visualization in general means the graphical or visual display of data or relations.

Key Points

Univariate graphics are the most basic ones and display exactly one variable. There are three plot types that are commonly used: dotplots, boxplots and histograms.

Dotplots draw every data point along one axis. This graphic shows clusters and gaps in the data. Overplotting can be a serious problem, because many points can be crowded around one position. Jittering of the points is then a possible solution. This means, that the points are diffused along a second axis.

Boxplots are also aligned on one axis, but show summary statistics (median, lower and upper hinges) and emphasize outliers. A boxplot gives a broad overview of the structure of the data and sets of boxplots are good for comparing several variables.

Histograms are displayed using two axes. The values of the variable, grouped in bins, are on the abscissa and the frequencies of the values on the ordinate. Histograms can be very informative but strongly depend on the choice of the anchorpoint (start of the first bin) and binwidth.

Bivariate data are typically drawn as scatterplots, which show the structure and dependencies of two variables. The two variables are plotted against each other in an orthogonal coordinate system. The points can be drawn in different colors or symbols to show special groups of the data. Scatterplots are good for detecting one- and two-dimensional outliers and can, together with interactive methods, also be used for large datasets.

Scatterplot matrices, also known as splom, are a generalization for n variables. The (i,j)-th entry is the scatterplot of variable i against variable j. Variable labels are often given in the matrix diagonal. Scatterplot matrices are only efficient for a small number of variables. For more variables parallel coordinates are a better choice. This space-saving graphic shows all variables in one display and hence can visualize the whole dataset.

Cross-references

- ▶ Data Visualization
- ▶ Multivariate Visualization Methods
- ▶ Parallel Coordinates
- ▶ Visualizing Categorical Data

Recommended Reading

1. Cleveland W. Visualizing Data. Hobert, Summit, NJ, USA, 1993.
2. Unwin A., Theus M., and Hofmann H. Graphics of Large Datasets. Springer, Berlin Heidelberg New York, 2006.

Visualizing Spatial Data

▶ Visual Interfaces for Geographic Data

Visualizing Trees

▶ Visualizing Hierarchical Data

Volume

Kaladhar Voruganti
Network Appliance, Sunnyvale, CA, USA

Synonyms

Logical volume; Physical volume

Definition

The storage provided by a storage device is offered in the form of volumes. A volume corresponds to a logically contiguous piece of storage. The volumes offered by storage devices are typically known as physical volumes.

Key Points

Storage controllers usually create volumes by stripping them across multiple disks that belong to a RAID group. RAID 1 (mirroring), RAID 5 (use of parity, but no dedicated parity disk), and RAID 6 (can tolerate two failures) are the most common RAID types. These volumes get mapped into LUNs by host OS, or aggregated into logical volumes by a logical volume manager, or aggregated into virtual volumes by virtualization software. A file system and its associated files ultimately reside in a physical volume on a storage device. Database table's data also ultimately resides in a physical volume on a storage device. Storage vendors typically use Volume as the unit of management. That is, they allow volume level data copy, migration, snapshots, access control, compression, and encryption.

Cross-references

- ▶ Logical Unit Number (LUN)
- ▶ Logical Volume Manager
- ▶ LUN
- ▶ LUN Mapping
- ▶ RAID

Volume Set Manager

▶ Logical Volume Manager (LVM)

Voronoi Decomposition

▶ Voronoi Diagram

Voronoi Diagrams

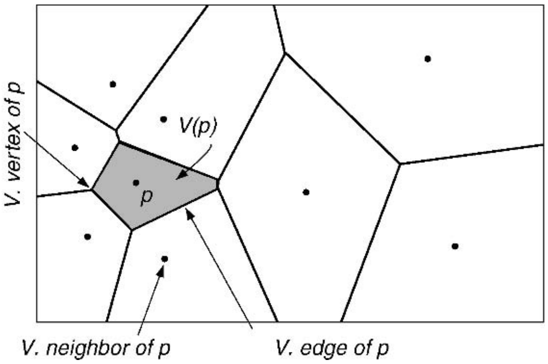
Cyrus Shahabi[1], Mehdi Sharifzadeh[2]
[1]University of Southern California, Los Angeles, CA, USA
[2]Google, Santa Monica, CA, USA

Synonyms

Voronoi tessellation; Voronoi decomposition; Dirichlet tessellation; Thiessen polygons

Definition

The Voronoi diagram of a given set $P = \{p_1,\ldots,p_n\}$ of n points in $\mathbb{R}^d$ partitions the space of $\mathbb{R}^d$ into n regions [2]. Each region includes all points in $\mathbb{R}^d$ with a common closest point in the given set P according to a distance metric $D(.,.)$. That is, the region



Voronoi Diagrams. Figure 1. The ordinary Voronoi diagram.

corresponding to the point $p \in P$ contains all the points $q \in \mathbb{R}^d$ for which the following holds:

$$\forall p' \in P, p' \neq p, \ D(q, p) \leq D(q, p')$$

The equality holds for the points on the borders of p's and p''s regions. Incorporating arbitrary distance metrics $D(.,.)$ results in different variations of Voronoi diagrams. As an example, *Additively Weighted* Voronoi diagrams are defined by using the distance metric $D(p, q) = L_2(p, q) + w(p)$ where $L_2(.,.)$ is the Euclidean distance and $w(p)$ is a numeric weight assigned to p. A thorough discussion on all variations is presented in [7]. Figure 1 shows the *ordinary* Voronoi diagram of nine points in $\mathbb{R}^2$ where the distance metric is Euclidean. The region $V(p)$ containing the point p is denoted as its Voronoi *cell*. With Euclidean distance in $\mathbb{R}^2$, $V(p)$ is a convex polygon. Each edge of this polygon is a segment of the perpendicular bisector of the line segment connecting p to another point of the set P. These edges and their end-points are called *Voronoi edges* and *Voronoi vertices* of the point p, respectively. For each Voronoi edge of the point p, the corresponding point in the set P is called a *Voronoi neighbor* of p. The point p is referred to as the *generator* of Voronoi cell $V(p)$. Finally, the set given by $VD(P) = \{V(p_1),...,V(p_n)\}$ is called the Voronoi diagram of the set P with respect to the distance function $D(.,.)$.

Key Points

Voronoi diagrams exhibit the following properties [9]:

Property 1: The Voronoi diagram of a set P of points, $VD(P)$, is unique.

Property 2: Given the Voronoi diagram of P, the nearest point of P to point $p \in P$ is among the Voronoi neighbors of p. That is, the closest point to p is one of generator points whose Voronoi cells share a Voronoi edge with $V(p)$.

Property 3: The average number of vertices per Voronoi cells of the Voronoi diagram of a set of points in $\mathbb{R}^2$ does not exceed six. That is, the average number of Voronoi neighbors of each point of P is at most six.

Empowered by the above properties, different variations of Voronoi diagrams have been used as index structures for the nearest neighbor search. Hagedoorn introduces a directed acyclic graph based on Voronoi diagrams [3]. He uses the data structure to answer exact nearest-neighbor queries with respect to general distance functions in $O(\log^2 n)$ time using only $O(n)$ space. In [6], Maneewongvatana proposes a hierarchical index structure for point data, termed overlapped split tree (os-tree). Each os-tree node is associated with a convex polygon referred as the *cover*, which includes all the points in space whose nearest neighbor is a data point associated with the node. The same principal used in Voronoi diagrams is utilized in os-trees to partition the space using subset of Voronoi edges into regions, each having different sets of nearest neighbors. In [11], Xu et al. study indexing location data in location-based wireless services. They propose the D-tree, an index structure that can be used to efficiently process NN queries (planar point queries in their terminology). D-tree simply indexes the point data using the subsets of the points' Voronoi edges that partition the entire set into two subsets. This partitioning principle is used in all levels of the tree.

Many studies also focus on utilizing individual Voronoi cells for query processing. Korn and Muthukrishnan [5] describe four examples of the Voronoi cell computation problem, drawn from different spatial/vector space domains in which the influence set of a given point is required. Stanoi et al. in [10] combine the properties of Voronoi cells (influence sets in their terminology) with the efficiency of R-trees to retrieve reverse nearest neighbors of a query point from the database. Zhang et al. [12] determine the so-called *validity region* around a query point as the Voronoi cell of its nearest neighbor. The cell is the region within which the result of the nearest neighbor query remains valid as the location of the query point is changing. To provide an efficient similarity search mechanism in a peer-to-peer data network, Banaei-Kashani and Shahabi propose that each node maintains its Voronoi cell based on its local neighborhood in the content space [1]. As a more practical example, Kolahdouzan and Shahabi [4] propose a Voronoi-based data structure to improve the performance of exact *k* nearest neighbor search in spatial network databases. Sharifzadeh and Shahabi [8] utilize Additively Weighted Voronoi diagrams to process a class of nearest neighbor queries.

Cross-references

▶ Road Networks
▶ Rtree

Recommended Reading

1. Banaei-Kashani F. and Shahabi C. SWAM: a family of access methods for similarity-search in peer-to-peer data networks. In Proc. Int. Conf. on Information and Knowledge Management, 2004, pp. 304–313.
2. de Berg M., van Kreveld M., Overmars M., and Schwarzkopf O. Computational Geometry: Algorithms and Applications, 2nd ed. Springer, Berlin Heidelberg New York, 2000.
3. Hagedoorn M. Nearest neighbors can be found efficiently if the dimension is small relative to the input size. In Proc. 9th Int. Conf. on Database Theory, 2003, pp. 440–454.
4. Kolahdouzan M. and Shahabi C. Voronoi-based K nearest neighbor search for spatial network databases. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 840–851.
5. Korn F. and Muthukrishnan S. Influence sets based on reverse nearest neighbor queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 201–212.
6. Maneewongvatana S. Multi-Dimensional Nearest Neighbor Searching with Low-dimensional Data. PhD thesis, Computer Science Department, University of Maryland, College Park, MD, USA, 2001.
7. Okabe A., Boots B., Sugihara K., and Chiu S.N. Spatial Tessellations, Concepts and Applications of Voronoi Diagrams, 2nd edn. Wiley, Chichester, UK, 2000.
8. Sharifzadeh M. and Shahabi C. Processing optimal sequenced route queries using voronoi diagrams. Geoinformatica 12(4), Springer Netherlands, December 2008, pp. 411–433.
9. Sharifzadeh M. Spatial Query Processing Using Voronoi Diagrams. PhD thesis, Computer Science Department, University of Southern California, Los Angeles, CA, 2007.
10. Stanoi I., Riedewald M., Agrawal D., and El Abbadi A. Discovery of influence sets in frequently updated databases. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 99–108.
11. Xu J., Zheng B., Lee W.-C., and Lee D.L. The D-tree: an index structure for planar point queries in location-based wireless services. IEEE Trans. Knowl. Data Eng., 16(12):1526–1542, 2004.
12. Zhang J., Zhu M., Papadias D., Tao Y., and Lee D.L. Location-based spatial queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 443–453.

Voronoi Tessellation

▶ Voronoi Diagram

VP

▶ Virtual Partitioning

VSM

▶ Vector-Space Model

W

W3C

Serguei Mankovskii
CA Labs, CA, Inc., Thornhill, ON, Canada

Synonyms

World Wide Web consortium

Definition

W3C is an international consortium for development of World Wide Web protocols and guidelines to ensure long-term growth of the Web.

Key Points

W3C was founded in 1994 by the inventor of the World Wide Web Tim Berners-Lee as a vendor-neutral forum for building consensus around Web technologies. The consortium consists of member organization and dedicated staff of technical experts. Membership is open to any organization or individual whose application is reviewed and approved by the W3C. Usually W3C members invest significant resources into the Web technologies.

W3C fulfils its mission by creation of recommendations enjoying status of international standards. In the first 10 years of existence, it produced over eighty W3C recommendations. W3C is responsible for such technologies as HTML, XHTML, XML, XML Schema, CSS, SOAP, WSDL and others. W3C members play a leading role in the development of the recommendations.

W3C initiatives involve international, national, and regional organizations on global scale. Global participation reflects broad adoption of the web technologies. Along with broad adoption comes growth in diversity of software and hardware used on the Web. W3C aims to facilitate hardware and software interoperability to avoid fragmentation of the Web.

W3C operations are jointly administered by the MIT Computer Science and Artificial Intelligence Laboratory, European Research Consortium for Informatics and Mathematics, and Keio University.

Cross-references

- ▶ Web 2.0/3.0
- ▶ Web Services
- ▶ XML
- ▶ XML Schema

Recommended Reading

1. W3C. Available at: http://www.w3.org/

W3C XML Path Language

▶ XPath/XQuery

W3C XML Query Language

▶ XPath/XQuery

W3C XML Schema

▶ XML Schema

WAN Data Replication

Maarten van Steen
VU University, Amsterdam, The Netherlands

Synonyms

Wide-area data replication

Definition

The field of WAN data replication covers the problems and solutions for distributing and replicating data across wide-area networks. Concentrating on databases alone, a wide-area database is defined as a collection of multiple, logically interrelated databases distributed and possibly replicated across sites that are connected through a wide-area network.

The characteristic feature is that data are spread across sites that are separated through wide-area links. Unlike links in local-area networks, the quality of communication through wide-area links is relatively poor. Links are subject to latencies of tens to thousands of milliseconds, there are often severe bandwidth restrictions, and connections between sites are much less reliable.

In principle, WAN data replication also covers the distribution and replication of plain files. These issues are traditionally handled by wide-area distributed file systems such as AFS [11] and NFS [3,12], which are both widely used. These distributed file systems aim at shielding data distribution from applications, i.e., they aim at providing a high degree of distribution transparency. As such, they tackle the same problems that wide-area databases need to solve. However, matters are complicated for databases, because relations *between* and *within* files (i.e., tables) also need to be taken into account.

Historical Background

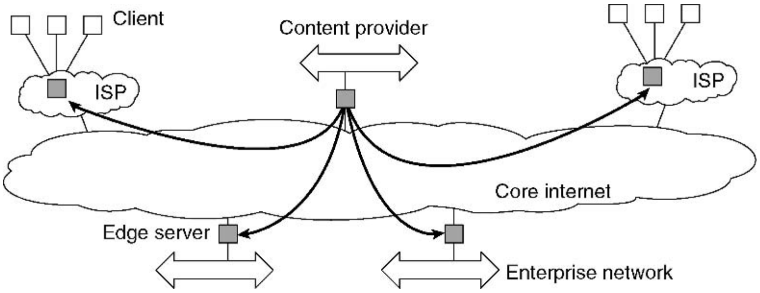
WAN data replication is driven by the need for improving application performance across wide-area networks. Performance is generally expressed in terms of client-perceived quality of service: response times (latency), data transfer rates (bandwidth), and availability. Replication may also be driven by the requirement for reducing monetary costs, as the infrastructure over which data distribution and replication takes place is generally owned by a separate provider who may be charging per transferred byte.

Replication of data has been explored in early wide-area systems such as Grapevine [2], Clearinghouse [4], and Lotus Notes [6]. All these systems concentrated on achieving a balance between data consistency, performance, and availability, recognizing that acceptable quality of service can be achieved only when weak consistency can be tolerated. In general, this required exploring application semantics making solutions more or less specific for an application.

In the mid-1990s, researchers from Xerox explored client-centric consistency models by which data were distributed and replicated in a wide-area database such that a notion of strong data consistency could be presented to users individually [8]. For example, data that had been modified by a user when accessing the database at location *A* would be propagated to location *B* before the user would access the system again at *B*, but not to other locations.

The need for WAN data replication became widely recognized with the explosion of the Web, which instantly revealed the shortcomings of its traditional client-server architecture. Up to date, virtually all Web sites are centrally organized, with end users sending requests to a single server. However, this organization does not suffice for large commercial sites, in which high performance and availability is crucial. To address these needs, so-called *Content Delivery Networks* (CDNs) came into play. A CDN is essentially a Web hosting service with many servers placed across the Internet (see Fig. 1). Its main goal is to ensure that a single Web site is automatically distributed and replicated across these servers in such a way that negotiated performance and availability requirements are met (see also [9]).

CDNs surfaced when most Web sites were still organized as a (possibly very large) collection of files that could be accessed through a single server. Modern sites, however, are no longer statically organized, but deploy full-fledged databases from which Web content



WAN Data Replication. Figure 1. Web-based data replication deploying servers that are placed at the edge of the Internet. Adapted from Tanenbaum, van Steen: Distributed Systems, 2nd edn. Prentice-Hall, Englewood, Cliffs, NJ, 2007.

is dynamically generated by application servers. As a consequence, the so-called edge servers to which client requests are initially directed are gradually turning into servers hosting partially or fully replicated databases, or database caches. These issues are discussed below.

Foundations

A popular model used to understand the various issues involved in wide-area data(base) is the one in which every data item has a single associated server through which all its updates are propagated. Such a primary or origin server as it is called, simplifies the handling of conflicts and maintenance of consistency. In practice, an origin server maintains a complete database that is partially or completely replicated to other servers. In contrast, in an *update anywhere* approach, updates may be initiated and handled at any replica server. The main problem with this approach is that it requires global consensus among the replica servers on the ordering of updates if strong consistency is to be preserved. Achieving such consensus introduces serious scalability problems, for which reason various optimistic approaches have been proposed (optimistic in the sense that corrective actions may later be necessary).

Queries are initially forwarded to edge servers, which then handle further processing. This could mean that subqueries are issued to different origin servers, but it is also possible that the edge server can compute the answer locally and send the response directly to the requesting client without further contacting the origin. In this context, key issues that need to be addressed for wide-area data replication are replica placement and consistency.

Replica Placement

Somewhat surprisingly, many researchers do not make a clear distinction between placement of server machines and placement of data on servers. Nevertheless, this distinction is important: while data placement can often be decided at runtime, this is obviously not the case for placement of server machines. Moreover, the criteria for placement are different: server placement should be done for many data objects, but deciding on the placement of data can be optimized for individual data objects.

Both problems can be roughly tackled as optimization problems in which the best K out of N possible locations need to be selected. There are a number of variations of this problem, but most important is the fact that heuristics need to be employed due to the exponential complexity of known solutions. For this reason, runtime data placement decisions deploy simpler solutions. An overview is provided in [13].

Relevant in this context is comparing different placements to decide which one is best. To this end, a general cost function can be used in various metrics which are combined:

$$\cos t = w_1\,\mathrm{res}_1 + w_2\mathrm{res}_2 + \cdots + w_n\,\mathrm{res}_n,$$
$$\mathrm{res}_k \geq 0, \quad w_k > 0$$

where res_k is a monotonically increasing variable denoting the cost of resource k and w_k its associated weight. Typical resources include distance (expressed in delay or number of hops) and bandwidth. Note that the actual dimension of *cost* is irrelevant. What matters is that different costs can be compared in order to select the best one.

Using a cost-driven placement strategy also implies that resource usage must be measured or estimated. In many cases, estimating costs may be more difficult than one would initially expect. For example, in order for an origin server to estimate the delay between a client and a given edge server may require mapping Internet locations to coordinates in a high-dimensional geometric space [7].

Data Consistency

Consistency of replicated data has received considerable attention, notably in the context of distributed shared-memory parallel computers. This class of computers attempts to mimic the behavior of traditional shared-memory multiprocessors on clusters or grids of computers. However, traditional distributed systems have often generally assumed that only strong consistency is acceptable, i.e., all processes concurrently accessing shared data see the same ordering of updates everywhere. The problem with strong consistency is that it requires timely global synchronization, which may be prohibitively expensive in wide-area networks. Therefore, weaker consistency models often need to be adopted.

To capture different models, Yu and Vahdat defined *continuous consistency* [15], a framework that captures consistency along three different dimensions: time, content, and ordering of operations. When updates are handled in a centralized manner, then notably the first two are relevant for WAN data replication. Continuous time-based consistency expresses to what extent copies of the same data are allowed to be stale with respect to each other. Continuous content-based consistency is used to express to what extent the respective *values* of replicated data may differ, a metric that is useful when dealing with, e.g., financial data. The differences in consistency are subsequently expressed as numbers. By exchanging this information between sites, consistency enforcement protocols can be automatically started without further interference from applications.

There are essentially three ways to bring copies in the same state. First, with *state shipping* the complete up-to-date state is transferred to a replica server. This update form can be optimized through *delta shipping* by which only the difference between a replica's current state and that of a fresher replica is computed and transferred. These two forms of update are also referred to as passive replication, or asymmetric update processing. In contrast, with active replication, *function shipping* takes place, meaning that the operations that led to a new state are forwarded to a replica and subsequently executed. This is also known as *symmetric update processing*.

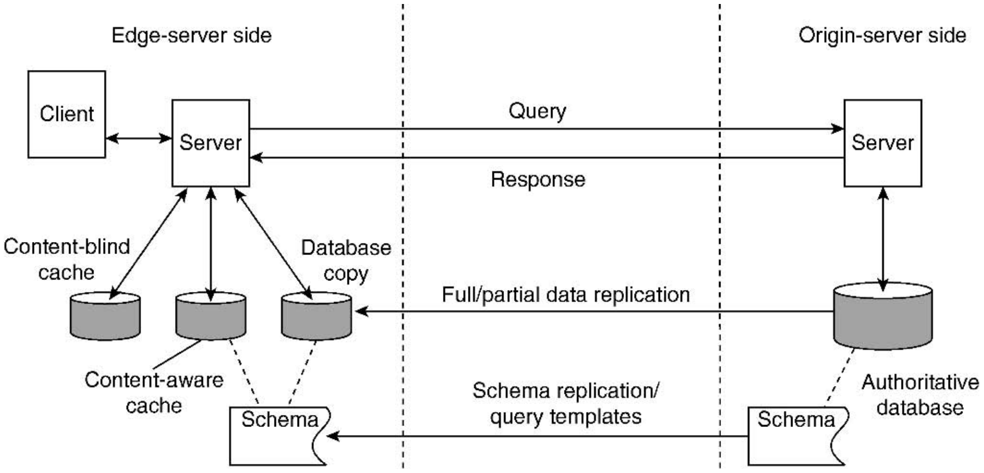
Differences may also exist with respect to the server taking the initiative for being updated. In *pull* approaches a replica server sends a request to a fresher replica to send it updates. In the *push* approach, updates are sent to a replica server at the initiative of a server that has just been updated. Note that for pushing, the server taking the initiative will need to know every other replica server as well as its state. In contrast, pulling in updates requires only that a replica server knows where to fetch fresh state. For these reasons, pull-based consistency enforcement is often used in combination with caches: when a request arrives at a caching server, the latter checks whether its cache is still fresh by contacting an origin server.

An important observation for WAN data replication is that it is impossible to simultaneously provide strong consistency, availability, and partition tolerance [5]. In the edge-server model, this means that despite the fact that an origin server is hosting a database offering the traditional ACID properties, the system as a whole cannot provide a solution that guarantees that clients will be offered continuous and correct service as long as at least one server is up and running. This is an important limitation that is often overlooked.

Key Applications

Wide-area data(base) replication is applied in various settings, but is arguably most prevalent in Web applications and services (see also [1]). To illustrate, consider a Web service deployed within an edge-server infrastructure. The question is what kind of replication schemes can be applied for the edge servers. The various solutions are sketched in Fig. 2.

First, full replication of the origin server's database to the edge servers can take place. In that case, queries can be handled completely at the edge server and problems evolve around keeping the database replica's



WAN Data Replication. Figure 2. Caching and replication schemes for Web applications. Adapted from Tanenbaum, van Steen: Distributed Systems, 2nd edn. Prentice-Hall, Englewood, Cliffs, NJ, 2007.

consistent. Full replication is generally a feasible solution when read/write ratios are high and queries are complex (i.e., spanning multiple database tables). The main problem is that if the number of replicas grow, one would need to resort to lazy replication techniques by which an edge server can continue to handle a query in parallel to bringing all replicas up-to-date, but at the risk of having to reconcile conflicting updates later on [10]. If queries and data can be organized such that access to only a single table is required (effectively leading to only simple queries), partial replication by which only the relevant table is copied to the edge server may form a viable optimization.

An alternative solution is to apply what is known as content-aware caching. In this case, an edge server has part of the database stored locally based on caching techniques. A query is assumed to fit a template allowing the edge server to efficiently cache and lookup query results. For example, a query such as "*select * from items where price < 50*" contains the answer to the more specific query "*select * from items where price < 20*." In this case, the edge server is actually building up its own version of a database, but now with a data schema that is strongly related to the structure of queries.

Finally, an edge server can also deploy content-blind caching by which query results are simply cached and uniquely associated with the query that generated them. In other words, unlike content-aware caching and database replication, no relationship with the structure of the query or data is kept track of.

As discussed in [14], each of these replication schemes has its merits and disadvantages, with no clear winner. In other words, it is not possible to simply provide a single solution that will fit all applications. As a consequence, distributed systems that aim to support WAN data replication will need to incorporate a variety of replication schemes if they are to be of general use.

Cross-references

- ▶ Autonomous Replication
- ▶ Consistency Models for Replicated Data
- ▶ Data Replication
- ▶ Distributed Databases
- ▶ Eventual Consistency Optimistic Replication and Resolution
- ▶ One-Copy Serializability
- ▶ Partial Replication
- ▶ Replica Control
- ▶ Replication Based on Group Communication
- ▶ Replication for High Availability
- ▶ Replication for Scalability
- ▶ Traditional Concurrency Control for Replicated Databases

Recommended Reading

1. Alonso G., Casati F., Kuno H., and Machiraju V. Web Services: Concepts, Architectures and Applications. Springer, Berlin Heidelberg New York, 2004.
2. Birrell A., Levin R., Needham R., and Schroeder M. Grapevine: an excercise in distributed computing. Commun. ACM, 25(4): 260–274, 1982.
3. Callaghan B. NFS Illustrated. Addison-Wesley, Reading, MA, 2000.
4. Demers A., Greene D., Hauser C., Irish W., Larson J., Shenker S., Sturgis H., Swinehart D., and Terry D. Epidemic algorithms for replicated database maintenance. In Proc. ACM SIGACT-SIGOPS 6th Symp. on the Principles of Dist. Comp., 1987, pp. 1–12.
5. Gilbert S. and Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant Web services. ACM SIGACT News, 33(2):51–59, 2002.
6. Kawell L., Beckhardt S., Halvorsen T., Ozzie R., and Greif I. Replicated document management in a group communication system. In Proc. 2nd Conf. Computer-Supported Cooperative Work. 1988, pp. 226–235.
7. Ng E. and Zhang H. Predicting Internet Network Distance with Coordinates-based Approaches. In Proc. 21st Annual Joint Conf. of the IEEE Computer and Communications Societies, 2002, pp. 170–179.
8. Petersen K., Spreitzer M., Terry D., and Theimer M. Bayou: replicated database services for world-wide applications. In Proc. 7th SIGOPS European Workshop, 1996, pp. 275–280.
9. Rabinovich M. and Spastscheck O. Web Caching and Replication. Addison-Wesley, Reading, MA, 2002.
10. Saito Y. and Shapiro M. Optimistic replication. ACM Comput. Surv., 37(1):42–81, 2005.
11. Satyanarayanan M. Distributed files systems. In Distributed Systems, 2nd edn., S. Mullender (ed.). Addison-Wesley, Wokingham, 1993, pp. 353–383.
12. Shepler S., Callaghan B., Robinson D., Thurlow R., Beame C., Eisler M., and Noveck D. Network File System (NFS) Version 4 Protocol. RFC 3530, 2003.
13. Sivasubramanian S., Szymaniak M., Pierre G., and van Steen M. Replication for Web Hosting Systems. ACM Comput. Surv., 36(3):1–44, 2004.
14. Sivasubramanian S., Pierre G., van Steen M., and Alonso G. Analysis of Caching and Replication Strategies for Web Applications. IEEE Internet Comput., 11(1):60–66, 2007.
15. Yu H. and Vahdat A. Design and Evaluation of a Conit-based Continuous Consistency Model for Replicated Services. ACM Trans. Comput. Syst., 20(3):239–282, 2002.

Watermarking

▶ Storage Security

Wavelets on Streams

Minos Garofalakis
Technical University of Crete, Chania, Greece

Definition

Unlike conventional database query-processing engines that require several passes over a static data image, streaming data-analysis algorithms must often rely on building concise, approximate (but highly accurate) *synopses* of the input stream(s) in real-time (i.e., in one pass over the streaming data). Such synopses typically require space that is significantly sublinear in the size of the data and can be used to provide *approximate query answers*.

The collection of the top (i.e., largest) coefficients in the *wavelet transform* (or, *decomposition*) of an input data vector is one example of such a key feature of the stream. *Wavelets* provide a mathematical tool for the hierarchical decomposition of functions, with a long history of successful applications in signal and image processing [10]. Applying the wavelet transform to a (one- or multi-dimensional) data vector and retaining a select small collection of the largest wavelet coefficient gives a very effective form of lossy data compression. Such *wavelet summaries* provide concise, general-purpose summaries of relational data, and can form the foundation for fast and accurate approximate query processing algorithms.

Historical Background

Haar wavelets have recently emerged as an effective, general-purpose data-reduction technique for approximating an underlying data distribution, and providing a foundation for approximate query processing over traditional (static) relational data. Briefly, the idea is to apply the Haar wavelet decomposition to the input relation to obtain a compact summary that comprises a select small collection of *wavelet coefficients*. The results of [3,9,11] have demonstrated that fast and accurate approximate query processing engines can be designed to operate solely over such compact wavelet summaries. Wavelet summaries can also give accurate *histograms* of the underlying data distribution at multiple levels of resolution, thus providing valuable primitives for effective data visualization.

In the setting of static relational tables, the Haar wavelet transform is well understood, and scalable wavelet decomposition algorithms for constructing wavelet summaries have been known for some time [3,11]. Typically, such algorithms assume at least linear space and several passes over the data. Data streaming models introduce the novel challenge of maintaining the topmost wavelet coefficients over a dynamic data distribution (rendered as a stream of updates), while only utilizing *small space and time* (i.e., significantly sublinear in the size of the data distribution).

Foundations

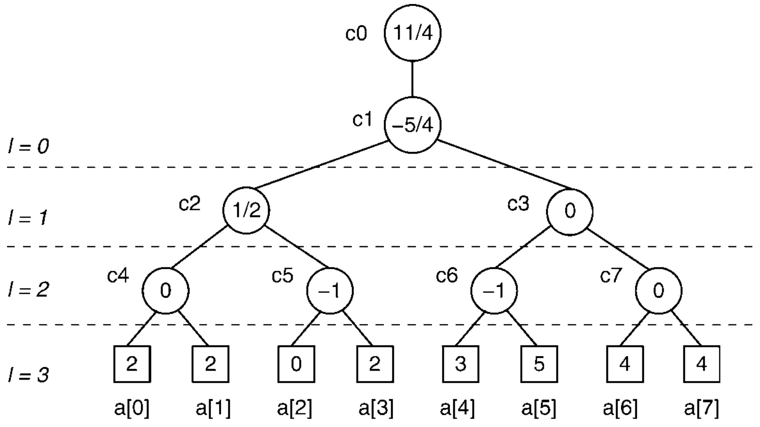
The Haar Wavelet Decomposition. Consider the one-dimensional data vector $a = [2,2,0,2,3,5,4,4]$ comprising $N = 8$ data values. In general, N denotes the data set size and $[N]$ is defined as the integer index domain $\{0,...,N - 1\}$; also, without loss of generality, N is assumed to be a power of 2 (to simplify notation). The Haar Wavelet Transform (HWT) of a is computed as follows. The data values are first averaged together pairwise to get a new "lower-resolution" representation of the data with the pairwise averages $\left[\frac{2+2}{2}, \frac{0+2}{2}, \frac{3+5}{2}, \frac{4+4}{2}\right] = [2,1,4,4]$. This averaging loses some of the information in a. To restore the original a values, *detail coefficients* that capture the missing information are needed. In the HWT, these detail coefficients are the differences of the (second of the) averaged values from the computed pairwise average. Thus, in this simple example, for the first pair of averaged values, the detail coefficient is 0 since $\frac{2-2}{2} = 0$, for the second it is -1 since $\frac{0-2}{2} = -1$. No information is lost in this process – one can reconstruct the eight values of the original data array from the lower-resolution array containing the four averages and the four detail coefficients. This pairwise averaging and differencing process is recursively applied on the lower-resolution array of averages until the overall average is reached, to get the full Haar decomposition. The final HWT of a is given by $w_a = [11/4, -5/4, 1/2, 0, 0, -1, -1, 0]$, that is, the overall average followed by the detail coefficients in order of increasing resolution. Each entry in w_a is called a *wavelet coefficient*. The main advantage of using w_a instead of the original data vector a is that for vectors containing similar values most of the detail coefficients tend to have very small values. Thus, eliminating such small coefficients from the wavelet transform (i.e., treating them as zeros) introduces only small errors when reconstructing the original data, resulting in a very effective form of lossy data compression [10].

A useful conceptual tool for visualizing and understanding the HWT process is the *error tree* structure [9] (shown in Fig. 1 for the example array a). Each internal tree node c_i corresponds to a wavelet coefficient (with the root node c_0 being the overall average), and leaf nodes $a[i]$ correspond to the original data-array entries. This view allows us to see that the reconstruction of any $a[i]$ depends only on the $\log N + 1$ coefficients in the path between the root and $a[i]$; symmetrically, it means a change in $a[i]$ only impacts its $\log N + 1$ ancestors in an easily computable way. The *support* for a coefficient c_i is defined as the contiguous range of data-array that c_i is used to reconstruct (i.e., the range of data/leaf nodes in the subtree rooted at c_i). Note that the supports of all coefficients at resolution level l of the HWT are exactly the 2^l (disjoint) *dyadic ranges* of size $N/2^l = 2^{\log N - l}$ over $[N]$, defined as $R_{l,k} = [k \cdot 2^{\log N - l},\ldots,(k + 1) \cdot 2^{\log N - l} - 1]$ for $k = 0,\ldots,2^l - 1$ (for each resolution level $l = 0,\ldots, \log N$). The HWT can also be conceptualized in terms of vector inner-product computations: let $\phi_{l,k}$ denote the vector with $\phi_{l,k}[i] = 2^{l - \log N}$ for $i \in R_{l,k}$ and 0 otherwise, for $l = 0,\ldots,\log N$ and $k = 0,\ldots,2^l - 1$; then, each of the coefficients in the HWT of a can be expressed as the inner product of a with one of the N distinct Haar *wavelet basis vectors*:

$$\{\frac{1}{2}(\phi_{l+1,2k} - \phi_{l+1,2k+1}) : l = 0,\ldots,\log N - 1; \quad k = 0,\ldots,2^l - 1\} \cup \{\phi_{0,0}\}$$

Intuitively, wavelet coefficients with larger support carry a higher weight in the reconstruction of the original data values. To equalize the importance of all HWT coefficients, a common normalization scheme is to scale the coefficient values at level l (or, equivalently, the basis vectors $\phi_{l,k}$) by a factor of $\sqrt{N/2^l}$. This normalization essentially turns the HWT basis vectors into an *orthonormal basis* – letting c_i^* denote the normalized coefficient values, this fact has two important consequences: (i) The *energy* (squared L_2-norm) of the a vector is preserved in the wavelet domain, that is, $||a||_2^2 = \langle a, a \rangle = \sum_i a[i]^2 = \sum_i (c_i^*)^2$(by Parseval's theorem); and, (ii) Retaining the B largest coefficients in *absolute normalized value* gives the (provably) best B-term approximation in terms of L_2 (or, sum-squared) error in the data reconstruction (for a given budget of coefficients B) [10].

The HWT and its key properties also naturally extend to the case of *multi-dimensional* data distributions; in that case, the input a is a d-dimensional data array, comprising N^d entries (Without loss of generality, a domain of $[N]^d$ is assumed for the d-dimensional case.), and the HWT of a results in a d-dimensional wavelet-coefficient array w_a with N^d coefficient entries. The supports of d-dimensional Haar coefficients are d-dimensional hyper-rectangles (over dyadic ranges in $[N]^d$), and can be naturally arranged in a generalized error-tree structure [5,6].



Wavelets on Streams. Figure 1. Error-tree structure for the example data array a ($N = 8$).

Wavelet Summaries on Streaming Data. Abstractly, the goal is to continuously track a compact summary of the B topmost wavelet coefficient values for a dynamic data distribution vector a rendered as a continuous stream of updates. Algorithms for this problem should satisfy the key *small space/time* requirements for streaming algorithms; more formally, streaming wavelet algorithms should (ideally) guarantee (i) *sublinear space usage* (for storing a synopsis of the stream), (ii) *sublinear per-item update time* (to maintain the synopsis), and (iii) *sublinear query time* (to produce a, possibly approximate, wavelet summary), where "sublinear" typically means polylogarithmic in the domain size N. The streaming wavelet summary construction problem has been examined under two distinct data streaming models.

- *Wavelets in the Ordered Aggregate (Time Series) Streaming Model.* Here, the entries of the input data vector a are rendered over time in the increasing (or, decreasing) order of the index domain values. This means, for instance, that $a[1]$ (or, the set of all updates to $a[1]$) is seen first, followed by $a[2]$, then $a[3]$, and so on. In this case, the set of the B topmost HWT values over a can be maintained *exactly* in small space and time, using a simple algorithm based on the error-tree structure (Fig. 1) [7]: Consider reading (an update to) item $a[i+1]$ in the stream; that is, all items $a[j]$ for $j \leq i$ have already streamed through. The algorithm maintains the following two sets of (partial) coefficients:
  1. Highest B HWT coefficient values for the portion of the data vector seen thus far.
  2. $\log N + 1$ *straddling* partial HWT coefficients, one for each level of the error tree. At level l, index i *straddles* the HWT basis vector $\phi_{l,k}$, where $j \in [k \cdot 2^{\log N - l}, (k+1) \cdot 2^{\log N - l} - 1]$. (Note that there is *at most one* such basis vector per level.)

  When the $(i+1)th$ data item is read, the value for each of the affected straddling coefficients is updated. With the arrival of the $(i+1)th$ item, some coefficients may no longer be straddling (i.e., their computation is now complete). In that case, the value of these coefficients is compared against the the current set of B highest coefficients, and only the B largest coefficient values in the combined set are retained. Also, for levels where a straddling coefficient has been completed, a new straddling coefficient is initiated (with an initial value of 0). In this manner, at every position in the time series stream, the set of the B topmost HWT coefficients is retained exactly.

Theorem 1 *[7] The topmost B HWT coefficients can be maintained exactly in the ordered aggregate (time series) streaming model using O(B + logN) space and O(B + logN) processing time per item.*

Similar algorithmic ideas can also be applied to more sophisticated wavelet thresholding schemes (e.g., that optimize for error metrics other than L_2) to obtain space/time efficient wavelet summarization algorithms in the ordered aggregate model [8].

- *Wavelets in the General Turnstile Streaming Model.* Here, updates to the data vector a can appear in any arbitrary order in the stream, and the final vector entries are obtained by implicitly aggregating the updates for a particular index domain value. More formally, in the turnstile model, each streaming update is a pair of the form $(i, \pm v)$, denoting a net change of $\pm v$ in the $a[i]$ entry; that is, the effect of the update is to set $a[i] \leftarrow a[i] \pm v$. (The model naturally generalizes to multi-dimensional data: for d data dimensions, each update $((i_1,\ldots,i_d), \pm v)$ effects a net change of $\pm v$ on entry $a[i_1,\ldots,i_d]$.) The problem of maintaining an accurate wavelet summary becomes significantly more complex when moving to this much more general streaming model. Gilbert et al. [7] prove a strong *lower bound* on the space requirements of the problem: for arbitrary turnstile streaming vectors, *nearly all* of the data must be stored to recover the top B HWT coefficients.

  Existing solutions for wavelet maintenance over turnstile data streams rely on randomized schemes that return only an *approximate* synopsis comprising (at most) B Haar coefficients that is provably near-optimal (in terms of the captured energy of the underlying vector) assuming that the data vector satisfies the "*small-B property*" (i.e., most of its energy is concentrated in a small number of HWT coefficients) – this assumption is typically satisfied for most real-life data distributions [7]. One of the key ideas is to maintain a randomized *AMS sketch* [2], a broadly applicable stream synopsis structure comprising randomized linear projections of the streaming data vector a. Briefly, an *atomic AMS sketch* of a is simply the *inner product*

$\langle a, \xi\rangle = \sum_i a[i]\xi(i)$, where ξ denotes a random vector of four-wise independent ± 1-valued random variates.

Theorem 2 *[1,2] Consider two (possibly streaming) data vectors a and b, and let Z denote the* $O(\log(1/\delta))$*-wise median of* $O(1/\in^2)$*-wise means of independent copies of the atomic AMS sketch product* $(\sum_i a[i]\xi(i))(\sum_i b[i]\xi_j(i))$. *Then,* $|Z - \langle a, b\rangle| \leq \epsilon||a||_2||b||_2$ *with probability* $\geq 1 - \delta$.

Thus, using AMS sketches comprising only $O(\frac{\log(1/\delta)}{\epsilon^2})$ atomic counters, the vector inner product $\langle a,b\rangle$ can be approximated to within $\pm\ \epsilon||a||_2||b||_2$ (hence implying an ϵ-relative error estimate for the squared L_2 norm $||a||_2^2$).

Since Haar coefficients of *a* are inner products with a fixed set of wavelet-basis vectors, the above theorem forms the key to developing efficient, approximate wavelet maintenance algorithms in the turnstile model. Gilbert et al. [7] propose a solution (termed "GKMS" in the remainder of the discussion) that focuses primarily on the one-dimensional case. GKMS maintains an AMS sketch for the streaming data vector *a*. To produce the approximate *B*-term representation, GKMS employs the constructed sketch of *a* to estimate the inner product of *a* with *all wavelet basis vectors*, essentially performing an exhaustive search over the space of all wavelet coefficients to identify important ones. More formally, assuming that there is a *B*-coefficient approximate representation of the signal with energy at least $\eta||a||_2^2$ ("small *B* property"), the GKMS algorithm uses a maintained AMS sketch to exhaustively estimate each Haar coefficient and selects up to *B* of the largest coefficients (excluding those whose square is less than $\eta\epsilon||a||_2^2/B$, where $\epsilon < 1$ is the desired accuracy guarantee). GKMS also uses techniques based on range-summable random variables constructed using Reed-Muller codes to reduce or amortize the cost of this exhaustive search by allowing the sketches of basis vectors (with potentially large supports) to be computed more quickly.

Theorem 3 *[7] Assuming there exists a B-term representation with energy at least* $\eta||a||_2^2$, *then, with probability at least* $1 - \delta$, *the GKMS algorithm finds a representation of at most B coefficients that captures at least* $(1 - \epsilon)\eta$ *of the signal energy* $||a||_2^2$, *using* $O(\log^2 N\ \log(N/\delta)B^2/(\eta\epsilon)^2)$ *space and per-item processing time.*

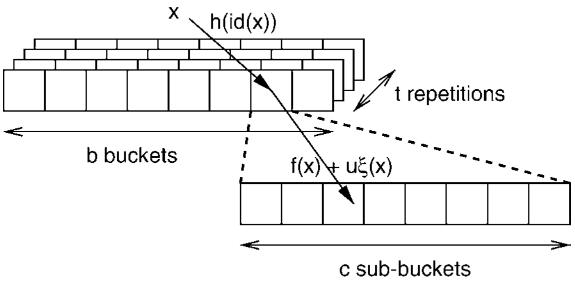
A potential problem lies in the query time requirements of the GKMS algorithm: even with the Reed-Muller code optimizations, the overall query time for discovering the top coefficients remains superlinear in *N* (i.e., at least $\Omega(\frac{1}{\epsilon^2} N \log N)$), violating the third requirement on streaming schemes. This also renders direct extensions of GKMS to multiple dimensions infeasible since it implies an exponential explosion in query cost (requiring at least $O(N^d)$) time to cycle through all coefficients in *d* dimensions). In addition, the update cost of the GKMS algorithm is *linear in the size of the sketch* since the whole data structure must be "touched" for each update. This is problematic for high-speed data streams and/or even moderate sized sketch synopses.

To address these issues, Cormode et al. [5] propose a novel solution that relies on two key technical ideas. First, they work *entirely in the wavelet domain*: instead of sketching the original data entries, their algorithms sketch the wavelet-coefficient vector w_a as updates arrive. This avoids any need for complex range-summable hash functions (i.e., Reed-Muller codes). Second, they employ *hash-based grouping* in conjunction with *efficient binary-search-like techniques* to enable very fast updates as well as identification of important coefficients in polylogarithmic time.

- *Sketching in the Wavelet Domain.* The first technical idea in [5] relies on the observation that it is possible efficiently produce sketch synopses of the stream *directly in the wavelet domain.* That is, the impact of each streaming update can be translated on the relevant wavelet coefficients. By the linearity properties of the HWT and the earlier description, an update to the data entries corresponds to only polylogarithmically many coefficients in the wavelet domain. Thus, on receiving an update to *a*, it can be directly converted to $O(polylog(N))$ updates to the wavelet coefficients, and an approximate (sketch) representation of the wavelet coefficient vector w_a can be maintained.
- *Time-Efficient Updates and Large-Coefficient Searches.* Sketching in the wavelet domain means that, at query time, an approximate representation of the wavelet-coefficient vector w_a is available, and can be employed to identify all those coefficients that are "large," relative to the total energy of the data $||\omega_a||_2^2 = ||a||_2^2$. While AMS sketches can provide such estimates (a point query is just a special case of an inner product), querying remains much

too slow taking at least $\Omega(\frac{1}{\epsilon^2}N)$ time to find which of the N coefficients are the B largest. Instead, the schemes in [5] rely on a *divide-and-conquer* or *binary-search-like* approach for finding the large coefficients. This requires the ability to efficiently estimate sums-of-squares for *groups* of coefficients, corresponding to dyadic subranges of the domain $[N]$. Low-energy regions can then be disregarded, recursing only on high-energy groups – this guarantees no false negatives, as a group that contains a high-energy coefficient will also have high energy as a whole. The algorithms of [5] also employ *randomized, hash-based grouping* of dyadic groups and coefficients to guarantee that each update only touches a small portion of the synopsis, thus guaranteeing very fast update times.

The key to the Cormode et al. solution is a hash-based probabilistic synopsis data structure, termed *Group-Count Sketch (GCS)*, that can estimate the energy of fixed groups of elements from a vector w of size N under the turnstile streaming model [5]. This translates to several streaming L_2-norm estimation problems (one per group). A simple solution would be to keep an AMS sketch of each group separately; however, there can be *many* (e.g., linear in N) groups, implying space requirements that are $O(N)$. Streaming updates should also be processed as quickly as possible. The GCS synopsis requires small, sublinear space and takes sublinear time to process each stream update item; more importantly, a GCS can provide a high-probability estimate of the energy of a group within additive error $\epsilon||\omega||_2^2$ in *sublinear time*. In a nutshell, the GCS synopsis first partitions items of w into their group using an `id()` function (which, in the case of Haar coefficients, is trivial since it corresponds to fixed dyadic ranges over $[N]$), and then randomly maps groups to buckets using a hash function $h()$. Within each bucket, a second stage of hashing of items to sub-buckets is applied (using another hash function $f()$, where each contains an atomic AMS sketch counter in order to estimate the L_2 norm of the elements in the bucket. As with most randomized estimation schemes, a GCS synopsis comprises t independent instantiations of this basic randomized structure, each with independently chosen hash function pairs $(h(), f())$ and ξ families for the AMS estimator; during maintenance, a streaming update (x, u) is used to update each of the t AMS counters corresponding to element x. (A pictorial representation is shown in Fig. 2.) To estimate the energy of a group g, for each independent instantiation $m = 1,\ldots,t$ of the bucketing structure, the squared values of all the AMS counters in the sub-buckets corresponding to bucket $h_m(g)$ are summed, and then the *median* of these t values is returned as the estimate.



Wavelets on Streams. Figure 2. The GCS data structure: Element x is hashed (t times) to a bucket of groups (using $h(\mathtt{id}(x))$) and then a sub-bucket within the bucket (using $f(x)$), where an AMS counter is updated.

Theorem 4 *[5] The GCS can estimate the energy of item groups of the vector w within additive error* $\epsilon||\omega||_2^2$ *with probability* $\geq 1 - \delta$ *using space of* $O\left(\frac{1}{\epsilon^3}\log\frac{1}{\delta}\right)$ *counters, per-item update time of* $O\left(\log\frac{1}{\delta}\right)$, *and query time of* $O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$.

To recover coefficients with large energy in the w vector, the algorithm employs a *hierarchical search-tree structure* on top of $[N]$: Each level in this tree structure induces a certain partitioning of elements into groups (corresponding to the nodes at that level), and per-level GCS synopses can be used to efficiently recover the high-energy groups at each level (and, thus, quickly zero in on high-energy Haar coefficients). Using these ideas, Cormode et al. [5] demonstrate that the accuracy guarantees of Theorem 3 can be obtained using $O(\frac{B^3\log N}{\epsilon^3\eta^3}\cdot\log\frac{B\log N}{\epsilon\eta\delta})$ space, $O(\log^2 N\cdot\log\frac{B\log N}{\epsilon\eta\delta})$ per item processing time, *and* $O(\frac{B^3}{\epsilon^3\eta^3}\cdot\log N\cdot\log\frac{B\log N}{\epsilon\eta\delta})$ query time. In other words, their GCS-based solution guarantees sublinear space *and* query time, as well as per-item processing times that are sublinear *in the size of the stream synopsis.* Their results also naturally extend to the multi-dimensional case [5].

Key Applications

Wavelet-based summaries are a general-purpose data-reduction tool, and the maintenance of such summaries over continuous data streams has several important applications, including large-scale IP network monitoring and network-event tracking (e.g., for detecting network traffic anomalies or Denial-of-Service attacks), approximate query processing over warehouse update streams, clickstream and transaction-log monitoring in large web-server farms, and satelite- or sensornet-based environmental monitoring.

Data Sets

Several publicly-available real-life data collections have been used in the experimental study of streaming wavelet summaries; examples include the US Census Bureau data sets (http://www.census.gov/), the UCI KDD Archive (http://kdd.ics.uci.edu/), and the UW Earth Climate and Weather Data Archive (http://www-k12.atmos.washington.edu/k12/grayskies/).

Future Directions

The area of streaming wavelet-based summaries is rich with interesting algorithmic questions. The bulk of the discussion here focuses on L_2-error synopses. The problem of designing efficient streaming methods for maintaining wavelet summaries that optimize for *non-L_2 error metrics* (e.g., general L_p error) under a turnstile streaming model remains open. Optimizing for non-L_2 error implies more sophisticated coefficient thresholding schemes based on dynamic programming over the error-tree structure (e.g., [6,8]); while such methods can be efficiently implemented over ordered aggregate streams [8], no space/time efficient solutions are known for turnstile streams. Dealing with *physically-distributed streams* also raises interesting issues for wavelet maintenance, such as trading off approximation quality with *communication* (in addition to time/space). The distributed AMS sketching algorithms of Cormode and Garofalakis [4] can be applied to maintain an approximate wavelet representation over distributed turnstile streams, but several issues (e.g., appropriate prediction models for local sites) remain open. Finally, from a systems perspective, the problem of incorporating wavelet summaries in industrial-strength data-streaming engines, and testing their viability in real-life scenarios remains open.

Cross-references

- ▶ Approximate Query Processing
- ▶ Data Compression in Sensor Networks
- ▶ Data Reduction
- ▶ Data Sketch/Synopsis
- ▶ Synopsis Structure
- ▶ Wavelets in Database Systems

Recommended Reading

1. Alon N., Gibbons P.B., Matias Y., and Szegedy M. Tracking join and self-join sizes in limited storage. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 10–20.
2. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. In Proc. 28th Annual ACM Symp. on Theory of Computing, 1996, pp. 20–29.
3. Chakrabarti K., Garofalakis M., Rastogi R., and Shim K. Approximate query processing using wavelets. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 111–122.
4. Cormode G. and Garofalakis M. Approximate continuous querying over distributed streams. ACM Trans. Database Syst., 33(2): 1–39, 2008.
5. Cormode G., Garofalakis M., and Sacharidis D. Fast approximate wavelet tracking on streams. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 4–22.
6. Garofalakis M. and Kumar A. Wavelet synopses for general error metrics. ACM Trans. Database Syst., 30(4): 888–928, December 2005.
7. Gilbert A.C., Kotidis Y., Muthukrishnan S., and Strauss M.J. One-pass wavelet decomposition of data streams. IEEE Trans. Knowl. Data Eng., 15(3):541–554, May 2003.
8. Guha S. and Harb B. Wavelet synopsis for data streams: minimizing non-euclidean error. In Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2005, pp. 88–97.
9. Matias Y., Vitter J.S., and Wang M. Wavelet-based histograms for selectivity estimation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 448–459.
10. Stollnitz E.J., DeRose T.D., and Salesin D.H. Wavelets for Computer Graphics – Theory and Applications. Morgan Kaufmann, San Francisco, CA, 1996.
11. Vitter J.S. and Wang M. Approximate computation of multidimensional aggregates of sparse data using wavelets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 193–204.

Weak Consistency Models for Replicated Data

Alan Fekete
University of Sydney, Sydney, NSW, Australia

Synonyms

Weak memory consistency; Copy divergence

Definition

Some designs for a distributed database system involve having several copies or replicas for a data item, at different sites, with algorithms that do not update these replicas in unison. In such a system, clients may detect a discrepancy between the copies. Each particular weak consistency model describes which discrepancies may be seen. If a system provides a weak consistency model, then the clients will require more careful programming than otherwise. Eventual consistency (q.v.) is the best-known weak consistency model.

Historical Background

In the late 1980s and early 1990s, replication research focused on systems that allowed replicas to diverge from one another in controlled ways. Epidemic or multi-master algorithms were introduced in the work of Demers et al. [4]. These researchers identified the importance of session properties [8], which ensure that clients see information that includes changes they could reasonably expect the system to know. Ladin et al. [6] extended this concept by allowing the clients to explicitly indicate information they expect the system to know when responding to requests. Consistency models for single-master designs with replicas that lag behind a master were also explored by several groups [1,7,9]. Much of the work was presented in two Workshops on the Management of Replicated Data, held in 1990 and 1992.

Much later, Bernstein et al. [3] defined a consistency model called relaxed currency, in which stale reads can occur within transactions that also perform updates.

Foundations

In designing a distributed database system, the same data is usually kept replicated at different sites, for faster access and for better availability. However there is generally a substantial performance penalty from trying to keep the replicas perfectly synchronized with one another, so that clients never observe that replicas exist. Thus many system designers have proposed system designs where the replica control (q.v.) allows the clients to see that the replicas have diverged from one another. A consistency model captures the allowed observations that clients will make. In contrast to the models which maintain an illusion of a single-site system, a weaker consistency model allows the clients to see that the system has replicas. Different models are characterized by the ways in which the divergence between replicas is revealed. In describing some of the models, the research community is sometimes divided between definitions that speak directly about the values in the replicas, and definitions that deal with the events that occur at the clients. Another axis of variation in consistency models comes from the possibility of having different consistency requirements for read-only transactions. It is common to require, for example, that all the transactions which modify the data should execute with the strong consistency model of 1-copy-serializability (q.v.), while some weak consistency is allowed for those transactions which contain only read operations. This section does not try to present every known weak consistency model, but rather indicate some characteristic examples of different styles.

To motivate the weak consistency models, consider how badly things can go wrong in a system with uncontrolled read-one-write-all operations, where each update can be done anywhere first (inside the global client transaction) and then the updates are propagated lazily to the other replicas. Suppose there is such a system with one logical data item z which is an initially empty relation R(p,q,r) where p is a primary key, and two sites A and B each with a replica of z. In this example, there will be two different operations that can be performed on the data item: u which inserts a row (INSERT INTO R VALUES (101, 10,"Fred")), and v which modifies some rows (UPDATE R SET q = q +1 WHERE r = "Fred"). Consider $T_{1,C}$ which just performs u and is submitted by client C, and $T_{2,D}$ which performs v and is submitted by client D. Here is a possible sequence of events. In the notation used, $u_1[z^A,1]$ represents the event where u is performed on the local replica of item z at site A, as part of transaction T_1. Since the operations in the following example are modifications, the notation just shows as the return value the number of rows changed (1 in this particular event). Notice that the subscript on the event type indicates the transaction involved, and the superscript on the item name indicates the site of the replica which is affected. The event where the client C running transaction T_1 learns the return value, and thus knows that the operation has been performed, can be represented by $u_{1,C}[z,1]$. Note, there is no superscript on the item, since this is the client's view. Many consistency models need to refer to where transactions start and finish, so there are events like $b_{1,C}$ for the start of transaction T_1 at client C, or $c_{1,C}$ for the commit of that transaction by the client, or indeed

$c_1{}^A$ for the commit of the local subtransaction of T_1 which is running at site A. The symbols $T_{\odot}{}^A$ and $T_{\otimes}{}^B$ are used for the copier transactions that propagate updates previously done at other sites.

$$b_{1,C}\ b_{2,D}\ u_1[z^A,1]\ u_{1,C}[z,1]\ c_1^A\ c_{1,C}\ v_2[z^B,0]\\ v_{2,D}[z,0]\ c_2^B\ c_{2,D}\ b_{\odot}^A\ v_{\odot}[z^A,1]\ c_{\odot}^A\ b_{\otimes}^B\ u_{\otimes}[z^B,1]\ c_{\otimes}^B \qquad (1)$$

Notice that in sequence (1), the operation v acts in a quite different way when performed at the two replicas (at z^B, no rows are found to match the where clause, while at the other replica a row is modified). This leads to a situation where the final state of replica z^A has a row (101, 11, "Fred"), whereas the final state of z^B has (101, 10, "Fred"). That is, the replicas have diverged from one another, and they will stay this way unless another operation is submitted to reset the row in question. What is worse, the effects of this divergence can spread through other operations, which will act quite differently when performed at the two sites. This situation has been called "system delusion" or "split brain." It is clearly unacceptable, and each weak consistency model should at least prevent this occurring.

Convergence. Several replica consistency models ensure that replicas eventually converge. Because this is a *liveness* feature, defining it formally involves looking at infinite sequences of events. However, informally, one can require that, provided the system enters a quiescent period without further changes, then a *convergence* time will be reached when all replicas have the same value, a value that includes the effects of all the updates on the item, performed in some order. Thus, after the convergence point any read on the logical item will show this value. Before the convergence time, a read is allowed to show any value that reflects some arrangement of some subset of the updates to that logical item (and different reads can show incompatible arrangements of different subsets of updates). This eventual consistency (q.v.) model was first proposed in general distributed computing research where operations are independent requests rather than combined into transactions; in replication of transactional DBMSs, there is an additional requirement that there should be an agreed serial order on all the updating transactions, so that the eventual state of every replica is the outcome of doing all the updating transactions in the given serial order.

The key mechanisms for ensuring eventual consistency involve detecting and then reconciling the situations where different replicas have applied operations in different orders. A common form of reconciliating involves having some deterministic rule to say which update should come first. A any replica where operations have been performed in the wrong order, inverse operations are used to rollback to an earlier state, and then the operations are reapplied in the correct order. For example, in the sequence (1) above, if the authoritative order has $T_{1,C}$ before $T_{2,D}$, then when $T_{\otimes}{}^B$ brings information to z^B about the missing operation u, the reconciler will use an inverse operation $v_{\otimes}{}^{-1}[z^B]$ to reverse the out-of-order operation v. This leads to the following sequence, in which the convergence point is reached at the end of $T_{\otimes}{}^B$ when the replicas are in consistent states.

$$b_{1,C}\ b_{2,D}\ u_1[z^A,1]\ u_{1,C}[z,1]\ c_1^A\ c_{1,C}\ v_2[z^B,0]\\ v_{2,D}[z,0]\ c_2^B\ c_{2,D}\ b_{\odot}^A\ v_{\odot}[z^A,1]\ c_{\odot}^A\ b_{\otimes}^B\ v_{\otimes}^{-1}[z^B]\\ u_{\otimes}[z^B,1]\ v_{\otimes}[z^B,1]\ c_{\otimes}^B \qquad (2)$$

In practice, the burden of reconciliation can be reduced somewhat, by noticing that the order does not matter, for many pairs of operations (those that *commute*, for example because they deal with different logical data items). Nevertheless, [5] has shown that the cost of reconciliation is a serious limit to scalability of multi-master designs that offer eventual consistency.

The eventual consistency model is focused on the way update operations are done; until convergence is reached, read-only transactions are quite unconstrained. Variations on the eventual consistency model are defined by limiting, in various ways, the nondeterminism in the values that can be returned in the read operations before the convergence point. For example, some consistency models enforce session properties. One notable property is called "read one's own writes." In this model, the subset of updates whose effects are seen in a read must include all writes (or more generally operations of update transactions) that were submitted at the same client, and further the arrangement of these writes must respect any cases where one transaction completed before another was started. More flexibly, some models, such as in [6] allow each client to indicate, when requesting a read, a set of previous transactions that must be included in the set of operations whose effects lead to the read's return value.

Stale replicas. It is hard to write reconciliation code to deal with all the different cases of out-of-order updates. A major class of system designs avoids this issue entirely, by having a single *master* or primary replica, where all updates of the item are done first (it is common, but not essential, for the master replicas for all logical items to be at a single site). It is easy in system designs like this, to ensure that the updates are propagated one after another from the master replica to the other replicas (called "slaves"). In this model, there is always a "correct" value for each logical data item; this is the value in the master replica. Any slave always has a value which the master held in the past; it reflects some prefix of the transactions whose updates are reflected in the master, and so one says that the slave replica is a (possibly) stale version of the data. The term "quasi-copy" has been applied to a replica with out-of-date information [2]. The weakest, most permissive, consistency model for this system design is "relaxed currency" [3]. In this model, the clients' view is required to be equivalent to that in a serial, unreplicated and multi-version database (Note the change from the definition of 1-copy-serializability (q.v.), where the execution is equivalent to a serial, unreplicated, *single-version* database.). There must be a serial order on all the transactions, so that whenever the client reads a logical item, it sees some version but not necessarily the current one (it may be an older version that is returned).

Most system designs that offer relaxed currency provide a very restricted form: they also require that all the update transactions satisfy 1-copy-serializability (because all the operations of the update transactions are done at the master). In these designs, a read operation inside an update transaction must see the current version of the item; seeing a stale version can happen only in a read-only transaction.

To illustrate the relaxed currency consistency model, here is a sequence of events that might happen in a system with two sites A and B. A is the master site for integer-valued items x and y, while B has a slave copy of y but no replica for x. $T_{3,C}$ executes a transfer of 2 units from y to x by reading and then updating both items at the master site. The read-only transaction $T_{4,C}$ determines the status of the two logical items, first reading x^A (since there is only one copy for x) and then reading the slave replica y^B. $T_\oplus$ denotes the copier transaction that propagates the change in y to the slave at site B. The event where a value 12 is written to the local replica of item x at site A, as part of transaction T_3, is shown as $w_3[x^A, 12]$, and $w_{3,C}[x, 12]$ represents the notification of this write to the client at C. Similarly $r_4[y^B, 20]$ is the event where T_4 reads the value 20 in the replica of y at site B. As before one also has events like $b_{4,C}$ for the start of transaction T_4 at client C, or $c_{4,C}$ for the commit of that transaction by the client, or indeed $c_4{}^A$ for the commit of the local subtransaction of T_4 which is running at site A.

$$\begin{aligned}&b_{3,C}\ b_3^A\ r_3[x^A, 10]\ r_{3,C}[x, 10]\ r_3[y^A, 20]\ r_{3,C}[y, 20]\\&w_3[x^A, 12]\ w_{3,C}[x, 12]\ w_3[y^A, 18]\ w_{3,C}[y, 18]\\&c_3^A\ c_{3,C} b_{4,C}\ b_4^A\ r_4[x^A, 12]\ r_{4,C}[x, 12]\ b_4^B\ r_4[y^B, 20]\\&r_{4,C}[y, 20]\ c_4^A\ c_4^B\ c_{4,C}\ b_\oplus^B\ w_\oplus[y^B, 18]\ c_\oplus^B\end{aligned}\tag{3}$$

At the end of the execution, the slave site is up-to-date, with the latest value for the logical item y it keeps a replica of. However, during the execution, the slave replica is sometimes behind the master, in particular this is true when T_4 reads the replica. Thus in sequence (3), the client view is

$$\begin{aligned}&b_{3,C}\ r_{3,C}[x, 10]\ r_{3,C}[y, 20]\ w_{3,C}[x, 12]\ w_{3,C}[y, 18]\\&c_{3,C}\ b_{4,C}\ r_{4,C}[x, 12]\ r_{4,C}[y, 20]\ c_{4,C}\end{aligned}\tag{4}$$

which could occur in a serial multiversion system with a single site, in the order $T_{3,C}$ then $T_{4,C}$. Notice that in the serial multiversion system, when $T_{4,C}$ reads x it sees the latest version (produced by $T_{3,C}$), but its read of y returns an older "stale" version taken from the initial state.

There has been much work involving enhancing weak consistency definitions with quantitative bounds on the divergence between replicas.

Key Applications

The commercial DBMS vendors all offer replication mechanisms with their products, which can be configured in ways that offer different consistency properties. The most common approaches give either eventual convergence (if multiple masters are allowed, and conflicts are reconciled correctly) or reads from stale replicas (if there is a single master). There are also a range of research prototypes which give the user various consistency models; in general these show a tradeoff, where improved performance comes from offering weaker consistency models.

Future Directions

Most recent research on replica control finds new algorithms that provide one of the known weak consistency models, either eventual consistency or else the combination of 1-copy serializable updates with reads from stale copies. The main focuses of research have been to restrict how stale the reads can be, and how to make reconciliation easier to code correctly.

Cross-references

▶ Data Replication

Recommended Reading

1. Alonso R., Barbará D., and Garcia-Molina H. Data caching issues in an information retrieval system. ACM Trans. Database Syst., 15(3):359–384, 1990.
2. Alonso R., Barbará D., Garcia-Molina H., and Abad S. Quasi-copies: Efficient data sharing for information retrieval systems. In Advances in Database Technology, Proc. 1st Int. Conf. on Extending Database Technology, 1988, pp. 443–468.
3. Bernstein P.A., Fekete A., Guo H., Ramakrishnan R., and Tamma P. Relaxed-currency serializability for middle-tier caching and replication. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 599–610.
4. Demers A.J., Greene D.H., Hauser C., Irish W., Larson J., Shenker S., Sturgis H.E., Swinehart D.C., and Terry D.B. Epidemic algorithms for replicated database maintenance. In Proc. ACM SIGACT-SIGOPS 6th Symp. on the Principles of Dist. Comp. 1987, pp. 1–12.
5. Gray J., Helland P., O'Neil P.E., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
6. Ladin R., Liskov B., Shrira L., and Ghemawat S. Providing high availability using lazy replication. ACM Trans. Comput. Syst., 10(4):360–391, 1992.
7. Sheth A.P. and Rusinkiewicz M. Management of interdependent data: Specifying dependency and consistency requirements. In Proc. 1st Workshop on the Management of Replicated Data, 1990, pp. 133–136.
8. Terry D.B., Demers A.J., Petersen K., Spreitzer M., Theimer M., and Welch B.B. Session guarantees for weakly consistent replicated data. In Proc. 3rd Int. Conf. on Parallel and Distributed Information Systems, 1994, pp. 140–149.
9. Wiederhold G. and Qian X. Consistency control of replicated data in federated databases. In Proc. 1st Workshop on the Management of Replicated Data, 1990, pp. 130–132.

Weak Coupling

▶ Loose Coupling

Weak Equivalence

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Snapshot equivalence; Temporally weak

Definition

Informally, two tuples are *snapshot equivalent* or *weakly equivalent* if all pairs of timeslices with the same time instant parameter of the tuples are identical.

Let temporal relation schema R have n time dimensions, D_i, $i = 1,...,n$, and let τ^i, $i = 1,...,n$ be corresponding timeslice operators, e.g., the valid timeslice and transaction timeslice operators. Then, formally, tuples x and y are weakly equivalent if

$$\forall t_1 \in D_1 ... \forall t_n \in D_n(\tau^n_{t_n}(...(\tau^1_{t_1}(x))...) \\ = \tau^n_{t_n}(...(\tau^1_{t_1}(y))...))$$

Similarly, two relations are snapshot equivalent or weakly equivalent if at every instant their snapshots are equal. Snapshot equivalence, or weak equivalence, is a binary relation that can be applied to tuples and to relations.

Key Points

The notion of weak equivalence captures the information content of a temporal relation in a point-based sense, where the actual timestamps used are not important as long as the same timeslices result. For example, consider the two relations with only a single attribute: {(a, [3, 9]} and {(a, [3, 5]), (a, [6, 9])}. These relations are different, but weakly equivalent.

Both "snapshot equivalent" and "weakly equivalent" are being used in the temporal database community. "Weak equivalence" was originally introduced by Aho et al. in 1979 to relate two algebraic expressions [1,2]. This concept has subsequently been covered in several textbooks. One must rely on the context to disambiguate this usage from the usage specific to temporal databases. The synonym "temporally weak" does not seem intuitive–in what sense are tuples or relations weak?

Cross-references

- ▶ Point-Stamped Temporal Models
- ▶ Snapshot Equivalence
- ▶ Temporal Database
- ▶ Time Instant
- ▶ Timeslice Operator
- ▶ Transaction Time
- ▶ Valid Time

Recommended Reading

1. Aho A.V., Sagiv Y., and Ullman J.D. Efficient optimization of a class of relational expressions. ACM Trans. Database Syst., 4(4):435–454, 1979.
2. Aho A.V., Sagiv Y., and Ullman J.D. Equivalences among relational expressions. SIAM J. Comput., 8(2):218–246, 1979.
3. Gadia S.K. Weak temporal relations. In Proc. 4th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1985, pp. 70–77.
4. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer-Verlag, Berlin, 1998, pp. 367–405.

Weak Memory Consistency

▶ Weak Consistency Models for Replicated Data

Web 2.0 Applications

▶ Social Applications

Web 2.0/3.0

Alex Wun
University of Toronto, Toronto, ON, Canada

Definition

Web 2.0 and Web 3.0 are terms that refer to the trends and characteristics believed to be representative of next generation web applications emerging after the dot-com collapse of 2001. The term Web 2.0 became widely used after the first O'Reilly Media Web 2.0 Conference (http://www.web2con.com/) in 2004.

Key Points

The term Web 2.0 does not imply a fundamental change in Internet technologies but rather, captures what industry experts believe to be the key characteristics of successful web applications following the dot-com collapse. In general, Web 2.0 applications typically leverage one or more of the following concepts:

- Active participation, collaboration, and the wisdom of crowds. Consumers explicitly improve and add value to the web application they are using either through volunteerism or in response to incentives and rewards. This typically involves explicitly creating and modifying content (Wikipedia entries, Amazon reviews, Flickr tags, YouTube videos, etc.)
- Passive participation and network effects: Consumers naturally improve and add value to the web application being used as a result of the application's inherent design and architecture. Only a small percentage of users can be relied on to participate actively, so user contribution can be made non-optional by design. Examples include the ranking of most-clicked search results, most-viewed videos, peer-to-peer file sharing, and viral marketing.
- The Long-tail: Catering to niche user interests in volume and breadth can be just as successful if not more so than narrowly catering to only the most popular interests. This is due to the fact that content distribution over the Internet is not constrained by the physical limitations (such as location, storage, and consumer-base) of real-world vendors and service providers. For example, an online music vendor incurs virtually no additional cost to "stock" songs that do not sell well individually but when taken together, have a large population of fringe consumers.
- The Internet as a platform: Web 2.0 applications become extremely lightweight, flexible, and dynamic when the Internet is treated like a globally available platform with vast resources and an international consumer-base. Such applications act more like software services than traditional software products and evolve quickly by staying perpetually in beta. For Web 2.0 applications, global connectivity also extends beyond traditional PCs to portable handheld devices and other mobile devices. Web applications that mimic traditional desktop applications (such as online Operating Systems, work processors,

and spreadsheets) are examples of Web 2.0 applications using the Internet as a platform.

Web 2.0 enabling technologies primarily build on top of existing Internet technologies in the form of APIs and web development tools – AJAX, MashUp development tools, and other tools for creating rich media content for example.

Web 3.0 is still purely hypothetical and intends to capture the trends and characteristics of web applications beyond Web 2.0. Hypotheses vary widely between industry experts, ranging from the semantic web to available bandwidth as being the key influences of Web 3.0 applications.

Cross-references

- ► AJAX
- ► MashUp
- ► Service

Web Advertising

Vanja Josifovski[1], Andrei Broder[2]
[1]Uppsala University, Uppsala, Sweden
[2]Yahoo! Research, Santa Clara, CA, USA

Synonyms

Online advertising; Contextual advertising; Search advertising

Definition

Web advertising aims to place relevant ads on web pages. As in traditional advertising, most of the advertising on the web can be divided into brand advertising and direct advertising. In the majority of cases, brand advertising is achieved by banners – images or multimedia ads placed on the web page. As opposed to traditional brand advertising, on the web the user can interact with the ad and follow the link to a website of advertisers choice. Direct advertising is mostly in the form of short textual ads on the side of the search engine result pages and other web pages. Finally, the web allows for other types of advertising that are hybrids and cross into other media, such as video advertising, virtual worlds advertising, etc. Web advertising systems are built by implementing information retrieval, machine learning, and statistical techniques in a scalable, low-latency computing platform capable of serving billions of requests a day and selecting from hundreds of millions of individual advertisements.

Historical Background

The Web emerged as a new publishing media in the late 1990. Since then, the growth of web advertising has paralleled the growth in the number of web users and the increased time people spend on the Web. Banner advertising started with simple placement of banners on the top of the pages at targeted sites, and has since evolved into an elaborate placement scheme that targets a particular web user population and takes into account the content of the web pages and sites. Search advertising has its beginnings in the specialized search engines for ad search. Combining the web search with ad search was not something web users accepted from the start, but has become mainstream today. Furthermore, today's search advertising platforms have moved from simply asking the advertiser to provide a list of queries for which the ad is to be shown to employing variety of mechanisms to automatically learn what ad is appropriate for which query. Today's ad platforms are large, scalable and reliable systems running over clusters of machines that employ state-of-the-art information retrieval and machine learning techniques to serve ads at rates of tens of thousands times a second. Overall, the technical complexity of the advertising platforms rivals those of the web search engines.

Foundations

Web advertising spans Web technology, sociology, law, and economics. It has already surpassed some traditional mass media like broadcast radio and it is the economic engine that drives web development. It has become a fundamental part of the web eco-system and touches the way content is created, shared, and disseminated – from static html pages to more dynamic content such as blogs and podcasts, to social media such as discussion boards and tags on shared photographs. This revolution promises to fundamentally change both the media and the advertising businesses over the next few years, altering a $300 billion economic landscape.

As in classic advertising, in terms of goals, web advertising can be split into *brand advertising* whose goal is to create a distinct favorable image for the advertiser's product, and direct-marketing advertising that involves a "direct response": buy, subscribe, vote, donate, etc, now or soon.

In terms of delivery, there are two major types:

1. *Search advertising* refers to the ads displayed alongside the "organic" results on the pages of the Internet search engines. This type of advertising is mostly direct marketing and supports a variety of retailers from large to small, including microretailers that cover specialized niche markets.
2. *Contextual advertising* refers to ads displayed alongside some publisher-produced content, akin to traditional ads displayed in newspapers. It includes both brand advertising and direct marketing. Today, almost all non-transactional web sites rely on revenue from content advertising. This type of advertising supports sites that range from individual bloggers and small community pages, to the web sites of major newspapers. There would have been a lot less to read on the web without this model!

Web advertising is a big business, estimated in 2005 at $12.5 billion spent in the US alone (Internet Advertising Board – www.iab.com). But this is still less than 10% of the total US advertising market. Worldwide, internet advertising is estimated at $18B out of a $300. Thus, even at the estimated 13% annual growth, there is still plenty of room to grow, hence an enormous commercial interest.

From an ad-platform standpoint, both search and content advertising can be viewed as a matching problem: a stream of queries or pages is matched in real time to a supply of ads. A common way of measuring the performance of an ad-platform is based on the clicks on the placed ads. To increase the number of clicks, the ads placed must be relevant to the user's query or the page and their general interests.

There are several data engineering challenges in the design and implementation of such systems.

The first challenge is the volume of data and transactions: Modern search engines deal with tens of billions of pages from hundreds of millions of publishers, and billions of ads from tens of millions of advertisers. Second, the number of transactions is huge: billions of searches and billions of page views per day. Third, to deal with that, there is only a very short processing time available: when a user requests a page or types her query, the expectation is that the page, including the ads, will be shown in real time, allowing for at most a few tens of milliseconds to select the best ads.

To achieve such performance, ad-platforms usually have two components: a *batch processing component* that does the data collection, processing, and analysis, and a *serving component* that serves the ads in real time. Although both of these are related to the problems solved by today's data management systems, in both cases existing systems have been found inadequate for solving the problem and today's ad-platforms require breaking new grounds.

The batch processing component of an ad-system processes collections of multiple terabytes of data. Usually the data are not shared and a typical processing lasts from a few minutes to a few hours over a large cluster of hundreds, even thousands of commodity machines. The concern here is only about recovering from failures during the distributed computation and most of the time data are produced once and read one or more times. To be able to perform the same operation, a commercial database system would need to be deployed on the same scale that is beyond the scope of both shared-nothing and shared disk architectures. The reason for this is that database systems are designed for sharing data among multiple users in presence of updates and have overly complex distribution protocols to scale and perform efficiently at this scale. The backbone of the batch processing components is therefore being built using simpler distributed computation models and distributed file systems running over commodity hardware. Several challenges lay ahead to make these systems more usable and easier to maintain. The first challenge is to define a processing framework and interfaces such that large scale data analysis tasks (e.g., graph traversal and aggregation) and machine learning tasks (e.g., classification and clustering) are easy to express. So far there are two reported attempts to define such a query language for data processing in this environment [6,10]. However, there has been no reported progress on mapping these languages to a calculus and algebra model that will lend itself to optimization to improve optimization time. To make the task easier, new machine learning and data analysis algorithms for large scale data processing are needed. At the data storage layer, the challenge lies in designing a data storage that resides on the same nodes where the processing is performed.

The serving component of an advertising platform must have high throughput and low latency. To achieve this, in most cases the serving component operates over read-only copy of the data replaced occasionally by the batch component. The ads are usually pre-processed and matched to an incoming query or page. The serving component has to implement reasoning that, based on a variety of features of the query/page and the ads, estimates the top few ads that have the maximum expected revenue within the constraints of the marketplace design and business rules associated to that particular advertising opportunity. The first challenge here is developing features and corresponding extraction algorithms appropriate for real-time processing. As the response time is limited, today's architectures rely on serving from a large cluster of machines that hold most of the searched data in-memory. One of the salient points of the design of the ad search is an objective function that captures the necessary trade-off between efficient processing and quality of results, both in terms of relevance and revenue. This function requires probabilistic reasoning in real-time.

Several different techniques can be used to select ads. When the number of ads is small, linear programming can be used to optimize for multiple ads grouped in *slates*. Banner ads are sometimes placed using linear programming as optimization technique [1,2]. In textual advertising the number of ads is too large to use linear programming. One alternative in such case is to use unsupervised methods based on information retrieval ranking [4,11]. Scoring formulas can be adapted to take into account the specificity of ads as documents: short text snippets with distinct multiple sections. Information retrieval scoring can be augmented with supervised ranking mechanisms that use the click logs or editorial judgments [9] to learn what ads work for a particular page/user/query. Here, the system needs to explore the space of possible matches. Some explore-exploit methods have been adapted to minimize the cost of exploration based on one-arm-bandit algorithms [3].

In summary, today's search and content advertising platforms are massive data processing systems that apply complicated data analysis and machine learning techniques to select the best advertising for a given query or page. The sheer scale of the data and the real-time requirements make this problem a very challenging task. Today's implementations have grown quickly and often in an ad-hoc manner to deal with a $15 billion fast growing market, but there is a need for improvement in almost every aspect of these systems as they adapt to even larger amounts of data, traffic, and new business models in web advertising.

Experimental Results

The reader is referred to [4,9,11] for the current published results on contextual advertising. A historical overview of search advertising is provided in [5]. Some results on query rewrites for search advertising are presented in [7,8].

Cross-references

▶ Information Retrieval Models
▶ Information Retrieval Operations

Recommended Reading

1. Abe N. Improvements to the linear programming based scheduling of Web advertisements. J. Electron. Commerce Res., 5(1):75–98, 2005.
2. Abrams Z., Mendelevitch O., and Tomlin J.A. Optimal delivery of sponsored search advertisements subject to budget constraints. In Proc. 8th ACM Conf. on Electronic Commerce, 2007, pp. 272–278.
3. Agarwal D., Chakrabarti D., Josifovski V., and Pandey S. Bandits for taxonomies: a model based approach. In Proc. SIAM International Conference on Data Mining, 2007.
4. Broder A., Fontoura M., Josifovski V., and Riedel L. A semantic approach to contextual advertising. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 559–566.
5. Fain D. and Pedersen J. Sponsored search: a brief history. In Proc. 2nd Workshop on Sponsored Search Auctions. Web publication, 2006.
6. Group C.S. Community systems research at Yahoo! ACM SIGMOD Rec., 36(3):47–54, 2005.
7. Jones R. and Fain D.C. Query word deletion prediction. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 435–436.
8. Jones R., Rey B., Madani O., and Greiner W. Generating query substitutions. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 387–396.
9. Lacerda A., Cristo M., Andre M.G., Fan W., Ziviani N., and Ribeiro-Neto B. Learning to advertise. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 549–556.
10. Pike R., Dorward S., Griesemer R., and Quinlan S. Interpreting the data: parallel analysis with Sawzall. Sci. Program. J., 13(4): 277–298, 2005.
11. Ribeiro-Neto B., Cristo M., Golgher P.B., and de Moura E.S. Impedance coupling in content-targeted advertising. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 496–503.

Web Application Server

▶ Application Server

Web Characteristics and Evolution

Dennis Fetterly
Microsoft Research, Mountain View, CA, USA

Definition

Web characteristics are properties related to collections of documents accessible via the World Wide Web. There are vast numbers of properties that can be characterized. Some examples include the number of words in a document, the length of a document in bytes, the language a document is authored in, the mime-type of a document, properties of the URL that indentifies a document, HTML tags used to author a document, and the hyperlink structure created by the collection of documents.

As in the physical world, the process of change that the web continually undergoes is identified as web evolution. The web is a tremendously dynamic place, with new users, servers, and pages entering and leaving the system continuously, which causes the web to change very rapidly. Web evolution encompasses changes in all web characteristics, as defined above.

Historical Background

As early as 1994, researchers were interested in studying characteristics of the World Wide Web. As documented by Woodruff et al. [13], the Lycos project at the Center for Machine Translation at Carnegie Mellon University calculated and published statistics on the web (HTTP, FTP, and GOPHER) pages it collected. Those initial statistics included the 100 "most weighty words" by TF-IDF, the document size in bytes, and the number of words in each document. The first estimate of the size of the web also occurred in 1994, when Bray seeded a web crawl with 40,000 documents and estimated the size of the web to be the number of unique URLs that could be fetched.

Using web documents collected in 1995, Woodruff et al. performed an extensive study [12] of 2.6 million HTML documents collected using the Inktomi web crawler. The motivations for this study included collecting information about the HTML markup to improve the language and measure the prevalence of non-standard language extensions. The authors performed a natural language analysis, correctness of the HTML markup, top-level domain distribution, in addition to nine per-page properties. The authors of that study also observed that the web evolved "exceptionally quickly" based on their observations of two data sets collected just months apart. They note that the properties of the documents that exist in both collections vary greatly and that many of the documents that exist in the first collection do not exist in the second.

In addition to characterizing documents discovered on the web, a number of usage based studies were performed under the auspices of the World Wide Web Consortium's (W3C) Jim Pitkow was the chair of the final working group, and he published a summary [11] of previous web characterization studies in 1999. These studies included characterizations of web requests, HTTP traffic, web document lifetime, and user behavior. In 2000, Broder et al. conducted a landmark study measuring the graph structure of the web [4] and demonstrating that connectivity of pages on the web could be classified into a structure that resembled a "bowtie."

Foundations

As highlighted by Baeza-Yates et al. [2], one of the principal issues faced by any initiative to characterize the web is how to select the sample of documents to characterize. The importance of this procedure is highlighted in a recent study that compares statistics from four different large web crawls [1] and finds that there are quantitative differences in the statistical characterizations of these collections. Several different methods for sampling have been utilized. The initial studies referred to in the "Historical Background" section were performed when the web was much smaller than it currently is and it was possible to crawl a reasonable fraction of the web in order to obtain a representative sample. However, even though those studies were able to crawl a reasonable fraction of the web, they were still only able to characterize content that they could fetch. A large portion of web content is inaccessible to the observer. This inaccessible content can be stored in a database, which is commonly called the deep web, be part of a corporate intranet, and thus not accessible from the public web, or could require authorization to access.

In order to estimate the size of the web, and the number of web servers providing web content, Lawrence and Giles chose Internet Protocol (IP) addresses at random and used the number of servers that successfully responded to estimate the total number of web servers publically accessible [1]. They also crawled all of the accessible pages on the first 2,500 web servers that responded to requests sent to their IP address.

In order to address the issue of collecting a representative sample of connected, publically accessible web pages, Henzinger et al. describe a method to generate a near-uniform sample of URLs using a biased random walk on the web [10].

Another approach to sampling that has been employed in at least ten cases is to seed the collection with all domains registered in a national domain, for example Chile's national domain is .cl. Baeza-Yates et al. compare characterizations from ten national domains and two multinational web spaces (Africa and Indochina) [2]. They find that web characterizations that include a power law are consistent across all tested collections.

Another area of web characterization that has been the focus of significant study is the detection of duplicate and near-duplicate documents on the web. Detecting exact matches is much simpler than detecting near duplicates because each document can be reduced to a single fingerprint. Measures for computing text similarity, such as cosine similarity or the Jaccard similarity coefficient, require pairwise comparisons between documents, which will not complete when dealing with the numbers of documents that exist on the web. To put this in perspective, in their 1997 paper "Syntatic Clustering of the Web" [3] Broder et al. state that a pairwise comparison would involve $O(10^{15})$ (a quadrillion) comparisons. There have been two general approaches for condensing the information retained for each document and minimizing the number or required comparisons. The Broder et al. approach approximates the Jaccard similarity coefficient by retaining the minimum value computed by each of several hash functions, and then using those retained values, called sketches or shingles, to summarize the document. The computational effort required to discover near-duplicate documents using this algorithm is further reduced via the use of super-shingles, or sketches of sketches. Near-duplicate documents are very likely to share at least one super-shingle, which reduces the complexity to checking each of a small number of super-shingles. Charikar developed the other approach, which uses random projections of the words in a document to approximate the cosine similarity of documents [5]. Henzinger later experimentally compared [9] these two approaches.

Several different approaches have been used to measure the evolution of the World Wide Web. One common approach has utilized traces from web proxies to study evolution of web pages which provides insight into improved polices for caching proxies. Douglis et al. investigated [7] the rate of change of documents on the web in order to validate two critical assumptions that apply to web caching: a significant number of requests are duplicates of past requests, and the content associated with those requests does not change between accesses.

The evolution of the web has been extensively studied from a crawling perspective. In 2000, Cho and Garcia-Molina performed the first study of web page evolution [6]. They downloaded 720,000 pages daily from 270 "popular" hosts and computed a checksum over the entire page contents. These checksums were then compared to measure rates of change. Fetterly, Manasse et al. [8] expanded upon this study in 2003 by performing a breadth-first crawl of 150 million pages each week for 11 weeks and measuring the degree of change that occurred within each page.

At the intersection of web characteristics and evolution, Bar-Yossef et al. expand on the studies of link lifetime referenced by Pitkow [12] and investigate the decay of the web. They introduce the notion of a "soft 404," which is an error page returned with a HTTP result code of 200 – "OK" instead of 404 indicating an error has occurred.

Key Applications

A significant cost incurred when operating a search service is the cost of downloading and indexing content. Being able to predict, as accurately as possible, when to re-crawl a particular page allows the search engine to wisely utilize its resources. A search engine also needs to be able to identify near-duplicate documents that exist within its index, in order to either remove them or to crawl them at a reduced rate.

Characterization of the web enables at least two important functions: interested parties can measure the evolution of the web along many axes, barring significant evolution of the measured characteristic, results of past characterizations can be used to validate whether a new sample is a representative one or not.

Data Sets

The Stanford WebBase Project provives access to their repository of crawled web content. The Laboratory for Web Algorithmics (LAW) at the Dipartimento di Scienze dell'Informazione (DSI) of the Università degli studi di Milano provides several web graph datasets. The Internet Archive is another potential source of data, although at the time of this writing, they do not currently grant new access to researchers.

http://dbpubs.stanford.edu:8091/~testbed/doc2/
Web Base/
http://law.dsi.unimi.it/index.php?option=com_include
&Itemid=65

Cross-references

- ▶ Data Deduplication
- ▶ Document Links and Hyperlinks
- ▶ Incremental Crawling
- ▶ Term Statistics for Structured Text Retrieval
- ▶ Web Search Result De-duplication and Clustering

Recommended Reading

1. Angeles Serrano M., Maguitman A., Santo Fortunato ná M.B., and Vespignani V. Decoding the structure of the www: A comparative analysis of web crawls. ACM Trans. Web, 1(2):10, 2007.
2. Baeza-Yates R., Castillo C., and Efthimiadis E.N. Characterization of national web domains. ACM Trans. Int. Tech., 7(2): 9, 2007.
3. Broder A.Z., Glassman S.C., Manasse M.S., and Zweig G. Syntactic clustering of the web. In Selected papers from the sixth International Conference on World Wide Web, 1997, pp. 1157–1166.
4. Broder A., Kumar R., Maghoul F., Raghavan P., Rajagopalan S., Stata R., Tomkins A., and Wiener J. Graph structure in the web: Experiments and models. In Proc. 8th Int. World Wide Web Conference, 2002.
5. Charikar M.S. Similarity estimation techniques from rounding algorithms. In Proc. 34th Annual ACM Symp. on Theory of Computing, 2002, pp. 380–388.
6. Cho J. and Garcia-Molina H. The evolution of the web and implications for an incremental crawler. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 200–209.
7. Douglis F., Feldmann A., Krishnamurthy B., and Mogul J.C. Rate of change and other metrics: a live study of the world wide web. In Proc. 1st USENIX Symp. on Internet Tech. and Syst., 1997.
8. Fetterly D., Manasse M., Najork M., and Wiener J. A large-scale study of the evolution of web pages. In Proc. 12th Int. World Wide Web Conference, 2003, pp. 669–678.
9. Henzinger M. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 284–291.
10. Henzinger M.R., Heydon A., Mitzenmacher M., and Najork M. On near-uniform url sampling. Comput. Netw., 33(1–6): 294–308, 2000.
11. Lawrence S. and Giles L.C. Accessibility of information on the web. Nature, 400(6740):107–107, July 1999.
12. Pitkow J.E. Summary of www characterizations. Comput. Netw., 30(1–7):551–558, 1998.
13. Woodruff A., Aoki P.M., Brewer E.A., Gauthier P., and Rowe L.A. An investigation of documents from the world wide web. Comput. Netw., 28(7–11):963–980, 1996.

Web Content Extraction

- ▶ Fully Automatic Web Data Extraction

Web Content Mining

- ▶ Data, Text, and Web Mining in Healthcare
- ▶ Data Integration in Web Data Extraction System

Web Crawler

- ▶ Web Crawler Architecture

Web Crawler Architecture

Marc Najork
Microsoft Research, Mountain View, CA, USA

Synonyms

Web crawler; Robot; Spider

Definition

A web crawler is a program that, given one or more seed URLs, downloads the web pages associated with these URLs, extracts any hyperlinks contained in them, and recursively continues to download the web pages identified by these hyperlinks. Web crawlers are an important component of web search engines, where they are used to collect the corpus of web pages indexed by the search engine. Moreover, they are used in many other applications that process large numbers of web pages, such as web data mining, comparison shopping engines, and so on. Despite their conceptual simplicity, implementing high-performance web crawlers poses major engineering challenges due to the scale of the web. In order to crawl a substantial fraction of the "surface web" in a reasonable amount of time, web crawlers must download thousands of pages per second, and are typically distributed over tens or hundreds of computers. Their two main data structures – the "frontier" set of yet-to-be-crawled URLs and the set of discovered URLs – typically do not fit into main memory, so efficient disk-based representations need to be used. Finally, the need to be "polite" to content providers and not to overload any particular web server, and a desire to prioritize the crawl towards high-quality pages and to maintain corpus freshness impose additional engineering challenges.

Historical Background

Web crawlers are almost as old as the web itself. In the spring of 1993, just months after the release of NCSA Mosaic, Matthew Gray [6] wrote the first web crawler, the World Wide Web Wanderer, which was used from 1993 to 1996 to compile statistics about the growth of the web. A year later, David Eichmann [5] wrote the first research paper containing a short description of a web crawler, the RBSE spider. Burner provided the first detailed description of the architecture of a web crawler, namely the original Internet Archive crawler [3]. Brin and Page's seminal paper on the (early) architecture of the Google search engine contained a brief description of the Google crawler, which used a distributed system of page-fetching processes and a central database for coordinating the crawl. Heydon and Najork described Mercator [8,9], a distributed and extensible web crawler that was to become the blueprint for a number of other crawlers. Other distributed crawling systems described in the literature include PolyBot [11], UbiCrawler [1], C-proc [4] and Dominos [7].

Foundations

Conceptually, the algorithm executed by a web crawler is extremely simple: select a URL from a set of candidates, download the associated web pages, extract the URLs (hyperlinks) contained therein, and add those URLs that have not been encountered before to the candidate set. Indeed, it is quite possible to implement a simple functioning web crawler in a few lines of a high-level scripting language such as Perl.

However, building a web-scale web crawler imposes major engineering challenges, all of which are ultimately related to scale. In order to maintain a search engine corpus of say, ten billion web pages, in a reasonable state of freshness, say with pages being refreshed every 4 weeks on average, the crawler must download over 4,000 pages/second. In order to achieve this, the crawler must be distributed over multiple computers, and each crawling machine must pursue multiple downloads in parallel. But if a distributed and highly parallel web crawler were to issue many concurrent requests to a single web server, it would in all likelihood overload and crash that web server. Therefore, web crawlers need to implement *politeness policies* that rate-limit the amount of traffic directed to any particular web server (possibly informed by that server's observed responsiveness). There are many possible politeness policies; one that is particularly easy to implement is to disallow concurrent requests to the same web server; a slightly more sophisticated policy would be to wait for time proportional to the last download time before contacting a given web server again.

In some web crawler designs (e.g., the original Google crawler [2] and PolyBot [11]), the page downloading processes are distributed, while the major data structures – the set of discovered URLs and the set of URLs that have to be downloaded – are maintained by a single machine. This design is conceptually simple, but it does not scale indefinitely; eventually the central data structures become a bottleneck. The alternative is to partition the major data structures over the crawling machines. Ideally, this should be done in such a way as to minimize communication between the crawlers. One way to achieve this is to assign URLs to crawling machines based on their host name. Partitioning URLs by host name means that the crawl scheduling decisions entailed by the politeness policies can be made locally, without any communication with peer nodes. Moreover, since most hyperlinks refer to pages on the same web server, the majority of links extracted from downloaded web pages is tested against and added to local data structures, not communicated to peer crawlers. Mercator and C-proc adopted this design [9,4].

Once a hyperlink has been extracted from a web page, the crawler needs to test whether this URL has been encountered before, in order to avoid adding multiple instances of the same URL to its set of pending URLs. This requires a data structure that supports set membership test, such as a hash table.

Care should be taken that the hash function used is collision-resistant, and that the hash values are large enough (maintaining a set of n URLs requires hash values with $\log_2 n^2$ bits each). If RAM is not an issue, the table can be maintained in memory (and occasionally persisted to disk for fault tolerance); otherwise a disk-based implementation must be used. Implementing fast disk-based set membership tests is extremely hard, due to the physical limitations of hard drives (a single seek operation takes on the order of 10 ms). For a disk-based design that leverages locality properties in the stream of discovered URLs as well as the domain-specific properties of web crawling, see [9]. If the URL space is partitioned according to host names among the web crawlers, the set data structure is partitioned in the same way, with each web crawling machine maintaining only the portion of the set containing its hosts. Consequently, an extracted URL that is not maintained by the crawler that extracted it must be sent to the peer crawler responsible for it.

Once it has been determined that a URL has not been previously discovered, it is added to the *frontier set* containing the URLs that have yet to be downloaded. The frontier set is generally too large to be maintained in main memory (given that the average URL is about 100 characters long and the crawling system might maintain a frontier of ten billion URLs). The frontier could be implemented by a simple disk-based FIFO queue, but such a design would make it hard to enforce the politeness policies, and also to prioritize certain URLs (say URLs referring to fast-changing news web sites) over other URLs. URL prioritization could be achieved by using a priority queue implemented as a heap data structure, but a disk-based heap would be far too expensive, since adding and removing a URL would require multiple seek operations. The Mercator design uses a frontier data structure that has two stages: a front-end that supports prioritization of individual URLs and a back-end that enforces politeness policies; both the front-end and the back-end are composed of a number of parallel FIFO queues [9]. If the URL space is partitioned according to host names among the web crawlers, the frontier data structure is partitioned along the same lines.

In the simplest case, the frontier data structure is just a collection of URLs. However, in many settings it is desirable to attach some attributes to each URL, such as the time when it was discovered, or (in the scenario of continuous crawling) the time of last download and a checksum or sketch of the document. Such historical information makes it easy to determine whether the document has changed in a meaningful way, and to adjust its crawl priority.

In general, URLs should be crawled in such a way as to maximize the utility of the crawled corpus. Factors that influence the utility are the aggregate quality of the pages, the demand for certain pages and topics, and the freshness of the individual pages. All these factors should be considered when deciding on the crawl priority of a page: a high-quality, highly-demanded and fast-changing page (such as the front page of an online newspaper) should be recrawled frequently, while high-quality but slow-changing and fast-changing but low-quality pages should receive a lower priority. The priority of newly discovered pages cannot be based on historical information about the page itself, but it is possible to make educated guesses based on per-site statistics. Page quality is hard to quantify; popular proxies include link-based measures such as PageRank and behavioral measures such as page or site visits (obtained from web beacons or toolbar data).

In addition to these major data structures, most web-scale web crawlers also maintain some auxiliary data structures, such as caches for DNS lookup results. Again, these data structures may be partitioned across the crawling machines.

Key Applications

Web crawlers are a key component of web search engines, where they are used to collect the pages that are to be indexed. Crawlers have many applications beyond general search, for example in web data mining (e.g., Attributor, a service that mines the web for copyright violations, or ShopWiki, a price comparison service).

Future Directions

Commercial search engines are global companies serving a global audience, and as such they maintain data centers around the world. In order to collect the corpora for these geographically distributed data centers, one could crawl the entire web from one data center and then replicate the crawled pages (or the derived data structures) to the other data centers; one could perform independent crawls at each data center and thus serve different indices to different geographies; or one could perform a single geographically-distributed crawl, where crawlers in a given data center crawl web servers that are (topologically) close-by, and then propagate the crawled pages to their peer data centers. The third solution is the most elegant one, but it has not been explored in the research literature, and it is not clear if existing designs for distributed crawlers would scale to a geographically distributed setting.

URL to Code

Heritrix is a distributed, extensible, web-scale crawler written in Java and distributed as open source by the Internet Archive. It can be found at http://crawler.archive.org/

Cross-references

- ▶ Focused Web Crawling
- ▶ Incremental Crawling
- ▶ Indexing the Web
- ▶ Web Harvesting
- ▶ Web Page Quality Metrics

Recommended Reading

1. Boldi P., Codenotti B., Santini M., and Vigna S. UbiCrawler: a scalable fully distributed web crawler. Software Pract. Exper., 34(8):711–726, 2004.
2. Brin S. and Page L. The anatomy of a large-scale hypertextual search engine. In Proc. 7th Int. World Wide Web Conference, 1998, pp. 107–117.
3. Burner M. Crawling towards eternity: building an archive of the World Wide Web. Web Tech. Mag., 2(5):37–40, 1997.
4. Cho J. and Garcia-Molina H. Parallel crawlers. In Proc. 11th Int. World Wide Web Conference, 2002, pp. 124–135.
5. Eichmann D. The RBSE Spider – Balancing effective search against web load. In Proc. 3rd Int. World Wide Web Conference, 1994.
6. Gray M. Internet Growth and Statistics: Credits and background. http://www.mit.edu/people/mkgray/net/background.html
7. Hafri Y. and Djeraba C. High performance crawling system. In Proc. 6th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, 2004, pp. 299–306.
8. Heydon A. and Najork M. Mercator: a scalable, extensible web crawler. World Wide Web, 2(4):219–229, December 1999.
9. Najork M. and Heydon A. High-performance web crawling. Compaq SRC Research Report 173, September 2001.
10. Raghavan S. and Garcia-Molina H. Crawling the hidden web. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 129–138.
11. Shkapenyuk V. and Suel T. Design and Implementation of a high-performance distributed web crawler. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 357–368.

Web Data Extraction

▶ Web Harvesting

Web Data Extraction System

Robert Baumgartner[1,2], Wolfgang Gatterbauer[3], Georg Gottlob[4]
[1]Vienna University of Technology, Vienna, Austria
[2]Lixto Software GmbH, Vienna, Austria
[3]University of Washington, Seattle, WA, USA
[4]Oxford University, Oxford, UK

Synonyms

Web information extraction system; Wrapper generator; Web macros; Web scraper

Definition

A web data extraction system is a software system that automatically and repeatedly extracts data from web pages with changing content and delivers the extracted data to a database or some other application. The task of web data extraction performed by such a system is usually divided into five different functions: (i) *web interaction*, which comprises mainly the navigation to usually pre-determined target web pages containing the desired information; (ii) support for *wrapper generation and execution*, where a wrapper is a program that identifies the desired data on target pages, extracts the data and transforms it into a structured format; (iii) *scheduling*, which allows repeated application of previously generated wrappers to their respective target pages; (iv) *data transformation*, which includes filtering, transforming, refining, and integrating data extracted from one or more sources and structuring the result according to a desired output format (usually XML or relational tables); and (v) *delivering* the resulting structured data to external applications such as database management systems, data warehouses, business software systems, content management systems, decision support systems, RSS publishers, email servers, or SMS servers. Alternatively, the output can be used to generate new web services out of existing and continually changing web sources.

Historical Background

The precursors of web data extraction systems were screen scrapers which are systems for extracting screen formatted data from mainframe applications for terminals such as VT100 or IBM 3270. Another related issue are *ETL* methods (ETL = Extract, Transform, Load), which extract information from various business processes and feed it into a database or a data warehouse. A huge gap was recognized to exist between web information and the qualified, structured data as usually required in corporate information systems or as envisioned by the Semantic Web. In many application areas there has been a need to automatically extract relevant data from HTML sources and translate this data into a structured format, e.g., XML, or into a suitable relational database format.

A first and obvious solution was the evolution from screen scrapers to web scrapers, which can navigate to web pages and extract textual content. However, web scrapers usually lack the logic necessary to define highly structured output data as opposed to merely text chunks or textual snippets. Moreover, they are usually unable to collect and integrate data from different related sources, to define extraction patterns that remain stable in case of minor layout changes, to transform the extracted data into desired output formats, and to deliver the refined data into different kinds of applications. For this reason, specific research on web data extraction was needed and several academic projects and some commercial research projects on web data extraction were initiated before or around the year 2000. While the academic projects such as XWRAP [11], Lixto [2], Wargo [14], and the commercial systems RoboMaker of Kapow technologies (Kapow Technologies. http://www.kapowtech.com/) and WebQL of QL2 Software (QL2 Software Inc. http://www.ql2.com/) focused on methods of strongly supervised "semi-automatic" wrapper generation, providing a wrapper designer with visual and interactive support for declaring extraction and formatting patterns, other projects were based on machine learning techniques. For example, WIEN [9], Stalker [13], and DEByE [10] focused on automatic wrapper induction from annotated examples. Some other projects focused on specific issues such as automatic generation of structured data sets from web pages [3] and particular aspects of navigating and interacting with web pages [1]. Finally, some systems require the wrapper designer to program the wrapper in a high level language (e.g., SQL-like languages for selecting data from a web page) and merely give visual support to the programmer; an example is the W4F system [15]. A number of early academic prototypes gave rise to fully fledged systems that are now commercially available. For example, Stalker gave rise to the Fetch Agent Platform of Fetch Technologies (Fetch Technologies, Inc. http://www.fetch.com/), the Lixto system gave rise to the Lixto Suite of the Lixto Software company (Lixto Software GmbH. http://www.lixto.com/), and the Wargo system evolved into the Denodo Platform [14]. A good and almost complete survey of web information extraction systems up to 2002 is given in [8].

Foundations

Formal foundations and semantics of data extraction

There are four main formal approaches to define (the semantics of) web wrappers:

- In the *functional approach*, a web wrapper is seen as a mapping f from the parse or DOM tree T of a web page to a set $f(T)$ of sub-trees of T, where each subtree corresponds to an extracted data object. Another step specifies the relabeling of extracted data to fit a previously defined output schema.
- The *logical approach* (see Logical foundations of web data extraction) consists of the specification of a finite number of monadic predicates, i.e., predicates, that define sets of parse-tree nodes and that evaluate to true or false on each node of a tree-structured document. The predicates can be defined either by logical formulas, or by logic programs such as monadic datalog, which was shown to be equivalent in expressive power to monadic second-order logic (MSO) [6]. Note that languages such as XPATH or regular path expressions are subsumed by MSO.
- The *automata theoretic approach* relies on tree automata, which are a generalization of finite state automata from words to trees. The equivalence of the two logical approaches, MSO and monadic datalog, and the unranked query automata approach shows that the class of wrappers definable in these formalisms is quite natural and robust. Fortunately, this class is also quite expressive, as it accommodates most of the relevant data extraction tasks and properly contains the wrappers definable in some specifically designed wrapper programming languages [7].
- Finally, the *textual approach* interprets web pages as text strings and uses string pattern matching for data extraction [13].

Methods of wrapper generation

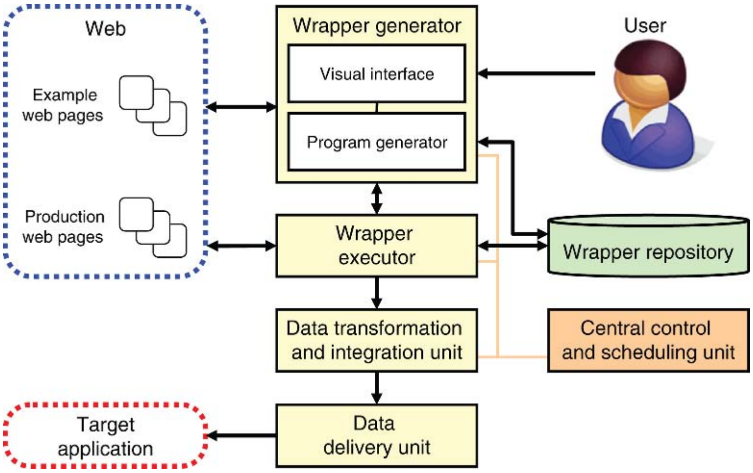
With regard to user involvement, three principal approaches for wrapper generation can be distinguished: (i) *Manual wrapper programming*, in which the system merely supports a user in writing a specific wrapper but cannot make any generalizations from the examples provided by the user; (ii) *wrapper induction*, where the user provides examples and counterexamples of instances of extraction patterns, and the system induces a suitable wrapper using machine learning techniques; and (iii) *semi-automatic interactive wrapper generation*, where the wrapper designer not only provides example data for the system, but rather accompanies the wrapper generation in a systematic computer-supported interactive process involving generalization, correction, testing, and visual programming techniques.

Architecture

Figure 1 depicts a high-level view of a typical fully-fledged semi-automatic interactive web data extraction system. This system comprises several tightly connected components and interfaces three external entities: (i) *the Web*, which contains pages with information of interest; (ii) a *target application*, to which the extracted and refined data will be ultimately delivered; and (iii) *the user*, who interactively designs the wrapper.

The wrapper generator supports the user during the wrapper design phase. It commonly has a *visual interface* that allows the user to define which data should be extracted from web pages and how this data should be mapped into a structured format such as XML. The visual interface itself can include several windows, such as: (i) a browser window that renders example web pages for the user; (ii) a parse tree window that shows the HTML parse tree of the current web page rendered by the browser; (iii) a control window that allows the user to view and control the overall progress of the wrapper design process and to input textual data (e.g., attribute names or XML tags); and (iv) a program window that displays the wrapper program constructed so far and allows the user to further adjust or correct the program.

The subunit that actually generates the wrapper is contained in what is referred to here as the *program generator*. This submodule interprets the user actions on the example web pages and successively generates the wrapper. Semi-automatic wrapper generators allow the user to either specify the URL of example web pages, which are structurally similar to the target web pages, or to navigate to such example pages. In the latter case, the navigation is recorded and can be automatically reproduced. The example pages are rendered in a browser window. Most wrapper generators also display the parse tree of the current page. In most systems, the wrapper designer can click on a node of the HTML parse tree, and the system immediately highlights the corresponding part of the example page in the browser, and vice versa. The designer can, thus, iteratively narrow down an example of a relevant data item after a few trials. Many web data extraction systems exhibit an XPATH-like path expression that precisely identifies the selected data item. The wrapper designer can then generalize this path expression by replacing some of the elements with wildcards.



Web Data Extraction System. Figure 1. Architecture of a typical web data extraction system.

The result is a generalized pattern that matches several similar data items on similarly structured pages. Some systems (e.g., Denodo and Lixto) allow a wrapper designer to select data items directly on the rendered web page. The wrapper generator often also provides support for associating a name or tag to each extraction pattern and for organizing patterns hierarchically. Systems based on wrapper induction allow the user to supply a large number of sample web pages with positive and/or negative examples of desired data items, from which the program generator tries to learn a generalized pattern. Systems for manual wrapper programming simply provide a visual environment for developing a wrapper in the underlying wrapper programming language, for testing the wrapper against example pages and for debugging. In addition, the visual interface usually allows an application designer to specify when a wrapper should be executed.

The *wrapper executor* is the engine that facilitates the deployment of the previously generated wrappers. Typically, a wrapper program comprises functions such as deep web navigation (e.g., the means to fill out forms), identification of relevant elements with declarative rules or procedural scripts, and generation of a structured output format (e.g., XML or relational tables). Typically, the wrapper executor can also receive additional input parameters such as a start URL, a predefined output structure (e.g., a particular XML schema), and special values for forms. In some approaches, the output includes metadata (e.g., verification alerts in case of violated integrity constraints) in addition to the actual data of interest.

The *wrapper repository* stores the generated wrappers together with metadata. In some approaches, wrapper templates and libraries of domain concepts can be reused and extended. Systems such as Dapper (Dapper. http://www.dapper.net/) offer a community-based extensible wrapper repository. In principle, systems can offer automatic wrapper maintenance and adaptation as part of the repository.

The *data transformation and integration unit* provides a means to streamline the results of multiple wrappers into one harmonized homogeneous result. This step includes data integration, transformation and cleaning, basically functions commonly found in the Transform step of ETL processes of data warehouses and mediation systems. Technologies and tools comprise query languages, visual data mapping, automatic schema matching and de-duplication techniques. Additionally, the data can be composed into desired result formats such as HTML, Excel or PDF.

The *data delivery unit* (capturing the Load step in the ETL philosophy) decides about the appropriate output channel such as email, Open Database Connectivity (ODBC) connection, or FTP. Advanced wrapper generation systems offer a number of system connectors to state-of-the-art databases and enterprise software such as Customer Relationship Management (CRM) or Enterprise Resource Planning (ERP), or partner with software providers in the area of Enterprise Application Integration (EAI) or Business Intelligence (BI) who offer such solutions.

The *central control and scheduling unit* is the heart of the processing engine. In general, synchronous and asynchronous extraction processes can be distinguished. A typical example of an asynchronous triggering is a real-time user request in a flight meta-search application. The request is queued, the respective wrappers are triggered, parameter mappings are performed, and the result is integrated and presented to the user in real-time. Sophisticated approaches offer a workflow-based execution paradigm, including parallelism, returning partial results, and interception points in complex booking transactions. On the other hand, market monitoring or web process integration are typical synchronous scenarios. An intelligent scheduler is responsible for distributing requests to the wrapper executor, iterating over a number of possible form inputs, and dealing with exception handling.

Commercial wrapper generation systems

Dapper. The Dapp Factory of Dapper offers fully server-based wrapper generation, and supports a community driven reusable wrapper repository. Dapper strongly relies on machine learning techniques for wrapper generation and concentrates exclusively on web pages that can be reached without deep web navigation. Users designing a wrapper label positive and negative examples and debug the result on a number of similarly structured web pages. Wrappers are hosted on the server and wrapper results can be queried via REST. Commercial APIs are offered for using Dapper in Enterprise scenarios.

Denodo (Denodo. http://www.denodo.com). The Denodo ITPilot, formerly known as Wargo, is a platform for creating and executing navigation and extraction scripts that are loosely tied together. It offers graphical wizards for configuring wrappers and allows DOM events to be processed while navigating web pages. Deep Web navigation can be executed in

Internet Explorer, and the result pages are passed on to the extraction program. Furthermore, ITPilot offers some wrapper maintenance functionalities. Denodo additionally offers a tool called Aracne for document crawling and indexing.

Lixto. The Lixto Suite comprises the Lixto Visual Developer (VD), a fully visual and interactive wrapper generation framework, and the Java-based Lixto Transformation Server providing a scalable runtime and data transformation environment, including various Enterprise Application connectors. VD is ideally suited for dynamic Web 2.0 applications as it supports a visual Deep Web macro recording tool that makes it possible to simulate user clicks on DOM elements. Tightly connected with the navigation steps and hidden from the wrapper designer, the expressive language Elog is used for data extraction and linking to further navigation steps. VD is based on Eclipse and embeds the Mozilla browser.

Kapowtech. The Kapow RoboSuite (recently rebranded to Kapow Mashup Server) is a Java-based visual development environment for developing web wrappers. It embeds a proprietary Java browser, implements a GUI on top of a procedural scripting language, and maps data to relational tables. The RoboServer offers APIs in different programming languages for synchronous and asynchronous communication. In addition, variants of the Mashup Server for specific settings such as content migration are offered.

WebQL. QL2 uses a SQL-like web query language called WebQL for writing wrappers. By default, WebQL uses its own HTML DOM tree model instead of relying on a standard browser, but a loose integration with Internet Explorer is offered. The QL2 Integration Server supports concepts such as server clustering and HTML diffing. Furthermore, an IP address anonymization environment is offered.

Key Applications

Web market monitoring. Nowadays, a lot of basic information about competitors can be retrieved legally from public information sources on the Web, such as annual reports, press releases or public data bases. On the one hand, powerful and efficient tools for Extracting, Transforming and Loading (ETL) data from internal sources and applications into a Business Intelligence (BI) data warehouse are already available and largely employed. On the other hand, there is a growing economic need to efficiently integrate external data, such as market and competitor information, into these systems as well. With the World Wide Web as the largest single database on earth, advanced data extraction and information integration techniques as described in this paper are required to process this web data automatically. At the same time, the extracted data has to be cleaned and transformed into semantically useful formats and delivered in a "web-ETL" process into a BI system. Key factors in this application area include scalable environments to extract and schedule processing of very large data sets efficiently, capabilities to pick representative data samples, cleaning extracted data to make it comparable, and connectivity to data warehouses.

Web process integration. In markets such as the automotive industry, business processes are largely carried out by means of web portals. Business critical data from various divisions such as quality management, marketing and sales, engineering, procurement and supply chain management have to be manually gathered from web portals. Through automation, suppliers can dramatically reduce the cost while at the same time improving speed and reliability of these processes. Additionally, the automation means to leverage web applications to web services, and hence wrapper generation systems can be considered as an enabling technology for Service Oriented Architectures (SOA) as envisioned and realized in Enterprise Application Integration (EAI) and B2B processes. Key factors in this application area are workflow capabilities for the whole process of data extraction, transformation and delivery, capabilities to treat all kinds of special cases occurring in web interactions, and excellent support of the latest web standards used during secure transactions.

Mashups. Increasingly, leading software vendors have started to provide mashup platforms such as Yahoo! Pipes or IBM QEDWiki. A mashup is a web application that combines a number of different websites into an integrated view. Usually, the content is taken via APIs by embedding RSS or atom feeds similar to REST (Representational State Transfer). In this context, wrapper technology transforms legacy web applications to light-weight APIs that can be integrated in mashups in the same way. As a result, web mashup solutions no longer need to rely on APIs offered by the providers of websites, but can rather extend the scope to the whole Web. Key factors for this application scenario include efficient real-time extraction capabilities for a large number of concurrent queries and detailed understanding of how to map queries to particular web forms.

Future Directions

▶ *Generic web wrapping.* Without explicit semantic annotations on the current Web data extraction systems that allow general web wrapping will have to move towards fully automatic wrapping of the existing World Wide Web. Important research challenges are (i) how to optimally bring semantic knowledge into the extraction process, (ii) how to make the extraction process robust (in particular, how to deal with false positives), and (iii) how to deal with the immense scaling issues. Today's web harvesting and automated information extraction systems show much progress (see e.g., [4,12]), but still lack the combined recall and precision necessary to allow for very robust queries. A related topic is that of *auto-adapting wrappers.* Most wrappers today depend on the tree structure of a given web page and suffer from failure when the layout and code of web pages change. Auto-adapting wrappers, which are robust against such changes, could use existing knowledge of the relations on previous versions of web page in order to automatically "heal" and adapt the extraction rules to the new format. The question here is how to formally capture change actions and execute the appropriate repair actions. ▶ *Wrapping from visual layouts.* Whereas web wrappers today dominantly focus on either the flat HTML code or the DOM tree representation of web pages, recent approaches aim at extracting data from the CSS box model and, hence, the visual representation of web pages [5]. This method can be particularly useful for layout-oriented data structures such as web tables and allows to create automatic and domain-independent wrappers which are robust against changes of the HTML code implementation. ▶ *Data extraction from non-HTML file formats.* There is a substantial interest from industry in wrapping documents in formats such as PDF and PostScript. Wrapping of such documents must be mainly guided by a visual reasoning process over white space and Gestalt theory, which is substantially different from web wrapping and will, hence, require new techniques and wrapping algorithms including concepts borrowed from the document understanding community. ▶ *Learning to deal with web interactions.* As web pages are becoming increasingly dynamic and interactive, efficient wrapping languages have to make it possible to record, execute and generalize macros of web interactions and, hence, model the whole process of workflow integration. An example of such a web interactions is a complicated booking transaction. ▶ *Web form understanding and mapping.* In order to automatically or interactively query deep web forms, wrappers have to learn the process of filling out complex web search forms and the usage of query interfaces. Such systems have to learn abstract representation for each search form and map them to a unified meta form and vice versa, taking into account different form element types, contents and labels.

Cross-references

▶ Business Intelligence
▶ Data Integration in Web Data Extraction System
▶ Data Mining
▶ Data Warehouse
▶ Datalog
▶ Deep-web Search
▶ Enterprise Application Integration
▶ Extraction, Transformation, and Loading
▶ Information Extraction
▶ Information Integration
▶ Logical Foundations Of Web Data Extraction
▶ MashUp
▶ Screen Scraper
▶ Semantic Web
▶ Service Oriented Architecture
▶ Snippet
▶ Web Harvesting
▶ Web Information Extraction
▶ Web Services
▶ Wrapper Induction
▶ Wrapper Maintenance
▶ Wrapper Stability
▶ XML
▶ Xpath/Xquery

Recommended Reading

1. Anupam V., Freire J., Kumar B., and Lieuwen D. Automating web navigation with the WebVCR. Comput. Network., 33(1–6):503–517, 2000.
2. Baumgartner R., Flesca S., and Gottlob G. Visual web information extraction with Lixto. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 119–128.
3. Crescenzi V., Mecca G., and Merialdo P. Road runner: towards automatic data extraction from large Web sites. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 109–118.
4. Etzioni O., Cafarella M.J., Downey D., Kok S., Popescu A., Shaked T., Soderland S., Weld D.S., and Yates Y. Web-scale information extraction in KnowItAll: (preliminary results). In Proc. 12th Int. World Wide Web Conference, 2004, pp. 100–110.

5. Gatterbauer W., Bohunsky P., Herzog M., Krüpl B., and Pollak B. Towards domain-independent information extraction from web tables. In Proc. 16th Int. World Wide Web Conference, 2007, pp.71–80.
6. Gottlob G. and Koch C. Monadic datalog and the expressive power of languages for web information extraction. J. ACM 51(1):74–113, 2002.
7. Gottlob G. and Koch C.A. Formal comparison of visual web wrapper generators. In Proc. 32nd Conf. Current Trends in Theory and Practice of Computer Science, 2006, pp. 30–48.
8. Kuhlins S. and Tredwell R. Toolkits for generating wrappers: a survey of software toolkits for automated data extraction from Websites. NODe 2002, LNCS:2591, 2003.
9. Kushmerick N., Weld D.S., and Doorenbos R.B. Wrapper induction for information extraction. In Proc. 15th Int. Joint Conf. on AI, 1997, pp. 729–737.
10. Laender A.H.F., Ribeiro-Neto B.A., and da Silva A.S. DEByE – data extraction by example. Data Knowl. Eng., 40(2):121–154, 2000.
11. Liu L., Pu C., and Han W. XWRAP: an XML-enabled wrapper construction system for web information sources. In Proc. 16th Int. Conf. on Data Engineering, 2000, pp. 611–621.
12. Liu B., Grossman R.L., and Zhai Y. Mining web pages for data records. IEEE Intell. Syst., 19(6):49–55, 2004.
13. Muslea I., Minton S., and Knoblock C.A. Hierarchical Wrapper Induction for Semistructured Information Sources. Autonom. Agents Multi-Agent Syst., 4(1/2):93–114, 2001.
14. Pan A., Raposo J., Álvarez M., Montoto P., Orjales V., Hidalgo J., Ardao L., Molano A., and Viña Á. The Denodo data integration platform. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
15. Sahuguet A. and Azavant F. Building intelligent web applications using lightweight wrappers. Data Knowl. Eng., 36(3):283–316, 2001.

Web Data Mining

▶ Data, Text, and Web Mining in Healthcare

Web Directories

▶ Lightweight Ontologies

Web ETL

Oliver Frölich
Lixto Software GmbH, Vienna, Austria

Synonyms

ETL using web data extraction techniques

Definitions

As ETL (acronym for Extraction, Transformation and Loading) is a well-established technology for the extraction of data from several sources, their cleansing, normalization and insertion into a Data Warehouse (e.g., a Business Intelligence System), *Web ETL* stands for an ETL process where the external data to be inserted into the Data Warehouse is extracted from semi-structured Web pages (e.g., in HTML or PDF format) using Web Data Extraction techniques.

Particularly, back-end interchange of structured data just *using* the Web, e.g., two database systems exchanging data with Web EDI technology (EDI (Electronic Data Interchange) stands for techniques and standards for the transmission of structured data, for example over the Web, in an application-to-application context.), is not a Web ETL process as no semi-structured data needs to be transformed using Web Data Extraction techniques.

Key Points

Powerful and efficient tools supporting ETL processes (Extracting, Transforming and Loading of data) in a Data Warehouse context exist for years now. They concentrate on the data extraction from *internal* applications. But there is also a growing need to integrate *external* data: For example in Competitive Intelligence, information about competitor activities and market developments is becoming a more and more important success factor for enterprises. The largest information source on earth is the World Wide Web. Unfortunately, data on the Web requires intelligent interpretation and cannot be easily used by programs, since the Web is primarily intended for human users. Therefore, the extraction from semi-structured information sources like Web pages was mostly done manually and was very time consuming just a few years ago. Today, sophisticated toolsets for Web Data Extraction exist for retrieving relevant data automatically from Web sites and for transforming it into structured data formats. An example of such a toolset is the Lixto Suite [1], allowing for the extraction and transformation of data from Web pages into structured XML formats. This structured XML data can be integrated in Data Warehouse systems. This whole process from Web Data Extraction to the integration of the cleansed and normalized information is called a Web ETL process. Web ETL itself can be a part of a Business Intelligence process, turning semi-structured data into business-relevant information in a Business Intelligence Data Warehouse. An example of a Business Intelligence application of Web ETL is given in [2].

Another application field of Web ETL besides Competitive Intelligence and Business Intelligence is Front-End System Integration, where Web interfaces of business applications are utilized for coupling these systems using Web ETL processes.

Cross-references

- ▶ Business Intelligence
- ▶ Competitive Intelligence
- ▶ Data Warehouse
- ▶ ETL
- ▶ Semi-Structured Data

Recommended Reading

1. Baumgartner R., Flesca S., Gottlob G. Visual web information extraction with Lixto. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 119–128.
2. Baumgartner R., Frölich O., Gottlob G., Harz P., Herzog M., and Lehmann P. Web data extraction for business intelligence: the Lixto approach, In Proc. Datenbanksysteme in Business, Technologie und Web (BTW), 2005, pp. 48–65.
3. Frölich O. Optimierung von Geschäftsprozessen durch Integrierte Wrapper-Technologien. Dissertation, Institute of Information Systems, Vienna University of Technology, 2006.

Web Harvesting

Wolfgang Gatterbauer
University of Washington, Seattle, WA, USA

Synonyms

Web data extraction; Web information extraction; Web mining

Definition

Web harvesting describes the process of gathering and integrating data from various heterogeneous web sources. Necessary input is an appropriate knowledge representation of the domain of interest (e.g., an *ontology*), together with example instances of concepts or relationships (*seed knowledge*). Output is structured data (e.g., in the form of a relational database) that is gathered from the Web. The term *harvesting* implies that, while passing over a large body of available information, the process gathers only such information that lies in the domain of interest and is, as such, relevant.

Key Points

The process of web harvesting can be divided into three subsequent tasks: (i) *data or information retrieval*, which involves finding relevant information on the Web and storing it locally. This task requires tools for searching and navigating the Web, i.e., crawlers and means for interacting with dynamic or deep web pages, and tools for reading, indexing and comparing the textual content of pages; (ii) *data or information extraction*, which involves identifying relevant data on retrieved content pages and extracting it into a structured format. Important tools that allow access to the data for further analysis are parsers, content spotters and adaptive wrappers; (iii) *data integration* which involves cleaning, filtering, transforming, refining and combining the data extracted from one or more web sources, and structuring the results according to a desired output format. The important aspect of this task is organizing the extracted data in such a way as to allow unified access for further analysis and data mining tasks.

The ultimate goal of web harvesting is to compile as much information as possible from the Web on one or more domains and to create a large, structured knowledge base. This knowledge base should then allow querying for information similar to a conventional database system. In this respect, the goal is shared with that of the Semantic Web. The latter, however, tries to solve extraction *à priori* to retrieval by having web sources present their data in a semantically explicit form.

Today's search engines focus on the task of finding content pages with relevant data. The important challenges for web harvesting, in contrast, lie in extracting and integrating the data. Those difficulties are due to the variety of ways in which information is expressed on the Web (*representational heterogeneity*) and the variety of alternative, but valid interpretations of domains (*conceptual heterogeneity*). These difficulties are aggravated by the Web's sheer size, its level of heterogeneity and the fact that information on the Web is not only complementary and redundant, but often contradictory too.

An important research problem is the optimal combination of automation (high recall) and human involvement (high precision). At which stages and in which manner a human user must interact with an otherwise fully automatic web harvesting system for optimal performance (in terms of speed, quality, minimum human involvement, etc.) remains an open question.

Cross-references

- ▶ Data Extraction
- ▶ Data Integration
- ▶ Fully-Automatic Web Data Extraction
- ▶ Information Retrieval
- ▶ Semantic Web
- ▶ Web Data Extraction
- ▶ Web Data Extraction System
- ▶ Web Scraper
- ▶ Wrapper

Recommended Reading

1. Ciravegna F., Chapman S., Dingli A., and Wilks Y. Learning to harvest information for the Semantic Web. In Proc. 1st European Semantic Web Symposium, 2004, pp. 312–326.
2. Crescenzi V. and Mecca G. Automatic information extraction from large websites. J. ACM, 51(5):731–779, 2004.
3. Etzioni O., Cafarella M.J., Downey D., Kok S., Popescu A.M., Shaked T., Soderland S., Weld D.S., and Yates A. Web-scale information extraction in KnowItAll: (preliminary results). In Proc. 12th Int. World Wide Web Conference, 2004, pp. 100–110.

Web Indexing

▶ Indexing the Web

Web Information Extraction

Rajasekar Krishnamurthy, Yunyao Li,
Sriram Raghavan, Frederick Reiss,
Shivakumar Vaithyanathan, Huaiyu Zhu
IBM Almaden Research Center, San Jose, CA, USA

Synonyms

Text Analytics; Information Extraction

Definition

Information extraction (IE) is the process of automatically extracting structured pieces of information from unstructured or semi-structured text documents. Classical problems in information extraction include *named-entity recognition* (identifying mentions of persons, places, organizations, etc.) and *relationship extraction* (identifying mentions of relationships between such named entities). Web information extraction is the application of IE techniques to process the vast amounts of unstructured content on the Web. Due to the nature of the content on the Web, in addition to named-entity and relationship extraction, there is growing interest in more complex tasks such as extraction of reviews, opinions, and sentiments.

Historical Background

Historically, information extraction was studied by the Natural Language Processing community in the context of identifying organizations, locations, and person names in news articles and military reports [9]. From early on, information extraction systems were based on the *knowledge engineering approach* of developing carefully crafted sets of *rules* for each task. These systems view the text as an input sequence of symbols and extraction rules are specified as regular expressions over the lexical features of these symbols. The formalism underlying these systems is based on cascading grammars and the theory of finite-state automata. One of the earliest languages for specifying such rules is the Common Pattern Specification Language (CPSL) developed in the context of the TIPSTER project [1]. To overcome some of the drawbacks of CPSL resulting from a sequential view of the input, the AfST system [2] uses a more powerful grammar that views its input as an object graph.

Beginning in the mid-1990s, as the unstructured content on the Web continued to grow, information extraction techniques were applied in building popular Web applications. One of the earliest such uses of information extraction was in the context of *screen scraping* for online comparison shopping and data integration applications. By manually examining a number of sample pages, application designers would develop ad hoc rules and regular expressions to eke out relevant pieces of information (e.g., the name of a book, its price, the ISBN number, etc.) from multiple Web sites to produce a consolidated Web page or query interface. Recently, more sophisticated IE techniques are being employed on the Web to improve search result quality, guide ad placement strategies, and assist in reputation management [7,12].

Foundations

Knowledge-engineered rules have the advantage that they are easy to construct in many cases (e.g., rules to recognize prices, phone numbers, zip codes, etc.), easier to debug and maintain when written in a high-level rule language, and provide a natural way to incorporate domain or corpus-specific knowledge. However, such rules are extremely labor intensive to develop and maintain. An alternative paradigm for producing extraction rules is the use of learning-based methods [6]. Such methods work well when training data is readily available and the extraction tasks are hard to encode manually. Finally, there has been work in the use of complex statistical models, such as Hidden Markov Models and Conditional Random Fields, where the rules of extraction are implicit within the parameters of the model [11].

For ease of exposition, the rule-based paradigm is used to present the core concepts of Web information extraction. Rule-based extraction programs are called *annotators* and their output is referred to as *annotations.* The two central concepts of rule-based extraction are *rules* and *spans.* A span corresponds to a substring of the document text represented as a pair of offsets *(begin, end).* A rule is of the form $A \leftarrow P$ (Fig. 1), where A is an annotation (specified within angled brackets) and P is a pattern specification. Evaluating the rule associates any span of text that matches pattern P with the annotation A.

The description of information extraction is organized around four broad categories of extraction tasks: *entity extraction*, *relationship extraction*, *complex composite extraction*, and *application-driven extraction.* The first two categories, while relevant and increasingly used in Web applications, are classical IE tasks that have been extensively studied in the literature even before the advent of the Web.

Category 1. *Entity extraction* refers to the identification of mentions of named entities (such as persons, locations, organizations, phone numbers, etc.) in unstructured text. While the task of entity extraction is intuitive and easy to describe, the corresponding annotators are fairly complex and involve a large number of rules and carefully curated dictionaries. For example, a high-quality annotator for person names would involve several tens of rules to capture all of the different conventions, shorthands, and formats used in person names all over the world.

Example 1. As an illustrative example, consider the simple annotator shown in Fig. 1 for recognizing mentions of person names. The annotator uses a CPSL-style cascading grammar specification. Assume that the input text has already been tokenized and is available as a sequence of ⟨Token⟩ annotations. Rules R_4, R_5, and R_6 are the lowest level grammar rules since they operate only on the input ⟨Token⟩ annotations. Each of these rules attempts to match the text of a token against a particular regular expression and produces output annotations whenever the match succeeds. Rule R_6 states that a span of text consisting of a single token whose text matches a dictionary of person names ("Michael," "Richard," etc.) is a ⟨PersonDict⟩ annotation. Rule R_4 similarly defines ⟨Salutation⟩ annotations (e.g., Dr., Prof., Mr., etc.) and R_5 defines annotations consisting of a single token beginning with a capital letter. Finally, rules R_1, R_2, and R_3 are the higher-level rules of the cascading grammar since they operate on the annotations produced by the lower-level rules. For example, R_1 states that a salutation followed by two capitalized words is a person name. Such a rule will recognize names such as "Dr. Albert Einstein" and "Prof. Michael Stonebraker."

Category 2. *Binary relationship extraction* refers to the task of associating pairs of named entities based on the identification of a relationship between the entities. For instance, Example 2 describes the task of extracting instances of the *CompanyCEO* relationship, i.e., finding pairs of person and organization names such that the person is the CEO of the organization.

Example 2. Assume that mentions of persons and organizations have already been annotated (as in Category 1 above) and are available as ⟨Person⟩ and ⟨Organization⟩ annotations respectively. Figure 2 shows rules that identify instances of the *CompanyCEO* relationship. Rule R_7 looks for pairs of ⟨Person⟩ and ⟨Organization⟩ annotations such that the text

<table>
<tr><td>⟨Person⟩</td><td>←</td><td>⟨Salutation⟩ ⟨CapsWord⟩ ⟨CapsWord⟩</td><td>(R_1)</td></tr>
<tr><td>⟨Person⟩</td><td>←</td><td>⟨PersonDict⟩ ⟨PersonDict⟩</td><td>(R_2)</td></tr>
<tr><td>⟨Person⟩</td><td>←</td><td>⟨CapsWord⟩ ⟨PersonDict⟩</td><td>(R_3)</td></tr>
<tr><td>⟨Salutation⟩</td><td>←</td><td>⟨Token⟩ [≃"Mr. |Mrs. |Dr. | ..."]</td><td>(R_4)</td></tr>
<tr><td>⟨CapsWord⟩</td><td>←</td><td>⟨Token⟩ [≃text() ≃ "[A-Z] [a-z] *"]</td><td>(R_5)</td></tr>
<tr><td>⟨PersonDict⟩</td><td>←</td><td>⟨Token⟩ [text() ≃ "Michael |Richard |Smith| ...."]</td><td>(R_6)</td></tr>
</table>

Web Information Extraction. Figure 1. Simple rules for identifying person names.

between these annotations satisfies a particular regular expression. The regular expression that is used in this example will match any piece of text containing the phrase “CEO of” with an optional comma before the phrase and an arbitrary amount of whitespace separating the individual words of the phrase. Thus, it will correctly extract an instance of this relationship from the text “Sam Palmisano, CEO of IBM.” As with Example 1, the rules presented here are merely illustrative and high-quality relationship annotators will involve significantly more rules.

Category 3. *Complex composite extraction.* The presence of vast amounts of user generated content on the Web (in blogs, wikis, discussion forums, and mailing lists) has engendered a new class of complex information extraction tasks. The goal of such tasks is the extraction of reviews, opinions, and sentiments on a wide variety of products and services. Annotators for such complex tasks are characterized by two main features: (i) the use of entity and relationship annotators as sub-modules to be invoked as part of a higher level extraction workflow, and (ii) the use of aggregation-like operations in the extraction process.

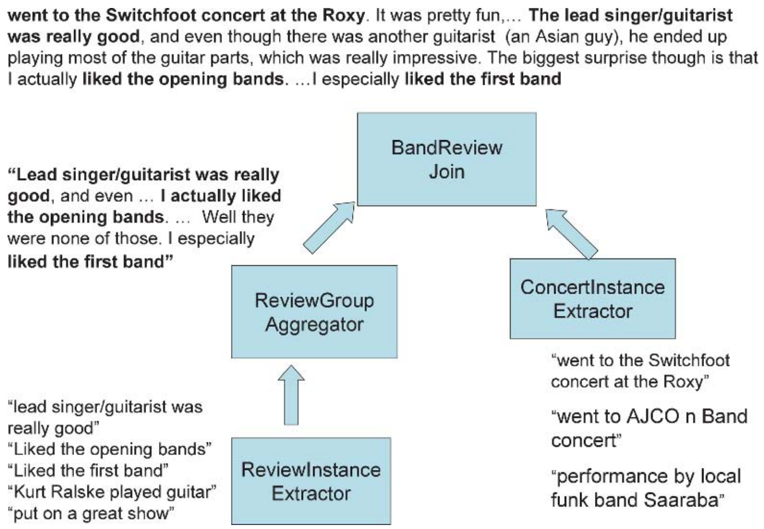
Example 3. Consider the task of identifying informal reviews of live performance of music bands embedded within blog entries. Figure 3 shows the high-level workflow of an annotator that accomplishes this task. The two individual modules, *ReviewInstance Extractor* and *ConcertInstance Extractor*, identify specific snippets of text in a blog. The ⟨ReviewInstance⟩ module identifies snippets that indicate portions of a concert review – e.g., “show was great,” “liked the opening bands” and “Kurt Ralske played guitar.” Similarly, the *ConcertInstance Extractor* module identifies occurrences of bands or performers – e.g., “performance by the local funk band Saaraba” and “went to the Switchfoot concert at the Roxy.” The output from the *ReviewInstance Extractor* module is fed into the *ReviewGroup Aggregator* module to identify contiguous blocks of text containing ⟨ReviewInstance⟩ snippets. Finally, a ⟨ConcertInstance⟩ snippet is associated with one or more ⟨ReviewGroups⟩ to obtain ⟨BandReviews⟩. Note that in addition to the complex high-level workflow, each individual module in Fig. 3 is itself fairly involved and consists of tens of regular expression patterns and dictionaries.

Category 4. *Application-driven extraction.* The last category of extraction tasks covers a broad spectrum of scenarios where IE techniques are applied to perform extraction that is unique to the needs of a particular

⟨CompanyCEO⟩ ← ⟨Person⟩ RegExp(“ \s+,?\s+CE0\s+of\s+ ”) ⟨Organization⟩ (R_7)

⟨CompanyCEO⟩ ← ⟨Organization⟩ RegExp(“ \s+CE0\s+ ”) ⟨Person⟩ (R_8)

Web Information Extraction. Figure 2. Simple rules for identifying CompanyCEO relationship instances.


went to the Switchfoot concert at the Roxy. It was pretty fun,... The lead singer/guitarist was really good, and even though there was another guitarist (an Asian guy), he ended up playing most of the guitar parts, which was really impressive. The biggest surprise though is that I actually liked the opening bands. ...I especially liked the first band
BandReview Join
“Lead singer/guitarist was really good, and even ... I actually liked the opening bands. ... Well they were none of those. I especially liked the first band”
ReviewGroup Aggregator
ConcertInstance Extractor
“went to the Switchfoot concert at the Roxy”
“went to AJCO n Band concert”
“performance by local funk band Saaraba”
“lead singer/guitarist was really good”
“Liked the opening bands”
“Liked the first band”
“Kurt Ralske played guitar”
“put on a great show”
ReviewInstance Extractor

Web Information Extraction. Figure 3. Extraction of informal band reviews.

Web application. While the early applications of IE were ad hoc and limited to screen scraping, there has been recent widespread use of IE in the context of improving search quality. A prototypical example is the use of specially crafted extraction rules applied to the URLs and titles of Web pages to identify high-quality index terms. Consider the following URLs:

U_1 http://en.wikipedia.org/wiki/Michelangelo
U_2 http://www.ibiblio.org/wm/paint/auth/michelangelo/
U_3 http://www.michelangelo.com/
U_4 http://www.artcyclopaedia.com/artists/michelangelo_buonarotti.html

It is easy to see that certain key segments of the URL string, such as the last segment of the path (as in U_1 and U_2), the portion of the hostname following the "www" (in U_3) and the actual resource name (as in U_4) provide fairly reliable clues as to the actual content of the corresponding Web page. Modern search engines, on the Web and in the intranet, are applying sophisticated patterns to extract such key segments from the URL and produce terms for their search index. In the above examples, such a technique would enable the term "michelangelo" or the phrase "michelangelo buonarotti" to be identified as high-quality index terms for the Web pages. In a similar fashion, extraction of key phrases from the captions of images and videos are being used to effectively index and search over online multimedia repositories. Other specialized search engines for verticals such as healthcare, finance, and people search, also make significant use of information extraction to identify domain-specific concepts.

Key Applications

As the vast majority of information on the Web is in unstructured form, there is growing interest, within the database, data mining, and IR communities, in the use of information extraction for Web applications. Several research projects in the areas of *intranet search*, *community information management*, and *Web analytics*, are already employing IE techniques to bring order to unstructured data. The common theme in all of these applications is the use of IE to process the input text and produce a structured representation to support search, browsing, and mining applications.

Web search engines are also employing IE techniques to recognize key entities (persons, locations, organizations, etc.) associated with a web page. This semantically richer understanding of the contents of a page is used to drive corresponding improvements to their search ranking and ad placement strategies.

Finally, there is continuing interest in techniques to reliably extract reviews, opinions, and sentiments about products and services from the content present in online communities and portals. The extracted information can be used to guide business decisions related to product placement, targeting, and marketing. The complex nature of these extraction tasks as well as the heterogeneous and noisy nature of the data pose interesting research challenges.

Future Directions

While the area of information extraction has made considerable progress since inception, several important challenges still remain open. Two of these challenges that are under active investigation in the research community are described below.

Scalability

As complex information tasks move to operating over Web-size data sets scalability has become a major focus [8]. In particular, IE systems need to scale to large numbers of input documents (data set size) and to large numbers of rules and patterns (annotator complexity). Two classes of optimizations being considered in this regard are

- Low-Level Primitives: Improving the performance of the low-level operations that dominate annotator execution time.
- High-Level Optimization: Automatically choosing more efficient orders for evaluating annotator rules.

Low-Level Primitives In many information extraction systems, low-level text operations like tokenization, regular expression evaluation and dictionary matching dominate execution time. Speeding up these low-level operations leads to direct improvements in scalability, both in terms of the number of documents the system can handle and the number of basic operations the system can afford to perform on a given document [5]. Some of the problems being addressed currently in this context are given below.

- When a large number of dictionaries are evaluated in an annotator (e.g., the *BandReview* annotator (Fig. 3) evaluates over 30 unique dictionaries), evaluating each dictionary separately is expensive as the document text is tokenized once per dictionary evaluation. To address this inefficiency, techniques are being explored to evaluate multiple dictionaries efficiently in a single pass over the document text.

- Evaluating complex regular expressions over every document is an expensive operation. An active area of research is the design of faster regular expression matchers for special classes of regular expressions. Another approach being considered is the design of "filter" regular expression indexes. These indexes can be used to quickly eliminate many of the documents that do not contain a match.

High-Level Optimization As web information extraction moves towards Categories 3 and 4 (more complex tasks) there is more opportunity for improving efficiency [3–14]. Some of these efficiency gains can be obtained using traditional relational optimizations such as "join reordering" and "pushing down selection predicates." There are also additional text-specific optimizations that may give significant benefit, two of which are described below.

- *Restricted Span Evaluation*: Let R denote a set of rules for a given information extraction task. Let *spans*(d, R')be a set of spans obtained by evaluating $R' \subset R$ over a document d. Let $r \in (R - R')$ be a rule to be evaluated over the complete text of d. Restricted Span Evaluation is the optimization technique by which theannotator specification can be rewritten so that rule r is evaluated only on *spans*$(d, R')$and not over the complete text of d.

  A specific instantiation of restricted span evaluation is presented below, using Rule R_1 in the *Person* annotator (Fig. 1). This rule identifies a salutation followed by two capitalized words. Note that R_1 uses R_4 and R_5 for identifying ⟨`Salutation`⟩ and ⟨`CapsWord`⟩ respectively. A naive evaluation strategy identifies all occurrences of ⟨`Salutation`⟩ and ⟨`CapsWord`⟩ over the complete document before R_1 is evaluated. An alternative approach is to first identify ⟨`Salutation`⟩ in the document and then look for ⟨`CapsWord`⟩ only in the immediate vicinity of all the ⟨`Salutation`⟩ annotations. The latter approach evaluates the ⟨`CapsWord`⟩ rule only on a smaller amount of text as the ⟨`Salutation`⟩ annotations occur infrequently. This can result in considerable performance improvements.

- *Conditional Evaluation*: Let R denote a set of rules for a given information extraction task and R_1 and R_2 be two non-overlapping subsets of R. Conditional Evaluation is the optimization technique by which the annotatorspecification can be rewritten so that R_1 is conditionally evaluated on a document d depending on whether the evaluation of R_2 over $d$satisfies certain predicates. An example predicate is whether the evaluationof R_2 produces at least one annotation.

  A specific instantiation of Conditional Evaluation in the context of the *BandReview* annotator (Fig. 3) is given below. There are two main modules in the annotator, *ConcertInstance* and *ReviewGroup* which can be conditionally evaluated – i.e., the absence of one in a document implies that the other need not be evaluated on that document. In general, Conditional Evaluation is applicable in large workflows at join conditions such as the *BandReview Join* in Fig. 3.

Initial results on combining efficient evaluation of lower-level primitives and higher-level optimizations are very encouraging and recent results have reported an order of magnitude improvement in execution time [13,14].

Uncertainty Management

Real-world annotators typically consist of a large number of rules with varying degrees of accuracy. For example, most annotations generated by Rule R_1 (Fig. 1) are likely to be correct since salutation is a very strong indicator of the existence of a person name. Rule R_3, on the other hand, will identify some correct names such as "James Hall" and "Dan Brown" along with some spurious annotations such as "Town Hall" and "Dark Brown." Since "Hall" and "Brown" are ambiguous person names that also appear in other contexts and these dictionary matches are combined with a capitalized word, R_3 has a lower accuracy than R_1. Formally this difference in accuracy between different rules can be captured by the notion of the *precision* of a rule.

Rule Precision Suppose rule R identifies N annotations over a given document corpus, of which $n_{correct}$ annotations are correct. Then the precision of rule R is the fraction $\frac{n_{correct}}{N}$. Each annotation a is associated with a confidence value that is directly related to the precision of the corresponding rule.

The confidence value associated with an annotation enables applications to meaningfully represent and manipulate the imprecision of information extracted from text [4]. For Category 1 tasks, the associating of such confidence values (such as probabilities) has been addressed in existing literature. On the other hand, associating confidence values with annotations for more complex extraction tasks is still open [10]. To illustrate, consider Rule R_6 in Fig. 2 for identifying the *Company-CEO* relationship. This rule uses existing annotations ⟨Person⟩ and ⟨Organization⟩ . Therefore the confidence of a *CompanyCEO* annotation is dependent not only on the precision of rule R_6 but also on the confidences of the ⟨Person⟩ and ⟨Organization⟩ annotations. By modeling the confidence numbers as probabilities it is possible to view the individual rules as queries over a probabilistic database. However, direct application of current probabilistic database semantics (possible worlds) is precluded as illustrated in the following example. Rule R_7 identifies an instance of the *CompanyCEO* relationship from the text "Interview with Lance Brown, CEO of PeoplesForum.com." This rule looks for the existence of the text "CEO of" between ⟨Person⟩ and ⟨Organization⟩ thereby making it a highly accurate rule. Even though the participating ⟨Person⟩ annotation has a low confidence (identified by a rule with low precision, Rule R_3), the resulting ⟨CompanyCEO⟩ annotations should have high confidence due to the high precision of Rule R_7. Such behavior cannot be modeled under "possible worlds semantics" where the confidence of ⟨CompanyCEO⟩ annotations is bounded by the confidences of the input ⟨Person⟩ and ⟨Organization⟩ annotations. Handling such anomalies in the calculation of probabilities for relationship and complex composite extraction tasks is an open problem.

Cross-references

▶ Fully Automatic Web Data Extraction
▶ Information Extraction
▶ Languages for Web Data Extraction
▶ Metasearch Engines
▶ Probabilistic Databases
▶ Query Optimization
▶ Text Analytics
▶ Text Mining
▶ Uncertainty and Data Quality Mgmt
▶ Web Advertising
▶ Web Data Extraction System
▶ Web Harvesting
▶ Wrapper Induction

Recommended Reading

1. Appelt D.E. and Onyshkevych B. The common pattern specification language. In Tipster Text Program Phase 3, 1999.
2. Boguraev B. Annotation-based finite state processing in a large-scale NLP architecture. In Proc. Recent Advances in Natural Language Processing. 2003, pp. 61–80.
3. Cafarella M.J. and Etzion O. A search engine for natural language applications. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 442–452.
4. Cafarella M.J. et al. Structured querying of Web text: A technical challenge. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 225–234.
5. Chandel A., Nagesh P.C., and Sarawagi S. Efficient batch top-k search for dictionary-based entity recognition. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
6. Cohen W. and McCallum A. Information extraction from the world wide web Tutorial at Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003.
7. Cunningham H. Information Extraction, Automatic. In Encyclopedia of Language and Linguistics, 2nd edn. 2005.
8. Doan A., Ramakrishnan R., and Vaithyanathan S. Managing information extraction: state of the art and research directions. Tutorial in Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006.
9. Grishman R. and Sundheim B. Message understanding conference-6: a brief history. In Proc. 16th Conf. on Computational, 1996, pp. 446–471.
10. Jayram T.S., Krishnamurthy R., Raghavan S., Vaithyanathan S., and Zhu H. Avatar information extraction system. Q. Bull. IEEE TC on Data Engineering, 29(1):40–48, May 2006.
11. Lafferty J., McCallum A., and Pereira F. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In Proc. 18th Int. Conf. on Machine Learning, 2001, pp. 282–289.
12. Li Y., Krishnamurthy R., Vaithyanathan S., and Jagadish H. Getting work done on the web: supporting transactional queries. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 557–564.
13. Reiss F., Raghavan S., Krishnamurthy R., Zhu H., and Vaithyanathan S. An algebraic approach to rule-based information extraction. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 933–942.
14. Shen W., Doan A., Naughton J., and Ramakrishnan R. Declarative information extraction using datalog with embedded extraction predicates. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 1033–1044.

Web Information Extraction System

▶ Web Data Extraction System

Web Information Integration and Schema Matching

► Data Integration in Web Data Extraction System

WEB Information Retrieval Models

Craig MacDonald, Iadh Ounis
University of Glasgow, Glasgow, UK

Synonyms

Web search engines

Definition

The Web can be considered as a large-scale document collection, for which classical text retrieval techniques can be applied. However, its unique features and structure offer new sources of evidence that can be used to enhance the effectiveness of Information Retrieval (IR) systems. Generally, Web IR examines the combination of evidence from both the textual content of documents and the structure of the Web, as well as the search behavior of users and issues related to the evaluation of retrieval effectiveness in the Web setting.

Web Information Retrieval models are ways of integrating many sources of evidence about documents, such as the links, the structure of the document, the actual content of the document, the quality of the document, etc. so that an effective Web search engine can be achieved. In contrast with the traditional library-type settings of IR systems, the Web is a hostile environment, where Web search engines have to deal with subversive techniques applied to give Web pages artificially high search engine rankings. Moreover, the Web content in heavily duplicated, for example by mirroring, which search engines need to account for, while the size of the Web requires search engines to address the scalibility of their algorithms to create efficient search engines.

The commonly known PageRank algorithm – based on a documents hyperlinks – is an example of a source of evidence about a document to identify high quality documents. Another example is the commonly applied anchor text of the incoming hyperlinks. These sources of evidence are two of the most popular examples of link-based sources of evidence.

Historical Background

The first search engines for the Web appeared around 1992–1993, notably with the full-text indexing WebCrawler and Lycos both arriving in 1994. Soon after, many other search engines arrived, including Altavista, Excite, Inktomi and Northern Light. These often competed directly with directory-based services, like Yahoo!, which added search engine facilities later.

The rise in prominence of Google, in 2001, was due to its recognition that the underlying Web search user task is not an adhoc task (where users want lots of relevant documents, but not any ones in particular) to a more precision oriented task, where the relevance of the top-ranked result is important. This set Google apart from other search engines, since by using link analysis techniques (such as PageRank), hyperlink anchor text and other heuristics such as the title of the page, it could find itself at rank #1 for the query "google." This is something none of the other search engines at that time could achieve.

Since then, there has been a distinct consolidation in the Web search engine market, with only three major players taking the majority of the English market: Google, Yahoo! and MSN Live. However, additional search engines are thriving in other areas: Baidu and Yandex have high penetration in the Chinese and Russian markets respectively, while Technorati, BlogPulse and other blog search engines are popular in the blogosphere. However, since the arrival of Google, the Web IR research field has become much more active, with many more research groups, and more conferences than before, to the point that it is almost a separate research field in its own right, with two separate ACM conferences (World Wide Web Conference, and Web Search and Data Mining Conference), in addition to the existing IR conferences.

Foundations

The Web can be considered as a large-scale document collection, for which classical text retrieval techniques can be applied. However, its unique features and structure offer new sources of evidence that can be used to enhance the effectiveness of IR systems. Generally, Web IR examines the combination of evidence from both the textual content of documents and the structure of the Web, as well as the search behavior of users, and issues related to the evaluation of retrieval effectiveness.

The information available on the Web is very different from the information contained in either libraries or classical IR collections. A large amount of information on the Web is duplicated, and content is often mirrored across many different sites. Moreover, many documents can be unintentionally, or intentionally inaccurate, or are intended to mislead search engines since this is their purpose. This means that Web IR models need to be developed so they can perform well in such a hostile environment.

The Web is based on a hypertext document model, where documents are connected with direct hyperlinks. This results in a virtual network of documents. A major development in the Web IR field was that of query independent evidence. In particular, Page et al's PageRank algorithm [1] determines the quality of a document by examining the quality of the documents linked to it. In particular, the PageRank scores correspond to the probability of visiting a particular node in a Markov chain for the whole Web graph, where the states represent Web documents, and the transitions between states represent hyperlinks. PageRank was reported to be a fundamental component of the early versions of the Google search engine [1], and is beneficial in high-precision user tasks, where the relevance and quality of the top-ranked documents are important.

Many other similar link analysis algorithms have been proposed, including those that can be applied in a query-dependent or independent fashion. Most are based on random-walks, calculating the probability of a random Web user visiting a given page. Examples are Kleinberg's HITS [6] and the Absorbing Model [13].

Other sources of document quality have been reported, including the use of the URL evidence to determine the type of the page (for instance, whether the URL is short or long, or how many "/" characters it contains [7]).

The integration of such query independent evidence into the ranking strategy is an important issue. In the language modeling framework, it is natural to see the integration of query independent evidence as a document prior, that defines the overall likeliness of the document's retrieval [7]. Craswell et al. [3] introduce a framework (FLOE) for integrating query independent evidence with the retrieval score from BM25. The integration of several priors remains an important problem. Peng et al. [10] propose a probabilistic mehod of combining several document priors in the language modeling framework, while Plachouras [11] examines how various query independent evidence can be applied on a query-by-query basis.

The algorithms HITS and PageRank, along with their extensions, explicitly employ the hyperlinks between Web documents, in order to find high quality, or authoritative Web documents. A form of implicit use of the hyperlinks in combination with content analysis is to use the anchor text associated with the incoming hyperlinks of documents. Web documents can be represented by an anchor text surrogate, which is formed from collecting the anchor text associated with the hyperlinks pointing to the document. The anchor text of the incoming hyperlinks provides a concise description for a Web document. The used terms in the anchor text may be different from the ones that occur in the document itself, because the author of the anchor text is not necessarily the author of the document. Indeed, Craswell et al. [2] showed that anchor text is very effective for navigational search tasks and more specifically for finding home pages of Web sites.

Several models have since been developed that use the anchor text of a document in addition to the content. Kraaij et al. [7] and Ogilvie and Callan [9] describe mixture language modeling approaches, where the probability of a term's occurrence in a document is the mixture of the probability of its occurrence in different textual representations of the document (e.g., content, title, anchor text).

Later, Robertson et al. [14] showed that due to the different term occurrence distributions of the different representations of a document, it is better to combine frequencies rather than scores. Indeed, shortly thereafter, Zaragoza et al. [16] and Macdonald et al. [8] devised weighting models where the frequency of a term occurring in each of a document's representations is normalized and weighted before scoring by the weighting model. This allows a fine-grained control over the importance of each representation of the document in the document scoring process. This has been further investigated by the use of multinomial Divergence from Randomness models to score structured documents [12].

Key Applications

Web Search Engines are heavy developers and users of Web IR technology. Yet, to prevent exploitation by spammers and imitation by commercial rivals, the models applied by the search engines remain closely-guarded secrets. However, in recent years, the IR research field has grown, and many groups are now researching topics related to Web IR. The TREC forum, discussed below, is a facilitator of much of this research, providing samples of the Web for research purposes

Future Directions

There are many open problems in Web IR research. In particular, spam is an ever-growing issue – many sites are created with the purpose of manipulating search engine rankings for financial gain (e.g., advertising revenue). Search engines are in a constant battle with the spammers, and are constantly becoming more robust in adversial conditions.

Large Web search engines have often identified a huge number of features for every web page. Indeed, recent reports from the Microsoft Live search engine suggest that they use over 300 features when ranking web pages. At this scale, how should these features be combined? This area of interest has been recently gaining much interest from the IR and machine learning communities, and was the subject of the Learning To Rank for Information Retrieval workshop [5]. Generally, speaking, the idea is that machine learning techniques can be applied to learn a function from the input data and the evaluation results, instead of developing a function from a theory. The Learning to Rank sub-field also encompasses the large-scale learning that search engines perform using the click-through data gleaned from user sessions, allowing them to learn the best results for popular queries.

The advent of the blogging phenomenon has created many new and interesting search and data mining tasks for search engines in the era of user-generated content. The blogging community (known as the blogosphere) often responds to real-life events with comment and rhetoric. The TREC Blog track has been created to investigate retrieval issues in this niche area of the Web, and has thus far investigated the retrieval of opinionated posts – finding blog posts which not only discuss a target, but also disseminate an opinion about the target.

As Web search becomes ubiquitious for the average knowledge worker, the need for internal company search engines to allow employees to search and mine the company intranet are becoming important. To this end, the TREC Enterprise track has been investigating retrieval tasks in the Enteprise setting.

Data Sets

Much research into techniques for Web IR has centered around the Text REtrieval Conference (TREC) Web track and other related tracks. In particular, TREC has investigated retrieval from various samples of the Web, and produced several test collections. Each test collection consists of a corpus of Web documents crawled from the Internet, combined with queries with information needs (known as topics), and relevance assessments.

Table 1 introduces the years that Web tasks ran at TREC. The document corpora used are described further in Table 2. It is of note over the early years of the Web tracks, the tasks were being developed, and hence in the early years, the user tasks are not the most common task that users perform in search engines (e. g., Adhoc retrieval). Instead, the focus on early precision and known-item retrieval tasks (e.g., Home page and Named page finding) were developed from studies of user interactions with search engines [15].

Table 2 describes the collections used for the TREC Web and Terabyte tracks, over the years 1999–2006. When developing Web IR techniques, where it is appropriate, it is common to apply the techniques on these

WEB Information Retrieval Models. Table 1. Tasks and collections applied by various Web IR tracks at TREC

TREC Track	Collection	Tasks
2006 Terabyte	.GOV2	Adoc, Named page
2005 Terabyte	.GOV2	Adhoc, Named page
2004 Terabyte	.GOV2	Adhoc
2004 Web	.GOV	Mixed (Home page, Named page, Topic Distillation)
2003 Web	.GOV	Home page, Named page, Topic Distillation
2002 Web	.GOV	Named page, Topic Distillation
2001 Web	WT10G	Adhoc, Home page
2000 Web	WT10G	Adhoc
1999 Web	WT2G	Adhoc

WEB Information Retrieval Models. Table 2. Web IR research test collections

Collection	# Documents	# Links
.GOV2	25,205,179	261,937,150
.GOV	1,247,753	11,110,989
WT10G	1,692,096	8,063,026
WT2G	247,491	1,166,146

standard test collections, and compare to appropriately trained baselines. Hawking and Crasswell [4] contains an overview of the Web track at TREC. Relatedly, TREC also has two test collections for Enterprise IR research, and a further collection for Blog IR research.

URL to Code

Text REtrieval Conference: http://trec.nist.gov Web and Blog Test Collections: http://ir.dcs.gla.ac.uk/test_collections ACM Special Interest Group IR: http://www.sigir.org/

Cross-references

- ▶ BM25
- ▶ Divergence from Randomness Models
- ▶ Indexing the Web
- ▶ Information Retrieval Models
- ▶ Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model
- ▶ Web Indexing
- ▶ Web Page Quality Metrics
- ▶ Web Spam Detection

Recommended Reading

1. Brin S. and Page L. The anatomy of a large-scale hypertextual Web search engine. Comput. Netw. ISDN Syst., 30(1–7): 107–117, 1998.
2. Craswell N., Hawking D., and Robertson S. Effective site finding using link anchor information. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 250–257.
3. Craswell N., Robertson S., Zaragoza H., and Taylor M. Relevance weighting for query independent evidence. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 416–423.
4. Hawking D. and Craswell N. The very large collection and Web tracks. In TREC: Experiment and Evaluation in Information Retrieval. Kluwer Academic Publishers, Dordrecht, 2004, pp. 199–232.
5. Joachims T., Li H., Liu T.Y., and Zhai C. SIGIR workshop report: learning to rank for information retrieval (LR4IR 2007). SIGIR Forum, 41(2):55–62, 2007.
6. Kleinberg J.M. Authoritative sources in a hyperlinked environment. J. ACM, 46(5):604–632, 1999.
7. Kraaij W., Westerveld T., and Hiemstra D. The importance of prior probabilities for entry page search. In Proc. 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2002, pp. 27–34.
8. Macdonald C., Plachouras V., He B., Lioma C., and Ounis I. University of Glasgow at WebCLEF 2005: Experiments in per-field normlisation and language specific stemming. In Proc. 6th Workshop, Cross-Language Evaluation Forum, 2005, pp. 898–907.
9. Ogilvie P. and Callan J. Combining document representations for known-item search. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 143–150.
10. Peng J., Macdonald C., He B., and Ounis I. Combination of document priors in Web information retrieval. In Proc. 8th Int. Conf. Computer-Assisted Information Retrieval, 2007.
11. Plachouras V. Selective Web Information Retrieval. PhD thesis, Department of Computing Science, University of Glasgow, 2006.
12. Plachouras V. and Ounis I. Multinomial randomness models for retrieval with document fields. In Proc. 29th European Conf. on IR Research, 2007, pp. 28–39.
13. Plachouras V., Ounis I., and Amati G. The static absorbing model for the Web. J. Web Eng., 165–186, 2005.
14. Robertson S., Zaragoza H., and Taylor M. Simple BM25 extension to multiple weighted fields. In Proc. Int. Conf. on Information and Knowledge Management, 2004, pp. 42–49.
15. Silverstein C., Henzinger M., Marais H., and Moricz M. Analysis of a very large AltaVista Query Log. Technical Report 1998-014, Digital SRC, 1998.
16. Zaragoza H., Craswell N., Taylor M., Saria S., and Robertson S. Microsoft cambridge at TREC-13: Web and HARD tracks. In Proc. the 4th Text Retrieval Conf., 2004.

Web Macros

- ▶ Web Data Extraction System

Web Mining

- ▶ Data, Text, and Web Mining in Healthcare
- ▶ Languages for Web Data Extraction
- ▶ Web Harvesting

Web Mashups

MARISTELLA MATERA
Politecnico di Milano University, Milan, Italy

Synonyms

Composite Web applications

Definition

Web mashups are innovative Web applications, which rely on heterogeneous content and functions retrieved from external data sources to create new composite services.

Key Points

Web mashups characterize the second generation of Web applications, known as Web 2.0. They are composite applications, usually generated by combining content, presentation, or application functionality from disparate Web sources.

Typical components that may be mashed-up, i.e., composed, are RSS/Atom feeds, Web services, programmable APIs, and also content wrapped from third party Web sites. Components may have a proper user interface that can be reused to build the interface of the composite application, they may provide computing support, or they may just act as plain data sources. Content, presentation and functionality, as provided by the different components, are then combined in disparate ways: via JavaScript in the browser, via server-side scripting languages (like PHP), or via traditional languages like Java.

Cross-references

- ▶ Visual Interaction
- ▶ Web 2.0/3.0

Web Ontology Language

▶ OWL: Web Ontology Language

Web Page Quality Metrics

Ravi Kumar
Yahoo Research, Santa Clara, CA, USA

Synonyms

Link analysis

Definition

The primary mission of web search engines is to obtain the best possible results for a given user query. To accomplish this effectively, they rely on two crucial pieces of information: the relevance of a web page to the query and some aspect of the quality of the web page that is independent of the query. Relevance, the extent to which the query matches the content of the web page, is formalized and extensively studied in the field of information retrieval. Quality, on the other hand, is more nebulous and less well-defined. Nevertheless, one can identify three concrete and somewhat complementary aspects to the quality of a web page. The first is based on the absolute goodness of the web page and its associated meta-data. This might depend on a variety of parameters, including the worth of content that exists on the web page, the reputation of the person who authored the web page, the importance of the web site that hosts the web page, and so on. The second way is to focus on the quality of the web page as perceived by other pages that exist on the web. This can be captured by links that point to the web page: a hyperlink from one web page to another can be viewed as an endorsement of the latter by the former. The third way to measure quality is to focus on the how the web users perceive this page. This can be realized by studying the traffic on the web page. The main focus of this entry will be the second aspect of web page quality.

Historical Background

Web search engines were originally built using mainly classical information retrieval techniques, which were used to calculate the relevance of a query to a web page. The techniques were often modified in simple ways to account for the fact that most of the content on the web was written in HTML; for example, the presence of a query term in the title of the web page can connote a higher relevance. These methods served well in the early days of the web, but became increasingly inadequate as the web grew. One reason was they failed to take into account a distinctive feature of web pages: the presence of explicit hyperlinks created by people to link one document to another.

At a coarse level, the role of hyperlinks in web pages is two-fold. The first is to aid in the browsability and navigability from one web page to another on a web site. The second is to refer to outside sources of information that are considered relevant and authoritative by the author of the web page. Independently and almost concurrently, Brin and Page [4] and Kleinberg [12] came up with different methods to exploit the presence of links in order to ascribe a notion of quality to a web page, in the context of a collection of web pages with hyperlinks between them. The idea of using links between documents to infer a quality measure has precedents in the field of bibliometrics. For example, *impact factor* was proposed by Garfield [7] to measure the quality of scientific journals. It was defined as the average number of citations to the journal. *Bibliographic coupling* was proposed by Kessler [11] to measure similarity of two hyperlinked documents. It was defined as the total number of documents linked to by both documents. An alternate measure for document similarity was proposed by Small [15]: count of the total number of documents that have a link to both the documents.

The above measures treat all links alike and do not account for the quality of the citing document. A pervading theme in the works of Brin and Page and Kleinberg is to treat different links differently. This simple principle has proved to be very effective in practice in terms of improving web search and has found tremendous use in commercial search engines, even beyond web search.

Foundations

For the remainder of the entry, it is convenient to view the web as a directed graph or as a matrix. The web pages are nodes in this graph and the hyperlinks from one web page to another constitute the directed edges. This graph defines a natural adjacency matrix M whose (p,q)-th entry M_{pq} is 1 if and only if page p has a hyperlink to page q and 0 otherwise.

In 1998, Brin and Page [4] proposed a simple iterative method to define quality of a web page. Their method, called *PageRank*, used the hyperlinks to arrive at the quality. One way to think about this method is to focus on the following browsing behavior of a web user. Most of the times, the user visits a web page and after browsing, chooses one of the hyperlinks on the page at random to select the next page to browse. There is also a small chance that the user abandons browsing the current page entirely and chooses a random web page to continue browsing.

This behavior can be formalized as follows. Each page p has a non-negative value $\pi(p)$, updated iteratively. A stochastic process can be defined to abstract the user behavior, and in the limit, $\pi(p)$ will be the fraction of the time spent at page p by the process. Let $\alpha \in (0,1)$ be a fixed constant. At each step of the process, with probability α, the process jumps to a page chosen uniformly at random from the entire set of web pages. With the remaining probability $1 - \alpha$, the process jumps to a web page q chosen uniformly at random from the set of web pages to which page p has a hyperlink. If p has no hyperlinks, then the process jumps to a web page chosen uniformly at random from the entire set of pages. Observe that this process can be modeled by the iteration $\vec{\pi}_{t+1} = P^T\vec{\pi}_t$ with $P = (1 - \alpha) M' + \alpha J$. Here, M' is a stochastic matrix corresponding to the above process, J is the matrix where each entry is inverse of the total number of nodes in the graph, and α is the teleportation probability, usually chosen around 0.15. Notice that M' can be easily derived from the adjacency matrix M.

Since the above iteration corresponds to a linear system, it converges to a fixed-point $\vec{\pi} = \lim_{t\to\infty}\vec{\pi}_t$, which is the principal eigenvector of P^{T}. The quality of a page p is given by the value $\pi(p)$.

Many variants and extensions of the basic PageRank method have been proposed. A particularly interesting extension, called topic-sensitive PageRank, was proposed by Haveliwala [10]. The goal of this method is to define a quality measure that is biased with respect to a given topic. First, a subset of pages that correspond to the topic is identified; this can be done, for instance, by building a suitable classifier for the topic. Next, the basic PageRank process is modified in the following manner. Instead of jumping with probability α to a random page chosen from the entire set of web pages, the process jumps to a page chosen uniformly at random from this subset of pages. As before, the iteration converges and a quality measure for each page that is specific to this topic can be obtained.

Also in 1998, Kleinberg [12] proposed a different iterative method to define a quality of a web page. His method, called *HITS*, is aimed at finding the most authoritative sources in a given collection of web pages. The intuition behind HITS is the following. Each web page has two attributes, namely, how good is the page as an authority and how good is the page as a hub in terms of the quality of web pages to which it links. These two attributes mutually reinforce one another: the quality of a page as an authority is determined by the hub quality of pages that have a hyperlink to it and the quality of a page as a hub is determined by the authority quality of the pages to which it hyperlinks.

This reinforcement can be mathematically written in the following manner. Each page p has a hub value $h(p)$ and an authority value $a(p)$. Translating the above intuition into iterative equations, one can obtain $a_{t+1}(p) = \sum_{q|q\to p} M_{qp}h_t(q)$ and $h_{t+1}(p) = \sum_{q|p\to q} M_{pq}a_t(q)$. Since these iterations correspond to linear systems, they converge to their respective fixed points $\vec{a}$ and $\vec{h}$. It is easy to see that $\vec{a}$ is the principal eigenvector of M^TM and $\vec{h}$ is the principal eigenvector of MM^T. The quality of a web page p as an authoritative page is then given by its authority value $a(p)$.

There have been several modifications and extensions to Kleinberg's original work. For instance, [2,5,6,13]. Chakrabarti et al. [5] used the anchortext of a link to generalize the entries in *M* to [0,1]. Bharat and Henzinger [2] proposed many heuristics, including ones to address the issue of a page receiving unduly high authority value by virtue of many pages from the same web site pointing to it. Lempel and Moran [13] proposed a HITS-like method with slightly different definitions of authority and hub values.

For a comparative account of PageRank and HITS, the readers are referred to the survey by Borodin et al. [3].

Key Applications

As mentioned earlier, web page quality can be a vital input to the effectiveness of search engines. Besides this obvious application, ideas behind PageRank and HITS have been used to tackle many other problems on the web. Four such applications are mentioned below. Gyöngyi et al. [9] developed link analysis methods to identify spam web pages. Their method is based on identifying a small, reputable set of pages and using the hyperlinks to discover more pages that are likely to be good. Rafiei and Mendelzon [14] used random walks and PageRank to compute the topics for which a given web page has a good reputation. Bar-Yossef et al. [1] proposed a random walk with absorbing states to compute the decay value of a web page, where the decay value of a page measures how well-maintained and up-to-date is the page. Gibson et al. [8] used the HITS algorithm as a means to identifying web communities.

Cross-references

- ▶ Information Retrieval
- ▶ Web Data Management

Recommended Reading

1. Bar-Yossef Z., Broder A., Kumar R., and Tomkins A. Sic transit gloria telae: towards and understanding of the web's decay. In Proc. 12th Int. World Wide Web Conference, 2004, pp. 328–337.
2. Bharat K. and Henzinger M. Improved algorithms for topic distillation in a hyperlinked environment. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 104–111.
3. Borodin A., Roberts G.O., Rosenthal J.S., and Tsaparas P. Link analysis ranking algorithms, theory, and experiments. ACM Trans. Internet Tech., 5:231–297, 2005.
4. Brin S. and Page L. The anatomy of a large-scale hypertextual web search engine. Comput. Netw., 30:107–117, 1998.
5. Chakrabarti S., Dom B., Gibson D., Kleinberg J., Raghavan P., and Rajagopalan S. Automatic resource compilation by analyzing hyperlink structure and associated text. Comput. Netw., 30:65–74, 1998.
6. Chakrabarti S., Dom B., Gibson D., Kumar R., Raghavan P., Rajagopalan S., and Tomkins A. Spectral filtering for resource discovery. In Proc. ACM SIGIR Workshop on Hypertext Analysis. 1998, pp. 13–21.
7. Garfield E. Citation analysis as a tool in journal evaluation. Science, 178:471–479, 1972.
8. Gibson D., Kleinberg J., and Raghavan P. Inferring Web communities from link topology. In Proc. ACM Conference on Hypertext, 1998, pp. 225–234.
9. Gyöngyi Z., Garcia-Molina H., and Pedersen J. Combating web spam with TrustRank. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 576–587.
10. Haveliwala T.H. Topic-sensitive PageRank: A context-sensitive ranking algorithm for web search. IEEE Trans. Knowl. Data Eng., 15:784–796, 2003.
11. Kessler M.M. Bibliographic coupling between scientific papers. Am. Doc., 14:10–25, 1963.
12. Kleinberg J. Authoritative sources in a hyperlinked environment. J. ACM, 46:604–632, 2000.
13. Lempel R. and Moran S. SALSA: the stochastic approach for link-structure analysis. ACM Trans. Inform. Syst., 19:131–160, 2001.
14. Rafiei D. and Mendelzon A.O. What is this page known for? Computing web page reputations. Comput. Netw., 33:823–835, 2000.
15. Small H. Co-citaton in the scientific literature: a new measure of the relationship between two documents. J. Am. Soc. Inform. Sci., 24:265–269, 1973.

Web QA

▶ Web Question Answering

Web Query Languages

▶ Semantic Web Query Languages

Web Question Answering

CHARLES L. A. CLARKE
University of Waterloo, Waterloo, ON, Canada

Synonyms

Web QA

Definition

A question answering (QA) system returns exact answers to questions posed by users in natural language, together with evidence supporting those answers. A Web QA system maintains a corpus of Web pages and other Web resources in order to determine these answers and to provide the required evidence.

A basic QA system might support only simple factual (or "factoid") questions. For example, the user might pose the question

Q1. *What is the population of India?*

and receive the answer "1.2 billion," with evidence provided by the CIA World Factbook. A more advanced QA system might support more complex questions, seeking opinions or relationships between entities. Answering these complex questions might require the combination and integration of information from multiple sources. For example, the user might pose the question

Q2. *What methods are used to transport drugs from Mexico to the U.S.?*

and hope to receive a summary of information drawn from newspapers articles and similar documents. These two examples bracket the capabilities of current systems. Most QA systems can easily answer Q1, while questions such as Q2 represent an area of active research.

It is important to note that this definition of Web QA does not encompass Websites that allow users to answer each other's questions, nor sites that employ human experts to answer questions.

Historical Background

While experimental question answering systems have existed since the 1960's, research interest in question answering increased substantially in 1999, with the introduction of a question answering track at the Text REtreival Conference (TREC) [14,19]. TREC is an annual evaluation effort supervised by the US National Institute of Standards and Technology (NIST). Since its inception in 1991, TREC has conducted evaluation efforts for a wide range of information retrieval technologies, including question answering.

Inspired by the TREC QA track, Kwok et al. [8] developed one of the earliest Web QA systems. Their MULDER system answered questions by issuing queries to a commercial Web search engine and selecting answers from the pages it returned. They compared the performance of their system to that of several commercial search engines on the task of answering factoid questions taken from the TREC QA track.

Also inspired by TREC, Radev et al. [16] developed a method for converting questions into Web queries by applying techniques borrowed from statistical machine translation. Clarke et al. [5] crawled pages from the Web in order to gather information for answering trivia questions. Work by Dumais et al. [6] demonstrated the value of fully exploiting the volume of information available on the Web. Lam and Özsu [9] applied information extraction techniques to mine answers from multiple Web resources.

Much of the work on question answering was conducted outside the context of the Web. A general survey of this work is given by Prager [14]. An overview of many influential methods and systems is provided by Strzalkowski and Harabagiu [17].

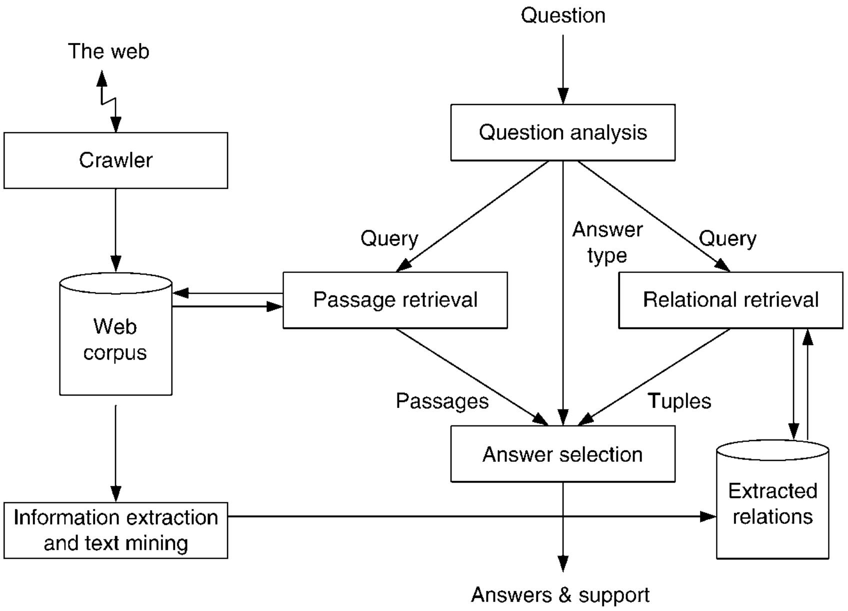
Foundations

Question answering is an extremely broad area, integrating aspects of natural language processing, machine learning, data mining and information retrieval. This entry focuses primarily on the use of Web resources for question answering.

Figure 1 provides a conceptual overview of the architecture of a basic QA system, showing the main components and processing steps. The system analyzes questions entered by the user, generates queries to collections of source material downloaded from the Web, and then selects appropriate answers from the resulting passages and tuples. The architecture shown in this figure emphasizes those components that are related to information retrieval and database systems, representing a composite derived from the published descriptions of a number of experimental systems [5,6,8,12,14,20]. The details of specific QA systems may differ substantially from this figure; an advanced QA system may include many additional components.

Sources for question answering are downloaded from the Web using a *crawler*, which is similar in construction and operation to the crawlers used by commercial Web search engines. After crawling, an inverted index is constructed, to allow the resulting collection to be searched efficiently.

Before construction of this inverted index, the crawler may annotate the collection to aid in question answering. For example, named entities, such as people, places, companies, quantities and dates, may be identified [15]. NLP techniques such as part-of-speech tagging and lightweight parsing may be applied. The creation of the index may also incorporate any of the standard indexing techniques associated with Web search, including link analysis.



Web Question Answering. Figure 1. Architecture of a Web question answering system.

Since the operation of a general Web crawler requires substantial effort and resources, many experimental QA systems substitute a commercial Web search engine for this crawling component [8]. To answer questions, one or more queries are issued to the search engine. Answers are then selected from the top documents returned by these queries.

As an additional post-processing step, distinct from crawling, the QA system may perform *information extraction and text mining* to identify events and relationships within the crawled pages. This extracted information, in the form of tuples, is used to populate tables of facts for reference by the QA system, essentially providing "prepared" answers for many factoid questions [2]. For example, associations between books and their authors, or between companies and their CEOs might be established in this way [1,4].

In addition, facts may be collected from pre-established sources that are known to be reliable. For example, current scores and schedules might be scraped from a sports Website; up-to-date movie and TV trivia might be taken from an entertainment site. Wrappers to collect this information might be constructed manually or learned from examples [7]. Since all answers should be appropriately supported by evidence, tuples containing extracted information should include links to the sources from which they are derived.

The result of these crawling and information extraction steps are two indexed collections. The first collection is a *Web corpus* of pre-processed Web pages, annotated to support question answering. The second collection is a database of *extracted relations*, providing pre-computed facts and answers. In its function and form, the first collection resembles a traditional information retrieval system; the second collection resembles a traditional relational database system.

These collections are prepared in advance of user queries. Answers are then determined by accessing and merging information from both collections. In a deployed QA system, these collections would be maintained on an on-going basis, much as an incremental crawler maintains the index of a commercial Web search engine.

After a user enters a question, the first stage in processing it is a *question analysis* step, in which an answer type is derived and queries to the retrieval components are constructed. The nature of the response required from the system will vary from question to question. The answer to a question might require a fact ("How many planets are there?"), a list ("What are the names of the planets?"), a yes-no response ("Is Pluto a planet?"), a definition ("What is a dwarf planet?"), or a description ("Why isn't Pluto a planet anymore?"). The role of the answer type is to convey the nature of the expected response to the answer selection component, providing a mapping of the question onto the domain supported by the system.

The depth and detail provided by these answer types varies substantially from system to system. Broadly, the answer type encodes constraints implied by the question, which must be satisfied by the answer. For example, the answer type associated with Q1 might be any of

- QUANTITY
- POPULATION
- POPULATION (COUNTRY ("India"))

where the notation in this example is strictly for illustrative purposes. Depending on the system, an answer type might be encoded in a wide variety of ways. The answer type associated with Q2, might be any of

- SNIPPET
- SUMMARY
- SUMMARY ("transport", "drugs", COUNTRY ("United States"), COUNTRY("Mexico"))

These examples merely suggest the range of possibilities for expressing an answer type, which may include enumerations, patterns, and trees. Since a QA system should ideally be able to handle any question, many systems will support a catch-all OTHER answer type.

In addition to the answer type, the question analysis component generates queries to the two retrieval components: the *passage retrieval* component, which searches the Web corpus, and the *relational retrieval* component, which searches the extracted relations. The passage retrieval component searches the Web corpus and returns passages that may contain the answer. At its simplest, a query to the passage retrieval component is a list of keywords and phrases, resembling a typical query to a Web search engine. For example, for Q1 the generated query might be "`population india`," and for Q2 it might be "`transport drugs mexico`". At its simplest, the relational component may be queried using SQL, returning tuples that may contain the answer. For Q1, the population of India could be retrieved from a table of country populations and passed to the answer selection component.

For more complex questions, the queries issued to the retrieval components will be correspondingly more complex. In some cases, multiple queries may be issued to either or both components, particularly when the answer requires more than a simple fact. For example, as part of processing Q2, the relational component might retrieve lists of illicit drugs, the names of drug cartels and their leaders, and geographic locations along the US/Mexican border. It is also possible for the retrieval components to associate quality or certainty scores with passages and tuples they return, and these scores may then be taken into consideration when selecting an answer.

Constraints implied by the answer type might also be incorporated into the query. For example, the query "`population india <quantity>`" includes an annotation tag indicating that a retrieved passage should contain a positive numeric value. The annotations generated during the creation of the Web corpus may allow the passage retrieval component to support structured queries that precisely specify the required content of an answer-bearing passage [3].

For question answering, passage retrieval is normally preferred over the document-oriented retrieval typical of other IR systems. In many cases, answers to questions are contained within a few sentences or paragraphs, and the use of passages reduces the processing load placed on the answer selection component. Even when an answer requires information from multiple sources, the information required from each source can often be found within a short passage.

Typically, passages are one or more sentences in length, and the passage retrieval component might return tens or hundreds of passages for analysis by the answer selection component. For Q1, the passage retrieval component might return passages such as:

- *Uttar Pradesh is not only the most populous state in India but is also one of the largest in area . . .*
- *The population of India grew by near 20% during the 1990's to more than 1.1 billion . . .*
- *India is home to roughly 1.2 billion people . . .*

Keywords associated with the query are underlined. As seen in this example, the terms in passages need not match the keywords exactly; matches against morphological variants and other related terms ("populous" and "people") may also be permitted. Passage retrieval algorithms for question answering usually treat proximity of query keywords as an important feature for retrieval. Good passages may contain most or all of the query keywords in close proximity [18].

Guided by the answer type, the *answer selection* component analyzes the passages and tuples returned by the retrieval components to generate answers.

Depending on the answer type, the output from this component may be a ranked list, a single best answer, or even a null answer, indicating that the system was unable to answer the question. In determining possible answers, the selection component may start by identifying a set of *candidate answers*. For example, given an answer type of QUANTITY and the list of passages above, the set of answer candidates might include "20%," "1990," "more than 1.1 billion" and "roughly 1.2 billion." If the answer type is more restrictive, the candidates "20%" and "1990" might be excluded from the set.

If a candidate answer appears in multiple passages, this repetition or *redundancy* may strengthen the support for that candidate. For example, the candidate answers "more than 1.1 billion" and "roughly 1.2 billion" provide redundant support for an answer of "1.2 billion." Redundancy is an important feature in many answer selection algorithms [5,6]. The answer selection component must balance the support provided by redundancy with other factors, in order to select its final answer.

In the leading research systems, answer selection is a complex process, and the development effort underlying this component may substantially outweigh the development effort underlying the rest of the system combined. In these advanced systems, the answer selection component may issue additional queries to the retrieval components as it tests hypotheses and evaluates candidate answers [13]. A discussion of advanced techniques for answer selection is beyond the scope of this entry. Additional information, and references to the literature, may be found in Prager [14].

Evaluation efforts have been undertaken annually by NIST since 1999. Each year, dozens of research groups from industry and academia participate in these experimental efforts. While these efforts do not specifically focus on question answering in a Web context, the evaluation methodologies developed as part of these efforts may be applied to evaluate Web QA systems.

Instead of a Web corpus, a test collection of newspaper articles and similar documents is provided by NIST, and this collection forms the source for answers and evidence. However, many of the participating groups augment this collection with Web pages and other resources in order to improve the performance of their systems [20]. These systems use a combination of all available resources to determine an answer, and then project this answer back onto the test collection to provide the required evidence.

While the details vary from year to year, the basic experimental structure remains the same. The test collection is distributed to participating groups well before the start of a formal experiment. At the start of an experiment, NIST distributes a test set of questions to each group. Participants are free to annotate the test collection and modify their systems prior to receiving this test set, but they must freeze the development of their systems once they receive it. Each group executes the questions using their QA system and returns the answers to NIST, along with supporting evidence.

Answers are manually judged by NIST assessors, where the evidence must fully support an answer for it to judged correct. The answers to factoid questions are judged on a binary basis (correct or not). For more complex questions, answers are evaluated against a list of information "nuggets," where each nugget represents a single piece of knowledge associated with the answer [11]. For example, the use of low-water crossings on the Rio Grande might form a single nugget in the answer to Q2. Answers are then scored on the basis of the nuggets they contain. While all judging is performed manually for the official NIST results, efforts have been made to construct automatic judging tools and re-usable collections, which allow QA system evaluation to take place outside the framework of the official experiments [10,11].

Key Applications

By providing exact answers, rather than ranked lists of Web pages, question answering has the potential to reduce the effort associated with finding information on the Web. Commercial search engines already incorporate basic question answering features. For example, most commercial search engines provide an exact answer in response to Q1. However, none currently treat Q2 as anything other than a keyword query. Bridging the gap between these questions would represent a major step in the evolution of Web search.

Data Sets

From 1999 to 2007, the evaluation of experimental QA system was conducted as part of TREC (http://trec.nist.gov). Starting in 2008, the evaluation of QA systems was incorporated into a new experimental conference series, the Text Analysis Conference (http://www.nist.gov/tac/). Test collections and other experimental data be found by visiting those sites.

Cross-references

- ▶ Indexing the Web
- ▶ Information Retrieval
- ▶ Inverted Files
- ▶ Snippet
- ▶ Text Mining
- ▶ Web Crawler Architecture
- ▶ Web Harvesting
- ▶ Web Information Extraction
- ▶ Web Search Relevance Ranking

Recommended Reading

1. Agichtein E. and Gravano L. Snowball: Extracting relations from large plain-text collections. In Proc. ACM International Conference on Digital Libraries. 2000, pp. 85–94.
2. Agichtein E. and Gravano L. Querying text databases for efficient information extraction. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 113–124.
3. Bilotti M.W., Ogilvie P., Callan J., and Nyberg E. Structured retrieval for question answering. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 351–358.
4. Brin S. Extracting patterns and relations from the World Wide Web. In Proc. Int. Workshop on the World Wide Web and Databases, 1998, pp. 172–183.
5. Clarke C.L.A., Cormack G.V., and Lynam T.R. Exploiting redundancy in question answering. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 358–365.
6. Dumais S., Banko M., Brill E., Lin J., and Ng A. Web question answering: Is more always better? In Proc. 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2002, pp. 291–298.
7. Kushmerick N., Weld D.S., and Doorenbos R.B. Wrapper induction for information extraction. In Proc. 15th Int. Joint Conf. on AI, 1997, pp. 729–737.
8. Kwok C., Etzioni O., and Weld D.S. Scaling question answering to the Web. ACM Trans. Inf. Syst., 19(3):242–262, 2001.
9. Lam S.K.S. and Özsu M.T. Querying Web data – the WebQA approach. In Proc. 3rd Int. Conf. on Web Information Systems Eng., 2002, pp. 139–148.
10. Lin J. and Katz B. Building a reusable test collection for question answering. J. Am. Soc. Inf. Sci. Technol., 57(7): 851–861, 2006.
11. Marton G. and Radul A. Nuggeteer: Automatic nugget-based evaluation using descriptions and judgements. In Proc. Human Language Technology Conf. of the North American Chapter of the Association of Computational Linguistics, 2006, pp. 375–382.
12. Narayanan S. and Harabagiu S. Question answering based on semantic structures. In Proc. 20th Int. Conf. on Computational linguistics, 2004, pp. 693–701.
13. Pasca M.A. and Harabagiu S.M. High performance question/answering. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 366–374.
14. Prager J. Open-domain question-answering. Found. Trends Inf. Retr., 1(2):91–231, 2006.
15. Prager J., Brown E., Coden A., and Radev D. Question-answering by predictive annotation. In Proc. 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2000, pp. 184–191.
16. Radev D.R., Qi H., Zheng Z., Blair-Goldensohn S., Zhang Z., Fan W., and Prager J. Mining the Web for answers to natural language questions. In Proc. Int. Conf. on Information and Knowledge Management, 2001, pp. 143–150.
17. Strzalkowski T. and Harabagiu S. (eds.). Advances in Open Domain Question Answering. Springer, Secaucus, NJ, USA, 2006.
18. Tellex S., Katz B., Lin J., Fernandes A., and Marton G. Quantitative evaluation of passage retrieval algorithms for question answering. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 41–47.
19. Voorhees E.M. Question answering in TREC. In TREC: Experiment and Evaluation in Information Retrieval, E.M. Voorhees and D.K. Harman (eds.). MIT, Cambridge, MA, USA, 2005, pp. 233–257.
20. Yang H., Chua T.-S., Wang S., and Koh C.-K. Structured use of external knowledge for event-based open domain question answering. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 33–40.

Web Resource Discovery

▶ Focused Web Crawling

Web Scraper

▶ Web Data Extraction System

Web Scraping

▶ Languages for Web Data Extraction

Web Search and Crawling

▶ Biomedical Scientific Textual Data Types and Processing

Web Search Engines

▶ WEB Information Retrieval Models

Web Search Query Rewriting

Rosie Jones[1], Fuchun Peng[2]
[1]Yahoo! Research, Burbank, CA, USA
[2]Yahoo! Inc., Sunnyvale, CA, USA

Synonyms

Query reformulation; Query expansion; Query assistance; Query suggestion

Definition

Query rewriting in Web search refers to the process of reformulating an original input query to a new query in order to achieve better search results. Reformulation includes but not limited to the following:

1. Adding additional terms to express the search intent more accurately
2. Deleting redundant terms or re-weighting the terms in the original query to emphasize important terms
3. Finding alternative morphological forms of words by stemming each word, and searching for the alternative forms as well
4. Finding synonyms of words, and searching for the synonyms as well
5. Fixing spelling errors and automatically searching for the corrected form or suggesting it in the results

Historical Background

Web search queries are the words users type into web search engines to express their information need. These queries are typically 2–3 words long [7]. Traditional information retrieval allows retrieval of documents that do not contain all of the query words. While early search engines such as Lycos and AltaVista also retrieved documents which contained a subset of the query terms, current search engines such as Yahoo, Google, and MSN typically require web documents to contain all of the query terms (with the exception of *stop-words* – words such as prepositions and conjunctions which do not have a large impact on overall topic meaning).

This is generally an appropriate behavior for web search, since the large number of documents available ensure that some documents will be found, and the overall task is then reduced to ranking those documents. It also greatly reduces the number of documents which must be considered.

A typographical error, such as a misspelling, or use of non-conventional terminology (such as colloquial rather than medical terminology), or use of an ambiguous term can lead the search engine to retrieve fewer or different documents than the full set of documents relevant to the user's intent. The tendency of web searchers to look just at the first 10–20 results [8] exacerbates the severity of this, since it is then even more important to have the most relevant documents at the top of the list.

Most web search engines offer interactive or automated *query rewriting*, to correct spelling, suggest synonyms for terms, and suggest related topics and subtopics the user might be interested in. In interactive query rewriting, the web searcher is shown one or more options for a spell correction or rephrasing of a query [1]. In automated query rewriting, a query is automatically modified to correct spelling, add terms or otherwise modify the original query, then used to retrieve documents without further interaction from the user [12]. Automatic query rewriting needs to be accurate in order to avoid query intent drifting. Hence, most query rewriting techniques are for interactive query reformulation.

The pre-cursors to query rewriting in web search are spell correction (first used in word processing), and relevance and pseudo-relevance feedback [3] in information retrieval.

Foundations

Stemming

Stemming is the process of merging words with different morphological forms, such as singular and plural nouns, and verbs in past and present tense. Stemming is a long studied technology. Many stemmers have been developed, such as the Lovins stemmer [11] and the Porter stemmer [13]. The Porter stemmer is widely used due to its simplicity and effectiveness in many applications. However, the Porter stemming makes many mistakes because its simple rules cannot fully describe English morphology. Corpus analysis is used to improve Porter stemmer [17] by creating equivalence classes for words that are morphologically similar and occur in similar context.

Traditionally, stemming has been applied to Information Retrieval tasks by transforming words in documents to the their root form before indexing, and applying a similar transformation to query terms. Although it increases recall, this naive strategy does not work well for Web Search since it lowers precision. Recently, a context-sensitive based on statistical machine translation [12] is successfully applied to Web search to improve both recall and precision, by considering the likelihood of different variants occurring in web pages. The most likely variants are used to expand the query, and both the original query, and the modified version can be used to retrieve web pages.

Spelling Correction

In Web search, spelling correction takes a user input query and provides a "corrected" form. Spelling correction is typically modeled as a *noisy channel process*. Separate models are built for character level errors (edit models which represent probabilities of character level insertions, substitutions and deletions), and overall word or word sequence probability (typically referred to as a language model) [2]. Web query logs can be used to improve the quality of models, by providing data for the language model, as well as the edit model [4]. Spell correction is commonly used as a query rewriting step before retrieving documents in web search, though the user is also sometimes consulted, with an interface asking "Did you mean ...?"

Recently, major search engines are taking one step further on spelling correction by directly retrieving results with the corrected query instead of asking users to click on "Did you mean ...?" suggestions. This significantly improves users experience as users may not notice that their original input query are misspelled. For example, a query "brittney spears" input to major search engines now can directly retrieve results for the correct form "Britney Spears." However, it is also quite risky as it largely depends on the quality of misspelling correction. If a wrong suggestion is used, the retrieve search results would be totally different from the original user intent. Thus, this implicit query rewriting has to be very conservative.

Query Expansion

Query expansion adds words to a query to better reflect the search intent. For example, query "palo alto real estate" may be expanded to "palo alto California real estate" if it is known that the user is interested in city of "Palo Alto" in California. The expanded word "California" can be inferred from many features, such as IP address, session analysis, geographic mapping information.

Another type of expansion is to expand an abbreviated form to its full form. For example, "HISD" can be expanded to "Houston Independent School District." This is referred as abbreviation or acronym rewriting. One word may be mapped to multiple expansions, for example, "abc" could mean "American border collie," "american baseball coaches," "American born Chinese" or "american broadcasting company." Depending on the query context, it can be expanded to the correct form [16].

Query expansion can be done on concept level instead of individual word level [6,14]. For example, "job hunting" can be expanded to "recruiting" or "employment opportunity." One way of extracting concepts from query is through query segmentation [15].

Query Substitution and Suggestion

Some queries can be reformulated with a totally different query through query substitution and suggestion. Approaches to query substitution and suggestion involve identifying pairs of queries for which there is evidence that their meaning is related. One approach looks at sequences of queries issued by web searchers as the reformulate and modify their queries [10]. Others identify pairs of queries as related if they lead to clicks on the same documents or retrieve similar documents [5]. For example, query "the biggest sports event this year" and query "olympic 2008" have the same intent (assuming one infers that *this year* in the first query is 2008).

Query Word Deletion

It is obvious that stop words can be dropped in most cases in search. Some non-stop words can also be dropped [9]. Query word deletion is not so popular as query expansion, as in most queries can have enough matched documents in the whole Web. Still, word deletion from query is important as it can help improving recall of tail queries (queries that are not very frequent).

Key Applications

Query rewriting and suggestion is useful for improving performance of web search, as well as more generally identifying related terms which can be used for other tasks.

Future Directions

One area of research for query rewriting is how to detect query intent drifting after rewriting. For example, stemming query "marching network" to "march networks," or "blue steel" to "blued steel" is bad since the query after rewriting has different intent of the input. How to model intent drifting is a challenging task in Web search.

Cross-references

- ▶ Query Expansion
- ▶ Query Expansion models
- ▶ Query Rewriting
- ▶ Web Search Relevance Feedback

Recommended Reading

1. Anick P. Using terminological feedback for Web search refinement – a log-based study. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 88–95.
2. Brill E. and Moore R.C. An improved error model for noisy channel spelling correction. In Proc. 38th Annual Meeting of the Assoc. for Computational Linguistics, 2000, pp. 86–293.
3. Croft W.B. and Harper D.J. Using probabilistic models of document retrieval without relevance information. J. Doc., 35 (4):285–295, 1979.
4. Cucerzan S. and Brill E. Spelling correction as an iterative process that exploits the collective knowledge of Web users. In Proc. Conf. on Empirical Methods in Natural Language Processing, 2004, pp. 293–300.
5. Cui H., Wen J.R., Nie J.Y., and Ma W.Y. Probabilistic query expansion using query logs. In Proc. 11th Int. World Wide Web Conference, 2002, pp. 325–332.
6. Fonseca B.M., Golgher P., Pssas B., Ribeiro-Neto B., and Ziviani N. Concept-based interactive query expansion. In Proc. 14th ACM Int. Conf. on Information and Knowledge Management, 2008, pp. 696–703.
7. Jansen B.J., Spink A., and Saracevic T. Real life, real users, and real needs: a study and analysis of user queries on the web. Inf. Process. Manage. Int. J., 36(2):207–227, 2000.
8. Joachims T., Granka L., Pang B., Hembrooke H., and Gay G. Accurately interpreting clickthrough data as implicit feedback. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 154–161.
9. Jones R. and Fain D. Query word deletion prediction. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 435–436.
10. Jones R., Rey B., Madani O., and Greiner W. Generating query substitutions. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 387–396.
11. Lovins J.B. Development of a stemming algorithm. Mech. Translat. Comput. Ling., 2:22–31, 1968.
12. Peng F., Ahmed N., Li X., and Lu Y. Context sensitive stemming for Web search. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 639–646.
13. Porter M.F. An algorithm for suffix stripping. Program, 14(3):130–137, 1980.
14. Qiu Y. and Frei H.P. Concept based query expansion. In Proc. 16th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 160–169.
15. Tan B. and Peng F. Unsupervised query segmentation using generative language models and wikipedia. In Proc. 17th Int. World Wide Web Conference, 2008, pp. 347–356.
16. Wei X., Peng F., and Dumoulin B. Analyzing Web text association to disambiguate abbreviation in queries. In Proc. 34th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2008, pp. 751–752.
17. Xu J. and Croft B. Query expansion using local and global document analysis. In Proc. 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1996, pp. 4–11.

Web Search Relevance Feedback

Hui Fang[1], ChengXiang Zhai[2]

[1]University of Delaware, Newark, DE, USA

[2]University of Illinois at Urbana-Champaign, Urbana, IL, USA

Definition

Relevance feedback refers to an interactive cycle that helps to improve the retrieval performance based on the relevance judgments provided by a user. Specifically, when a user issues a query to describe an information need, an information retrieval system would first return a set of initial results and then ask the user to judge whether some information items (typically documents or passages) are relevant or not. After that, the system would reformulate the query based on the collected feedback information, and return a set of retrieval results, which presumably would be better than the initial retrieval results. This procedure could be repeated.

Historical Background

Quality of retrieval results highly depends on how effective a user's query (usually a set of keywords) is in distinguishing relevant documents from non-relevant ones. Ideally, the keywords used in the query should occur only in the relevant documents and not in any non-relevant document. Unfortunately, in reality, it is often difficult for a user to come up with good keywords, mainly because the same concept can be described using different words and the user often has no clue about which words are actually used in the relevant documents of a collection. A main motivation for relevance feedback comes from the observation that although it may be difficult for a user to formulate a good query, it is often much easier for the user to judge whether a document or a passage is relevant.

Relevance feedback was first studied in the vector space model by Rocchio [8]. After that, many other relevance feedback methods have been proposed and studied in other retrieval models [6,15], including classical probabilistic models and language modeling approach. Although these methods are different, they all try to make the best use of the relevance judgments provided by the user and generally rely on some kind of learning mechanism to learn a more accurate representation of the user's information need. Relevance feedback has proven to be one of the most effective methods for improving retrieval performance [8,6,10].

Unfortunately, due to the extra effort that a user must make in relevance feedback, users are often reluctant to provide such explicit relevance feedback. When there are no explicit relevance judgments available, *pseudo feedback* may be performed, also known as *blind relevance feedback* [2,7]. In this method, a small number of top-ranked documents in the initial retrieval results are assumed to be relevant, and relevance feedback is then applied. This method also tends to improve performance on average (especially recall). However, pseudo feedback method does not always work well, especially for queries where none of the top N documents is relevant. The poor performance is expected because the non-relevant documents are assumed to be relevant, which causes the reformulated query to shift away from the original information need.

Somewhere inbetween relevance feedback and pseudo feedback is a technique called *implicit feedback* [5,4,11]. In implicit feedback, a user's actions in interacting with a system (e.g., clickthroughs) are used to infer the user's information need. For example, a result viewed by a user may be regarded as relevant (or more likely relevant than a result skipped by a user). Search logs serve as the primary data for implicit feedback; their availability has recently stimulated a lot of research in learning from search logs to improve retrieval accuracy (e.g., [3]).

Relevance feedback bears some similarity to query expansion in the sense that both methods attempt to select useful terms to expand the original query for improving retrieval performance. However, these two methods are not exactly same. First, query expansion does not necessarily assume the availability of relevance judgments. It focuses on identifying useful terms that could be used to further elaborate the user's information need. The terms can come from many different sources, not just feedback documents (often called local methods for query expansion). For example, they can also come from any document in the entire collection (called global methods) [14] or external resources such as a thesaurus. Second, while relevance feedback is often realized through query expansion, it can be otherwise. For example, when all available examples are non-relevant, other techniques than query expansion may be more appropriate [13].

Foundations

The basic procedure for relevance feedback includes the following steps. A user first issues a query, and the system returns a set of initial retrieval results. After returning the initial results, the user is asked to judge whether the presented information items (e.g., documents or passages) are relevant or non-relevant. Finally, the system revises the original query based on the collected relevance judgments typically by adding additional terms extracted from relevant documents, promoting weights of terms that occur often in relevant documents, but not so often in non-relevant documents, and down-weighting frequent terms in non-relevant documents. The revised query is then executed and a new set of results is returned. The procedure can be repeated more than once.

Technically, relevance feedback is a learning problem in which one learns from the relevant and non-relevant document examples how to distinguish new relevant documents from non-relevant documents. The learning problem can be cast as to learn either a binary classifier that can classify a document into relevant vs. non-relevant categories or a ranker that can rank relevant documents above non-relevant documents. Thus in principle, any standard supervised learning methods can be potentially applied to perform relevance feedback. However, as a learning problem, relevance feedback poses several special challenges, and as a result, a direct application of a standard supervised learning method is often not very effective. First, many supervised learning methods only work well when there are a relatively large number of training examples, but in the case of relevance feedback, the number of positive training examples is generally very small. Second, the training examples in relevance feedback are usually from the top-ranked documents, thus they are not a random sample and can be biased. Third, it is unclear how to handle the original query. The query may be treated as a special short example of relevant documents, but intuitively this special relevant example is much more important than other relevant examples and the terms in the query should be sufficiently emphasized to avoid drifting away from the original query concept. These challenges are exacerbated in the case of pseudo feedback as the query would be the only reliable relevance information provided by the user.

In the information retrieval community, many relevance feedback algorithms have been developed for different retrieval models. A commonly used feedback algorithm in vector space models is the *Rocchio algorithm* developed in early 1970s [8], which remains a robust effective state-of-the-art method for relevance feedback. In Rocchio, the problem of relevance feedback is defined as finding an optimal query to maximize its similarity to relevant documents and minimize its similarity to non-relevant documents. The revised query can be computed as

$$Q_r = \alpha \cdot Q_o + \frac{\beta}{|D_r|} \cdot \sum_{d_i \in D_r} d_i - \frac{\gamma}{|D_n|} \cdot \sum_{d_j \in D_n} d_j,$$

where Q_r is the revised query vector, Q_o is the original query vector, D_r and D_n are the sets of vectors for the known relevant and non-relevant documents, respectively. α, β, and γ are the parameters that control the contributions from the two sets of documents and the original query. Empirical results show that positive feedback is much more effective than negative feedback, so the optimal value of γ is often much smaller than that of β. A term would have a higher weight in the new query vector if it occurs frequently in relevant documents but infrequently in non-relevant documents. The IDF weighting further rewards a term that is rare in the whole collection.

The feedback method in classical probabilistic models is to select expanded terms primarily based on *Robertson/Sparck-Jones weight* [6] defined as follows:

$$w_i = log \frac{r_i/(R - r_i)}{(n_i - r_i)/(N - n_i - R + r_i)},$$

where w_i is the weight of term i, r_i is the number of relevant documents containing term i, n_i is the number of documents containing term i, R is the number of relevant documents in the collection, and N is the number of documents in the collection.

In language modeling approaches, feedback can be achieved through updating the query language model based on feedback information, leading to *model-based feedback methods*[15]. Relevance feedback received little attention in logical model, but some possible feedback methods have been discussed in [9].

Despite the difference in their way of performing relevance feedback, all these feedback methods implement the same intuition, which is to improve query representation by introducing and rewarding terms that occur frequently in relevant documents but infrequently in non-relevant documents. When optimized, most of these methods tend to perform similarly. Improvements over these basic feedback methods include (1) *passage feedback* [1] which can filter out the non-relevant part of a long relevant document in feedback, (2) *query zone* [12] which can improve the use of negative feedback information, and (3) *pure negative feedback* [13] which aims at exploiting purely negative information to improve performance for difficult topics.

Most studies focus on how to use the relevance judgments to improve the retrieval performance. The issue of choosing optimally documents to judge for relevance feedback has received considerably less attention. Recent studies on active feedback [11] have shown that choosing documents with more diversity for feedback is a better strategy than choosing the most relevant documents for feedback in the sense that the relevance feedback information collected with the former strategy is more useful for learning. However, this benefit may be at the price of sacrificing the utility of presented documents from a user's perspective.

Although relevance feedback improves retrieval accuracy, the improvement comes at the price of possibly slowing down the retrieval speed. Indeed, virtually all relevance feedback algorithms involve iterating over all the terms in all the judged examples, so the computational overhead is not negligible, making it a challenge to use relevance feedback in applications where response speed is critical.

Key Applications

Relevance feedback is a general effective technique for improving retrieval accuracy for all search engines, which is applicable when users are willing to provide explicit relevance judgments. Relevance feedback is particularly useful when it is difficult to completely and precisely specify an information need with just keywords (e.g., *multimedia search*); in such a case, relevance feedback can be exploited to learn features that can characterize the information need from the judged examples (e.g., useful non-textual features in multimedia search) and combine such features with the original query to improve retrieval results. Relevance feedback is also a key technique in *personalized search* where both explicit and implicit feedback information could be learned from all the collected information about a user to improve the current search results for the user.

Despite the fact that relevance feedback is a mature technology, it is not a popular feature provided by the current web search engines possibly because of the computational overhead. However, the feature "finding similar pages" provided by Google can be regarded as relevance feedback with just one relevant document.

Future Directions

Although relevance feedback has been studied for decades and many effective methods have been proposed, it is still unclear what is the optimal way of performing relevance feedback. The recent trend of applying machine learning to information retrieval, especially research on learning to rank and statistical language models, will likely shed light on the answer to this long-standing question.

So far, research on relevance feedback has focused on cases where at least several relevant examples can be obtained in the top-ranked documents. However, when a topic is difficult, a user may not see any relevant document in the top-ranked documents. In order to help such a user, negative feedback (i.e., relevance feedback with only non-relevant examples) must be performed. Traditionally negative feedback has not received much attention due to the fact that when relevant examples are available, negative feedback information tends not to be very useful. An important future research direction is to study how to exploit negative feedback to improve retrieval accuracy for difficult topics as argued in [13].

Active feedback [11] is another important topic worth further studying. Indeed, to minimize a user's effort in providing explicit feedback, it is important to select the most informative examples for a user to judge so that the system can learn most from the judged examples. It would be interesting to explore how to apply/adapt many active learning techniques developed in the machine learning community to perform active feedback.

Experimental Results

The effectiveness of a relevance feedback method is usually evaluated using the standard information retrieval evaluation methodology and test collections. A standard test collection includes a document collection, a set of queries and judgments indicating whether a document is relevant to a query. The initial retrieval results are compared with the feedback results to show whether feedback improves performance. The performance is often measured using Mean Average Precision (MAP) which reflects the overall ranking accuracy or precision at top k (e.g., top 10) documents which reflects how many relevant documents a user can expect to see in the top-ranked documents.

When comparing different relevance feedback methods, the amount of feedback information is often controlled so that every method uses the same judged documents. The judged or seen documents are usually excluded when computing the performance of a method to more accurately evaluate the performance of a method on *unseen* documents. The evaluation is significantly more challenging when the amount of feedback information can not be controlled, such as when different strategies for selecting documents for feedback are compared or a feedback method and a non-feedback method are compared. In such cases, it is tricky how to handle the judged documents. If they are not excluded when computing the performance of a method, the comparison would be unfair and favor a feedback that has received more judged examples. However, if they are excluded, the performance would not be comparable either because the test set used to compute the performance of each method would likely be different and may contain a different set of relevant and non-relevant documents.

Data Sets

Many standard information retrieval test collections can be found at: http://trec.nist.gov/

URL to Code

Lemur project contains the code for most existing relevance feedback method, which can be found at: http://lemurpoject.org/.

Cross-references

▶ Implicit Feedback
▶ Pseudo Feedback
▶ Query Expansion

Recommended Reading

1. Allan J. Relevance feedback with too much data. In Proc. 18th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 337–343.
2. Buckley C. Automatic query expansion using SMART: TREC-3. In Overview of the Third Text Retrieval Conference (TREC-3), D. Harman (ed.), 1995, pp. 69–80.
3. Burges C., Shaked T., Renshaw E., Lazier A., Deeds M., Hamilton N., and Hullender G. Learning to rank using gradient descent. In Proc. 22nd Int. Conf. on Machine Learning, 2005, pp. 89–96.
4. Joachims T. Optimizing search engines using clickthrough data. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 133–142.
5. Kelly D. and Teevan J. Implicit feedback for inferring user preference. SIGIR Forum, 37(2):18–28, 2003.
6. Robertson S.E. and Jones K.S. Relevance weighting of search terms. J. Am. Soc. Inf. Sci., 27(3):129–146, 1976.
7. Robertson S.E., Walker S., Jones S., Hancock-Beaulieu M.M., and Gatford M. Okapi at TREC-3. In Proc. The 3rd Text Retrieval Conference, 1995, pp. 109–126.
8. Rocchio J. Relevance feedback in information retrieval. In The SMART Retrieval System: Experiments in Automatic Document Processing. Prentice-Hall, Englewood Cliffs, NJ, 1971, pp. 313–323.
9. Ruthven I. and Lalmas M. A survey on the use of relevance feedback for information access system. Knowl. Eng. Rev., 18(2):95–145, 2003.
10. Salton G. and Buckley C. Improving retrieval performance by relevance feedback. J. Am. Soc. Inf. Sci., 44(4):288–297, 1990.
11. Shen X. and Zhai C. Active feedback in ad hoc information retrieval. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 59–66.
12. Singhal A., Mitra M., and Buckley C. Learning routing queries in a query zone. In Proc. 20th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1997, pp. 25–32.
13. Wang X., Fang H., and Zhai C. A study of methods for negative relevance feedback. In Proc. 34th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2008, pp. 219–226.
14. Xu J. and Croft W.B. Query expansion using local and global document analysis. In Proc. 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1996, pp. 4–11.
15. Zhai C. and Lafferty J. Model-based feedback in the language modeling approach to information retrieval. In Proc. Int. Conf. on Information and Knowledge Management, 2001, pp. 403–410.

Web Search Relevance Ranking

Hugo Zaragoza[1], Marc Najork[2]
[1]Yahoo! Research, Barcelona, Spain
[2]Microsoft Research, Mountain View, CA, USA

Synonyms

Ranking; Search ranking; Result ranking

Definition

Web search engines return lists of web pages sorted by the page's relevance to the user query. The problem with web search relevance ranking is to estimate relevance of a page to a query. Nowadays, commercial web-page search engines combine hundreds of features to estimate relevance. The specific features and their mode of combination are kept secret to fight spammers and competitors. Nevertheless, the main types of features at use, as well as the methods for their combination, are publicly known and are the subject of scientific investigation.

Historical Background

Information Retrieval (IR) Systems are the predecessors of Web and search engines. These systems were designed to retrieve documents in curated digital collections such as library abstracts, corporate documents, news, etc. Traditionally, IR relevance ranking algorithms were designed to obtain high recall on medium-sized document collections using long detailed queries. Furthermore, textual documents in these collections had little or no structure or hyperlinks. Web search engines incorporated many of the principles and algorithms of Information Retrieval Systems, but had to adapt and extend them to fit their needs.

Early Web Search engines such as Lycos and AltaVista concentrated on the scalability issues of running web search engines using traditional relevance ranking algorithms. Newer search engines, such as Google, exploited web-specific relevance features such as hyperlinks to obtain significant gains in quality. These measures were partly motivated by research in citation analysis carried out in the bibliometrics field.

Foundations

For most queries, there exist thousands of documents containing some or all of the terms in the query. A search engine needs to rank them in some appropriate way so that the first few results shown to the user will be the ones that are most pertinent to the user's need. The interest of a document with respect to the user query is referred to as "document relevance." this quantity is usually unknown and must be estimated from features of the document, the query, the user history or the web in general. Relevance ranking loosely refers to the different features and algorithms used to estimate the relevance of documents and to sort them appropriately.

The most basic retrieval function would be a Boolean query on the presence or absence of terms in documents. Given a query "word1 word2" the Boolean AND query would return all documents containing the terms word1 and word2 at least once. These documents are referred to as the query's "AND result set" and represent the set of potentially relevant documents; all documents not in this set could be considered irrelevant and ignored. This is usually the first step in web search relevance ranking. It greatly reduces the number of documents to be considered for ranking, but it does not rank the documents in the result set. For this, each document needs to be "scored", that is, the document's relevance needs to be estimated as a function of its relevance features. Contemporary search engines use hundreds of features. These features and their combination are kept secret to fight spam and competitors. Nevertheless, the general classes of employed features are publicly known and are the subject of scientific investigation. The main types of relevance features are described in the remainder of this section, roughly in order of importance. Note that some features are query-dependent and some are not. This is an important distinction because query-independent features are constant with respect to the user query and can be pre-computed off-line. Query-dependent features, on the other hand, need to be computed at search time or cached.

Textual Relevance

Modern web search engines include tens or hundreds of features which measure the textual relevance of a page. The most important of these features are matching functions which determine the term similarity to the query. Some of these matching functions depend only on the frequency of occurrence of query terms; others depend on the page structure, term positions, graphical layout, etc. In order to compare the query and the document, it is necessary to carry out some non-trivial preprocessing steps: tokenization (splitting the string into word units), letter case and spelling normalization, etc. Beyond these standard preprocessing steps, modern web search engines carry out more complex query reformulations which allow them to resolve acronyms, detect phrases, etc.

One of the earliest textual relevance features (earliest both in information retrieval systems and later in commercial web search engines) is the vector space model scoring function. This feature was used by early search engines. Since then other scoring models have been developed in Information Retrieval (e.g., Language Models and Probabilistic Relevance Models) [8] and have probably been adopted by web search engines. Although web search engines do not disclose details about their textual relevance features, it is known that they use a wide variety of them, ranging from simple word counts to complex nonlinear functions of the match frequencies in the document and in the collection.

Furthermore, web search engines make use of the relative and absolute position of the matches in the document. In Information Retrieval publications, there have been many different proposals to make use of term position information, but no consensus has been reached yet on the best way to use it. Most known approaches are based on features of the relative distances of the match terms such as the minimum (or average, or maximum) size of the text span containing all (or some, or most) of the term matches. Web search engines have not disclosed how they use position information.

Besides match position, web search engines exploit the structure or layout of documents, especially HTML documents. There are a number of ways to do this. One of the simplest is to compute textual similarity with respect to each document element (title, subtitles, paragraphs). More complex solutions integrate matches of different structural elements into a single textual relevance score (see for example [8]).

Another type of textual relevance information is provided by the overall document quality. For example, Web search engines use automatic document classifiers to detect specific document genres such as adult content, commercial sites, etc. Specialized techniques are also used to detect spam pages. Pages may be eliminated or demoted depending on the result of these analyses.

Hyperlink Relevance

The web is a hyperlinked collection of documents (unlike most previously existing digital collections, which had only implicit references). A hyperlink links a span of text in the source page (the "anchor text") to a target page (the "linked page"). Because of this, one can think of a hyperlink as a reference, an endorsement, or a vote by the source page on the target page. Similarly, one can think of the anchor text as a description or an explanation of the endorsement. One of the innovations introduced by Web Search Engines was leveraging the hyperlink-structure of the web for relevance ranking purposes.

The hyperlink graph structure can be used to determine the importance of a page independently on the textual content of the pages. This idea of using web hyperlinks as endorsements was originally proposed by Marchiori [9] and further explored by Kleinberg [6] and Page et al. [11] (who also introduced the idea of using the anchor text describing the hyperlink to augment the target page).

Exploiting User Behavior

As stated above, hyperlink analysis leverages human intelligence, namely peer endorsement between web page authors. Web search engines can also measure users' endorsements by observing the search result links that are being clicked on. This concept was first proposed by Boyan et al. [5], and subsequently commercialized by DirectHit [3]. For an up-to-date summary of the state of the art, the reader is referred to [5]. Besides search result clicks, commercial search engines can obtain statistics of page visitations (i.e., *popularity*) from browser toolbars, advertising networks or directly from Internet service providers. This form of quality feedback is query-independent and thus less informative but more abundant.

Performance

There are several aspects of the performance of a web relevance ranking algorithm. There are the standard algorithmic performance measures such as speed, disk and memory requirements, etc. Running time efficiency is crucial for web search ranking algorithms, since billions of documents need to be ranked in response to millions of queries per hour. For this reason most features need to be pre-computed off-line and only their combination is computed at query time. Some features may require specialized data structures to be retrieved especially fast at query time. This is the case for example of term-weights (which are organized in inverted indices [1]), or query-dependent hyperlink features [10].

A more fundamental aspect of the performance of a relevance ranking algorithm is its accuracy or precision: how good is the algorithm at estimating the relevance of pages? This is problematic because relevance is a subjective property, and can only be observed experimentally, asking a human subject. Furthermore, the performance of a ranking algorithm will not depend equally on each page: the best ranked pages are those seen by most users and therefore the most important to determine the quality of the algorithm in practice. Performance evaluation measures used for the development of relevance ranking algorithms take this into account [8].

There exist other, less explicit measures of performance. For example, as users interact with a search engine, the distribution of their clicks on the different ranks give an indication of the quality of the ranking (i.e., rankings leading to many clicks on the first results may be desired). However, this information is private to the search engines, and furthermore it is strongly biased by the order of presentation of results.

Feature Combination

All of the features of a page need to be combined to produce a single relevance score. Early web search engines had only a handful of features, and they were combined linearly, manually tuning their relative weights to maximize the performance obtained on a test set of queries. Modern search engines employ hundreds of features and use statistical methods to tune these features. Although the specific details remain secret, a number of research publications exist on the topic (see for example [13]).

Key Applications

The key application of web search relevance ranking is in the algorithmic search component of web search engines. Similar methods are also employed to bias the ranking of the advertisements displayed in search results. Some of the principles have been applied in other types of search engines such as corporate search (intranet, email, document archives, etc.).

Future Directions

Research continues to improve all of the relevance features discussed here. This research has lead to a continuous improvement of search engine quality. Nevertheless, current relevance features are becoming increasingly hard to improve upon. Considerable research is centered today on discovering new types of features which can significantly improve search quality. Only two of the most promising areas are mentioned here:

Query-understanding: Different types of queries may require very different types of relevance ranking algorithms. For example, a shopping query may require very different types of analysis from a travel or a health query. Work on algorithms that understand the intent of a query and select different relevance ranking methods accordingly could lead to dramatic increases in the quality of the ranking.

Personalization: In principle it is possible to exploit user information to "personalize" web search engine results. Different results would be relevant to a query issued by a layperson than a topic expert, for example. There are many different ways to personalize results: with respect to the user search history, with respect to the user community, with respect to questionnaires or external sources of knowledge about the user, etc. Many scientific papers have been written on this topic, but the problem remains unsolved. Commercial web search engines have mainly shied away from personalized algorithms. Google has proposed several forms of personalized search to its users, but this feature has not had much success. Nevertheless, the search continues for the right way to personalize relevance ranking.

Experimental Results

Evaluation of web relevance ranking is difficult and very costly, since it involves human judges labeling a collection of queries and results as to their relevance. The most careful evaluations of web relevance features are carried out by web search engine companies, but they are not disclosed to the public. There have been very many partial evaluations of search engines published, but they are always controversial due to their small scale, their experimental biases and their indirect access to the search engine features.

A number of experimental benchmarks have been constructed for public scientific competitions. Although they are small and partial they can be used for experimentation (see below).

Data Sets

Commercial search engines dispose of very large data sets comprising very many documents (e.g., hundreds of millions), queries (e.g., tens of thousands), and human relevance evaluations (e.g., hundreds of thousands). These data sets are routinely used to develop and improve features for relevance ranking. See [10,13] for examples of this.

Publicly available data sets for experimentation are very small compared to those used by commercial search engines. Nevertheless, they may be used to investigate some features and their combination. The most important datasets for web relevance ranking experiments are those developed in the Web-track of the TREC competition organized by NIST [4].

URL to Code

Due to the extraordinary cost of developing and maintaining a full-scale web search engine, there are no publically available systems so far. The Nutch project (http://lucene.apache.org/nutch/) is aiming to build an open-source web-scale search engine based on the Lucene search engine. Other retrieval engines capable of crawling and indexing up to several millions of documents include INDRI (http://www.lemurproject.org/indri/), MG4J (http://mg4j.dsi.unimi.it/) and TERRIER (http://ir.dcs.gla.ac.uk/terrier/).

Cross-references

- ▶ Anchor Text
- ▶ BM25
- ▶ Document Links and Hyperlinks
- ▶ Field-Based Information Retrieval Models
- ▶ Information Retrieval
- ▶ Language Models
- ▶ Relevance
- ▶ Relevance Feedback
- ▶ Text Categorization
- ▶ Text Indexing and Retrieval
- ▶ Vector-Space Model
- ▶ WEB Information Retrieval Models
- ▶ Web Page Quality Metrics
- ▶ Web Search Relevance Feedback
- ▶ Web Spam Detection

Recommended Reading

1. Baeza-Yates R. and Ribeiro-Neto B. Modern Information Retrieval. Addison Wesley, Reading, MA, 1999.
2. Boyan J., Freitag D., and Joachims T. A machine learning architecture for optimizing web search engines. In Proc. AAAI Workshop on Internet Based Information Systems, 1996.

3. Culliss G. The Direct Hit Popularity Engine Technology. A White Paper, DirectHit, 2000. Available online at https://www.uni-koblenz.de/FB4/Institutes/ICV/AGKrause/Teachings/SS07/DirectHit.pdf. Accessed on 27 Nov 2007.
4. Hawking D. and Craswell N. Very large scale retrieval and Web search. In TREC: Experiment and Evaluation in Information Retrieval, E. Voorhees and D. Harman (eds.). MIT Press, Cambridge, MA, 2005.
5. Joachims T. and Radlinski F. Search engines that learn from implicit feedback. IEEE Comp., 40(8):34–40, 2007.
6. Kleinberg J. Authoritative sources in a hyperlinked environment. Technical Report RJ 10076, IBM, 1997.
7. Langville A.N. and Meyer C.D. Google's PageRank and Beyond: The Science of Search Engine Rankings. Princeton University Press, Princeton, NJ, 2006.
8. Manning C.D., Raghavan P., and Schütze H. Introduction to Information Retrieval. Cambridge University Press, Cambridge, UK, 2008.
9. Marchiori M. The quest for correct information on the Web: hyper search engines. In Proc. 6th Int. World Wide Web Conference, 1997.
10. Najork M. Comparing the effectiveness of HITS and SALSA. In Proc. Conf. on Information and Knowledge Management, 2007, pp. 157–164.
11. Page L., Brin S., Motwani R., and Winograd T. The PageRank citation ranking: bringing order to the Web. Technical Report, Stanford Digital Library Technologies Project.
12. Richardson M., Prakash A., and Brill E. Beyond PageRank: machine learning for static ranking. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 707–715.
13. Taylor M., Zaragoza H., Craswell N., Robertson S., and Burges C. Optimisation methods for ranking functions with multiple parameters. In Proc. Conf. on Information and Knowledge Management, 2006, pp. 585–593.

Web Search Result Caching and Prefetching

Ronny Lempel[1], Fabrizio Silvestri[2]
[1]Yahoo! Research, Haifa, Israel
[2]ISTI-CNR, Pisa, Italy

Synonyms

Search engine caching and prefetching; Search engine query result caching; Paging in Web search engines

Definition

Caching is a well-known concept in systems with multiple tiers of storage. For simplicity, consider a system storing N objects in relatively slow memory, that also has a smaller but faster memory buffer of capacity k which can store copies of k of the N objects ($N >> k$). This fast memory buffer is called the *cache*. The storage system is presented with a continuous stream of queries, each requesting one of the N objects. If the object is stored in the cache, a *cache hit* occurs and the object is quickly retrieved. Otherwise, a *cache miss* occurs, and the object is retrieved from the slower memory. At this point, the storage system can opt to save the newly retrieved object in the cache. When the cache is full (i.e., already contains k objects), this entails *evicting* some currently cached object. Such decisions are handled by a *replacement policy*, whose goal is to maximize the *cache hit ratio* (or *rate*) – the proportion of queries resulting in cache hits.

Often, access patterns to objects, as found in query streams, are temporally correlated. For example, object y might often be requested shortly after object x has been requested. This motivates *prefetching* – the storage system can opt to retrieve and cache y upon encountering a query for x, anticipating the probable future query for y.

The setting with respect to search result caches in Web search engines is somewhat different. There, the objects to be cached are result pages of *search queries*. A search query is defined as a triplet $q = (qs, \mathit{from}, n)$ where qs is a query string, *from* denotes the relevance rank of the first result requested, and n denotes the number of requested results. The result page corresponding to q would contain the results whose relevance rank with respect to qs are $\mathit{from}, \mathit{from} + 1, \ldots, \mathit{from} + n - 1$. The value of n is typically 10. Search engines set aside some storage to cache such result pages. However, a search engine is not a typical two-tiered storage structure, as results not found in the cache are not stored in slower storage but rather need to be generated through the query evaluation process of the search engine. Prefetching of search results occurs when the engine computes and caches $j \cdot n$ results for the query (qs, from, n), with j being some small integer, in anticipation of follow-up queries requesting additional result pages for the same query string.

Historical Background

Markatos was the first to study search result caching in depth in 2000 [11]. He experimented with various known cache replacement policies on a log of queries submitted to the Excite search engine, and compared the resulting hit-ratios. In 2001 Saraiva et al. [12]

proposed a two-level caching scheme that combines caching of search results with the caching of frequently accessed postings lists.

Prefetching of search engine results was studied from a theoretical point of view by Lempel and Moran in 2002 [6]. They observed that the work involved in query evaluation scales in a sub-linear manner with the number of results computed by the search engine. Then they proceeded to minimize the computations involved in query evaluations by optimizing the number of results computed per query. The optimization is based on a workload function that models both (i) the computations performed by the search engine to produce search results and (ii) the probabilistic manner by which users advance through result pages in a *search session*.

In 2003, two papers proposed caching algorithms specifically tailored to the locality of reference present in search engine query streams: *PDC – Probability Driven Caching* [7], and *SDC – Static Dynamic Caching* [14] (extended version in [4]). The next section will focus primarily on those caching strategies and follow-up papers, e.g., the AC strategy proposed by Baeza-Yates et al. in [2].

In 2004, Lempel and Moran studied the problem of caching search engine results in the theoretical framework of competitive analysis [8]. For a certain stochastic model of search engine query streams, they showed an online caching algorithm whose expected number of cache misses is no worse than four times that of any online algorithm.

Note that search engine result caching is just one specific application of caching or paging; the theoretical analysis of page replacement policies dates back to 1966, when Belady [3] derived the optimal offline page replacement algorithm. In 1985, Sleator and Tarjan published their seminal paper on *online paging* [15], showing the optimal competitiveness of the Least Recently Used (LRU) policy.

Foundations

The setting for the caching of search results in search engines is as follows. The engine, in addition to its *index*, dedicates some fixed-size fast memory cache that can store up to *k* result pages. It is then presented with a stream of user-submitted search queries. For each query it consults the cache, and upon a cache hit returns the cached page of results to the user. Upon a cache miss, the query undergoes evaluation by the index, and the requested page of results (along with perhaps additional result pages) are computed. The requested results are returned to the user, and all newly computed result pages are forwarded to the cache replacement algorithm. In case the cache is not full, these pages will be cached. If the cache is full, the replacement algorithm may decide to evict some currently cached result pages to make room for the newly computed pages.

The primary goal of caching schemes (replacement policies and prefetching strategies) is to exploit the *locality of reference* that is present in many real life query streams in order to exhibit high hit ratios. Roughly speaking, locality of reference means that after an object x is requested, x and a small set of related objects are likely to be requested in the near future. Two types of locality of reference present in search engine query streams are detailed below.

Topical Locality of Reference

The variety of queries encountered by modern Web search engines is enormous. People submit queries on any and all walks of life, in all languages spoken around the world. Nevertheless, some query strings are much more popular than others and occur with high frequency, motivating the caching of their results by the search engine. Frequent queries can be roughly divided into two classes: (i) queries whose popularity is stable over time, and (ii) queries with bursty popularity, which may rise suddenly (e.g., due to some current event) and drop quickly soon thereafter.

Sequential Locality of Reference

This is due to the discrete manner in which search engines present search results to users – in batches of n (typically 10) results at a time. Some users viewing the page of results for the query $q = (qs, from, n)$ will not be fully satisfied, and will soon thereafter submit the follow-up query $(qs, from + n, n)$. In general, when many users have recently queried for $(qs, from, n)$, the higher the likelihood that at least one of them querying for $(qs, from + n, n)$. This behavior of users, coupled with the observation that the work involved in computing a batch of size $j \cdot n, j > 1$ results for a query string is significantly smaller than j independent computations of batches of n results, motivates the prefetching of search results in search engines [9].

Several studies have analyzed the manner in which users query search engines and view result pages [11,7, 13,1]. In general, it has been observed on numerous query logs that query popularities obey a power-law distribution, i.e., the number of query strings appearing n times in the log is proportional to n^{-c}. For example, Lempel and Moran [7] report a value of c equaling about 2.4 on a log of seven million queries submitted to AltaVista in late 2001 (The number of extremely popular queries typically exceeds the value predicted by the power law distribution.). While the 25 or so most popular queries in the query stream may account for over 1% of an engine's query load, a large fraction of any log is composed of queries that appear in it only once. For example, Baeza-Yates et al. [1] analyze a log in which the vast majority (almost 88%) of unique query strings were submitted just once, accounting for about 44% of the total query load. Such queries are bound to cause cache misses, and along with the misses incurred for the first occurrence of any repeating query imply an upper bound of about 50% for cache hit rates on that log.

Many of the studies cited above report that users typically browse through very few result pages of a query, and that the "deeper" the result page, the less users view it. While the exact numbers vary in each study, the reports agree that the majority of queries will have only their top-ten results viewed, and that the percentage of queries for which more than three result pages are viewed are very small. However, note that statistical data in the order of 10^{-2} (single percentage points) are powerful predictors for the purpose of caching result pages.

Beyond achieving high hit ratios, caching schemes must also be efficient, i.e., the logic involved in cache replacement decisions must be negligible as compared with the cost of a cache miss. Furthermore, they should be designed for high throughput, as search engines serve many queries concurrently and it would be counterproductive to have cache access become a bottleneck in their computation pipeline.

Different caching schemes have been applied to search results, ranging from "classic" ones to policies specifically designed for search engines workloads.

"Classic" Caching Schemes

These schemes include the well-known *Least Recently Used* (LRU) replacement policy and its variations (e.g., SLRU and LRU2) [15]. To quickly recap, upon a cache miss, LRU replaces the least recently accessed of the currently cached objects with the object that caused the cache miss. SLRU is a two-tiered LRU scheme: cache misses result in new objects entering the lower-tier LRU. Cache hits in the lower-tier LRU are promoted to the upper tier, whose least recently accessed object is relegated to be the most recently accessed object of the lower tier. The above replacement policies are very efficient, requiring O(1) time per replacement of an item. Furthermore, they are orthogonal to prefetching policies in the sense that one can decide to prefetch and cache any number of result pages following a cache miss.

PDC – Probability Driven Cache

In PDC the cache is divided between an SLRU segment that caches result pages for top-n queries (i.e., queries of the form (qs, $from$, n) where $from = 1$), and a priority queue that caches result pages of follow-up queries. The priority queue estimates the probability of each follow-up results page being queried in the near future by considering all queries issued recently by users, and estimating the probability of at least one user viewing each follow-up page. The priority queue's eviction policy is to remove the page least likely to be queried. One drawback of PDC is that maintaining its probabilistic estimations on the worthiness of each results page requires an amortized time that is logarithmic in the size of the cache per each processed query, regardless of whether the query caused a hit or a miss. An advantage of PDC is that it is better suited for prefetching than other schemes, since prefetched pages are treated separately and independently by the priority queue and are cached according to their individual worthiness rather than according to some fixed prefetching policy.

SDC – Static Dynamic Cache

SDC divides its cache into two areas: the first is a read-only (static) cache of results for the perpetually popular queries (as derived from some pre-analysis on the query log), while the second area dynamically caches results for the rest of the queries using any replacement policy (e.g., LRU or PDC).

Introducing a static cache to hold results for queries that remain popular over time has the following advantages:

- The results of these queries are not subject to eviction in the rare case that the query stream exhibits some "dry spell" with respect to them.
- The static portion of the cache is essentially read-only memory, meaning that multiple query threads

can access it simultaneously without the need to synchronize access or lock portions of the memory. This increases the cache's throughput in a real-life multithreaded system that serves multiple queries concurrently.

SDC can be applied with any prefetching strategy. See the discussion below for the specific prefetching strategy proposed in [14].

AC

AC was proposed by Baeza-Yates et al. in 2006 [2]. Like PDC and SDC, this scheme also divides its cache into two sections. The intuition behind AC is to identify the large fraction of rare "tail" queries and to relegate them to a separate portion of the cache, thus preventing them from causing the eviction of more worthy results.

Prefetching Policies

The prefetching policies appearing so far in the literature are rather simple. One straightforward policy is to simply choose some fixed amount k of result pages to prefetch upon any cache miss. The value of k can be optimized to maximize the cache hit ratio [7] or to minimize the total workload on the query evaluation servers [9]. These two objective functions do not necessarily result in the same value of k, since it could be that it is easier to compute more queries with fewer results per query than less queries with more results per query.

An adaptive prefetching scheme that also prefetches, in some cases, following a cache hit was proposed in [14]:

- Whenever a cache miss occurs for the top-n results of a query (i.e., when $from = 1$), evaluate the query and also prefetch its second page of results.
- For any cache miss of a follow-up query ($from > 1$), evaluate the query and prefetch some fixed number k of additional result pages.
- If the second page of results for a query is requested and is a cache hit, return the cached page and prefetch in the background the next k result pages.

Note that performing query evaluations following cache hits is a speculative operation that does not reduce the overall load on the backend in terms of query evaluations. It may balance that load somewhat and reduce the latency experienced by searchers when submitting follow-up queries.

As explained above, the choice of prefetching policy is orthogonal to the choice of replacement policy. However, PDC and AC are better suited to assigning different priorities to prefetched pages rather than treating them all as "recently used."

Note that prefetching is only effective when caching prefetched pages will result in more cache hits than the pages they replace in the cache. This depends on the characteristics of the cache (its size and replacement policy) as well as on the observed interaction of the users with the specific search engine in question (how often they tend to browse deep result pages).

Key Applications

Caching and Prefetching of results in Web search engines is a key application "*per se*."

Future Directions

Implementing search result caching in live, Web-scale systems brings with it many engineering challenges that have to do with the particular architecture of the specific search engine, and the specific nature of the application. This section briefly points at several such issues, each deserving of further research attention.

Search Result Caching in Incremental Search Engines

The previous sections assumed that the results of query q are stationary over time. In the case of evolving collections cached results may become stale, and serving them defeats the purpose of an incremental engine. Thus, more advanced cache maintenance policies are required.

Caching in the Presence of Personalized Search Results

Search engines are increasingly biasing search results toward the preferences of the individual searchers. As a consequence, two searchers submitting the same query might receive different search results. In this case, cache hit ratios decrease since results cannot always be reused across different searchers.

Level of Details to Cache

A page of search results is comprised of many different components (e.g., result URLs, summary snippets, ads), with each component logically coming from different computational processes and often from different sets of physical servers. While caching is meaningless without saving the URLs of the results, the other components can be computed separately, and whether to cache or recompute them is a matter of computational tradeoff.

Holistic View

There are many types of objects other than result pages that merit caching in a search engine, e.g., postings lists of popular query terms, user profiles, summary snippets, various dictionaries, and more [12,10]. The optimal allocation of RAM for the caching of different objects is a matter for holistic considerations that depend on the specific architecture of the search engine.

Experimental Results

The literature reports on many experiments with many query logs and caching schemes. All caching schemes have at least one tunable parameter – the cache size – whereas schemes with several cache segments (e.g., SLRU) have parameters governing the size of each segment. Search-specific caching schemes (PDC, SDC, AC) have additional tunable parameters, and when coupled with prefetching schemes, the number of reported experimental configurations grows significantly. Therefore, the summary below focuses on qualitative observations rather than quantitative ones. With respect to caching without prefetching, the literature finds that:

- Even generic caching schemes attain decent hit ratios. For example, LRU achieves on some query logs hit ratios as high as 80% of what an infinite cache would achieve. In terms of absolute values, hit ratios of 20–30% are common for LRU based schemes. In fact, recency-based caching schemes are well suited for the large topical locality of reference exhibited by power-law query distributions, where significant portions of the query stream are comprised of a small set of highly popular queries.
- Search-specific caching schemes outperform generic caching schemes. However, the relative gains in terms of hit ratio are rarely higher than 10%.

Throughout the literature, prefetching improved the achieved cache hit ratios, in many cases by 50% and more as compared with the same experimental setup without prefetching. The following are additional observations:

- The largest relative gains in hit ratio are achieved for moderate amounts of prefetching, namely fetching one or two result pages beyond what is requested in the query. More aggressive prefetching improves the hit ratio by modest amounts at best, and in small caches may even cause it to deteriorate.
- As a general rule, larger caches merit more prefetching, since much of a large cache is devoted to low-merit result pages, which are less likely to be queried than follow-up result pages of current queries.
- Not surprisingly, the adaptive prefetching scheme proposed in [14] outperforms the simplistic policy of always prefetching the same amount of result pages regardless of the identity of the page that generated the cache miss.

Cross-references

▶ Buffer Management
▶ Cache Replacement as Trade-Off Elimination
▶ Cache-Conscious Query Processing
▶ Competitive Ratio
▶ Indexing the Web
▶ Inverted Files

Recommended Reading

1. Baeza-Yates R., Gionis A., Junqueira F., Murdock V., Plachouras V., and Silvestri F. The impact of caching on search engines. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 183–190.
2. Baeza-Yates R., Junqueira F., Plachouras V., and Witschel H.F. Admission policies for caches of search engine results. In Proc. 14th Int. Symp. String Processing and Information Retrieval, 2007, pp. 74–85.
3. Belady L.A. A study of replacement algorithms for a virtual-storage computer. IBM Syst. J., 5(2):78–101, 1966.
4. Fagni T., Perego R., Silvestri F., and Orlando S. Boosting the performance of web search engines: caching and prefetching query results by exploiting historical usage data. ACM Trans. Inf. Syst., 24(1):51–78, 2006.
5. Karedla R., Love J.S., and Wherry B.G. Caching strategies to improve disk system performance. Computer, 27(3):38–46, 1994.
6. Lempel R. and Moran S. Optimizing result prefetching in web search engines with segmented indices. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 370–381.
7. Lempel R. and Moran S. Predictive caching and prefetching of query results in search engines. In Proc. 12th Int. World Wide Web Conference, 2003, pp. 19–28.
8. Lempel R. and Moran S. Competitive caching of query results in search engines. Theor. Comput. Sci., 324(2):253–271, 2004.
9. Lempel R. and Moran S. Optimizing result prefetching in web search engines with segmented indices. ACM Trans. Internet Tech., 4:31–59, 2004.
10. Long X. and Suel T. Three-level caching for efficient query processing in large web search engines. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 257–266.

11. Markatos E.P. On caching search engine query results. Comput. Commun., 24(2):137–143, 2001.
12. Saraiva P., Moura E., Ziviani N., Meira W., Fonseca R., and Ribeiro-Neto B. Rank-preserving two-level caching for scalable search engines. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 51–58.
13. Silverstein C., Henzinger M., Marais H., and Moricz M. Analysis of a very large altavista query log. Technical Report 1998-014, Compaq Systems Research Center, October 1998.
14. Silvestri F., Fagni T., Orlando S., Palmerini P., and Perego R. A hybrid strategy for caching web search engine results. In Proc. 12th Int. World Wide Web Conference, 2003 (Poster).
15. Sleator D.D. and Tarjan R.E. Amortized efficiency of list update and paging rules. Commun. ACM, 28:202–208, 1985.

Web Search Result De-duplication and Clustering

Xuehua Shen[1], ChengXiang Zhai[2]
[1]Google, Inc., Mountain View, CA, USA
[2]University of lllinois at Urbana-Champaign, Urbana, IL, USA

Definition

Web search result de-duplication and clustering are both techniques for improving the organization and presentation of Web search results. De-duplication refers to the removal of duplicate or near-duplicate web pages in the search result page. Since a user is not likely interested in seeing redundant information, de-duplication can help improve search results by decreasing the redundancy and increasing the diversity among search results.

Web search result clustering means that given a set of web search results, the search engine partitions them into subsets (clusters) according to the similarity between search results and presents the results in a structured way. Clustering results helps improve the organization of search results because similar pages will be grouped together in a cluster and a user can easily navigate into the most relevant cluster to find relevant pages. Hierarchical clustering is often used to generate a hierarchical tree structure which facilitates navigation into different levels of clusters.

De-duplication can be regarded as a special way of clustering search results in which only highly similar (i.e., near-duplicate) pages would be grouped into a cluster and only one page in a cluster is presented to the user. Although standard clustering methods can also be applied to do de-duplication, de-duplication is more often done with its own techniques which are more efficient than clustering.

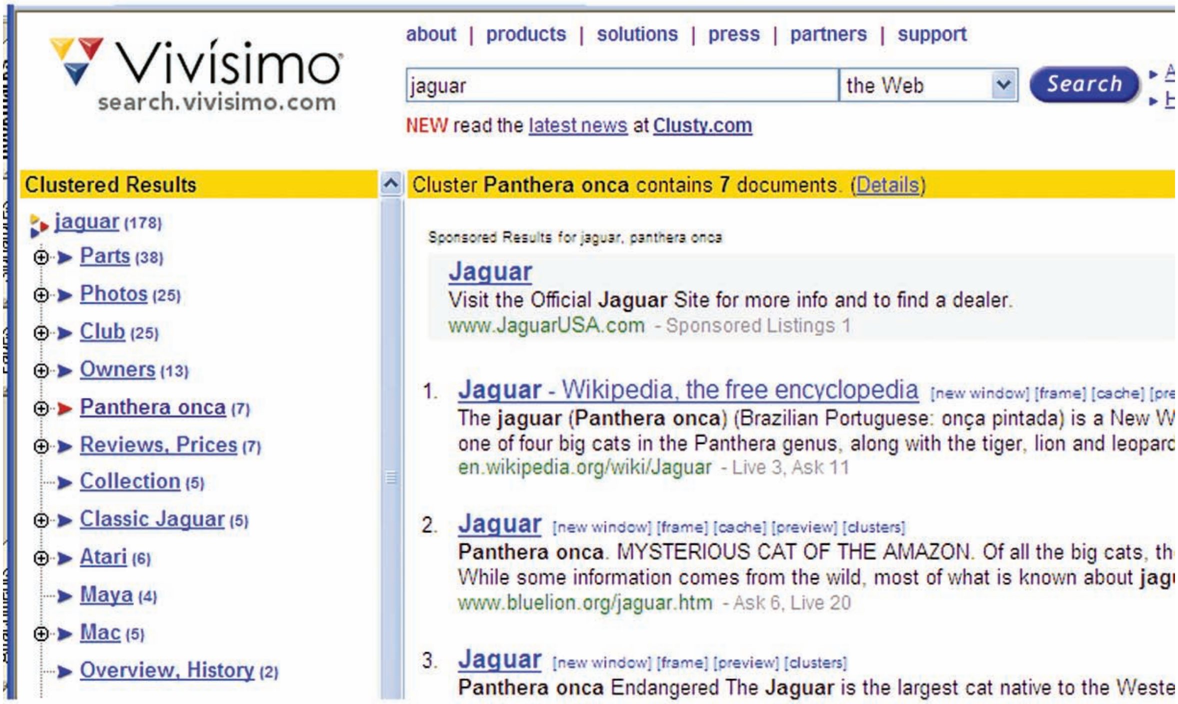
For example, Fig. 1 shows a screenshot of the clustered search results of the query "jaguar" from the vivisimo meta-search engine (http://www.vivisimo.com), which clusters web search results and presents a hierarchical tree of search results. On the left are some clusters labeled with phrases; each cluster captures some aspect of the search results, and different senses of "jaguar" (e.g., jaguar animals vs. jaguar cars) have been separated. On the right are the documents in the cluster labeled as "Panthora onca," which are displayed as the user clicks on this cluster on the left panel.

Historical Background

The idea of clustering search results appears to be first proposed in [6]. However, the application of document clustering to improve the performance of information retrieval has a much longer history [14], dating back to at least 1971 [9]. Most of this work has been motivated by the so-called cluster hypothesis, which says that closely associated documents tend to be relevant to the same requests [9]. The cluster hypothesis suggests that clustering can be potentially exploited to improve both search accuracy (e.g., similar documents can reinforce each other's relevance score) and search efficiency (e.g., scoring clusters takes less time than scoring all the documents in the whole collection). Research has been done in both directions with mixed findings.

In [3], it was proposed that clustering can be used to facilitate document browsing, and a document browsing method called Scatter/Gather was proposed to allow a user to navigate in a collection of documents. In [6], Scatter/Gather was further applied to cluster search results and was shown to be more effective in helping a user find relevant documents than simply presenting results as a ranked list.

Clustering web search results appears to be first studied in [15], which shows that it is feasible to cluster web search results on-the-fly. Vivisimo (http://www.vivisimo.com/) is among the first Web search engines to adopt search result clustering. More recent work in this line has explored presenting search results in clusters (categories) defined based on context (e.g., [4,13]), hierarchical clustering (e.g., [5]), efficiency



jaguar | the Web

NEW read the latest news at Clusty.com

Cluster Panthera onca contains 7 documents. (Details)

Sponsored Results for jaguar, panthera onca

Jaguar
Visit the Official Jaguar Site for more info and to find a dealer.
www.JaguarUSA.com - Sponsored Listings 1

1. Jaguar - Wikipedia, the free encyclopedia [new window] [frame] [cache] [pre
The jaguar (Panthera onca) (Brazilian Portuguese: onça pintada) is a New W
one of four big cats in the Panthera genus, along with the tiger, lion and leoparc
en.wikipedia.org/wiki/Jaguar - Live 3, Ask 11

2. Jaguar [new window] [frame] [cache] [preview] [clusters]
Panthera onca. MYSTERIOUS CAT OF THE AMAZON. Of all the big cats, th
While some information comes from the wild, most of what is known about jag
www.bluelion.org/jaguar.htm - Ask 6, Live 20

3. Jaguar [new window] [frame] [preview] [clusters]
Panthera onca Endangered The Jaguar is the largest cat native to the Weste

Web Search Result De-duplication and Clustering. Figure 1. Sample clustered results about "jaguar" from Vivisimo search engine.

of clustering, incorporating the prior knowledge or user feedback into clustering algorithms in a semi-supervised manner.

Compared with clustering, document de-duplication has a relatively short history. The issue of document duplication was not paid much attention to by information retrieval researchers in early days, probably because it was not a serious issue in the traditional applications of retrieval techniques (e.g., library systems). However, the problem of detecting duplicate or near-duplicate documents has been addressed in some other contexts before it was studied in the context of the Web. For example, the problem has naturally occurred in plagiarism detection. The problem also arose in file systems [10], where one document could be easily copied, slightly modified, saved as another file, or transformed into another format, leading to duplicate or near-duplicate documents. The SCAM system [12] is among the early systems for detecting duplicate or near-duplicate documents. The algorithms proposed in these studies are often based on signatures – representing and comparing documents based on their signatures, or values generated by a hashing function based on substrings from documents.

Web page de-duplication was first seriously examined in [1], where the authors pointed out that document duplication arises in two ways on the Web: one is that the same document is placed in several places, while the other is that the same document are in almost identical incarnations. The authors further proposed an effective de-duplication algorithm based on representing and matching documents based on shingles (i.e., contiguous subsequences of tokens in a documents). Now it has been realized that Web page de-duplication is needed at stages of indexing, ranking, and evaluation [8] to decrease the index storage, increase user satisfaction level, and improve ranking algorithm, respectively.

Foundations

Since de-duplication can be regarded as a special case of clustering, it is not surprising that de-duplication and clustering share very similar fundamental challenges, which include (i) designing an appropriate representation of a document; (ii) designing an effective measure of document similarity based on document representation; (iii) finding/grouping similar documents (duplicate or near-duplicate documents in the case of de-duplication) quickly. Most research work attempts to solve all or some of these challenges.

How to represent a document and how to measure similarity between documents are closely related to each other in that a similarity function making sense for one representation may not work for another representation. Both highly depend on how the notion of similarity is defined. This is a non-trivial issue as two documents may be similar in many different ways (e.g., in topic coverage, in genre, or in structure), and depending on the desired perspective of similarity, one may need different representations of documents and different ways are needed to measure similarity. As a result, the optimal choice often depends on the specific applications.

For de-duplication, it often suffices to represent documents primarily based on simple tokenization of documents or even surface string representation of documents since duplicate or near-duplicate documents generally share a lot of surface features and may contain very long identical strings. For the same reason, exact matching of two representations may also be sufficient most of the time. Most proposed methods for detecting duplicate or near-duplicate documents can be roughly classified into two categories: signature matching and content matching. They mainly differ in how a document is represented. In signature matching, a document is represented by a signature of the document which is often generated using a hash function based on some (not necessarily meaningful) subsequences of tokens in a document, thus the similarity between documents is mostly measured based on syntactic features of documents. In content matching, the basic unit of document representation is often a meaning unit such as a word or phrase intended to capture the goal of content-matching. Moreover, the basic units are often weighted to indicate how well they reflect the content of a document using heuristic weighting methods such as TF-IDF weighting proposed in information retrieval. Since document representation reflects more the content of a document in the content matching approach, the similarity between documents reflects their semantic similarity better than in the fingerprint matching approach. In general, content matching captures more semantics in de-duplication but is generally more expensive than signature matching. A comprehensive evaluation of these methods can be found in [7].

The shingling algorithm proposed in [1] remains a major state-of-the-art method for Web page de-duplication. In this algorithm, every *k* subsequence of tokens of a web page is used as a unit of representation (called a shingle), and a page is represented by the set of unique shingles occurring in the page. The similarity between two web pages is measured by the percentage of unique shingles shared by the two pages. Sampling can be used to estimate this similarity and improve efficiency of de-duplication.

In the I-Match method [2], collection statistics is used to filter terms in a signature matching approach, thus achieving content-matching to a certain extent. In a recent work [8], it is shown that additional content-based features can be exploited to significantly improve de-duplication accuracy.

Clustering of search results is almost exclusively based on content matching rather than signature matching because the goal is to group together semantically similar pages which are not always syntactically similar. A document is often represented by a content-capturing feature vector. To construct such a vector, a set of features are selected and assigned appropriate weights to them. Many features may be adopted for clustering, including, e.g., terms from the full content of the html web page, web page snippet (title and summary), URL of the web page, and anchor text. These features need to be weighted appropriately to reflect the content of a page accurately. These challenges are essentially the same as in text retrieval where there is a need to represent a document appropriately to match a document with a query accurately. In general, document similarity can be computed in a similar way to computing the similarity between a query and a document in a standard retrieval setting. Thus many methods developed in information retrieval, especially TF-IDF term weighting and language models, can applied to clustering search results. The basic idea of TF-IDF weighting is to give a term a higher weight if the term occurs frequently in a document and occurs infrequently in the entire collection of documents. Similarity is often computed based on the dot product or its normalized form (i.e., cosine) of two TF-IDF vectors. Language models achieve a similar weighting heuristic through probabilistic models. Once similarities between documents are computed, many clustering methods can be applied, such as k-means, nearest neighbor, hierarchical agglomerative clustering, and spectral clustering. Instead of relying on an explicit similarity function, a different strategy for clustering is to cast the clustering problem as fitting a mixture model to the text data; the estimated parameter values can then be used to easily obtain clusters. Methods of this kind are called model-based clustering methods. Finite multinomial mixture, probabilistic latent semantic indexing (PLSA), and latent Dirichlet allocation (LDA) are some examples in this category. In model-based clustering, the notion of similarity is implied by the model. Due to the lack of systematic comparisons among these different methods, it is unclear which clustering method is the best for clustering search results.

Efficiency is a critical issue for clustering search results since a search engine must do that on-the-fly after the user submits a particular query. Usually clustering is applied after the search engine retrieves a ranked list of web pages. Thus the time spent on clustering is in addition to the original latency of web search. If the latency is noticeable, the user may be discouraged from using a clustering interface. Some clustering algorithms such as k-means and some model-based algorithms are "any time" algorithms in that they are iterative and can stop at any iteration to produce (less optimal) clustering results. These algorithms would thus be more appropriate for clustering search results when efficiency is a concern. Speeding up clustering is an active research topic in data mining, and many efficient algorithms proposed in data mining can be potentially applied to clustering search results.

Another important challenge in clustering search results is to label a cluster appropriately, which clearly would directly affect the usefulness of clustering, but is yet under-addressed in research. A good cluster label should be understandable to the user, capturing the meaning of the cluster, and distinguishing the cluster from other clusters. In [11], a probabilistic method is proposed to automatically label a cluster with phrases. Another approach to improving cluster labeling and clustering of results in general is to learn from search logs to discover interesting aspects of a query and cluster the search results according to these aspects which can then be naturally labeled with some typical past queries [13].

Key Applications

The major application of search results de-duplication is to reduce the redundancy in search results, which presumably improve the utility of search results. Document de-duplication, however, has many other applications in Web search. For example, it can reduce the size of storage for index, and allow a search engine to load index of more diverse web page sets to the memory. Eliminating duplicate or near-duplicate documents can also reduce the noise in the document set for evaluation, and thus would allow an evaluation measure to reflect more accurately the real utility of a search algorithm [8]. Besides applications for search engines, de-duplication also has many other applications such as plagiarism detection and tracking dynamic changes of the same web page.

Clustering search results generally help a user navigate in the set of search results. It is expected to be most useful when the search results are poor or diverse (e.g., when the query is ambiguous) or when the user has a high-recall information need. Indeed, when the user has a high-precision information need (e.g., navigational queries) and the search results are reasonably accurate, presenting a ranked list of results may be the best and clustering may not be helpful. However, due to the inevitable mismatches between a query and web pages, the search results are more often non-optimal. When the search results are poor, it would take a user a lot of time to locate relevant documents from a ranked list of documents, so clustering is most likely helpful because it would enable a user to navigate into the relevant documents quickly. In the case of poor results due to ambiguity of query words, clustering can be expected to be most useful. When the user has a high-recall information need, clustering can also help the user quickly locate many additional relevant documents once the user identifies a relevant cluster. In addition, the overview of search results generated from clustering may be useful information itself from a user's perspective. Data visualization techniques can be exploited to visualize the clustering presentation as in Kartoo (http://www.kartoo.com). Appropriate visualization may increase the usefulness of clustering search results.

Clustering search results, when used in an iterative way, can help a user browse a web pages collection. In Scatter/Gather [3], for example, the user would interact with an information system by selecting interesting clusters, and the information system would do clustering again after the user selects some clusters. Thus clustering search results can effectively support a user to interactively access information in a collection.

URL to Code

Lemur (http://www.lemurproject.org/). Lemur is an open-source toolkit designed to facilitate research in information retrieval. Document clustering is included.

CLUTO(http://glaros.dtc.umn.edu/gkhome/views/cluto/). CLUTO is a software package for clustering low- and high-dimensional datasets and for analyzing the characteristics of clusters.

Cross-references

▶ Data Clustering
▶ Data Visualization

Recommended Reading

1. Broder A.Z., Glassman S.C., Manasse M.S., and Zweig G. Syntactic clustering of the web. Comput. Networks, 29(8–13): 1157–1166, 1997.
2. Chowdhury A., Frieder O., Grossman D.A., and McCabe M.C. Collection statistics for fast duplicate document detection. ACM Trans. Inf. Syst., 20(2):171–191, 2002.
3. Cutting D.R., Pedersen J.O., Karger D., and Tukey J.W. Scatter/gather: a cluster-based approach to browsing large document collections. In Proc. 15th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1992, pp. 318–329.
4. Dumais S.T., Cutrell E., and Chen H. Optimizing search by showing results in context. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2001, pp. 277–284.
5. Ferragina P. and Gulli A. A personalized search engine based on Web-snippet hierarchical clustering. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 801–810.
6. Hearst M.A. and Pedersen J.O. 1Reexamining the cluster hypothesis: scatter/gather on retrieval results. In Proc. 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1996, pp. 76–84.
7. Hoad T. and Zobel J. Methods for identifying versioned and plagiarised documents.. J. Am. Soc. Inf. Sci. Technol., 54(3): 203–215, 2003.
8. Huffman S., Lehman A., Stolboushkin A., Wong-Toi H., Yang F., and Roehrig H. Multiple-signal duplicate detection for search evaluation. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 223–230.
9. Jardine N. and van Rijsbergen C. The use of hierarchic clustering in information retrieval.. Inf. Storage Retr., 7(5):217–240, 1971.
10. Manber U. Finding similar files in a large file system. In Proc. USENIX Winter 1994 Technical Conference, 1994, pp. 1–10.
11. Mei Q., Shen X., and Zhai C. Automatic labeling of multinomial topic models. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007, pp. 490–499.
12. Shivakumar N. and Garcia-Molina H. SCAM: a copy detection mechanism for digital documents. In Proc. 2nd Int. Conf. in Theory and Practice of Digital Libraries, 1995.
13. Wang X. and Zhai C. Learn from Web search logs to organize search results. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 87–94.
14. Willett P. Recent trends in hierarchic document clustering: a critical review.. Inf. Process. Manage., 24(5):577–597, 1988.
15. Zamir O. and Etzioni O. Grouper: a dynamic clustering interface to Web search results. In Proc. 8th Int. World Wide Web Conference, 1999.

Web Services

Eric Wohlstadter[1], Stefan Tai[2]
[1]University of British Columbia, Vancouver, BC, Canada
[2]University of Karlsruhe, Karlsruhe, Germany

Synonyms

e-Services

Definition

Web services provide the distributed computing middleware that enables machine-to-machine communication over standard Web protocols. Web services are defined most precisely by their intended use rather than by the specific technologies used, since different technologies are popular [1]. Web services are useful in a compositional approach to application development; where certain key features of an integrated application are provided externally through one or more remote systems. Additionally, Web service standards are a popular platform for wrapping existing legacy applications in a more convenient format for interoperability between heterogeneous systems. To provide interoperability Web services should follow standards for formatting application messages, describing service interfaces, and processing messages. Two popular technology choices discussed in this entry are the SOAP [5] based services and the REST (REpresentational State Transfer) [4] based services.

In contrast to traditional World Wide Web (WWW) resources, Web services decouple the user interface from the underlying programmatic interface, to provide an application programming interface (API) to clients. This API provides the means through which Web services expose remote computational resources or informational content over the Internet. Web services are generally designed to follow a service-oriented architecture which promotes the loose coupling useful for interaction in a wide-area environment. This loose coupling comes at the price of giving up more powerful mechanisms such as stateful objects which introduce tighter coupling [2,7].

Historical Background

Web services address enterprise application integration (EAI) in a wide area environment. Historically, they were designed in response to the need for a simpler base of standards than provided by distributed object computing. The original WWW was also unsatisfactory due to its tight coupling with the user interface.

Distributed object computing provides the abstraction of remote stateful objects whose methods can be invoked transparently by clients. This form of remote method invocation is popular in proprietary distributed systems but has never been widely deployed on the Internet. A standardized technology known as the Common Object Request Broker Architecture (CORBA) is commonly used to develop and deploy distributed object applications. Although the CORBA platform provides powerful abstractions for application programmers, these abstractions tended to introduce a high level of complexity in the underlying infrastructure. Consensus was never reached on several important technical problems introduced by the stateful nature of objects such as distributed garbage collection and object identification [2,7]. Thus, distributed object computing is more appropriate in single enterprise deployment scenarios (e.g., telecommunications and military applications).

With the rise in acceptance of XML, it made sense to leverage this format as a means of machine-to-machine communication. In creating new standards around this format, certain problematic features such as stateful objects were not included in the newly developed standards leading to more lightweight and loosely coupled Web services.

The WWW was created as a collaborative platform for distributing information through web pages: documents with text and images including the capability to link between related content. The standard format for web pages, Hypertext Markup Language, included mixed elements of both informational content and user interface design. With the tremendous popularity of the WWW, came the opportunity to bring rich up-to-date informational data sources to a large audience. This also required complex business processes to manage requests for data. Since the retrieved data was tangled with user interface code, it was difficult to separate the two concerns for users interested in post-processing the retrieved information. "Wrappers" had to be used to parse, or "scrape" useful content from delivered pages. To simplify the distribution of information, some websites began offering remote APIs for information retrieval which has led to the interest in standardized Web services.

Foundations

Since interoperability is a key element for any Web service, some standardized approach must be taken to export service functions. The most formal approach to building Web services is built around two core standards known as SOAP and the Web Services Description Language (WSDL) [8]. The standards are being developed by organizations such as the W3C and OASIS. Other lightweight and dynamic approaches are generally categorized under the moniker of REST services. The REST approach requires few standardized technologies beyond HTTP. For both approaches, the standard way to identify each service is through means of an Internet Uniform Resource Identifier (URI). To locate an existing service, a URI is typically obtained out-of-band through informal means, although other automated approaches to service discovery are being explored.

In their most basic form, Web services provide only the basic primitives for exchanging documents between clients and servers. Standards for building robust, transactional, and secure services have been proposed but are not currently widely deployed. The remainder of this entry focuses on the core Web services fundamentals and describes how both SOAP/WSDL and REST deal with the details of service implementation: message formatting, interface description, and message processing.

Web services use a human readable message format for transport of application specific data. This practice simplifies the task of debugging and loosens a client's reliance on complex implementations for marshalling data. A popular message format is XML due to its self-described nature and abundance of third-party support for processing and storage. The use of XML is mandated by the SOAP/WSDL approach and is popular in REST services also. Conversion of XML to data structures native for a specific programming language can be done through standardized mappings. These mappings are supported by middleware which automate the mapping process. When REST services are consumed directly from a browser agent through JavaScript, a format known as JSON (JavaScript Object Notation) [3] can be used. Unlike XML, JSON provides a direct human readable serialization of JavaScript objects. This can reduce problematic mismatches between the differing type systems of XML Schema [10] and JavaScript.

As an API, Web services often provide a number of distinct functions each with their own input requirements and output guarantees. This interface can be specified formally or informally. Formal description of Web service interfaces is provided through WSDL. WSDL is an extension of the XML Schema specification and most importantly provides description of the set of functions exposed by a service. WSDL specifications add the possibility of providing statically typed-checked service use for clients. This is done by implementing message format mapping through a code generation and compilation process.

With REST, the inputs to service functions are specified as standard web URIs, making use of URI parameters for data input. Output can be produced using any standard HTTP supported encoding and XML is often used for this purpose. The original description of the REST architecture placed considerable emphasis on the fact that services should be described as a set of resources which could be manipulated using predefined HTTP methods (i.e., GET, POST, DELETE, etc.), rather than an application specific set of functions. However in practice, the term REST more commonly refers to any service accessible with little technology beyond HTTP, whether or not the service is modeled primarily as resources or functions. REST currently provides no formal treatment for description of services, although discussion of a standard REST description language is underway as of this writing.

Although not required, many Web services middleware platforms are built around a common architectural style referred to as the Chain of Responsibility pattern [6]. On a local scale incoming and outgoing Web service messages pass through a series of components, called handlers or interceptors, before being passed to lower network layers. This architecture promotes loose-coupling within the middleware itself. Handlers addressing separate concerns such as reliability, transactions or security can easily be plugged into the middleware and configured on a per application basis. A large number of standards known as WS-* [9] are being developed to specify the formats and protocols for managing these concerns in middleware implementations.

SOAP provides a standard for encapsulating XML based Web service messages with an envelope that can be used to communicate such extra-functional information such as security tokens, time stamps, etc. Using SOAP, the Chain of Responsibility architecture can be extended to a distributed series of message processing steps. In the distributed scenario each step can be handled at a potentially different intermediary location on the network, giving rise to Web service intermediaries. SOAP also standardizes the fault handling semantics for undeliverable messages and provides a number of message exchange patterns. These patterns capture certain reliability guarantees which can potentially be of use by the underlying middleware for optimizing resource usage.

REST messages are transported according to the semantics of the Hypertext Transport Protocol resource access methods. REST architectural principles mandate the use of stateless protocols wherever possible. REST message processors should consider only point-to-point connections as opposed to an end-to-end connection. This simplifies the interposition of caches or other intermediaries for message processing. REST services should rely only on standardized HTTP message headers for communicating extra-functional processing instructions, to keep implementation infrastructure lightweight.

Key Applications

Web services expose data and application functionality using standard Internet technology for consumption by clients over the Web. For example, Web services are used to provide access to very large product data. Today, the range of Web services offerings spans from very simple services to more advanced offerings in support of specific business requirements. Simple services include format conversion services, transformation services and real-time information services such as news feeds and stock quotes. Business offerings include the full range of e-commerce, CRM, marketing, finance, and communication services, among them verification services such as credit checking, accounting services, or customized notification services. Further, entire business processes such as the shipment of goods are provided as Web services.

The modular nature of Web services supports the - de-composition of application functionality and business processes and their (re-)composition into new applications. Services composition applies both within an organization and across different organizations. Compositions can take various forms, including workflow applications that require the use of a process composition language and centralized workflow engine. Web services in this way support EAI using standard Internet technology. Another form of composition are Mashups, re-purposing Web-accessible content and functionality in new Web applications that provide an interactive (human) end-user experience. For example, a Mashup may visualize location information of social event feeds on a map.

Web services are also being used in re-architecting middleware software platforms as service-oriented architectures. Middleware services such as message queuing and data storage are provided as Web services; middleware functionality can thus be externalized and used as a remote service. This model targets small and medium sized enterprises in particular, but applies equally to larger size enterprises as well. More advanced services offerings include application hosting and execution services, which allow clients to have entire software applications hosted and executed externally.

Future Directions

Today, there exists a variety of Web services protocols, APIs, languages, and specifications. Different combinations of Web services technology are being used for different purposes. For example, SOAP and Web services specifications in support of security and reliability are used in enterprise computing, whereas REST services are popular for simple services like data access and applications in the social Web.

From a business viewpoint, Web services establish a new market for trading and contracting electronic services that complement and transform the traditional consulting services and software market. The development of this market today is accompanied by the emergence of intermediaries for buyers and sellers to exchange services. These intermediaries distinguish themselves using different pricing mechanisms, co-marketing efforts, billing and accounting management, services usage monitoring, data integration models, and infrastructure quality guarantees for services availability. New Web services technology in support of this business development is expected to emerge.

Cross-references

▶ AJAX
▶ Coupling and De-Coupling
▶ Discovery
▶ Enterprise Application Integration
▶ Interface
▶ Loose Coupling
▶ Mashups
▶ OASIS
▶ Request Broker
▶ Service Oriented Architecture
▶ SOAP
▶ Tight Coupling
▶ W3C
▶ Web 2.0 (3.0)

Recommended Reading

1. Alonso G., Casati F., Kuno H., and Machiraju V. Web Services: Concepts Architectures and Applications. Springer, Berlin, 2003.
2. Birman K. Like it or not, web services are distributed objects. Commun. ACM, 47(12):60–62, 2004.
3. Crockford D. The application/json media type for JavaScript object notation. Network Working Group, RFC 4627, 2006.
4. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. Ph.D. Dissertation, University of California, 2000.
5. SOAP Version 1.2. W3C Recommendation, 2007.
6. Vinoksi S. Chain of responsibility. IEEE Internet Comput., 6(6):80–83, 2002.
7. Vogels W. Web Services are not Distributed Objects. IEEE Internet Comput., 7(6):59–66, 2003.
8. Web Services Description Language Version 2.0. W3C Recommendation, 2007.
9. Weerawarana S., Curbera F., Leymann F., Storey T., and Ferguson D. Web Services Platform Architecture. Prentice Hall, Upper Saddle River, NJ, 2005.
10. XML Schema. W3C Recommendation, 2004.

Web Services and the Semantic Web for Life Science Data

Cartik R. Kothari, Mark D. Wilkinson
University of British Columbia, Vancouver, BC, Canada

Definitions

Web services are frameworks for communication between computer applications on the World Wide Web. A general feature of "canonical" Web services is that they expose Web-based application interfaces in the form of a Web Services Description Language (WSDL) document with machine-readable content describing the input(s), output(s), and location of an application. The *Semantic* *web* extends the traditional Web by applying human and machine-readable labels to the links between resources, and encouraging those resources to be "typed" into a set of well-grounded categories, defined by shared *ontologies*. Machines can thus explore and process the content of the Semantic Web in meaningful ways. Moreover, the Semantic Web moves beyond simple documents and allows linking of individual data points, analytical tools, and even conceptual entities with no physical representation. O*ntologies* are formal descriptions of a knowledge domain, and range from simple controlled vocabularies to explicit logical definitions of the defining properties for all concepts and inter-concept relationships within that knowledge domain. In the *life sciences*, with its multitude of specialized data repositories and data processing applications, the Semantic Web paradigm promises to overcome the hurdles to data discovery, integration, mining, and reuse that continues to hinder biomedical investigation in the post-genomic era.

Historical Background

The life sciences are experiencing what might best be described as a feedback-loop – the ability of life scientists to generate raw "omics" data in vast quantities is perhaps only exceeded by the quantity of data output from tools analyze this raw data, and both become the fodder for new downstream analyses. Though much of the data is stored in one of several generalized Web-based repositories – for example UniProt (http://www.pir. uniprot.org), GenBank (http://www.ncbi.nlm.nih.gov/GenBank), or EMBL (http://embl.org) – still more data is housed in specialized repositories such as Gene Expression Omnibus (http://www.ncbi.nlm.nih.gov/geo/) (GEO), for Gene Expression Data, or species-specific Web databases such as Wormbase (http://www.wormbase.org/), Flybase (http://flybase.bio.indiana.edu/), DragonDB (http://antirrhinum.net), and TAIR (http://www.arabidopsis.org/). Similarly, many common analytical tools are also made available on the Web, through traditional CGI or JavaScript interfaces. In the course of their daily experimental investigations, Biologists frequently need to use various combinations of these distributed data and/or analytical resources. For most biologists, this is achieved by manual copy/paste of the output from one resource into the input form fields of the next. Computer-savvy biologists might undertake to construct "screen scrapers" – small computer programs that automatically process the output from Web resources and pipelined data from one resource to the next – and this is further facilitated by the creation of centralized code repositories such as BioPerl (http://www.bioperl.org) and BioJava (http://biojava.org) that provide a uniform API into a standard and curated set of "scrapers." Nevertheless, automation of data extraction from traditional Web forms is fraught with difficulties; with the lack of a common data interchange format and shareable semantics being the biggest hurdle.

The introduction of Web Services provided a more standardized, and platform/language-independent method of exposing data and analytical tools on the Web, and encouraged the use of XML as the standard data exchange format between different web resources. Moreover, the interface to and functionality of a Web Service could be described in a standardized machine processable syntax – Web Services Description Language (http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/) (WSDL), and a novel transport protocol – Simple Object Access Protocol (http://www.w3.org/TR/2007/REC-soap12-part0–20070427/) (SOAP) – was introduced to hide the complexities of the data exchange process and thus facilitate design of simple object- oriented accessors to remote Web resources. A paradigm for Web Services discovery (UDDI) was also proposed; however this has not been widely adopted, and will not be discussed here.

Although Web Services greatly aided the standardization of interfaces, the problem of interoperability had not been solved – the pipelining of Web Services continued to require sophisticated knowledge of the interfaces such that appropriate data-structures could be extracted from one service and fed into the next. This is because the XML data syntax lacks semantic grounding, and thus the input and output to/from any Web Service is opaque to automated interpretation, which thwarts attempts at automated Web Service workflow composition. The advent of the Semantic Web and ontologies constructed with Semantic Web compatible knowledge representation formalisms promises to overcome this hurdle.

The Semantic Web is being implemented as a Web of machine processable data. This is in contrast to the first version of the World Wide Web where machines were primarily concerned with the presentation of data; consumption and interpretation were limited to humans. Data on the Semantic Web is rendered machine processable by annotation with syntactic constructs whose semantics are grounded in formal mathematical logic. Knowledge representation formalisms such as the Resource Description Framework (http://www.w3.org/RDF/) (RDF) and the Web Ontology Language (http://www.w3.org/2004/OWL/) (OWL) provide syntactic constructs with logically grounded semantics that can be used to annotate web services.

Ontologies such as the OWL-Services (http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/) ontology and the Web Services Modeling Ontology (http://www.w3.org/Submission/WSMO/) (WSMO) are fresh initiatives to leverage the Semantic Web infrastructure for web service composition. A web service on the Semantic Web can be described by instances of concepts from the OWL-S or WSMO and easily invoked and composed with other compatible web services.

Foundations

Web Services

Web Services have revolutionized the field of distributed computation. Web Services enable the remote discovery and execution of software applications across the dimensions of the Web in an implementation agnostic manner. According to the World Wide Web Consortium (W3C), a Web Service is "a software system designed to support interoperable machine to machine interaction over a network." The functionality of Web Services can be easily understood from three different perspectives viz. the service provider, the service client, and the service registry.

The Service Provider A service provider is a person or organization that is responsible for the development and maintenance of the Web Service. The service provider is familiar with the implementation details of the service, such as the architecture of the underlying application and the programming language used. These details are encapsulated by the Web Service interface, which describes the function of the Web Service, its location, input parameters and output parameters and their data-types. Consider an application that performs a search for relevant publication in an online, open access repository such as PubMed (http://www.pubmed.org). This application would be encapsulated in a web interface (the Web Service), which advertises the function of the service (searching for relevant publications), the location of the service (a resolvable URI, usually a URL), the input parameters (keyword, name of the author, citation details such as journal name and/or date of publication, all of which would be string-literal data-types), and the output parameters of the service (title of the publication, abstract, and citation details; all of which are string-literals). This Web Service description is published as a WSDL document and is (optionally) registered in a registry. Note the description does not include implementation details. The underlying application could be implemented on a Linux or Microsoft platform, using any programming or scripting language desired. More importantly, can also be accessed by a client application on any platform using any language.

The Service Client The service client locates a specific Web Service from its description. The client then remotely invokes the discovered service and executes it. In the PubMed example, consider a service client that is looking for publications about the Vascular Endothelial Growth Factor (VEGF) protein. The service client discovers the book browsing service from its WSDL description. The WSDL description details the input parameters accepted by the Web Service, one of which is "keyword." The client sends the text string "VEGF" into the interface as the "keyword" parameter and thereby invokes the service. The communication between the client and the service generally utilizes the HTTP protocol, and is often supplemented by a standardized messaging format such as SOAP.

Note that "keyword" is a *human-readable* label indicating the purpose of that input parameter – the WSDL interface definition must be interpreted by a person in order to determine the intent of that parameter, emphasizing the point that, in traditional Web Services, the interfaces are opaque due to the lack of ontological grounding of XML tags.

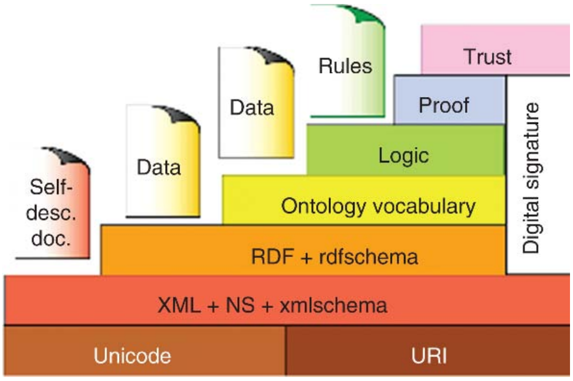
The Service Registry The service registry acts as a "yellow pages" for Web Service discovery, and brokers the direct interaction between a client and a chosen service. Specifically, the registry holds WSDL descriptions of Web Services, including their functional characteristics such as input and output parameters, and location. In addition, human readable descriptions of the web service may also be included in the WSDL description. Details of the service provider, and classification information (e.g., the UNSPSC category of the service), and reliability estimates may also be provided. In addition, Web Service registries may perform ancillary tasks such as periodic polling to identify inactive services, and collecting performance metrics and reliability estimates.

The Semantic Web

The Semantic Web initiative aims to make the vast amounts of information on the Web processable by machines. Previously, computers were involved only in the presentational aspects of this information, which was marked up with tags from the HyperText Markup Language (http://www.w3.org/html/) (HTML). HTML tags specify how the tagged information is to be presented on a browser. However, there are no semantics associated with HTML tags, nor are there semantics associated with the hyperlinks between documents, thus preventing automated interpretation of the contained information. The development of knowledge representation formalisms for the Semantic Web such as RDF, the Ontology Inference Layer (http://www.ontoknowledge.org/oil/) (OIL), the DARPA Advanced Markup Language (http //www.daml.org) (DAML), from the Defense Advanced Research Projects Agency (http://www.darpa.mil) (DARPA), and OWL have been an attempt to overcome these limitations of the Web.

OWL is the most advanced formalism for markup on the Semantic Web, providing a very expressive and decidable set of constructs for ontology development, and has been adopted by the W3C as a standard. The semantics of constructs from OWL (and OIL and DAML) are grounded in mathematical logic. Therefore, the information content is now accessible to computers and can be used by logical inference axioms to deduce implicit knowledge. Lastly, the information content can be queried by machines across the Web. Intelligent, semantic based searches become a possibility, replacing the prevalent simple keyword based searches and word and hyperlink based algorithms. Three components are critical to understanding the Semantic Web architecture; ontologies, query engines and inference engines. Figure 1 shows the Semantic Web architecture proposed by Tim Berners-Lee.

Ontologies Ontologies are machine interpretable models of knowledge domains. An ontological representation of a knowledge domain contains logically grounded definitions of concepts and inter-concept relations that are specific to that domain. Since these concepts and relations are defined in a rigorous logical framework, their semantics can be shared and processed by machines without loss of clarity. The information content of Web pages on the Semantic Web can be marked up as instances of ontologically defined concepts and relations. Knowledge representation formalisms such as OWL can be used to construct ontologies of different knowledge domains. Ontology editing tools such as Protégé (http://protege.stanford.edu/), TopBraid suite (http://www.topquadrant.com/topbraid suite.html), and Altova SemanticWorks (http://www.altova.com/products/semanticworks/semantic_web_rdf_owl_editor. html) have made the process of ontology development very straightforward for knowledge do main experts. This is attested by the proliferation of ontologies, especially in the life sciences domain.



Web Services and the Semantic Web for Life Science Data. Figure 1. Proposed semantic web architecture (Berners-Lee et al., 2001).

Query Engines Marking up the information content of Web pages with semantically rich HTML tags makes it possible for query engines and languages to retrieve this information. Two of the earliest query languages for retrieving RDF annotated information were SPARQL (http://www.w3.org/TR/rdf-sparql-query/) and the RDF Query Language (http://139.91.183.30:9090/RDF/RQL/) (RQL). SeRQL (//www.openrdf.org/doc/sesame/users/ch06.html), nRQL [9], and RDQL (http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/) are other querying formalisms that have been developed to retrieve RDF annotated documents from the Semantic Web.

Inference Engines Inference is the process of deducing implicit information from the combination of explicitly stated information and inference axioms. For example, given that any person who is the parent of a parent is a grandparent (the axiom), and given that Lisa is the parent of Charles and Charles is the parent of Harry (explicitly stated facts), it can be deduced that Lisa is a grandparent of Harry (inference). In this fashion, logical inference axioms can be used to reason with the concept (and relation) definitions and their instances. The information content of the Semantic Web can be reasoned with by specialized inference engines such as FaCT + + (http://owl.man.ac.uk/factplusplus/), Pellet (http://pellet.owldl.org), and RACERPro (http://www.racer-systems.com/products/racerpro/index.phtml).

Semantic Web Services

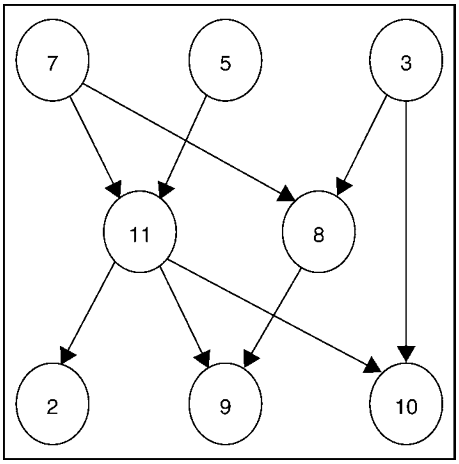
The possibility of marking up Web Services with semantically expressive tags led to the adoption of the Semantic Web Services initiative (http://www.swsi.org/). The objective of the Semantic Web Services initiative is the use and leverage of the framework of Semantic Web technologies to enable the automation and dynamism of Web Services annotation, publication, discovery, invocation, execution, monitoring, and composition. Referring again to the PubMed search example, the input and output parameters of the search web service can be described using ontologically defined concept instances. WSDL provides a means to specify the acceptable input parameters of the Web Service to be one of three string literals, viz. keyword, citation details, and author name. However, the fact that the author name, citation details, and keyword are specialized types of string literals cannot be explicitly stated with WSDL. Using an ontology to ground these Web Service parameters as specific "types" of entity, it then becomes possible to indicate, to both human and machine, the purpose of each parameter field. Moreover, the concept of a "Literature Search Tool" could be defined as a Web Service that had parameters of type "author" and "keyword." An inference engine could then be used to discover this PubMed Web Service in response to a user-request for Literature Search Tools.

Semantic Web Services in the Life Sciences

Computational methods, also known as *in silico experiments*or *workflows*, complement in vivo and in vitro methods in biological research, and provide high-throughput access to tools that can manage the large volumes of genomic, proteomic, and other types of data required by modern biomedical experiments. Since many analytical tools are now available as Web Services, it is now possible to "pipeline" these tools together to create *in silico*workflows, with the biologist manually connecting the output of one service to the input of the next based on a human readable WSDL description. This, however, is a complex task, and requires the biologist to be aware of the individual capabilities and operating principles of each Web Service. Availability of a comprehensive textual description of a Web Service's capabilities and operating principles is rare, and not always helpful to a biologist who is not computer-savvy. Therefore, the constructed workflows maybe error-prone and/or procedurally flawed. Moreover, many applications go through versions or become unavailable from time to time, making workflows somewhat fragile. The burden of "re-wiring" a workflow to accommodate a new version of an interface; replacing a dead interface with an equivalent functional one; and discovering new or improved applications as and when they become available, is also placed on the biologist and requires knowledge and awareness outside of their domains of expertise. Machine processable descriptions of Web Services are, thus, crucial to enabling their automatic discovery and invocation, and most importantly, to their automated composition into stable, self-repairing analytical workflows.

Registries of semantically annotated Web Services such as myGrid (http://www.mygrid.org.uk/) and Moby Central (http://biomoby.org/RESOURCES/MOBY-S/ServiceInstances) have been used to facilitate the discovery and use of Web Services by biologists. However, biologists still need to familiarize themselves with datatype hierarchies to discover web services that can process their data. Recent innovations capable of inferring datatypes from actual data [4] promise to relieve biologists of this requirement as well. The automated composition of Web services is more troublesome. Automated pipelining the output of one web service into the input of another is the simplest of the proposed approaches to service composition. In mathematical terms, this is analogous to finding a route in a directed graph. A web services network can be visualized as a directed graph as shown in Figure 2.

Every node in Figure 2 corresponds to a web service. Directed edges leading to a webservice correspond to input parameters, and those leading away from a



Web Services and the Semantic Web for Life Science Data. Figure 2. A directed graph representation of a web service network.

service, to output parameters. Finding a route from node 7 (which could represent a service that queries a genome database given an accession number, for instance) to node 10 (which may represent a service that performs a sequence comparison and outputs a report) in this figure is a computationally complex problem, capable of consuming large extents of computational time and memory. Even with finding the shortest possible route in the figure from node 7 to node 10, uninformed search algorithms are exponentially complex. This means that the time taken to find a solution (and the computational memory used) increases exponentially as the number of possible solutions. Heuristics are required to prune the search space in such problems and reduce the complexity to decidable and tolerable limits. In web service composition, the complexity of the problem is compounded due to several factors. Some services are asynchronous, meaning they take in input parameters without outputting anything. Other services may be notifiers, giving output parameters without (seemingly) taking in any input. Some edges many not necessarily lead to other services, being dead end points in the route finding algorithm. Lastly, the shortest combination of services may not be desirable to the biologist as well. The automated composition of web services is therefore, an open and fascinating research problem. The interested reader may refer the resources at the Web Services Choreography Working Group (http://www.w3.org/2002/ws/chor/) Web site at the W3C for more information.

Key Applications

BioMoby [10] is an open source, extensible framework that enables the representation, discovery, retrieval, and integration of biological Web Services. By registering their analysis and data access services with BioMoby, service providers agree to use and provide service specifications in a shared semantic space. The BioMoby Central registry now hosts more than a thousand services in the United States, Canada, and several other countries across the world. BioMoby uses a datatype hierarchy to facilitate automated discovery of Web Services capable of handling specific input datatypes. As a minimal Web based interface, the Gbrowse Moby (http://moby.ucalgary.ca/gbrowse_moby) service browser can be used by biologists to discover and invoke biological web services from the Moby registry and seamlessly chain these services to compose multi-step analytical workflows. The process is data centric, relying on input and output datatype specifications of the services. The Seahawk client interface [4] can infer the datatype of the input data files directly and immediately present the biologist with a list of BioMoby Web Services that can process that input file. Users of the Seahawk interface are relieved of the necessity to familiarize themselves with datatype hierarchies, and instead are free to concentrate on the analytical aspects of their work. Other clients for BioMoby services include Remora (http://lipm-bioinfo.toulouse.inra.fr/remora/cgi/remora.cgi), Ahab (http://bioinfo.icapture.ubc.ca/bgood/Ahab.html), and MOWserv (http://www.inab.org/MOWServ/).

myGrid [3] is an example of a computational grid architecture that brings together various computational resources to support *in silico* biological experiments. The computational resources include analysis and data retrieval services. As such, myGrid is a service grid and the myGrid philosophy regards *in silico* experiments to be distributed queries and workflows. Taverna [5], a module of myGrid, provides a means to integrate the diverse resources on the myGrid framework into reusable *in silico* workflows. Grimoires [1] is a semantics enabled service registry that enables discovery of myGrid services. Taverna also provides an interface to access BioMoby services [6], a promising development towards the integration of research initiatives in the area of biomedical informatics.

The PathPort framework [11] developed at The Virginia Bioinformatics Institute presents a Web based interface that makes it possible for end users to invoke local and distributed biological Web Services that are described in WSDL, in a location and platform independent manner. Sembowser [8] is an ontology based implementation of a service registry that facilitates Web Service annotation, search and discovery based upon domain specific keywords whose semantics are grounded in mathematical logic. Sembowser associates every published Web Service with concepts defined in an ontology and further, classifies the published service on the basis of the task it performs and the domain with which it is associated.

The Simple Semantic Web Architecture and Protocol (http://semanticmoby.org) (SSWAP), is a Semantic Web Services architecture that uses ontology based descriptions of biological Web Services from the Virtual Plant Information Network (http://vpin.ncgr.org/) (VPIN) to discover, and remotely execute Web services. Formerly known as Semantic Moby, this architecture is the closest realization of the conventional definition of a Semantic Web Services framework, and utilizes the knowledge embedded in any Web-based ontology to assist in discovery and pipelining of Web Services.

The need for semantic descriptions to enable automated Web Service discovery, invocation and composition has led to the development of prototypical ontology based frameworks. The OWL-S framework, the METEOR-S framework (http://lsdis.cs.uga.edu/projects/meteor-s/), and the Web Services Modeling Framework (WSMF) [2] are examples of research initiatives in this area. Ontology based Web Service architectures are in very early stages of development and their evolution into industrial strength technologies will be eagerly anticipated.

Web services with machine processable descriptions are of great value in the development of mashups. A mashup is a composite application that brings together data and functionality from diverse sources using technologies such as AJAX and RSS. Mashup development has been of specific interest to the Web 2.0 community, with its emphasis on mass collaborative information gathering techniques, as exemplified by Wikipedia (http://www.wikipedia.org), Flickr (http://www.flickr.com), and Delicious (http://del.icio.us). Very recently, diverse applications have been embedded with Semantic Web ontologies, query engines, and inference engines to create mashups. Mashups have been used to answer specific questions about genes responsible for Alzheimer's disease and also visualize these genes [7]. In this demo, results from SPARQL queries have been integrated with data extracted from the Allen Brain Atlas site (http://www.brainatlas.org/aba/) using screen scraping techniques, and finally visualized using Google Maps (http://maps.google.com) and the Exhibit (http://simile.mit.edu/exhibit/)visualization toolkit.

Future Directions

The Semantic Web, with its promise of information sharing, reuse and integration, has seen widespread adoption by the life sciences community. Currently, the curators of many of the major genome, proteome, transcriptome, and interactome repositories are adopting Semantic Web standards, while more and more service providers are gravitating towards the Semantic Web Services paradigm. Semantic Web standards are crucial to a synergistic approach to a full understanding of, and representation of, complexity in Life Sciences.

Cross-references

▶ Controlled Medical Vocabulary
▶ Data Structures and Models for Biological Data Management
▶ Data Types in Scientific Data Management Systems
▶ Grid and Workflows
▶ Mashups
▶ Ontologies
▶ Ontologies and Life Science Data Management
▶ Ontologies in Scientific Data Integration
▶ Ontology
▶ Ontology-Based Data Models
▶ OWL: Web Ontology Language
▶ Query Languages for Ontological Data
▶ Query Languages for the Life Sciences
▶ RDF
▶ Scientific Workflows
▶ Screen Scraper
▶ Semantic Web
▶ Semantic Web Query Languages
▶ Semantic Web Services
▶ SOAP
▶ W3C
▶ Web 2.0 (3.0)
▶ Web Services
▶ Workflow Management and Workflow Management Systems
▶ Workflow Model
▶ Workflow Modeling
▶ XML

Recommended Reading

1. Fang W, et al. Performance analysis of a semantics enabled service registry. In Proc. 4th All Hands Meeting. Nottingham, UK, 2005. Available at: http://users.ecs.soton.ac.uk/lavm/papers/Fang-AHM05.pdf
2. Fensel D. and Bussler C. The Web Services Modeling Framework (WSMF), Electron. Commerce Res. Appl., 1(2):113–137, 2002.
3. Goble C., Pettifer S., Stevens R., and Greenhalgh C. Knowledge integration: in silico experiments in bioinformatics. In the Grid: Blueprint for a New Computing Infrastructure (2nd edn.), I. Foster, C. Kesselman (eds.). Morgan Kaufmann, Los Altos, CA, 2003.
4. Gordon P. and Sensen C. Seahawk: Moving beyond HTML in Web-based Bioinformatics Analyses. BMC Bioinformatics, 8(June):208, 2007.
5. Hull D., Wolstencroft K., Stevens R., Goble C., Pocock M., Li P., and Oinn T. Taverna: a tool for building and running workflows of services. Nucleic Acids Res., 34: W729–W732, 2006.
6. Kawas E., Senger M., and Wilkinson M. BioMoby extensions to the taverna workflow management and enactment software. BMC Bioinformatics, 7:523, 2006.
7. Ruttenberg A, et al. Advancing translational research with the semantic web. BMC Bioinformatics, 8(Suppl. 3):S2, May 2007.
8. Sahoo S., Sheth A., Hunter B., and York W. Sembowser: adding semantics to a biological web services registry. In Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences, C. Baker, K. Cheung (eds.). Springer 2007.
9. Wessel M. and Möller R. A high performance semantic web query answering engine. In Proc. Int. Workshop on Description Logics, 2005.
10. Wilkinson M. and Links M. BioMOBY: an open-source biological web services proposal. Brief. Bioinform., 3(4): 331–341, 2002.
11. Xue T., Yang B., Will R., Sharp B., Kenyon R., Crasta O., and Sobral B. A generalized framework for pathosystems informatics and bioinformatics web services. In Proc. 2007 Int. Conf. on Bioinformatics and Computational Biology, 2007.

Web Services Business Process Execution Language

▶ Composed services and WS-BPEL

Web Site Wrappers

▶ Languages for Web Data Extraction

Web Spam Detection

Marc Najork
Microsoft Research, Mountain View, CA, USA

Synonyms

Spamdexing; Google bombing; Adversarial information retrieval

Definition

Web spam refers to a host of techniques to subvert the ranking algorithms of web search engines and cause them to rank search results higher than they would otherwise. Examples of such techniques include content spam (populating web pages with popular and often highly monetizable search terms), link spam (creating links to a page in order to increase its link-based score), and cloaking (serving different versions of a page to search engine crawlers than to human users). Web spam is annoying to search engine users and disruptive to search engines; therefore, most commercial search engines try to combat web spam. Combating web spam consists of identifying spam content with high probability and – depending on policy – downgrading it during ranking, eliminating it from the index, no longer crawling it, and tainting affiliated content. The first step – identifying likely spam pages – is a classification problem amenable to machine learning techniques. Spam classifiers take a large set of diverse features as input, including content-based features, link-based features, DNS and domain-registration features, and implicit user feedback. Commercial search engines treat their precise set of spam-prediction features as extremely proprietary, and features (as well as spamming techniques) evolve continuously as search engines and web spammers are engaged in a continuing "arms race."

Historical Background

Web spam is almost as old as commercial search engines. The first commercial search engine, Lycos, was incorporated in 1995 (after having been incubated for a year at CMU); and the first known reference to "spamdexing" (a combination of "spam" and "indexing") dates back to 1996. Commercial search engines began to combat spam shortly thereafter, increasing their efforts as it became more prevalent.

Spam detection became a topic of academic discourse with Davison's paper on using machine learning techniques to identify "nepotistic links," i.e., link spam [4], and was further validated as one of the great challenges to commercial search engines by Henzinger et al. [9]. Since 2005, the workshop series on *Adversarial Information Retrieval on the Web* (AIRWeb) provides a venue for researchers interested in web spam.

Foundations

Given that the objective of web spam is to improve the ranking of select search results, web spamming techniques are tightly coupled to the ranking algorithms employed (or believed to be employed) by the major search engines. As ranking algorithms evolve, so will spamming techniques. For example, if web spammers were under the impression that a search engine would use click-through information of its search result pages as a feature in their ranking algorithms, then they would have an incentive to issue queries that bring up their target pages, and generate large numbers of clicks on these target pages. Furthermore, web spamming techniques evolve in response to countermeasures deployed by the search engines. For example, in the above scenario, a search engine might respond to facetious clicks by mining their query logs for many instances of identical queries from the same IP address and discounting these queries and their result clickthroughs in their ranking computation. The spammer in turn might respond by varying the query (while still recalling the desired target result), and by using a "bot-net" (a network of third-party computers under the spammer's control) to issue the queries and the click-throughs on the target results.

Given that web spamming techniques are constantly evolving, any taxonomy of these techniques must necessarily be ephemeral, as will be any enumeration of spam detection heuristics. However, there are a few constants:

- Any successful web spamming technique targets one or more of the features used by the search engine's ranking algorithms.
- Web spam detection is a classification problem, and search engines use machine learning algorithms to decide whether or not a page is spam.
- In general, spam detection heuristics look for statistical anomalies in some of the features visible to the search engines.

Web Spam Detection as a Classification Problem

Web spam detection can be viewed as a binary classification problem, where a classifier is used to predict whether a given web page or entire web site is spam or not. The machine learning community has produced a large number of classification algorithms, several of which have been used in published research on web spam detection, including decision-tree based classifiers (e.g., C4.5), SVM-based classifiers, Bayesian classifiers, and logistic regression classifiers. While some classifiers perform better than others (and the spam detection community seems to favor decision-tree-based ones), most of the research focuses not on the classification algorithms, but rather on the features that are provided to them.

Taxonomy of Web Spam Techniques

Content spam refers to any web spam technique that tries to improve the likelihood that a page is returned as a search result and to improve its ranking by populating the page with salient keywords. Populating a page with words that are popular query terms will cause that page to be part of the result set for those queries; choosing good combinations of query terms will increase the portion of the relevance score that is based on textual features. Naïve spammers might perform content spam by stringing together a wide array of popular query terms. Search engines can counter this by employing language modeling techniques, since web pages that contain many topically unrelated keywords or that are grammatically ill-formed will exhibit statistical differences from normal web pages [11]. More sophisticated spammers might generate not a few, but rather millions of target web pages, each page augmented with just one or a few popular query terms. The remainder of the page may be entirely machine-generated (which might exhibit statistical anomalies that can be detected by the search engine), entirely copied from a human-authored web site such as Wikipedia (which can be detected by using near-duplicate detection algorithms), or stitched together from fragments of several human-authored web sites (which is much harder, but not impossible to detect).

Link spam refers to any web spam technique that tries to increase the link-based score of a target web page by creating lots of hyperlinks pointing to it. The hyperlinks may originate from web pages owned and controlled by the spammer (generically called a *link farm*), they may originate from partner web sites (a technique known as *link exchange*), or they may originate from unaffiliated (and sometimes unknowing) third parties, for example web-based discussion forums or in blogs that allow comments to be posted (a phenomenon called *blog spam*). Search engines can respond to link spam by mining the web graph for anomalous components, by propagating distrust from spam pages backwards through the web graph, and by using content-based features to identify spam postings to a blog [10]. Many link spam techniques specifically target Google's PageRank algorithm, which not only counts the number of hyperlinks referring to a web page, but also takes the PageRank of the referring page into account. In order to increase the PageRank of a target page, spammers should create links on sites that have high PageRanks, and for this reason, there is a marketplace for expired domains with high PageRank, and numerous brokerages reselling them. Search engines can respond by temporarily dampening the endorsement power of domains that underwent a change in ownership.

Click spam refers to the technique of submitting queries to search engines that retrieve target result pages and then to "click" on these pages in order to simulate user interest in the result. The result pages returned by the leading search engines contain client-side scripts that report clicks on result URLs to the engine, which can then use this implicit relevance feedback in subsequent rankings. Click spam is similar in method to *click fraud*, but different in objective. The goal of click spam is to boost the ranking of a page, while the goal of click fraud (generating a large number of clicks on search engine advertisements) is to spend the budget associated with a particular advertisement (to hurt the competitor who has placed the ad or simply to lower the auction price of said ad, which will drop once the budget of the winning bidder has been exhausted). In a variant of click fraud, the spammer targets ads delivered to his own web by an ad-network such as Google AdSense and obtains a revenue share from the ad-network. Both click fraud and click spam are trivial to detect if launched from a single machine, and hard to detect if launched from a bot-net consisting of tens of thousands of machines [3]. Search engines tackle the problem by mining their click logs for statistical anomalies, but very little is known about their algorithms.

Cloaking refers to a host of techniques aimed at delivering (apparently) different content to search engines than to human users. Cloaking is typically used in conjunction with content spam, by serving a page containing popular query terms to the search engine (thereby increasing the likelihood that the page will be returned as the result of a search), and presenting the human user with a different page. Cloaking can be achieved using many different techniques: by literally serving different content to search engines than to ordinary users (based for example on the well-known IP addresses of the major search engine crawlers), by rendering certain parts of the page invisible (say by setting the font to the same color as the background), by using client-side scripting to rewrite the page after it has been delivered (relying on the observation that search engine crawlers typically do not execute scripts), and finally by serving a page that immediately redirects the user's browser to a different page (either via client-side scripting or the HTML "meta-redirect" tag). Each variant of cloaking calls for a different defense. Search engines can guard against different versions of the same page by probing the page from unaffiliated IP addresses [13]; they can detect invisible content by rendering the page; and they can detect page modifications and script-driven redirections by executing client-side scripts [12].

Key Applications

Web spam detection is used primarily by advertisement-financed general-purpose consumer search engines. Web spam is not an issue for enterprise search engines, where the content providers, the search engine operator and the users are all part of the same organization and have shared goals. However, web spam is bound to become a problem in any setting where these three parties – content providers, searchers, and search engines – have different objectives. Examples of such settings include vertical search services, such as product search engines, company search engines, people search engines, or even scholarly literature search engines. Many of the basic concepts described above are applicable to these domains as well; the precise set of features useful for spam detection will depend on the ranking algorithms used by these vertical search engines.

Future Directions

Search engines are increasingly leveraging human intelligence, namely the observable actions of their user base, in their relevance assessments; examples include click-stream analysis, toolbar data analysis, and analysis of traffic on affiliate networks (such as the Google AdSense network). It is likely that many of the future spam detection features will also be based on the behavior of the user base. In many respects, the distinction between computing features for ranking (promoting relevant documents) and spam detection (demoting facetious documents) is artificial, and the boundary between ranking and spam suppression is likely to blur as search engines evolve.

Experimental Results

Several studies have assessed the incidence of spam in large-scale web crawls at between 8% and 13% [11,5]; the percentage increases as more pages are crawled, since many spam sites serve a literally unbounded number of pages, and web crawlers tend to crawl high-quality human-authored content early on. Ntoulas et al. describe a set of content-based features for spam detection; these features, when combined using a decision-tree-based classifier, resulted in an overall spam prediction accuracy of 97% [11].

Data Sets

Castillo et al. have compiled the WEBSPAM-UK2006 data set [2], a collection of web pages annotated by human judges as to whether or not they are spam. This data set has become a reference collection to the field, and has been used to evaluate many of the more recent web spam detection techniques.

Cross-references

- ▶ Indexing the Web
- ▶ Web Page Quality Metrics
- ▶ Web Search Relevance Feedback
- ▶ Web Search Relevance Ranking

Recommended Reading

1. Becchetti L., Castillo C., Donato D., Leonardi S., and Baeza-Yates R. Using rank propagation and probabilistic counting for link-based spam detection. In Proc. KDD Workshop on Web Mining and Web Usage Analysis, 2006.
2. Castillo C., Donato D., Becchetti L., Boldi P., Leonardi S., Santini M., and Vigna S. A reference collection for Web spam. ACM SIGIR Forum, 40(2):11–24, 2006.
3. Daswani N. and Stoppelman M. and the Google Click Quality and Security Teams. The anatomy of clickbot.A. In Proc. 1st Workshop on Hot Topics in Understanding Botnets, 2007.
4. Davison B.D. Recognizing nepotistic links on the web. In Proc. AAAI Workshop on Artificial Intelligence for Web Search, 2000.
5. Fetterly D., Manasse M., and Najork M. Spam, damn spam and statistics. In Proc. 7th Int. Workshop on the Web and Databases, 2004, pp. 1–6.
6. Gyöngyi Z., and Garcia-Molina H. Spam: its not just for Inboxes anymore. IEEE Comput., 38(10):28–34, 2005.
7. Gyöngyi Z. and Garcia-Molina H. Web Spam Taxonomy. In Proc. 1st Int. Workshop on Adversarial Information Retrieval on the Web, 2005, pp. 39–47.
8. Gyöngyi Z., Garcia-Molina H., and Pedersen J. Combating Web spam with TrustRank. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 576–587.
9. Henzinger M., Motwani R., and Silverstein C. Challenges in web search engines. ACM SIGIR Forum 36(2):11–22, 2002.
10. Mishne G., Carmel D., and Lempel R. Blocking blog spam with language model disagreement. In Proc. 1st Int. Workshop on Adversarial Information Retrieval on the Web, 2005, pp. 1–6.
11. Ntoulas A., Najork M., Manasse M., and Fetterly D. Detecting spam web pages through content analysis. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 83–92.
12. Wang Y.M., Ma M., Niu Y., and Chen H. Spam double-funnel: connecting Web spammers with advertisers. In Proc. 16th Int. World Wide Web Conference, 2007, pp. 291–300.
13. Wu B. and Davison B. Detecting semantic cloaking on the web. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 819–828.

Web Structure Mining

▶ Data, Text, and Web Mining in Healthcare

Web Transactions

Heiko Schuldt
University of Basel, Basel, Switzerland

Synonyms

Internet transactions

Definition

A *Web Transaction* is a transactional interaction between a client, usually a web browser, and one or several databases as backend of a multi-tier architecture. The middle tier of the architecture includes a web server which accepts client requests via HTTP. It forwards these requests either directly to the underlying database or to an application server which, in turn, interacts with the database.

Key Points

The main application of Web transactions is in eCommerce applications. In a minimal configuration, the architecture for Web transactions consists of a client, a web server and a database server. Database access is provided at the web server level, i.e., by embedding database access into Java servlets or JavaServer Pages (JSP). More sophisticated architectures consider a dedicated application server layer (i.e., an implementation of the Java Enterprise Edition specification Java EE) and support distributed databases. Communication between application and database layer is usually implemented on the basis of JDBC (Java Database Connectivity), ODBC (Open Database Connectivity), or native database protocols. Transaction support at application server level is provided by services like JTS (Java Transaction Service) via the Java Transaction API (JTA) in the Java world, or OTS (Object Transaction Service) in CORBA. Essentially, these services allow to associate several application server calls and their interactions with resource managers with the same transaction context and to coordinate them by using a two phase commit (2PC) protocol. Thus, Web transactions mostly focus on atomic commit processing. Similarly, protocols on the Web service stack exploit 2PC to atomically execute several Web services.

Multi-tier web applications require special support for failure handling at application level (application recovery). In order to increase the degree of scalability of multi-tier architectures for Web transactions, application servers can be replicated. To avoid that the database backend then becomes a bottleneck, dedicated caching strategies at the middle tier are applied. Web transactions can be part of Transactional processes.

Cross-references

- ▶ Application Server
- ▶ Caching
- ▶ Transactional Processes
- ▶ Web Service

Recommended Reading

1. Barga R., Lomet D., Shegalov G., and Weikum G. Recovery Guarantees for Internet Applications. ACM Trans. Internet Technol., 4(3):289–328, 2004.
2. Burke R. and Monson-Haefel R. Enterprise JavaBeans 3.0. O'Reilly, 2006.
3. Luo Q., Krishnamurthy S., Mohan C., Pirahesh H., Woo H., Lindsay B., and Naughton J. Middle-tier database caching for e-business. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 600 –611.

Web Usage Mining

▶ Data, Text, and Web Mining in Healthcare

Web Views

Alexandros Labrinidis
Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

Web Views; HTML fragment

Definition

Web Views are web pages or web page fragments that are automatically created from base data, which are typically stored in a database management system (DBMS).

Key Points

Although caching of HTML pages (and fragments) has been proposed in the literature as early as the mid-1990s, the term Web View was introduced in 1999 [2] to denote that these fragments are generated through queries made to a back-end DBMS.

The concept of a *Web View* facilitates the materialization of dynamically generated HTML fragments outside the DBMS. The main advantage of materializing Web Views (outside the DBMS, e.g., at the web server) is that the web server need not query the DBMS for every user request, thus greatly improving query response time for the user [3]. On the other hand, for the quality of the data returned to the user to be high, the system must keep materialized Web Views fresh (in the background). This essentially decouples the processing of queries at the web server from the processing of updates at the DBMS (which are also propagated to materialized Web Views).

Materializing a Web View presents a trade-off. On the one hand, it can greatly improve response time for user-submitted queries. On the other hand, it generates a background overhead for the Web View to be kept fresh (essentially a materialization decision can be viewed as a "*contract*" for the Web View to be kept fresh). As such, selecting which Web View to materialize is an important problem that has received attention [5,6]. This problem is essentially similar to the view selection problem in traditional DBMSs [5], with added complexity due to the online nature of the Web and the need for any solution to be highly dynamic, constantly adapting to changing conditions and workloads.

Having identified the set of Web Views to materialize, there is the issue of determining the order by which to propagate updates to them. Since one update to a base relation can trigger the refresh of multiple different materialized Web Views, the order by which these are updated can make an impact on the overall quality of data returned to the user. This is essentially a special-case online scheduling problem, which has been addressed in [4].

Web Views have been used successfully to decouple the processing of queries (to generate dynamic, database-driven web pages) from that of updates (to the content stored inside the DBMS used to driven the web site). This enables much better performance (in terms of response to user queries) without sacrificing the freshness of the data served back to the user.

Cross-references

▶ View Maintenance

Recommended Reading

1. Gupta H. and Mumick I.S. Selection of views to materialize under a maintenance cost constraint. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 453–470.
2. Labrinidis A. and Roussopoulos N. Alexandros On the materialization of Web views. In Proc. ACM SIGMOD Workshop on The Web and Databases, 1999, pp. 79–84.
3. Labrinidis A. and Roussopoulo N. Web View materialization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 367–378.
4. Labrinidis A. and Roussopoulos N. Update propagation strategies for improving the quality of data on the Web. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 391–400.
5. Labrinidis A. and Roussopoulos N. Balancing performance and data freshness in Web database servers. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 393–404.
6. Labrinidis A. and Roussopoulos N. Exploring the tradeoff between performance and data freshness in database-driven Web servers. VLDB J., 13(3):240–255, 2004.

Web Widget

▶ Snippet

What-If Analysis

Stefano Rizzi
University of Bologna, Bologna, Italy

Definition

In order to be able to evaluate beforehand the impact of a strategic or tactical move so as to plan optimal strategies to reach their goals, decision makers need reliable predictive systems. What-if analysis is a data-intensive simulation whose goal is to inspect the behavior of a complex system, such as the corporate business or a part of it, under some given hypotheses called scenarios. In particular, what-if analysis measures how changes in a set of independent variables impact a set of dependent variables with reference to a given simulation model such a model is a simplified representation of the business, tuned according to the historical corporate data. In practice, formulating a scenario enables the building of a hypothetical world that the analyst can then query and navigate.

Historical Background

Though what-if analysis can be considered as a relatively recent discipline, its background is rooted at the confluence of different research areas, some of which date back decades ago.

First of all, what-if analysis lends some of the techniques developed within the simulation community, to contextualize them for *business intelligence*. Simulations are used in a wide variety of practical contexts, including physics, chemistry, biology, engineering, economics, and psychology. Much literature has been written in this field over the years, mainly regarding the design of simulation experiments and the validation of simulation models [5,8,9].

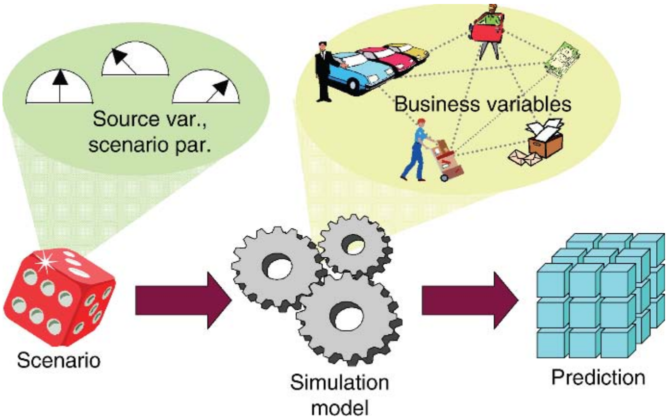
Another relevant field for what-if analysis is economics that provide the insights into business processes necessary to build and test simulation models. For instance, in [1] a set of alternative approaches to forecasting are surveyed, and useful guidelines for selecting the best ones according to the availability and reliability of knowledge are given.

Finally, what-if analysis relies heavily on database and *data warehouse* technology. Though data warehousing systems have been playing a leading role in supporting the decision process over the last decade, they are aimed at supporting analysis of past data ("what-was") rather than giving conditional anticipations of future trends ("what-if"). Nevertheless, the historical data used to reliably build what-if predictions are normally taken from the enterprise data warehouse. Besides, there is a tight relationship between what-if analysis and *multidimensional modeling* since input and output data for what-if analysis are typically stored within *cubes* [7]. In particular, in [2] the SESAME system for formulating and efficiently evaluating what-if queries on data warehouses is presented. Here, scenarios are defined as ordered sets of hypothetical modifications on multidimensional data. Finally, there are relevant similarities between simulation modeling for what-if analysis and the modeling of *Extraction, Transformation and Loading* applications; in fact, both ETL and what-if analysis can both be seen as a combination of elementary processes each transforming an input data flow into an output.

Foundations

As sketched in Fig. 1, a what-if application is centered on a *simulation model*, that establishes a set of complex relationships between some *business variables* corresponding to significant entities in the business domain (e.g., products, branches, customers, costs, revenues, etc.). A simulation model supports one or more *scenarios*, each describing one or more alternative ways to construct a *prediction* of interest for the user. The prediction typically takes the form of a multidimensional *cube*, whose *dimensions* and *measures*-correspond to business variables, to be interactively explored by the user by means of any On-Line Analytical Processing (OLAP) front-end. A scenario is characterized by a subset of business variables, called *source variables*, and by a set of additional parameters, called *scenario parameters*, that the user has to value in order to execute the model and obtain the prediction. While business variables are related to the business domain, scenario parameters convey information technically related to the simulation, such as the type of regression adopted for forecasting and the number of past years to be considered for regression. Distinguishing source variables among business variables is important since it enables the user to understand which are the "levers" that she can independently adjust to drive the simulation. Each scenario may give rise to different simulations, one for each assignment of the source variables and of the scenario parameters.

A simple example of a what-if query in the marketing domain is: *How would my profits change if I run a* 3×2 *(pay 2 and take 3) promotion for one week on all audio products on sale?* Answering this query requires building a simulation model capable of expressing the complex relationships between the business variables that determine the impact of promotions on product sales, and to run it against the historical sale data in order to determine a reliable forecast for future sales. In particular, the source variables for this scenario are the type of promotion, its duration, and the product category it is applied to. Possible scenario parameters could be the type of regression used for forecasting and the number of past years to be considered for regression. The specific simulation expressed by the what-if query reported in the text is determined by giving values "3 × 2," "one week" and "audio," respectively,



What-If Analysis. Figure 1. Functional sketch for what-if analysis.

to the three source variables. The prediction could be a *cube* with dimensions week and product and measures revenue, cost and profit.

Importantly, what-if analysis should not be confused with *sensitivity analysis*, aimed at evaluating how sensitive the behavior of the system is to a small change of one or more parameters. Besides, there is an important difference between what-if analysis and simple *forecasting*, widely used especially in the banking and insurance fields. In fact, while forecasting is normally carried out by extrapolating trends out of the historical series stored in information systems, what-if analysis requires simulating complex phenomena whose effects cannot be simply determined as a projection of past data.

On the other hand, applying forecasting techniques is often required during what-if analysis. In [4] the authors report a useful classification of forecasting methods into *judgmental*, such as those based on expert opinions and role-playing, and *statistical*, such as extrapolation methods, expert systems and rule-based forecasting. The applicability of these methods to different domains is discussed, and an algorithm for selecting the best method depending on the specific characteristics of the problem at hand is reported.

A separate mention is in order for *system dynamics* [4,11], an approach to modeling the behavior of nonlinear systems, in which cause-effect relationships between abstract events are captured as dependencies among numerical variables; in general, such dependencies can give rise to retroactive interaction cycles, i.e., feedback loops. From a mathematical standpoint, systems of differential equations are the proper tool for modeling such systems. In the general case, however, a solution cannot always be found analytically, so numerical techniques are often used to predict the behavior of the system. A system dynamics model consists of a set of variables linked together, classified as *stock* and *flow* variables; flow variables represent the rate at which the level of cumulation in stock variables changes. By running simulations on such a model, the user can understand how the system will evolve over time as a consequence of a hypothetical action she takes. She can also observe, at each time step, the values assumed by the model variables and (possibly) modify them. Thus, it appears that system dynamics can effectively support what-if applications in which the current state of any part of the system could influence its own future state through a closed chain of dependency links.

Designing a what-if application requires a methodological framework; the one presented in [6] relies on seven phases:

1. *Goal analysis*, aimed at determining which business phenomena are to be simulated, and how they will be characterized. The goals are expressed by (i) identifying the set of business variables the user wants to monitor and their granularity; and (ii) defining the relevant scenarios in terms of source variables the user wants to control.
2. *Business modeling*, which builds a simplified model of the application domain in order to help the designer to understand the business phenomenon as well as give her some preliminary indications about which aspects can be either neglected or simplified for simulation.
3. *Data source analysis*, aimed at understanding what information is available to drive the simulation and how it is structured.
4. *Multidimensional modeling*, which defines the multidimensional schema describing the prediction by taking into account the static part of the business model produced at phase 2 and respecting the requirements expressed at phase 1.
5. *Simulation modeling*, whose aim is to define, based on the business model, the simulation model allowing the prediction to be constructed, for each given scenario, from the source data available.
6. *Data design and implementation*, during which the multidimensional schema of the prediction and the simulation model are implemented on the chosen platform, to create a prototype for testing.
7. *Validation*, aimed at evaluating, together with the users, how faithful the simulation model is to the real business model and how reliable the prediction is. If the approximation introduced by the simulation model is considered to be unacceptable, phases 4–7 should be iterated to produce a new prototype.

The three modeling phases require a supporting formalism. Standard UML can be used for phase 2 (e.g., a use case diagram and a class diagram coupled with activity diagrams) and any formalism for conceptual modeling of multidimensional databases can be effectively adopted for phase 4. Finding a suitable formalism to give broad conceptual support to phase 5 is much harder, though some examples based on the use of colored Petri nets, event graphs and flow charts can be found in the simulation literature [10].

Key Applications

Among the killer applications for what-if analysis, it is worth mentioning profitability analysis in commerce, hazard analysis in finance, promotion analysis in marketing, and effectiveness analysis in production planning. Less traditional, yet interesting applications described in the literature are urban and regional planning supported by *spatial databases*, index selection in relational databases, and ETL maintenance in data warehousing systems.

Either spreadsheets or OLAP tools are often used to support what-if analysis. Spreadsheets offer an interactive and flexible environment for specifying scenarios, but lack seamless integration with the bulk of historical data. Conversely, OLAP tools lack the analytical capabilities of spreadsheets and are not optimized for scenario evaluation [2]. Recently, what-if analysis has been gaining wide attention from vendors of business intelligence tools. For instance, both SAP SEM (Strategic Enterprise Management) and SAS Forecast Server already enable users to make assumptions on the enterprise state or future behavior, as well as to analyze the effects of such assumptions by relying on a wide set of forecasting models. Also Microsoft Analysis Services provides some limited support for what-if analysis. This is now encouraging companies to integrate and finalize their business intelligence platforms by developing what-if applications for building reliable business predictions.

Future Directions

Surprisingly, though a few commercial tools are already capable of performing forecasting and what-if analysis, and some papers describe relevant applications in different fields, very few attempts have been made so far to address methodological and modeling issues in this field (e.g., [6]). On the other hand, facing a what-if project without the support of a design methodology is very time-consuming, and does not adequately protect the designer and his customers against the risk of failure. The main problem related to the design of what-if applications is to find an adequate formalism to conceptually express the simulation model, so that it can be discussed and agreed upon with the users. Unfortunately, no suggestion to this end is given in the literature, and commercial tools do not offer any general modeling support. Another relevant problem is to establish a general framework for estimating the loss of precision that is introduced when modeling low-level phenomena with higher-level dependencies. This could allow designers to assess the reliability of the prediction as a function of the quality of the historical data sources and of the precision of the simulation model.

Decision makers are used to navigating multidimensional data within OLAP sessions, that consist in the sequential application of simple and intuitive OLAP operators, each transforming a cube into another one. Consequently, it is natural for them to ask for extending this paradigm for accessing information also to what-if analysis. This would allow users to mix together navigation of historical data and simulation of future data into a single session of analysis. In the same direction, an approach has recently been proposed for integrating OLAP with data mining [3]. This raises an interesting research issue. In fact, OLAP should be extended with a set of new, well-formed operators specifically devised for what-if analysis. An example of such operator could be *apportion*, which disaggregates a quantitative information down a hierarchy according to some given criterion (*driver*). For instance, a transportation cost forecasted by branch and month could be apportioned by product type proportionally to the quantity shipped for each product type. In addition, efficient techniques for supporting the execution of such operators should be investigated.

Cross-references

▶ Business Intelligence
▶ Cube
▶ Data Warehousing Systems: Foundations and Architectures
▶ On-Line Analytical Processing

Recommended Reading

1. Armstrong S. and Brodie R. Forecasting for marketing. In Quantitative methods in marketing. G. Hooley and M. Hussey (eds.). Int. Thompson Business Press, London, 1999, pp. 92–119.
2. Balmin A., Papadimitriou T., and Papakonstantinou Y. Hypothetical Queries in an OLAP Environment. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 220–231.
3. Chen B., Chen L., Lin Y., and Ramakrishnan, R. Prediction cubes. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 982–993.
4. Coyle R.G. System Dynamics Modeling: A Practical Approach. Chapman and Hall, London, 1996.
5. Fossett C., Harrison D., and Weintrob H. An assessment procedure for simulation models: a case study. Oper. Res., 39(5): 710–723, 1991.

6. Golfarelli M., Rizzi S., and Proli A. Designing what-if analysis: towards a methodology. In Proc. ACM 9th Int. Workshop on Data Warehousing and OLAP, 2006, pp. 51–58.
7. Koutsoukis N.S., Mitra G., and Lucas C. Adapting on-line analytical processing for decision modeling: the interaction of information and decision technologies. Decis. Support Syst., 26(1):1–30, 1999.
8. Kreutzer W. System Simulation – Programming Styles and Languages. Addison Wesley, Reading, MA, 1986.
9. Law A.M. and Kelton W.D. Simulation Modeling and Analysis. McGraw-Hill Higher Education, Boston, MA, 1999.
10. Lee C., Huang H.C., Liu B., and Xu Z. Development of timed colour petri net simulation models for air cargo terminal operations. Comput. Ind. Eng., 51(1):102–110, 2006.
11. Roberts E.B. Managerial applications of system dynamics. Pegasus Communications, 1999.

While Loop

► Loop

Wide-Area Data Replication

► WAN Data Replication

Wide-Area Storage Systems

► Peer-to-Peer Storage

WIMP Interfaces

Stephen Kimani
CSIRO Tasmanian ICT Centre, Hobart, TAS, Australia

Definition

There exist many types of interaction styles. They include but are not limited to: command line interface, natural language, question/answer and query dialog, form-fills and spreadsheets, WIMP, and three-dimensional interfaces. The most common of the foregoing interaction styles is the WIMP. WIMP is an acronym for Windows, Icons, Menus and Pointers. Alternatively, it is an acronym for Windows, Icons, Mice and Pull-down menus. Examples of user interfaces that are based on the WIMP interaction style include: Microsoft Windows for PCs, MacOs for Apple Macintosh, various X Windows-based systems for UNIX, etc.

Historical Background

WIMP interfaces were invented at the SRI laboratory in California. The development of WIMP interfaces continued at Xerox PARC. The 1981 Xerox Star workstation is considered to be the first production computer to use the desktop metaphor, productivity applications and a three-button mouse. WIMP was popularized by the Apple Macintosh in the early 1980s. The interaction style/paradigm has now been copied by the Microsoft Windows operating system, Motif, the X Window System, etc. The rapid rise of Microsoft Windows has made WIMP interfaces become the dominant interface paradigm. WIMP interfaces have been improved with a set of new user interface widgets during the years. However, the basic structure of a WIMP interface usually does not change. WIMP interfaces typically present the work space using a desktop metaphor [5]. Everything is presented in a two dimensional space which has windows. The functionality of the application is made available through interface widgets such as: menus, dialog boxes, toolbars, palettes, buttons, etc.

Foundations

In this section is a description of the elements of the WIMP interface.

Windows

Windows are areas of the display that behave as if they were independent terminals. They are typically rectangular areas of the display that can be manipulated independently on the display screen. In most cases, the view of the contents of a window can be changed by scrolling or editing. Windows can contain text and/or graphics. They can be moved, resized, closed, maximized, or minimized (reduced to an icon). Many windows can be displayed on the screen simultaneously thereby allowing multiple separated tasks to be visible at the same time. The user can switch from one from one task to another by moving from one window to another.

The components of windows usually include:

- Scrollbars: They enable users to move the contents of the window up and down (vertical scrollbar), or from side to side (horizontal scrollbar).

- Title bars: They describe the name of the window.
- Status bar: It displays the status of the task/process in the window.
- Boxes/widgets for resizing and closing the window.

Oftentimes, all windows in use will not fit on the screen space at once. Several strategies exist for managing multiple windows:

1. Iconification/minimizing: This strategy allows screen space to be saved by reducing windows to window icons. The user can re-expand the icons at will. A window icon therefore serves as a visual reminder of the window.
2. Tiling: In this case, the system uses all the available screen space to display the windows. The windows do not overlap. Tiling can take many forms. For instance: some systems use a fixed number of tiles while others allow variable numbers of tiles.
3. Overlapping: This strategy is probably now the most popular. In the strategy, windows are allowed to partially obscure each other like overlapping papers arranged on a desk. Cascading is a form of overlapping where the windows are automatically arranged fanned out, usually in a diagonal line so that the title and one other border of each window can be seen. With cascading, many windows can therefore be displayed in a limited screen space at the same time (Figs. 1 and 2).

When multiple windows are displayed on the screen at the same time, the active window is usually distinguished by a shaded/bold title bar. The active window is said to have focus. Windows have three size states: maximized, minimized/iconified and normal/restored. Some systems support windows within windows e.g., in Microsoft PowerPoint (MS Office 2000).

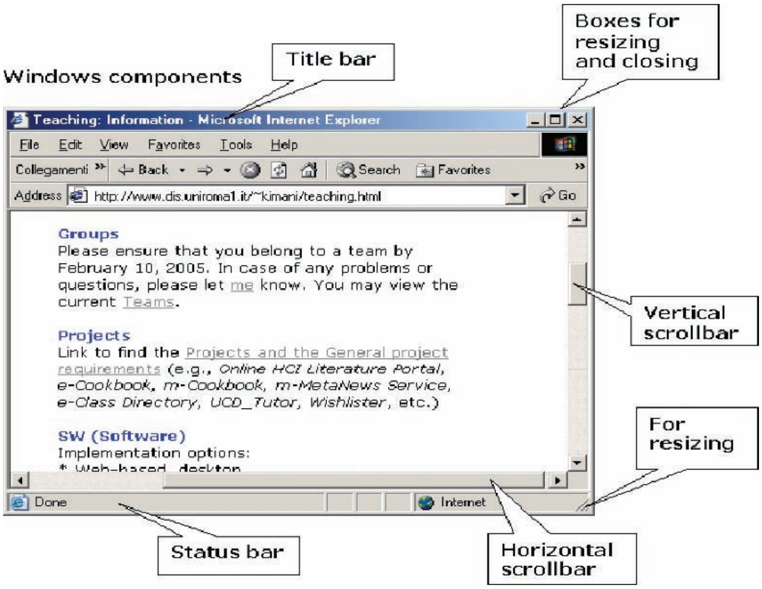
Icons

In the context of WIMP, an icon is a small picture or image, used to represent some aspect of the system (such as a printer icon to represent the printing action) or to represent some entity/object (such as a window) (Fig. 3).

As it was indicated earlier, a window may among other things be closed completely. Alternatively it may be reduced to this small representation (iconified/minimized). An icon therefore can save screen space and therefore many windows can be available on the screen simultaneously. Moreover, it can serve as a reminder to the user that s/he can resume dialog (or interaction with the represented application) by simply opening up the window.

Pointers

A pointer is a cursor on the display screen. It is worth noting that interaction in WIMP interfaces relies a lot on pointing and selecting interaction objects such as

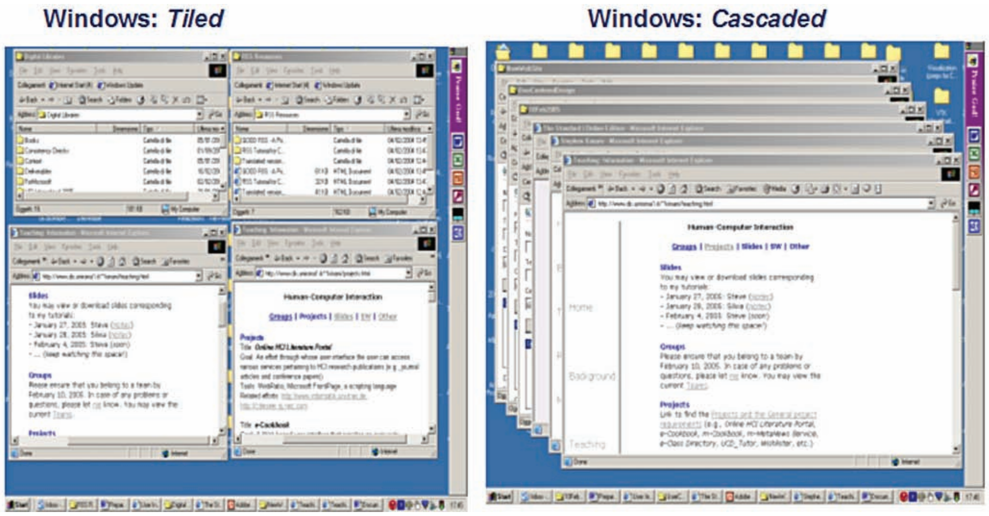

Windows components
Title bar
Boxes for resizing and closing
Teaching: Information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Collegamenti »
Back
Search
Favorites
Address
http://www.dis.uniroma1.it/~kimani/teaching.html
Go
Vertical scrollbar
For resizing
Done
Internet
Status bar
Horizontal scrollbar

Groups
Please ensure that you belong to a team by February 10, 2005. In case of any problems or questions, please let me know. You may view the current Teams.

Projects
Link to find the Projects and the General project requirements (e.g., *Online HCI Literature Portal, e-Cookbook, m-Cookbook, m-MetaNews Service, e-Class Directory, UCD_Tutor, Wishlister*, etc.)

SW (Software)
Implementation options:
* Web-based, desktop

WIMP Interfaces. Figure 1. Common components of a window.



WIMP Interfaces. Figure 2. Tiling versus cascading.

WIMP Interfaces. Figure 3. Examples of icons.

icons and menu items. Pointers are therefore important in WIMP interfaces. There are at least two types of cursors: mouse cursor and text cursor. A mouse cursor shows the user where the current position of the mouse is considered to be with respect to the windows on screen. Usually, the shape and behavior of the mouse cursor can be changed. A text cursor shows where input will be directed from the keyboard. The mouse is the most common device for pointing and clicking. However, other input devices such as joystick, trackball, cursor keys or keyboard shortcuts too can be used for such pointing and selecting tasks. In a particular system, different cursors are used to represent different modes/states e.g., normal cursor as an arrow, double-arrowed cursor for resizing windows, hour-glass when system is busy, etc. A hot-spot is the location to which the cursor points. Selection occurs at the coordinate of the hot-spot. The hot-spot of the cursor should be obvious to the user (Fig. 4).

Menus

A menu is an interaction feature that presents a set of options displayed on the screen where the selection and execution of one (or more) of the options results in a change in the state of the interface [8]. The human beings' ability to recognize information on being given a visual cue is superior to their ability to recall the information. Menus therefore can serve as cues for the operations or services that the system can perform. Therefore, the labels/names used for the commands in menus should be informative and meaningful. A potential candidate for selection can be indicated by using a pointing device (e.g., mouse, arrow keys, etc) to move the pointer accordingly. As the pointer moves, visual feedback is typically given by some kind of visual emphasis such as highlighting the menu item. Selection of a menu item can be realized by some additional user action (such as pressing a button on the pointing devices, pressing some key on the keyboard, etc). Keyboard accelerators, which are key combinations that have the same effect as selecting the menu item, are sometimes offered. When there are too many items, the menu items are often grouped and layered. The main menu, usually represented as a menu bar, should be conspicuous or readily available/accessible. Menu bars are often placed at one of the sides of the screen or window. For instance: at the top of the screen (e.g., MacOS), at the top of each window (e.g., Microsoft Windows). Most systems show the currently selected state of any group of menu items by displaying them in bold or by a tick. Entries that are disallowed in the current context are often shown in a dimmed font.

Types of menus:

- Pull-down menus: They are attached to a main menu under the title bar or to buttons. Pull-down menus are dragged from the main menu by

WIMP Interfaces. Figure 4. Examples of pointers.

moving the pointer into the menu bar and pressing the button.

- Fall-down menus: They automatically appear from the main menu when the pointer enters the menu bar, without having to press the button.
- Pop-up menus: They appear when a particular region of the screen or window, maybe designated by an icon, is selected, but they remain in position until the user instructs it to disappear again e.g., by clicking on a "close box" in the border of the menu's window, by releasing the mouse button, etc.
- Pin-up menus: They can be "pinned" or attached to the screen, staying in place until explicitly asked to go away.
- Pie menus: The menu options in a pie menu have a circular arrangement with the pointer at the center.

There are two main challenges with menus: deciding which items to include and how to group those items. Here are some guidelines on menu design:

- A menu label in pull-down menus should reflect the functions of the underlying menu items.
- The items in pull-down menus should be grouped by function.
- Menu groupings in pull-down menus should be consistent (to facilitate the transfer of learning and confidence to the user).
- Menu items should be ordered in the menu according to importance and frequency of use.
- Opposite functionalities (e.g., "save" and "delete") should be kept apart to prevent accidental selection of the wrong function.

Additional Interaction Elements

Many other interaction elements may be present in a WIMP interface. For instance: buttons, sliders, toolbars, palettes, dialog boxes, etc. Buttons are individual and isolated regions within a display that can be selected by the user to invoke a specific operation or state. Sliders are used to set quantities that vary continuously within given limits or even to enable the user to choose between large numbers of options [9]. A toolbar is a collection of small buttons, each with icons, that provides convenient access to commonly used functions. It should be pointed out that although the function of a toolbar is similar to that of a menu bar, the icons in a toolbar are smaller than the equivalent text and therefore more functions can be simultaneously displayed. A palette is a mechanism for making the set of possible modes and the active/current mode visible to the user. A dialog box is an information window used by the system to provide contextual information.

Key Applications

WIMP interfaces have made computer usage more accessible to users who previously could not use the earlier types of user interfaces. Such users include: young children (who cannot yet read or write), managers, and non-professional home users [11]. WIMP interfaces can be credited for the increased emphasis on the incorporation of user interface design and usability evaluation in the software development process in the late 1980s and early 1990s. On the whole, WIMP interfaces can be said to be instrumental in the realization of applications that are characterized by relative ease of learning, ease of use, and ease of transfer of knowledge due to the consistency in WIMP-based designs [11].

Cross-references

▶ Human-Computer Interaction
▶ Icon
▶ Visual Interaction
▶ Visual Interfaces
▶ Visual Metaphor

Recommended Reading

1. Alistair E. The design of auditory interfaces for visually disabled users. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1988, pp. 83–88.
2. Balakrishnan R. and Kurtenbach G. Exploring bimanual camera control and object manipulation in 3D graphics interfaces.

In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1999, pp. 56–62.

3. Beaudouin-Lafon M. Designing interaction, not interfaces. In Proc. Working Conf. on Advanced Visual Interfaces, 2004, pp. 15–22.
4. Beaudouin-Lafon M. and Lassen H.M. The architecture and implementation of CPN2000, a post-WIMP graphical application. In Proc. 13th Annual ACM Symp. on User Interface Software and Technology, 2000, pp. 181–190.
5. Cesar P. Tools for adaptive and post-WIMP user interfaces. New Directions on Human Computer Interaction, 2005.
6. Dix A., Finlay J., Abowd G., and Beale R. Human-Computer Interaction. Prentice Hall, Englewood Cliffs, NJ, 2003.
7. Green M. and Jacob R. SIGGRAPH: '90 Workshop report: software architectures and metaphors for non-WIMP user interfaces. Comput. Graph., 25(3): 229–235, 1991.
8. Paap K.R. and Roske-Hofstrand R.J. Design of menus. In Handbook of Human-Computer Interaction, M. Helander (ed.). Amsterdam, North-Holland, 1998.
9. Preece J., Rogers Y., Sharp H., Benyon D., Holland S., and Carey T. Human-Computer Interaction. Addison-Wesley, Reading, MA, 1994.
10. Odell D.L., Davis R.C., Smith A., and Wright P.K. Toolglasses, marking menus, and hotkeys: a comparison of one and two-handed command selection techniques. In Proc. 2004 Conf. on Graphics Interface, 2004, pp. 17–24.
11. van Dam A. Post-WIMP user interfaces. Commun. ACM, 40(2):63–67, 1997.

Window-based Query Processing

WALID G. AREF
Purdue University, West Lafayette, IN, USA

Synonyms

Stream query processing

Definition

Data Streams are infinite in nature. As a result, a query that executes over data streams specifies a "window" of focus or the part of the data stream that is of interest to the query. When new data items arrive into the data stream, the window may either expand or slide to allow the query to process these new data items. Hence, queries over data streams are continuous in nature, i.e., the query is continuously re-evaluated each time the query window slides. Window-based query processing on data streams refers to the various ways and techniques for processing and evaluating continuous queries over windows of data stream items.

Historical Background

Windows over relational tables have already been introduced into Standard SQL (SQL:1999) in order to support data analysis, decision support, and more generally, OLAP-type operations.

However, the motivation for having windows in data stream management systems is quite different. Since data streams are infinite, it is vital to limit and focus the scope of a query to a manageable and finite portion of the data stream. Earlier works on window query processing have focused on processing tuple-count and time-sliding windows. Ever since, window query processing techniques have been developed to deal with out-of-order tuple arrivals, e.g., [14], revision tuple processing, e.g., [1,13], incremental evaluation techniques, e.g., [6], stream punctuation techniques, e.g., [17], multi-query optimization techniques, e.g., shared execution of window queries [4,7,9,10], and adaptive stream query processing techniques, e.g., [12].

Foundations

Queries over data streams are continuous in nature. A continuous query progressively produces results as new data items arrive into the data stream. Since data streams are infinite, queries that execute over a data stream need to define a region of interest (termed a window). There are several ways by which a query can define its window(s) of interest. These include stream-based versus operation-based windows, tuple-count versus time-sliding windows, and sliding versus predicate windows. These ways for specifying windows are orthogonal and can hence be combined. For example, a window can be time-sliding and at the same time, operation-based. Similarly, a stream-based window can also be predicate-based, etc.

Query Processing Techniques: Incremental Evaluation Versus Reevaluation

Whenever the data items within a window change, e.g., when the window slides with the arrival of new tuples or with the expiration of some old tuples from the window, a continuous query's answer needs to be reproduced. There are two mechanisms for reproducing the answer to a continuous query, namely (i) query reevaluation and (ii) incremental evaluation, which are explained in the following sections.

Query Reevaluation

In query reevaluation, a window over a data stream is viewed as an instantiation of a table that contains the data tuples in the window. When the window slides, a new table is formed (or opened). So, from the point of view of a continuous query, a data stream is a sequence of tables. With the arrival of a new table (window of data stream items), the query is reevaluated to produce a new output result. As a consequence, an important feature of query reevaluation is that the semantics of the traditional query processing operators, e.g., selects and joins, do not change. When the slide of a window is less than its range, multiple overlapping windows will be open concurrently over the same stream. Hence, it is possible that a newly arriving tuple contribute to multiple windows.

Query Processing using Revision Tuples At times, data stream items may be noisy or erroneous due to the nature of the data streaming applications. As a result, stream data sources may need to "revise" previously issued tuples in the form of "revision tuples" [1,13]. So, in a sense, processing a revised tuple may involve reprocessing of the input tuples that were in the same window or computation as the tuple being revised to produce revised output results. Consequently, some query operators will need to preserve "state" in order to reprocess the revised input tuples. For example, aggregates may need to store the individual values (or some summary of these values) that were used to produce the aggregate result so that the aggregate operator may be recomputed given the revised input tuple. This approach is referred to as "upstream processing." It is possible to go downstream, i.e., from the output tuples backwards. The idea is to correct the previously generated output tuples using the new and original values of the revised tuple as well as some state information depending on the nature of the participating query operators. More detail about processing revision tuples can be found in [1,13].

Query Processing using Punctuations Some stream query operators are stateful while others are blocking. Since data streams are infinite, the state of these query operators may be unbounded in size. Similarly, blocking operators cannot wait indefinitely as the data streams never end. Knowing more a priori knowledge about the stream semantics, it is possible to embed, within the data stream, special annotations, termed "Punctuations" that break the infinite data stream into finite sub-streams. Then, stateful and blocking query operators can be applied successfully to each of the substreams.

Stream punctuations can take multiple forms. For example, one stream can send a "last-reading-within-this-hour" punctuation to reflect that no more readings for this hour are expected from that source. In this case, a blocking operator can produce results related to this source once the operator receives this punctuation. Similarly, in a bidding application, sending a "no-more-bids-for-item:%itemid" punctuation would allow the bid to be finalized for that item. In general, a punctuation can be viewed as a predicate that evaluates to "false" for every tuple in the stream that follows the punctuation. This helps query operators generate output tuples following a punctuation as well as help reduce the size of the state kept per operator. Detailed discussion about query processing using punctuations can be found in [17].

Query Processing Using Heartbeats Stream data sources often assign a timestamp to each data element they produce. Due to the distributed nature of the stream data sources, data elements may arrive to the data stream management system out of order. System buffers need to store the out-of-order tuples to present them later to the query processor in increasing timestamp order. Before processing a tuple t with timestamp ts, it is important to guarantee that no more tuples with timestamp less than ts will arrive to the system. Stream heartbeats are a mechanism that provide such assurance. Each data source is responsible for generating its own heartbeats periodically and embeds them within its own data stream. In cases when a data source does not have this capability, the data stream management system should be able to synthesize and generate heartbeats for each data stream based on parameterized knowledge of the environment, e.g., the maximum network delay.

When processing a query that accesses multiple streams, say s_1, s_2,...,s_n, the query processor computes the minimum timestamp, say ts_{min}, of the heartbeats of all the n streams. All the tuples in the system buffer with timestamps less than ts_{min} are forwarded to the query for processing. Such query-level heartbeats are simple and easy to implement. However, they can block the processing of a query unnecessarily, e.g., when one of the streams is temporarily blocked while the others are available. Alternatively, in contrast to a query-level heartbeat, in an operator-level heartbeat, the query processor computes the minimum timestamps of heartbeats for streams input to each query operator in the plan. In this case, each operator will have its own heartbeat.

Input tuples with timestamps less than an operator's heartbeat are forwarded to that operator. Latency and memory requirements of both query-level and operator-level heartbeats are further detailed in [14].

Incremental Query Evaluation

In incremental query evaluation, whenever the window contents change, the query does not reprocess all the tuples inside the window. Instead, the query only processes the changed tuples. Moreover, only the changes to the query answer are reported. The answer to the query is considered more as a materialized view in the sense that only the changes in the base tables (the deltas) need to be processed by the view expression. As a result, some new tuples get inserted into the view while others get deleted from it.

Two types of events need to be handled in the case of incremental query evaluation. The first event type is when a new tuple arrives into the data stream and hence becomes inside of the stream's window. The second event type is when a tuple exits the window. For example, when a time-sliding window slides, a tuple's timestamp may become outside of the time interval covered by the window, and hence the tuple expires and exits the window.

Systems vary on how they handle both types of events. STREAM [16] generates two types of streams: an Insert (I) stream that contains the newly arriving tuples, and a Delete (D) stream that contains the expiring tuples. In contrast to STREAM that has two types of streams, Nile [8] maintains only one stream but with two types of tuples: Positive and Negative Tuples. Positive tuples correspond to the new tuples that arrive into the window while negative tuples correspond to the expiring tuples. Positive tuples are the regular tuples that a traditional query processor handles. In contrast, negative tuples are processed differently by each query operator.

Incremental Query Processing Using Negative Tuples

In a traditional relational query evaluation plan (QEP), a table-scan operator is usually at the bottom (leaf) level of the QEP and serves as the interface operator that pipelines the tuples of a relation to the rest of the QEP. In incremental query evaluation, the equivalent to a table-scan operator is the window-expire operator (W-expire, for short). A W-expire operator is assigned to each stream in the query and is added at the bottom (leaf) level of a QEP. The inputs to the W-expire operator are the raw stream tuples as well as the definition of the window (e.g., say a window of range 10 min and slide 2 min). The function of the W-expire operator is two-fold: (i) introduce the new stream tuples to the QEP as they arrive into the stream, and (ii) generate (or synthesize) negative tuples that correspond to stream tuples that expire from the window and introduce them to the QEP.

Conceptually, a negative tuple, say t^-, needs to trace the same path in the QEP that its corresponding positive tuple, say t^+, took to produce the same results that t^+ produced (but with negative sign). Once it receives a negative tuple, each query operator, in turn, processes the negative tuple and produces 0 or more negative tuples as output that get processed by the operators next in the pipeline. The semantics of all query operators are extended to handle positive as well as negative tuples. Additionally, each query operator needs to store additional state information to be able to process negative tuples successfully. This is similar to the case of processing revision tuples in Borealis. Details about the extended semantics and state information of query operators for incremental query processing as well as optimizations to reduce the cost of handling negative tuples can be found in [6].

Query Processing for Predicate Windows

In contrast to a sliding window, in a predicate window, a window is specified by a predicate that defines the stream data tuples of interest to a query. For example, a query may define its window of interest to be the room identifiers of the rooms with temperatures greater than 110 F. When a room's temperature is below 110 F, that room identifier exits the window, otherwise it remains inside the window of interest.

Predicates that define a window may greatly vary in complexity. In general, a predicate window can be expressed using a regular select-from-where query. Consider the predicate window that is defined by following Select query over Stream roomTemperatures: Select room-id, temperature from roomTemperatures where temperature >120. The room-ids and temperature values of the high-temperature rooms are maintained inside this predicate window.

Three types of tuples need to be maintained in this scenario (insert, delete, and update tuples). Insert and delete tuples are the same as the positive and negative tuples used for incremental evaluation, while update tuples are similar to revision tuples. When the temperature of Room 5 increases from 110 to 130 F (i.e., the room did not have a high temperature before), the corresponding tuple (room-id, temperature): (5, 130) enters into the predicate window via an insert tuple. Similarly, when the temperature of the same room changes from 130 to 140 F, the corresponding tuple (5, 140) replaces the old tuple (5, 130) via an update tuple. Finally, when the temperature of Room 5 goes below 120 F, e.g., 70 F, the corresponding tuple (5, 140) is eliminated from the predicate window via a delete tuple. Because of the generality of predicate windows and the use of insert, delete, and update tuples, predicate windows can be used to support views over data streams [5].

Shared Execution of Multiple Window-based Queries

In continuous stream query processing, it is imperative to exploit resources and commodity among query expressions to achieve improved efficiency. Shared execution is one important means of achieving this goal. The role of joins in particular is further enhanced in stream query processing due to the use of "selection pull up." In the NiagaraCQ system [9,10], it was shown that the traditional heuristic of pushing selection predicates below joins is often inappropriate for continuous query systems because early selection destroys the ability to share subsequent join processing. Since join processing is expensive relative to selections, it is almost always beneficial to process the join once in shared mode and then subject the produced join output tuples to the appropriate selection operations. A similar argument holds for aggregation operations and Group-by.

Multiple windowed query operators, e.g., window joins, can share their execution as the windows of interest over one data stream overlap. For example, consider the two streams A and B and the two join operators J and K that use the same join predicate, e.g., A.a = B.b. Assume an operation-based time-sliding windows of sizes w_J for J and w_K for K, e.g., $w_J = 1$ h or $w_K = 1$ day. Since J and K have the same join predicate and $w_J < w_K$, the join output tuples of J are a subset of the join output tuples. Notice that it is possible that two tuples, say A_1 and B_1, join and their timestamps are more than 1 h apart but less than 1 day. Hence, the join output pair $<A_1, B_1>$ is part of K's join output tuples but is not part of J's. Executing both operations separately wastes system resources. Sharing the execution of the join operator would save on CPU and memory resources. One feasible policy is to perform only the join with the largest window (K in the example, since w_K is the larger of the two windows) [4,12]. The output of this join is then routed to multiple output streams (two streams in the example), where the output tuples for the joins with smaller windows get filtered out based on their timestamps. One advantage of this policy is its simplicity since arriving tuples are completely processed before considering the next incoming tuple. However, this policy has a disadvantage in that it delays the processing of small-window joins until the largest-window join is completely processed, hence negatively affecting the response time of small-window joins.

An alternative policy for shared execution of window joins is to perform the join with the smallest window first by all new tuples, then the next larger window joins, and so on until the largest window join is processed [7]. As long as there are tuples to be processed by the smallest window, they are processed first before any tuples are processed by the larger windows. Notice that the join output pairs produced for the smallest windows are also directly output tuples for all the larger windows, but the converse is not true, as explained above. The response time for the small-window joins is significantly enhanced. However, there is significant state information that needs to be maintained for each tuple to know at what point inside the windows from which that tuple resumes execution with one of the window joins. More detail and tradeoffs of each of the policies as well as better optimized policies for shared execution of joins can be found in [7].

Out-of-order Tuple Processing

In processing queries over data streams, it is often the case that the window operators are order sensitive. For example, when continuously computing the running average over the most-recent ten tuples in a stream, the output result depends on the order of the tuples in the stream. With the ordered arrival of a new tuple, the 11th most-recent tuple gets dropped and the new average is computed. The same is true for time-sliding windows. When tuples arrive in order, the timestamp of the new tuple is used to slide the window. This is not the case when tuples arrive out of order. In this case, none of the windows can get closed and any produced answer pertaining to a given window cannot be considered final since at any future point in time, a tuple can arrive into that window and hence the output needs to be recalculated.

There are several ways to deal with out-of-order tuples. As explained in Section "Query Processing using Revision Tuples," revision tuples are one way to deal with out-of-order tuples. The timestamp, say TS, of the most-recent tuple of a stream is maintained. Whenever an out-of-order tuple arrives (this tuple's timestamp is less than TS), a revision tuple is issued to revise the output of the affected window(s). When the maximum possible delay of a tuple is known in advance, the stream query processor can leave open all the potential windows that are within this delay window. Once the maximum delay is reached, the corresponding windows get closed and the tuples inside the window are processed to produce the final answers for that window. The input stream manager can synthesize and issue a punctuation or a heartbeat to that effect so that the window results can be computed and finalized.

Key Applications

Key applications for window-based stream query processing include computer network traffic analysis, network performance monitoring, intrusion detection. stock market data analytics, web browsing clickstream analytics, algorithmic stock trading, sensor network data summarization and processing, moving object tracking, traffic monitoring, and surveillance applications.

Cross-references

- ▶ Adaptive Stream Processing
- ▶ Continuous Queries in Sensor Networks
- ▶ Continuous Query
- ▶ Data Stream
- ▶ Distributed Streams
- ▶ Event Stream
- ▶ Fault-Tolerance and High Availability in a Data Stream Management Systems
- ▶ Histograms on Streams
- ▶ Publish/Subscribe over Streams
- ▶ Stream Models
- ▶ Stream Processing
- ▶ Stream-Oriented Query Languages and Operators
- ▶ Windows
- ▶ XML-Stream Processing

Recommended Reading

1. Abadi D., Ahmad Y., Balazinska M., Cetintemel U., Cherniack M., Hwang J-H., Lindner W., Maskey A.S., Rasin A., Ryvkina E., Tatbul N., Xing Y., and Zdonik S. The design of the Borealis stream processing engine. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 277–289.
2. Abadi D., Carney D., Cetintemel U., Cherniack M., Convey C., Lee S., Stonebraker M., Tatbul N., and Zdonik S. Aurora: a new model and architecture for data stream management. VLDB J., 12(2):120–139, 2003.
3. Bai Y., Thakkar H., Luo C., Wang H., and Zaniolo C. A data stream language and system designed for power and extensibility. In Proc. Int. Conf. on Information and Knowledge Management, 2006, pp. 337–346.
4. Chandrasekaran S. and Franklin M.J. Streaming queries over streaming data. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 203–214.
5. Ghanem T.M. Supporting Views in Data Stream Management Systems. Ph.D. Dissertation. Department of Computer Science, Purdue University, 2007.
6. Ghanem T.M., Hammad M.A., Mokbel M.F., Aref W.G., and Elmagarmid A.K. Incremental evaluation of sliding-window queries over data streams. IEEE Trans. Knowl. Data Eng., 19(1): 57–72, 2007.
7. Hammad M.A., Franklin M.J., Aref W.G., and Elmagarmid A.K. Scheduling for shared window joins over data streams. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 297–308.
8. Hammad M.A., Mokbel M.F., Ali M.H., Aref W.G., Catlin A.C., Elmagarmid A.K., Eltabakh M., Elfeky M.G., Ghanem T., Gwadera R., Ilyas I.F., Marzouk M., and Xiong X. Nile: a query processing engine for data streams. In Proc. 20th Int. Conf. on Data Engineering, 2004, p. 851.
9. Jianjun C., DeWitt D.J., Feng T., and Yuan W. NiagaraCQ: a scalable continuous query system for internet databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 379–390.
10. Jianjun C., DeWitt D.J., and Naughton J.F. Design and evaluation of alternative selection placement strategies in optimizing continuous queries. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 345–356.
11. Johnson T., Muthukrishnan S., Shkapenyuk V., and Spatscheck O. A heartbeat mechanism and its application in gigascope. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 1079–1088.
12. Madden S., Shah M.A., Hellerstein J.M., and Raman V. Continuously adaptive continuous queries over streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 49–60.
13. Ryvkina E., Maskey A.S., Cherniack M., and Zdonik S. Revision processing in a stream processing engine: a high-level design. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
14. Srivastava U. and Widom J. Flexible time management in data stream systems. In Proc. 23rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2004, pp. 263–274.

15. Stonebraker M., Cetintemel U., and Zdonik S. The 8 requirements of real-time stream processing. ACM SIGMOD Rec., 34 (4):42–47, 2005.
16. The STREAM Group. STREAM: the Stanford stream data manager. IEEE Data Eng. Bull., 26(1):19–26, 2003.
17. Tucker P.A., Maier D., Sheard T., and Fegaras L. Exploiting punctuation semantics in continuous data streams. IEEE Trans. Knowl. Data Eng., 15(3):555–568, 2003.

Windows

Carlo Zaniolo
University of California-Los Angeles, Los Angeles, CA, USA

Synonyms

Logical window = Time-based window; Physical window = Tuple-based windows

Definition

Windows were introduced as part of SQL:1999 OLAP Functions. For instance, given a sequence of bids it is possible to use the following SQL:2003 statement to find the last 40 offers (the current offer and the previous 39) for item 0021:

```
SELECT itemID, avg(Offer)
OVER(ROWS 39 PRECEDING ORDER BY TIME)
FROM BIDS
WHERE ItemID=0021
```

When BIDS is instead a data stream, the "ORDER BY TIME" clause becomes redundant, and clauses such as "FOLLOWING" are often not supported in continuous query languages. However, these languages still provide a "PARTITION BY" clause (or the more traditional "GROUP BY" clause), whereby a user can specify that the average of the last 40 offers must be computed for all items, not just item 0021. In addition to entailing powerful and flexible analytical queries on ordered sequences and time-series, as in databases, windows on data streams play the key role of *synopses*, and are widely employed in this capacity. In particular, since it is not feasible to memorize unbounded data streams, window joins are used instead. In window joins, the newly arriving tuples in one data stream are joined with the recent tuples in the window of the other data streams (and symmetrically). The design and implementation of window-based extensions for aggregates and joins operators for data streams has generated significant research work [1,2,3].

Key Points

Traditional SQL aggregates are blocking (and thus not suitable for continuous queries on data streams) but their window versions are not. Therefore, the window concept is the cornerstone of many continuous query languages, where its functionality has also been extended with new constructs, such as *slides*, *tumbles*, and *landmark* windows, that are not in SQL:2003. In a sliding window of w tuples (seconds) the aggregate computed over w is returned for each new incoming tuple: when a slide of size s is also specified, then results are only returned every s tuples (seconds). With $s = w$ we have a tumbling window in which results are only returned at the end of each window. When $w/s = k$, then the window, and the computation of the aggregate, is partition into k panes [2]. A landmark window is one where an occurrence of some event of semantic significance, e.g., a punctuation mark [3], defines one or both endpoints. Efficient implementation requires delta computations that exploit the algebraic properties of aggregates – e.g., by increasing (decreasing) the current sum with the value from the tuple entering (leaving) the window – and architectures that consolidate the vast assortment of windows and aggregates at hand [1,2]. Windows provide the basic synopsis needed to support joins with limited memory. Special techniques are used to optimize multi-way joins, and response time for all joins. Further optimization issues occur when load-shedding is performed by either (i) using secondary store to manage overflowing window buffers, or (ii) dropping tuples from the windows in such a way that either a max-subset or a random sample of the original window join is produced. Aged-based policies, where older tuples are dropped first, is preferable in certain applications. Sketching techniques are often used to estimate the productivity of tuples, whereby the least productive tuples are dropped first [3]. The same techniques can also be used to estimate duplicates in DISTINCT aggregates; reservoir-sampling inspired techniques have instead been proposed for aggregates where duplicates are not ignored.

Cross-references

▶ Data Sketch/Synopsis

▶ Join

- ▶ One-Pass Query Processing
- ▶ Punctuations
- ▶ SQL
- ▶ Stream Processing

Recommended Reading

1. Bai Y. et al. A data stream language and system designed for power and extensibility. In Proc. Int. Conf. on Information and Knowledge Management, 2006, pp. 337–346.
2. Li J. et al. Semantics and evaluation techniques for window aggregates in data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 311–322.
3. Maier D., Tucker P.A., and Garofalakis M. Filtering, punctuation, windows and synopses. In Stream Data Management, Vol. 30, N. Chaudhry, K. Shaw, M. Abdelguerfi (eds.). Kluwer, Dordecht, 2005, pp. 35–56.

Wireless Sensor Networks

▶ Sensor Networks

Within-Element Term Frequency

▶ Term Statistics for Structured Text Retrieval

Word Conflation

▶ Stemming

Word of Mouth

- ▶ Reputation and Trust
- ▶ Trust and Reputation in Peer-to-Peer Systems

Work Element

▶ Activity

Work Performer

▶ Actors/Agents/Roles

Workflow

▶ Workflow Management

Workflow Constructs

Nathaniel Palmer
Workflow Management Coalition, Hingham,
MA, USA

Synonyms

Process semantics

Definition

The elements of a Workflow Model defining how the process is executed.

Key Points

Workflow Constructs represent the elements of a Workflow Model, such as Control Data, Splits, Joins, Activities and Actors, which combined define the parameters of how a process instances is executed. The “Semantics” of the process refer to how it is executed in the context of workflow rules, such the steps, sequencing and dependencies; whereas the “Syntax” of the process refers to its definition within the context of a specific language. Workflow Constructs is directly related to semantics, however, does not define specific syntax.

Cross-references

- ▶ Activity
- ▶ Actors/Agents/Roles
- ▶ Control Data
- ▶ Join
- ▶ Split
- ▶ Workflow Model

Workflow Control Data

▶ Control Data

Workflow Enactment Service State Data

▶ Control Data

Workflow Engine State Data

► Control Data

Workflow Evolution

PETER DADAM, STEFANIE RINDERLE
University of Ulm, Ulm, Germany

Synonyms

Process evolution; Schema evolution in workflow management systems; Schema evolution in process management systems; Adaptive workflow/process management; Workflow/process instance changes

Definition

The term evolution has been originally used in biology and means the progressive development of a species over time, i.e., the adaptation to changing environmental requirements. Business processes (which are often called workflows when implemented and thus automated within a workflow management system) also "live" within an environment (e.g., the enterprise or the market). This environment is typically highly dynamic and thus the running workflows have to adapt to these changing requirements – i.e., to *evolve* – in order to keep up with the ever-changing business environment and provide their users with the competitive edge.

Workflow evolution implies two basic challenges: *change realizatio*n and *change discovery*. Change realization means that it must be *technically* possible to adapt workflows. Change discovery refers to the question which workflow modifications are required when the environmental requirements change. So far, research has focused on the first topic whereas the second one is subject to future research.

Regarding the technical realization, workflow changes can take place at two levels – at the workflow instance and the workflow type level. Instance-specific changes are often applied in an ad-hoc manner and become necessary in conjunction with real-world exceptions. They usually affect only single workflow instances. As opposed to this, in conjunction with workflow schema changes at the workflow type level, a collection of related instances may have to be adapted, i.e., workflow schema changes are *propagated* to running workflow instances or running workflow instances are *migrated* to the changed workflow schema.

Historical Background

In research workflow evolution has been mainly addressed from a technical point of view so far. More precisely, different approaches have been developed for the realization of workflow change during the last 10 years [8]. These approaches can be divided into different categories, i.e., pre-planned workflow changes, flexibility by design, and flexible workflow management technology.

Pre-planned workflow changes: In this category workflow changes are planned in advance [3,6]. Then, based on different strategies, one or several of these changes are executed during runtime if necessary (e.g., if an exception occurs). Strategies to initiate a workflow change can be rule-based (e.g., using ECA rules), goal-based (e.g., using AI planning techniques), or process-driven (e.g., based on graph grammars).

Flexibility by design: Workflow adaptations can be also foreseen in the workflow description [1,12] (e.g., in the workflow graph). The approaches range from simply modeling alternative branches with associated selection codes to approaches which specify workflows only as a rough "skeleton" in advance and leave the detailed specification of the workflow to the user at runtime. Then, for example, the user can select which activities are to be executed in which order. Some approaches also integrate the support of planning methods from Artificial Intelligence for the runtime workflow specification.

Flexible workflow management technology: Approaches in this field either deal with the modification of workflow instances [7] or with workflow schema evolution (i.e., changes at workflow type level) [2,11,13,14]. Basic questions, however, are very similar for workflow schema and instance changes, mainly concerning the correctness of the workflow management systems after changes at either level have been carried out. Consequently, different change frameworks on workflows have been defined [15] and several correctness criteria have been developed which partly depend on the used workflow meta model (e.g., Petri Nets) [8]. Whereas the correctness question is more or less sufficient to deal with workflow instance changes, other approaches have discovered that, for workflow schema changes, additional challenges arise. The reason is that if a workflow schema is modified, a possibly large number of workflow instances running according to the workflow schema is affected as well. Thus, approaches arose to deal with efficiency and usability aspects. Furthermore, some proof-of-concept prototypes were implemented.

The last development stage of technical workflow evolution research has been made when considering workflow schema and instance changes in their interplay and not in an isolated manner [9].

Currently, more and more approaches start to integrate semantic knowledge into workflow management systems. This can be used as a basis for developing intelligent adaptive systems which are able to learn from previous changes and therefore automatically adapt their workflows when the environment changes (cf. Future Directions).

Foundations

As already mentioned, workflow evolution comprises two basic challenges – *change discovery* and *change realization*. Change discovery refers to the detection of necessary business process optimizations which should be brought into the running workflows. For change discovery different approaches have emerged recently which are, for example, based on process mining or case-based reasoning techniques. Since these trends are currently "in the flow" the reader is referred to the "Future Directions" section for more details.

Contrary, the topic of workflow change (i.e., how to bring the discovered changes into the running system) has been investigated in great detail. Note that the following discussion of scientific challenges in the context of workflow evolution refers to dynamic workflow changes (i.e., changing a workflow instance or a workflow schema during runtime) and not to pre-planned workflow changes. Fig. 1a shows an example of an order workflow based on which the problem spectrum is motivated in the following.

At workflow type level, changes are handled by modifying the affected *workflow schema S* based on which a collection of workflow instances *I*1,..., *In* is running. Intuitively, it is necessary that the modification of a *correct* workflow schema *S* again results in a *correct* workflow schema S' (a workflow schema is called correct if it satisfies correctness constraints set out by the underlying workflow meta model, i.e., the formalism used to describe the business processes, e.g., Petri Nets). This so called *static* schema correctness can be preserved, for example, by the applied change operations.

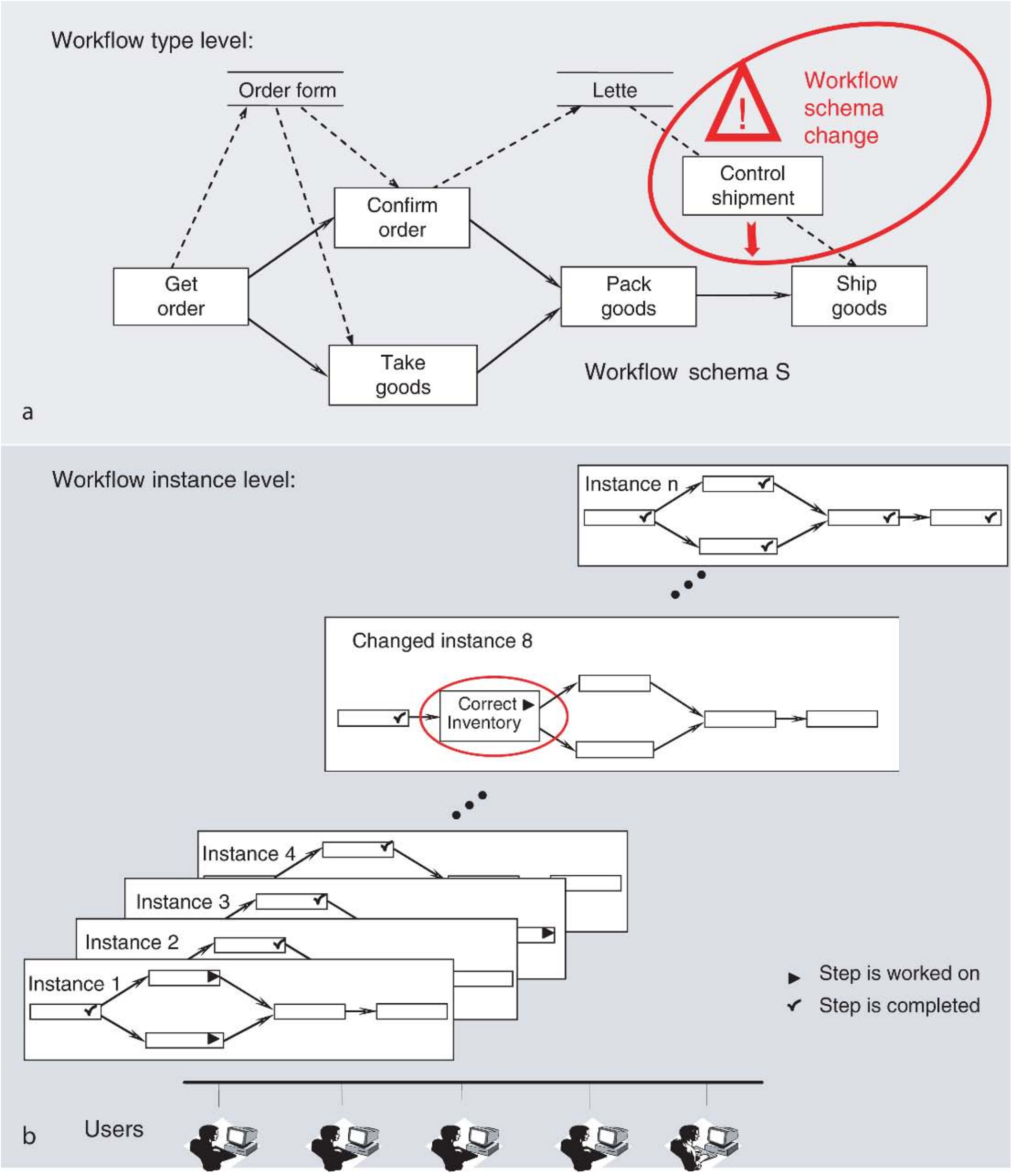
After changing workflow schema *S* it is also necessary to deal with process instances I1,..., *In* running on S (cf. Fig. 1b). Different strategies for this so-called *dynamic process schema evolution* have been proposed in literature. One possibility is to *cancel* all running instances and to restart them according to the new workflow schema. However, this strategy may cause an immense loss of work and would usually not be accepted by users. Therefore, at minimum, it is claimed that workflow instances I1,..., In started according to workflow schema *S* can be finished on *S* without being interrupted. This strategy can be implemented by providing adequate versioning concepts and may be sufficient for workflows of short duration. However, doing so raises severe problems in conjunction with long-running workflows as often found in, for example, clinical or engineering environments. Due to the resulting mix of workflow instances running on old and new schema versions a chaos within the production or the offered services may occur. Furthermore, often, it is not acceptable to run instances on the old schema version if laws or business rules (e.g., clinical guidelines) are violated.

For these reasons, it is quite important to be able to apply workflow schema changes to running instances as well. This scenario is called the *propagation* of workflow schema changes to running workflow instances or, in other words, the *migration* of the ("old") running instances to the changed workflow schema. Note that the number of running instances *I*1,..., *In* may be very large. In environments like hospitals or telecommunication companies, for example, $n > 10{,}000$ may easily hold.

To better understand the challenge of propagating a workflow schema change to running workflow instances, one can compare a running workflow instance with an application program. This program is currently executed and consists of several procedures. These steps can be successively executed (sequential execution), can be situated within if-then-else conditions (alternative branches), or can be even executed in parallel threads. Workflow schema evolution can then be compared to manipulating the running program by inserting one or more procedures, deleting procedures, or changing the order of procedures in the midst of program execution. In particular, the changes have to be carried out by maintaining a correct program execution (e.g., correct parameter provision) for all possible execution alternatives.

Consequently, it is a non-trivial but crucial task for workflow management systems to enable workflow schema evolution. The following main requirements have been identified:

1. *Completeness*: Workflow designers must not be restricted, neither by the used workflow meta model nor by the offered change operations.
2. *Correctness*: The ultimate ambition of any adaptive/flexible workflow management system must be the correctness of workflow changes; i.e., introducing changes to the runtime system without causing inconsistencies or errors (like deadlocks or improperly invoked activity programs). Therefore, adequate *correctness criteria* are needed. These criteria must not be too restrictive, i.e., no workflow instance should be needlessly excluded from applying a dynamic change.
3. *Efficient Correctness Checks*: Assume that an appropriate correctness criterion for workflow changes has been found. Then the challenge is to check this criterion *efficiently*. This is especially important for



Workflow Evolution. Figure 1. Example order workflow.

large-scale environments with hundreds (up to thousands) of running workflow instances.

4. *Usability*: The migration of workflow instances to a changed workflow schema must not require expensive user interactions. First of all, such interactions may lead to delays within the instance executions. Secondly, users (e.g., designers or administrators) must not be burdened with the job to adapt workflow instances to changed workflow schemes. Complex process structures and extensive state adaptations might overstrain them quickly. Therefore, it is crucial to offer methods for the automatic adaptation of workflow instances.

However, supporting workflow schema changes is not sufficient to enable workflow evolution. In fact, it must be possible to modify single workflow instances as well. Such workflow instance changes are often carried out in an ad-hoc manner in order to deal with an exceptional situation, e.g., an unforeseen correction of the inventory information in an order workflow as depicted for instance 8 in Fig. 1b.

In the literature, workflow schema and instance changes have been an important research topic for several years. So far, there are only a few adaptive systems supporting both kinds of changes in one system [4,14], however, only in a separated manner (i.e., already individually modified workflow instances are excluded from migration to the changed workflow schema). As discussed, this approach is not sufficient in many cases, particularly in connection with long-running workflows. Assume, for example, a medical treatment workflow which is normally executed in a standardized way for every patient. Assume further that due to an unforeseen reaction, an additional drug is given to a certain patient. However, this deviation from the standard procedure must not imply that the affected workflow instance (and therefore the patient) is excluded from further workflow optimizations. Therefore, it must be possible to propagate workflow schema changes at the type level to individually modified instances as well.

This *interplay* between *concurrently* applied workflow schema and instance changes raises many challenges, for example, the question to which degree the changes *overlap*. Overlapping means that workflow schema and instance change (partly) have the same effects on the underlying workflow schema. Such an overlap occurs, for example, if a workflow instance has (partly) anticipated a future workflow schema change in conjunction with a flawed process design. In this situation conflicts may arise between the overlapping workflow schema and instance changes (e.g., if the same activities are deleted).

Different approaches have been developed to deal with concurrent workflow schema and instance changes (e.g., based on workflow schema and change comparisons in order to determine overlapping degrees between changes) [9]. Thus, the current state-of-the art comprises sufficient solutions for the technical realization of workflow changes. However, there are advanced challenges regarding the semantic realization of workflow changes and the acquisition of the workflow changes. These challenges are described within the "Future Directions" section.

Key Applications

Production workflow (process) management systems

Clinical workflows
Office automation
Development workflows (e.g., in the automotive domain)
Enterprise application integration (?)

Future Directions

Currently, there are comprehensive solutions available regarding the technical support of workflow changes. However, this is not sufficient for a complete workflow evolution solution. First of all, in addition to the technical requirements as discussed in the "Foundations" section, also semantic aspects must be taken into account. In order to make this technology broadly applicable, it is not sufficient to check only whether the planned workflow changes violate any structural or state-related correctness properties. In addition, it must also be possible to verify if semantic constraints (e.g., business rules, four eyes principle) imposed on the affected workflow are violated [5].

In some cases the necessity to change a workflow schema has reasons like changes in legislation, in global business rules, or in the structural organization of the enterprise etc. In many cases, however, workflow schema changes are motivated by the observation that the existing workflow does not meet some real-world requirements. For example, the order of steps is not optimal, some steps are missing or not necessary, or important cases are not reflected in the workflow schema. It should not be left to users to detect necessary changes. If users find "work-arounds" to locally solve their problem, this information may never reach the person in charge. More promising is to let the system analyze execution logs in order to discover repetitive deviations (e.g., using process mining), to compute an alternative workflow schema covering these cases, and to support the process designer to (semi-)automatically perform the necessary workflow schema evolution [10].

Cross-references

- ▶ Business Intelligence
- ▶ Business Process Execution Language
- ▶ Business Process Management
- ▶ Business Process Reengineering
- ▶ Composed Services and WS-BPEL
- ▶ Petri Nets
- ▶ Process Mining
- ▶ Process Optimization
- ▶ Workflow Constructs
- ▶ Workflow Management and Workflow Management System
- ▶ Workflow Management Coalition
- ▶ Workflow Model
- ▶ Workflow Patterns
- ▶ Workflow Schema
- ▶ Workflow Transactions

Recommended Reading

1. Adams M., ter Hofstede A.H.M., Edmond D., and van der Aalst W.M.P. A service-oriented implementation of dynamic flexibility in workflows. In Proc. Int. Conf. on Cooperative Inf. Syst., 2006.
2. Casati F., Ceri S., Pernici B., and Pozzi G. Workflow evolution. Data Knowl. Eng., 24(3):211–238, 1998.
3. Heimann P., Joeris G., Krapp C., and Westfechtel B. DYNAMITE: dynamic task nets for software process management. In Proc. 18th Int. Conf. on Software Eng., 1996, pp. 331–341.
4. Kochut K., Arnold J., Sheth A., Miller J., Kraemer E., Arpinar B., and Cardoso J. IntelliGEN: a distributed workflow system for discovering protein-protein interactions. Distr. Parallel Databases, 13(1):43–72, 2003.
5. Ly L.T., Rinderle S., and Dadam P. Semantic correctness in adaptive process management systems. In Proc. Int. Conf. Business Process Management, 2006, pp. 193–208.
6. Müller R., Greiner U., and Rahm E. AgentWork: a workflow system supporting rule-based workflow adaptation. Data Knowl. Eng., 51(2):223–256, 2004.
7. Reichert M. and Dadam P. ADEPTflex – supporting dynamic changes of workflows with-out losing control. J. Intell. Inform. Syst., 10(2):93–129, 1998.
8. Rinderle S., Reichert M., and Dadam P. Correctness criteria for dynamic changes in workflow systems – a survey. Data Knowl. Eng., 50(1):9–34, 2004.
9. Rinderle S., Reichert M., and Dadam P. Disjoint and overlapping process changes: challenges, solutions, applications. In Proc. Int. Conf. on Cooperative Inf. Syst., 2004, pp. 101–120.
10. Rinderle S., Weber B., Reichert M., and Wild W. Integrating process learning and process evolution – a semantics based approach. In Int. Conf. Business Process Management, 2005, pp. 252–267.
11. Sadiq S., Marjanovic O., and Orlowska M. Managing change and time in dynamic workflow processes. Int. J. Cooper. Inform. Syst., 9(1, 2):93–116, 2000.
12. Sadiq S., Sadiq W., and Orlowska M. Pockets of flexibility in workflow specifications. In Proc. 20th Int. Conf. on Conceptual Modeling, 2001, pp. 513–526.
13. van der Aalst W.M.P. Exterminating the dynamic change bug: a concrete approach to support workflow change. Inform. Syst. Front., 3(3):297–317, 2001.
14. Weske M. Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In Proc. Hawaii Int. Conf. System Sciences, 2001.
15. Weber B., Rinderle S., and Reichert M. Change patterns and change support features in process-aware information systems. In Proc. Int. Conf. Advanced Information Systems Engineering, 2007, pp. 574–588.

Workflow Join

Nathaniel Palmer
Workflow Management Coalition, Hingham,
MA, USA

Synonyms

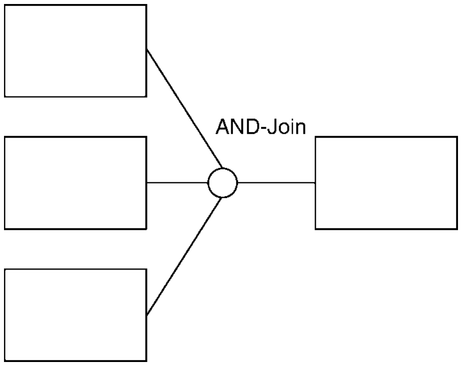
AND-join; Rendezvous; Synchronization join

Definition

A point in the workflow where two or more parallel executing activities converge into a single common thread of control.

Key Points

The execution of parallel activities commences with an AND-Split and concludes with an AND-Join. For example, in a credit application process there may be a split in the workflow at which point multiple activities are completed separately (in parallel, if not simultaneously.) Each parallel executing thread is held until the set of all thread transitions to the next activity is completed at which point the threads converge and the next activity is initiated.



Cross-references

- ▶ AND-Split
- ▶ OR-Join
- ▶ Process Life Cycle
- ▶ Workflow Management and Workflow Management System

Workflow Lifecycle

- ▶ Process Life Cycle

Workflow Loop

- ▶ Loop

Workflow Management

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

Workflow; Business process management; Case management

Definition

The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

Key Points

The automation of a business process is defined within a Process Definition, which identifies the various process activities, procedural rules and associated control data used to manage the workflow during process enactment.

Many individual process instances may be operational during process enactment, each associated with a specific set of data relevant to that individual process instance (or workflow "Case"). A loose distinction is sometimes drawn between production workflow, in which most of the procedural rules are defined in advance, and ad-hoc workflow, in which the procedural rules may be modified or created during the operation of the process.

Cross-references

- ▶ Business Process Management
- ▶ Event-Driven Business Process Management
- ▶ Workflow Management and Workflow Management System
- ▶ Workflow Model
- ▶ XML Process Definition Language

Workflow Management and Workflow Management System

Johann Eder
University of Vienna, Vienna, Austria

Synonyms

Business process management

Definition

Business Process: A set of one or more linked procedures or activities that collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships [10].

Workflow: The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules [10].

Workflow Management System (WFMS): A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engine, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications [10].

Historical Background

In the 1980s, office automation was a major research focus with several topics, the support for the including movement of documents through offices. Some proposals were made to model document or form flows and to support the "paperless" office.

Another driver for the development of workflows came from extended transaction models. The problem was to solve integrity problems in heterogenous information systems where, within one business process, several databases are updated. Integrity was compromised if some of the updates failed or were not carried out. Extended transaction models aimed to provide assurance that all steps were performed properly.

A third development stream came from computer supported cooperative work, where the need arose to better support repeating processes.

In the 1980s revolutionary ideas for improving business by business process reengineering also emerged. Enterprises were urged to organize themselves around their core business processes and not according to functional considerations as before. This movement in business administration called for software to support process driven organizations.

Since the 1990s, hundreds of workflow systems have been developed, both academic prototypes and commercial products with quite different functionalities.

Foundations

Workflows automate and/or support the execution of business processes. The core idea of workflow management is to separate the process logic from the functional application logic. This means that applications provide functionality, and the workflow system controls, when to involve which functionality. So the workflow is responsible for *who* (which actor) does *what* (which activity), with *which data* and resources, *when* (control logic) under *which constraints*. And it is the job of the application system or workflow participant to know *how* an activity is performed. So the workflow system is responsible for the logistics of a process but not for the execution of the individual steps. This separation of concerns, to separate the doing of things from the organization of the process is supposed to make software more flexible and to integrate heterogeneous applications.

Aspects of Workflows

A workflow consists of a set of activities and the dependencies between them, the so-called business logic. These dependencies can be expressed in different forms. The most frequent way is to express the dependencies by control flow definitions, often supported by graphical notations. Other ways are data flow or rules. However, the control flow is only one aspect of workflow management, although it has received most attention.

The following aspects of workflows can be distinguished:

Activities: Activities are units of work to be performed, either by a specific program or by an user, typically interacting with some application programs. Activities can be elementary, i.e from the point of the workflow systems they are the smallest units of work. Elementary activities are frequently called tasks. Activities can be composed to complex activities.

Process: The process defines the control flow, i.e., the admissible relative orders of activity executions. Typical elements of control flow are sequence, and-parallelism (concurrent execution of activities), or-parallelism (selection of different activities, inclusive or exclusive). Workflow models provide the means for defining the process. Many process models rely on (variants of) Petri-nets or (simpler) variants of Petri-nets, in particular, so called workflow nets. Workflow nets are graphs, where the nodes are activities and the directed edges represent a precedence relation between activities. Additionally, there are control nodes: and-split for invoking several successor activities concurrently, and-join for synchronizing parallel threads, xor-splits for selecting one of several successor activities and xor-join for joining alternate paths again.

Actors: Actors perform the tasks defined in the workflow specification. Actors can be humans or systems. Systems perform so-called automatic tasks, i.e., execute some application programs. Human actors can be identified directly (by a user name), or indirectly by reference to a position in the organization, by roles, representing a set of users competent for fulfilling a tasks, by organizational unit, by specification of necessary skills and competencies, or by reference to some data item containing the actor. For indirect representation of the individual actor, the workflow system has to resolve the role at runtime and assign a user to a particular task. For the resolution of actors specified by organizational units, the workflow system maintains a model of the organization.

Data: Data are needed for the processing of a workflow. Three kinds of data comprise workflows. *Workflow control data* are managed by a WFMS and describe workflow execution (e.g., control and data flow among activities), relevant internally for a WFMS and without a long-term impact beyond the scope of the current workflow (e.g., a termination state of an activity). *Application data* are managed by the applications supporting the process instance and generally are never seen by the WFMS. *Workflow relevant data* are used by the WFMS to determine the state transitions of a workflow (e.g., transition conditions) and may be accessed both by the WFMS and the applications. Further, a workflow system manages *organizational data* which it uses at runtime to resolve actors, and *audit data* kept in the workflow log documenting the execution details of workflow instances.

Applications: Applications are pieces of software that can be invoked within a workflow to fulfill a certain activity. A WFMS has to administer applications and the details of their invocation.

Other aspects: Other aspects of workflows include constraints that have to be satisfied (e.g., pre- and postcondition of activities, quality-of-service constraints), in particular temporal constraints specifying deadlines and bounds for the duration between activities. Business rules may restrict the process.

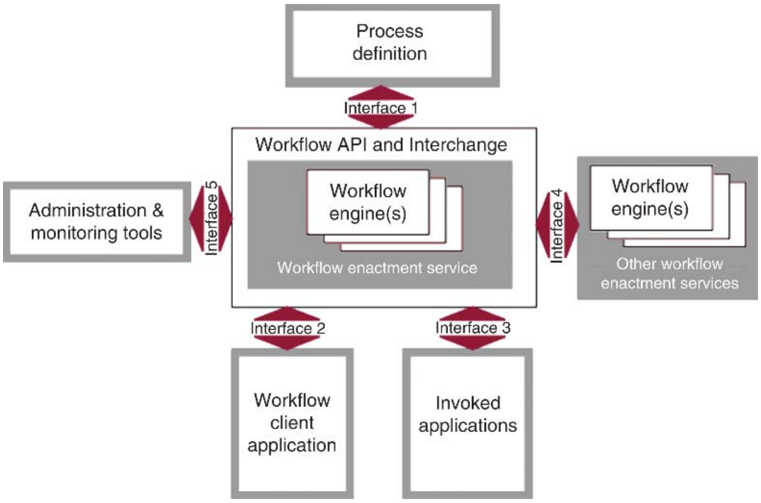
Workflow Management Systems

Workflow Management Systems are generic software systems that are driven by workflow models and control the execution of workflows.

The Workflow Management Coalition (WfMC) developed a reference architecture for WFMS that has been widely adopted. Figure 1 shows the model with its components and interfaces.

The Workflow Enactment Service, consisting of one or several workflow engines, is the core of a workflow management system. It accepts a process description, interprets it, creates instances (workflows), and controls the execution of the process instances. This means that the activities of the workflows are invoked according to the logic expressed in the process model, activities are dispatched to actors, and, if necessary, external services and external applications are invoked or process instances are forwarded to other workflow enactment services.

Interface 1 accepts the process definition that contains all information necessary for the execution of process instances. Process definition tools are employed to describe processes in a proper format. Some process definition languages have been proposed as standards (e.g., XPDL or BPEL) to have uniform interfaces between process definition tools and workflow enactment services. Nevertheless, many workflow management systems feature proprietary languages, partly because of the different types of workflows they intend to support.


Process definition
Interface 1
Workflow API and Interchange
Workflow engine(s)
Workflow enactment service
Administration & monitoring tools
Interface 5
Interface 4
Workflow engine(s)
Other workflow enactment services
Interface 2
Interface 3
Workflow client application
Invoked applications

Workflow Management and Workflow Management System. Figure 1. Workflow Management Systems: WfMC reference architecture [2].

Interface 2 links the workflow enactment service with workflow client applications, the piece of software the workflow participants (or actors) work with. The main feature of this component is to maintain the worklist, a collection of all activities assigned to a particular user. Users can typically accept an activity, delegate it, refuse it, work on it, or complete it.

Interface 3 describes how other applications are invoked. This interface is frequently used to invoke legacy applications. Since applications (in particular interactive programs) can be invoked from the workflow client application as well, the interfaces 2 and 3 are typically combined permitting the invocation of applications both directly from the workflow enactment service and on user request from the workflow client tool.

Interface 4 addresses workflow interoperability. It defines the interaction between workflow enactment services, the invocation of subworkflows, the synchronization between cooperating workflows, and the forwarding of workflows to other workflow applications.

Interface 5 specifies the interaction with administration and monitoring tools. Administration tools provide the general administration of the workflow system: the registration of user, of application programs, the security management, in particular the administration of rights. The monitoring tools allow for the inspection of the progress of workflow instances and typically provide sophisticated statistics about the execution of workflows to support process managers and to provide data for process improvement.

Key Applications

Workflow systems are mainly employed to support the management and execution of business processes. The typical architecture is that a workflow management system is used for managing the business processes and for integrating the different application areas. Such an architecture can be found in various enterprises and organizations. A few example applications include insurance claim processing, loan processing, product development, bug reporting, order processing, media production, etc.

However, there are additional types of workflow applications:

- Groupware like systems with high flexibility to support workflow that cannot be fully modeled beforehand but develop as the process is executed. These systems primarily support the cooperation between individuals and care for highly dynamic processes.
- There are workflow systems as top layers on large enterprise software systems, adding processes to the rich integrated functionality of these systems.
- Workflow systems are used to integrate heterogeneous application systems and provide an integration platform for these systems. Workflow systems are a layer in the architecture of software systems caring for the process aspects just as data management systems are responsible for storing and retrieving data (process brokers).

Workflows are frequently classified into three different groups according to the criteria of how much they can be automated, how frequent they are executed, and how much they can be specified:

- *Ad-hoc workflows* are highly dynamic workflows that are not fully specified at build time and consist mainly of manual activities. Examples are document routing, or review processes.
- *Administrative workflows* are well structured and more repetitive. They are the typical office paper shuffling type of processes. Examples are budgeting, purchase order processing, etc.
- *Production workflows* are highly repetitive, well structured and fully automated. They represent the core business processes of an organization. Examples are insurance claim handling, bug reporting, order processing, etc.

The employment of workflow technology is supposed to bring the following benefits:

- Efficiency: The automation of process control, the increased possibilities for concurrent execution of activities of the same workflow instance, and the automation of the forwarding of work to the next actor increase the efficiency of business processes as measured in throughput and in turn-around time.
- Process modeling: The business processes are explicitly specified and do no longer solely exist in the heads of the participants or in manuals. This is a necessary prerequisite to reflect on processes and to improve processes.
- Automated process control: the centralized automatic control of the process execution guarantees that business rules are respected. Deviations from the specified process, which are of course possible, are documented and can be used for auditing and

process improvement. Users are informed of work to do, and the data used for performing activities are automatically delivered to the respective workers.

- Integration: Workflow systems allow to bridge heterogeneous applications. Functionalities in application systems are called external applications. The workflow describes and the WFMS enforces the proper sequences of invocation of functionalities in different application systems and transports data between different systems. In general, the integrity of the whole system consisting of heterogeneous applications is achieved.
- Transparency: Transparency is increased through explicit process specification and documentation. All steps and decisions in the execution of a process are centrally documented (workflow log or workflow history). Therefore, it is easy to report the actual state of a process instance.
- Monitoring: The progress of workflows can be easily monitored. For example, delays or exceptional situations are immediately recognized allowing process managers to react timely to assure that deadlines are met.
- Documentation: The execution of workflows is automatically documented. Each process can be traced, e.g., for the purpose of revision. The process documentation is also a valuable source for statistics about the processes to identify weaknesses of the process definition and for process improvements.
- Quality assurance: Quality assurance procedures like the ISO 9000 Suite or TQM (Total Quality Management) require in particular that processes are specified, that processes are executed according to their specification, and that all processes are documented. All that is supported by workflow systems.
- Process oriented management: Workflow systems support the management of process driven organization.

Workflow systems are a flourishing market. Applications can be found in a large number of application domains: e.g., insurances, banks, government agencies, hospitals, etc. Scientific workflows are successfully applied in life science, physics, and chemistry, etc. There are many applications with embedded workflow technology, e.g., document management systems, content management systems, or ERP-systems.

Future Directions

An important development of workflow systems is their combination with web service technology. Web service composition relies on the same principles as workflow, the main difference is that the activities are web services.

Another important topic is the support of interorganizational business processes. The benefits of workflow technology should be exploited also for business processes that span several organizations (enterprizes). While this is already achieved for some types of virtual enterprizes, where the partners (suppliers, costumers, consultants, service providers, etc.) involved and the interfaces are fairly stable, it is still a challenge to support processes with changing partners or multitudes of partners. Here a lot of research is still needed, in particular, integration with semantic web services.

Cross-references

▶ Database Techniques to Improve Scientific Simulations
▶ Extended Transaction Models
▶ Workflow Management Coalition
▶ Workflow Model
▶ Workflow Transactions

Recommended Reading

1. van der Aalst W. et al. Workflow Management: models, methods, and systems. MIT, Cambridge, MA, 2002.
2. Dumas M., van der Aalst W.M., and ter Hofstede A.H. Process-aware information systems: bridging people and software through process technology. Wiley, 2005.
3. Eder J., Groiss H., and Liebhart W. The Workflow Management System Panta Rhei. In Workflow Management Systems and Interoperability. Kalinichenko, A. Dogac, M.T. Özsu, A. Sheth (eds). 1998. pp. 129–144.
4. Ellis C.A. and Nutt G.J. Office Information Systems and Computer Science. ACM Comput. Surv., 12(1): 27–60, 1980.
5. Ellis C.A. and Nutt G.J. Modeling and Enactment of Workflow Systems. In Proc. 14th Int. Conf. on Application and Theory of Petri Nets., 1993, pp. 1–16.
6. Georgakopoulos D., Hornick M., and Sheth A. An overview of workflow management: From process modeling to workflow automation infrastructure. Distrib. Parallel Databases, 3(2):119–153, 1995.
7. Jablonski S. and Bussler C. Workflow management: modeling concepts, architecture and implementation. Int. Thomson Computer, 1996.
8. Leymann F. and Roller D. Production workflow. Prentice-Hall, Englewood, Cliffs, NJ, 2000.
9. The Workflow Reference Model. Workflow Management Coalition, 1995.
10. Workflow management coalition terminology and glossary [R]. Workflow Management Coalition, 1999.

Workflow Management Coalition

NATHANIEL PALMER
Workflow Management Coalition, Hingham, MA, USA

Synonyms

XPDL

Definition

The Workflow Management Coalition, founded in August 1993, is a non-profit, international organization of workflow vendors, users, analysts and university/research groups. The Coalition's mission is to promote and develop the use of workflow through the establishment of standards for software terminology, interoperability and connectivity among BPM and workflow products. Comprising more than 250 members worldwide, the Coalition is the primary standards body for this software market.

Key Points

The WfMC creates and contributes to process related standards, educates the market on related issues, and is the only standards organization that concentrates purely on process. The WfMC created Wf-XML and XPDL, the leading process definition used today in over 70 solutions to store and exchange process models. XPDL is not an executable programming language but a process design format for storing the visual diagram and process syntax of business process models, as well as extended product attributes.

The Coalition has developed a framework for the establishment of workflow standards. This framework includes five categories of interoperability and communication standards that will allow multiple workflow products to coexist and interoperate within a user's environment. These interface points are structured around the workflow reference model which provides the framework for the Coalition's standards program. The reference model identifies the common characteristics of workflow systems and defines five discrete functional interfaces through which a workflow management system interacts with its environment – users, computer tools and applications, other software services, etc. Working groups meet individually, and also under the umbrella of the Technical Committee, which is responsible for overall technical direction and coordination.

Cross-references

- ► Workflow Management and Workflow Management System
- ► Workflow Model
- ► XML Process Definition Language

Workflow Meta-Model

► Workflow Schema

Workflow Mining

► Process Mining

Workflow Model

NATHANIEL PALMER
Workflow Management Coalition, Hingham, MA, USA

Synonyms

Business process model; Process definition

Definition

A set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships.

Key Points

- A business process is typically associated with operational objectives and business relationships, for example an Insurance Claims Process, or Engineering
- Development Process. A process may be wholly contained within a single organizational unit or may span several different organizations, such as in a customer-supplier relationship.
- A business process has defined conditions triggering its initiation in each new instance (e.g., the arrival of a claim) and defined outputs at its completion.
- A business process may involve formal or relatively informal interactions between participants; its duration may also vary widely.

- A business process may consist of automated activities, capable of workflow management, and/or manual activities, which lie outside the scope of workflow management.

Cross-references

▶ Workflow Management and Workflow Management System

▶ Workflow Modeling

Workflow Model Analysis

W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven,
The Netherlands

Definition

The correctness, effectiveness, and efficiency of the business processes supported by a workflow management systems are vital to the organization. A workflow process definition which contains errors may lead to angry customers, back-log, damage claims, and loss of goodwill. Flaws in the design of a workflow definition may also lead to high throughput times, low service levels, and a need for excess capacity. This is why it is important to *analyze* a workflow process definition before it is put into production.

In order to analyse a workflow definition, it is necessary to translate the definition into a model suitable for analysis, e.g., a simulation model or a model that allows for verification techniques (e.g., a Petri net).

Key Points

Process-aware information systems are driven by process models. A typical example of a process-aware information system is a system generated using workflow management software. In such cases there is an explicit process model that can be used as a starting point for analysis [3,4].

There are three types of workflow model analysis:

1. *Validation*, i.e., testing whether the workflow behaves as expected.
2. *Verification*, i.e., establishing the correctness of a workflow.
3. *Performance analysis*, i.e., evaluating the ability to meet requirements with respect to throughput times, service levels, and resource utilization.

Validation can be done by interactive simulation: a number of fictitious cases are fed to the system to see whether they are handled well. For verification and performance analysis more advanced analysis techniques are needed. Fortunately, many powerful analysis techniques have been developed for formal methods such as Petri nets [2,3]. Linear algebraic techniques can be used to verify many properties, e.g., place invariants, transition invariants, and (non-)reachability. Coverability graph analysis, model checking, and reduction techniques can be used to analyze the dynamic behavior of the workflow process. Simulation, queueing networks, and Markov-chain analysis can be used for performance evaluation (cf. [1,2]).

Note that all three types of analysis assume that there is some initial workflow model. This implies that analysis may require some modeling or some translation from one language to another. In some cases, *process mining* can be used to discover the underlying workflow model.

Cross-references

▶ Business Process Management

▶ Petri Nets

▶ Process Mining

▶ Workflow Management

Recommended Reading

1. Marsan M.A., Balbo G., and Conte G. et al. Modelling with generalized stochastic Petri nets. Wiley Series in Parallel Computing. Wiley, New York, NY, USA, 1995.
2. Reisig W. and Rozenberg G. (eds.) Lectures on Petri Nets I: Basic Models. LNCS. 1491. Springer, Berlin Heidelberg New York, 1998.
3. van der Aalst W.M.P. The application of Petri nets to workflow management. J. Circ. Syst. Comput., 8(1):21–66, 1998.
4. van der Aalst W.M.P. and van Hee K.M. Workflow Management: Models, Methods, and Systems. MIT, Cambridge, MA, 2004.

Workflow Modeling

Marlon Dumas
University of Tartu, Tartu, Estonia

Synonyms

Business process modeling

Definition

A workflow model is a representation of the way an organization operates to achieve a goal, such as delivering a product or a service. Workflow models may be given as input to a workflow management system to automatically coordinate the tasks composing the workflow model. However, workflow modeling may be conducted purely for documentation purposes or to analyze and improve the operations of an organization, without this improvement effort implying automation by means of a workflow system.

A typical workflow model is a graph consisting of at least two types of nodes: task nodes and control nodes. Task nodes describe units of work that may be performed by humans or software applications, or a combination thereof. Control nodes capture the flow of execution between tasks, therefore establishing which tasks should be enabled or performed after completion of a given task. Workflow models, especially when they are intended for automation, may also include object nodes denoting inputs and outputs of tasks. Object nodes may correspond to data required or produced by a task. But in some notations, they may correspond to physical documents or other artifacts that need to be made available or that are produced or modified by a task. Additionally, workflow models may include elements for capturing resources that are involved in the performance of tasks. For example, in two notations for workflow modeling, the Unified Modeling Language (UML) Activity Diagrams, and the Business Process Modeling Notation (BPMN), resource types are captured as *swimlanes*, with each task belonging to one swimlane (or multiple in the case of UML). In other notations, such as Event-driven Process Chains (EPC), this is represented by attaching resource types to each task.

Historical Background

The idea of documenting business processes to improve customer satisfaction and business operations dates back to at least the 1960s when it was raised, among others, by Levitt [8]. Levitt contended that organizations should avoid focusing exclusively on goods manufacturing, and should view their "entire business process as consisting of a tightly integrated effort to discover, create, arouse, and satisfy customer needs." However, it is not until the 1970s that the idea of using computer systems to coordinate business operations based on process models emerged. Early scientific work on business process modeling was undertaken in the context of *office information systems* by Zisman [13] and Ellis [2] among others. During the 1970s and 1980s there was optimism in the IT community regarding the applicability of process-oriented office information systems. Unfortunately, few deployments of this concept succeeded, partly due to the lack of maturity of the technology, but also due to the fact that the structure and operations of organizations were centered around the fulfillment of individual functions rather than end-to-end processes. Following these early negative experiences, the idea of process modeling lost ground during the next decade.

Towards the early 1990s, however, there was a renewed interest in business process modeling. Instrumental in this revival was the popularity gained in the management community by the concept of *Business Process Reengineering* (BPR) advocated by Hammer [3] and Davenport [1] among others. BPR contended that overspecialized tasks carried across multiple organizational units should be unified into coherent and globally visible processes. This management trend fed another wave of business process technology known as workflow management. Workflow management, in its original form, focused on capturing business processes composed of tasks performed by human actors and requiring data, documents and forms to be transferred between them. Workflow management was successfully applied to the automation of routine and high-volume administrative processes such as order-to-cash, procure-to-pay and insurance claim handling. However it was less successful in application scenarios requiring the integration of heterogeneous information systems, and those involving high levels of evolution and change. Also, standardization efforts in the field of workflow modeling and management, led by the Workflow Management Coalition (WfMC), failed to gain wide adoption.

The BPR trend also led to the emergence of business process modeling tools that support the analysis and simulation of business processes, without targeting their automation. The ARIS platform, based on the EPC notation, is an example of this family of tools. Other competing tools supported alternative notations such as IDEF3 and several variants of flowcharts.

By the late 1990s, the management community had moved from the concept of BPR to that of *Business Process Improvement* (BPI), which advocates an incremental and continuous approach to adopting process-orientation. This change of trend combined with the limited success of workflow management and its failure to reach standardization, led to the term *workflow* acquiring a negative connotation. Nonetheless, the key concepts underpinning workflow management reemerged at the wake of the twenty-first century under the umbrella of *Business Process Management* (BPM). BPM adopts a more holistic view than workflow management, covering not only the automated coordination of processes, but also their monitoring and continuous analysis and improvement. Also, BPM does not only deal with the coordination of human tasks, but also the integration of heterogeneous information and software systems.

With BPM came the realization that business processes should be analyzed and designed both from a business viewpoint and from a technical viewpoint, and that methods are required to bridge these viewpoints. Also, the BPM trend led to a convergence of modeling languages. BPMN has emerged as a standard for process modeling at the analysis level, while at the implementation level, the place is taken by WS-BPEL. Other notations include UML Activity Diagrams and EPCs at the analysis level, and YAWL (Yet Another Workflow Language) [6] at the execution level.

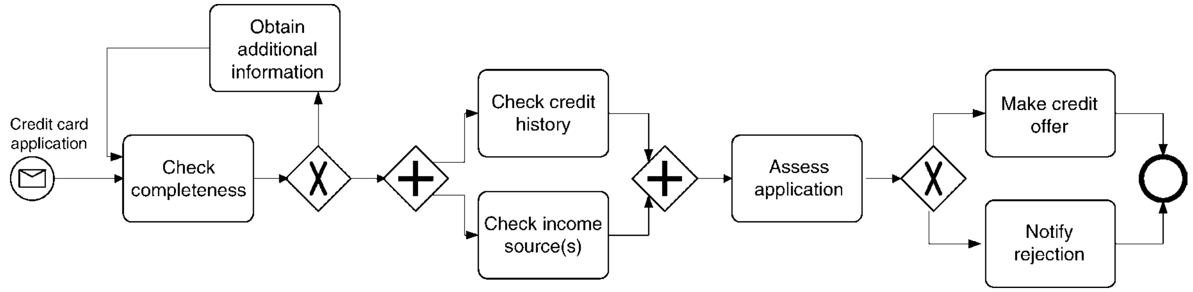
Foundations

Workflow modeling can be approached from a number of perspectives [4]. The *control-flow perspective* describes the ordering and causality relationships between tasks. The *data perspective* (or information perspective) describes the data that are taken as input and produced by the activities in the process. The *resource perspective* describes the structure of the organization and identifies resources, roles, and groups. The *task perspective* describes individual steps in the processes and thus connects the other three perspectives.

Many workflow modeling notations emphasize the control-flow perspective. This is the case for example of process modeling notation based on *flowcharts.* Flowcharts consist of actions (also called activities) and decision nodes connected by arcs that denote sequential execution. In their basic form, flowcharts do not make a separation between actions performed by different actors or resources. However, an extension of flowcharts, known as cross-functional flowcharts, allow one to perform this partitioning. Flowcharts have inspired many other contemporary process modeling notations, including UML Activity Diagrams and BPMN.

A simplified business process model for credit approval in BPMN is given in Fig. 1. An execution of this process model is triggered by the receipt of a credit application, denoted by the leftmost element in the model. Following this, the application is checked for completeness. Two outcomes are possible: either the application is marked as complete or as incomplete. This choice is represented by the *decision gateway* depicted by the diamond labeled by an “X” sign. If the application is incomplete, additional information is sought from the applicant. Otherwise, two checks are performed in parallel: a credit history check and an income check. This is denoted by a *parallel split gateway* (the first diamond labeled by a “+” sign). The two parallel checks then converge into a *synchronization gateway* (the second diamond labeled by a “+”). The credit history check may be performed automatically while the income source check may require an officer to contact the applicant’s employer by phone. After these checks, the application is assessed and it is either accepted or rejected.

A second family of notations for business process modeling are those based on state machines. State machines provide a simple approach to modeling behavior. In their basic form, they consists of states connected by transitions. When used for business process



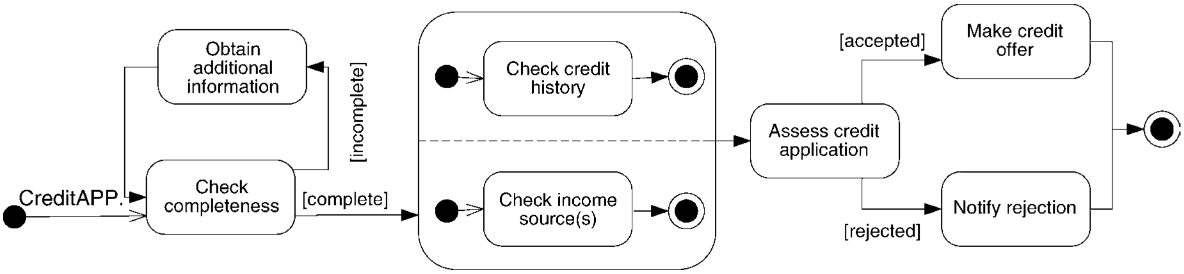
Workflow Modeling. Figure 1. Example of a business process model in BPMN.

modeling, it is common for the transitions in a state machine to be labeled by event-condition-action (ECA) rules. The semantics of a transition labeled by an action is the following: if the execution of the state machine is in the source state, an occurrence of the event (type) has been observed, and the condition holds, then the transition may be taken. If the transition is taken, the action associated with the transition is performed and the execution moves to the target state. A transition may contain any combination of these three elements: e.g., only an event, only a condition, only an action, an event and a condition but no action, etc. In some variants of state machines, it is possible to attach activities to any given state. The semantics is that when the state is entered, the activity in question may be started, and the state can only be exited once this activity has completed. Several commercial business process management systems support the execution of process models defined as state machines. Examples include the "Business State Machine" models supported by IBM Websphere and the "Workflow State Machine" models supported by Microsoft Windows Workflow Foundation.

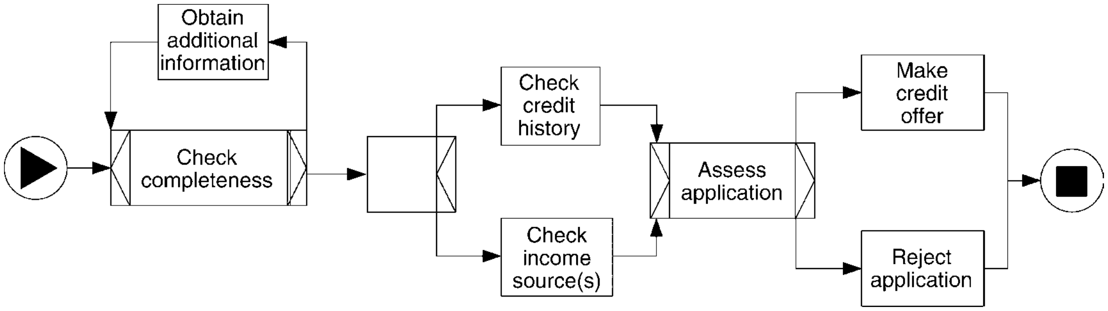
A disadvantage of using basic state machines for process modeling is their tendency to lead to state explosion. Basic state machines represent sequential behavior: the state machine can only be in one state at a time. This, when representing a business process model with parallel execution threads, one essentially needs to enumerate all possible permutations of the activities that may be executed concurrently. Another source of state explosion is exception handling. Statecharts have been proposed as a way to address this state explosion problem. Statecharts include a notion of concurrent components as well as transitions that may interrupt the execution of an entire component of the model. The potential use of statecharts for workflow modeling has been studied in the research literature, for example in the context of the Mentor research prototype [10], although it has not been adopted in commercial products.

Figure 2 shows a statechart process model intended to be equivalent to the BPMN "Credit Approval" process model of Fig. 1. It can be noted that decision points are represented by multiple transitions sharing the same source state and labeled with different conditions (conditions are written between square brackets). Parallel execution is represented by means of a composite state divided into two concurrent components. Each of these components contains one single state. In the general case, each concurrent component may contain an arbitrarily complex state machine.

Finally, a third family of notations for control-flow modeling of business processes are those based on *Petri nets*. In the research literature, workflow modeling notations based on Petri nets have been used mainly for studying expressiveness and verification problems (e.g., detecting deadlocks in workflow models). Also, certain Petri net-based notations are supported by business process modeling tools and workflow management systems. A notable example is YAWL. YAWL is an extension of Petri nets designed on the basis of the *Workflow Patterns*. A key design goal of YAWL is to support as many workflow patterns as possible in a direct manner, while retaining the theoretical foundation of Petri nets. Like Petri nets, YAWL is based on the concepts of place and transition (Transitions are called *tasks* in YAWL while places are called *conditions*.). YAWL also incorporates a concept of *decorator*, which is akin to the concept of gateway in BPMN. In addition, it supports the concepts of sub-process (called *composite task*) and cancelation region. The latter is used to capture the fact that the execution of a set of tasks may be interrupted by the execution of another task, and can be used for example for fault handling. Figure 3 shows a YAWL process model intended to be equivalent to the BPMN "Credit Approval" process



Workflow Modeling. Figure 2. Example of a business process model captured as a statechart.



Workflow Modeling. Figure 3. Example of a business process model captured in YAWL.

model of Fig. 1 and the statechart process model of Fig. 2. The symbols attached to the left and the right of each task are the decorators. The decorator on the left of task "Check Completeness" is an XOR-merge decorator, meaning that the task can be reached from either of its incoming flows. The decorator of the right of this same task is an XOR-split, which corresponds to the decision gateway in BPMN. The decorator just before "Check Credit History" is an AND-split (akin to a parallel split gateway) and the one attached to the left of "Assess Application" is an AND-join and it is used to denote a full synchronization.

The data perspective of process modeling can be captured using notations derived from dataflow diagrams. Dataflow diagrams are directed graphs composed of three types of nodes: *processes*, *entities* and *data stores*. Entities denote actors, whether internal or external to an organization. Processes denote units of work that transform data. Data stores denote units of data storage. A data store is typically used to store data objects of a designated type. The arcs in a dataflow diagram are called *data flows*. Data flows denote the transfer of data between entities, processes and data stores. They may be labeled with the type of data being transferred.

Data flow diagrams are often used very liberally, with almost no universally accepted syntactic constraints. However, best practice dictates that each process should have at least one incoming data flow and one outgoing data flow. Data flows may connect entities to processes and vice-versa, but can not be used to connect entities to data stores or to connect one entity to another directly. It is possible to connect two processes directly by means of a data flow, in which case it means that the output of the first process is the input of the second. It is also possible to assign ordinal numbers to processes in order to capture their execution order. Dataflow diagrams also support decomposition: Any process can be decomposed into an entire dataflow diagram consisting of several subprocesses.

Data flows are quite restricted insofar as they do not capture decision points and they are not suitable for capturing parallel execution or synchronization. Because of this, dataflow diagrams are not used to capture processes in details, instead, they are used as a high-level notation.

On the other hand, some elements from data flow diagrams have found their way into other process modeling notations such as EPCs, which incorporate constructs for capturing input and output data of functions. Similarly, Activity Diagrams have a notion of object nodes, used to represent inputs (outputs) consumed (produced) by activities. Also, more sophisticated variants of dataflow languages have been adopted in the field of *scientific workflow*, which is concerned by the automation of processes involving large datasets and complex computational steps in fields such as biology, astrophysics and environmental sciences.

When modeling business processes for execution (e.g., by a workflow system), it is important to capture not only the flow of data into and out of tasks, but also data transformation, extraction and aggregation steps. At this level of detail, it may become cumbersome to use simple data flows. Instead, executable process modeling languages tend to rely on *scoped variables*. Variables may be defined for example at the level of the process model. Tasks may take as input several input and output parameters, and data mappings are used to connect the data available in the process model variables, and the input and output parameters. Data mappings may be inbound (i.e., expressions that extract data from variables in the task's encompassing scope and assign values to the task's input parameter) or outbound (from the output parameters to the variables in the encompassing scope). This is the approach used for example in YAWL, Mentor and AdeptFlex [9]. It is also the approach adopted by many commercial workflow products and by WS-BPEL.

One area of research in the area of workflow modeling is the analysis of their expressiveness [6]. Several classes of workflow notations have been identified from this perspective: structured workflow models, synchronizing workflow models, and standard workflow models. Structured workflow models are those that can be composed out of four basic control-flow constructs, one for choosing one among multiple blocks of activities, another for executing multiple blocks of activities in parallel, a third one for executing multiple blocks of activities in sequence, and finally a fourth one for repeating a block of activities multiple times. An activity by itself can be a block, and more complex blocks can be formed by composing smaller blocks using one of the four constructs. It has been shown that structured workflow models are well-behaved, meaning in particular that their execution can not result in deadlocks [5]. On the other hand, this class of models has low expressive power. It is not difficult to construct examples of unstructured workflow models that can not be translated to equivalent structured ones under reasonable notions of equivalence. Synchronizing workflow models are acyclic workflow models composed of activities that are connected by transitions. These transitions correspond to control links in the WS-BPEL terminology. They impose precedence relations and they can be labeled with conditions. An activity can be executed if all its predecessors have been either completed or skipped. Like structured workflow models, this class of models are well-behaved. However, they are not very expressive. Finally, standard workflow models correspond to the subset of BPMN composed of tasks, decision gateways, merge gateways (where multiple alternative branches converge), parallel split gateways and synchronization gateways. This class of models is more expressive than the former two, but on the other hand, some of these models may contain deadlocks. Other classes of workflow models are those corresponding to Workflow Nets, which are more expressive than standard workflow models as they can capture different types of choices. YAWL models are even more expressive due to the presence of the cancelation region construct, and another construct (called the "OR-join") for synchronization of parallel branches, where some of these branches may never complete. A significant amount of literature in the area of workflow verification has dealt with studying properties of modeling languages incorporating a notion of OR-join [7].

Key Applications

Business process modeling can be applied both for documentation, business analysis, business process improvement and automation. In some cases, organizations undertake business process modeling projects for the sake of documenting key activities and using this documentation for employee induction and knowledge transfer. business process models are sometimes required for compliance reasons, for example as a result of legislation in the style of the Sarbannes-Oxley act in the United States which requires public companies to have internal control processes in place to detect and prevent frauds. A process model that captures how an organization currently works, is called an "as is" process model. Such "as is" models are useful in understanding bottlenecks and generating ideas for improvement using process analysis techniques, including simulation and activity-based costing. The integration of improvement ideas in a business process leads to a "to be" process model. Some of these improvement ideas may include automating (parts of) the business process. At this stage, business process models need to be made more detailed. Once they have been made executable, business process models may be deployed in a business process execution engine (also called a workflow engine).

Future Directions

Standardization efforts in the field of business process modeling have led to the current co-existence of two standard process modeling languages: BPMN at the business analysis level and BPEL at the implementation and execution level. One question that arises from the co-existence of these two standards is whether or not models in BPMN should be synchronized with models defined in BPEL and how. The assumption behind this question is that business analysts will define models in BPMN, while developers are likely to work at the level of BPEL. For facilitate the collaboration between these stakeholders, it would be desirable to relate models defined in these two languages. An alternative would be for the two languages to converge into a single one.

Another related area where open problems exist is that of scientific workflow modeling. Researchers have argued for some time that workflow-related problems in scientific fields such as genetics and ecological sciences, do not have the same requirements as those found in the automation of business processes. These differences in requirements call for new paradigms for workflow modeling. While several proposals have been put in place [9], e.g., workflow modeling languages based on variants of dataflow modeling, consolidation still needs to occur and consensus in the area is yet to be reached.

Cross-references

▶ Activity Diagrams
▶ Business Process Execution Language
▶ Business Process Management
▶ Business Process Modeling Notation
▶ Business Process Reengineering
▶ Petri Nets
▶ Workflow Model Analysis
▶ Workflow Patterns

Recommended Reading

1. Davenport T.H. Process Innovation: Reengineering Work through Information Technology. Harvard Business School, Boston, 1992.
2. Ellis C.A. Information control nets: a mathematical model of office information flow. In Proc. Conf. on Simulation, Measurement and Modeling of Computer Systems. 1979, pp. 225–240.
3. Hammer M. Reengineering work: don't automate, obliterate. Harvard Bus. Rev., July/August 1990, pp. 104–112.
4. Jablonski S. and Bussler C. Workflow Management: Modeling Concepts, Architecture, and Implementation. Int. Thomson Computer, London, UK, 1996.
5. Kiepuszewski B., ter Hofstede A.H.M., and Bussler C. On structured workflow modelling. In Proc. 12th Int. Conf. on Advanced Information Systems Engineering, 2000, pp. 431–445.
6. Kiepuszewski B., ter Hofstede A.H.M., and van der Aalst W.M.P. Fundamentals of control flow in workflows. Acta Inform., 39 (3):143–209, 2003.
7. Kindler E. On the semantics of epcs: resolving the vicious circle. Data Knowl. Eng., 56(1):23–40, 2006.
8. Levitt T. Marketing myopia. Harvard Bus. Rev., July/August 1960, pp. 45–56.
9. Ludäscher B., Altintas I., Berkley C., Higgins D., Jaeger E., Jones M., Lee E.A., Tao J., and Zhao Y. Scientific workflow management and the Kepler system. Concurrency Comput.–Pract. Exp., 18(10):1039–1065, 2006.
10. Muth P., Wodtke D., Weissenfels J., Dittrich A., and Weikum G. From centralized workflow specification to distributed workflow execution. J. Intell. Inform. Syst., 10(2), 1998.
11. Reichert M. and Dadam P. ADEPTflex: supporting dynamic changes of workflow without loosing control. J. Intell. Inform. Syst., 10(2):93–129, 1998.
12. van der Aalst W.M.P. and ter Hofstede A.H.M. YAWL: yet another workflow language. Inform. Syst., 30(4):245–275, 2004.
13. Zisman M.D. Representation, Specification and Automation of Office Procedures. PhD thesis, University of Pennsylvania, Warton School of Business, 1977.

Workflow on Grid

▶ Grid and Workflows

Workflow Participant

▶ Actors/Agents/Roles

Workflow Patterns

W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven, The Netherlands

Definition

Differences in features supported by the various contemporary commercial workflow management systems point to different insights of *suitability* and different levels of *expressive power*. One way to characterize these differences and to support modelers in designing non-trivial workflows, is to use a patterns-based approach. Requirements for workflow languages can be indicated through workflow *patterns*, i.e., frequently recurring structures in processes that need support from some process-aware information system. The Workflow Patterns Initiative [6] aims at the systematic identification of workflow requirements in the form of patterns.

The Workflow Patterns Initiative resulted in various collections of patterns. The initial twenty control-flow patterns have been extended with additional control-flow patterns and patterns for other perspectives such as the resource, data, and exception handling perspectives. All of these patterns have been used to evaluate different systems in a truly objective manner.

Key Points

The Workflow Patterns Initiative [6] was established with the aim of delineating the fundamental requirements that arise during business process modeling on a recurring basis and describe them in an imperative way. Based on an analysis of contemporary workflow products and modeling problems encountered in various workflow projects, a set of twenty patterns describing the control-flow perspective of workflow systems was identified [1]. Since their release, these patterns have been widely used by practitioners, vendors and academics alike in the selection, design and development of workflow systems.

An example of one of the initial workflow patterns is the *Discriminator* pattern which refers to the ability "to depict the convergence of two or more branches such that the first activation of an incoming branch results in the subsequent activity being triggered and subsequent activations of remaining incoming branches are ignored." It can be seen as a special case of the more general *N-out-of-M* pattern, where N is equal to one. While the more recent languages and systems support this pattern, many of the traditional workflow management systems have severe problems supporting this pattern. As a result, the designer is forced to change the process, to work around the system, or to create "spaghetti diagrams."

Based on many practical applications the original set of twenty control-flow patterns was extended in two directions. First of all, the set of control-flow patterns was extended and refined. There are now more than forty control-flow patterns, all formally described in terms of Petri nets. Second, patterns for other perspectives were added. The data perspective [5] deals with the passing of information, scoping of variables, etc, while the resource perspective [4] deals with resource to task allocation, delegation, etc. More than eighty patterns have been defined for both perspectives. The patterns for the exception handling perspective deal with the various causes of exceptions and the various actions that need to be taken as a result of exceptions occurring.

The resulting sets of patterns have been used for: (i) improving the understanding of workflow management and clarifying the semantics of different constructs, (ii) analyzing the requirements of workflow projects, (iii) evaluating products, and (iv) the training of workflow designers.

Cross-references

▶ Workflow Constructs
▶ Workflow Management
▶ Workflow Schema

Recommended Reading

1. van der Aalst W.M.P., ter Hofstede A.H.M., Kiepuszewski B., and Barros A.P. Workflow patterns. Distrib. Parallel Database, 14(1):5–51, 2003.
2. Casati F., Castano S., Fugini M.G., Mirbel I., and Pernici B. Using patterns to design rules in workflows. IEEE Trans. Softw. Eng., 26(8):760–785, 2000.
3. Hohpe G. and Woolf B. Enterprise Integration Patterns. Addison-Wesley, Reading, MA, 2003.
4. Russell N., van der Aalst W.M.P., ter Hofstede A.H.M., and Edmond D. Workflow resource patterns: identification, representation and tool support. In Proc. 17th Int. Conf. on Advanced Information Systems Eng., 2005, pp. 216–232.
5. Russell N., ter Hofstede A.H.M., Edmond D., and van der Aalst W.M.P. Workflow data patterns: identification, representation and tool support. In Proc. 24th Int. Conf. on Conceptual Modeling, 2005, pp. 353–368.
6. Workflow Patterns Home Page. http://www.workflowpatterns.com.

Workflow Scheduler

▶ Scheduler

Workflow Schema

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

Workflow Meta-Model

Definition

The meta structure of a business process defined in terms of a data model.

Key Points

In contemporary workflow the Workflow Schema is represented by a standard process definition such as XPDL (XML Process Definition Language) based on XML Schema. In non-standard environments or workflows defined with in specific application (not a BPMS or workflow management system) the Workflow Schema is built as a proprietary data model.

Cross-references

▶ Workflow Model
▶ Workflow Modeling
▶ XML Process Definition Language

Workflow Transactions

Vladimir Zadorozhny
University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

Transactional workflows; Transactional processes; Transactional business processes

Definition

Workflow Transaction is a unit of execution within a Workflow Management System (WFMS) that combines process-centric approach to modeling, executing and monitoring of a complex process in an enterprise with data-centric Transaction Management ensuring the correctness and reliability of the workflow applications in the presence of concurrency and failures. Workflow Transactions accommodate application-specific Extended Transaction Models (ETMs) that relax basic ACID properties of a classical database transaction models.

Key Points

Consider a Workflow Management System (WFMS) that models complex processes in an enterprise (e.g., processing insurance claim) using a workflow structured as a set of business tasks that should be performed in a special order. Some of the business tasks can be performed by humans while some of them are implemented as processes that typically create, process and manage information using loosely coupled distributed, autonomous and heterogeneous information systems. This *process-centric* approach to modeling of a complex enterprise process can be contrasted with *data-centric* approach accepted in database Transaction Management (TM) that keeps track of data dependencies, support concurrent data accesses, and crash recovery. Traditional TM approach is too strict for workflow applications, which requires selective use of transactional properties to allow flexible inter-task collaboration within a workflow. For example, classical ACID transaction model enforcing task isolation makes impossible task cooperation. Another example is large amount of task blocking and abortions caused by transaction synchronization, which is not acceptable for large-scale workflows. In order to address those issues Workflow Transactions accommodate application-specific Extended Transaction Models (ETMs) that relax some of the ACID properties.

There are several variations of the workflow transaction concept that depends on the way how the Workflow and Transaction Managements are merged. In general, workflows are more abstract than transactions and transactions provide specific semantics to workflow models. Alternatively, there are approaches where transactions are considered on a higher level of abstraction, while workflows define process structure for the transaction models. Workflows and transaction can also interact as peers at the same abstraction level within a loosely coupled process model. While workflow transaction commonly supports business applications, the workflow technology has also been adopted to support workflows for scientific explorations and discovery.

Cross-references

▶ Transaction Management
▶ Workflow Management

Recommended Reading

1. Georgakopoulos D., Hornick M., and Sheth A. An overview of workflow management: from process modeling to workflow automation infrastructure. Distrib. Parallel Databases, 3(2):119–153, 1995.
2. Grefen P. and Vonk J. A taxonomy of transactional workflow support. Int. J. Coop. Inf. Syst., 15(1):87–118, 2006.

Workflow/Process Instance Changes

▶ Workflow Evolution

World Wide Web Consortium

▶ W3C

WORM

▶ Storage Devices
▶ Storage Security
▶ Write Once Read Many

Wrapper

▶ Temporal Strata

Wrapper Adaptability

▶ Wrapper Stability

Wrapper Induction

Max Goebel[1], Michal Ceresna[2]
[1]Vienna University of Technology, Vienna, Austria
[2]Lixto Software GmbH, Vienna, Austria

Synonyms

Wrapper Generation; Information Extraction

Definition

Wrapper induction (or query induction) is a subfield of wrapper generation, which itself belongs to the broader field of information extraction (IE). In IE, wrappers transform unstructured input into structured output formats, and a wrapper generation systems describes the transformation rules involved in such transformations. Wrapper induction is a solution to wrapper generation where transformation rules are learned from examples and counterexamples (inductive learning). The induced wrapper subsequently is applied to unseen input documents to collect further label relations of interest. To ease annotation of examples by the user, the learning framework is often implemented within a visual annotation environment, where the user selects and deselects elements visually.

The term "wrapper induction" was first conceptualized by Nicholas Kushmerick in his influential PhD thesis in 1997 in the context of semi-structured Web documents [10], where the author also laid the theoretic foundations of wrapper induction in the PAC learning framework.

Historical Background

IE is a very mature research field that has spun off the database community in the need to integrate a constantly increasing number of unstructured or semi-structured documents into structured database systems.

The World Wide Web (WWW) as a means of an information sink has been accepted very quickly since its mainstream adoption in the early 1990s. Its low cost and wide spread has resulted in an enormous surge of data to be available on the Web. Particularly in the past decade, the number of documents of high informational value on the Web has exploded. The lack of rigid structure in Web documents together with the Web's vast value has initiated an independent research field for wrapper generation, which studies IE approaches particularly for semi-structured Web documents.

First IE approaches in the Web domain have seen systems where wrappers were constructed manually [13,16], yet it soon showed that manual wrapper systems did not scale well and were inflexible to change in document structure. The urge for more principled, and automated wrapper generation systems sparked the beginning of a whole new research sub-field of wrapper generation.

The beginnings of wrapper induction can be found in [10]. In this original work of Kushmerick, the author identified six wrapper classes and evaluated them in terms of learning theoretical concepts. The conclusion of his work was that quite expressive wrapper systems can be learned from surprisingly simple wrappers. His implementation prototype was called WIEN (1998), and it presents the first wrapper induction system.

Shortly after, SoftMealy and STALKER have been published. The SoftMealy system [7] (1998) relaxed the finite state transducer (Mealy machine) for Web documents, which brought great performance improvements over the WIEN system. The STALKER system [9,14] introduced hierarchical wrappers generated from landmark automata. In [17] (1999), the state machine approach was augmented with observation probabilities to form a Hidden Markov model for header extraction from research papers. BWI [6] (boosted wrapper induction, 2000) extended wrapper induction from its initial application on Web documents back to free text documents. It uses a probabilistic model for tuple length classification together with a boosting algorithm for delimiter learning.

In more recent work, [8] generate node predicates on the DOM tree representation of a HTML document (such as tagName, tagAttribute, #children, #siblings). These are constructed for each example node in the training set together with those nodes along the path towards the root from each such node – until a common ancestor is reached. Verification rules are generated from special verification predicates to be checked later. Finally, patterns with equal extraction results are combined.

In [3], the authors also use the DOM tree representation of web documents to detect repetitive occurrences of pre/post subtree constellations. They use boolean logic to combine all rules into a minimal disjunctive normal form over all training examples.

Applying supervised learning to Web wrapper generation bears a crucial drawback: sufficiently large sets of documents with annotated examples are hard to obtain and require expertise in the making. Labor and time investments renders the labeling of examples costly, creating a bottleneck in IE from Web documents. As a result, part of the research focused on tools that allow non-expert users to create wrappers interactively via visual user interaction. Starting with a set of non-annotated documents, the algorithm tries to infer the correct wrapper by asking the user to annotate a minimal number of examples.

Lixto [2], NoDoSe [1], Olera [4] are examples of interactive tools that allow users to build wrappers through a GUI. Finally, DEByE [12] (2002) is similar to STALKER and WIEN as it also used landmark automata for rule generation. Unlike all previously mentioned wrapper induction systems that generate extraction rules top-down (most general first), DEByE approaches the problem bottom-up, starting with the most specific extraction rule and generating from there.

Foundations

From Free Text to Web Documents

Wrapper generation systems nowadays focus on the Web domain, where documents are of a semi-structured format. Originally, wrapper generation systems were developed for free text documents. Free text exhibits no implicit structure at all, and extraction requires natural language processing techniques such as *Part-of-Speech tagging*. With the advent of the Web as a content platform, the importance of free text has dwindled in IE with respect to the importance of Web documents. Web documents are an intermediate format, lying between free text (unstructured) and structured text (e.g., XML). The use of HTML tags allows to apply relations between individual text segments in the sense of specifying structural information (see Fig. 1). One genuine problem for Web IE is that the lack of strict rules on these relations yields room for ambiguities of a tag's semantic meaning in different contexts. Web wrapper induction is mainly concerned with the exploitation of tag tree (DOM) structures.

Wrapper Induction

Kushmerick defined the wrapper induction problem as a simple model of information extraction. Input collection is referred to as *set of documents S*. An *extraction rule* (also query) allows to get a subset of the document according to some user-defined filter arguments. The result of a query is a set of information fields from *S*.

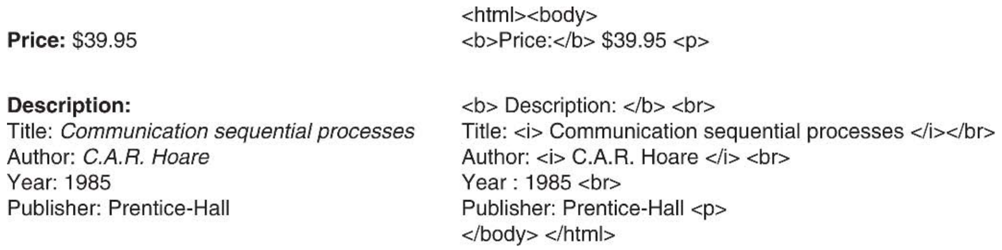
Information fields are made up of vectors of unique *attributes* which are referred to as *tuples*. The concept of attributes is borrowed from relational databases, where attributes correspond to table columns in the relational model. In some systems the order of the attributes is relevant, others allow permutations and missing attributes. The collection of all tuples describes the *content* of *S*. Given this definition, a wrapper can be formalized as the function mapping a page to its content.

$$page \Rightarrow wrapper \Rightarrow content$$

Different wrappers define different mappings. In turn, this means that the output of two different wrappers are two different subsets of all tuples defined in the page's content.

Patterns and Output Structure

Wrapping semi-structured resources is equivalent to extracting a hierarchy of elements from a document and returning as result a structured, relational



Price: $39.95

Description:
Title: *Communication sequential processes*
Author: *C.A.R. Hoare*
Year: 1985
Publisher: Prentice-Hall

```
<html><body>
<b>Price:</b> $39.95 <p>

<b> Description: </b> <br>
Title: <i> Communication sequential processes </i></br>
Author: <i> C.A.R. Hoare </i> <br>
Year : 1985 <br>
Publisher: Prentice-Hall <p>
</body> </html>
```

Wrapper Induction. Figure 1. Relational data in free text (*left*) and formatted in semi-structured HTML (*right*).

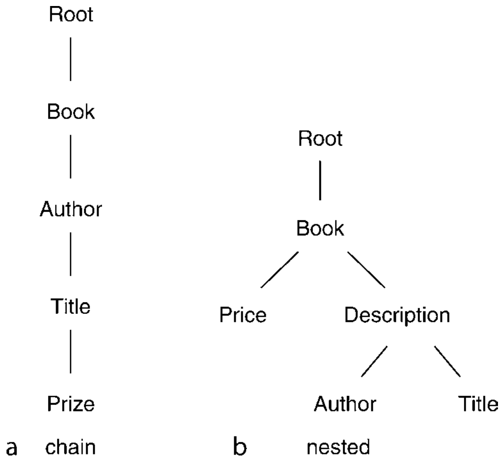
data model that reflects the extracted hierarchy. The hierarchy consists of *patterns* which can be seen as elementary wrapping operators. Each pattern in the hierarchy defines an extraction of one atomic tuple class from a given parent pattern. In the simplest case, data is structured in tabular form. Each tuple can be extracted separately and the attributes of each tuple are arranged in linear chains. A more difficult pattern structure are nested patterns where a parent pattern branches into multiple different child patterns (e.g., similar to a tree).

In Fig. 2, this is illustrated by extracting the pattern *author* from the parent pattern *description*.

Kushmerick's Wrapper Classes

In his PhD thesis, Kushmerick has analyzed delimiter-based wrappers for IE from strings. He devised six wrapper classes of varying complexity that can deal with the extraction of different pattern scenarios by identifying start and stop delimiters. In his formulation, Kushmerick defined the wrapper induction problem as a *Constraint Satisfaction Problem* (CSP) with all possible combinations of left and right delimiters as input. The constraints are formulated on the forbidden combinations on the input, which in turn vary with the wrapper class. What is interesting is that despite their simplicity, these wrapper classes yield decent results on a large benchmark set.

Wrapper Class LR LR (Left–Right) is the simplest wrapper class on which all subsequent classes are build upon. LR wrappers identify one delimiter token on each side of the tuple to be extracted, where L denotes the start token, and R denotes the end token. By taking all possible prefix and postfix strings in the document string for a given example tuple, the learning algorithm first identifies the candidate sets for the left and right delimiters. With L and R being mutually independent, a principled search can be performed via checking the validity of each candidate delimiter for the complete training set. This can also be expressed as constraints on the search space containing of all permutations of L and R delimiters.



Wrapper Induction. Figure 2. A wrapper represented by a hierarchy of patterns. Each pattern is defined by a set of rules describing how to locate its instances from its parent pattern.

Wrapper Class HLRT A more expressive wrapper class is Head-Left–Right-Tail, which is a direct extension from LR. It includes a page's head H and tail T for cases where similar delimiters are used for relevant and irrelevant information fields on a page. Delimiter H determines where to start looking for the LR pattern (the end of the head section), and T respectively determines where the search should end. Together, H and T define the search body of the page. The learning algorithm follows directly from the LR algorithm with the new constraints included. H and T must be learned jointly on l_1.

Wrapper Class OCLR Open–Close-Left–Right is an alternative to the HLRC wrapper class, where the head and tail delimiters are replaced with open and close delimiters. The difference here is that, contrary to the HLRT class where the complete search body for the wrapper is specified by the H and T delimiters, the O and C delimiters merely specify the beginning and end of a tuple in a page. It thus extends LR to a multi-slot extractor. O and C must be learned jointly on l_1.

Wrapper Class HOCLRT Head-Open–Close-Left–Right-Tail is the combination of the HLRT and the OCLR wrapper classes. It is the most powerful of the four basic wrapper classes, yet all four additional delimiters H, T, O and C must be learned jointly on l_1.

Wrapper Class N-LR The previous four wrapper classes work well on tabular (relational) data sets. As it is common to have hierarchical nestings in relational data, Kushmerick devised two extensions to the previous wrapper classes that can deal with nested data. Nested data are tree-like data structures where the attributes are nested hierarchically. The first nested wrapper class, N-LR, extends the simple LR class. It uses the relative position among attributes to determine which attribute to extract next, given a current attribute. Here, all attributes must be learned jointly.

Wrapper Class N-HLRT Finally, wrapper class N-HLRT combines the expressiveness of nested attributes with the head tail constraints from the HLRT class. It is exactly N-LR with the additional delimiters *H* and *T* to be considered in the hypothesis space of all candidate wrappers.

While in practice, Kushmerick showed these wrapper classes to be able to learn the correct wrapper from relatively few examples, the theoretical PAC bounds for the same wrappers are very loose [11].

Other Wrapper classes

Apart from delimiter-based wrapper classes, there also exist other wrapper classes that can prove highly effective, but which depend on the domain of the input documents. For Web wrapper induction, interesting classes include:

1. *Edit distance similarity.* The edit distance (Levensthein Distance) exists both for string and tree structures. It measures the difference between two strings or trees as the number of basic edit operations that need to be performed to turn one argument into the other. Due to the tree structure of the DOM (Document Object Model) of Web pages, this method lends itself very well to identifying tree patterns in Web documents.
2. *Rule-based systems.* Prolog-like rules are induced from examples. Typically, systems generate a most-general set of rules from the given examples and use covering algorithms to reduce the rules to a minimum set covering all examples. The advantage of rule-based wrappers is the relative ease to interpret rules by a user.

Active Learning

The key bottleneck of wrapper induction is the need for labeled training data to be available. An effective solution to reduce the number of annotations in a training corpus is offered by *active learning*. Active learning tries to identify examples that are not labeled but which are informative to the learning process, and the user is asked to label only such informative examples. Active learning has been applied to the Stalker wrapper generation system in [15], using *co-testing*. In co-testing, redundant views of a domain are implemented through mutually-exclusive feature sets per view. Views can be exploited by learning a separate classifier per view, as the ensemble of all classifiers reduces the hypothesis space for the wrapper. ***[5] provides a comprehensive overview of different active learning strategies, including common machine learning techniques such as bagging (multiple partitioning of training data) and boosting (ensemble learning).

Key Applications

Several of the presented wrapper induction systems are known for being successfully commercialized. The Lixto system [2] has emerged into an equally named spin-off company of the Technical University Vienna, Austria (http://www.lixto.com). Kushmerick's work [10] is being integrated into products of a US-based company called QL2 (http://www.ql2.com). Similarly, technologies created by the authors of the STALKER system [14] are used in products of another US company called Fetch Technologies (http://www.fetch.com). Web Intelligence solutions offered by these companies have customers for example in the travel, retail or consumer goods industries. Typical services built on top of the technology are on-demand price monitoring, revenue management, product mix tracking, Web process integration or building of Web mashups.

Wrapper Induction Algorithms

WIEN is the first system for wrapper induction from semi-structured documents. The system learns extraction rules in two stages: In a fist step, simple LR rules are generated from the training data, that specify left and right delimiters for each example. The second step refines the LR rules into example conforming HLRT rules, where additional Head and Tail delimiters are added if necessary. The WIEN algorithm assumes attributes to be equally ordered among all tuples, and it cannot deal with missing attributes. WIEN supports both single-slot and multi-slot patterns.

Softmealy is based upon a finite-state transducer and it utilizes contextual rules as extraction rules. Before being input to the transducer, a document is tokenized and special separators are added between two consecutive tokens. The transducer regards its input as a string of separators, where state transitions are governed by the left and right context tokens surrounding the current input separator. SoftMealy machines have two learnable components: the graph structure of the state transducer and the contextual rules governing the state transitions. The graph structure should be kept simple and small, which is achieved via a conservative extension policy. Contextual rules are learned by a set-covering algorithm that finds the minimal number of contextual rules covering all examples.

Stalker generates so-called landmark automata (SkipTo functions) from a set of training data. Nesting patterns allows the system to decompose complex extraction rules into a hierarchy of more simple extraction rules. Stalker employs a covering algorithm to reduce its initial set of candidate rules to a best-fit set of rules. Co-testing learns both forward and backward rules on the document. In case of agreement the system knows that the selected rule is correct, otherwise the user is given the conflicting example for labeling in order to resolve the conflict (active learning).

In boosted wrapper induction (*BWI*), delimiter-based extraction rules are generated from simple contextual patterns. Boundaries that uniquely describe relevant tuples are identified and assigned *fore* and *aft* scores. Learning in the BWI approach constitutes of determining fore detectors *F*, aft detectors *A*, and the pattern length function. The latter is a maximum likelihood estimator on the word occurrences in the training set. *F* and *A* are learned using a *boosting algorithm*, where several weak classifiers are boosted into one strong classifier. BWI works both in natural language and wrapper domains.

NoDoSe is an interactive tool that allows the user to hierarchically decompose a complex wrapper pattern into simple sub-wrappers. The system works both in the free text and the Web domain with the help of two distinct mining components. Learning rules consist of start and stop delimiters, either on text level (first component) or on HTML structure level (second component). Parsing generates a concept tree of rules that support multi-slot patterns and attribute permutation.

DEByE is another interactive induction system that applies bottom-up extraction rules. The authors define so-called attribute-value pair (AVP) patterns, and use window passages surrounding an example tuple to build lists of AVP patterns. Unlike other wrapper induction systems that typically work top-down, their approach allows for more flexibility as the order of individual patterns here is no longer relevant. They claim that, unlike other wrapper induction systems that typically work top-down, identifying atomic attribute objects prior to identifying the nested object itself adds flexibility to a system as partial objects are detected as well as nested objects with differing attribute orders. On the downside, this approach adds quite some complexity to the wrapper induction process.

Future Directions

Several trends can be identified in current wrapper induction research.

Shift to automation. The shifting of (semi-)supervised IE applications from free text to Web documents emphasized the cost problem of obtaining annotated examples from such sources. Unsupervised, fully automated methods using methods such as alignment, entity recognition, entity resolution or ontologies are motivated to overcome the so-called labelling bottleneck. Similarly, in semi-supervised systems, effort further focused to minimize the number of required examples (e.g., through active learning, see above).

Adoption of new technologies. Over time the complexity of Web sites has substantially increased. Technologies such as Web forms, AJAX or Flash are today used to build the so-called Deep Web. Many documents that are interesting for wrapper induction are therefore hidden in the Deep Web. Because usual crawling algorithm do not work well, new techniques such as focused crawling, Deep Web navigations, form mapping or metasearch receive increased research attention.

Shift to semantic annotations. The results of state-of-the-art wrapper induction systems support high confidence in success for many classic IE scenarios. With the quick and thourough adoption of the Web as a media and data communication protocol there comes an increasing demand for semantic services that allow users to enrich and interlink existing information with ontological meta-annotations. The envisioned semantic Web that would offer this functionality quickly proved to be a rather slow development process with no end in sight. Web IE systems on the other hand have started to add support for semantic labeling and even automatic detection of such relational information from labeled and unlabeled sources.

Data Sets

The most known data set used in information extraction is called RISE (http://www.isi.edu/info-agents/RISE). RISE is a collection of resources both from the Web and the natural text domains. Among others, it contains documents related to seminar announcements, Okra search, IAF address finder. Large fraction of the pages in the corpus originates before the year

2000. Because of their simplicity are those pages considered out-dated with respect to the current structure and layout of modern Web pages.

Cross-references

▶ Data Clustering (VII.b)
▶ Data Mining
▶ Decision Tree Classification
▶ Information Retrieval
▶ Semi-Structured Text Retrieval
▶ Text Mining

Recommended Reading

1. Adelberg B. NoDoSE: a tool for semi-automatically extracting structured and semistructured data from text documents. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 283–294.
2. Baumgartner R., Flesca S., and Gottlob G. Visual Web Information Extraction with Lixto. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 119–128.
3. Carme J., Ceresna M., and Goebel M. Web Wrapper Specification Using Compound Filter Learning. In Proc. IADIS Int. Conf. WWW/Internet 2006, 2006.
4. Chang C.H. and Kuo S.C. OLERA: Semisupervised web-data extraction with visual support. IEEE Intell. Syst., 19(6):56–64, 2004.
5. Finn A. and Kushmerick N. Active learning selection strategies for information extraction. In Proc. Workshop on Adaptative Text Extraction and Mining, 2003.
6. Freitag D. and Kushmerick N. Boosted Wrapper Induction. In Proc. 12th National Conf. on AI, 2000, pp. 577–583.
7. Hsu C.N. and Dung M.T. Generating Finite-state Transducers for Semi-structured Data Extraction from the Web. Inf. Syst., 23(8):521–538, 1998.
8. Irmak U. and Suel T. Interactive wrapper generation with minimal user effort. In Proc. 15th Int. World Wide Web Conf., 2006, pp. 553–563.
9. Knoblock C.A., Lerman K., Minton S., and Muslea I. Accurately and Reliably Extracting Data from the Web: a Machine Learning Approach. Q. Bull, IEEE TC on Data Eng., 23(4):33–41, 2000.
10. Kushmerick N. Wrapper Induction for Information Extraction. Ph.D. thesis, University of Washington, 1997.
11. Kushmerick N. Wrapper induction: Efficiency and expressiveness. Artif. Intell., 118(1–2):15–68, 2000.
12. Laender A.H.F., Ribeiro-Neto B., and da Silva A.S. DEByE – Date extraction by example. Data Knowl. Eng., 40(2):121–154, 2002.
13. Liu L., Pu C., and Han W. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In Proc. 16th Int. Conf. on Data Engineering, 2000, pp. 611–621.
14. Muslea I., Minton S., and Knoblock C. STALKER: Learning extraction rules for semistructured, Web-based information sources. 1998, URL citeseer.ist.psu.edu/muslea98stalker.html.
15. Muslea I., Minton S., and Knoblock C.A. Selective Sampling with Redundant Views. In Proc. 12th National Conf. on AI, 2000, pp. 621–626.
16. Sahuguet A. and Azavant F. WysiWyg web wrapper factory (W4F). 2001, URL http://citeseer.ist.psu.edu/553711.html; http://www.ai.mit.edu/people/jimmylin/papers/Sahuguet99.ps.
17. Seymore K., McCallum A., and Rosenfeld R. Learning hidden Markov model structure for information extraction. In Proc. AAAI 99 Workshop on Machine Learning for Information Extraction. 1999.

Wrapper Generation

▶ Wrapper Induction

Wrapper Generator

▶ Web Data Extraction System

Wrapper Generator GUIs

▶ GUIs for Web Data Extraction

Wrapper Maintenance

Kristina Lerman, Craig A. Knoblock
University of Southern California, Marina del Rey, CA, USA

Synonyms

Wrapper verification and reinduction; Wrapper repair

Definition

A Web wrapper is a software application that extracts information from a semi-structured source and converts it to a structured format. While semi-structured sources, such as Web pages, contain no explicitly specified schema, they do have an implicit grammar that can be used to identify relevant information in the document. A wrapper learning system analyzes page layout to generate either grammar-based or "landmark"-based extraction rules that wrappers use to extract data. As a consequence, even slight changes in the page layout can break the wrapper and prevent it from extracting data correctly. Wrapper maintenance is a composite task that (i) verifies that the wrapper continues to extract data correctly from a source, and (ii) repairs the wrapper so that it works on the changed pages.

Historical Background

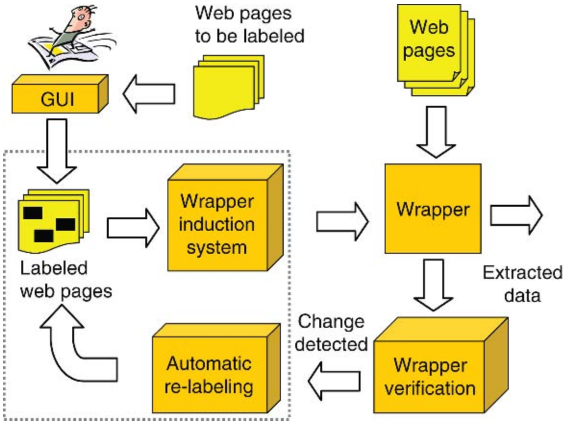
Wrapper induction algorithms [6,3,11] exploit regularities in the page layout to find a set of extraction rules that will accurately extract data from a source. As a consequence, even minor changes in page layout break wrappers, leading them to extract incorrect data, or fail to extract data from the page altogether. Researchers addressed this problem in two stages. *Wrapper verification* systems [5,9] monitor wrapper output to confirm that it correctly extracts data. Assuming that the structure and the format of data does not change significantly, they compare new wrapper output to that produced by the wrapper when it was successfully invoked in the past and evaluate whether the two data sets are similar. Wrapper verification systems learn data descriptions, also called content features, and evaluate wrapper output similarity based on these features. Two different representations of content features have been studied in the past. Kushmerick [4,5] and Chidlovskii [1] represent wrapper output by global numeric features, e.g., word count, average word length, HTML tag density, etc. Lerman et al. [8,9] instead learn syntactical patterns associated with data, e.g., they learn that addresses start with a number and are followed by a capitalized word. Both systems ascertain that the distribution of features over the new output is similar to that over data extracted by the wrapper in the past. If statistically significant changes are found, the verification system notifies the operator that the wrapper is no longer functioning properly.

If the wrapper is not working correctly, the next task is to automatically repair it by learning new extraction rules. This task is handled by a *wrapper reinduction* system. Since the wrapper learning algorithm simply needs a set of labeled examples of data on new pages in order to learn extraction rules, researchers have addressed the reinduction problem by automatically labeling data instances on new pages. The problem is more complex than simply looking for instances of old wrapper output on new pages because data can change in value (e.g., weather, stock quotes) or format (abbreviations added, etc.). Instead, to identify the new data instances, reinduction systems combine learned content features, heuristics based on page structure, and similarity with the old wrapper output. Lerman et al. [9] use syntactical patterns learned from old wrapper output to identify possible data instances on new pages. They then exploit regularities in the structure of data presented on the page to narrow candidates to ones likely to be the correct instances of a field. For example, a Web source listing restaurant information displays city, state and zipcode in the same order in all records. Likewise, instances of the same field are, in most cases, surrounded by the same HTML markup on all pages. Chidlobskii [1] take a similar approach, selecting data instance candidates based on learned numeric features, and exploiting page grammar and data structure to identify correct examples. Meng et al. [10] and Raposo et al. [12], on the other hand, rely on the document-object-model (DOM) tree to help identify correct examples of data on new pages.

Foundations

Figure 1 shows the life cycle of a wrapper. During its normal operation the wrapper is extracting data from the source, and learning content features that describe the data. The wrapper verification system monitors wrapper output to confirm that it is working correctly. Once it determines that a wrapper has stopped working, it uses the learned content features and page and data structure heuristics to identify data on the new pages. The labeled pages are then submitted to the wrapper induction system which learns new extraction rules.

Content-based features play a central role in both verification and reinduction. Data returned by online sources usually has some structure: phone numbers, prices, dates, street addresses, etc. all follow some format. Unfortunately, character-level regular expressions are too fine-grained to capture this structure. Instead,



Wrapper Maintenance. Figure 1. Life cycle of a wrapper, illustrating verification and reinduction stages.

researchers represent the structure of data as a sequence of tokens, called a pattern. Tokens are strings generated from an alphabet containing different character types: alphabetic, numeric, punctuation, etc. The token's character types determine its assignment to one or more syntactic categories: alphabetic, numeric, etc. The categories form a hierarchy, which allows for multi-level generalization. Thus, a set of zipcodes "90210" and "90292" can be represented by specific tokens "90210" and "90292," as well as more general types `5Digit`, `Number`, `Alphanum`. An efficient algorithm to learn patterns describing a field from examples was described in [9]. An alternate approach represents the content of data by global numeric features, such as character count, average word length, and density of types, i.e., proportion of characters in the training examples that are of an HTML, alphabetic, or numeric type. This approach would learn the following features for zipcodes: Count = 5.0, Digits = 1.0, Alpha = 0.0, meaning that zipcodes are, on average, five characters long, containing only numeric characters.

Wrapper Verification

Wrapper verification methods attempt to confirm that wrapper output is statistically similar to the output produced by the same wrapper when it was successfully invoked in the past. Henceforth, successful wrapper output are called training examples. Wrapper verification system learns content-based features and their distribution over the training examples. In order to check that the learned features have a similar distribution over the new wrapper output, one approach [5] calculates the probability of generating the new values randomly for each field. Individual field probabilities are then combined to produce an overall probability that the wrapper has extracted data correctly. Another approach to verification [9] uses training examples to learn data patterns for each field. The system then checks that this description still applies to the new data extracted by the wrapper.

Wrapper Reinduction

Content-based features are used by the wrapper reinduction system to automatically label the new pages. It first scans each page to identify all text segments that match the learned features. The candidate text segments that contain significantly more or fewer tokens than expected are eliminated from consideration. The learned features are often too general and will match many, possibly hundreds, of text segments on each page. Among these spurious text segments are the correct examples of the data field. Researchers use additional heuristics to help narrow the set to correct examples of the field: true examples are expected to appear in the same position (e.g., after a <b> tag) and in the same context (e.g., restaurant address precedes city name) on each page. In addition, they are expected to be visible to the user (not part of HTML tag) and appear within the same block of HTML page (or the same DOM sub-tree). This information is used to split candidate extracts into groups. The next step is to score groups based on their similarity to the training examples. This technique generally works well, because at least some of the data usually remains the same when the page layout changes. Of course, this assumption does not apply to data that changes frequently, such as weather information, flight arrival times, stock quotes, etc. However, previous research has found that even in these sources, there is enough overlap in the data that the approach works.

The final step of the wrapper reinduction process is to provide the extracts in the top-scored group to the wrapper induction algorithm, along with the new pages, to learn new data extraction rules.

For some types of Web sources automatic wrapper generation is a viable alternative to wrapper reinduction. One such system, developed by Crecenzi et al. [2], learns grammar-based extraction rules without requiring the user to label any pages. Once wrapper verification system detects that the wrapper is not working correctly, it collects new sample pages from which it learns new extraction rules. Still another approach to data extraction uses a combination of content-feature learning and page analysis to automatically extract data from Web pages [7].

Key Applications

Automatic wrapper maintenance can be incorporated into a wrapper learning system to reduce the amount of user effort involved in monitoring wrappers for validity and repairing them.

Experimental Results

To evaluate the performance of wrapper maintenance systems, researchers collected data over time, saving Web pages along with data extracted from them by wrappers. In one set of experiments researchers [9] monitored 27 wrappers (representing 23 distinct data sources) by storing results of 15–30 queries periodically over 10 months. The wrapper verification system learned content descriptions for each field and made a decision about whether the new output was statistically similar to the training set.

A manual check of the 438 comparisons revealed 37 wrapper changes attributable to changes in the page layout. The verification algorithm correctly discovered 35 of these changes and made 15 mistakes. Of these mistakes, 13 were *false positives*, which means that the verification program decided that the wrapper failed when in reality it was working correctly. Only two of the errors were the more important *false negatives*, meaning that the algorithm did not detect a change in the data source. Representing the structure of content by global numeric features instead would have missed 17 wrapper changes.

Researchers also evaluated the reinduction algorithm only on the ten sources that returned a single tuple of results per page. They used the algorithm to extract (35) fields from ten Web sources for which correct output is known, regardless of whether the source had actually changed or not.

The output of the reinduction algorithm is a list of tuples extracted from ten pages, as well as extraction rules generated by the wrapper induction system for these pages. Though in most cases they were not able to extract every field on every page, they still learned good extraction rules as long as a few examples of each field were correctly labeled. The algorithm was evaluated in two stages: first, researchers checked how many fields were successfully identified; second, they checked the quality of the learned extraction rules by using them to extract data from test pages.

The algorithm was able to correctly identify fields 277 times across all data sets making 61 mistakes, of which 31 were attributed to false positives and 30 to the false negatives. On the extraction task, the automatically learned extraction rules achieved 90% precision and 80% recall.

The results above apply to sources that return a single item per page. Other researchers have applied reinduction methods to repair wrappers for sources that return lists of items per page [1,13] with similar performance results.

Data Sets

The data set used in the wrapper maintenance experiments discussed above is available from the University of Southern California Information Sciences Institute: http://www.isi.edu/integration/datasets/reinduction/wrapper-verification-data2.zip.

The data contains results of invocations of wrappers for different Web sources over a period of 10 months. Each wrapper was executed with the same small set of queries (if they contained no time-dependent parameters). The data set contains both the returned Web pages and data extracted by the wrapper from the source. Over the data collection period, several sources changed in a way that prevented wrappers from correctly extracting data. The wrappers were repaired manually in these cases and data collection continued. The performance of the USC's verification and reinduction systems on this dataset is reported in [9].

Cross-references

- ▶ Web Data Extraction System
- ▶ Web Information Extraction
- ▶ Wrapper Induction

Recommended Reading

1. Chidlovskii B. Automatic repairing of web wrappers by combining redundant views. In Proc. 14th IEEE Int. Conf. Tools with Artificial Intelligence, 2002, pp. 399–406.
2. Crescenzi V. and Mecca G. Automatic information extraction from large websites. J. ACM, 51(5):731–779, 2004.
3. Hsu C.-N. and Dung M.-T. Generating finite-state transducers for semi-structured data extraction from the web. J. Inform. Syst., 23:521–538, 1998.
4. Kushmerick N. Regression testing for wrapper maintenance. In Proc. 14th National Conf. on AI, 1999, pp. 74–79.
5. Kushmerick N. Wrapper verification. World Wide Web J., 3(2):79–94, 2000.
6. Kushmerick N., Weld D.S., and Robert B. Doorenbos. Wrapper induction for information extraction. In Proc. 15th Int. Joint Conf. on AI, 1997, pp. 729–737.
7. Lerman K., Gazen C., Minton S., and Knoblock C.A. Populating the semantic web. In Proc. AAAI Workshop on Advances in Text Extraction and Mining, 2004.
8. Lerman K. and Minton S. Learning the common structure of data. In Proc. 12th National Conf. on AI, 2000, pp. 609–614.
9. Lerman K., Minton S., and Knoblock C. Wrapper maintenance: a machine learning approach. J. Artif. Intell. Res., 18:149–181, 2003.
10. Meng X., Hu D., and Li C. Schema-guided wrapper maintenance. In Proc. of 2003 Conf. on Web Information and Data Management, 2003, pp. 1–8.
11. Muslea I., Minton S., and Knoblock C.A. Hierarchical wrapper induction for semistructured information sources. Auton. Agent Multi Agent Syst., 4:93–114, 2001.

12. Raposo J., Pan A., Alvarez M., and Hidalgo J. Automatically generating labeled examples for web wrapper maintenance. In Proc. of 2005 IEEE/WIC/ACM Int. Conf. on Web Intelligence, 2005, pp. 250–256.
13. Raposo J., Pan A., Álvarez M., and Hidalgo J. Automatically maintaining wrappers for semi-structured web sources. Data Knowl. Eng., 61(2):331–358, 2007.

Wrapper Repair

▶ Wrapper Maintenance

Wrapper Robustness

▶ Wrapper Stability

Wrapper Stability

Georg Gottlob
Oxford University, Oxford, UK

Synonyms

Wrapper robustness; Wrapper adaptability

Definition

A *wrapper*, whether hand written or generated by a *Web data extraction system*, is a program that extracts data from information sources of changing content and translates the data into a different format. The *stability* of a wrapper is the degree of insensitivity to changes of the presentation (i.e., formatting, layout, or syntax) of the data sources. A stable wrapper is ideally able to extract the desired data from source documents even if the layout of the current documents differs from the layout of those documents that were used as examples at the time the wrapper was generated. Thus, a wrapper is stable or robust if it is able of coping with perturbations of the original layout.

Key Points

Documents, and, Web pages in particular are usually susceptible to slight layout changes over time. Most of these changes are minor changes. For example, an advertisement may be added to the top of a Web page. One would ideally wish that changes in the layout of the source will not prevent a wrapper to extract the desired data, i.e., that the wrapper remains stable under such changes (or, equivalently, robust to these changes), as long as the data items to be extracted still appear in the document. For obvious reasons, totally stable wrappers can hardly be generated. However, there are sensible differences in stability between simple *screen scrapers* or *Web scrapers* on one hand, and wrappers generated by advanced *Web data extraction systems*, on the other hand. Screen scrapers or Web scrapers often rely on exact data positions in a document or in the parse tree of an HTML Web page. For example, a screen scraper may memorize that a certain price to extract from a Web page is the data item occurring as the 37th leaf of the Web page's parse tree. Of course, in case the page's layout is altered by adding, say, an advertisement in front of the data, such a wrapper will fail to extract the intended price. Advanced Web data extraction systems, on the other hand, characterize the objects to be extracted via syntactic, contextual, or semantic properties rather than by their position only. For example, the Lixto system [1] could characterize a desired price as a numeric data item preceded by a currency symbol, occurring in the second column of a table entitled "Price list," such that a certain article name appears in the first column of the same row. Clearly, such a characterization is much more stable than a purely positional one. There are only very few studies on Wrapper stability, among which, for example [2], and this topic remains an important one for future research. One issue of particular interest is the automatic adaptation of wrappers to new layouts [2–5].

Cross-references

▶ Web Data Extraction System

Recommended Reading

1. Baumgartner R., Flesca S., and Gottlob G. Visual web information extraction with Lixto. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 119–128.
2. Davulcu H., Yang G., Kifer M., and Ramakrishnan I.V. Computational aspects of resilient data extraction from semi-structured sources. In Proc. 19th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2000, pp. 136–144.
3. Meng X., Hu D., and Li C. Schema-guided wrapper maintenance for web-data extraction. In Proc. Fifth ACM CIKM Int. Workshop on Web Information and Data Management, 2003, pp. 1–8.
4. Phelps T.A. and Wilensky R. Robust intra-document locations. In Proc. 9th Int. World Wide Web Conf., 2000, pp. 105–118.

5. Wong T. and Lam W. A probabilistic approach for adapting information extraction wrappers and discovering new attributes. In Proc. 2004 IEEE Int. Conf. on Data Mining, 2004, pp. 257–264.

Wrapper Verification and Reinduction

► Wrapper Maintenance

Write Once Read Many

Kenichi Wada
Hitachi Limited, Tokyo, Japan

Synonyms

Write once read mostly; Write one read multiple; WORM

Definition

A data storage technology that allows information to be written to storage media a single time only, and prevents the user from accidentally or intentionally altering or erasing the data. Optical disc technologies such as CD-R and DVD-R are typical WORM storage.

Key Points

There are two types of WORM technology: physical and logical WORM.

1. Physical WORM uses storage media which physically can be written only once, and prevents the user from accidentally or intentionally altering or erasing the data. Optical storage media such as CD-R and DVD-R are good examples. Offering fast access and long-term storage capability, physical WORM has historically been used for archiving data that requires a long retention period.
2. Logical WORM uses storage systems which provide a WORM capability by using electric keys or other measures to prevent rewriting even if with non-WORM storage media, such as disk drives. Driven by the recent growth of legislation in many countries, logical WORM is now increasingly being used to archive corporate data, such as financial documents, emails, and certification documents, and to find them quickly in case of discovery.

Cross-references

► Digital Archives and Preservation

Write Once Read Mostly

► Write Once Read Many

Write One Read Multiple

► Write Once Read Many

WS-BPEL

► Composed services and WS-BPEL

WS-Discovery

► Discovery

X

XA Standard

▶ Two-Phase Commit Protocol

XMI

▶ XML Metadata Interchange

XML

Michael Rys
Microsoft Corporation, Sammamish, WA, USA

Synonyms

Extensible markup language; XML 1.0

Definition

The Extensible Markup Language or XML for short is a markup definition language defined by a World Wide Web Consortium Recommendation that allows annotating textual data with tags to convey additional semantic information. It is extensible by allowing users to define the tags themselves.

Historical Background

XML was developed as a simplification of the ISO Standard General Markup Language (SGML) in the mid 1990s under the auspices of the World Wide Web Consortium (W3C). Some of the primary contributors were Jon Bosak of Sun Microsystems (the working group chair), Tim Bray (then working at Textuality and Netscape), Jean Paoli of Microsoft, and C. Michael Sperberg-McQueen of then the University of Chicago. Initially released as a version 1.0 W3C recommendation on 10 Feb. 1998, XML has undergone several revisions since then. The latest XML 1.0 recommendation edition is the fourth edition as of this writing. A fifth edition is currently undergoing review. The fifth edition is adding some functionality into XML 1.0 that was part of the XML 1.1 recommendation, that has achieved very little adoption.

Based on the XML recommendation, a whole set of related technologies have been developed, both at the W3C and other places. Some technologies are augmenting the core syntactic XML recommendation such as the XML Namespaces and XML Information Set recommendations, others are building additional infrastructure on it such as the XML Schema recommendation or the XSLT, XPath and XQuery family of recommendations. Since XML itself is being used to define markup vocabularies, it also forms the basis for vertical industry standards in manufacturing, finance and other areas, including standard document formats such as XHTML, DocBook, Open Document Format (ODF) and Office Open XML (OOXML) and forms the foundation of the web services infrastructure.

Foundations

XML's markup format is based on the notion of defining well-formed documents and is geared towards human-readability and international usability (by basing it on Unicode). Among its design goals were ease of implementation by means of a simple specification, especially compared to its predecessor, the SGML specification, and to make it available on a royalty-free basis to both implementers and users.

Well-formed documents basically contain markup elements that have a begin tag and an end tag as in the example below:

```
< tag > character data < /tag >
```

Element tags can have attributes associated with it that provide information about the tag, without interfering with the flow of the textual character data that is being marked up:

```
<tag attribute1 = “value” attribute2 = “42”>
    character data </tag>
```

Processing instructions can be added to convey processing information to XML processors and comments can be added. And comments can be added for commenting and documentation purposes. = They follow different syntactic forms than element tags and can appear anywhere in a document, except within the begin tag and end tag tokens themselves:

```
<?xml-stylesheet type=“application/xslt + xml”
              href=“#style1”" ? >
<!- This is a comment –>
```

A well-formed document must also have exactly one top-level XML element, and can contain several processing instructions and comments on the top-level next to the element.

The order among these elements is information-bearing, since they are meant to mark up an existing document flow. Thus, the following two well-formed XML documents are not the same:

```
<doc> <element> value1 </element>
<element> value2 </element> </doc>
<doc> <element> value2 </element>
<element> value1 </element> </doc>
```

The XML information set recommendation defines an abstract data model for these syntactic components, introducing the notion of document information items for a document, element information items for element tags, attribute information items for their attributes, character information items for the marked up character data etc.

The XML namespace recommendation adds the ability to scope an element tag name to a namespace URI, to provide the ability to scope markup vocabularies to a domain identifier.

Besides defining an extensible markup format, the XML recommendation also provides a mechanism to constrain the XML document markup to follow a certain grammar by restricting the allowed tag names and composition of element tags and attributes with document type declarations (DTDs). Documents that follow the structure given by DTDs are not only well-formed but also valid. Note that the XML Schema recommendation provides another mechanism to constrain XML documents.

Finally, XML also provides mechanisms to reuse parts of a document (down to the character level) using so called entities.

For more information about XML, please refer to the recommended reading section.

Key Applications

While XML was originally designed as an extensible document markup format, it has quickly taken over tasks in other areas due to the wide-availability of free XML parsers, its readability and flexibility. Besides the use for document markup, two of the key application scenarios for XML are the use for interoperable data interchange in loosely-coupled systems and for adhoc modeling of semi-structured data.

XML's first major commercial applications actually have been to describe the data and, with DTDs or XML schema formats, structures of messages that are being exchanged between different computer systems in application to application data exchange scenarios and web services. XML is not only being used to describe the message infrastructure format and information such as the SOAP protocol, RSS or Atom formats, but also to describe the structure and data of the message payloads. Often, XML is also used in more adhoc micro-formats for more REST-ful web services.

At the same time that XML was being developed, several researcher groups were looking into data models that were less strict than relational, entity-relationship and object-oriented models, by allowing instance based properties and heterogeneous structures. XML's tree model provides a good fit to represent such semi-structured, hierarchical properties, and its flexible format is well–suited to model the sparse properties and rapidly changing structures that are often occurring in semi-structured data. Therefore, XML has often been used to model semi-structured data in data modeling.

XML support has been added to databases on form of either pure XML databases or by extending existing database platforms such as relational database systems to enable databases to manage XML documents serving all these three application scenarios.

Cross-references

- ▶ XML Attribute
- ▶ XML Document
- ▶ XML Element
- ▶ XML Schema
- ▶ XPath/XQuery
- ▶ XSL/XSLT

Recommended Reading

1. Namespaces in XML 1.0, latest edition. Available at: http://www.w3.org/TR/xml-names
2. Wikipedia entry for XML. Available at: http://en.wikipedia.org/wiki/XML
3. XML 1.0 information Set, latest edition. Available at: http://www.w3.org/TR/xml-infoset
4. XML 1.0 recommendation, latest edition. Available at: http://www.w3.org/TR/xml
5. XML 1.1 recommendation, latest edition. Available at: http://www.w3.org/TR/xml11

XML (Almost)

► Semi-structured Data

XML 1.0

► XML

XML Access Control

Dongwon Lee[1], Ting Yu[2]

[1]The Pennsylvania State University, University Park, PA, USA

[2]North Carolina State University, Raleigh, NC, USA

Definition

XML access control refers to the practice of limiting access to (parts of) XML data to only authorized users. Similar to access control over other types of data and resources, XML access control is centered around two key problems: (i) the development of formal models for the specification of access control policies over XML data; and (ii) techniques for efficient enforcement of access control policies over XML data.

Historical Background

Access control is one of the fundamental security mechanisms in information systems. It is concerned with who can access which information under what circumstances. The need for access control arises naturally when a multi-user system offers selective access to shared information. As one of the oldest problems in security, access control has been studied extensively in a variety of contexts, including operating systems, databases, and computer networks.

The most influential policy models today are discretional access control (DAC), mandatory access control (MAC), and role-based access control (RBAC) models. In DAC, the owner of an object (e.g., a file or database table) solely determines which subjects can access that object, and whether such privileges can be further delegated to other subjects. In MAC, whether a subject can access an object or not is determined by their security classes, not by the owner of the object. In RBAC, privileges are associated with roles. Users are assigned to roles, and thus can only exercise access privileges characterized by their roles.

Typical implementations of access control are in the form of access control lists (ACLs) and capabilities. In ACLs, a system maintains a list for each object of subjects who have access to that object. In capabilities, each subject is associated with a list that indicates those objects to which it has access. ACLs and capabilities are suitable to enforce coarse-grained access control over objects with simple structures (e.g., file systems or table-level access control in relational databases). They are often not efficient for fine-grained access control over objects with complex structures (e.g., element-level access control in XML, or row-level and cell-level access in relational databases).

Foundations

XML access control is fine-grained in nature. Instead of controlling access to the whole XML database or document, it is often required to limit a user's access to some substructures of an XML document (e.g., some subtrees or some individual elements).

An XML access control policy can be typically modeled as a set of access control rules. Each rule is a 5-tuple (*subject*, *object*, *action*, *sign*, *type*), where (i) *subject* defines a set of subjects; (ii) *object* defines a set of elements or attributes of an XML document; (iii) *action* denotes the actions that can be performed on XML (e.g., read, write, and update); (iv) *sign* $\in\{+, -\}$ indicates whether this rule is a grant rule or a deny rule; and (v) *type* $\in\{LC, RC\}$ refers to either *local check* or *global check*. Intuitively, an access control rule specifies that subjects in *subject* can (when *sign*=+) or cannot (when *sign*=−) perform action specified by *action* on those elements or attributes in *objects*. When *type*=*RC*, this authorization decision also applies to the descendants of those in *object*.

Object is usually specified using some XML query languages such as XPath expressions. There are multiple ways to identify *subject*. Following RBAC, *subject* can be specified as one or several roles (e.g., student and faculty) [4]. It may also follow *attribute-based access control*, where each user features a set of attributes and *subject* is defined based on the values of those attributes (e.g., those subjects whose age is over 21). Some access control policy in the literature also follows MAC, where *subject* refers to security classes.

Example 1. Consider two access control rules for the subject of "admin" as follows:

R_1: (*admin, /people/person/name, read,* −*, LC*)
R_2: (*admin, /people//address//*, read/update,* +*, RC*)

R_1 Indicates that users belonging to the admin role cannot read textual and attribute data of XML node <name>, the child of <person> that is the child of the root node <people>. On the other hand, R_2 specifies that the same users of the admin role can read and even update any XML data that are descendents of XML node if they are descendents of <address> under <people>.

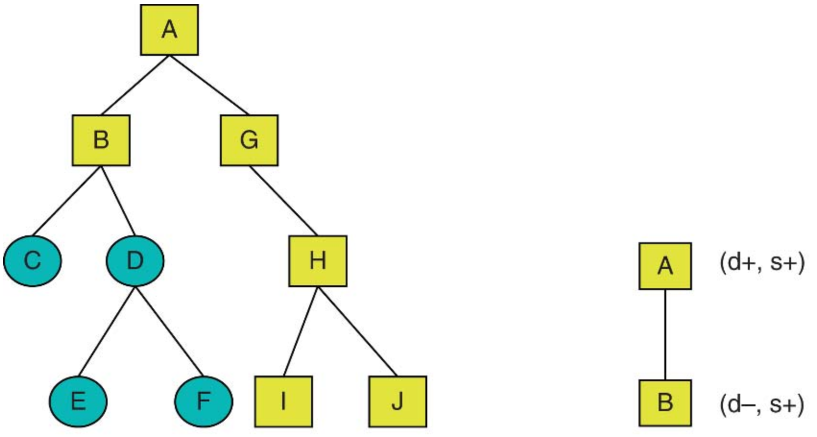
Once an access control policy is specified, there are two problems that need to be addressed. First, given any access request, one needs to determine whether or not a rule exists that applies to the request. If so, the policy is said to be *complete*. Most access control systems adopt a closed world assumption. That is, if no rules apply to a request, the request is denied. Second, multiple rules with different authorization decisions may apply to a request. One typical way to resolve such conflicts is to let denial override permit. For XML access control, it may also be solved by having more explicit rules override less explicit ones. For instance, an LC rule may override an RC rule. For two RC rules, the one that applies to a node's nearest ancestor usually dominates.

Languages for specifying access control policy are proposed in such efforts as XACL by IBM [7]. Therefore, it is also possible to use much more expressive languages to specify access control policy within XML access control models. Finally, the use of authorization priorities with propagation and overriding are related to similar techniques studied in object-oriented databases.

Once XML access control is specified in a given model, it can be enforced in a variety of ways. By and large, most of the existing XML access control methods are either *view-based* or rely on the XML engine to enforce access control at the node-level of XML trees. The idea of view-based enforcement (e.g., [5,11]) is to create and maintain a separate view for each user (or role) who is authorized to access a specific portion of an XML data [4]. The view contains exactly the set of data nodes that the user is authorized to access. After views are constructed, during run time, users can simply run their queries against the views without worrying about access control issues. Although views can be prepared offline, in general, view-based enforcement schemes suffer from high maintenance and storage costs, especially for a large number of roles: (i) a virtual or physical view is created and stored for each role; (ii) whenever a user prompts *update* operation on the data, all views that contain the corresponding data need to be synchronized. To tackle this problem, people proposed a method using compressed XML views to support access controls [11]. The basic idea is based on the observation of accessibility locality, i.e., elements close to each other tend to have the same accessibility for a subject. Based on this observation, a compressed accessibility map only maintains some "crucial" nodes in the view. With simple annotation on those crucial nodes, other nodes' accessibility can be efficiently inferred instead of explicitly stored. Each node in the compressed view is associated with a label (*desc*, *self*), where *desc* can be either d+ or d−, indicating whether its descendants are accessible or not, and *self* can be either s+ or s−, indicating whether the node itself is accessible. Given any node in an XML tree, by its relationship to those labeled nodes in the compressed view, we can infer its accessibility.

Example 2. Consider the XML tree in Fig. 1a with squares and circles denoting accessible and inaccessible nodes, respectively. The corresponding compressed view is shown in Fig. 1b. Note that since node C is a descendant of B and B is labeled (d−, s+), C can be inferred to be inaccessible.

In the non view-based XML access control techniques (e.g., [2,9,10]), an XML query is pre-classified against the given model to be "entirely" authorized, "entirely" prohibited, or "partially" authorized before being submitted to an XML engine. Therefore, without using pre-built views, those entirely authorized or prohibited queries can be quickly processed. Furthermore, those XML queries that are partially authorized are re-written using state machines such that they request for only data that are granted to the users or roles.



a XML document as a tree

b Compressed view

Example 3. Consider three access control rules for the security role "admin." Furthermore, an administrator "Bob" requested three queries, Q_1 to Q_3 in XPath as follows:

R_1: (admin, /people/person/name, read, –, LC)
R_2: (admin, /people//address//, read/update, +, RC)*
R_3: (admin, /regions/namerica/item/name, read, +, LC)
Q_1: /people/person/address/street
Q_2:/people/person/creditcard
*Q_3:/regions//**

Then, Q_1 by Bob can be entirely authorized by both R_1 and R_2, but entirely denied by R_3. Similarly, Q_2 is entirely authorized by R_1, entirely denied by both R_2 and R_3. Finally, Q_3 is entirely accepted by R_1, entirely denied by R_2, and partially authorized by R_3 and needs to be re-written to /regions/namerica/item/name in order not to be conflicted with $R_{3.}$

Key Applications

As XML has been increasingly used not only as a data exchange format but as a data storage format, the problem of controlling selective access over XML is indispensable to data security. Therefore, XML access control issues have been tightly associated with secure query processing techniques in relational and XML databases (e.g., [3,5]). The access control policy model of XML can also be extended to express security requirements of other semi-structured data such as LDAP and object-oriented databases. The aforementioned access control enforcement techniques can be further extended to protect privacy during the exchange of XML data in distributed information sharing system. For instance, in PPIB system [8], XML access control is used to hide what query content is or where data objects are located, etc. In an environment where XML data are stored in a distributed fashion and users may ask sensitive queries whose privacy must be kept to is utmost extent (e.g., HIV related queries in health information network), XML access control techniques can be used, along with XML content-based routing techniques [6].

Cross-references

► Relational Access Control
► Secure XML Query Processing

Recommended Reading

1. Bertino E. and Ferrari E. Secure and selective dissemination of XML documents. ACM Trans. Inform. Syst. Secur., 5(3):290–331, 2002.
2. Bouganim L., Ngoc F.D., and Pucheral P. Client-based access control management for XML documents. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 84–95.
3. Cho S., Amer-Yahia S., Lakshmanan L.V.S., and Srivastava D. Optimizing the secure evaluation of twig queries. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 490–501.
4. Damiani E., Vimercati S., Paraboschi S., and Samarati P. A fine-grained Access Control System for XML Documents. ACM Trans. Inform. Syst. Secur., 5(2):169–202, 2002.
5. Fan W., Chan C.-Y., and Garofalakis M. Secure XML querying with security views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 587–598.
6. Koudas N., Rabinovich M., Srivastava D., and Yu T. Routing XML queries. In Proc. 20th Int. Conf. on Data Engineering, 2004, p. 844.
7. Kudo M. and Hada S. XML document security based on provisional authorization. In Proc. 7th ACM Conf. on Computer and Communications Security, 2002, pp. 87–96.
8. Li F., Luo B., Liu P., Lee D., and Chu C.H. Automaton segmentation: a new approach to preserve privacy in XML information brokering. In Proc. 14th ACM Conf. on Computer and Communications Security, 2007, pp. 508–518.
9. Luo B., Lee D., Lee W.C., and Liu P. QFilter: fine-grained run-time XML access control via NFA-based query rewriting.

In Proc. Int. Conf. on Information and knowledge Management, 2004, pp. 543–552.

10. Murata M., Tozawa A., and Kudo M. XML access control using static analysis. In Proc. 10th ACM Conf. on Computer and Communication Security, 2003, pp. 73–84.
11. Yu T., Srivastava D., Lakshmanan L.V.S., and Jagadish H.V. A compressed accessibility map for XML. ACM Trans. Database Syst., 29(2):363–402, 2004.

XML Algebra

▶ XML Tuple Algebra

XML Application Development

▶ XML Programming

XML Attribute

Michael Rys
Microsoft Corporation, Sammamish, WA, USA

Synonyms

XML attribute; XML attribute node

Definition

An XML attribute is used to represent additional information on an XML element in the W3C XML recommendation [2].

Key Points

The name of an XML attribute has to be unique for a given element. Therefore, the following XML element is not allowed

<e a1 = “v1” a1 = “v2”/>
while the following is well-formed
<e1 a1 = “v1”> <e2 a1 = “v2”/> </e1>

An XML attribute has a value that follows the attribute value normalization rules outlined in section 3.3.3 of [2]. This means that several whitespace characters do not get preserved in attribute values, unless they are explicitly represented with a character entity (e.g., &#xD; for carriage return).

Attributes can be constrained and typed by schema languages such as XML DTDs [2] or XML Schema [3].

An XML attribute is represented as an XML attribute information item in the XML Information Set [1] and an XML attribute node in the XPath and XQuery data model [4].

Cross-references

▶ XML
▶ XML Document
▶ XML Element

Recommended Reading

1. XML 1.0 Information Set, latest edition available online at: http://www.w3.org/TR/xml-infoset.
2. XML 1.0 Recommendation, latest edition available online at: http://www.w3.org/TR/xml.
3. XML Schema Part 0: Primer, latest edition available online at: http://www.w3.org/TR/xmlschema-0/.
4. XQuery 1.0 and XPath 2.0 Data Model (XDM), latest edition available online at: http://www.w3.org/TR/xpath-datamodel/.

XML Attribute Node

▶ XML Attribute

XML Benchmarks

Denilson Barbosa[1], Ioana Manolescu[2],
Jeffrey Xu Yu[3]
[1]University of Alberta, Edmonton, AB, Canada
[2]INRIA, Saclay–Île-de-France, Orsay, France
[3]The Chinese University of Hong Kong, Hong Kong, China

Definition

An XML benchmark is a specification of a set of meaningful and relevant tasks, intended to assess the functionality and/or performance of an XML processing tool or system. The benchmark must specify the following: (i) a deterministic workload, consisting of a set of XML documents and/or a procedure for obtaining these and a set of operations to be performed;

(ii) detailed rules for executing the workload and making the measurements; (iii) the metrics used to report the results of the benchmark; and (iv) standard ways of interpreting the results.

Historical Background

XML has quickly become the preferred format for representing and exchanging data on the Web age. The level of acceptance of XML is astonishing, especially when one considers that this technology was introduced only in 1997. XML is an enabling technology with applications in virtually all domains of information processing. At the time of writing, XML is widely used in content distribution on the Web (e.g., RSS feeds), as the foundation of large initiatives such as the Semantic Web and Web Services, and is the basis for routinely used productivity tools, such as text editors and spreadsheets.

Such complexity led to the development of a large number of fairly narrowly-scoped benchmarks for XML. Moreover, there is still no clear understanding of what an XML benchmark should look like.

The first XML benchmarks were developed in academia for testing specific processing tasks and/or relatively narrow applications. In fact, some of these earlier benchmarks are categorized as micro-benchmarks. For example, XMach emulated the scenario of an XML file servers using a large number of simple XML files, while XMark modeled an online auction application using a single complex XML document. Over time, XML benchmarks evolved to contemplate more realistic and complex application scenarios, in an attempt to emulate the typical workload of larger applications. For instance, XBench modeled four scenarios, resulting from the combination of two factors: (i) the nature of the documents (data-centric vs. document-centric); and (ii) the number of documents in the workload (single-document vs. multi-document).

The first XML Benchmark developed entirely by industry was TPoX (Transaction Processing over XML), which simulates a financial application in a multi-user environment with concurrent access to the XML data. TPoX is intended for testing all aspects of a relational-based XML storage and processing system.

Another development worth mentioning concerns the declarative synthetic data generators for XML. While, strictly speaking, these are not benchmarks, these tools help in obtaining appropriate testing data with reasonably low effort and high enough customizability.

Foundations

Benchmarks should be simple, portable, scale to different workload sizes, and allow the objective comparisons of competing systems and tools [4]. It should be noted that the diversity and complexity of its applications, developing meaningful and realistic benchmarks for XML is a truly herculean task. Also, XML processing tools fall into many categories, from simple storage services to sophisticated query processors, thus adding to the complexity of developing relevant and *realistic* XML benchmarks.

The two factors above have led to the development of benchmarks that are relatively narrow in scope, focusing on very specific tasks. Moreover, the lack of universally accepted, comprehensive XML benchmarks, resulted in the development of general-purpose synthetic data generators, which allow the user to obtain customized test data with relatively low effort. This is significant, as such tools were not popular until the advent of XML.

XML Microbenchmarks

A micro-benchmark is a narrowly-defined benchmark aimed at testing very specific aspects of a tool and/or system.

The Michigan Benchmark The Michigan Benchmark is an XML Micro-benchmark developed at the University of Michigan [8]. It uses a single synthetic document that does not resemble a typical document from any real world application domain. Instead, it is carefully designed to allow the testing of the following query processing operations: matching attributes by value; selecting elements by name; evaluation of positional predicates; selection of nodes based on predicates over their parent, children, ancestors or descendants; join operations; computing aggregate functions; and processing updates. The authors applied the benchmark to three database systems: two native XML DBMSs, and a commercial ORDBMS.

XML Application Benchmarks

An application benchmark is a comprehensive set of tasks that approximates the workload of a typical application in the respective domain. Four important

XML application benchmarks are XMach-1 [2], XMark [9], XBench [10], and TPOX [7].

XMach-1 XMach-1 (*X*ML Data *Ma*nagement Ben*ch*-mark) is a multi-user benchmark developed at the University of Leipzig, Germany [2]. Unlike most existing XML benchmarks that are designed to test the query processors of database management systems, XMach-1 is designed to test database management systems which include a query processor as well as the other key components. In terms of measurement, XMach-1 evaluates systems based on throughput performance (XML queries per second) instead of response time for user-given XML queries.

XMach-1 considers a system architecture to support web applications, which consists of four main components, namely, XML database, application servers, loaders and browser clients. In the XML database, there are multiple schemas, and there are between 2 and 100 documents per schema. Each document is generated using 10,000 most frequent English words, and occupies between 2 and 100 KB of storage. The workload contains eight queries and three update operations. Some evaluation results can be found in [6].

XMark XMark is the result of the XML benchmark project, led by a team at CWI [9]. It models an Internet auctioning application. The workload consists of a large database, in a single XML document, containing: (i) items for auction in geographically dispersed areas; (ii) bidders and their interests; and (iii) detailed information about existing auctions, which can be open or closed. XMark's workload includes 20 queries that cover the following broad kinds of operations: simple node selections; document queries for which order information is relevant; navigational queries; and computing aggregate functions.

The XMark data generator employs several kinds of probability distributions and uses several real data values (e.g., names of countries and people) to produce realistic data; also, the textual content uses real words of the English language. XMark is by far the most widely used XML benchmark at the time of writing.

XBench XBench is a benchmark suite developed at the University of Waterloo [10]. XBench defines application benchmarks categorized according to two criteria: single-document versus multi-document and *data-centric* versus *text-centric* domains. The latter criterion distinguishes data management and exchange scenarios from content management applications (e.g., electronic publishing). Document collections in XBench range in size from a few kilobytes to several gigabytes, and its workload consists of bulk-loading as well as various query and text-based search operations. Results of an evaluation of four different systems, comprising both native XML stores as well as relational-based stores, are provided in [10].

TPoX TPoX (Transaction Processing over XML) is a comprehensive application benchmark developed jointly by IBM and Intel [7]. TPoX simulates a financial application domain (security trading) and is based on the industry-standard XML Schema specification FIXML [3]. The testing environment in TPoX covers several aspects of XML management inside DBMS, including the use of XQuery, SQL/XML, updates, and concurrent access to the data. The authors report on an experimental evaluation of the IBM DB2 product for storing and processing XML data [7].

Synthetic Data Generators

Synthetic data have other applications besides benchmarking, such as testing specific components of a complex system or application. In this setting, an important requirement for a data generator, besides generating *realistic* data (i.e., synthetic data whose characteristics match those of typical real data in the application), is the the ability of easily customizing the test data (e.g., its structure).

Declarative synthetic data generators, on the other hand, are tools that produce synthetic data according to specifications that describe *what* data to generate, as opposed to *how* to generate such data, thus facilitating the generation of synthetic data. Declarative data generators are intended for easing the burden in obtaining test data, unlike the data generators of standardized benchmarks, which have the characteristics of the data they produce embedded in their source code.

Declarative data generators rely on formalisms providing higher levels of abstraction than programming languages, such as conceptual schema languages annotated with probabilistic information (for describing the characteristics of the intended data). Such probabilistic information are needed because schema languages specify only what content is allowed in valid document instances. A realistic data generator must allow the specification of the characteristics of *typical* documents as well. For example, while a schema formalism will specify that a book element may contain between 1 and 10 authors, a realistic data generator will allow one to define a probability distribution for the number of authors in the test data. Another desirable feature of realistic synthetic data is that it satisfies integrity and referential constraints. For instance, an XML document describing a book review should refer to an existing book in the test data.

Two examples of declarative XML data generators are the IBM XML Generator [3], whose data specifications are based on Document Type Definitions, and ToXgene [1], which relies on XML Schema specifications. Both tools allow the specification of skewed probability distributions for elements, attributes, and textual nodes. ToXgene, being based on XML Schema, supports different data types, as well as key and referential constraints. ToXgene also offers a simple declarative query language that allows one to model relatively complex dependencies among elements and attributes involving arithmetic and string manipulation operations. For instance, it allows one to model that the total price of an invoice should be the sum of the individual prices of the items in that invoice multiplied by the appropriate tax rates. Finally, ToXgene offers support for generating recursive XML documents.

Key Applications

Meaningful benchmarks are essential for the development of new technologies, as they allow developers to assess progress and understand intrinsic limitations of their tools. Applications include functionality testing, performance evaluation, and system comparisons.

Recommended Reading

1. Barbosa D. and Mendelzon A.O. Declarative generation of synthetic XML data. Softw. Pract. Exper. 36(10):1051–1079, 2006.
2. Böhme T. and Rahm E. XMach-1: a benchmark for XML data management. In Proc. German Database Conference. Springer, Berlin, 2001, pp. 264–273; Multi-user evaluation of XML data Management Systems with XMach-1. LNCS, Vol. 2590, 2003, pp. 148–159.
3. Financial Information Exchange Protocol. FIXML 4.4 Schema Version Guide. Available at: http://www.fixprotocol.org.
4. Gray J. (ed.). The Benchmark Handbook for Database and Transaction Systems (2nd edn.). Morgan Kaufmann, San Francisco, CA, USA, 1993, ISBN 1-55860-292-5.
5. IBM XML Generator. Available at: http://www.alphaworks.ibm.com/tech/xmlgenerator, 2007.
6. Lu H., Yu J.X., Wang G., Zheng S., Jiang H., Yu G., and Zhou A. What makes the differences: benchmarking XML database implementations. ACM Trans. Internet Technol., 5(1):154–194, 2005.
7. Nicola M., Kogan I., and Schiefer B. An XML transaction processing benchmark. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 937–948.
8. Runapongsa K., Patel J.M., Jagadish H.V., Chen Y., and Al-Khalifa S. The Michigan benchmark: towards XML query performance diagnostics. Inf. Syst., 31(2):73–97, 2006.
9. Schmidt A., Waas F., Kersten M.L., Carey M.J., Manolescu I., Busse R. XMark: a benchmark for XML data management. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 974–985.
10. Yao B.B., Özsu M.T., and Khandelwal N. XBench Benchmark and Performance Testing of XML DBMSs. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 621–633.

XML Cardinality Estimation

► XML Selectivity Estimation

XML Compression

Jayant R. Haritsa[1], Dan Suciu[2]
[1]Indian Institute of Science, Bangalore, India
[2]University of Washington, Seattle, WA, USA

Definition

XML is an extremely verbose data format, with a high degree of redundant information, due to the same tags being repeated over and over for multiple data items, and due to both tags and data values being represented as strings. Viewed in relational database terms, XML stores the "schema" with each and every "record" in the repository. The size increase incurred by publishing data in XML format is estimated to be as much as 400% [14], making it a prime target for compression. While standard general-purpose compressors, such as `zip`, `gzip` or `bzip`, typically compress XML data reasonably well, specialized XML compressors have been developed over the last decade that exploit the specific structural aspects of XML data. These new techniques fall into two classes: (i) *Compression-oriented*, where the goal is to maximize the compression ratio of the data, typically up to a factor of two better than the general-purpose compressors; and (ii) *Query-oriented*, where the goal is to integrate the compression strategy with an XPath query processor such that queries can be processed directly on the compressed data, selectively decompressing only the data relevant to the query result.

Historical Background

Research into XML compression was initiated with Liefke and Suciu's development in 2000 of a compressor called `XMill` [6]. It is based on three principles: separating the structure from the content of the XML document, bucketing the content based on their tags, and compressing the individual buckets separately. `XMill` is a compression-oriented technique, focusing solely on achieving high compression ratios and fast compression/decompression, ignoring the query processing aspects. Other compression-oriented schemes that appeared around the same time include Millau [4], designed for efficient encoding and streaming of XML structures; and XMLPPM, which implements an extended SAX parser for online processing of documents [2]. There are also several commercial offerings that have been featured on the Internet (e.g., [11,13,15]).

Subsequently, the focus shifted to the development of query-oriented techniques intended to support query processing directly on the compressed data. This stream of research began with Tolani and Haritsa [9] presenting in 2002 a system called XGrind, where compression is carried out at the granularity of individual element/attribute values using a simple context-free compression scheme – tags are encoded by integers while textual content is compressed using non-adaptive Huffman (or Arithmetic) coding. XGrind consciously maintains a *homomorphic encoding*, that is, the compressed document is still in XML format, the intention being that all existing XML-related tools (such as parsers, indexes, schema-checkers, etc.) could continue to be used on the compressed document.

In 2003, Min et al. [7] proposed a compressor called XPRESS that extended the homomorphic compression approach of XGrind to include effective evaluation of query path expressions and range queries on numerical attributes. Their scheme uses a reverse arithmetic path-encoding that encodes each path as an interval of real numbers between 0 and 1. The following year produced the XQueC system [1] from Arion et al., which supports cost-based tradeoffs between compact storage and efficient processing.

An excellent survey of the state-of-the-art in XML compressors is available in [1].

Foundations

The basic principles for compressing XML documents are the following:

Separate structure from data: The *structure* consists of XML tags and attributes, organized as a tree. The *data* consists of a sequence of items (strings) representing element text contents and attribute values. The structure and the data are compressed separately.

Group data items with related meaning: Data items are logically or physically grouped into *containers*, and each container is compressed separately. By exploiting similarities between the values in a container, the compression improves substantially. Typically, data items are grouped based on the element type, but some systems (e.g., [1]) chose more elaborate grouping criteria.

Apply specialized compressors to containers: Some data items are plain-text, while others are numbers, dates, etc., and for each of these different domains, the system uses a specialized compressor.

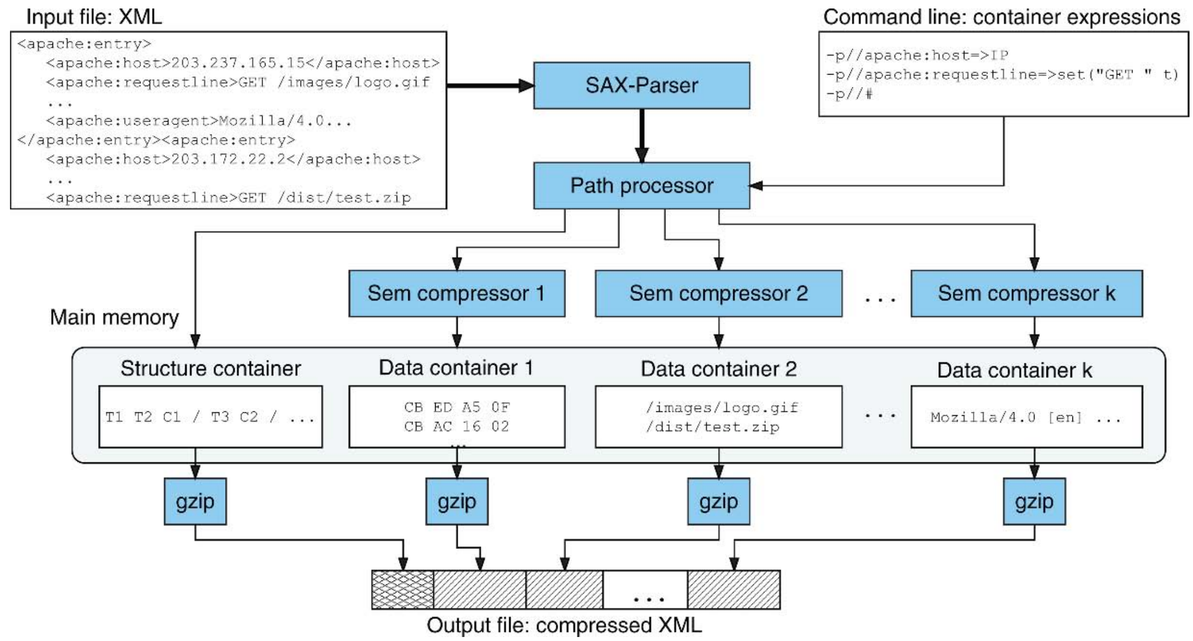
Compression-Oriented XML Compressors

The architecture of the `XMill` compressor [6], which is typical of several XML compressors, is depicted in Fig. 1. The XML file is parsed by a SAX parser that sends tokens to the *path processor*. The purpose of the path processor is to separate the structure from the data, and to further separate the data items according to their semantics.

Next, the structure container and all data containers are compressed with `gzip`, then written to disk. Optionally, the data items in certain containers may be compressed with a user-defined semantic compressor. For example, numerical values or IP addresses can be binary encoded, dates can be represented using specialized data structures, etc.

By default, `XMill` groups data items based on their innermost element type. Users, however, can override this, by providing *container expressions* on the command line. The path processor uses these expressions to determine in which container to store each data item. Path expressions also determine which semantic compressor to apply (if any).

The amount of main memory holding all containers is fixed. When the limit is exhausted all containers are `gzip`-ed, written to disk, as one logical block, then the compression resumes. In effect, this partitions the



```
<apache:entry>
   <apache:host>203.237.165.15</apache:host>
   <apache:requestline>GET /images/logo.gif
   ...
   <apache:useragent>Mozilla/4.0...
</apache:entry><apache:entry>
   <apache:host>203.172.22.2</apache:host>
   ...
   <apache:requestline>GET /dist/test.zip
```

```
-p//apache:host=>IP
-p//apache:requestline=>set("GET " t)
-p//#
```

XML Compression. Figure 1. Architecture of the XMill compressor [4].

input XML file into logical blocks that are compressed independently.

The decompressor, XDemill, is similar, but proceeds in reverse. It reads one block at a time in main memory, decompresses every container, then merges the XML tags with the data values to produce the XML output.

Example. To illustrate the working of XMill, consider the following snippet of Web-server log data, where each entry represents one HTTP request:

```
<apache:entry>
   <apache:host> 202.239.238.16 </host>
   <apache:requestLine>GET / HTTP/1.0
   </apache:requestLine>
   <apache:contentType> text/html
   </apache:contentType>
   <apache:statusCode> 200
   </apache:statusCode>
   <apache:date> 1997/10/01-00:00:02
   </apache:date>
   <apache:byteCount> 4478
   </apache:byteCount>
   <apache:referer>
     http://www.so-net.jp/
   </apache:referer>
   <apache:userAgent> Mozilla/3.0 [ja]
   </apache:userAgent>
</apache:entry>
```

After the document is parsed, the path processor separates it into the *structure* and the *content*, and further separates the content into different containers. The structure is obtained by removing all text values and attribute values and replacing them with their container number. Start-tags are dictionary-encoded, i.e., assigned an integer value, while all end-tags are replaced by the same, unique token. For illustration purposes, start-tags are denoted with T1, T2,..., the unique end-tag with /, and container numbers with C1, C2,.... In this example the structure of the entry element is:

```
   T1 T2 C1 / T3 C2 / T4 C3 / T5 C4 /
T8 C7 / T8 C7 / T11 C10 / T12 C11 / /
```

Here T1 = apache:entry, T2 = apache:host, and so on, while / represents any end tag. Internally, each token is encoded as an integer: tags are positive, container numbers are negative, and \ is 0. Numbers between (− 64, 63) take one byte, while numbers outside this range take two or four bytes; the example string is overall coded in 26 bytes. The structure is compressed using gzip, which is based on Ziv-Lempel's LZ77 algorithm [10]. This results in excellent compression, because LZ77 exploits very well the frequent repetitions in the structure container: the compressed structure usually amounts to only 1–3% of the compressed file.

Next, data items are partitioned into containers, then compressed. Each container is associated with an XPath expression that defines which data items are stored in that container, and an optional semantic compressor for the items in that container. By default there is a container for each tag `tag` occurring in the XML file, the associated XPath expression is `//tag`, and there is no associated semantic compressor. Users may override this on the command line. For example:

```
xmill -p //shipping/address -p
//billing/address file.xml
```

creates two separate containers for `address` elements: one for those occurring under `shipping`, and one for those occurring under `billing`. If there exist `address` elements occurring under elements other than `shipping` or `billing`, then their content is stored in a third container. Note that the container expressions only need to be specified to the compressor, not to the decompressor. Overriding the default grouping of data items usually results in only modest improvements in the compression ratio. Much better improvements are achieved, however, with semantic compressors.

Query-Oriented XML Compressors

XGrind. The technique described in [9] is intended to simultaneously provide efficient query-processing performance and reasonable compression ratios. Basic requirements to achieve the former objective are (i) fine-grained compression at the element/attribute granularity of query predicates, and (ii) context-free compression assigning codes to data items independent of their location in the document. Algorithms such as LZ77 are not context-free, and therefore XGrind uses non-adaptive Huffman (or Arithmetic) coding, in which two passes are made over the XML document – the first to collect the statistics and the second to do the actual encoding. A separate character-frequency distribution table is used for each element and non-enumerated attribute, resulting in fine-grained characterization. The DTD is used to identify enumerated-type attributes and their values are encoded using a simple binary encoding scheme, while the compression of XML tags is similar to that of `XMill`. With this scheme, exact-match and prefix-match queries can be completely carried out directly on the compressed document, while range or partial-match queries only require on-the-fly decompression of the element/attribute values that feature in the query predicates.

A distinguishing feature of the `XGrind` compressor is that it ensures *homomorphic* compression – that is, its output, like its input, is semi-structured in nature. In fact, the compressed XML document can be viewed as the original XML document with its tags and element/attribute values replaced by their corresponding encodings. The advantage of doing so is that the variety of efficient techniques available for parsing/querying XML documents can also be used to process the compressed document. Second, indexes can be built on the compressed document similar to those built on regular XML documents. Third, updates to the XML document can be directly executed on the compressed version. Finally, a compressed document can be directly checked for validity against the compressed version of its DTD.

As a specific example of the utility of homomorphic compression, consider repositories of genomic data (e.g., [12]), which allow registered users to upload new genetic information to their archives. With homomorphic compression, such information could be compressed by the user, then uploaded, checked for validity, and integrated with the existing archives, all operations taking place completely in the compressed domain.

To illustrate the working of XGrind, consider the XML student document fragment along with its DTD shown in Figs. 2 and 3. An abstract view of its XGrind compressed version is shown in Fig. 4, in which *nahuff*(s) denotes the output of the Huffman-Compressor for an input data value `s`, while *enum*(s) denotes the output of the Enum-Encoder for an input data value `s`, which is an enumerated attribute. As is evident from Fig. 4, the compressed document output in the second pass is semi-structured in nature, and maintains the property of validity with respect to the compressed DTD.

The compressed-domain query processing engine consists of a lexical analyzer that emits tokens for encoded tags, attributes, and data values, and a parser built on top of this lexical analyzer does the matching and dumping of the matched tree fragments. The parser maintains information about its current path location in the XML document and the contents of the set of XML nodes that it is currently processing. For exact-match or prefix-match queries, the query path and the query predicate are converted to the compressed-domain equivalents. During parsing of the compressed XML document, when the parser detects that the current path matches the query path, and that the compressed data value matches the compressed

```
<!- student.xml -->
<STUDENT rollno = "604100418">
    <NAME>Edgar Codd</NAME>
    <YEAR>2000</YEAR>
    <PROG>Master of Science</PROG>
    <DEPT name = "Computer_Science">
</STUDENT>
```

XML Compression. Figure 2. Fragment of student database.

```
<!- DTD for the Student database -->
<!ELEMENT STUDENT (NAME, YEAR, PROG, DEPT)>
<!ATTLIST STUDENT rollno CDATA #REQUIRED>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT YEAR (#PCDATA)>
<!ELEMENT PROG (#PCDATA)>
<!ELEMENT DEPT EMPTY>
<!ATTLIST DEPT name (Computer_Science
    | Electrical_Engineering
          .
          :
    | Physics | Chemistry)
>
```

XML Compression. Figure 3. DTD of student database.

```
T0 A0 nahuff(604100418)
    T1 nahuff(Edgar Codd) /
    T2 nahuff(2000) /
    T3 nahuff(Master of Science) /
    T4A1 enum(Computer_Science) /
/
```

XML Compression. Figure 4. Abstract view of compressed XGrind database.

query predicate, it outputs the matched XML fragment. An interesting side-effect is that the matching is more efficient in the compressed domain as compared to the original domain, since the number of bytes to be processed have considerably decreased.

XPRESS. While maintaining the homomorphic feature of XGrind, XPRESS [7] significantly extends its scope by supporting both path expressions and range queries (on numeric element types) directly on the compressed data. Here, instead of representing the tag of each element with a single identifier, the element label path is encoded as a distinct interval in [0.0, 1.0). The specific process, called reverse arithmetic encoding, is as follows: First, the entire interval [0.0, 1.0) is partitioned into disjoint sub-intervals, one for each distinct element. The size of the interval is proportional to the normalized frequency of the element in the data. In the second step, these element intervals are reduced by encoding the path leading to this element in a depth-first tree traversal from the root. An example from [7] is shown in Table 1, where the element intervals are computed from the first partitioning, and the corresponding reduced intervals by following the path labels from the root.

XML Compression. Table 1. XPRESS interval scheme

Element tag	Path label	Element interval	Path interval
book	*book*	[0.0, 0.1)	[0.0, 0.1)
section	*book.section*	[0.3, 0.6)	[0.3, 0.33)
subsection	*book.section. subsection*	[0.6, 0.9)	[0.69, 0.699)

For each node in the tree, the associated interval is incrementally computed from the parent node. The intervals generated by reverse arithmetic encoding guarantee that *"If a path P is represented by the interval I, then all intervals for suffixes of P contain I."* Therefore, the interval for *subsection* which is [0.6, 0.9) contains the interval [0.69, 0.699) for the path *book.section.subsection*. Another feature of XPRESS is that it infers the data types of elements and for those that turn out to be numbers over large domains, the values are compressed by first converting them to binary and then using differential encoding, instead of the default string encoding. Recently, XPRESS has been extended in [8] to handle updates such as insertions or deletions of XML fragments.

Finally, an index-based compression approach that improves on both the compression ratio and the querry processing speed has been recently proposed in [3].

Key Applications

Data archiving, data exchange, query processing.

Cross-references

- ▶ Data Compression in Sensor Networks
- ▶ Indexing Compressed Text
- ▶ Lossless Data Compression
- ▶ Managing Compressed Structured Text
- ▶ Text Compression
- ▶ XML
- ▶ XPath/XQuery

Recommended Reading

1. Arion A., Bonifati A., Manolescu I., and Pugliese A. XQueC: a query-conscious compressed XML database. ACM Trans. Internet Techn., 7(2):1–35, 2007.
2. Cheney J. Compressing XML with multiplexed hierarchical PPM models. In Proc. Data Compression Conference, 2001, pp. 163–172.
3. Ferragina P., Luccio F., Manzini G., and Muthukrishnan M. Compressing and Searching XML Data Via Two Zips. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 751–760.
4. Girardot M. and Sundaresan N. Millau: an encoding format for efficient representation and exchange of XML over the Web. In Proc. 9th Int. World Wide Web Conference, 2000.
5. Liefke H. and Suciu D. An extensible compressor for XML data. ACM SIGMOD Rec., 29(1):57–62, 2000.
6. Liefke H. and Suciu D. XMill: an efficent compressor for XML data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 153–164.
7. Min J.K., Park M., and Chung C. XPRESS: a queriable compression for XML data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 122–133.
8. Min J.K., Park M., and Chung C. XPRESS: a compressor for effective archiving, retrieval, and update of XML documents. ACM Trans. Internet Techn., 6(3):223–258, 2006.
9. Tolani P. and Haritsa J. XGRIND: a query-friendly XML compressor. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 225–235.
10. Ziv J. and Lempel A. A universal algorithm for sequential data compression. IEEE Trans. Inf. Theory, 23(3):337–343, 1977.
11. www.dbxml.com
12. www.ebi.ac.uk
13. www.ictcompress.com
14. www.ictcompress.com/xml.html
15. www.xmlzip.com

XML Data Dependencies

► XML Integrity Constraints

XML Data Integration

► XML Information Integration

XML Database

► XML Storage

XML Database Design

► Semi-structured database design

XML Database System

► XQuery Processors

XML Document

Michael Rys
Microsoft Corporation, Sammamish, WA, USA

Synonyms

XML document

Definition

An XML document is a text document that follows the W3C XML recommendation [3].

Key Points

An XML document is considered well-formed, if it satisfies the XML 1.0 well-formedness constraints of having exactly one top-level XML element, balanced begin- and end-tags, and all the other rules outlined in [3].

It is considered valid, if it validates against a schema language such as the XML DTDs [3] or XML Schema [1].

An XML document is represented as an XML document information item in the XML Information Set [2] and an XML document node in the XPath and XQuery data model [4].

Cross-references

► XML
► XML Attribute
► XML Element

Recommended Reading

1. Schema Part 0: Primer, latest edition available online at: http://www.w3.org/TR/xmlschema-0/.
2. XML 1.0 Information Set, latest edition available online at: http://www.w3.org/TR/xml-infoset.

3. XML 1.0 Recommendation, latest edition available online at: http://www.w3.org/TR/xml.
4. XQuery 1.0 and XPath 2.0 Data Model (XDM), latest edition available online at: http://www.w3.org/TR/xpath-datamodel/.

XML Element

MICHAEL RYS
Microsoft Corporation, Sammamish, WA, USA

Synonyms

XML element

Definition

An XML element is a markup element in the W3C XML recommendation [3] that consists of a beginning markup tag and an end tag with optional content and optional attributes.

Key Points

An XML element is a markup element in the W3C XML recommendation [3] that consists of a beginning markup tag and an end tag.

It has to have balanced begin- and end-tags as in this example

<element>content</element>.

XML elements can be nested as in

<e1>some optional text <e2>nested element </e2> some optional text</e1>

They can have XML attributes as in

<element attribute="value">content</element>

And they can be empty, meaning have no content as in

<element/>.

The structure of elements is free within the grammatical rules of [3], but elements can be constrained and typed by schema languages such as XML DTDs [3] or XML Schema [1].

An XML element is represented as an XML element information item in the XML Information Set [2] and an XML element node in the XPath and XQuery data model [4].

Cross-references

- ▶ XML
- ▶ XML Attribute
- ▶ XML Document

Recommended Reading

1. Schema Part 0: Primer, latest edition available online at: http://www.w3.org/TR/xmlschema-0/.
2. XML 1.0 Information Set, latest edition available online at: http://www.w3.org/TR/xml-infoset.
3. XML 1.0 Recommendation, latest edition available online at: http://www.w3.org/TR/xml.
4. XQuery 1.0 and XPath 2.0 Data Model (XDM), latest edition available online at: http://www.w3.org/TR/xpath-datamodel/.

XML Enterprise Information Integration

▶ XML Information Integration

XML Export

▶ XML Publishing

XML Filtering

▶ XML Publish/Subscribe

XML Indexing

XIN LUNA DONG, DIVESH SRIVASTAVA
AT&T Labs–Research, Florham Park, NJ, USA

Definition

XML employs an ordered, tree-structured model for representing data. Queries in XML languages like XQuery employ twig queries to match relevant portions of data in an XML database. An XML Index is a data structure that is used to efficiently look up all matches of a fragment of the twig query, where some of the twig query fragment nodes may have been mapped to specific nodes in the XML database.

Historical Background

XML path indexing is related to the problem of join indexing in relational database systems [15] and path indexing in object-oriented database systems (see, e.g., [1,9]). These index structures assume that the schema is homogeneous and known; these assumptions do not hold in general for XML data. The DataGuide [7] was the first path index designed specifically for XML data, where the schema may be heterogeneous and may not even be known.

Foundations

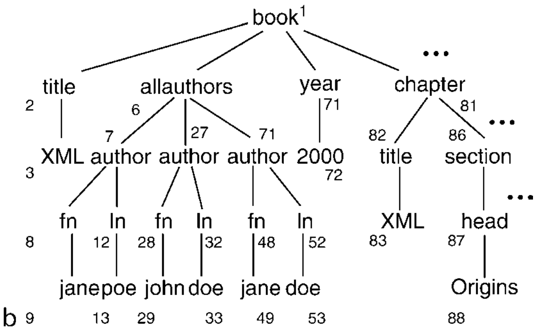
Notation

An XML document *d* is a rooted, ordered, node-labeled tree, where (i) each node corresponds to an XML element, an XML attribute, or a value; and (ii) each edge corresponds to an element-subelement, element-attribute, element-value, or an attribute-value relationship. Non-leaf nodes in *d* correspond to XML elements and attributes, and are labeled by the element tags or attribute names, while leaf nodes in *d* correspond to values. For the example XML document of Fig. 1a, its tree representation is shown in Fig. 1b. Each node is associated with a unique number referred to as its Id, as depicted in Fig. 1b. An XML database *D* is a set of XML documents.

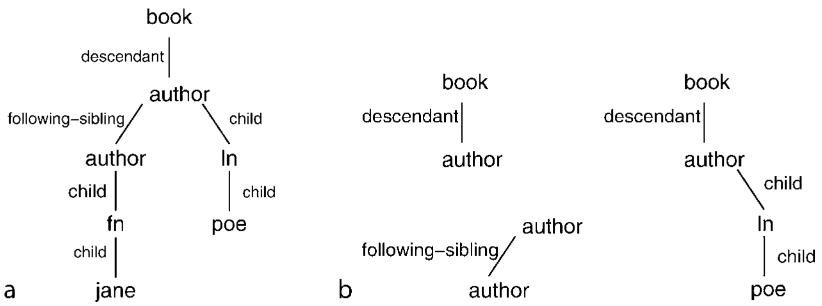
Queries in XML languages like XPath and XQuery make fundamental use of twig queries to match relevant portions of data in an XML database. A twig query *Q* is a node- and edge-labeled tree, where (i) nodes are labeled by element tags, attribute names, or values; (ii) edges are labeled by an XPath axis step, e.g., `child`, `descendant`, or `following-sibling`. For example, the twig query in Fig. 2a corresponds to the path expression `/book[./descendant:: author[./following-sibling :: author [fn =‘‘jane’’]][ln =‘‘poe’’]]`.

Given an XML database *D*, and a twig query *Q*, a match of *Q* in *D* is identified by a mapping from nodes in *Q* to nodes in *D*, such that: (i) *Q*'s node labels (i.e., element tags, attribute-names and values) are preserved under the mapping; and (ii) *Q*'s edge labels (i.e., XPath axis steps) are satisfied by the corresponding pair of nodes in *D* under the mapping. For example, the twig query of Fig. 2a matches a root book element that has a descendant author element that (i) has a child `ln` element with value `poe`; and (ii) has a following sibling `author` element that has a child `fn`

```
<book>
  <title> XML </title>
  <allauthors>
     <author>
        <fn>jane</fn>
        <ln>poe</ln>
     </author>
     <author>
        <fn>john</fn>
        <ln>doe</ln>
     </author>
     <author>
        <fn>jane</fn>
        <ln>doe</ln>
     </author>
  </allauthors>
  <year> 2000 </year>
  <chapter>
     <title> XML </title>
     <section>
        <head> Origins </head>
        ...
     </section>  ...
  </chapter>  ...
</book>
```
a



XML Indexing. Figure 1. (a) An XML database fragment. (b) XML tree (Id numbering).



XML Indexing. Figure 2. (a) Twig query. (b) Twig query fragments.

element with value jane. Thus, the book element in Fig. 1b is a match of the twig query in Fig. 2a.

An *XML Index* I is a data structure that is used to efficiently look up all matches of a *fragment* of the twig query Q, where some of the twig query fragment nodes may have been mapped to specific nodes in the XML database. Some fragments of the twig query of Fig. 2a are shown in Fig. 2b and include the edges book/descendant::author and author/following-sibling::author, and the path book/descendant::author[ln =``poe'']. The matches returned by XML Index lookups on different fragments of a twig query Q can be "stitched together" using query processing algorithms to compute matches to Q.

The following sections describe various techniques that have been proposed in the literature to index node, edge, path and twig fragments of twig queries.

Node Indexes

When the fragment of a twig query Q that needs to be looked up in the index is a single node labeled by an element tag, an attribute name, or a value, a classical *inverted index* is adequate. This index is constructed by associating each element tag (or attribute name, value) with the list of all the node Ids in the XML database with that element tag (attribute name, value, respectively). For example, given the data of Fig. 1b, the list associated with the element tag author would be [7,27,47], and the list associated with the value jane would be [9,49].

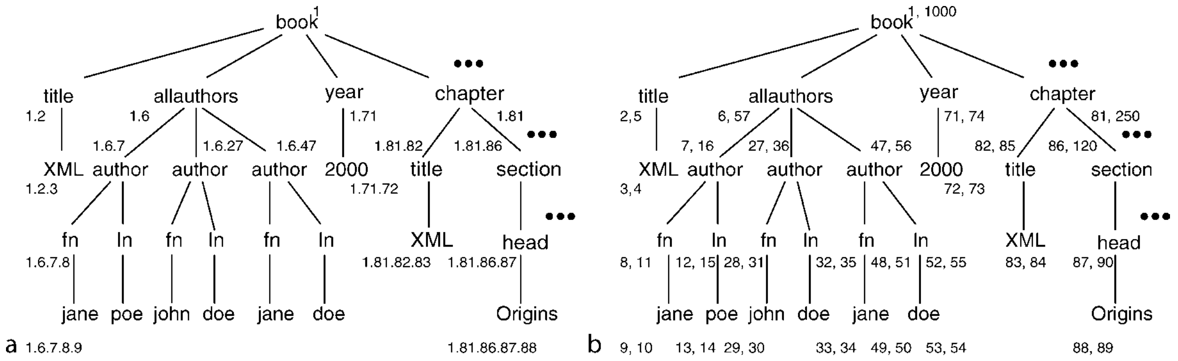
Positional Numberings, Edge Indexes

Now consider the case when the fragment of a twig query Q that needs to be looked up in the index is an edge labeled by an XPath axis step. For example, find all author nodes that are descendants of the book node with Id = 1. As another example, find all author nodes that are following-siblings of the author node with Id = 7.

Using Node Ids A simple solution is to use an inverted index that associates each (node_1 label, node_2 label, XPath axis) triple to the list of all pairs of XML database node Ids that satisfy the specified node labels and axis relationship. For example, given the data of Fig. 1b, the list associated with the triple (book, author, descendant) would be [(1,7),(1,27),(1,47)], and the list associated with the triple (author, author, following-sibling) would be [(7,27),(7,47),(27,47)].

In general, this inverted index could be much larger than the number of nodes in the original XML database, especially for XPath axes such as descendant, following-sibling and following. To overcome this limitation, more sophisticated approaches are required. A popular approach has been to (i) use a positional numbering system to identify nodes in an XML database; and (ii) demonstrate that each XPath axis step corresponds to a predicate on the positional numbers of the corresponding nodes. Two such approaches, using Dewey numbering and using Interval numbering, are described next.

Using Dewey Numbering An elegant solution for edge indexing is to associate each XML node *n* with its DeweyId, proposed by [14], and obtained as follows: (i) associate each node with a numeric Id ensuring that sibling nodes are given increasing numbers in a left-to-right order; and (ii) the DeweyId of a node is obtained by concatenating the Ids of all nodes along



XML Indexing. Figure 3. (a) Dewey numbering. (b) Interval numbering.

the path from the root node of n's XML document to n itself (The similarity with the Dewey Decimal System of library classification is the reason for its name). Figure 3a shows the DeweyIds of some nodes in the XML tree, using the numeric Ids associated with those nodes in Fig. 1b. For example, the fn node with Id = 8 has DeweyId = 1.6.7.8.

DeweyIds can be used to easily find matches to various XPath axis steps. In particular, node n_2 is a descendant of node n_1 if and only if (i) n_1.DeweyId is a prefix of n_2.DeweyId. For example, in Fig. 3a, the jane node with DeweyId = 1.6.7.8.9 is a descendant of the author node with DeweyId = 1.6.7. Similarly, node n_2 is a child of node n_1 if and only if (i) n_2 is a descendant of n_1; and (ii) n_2's DeweyId extends n_1's DeweyId by one Id. By maintaining DeweyIds of nodes in classical trie data structures, various XPath axis lookups can be done efficiently.

The main limitation of DeweyIds is that their size depends on the depth of the XML tree, and can get quite large. This limitation is overcome by using Interval numbering, described next.

Using Interval Numbering The position of an XML node n is represented as a 3-tuple: (LeftPos, RightPos, PLeftPos), where (i) numbers are generated in an increasing order by visiting each tree node twice in a left-to-right, depth-first traversal; n.LeftPos is the number generated before visiting any node in n's subtree and n.RightPos is the number generated after visiting every node in n's subtree; and (ii) n.PLeftPos is the LeftPos of n's parent node (0 if n is the root node of DocId). Figure 3b depicts the LeftPos and RightPos numbers of each node in the XML document.

It can be seen that each XPath axis step between a pair of XML database nodes can be tested using a conjunction of equality and inequality predicates on the components of the 3-tuple. In particular, node n_2 is a descendant of node n_1 if and only if: (i) n_1.LeftPos $<$ n_2.LeftPos; and (ii) n_1.RightPos $>$ n_2.RightPos. An element n_2 is a child of an element n_1 if and only if n_1.LeftPos = n_2.PLeftPos. An element n_2 is a following-sibling of an element n_1 if and only if: (i) n_1.RightPos $<$ n_2.LeftPos; and (ii) n_1.PLeftPos = n_2.PLeftPos. For example, in Fig. 3b, the jane node with interval number (9,10,8) is a descendant of the author node with interval number (7,16,6).

Thus, the set of 3-tuples corresponding to the interval numbering of nodes of an XML database can be indexed using a 3-dimensional spatial index such as an R-tree, and the different XPath axis steps correspond to different regions within the 3-dimensional space. Variations of this approach have been considered in, e.g., [10,2,8].

Note that for both Dewey numbering and Interval numbering, one would need to leave gaps between numbers to allow for insertions of new nodes in the XML database [5].

Path Indexes

When the fragment of a twig query Q that needs to be looked up in the index is a subpath of a root-to-leaf path in Q, where some (possibly none) of the nodes have been mapped to specific nodes in the XML database, XML path indexes are very useful. The works in the literature have primarily focused on the case where each edge in the subpath of Q is labeled by child, i.e., all matches are subpaths of root-to-leaf paths in an XML document. For example, a path index can be used to efficiently look up all matches to the path fragment author[ln=``poe''] of the twig query depicted in Fig. 2a.

A framework by Chen et al. [3] is described next, which covers most existing XML path index structures, and solves the BoundIndex problem.

Problem BoundIndex: Given an XML database D, a subpath query P with k node labels and each edge labeled by child, and a specific database node id n, return all k-tuples $(n_1,\ldots,n_k)$ that identify matches of P in D, rooted at node n.

The framework of [3] requires each node in an XML document to be associated with a unique numeric identifier; this could be, e.g., the Id of the node in Fig. 1b. To create a path index, [3] conceptually separates a path in an XML document into two parts: (i) a *schema path*, which consists solely of schema components, i.e., element tags and attribute names; and (ii) a *leaf value* as a string if the path reaches a leaf. Schema paths can be dictionary-encoded using special characters (whose lengths depend on the dictionary size) as designators for the schema components. Most of the works in the literature have focused on indexing XML schema paths (see, e.g., [7,12,4]). Notable exceptions that also consider indexing data values at the leaves of paths include [6,17,3].

In order to solve the BoundIndex problem, one needs to explicitly represent paths that are arbitrary subpaths of the root-to-leaf paths, and associate each such path with the node at which the subpath is rooted. Such a relational representation of *all* the paths in an XML database is (HeadId, SchemaPath, LeafValue, IdList), where HeadId is the id of the start of the path, and IdList is the list of all node identifiers along the schema path, except for the HeadId. As an example, a fragment of the 4-ary relational representation of the data tree of Fig. 1b is given in Table 1; element tags have been encoded using boldface characters as designators, based on the first character of the tag, except for allauthors which uses **U** as its designator.

XML Indexing. Table 1. The 4-ary relation for path indexes

HeadId	SchemaPath	LeafValue	IdList
1	**B**	null	[]
1	**BT**	null	[2]
1	**BT**	XML	[2]
1	**BU**	null	[6]
1	**BUA**	null	[6,7]
1	**BUAF**	null	[6,7,8]
1	**BUAF**	jane	[6,7,8]
1	**BUAL**	null	[6,7,12]
1	**BUAL**	poe	[6,7,12]
	. . .		
6	**U**	null	[]
6	**UA**	null	[7]
6	**UAF**	null	[7,8]
6	**UAF**	jane	[7,8]
6	**UAL**	null	[7,12]
6	**UAL**	poe	[7,12]
	. . .		

Given the 4-ary relational representation of XML database D, each index in the family of indexes: (i) stores a subset of all possible SchemaPaths in D; (ii) stores a sublist of IdList; and (iii) indexes a subset of the columns HeadId, SchemaPath, and LeafValue.

Given a query, the index structure probes the indexed columns in (iii) and returns the sublist of IdList stored in the index entries. Many existing indexes fit in this framework, as summarized in Table 2. For example, the DataGuide [7] returns the last Id of the IdList for every root-to-leaf prefix path. Similarly, IndexFabric [6] returns the Id of either the root or the leaf element (first or last Id in IdList), given a root-to-leaf path and the value of the leaf element. Finally, the DATAPATHS index is a regular B$^+$-tree index on the concatenation of HeadId, LeafValue and the reverse of SchemaPath (or the concatenation LeafValue·HeadId·ReverseSchemaPath), where the SchemaPath column stores all subpaths of root-to-leaf paths, and the complete IdList is returned; the DATAPATHS index can solve the BoundIndex problem in one index lookup.

Twig Indexes

ViST [16] and PRIX [13] are techniques that encode XML documents as sequences, and perform subsequence matching to look up all matches to twig queries.

XML Indexing. Table 2. Members of family of path indexes

Index	Subset of SchemaPath	Sublist of IdList	Indexed Columns
Value [11]	paths of length 1	only last Id	SchemaPath, LeafValue
Forward link [11]	paths of length 1	only last Id	HeadId, SchemaPath
DataGuide [7]	root-to-leaf path prefixes	only last Id	SchemaPath
Index Fabric [6]	root-to-leaf paths	only first or last Id	reverse SchemaPath, LeafValue
ROOTPATHS [3]	root-to-leaf path prefixes	full IdList	LeafValue, reverse SchemaPath,
DATAPATHS [3]	all paths	full IdList	LeafValue, HeadId, reverse SchemaPath

Key Applications

XML Indexing is important for efficient XML query processing, both in relational implementations of XML databases and in native XML databases.

Future Directions

It is important to investigate XML Path Indexes for the case of path queries with edge labels other than child, especially when different edges on a query path have different edge labels. Another extension worth investigating is to identify classes of twig queries that admit efficient XML Twig Indexes.

Data Sets

University of Washington XML Repository: http://www.cs.washington.edu/research/xmldatasets/.

Cross-references

- ▶ XML Document
- ▶ XML Tree Pattern, XML Twig Query
- ▶ XPath/XQuery
- ▶ XQuery Processors

Recommended Reading

1. Bertino E. and Kim W. Indexing techniques for queries on nested objects. IEEE Trans. Knowledge and Data Eng., 1 (2):196–214, 1989.
2. Bruno N., Koudas N., and Srivastava D. Holistic twig joins: optimal XML pattern matching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 310–321.
3. Chen Z., Gehrke J., Korn F., Koudas N., Shanmugasundaram J., and Srivastava D. Index structures for matching XML twigs using relational query processors. Data Knowl. Eng., 60 (2):283–302, 2007.
4. Chung C.-W., Min J.-K., and Shim K. APEX: an adaptive path index for XML data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 121–132.
5. Cohen E., Kaplan H., and Milo T. Labeling dynamic XML trees. In Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 2002, pp. 271–281.
6. Cooper B.F., Sample N., Franklin M.J., Hjaltason G.R., and Shadmon M. A fast index for semistructured data. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 341–350.
7. Goldman R. and Widom J. DataGuides: enabling query formulation and optimization in semistructured databases. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 436–445.
8. Grust T. Accelerating XPath location steps. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 109–120.
9. Kemper A. and Moerkotte G. Access support in object bases. ACM SIGMOD Rec., 19(2):364–374, 1990.
10. Kha D.D., Yoshikawa M., and Uemura S. An XML indexing structure with relative region coordinate. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 313–320.
11. McHugh J. and Widom J. Query optimization for XML. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 315–326.
12. Milo T. and Suciu D. Index structures for path expressions. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 277–295.
13. Rao P. and Moon B. PRIX: Indexing and querying XML using Pruffer sequences. In Proc. 20th Int. Conf. on Data Engineering, 2004, p. 288.

14. Tatarinov I., Viglas S., Beyer K., Shanmugasundaram J., Shekita E., and Zhang C. Storing and querying ordered XML using a relational database system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 204–215.
15. Valduriez P. Join indices. ACM Trans. Database Syst., 12 (2):218–246, 1987.
16. Wang H., Park S., Fan W., and Yu P. ViST: a dynamic index method for querying XML data by tree structures. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 110–121.
17. Yoshikawa M., Amagasa T., Shimura T., and Uemura S. XRel: a path-based approach to storage and retrieval of XML documents using relational databases. ACM Trans. Internet Tech., 1(1):110–141, 2001.

XML Information Integration

Alon Halevy
Google Inc., Mountain View, CA, USA

Synonyms

XML data integration; XML enterprise information integration

Definition

Information integration systems offer uniform access to a set of autonomous and heterogeneous data sources. Sources can range from database systems and legacy systems to forms on the Web, web services and flat files. The data in the sources need not be completely structured as in relational databases. The number of sources in a information integration application can range from a handful to thousands. XML information integration systems are ones that based on an XML data model and query language.

Key Points

XML played a significant role in the development of information integration, from the research and from the commercialization perspectives. The emergence of XML fueled the desire for information integration, because it offered a common syntactic format for sharing data. Once this common syntax was in place, organizations could start thinking about sharing data with each other (or even, within the organization) and integrating data from multiple sources. Of course, XML did very little to address the semantic integration issues – sources could still share XML files whose tags were completely meaningless outside the application. However, the fact that XML is semi-structured was an advantage when modeling heterogeneous data that may have different schematic structure.

From the technical perspective, several information integration systems were developed with an XML data model at their core. Developing these systems lead to many advances in XML data management, including the development of query and update languages, languages for schema mapping, algorithms for query containment and answering queries using views, and efficient processing of XML data streaming from external sources. Every aspect of information integration systems had to be reexamined with XML in mind. Interestingly, since the push to commercialize information integration came around the same time as the emergence of XML, many of the innovations mentioned above had to be developed by start-ups on a very short fuse.

Key Applications

Some of the key applications of information integration are:

- Enterprise data management, querying across several enterprise data repositories
- Accessing multiple data sources on the web (and in particular, the deep web)
- Large scientific projects where multiple scientists are independently producing data sets
- Coordination accross mulitple government agencies

Cross-references

▶ Adaptive Query Processing
▶ Information Integration
▶ Model Management

Recommended Reading

1. Abiteboul S., Benjelloun O., and Milo T. The active XML project: an overview. VLDB J., 17(5):1019–1040, 2008.
2. Deshpande A., Ives Z., and Raman V. Adaptive Query Processing. Foundations and Trends in Databases. Now Publishers, 2007. www.nowpublishers.com/dbs.
3. Haas L. Beauty and the Beast: The theory and practice of information integration. In Proc. 11th Int. Conf. on Database Theory, 2007, pp. 28–43.
4. Halevy A.Y., Ashish N., Bitton D., Carey M.J., Draper D., Pollock J., Rosenthal A., and Sikka V. Enterprise information integration: successes, challenges and controversies. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 778–787.

XML Information Retrieval

▶ INitiative for the Evaluation of XML retrieval (INEX)

XML Integrity Constraints

Marcelo Arenas
Pontifical Catholic University of Chile, Santiago, Chile

Synonyms

XML data dependencies

Definition

An XML integrity constraint specifies a semantic restriction over the data stored in an XML document. A number of integrity constraint languages have been proposed for XML, which can be used to enforce different types of semantic restrictions. These proposals, together with some languages for specifying restrictions on the element structure of XML documents (e.g., DTD and XML Schema), are currently used to specify the schema of XML documents.

Historical Background

The problem of defining and manipulating integrity constraints is one of the oldest problems in databases. Soon after the introduction of the relational model by Codd in the '70s, researchers developed several languages for specifying integrity constraints, and studied many fundamental problems for these languages.

In the relational model, a database is viewed as a collection of relations or tables. For instance, a relational database storing information about courses in a university is shown in Fig. 1. This database consists of a time-varying part, the data about courses, and a part considered to be time independent, the *schema* of the relations, which is given by the name of the relations (*CourseInfo* and *CourseTerm*) and the names of the attributes of the relations.

Usually, the information contained in a database satisfies some *semantic* restrictions. For example, in the relation *CourseInfo* shown in Fig. 1, it is expected that only one title is associated to each course number. By providing the schema of a relation, a syntactic constraint is specified (the structure of the relation), but a semantic constraint as the one mentioned above cannot be specified. To overcome this problem, it is necessary to specify separately a set of semantic restrictions. These restrictions are called *integrity constraints*, and they are expressed by using suitable languages.

Although a number of integrity constraints languages were developed for relational databases, functional and inclusion dependencies are the ones used most often. A functional dependency is an expression of the form $X \rightarrow Y$, where X and Y are sets of attributes. A relation satisfies $X \rightarrow Y$ if for every pair of tuples t_1, t_2 in it, if t_1 and t_2 have the same values on X, then they have the same values on Y. An inclusion dependency is an expression of the form $R[X] \subseteq S[Y]$, where R and S are relation names, and X, Y are sets of attributes of the same cardinality. A relational database satisfies $R[X] \subseteq S[Y]$ if for every tuple t_1 in R, there exists a tuple t_2 in S such that the values of t_1 on X are the same as the values of t_2 on Y. For example, relation *CourseTerm* shown in Fig. 1 satisfies functional dependency $\{Number, Section\} \rightarrow Room$, since every section of a course is given in only one room, and relations *CourseInfo* and *CourseTerm* satisfy inclusion dependency $CourseTerm[Number] \subseteq CourseInfo[Number]$, since every course number mentioned in *CourseTerm* is also mentioned in *CourseInfo* (and thus every course given in some term has a name).

Integrity constraint languages have also been developed for more recent data models, such as nested relational and object-oriented databases. Functional and inclusion dependencies are also present in all these models. In fact, two subclasses of functional and inclusion dependencies, namely, keys and foreign keys, are most commonly found in practice. For the case of relational databases, a key dependency is a functional dependency of the form $X \rightarrow U$, where U is the set of attributes of a relation, and a foreign key dependency is formed by an inclusion dependency $R[X] \subseteq S[Y]$ and a key dependency $Y \rightarrow U$, where U is the set of attributes of S. For example, $Number \rightarrow Number, Title$ is a key dependency for relation *CourseInfo*, while $CourseTerm[Number] \subseteq CourseInfo[Number]$ and $Number \rightarrow Number, Title$ is a foreign key dependency. Keys and foreign keys are fundamental to conceptual database design, and are supported by

CourseInfo	Number	Title
	CSC258	Compilers
	CSC434	Data management systems

CourseTerm	Number	Section	Room
	CSC258	1	LP266
	CSC258	2	GB258
	CSC434	1	GB248

XML Integrity Constraints. Figure 1. Example of a relational database.

the SQL standard. They provide a mechanism by which objects can be uniquely identified and referenced, and they have proved useful in update anomaly prevention, query optimization and index design.

Foundations

A number of integrity constraint specifications have been proposed for XML. The hierarchical nature of XML data calls for not only *absolute constraints* that hold on an entire document, such as dependencies found in relational databases, but also *relative constraints* that only hold on sub-documents, beyond what it is encountered in traditional databases. Here, both absolute and relative functional dependencies, keys, inclusion dependencies and foreign keys are considered for XML.

In general, integrity constraints for XML are defined as restrictions on the nodes and data values reachable by following paths in some tree representation of XML documents. For example, Fig. 2 shows an XML document storing information about courses at the University of Toronto (UofT), and Fig. 3 shows a tree representation of this document. Given an XML document D, element types $\tau_1,\ldots,\tau_n$ in D and a symbol ℓ such that ℓ is either an element type in D or an attribute in D or the reserved symbol `text()`, the string $\tau_1,\ldots,\tau_n.\ell$ is a path in D if there exist nodes $u_1,\ldots,u_n$ in D such that (i) each u_i is if type τ_i, (ii) each u_{i+1} is a child of u_i, (iii) if ℓ is an element type, then there exists a node u of type ℓ that is a child of u_n, (iv) if ℓ is an attribute, then ℓ is defined for u_n, and (v) if ℓ = `text()`, then u_n has a child of type PCDATA. For example, `student.taking`, `student.taking.cno`, `UofT.course.title` and `UofT.course.title.text()` are all paths in the XML document shown in Figs. 2 and 3.

Given a node u in an XML document D and a path p in D, *reach*(u, p) is defined to be the set of all nodes and values reachable by following p from u in D. Furthermore, if P is a regular expression over an alphabet consisting of the reserved symbol `text()` and the element types and attributes in an XML document D, then a node v is reachable from a node u in D by following P, if there exists a string p in the regular language defined by P such that $v \in$ *reach*(u, p). The set of all such nodes is denoted by *reach*(u, P). For example, for the XML tree shown in Fig. 3, *reach* (u_1, `UofT.student`) = {u_2}, *reach*(u_1, `UofT.student.name`) = {`J. Smith`}, *reach*(u_1, `UofT.course.title.text()`) = {`Compilers`} and

reach(u_1,`UofT.(student.taking+course).cno`) = {`CSC258,CSC309`}.

Keys and Functional Dependencies for XML

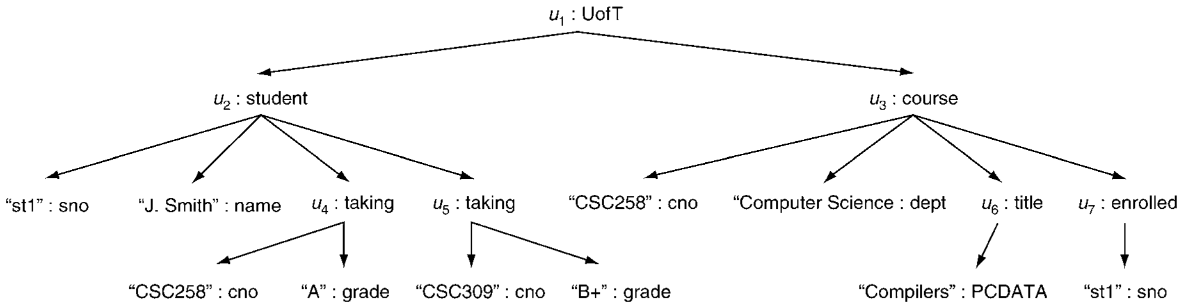
Absolute keys for XML were first considered by Fan and Simeón [7]. Let D be an XML document and τ an element type in D. Then *ext*(τ) is defined to be the set of all nodes of D of type τ. For example, if D is the XML tree shown in Fig. 3, then *ext*(`taking`) = {u_4, u_5}. Moreover, given a node υ and a list of attributes $X = [a_1,\ldots,a_k]$ that are defined for υ, $\upsilon[X]$ is defined as $[\upsilon.a_1,\ldots,\upsilon.a_k]$, where $\upsilon.a_i$ is the value of attribute a_i for node υ. For example, u_2[`sno`, `name`] is [`st1`, `J. Smith`] in the XML tree shown in Fig. 3.

Absolute keys for XML are defined as follows [6,7]. An absolute key is an expression of the form $\tau[X] \rightarrow \tau$, where τ is an element type and X is a nonempty set of attributes defined for τ. An XML document D satisfies this constraint, denoted by $D \vDash \tau[X] \rightarrow \tau$, if for every v_1, $v_2 \in$ *ext*(τ), if $v_1[X] = v_2[X]$, then $v_1 = v_2$. Thus, $\tau[X] \rightarrow \tau$ says that the X-attribute values of a τ-element uniquely identify the element in *ext*(τ). Notice that two notions of equality are used to define keys: value equality is assumed when comparing attributes, and node identity is used when comparing elements. The same symbol ' = ' is used for both, as it will never lead to ambiguity.

The following are typical keys for the XML document shown in Fig. 2: `student[sno]` → `student` and `course[cno]` → `course`. The first constraint says that student number (`sno`) is an identifier for students, and the second constraint says that course number

```
<UofT>
  <student sno = “st1” name = “J. Smith”>
     <taking cno = “CSC258” grade = “A”/>
     <taking cno = “CSC309” grade = “B+”/>
  </student>
  <course cno = “CSC258” dept = “Computer Science”>
     <title> Compilers </title>
     <enrolled sno = “st1”/>
  </course>
</UofT>
```

XML Integrity Constraints. Figure 2. Example of an XML document.



XML Integrity Constraints. Figure 3. Tree representation of the XML document shown in Fig. 2.

(cno) is an identifier for courses. It should be noticed that if courses in different departments can have the same course number, then key course[cno] → course has to be replaced by course[cno, dept] → course.

Since XML documents are hierarchically structured, users may be interested in the entire document as well as in its sub-documents. The latter give rise to relative keys, that only hold on certain sub-documents. The extension of Fan and Simeón's language to the relative case was done by Arenas et al. in [8]. This language is introduced below. Notation $v_1 \prec v_2$ is used when v_1 and v_2 are two nodes in an XML document and v_2 is a descendant of v_1. For example, $u_1 \prec u_4$ and $u_3 \prec u_7$ in the XML tree shown in Fig. 3.

A relative key is an expression of the form $\tau(\tau_1[X] \rightarrow \tau_1)$, where τ, τ_1 are element types and X is a nonempty set of attributes that is defined for τ_1. This constraint says that relative to each node v of type τ, the set of attributes X is a key for all the τ_1-nodes that are descendants of v. That is, an XML document D satisfies this constraint, denoted by $D \vDash \tau(\tau_1[X] \rightarrow \tau_1)$, if for every $v \in ext(\tau)$, and for every $v_1, v_2 \in ext(\tau_1)$ such that $v \prec v_1$ and $v \prec v_2$, if $v_1[X] = v_2[X]$, then $v_1 = v_2$. For example, the following is a typical relative key for the XML document shown in Fig. 2: course (enrolled[sno] → enrolled). This constraint says that relative to the elements of type course, sno is an identifier for the elements of type enrolled. Thus, this constraint states that every student can enroll at most once in a given course. It should be noticed that the previous constraint cannot be replaced by the absolute key enrolled[sno] → enrolled, since this states that every student can enroll in at most one course. It should also be noticed that absolute keys are a special case of relative keys when τ is taken to be the type of the root of the XML documents. For example, for the case of the XML document shown in Fig. 2, absolute key student [sno] → student is equivalent to relative key UofT (student[sno] → student).

A more powerful language for expressing XML keys was introduced by Buneman et al. [4,3]. This language allows the definition of absolute and relative keys. More precisely, an absolute key is an expression of the form $(P, \{Q_1,...,Q_n\})$, where P, $Q_1,...,Q_n$ are regular expressions. An XML document D satisfies this key if for every pair of nodes $u, v \in reach(root, P)$, where $root$ is the node identifier of the root of D, if $reach(u, Q_i) \cap reach(v, Q_i) \neq \emptyset$, for every $i \in [1, n]$, then u and v are the same node. For example, a key dependency can be used to express that name is an identifier for students in the University of Toronto database: (`UofT.student`, {`name`}). Notice that if a nested structure is used in this database to distinguish first names from last names:

```
<UofT>
  <student sno="st1">
    <name>
      <first> John </first>
      <last> Smith </last>
    </name>
  ...
</UofT>
```

then to characterize name as an identifier for students, two paths are included in the right-hand side of the key dependency: (`UofT.student`, {`name.first.text()`, `name.last.text()`}).

In [4,3], Buneman et al. defined a relative key as a pair of the form (P', K), where P' is a regular expression and K is an absolute key of the form $(P, \{Q_1,...,Q_n\})$. An XML document satisfies this key if every node reached from the root by following a path in P' satisfies K, that is, for every $u \in reach(root, P')$ and for every $v_1,v_2 \in reach(u, P)$, if $reach(v_1, Q_i) \cap reach(v_2, Q_i) \neq \emptyset$, for every $i \in [1, n]$, then v_1 and v_2 are the same node. For example, a relative key constraint can be used to express that a student cannot take the same course twice in the XML database shown in Fig. 2: (`UofT.student`, (`taking`, {`cno`})). This key dependency is relative since two distinct students can take the same course. It should be noticed that every key $\tau(\tau_1[a_1,...,a_k] \rightarrow \tau_1)$ in Arenas et al.'s language [8] can be represented as $(\Sigma^*.\tau,(\Sigma^*.\tau_1, \{a_1,...,a_k\}))$ in Buneman et al.'s language [4,3], where Σ is the alphabet consisting of all the element types in an XML document.

In [2], Arenas and Libkin introduced a functional dependency language for XML. In this language, a functional dependency over an XML document D is an expression of the form $X \rightarrow p$, where $X \cup \{p\}$ is a set of paths in D. To define the notion of satisfaction for functional dependencies, Arenas and Libkin [2] used a relational representation of XML documents. Given an XML document D, let $paths(D)$ be the set of all paths in D. Then a tree tuple over D is a mapping t that assigns to each path $p \in paths(D)$ either a node identifier or a data value or the null value $\perp$, in a way that is consistent with the tree representation of D. Formally, if $p \in paths(D)$, the last symbol of p is an element type τ and $t(p) \neq \perp$, then (i) $t(p) = u$, where u is a node identifier in D of type τ; (ii) if p' is a prefix of p, then $t(p')$ is a node identifier that lies on the path from the root to u in D; (iii) if a is an attribute defined for u in D, then $t(p.a) = u.a$; and (iv) if p.`text()` is a path in D, then there is a child s of u of type `PCDATA` such that $t(p.$`text()`$) = s$. A tree tuple is maximal if it cannot be extended to another one by changing some nulls to either node identifiers or data values. The set of maximal tree tuples in D is denoted by $tuples(D)$. Then functional dependency $\{p_1,...,p_m\} \rightarrow p$ is true in D if for every pair $t_1, t_2 \in tuples(D)$, whenever $t_1(p_i) = t_2(p_i) \neq \perp$ for all $i \leq n$, then $t_1(p) = t_2(p)$ holds.

For example, let D be the XML document shown in Figs. 2 and 3. In this database, there is at most one name associated with each student number, which can be represented by means of functional dependency `UofT.student.sno` → `UofT.student.name`. XML document D satisfies this constraint, which can be proved formally by constructing $tuples(D)$. Table 1 shows the two tree tuples contained in this set. These tuples satisfy `UofT.student.sno` → `UofT.student.name` since they have the same values in `UofT.student.sno` and also in `UofT.student.name`.

Arenas and Libkin's language [2] can also be used to express relative functional dependencies. For example, in the XML document shown in Fig. 2, every student has at most one grade in each course (which is the final grade for the course). This relative functional dependency can be expressed as {`UofT.student`, `UofT.student.taking.cno`} → `UofT.student.taking.grade`, whose satisfaction can be checked by considering again the tree tuples shown in Table 1.

Inclusion Dependencies and Foreign Keys for XML

One of the first XML integrity constraint languages was proposed by Abiteboul and Vianu [1]. They considered inclusion dependencies of the form $P \subseteq Q$, where P and Q are regular expressions. An XML document D rooted at u satisfies this constraint if $reach(u, P) \subseteq reach(u,Q)$. For example, in the database shown in Fig. 2, the following constraint expresses that the set of courses taken by each student is a subset of the set of courses given by the university: `UofT.student.taking.cno` ⊆ `UofT.course.cno`. An inclusion constraint $P \subseteq Q$ where P are Q are paths, like in the previous example, is called a path constraint [1]. In [5], Buneman et al. introduced a more powerful path constraint language. Given an XML document D and paths p_1, p_2, p_3 in D, in this language a constraint is an expression of either the forward form $\forall x \forall y(x \in \mathit{reach}(\mathit{root}, p_1) \wedge y \in \mathit{reach}(x, p_2) \rightarrow y \in \mathit{reach}(x, p_3))$, where *root* represents the root of the XML document, or the backward form $\forall x \forall y(x \in \mathit{reach}(\mathit{root}, p_1) \wedge y \in \mathit{reach}(x, p_2) \rightarrow x \in \mathit{reach}(y, p_3))$. This language can be used to express relative inclusion dependencies. For example, if the document shown in Fig. 2 is extended to store information about students and courses in many universities, ⟨UofT⟩ is replaced by ⟨university⟩:

```
<db>
  <university name="UofT"> ...
  </university>
  <university name="UCLA"> ...
  </university>
</db>
```

Then the following dependency in Buneman et al.'s language [5] can be used to state that for each university, the set of courses taken by each one of its students is a subset of the set of courses given by that university:

$$\forall x \forall y(x \in \mathit{reach}(\mathit{root}, \text{db.university}) \wedge y \in \mathit{reach}(x, \text{student.taking.cno}) \rightarrow y \in \mathit{reach}(x, \text{course.cno})).$$

This constraint is relative to each university and, thus, it cannot be expressed by using Abiteboul and Vianu's path constraints [1], as this language can only express constraints on the entire document. By using Abiteboul and Vianu's approach, it can only be said that if a student is taking a course, then this course is given in some university: `db.university.student.taking.cno` $\subseteq$ `db.university.course.cno`.

A language for expressing absolute foreign keys for XML was proposed by Fan and Libkin in [6]. In this language, an absolute foreign key is an expression of the form $\tau_1[X] \subseteq_{FK} \tau_2[Y]$, where τ_1, τ_2 are element types, and X, Y are nonempty lists of attributes defined for τ_1 and τ_2, respectively, and $|X| = |Y|$. This constraint is satisfied by an XML document D, denoted by $D \vDash \tau_1[X] \subseteq_{FK} \tau_2[Y]$, if $D \vDash \tau_2[Y] \rightarrow \tau_2$, and in addition for every $v_1 \in \mathit{ext}(\tau_1)$, there exists $v_2 \in \mathit{ext}(\tau_2)$ such that $v_1[X] = v_2[Y]$. The extension of Fan and Libkin's proposal to the relative case was done by Arenas et al. in [8]. In this proposal, a relative foreign key is an expression of the form $\tau(\tau_1[X] \subseteq_{FK} \tau_2[Y])$, where τ, τ_1, τ_2 are element types, X and Y are nonempty lists of attributes defined for τ_1 and τ_2, respectively, and $|X| = |Y|$. It indicates that for each node v of type τ, X is a foreign key of descendants of v of type τ_1 that references a key Y of τ_2-descendants of v. That is, an XML document D satisfies this constraint, denoted by $D \vDash \tau(\tau_1[X] \subseteq_{FK} \tau_2[Y])$, if $D \vDash \tau(\tau_2[Y] \rightarrow \tau_2)$ and for every $v \in \mathit{ext}(\tau)$ and every $v_1 \in \mathit{ext}(\tau_1)$ such that $v \prec v_1$, there exists $v_2 \in \mathit{ext}(\tau_2)$ such that $v \prec v_2$ and $v_1[X] = v_2[Y]$.

Key Applications

XML integrity constraints are essential to schema design, query optimization, efficient storage, index

XML Integrity Constraints. Table 1. Maximal tree tuples in the XML tree shown in Fig. 3

Path	t_1	t_2
UofT	u_1	u_1
UofT.student	u_2	u_2
UofT.student.sno	st1	st1
UofT.student.name	J. Smith	J. Smith
UofT.student.taking	u_4	u_5
UofT.student.taking.cno	CSC258	CSC309
UofT.student.taking.grade	A	B+
UofT.course	u_3	u_3
UofT.course.cno	CSC258	CSC258
UofT.course.dept	Computer Science	Computer Science
UofT.course.title	u_6	u_6
UofT.course.title.text()	Compilers	Compilers
UofT.course.enrolled	u_7	u_7
UofT.course.enrolled.sno	st1	st1

design, data integration, and in data transformations between XML and relational databases.

Cross-references

- ▶ Database Dependencies
- ▶ Functional Dependency
- ▶ Key

Recommended Reading

1. Abiteboul S. and Vianu V. Regular path queries with constraints. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997, pp. 122–133.
2. Arenas M. and Libkin L. A normal form for XML documents. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 85–96.
3. Buneman P., Davidson S., Fan W., Hara C., and Tan W.C. Reasoning about keys for XML. In Proc. 8th Int. Workshop on Database Programming Languages, 2001, pp. 133–148.
4. Buneman P., Davidson S., Fan W., Hara C., and Tan W.C. Keys for XML. In Proc. 10th Int. World Wide Web Conference, 2001, pp. 201–210.
5. Buneman P., Fan W., and Weinstein S. Path constraints in semistructured and structured databases. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 129–138.
6. Fan W. and Libkin L. On XML integrity constraints in the presence of DTDs. J. ACM, 49(3):368–406, 2002.
7. Fan W. and Siméon J. Integrity constraints for XML. In Proc. 19th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2000, pp. 23–34.
8. Marcelo Arenas, Wenfei Fan and Leonid Libkin. On the Complexity of Verifying Consistency of XML Specifications. SIAM Journal on Computing, 38(3):841–880, 2008.

XML Message Brokering

▶ XML Publish/Subscribe

XML Metadata Interchange

Michael Weiss
Carleton University, Ottawa, ON, Canada

Synonyms

XMI

Definition

XMI (XML Metadata Interchange) is an XML-based integration framework for the exchange of models, and, more generally, any kind of XML data. XMI is used in the integration of tools, repositories, applications, and data warehouses. The framework defines rules for generating XML schemas from a metamodel based on the Metaobject Facility (MOF). XMI is most frequently used as an interchange format for UML, although it can be used with any MOF-compliant language.

Key Points

The motivation for introducing XMI was the need to provide a standard way through which UML tools could exchange UML models. XMI produced by one tool can generally be imported by another tool, which allows exchange of models among tools by different vendors, or the exchange of models with other types of tools upstream or downstream the tool chain. As stated above, XMI is not limited to mapping UML to XML, but it provides rules to generate DTDs or XML schemas and XML documents from any MOF-compliant language. Thus, a model that conforms to some MOF-compliant metamodel can be translated to an XML document that conforms to a schema generated according to the rules of the XMI standard.

However, since XMI represents UML models in XML, XMI is more widely applicable. For example, XMI has been used for the analysis of models and for the integration of applications, both internal and with external parties. The key idea underlying XMI is to provide rules for generating schemas from a metamodel. These include the mapping of model attributes to XML elements and attributes, and model associations to containment relationships between XML elements or to links. Notably, XMI maps inheritance to composition in XML, in which elements and attributes from superclasses in the model are "copied down" into the XML document, as XML lacks inheritance.

Cross-references

- ▶ Extensible Markup Language
- ▶ Metamodel
- ▶ Meta Object Facility
- ▶ UML

Recommended Reading

1. Carlson D. Modeling XML Applications with UML. Addison-Wesley, Reading, MA, 2001.
2. Grose T., Doney G., and Brodsky S. Mastering XMI. Wiley, New York, 2002.
3. OMG, MOF 2.0/XMI Mapping, version 2.1.1, 2007, http://www.omg.org/spec/XMI/2.1.1

XML Parsing, SAX/DOM

Chengkai Li
University of Texas at Arlington, Arlington, TX, USA

Definition

XML parsing is the process of reading an XML document and providing an interface to the user application for accessing the document. An XML parser is a software apparatus that accomplishes such tasks. In addition, most XML parsers check the *well-formedness* of the XML document and many can also validate the document with respect to a DTD (Document Type Definition) or XML schema. Through the parsing interface, the user application can focus on the application logic itself, without dwelling on the tedious details of XML.

There are mainly two categories of XML programming interfaces, DOM (Document Object Model) and SAX (Simple API for XML). DOM is a tree-based interface that models an XML document as a tree of nodes, upon which the application can search for nodes, read their information, and update the contents of the nodes. SAX is an event-driven interface. The application registers with the parser various event handlers. As the parser reads an XML document, it generates events for the encountered nodes and triggers the corresponding event handlers. Recently, there have been newly proposed XML programming interfaces such as *pull-based parsing*, *e.g.*, StAX (Streaming API for XML), and *data binding*, *e.g.*, JAXB (Java Architecture for XML Binding).

Historical Background

DOM (Document Object Model) was initially used for modeling HTML (HyperText Markup Language) by various Web browsers. As inconsistencies existed among the individual DOM models adopted by different browsers, inter-operability problems arose in developing browser-neutral HTML codes. W3C (World Wide Web Consortium) standardized and released DOM Level 1 specification on October 1, 1998, with support for both HTML and XML. DOM Level 2 was released in November 2000 and added namespace support. The latest specification DOM Level 3 was released in April 2004.

SAX (Simple API for XML) was developed in late 1997 through the collaboration of several implementers of early XML parsers and many other members of the XML-DEV mailing list. The goal was to create a parser-independent interface so that XML applications can be developed without being tied to the proprietary API (Application Programming Interface) of any specific parser. SAX1 was finalized and released on May 11, 1998. The latest release SAX2, finalized in May 2000, includes namespace support.

Foundations

XML parsing is the process of reading an XML document and providing an interface to the user application for accessing the document. An XML parser is a software apparatus that accomplishes such tasks. In addition, most XML parsers check the *well-formedness* of the XML document and many can also validate the document with respect to a DTD (Document Type Definition) [2] or XSD (XML Schema (W3C)) [11]. Through the parsing interface, the user application can focus on the application logic itself, without dwelling on the tedious details of XML, such as Unicode support, namespaces, character references, well-formedness, and so on.

XML Programming Interfaces

Most XML parsers can be classified into two broad categories, based on the types of API that they provide to the user applications for processing XML documents.

Document Object Model (DOM): DOM is a tree-based interface that models an XML document as a tree of various nodes such as elements, attributes, texts, comments, entities, and so on. A DOM parser maps an XML document into such a tree rooted at a `Document` node, upon which the application can search for nodes, read their information, and update the contents of the nodes.

Simple API for XML (SAX): SAX is an event-driven interface. The application receives document information from the parser through a `ContentHandler` object. It implements various event handlers in the interface methods in `ContentHandler`, and registers the `ContentHandler` object with the SAX parser. The parser reads an XML document from the beginning to the end. When it encounters a node in the document, it generates an event that triggers the corresponding event handler for that node. The handler thus applies the application logic to process the node specifically.

The SAX and DOM interfaces are quite different and have their respective advantages and disadvantages. In general, DOM is convenient for random access to arbitrary places in an XML document, can not only read but also modify the document, although it may take a significant amount of memory space. To the contrary, SAX is appropriate for accessing local information, is much more memory efficient, but can only read XML.

- DOM is not memory efficient since it has to read the whole document and keep the entire document tree in memory. The DOM tree can easily take as much as ten times the size of the original XML document [4]. Therefore it is impossible to use DOM to process very large XML documents, such as the ones that are bigger than the memory. In contrast, SAX is memory efficient since the application can discard the useless portions of the document and only keep the small portion that is of interests to the application. A SAX parser can achieve constant memory usage thus easily handle very large documents.
- SAX is appropriate for streaming applications since the application can start processing from the beginning, while with DOM interface the application has to wait till the entire document tree is built before it can do anything.
- DOM is convenient for complex and random accesses that require global information of the XML document, whereas SAX is more suited for processing local information coming from nodes that are close to each other. The document tree provided by DOM contains the entire information of the document, therefore it allows the application to perform operations involving any part of the document. In comparison, SAX provides the document information to the application as a series of events. Therefore it is difficult for the application to handle global operations across the document. For such complex operations, the application would have to build its own data structure to store the document information. The data structure may become as complex as the DOM tree.
- Since DOM maintains information of the entire document, its API allows the application to modify the document or create a new document, while SAX can only read a document.

DOM and SAX are the two standard APIs for processing XML documents. Most major XML parsers support them. There are also alternative tree-based XML models that were designed to improve upon DOM, including JDOM and DOM4J. In addition to DOM and SAX, other types of APIs for processing XML documents have emerged recently and are supported by various parsers. Two examples are *pull-based parsing* and *Java data binding*.

Pull-Based Parsing: Evolving from XMLPULL, StAX (Streaming API for XML) also works in a streaming fashion, similar to SAX. Different from SAX where the parser pushes document information to the application, StAX enables the application to pull information from the parser. This API is more natural and convenient to the programmer since the application takes full control in processing the XML document.

Java Data Binding: A data-binding API provides the mapping between an XML document and Java classes. It can construct Java objects from the document (*marshalling*) or build a document from the objects (*unmarshalling*). Accessing and manipulating XML documents thus become natural and intuitive, since such operations are performed through the methods of the objects. The application can thus focus on the semantics of the data themselves instead of the details of XML. JAXB (Java Architecture for XML Binding) is a Java specification based on this idea.

Validating Parsers

In addition to accessing XML documents, another critical functionality of XML parsers is to validate the correctness of the documents. Given that XML is a popular data model for data representation and exchange over the Internet, the correctness of XML documents is important for applications to work properly. It is difficult to let the application itself handle incorrect documents that it does not expect. Fortunately, most major XML parsers have the ability to validate the correctness of XML documents.

The correctness of an XML document can be defined at several levels. At the bottom, the document should follow the syntax rules of XML. Such a document is called a *well-formed* document. For example, every non-empty element in a well-formed document should have a pair of starting tag and ending tag. Furthermore, the document should conform to certain semantic rules defined for the application domain, such as a "`state`" node containing one and only one "`capital`" node. Such semantic rules can be defined by XML schema specifications such as DTD or XSD (XML Schema (W3C)). An XML document that complies with a schema is called a *valid* document. Finally, the application may enforce its own specific semantic rules.

In principle all the parsers are required to perform mandated checks of well-formedness, although there are parser implementations that do not. A parser that checks for the validity of XML documents with respect to XML schema in addition to their well-formedness is a *validating parser*. Schema validation is support by both DOM and SAX API. Most major XML parsers are validating parsers, although some may turn off schema validation by default.

XML Parsing Performance

There are relatively few studies in the literature on performance of XML parsing. However, as the first step in every application that takes XML documents to process, parsing can easily become the bottleneck of the application performance.

DOM is memory intensive since it has to hold the entire document tree in memory, making it incapable in handling very large documents. Therefore, efforts have been made to improve DOM parser performance by exploiting *lazy* XML parsing [7]. The key idea is to avoid loading unnecessary portions of the XML document into the DOM tree. It consists of two stages. The pre-parsing stage builds a virtual DOM tree and the progressive parsing stage expands the virtual tree with concrete contents when they are needed by the application. Farfán et al. [3] further extended the idea to reduce the cost of the pre-parsing stage by partitioning an XML document, thus only reading a partition into the DOM tree when it is needed.

Nicola et al. [6] investigated several real-world XML applications where the performance of SAX parsers is a key obstacle to the success of the projects. They further verified that schema validation can add significant overheads, which can sometimes even be several times more expensive than parsing itself.

Validation often incurs significant processing costs. One reason for such low efficiency is the division of parsing and validation steps. In conventional parsers these two steps are separate this is because validation often requires the entire document to be in the memory thus having to wait till the parsing is finished. Therefore, even for a SAX parser, the advantage of memory efficiency is lost. To cope with this challenge, there have been studies on integrating parsing and validation into a *schema-specific parser* [1,9,12]. For example, [1] constructs a push-down automaton to combine parsing and validation. Van Engelen et al. [10] uses deterministic finite state automata (DFA) to integrate them and the DFA is built upon the schema according to mapping rules. Kostoulas et al. [5] further applies compilation techniques to optimize such integrated parsers.

Takase et al. [8] explores a different way to improve parser performance. It memorizes parsed XML documents as byte sequences and reuses previous parsing results when the byte sequence of a new XML document partially matches the memorized sequences.

Key Applications

Every XML application has to parse an XML document before it can access the information in the document and perform further processing. Therefore, XML parsing is a critical component in XML applications.

URL to Code

List of XML parsing interfaces

DOM, http://www.w3.org/DOM/
JDOM, http://jdom.org/
DOM4J, http://dom4j.org/
SAX, http://www.saxproject.org/
StAX, http://jcp.org/en/jsr/detail?id=173
XMLPULL, http://www.xmlpull.org/
JAXB, http://www.jcp.org/en/jsr/detail?id=222

List of XML parsers

Ælfred, http://saxon.sourceforge.net/aelfred.html
Crimson, http://xml.apache.org/crimson/
Expat, http://expat.sourceforge.net/
JAXP, https://jaxp.dev.java.net/
Libxml2, http://xmlsoft.org/index.html
MSXML, http://msdn.microsoft.com/en-us/library/ms763742.aspx
StAX Reference Implementation (RI), http://stax.codehaus.org/
Sun's Stax implementation, https://sjsxp.dev.java.net/
XDOM, http://www.philo.de/xml/
Xerces, http://xerces.apache.org/

Cross-references

▶ XML
▶ XML Attribute
▶ XML Document
▶ XML Element
▶ XML Programming
▶ XML Schema

Recommended Reading

1. Chiu K., Govindaraju M., and Bramley R. Investigating the limits of SOAP performance for scientific computing. In Proc. 11th IEEE Int. Symp. on High Performance Distributed Computing, 2002, pp. 246–254.
2. Document Type Declaration, http://www.w3.org/TR/REC-xml/#dt-doctype
3. Farfán F., Hristidis V., and Rangaswami R. Beyond lazy XML parsing. In Proc. 18th Int. Conf. Database and Expert Syst. Appl., 2007, pp. 75–86.
4. Harold E.R. Processing XML with Java(TM): a Guide to SAX, DOM, JDOM, JAXP, and TrAX. Addison-Wesley, MA, USA, 2002.
5. Kostoulas M., Matsa M., Mendelsohn N., Perkins E., Heifets A., and Mercaldi M. XML screamer: an integrated approach to high performance XML parsing, validation and deserialization. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 93–102.
6. Nicola M. and John J. XML parsing: a threat to database performance. In Proc. Int. Conf. on Information and knowledge Management, 2003, pp. 175–178.
7. Noga M., Schott S., and Löwe W. Lazy XML processing. In Proc. 2nd ACM Symp. on Document Engineering, 2002, pp. 88–94.
8. Takase T., Miyashita H., Suzumura T., and Tatsubori M. An adaptive, fast, and safe XML parser based on byte sequences memorization. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 692–701.
9. Thompson H. and Tobin R. Using finite state automata to implement W3C XML schema content model validation and restriction checking. In Proc. XML Europe, 2003, pp. 246–254.
10. Van Engelen R. Constructing finite state automata for high performance XML web services. In Proc. Int. Symp. on Web Services, 2004, pp. 975–981.
11. XML Schema (W3C), http://www.w3.org/XML/Schema
12. Zhang W. and Van Engelen R. A table-driven streaming XML parsing methodology for high-performance web services. In Proc. IEEE Int. Conf. on Web Services, 2006, pp. 197–204.

XML Persistence

▶ XML Storage

XML Process Definition Language

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

XPDL

Definition

The primary standards body for workflow management and business process interoperability.

Key Points

The XML Process Definition Language (XPDL) is a format standardized by the Workflow Management Coalition to interchange business process definitions between different modeling tools, BPM suites, workflow engines and other software applications. XPDL defines a XML schema for specifying the declarative part of workflow.

XPDL is designed to exchange the process design, both the graphics and the semantics of a workflow business process. XPDL contains elements to hold the X and Y position of the activity nodes as well as the coordinates of points along the lines that link those nodes. XPDL provides the serialization for BPMN. This distinguishes XPDL from BPEL, which is also a process definition format, but does not contain elements to represent the graphical aspects of a process diagram and BPEL focuses exclusively on the executable aspects of the process.

Cross-references

▶ Business Process Execution Language
▶ Business Process Modeling Notation
▶ Workflow Model
▶ Workflow Schema

XML Programming

Peter M. Fischer
ETH Zurich, Zurich, Switzerland

Synonyms

XML application development

Definition

XML programming [2] covers methods and approaches to process, transform and modify XML data, often within the scope of a larger application which uses imperative programming languages. Similar to database programming, an important issue in XML programming is the impendence mismatch between the existing programming models, which are mostly based on an object-oriented data model and use an imperative style, and XML programming approaches, which are based on an XML data model, and apply various programming styles. A plethora of XML programming approaches exists, driven by different usage patterns of XML in applications. The XML programming approaches can be classified into three areas: (i) XML APIs to existing languages, (ii) XML extensions of existing programming languages, and (iii) Native XML processing languages. The varying sets of XML programming requirements and XML programming approaches make it impossible to declare a clearly preferable approach. Careful analysis by the application designer is needed to determine which technique is best suited for a particular setting.

Historical Background

The need for XML programming arose soon after XML had been established as a simplified derivative of SGML, capable of representing any kind of semistructured data. Historically, three areas were influential to shape the directions in XML programming:

- Low-level APIs oriented towards document parsing: XML being regarded as a structured document format, a popular way to program XML is based on letting a document parser transform the XML into some internal structure of the target programming language, based on concepts of the areas of compiler construction. This approach had already been used for HTML with great success, making DOM the default API for client-side web programming.
- Document Transformation languages are a second influence also based on the document nature of XML. Such languages specify rules to apply to specific fragments of a document and generate new document parts out of the matched fragments and modification/creation statements. A well-known transformation language out of the SGML world is DSSSL, which is often used in the context of DocBook for scientific document processing.
- Database programming: a third influence stems from treating XML as a generic data representation format that, in turn, can be maintained and queried in a similar way as relational data. In database programming, two different data models (relational/OO) and two different programming styles (declarative/imperative) need to be reconciled as well. Four main directions were developed:

1. Call-level interfaces: a literal string of the database language is given as a parameter to a function of the imperative host language. This function is used to interface with the DBMS and returns a result that can be turned into data types of the host language. Typical examples of these are ODBC or JDBC.
2. Embedded SQL: the query expressions of the relational language are embedded in the host application code, allowing for better type checking and hiding the details of the actual interfacing to the DBMS.
3. Automatic mapping layers: instead of modeling both the application with (usually) object-oriented methodologies (UML) and the database with relational methodologies (E/R) and writing expressions in both the database and the host language, only the application is modeled and the code for the application is written. The necessary database schema and the query expressions to retrieve and insert data into the DBMS are automatically generated from the application. The database aspect is hidden and development is simplified. The drawbacks are the lack of flexibility and often lower performance than with a separate database design.
4. Procedural extensions to SQL: an inverse approach to hiding the database is to integrate the application into the database, thus benefitting from the stability and scalability of the DBMS environment while exploiting performance benefits by being "close" to the data. For this approach, SQL has been extended with imperative constructs to enhance its expressive power (Turing-completeness) and make the development style more suitable to general applications. Well-known examples of such SQL extensions are PL/SQL (Oracle) or Transact-SQL (Sybase, Microsoft).

Foundations

Specific Requirements of XML Programming

While there are certain similarities to related areas like database programming (in particular, the impedance mismatch), XML programming has its own special set of requirements and challenges, which can be traced back to two specific areas: (i) the advantages and deficiencies of XML as data and programming model, and the resulting differences to other, established programming methodologies, and (ii) the widely varying and non-uniform use of XML in terms of usage and operations.

Conceptual Aspects of XML

XML has a clear set of advantages that set it apart from other approaches used to represent data, and led to its rapid acceptance.

A determining factor of the success of XML has been the independence from particular vendors and platforms. This advantage has been further strengthened by a large number of high-quality tools for XML technologies that are often available under permissive licenses, making the integration of XML technology into existing or new applications easy. As a result, knowledge about XML and related technologies is available freely.

The XML syntax is both human readable and machine readable, avoiding problems that occur if a format is only specialized for one way of interpretation, such as unstructured text or complex binary encodings.

XML is not just a document format, but includes methods and technologies for metadata description, RPC, workflow management, declarative querying, security, document processing and many more. These technologies and methods cover a large part of the required features for application development and ensure high interoperability not just among the classes of XML technology, but also among the applications building on them.

From a data and application design point of view, XML provides a number of benefits that make it a good choice over competing approaches: in contrast to relational or object-oriented approaches, data and its interpretation are decoupled. This decoupling allows writing code that works on schema-less data, but does not prohibit adding schema when needed. Working with schema-less data is particularly helpful to shorten time-to-market times when building new applications, as the time-consuming and tedious schema design phase can be shortened. Similarly, it is helping with long-lived data (common in many business settings) where the code is already outdated, but the data will still be needed for new applications. The benefit of not being forced to have a schema is being further enhanced by the fact that the semi-structured model of XML allows representing a large spectrum of data "shapes," reaching from unstructured data like annotated text to highly structured data like a relational table. The XML syntax and data models are not limited to represent data, but are also used to represent metadata and code, allowing uniform management and modifications.

XML, however, does have limitations that reduce its usefulness for certain applications: many standardization efforts, as well as many development efforts, follow a bottom-up approach by defining small entities with low complexity. While this approach ensures that the individual standards and components are easy to understand and use, a combination of standards or components does not always cover all required aspects, thus leaving room for interpretation and incompatibility. A more serious conceptual problem is the limitation of the data model towards tree structures, making it difficult to express arbitrary graph structures or N:M relations. This issue is being aggravated because there is not a commonly accepted way to specify references in XML data. Another important deficiency is the lack of standard design methodology. UML and Entity-Relationship-Modeling have greatly helped to establish the concepts of object-oriented and relational technologies, respectively. By using them, modeling and developing applications are greatly simplified.

While not being a deficiency of XML per se, the mismatch between XML concepts and programming-language/database concepts complicates the use of XML and makes programming for it more difficult: since the data model of XML is neither object-based nor relational, translation needs to be done from and to XML, keeping it outside the usual type system and program analysis of the existing environments. The ability of XML to both work without any schema and represent many shapes of data makes direct mapping of arbitrary XML instances to a strict object-based structure (or to relational schema) often impossible, forcing the use of generic and less useable APIs. The object-oriented approach of hiding the data and binding the methods tightly to this data are juxtaposed to the nature of XML where data are explicitly exposed and accessible to many different methods.

Differences in XML Usage

The second main issue in programming is the wide variety of ways in which XML is used. This usage includes the type of content that an XML instance represents, the operations that are performed on the data and some non-functional properties that have an impact on the processing such as size, structure, persistence etc.

XML is used to represent a large number of different *types of content*, each leading to different functional requirements, favoring specific approaches for XML programming.

The first, and currently most popular, class of using XML is as a *document storage format*. Typical examples are XHTML, office documents (OOXML, ODF), and graphics formats (SVG). These files are usually used to be presented in a human readable format by an application, transformed into another document format, modified and stored again. A second class is *database content*, either in forms of document collections or representing complex data in XML format. Here, the focus is on maintaining a large data set of XML and being able to successfully retrieve matching data. A third class is about program *code*, *metadata* and *configuration data*, where XML is used to determine the behavior of a data processing system. A fourth class is to use it as *a communication format*, in cases such as SOAP, or REST, putting the focus of transferring data or state in a loosely coupled, yet efficient way from one system to the other. A fifth and upcoming class is to use XML for *log files*, *event streams* or *scientific data streams*, requiring the analysis of the data by correlating data or detecting event patterns.

On these types of content, different *operations* are performed, which in turn are better supported by certain XML programming approaches: Next to the complete retrieval of the XML content, limited or full-scale *query operations* such as filtering/selection, projection or the joins are common operations. A typical operation for many scenarios is also the *creation of new XML data*. *Updating existing XML data* is common in database settings, document management, code and metadata. More specialized operations are *full text search*, relevant for document collections and databases and *trigger processing* and *event generation*, which are commonly used either in databases or data streams.

In addition to the types of content and the operations, a number of non-functional properties have an impact on which XML programming approach to use in particular scenarios. Having large *volumes* of XML requires an approach that supports the relevant technologies for scaling well, but for small volumes such an approach might be too heavyweight. Similarly, dealing with *persistent data* requires different, more elaborate approaches if only *temporary data* are used. Using very *structured data* allows different optimization in storage and processing of XML than using *unstructured data*, since implementation techniques out of either relational databases or text processing are more appropriate. Again, different XML programming approaches are better suited for one or the other. Similarly, some approaches can deal better with data that is *read only*, *append only* or freely *updateable*.

Additional Classification Criteria

Next to the criteria that come from the basic properties of XML and the variety of the XML usage scenarios, there are additional classification criteria that – in the widest sense – are concerned with architecture, “user experience” of the approaches, typing, compatibility and performance.

The *integration of XML processing into the architecture* of an application is an important aspect of an XML programming approach. The traditional approach is to use XML just as the input and/or serialization format and leave the architecture of the application unaffected. A second, more disruptive but also more powerful approach, is to use an XML type inside the type system of the host programming language. This type is usually also augmented by more or less powerful expressions that work on it. The third, and most intrusive approach, is to use an XML data model and a native XML expression language throughout the whole application. These three approaches show an increasing amount of *disruptiveness*, forcing developers and architects to give up on the existing knowledge on application development. On the other hand, the three approaches also show an increasing ability to utilize the advantages of XML, such as schema-less processing while reducing the impedance mismatch between the XML world and the application. The first approach allows a programmer to have high *productivity* quickly, since existing programming knowledge can be re-used, but on the long run the complications of dealing with the impedance mismatch in this approach make the second and the third approach a more compelling option. *Compliance to W3C standards* is important when interoperating with other XML-based applications, but certain standards, such as XML Schema, can add a significant amount of complexity to an approach. Closely related are the aspects of the *XML data model* (e.g., Infoset, XDM, proprietary), *type support* for the XML data (general node types versus full schema types) and the support for *static type analysis*, because an extended XML type support adds complexity to an API, but allows for more strict correctness checking and better optimizations. Important aspects of the acceptance of an XML programming approach are *performance* and *optimizability*. Low-level approaches that are not deeply integrated into the architecture tend to have the best performance on the short run, as developers can choose their access pattern on the data and optimize a program written in a language they are familiar with. On the long run, declarative solutions with a uniform data and expression model and strong typing support hold the most promise, as they can shift the burden of optimizing from the programmer to optimizers built into the system.

Approaches to XML Programming

A large number of approaches to develop XML-based applications exist, they are influenced by document processing and database programming methods. In this section, the approaches are clustered along the amount of change to the architecture and programming style they require compared to object-oriented/imperative programming: when making a decision on a specific approach to use, the main trade-off is how much of the XML advantages should/need to be used compared to the disruption caused by moving away from the well-known application development environments.

The three main classes of approaches are XML interfaces to existing languages, XML extensions to existing languages and XML-oriented/native XML programming languages.

The first class, *interfaces to existing languages*, provides methods to maintain all of the program logic and complexity inside the established imperative/object-oriented languages. XML is treated like any other source of outside data by limiting its impact to an adaptation layer/API and representing XML as instances of the native, non-extended type system, e.g., as tree of node objects. By doing so, the impact on existing programming models is kept low, but at the cost of either having very generic APIs with low productivity or limited flexibility. The impedance mismatch in the data model and the expressions as well as the purely imperative programming style limits the possibilities for optimization.

This first class can further be broken down into two subclasses, *generic XML-oriented APIs* and *schema-driven code generation*. Generic XML-oriented APIs do not require any knowledge of the particular structure of an XML instance, thus representing XML as generic, low-level objects in the host language such as trees of nodes, event sequences or node item sequences. The majority of these APIs provide low level, parser-oriented programming interfaces such as DOM, SAX and StaX. They support a limited set of querying operations such as selection or projection, the creation of new XML and updates to existing XML; any more high-level functionality needs to be implemented in the imperative language. This allows for a large amount of generality and possibly high performance, since access and processing can be tailored to the needs of a specific application. This advantage, however, comes at a high price: the generic and low-level nature of the APIs cause low developer productivity. There also exist call-level interfaces to XML-oriented programming languages or database system such as XQJ, which allow shifting/moving of some XML-oriented operations outside the imperative program. This frees the developer from low-level work, but requires learning and understanding of a separate expression language with a different data model and different semantics.

Schema-driven code generation increases the abstraction level and developer productivity by automatically creating high-level host programming language objects representing specific, typed XML, e.g., by turning an XML item representing "person" data into a person object. This object can then be accessed and manipulated by the methods the object exposes, e.g., a method to get the name of a person. By doing so, the level of abstraction is increased, developers do not need to learn many details about XML and can stay within their well-understood object-oriented/imperative world, all leading to higher productivity. Schema knowledge (e.g., DTD, XML schema or an ad-hoc format) is needed to perform this automatic code generation, restricting this approach to scenarios where the XML instances are highly structured and this structure information is known in advance. This limits the flexibility of the code generation approach, as it reduces XML to a representation format of host language objects. Well-known examples of such code-generation approaches are XML Beans or JAXB.

The second class of XML programming approaches, *XML extensions to existing programming languages*, still maintains all the logic and complex application code in the imperative/OO programming language, but extends the host language to represent XML as a first-class data type of this language. This extension of the type system is often accompanied by expressions that work on that new type, which can range from accessor methods to (limited) declarative querying possibilities. The XML type system support is often aligned with the properties of the host language type system, thus reducing the mismatch, but also limiting the compatibility with W3C standards (e.g., XML Schema not being implemented). The tighter integration with the host programming language increases programmer productivity, without restricting the flexibility as much as schema-driven code generation.

This second class can again be broken down into subclasses: XML as a native programming language type, XML as a native ORDBMS database type and query capabilities inside the programming language.

Adopting *XML as a native programming language data type* is an approach that has been taken especially by web-oriented programming languages such as Javascript/Ecmascript and PHP, because using XML interfaces was not considered sufficient any more. XML is a first-rate data type that can be constructed inside the program or read from a file. Instances of this XML data type can be accessed by functions that incorporate a subset of path navigation with a syntax that resembles the access to a field or a class member, e.g., `x.balance` to access the balance child of the XML variable x in XML extension of Javascript, E4X. While the mismatch to the host language is being reduced, a mismatch to W3C XML standards is often created. The imperative nature of the languages often limits the optimizability.

Using *XML as native DB data type* tries to achieve similar effects in the relational/object-relation database scenario, since there also exist impedance mismatches between the relational space and the XML space. Adding XML as a column type provides one possible solution to store XML in relational database system. On the language side, SQL/XML [3] blends SQL (as a relational query language) with XQuery as an XML query language by providing methods to map from one data model to the other, and embed XQuery expressions into SQL. This approach takes advantage of the DBMS infrastructure including triggers, transactional support, scalability, clustering, and reliability. Since both languages are declarative and there is mapping between the data models, global optimization over both XML and relational expression is possible. SQL/XML is supported by the major database vendors, making it well-known and providing good tool support and documentation. The drawbacks include the high overhead of loading all the XML data into the database, which is not useful for temporary XML or small volumes of data, and the complexity and cost of running a database server. The combination of two different query languages with different syntax, semantics and data models hinders the productivity of developers.

Adding *query capabilities inside existing imperative language* takes imperative languages a step closer to the database and increases the productivity, flexibility and optimizability of an existing programming language. The most prominent example of including such capabilities is the LinQ extension of the Microsoft .NET framework, which is based on the COmega research prototype [4]. On top of XML data type extensions and basic accessors eventwell (as described in the previous cases) and similar extension for relation data, LinQ provides query capabilities over all supported data models. These query capabilities come in the form of explicit query operators similar to relational algebra including collection-oriented selection and projection, joins, grouping etc., that work on all data types. On top of these operations, LinQ provides declarative queries similar in style to the SQL Select-From-Where. LinQ provides a good integration of the different data models and with the rest of the language, including the libraries, and yields a high productivity for developers familiar with .NET. Significant drawbacks are the lack of support for typed XML, the limited scope of static analysis, and the dominance of imperative constructs in the language, making database-style optimizations like lazy evaluation, streaming and indexing hard to do automatically.

The third class, *XML-oriented programming languages*, handles all application logic in an environment that is based purely on an XML data model and expressions working on this data model. Doing so represents a significant disruption from conventional languages, including giving up existing tools and design approaches. Compliance to W3C standard is high as is productivity related to XML processing. Many of these languages are, however, not designed to be general-purpose programming languages, lacking libraries and support for areas like GUI programming or numerical computations.

This third class can again be split into three subclasses: domain-specific languages, expression languages with an XML type system, and XML scripting.

Domain-specific languages utilize XML types and expression for a specific task that is often related to one particular use of XML. Clearly, such a language only works well within its intended design domain, but the close match to XML technologies and the restriction to relevant concepts facilitate high productivity. A very prominent example is BPEL, a workflow description language with XML syntax, which is used to "orchestrate" Web Services. It is transparent/agnostic to the actual XML data model, query language and expression language by just focusing on the control flow expressions needed in workflow environments. It provides a high abstraction level and many useful concepts needed in workflow environment. The separation of control flow and the actual expressions restricts the optimizability even in its intended domain.

Languages with an XML Type System are designed to handle operations on XML data in the most useable and efficient way, but not as general programming languages. The best known languages are XPath 2.0, XSLT 2.0 and XQuery. All of these languages work with a consistent data model (XDM) and provide high compliance to other W3C standards. The operations covered-include querying XML instances, construction of new XML instances (XQuery+XSLT), updating existing XML instances (XQuery) and full text search (XQuery). XSLT carries an XML syntax and uses a recursive pattern approach of document transformation language. It works best for small volumes of temporary XML data that are of unknown structure. XQuery uses a declarative style based in iterations and support for static type analysis. It works well for structured data and provides the facilities for good optimizability. Implementations exist for persistent and temporary data, large and small volumes of data in database systems, and as stand-alone expression engines. *XML Scripting* languages represent a recent development where a declarative XML type language (often XQuery) is extended with imperative constructs. The goal is to gain usability and a certain level of expressiveness by allowing (limited) side effects while still maintaining optimizability. Such a language could be used for at all tiers of XML software stacking, proving the potential for a truly XML-only application development. Major challenges in the long run will be developer acceptance and the building of good compilers/optimizers to actually achieve the potential performance benefits. Compared to imperative languages with query capabilities, the chances for optimizability are better, since the starting point is already well optimizable language. The most prominent example of such an XML scripting language is [1].

Key Applications

XML programming is relevant for almost all usage scenarios of XML that go beyond simple storage and retrieval. The large variety of use cases for XML is reflected in the different requirements for XML programming. A very common case is web application programming, where the browser, as the client layer, and the database server layer both contain and manipulate XML data. Web services, as a way of loosely coupling applications via XML messages, have become popular for information and application integration. An area in between these two use cases are mashups, were data and services from different sources are combined to form new applications.

Future Directions

A promising future direction is the integration of continuous/streaming XML data in the "regular" programming environments. Many data stream sources already produce their data as XML, and this data needs to be combined with other streaming or static data. Semantic querying, which is now handled using RDF/OWL, can be brought towards more standard XML programming models, thus enabling more effective ways for data integration. The interaction between XML integrity constraints and programming will become more important, as reasoning over programs and data (including verification) will improve the quality of applications. Automatic maintenance of a well-defined state (similar to relational integrity constraints in RDBMSs) will simplify programming XML applications in data-intensive environments. Native XML languages have the potential to change the architecture for many data-intensive applications which currently are built in a multi-tier fashion. Using the same data model and expressions allows-collapsing layers/tiers when needed, setting the main direction of partitioning not along the tiers, but along services and their respective data.

Cross-references

- ▶ Active XML
- ▶ AJAX
- ▶ Composed Services and WS-BPEL

- ▶ Entity Relationship Model
- ▶ Java Database Connectivity
- ▶ Open Database Connectivity
- ▶ SOAP
- ▶ Unified Modeling Language
- ▶ W3C
- ▶ Web Services
- ▶ XML
- ▶ XML Information Integration
- ▶ XML Integrity Constraints
- ▶ XML Parsing, SAX/DOM
- ▶ XML Process Definition Language
- ▶ XML Schema
- ▶ XML Stream Processing
- ▶ XPath/XQuery
- ▶ XQuery Full-Text
- ▶ XQuery Processors
- ▶ XSL/XSLT

Recommended Reading

1. Chamberlin D., Carey M.J., Fernandez M., Florescu D., Ghelli G., Kossmann D., Robie J., and Simeon J. XQueryP: an XML application development language. In Proc. XML 2006 Conference. 2006.
2. Florescu D. and Kossmann D. Programming for XML. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, p. 801.
3. Funderburk J.E., Malaika S., and Reinwald B. XML programming with SQL/XML and XQuery. IBM Syst. J., 41(4):642–665, 2002.
4. Meijer E., Schulte W., and Bierman G. Unifying tables, objects and documents. In Proc. Workshop on Declarative Programming in the Context of Languages, 2003.

XML Publish/Subscribe

Yanlei Diao[1], Michael J. Franklin[2]

[1]University of Massachusetts Amherst, MA, USA

[2]University of California-Berkeley, Berkeley, CA, USA

Synonyms

Selective XML dissemination; XML filtering; XML message brokering

Definition

As stated in the entry "Publish/Subscribe over Streams," publish/subscribe (pub/sub) is a many-to-many communication model that directs the flow of messages from senders to receivers based on receivers' data interests. In this model, publishers (i.e., senders) generate messages without knowing their receivers; subscribers (who are potential receivers) express their data interests, and are subsequently notified of the messages from a variety of publishers that match their interests.

XML publish/subscribe is a publish/subscribe model in which messages are encoded in XML and subscriptions are written in an XML query language such as a subset of XQuery 1.0. (In the context of XML pub/sub, "messages" and "documents" are often used exchangeably.)

In XML-based pub/sub systems, the message brokers that serve as central exchange points between publishers and subscribers are called XML message brokers.

Historical Background

As described in "Publish/Subscribe over Streams," XML pub/sub has emerged as a solution for loose coupling of disparate systems at both the communication and content levels. At the communication level, pub/sub enables loose coupling of senders and receivers based on the receivers' data interests. With respect to content, XML can be used to encode data in a generic format that senders and receivers agree upon due to its flexible, extensible, and self-describing nature; this way, senders and receivers can exchange data without knowing the data representation in individual systems.

Foundations

XML pub/sub raises many technical challenges due to the requirements of large-scale publish/subscribe (as described in the entry "Publish/Subscribe over Streams"), the volume of XML data, and the complexity of XML processing. Among all, two challenges are highlighted below:

- *XML stream processing*. In XML-based pub/sub systems, XML data continuously arrives from external sources, and user subscriptions, stored as continuous queries in a message broker, are evaluated every time when a new data item is received. Such XML query processing is referred to as *stream-based*. In cases where incoming messages are large, stream-based processing also needs to start before the messages are completely received in order to reduce the delay in producing results.

- *Handling simultaneous XML queries*. Compared to XML stream systems, a distinguishing aspect of XML pub/sub systems lies in the size of their query populations. All the queries stored in an XML message broker are simultaneously active and need to be matched efficiently with each incoming XML message. While multi-query processing has been studied for relational databases and relational streams, the complexity of XML processing, including structure matching, predicate evaluation, and transformation, requires new techniques for efficient multi-query processing in this new context.

Foundation of XML Stream Processing for Publish/Subscribe

Event-based parsing. Since XML messages can be used to encode data of immense sizes (e.g., the equivalent of a database's worth of data), efficient query processing requires fine-grained processing upon arrival of small constituent pieces of XML data. Such fine-grained XML query processing can be implemented via an event-based API. A well known example is the *SAX* interface that reports low-level parsing events incrementally to the calling application. Figure 1 shows an example of how a SAX interface breaks down the structure of the sample XML document into a linear sequence of events. "Start document" and "end document" events mark the beginning and the end of the parse of a document. A "start element" event carries information such as the name of the element and its attributes. A "characters" event reports a text string residing between two XML tags. An "end element" event corresponds to an earlier "start element" event and marks the close of that element. To use the SAX interface, the application receiving the events must implement handlers to respond to different events. In particular, stream-based XML processors can use these handlers to implement event-driven processing.

An automata-based approach. A popular approach to event-driven XML query processing is to adopt some form of *finite automaton* to represent path expressions [1,13]. This approach is based on the observation that a path expression (a small, common subset of XQuery) written using the axes ("/," "//") and node tests (element name or "*") can be transformed into a regular expression. Thus, there exists a finite automaton that accepts the language described by such a path expression [11]. Such an automaton can be created by mapping the location steps of the path expression to the automaton states. Figure 2 shows an example automaton created for a simple path expression, where the two concentric circles represent the accepting state. When arriving, XML messages are parsed with an event-based parser, the events raised during parsing are used to drive the execution of the automaton. In particular, "start element" events drive the automaton through its various transitions, and "end element" events cause the execution to backtrack to the previous states. A path expression is said to match a message if during parsing, the accepting state for that path is reached.

XML Filtering

In XML filtering systems, user queries are written using path expressions that can specify constraints over both *structure* and *content* of XML messages. These queries are applied to individual messages (hence, stateless processing). Query answers are "yes" or "no" – computing only Boolean results in XML filtering avoids complex issues of XML query processing related to multiple matches such as ordering and duplicates, hence enabling simplified, high-performance query processing.

XFilter [1], the earliest XML filtering system, considers matching of the structure of path expressions and explores *indexing* for efficient filtering. It builds a dynamic index over the queries and uses the parsing events of a document to probe the query index. This approach quickly results in a smaller set of queries that can be potentially matched by a message, hence avoiding processing queries for which the message is irrelevant. Built over the states of query automata, the dynamic index identifies the states that the execution of these automata is attempting to match at a particular moment. The content of the index constantly changes as parsing events drive the execution of the automata.

YFilter [4] significantly improves over XFilter in two aspects. By creating a separate automaton per query, XFilter can perform redundant work when significant commonalities exist among queries. Based on this insight, YFilter supports *sharing* in processing by using a combined automaton to represent all path expressions; this automaton naturally supports shared representation of all common prefixes among path expressions. Furthermore, the combined automaton is implemented as a *Nondeterministic Finite Automaton* (NFA) with two practical advantages: i) a relatively small number of states required to represent even

```
<?xml version="1.0" ?>
<report>
  <section id="intro" difficulty="easy">
      <title>Pub/Sub</title>
      <section difficulty="easy">
          <figure source="g1.jpg">
             <title>XML Processing</title>
          </figure>
      </section>
      <figure source="g2.jpg">
          <title>Scalability</title>
      </figure>
  </section>
</report>
```

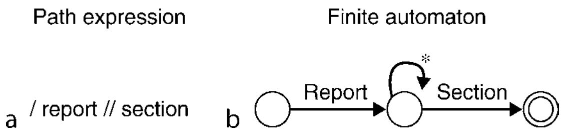
```
<Start Document
<Start Element:      report
<Start Element:      section
<Start Element:      title
  characters:        Pub/Sub
>End Element:        title
<Start Element:      section
<Start Element:      figure
<Start Element:      title
  Characters:        XML processing
>End Element:        title
>End Element:        figure
>End Element:        section
  ...
>End Element:        section
>End Element:        report
>End Document
```

XML Publish/Subscribe. Figure 1. An example XML document and results of SAX parsing.

large numbers of path expressions and complex queries (e.g., with multiple wildcards "*" and descendent axes "//"), and ii) incremental maintenance of the automaton upon query updates. Results of YFilter show that its shared path matching approach can offer order-of-magnitude performance improvements over XFilter while requiring only a small maintenance cost.

Structure matching that XFilter considers is one part of the XML filtering problem; another significant part is the evaluation of predicates that are applied to path expressions (e.g., addressing attributes of elements, text data of elements, or even other path expressions) for additional filtering. Since shared structure matching has been shown to be crucial for performance, YFilter supports predicate evaluation using *post-processing* of path matches after shared structuring matching, and further leverages relational processing in such post-processing.

Figure 3 shows two example queries and their representation in YFilter. Q1 contains a root element "/nitf" with two nested paths applied to it. YFilter decomposes the query into two linear paths "/nitf/head/pubdata[@edition.area="SF"]," and "/nitf//tobject.subject[@tobject.subject.type ="Stock"]." The structural part of these paths is represented using the NFA with the common prefix "/nitf" shared between the paths. The accepting states of these paths are state 4 and state 6, where the network of operators (represented as boxes) for the remainder of Q1 starts. At the bottom of the network, there is a selection (σ) operator above each accepting state to handle the value-based predicate in the corresponding path. To handle the correlation between the two paths (e.g., the requirement that it should be the same "nitf" element that makes these two paths evaluate to true), YFilter applies a join (▷◁) operator after the two selections. Q2 is similar to Q1 and hence shares a significant portion of its representation with Q1.



XML Publish/Subscribe. Figure 2. A path expression and its corresponding finite automaton.

Index-Filter [2] builds indexes over both queries and streaming data. The index over data speeds up the processing of large documents while its construction overhead may penalize the processing of small ones. Results of a comparison between Index-Filter and YFilter show that Index-Filter works better when the number of queries is small or the XML document is large, whereas YFilter's approach is more effective for large numbers of queries and short documents.

XMLTK [8] converts YFilter's NFA to a *Deterministic Finite Automaton* (DFA) to further improve the filtering speed. A straightforward conversion could theoretically result in severe scalability problems due to an explosion in the number of states. This work, however, shows that such explosion can be avoided in many cases by using lazy construction of the DFA and placing certain restrictions on the types of documents and queries supported (when suitable for the application). *XPush* [9] further explores a pushdown automaton for shared processing of both structure and value-based constraints. Such an automaton can provide high efficiency when wildcard ("*") and descendant ("//") operators are rare in queries and periodic reconstruction of the automaton can be used.

FiST [14] views path expressions with predicates as twig patterns and considers ordered twig pattern matching. For such ordered patterns, it transforms the patterns as well as XML documents intro sequences using Prufer's method. This approach allows holistic matching of ordered twig patterns, as opposed to matching individual paths and then merging their matches during post-processing in YFilter (which works for both ordered and unordered patterns), resulting in significant performance improvements over YFilter.

XML Filtering and Transformation

XML filtering solutions presented above have not addressed the transformation of XML messages for customized result delivery, which is an important feature in XML-based data exchange and dissemination. For XML transformation, queries are written using a richer subset of XQuery, e.g., the *For-Where-Return* expressions.

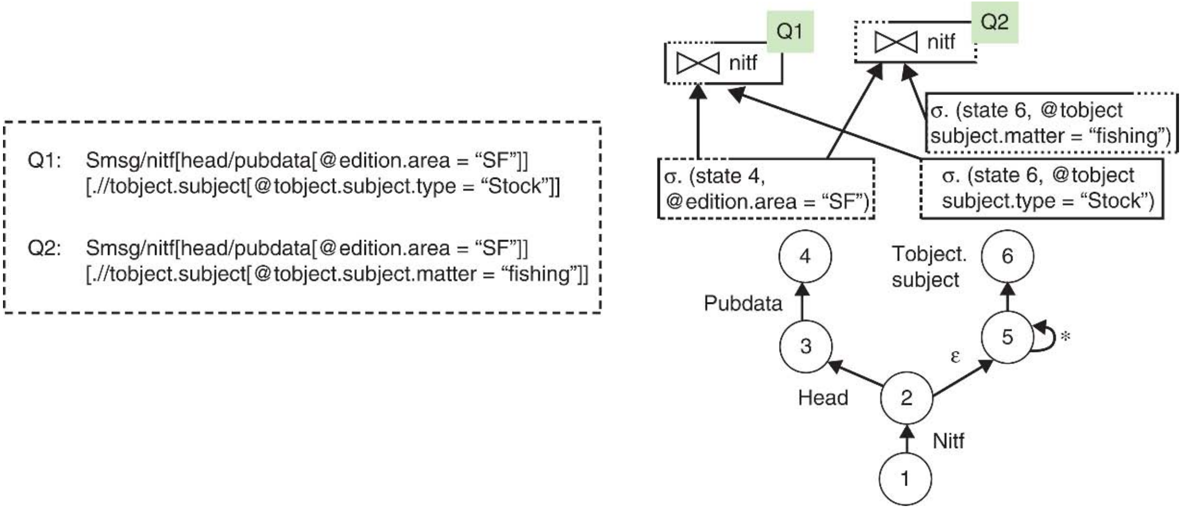
To support efficient transformation for many simultaneous queries, YFilter [5] further extends its NFA-based framework and develops alternatives for building transformation functionality on top of shared path matching. It explores the tradeoff between shared path matching and post-processing for result customization, by varying the extent to which they push paths from the For-Where-Return expressions into the shared path matching engine. To further reduce the remarkable cost of post-processing of individual queries, it employs provably correct optimizations based on query and DTD (if available) inspection to eliminate unnecessary operations and choose more efficient operator implementations for post-processing. Moreover, it provides techniques for also sharing post-processing across multiple queries, similar to those in continuous query processing over relational streams.

Stateful XML Publish/Subscribe

In [10], efficient processing of large numbers of continuous inter-document queries over XML Streams (hence, stateful processing) is addressed. The key idea that it exploits is to dissert query specifications into tree patterns evaluated within individual documents and value-based joins preformed across documents. While employing existing path evaluation techniques (e.g., YFilter) for tree pattern evaluation, it proposes a scalable join processor that leverages relational joins and view materialization to share join processing among queries.

XML Routing

As described in "publish/subscriber over streams" distributed pub/sub systems need to efficiently route messages from their publishing sites to the brokers hosting relevant queries for complete query processing. While the concept of content-based routing and many architectural solutions can be applied in XML-based



Q1: Smsg/nitf[head/pubdata[@edition.area = "SF"]]
[.//tobject.subject[@tobject.subject.type = "Stock"]]

Q2: Smsg/nitf[head/pubdata[@edition.area = "SF"]]
[.//tobject.subject[@tobject.subject.matter = "fishing"]]

Q1
⋈ nitf
Q2
⋈ nitf
σ. (state 6, @tobject subject.matter = "fishing")
σ. (state 4, @edition.area = "SF")
σ. (state 6, @tobject subject.type = "Stock")
4
Tobject. subject
6
Pubdata
3
5
*
ε
Head
2
Nitf
1

XML Publish/Subscribe. Figure 3. Example queries and their representation in YFilter.

pub/sub systems, routing of XML messages raises additional challenges due to the increased complexity of XML query processing.

Aggregating user subscriptions into compact routing specifications is a core problem in XML routing.

Chan et al. [3] aggregate tree pattern subscriptions into a smaller set of generalized tree patterns such that (i) a given space constraint on the total size of the subscriptions is met, and (ii) the loss in precision (due to aggregation) during document filtering is minimized (i.e., a constrained optimization problem). The solution employs tree-pattern containment and minimization algorithms and makes effective use of document-distribution statistics to compute a precise set of aggregate tree patterns within the allotted space budget.

ONYX [6] leverages YFilter technology for efficient routing of XML messages. While subscriptions can be written using For-Where-Return expressions, the routing specification for each output link at a broker consists of a *disjunctive normal form* (DNF) of absolute linear path expressions, which generalizes the subscriptions reachable from that link while avoiding expensive path operations. These routing specifications can be efficiently evaluated using YFilter, even with some work shared with complete query processing at the same broker. To boost the effectiveness of routing, ONYX also partitions the XQuery-based subscriptions among brokers based on exclusiveness of data interests.

Gong et al. [7] introduce Bloom filters into XML routing. The proposed approach takes a path query as a string and maps all query strings into a Bloom filter using hash functions. The routing table is comprised of multiple Bloom filters. Each incoming XML message is parsed into a set of candidate paths that are mapped using the same hash functions to compare with the routing table. This approach can filter XML messages efficiently with relatively small numbers of false positives. Its benefits in efficiency and routing table maintenance are significant when the number of queries is large.

Key Applications

Personalized News Delivery. News providers are adopting XML-based formats (e.g., *News Industry Text Format* [12]) to publish news articles online. Given articles marked up with XML tags, a pub/sub-based news delivery service allows users to express a wide variety of interests as well as to specify which portions of the relevant articles (e.g., title and abstract only) should be returned. *Really Simple Syndication* (RSS) provides similar yet simpler services based on URL-and/or keyword-based preferences.

Application Integration. XML publish/subscribe has been widely used to integrate disparate, independently-developed applications into new services. Messages exchanged between applications (e.g., purchase orders and invoices) are encoded in a generic XML format. Senders publish messages in this format. Receivers subscribe with specifications on the relevant messages and the transformation of relevant messages into an internal data format for further processing.

Mobile services. In mobile applications, clients run a multitude of operating systems and can be located anywhere. Information exchange between information providers and a huge, dynamic collection of heterogeneous clients has to rely on open, XML-based technologies and can be further facilitated by pub/sub technology including filtering and transformation for adaptation to wireless devices.

Data Sets

XML data repository at University of Washington, http://www.cs.washington.edu/research/xmldatasets/ and Niagara experimental data, http://www.cs.wisc.edu/niagara/data.html.

URL to Code

YFilter is an XML filtering engine that processes simultaneous queries (written in a subset of XPath 1.0) against streaming XML messages in a shared fashion. For each XML message, it returns a result for every matched query (http://yfilter.cs.umass.edu/).

ToXgene is a template-based generator for large, consistent collections of synthetic XML documents (http://www.cs.toronto.edu/tox/toxgene/).

XMark is an XQuery benchmark suite to analyze the capabilities of an XML database (http://www.xml-benchmark.org/).

Cross-references

- ▶ Continuous Queries
- ▶ Publish/Subscribe Over Streams
- ▶ XML
- ▶ XML Document
- ▶ XML Parsing
- ▶ XML Schema
- ▶ XML Stream Processing
- ▶ XPath/XQuery

Recommended Reading

1. Altinel M. and Franklin M.J. Efficient filtering of XML documents for selective dissemination of information. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 53–64.
2. Bruno N., Gravano L., Doudas N., and Srivastava D. Navigation- vs. Index-based XML Multi-query processing. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 139–150.
3. Chan C.Y., Fan W., Felber P., Garofalakis M.N., and Rastogi R. Tree pattern aggregation for scalable XML data dissemination. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 826–837.
4. Diao Y., Altinel M., Zhang H., Franklin M.J., and Fischer P.M. Path sharing and predicate evaluation for high-performance XML filtering. ACM Trans. Database Syst., 28, (4)467–516, 2003.
5. Diao Y. and Franklin M.J. Query processing for high-volume XML message brokering. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 261–272.
6. Diao Y., Rizvi S., and Franklin M.J. Towards an Internet-Scale XML dissemination service. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 612–623.
7. Gong X., Qian W., Yan Y., and Zhou A. Bloom filter-based XML packets filtering for millions of path queries. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 890–901.
8. Green T.J., Gupta A., Miklau G., Onizuka M., Suciu D. Processing XML streams with deterministic automata and stream indexes. ACM Trans. Databases, 29(4):752–788, 2004.
9. Gupta A.K. and Suciu D. Streaming processing of XPath queries with predicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 419–430.
10. Hong M., Demers A.J., Gehrke J., Koch C., Riedewald M., and White W.M. Massively multi-query join processing in publish/subscribe systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 761–772.
11. Hopcroft J.E. and Ullman J.D. Introduction to Automata Theory, Languages and Computation. Addition-Wesley, Boston, MA, 1979.
12. Internal Press Telecommunications Council. News Industry Text Format. Available online at: http://www.nitf.org/, 2004.
13. Ives Z.G., Halevy and A.Y., Weld D.S. An XML query engine for network-bound data. VLDB J., 11(4): 380–402, 2002.
14. Kwon J., Rao P., Moon B., and Lee S. FiST: scalable XML document filtering by sequencing twig patterns. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 217–228.

XML Publishing

ZACHARY IVES
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

XML export

Definition

XML Publishing typically refers to the creation of XML output (either in the form of a character stream or file) from a relational DBMS. XML Publishing typically must handle three issues: converting an XML query or view definition into a corresponding SQL query; encoding hierarchy in the SQL data; and generating tags around the encoded hierarchical data. Since in some cases the relational data may have originated from XML, the topics of *XML Storage* and XML Publishing are closely related and often addressed simultaneously.

Historical Background

The topic of XML Publishing arose very soon after database researchers suggested a connection between XML and semi-structured data [5], a topic that had previously been studied in the database literature [1,2,8]. Initially the assumption was that XML databases would probably need to resemble those for semi-structured data in order to get good performance. Florescu and Kossmann [11] showed that storing XML in a relation database could be more efficient than storing it in a specialized semi-structured DBMS. Concurrently, Deutsch et al. were exploring hybrid relational/semi-structured storage in STORED [7]. Soon after, the commercial DBMS vendors became interested in adding XML capabilities to their products. The most influential developments in that area came from IBM Almaden's XPERANTO research project [4,14], which formed the core of Shanmugasundaram's thesis work [9]. Today, most commercial DBMSs use a combination of all of the aforementioned techniques: storing XML data in relations, storing hybrid relational/semi-structured data, and storing XML in a hierarchical format resembling semi-structured data.

Foundations

As every student of a database class knows, a good relational database schema is in first normal form (1NF): separate concepts and multi-valued attributes are split into separate tables, and taken together the tables can be visualized as a graph-structured Entity-Relationship Diagram. In contrast, XML data are fundamentall *not* in 1NF: an XML document has a single root node and *hierarchically* encodes data.

Thus, there are two main challenges in XML Publishing: first, taking queries or templates describing XML output and mapping them into operations over 1NF tables; and second, efficiently computing and adding XML tags to the data being queried. Typically, the latter problem is addressed with two separate modules, and hence the XML Publishing problem consists of three steps:

1. *Converting queries*, which typically are posed in a language other than SQL, into an internal representation from which SQL can be constructed.
2. *Composition and optimization* of the resulting queries, such that redundant work is minimized.
3. *Adding tags*, which is often done in a postprocessing step outside the DBMS.

Each of these topics is addressed in the remainder of this section. The discussion primarily focuses upon the XPERANTO and SilkRoute systems, which established many of the basic algorithms used in XML Publishing.

Converting Queries

The first issue in XML Publishing is that of taking the specification of the XML output, and converting it into some form from which SQL can be constructed. A number of different forms have been proposed for specifying the XML output:

Proprietary XML template languages, such as IBM DB2's Document Access Definition (DAD) or SQL Server's XML Data Reduced (XDR), which essentially provide a scheme for annotating XML templates with SQL queries that produce content.

SQL extensions, such as the SQL/XML standard [12], which extend SQL with new functions to add tags around relational attributes and intermediate table results.

Relational-XML query languages, such as the RXL language in early versions of SilkRoute [10], which provide a language that creates hierarchical XML content using relational tables. RXL resembles a combination of Datalog and an XML query language (specifically, XML-QL [6]), and it can be used to define XML views that can be queried using a standard XML query language.

XML query languages with built-in XML views of relations, an approach first proposed in XPERANTO [3], where each relational table is mapped into a virtual XML document and a standard XML language (like XPath or XQuery) can be used to query the relations and even to define XML views.

Over time, the research community has come to the consensus that the last approach offers the best set of trade-offs, as it offers a single compositional language for all XML queries over the data in the RDBMS. The commercial market has settled on a combination of this same approach, for XML-centric users, plus SQL extensions for relation-centric users. Here the discussion assumes an XML-centric perspective, with XQuery as the language, since it is representative.

XML Publishing systems generally only attempt to tackle a subset of the full XQuery specification, focusing on the portions that are particularly amenable to execution in a relational DBMS. There are many commonalities between basic XQueries and SQL: selections, projections, and joins can be similarly expressed in both of these languages, views can be defined, and queries can be posed over data in a way that is agnostic as to whether the data comes from a view or a raw source. The two main challenges in conversion lie in the input to the XQuery – where XPaths must be matched against relations or XML views defined over relations – and in creating the hierarchical nested XQuery output.

XPath matching over built-in views of relations is, of course, trivial, as each relation attribute maps to an XML element or attribute in the XPath. Conversion gets significantly more complex when the XPaths are over XML views constructed over the original base relations: this poses the problem of XML query composition, where the DBMS should not have to compute each composed view separately, but rather should statically *unfold* them.

Composition and Optimization

As a means of composing queries, SilkRoute takes each XQuery definition and creates an internal *view forest* representation that describes the structure of the XML document; it converts an XQuery into an canonical representation called XQueryCore, and its algorithms compose XQueryCore operations over an input document. XPERANTO, based in large part on IBM's DB2, performs very similar operations, but starts by creating an internal representation of each query block called in a model XQGM, then executes a series of rewrite rules to merge the blocks by merging steps that create hierarchy in a view's output with steps that traverse that hierarchy in a subsequent query's input.

The resulting SQL is often highly complex and repetitive, with many SQL blocks being unioned together: each level of the XML hierarchy may require a separate SQL block, and each view composition step in the optimization process may create multiple SQL query blocks (an XPath over a view may match over multiple relational attributes from the base data).

Thus, optimizing the resulting SQL becomes of high importance. XML publishing systems typically convert from XQuery to SQL, not to actual executable query plans, so they can only do a limited amount of optimization on their own. Here they use a significant number of heuristics [10,14] to choose the best form for the sets of SQL queries, and rely on the RDBMS's cost-based optimizer to more efficiently optimize the queries. (Recently, commercial vendors have invested significant effort in improving their cost-based query optimizers for executing the types of queries output by XML Publishing systems.)

Adding Tags

The final step in XML Publishing is that of adding tags to the data from the relational query. There have been two main approaches to this task: *in-engine*, relying on SQL functions to add the tags; and *outside the DBMS*, relying on an external middleware module to add the tags. Each is briefly described.

In-engine Tagging In [14], the authors proposed a method for adding XML tags using the SQL functions XMLELEMENT, XMLATTRIBUTE, and XMLAGG: each of these takes a relational attribute or tuple and converts it into a partial XML result, encoded in a CLOB datatype. Naturally, the first two create an element or attribute tag, respectively, around a data value. The third function, XMLAGG, functions as an SQL aggregate function: it takes an SQL row set (possibly including XMLELEMENT or XMLATTRIBUTE values) and outputs a tuple with a CLOB containing the XML serialization of the rowset's content. The drawback to the in-engine tagging approach is that CLOBs are not always handled efficiently, as they are typically stored separately from the tuples with which they are associated. Hence, when this method was incorporated into the SQL/XML standard, a new XML datatype was proposed that could be implemented in a manner more efficient than the CLOB.

Tagging Middleware A more common approach is to simply a tuple representation of the XML document tree/forest within the SQL engine, and to convert this tuple stream into XML content externally, using a separate tagging module [9,14]. There are two common techniques for encoding hierarchy in tuple streams: *outer join* and *outer union*. Suppose there are relationaltables in Fig. 1a, encoding an element *a* and its child element *b*, and one wants to encode their output as the following XML:

```
<a id="1" val="123"><b id="2"
val="234"/><b id="3" val="456"/></a>
<a id="2" val="789"><b id="4"
val="832"/></a>
```

Outer Join. In SilkRoute, tuples are generated using outerjoins between parent and child relations. The tuple stream will be sorted in a way that allowsthe tagger to hold the last tuple, not all data (see Fig. 1b). The "parent" portion of the tree (the *a* element) will appear in both tuples, and the external XML tagger will compare consecutive tuples in the tuple stream to determine the leftmost position where a value changed – from which it can determine what portion of the XML hierarchy has changed.

Outer union. In contrast, XPERANTO uses a so-called *sorted outer union* representation (Fig. 1c), which substitutes null values for repeated copies of values (but not keys). Its approach relies on two characteristics of IBM's DB2 engine that are shared by some but not all other systems: (i) null values can be very efficiently encoded and processed, meaning that the query is more efficient; (ii) null values *sort high*, i.e., DB2's considers null values to have a sort value greater than any non-null value. The tagger simply needs to look for the first non-null attribute to determine what portion of a the XML to emit.

Key Applications

XML publishing has become a key capability in the relational database arena, as increasingly data

a

id	*val*
1	123
2	789

b

id	*pid*	*val*
2	1	234
3	1	456
4	2	832

a Relations for elements a and b

id_a	*val_a*	*id_b*	*val_b*
1	123	2	234
1	123	3	456
2	789	4	832

b SilkRoute

id_a	*val_a*	*id_b*	*val_b*
1	123	2	234
1	–	3	456
2	789	4	832

c XPERANTO

XML Publishing. Figure 1. Tuple representations of XML data.

interchange mechanisms and Web services are being built using XML data from an RDBMS. Today the "Big Three" commercial DBMS vendors all use some techniques for XML Publishing, and it is likely that smaller DBMSs, including those in open source, will gradually incorporate them as well.

Cross-references

- ▶ Approximate XML Querying
- ▶ XML Storage
- ▶ XML Views

Recommended Reading

1. Abiteboul S., Quass D., McHugh J., Widom J., and Winer J.L. The Lorel query language for semistructured data. In Int. J. Digit. Libr., 1(1):68–88, 1997.
2. Buneman P., Davidson S.B., Fernandez M.F., and Suciu D. Adding structure to unstructured data. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 336–350.
3. Carey M.J., Florescu D., Ives Z.G., Lu Y., Shanmugasundaram J., Shekita E., and Subramanian S. XPERANTO: publishing object-relational data as XML. In Proc. 3rd Int. Workshop on the World Wide Web and Databases, 2000, pp. 105–110.
4. Carey M., Kiernan J., Shanmugasundaram J., Shekita E., and Subramanian S. XPERANTO: a middleware for publishing object-relational data as XML documents. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 646–648.
5. Deutsch A., Fernández M.F., Florescu D., Levy A.Y., and Suciu D. XML-QL. In Proc. The Query Languages Workshop, 1998.
6. Deutsch A., Fernandez M.F., Florescu D., Levy A., and Suciu D. A query language for XML. Comp. Networks, 31(11–16):1155–1169, 1999.
7. Deutsch A., Fernandez M.F., and Suciu D. Storing semistructured data with STORED. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 431–442.
8. Fernandez M.F., Florescu D., Kang J., Levy A.Y., and Suciu D. Catching the boat with strudel: experiences with a web-site management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 414–425.
9. Fernandez M.F., Kadiyska Y., Suciu D., Morishima A., and Tan W.C. SilkRoute: A framework for publishing relational data in XML. ACM Trans. Database Syst., 27 (4):438–493, 2002.
10. Fernandez M., Tan W.C., and Suciu D. SilkRoute: trading between relations and XML. Comp. Networks, 33 (1–6):723–745, 2000.
11. Florescu D. and Kossmann D. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. Tech. Rep. 3684, INRIA, 1999.
12. ISO/IEC 9075-14:2003 Information technology – Database languages – SQL – Part 14: XML-Related Specifications (SQL/XML).
13. Shanmugasundaram J. Bridging Relational Technology and XML. Ph.D. thesis, University of Wisconsin-Madison, 2001.
14. Shanmugasundaram J., Shekita E.J., Barr R., Carey M.J., Lindsay B.G., Pirahesh H., and Reinwald B. Efficiently publishing relational data as XML documents. VLDB J., 10(2–3):133–154, 2001.

XML Retrieval

Mounia Lalmas[1], Andrew Trotman[2]
[1]Queen Mary, University of London, London, UK
[2]University of Otago, Dunedin, New Zealand

Synonyms

Structured document retrieval; Structured text retrieval; Focused retrieval; Content-oriented XML retrieval

Definition

Text documents often contain a mixture of structured and unstructured content. One way to format this mixed content is according to the adopted W3C standard for information repositories and exchanges, the *eXtensible Mark-up Language* (XML). In contrast to HTML, which is mainly layout-oriented, XML follows the fundamental concept of separating the *logical structure* of a document from its layout. This logical document structure can be exploited to allow a more focused sub-document retrieval.

XML retrieval breaks away from the traditional retrieval unit of a document as a single large (text) block and aims to implement *focused retrieval* strategies aiming at returning document components, i.e., XML elements, instead of whole documents in response to a user query. This focused retrieval strategy is believed to be of particular benefit for information repositories containing long documents, or documents covering a wide variety of topics (e.g., books, user manuals, legal documents), where the user's effort to locate relevant content within a document can be reduced by directing them to the most relevant parts of the document.

Historical Background

Managing the enormous amount of information available on the web, in digital libraries, in intranets, and so on, requires efficient and effective indexing and retrieval methods. Although this information is available in different forms (text, image, speech, audio, video etc), it remains widely prevalent in text form. Textual information can be broadly classified into two categories, structured and unstructured.

Unstructured information has no fixed pre-defined format, and is typically expressed in natural language. For instance, much of the information available on the web is unstructured. Although this information is mostly formatted in HTML, thus imposing some structure on the text, the structure is only for presentation purposes and carries essentially no semantic meaning. Correct nesting of the HTML structure (that is, to form an unambiguous document logical structure) is not imposed. Accessing unstructured information is through flexible but mostly simplistic means, such as a simple keyword matching or bag of words techniques.

Structured information is usually represented using XML, a mark-up language similar to HTML except that it imposes a rigorous structure on the document. Moreover, unlike HTML, XML tags are used to specify semantic information about the stored content and not the presentation. A document correctly marked-up in XML has a fixed document structure in which semantically separate document parts are explicitly identified – and this can be exploited to provide powerful and flexible access to textual information.

XML has been accepted by the computing community as a standard for document mark-up and an increasing number of documents are being made available in this format. As a consequence numerous techniques are being applied to access XML documents. The use of XML has generated a wealth of issues that are being addressed by both the database and information retrieval communities [3]. This entry is concerned with content-oriented XML retrieval [2,5] as investigated by the information retrieval community.

Retrieval approaches for structured text (marked-up in XML-like languages such as SGML) were first proposed in the late 1980s. In the late 1990s, the interest in structured text retrieval grew due to the introduction of XML in 1998. Research on XML information retrieval was first coordinated in 2002 with the founding of the Initiative for the Evaluation of XML Retrieval (INEX). INEX provides a forum for the evaluation of information retrieval approaches specifically developed for XML retrieval.

Foundations

Within INEX, the aim of an XML retrieval system is "to exploit the logical structure of XML documents to determine the best document components, i.e., best XML elements, to return as answers to queries" [7]. Query languages have been developed in order to allow users to specify the nature of these best components. Indexing strategies have been developed to obtain a representation not only of the content of XML documents, but their structure. Ranking strategies have been developed to determine the best elements for a given query.

Query Languages

In XML retrieval, the logical document structure is additionally used to determine which document components are most meaningful to return as query answers. With appropriate query languages, this structure can be specified by the user. For example, "I want a paragraph discussing penguins near to a picture labeled Otago Peninsula." Here, "penguins" and "Otago Peninsula" specify *content* (textual) constraints, whereas "paragraph" and "picture" specify *structural* constraints on the retrieval units.

Query languages for XML retrieval can be classified into content-only and content-and-structure query languages. Content-only queries have historically been used as the standard form of input in information retrieval. They are suitable for XML search scenarios where the user does not know (or is not concerned with) the logical structure of a document. Although only the content aspect of an information need can be specified, XML retrieval systems must still determine the best granularity of elements to return to the user.

Content-and-structure queries provide a means for users to specify conditions referring both to the *content* and the *structure* of the sought elements. These conditions may refer to the content of specific elements (e.g., the returned element must contain a section about a particular topic), or may specify the type of the requested answer elements (e.g., sections should be retrieved). There are three main categories of content-and-structure query languages [1]:

1. Tag-based queries allow users to annotate words in the query with a single tag name that specifies the type of results to be returned. For example the query "section:penguins" requests section elements on "penguins."
2. Path-based queries are based upon the syntax of XPath. They encapsulate the document structure in the query. An example the NEXI language is: "//document[about(.,Otago Peninsula)]//section [about(.//title, penguins)]." This query asks for sections that have a title about "penguins," and that are contained in a document about "Otago Peninsula."
3. Clause-based queries use nested clauses to express information needs, in a similar way to SQL. The most prominent clause-based language for XML retrieval is XQuery. A second example is XQuery Full-Text, which extends XQuery with text search predicates such as proximity searching and relevance ranking.

The complexity and the expressiveness of content-and-structure query languages increases from tag-based to clause-based queries. This increase in expressiveness and complexity often means that content-and-structure queries are viewed as too difficult for end users (because, they must, for example, be intimate with the document structure). Nonetheless they can be very useful for expert users in specialized scenarios, and also have been used as an intermediate between a graphical query language (such as Bricks [12]) and an XML search engine.

Indexing Strategies

Classical indexing methods in information retrieval make use of term statistics to capture the importance of a term in a document; and consequently for discriminating between relevant and non-relevant content. Indexing methods for XML retrieval require similar term statistics, but for each element. In XML retrieval there are no a priori fixed retrieval units. The whole document, one of its sections, or a single paragraph within a section, all constitute potential answers to a single query. The simplest approach to allow the retrieval of elements at any level of granularity is to index each element separately (as a separate document in the traditional sense). In this case, term statistics for each element are calculated from the text of the element and all its descendants.

An alternative is to derive the term statistics through the aggregation of term statistics of the element own text, and those of each of its children. A second alternative is to only index leaf elements and to score non-leaf elements through propagation of the score of their children elements. Both alternatives can include additional parameters incorporating, for instance, element relationships or special behavior for some element types.

It is not uncommon to discard elements smaller than some given threshold. A single italicized word, for example, may not be a meaningful retrieval unit. A related strategy, selective indexing, involves building separate indexes for those element types previously seen to carry relevant information (sections, subsections, etc, but not italics, bold, etc.). With selective indexing the results from each index must be merged to provide a single ranked result list across all element types.

It is not yet clear which indexing strategy is the best. The best approach appears to depend on the collection, the types of elements (i.e., the DTD) and their relationships. In addition, the choice of the indexing strategy currently has an effect on the ranking strategy. More details about indexing strategies can be found in the entry on Indexing Units.

Ranking Strategies

XML documents are made of XML elements, which define the logical document structure. Thus sophisticated ranking strategies can be developed to exploit the various additional (structural) evidence not seen in unstructured (flat) text documents.

Element Scoring Many of the retrieval models developed for flat document retrieval have been adapted for XML retrieval. These models have been used to estimate the relevance of an element based on the evidence associated with the element only. This is done by a scoring function based, for instance, on the vector space, BM25, the language model, and so on. They are typically adapted to incorporate XML-specific features. As an illustration, a scoring function based on language models [6] is described next:

Given a query $q = t_1,\ldots,t_n$ made of n terms, an element e and its corresponding element language model θe, the element *e* is ranked using the following scoring function:

$$P(e|q) \propto P(e) \times P(q|\theta_e)$$

where *P(e)* is the prior probability of relevance for element *e* and *P(q|θe)* is the probability of the query *q* being "generated" by the element language model and is calculated as follows:

$$P(t_1,\ldots,t_n|\theta_e) = \prod_{i=1}^{n} \lambda p(t_i|e) + (1-\lambda)p(t_i|C)$$

Here $P(t_i|e)$ is the maximum likelihood estimate of term t_i in element *e*, $P(t_i|C)$ is the probability of query term in the collection, and λ is the smoothing parameter. $P(t_i|e)$ is the element model based on element term frequency, whereas $P(t_i|C)$ is the collection model based on inverse element frequency. An important XML-specific feature is element length, since this can vary radically – for example, from a title to a paragraph to a document section. Element length can be captured by setting, the prior probability *P*(*e*), as follows:

$$p(e) = \frac{length(e)}{\sum_{C} length(e)}$$

length(*e*) is the length of element *e*. Including length in the ranking calculation has been shown to lead to more effective retrieval than not doing so.

Contextualization The above strategy only scores an element based on the content of the element itself. Considering additional evidence has shown to be beneficial for XML retrieval. In particular for long documents, using evidence from the element itself as well as its context (for example the parent element) has shown to increase retrieval performance. This strategy is referred to as contextualization. Combining the element score and a separate document score has also been shown to improve performance.

Propagation When only leaf elements are indexed, a propagation mechanism is used to calculate the relevance score of the non-leaf elements. The propagation combines the retrieval scores of the leaf elements (often using a weighted sum) and any additional element characteristics (such as the distance between the element and the leaves). A non-trivial issue is the estimation of the weights for the weighted sum.

Merging It has also been common to obtain several ranked lists of results, and to merge them to form a single list. For example, with the selective indexing strategy [10], a separate index is created for each a priori selected type of element (such as article, abstract, section, paragraph, and so on). A given query is then submitted to each index, each index produces a separate list of elements, normalization is performed (to take into account the variation in size of the elements) and the results are merged. Another approach to merging produces several ranked lists from a single index and for all elements in the collection (a single index is used as opposed to separate indices for each element). Different ranking models are used to produce each ranked list. This can be compared to document fusion investigated in the 1990s.

Processing Structural Constraints Early work in XML retrieval required structural constraints in content-and-structure queries to be strictly matched but specifying an information need including structural constraints is difficult; XML document collections have a wide variety of tag names. INEX now views structural constraints as hints as to where to look (what sort of elements might be relevant). Simple techniques for processing structural constraints include the construction of a dictionary of tag synonyms and structure boosting. In the latter, the retrieval score of an element is generated ignoring the structural constraint but is then boosted if the element matches the structural constraint. More details can be found in the entry on Processing Structural Constraints.

Processing Overlaps It is one task to provide a score expressing how relevant an element is to a query but a different task to decide which of a set of several overlapping relevant elements is the best answer. If an element has been estimated relevant to a query, it is likely that its parent element is also estimated relevant to the query as these two elements share common text. But, returning chains of elements to the user should be avoided to ensure that the user does not receive the same text several times (one for each element in the chain). Deciding which element to return depends on the application and the user model. For instance, in INEX, the best element is one that is highly relevant, but also specific to the topic of request (i.e., does not discuss unrelated topics).

Removing overlap has mostly been done as a post-ranking task. A first approach, and the most commonly adopted one, is to remove elements directly from the result list. This is done by selecting the highest ranked element in a chain and removing any ancestors and descendents in lower ranks. Other techniques looked at the distribution of retrieved elements within each document to decide which ones to return. For example, the root element would be returned if all retrieved elements were uniformly distributed in the document. This technique was shown to outperform the simpler techniques. Efficiency remains an issue as the removal of overlaps is done at query time. More details can be found in the entry on Processing Overlaps.

Key Applications

XML retrieval approaches (from query languages to ranking strategies) are relevant to any applications concerned with the access to repositories of documents annotated in XML, or similar mark-up languages such as SGML or ASN.1. Existing repositories include electronic dictionaries and encyclopedia such as the Wikipedia [4], electronic journals such as the journals of the IEEE [7], plays such as the collected works of William Shakespeare [8], and bibliographic databases such as PubMed. (www.ncbi.nlm.nih.gov/pubmed/) XML retrieval is becoming increasingly important in all areas of information retrieval, the application to full-text book searching is obvious and such commercial systems already exist [11].

Experimental Results

Since 2002 work on XML retrieval has been evaluated in the context of INEX. Many of the proposed approaches have been presented at the yearly INEX workshops, held in Dagsthul, Germany. Each year, the INEX workshop pre-proceedings (which are not peer-reviewed) contain preliminary papers describing the details of participant's approaches. Since 2003 the final INEX workshop proceedings have been peer-reviewed, and since 2004 they have been published by Springer as part of the Lecture Notes in Computer Science series. Links to the pre- and final proceedings can be found on the INEX web site (http://www.inex.otago.ac.nz/).

Data Sets

Since 2002 INEX has collected data sets that can be used for conducting XML retrieval experiments [9]. Each data set consists of a document collection, a set of topics, and the corresponding relevance assessments. The topics and associated relevance assessments are available on the INEX web site (http://www.inex.otago.ac.nz/). It should be noted that the relevance assessments on the latest INEX data set are released first to INEX participants.

Cross-references

- ▶ Aggregation-Based Structured Text Retrieval
- ▶ Content-and-Structure Query
- ▶ Evaluation Metrics for Structured Text Retrieval
- ▶ Indexing Units
- ▶ Information Retrieval Models
- ▶ INitiative for the Evaluation of XML Retrieval
- ▶ Integrated DB&IR Semi-Structured Text Retrieval
- ▶ Logical Structure
- ▶ Narrowed Extended XPath I
- ▶ Presenting Structured Text Retrieval Results
- ▶ Processing Overlaps
- ▶ Processing Structural Constraints
- ▶ Propagation-Based Structured Text Retrieval
- ▶ Relationships in Structured Text Retrieval
- ▶ Structure Weight
- ▶ Structured Document Retrieval
- ▶ Structured Text Retrieval Models
- ▶ Term Statistics for Structured Text Retrieval
- ▶ XML
- ▶ XPath/XQuery
- ▶ XQuery Full-Text

Recommended Reading

1. Amer-Yahia S. and Lalmas M. XML search: languages, INEX and scoring. ACM SIGMOD Rec., 35(4):16–23, 2006.
2. Baeza-Yates R., Fuhr N., and Maarek Y.S. (eds.). Special issue on XML retrieval, ACM Trans. Inf. Syst., 24(4), 2006.
3. Blanken H.M., Grabs T., Schek H.-J., Schenkel R., and Weikum G. (eds.). Intelligent Search on XML Data, Applications, Languages, Models, Implementations, and Benchmarks, Springer, Berlin, 2003.

4. Denoyer L. and Gallinari P. The Wikipedia XML corpus, comparative evaluation of XML information retrieval systems. In Proc. 5th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2007, pp. 12–19.
5. Fuhr N. and Lalmas M. (eds.). Special issue on INEX, Inf. Retrieval, 8(4), 2005.
6. Kamps J., de Rijke M., and Sigurbjörnsson B. The importance of length normalization for XML retrieval. Inf. Retrieval, 8(4):631–654, 2005.
7. Kazai G., Gövert N., Lalmas M., and Fuhr N. The INEX Evaluation Initiative. In Intelligent search on XML data, applications, languages, models, implementations, and benchmarks, H.M. Blanken, T. Grabs, H. Schek, R. Schenkel, G. Weikum (eds.). Springer, 2003, pp. 279–293.
8. Kazai G., Lalmas M., and Reid J. Construction of a test collection for the focused retrieval of structured documents, In Proc. 25th European Conf. on IR Research, 2003, pp. 88–103.
9. Lalmas M. and Tombros A. INEX 2002–2006: understanding XML retrieval evaluation. In Proc. 1st Int. DELOS Conference, 2007, pp. 187–196.
10. Mass Y. and Mandelbrod M. Component ranking and automatic query refinement for XML retrieval. In Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, pp. 73–84.
11. Pharo N. and Trotman A. The use case track at INEX 2006. SIGIR Forum, 41(1): 64–66, 2007.
12. van Zwol R., Baas J., van Oostendorp H., and Wiering F. Bricks: the building blocks to tackle query formulation in structured document retrieval. In Proc. 28th European Conf. on IR Research, 2006, pp. 314–325.

XML Schema

Michael Rys
Microsoft Corporation, Sammamish, WA, USA

Synonyms

XML schema; W3C XML schema

Definition

XML Schema is a schema language that allows to constrain XML documents and provides type information about parts of the XML document. It is defined in a World Wide Web Consortium recommendation [3–5].

Key Points

XML Schema is a schema language that allows the constraint of documents and provides type information about parts of the XML document. It is defined in a World Wide Web Consortium recommendation [3–5]. The current version is 1.0.

Unlike document type descriptions (DTDs) that have been defined in the XML recommendation [1], XML Schema is using an XML based vocabulary to describe the schema constraints.

A schema describes elements and attributes and their content models and types. It provides for complex types that describe the content model of elements and simple types that describe the type of attributes and leaf element nodes. Types can be related to each other in so called derivation hierarchies, either by restricting or extending a super type. Elements with different names can be grouped together into substitution groups if their types are in a derivation relationship.

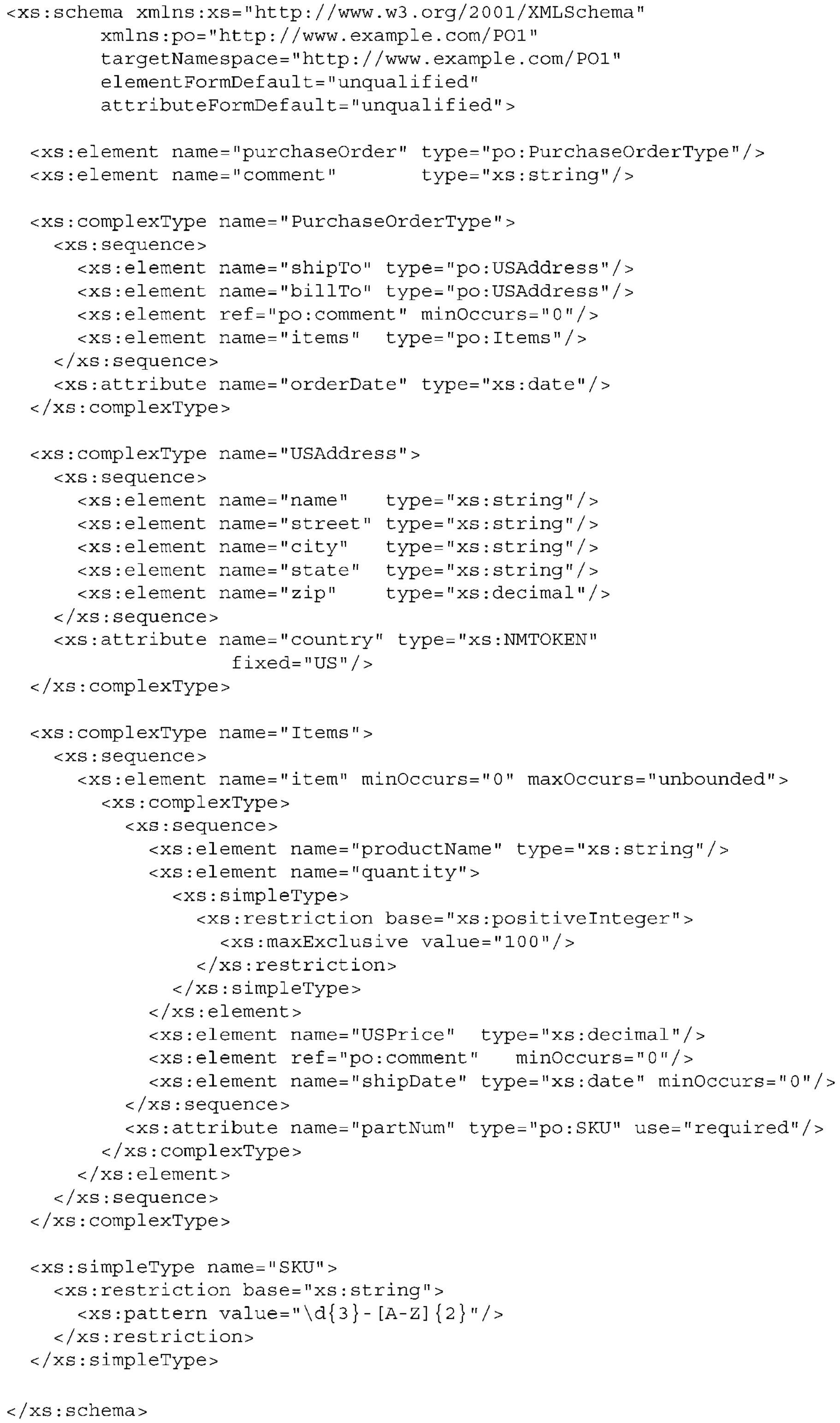
The element, attribute and type declarations are called schema components. A schema is normally associated with a target namespace to which these schema components belong. Figure 1 depicts an example schema (based on examples in [10]) that defines a schema for the target namespace http://www.example.com/PO1 containing the following schema components: two global element declarations, three global complex type declarations and 1 global simple type declaration:

Besides constraining XML documents, XML Schemas are also being used to define vocabularies and semantic models, often in the context of information exchange scenarios to define and constrain the data contracts for data exchanged between clients and services. The schema components can in addition be used for providing additional type information in XQuery [7,8], XPath [2,8] and XSLT [9].

Note that version 1.1 of XML Schema is currently under development at the W3C [6,10].

Cross-references

▶ XML
▶ XML Attribute
▶ XML Document
▶ XML Element
▶ XPath/XQuery
▶ XQuery/XQuery
▶ XSLT/XSLT



```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1"
        elementFormDefault="unqualified"
        attributeFormDefault="unqualified">

  <xs:element name="purchaseOrder" type="po:PurchaseOrderType"/>
  <xs:element name="comment"       type="xs:string"/>

  <xs:complexType name="PurchaseOrderType">
    <xs:sequence>
      <xs:element name="shipTo" type="po:USAddress"/>
      <xs:element name="billTo" type="po:USAddress"/>
      <xs:element ref="po:comment" minOccurs="0"/>
      <xs:element name="items"  type="po:Items"/>
    </xs:sequence>
    <xs:attribute name="orderDate" type="xs:date"/>
  </xs:complexType>

  <xs:complexType name="USAddress">
    <xs:sequence>
      <xs:element name="name"   type="xs:string"/>
      <xs:element name="street" type="xs:string"/>
      <xs:element name="city"   type="xs:string"/>
      <xs:element name="state"  type="xs:string"/>
      <xs:element name="zip"    type="xs:decimal"/>
    </xs:sequence>
    <xs:attribute name="country" type="xs:NMTOKEN"
                   fixed="US"/>
  </xs:complexType>

  <xs:complexType name="Items">
    <xs:sequence>
      <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="productName" type="xs:string"/>
            <xs:element name="quantity">
              <xs:simpleType>
                <xs:restriction base="xs:positiveInteger">
                  <xs:maxExclusive value="100"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
            <xs:element name="USPrice"  type="xs:decimal"/>
            <xs:element ref="po:comment"   minOccurs="0"/>
            <xs:element name="shipDate" type="xs:date" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute name="partNum" type="po:SKU" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="SKU">
    <xs:restriction base="xs:string">
      <xs:pattern value="\d{3}-[A-Z]{2}"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

XML Schema. Figure 1. Example XML Schema.

Recommended Reading

1. XML 1.0 Recommendation, latest edition. Available at: http://www.w3.org/TR/xml
2. XML Path Language (XPath) 2.0, latest edition. Available at: http://www.w3.org/TR/xpath20/
3. XML Schema Part 0: Primer, latest edition. Available at: http://www.w3.org/TR/xmlschema-0/
4. XML Schema Part 1: Structures, latest edition. Available at: http://www.w3.org/TR/xmlschema-1/
5. XML Schema Part 2: Datatypes, latest edition. Available at: http://www.w3.org/TR/xmlschema-2/
6. XML Schema 1.1 Part 2: Datatypes, latest edition. Available at: http://www.w3.org/TR/xmlschema11–2/
7. XQuery 1.0: An XML Query Language, latest edition. Available at: http://www.w3.org/TR/xquery/
8. XQuery 1.0 and XPath 2.0 Data Model (XDM), latest edition. Available at: http://www.w3.org/TR/xpath-datamodel/
9. XSL Transformations (XSLT) Version 2.0, latest edition. Available at: http://www.w3.org/TR/xslt20/
10. W3C XML Schema Definition Language (XSDL) 1.1 Part 1: Structures, latest edition. Available at: http://www.w3.org/TR/xmlschema11–1/

XML Schemas

▶ XML Types

XML Selectivity Estimation

Maya Ramanath[1], Juliana Freire[2],
Neoklis Polyzotis[3]
[1]Max-Planck Institute for Informatics, Saarbrücken, Germany
[2]School of Computing, University of Utah, UT, USA
[3]University of California Santa Cruz, Santa Cruz, CA, USA

Synonyms

XML cardinality estimation

Definition

Selectivity estimation in database systems refers to the task of estimating the number of results that will be output for a given query. Selectivity estimates are crucial in query optimization, since they enable optimizers to select efficient query plans. They are also employed in interactive data exploration as timely feedback about the expected outcome of user queries, and can even serve as approximate answers for count queries.

Selectivity estimators apply an *estimation procedure* on a *synopsis* of the data. Due to the stringent time and space constraints of query optimization, of which selectivity estimation is only one of the steps, selectivity estimators are faced with two, often conflicting, requirements: they have to accurately and efficiently estimate the cardinality of queries while keeping the synopsis size to a minimum.

While there is a large body of literature on selectivity estimation in the context of relational databases, the inherent complexity of the XML data model creates new challenges. Synopsis structures for XML must capture statistical characteristics of document structure in addition to those of data values. Moreover, the flexibility allowed by the use of regular expressions to define XML elements typically results in data that has *highly-skewed structure*. Finally, queries over XML documents require path expressions over possibly long paths, which essentially translates to a number of joins that is much larger than what is found in typical database applications. This increases the scope for inaccurate estimates, since it is known that errors propagate rapidly when estimating multi-way join queries.

Historical Background

Selectivity estimation for XML queries has its roots in early works on *semi-structured data* [7]. These include both lossy techniques for structural summarization [9] and lossless techniques for deriving structural indices [4,8]. The development of XML and associated query languages (XPath and XQuery) has created new challenges to the problem of selectivity estimation. In particular, XML query languages involve a richer set of structure- and content-related operators and hence require more sophisticated techniques for synopsis generation and estimation. A similar observation can be made for the XML data model itself, which is more involved compared to the early models for semi-structured data. This diversity has given rise to a host of different XML selectivity estimation techniques that address different aspects of the general problem. These techniques can be characterized with respect to a number of different features, including: information captured in the synopsis structure (e.g., tree- vs. graph-based view of XML data, structure only vs. structure and content); the class of supported queries (e.g., simple path expressions, XPath); the use of schema information; the ability to provide accuracy guarantees; synopsis generation strategy (e.g., streaming vs.

non-streaming); support for maintaining the synopsis in the presence of updates to the data.

Foundations

In general, XML selectivity estimation techniques consist of two components: a synopsis structure that captures the statistical features of the data, and an estimation framework that computes selectivity estimates based on the constructed synopsis. The techniques presented in the recent literature cover a wide spectrum of design choices in terms of these two components. For instance, Bloom histograms [17] and Markov tables [1] base the synopsis on an enumeration of *simple paths*, while XSketch [11], XSeed [19], and StatiX [3] build synopses structures that capture the *branching path structure* of the data set. Other points of variation include: the supported query model, e.g., structure-only queries [1] vs. queries with *value predicates* [3,11,10]), or linear queries [1] vs. twig queries [2,12]; the use of schema information [3,18]; probabilistic guarantees on estimation error [17]; explicit support for data updates [5,13,17].

Table 1 provides an overview of the current literature on XML selectivity estimation techniques. A detailed description of each technique is beyond the scope of this entry. Instead, the following classification of techniques is discussed: synopses based on path enumeration, graph-based synopses, and updatable synopses.

Path Enumeration

A straightforward approach to handle *simple path expression* queries is to enumerate all paths in the data and store the count for each path. Of course, the number of paths in the data can be very large and thus summarization techniques are required to concisely store this information. Markov tables [1] record information on paths of up to specified length *m* only, and further reduce the number of paths by deleting paths with low frequencies. The selectivity of path expressions for paths with length less than or equal to *m* is derived by a lookup of this table. For longer path expressions, selectivity is computed by combining the counts of shorter paths. In contrast, the bloom histogram method, proposed by Wang et al. [17], groups together simple paths with "similar" frequencies into buckets of a histogram. Each bucket of paths is then summarized using a bloom filter to represent the contents (paths) of the bucket and a representative count is associated with it. Selectivities are estimated by first identifying the bucket containing the query path and retrieving its count. Estimation errors are bounded and depend on the number of histogram buckets and the size of the bloom filter.

While the above techniques use path enumeration as a basic operation to estimate the selectivity of simple path expressions, the same concept is used in [2] to answer more complex "twig" queries, i.e., path queries with branches. The idea is to record, for each path, a small fixed-length signature of the element ids that constitute its roots. These signatures can be "intersected" in order to estimate the number of common roots for several simple paths, or equivalently, the number of matches for the twig query that is formed by joining the simple paths at their root. The set of simple paths and their counts as well as their signatures are organized into a summary data structure called the *correlated subtree* (CST). Given a twig query, it is first broken down in a set of constituent simple paths based on the CST, and the corresponding signatures and counts are combined in order to estimate the overall selectivity.

Graph-Synopsis-Based Techniques

At an abstract level, a graph synopsis summarizes the basic graph structure of an XML document. More formally, given a data graph $G = (V_G, E_G)$, a graph synopsis $\mathcal{S}(G) = (V_\mathcal{S}, E_\mathcal{S})$ is a directed node-labeled graph, where (i) each node $v \in V_\mathcal{S}$ corresponds to a subset of element (or attribute) nodes in V_G (termed the *extent* of) that have the *same label*, and (ii) an edge in $(u,v) \in E_G$ is represented in $E_\mathcal{S}$ as an edge between the nodes whose extents contain the two endpoints *u* and *v*. For each node *u*, the graph synopsis records the common tag of its elements and a count field for the size of its extent.

In order to capture different properties of the underlying path structure and value content, a graph-synopsis is augmented with appropriate, localized distribution information. As an example, the XSketch-summary mechanism [11], which can estimate the selectivity of simple path expressions with branching predicates, augments the general graph-synopsis model with: (i) localized per-edge stability information, indicating whether the synopsis edge is *backward- and/or forward-stable*, and, (ii) localized per-node value distribution summaries. In short, an edge (u,v) in the synopsis is said to be *forward-stable* if all the elements of *u* have at least one child in *v*; similarly, (u,v) is backward stable if all the elements in *v* have at least one parent in *u*. Accordingly,

XML Selectivity Estimation. Table 1. Summary of work on XML selectivity estimation

Proposal	Input	Summary structure	Structure predicates	Value predicates	Updates	Error guarantees
Chen et al. [2]	Data	Correlated subpath tree	Tree pattern	Substring	No	No
Aboulnaga et al. [1]	Data	Path tree and Markov tables	Simple paths	No	No	No
XPathLearner [5]	Query feedback	Markov tables	Simple Paths	Equality Predicates	Yes	No
CXHist [6]	Query feedback	Markov tables	Simple Paths	Substrings	Yes	No
Wu et al. [18]	Data, Schema, Predicates	Position Histogram	Tree pattern	General predicates	No	No
XSketches [11]	Data	Graph	Tree Pattern	Numerical range	No	Yes
XSeed [19]	Data	Graph	Tree pattern	No	No	Yes
XCluster [10]	Data	Graph	Tree pattern	Numerical range, Term Containment and Substring	No	No
StatiX [3]	Data and Schema	Schema graph and Histograms	Tree Pattern	Numerical Range	No	No
Sartiani [15]	Data and Schema	Tagged region graph	Tree Pattern	General Predicates	No	No
Bloom Histograms [17]	Data	Bloom Filter	Simple Paths	No	Yes	Yes
IMAX [13]	Data and Schema	Schema graph and Histogram	Tree pattern	Numerical Range	Yes	No
Wang et al. [16]	Data	Histograms on data samples	Simple Paths	No	No	No
SketchTree [14]	Data Stream	Randomized Sketch	Tree pattern with child axis	General predicates	Yes	Yes

for each node u that represents elements with values, the synopsis records a summary $H(u)$ which captures the corresponding value distribution and thus enables selectivity estimates for value-based predicates.

Recent studies [10,12,19] have proposed a variant of graph-synopses that employ a clustering-based model in order to capture the path and value distribution of the underlying XML data. Under this model, each synopsis node is viewed as a "cluster" and the enclosed elements are assumed to be represented by a corresponding "centroid" that aggregates their characteristics. The TreeSketch [12] model, for instance, defines the centroid of a node u as a vector of average child counts $(c_1,c_2,\ldots,c_n)$, where c_j is the average child count from elements in u to every other node v_j in the synopsis. Thus, the assumption is that each element in u has exactly c_j children to node v_j. Furthermore, the clustering error, that is, the difference between the actual child counts in u and the centroid, provides a measure of the error of approximation.

While the above techniques derive the graph structure from the data itself, StatiX [3] exploits the graph structure provided by the XML Schema to build synopses. In a way, the schema can be regarded as a synopsis of the data since it describes the general structure of the data. StatiX makes use of the mapping between an element in the data and a type in the schema (typically assigned during the validation phase) in order to build its synopsis structure. In addition, the statistics gathering phase consists of assigning ordinal numbers to each type found in the data. The schema graph, which forms a basic synopsis, is augmented with histograms which capture parent-child distributions (built using ordinal numbers) and value distributions (built using values occurring in the data). By appropriately tuning the number of types through a set of *schema transformations* and by decreasing or increasing the number of histogram buckets, coarse to fine-grained synopsis structures can be built. Selectivities of branching path expressions with value predicates can be estimated using either a simple lookup (possible for simple path expressions where the return value for the query corresponds to a single type) or histogram-based cardinality estimation techniques to combine value histograms with parent-child distributions (needed for branching path expressions with value predicates).

Updatable Synopses

Many selectivity estimation techniques for XML assume that the underlying data are *static* and require an *offline scan* over the data in order to gather statistics and build synopsis structures. Although this assumption is valid for applications that primarily use XML as a document exchange format, it does not hold for scenarios where XML is used as a native storage format. For these, it is important that estimators support *synopsis maintenance* in addition to synopsis construction.

The Bloom Histogram technique [17] outlined in Table 1 supports updates by maintaining a "full" summary (maybe on disk) and rebuilding the bloom histogram periodically. That is, each update is first parsed into paths, and a table of paths is updated with the new cardinalities. The bloom histogram which is used for selectivity estimation will be rebuilt from this path table when a sufficient number of updates have been received and the bloom histogram estimates are no longer reliable.

IMAX [13], which is built on top of StatiX [3], updates the synopsis structure directly as and when the update is received. The key concept in this technique is to first identify the correct location of the update (corresponding to the ordinal numbers of the types in the update) and then to estimate and update the correct buckets of the required histograms. It is possible to estimate the amount of error in the histograms and to schedule a re-computation of the required set of histograms from the data if the error becomes too large.

While the approaches outlined above respond directly to changes in the data, XPathLearner [5] and CXHist [6] observe the query processor and use a *query feedback loop* to maintain the synopsis structure. In short, these techniques obtain the true selectivity of each query as it is processed, and use this information in order to update the synopsis accordingly. As a result, the synopsis becomes more refined with respect to frequent queries and can thus provide more accurate selectivity estimates for a significant part of the workload.

Key Applications

The main utility of cardinality estimation is to serve as input into a query optimizer which determines the best execution plan for a query. These estimates can also serve as approximate answers to "COUNT" queries. In addition, they are used in interactive data exploration in order to provide users with early feedback regarding the number of results to the submitted query.

Cross-references

- ▶ Query Optimization
- ▶ Query Processing
- ▶ XML
- ▶ XML Tree Pattern, XML Twig Query
- ▶ XPath/XQuery

Recommended Reading

1. Aboulnaga A., Alameldeen A.R., and Naughton J. Estimating the selectivity of XML path expressions for internet scale applications. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 591–600.
2. Chen Z., Jagadish H.V., Korn F., Koudas N., Muthukrishnan S., Ng R.T., and Srivastava D. Counting twig matches in a tree. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 453–462.
3. Freire J., Haritsa J., Ramanath M., Roy P., and Siméon J. StatiX: Making XML Count. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 181–191.
4. Goldman R. and Widom J. Dataguides: enabling query formulation and optimization in semistructured databases. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 436–445.
5. Lim L., Wang M., Padmanabhan S., Vitter J., and Parr R. XPath-Learner: An on-line self-tuning markov histogram for XML path selectivity estimation. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 442–453.
6. Lim L., Wang M., and Vitter J. CXHist: an on-line classification-based histogram for XML string selectivity estimation. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 1187–1198.
7. McHugh J., Abiteboul S., Goldman R., Quass D., and Widom J. A database management system for semistructured data. ACM SIGMOD Rec., 26(3):54–66, September 1997.
8. Milo T. and Suciu D. Index structures for path expressions. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 277–295.
9. Nestorov S., Ullman J., Wiener J., and Chawathe S. Representative objects: concise representations of semistructured,

hierarchical data. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 79–90.

10. Polyzotis N. and Garofalakis M. XCluster Synopses for structured XML content. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 63.
11. Polyzotis N. and Garofalakis M. XSketch synopses for XML data graphs. ACM Trans. on Database Syst., 31(3):1014–1063, 2006.
12. Polyzotis N., Garofalakis M., and Ioannidis Y. Approximate XML query answers. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 263–274.
13. Ramanath M., Zhang L., Freire J., and Haritsa J. IMAX: incremental maintenance of schema-based XML statistics. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 273–284.
14. Rao P. and Moon B. Sketchtree: approximate tree pattern counts over streaming labeled trees. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 80.
15. Sartiani C. A framework for estimating XML query cardinality. In Proc. 6th Int. Workshop on the World Wide Web and Databases, 2003, pp. 43–48.
16. Wang W., Jiang H., Lu H., and Yu J.X. Containment join size estimation: models and methods. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 145–156.
17. Wang W., Jiang H., Lu H., and Yu J.X. Bloom histogram: path selectivity estimation for XML data with updates. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 240–251.
18. Wu Y., Patel J.M., and Jagadish H.V. Estimating answer sizes for XML queries. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 590–608.
19. Zhang N., Özsu M.T., Aboulnaga A., and Ilyas I.F. XSEED: accurate and fast cardinality estimation for XPath queries. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 61.

XML Storage

Denilson Barbosa[1], Philip Bohannon[2], Juliana Freire[3], Carl-Christian Kanne[4], Ioana Manolescu[5], Vasilis Vassalos[6], Masatoshi Yoshikawa[7]

[1]University of Alberta, Edmonton, AB, Canada
[2]Yahoo! Research, Santa Clara, CA, USA
[3]University of Utah, Salt Lake City, UT, USA
[4]University of Mannheim, Mannheim, Germany
[5]INRIA Saclay–Île-de-France, Orsay, France
[6]Athens University of Economics and Business, Athens, Greece
[7]University of Kyoto, Kyoto, Japan

Synonyms

XML persistence; XML database

Definition

A wide variety of technologies may be employed to physically persist XML documents for later retrieval or update, from relational database management systems to hierarchical systems to native file systems. Once the target technology is chosen, there is still a large number of *storage mapping* strategies that define how parts of the document or document collection will be represented in the back-end technology. Additionally, there are issues of optimization of the technology and strategy used for the mapping. XML Storage covers all the above aspects of persisting XML document collections.

Historical Background

Even though the need for XML storage naturally arose after the emergence of XML, similar techniques had been developed earlier, since the mid-1990's, to store semi-structured data. For example, the LORE system included a storage manager specifically designed for semi-structured objects, while the STORED system allowed the definition of mappings from semi-structured data to relations. Even earlier, storage techniques and storage systems had been developed for object-oriented data. These techniques focused on storing individual objects, including their private and public data and their methods. Important tasks included performing garbage collection, managing object migration and maintaining class extents. Object clustering techniques were developed that used the class hierarchy and the *composition* hierarchy (i.e., which object is a component of which other object) to help determine object location. These techniques, and the implemented object storage systems, such as the O2 storage system, influenced the development of subsequent semi-structured and XML storage systems.

Moreover, the above solutions or ad-hoc approaches had also been used for the storage of large SGML (Standard Generalized Markup Language, a superset and precursor to XML) documents.

Foundations

Given the wide use of XML, most applications need or will need to process and manipulate XML documents, and many applications will need to store and retrieve data from large documents, large collections of documents, or both. As an exchange format, XML can be simply serialized and stored in a file, but serialized document storage often is very inefficient for query processing and updates.

As a result, a large-scale XML storage infrastructure is critical to modern application performance.

Figure 1a shows a simple graphical outline of an XML DTD for movies and television shows.

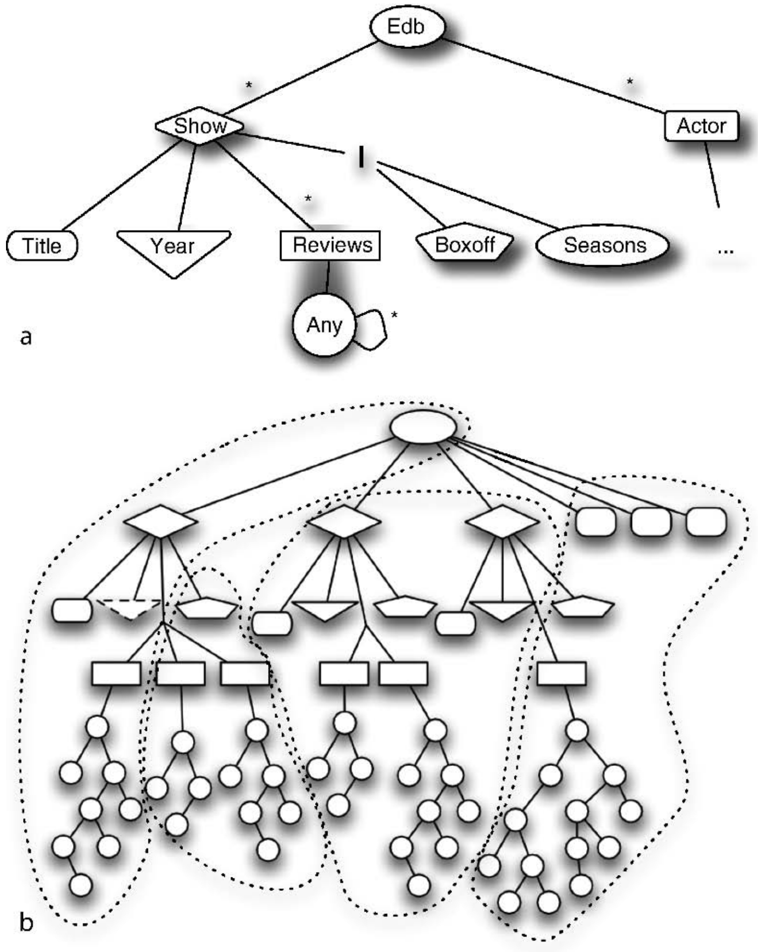
As this example shows, XML data may exhibit great variety in their structure. At one extreme, relational-style data like `title`, `year` and `boxoff` children of `show` in Fig. 1a may be represented in XML. At the opposite extreme are highly irregular structures such as might be found under the `reviews` tag. Figure 2 shows a similar graphical representation of a real-life DTD for scientific articles. Since every HTML structure or formatting element is also an XML element or attribute, the corresponding XML tree is very deep and wide, and no two sections are likely to have the same structure.

XML processing workloads are also diverse. Queries and updates may affect or return few or many nodes. They may also need to "visit" large portions of an XML tree and return nodes that are far apart, such as all the box-office receipts for movies, or may only return or affect nodes that are "close" together, such as all the information pertaining to a single review.

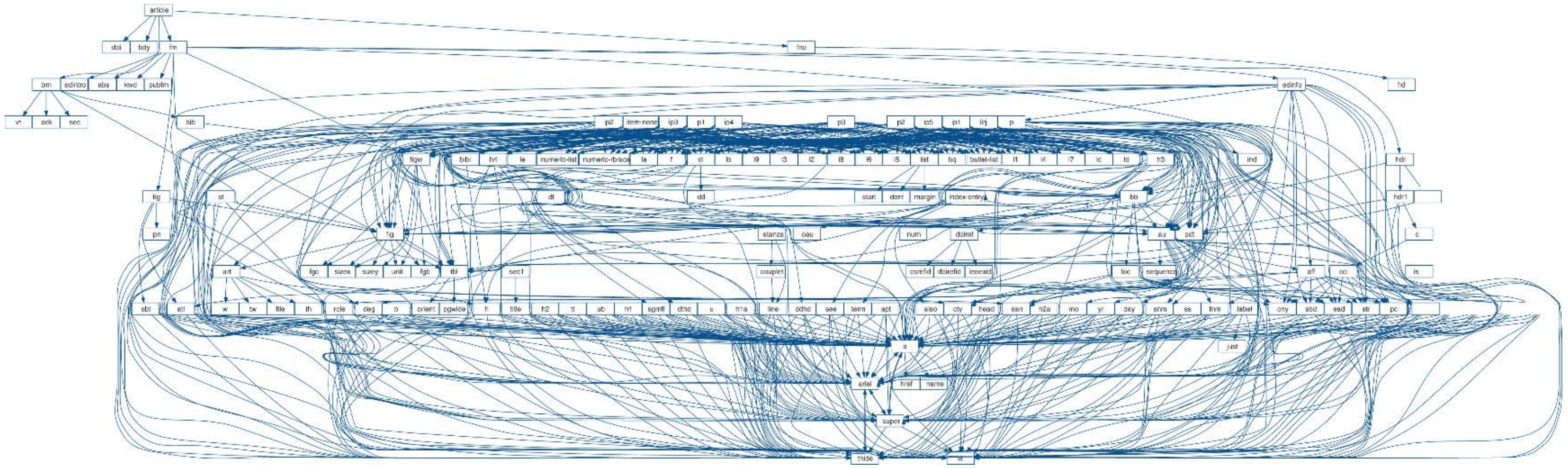
A few different ways of persisting XML document collections are used, and each addresses differently the challenges posed by the varied XML documents and workloads.

Instance-Driven Storage

In *instance-driven storage*, the storage of XML content is driven by the tree structure of the individual document, ignoring the types assigned to the nodes by a schema (if one exists). In some cases, e.g., when documents have irregular structure or an application mostly generates navigations to individual elements, instance-driven storage can greatly simplify the task of storing XML content. One instance-driven technique is to



XML Storage. Figure 1. Movie DTD and example native storage strategy.



XML Storage. Figure 2. Graphical outline of a complex article DTD (strongly simplified).

store nodes and edges in one or more relational tables. A second approach is to implement an XML data model natively.

Tabular Storage of Trees A relational schema for encoding any XML instance may include relations `child`, modeling the parent-child relationship, and `tag`, `attr`, `id`, `text`, associating to each element node respectively a tag, an attribute, an identity and a text value, as well as sets that contain the root of the document and the set of all its elements. Notice that such a schema does not allow full reconstruction of an original XML document, as it does not retain information on element order, whitespace, comments, entity references etc. The encoding of element order, which is a critical feature of XML, is discussed later in this article.

A relational schema for encoding XML may also need to capture *built-in integrity constraints* of XML documents, such as the fact that every child has a single parent, every element has exactly one tag, etc.

Tabular storage of trees as described enables the use of relational storage engines as the target storage technology for XML document collections. While capable of storing arbitrary documents, with this approach a large number of joins may be required to answer queries, especially when reconstructing subtrees. This is the basic storage mapping supported by Microsoft's SQL Server as of 2007.

Native XML Storage Native XML storage software implements data structures and algorithms specifically designed to store XML documents on secondary memory. These data structures support one or more of the XML data models. Salient functional requirements implied by standard data models include the preservation of child order, a stable node identity, and support for type information (depending on the data model supported). An additional functional requirement in XML data stores is the ability to reconstruct the exact textual representation of an XML document, including details such as encoding, whitespace, attribute order, namespace prefixes, and entity references.

A native XML storage implementation generally maps tree nodes and edges to storage blocks in a manner that preserves *tree locality*, i.e., that stores parents and children in the same block. The strategy is to map XML tree structures onto records managed by a storage manager for variable-size records. One possible approach is to map the complete document to a single Binary Large Object and use the record manager's large object management to deal with documents larger than a page. This is one of the approaches for XML storage supported by the commercial DBMS Oracle as of 2007. This approach incurs significant costs both for update and for query processing.

A more sophisticated strategy is to divide the document into partitions smaller than a disk block and map each partition to a single record in the underlying store. Large text nodes and large child node lists are handled by chunking them and/or introducing auxiliary nodes. This organization supports efficient local navigation and tree reconstruction without, for example, loading the entire tree into memory. Such an approach is used in the commercial DBMS IBM DB2 as of 2007 (starting with version 9). Native stores can support efficiently updates, concurrency control mechanisms and traditional recovery schemes to preserve durability (see ACID Properties).

Figure 1b shows a hypothetical instance of the schema of Fig. 1a. The types of nodes are indicated by shape. One potential assignment of nodes to physical storage records is shown as groupings inside dashed lines. Note that `show` elements are often physically stored with their `review` children, and reviews are frequently stored with the next or previous review in document order.

Physical-level heuristics that can be implemented to improve performance include compressed representation of node pointers inside a block, and string dictionaries allowing integers to replace strings appearing repeatedly, such as tag names and namespace URIs.

Schema-Driven Storage

When information about the structure of XML documents is given, e.g., in a DTD or an XML Schema (see XML Schema), techniques have been developed for XML storage that exploit this information. In general, nodes of the same type according to the schema are mapped in the same way, for example to a relational table. Schema information is primarily exploited for tabular storage of XML document collections, and in particular in conjunction with the use of a relational storage engine as the underlying technology, as described in the next paragraph. In *hybrid* XML storage different data models, and potentially even different systems, store different document parts.

Relational Storage for XML Documents

Techniques have been developed that enable the effective use of a relational database management system to store XML. Figure 3a illustrates the main tasks that must be performed for storing XML in relational databases. First, the schema of the XML document is mapped into a suitable relational schema that can preserve the information in the original XML documents (Storage Design). The resulting relational schema needs to be optimized at the physical level, e.g., with the selection of appropriate file structures and the creation of indices, taking into account the distinctive characteristics of XML queries and updates in general and of the application workload in particular. XML documents are then shredded and loaded into the flat tables (Data Loading). At runtime, XML queries are translated into relational queries, e.g., in SQL, submitted to the underlying relational system and the results are translated back into XML (Query Translation). Schema-driven relational storage mappings for XML documents are supported by the Oracle DBMS.

An XML-to-relational mapping scheme consists of *view definitions* that express what data from the XML document should appear in each relational table and constraints over the relational schema. The views generally map elements with the same type or tag name to a table and define a *storage mapping*. For example, in Fig. 3b, two views, `V1` and `V2` are used to populate the `Actors` and `Shows` tables respectively. A particular set of storage views and constraints along with physical storage and indexing options together comprise a *storage design*. The process of parsing an XML document and populating a set of relational views according to a storage design is referred to as *shredding*.

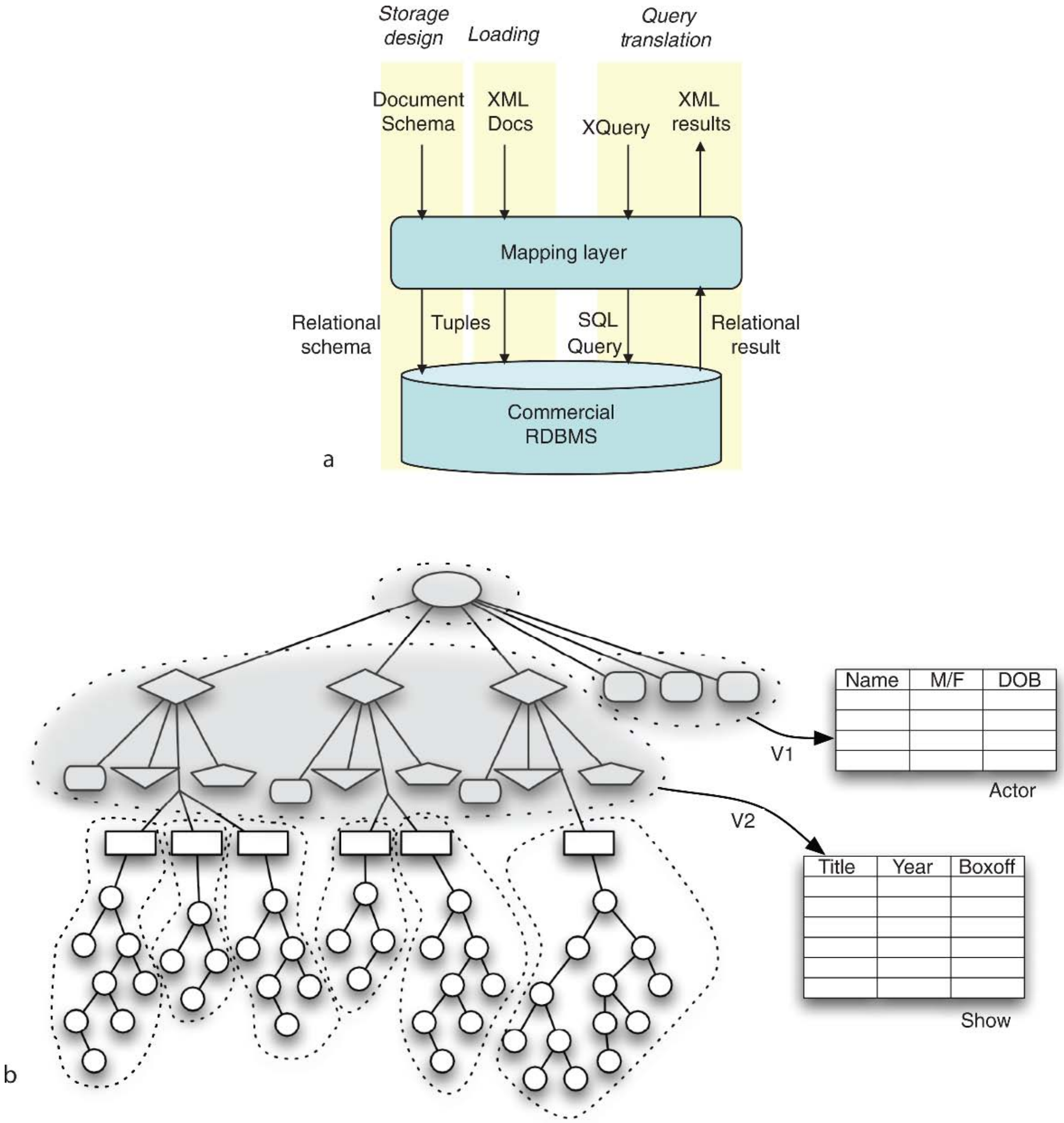
Due to the mismatch between the tree-structure of XML documents and the flat structure of relational tables, there are many possible storage designs. For example, in Fig. 3b, if an element, such as `show`, is guaranteed to have only a single child of a particular type, such as `seasons`, then the child type may optionally be *inlined*, i.e., stored in the same table as the parent.

On the other hand, due to the nature of XML queries and updates, certain indexing and file organization options have been shown to be generally useful. In particular, the use of B-tree indexes (as opposed to hash-based indexes) is usually beneficial, as the translation of XML queries into relational languages often involves range conditions. There is evidence that the best file organization for the relations resulting from XML shredding is index-organized tables, with the index on the attribute(s) encoding the order of XML elements. With such file organization, index scanning allows the retrieval of the XML elements in document order, as required by XPath semantics, with a minimum number of random disk accesses. The use of a path index that stores complete root-to-node paths for all XML elements also provides benefits.

Cost-Based Approaches A key quality of a storage mapping is efficiency – whether queries and updates in the workload can be executed quickly. Cost-based mapping strategies can derive mappings that are more efficient than mappings generated using fixed strategies. In order to apply such strategies, statistics on the values and structure of an XML document collection need to be gathered. A set of transformations and annotations can be applied to the XML schema to derive different schemas that result in different relational storage mappings, for example by merging or splitting the types of different XML elements, and hence mapping them into the same or different relational tables. Then, an efficient mapping is selected by comparing the estimated cost of executing a given application workload on the relational schema produced by each mapping. The optimizer of the relational database used as storage engine can be used for the cost estimation. Due to the size of the search space for mappings generated by the schema transformations, efficient heuristics are needed to reduce the cost without missing the most efficient mappings. Physical database design options, such as vertical partitioning of relations and the creation of indices, can be considered in addition to logical database design options, to include potentially more efficient mappings in the search space.

The basic principles and techniques of cost-based approaches for XML storage are shared with relational cost-based schema design.

Correctness and Losslessness An important issue in designing mappings is correctness, notably, whether a given mapping preserves *enough* information. A mapping scheme is *lossless* if it allows the reconstruction of the original documents, and it is *validating* if all legal relational database instances correspond to a valid XML document. While losslessness is enough for applications involving only queries over the documents, if documents must conform to an XML schema



XML Storage. Figure 3. Relational storage workflow and example.

and the application involves both queries and updates to the documents, schema mappings that are validating are necessary. Many of the mapping strategies proposed in the literature are (or can be extended to be) lossless. While none of them are validating, they can be extended with the addition of constraints to only allow updates that maintain the validity of the XML document. In particular, even though losslessness and validation are undecidable for a large class of mapping schemes, it is possible to guarantee information preservation by designing mapping procedures which guarantee these properties by construction.

Order Encoding Schemes Different techniques have been proposed to preserve the structure and order of XML elements that are mapped into a relational schema. In particular, different labeling schemes have been proposed to capture the positional information of each XML element via the assignment of node labels. An important goal of such schemes is to be able to express structural properties among nodes, e.g., the child, descendant, following sibling and other relationships, as conditions on the labels. Most schemes are either *prefix-based* or *range-based* and can be used with both schema-driven and instance-based relational storage of XML.

In prefix-based schemes, a node's label includes as a prefix the label of its parent. Dewey-based order encodings are the best known prefix-based schemes. The Dewey Decimal Classification was originally developed for general knowledge classification. The basic *Dewey-based encoding* assigns to each node in an XML tree an identifier that records the position of a node among its siblings, prefixed by the identifier of its parent node. In Fig. 1b, the Dewey-based encoding would assign the identifier 1.1.2 to the dashed-line `year` element. In range-based order encodings, such as *interval* or *pre/post encoding*, a unique {*start*,*end*} interval identifies each node in the document tree. This interval can be generated in multiple ways. The most common method is to create a unique identifier, *start*, for each node in a preorder traversal of the document tree, and a unique identifier, *end*, in a postorder traversal. Additionally, in order to distinguish children from descendants, a level number needs to be recorded with each node.

An important consideration for any order-encoding scheme is to be able to handle updates in the XML documents, and many improvements have been made to the above basic encodings to reduce the overhead associated with updates.

Hybrid XML Storage

Some XML documents have both very structured and very unstructured parts. This has lead to the idea of *hybrid* XML storage, where different data models, and even systems using different storage technologies, store different document parts. For example, in Fig. 3b, `review` elements and their subtrees can be stored very differently from `show` elements, for example by serializing each review according to the dashed lines in the figure or storing them in a native XML storage system.

Prototype systems such as MARS and XAM have been proposed that support a hybrid storage model at the system level, i.e., provide physical data independence. In these systems, different access methods corresponding to the different storage mappings are formally described using views and constraints, and query processing involves the use of query rewriting using views. Moreover, an appropriate tool or language is necessary to specify hybrid storage designs effectively and declaratively.

An additional consideration in favor of hybrid XML storage is that storing some information redundantly using different techniques can improve the performance of querying and data retrieval significantly by combining their benefits. For example, schema-directed relational storage mappings often give better performance for identifying the elements that satisfy an XPath query, while native storage allows the direct retrieval of large elements. In environments where updates are infrequent or update cost less important than query performance, such as various web-based query systems, such redundant storage approaches can be beneficial.

Key Applications

XML Storage techniques are used to efficiently store XML documents, XML messages, accumulated XML streams and any other form of XML-encoded content. XML Storage is a key component of an XML database management system. It can also provide significant benefits for the storage of semi-structured information with mostly tree structure, including scientific data.

Cross-references

- ▶ Dataguide
- ▶ Deweys
- ▶ Intervals
- ▶ Storage Management
- ▶ Top-k XML Query Processing
- ▶ XML Document
- ▶ XML Indexing
- ▶ XML Schema
- ▶ XPath/XQuery

Recommended Reading

1. Arion A., Benzaken V., Manolescu I., and Papakonstantinou Y. Structured materialized views for XML queries. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 87–98.
2. Barbosa D., Freire J., and Mendelzon A.O. Designing information-preserving mapping schemes for XML. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 109–120.
3. Beyer K., Cochrane R.J., Josifovski V., Kleewein J., Lapis G., Lohman G., Lyle B., Özcan F., Pirahesh H., Seemann N., Truong T., der Linden B.V., Vickery B., and Zhang C. System RX: one part relational, one part XML. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 347–358.
4. Chaudhuri S., Chen Z., Shim K., and Wu Y. Storing XML (with XSD) in SQL databases: interplay of logical and physical designs. IEEE Trans. Knowl. Data Eng., 17(12):1595–1609, 2005.

5. Deutsch A., Fernandez M., and Suciu D. Storing semi-structured data with STORED. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 431–442.
6. Fiebig T., Helmer S., Kanne C.C., Moerkotte G., Neumann J., Schiele R., and Westmann T. Anatomy of a native XML base management system. VLDB J., 11(4):292–314, 2003.
7. Georgiadis H. and Vassalos V. XPath on steroids: exploiting relational engines for XPath performance. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 317–328.
8. Härder T., Haustein M., Mathis C., and Wagner M. Node labeling schemes for dynamic XML documents reconsidered. Data Knowl. Eng., 60(1):126–149, 2007.
9. McHugh J., Abiteboul S., Goldman R., Quass D., and Widom J. Lore: a database management system for semistructured data. ACM SIGMOD Rec., 26:54–66, 1997.
10. Shanmugasundaram J., Tufte K., He G., Zhang C., DeWitt D., and Naughton J. Relational databases for querying XML documents: limitations and opportunities. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 302–314.
11. Vélez F., Bernard G., and Darnis V. The O2 object manager: an overview. In Building an Object-Oriented Database System, The Story of O2. F. Bancilhon, C. Delobel, and P.C. Kanellakis (eds.), Morgan Kaufmann, San Francisco, CA, USA, 1992, pp. 343–368.

XML Stream Processing

CHRISTOPH KOCH
Cornell University, Ithaca, NY, USA

Definition

XML Stream Processing refers to a family of data stream processing problems that deal with XML data. These include XML stream filtering, transformation, and query answering problems.

A main distinguishing criterion for XML stream processing techniques is whether to filter or transform streams. In the former scenario, XML streams are usually thought of as consisting of a sequence of rather small XML documents (e.g., news items), and the (Boolean) queries decide for each item to either select or drop it. In the latter scenario, the input stream is transformed into a possibly quite different output stream, often using an expressive transformation language.

Historical Background

With the spread of the XML data exchange format in the late 1990s, the research community has become interested in processing streams of XML data. The selective dissemination of information that is not strictly tuple-based, such as electronically disseminated news, was one of the first, and remains one of the foremost applications of XML streams. Subscriptions to items of interest in XML Publish-Subscribe systems are usually expressed in weak XML query languages such as tree patterns and fragments of the XPath query language. This problem has been considered with the additional difficulty that algorithms have to scale to large numbers of queries to be matched efficiently in parallel.

In addition to XML publish-subscribe in the narrow sense, a substantial amount of research has addressed the problems of processing more expressive query and stream transformation languages on XML streams. This includes automata as well as well as data-transformation query language-based approaches (e.g., XQuery).

Foundations

Controlling Memory Consumption. Efficient stream processing is only feasible for data processing problems that can be solved by strictly linear-time one-pass processing of the data using very little main memory. For the XML stream filtering problem, and restricted query languages such as XML tree patterns, linear-time evaluation techniques are not hard to develop (see e.g., [1] for an extensive survey). However, developing one-pass filtering techniques that require little main memory is nontrivial.

Techniques from communication complexity have been used to study memory lower bounds of streaming XPath evaluation algorithms. It has been observed in [5] that there can be no streaming algorithm with memory consumption sublinear in the depth of the data tree, even for simple tree pattern queries and very small XPath fragments. This is a tight bound. Boolean queries in query languages of considerable expressiveness can be processed on XML streams using memory linear in the depth of the tree and independent of any other aspects of its size. By classical reductions between logics and automata on trees, this includes queries in first-order and even monadic second-order logic and as a consequence tree pattern queries and a large class of XPath queries [1]. Since the nesting depth of XML documents tends to be shallow, this is a positive result.

Selecting Nodes or Subtrees. The above observation – that memory consumption does not depend on the size of the XML document but only on the depth of the parse tree – only holds for the problem of testing, for each document, whether it matches the tree pattern or not. If output is to be produced by selecting documents or subtrees, then there are simple queries which require most of the document to be buffered. For an example, consider the following two XML documents.

$$\langle A\rangle\langle B/\rangle\ldots\langle B/\rangle\langle C/\rangle\langle /A\rangle \quad \langle A\rangle\langle B/\rangle\ldots\langle B/\rangle\langle D/\rangle\langle /A\rangle$$

and the XPath query /*[C]. Any implementation of this query that is to output selected documents must select the entire left document but not the right. Hence such an implementation will have to buffer the prefix of either document up to the C resp. D node. This may amount to buffering almost all the document.

The problem of efficiently selecting nodes using XPath on XML streams with close to minimum space requirements was studied in several works, and results are usually space bounds depending linearly on the depth of the data tree, a function of certain properties of the query, and the number of candidate output nodes from the data tree, which in some cases can be nearly all the nodes in the parse tree of the XML document.

Automata-Based Stream Processing. A large part of the research into efficient XML stream processing is based on compiling queries or subscriptions into automata that then run on the data stream. This is not surprising since automata provide a natural one-pass processing model. For most forms of automata one can analyze the runtime memory usage easily. Note, however, that finite word automata are not sufficiently expressive to keep track of the position in the nested XML stream.

Translating XPath queries into pushdown automata has been studied in several works. Pushdown automata yield document depth-bounded space usage. The pushdown automaton nature is somewhat concealed in some of this work; the processing model can be thought of as a finite word automaton for a query path expression which runs on the path from the root node of the XML tree to the current data tree node. There is also a pushdown automaton, independent of the path expression, that acts as a controller for the word automaton, managing the current path using a stack and conceptually rerunning the word automaton every time a new node in the stream is encountered.

The blow-up required to compute *deterministic* finite word automata for query path matching is exponential in size of the query. The sources of this exponentiality were explored in [4]. An alternative is to maintain a nondeterministic finite word automaton for path matching as done in YFilter [3]. This technique is described in detail in.

Transducer Networks. In the following, a technique for constructing automata that have the power to effect stream transformation while being deterministic, consuming little main memory, and being of size polynomial in the size of the query is described. The exponential size of deterministic automata is avoided by not compiling automata for managing and recognizing the subexpressions of an XPath query into a single "flat" automaton. These automata are instead kept apart, as a *transducer network* [6].

A transducer network consists of a set of synchronously running transducers (here, deterministic pushdown transducers) where each transducer runs, possibly in parallel with some other transducers, either on the input XML stream, or on the output of another transducer (in which case the input is the original stream where some nodes may have been annotated using labels). Two transducers may also be "joined," producing output whose annotations are pairs consisting of the annotations produced by the two input transducers.

Next, this is formalized, and some of the transducers that form part of a transducer network are exhibited.

XPath queries are first rewritten into nested filters with paths of length one; for instance, query child::A/descendant::B is first rewritten into child[lab() = $A \wedge$ descendant[lab() = B]]. To emphasize the goal of checking whether the query can be successfully matched, rather than computing nodes matched by a path, axis filters are written as ∃child[ϕ] and ∃descendant[ϕ]. The rewritten queries will now be translated into transducer networks inductively.

The axes used have to be forward axes (child, next sibling, descendant, and following). A large class of XPath queries with other axes (e.g., ancestor) can be rewritten to use just forward axes [2].

A deterministic pushdown transducer T is a tuple $(\Sigma, \Gamma, \Omega, Q, q_0, F, \delta)$ with input alphabet Σ, stack alphabet Γ, output alphabet Ω, set of states Q, start state q_0, set of final states F, and transition function $\delta : Q \times \Sigma \times (\varepsilon \cup \Gamma) \rightarrow Q \times \Gamma^* \times \Omega$. For no $q \in Q$, $s \in \Sigma$, $\gamma \in \Gamma$, both $\delta(q, s, \varepsilon)$ and $\delta(q, s, \gamma)$ are defined. Here ε denotes the empty word. All transducers will have $Q = F$; that is, all states are final states, so all valid runs will be accepting. If the transducer T is in state q and has uv on the stack, and if $\delta(q, s, v) = (q', w, s')$, then T makes a transition to state q' and stack uw ($u,v, w \in \Gamma^*$) on input s, and produces output o, denoted $(q,uv) \xrightarrow{s/o} (q',uw)$. A run on input $s_1 \ldots s_n$ is a sequence of transitions $(q_0,\varepsilon) \xrightarrow{s_1/o_1} \cdots \xrightarrow{s_n/o_n} (q,u)$ that produces output $o_1 \ldots o_n$.

A transducer $T[\exists\text{descendant}[\phi]]$ running on the output stream of transducer $T[\phi]$ is a deterministic pushdown transducer with $\Sigma = \Omega = \{\langle\,\rangle, t, f\}$, $\Gamma = \{t,f\}$, $Q = F = \{q_f, q_t\}$, $q_0 = q_f$, and transition function

$$\delta : \begin{cases} q_x, \langle\rangle, \epsilon) \mapsto (q_f, x, \langle\rangle) \\ (q_x, y \in \{t,f\}, z) \mapsto (q_{x \vee y \vee z}, \epsilon, x) \end{cases}$$

On seeing an opening tag of a node, this transducer memorizes on the stack whether ϕ was matched in the subtrees of the previously seen siblings of that node. On returning (i.e., seeing a closing tag), the transducer labels the node (by its proxy the closing tag) with t or f (true or false) depending on whether ϕ was matched in the node's subtree, which is encoded in the state. Example 1. On input $\langle\,\rangle\,\langle\,\rangle\,\langle\,\rangle\,\langle\,\rangle\, f\, t\, t \langle\,\rangle\,\langle\,\rangle\,\langle\,\rangle\, t\, f\, f\, t$, $T[\exists\text{descendant}[\cdot]]$ has the run

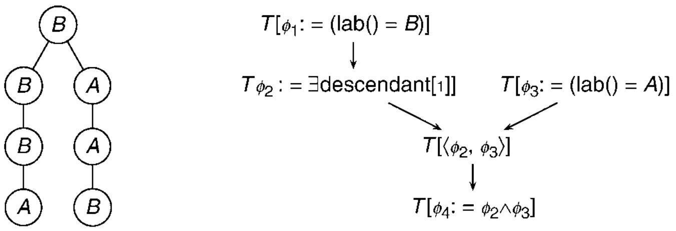
$$\begin{aligned}&(q_f, \epsilon) \xrightarrow{\langle\rangle/\langle\rangle} (q_f, f) \xrightarrow{\langle\rangle/\langle\rangle} (q_f, ff) \xrightarrow{\langle\rangle/\langle\rangle} (q_f, fff) \xrightarrow{\langle\rangle/\langle\rangle} \\ &(q_f, ffff) \xrightarrow{f/f} (q_f, fff)) \xrightarrow{t/f} (q_t, ff) \xrightarrow{t/t} (q_t, f) \xrightarrow{\langle\rangle/\langle\rangle} \\ &(q_f, ft) \xrightarrow{\langle\rangle/\langle\rangle} (q_f, ftf) \xrightarrow{\langle\rangle/\langle\rangle} (q_f, ftff) \xrightarrow{t/f} (q_t, ftf) \xrightarrow{f/t} \\ &(q_t, ft) \xrightarrow{f/t} (q_t, f) \xrightarrow{t/t} (q_t, \epsilon)\end{aligned}$$

and produces output $\langle\,\rangle\,\langle\,\rangle\,\langle\,\rangle\,\langle\,\rangle\, fft\, \langle\,\rangle\,\langle\,\rangle\,\langle\,\rangle\, fttt$ (see Fig. 1).

A transducer $T[\exists\text{child}[\phi]]$ can be defined similarly.

The transducers for testing labels and computing conjunctions of filters do not need a stack. The transducer $T[\text{lab}() = A]$ has the opening and closing tags of the XML document as input alphabet Σ, $\Omega = \{\langle\,\rangle, t, f\}$, $Q = F = \{q_0\}$, and $\delta = \{(q_0, \langle\cdot\rangle, \varepsilon) \mapsto (q_0, \varepsilon, \langle\,\rangle), (q_0, \langle/A\rangle, \varepsilon) \mapsto (q_0, \varepsilon, t), (q_0, \langle/B\rangle, \varepsilon) \mapsto (q_0, \varepsilon, f)\}$ (where B stands for all node labels other than A). The transducer $T[\phi \wedge \psi]$ has $\Sigma = \{\langle\,\rangle\} \cup \{t,f\}^2$, $\Omega = \{\langle\,\rangle, t, f\}$, $Q = F = \{q_0\}$ and $\delta = \{(q_0, \langle\,\rangle, \varepsilon) \mapsto (q_0, \varepsilon, \langle\,\rangle), (q_0, (x,y), \varepsilon) \mapsto (q_0, \varepsilon, x \wedge y)\}$.

The overall execution of a transducer network is exemplified in Fig. 1, where the filter that matches the XPath expression self:: A/descendant:: B, rewritten into $(\exists\text{descendant}[\text{lab}() = B]) \wedge \text{lab}() = A$ is evaluated using a transducer network. The transducers for the different subexpressions run synchronously; each symbol (opening or closing tag) from the input stream is first transformed by $T[\phi_1]$ and $T[\phi_3]$; the output of $T[\phi_1]$ is piped into $T[\phi_2]$ and the output of both $T[\phi_2]$ and $T[\phi_3]$, as a pair of symbols, is piped into $T[\phi_4]$. Only then is the next symbol of the input stream processed, which is handled in the same way, and so on. In the example of Fig. 1, the final transducer labels exactly those nodes t on which the filter is true. Checking whether the filter can be matched on the root node, which is not the case in this example, can be done using an additional pushdown automaton which is not exhibited here but is simple to define.



	Time ⟶													
Input stream	⟨B⟩	⟨B⟩	⟨B⟩	⟨A⟩	⟨/A⟩	⟨/B⟩	⟨/B⟩	⟨A⟩	⟨A⟩	⟨A⟩	⟨/B⟩	⟨/A⟩	⟨/A⟩	⟨/B⟩
transducer	Synchronous output													
$T[\phi_1: = (\text{lab}() = B)]$	⟨⟩	⟨⟩	⟨⟩	⟨⟩	*f*	*t*	*t*	⟨⟩	⟨⟩	⟨⟩	*t*	*f*	*f*	*t*
$T[\phi_2: = \exists\text{descendant}\ [\phi_1]]$	⟨⟩	⟨⟩	⟨⟩	⟨⟩	*f*	*f*	*t*	⟨⟩	⟨⟩	⟨⟩	*f*	*t*	*t*	*t*
$T[\phi_3: = (\text{lab}() = A)]$	⟨⟩	⟨⟩	⟨⟩	⟨⟩	*t*	*f*	*f*	⟨⟩	⟨⟩	⟨⟩	*f*	*t*	*t*	*f*
$T[\langle\phi_2, \phi_3\rangle]$	⟨⟩	⟨⟩	⟨⟩	⟨⟩	(*f*,*t*)	(*f*,*f*)	(*t*,*f*)	⟨⟩	⟨⟩	⟨⟩	(*f*,*f*)	(*t*,*t*)	(*t*,*t*)	(*t*,*f*)
$T[\phi_4 : = \phi_2 \wedge \phi_3]$	⟨⟩	⟨⟩	⟨⟩	⟨⟩	*t*	*f*	*f*	⟨⟩	⟨⟩	⟨⟩	*f*	*t*	*t*	*f*

XML Stream Processing. Figure 1. Document tree (top left), transducer network (top right), and run of the transducer network (bottom).

Key Applications

See XML Publish-Subscribe.

Cross-references

▶ Data Stream
▶ XML Publish/Subscribe

Recommended Reading

1. Benedikt M. and Koch C. Xpath Unleashed. ACM Comput. Surv., 41(3), 2009.
2. Bry F., Olteanu D., Meuss H., and Furche T. Symmetry in XPath. Tech. Rep. PMS-FB-2001-16, LMU München, 2001, short version.
3. Diao et al. Y. YFilter: efficient and scalable filtering of XML documents. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 341–342.
4. Green T.J., Miklau G., Onizuka M., and Suciu D. Processing XML streams with deterministic automata. In Proc. 9th Int. Conf. on Database Theory, 2003, pp. 173–189.
5. Grohe M., Koch C., and Schweikardt N. Tight lower bounds for query processing on streaming and external memory data. Theor. Comput. Sci., 380(1–2):199–217, 2007.
6. Olteanu D. SPEX: streamed and progressive evaluation of XPath. IEEE Trans. Knowl. Data Eng., 19(7):934–949, 2007.

XML Tree Pattern, XML Twig Query

Laks V. S. Lakshmanan
University of British Columbia, Vancouver, BC, Canada

Synonyms

Tree pattern queries; TPQ; TP; Twigs

Definition

A *tree pattern query* (also known as *twig query*) is a pair $Q = (T, F)$, where T is node-labeled and edge-labeled tree with a distinguished node $x \in T$ and F is a boolean combination of constraints on nodes. Node labels are variables such as *$x, $y*. Edge labels are one of "pc," "ad," indicating parent-child or ancestor-descendant. Node constraints are of the form *$x.tag* = TagName or *$x.data* relOp *val*, where *$x.data* denotes the data content of node *$x*, and *relOp* is one of $=, <, >, \leq, \geq, \neq$.

Informally, a tree pattern query specifies a pattern tree, with a set of constraints. Some of the constraints specify what the node labels (tags) should be. Some of them specify how pairs of nodes are related to one another – as a parent-child or as an ancestor-descendant. Finally, constraints on the data content of nodes enforce what data values are expected to be present at the nodes. Taken together, a tree pattern is similar in concept to a selection condition in relational algebra. There, the condition specifies what it takes a tuple to be part of a selection query result. A tree pattern similarly specifies what it takes an XML fragment (i.e., a subtree of the input data tree) to be part of a selection query result. This will be formalized below.

Historical Background

One of the earliest papers to use the term *twig query* was Chen et al. [1]. In this paper, the authors were interested in estimating the number of matches of a twig query against an XML data tree, using a summary data structure. However, the notion of a twig in this paper was confined to tree patterns without ancestor-descendant relationships. Furthermore, data values were required to be strings over an alphabet (i.e., PCDATA) and a match was defined based on identity of node labels in the twig query and the labels of corresponding nodes in the data tree.

Amer-Yahia et al. [2] introduced the notion of tree pattern query where the authors allowed both parent-child and ancestor-descendant relationships between pattern nodes. They motivated such queries in the context of querying LDAP-style network directories and as a core operation in query languages such as XML-QL and Quilt (the prevalent predecessors of XQuery). They focused on minimization of tree pattern queries. Jagadish et al. [3] was the first paper to define tree pattern queries in the current general form. They did so in the context of TAX – a tree algebra for XML data manipulation. Actually, tree patterns as defined in [3] allow for a richer class of node predicates than defined above. An XML data management system called TIMBER that was developed using the foundations of TAX algebra is described in Jagadish et al. [4]. Subsequently, an extensive body of work has flourished on various aspects of tree pattern (or twig) queries, ranging from their minimization [2,5,6], their efficient evaluation via the so-called structural and holistic joins [7,8] and their extensions leveraging indices [9,10]. Twig queries have served as a basis for defining the semantics of group-by for XML data [11]. Subsequently, efficient algorithms for computing group-by queries over XML data were reported in [12] while [13] addresses efficient computation of XML cube. All of these works use tree patterns in an essential way. Given the importance of approximate match when searching an XML document, papers dealing with approximate match have also taken tree patterns as a basis. These include tree pattern relaxation [14] and FleXPath [15]. Last but not the least, given the importance of twig queries, much work has been done on estimating selectivity of twig queries [16–18].

Foundations

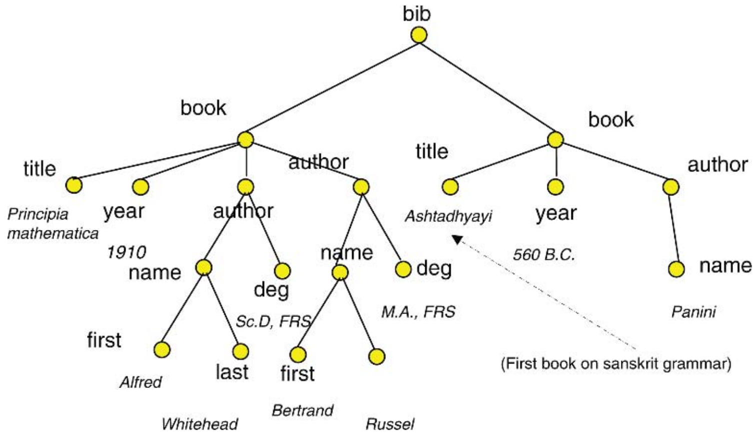
Let $D = (N, A)$ be an XML data tree, i.e., an ordered node-labeled tree whose nodes are labeled by tags, which are element names drawn from an alphabet $\prod$ and whose leaf nodes are additionally labeled by strings from Σ^*. The latter correspond to PCDATA and represent the data content of leaf nodes. A twig query specifies a pattern tree. The semantics of such a query is based on the notion of a match. Let $Q = (T, F)$ be a tree pattern, where $T = (V, E)$ is a node-labeled and edge-labeled tree and let D be an XML data tree. Then a match is defined based on the notion of an embedding. Given Q and D as above, an *embedding* is a function h: V → N, mapping query or pattern nodes to data nodes such that the following conditions are satisfied:

- Whenever $(x, y) \in E$is an edge labeled "pc" (resp., "ad"), $h(y)$ is a child (resp., descendant) of $h(x)$ in D.
- The boolean combination of conditions of the form and *$x.tag* = TagName *and $x.data relOp val* in F is satisfied:
  - An atom *$x.tag* = TagName is satisfied provided *$x* is the label of node v in Q and the tag of the data node $h(v)$ is TagName.
  - An atom *$x.data relOp val is satisfied provided $x* is the label of node v of Q and the node label of the data node $h(v)$ stands in relationship *relOp* to the constant *val*.
- Satisfaction w.r.t. boolean combinations of atoms is defined in the standard way.

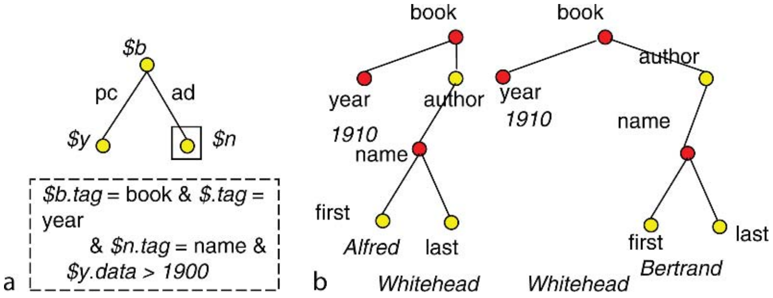
Figure 1 shows a sample data tree. Figure 2 shows a tree pattern query and illustrates matches. In this figure, the distinguished node (node labeled *$n*) is identified by a surrounding box. Informally, this pattern says find the immediate or transitive name sub-elements of book elements such that the book was published after 1900, as indicated by an immediate year sub-element of the book element. Note the edge labels "pc" and "ad" in Fig. 2(a) that specify the intended relationship between elements. In Fig. 2(b), the matching data nodes are indicated in red. The result of a selection query with this tree pattern against the input of Fig. 1(a) consists of the two name sub-elements corresponding to matches of the distinguished node of the tree pattern.

Key Applications

As mentioned earlier, a substantial amount of work on XML query evaluation and optimization has flourished using twig/tree patterns as a basis. One of the reasons for this is that tree pattern queries can be seen as an abstraction of a "core" subset of XPath [19]. In addition to the query evaluation issues mentioned above, tree pattern queries have also attracted attention from the point of view of query static analysis, study of structural properties, and query answering using views. In static analysis, Hidders [20], Lakshmanan



XML Tree Pattern, XML Twig Query. Figure 1. Example XML data tree.



XML Tree Pattern, XML Twig Query. Figure 2. (a) Example tree pattern query and (b) Matches.

et al. [21], and Benedikt et al. [22] study satisfiability of fragments of XPath queries, which can be modeled as tree patterns with possible extensions. Benedikt et al. [23] study structural properties of several XPath fragments corresponding to different extensions of the basic class of tree patterns as defined in this chapter. Xu and Ozsoyoglu [24], Deutsch and Tannen [25], and Lakshmanan et al. [26] study different formulations of the query answering using views problem for XML.

It is important to investigate richer extensions of tree patterns, capturing a greater subset of the features found in XPath. By far, the most important such feature that is lacking in tree patterns is element order. Various papers have examined XPath fragments together with sibling axis. An alternative approach is to consider tree patterns with two kinds of internal nodes – ordered and unordered. An unordered node is just like a node in a standard tree pattern. An ordered internal node is one for which the order of its children in the pattern should obey a specified order. e.g., suppose *x* is an ordered internal node and it has children *y*, *z*, *w* in that order, in the pattern. Then this imposes a constraint that a match of node *z* should follow the corresponding match of *y*. Similarly, a match of *w* should follow the corresponding match of *z*. Another example extension that is worthwhile investigating is a notion of relaxation of a tree pattern with order and keyword search that is appropriate for searching XML documents.

Cross-references

▶ Top-k XML Query Processing
▶ XML Retrieval
▶ XML Schema
▶ XML Selectivity Estimation
▶ XML Views
▶ XPath/XQuery
▶ XSL/XSLT

Recommended Reading

1. Al-Khalifa S., Jagadish H.V., Koudas N., Patel J.M., Srivastava D., and Wu Y. Structural joins: a primitive for efficient XML query pattern matching. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 141–152.
2. Amer-Yahia S., Cho S.R., Lakshmanan L.V.S., and Srivastava D. Minimization of tree pattern queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 497–508.
3. Amer-Yahia S., Cho S.R., and Srivastava D. Tree pattern relaxation. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 496–513.
4. Amer-Yahia S., Lakshmanan L.V.S., and Pandit S. FleXPath: flexible structure and full-text querying for XML. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 83–94.
5. Benedikt M., Fan W., and Geerts F. XPath satisfiability in the presence of DTDs. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 25–36.
6. Benedikt M., Fan W., and Kuper G.M. Structural properties of XPath fragments. In Proc. 9th Int. Conf. on Database Theory, 2003, pp. 79–95.
7. Bruno N., Koudas N., and Srivastava D. Holistic twig joins: optimal XML pattern matching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 310–321.
8. Chen Z., Jagadish H.V., Korn F., Koudas N., Muthukrishnan S., Ng R., and Srivastava D. Counting twig matches in a tree. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 595–604.
9. Chien S.Y., Vagena Z., Zhang D., Tsotras V.J., and Zaniolo C. Efficient structural joins on indexed XML documents. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 263–274.
10. Deutsch A. and Tannen V. Reformulation of XML queries and constraints. In Proc. 9th Int. Conf. on Database Theory, 2003, pp. 225–241.
11. Flesca S., Furfaro F., and Masciari E. On the minimization of Xpath queries. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 153–164.
12. Freire J., Haritsa J., Ramanath M., Roy P., and Simeon J. StatiX: making XML count. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 181–191.
13. Gokhale C., Gupta N., Kumar P., Lakshmanan L.V.S., Ng R., and Prakash B.A. Complex group-by queries for XML. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 646–655.

14. Hidders J. Satisfiability of XPath expressions. In Proc. 9th Int. Workshop on Database Programming Languages, 2003, pp. 21–36.
15. Jagadish H.V., Lakshmanan L.V.S., Srivastava D., and Thompson K. TAX: a tree algebra for XML. In Proc. 8th Int. Workshop on Database Programming Languages, 2001, pp. 149–169.
16. Jagadish H.V., Al-Khalifa S., Chapman A., Lakshmanan L.V.S., Nierman A., Paparizos S., Patel J.M., Srivastava D., Wiwatwattana N., Wu Y., and Yu C.TIMBER: a native XML database. VLDB J., 11(4):274–291, 2002.
17. Jiang H., Lu H., Wang W., and Ooi B.C. XR-tree: indexing XML data for efficient structural joins. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 253–263.
18. Lakshmanan L.V.S., Ramesh G., Wang H., and (Jessica) Zhao Z. On testing satisfiability of tree pattern queries. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 120–131.
19. Lakshmanan L.V.S., Wang H., and (Jessica) Zhao Z. Answering tree pattern queries using views. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 571–582.
20. Miklau G. and Suciu D. Containment and equivalence for a fragment of XPath. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 65–76.
21. Paparizos S., Al-Khalifa S., Jagadish H.V., Lakshmanan L.V.S., Nierman A., Srivastava D., and Wu Y. Grouping in XML. In EDBT 2002 Workshop on XML-Based Data Management LNCS, vol. 2490, Springer, Berlin, 2002, pp. 128–147.
22. Polyzotis N. and Garofalakis M. XSketch synopses for XML data graphs. ACM Trans. Database Syst., September 2006.
23. Polyzotis N., Garofalakis M., and Ioannidis Y. Selectivity estimation for XML twigs. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 264–275.
24. Wiwatwattana N., Jagadish H.V., Lakshmanan L.V.S., and Srivastava D. X^3: a cube operator for XML OLAP. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 916–925.
25. XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath.
26. Xu W. and Meral Özsoyoglu Z. Rewriting XPath queries using materialized views. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 121–132.

XML Tuple Algebra

Ioana Manolescu[1], Yannis Papakonstantinou[2], Vasilis Vassalos[3]

[1]INRIA Saclay–Île-de-France, Orsay, France
[2]University of California-San Diego, La Jolla, CA, USA
[3]Athens University of Economics and Business, Athens, Greece

Synonyms

Relational algebra for XML; XML algebra

Definition

An XML tuple-based algebra operates on a domain that consists of sets of tuples whose attribute values are *items*, i.e., atomic values or XML elements (and hence, possibly, XML trees). Operators receive one or more sets of tuples and produce a set, list or bag of tuples of items. It is common that the algebra has special operators for converting XML inputs into instances of the domain and vice versa. XML tuple-based algebras, as is also the case with relational algebras, have been extensively used in query processing and optimization [1–10].

Historical Background

The use of tuple-based algebras for the efficient set-at-a-time processing of XQuery queries follows a typical pattern in database query processing. Relational algebras are the most typical vehicle for query optimization. Tuple-oriented algebras for object-oriented queries had also been formulated and have a close resemblance to the described XML algebras.

Note that XQuery itself as well as predecessors of it (such as Quilt) are also algebras, which include a list comprehension operator ("FOR"). Such algebras and their properties and optimization have been studied by the functional programming community. They have not been the typical optimization vehicle for database systems.

Foundations

The emergence of XML and XQuery motivated many academic and industrial XML query processing works. Many of the works originating from the database community based XQuery processing on tuple-based algebras since in the past, tuple-based algebras delivered great benefits to relational query processing and also provided a solid base for the processing of nested OQL data and OQL queries. Tuple-based algebras for XQuery carry over to XQuery key query processing benefits such as performing joins using set-at-a-time operations.

A generic example algebra, characteristic of many algebras that have been proposed as intermediate representations for XQuery processing, is described next. It is based on a *Unified Data Model* that extends the XPath/XQuery data model with sets, bags, and lists of tuples; notice that the extensions are only visible to algebra operators.

Given an XML algebra, important query processing challenges include efficient implementation of the operators as well as cost models for them, algebraic properties of operator sequences and algebraic rewriting techniques, and cost-based optimization of algebra expressions.

Unified Data Model

The *Unified Data Model* (UDM) is an extension of the XPath/XQuery data model with the following types:

- *Tuples*, with structure [$\$a_1 = val_1,...,\$a_k = val_k$], where each $\$a_i = item_i$ is a *variable-value* pair. Variable names such as $\$a_1$,$\$a_2$,etc. follow the syntactic restrictions associated with XQuery variable names; Variable names are unique within a tuple. A *value* may be (i) thespecial constant $\perp$(*null*) (The XQuery Data Model also has a concept of *nil*. For clarity of exposition, the two concepts should be considered as distinct, in order to avoid discrepancies that may stem from the overloading.), (ii) an *item*, i.e., a node (generally standing for the root of an XML tree)or atomic value, or (iii) a (nested) set, list, or bag of tuples. Given a tuple [$\$a_1 = val_1,...,\$a_k = val_k$] the list of names [$\$a_1,...,\$a_k$] is called the *schema* of the tuple. We restrict our model to *homogeneous* collections (sets, bags or lists) of tuples. That is, the values taken by a givenvariable $\$a$ in all tuples are of the same kind: either they are all items (some of which may be null), or they are all collections of tuples. Moreover, in the lattercase, all the collections have the same schema.

  Note that the nested sets/bags/lists are typically exploited for buildingefficient evaluation plans.

  If the value of variable $t.\$a$ is a set, list or bag of tuples, let [$\$b_1$, $\$b_2$,...,$\$b_m$] be the schema of a tuple in $t.\$a$. In this case, for clarity, we may denote a $\$b_i$ variable, $1 \leq i \leq m$, by $\$a.\$b_i$ (concatenating the variable names, from the outermost to the innermost, and separating them by dots).
- *Lists, bags and sets of tuples*, denoted as $\langle t_1,...,t_n\rangle$, $\{\{t_1,...,t_n\}\}$, and $\{t_1,...,t_n\}$ and referred to collectively as *collections*. In all three cases the tuples t_1,..., t_n must have the same schema, i.e., collections are homogeneous. Sets have no duplicate tuples, i.e., no two tuples in a set are *id-equal* as defined below.

Two tuples are id-equal, denoted as $t_1 =_{id} t_2$, if they have the same schema and the values of the corresponding variables either (*i*) are both null, or (*ii*) are both equal atomic values (i.e., they compare equal via $=_v$), or are both nodes with the same id (i.e., they compare equal via $=_{id}$), or (*iii*) are both sets of tuples and each tuple of a set is id-equal to a tuple of the other set. For the case (*iii*), similar definitions apply if the variable values are bags or lists of tuples, by taking into account the multiplicity of each tuple in bags and the order of the tuples for lists.

Notation Given a tuple $t = [...\$x = v...]$ it is said that $\$x$ maps to v in the context of t. The value that the variable $\$x$ maps to in the tuple t is $t.\$x$. The notation $t' = t + (\$var = v)$ indicates that the tuple t' contains all the variable-value pairs of t and, in addition, the variable-value pair $\$var = v$. The tuple $t' = t + t'$ contains all the variable-value pairs of both tuples t and t'. Finally, (id) denotes the node with identifier id.

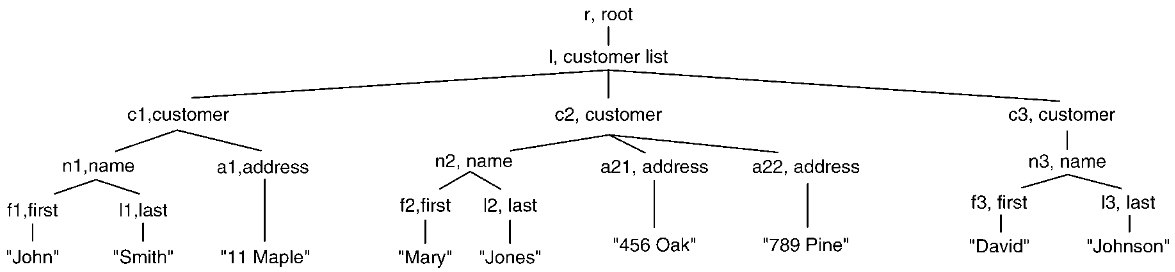
Sample document and query A sample XML document is shown in tree form in Fig. 1. For readability, the figure also displays the tag or string content of each node. The following query will be used to illustrate the XML tuple algebra operators:

```
for $C in $I//customer
return <customer>
  {for $N in $C/namē
       $F in $N/first
       $L in $N/last
  return {$F, $L, $C/address } }
<customer>                              (Q1)
```

Unified Algebra

Tuple-based XQuery algebras typically consist of operators that: (i) perform navigation into the data and deliver collections of bindings (tuples) for the variables; (ii) construct XPath/XQuery Data Model values from the tuples; (iii) create nested collections; (iv) combine collections applying operations known from the relational algebra, such as joins It is common to have redundant operators for the purpose of delivering performance gains in particular environments; structural joins are a typical example of such redundant operators.

An XQuery q, with a set of free variables $\overline{V}$, is translated into a corresponding algebraic expression f_q, which is a function whose input is a collection with schema $\overline{V}$. The algebraic expressions outputs a collection in the *Unified Data Model*. The trivial operators used to convert the collection into an XML-formatted document are not discussed.



XML Tuple Algebra. Figure 1. Tree representation of sample XML document.

Selection The selection operator σ is defined in the usual way based on a logical formula (predicate) that can be evaluated over all its input tuples. The selection operator outputs those tuples for which the predicate evaluated to true.

Projection The projection operator π is defined by specifying a list of column names to be retained in the output. The operator π does not eliminate duplicates; duplicate-eliminating projections are denoted by π^0.

Navigation The navigation operator follows the established tree pattern paradigm. XQuery tuple-based algebras involve tree patterns where the root of the navigation may be a variable binding, in order to capture nested XQueries where a navigation in the inner query may start from a variable of the outer query. Furthermore, XQuery tuple-based algebras have extended tree patterns with an option for "optional" subtrees: if no match is found for optional subtrees, then those subtrees are ignored and a $\perp$ (null) value is returned for variables that appear in them. This feature is similar to outerjoins, and has been used in some algebras to consolidate the navigation of multiple nested queries (and hence of multiple tree patterns) into a single one. In what follows, the tree patterns are limited to child and descendant navigation. The literature describes extensions to all XPath axes.

An *unordered tree pattern* is a labeled tree, whose nodes are labeled with (i) an element/attribute or the wildcard $*$ and (ii) optionally, a variable $\$var$. The edges are either *child* edges, denoted by a single line, or *descendant* edges, denoted by a double line. Furthermore, edges can be *required*, shown as continuous lines, or *optional*, depicted with dashed lines. The variables appearing in the tree pattern form the *schema* of the tree pattern.

The semantics of the navigation operator is based on the *mapping* of a tree pattern to a value. Given a variable-value pair $\$R = r_i$ and a tree pattern T with schema V, a mapping of T to r_i is a mapping of the pattern nodes to nodes in the XML tree represented by r_i such that the structural relationships among pattern nodes are satisfied. A *mapping tuple* (also called *binding*) has schema V and pairs each variable in V to the node to which it is mapped corresponding to such a mapping. The function *map* (T, r_i) returns the set of bindings corresponding to all mappings.

A mapping may be partial: nodes connected to the pattern by optional edges may not be mapped, in which case this node is mapped to a $\perp$.

To formally define embeddings for tree patterns T that have optional edges, the auxiliary *pad* and *sp* functions are introduced.

Given a set of variables $\overline{V}$, the function $pad_{\overline{V}}(t)$ extends the tuple t to have a variable-value pair $\$V = \perp$ for every variable $\$V$ of $\overline{V}$ that is not included in t. i.e., *pad* pads t with nulls so that it has the desired schema $\overline{V}$. The function is overloaded to apply on a set of tuples S, so that $pad_{\overline{V}}(S)$ extends each tuple of S. Given two tuples t and t', it is said that t is *more specific* than t' if for every attribute/variable $\$V$, either $t.\$V = t'.\$V$ or $t'.\$V = \perp$. For example, $[\$A = n_a, \$B = \perp, \$C = \perp]$ is less specific than $[\$A = n_a, \$B = n_b, \$C = \perp]$. Given a set S of tuples named $sp(S)$ the set that consists of all tuples S that are not less specific than any other tuple of S. For example, $sp(\{[\$A = n_a, \$B = \perp, \$C = \perp], [\$A = n_a, \$B = n_b, \$C = \perp]\}) = \{[\$A = n_a, \$B = n_b, \$C = \perp]\}$.

Then given a tree pattern T with set of variables $\overline{V}$ and optional edges, the set of tree patterns $T^1,\ldots,T^n$ are created that have no optional edges and are obtained by non-derministically replacing each optional edge with a required edge, or removing the edge and the subtree that is adjacent to it. The embeddings of the pattern T were than defined as:

$$sp(pad_{\overline{V}}(map(T^1, I) \cup \ldots \cup map(T^n, I)))$$

mapping an unordered tree pattern to a value produces unordered tuples. In an *ordered tree pattern* the variables are ordered by the preorder traversal sequence of the tree pattern, and this ordering translates into an order of the attribute values of each result tuple. Tuples in the mapping result are then ordered lexicographically according to the order of their attribute values.

The navigation operator nav_T inputs and outputs a list of tuples with schema V. The parameter T is a tree pattern, whose root must be labeled with a variable $\$R$ that also appears in V. The input to the operator is a list of the form $\langle t_1,...,t_n\rangle$, where each t_i, $i = 1,..., n$ is a tuple of the form $[...,\$R = r_i,...]$. The output of the operator is the list of tuples $t_i + t'_i$, for all i, where t'_i is defined as $t'_i \in map(T, r_i)$.

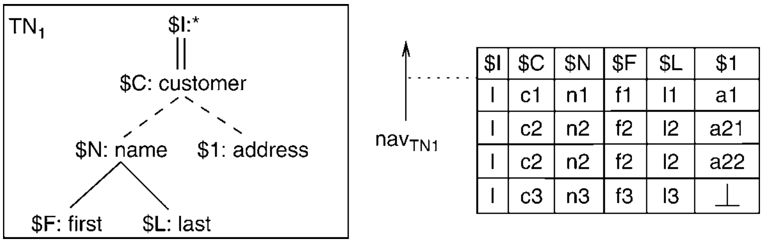
Figure 2 shows the pattern TN_1 corresponding to the navigation part in query Q1, and the result of nav_{TN_1} on the sample document. The order of the tuples is justified as follows: The depth-first pre-order of the variables in the tree pattern is ($\$C$, $\$N$, $\$F$, $\$L$, $\$1$). Consequently, the first tuple precedes the second because $c_1 << c_2$ and the second tuple precedes the third tuple because $a_{21} << a_{22}$.

Construction Tuple-based XQuery algebras include operators that construct XML from the bindings of variables. This is captured by the $crList_L$ operator, which inputs a collection of tuples and outputs an XML tree/forest. The parameter L is a list of *construction tree patterns*, called a *construction pattern list*. For example, consider the query:

```
for$C in $I//customer, $N in $C/name,
   $F in $N/first, $L in $N/last,
   $A in $C/address
return <customer> { $F, $L, $A } </customer>
                                          (Q2)
```

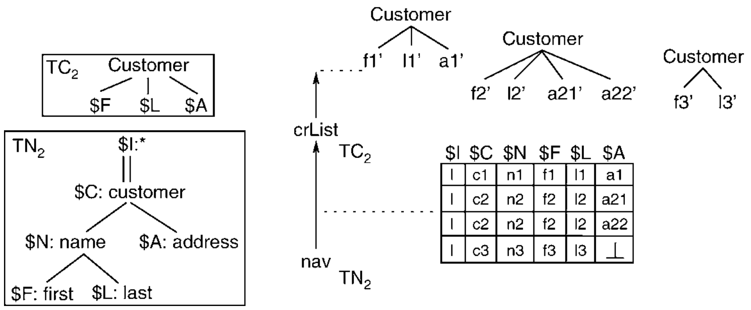
Figure 3 depicts the navigation pattern TN_2, the construction pattern list TC_2, which consists of a single construction pattern, and a simple algebraic expression, corresponding to Q2.

Nested Plans The combination of navigation and construction operators captures the navigation and construction of unnested FLWR XQuery expressions. The following operators can be used to handle nested queries.



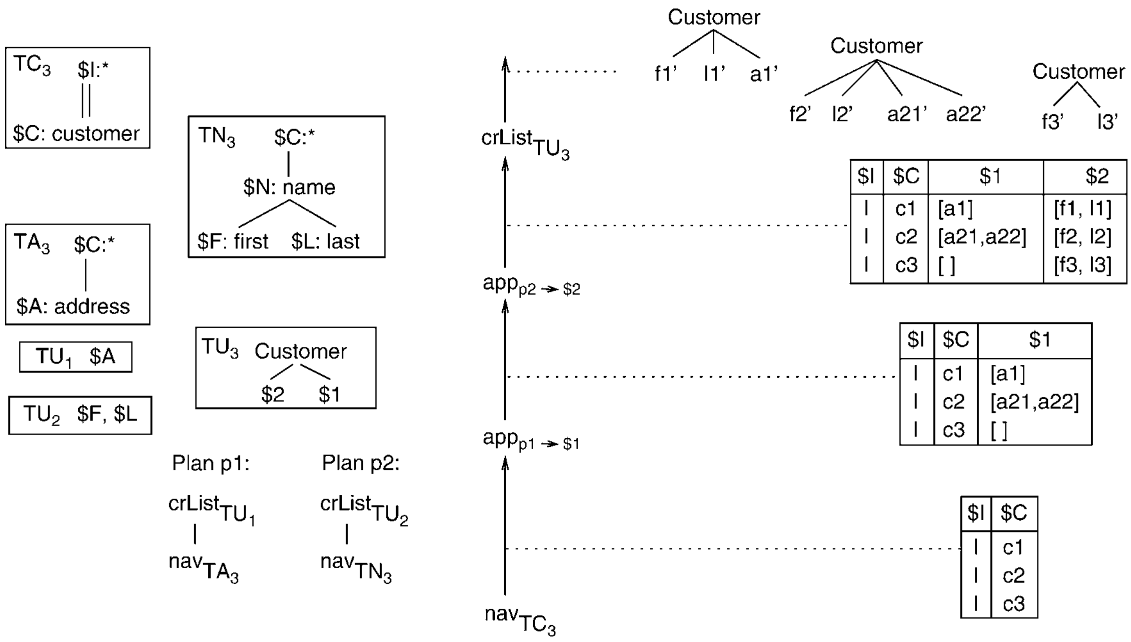
$I	$C	$N	$F	$L	$1
I	c1	n1	f1	l1	a1
I	c2	n2	f2	l2	a21
I	c2	n2	f2	l2	a22
I	c3	n3	f3	l3	⊥

XML Tuple Algebra. Figure 2. Tree pattern for XQuery Q1.



$I	$C	$N	$F	$L	$A
I	c1	n1	f1	l1	a1
I	c2	n2	f2	l2	a21
I	c2	n2	f2	l2	a22
I	c3	n3	f3	l3	⊥

XML Tuple Algebra. Figure 3. Navigation and construction for Q2.

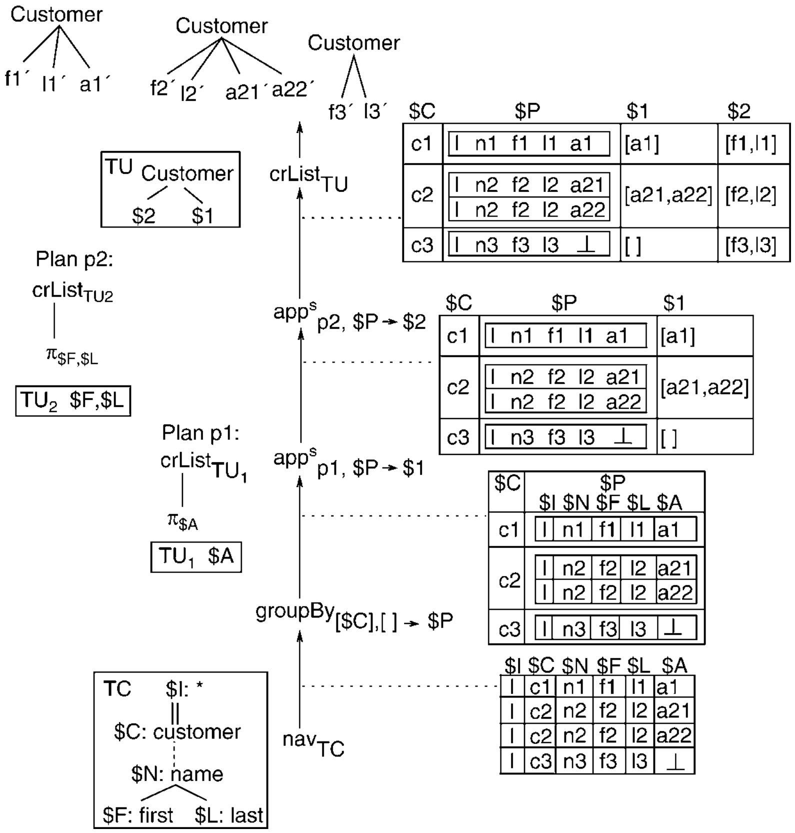


$I	$C	$1	$2
I	c1	[a1]	[f1, l1]
I	c2	[a21,a22]	[f2, l2]
I	c3	[ ]	[f3, l3]

$I	$C	$1
I	c1	[a1]
I	c2	[a21,a22]
I	c3	[ ]

$I	$C
I	c1
I	c2
I	c3

XML Tuple Algebra. Figure 4. Algebraic plan for Q1.



$C	$P	$1	$2
c1	I n1 f1 l1 a1	[a1]	[f1,l1]
c2	I n2 f2 l2 a21 I n2 f2 l2 a22	[a21,a22]	[f2,l2]
c3	I n3 f3 l3 ⊥	[ ]	[f3,l3]

$C	$P	$1
c1	I n1 f1 l1 a1	[a1]
c2	I n2 f2 l2 a21 I n2 f2 l2 a22	[a21,a22]
c3	I n3 f3 l3 ⊥	[ ]

$C	$P $I $N $F $L $A
c1	I n1 f1 l1 a1
c2	I n2 f2 l2 a21 I n2 f2 l2 a22
c3	I n3 f3 l3 ⊥

$I	$C	$N	$F	$L	$A
I	c1	n1	f1	l1	a1
I	c2	n2	f2	l2	a21
I	c2	n2	f2	l2	a22
I	c3	n3	f3	l3	⊥

XML Tuple Algebra. Figure 5. Alternative algebraic plan for Q1.

Apply The $app_{p\mapsto\$R}$ (as in "*app*ly plan") unary operator takes as parameter an algebra expression p, which delivers an XML "forest", and a result variable $\$R$, which should not appear in the schema V of the input tuples. Intuitively, for every tuple t in the input collection I, $p(\{t\})$ is evaluated and the result is assigned to $\$R$.

$$app_{p\mapsto R}(I) = \{t + (R = r) | t \in I, R = p(\{t\})\}$$

For example, query Q1 produces a `customer` output element for every `customer` element in the input. The output element includes the first name and last name pairs (if any) of the customer and for each name all the address children (if any) of the input element. Figure 4 depicts an algebraic plan for Q1, using the *app* operator. TC_3, TA_3 and TN_3 are navigation patterns, while TU_1, TU_2 and TU_3 are construction pattern lists, consisting of one, two, and one, respectively, construction patterns. The partial plans p_1 and p_2 each apply some navigation starting from $\$C$ and construct XML output in \$1 and, respectively, \$2. The first app operator reflects the XQuery nesting, while the second app corresponds to the inner return clause. The final crList operator produces `customer` elements.

The *app* operator is defined on one tuple at a time. However, many architectures (and algebras) consider also a set-at-a-time execution. For instance, instead of evaluating $app_{p1\mapsto\$1}$ on one tuple at a time (which means twice for customer *c2* since she has two addresses), one could group its input by customer ID, and apply p_1 on one group of tuples at a time. In such cases, the following pair of operators is useful.

Group-By The $groupBy_{\overline{G_{id}},\overline{G_v}\mapsto\$R}$ has three parameters: the list of group-by-id variables $\overline{G_{id}}$, the list of group-by-value variables $\overline{G_v}$, and the result variable $\$R$. The operator partitions its input I into sets of tuples $P_{\overline{G}}(I)$ such that all tuples of a partition have id-equal values for the variables of $\overline{G_{id}}$ and equal values for the variables of $\overline{G_v}$. The output consists of one tuple for each partition. Each output tuple has the variables of $\overline{G_{id}}$ and $\overline{G_v}$ and an additional variable $\$R$, whose value is the partition (Some algebras do not repeat the variables of $\overline{G_{id}}$ and $\overline{G_v}$ in the partition, for efficiency reasons.). Note that input tuples that have a $\perp$ in any of the of the $\overline{G_{id}}$ or $\overline{G_v}$ variables are not considered.

Apply-on-Set The $app^s_{p,\$v\mapsto\$R}$ operator assumes that the variable $\$V$ of its input I is bound to a collection of tuples. The operator applies the plan p on $t.\$V$ for every tuple t from the input, and assigns the result to the new variable $\$R$.

For example, Fig. 5 shows another possible plan for Q1. This time a single navigation pattern TC, is used which includes optional edges. The navigation result may contain several tuples for each customer with multiple addresses. Thus, the navigation result is grouped by $\$C$ before applying p_1 on the sets.

The benefits of implementing nested plans by using grouping and operators that potentially deliver null tuples (outerjoin in particular) had first been observed in the context of OQL.

Comparing Fig. 5 with Fig. 4 shows that the same query may be expressed by different expressions in the unified algebra. Providing multiple operators (or combinations thereof) which lead to the same computation is typical in algebras.

Key Applications

An XML tuple algebra is an important intermediate representation for the investigation and the implementation of efficient XML processing techniques. An XML tuple algebra or similar extensions to relational algebra are used as of 2008 by XQuery processing systems such as BEA Aqualogic Data Services Platform and the open source MonetDB/XQuery system.

Cross-references

- ▶ Top-k XML Query Processing
- ▶ XML Document
- ▶ XML Element
- ▶ XML Storage
- ▶ XML Tree Pattern, XML Twig Query
- ▶ XPath/XQuery

Recommended Reading

1. Arion A., Benzaken V., Manolescu I., Papakonstantinou Y., and Vijay R. Algebra-based identification of tree patterns in XQuery. In Proc. 7th Int. Conf. Flexible Query Answering Systems, 2006, pp. 13–25.
2. Beeri C. and Tzaban Y. SAL: an algebra for semistructured data and XML. In Proc. ACM SIGMOD Workshop on The Web and Databases, 1999, pp. 37–42.
3. Cluet S. and Moerkotte G. Nested Queries in Object Bases. Technical report, 1995.
4. Deutsch A., Papakonstantinou Y., and Xu Y. The NEXT logical framework for XQuery. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 168–179.

5. Michiels P., Mihaila G.A., and Siméon J. Put a tree pattern in your algebra. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 246–255.
6. Papakonstantinou Y., Borkar V.R., Orgiyan M., Stathatos K., Suta L.,Vassalos V., and Velikhov P. XML queries and algebra in the Enosys integration platform. Data Knowl. Eng., 44 (3):299–322, 2003.
7. Re C., Siméon J., and Fernández M. A complete and efficient algebraic compiler for XQuery. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 14.
8. The XQuery Language. www.w3.org/TR/xquery, 2004.
9. XQuery 1.0 and XPath 2.0 Data Model. www.w3.org/TR/xpath-datamodel.
10. XQuery 1.0 Formal Semantics. www.w3.org/TR/2005/WD-xquery-semantics.

XML Typechecking

Véronique Benzaken[1], Giuseppe Castagna[2]
Haruo Hosoya[3], Benjamin C. Pierce[4],
Stijn Vansummeren[5]
[1]University Paris 11, Orsay Cedex, France
[2]C.N.R.S. and University Paris 7, Paris, France
[3]The University of Tokyo, Tokyo, Japan
[4]University of Pennsylvania, Philadelphia, PA, USA
[5]Hasselt University and Transnational University of Limburg, Diepenbeek, Belgium

Definition

In general, *typechecking* refers to the problem where, given a program *P*, an input type σ, and an output type τ, one must decide whether *P* is *type-safe*, that is, whether it produces only outputs of type τ when run on inputs of type σ. In the XML context, typechecking problems mainly arise in two forms:

- *XML-to-XML transformations,* where *P* transforms XML documents conforming to a given type into XML documents conforming to another given type.
- *XML publishing,* where *P* transforms relational databases into XML views of these databases and it is necessary to check that all generated views conform to a specified type.

A *type* for XML documents is typically a regular tree language, usually expressed as a schema written in a schema language such as DTD, XML Schema, or Relax NG (see *XML Types*). In the XML publishing case, the input type σ is a relational database schema, possibly with *integrity constraints.*

Typechecking problems may or may not be decidable, depending on (i) the class of programs considered, (ii) the class of input types (relational schemas, DTDs, XML Schemas, Relax NG schema, or perhaps other subclasses of the regular tree languages), and (iii) the class of output types. In cases where it is decidable, typechecking can be done *exactly.* In cases where it is undecidable, one must revert to *approximate* or *incomplete* typecheckers that may return false negatives – i.e., may reject a program even if it is type-safe. Even when exact typechecking is possible, approximate typechecking may be preferable as this is often computationally cheaper than exact typechecking.

In the programming languages literature, typechecking often not only entails verifying that all outputs are of type τ, but also requires detecting when the program may abort with a run-time error on inputs of type *σ* [3]. The above definition encompasses such cases: view run-time errors as a special result value error and then typecheck a program against an output type that does not contain the value error.

Historical Background

Although typechecking is a fundamental and well-studied problem in the theory of programming languages [3], the types necessary for XML typechecking (based on regular tree languages) differ significantly from the conventional data types usually considered (i.e., lists, records, classes, and so on). Indeed, although it is possible to encode XML types into conventional datatypes, this encoding lacks flexibility in the sense that programs tend to need artificial changes when types evolve [22]. For this reason, Hosoya et al. [22] proposed regular tree languages as the "right" notion of types for XML and presented an approximate typechecker in this context. The typechecker was implemented in the XML-to-XML transformation language XDuce [21] whose approach was later extended to general purpose programming by CDuce (functional programming) and Xtatic (object-oriented programming). XDuce's approach also lies at the basis of XQuery's typechecking algorithm [6].

The contemporary study of exact typechecking for XML-to-XML transformations started with an investigation of relatively simple transformation languages [29,32,33]. Ironically enough, the fundamentals of exact typechecking for more advanced transformation languages were already investigated a long time before XML appeared [7,8]. These fundamentals were revived in the XML era by Milo, Suciu, and Vianu in their seminal work on *k*-pebble tree transducers [30], which was later extended to other transformation languages [26,39,40]. The computational complexity of exact typechecking was investigated in [14,27,28]. Exact typechecking algorithms for XML publishing scenarios were given by Alon et al. [1].

Foundations

Exact Typechecking

XML-to-XML Transformations Recall that in this setting, P is a program that should transform XML documents of a type σ into documents of a type τ. When the languages in which the transformation and the types are expressed are sufficiently restricted in power, exact typechecking is possible. There are two major approaches to the construction of an exact typechecking algorithm: *forward inference* and *backward inference*. Forward inference solves the typechecking problem directly by first computing the image O of the input type σ under the transformation P, i.e., $O := \{P(t) | t \in \sigma\}$, and then checking $O \subseteq \tau$ [28,30,32,33,37]. This approach does not work if O goes beyond *context-free tree languages* as checking $O \subseteq \tau$ then becomes undecidable. Sadly, this is already the case when P is written in very simple transformation languages, such as the *top-down tree transducers* (this fact is known as folklore; see, e.g., [14].) Also, computing O itself becomes undecidable for more advanced transformation languages.

Backward inference, on the other hand, first computes the pre-image I of the output type τ under P, i.e., $I := \{t \, | P(t) \in \tau\}$, and then checks $\sigma \subseteq I$. Backward inference often works even when the transformation language is too expressive for forward inference. The technique has successfully been applied to a range of formal models of real-world transformation languages like XSLT, from the top-down and bottom-up tree transducers [7], to macro tree transducers [8,14,27], macro forest transducers [34], *k*-pebble tree transducers [30], tree transducers based on alternating tree automata [39], tree transducers dealing with atomic data values [37], and high-level tree transducers [40].

As mentioned in the definition, static detection of run-time errors can be phrased as a particular form of typechecking by introducing a special output value error. Exact typechecking in this form has been investigated for XQuery programs written in the non-recursive for-let-where return fragment of XQuery without automatic coercions but with the various XPath axes; node constructors; value and node comparisons; and node label and content inspections, in the setting where the input type σ is given by a recursion-free regular tree language. The crux of decidability here is a small-model property: if $P(t)$ = error for some t of type σ then there exists another input t' of type σ whose size depends only on P and σ such that $P(t')$ = error. It then suffices to enumerate all inputs $t' \in \sigma$ up to the maximum size and check $P(t')$ = error. There are only a finite number of such t' (up to isomorphism, and P cannot distinguish between isomorphic inputs), from which decidability follows [41]. This small model property continues to hold when we extend the above XQuery fragment with arbitrary primitives satisfying some general niceness properties [41].

XML Publishing In this setting, the input to the program P is a relational database D. Suppose that P computes its output XML tree by posing simple select-project-join queries to D, nesting the results to these queries, and constructing new XML elements. Exact typechecking for programs of this form, when the input type σ is a relational database schema with key and foreign key constraints, and the output type τ is a "star-free" DTD, is decidable [1]. (See [1] for a precise definition and examples of the concept "star-free.") As was the case for detecting runtime errors in XQuery programs, the crux of decidability here is again a small model property. Typechecking remains decidable for output DTDs τ that are not star-free, but then the queries in P must not use projection. Typechecking unfortunately becomes undecidable when the output types τ are given by XML Schemas or Relax NG Schemas [1]. Typechecking also becomes undecidable when P uses queries more expressive than select-project-join queries.

Approximate Typechecking

The expressive power that realistic applications require of practical transformation languages is often too high to allow for exact typechecking. In such cases, one must revert to approximate or incomplete typecheckers that guarantee that all successfully checked programs are type-safe, but that may also reject some type-safe programs. Existing techniques can be grouped into two categories: type systems and flow analyses.

Type Systems Many conventional programming languages (such as C and Java) specify what programs to accept by a *type system* [35]. Typically, such a system consists of a set of *typing rules* that determine the type of each subexpression of a program. Often, in order to help the typechecker, the programmer is required to supply *type annotations* on variable declarations and in other specified places.

The pioneer work applying this approach to the XML setting was the XDuce (transduce) language [21], whose type system is based on regular tree languages. One significant point in this work is its definition of a natural notion of *subtyping* as the inclusion relation between regular tree languages and its demonstration of the usefulness of allowing a value of one type to be viewed as another type with a syntactically completely different structure [22]. In addition, although the decision problem for subtyping is known to be EXPTIME-complete, the "top-down algorithm" used in the XDuce implementation is empirically shown to be efficient in most cases that actually arise in typechecking [13,22,36]. Such a type system also needs machinery to reduce the amount of type annotations that otherwise tends to be a burden to the user, in particular when the language supports a non-trivial mechanism to manipulate XML documents such as regular expression patterns [20] or filter expressions [17]. A series of works address this problem by proposing automatic type inference schemes that have certain precision properties in a sense similar to the exact typechecking in the previous section [17,20,42]. These ideas have further been extended for XML attributes [19] and para-metric polymorphism [18,43].

CDuce (pronounced "seduce") extends XDuce in various ways [2]. From a language point of view, CDuce embraces XDuce's approach of a functional language based on regular expression patterns [20] and extends it with finer-grained pattern matching, complete two-way compatibility with programs and libraries in the OCaml programming language, Unicode, queries, XML Schema validation, and, above all, higher-order and overloaded functions. XML types are enriched with general purpose data types, intersection and negation types, and functional types. Finally, the CDuce type inference algorithm for patterns is implemented by a new kind of tree automaton and proved to be optimal [10]. Among these extensions, the addition of higher-order functions is significant. Theoretically, this extension is not trivial, first because functions do not fit well in the framework of finite tree automata, but more deeply because this entails a definitional cycle: the definition of typechecking uses subtyping, whose definition then uses the semantics of types (NB: subtyping is defined as inclusion between the sets denoted by given two types), whose definition in turn uses well-typedness of values; the last part depends on typechecking in the presence of higher-order functions since typechecking of a function abstraction $\lambda x.e$ requires analysis of its internal expression e. Some solutions are known for breaking this circularity [14,43]. Also, an approach to combine one of these treatments with high-order functions has been proposed [43].

Several research groups explore ways of mixing a XDuce-like type system with an existing popular language. Xtatic [15] carries out this program for the C# language, developing techniques to blend regular expression types with an object-oriented type system. XJ [16] makes a closely related effort for Java. OCamlDuce [11] mixes with OCaml, proposing a method to intermingle a standard ML type inference algorithm with XDuce-like typechecking. XHaskell [25] is also another instance for Haskell; their approach is, however, to embed XML types into Haskell typing structures (such as tuples and disjoint sums) in the style of data-binding, yet support XDuce-like subtyping in its full flexibility by deploying a coercion technique [25].

The formal semantics of XQuery defined by the W3C contains a type system based on a set of inductive typing rules [6]. Their first draft was heavily based on XDuce's type system [9]. Later, they switched to a different one that reflects the object-oriented hierarchical typing structure adopted by XML Schema.

As byproducts of the above pieces of work, several optimization and compilation techniques that exploit typing information have been proposed [10,24].

Flow-Analysis Flow analysis is a static analysis technique that has long been studied in the programming language community. A series of investigations has been conducted for adapting flow analysis to approximate XML typechecking, concurrently to XDuce-related work, [3,23,31]. In this approach, the user needs to write no type annotations for intermediate values like in XDuce, but instead the static analyzer completely infers them, thus providing a more user-friendly system. One potential drawback is that the specification is rather informal and therefore, when the analyzer raises an error, the reason can sometimes be unclear; empirically, however, such false negatives are rare.

Flow analysis is applied first to Bigwig language system, an extension of Java with an XML-manipulating facility called "templates" [3]. Though this first attempt handles only XHTML types, they naturally generalize it to arbitrary XML types, calling the resulting system XAct [40]. Their techniques are further extended and applied to static analysis of XSLT [31].

Key Applications

XML typechecking is a key component of XQuery, the standard XML query language. As outlined above, XML typechecking in XQuery is based on a set of inductive typing rules that reflects the object-oriented hierarchical typing structure adopted by XML Schema. Different approaches to XML typechecking may be found in research prototypes like CDuce, OcamlDuce, XDuce, XAct, XHaskell, and Xtatic for which references are given below.

Url to Code

CDuce: http://www.cduce.org
OCamlDuce: http://www.cduce.org/ocaml.html
XAct: http://www.brics.dk/Xact/
Xduce: http://xduce.sourceforge.net
XHaskell: http://taichi.ddns.comp.nus.edu.sg/taichi-wiki/XhaskellHomePage
Xtatic: http://www.cis.upenn.edu/~bcpierce/xtatic

A gentle introduction to exact typechecking for both XML-to-XML transformations and XML Publishing can be found in [37]. A non-technical presentation of Regular Expression Types and Patterns and their use in query languages can be found in the joint DPBL and XSym 2005 invited talk [4]. For a more complete presentation of Regular Expression Types and Patterns and the associated type-checking and subtyping algorithms we recommend the reader to refer to the seminal JFP article by Hosoya, Pierce, and Vouillon [22]. The joint ICALP and PPDP 2005 keynote talk [5] constitutes a relatively simple survey of the problem of type-checking higher-order functions and an overview on how to derive subtyping algorithms semantically: full technical details can be found in an extended version published in the JACM [13].

Cross-references

▶ Database Dependencies
▶ XML
▶ XML Types
▶ XPath/XQuery

Recommended Reading

1. Alon N., Milo T., Neven F., Suciu D., and Vianu V. Typechecking xml views of relational databases. ACM Trans. Comput. Log., 4(3):315–354, 2003.
2. Benzaken V., Castagna G., and Frisch A. CDuce: an XML-centric general-purpose language. In Proc. 8th ACM SIGPLAN Int. Conf. on Functional Programming, 2003, pp. 51–63.
3. Brabrand C., Møller A., and Schwartzbach M.I. The <bigwig> project. ACM Trans. Internet Tech., 2(2):79–114, 2002.
4. Castagna G. Patterns and types for querying XML. In Proc. of DBPL 2005, Tenth International Symposium on Database Programming Languages, 2005, pp. 1–26.
5. Castagna G. and Frisch A. A gentle introduction to semantic subtyping. In Proc. 7th Int. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming, 2005, pp. 198–208.
6. Draper D., Fankhauser P., Ashok Malhotra M.F., Rose K., Rys M., Siméon J., and Wadler P. XQuery 1.0 and XPath 2.0 Formal Semantics, 2007. http://www.w3.org/Tr/query-semantics/.
7. Engelfriet J. Top-down tree transducers with regular look-ahead. Math. Syst. Theory, 10:289–303, 1977.
8. Engelfriet J. and Vogler H. Macro tree transducers. J. Comput. Syst. Sci., 31(1):710–146, 1985.
9. Fernández M.F., Siméon J., and Wadler P. A semi-monad for semi-structured data. In Proc. 8th Int. Conf. on Database Theory, 2001, pp. 263–300.
10. Frisch A. Regular tree language recognition with static information. In Proc. 3rd IFIP Int. Conf. on Theoretical Computer Science, 2004, pp. 661–674.
11. Frisch A. OCaml+CDuce. In Proc. 11th ACM SIGPLAN Int. Conf. on Functional Programming, 2006, pp. 192–200.
12. Frisch A., Castagna G., and Benzaken V. Semantic subtyping. In Proc. 17th IEE Conf. on Logic in Computer Science, 2002, pp. 137–146.
13. Frisch A., Castagna G., and Benzaken V. Semantic subtyping: dealing set-theoretically with function, union, intersection, and negation types. J. ACM, 55(4):1–64, 2008.
14. Frisch A. and Hosoya H. Towards practical typechecking for macro tree transducers. In Proc. 11th Int. Workshop on Database Programming Languages, 2007, pp. 246–261.
15. Gapeyev V., Levin M.Y., Pierce B.C., and Schmitt A. The Xtatic experience. In Proc. Workshop on Programming Language Technologies for XML (PLAN-X). January 2005. University of Pennsylvania Technical Report MS-CIS-04-24, 2004.
16. Harren M., Raghavachari M., Shmueli O., Burke M.G., Bordawekar R., Pechtchanski I., and Sarkar V. XJ: facilitating XML processing in Java. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 278–287.
17. Hosoya H. Regular expression filters for XML. J. Funct. Program., 16(6):711–750, 2006.

18. Hosoya H., Frisch A., and Castagna G. Parametric polymorphism for XML. In Proc. 32nd ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages, 2005, pp. 50–62.
19. Hosoya H. and Murata M. Boolean operations and inclusion test for attribute-element constraints. Theor. Comput. Sci., 360 (1–3):327–351, 2006.
20. Hosoya H. and Pierce B.C. Regular expression pattern matching for XML. J. Funct. Program., 13(6):961–1004, 2002.
21. Hosoya H. and Pierce B.C. XDuce: a typed XML processing language. ACM Trans. Internet Tech., 3(2):117–148,2003.
22. Hosoya H., Vouillon J., and Pierce B.C. Regular expression types for XML. ACM Trans. Program. Lang. Syst., 27(1):46–90, 2004.
23. Kirkegaard C. and Møller A. Xact – XML transformations in Java. In Proc. Programming Language Technologies for XML, 2006, p. 87.
24. Levin M.Y. and Pierce B.C. Type-based optimization for regular patterns. In Proc. 10th Int. Workshop on Database Programming Languages, 2005, pp. 184–198.
25. Lu K.Z.M. and Sulzmann M. XHaskell: regular expression types for Haskell. Manuscript, 2004.
26. Maneth S., Perst T., Berlea A., and Seidl H. XML type checking with macro tree transducers. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 283–294.
27. Maneth S., Perst T., and Seidl H. Exact XML type checking in polynomial time. In Proc. 11th Int. Conf. on Database Theory, 2007, pp. 254–268.
28. Martens W. and Neven F. Frontiers of tractability for typechecking simple xml transformations. J. Comput. Syst. Sci., 73(3):362–390, 2007.
29. Milo T. and Suciu D. Type inference for queries on semistructured data. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 215–226.
30. Milo T., Suciu D., and Vianu V. Typechecking for XML transformers. J. Comput. Syst. Sci., 66(1):66–97, 2003.
31. Møller A., Olesen M.O., and Schwartzbach M.I. Static validation of XSL transformations. ACM Trans. Programming Languages and Syst., 29(4): Article 21, 2007.
32. Murata M. Transformation of documents and schemas by patterns and contextual conditions. In Proc. 3rd Int. Workshop on Principles of Document Processing, 1996, pp. 153–169.
33. Papakonstantinou Y. and Vianu V. DTD inference for views of XML data. In Proc. 19th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2000, pp. 35–46.
34. Perst T. and Seidl H. Macro forest transducers. Inf. Process. Lett., 89(3):141–149, 2004.
35. Pierce B.C. Types and Programming Languages. MIT, 2002.
36. Suda T. and Hosoya H. Non-backtracking top-down algorithm for checking tree automata containment. In Proc. 10th Int. Conf. Implementation and Application of Automata, 2005, pp. 83–92.
37. Suciu D. The XML typechecking problem. ACM SIGMOD Rec., 31(1):89–96, 2002.
38. Sulzmann M. and Lu K.Z.M. A type-safe embedding of XDuce into ML. Electr. Notes Theor. Comput. Sci., 148(2):239–264, 2006.
39. Tozawa A. Towards static type checking for XSLT. In Proc. 1st ACM Symp. on Document Engineering, 2001, pp. 18–27.
40. Tozawa A. XML type checking using high-level tree transducer. In Proc. 8th Int. Symp. Functional and Logic Programming, 2006, pp. 81–96.
41. Vansummeren S. On deciding well-definedness for query languages on trees. J. ACM, 54(4):19, 2007.
42. Vouillon J. Polymorphism and XDuce-style patterns. In Proc. Programming Languages Technologies for XML, 2006, pp. 49–60.
43. Vouillon J. Polymorphic regular tree types and patterns. In Proc. 33rd ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages, 2006, pp. 103–114.

XML Types

Frank Neven
Hasselt University and Transnational University of Limburg, Diepenbeek, Belgium

Synonyms

XML schemas

Definition

To constrain the structure of allowed XML documents, for instance with respect to a specific application, a target schema can be defined in some schema language. A schema consists of a sequence of type definitions specifying a (possibly infinite) class of XML documents. A type can be assigned to every element in a document valid w.r.t. a schema. As the same holds for the root element, the document itself can also be viewed to be of a specific type. The schema languages DTDs, XML Schema, and Relax NG, are, on an abstract level, different instantiations of the abstract model of unranked regular tree languages.

Historical Background

Brüggemann-Klein et al. [3] were the first to revive the theory of regular unranked tree automata [14] for the modelling of XML schema languages. Murata et al. [10] provided the formal taxonomy as presented here. Martens et al. [8] characterized the expressiveness of the different models and provided type-free abstractions.

Foundations

Intuition

Consider the XML document in Fig. 1 that contains information about store orders and stock contents.

```
<store>
  <order>
    <customer>
       <name>John Mitchell</name>
       <email> j.mitchell@yahoo.com </email>
    </customer>
    <item> <id> I18F </id>
           <price> 100 </price>
    </item>
    <item>... </item>... <item>... </item>
  </order>
  <order>... </order>... <order>... </order>
  <stock>
    <item>
      <id> IG8 </id> <qty> 10 </qty>
      <supplier> <name> Al Jones </name>
                 <email> a.j@gmail.com </email>
                 <email> a.j@dot.com </email>
       </supplier>
     </item>
    <item>
       <id> J38H </id> <qty> 30 </qty>
       <item>
          <id> J38H1 </id> <qty> 10 </qty>
          <supplier> ... </supplier>
       </item>
       <item>
          <id> J38H2 </id> <qty> 1 </qty>
          <supplier> ... </supplier>
       </item>
       <item> ... </item> ... <item> ... </item>
    </item>
    ...
    <item> ... </item>
  </stock>
</store>
```

XML Types. Figure 1. Example XML document.

Orders hold customer information and list the items ordered, with each item stating its id and price. The stock contents consists of the list of items in stock, with each item stating its id, the quantity in stock, and – depending on whether the item is atomic or composed from other items – some supplier information or the items of which they are composed, respectively. It is important to emphasize that order items do not include supplier information, nor do they mention other items. Moreover, stock items do not mention prices. DTDs are incapable of distinguishing between order items and stock items because the content model of an element can only depend on the element's name in a DTD, and not on the context in which it is used. For example, although the DTD in Fig. 2 describes all intended XML documents, it also allows supplier information to occur in order items and price information to occur in stock items.

The W3C specification essentially defines an XSD as a collection of *type definitions*, which, when abstracted away from the concrete XML representation of XSDs, are rules like

$$store \rightarrow \mathtt{order}[order]^*, \mathtt{stock}[stock] \qquad (\star)$$

that map type names to regular expressions over pairs $a[t]$ of element names a and type names t. Intuitively, this particular type definition specifies an XML fragment to be of type *store* if it is of the form where $n \geq 0$; $f_1,\ldots,f_n$ are XML fragments of type *order*; and g is an XML fragment of type *stock*. Each type name that occurs on the right hand side of a type definition in an XSD must also be defined in the XSD, and each type name may be defined only once. Using types, an XSD can specify that an item is an order item when it occurs under an order element and is otherwise a stock item. For example, Fig. 2 shows an XSD describing the intended set of store document. Note in particular the use of the types $item_1$ and $item_2$ to distinguish between order items and stock items.

It is important to remark that the "Element Declaration Consistent" constraint of the W3C specification requires multiple occurrences of the same element name in a single type definition to occur with the same type. Hence, type definition ($\star$) is legal, but

$$persons \rightarrow (\mathtt{person}[male] + \mathtt{person}[female])^+$$

is not, as person occurs both with type *male* and type *female*. Of course, element names in *different* type definitions can occur with different types (which is exactly what yields the ability to let the content model of an element depend on its context). On a structural level, ignoring attributes and the concrete syntax, the structural expressiveness of Relax NG corresponds to XSDs without the EDC constraint.

A Formalization of Relax NG

An XML fragment $f = f_1 \ldots f_n$ is a sequence of labeled trees where every tree consists of a finite number of nodes, and every node v is assigned an element name denoted by $lab(v)$. There is always a virtual root which

```
<!ELEMENT store (order*, stock)>
<!ELEMENT order (customer, item+)>
<!ELEMENT customer (name, email*)>
<!ELEMENT item (id,price+ (qty,(supplier
                                  +item+)))>
<!ELEMENT stock (item+)>
<!ELEMENT supplier (name, email*)>
```

```
root   →  store [store]
store  →  order [order]*, stock [stock]
order  →  customer [person], item [item1]+
person →  name [emp], email [emp]+
item1  →  id [emp], qty [emp], price [emp]
stock  →  item [item2]+
item2  →  id [emp], qty[emp],
                 (supplier [person] + item [item2]+)
emp    →  ε
```

XML Types. Figure 2. A DTD and an XSD describing the document in Fig. 1.

acts as the common parent of the roots of the different f_i. For a set EName and Types of element and type names, respectively, the set of elements is defined as $\{a[t] | a \in \text{EName}, t \in \text{Types}\}$. The set of regular expressions is given by the following syntax:

$$r ::= \epsilon \mid \alpha \mid r, r \mid r + r \mid r^* \mid r^+ \mid r?$$

where ε denotes the empty string and α is an element. Their semantics is the usual one and is therefore omitted.

An *XSchema* is a tuple $S = (\text{EName}, \text{Types}, \rho, t_0)$ where EName and Types are finite sets of elements and types, respectively, ρ is a mapping from Types to regular expressions, and, $t_0 \in$ Types is the start type.

A *typing* τ of f is a mapping assigning a type $\tau(v) \in$ Types to every node v in f (including the virtual root). For a node v with children $v_1,\ldots,v_m$, define child-string (τ, v) as the string $\text{lab}(v_1)[\tau(v_1)]\ldots\text{lab}(v_1)[\tau(v_1)]$. An XML fragment f then *conforms to* or is *valid w.r.t.S* if there is a typing τ of f such that for every node v, child-string (τ, v) matches the regular expression $\rho(\tau(v))$, and $\tau(\text{root}) = t_0$. The mapping τ is then called a *valid typing*.

Despite the clean formalization, the above definition does not entail a validation algorithm. One possibility is to compute for each node v in f a set of possible types $\Delta(v) \subseteq$ Types such that for each type $t \in \Delta(v)$, the XML subfragment rooted at v is valid w.r.t. the schema with start type t. The XML fragment is then valid w.r.t. S itself when the start type t_0 belongs to $\Delta(\text{root})$. The sets $\Delta(v)$ can be computed in a bottom-up fashion. Indeed, $t \in \Delta(v)$ iff (i) v is a leaf node and $\rho(t)$ contains the empty string; or, (ii) v is a non-leaf node with children $v_1,\ldots,v_n$ and there are $t_1 \in \Delta(v_1),\ldots,t_n \in \Delta(v_n)$ such that $\text{lab}(v_1)[t_1]\ldots\text{lab}(v_n)[t_n] \in \rho(t)$. A valid typing can then be computed from the sets Δ by an additional top-down pass through the tree. Although this kind of bottom-up validation is a bit at odds with the general concept of top-down or streaming XML processing, the algorithm can be adapted to this end (cf. for instance, [10,13]). For general XSchema's, a valid typing is not necessarily unique and cannot always be computed in a single pass [8].

XSchemas as defined above correspond precisely to the class of unranked regular hedge languages [3] and can be seen as an abstraction of Relax NG. Note that the present formalization is overly simplistic w.r.t. attributes as Relax NG treats them in a way uniform to elements using attribute–element constraints [6].

Relationship with Tree and Hedge Automata

Although an XML fragment can consist of a sequence of labeled trees, in the literature it is accustomed to restrict this sequence to simply one tree. XSchemas as defined above then define precisely the unranked regular tree languages [3,11]. Although several automata formalism capturing this class have been defined [3,4,5], each with their own advantages [9], XSchemas correspond most closely to the model of Brüggemann-Klein et al. [3], which is defined as follows. A tree automaton $A = (Q, \text{EName}, \delta, q_0)$, where Q is the set of states (or, types), $q_0 \in Q$ is the start state (start type), and δ maps pairs $(q,a) \in Q \times \text{EName}$ to regular expressions over Q. An input tree f is accepted by the automaton if there exists a mapping τ from the nodes of f to Q, called a run (or, a typing), such that the root is labeled with the start state, and for every non-root node v with children $v_1,\ldots,v_n$, the string $\text{lab}(v_1)\ldots\text{lab}(v_n)$ matches $\delta(\tau(v), lab(v))$. The translation between XSchemas and tree automata is folklore and can for instance be found in [3].

Deterministic Regular Expressions

The unique particle attribution constraint (UPA) requires regular expressions to be deterministic in the following sense: the form of the regular expression should allow each symbol of the input string to match uniquely against a position in the expression when processing the input string in one pass from left to right. That is, without looking ahead in the string. For instance, the expression $r = (a+b)^* a$ is not deterministic as the first symbol in the string *aaa* can already be matched to two different *a*'s in *r*. The equivalent expression $b^* a(b^* a)^*$, on the other hand, is deterministic. Unfortunately, not every regular expression can be rewritten into an equivalent deterministic one [2]. Moreover, it is not a very robust subclass, as it is not closed under union, concatenation, or Kleene-star, prohibiting an elegant constructive definition [2]. Deterministic regular expressions are characterized as one-unambiguous regular expressions by Brüggemann-Klein and Wood [2]. Deciding whether a regular expression is one-unambiguous can be done in quadratic time [1]. Furthermore, it can be decided in EXPTIME whether there is a deterministic regular expression equivalent to a given regular expression [2]. If so, the algorithm can return an expression of a size which is double exponential. It is unclear whether this can be improved.

A Formalization of DTDs and XSDs

Let $S = (\text{EName}, \text{Types}, \rho, t_0)$ be an XSchema. Then, S is *local* when EName = Types and regular expressions in ρ are defined over the alphabet $\{a[a] \mid a \in \text{EName}\}$. This simply means that the name of the element also functions as its type. Furthermore, S is *single-type* when there are no elements $a[t_1]$ and $a[t_2]$ in a $\rho(t)$ with $t_1 \neq t_2$. A DTD is then a local XSchema where regular expressions are restricted to be deterministic. Finally, an XSD is then a single-type XSchema where regular expressions are restricted to be deterministic.

Expressiveness and Complexity

XSchemas form a very robust class, for instance, equivalent to the monadic second-order logic (MSO) definable classes of unranked trees [12], and are closed under the Boolean operations. XSDs on the other hand are not closed under union or complement [8,10]. They define precisely the subclasses of XSchemas closed under ancestor-guarded subtree exchange, and are much closer to DTDs than to XSchemas as becomes apparent from the following equivalent type-free alternative characterization. A *pattern-based XSDP* is a set of rules $\{r_1 \rightarrow s_1, \ldots, r_m \rightarrow s_m\}$ where all r_i are horizontal regular expressions and all s_i are deterministic vertical regular expressions. An XML fragment f is *valid* with respect to P if, for every node v of f, there is a rule $r \rightarrow s \in P$ such that the string formed by the labels of the nodes on the path from the root to v match r and the string formed by the children of v match s (cf. [7,8] for more details). The single-type restriction further ensures that XSDs can be uniquely typed in a one-pass top-down fashion. To be precise, one-pass typing in a top-down fashion means that the first time a node is visited, a type should be assigned (so only based on what has been seen up to now), and that a child can be visited only when its parent has already visited. Type inclusion and equivalence is EXPTIME-complete for XSchemas and is in PTIME for DTDs and XSDs. In fact, w.r.t. the latter, the problem reduces to the corresponding problem for the class of employed regular expressions [8].

Cross-references

▶ XML Schema
▶ XML Typechecking

Recommended Reading

1. Brüggemann-Klein A. Regular expressions into finite automata. Theor. Comput. Sci., 120(2):197–213, 1993.
2. Brüggemann-Klein A. and Wood D. One unambiguous regular languages. Inform. Comput., 140(2):229–253, 1998.
3. Brüggemann-Klein A., Murata M., and Wood D. Regular tree and regular hedge languages over unranked alphabets. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.
4. Carme J., Niehren J., and Tommasi M. Querying unranked trees with stepwise tree automata. In Proc. 15th Int. Conf. Rewriting Techniques and Applications, 2004, pp. 105–118.
5. Cristau J., Löding C., and Thomas W. Deterministic automata on unranked trees. In Proc. 15th Int. Symp. Fundamentals of Computation Theory, 2005, pp. 68–79.
6. Hosoya H. and Murata M. Boolean operations and inclusion test for attribute-element constraints. Theor. Comput. Sci., 360(1–3):327–351, 2006.
7. Martens W., Neven F., and Schwentick T. Simple off the shelf abstractions for XML schema. ACM SIGMOD Rec., 36(4):15–22, 2007.
8. Martens W., Neven F., Schwentick T., and Bex G.J. Expressiveness and complexity of XML schema. ACM Trans. Database Syst., 31(3):770–813, 2006.
9. Martens W. and Niehren J. Minimizing tree automata for unranked trees. In Proc. 10th Int. Workshop on Database Programming Languages, 2005, pp. 232–246.
10. Murata M., Lee D., Mani M., and Kawaguchi K. Taxonomy of XML schema languages using formal language theory. ACM Trans. Internet Tech., 5(4):660–704, 2005.
11. Neven F. Automata theory for XML researchers. ACM SIGMOD Rec., 31(3):39–46, 2002.

12. Neven F. and Schwentick T. Query automata on finite trees. Theor. Comput. Sci., 275:633–674, 2002.
13. Segoufin L. and Vianu V. Validating streaming XML documents. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 53–64.
14. Thatcher J.W. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. J. Comput. Syst. Sci., 1(4):317–322, 1967.

XML Updates

Giorgio Ghelli
University of Pisa, Pisa, Italy

Definition

The term *XML Updates* refers to the act of modifying XML data while preserving its *identity*, through the operators provided by an XML manipulation language. Identity preservation is crucial to this definition: the production of XML data from XML data *without* preserving the original data identity is called *XML transformation*. The general notion of *identity* has many concrete incarnations. The XQuery/XPath data model (see [14]) associates a *Node Identity* to each node of the XML syntax tree. In a language based on this data model, *updates* differ from *transformations* because the former modify the data but preserve node identities. Another hallmark of updates is that an expression that refers to the data being updated may have a different value after the update is evaluated, while the evaluation of XML transformations does not change the value of any other expression.

XML updates may be embedded in any XML manipulation language, but this entry will be focused on XML updates in the context of languages of the XQuery family.

Historical Background

The first languages proposed to manipulate XML data did not support XML updates, because XML transformations can often be used as a substitute for XML updates, but the former are simpler to optimize and have cleaner semantics. However, many applications need to update persistent XML data, and eventually the problem was tackled. The first widely-known proposal in the scientific literature was presented in [11], and was clearly influenced by previous work on SQL updates and on primitives for tree updates.

Foundations

The design of an XML update mechanism has three important aspects:

- Definition of the update operators
- Revalidation of modified data after the updates
- Embedding of the operators in the language

Operators

There is a wide agreement on the basic update operators. Almost every proposal includes operators to delete a subtree, insert a subtree in a specific position, rename a node, and replace the value of a node. The deletion of a node *T* may be defined as an operation that just detaches *T* from its parent (*detach* semantics) or as an operation that invalidates every node of *T* (*erasure* semantics). The first choice needs to be supported by a garbage-collector, while the second may require the management of pointers to invalidated nodes. The insertion of a tree *T* below a node *n* may break the tree-structure of XML data, if the tree *T* has already a parent, and would create a cycle if *n* were a node of *T*. For this reason, the insertion operator typically copies *T* before inserting it below *n*. Node renaming must be defined with some care because it may break invariants connected to name spaces. Some proposals also include a *move T1 into T2* operation to move a subtree *T*1 to a different location without altering its identity. This operation cannot be encoded by inserting *T*1 into *T*2, and then deleting the original *T*1, because the insertion performs a copy, hence this encoding does not preserve the identity of *T*1.

Revalidation

Dynamic revalidation is, currently, the technique of choice, in order to ensure that modified data still respect type invariants. The complexity of revalidation depends on the language used to express structural invariants. For example, if a DTD is used, when a valid subtree rooted at an element with name *q* is moved from a position to another, it is only necessary to verify whether an element with name *q* may be found in that position. If XML Schema is used, then the content model of the moved subtree may depend on its position, hence the subtree has to be revalidated. In any case, a full revalidation of the updated structure is usually not needed, and many optimizations are generally possible. In some cases, static code analysis may avoid dynamic revalidation altogether. Work on this aspect is only at its beginning stages.

The Operators and the Full Language

The presence of update operators in an XML language may heavily affect the possibility of optimization. Any important optimization aims at reducing the number of times an expression is evaluated. Such a reduction is typically possible if the value of that expression at a certain time is guaranteed to be equal to its value when it was last evaluated. Updates make this property very difficult to prove. This problem has been faced with two main approaches: separation of queries and updates, and delayed update application.

The separation approach is exemplified by [6,10,11]: these languages distinguish between *expressions* and *statements* and put strong limitations on the places where statements may appear. The proposals in [2,3] perform a "partial" separation of queries and updates. They use the same syntax for non-updating expressions and updating expressions, which means, for example, that standard FLWOR expressions are used to iterate both classes of expressions. However, the places where updating expressions may appear are severely limited: for example, in a FLWOR expression, they can only appear in the *return* clause. The languages XQuery! and LiXQuery$^+$ [8,9], instead, have just one syntactic category (*expressions*) and no syntactic limitation on where the updates may appear.

The delayed-application approach is based on the definition of a *snapshot scope*; all the update expressions, or statements, in the scope are not executed immediately, but only when the scope is closed. This ensures that, inside the scope, the value of an expression is not affected by updates. For example, the proposals of [3,10,11] define a whole-query snapshot scope, while in XQueryP [2] every update is applied immediately. Finally, in XQuery! and LiXQuery$^+$ [8,9], the programmer has a full control of the snaphot scope.

Apart from optimization, the delayed approach is also proposed for consistency reasons. The partial execution of a set of correlated updates would create consistency problems. These problems can be avoided by collecting these updates in a scope, and by imposing that all the updates in each snapshot scope are evaluated atomically.

Key Applications

Updates are unavoidable in any language that manipulates XML persistent data, as the one proposed in [11], and are very useful in any language for scripting purposes, as XQueryP [2]. Application scenarios are detailed in [4] and in [5].

Future Directions

The most important open problems, in this field, are optimization and static analysis. Some work on the optimization of queries with updates has been done (see [1,7], for example), but the field is huge. Static analysis is also an area where lot of work has to be done. Here, a crucial issue is the design of algorithms to substitute dynamic post-update revalidation with some form of static type-checking.

Experimental Results

Some of the proposals have a public implementation that has been use to experiment with the semantics and the performance of the language; for example, XL has a demo reachable from [13], and XQuery! has a demo reachable from [15]. The W3C maintains a list of XQuery implementations in [12].

Cross-references

- ▶ XML
- ▶ XML Algebra
- ▶ XML Query Processing
- ▶ XML Type Checking
- ▶ XPath/XQuery

Recommended Reading

1. Benedikt M., Bonifati A., Flesca S., and Vyas A. Adding updates to XQuery: Semantics, optimization, and static analysis. In Proc. 2nd Int. Workshop on XQuery Implementation, Experience and Perspectives, 2005.
2. Carey M., Chamberlin D., Fernandez M., Florescu D., Ghelli G., Kossmann D., Robie J., and Siméon J. XQueryP: an XML application development language. In Proc. of XML, 2006.
3. Chamberlin D., Florescu D., and Robie J. XQuery update facility. W3C Working Draft, July 2006.
4. Chamberlin D. and Robie J. XQuery update facility requirements. W3C Working Draft, June 2005. http://www.w3.org/TR/xquery-update-requirements/.
5. Engovatov D., Florescu D., and Ghelli G. XQuery scripting extension 1.0 requirements. W3C Working Draft, June 2007. http://www.w3.org/TR/xquery-sx-10-requirements.
6. Florescu D., Grünhagen A., and Kossmann D. XL: an XML programming language for Web service specification and composition. In Proc. 11th Int. World Wide Web Conference, 2002, pp. 65–76.
7. Ghelli G., Onose N., Rose K., and Siméon J. XML query optimization in the presence of side effects. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2008, pp. 339–352.

8. Ghelli G., Ré C., and Siméon J. XQuery!: an XML query language with side effects. In Proc. 2nd Int. Workshop on Database Technologies for Handling XML Information on the Web, 2006, pp. 178–191.
9. Hidders J., Paredaens J., Vercammen R., and Demeyer S. On the expressive power of XQuery-based update languages. In Database and XML Technologies, 5th Int. XML Database Symp., 2006, pp. 92–106.
10. Sur G.M., Hammer J., and Siméon J. An XQuery-based language for processing updates in XML. In Proc. Programming Language Technologies for XML(PLAN-X), 2004.
11. Tatarinov I., Ives Z., Halevy A., and Weld D. Updating XML. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 413–424.
12. W3C. W3C XQuery site. http://www.w3.org/XML/Query.
13. XL team. XL site. http://xl.inf.ethz.ch.
14. XQuery 1.0 and XPath 2.0 data model (XDM). W3C Recommendation, January 2007.
15. XQuery! team. XQuery! site. http://xquerybang.cs.washington.edu.

XML Views

Murali Mani
Worcester Polytechnic, Institute, Worcester, MA, USA

Synonyms

XML publishing

Definition

Database applications provide an XML view of their data so that the data is available to other applications, especially web applications. Database systems provide support for the client applications to use (query and/or manipulate) the data. The operations specified by the client applications are composed with the view definitions by the database system, thus performing these actions. The internal data model used by the database application, as well as how the operations are performed are transparent to the client applications; they see only an XML view of the entire system. XML views help the database systems to maintain their legacy data, as well as utilize the optimization features present in legacy systems (especially SQL engines), and at the same time make the data accessible to a wide range of web applications.

Historical Background

Views (external schema) are a feature [12] present universally in almost all database systems. Views provide data independence as well as the ability to control access of portions of data to different classes of users. With XML [2] becoming the standard for information exchange over the web since 1998, database applications have used XML views to publish their data and to make the data accessible to web applications. Nowadays, XML views are supported by most major database engines like Microsoft SQL server, Oracle, and IBM DB2.

Foundations

The views that database systems support can either be virtual or materialized [12]. When the view is virtual, only the view definition is stored in the system. Whenever a client application accesses the data by issuing a query over the view, the database system composes the user query with the view definition and this combined query is executed over the underlying data. The advantage of virtual views are that the data is never out-of-date as the data is stored, accessed from and manipulated in only one place. Materialized views on the other hand store the data for the view along with the view definition. Therefore the same data is now in more than one location, and thus the different copies of the data need to be kept consistent by maintaining the materialized views whenever the underlying data changes. Incremental and efficient maintenance of materialized views have been studied [6,9,11] and are supported by most commercial SQL engines. The advantage of materialized views is that the user query can potentially be answered faster as the user query can be directly answered from the materialized view, and does not require composing it with the view definition. In this article, both virtual and materialized XML views that are defined primarily over relational data are considered, and state-of-the-art techniques, and open problems for these are described.

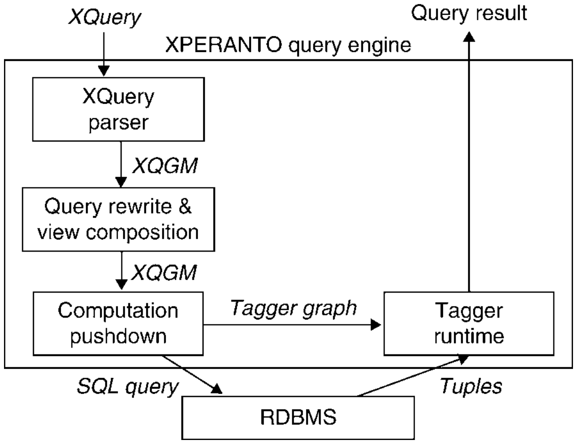
Mapping Between the XML View and the Underlying Data

The mapping between the XML view and the underlying data are used for publishing the data, as well as for performing the user requested actions (such as answering queries) when the view is virtual. Different ways of specifying mappings between the XML view and the underlying data are possible. The *canonical mapping* [13] is a very simple one where there is a 1-1 mapping between the relational tuples and the XML elements in the view. An XML element is constructed for every row in every table in the relational database. Thus the entire data in the relational database is captured in this canonical XML view. Slightly more complex mappings that still capture the entire relational data in the XML view are studied in [8], where the key-foreign key constraints are used to nest XML elements within each other. The translation of queries (especially navigation queries as in XPath) in the above mapping schemes is quite straight forward.

However, often times, a database application needs to publish their data as an XML view that conforms to a standard schema. In such cases, a more complex mapping scheme is needed as all of the underlying data may not be exposed in the view; also the underlying data might need to be restructured to conform to the schema. In [10], the authors study how the user can specify relationships between the underlying schema and the view schema diagrammatically. Based on some assumptions, the system then analyzes the user specified relationships and translates them into meaningful mappings that can be understood by the system (such as SQL queries). Also the user is consulted when there is potential ambiguity. In [5,13], the mapping is specified using XQuery language [15]. This is similar to the scenario that is well-understood by SQL engines (where view definitions are specified in SQL). The database administrator can specify the XML view using an XQuery expression that conforms to the required schema.

User Queries over XML Views

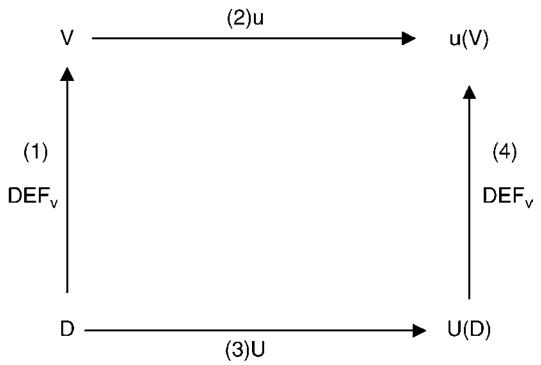
The systems that support XML views need to provide the capability for users to query the data. In [5,13], the authors study how the user queries specified using XQuery can be answered efficiently in the scenario where the XML view is also specified using XQuery. The architecture for such a system as described in [13] is shown in Fig. 1. One of the main assumptions made by these systems is that the SQL engine is best equipped to handle computations efficiently (those computations that an SQL engine can handle); therefore one needs to push down as much computation as possible within the SQL engine. The view composer shown in Fig. 1 composes the user query with the view query. The computation pushdown module then rearranges the combined query plan so that the SQL portion is at the bottom of the query plan, and it includes everything that can be done by the SQL engine. In such a case, the middle-ware only needs to do tagging and this is done in a single pass over the data returned by the SQL engine.



XML Views. Figure 1. Typical architecture for processing user queries over XML views.

User Updates over XML Views

While a lot of work has focussed on how to answer user queries over XML views, very little work has focussed on handling user updates over virtual XML views. As the view is virtual, the view update needs to be performed by updating the base data in such a way that the effect expected by the user is achieved. A common semantics used for view updates is the side-effect free semantics shown in Fig. 2, as described in [3,7]. In the figure, D represents the database instance, DEF_v represents the view definition, V is the view instance, u is the user specified view update. u(V) represents the effect that the user wants to achieve on the view, the view update problem therefore is to find an update U over the base data such that the user desired effect is



XML Views. Figure 2. Illustrating side-effect free semantics.

achieved on the view. It is possible that such an update U, does not exist in which case the view update cannot be performed. In some cases, there could be a unique U, and in other cases, it is possible that multiple such updates over the base data, exist in which case the ambiguity needs to be resolved using heuristics, by the user, or by rejecting the view update.

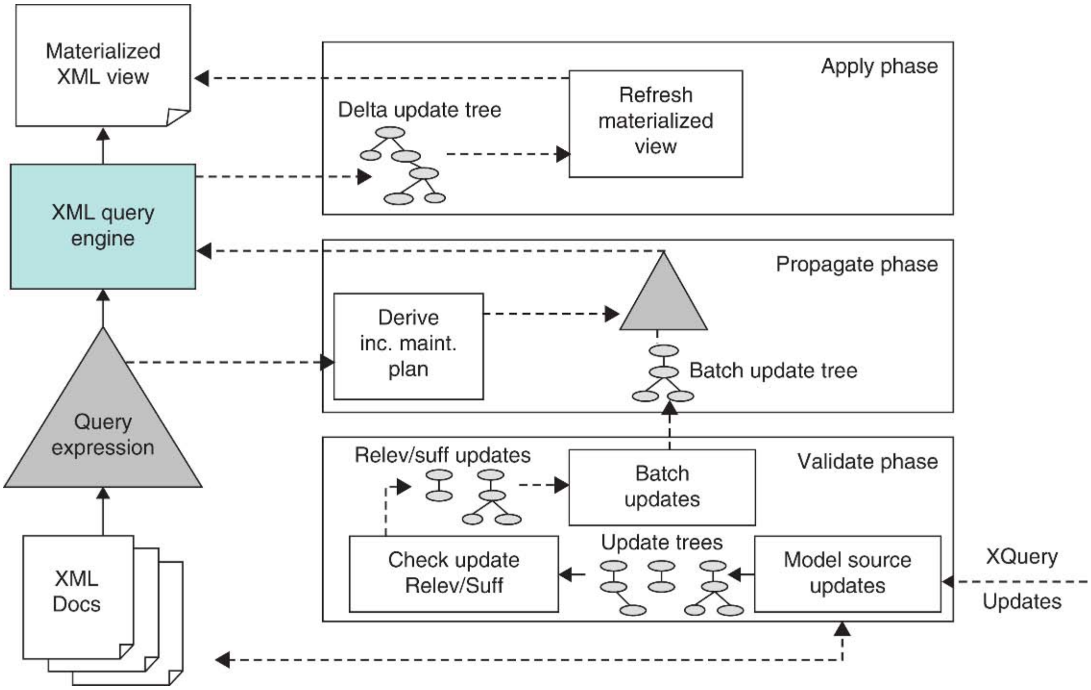
Updating SQL views itself is considered a hard problem, and the existing solutions handle only a subset of the view definitions. When a user specifies updates over view definitions that use "non-permissible" operators (such as aggregation operators), the system rejects these updates. Most solutions, including commercial ones, use a schema level analysis to perform/reject the view update, where they utilize the base schema, the view definition and the user specified update statement. Some solutions also examine the base data, in which case fewer view updates need to be rejected.

The main approaches that study updates over XML views of relational databases include [1,14]. In [1], the authors translate the XML view into a set of SQL views; now the solutions for SQL views can be utilized. In [14], the authors identify that the XML view update problem is harder than the SQL view update problem – SQL view update problem boils down to the case where the XML view schema has only one node. For a general XML view, given an update (such as delete) to be performed on an XML view element, the authors partition the XML view schema nodes into three categories – for one of the categories, the authors utilize the results from the SQL view update research, for the other two categories, the authors propose new approaches to check for side-effects. As follow up work to [14], the authors have studied how data level analysis can be used for the XML view update problem as well.

Maintenance of Materialized XML Views

In order to keep materialized views consistent with the underlying base data, one approach is to recompute the view every time there is a base update. However, this approach is not efficient as the base updates are typically very small compared to the entire base data. It would be efficient if *incremental view maintenance* could instead be performed, where only the update to the view is computed. Incremental view maintenance consists of typically two steps – in the *propagate* phase, the delta change to the view is computed using a *incremental maintenance plan*, and in the *apply* phase, the view is refreshed using this delta change to the view.

Incremental maintenance of XML views is more complex than maintenance of SQL views, because of



XML Views. Figure 3. Architecture for maintaining materialized XML views.

the more complex features of XML including nested structure (a form of aggregation) and order, and of XML query languages such as XQuery. In [4], the authors study how to incrementally maintain XML views defined over underlying data sources that are also XML (note that the solutions apply to the case where the underlying data sources are relational as well). The architecture of their approach is shown in Fig. 3. As the underlying base is XML, the base updates can come in many different granularities – this requires a *validate phase* which combines the update with the base data to make it a complete update. This is then fed to the incremental maintenance plan in the *propagate phase*, which computes the delta change to the view. In the *apply phase*, the view is refreshed using the delta change to the view.

Key Applications

XML views are useful to any database application that wishes to publish their data on the web.

Future Directions

XML views are already being used widely by database applications, and their usage is expected to increase further in future. There are still several issues that need to be studied to make such systems more efficient. For processing queries, query composition will result in several unnecessary joins. Therefore the query optimizer must be able to remove unnecessary joins, this requires the query optimizer to be able to infer key constraints in the query plan. For processing view updates, a combined schema and data analysis promises to be an efficient approach and needs to be investigated. Incremental view maintenance further requires new efficient approaches for handling operations (such as aggregation) present in XML query languages.

Cross-references

- ▶ Top-k XML Query Processing
- ▶ XML Information Integration
- ▶ XML Publishing
- ▶ XML Updates
- ▶ XPath/XQuery

Recommended Reading

1. Braganholo V.P., Davidson S.B., and Heuser C.A. From XML view updates to relational view updates: Old solutions to a new problem. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 276–287.
2. Bray T., Paoli J., Sperberg-McQueen C.M., Maler E., and Yergeau F. Extensible Markup Language (XML) 1.0, W3C Recommendation. Available at: http://www.w3.org/XML
3. Dayal U. and Bernstein P.A. On the correct translation of update operations on relational views. ACM Trans. Database Syst., 7(3):381–416, September 1982.
4. El-Sayed M., Rundensteiner E.A., and Mani M. Incremental maintenance of materialized XQuery views. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 129.
5. Fernandez M., Kadiyska Y., Suciu D., Morishima A., and Tan W.-C. SilkRoute: A framework for publishing relational data in XML. ACM Trans. Database Syst., 27(4):438–493, December 2002.
6. Griffin T. and Libkin L. Incremental maintenance of views with duplicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 328–339.
7. Keller A.M. Algorithms for translating view updates to database updates for views involving selections, projections and joins. In Proc. 4th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1985, pp. 154–163.
8. Lee D., Mani M., Chiu F., and Chu W.W. NeT & CoT: Translating relational schemas to XML schemas using semantic constraints. In Proc. Int. Conf. on Information and Knowledge Management, 2002, pp. 282–290.
9. Palpanas T., Sidle R., Cochrane R., and Pirahesh H. Incremental maintenance for non-distributive aggregate functions. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 802–813.
10. Popa L., Velegrakis Y., Miller R.J., Hernandez M.A., and Fagin R. Translating Web data. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 598–60.
11. Quass D. Maintenance expressions for views with aggregates. In Proc. workshop on Materialized Views: Techniques and Applications, 1996, pp. 110–118.
12. Ramakrishnan R. and Gehrke J. Database Management Systems. McGraw Hill, 2002.
13. Shanmugasundaram J., Kiernan J., Shekita E., Fan C., and Funderburk J. Querying XML views of relational data. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 261–27.
14. Wang L., Rundensteiner E.A., and Mani M. Updating XML views published over relational databases: Towards the existence of a correct update mapping. Doc. Knowl. Eng. J., 58 (3):263–298, 2006.
15. W3C XQuery Working Group. Available at: http://www.w3.org/XML/Query/

XPath/XQuery

Jan Hidders, Jan Paredaens
University of Antwerp, Antwerpen, Belgium

Synonyms

W3C XML path language; W3C XML query language

Definition

XPath (XML path language) and XQuery (XML query language) are query languages defined by the W3C (World Wide Web Consortium) for querying XML documents.

XPath is a language based on path expressions that allows the selection of parts of a given XML document. In addition it also allows some minor computations resulting in values such as strings, numbers or booleans. The semantics of the language is based on a representation of the information content of an XML document as an ordered tree. An XPath expression consist usually of a series of steps that each navigate through this tree in a certain direction and select the nodes in that direction that satisfy certain properties.

XQuery is a declarative, statically typed query language for querying collections of XML documents such as the World Wide Web, a file system or a database. It is based on the same interpretation of XML documents as XPath, and includes XPath as a sublanguage, but adds the possibility to query multiple documents in a collection of XML documents and combine the results into completely new XML fragments.

Historical Background

The development of XPath and XQuery as W3C standards is briefly described in the following.

XPath 1.0 XPath started as an initiative when the W3C XSL working group and the W3C XML Linking working group (which was working on XLink and XPointer) realized that they both needed a language for matching patterns in XML documents. They decided to develop a common language and jointly published a first working draft for XPath 1.0 in July 1999, which resulted in the recommendation [8] in November 1999.

XQuery 1.0 and XPath 2.0 The history of XQuery starts in December 1998 with the organization by W3C of the workshop *QL '98* on XML query languages. This workshop received a lot of attention from the XML, database, and full-text search communities, and as a result the W3C started the XML Query working group [7]. Initially, its goals were only to develop a query language, and a working draft of the requirements was published in January 2000 and a first working draft for XQuery in February 2001. More than 3 years later, in August 2004 the charter of the group was extended with the goal of codeveloping XPath 2.0 with the XSL working group. Finally, in January 2007 the recommendations for both XQuery 1.0 [10] and XPath 2.0 [9] were published.

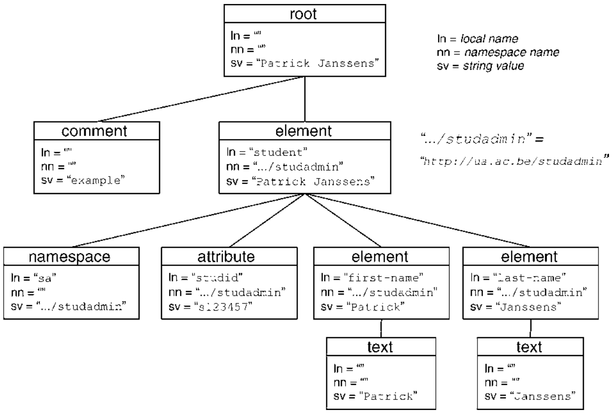
Historical Roots of XQuery 1.0 Most features of XQuery can be traced back to its immediate predecessor Quilt [2]. This language combined ideas from other predecessors such as XPath 1.0 and XQL from which the concept of path expressions was taken. From XML-QL came the idea of using variable bindings in the construction of new values. Older influences where SQL whose SELECT-FROM-WHERE expressions formed the inspiration for the FLWOR expressions, and OQL which showed the benefit of a functional and fully orthogonal query language. Finally there were also influences from other query languages for semi-structured data such as Lorel and YATL.

Foundations

The organization of this section is as follows. First, the XPath 1.0 data model is presented, then the XPath 1.0 language, which is followed by a description of XQuery 1.0 and finally XPath 2.0 is briefly discussed.

The XPath 1.0 Data Model The information content of an XML document is represented by an ordered tree that can contain seven types of nodes: *root* nodes, *element* nodes, *attribute* nodes, *text* nodes, *comment* nodes, *processing instruction* nodes and *namespace* nodes. Three properties can be associated with these nodes: a *local name*, a *namespace name* and a *string-value*. The local name is defined for element, attribute, processing instruction and namespace nodes. For the latter two it represents the target for the processing instruction and the prefix for the namespace, respectively. The namespace name represents the full namespace URI of a node and is defined for the element and attribute nodes. The string-value is defined for all types of nodes but for the root and element node it is derived from the string-values of the nodes under it. For attribute nodes it is the normalized attribute value, for text nodes it is the text content of the node, for processing instruction nodes it is the processing instruction data, for comment nodes it is the text of the comment and for namespace nodes it is the absolute URI for the namespace. An example of an XML document and its corresponding data model is given in Fig. 1. Over the nodes in this tree a *document*

```
<?xml version="1.0"?>
<!-- example -->
<sa:student
    xmlns:sa="http://ua.ac.be/studadmin"
    sa:studid="s1234567" >
  <sa:first-name>Patrick</sa:first-name>
  <sa:last-name>Janssens</sa:last-name>
</sa:student>
```



XPath/XQuery. Figure 1. An XML document and its XPath 1.0 data model.

order is defined that orders the nodes as they are encountered in a pre-order walk of the tree.

The XPath 1.0 Language The most important type of expression in XPath is the *location path* which is a path expression that selects a set of nodes from the tree describing the document. A location path can be relative, in which case it starts navigating from a certain context node, or absolute, in which case it starts from the root node of the document.

In its simplest form a relative location path consist of a number of steps separated by / such as for example `class/student/@id` where `class` and `student` are steps that navigate to children element nodes with those names, and `@id` navigates to attribute nodes with name `id`. It selects all nodes that can be reached from the context node by these steps, i.e., the `id` attribute nodes that are directly below `student` element nodes that are in turn children of `class` element nodes that are children of the context node. The path expression becomes an absolute location path, i.e., it starts navigation from the root node, if it is started with / as in `/class/student/@id`. The wildcards `*` and `@*` can be used to navigate to element nodes and attribute nodes, respectively, with any name.

The steps can also be separated by `//` which means that the path expression navigates to the current node and all its descendants before it applies the next step. So `class//first-name` navigates to all `first-name` elements that are directly or indirectly below a `class` element directly below the context node. A special step is the self step which is denoted as . and remains in the same place, so `class//.` returns the `class` element node and all the nodes below it. Another special step is the parent step which is denoted as `..` and navigates to the parent of the current context node. A location path can also start with `//` which means that it is an absolute location path whose first step navigates to the root node and all the nodes below it. For example, `//student` will select all `student` element nodes in the document. Path expressions can also be combined with the union operator, denoted as `|`, which takes the union of the results of two path expressions. For example, `//(student | teacher)/(last-name | first-name)` returns the first and last names of all students and teachers.

With each step zero or more predicates can be specified that must hold in order for nodes to be selected. Such predicates are indicated in square brackets such as for example in `//student[@id=''s123456'']/grade` which select the `grade` elements below `student` elements with a certain `id` attribute value. Conditions that can be specified include (i) comparisons between the string-values of the results of path expressions and other expressions, (ii) existential predicates that check whether a certain path expression returns a nonempty result and (iii) positional predicates that check the position of the current node in the result of the step for the context node. Comparisons between path expressions that return more than one node have an existential semantics, i.e., the comparison as assumed to hold if at least one of the returned nodes satisfies the equation. For example, `//course[enrolled/student/last-name=''Janssen'']` returns the courses in which at least one student with the last name "Janssen" enrolled. Available comparison operators include `=`, `!=` (not equals), < and <=. An example of an existential predicate is //course[enrolled/student] that selects courses in which at least one student enrolled. Finally, an example of a positional predicate is given in //course/teacher[1]/name that selects for each course the name of the first teacher of that course. If there are multiple predicates then a positional predicate takes the preceding predicates into account. For example, //course[subject=''databases''][1] selects the first of the courses that have databases as their subject. The comparisons and existential conditions can be combined with and and or operations, and the not() function.

For path expressions that have to navigate over more types of nodes and require other navigation steps a more general syntax is available. In this syntax a single step is specified as axis::node-test followed by zero or more predicates. Here axis specifies the direction of navigation and node-test a test on the name or the type of the node that has to be satisfied. There are 13 possible navigation axes: child, attribute, descendant, descendant-or-self, parent, ancestor, ancestor-or-self, following, preceding, following-sibling, preceding-sibling self and namespace. The following axis navigates to all nodes that are larger in document order but not descendants, and the following-sibling axis navigates to all siblings that are larger in document order. Possible node-tests are names, the * wild-card and tests for specific node types such as comment(), text(), processing-instruction() or node() which matches all nodes. Finally two special functions position() and last() can be used in predicates and denote the position of the current node in the result of the step and the size of that result, respectively. To illustrate, the location path /class//student[1] can also be written in this syntax as /child::class/descendant-or-self::node()/child::student[position()=1].

Next to the operators for navigation XPath also has functions for value manipulation of node sets, strings, booleans and numbers. This includes the arithmetic operators +, *, −, div and mod, aggregation operators such as count() and sum(), string manipulation such as string concatenation, space normalization, substring manipulation, etc., type conversion operators such as string(), number() and boolean(), and finally functions that retrieve properties of nodes such as name(), string() (also overloaded for type conversion) and namespace-uri().

XQuery 1.0 The data model for XML documents is for XQuery 1.0 largely the same as for XPath 1.0. The main changes are that the root of a document is represented by a *document node*, and that typed values are associated with nodes. The types of these typed values include the built-in types from XML Schema and are associates with all values computed in XQuery. However, for the sake of brevity these types are mostly ignored here.

Another important change is that the fundamental data structure is no longer sets of nodes but ordered sequences of nodes and atomic values as defined by XML Schema. All results of expressions are such sequences and single values such as 1 and ''mary'' are always interpreted as singleton sequences. Note that these sequences are flat, i.e., they cannot contain nested sequences. The concatenation of sequences s_1 and s_2 is denoted as s_1, s_2. So the expression (''a,'' ''b'') denotes in fact the concatenation of the singleton sequence ''a'' and the singleton sequence ''b.'' Therefore (1, (2, 3)) is equivalent to (1, 2, 3).

The path expressions as defined in XPath 1.0 are almost all included and have mostly the same semantics. An important difference is that they do not return a set of nodes but a sequence of nodes sorted in document order. Several extensions to path expressions are introduced. Next to the union operator that is equivalent with the | operator in XPath 1.0, there is also the intersect and difference operators that correspond to the set intersection and set difference. These also return their results sorted in document order. The aggregation functions such as count() and sum() are naturally generalized as fn:count() and fn:sum() for sequences, and new ones such as fn:min(), fn:max() and fn:avg() are added. Note that built-in functions in XQuery are prefixed with a namespace, usually fn. Next to the old value comparisons new types of comparisons are introduced, such as is and is not that compare the node identity of two nodes, and << that checks precedence in document order. For navigating over references, the fn:id() function is introduced that given an identifier retrieves the element node with that identifier, and the function fn:idref() that given an identifier retrieves the element nodes that refer to this identifier.

The access to collections of XML documents is provided by the functions fn:doc() and fn:collection() that both expect as argument a string containing a URI. These functions retrieve the requested

XML fragments associated with this URI and, if this was not already done before, construct their data models and return a document node or a sequence of nodes that are the roots of the fragments. An example of their use would be `fn:doc(``courses.xml'')/course[@code=``DB201'']/enrolled/student` that retrieves the students enrolled in the course `DB201` from the file `courses.xml`.

The core expressions of XQuery are the FLWOR expressions which are illustrated by the following example:

```
for $s in fn:doc(students.xml)//student,
   $e  in  fn:doc(``enrollments.xml'')//
    enrollment
let $cn := fn:doc(``courses.xml'')//course
[@crs-code=e/@crs-code]/name
where $s/@stud-id = $e/@stud-id
order by $cn
return <enroll> {$s/name ,$cn} </enroll>
```

A FLWOR expression starts with one or more `for` and `let` clauses that each bind one or more variables (that always start with `$`). The `for` clause binds variables such that they iterate over the elements of the result sequence of an expression, and the `let` clause binds the variable to the entire sequence. This is followed by an optional `where` clause with a selection condition, an optional `order by` clause that specifies a list of sorting criteria and a `return` clause that contains an expression that constructs the result. In the example the `for` clause binds the variable `$s` such that it iterates over the student elements in the file `students.xml` in the order that they appear there. Then, for each of those, it binds the variables `$e` such that it iterates over all the enrollments in the file `enrollments.xml` in the order that they appear there. For every binding of the variables it evaluates the `let` clause where it binds `$cn` with the name of the course in the enrollment `$e`. Then it selects those combinations for which the condition in the `where` clause is true, i.e., if the student `$s` belongs to the enrollment `$e`. The resulting bindings are sorted by the `order by` clause on the course name in `$cn`. Finally, the `return` clause creates for each binding in the result of the preceding clause an `enroll` element that contains the `name` element of student `$s` and the `name` element in `$cn`. Note that if there had been no `order by` clause then the result would have been sorted in the order that the students are listed in `students.xml`.

An additional optional feature for `for` clauses is the `at` clause that binds an extra variable to the position of the current binding. For example, the expression `for $x at $p in (``a,'' ``b,'' ``c'') return ($p, $x)` returns the sequence `(1,``a,'' 2,``b,'' 3,``c'')`.

New nodes can be constructed in two ways. The first is the direct constructor which is demonstrated in the first FLWOR example and consists of literal XML with embedded XQuery expression between curly braces. For the literal XML the corresponding nodes are created and deep copies of the results of the expressions are inserted into that. This can be used to compute the content and the attribute values of the new node such as in `<airport code=``{$x/code}''>{$x/full-name}</airport>`. An alternative that also allows computation of the element name are the computed element constructors of the form element{e_1}{e_2} that construct a new element node with the name as computed by e_1 and the content, i.e., all nodes directly below it, deep copies of those computed by e_2. For all types of nodes, except namespace nodes, are such computed constructors available.

XQuery also adds conditional expressions such as an if (e_1) then e_2 else e_3, and type switches of the form typeswitch(e) case t_1 return e_1 case t_2 return e_2 ... default return e_d that returns the result of the first e_i such that the result of e matches type t_i or the result of e_d if none of the types match. It also has logical quantifiers such as some \$$v$ in e_1 satisfies e_2 and every \$$v$ in e_1 satisfies e_2.

In order to allow query optimization techniques for unordered data formats there is a function `fn:unordered()` that allows the user to indicate that the ordering of a result is of no importance. In addition, there is a global parameter called the `ordering mode` such that when declared as `unordered` it means that, informally stated, the path expressions may produce unordered results and FLWOR expressions without an `order by` clause may change the iteration order. This ordering mode can be reset locally for an expression e with the statements unordered{e} and ordered{e}.

A very powerful feature is the possibility to start a query with a list of possibly recursive function definitions. For example, `declare function countElem($s){fn:count($s/self::element()) + fn:sum(for $e in $s/* return countElem($e))}` defines a function that counts the number of element nodes in a fragment. This feature makes XQuery Turing complete and gives it the expressive power of a full-blown programming language. Finally, there is a wide range of predefined functions for the manipulation and conversion of atomic values as defined in XML Schema, and functions for sequences such as `fn:distinct-values()` that removes duplicate values, `fn:reverse()` that reverses a sequence and `fn:deep-equal()` that tests if two sequences are deep equal.

XPath 2.0 This version of XPath is based on the same data model as XQuery 1.0 and is semantically and syntactically a subset of XQuery 1.0. The main omissions are (i) user-defined functions, (ii) all clauses in FLWOR expressions except the `for` clause without `at` and the `return` clause, (iii) the node constructors and (iv) the `typeswitch` expression.

Key Applications

The XPath language is used in several W3C standards such as DOM (Level 3), XSL, XLink, XPointer, XML Schema and XForms, and also in ISO standards such as Schematron. Most programming languages that offer some form of support for XML manipulation also support XPath for identifying parts of XML fragments. This includes C/C++, Java, Perl, PHP, Python, Ruby, Schema and the .Net framework.

The usage of the XQuery language can be roughly categorized into three different but not completely disjoint areas, for which different types of implementations are available. The first is that of *standalone XML processing*, where the XML data that is to be queried and transformed consists of documents stored on a file system or the World Wide Web, and these data are processed by a standalone XQuery engine. The second area is that of *database XML processing* where the XML documents are stored in an XML-enabled DBMS that has an integrated XQuery engine. The final and third area is that of *XML-based data integration* where data from different XML and non-XML sources are integrated into an XML view that can be queried and transformed with XQuery.

For each of the mentioned areas the XQuery engine faces different challenges. For example, for database XML processing it must optimally use the data structures provided by the DBMS, which might have an XML-specific storage engine, a relational storage engine or a mixture of both. On the other hand, for data integration it may be more important to determine how to optimally combine data from streaming and non-streaming sources and how to recognize and delegate query processing tasks to the data sources that have those capabilities themselves. As a consequence, different XQuery engines are often specialized in one of the mentioned application areas.

Future Directions

Since XPath 2.0 and XQuery 1.0 have become W3C recommendations, the involved working groups have continued to work on several extensions of these languages:

The XQuery Update Facility This extension adds update operations to XQuery. It allows expressions such as

```
for $s in /inventory/clothes/shirt[@size =
"XXL"]
return do replace value of $s/@price with
$s/@price - 10
```

that combine the XQuery syntax with update operations such that multiple elements can be updated at once. A last call working draft for the XQuery Update facility was published by W3C in August 2007.

XQuery 1.1 and XPath 2.1 In March 2007 the XML Query working group published a first version of the XML Query 1.1 requirements, and they plan to produce with the XSL working group the requirements for XPath 2.1. The proposed extensions for XQuery include grouping on values, grouping on position, calling external functions that for example, invoke web services, adding explicit node references that can be used as content and can be dereferenced, and finally higher order functions.

XQuery 1.0 and XPath 2.0 Full-Text This extension adds constructs for doing full text searches on selections of documents, text-search scoring variables that can be used in FLWOR expressions, and full-text matching options that can be defined in the query prolog. A last call working draft was published in May 2007.

XQuery Scripting Extensions This adds imperative features such that the resulting language can be more readily used for tasks that would otherwise typically be accomplished using an imperative language with XML capabilities. Proposed extensions include constructs for controlling the order of computation in FLWOR expressions, operations that cause side effects such as variable assignments, and operations that observe these side effects. A first working draft describing the requirements for this extension was published by the XML Query working group in March 2007.

Cross-references

- ▶ XML Information Integration
- ▶ XML Programming
- ▶ XML Storage
- ▶ XML Tuple Algebra
- ▶ XML Updates
- ▶ XML Views
- ▶ XQuery Full-Text
- ▶ XQuery Processors
- ▶ XSL/XSLT

Recommended Reading

1. Brundage M. XQuery: The XML Query Language. Pearson Higher Education, Addison-Wesley, Reading, MA, USA, 2004.
2. Chamberlin D.D., Robie J., and Florescu D. Quilt: An XML query language for heterogeneous data sources. In Proc. 3rd Int. Workshop on the World Wide Web and Databases, 2000, pp. 53–62.
3. Hidders J., Paredaens J., Vercammen R., and Demeyer S. A light but formal introduction to XQuery. In Database and XML Technologies, 2nd Int. XML Database Symp., 2004, pp. 5–20.
4. Katz H., Chamberlin D., Kay M., Wadler P., and Draper D. XQuery from the experts: A guide to the W3C XML query language. Addison-Wesley Longman, Boston, MA, USA, 2003.
5. Melton J. and Buxton S. Querying XML: XQuery, XPath, and SQL/XML in context. Morgan Kaufmann, San Francisco, CA, USA, 2006.
6. Walmsley P. XQuery. O'Reilly Media, 2007.
7. W3C. W3C XML query (XQuery). http://www.w3.org/XML/Query/.
8. W3C. XML path language (XPath), version 1.0, W3C recommendation 16 November 1999. http://www.w3.org/TR/xpath/, November 1999.
9. W3C. XML path language (XPath) 2.0, W3C recommendation 23 January 2007. http://www.w3.org/TR/xpath20/ January 2007.
10. W3C. XQuery 1.0: An XML query language, W3C recommendation 23 January 2007. http://www.w3.org/TR/xquery/ January 2007.

XPDL

- ▶ Workflow Management Coalition
- ▶ XML Process Definition Language

XQFT

- ▶ XQuery Full-Text

XQuery 1.0 and XPath 2.0 Full-Text

- ▶ XQuery Full-Text

XQuery Compiler

- ▶ XQuery Processors

XQuery Full-Text

Chavdar Botev[1], Jayavel Shanmugasundaram[2]
[1]Yahoo Research!, Cornell University, Ithaca, NY, USA
[2]Yahoo Research!, Santa Clara, USA

Synonyms

XQuery 1.0 and XPath 2.0 Full-Text; XQFT

Definition

XQuery Full-Text [11] is a full-text search extension to the XQuery 1.0 [9] and XPath 2.0 [8] XML query languages. XQuery 1.0, XPath 2.0, and XQuery Full-Text are query languages developed by the World Wide Web Consortium (W3C).

Historical Background

The XQuery [9] and XPath languages [8] have evolved as powerful languages for querying XML documents. While these languages provide sophisticated structured query capabilities, they only provide rudimentary capabilities for querying the text (unstructured) parts of XML documents. In particular, the main full-text search predicate in these languages is the `fn:contains`

`($context, $keywords)` function (*http://www.w3.org/TR/xpath-functions/#func-contains*), which intuitively returns the Boolean value `true` if the items in the `$context` parameter contain the strings in the `$keywords` parameter. The `fn:contains` function is sufficient for simple sub-string matching but does not provide more complex search capabilities. For example, it cannot support queries like "*Find titles of books (//book/title) which include "xquery" and "full-text" within five words of each other, ignoring capitalization of letters.*" Furthermore, XQuery does not support the concept of relevance scoring, which is very important in the area of full-text search.

To address these short-comings, W3C has formulated the XQuery 1.0 and XPath 2.0 Full-Text 1.0 Requirements [12]. These requirements specify a number of features that must, should or may be supported by full-text search extensions to the XQuery and XPath languages. These include the level of integration with XQuery 1.0 and XPath 2.0, support for relevance scoring, and extensibility as the most important features of such extensions.

W3C has also described a number of use cases for full-text search within XML documents that occur frequently in practice. These use cases can be found in the document XQuery 1.0 and XPath 2.0 Full-Text 1.0 Use Cases [13]. The use cases contain a wide range of scenarios for full-text search that vary from simple word and phrase queries to complex queries that involve word proximity predicates, word ordering predicates, use of thesauri, stop words, stemming, etc.

The XQuery Full-Text language has been designed to meet both the XQuery 1.0 and XPath 2.0 Full-Text 1.0 Requirements and the XQuery 1.0 and XPath 2.0 Full-Text 1.0 Use Cases.

XQuery Full-Text is also related to previous work on full-text search languages for semi-structured data: ELIXIR [5], JuruXML [4], TEXQuery [3], TiX [1], XIRQL [6], and XXL [7].

Foundations

XQuery Full-Text provides a full range of query primitives (also known as *selections*) that facilitate the search within the textual content of XML documents. The textual content is represented as a series of *tokens* which are the basic units to be searched. Intuitively, a token is a character, n-gram, or sequence of characters. An ordered sequence of tokens that should occur in the document together is referred to as a *phrase*. The process of converting the textual content of XML documents to a sequence of tokens is known as *tokenization*.

XQuery Full-Text proposes four major features for support of full-text search the XQuery and XPath query languages:

1. Tight integration with the existing XQuery and XPath syntax and semantics
2. Query primitives for support of complex full-text search in XML documents
3. A formal model for representing the semantics of full-text search
4. Support for relevance scoring.

Each of these features is discussed below.

XQuery Full-Text Integration

One of the main design paradigms of XQuery Full-Text is the tight integration with the XQuery and XPath query languages. The integration is both on the syntax and data-model levels.

The syntax level integration is achieved through the introduction of a new XQuery expression, the *FTContainsExpr*. The *FTContainsExpr* acts as a regular XQuery expression and thus, it is fully composable with the rest of the XQuery and XPath expressions. The basic syntax of the *FTContainsExpr* is:

```
Expr ''ftcontains'' FTSelection
```

The XQuery expression *Expr* on the left-hand side is called the *search context*. It describes the nodes from the XML documents that need to be matched against the full-text search query described by the *FTSelection* on the right-hand side. Expr can be any XQuery/XPath expression that returns a sequence of nodes. The syntax of the *FTSelection* will be described later. The entire *FTContainsExpr* returns `true` if some node in *Expr* matches the full-text search query *FTSelection*. Note, that since *FTContainsExpr* returns results within the XQuery data model, *FTContainsExpr* can be arbitrary nested within XQuery expressions. Consider the following simple example.

```
//book[./title ftcontains ''informa-
tion'' ftand ''retrieval'']//author
```

The above example returns the authors of books whose title contains the tokens "information" and "retrieval." The expression `./title` defines the search context for the *FTContainsExpr* in predicate expression within the brackets [] which is the `title` child element node of the current `book` element node. The *FTSelection* “`information'' ftand ``retrieval''` specifies that book titles must contain the tokens “information” and “retrieval” to be considered a match.

Integration can be achieved not only for XQuery Full-Text expressions within XQuery/XPath expressions but vice versa. Consider the following example.

```
   for $color in (``red,'' ``yellow'')
   for $car in (``ferrari,''
   ``lamborghini,'' ``maserati'')
   return //offer[. ftcontains
{$color, $car} phrase]
```

The above query returns `offer` element nodes which contain any of a number of combinations of colors and cars as a phrase. The *FTContainsExpr* will match phrases like “red lamborghini” or “yellow ferrari” but it will not match “red porsche” or “yellow rusty ferrari.”

XQuery Full-Text Query Primitives

This section describes the basic XQuery Full-Text query primitives and how they can be combined to construct complex full-text search queries.

The XQuery Full-Text query primitives include token and phrase search, token ordering, token proximity, token scope, match cardinality, and Boolean combinations of the previous. Further, XQuery Full-Text allows control over the natural language used in the queried documents, the letter case in matched tokens, and the use of diacritics, stemming, thesauri, stop words, and regular-expression wildcards though the use of *match options*.

The XQuery Full-Text query primitives are highly composable within each other. This allows for the construction of complex full-text search queries. The remainder of this section will briefly describe some of the available query primitives and show how they can be composed.

The most basic query primitive is to match a sequence of tokens also known as *FTWords*. For example, the above car offer query can also be written as:

```
   //offer[. ftcontains {``red ferrari,''
   ``yellow ferrari,''
   ``red lamborghini,'' ``yellow lambor-
   ghini,'' ``red maserati,''
   ``yellow maserati''} any]
```

The above *FTContainsExpr* matches `offer` nodes which contain any of the listed phrases. The `any` option specifies that it is sufficient to match a single phrase within an `offer` node. It is also possible to use the option `all` which specifies that all phrases should be matched within a single node. As it was shown earlier, the `phrase` options specifies that all nested phrases should be combined into a single phrase. Other possible options are `any word` or `all word` which specify that the nested phrases need to be first broken into separate tokens before applying the respective disjunctive or conjunctive match semantics. For example, the expression

```
   //offer[. ftcontains {``red ferrari,''
   ``yellow lamborghini''} all word]
```

is equivalent to

```
   //offer[. ftcontains {``red,'' ``fer-
   rari,'' ``yellow,'' ``lamborghini''}
   all]
```

Two other basic query primitives are the ability to restrict the proximity of the matched tokens. There are two flavors of proximity predicates. The *FTWindow* primitive specifies that the matched tokens have to be within a window of a specified size. Consider the example

```
   //offer[. ftcontains {``red ferrari,''
   ``yellow lamborghini''} all
```

window at least 6 words]

The above expression will return `offer` nodes which contain, say, “red ferrari and brand new yellow lamborghini” because both phrases occur within a window of seven words, i.e., the window matches the *FTWindow* size restriction of at least six words. The expression will not match nodes which contain “red ferrari and yellow lamborghini” because the phrases occur within a window of five words.

The other flavor of proximity primitive is *FTDistance*. It can be used to restrict the number of intervening tokens between the matched tokens or phrases. For example, consider the expression

```
   //offer[. ftcontains {``new,'' ``brand,''
   ``ferrari''} all
```

distance at most 1 word]

The above expression will return `offer` nodes which contain the specified query tokens with at most one intervening token between consecutive occurrences of the query tokens. For example, the expression will return nodes which contain "brand new red ferrari" because there are no intervening tokens between "brand" and "new" and one intervening token between "new" and "red." On the other hand, the expression will not return nodes which contain "new car of the ferrari brand" because there are three intervening tokens between "new" and "ferrari."

XQuery Full-Text allows also the specification of the order of the matched tokens using the *FTOrder* primitive.

```
//offer[. ftcontains {''new,'' ''brand,''
''ferrari''} all
```

distance at most 1 word ordered]

The above query expands the previous *FTDistance* example by specifying that the query tokens can occur in the nodes only in the specified order.

More complex query expressions can be built using *FTAnd*, *FTOr*, *FTUnaryNot*, and *FTMildNot*. They allow for the combination of other *FTSelections* into conjunctions, disjunctions, and negations. Here is a complex example using *FTAnd* and some of the *FTSelections* introduced earlier.

```
//offer[. ftcontains
(({''red,'' ''ferrari''} all
window at most 3 words)
ftand
({''yellow,'' ''lamborghini''} all win-
dow 3 words))
window at most 20 words]
```

The above expression specifies that the resulting `offer` nodes must contain "red" and "ferrari" within a window of 3 words, "yellow" and "lamborghini" within a window of 3 words, and all of them within a window of 20 words.

```
//offer[. ftcontains ''red ferrari''
ftnot ''yellow lamborghini'']
```

This example of *FTNot* will return `offer` nodes which contain the phrase "red ferrari" but not the phrase "yellow lamborghini." Sometimes, this kind of negation can be too strict. For example, consider a query that looks for articles about the country Mexico but not the state New Mexico.

```
//article[. ftcontains ''Mexico'' ftnot
''New Mexico'']
```

The above query will not return articles which talk about both the country Mexico and the state "New Mexico." The query can be rewritten using *FTMildNot*.

```
//article[. ftcontains ''Mexico'' not in
''New Mexico'']
```

Intuitively, the above query specifies that the user wants articles where the token "Mexico" has occurrences not part of the phrase New Mexico (although there are still be occurrences of the phrase New Mexico).

The power of XQuery Full-Text queries can be further increased with the use of match options that control the way tokens are matched within the document. For example, if the user wants to find offers containing "FERRARI" (all capital letters), she can use the "uppercase" match option:

```
//offer[. ftcontains
''ferrari'' uppercase]
```

This query will match only tokens "FERRARI" regardless of how the token is specified in the query.

Match options can be used to encompass several *FTSelections*. For example, consider the query

```
//article[. ftcontains (''car'' ftand
''aircraft'')
window at most 50 words
with thesaurus at ''http://acme.org/
thesauri/Synonyms"]
```

The above query searches for articles that contain the tokens "car" and "aircraft" or their synonyms using the thesaurus identified by the URI "http://acme.org/thesauri/Synonyms." The matched tokens have to be within a window of at most 50 words.

As mentioned in the beginning of this section, there are other *FTSelections* and match options provided by XQuery Full-Text. The description of all of these features goes beyond the scope of the article. The reader is referred to the complete specification of the language available at [11].

This section will finalize the brief overview of the language primitives in XQuery Full-Text with the description of the feature *extension points*. Extension points be used to provide additional implementation-defined full-text search functionality. An example of the use of such an extension can be found below.

```
declare namespace acmeimpl = ''http://
acme.org/AcmeXQFTImplementation;''
//book/author[name ftcontains (#
acmeimpl:use-entity-index#){''IBM''}]
```

The above example shows the use of the extension use-entity-index provided by the ACME implementation of XQuery Full-Text. It directs the implementation to use an entity index for the token "IBM" so it can also match other references to the same entity, such as "International Business Machines" or even "Big Blue." If the above query is evaluated using another implementation the use-entity-index extension will be ignored.

XQuery Full-Text Formal Model

Amer-Yahia et al. [3] have shown that the XQuery 1.0 and XPath 2.0 Data Model (XDM) [10] is inadequate to support the composability of the complex full-text query primitives described in the previous section. Intuitively, XDM represents data only about entire XML nodes but for sub-node entities like the positions of the tokens within a node. This precludes, for example, the evaluation of nested proximity *FTSelectionsFTWindow* and/or *FTDistance* such as the ones used in the *FTAnd* example.

Therefore, XQuery Full-Text needs a more precise data model that can support the complex full-text search operations and their composability. The language specification [11] introduces the *AllMatches* model. The result of every *FTSelection* applied on a node from the search context can be represented as an *AllMatches* object within this model.

The *AllMatches* object has a hierarchical structure. Each *AllMatches* object consists of zero, one, or more *Match* objects. Intuitively, a *Match* object describes one set of positions of tokens in the node that match the corresponding *FTSelection*. The *AllMatches* contains all such possible *Match* objects. The positions of tokens in a match are described using *TokenInfo* objects which contain data about the relative position of the token in the node and to which query token it corresponds. This model is sufficient to describe the semantics of all *FTSelections*.

The semantics of every *FTSelection* can be represented as a function that takes as input one or more *AllMatches* objects from its nested *FTSelections* and produces an output *AllMatches* object. The remainder of this section will illustrate the process. The illustration will use some simplifications not to burden the exposition with excessive details. Nevertheless, the example below will illustrate some of the basic techniques in the workings of the model.

Consider the simple query

```
//offer[. ftcontains {``red,'' ``ferrari''} all window at most 3 words]
```

Let's assume that the token "red" occurs in the current node from the search context in relative positions 5 and 10 and the token "ferrari" occurs in relative positions 7 and 25. The *AllMatches* objects for each of these tokens contain two *Match* objects – one for each position of occurrence:

```
``red'': AllMatches { Match
{TokenInfo{token:``red,'' pos:5}},
Match{TokenInfo{token:``red,''
pos:10}}}
``ferrari'': AllMatches { Match{TokenInfo{token:``ferrari,'' pos:7}},
Match{TokenInfo{token:``red,''
pos:25}}}
```

To obtain the *AllMatches* for the *FTWords* {``red,'' ``ferrari''} all, each pair of positions for "red" and "ferrari" has to be combined into a single *Match* object.

```
(``red,'' ``ferrari'') all:
AllMatches { Match
{TokenInfo{token: ``red,'' pos:5}},
TokenInfo{token:``ferrari,'' pos:7}},
Match{TokenInfo{token:``red,''
pos:10},
TokenInfo{token:``ferrari,'' pos:7}},
Match{TokenInfo{token:``red,''
pos:5}},
TokenInfo{token:``red,'' pos:25}},
Match{TokenInfo{token:``red,''
pos:10}},
TokenInfo{token:``red,'' pos:25}}}
```

Finally, to obtain the *AllMatches* for the outer most *FTWindow*, those *Matches* which violate the window size condition have to be filtered out. Only the first *Match* satisfies it: the window sizes are 3, 4, 21, and 16 respectively. Therefore, the final *AllMatches* is:

```
(``red,'' ``ferrari'') all window at most
3 words:
AllMatches { Match{TokenInfo{token:
``red,'' pos:5}},
TokenInfo{token:``ferrari,'' pos:7}}}
```

The resulting *AllMatches* object is non-empty and therefore, the current node from the search context satisfies the *FTSelection* and the *FTContainsExpr* should return `true`.

The semantic of every *FTSelection* can be defined in terms of similar operations on *AllMatches* objects. The description of all such operations goes beyond the scope of this article. The reader is referred to the language specification [11] for further details.

Relevance Scoring in XQuery Full-Text

One of the core features of full-text search is the ability to rank the results in the order of their decreasing relevance. Usually, the relevance of a result is represented using a number called a *score* of the result. Higher scores represent more relevant results.

To support scores and relevance ranking, XQuery Full-Text extends the XQuery language with the support for *score variables*. Score variables give access to the underlying scores of the results of the evaluation of an XQuery Full-Text expression. Score variables can be introduced as part of a for- or let-clause in a FLWOR expression. Here is a simple example

```
for $o score $s in //offer[. ftcontains
{‘‘red,’’ ‘‘ferrari’’}
all window at most 3 words]
order by $s descending
return <offer id=‘‘
{$o/@id}’’ score=‘‘{$s}’’ />
```

The above is a fairly typical query iterates over all offers that contain "red" and "ferrari" within a window of three words using the `$o` variable, binds the score of each offer to the `$s` variable, and uses those variables to return the id of offers and their scores in order of decreasing relevance.

Score variables can also be bound in let clauses. This allows for the use of a scoring condition that is different than the one uses for filtering the objects. Consider the following example.

```
for $res at $p in (
for $o in //offer[. ftcontains {‘‘red,’’
‘‘ferrari’’}all ]
let score $s := $o ftcontains {‘‘excel-
lent,’’ ‘‘condition’’}
all window at most 10 words
order by $s descending
return $o)
where $p <= 10
return $res
```

The above query obtains the top 10 offers for red Ferrari's whose relevance is estimated using the full-text search condition `‘‘excellent condition’’ all window at most 10 words`.

It should be noted that XQuery Full-Text does not specify how scoring should be implemented. Each XQuery Full-Text implementation can use a scoring method of their choice as long the generated scores are in the range [0,1] and higher scores denote greater relevance.

Key Applications

Support for full-text search in XML documents.

Cross-references

▶ Full-Text Search
▶ Information Retrieval
▶ XML
▶ XPath/XQuery

Recommended Reading

1. Al-Khalifa S., Yu C., and Jagadish H. Querying structured text in an XML database. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 4–15.
2. Amer-Yahia S., Botev C., Doerre J., and Shanmugasundaram J. XQuery full-text extensions explained. IBM Syst. J., 45 (2):335–351, 2006.
3. Amer-Yahia S., Botev C., and Shanmugasundaram J. TE XQuery: A full-text search extension to XQuery. In Proc. 12th Int. World Wide Web Conference, 2004, pp. 583–594.
4. Carmel D., Maarek Y., Mandelbrod M., Mass Y., and Soffer A. Searching XML documents via XML fragments. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 151–158.
5. Chinenyanga T. and Kushmerick N. Expressive and efficient ranked querying of XML data. In Proc. 4th Int. Workshop on the World Wide Web and Databases, 2001, pp. 1–6.
6. Fuhr N. and Grossjohann K. XIRQL: An extension of XQL for information retrieval. In Proc. ACM SIGIR Workshop on XML and Information Retrieval, 2000, pp. 172–180.
7. Theobald A. and Weikum G. The index-based XXL search engine for querying XML data with relevance ranking. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 477–495.
8. XML Path Language (XPath) 2.0. W3C Recommendation. Available at: http://www.w3.org/TR/xpath20/
9. XQuery 1.0: An XML Query Language. W3C Recommendation. Available at: http://www.w3.org/TR/xquery/
10. XQuery 1.0 and XPath 2.0 Data Model (XDM). W3C Recommendation. Available at: http://www.w3.org/TR/xpath-datamodel/

11. XQuery 1.0 and XPath 2.0 Full-Text 1.0. W3C Working Draft. Available at: http://www.w3.org/TR/xpath-full-text-10/
12. XQuery 1.0 and XPath 2.0 Full-Text 1.0 Requirements. W3C Working Draft. Available at: http://www.w3.org/TR/xpath-full-text-10-requirements/
13. XQuery 1.0 and XPath 2.0 Full-Text 1.0 Use Cases. W3C Working Draft. Available at: http://www.w3.org/TR/xpath-full-text-10-use-cases/

XQuery Interpreter

► XQuery Processors

XQuery Processors

Torsten Grust[1], H. V. Jagadish[2], Fatma Özcan[3], Cong Yu[4]
[1]University of Tübingen, Tübingen, Germany
[2]University of Michigan, Ann Arbor, MI, USA
[3]IBM Almaden Research Center, San Jose, CA, USA
[4]Yahoo! Research, New York, NY, USA

Synonyms

XML database system; XQuery compiler; XQuery interpreter

Definition

XQuery processors are systems for efficient storage and retrieval of XML data using XML queries written in the XQuery language. A typical XQuery processor includes the data model, which dictates the storage component; the query model, which defines how queries are processed; and the optimization modules, which leverage various algorithmic and indexing techniques to improve the performance of query processing.

Historical Background

The first W3C working draft of XQuery was published in early 2001 by a group of industrial experts. It is heavily influenced by several earlier XML query languages including Lorel, Quilt, XML-QL, and XQL. XQuery is a strongly-typed functional language, whose basic principals include simplicity, compositionality, closure, schema conformance, XPath compatibility, generality and completeness. Its type system is based on XML schema, and it contains XPath language as a subset. Over the years, several software vendors have developed products based on XQuery in varying degrees of conformance. A current list of XQuery implementations is maintained on the W3 XML Query working group's homepage (http://www.w3.org/XML/XQuery). There are three main approaches of XQuery processors. The first one is to leverage existing relational database systems as much as possible, and evaluate XQuery in a purely relational way by translating queries into SQL queries, evaluating them using SQL database engine, and reformatting the output tuples back into XML results. The second approach is to retain the native structure of the XML data both in the storage component and during the query evaluation process. One native XQuery processor is Michael Kay's Saxon, which provides one of the most complete and conforming implementations of the language available at the time of writing. Compared with the first approach, this native approach avoids the overhead of translating back and forth between XML and relational structures, but it also faces the significant challenge of designing new indexing and evaluation techniques. Finally, the third approach is a hybrid style that integrates native XML storage and XPath navigation techniques with existing relational techniques for query processing.

Foundations

Pathfinder: Purely Relational XQuery

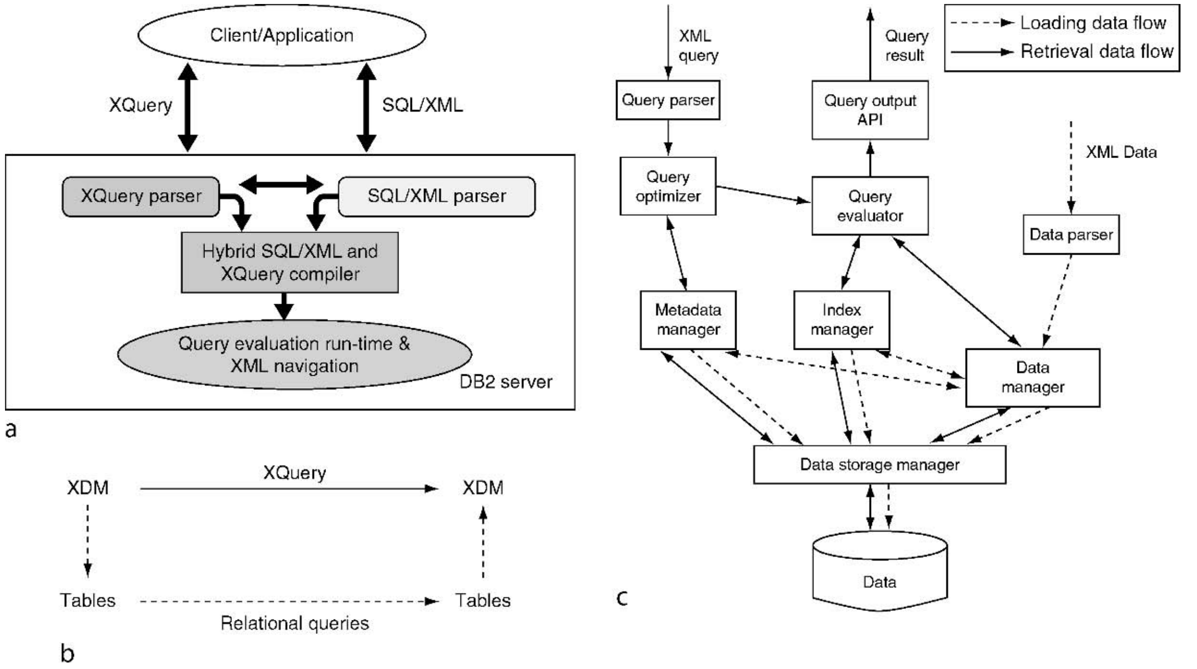
The *Pathfinder* XQuery compiler has been developed under the main hypothesis that the well-understood relational database kernels also make for efficient XQuery processors. Such a relational account of XQuery processing can indeed yield scalable XQuery implementations – provided that the system exploits suitable relational encodings of both, (i) the XQuery Data Model (XDM), *i.e.*, tree fragments as well as ordered item sequences, and (ii) the dynamic semantics of XQuery that allow the database back-end to play its trump: set-oriented evaluation. *Pathfinder* determinedly implements this approach, effectively realizing the dashed path in Fig. 1b. Any relational database system may assume the role of *Pathfinder*'s back-end database; the compiler does not rely on XQuery-specific builtin functionality and requires no changes to the underlying database kernel.

Internal XQuery and Data Model Representation. To represent XML fragments, *i.e.*, ordered unranked trees of XML nodes, *Pathfinder* can operate with any node-level tabular encoding of trees (node $\hat{=}$ row) that – along other XDM specifics like tag name, node kind, *etc.* – preserves node identity and document order. A variety of such encodings are available, among these are variations of *pre/post* region encodings [8] or ORDPATH identifiers [15]. Ordered sequences of items are mapped into tables in which a dedicated column preserves sequence order.

Pathfinder compiles incoming XQuery expression into plans of relational algebra operators. To actually operate the database back-end in a set-oriented fashion, *Pathfinder* draws the necessary amount of independent work from XQuery's `for`-loops. The evaluation of a subexpression $\in$ in the scope of a `for`-loop yields an ordered sequence of zero or more items in each loop iteration. In *Pathfinder*s relational encoding, these items are laid out in a *single table for all loop iterations*, one item per row. A plan consuming this *loop-lifted* or "unrolled" representation of $\in$ may, effectively, process the results of the individual iterated evaluations of $\in$ in any order it sees fit – or in parallel [10]. In some sense, such a plan is the embodiment of the independence of the individual evaluations of an XQuery `for`-loop body.

Exploitation of Type and Schema Information. Pathfinder's node-level tree encoding is schema-oblivious and does not depend on XML Schema information to represent XML documents or fragments. If the DTD of the XML documents consumed by an expression is available, the compiler annotates its plans with node location and fan-out information that assists XPath location path evaluation but is also used to reduce a query's runtime effort invested in node construction and atomization. *Pathfinder* additionally infers static type information for a query to further prune plans, *e.g.*, to control the impact of polymorphic item sequences which present a challenge for the strictly typed table column content in standard relational database systems.

Query Runtime. Internally, *Pathfinder* analyzes the data flow between the algebraic operators of the generated plan to derive a series of operator annotations (keys, multi-valued dependencies, *etc.*) that drive plan simplification and reshaping. A number of



XQuery Processors. Figure 1. (a) *DB2 XML* system architecture. (b) Pathfinder: purely relational XQuery on top of vanilla database back-ends. (c) Timber architecture overview [11].

XQuery-specific optimization problems, *e.g.*, the stable detection of value-based joins or XPath twigs and the exploitation of local order indifference, may be approached with simple or well-known relational query processing techniques.

The *Pathfinder* XQuery compiler is retargetable: its internal table algebra has been designed to match the processing capabilities of modern SQL database systems. Standard B-trees provide excellent index support. Code generators exist that emit sequences of SQL:1999 statements [9] (no SQL/XML functionality is used but the code benefits if OLAP primitives like DENSE_RANK are available). Bundled with its code generator targeting the extensible column store kernel *MonetDB*, *Pathfinder* constitutes XQuery technology that processes XML documents in the Gigabyte-range in interactive time [7].

Timber: A Native XML Database System

Timber [11] is an XML database system which manages XML data natively: i.e., the XML data instances are stored in their actual format and the XQueries are processed directly. Because of this native representation, there is no overhead for converting the data between the XML format and the relational representation or for translating queries from the XQuery format into the SQL format. While many components of the traditional database can be reused (for example, the transaction management facilities), other components need to be modified to accommodate the new data model and query language. The key contribution of the Timber system is a comprehensive set-at-a-time query evaluation engine based on an XML manipulation algebra, which incorporates novel access methods and algebraic rewriting and cost based optimizations. An architecture overview of Timber is shown in Fig. 1c, as presented in [11].

XML Data Model Representation. When XML documents are loaded into the system, Timber automatically assigns each XML node with four labels $\langle D, S, E, L\rangle$, where D indicates which document the node belongs to, and S, E, L represents the *start key*, *end key*, and *level* of the node, respectively. These labels allow quick detection of relationships between nodes. For example, a node $\langle d_1,s_1,e_1,l_1\rangle$ is an ancestor of another node $\langle d_1,s_2,e_2,l_2\rangle$ iff $s_1 < s_2 \wedge e_1 > e_2$. Each node, along with the labels, are stored natively, and in the order of their start keys (which correspond to their document order), into the Timber storage backend.

XQuery Representation. A central concept in the Timber system is the TLC (Tree Logical Class) algebra [12,18,17], which manipulates sets (or sequences) of heterogeneous, ordered, labeled trees. Each operator in the TLC algebra takes as input one or more sets (or sequences) of trees and produces as output a set (sequence) of trees. The main operators in TLC include *filter*, *select*, *project*, *join*, *reordering*, *duplicate-elimination*, *grouping*, *construct*, *flatten*, *shadow/illuminate*. There are several important features of the TLC algebra. First, like the relational algebra, TLC is a "proper" algebra with compositionality and closure. Second, TLC efficiently manages the heterogeneity arising in XML query processing. In particular, heterogenous input trees are reduced to homogenous sets for bulk manipulation through tree pattern matching. This tree pattern matching mechanism is also useful in selecting portions of interest in a large XML tree. Third, TLC can efficiently handle both set and sequence semantics, as well as a hybrid semantics, where part of the input collection of trees is ordered [17]. Finally, the TLC algebra covers a large fragment of XQuery, including nested FLWOR expressions. Each incoming XQuery is parsed and compiled into the TLC algebra representation (i.e., logical query plans) before being evaluated against the data.

The Timber system, and its underlying algebra, has been extended to deal with text manipulation [2] and probabilistic data [14]. A central challenge with querying text is that exact match retrieval is too crude to be satisfactory. In the field of information retrieval, it is standard practice to use scoring functions and provide ranked retrieval. The TIX algebra [2] shows how to compute and propagate scores during XML query evaluation in Timber. Traditionally, databases have only stored facts, which by definition are certain. Recently, there has been considerable interest in the management of uncertain information in a database. The bulk of this work has been in the relational context, where it is easy to speak of the probability of a tuple being in a relation. ProTDB develops a model for storing and manipulating probabilistic data efficiently in XML [14]. The probability of occurrence of an element in a tree (at that position) is recorded conditioned on the probability of its parent's occurrence.

Query Runtime: Evaluation and Optimization. The heart of the Timber system is the query evaluation engine, which compiles logical query plans into the physical algebra representation (i.e., physical query plans) and evaluates those query plans against the stored XML data to produce XML results. It includes two main subcomponents: the *query optimizer* and the *query evaluator.*

The query evaluator executes the physical query plan. The separation between the logical algebra and the physical algebra is greater in XML databases than in relational databases, because the logical algebra here manipulates trees while the physical algebra manipulates "nodes" – data are accessed and indexed at the granularity of nodes. This requires the design of several new physical operators. For example, for each XML element, the query may need to access the element node itself, its child sub-elements, or even its entire descendant subtree. This requires the *node materialization* physical operator, which takes a (set of) node identifier(s) and returns a (set of) XML tree(s) that correspond to the identifier(s). When and how to materialize the nodes affects the overall efficiency of the physical query plan and it is the job of the optimizer to make the right decision. *Structural join* is another physical operator that is essential for the efficient retrieval of data nodes that satisfy certain structural constraints. Given a parent-child or ancestor-descendant relationship condition, the structural join operator retrieve all pairs of nodes that satisfy the condition. Multiple structural join evaluations are typically required to process a single tree-pattern match. In Timber, a whole stack-based family of algorithms has been developed to efficiently process structural joins, and they are at the core of query evaluation in Timber [1].

The query optimizer attempts to find the most efficient physical query plan that corresponds to the logical query plan. Every pattern match in Timber is computed as a sequence of structural joins and the order in which these are computed makes a substantial difference to the evaluation cost. As a result, *join order selection* is the predominant task of the optimizer. Heuristics developed for relational systems often do not work well for XML query optimization [20]. Timber employs a dynamic programming algorithm to enumerate a subset of all the possible join plans and picks the plan with the lowest cost. The cost is calculated by the *result size estimator,* which relies on the *position histogram* [19] to estimate the lower and upper bounds of each structural join.

DB2 XML: A Hybrid Relational and XML DBMS

DB2 XML (DB2 is a trademark of IBM Corporation.) is a hybrid relational and XML database management system, which unifies new native XML storage, indexing and query processing technologies with existing relational storage, indexing and query processing. A *DB2 XML* application can access XML data using either SQL/XML or XQuery [6,16]. The general system architecture is shown in Fig. 1a. It builds on the premises that (i) relational and XML data will co-exist and complement each other in enterprise information management solutions, and (ii) XML data are different enough that it requires its own storage and processor.

Internal XQuery and Data Model Representation. At the heart of *DB2 XML*'s native XML support is the XML data type, introduced by SQL/XML. *DB2 XML* introduces a new native XML storage format to store XML data as instances of the XQuery Data Model in a structured, type-annotated tree. By storing the binary representation of type-annotated XML trees, *DB2 XML* avoids repeated parsing and validation of documents.

In *DB2 XML*, XQuery is not translated into SQL, but rather mapped directly onto an internal query graph model (QGM) [6], which is a semantic network used to represent the data flow in a query. Several QGM entities are re-used to represent various set operations, such as iteration, join and sorting, while new entities are introduced to represent path expressions and to deal with sequences. The most important new operator is the one that captures XPath expressions. *DB2 XML* does not normalize XPath expressions into FLWOR blocks, where iteration between steps and within predicates is expressed explicitly. Instead, XPath expressions that consist of solely navigational steps are expressed as a single operator. This allows *DB2 XML* to apply rewrite and cost-based optimization [4] to complex XQueries, as the focus is not on ordering steps of an XPath expression.

Exploitation of Type and Schema Information. DB2 XML provides an XML Schema repository (XSR) to register and maintain XML schemas and uses those schemas to validate XML documents. An important feature of *DB2 XML* is that it does not require an XML schema to be associated with an XML column. An XML column can store documents validated according to many different and evolving schemas, as well as schema-less documents. Hence, the association between schemas and XML documents is on per document basis, providing maximum flexibility.

As *DB2 XML* has been targeted to address schema evolution [5], it does not support schema import or static typing features of XQuery. These two features are too restrictive because they do not allow conflicting schemas and each document insertion or schema update may result in recompilation of applications. Hence, *DB2 XML* does not exploit XML schema information in query compilation. However, it uses simple data type information for optimization, such as selection of indexes.

Query Runtime. DB2 XML query evaluation runtime contains three major components for XML query processing:

1. *XQuery Function Library: DB2 XML* supports several XQuery functions and operators on XML schema data types using native implementations.
2. *XML Index Runtime: DB2 XML* supports indexes defined by particular XML path expressions, which can contain wildcards, and descendant axis navigation, as well as kind tests. Under the covers, an XML index is implemented with two B+Trees: a *path index*, which maps distinct reverse paths to generated path identifiers, and a value index that contains path identifiers, values, and node identifiers for each node that satisfy the defining XPath expression. As indexes are defined via complex XPath expressions, *DB2 XML* employs the XPath containment algorithm of [3] to identify the indexes that are applicable to a query.
3. *XML Navigation:* XNAV operator evaluates multiple XPath expressions and predicate constraints over the native XML store by traversing parent-child relationship between the nodes [13]. It returns node references (logical node identifiers) and atomic values to be further manipulated by other runtime operators.

Key Applications

Scalable systems for XML data storage and XML query processing are essential to effectively manage increasing amount of XML data on the web.

URL to Code

Pathfinder

The open-source retargetable Relational XQuery compiler *Pathfinder* is available and documented at www.pathfinder-xquery.org. *MonetDB/XQuery – Pathfinder* bundled with the relational database back-end *MonetDB* – is available at www.monetdb-xquery.org.

Timber

Timber is available and documented at www.eecs.umich.edu/db/timber.

Cross-references

- ▶ Top-k XML Query Processing
- ▶ XML Benchmarks
- ▶ XML Indexing
- ▶ XML Storage
- ▶ XPath/XQuery

Recommended Reading

1. Al-Khalifa S., Jagadish H.V., Patel J.M., Wu Y., Koudas N., and Srivastava D. Structural joins: a primitive for efficient XML query pattern matching. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 141–152
2. Al-Khalifa S., Yu C., and Jagadish H.V. Querying structured text in an XML database. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 4–15
3. Balmin A., Özcan F., Beyer K.S., Cochrane R.J., and Pirahesh H. A Framework for using materialized XPath views in XML query processing. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, p. 6071.
4. Balmin A. et al. Integration cost-based optimization in DB2 XML. IBM Syst. J., 45(2):299–230, 2006.
5. Beyer K.S. and Özcan F. et al. System RX: one part relational, one part XML. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 347–358.
6. Beyer K.S., Siaprasad S., and van der Linden B. DB2/XML: designing for evolution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 948–952.
7. Boncz P.A., Grust T., van Keulen M., Manegold S., Rittinger J., and Teubner J. MonetDB/XQuery: a fast XQuery processor powered by a relational engine. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 479–490.
8. Grust T. Accelerating XPath location steps. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 109–220.
9. Grust T., Mayr M., Rittinger J., Sakr S., and Teubner J. A SQL: 1999 code generator for the pathfinder XQuery compiler. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1162–1164.
10. Grust T., Sakr S., and Teubner J. XQuery on SQL hosts. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 252–263.
11. Jagadish H.V., Al-Khalifa S., Chapman A., Lakshmanan L.V.S., Nierman A., Paparizos S., Patel J., Srivastava D., Wiwatwattana N., Wu Y., and Yu C. TIMBER: a native XML database. VLDB J., 11:274–291, 2002.
12. Jagadish H.V., Lakshmanan L.V.S., Srivastava D., and Thompson K. TAX: a tree algebra for XML. In Proc. 8th Int. Workshop on Database Programming Languages, 2001, pp. 149–164.
13. Josifovski V., Fontoura M., and Barta A. Querying XML streams. VLDB J., 14(2):197–210, 2005.

14. Nierman A. and Jagadish H.V. ProTDB: probabilistic data in XML. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 646–657.
15. O'Neil P., O'Neil E., Pal S., Cseri I., Schaller G., and Westburg N. ORDPATHs: insert-friendly XML node labels. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 903–908.
16. Özcan F., Chamberlin D., Kulkarni K.G., and Michels J.-E. Integration of SQL and XQuery in IBM DB2. IBM Syst. J., 45 (2):245–270, 2006.
17. Paparizos S. and Jagadish H.V. Pattern tree algebras: sets or sequences? In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 349–360.
18. Paparizos S., Wu Y., Lakshmanan L.V.S., and Jagadish H.V. Tree logical classes for efficient evaluation of XQuery. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 71–82.
19. Wu Y., Patel J.M., and Jagadish H.V. Estimating answer sizes for XML queries. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 590–608.
20. Wu Y., Patel J.M., and Jagadish H.V. Structural join order selection for XML query optimization. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 443–454.

XSL Formatting Objects

► XSL/XSLT

XSL/XSLT

BERND AMANN
Pierre & Marie Curie University (UPMC), Paris, France

Synonyms

eXtensible Stylesheet Language; eXtensible Stylesheet Language transformations; XSL-FO; XSL formatting objects

Definition

XSL (eXtensible Stylesheet Language) is a family of W3C recommendations for specifying XML document transformations and typesettings. XSL is composed of three separate parts:

- XSLT (eXtensible Stylesheet Language Transformations): a template-rule based language for the structural transformation of XML documents.
- XPath (XML Path Language): a structured query language for the pattern, type and value-based selection of XML document nodes.
- XSL-FO (XML Formatting Objects): an XML vocabulary for the paper document oriented typesetting of XML documents.

Historical Background

The development of XSL was mainly motivated by the need for an open typesetting standard for displaying and printing XML documents. Its conception was strongly influenced by the DSSSL (Document Style Semantics and Specification Language) ISO standard (ISO/IEC 10179:1996) for SGML documents. Like DSSL, XSL separates the document typesetting task into a *transformation* task and a *formatting* task. Both languages are also based on *structural recursion* for defining transformation rules, but whereas DSSL applies a functional programming paradigm, XSL uses XML-template rules and XPath pattern matching for defining document transformations.

The W3C working group on XSL was created in December 1997 and a first working draft was released in August 1998. XSLT 1.0 and XPath 1.0 became W3C recommendations in November 1999, and XSL-FO reached recommendation status in October 2001. During the succeeding development of XQuery, both the XQuery and XSLT Working Groups shared responsibility for the revision of XPath, which became the core language of XQuery. XSLT 2.0, XPath 2.0 and XQuery 1.0 achieved W3C recommendation status in January 2007.

Foundations

XSLT Programming

XSLT programming consists in defining collections of *transformation rules* that can be applied to different classes of document nodes. Each rule is composed of a *matching pattern* and a possibly empty *XML template*. The matching pattern is used for dynamically binding rules to nodes according to their local (name, attributes, attribute values) and structural (document position) properties. Rule templates are XML expressions composed of static XML output fragments and dynamic XSLT instructions generating new XML fragments from the input data.

The following example illustrates the usage of XSLT template rules for implementing some simple relational queries on the XML representation of a relational database `db`. The database contains two relations `R(a: int,b: int)` and `S(b: int, c: int)`. Each relation is represented by a single element containing a sub-element of element type `t` for each tuple:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<db>
<R>
    <t a="1" b="5" />
    <t a="2" b="5" />
    <t a="3" b="6" />
    <t a="4" b="7" />
</R>
<S>
    <t b="5" c="1" />
    <t b="6" c="3" />
    <t b="6" c="4" />
</S>
</db>
```

The first transformation is defined by the following two rules:

Rule 1:

```
<xsl:template match="/">
    <T>
        <xsl:apply-templates select="db/*/t" />
    </T>
</xsl:template>
```

Rule 2:

```
<xsl:template match="t">
    <xsl:copy-of select="." />
</xsl:template>
```

Both rules specify the class of nodes to which they can be applied by an absolute (starting from the document root ´/´) or a relative XPath expression. The first rule applies to the document root ´/´, whereas the second rule matches all document elements of type `t` (independently of their absolute position in the document).

The XSLT processing model is based on the recursive application of template rules to a *current node list*, which is initialized by the list containing only the document root. The transformation consists of binding each node to exactly one transformation rule (The existence of at least one matching rule is guaranteed by a default rule for each kind of document node.) and replacing it by the corresponding rule template. Templates contain static XML data and dynamic XSLT instructions which recursively generate new transformation requests (XSLT instructions, are distinguished from static fragments by using the XSLT namespace http://www.w3.org/1999/XSL/Transform which is generally (but not necessarily) bound to the prefix <<xsl:>>). The whole transformation process stops when all transformation requests have been executed.

The rule template of rule 1 generates an element of type `T`, which contains a new transformation request type `xsl:apply-template` for all tuples in the database returned by the XPath expression `db/*/t`. This expression is evaluated against the *context node* of the rule (the document root) and returns a new *current node list* containing all elements of type `t` in `R` and `S` (wildcard * matches all element names). Rule 2 matches all elements (tuples) of type `t` and creates a copy by instruction `xsl:copy-of select="."`. The final result is a relation T containing the *union* of R and S.

The previous example illustrates the usage of XPath for the dynamic selection of template rules and for recursively generating new current node lists for transformation. Rule patterns can simply match nodes by their element types, but they can also use more complex XPath patterns defining value-based and structural matching constraints. For example, in order to copy only tuples of `R`, one might replace rule 2 by the following two rules:

Rule 3:

```
<xsl:template match="R/t">
    <xsl:copy-of select="." />
</xsl:template>
```

Rule 4:

```
<xsl:template match="S/t" />
```

Rule 3 matches and copies sub-elements `t` of some element `R`, whereas rule 4 matches tuples of `S` without creating any node in the output tree. Observe that the same result can be achieved by keeping rule 2 and replacing rule 1 by the following rule:

Rule 5:

```
<xsl:template match="/">
    <T>
        <xsl:apply-templates select="db/R/t" />
    </T>
</xsl:template>
```

The following three rules simulate relational selection $\sigma_{b=5}(R)$ by copying only of tuples in R with attribute value b=5:

Rule 6:

```
<xsl:template match="R/t[@b=5]">
```

```
        <xsl:copy-of select="." />
    </xsl:template>
Rule 7:
    <xsl:template match="R/t" />
Rule 8:
    <xsl:template match="S/t" />
```

Rule 6 matches only tuples of `R` with attribute value `b` equal to 5, and creates a copy of this subset in the output tree. Rule 7 applies to *all* tuples of `R` without any other restriction. The resulting binding conflict is solved by a set of laws based on import precedence, priority attribute values and certain syntactical criteria, which generally choose the rule with the most specific matching criteria. For example, rule 7 applies to all nodes that can be transformed by rule 6, whereas the opposite is not true. The inclusion problem for XPath patterns has been formally shown to be PSPACE-complete [9], and the recommendation document defines an incomplete set of syntactical criteria that can easily be implemented and allowed to solve a large set of conflicts in practice. If the conflict resolution algorithm for template rules leaves more than one matching rule, the XSLT processor must generate a recoverable dynamic error where the optional recovery action is to select, from the matching template rules that are left, the one that occurs last in declaration order.

XSLT rules can be assigned to different computation *modes* by using an optional *mode* attribute. For example, the following three rules compute the natural join of R and S on attribute `b` by a simple nested loop. Rule 9 applies to all tuples of R in *default* mode and represents the outer loop of the join: XPath expression `/db/S/t[@b=current()/@b]` selects for each *current* tuple in R, all tuples in S with the same `b` attribute value. The inner loop is represented by rule 10 which is triggered by rule 9 and joins parameter `$tuple` with some tuple in S. The template of this rule creates a new tuple by copying all attributes element `$tuple` and all attributes of the current node (tuple). Attribute `mode` is used for distinguishing between the outer loop (mode `default`) and the inner loop (mode `join`). In order to "neutralize" the `default` transformation of S, rule 11 applies to S in default mode and generates nothing.

```
Rule 9:
    <xsl:template match="R/t">
        <xsl:apply-templates select="/db/S/t[@b=current()/@b]" mode="join">
            <xsl:with-param name="tuple" select="." />
        </xsl:apply-templates>
    </xsl:template>
Rule 10:
    <xsl:template match="S/t" mode="join">
    <xsl:param name="tuple"> <t /> </xsl:param>
    <t>
      <xsl:copy-of select="$tuple/@*" />
      <xsl:copy-of select="@*" />
    </t>
    </xsl:template>
Rule 11:
    <xsl:template match="S/t" />
```

XSL-FO document typesetting

XSL-FO is a vocabulary of XML element types for defining the main typographic objects (chapters, pages, paragraphs, figures, etc.) and their properties (character set, indentation, justification, etc.) used for the paper-oriented typesetting of documents. The main structure of an XSL-FO document is defined by a document model (fo:layout-master), one or several page sequences (fo:page-sequence) composed of one or several flows (fo-flow) of blocks (fo-block). Each of these elements can be parametrized by a set of attribute values for configuring the layout of pages (page-height, margins), tables (column width and height, padding, etc.), lists (item label and distance) and blocks (font-family and size, text alignment etc.). The richness and complexity of the XSL-FO typesetting model is illustrated by the size of the recommendation document (400 pages), and the reader is invited to consult the W3C's XSL-FO tutorial for a detailed introduction.

Theoretical Foundations of XSLT

The most important theoretical aspects of XSL concern XSLT and its interaction with XPath. XSLT and XPath have been theoretically evaluated and compared to other XML query languages by using different theoretical frameworks. For example, the formal semantics of XSLT has been defined by a rewriting process starting from an empty ordered output document, a set of rewriting rules (the XSLT program) and an ordered labeled input tree with "pebbles" (the current node list) [6]. Based on this representation it was possible to show that, under certain constraints (no join on data values), given two XML schemas (regular tree languages) τ_1 and τ_2 and an XSLT stylesheet (k-pebble transducer) T, it is possible to type-check T with respect to its input type τ_2: $\forall\ t \in \tau_1{:}T(t) \subseteq \tau_2$ is decidable. Similarly, [2] proposed a formal model for a subset of XSLT using tree-walking tree-transducers with registers and look-ahead, which can compute all unary monadic second-order (MSO) structural patterns. This expressiveness result provides an important theoretical foundation for XSLT, since MSO captures different robust formalisms like first order logics with set quantification, regular tree languages, query automata and finite-valued attribute grammars.

The computation model of XSLT can also be compared with that of s*tructural recursion* for semi-structured data [1]. Structural recursion is proposed by functional programming languages like CAML for the dynamic selection of function definitions by matching function parameter values with pattern signatures. For example, the following transformation function `f` removes all elements of type `E` and renames all elements of type `A` to `B`:

```
f(v)=v
f({})={}
f({′A′ : t})={ ′B′ : f(t)}
f({′E′ : t})=f(t)
```

This function can be defined in XSLT as follows:

```
<xsl:template match=′*′>
    <xsl:copy><xsl:apply-templates select=′*′>
    </xsl:copy>
</xsl:template>
<xsl:template match=′E′ />
<xsl:template match=′A′>
    <B><xsl:apply-templates select=′.′></B>
</xsl:template>
```

There are two main differences between XSLT and structural recursion. First, XSLT is defined on trees, whereas structural recursion can be applied on arbitrary graphs. Second, all structural recursion programs are guaranteed to terminate, whereas it is easily possible to write *infinite* XSL transformations as it is shown in the following rule, which recursively generates for any element a new element containing the result of its own transformation:

```
<xsl:template match=′*′>
    <a><xsl:apply-templates select=′.′></a>
</xsl:template>
```

The result of this transformation rule is an infinite tree <a><a><a>. . .</a></a></a>.

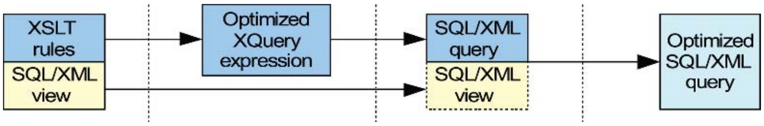
XSLT and XQuery

XSLT 2.0 is the result of the collaboration between the W3C working groups on XSL and XQuery. Both languages, XQuery 1.0 and XSLT 2.0 obtained recommendation status in January 2007 and share the same data model, type system and function library. The most notable change concerning the XSL recommendation is the evolution of XPath 1.0 to XPath 2.0 and the corresponding changes in the underlying data model. XPath 2.0 is the core language of XQuery 1.0 and integrates the type system of XML Schema with a rich library of built-in types, functions and operators. Every XSLT value is now a sequence and XPath 1.0 node-sets are replaced by node sequences. Variables can be bound to arbitrary sequences of values and trees, which simplifies certain programming issues observed in XSLT 1.0. Other new important features concern grouping of nodes, text matching functions, user defined functions and the possibility to generate multiple result trees by one transformation.

XSLT 2.0 and XQuery 1.0 are two declarative languages for querying and transforming XML document trees with similar expressive power. Both languages are Turing complete and [3] shows how to translate XSLT 2.0 programs into XQuery 1.0. However, the objectives of both languages are specific and their implementations are optimized for a particular usage. XSLT has been conceived for transforming individual documents into some fixed output format whereas XQuery is a query language that allows the retrieved relevant information retrieval of from large document collections. The dynamic binding of rule templates makes XSLT a flexible and modular language that facilitates code reuse and the development of large applications. In particular, it is possible to *import* XSLT template rule collections defined for external document fragments that are integrated into a unique XML document. On the other hand, modern XQuery engines are able to efficiently query large collections of documents using advanced query optimization techniques based on static type analysis.

Optimizing XSLT

XML is used in many applications as a publishing and exchange format for structured data. In this kind of application, XSLT serves for publishing XML query results according to the different usages and clients. Each data exchange might need one or several XSLT transformations on the data producer and the data



XSL/XSLT. Figure 1. XSLT optimization in XML-enabled RDBMS.

consumer side, and optimizing the transformation process becomes an important issue.

[7] proposes an XSLT optimization approach based on algorithms for optimizing the template rule selection process and for rewriting algebraic translations of XSLT stylesheets. XSLT optimization is revealed to be a hard problem because of two undecidability results concerning rule conflict detection at compile time, and finding optimal evaluation plans under any reasonable cost function. [5] study the problem of XSLT processing and optimization in an XML-enabled relational database system (RDBMS). The whole process is illustrated in Fig. 1.

The optimization process starts from a collection of XSLT transformation rules, which are applied to a XML data generated by a SQL/XML query on top of a relational database. The basic idea is to translate the XSLT rules into a SQL/XML query on a SQL/XML view, which can then be optimized by a standard relational query optimizer. The translation step generates an intermediate XQuery expression, which is optimized by exploiting structural schema information obtained from the underlying RDBMS.

Key Applications

The possibility to define transformations from any XML document structure A to any other structure B makes XSLT a powerful and generic tool in many XML applications. The main key applications are information publishing, data exchange and model transformation.

Information Publishing

XSL was initially designed for the publishing of XML documents, where document transformation consists of transforming any document into some specific format for screen display and printing. The final document format can be conformed to XML like XHTML, SMIL, DocBook or WML, but it also can be non XML-compliant like HTML (which is not XML), PDF or RTF. In the first case, XSLT is sufficient for producing the output. In the second case, the transformation process is followed by a *formatting* step that is implemented by a specific software mapping XML data to a non-XML format. This formatting step can be a simple mapping from XHTML to HTML by changing some small syntactic differences, and a more complex generation of high-quality electronic documents in a binary format like RTF and PDF. For the second case, XSL proposes XSL-FO, a standard XML vocabulary for typesetting XML documents.

Data Exchange and Service Integration

Service-oriented architectures (SOA) based on the W3C SOAP/WSDL web service recommendations represent a modern solution for XML-based data and application integration. These kind of infrastructures strongly depend on efficient XML data transformation tools for wrapping data exchanged between heterogeneous web services. By its declarative and modular nature, XSLT is a powerful language for defining data and service wrappers, and is proposed as such by standard business process modeling languages (BPEL) and infrastructures (JBI).

Model Exchange and Transformation

On a more abstract level, XSLT has been proposed for the transformation of XMI (XML Metadata Interchange) documents. XMI is a standard interchange mechanism used model driven in component based development and deployment infrastructures (MDA). The most common use of XMI is as an interchange format for UML models, although it can also be used for serialization of models of other languages (metamodels). In this kind of environment, XSLT can be used for transforming definitions a some model A, for example UML, into a different model B, for example XML Schema.

Url to Code

XSLT code examples can be found at the following URL:

http://www-poleia.lip6.fr/~amann/XSLT/index.xml

Cross-references

- ▶ Tree Grammars and Languages
- ▶ Web Services
- ▶ XML
- ▶ XPath/XQuery

Recommended Reading

1. Abiteboul S., Buneman P., and Suciu D. Data on the Web: from relations to semistructured data and XML. Morgan Kaufmann, Los Altos, CA, 1999.
2. Bex G.J., Maneth S., and Neven F. A formal model for an expressive fragment of XSLT. In Proc. 1st Int. Conf. Computational Logic, 2000, pp. 1137–1151.
3. Fokoue A., Rose K.H., Siméon J., and Villard L. Compiling XSLT 2.0 into XQuery 1.0. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 682–691.
4. Kay M. XSLT Programmer's Reference, 2nd edition, WROX Press Ltd., 2002.
5. Liu Z.H. and Novoselsky A. Efficient XSLT processing in relational database system. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 1106–1116.
6. Milo T., Suciu D., and Vianu V. Typechecking for XML Transformers. In Proc. 19th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2000, pp. 11–22.
7. Moerkotte G. Incorporating XSL processing into database engines. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 107–118.
8. Muench S. Building Oracle XML Applications, O'Reilly, 2000.
9. Neven F. and Schwentick T. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. Logic. Methods Comput. Sci., 2(3), 2006.
10. W3C XSL-FO Tutorial, http://www.w3schools.com/xslfo/default.asp.
11. W3C. XSL Transformations (XSLT) Version 1.0, W3C Recommendation, J. Clark (ed.). http://www.w3.org/TR/xslt, 1999.
12. W3C. XML Path Language (XPath) Version 1.0, W3C Recommendation, J. Clark and S. DeRose (eds.). http://www.w3.org/TR/xpath, 1999.
13. W3C. Extensible Stylesheet Language (XSL) Version 1.0, W3C Recommendation, S. Adler, A. Berglund, J. Caruso, S. Deach, T. Graham, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, S. Zilles, (eds.). hhttp://www.w3.org/TR/2001/REC-xsl-20011015/, 2001.
14. W3C. XML Path Language (XPath) 2.0, W3C Recommendation, A. Berglund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, J. Siméon (eds.). http://www.w3.org/TR/xpath20, 2007.
15. W3C. XSL Transformations (XSLT) Version 2.0, W3C Recommendation, M. Kay, (ed.). http://www.w3.org/TR/xslt20, 2007.

XSL-FO

▶ XSL/XSLT

Z

Zero-One Laws

Nicole Schweikardt
Johann Wolfgang Goethe-University Frankfurt am Main, Frankfurt, Germany

Definition

A query language is said to have the *0-1 law* if every Boolean query that contains no constants (i.e., the query does not mention any particular element from the domain of potential data values) is almost surely true or almost surely false. The notions of being "almost surely true," respectively, "almost surely false" are defined as follows: Let σ be a fixed database schema. For each natural number n, let $DB_n(\sigma)$ be the set of all database instances of schema σ whose active domain is a subset of $\{1,\ldots,n\}$ (i.e., all database entries belong to $\{1,\ldots,n\}$). For a Boolean query q of schema σ let $\mu_n(q)$ be the probability that a database D chosen uniformly at random from $DB_n(\sigma)$ is a "yes"-instance of query q. In other words, $\mu_n(q)$ is the number of databases in $DB_n(\sigma)$ on which q evaluates to "yes," divided by the number of all databases in $DB_n(\sigma)$. Query q is said to be *almost surely true* (respectively, *almost surely false*), if the limit $\mu(q) := \lim_{n\to\infty}\mu_n(q)$ exists and is equal to 1 (respectively, 0).

Key Points

0-1 laws can be used as a tool for proving expressivity bounds for query languages: If q is a Boolean query for which the limit $\mu(q) = \lim_{n\to\infty}\mu_n(q)$ either does not exist or is different from 0 and 1, then q cannot be expressed by any query language that has the 0-1 law.

For example, let σ be the schema consisting of one binary relation symbol E, and let q_{even} be the query "Does the given database contain an even number of tuples?" It is not difficult to see that the limit $\mu(q_{even}) = \lim_{n\to\infty}\mu_n(q_{even})$ exists and is equal to 1/2. Thus, query q_{even} is neither "almost surely true" nor "almost surely false" and hence cannot be expressed by a query language that has the 0-1 law.

Historically, the first query language for which a 0-1 law was proven was the relational calculus (i.e., first-order logic). The proof also shows that there exists an algorithm which, given a Boolean relational calculus query q, computes $\mu(q)$. Since then, 0-1 laws have been shown for many different query languages, among them fixed point logic, infinitary logic $L^{\omega}_{\infty\omega}$, and various fragments of second-order logic. Variants of the 0-1 law are also known for several other probability measures and for restrictions to particular classes of databases.

A common method for proving that a query language has the 0-1 law is based on so-called *extension axioms* and an *Ehrenfeucht-Fraïssé game* argument. 0-1 laws are also closely related to the theory of the *countable random graph*. For an overview of results and proof techniques, refer to the textbooks [1,2,4] and the survey [3].

To point out the limitations of the use of 0-1 laws for proving inexpressibility results, it should be noted that there do exist queries that are almost surely true but nevertheless are not expressible in the relational calculus (an example is the query q_{conn}: "Does the given database relation E form a connected graph?").

Furthermore, several query languages are known *not* to have the 0-1 law, e.g., existential second-order logic and monadic second-order logic (cf. [1,3,4]).

Cross-references

▶ Expressive Power of Query Languages

Recommended Reading

1. Ebbinghaus H.-D. and Flum J. Finite Model Theory, 2nd edn. Springer, Berlin, 1999.
2. Hodges W. Model Theory. Cambridge University Press, Cambridge, New York, USA, 1993.
3. Kolaitis P. and Vardi M.Y. 0-1 laws for fragments of existential second-order logic: a survey. In Proc. 25th Int. Symp. on Mathematical Foundations of Computer Science, 2000, pp. 84–98.
4. Libkin L. Elements of Finite Model Theory. Springer, New York, NY, USA, 2004.

ZF-Expression

▶ Comprehensions

Zoning

▶ Storage Security

Zoomable User Interface (ZUI)

▶ Zooming Techniques

Zooming Techniques

Harald Reiterer[1], Thorsten Büring[2]
[1]University of Konstanz, Konstanz, Germany
[2]Ludwig-Maximilians-University Munich, Munich, Germany

Synonyms

Zoomable user interface (ZUI); Multiscale interface; Scaling

Definition

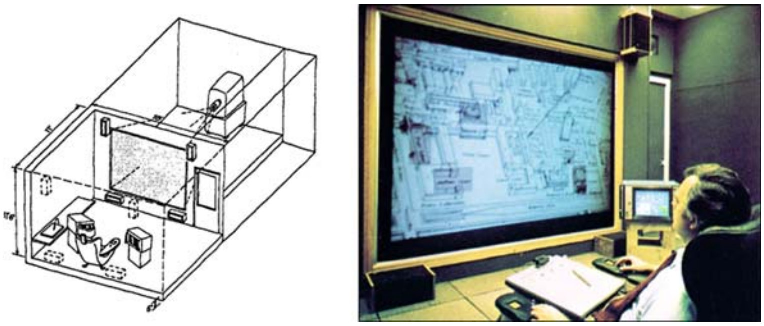
Zooming facilitates data presentation on limited screen real-estate by allowing the users to alter the scale of the viewport such that it shows a decreasing fraction of the information space with an increasing magnification. Hence the system may first present a global overview of the information space for the benefit of orientation, and in a second step, the users can then dynamically re-allocate the screen space based on the information objects they are interested in. A navigation technique commonly used in conjunction with zooming is *panning*: a movement of the viewport over the information space at a constant scale.

Historical Background

The first application to use zooming as a fundamental interface approach was the Spatial Data Management System (SDMS) [5] in 1978 (see Fig. 1). The SDMS system relied heavily on custom hardware, including an octophonic sound system and an instrumented chair equipped with pressure-sensitive joysticks, two touch-sensitive tablets and a digital lapboard. The data was presented via a rear-projected color television display. The flat information space was called Dataland. It presented the content of a database with the help of tiny pictures of faces, maps, television sets, letters, book covers, a calendar, a telephone, and a calculator. Items of similar sort have been grouped together on distinctive color backgrounds. SDMS enabled users to manage and zoom into the visual database representation with the help of a joystick, by touch commands, or by voice. This early attempt of a zoomable interface included a variety of elementary design concepts: a "what you see is what you get" representation of different multimedia data types with the help of icons, a spatial arrangement depending on the users' choice or automatic by semantic (e.g., clustering of related data), the possibility to fly over Dataland in a helicopter-like fashion (pan), and the possibility to change the granularity of the information presentation by zooming in and out.

The Computer Corporation of America (CCA) of Cambridge, Massachusetts built a "field version" of the SDMS. It consisted of three color TV monitors lined up on a table-top with a joystick, a keyboard, and a graphic tablet [9]. The system also offered a scalable data surface with the enhanced possibility to zoom in a semantic way. Therefore each data item on the surface was stored at several levels of detail. Zooming in on such a data item caused the more detailed version to appear (e.g., shape of a ship first, and then shape with further text, and finally a full picture of the ship with all details). Another important new concept was the idea of ports. There was not only one single data surface but an entire set of them. The transition points between them were called ports and shown as special pictures (icons). When zooming in on them, these ports acted as trapdoors down into other information spaces. The result was a hierarchical set of information spaces through which the user could navigate. Ports allowed presenting alternate views of the same data items, and could also be used to activate programs external to SDMS (e.g., electronic mail, text editors).

In 1993, a zoomable user interface (ZUI) called Pad [15] was developed. It introduced mostly the same fundamental design concepts for zooming as SDMS,



Zooming Techniques. Figure 1. The Spatial Data Management System [5].

but the main difference was its ability to run on conventional PCs or Minicomputers. Pad aimed to provide an alternative to the Windows paradigm. The system visualized an infinite two dimensional information plane populated by objects that users could interact with. Such Pad objects could, for instance, be text files, a clock program, or a personal calendar. Each of these entities occupied a well-defined region on the Pad surface and was visualized by means of graphics and portals. Portals showed portions of the Pad surface at different scales, and may also look recursively at other portals. One way to use a portal, for instance, would be to show a miniature overview of a large Pad object. Users can manipulate the view's scale and data representation by performing semantic zoom operations. Another important concept is portal filters that transform data into other complex views, e.g., present tabular data as a bar chart.

Only one year later, the successor to Pad, Pad++ [3], was presented, a system that also constituted the first ZUI toolkit. Pad++ aimed to serve as the basis for exploration of novel interfaces for information visualization by providing a framework for simplifying the creation of multiscale applications. It introduced some, mostly technical, enhancements over the original Pad implementation. For instance, much effort had been devoted to realizing smooth semantic zooming, even with hundreds of thousands of objects loaded into the information space. To achieve this, the rendering with Pad++ followed a "parallel lazy loading" strategy, i.e., only the portion of the database that is currently visible is loaded. One important design objective of Pad++ was to support a wide range of platforms ranging from high-end workstations to PDAs and Set-top boxes. However, an increased level of platform independency was only achieved by later ZUI toolkits such as Jazz (2000) [4] and Piccolo (2004) [2].

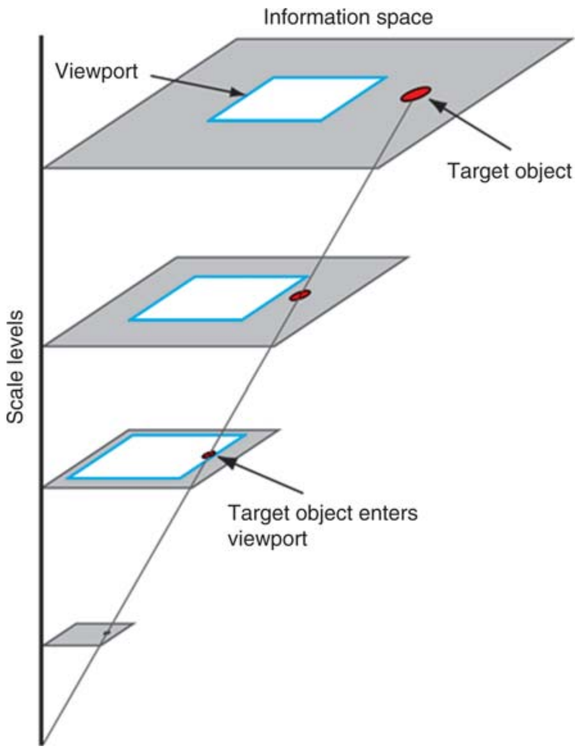
Foundations

A method of illustrating zooming is that of space-scale diagrams [7]. Figure 2 models a ZUI in which the 2D information space is shown at different magnification levels and aligned by the vertical axis representing scale. The inner rectangular outline represents the viewport, i.e., the portion of the information space that is visible. The outer gray area is the off-screen space. In the diagrammatic example, users are searching for the red target object that is off-screen (top-most scale level). Instead of panning the view around, as in the scrolling interface, users zoom out until the target object enters the viewport. In a second step, they can then access the object's details by zooming back in.

Navigation in Information Spaces

In contrast to scrolling interfaces, which are only effective for small spaces, ZUIs develop their full potential as the size of the information space grows. Even if users know the precise location of an off-screen target, in most cases a pan operation would still be a slow way of navigating. Panning only covers distance at a constant pace, while zooming allows users to view off-screen content in a non-linear fashion. This advantage is due to the special properties of multiscale interfaces, in which the shortest path between two points is usually not a straight line, but a combination of zoom and pan operations.

A common problem with ZUIs is the lack of context. Even after a short period of navigation, users lose their



Zooming Techniques. Figure 2. Space-scale diagram [7].

overview due to the continuous clipping of orientation cues during zooming. The most straightforward way to rediscover context in ZUIs is to zoom out. While this approach may help users to generate or refresh their internal model of the information space, it also implies frequent interaction that, after some time, users are likely to find tedious. Especially so in cases in which they have to zoom out extensively to regain context. While this problem seems to be inherent to ZUIs, a strategy that at least reduces the burden of frequent zooming is to provide a fast and precise interaction design. The less time-consuming and cognitively demanding the zooming is, the less it will annoy users.

A more severe type of orientation problem that cannot be solved solely by interaction has been termed desert fog [12]. Desert fog describes a condition in which users zoom into the white space between objects up to the point where the viewport goes completely blank. On the one hand, the empty screen could be indicating that there are no objects to be found in that direction, in which case users need to zoom out. Another possibility, though, is that there are indeed objects, but they are still too far away to be visible. In this case, users need to zoom in further to approach these objects. A solution to ease such desert-fog conditions is to provide navigational assistance by generating multiscale residues for all objects in view. Such landmarks are drawn across scale and indicate that a particular object exists in that direction. If no residues are seen, the users know that they have to zoom out to find other objects. To avoid visual clutter, navigational information should be clustered.

Interaction Techniques

Different interaction techniques for zooming can be distinguished. Earlier systems were often limited to centralized zooming. Users click a device (or onscreen) button to increase the scale and another button to decrease it. Since the view is only scaled and not translated, this approach implies that for targeting information objects in space, users first have to move the object to the center of the screen. Furthermore, for large information spaces, users may frequently have to interrupt the scaling operation to readjust the focus. A more elegant and effective scaling technique is point-directed zooming, which allows users to magnify and center information objects at the same time. One example is the default event handler in Jazz, and now in the Piccolo framework. While pressing the right mouse button, users scale the view by dragging the mouse right or left to zoom in or out, respectively. The zoom speed increases with the distance the mouse is dragged, and vice versa. During the operation, the point that the cursor was over when the dragging started moves to the center. Another interaction technique, which assumes that users want to return to the highest magnification level after each zoom operation, is speed-dependent automatic zooming (SDAZ) [10]. SDAZ couples rate-based scrolling with scaling in a single operation and was developed to avoid visual blur when navigating large information spaces at high speed. The users control the velocity of a continuous panning operation by dragging the pointing device. To keep the visual flow constant, the system automatically zooms out when the scrolling speed increases and zooms back in when the scrolling speed decreases.

Zoom interaction design may also be based on visual mediators such as sliders. A slider widget has the advantage that the user is provided with additional visual feedback. The position of the slider thumb reveals the current zoom level of the view in relation to the range of scale factors available (e.g., the slider in Google Earth). Users can drag the control to increase or decrease the zoom level. Another example of a visual mediator is zoom bars [11]. Equipped with three thumbs, a zoom bar is a slider that controls the range boundaries of an axis dimension. Moving the two extreme thumbs, users can increase or decrease the upper and lower range boundary, causing a zoom in or a zoom out by changing the scale on the corresponding display axis. That way, any rectangular region can be enlarged to full diagram size. This is also the reason why zoom bars are usually limited to abstract information spaces, in which the two dimensions do not require a fixed aspect ratio. While their similarity to a regular scrollbar may support first-time users in operating the widget, a drawback is that the usability of a zoom bar deteriorates with a decreasing ratio of the physical slider size and the attribute range of the related dimension.

A similar zoom effect as with zoom bars, but without occluding the view, can be achieved with a bounding-box tool. Users drag the pointing device over the view and a focus rectangle is drawn that takes the drag starting point as a corner location and the drag distance as a diagonal. When the users release the pointing device, the defined region is magnified to fit the full size of the view. In cases such as images and text, in which a constant aspect ratio is desired, the focus region is only scaled but not distorted. A bounding-box is a visual mediator that does not require permanent display space, and thus lends itself to the application on small screens such as featured by mobile devices.

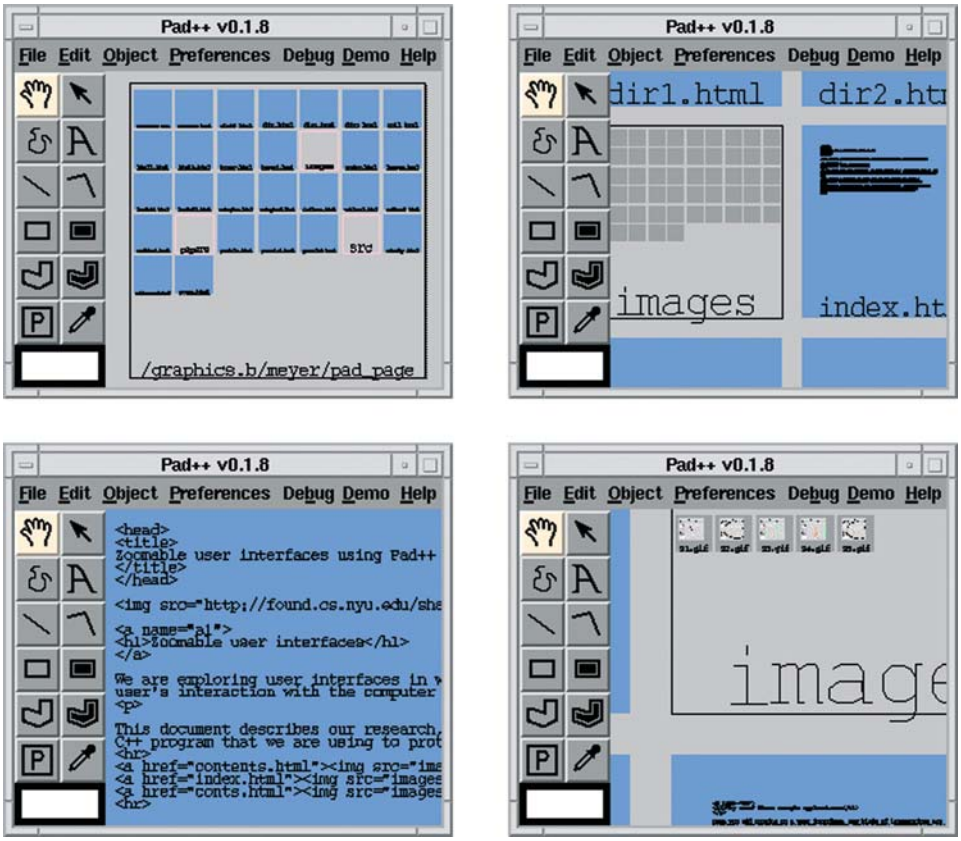
Zoom Granularity and Manipulation

Apart from the interaction design ZUIs can further be categorized by the granularity of zooming they provide. The most basic ZUI is the two-level zoom. It lets users switch between a single overview and a single detail zoom level, and thus only works for small information spaces. To manage larger data sets, several intermediate zoom levels between the minimum and maximum scale must be introduced. In older systems such as, for instance, Pad [15], navigation between the levels is accomplished by discrete jumps. This approach is easy to implement and computationally very effective, but it hampers the usability of the system. Coarse jumps can irritate and disorient the users and thus may hinder the cognitive and perceptual processing required for navigation. Accordingly, a lot of effort has been put into equipping multiscale interfaces with smooth continuous zooming.

ZUIs can offer different types of zooming. Most common is geometric zoom, in which objects are simply magnified. Zooming in, the object's size increases, and vice versa. This approach is found in many standard software applications such as PDF readers or image editors. Semantic zooming, in contrast, is a more sophisticated concept, in which objects change their appearance as the amount of screen real estate available to them changes. In the Pad++ -based directory browser shown in Fig. 3, for instance, subfolders and files are first represented by small-sized icons that only show the name of the object. Increasing the scale, the icons change their appearance to present some more detailed information, e.g., the number of images in a folder, or the structure and amount of text contained in a document. Zooming in further, images become visible and text is magnified to a readable size. At this level, users may also be provided with additional functionality to manipulate the object in focus. Overall, the goal of semantic zooming can be summarized as providing the users with the most meaningful object representation at each magnification level. The difficulty with this approach is that the appropriate representations for all scale levels must be determined in advance by an expert user. However, for complex objects, several representations may be suitable for a given portion of display size. In this case, it is hard to reliably predict the users' requirements. Systems such as DataSplash [14] try to overcome this problem by enabling users to visually program how objects behave during zooming.

Transition Between Zooming

Another important concept for ZUIs is animated transitions. Users may, for instance, click on a hyperlink to automatically move the viewport to a remote location. Or they initiate a scale manipulation via a bounding-box. In both cases the viewport needs to be adjusted. The transition between these interface states can either be instantaneous, which is fastest, or it can be animated by showing intermediate frames. The benefit of smooth and animated transitions is that they help users to maintain relations between



Pad++ v0.1.8
File Edit Object Preferences Debug Demo Help
/graphics.b/meyer/pad_page
src
dir1.html
dir2.htm
images
index.ht
image

```
<head>
<title>
Zoomable user interfaces using Pad++
</title>
</head>

<img src="http://found.cs.nyu.edu/she

<a name="a1">
<h1>Zoomable user interfaces</h1>
</a>

We are exploring user interfaces in w
user's interaction with the computer
<p>

This document describes our research,
C++ program that we are using to prot
<hr>
<a href="contents.html"><img src="ima
<a href="index.html"><img src="images
<a href="conts.html"><img src="images
<hr>
```

Zooming Techniques. Figure 3. A directory browser featuring semantic zooming (http://www.cs.umd.edu./hcil/pad++).

application states. Users are not required to consciously make connections between the changes of interface content and thus they can stay focused on the task. Smooth transitions were also found to have a positive effect on the users' ability to build a mental map of the information space. In a study [1], users were asked to navigate a virtual family tree, in which each node showed a picture of a family member and hyperlinks to connected nodes. Only one or two nodes could be seen at a time. To move the viewport, users clicked on the hyperlinks. The authors discovered that animated transitions of 1 second improved the users' ability to reconstruct the information space, with no penalty on task performance time. To determine the optimal trajectory between two locations in a multiscale interface, some prior research has investigated how to calculate the shortest path with zooming and panning [7], or the path that specifically supports smooth animations [16].

Key Applications

The most common application domain to incorporate zooming techniques is geographical information system, e.g., Google Earth. However, an increasing amount of research prototypes use ZUIs for presenting data without inherent spatial relations. Example domains include web, image, document and database browsers, browsing history widgets, slide show programs, thought organizers, 3D character animation controls, and novel desktop systems.

Future Directions

ZUIs make more efficient use of limited screen real estate, and thus are considered to have a great potential on small-sized mobile devices. Examples include mobile calendars, application explorer, image-, web- and scatterplot-browsers. The Apple iPhone includes a variety of such applications and makes extensive use of zooming techniques in combination with a touch sensitive display.

Experimental Results

Different research studies indicate that, on desktop computers, ZUIs reliably outperform scrolling interfaces in terms of performance and preference [6,13]. Similar effects can be observed for small screens as featured by mobile devices. One study, in which different interfaces were tested on a simulated handheld display, found that a two-level zoom was significantly faster for accomplishing editing and monitoring tasks than a scrolling interface. Even in cases where users failed to achieve optimal performance with the two-level zoom, they preferred it to the other experimental interfaces [8].

Cross-references

▶ Browsing in Digital Libraries
▶ Data Visualization
▶ Human-Computer Interaction
▶ Interface
▶ Mobile Interfaces
▶ Navigation
▶ Scientific Visualization
▶ Usability
▶ Visual Analytics
▶ Visual Data Mining
▶ Visual Interaction
▶ Visual Interfaces
▶ Visualization for Information Retrieval

Recommended Reading

1. Bederson B.B. and Boltman A. Does animation help users build mental maps of spatial information? In Proc. IEEE Symp. on Information Visualization, 1999, p. 28.
2. Bederson B.B., Clamage A., Czerwinski M.P., and Robertson G.G. Datelens: a fisheye calendar interface for PDAs. ACM Trans. Comput. Human Interact., 11(1):90–119, 2004.
3. Bederson B.B. and Hollan J.D. Pad ++ : a zooming graphical interface for exploring alternate interface physics. In Proc. 7th Annual ACM Symp. on User Interface Software and Technology, 1994, pp. 17–26.
4. Bederson B.B., Meyer J., and Good L. Jazz: an extensible zoomable user interface graphics toolkit in java. In Proc. 13th Annual ACM Symp. on User Interface Software and Technology, 2000, pp. 171–180.
5. Donelson W.C. Spatial management of information. In Proc. 5th Annual Conf. Computer Graphics and Interactive Techniques, 1978, pp. 203–209.
6. Donskoy M. and Kaptelinin V. Window navigation with and without animation: a comparison of scroll bars, zoom, and fisheye view. In CHI'97 Extended Abstracts on Human Factors in Computing Systems, 1997, pp. 279–280.
7. Furnas G.W. and Bederson B.B. Space-scale diagrams: understanding multiscale interfaces. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1995, pp. 234–241.
8. Gutwin C. and Fedak C. A comparison of fisheye lenses for interactive layout tasks. In Proc. Graphics Interface, 2004, pp. 213–220.
9. Herot C.F., Carling R., Friedell M., and Kramlich D. A prototype spatial data management system. In Proc. 7th Annual Conf. Computer Graphics and Interactive Techniques, 1980, pp. 63–70.
10. Igarashi T. and Hinckley K. Speed-dependent automatic zooming for browsing large documents. In Proc. 13th Annual ACM Symp. on User Interface Software and Technology, 2000, pp. 139–148.
11. Jog N.K. and Shneiderman B. Starfield visualization with interactive smooth zooming. In Proc. 3rd IFIP WG2.6 Working Conference on Visual Database Systems, vol. 3, 1995, pp. 3–14.
12. Jul S. and Furnas G.W. Critical zones in desert fog: aids to multiscale navigation. In Proc. 11th Annual ACM Symp. on User Interface Software and Technology, 1998, pp. 97–106.
13. Kaptelinin V. A comparison of four navigation techniques in a 2D browsing task. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1995, pp. 282–283.
14. Olston C., Woodruff A., Aiken A., Chu M., Ercegovac V., Lin M., Spalding M., and Stonebraker M. Datasplash. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 550–552.
15. Perlin K. and Fox D. Pad: an alternative approach to the computer interface. In Proc. 20th Annual Conf. Computer Graphics and Interactive Techniques, 1993, pp. 57–64.
16. van Wijk J.J. and Nuij W.A.A. Smooth and efficient zooming and panning. In Proc. IEEE Symp. on Information Visualization, 2003, pp. 15–22.

List of Entries

Absolute Time
Abstract Versus Concrete Temporal Query Languages
Abstraction
Access Control
Access Control Administration Policies
Access Control Policy Languages
Access Methods
Access Path
Accountability
ACID Properties
ACID Transaction
Acquisitional Query Languages
Active and Real-Time Data Warehousing
Active Database, Active Database (Management) System
Active Database Management System Architecture
Active Database Coupling Modes
Active Database Execution Model
Active Database Knowledge Model
Active Database Rulebase
Active Databases
Active Disks
Active Document
Active Storage
Active XML
Activity
Activity Diagrams
Actors/Agents/Roles
Ad hoc Retrieval models
Adaptation
Adaptive Database Replication
Adaptive Interfaces
Adaptive Message-Oriented Middleware
Adaptive Metric Techniques
Adaptive Middleware for Message Queuing Systems
Adaptive Query Optimization
Adaptive Query Processing
Adaptive Stream Processing
Adaptive Workflow/Process Management
ADBMS
ADBMS Framework
ADBMS Infrastructure
Adding Noise
Additive Noise
Administration Model for RBAC
Administration Wizards
Advanced Transaction Models
Adversarial Information Retrieval
Affix Removal
AFI
Aggregate Queries in P2P Systems
Aggregation
Aggregation Algorithms for Middleware Systems
Aggregation and Threshold Algorithms for XML
Aggregation: Expressiveness and Containment
Aggregation-Based Structured Text Retrieval
AGMS Sketch
Air Indexes for Spatial Databases
AJAX
Allen's Relations
AMOSQL
AMS Sketch
Analogy
Anchor
Anchor Text
Anchor Text Surrogate
AND-Join
AND-Split
Animation
Annotation
Annotation-based Image Retrieval
Anomaly Detection on Streams
Anonymity
Anonymity in Location-based Services
ANSI/INCITS RBAC Standard
Answering Queries Using Views
Anti-monotone Constraints
AP@n
Applicability Period
Application Benchmark
Application Recovery
Application Server
Application Server Clustering
Application-Centric Interfacing
Application-Level Tuning
Applications of Emerging Patterns for Microarray Gene Expression Data Analysis
Applications of Sensor Network Data Management
Application-to-Application Integration
Approximate Queries in Peer-to-Peer Systems

Approximate Query Answering
Approximate Query Processing
Approximate Querying
Approximate Reasoning
Approximate XML Querying
Approximation of Frequent Itemsets
Apriori Property and Breadth-First Search Algorithms
ARBAC97
Architecture-aware Database System
Architecture-Conscious Database System
Architecture-Sensitive Database System
Archiving Experimental Data
Armstrong Axioms
Array
Array Databases
Association
Association Rule Mining on Streams
Association Rule Visualization
Association Rules
Associative Classification
Asymmetric Encryption
ATA
Atelic Data
Atomic Event
Atomicity
Attribute or Value Correspondence
Audible Sound
Audio
Audio Categorization
Audio Characterization
Audio Classification
Audio Content Analysis
Audio Feature Extraction
Audio Indexing
Audio Information Retrieval
Audio Metadata
Audio Parsing
Audio Recognition
Audio Representation
Audio Segmentation
Audit Trail
Auditing and Forensic Analysis
Auditory Scene Detection
Authentication
Authentication Trees
Authorization Administration Policies
Authorization Administration Privileges
Authorization Policy Languages
Authorization Verification
Auto-administration and Auto-Tuning of Database Systems
Auto-Annotation
Automata Induction
Automatic Abstracting
Automatic Image Annotation
Automatic Induction
Automatic Language Induction
Automatic Wrapper Induction
Autonomic Database Replica Allocation
Autonomic Database Systems
Autonomic Query Processing
Autonomous Message Queuing Systems
Autonomous Replication
Autonomous Message-oriented Middleware
Average Precision
Average Precision at n
Average Precision Histogram
Average R-Precision
AXML
B+-Tree
Backup
Backup and Restore
Backup Copy
Backup Mechanisms
Backward Recovery
Bag Semantics
Bagging
Base-line Clock
Bayes Classifier
Bayesian Classification
BCNF
Belief Time
Benchmark
Biased Distribution
Bibliography
Bi-clustering
Bioinformatics
Biological Data Retrieval, Integration, and Transformation
Biological Metadata Management
Biological Networks
Biological Pathways
Biological Query Languages
Biological Resource Discovery
Biological Sequences
Biomedical Data Annotation
Biomedical Data/Content Acquisition, Curation
Biomedical Image Data Types and Processing

Biomedical Informatics
Biomedical Literature
Biomedical Scientific Textual Data Types and Processing
Biostatistics and Data Analysis
BIR Model
Bit Vector Join
Bi-temporal Access Methods
Bitemporal Algebras
Bitemporal Data Model
Bi-Temporal Indexing
Bitemporal Interval
Bitemporal Relation
Bitmap Index
Bitmap-based Index Structures
Blind Signatures
Bloom Filter Join
Bloom Filters
Bloom Join
BM25
Boolean Model
Boosting
Bootstrap
Bootstrap Aggregating
Bootstrap Estimation
Bootstrap Sampling
Bottom-up Semantics
Boyce-Codd Normal Form
BP-Completeness
BPEL
BPEL4WS
BPMN
Bpref
Branch
Bridging
Browsing
Browsing in Digital Libraries
B-Tree
B-Tree Concurrency Control
B-Tree Locking
Buffer Management
Buffer Manager
Buffer Pool
Business Intelligence
Business Process Execution Language
Business Process Management
Business Process Model
Business Process Modeling
Business Process Modeling Notation
Business Process Monitoring
Business Process Optimization
Business Process Redesign
Business Process Reengineering
Cache Manager
Cache Performance
Cache-Aware Query Processing
Cache-Conscious Query Processing
Cache-Sensitive Query Processing
Calculus Expression
Calendar
Calendric System
Camera Break Detection
Capsule
Cardinal Direction Relationships
Cartesian Product
Cartography
CAS
CAS Query
Case Handling
Case Management
Case Report Forms
Cataloging
Cataloging in Digital Libraries
CDA
CDA R1
CDA R2
CDP
CDs
CDS
Cell Complex
Certain (and Possible) Answers
Chandra and Harel Complete Query Languages
Change Detection and Explanation on Streams
Change Detection on Streams
Channel-Based Publish/Subscribe
Chart
Chase
Checkpoint
Checksum and Cyclic Redundancy Check Mechanism
Choreography
Chronicle Recognition
Chronon
CIFS
Cipher
Citation
CLARA (Clustering LARge Applications)
CLARANS (Clustering Large Applications Based Upon Randomized Search)

Classification
Classification by Association Rule Analysis
Classification Learning
Classification in Streams
Classification Tree
Classification Trees
Classifier Combination
Client-Server DBMS
Clinical Classifications
Clinical Content Database
Clinical Content Registry
Clinical Content Repository
Clinical Data Acquisition, Storage and Management
Clinical Data and Information Models
Clinical Data Management Systems
Clinical Data Quality and Validation
Clinical Decision Support
Clinical Document Architecture
Clinical Event
Clinical Genetics
Clinical Genomics
Clinical Judgment
Clinical Knowledge Base
Clinical Knowledge Directory
Clinical Knowledge Management Repository
Clinical Knowledge Repository
Clinical Nomenclatures
Clinical Observation
Clinical Ontologies
Clinical Order
Clinical Research Chart
Clinical Result
Clinical Terminologies
Clinical Test
Clock
Closed Itemset Mining and Non-redundant Association Rule Mining
Closest Pairs
Closest-Pair Query
Cloud Computing
Cluster and Distance Measure
Cluster Database Replication
Cluster Databases
Cluster Replication
Cluster Stability
Cluster Validation
Clustering
Clustering for Post Hoc Information Retrieval
Clustering Index
Clustering on Streams
Clustering Overview and Applications
Clustering Validity
Clustering with Constraints
CM Sketch
CMA
CO Query, Content-Only Query
CO+S Query
Co-clustering
CODASYL Data Model
Collaborative Software
Co-locations
Colored Nets
Column Segmentation
Committee-based Learning
Common Object Request Broker Architecture
Common Subexpression Elimination
Common Warehouse Metadata Interchange (CWMI)
Common Warehouse Metamodel
Communication Boundary of a DBMS
Compact Suffix Tries
Comparative Analysis
Comparative Visualization
Compensating Transactions
Computationally Complete Relational Query Languages
Complex Event
Complex Event Processing
Complex Event Processing (CEP)
Compliance
Component Abstraction
Composed Services and WS-BPEL
Composite Event
Composite Event Query
Composite Web Applications
Composition
Comprehensions
Compressed and Searchable Data Format
Compressed Full-Text Indexing
Compressed Suffix Array
Compressed Suffix Tree
Compressing XML
Compression of Mobile Location Data
Computational Media Aesthetics
Computational Ontology
Computer Human Interaction (CHI)
Computer-based Physician Order Entry
Computer-based Provider Order Entry
Computer-Interpretable Formalism

Computerized Order Entry (COE)
Computerized Physician Order Entry
Computerized Provider Order Entry
Concept Languages
Conceptual Data Model
Conceptual Image Data Model
Conceptual Model
Conceptual Modeling
Conceptual Modeling for Geographic Information System
Conceptual Modeling for Spatio-Temporal Applications
Conceptual Schema Design
Conceptual Schemas
Concurrency Control
Concurrency Control – Traditional Approaches
Concurrency Control and Recovery
Concurrency Control Manager
Condition Event Nets
Conditional Branching
Conditional Routing
Conditional Tables
Confidentiality Protection
Conflict Serializability
Conjunctive Query
Connection
Connectionist Model
Consistency in Peer-to-Peer Systems
Consistency Models For Replicated Data
Consistency Preservation
Consistent Facts
Consistent Query Answering
Constant Span
Constrained Frequent Itemset Mining
Constraint Databases
Constraint Query Languages
Constraint-Driven Database Repair
Content Delivery Networks
Content-and-Structure Query
Content-based Image Retrieval (CBIR)
Content-Based Publish/Subscribe
Content-based Retrieval
Content-Based Video Retrieval
Content-Only Query
Content-oriented XML Retrieval
Context
Context-aware Interfaces
Contextual Advertising
Contextualization
Continuous Backup
Continuous Data Feed
Continuous Data Protection
Continuous Monitoring of Spatial Queries
Continuous Multimedia Data Retrieval
Continuous Queries in Sensor Networks
Continuous Query
Continuous Query Languages
Continuous Query Processing Applications
Continuous Query Scheduling
ConTract
ConTracts
Contrast Pattern
Contrast Pattern Based Classification
Control Data
Control Flow Diagrams
Controlled Vocabularies
Controlling Overlap
Convertible Constraints
Cooperative Classification
Cooperative Content Distribution
Cooperative Storage Systems
Coordination
||-Coords
Copy Divergence
Copy Transparency
Copyright Issues in Databases
CORBA
Corpora
Corpus
Correctness Criteria Beyond Serializability
Correctness Criterion for Concurrent Executions
Correlated Data Collection
Correlation
Correlation Clustering
Cost Estimation
Count-Min Sketch
Coupling and De-coupling
Coverage
Covering Index
Covert Communication
CPU Cache
Crabbing
Crash Recovery
Crawler
Credulous Reasoning
Cross Product
Cross-language Cross-Language Mining and Retrieval
C217 Informational Retrieval

Cross-Language Mining and Retrieval
Cross-language Text Mining
Cross-language Web Mining
Cross-lingual Information Retrieval
Cross-lingual Text Mining
Cross-media Information Retrieval
Cross-Modal Multimedia Information Retrieval
Cross-Validation
Cryptographic Hash Functions
C-Tables
Cube
Cube Implementations
Cube Materialization
Cube Precomputation
Curation
Current Date
Current Semantics
Current Time
Current Timestamp
Curse of Dimensionality
Cursor
CW Complex
CWM
Cyclic Redundancy Check (CRC)
DAC
Daplex
DAS
Data Acquisition
Data Acquisition and Dissemination in Sensor Networks
Data Aggregation in Sensor Networks
Data Analysis
Data Anomalies
Data Broadcasting, Caching and Replication in Mobile Computing
Data Cache
Data Cleaning
Data Collection
Data Compression in Sensor Networks
Data Confidentiality
Data Conflicts
Data Copy
Data Corruption
Data Deduplication
Data Dependency
Data Dictionary
Data Dissemination
Data Encryption
Data Errors
Data Estimation in Sensor Networks
Data Exchange
Data Expiration
Data Extraction
Data Flow Diagrams
Data Fusion
Data Fusion in Sensor Networks
Data Gathering
Data Grids
Data Imputation
Data Inconsistencies
Data Integration
Data Integration Architectures and Methodology for the Life Sciences
Data Integration in Web Data Extraction System
Data Integrity Services
Data Lineage
Data Manipulation Language
Data Map
Data Mart
Data Migration
Data Mining
Data Mining in Bioinformatics
Data Mining in Computational Biology
Data Mining in Moving Objects Databases
Data Mining in Systems Biology
Data Mining Pipeline
Data Mining Process
Data Model Mapping
Data Organization
Data Partitioning
Data Pedigree
Data Perturbation
Data Privacy and Patient Consent
Data Problems
Data Profiling
Data Protection
Data Provenance
Data Quality
Data Quality Assessment
Data Quality Attributes
Data Quality Benchmarking
Data Quality Criteria
Data Quality Dimensions
Data Quality Measurement
Data Quality Models
Data Quality Problems
Data Quality Representations
Data Rank/Swapping

Data Reconciliation
Data Reduction
Data Replication
Data Replication Protocols
Data Sampling
Data Sketch/Synopsis
Data Skew
Data Sorts
Data Standardization
Data Storage and Indexing in Sensor Networks
Data Stream
Data Stream Algorithm
Data Stream Management Architectures and Prototypes
Data Stream Processing
Data Suppression
Data Swapping
Data Time
Data Tracking
Data Transformation
Data Translation
Data Types for Moving Objects
Data Types in Scientific Data Management
Data Types: Image, Video, Pixel, Voxel, Frame
Data Uncertainty Management in Sensor Networks
Data Utility Measures
Data Visualizations
Data Visualization
Data Warehouse
Data Warehouse Back Stage
Data Warehouse Design Methodology
Data Warehouse Life-Cycle and Design
Data Warehouse Maintenance, Evolution and Versioning
Data Warehouse Indexing
Data Warehouse Integration
Data Warehouse Metadata
Data Warehouse Query Processing
Data Warehouse Refreshment
Data Warehouse Security
Data Warehousing for Clinical Research
Data Warehousing Systems: Foundations and Architectures
Data, Text, and Web Mining in Healthcare
Database Adapter and Connector
Database Clustering Methods
Database Clusters
Database Connectivity
Database Constraints
Database Dependencies
Database Design
Database Design Recovery
Database Engine
Database Implementation
Database Interaction
Database Languages for Sensor Networks
Database Machine
Database Management System
Database Materialization
Database Middleware
Database Physical Layer
Database Profiling
Database Protection
Database Provisioning
Database Redocumentation
Database Repair
Database Replication
Database Reverse Engineering
Database Scheduling
Database Security
Database Socket
Database Storage Layer
Database Techniques to Improve Scientific Simulations
Database Trigger
Database Tuning using Combinatorial Search
Database Tuning using Online Algorithms
Database Tuning using Trade-off Elimination
Database Use in Science Applications
Databases for Biomedical Images
Dataguide
Datalog
Datalog Query Processing and Optimization
Datastream Distance
Datawarehouses Confidentiality
DBC
DBMS
DBMS Component
DBMS Interface
DBTG Data Model
DCE
DCOM
Deadlocks in Distributed Database Systems
Decay Models
Decentralized Data Integration System
Decision Rule Mining in Rough Set Theory
Decision Rules
Decision Support
Decision Tree

Decision Tree Classification
Decision Trees
Declarative Networking
Declarative Overlay Networks
Deductive Data Mining using Granular Computing
Deductive Data Mining, Model for Automated Data Mining
Deductive Databases
Dedup
Deduplication
Deduplication in Data Cleaning
Deep-Web Search
Degrees of Consistency
DEMs
Degrees of Cosistency
Dendrogram
Dense Index
Dense Pixel Displays
Density-based Clustering
Dependencies
Derived Event
Description Logics
Design for Data Quality
Design for Quality
Desktop Metaphor
Detail-in-Context
Deviation from Randomness
Dewey Decimal Classification
Dewey Decimal System
DHT
Diagram
Diagrammatic Representation
Difference
Digital Archives and Preservation
Digital Curation
Digital Elevation Models
Digital Image
Digital Libraries
Digital Rights Management
Digital Signatures
Digital Surface Model
Digital Terrain Model (DTM)
Digital Video Retrieval
Digital Video Search
Dimension
Dimensional Modeling
Dimensionality Curse
Dimensionality Reduction
Dimension Reduction Techniques for Clustering
Dimension-Extended Topological Relationships
Direct Attached Storage
Direct Manipulation
Direct Manipulation Interfaces
Directional Relationships
Dirichlet Tessellation
Disaster Recovery
Disclosure Risk
DISCO
Discounted Cumulated Gain
Discovery
Discrete Wavelet Transform and Wavelet Synopses
Discretionary Access Control
Disk
Disk Array
Disk Drive
Disk Power Saving
Disk Process
Disk-based Model
Distance between Streams
Distance Indexing
Distance Space
Distance-preserving Mapping
Distillation
Distortion Techniques
Distributed Architecture
Distributed Commit Protocol
Distributed Component Object Model
Distributed Computing Environment
Distributed Concurrency Control
Distributed Data Streams
Distributed Database Design
Distributed Database Management System (DDBMS)
Distributed Database Systems
Distributed Databases
Distributed DBMS
Distributed Deadlock Management
Distributed Hash Table
Distributed Join
Distributed Query
Distributed Query Optimization
Distributed Query Processing
Distributed Recovery
Distributed Sensor Fusion
Distributed Source Coding
Distributed Spatial Databases
Distributed Storage Systems
Distributed Transaction Management
Divergence Control

Divergence from Randomness Models
DNA Sequences
Document
Document Clustering
Document Databases
Document Field
Document Formats
Document Identifier
Document Index and Retrieval
Document Length Normalization
Document Links and Hyperlinks
Document Management
Document Path Query
Document Repositories
Document Representations (Inclusive Native and Relational)
Document Retrieval
Document Segmentation
Document Summarization
Document Term Weighting
Document Visualization
Documents
Domain Relational Calculus
Downward Closure Property
DRM
Dublin Core
Dump
Duplicate Detection
Duplicate Semantics
Duplication
Durability
Duration
DVDs
DW
Dynamic Graphics
Dynamic Integrity Constraints
Dynamic Taxonomies
Dynamic Web Pages
eAccessibility
EAI
EC Transactions
ECA Rule Action
ECA Rule Condition
ECA Rules
ECM
e-Commerce Transactions
Eddies
Edge Detection
eDictionary
eEncyclopedia
EERM, HERM
Effectiveness Involving Multiple Queries
EF-Games
egd
EHR
Ehrenfeucht-Fraïssé Games
Ehrenfeucht Games
Electronic Commerce Transactions
Electronic Data Capture
Electronic Dictionary
Electronic Encyclopedia
Electronic Health Record
Electronic Ink Indexing
Electronic Libraries
Electronic Newspapers
Eleven Point Precision-recall Curve
Embedded Networked Sensing
Emergent Semantics
Emerging Pattern Based Classification
Emerging Patterns
Encryption
End User
Ensemble
Enterprise Application Integration
Enterprise Content Management
Enterprise Information Integration
Enterprise Service Bus
Enterprise Terminology Services
Entity Relationship Model
Entity Resolution
Entity-Relationship Model
EPN
Equality Query
Equality Selection
Equality-Generating Dependencies
ER Model
ERM
ESB
Escrow Transactions
e-Services
ETL
ETL Process
ETL Tool
ETL Using Web Data Extraction Techniques
European Law in Databases
Evaluation Forum
Evaluation in Information Retrieval
Evaluation Measures

Evaluation Metrics for Structured Text Retrieval
Evaluation of Fuzzy Queries Over Multimedia Systems
Evaluation of Relational Operators
Evaluation of XML Retrieval Effectiveness
Event
Event and Pattern Detection over Streams
Event Broker
Event Causality
Event Causality Graph
Event Channel
Event Cloud
Event Composition
Event Composition (Partial Overlap)
Event Connection
Event Consumer
Event Control
Event Declaration
Event Definition
Event Detection
Event Driven Architecture
Event Driven Service-oriented Architecture
Event Emitter
Event Extraction
Event Flow
Event in Active Databases
Event in Temporal Databases
Event Lineage
Event Mapping
Event Network Edge
Event Pathway
Event Pattern Detection
Event Pedigree
Event Pipe
Event Prediction
Event Processing
Event Processing Agent
Event Processing Component
Event Processing Mediator
Event Processing Network
Event Processing Systems
Event Producer
Event Relation
Event Service
Event Sink
Event Source
Event Specification
Event Stream
Event Stream Processing
Event Stream Processing (ESP)
Event Topic
Event Trace Analysis
Event Transformation
Event Uncertainty
Event-Condition-Action Rules
Event-Driven Business Process Management
Eventual Consistency
Evidence Based Medicine
Evidence Based Practice
Evolutionary Algorithms
Evolutionary Computation
Evolutionary Semantics
Exactly Once Execution
Executable Knowledge
Execution Skew
Exhaustivity
Existence Time
Explicit Event
Exploratory Data Analysis
Exploring
Expressive Power of Query Languages
Extended Entity-Relationship Model
Extended Functional Dependencies
Extended Relations
Extended Transaction Models
Extended Transaction Models and the ACTA Framework
Extendible Hashing
Extensible Markup Language
eXtensible Stylesheet Language
eXtensible Stylesheet Language Transformations
Extensional Relational Database (ERDB)
External Hashing
Extraction, Transformation, and Loading
Extrinsic Time
Faceted Browsing
Faceted Classifications
Faceted Search
Facility-Location Problem
Fact-Oriented Modeling
Failure Handling
False Negative Rate
Fault Tolerant Applications
Fault-Tolerance
Fault-Tolerance and High Availability in Data Stream Management Systems
FCP
FD
Feature Extraction for Content-Based Image Retrieval

Feature Selection for Clustering
Feature-Based 3D Object Retrieval
Federated Database
Federated Database Systems
Federated Search
Federated Search Engine
Feedback Systems
Field-Based Information Retrieval Models
Field-Based Spatial Modeling
File Compression
File Format
Filter/Refinement Query Processing
Finding of Observation
Finiteness
First-Order Logic: Semantics
First-Order Logic: Syntax
First-Order Query
Fisheye Views
Fixed Time Span
Flajolet-Martin Algorithm
Flajolet-Martin Sketch
Flake
Flash
Flex Transactions
Flexible Metric Computation
Flexible Transactions
Flowcharts
FM Sketch
FM Synopsis
F-Measure
Focused Retrieval
Focused Web Crawling
Focus-Plus-Context
FOL
FOL Modeling of Integrity Constraints (Dependencies)
Forever
Form
Forms-based Interfaces
Fourth Normal Form
FQL
Fractal
Frequency Moments
Frequent Concepts
Frequent Elements
Frequent Graph Patterns
Frequent Items on Streams
Frequent Itemset Mining with Constraints
Frequent Itemsets and Association Rules
Frequent Partial Orders
Frequent Pattern Mining with Constraints
Frequent Patterns
Frequent Set Mining with Constraints
Frequent Subsequences
Freshness Control
Full Text Inverted Index
Fully-Automatic Web Data Extraction
Fully Temporal Relation
Functional Data Model
Functional Dependencies for Semi-Structured Data
Functional Dependency
Functional Query Language
Fuzzy Information Retrieval
Fuzzy MCDM
Fuzzy Models
Fuzzy Multicriteria Decision Making
Fuzzy Relation
Fuzzy Set
Fuzzy Set Approach
Fuzzy Time
Fuzzy/Linguistic IF-THEN Rules and Linguistic Descriptions
Gaifman-Locality
Gazetteers
Gene Expression Arrays
Generalisation
Generalization of ACID Properties
Generalized Search Tree
Generative Models
Genetic Algorithms
Geographical Information Retrieval
Geographic Information Services
Geographic Information System
Geographic Information
Geographic Web Search
Geographical Analysis
Geographical Data Analysis
Geographical Databases
Geographical Metadata
Geography Markup Language
Geometric Data Types
Geometric Mean Average Precision
Geometric Stream Mining
GEO-RBAC Model
Georeferencing
Geoscientific Information System
Geospatial Information System
Geospatial Metadata

Geospatial Referencing
Geo-Targeted Web Search
GIS
GIST
GiST
Global Query Optimization
Glyphs
GMAP
Google Bombing
Grammar Induction
Grammar Inference
Grammatical Induction
Grammatical Inference
Graph
Graph
Graph Data Management in Scientific Applications
Graph Data Structure
Graph Database
Graph Database Mining
Graph Drawing
Graph Embedding
Graph Layout
Graph Management in the Life Sciences
Graph Mining on Streams
Graph Streams
Graph Theory
Graph-based Clustering
Graphic
Graphic Design
Graphic Representation of Data
Graphical Displays of Many Variables
Graphical Interaction
Graphical Representation
Graphical User Interfaces
Graphics
Graphics for Continuous Data
Graphics for Discrete Data
Grid and Workflows
Grid File
Grid File (and Family)
Grid Workflow
Group Difference
Grouping
GUID
GUIs for Web Data Extraction
Handhelds Interfaces
Handwritten Text
Hanf-Locality
Hard Disk
Hardware-Conscious Database System
Harmonic Mean of Recall and Precision
Hash File
Hash Filter
Hash Filter Join
Hash Functions
Hash Join
Hash Trees
Hash-based Indexing
Hater's Model
HCC
HCI
HCM
Health Informatics
Healthcare Informatics
Heat Map
Heavy Hitters
Heterogeneous Distributed Database Systems
Heterogeneously Distributed Data
HHH
Hidden-Web Search
Hierarchial Clustering
Hierarchical Data Model
Hierarchical Data Organization
Hierarchical Data Summarization
Hierarchical Entity-Relationship Model
Hierarchical Faceted Metadata
Hierarchical Graph Layout
Hierarchical Heavy Hitter Mining on Streams
Hierarchical Memory System
Hierarchical Regular-Decomposition Structures
Hierarchical Spatial Indexes
Hierarchical Storage Management
Hierarchical Visualization
Hierarchies
Hierarchy
High Dimensional Indexing
High-Dimensional Clustering
Higher-Order Entity-Relationship Model
Histogram
Histograms on Streams
Historical Algebras
Historical Data Model
Historical Data Models
Historical Database
Historical Query Languages
Historical Spatio-Temporal Access Methods
History
History in Temporal Databases

Homogeneous Distributed Database Systems
Homogeneously Distributed Data
Homomorphic Encryption
Horizontal Fragmentation
Horizontally Partitioned Data
Horn Clause Query
Hot Items
Hotspots
HSM
HTML Fragment
Human Centered 38 H Human-Computer Interaction Computing
Human Factors
Human Interface
Human-centered Computing: Application to Multimedia
Human-Centered Multimedia
Human-Computer Interaction
Hypercube
Hypermedia
Hypermedia Metadata
Hypertexts
Hypothesis Generation and Exploration from Biological Resources
I/O cache
I/O Model of Computation
Icon
Iconic Displays
Iconographics
Icons
Identity Disclosure
Identity-based Access Control
IDF
IF
ILM
Image
Image Classification
Image Compression
Image Content
Image Content Modeling
Image Data Model
Image Database
Image Distance
Image Indexing
Image Management for Biological Data
Image Management for Life Sciences
Image Metadata
Image Query Processing
Image Querying
Image Representation
Image Similarity
Image Retrieval
Image Retrieval and Relevance Feedback
Image Retrieval System
Image Segmentation
Image Standards
Image/Video/Music Search
Immersive Data Mining
Implementation Abstraction
Implication of Constraints
Implications of Genomics for Clinical Informatics
Implicit Event
Imprecise Data
Imprecise Spatial Queries
Imprecise Time
Imputed Data
IMS Data Model
In Silico Experiment
Incoherency Bounds
Incomplete Information
Inconsistent Databases
Incremental Computation of Queries
Incremental Crawling
Incremental Maintenance of Views with Aggregates
Incremental k-Distance Join
Incremental Maintenance of Recursive Views
Incremental View Maintenance
Indefinite Information
Index Creation and File Structures
Index Join
Index Loop Join
Index Nested Loop Join
Index Sequential Access Method (ISAM)
Index Structures for Biological Sequences
Index Tuning
Indexed Sequential Access Method
Indexed Sequential File
Indexing
Indexing and Similarity Search
Indexing Compressed Text
Indexing for Online Function Approximation
Indexing for Similarity Search
Indexing Granularity
Indexing Historical Spatio-Temporal Data
Indexing Metric Spaces
Indexing of Data Warehouses
Indexing of the Current and Near-Future Positions of Moving Objects

Indexing the Past
Indexing the Web
Indexing Units
Individual Data
Individually Identifiable Data
INEX
Inference Control in Statistical Databases
Infinity
Information
Information Browsing
Information Displays
Information Extraction
Information Filtering
Information Foraging
Information Graphic
Information Hiding
Information Integration
Information Integration Techniques for Scientific Data
Information Lifecycle Management
Information Loss Measures
Information Navigation
Information Quality: Managing Information as a Product
Information Quality and Decision Making
Information Quality Assessment
Information Quality Policy and Strategy
Information Repository
Information Retrieval
Information Retrieval Models
Information Retrieval Models/Metrics/Operations
Information Retrieval Operations
Information Retrieval Processing
Information Seeking
Information Visualization
Information visualization on hierarchies
Information Visualization on Networks
INitiative for the Evaluation of XML Retrieval
Initiator
In-Memory DBMS
In-Network Aggregation
In-Network Query Processing
Instance Identification
Instance-Completeness
Instant
Instant Relation
Instruction Cache
Integrated DB&IR Semi-Structured Text Retrieval
Integration of Rules and Ontologies
Intellectual Property
Intelligent Disks
Intelligent Storage
Intelligent Storage Systems
Interaction Design
Interactive Capture
Interactive Information Exploration
Interactive Layout
Interactive Visual Exploration of Multidimensional Data
Interface
Interface Engines in Healthcare
Internet Transactions
Interoperability in Data Warehouses
Interoperation of NLP-based Systems with Clinical Databases
Inter-Operator Parallelism
Inter-Query Parallelism
Interval-based Temporal Models
Intra-operator Parallelism
Intra-Query Parallelism
Intrinsic Time
Intrusion Detection Technology
Inverse Document Frequency
Inverse Element Frequency
Inverted Files
Inverted Index
Inverted Indexes
IP Storage
ISAM File
iSCSI
ISO 19136
Isolation
Iteration
Iterator
J2EE
Java Annotations
Java Application Server
Java Database Connectivity
Java EE
Java Enterprise Edition
Java Metadata Facility
JD
JDBC
Join
Join Dependency
Join Index
Join Indices
Join Order
Join Processing

Join Sequence
JSR 175
k-Anonymity
k-Closest Pair Join
k-Closest Pair Query
KDD Pipeline
KDD Process
k-Distance Join
Key
Key Range Locking
Key Value Locking
KL-ONE Style Languages
K-Means and K-Medoids
K-Means Partition
k-Nearest Neighbor Classification
k-NN Classification
kNN Query
Knowledge Creation
Knowledge Discovery from Biological Resources
Knowledge Discovery from Data
Knowledge Discovery in Streams
Knowledge Discovery in Text (KDT)
Knowledge Management
Knowledge Organization Systems
Knowledge-based Systems
Koch Snowflake
L1 Cache
L2 Cache
L3 Cache
Language Models
Languages for Web Data Extraction
Large Itemsets
Latch Coupling
Latching
Latent Semantic Indexing
Layer Algebra
Layered Architecture
Layered Transactions
Lazy Replication
LBS
Lévy Skew a-Stable Distribution
Learning Distance Measures
Learning in Streams
Length Normalization
Level-of-Detail (LOD) Terrain Modeling
Levelwise Search
Lexical Affinities
Lexical Analysis of Textual Data
Lexical Processing
Lexical Relations
Library of Congress METS
License
Licensing and Contracting Issues in Databases
Lifespan
Life-span (in Part)
Lightweight Ontologies
Lineage
Linear Hashing
Linear Regression
Linearization
Link Analysis
Link Database
Linked Brushing
Linked Views
Linking and Brushing
List
List Comprehension
Literature-based Discovery from Biological Resources
Load Balancing
Load Balancing in Peer-to-Peer Overlay Networks
Load Shedding
LOC METS
Local Web Search
Localization Abstraction
Locality
Locality of Queries
Locality of Reference
Locality Principle
Locality-Preserving Mapping
Location Prediction
Location Services
Location-Based Services
Lock Coupling
Lock Manager
Lock Tuning
Locking Granularity and Lock Types
Locking Protocol
Log Component
Log Manager
Logging and Recovery
Logging/Recovery Subsystem
Logic of Time
Logical Database Design: from Conceptual to Logical Schema
Logical Foundations of Web Data Extraction
Logical Models of Information Retrieval
Logical Query Processing and Optimization
Logical Schema Design

Logical Story Unit Segmentation
Logical Structure
Logical Time
Logical Unit Number
Logical Unit Number Mapping
Logical Volume
Logical Volume Manager
Logical Window
Log-Linear Regression
Long Running Queries
Longitudinal Health Record
Looking Over/Through
Loop
Loop Join
Loose Coupling
Lossless Data Compression
LoT-RBAC
Low Coupling
Lp Distances
Lp Norms
LSID
LUN
LUN Mapping
LUN Masking
LVM
MAC
Machine Learning in Bioinformatics
Machine Learning in Computational Biology
Machine Learning in Systems Biology
Machine-Readable Dictionary (MRD)
Macro
Magnetic Disk
Maid
Main Memory
Main Memory DBMS
Maintenance of Materialized Views with Outer-Joins
Maintenance of Recursive Views
Managing Compressed Structured Text
Mandatory Access Control
MANET Databases
Manmachine Interaction (Obsolete)
Many Sorted Algebra
MAP
Map Algebra
Map Matching
Mapping
Mapping Composition
Mapping Engines
Markup Language
MashUp
Massive Array of Idle Disks
Matching
Materialized Query Tables
Materialized View Maintenance
Materialized View Redefinition
Materialized Views
Matrix
Matrix Masking
Maximal Itemset Mining
Max-Pattern Mining
Maybe Answer
MDIS
MDR
MDS
Mean Average Precision
Mean Reciprocal Rank
Mean Reciprocal Rank of the First Relevant Document
Measure
Media Recovery
Media Semantics
Median
Mediation
Mediation and Adaptation
Medical Genetics
MEDLINE/ PubMed
Membership Query
Memory Consistency
Memory Hierarchy
Memory Locality
Merge Join
Merge-purge
Merkle Hash Trees
Merkle Trees
Message Authentication Codes
Message Integrity Codes
Message Queuing Systems
Message-Oriented Middleware (MOM)
Message-oriented Systems
Messaging Engines
Messaging Systems
Meta Data Base
Metadata Interchange Specification
Meta Data Management System
Meta Data Manager
Meta Data Registry
Meta Data Repository
Meta Model
Meta Object Facility

Metadata
Metadata Encoding and Transmission Standard
Metadata Registry, ISO/IEC 11179
Metadata Repository
Meta-Knowledge
Metamodel
Metaphor
Metasearch Engines
Metric Space
Microdata
Microaggregation
Microbenchmark
Microdata Rounding
Middleware Support for Database Replication and Caching
Middleware Support for Precise Failure Semantics
Mini
Minimal-change Integrity Maintenance
Mining of Chemical Data
Mixed Evidence
Mixed-Media
MM Indexing
MMDBMS
Mobile Ad hoc Network Databases
Mobile Database
Mobile Interfaces
Mobile Map Services
Mobile Sensor Network Data Management
Mobile Wireless Sensor Network Data Management
Model Management
Model-based Querying in Sensor Networks
Model-driven Data Acquisition
Module
MOF
Molecular Interaction Graphs
Moment
Monitoring
Monitoring of Real-Time Logic Expressions
Monotone Constraints
Monotonic Constraints
Monotonicity Property
Motion Graphics
Moving Object
Moving Object Trajectories
Moving Objects Databases and Tracking
Moving Objects Interpolation
Moving Span
MRR
MRR1
MSN Data Management
Multi-Database
Multidatabases
Multidimensional Clustering
Multidimensional Data Formats
Multidimensional Database Management System
Multi-dimensional Mapping
Multidimensional Modeling
Multidimensional Scaling
Multidimensional Visualization
Multi-Granularity Modeling
Multi-Layered Architecture
Multi-Level Recovery and the ARIES Algorithm
Multilevel Secure Database Management System
Multilevel Security
Multilevel Transactions and Object-Model Transactions
Multi-Level Visualization
Multilingual Information Retrieval
Multi-Master System
Multimedia
Multimedia Content Enrichment
Multimedia Data
Multimedia Data Buffering
Multimedia Data Indexing
Multimedia Data Querying
Multimedia Data Storage
Multimedia Databases
Multimedia Information Discovery
Multimedia Information Retrieval
Multimedia Information Retrieval Model
Multimedia Metadata
Multimedia Presentation Databases
Multimedia Resource Scheduling
Multimedia Retrieval Evaluation
Multimodal Data
Multimodal Databases
Multi-modal Information Retrieval
Multimodal Interfaces
Multi-Pathing
Multiple Classifier System
Multiple Imputation
Multiple Linked Plots
Multiple Query Optimization
Multiple Representation Modeling
Multiplicity
Multiprocessor Data Placement
Multiprocessor Database Management
Multiprocessor Query Processing

Multi-Query Optimization
Multi-Resolution
Multi-Resolution Terrain Modeling
Multi-scale
Multiscale Views
Multiscale Interface
Multiset Semantics
Multi-Step Query Processing
Multi-Tier Architecture
Multivalued Dependency
Multivariate Data Visualization
Multivariate Visualization Methods
Multi-Version Concurrency Control
Multi-Version Concurrency Control Algorithms
Multi-Version Database
Multi-Version Databases
Multi-version Serializability and Concurrency Control
Music Metadata
Music Retrieval
MVD
Naive Tables
Name Matching
Namelessness
Narrowed Extended XPath I
NAS
NAS Servers
NASD
Natural Human-Computer Interaction (NHCI)
Natural Interaction
Natural Language Generation (NLG)
Navigation System Interfaces
Near-Duplicate Video Retrieval
Nearest Neighbor Classification
Nearest Neighbor Query
Nearest Neighbor Query in Spatio-temporal Databases
Negative Dictionary
Nested Loop Join
Nested Transaction Models
.NET Remoting
Network Attached Secure Device
Network Attached Storage
Network Data Model
Network Database
Network Topology
Neural Networks
New Media Metadata
NEXI
NFS
NF-SS
N-Gram Models
NIAM
NN Classification
NN Query
NN Search
Node
Noise Addition
Non-Clustering Index
Non-Dense Index
Nonidentifiability
Nonlinear Magnification
Non-Metric Temporal Reasoning
Nonparametric Data Reduction Techniques
Non-Perturbative Masking
Non-Perturbative Masking Methods
Non-Pipelineable Operator
Nonsequenced Semantics
Nontemporal Semantics
Non-Uniform Distribution
Normal Form ORA-SS Schema Diagrams
Normal Forms and Normalization
Normalized Discounted Cumulated Gain (nDCG)
Normalizing ORA-SS Diagrams
Now in Temporal Databases
n-Tier Architecture
Null Values
Numeric Association Rules
Numerical Fact
Nymity
OASIS
Object Constraint Language
Object Data Models
Object Detection and Recognition
Object Flow Diagrams
Object Identification
Object Identification
Object Identifier
Object Identity
Object Labeling
Object Monitor
Object Query Language
Object Recognition
Object Reference
Object Relationship Attribute Data Model for
Semi-structured Data
Object Request Broker
Object-based Storage Device
Object-Role Modeling
OCL

ODB (Object Database)
ODBC
Office Automation
Oid
OKAPI Retrieval Function
OLAP
On-Disk Security
One-Copy-Serializability
One-Pass Algorithm
One-Way Hash Functions
Online Advertising
On-Line Analytical Processing
Online Handwriting
Online Recovery
Online Recovery in Parallel Database Systems
Ontological Engineering
Ontologies
Ontologies and Life Science Data Management
Ontology
Ontology Acquisition
Ontology Argumentation
Ontology Elicitation
Ontology Engineering
Ontology Learning
Ontology Negotiation
Ontology Query Languages
Ontology Visual Querying
On-Wire Security
OODB (Object-Oriented Database)
Open Database Connectivity
Open Nested Transaction Models
Open Nested Transactions
Operating Characteristic
Operator-Level Parallelism
Operator Scheduling
Operator Tree
Opinion Mining
Optical Storage
Optimistic Replication
Optimistic Replication and Resolution
Optimization and Tuning in Data Warehouses
Optimization of DAG-Structured Query Evaluation Plans
Optimization of Parallel Query Plans
OQL
ORA-SS Data Model
ORA-SS Schema Diagram
Orchestration
ORDB (Object-Relational Database)
Order Item
Order Statistics
Ordering
Orientation Relationships
Oriented Clustering
Origin
OR-Join
OR-Split
OSD
OSQL
Overlay Network
OWL: Web Ontology Language
P/FDM
P@n
P2P Database
Page Cache
Page Locking
Page Model
Page Representations
Paging in Web Search Engines
PAM (Partitioning Around Medoids)
Parallel and Distributed Data Warehouses
Parallel Axes
Parallel Coordinates
Parallel Coordinates Plot (PCP)
Parallel Coordinates System (PCS)
Parallel Data Placement
Parallel Database
Parallel Database Management
Parallel Distributed Processing
Parallel Hash Join, Parallel Merge Join, Parallel Nested Loops Join
Parallel Join Algorithms
Parallel Query Execution Algorithms
Parallel Query Optimization
Parallel Query Processing
Parallel SCSI
Parameterized Complexity of Queries
Parametric Data Reduction Techniques
Partial Replication
Partitioned Query Execution
Partitioning
Passage Retrieval
Path Functional Dependencies
Path Index
Path Query
Pattern Based Clustering
Pattern Discovery
Pattern-Growth Methods

PCA
PDMS
Pedigree
Peer Data Management
Peer Data Management System
Peer Database Management
Peer to Peer Network
Peer to Peer Overlay Networks: Structure, Routing and Maintenance
Peer-To-Peer Content Distribution
Peer-to-Peer Data Integration
Peer-to-peer Database
Peer-to-peer File Sharing
Peer-to-peer Network
Peer-to-peer Overlay
Peer-to-Peer Publish-Subscribe Systems
Peer-to-Peer Storage
Peer-to-Peer System
Peer-to-Peer Web Search
Performance Analysis of Transaction Processing Systems
Performance Benchmark
Performance Measures
Performance Metrics
Performance Monitoring Tools
Period-Stamped Temporal Models
Persistence
Persistent Applications
Persistent Archives
Personal Data
Personalized Interfaces
Personalized Search
Personalized Web
Personalized Web Search
Personally Identifiable Data
Perturbation Techniques
Perusal
Pessimistic Scheduler
Petri Nets
Photograph
Physical Clock
Physical Database Design for Relational Databases
Physical Layer Tuning
Physical Time
Physical Volume
Physical Window = Tuple-based Windows
Physician Order Entry
Pictorial Metadata
Picture
Picture Metadata
Piecewise-Constant Approximations
Pipeline
Pipelined and Independent Parallelism
Pipelining
PiT Copy
Pixed Oriented Visualiyation Techniques
Pixel Classification
Place Names
Place Transition Nets
Player
Plot
Plots for Qualitative Information
Point-based Temporal Models
Point-based Temporal Data
Point-in-Time Copy
Point-Stamped Temporal Models
Point-versus Period-based Semantics
Polyhedron
Polytransactions
Port Binding
Position Snapping
Positive Infinity
Positive Predictive Value
Positive Relational Algebra
Possible Answers
Postings File
Post-Randomization Method
PRAM
Precision
Precision and Recall
Precision at n
Precision-Oriented Effectiveness Measures
Predicate Calculus
Predicate Logic
Prediction of Event Occurrence
Prediction Regarding Future Events
Prefix Tree
Presenting Structured Text Retrieval Results
Preservation
Preserving Database Consistency
Preview
Primary Index
Primary Memory
Primitive Event
Principal Component Analysis
Principle of Locality
Privacy
Privacy Measures

Privacy Metrics
Privacy Policies and Preferences
Privacy Protection
Privacy-Enhancing Technologies
PETs
Privacy-Preserving Data Mining
Privacy-Preserving Spatial Queries
Probabilistic Analysis
Probabilistic Data
Probabilistic Databases
Probabilistic Model
Probabilistic Model of Indexing
Probabilistic Querying
Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model
Probabilistic Spatial Queries
Probabilistic Temporal Databases
Probability Ranking Principle
Probability Smoothing
Procedure Order
Procedure Request
Process Composition
Process Definition
Process Evolution
Process Life Cycle
Process Management
Process Mining
Process Optimization
Process Semantics
Process State Model
Process Structure of a DBMS
Processing Overlaps
Processing Structural Constraints
Processor Cache
Production-based Approach to Media Analysis
Projected Clustering
Projection
Projection Index
Propagation-based Structured Text Retrieval
Protein Sequence
Protein-Protein Interaction Networks
Provenance
Provenance Metadata
Provenance in Scientific Databases
Proximity
PRP
p-Sensitive k-Anonymity
Pseudonymity
Public-Key Encryption
Publish/Subscribe
Publish/Subscribe over Streams
Punctuations
Push Transactions
Push/Pull Delivery
QE, Query Enhancement
QoS-Based Web Services Composition
Quadtree Variations
Quadtrees (and Family)
Qualitative Relations between Time Intervals
Qualitative Temporal Constraints between Time Intervals
Qualitative Temporal Reasoning
Quality and Trust of Information Content and Credentialing
Quality Assessment
Quality of Data Warehouses
Quantiles on Streams
Quantitative Association Rules
QUEL
Query Answering in Analytical Domains
Query Assistance
Query by Example
Query by Humming
Query Compilation
Query Compilation and Execution
Query Containment
Query Engine
Query Evaluation
Query Evaluation Plan
Query Evaluation Techniques for Multidimensional Data
Query Execution Engine
Query Execution in Star/ Snowflake Schemas
Query Execution Plan
Query Expansion
Query Expansion for Information Retrieval
Query Expansion Models
Query Language
Query Languages and Evaluation Techniques for Biological Sequence Data
Query Languages for the Life Sciences
Query Load Balancing in Parallel Database Systems
Query Mapping
Query Optimization
Query Optimization (in Relational Databases)
Query Optimization for Multidimensional Systems
Query Optimization in Distributed Database Systems
Query Optimization in Sensor Networks

Query Parallelism
Query Plan
Query Planning and Execution
Query Point Movement Techniques for Content-Based Image Retrieval
Query Processing
Query Processing (in Relational Databases)
Query Processing and Optimization in Object Relational Databases
Query Processing in Data Warehouses
Query Processing in Deductive Databases
Query Processor
Query Reformulation
Query Rewriting
Query Rewriting Using Views
Query Suggestion
Query Transformations
Query Translation
Query Tree
Query Tuning
Querying DNA Sequences
Querying Protein Sequences
Querying Semi-Structured Data
Queuing Analysis
Queuing Mechanism
Queuing Systems
Quorum Systems
RAID
Random Access Memory (RAM)
Randomization Methods to Ensure Data Privacy
Range Partitioning
Range Query
Range Search
Range Selection
Rank Swapping
Ranked Multimedia Retrieval
Ranked XML Processing
Ranking
Raster Data Management
Raster Data Management and Multi-Dimensional Arrays
Raster Databases
RBAC
RBAC Standard
RDF
Reactive Rules
Read/Write Model
Real and Synthetic Test Datasets
Real-Time Transaction Processing
Real-World Time
Reasoning with Qualitative Temporal Constraints
Recall
Receiver Operating Characteristic
Recodings
Reconciliation-based Data Replication
Record Extraction
Record Linkage
Record Matching
Records Management
Recovery Guarantees
Recovery in Distributed Commit Protocols
Recovery in Distributed Database Systems
Recovery in Replicated Database Systems
Recovery Manager
Recursive Query Evaluation
Recursive View Maintenance
Redo
Redundant Arrays of Independent Disks
Reference
Reference Collections
Reference Knowledge
Reference Reconciliation
Refinement
Region Algebra
Region Segmentation
Registration Time
Regulatory Compliance in Data Management
Re-identification
Re-Identification Risk
Relational Algebra
Relational Algebra for XML
Relational Calculus
Relational Database
Relational Integrity Constraints
Relational Model
Relational Query Processor
Relation-Completeness
Relations with Marked Nulls
Relationship of Reliance
Relationships in Structured Text Retrieval
Relative Operating Characteristic
Relative Time
Relevance
Relevance Evaluation of IR Systems
Relevance Feedback
Relevance Feedback for Content-Based Information Retrieval
Relevance Feedback for Text Retrieval

Relevance Propagation
Remote Method Invocation
Removing Overlap
Rendezvous
Replica and Concurrency Control
Replica Consistency
Replica Control
Replica Freshness
Replicated Database Concurrency Control
Replication
Replication Based on Group Communication
Replication for High Availability
Replication for Scalability
Replication in Multi-Tier Architectures
Report Writing
Representation
Reputation
Reputation and Trust
Request Broker
Residuated Lattice
Resource Allocation Problems in Spatial Databases
Resource Description Framework
Resource Description Framework (RDF) Schema (RDFS)
Resource Identifier
Resource Scheduling
Restart Processing
Restricted Data
Result Display
Result Overview
Result Ranking
Retrieval Models
Retrieval Models for Text Databases
Retrospective Event Processing
Reverse Nearest Neighbor Query
Reverse Nearest Neighbor Search
RF
Rich Media
Right-Time Data Warehousing
Risk-Utility Tradeoff
Rewriting Queries using Views
RMI
RNN Query
Road Network Databases
Road Networks
Road Vector Data
Robot
ROC
Rocchio's Formula
Role Based Access Control
Role Based Security
Rollback
Rollback Operator
Rotation
Rotation Estimation
Rough Computing
Rough Set Theory (RST)
Rough Set Theory, Granular Computing on Partition
Rounding
Row-Level Locking
Row-Versioning
R-Precision
RSJ Model
Rtree
R-Tree (and Family)
Rule Bases
Rule-based Classification
S@n
Safety and Domain Independence
Sagas
Samba
Sampling
Sampling Techniques for Statistical Databases
SAN
SAN File System
SARBAC
SAS
SATA
SBQL
SCA
Scalable Classification Tree Construction
Scalable Database Replication
Scalable Decision Support Systems High Performance Data Warehousing
Scalable Decision Tree Construction
Scalable Replication
Scalable Top-Down Decision Tree Construction
Scale Out
Scale-Out Databases
Scale-Up Databases
Scaling
Scanning
Scene Change Detection
Scheduler
Scheduling
Scheduling Policies
Scheduling Strategies for Data Stream Processing
Schema Evolution

Schema Evolution in Process Management Systems
Schema Evolution in Workflow Management Systems
Schema Mapping
Schema Mapping Composition
Schema Matching
Schema Normalization
Schema Tuning
Schema Versioning
Scientific Databases
Scientific Knowledge Bases
Scientific Medicine
Scientific Query Languages
Scientific Visualization
Scientific Workflows
Score Propagation
Screen Scraper
Screen Scraping
Screen Wrapper
SCSI Target
SDI, Selective Dissemination of Information
Search Advertising
Search Engine Caching and Prefetching
SDC Score
Search Engine Metrics
Search Engine Query Result Caching
Search Ranking
Searching Compressed XML
Searching Digital Libraries
Second Normal Form (2NF)
Secondary Index
Secret-Key Encryption
Secure Data Outsourcing
Secure Database Design
Secure Database Development
Secure Database Systems
Secure Datawarehouses
Secure DBMS Development
Secure Hardware
Secure Multiparty Computation Methods
Secure Third-Party Data Management
Secure Transaction Processing
Security Services
Segmentation
Selection
Selective XML Dissemination
Selectivity Estimation
Selectivity for Predictive Spatio-Temporal Queries
Self-Maintenance of Views
Self-Management Technology in Databases
Self-Managing Database Systems
Self-Tuning Database Systems
Semantic Analysis of Video
Semantic Data Integration for Life Science Entities
Semantic Data Model
Semantic Image Retrieval
Semantic Inference in Audio
Semantic Mapping Composition
Semantic Matching
Semantic Modeling and Knowledge Representation for Multimedia Data
Semantic Modeling for Geographic Information Systems
Semantic Overlay Networks
Semantic Overlays
Semantic Web
Semantic Web Query Languages
Semantic Web Services
Semantic-based Retrieval
Semantic Atomicity
Semantics-based Concurrency Control
Semijoin
Semijoin Filter
Semi-Structured Data
Semi-Structured Query Languages
Semijoin Program
Semijoin Reducer
Semi-Streaming Model
Semi-Structured Data
Semi-Structured Data Model
Semi-Structured Database
Semi-Structured Database Design
Semi-Structured Query Languages
Semi-Structured Text Retrieval
Semi-Supervised Classification
Semi-Supervised Clustering
Semi-Supervised Learning
Sense and Respond Systems
Sensitivity
Sensor Network Systems
Sensor Networks
Sensornet
Sentiment Analysis
SEQUEL
Sequence Data Mining
Sequenced Semantics
Sequential Patterns
Serializability
Service Bus

Service Buses
Service Choreography
Service Component Architecture (SCA)
Service Composition
Service Item
Service Orchestration
Service Order
Service Oriented Architecture
Service Request
Session
Set Abstraction
Set-Difference
Shape Descriptors
Shared-Disk File System
Shared-Disk Architecture
Shared-Disk Databases
Shared-Everything
Shared-Everything Databases
Shared Health Record
Shared-Memory Architecture
Shared-Nothing Architecture
Shared-Nothing Databases
Shot Boundary Detection
Shot Segmentation
Shotcut Detection
SI
Side-Effect-Free View Updates
Sight
Signal Transduction Networks
Signature Files
Signatures
Similarity and Ranking Operations
Similarity in Video
Similarity Measure
Similarity-based Data Partitioning
Simplicial Complex
Simulated Data
Single Instancing
Single Instruction Multiple Data (SIMD) Parallelism
Singular Value Decomposition
Sketch
SMI-S
Snapshot
Snapshot Data
Snapshot Equivalence
Snapshot Isolation
SNIA
Snippet
Snowflake Join Schema
Snowflake Schema
SOA
SOA Replication
SOAP
Social Applications
Social Networks
Software Transactional Memory
SONs
Sort-Merge Join
Source
Space-Filling Curves
Space Partitioning
Space Segmentation
Space-Filling Curve
Space-Filling Curves for Query Processing
Space-Span (in Part)
Spamdexing
Span
Sparse Index
Spatial Access Methods
Spatial Analysis
Spatial and Spatio-Temporal Data Models and Languages
Spatial Anonymity
Spatial Autocorrelation
Spatial Data
Spatial Data Analysis
Spatial Data Mining
Spatial Data Types
Spatial Graph Databases
Spatial Indexing Techniques
Spatial Information System
Spatial Join
Spatial k-Anonymity
Spatial Network Databases
Spatial Operations and Map Operations
Spatial Outliers
Spatial Referencing
Spatial Statistics
Spatio-Temporal Approximation
Spatio-Temporal Benchmarking
Spatio-Temporal Data Generator
Spatio-Temporal Data Mining
Spatio-Temporal Data Reduction
Spatio-Temporal Data Types
Spatio-Temporal Data Warehouses
Spatio-Temporal Databases
Spatiotemporal Estimation
Spatio-Temporal Graphs

Spatiotemporal Interpolation Algorithms
Spatio-Temporal OLAP
Spatio-Temporal Online Analytical Processing
Spatio-Temporal Representation
Spatio-Temporal Selectivity Estimation
Spatio-Temporal Stream Processing
Spatio-Temporal Trajectories
SPC Query
SPCU-Algebra
Specialization
Specialization and Generalization
Specificity
Spectral Clustering
Spider
Spidering
SPJRU-Algebra
Split
Split Transactions
SQL
SQL Isolation Levels
SQL-Based Temporal Query Languages
SRM
Stability-based Validation of Clustering
Stable Distribution
Stack-based Query Language
Staged Database Systems
Staged DBMS
Standard Effectiveness Measures
Standing Query
Star Index
Star Join Schema
Star Schema
Star Schema Modeling
State Query
State-based Publish/Subscribe
Statistical Correctness
Statistical Data Management
Statistical Database
Statistical Decision Techniques
Statistical Disclosure Control (SDC)
Statistical Disclosure Limitation (SDL)
Statistical Disclosure Limitation For Data Access
Steganography
Stemming
Step
Stewardship
Stop-&-go Operator
Stoplists
Stopwords
Storage Access Models
Storage Area Network
Storage Array
Storage Broker
Storage Consolidation
Storage Controllers
Storage Devices
Storage Grid
Storage Layer
Storage Management
Storage Management Initiative-Specification
Storage Manager
Storage Network Architectures
Storage Networking Industry Association
Storage of Large Scale Multidimensional Data
Storage Power Management
Storage Protection
Storage Protocols
Storage Resource Management
Storage Security
Storage Servers
Storage Systems
Storage Virtualization
Stored Procedure
S-Transactions
Stream Data Analysis
Stream Mining
Stream Models
Stream Processing
Stream Query Processing
Stream Sampling
Stream Similarity Mining
Streaming Algorithm
Streaming Applications
Streaming Database Systems
Stream-Oriented Applications
Stream-Oriented Query Languages and Operators
Strong Consistency Models for Replicated Data
Strong Coupling
Strong Memory Consistency
Structural Index
Structural Summary
Structure Indexing
Structure of Truth Values
Structure Weight
Structured Data in Peer-to-Peer Systems
Structured Document Retrieval
Structured Query Language
Structured Text Retrieval

Structured Text Retrieval Models
Structured Text Retrieval Tasks
Subject Spaces
Subject-based Publish/Subscribe
Subspace Clustering Techniques
Subspace Selection
Subsumed by Windows Communication Framework
Success at n
Succinct Constraints
Suffix Stripping
Suffix Tree
Suffixing
Summarizability
Summarization
Summarization Correctness
Summary
Supervised Learning
Support Vector Machine
Supporting Transaction Time Databases
Surfing
SVD Transformation
SVM
Switch
Symbol Graph
Symbol Plot
Symbolic Graphic
Symbolic Representation
Symmetric Encryption
Synchronization Component
Synchronization Join
Synchronizing Distributed Transactions
Synchronous Join
Synchronous Pipelines
Synopsis
Synopsis Structure
Synthetic Data
Synthetic Image
Synthetic Microdata
System Catalog
System R (R*) Optimizer
System Recovery
Table
Table Design
Table Normalization
Tabular Data
Tamper-Proof Hardware
Tape Libraries
Tapes
Task
Taxonomies
Taxonomy: Biomedical Health Informatics
Telic Distinction in Temporal Databases
Telos
Temporal Access Control
Temporal Aggregation
Temporal Algebras
Temporal Assignment
Temporal Association Mining
Temporal Coalescing
Temporal Compatibility
Temporal Conceptual Models
Temporal Constraints
Temporal Data Mining
Temporal Data Models
Temporal Data Warehousing
Temporal Database
Temporal Dependencies
Temporal Domain
Temporal Element
Temporal Evolution
Temporal Expression
Temporal Generalization
Temporal Granularity
Temporal Homogeneity
Temporal Indeterminacy
Temporal Information Retrieval
Temporal Integrity Constraints
Temporal Joins
Temporal Layer
Temporal Logic
Temporal Logic in Database Query Languages
Temporal Logical Models
Temporal Middleware
Temporal Object-Oriented Databases
Temporal Periodicity
Temporal Projection
Temporal Query Languages
Temporal Query Processing
Temporal Relation
Temporal Relational Calculus
Temporal Restriction
Temporal Semi-Structured Data
Temporal Specialization
Temporal Strata
Temporal Structure
Temporal Type
Temporal Upward Compatibility
Temporal Vacuuming

Temporal Value
Temporal Visual Interfaces
Temporal Visual Languages
Temporal Visual Queries
Temporal XML
Temporally Indeterminate Databases
Temporally Uncertain Databases
Temporally Weak
Term Expansion
Term Expansion Models
Term Frequency by Inverse Document Frequency
Term Frequency Normalization
Term Processing
Term Proximity
Term Statistics for Structured Text Retrieval
Term Weighting
Term-Document Matching Function
Terminologic Languages
Terminological Database
Test Collection
Text Analytics
Text Categorization
Text Classification
Text Clustering
Text Compression
Text Data Mining
Text Databases
Text Extraction
Text Generation
Text Index Compression
Text Indexing and Retrieval
Text Indexing Techniques
Text Mining
Text Mining of Biological Resources
Text Representation
Text Retrieval
Text Segmentation
Text Semantic Representation
Text Streaming Model
Text Summarization
Text Visualization
Text/Document Summarization
Text-based Image Retrieval
TF*IDF
tgd
Thematic Map
Theme Algebra
Thesauri Business Catalogues
Thiessen Polygons
Third Normal Form
Thread Lifecycle
Three-Dimensional GIS and Geological Applications
Three-Dimensional Similarity Search
Three-Phase Commit
Thresholding
Tight Coupling
Time Aggregated Graphs
Time and Information Retrieval
Time Dependent Geometry
Time Distance
Time Domain
Time Granularity
Time in Philosophical Logic
Time Instant
Time Interval
Time Period
Time Period Set
Time Point
Time Quantum
Time Sequence
Time Sequence Query
Time Sequence Search
Time Series
Time Series Data Mining
Time Series Database Querying
Time Series Query
Time Series Search
Time Span
Time Unit
Time-based Access Control
Time-based Window
Time-Constrained Transaction Management
Time-Dependent Graphs
Time-Dependent Networks
Time-Line Clock
Time-Oriented Database
Time-Segment Clock
Timeslice Operator
TIN
Tiny Aggregation (TAG)
TinyDB
TinySQL
t-Norm
Topic Detection and Tracking
Topic Hierarchies
Topic Maps
Topical-Hierarchical Relevance
Topic-based Publish/Subscribe

Topic-Directed Web Crawling
Top-k Queries in P2P Systems
Top-K Selection Queries on Multimedia Datasets
Top-k XML Query Processing
Topological Data Models
Topological Data Structure
Topological Fabric
Topological Relationships
Topology
Toponyms
Tour
TP
TP Monitor
TPQ
Traditional Concurrency Control for Replicated Databases
Traditional Data Replication
Traditional Replica and Concurrency Control Strategies
Trajectory
Trajectory Databases
Trajectory Indexing
Transaction
Transaction Chopping
Transaction Commit Time
Transaction Execution
Transaction Management
Transaction Management in Distributed Database Systems
Transaction Manager
Transaction Model
Transaction Models – the Read/Write Approach
Transaction Processing
Transaction Scheduling
Transaction Service
Transaction Time
Transactional Business Processes
Transactional Consistency in a Replicated Database
Transactional Middleware
Transactional Processes
Transactional Workflows
Transaction-Time Access Methods
Transaction-Time Algebras
Transaction-Time Data Model
Transaction-Time Indexing
Transcriptional Networks
Transformation
Transformation Engines
Translation Lookaside Buffer (TLB)
Translingual Information Retrieval
Tree Drawing
Tree Pattern Queries
Tree-based Indexing
Treemaps
Tree-Structured Classifier
Triangular Norms
Triangulated Irregular Network
Triangulated Terrains
Trie
Triggers
True Answer (Maybe Answer)
Trust and Reputation in Peer-to-Peer Systems
Trust in Blogosphere
Trusted Database Systems
Trusted Hardware
TSQL2
Tug-of-War Sketch
Tuning Concurrency Control
Tuning the Application Interface
Tuple Relational Calculus
Tuple-Generating Dependencies
Twigs
Two-Dimensional Shape Retrieval
Two-Phase Commit
Two-Phase Commit Protocol
Two-Phase Locking
Two-Poisson model
Two-Sorted First-Order Logic
Type Theory
Type-based Publish/Subscribe
UML
Uncertain Databases
Uncertain Information
Uncertainty in Events
Uncertainty Management in Scientific Database Systems
Undo
Unicode
Unified Modeling Language
Uniform Resource Identifier
Union
Uniqueness Constraint
Unnoticability
Unobservability
Unsupervised Learning
Unsupervised Learning on Document Datasets
Until Changed
Update Propagation in Peer-to-Peer Systems

Updates and Transactions in Peer-to-Peer Systems
Updates through Views
URI
Usability
User Centered Design
User Classifications
User-Centred Design
User-Defined Time
User-Level Parallelism
Using Efficient Database Technology (DB) for Effective Information Retrieval (IR) of Semi-Structured Text
Utility Computing
UUID
Valid Time
Validity (Satisfiability)
Valid-Time Access Methods
Valid-Time Algebras
Valid-Time and Transaction-Time Relation
Valid-Time Data Model
Valid-Time Indexing
Value Equivalence
Variable Time Span
VDM
Vector-Space Model
Vertical Fragmentation
Vertically Partitioned Data
Video
Video Abstraction
Video Analysis
Video Annotation
Video Chaptering
Video Compression
Video Content Analysis
Video Content Description
Video Content Modeling
Video Content Processing
Video Content Structure
Video Data Modeling
Video Format
Video Indexing
Video Metadata
Video Partitioning
Video Querying
Video Representation
Video Retrieval
Video Scene
Video Scene and Event Detection
Video Search
Video Segmentation
Video Sequence Indexing
Video Shot Detection
Video Shot-Cut Detection
Video Skimming
Video Structure Analysis
Video Structuring
Video Summarization
View Adaptation
View Definition
View Maintenance
View Maintenance Aspects
View Expression
View Update
View-based Data Integration
Views
Virtual Disk Manager
Virtual Health Record
Virtual Partitioning
Vision
Visual Analysis
Visual Analytics
Visual Association Rules
Visual Classification
Visual Clustering
Visual Content Analysis
Visual Data Analysis
Visual Data Exploration
Visual Data Mining
Visual Discovery
Visual Displays of Nonnumerical Data
Visual Displays of Numerical Data
Visual Formalisms
Visual Interaction
Visual Interaction Design
Visual Interfaces
Visual Interfaces for Geographic Data
Visual Metaphor
Visual Mining
Visual Multidimensional Analysis
Visual On-Line Analytical Processing (OLAP)
Visual Perception
Visual Query Language
Visual Query System
Visual Representation
Visual Similarity
Visual Web Data Extraction
Visual Web Information Extraction
Visualization for Information Retrieval
Visualization Pipeline

Visualization Reference Model
Visualizing Categorical Data
Visualizing Clustering Results
Visualizing Hierarchical Data
Visualizing Network Data
Visualizing Quantitative Data
Visualizing Spatial Data
Visualizing Trees
Volume
Volume Set Manager
Voronoi Decomposition
Voronoi Diagrams
Voronoi Tessellation
VP
VSM
W3C
W3C XML Path Language
W3C XML Query Language
W3C XML Schema
WAN Data Replication
Watermarking
Wavelets on Streams
Weak Consistency Models for Replicated Data
Weak Coupling
Weak Equivalence
Weak Memory Consistency
Web 2.0 Applications
Web 2.0/3.0
Web Advertising
Web Application Server
Web Characteristics and Evolution
Web Content Extraction
Web Content Mining
Web Crawler
Web Crawler Architecture
Web Data Extraction
Web Data Extraction System
Web Data Mining
Web Directories
Web ETL
Web Harvesting
Web Indexing
Web Information Extraction
Web Information Extraction System
Web Information Integration and Schema Matching
WEB Information Retrieval Models
Web Macros
Web Mining
Web Mashups
Web Ontology Language
Web Page Quality Metrics
Web QA
Web Query Languages
Web Question Answering
Web Resource Discovery
Web Scraper
Web Scraping
Web Search and Crawling
Web Search Engines
Web Search Query Rewriting
Web Search Relevance Feedback
Web Search Relevance Ranking
Web Search Result Caching and Prefetching
Web Search Result De-duplication and Clustering
Web Services
Web Services and the Semantic Web for Life Science Data
Web Services Business Process Execution Language
Web Site Wrappers
Web Spam Detection
Web Structure Mining
Web Transactions
Web Usage Mining
Web Views
Web Widget
What-If Analysis
While Loop
Wide-Area Data Replication
Wide-Area Storage Systems
WIMP Interfaces
Window-based Query Processing
Windows
Wireless Sensor Networks
Within-Element Term Frequency
Word Conflation
Word of Mouth
Work Element
Work Performer
Workflow
Workflow Constructs
Workflow Control Data
Workflow Enactment Service State Data
Workflow Engine State Data
Workflow Evolution
Workflow Join
Workflow Lifecycle
Workflow Loop
Workflow Management

Workflow Management and Workflow Management System
Workflow Management Coalition
Workflow Meta-Model
Workflow Mining
Workflow Model
Workflow Model Analysis
Workflow Modeling
Workflow on Grid
Workflow Participant
Workflow Patterns
Workflow Scheduler
Workflow Schema
Workflow Transactions
Workflow/Process Instance Changes
World Wide Web Consortium
WORM
Wrapper
Wrapper Adaptability
Wrapper Induction
Wrapper Generation
Wrapper Generator
Wrapper Generator GUIs
Wrapper Maintenance
Wrapper Repair
Wrapper Robustness
Wrapper Stability
Wrapper Verification and Reinduction
Write Once Read Many
Write Once Read Mostly
Write One Read Multiple
WS-BPEL
WS-Discovery
XA Standard
XMI
XML
XML (Almost)
XML 1.0
XML Access Control
XML Algebra
XML Application Development
XML Attribute
XML Attribute Node
XML Benchmarks
XML Cardinality Estimation
XML Compression
XML Data Dependencies
XML Data Integration
XML Database
XML Database Design
XML Database System
XML Document
XML Element
XML Enterprise Information Integration
XML Export
XML Filtering
XML Indexing
XML Information Integration
XML Information Retrieval
XML Integrity Constraints
XML Message Brokering
XML Metadata Interchange
XML Parsing, SAX/DOM
XML Persistence
XML Process Definition Language
XML Programming
XML Publish/Subscribe
XML Publishing
XML Retrieval
XML Schema
XML Schemas
XML Selectivity Estimation
XML Storage
XML Stream Processing
XML Tree Pattern, XML Twig Query
XML Tuple Algebra
XML Typechecking
XML Types
XML Updates
XML Views
XPath/XQuery
XPDL
XQFT
XQuery 1.0 and XPath 2.0 Full-Text
XQuery Compiler
XQuery Full-Text
XQuery Interpreter
XQuery Processors
XSL Formatting Objects
XSL/XSLT
XSL-FO
Zero-One Laws
ZF-Expression
Zoning
Zoomable User Interface (ZUI)
Zooming Techniques

Subject Index

A

Abstraction, 6–7
Abstract types, 1554
Acceptance-rejection sampling, 2467
Access control, 2447–2452
– mechanism, 7–10
– policy, 11
Access control lists (ACLs), 3573
Access latency, 72, 73, 76
Access methods, 1448, 1449, 2224, 2681
Access path, 19
Access rights, 864, 865
Accuracy, 2229
ACID properties, 926–927, 1644, 1896–1898, 1978, 1980, 2761, 2762, 3559
– atomicity, 19–20
– durability, 20–21
– isolation, 20
– transaction, 3150
Acronym for redundant arrays of independent disks, 2823
ACTA, 1091–1092
Action, 959
Active-active mode, 1843
Active and real-time data warehousing, 21–26
Active conceptual modeling, 2228
Active database, 28–33
– coupling modes, 33–35
– execution model, 35–36
– knowledge model, 36
– management system architecture, 28–33
– rulebase, 37
Active database (management) system (aDBS/aDBMS), 28–33
– active database (aDB), 27
– database transitions, 27
– external signals, 27
– periodically polling the database, 27
– time events, 27
Active Data Warehousing, 21–26
Active disk, 37–38
Active learning, 3563
Active replicas, 1111–1114
Active-standby mode, 1843
Active storage, 37–38, 1552
Active XML (AXML) documents, 38–40
Activity, 41
– block, 1661
– diagrams, 41–45
Actor, 46
AdaBoost, 261, 262
Adaptability techniques, 51
Adapter, 696–699, 1978,
Additive noise, 1911
Adaptive query processing, 50–52
Adaptive stream processing, 52–56, 1636
Adaptive user interfaces, 46–47
Additionally marked information, 85
Ad-hoc workflows, 3548
Administration model for RBAC, 57
Administration policies, 12–14
Administration wizards, 58
Administrative RBAC family of models, 57
Administrative workflows, 3548
ADO.NET, 449
Advanced transaction models, 1091–1092
Aesthetic principle, 430
Affinity matrix, 2749, 2751
Affix removal, 2790, 2793
Agent, 46
Agglomerative clustering, 933
Aggregate-storing indices, 1456–1457
Aggregate queries, 59–62
– peer-to-peer, 111–113
Aggregation, 59–70, 1028, 1029
– distance, 1890
– group, 2924–2926
– function, 2924–2926, 2928
– operators over time windows, 2996
– views, 1421, 1423
Air index, 71–77
Air indexes for spatial databases, 71–77
Algebra, 258, 259, 2360–2363
Algebraic and logical models, 1524, 1527
Algebras for tuple timestamping, 2930–2931
Algorithms, 378, 380–382
Alignment with business strategy, 1516
All-distances sketches, 759, 760
Allen's algebra, 2225–2227
Allen's relations, 78–79
Alternative composition, 420
Amercia II Project (MOA2), 1636
AMOSQL, 79–80
AMQP, 2210
AMS sketch, 80–83
Analysis of visual information, 3317
Analytical interest, 1500
Analytical validity, 1500
Analytic hierarchy process, 1211
Analytics, 3366, 3368
Anchor text, 84
– document surrogate, 84
– home-page finding, 84
– incoming hyperlinks, 84
AND-Join, 2006, 2752, 3544
AND-split, 2752
Animation, 950
ANN architecture, 1906
Annotated data, 85
Annotation, 85, 1580, 3072, 3076, 3078
Annotation-based image retrieval, 85–88
Anomaly detection, 1569
– on streams, 88–90
Anonymity, 90–91, 2138, 2142–2146
Anonymity set, 91
Anonymizer, 2686–2688, 2690
ANOVA (analysis of variance), 238
ANSI/ICITS RBAC standard, 2448
ANSI RBAC, 91–92
ANSI standard, 2755, 2756, 2760
Answering queries using views, 92–97, 3324
Anti-monotone
– constraints, 98
Appearance-based object model, 1937
Application level protocol standard, 2666
Application-level tuning, 105–106
Application(s), 107–110
– benchmark, 99
– including secure databases, 2531
– multimedia, 1323–1326
– persistence, 100, 101, 103
– programming interfaces (APIs), 1557
– recovery, 100–103
– server, 104, 1578, 1579
Applying supervised learning to web wrapper generation, 3561
Approximate itemset, 120–123
Approximate mining, 137, 138
Approximate queries(ing), 1764–1768
– peer-to-peer, 111–113
Approximate query answer, 884, 889
Approximate query processing, 113–119
Approximate reasoning, 119–120
ε-Approximation, 1240, 1241, 1244
Approximation
– frequent itemset, 120–123
– randomization, 884–885
Apriori property, 98, 124–126
ARBAC97, 57
Architectural model, 2638
Architecture, 3012
Architecture-conscious database system, 127–131
Archive recovery, 517, 518, 521–522

ARC/INFO, 3132, 3136
Area under ROC curve (AUC), 2351, 2352
Argus, 1898
ARIES protocol, 1784–1788
Aristotle's ontology, 1964
Arithmetic coding, 3047
Armstrong axioms, 135, 1200
Array databases, 2332, 2333, 2336–2338
AR sampling, 2467
Artificial neural network (ANN), 1906, 1907
Assessment activity, 608–610
Association rule mining, 126, 136–139, 1184–1186, 3346, 3348
Association rule(s), 140–141, 365–368, 574, 2051, 3346–3351
Associative similarity, 2648–2649, 2651
Asynchronous JavaScirpt and XML (AJAX), 77–78
ATA, 2825–2826
Atelic data, 142–143
Atomic clocks, 2108
Atomic event, 143
Atomicity, 143–146, 2588–2590, 2627, 2630
Atomicity of actions, 143–144
Attribute disclosure, 848
Attribution evolution, 2480
Audio
– classification, 148–154, 156
– compression, 3300–3301
– content analysis, 154–156
– effect detection, 149–154
– elements, 167–171
– feature, 160–166
– indexing, 149
– metadata, 157–160
– parsing, 167–171
– recognition, 149
– representation, 156
– segmentation, 156, 167–171
Auditing and forensic analysis, 172–176
Auditory scenes, 167–169, 171
Auditory sence segmentation, 167
Augmentation, 67–69
Aurora/Borealis project, 1633, 1635
Authentication, 176–180
Authorization, 7–9
– revocation, 12
Automated page wrappers, 1191
Automatic detection of auditory scenes, 167
Automatic form generation, 1164–1165
Automatic image annotation, 180–187
Automatic query expansion, 2258, 2259
Autonomous database replication, 188
Autonomous replication, 188–192
Autonomous systems, 47–50
Availability, 729–731
Average precision, 192–193, 1256
– histogram, 194
– value for a given topic, 1691
Average R-precision, 195
Axioms, 135, 1865
AXML languages, 38

B

Bag, 201–206
Bag relational algebra, 202, 204
Bag semantics, 201–206
Backup and restore, 200–201
Backwards induction, 546
Bagging, 206–210, 988–990
Bag-of-visual words representation, 183
Bag of Words (BOW) model, 3069
– of datalog queries, 205
Bag-set semantics, 201, 202, 204, 205
Balanced pipeline, 2116
Balanced trees, 1308, 1309
Banking, 290, 292
Base tables, 3244–3246
Basic database recovery methods, 1785
Basic focused web crawl, 1150, 1152
Basic retrieval performance measure, 2348
Bayes classifiers, 210–214, 341
Bayesian models, 1524, 1526–1527
Bayes theorem, 211
BCNF, 264–265
Behavior, 3017–3018
Best Match (BM), 255
Best matches only, 1371
Bi-clustering, 2873
Bigram, 1910
Binary feature vector, 2157
Binary Independence Retrieval (BIR), 2156–2160
Binary large objects (BLOBs), 1818
Binary polynomials, 328
Binary semantic model, 438
Bioinformatics, 1401–1403
Biological metadata management, 215–219
Biological molecules, 220
Biological networks, 220
Biological sequence, 223–224
Biomedical data
– annotation, 224–228
– types, 229–232
Biomedical image databases, 229
Biomedical images, 1358–1360
Biomedical informatics, 1401–1404
Biomedical literature, 233–235
Biostatistics, 236–238
Biostatistics and data analysis, 236–238
BIR Model, 2156–2160
Bisimilarity, 2861
Bisimulation, 2601, 2604
Bitemporal chronon, 243
Bitemporal query language, 243–244
Bitemporal relation, 2094
Bitmap-based indexing, 248–251
Bitmap-based index structures, 248–251
Bitmap compression, 250–251
Bitmap index(ices), 244–248, 1454–1457
Bitmapped images, 1374–1376
Bitmap table, 249
Bitslice indices, 1454, 1456
Bitstrings, 2643, 2645
Blanking, 1913
BLAST, 224
Blind feedback (BF), 2382
Blinding, 252
Blind relevance feedback, 2447
Blind signatures, 251–252
– generation, 252
Block ciphers, 572–573
Block-level I/O services, 1574
Block nested loop join, 1895
Bloom filter(s), 252–254, 2865
Bloomjoin, 253
BM25, 255–257, 1529, 1530
Boolean algebra, 2418, 2419
Boolean expression, 2967
Boosting, 259–263, 988–990
Boostrap, 263–264
Bottom granularity, 2971
Boyce-Codd Normal Form (BCNF), 264–265
BP-complete, 265
BP-Completeness, 265–266, 407Brand advertising, 3457, 3458
Breadth-first search algorithms, 124–126
Broker, 2415–2418
Broker architectures, 500, 1899
Browsing, 267–268
– in digital libraries, 268–273
– and searching, 1104
Brushing, 1623–1626
B-tree, 197–200
B+-tree index, 197–200
B-tree locking, 273–277
B+-trees, 197–200, 242, 1300–1302, 1436–1437, 2681, 3172, 3173
Bucket algorithm and its variants, 2440
Buffer management, 277–281
Buffer pool, 282–287
Buffer pool data structure, 286
Buffer replacement, 485–487
Buffer replacement algorithms, 1800–1802
Build phase, 1288
Burrows-Wheeler Transform, 1443, 1444, 3048
Burst detection, 89
Business activity management (BAM), 413

Business agenda, 1515–1516, 1518
Business intelligence, 287–288, 561, 563, 3525, 3528
– data warehouse, 3472
– systems, 287
Business logic, 998, 999
Business performance management, 287–288
Business process, 289–293, 3545, 3546, 3548, 3549, 3552–3557
– management, 1068–1071, 2170
– management life-cycle, 291
– modeling, 293–294
– reengineering, 295–299
Business process execution language (BPEL), 288–289, 3601
Business process modeling notation (BPMN), 3601
Business-to-Business, 292
Business to business (B2B) integration, 991
Business workflows, 2507
By-product, 1515

C

Cache
– line, 2195, 2196
– management, 744–747
Cache-conscious query processing
– cache, 301–303
– cache miss, 301, 302
– locality, 301–303
– memory latency, 302, 303
Caching, 1738–1742
Caching in search engines, 3501–3504
Calculus expression, 420–421
Calendar, 305, 2969, 2971–2973
– algebra, 2971, 2972
– cyclic time, 304
– date(s), 304
Calendric system, 305
Call level interface (CLI), 2755
Camelot, 1898
Camera break (or video edit), 3316
Camera-takes, 3308
Candidate key, 1587, 1588
Canonical sensor network platform, 1538
Cartesian product join, 309
Cartography, 3383, 3384, 3386
Catalog, 570
Cataloging in digital libraries, 309–314
– classification, 310–311
Categorical data, 836, 3416–3417
Categorical microaggregation, 1736
Categorical variables, 1660
Causality relations, 1035, 1044
Cayuga, 1029–1032
CDP implementations, 478
Cell suppression, 1473
Cellular phones, 1751–1753
Centralized deadlock detection, 900, 902
Centroid, 699
Certain answers, 1407–1409, 1875, 1876
Certain (and possible) answers, 315–317
– conditional tables, 316
– incomplete information, 317
– naive tables, 317
– null values, 317
Chandra and Harel completeness, 406–411
Change detection on Streams, 317–321
Change point analysis, 317
Change-table technique, 1421–1424
Changing data needs, 1516–1517
Channel-based publish/subscribe model
– pull style, 322
– push style, 322
CHAP, 2828
Chart, 323
Chase, 1563
Checksum, 327–328
– and cyclic redundancy check mechanism, 327–328
Chip multiprocessor (CMP), 127–131
Cholesky decomposition.
Choreography, 2004, 2005Chronicle event context, 419
Chronon
– period, 329
– time interval, 329
CIFS, 2824, 2826
Citation
– analysis, 330
– bibilometrics, 330
– co-citation, 330
– matching, 330
– reference, 330
Class, 2914–2920
Classical MDS, 1784
Classification, 985, 3352–3355
– algorithms, 3041–3043
– divide and conquer, 332
– error rate, 334
– learning, 331–335
– lightweight ontologies, 1614–1617
– model, 1886
– overfitted, 334
– statistical modeling, 332
– in streams, 340–341
– symbolic learning techniques, 331
– tree, 766, 767, 2469–2473
Classification by association rule analysis
– association rules, 335–339
– associative classification, 335–339
– classification based on associations (CBA), 335, 337, 338
– classificatuon, 335–339
Classifier learning, 151–152
Client-Server DBMS, 342–343
Clinical, 356–359
– data acquisition, 344–348
– data and information models, 348–349
– databases, 1564–1565
– data management systems, 344–347
– data quality and validation
– accuracy, 349, 350
– clinical data quality, 349–350
– completeness, 349, 350
– reliability, 350
– decision support system, 350–353
– documents, 354
– event, 355
– information systems, 351
– observation, 359–360
– ontologies, 360–363
– order, 363–364
– order and purchase order system, 363–364
Clinical document architecture (CDA), 353–354
Closed Itemsets, 365–368
Closed subsequence, 2622, 2625
Closest-pair query, 368–373
Cluster and distance measure, 374–375
Cluster-based classifiers, 341
Clustering, 374–387, 393–396, 699–700, 781–783, 795–798, 1291–1294, 3355–3360, 3417–3425, 3506–3510
– algorithms taxonomy, 384
– index, 1434
– overview and applications, 383–387
– validity, 388–392
– index, 388, 391
– methods, 388
Cluto, 937
Coalescing, 2932–2935
Coarse-grained provanence, 2203
Co-clustering, 2873, 2874, 3045, 3046
CODASYL, 2289
Codd E.F. 2753, 2754, 2759, 2760
Cognitive overload, 259
Collaborative ontology elicitation, 1967, 1969
Collaborative software, 2667
Collection of time-referenced data, 2957
Collections and collection operations, 1928
Color model, 1375, 1377
Combination of evidence, 3479
Combinatorial search, 738–741
Common internet file system (CIFS), 1900
Common Warehouse Metadata Interchange, 401–404
Common Warehouse Metamodel, 401–404

Compact encoding, 2, 3
Compact representation of decision rules, 769
Comparative analysis, 405
Comparative visualization, 405
Compatibility, 2936–2945
Compensating transactions, 2588–2590
– subtransactions, 406
Complacency, 1510, 1511
Completeness, 3017
Complete query language, 406–411
Complex event, 411–413
Complex event processing (CEP), 412–413, 1029–1032, 1055, 1068
Complexity theory, 2041
Complex query predicates, 1858
Complex set of decisions, 769
Complex shapes, 809
Component abstraction, 6
Component based software development (COSD), 414
Composed services, 413–418
Composed services and WS-BPEL, 413–418
Composite event, 418–419, 1045, 1059–1062
Composite web application, 3482–3483
Composition, 419–420
Composition of fuzzy relations, 1209
Compressed bitmap indices, 1456
Compressed full-text indexes, 1443–1447
Compressed structured text, 1679–1684
Compressed suffix
– array, 1443
– tree, 1446
Compression, 421–429, 3051, 3052, 3054
– distance function, 424–426
– mobile location data
– scheme, 1684
– techniques, 621, 622, 625
Computer aided annotation, 1385, 1388
3D computer graphics, 2506
Computerized physician order entry (CPOE)
– clinical decision support, 434–436
Computerized provider order entry, 432–437
Computer performance, 2086
Computer Supported Collaborative Work (CSCW), 1327
Concatenated transactions, 493
Conceptual data model, 2559
Conceptual design, 2940, 2944
Conceptual interfaces, 1554, 1555
Conceptual model, 1004, 1008, 1009, 2559, 2944
Conceptual model for integration of software systems, 2633
Conceptual modeling, 1942, 1966, 1967, 2572
Conceptual schema, 1645–1648
Conceptual schema design, 438–442
Conceptualization, 723–728
Concurrency, 2627, 2631
– control, 443–444, 2591, 3144–3156
– control and recovery in B-tree, 274
Concurrency control manager (CCM), 445
Concurrent programming languages, 1554–1556
Condition, 960, 2547
Condition-action coupling, 34
Conditional dependencies, 458, 460, 462
Conditional logic, 2006
Conditional random fields, 397, 399, 400
Conditional tables, 1407–1409
– certain answers, 447
– incomplete information, 447
– naive tables, 447
– null values, 446
Condition evaluation, 959
Confidentiality, 729–730, 2783–2789
Confidential outcome attribute, 1735
Conflation, 1608, 1609, 2790, 2791, 2793
Conflict(s), 1991–1994
– equivalence, 1948
– *vs.* non-conflict operations, 1794
– resolution, 628
– resolution mechanism, 1518
Confluence, 37
Conjunction, 448, 449
Conjunctive query(ies), 93, 448–449
Connection, 449
Connection pool, 449
Connectivity, 696–699, 1977–1978
Connectivity logic, 998–1000
Connector, 696–699, 1978
Consensus, 1510, 1511
Consequence of an event, 1046
Consistency, 722, 1071–1072, 1510, 1925
– data, 452
– models, 626–628
– problem, 462
– query answering, 452, 1410, 1413, 1414
Consolidated storage, 2798
Constant interval, 2925, 2926, 2928, 2995
Constraint(s), 323–327, 704, 705, 708, 1179–1183, 1940–1941, 2754, 2757–2760
– based mining, 1182
– databases, 452–457
– model, 454
– query languages, 454–457
– RBAC, 91–92
Containment relationships, 1308
Content acquisition, 224–228
Content-and-structure query, 463–464
Content-based image retrieval (CBIR), 229, 231–232, 1354, 1356, 1357, 1368–1370, 1373, 2379–2381
– relevance feedback, 2282–2285, 2287
– target search, 2282, 2284, 2287
Content-based indexing, 1350
Content-based profilers, 2093
Content-based publish/subscribe, 464–466
Content-based publish/subscribe systems over peer-to-peer networks, 2070
Content delivery network (CDNs), 2800, 3442
Content description based on invariant features, 1364
Content description based on semantic features, 1364–1365
Content determination, 3049
Content distribution, 2061–2064
Content links, 941
Content modeling, 3276–3280
Content-only query, 473
Context, 473–474
– aware user interfaces, 46–47
– dimension, 474
– of a time, 2377
– of use, 46–47
Contextual advertising, 3458
Contingency transaction, 1979
Continuous data, 3437
– consistency, 3443–3444
– stream, 492
Continuous data protection (CDP), 478
Continuous event context, 419
Continuous events, 35
Continuous κ-nearest-neighbor (C-κNN) queries, 2443, 2445
Continuous media data (audio/video), 1814
Continuous microaggregation, 1736
Continuous monitoring of spatial queries, 479–483
Continuous multimedia, 484–487
Continuous multimedia data retrieval, 484–487
Continuous queries, 492–493, 3533, 3534, 3536
– models, 490
– in sensor networks, 488–492
Continuous relevance scale, 2747, 2748
Continuous spatial query, 479
Continuous time model, 3112
Continuous uncertainty, 648, 649
Continuous variables, 1660
ConTract, 493–494
Contrast pattern based classification, 985
Control data, 494
Control-flow, 3553, 3554, 3556
– diagrams, 41–45

– patterns, 3557, 3558
– statement, 1661
Controlled rounding, 1473
Controlled tabular adjustment, 1473
Conventional query processing, 51
Conventional storage subsystem, 1899
Convergence, 3453, 3454
Convertible anti-monotone, 494, 495
Convertible constraints, 494–495
Convertible monotone, 494, 495
Convex subset, 243, 2966
Coordination, 495, 496
1-Copy correctness, 2385
Copy on write, 2118
Copyright, 496–500
– issues in databases, 496–500
– law, 1013
– protection, 826, 828
Copyright owner's rights, 498
Copyrightable elements of a database, 1611
1-Copy-serializability, 2854
1-Copy-SI, 2854, 2856
Co query, 473
CORBA event service, 2210
CORBA notification service, 2210
Core RBAC, 91–92
ε-Coresets, 1240–1242
CORIE, 2254
Corpora, 938
Corpus, 3041, 3042
Correlated noise addition, 1911
Correlation analysis, 89
Correlation clustering, 2873–2875
Cost-based optimization, 1996
Cost-based query optimization, 2290
Cost estimation, 506–510, 2548
Cougar, 710
Counting triangles, 1273–1274
Count-Min sketch, 511–515
Coupling, 516, 1661–1662
– modes, 33–35
– of system components, 516
Coverage, 2748
Covering index, 516–517
CPU register, 1707
CQL, 2521
Crash, 517–522
Crash recovery, 517–522, 1643
Crawl-and-index, 785, 787
Crawler, 1419–1420, 1463, 1464, 1466
Creation of index files, 1529
Cross-language information retrieval, 523–527
Cross-language web mining, 523–527
Cross-lingual query suggestion, 525–526
Cross-modal multimedia information retrieval (CMIR), 528–532
Cross product, 308–309
Cross-references, 941
Cross-validation, 532–537
Cube, 538–539, 1950–1953
Cube implementations
– aggregation, 539, 541, 543
– dimension, 539–543
– fact table, 539–543
– materialized view, 540
– measure, 539, 542
Cumulated gain, 1016, 1020, 1021
Cumulative event context, 419
Curation, 224–228, 816
Current, 1920–1924
Current semantics
– temporal databases, 544
– temporal upward compatibility, 544–545
– timeslice, 544, 545
Curse of dimensionality, 545, 546, 1439
Cut, 3316–3319
Cut criterion, 2749
Cyberinfrastructure, 825
Cyclic redundancy check (CRC), 327–328, 2823

D

DAC policies, 864
DAML+OIL, 2008
Daplex
– functional data model, 547
– query language, 547
DAS, 2824, 2826
Data access, 2783–2789
Data acquisition and dissemination in sensor networks
– data acquisition and dissemination protocol, 548
– wireless sensor networks, 548
Data aggregation, 2617
Data allocation, 890, 891
Data architecture, 1516
Data authentication, 177, 179
Database, 257–259, 3018–3023
– auditing, 172–174
– buffer, 277, 280
– buffer pool, 285
– design, 708–709
– forensics, 174–176
– machine, 714
– middleware, 719–721
– profile, 604–607
– repair, 722–723
– replica, 188–192
– reverse engineering, 723–728
– state, 36
– techniques to improve scientific simulations, 733–738
– trigger, 738
– tuning, 738–744
Database cluster (DBC), 700–703
Database cluster architecture, 701–702
Database fact, 3162
Database integrity, 2972–2986
Database interfaces, 1162, 1164
Database Language SQL, 2753, 2755, 2760
Database machine, 1551, 1552, 2027, 2028
Database managed file, 2797
Database management systems (DBMSs), 714–718, 1162, 1164, 1165, 2178, 2180, 2181, 2184
Database repair, 452
Database replication, 1738–1742, 2392–2396, 2403–2407
Database server, 342, 343
Database system layers, 3156
Database transaction, 3150
Database tuning using combinatorial search, 738–741
Database tuning using trade-off elimination
– tuning, 744–747
– workload, 744–747
(Lack of) Database use in science applications, 748–750
Database vacuuming, 3023
Data broadcasting, caching and replication in mobile computing
– data broadcasting, data caching and replication, 558–560
– intermittent connectivity, 557–558
– multi-hop infrastructures, 557
– single-hop infrastructures, 557
Data-centric techniques, 2618
Data cleaning, 561–563, 566, 568, 569, 780–784
Data cleaning and curation, 749
Data collection, 564–565
Data complexity, 2041
Data compression, 564–565
Data consistency, 3442–3444
Data cube, 2014, 2016
Data dictionary, 570–571
Data dissemination, 1687–1689
Data encryption, 571–573
Data encryption standard (DES), 572
Data error, 565–569
Data estimation, 574
Data exchange, 575–581
– problem, 575, 578, 580
– setting, 575, 576, 578–580
Data expiration, 3023, 3025, 3027
Data exploration, 3342
Data extraction, 3465–3470, 3472, 3569

Data flow, 3555
Data formats, 133
Data fusion, 1705, 2556, 2558
Data generator, 2339–2343
Data grids, 133, 813–815
Data inconsistency, 565–569
Data independence, 2011, 2754
Data integration, 590–593, 1491–1495, 1564, 1719, 1721, 2055, 2065–2069, 3332–3339, 3472
Data integration architectures and methodology for the life sciences
– data integration, 585–589
– link-based integration, 586–587
– materialized integration, 586, 587
– virtual integration, 586, 587, 589
Data intensive simulation, 3525
Datalog, 751–754, 1547–1550, 2302–2306
Data management, 107–110
– for large graphs, 1262
Data manipulation, 2754, 2757, 2758, 2760
Data mart, 594, 686, 688–691
Data mining, 574, 595–598, 692–695, 986, 1180, 1183, 1291, 2725–2730, 2948–2950, 3365–3369
– classification, 595, 596
– clustering, 596
– outlier analysis, 595, 596
– pattern mining, 595, 598
– predictive modeling, 595, 596
– process workflow, 1586
– techniques, 2147, 2149
Data mining process, 1586
Data model(s), 1900–1905, 1940, 2119–2122, 2559–2561, 2571–2574, 2599–2605, 2932–2935, 2952–2957, 3132–3139
– for spatial networks, 2714–2718
Data-oriented models, 616, 620
Data outsourcing, 2523–2528
Data partitioning, 599–600, 1292, 1314, 1315, 2024–2026, 3341
Data patterns, 3557, 3558
Data perturbation, 2148
Data placement, 1567
Data privacy, 600–604
– the notion of consent, 600
– and patient consent, 600–604
Data profiling, 561–563, 604–607
– samples, 606–607
Data provenance, 608, 2203, 2205
– the origin of a piece of data, 608
Data quality, 561, 562, 612
– assessment, 608–612
– benchmarking, 609
– dimensions, 612–615
– environment, 1516, 1517
– measurement, 608–612, 1512–1514
– models, 616–620
– policy, 1515, 1516
– roles, 1516, 1517
– tags, 1511
Data rank/swapping, 620–621
Data reduction, 421–429, 621–626
– in database, 622, 623, 625
Data redundancy, 1917
Data replication, 626–630, 2383–2387, 2392
Data retrieval, 3472
Data sampling, 630–633
Data sketch, 634
Data skew, 634–635
Data standards, 1517
Data storage, 3438
– and indexing in sensor networks, 635–637
– management, 1814, 1817
Data stream(s), 136–139, 340, 341, 378, 380, 382, 2768, 2770, 2834, 2836, 2837, 3533–3536, 3610, 3634, 3635
– algorithm, 1948–1949
– management, 52–55
– management architectures and prototypes, 639–643
– model, 2236
– processing architecture, 639–643
– summary, 512, 514, 515
– synopsis, 1143, 3538
Data stream management systems (DSMS), 2475, 3538
Data swapping, 620–621
Data types
– in scientific data management
– arrays, 644, 646
– finite element meshes, 645
– graphs, 645–646
– time series, 644–645
– streams, 1063
Data uncertainty management in sensor networks, 647–651
Data virtualization, 812–813
Data visualization, 652–657, 2430–2431, 2897, 3410
Data volume, 3438
Data warehouse(s), 594, 657–663, 669–674, 684–692, 1454–1457, 1560–1564, 2731–2735, 3330
– architecture, 690–692
– life-cycle and design, 658–663
– maintenance, 664–668
– metadata, 669–674
– quality, 2230–2235
– security, 675–679
– technology, 3525
– versioning, 665–667
Data warehousing, 679–683
– for clinical research
– healthcare, 680, 682
– research, 679–683
– systems, 684–692
DB&IR integration, 1543–1546
DBMS, 1295, 1296, 1901, 1902, 1904, 1905
– component, 755
– interface, 755–756
DDBS, 894–896
Deadlock resolution, 900, 902–903
Decaying aggregation, 758–760
Decaying sum, 758–760
Decision support, 356–358
Decision table (DT), 761–764
Decision tree classification, 765–768
Decision tree classifiers, 341, 765–768
Decision trees, 769, 2469–2474, 3352–3355
Declarative networking, 770–771
Declarative query
– interface, 488
– over distributed network state, 770
Decomposition, 704, 705, 707, 708
De-coupling, 516
Deduction method, 119
Deductive data mining using, 772–778
Deduplication, 562, 779, 2558, 3506–3510
– an address mapping table, 779
– techniques, 779
Deep web, 784–787, 2519
Deferred coupling, 34
Deferred view maintenance, 3326, 3327
Definable sets, 453
Definition of a view, 3325
Deletion anomaly, 1918
Demand-driven pipelining, 2117
Denial constraints, 458, 459
Denomalization, 709
Dense index, 1433, 1434
Dense pixel displays
– very large multidimensional data sets, 789
Dense time model, 3112
Density-based clustering
– efficiency, 795–798
– parameters, 795, 797, 798
Dependency(ies), 704–708, 1009–1010
Dependency preservation, 1919
Depth-first (DF), 1891
Derivation of a collection of events, 411–412
Derivation of an event as a function, 412
Derived fragmentation, 892
Description, 215–217, 221, 222
Description logics (DL), 799–802
Descriptive lightweigh ontologies, 1614–1616
Design for data quality, 803–807

Design of effective icons, 1334
Design patterns, 3557, 3558
Desktop metaphor, 846, 847, 3387
Detail-in-context, 869–871
Deterministic algorithm, 1287
Dewey decimal classification, 808
DF algorithms, 1893
DHT, 2863–2866
Diagrams, 809–810, 3370–3373
– connections, 809
Dialogue specification, 1329
Dictionaries, 965, 966
Difference, 811
Differential backup, 201
Differential privacy constraints, 2147, 2148
Digital archives, 811–816
Digital archives and preservation
– authenticity, 811–813
– chain of custody, 811–813, 815
– integrity, 811–813, 815
– preservation, 811–813, 815–816
– provenance information, 811, 812
Digital asset management, 979
Digital curation, 816
Digital elevation models (DEMs), 817–821
Digital gezetter, 1217
Digital image, 1115, 1116, 1343–1347, 1362, 1367
Digital information, 816
Digital libraries (DLs), 309–314, 821–825, 2518–2521
– federation initiative, 1636–1637
– systems, 823–825
Digital rights management (DRM), 825–830
Digital signature algorithm (DSA), 832, 833
Digital signatures, 830–835, 1715
– authenticity, 830, 832, 835
– integrity, 830, 832, 833, 835
– schemes, 252
Digital video, 466–472, 3271, 3286, 3287, 3289
DIG35 specification, 1349
DIME, 3132–3135
Dimension, 594, 836, 2665, 2666, 2779–2780
– for the knowledge model, 36
– reduction techniques for clustering, 838–842
– table, 2665, 2779–2780
Dimensionality reduction, 3042
Dimensionality reduction technique, 546
Dimensional model, 658, 1777–1783
Dimension-extended topological relationships, 843–845
Direct attached storage (DAS), 846
Direct lateral communication, 295
Direct manipulation, 846–847, 3379, 3380
Direct-marketing advertising, 3457
Dirty data, 566–569
Disaster recovery, 847–848, 1643, 1644
Disclosure, 2783–2789
– metrics, 2138
– risk, 848, 2512, 2513
– re-identification, 848
– uniqueness, 848
Discounted cumulated gain, 849–853
Discovery, 853–857, 3366–3368
Discovery-and-forward, 785
Discovery and integration (UDDI), 854, 855, 857
Discrete models, 1853, 1854
Discrete time model, 3112
Discrete uncertainty, 648
Discrete wavelet, 857–863
Discretionary access control (DAC), 864–866, 2448, 2450
Discriminative approaches, 183
Disease understanding and treatment, 107
Disk, 2799, 2800
– drive, 866–868, 1900
– I/Os, 1333
– power saving, 867–868
– scheduling, 1832–1833
Disk-based index structure, 1222
Display advertising, 3458
Dissolve, 3316, 3318
Distance-based record linkage, 2353
Distance
– between streams, 2842–2844
– function, 1451–1454, 1734
– linkage disclosure risk, 2513
– measures, 374–375
– metric, 1379–1383
Distinct data type in temporal database, 3113
Distinct sampling, 2839–2841
Distinct-values estimation, 1143, 1146
Distinction between telic and atelic data, 2911
Distortion viewing, 869–874
Distributed architecture, 875–879
Distributed computing middleware, 3510
Distributed concurrency control, 879–883
Distributed database, 875–879, 896, 898, 912, 914–916
Distributed database systems (DDBS), 894–896
Distributed data fusion, 583
Distributed data streams, 883–889
Distributed data warehouse, 2012–2018
Distributed DBMS, 875–878, 896–898
– distributed coordination, 897, 898
– one-copy semantics, 897
– update propagation, 897
Distributed deadlock detection, 901–903
Distributed hash tables (DHTs), 903–904, 1627–1630, 2056, 2082
– overlay networks, 903, 904
– peer-to-peer systems, 904
Distributed join, 904–907
Distributed mediator system, 80
Distributed query optimization, 908–911
Distributed query optimization algorithm, 908–909, 911
Distributed query processing, 2592
– join, 914–916
Distributed recovery
– logging, 917, 918
– three-phase commit, 917
– two-phase commit, 917, 918
Distributed representations, 1847
Distributed source coding, 564–565
Distributed spatial index (DSI), 76, 920, 921, 923–924
Distributed spatial join, 920–922
Distributed systems, 2066
Distributed transaction management
– schedules, 927, 928
Distributed transactions, 879–883
Distributed two phase locking, 880, 881
Distributional statistics, 605
Distribution database design, 3340
Divergence between replicas, 451
Divergence from randomness (DFR)
– framework, 1130
– information retrieval models, 929
– models, 929–932
DL-based knowledge base, 801–802
DNA
– microarray, 1218
– of an organism, 1218
3D objects retrieval, 1125–1128
Document(s), 932
– clustering, 933–937
– criterion function, 934–936
– databases
– document repositories, 938
– text databases, 938
– exchange, 38
– field, 939–940
– footprint, 1253, 1254
– formatting systems, 942
– frequency (DF), 1570–1571
– generation models, 2169, 2170
– index, 1425
– layout, 1658
– length, 3937–3942
– length normalization, 940–941
– links and hyperlinks, 941
– management, 997
– management systems, 938
– markup standard, 354

– metadata, 939
– relevance, 2514
– representations (inclusive of native and relational), 942–946, 3041–3042
– structure, 939, 3049
Domain, 2967
– evolution, 2480
– relational calculus, 2370–2371
Domain-independent query, 2463, 2464
Dominance rule, 1473
Downward inheritance, 2911, 2913
DP-matching, 3295
DRAM main-memory, 1707
Drift, 2388, 2389
DTD (Document Type Description), 3572, 3621, 3650–3653
D-tree, 73, 74
Duality transformation, 1459, 1461
Dublin core, 947–949, 1724
– encoding scheme, 948
– metadata element set, 947, 948
Duplicator, 963, 964
Duration of copyright, 497
Dynamic attribute, 1771, 1772
Dynamic graphics, 950–953
Dynamic integrity constraints, 2976, 2982
Dynamic partitioning, 1567
Dynamic random access memory (DRAM), 1668–1669
Dynamic revalidation, 3654
Dynamic web pages, 954

E

Eager propagation, 2856
ECA rules, 35–36, 959–960, 1080, 1404, 1405
– action, 959
– condition, 959
e-Commerce transactions, 960
Eddy operator, 51
Edge detection, 1390–1395
Edge-server system, 3444
Effectiveness involving multiple queries, 961–963
Effectiveness of IR, 1522
Effectivess evaluation, 961
Efficient query processing, 1861
EF-game, 963–964
Egds, 1009, 1010
Ehrefeucht-fraisse games, 963–964
Ehrenfeucht game, 963–964
e_i, 476
Eigenvalue, 2136
Electric power consumption of storage devices, 2821
Electronic dictionary, 965–966
Electronic encyclopedia, 967–968
Electronic health records (EHRs), 344, 968–972
Electronic ink indexing, 974–977
Electronic transaction, 960
Elementary sketches, 81–83
Element shapes, 809
Eleven point precision-recall curve, 981–982
Eligibility criteria, 1013
Embedded dependency(ies), 1156–1158, 2489
Embedded functions, 38
Emergent semantics, 982–984, 2056
Emerging pattern, 107, 985
Emerging pattern based classification, 985
Encina, 1898
Encoded bitmap index (EBI), 246
Encryption, 571–573
Encyclopedia, 967
End-user map operations, 2720
Energy-efficient disk array, 1697
Ensembles, 210
– classifiers, 341
– methods, 262, 988–990
Enterprise application integration (EAI), 991–996
Enterprise architecture, 1516, 1518
Enterprise content management (ECM), 997
Enterprise information integration, 1495
Enterprise java beans, 1578, 1579
Enterprise privacy authorization language (EPAL), 2142
Enterprise service bus, 997–1000
Enterprise terminology services, 1000–1003
Entity beans, 1578, 1579
Entity relationship diagram (ERD), 1004, 1005
Entity relationship model (ERM), 1003–1009, 1083–1091
Environment stack, 2772
Epidemic dissemination, 2082
e-prescribing, 434
Epsilon-serializability, 504
Equality-generating dependencies, 1009–1010
Equality join predicate, 1288
Equality predicate, 1427
Equivalent execution plans, 2548
Error-correcting code (ECC), 2823
Escrow commits and aborts, 1011–1012
Escrow transactions, 1010–1012
Estimation of term probability, 2158, 2160
ETL
– processes, 1095, 1096, 1098–1101, 3471–3472
– systems, 1095
ETM, 3559
European law in databases, 1013–1015
Evaluation, 193, 2448, 3479, 3481
– of filtering systems, 1483–1484
– of information retrieval systems, 192–193
– measures, 1016, 2747, 2748
– metrics, 1015–1024
– and optimization of Datalog, 753–754
– of relational operators, 1024–1029
Evaluative linguistic expressions, 1213
Event(s), 959, 960, 1041–1043, 1053–1058
– in active databases, 1044–1045
– causality, 1033–1034
– channel, 1034
– cloud, 1034–1035
– detection, 1035–1039
– driven architecture, 1040–1043, 1054
– flow, 1044
– instances, 1044
– lineage, 1046
– parameters, 1045
– pattern detection, 1029, 1032, 1046–1048
– pipe, 1063
– prediction, 1048–1052
– relation, 1045
– sink, 1058
– source, 1059
– specification, 1059–1062
– stream, 1063
– stream processing, 1029–1032
– transformation, 1064–1067
– translation, 1064, 1066, 1067
– triggering rules, 959
– types, 1035, 1036, 1059–1062
Event-component information model, 349
Event-condition coupling, 34
Event consumption modes, 35
Event detection, 3303–3308
Event detection and recognition, 1477
Event-driven applications, 1041, 1043
Event-driven business process management, 1068–1071
Event driven process chains, 293
Event-driven programming, 355
Event monitoring, 29, 31, 32, 2093
Event pattern, 2431, 2433
Event probability, 3224
Event processing, 1033, 1034
– agent, 1052–1053
– network, 1033, 1034, 1046, 1052, 1053, 1058, 1059, 1063
– systems, 1053–1058
Event-subscription matching over structured p2p networks, 2070
Eventual consistency, 3452, 3453, 3455

Event uncertainty, 3221–3224
Evidence based medicine, 1072–1073
Evolution and versioning, 665–667
Evolutionary algorithms, 1224–1227
Evolutionary computation, 1224–1227
Exactly once application, 100
Exactly once execution, 100–103
Exact ranking of the relevant documents returned, 195
Exchangeable image file (EXIF), 1349
Excluded terms, 2795
Exclusion rights, 496, 497
Executable descriptions of distributed algorithms, 770
Executable processes, 288
Execution cost, 506, 507
Execution order dependencies, 1142
Execution plan, 1575
Execution skew, 1079
Experimental data, 132–135
Experimental tool, 1737
Explicit cross-references, 941
Explicit event, 1080, 1404, 1405
Exploratory data analysis (EDA), 1080, 2136
Exploratory stage of data mining, 1080
Explortary spatial data analysis (ESDA), 2691
Exponential decay, 758–759
Exponential histograms, 759, 760
Expressiveness, 1081, 1083
Expressiveness of query languages, 406
Expressive power of query languages, 1081–1083
Extended entity-relationship (EER)
– data model, 438, 1083–1091
– schema, 1646, 1647, 1649
Extended S-reducibility, 2620–2621
Extended transaction models (ETMs), 493, 3559
Extendible, 1093–1094
Extensible architecture, 3013–3015
Extension, 2936–2937, 2941, 2943
0-Extension, 122
1-Extension, 122
Extensionalization, 1923
Extensional relational databases (ERDB), 761, 762
Extent (of temporal granularity), 2970
External DBMS interface, 756
External hashing, 1289–1290
External quality measure, 3046
External storage, 635–637
Extraction, 2511
– of metadata from raw video, 3271
– transformation, and loading (ETL), 1095–1101

F

Faceted browsing, 1973, 1976
Faceted metadata, 1103, 1109
Faceted search, 1103–1108
Facility location, 2419–2423
Factoid question answering, 3486, 3487, 3489
Fact-oriented modeling, 1941–1946
Fact table, 2665, 2666, 2779, 2780
Faithful stream operator, 2216
False positive probability, 253
False positive rate (FPR), 2349–2351
Fault tolerance, 2397–2402
FCP, 2824–2826
Feature-based approaches, 1125
Feature-based 3D object retrieval, 1125–1128
Feature-based object model, 1938
Feature extraction, 161
Feature extraction for content-based image retrieval, 1115–1119
Feature selection, 161
– clustering, 1119–1124
– high-dimensionality, 1119, 1124
Feature space, 1379, 1382
Feature vector approach, 1126
Federated search, 2518
Feedback, 2414–2415
Feedback control, 2553–2554
Feed-forward ANN, 1906
Fellegi–Sunter model, 2354, 2356
Fetch as needed, 905
Fiber channel, 2824–2826, 2829
Fibre Channel and iSCSI, 2812
Field-based information retrieval models
– combination of field information, 1129, 1130
– document representation, 1129
Field-based spatial modeling
– discrete machines, 1133
– field, 1132–1137
Field-dependent normalization, 1130
File sharing networks, 2076
File system, 2796, 2797
Filtering systems, 1482–1484
Filter/refinement query processing, 1860–1862
Fine-grained provenance, 2203
Finite automata, 3635
Finite axiomatizability, 1398
Finite element meshes, 645
The finite implication problem, 1396–1400
First committer wins, 2661, 2662
First-order formulae, 1140
First-order incremental algorithms
First-order logic (FOL), semantics
– definability, 1138, 1139
– first-order structures, 1138
First-order logic, syntax theorems, 1140, 1141
First-order vocabulary, 1140
First principal component, 2136
Fisheye views, 869, 870
Five RAID levels, 2359
Fixed-group microaggregation, 1736
Fixed-parameter tractability, 2041, 2043
Fixed time span
– time interval, 1141
Fixpoint theory, 753
Flash, 2798–2800
Flashback query, 3120
Flash memory, 286, 287
FlexPath, 3638
Flex transactions, 1142, 1222
Flux, 1111–1113
F-measure, 2776–2778
– a compromise between precision and recall, 1147
– precision, 1147
– recall, 1147
– weighted harmonic mean, 1147
FM synopsis algorithm, 1143–1146
Focused crawling, 2520
Focused crawling using context graphs, 1150–1151
Focused retrieval, 2867, 2868
Focused web crawling, 1147–1154
Focus-plus-context, 869
Folding, 2933
FOL modeling of integrity constraints (dependencies), 1155–1160
For "Connection": Connection, 2636
Foreign key, 2756, 2758
Forever
– end time, 1161
– infinity, 1161
– valid timetime value, 1161
Formal concept analysis, 365
Format, 217, 221, 222
Forms-based interfaces, 1162, 1164–1166
For "Session": Session, 2636
Fourth normal form (4NF), 1166–1167
Fractals, 2675
– space-filling curve, 1168–1169
Fragmentation and replication, 909
Fragment of genetic or protein material, 224
Frame-by-frame comparison by continuous dynamic time wraping (DTW), 3295
Frecpo algorithm, 1189
Frequency moments
– computational complexity of frequency moments, 1170
– frequency, 1169–1170
– stream, 1169–1170

Frequency table, 2908
Frequent closed itemsets, 140
Frequent elements, 1175–1179
Frequent graph patterns, 1170–1175
Frequent itemset mining, 124–126, 365, 367, 1179–1183
Frequent itemset mining with constraints, 1769
Frequent itemsets, 136–139, 1179–1187, 2051, 2052, 2054
Frequent items on streams
 – heavy hitters, 1175–1179
 – hot items, 1175–1179
Frequent pattern, 1188
Frequent subgraph mining, 1170, 1171, 1173, 1174
Frequent subsequence, 2622–2625
Frequent updates, 1458
Freshness, 2386
Full-text index, 1571
Full-text search extension to the XQuery 1.0, 3665
Fully-automatic web data extraction, 1200191–1193
Functional data model, 1193–1198, 1201, 2011
Functional dependency(ies) (FD), 1010, 1155–1158, 1199, 1200, 1411, 2960–2962
Functional dependency implication, 135
Functional design, 1973
Functional expressions, 1194
Functional query language and data model, 2007
Functional query languages (FQL), 79–80, 1201–1204
Functions, 1194, 1195, 1197
Fundamental requirements of stream processing, 2849
Fuzzy generalizations, 1205, 1206
Fuzzy IF-THEN rules, 214, 1211, 1213
Fuzzy/Linguistic IF-THEN rules and linguistic descriptions, 1214–1215
Fuzzy logic, 2418, 2419
Fuzzy models, 1205–1209, 1808, 1812, 1813
Fuzzy preference relation, 1212
Fuzzy relations, 1210
Fuzzy set, 1209–1210
Fuzzy set theory, 1207, 2418

G

Gazetteer, 1252
Gene expression arrays, 1218–1220
Gene ontology, 1960–1962
Generalization, 989, 990, 1913, 2746–2747, 3018
Generalized ACID, 1091–1092
Generalized search tree (GiST), 1222–1224
General network connecting storage devices, 2797
General secure database systems, 2530
Generative algorithms, 183
Genetic algorithms, 1224–1227
Genetics/Genomics, 1400–1404
Genomics algebra, 2262
Geo coding, 1252–1253
Geo crawling, 1252
Geographical databases, 2571
Geographically-distributed crawl, 3465
Geographically referenced data and information, 1246, 1248, 1249
Geographic data modeling, 1133
Geographic focus, 1252, 1253
Geographic information retrieval (GIR), 1227–1231, 1251, 1254
Geographic(al) Information Systems (GIS), 1231–1235, 1639–1642, 2571–2575, 2655–2656
Geographic scope, 1253
Geographic score, 1253
Geographic web search, 1251–1255
Geography markup language (GML), 1236–1238
Geometrical primitives, 1374
Geometric match, 1230
Geometric mean average precision (GMAP), evaluation metric 1256
Geometry-based object model, 1937
Geo parsing, 1252–1253
Geo raster services, 2332, 2333, 2336
GEO-RBAC Model, 1244, 1245
Georeferencing, 1246–1249
Geo search, 1251–1253
Geospatial, 1249–1251
 – coordinates, 1246
 – referencing, 1246–1249
Geo tags, 1251, 1252
Geo-targeted search, 1251–1255
Gexetter entries, 1217
Gezetter(s), 1217–1218
GIRAS, 3132, 3135, 3136
GIS operations, 2720
GIS software, 1232, 1234, 1235
Global and Local as View (GLAV), 338, 2489, 3332, 3333
Global as View (GAV), 1492, 2489, 3332–3334
Global concurrency control strategies for replicated databases, 2391
Global features, 1379–1381
Global positioning system (GPS), 1639
Global recoding, 1912, 1913
Global serializability, 926–928
Global transaction, 1142
Google earth, 1854
Google map, 1232
GOPPR, 1728
Gossiping-based methods, 1688
Government, 292
GPS data, 1692–1695
Grammar, 1256–1259
Grammatical data stream, 2216
Gramm-Leach-Bliley act (GLB ACT), 2140
Grand tour, 950, 951, 953
Granular computing, 772–778
Granularity, 2968–2973
Granularity of the occurrence time, 2975
Granularity of the reference to time, 2975
Graph, 1260–1263
 – classification, 1749
 – clustering, 1749
 – connectivity, 1272
 – databases, 1268–1269
 – kernel, 1748, 1750
 – management 1266–1271
 – mining, 1749
 – search, 1749
 – XPath, 1265
Graphical language, 3400
Graphical User Interfaces (GUIs), 847, 3379, 3387
Graphic techniques, 1080
Graph query languages, 1269–1270
Gregorian calendar, 304
Grid and workflows, 1276–1278
Grid file (and Family), 1279–1282
Grid-partition index, 74
Grid quorums, 2314, 2315, 2317
Grid workflow, 1276–1278
Group communication, 2392–2397
GUID, 2428–2429
Guideline modeling languages, 1075
GUIs for web data extraction, 1282–1286

H

Haar wavelet, 3446, 3447
Hamming code, 2359
Handwritting recognition, 973, 974
Hard disk, 866–867
Hard disk drives, 2821
Hardware architecture, 127–131
Hardware tuning, 2114
Hash-based index, 1289–1290, 1707
Hash-coding methods, 252
Hash functions, 1287–1288
Hashing, 1093–1094, 1619–1622
Hash join, 1026–1028, 1288–1289
Hash partitioning, 2025, 2026
Hash structures, 1433, 1434
Healthcare, 292–293, 358, 359
Health insurance, 2140

Health Insurance Portability and Accountability Act (HIPAA), 2364
Health level seven (HL7), 354
Health Level 7 (HL7) standards, 1558
Heap, 1434
Heavy hitters, 511, 513, 1305–1307
Heavy hitters on data streams, 1305
Heterogeneity, 1704–1706
Heuristic microaggregation, 1736
Hidden markov models, 398
Hidden web, 784, 1466, 2519, 2520
Hierarchical clustering, 380, 381
Hierarchical data format (HDF), 1777
Hierarchical data model, 715
Hierarchical data summarization (HDS), 1300–1304
Hierarchical deadlock detection, 900–902
Hierarchical entity-relationship model, 1083–1091
Hierarchical heavy hitters, 1304–1307
Hierarchical memory system, 1707, 1708
Hierarchical model, 1295, 1298, 1299
Hierarchical RBAC, 91–92
Hierarchical relationship, 2606, 2608
Hierarchical storage management (HSM), 1499
Hierarchical task analysis (HTA), 1329
Hierarchichal data, 836
Hierarchy, 1308–1309
High availability, 2397–2402
High dimensional datasets, 838, 1309, 1310, 1313
High-dimensional feature vectors, 1806
High-dimensional indexing, 837, 2571, 3314, 3315
Higher-order entity-relationship, 1083–1091
High Performance Computing (HPC), 2816, 2817, 2820
Hilbert curve index (HCI), 75
Hilbert's spaces, 3260
Histogram intersection, 1380–1383
Histograms, 1315–1318, 2898
HITS, 3484, 3485
HL7 reference information model (RIM), 349, 355, 1558
Hold-out validation, 534
Homomorphic encryption schemes, 1320, 2535–2537
Homomorphic property, 1320
Homotopy, 2652, 2653
Horizontal partitioning, 599–600
Horizontal fragmentation, 892
Horizontal partitioned data, 1321–1322, 3264, 3341
Horizontal replication, 2409, 2411, 2412
Huffman Coding, 1375, 1377, 3047, 3048
Human-based computation, 1324
Human-centered computing (HCC), 1323–1326
Human expressions, 1880, 1881
Human factors and ergonomics, 3247, 3248
Human visualmeans, 1335
H.26x, 3297, 3298, 3301, 3302
Hypercube, 538–539
Hypergraph, 2043
Hyperlink analysis, 84
Hyperlink relevance, 3499
Hypertext, 268, 269, 941
Hypertext interfaces, 3376–3378
Hypertexts, 1331–1332
Hypothesis generation, 3065, 3068
HyTime, 3124

I

Icon, 1334
Iconic displays, 1335–1342
Identification, 176–180
Identifier, 1735
Identity-based relationship, 1936
Identity disclosure, 848
IDF weight, 1571
IDMS, 1901
Image, 1343–1348
Image (of temporal granularity), 2970
Image acquisition, 1344–1345
Image annotation, 1358–1361
Image compression, 1375, 1378
Image content modeling, 1349–1353
Image database, 1353–1361
Image database schema, 1354–1355
Image metadata, 1358–1360, 1362–1368
Image processing, 229, 230
Image processing and analysis, 1345, 1347
Image representation, 1345, 1348, 1374–1379
Image retrieval, 85–88, 1384–1388
Image segmentation, 1370, 1373
Image similarity, 1379–1383
Immediate coupling, 34
Immediate view maintenance, 3326, 3327
Implementation, 2893, 2895
Implementation abstraction, 6, 7
Implicit event, 1404–1405
Improving storage availability, 2798
IMS, 1294–1299
Inclusion dependencies, 1156–1158
Incomplete information, 1405–1410, 1876
Inconsistent databases, 452, 1410–1414
Incremental backup, 201
Incremental computation of transitive closure, 1417
Incremental maintenance, 2549
Incremental maintenance of views with aggregates, 1421–1424
Incremental query computation, 1414, 1415, 1417
Incremental view maintenance, 1414, 1417, 3326
Independent parallelism, 1566
Index, 244–248, 2877, 2879, 2894
Index, B+-tree, 1433
Index creation and file structures, 1425–1427
Indexed sequential access method (ISAM), 197
Indexing (or Text indexing), 257, 258, 788–789, 1309–1313, 1437, 1448–1450, 1458–1462, 1467–1471, 2643, 2646, 2680–2681
Indexing
– granularity, 1468–1471
– data warehouses, 1454–1457
– schemes, 2109
– units, 1467–1471
Index intersection techniques, 517
Index join, 1427–1428
Index-only execution plan, 516
Index sequential access method (ISAM), 2681
Index structures, 1453
Index structures for biological sequences, 1428–1432
Index
– term, 3259–3262
– structures, 1453
– tuning, 1433–1435
Indices, 1454–1457
Indices for multi-dimensional points, 2703–2704
Individualism, 2136
Individually identifiable data, 1471–1472
Induction, 1256–1259
INEX, 1545, 1876–1879
Inference, 705–706, 1256–1259
– control in databases, 1472
– problem, 1789–1791
INFO-D, 2210
Information
– architecture, 1103, 1104, 1109
– browsing, 3376
– extraction, 233, 234, 397–400, 1476–1481, 1595–1600, 3472–3478, 3486, 3487, 3569
– extraction function, 1649
– filtering, 1481–1484
– foraging, 267, 1485–1490
– hiding, 826, 829–830
– integration techniques for scientific data, 1496–1498

– lifecycle, 2364
– lifecycle management, 1499
– loss, 2512, 2513
– losslessness, 1919
– loss measures, 1499–1501
– modeling, 1941, 1942
– navigation, 1501–1502
– network analysis, 595, 596
– theoretic models, 1524, 1525
Information product (IP) approach, 1502, 1503, 1505–1508, 1515, 1516
Information product map (IP-MAP), 616, 618–619
Information quality (IQ), 1502–1511
– dimensions, 1503–1505, 1507, 1508
– managing information as a product, 1502–1508
– measurement, 1512, 1514
– policy, 1515–1519
Information queries, peer-to-peer, 111–113
Information retrieval (IR), 233–235, 266, 375–378, 849, 1016–1019, 1022–1024, 1519–1528, 1703, 2127–2129, 2377, 2875, 3072, 3075, 3472, 3506–3508
– operations, 1528–1530
– system effectiveness, 1528
Information retrieval systems, 962, 1520, 1521
Information-seeking behavior, 1485
Information seeking process, 3410, 3411, 3413
Information theoretic models, 1524, 1525
Information visualization, 2506
Information visualization and visual analytics, 3388, 3390, 3392
Informedia, 466, 469
Infrastructure independence, 812, 813
Initial nodes in the event processing network, 1059
INitiative for the Evaluation of XML retrieval (INEX), 1531–1537, 2191–2194, 2197
Initiator, 143, 1537–1538
In-memory
– buffers, 1800
– DBMS, 1669
In-network
– indexing, 635–637
– processing, 884, 888, 889
– processing of aggregates, 1540–1541
– storage, 635–637, 1757
In-Network Query Processing, 1538–1542
Insertion anomaly, 1918
Insertion-deletion technique, 1423
Instance-based learning, 1886–1888
Instance-driven storage, 3628–3630
Instances of roles, 1245
Instantaneous event, 143
Instant relation, 1045
Integrated data store (IDS), 1901
Integrated temporal reasoning approaches, 2227
Integration across enterprises, 991
Integration of rules and ontologies, 1546–1550
Integrity, 730–731
– and authenticity of a message, 1715
– constraints, 722–723, 1410–1412, 1414, 2972–2986
Intelligent disk, 37–38
Intelligent storage systems, 38, 1551–1553
Interaction, 3366, 3368, 3369
– between molecules, 220
– styles, 3529
Interactive analysis, 3343
Interactive browsing, 377
Interactive graphics, 1866, 1869
Interactive integration, 1498
Interactive layout, 3382
Interactive query-driven analysis, 3388
Interactive retrieval, 1384, 1386, 1388
Interactivity, 78
Interface, 955, 957, 1553–1557
Interface engines in healthcare, 1557–1560
Interfunctional integration of business processes, 295
Internal quality measure, 3046
Internet SCSI (iSCSI), 1574, 1575
Internet transactions, 3523
Interoperability, 1560–1564
Inter-operator
– load balancing, 2269–2271
– parallelism, 1566, 1982
Interpolation-function, 2995
Interpolation-granularity, 2994, 2995
Inter-query parallelism, 702, 1566–1567, 2038–2040
Interval encoding, 246
Interval preservation, 2620, 2621
Intra-operator
– load balancing, 2269–2272
– parallelism, 1567
Intra-query parallelism, 1567–1568, 2038–2040, 3340, 3341
Intrinsic skew, 635
Intrusion detection, 89–90, 1568–1570
Invariance, 3362
Inverse distance weighting, 2737
Inverse Document Frequency (IDF), 1570–1571, 3937–3942
Inverse-rule algorithm, 2438, 2440, 2441
Inverted file, 3058, 3059, 3061
Inverted index, 1425–1427, 1571, 3051–3055
Inverted index compression, 3052, 3054
Inverted list, 1571–1574
Inverted list compression or inverted file, 3052
Invisible data mining, 597
Invisible web, 2519
I/O computation, 1333
I/O model of computation, 1333–1334
IP storage, 1574–1575
IR effectiveness, 849, 851
IR evaluation metrics, 849, 853
IR relevance ranking algorithms, 3497
ISAM file (a static tree), 3173
iSCSI, 2824–2826
iSCSI target, 1575
ISO/IEC 11179, 1724–1727
Isolation property, 144
Isomorphism, 2603–2604
ISO standard, 2754, 2755, 2759
Iterator, 1575–1576

J

Java application server, 104
Java Database Connectivity (JDBC), 449
– connectivity, 1577
Java enterprise edition (JEE), 1578–1579
Java metadata facility, annotation types, 1580
Java program elements, 1580
J2EE, 1578
JMS, 2210
Join, 1580–1582, 2592–2595, 2673–2674, 3544–3545
Join dependency(ies)(JD), 1156–1158, 1581–1582, 1865
Join index, 1582
Join order, 1582–1583
Join ordering, 2273, 2275–2277
Juke box, 2799

K

k-anonymity, key attribute, 1585
KDD process, organisational scheme, 1586
Kerberos, 2828
ε-Kernel, 1240, 1242
Kernel estimation, 758
Key attribute, 1735
Key dependency, 1200
Keyframe extraction, 3281, 3284
Key range locking, 273–277
k-fold cross-validation, 533, 534, 536
K-means, 1588, 1589
K-means clustering, 3045
κ-Medoid, 1588, 1589, 2419, 2420, 2422, 2423
k-nearest neighbor query (kNN), 1890
K-nearest neighbors, 1371

Knowledge, 356–359
- acquisition, 1966
- base, 3472
- discovery, 3065
- management process, 297–298
- representation, 1074, 2914, 2915, 2920
- representation languages, 799, 802, 2427

Knowledge discovery in databases (KDD), 3061–3065

L

Label (of temporal granule), 2971
Labeled graph, 1170–1172
Laboratory information management system, 1402
Lamport-Diffie one-time signature scheme, 1715
Language(s), 257–260, 3370–3373
Language model(s), 1090, 3058, 3069
Large scale scientific experiments, 2816
Large-scale text retrieval, 3479
Latch coupling, 273–277
Latches, 273–275
Latent Semantic Indexing (LSI), 3076, 3078, 3260–3262
Latin hypercube sampling, 2900
0–1 laws, 3683
Layer, 3018–3023
Layered, 3012–3015
Layers of DBMS, 2179–2180
L1 cache, 2195–2196
L2 cache, 2195–2196
Learning distance measures
- classification, 1601–1604
- clustering, 1601–1604
- nearest neighbor methods, 1603
- semi-supervised clustering, 1604–1605
- support vector machines, 1604

Learning environment, 1517
Learning from unbalanced data, 1664
Learning to rank, 3481
Least recently used (LRU), 1800
Leave-one-out cross-validation, 534
Legacy software, 2936, 2937, 2943
Legacy system, 1299, 1901
Lexical analysis of documents, 1528
Lexical relations, 3036
Lexicographic information, 965
Library of Congress (LOC), 1636
- classification, 808

Licensing and contracting issues in databases
- contract, 1610–1612
- database license, 1610–1612
- license, 1610–1612

Life science data management, 1960–1962, 3519
Life sciences, 1266–1271
Lifespan, 98–99
- existence time, 1612
- valid time, 1612

Lightweight ontologies, 1613–1619
Linda, 495, 496
Linear, 1619–1622
Linear content, 1825
Linear dimension reduction, 838–842
Linear hashing, disk-based, 1619
Linearizability, 1743
Linear regression, 1622, 2044, 2045
Linear regression analysis, 1622
Linear speed-up, 2031, 2032
Linguistic descriptions, 1214–1215
Link analysis, 3485
Linking, 1623–1626
Linking and brushing, 1623–1626
Links between document fragments, 1331
LinQ, 3606
List comprehension, 420–421
List or file compression, 3051–3055
Literature-based discovery and exploration, 3065
Load-balancing, 719–721, 1627–1632, 2637
Load balancing problems, 2269, 2271
Load shedding, 54–56, 1632–1635
Local advertising, 1251, 1254, 1255
Local as View (LAV), 1492, 3332, 3333, 3336
Local features, 1380, 1383
Locality, 59, 1637–1638, 1713–1714
Locality-preserving mapping, 2675, 2679
Local split decision tree (LSD-tree), 2703–2704
Local storage, 635–637
Local suppression, 1912, 1913
Location and time based RBAC (LoT-RBAC), 1245
Location-based services (LBS), 2718
- privacy, 1640

Location
- predictions, 2695–2698
- transparency, 2416, 2417

Locking granularity and lock types, 1641–1642
Lock manager, 445
LOC METS, 1636–1637
Log component, 1645
Log data, 517, 519
Logging, 1957
- methods, 519
- and recovery, 1643–1644

Logging/recovery subsystem (LSR), 1644–1645
Logical connectives, 1928
Logical database design: from conceptual to logical schema, 1645–1649
Logical definition of relevance, 1652
Logical foundations of web data extraction, 1649–1651
Logical models of information retrieval, 1652–1657
Logical observation identifiers names and codes (LOINC), 364
Logical organization of the content, 1658
Logical schema, 708, 709, 1645–1649
Logical storage, 2830
Logical Story Unit, 3303
Logical structure, 1658, 2376
Logical uncertainty principle, 1652, 1653, 1656
Logical unit number (LUN), 1658–1659
Logical unit number mapping, 1659
Logical volume (LV), 1659, 1660, 3438
Logical volume manager (LVM), 1659–1660
Logical WORM, 3570
Logic-based wrapping, 1649
Logic programming, 2302, 2304
Logic programming language Prolog, 2011
Log-linear regression, 1660
Long-duration transaction, 2588–2589
Long-lived transaction, 493, 494
Long-running processes (macroflows), 3166
Long running simulation, 734, 737
Loop, 1661
Loose coupling, 1661–1662
Lossless join, 1581
Low-power modes, 2822
LRU (least recently used), 283
- replacement, 278–280

LRU-K, 747
LUN masking, 1659

M

MAC
- for database, 1685
- functions, 1715
- for networks, 1685

Machine learning, 262, 988–990
- based IE, 1478
- in computational biology, 1663–1667
- methods, 1885, 3041

Magnetic disk, 866–867
Magnitude table, 2908
MAID disk, 1697
Mainframe, 2511
Main memory and cache optimization, 128–129
Maintenance of recursive views, 1674–1679
Majority quorum system, 2314
Management of physical and logical storage resources, 2827

Management policies, 134
Management virtualization, 812
Managing compressed structured text, 1679–1684
Mandatory access control (MAC) models, 1684–1685, 2448, 2450
MANET Databases, 1685–1690
Map operations, 2719–2724
Mapper, 1706
Mapping between the XML view and the underlying data, 3656–3657
Mappings, 1760–1763
MapQuest, 1232
Margin, 262
Markup language, 1696
Mashup, 1696–1697
Massive array of Idle disks (MAID), 1697
Massively parallel processor (MPP), 2638
Master-slave replication, 1738
Matching type, 1371, 1372, 2561–2566
Materialization, 2117
Materialization of dynamically generated HTML fragments, 3524
Materialized views, 1670–1673, 3324, 3325, 3328–3331, 3340
Materialized web views, 3524
Mathematical dimension reduction technique, 1784
Mathematical model of approximate reasoning, 120
Mathematical model of meaning, 1215
Matrix masking, 1698
Maximal patterns, 1699–1702
Maximal set of tuples, 92
Maximum distance to average vector (MDAV), 1736
MDS models, 1784
MDX, 1952, 1953
Mean average precision (MAP), 192–193, 1691–1692
Meaning, 3051–3055
Measure, 1703–1704, 2665–2666, 2779, 2780
Mechanism for information hiding, 2789
Media content analysis, 429–431
Media content annotation, 429, 430
Media (crash) recovery, 1644
Media production, 429–431
Mediation, 1704–1706
Mediator, 720, 1704–1706
Medical record, 968–970
Membership degree, 1209–1210
Membership function, 1209
Membership query, 1436, 1438, 1707
Memory
– hierarchy, 1707–1713
– locality, 1713–1714
Mergepurge, 780
Message, 1716
– broker, 2212–2214
– router, 2209
– translation, 1064
Message authentication code (MAC), 1287, 1288, 1715–1716
Message-oriented middleware (MOM), 47, 1716
Message systems, 1716
Messaging queueing system, 719, 721
Meta-class, 2918
Metadata, 215–222, 401, 939, 979, 980, 1249–1251, 1362–1368, 1717–1721, 1724–1727, 2497, 3286–3292
– interchange, 401–404
– management, 215–219, 1718–1720
– repository, 570, 571, 670–672, 674
– standards, 215–219, 673–674
Metadata encoding and transmission standard (METS), 1636–1637
Meta data interchange specification (MDIS)
– interchange, 1717–1718
Metadata registry (MDR), 1724–1727
Meta model, 1719, 1720
Meta object facility (MOF)
– interchange, 1722–1723
– metadata, 1722–1723
Metaphor, 3387–3388
Metasearch, 1730–1734
Metasearch engines, 1730–1734
Metasearcher, 2520
Method of concurrency control, 2672
Methods for video shot-cut detection, 3309
Methods of data analysis, 2691
Metric, 1734–1735
– index, 1439
– space, 1451–1454, 1734–1735
MGED ontology, 1960, 1962
Microaggregation, 1736
Microarchitecture optmizations, 129
Microarray gene expression data, 107
Microbenchmark, 1737
Microdata, 1735
– masking, 1474
– protection, 1473, 1474
– rounding, 1737–1738
Middleware, 1705, 1706
– systems, 2408, 2409
– technologies, 501, 1899
Middleware support for database replication and caching, 1738–1742
Middleware support for precise failure semantics
– consistency, 1743, 1746, 1747
– failure(s), 1743–1748
– linearizability, 1743
– serializability, 1743
– transactions, 1743, 1747, 1748
Minimal key, 1587, 1588
Minimum bounding rectangle (MBR), 2453–2456
Minimum confidence, 140
Minimum support, 140
Minkowski-form distance, 1381
Min-wise hashing, 2845
Mirrored disk, 2359
Mirroring, 1420
Misuse detection, 1569
MLS systems, 2540
Mobile ad-hoc sensor network (MANET), 1685–1690
Mobile data, 428, 429
Mobile database, 1751
Mobile devices, 1751
Mobile e-Commerce transaction, 960
Mobile interfaces, 1751–1754
Mobile sensor network (MSN) data management, 1755–1759
Modality, 1838, 1840–1842
Model-based querying, 1764–1768
Modeling, 1777–1783
Model management
– mappings, 1760–1763
– meta data, 1760
– meta model, 1760–1763
– schema integration, 1760–1762
– schema matching, 1760–1762
Models of information foraging, 1485, 1489
Modification, 98–99
MOF, 1729
Momolingual dictionary, 966
MonetDB, 1669
Monitoring data streams, 492
Monolithic architecture, 3014, 3015
Monotone constraints, 1769
Mosaicplots, 1868, 1869
MOST model, 1772, 1774
Moving object(s), 1458–1462, 1770, 2682, 2683, 2685
Moving Picture Expert Group (MPEG), 3287, 3290–3292, 3297–3299, 3301, 3302
Moving point, 1770, 1771, 1773–1775, 2730–2731
Moving region, 1770, 1771, 1773, 1775, 2730–2731
MPEG–7, 1349–1353
Multidimensional, 1950–1952
– cube, 538
– data, 1776–1777, 2253–2254, 2679
– data formats, 1776–1777, 2024
– data warehouse, 664, 665
– indexing scheme, 1279
– information, 789
– model, 2780, 3526, 3527

– modeling, 1777–1783, 2024
– points, 2703–2704
– range query, 2325
– temporal aggregation, 2925
Multidimensional scaling (MDS), 840, 842, 1784
Multigranulity locking, 1642
Multi-label classification, 1664
Multi-layered architecture, 1862–1864
Multi-level grid file, 1281
Multi-level recovery, 1784–1788
Multilevel secure database management systems (MLS/DBMS), 1789–1791
Multilevel secure (MLS) databases, 2540–2543
Multilevel secure relational data model, 1791
Multilevel transaction protocols, 273, 274, 1792–1796
Multi-master replication, 1738, 1740
Multimedia, 1825–1829
– asset, 1825, 1826
– data, 1800–1804
– alphanumeric, 1798–1800
– audio, 1797–1800
– buffering, 1800–1804
– querying, 1808–1814
– storage, 1814–1817
– video, 1797–1800
– database(s), 2244
– audio, 1818, 1819
– image, 1818, 1819
– video, 1818–1820
– documents, 1821, 1824
– metadata, 1825–1829
– mining, 595, 596
– objects, 1830
– presentation databases, 1829–1831
– presentations, 1829–1831
– resource scheduling, 1832–1835
– retrieval, 837
– systems, 1834
Multimedia data indexing
– metric indexes, 1807
Multimedia information retrieval (MIR) model, 528–532, 1820–1824, 2566–2567, 2570
Multimedia retrieval evaluation
– laboratory test, 1836
– operational test, 1836
– precision, 1836, 1837
– recall, 1836
Multimodal, 1838–1843
– interfaces, 958, 1838–1843
– streams, 3304, 3305
– systems, 1842
Multimodality based approaches, 3304
Multi-pathing, 1659, 1843
Multi-pathing software, 1843
Multi-perception, 1844
Multiple class labels, 1885
Multiple client-server architecture, 343
Multiple CQ scheduling, 2476
Multiple distance learning, 183
Multiple imputation, 2899
Multiple versions of a database schema, 2499
Multi-processor cache (MPC) machine, 714
Multiprocessor
– database management, 2026–2029
– data placement, 2024–2026
Multi-query optimization, 1849–1852, 3609
Multi-relational data mining, 596
Multi-representation, 1845–1847
Multi-resolution, 1853–1857
– models, 1853, 1857
– terrain modeling, 1853–1857
Multi-scale, 1844–1848
Multi-step query processing, 1860–1862
Multi-stream query, 2475
Multi target applications, 1753–1754
Multi-tier architecture, 1862–1864
Multivalued dependency (MVD), 1581, 1865
Multi-version concurrency control, 2659, 2660
Multi-version databases, 1870–1872
Multi-version schedule, 1870, 1871
Multi-version serializability, 502–504
– and concurrency control, 1870–1872
Multi-version serialization graph (MVSG), 2662
Music metadata, 157–160

N

Naive Bayes, 207
Naïve document representation, 943–945
Naive tables, 1407–1410, 1875–1876
n-Ary relations, 2372
National Institute of Standards and Technology (NIST), 2776
Natural human-computer interface (NHCI), 1880–1884
Natural interaction, 1880–1884
Natural interface systems, 1881–1883
Natural join, 1581
Natural language generation (NLG), 3048, 3049
Natural language information analysis method (NIAM), 1941–1946
Natural language processing (NLP), 234, 1564, 3072, 3074, 3075
Navigating and searching in compressed form, 1681
Navigation, 941, 1928
– space, 1501–1502
– tasks, 1501, 1502
Nearest neighbor classification, 1885–1889
Nearest neighbor classifiers, 341
Nearest neighbor (NN) of a query point, 1891
Nearest neighbor query, 1890
Nearest neighbor query in spatio-temporal databases, 1891–1894
Negative Tuple, 3535, 3536
Negotiation based methods, 1688
Nested loop join, 1026, 1895
Nested loop join using index, 1427, 1428
Nested transaction, 1222, 1896–1898
Network attached secure device (NASD), 1899–1900
Network attached storage (NAS), 1900, 2798–2800, 2824, 2826
Network common data form (NetCDF), 1777
Network data model, 715
Network distance, 2442–2446
Network file system (NFS), 1900
Network formation and communication, 1539
Network model, 1901, 1904, 1905
Network Voronoi, 2443, 2445
Network with blocked I/O services, 2797
Neural networks, 1906–1909
Neurons, 1906–1908
Newspapers, 978–981
NEXI, 1545, 1876–1879
NFS, 2824, 2826, 2827
NiagaraCQ Project, 1634
NIST text retrieval conference (TREC), 1692
Node in the edge of the event processing network, 1058
Node in the event processing network, 1052
Noise addition, 1911
Noise tolerant support, 121–122
Non-blocking concurrency, 1010
Non-clustering index, 1434, 2522
Non-confidential outcome attribute, 1735
Non-copyrightable data, 1611, 1612
Non-linear content, 1825
Non-metric temporal constraints, 2225
Non-parametric data reduction (NDR), 2044
Non-perturbative masking, 1474, 1912–1913
Non-pipelineable operator, 2794
Non-profit trade association, 2815
Non-redundant rules, 365–368
Non-schematic integration, 1497
Nonsequenced, 1913–1915

Non-standard database, 1860, 1861
Nontemporal, 1914, 1915
Normal form, 1915–1920
Normalization, 709, 1010, 1917–1920, 2498
Normalized cut criterion, 2749
Normalized schema, 2498
Now relative, 1922–1924
n-tier architecture, 1862–1864
Null values, 1405–1408, 1876
Numerical fact, 1703–1704

O

OAI, 2520, 2521
OAI-PMH, 2520
OAIS model, 132
OASIS, 1927
Object constraint language (OCL), 1729, 1927–1928
Object database (ODB), 1929–1935
Object data model(s), 411, 1936
Object identification, 2556, 2557
Object identifier, 1935–1936
Object Modeling Technique (OMT), 3232
Object-model transactions, 1792–1796
Object-oriented database (OODB), 1929–1933, 2771, 2772
Object-oriented encapsulation, 105
Object-oriented programming languages, 1553–1555
Object query language (OQL), 2003–2004
Object recognition, 1936–1939
Object-relational database (ORDB), 1929–1931, 1933, 1934, 2293–2296
Object-role modeling (ORM), 1941–1946
Object storage device (OSD), 1552
Observation, 359–360
Observe-predict-react, 2552
ODYSSEY, 3132
Office automation, 997
Office information systems, 290
Offline View Maintenance, 3327
OIL, 2008
Okapi BM25, 255
OMG data dissemination service, 2210
On-chip parallelism, 129–131
One-copy serializability, 1947–1948, 2397
One-pass algorithm, 1948–1949
Online algorithms, 741–744
Online analytic(al) processing (OLAP) databases, 539, 594, 1949–1953, 2297, 2298, 2300, 2755, 2758, 2880, 2883
Online communities, 2667
Online recovery, 1954–1959
On-line transaction processing (OLTP), 2297–2299
Online View Maintenance, 3327–3328
Ontologies, 216–218, 222, 360, 1546–1550, 1613–1619, 1960–1965, 2579–2582, 3472
– definition language, 1973, 1974, 1976
– elicitation, 1966–1971
– engineering, 1965, 1972–1973
– life cycle, 1972
– matching, 2561–2562, 2565
– visual querying, 1973–1977
OO7 Benchmark, 99
Open Archives Inititiative, 2519
Open database connectivity (ODBC), 449
Open EHR reference information model, 349
Open nested transaction, 1792–1796, 1978–1980
Operator failures, 100
Operator-level parallelism, 1981–1985
Opinion mining, 1986–1990
Optical storage, 2798, 2799
Optimal foraging theory, 1485, 1487
Optimal join order, 1583
Optimal location, 2419–2943
Optimal microaggregation, 1736
Optimal partitioning, 388
Optimal sequenced route (OSR) query, 2443–2444
Optimistic concurrency control (OCC), 2391, 3148, 3149
Optimistic replication, 1991–1995
Optimization, 47, 2550, 2552–2553
– of multiple queries, 1849
– problems, 790–793
– techniques for OQL, 2003
– and tuning in data warehouses, 1995–2002
OQL expressions, 2003
Oracle, 3162
Oracle DBMS, 3120
Orchestration, 2004–2005
Order relation, 1063
Order statistics of data, 2235
Organizational links, 941
Organizational theory, 1661
Orientation and directional relationships, 305–308
OR-join, 2006
OR-split, 2006, 2752
OSQL data model, 2007
Outer join, 1581, 1670–1673
Outlier detection, 88–90
Out-Of-Vocabulary (OOV), 523
Overflow, Hash-key, 1289–1290
Overlap, 2187–2191
Overlay networks, 2008, 2056–2060, 2081, 2576–2579
Ownership and transfer of copyright, 497–498

P

Page-level recovery, 1785, 1786
Page-oriented representation, 943, 945
PageRank, 3484, 3485
Page replacement policy, 286
Panning, 3684–3686, 3688
Parallel and distributed data warehouses, 2012–2018
Parallel composition, 419, 420
Parallel coordinate plots, 1866, 1867, 1869
Parallel coordinates, 2018–2023
Parallel databases, 2264–2268
Parallel database systems, 2036
Parallel data warehouse, 2012–2018
Parallel DBMS, 2638
Parallel hash join, 2029–2030
Parallel join algorithms, 2029–2030
Parallel merge join, 2029–2030
Parallel nested loops join, 2029–2030
Parallel query execution algorithms, 2030–2035, 2268
Parallel query optimization, 2035–2037
Parallel query processing, 2038–2040
Parallel SCSI, 2824–2826
Parallel versions of the traditional serial join algorithms, 2029
Parametric data reduction (PDR) techniques, 2044–2045
Parents, 1308–1309
Partially ordered graph, 1044
Partially ordered set of events, 1034
Partial order, 1187–1190
Partial replication, 188–192, 2045–2046
Participatory design, 1329
Partitional Clustering, 933, 934
Partition-based spatial merge join, 2710
Partitioning, 2030–2035
Partition skew, 635
Passage retrieval, 2867, 2868
Passive standby, 1110–1114
Password-based authentication, 177–179
Pathfinder, 3671–3673, 3675
Path index, 2857, 3586, 3588–3590
Patient registry, 346, 348
PATRICIA tree
Pattern based clustering, 2873–2874
Pattern detection, 1046–1048
Pattern matching based IE, 1477–1478
Pattern relaxation, 3638
Patterns as queries, 2433
Patterns of events, 1035
Pattern types, 1046, 1047
Peer data management system, 2863
Peers, 2081, 2082
Peer-to-peer (P2P), 2061–2064, 2415, 2578, 2863–2866
– computing paradigm, 2066, 2082

– data integration, 2056, 2065–2069
– storage systems, 2060, 2075–2077, 2080–2082
– systems, 875, 876, 2008, 3241–3243
– transaction, 3241–3243
– web search, 2082–2085
Perception, 3395–3399
Perception-based logic deduction, 120
Perceptual similarity, 1116
Performance differences, 194
Performance enhancing technique, 2116
Performance monitoring tools, 2093–2094
Periodic events, 3004, 3007
Period-stamped temporal models, 2094–2098
Permission-role assignment (PRA97) model, 57
Persistent stored modules (PSM), 2755, 2759
Persistent uniform resource locator (PURL), 2429
Personal autonomy, 2136
Personal digital assistants (PDAs), 1751–1753
Personal information protection and electronic documents act (PIPEDA), 2140
Personalization, 981, 2100–2102
Personalized user interfaces, 46–47
Personalized web search, 2099–2103
Perturbative masking, 1474
Petri nets, 3551
– concurrency, 2104
– firing rule, 2104, 2105, 2107
– places, 2104, 2105
– transitions, 2104–2107
– workflow management, 2104, 2107
P/FDM, 2011–2012
PFL, 1201, 1203, 1204
Pharmacogenomics, 1403
2 Phase commit (2PC), 3164–3166
Physical clock, 2108
Physical database design for relational databases, 2108–2113
Physical data independence, 3340
Physical layer tuning, 2114–2115
Physical operators, 1024, 1025, 1575, 1576
Physical placement of data, 2135
Physical schema, 708, 709
Physical storage devices, 2830
Physical volume, 3438
Physical WORM, 3570
Pipeline, 2116–2117
Pipelined parallelism, 1566
Pipelined query execution, 1025
Pipeline stage(s), 2116
Pipelining, 2031–2033, 2035, 2116, 2117
Pivoted normalization, 940–941
Pixel-oriented visualization techniques, 789–794, 3355
Pixels, 1116
Point-in-time copy (PiT copy), 2118–2119
Point-To-Point mediation, 1706
Polytransaction(s), 2123–2124
POLYVRT, 3134–3135
Portability and accountability act, 2140
Portals, 1696–1697
Position snapping, 1692–1695
Positive relational algebra, 2124–2125
Possible answers
– certain answers, 2125
– conditional tables, 2125
– incomplete information, 2125
– naive tables, 2125
– null values, 2125
Possible worlds, 2151–2153
Post-competition analysis, 194
Post-randomization method (PRAM), 2126
Post-triggers, 738
Power modes of modern disk drives, 868
Power of first-order (FO) logic, 1081–1083
Power-of-two choices, 1627
Powerset of items, 2876
Practical schema versioning, 2500
PRAM Matrix, 2126
Precedes relation, 1035
Precision, 192–193, 981–982, 1522, 2126–2127, 2348, 2386, 2776–2778
Precision-oriented effectiveness measures
– evaluation, 2128, 2129
– precision-oriented evaluation, 193, 266, 1703, 2128, 2129, 2453, 2876
Precision-recall, 1016, 1022
Precision-recall curve, 981–982
Predicate window, 3533, 3535–3536
Predictive approach, 279–281
Preimage resistance property, 1287
Pre-processing of text, 1529
Present, 1920, 1921, 1923
Presentation
– layer, 2511
– techniques, 654, 655
Presenting structured text retrieval results
– result presentation, 2130
– retrieval task, 2130–2134
Pre-triggers, 738
Primary (clustering) index, 2523
Primary index, 2135
Primary key, 1587, 2756, 2758, 2759
Primary memory, 1668–1669
Primitive event, 1045
Principal component analysis (PCA), 838, 2136
Privacy, 91, 1472, 2136–2137, 2140, 2142, 2783, 2784, 2788, 2789
Privacy-enhancing technologies (PETs), 2142–2147
Privacy metrics, 2137–2139
Privacy policy, 2140–2142
Privacy-preserving data mining, 2147–2150, 2535, 2537
Privacy-related legislative provisions, 601
Proactive functionality, 1052
Probabilistic counting with stochastic averaging (PCSA), 1145
Probabilistic data, 2150–2155
Probabilistic databases, 3227, 3230, 3478
Probabilistic disclosure risk, 2513
Probabilistic inference, 152
Probabilistic latent semantic indexing, 3076, 3078
Probabilistic model(s), 1524, 1527, 1764, 1765, 1808, 1813, 3058
Probabilistic range query (PRQ), 2161, 2162
Probabilistic record linkage, 2353
Probabilistic retrieval, 2649–2650
Probabilistic retrieval models, 1130, 2156–2160
Probabilistic spatial query, 2160–2164
Probabilistic spatio-temporal reasoning, 2168
Probabilistic temporal databases, 2165–2168
Probability driven caching (PDC), 3502
Probability ranking principle, 2168–2169
Probe phase, 1288, 1289
Process definition, 3545
Process evolution in workflow management systems, 3540, 3541
Processing of continuous data streams, 2847
Processing of queries, 1529
Process life cycle, 2170–2171
Process manager, 755
Process mining, 3551
– conformance checking, 2172
– event logs, 2171, 2172
– process discovery, 2172
Process optimization
– business processes, 2173–2177
– quality of service, 2173, 2177
– web service composition, 2173–2177
– workflow, 2173, 2177
Process-oriented models, 616, 618–619
Processor-per-disk (PPD) machine, 714
Process structure of a DBMS, 2178–2186
Production workflows, 3548
Programming in the large, 288
Projected clustering, 2873–2875
Project indices, 1454–1456
Projection, 2196–2197, 2737
Proof system, 1140, 1141

Proof-theoretic approach, 752
Propagation, 64, 65, 70
Propagation-based structured text retrieval, 2197–2201
Propositional logic, 1140
Protection of the database right, 1014
Protein-protein interaction graph, 220
Provenance, 2202, 2827, 2829
  – of an output data, 2202
  – of scientific databases, 2202–2206
p-Sensitive *k*-anonymity, 1585
Pseudonymity, 2142, 2143, 2207
Pseudonyms, 2207
Pseudo-relevance feedback, 2382
p-stable distributions, 2843–2845
Publication data model, 464, 465
Publication-event processing with complex predicates, 2070, 2071
Public-key cryptography, 142
Public-key encryption, 142, 572, 573, 1320, 2535–2536
Publish/subscribe, 464–466, 2208–2211, 2780–2782, 2871, 2872, 3608–3612, 3634
  – message broker, 2209
  – over streams, 2211–2215

Q

Quadtrees, 1300–1302, 1304, 2219–2224, 2703–2706
Qualitative data, 3416
Qualitative relations between time intervals, 78–79
Qualitative spatial relationships (QSR), 306
Qualitative temporal reasoning, 2225–2228
Qualitative temporal relationship, 2377
Quality and trust of information content and credentialing
  – accuracy, 2229
  – completeness, 2229
  – credentialing, 2229
  – transparency, 2229
Quality metrics, 2230
Quality of data, 803–807
Quality of retrieval results, 3493
Quality of schema, 804, 806, 807
Quantification of variables, 1140
Quantiles, 513
Quantiles on streams, 2235–2239
Quantitative association rules
  – distributional rules, 2240, 2242
  – frequent rules, 2240, 2241
  – special type of association rules, 2240
Quantitative data, 2024, 3437–3438
Quantitative techniques, 1080
Quantitative temporal relationship, 2377
Quantum mechanics, 3260
Quasi-copy, 3454
Quasi-identifier, 1735
Quasi serializability, 504–505
QUEL
  – relational database query language, 2244
  – tuple relational calculus language, 2244
Query, 368–373
  – camouflage, 1473, 1474
  – capability, 2309
  – compiler, 2288, 2290
  – concept learning, 2567–2568
  – containment, 2249–2252
    – aggregation, 61–62
    – and equivalence, 93
  – equivalence, 2249–2251
  – execution engine, 2117, 2282
  – expansion, 3492
    – automated query expansion (AOE), 2258–2259
    – blind feedback (BF), 2256
    – manual query expansion, 2258
    – pseudo-relevance feedback, 2255–2257
    – relevance feedback (RF), 2254–2257
  – expressiveness
    – aggregation, 59–63
  – footprint, 1253, 1254
  – generation models, 2169
  – graph, 2290, 2291
  – heterogeneity, 2309–2311
  – language for relations, 2372
  – load balancing in parallel database systems, 2264–2268
  – minimization, 2251
  – optimization, 2272–2281, 2304, 2900, 2902, 2905
  – optimizer, 1583
  – parallelism, 1567–1568
  – perturbation, 1473
  – plan, 2282
  – plan explainer, 2093
  – reconciliation, 2310
  – restriction, 1473, 1474
  – routing tree, 1757
  – shipping, 2864–2866
  – translation, 523–526, 2309–2312
  – over data streams, 3533, 3536
Queryable database protection, 1472–1473
Query-by-example, 1451
Query by image context (QBIC) system, 1354
Query-dependent hyperlink features, 3499
Query-document matching functions, 1519, 1523
Query execution plan (QEP), 506, 2293, 2294, 2595, 2598
Querying semi-structured data, 2867–2868
Querying temporal indeterminacy, 2975
Query-language model, 1522
Query languages, 1–5, 406–411, 710–713, 1877, 1879, 2047–2057, 2762–2768, 2771–2772, 2958, 3659, 3660
  – data manipulation statements, 2261
  – and evaluation techniques for biological sequence data
    – biological sequence data, 2261–2264
    – sequence similarity, 2262
  – for the life sciences
    – scientific objects, 2264, 2267
    – a scientific query language, 2264, 2266
  – specialized programming language, 2261
Query processing, 113–119, 912–916, 1572–1574, 2278–2280, 2288–2293, 2301–2306, 2675–2680
Query processing and optimization in object relational databases
  – four phases of query processing, 2294
  – object relational databases, 2293–2296
  – query execution plan (QEP), 2293, 2294
Query processing in data warehouses
  – materialized views, 2298, 2299
  – multi-query optimization, 2299
Query processor
  – query executor, 2307, 2308
  – query parser, 2307
  – query rewriter, 2307, 2308
Query rewriting, 2273, 2438, 2440, 2441, 3491–3493
  – the goal of query rewriting, 2308
  – query rewriter, 2308
  – responsibilities of the query rewriter, 2308
Query term frequency (qtf), 2169
Question answering (QA), 3485–3490
Queues, 1716
Quilt, 3637
Quorum system, 2313–2317

R

RAID (Redundant array of inexpensive disks), 2823, 2824, 3438
Random access memory (RAM), 1668–1669
Randomization methods to ensure data privacy
  – input randomization techniques, 2320
  – local randomization techniques, 2319, 2320

– output randomization techniques, 2320
– privacy preserving randomization method, 2319
Randomized algorithms, 2770–2771
Randomized summary, 80
Range partitioning, 2025, 2026
Range predicate, 1427
Range query, 1437, 1438, 2324–2325
Ranked XML processing, 2325–2331
Ranking, 2647–2651
Ranking function, 255
Ranking model, 1371–1373
Rank swapping, 620–621
Rapid serial visual presentation (RSVP), 267
Raw storage, 2796–2797
Rayleigh quotient, 2750, 2751
RBAC96, 2448–2451
RBAC models, 1245
RDF, 1724, 2425–2428, 2579
RDF graph, 2424
RDFS, 2425–2428
RDF schema, 2424
Read/write model, 3158–3161
Real-time analytics, 25
Real-time applications, 2344, 2347
Real-time data warehousing, 21–26
Real-time transaction processing, 2344–2348
Recall, 981–982, 1522, 2126–2127, 2348, 2776–2778
Receiver anonymity, 91
Reconciliation, 1991–1995
Record extraction, 397–400
Record linkage, 2353–2354
Record matching, 780–784, 2354–2357
Records management, 997
Record trail, 608
Recover data from backups, 478
Recovery, 917–919, 3150, 3151
Recovery component, 1645
Recovery manager, 1645
Recovery point objective (RPO), 848
Recursive query(ies), 2304, 2758
Recursive view definition, 1674–1676
Reducer and full reducer, 2596–2597
Redundancy, 2075–2081
Redundant arrays of independent disks (RAID), 2359
Reference information model (RIM), 360
Reference knowledge, 2360
Reference knowledge base, 2360
Reference monitor, 9, 10
Reference reconciliation, 780
Referential transparency, 1201
Region, 2360–2363
Region growing, 1389, 1390, 1392, 1394
Registry metamodel, 1725–1726
Regression coefficients, 1622
Regression tree, 2469
Regular expression, 1264–1265
Regular square grids (RSGs), 817, 818
Relational algebra, 308, 811, 2124–2125, 2196, 2369–2370, 2547, 3239, 3240
Relational-algebra incremental algorithms, 1415, 1416
Relational calculus, 2370–2371
Relational calculus and algebra, 406, 407
Relational database, 265–266
Relational database scheme, 265–266
Relational data model, 715, 2753, 2754, 2759
Relational expression, 2967
Relational Interval Tree (RI-tree), 3256
Relational model, 2372–2375
Relational operator, 1575, 1576, 2282
Relational query processor, 755
Relational storage for XML documents, 3631
Relation detection and recognition, 1477
Relationship type, 1915–1917, 1940–1941
Relative atomicity, 2627, 2630–2631
Relative error, 83
Relaxed atomicity or extended atomicity, 145
Relax NG, 3650–3652
Relevance, 1652–1656, 2648–2650
Relevance dimension, 2747, 2748
Relevance feedback, 1384–1388, 2378–2382, 3493–3497
Relevance propagation method, 2197
Relevance score of an element, 2197, 2199–2201
Reliability, 350
Relicated databases, 1947
Relicated history (RH), 1947, 1948
Remote invocation, 2415
Renaming, 811, 3239, 3240
Repair, 722–723
Replacement policy, 283–285
Replica control, 626–628, 1947, 1948, 1957, 1958, 2383–2387, 3144–3147
Replica control component of the replicated system, 2391
Replica placement, 3443
Replicated database concurrency control, 2390–2391
Replicated databases, 2390–2391
Replicated MDS, 1784
Replication, 1071, 1991–1995, 2045–2046, 2391–2392, 2397–2402
– for high availability, 2397–2402
– in multi-tier architectures, 2408–2413
– for performance, 3442
– techniques, 2392
Repository, 356–359
Representation information, 132
Representation of medical evidence, 1072
Reputation, 2414–2415
Reservoir sampling, 2467, 2838–2841
Residuated lattice, 2418–2419
Resolution, 1374, 1375, 1378
Resource
– allocation, 2419–2423
– classification, 217
– composition, 221, 222
Resource description framework (RDF), 2423–2425
Resource discovery, 218
Resource patterns, 3557, 3558
Respondent privacy, 1472
Restore, 200–201
Restriction on free variables, 3016
Resubstitution validation, 534
Result presentation, 3506
Retrieval of multiple streams
– on multiple disks, 486–487
– on a single disk, 486
Retrieval of single stream
– on multiple disks, 485–486
– on a single disk, 485
Retrieval process, 1528, 1529
Retrieval status value (RSV), 1521
Retrieval techniques for handwritten data, 973
Retrospective context, 2431–2432
Retrospective event processing, 2431–2433
Reverse engineering, 723–728
Reverse engineering tools, 727–728
Rewriting queries using views, 2438–2441
Rewritings, 93–97
Rhetorical relationships, 1331
Road networks, 1692–1695, 2442–2446
Robertson and Sparck-Jones (RSJ) model, 2169
Rocchio's formula, 2447
Rocchio's relevance feedback algorithm, 2447
Role, 46
Role based access control (RBAC), 91–92, 2447–2452
Role hierarchy, 2448–2450
Role schema, 1245
Root, children, 1308
Rough set theory (RST), 761–764
Rounding, 1737–1738
Rounding set, 1737
Round-robin partitioning, 2025
Routing vehicle in event processing network, 1034
ROWA, 928
Row-level triggers, 738

R-precision, 195
RSA, 832, 833
R-tree joins, 2709
R-trees, 241, 368–373, 1439–1441, 1891, 1893, 1894, 2453–2459, 2703–2705, 2707
Rule action execution, 959
Rule based data grids, 134
Rule-based information extraction, 3474
Rule-based optimizer, 1996
Rule evaluation, 29–32
Rule induction, 2459–2461
Rule ranking, 2459–2461
Russian calendric system, 305

S

Safe plan, 2154, 2155
Safe query, 2151–2153, 2463
Safety and Domain Independence, 2463–2466
Saga(s), 1222, 1979, 2466–2467, 2824, 2826
- – compensation, 2466
- – transaction, 2466–2467

Sample and count, 2839
Sampling, 630–633, 1912, 2467
- – methods, 631–632
- – with replacement, 263
- – synopsis, 631–633

SAN File System, 2467–2468
SAN islands, 1575
SAN management in multi-vendor environments, 2806
Sarbanes-Oxley Act, 2364
SAS, 2824–2826
SATA, 2824–2826
Satisfiability problem, 1397, 1398
SAX/DOM, 3598–3600
SBQL, 2771–2772
SCA assembly model, 2632
SCA component, 2632
Scalability, 133, 2045, 2403–2407
Scalable Decision Tree Construction, classification, 2469–2473
Scale-out, 2403–2407
Scale-up, 2031, 2032
Scatterplots, 1335, 1338
Scene(s), 3309, 3311
- – boundary detection, 3311–3313
- – detection, 3303–3308
- – grouping, 3281–3284

Scheduling, 719–721
Scheduling Strategies for Data Stream Processing, 2475–2479
Schema, 1940–1941, 2494–2497
- – changes, 2479, 2480
- – integration, 1760–1762, 2494, 2497
- – manipulation statement, 2831
- – mapping, 2494, 2497
- – matching, 1760–1762, 1973, 2494–2497, 2562
- – tuning, 2497–2499versioning, 2499–2502

Schema-driven storage, 3630
Schema evolution, 2479–2481
- – in workflow management systems, 3540, 3541

Schema mapping, 1492–1494, 2481–2493
- – semantics, 2482
- – types, 2483–2487

Science application, 748–750
Scientific data, 2506, 3225–3227, 3230
Scientific databases, 2202–2206, 2339, 2502–2505
Scientific query, 2264
Scientific query language, 2264, 2266
Scientific simulations, 733–738
Scientific workflows, 2507–2510
Scientific workflow tasks, 2507
Scope of rights, 1014
Scraper (-ing), 2511
SCSI, 2512
- – block level protocol, 1538
- – bus cables, 2812
- – initiators, 1537, 1538
- – interfaces, 846
- – protocol, 1537, 1658
- – storage devices, 846

SD-Rtree, 921, 923, 924
Searchable compressed format, 1442, 1447
Search advertising, 3458
Search algorithms, 739–741
Search engine metric, 2513–2518
- – BLEU score, 2514
- – precision, 2514–2518
- – recall, 2513–2517
- – TREC, 2514, 2516–2518

Search engines, 1251–1254, 1463–1466, 2513–2518, 3462, 3464
Search engine selection, 1730–1732, 1734
Search space, 739, 740
Search tasks, 3480
Secondary index, 2522–2528
Secondary storage, 1707
Second chance (replacement policy), 283
Second normal form (2NF), 2522
Second principal component, 2136
Secret-key encryption, 572, 2897
Secure database development, 2528–2534
Secure database development using patterns, 2530–2531
Secure data outsourcing, 2523–2528
Secure multiparty computation (SMC), 2148, 2149, 2535–2539
Secure transaction processing, 1789, 1791, 2540–2545
Security, 728–732
- – in data sources, 676–677
- – in data warehouse modeling, 677–678
- – in extraction-transformation-loading (ETL) processes, 677
- – mechanisms, 728, 731, 732
- – in OLAP tools, 678–679

Security requirement in secure data outsourcing, 2524–2525, 2528
Security services, 2546–2547
Security services (No. 1481), 2546–2547
Segmentation, 1389–1395
Selection, 2547
Selective dissemination of information (SDI), 1482
Selectivity, 2739–2741
Selectivity estimation, 2548, 2739–2741
Self describing, 2607
Self-describing data, 1265
Self-monitoring, 2552, 2554
Self-organizing map (SOM), 3356
Self-tuning histograms, 743
Semantic(s), 132, 2576–2579, 3017
- – atomicity, 2588–2590
- – data models, 1193, 1194, 1197, 2559–2561
- – dimensions, 473
- – distributed system, 982
- – gap, 3361
- – heterogeneity, 1496–1497, 2561
- – indexing, 429–431
- – information processing (SIP), 1821
- – integration, 2557–2558
- – labels, 149
- – map, 3124
- – matching, 2561–2566
- – model, 2559–2561

Semantic/syntactic information, 939
Semantic web, 1546, 1547, 1550, 2579–2587, 3076–3079, 3472
- – approach, 2580
- – query languages, 2583–2586

Semantic web services (SWS), 2586–2587, 3513–3519
Semi-automated web data extraction tools, 1283
Semijoin, 904, 906, 1581, 2592–2595
Semijoin program, 2595–2598
Semiology, 1335
Semiotics, 1334
Semi-streaming model, 1271–1274
Semi-structured, 1915, 2599–2600
- – data, 1198–1200, 2601–2610
- – query language (No. 339)
- – searching, 1876, 2191, 2192
- – text retrieval, 463, 2187

Semi-supervised clustering, 393–396, 2613

Semi-supervised learning, 393, 2613–2615
Sender anonymity, 91
Sender pseudonymity, 2207
Sensor databases, 107–110
Sensor data model, 489–490
Sensor network indexing, 636–637
Sensor networks, 574, 710–713, 1538–1540, 1542, 2278–2281, 2616–2619
– aggregation, 552–556
– approximation, 554–556
– continuous queries, 553, 555, 556
– model-based querying
– storage, 635–637
Sensor selection, 583
Sensory gap, 1115, 1116, 3361–3363
Sentence segmentation, 3072, 3073
Sentiment analysis, 1986–1990
SEQUEL, 2753, 2754, 2760
Sequence database, 2621–2623
Sequence matching, 837, 3315
Sequenced, 2619–2621
Sequential composition, 419, 420
Sequential I/O, 1333
Sequential pattern and sequential association rules, 140
Serializability, 1642, 2383, 2385, 2591, 2626–2631, 2761, 2762
– theory, 927–928
– graph, 1947, 1948
Server-side scripting languages, 954
Service component architecture (SCA), 2632–2633
Service composition, 1697
Service consumers, 2634, 2635
Service oriented architecture (SOAs), 328–329, 414, 997–999, 2004, 2633–2636
Service outsourcing, 2523
Service producers, 2634, 2635
Session beans, 1579
Session consistency, 2636
Session property, 2856, 3452, 3453
Set abstraction, 420–421
SGML, 3124
Shared-disk, 2637
– architecture, 2028, 2637
– file system, 2467, 2468
Shared-memory, 2638
Shared-memory architecture, 2028, 2638–2639
Shared nothing partitioning, 2109, 2112
Ship whole, 904–907
Short-range wireless communication protocols, 1751
Short running processes (microflows), 3166
Shot-cut detection, 3309–3311
Shot(s), 3308–3311, 3316–3320
– boundary detection, 467
– detection, 3281, 3282
Side-effect free translation of view update, 2639, 2641
Side-effect-free view updates, 2639–2642
Side-effects, 2639–2642
Signal detection theory, 2349
Signatures, 2642–2646
Signature unblinding, 252
SilkRoute, 3614, 3615
Similarity, 1451, 1452, 2647–2651
– function, 699
– search, 3115–3118
Simple metadata model, 947
Simplicial complex, 2651–2656
Simulation, 3525–3528, 3551
Single client-server architecture, 342
Single instruction multiple data (SIMD) instructions, 129
Single-link clustering, 3045
Single query optimization, 1849
Single-site illusion, 451
Single stream query, 2475, 2476
Singular value decomposition (SVD), 2045, 2657–2658
Situational applications, 593
Situation reinforcement, 2431, 2432
Sketch, 511–515, 2898, 2899
Sketching, 89
Skewed distribution, 634
Sliding *versus* predicate windows, 3533
Sliding windows, 137, 138, 2845–2846
SMI-S (Storage Management Initiative-Specification), 1552
Snake model, 1390–1393
Snapshot(s), 2383, 2385, 3328
– isolation, 2383, 2385
– reducibility, 2621, 2983
Snippet, 2664–2665
SOAP (Simple Object Access Protocol), 2666–2667, 3572
– messages, 2666–2667
– specification, 2666
SOA systems, 2634–2636
Social applications, 2667
Social network analysis (SNA), 2668, 2669, 2672
Social networks, 2415, 2667–2672
Social tagging, 1522
Soft hardware failures, 100
Software layer within a database management system, 2807
Software measurement, 2232
Software non-determinism, 100
Software transactional memory (STM), 2672–2673
Sort-Merge join, 1026, 1027, 2673–2674
Source-to-target dependency, 576
Space-filling curves (SFC), 2674–2680
Space partitioning index, 1439, 1440
Spam detection, 3520–3523
Sparse index, 1433, 1434
Spatial, 2725–2729
– access method, 368, 373, 2454, 2459
– anonymity, 2685–2690
– data, 3132
– data analysis, 2691–2695
– databases, 71–77, 368, 369, 372, 373, 1639, 2459
– data management system (SDMS), 3684, 3685
– data mining, 2691, 2692, 2695–2698
– data structures, 2219–2224
– data type, 2681–2683, 2685, 2698–2702
– *extenders*, 3142
– hash joins, 2710
– histogram, 2740
– hotspots, 2695, 2696, 2698
– indexing, 2224
– indexing techniques, 2702–2707
– join, 2707–2714
– join methods, 2708
– k-anonymity (SKA), 2686–2688, 2690
– locality, 1713–1714
– monitoring method, 479
– network, 2698
– network databases, 2714–2719
– objects, 2719, 2720, 3140, 3141
– operation(s), 2699–2701, 2719–2724
– outliers, 2695, 2696, 2698
– partition, 2699–2701
– pattern families, 2695
– projection, 2737
– queries, 3383, 3386
– range query, 2325
– SQL, 3142, 3143
– and spatio-temporal data models, 2681–2685
– uncertainty model, 2161, 2162
Spatio-temporal, 2725–2730, 2736–2738
– benchmarking, 2339–2343
– data, 421–429
– database(s)1448–1450, 1462
– data generator, 2339–2343
– data mining, 595
– data type(s), 1770–1773, 2682, 2683, 2730–2731
– graph, 3097
– networks, 3097, 3098
– predicate, 2684, 2685
– test datasets, 2340, 2342
Spatio-temporal distance join query (STDJQ), 2740, 2741

Spatio-temporal trajectories, 2742–2746
- moving point, 2742, 2743, 2746
- moving region, 2742

Spatiotemporal window query (STWQ), 2739–2740
SPC algebra, 449
SPCU-algebra, 2124–2125
Special case of range query, 1707
Specialization, 2746–2747, 3017
Specificity, 2747–2748
Spectral feature, 161
Speed-up, 2031, 2032, 2035
Spider, 1418
SPJRU-algebra, 2124–2125
Split
- attribute and split point selection, 767–768
- mirror, 2118
- policies, 2456
- transaction, 2752û2753

Spoiler, 963
SQL, 2753–2760, 2762–2768, 3018–3023
SQL isolation levels, 2761–2762
SQL standard, 1925
SQL/temporal, 3192, 3193, 3197
SQL/XML, 3606, 3614, 3615
SQuery languages, 2583–2586
SRAM, 2195, 2196
S-reducibility, 2620–2621
SRU, 2519, 2521
SSD, 2799, 2800
Stack-based architecture, 2772
Staged database, 2773–2776
Staleness, 2388, 2389
Standard effectiveness evaluation in IR, 2776
Standard effectiveness measures, 2776–2779
Standard generalized markup language (SGML), 1927
Standard storage management interface, 2806
Standardized metadata set, 947
Standards for obtaining copyright, 497
Stand-by mode, 2821, 2822
Star index, 2779
Star or snowflake schema, 2779
Star schema, 658, 661, 2014
State-based publish/subscribe, 2780–2782
Statechart, 3554, 3555
State diagram, 41, 42, 45
State equivalence, 1071
Stateful methods, 1688
Stateless methods, 1688
State-machine replication, 2392, 2393
Statement-level triggers, 738
Statement replication, 1740–1741
State transition, 355
Static dynamic caching (SDC), 3502
Static random access memory (SRAM), 1669
Stationary MSN, 1756
Statistical database, 2467
Statistical data management, 2782
Statistical disclosure control (SDC), 1472, 1473, 2357
Statistical disclosure limitation, 1698
Statistical disclosure limitation for data access, 2783–2789
Statistical estimation, 634
Statistical method, 263
Statistical test, 318–321
Statistics, 236–238
Statistics databases, 2333
Steganographic techniques, 2789
Steganography, 826, 2789–2790
Steganography *vs.* watermarking, 2789
Stemming, 1610, 2790–2793, 3492–3493
Stop-&-Go operator, 2794
Stoplists, 2794–2796
Stopwords, 2794, 2795
Storage access model, 2796–2797
Storage and management, 344–348
Storage area network (SAN), 2467–2468, 2797–2798
Storage consolidation, 2798
Storage device, 1574, 1575, 1900, 2798–2800
Storage grid, 2800–2802
Storage layer concepts, 2808
Storage management, 2802–2806
- implementation, 2804
- initiative-specification, 2806–2807
- technologies, 2802

Storage manager, 2807–2812
Storage mapping strategies, 3627
Storage network architecture (SAN), 1574, 1575, 2812–2815
Storage Networking Industry Association (SNIA), 1499, 1552, 2806, 2807, 2815–2816
Storage networks, 1552
Storage of Large Scale Multidimensional Data, 2816–2821
Storage power management, 2821–2823
Storage protection, 2823–2824
Storage protocol, 2824–2827
Storage resource management, 2827
Storage resource management software tools, 2827
Storage security, 2827–2829
Storage virtualization, 1659, 2830
- software, 1659
- technologies, 2830

Storage volume identifier, 1658–1659
Stored procedure, 2831
STORM, 2782
Story identification, 3281, 3283, 3285
Streambased *versus* operation-based windows, 3533
Stream ciphers, 572
Stream fault-tolerance, 1109–1114
Stream high availability, 1109–1114
Stream incremental evaluation, 3533
Streaming algorithm, 1948–1949
Streaming applications, 2847–2848
Stream mining, algorithms, 2831–2833
Stream-oriented Query Languages and Operators, 2848–2853
Stream-oriented representation, 943–944
Stream processing, 1633, 1635, 2837–2838
Stream processing enginer (SPE), 639–641
STREAM project, 1634
Stream query language, 2849
Stream query processing, 3533, 3536, 3537
Stream replication, 1110, 1113, 1114
Stream scheduling (session scheduling), 1832–1834
5S (Streams, Structures, Scenarios, Spaces, and Societies) framework, 268
String similarity, 2355, 2357
Structural relationship, 2376
Structural summaries, 606, 2858–2860
Structure, 2377, 2862, 3036, 3037
Structured document retrieval, 2867–2868
Structured overlays, 1628–1630
Structured Query Language, 2753
Structured text retrieval, 2867–2868
- models, 2868–2871
- system, 2130–2134

Structure indexing, 2857–2861
Study of signs, 1334
Subgraph isomorphism, 1170–1171
Subject space, 2871–2872
Subscription language, 465
Subscription processing with complex predicates, 2071
Subshot segmentation, 3281, 3284, 3285
Subtransactions, 1142, 2588–2590
Succinct constraints, 2876
Suffix, 2790–2793
- removal, 1610
- trees, 2876–2880, 3059–3061
- trie, 2876–2878

Summarizability, 2880–2883
Summarization, 2884–2889
- systems, 2884–2889
- techniques, 2884, 2885, 2888

Summary database, 2880
Summary graph, 2857
Summary statistics, 2882
Superkey, 1587, 1588

Supervised learning, 1664
Supply chain management process engineering, 296–297
Support, 124–126
 – vector machine, 2890–2892
 – vector regression, 2890
Support of a graph, 1171
SVM, 2890–2892
SVR, 2890
SWS framework, 2587
Symbolic representation, 2897
Symmetric multiprocessors (SMP), 2638
Synchronization protocols, 445
Synchronous propagation, 2856
Synopsis, 634, 2898–2899
Syntactic structures, 982, 983
Synthetic microdata, 1475, 2899–2900
System configuration, 2551–2552
System dynamics, 3527
System R, 2289, 2290, 2293, 2754, 2760, 2900–2905
System (crash) recovery, 1643–1644
Systems biology, 1266–1270
Systems biology ontology, 1960
Systems medicine, 360

T

Tabular data, 2908
Tabular data protection, 1472
Tabular storage of trees, 3630
Tags, 1696
Tape, 2798–2800
Task, 41
Taxonomy: biomedical health informatics, 2908–2911
TelegraphCQ project, 1634
Telic distinction in temporal databases, 2911–2914
Telos, 1728, 2914–2920
Temporal, 2736–2738, 2762–2768, 2932–2935
 – access control, 2920–2924
 – algebra basics, 2929–2930
 – algebras, 2929–2932
 – and derived data authorization model, 2920, 2922
 – aspect, 3253
 – assignment, 3008
 – authorization model, 2921–2922
 – coalescing, 2932–2935
 – compatibility, 2936–2945
 – conceptual models, 2940–2945
 – consistency, 2344, 2345, 2347
 – constraints on sets of time points, 2946
 – constraints on time intervals, 2946, 2947
 – constraints on time points, 2945, 2947
 – database(s), 99, 142–143, 241, 242, 2892, 2893, 2896, 2957–2960, 3028, 3258, 32359
 – database system architecture, 3013, 3015
 – data models, 329, 2098, 2119–2122
 – data semantics, 2941, 2942, 2944
 – dependencies (i.e. the title of the entry), 2960–2966
 – evolution, 1319
 – feature, 161
 – functions, 1915
 – granularity, 2962, 2965
 – homogeneity tuple timestamping, 2973
 – indeterminacy, 79, 2973–2982
 – index, 3167, 3254, 3258
 – integrity constraints, 2960, 2965, 2972–2986
 – join, 2982–2987
 – join evaluation, 2984, 2986.
 – join operators, 2982, 2985
 – locality, 1713–1714
 – logic, 2945–2948, 2958, 2976–2978
 – logical models, 2992–2996
 – normalization, 2965
 – object data models, 3000
 – object-oriented databases, 2998–3008
 – object query languages, 3002
 – periodicity, 3004–3008
 – periodic pattern of repetition, 3004
 – predicates, 1915
 – probabilistic aggregates, 2168
 – query languages, 2762–2768, 2958
 – relation, 243–244
 – relational calculus (TRC), 3015–3016
 – role based access control model, 2920–2922, 2924
 – SQL, 2762–2768
 – type, 2968
 – visual languages, 3027–3033
Temporal Cartesian product, 2983
Temporal query language extending the relational calculus, 3015
Tentative update, 1991–1994
Term dependence, 3036
Term-document matching functions, 929
Term-document model, 1522
Term frequency (tf), 2169, 3937–3942
Term frequency-inverse document frequency weighting scheme (tf*idf), 1529
Termination, 37
Terminator, 143
Terminology data model, 1002
Terminology systems, 1000, 1003
Term proximity, 3036–3037
Term weight, 3036
Terrain, 817–821
Terrain models, 818, 820, 821
Tertiary storage media, 2817–2820
Test collections, 3481, 3482
Test datasets, 2339–2343
Text analytics, 3473
Text categorization, 3041–3044
Text classification, 3044–3046
Text clustering, 3063
Text document formatting systems, 1696
Text generation, 3048–3051
Text indexing, 3055–3058, 3937–3942
Text indexing and retrieval, 3055–3058
Text mining, 235, 595, 693–695, 3061–3063, 3065–3068
Text preprocessing, 3062
Text processing, 979
Text retrieval, 3055–3058
Text segmentation, 397–400
Text streaming model (TSM), 3078–3079
 – bag of words model (BOW), 3078
 – vector space model (VSM), 3078
Text summarization, 3079–3083
 – an abstract-summary, 3080
 – an extract-summary, 3080
 – query-oriented summary, 3080
 – supervised methods, 3081–3082
 – unsupervised methods, 3080–3081
Text summarizer, 3063
Textual annotation, 180, 181
Textual summaries, 605–607, 2884
Text visualization, 3083–3085
 – information visualization, 3083
 – natural language processing, 3084, 3085
 – term frequency (TF), 3084
TF*IDF, 3085–3086
 – document length, 3086
 – inverse document frequency, 3085
Thematic map, 3086–3087
 – coropleth map, 3086–3087
 – dasymmetric map, 3087
 – dot map, 3087
 – isarithmic map, 3087
 – proportional symbol map, 3087
 – topographic map, 3087
Theory of indexability, 1223
Theta join, 1581
Third Normal Form (3NF), 3087–3088
Three-Dimensional GIS and Geological Applications, 3088–3091
Three-level memory hierarchy, 1707
Three-Phase Commit, 3091–3097
 – atomic commit protocols (ACPs), 3092, 3097
 – voting protocols, 3094, 3096

Thresholding, 1389, 1390
TIGER, 3132, 3133, 3135–3139
Tight coupling, 2211, 2212, 3097–3098
– loosely coupled approach, 3097
– strong dependency between software components, 3097
TIGRIS, 3132
TIMBER, 3637
Time, 2914, 2915, 2919, 2920, 2992–2996, 3018–3023
Time and information retrieval
– document exploration, 3099
– document retrieval, 3100–3101
– temporal expression, 3098–3102
– timeline, 3098–3102
Time-based profilers, 2093
Time-cognizant behavior, 2344
Time decay, 758–759
Time-dependent geometry, 1770
Time-dependent graphs, 3097
Time domain, 1161, 3103–3106
Time granularity, 2968, 2969, 2972, 2994, 2995
Time in Philosophical Logic
– A.N. Prior, 3108–3110
– branching time, 3108–3110
– J.E. McTaggart, 3108
– philosophical logic, 3107–3111
– temporal logic, 3108, 3109, 3111
– tense logic, 3109, 3111
– tenses, 3107–3110
Time interval, 3113
– convex subset, 3112
– duration, 3112, 3113
– SQL, 3113
– time duration, 3112, 3113
(convex) Time interval, 78–79
Time-line clock, 2108, 3120
– *physical clocks*, 3120
– *synchronization point*, 3120
Time period, 3113
Time period set, 2966
Time-referenced fact, 1
Time-references, 2957
Time series, 2245–2247, 2952, 3114–3118
Time slice, 2736–2738
Timeslice operator, 2659, 3455–3456
Time span
– data type, 3119
– duration, 3119
– interval, 3119
– time, 3119
Timestampe ordering (TO), 2541–2542
Timestamping, 2895
Timestamps, 2932–2935, 2966
TimesTen, 1669
TinyDB, 710–713
TinySQL, 711–713
$_T$j t_j, 477
t-norm (Triangular norm), 3177–3178
Tokenization, 3062
Top/bottom coding, 1912, 1913
Topic-based publish/subscribe, 3127–3128
– publish/subscribe, 3127–3128
Topic Detection and Tracking (TDT), 3121–3124
Topic map, 3124–3126
Top-k queries, peer-to-peer, 111–113
Top-κ query processing for XML data, 2325
Top-K selection queries on multimedia datasets
– multimedia, 3129–3131
– query processing, 3129–3131
– ranking, 3129, 3131
Top-k summary, 2898
Top-k XPath Query Processing
Topological data model, 3132–3139
Topological invariants, 843, 845
Topological predicate, 2701–2702
Topological relationships, 843–845, 3140–3143
Topology, 2652, 2653
Tours, 950, 951, 953
TPR-tree, 1458–1461
Tracking, 1693, 1695
Traditional concurrency control for replicated databases
– locking, 3146, 3149
– replication, 3144–3146, 3148
Traditional sequential algorithms with multiple threads, 2031
Trajectory, *moving point*, 3150
Transaction, 1091–1092, 2761, 2762, 3150–3151
Transactional ACID property, 144
Transactional interface, 2673
Transactional processes, 3166
Transactional storage manager, 755
Transaction chopping
– chopping, 3151–3152
– locking, 3151
– performance, 3151
– serializability, 3152
– transaction, 3151–3152
Transaction database (TDB), 2876
Transaction management, 3153–3156
– recovery, 3153–3156
Transaction manager (TxM), 3157
– ACID transactions, 3157
– active transactions, 3157
– state transition, 3157
Transaction models, 3158–3161
Transaction models-the read/write approach
– commutativity (of operations), 3160
Transaction models-the read/write approach
– schedule, 3158–3160
– transaction, 3158–3161
– transaction history, 3159
Transaction performance council (TPC), 99
Transaction processing, 3163
Transaction processing monitors (TPM), 3163
Transaction pseudonym, 2207
Transaction recovery, 1643
Transaction replication, 1740–1741
Transactions, 1142, 1947–1948, 1954–1958, 2626–2631
Transaction scheduling, 443–444, 3153
Transaction-time, 2892–2896, 3252
Transaction-time databases, 3167, 3169, 3254, 3255
Transaction-time indexing, 239, 240, 3167–3171
Transducer networks, 3635–3636
Transductive learning, 2613
Transformation and loading (Web ETL), 3471–3472
Transitive reduction, 1189
TREC (Text REtrieval Conference), 2776, 3041, 3042
TRECVid, 470, 472
Tree, 2599–2600
Tree-based indexing, 3172–3173
Tree-based indices, 2135, 2522
Tree-like indexing structure, 197
Tree-like models, 1853–1855
Treemaps, 3173–3177
– treemaps algorithm, 3174
– treemaps visualizations, 3174
Tree pattern query, 3637–3639
Tree quorums, 2314, 2315
Tree-structured data mining model, 2469
Tree structures, 2219
Trellis plots, 1866, 1868
Triangular norm (t-norm), 3177–3178
Triangulated Irregular Network (TIN), 3178–3179
– delaunay triangulation, 3179
– terrain, 3178, 3179
Trie, 2877–2879, 3179–3182
– automaton, 3179
– PATRICIA tree, 3181, 3182
– prefix tree, 3179
Trigger, 1979, 1980, 2754, 2755, 2757, 2758, 2760
Trigram, 1910
True positive rate (TPR), 2349–2351
Trust, 2414–2415

Trust and Reputation in Peer-to-Peer Systems, 3183–3187
– reputation, 3183, 3184, 3186, 3187
– trust, 3183–3187
– trust management system, 3183
Trusted hardware, 3191–3192
– tamper-proof hardware, 3191
– Trusted Platform Module (TPM) specifications, 3191
Trust in Blogosphere, 3187–3191
– blogosphere, 3187–3091
– distrust, 3189, 3190
– social network, 3187–3191
– trust, 3187–3191
Trust virtualization, 812, 813
TSQL2 (Temporal Structured Query Language), 3192–3197
– SQL/Temporal, 3192, 3193, 3197
– temporal extension of SQL–92, 3192, 3194
t_1, t_2, t_d, 477
Tuning application interface, 105–106
Tuning concurrency control, 3198–3201
– the concurrencycontrol, 3198
– serializability, 3199, 3200
– tuning by Leveraging Application Semantics, 3198–3199
Tuning in operating system, 2114–2115
Tuning time, 72, 74, 76
Tuple-based algebras for Xquery, 3640
Tuple-count *versus* time-sliding windows, 3533
Tuple equivalence, 2659, 3455
Tuple-generating dependencies, 3201–3202
– normalization, 3201, 3202
– schema mapping, 3201, 3202
– tgds, 3201, 3202
Tuple-generating dependency, 2489
Tuple relational calculus, 2370–2371
Turing machine, 495
Twig index, 3589, 3590
Twig query, 3637–3639
Two-dimensional shape retrieval, 3202–3203
– 2D shape, 3202
Two-level hierarchical scheduling policy, 1832, 1834
Two-level serializability, 505
Two-phase commit, 3204–3209
– atomic commit protocols, 3204, 3206–3208
– commit optimizations, 3204
– commit protocol, 3209–3213
Two-Phase Commit Protocol, 3209–3213
– commit protocol, 3209–3213
– distributed, 3209, 3210, 3213
Two-phase locking (2PL), 443–444, 2541, 3214–3217
– conflict-serializability, 3215
– a locked transaction, 3214–3216
– a locking policy, 3214, 3215
– a safe locking policy, 3214, 3216, 3217
Two-Poisson Model, 3218–3219
– a linear combination of two Poisson distributions, 3218
– a probabilistic model of indexing, 3218
Type-based Publish/Subscribe, 3220
– publish/subscribe, 3220
Type-checking, 3646–3649
Type of evidence, 1072
Types for temporal dimensions, 2999–3001
Types of locality, 1713
Types of schema evolution, 2500, 2502

U

UCC-user created content, 466, 467, 469, 470
Unbalanced hierarchy, 1308
Uncertain event inference, 3221–3223
Uncertainty, 1924
– in events, 3221–3225
– or impresion of data, 1808
– management, 3225–3231
– in sensor data, 647–651
Uncorrelated noise addition, 1911
Undirected graphs, 1261
Unicode, 3231–3232
Unicode standard, 3231
Unified Data Model (UDM), 3641
Unified modeling language (UML), 1927, 3232–3239, 3371–3373, 3552
– activity diagrams, 293
– class diagrams, 438, 439, 441, 442, 3233–3236, 3239
– model, 3597
– modeling language, 41
Uniform resource identifier (URI), 2428–2429
Unigram, 1910
Union, 3239–3240
Uniqueness constraint, 1587
Universal decimal classification, 808
Universal description, discovery and integration (UDDI), 854, 855, 857
Universal models, 324, 327
Universe, 1209–1210
Unobservability, 2142, 2143, 2145
Unrestricted implication problem, 1396
Unsupervised learning, 1664
Update anomaly, 1918, 1920
Update semantics, 99
Update through views problem, 3246
Updates on the views, 2639, 2642
Updates through Views, 3244–3247
Upstream backup, 1110–1113
Upward and downward inheritance, 142–143
Upward compatibility, 2937
Upward inheritance, 2911, 2912
Usability, 955, 958, 3247–3251
User-centered approach, 3247
User defined functions (UDFs), 2831
User-defined temporal periodicity, 3005
User interface architecture, 1329, 1330
User-level parallelism, 1566
User model, 2100, 2102
User profile(s), 2099, 2100, 2102
User queries over XML views, 3657
User-role assignment (URA97) model, 57
User updates over XML views, 3657–3658
Using parity and striping, 2359
UTF–8 (a 8-bit, variable-width encoding), 3231
UTF–16 (a 16-bit, variable-width encoding), 3231
UUID, 2428–2429

V

Validity region, 3439
Valid time, 3252
Valid-time database, 3254–3256
Valid-time index, 3254, 3255
Valid-time indexing, 240, 3254–3258
Value equivalence, 3258–3259
Value-List index, 244–247
Value reduct, 761–762, 764
Variable database, 1922, 1923
Variable group-size microaggregation, 1736
Variable time span
– calendar, 3259
– time interval, 3259
Vector graphics, 1374, 1376
Vector space model, 1571, 3059
Vendor neutral technology center, 2816
Venn, 3370–3372
Verbosity hypothesis, 940
Verification, 3551
Vertical fragmentation, 891, 892
Vertical partitioned data, 3263–3265
Vertical partitioning, 599–600
Vertical replication, 2409, 2412, 2413
Vertical search, 592
Vertical web search, 1154
Video application, 3265, 3266, 3268
Video clips, 3320, 3321, 3323
Video compression, 3298, 3301, 3302
Video content analysis, 468–472, 3271–3276, 3321
Video content analysis algorithms, 3271

Video data, 3292–3295
Video data model, 3277, 3278
Video format, 3296, 3297
Video history, 3265
Video indexing, 3360
Video metadata, 3286–3292
Video querying, 3292–3296
Video representation, 3265, 3266, 3270
Video retrieval, 3365
Video scene, 3303
Video scene and event detection, 3303–3308
Video search engine(s), 468–470
Video segment, 3292–3294, 3296
Video (temporal) segmentation, 3308
Video shot detection, 3316–3320
Video summarization, 3320–3324
View(s), 92–97, 1674–1679, 2639–2642, 3244–3247, 3326–3328, 3339–3340
– adaptation, 3324–3325
– definition, 3325
– dependencies, 1396, 1399–1400
– maintenance, 1670, 1672, 2549, 3326–3331
– recomputation, 3326
– self-maintenance, 2548–2550
– update, 3326–3328
– update translations, 3245, 3246
– View-based data integration, 3332–3339
Virtual data integration, 2481
Virtual medical record (VMR), 970
Virtual memory (VM), 278–279
Virtual relations, 3325, 3339
Vision, 3396–3398
Visual analytics, 3341–3346
Visual association rules, 3346–3351
Visual clustering, 3355–3360
Visual data exploration, 1623
Visual data mining, 3355, 3360
Visual distances and visual sangles of a document, 3413
Visual exploration, 405
Visual features, 1379, 1380
Visual information retrieval system, 1379
Visual interfaces, 267–268, 3379–3382
– metaphor, 3387–3388
– perception, 3395–3399
– query, 3399–3404
– formulation, 3028, 3033
– language, 3399–3404
– systems (VQSs), 3375, 3376, 3399
– representation, 3405–3410
– summaries, 3384
– system, 3395, 3396, 3398
– wrapping, 1649, 1651
Visual on-line analytical processing (OLAP), 3388–3395
Visualization, 3352–3354, 3366–3368
– of association rules, 3348, 3351
– of data, 749
– for information retrieval, 3410–3414
– methods, 655, 656, 3411
– pipeline, 3414–3416
– process, 3414
– of spatial data, 3382
– task, 3414, 3415
Visualization models for multiple reference points, 3412
Visualizing, 2019, 2020, 2022, 3416–3417, 3437–3438
– clustering results, 3417–3425
– clusters, 3417–3425
– hierarchical data, 3425–3432
– network data, 3432–3437
Visual metaphor, 3387–3388
Visual on-line analytical processing (OLAP), 3388–3395
Visual perception, 3395–3399
Visual query, 3399–3404
– formulation, 3028, 3033
– query language, 3399–3404
Visual query systems (VQSs), 3375, 3376, 3399
Visual representation, 3405–3410
Visual summaries, 3384
Visual system, 3395, 3396, 3398
Visual wrapping, 1649, 1651
Voronoi cell, 3439
– Voronoi diagram, 1892, 3438–3440
– Voronoi edges and voronoi vertices, 3439
– Voronoi neighbor, 3439
VP-trees, 1439–1441
Vulnerable edge, 2662

W

Wait-for graph (WFG), 899–902
WAN data replication, 3441–3445
Watermarking, 825–829, 2827, 2829
Wavelet, 857–863, 2898, 2899, 3446–3451
– based approximation, 1303
– synopsis, 860, 862, 863
– transform, 3446, 3447
W3C (World Wide Web consortium), 3441
– initiatives, 3441
– operations, 3441
– XML recommendation, 3584
Weak consistency, 1072
Weak temporal relation, 2659, 3455
Web 2.0/3.0, 3456–3457
– advertising, 3457–3459
– application, 1696–1697
– 2.0 applications, 2667
– 3.0 applications, 3457
– application server, 104
– browsing, 268
– characteristics, 3460–3462
– characteristics and evolution, 3460–3462
– crawler, 1147, 3462–3465
– crawler architecture, 3462–3465
– crawling, 1600
– data extraction, 590–593, 1595–1600, 3465–3470
– data extraction system, 3465–3470
– directory, 1463, 1464
– documents, 1191
– 2.0 enabling technologies, 3457
– evolution, 3460–3462
– extraction, 3471–3472
– forms, 1162
– graph, 1274
– index, 1463, 1465, 1466
– intelligence, 592–593
– marts, 3377
– mashups, 3378, 3482–3483
– mining, 595, 693–695, 3472
– ontology language (OWL), 2008–2009
– page cleaning, 1191
– question answering, 3485–3490
Web search, 1251–1255, 2082–2085
– engines, 3497–3500
– relevance feedback, 3493–3497
– relevance ranking, 3497–3500
– result caching, 3501–3505
– result prefetching, 3501–3505
Web service discovery, 857
Web service standards, 3510
– spam, 3520–3523
– spam detection, 3520–3523
– transactions, 3523–3524
– views, 3524–3525
– wrappers, 1595–1600, 3565
Web service description language (WSDL), 854, 2587
Web services business process execution language (WS-BPEL), 413–418
Weight, 2862
Weighted MDS, 1784
Weighting schemes, 3937–3942
What-if analysis, 3525–3528
Wide-area database, 3442
Wide-area data replication, 3443
Wikipedia, 967, 968
Wikipedia article, 1537
WIMP interfaces, 3529–3532
Window, 1064, 1065, 1067
– aggregates, 3538
– based query processing, 3533–3537

– joins, 3538
– query, 3533
Wipe, 3316, 3318
Wireless broadcast channel, 73
ω_j t_j, 477
$w_{mix}(q, e)$, 477
Workflow(s), 1077–1078, 1276–1278, 3539, 3545–3549, 3551–3557
– activity cycle, 1661
– branching, 2006
– control data, 494
– engine state data, 494
– evolution, 3540–3544
– generation, 3514
– join, 3544–3545
– management, 289–293, 3545, 3557, 3558
– management and workflow management system, 3545–3549
– management coalition, 3550
– model, 3550–3551
– provenance, 2202–2204
– schedule, 2474
– schema, 3558
– step, 41
– transaction, 3559
Workflow management coalition (WfMC), 1276
Workflow management systems (WFMS), 3547–3548
World Wide Web, 1417, 1418
WORM, 2798–2800
WORM storage, 3570
Wrapper(-ing), 1282–1286, 2511, 3472–3473, 3565–3568
– generation, 3465–3469
– induction, 3560–3568
– induction algorithms, 3563–3564
– learning, 3565–3567
– programming languages, 1649
– reinduction, 3566–3567
– repair, 3566–3568
– robustness, 3569
– verification, 3566–3568
Write ahead log (WAL), 517, 519
Write once read many (WORM), 2364, 2365, 3570
Write Skew (WS), 2661
– brokered notifications, 2210
– eventing, 2210
– notifications, 2210
– topics, 2210

X

XACML, 2142
XJoin, 904, 906, 907
XML (eXtensible Markup Language), 2047–2050, 2187, 2188, 2191, 2290, 2298, 2599–2600, 2932–2935, 3571–3572, 3576–3579, 3585, 3591, 3598–3600, 3608–3612, 3623–3626, 3634–3636, 3646–3649, 3659–3665
– absolute constraints, 3593
– access control, 3573–3575
– access control policy, 3573
– algebra, 3640, 3641
– API, 3602
– attribute, 3576
– benchmarking, 3576–3579
– cardinality, 3623, 3626
– estimation, 3623–3626
– graph synopsis, 3624–3626
– histograms, 3624–3626
– measurement, 3577, 3578
– performance, 3576, 3578
– selectivity, 3623–3626
– synopsis, 3623–3626
– synthetic data generators, 3577–3579
XML-based integration framework, 3597
XML benchmark, 1737
XML compression
– compression, 3579–3583
– data compression, 3580, 3583
– decompression, 3580
– homomorphic compression, 3580, 3582
– structure compression, 3581
XML databases, 1564
XML data reduced (XDR), 3614
XML document, text document, 3584
XML element, element, 3585
XML filtering, 3608–3612
XML foreign keys, 3592, 3593, 3595–3596
XML full-text search, 1544, 1545
XML functional dependencies, 3592–3594
XMLHttpRequest, 78
XML inclusion dependencies, 3592, 3593, 3595–3596
XML information integration
– data integration, 3591
– enterprise information integration, 3591
XML information item, 3585
XML Information Retrieval (XML IR), 1531–1534, 1536
XML information set, 3584
XML integrity constraint, 3592–3597
XML-IR, 1545, 1546
XML keys, 3594
XML language extensions, 3602, 3605, 3606
XML manipulation language, 3654
XML Metadata Interchange (XMI), 3597
XML parser, 3598–3600
XML parsing, validation, 3600
XML process definition language (XPDL), 3558, 3601
XML programming interface, 3598–3599
XML publishing, 3613–3616
XML-QL, 3614, 3637
XML queries, 2326
XML relative constraints, 3593
XML retrieval, 463, 2376, 2867, 2868, 3036, 3037, 3616–3620
XML schema, 3621–3622, 3650
XML scripting, 3607
XML signature, 832–834
XML storage, 3627–3633
XML tree pattern, 2050
XML tuple algebra, 3640–3645
XML typechecking, types, 3646–3649
XML types, 3646, 3648, 3649
– regular expression, 3651–3653
– validation, 3652
XML updates, 3654–3655
XML views, 3656–3659
XML vocabulary, 1236
XPath, 1876–1879, 3634–3636
XPath languages, 3665, 3666
Xpath/Xquery, 2047–2049
XPDL, 3550
XPERANTO, 3613–3615
Xquery, 3591, 3665–3670
XQuery Full-Text, 3665–3670
XQuery processors
– DB2 XML, 3672, 3674–3675
– timber, 3672–3675
XSL (eXtensible Stylesheet Language), 3676–3680
XSLT (eXtensible Stylesheet Language Transformations), 3676–3680
XSLT programming, 3676–3678
XSL/XSLT, 3676–3680
XTM, 3125

Z

Z39.50, 2519, 2521
Zero-One laws, 1081, 1082, 3683
ZF-expression, 420–421
Zoomable user interface (ZUI), 3684–3689
Zooming, 3684–3689
Zooming techniques, 3684–3689